Question Answering and Similar Question Generation

Dr. David Quigley - Nischal Paramashivaiah - Pradyoth Srinivasan - Shreyas Gopalakrishna

I. Abstract

Question answering is a domain in the field of Information retrieval and natural language processing where a tool answers different questions posted in natural language. The answer is based on the context available for the program. Question generation is a part of the same domain where the context is used to generate questions for which the answers do not lie outside the context.

II. Problem Space

In a question answering implementation, usually a computer program may construct its answers by querying a structured database of knowledge or information, usually a knowledge base. This is known as Closed Domain Question Answering system wherein the answer gets constructed by querying a structured collection of natural language documents [1]. Alternatively there is Open Domain Question Answering which deals with questions about nearly anything, and can only rely on general ontologies and world knowledge. There is no limit to the amount of data that can be provided as input to the system and it can extract answers from literally any piece of information available [2].

In our project, we built a question answering tool based on the Stanford Question Answer (SQuAD) dataset and also try to generate similar questions based on the question and the available context for that particular question. The problem we aim to address is automation of answering in different domains and to also create knowledge base for potential questions which may need answering in these domains.

These systems will be useful in many real life scenarios. One of the biggest use case is answering day to day questions of humans acting as assistants. They can also be used in providing automated responses to user queries for products in different domains. Few other areas where these systems are getting popular are Social Media Analysis, domains where answers could be re-used or cached, Sentiment Analysis Applications and Image captioning for visual question answering. [3].

III. Approach

Our problem can be split into two parts, the question answering and generation. The initial pre-processing and features will be the same for both parts and that acted as our start point. The overview of our approach towards this project was to pre-process the data, clean it, apply feature engineering, and then use it to build a supervised and Bi-LSTM approach for answering and generation. We would then use it to compare with existing trained models. In the following subsections we go over the stages of our approach.

A. *Data gathering and processing*

This part involved gathering our data for both tasks. We used the available SQuAD data which is a gathering of Wikipedia articles. Working with unstructured data of any form is hard, when we are trying to build systems that attempt to understand natural free flowing queries of users. We have used techniques to process noisy, textual data into some structured vectorized formats that can be easily understood by any of the machine learning algorithms we are planning to use. We need more sophisticated methods than just a bag of words representation which captures the semantics, structure and the context in and around documents. We pre-process text data, where we used a simple text processor to remove special characters, extra whitespaces,punctuations, digits as well as stop words, and converting the corpus to lower case. We have also reduced multiple copies of a word by using TextBlob, a text processing library which helps to incorporate spelling correction. And alos as the data had to be converted to embedding we explored GloVe which had a vector representation for words. A script to download this data and modify it to match our format we needed is done.

B. *Embedding of Textual Data*

The main purpose of creating embedding for textual data is to represent text as a set of numerical features. By representing data as a set of dimensional vectors we are improving the ability of our model to learn from underlying data and also perform mathematical operations on our data. And this is the approach taken by most of the machine learning models which has textual input.

Representing words and sentences as embedding involved exploring GloVe and universal encoder. We started by using Glove which provides a word to vector mapping. Using this information, sentence embeddings get created for every sentence in the context as well as as the question. The next approach uses Google's Universal Sentence Encoder to embed contexts and questions. While using the universal sentence encoder to embed sentences, along with individual words the whole context of the sentence needs to be captured in the vector. The universal sentence encoder converts text into higher dimensional vectors to assist in a variety of natural language processing tasks. The universal sentence encoder is publicly available in Tensorflow hub which is what we have used to help us encode contexts and questions. Using

both these to perform training led us to choose the universal encoder since it offered less dimensional vectors compared to GloVe and were in concurrent with the computational power available with the team.

The output of this stage involved a dictionary of sentence to embedding mapping of all sentences and questions in our data. This mapping is stored as a pickle object to be used for features and training.

