UCLA Winter 2022 CS247 Course Project
Open-Domain Question Answering System

Anthony Ortiz [1] **Daniel Hulbert** [1] **Edgar Salvador** [1]

Abstract

Open-Domain Questions Answering (QA) is an area of Natural Language Processing (NLP) with exciting applications. From search engines to voice control systems, QA applications continue attracting more data scientists. One of the essential QA applications uses advanced text search algorithms to answer open-domain questions by leveraging information inside Wikipedia articles. This paper describes the use of TF-IDF Vectorization algorithms to create an Information Retrieval System that identifies the most relevant Wikipedia articles that could answer a particular open-domain question. This paper also presents specific variations of the BERT transformer model to identify the most probable answers for such questions. All code referenced in this paper can be found at: https://github.com/DannyH10/CS247_Final_Project

1. Problem Statement

In this project we intend to answer open-domain questions using extracted paragraphs from Wikipedia articles. Using an advanced text search algorithms, we will identify which articles would be most likely to contain the information most pertinent to answering a given open-domain.

The motivation for this project was the academic and professional research paper *"Reading Wikipedia to Answer Open-Domain Questions."* by Chen and Danqi (Chen). In this research paper, a method for obtaining high accuracy for answering open-domain questions using Wikipedia articles as the information source is described along with the results (accuracy) of the designed model's implementation.

The applications for such fact finding solutions in large textual databases (Wikipedia) could be very useful for a variety commercial, academic, and educational applications. Wikipedia represents one of the largest open-source text-based information platforms in the world. At its surface, Wikipedia is a collection of documents in a database with a very unsophisticated look up engine. By writing a more targeted text-based search algorithm to find answers to factual questions, new features for web browsers and voice recognition systems can be implemented into existing platforms such as Wikipedia itself, Amazon's Alexa, Apple's Siri, Youtube (video thumbnails and descriptions), and many others. Such an algorithm could be trained on any similar data set (a set of reference books, newspaper articles, etc.). Due to the graph like nature of the Wikipedia database, more complex graph based algorithms can also be explored for improving the accuracy of answers for a myriad of text-based questions.

2. Literature Review

2.1. Reading Wikipedia to Answer Open-Domain Questions (2017)

The motivation for this project was the academic and professional research paper *"Reading Wikipedia to Answer Open-Domain Questions."* by Chen. In this paper, the answer generation process is broken down into two distinct parts; a document retrieval system to generate a list of 5 articles that are most likely to contain the answer to the question and an answer retrieval system, which takes the 5 articles returned from the document retrieval system and returns a single answer to the question, in what they call a Document Retrieval Question Answering Algorithm (DrQA). The authors benchmark their results on multiple data sets including the SQuAD 2.0 data set.

2.1.1. Article Retrieval

The document retrieval portion implements a TF-IDF algorithm using an n-gram model to encode the document and question information. The similarity between the question and all of the articles is calculated and the top 5 most similar articles to the question are returned. The number of terms used in the n-gram encoding was swept a bigram model (n=2 terms) was chosen for optimal performance.

2.1.2. Answer Retrieval

The answer retrieval portion involved using two custom neural networks to generate embedding's for the questions and the articles, as well as a third custom neural network implemented using a linear classifier is used to retrieve the

Table 1. Google BERT PCE Hyper-parameters

PARAMETER	DESCRIPTION
DATA	WIKIPEDIA (2.5B WORDS) + BOOKCORPUS (800M WORDS)
BATCH SIZE	131,072 WORDS (1024 SEQUENCES * 128 LENGTH OR 256 SEQUENCES * 512 LENGTH)
TRAINING TIME	1M STEPS (40 EPOCHS) 4X4 OR 8X8 TPU SLICE FOR 4 DAYS
OPTIMIZER	ADAMW, 1E-4 LEARNING RATE LINEAR DECAY
LOSS	MLM AND NSP COMBINED

answer. The embedding for the paragraphs is done using a recurrent neural network (RNN), which can make use of the ordering of particular words to better embed the context of the paragraphs. Additional functionality is added to directly compute matches between the passage and question, both using direct matches and in a probabilistic way.

2.1.3. STRENGTHS AND LIMITATIONS

The two step algorithm proposed efficiently reduces the task of querying the corpus of interest in two stages, greatly enhancing the speed at which the Natural Language Processing part of the algorithm can retrieve the answer. However, by dividing the answer query into two stages, a second error source has been added in the stage generating the article subset to apply the answer retrieval on. Accuracy rates of both stages must be optimized and improved to enhance the total algorithms accuracy, as one of the two stages under performing will result in lower total performance.

