CSC 8980 Natural Language Processing

Q&A system for Wikipedia dataset and chat bot for user queries

Krishna Rukmini, Puthucode
Ruthuparna Naikar
Kausik Amancharla

Things we talk about

- Inspiration
- Definition
- History/Research
- Dataset
- Methodology
- Project issues
- Conclusion

Inspiration for the Project

- Search engines have always been effective when dealing with large amounts of data, and they are generally good at retrieving documents that contain the user query.
- The next logical step will be to try to automate the part of the process where the user searches for a response in the retrieved text or document.
- Question answering (QA) is a branch of computer science concerned with developing systems that automatically address questions asked by humans in natural language.
- So, here we are, on the verge of putting the Q&A framework into action.

What is a QnA System

A question-answering implementation can generate answers by querying a structured database of knowledge or information, most commonly a knowledge base. More commonly, to extract answers from a jumble of unstructured natural language documents.

The following are some examples of natural language document sets that have been used in question answering systems:

- a local library with reference books
- newswire reports obtained from internal agency records and web sites
- a subset of Wikipedia articles on the World Wide Web

Cont.

- An open domain question answering system in information retrieval aims to provide a response to the user's query. Instead of a list of related documents, the answer is in the form of short texts.
- The system takes a natural language question as an input rather than a set of keywords.

History

- List-Structured database system
- Graphic database system
- Text-based system
- Logical Inference system

- Using New Deep Learning Techniques

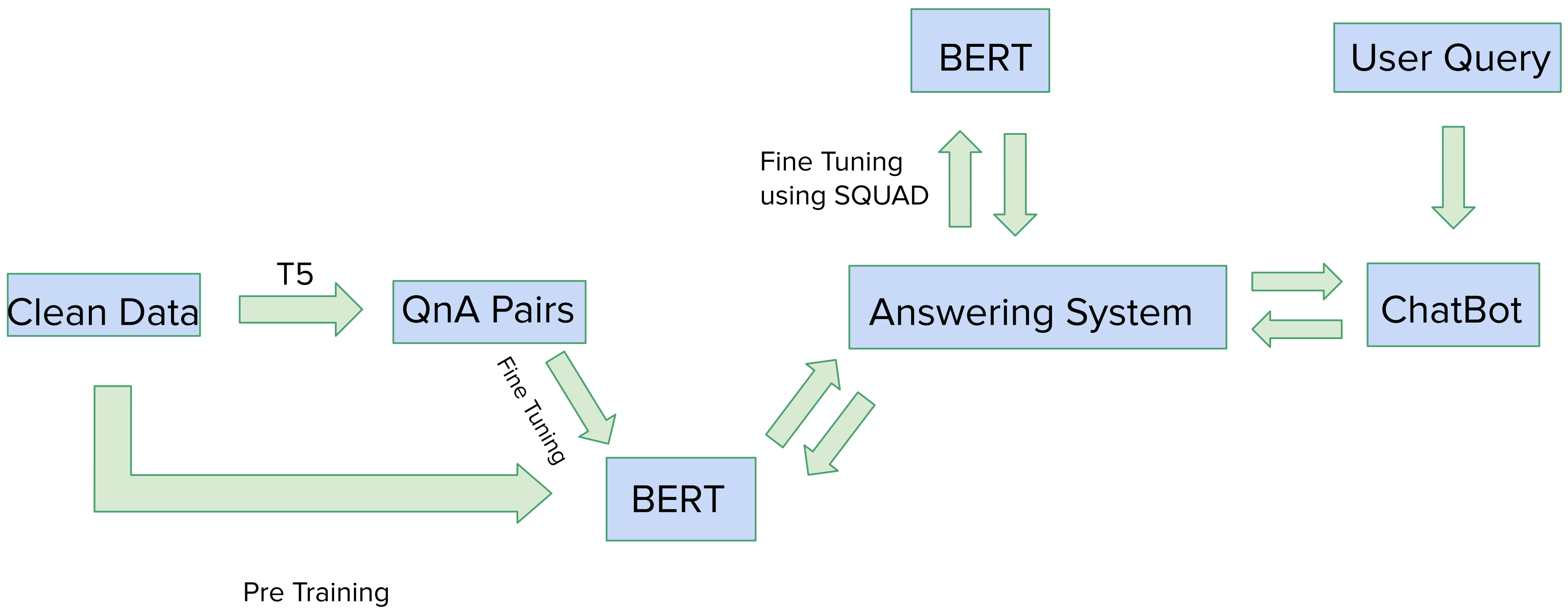
Picking Our Dataset (Corpus)

- Wikipedia was the choice of Corpus as it is the largest collection of open source factual information available.
- Wiki data can be accessed through data dumps available at "https://dumps.wikimedia.org/enwiki/latest/"
- The whole compressed dataset is around 220 gb so we use a condensed wiki dataset from Hugging Face.
- This includes articles from English Language.
- As this is an inferential study we picked ~300K articles randomly from 6M articles.

