Adapting and Improving QANet to SQuAD 2.0

Stanford CS224N Default Project
Track: SQuAD-IID
Mentor: Kaili Huang

Jenny Yang
Computer Science
Stanford University
jjyang1@stanford.edu

Brad Nikkel
Symbolic Systems
Stanford University
bnikkel@stanford.edu

Abstract

We built a question answering (QA) model based on QANet, developed for the Stanford Answering Dataset(SQuAD) version 1.1, and apply it toward the SQUAD 2.0 data set. Without pre-trained language models, we aimed to achieve similar performance as QANet's authors. Our contributions were implementing and exploring the performance of both BIDAF and QANet. Our highest scoring model is an ensemble of QANet and BIDAF models achieving F1/EM scores of 66.03/62.77 on the test leaderboard. Our highest scoring non-ensemble model comes from an adaptation of BiDAF, achieving a F1/EM performance of 64.55/60.91 on the test leaderboard.

1 Introduction

Given an inputted natural language query, effective QA models should output a relevant answer from a document or database. Such answers can be generated or simply selected from a context text. Our project aims for the latter; that is, we either returns a sequential selection of text from a context or returns "N/A" if no such answer exists. Such systems important because they are now ubiquitous in search engines and dialogue systems. Since people now routinely rely on computing systems to answer their questions, it is imperative that we improve the QA-systems' accuracy [1].

QA systems often struggle to accurately answer questions for several reasons. For one, answers might require bits of information spread throughout a context, meaning a model needs to somehow keep track of earlier information while processing further information, similar to how humans' working memory can remember a chunk of information from early in a paragraph as they read the end of the paragraph and can understand how that earlier chunk relates to the latter text. This is known as a long range dependency. Second, different question variants (who/what/when/where/why/how) often require different approaches toward retrieving a reasonable answer. Thus, creating a QA-model that generalizes to all question variants *and* works well with long range dependencies is difficult. Finally, a QA system should model its confidence that an answer exists so that it does not needlessly return faulty answers to users.

We were motivated to implement QANet because it was significantly faster to train on SQuAD 1.1 and faster at inference than most models in 2018 when the paper was published [2]. We believed this increased training speed would allow us more experimentation, though we did not find this to be the case. We also were curious how QANet would work on SQuAD 2.0's "no answer" cases. Our general approach is as follows:

1. We generated BiDAF baselines
2. We implemented QANet close to Yu et al.'s design [2]
3. We tested the original QANet architecture on squad 2.0 and tested alterations to components.
4. We ensemble our best (by development results) models.

Hoping to improve Yu et al.'s design, we altered the number of attention heads and encoder blocks, testing different permutations. Given our GPUs memory constraints, we had to also adjust batch sizes when altering QANet's attention heads and encoder block quantities limiting our ability to draw strong conclusions comparing our model's performance the original QANet. Given our GPUs, though, our best QANet model had 4 attention heads and 7 encoder blocks, whereas Yu et al. [2] used 8 attention heads and 7 encoder blocks.

2 Related Work

Many of the QA models tested on SQuAD employ some type of attention. When introduced in 2016, BiDAF used context-query attention sequence aligned Recurrent Neural Networks (RNNs) to achieve 77.3 F1 and 68.0 EM scores on the SQuAD leaderboard [3]. BiDAF's key contribution was extending its attention mechanism beyond question-to-context to also include context-to-question awareness. BiDAF then applies Long Short Term Memory (LSTM) to its bidirectional attention representation to form span predictions.

Later, the Tranformer model replaced sequence-aligned RNNs with self-attention to represent inputs [4]. Then, QANet achieved better results (84.6 F1, 76.2 EM) on the SQuAD leader board by combining elements of the Transformer and BiDAF model [2]. Maintaining BiDAF's context-query attention, QANet swaps BiDAF's recurrence-based encoder with a transformer-encoder, relying entirely on convolutions and self-attention. This allows for parallelized computation, resulting in a more training speeds 3-13 times faster than other State of the Art (SOA) models at that time. The increased speed allowed the QANet authors to feed their model more data, which they did via back-translation.

After QANet, Bidirectional Encoder Representations from Transformers (BERT) achieved even higher SQuAD leaderboard scores but pretrained models that encoded bidirectional representations of unlabeled data, allowing a single additional output layer, with trivial modifications, to address various language tasks including QA. Some of the top ten SQuAD-leaderboard models are still BERT-based models.

