Question Answering (QA) System using NLP with SQuAD

Megha Chandra Nandyala
Dept. of Electrical and Computer Engineering
University of Washington
Seattle, WA 98195
nvmcr@uw.edu

Amisha Himanshu Somaiya
Dept. of Electrical and Computer Engineering
University of Washington
Seattle, WA 98195
amishahs@uw.edu

Abstract

Question answering is a reading comprehension task, one of the most prevalent research problems in NLP. It takes 2 inputs: the passage which can be one or more paragraphs and the query which is the question posed by the user. The output is the response to the question, a single word or longer, and is a substring of the passage which is most contextually relevant to the question. This project implements a closed-domain, extractive Question-answering system using NLP with SQuAD using 2 traditional ML and 2 DL approaches. The traditional ML approaches Multinomial Logistic Regression and Random Forest use hand-crafted features only and hence are unable to comprehend the complex linguistic patterns correctly and are unable to correctly capture the contextual understanding of words and phrases leading to low scores. The third model implementation uses pre-trained BERT and achieves F1: 88.7 and EM: 85.4 high scores because BERT being based on the Transformer architecture, understands context from the entire input sequence and not just a fixed window. Also, BERT is pre-trained on a large corpus using unsupervised learning which helps it learn complex contextual patterns easily. However, BERT is a large model with 340 million parameters and hence becomes impractical to users with limited resources. Thus, the final project implementation is 'DistilBERT backbone + additional head' which retains the first few layers of DistilBERT backbone, and then with an additional head with 3 layers, the model is trained specifically for the QA task. This final implementation achieves F1: 84.56 and EM: 75.846 which is comparable to BERT in just 3 epochs while at the same time having 40% lesser parameters, lightweight, and computationally efficient.

1 Introduction

This project builds a closed-domain, extractive Question-answering (QA) System using NLP with SQuAD. The system takes 2 inputs: the passage which can be one or more paragraphs and the query which is the question posed by the user. The output is the response to the question, which can be 1 word or longer. It is a substring of the passage that is most contextually relevant to the question based on the calculated similarity score. This system is closed-domain which means the answer to the question is expected within the passage itself and in case the user poses an open-domain question, the system will still respond with an answer with the highest similarity score, but in such cases, the answer may not always be correct. Examples of this are specified in the Experimental Results section. The system is extractive and factoid which means the system takes the passage as a fact and responds with the substring extracted from the passage itself based on contextual understanding. It does not generate abstractive generative responses outside of the passage.

There are two major sub-tasks for this system, one to develop a highly efficient contextual understanding of the passage and second, to find the best substring from the passage most contextually relevant to the query by calculating similarity scores between the substrings and the query. This project builds this system using 4 approaches: 2 traditional ML baseline and 2 DL approaches. The 2 traditional ML approaches are unable to correctly comprehend complex linguistic patterns and are unable to correctly capture the contextual understanding of words and phrases leading to low scores. Thus, Multinomial Logistic Regression [1] achieved baseline F1: 22.42 and Random Forest [2] F1: 30.24, both of which are not sufficient. This motivates us to look at DL approaches such as BERT [3]. BERT is based on the Transformer architecture [4], and hence it can understand context from the entire input sequence and not just a fixed window size. Also, BERT is pre-trained on a large corpus using Unsupervised Learning which helps it understand complex contextual patterns easily. The third model implementation uses pre-trained BERT and achieves F1: 88.7 and EM: 85.4. This was implemented mainly for comparison to the final implementation. [5]

The final project implementation is 'DistilBERT backbone + additional head[3]. This implementation retains the first few layers of DistilBERT backbone and thus has a high contextual understanding of words, phrases, and patterns resulting in very high performance as compared to traditional ML baselines. The model is further appended with an additional head with 3 layers and then trained specifically for the QA task by which it achieves F1: 84.56 and EM: 75.846 which is comparable to a large model like BERT in just 3 epochs while at the same time having far lesser parameters, lightweight and computationally efficient [5].

