Integrating the QANet Structure into the Default Model BiDAF for Question Answering

Stanford CS224N Default Project

Pham Thanh Huu
Department of Computer Science
Stanford University
huupt@stanford.edu

Abstract

This project addresses the problem that the default BiDAF machine comprehension and question answering model is RNN based and thus slow as it is not parallelizable. This project aims to integrate the QANet model, which uses convolutions and self-attentions to form a model architecture that is faster than recurrent-based approaches, into the default BiDAF model to enhance the baseline scores by tuning existing baseline model hyperparameters and changing the model architectures by adding new layers. Findings were variable, with some successes in improving the score and some failures in testing. The best model, BiDAF (with Character Embedding), achieved 63.646(F1) / 60.254(EM), while two other variants of the QANet integrated model were 47.811(F1) / 47.811(EM) and 54.708(F1) / 51.784(EM) evaluated on the course test set.

1 Key Information to include

- Mentor: Lucia (zlucia@stanford.edu)
- External Collaborators (if you have any): N/A
- Sharing project: N/A

2 Introduction

In [1], Adams et al. proposed QANet, a question-answering model based on convolutions and self-attention instead of recurrent neural networks. At the time of publication of the paper (Apr 23, 2018), the most successful question-answering models were generally based on recurrent neural networks (RNNs) with some attention mechanisms. One weakness of these models is that their recurrent nature prevents parallel computation, making training and inference slow. There were efforts to speed up the RNNs by avoiding bi-directional attention [2] or deleting the context-query attention module [3], but these models had to sacrifice accuracy: On the SQuAD v1.1 dataset, their F1 scores were around 77.

QANet solves this problem by moving away from RNNs instead of using convolutions and self-attention as the main building blocks. This feed-forward nature gives the model a significant speed advantage (reportedly between 3 to 13 times faster in training and 4 to 9 times in inference) over its RNN counterparts. Taking advantage of this speed boost, the authors trained the model with augmented data and achieved an F1 score of 84.61 on the SQuAD dataset, which was significantly better than the best-published result of 81.8 at the time.

QANet on PyTorch was implemented in this project and tested on SQuAD 2.0. While the transition to SQuAD 2.0 is straightforward, it is challenging to reproduce the performance, especially the speed, reported in the original paper. A number of the author's peers in the CS224N class and open Github repositories reported the same issue. While the issue was not resolved, a few factors which might play an essential role in the model's performance were identified. These are outlined in section 5.3 Experimental details.

3 Related Work

QANet took inspiration from previous studies such as BiDAF [4]. Before BiDAF, most architectures in this space used a unidirectional attention mechanism in which the query attended to the context. BiDAF introduced bidirectional attention from context to query and query to context. Further, to reduce information loss caused by intermediate representations, BiDAF's attention is not used to summarize the context into a fixed-length vector but is instead computed at every step, and the attention vector and previous layer representations are allowed to propagate to further modeling layers. As described above, QANet departs from BiDAF in that the encoder blocks have no recurrence but rather just convolutions and multi-head self-attention. This is similar to the Transformer paper with convolutional layers [5]. Transformers were first introduced using just self-attention and feed-forward blocks to model natural language phenomena. The authors introduce the concept of multi-head attention used in QANet. Instead of performing a single attention function, the authors found that by computing the attention 'h' times, the model could jointly attend to information from different representational subspaces at different positions. Since QANet has no concept of recurrence, it needs to encode some form of positional information. This concept is also inspired by the Transformer paper, in which the writers use sin and cosine functions of different frequencies. The positional embeddings have the same hidden model dimension to be summed. These positional embeddings are more suited than learned positional embeddings because they can extrapolate to longer lengths.

4 Approach

4.1 Model Overview: The implementation of QANet in this project closely followed the original paper. The model consists of 5 main parts, illustrated in the left part of figure 1: An input embedding layer, an embedding encoding layer, a context-query bi-attention layer, three repeated model encoding layers, and an output layer.

