Improved QANet on SQuAD 2.0

Stanford CS224N Default Project

Nicole Lee & Matthew Reed
Department of Computer Science
Stanford University
nicole01@stanford.edu & mattreed@stanford.edu

Abstract

The task of question answering is trivial for humans, but it has long been a known problem to solve for NLP models. Neural networks like Bidirectional Attention Flow for Machine Comprehension (BiDAF) and Bidirectional Encoder Representations from Transformers (BERT) have been dominating the leaderboard when tested on the Stanford Question Answering Dataset (SQuAD) 2.0, differentiated from SQuAD 1.0 due to a larger corpus of unanswerable questions. We invalidated our initial hypothesis that an extended BiDAF model with a Transformer encoder would improve upon baseline BiDAF results. However, we propose that with certain alterations, QANet can perform better than the baseline BiDAF model when evaluated on SQuAD 2.0. Through many iterations and experiments, we find that an improved QANet with data augmentation and learned positional encodings outperforms the BiDAF baseline.

1 Key Information to include

- Mentor: Grace Lam
- External Collaborators (if you have any): N/A
- Sharing project: N/A

2 Introduction

Question answering has been a problem that many NLP models have tried to solve in the past years. Models are given a question and corresponding paragraph as input, then expected to extract the correct span (ie. segment of text) from the passage that answers the question.

Initially, this question answering task was solved for niche categories like Major League Baseball through BASEBALL and geological analysis of rocks returned by the Apollo moon missions through LUNAR. These models used similar architecture to that of ELIZA and performed extremely well in their particular domains. Later, models like LSTM, BiDAF, and QANet emerged to perform well on a more broad SQuAD 1.0 dataset [1]. Soon, the BERT model outperformed both LSTM and BiDAF. With the introduction of more unanswerable questions within the new SQuAD 2.0 dataset, these models began to show worsened performance.

While some models have been tested on the SQuAD 2.0 dataset, the performance of QANet has not yet been thoroughly researched. We propose that the introduction of the QANet model will improve performance for the SQuAD 2.0 dataset. Specifically, we implemented a QANet model from scratch and introduced additional data augmentation and learned positional encodings.

3 Related Work

The baseline implementation is a provided Bidirectional Attention Flow network [2]. This hierarchical multi-stage architecture models the representations of given context paragraphs at different levels

of granularity. The most significant part of the BiDAF model is its bidirectional attention flow that allows for query-aware context representation. Specifically, its attention layer is computed at every time step and flows to the following model layer, reducing information loss originally caused by summarization. BiDAF also simplifies the attention layer by using a memory-less mechanism that does not directly depend on the attention from the previous time step.

Our base QANet architecture is mainly adopted from the question answering task implementation of Yu et al. [3] Without the use of recurrent networks, QANet is faster than the sequential framework of RNNs. In fact, Yu shows that the QANet model is 3x-13x faster in training and 4x-9x faster in inference while still maintaining equal accuracy scores compared to recurrent models.

4 Approach

Originally, we hypothesized that using a Transformer architecture within the BiDAF model would improve performance. We specifically replaced the encoder layer with a Transformer instead of LSTM and introduced character-level embeddings. Because character-level embeddings better handle out-of-vocabulary words and can condition on the internal structure of words, we hypothesized that this would improve the model. However, our results showed that this did not improve upon the baseline scores.

We then pivoted to a different approach: implementing a QANet model from scratch. The QANet model has yet to be thoroughly trained and tested on SQuAD 2.0, so we adopted the approach of Yu et al. [3] as outlined in Figure 1.

