Question Answering on Natural Questions

Yiling Chen, Rena Sha
Department of Management Science and Engineering
Stanford University
{yilingc, renasha}@stanford.edu

Abstract

Question Answering is an important natural language processing task wherein a system, given a natural language question and a context document, returns the correct answer to the question. We tackle Natural Questions (NQ) [Kwiatkowski et al., 2019], a question answering challenge released by Google, by implementing a Bidirectional Gated Recurrent Unit model (BERT Bi-GRU). Our contribution is a BERT Bi-GRU model that is computationally more efficient than and outperforms the Google DecAtt+DocReader baseline [Kwiatkowski et al., 2019]. Our main findings include the viability of tf-idf for reducing the scope of the problem and the advantage in using Bi-GRUs for context comprehension.

1 Introduction

In this paper we seek to tackle Natural Questions (NQ) [Kwiatkowski et al., 2019], a question answering challenge released by Google in January 2019. Question Answering (QA) is an important natural language processing task wherein a system, given a natural language question and a context document, returns the correct answer to the question. High performance question answering is not only a benchmark task for artificial intelligence, it also has many potential applications such as helping machines better understand questions in order to develop more intelligent chatbots or building better information retrieval systems that can extract relevant information from large corpuses such as the internet.

The task required by the NQ challenge is illustrated in Figure 1. The NQ dataset differs from existing QA datasets such as Stanford Question Answering Dataset (SQuAD) [Rajpurkar et al., 2018] in a couple of crucial aspects. First, the questions presented are real queries issued to the Google search engine by users. Second, the context document provided is an entire Wikipedia page instead of a short reference paragraph. Third, the tasks that NQ seeks to achieve are much more complex, which includes: a) identifying the long answer (if any) from the document, b) identifying the short answer span (if any) from the document or c) outputting a "Yes/No" short answer where appropriate. Thus, the NQ dataset presents a challenging new problem that closely resembles real life search engine querying.

There is limited current work for the NQ challenge as it was only released recently. Out of the baseline models provided by Google we focus on the two with the best performance; the Decomposable Attention + Document Reader model (DecAtt+DocReader) [Kwiatkowski et al., 2019] and the BERT based model [Alberti et al., 2019].

The BERT based model dominates DecAtt+DocReader in terms of performance, especially in short answer prediction. Both models hover around 50-60% accuracy for long answer and 30-50% for short answer prediction compared to around 70% and 60% accuracy respectively for human performance. This signifies that, while there is significant room for improvement on these baselines, the task is challenging one.

Our approach, the BERT Bi-GRU model, combines information retrieval heuristics with state of the art neural network approaches to create a time efficient solution to the NQ challenge. We first pare down long answer candidates using tf-idf, then extract key dimensions in the data using BERT and finally feed it into a series of Bi-GRUs to obtain results.

Our model achieves a 56.4 F1 score on long answer prediction and a 20.4 F1 score on short answer prediction. Our long answer prediction outperforms the Google DecAtt+DocReader baseline model but falls short on the short answer front. There is ample potential for our model however, as we only used a small subset of the training data to develop it. In addition, our model is computationally faster and more efficient than our baselines.

2 Related Work

As the NQ challenge was only released recently there is limited published work on this topic. Instead we reference current popular methods for addressing similar question answering tasks. Perhaps the most popular is using BERT based models wherein pre-trained word embeddings from BERT are used for feature extraction from the dataset. State of the art SQuAD models all utilize this method.

A key issue we encounter in the NQ challenge is adapting neural models to process document-level input. Due to the size and complexity of neural networks they are only able to handle paragraph-sized input at a time. Current methods to address this scaling issue are bifurcated - either a pipeline model is adopted wherein a paragraph is first selected from the document and then processed, or a confidence score is calculated for each paragraph and the answer within the paragraph with the highest confidence score is returned.

Clark and Gardner [2017] addresses this issue with the TriviaQA dataset, a dataset consisting of questions and answers from trivia quizzes found online. They implement an improved pipeline model with a linear classifier using bi-directional attention and self attention. Clark and Gardner [2017] also contributes to confidence score calculation by using a shared-normalization objective. This forces the model to output globally correct confidence scores that are comparable between paragraphs but still process individual paragraphs independently.