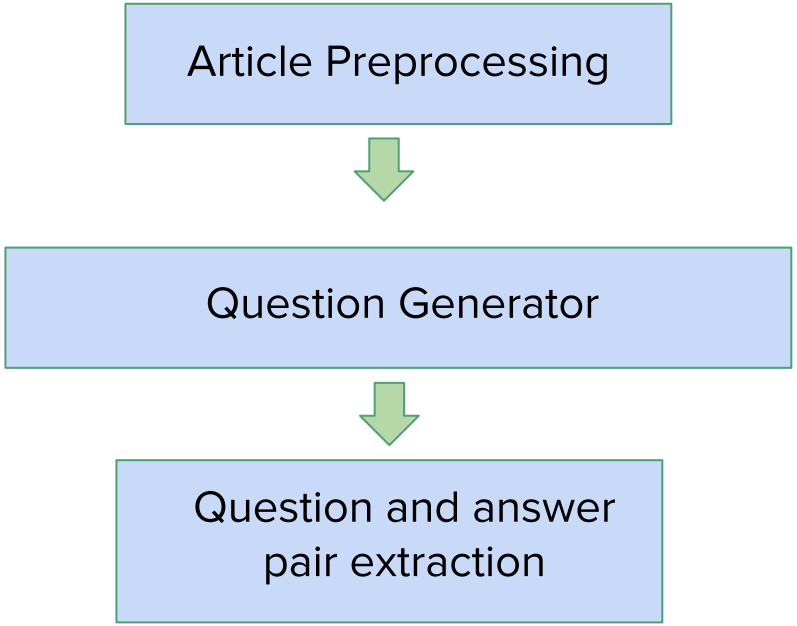
Cleaning The Data

- Using Regular Expressions we removed References,Symbols,images,hyperlinks,\n,\t
- It was decided that we won't remove stop words as they might play a significant role in letting the model understand the context behind the sentences
- The dataset is formatted and saved into a .tsv file to use further.

Methodology



Structure Question Generation



Question Generation

1. <u>Article Pre-Processing</u>

   Given a Article:

   - Converted whole article text to unicode

   - Sentence tokenizer
   - POS Tagging

We use NLTK and SpaCy

Pre-trained models such as ULMFiT, BERT, GPT, and others have been open-sourced to the NLP community in recent years. Given the size of such massive models and the amount of data and computation needed, training such networks from scratch is virtually impossible. This is where "Transfer Learning," a new learning paradigm, comes into play.

Understanding T5 Model : Text to Text Transfer Transformer Model

T5: Text-to-Text-Transfer-Transformer model proposes recasting all NLP tasks as a single text-to-text format, with text strings as the input and output. Because of the formatting, a single T5 model may be used for a variety of purposes.

It takes text input from the left for various NLP tasks and outputs the text for that task, as seen in image displayed.

source

Since answer aware models need answers in order to generate questions, we'll need something that can extract answers from the text in the form of spans. NER, noun-extraction, and other techniques may be used to do this. However, here the text-to-format feature is used to extract answers in T5.

The input for answer extraction is processed as follows since the highlight format would need to know the location of extracted answer spans.

- Sentence segmentation from an article
- Highlight each sentence with <hl> tokens, for each one that has a response.
- Join the answers in that sentence with <sep> tokens for the goal text.

Example:

Input text: <hl> Neuro-linguistic programming is a pseudoscientific approach <hl> created by Richard Bandler in the 1970s. **target text:** pseudoscientific <sep>

Input text: Neuro-linguistic programming is a pseudoscientific approach <hl> created by Richard Bandler in the 1970s. <hl>. **target text:** Richard Bandler <sep> 1970 <sep>

T5ForConditionalGeneration

T5ForConditionalGeneration.generate( ) is recommended for sequence-to-sequence generation. This method feeds the encoded input to the decoder through cross-attention layers and generates the decoder output auto-regressively. Relative scalar embeddings are used in T5. Padding the encoder input can be achieved on both the left and right sides.