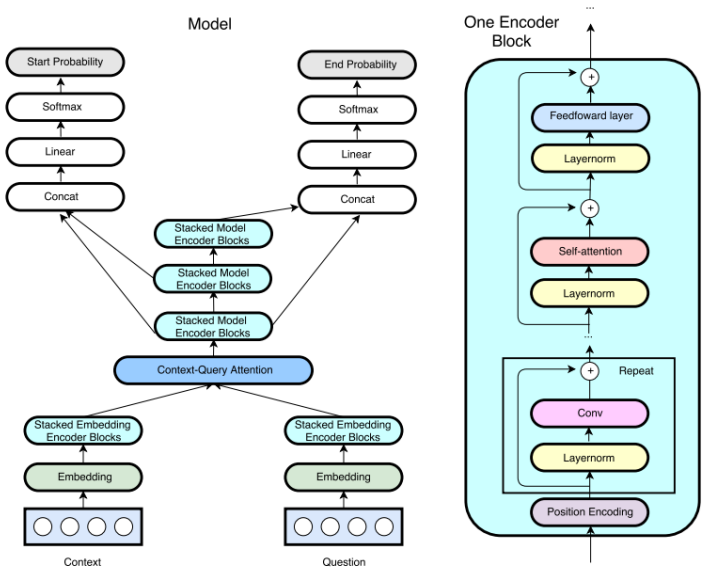

Figure 1. QANet model architecture [3]

QANet is composed in a similar fashion to other existing models. It has five main components: an embedding layer, an embedding encoder layer, a context-query attention layer, a model encoder layer, and an output layer. However, instead of utilizing RNNs within the embedding and modeling encoder layers, we use convolutional and self-attention mechanisms. This combination greatly increases the speed of experimentation.

One of our main alterations to the default QANet model is the introduction of data augmentation and learned positional encodings. Data augmentation in the form of "word-dropout" can help with generalization [4] while learned parameters for positional encodings can allow the model to learn better.

5 Experiments

5.1 Data

We are using the Stanford Question Answering Dataset (SQuAD), specifically SQuAD 2.0 which unlike SQuAD 1.0, has a large corpus of unanswerable questions [1]. SQuAD 2.0 contains more than 100,000 questions for machine comprehension. A group of crowdworkers generated questions on a set of Wikipedia articles in which the answer to a given question is a span (ie. segment of text) from the corresponding passage.

Using this data, we send our model a question and corresponding paragraph of text as input. The goal is for the model to output a prediction for the span of the passage that correctly answers the given question.

5.2 Evaluation method

Given a question, the task is for the model to correctly predict the span of text from the corresponding passage that answers the question. We evaluate the accuracy of the model's predictions using three different scores: F1, EM, and NLL. The F1 score measures the harmonic mean of precision and recall while the EM metric is a binary evaluation and NLL communicates the magnitude of incorrect predictions. The combination of all three of these metrics is necessary in evaluating the model's performance. Specifically, the F1 score is much less harsh in its punishment for answer variations as compared to the need for precision in EM computations.

5.3 Experimental details and analysis

Early on in our experimental process, we hypothesized that replacing the bidirectional LSTM with a Transformer encoder and introducing character-level embeddings would improve upon the baseline scores. After spending many hours implementing this new structure, running 3 separate experiments that failed to outperform the baseline, and attending office hours, we found that this was not the best approach. Figure 2 below visualizes the worsening performance of each iteration, none of which outperformed the baseline.

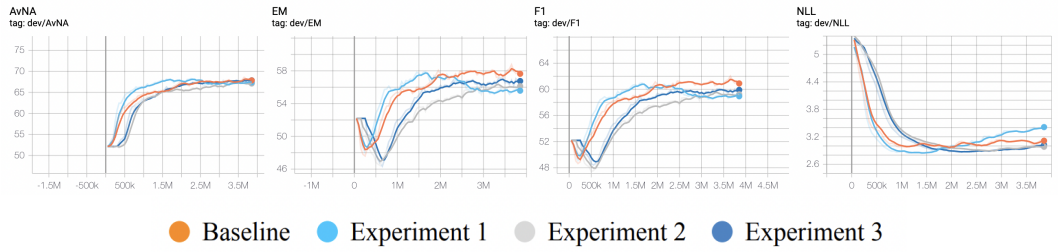

Figure 2. First experiment phase of BiDAF model using a Transformer encoder

After analyzing these results, we pivoted to implementing QANet from scratch instead. We mainly referenced the works of Yu et al. [3] and created a QANet model by altering the original BiDAF embedding, embedding encoder, context-query attention, model encoder, and output layers. The main difference between the implementation of the QANet and BiDAF models is the structure of their respective encoding mechanisms. BiDAF utilizes RNNs within the embedding encoding and modeling layers while QANet is focuses on convolution and self-attention layers in the encoder block. QANet also uses character-level embeddings, which the original baseline BiDAF model does not have. Our first approach to implementing QANet is visualized in Figure 3 below.

