Default Project: QANet for SQuAD 2.0

Anuprit Kale
Stanford University
anuprit@stanford.edu

Edgar G. Velasco
Stanford University
edgarv1@stanford.edu

Abstract

The Stanford Question Answering Dataset(SQuAD)[10] has spurred many novel architectures for extractive question answering, but there still remains the need for models which can handle unanswerable questions and that can train and predict efficiently. One such efficient architecture is QANet[13] which replaces RNNs with Transformer-like encoder blocks. We implement this architecture and improve its performance beyond a well tuned baseline. We do this by tweaking the architecture, incorporating some tag features as mentioned in DrQA[2], experiment with augmented loss functions, and by performing data augmentation through back translation. We also experimented with variants of local attention. We achieve a test F1 of 63.8 and a dev F1 of 68.0 with a single model in the non-PCE category.

1 Introduction

SQuAD has been a huge boon to NLP research in the domain of extractive question answering. Many methods now exceed human performance on SQuAD 1.1, but there is still much to improve on for true language understanding in QA. In this task we take on SQuAD 2.0 [9], which adds unanswerable questions to the mix. This is more challenging since the model must have a higher level of understanding to abstain from answering. It is not enough to look for passages that contain similar phrases or look for specific entities. As a result of this extra challenge, models on SQuAD 2.0 are still below human performance. In addition, there are additional challenges in training these models. Training these models from scratch requires lots of data and compute. This motivated us to look into methods avoiding recurrence and were intrigued by QANet which avoids RNNs completely and did some clever data augmentation which inspired our own approach.

2 Related Work

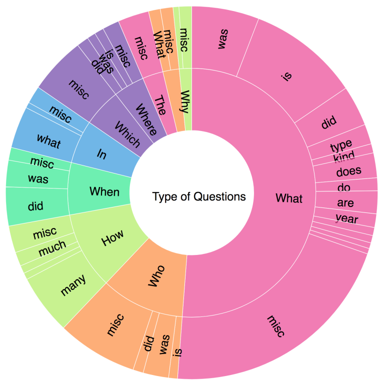
Given our interest in non-recurrent methods and their success in the leaderboards, we were intrigued by methods which utilized Transformers[12] such as the original paper itself and BERT[3]. We also wanted to implement the QANet architeccture from scratch to see how it would perform on SQuAD 2.0. We looked to recurrent methods that performed well on SQuAD 2.0 as well for inspiration for additions to our QANet implementation. In particular, Stochastic Answer Networks (SAN)[7] experimented with a joint loss function that also tries to do binary classification of whether the question is unanswerable or not. We experimented with this idea in QANet and also their addition of tag features such as POS, dependency, and entity tags.

3 Approach

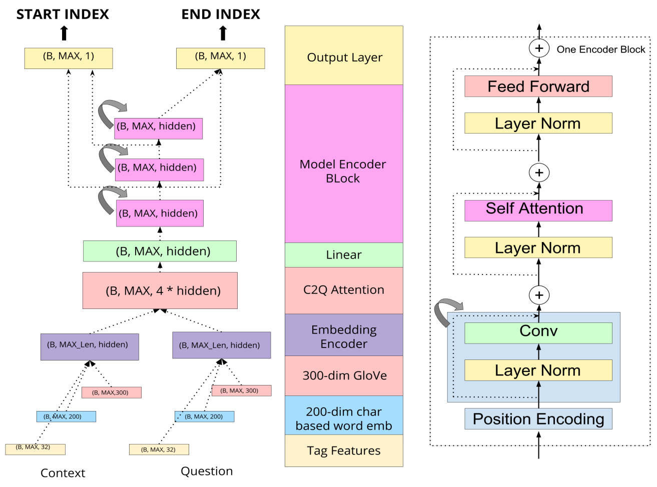
We begin by comparing the provided baseline with an improved baseline which adds character-embeddings to the embedding step. In addition, we start with some preliminary hyperparameter searches for learning rate decay and dropout. After this we implemented the QANet architecture

.

baseline and increased the score up to 68.05 on the dev set with the addition of features, better hyper-parameters, and slight architecture tweaks. We also generated question paraphrases by translating the question into French and then back to English. This approach is different than mentioned in QANet and is something new we tried. Finally, we modified the self-attention architecture to use variants of local attention instead of global attention and have gotten encouraging results thus far.