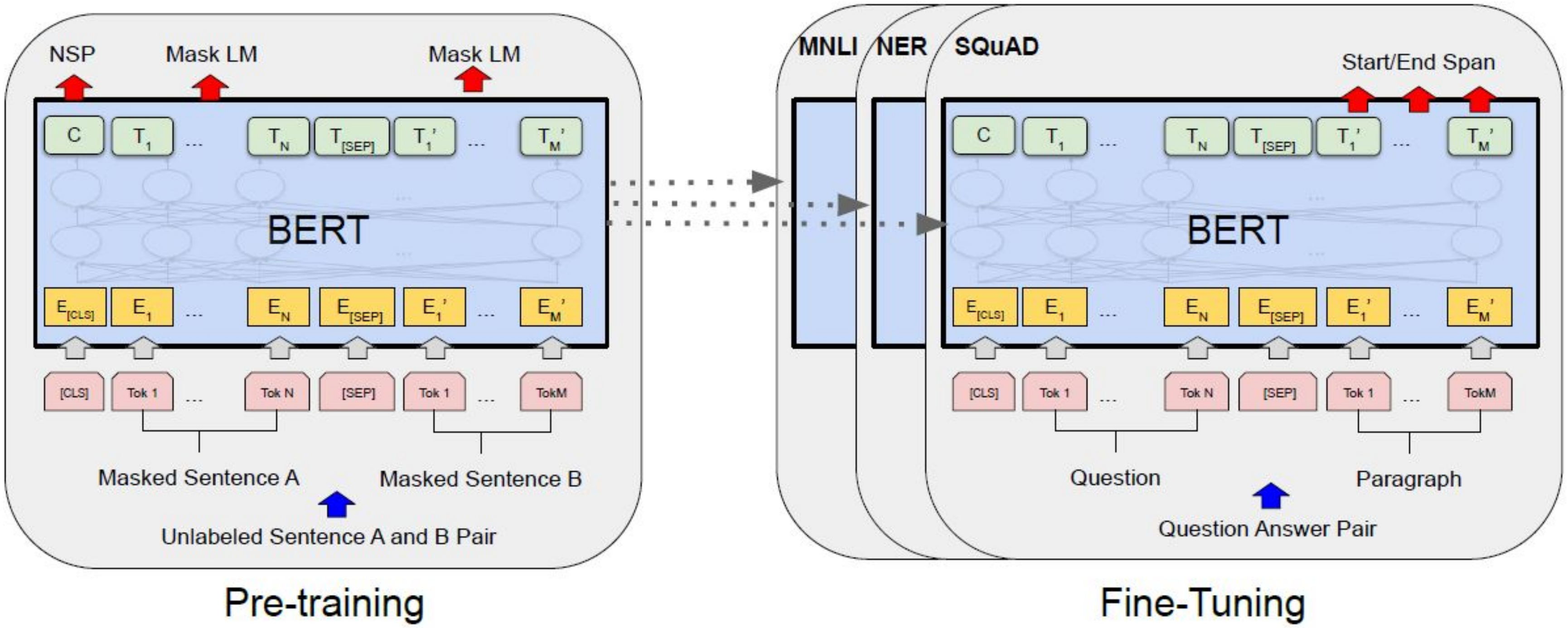
It is forced to learn by the trainer. This implies that we always need an input sequence and a goal sequence for preparation. Inputid is used to feed the input sequence to the construct. The target sequence is moved to the right, i.e., a start-sequence token is appended to it, and the decoderinputid is used to feed it to the decoder.