2 Background

Question answering is one of the most prevalent research problems in NLP. Some of its applications are chatbots, information retrieval, and dialog systems, among others. It serves as a powerful tool to automatically answer questions asked by humans in natural language, with the help of either a pre-structured database or a collection of natural language documents. Models like BiDAF, BERT, and XLNet can be used for question-answering projects. Stanford Question Answering Dataset (SQuAD), and Conversational Question Answering systems (CoQA), are some datasets useful for this task [6].

2.1 SQuAD Dataset

The Stanford Question Answering Dataset (SQuAD) is a collection of question-answer pairs derived from Wikipedia articles. In SQuAD, the correct answers to questions can be any sequence of tokens in the given text. Because the questions and answers are produced by humans through crowdsourcing, it is more diverse than some other question-answering datasets. SQuAD 1.1 contains 107,785 question-answer pairs on 536 articles. One such context and pair is shown in Figure 1. SQuAD2.0 (open-domain SQuAD, SQuAD-Open), the latest version, combines the 100,000 questions in SQuAD1.1 with over 50,000 un-answerable questions written adversarially by crowd workers in forms that are similar to the answerable ones.

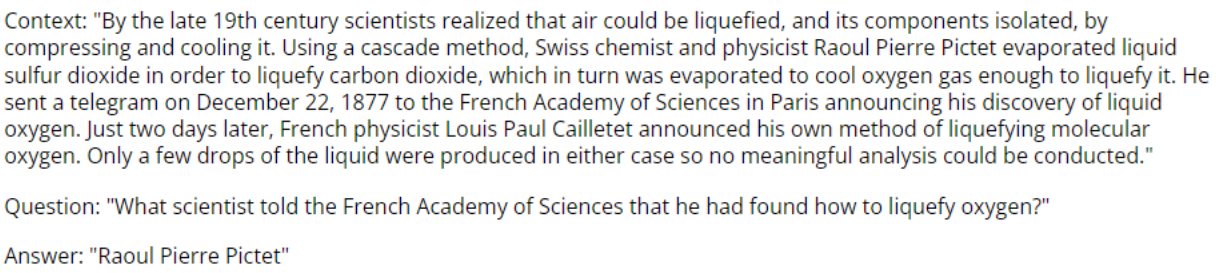

Figure 1: Sample from SQuAD Dataset

3 Data Pre-Processing

Figure 2 shows the flowchart of data pre-processing. Before giving the inputs to the model, the data needs to be pre-processed. First, we convert the passage to blobs of paragraphs using an instance of Textblob library [7]. Paragraphs are then split into sentences and a maximum length is pre-determined based on model capability. If the sentence length is more than this, it is truncated and if it is less, the sentence is padded with special padding tokens. The sentences are then split to words and tokenized. We are using the Punkt tokenizer from NLTK [8]. Then a vocabulary of words is generated using the Infersent model [9] from Facebook that uses GLoVE word to vector embeddings [10]. Then, separate embeddings are generated for the passage and for the questions. Embeddings are vectors that group words with similar contexts together based on their similarity score. For example, apples and bananas will have the same grouping but apples and car will not.

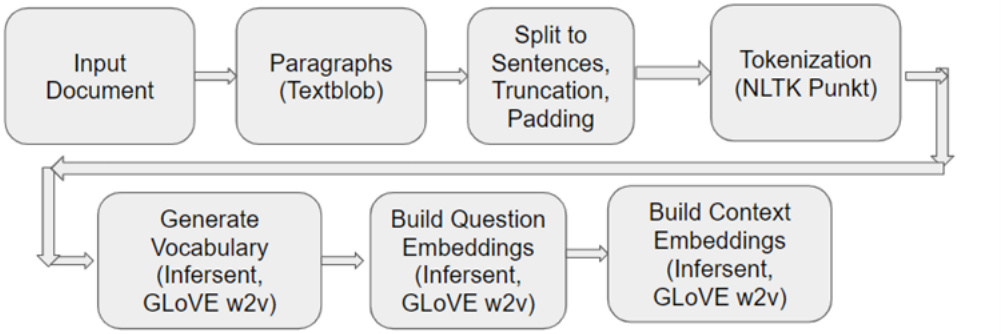

Figure 2: Data Pre-processing

4 Approach

4.1 Traditional Machine Learning Models

After the data pre-processing, hand-crafted features of Euclidean Distance and Cosine Similarity are generated for every sentence and appended to the data frame. These resultant data frames (shown in fig 3 with context, question, sentence embeddings, question embeddings, and hand-crafted features are used to fit and predict on 2 classifiers multinomial logistic regression and random forest. The performance is then evaluated using F1 and EM scores [11], [12].

