Question Answering model using BERT and its derivatives

Group 21: Omkar Pradhan, Soham Shinde, Shuchita Mishra

Abstract

Question Answering Systems allow a user to express a question in natural language and get an immediate and brief response. It is an important task in the field of information retrieval and natural language processing (NLP). They enable us to properly use the huge amounts of information by helping in saving time and effort to scan the entire corpus. QA systems are found in search engines and phone conversational interfaces, and they're good at answering simple snippets of information. The dataset being used in this project is the SQuAD1.1 which contains 100,000+ text answers to every question from a Wikipedia based reading passage. Our project aims to predict these answers using embedding (w2v, SIF) and transformer-based models (BERT & variants) and compare the respective evaluation results obtained.

1.Introduction

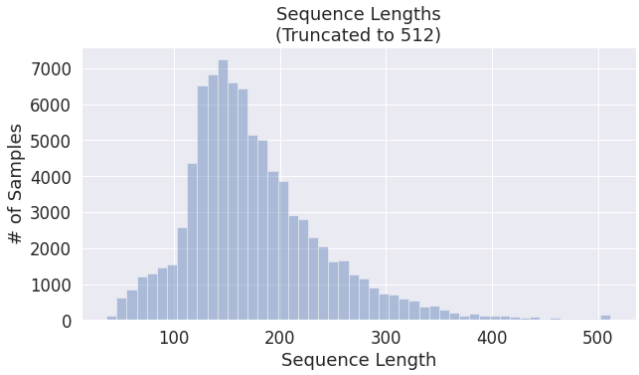
Designing a Question and Answering (QA) system is a critical problem in NLP and has tremendous applications. It pertains to creating systems which can answer natural language queries by extracting relevant answers from a vast pool of information. Thus, they form an important asset in business, academia, and day-to-day personal use as well. Our project aims to build a QnA model utilizing embedding models and transformer models such as BERT, that given a reading comprehension prose and a set of questions, must answer these questions using the knowledge gained from the reading comprehension prose. The project also uses different methods to assess the similarity of the answers, starting from a geometric metric (Cosine similarity) and including state of the art transformer-based models. We also compared the performance of the models implemented to get the best model.

2.Methods

2.1 Dataset

The data set we use is the Stanford Question Answering Dataset (SQuAD), specifically SQuAD1.1. It is a reading comprehension dataset composed of 100,000+ questions posed by crowd workers on a collection of Wikipedia articles, with each question answer being a text segment or span, from the corresponding reading passage. It consists of text answers to every question from a reading passage.

Each row consists of an id, title, context, question, answer, and answer start index. The data was originally in .json format which we parsed through a customized function to convert to a .csv file. As a part of data pre-processing, we also generated whole answer sentences from the context to be able to evaluate the similarity of the answers and hence predict the best answer for the question. We also created two separate columns for answer text and answer start for our embedding models to further normalize the data and improve efficiency. The below figure shows the distribution plot of Sequence lengths. According to BERT, the maximum sequence length is chosen to be 384 which would result in 1.2% of the records being truncated. At the same time, this length would relatively decrease the training time and be compatible with Collab provided RAM.



Distribution plot of token lengths

2.2 Models

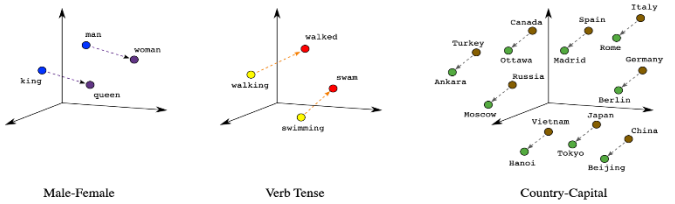
2.2.1 Word embeddings

Word2vec is an algorithm invented at Google for training word embeddings. We leverage the ability of such models and adapt them to a question answering scenario as our baseline models. Our implementations of embedding based approaches are described in detail as follows:

A. Google W2V: Word2Vec embedding trained on google news articles

The word2vec tool first constructs a vocabulary from the training text data and then learns vector representation of

words. The resulting word vector file can be used as features. Word embeddings is a technique where individual words of a domain or language are represented as real-valued vectors in a lower dimensional space.



