Reformed QANet - Optimizing the Spatial Complexity of QANet

Stanford CS224N Default (IID SQuAD Track) Project

Alex Fuster
Department of Computer Science
Stanford University
akfuster@stanford.edu

Andrew Hojel
Department of Computer Science
Stanford University
ahojel@stanford.edu

Dina Voevodsky
Department of Mathematics
Stanford University
dinsuhin@stanford.edu

Abstract

The feed-forward QANet architecture replaced the bidirectional LSTMs of traditional Q&A models' encoder components with convolution + self-attention to increase the speed of the model without sacrificing accuracy [1]. We achieved scores of 64.5 EM/67.9 F1 on the dev set and 61.64 EM/65.30 F1 on the test set. While the parallel nature of QANet's CNN architecture allows for a significant speed boost, it entails GPU memory requirements to reap those benefits. We preform an ablation study to measure changes to spatial complexity, speed, and performance on the QANet architecture, replacing the self attention and feed-forward layer with LSH attention, reversible residual networks, and an entire reformer. We found that implementing LSH attention successfully decreased memory usage while maintaining reasonable performance. While the other modifications did not quite maintain the original QANet model's EM and F1 scores, they significantly improved memory complexity.

1 Key Information to include

External collaborators: None, Mentor: Mandy Lu, Sharing project: False

2 Introduction

Since the release of the SQuAD1.0 dataset [2] in 2016, there has been significant progress in machine reading comprehension Q&A tasks. With the rise of pre-trained contextual embeddings (PCEs) in 2018, BERT [3] managed to even surpass human performance on the SQuAD1.0 dataset. That same year, the SQuAD2.0 dataset [4] introduced a large number of unanswerable questions to SQuAD1.0 to make the Q&A task even more difficult. Soon enough, PCE methods [5] surpassed human performance on SQuAD2.0 too. While PCE methods have taken over as state of the art on the SQuAD datasets, non-PCE methods offer more scope for creativity and opportunities for deep learning practitioners to explore different techniques and develop intuitions behind them. One of the more successful non-PCE approaches to machine comprehension is the QANet architecture.

The feed-forward QANet architecture replaced the bidirectional LSTMs of traditional Q&A models' encoder components with convolution + self-attention to increase the speed of the model without sacrificing accuracy [1]. Although QANet's parallel CNN architecture speeds up the model significantly, it also requires an abundance of GPU memory. We preform an ablation study to find optimal spatial

complexity, speed, and performance on the QANet architecture by adapting the self attention and feed-forward layers to utilize LSH attention, reversible residual networks, and the entire reformer.

3 Related Work

Our baseline QANet model is based on the architecture outlined in Yu et al. [1], who were the first to propose a recurrency-free model for the Q&A task that captures the local structure of the text as well as the global interaction between each pair of words. At a high level, the architecture consists of the standard 5 layers shared by many neural reading comprehension models: an embedding layer, an embedding encoder layer, a context-query attention layer, a model encoder layer, and an output layer. We experiment by making various changes to the structure of the encoder block, taking inspiration from ideas and techniques introduced in other works.

Specifically, we focused on drawing from aspects introduced in the reformer model outlined in Kitaev et. al's [6]. We drew from their implementation of Locally Sensitive Hashing (LSH) in the attention component, as well as their reversible architecture which we applied to our self-attention and feed-forward layers. We also experimented with using their entire encoder block and weaving it into Yu et al.'s [1] QANet model.

Kitaev et. al [6] and Yu et al. [1] were the main sources of inspiration for our work. Although all of the mathematical and architectural techniques that we implemented drew from those introduced in their work, to the best of our knowledge, no one has yet combined components from both.

