Improving Question Answering on SQuAD 2.0: Exploring the QANet Architecture

Stanford CS224N Default Project: Final Report

Aayush Agrawal
Department of Computer Science
Stanford University
aayush2k@stanford.edu

Jiwon Lee
Department of Computer Science
Stanford University
jiwonlee@stanford.edu

Laikh Tewari
Department of Computer Science
Stanford University
laikh@cs.stanford.edu

Abstract

In this project, we investigated QANet [1] - an end-to-end, non-recurrent model that is based on the use of convolutions and self-attention. Our first goal was to reimplement the QANet model from scratch and compare its performance to that of our baseline BiDAF [2] - a model that relies on recurrent neural networks with attention. Both of the QA answering systems were tested on SQuAD 2.0 (the Stanford Question Answering Dataset) which includes both questions that are answerable given a context and questions that are not answerable given the context. Finally, after evaluation of our "vanilla" QANet and investigation of related work, we implemented an extended model called EQuANT [3]. The model adds an additional output to explicitly predict the answerability of a question given the context. Our best model (QANet with tuned hyperparameters) achieves $F1 = 57.56$ and $EM = 54.66$ on the developmental set, and $F1 = 56.76$ and $EM = 53.34$ on the test set.

1 Introduction

Question answering and machine reading comprehension tasks have been receiving growing interest in the last five years. As a result, neural models have been making large strides in terms of performance on large, open-domain datasets. These architectures are split into two approaches: PCE (Pre-trained Contextual Embeddings) methods and non-PCE methods. Examples of the former include ELMo and Bert, while examples of the latter are BiDAF and QANet (models we investigate in the paper).

Built by Yu et al. [1], the original QANet was able to achieve a state-of-the-art performance on SQuAD 1.0. They were able to gain this performance boost over previous models like our baseline BiDAF by replacing the use of recurrent neural networks with convolutions and self-attention that boosted speed and efficiency. However, QANet was not tested on a dataset with unanswerable questions like SQuAD 2.0. We wanted to investigate the performance of QANet in this novel territory.

Thus, the focus on this project was to first reimplement QANet, then evaluate its performance against our baseline BiDAF on SQuAD 2.0, and finally explore ways to extend and improve its architecture. Once we had constructed our QANet model, which was faithful to the original implementation in terms of layers, convolution blocks, and hyperparameters, we decided to explore an extension of the model called EQuANT [3] that would compute answerability separately from the probabilities of the span. Finally, we analyzed our model outputs in order to gain intuition about their architecture, understand when and why they were succeeding or failing, and suggest possible improvements to our implementation.

2 Related Work

A particularly successful RNN-based question answering model is the Bidirectional Attention Flow (BiDAF) model by Seo et al [2]. An ensemble model based on this architecture was able to gain state-of-the-art performance in 2016. However, a disadvantage of this model is its heavy use of RNNs that hinders the ability to parallelize computations. This is the baseline that we use as our main point of comparison for our non-recurrent models. It is important to note that the original BiDAF model's EM and F1 scores are based off of the SQuAD 1.0 dataset, not SQuAD 2.0.

The QANet model by Yu et al. [1] that we implemented borrows ideas from NMT in the Transformer architecture. Particularly, the embedding and model encoders discard RNNs and uses a combination of convolutions and self-attention in order to both model local features and process global interactions. QANet was estimated to train 4 times faster than BiDAF and evaluate 7 times faster, meaning that data augmentation could be used to process more training data than BiDAF. Like the BiDAF model, the original paper bases their scores off of the SQuAD 1.0 dataset.

With the release of the SQuAD 2.0 dataset introducing unanswerable questions, the EQuANT model by Aubet et al. [3] focuses on answerability as an extension of the QANet model. This model implements an answerability module on top of the QANet architecture that outputs a probability score that is used to determine the model's confidence in outputting its answer. Having this mechanism in place in their EQuANT model increased the EM and F1 scores from their vanilla QANet implementation.

3 Approach

We began our project by implementing the QANet model as described in the Yu et al. paper [1] from scratch. The high-level structure of our model contains an input embedding layer, embedding encoder layer, context-query attention layer, model encoder layer, and output layer.