C. *Feature Engineering*

The main feature for our model is using the embedding for all the sentences(context) and the question. The TextBlob library was used the perform tokenization, dividing the text into a sequence of words. We have represented text as bigrams and trigrams to capture language structure. We have also used Term Frequency, Inverse Document Frequency (TF-IDF) to upscale rare words and down-weigh commonly occurring words across the corpus as features.
Another important feature is the cosine similarity of the context embedding with the question embedding. This acts as a feature which can tell us the probable position of the answer. The euclidean distance from each sentence embedding and the question also is used as a feature. The Spacy and nltk library is used to identify root words and perform parts-of speech tagging which will be used as features for question generation.

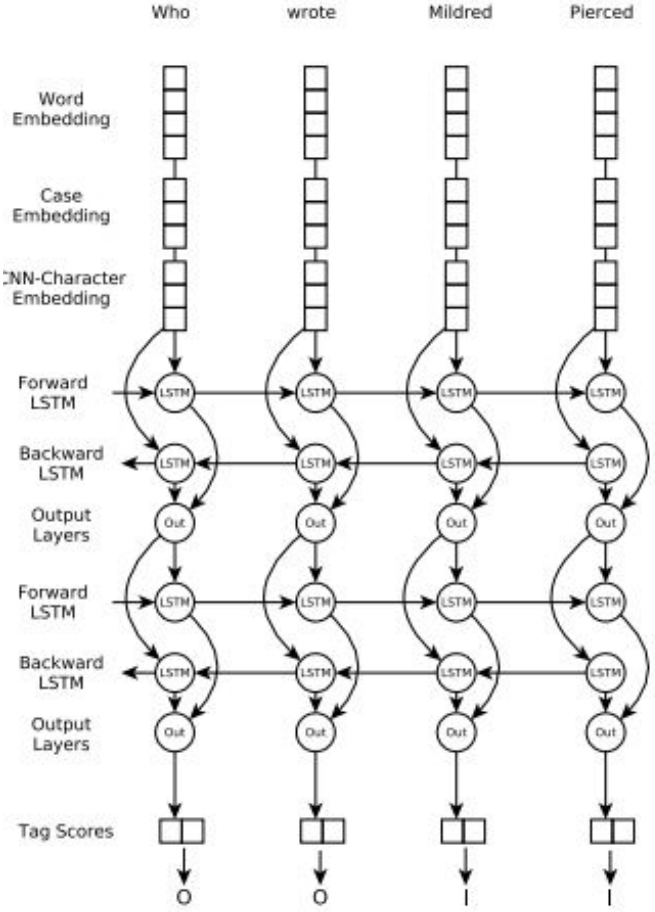
D. *Machine learning models*

1) ***Supervised Training****:* The problem was initially approached a supervised learning task of by using these features and identifying the position of answer start. Using the features generated, we used the embedding, split cosine similarity and euclidean distance as separate column features to be scaled and normalised accordingly if needed. This final structure of our train data is used to apply some of the models discussed in our class.

We, then implemented the algorithms Multinomial Logistic Regression, Decision trees, Random Forest, CatBoost and XGBoost. Among these XGBoost performed the best after some fine tuning. The best accuracy obtained using supervised learning was close to 58 percent. From this task we learnt that better models are needed to get a better matching of our answers and those models would help better in question generation as well. The next involved using Bi-LSTM with attention for our task and the output received from these layers acted as input for the question generation.[12]

2) ***Bi-LSTM Model****:* The next model we looked at was Bi-LSTM (Bi-Directional Long Short Term Memory). Bidirectional LSTM's are an extension of traditional LSTM's that can improve model performance on Question Answering problem. The architecture is based on a standard LSTM model trained to recognize the span of the subject in the question and on a linking component that links the subject span to an entity in the knowledge base. The architecture for a predicate inference is based on a standard softmax classifier ranging over all predicates as the output. The system needs to identify the relevant entity (subject) in the example question and infer the appropriate predicate. BiLSTM-Softmax architecture uses a standard BiLSTM softmax classifier to predict the property in a question where the output ranges over all properties seen during training.