- **Input Embedding Layer**: The model uses the standard technique of representing words by concatenating their word vectors and their character vectors. The word vectors are pre-trained vectors from GloVe and are fixed during training, while the character vectors are trainable vectors obtained by convolving the vectors of every character of the word. The concatenated vectors are then passed through a two-layer highway network.
- **Embedding Encoding Layer and Model Encoding Layers**: Each embedding encoding layer and model encoding layer is built from encoder blocks, illustrated in the right part of figure 1. Each encoder block consists of a positional encoding layer, several convolution layers, a self-attention layer, and a feed-forward layer. The idea is that "convolution captures the local structure of the text, while self-attention learns the global interaction between each pair of words." The positional encoding sublayer and self-attention sublayer are the same as those in the Transformer model. The convolutional sublayers use depthwise separable convolutions with fewer parameters than traditional convolutions. The feed-forward sublayer is a composition of linear layers and ReLU activation. The Embedding Encoder Layer precedes the Context-Query Bi-attention Layer, followed by three repeated Model Encoder Layers.
- **Context-Query Bi-attention Layer**: This module is relatively standard with a similarity matrix S between context and query words is used to compute the context-to-query and query-to-context attentions: $A = \bar{S}.Q^T$ and $B = \bar{S}.\bar{\bar{S}}.C^T$, where Q and C are the encoded query and context, and $\bar{S}$ and $\bar{\bar{S}}$ are the row- and column-normalized matrix of S using softmax.
- **Output Layer**: The outputs of the first 2 model encoder blocks are passed through a linear and then a softmax layer to compute the probability distribution of the starting position of the answer. Similarly, the probability distribution for ending positions is computed using the first and the third encoder block. The loss function is then the negative log probabilities of the actual starting and ending positions, averaged over the dataset.

$$p^1 = softmax(W_1([M_0; M_1])), \qquad p^2 = softmax(W_2([M_0; M_2]))$$
$$L(\theta) = -\frac{1}{N}\sum_{1}^{N}[log(P^1_{y^1_i}) + log(P^2_{y^2_i})] \tag{1}$$

- **Inference**: At inference time, the answer is chosen to maximize the product of the probabilities of starting and ending positions. An out-of-vocabulary word was at the beginning of every context paragraph and every question to accommodate unanswerable questions.