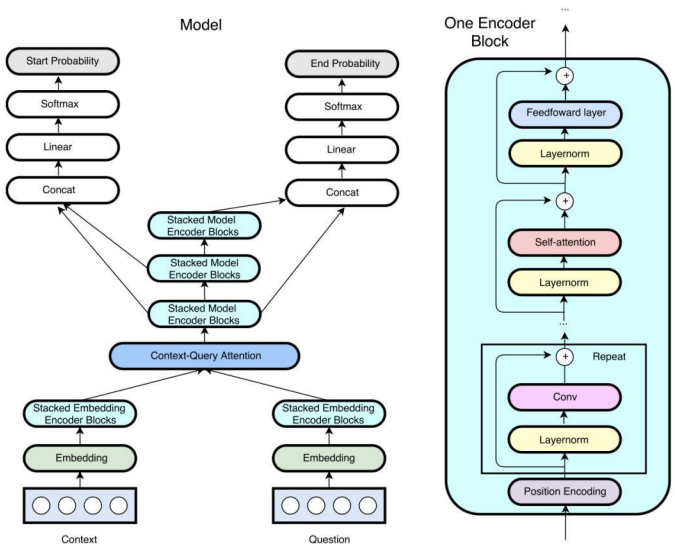

Figure 1: Diagram of QANet architecture, single encoder block sublayer structure shown on right

The Input Embedding Layer obtains the embedding of each word of the context and query passed in by concatenating its word embedding and character embedding. We then leverage the two-layer highway network from the BiDAF baseline to pass the embedding through and generate the output.

The Embedding Encoder Layer is built through the following building block, which uses depthwise separable convolution layers and an 8 headed self-attention layer:

$$(\text{convolution-layer} \times \# + \text{self-attention-layer} + \text{feed-forward-layer}).$$

This layer consists of 1 encoder block consisting of 4 convolution layers with kernel size 7, and each operation in the block is placed inside a residual block, for which the output is

$$f(\text{layernorm}(x)) + x$$

given an input x and an operation f. At the beginning of the layer, a positional encoding is added to the input as defined by Vaswani et al. [4]. The code for the implementation of this positional encoder is taken from the PyTorch documentation. We also use stochastic depth layer dropout [5] in this and the Model Encoder layer between the convolution sublayers where sublayer l has survival probability

$$p_l = 1 - (l/L)(1 - p_L)$$

where L is the last layer and $p_L = 0.9$.

The Context-Query Attention Layer is identical to that of the BiDAF model, for which we leverage the existing code of the baseline and incorporate it directly into our model.

For the Model Encoder Layer, the overall structure is made up of the same encoder building block as that of the Embedding Encoder Layer. One layer is composed of 7 stacked encoder blocks, each with 2 convolution layers with kernel size 5. There are 3 overall repetitions of this whole layer, with each output passed into the Output Layer.

Finally, in the Output Layer, the three model encoder outputs are used to compute the start probabilities and the end probabilities as follows:

$$p_{\text{start}} = \text{softmax}(W_1[M_0, M_1]), p_{\text{end}} = \text{softmax}(W_2[M_0, M_2]).$$

The start index is selected as the index corresponding to the max probability in the start probability vector, and the end index is then selected as the index corresponding to max probability in the end probability vector. Notably, SQuAD 2.0 introduces unanswerable questions, so the output answers are filtered based on the joint probability of the start and end probability vectors.

We then redesigned the goal for a multi-task learning setting to explicitly predict the answerability probability in addition to the answer span. To predict the answerability of a question given the context, we extended our vanilla QANet model by adapting the answerability module from EQuANT [3]. This module aims to generate a score for each answer start-end probability pair that determines whether or not the model should output an answer based on a confidence threshold.

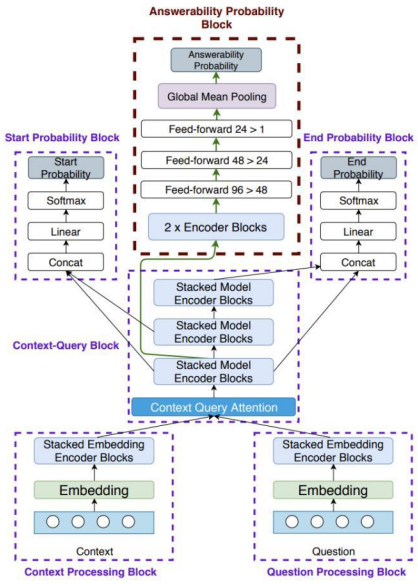

Figure 2: EQuANT architecture builds on QANet by adding a third flow for output from the first model encoder block consisting of 2 encoder blocks, 3 fully connected layers and pooling to reduce the dimension to a scalar, and a final output through a sigmoid