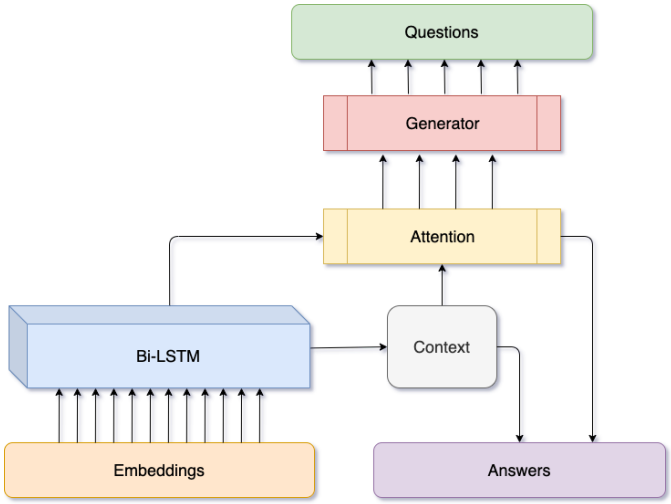
It is first important to train a Named Entity Recognizer (NER) system. The architecture is based on Bidirectional LSTMs (BiLSTM). It is composed of two LSTM layers. The model uses words and characters as features along with case of words (lowercase, uppercase). These features are concatenated and fed into a neural network. The input sentence is tokenized. Each token in the sentence is converted into a word embedding representation. Each token is also represented in terms of characters by converting the token into a matrix where each vector corresponds to a one-hot encoding vector of a character. The character matrix is fed into a Convolutional Neural Network (CNN). The CNN, then applies a convolution function to input vectors. The final step involves applying a Max-Pooling layer on the CNN output layer that represents the most important character embeddings given the token. This process is shown in the below diagram. A sigmoid function is applied to the output layer to infer the maximally scoring label for each token.



Now, coming specifically to the Bi-LSTM Softmax architecture. It is basically a BiLSTM classifier that predicts the target predicate given the question text. This is a standard model to predict multiple class labels using a softmax layer by encoding the input text using word and character embeddings. Before passing the question text to our network, we replace the entity name with a special placeholder token 'e' that abstracts away the subject mention.

Moreover, the architecture is very similar to the previously described architecture. The question text is encoded at the word and character level. Character level word embeddings are computed by applying a CNN layer with Max-Pooling on the characters of each token. Word and character embeddings are concatenated and passed through a BiLSTM layer. The final states of the BiLSTM are concatenated and fed into a feed-forward layer with softmax activation function, which calculates a probability distribution over a set of predicates [6]. Using this architecture, we predicted the most probable start and end position of answer for the question answering part of the problem.

3) ***Question Generation with BiLSTM****:* The main idea behind using the context for question generation is that question generally is formed using the words around the answer. So, identifying the key elements which form good answers in a context paragraph will help in question generation. Questions can be generally formed using the parts of speech of a sentence. Noun, verb, pronoun will help in using the start of a question such as What, when, who. Named entity recognition also acts as feature to form a question. Using the similar approach as Bi-LSTM in answering, for generation we are using Bi-LSTM with attention and context. We start by using bidirectional LSTM to encode the sentence with both a forward and backward pass to have a better understanding of the context.