We also attempt to use data augmentation methods inspired by Yu et. al [1]. There has been much work in the area of data augmentation in natural language processing. For example, Zhang et. al [7] proposed to enhance the dataset by replacing the words with their synonyms and showed its effectiveness in text classification. Raiman et. al [8] suggest using type swap to augment the SQuAD1.0 dataset, which essentially replaces the words in the original paragraph with others with the same type. Although they show an improvement in accuracy, the augmented data still has the same syntactic structure as the original data. Zhou et. al [9] improved the diversity of the SQuAD1.0 data by generating more questions. However, as reported by Wang et. al [10], their method did not help improve the performance. Yu et. al [1] were able to improve performance by paraphrasing contexts by translating the original text back and forth. This method adds more syntactical diversity to the enhanced data and seems the most promising path for data augmentation. This inspired the data augmentation methods used in this paper.

4 Approach

The baseline model we used in this project is the original BiDAF model [11] that was implemented in the starter code. Our model is built upon the QANet architecture proposed by Yu et al. [1] shown in Fig. 1. We used the following PyTorch Implementation of QANet on GitHub to get started. We soon found the implementation was not correctly modified to handle the Squad 2.0 dataset, inspite of the repository having code and clear documentation stating that Squad 2.0 was properly integrated. In fact, there was no ability for the model to predict no answer. We therefore made significant modifications to allow our implementation to handle the SQuAD 2.0 dataset, and train the correct implementation.

From there, we conducted an ablasion study by experimenting with changes to the QANet encoder block by adapting techniques from Kitaev et. al [6] to decrease spatial complexity. We created 5 versions of the original QANet model, each with a distinct combination of features from Kitaev et. al's [6] reformer model. See the Experiments section for a description of each version.

Here, we describe the techniques used by one or more versions of the modified model.

Locally Sensitive Hashing (LSH): This is a technique based on the LSH scheme in Andoni et. al [12] and the general idea of hashing attention [6]. It is used on the self-attention layer to improve memory usage and decrease spacial complexity from $O(L^2)$ to $O(LlogL)$ where L is the length of the sequence. The attention matrix for full attention is typically sparse, but the computation does not take advantage of this sparsity. In LSH attention, the queries and keys can be sorted according to their hash bucket. Since similar items fall in the same bucket with high probability, the full attention pattern can be approximated by only allowing attention within each bucket. We used this

PyTorch Reformer Implementation as a starting point for LSH attention, and embedded it into our QANet architecture. We decided to use 5 hashes, as this balanced lower computation time and high performance as demonstrated (with 4 heads) in Kitaev et. al [6]. To the best of our knowledge, we are the first to integrate LSH attention into the QANet architecture.

Reversibile Residual Network: The main idea of a reversible is to allow the activations at any given layer to be recovered from the activations at the next layer, using only the model parameters. Instead of having to checkpoint intermediate values to later use in the backward pass, we can reverse layers one-by-one as back-propagation proceeds from the output to the input. A normal residual layer performs a function $x \to y$ and has the form $y = x + F(x)$. A reversible layer performs a function: $(x_1, x_2) \to (y_1, y_2)$, and has the form:

$$y_1 = x_1 + F(x_2) \qquad y_2 = x_2 + G(y_1)$$

To reverse this, we simply do:

$$x_1 = y_1 - F(x_2) \qquad x_2 = y_2 - G(y_1)$$

We apply this idea to our attention and feed-forward layers. So, our equations become:

$$Y_1 = X_1 + \text{Attention}(X_2) \qquad Y_2 = X_2 + \text{FeedForward}(Y_1)$$

Implementing this method eliminates the need to store activations in each layer.

Chunking: This technique, aimed at decreasing spacial complexity, takes advantage of the independence of computations in feed-forward layers across positions in a sequence. In other words, the computation in the feed-forward layer can be split into "chunks" as follows:

$$Y_2 = \left[Y_2^{(1)}; \ldots ; Y_2^{(N)}\right] = \left[X_2^{(1)} + \text{FeedForward}(Y_1^{(1)}); \ldots ; X_2^{(N)} + \text{FeedForward}(Y_1^{(N)})\right]$$

Chunking is performed on feed forward layers during both forward and backward pass computation.