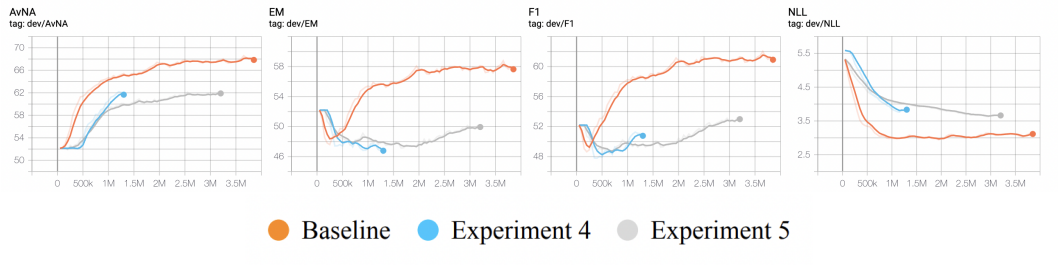

Figure 3. First phase of experiments for implementing QANet

Unfortunately, our two experiments (4 and 5) within this iteration both performed much worse than the baseline with little to no improvement from the original starting point. Because these implementations were not making improvements and to conserve Azure credits, we decided to halt both of these experiments.

We hypothesize that the poor performance was due to the sheer size of our model. The 3 encoder blocks each contained 3 layers and the embedding encoder layer had 5 layers. We had also not yet implemented a linear projection layer to follow the embedding and BiDAF attention layers. Because of this, we were not projecting the embeddings to the hidden size, causing the size to exponentially increase. By the time we concatenated the two encoders, the embeddings had grown to 16x the hidden size. Another note is that we were forced to reduce the batch size from 64 to 32 due to a combination of exponentially large embeddings, a consequently substantial amount of parameters, and the limited CUDA storage available. This decrease in batch size introduced noise and could have further worsened performance.

When neural networks are too large and have too many parameters, possible solutions include stopping training early, regularization, or simplifying the model. We found that halting training early did not improve the performance, so we decided to simplify the model by decreasing the hidden size to 128 and consequently avoiding overfitting. We also implemented a linear projection layer to avoid creating exponentially large embeddings and number of parameters. The graphs below in Figure 4 represent this iteration's results.

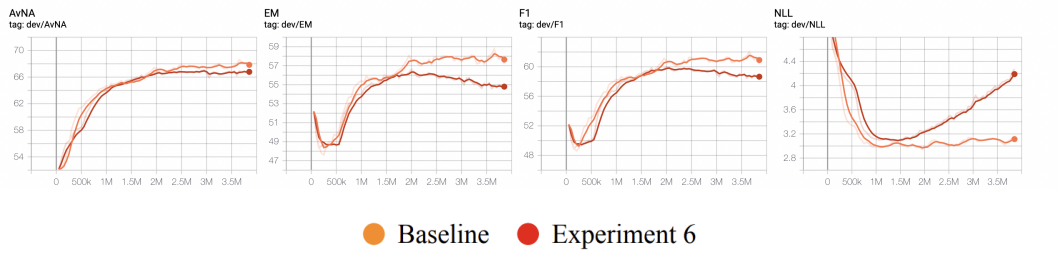

Figure 4. Second phase of experiments for implementing QANet with simplification

After implementing these new features, however, we observed the upward projection of the model's loss halfway through the 30 epochs as shown in Figure 3. Our model overfit to the training data, resulting in overly sensitive adaptations to the dev set and worsening performance around epoch 15. To attempt fixing this overfitting problem, we further decreased the hidden size to 100 and changed the learning rate from 0.5 to 0.75. While this iteration performed better than the last, it still failed to outperform the baseline results (Figure 5).