(a) SQuAD 2.0 major types of questions



(b) Model architecture

Figure 1

3.1 Architecture

QANet is a model architecture that was designed to be efficient by utilizing an encoder block (Fig. 1b). This block is described below and is used by many of the five total layers of the architecture.

Encoder Block An encoder block consists of three types of residual sub-layers stacked upon each other and preceded by the addition of positional embeddings to the input of the block. We used the HarvardNLP implementation of the sinusoidal positional embeddings. After the addition of the positional embeddings, every encoder block starts with N repetitions of a convolutional residual sublayer. It is then followed by a residual self-attention sublayer where self-attention functions exactly like the multi-head attention in the Transformer architecture. Finally, there is a Feedforward Network(FFN) residual sublayer which also functions exactly as in the Transormer architecture. Each sublayer begins with Layer Normalization[1] which is similar to Batch Norm but instead normalizes across the features in each example. This has been shown to work well in sequential settings. The hidden size for each layer and sublayer is H = 128, the max length is L=600, the batch size is B=32, and dropout is applied between the five layers described after the sublayers.

1. **Convolution Residual Sublayer** This sublayer performs 1D convolutions on the input of shape (B, H, L) over the third dimension. We use a kernel of size seven and four in the embedding encoder and model encoder blocks, respectively. The number of filters is H = 128 for all convolutional sublayers.
2. **Self Attention Residual Sublayer** We implement multi-head self attention in a vectorized fashion as is typical for transformers. In this sublayer we make three copies of an input $X \in \mathbb{R}^{B \times L \times H}$ which we use to compute the queries, keys, and values of the input. This is done by multiplying the three input copes by three linear layers

$$W_q, W_k, W_v \in \mathbb{R}^{H \times H}$$

and reshaping to get

$$Q, K, V \in \mathbb{R}^{B \times L \times D \times N}$$

where

$$N = 4$$

is the number of heads and

$$D = H/N = 128/8 = 32$$

is the dimension of the heads. From here we use the following attention equation,

$$Attention(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{D}})V = HEAD \in \mathbb{R}^{B \times L \times D \times N}$$

to compute attention for each head in a vectorized way. Then we reshape, to concat the different heads together so that

$$HEAD \in \mathbb{R}^{B \times L \times ND}$$

. Finally, the multi-head attention is computed by multiplying this concatenation with a linear layer as follows

$$Multihead(Q, K, V) = HEADW_o$$

where

$$W_o \in \mathbb{R}^{H \times H}$$

3. **FFN Residual Sublayer** This sublayer computes a position wise feed-forward network following this equation

   $$FFN(X) = ReLU(XW_1 + b_1)W_2 + b_2$$

   on the input X of shape (B, L, H).

Here we describe the 5 major layers of QANet:

1. Input Embedding Layer. We use pre-trained 300-dimensional GloVe[8] word vectors. We also learn 200-dimensional word embeddings using a char-CNN. The two embeddings are concatenated and passed through a projection layer of hidden size 128 and 2 highway layers.

2. Embedding Encoder Layer. This layer is composed of one encoder block with four convolutional sublayers. The weights are shared between the question and context input embedding layer in a Siamese fashion.