	contexts	questions	answers	titles	sentences	target	question_embedding	context_embedding	cosine_similarity	euclidean
0	Architecturally, the school has a Catholic cha...	To whom did the Virgin Mary allegedly appear i...	Saint Bernadette Soubirous	University_of_Notre_Dame	[Architecturally, the school has a Catholic ch...	5	[0.1101008, 0.1142294, 0.11560897, 0.054894753...	[[0.055199962, 0.05013141, 0.047870383, 0.0162...	[0.5752636, 0.5752636, 0.5752636, 0...	[3.8162625, 3.8162625, 3.8162625, 3.8162625, 3...
1	Architecturally, the school has a Catholic cha...	What is in front of the Notre Dame Main Building?	a copper statue of Christ	University_of_Notre_Dame	[Architecturally, the school has a Catholic ch...	2	[0.10951651, 0.11030623, 0.05210006, 0.0305399...	[[0.055199962, 0.05013141, 0.047870383, 0.0162...	[0.5459254, 0.5459254, 0.5459254, 0...	[3.590196, 3.590196, 3.590196, 3.590196, 3.590...

Figure 3: Resultant Data Frame

4.2 BERT

BERT (Bidirectional Encoder Representations from Transformers) [3] is a natural language processing (NLP) model developed by Google in 2018. BERT is based on the transformer architecture[4], which is a type of neural network architecture that has proven highly effective in various NLP tasks. Transformers are designed to process sequential data by considering the entire context simultaneously, as opposed to traditional sequential models that process data in a linear fashion. Unlike traditional language models that process text in a left-to-right or right-to-left manner, BERT takes into

account both the left and right context of each word in a sentence. This bidirectional approach allows BERT to capture more comprehensive contextual information. It has been particularly successful in question-answering tasks due to its ability to capture contextual information and understand the relationships between words in a sentence. As shown in the figure 4 there are two main steps in Bert.

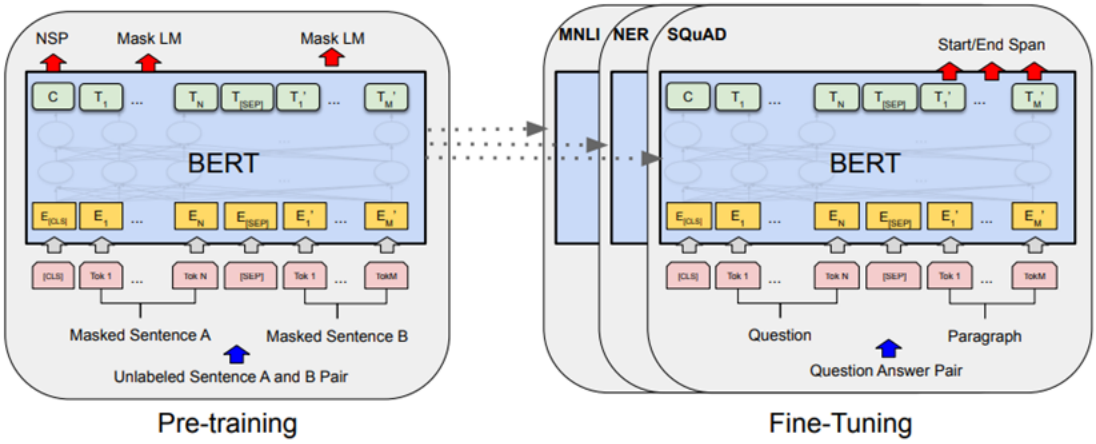

Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. `[CLS]` is a special symbol added in front of every input example, and `[SEP]` is a special separator token (e.g. separating questions/answers).