2.2. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (2019)

BERT was introduced by Google AI Language. Google published the academic and professional research paper ”*BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* ” by Devlin, et al. BERT is the transformer encoder used to solve the problem statement presented in this paper.

2.2.1. GOOGLE BERT MODEL EXPLAINED

BERT stands for Bidirectional Encoder Representations from Transformers. It is an open-source library created in 2018 at Google and BERT’s advantages comes from the the application of the Bidirectional training of Transformers, the use of Masked Language Model (MLM), and the application of Next Sentence Prediction (NSP) techniques. BERT is PCE model that was originally trained with a series of specific Hyper-parameters which are presented in table 1.

Internally BERT is a Transformer. Transformer are deep learning models that uses an Attention layer to identify the context of the input instead of the regular meaning of it. Long Short-Term Memory (LSTM), Recurrent Neural Network (RNN) or other methods generate knowledge sequentially which means that they depend on the order in which the elements are processed. Transformers can process data in parallel fashion; therefore, it can process more data in less time. It is important to mention that in general transformer has an encoder-decoder architecture; however, BERT does only have the encoder part. Therefore, a BERT model needs to be fine-tuned in order to decode the information and predicts results.

BERT Transformer Encoder has an Attention-based architecture is made of the following layers:

- **Positional embedding:** It makes use of the order of the sequence. introduces relative positioning knowledge
- **Multi-headed self-attention:** it has Queries, Keys, and Values as input, and it is used to avoid Local Bias and to model context
- **Norm and residuals:** It increases the health of deep networks
- **Feed-forward:** It captures non-linear hierarchical features

Typical RNNs are unidirectional models. This means that the model trains the information from left to right or right to left and always in sequential order. ELMO, OpenAI GPT, ULMFIT, etc. are examples of unidirectional.

BERT is a bidirectional model that trains data from left to right and right to left. It reads sequences of words at once to acquire knowledge of word context from surroundings; Words can ”See Themselves”. Figure 1 illustrates unidirectional and bidirectional transformers.

BERT can achieve bidirectional learning thanks to a technique during the training process called Masked Language Model (MLM). MLM uses the context of surrounding words and masks tokens to predict what the masked word should be.

There should be a balance in the amount of masking that is performed on the sentences. Little masking increases the difficulty of the training process, while much masking decreases the amount of context knowledge that can be captured. Approximately 15% of the words are masked where 80% of masked words replace with [mask], 10% of masked words replace with random words, and 10% of masked words kept the same.

The relationship between two sentences is not typically captured when training an NLP model. Relationship knowledge

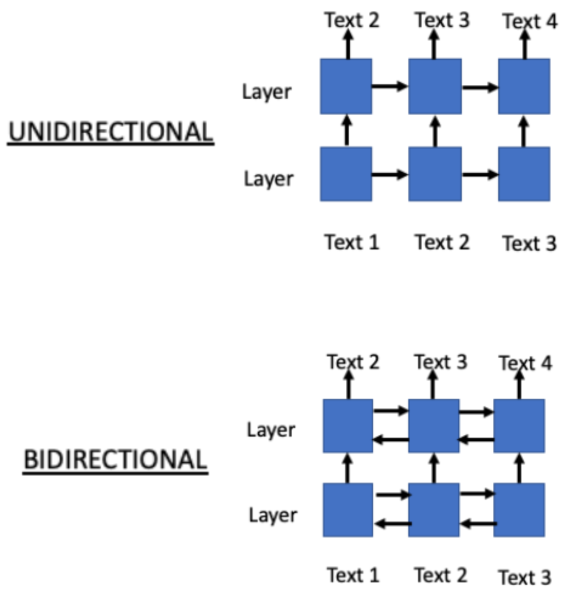

Figure 1. Unidirectional and Bidirectional Transformer

is captured by a different technique known as Next Sentence Prediction (NSP). Sentences are input into the model as pairs, and the model learns to predict if any second sentence is subsequent to the first sentence in the original document. NSP consists of providing inputs such as 50% inputs are two subsequent sentences, and 50% inputs are two random sentences from the corpus.

As mentioned earlier, BERT can be fine-tuned to be able to decode. Different applications out of a BERT model can be derived depending on what fine-tuning process is used. To perform QA tasks, BERT needs to be fine-tuned by adding a layer of vectors that indicate the beginning and end of a sentence, and from here calculate the probability that answers correspond to the start and end of the answer span.