In our project, we hope to build on prior work by implementing QANet and validating its performance on SQuAD 2.0. In particular, we hope to be able to improve the model to account for non-answerable questions in SQuAD 2.0. differences in the dataset. Furthermore, due to our financial, computational, and memory limitations as students, we hoped to explore ways to made QANet smaller while retaining performance.

3 Approach

3.1 Baseline BiDAF

For our first baseline, we used the provided Bidirectional Additional Flow (BiDAF) starter code for Stanford Question Answering Dataset (SQUAD) 2.0 [5]. Next, since Yu et al. constructed their QANet architecture with character embeddings concatenated to word embeddings [2] and since the BiDAF starter code did *not* use character embeddings, we added character embeddings to BiDAF for our second baseline. Like Yu et al. [2], we represent each character with a 200 dimension, trainable vector, though we experimented with 64 dimension character embeddings too.

3.2 QANet

Once we produced our baselines, we proceeded to implement QANet close to the manner described by Yu et al. [2]. This entails five key components: (1) input embedding layer, (2) embedding encoding layer, (3) context-query attention layer, (4) mode-encoder layer, (5) output layer.

(1): For our input embedding layer (1), we used pretrained Glove word embeddings concatenated to corresponding convolved, trainable character embeddings. We then put this through the highway encoder provided in the BiDAF starter code [6].

(2): Next, we structured our embedding-encoder blocks similar to Yu et al.'s design shown in the right side of Figure 1 1. The first layer is a cosine-sine positional encoding that Yu et al. used [2].

The next part is a series of four layernorms (from Pytorch) and depthwise separable convolutions [7]. After this series is another layernorm and then a multi-head self-attention layer which we coded from scratch, using dot-product similarity [4]. We experimented with different numbers of heads, which we will detail in section 5. Then, we finish the basic encoder block with another layernorm and a simple convolution feedforward network. Each of these three basic components (convolutions/self-attention/feedforward net) is connected via a residual block, meaning that for any input x and function f, we output $f(layernorm(x)) + x$.

(3): After the embedding-encoder block, we use a context-query attention block similar the one Yu et al.'s describes [2]. Our context-query attention computes similarities between context-query word pairs using the similarity matrix S from the BiDAF starter code [**?**] with slight modifications. We then apply a softmax to S to normalize each row.

(4): Next, like Yu et al. [2], we use a model-encoder layer similar to Seo et al. [8], using our same Embedding Encoder Block architecture, except that here we convolve 2 times instead of 4 and use 7 blocks instead of 1. We experimented with 5 and 9 blocks too, which we will detail in section 5.

(5): Finally, we use an output layer. This layer computes probabilities of each word being an answers' start or end positions with a strategy similar to the standard BiDAF [8], where the start and end point probabilities are

$$p1 = LogSoftmax(W1[M1; M2]), p2 = LogSoftmax(W2[M1; M3])$$

where $M1, M2$ and $M3$ and the model encoders from bottom to top 1 and $W1$ and $W2$ are trainable variables.

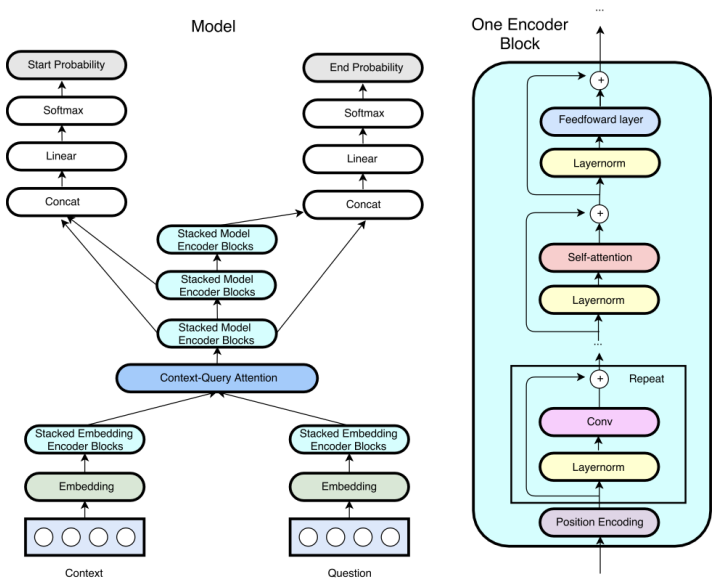

Figure 1: QANet Architecture

3.3 QANet Extensions

Upsized and downsized blocks: Yu et al. [2] used stacks of 7 encoder blocks in their experiments. Due to memory constraints and in the interest of speeding up training time, We experimented with different numbers of blocks. We discuss our findings from these experiments in Sections 4 and 5.