Output sample of T5 Question generator

```
Questions = get question pipeline("generate questions")
Questions( 'HIS "Hightech Information System Limited" established 1987 , is a Hong Kong based graphics
card manufacturer that produces AMD formerly known as ATI Radeon graphics cards. Its headquarters are in
Hong Kong, with additional sales offices and distribution networks in Europe, the Middle East, North
America and Asia Pacific Regions.The current distributor in Hong Kong is JunMax Technology. Products
manufactures and sells AMD Radeon series video cards. They are known for their IceQ cooling technology
as well as producing the latest and fastest PCI cards like AMD Radeon RX 590, RX 5700 and RX 5700 XT.
\In 2019, HIS launched new versions of the RX 5700 XT in pink and blue. External links HIS Ltd.')
```

```
Output:
```

```
[{'answer': 'HIS "Hightech Information System Limited"',
  'question': 'What is the name of the company that produces AMD formerly known as ATI Radeon graphics cards?'},
 {'answer': '1987',
  'question': 'When was HIS "Hightech Information System Limited" established?'},
 {'answer': 'Hong Kong', 'question': "Where is HIS' headquarters located?"},
 {'answer': 'JunMax Technology',
  'question': 'What is the current distributor of AMD Radeon graphics cards in Hong Kong?'},
 {'answer': 'AMD Radeon',
  'question': 'What series of video cards does HIS manufacture and sell?'},
 {'answer': 'IceQ',
  'question': 'What type of cooling technology is HIS known for?'},
 {'answer': 'pink and blue',
  'question': 'In what colors did HIS launch the RX 5700 XT?'},
 {'answer': 'HIS Ltd.',
  'question': 'What is the name of the company that manufactures and sells AMD Radeon graphics cards?'}]
```


NSP
Mask LM
Mask LM
C
T1
TN
T[SEP]
T1'
TM'
BERT
E[CLS]
E1
EN
E[SEP]
E1'
EM'
[CLS]
Tok 1
Tok N
[SEP]
Tok 1
TokM
Masked Sentence A
Masked Sentence B
Unlabeled Sentence A and B Pair
Pre-training
MNLI
NER
SQuAD
Start/End Span
Question
Paragraph
Question Answer Pair
Fine-Tuning

Source

Data for Pre-Training

Data used :

300,000 wikipedia articles

Pre-processing data :

We had to format training data into one sentence per line in order for the NSP task to function properly, using an empty line to separate each text and store it in a text file before training tokenizer.

We have used Google Cloud technology to pre-train **BERT**, a state-of-the-art Natural Language Understanding model.

Pre-training BERT

Step 1: Setting up the training Environment:

- Importing packages and authorizing in google cloud.
- We can use any model suitable for out use-case when building your tf.Module. As we have wikipedia articles of english language, we are using the english model trained by Google, BERT base checkpoint. This was imported from the official github of bert.

Step 2: Getting the data set:

- We can normalise the data into any particular format, as we have already preprocessed the data, we just got our dataset text file that we have generated earlier.

Step-3 : Building the vocabulary:

In this step we have built the vocabulary that represents the dataset.

The tokenizer used in the BERT paper is WordPiece. We've instead used the SentencePiece tokenizer in unigram mode. Although it is not directly compatible with BERT, we can make it work.

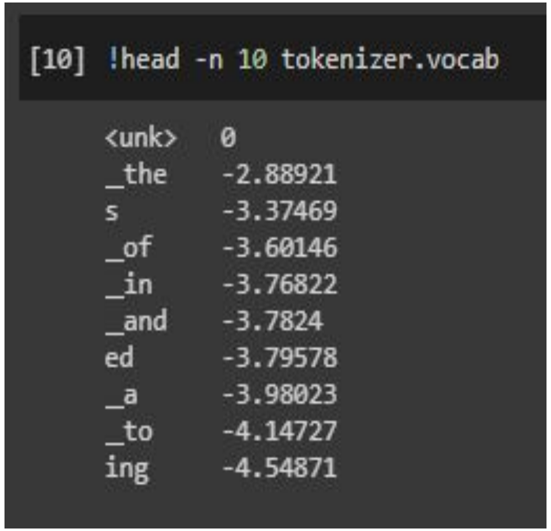
SentencePiece uses a lot of memory, so running it on the entire dataset in Colab would cause the kernel to crash. To prevent this, we have constructed the vocabulary by randomly sampling portions of the dataset.

SentencePiece also automatically introduces BOS and EOS control symbols to the vocabulary. We specifically disable them by setting their indices to -1. VOC SIZE is usually set to a value between 32000 and 128000, we used 32000.

The WordPiece tokenizer prepends the subwords which occur in the middle of words with '##'. The subwords occurring at the beginning of words are unchanged. If the subword occurs both in the beginning and in the middle of words, both versions (with and without '##') are added to the vocabulary.

This will add two files tokenizer.model and tokenizer.vocab

Below is a snap of the values in our vocab file:



```
[10] !head -n 10 tokenizer.vocab

    <unk>   0
    _the    -2.88921
    s       -3.37469
    _of     -3.60146
    _in     -3.76822
    _and    -3.7824
    ed      -3.79578
    _a      -3.98023
    _to     -4.14727
    ing     -4.54871
```

We also add some special control symbols which are required by the BERT architecture. By convention, we put those at the beginning of the vocabulary.