2.2.2. Strengths and Limitations

The bidirectional learning combined with the 12 layer model used in BERT leads to much higher accuracy for general question answering tasks for simple passages. However, the added complexity used to generate more precise answers creates a tradeoff with the runtime of the model, in that the runtime is significantly longer for the complex model. The prevalence of this model also allows for many pre-trained models to be tested and used on our data set, including models that are trained on the SQuAD 2.0 dataset itself.

3. Methodology

The two main challenges are the size of the data available to us, along with splitting of the tasks and accuracy required from each task to reliably answer the questions. A lot of the possible questions rely on very few words to ask and therefore must be able to reliably identify a set of N number of articles that will contain the paragraph with the information needed to answer the question. It is not feasible to compare every paragraph on Wikipedia to the question we need to answer and thus we need to reliably filter all the useless articles and decide what needs to be searched through more thoroughly.

With the intention of objectively and systematically overcoming the described challenges, this paper presents a design made of two main subsystems: Information Retrieval and Answering Identification system, similar to the two part implementation of the DrQA. Higher performance information retrieval algorithms, such as the one developed by Devlin et al. can be implemented in the second part of the two step algorithm to answer specific questions. Figure 2 presents the architecture diagram and the work flow for the solution based on these two subsystems.

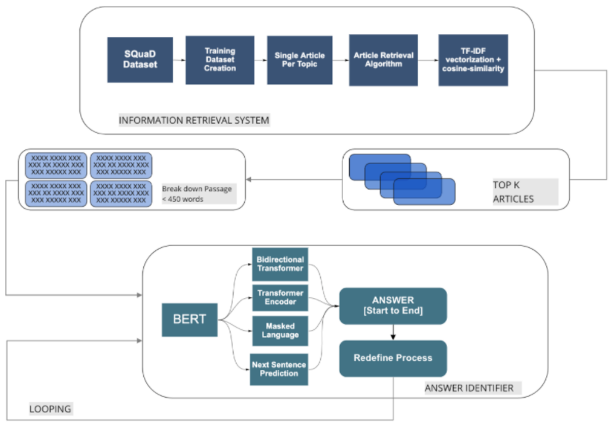
SQuaD Dataset
Training Dataset Creation
Single Article Per Topic
Article Retrieval Algorithm
TF-IDF vectorization + cosine-similarity
INFORMATION RETRIEVAL SYSTEM
Break down Passage < 450 words
TOP K ARTICLES
BERT
Bidirectional Transformer
Transformer Encoder
Masked Language
Next Sentence Prediction
ANSWER [Start to End]
Redefine Process
ANSWER IDENTIFIER
LOOPING

Figure 2. Outline of data flow from SQuAD2.0 json file to answer retrieval. The Upper Box represents the Document retrieval algorithm, while the lower box represents the Question Answering algorithm. The middle boxes show the document partitioning to feed segments into the QA portion.

3.1. Data Processing

3.1.1. SQuAD Dataset

The Stanford Question Answering Dataset (SQuAD), is a dataset that contains general topics, questions related to these topics, Wikipedia paragraphs containing answers to the questions, and direct text answers to these questions.The data set can be downloaded from the following link: https://rajpurkar.github.io/SQuAD-explorer/. The

SQuAd dataset is a popular dataset used for benchmarking text-based algorithms.

SQuAD 2.0 is the version of the dataset used for this analysis. This dataset contains a combination of 100,000 answered questions with almost 50,000 questions with no answers within more than 500 Wikipedia articles. Answers to every question is extracted from this dataset as a segment of text, or span, from the corresponding reading passage, or the question might be unanswerable.

The SQuAD 2.0 data set is structure as a JSON file. The different elements of this JSON structure are presented in Figure 3.