This module is implemented from scratch in the same way as described in EQuANT. The first output of the Model Encoder Layer is passed through 2 encoder blocks and 3 feed-forward layers that decrease the hidden size down to 1. Taking the mean across the sequence length dimension and squeezing the size 1 dimensions, we are left with an answerability score of size 1. This score is compared to a threshold of $p = 0.5$, where the prediction is outputted if it exceeds the threshold, and not given otherwise. Additionally, we adjust the calculation of the training loss to the given function

$$l(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left[\mathcal{L}_0^i(p_0^{(i)}) + \delta^{(i)}\left(\mathcal{L}_1^i(p_1^{(i)}) + \mathcal{L}_2^i(p_2^{(i)})\right)\right]$$

where p_0 is the answerability probability, δ is the ground truth, p_1 and p_2 is the start-word and end-word probability, and $\mathcal{L}_j(p_j)$ for $j = 1, 2, 3$ is the cross entropy loss associated with answerability, start-word and end-word predictions respectively.

Lastly, the baseline model that we compare our model's performance to is the provided BiDAF model implementation from https://github.com/minggg/squad.git which is based on the paper by Seo et al.

4 Experiments

4.1 Data

As previously mentioned, we trained our models on the SQuAD 2.0 dataset. This is in contrast to the original QANet model which was trained on SQuAD 1.0 dataset, which did not have any unanswerable questions. SQuAD 2.0 features data points that each contain a context, question, and answer. The context (which have been truncated to 400 tokens) has been extracted from Wikipedia. The answers to the questions are either unanswerable or a span of the context. The original SQuAD 2.0 dataset has a train/dev/test split of 129941/5951/5915 examples [6].

Roughly 33% of the questions in the train set are unanswerable, while roughly 52% of the questions in the dev set are unanswerable [6]. We performed exploratory data analysis to understand our data in more detail and uncover more information about SQuAD 2.0 as shown in Figure 3.

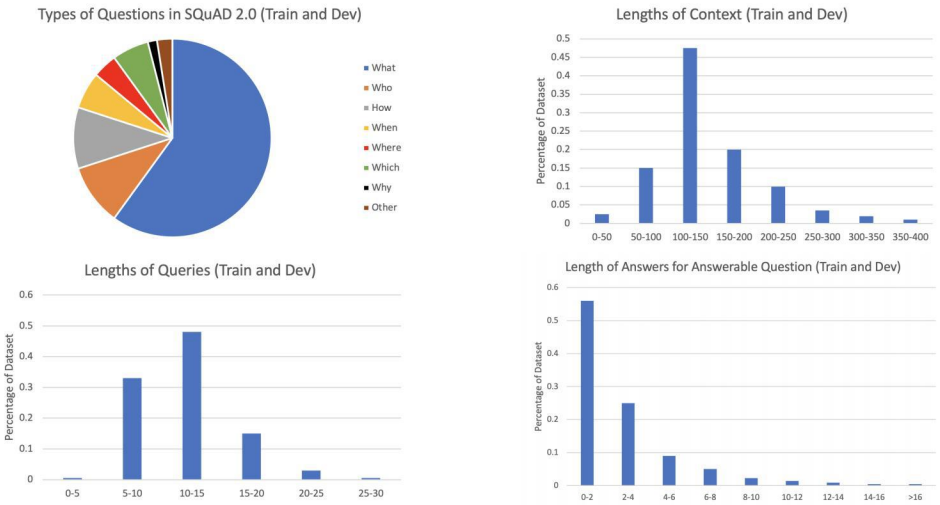

Figure 3: Analysis of SQuAD 2.0 Dataset

We found that the question types are mainly $wh*$ type, heavily biased towards $what$ questions. The lengths of the contexts were particularly important as Transformer-based architectures are relatively poor at modelling long-range dependencies. However, as the lengths were primarily in the 50-200 word range, we decided against implementing Transformer XL [7] features.

Finally, analyzing sample question-answer pairs in the dev set revealed that there were a number of errors in the ground truth. For example, we found a number of illogical/incorrect question and answer pairs:

1. **Question**: What is the force equivalent of torque compared to angular momentum? The answer did not include "momentum", the right answer.

2. **Question**: What are there no longer limitations on since 1990? The question is improper in its phrasing.

4.2 Evaluation method

In order to evaluate our models, we used EM (Exact Match) and F1 scores. EM is a measure of whether the model prediction span exactly matches the ground truth span. On the other hand, F1 is the harmonic mean of recall and precision; recall is the number of right words divided by the number of ground truth words, while precision is the number of right words divided by the number of predicted words. Additionally, we record AvNA scores in our results. AvNA is a measure of whether is model is answering answerable questions and not answering unanswerable ones.