Figure 4: Bert Stages

BERT is first pre-trained on a large corpus of text data using unsupervised learning. During this pre-training phase, the model learns to predict missing words in a sentence (masked language model task) and understands the relationships between different sentences (next sentence prediction task). This pre-training allows BERT to capture general language understanding. After pre-training, BERT can be fine-tuned for specific downstream tasks, such as question answering. In the fine-tuning phase, the model is trained on a labeled dataset containing question-answer pairs. The input to the model consists of a question and a passage of text containing the answer. BERT uses WordPiece tokenization, breaking down words into smaller subwords. This allows BERT to handle out-of-vocabulary words and capture more meaningful subword representations. BERT consists of multiple layers of self-attention mechanisms. Each layer refines the representation of the input text based on contextual information. The attention mechanism enables the model to assign different weights to different parts of the input sequence, focusing on the most relevant information for a given task.BERT uses embeddings to represent words, subwords, and positional information. Positional embeddings help the model understand the order of words in a sequence.

4.3 DistilBert

Distillbert[5] is a distilled version of the BERT model, developed by Hugging Face. The primary goal of DistilBERT is to reduce the computational complexity and memory requirements of the original BERT model while retaining a similar level of performance on downstream natural language processing (NLP) tasks. The term ”distillation” in DistilBERT refers to the process of training a smaller, more efficient model using knowledge distilled from a larger, more complex model (in this case, BERT). The idea is to transfer the knowledge learned by the larger model to the smaller one, making it computationally more efficient while maintaining performance. The larger BERT model serves as the teacher, providing supervision to the smaller DistilBERT model, which acts as the student. The goal is to transfer the knowledge learned by BERT to the smaller model. During the distillation process, the student model is trained to approximate the output probabilities of the

teacher model. The key challenge is to distill the important information and knowledge from the teacher model into the smaller student model, enabling it to make similar predictions. DistilBERT retains the transformer architecture used in BERT but reduces the number of layers. DistilBERT employs parameter sharing between the encoder and the decoder layers, further reducing the number of parameters. This sharing of parameters contributes to the model's compactness. So we decided to train the question-answering system using Distilbert rather than BERT.

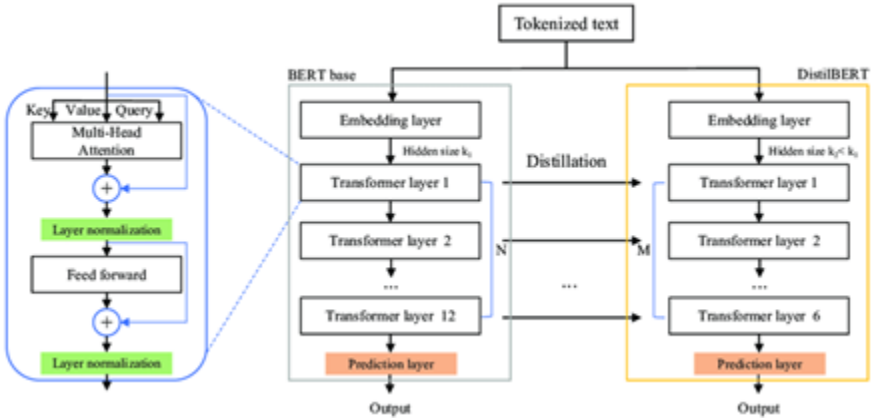

Figure 5: DistilBert

Distilbert in the hugging face library uses a single linear layer with no activation function. It reduces the 768 output dimension to 2. Similar to BERT, DistilBERT uses WordPiece tokenization to break down words into smaller subwords. The input to the model consists of tokenized versions of the question and the passage containing the potential answer. The [CLS] token is appended to the beginning of the input sequence to represent the entire context for downstream tasks.