3. Context-Query Attention Layer. As described in the default project handout and BiDAF paper[11], we first calculate the similarity matrix S between a context and a question pair. Given the outputs of the embedding encoding $c_1, ..., c_N \in \mathbb{R}^H$ and $q_1, ..., q_M \in \mathbb{R}^H$ for the context and question, we compute

$$S_{ij} = w_{sim}^T[c_i; q_j; c_i \odot q_j] \in \mathbb{R}$$

We then take row-wise softmax of this matrix to learn attention weights for each of the context token.

$$\bar{S}_{i,:} = softmax(S_{i,:}) \in \mathbb{R}^M \; \forall i \in (1, ..., N)$$

We then take a weighted sum using these weights with the hidden states to obtain context-to-query attention.

$$a_i = \sum_{j=1}^{M} S_{i,j} q_j \in \mathbb{R}^{2H} \; \forall i \in (1, ..., N)$$

QANet performs Query to Context attention in the same fashion as the BiDAF model. Namely, we get the following weighted sum of the context embedding encodings $c_1, ..., c_L$:

$$\bar{\bar{S}}_{:,j} = softmax(\bar{S}_{:,j}) \in \mathbb{R}^{\mathbb{N}} \; \forall j \in (1, ..., M)$$

$$S' = \bar{S}\bar{\bar{S}}^T \in \mathbb{R}^{N \times N}$$

$$b_i = \sum_{j=1}^{N} S'_{i,j} c_j \in \mathbb{R}^{2H} \; \forall i \in (1, ..., N)$$

We use the computed a_i, b_i, c_i to generate g_i :

$$g_i = [c_i; a_i; c_i \odot a_i; c_i \odot b_i] \in \mathbb{R}^{8H}$$

We then pass the BiDAF output through a linear layer to project the feature space down from 8 * hidden size to hidden size.

4. Model Encoder Layer. This layer is composed of seven encoder blocks with two convolutional sublayers in each block. This same layer is repeated three times consecutively with shared weights.

5. Output Layer. There are two output layers corresponding to predictions for the start and end span. The start output layer takes as input the output of the 1st and 2nd model encoder outputs. The end output layer takes as input the 1st and 3rd model encoder outputs. Each output layer takes the two specified inputs and computes

$$p_o = softmax(W_o[M_x; M_y])$$

where x and y specifies the index number of which model encoder layer outputs to concatenate.

4 Experiments

4.1 Data

The SQuAD 2.0 dataset provided was used for training and evaluation. The training set contains 129,941 samples. The dev and test set contain 6078 and 5915 samples, respectively. The dev and test set contain approximately equal number of answerable and unanswerable questions. We used the pre-processing scripts provided in default repository with modifications for augmented data and feature generation. In Fig. 1.a, a breakdown of the major question types can be seen for SQuAD 2.0. The dataset contains questions starting with *What, Who, How, When, In, Which, Where, Why* among other words. The majority of questions begin with *What*.

4.2 Evaluation method

We used the F1 score between predicted answer and the three potentially correct answers tokens as our primary metric. EM score is the proportion of predicted answers which exactly match one of the 3 potentially correct answers as the secondary metric. These two metrics have been widely used for evaluation on SQuAD 1.1 and 2.0 datasets by researchers.

4.3 Experimental details

For the baseline, we used the Adadelta[14] optimizer with a constant learning rate of 0.5 and weight decay of 1e-3. Adadelta adapts learning rates based on a window of past gradients. We didn't experiment with the learning rate as it is recommended to leave the parameters for this optimizer at the default values. Batch size was set to 64. The augmented baseline model took about 7 hours to train 40 epochs on a single NVIDIA Titan V GPU.

For the QANet model, We used kernel sizes of 7 and 5 for the CNNs in the embedding encoder layer and model encoder layer, respectively. We set the number of filters to be our hidden size of 128. After experimenting with a smaller hidden size of 96 we reverted to 128 throughout due to better performance as recommended in the QANet paper. Since this model was tricky to get working we stuck with the original training parameters to perform optimization using ADAM[5] with $\beta_1 = 0.8, \beta_2 = 0.999, \epsilon = 10^7$. We also use inverse exponential learning rate warmup for 1000 steps up to 1e-3 and remained constant there for the remainder of training. We also apply an Exponential Moving Average on the parameters with a decay rate of 0.9999.

QANet Experiments The bulk of our time was spent trying to improve our QANet performance and performing ablation studies. We explored several enhancements or changes to try to boost the score. Our progression is described below.