Word2Vec embedding geometrical representation [10]

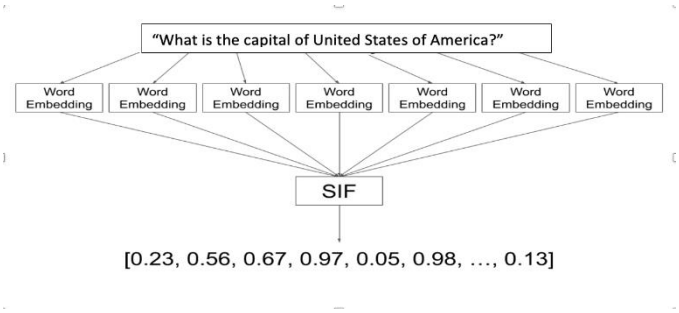
In the example above, real embeddings are shown which showcase the geometrical relationships that capture semantic relations like the relation between a country and its capital, male and female words, etc. Take the example of the first visualization: a man is to a woman what a king is to a queen. Such grammatical relations are calculated and shown geometrically using word2vec embeddings. Word2vec relies on the distributional hypothesis which states that words which often have the same neighboring words tend to be semantically similar. This helps to map semantically similar words to geometrically close embedding vectors. In our project, we have implemented the word2vec model using skipgrams. Skip gram does not predict the current word based on the context; instead it uses each current word as an input to a log-linear classifier with continuous projection layer, and predicts words within a certain range before and after the current word. Our project uses the pre-trained google-news-300 word2vec model. The model contains 300-dimensional vectors for 3 million words and phrases. Once trained, this model can detect similarity within words and can be used to predict synonyms, as well as measure the cosine similarity between distinct words. Once we have the embeddings for the question, and for every sentence in the answer text, we use cosine similarity to find the similarity between the embeddings of the question and each article sentence. The sentence which has the highest similarity with the question is predicted as the output of the model for the given question. This very simple question answering system is able to predict the correct answer with an exact match score of 69.6%.

B. SIF embeddings (Smooth Inverse Frequency)

SIF is a statistical method, which computes sentence embeddings using a simple weighted average of word vectors. Our project implements a SIF-based model for question-answering. Sentence embeddings embed an entire sentence into a vector space, like normal word embeddings like Word2Vec explained earlier. This model can identify similarities between texts after being trained. We have used this to create a Q&A system. On our dataset, we were able to apply the following SIF embeddings to a Q&A scenario as follows:

1. To start, we initialized a SIF embedding model that has been previously trained on the google-news-300 dataset. Because the context articles in our dataset are based on more than 1000 Wikipedia articles, this dataset serves perfectly as our pre-trained base. The google pre-trained SIF embedding model is then trained using all the answer text from our own dataset.
2. After the model has been trained, we start providing answers. The SIF embedding for the question is first discovered. Then, for each sentence in our answer text, we locate the SIF embedding.
3. Once we obtain the embeddings for the question and each sentence in the context, we can compare the embeddings of the question and each sentence in the answer using the cosine similarity method.

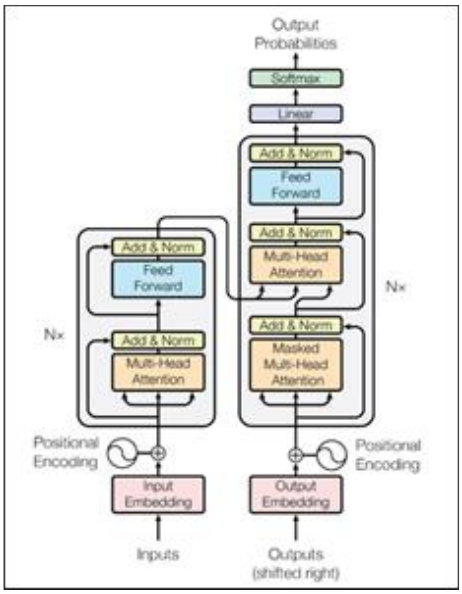
The model's anticipated output for the given question is the sentence that most closely resembles the query. In this way, the SIF embeddings help us select a sentence from the article text that has the highest embedding similarity with the given question, and outputs it as the answer. This question answering system can predict the correct answer with improved exact match score over Word2Vec embedding model (78.6%)



SIF embeddings calculation example [12]

2.2.2 Transformer models

Transformers can solve sequence to sequence tasks relying entirely on self-attention representations, simultaneously handling long-range dependencies was first introduced in the paper Attention Is All You Need.



