
Incremental Reading for Question Answering

Samira Abnar*
University of Amsterdam
s.abnar@uva.nl

Tania Bedrax-Weiss
Google
tbedrax@google.com

Tom Kwiatkowski
Google
tomkwiat@google.com

William W Cohen
Google
wcohen@google.com

Abstract

Any system which performs goal-directed continual learning must not only learn incrementally but process and absorb information incrementally. Such a system also has to understand when its goals have been achieved. In this paper, we consider these issues in the context of question answering. Current state-of-the-art question answering models reason over an entire passage, not incrementally. As we will show, naive approaches to incremental reading, such as restriction to unidirectional language models in the model, perform poorly. We present extensions to the DocQA [2] model to allow incremental reading without loss of accuracy. The model also jointly learns to provide the best answer given the text that is seen so far and predict whether this best-so-far answer is sufficient.

1 Introduction

Humans can read and comprehend text incrementally. For instance, given a piece of text, our mental state gets updated as we read [10]. We do not necessarily wait until the end of a long document to understand its first sentence. This incremental reading mechanism allows us to avoid consuming more input if we have already reached our goal, and is analogous to the problem faced by a goal-directed continuous learning system, which must also incrementally absorb new information, determine how to use it, and determine if its goals have been achieved. Inspired by how humans read and learn from text incrementally, we introduce incremental models for text comprehension. Our primary goal is to address the problem of incremental reading in the context of language comprehension and Question Answering (QA). We formulate the problem as designing a model for question-answering that consumes text incrementally.

By design and definition, Recurrent Neural Networks (RNNs), e.g. Long Short-Term Memory networks (LSTMs) [5], process data sequentially, and update their internal states as they read the new tokens. However, on tasks like Question Answering, in all the existing well-performing models, RNNs are employed in a bidirectional way, or a self-attention mechanism is employed [11, 2, 4, 8]. This means these models need to process the whole input sequence to compute the final answer. This is a reasonable approach if the input sequence is as short as a sentence, but it becomes less effective and efficient as the length of the input sequence increases.

We introduce a new incremental model based on DocQA[2], which is an RNN based model proposed for QA. The incremental DocQA performs similarly to the original system but can process the input text incrementally. We propose the use of *slicing* to build incremental models. Slicing RNNs were introduced in [13] with the motivation of enabling parallelization and speeding up sequence processing. Here, we explore using slicing to facilitate incremental processing of the input sequence.

*Work done during Samira's Internship at Google.

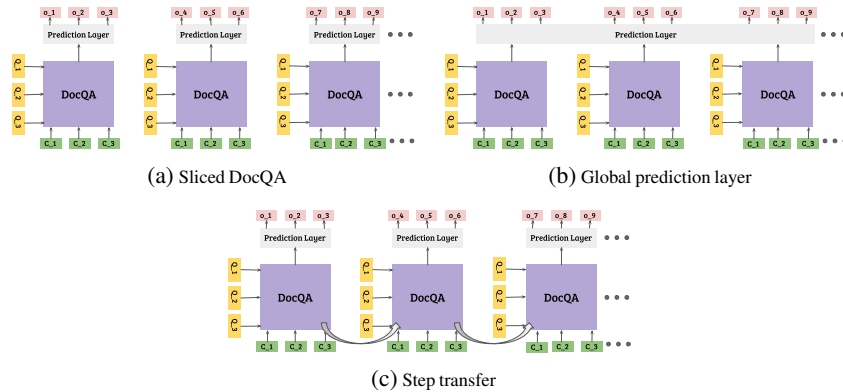


Figure 1: Sliced Models

Besides the fact that incremental reading is more plausible from the cognitive perspective, it can also provide an inductive bias for the models, which will make it easier for them to find a more generalizable solution [1]. Moreover, incrementality allows the model to be applied to tasks where we do not have the whole input in the beginning, and we need to compute some intermediate outputs. E.g. when the input is a stream, or we are in the context of a dialogue. We observed that, even if the whole input text is available, it is not always necessary to read the whole text to answer a given question. In fact, our model achieves its highest performance when it is limited to read only a few tokens after the answer, rather than when it is allowed access to the entire context. Considering this, we also augment the incremental DocQA with an “early stopping” strategy, where it stops consuming the rest of the input text as soon as it is confident that it has found the answer. Learning to stop reading or reasoning when the goal is reached is addressed for tasks like text classification [3] and question answering [9, 12, 6], but the main challenge is that implementing early stopping strategies is only possible when we have an incremental model.