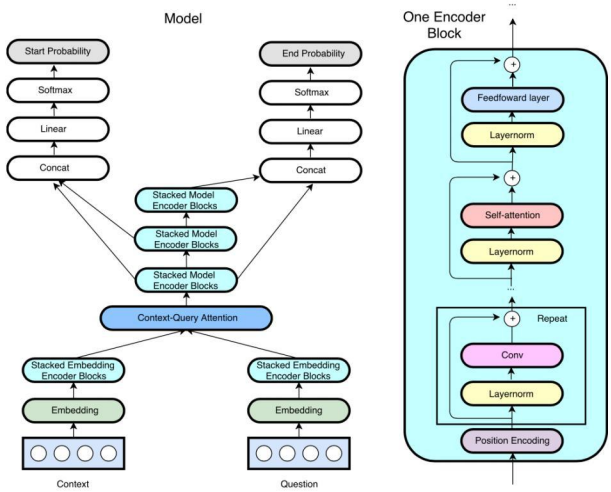


Figure 1: QANet Architecture [1]

5 Experiments

5.1 Data

We are using the official SQuAD 2.0 dataset, broken up into train (129,941 examples), dev (6078 examples), and test (5915 examples) sets.

5.1.1 Data Augmentation by Back Translation

We combine our model with a simple data augmentation technique to enrich the training data. The idea is to use the Google Translate API to produce a translation from the source language to the target language, and then apply another translation from the target language back to the source. This approach helps automatically increase the amount of training data for any language-based tasks including the reading comprehension task that we are interested in. With more data, we expect to better regularize our models.

This approach is similar to the data augmentation done in Yu et. al [1], which was the first attempt to apply back translation to enrich training data for down-stream tasks (in this case, the question answering task). It is worth noting that Dong et al [13] use paraphrasing techniques to improve question answering; however, they only paraphrase questions and did not focus on the data augmentation aspect we use in this paper.

We use data augmentation $\times$ 3 (3:1:1), which literature [1] shows 1.5/1.1 gain over the base model on EM/F1 on SQuAD 1.0. Data augmentation "$\times N$" means the data is enhanced to N times as large as the original size, while the ratio in the bracket indicates the sampling ratio among the original, English-French-English and English-German-English data during training.

Each training example of SQuAD is a triple of (d, q, a) in which document d is a multi-sentence paragraph that has the answer a. When paraphrasing, we keep the question q unchanged (to avoid accidentally changing its meaning) and generate new triples of (d', q, a') such that the new document d' has the new answer a' in it. The procedure happens in two steps: (i) *document paraphrasing* – paraphrase d into d' and (b) *answer extraction* – extract a' from d' that closely matches a.

For the document paraphrasing, a new document d' is formed using back translation via the Google Translate API. Let d be the original document that contains the original answer a and d' be its paraphrase. We identify the newly-paraphrased answer a' by finding the most similar substring in d' to a using the SequenceMatcher class from the python module difflib. We then create a list of indices where a' occurs in d' and pick the index closest to the index of the original a to handle the case in which the substring appears multiple times in the document.

5.2 Evaluation method

5.2.1 Exact Match (EM), F1, Answer v. No Answer (AvNA)

These evaluation metrics are all automatically calculated by comparing the model prediction to the gold standard answer. All three of these metrics are explained in the project description. Yet, a brief summary is that EM (Exact Match) measures the percentage of predicted answers that exactly match the gold standard ansewr. F1 score measures the mean of prediction and recall – how the content of the prediction compares to the gold standard and how much overlap in terms of exact words. Finally, AvNA measure the percent of the time that the model correctly predicts whether a question is answerable or not.

5.2.2 Negative Log-Likelihood (NLL)

Negative Log-Likelihood is used as the loss when training all of our models. The Negative Log-Likelihood measures loss by comparing the distribution of the probabilities of the start and end indices with the start and end index of the gold standard answer.

5.2.3 Speed

We are measuring speed with the help of the `tqdm` package which measures the number of training examples that were processed in a second and reports the examples / second rate. All of the speed measurements were performed on an NC6_Promo Amazon Azure VM Instance (details can be found in Table 1); therefore, the speed metric can be compared across models. We measured speed to determine how changes to the model affected training speed.