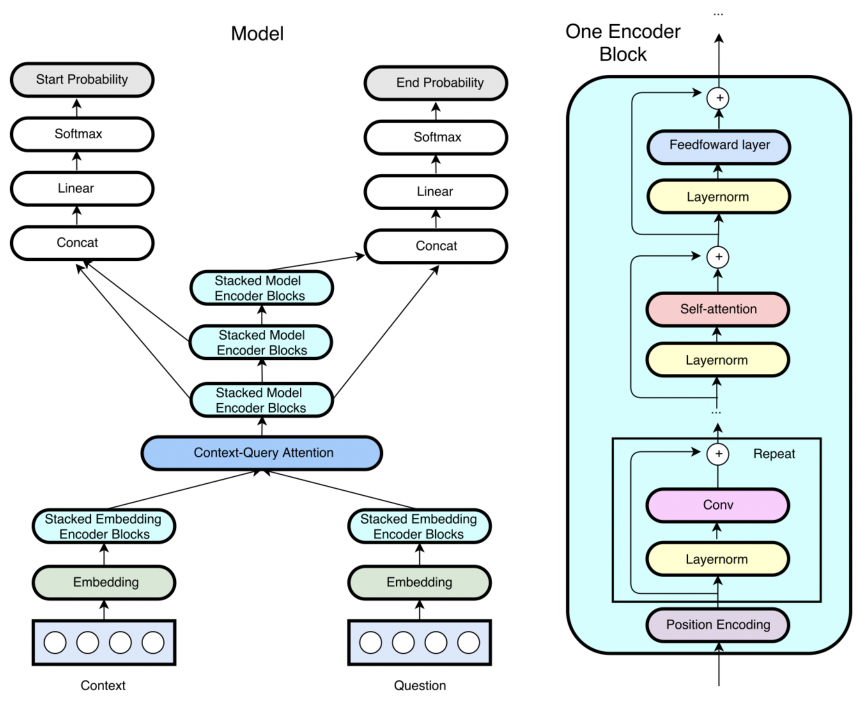


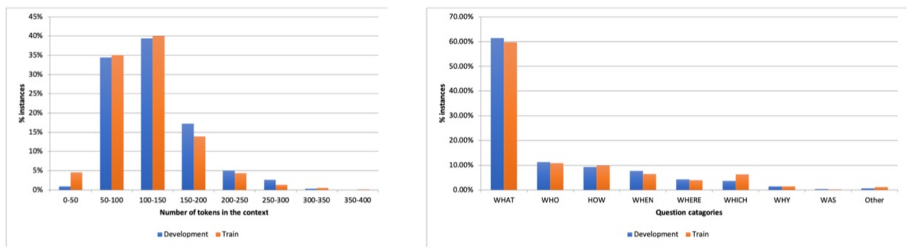
Figure 1: An overview of the QANet architecture (left) which has several Encoder Blocks. We use the same Encoder Block (right) throughout the model, only varying the number of convolutional layers for each block. We use layernorm and residual connection between every layer in the Encoder Block. We also share weights of the context and question encoder, and of the three output encoders. A positional encoding is added to the input at the beginning of each encoder layer consisting of sin and cos functions at varying wavelengths, as defined in (Vaswani et al., 2017a). Each sub-layer after the positional encoding (one of convolution, self-attention, or feed-forward-net) inside the encoder structure is wrapped inside a residual block.

4.2 Baseline: For the baseline, the BiDAF model was used. The course starter code implements this model without the character-level embedding layer. This layer was added to the starter code to obtain a slightly more robust baseline.

4.3 Implementation: The character-embedding layer using the previous course assignment was implemented. Transformer tutorial [6] and QANet PyTorch implementation [7] were referred to throughout the experiment, while the course starter code for QANet [1] was changed.
Initially, the data augmentation process described in the original QANet [1] paper was planned. However, the model used in this experiment was slow, resulting in the assumption that the data augmentation process only fits nicely into the original paper because the QANet [1] model is much faster than everything else. Thus, the decision was made to focus on speeding up our model, but unfortunately, this was unsuccessful.

5 Experiments

5.1 Data



(a) Context lengths **(b)** Question categories

Figure 2: Dataset Distributions

The SQuAD 2.0 dataset [8] provided in the course starter code was used. The official SQuAD dataset consists of roughly 150k pairs of paragraph-question. The paragraphs are from Wikipedia, and each question is either not answerable using the provided paragraph or has an answer that is a chunk of text taken directly from the paragraph, meaning that the model has to decide whether a question is answerable and, if so, select a span of text in the paragraph that answers the question. Roughly half of the questions are unanswerable.

The training set (129,941 examples) used is similar to the official SQuAD 2.0 training set. The course splits the official SQuAD dev set into a smaller dev set (consisting of 6078 examples) and a course test set (5915 examples). The course test set is released to students, who are expected to submit their answers in a CSV file. This makes the submission process more straightforward and, at the same time, keeps the official SQuAD test set secret.

5.2 Evaluation method

The default EM and F1 metrics from SQuAD were used. The EM (Exact Match) metric gives 1 point for answers the same as the reference answers and 0 points for others. The F1 metric is less strict and is the harmonic mean of the precision and recall of predictions. The F1 score is basically $(2 * precision * recall)/(precision + recall)$. We also report AvNA, which is the percentage of correct predictions of whether or not a question is answerable.

5.3 Experimental details

This section describes the experiments performed. Since many experiments were carried out, performance effects will be shared here instead of in the result section.

5.3.1 Hyperparameters:

- **Common hyperparameters**: The settings described in the original paper were used for most of the implementation. Adam optimizer with $B1 = 0.8$; $B2 = 0.999$; $E = 10^{-7}$ and $3 * 10^{-7}$ $L2$ weight decay was used; the learning rate increases inverse-exponentially from 0 to 0.001 in the first 1000 steps then stays constant after that. A dropout rate of 0.1 for word-level embeddings 0.05 for character-level embeddings was used. Inside each embedding encoding layer and model encoding layer, the sublayer at position l-th has a dropout rate of 0.1 I/L, where L is the total number of sublayers in the layer. There is also a dropout layer of rate 0.1 between every main layer of the model.
- **Embedding dimension**: For word vectors, 300-dimensional pre-trained GloVe vectors were used for word-level embedding and experimented with 100- and 200-dimensional learnable character-level embedding. Even though the original paper uses 200 for character-level embedding dimensions, better results were obtained with 100.