2 Architecture of the models

We use DocQA as the baseline model [2]. The architecture of DocQA is illustrated in Figure 4a. The output of this model is two vectors with the length of the given context. One of the vectors indicates the probability of each token in the context to be the start token of the answer span. The other vector indicates the probability of each token in the context to be the end of the answer span. The gold labels indicate the ground truth begin and end of the answer spans. DocQA is inherently bidirectional thus making it hard to process the input incrementally. It is possible to remove the bidirectionality by replacing the bidirectional layers with single-directional layers and replacing the global attention with an attention layer that only attends to the past, but this change significantly reduces performance.

Sliced DocQA In order to enable the DocQA model to process the context sequence incrementally despite having bidirectional RNN and attention layers, we use the concept of slicing [13]. We divide the context sequence into slices and apply the model to each slice. We call this “Sliced DocQA”. We explore different ways of using Sliced DocQA for incremental question answering.

Sliced Prediction Layer In the simple sliced model, we predict the output for each slice independently. Thus, as the model processes each slice, it predicts whether the answer span exists in that slice or not. We call this *sliced prediction* (see Figure 1a). To aggregate, we use a *softmax* layer on the concatenation of the outputs of all slices. This architecture allows us to process the input incrementally, but each slice is now processed independently: i.e., when reading the second slice, the model can make no use of what was read in the first slice. This architecture hence ignores the order of the slices. We propose two solutions to solve this issue, discussed in more detail below. The first solution is to let the model access all representations from all slices in the prediction layer: we call this strategy *global answer prediction*. The second solution is a novel mechanism to transfer knowledge between slices called *step transfer*.

Global Answer Prediction While the sliced model processes the context sequence in slices, in the last layer, we use the encoded information from all the slices of the context to predict the answer. This means the prediction layer is not sliced. This model is illustrated in Figure 1b. Global answer

prediction can be made incremental by applying the prediction once for each slice, with information from the future slices masked out.

Step Transfer To connect slices incrementally, we can transfer the knowledge between slices by using the knowledge learned until the current slice to initialize the next slice, as illustrated in Figure 1c. Thus, at the global level we have a uni-directional RNN, but, locally, we can have bidirectional or self-attention layers. To do this, we use encoded information from the current slice as an input to a fully connected network to predict the initial state for the next slice.

Early Stopping These incremental models can be used to support *early stopping* [3, 12]. We use a supervised approach to predict when the model should stop, with a classifier that is simply trained on detecting whether an answer is contained in a given context or not. We train this classifier in a multi-task framework by adding an extra objective function to the QA system. Hence, at each training step, the model not only tries to predict an answer but also predicts whether the true answer is within the currently processed input or not. The early stopping classifier is a two-layer fully connected neural network with *RELU* activations and a *sigmoid* output layer. The input to this network is the average of the representations in the last layer of the current slice, and the output is a scalar indicating the probability of stopping. At test time the overall decision as to whether to stop early is based on thresholding the cumulative sum of these predictions, i.e. the probability of stopping at slice i is the sum of the predicted probabilities of stopping at slices 1 to i . Equation 1 shows how we compute the loss for the early stopping model.

$$stop_loss = \sum_i (predicted_stop_prob_i - gold_stop_label_i)^2 * extra_length_i \quad (1)$$

Here $extra_length_i$ is a factor for scaling the early stopping loss, based on the distance from the gold stop point, in particular: $extra_length = \log(\max(dist_threshold, |length_read - answer_end|))$. In this equation, $dist_threshold$ is a minimum number of tokens after which we start scaling the loss. Thus, in case the model has not yet reached the answer and decides to stop it will be punished more if it is too early in contrast to when it will reach the answer by reading a few extra tokens. Similarly, if the model has already passed the answer span, the more it reads, the bigger the loss will be. If we are in a distance less than the $dist_threshold$ from the gold stop point, the scaling factor is $\log(dist_threshold)$. In the end, this loss is added to the answer prediction loss to form the total loss. If the true answer span is not before the chosen stopping point, the answer prediction loss is set to zero for all possible answers, and the model is only trained to choose a better stopping point.

3 Experiments and Discussion

Task and Dataset: SQuAD We study the performance of incremental models on short answer questions answering.

We Evaluate our model on SQuAD v1 [7] which is a reading comprehension dataset. It contains question and context pairs, where the answer to the question is a span in the context. We will open source the code for reproducing the experiments.

3.1 Experiments

We experiment with different slice sizes for Sliced DocQA in three different modes: with a sliced prediction layer, with a sliced prediction layer and step transfer, without step transfer but with a global prediction layer. In the sliced models, the size of each slice can play a crucial role. The extreme cases are when the length of the slices are 1, which means we have no bidirectional layer, and when the slices are as long as the whole sequence, which means we have a fully bidirectional model. Notice that when slice size is 1, the sliced model with step transfer is not equivalent to the uni-directional DocQA, since the attention layer is implemented differently.

