

Exploring the Architecture of QANet

Stanford CS224N Default Project

Joseph Pagadora, Steve Gan

Stanford University

jcp737@stanford.edu zgan@stanford.edu

Abstract

Before the advent of QANet, dominant question-answering models were based on recurrent neural networks. QANet shows that self-attention and convolutional neural networks can replace recurrent neural networks in question-answering models. We first implemented a version of QANet using the same architecture as that of the original QANet model, and then we conducted experiments on hyperparameters and model architecture. We incorporated attention re-use, gated self-attention, and conditional output into the QANet architecture. Our best QANet model obtained 59.3 EM and 62.82 F1 on the evaluation set. The ensemble of the two best QANet models and one BiDAF model with self-attention mechanism achieved 62.73 EM and 65.77 F1 on the evaluation set and 60.63 EM and 63.69 F1 on the test set.

1 Introduction

Transformers and the self-attention mechanism have transformed the landscape of neural language machine learning. While recurrent neural networks were once deemed an essential building block of a successful neural language model as they can absorb sequential information of a sequence of word vectors, self-attention shows that this time dependency is not necessary for language modeling. Because self-attention does not use time-dependency, it can effectively use parallelism in computation and better prevents the vanishing gradient problem common in recurrent neural networks [1].

The self-attention mechanism was first used in question answering by the QANet model [2]. This model mainly uses self-attention and convolutions to learn and is able to get rid of recurrent neural networks completely. To better understand these two important techniques in modern deep learning, we decided to implement the QANet model and experimented with different changes in the model architecture.

The report is organized as follows: Section 2 provides an overview of the development of self-attention and its use in question answering. Section 3 gives the details of each layer of our QANet's architecture. Section 4 shows experimental results of our model's performance evaluated using EM and F1 metrics. Section 5 evaluates our final model qualitatively by looking at the error distribution on the evaluation set and providing several error case-studies. Section 6 concludes the paper and describes the future research direction.

2 Related Works

Our project builds on the works of attention and transformers. The attention mechanism was first proposed by Bahdanau et al. for neural machine translators. It was used to generate word vector predictions in the decoder by attending the input word vector sequence [3]. The attention mechanism was soon to be discovered to have large potential as a building block in neural networks for natural language processing tasks. Vaswani et al. introduced the self-attention mechanism, in which a sequence of word vectors can attend itself [4]. As opposed to previous successful neural machine

translators, their self-attention-based model did not use any recurrent neural networks. Because of the lack of recurrent layers, transformers are able to take advantage of computational parallelism.

However, vanilla attention mechanisms suffer from the issue that the positional information of each word in a sentence is lost. Vaswani et al. attempted to encode the position of each word in a sequence by a positional encoder that generated a table of values of cosine and sine functions with different frequencies [4]. Another approach to solve this problem is to combine the self-attention mechanism with recurrent neural networks. R-Transformer uses self-attention to capture global information while employing recurrent neural networks to capture local positional information [5].

In the realm of question answering, Seo et al. combined recurrent neural networks with context-to-query and query-to-context attention in their BiDAF model and achieved success [6]. Incorporating the bidirectional attention flow mechanism, Yu et al.'s QANet model showed that as in neural machine translation, recurrent neural networks can be replaced in question-answering models by self-attention and convolutional neural networks [2]. At the same time as the release of QANet, the natural language processing community was stepping into the era of large pretrained transformers such as Bert and Albert [7], [8]. These models were finetuned for various downstream tasks and are the current state-of-art in question answering [9].

3 Approach

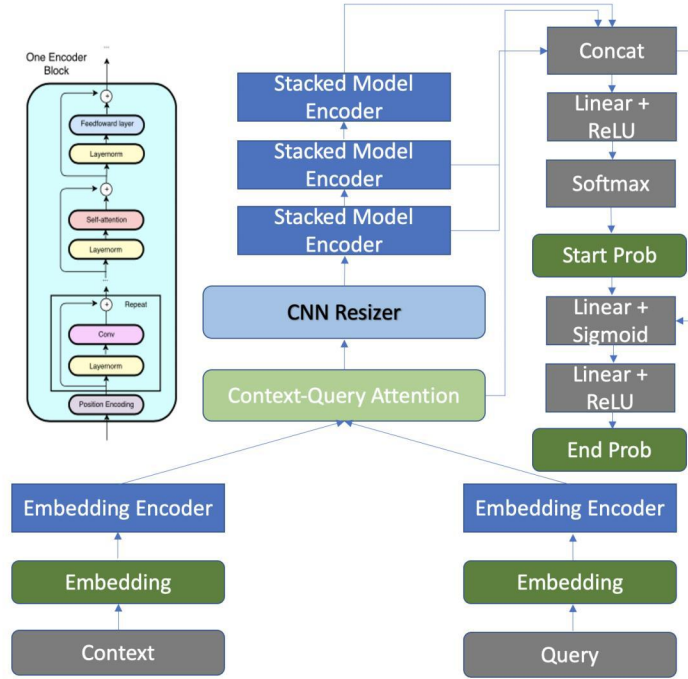


Figure 1: Architecture of QANet with attention re-use and conditional output. Each encoder block has the same the architecture as the original QANet [2].