- **Positional Encoding**: In [9], a positional encoding is added to the input embeddings to encode the information of the position of tokens in the sentence. The same positional encoding was used as shown following:

$$
\begin{aligned}
PE_(pos, 2i) &= sin(Pos/10000^{2i/d_{model}}) \\
PE_(pos, 2i+1) &= cos(Pos/10000^{2i/d_{model}})
\end{aligned}
\tag{2}
$$

- **Hidden size**: The hidden size throughout the model is 128, as in the original paper. The hidden size 96 was also experimented with and was found to decrease both F1 and EM scores by around 0.5 to 1.0 points while not giving any training speed gain, meaning that switching the hidden size from 128 to 96 from the best model of this project decreased both F1 and EM scores by 0.7.
- **Attention heads and batch size** : The attention heads and batch size were set to 8 and 32 throughout the experiments. The course default starter code had batch size 64, which gave CUDA out of memory error on the Azure virtual machine provided in the course, which has 16 GB memory.

5.3.2 Layer Implementation:

- **Weight Initialization** : For most layers, PyTorch's default initialization was used. However, Xavier initialization and Kaiming initialization experimented for most linear projections. As can be seen, the important thing is that it was crucial to use Xavier initialization for the linear layers in the Context-Query Bi-attention layer and the Output layer. Initially, the performances that were significantly worse than those obtained by the TensorFlow implementations such as [10]: PyTorch by default uses Kaiming initialization for linear layers, while TensorFlow uses Xavier initialization.
- **Layer Normalization**: This was first introduced in [11]. One of the challenges of deep learning is that the gradients concerning the weights in one layer are highly dependent on the outputs in the previous layer. Batch normalization is designed to fix this problem. However, it is hard to apply to recurrent neural networks [11]. Layer normalization was designed to overcome the drawback of batch normalization. Suppose H is the number of hidden units of a layer and l denotes one layer in the model. The layer normalization computes:

$$
\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l, \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2} \tag{3}
$$

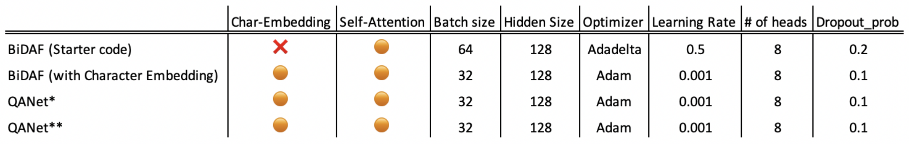
and then normalize the inputs by

$$
\hat{\mathbf{a}}^l = \frac{\mathbf{a}^l - \mu^l}{(\sigma^l)^2 + \epsilon}, \tag{4}
$$

where ϵ is a small positive number to avoid dividing by 0 problems.
- **Depth-wise Separable Convolution**: The default way to implement depthwise separable convolution in PyTorch is to use the groups feature in Conv1d.
- **Self-attention**: The nn.MultiheadAttention module from PyTorch was used.
- **Sharing weights in the model encoder**: As per the original paper, different weights were used for the 7 encoder blocks in each repetition of the model encoder layer. The same weights for all these blocks was also experimented with. In the latter approach, a linear layer with skip connection was applied between every two blocks to add more expressive power. In the experiments, sharing weights reduced the dev F1 and EM by 1.3. This approach was also slower and demands more memory, probably because its computational graph has an extra linear layer.
- **Inference**: The default inference method in the starter code is as follows: Recall that adding an out-of-vocabulary word at the beginning of every context paragraph represents the "not answerable" option. A pair of (start; end) positions is valid if:

– $start <= end < start + M$: for some chosen max length M. In our case $M = 15$.
– If $start = 0$ then $end = 0$: Here 0 is the index of the added out-of-vocabulary word.