Original Question	Paraphrase
When did Beyonce become popular?	When did Beyonce start becoming popular ?
What did the officials order the Chinese media to stop broadcasting?	What did officials order Chinese news media to stop reporting ?
What can be used to shoot without manually targeting enemies?	What can be used to shoot without the need to manually target enemies ?

Table 1: Generated paraphrases of questions using backtranslation

Stochastic Depth (Layer Dropout)[4] The paper describes using this technique to "shut off" sublayers with an encoder block. Recall that within an encoder block there are N convolutional sublayers and then the self-attention and FFN sublayers. Let $L = 2 + N$ be the number of total sublayers in a given block. Then, within each individual block throughout this architecture the sublayer at depth l "survives" according to the following probability equation:

$$p_l = 1 - \frac{1}{L}(1 - p_L)$$

where $p_L = 0.9$ is the survival probability of the final layer.

Changing the Projection Size in the FFN Sublayer One change which lead to a large increase in Dev Set performance was changing the projection size in the FFN layer from 4*hidden size to hidden size. Concretely, we use weight matrices $W_1 \in \mathbb{R}^{H\times H}$ and $W_2 \in \mathbb{R}^{H\times H}$ instead of what is used in transformers, $W_1 \in \mathbb{R}^{H\times 4H}$ and $W_2 \in \mathbb{R}^{4H\times H}$. These hyperparams are not mentioned in the paper so it was up to us to discover this.

Data Augmentation by Backtranslation We took a slightly different approach than described in the QANet. Instead of creating paraphrases for context we created paraphrases for questions. The intuition behind it is that there can be multiple ways of asking the same question leading to the same answer. Using Google translate APIs the questions were first translated to French and then back to English. Some of the examples can be seen in Table 1. Data augmentation gave us an increase of 0.8% in F1 score on the dev set. Data augmentation also prevented the dev loss from increasing and instead it started plateauing.

Tag Features Inspired by SAN and DrQA, we decided to add some POS, dependency, and entity tags to input. These are incorporated by converting the tags to indices and then creating embedding matrices for each of the types of tags. The embedding size for each are 16, 8, and 8, respectively. These embeddings are integrated within the Input Embedding Layer. Previously, we only concatenated the word embeddings and the word char-CNN embedding, but now we further concatenate these additional embeddings to get a 540 feature dimension size for each token in the sequence. We adjust the project layer following this accordingly and subsequently pass through a highway network as before.

Augmented Loss function Inspired by SAN we experimented with an augmented loss function which not only takes into account the NLL loss of the start and end span but also adds a BCE loss term for binary classification of whether the question was predicted to be unanswerable or not. This loss is then given by:

$$L_{joint} = L_{span} + L_{classifier}$$

In practice, this helped to speed up training especially at the start but did not improve or hurt performance overall.

Ablation Study We performed an ablation study(Table 2) on a fully integrated QANet model with layer dropout, data augmentation, tag features, augmented loss, a hidden size of 128, the hidden size projection space in the FFN sublayer, and 4 attention heads in the self-attention sublayer. Additionally, dropout was 0.1 after each layer and on the word embeddings. The char-embeddings received a

Component	F1	EM	AvNA
Full Model	67.15	63.65	74.07
Aug. Loss	67.39	63.91	74.51
Tag Features	65.79	62.22	72.12
Char. Embeds	58.1	56.27	66.29
Data Aug.	67.72	64.37	74.62

Table 2: Ablation study on added components to the QANet model

dropout of 0.05. We remove some select components to see its effect on the model. Removing char. embeddings had the largest effect followed by the tag features. It was surprising to see how little the other additions changed the model.

