Team BigBert: A novel scoring system for question generation and extractive and boolean question answering

Advaith Sridhar, Harnoor Dhingra, Sayali Deshpande, Sajal Duppatla
{advaiths, hdhingra, sayalidd, sduppatl}@andrew.cmu.edu

Abstract

In this report, we present a novel pipeline for both question generation and answering. Our unique scoring system for Question Generation produces questions 16.0% better than the model baseline, and our unique extractive and boolean models for Question Answering produces answers more accurately and efficiently than model baseline. We performed several ablation experiments to achieve these results, which we have outlined below.

1 Introduction

In this project, we have built a question generation pipeline and question answering pipeline. The question generation pipeline was built based on manual observations from the generated questions from baseline model, several heuristics, better prompt engineering principles, and other automatic ways. The additional features added to the baseline helped improve the question generation process by around 16% in terms of the confidence level of a trained question answering model in answering the questions generated by our system. This helped us get good quality questions that were syntactically and semantically correct. Similarly, several heuristics and techniques were used in the development of the Question Answering model to improve its performance over the baseline. Question Answering was split into extractive and boolean, and then techniques such as summarization were used to allow our models to analyze the core parts of the text while adhering to model size limitations.

2 System Architecture

We have two system architectures - one for question answering and one for question generation.

2.1 QG System Architecture

The QG system receives a document and the number of questions to be generated(n) as inputs and it provides n questions that are generated from the document as output. Our QG system, shown in figure 1 consists of the modules described below.

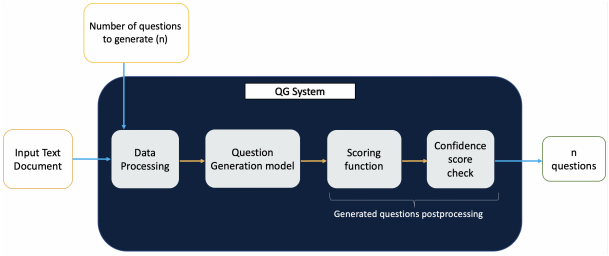

Figure 1: QG System Architecture

2.1.1 Data Processing

For processing the input Wikipedia articles, we first remove all the white space in the text documents. Then, we perform Named Entity Recognition of the entities present in the document.

One of the things that we tried for the QG system was preprocessing the text with neural co-reference to perform co-reference resolution (Lu et al., 2022). Examples of co-reference resolution can be seen in the figures 2 and 3. These figures motivate the importance of co-reference resolution in question generation, as it replaces pronouns and words that refer to an object with proper nouns that provide more context to the question generator. However, due to computation constraints, we did not include neural corefernce in our final implementation on docker. Neural coreference experiments and results can be found in Section 3.1.1.

He later returned to the Bundesliga for a three-month spell at the start of 2009 on loan to Bayern Munich, and twice went on short loans to English Premier League team Everton, in 2010 and 2012.	Landon Donovan later returned to the Bundesliga for a three-month spell at the start of 2009 on loan to Bayern Munich, and twice went on short loans to English Premier League team Everton, in 2010 and 2012.

Figure 2: neural co-reference substitution example 1 (left: before co-reference resolution; right - after co-reference resolution)

... he wasn't punished for the incident as referee Martin Atkinson didn't report it to The Football Association	... John Terry wasn't punished for the incident as referee Martin Atkinson didn't report the incident to The Football Association

Figure 3: neural co-reference substitution example 2 (left: before co-reference resolution; right - after co-reference resolution)

2.1.2 Question Generation Model

This module takes the processed document as input and generates a question as the output. We used a fine-tuned T5 model (Romero, 2021) trained on SQuAD v1.1 (Rajpurkar et al., 2016) for question generation.

The reason for choosing this model specifically was that the Squad dataset is specifically for question answering tasks based on Wikipedia articles. As the project was for question generation on Wikipedia articles, it was a design choice to use an optimally trained model for performing the same. This model takes in an answer and a text span within the document which provides context for the possible answer on which question is to be generated.

We provided sentences from the input text as 'context' for the model and passed named entities from the sentence as the 'answer' for the model to work with. The named entities PER, LOC, GPE and TIME were preferred as the questions generated from them were of better quality.