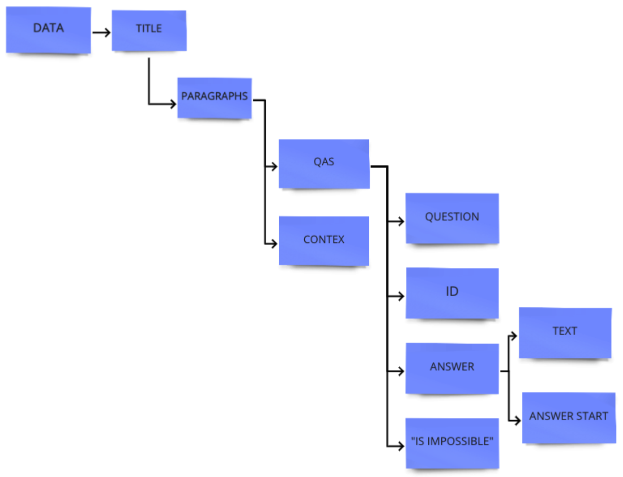

Figure 3. SQuAD JSON Structure

3.1.2. Training and Test Datasets

The SQuAD dataset download file is a JSON file. The JSON file structure is a table where each row contains a series of embedded dictionaries. We imported the JSON file using Python, then imported the function json normalize from the Pandas python packaged. We used this function and custom functions that we wrote to collapse the 'tree' of dictionaries into a flat-dataframe. We then created a single .csv file with the following headers that can be imported as a dataframe. This made it easier for us to select columns from this table and convert them to lists to send out article and questions text training data to our article retrieval algorithm. Using our custom training dataset .csv file, we are now able to import this as a simple dataframe and create a single 'Wiki' article for each unique topic in the dataset using the answer contexts or paragraphs for each true answer. We created a .csv file to store the topics and their articles for further simplicity. Each 'Wiki' article will correspond to a row in our bag of words list. This way, we can run stemming and transform the articles lists into word count matrices. The data preparation process is then repeated for the SQuAD test dataset JSON file.

3.2. Article Retrieval

3.2.1. Article Retrieval Algorithm

The Article Retrieval Algorithm leverages information in the articles themselves and can be broken down as follows:

- Generate a corpus out of all the Wikipedia articles.
- Add in the question we care to answer and perform vectorization on the corpus using a TF-IDF method.
- Reduce the dimensionality of the TF-IDF matrix by using Singlular Value Decomposition. This way it can be ensured that the most relevant tokens are being considered.
- Run cosine similarity for the dimensionally reduced token TF-IDF matrix between the question and all of the documents and identify the top 5 articles with the highest probability of being similar.

3.2.2. TF-IDF Vectorization Score and Benchmarking

The TF-IDF score is a value between 0 and 1 that gives the weight of a term in the collection with respect to its frequency across all documents in the data set. 0 signifies no similarity, and 1 signifies a direct relationship. In practice, a document-term matrix can be converted into a TF-IDF matrix. The TF-IDF matrix allows us to eliminate the significance of words that are frequently used in a document, but present no useful context for the purposes of classification. The TF-IDF matrix is produced using the sklearn.feature_extraction.text TfidfVectorizer class fit_transform() function.

Next, the TF-IDF matrix is fed into a singular value decomposition function. Before it's dimensions are reduced, the TF-IDF matrix has a high column rank because the TfidfVectorizer will compute a value for every word located in the question plus answer context string. Using the sklearn.decomposition TruncatedSVD class, the TF-IDF matrix's column rank is reduced from a number of dimensions down to the number of words in the test query plus 250 additional feature columns. Therefore, for any test query, we are limiting the dimensionality of the TF-IDF matrix down to n rows by 250+(number of words in the query) columns.

3.3. Answer Retrieval

In order to solve any type of NLP problems, it is necessary to group words with similar meaning by place them close to each other within a particular vector space. The location of these words is represented by vectors. Therefore, words with similar meaning will have similar vectors. The process for assigning spacial vectors to each word of a vocabulary is called word embedding.

The results of an Answer Identification models depends on the type of word embedding technique used in the model. In fact Answer Identification modes could be classified based on the use of Pre-trained Contextual Embedding (PCE).

Using PCE means that the model make use of pre-training weights on a large-scale language modeling dataset instead of the model performing its own embedding. Figure 4 shows different types of Answer Identification models based on PCE. It has been shown that PCE Models have better performance than Non-PCE (Devlin).

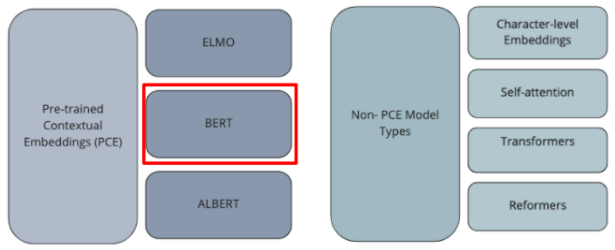

Figure 4. Types of Answer Identification Models

3.3.1. IMPLEMENTING BERT ON LARGE ARTICLES

Most BERT QA models used in answer retrieval algorithms take in n token inputs that contain the question and passage to search for the answer. The limit of how large n needs to be is traditionally dictated by the article size. While large scale BERT implementations that can take in larger documents exist, we have implemented a smaller scale BERT retrieval algorithm that takes in a maximum of 512 tokens, due to its increased levels of performance and faster run times. All of the Wikipedia articles in the training and test data sets contain significantly more than 512 tokens, thus we have partitioned the top k articles into smaller input segments of 450 tokens and have each individual fed into BERT to return an answer for each 450 token subgroup.