Local Attention Experiments Self-attention is a compute intensive task and is quadratic in time complexity. For the SQuAD task, context size is relatively small and hence it doesn't have a huge impact on the performance. But for newly released dataset like Natural Questions (NQ)[6] by Google, the input is the whole Wikipedia article. The slowdown will further increase if character embeddings are used. To optimize the self-attention, we first changed it to naive local attention by splitting the context sequence into K sub-sequences and performing self-attention individually on each sub-sequence. In our experiment, we found that naive local attention gave a F1 score of 0.75 on Squad 1.1 dataset on pre-trained BERT models. As seen in Fig 2., to enable the information flow across K context sequences, we implemented an *offset attention*. To add an offset padding was appended to every alternate layer of self attention. This increased the F1 score to 0.80. For offset attention most context blocks are unaware of the query. We believe the score can further be increased by learning a query representation and adding it to context sequences as this will enable the information flow from question-to-context as well as context-to-question. We observed a speedup of 12% in training time per iteration for max context length of 384. We think that as the context length increases this speedup should increase significantly. We were not able to integrate this into our QANet model due to time constraints but it is a promising direction.

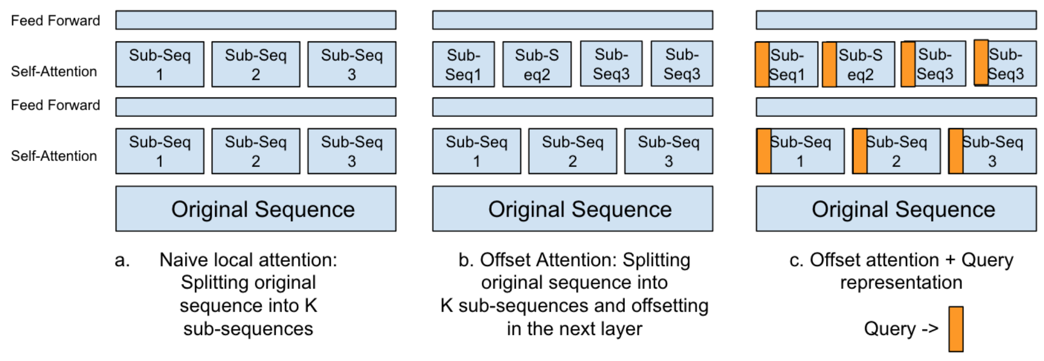

Feed Forward
Self-Attention
Feed Forward
Self-Attention
Sub-Seq 1
Sub-Seq 2
Sub-Seq 3
Original Sequence
a. Naive local attention: Splitting original sequence into K sub-sequences
Sub-Seq1
Sub-Seq2
Sub-Seq3
Sub-Seq3
b. Offset Attention: Splitting original sequence into K sub-sequences and offsetting in the next layer
c. Offset attention + Query representation
Query ->

Figure 2: Showing modifications to encoder/decoder layers in the Transformer architecture

4.4 Results and Analysis

BiDAF Baseline The baseline provided for the default project performed at 0.60 and 0.59 for F1 and EM scores. After adding character embeddings and some hyperparameter tuning the F1 and EM scores jumped to 0.64 and 0.61 respectively. After using weight decay of 0.001 the dev loss decreased consistently as seen in Fig. 6.a. This makes sense as the weight decay penalizes large weights and acts as a regularizer. Three values of dropout were tried and the dropout of 0.1 gave better performance than 0.2 and 0.3 as seen in Fig. 6.b. The model only achieved an F1 score of 0.52 across unanswerable questions. We found that character embeddings increased F1 score of answerable questions 0.15. But interestingly, the performance of unanswerable questions increased by 0.6 (Fig. 6.c).

QANet Our best QANet test set submission received a test score of 63.841 F1 and 60.389 EM which was disappointing given that we received scores of 68.05 F1 and 64.561 EM on the dev set with no significant signs of over-fitting. We speculate that the discrepancy might be due to a small vocab. size or an inability to generalize to larger sequences since the test set has a max context size of 1000 whereas the dev set had a default max context size of 400.