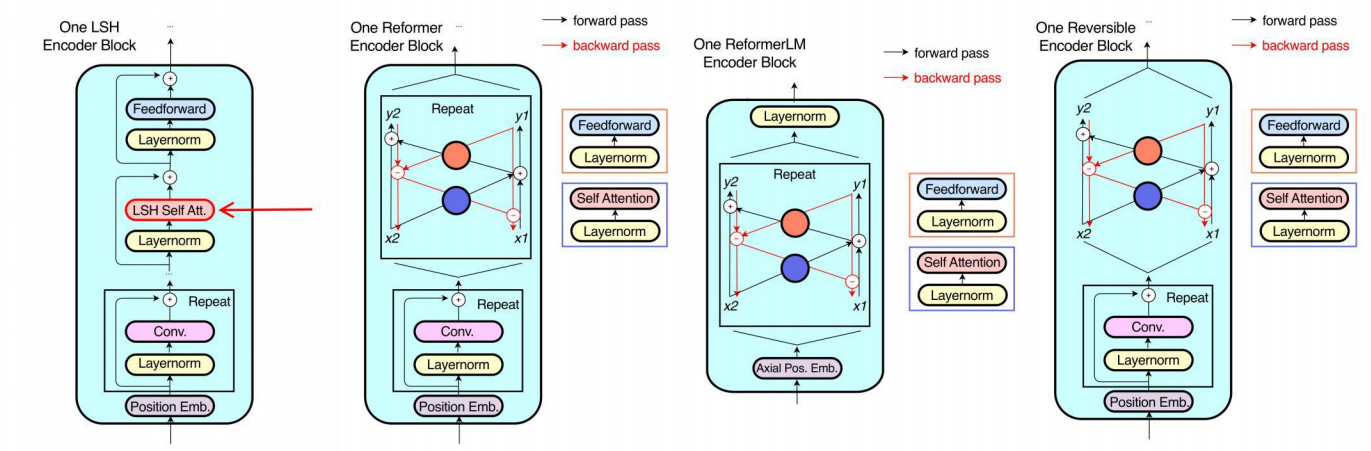
Size	vCPU	CPU	GPU	GPU Memory	Usable GPU Memory
Standard NC6_Promo	6	Intel Xeon E5-2690 v3	1 x NVIDIA Tesla K80	12 GiB	11441 MiB

Table 1: Relevant details about the NC6_Promo VM Instance that was used for testing speed and memory performance. More information can be found here.

5.2.4 Memory

Memory, which we are measuring in megabytes (MiB), measures the amount of memory that a model is using on GPU on the NC6_Promo Amazon Azure VM Instance (detaiils can be found in Table 1). This metric is being used to measure the affect of the memory optimizations implemented to QANet. We measure the memory usage by extracting the information returned from calling `nvidia-smi`. Whenever the memory is not reported, it means that the GPU ran out of memory when attempting to run the model.

5.3 Experimental details



(a) QANet-LSH (b) QANet-Reformer (c) QANet-ReformerLM (d) QANet-Reversible

Figure 2: Encoders of various QANet models with different memory optimizations

In order to find the optimal model configuration for memory optimization, we ran the following configurations of QANet (each with 96 hidden layers and 1 head). All of the reformer components were adapted from this PyTorch Reformer Implementation. Modifications were necessary to integrate components and use individual pieces of the reformer architecture while keeping all other non-changed components of the QANet architecture constant.

Small-QANet: This is the original model outlined in Yu et al. [1]

Small-QANet-LSH: This model replaced the self-attention layer in the original QANet with LSH Self Attention scheme proposed in Kitaev et. al's [6]. The rationale behind this modification is that the $O(L^2)$ spatial complexity of dot product attention significantly limits the maximum sequence length of QANet; therefore, by replacing dot product attention with LSH attention, we were hoping to improve spatial complexity for longer sequences. See Figure 2a for a diagram of the new encoder block used in this model.

