# Question Answering for Long Texts

Alfred Wechselberger
alfredhw@stanford.edu

Rekha Kumar
rekha123@stanford.edu

Shreyas Vinayakumar
vshreyas@stanford.edu

## Abstract

*Answering questions from real world text is both an important and a challenging problem with many applications. State of the art Self-Attention based models like Bert have been demonstrated to achieve good performance on question answering benchmarks, but have a high inference time, especially for longer texts. Our idea to reduce the inference time is to add a selection stage that processes chunks of a text and filters out candidates that are less likely to contain answers, and feed only the remaining portions to a self-attention based model. Our results show that by adding a selection stage consisting of a classifier based on a smaller model like DistilBert, we are able to achieve significant improvement in the run time of inference for long texts, without significantly sacrificing accuracy*

## 1. Related Work

Early approaches to Question Answering include word2vec[7] and other approaches based on extracting word embeddings. Subsequently, the Transformer[8], a self-attention based model, was proposed for better capturing the context of each word. Devlin et.al showed state-of-the-art results on a wide variety of natural language processing tasks using BERT[8], a model based on Bidirectional Encoder Representations using Transformers. This model can be pre-trained for a domain, and then fine-tuned for specific tasks like Question Answering on datasets like SQuAD v1.1.

One of the challenges with using pre-trained models relying on Self-Attention is that although it takes relatively less time to fine-tune them for specific domain datasets, the inference time is still fairly high. In fact the best models have time complexity that is quadratic in the token sequence length being considered.

For production systems inference time is a critical requirement to be able to serve customers. We wish to explore this area of improving the time for inference, while keeping the F1 score mostly intact. Prior approaches to solving this problem focused on knowledge distillation, quantization and network pruning to create simpler models that run faster[10, 11, 12] . One drawback is that these models are not as good at generalizing to different datasets and also significantly sacrifice accuracy when optimized for very low inference times.

We believe our work to be complementary to the aforementioned techniques. Our approach was inspired by related work on related to coarse-grained and fine-grained classifiers[2, 3]. The previous works focused on improving the question answering performance, as contrasted with the inference time.

## 2. Dataset and Features

We are using a subset of the Kaggle dataset for the Tensorflow 2.0 Question Answering competition, which is itself derived from the Google Natural Questions dataset. Each sample contains a Wikipedia article, and a related question. The training examples also provide the correct long and short form answer or yes/no answers for the sample, if any exist. For each article and question pair, the expectation is to predict or select long and short form answers to the question drawn directly from the article.

### **2.1.** Data Distribution

Within the training dataset, we had the following distribution:
Number of questions: 10,000
Average Answer Candidate Count per Question: 124
Average Answer Candidate Count: 324
Average Answer Candidate Length: 60
Max Candidate Length: 70,751
Percent of questions with no answer: 27%

## 3. Methods

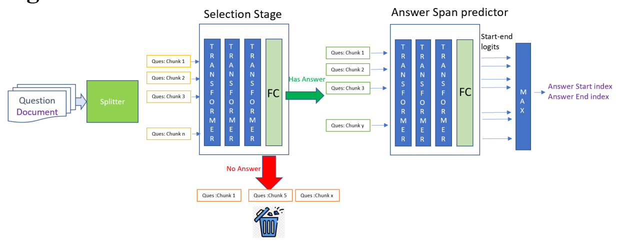
### 3.1. Baseline Model

The baseline was established taking the Bert "base-uncased" model with linear output layer for question answering, and fine-tuning it on our dataset. The model has 12 repeating layer, 128 embedding, 4096-hidden, 64-heads and 223M parameters. The maximum sequence length handled by this model is 384 tokens. A sliding window of 128 tokens was used to ensure no missing answers. The time it took for fine-tuning the above model was 6.7hrs on an AWS p2.xlarge instance.

### 3.2. Selected approach

At a high-level, the approach is to use a Selection Stage (i.e. classifier) to filter out candidates which do not contain the answer and then feeding the remaining candidates to the BERT model fine-tuned for QA on our dataset, to improve the overall inference time. It is our hypothesis that optimizing the Selection Stage for recall would potentially reduce the total inference time by passing on fewer candidates to the span prediction stage.

**Figure 1. Architecture**



The classifier must have better recall than the BERT fine-tuned model, while also having a sufficiently low inference time to offset the inclusion of an additional model. More specifically, text portions which are passed through have **increased** inference time since they are evaluated by both of the models of the models, however text portions which are not passed through have **decreased** inference time since they are evaluated only by the coarse model.

We use DistilBERT fine-tuned as a classifier model because we hypothesize that the classifier needs to have a fair understanding of the language, and yet be simpler than BERT in order to perform the inference faster.