4.3 Experimental details

4.3.1 Implementation of QANet

We began by adapting the model described in the Yu et al. paper [1] with the same parameters, as laid out in the approach section of this report. With our first run, we ran into an issue of exploding gradients. After training for a few batches, the NLL would quickly shoot up to around 10^{15} and then to NaN after a few more batches. Our first attempt to fix this issue was to implement a scheduler that would start the learning rate at 0.0 and increase it to the default learning rate of 0.05 in the first 1000 steps, and then maintain a constant learning rate for the remainder of training. Although this decreased the degree of exploding gradients with an NLL of around 10^8, the problem of having an NaN loss persisted. We eventually traced the issue back to the fixed learning rate, which we decreased from 0.5 to 0.001. This solved our issue, giving us a single digit loss with the first few seconds of training.

Another issue we ran into shortly after was that the standard NC6 GPU that we used to perform some experiments did not have enough memory to allow for a batch size of 32 given our implementation. In order to allow for the model to train without terminating due to insufficient RAM, we reduced the batch size from 32 to 16, which reduced the training speed to approximately 90 minutes per epoch. At this point, we were able to have our first full run, for which we ended up with the following hyper-parameters. We used an Adam optimizer with $\beta_1 = 0.8, \beta_2 = 0.999, \epsilon = 10^{-7}$, and a learning rate of 0.001. For dropout, we used an inter-layer dropout rate of 0.1 as well as different dropout layers for our word and character embeddings with dropout rates of 0.1 and 0.05 respectively.

We let the model run for 12 epochs, but we observed that although the training loss curve looked reasonable, the EM and F1 scores constantly declined from each evaluation step to the next. After taking several deep dives into our code, we reasoned that we may have been masking the multi-head self attention layer in our encoder block incorrectly. After fixing this issue, we also added stochastic layer depth in our encoder block [5] to increase regularization between the convolution layers of the encoder block as described in our approach section. Lastly, we upgraded our GPU to a NC6 v3 GPU, which increased our training speed by a factor of 3, giving us roughly 30 minutes per epoch. We ended up with a baseline QANet model that trained correctly and did indeed increase EM and F1 scores as it trained for more and more epochs. However, we did see some overfitting in the dev NLL loss, which led to our experimentation with hyper-parameter tuning.

4.3.2 Hyper-parameter Tuning

Char Rep	Num Heads	Hidden Size	EM	F1	AvNA
200	8	128	**51.04**	**52.98**	60.53
200	8	96	49.7	51.9	60.54
200	8	48	50.4	52.8	61.52
200	4	128	49.5	51.2	**60.62**
200	2	128	49.2	51.0	60.11
152	8	128	49.67	51.35	61.37
96	8	128	50.26	51.06	59.10

We systematically explored hyper-parameter settings of our model to improve performance. Specifically, we investigated how the performance of the QANet architecture changed by varying the number of heads used in multiheaded attention, the dimension of representation for character embeddings, and the hidden size used in the majority of the model. We were interested in the effect on evaluation metrics and on training speed and memory efficiency. Notably, we could only run our hyperparameter experiments for 5 epochs (which is why the EM, F1, and AvNA scores are low).

4.3.3 Answerability Extension: Implementation of EQuANT

After running our hyper-parameter experiments, we built an additional layer on top of our QANet paper based off of the answerability module described in the EQuANT model by Aubet et al [3]. The implementation details are described in our approach section, along with how we modified the loss function to incorporate our answerability scores. In order to check our scores against the threshold of $p = 0.5$, we implemented a custom discretize function based on the baseline function provided in `utils.py`. We performed a full run with this model using 96 as our character embedding size instead of 128 in order to follow the lead of Aubet et al. paper [3], as well as the hidden size as 96 in a similar vein. However, after training our model for 15 epochs, we found that the EM and F1 scores were worse than our original QANet implementation.

4.4 Results

Model	EM	F1
Baseline BiDAF	55.92	59.73
QANet	**53.34**	**56.74**
EQuANT	52.51	55.87

The table above summarize our results on the SQuAD test set. In Figures 8 and 7, we also see the longitudinal dev set performance for all three models. As our QANet model outperformed out EQuANT, we submitted an **F1 score of 56.74 and EM score of 53.34** to the IID Squad Track leaderboard. Notably, our test set scores were lower than our dev set scores: we achieved a **F1 score of 57.56 and EM score of 54.66** on the dev set. The difference in the performance could be attributed to the natural variance in scores because the dev set is much smaller than the test set because of the differences in data distribution.