To fine-tune the Distilbert for question-answering tasks we added custom heads. We aimed to achieve better results by carefully tuning the model architecture, focusing on the reduction of dimensions by adding more layers at the top. We added 3 custom linear layers at the top of the distillery architecture thus reducing the dimensions from 768 to 512 and then from 512 to 32 and finally 32 to 2. We added an activation function as well between these added layers. We used GELU (Gaussian Error Linear Units) as the activation function. We tried two variants of GELU. One from the original BERT paper and the other from the BERT Google Github repository. We also added a normalization layer as the final layer. Unlike traditional batch normalization, this layer directly estimates normalization statistics from the summed inputs to neurons within a hidden layer. This unique characteristic allows the normalization process without introducing dependencies between training cases

5 Experimental Results

Several experiments were conducted to decide the best hyperparameters like learning rate, batch size, and weight decay. Table 1 shows the runs. From the metrics, it can be observed that Distilbert reached its peak within a few epochs. A learning rate of 5e-05, batch size of 32, and a weight decay of 0.01 are chosen as the best hyperparameters.

Table 2 shows the final result comparisons between different models tested on the squad dataset.

6 Conclusion

This project implements a closed-domain, extractive Question-answering system using NLP with SQuAD using traditional ML and DL approaches. The 2 traditional ML approaches Multinomial Logistic Regression and Random Forest use hand-crafted features only and hence are unable to correctly comprehend the complex linguistic patterns and correctly capture the contextual understanding of words and phrases leading to low scores. The third model implementation uses pre-trained

Table 1: Hyperparameter Experiment Runs

Epocs	Learning Rate	Batch Size	Weight Decay	Train Loss	Val Loss	EM	F1
3	5e-05	32	0.01	0.5299	1.4072	60.25	75.83
5	5e-05	32	0.01	0.308	1.773	58.56	74.67
7	5e-05	32	0.01	0.153	2.23	57.8	74.17
3	5e-05	32	0.01	0.5299	1.4072	60.25	75.83
3	2e-05	32	0.01	0.898	1.318	58.488	74.58
3	5e-04	32	0.01	1.053	1.54	56.19	71.99
3	5e-05	8	0.01	0.4588	1.52	59.86	75.64
3	5e-05	16	0.01	0.597	1.356	60.04	75.81
3	5e-05	32	0.01	0.5299	1.4072	60.25	75.83
3	5e-05	32	0.001	0.2283	1.8501	59.356	75.18
3	5e-05	32	0.01	0.5299	1.4072	60.25	75.83
3	5e-05	32	0.1	0.1924	1.956	58.6	74.66

Table 2: SQuAD Dataset Classifier Results

Classifier	Configuration	Training Loss	Validation Loss	EM	F1
Logistic Regression	Tokenization: Punkt Embeddings: Infersent Hand-Crafted Features: Cosine Similarity, Euclidean Distance	32	0.01	0.5299	1.4072
Random Forest	Tokenization: Punkt Embeddings: Infersent Hand-Crafted Features: Cosine Similarity, Euclidean Distance	32	0.01	0.5299	1.4072
BERT (Pre-trained)	Learning rate: 2e-5 Optimizer: Adam Batch Size: 8 Epochs: 10	0.53	1.1	88.7	85.4
DistilBERT head 768-512-32-2	Learning rate: 5e-5 Batch Size: 32 Epochs: 3 Activation: GeLU	0.602	1.413	84.38	75.52
DistilBERT head 768-512-32-2	Learning rate: 5e-5 Batch Size: 32 Epochs: 3 Activation: GeLU New	0.597	1.356	84.56	75.846

BERT and achieves F1: 88.7 and EM: 85.4 high scores because BERT is based on the Transformer architecture, it understands context from the entire input sequence and not just a fixed window. Also, BERT is pre-trained on a large corpus using unsupervised learning which helps it learn complex contextual patterns easily. However, BERT is a large model with 340 million parameters and hence becomes impractical to users with limited computing. Thus, the final project implementation is 'DistilBERT backbone + additional head' has a high contextual understanding of words, phrases, and patterns because it retains the first few layers of DistilBERT backbone and then with an additional head with 3 layers, the model is trained specifically for QA task. This final implementation

achieves F1: 84.56 and EM: 75.846 which is comparable to large models like BERT in just 3 epochs while at the same time having 40% lesser parameters, lightweight, and computationally efficient.