Transformer architecture

In the above figure, the Encoder part comprises a Multi-Head Attention layer followed by Feed Forward Neural Network layer. The Decoder part comes with an additional Multi-Head Attention layer. Both the stacks have equal numbers of units. The first encoder is fed the input word-embedding sequence, which in turn transforms it. Eventually, the last encoder passes its output to all the decoders. The decoder can concentrate on the appropriate parts of the sequence using the Encoder-Decoder Attention layer.

The most crucial component of any transformer is the Self-Attention layer. It generates a representation based on the relations between the different positions of that sequence. Self-Attention is calculated multiple times with the scores being derived from the input vector and other 3 vectors. Firstly, we take the dot product of the Query vector (q1) with the key vectors (k1, k2, k3) to find the most similar key k. Then divide it by the square root of the dimension of the key vector for better gradient flow . Then these scores are normalized using the SoftMax activation function and then multiplied by the valuer vector v. Through this process, these vectors get updated as and when we process a word in the input sequence.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

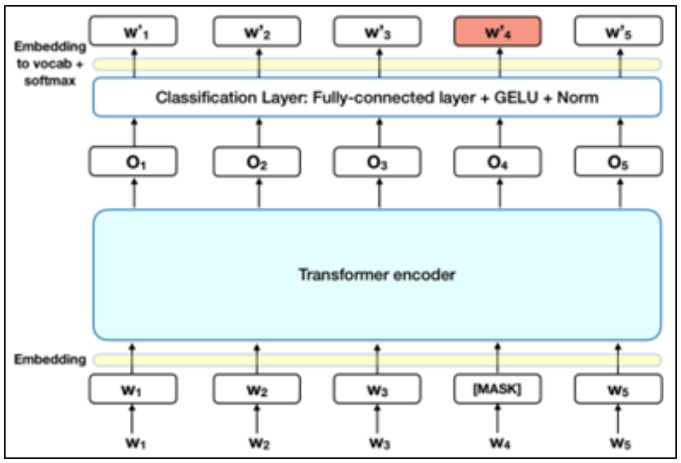
Attention- SoftMax function

The Multi-headed attention layer is an efficient modification of self-attention to handle the high-valued self-attention calculations. It uses multiple smaller sets of the vectors v, k ,q and concatenates the outputs from each set to get the final attention score z.

A. BERT

Bidirectional Encoder Representations from Transformers (BERT) is an open-source transformer-based machine-learning framework developed by Google. In our specific case, BERT model consisting of a variable number of stacked encoding layers is chosen along with its variants. Unlike traditional language models, it is designed to read bidirectionally and pre-trained on Masked Language Modeling(MLM) and Next Sentence Prediction(NSP) tasks.

MLM is done by hiding a word in a sentence and then predicting what word using its context. Prior to entering sequences into BERT, 15% of the words in each sequence are replaced with a [MASK] token. After this, the model tries to predict the value of the currently masked words depending on the context given by the non-masked words. The encoder output is passed through a classification layer and then multiplied by embedding matrix into vocabulary dimension. Lastly, we calculate the probability of each word in the vocab by SoftMax.



BERT Architecture

NSP determines the type of relationship between the two given sentences. It can be logical, sequential, or simply random. In other words, it predicts if a given sentence follows the previous sentence or not. During training, 50% of the input pairs have the second sentence as the subsequent sentence, while the other 50% contains a random sentence from the corpus. The token [CLS] is inserted at the beginning and [SEP at the end of each sentence. A sentence embedding indicating Sentence A or Sentence B is added to each token along with a positional embedding. Eventually, BERT model, Masked LM and Next Sentence Prediction are trained together with the objective of minimum combined loss.