5 Analysis

5.1 Initial Decrease in Dev Performance

We observed in the initial <150k training steps of QANet that the F1 and EM scores drop despite decreasing loss. The textual predictions on the dev set during this period reveal that the model initially chose to optimize by only predicting "no answer" since roughly half of the data is unanswerable yielding around 50% AvNA.

A similar experiment with low evaluation steps showed that the EQuANT model yielded single digit metrics on the dev set initially. While the traditional QANet architecture yields only probabilities for the answer span, the EQuANT architecture explicitly provides an answerability probability. A model with untrained parameters will yield near random outputs, notably resulting in an approximately uniform distribution across the context for the span selection and randomly predicting answerability.

As such, while the QANet architecture will initially struggle to find start and end probabilities that exceed the dynamically selected threshold and therefore predict "no answer" frequently, EQuANT will effectively randomly predict that a question is answerable given the context and therefore select an effectively random span from the context as the answer leading to drastically worsened F1 and EM scores.

5.2 Memory and Processing Efficiency

The purported benefit of the QANet architecture is the replacement of recurrent blocks with convolutional blocks for encoding. While recurrent blocks operate sequentially, convolutional blocks do not feature the inherent dependence on previous state and are therefore more readily parallelizable for efficient processing. The increase in performance would allow for more data to be processed given the same compute resources and time. However, our implementation of QANet runs considerably slower than the baseline BiDAF. We consider two potential explanations and avenues for improving performance relative to BiDAF.

Firstly, the BiDAF code uses packed sequences when processing the data. Batching examples for simultaneous training allows for more efficient processing on parallelized systems like a GPU, though doing so with sequences of variable length poses a representational challenge. As such, sequences are padded to the maximum sequence length in the batch, allowing contiguous memory to be transferred to the GPU. However, this means that the kernels on the GPU may process large quantities of factitious "pad" data resulting in wasted computation. As such, packing the sequence after transfer to the GPU prevents computation on the pad data, yielding improved performance. In comparison, our implementation directly processes the padded data yielding sub-optimal performance, so packing the padded sequences before passing the data through our layers may improve performance.

Secondly, while training various models, we monitored our GPU utilization and observed that while it occasionally spiked to near full utilization, it would quickly drop such that our average utilization was 65% (Figure 4). This suggests that our implementation was not compute bound on the system, but rather potentially memory bandwidth bound. Therefore, more frugal use of memory transfers on the GPU system may improve performance. One solution to loosen the memory bottleneck could be to use more in-place operations, though doing so would require careful analysis as our implementation of residual connections, for example, leverage the use of reassigning references for more manageable code and would break with in-place operations. Additionally, fusing the kernels defined by our layers together and utilizing shared memory on SMs to reduce transfers to and from memory would improve GPU utilization given that the training is bandwidth bound, though doing so would require developing custom low-level kernels.

Reducing memory usage overall would also yield improvement. Specifically, by reducing RAM usage, training could be performed with a larger batch size, since we observed that training with a batch size greater than 16 would quickly lead to memory usage that exceeded the capabilities of our system.

5.3 Data Augmentation

These optimizations for faster and more efficient training enable the augmentation of training data given the same compute and time resources. One notable example of this would be using backtranslated data where examples are translated into a series of target languages before being translated back into the source language. This backtranslation method increases the syntactical variation in examples and augments the training data. While performing backtranslation with data-rich language pairs such as English-French are common, the potential linguistic connections of these languages may limit the introduced variation. As such, using linguistically dissimilar languages such as English to a non-Romance language may yield increased variety. Other data augmentation techniques may be applied such as replacing tokens based on thesaurus synonyms or a nearest neighbor by word embeddings.

5.4 Architectural Improvements

Our model uses BiDAF Attention Flow instead of Dynamic Coattention Network. While both involve two-way attention between the context and the question, DCN involves a second-level attention

computation - i.e., attending over representations that are themselves attention outputs. However, we were discouraged of pursuing DCN as a possible extension because of the original QANet paper [1] that mentions that the use of DCN over BiDAF Attention Flow does not increase performance significantly.

We instead consider two ways to improve the output layer. Firstly, the EQuANT answerability module outputs a probability estimate that is then thresholded statically to determine the answerability of the question given the context. Though the threshold may be tuned post training to optimize F1 (or any metric) on the dev set, this static thresholding method is inflexible compared to the baseline answerability prediction. Consider the predicted span probabilities obtained from the softmax function of a short unanswerable example and a those of a long answerable sequence. Based on these values alone, we may artificially see uniformly higher probabilities in the first example while lower (though not uniform) values in the second. Since the answerability output is derived from the first model encoder, the result may incorporate these artificial differences into the output. One avenue for exploration may be to dynamically select the threshold based on the distribution of span probabilities. Another possible exploration could be in incorporating sequence length as an explicit feature in the answerability block to mitigate differences introduced by the sequence length.