3.1 Model Overview

We have implemented a version of the QANet model whose architecture is summarized in Figure 1. As in the original paper, we only use self-attention and convolutional neural networks to encode information [2]. Our model consists of six layers: the embedding layer, the embedding encoder layer, the context-query attention layer, the convolutional neural network resizer layer, the model encoder layer, and the conditional output layers. The embedding encoder and model encoder layers both consist of encoder blocks.

3.2 Embedding Layer

Words are embedded using GLoVE vectors (300-dimensional) and character embeddings (64-dimensional). Words are truncated or padded to have a length of 16 characters as described in Seo et al. [6]. We passed the character embeddings through a convolutional layer and max-pooled the character sequence in each word. Character embeddings are allowed to be updated during the process of training. For each word, these two resulting embeddings are concatenated to form a 364-dimensional embedding. We then project these embeddings to a lower dimension and pass them through a 2-layer highway network [10]. This process is performed separately for context words and question words.

3.3 Embedding Encoder Layer

The model encoder layer has a single encoder block, which has 4 depthwise separable convolutions. The encoder block of our model consists of the following: a position-encoder [4], a specified number of depthwise separable 1-dimensional convolutions with kernel size 7 and filter size 128 [11], a self-attention layer, and finally a feed-forward layer. The convolutional and self-attention layers have a residual block, in which the main layer is preceded by layer-norm and followed by stochastic dropout. We implement the later on our own. We also drop out every other convolutional layer as in the original paper [2].

The positional encoding scheme is based on the one implemented in the original transformer [4]. Assuming hidden size is an even number, define c as $(\text{hidden size} / 2 - 1)$.¹ It is given as the following formula:

$$\begin{aligned} PE_{pos,i} &= \sin(pos/10000^{2i/d_{\text{model}}}) \quad 0 \leq i \leq c, \\ PE_{pos,j} &= \cos(pos/10000^{2(j-c-1)/d_{\text{model}}}) \quad c+1 \leq j \leq \text{hidden} - 1, \end{aligned}$$

For self-attention, we add the gated attention mechanism used in Reinforced Mnemonic Reader to better fuse attention information into the vector representation of the context [12]. The output of the self-attention y is fused with the original input to the layer x in the following way:

$$\begin{aligned} \tilde{y} &= \text{ReLU}(W_1[y; x; y \odot x; y - x]), \\ g &= \sigma(W_2[y; x; y \odot x; y - x]), \\ o &= g \odot \tilde{y} + (1 - g) \odot y, \end{aligned}$$

where $\odot$ denotes the Hadamard product. The application of the gate g allows the model to control how much attention information to be incorporated to the original input to generate the final output o .

We adopt the position-wise feed-forward network in the original transformer paper as our feed-forward layer [4]. This network is described by the following equation:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2.$$

This feed-forward layer is equivalent to applying two convolutions with kernel size 1 with a ReLU activation in between.

3.4 Context-Query Attention

The output of the embedding encoder layer is then passed through a context-query attention layer. Instead of using the DCN context-query attention, we use the attention layer in BiDAF [6]. The output of this layer goes to both the resizer and the conditional output layer. In other words, we re-use context-query attention to predict the answer span.

¹Note that the following formula is not exactly the same as the original positional encoding where each sine and cosine values alternate from dimension to dimension. The following way basically concatenates the sine positional encoding and cosine positional encoding. We tried the original alternating positional encoding, and it did not work (training loss did not go down). One TA suggested that we should look up a tensorflow implementation of QANet and checked our implementation against theirs. We got to know the current scheme from the following tensorflow implementation of QANet: <https://github.com/NLPLearn/QANet>.

3.5 Convolutional Resizer

Because the output of the context-query attention layer is the concatenation of four vectors of the hidden size, we resize this output back to the hidden size. We achieve this by using one convolution with kernel size 1 activated by a ReLU function.

3.6 Model Encoder Layer

The output of the resizer flows into three stacked model encoders. Each model encoder layer has a specified number of encoder blocks. The outputs of the first and second stacked encoder are also connected to the conditional output layer. We use weight-sharing among these three encoders, where each encoder block in a model encoder has 2 depthwise separable convolutions.