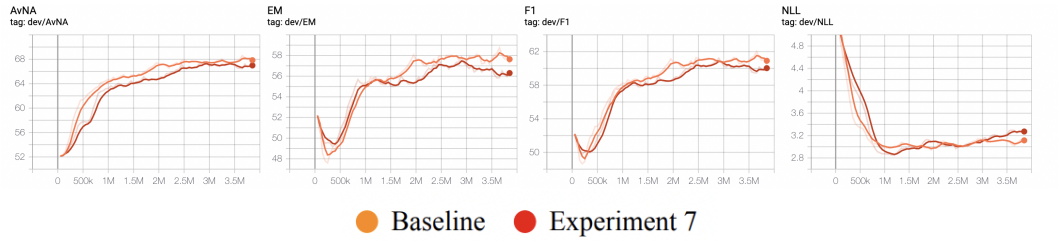

Figure 5. Third phase of experiments for implementing QANet with altered learning rates and hidden sizes

For our next experiment, we decided to use a convolution layer as our projection layer instead of a linear layer. We were skeptical of this change improving the model, as language depends on long-term dependencies rather than short-term dependencies and convolution focuses on the latter. Our hypothesis that convolution would not improve the model was correct and the results are as follows in Figure 6.

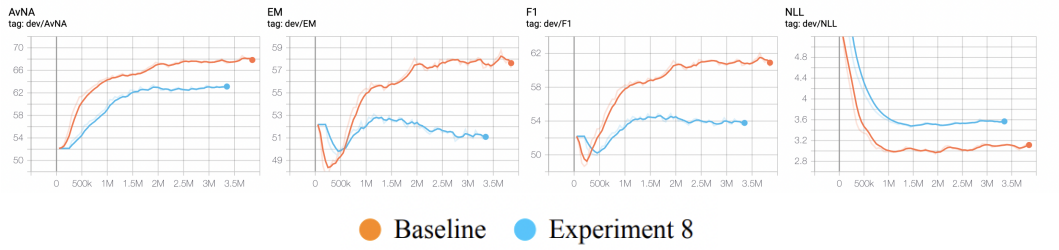

Figure 6. Fourth phase of experiments for implementing QANet with convolution

From the training loss data, we found that our model does not seem to be as generalizable. Thus, we discarded the convolution projection layer and decided to use data augmentation for the question query instead. We specifically used a dropout probability of 0.1 for each word in the question query, excluding the <START> token. Li et al. explains that the concept of "word-dropout" can improve generalization by reducing the information in sentences through "_" placeholders for random words [4].

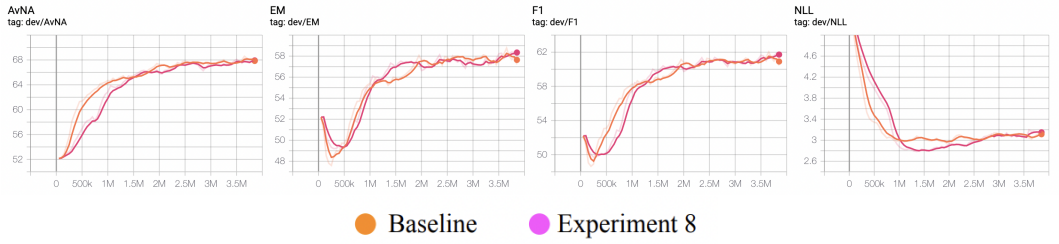

Figure 6. Fifth phase of experiments for implementing QANet with data augmentation

In our next iteration, we kept the same data augmentation implementation but also added learnable positional encodings to the encoding layer. The original architecture encoded positions into hidden layers using a sinusoidal lookup table, but we hypothesized that learnable parameters would help the model learn positional encodings better. One advantage is the flexibility to deal with non-sequential insertions into the data that fixed positional encodings would not handle well.