Voting Ensemble: To take advantage of the fact that different models performed better and worse on different questions, we used an ensemble voting model to compute the output on the dev and test sets. First, we assign each model a weight, which is an indicator of how good that model's performance is. The weights were computed as the L1 norm of the model's F1 score on the Gradescope dev submission. Next, we feed in the dev/test dataset into each model that we are using in the ensemble.

Using their outputs, we construct a voting system. If any two models have the same output, then the score of their answers is summed together. Finally, for each question, we pick the answer as the one receiving the max score after voting. In this way, we effectively combine the outputs of multiple models with minimal additional processing and memory.

How many heads? Yu et al. [2] used 8-headed attention throughout their paper. Increasing the number of heads increases the number of learnable parameters, which may not always improve performance. Additionally, in our initial trials, we found that using 8 heads both resulted in poor perofmrance and took very long to train. As a result, we experimented with different numbers of heads.

Leaky ReLU and PReLU: In our initial model training, we found that our model would try to always answer unanswerable questions. Theorizing that our model was zeroing out most of our negative values, we experimented with leaky ReLUs and PReLU. We hoped that this would work better with our negative log likelihood loss function. However, we found that these activation functions needed to be tuned meticulously anytime we altered other hyperparameters. As a result, our most successful models ended up using normal ReLUs.

4 Experiments

4.1 Data

Our data set is from SQuAD 2.0 which contains approximately 150k questions, roughly half answerable and half unanswerable [5]. Our training set, however, contains roughly 130k examples from SQuAD 2.0's training set and our development and test sets contain approximately 6k examples each [9]. SQuAD 2.0's official test set is restricted from public use.

4.2 Evaluation method

We quantitatively evaluate our models' performance on SQuAD 2.0 using the F1 score. We also use the Exact Match (EM) and Answer vs. No-Answer (AvNA) scores. The EM score is binary, awarding points only if our returned answer is exactly the provided answer. The F1 score is the harmonic mean of precision and recall [9]. AvNA measures a model's accuracy in determining if a question has an answer or not. These scores are based on the highest single score that our model achieves from three human crowd-sourced golden answers available per question.

4.3 Experimental details

BiDAF Our first 4 experiments used the BiDAF starter code and paper's parameters, namely 0.5 learning rate, Adadelta optimizer, 100 hidden state size, 0.2 dropout rate, and 30 training epochs [3].

For our baseline we use the provided BiDAF *without* character embeddings, using the default 64 batch size. Training took approximately 3 hours and 14 minutes. Next, we added 64-dimension, trainable character embeddings to BiDAF *with* using 64 batch size. This trained for approximately 5 hours and 4 minutes. Then we added 200-dimension, trainable character embeddings to BiDAF *with*, using 32 batch size. This took approximately 7 hours to train. For our final BiDAF experiment, we added 400 dimension character embeddings to BiDAF *with*, using 200 batch size.

QANet: We then performed QANet experiments. Since 200-dimension, trainable character embeds performed the best in our BiDAF experiments, we used 200-dimension character embeds for all QANet experiments. We used 128 hidden state size. With the following hyperparameters, following Yu et al.'s QANet architecture [2]. We used L2 weight decay of $3 \cdot 10^{-7}$, stochastic depth layer dropout inside each encoder layer, and a general dropout rate of 0.1 between layers to regularize our training. Additionally, we use an Adam learning rate warm up scheme that settles at 0.001 with $\beta 1 = 0.8$, $\beta 2 = 0.999$, and $\epsilon = 10^{-7}$. Unfortunately, we could not always replicate Yu et al.'s 32 batch size due to memory constraints. Unless stated otherwise, we ran QANet experiments for 30 epochs.

We ran experiments on Azure's Standard NC6s v3 (6 vCpus, 112 GiB memory) and Google Colab Pro+, whose machines varied from K80, P100, T4 with 52 GiB memory). Because we used varying

machines, we will not draw strong conclusions about training times but will comment on relative training times as reported in Tensorboard.

Varying Block Depth and Heads: With QANet, we experimented with one, two, four, and eight headed self attention and experimented with 5, 7, and 9 encoder block depths.