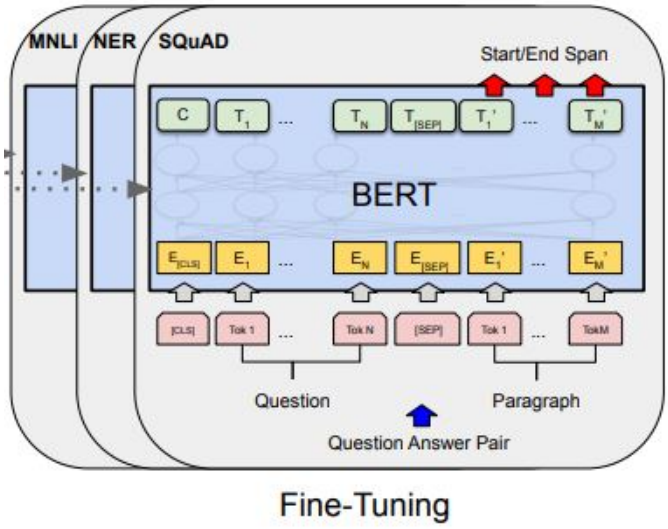
Then in order to get attention-based encoding we first get the context dependent token representation and take the weighted average. Additional features such as the parts of speech and named entity recognition are extracted out to be used as additional context information to then identify the most probable start and end places where answers are found. This is used by the generator to form questions on the paragraph passed. This is represented in the diagram.

4) ***BERT Model****:* After these approach, we then looked at BERT model to solve the problem of question answering. BERT stands for Bidirectional Encoder Representations from Transformers. BERT was introduce in 2018 by the team from Google AI Language Research team. The release of BERT model has substantially advanced the state-of-the-art in a number of Natural Language Processing tasks like Question Answering [4].

BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task specific architecture modifications. Although being conceptually simple, it is empirically quite powerful. It constantly obtains new state-of-the-art results on several natural language processing tasks like SQuAD, GLUE, MultiNLI and others.

The input to the BERT system consists of the input embeddings which in turn is the sum of the token embeddings, the segmentation embeddings and the position embeddings. As a result of this representation, the position of the word in the sentence also matters a lot for the BERT system. The Pre-training is done for the BERT model. Several unlabeled sentence pairs are fed into the system after tokenizing each words in the sentence. Once the computation happens inside the model, the required output is got from the system. Once Pre-training is done, the model can then be used for variety of tasks like MultiNLI, SQuAD and others. Specifically for the SQuAD challenge, the question answer pair along with the context is fed as the input after tokenizing. By fine-tuning the parameters, the accuracy level of the system can be boosted. This is represented by the below diagram.

The popularity of the system can be gauged by looking at the leaderboard for the top algorithms in the SQuaD challenge. As of today, the top 17 systems on the SQuAD 2.0 leaderboard are all based directly on BERT or some modifications done on top of the baseline BERT system [5]. This is an on-going challenge and new state-of-the-art models are found out every day by using BERT as the base model. We fine tuned BERT to compare with the results we obtained in BiLSTM with question generation previously.



Fine-Tuning

*5) **Comparison with Other Pre-Trained Models**:* The final approach we followed was to use several pre-trained models which were available through the Transformers package in Python. This was done so as to do a comparitive study of the accuracy levels which could be obtained through different models. Also, the parameters for each of the models was varied so as to boost the baseline accuracy obtained through such models. Most of the selected models for this task had BERT as the base model, on top of which some modifications were done to get the required system.

The models used for this task include the following : DistilBERT, RoBERTa , AlBERT-Base, AlBERT-Xlarge. These four were chosen, as they provided accuracy very close the one obtained through normal BERT model or the Bi-LSTM model.

DistilBERT is a distilled version of BERT. It has the same general architecture as BERT. The token-type embeddings and the pooler are removed while the number of layers is reduced by a factor of 2. Most of the operations used in the Transformer architecture (linear layer and layer normalisation) are highly optimized in modern linear algebra frameworks and also the variations on the last dimension of the tensor (hidden size dimension) have a smaller impact on computation efficiency (for a fixed parameters budget) than variations on other factors like the number of layers. Thus, DistilBERT was built by focusing on reducing the number of layers. As Transfer Learning from large-scale pre-trained models becomes more prevalent in Natural Language Processing (NLP), operating these large models in on-the-edge and under constrained computational training or inference budgets remains challenging. Thus DistilBERT acts as a method to pre-train a smaller general-purpose language representation model, which can then be fine-tuned with good performances on a wide range of tasks like its larger counterparts. Here knowledge distillation is used during the pre-training phase and using which it is possible to reduce the size of a BERT model by 40%, while retaining 97% of its language understanding capabilities and being 60% faster [7].