We observe, SentencePiece does quite the opposite to WordPiece.

SentencePiece first escapes the whitespace with a meta-symbol "_" (U+2581) as follows:

For example:
Hello_World.
segmented into small pieces, as follows:
[Hello] [_Wor] [ld] [.]

To create a WordPiece-like vocabulary, we need to perform a basic conversion, deleting "_" from tokens that contain it and adding "##" to those that don't.

Step-4 : Generating pre-training data:

We can now produce pre-training data for the BERT model using the vocabulary we've collected. We had split our dataset into shards because it may be very huge.

Before starting to generate the data, we had to set few model specific parameters.

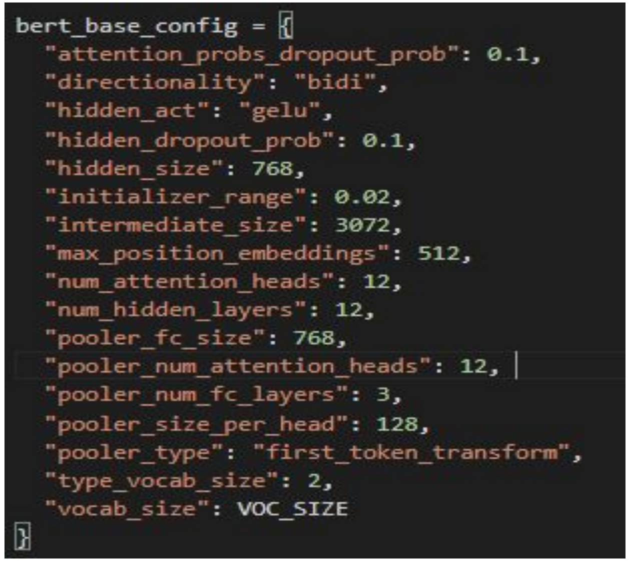
```
MAX_SEQ_LENGTH = 128

MASKED_LM_PROB = 0.15

MAX_PREDICTIONS = 20

DO_LOWER_CASE = True

PROCESSES = 2

PRETRAINING_DIR = "pretraining_data"
```

Now, for each shard we need to call create_pretraining_data.py script. for that we used the xargs command.

Running this took quite some time and it depends on the size of your dataset

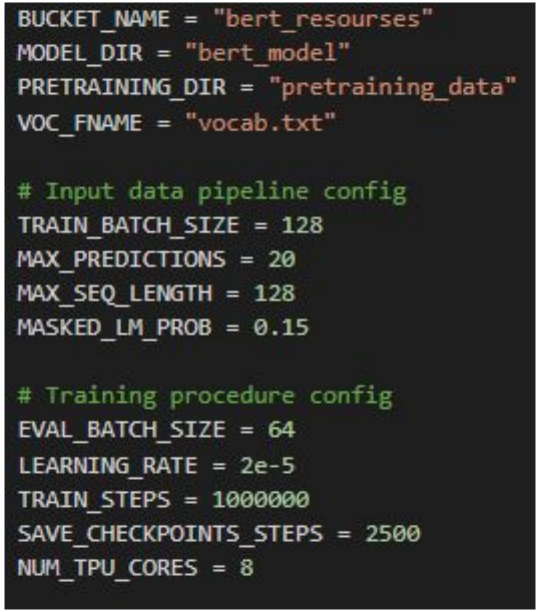
We have saved our hard-earned assets to Google Cloud Storage in order to protect them. We've developed the GCS bucket and two GCS directories, one for data and the other for model. The model vocabulary and configuration file will be placed in the model directory.

Hyperparameter configuration for bert base, used the default config details:



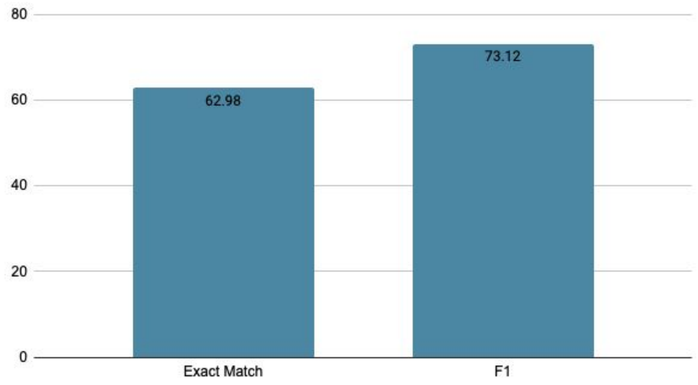
```
bert_base_config = {
  "attention_probs_dropout_prob": 0.1,
  "directionality": "bidi",
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "max_position_embeddings": 512,
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pooler_fc_size": 768,
  "pooler_num_attention_heads": 12,
  "pooler_num_fc_layers": 3,
  "pooler_size_per_head": 128,
  "pooler_type": "first_token_transform",
  "type_vocab_size": 2,
  "vocab_size": VOC_SIZE
}
```