Chen et al. [2017] addresses this issue using Wikipedia articles - a dataset that is exceedingly similar to the NQ challenge. Chen develops DrQA, a model that first retrieves a subset of Wikipedia articles relevant to the question and then annotates the short answer spans within these documents. The Document Retriever uses bigram hashing and tf-idf heuristic which we draw inspiration from in our long answer selection. The Document Reader, responsible for short answer annotation, uses a multi-layer recurrent neural network (RNN) which we simulate with bi-GRUs.

For NQ specific work, we reference two papers provided by Google proposing NQ baseline models. Kwiatkowski et al. [2019] explores a wide variety of baselines including untrained baselines, applying

Figure 1: The Natural Questions Challenge

Document-QA and a customized DecAtt+DocReader model. Alberti et al. [2019] details a BERT based implementation.

We select the DecAtt+DocReader and BERT based models as our baseline as they provide the best performance. The BERT based model is a simple application of BERT on the NQ challenge. Our model has a similar BERT foundation and further extends it with other techniques.

DecAtt+DocReader is a pipeline model that draws inspiration from natural language inference (NLI). Kwiatkowski et al. [2019] postulate that there is a fundamental difference in long answer and short answer prediction. Long answers must contain all the information needed for answer inference while short answers must be adequately encompassed by this information. As thus, long and short answers are predicted sequentially and independent of each other using a DecAtt NLI model and a DocReader model (from DrQA) respectively.

3 Approach

3.1 Data Preprocessing

The Natural Questions dataset represent each example as a quadruple `(question, wikipedia page, long answer candidates, annotation)`. Each long answer candidate is an HTML bounding box within the Wikipedia page, which may or may not be contained within another long answer candidate. The `annotation` indicates whether the training example has a long answer from the candidates, and if yes, if the long answer contains a short answer, which is a text span contained in the long answer, or a Yes/No answer.

Similar to the approach by Devlin et al. [2018] and Alberti et al. [2019], we tokenize each train example with the wordpiece vocabulary, and generate training instances in the form of `[[CLS], <question>, [SEP], <long answer candidate>, [SEP](, [PAD], ...)]`, where `[CLS]`, `[SEP]`, `[PAD]` are special tokens that signal the start of the training instance, the end of each sentence, and padding respectively.

We compute the start, end token and type for each training instance based on the annotations. If the training instance contains the short answer, we set the start and end location to point to the corresponding short answer span. If the training instance corresponds to a long answer without a short answer, we point the start position to the left `[SEP]` and the end position to the right `[SEP]`. If the instance does not correspond to the long answer, we set both the start and end position to point to the `[CLS]` token. The type of an instance is defined as: 0 - no answer; 1 - only has long answer; 2 - YES; 3 - NO; 4 - text span short answer.

The data preprocessing will be covered in more detail in Section 4.

3.2 Baseline

As mentioned above we take the Google DecAtt+DocReader model and BERT based model as our baselines. We also implemented a simple BERT based model that serves as our custom baseline. This model is simply a linear feed forward layer with output size 2 on top of the pretrained BERT model. We jointly train the linear layer with the BERT model.

In this model we aggressively downsampled null instances (i.e. `[<question>, <context>]` pairs wherein the `context` is not the long answer) by selecting 10 long answer candidates from each training example. The correct long answer candidate, if it exists, is guaranteed to be included in our downsampled set.

3.3 BERT + Bidirectional Gated Recurrent Unit (BERT Bi-GRU)

3.3.1 TF-IDF Candidates Retrieval

We first calculate the tf-idf scores for both the query and the long answer candidates. Then, we find the top long answer candidates for each query by comparing the cosine similarities between the query and candidate. We calculate tf-idf using the equations below:

$$tf(t,d) = 0.5 + 0.5 \cdot \frac{f_{t,d}}{\max\{f_{t',d} : t' \in d\}},\ idf(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