3.3.2. BERT FOR ANSWER GENERATION WITH LOOPING

Running BERT across an article in subsections of 450 tokens frequently returns multiple answers for a set of k articles. BERT returns these answers by normalizing the probabilities that a particular token in the 450 token set is the answer relative to the probabilities of the other 449 tokens input using a soft-max implementation, but tells us nothing about answers returned from different subsections.

Comparisons of the answers returned by different subsections of the articles are determined by taking the answers returned from BERT for each subsection that do not contain the token '[CLS]' (the token returned when an answer is not present) and retrieving the answer plus its surrounding context information from the original article. We ensure that all relevant context information is included for a particular answer by retrieving the adjacent sentences from the article and forming a new document (we call the "compressed corpus") that contains this context information for valid answers from the first pass of the BERT run on the subsections of the k articles. This compressed corpus is then re-fed back into BERT in the same manner as the subsections were prior to it as shown in figure 5 to generate a new set of answers.

By appending all of the answers and their context information into a single document we are able to leverage the sophistication of the BERT QA retrieval to differentiate between similar answers and truncate our answer list down to a single answer. The process described above is repeated until the answer list contains zero or one answer as shown in figure 5, and this answer is returned.

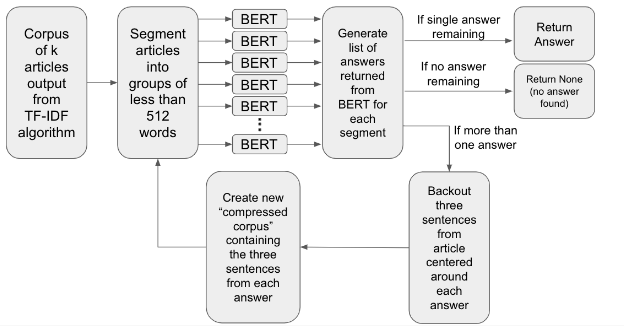
Corpus of k articles output from TF-IDF algorithm
Segment articles into groups of less than 512 words
BERT
BERT
BERT
BERT
BERT
BERT
Generate list of answers returned from BERT for each segment
If single answer remaining
Return Answer
If no answer remaining
Return None (no answer found)
If more than one answer
Backout three sentences from article centered around each answer
Create new "compressed corpus" containing the three sentences from each answer

Figure 5. Implementation of the answer retrieval algorithm using BERT to generate answers and them compare them to each other.

3.3.3. BERT MODELS TESTED

Multiple BERT models were applied to our training data sets to characterize the QA performance of the algorithm presented in figure 5. In this section we will touch on two of them, a pre-trained model that was imported from the transformers package and a custom model we trained ourselves.

SQuAD 2.0 Case Sensitive Model

Initial implementation and characterization of the answer retrieval algorithm was performed using a pre-trained BERT model from the transformers package that had been trained on the SQuAD 2.0 data set. The case sensitive model was used to match the outputs of article retriever portion of the algorithm, which was most successful when letter casing and punctuation was included.

BERT Custom Model

In an attempt to improve the performance of our question-answering system, we decided to take a blank model of the Transformers BERTForQuestionAnswering model and perform a customized training procedure on the model.

The following is a brief description of the custom model training process:

1. Process the training and test datasets
2. Encode the data
3. Convert the start and end positions into tokens using the Transformer Autotokenizer class, *Autotokenizer.encodings.char_to_token*
4. Convert the encoded tokens that represent the dataset questions and answer contexts into a numpy matrix of tensors using PyTorch *torch.as_tensor*
5. Define a blank *BERTForQuestionAnswering* model from the Transformers package (un-trained model).
6. Train the model using *torch.optimize.AdamW* as the optimizer and
7. Evaluate the model and use the predicted start and end positions to determine whether the predicted answer is correct.

We use the base Transformers BERTForQuestionAnswering model and create an instance of the model using,

BERTForQuestionAnswering.from_pretrained('bert-base-uncased')

The base model consists of the following layers:

- Bert Model
  - Embedding Layer
    - The embedding layer handles the following:
      - Word Embeddings (30522 by 768 matrix)
      - Start/End Position Embeddings (512 by 786 matrix)
      - Token Type Embeddings (2 by 768 matrix)
      - LayerNorm
      - Dropout
  - Encoding Layer
    - The encoding layer is composed of 12 identical BertLayer layers. Each of these BertLayers is composed of the following:
      - Attention Layer
      - Intermediate Layer
      - Output Layer
  - Question Answer Output Layer
    - Linear Transformation Layer
      - The linear transformation layer takes in 768 inputs from the output of the attention layers (softmax probability) and transforms the inputs into 2 outputs (start and end positions of the answer in the test paragraph).