Effect of slice size Figure 2 shows the performance of the Sliced DocQA models with respect to different slice sizes. With a sliced prediction layer, as expected, increasing the slice size leads to an increase in the performance. In this case, having a slice size of 1 means to predict the answer based on single word representations and still, we can get an accuracy of 31%. We can explain this by the fact

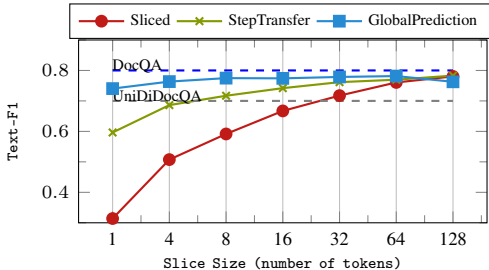
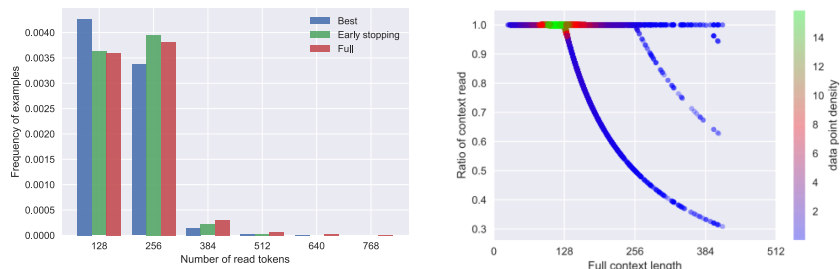


Figure 2: Performance of different versions of the Sliced DocQA with respect to different slice sizes



(a) Distribution of consumed length of the sequence. (b) Ratio of consumed context per full context length for the early stopping model.

Figure 3: Earliestopping Results (For the earliestopping model slice size is 128).

that 30% of answers in the SQuAD are single words, e.g. when the questions are of type when, where or who. We expect to find a slice size for which the performance of the model reaches the performance of the complete, not sliced, model. This is because, only local attention might be enough to correctly understand the meaning of the input sequence at each step, and also to predict whether each token is part of the answer or not. We observe that by slice size = 32, the model already reaches the performance of the uni-directional DocQA, and by slice size = 128 it almost reaches the performance of the normal DocQA. The interesting finding here is that there is a limit to which increasing the slice size leads to an increase in the performance, e.g. for the SQuAD, as soon as the slice size reaches 128 tokens, the increase in the performance is almost not noticeable.

Step transfer is useful When we have a step transfer mechanism, we observe that for each slice size, the performance of the model is higher. This means transferring knowledge between slices is useful. Among these models, the Sliced DocQA with step transfer models incremental comprehension and we see that for a slice size of 64 its performance is comparable with normal DocQA.

Global answer prediction is effective Surprisingly, with the global prediction layer, the effect of the slice size is much less. In general, when we have a global prediction layer, the performance of the model with all slice sizes is higher than when we have a sliced prediction layer. It is interesting to see that in this model, even with slice size of 1, which means using word embeddings to predict the answer span, we still get a performance close to when we have larger slice sizes.

<i>Slice size</i>	32	64	128
<i>Sliced DocQA w/o early stopping</i>	0.76	0.77	0.78
<i>Sliced DocQA w early stopping</i>	0.54 (%71)	0.69(%90)	0.77 (%99)

Table 1: Performance of Sliced DocQA with step transfer and early stopping in terms of text-f1.

Effect of early stopping In our early stopping experiments, we employed Sliced DocQA with Step Transfer as a truly incremental model. In Table 1, we compare the performance of this model with different slice sizes. As it can be seen, at slice size of 128 it almost achieves the performance of the model without early stopping (%99). Next, we investigate whether the early stopping model is more efficient regarding the number of read tokens. We looked into the performance of the model without early stopping at different context length to find the earliest point at which the model achieves its highest performance. This can be assumed as an oracle for early stopping model (i.e. best possible stopping point). In Figure 3a we compare the distribution of best stopping points with the stopping points of our early stopping model. We observe that (1)For a large number of examples, we have to read the full context to achieve the best performance. (2)For some examples, the early stopping model is reading more than it should, and for some, it stops earlier than it should. (3)In general, while with the best stopping offsets we can read about %15 less text, our model reads about %8 less. We also studied if there is a correlation between the context length and the ratio of read context length. In Figure 3b, we see for context length up to 128, we read full context, which is because our slice size is 128. After that point, we see the trend of reading smaller ratio of longer contexts.

Conclusion

In this paper, we propose a model that reads and comprehends text incrementally. As a testbed for our approach, we have chosen the question answering task. We aim to build a model that can learn

incrementally from text, where the learning goal is to answer a given question. In standard question answering, we do not care how the context is presented to the model, and for the models that achieve state of the art results, e.g. [11, 2], they process the full context before making any decisions. We show that it is possible to modify these models to be incremental while achieving similar performance. Having an incremental model, allows us to employ an early stopping strategy where the model avoids reading the rest of the text as soon as it reaches a state where it thinks it has the answer.