Ensemble: After our single model experiments, we ensembled a few different permutations of models. For one ensemble, we ensembled out top three highest performing models, BiDAF with 200-character embeds, ... For another ensemble, we ensembled our QANet model, trained from three different seeds. We noticed, when experimenting with running QANet from different seeds, that the results varied significantly, suggesting that the earlier questions our model trained on significantly effected its overall performance. Thus, we hypothesized that ensembling our model from several training run, all from different seeds, may make the ensembled model more generalizable.

4.4 Results

Our highest scores on the IID SQuAD track test leader board are F1: , EM: . In Table 1, we show the results of our single-model experiments; the highest single model of each type is in bold.

Single Model	**Details**	**Dev F1**	**Dev EM**	**Dev AnVA**	**Test F1**	**Test EM**
BiDAF (baseline)	100 hidden size	60.98	57.87	67.29	-	-
BiDAF w/ char embeds	64-dim	64.24	60.66	71.06	-	-
BiDAF w/ char embeds	**200-dim**	**66.17**	**62.63**	**72.34**	64.54	60.91
BiDAF w/ char embeds	400-dim	61.15	58.21	68.81	-	-
QANet(2-heads)	5 enc. blocks	61.35	57.18	68.51	-	-
QANet(2-heads)	7 enc. blocks, 64 batch	60.22	56.38	67.4	-	-
QANet(4-heads)	5 enc. blocks	52.61	52.19	62.64	-	-
QANet(4-heads)	7 enc.blocks,300-seed,50 epochs	63.01	58.88	70.06	-	-
QANet(4-heads)	**7 enc. blocks, 50 epochs**	**65.04**	**61.2**	**71.85**	-	-
QANet(4-heads)	9 enc. blocks	52.69	52.19	63.45	-	-
QANet(8-heads)	7 enc. blocks	52.31	52.19	62.83	-	-
QANet(4-heads)	7 enc. blocks,***Squad 1.1***	62.35	54.6	-	-	-

Table 1: Performance of Single Models

Ensemble Model	**Dev F1**	**Dev EM**	**Test F1**	**Test EM**
BiDAF 200-dim + 64-dim char embed + QANet 4head	**67.29**	**63.89**	66.03	62.77
QANet from 2 seeds + BiDAF 200-dim	64.47	61.03	-	-
BiDAF 64-dim + 200-dim + 400-dim char embed	66.17	62.98	-	-

Table 2: Performance of Ensemble Models

The results of our ensemble experiments are in Table 2. Our highest performing ensemble model is in bold. We had expected our QANet would achieve higher F1 and EM scores than our model achieved but we did anticipate that SQuAD 2.0's "no answers" might result in altered performance relative to the original QANet paper's results on SQuAD 1.0. But, we did not think the scores would be nearly 15 points less that the QANet paper's top performance of 80 F1 score (on SQuAD 1.1). Interestingly, our QANet training F1 scores for for SQuAD 2.0 and SQuAD 1.1 did not vary significantly. Though not certain, we suspect with more memory and increased batch sizes, our QANet model might perform better with 8 heads than it did. This suggest to us that we might have increased our models performance by modifying Yu et al's [2] design to work better with limited memory and thus limited batch sizes.