3.7 Conditional Output Layer

Our final output layer is inspired by the boundary answer pointer layer, as described in [13]. For each word in the context, this layer computes the probabilities of it being the start and end of the answer span. The end probability is conditioned on the start probability. We implement this conditional relationship in the following scheme:

$$\begin{aligned} H &= [M_0, M_1, M_3, Att], \\ F_{start} &= ReLU(HW_1), \\ p_{start} &= Softmax(F_{start}v_1), \\ h &= \sigma(V(Hp_{start}) + \beta), \\ F_{end} &= ReLU(HW_2 + h \otimes e_{seq}), \\ p_{end} &= Softmax(F_{end}v_2), \end{aligned}$$

where $W_1, W_2 \in \mathbb{R}^{7l \times l}$, $V \in \mathbb{R}^{l \times l}$, $v_1, v_2 \in \mathbb{R}^{l \times 1}$, $\beta \in \mathbb{R}^{l \times 1}$, $\otimes e_{seq}$ denotes broadcasting over the word sequence, M_0 , M_1 , and M_2 denote the output of three model encoder blocks, and Att denotes the output of the context-query attention layer. The information from the start probability is encoded in the vector h , which is used in the process of computing the end probability. The main difference between our implementation and the original boundary answer pointer is that in the step to compute the vector h , our model simply uses a linear transformation followed by a sigmoid function while the original boundary answer pointer passes Hp_{start} into a LSTM cell. In our experimentation, we discovered that when a LSTM cell was used in this step, the training loss went down very slowly and our model barely got trained over a long period of time. To speed up the training, we decided to only preserve a single set of non-linearity and trainable parameters.

3.8 BiDAF with Character Embeddings and Self-Attention

In addition to QANet, we also modified the original BiDAF model by adding a character-embedding component and a gated self-attention component. These two components are the same as their counterparts in our QANet architecture. This BiDAF model was used to form an ensemble model with our QANet models.

4 Experiments

4.1 Data

The data used in this study is a subset of the official SQuAD 2.0 dataset (Stanford Question Answering Dataset). The dataset is divided using the standard three-split pattern: train (129,941 examples), dev (6078 examples), test (5915 examples).

4.2 Training Details

We trained our models using the ADAM optimizer [14] with $\beta_1 = 0.8$, $\beta_2 = 0.999$, and $\epsilon = 10^{-7}$. We also used exponential moving average with a decay rate 0.9999 and a learning rate warm-up scheme that increased the learning rate from 0 to 0.001 in the first 1,000 steps with the logarithmic

function. We set the dropout probability to 0.1. In the embedding layer, we additionally dropped out word and character embeddings with probabilities 0.1 and 0.05, respectively, before feeding them into the highway network. These procedures were implemented according to the original QANet paper [2].

4.3 Evaluation Methods

We evaluated our models using the exact-match (EM) and F1 scores in the process of model development. We also examined the types of errors and areas of strength of our final model in Section 5.

4.4 Results

We proceeded with our experiments in two stages. In the first stage, we implement our baseline model: QANet with a hidden size of 96, 1-head attention, and a 4-block model encoder. Each time, we changed one hyperparameter or one basic structure to see the difference. In the second stage, we implemented three architecture changes that we have described in the approach section: attention re-use, gated self-attention, and conditional output. The results of the first stage are summarized in the following table:

Model	Dev EM	Dev F1
Baseline QANet	54.09	57.12
Hidden Size = 64	52.60	56.22
Hidden Size = 128	55.67	59.27
8-Head Attention	53.60	57.58
7-Block Model Encoder	55.07	58.00
Without Stochastic Dropout	54.63	57.04
Normal CNNs (not depthwise separable)	53.32	54.36

In the first stage of experiments, we noticed that a larger hidden size, deeper encoders, and the use of depthwise separable convolutions were features that helped our QANet model learn. While 8-head attention is presented in both Vaswani et al. [4] and Yu et al. [2], we did not find that it improved our model’s performance. Moreover, 8-head attention required significantly more time to compute, so we decided not to include it in our second stage of experiment. Although stochastic dropout did not bring an improvement to our model’s performance, it reduced computation time, so we included this architecture feature in the second stage.

After we finished the first stage, we planned to proceed with a QANet model with a hidden size of 128. However, combined with attention re-use, the model did not seem to learn as the Dev EM and F1 metrics kept fluctuating over a long period of time, and we decided to switch back to a hidden size of 96. Except changing the model encoder to have 7 blocks, we kept all other model architecture features.

