A Comparison of RNN and Transformer-based Question Answering Systems

Diana Moncoqut
Department of Computer Science
Stanford University
ines1703@stanford.edu

Adam Jensen
Department of Computer Science
Stanford University
oojensen@stanford.edu

Abstract

In this paper, we compare the merits of a question answering system based on RNNs and bidirectional attention flow (Minjoon Seo, et al) with a system based on Transformer encoder blocks (Vaswani et al.). In our experiments, we find that the RNN based system is strictly superior to every system we built with Transformer blocks. In short, though Adams Wei Yu achieved state of the art results with transformer blocks, we could never get a transformer-based system to significantly reduce the loss function. No matter whether we tried a simple architecture, or a more complicated one, based on either our own implementation of transformer blocks, or Vaswani's tensor2tensor library, all of them exhibited similar training characteristics and failed to train. We hope this paper is useful as a cautionary tale and example model/training strategy graveyard for those interested in applying Transformer networks to question answering in future classes.

1 Introduction

This section introduces the problem, and our overall approach to the problem.

1.1 The Task

'Question Answering' (QA) applied to the Stanford Question Answering Dataset (SQuAD) is a competitive task in academic machine learning. It also has real-world applications to domains that require reading comprehension (such as customer support). QA is a benchmark for how well models can mimic reading comprehension or perform 'machine comprehension' (MC). More concretely, the task is: given a question, and a paragraph that contains the answer to that question: highlight the answer to the question. The input is a `question` (q) sentence and `context` paragraph (c). The output is the `start index` where the answer to the question begins in the context, and the `end index`. In order to do well at this task, a system must model complex interactions between the tokens in the question and the context.

1.2 The Importance of Attention

In order to model complex, long-range interactions between the query and context tokens, an attention component is a key part of these systems. The attention component is responsible for complex skills such as coreference/anaphora resolution, commonsense reasoning, causal relations, and spatiotemporal relations [5]. As summarized by Seo et all [3], "the computed attention weights are often used to extract the most relevant information from the context for answering the question by summarizing the context."

1 CS224n (2018), Stanford, California.

2 Background and Related Work

The SQuAD leader board is very competitive, and there is a lot of work in this area. Virtually all top models use some form of attention mechanism according to BNU [5]. This section provides background information on the two models that we were inspired by in this paper.

Seo et all proposed "Bidirectional Attention Flow" [3] to solve the information losses resulting from the unidirectional nature of attentions, and from premature summarization into fixed-length vectors. Adams Wei Yu proposed another way to achieve bidirectional information flow and prevent early summarization that involves no RNN components (in order to facilitate more parallelism and train either in less wall clock time or with more data).

In this paper, we compare a baseline system, a system inspired by Seo et all, and a system inspired by Adams Wei Yu [1]. The results that Adams Wei Yu achieved with his fast model and a clever data augmentation scheme are the state of the art as of early 2018. However, we achieve much better results with the system inspired by Seo et al. We were not able to reproduce the results of Adams Wei Yu despite following his architecture description and training regimen carefully.

2.1 BiDAF Background

The Bidirectional Attention Flow model is a hierarchical network with a layers for: character embedding, a word embedding, a contextual embedding, attention flow, recurrent modelling, and an output layer. The layer that is most of interest is the attention flow layer.

The inputs to this layer are the vector representations of the context (C) of shape `(batch_size, context_length, hidden_dim)` and the query `(batch_size, question_length, hidden_dim)`. The layer is responsible for outputting query-aware representations of the context words A_C. The layer also outputs the embeddings computed by previous layers.

The attention flow layer computes attention in two directions, from context to query as well as from query to context, in two different steps.

2.2 Transformer Background

Based on the observation that GRUs and LSTMs still require attention models to capture long-range interactions between tokens, Vaswani et al. proposed sequence encoder-decoder architecture in Attention is All You Need. For QA, the encoder and the multi-head attention mechanism are most of interest because the 'decoding' step is done by a softmax layer that points to indices in the context to create the answer. We do not use the decoder from Vaswani et al for QA and niether does Adam Wei Yu.

The Transformer encoder consists of: positional encoding, and then `n` repeating blocks of:

- $Out_1 = \text{MultiheadAttention}(I_{prev})$
- $Out_2 = \text{LayerNorm}(Out_1 + I)$
- $Out_3 = \text{FeedForward}(Out_2)$
- $I_{next} = \text{LayerNorm}(Out_3 + Out_2)$

Where LayerNorm is described in [6] and $\text{MultiheadAttention}$ is attention applied to `h` 'heads' where each head is a down-projection of the input to a lower-dimensional space.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

$$\text{MultiheadAttention}(Q, K, V) = \text{concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

$$where \mid \text{head}_\text{i} = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

For the encoder, $Q = K = V$ and all of the down projections are to d_k dimensions. Vaswani et al typically used $\text{h} = 8$.