## 4. Implementation

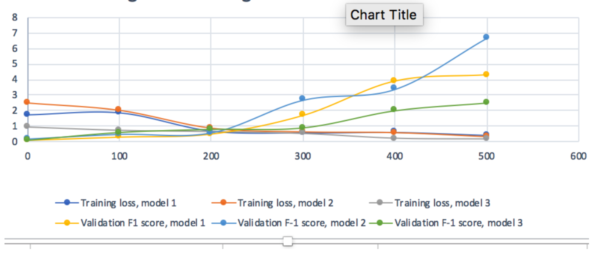
We obtained a baseline model code from Kaggle competition starter code[17]. We modified this code to add a selection stage using DistilBERT model as a classifier. The models have **6 hidden layers with 3072 units**. The output layer is a fully connected linear layer which takes the 768 outputs of the hidden layer and predicts one "No Answer" and some "Has Answer" labels(Yes or No questions which do not have a corresponding span are treated as "Has Answer"). We trained the classifier for 2 epochs on an AWS p2.xlarge instance. The machine has an Intel x5 4 core processor, 61 GB of RAM and an NVIDIA Tesla K80 GPU with 12GB of RAM. We used the same machine for both training and inference. Training time for 2 epochs was around 1.2hrs.

## 5. Results

**Table 1. Selection Stage – Recall & Precision of top models**

| **Parameters** Weight of "No Answer", dropout, initial LR, LR scheduler, gradient clipping | Validation set Precision | Validation set Recall |
|---|---|---|
| 0.3, 0.1, 2e-5 LRReduceonPlateau, 30 | 0.5306 | 0.9811 |
| 0.35, 0.1, 1e-6 WarmupLinearScheduler, ∞ | 0.5385 | 0.9246 |
| 0.29, 0.1, 1e-6 WarmupLinearScheduler, ∞ | 0.4724 | 0.9932 |

Figure 2. Training loss and validation score



Since most of the sections of the dataset did not contain the answer, this skewed the dataset for the classifier heavily towards negative examples. Instead of artificially augmenting the dataset with constructed positive examples, we chose to adjust the weighting on the "No answer" class to increase the recall. 0.3 was chosen as the weight in order to have a good recall and reasonable precision.

For the training, we used Adam with weighted decay as the optimizer. We also changed the learning rate scheduler to reduce it when the loss did not decrease for more than 1 iteration, since the loss appeared to oscillate initially. Hyperparameters such as weight decay rate, momentum term appeared to have little effect on the convergence.

At this point we experimented with different values of the Dropout Probability, since the model appeared to be overfitting the training set. The best score on validation set was achieved with dropout value = 0.1.

**Table 2. Inference Results**

| **Model** | **#Train samples** | **F1 score** | **Prec.** | **Recall** |
|---|---|---|---|---|
| BERT Baseline | 5,000 | 0.8220 | 0.7708 | 0.8809 |
| DistilBert model 1 + BERT | DistilBert 10,000 Bert 5000 | 0.5428 | 0.5588 | 0.5277 |
| DistilBert model 2 + BERT | DistilBert 10,000 Bert 5,000 | 0.5743 | 0.6041 | 0.5471 |
| DistilBert model 3 + BERT | DistilBert 10,000 Bert 5,000 | 0.6734 | 0.5892 | 0.7857 |

| **Model** | **Total Eval time/sample, avg over 100 documents** | **Eval time in classifier stage per sample** | **% document text filtered out by classifier** |
|---|---|---|---|
| BERT Baseline | 5.76 s | N/A | N/A |
| DistilBert model 1 + BERT | 2.34 s | 2.03 s | 70.1% |
| DistilBert model 2 + BERT | 2.74 s | 2.14s | 49.9% |
| DistilBert model 3 + BERT | 3.96 s | 2.03s | 30.6% |

## 5.1. Analyzing the inference time of the system

We estimated the inference time of the system as shown in Figure 1 by the following formula:

$$T = n_{forward} (t_{selection} + t_{span}) + n_{reject} t_{selection}$$

$T$ = Total Inference Time, $n_{forward}$ = Total number of positives predicted by selection stage $n_{reject}$ = Total number of negatives predicted, $t_{selection}$ = DistilBERT(selection stage model) inference time, $t_{selection}$ = BERT fine-tuned inference time

In the above formula, for our setup, $t_{selection}$ is almost constant for the given sequence length and number of hidden layers used. The variable term is the percentage of examples forwarded which directly relates to the precision of the tuned classifier. For a given amount of training effort, there is an upper bound on how much the classifier can achieve in terms of precision without hurting the recall, which can be seen from the results in Table 1.