Once the probability distributions p1 and p2 for the starting and ending positions of the answer are predicted, the joint probability p1 p2 was computed for each (start; end) pair and chose the pair with the highest probability among those which are valid. If $start = end = 0$, the model predicts “not answerable.”.



	Char-Embedding	Self-Attention	Batch size	Hidden Size	Optimizer	Learning Rate	# of heads	Dropout_prob
BiDAF (Starter code)	✗	●	64	128	Adadelta	0.5	8	0.2
BiDAF (with Character Embedding)	●	●	32	128	Adam	0.001	8	0.1
QANet*	●	●	32	128	Adam	0.001	8	0.1
QANet**	●	●	32	128	Adam	0.001	8	0.1

Figure 3: Model summary

5.4 Results

The best model of this project achieved a 63.646(F1) and 60.254(EM) score on the course test set. This model uses 100-dimensional character-level embedding, hidden size 128, 8 attention heads, batch size 32, and Xavier initialization for most linear weights. It took about 30 minutes per epoch and 15 hours to train 30 epochs.

Model	Dev(F1)	Dev(EM)	Test(F1)	Test(EM)	Evaluation
BiDAF (Starter code)	61.722	58.427	–	–	Baseline
BiDAF (with Character Embedding)	62.483	59.469	63.646	60.254	Success
QANet*	53.208	50.062	47.811	47.811	Failure
QANet**	56.119	53.487	54.708	51.784	Failure

Table 1: Overall Performance Comparison.

QANet*: QANet model with starter code along with character level embedding.
QANet**: QANet model with hyperparameter tuning discussed section 5.3

In table 1, the F1, EM scores on dev, and test set for the different models. These models are the default baseline BiDAF model, BiDAF with character embedding, QANet* model with starter code and character level embedding, and QANet** model with hyperparameter tuning described in the experiment details section.

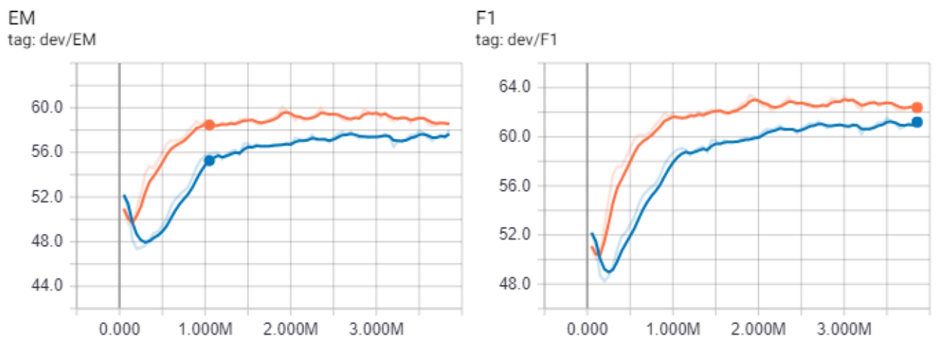


Figure 4: BiDAF with Character Embeddings. From the right, the graphs are produced on the dev set and represent EM, F1 and the loss. The orange curve represents BiDAF with word + character embeddings and the blue curve represents the baseline BiDAF

F1 and EM scores lower than the original BiDAF starter code provided with the course were given. Based on the remark in [8] that “a strong neural system that gets 86% F1 on SQuAD 1.1 achieves only

56% F1 on SQuAD 2.0" and the fact that [1] reported an 84.61 F1 score on SQuAD 1.1, leading to believe that the QANet model underperformed as expected. After exploring more about the SQuAD 2.0 dataset, there is the finding that about 50% of the data consists of questions that have no answers in the context paragraph. This difference in the SQuAD 1.1 and 2.0 datasets might have played a vital role in achieving lower scores.

However, the training speed was disappointing. The training process was slow compared to the original BiDAF starter code. It can be assumed that GPU memory was insufficient to use the parallelization feature of the QANet model [1] to show an advantage over the BiDAF starter code.

6 Analysis

Below are some randomly picked questions where the models of this project did not answer correctly.