For implementation in python, the necessary datasets are imported from SQUAD source and the JSON data is parsed. Helper functions to keep time and GPU data are written. In Data Preprocessing, we tokenize the data according to the input fields required by specific BERT models like input_ids, token_ids, attention_maks, etc. Then we make use of the base version models and Fine tune them by setting the Batch size and Data Loaders. The hyper parameters of Learning and Number of epochs are tuned as recommended.The model is trained storing the information of validation loss and accuracy for each epoch and checking for any overfitting. These learned weights are then used for predictions. Similar process is used to make the test dataset ready for evaluation.

B. ALBERT

The ALBERT architecture proposed 3 main changes over BERT as follows:

Factorized embedding parameterization: The input and hidden layer embeddings have the same size in BERT, but these two embeddings are separated in ALBERT. This is because input-level embedding (E) needs to refine only context-independent learning but hidden level embedding (H) requires context-dependent learning. This leads to 80% parameter reduction with a minor drop in performance compared to BERT

Cross-layer parameter sharing: BERT has encoder layers stacked on top of each other and the model has to do the same operation on different layers. ALBERT proposes sharing of parameters between different layers of model to decrease redundancy and improve accuracy. Three types of parameter sharing proposed are i. Share all parameters (default) ii. Only share the attention parameters iii. Only share feed forward network parameters. This step leads to 70% reduction in parameters.

Inter Sentence Coherence loss: ALBERT used a new loss called SOP (Sentence Order Prediction) over NSP (Next Sentence Prediction) loss used by BERT. NSP checks for coherence as well as topic to identify next sentence, however SOP only looks for sentence coherence.

C. DISTILBERT

DISTILBERT uses a technique called distillation which approximates BERT i.e large neural network with a smaller one. The main idea of DISTILBERT is that once a large neural network has been trained, its full output distributions can be approximated using a smaller network. In order to do this, DISTILBERT uses Kulback Leiber divergence. It also does not have token-type embeddings. DISTILBERT has 40% less parameters than bert-base-uncased, runs 60% faster while preserving over 95% of BERT's performances as measured on the GLUE language understanding benchmark.

D.ENSEMBLE : BERT+DISTILBERT

To further improve the results of our individual models, we did an ensemble of two models, BERT and DISTILBERT. The procedure for doing ensemble is as follows: For a particular question, we find the exact match score of BERT and DISTILBERT separately and take the maximum of both. The idea here is to choose the best answer of both models.

3. Evaluation

3.1 Cosine similarity:

Cosine Similarity computes the similarity of two vectors as the cosine of the angle between two vectors. It determines whether two vectors are pointing in roughly the same direction. So, if the angle between the vectors is 0 degrees, then the cosine similarity is 1. To compare the two documents, we have used cosine similarity metric in our project's embedding models. It is used to measure how similar the documents are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance because of the size they could still have a smaller angle between them. Smaller the angle, higher the similarity.

$$Cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \ \|\vec{b}\|} = \frac{\sum_1^n a_i b_i}{\sqrt{\sum_1^n a_i^2} \ \sqrt{\sum_1^n b_i^2}}$$

Cosine similarity formula [15]

After calculating the similarity scores for each sentence and question pair, we choose the sentence from context as our most probable answer which has the highest similarity score with the question.

3.2 Exact Match and F1

There are two standard approaches to scoring results on the SQuAD benchmark:

1. Exact Match

The number of predicted start indices that are equal to the correct ones are added up for this metric. It is done for the

end indices as well, such that there are actually two "points" for every sample. To handle the 3 possible answers, we score our predictions against each of the answers separately, and select the answer which best matches our prediction. So for each test sample, the highest possible score is 2.

2. F1 Score

The F1 score gives our model credit for predicting a span which partially intersects the correct one.

precision = 1.0 * num_same / len(pred_toks)

recall = 1.0 * num_same / len(all_toks)

f1 = (2 * precision * recall) / (precision + recall)

To handle the 3 possible answers, F1 score for each sample is calculated separately and the one with the highest score is considered. The final F1 score is determined by taking the average over all the test samples.