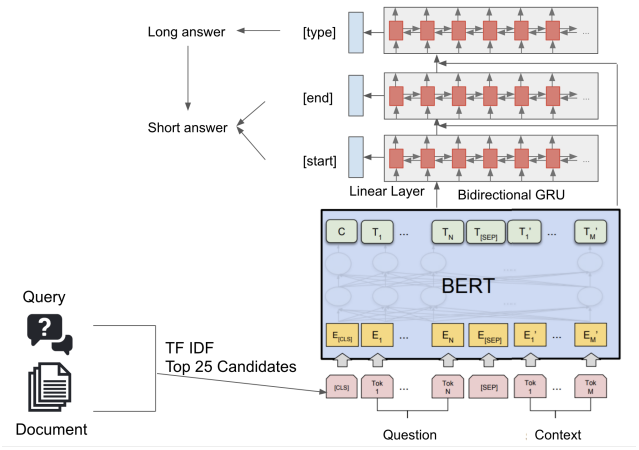

Figure 2: Overall architecture

$$tfidf = tf(t, d) \cdot idf(t, D)$$

where t is the term, d is the query/ long answer candidate, D is the full text in the Wikipedia page, N is the number of long answer candidates in the Wikipedia page and $f_{t,d}$ is the number of times t occurs in d.

During training we feed the top two candidates to our BERT layer so as to provide ample positive examples (i.e. context paragraphs that do contain the correct long answer) for our model to learn from. At test time we select the top 25 candidates to improve recall.

3.3.2 BERT

We use the BERT-base uncased model. We start with a pretrained BERT model, then jointly train our network with the BERT model. BERT uses Transformer Architecture [Vaswani et al., 2017] which has a "Multi-Head Attention" block. The Multi-Head attention block computes multiple attention weighted sums by applying different linear transformations on each "head" or focal point of attention.

Concretely, using weighted sums of the sequence elements' hidden states, the multi-head attention layer (MH) computes a representation of the sequence. The weights for the ith attention mechanism are calculated using the equation below:

$$\texttt{Attention}_i(\mathbf{h}_j) = \sum_k \texttt{softmax}(\frac{W_i^q \mathbf{h}_j \cdot W_i^k \mathbf{h}_k}{\sqrt{d/n}})$$

and the multi-head attention is computed by:

$$\texttt{MH}(h) = W^o[\texttt{Attention}_1(h), ..., \texttt{Attention}_n(h)]$$

where h_j is the hidden vector for a particular sequence, k includes every element in the sequence and W_i^q, W_i^k are matrices defined in BERT.

3.3.3 Output Layer

We sequentially predict the start location, end location and answer type in 3 different output layers. Answer type refers to the type of response to the query which includes: short answer, long answer, "Yes/No" binary response and no answer exists. Each output layer consists of a bi-GRU layer followed by a feed forward layer.

We also concatenated each prior prediction and the BERT output (see Figure 2) to predict the next item. As such the end location uses the start location information and the answer type uses both start and end location information.

3.3.4 Training Loss

The loss of our model is defined as follows:

$$L = H(l_s, s) + H(l_e, e) + H(l_t, t)$$

where $H(x, class) = -x[class] + log(\sum_j \exp(x[j]))$ is the cross entropy loss between x and $class$, $l_s, l_e \in R^{512}$ is the output of the linear layer corresponding to the start/end position, and s, e are the ground truth of start/end position from the training example annotation. $l_t \in R^5$ is the output of the linear layer corresponding to the type which we defined in section 3.1, and t is the ground truth of the type of the instance.

3.3.5 Inference

Since our final task is to make a single prediction for each question and Wikipedia page pair, at inference time, we first process the top 25 long answer candidates in the development/test examples individually to produce the evaluation instances, then each instance is processed by the model to produce the score l_s, l_e and l_t.

For each long answer candidate, we define its long answer score to be $1 - l_t[0]$, which is the likelihood that this candidate has an answer. We then report the candidate with the highest score as the long answer.

Within the proposed short answer, we then compare of the rest of l_t to determine whether to report YES, NO, or proceed with finding a text span. If we determine that a text span is a more appropriate short answer, we then go through each of the span (s, e), and calculate its score by

$$g(s, e) = l_s[s] + l_e[e]$$

The top span is then reported as the short answer.

Note that following Alberti et al. [2019], we always produces a short answer and a long answer with score, and rely on the official Natural Question evaluation script to determine the optimum threshold to produce no answer.

4 Experiments

4.1 Data

The NQ dataset includes 307,373 training examples, 7830 development examples, and 7842 held-out test examples. The training and development examples combined is 42GB, which is substantially larger than existing popular question answer datasets, like SQuAD 2.0, which is 44MB. Consequentially, we have to aggressively downsample the dataset. For training, we select two long answer candidates from each example, which gives us 115K training instances (question, long answer candidate pair). We use a random sample of 200 out of the 7830 development examples as our dev set and the full 7830 development examples as our test set as the official test set is not available to us.