Small-QANet-Reformer: This model adapted the LSH Self Attention and feed forward layers from Small-QANet-LSH into a reversible architecture using the reversible residual network design from Gomez et. al's [14]. In addition, this model made use of chunking proposed in Kitaev et. al's [6] to futher improve memory performance. To determine the number of repetitions of the reversible residual network, we used the same number of iterations of the convolutional layer. The rationale behind this modification was to completely adapt the original self-attention and feed forward layers

in the original QANet architecture with all applicable optimizations proposed in Kitaev et. al's [6]. See Figure 2b for a diagram of the encoder block used in this model.

Small-QANet-ReformerLM: This model replaced the self-attention and feed forward layers found in the original QANet using the same method used for Small-QANet-Reformer. In addition to these changes, this model removed QANet's convolution layer, replaced the position embedding with Axial Position Embedding, and added a Layernorm layer at the end of the encoder block. The rationale behind these changes was to investigate the performance of QANet without the convolutional layer and to better understand how the convolutional layer affects memory performance. See Figure 2c for a diagram of the encoder block used in this model.

Small-QANet-Reversible: This model adapted the self-attention (dot product attention) and feed forward layers from the original QANet into a reversible architecture using the reversible residual network design. The rational behind this modification was to test the impact of the reversible architecture on model performance and memory performance. This optimization is appealing because it is not dependent on sequence length; instead, the improvements in the memory performance come from the model's architecture. See Figure 2d for a diagram of the encoder block used in this model.

In addition to testing various model configuration with memory optimization, we ran the following experiments with different model sizes and data sets created using data augmentation by back translation to measure the effects on model performance:

Large-QANet: This model used a hidden dimension size of 128 and had 8 attention heads. The goal was to determine whether a larger model could achieve better performance than Small-QANet.

Small-QANet-Augmented: This is the same model as Small-QANet, yet it was trained on the augmented data set, which is three times the size of the original data set. Yu et. al [1] found that this data augmentation could improve performance on SQuAD1.0; we wanted to test whether these improvements hold with SQuAD2.0.

Large-QANet-Augment: This model is the same model as Large-QANet, yet it was trained on the augmented data set. The rationale for this model was to determine whether data augmentation had a different affect on a larger model.

5.4 Results

The performance metrics on SQuAD2.0 and hyperparameters of the models evaluated throughout this research can be found in Table 2. The speed and spatial performance of the models can be found in Table 3. Our top performing model was Small-QANet, which achieved 64.48 / 67.93 EM and F1 respectively on development set and 61.64 / 65.30 EM and F1 respectively on the test set.

In terms of model performance on SQuAD2.0, we were surprised to see that Large-QANet performed worse than Small-QANet. In Figure 4, we can see that the NLL for Large-QANet increased significantly more in later epochs, which is an indicator of overfitting. We were also surprised to find Small-QANet-Augmented performed worse in spite of having more data. In Figure 5, it is clear that this model also suffered significant overfitting. The performance of Large-QANet-Augmented was an improvement over Large-QANet (improvement of +0.25 EM and +0.58 F1), yet its performance still did not reach Small-QANet's.

Model	Hidden	Heads	Dev EM	Dev F1	Dev AvNa	Dev NLL	Test EM	Test F1
Small-QANet	96	1	64.48	67.93	74.14	2.74	61.64	65.30
Small-QANet-LSH	96	1	64.16	67.79	74.04	2.70	-	-
Small-QANet-Reformer	96	1	62.69	66.40	73.06	3.13	-	-
Small-QANet-ReformerLM	96	1	51.85	51.84	52.11	5.21	-	-
Small-QANet-Reversible	96	1	62.70	66.41	72.91	2.98	-	-
Small-QANet-Augmented	96	1	62.11	65.83	72.37	3.24	-	-
Large-QANet	128	8	63.15	66.70	73.32	3.08	-	-
Large-QANet-Augmented	128	8	63.40	67.28	73.72	3.33	61.17	65.15

Table 2: Dev and test set results from variations of QANet all trained to 30 epochs with a maximum context sequence length of 400 words.