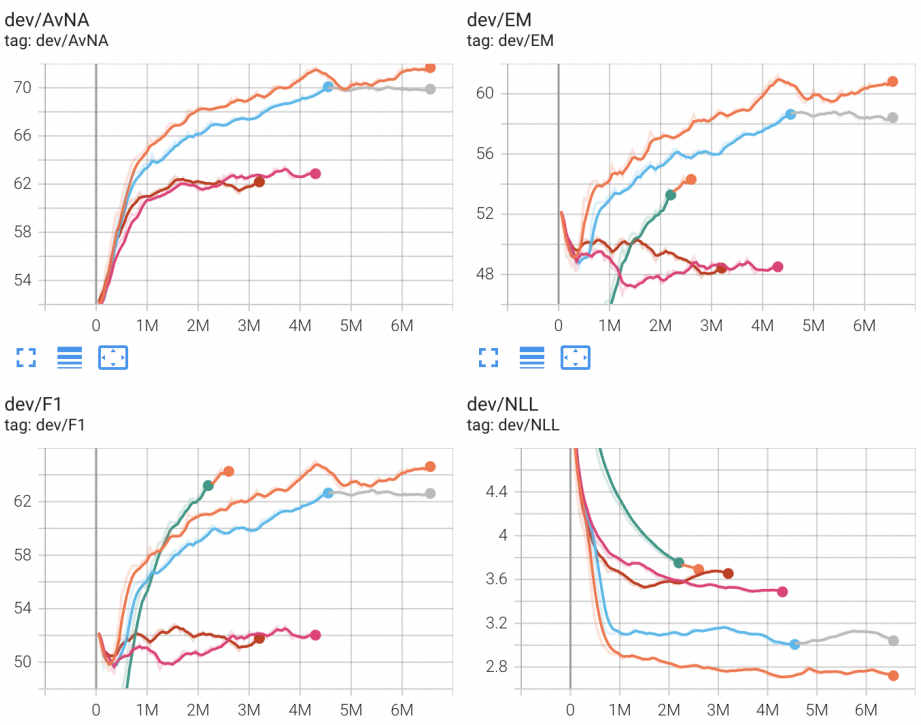

Figure 2: Four-Headed QANet Training Tensorboard Plots

5 Analysis

Encoder Block Depth and Attention Heads: We experimented with different numbers of the stacked encoder blocks and the number of attention heads. For attention heads, we found that 8-headed attention, which the original QANet used, performed the worst for our model. Due to memory constraints, however, we had to reduce the batch size for 8-headed attention to 24, where Yu et al. [2] used 32 batch size. It could be that with a higher batch size, our model would perform better with 8-headed attention or we might have a flaw in our design. For our QANet, 4-headed attention performed the best. We found 2-headed attention performed slightly worse than 4-headed attention, but training with 2-heads was significantly faster due to our ability to increase the batch size. Our encoder-block depth experiments showed that with lesser heads (2-heads), we could get similar results with a shallower (depth 5) encoder-block stack as we could with the the original QANet stack size (depth 7). But, when we increased the heads to 4, we found that depth 7 performed much better than a 5 or 9 depth encoder-block stack. This suggests that the ideal encoder-block stack depth is somewhat dependent on the number of attention heads and optimizing these might increase the model's training time and accuracy. The plot is shown above.

1. Orange = Four-headed QANet with 7 stacked encoder blocks, 50 epochs
2. Cyan = Four-headed QANet with 7 stacked encoder blocks, 50 epochs, seed 300
3. Pink = Four-headed QANet with 9 stacked encoder blocks
4. Red = Four-headed QANet with 5 stacked encoder blocks
5. Green = Four-headed QANet with 7 stacked encoder blocks, ***SQuAD 1.1***

There are more training and dev plots comparing the difference in performance and learning among models with different numbers of heads and encoder blocks in the appendix.

Performance Breakdown by Question Type: We divided all the questions on the dev set into 9 categories: which, who, what, where, when, why, how, quantitative, and other. We found that BiDAF and QANet performed significantly different on certain question types. Namely, BiDAF significantly

outperformed QANet on quantitative questions. QANet significantly outperformed BiDAF on "why" questions. Both QANet and BiDAF performed their worse on "other" questions. Manual inspection revealed that the category of "other" had some questions that were mispelled, some fill in the blank questions, and some yes/no questions. We attribute the success of our ensemble model to the ability of each model to perform well on different types of questions.

Question type	Which	Who	What	Where	When	Why	How	Quantitative	Other
QANet	68.39	65.03	65.34	61.69	70.01	65.02	64.04	55.61	44.23
BiDAF	70.62	68.54	65.01	65.43	74.62	61.27	66.67	71.05	51.82
Count	215	668	3597	275	442	83	524	32	115

Table 3: F1 score and count across categories

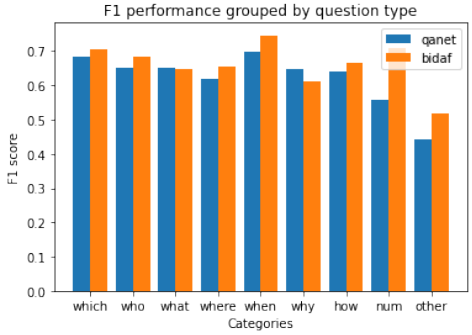

Figure 3: F1 of different question types

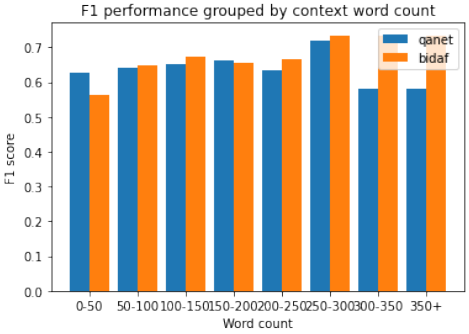

Figure 4: F1 of different context lengths

Performance breakdown by context length Given that the contexts provided with each query had varying length, we were curious how different models would handle the challenge of different context lengths. To begin, we bucketed all the contexts provided from the dev set into one of 8 buckets, each one spanning 50 words. These buckets started from 0-50 to 350+. The F1 performances across each bucket are shown in the figure above. We were surprised to find that QANet did notable worse on longer queries of 300 words or longer. We theorized that BiDAF's RNN encoder would allow it to process and extract information from longer peaces of text. We also found that both models performed the best on contexts with 250-300 words.