Since we are feeding our training instances into BERT, we need to specify the maximum sequence length, which means some contexts might be too long to fit into one single training instance. Alberti et al. [2019] and Devlin et al. [2018] handled the issue by keeping a sliding window of the context, and generating multiple instances with overlapping text. In our analysis of the training dataset, we found the questions have a maximum of 31 tokens, and for the long answer candidates that contain a short answer, 95% of short answers are contained within the first 256 words in the long answer candidate, so we expect the impact of truncating to be small. Thus, we decided to use a maximum sequence length of 512 and simply truncate the longer context to keep the number of training instances small.

4.2 Evaluation Method

For each model we trained, we use 3 measures described in the original NQ paper to evaluate our model: F1 Measure, Precision and Recall, which we detail below.

Considering a model f_θ that maps the input (q, d) to an answer (either long or short answer), given the evaluation examples, $\{q^{(i)}, d^{(i)}, a^{(i)}\}$ for $i = 1, \cdots, n$, where each $a^{(i)}$ is a answer vector from

5-way annotation, the accuracy for this model is defined as:

$$A(f_\theta) = \frac{1}{n}\sum_{i=1}^{n} h(a^{(i)}, f_\theta(q^{(i)}, d^{(i)}))$$

where $h(a, l) = 1$ if the number of annotation if a has two or more non-null entry and l appears in a, or a has less than two non-null and l is null, and $h(a, l) = 0$ otherwise. Based on the above defined accuracy measure, precision, recall and f-measure is calculated. The F1 Score is then used to compare the different models.

4.3 Experimental Details

We use the huggingface PyTorch implementation of BERT [hug], and adapted their scripts for the SQuAD question answering task for data pre-processing, training and inference. We used the pretrained bert-base-uncased model released by Google[Devlin et al., 2018], which is a 12-layer bidirectional Transformer encoder with 12 attention heads, and 768 hidden cells in each layer. For Bi-GRU, we use a hidden size of 384. In all our experiments we used an Adam Optimizer with a batch size of 32, learning rate of 5e-5 with linear warm up on the first 10% of data and linear decay after. Since we uses GPUs with limited memory, we also leverage gradient accumulation and 16-bit float precision with dynamic loss scaling. One training epoch with 2 training instances from each of the 307,373 training examples takes around 17 hours on a NV12 Azure instance with two M60 GPUs.

4.4 Results

The evaluation results of our experiments are summarized in Table 1. Our simple baseline with a linear feed forward layer for start and end position failed to predict any short answer correctly, so we improved it by adding type prediction into the linear feed forward layer and produced an improved baseline. Our model significantly outperforms the simple baseline. We will perform an ablation analysis in Section 5 to understand where the improvement is coming from.

Note that our model outperforms the original Google DecAtt+DocReader Baseline in the long answer prediction task. Its performance is worse than the BERT baseline by Alberti et al. [2019], but the result is not surprising given that we use a smaller BERT model and train with less instances (115k vs 500k training instances). Our model is also faster to evaluate. Although Alberti et al. [2019] did not mention about training time, they mentioned evaluation takes about 5 hours with a single Tesla P100 GPU. Our model evaluation takes less than two hours on two M60 GPUs (one M60 card).

	Long Answer Dev			Long Answer Test			Short Answer Dev			Short Answer Test		
	F1	P	R	F1	P	R	F1	P	R	F1	P	R
Baseline[1]	33.7	44.4	27.2	-	-	-	0	0	0	-	-	-
Baseline + Type[1]	33.7	28.8	40.8	-	-	-	5.6	6.0	5.3	-	-	-
Bi-GRU + TD-IDF[1]	49.0	49.5	48.5	**56.4**	51.6	62.1	26.3	29.0	24.0	20.4	18.6	22.6
DecAtt+DocReader [2]	54.8	52.7	57.0	55.0	54.3	55.7	31.4	34.3	28.9	31.5	31.9	31.1
BERT$_{joint}$ [3]	64.7	61.3	68.4	66.2	64.1	68.3	52.7	59.5	47.3	52.1	63.8	44.0
Human [2]	73.4	80.4	67.6	-	-	-	57.5	63.4	52.6	-	-	-

[1] Evaluated with a 200 sample dev set as Dev and the original 7830 dev set as Test
[2] Kwiatkowski et al. [2019]
[3] Alberti et al. [2019]

Table 1: Evaluation results

5 Analysis

Next we perform ablation analysis on the components of our proposed model. The scores on the 200 sample dev set are shown in Table 2. Note that the models in this section is trained with 5961 training examples, which is 2% of the full training dataset, unless marked otherwise.