Secondly, the span output prediction yields two probability distributions that are independently created from the model encoder layer. Instead, probabilities for the start index may be computed directly and probabilities for the end index may be computed conditionally on the start vector. This formulation seems more natural than creating the two independently and may yield improved performance.

6 Conclusion

In this project, we explored the QANet architecture and its performance on the SQuAD 2.0 dataset. We first reimplemented QANet from scratch by taking heavy inspiration from the original paper [1]. Unfortunately, we were not able to achieve better results than our baseline BiDAF, even after we experimented with tuned hyperparameters. As QANet was first created with SQuAD 1.0 in mind, we decided to introduce a new answerability module so our model would work better on SQuAD 2.0. This new model, EQuANT, also was not able to improve on our baseline performance. As our models were not successful in the conventional sense, we conducted a full analysis of possible errors and fixes to our implementation. The motivation and power of the QANet model stems from the feed forward nature of the architecture which allows for more efficient processing and therefore the inclusion of augmented data during training. A more memory efficient implementation may allow for a larger batch size during training, which again could make training with augmented data feasible. Additionally, we considered architectural variations for future work to improve performance such as attention mechanisms, improved feature sets for explicitly predicting answerability, and conditioning the end index probabilities on the start index probabilities. While we were not able to see the full power of QANet with respect to convolutions and self-attention, we learned about the importance of implementation details and real-time result analysis.

7 Acknowledgements

We would like to thank Chris Manning, the CS 224N course assistants, and especially Andrew Wang for their guidance and particularly helping us approach solving exploding gradients during our models training.

References

[1] Adams Wei Yu, David Dohan, and Minh-Thang Luong. Qanet: Combining local convolution with global self-attention for reading comprehension. In *International Conference on Learning Representations*, 2018.

[2] Ali Farhadi Min Joon Seo, Aniruddha Kembhavi and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. In *CoRR, abs/1611.01603*, 2016.

[3] Dominic Danks Francois-Xavier Aubet and Yuchen Zhu. Equant (enhanced question answer network). In *CoRR, abs/1907.00708*, 2019.

[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 *CoRR, abs/1706.03762*, 2017.

[5] Zhuang Liu Daniel Sedra Gao Huang, Yu Sun and Kilian Q. Weinberger. Deep networks with stochastic depth. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, 2016.

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

[7] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR*, abs/1901.02860, 2019.

A Figures

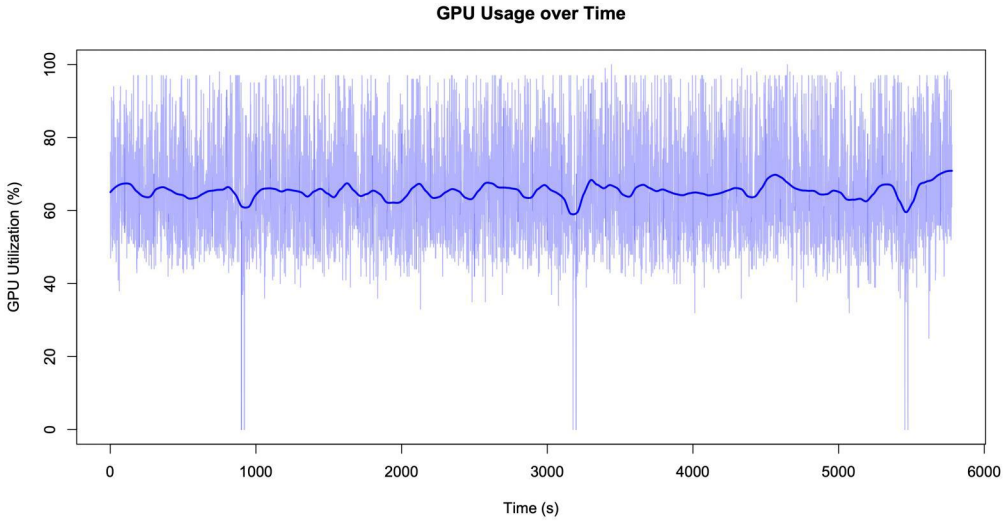

Figure 4: (Smoothed) GPU usage over time, frequent jumps between high and low utilization may suggest memory bandwidth issues