Word Count	0-50	50-100	100-150	150-200	200-250	250-300	300-350	350+
QANet	62.51	64.04	65.17	66.23	63.36	72.01	57.88	57.91
BiDAF	56.43	64.9	67.07	65.41	66.68	73.42	73.19	73.21
Count	56	1986	2357	1065	310	158	29	117

Table 4: F1 score and count across context word counts

Ensemble analysis Because of the success of our ensemble model, we were motivated to understand which of the models in the ensemble were contributing the most final answers. On our best ensemble model, we found that when applied to the dev dataset, BiDAF with 200-dim character embeddings contributed 4264 answers, BiDAF with 64-dim character embeddings contributed 3717 answers, and the QANet contributed 4172. The numbers add up to more than the total number of queries in the dev set because often times two models contributed the same answer. We see that the number of answers contributed by each model is approximately proportional to the model's F1 score on the dev dataset, showing that our weighted voting metric preserved relative strengths of each model.

6 Conclusion

In this project, we implemented QANet to test its performance on SQuAD 2.0. Overall, we found that our implementation of the original QANet architecture did not train as fast nor perform as well

as we though it would on SQuAD 2.0, falling just short of our BiDAF with 200-dimension character embeddings. We did, however, achieve significant improvement over baseline BiDAF.

We learned that when implementing someone else's model, it is important to consider the computing power they had access to and modify our implementation accordingly. We found that it was unrealistic to replicate all aspects of QANet as it had been presented in the paper. We found through experimentation that we could try changing the layer depth, numbrer of convolutions, and number of heads if using multi headed attention. We further found that each of these hyperparameters had some impact on each other, which made tuning them difficult.

Additionally, we learned the power of error analysis and model ensembling. When we realized that BiDAF and QANet each had their areas of expertise in answering certain types of questions or identifying the answer from different context lengths, we were inspired to implement a voting ensemble, which improved our model performance.

Though we ran out of time, we had wanted to combine elements of Transformer-XL into our QANet architecture. Incorporating these elements would allow our model to learn longer-term dependencies, which should improve performance on question answering tasks. More specifically, we wanted to implement a segment-level recurrence mechanism, meaning the representations computed for the previous segment would be reused as an extended context when the model processes the next new segment. Transfomer-XL is additionally reported to have a faster performance than a vanilla transformer, which might have helped our model train faster as our current average train time for 30 epochs is on the scale of dozens of hours[10].

Additionally, in the future, we would like to try the data augmentation technique that Yu et al. [2] use in their QANet. This entails using ML translation to convert the SQUAD corpora to another language and then back to English. We would then use this second, altered corpora to augment our original SQUAD 2.0 training corpora, providing our model more data to train on without relying on non-SQUAD data.

References

[1] Ravana S. D. Hamid S. Ismail M. A. Shah, A. A. Web pages credibility scores for improving accuracy of answers in web-based question answering systems. In *IEEE Access, 8, 141456-141471*, 2020.

[2] Dohan D. Luong M. T. Zhao R. Chen K. Norouzi M. Le Q. V. Yu, A. W. Qanet: Combining local convolution with global self-attention for reading comprehension. In *arXiv preprint arXiv:1804.09541.*, 2018.

[3] Aniruddha Kembhavi Ali Farhadi Seo, Minjoon and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. In *arXiv:1611.01603*, 2016.

[4] Niki Parmar Jakob Uszkoreit Llion Jones Aidan N. Gomez Lukasz Kaiser Ashish Vaswani, Noam Shazeer and Illia Polosukhin. Attention is all you need. In *http://arxiv.org/abs/1706.03762*, 2017.

[5] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Association for Computational Linguistics (ACL)*, 2018.

[6] Klaus Greff Srivastava, Rupesh Kumar and Jürgen Schmidhuber. Highway networks. In *https://arxiv.org/abs/1505.00387*, 2015.