One thing to note is that there is a tradeoff in terms of memory usage on the GPU by the Selection Stage model which could potentially limit the batch size for the Span Prediction stage model, thereby increasing the total inference time. The DistilBERT model that we picked is fairly small (~255 MB), while the BERT model is much larger (~1.2 GB), and is further dwarfed by the data itself. Therefore this effect does not appear to be significant in our implementation.

## 6. Conclusion and Future work

Our preliminary results are encouraging since the total inference time has decreased by 40%, although the F1 score of the combined model is slightly lower. The training loss continued to decrease for larger training sets, and more epochs as seen in Figure 2, which indicates that the F-1 score for the classifier can be improved with more training resources. The training time for DistilBert is also fairly low, which could make it a worthwhile investment.

For the Selection stage we would like to compare our performance with that of a naive classifier based on, say, cosine similarity between word embeddings of the question and the text portion. Although this classifier would probably not be very accurate, and would either be too aggressive or too conservative in rejecting portions, it would be interesting to see the effect on total running time on the system as it would be cheaper to run than DistilBert.

It would also be interesting to explore different simpler architectures for the classifier like BiLSTM, and apply Teacher-student training to see if they can be made accurate enough for our task[1].

We would also like to compare our result to just using a single stage with different techniques to speed up the inference: using fewer hidden layers of BERT, using pruned versions, increasing the sliding window stride at the cost of missing some answers, and reducing the floating point precision.

## 7. Contributions

**Shreyas** suggested the proposal for reducing inference time by selecting a subset of the text, conducted a literature survey of the area, developed the end-to-end pipeline combining selection and span prediction, created a framework for training the classifier and the QA model using different hyperparameters and setup the AWS infrastructure to host the project. Shreyas co-ordinated as well as carried out several experiments for hyperparameter tuning to get a better F-1 score. Shreyas's cousin Sukrit Ganesh also voluntarily contributed by writing scripts to transform the dataset and to collect some statistics on the dataset.

**Rekha** worked on Jupyter notebook setup, developed most of the baseline code, prototyped the selection stage classifier using Distilbert, made several code changes for running experiments faster, hyper parameter tuning, contributed to milestone reports and final report and presentation. Rekha suggested Question Answering as a topic and found a good dataset to use. Rekha helped develop the evaluation score for the classifier and fixed several bugs in the end-to-end pipeline.

**Alfred** suggested and documented the overall structure of the solution based on the literature review, led the setup of the local and remote development environments. Alfred also helped with locating and downloading some of the dataset files, with moving the larger files to server, and with performing some initial analysis on the dataset since it was large. Alfred also helped profile the classifier code and made a few enhancements for it to execute more quickly. Alfred also setup his local machine for training the models on more training data and performed multiple runs in order to conserve budget on AWS. Alfred also performed the initial review of the Milestone #2, and general Final Report formatting.

## 8. References

1. Tang et al. "Distilling Task-Specific Knowledge from BERT into Simple Neural Networks" https://arxiv.org/abs/1903.12136
2. Zhang et al. "Retrospective Reader for Machine Reading Comprehension" https://arxiv.org/abs/2001.09694
3. Choi et al. "Coarse-to-Fine Question Answering for Long Texts" teaches a hierarchical question answering similar to how people skim texts. https://www.aclweb.org/anthology/P17-1020.pdf
4. Kitaev et al. "Reformer: The Efficient Transformer" teaches a transformer using locality-sensitive hashing attention to efficiently handle long sequences with small memory. https://arxiv.org/abs/2001.04451
5. Le et al. "Distributed Representations of Sentences and Texts" teaches learning a fixed paragraph vector from a variable length of text https://arxiv.org/abs/1405.4053
6. Liu et al. "Text Summarization with Pretrained Encoders" https://arxiv.org/abs/1908.08345
7. Efficient Estimation of Word Representations in Vector Space by T Mikolov - 2013 https://arxiv.org/abs/1301.3781
8. Attention is All you Need by A Vaswani - 2016 https://www.aclweb.org/anthology/P17-1020.pdf
9. Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" https://arxiv.org/abs/1810.04805
10. Lan, et al. "ALBERT: A Lite BERT for Self-supervised Learning" https://arxiv.org/abs/1909.11942
11. Liu, et al. "RoBERTa: A Robustly Optimized BERT Pretraining Approach" https://arxiv.org/abs/1907.11692
12. O Zafrir, "Q8BERT: Quantized 8Bit BERT" https://arxiv.org/abs/1910.06188
13. "tf-idf". Wikipedia https://en.wikipedia.org/wiki/Tf%E2%80%93idf
14. Huggingface models for BERT and DistilBERT https://huggingface.co/transformers/
15. "TensorFlow 2.0 Question Answering". Kaggle https://www.kaggle.com/c/tensorflow2-question-answering
16. Code reference - https://www.kaggle.com/sakami/tfqa-pytorch-baseline