Whereas, RoBERTa is a Robustly Optimized BERT Pretraining Approach. RoBERTa was built on the basis that Language model pre-training has led to significant performance gains but careful comparison between different approaches is challenging. Also, training is computationally expensive, often done on private datasets of different sizes, and hyperparameter choices have significant impact on the final results. Thus it is important to measure the impact of many key hyperparameters and training data size. The conclusion was that BERT was significantly undertrained. By using the best model of RoBERTa achieves state-of-the-art results on many tasks. These results highlight the importance of previously overlooked design choices [8].

The final model used ALBERT is a Lite BERT for Self-supervised Learning of Language Representations. Increasing model size when pretraining natural language representations often results in improved performance on downstream tasks. However, at some point further model increases becomes harder due to GPU memory limitations and longer training times. To address these problems two parameter-reduction techniques exists. One is to lower memory consumption and the other is to increase the training speed of BERT. This led to the formation of AlBERT model that scales much better when compared to the original BERT. It also uses a self-supervised loss that focuses on modeling inter-sentence coherence, and it consistently helps downstream tasks with multi-sentence inputs. Thus, it gives quite a good accuracy while having fewer parameters compared to BERT [9].

IV. Data

We used the Stanford Question Answering Dataset (SQuAD)which is a reading comprehension dataset. The dataset consists of questions posed by workers on a set of wikipedia articles, where the answer to every question is a segment of text from the corresponding reading passage. Sometimes if the answer is not present in the context the question might be unanswerable. In our data we had a total of around 12 features which was further broken down to multiple columns in a dataframe as around 65 features. This split was using the embeddings. We had a total of 130319 entries which we used to train our models. We have taken pre-trained models and have modified the parameters to find the ideal performance on Squad 2.0 (Dev Dataset) - to do this we had to determine when the answer was not being supported by the context and return a null string in that case. The size of our train data is 40MB and test data is about 4MB. A couple of visual examples taken from our data to illustrate our work is follows:

1) **Question** - where is the world's largest ice sheet located today?

Context - The Antarctic ice sheet is the largest single mass of ice on Earth. It covers an area of almost 14 million km2 and contains 30 million km3 of ice. Around 90% of the Earth's ice mass is in Antarctica, which, if melted, would cause sea levels to rise by 58 meters. The continent-wide average surface temperature trend of Antarctica is positive and significant at 0.05 C/decade since 1957.

Answer - Antarctica

Generated questions: What covers almost 14 millionkm2 ? Where is largest single mass of ice on Earth?

2) **Question** - who lives in the imperial palace in tokyo?

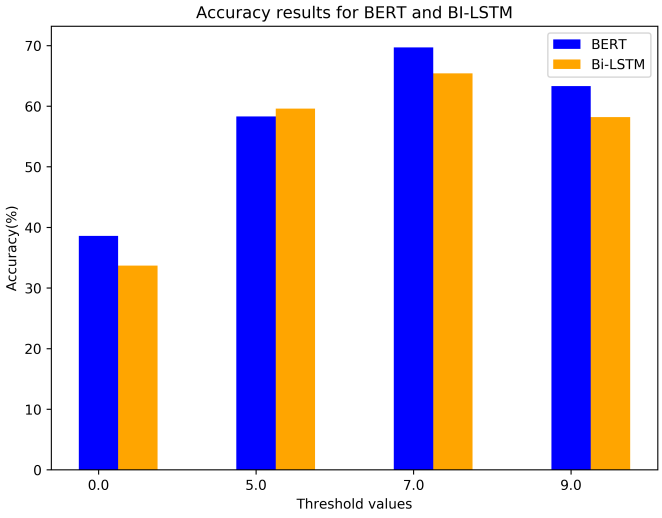
Context - The Tokyo Imperial Palace is the primary residence of the Emperor of Japan. It is a large park-like area located in the Chiyoda ward of Tokyo and contains buildings including the main palace (, Kyūden), the private residences of the Imperial Family, an archive, museums and administrative offices.