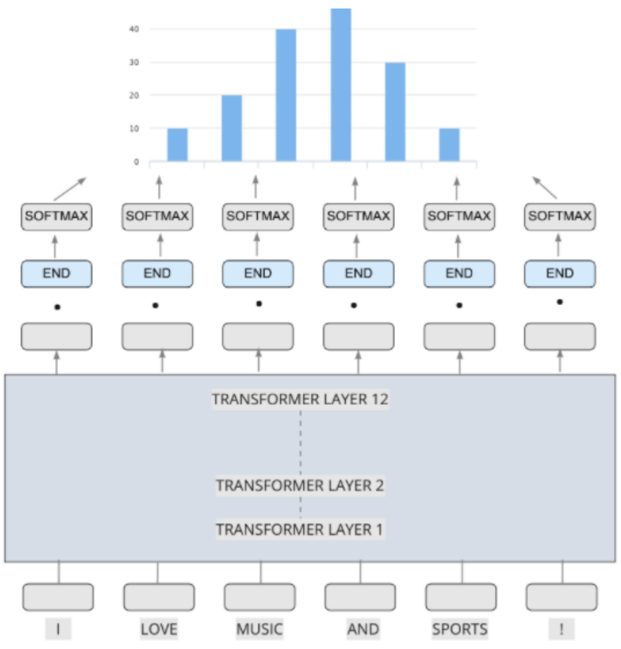

Figure 6. BERT Custom Model

The custom BERT model, shown in figure 6 was trained using 86,800 random questions from the SQuAD 2.0 training dataset. We tried multipled learning rates, batch sizes, and epochs, but were limited in our ability to fully characterize the performance of the model for varying hyperparameters, because of the overall computing time required.

Initially, we started the training process on a Macbook with an Intel i9 processor and 16GB of RAM, but the backwards propagation computations of the Bert model took very long to complete, so we looked for an alternative computing resource.

Next, we made an account with Google's Cloud Platform and ran our training code on a server with 16 CPUs and 128GB of RAM (we were denied access to servers with GPUs), this process was considerably faster, but still took almost 8 hours per epoch to train.

After some further research, we converged on a computing resource solution. We were able to train the un-trained BERT model using Google Colaboratory Pro. Colaboratory Pro gives users priority access to servers with multiple GPUs for computation. This reduced our epoch computational time for 86,800 training questions down to 1 hour and 20 minutes per epoch. The hyper-parameter5s used to train the custom BERT model are listed in table 2.

Custom-Trained (Fine-tuned) Model Loss:

Table 2. Custom-trained BERT Model Hyper-parameters

PARAMETER	DESCRIPTION
DATA	SQUAD 2.0 (86,800 QUESTIONS)
BATCH SIZE	10 QUESTIONS (EACH QUESTION IS COMPOSED OF TOKENS FOR QUESTION AND ANSWER CONTEXT WORDS AND THE TOKENS FOR THE START AND END POSITIONS OF THE ANSWERS)
TRAINING TIME	5 EPOCHS (7 HOURS) GPU: NVIDIA K80S, T4S, P4S, P100S (GOOGLE SWITCHES BETWEEN)
OPTIMIZER	ADAMW, 5E-5 LEARNING RATE LINEAR DECAY
LOSS	MLM AND NSP COMBINED

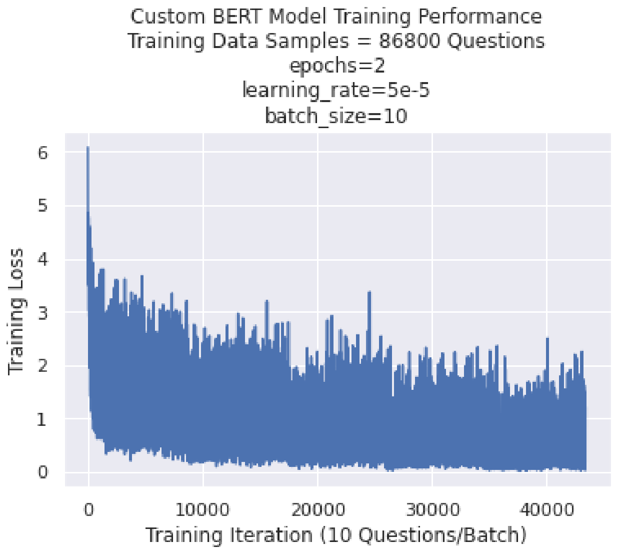

Figure 7. Training Loss over Iteration

From figure 8 we can see that the loss over training iterations is noisy and did not settle down to a very low value. Though,