[7] François. Chollet. Xception: Deep learning with depthwise separable convolutions. In *https://arxiv.org/abs/1610.02357*, 2017.

[8] Ali Farhadi Minjoon Seo, Aniruddha Kembhavi and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. In *arXiv preprint arXiv:1611.01603*, 2016.

[9] Cs 224n default final project: Building a qa system (iid squad track). In *https://web.stanford.edu/class/cs224n/project/default-final-project-handout-squad-track.pdf*, 2022.

[10] Zhilin Yang Yiming Yang Jaime Carbonell Quoc V. Le Ruslan Salakhutdinov Dai, Zihand. Transformer-xl: Attentive language models beyond a fixed-length context. In *https://arxiv.org/abs/1901.02860*, 2019.

A Appendix

The below plots display the progress of our models when training on the development set. If a plots changes colors, that means we resumed training for that model from a checkpoint, thus the key refers to the color that each model starts as.

This first group of plots, in Figure 5, are all our BiDAF dev runs.

1. Red = BiDAF with 200 dimension character embeddings
2. Blue = BiDAF with 64 dimension character embeddings
3. Cyan = BiDAF with 400 dimension character embeddings
4. Orange = Baseline BiDAF

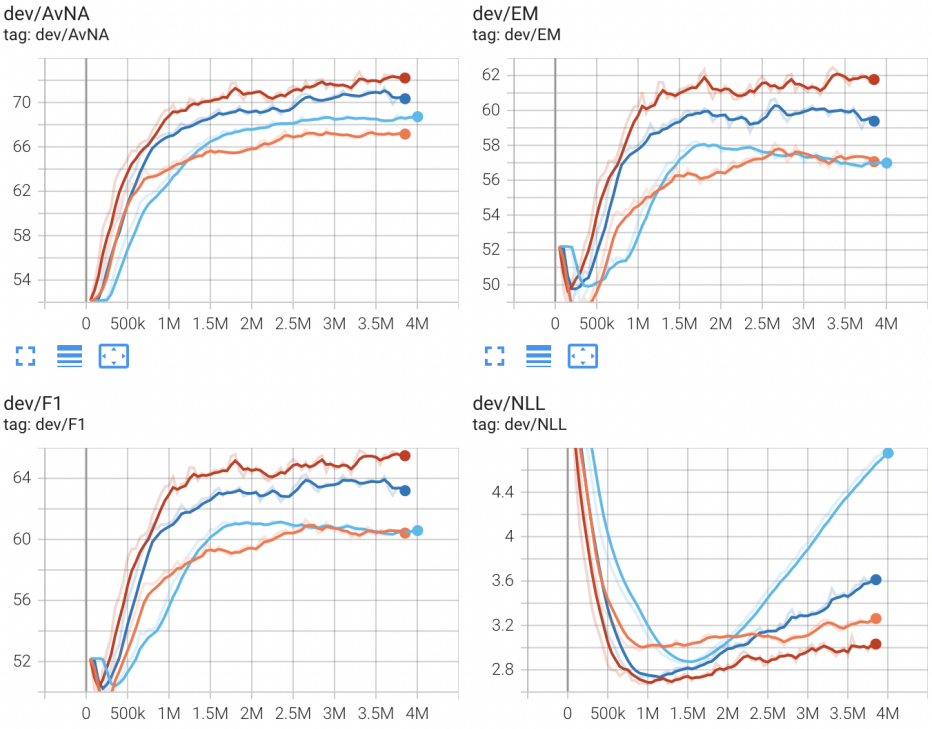

Figure 5: BiDAF Training Tensorboard Plots

Figure 6 is our plots for two-headed attention QANet dev runs.