Step-5: Training the model.



```
BUCKET_NAME = "bert_resourses"
MODEL_DIR = "bert_model"
PRETRAINING_DIR = "pretraining_data"
VOC_FNAME = "vocab.txt"

# Input data pipeline config
TRAIN_BATCH_SIZE = 128
MAX_PREDICTIONS = 20
MAX_SEQ_LENGTH = 128
MASKED_LM_PROB = 0.15

# Training procedure config
EVAL_BATCH_SIZE = 64
LEARNING_RATE = 2e-5
TRAIN_STEPS = 1000000
SAVE_CHECKPOINTS_STEPS = 2500
NUM_TPU_CORES = 8
```

We set up the training run, built the estimator and input feature, and turned on the bass cannon (run_pretraining.py)

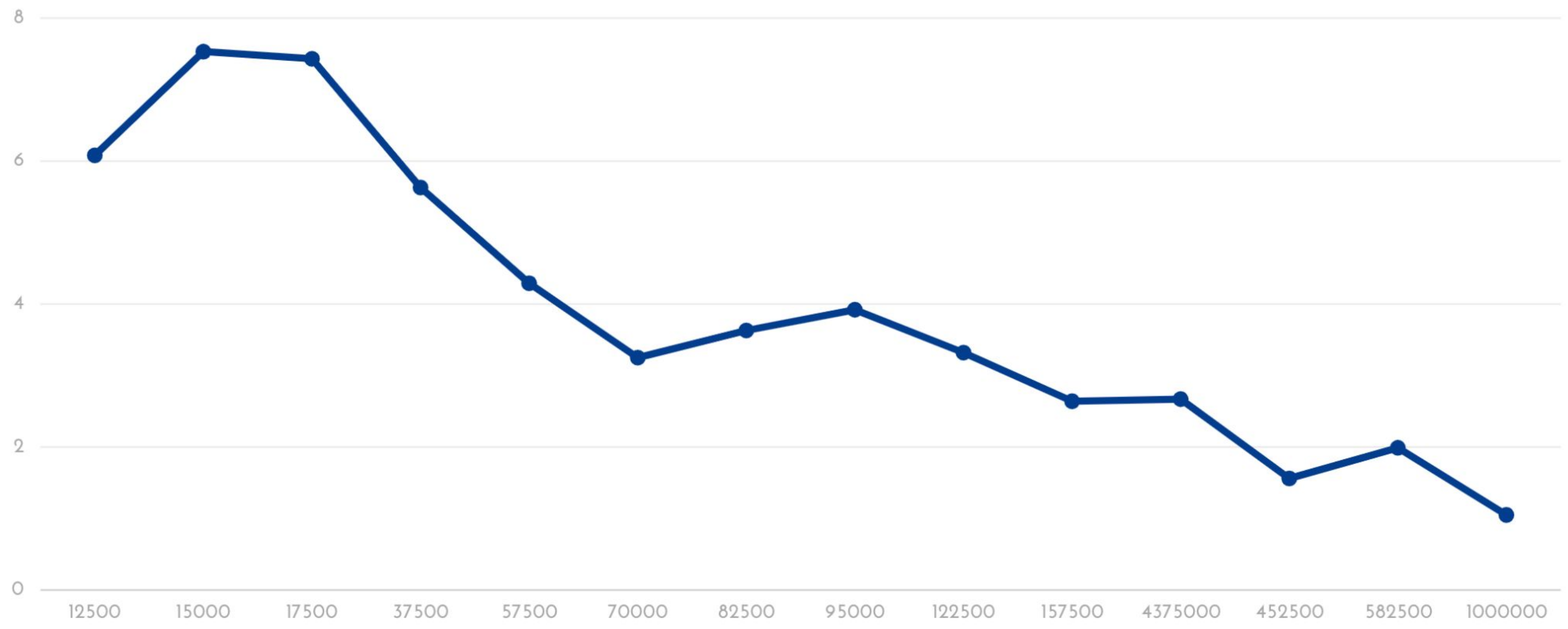


Global_Ste	1000000
Loss	1.9906723
Masked_lm_Accuracy	0.61764705
Masked_lm_loss	1.9346626
Next_Sentence_Accuracy	0.9925
Next_Sentence_loss	0.025408989