Model	Speed (400 seq)	400 seq	1024 seq	2048 seq	4096 seq	8192 seq
Small-QANet	19 ex/sec	1089 MiB	2539 MiB	6891 MiB	-	-
Small-QANet-LSH	11 ex/sec	1703 MiB	2959 MiB	5355 MiB	10065 MiB	-
Small-QANet-Reformer	2 ex/sec	899 MiB	1285 MiB	1979 MiB	3368 MiB	7167 MiB
Small-QANet-ReformerLM	5 ex/sec	841 MiB	1115 MiB	1609 MiB	2529 MiB	5449 MiB
Small-QANet-Reversible	17 ex/sec	808 MiB	1355 MiB	2413 MiB	4634 MiB	10276 MiB
Large-QANet	13 ex/sc	2551 MiB	11041 MiB	-	-	-

Table 3: Memory and speed of all variations of QANet trained with a batch size of 4 on a NC6_Promo VM instance (details about this VM can be found in Table 1). A "-" signifies that CUDA ran out of memory. Please refer to Table 2 for hidden dimension and number of heads of each model. Please refer to the speed and memory sections for details about the metrics and how they were collected.

5.5 Spatial Complexity Ablation Study

In this ablation study, we will investigate the effect of each memory optimization on the speed, memory performance, and SQuAD2.0 performance. The performance of all the models on SQuAD2.0 can be found in Table 2 and the memory / speed performance can be found in Table 3.

Small-QANet: This model was used as our baseline for memory performance, performance on SQuAD2.0, and speed. It is not surprising that it achieved the best performance on SQuAD2.0 because none of our memory optimizations were targeted at improving this performance. In terms of speed, we see that this is the fastest model. In terms of memory, at a sequence length of 400 words this model is middle-of-the-pack, yet as the sequence length scales up it is quickly outperformed by other models. Specifically, when the sequence length is 4096 words, CUDA ran out of memory (whereas other models we implemented were able to train on much longer sequences).

Small-QANet-LSH: The results for this model were interesting because it had the lowest difference in performance from Small-QANet (performing only marginally worse on both EM and F1 scores). The runtime increases by 40%, which is expected because LSH Attention is a costly operation, yet this could be addressed by changing the number of hashes used, which would speed up the model but potentially decrease performance. In terms of memory, Small-QANet-LSH performs worse than Small-QANet on sequences of 400 or 1024 words, but performs better on 2048 words and is able to run with a sequence length of 4096 words. Therefore, it is clear that this is a good option for improving spatial complexity on longer sequences with small affects on performance (yet with significantly slower speeds).

Small-QANet-Reformer: The results for this model were interesting because it achieved the best memory performance while achieving acceptable performance on SQuAD2.0 (yet there is a non-negligible decrease of -1.79 EM and -1.53 F1 scores). The downside of this variation is that it is 90% slower than Small-QANet. We believe this is caused by the repetition of LSH Self Attention; therfore, this could be addressed by decreased the number of reversible residual networks in the reformer and/or changing to dot product attention. For a longer sequence length, it would not be possible to use dot product attention because of its spatial complexity. If the user is optimizing for memory performance with reasonable results, this is the best model.

Small-QANet-ReformerLM: This model achieved unacceptable EM/F1 scores. We wanted to test whether removing the convolutions had a significant affect on model performance, which we found was the case. The decreased memory usage of this model in comparison to Small-QANet-Reformer could be caused either by the removal of the convolutions or by the switching of the absolute position embedding layer with an axial position embedding layer.

Small-QANet-Reversible: This model achieved performance almost identical to Small-QANet-Reformer. The primary difference is that it is significantly faster (with only a 10% speed decrease from Small-QANet). This affirms the point that Small-QANet-Reformer is likely so slow because of LSH attention. Another significant difference from Small-QANet-Reformer is as the sequence length increases, it uses more memory than Small-QANet-Reformer (on a sequence length of 8192 words it uses 40% more memory). Therefore, on shorter-length sequences this model achieves high-speed, good memory efficiency, and relatively good EM/F1 Scores.