1. Orange = Two-headed QANet with 5 stacked encoder blocks
2. Blue = Two-headed QANet with 7 stacked encoder blocks

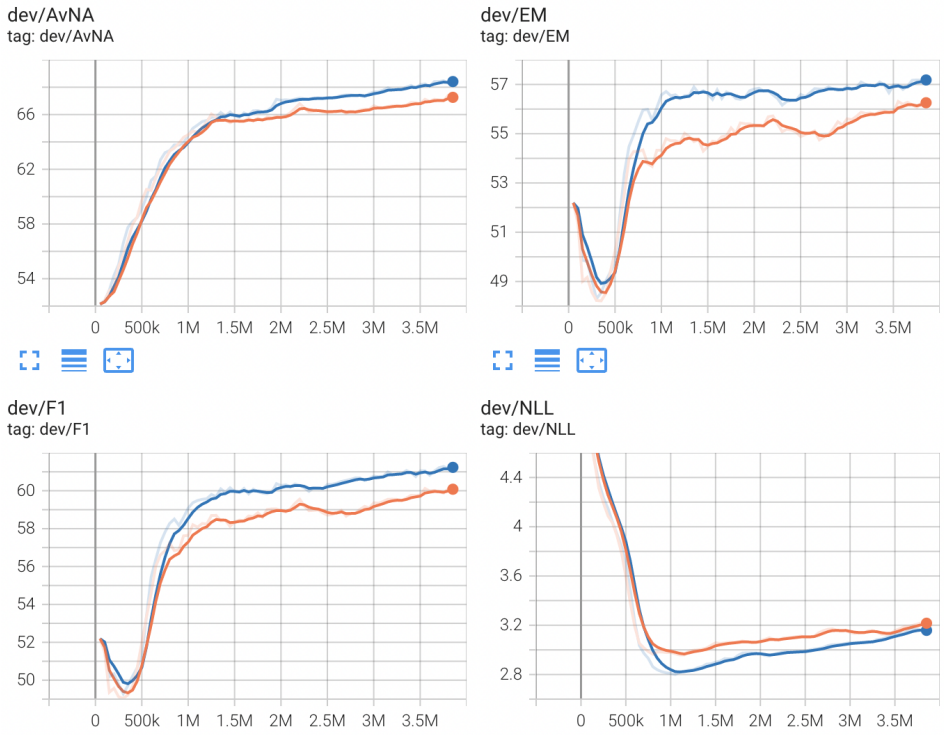

Figure 6: Two-Headed QANet Training Tensorboard Plots

Figure 7 is our eight-headed attention QANet training session. We can see that it over-fitted quite early.

1. Blue = Eight-headed QANet with 7 stacked encoder blocks, 24 batch
2. Orange = Eight-headed QANet with 7 stacked encoder blocks, 24 batch, with leaky ReLUs (0.1 negative slopes)

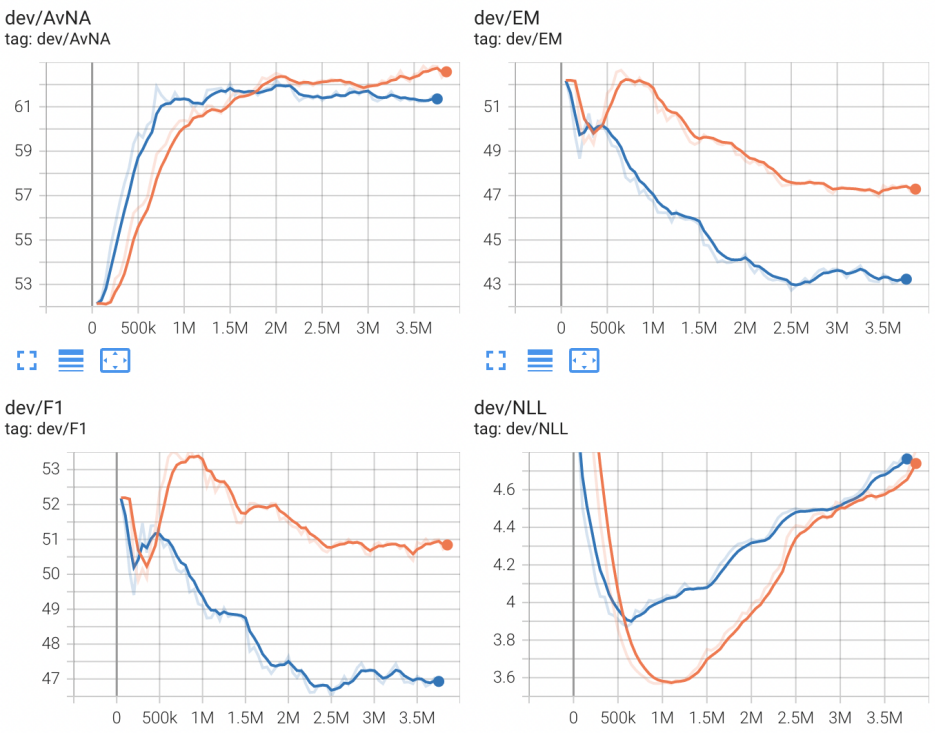

Figure 7: Eight-Headed QANet Training Tensorboard Plots