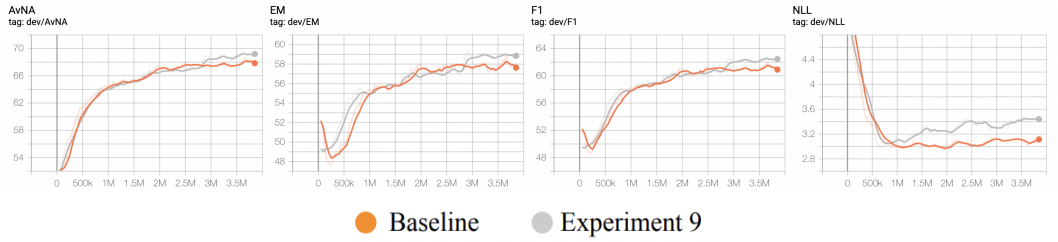

Figure 7. Final phase of experiments for implementing QANet with data augmentation and learnable positional encodings

For the dev set, our implementation beat the baseline with scores of **F1: 62.690, EM: 59.120, and NLL: 2.993**. More dev set results are shown in Figure 7.

All of the mentioned experiments completed in approximately 5 hours, with about 10 minutes dedicated to each epoch. These experiments were also all run on Azure NC6s v3 virtual machine with evaluation performed every 20,000 iterations. The learning rate for each experiment ranged from 0.5 to 0.75, and the last experiment with our best results used a learning rate of 0.75.

5.4 Results

The below table quantifies the dev set results from our original introduction of transformer architecture and our later QANet implementation. We found that our improved QANet implementation with data augmentation and learned positional encodings produced the best scores, outperforming the baseline BiDAF model. Our QAnet implementation with only data augmentation, however, outputted the lowest NLL loss.

Phase	Experiment	F1 Score	EM Score	NLL
	Baseline	62.027	58.763	2.837
Transformer	Experiment 1	60.960	57.920	2.822
	Experiment 2	59.856	56.646	2.884
	Experiment 3	60.090	57.120	2.867
QANet	Experiment 4	52.190	52.190	3.760
	Experiment 5	53.110	52.190	3.639
	Experiment 6	59.960	56.610	3.083
	Experiment 7	61.200	57.860	2.851
	Experiment 8	61.869	58.478	**2.786**
	Experiment 9	**62.695**	**59.116**	2.993

Figure 7. Experimental dev set results

In terms of the test leaderboard (non-PCE), our highest scores are as follows: **F1: 57.819, EM: 61.465**.

These results validate our hypothesis that the QANet architecture outperforms the BiDAF model on SQuAD 2.0. However, we had expected our implementation of QANet with only data augmentation (Experiment 8) to also produce better results than the baseline. With the new introduction of data augmentation, this model produced worse F1 and EM scores, but a lower NLL loss. We suspect that our low scores may be due to the structure of the loss function that does not optimize for F1 and EM scores.

6 Analysis

Note that most of our analysis is in Section 5.3, as we investigated different methods to improve the model during each experimental iteration. Nonetheless, we performed data analysis on our model's performance for the dev set below in Figure 8. We evaluated the prediction as correct if the general segment encompassed the same idea as the actual answer (ie. "money from foreign Islamist banking systems" matches "with money from foreign Islamist banking systems). We recorded the accuracy

percentages of 100 dev set examples available on Tensorboard. We expected that the model would exhibit worse performance on "why" questions, as these tend to require more logical reasoning that is difficult for models to learn. Beware that Figure 8 is slightly misleading, however, because there was only one "why" example in the dataset ($n = 1$). Excluding "why" questions, we found that our model performs the best on "when" questions ($n = 13$). We hypothesize that this is due to the explicit trivial nature of "when" questions that most likely have a calendar date, month, or year as the answer. There were also very few "how" ($n = 11$) and "where" ($n = 4$) examples as well. "What" questions ($n = 54$) did not produce high accuracy either, and we believe that this can be attributed to the broad set of possible answer categories for this type of question.