Answer - Emperor of Japan

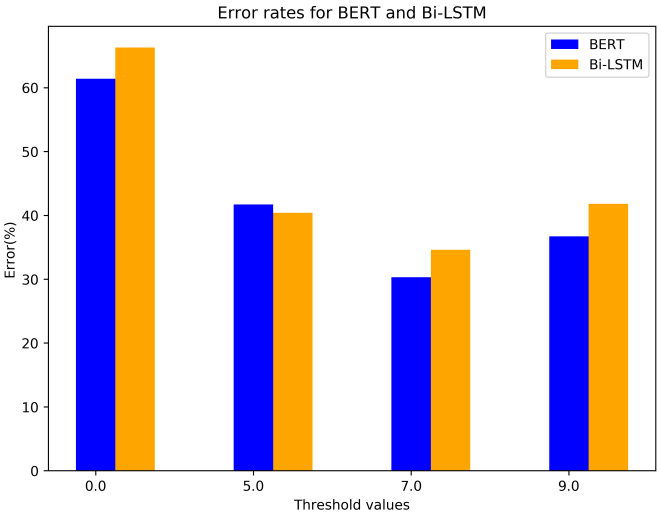
Generated questions: How is the residence?
Where is the primary residence of Emperor of Japan?

V. Results

The accuracy results for different values of threshold for BERT and BiLSTM classifier are as shown below :



The error rates for different values of threshold for BERT and BiLSTM classifier are obtained as below :



Model Used	T=0.0	T=5.0	T=7.0	T=9.0
RoBERTa_Base	33.17	54.32	**62.01**	56.25
DistilBERT_Cased	34.13	44.70	**50.96**	49.90
AlBERT_Base	30.23	47.59	**53.84**	50.84
AlBERT_XLarge	32.69	54.60	60.25	**63.09**
BERT	38.60	58.30	**69.70**	63.30
Bi-LSTM	33.70	59.60	58.40	**63.20**

The accuracy results for several pre-trained models in comparison to the BERT and Bi-LSTM are as shown in the above table. Here 'T' stands for Threshold value. The interpretation of the threshold values are discussed in the next section. The highlighted accuracy values shows the highest accuracy obtained for that particular model across various threshold values. The above table shows that BERT model is the one which gives the highest accuracy result of 69.70 % which is much better than the other pre-trained models on T value equal to 7.0. The accuracy results obtained from other models are comparable to this, although they are a bit less.

VI. Discussion

A. Interpretation of Results

The results obtained for question answering were comparable with BERT and gave correct answers. The scenarios when this failed is when embedding change such as using calculations in question or asking a question which points at beginning of a context but answer exists at the very end. We believe our model couldn't capture long connections in a group of sentences.
With the questions generated, they were basic and within the context. There were quite a few times where no questions were generated or when sometimes the formed questions were not grammatically correct.

In the comparison of models, based on the probability distribution of the start indices array and the end indices array, we have tweaked our threshold parameter and have experimented with different threshold values. Threshold value = 0.0 represents the baseline model and generally gives us the worst accuracy results and highest error rates out of all. In contrast Threshold value = 7.0 gives us lowest error rates and highest accuracies. Threshold values of 5.0 and 9.0 yield values that are much better than the baseline result, but still significantly lesser than the optimal value of 7.0. This can be attributed to the fact that only significant values of probabilities needs to be considered while generating the answer text from the context given. Plus, a lot depends on the probability matrix which is generated as the output of the model. Depending on the threshold value, the cutoffs for choosing the probabilities are specified.

B. Future Work

Currently, we are performing closed domain question answering on the datasets that we have downloaded. At the moment our QA system answers questions that ask us about certain specific domains but are not useful for domains that are beyond what they are built for.