Qualitative Analysis Fig. 3. shows context to query attention. There are mentions of multiple places in the context but *overseas colonies*, which is the right answer, is activated more than actual places like *Vietnam* and *colonies in Africa*. In appendix Fig. 6, an attention plot for an unanswerable question can be seen. In this case the question is asking about a mall north of downtown Ann Arbor but the context is about what's south of Ann Arbor. The model fails to understand this information and predicts an answer span. While doing error analysis we noticed a lot of out of vocabulary words on the dev set. Thus, we increased the size of the vocabulary from 90K to 2M to try to fix the problem.

Quantitative Analysis We analyzed the results of our top model on the dev set to see which of the major question types (as determined by the starting word in the question) it performed well on. You can see the full results on Fig. 4c. As expected, the model performed best on "when" questions since it is more straightforward to seek date or time entities in the text. The model achieves scores of 71.5 F1, 69.47 EM and 78.68 AvNA on this segment of questions. The worst performing questions are the "why" questions which may indicate a lack of reasoning ability by the model. The scores here are the lowest for F1 and EM with values of 59.38 and 49.36, respectively. Somewhat surprisingly, this segment did not have the lowest AvNA score. Instead, "in" questions performed worst on that metric. Keep in mind that "why" is one of the smallest categories with only 88 out of 6078 dev questions falling into this bucket."What" questions account for the largest group with 2681 questions falling into that group. In Fig. 4d. you can see the QANet model performance by context length. In the dev set the max length was 400 and we did not see a performance degradation for large contexts, but it remains to be seen whether that would be the case in the test set where the max length was 1000.

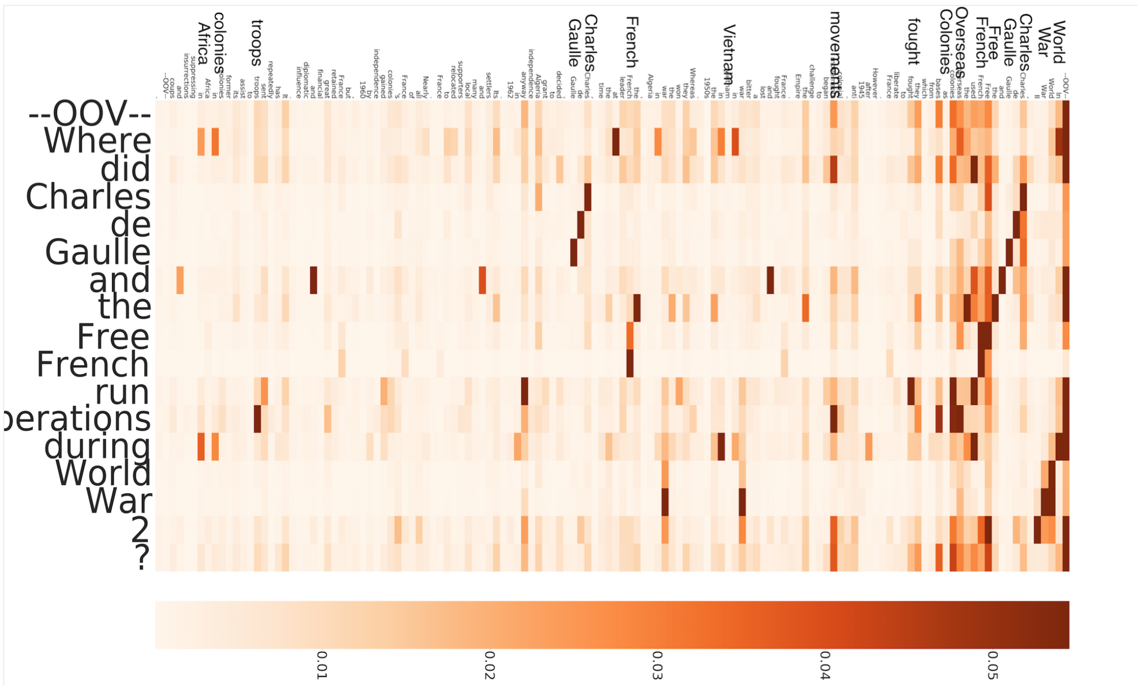
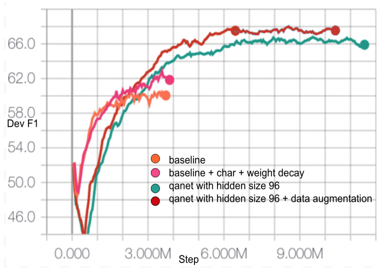
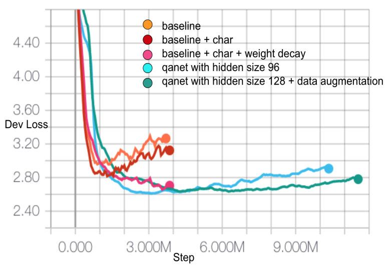