7 Future Work

We used the SQuAD v1 dataset. We want to work on the SQuAD v2 dataset in the future which has more open-ended questions and includes questions that do not have answers in the context.

8 Contributions

Both team members have contributed equally towards the project design, ML and DL approaches, presentation, and report.

9 References

[1] "Sklearn.linear_model.logisticregression," scikit, https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (accessed Dec. 12, 2023).

[2] "Sklearn.ensemble.randomforestclassifier," scikit, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (accessed Dec. 12, 2023).

[3] J. Devlin, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Oct. 2018. doi: https://doi.org/10.48550/arXiv.1810.04805

[4] Vaswani, A et al. (2017) Attention is All You Need, Jun 2017. doi: https://doi.org/10.48550/arXiv.1706.03762.

[5] V. Sanh, DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter, Oct. 2019. doi:https://doi.org/10.48550/arXiv.1910.01108

[6] "Squad," The Stanford Question Answering Dataset, https://rajpurkar.github.io/SQuAD-explorer/ (accessed Dec. 12, 2023).

[7] "Simplified text processing," TextBlob, https://textblob.readthedocs.io/en/dev/ (accessed Dec. 12, 2023).

[8] NLTK, https://www.nltk.org/api/nltk.tokenize.punkt.html (accessed Dec. 12, 2023).

[9] Facebookresearch, "Facebookresearch/infersent: Infersent sentence embeddings," GitHub, https://github.com/facebookresearch/InferSent (accessed Dec. 12, 2023).

[10] J. Pennington, Glove: Global vectors for word representation, https://nlp.stanford.edu/projects/glove/ (accessed Dec. 12, 2023).

[11] "F1 score in Machine Learning: Intro & Calculation," V7, https://www.v7labs.com/blog/f1-score-guide (accessed Dec. 12, 2023).

[12] "Exact match - a hugging face space by evaluate-metric," Hugging Face, https://huggingface.co/spaces/evaluate-metric/exact_match (accessed Dec. 12, 2023).

[13] "Models," Hugging Face, https://huggingface.co/models (accessed Dec. 12, 2023).

[14] "Nlpunibo," Hugging Face, https://huggingface.co/nlpunibo (accessed Dec. 12, 2023).

[15] "Transformers Model Doc," Hugging Face, https://huggingface.co/docs/transformers/model_doc/auto (accessed Dec. 12, 2023).

[16] "Transformers Main Classes," Hugging Face, https://huggingface.co/docs/transformers/main_classes/data_collator (accessed Dec. 12, 2023).

[17] Mgreenbe, "Squad/bert: Why max_length=384 by default and not 512?," Hugging Face Forums, https://discuss.huggingface.co/t/

squad-bert-why-max-length-384-by-default-and-not-512/11693 (accessed Dec. 12, 2023).

10 Appendix

10.1 GLoVE

GLoVE[10] is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. It has a 6 billion word corpus. GloVe is essentially a log-bilinear model with a weighted least-squares objective.

The training objective of GloVe is to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence. Because the logarithm of a ratio equals the difference of logarithms, this objective associates (the logarithm of) ratios of co-occurrence probabilities with vector differences in the word vector space. Because these ratios can encode some form of meaning, this information gets encoded as vector differences as well. For this reason, the resulting word vectors perform very well on word analogy tasks, such as those examined in the word2vec package.

GLoVE measures the linguistic or semantic similarity of corresponding words by measuring the Euclidean distance or cosine similarity between 2-word vectors. The similarity metrics used for nearest-neighbor evaluations produce a single scalar that quantifies the relatedness of two words. This simplicity can be problematic since two given words almost always exhibit more intricate relationships than can be captured by a single number. To capture quantitatively the nuance necessary to distinguish man from woman, it is necessary for a model to associate more than a single number with the word pair. A natural and simple candidate for an enlarged set of discriminative numbers is the vector difference between the two-word vectors. GLoVE is designed so that such vector differences capture as much as possible the meaning specified by the juxtaposition of two words.