Repeating these Multihead self-attention blocks allows the model to learn complex interactions between words in the inputs. For our purposes, this means that it can encode a complex representation of the question and the context.

2.3 Background on Transformers Applied to QA

Adam Wei Yu used 1 transformer block to encode the context, and a different one to encode the question. He then fed these encodings into an attention layer (that did not use transformer concepts) and applied another 3-layer transformer encoder to the outputs of that self attention layer. In order to produce the output, he fed the output of the first and second layers to a softmax layer to produce the `start_index` and the 1st and 3rd layer to another softmax layer to produce the `end_index`.

Importantly, Adam Wei Yu used a learning rate of 0.001 and an exponential learning-rate warmup plan from 0 to 0.001 over 1000 iterations.

2.4 Approach

Figure 1 shows the abstract plan that these models all followed, and the details of different implementations are below. The embedding layer is omitted from the figure because it is common to all models.

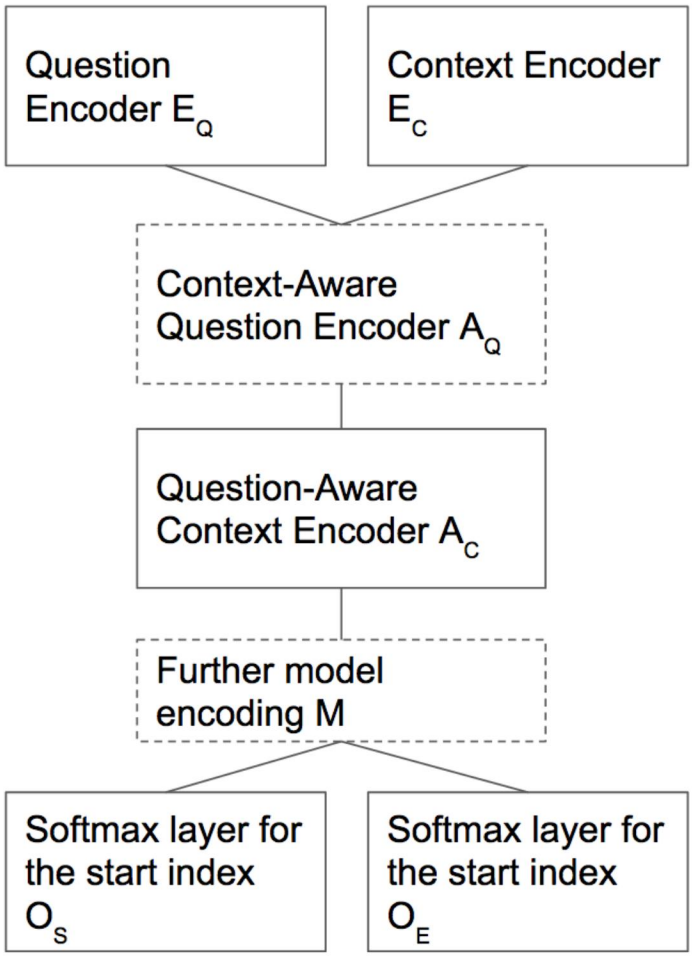

Figure 1: Abstract model representation. Optional parts have outlines

2.5 Baseline Model

The baseline model is well described in the assignment handout. It consists of an embedding layer using 100 dimensional GloVE vectors, a shared 100 dimensional GRU encoder for the query and the context (E_Q and E_C are produced by the same weights), a basic attention layer that produces a question-aware context encoding A_C using dot product attention. Then it feedsA_C and E_C to two softmax layers that produce the `start_index` or O_S and `end_index` or O_E.

2.6 Bidaf approach

For the Bidirectional Attention Flow model, we did a straightforward implementation of the model described in the paper. We used Glove vectors with 100 units, 200 hidden units in the encoder RNN, 600 amount of weights in the attention layer, and 400 hidden units in the Model layer above the attention layer.

2.7 Transformer network approach and experiments

We tried a large number of different transformer-based networks. With none of the following were able to exceed 20% F1 performance. The details of each model is included together with the parameters of the failed training experiment for legibility and to avoid sprawl.

Training these models all resulted in the same behavior. At first, there would be healthy gradients (i.e. above 1 and below ten). Then over time, the parameter norms would increase (doubling or tripling) and the gradients would become small while the loss was still high (often around 9.6), and the model would fail to significantly reduce the cost function with each iteration.