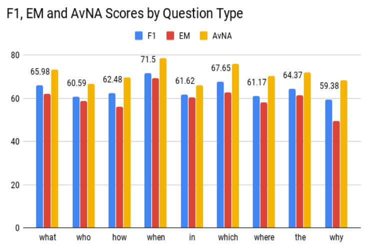
Figure 3: Context to Query Attention visualization from the QANet model for answerable question



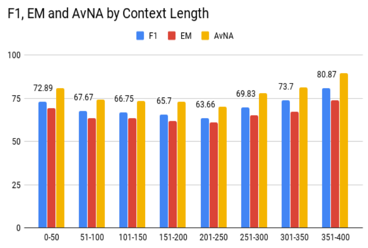
(a) Dev F1 scores vs steps



(b) Dev Loss vs steps

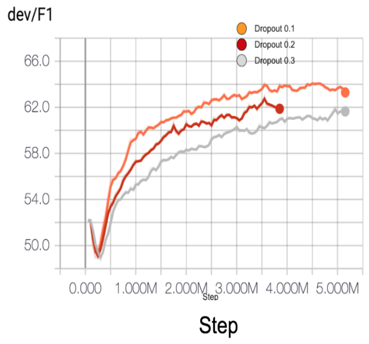


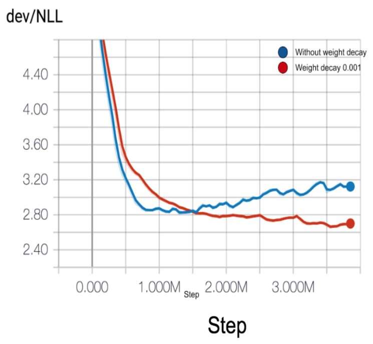
(c) Performance by Question Type



(d) Performance by Context Length

Figure 4: a. F1 score for different models. b. Showing dev loss for different models. c. Performance by question type for top QANet model. d. Performance by context len for top QANet model.





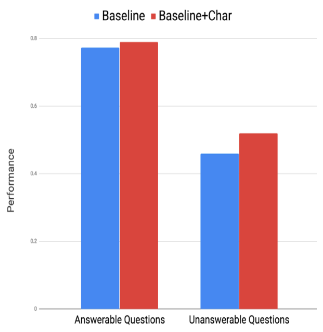


Figure 5: a. Dev F1 vs steps for different values of dropouts. b. Dev loss vs steps for different values of weight decay c. Performance of two models across answerable and unanswerable question types

Conclusion/Future Work

We showed that QANet can perform above a well tuned baseline for the SQuAD 2.0 task and our single model reaches a score of 63.8 and 68.05 F1 on the test and dev leaderboard. Given more time we would train a model on longer context length and apply more regularization techniques to check if that reduces the gap in our dev and test F1 scores. We believe that the transformer block can be made more efficient since, in practice, we did not experience speed-ups in training compared to the baseline. The model has difficulty with reasoning questions such as "why" questions. In addition, unanswerable questions which repeat multiple tokens from the passage confuse the model into providing a wrong answer where it should return none, since it lacks language understanding such as common sense.