4. Results

QA snippets

```
Question:  how many schools have a similar mens basketball record to notre dame in terms of wins
Actual answer:  12
Word2Vec answer: the mens basketball team has over 1600 wins one of only 12 schools who have reached that mark and have appe
SIF Answer:  the mens basketball team has over 1600 wins one of only 12 schools who have reached that mark and have appeared
BERT answer:  12
```

Metrics Tables

Model	**train accuracy**	**val accuracy**
W2v	66.0	69.6
SIF	76.8	78.6

Table 3.1: Embedding model performance

Models	**EM**	**F1**
BERT	84.06	86.5
ALBERT	63.33	73.8
DISTILBERT	80.69	82.7
ENSEMBLE	84.67	86.9

Table 3.2: Transformers model performance

Table 3.1 and 3.2 summarizes the results of various models we implemented. For the embedding models, SIF certainly performs better as compared to Word2Vec as expected. The validation accuracy of SIF is 78.6%. For transformer models, our trained BERT model performs decently and we achieved an EM score of 86.06%. ALBERT has a less accuracy because it has significantly less number of parameters compared to the BERT model, which compromises the accuracy to a certain limit. Our ensemble model performs marginally better than the BERT model.

5. Conclusion

Discuss and Future Work(we can try more ensembles)

In this project, we worked on the Question-Answering system, which is an important and challenging problem in the domain of NLP. We used benchmark SQUAD 1.1 for testing our model. We implemented six different models on the SQUAD 1.1 dataset. All the models were trained from scratch by us without using any pre-tuned model. The implemented models can be categorized into three categories as follows.

i. Embeddings models - Word2Vec and SIF.

ii. Transformers models: BERT, ALBERT, DISTILBERT

iii. Ensemble model: DISTILBERT+BERT.

We used cosine similarity as metrics for embeddings model and EM and F1 score as metric for the other two categories. Our models performed quite well, with an EM score of above 84% for the BERT and ensemble models. In future, we would try to explore more efficient variants of BERT like ELECTRA and ROBERTa and explore top performing ensemble architectures like IE-Net which achieves accuracy of 90% with EM metrics.

6. Responsibilities

The team consists of three members: Soham Shinde, Omkar Pradhan, and Shuchita Mishra. Overall, the work has been evenly split among all the members. Each member performed research work and implemented models. Shuchita is responsible for implementing the embedding models of Word2Vec and SIF along with necessary data preprocessing and training. Soham implemented the BERT model with its DistilBERT variant. Omkar implemented the BERT model and trained the variant model of ALBERT. The ensemble model was attempted by Omkar and Soham. In the end, respective models were evaluated and summarized by all the member

6.1 References:

Code repository can be found on: https://github.com/shuchita28/CS6120-Question-Answering-model

[1] Dataset: https://rajpurkar.github.io/SQuAD-explorer/

[2] SQuAD 1.1: https://rajpurkar.github.io/SQuAD-explorer/explore/1.1/dev/

[3] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding: https://arxiv.org/abs/1810.04805

[4] A Review of Question Answering Systems: https://www.riverpublishers.com/journal_read_html_article.php?j=JWE/17/8/5

[5] Exploring cosine-simliarity functions: https://www.kaggle.com/code/lidaghr/word2vec-bert-qa-diagnostics-group6-cord-19

[6] Converting SQuAD v1.1 JSON files to .csv format: https://www.kaggle.com/code/sanjay11100/squad-stanford-q-a-json-to-pandas-dataframe

[7] Converting SQuAD v1.1 JSON files to .csv format: https://www.kaggle.com/code/jagannathpatta/reading-json-data-getting-dataframe

[8] Word embeddings introduction: https://towardsdatascience.com/word-embeddings-for-nlp-5b72991e01d4

[9] Embeddings: Translating to a Lower-Dimensional Space: https://developers.google.com/machine-learning/crash-course/embeddings/translating-to-a-lower-dimensional-space

[10] SIF embedding research paper: https://openreview.net/forum?id=SyK00v5xx

[11] SIF Embeddings calculation: https://engineering.talkdesk.com/what-are-sentence-embeddings-and-why-are-they-useful-53ed370b3f35

[12] Evaluating word embedding models: https://stackoverflow.com/questions/52645459/how-to-evaluate-word2vec-model

[13] Cosine similarity: https://www.machinelearningplus.com/nlp/cosine-similarity/

[14] Cosine similarity calculation: https://medium.com/@gshriya195/top-5-distance-similarity-measures-implementation-in-machine-learning-1f68b9ecb0a3

[15] Attention is all you need https://arxiv.org/abs/1706.03762?context=cs