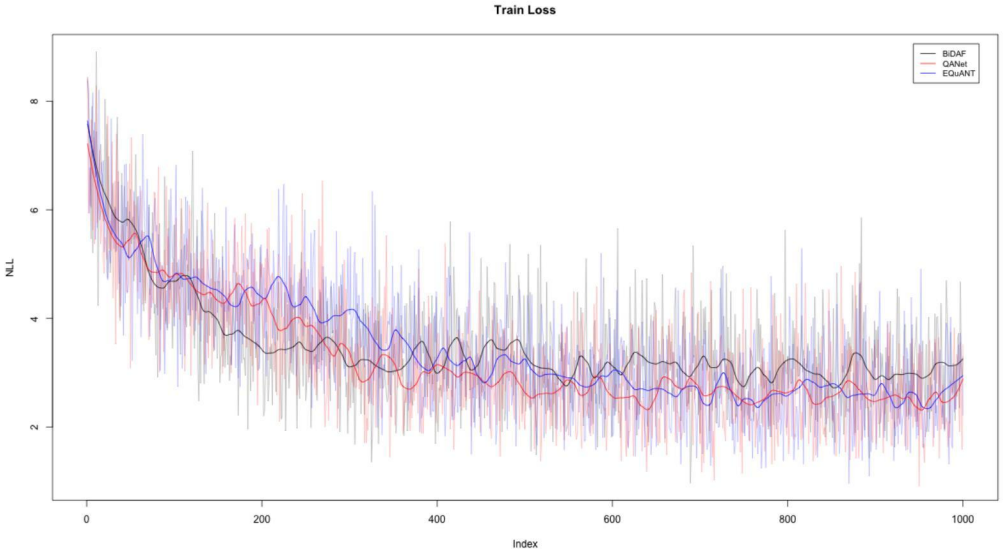

Figure 5: Training loss (NLL) of select models

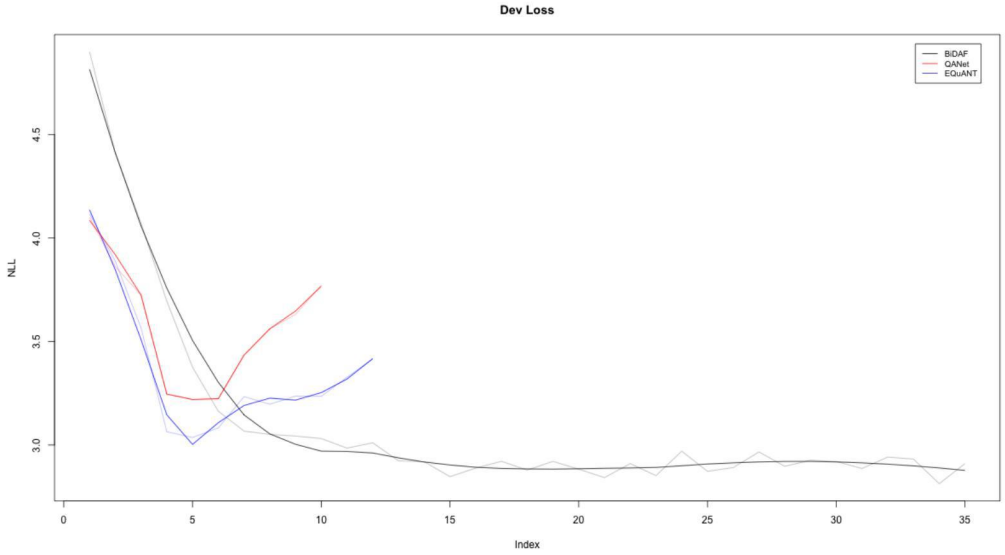

Figure 6: Dev loss (NLL) of select models

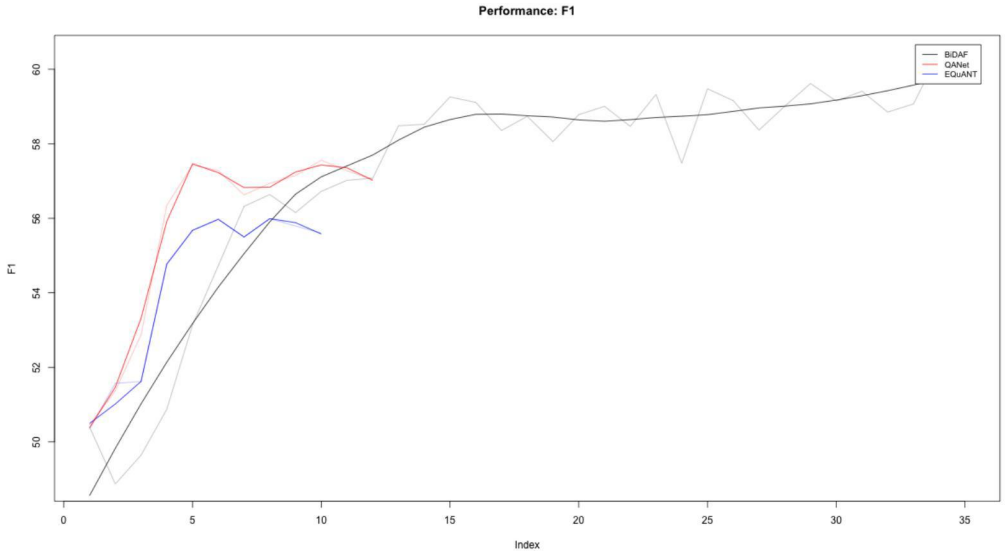