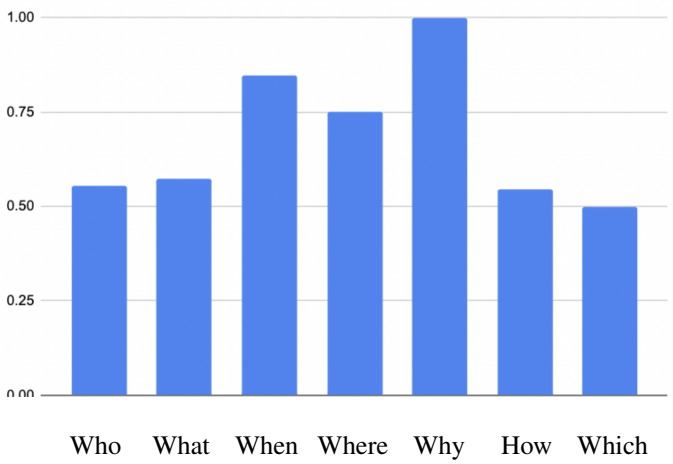

Figure 8. Accuracy percentages for different types of questions on dev set

We also observed that questions that do not start with either "what," "when," "who," "why," or "how" are less likely to be predicted correctly by our model. We suspect that this is because our model is unable to generalize the question without having the specific target as the query's first word. Examples of incorrect predictions are shown below in Figure 9.

Question: Orientalism refers to how the South developed a what of the North?

- **Answer:** N/A, **Prediction:** imaginative geography

Question: The Presiding Officer tries the achieve a balance between similar viewpoints and whom?

- **Answer:** N/A, **Prediction:** political parties

Question: The time required to output an answer on a deterministic Turing machine is expressed as what?

- **Answer:** state transitions, **Prediction:** "yes" or "no")

Figure 9. Incorrect question predictions of QANet implementation for indirect queries

Though our reported QANet dev set results for Experiment 8 did not beat the baseline BiDAF scores, we believe that our model with data augmentation can still perform better in terms of negative log likelihood. The training loss proved to be lower than that of the baseline, but we were unable to evaluate this metric in the test leaderboard. Further examining the NLL loss can give us more perspective on the performance of our model, rather than only comparing F1 and EM scores.

7 Conclusion

For this project, we evaluated many different model implementations on SQuAD 2.0 in pursuit of improving the default BiDAF baseline F1 and EM scores.

Our initial hypothesis that an extended BiDAF model with a Transformer encoder would improve upon the baseline BiDAF scores. Though this was invalidated, we believe that this was an interesting finding to note.

We pivoted to implementing a QANet model with built-in character-level embeddings and learned that a simple QANet architecture is not enough to outperform the original BiDAF model. Our main findings were that a QANet implementation with additional data augmentation and learned positional encodings produces better F1 and EM scores when compared with the BiDAF baseline.

Our QANet implementation with data augmentation produced the lowest NLL loss and we believe that with more time, it would have been possible for this model to also outperform the baseline. Further examining the loss function and potentially making changes could better approximate the F1 and EM metrics.

With our limited time and pivot from transformers to QANet models halfway through the process, we were unable to test whether the Transformer-XL architecture could improve the QANet model for SQuAD 2.0. However, we believe that this would have further improved our model and maintained higher scores than the baseline due to its recurrence and attention mechanism that better handles long-term dependencies. More research in the space of QANet model implementation for SQuAD 2.0 is encouraged.

References

[1] Pranav Rajpurkar et al. Squad: 100,000+ questions for machine comprehension of text. In *Stanford University*, 2016.

[2] Minjoon Seo et al. Bi-directional attention flow for machine comprehension. In *University of Washington, Allen Institute for Artificial Intelligence*, 2018.

[3] Adams Wei Yu et al. Qanet: Combining local convolution with global self-attention for reading comprehension. In *Carnegie Mellon University, Google Brain*, 2018.

[4] Bohan Li et al. Data augmentation approaches in natural language processing. In *Harbin Institute of Technology, Harbin, China*, 2021.