In Open domain answering systems, the questions are not limited to predefined domains and existential domain knowledge. Instead of providing the exact context for the question to be answered we would like our system to shift through a large collection of documents to answer a question, much like how we use Google search to search for answers on the web. The main challenge of an open domain QA system is to narrow down the large collection of documents to manageable amounts, using scalable approaches to finally arrive at the answer. This could be applied in E-learning where students can ask questions to the system about any topic. We would ideally like to look at more powerful neural network based QA models to the open domain task.

The other approach would be to improve on the accuracy results which we obtained. This could be done through several methods - Initially, we can improve on the way we preprocess the text data. Embeddings such as FastText and Elmo can be used to improve results. Using other embeddings might help increase accuracy levels. Our other approach would be to use more sophisticated neural network training models by adding more layers and experimenting with the weights associated with them. But there is a trade-off between the level of sophistication and the computational hardware resources that are available at our disposal. Having a highly sophisticated model might require very large training time as well as high memory requirements. Thus striking the correct balance forms the crux of this issue. As we have used Supervised learning approaches to tackle the problem of Questions Answering, another potential approach could be to use Unsupervised methods to solve this problem. This problem could be solved by first learning to generate context, question and answer triples in an unsupervised manner, which can then be used to synthesize Extractive QA training data automatically. To generate such triples, firstly we need to sample random context paragraphs from a large corpus of documents and then random noun phrases or named entity mentions from these paragraphs as answers. Next the answers can be converted in context to fill-in-the-blank cloze questions and finally translate them into natural questions [11].

The question generation can be improved by using better networks and more rich features. An approach similar to BERT and also using GRU might be promising to obtain better questions. This problem of answer and question generation can be treated as Reinforcement learning or even with Generative adversarial network which may improve the problem space.

REFERENCES

[1] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov Natural Questions: A Benchmark for Question Answering Research Transactions of the Association for Computational Linguistics 2019 Vol. 7, 453-466 https://www.mitpressjournals.org/doi/full/10.1162/tacl_a_00276#authorsTabList

[2] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to answer open-domain questions. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). https://arxiv.org/pdf/1704.00051.pdf

[3] Question Answering: Wikipedia https://en.wikipedia.org/wiki/Question_answering

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," arXiv preprint arXiv:1810.04805, 2018. https://arxiv.org/abs/1810.04805

[5] Chris Alberti, Kenton Lee, and Michael Collins. 2019. A BERT Baseline for the Natural Questions https://arxiv.org/abs/1901.08634

[6] Sherzod Hakimov, Soufian Jebbara, and Philipp Cimiano. Evaluating architectural choices for deep learning approaches for question answering over knowledge bases. In 13th IEEE International Conference on Semantic Computing, ICSC 2019, Newport Beach, CA, USA, January 30 - February 1, 2019, pages 110–113, 2019. https://arxiv.org/pdf/1812.02536.pdf

[7] V. Sanh, et al. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. Arxiv, 2019. https://arxiv.org/abs/1910.01108

[8] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. arXiv preprint arXiv:1907.11692, 2019. https://arxiv.org/abs/1907.11692

[9] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. arXiv preprint arXiv:1909.11942, 2019. https://arxiv.org/abs/1909.11942

[10] SQuAD Explorer. https://rajpurkar.github.io/SQuAD-explorer/

[11] Patrick Lewis, Ludovic Denoyer, and Sebastian Riedel. Unsupervised question answering by cloze translation. arXiv preprint arXiv:1906.04980, 2019. https://arxiv.org/abs/1906.04980

[12] Building a question answering system. https://towardsdatascience.com/building-a-question-answering-system-part-1-9388aadff507

[13] Xinya Du1, Junru Shao2 and Claire Cardie1. Learning to Ask: Neural Question Generation for Reading Comprehension. https://arxiv.org/pdf/1705.00106.pdf