6.1 Tackling Unanswerable Question:

Question: How many miles is Montpellier from Paris?
Context: Montpellier was among the most important of the 66 "villes de sûreté" that the Edict of 1598 granted to the Huguenots. The city's political institutions and the university were all handed over to the Huguenots. Tension with Paris led to a siege by the royal army in 1622. Peace terms called for the dismantling of the city's fortifications. A royal citadel was built, and the university and consulate were taken over by the Catholic party. Even before the Edict of Alès (1629), Protestant rule was dead and the ville de sûreté was no more.[citation needed]
Answer: N/A
Prediction: 66
Analysis: It can be seen that QANet sometimes also gives answers to unanswerable questions. In this case, 66 is not the distance from Montpellier to Paris, but the model gives this answer. One possible explanation is that the model cannot understand the query quite well. Although there is no answer to this query based on the context, the model is still trying to generate some answers. If thinking about the problem in this way, one possible direction is to model better the semantic relation between the query and the context, i.e., should let the model learn if the query and the context are related.

6.2 Unmatchable Word:

Question: What type of arts flourished in the Yuan?
Context: In the China of the Yuan, or Mongol era, various vital developments in the arts occurred or continued in their development, including painting, mathematics, calligraphy, poetry, and theater, with many great artists and writers being famous today. [...] Another important consideration regarding Yuan dynasty arts and culture is that so much of it has survived in China, relative to works from the Tang dynasty and Song dynasty, which have often been better preserved in places such as the Shosoin, in Japan.
Answer: painting, mathematics, calligraphy, poetry, and theater
Prediction: N/A
Analysis: Maybe the model could not answer this question because it could not match the word "flourished" to any word in the context. It could be argued that this is relatively difficult because the sentence containing the answer does not have any word that is an exact synonym of "flourished." However, the baseline BiDAF model answered this question correctly. The reason has not been explored in this project.

7 Conclusion

The author of this project carried out a custom implementation of QANet to increase the rate of training and inference of reading comprehension models. The core differentiator of the model is that it drops all recurrent layers in favor of convolutions and self-attention layers. While the feed-forward nature of QANet is ideal for parallel computation, it is difficult to take advantage of this feature. In contrast to the findings of the original paper, no speed increase could be observed on the baseline

BiDAF model. There are four key findings of this project: firstly, that character embeddings in conjunction with word embeddings help improve model performance regardless of the architecture; secondly, since the model implemented is capable of overfitting the data, more regularization might be required in order to allow the loss on the validation set to continue to drop; thirdly, the use of multi-head attention gives the model a clear path to interpretability on examples, as it is easy to see which part of a context or query the model is focused on; and fourthly, the model produced by this implementation does not match the performance of the original QANet, so potential changes may be required to QANet in order to improve performance.

8 Acknowledgements

Thank the CS224N course instructors and course assistants for implementing the baseline model and providing guidance in project development and Microsoft Azure for GPU service and support. I greatly appreciate Kathy Yu for providing valuable instructions and explanations throughout the course. I am immensely grateful to Lucia Zheng for her help in interpreting the articles, providing direction, and providing valuable feedback to complete the project.

References

[1] 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. CoRR, abs/1804.09541, 2018.
[2] Jonathan Raiman and John L. Miller. Globally normalized reader. In EMNLP, 2017.
[3] Dirk Weissenborn, Georg Wiese, and Laura Seiffe. Making neural qa as simple as possible but not simpler. In CoNLL, 2017.
[4] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi: "Bidirectional Attention Flow for Machine Comprehension", arXiv:1611.01603, 2016
[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser: "Attention Is All You Need", arXiv:1706.03762, 2017
[6] The annotated transformer. http://nlp.seas.harvard.edu/2018/04/03/attention.html .
[7] A pytorch implementation of qanet for machine reading comprehension. https://github.com/BangLiu/QANet-PyTorch
[8] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 784–789, Melbourne, Australia, July 2018. Association for Computational Linguistics.
[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems 30, pages 5998–6008. Curran Associates, Inc., 2017. http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf
[10] A tensorflow implementation of qanet for machine reading comprehension. https://github.com/NLPLearn/QANet
[11] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, arXiv:1607.06450, 2016