We performed an ablative analysis working backwards to our baseline model, but were not able to train any of the transformer-based networks including one that only differed from the baseline in 1 place: the question and context encoder. We also removed our implementation of the transformer blocks as a possible culprit by integrating google's tensor2tensor transformer library. The canonical implementation exhibited identical training behavior. That is, we couldn't get the loss function to go down much using perfect module code, either. That means that the issue must have something to due with the way we attempted to train the transformer networks, but we couldn't figure it out in the time we had.

2.7.1 Multiheaded question-aware context encoder, with Adams We Yu's output layer

This architecture is the closest one that we tried to Adams Wei Yu's architecture. The question and context encoder are transformer self-attention blocks. Multihead attention is used to create a weighted sum of the questions based on the context (i.e. a context-aware question encoding). W used the context-aware question encoding to create a question-aware context encoding. Then we passed the question-aware context encoding to a 3 layer self-attention layer and passed the context and the different layers of the question-aware context encoding to two different softmax layers for the output.

We tried $\mathtt{n} = 2$ blocks for the transformer sections, $\mathtt{h} = 8$ heads with a hidden size of 128 and $\mathtt{h} = 10$ heads with a hidden size of 100.

For this model, we used $\mathtt{h} = 10$ heads, a learning rate of 0.001 and a piece-wise exponential warm-up scheme from 0.000001 to 0.001 over 1000 iterations. This is very similar to Adams Wei Yu's model and training scheme, but it did not work.

2.7.2 Multihead question-aware context encoder

This model is exactly the same as the Baseline Model, except we replaced the `Question Encoder` and `Context Encoder` with transformer self-attention blocks, and we replaced the basic dot-product attention with transformer blocks. As against the last model, we ablated away the custom output layer, and the context-aware question encoding.

For this model, we used $\mathtt{h} = 10$ heads, a learning rate of 0.001 and a piece-wise exponential warm-up scheme from 0.000001 to 0.001 over 1000 iterations. This is very similar to Adams Wei Yu's model and training scheme, but it did not work.

2.7.3 Simplest possible transformer-based model

This model is exactly the same as the Baseline Model, except we replaced the `Question Encoder` and `Context Encoder` with transformer self-attention blocks. For this configuration, we used `h` $= 10$ heads because the dimensionality of the embeddings is 100. We used `n = 2` transformer blocks, a learning rate of 0.2 (which is the default in tensor2tensor), and linear learning rate warmup plan from 0 to 0.2 over 6400 iterations (ten epochs). Comparing this to the exponential learning-rate warmup plan from 0 to 0.001 over 1000 iterations that Adams Wei Yu used, and we can see that the schedule we used is slightly more aggressive. At iteration 1000, our learning rate is 0.03 whereas his has just reached 0.001. This may be the cause of the issues we experienced training this model but experiments with learning rates of 0.001 and architectures closer to Adams Wei Yu also did not result in significant reductions of the cost function. At this point we ran out of time to run experiments.

2.8 BiDaf Experiments

In this section, we describe:

- the dataset we used
- how we ran our experiments (e.g. model configurations, learning rate, training time, etc.)
- The evaluation metric(s) we used
- our baselines for the evaluation metrics
- our quantitative results for the evaluation metrics
- a qualitative evaluation (example results, etc.).

2.8.1 SQuAD dataset

SQuAD consists of 100,000+ question, answer pairs on taken from 500+ wikipedia articles. The questions are almost all shorter than 30 tokens, and the contexts are nearly all shorter than 400 tokens. We trained on 80% of these tokens and had 10% as a development set. The other 10% is used by the course staff as a testing set.

2.8.2 How we ran our Bidaf experiments

In our implementation, we swapped the basic attention layer with the attention flow layer. The inputs to this layer are vector representations of the Query and the Contexts. We used the Similarity Matrix between the Contexts and the Query as described in the original paper. The implementation made use of the tensorflow "tile" which resulted in fairly large 4-dimensional matrices. This led to out of memory issues during training. This issue was mitigated by adjusting the model hyperparameters such as the batch size and the hidden size. We also tried segmenting the trainable weight vector W into 3 components, but we did not experience real improvements with this method. The parameters used in our training were as follows: learning rate: 0.001, batch size: 30, hidden size: 100 and context size: 300.