We can see that including more training data does help with performance. When using the full (down-sampled) training data, the F1 increases by 10 for both long answer and short answer task. Notably, although the correct long answer candidates are contained in the top 25 candidates retrieved by TD-IDF only 87% of the time, pre-filtering the candidates with TD-IDF not only increased the speed of our model but also boast the performance. We hypothesize this is because the score produced by our model is not well tuned for comparison between instances, and therefore including more candidates leads to poorer performance.

We also examine the impact of heavily down sampling the null instances by comparing the performance of a model trained with 50 instances per example. We do see marginal performance gain compared with using 2 instances per example, but the model trains 25 times slower. With the extra training time, it is much more efficient to use more training examples than to use more instances per example.

	Long Answer			Short Answer		
	F1	P	R	F1	P	R
Bi-GRU + TD-IDF (all data)	49.0	49.5	48.5	26.3	29.0	24.0
Bi-GRU + TD-IDF	38.8	32.3	48.5	11.9	55.6	6.7
Bi-GRU	34.0	28.7	41.7	4.02	2.9	6.7
Bi-GRU 50 instance per example	35.1	52.9	26.2	8.7	9.7	8.0
Feedforward	27.1	23.6	32.0	8.2	17.4	5.3

Table 2: Evaluation results with different approaches

When evaluating our performance against the Google BERT baseline model, we find a crucial difference between our approaches to be our treatment of data preprocessing. We choose generate separate training instances for each long answer candidate and truncate our training instances at 512 tokens. In contrast, Alberti et al. [2019] uses a sliding window over the entire document.

Upon further consideration we believe a sliding window mechanism has a couple of crucial advantages over truncation. First, for long answer candidates significantly shorter than 512 tokens, our training instances fill the gap using a padding token which does not assist with learning. With a sliding window however, if a long answer candidate is too short, the surrounding context will be learned instead. Second, since a sliding window generates multiple instances with overlapping text, for a given training instance with a short answer span, the short answer span will be learned multiple times as the window slides over. This not only increases the ratio of positive examples to null instances (which we have seen plays a crucial role in performance), it also helps the network learn when to end an answer span.

6 Conclusion

In our work we found that simple heuristics such as tf-idf should not be blindly discounted in favor of neural methods. Sometimes simple heuristics such as tf-idf are extremely useful in narrowing down the scope of the problem (i.e. paring down long answer candidates) with an added bonus of saving time and compute resources.

We also found that more complex architectures and having dedicated neurons specialize in sub-tasks (i.e. predicting start and end locations) helped performance. In particular, the ability of bi-GRUs to share features across different elements in a sequential input bidirectionally allow it to perform better than a simple feed forward network.

Another key learning is the importance of data preprocessing. A single difference such as using truncation instead of a sliding window can have lasting impacts on the distribution of data and therefore the training outcome.

In our work we have achieved long answer prediction slightly better than the DecAtt+DocReader model with short answer prediction lagging slightly behind. Although our results do not significantly outperform our baselines, this is expected as we used significantly less training and test data and

training time. Our model also has the advantage of being quicker to evaluate and more computationally efficient.

The primary limitation of our work is our limited time and resources to take advantage of the full training dataset. Another crucial limitation is our truncation of long answer candidates. In furture work we would like to explore combining our model with a sliding window pre-processing mechanism.

7 Additional Information

Mentor: Amita Kamath

References

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina N. Toutanova, Llion Jones, Ming-Wei Chang, Andrew Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: a benchmark for question answering research. *Transactions of the Association of Computational Linguistics*, 2019.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789. Association for Computational Linguistics, 2018. URL `http://aclweb.org/anthology/P18-2124`.

Chris Alberti, Kenton Lee, and Michael Collins. A bert baseline for the natural questions. *CoRR*, abs/1901.08634, 2019.

Christopher Clark and Matt Gardner. Simple and effective multi-paragraph reading comprehension. *arXiv preprint arXiv:1710.10723*, 2017.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. *arXiv preprint arXiv:1704.00051*, 2017.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

Pytorch pretrained bert. `https://github.com/huggingface/pytorch-pretrained-BERT`.