The results of our ablation study demonstrate that if QANet is being run on larger sequences, the best option for maintaining the best performance on SQuAD and decreasing memory complexity is Small-QANet-LSH. The most optimal model for memory performance on short sequences is Small-QANet-Reversible, yet it does suffer a decrease in EM/F1 scores. For long sequences the most optimal model for memory performance is Small-QANet-Reformer, yet it suffers from a decrease in EM/F1 scores (almost identical to Small-QANet-Reversible) and a significant decrease in speed, which could be addressed by decreasing the number of reversible residual layer repetitions.

5.6 Data Augmentation

In a few cases, the answer a was lost in translation. See Table 4 for an example. This occurred for 49 out of 389,823 training examples ($\sim 0.0125\%$). Since this is such a rare occurrence, we simply drop the training examples where this is the case.

	Sentence that contains an answer	Answer
Original	'Developing two versions would mean delaying the previously announced 2005 release, still disappointing the consumer.'	'2005'
Paraphrase	'Develop two versions would mean delaying the previously announced publication, still disappointing the consumer.'	''

Table 4: Example of back translation completely losing answer.

The quality and diversity of paraphrases are essential to the data augmentation method. It is still possible to improve the quality and diversity of this method. The quality can be improved by using better translation models. For example, we find that the translations often lose proper nouns that appear in answers like book titles, song titles, names, and more (see Table 5 for an example). It is often the case that such proper nouns are the answers to some of the questions, and thus by losing them in translation we lose the ability to correctly answer questions. In addition, we can combine this method with other data augmentation methods, such as the type swap method (Raiman et. al [8]), to acquire more diversity in paraphrases.

	Sentence that contains a proper noun in the form of song title	Song title
Original	Following the disbandment of Destiny's Child in June 2005, she released her second solo album, B'Day (2006), which contained hits "Déjà Vu", "Irreplaceable", and "Beautiful Liar".	"Déjà Vu"
Paraphrase	Following the dissolution of the child of destiny in June 2005, she published her second solo album, B'Day (2006), which contained hits "already seen", "irreplaceable" and "beautiful liar".	"already seen"

Table 5: Example of back translation losing correct proper noun.

6 Conclusion

Overall, we were able to decrease memory complexity while maintaining performance comparable to the Small-QANet with Small-QANet-LSH. Other major changes to the QANet architecture lead to a slight performance decrease, but they achieved a major decrease in memory usage as we hoped. Additionally, we find that we are able to achieve slight gains on Large-QANet by utilizing data augmentation consisting of translating the context to and from another language as a way of paraphrasing.

We also believe that we can even further improve memory optimization in the QANet model with successful EM/F1 scores beyond applying optimizations introduced for the reformer model. In the future, we hope to explore novel optimization techniques specifically derived for the QANet model, and continue to challenge the lower limit for the model's necessary memory consumption. For example, it would be interesting to attempt to decrease the memory usage of the convolution layers.

References

[1] 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, 2018.

[2] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: pre-training of deep bidirectional transformers for language understanding, 2018.

[4] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad, 2018.

[5] Zhuosheng Zhang, Junjie Yang, and Hai Zhao. Retrospective reader for machine reading comprehension, 2020.

[6] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations (ICLR)*, 2020.

[7] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 649–657, 2015.

[8] Jonathan Raiman and John Miller. Globally normalized reader. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 1070–1080, 2017.

[9] Qingyu Zhou, Nan Yang, Furu Wei, Chuanqi Tan, Hangbo Bao, and Ming Zhou. Neural question generation from text: A preliminary study., 2017.

[10] Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, , and Ming Zhou. Gated self-matching networks for reading comprehension and question answering. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, page 189–198, 2017.