Figure 7: Performance (F1) of select models

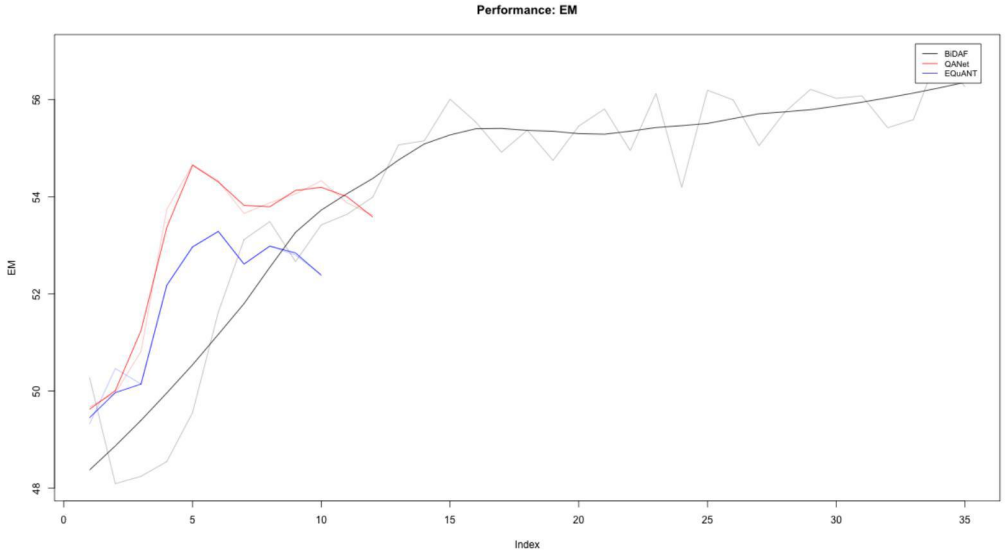

Figure 8: Performance (EM) of select models

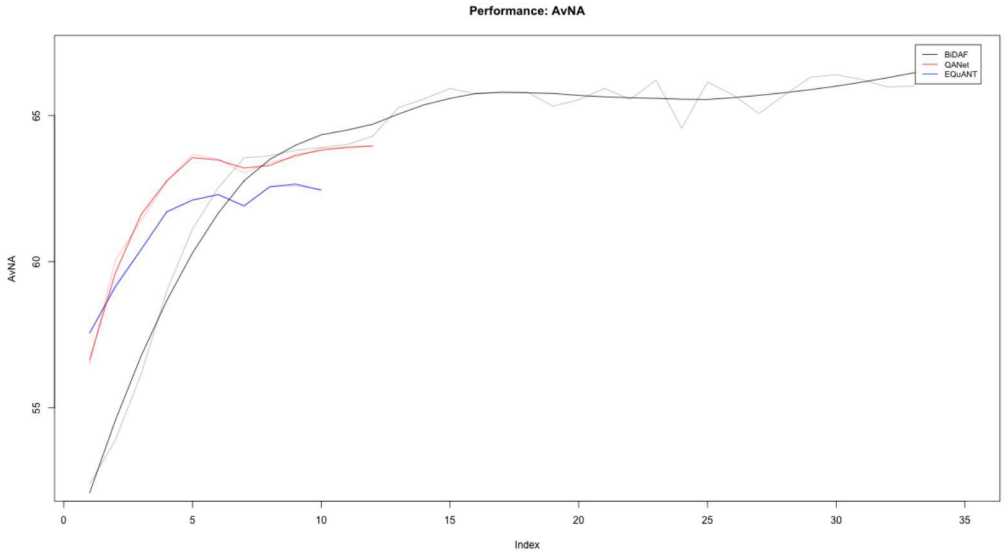

Figure 9: Performance (AvNA) of select models