4. Results

The accuracy of the entire answer retrieval algorithm can be expressed in terms of the probability of success of the document and answer retrievers as specified in equation 1.

$$P(IA) = P(IA|CA)P(FCA) \tag{1}$$

where:

IA = Identify Correct Answer
CA = Correct Article
FCA = Find Correct Article

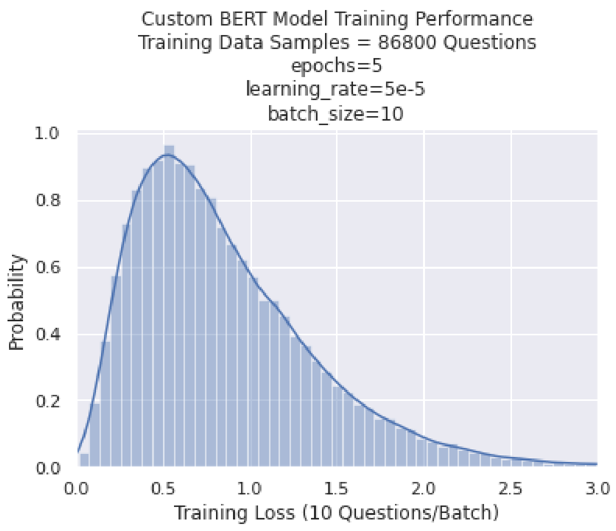

Figure 8. Training Loss Probability Distribution

To fully characterize the performance of our answer retriever we benchmarked each of the two components individually, then take these two results predict an expected accuracy to compare against the results of the entire algorithm.

4.1. Article Retriever

To benchmark our article retrieval algorithm, we want the article retriever to select k articles and determine whether the topic of any of the k articles matches the topic of the test question. To test this, we swept the number or retrieved articles (k) from 1 to 10, using 1000 random sample questions questions with answers from the 130,000 SQuAD training dataset we fed into our custom article retrieval algorithm for each k. The results are shown below in figure 9. Additionally, we also selected a sample of 5000 random questions and our article retrieval algorithm was able to correctly retrieve the articles that contains the answer to the random query with an accuracy of 74.2%. As a reminder, the original research paper (Chen) successfully identifies 78% of articles from the questions.

4.2. Answer Identification

Benchmarking of the answer retrieval part of the algorithm was completed by feeding 1000 random questions and their articles from the 100,000 question test set into each BERT model and forming a list with the answers from each of the segments. If the answer to the question was partially contained within or partially contained an answer in the list of answers, the question was marked as answered correctly.

Table 3 presents the results for the pretrained BERT model and our custom BERT model. In comparison with the full

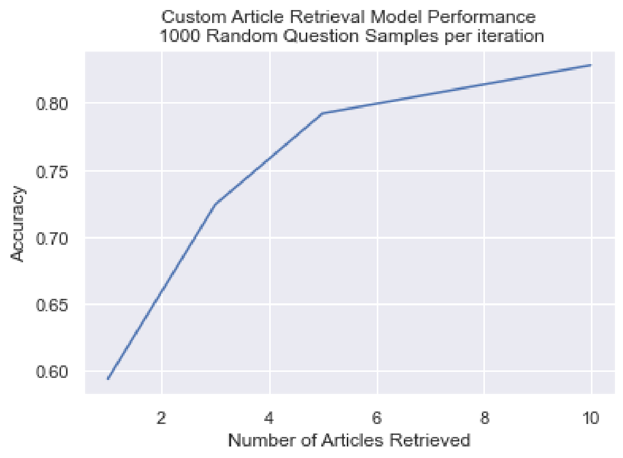

Figure 9. Accuracy for finding the article containing the answer to a given question vs the number of articles returned by the article retrieval function.

DrQA (Chen) our two models perform significantly worse. We attribute the majority of this difference in performance to extra embedding functionality implemented to match words in questions and paragraphs that enhances the weights of particular sentences when trying to identify answers. When this functionality is not present in the DrQA model, the BERT models perform much closer to the performance of the DrQA model (also shown in table 3).

Model	SQuAD 2.0 Training
BERT Pre-Trained	0.550
BERT Custom-Trained	0.230
DrQA (Chen)	0.785
DrQA w/out Q + A matching	0.594

Table 3. Comparison of individual answer retrieval models performance, tested on the SQuAD 2.0 dataset.