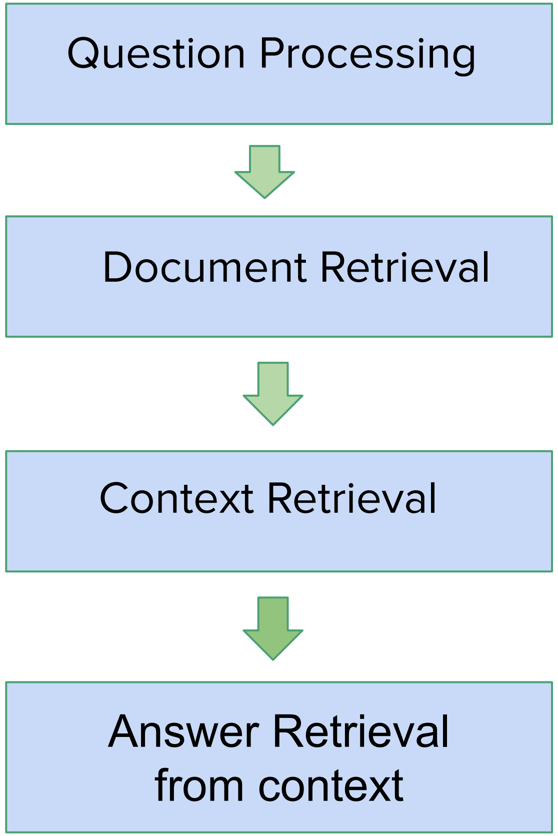
Exact Match	62.9801
F1	73.1239

Task	Batch Size	TPU/GPU	Run Time Taken
Text file - data for pre-training Bert	300k	Tesla T100 GPU	15hrs
run_pretraining.py	300k	GCS TPUv2 Instance	26hrs

```
2021-04-21 19:13:53,345 :  Loss for final step: 1.5081744.
2021-04-21 19:13:53,347 :  training loop marked as finished
<tensorflow estimator.python.estimator.tpu.tpu_estimator.TPUEstimator at
0x7f34db8f6250>
```

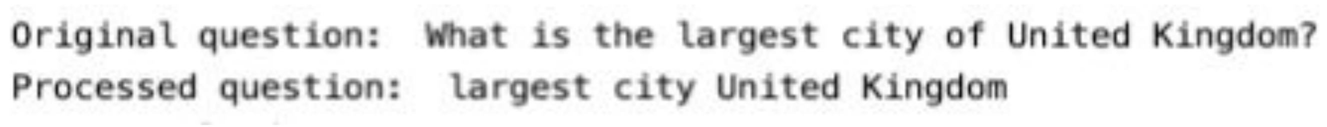


Answering System



Question Processing

- Use spaCy's NLP model tokenizer to tokenize the question
- Identify the parts of speech of all the words in the question
- Keep only the parts of speech : nouns, proper nouns, and adjectives



```
Original question:  What is the largest city of United Kingdom?
Processed question:  largest city United Kingdom
```

Document Retrieval

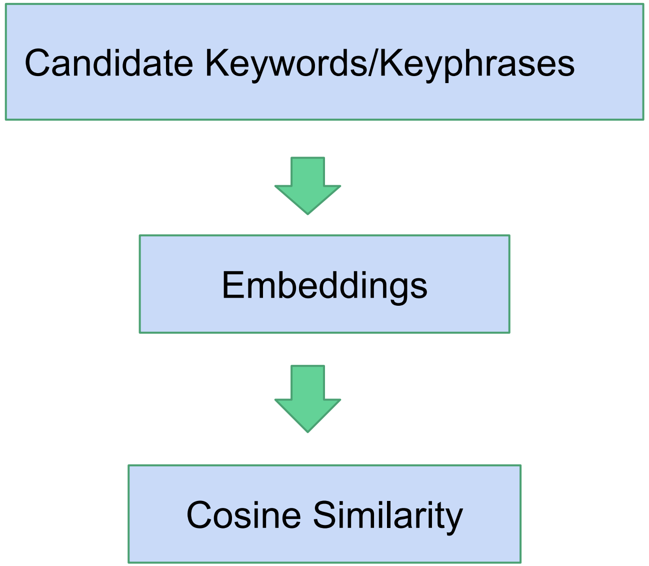
- The main aim of this is to find the documents which have the highest probability of having the answer to our question. Here, we have employed a multistep process