2.1.3 Scoring function

This post-processing step selects questions that are syntactically and semantically correct. Penalties (or negative scores) are awarded to a question if it fails certain conditions. 8 such conditions (or rules) were implemented after rigorous experimentation. The details of each rule are described in Section 3 and in Table 1. Questions which did not fail any rules were printed immediately, and the rest of the questions (if needed) were printed in descending order of their final score. The scoring function was developed after conducting several manual inspections of generated questions that were syntactically and semantically poor.

2.1.4 Confidence score check

Two of the rules implemented in the scoring system above used a question answering model. Confidence scores from the question answering model were used as a proxy for question quality. We used a pre-trained Roberta Base model fine-tuned on Squad 2 (Chan et al.) model as our question answering model.

2.2 QA System Architecture

The QA system receives a context document and a question as input and provides an answer to the question as the output. Our QA system consists of the modules described below.

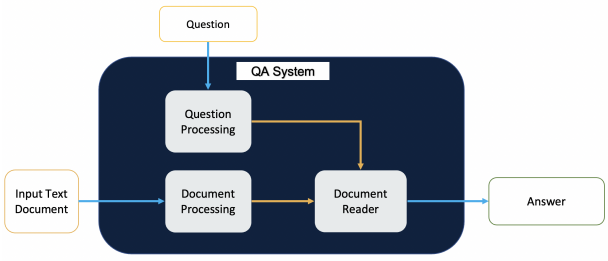

Figure 4: QA System Architecture

2.2.1 Question Processing

Question processing consists of two parts: question preprocessing and classification. We perform standard preprocessing on the question such as converting the question to lowercase to prepare a query for the QA model. After preprocessing we try to identify the type of question being asked. Some input questions may require a span extracted from the context document as the answer, whereas some questions may expect boolean ('Yes'/'No') answers. We would like to process these types of question differently in our QA system. Therefore, we identify whether a question expects an 'extractive' or 'boolean' answer. This is done by simply looking at the start of the question. If the question begins with "is", "was", "did", "do", "can", "could", etc., we classify it as a boolean question, else it is classified as extractive.

2.2.2 Document Processing

In this module, we first remove the sections which are not required such as 'References' and 'See Also'. We then convert the input raw file contents into a unified document format. The document is then further preprocessed by normalizing white spaces, removing headers and footers, cleaning empty lines, splitting the document into smaller pieces and finally creating an index to make the document ready for search. We also summarize the preprocessed document using the BART(Lewis et al., 2019) model. This summarized document is used for boolean question-answering.

2.2.3 Document Reader

This module takes the processed question and document as input and generates an answer as the output.

For extractive QA, we used a trained tiny-roberta transformers model (Liu et al., 2019). Using the 'tiny' version of roberta helps in faster inference while maintaining a good accuracy. The model selects a text span within the document that is the most probable answer to the input question. For boolean QA, we used the roberta-base model trained for text classification. The model takes the summarized document as input and outputs a boolean answer.

3 Experiments

3.1 Question Generation

We performed a thorough study of the baseline model by qualitatively analysing 50 questions generated by the model on each of the 5 categories of Wikipedia articles given to us (Chinese Dynasties, Constellations, Indian States, Languages and Pokemon Characters). We observed 18 poorly made questions, which we broke down into the following categories:

- *Lack of context in questions*: 72% (13 out of 18) of the questions were poor because they lacked sufficient context (example: *"Who is the producer of the show?"*. This is a bad question because we do not know enough about the show to determine its producer.). Lack of context is by far the largest problem with our baseline model.
- *Similar questions* 17% (3 out of 18) of questions were very similar to other questions generated by the model. We observed that this issue occurred when the model generated multiple questions from the same context sentence.
- *Grammatically incorrect questions*: 11% (2 out of 18) of generated questions were grammatically incorrect (example: *"What does the Arabic speaker generally not distinguish between Modern Standard Arabic and?"*).

We performed experiments to tackle each of the above 3 issues. We used 2 major approaches - neural coreferencing to add context, and a scoring system that penalised poorly generated questions. 12 different scoring rules to penalise poor questions, and results have been reported in Table 1 for the 8 that were included in our final submission. To judge rule quality, we used a combination of the average confidence score of a QA model in answering a set of 50 generated question, as well as human evaluation of the generated questions. We used a DistilBERT-base model (Sanh et al., 2020) trained on the SQuAD dataset (Rajpurkar et al., 2016) as our QA model for generating confidence scores. 4 rules were included in the final submission even though they had nil/negative impact on QA model confidence, as we believe they improved question quality.