The custom model does not perform well enough because we are only feeding it with 86,800 questions over 5 epochs. We also do not know what the optimal hyper-parameters should be. We based our selection of hyper-parameters based on several examples and articles we read online. Even when using GPU's for tensor computations, the model takes hours to complete more than 5 epochs. This prevented us from being able to fully sweep different hyper-parameters and find an optimal training method. One of the reasons for the low performance of the custom-trained model could be due to improper start and end answer position tokenizing. If this process is done incorrectly, then the model will be trained on correct answers that do not necessarily reflect the true correct answer from the dataset.

4.3. Question Answer Retrieval

For complete benchmarking of the question answering algorithm, the article retrieval algorithm was setup to return 10 articles and the full question answering algorithm (shown in figure 5) was ran on 100 samples. A sample size of 100 was used due to the slow run time to run BERT across 10 articles for 100 questions with the looping.

Model	Exact Match	Partial Match
BERT Pre-Trained	0.280	0.440
BERT Custom-Trained	<0.05	<0.10
DrQA (Chen)	0.284	N/A

Table 4. Accuracy of BERT with looping versus DrQA model for perfect (strings are identical) and partial (model returns start or stop index within true answer).

Table 4 reports the accuracy of the pretrained BERT model along with the DrQA model. The BERT pretrained algorithm proves comparable to the DrQA on the SQuAD 2.0 dataset. For further verification of this accuracy, we use equation 1 to calculate an expected performance of 0.457 that is based on the performance of each of the two stages bench marked independently. This expected performance differs significantly from the measured performance of the pretrained model for exact matches, but aligns quite well with the performance of the partial matches, where partial matches are defined as the BERT answer returning character article indices that are contained within the actual answer. We attribute the discrepancies in the two metrics here to two major error sources; the model itself and the decoding of the embedding back to the text.

5. Conclusion

Our document retrieval question answering algorithm performs comparable to the DrQA algorithm presented by Chen. The document retrieval part of our algorithm outperforms that of the one listed in the paper, while our pretrained and custom trained models perform significantly worse than those presented in their findings. Ultimately, we attribute this discrepancy in the question answering performance to the lack of time and sufficient data required to train a good BERT custom model.

As a recommendation for future improvement, we believe a larger training set than the 86,000 question set used to train our custom BERT model is required to fully leverage the power of the 12 layer BERT model. A secondary consideration which was not pursued in this paper, would be the implementation of more complex multi-layer models such as ALBERT from the transformers package.

6. Work Distribution

- Literature Review
  - BERT - Anthony, Edgar
  - Document Retrieval - Anthony, Daniel
- Algorithm Development
  - Researched RNN Code Implementation - Edgar
  - BERT Implementation - Anthony, Daniel
  - Document Retrieval - Anthony, Daniel
  - BERT Looping - Daniel
  - BERT Custom Model Training - Anthony
- Code/Benchmarking
  - BERT - Anthony
  - Document Retrieval - Anthony, Daniel
  - BERT Looping - Daniel
- Presentation
  - Anthony, Daniel, Edgar
- Paper
  - Anthony, Daniel, Edgar

7. Citations and References

References

[1] Chen, D., Fisch, A., Weston, J., amp; Bordes, A. (2017). Reading wikipedia to answer open-domain questions. Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). https://doi.org/10.18653/v1/p17-1171

[2] Devlin, Jacob, et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding” arXiv preprint arXiv:1810.04805 (2018)

[3] Vaswani, ashish, et al. “Attention Is All You Need” arXiv preprint arXiv:1706.03762 (2017).

[4] Singh, Ankur.”End-to-end Masked Language Modeling with BERT” (2020).

[5] Horev, Rani. “BERT Explained: State of the art language model for NLP” Towards Data Science (2018)

[10] Quang -Nhat, Minh “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding” [PowerPoint slides] (2018)

[7] McCormick, Chris et al.“BERT Fine-Tuning Tutorial with PyTorch” (2019)

[8] Lan, Zhenzhong et al. “ALBERT: A Lite BERT for Self-supervised Learning of Language Representations” arXiv preprint arXiv:11909.11942 (2019).

[9] Donges, N. (n.d.). A guide to RNN: Understanding recurrent neural networks and LSTM Networks. Built In. Retrieved January 30, 2022,from https://builtin.com/data-science/recurrent-neuralnetworks-and-lstm

[10] Shri Varsheni R. ”BERT: BERT Transformer: Text Classification Using BERT.” Retrieved March 12, 2022,from https://www.analyticsvidhya.com/blog/2021/06/why-and-how-to-use-bert-for-nlp-text-classification/