2.8.3 The metric(s) we used

In order to train our models, we optimized the sum of the usual cross-entropy loss function for the start index and the end index.

$$CE(y, \hat{y}) = -\sum_{i=1}^{\mathrm{N_c}} y_i \log(\hat{y}_i)$$

$$Loss(y, \hat{y}) = CE_{start\_index}(y_{start}, \hat{y_{end}}) + CE_{end\_index}(y_{end}, \hat{y_{end}})$$

To evaluate our model, we used the average F1 score and the average Exact Match score. For a single example, the exact match is: do all the words in the prediction exactly match the gold answer? This is binary for each prediction (either it is an exact match or it isn't). Note that the exact match score is not computed based on the indices, in to account for the fact that the answer text (e.g. Golden State Warriors) might appear in different places in a paragraph.

The F1 score for a single example is a combination of precision and recall applied to the common words in the question and the answer, as against the length of the question and the answer. The code is:

2.8.4 Quantitative results

We stopped training the baseline model after 15,000 iterations. Our best registered dev loss is 4.71 and our dev F1 and EM are at 39.4 and 28.2, respectively. With the BiDAF model our F1 score improved by about 2%. We have not yet submitted to Codalab due to the issues everyone is experiencing, as documented on Piazza.

2.8.5 Qualitative Results

The bi-directional attention model showed some improvement over the classical attention model. The model is easy to easy to understand and implement. However, improvements remain modest compared to the increase in the size of the model. I believe that results of the BiDAF implementation could have been further improved by adding 1) the character embedding layer that is used to better handle out-of-vocab or rare words and 2) the additional modeling layer used in the original paper between the attention flow layer and the output layer. In addition, combining BiDAF with other attention schemes such as co-attention, self-attention or multi-hop attention could have led to better results.

3 Conclusion and Discussion

Diana and I found that our RNN-based system was much easier to train than the transformer-based system that was intended for the FAST SQuAD task. Since the results were not competitive, it does not make sense to comment on the run-time of the transformer system. Quickly getting wrong answers is nothing to talk about.

As discussed, Adam Jensen did not succeed in training a transformer-block based system. In terms of the model, the only differences between his models and the ones that Adams Wei Yu et al. described were simplifications intended to improve the feasibility of training the network. We know that a bug our implementation of transformer network blocks is not the cause of the issue, because the canonical implementation of a transformer block runs into the same issues at train time. I suspect that the learning rate schedule might have been the cause of the issue, even though the difference between the warm-up schedules that I tried and and the ones that Adams Wei Yu et al described are small.

I still think the possibilities of applying non-recurrent architectures to natural language processing are exciting. For future work, I would want to find a mentor who has trained transformer networks before. I would also want to perform more experiments with the learning rate schedules defined in the tensor2tensor `learning_rate` module, to exactly replicate the exponential learning rate warmup that Adams Wei Yu used. If I could do it all over again I would first implement an architecture with tensor2tensor until it was reliably training, and then produce my own implementation of the transformer network. Unfortunately, I wasn't fully aware how hard they are to train or how few of the teaching assistants have experience training them. I started my own implementation first (which is a significant programming task in itself).

Once a transformer network implementation and reliable training scheme were established, it would be intriguing to tweak the transformer model in two ways: 1) to see whether a leaky-relu activation function improves training time or the final result, and 2) to experiment with more skip connections. The classic transformer network has a residual connection between the layers of each transformer block [2] but it does not have residual/skip connections from one block to the next, or that skip entire blocks. Perhaps increasing the skip connections would make these networks easier to train by facilitating the backwardpropagation of the gradients.

Acknowledgments

We acknowledge Microsoft for providing Azure credits. We also acknowledge Nick at the Apple store in Palo Alto, who fixed Adam's Laptop when he messed it up during the middle of the project.

References

References to all literature that informed our project work.

[1] Adams Wei Yu & Dohan, D. & Luong, M.T. (2018) FAST AND ACCURATE READING COMPREHENSION BY COMBINING SELF-ATTENTION AND CONVOLUTION. *ICLR* https://openreview.net/forum?id=B14TlG-RW

[2] Vaswani, A. et al. (2017) ATTENTION IS ALL YOU NEED *31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA* https://arxiv.org/pdf/1706.03762.pdf

[3] Seo, M. et al. (2017) BI-DIRECTIONAL ATTENTION FLOW FOR MACHINE COMPREHENSION *ICLR* https://arxiv.org/pdf/1611.01603.pdf

[4] Vaswani, A. et al. (2018) Tensor2Tensor: a library of deep learning models *https://github.com/tensorflow/tensor2tensor*

[5] FNU Budianto (2017) Reading Comprehension on the SQuAD Dataset http://web.stanford.edu/class/cs224n/reports/2762006.pdf

[6] Jimmy Lei Ba et al. (2016) LAYER NORMALIZATION https://arxiv.org/abs/1607.06450