3.1.1 Co-reference Resolution

In order to provide more context in sentences, we pre-processed our input data using co-reference resolution. We believed that this would enable us to generate questions that were more 'complete'. However, neural coreference did not make it to our final submission due to computational constraints.

As mentioned previously, a lack of context was the biggest drawback with our questions. The Question Generation system generated questions like *What club did she join at 14?*, *Who did she marry in December 2014?*, *Who did he lose to in December?*, etc when provided with an unprocessed input document.It is unclear as to who is referred to by the pronouns in these questions, and hence these questions are of poor quality.

On the other other hand, when the Question Generation system used input document that had substitutions in it because of co-reference resolution, it generated questions like *Who scored 25 goals in 71 appearances for the New England Revolution?*, *Who led the California Golden Bears in scoring in their first year?*, *Who joined Cypress Elite at 14?*, etc. These questions did not have pronouns and are overall more descriptive.

The final average QA model confidence score obtained on the processed input document with co-reference resolution was 96.1%. However, because of computation constraints, we did not include this processing step in our final pipeline.

3.1.2 Question Scoring

We deployed rules (listed below) to score question quality. The improvement in question quality caused by each rule can be seen in Table 1. Some rules were included in the final submission despite the drop in average QA model confidence, as we observed that they resulted in better questions. The rules are as follows:

- *Rule 1: Short questions*- Questions with length less than 4 words were penalised with a score of -5

Pipeline	Average QA Model confidence	Increase over baseline
Baseline Model	81.9%	-
(rule 7a) QA model with threshold 0.8	94.9%	15.8%
(rule 7b) QA model with threshold 0.4	85.0%	3.7%
(rule 4) Questions from the same context	84.8%	3.5%
(rule 3) Questions without context	82.9%	1.2%
(rule 1) Short Questions	81.9%	0.0%
(rule 2) Grammatically incorrect questions	81.9%	0.0%
(rule 5) Questions without wh-pronouns	81.5%	-0.5%
(rule 6) Questions without direct objects	78.9%	-3.7%
Combined Pipeline	**95.0%**	**16.0%**

Table 1: Ablation experiment to measure the impact of every rule on average QA model confidence of answering generated questions

- *Rule 2: Grammatically incorrect questions*- Questions ending with words like 'and' were penalised with a score of -5
- *Rule 3: Questions without context*- We ran Named Entity Recognition on the generated questions. Questions with 0 named entities were penalised with a score of -3 as the lack of named entities implies a lack of context.
- *Rule 4: Questions generated from the same context*- Questions generated from contexts that had already been used for were penalised with a score of -1 in order to avoid generating similar questions
- *Rule 5: Questions without wh-pronouns*- Questions without words such as 'What','When','How','Where','Why' or 'Who' were penalised with a score of -5
- *Rule 6: Questions without direct objects*- Questions whose contexts did not include direct objects were given a slight penalty of -1
- *Rule 7: Questions with low answering confidence*- Lastly, we ran each generated question through our QA model. Questions that generated a model confidence less than 0.8 were penalised with a score of -1, and questions with a confidience below 0.4 were heavily penalised by -4

3.2 Question Answering

Through our initial testing and searching for QA models, we discovered that it was difficult to handle both extractive and boolean questions. Therefore, we decided to have separate models to handle Extractive QA and Boolean QA. We were able to parse the majority of questions as boolean or extractive by looking at the start of the question.

3.2.1 Extractive Question Answering

We initially experimented with directly using pretrained transformers models like DistilBERT and RoBerta on raw text documents for extractive QA. However, these models have a character limit on the context input. This posed a problem for our particular use case as the size of context documents is large. This indicates the need to perform preprocessing on the documents and the creation of an indexed knowledge base for querying.