References

- [1] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.
- [2] Christopher Clark and Matt Gardner. Simple and effective multi-paragraph reading comprehension. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 845–855. Association for Computational Linguistics, 2018.
- [3] Gabriel Dulac-Arnold, Ludovic Denoyer, and Patrick Gallinari. Text classification: A sequential reading approach. In *European Conference on Information Retrieval*, pages 411–423. Springer, 2011.
- [4] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701, 2015.
- [5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [6] Alexander Rosenberg Johansen and Richard Socher. Learning when to skim and when to read. *arXiv preprint arXiv:1712.05483*, 2017.
- [7] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392. Association for Computational Linguistics, 2016.
- [8] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.
- [9] Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. Reasonet: Learning to stop reading in machine comprehension. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1047–1055. ACM, 2017.
- [10] Matthew J Traxler, Michael D Bybee, and Martin J Pickering. Influence of connectives on language comprehension: eye tracking evidence for incremental interpretation. *The Quarterly Journal of Experimental Psychology: Section A*, 50(3):481–497, 1997.
- [11] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le. Qanet: Combining local convolution with global self-attention for reading comprehension. *arXiv preprint arXiv:1804.09541*, 2018.
- [12] Adams Wei Yu, Hongrae Lee, and Quoc Le. Learning to skim text. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1880–1890. Association for Computational Linguistics, 2017.
- [13] Zeping Yu and Gongshen Liu. Sliced recurrent neural networks. In *COLING*, 2018.

Appendix

3.2 Architecture of Baseline Models

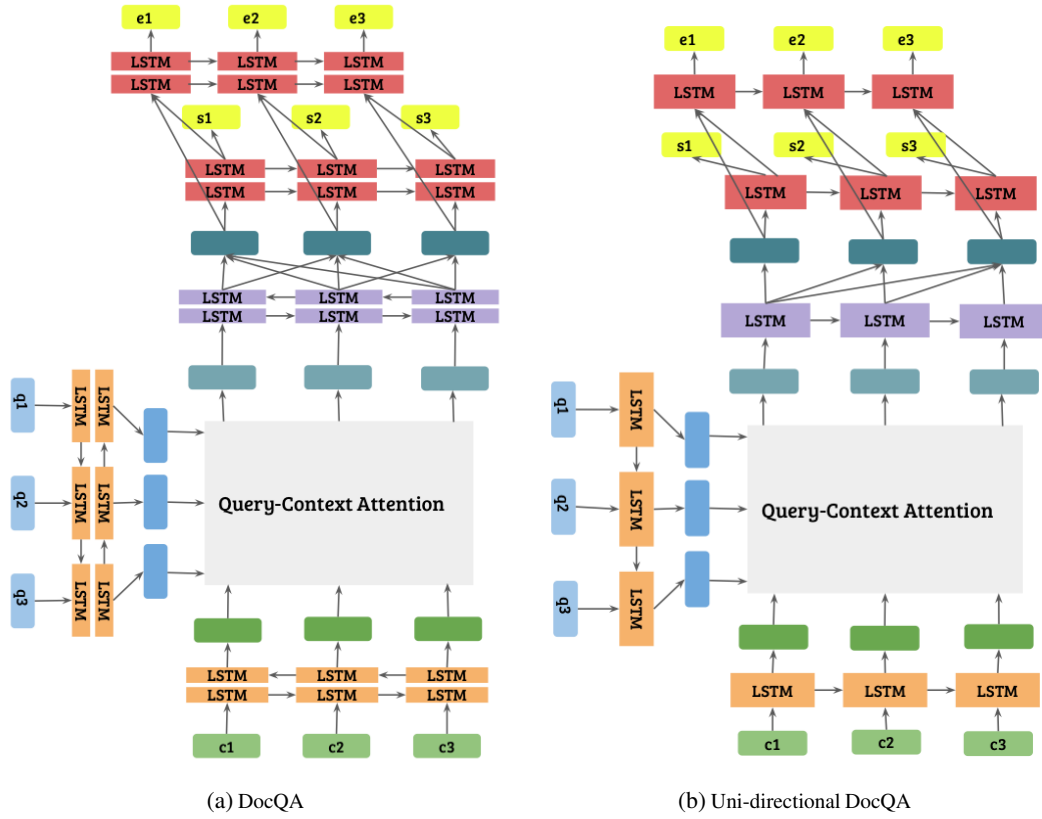


Figure 4: Architecture of Baseline Models. In these figures q_i refers to question tokens, c_i refers to context tokens, and s_i and e_i refer to probability of each token being the start and end of the answer span.