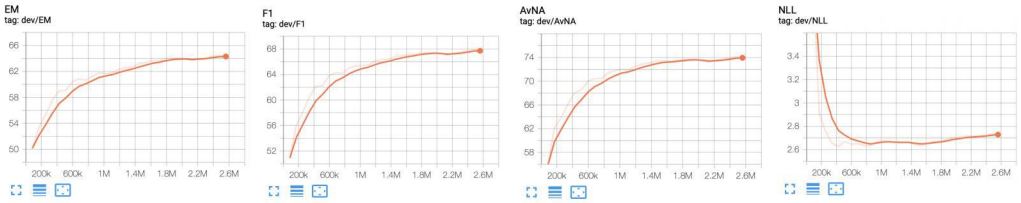
[11] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension, 2016.

[12] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya P. Razenshteyn, and Ludwig Schmidt. Practical and optimal lsh for angular distance, 2015.

[13] Li Dong, Johnathan Mallison, Siva Reddy, and Mirella Lapata. Learning to paraphrase for question answering. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics:875–886, 2017. URL http://aclweb.org/anthology/D17-1091.

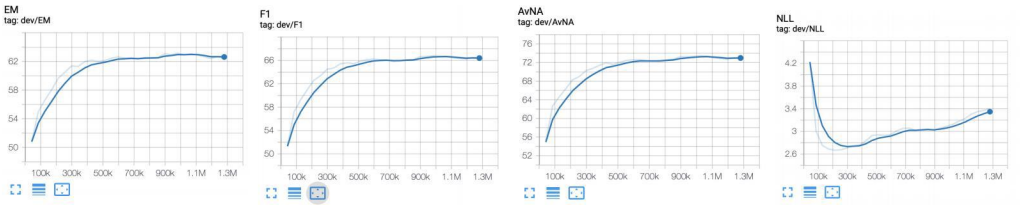
[14] Aidan N. Gomez, Mengye Ren, Raquel Urtasun, and Roger B. Grosse. The reversible residual network: Backpropagation without storing activations, 2017.

A Appendix



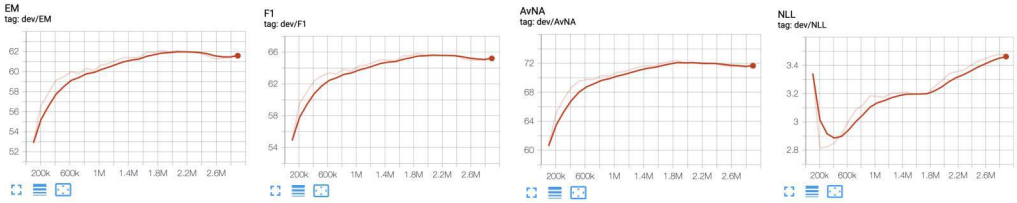
(a) EM (b) F1 (c) AvNA (d) NLL

Figure 3: Small-QANet Tensorboards



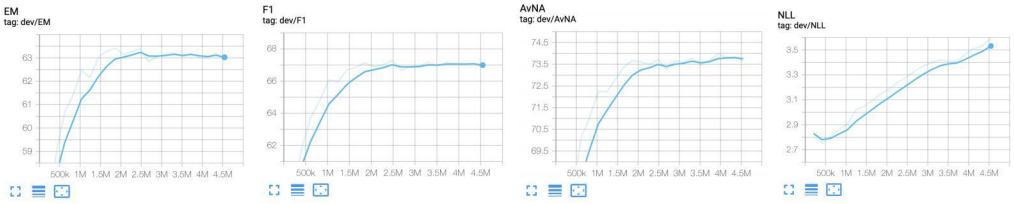
(a) EM (b) F1 (c) AvNA (d) NLL

Figure 4: Large-QANet Tensorboards



(a) EM (b) F1 (c) AvNA (d) NLL

Figure 5: Small-QANet-Augmented Tensorboards



(a) EM (b) F1 (c) AvNA (d) NLL

Figure 6: Large-QANet-Augmented Tensorboards