With our current extractive QA system, we performed a study of the model success and failure cases. Our observations are as follows:

- The model consistently performs well in answering easy and medium level questions.
  Q: Leo remains one of how many modern constellations?
  A: 88
  Expected Answer: 88
- The model is able to answer several difficult questions correctly but struggles with a few:
  Q: What two months do Leonid showers occur?
  A: January 1 and 7
  Expected Answer: November and January
- When there are multiple questions with the same answer, the model is sometimes inconsistent with the answer it returns. For example:

Q: Who added 8 more goals in 2006?
A: Clint Dempsey

Q: Who recorded the then fastest goal in US qualifying history with a chest trap and sliding shot 53 seconds into an 8–0 defeat of Barbados?
A: Dempsey

In both examples, the sentence containing the answer only has the word 'Dempsey'. Thus, there is inconsistency here. Even though both answers are technically correct, having consistency would make the model better and easier to evaluate. We could improve entity resolution to improve this.

3.2.2 Boolean Question Answering

Similarly to Extractive Question Answering, initial efforts with pretrained transformers models resulted in the raw text documents being far too large to process. Thus, we performed preprocessing to cut down on unnecessary information for our model. We also adopted a summarizer pipeline to cut out unnecessary parts of the text so that we could still use the RoBerta transformers model which had a token limit of 512. We started with the roberta-base model which we trained on the SQuAD dataset (Rajpurkar et al., 2016). With this trained model each question would produce a probability that the answer was yes, and a probability that the answer was no. We returned the option that was greater. With our current boolean QA system, we performed a study of the model success and failure cases. Our observations are as follows:

- The model consistently performs well in answering easy and medium level questions.
  Q: Is Squirtle available as a starter pokemon?
  A: Yes
  Expected Answer: Yes

- The model tends to struggle with boolean questions regarding quantities:
  Q: Does Squirtle have at least two evolutions after its base form?
  A: Yes (p=0.67)
  Expected Answer: No

  Overall, the model seems to lean towards *Yes* when finding matching entities from the question in the text.

4 Discussion

Our experimentation revealed several interesting and novel points about our Question Generation architecture:

- Adding enough context was critical for generating good questions. We did this by preprocessing the text with a neural coreference model, as well as by analysing the generated question to see if it had enough context (rules 3 and 6).

- Automatically evaluating question quality using a question answering model gave us large improvements in question quality (as apparent in Table 1). Such approaches are well worth the time-accuracy tradeoff, due to the large improvements in question quality.

- A question ranking system allows us to evaluate questions using multiple criteria, while also simultaneously ensuring that enough questions are generated. In future iterations, the scores and penalties for such an approach can perhaps be learned using a simple model like linear regression

Our QA System had the following unique contributions:

- The QA system was able to handle both extractive and boolean question-answering through separate models. We used a simple but effective method to identify if a question expects a boolean answer or not.

- Text summarization and preprocessing allowed us to effectively narrow down the passage to a smaller size that still contained a majority of the essential information so that our model could generate the best answer possible.

5 Conclusions

Overall, we were able to improve the performance of our systems over the baselines. For the Question Generation model, we observed that better prompts to the trained models help make better questions. The example of using such prompt engineering principles in our pipeline was in the data processing step with co-reference resolution. Moreover, to get syntactically and semantically correct questions, manual inspection of several questions led

to us creating a scoring function that was based on several heuristics and automatic ways to generate better questions. For the Question Answering model, we found our biggest improvement by splitting extractive and boolean QA into two different models.

As future work, we can further improve the Question Generation model by experimenting with context size, providing text summaries, or preprocessing the data in other ways to better formulate our prompts. Another area of improvement is to provide better 'answer' prompts to the trained question generation model. One possible way to do this is to analyse the syntactic tree of the sentence and then choose an entity.

The Question Answering model can also be improved using many of the measures mentioned above for the Question Generation model. Another place for improvement would be fine tuning the models, as well as finding other methods to handle boolean questions about quantities. This could include processing to explicitly count and gather terms to make such decisions.

References

Branden Chan, Tanay Soni, Malte Pietsch, and Timo Möller. Deepset/roberta-base-squad2 · hugging face.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2019. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Yaojie Lu, Hongyu Lin, Jialong Tang, Xianpei Han, and Le Sun. 2022. End-to-end neural event coreference resolution. *Artificial Intelligence*, 303:103632.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. ArXiv:1606.05250 [cs].

Manuel Romero. 2021. T5 (base) fine-tuned on squad for qg via ap. https://huggingface.co/mrm8488/t5-base-finetuned-question-generation-ap.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. ArXiv:1910.01108 [cs].