- The approach implemented here is Keyword Extraction

Keyword Extraction

Keyword extraction is the automated process of extracting the words and phrases that are most relevant to a document

Keyword Extraction



Keyword Extraction

1. Candidate Keywords/Keyphrases
   - A list of candidate keywords or keyphrases from a document.
   - Using Scikit-Learn's `CountVectorizer` we generate keywords/key phrases.
2. Embeddings
   - Converting both the document as well as the candidate keywords/keyphrases to numerical data
   - `sentence-transformers` package creates high-quality embeddings that work quite well for sentence- and document-level embeddings
   - We are using Distilbert as it has shown great performance
3. Cosine Similarity
   - Find the candidates that are most similar to the document

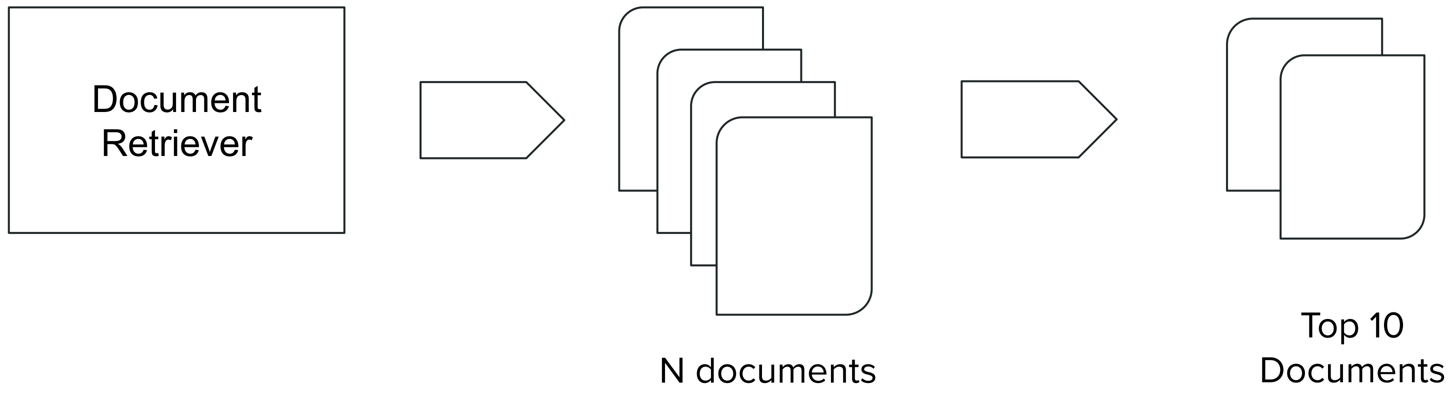
Keyword Extraction Output

Article
Talk
Psychology
From Wikipedia, the free encyclopedia

```
['university psychology',
 'work university psychology',
 'psychology departments',
 'psychology psychologists diverse',
 'psychology includes study',
 'social psychology psychologists',
 'cognitive scientist psychologists',
 'psychology psychologists',
 'psychology departments teach',
 'university psychology departments']
```

Document Retrieval

- With the Processed Question and Generated Keywords, we extract the top 10 documents which have the highest probability of having the answer to our question.
- We use cosine similarity for this task



Context Retrieval from the documents

- Get a list of all sentences in top documents
- Tokenize all our sentences and use lemmas of the words instead of the original words.
- We're also doing it for the question text.
- Use the BM25 ranking function to rank all our sentences against the given query.
- Extract the top *N* results from the step above and build a paragraph out of all those *N* sentences

Project Outcomes

Using Pre-Trained models for Extracting the Answer

- Use of BERT to extract the Answer from the given Question and context.
- Promising outcomes of our pre-training model

Limitations

- Corpus Size
  - Dealing with 300,000 wikipedia articles
- Keyword Generation in Document Retrieval
  - Regardless of using efficient methods, it is hard go generate relevant keywords for the whole document
- Pre-Training the model
  - the model would run for more than 8 hours and the TPU session in collab would time out.