We would like do more experiments with modifying the self attention mechanism for efficiency gains and incorporate them into QANet. Our local attention experiments were promising as we demonstrated a speedup and information flow in local attention using offset for BERT fine-tuning, but did not have time or compute to integrate these into QANet. Lastly, it may be premature to completely do away with recurrent models. One of our guest lectures remarked that using RNNs before or after Transformers helps with training and perhaps performance as well. We would hope to explore this as well.

Mentor

We want to thank Kevin Clark from the NLP group at Stanford. Kevin mentored us and gave ideas about how to modify self attention.

References

[1] Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016.

[2] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. *CoRR*, abs/1704.00051, 2017.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[4] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. *CoRR*, abs/1603.09382, 2016.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[6] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina N. Toutanova, Llion Jones, Ming-Wei Chang, Andrew Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: a benchmark for question answering research. *Transactions of the Association of Computational Linguistics*, 2019.

[7] Xiaodong Liu, Wei Li, Yuwei Fang, Aerin Kim, Kevin Duh, and Jianfeng Gao. Stochastic answer networks for squad 2.0. *CoRR*, abs/1809.09194, 2018.

[8] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *In EMNLP*, 2014.

[9] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *CoRR*, abs/1806.03822, 2018.

[10] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016.

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

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

[13] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le. Qanet: Combining local convolution with global self-attention for reading comprehension. *arXiv preprint arXiv:1804.09541*, 2018.

[14] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

Appendix

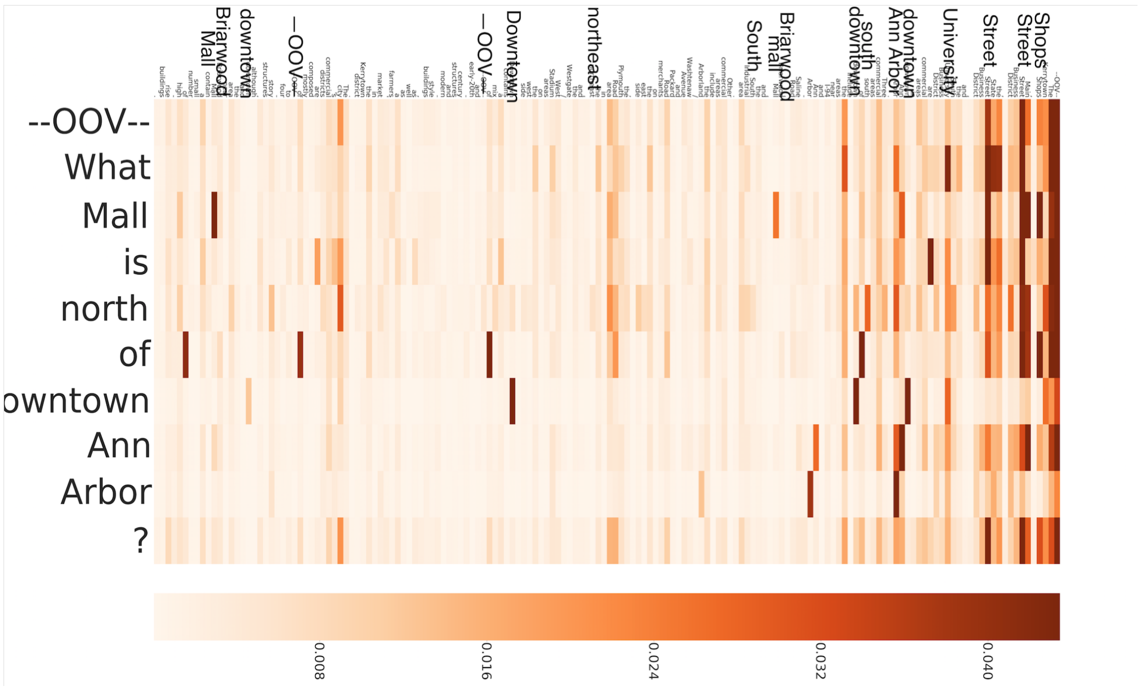


Figure 6: Context to Query Attention visualization from the QANet mode for unanswerable question