Model	Dev EM	Dev F1
Hidden Size = 96, 1-Attention, 7-Block Model Encoder	55.07	58.00
Attention re-use	57.20	60.64
Gated Self-Attention + Attention re-use	57.27	60.67
Conditional Output + Gated Self-Attention + Attention re-use	57.18	60.3
Gated Self-Attention + Attention re-use (40 epochs)	59.3	62.82
Conditional Output + Gated Self-Attention + Attention re-use (40 epochs)	58.26	61.83
BiDAF with Character Embeddings	60.10	63.15
BiDAF with Character Embeddings and Gated Self-Attention (40 epochs)	61.21	64.17
Ensemble	62.73	65.77

From the table above, we can see that attention re-use alone brought more than 2-point increases in both Dev EM and Dev F1. Adding gated self-attention to the model, we gained a light increase in performance. Conditional Output failed to bring further improvement to the model within 20 epochs. We noticed that the loss of the model with only attention re-use started to rise in 20 epochs as shown

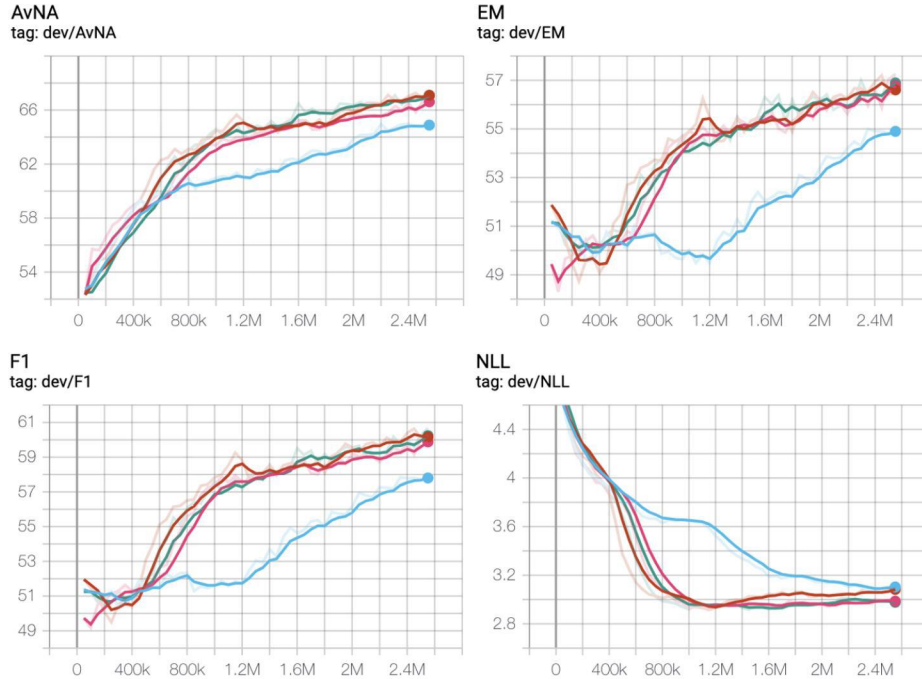


Figure 2: Comparison of different modifications of QANet’s architecture (number of encoder blocks in the model encoder is 7 for all the models). Pink: conditional output + gate + attention re-use, Green: gate + attention re-use, Red: attention re-use, Blue: original architecture

in Figure 2. Therefore, although all three models were in an upward trend, we conjectured that the later two models had more potential to learn if we could train them longer. We trained them for 20 more epochs and achieved more than 2-point increases in both metrics using the model with gated self-attention and attention re-use.

After completing our experiments with QANet, we put a character embedding layer and a self-attention layer to the base BiDAF model. We let the BiDAF model have a hidden size of 100. Then we used this model and two best QANet models to generate an ensemble model.

For each question, we averaged the log-start and log-end probabilities of these three models to generate the answer span. This final model achieved 60.63 EM and 63.69 F1 on the test set.

5 Analysis

We analyzed some results of our final ensemble model on the hold-out validation data. First, we wanted to analyze the performance of our model on different types of questions, that is, "who," "what," "when," "where," and "why." We also looked at questions with "which" and "how." This analysis gave us insights into how our QANet model worked, especially if the performance depended on question type. First, we looked at the breakdown of question type by whether our model answered correctly or incorrectly. We defined "correctness" by whether the given correct answer in the data (as a list) included the predicted answer (as a string). That is, we did not look at spans. "Correct" also included not answering a question if there was no given answer, i.e., unanswerable from the context. If these were not the case for a given datapoint with its prediction, then we considered the answer incorrect.

Unsurprisingly, "what" questions were the most common type among all questions, garnering more than half (50%) of all questions in the validation data. An important thing to take away from the analysis in Figure 3 is that the correctness of the model by question type are nearly equivalent. We conjecture that this was the effect of creating an ensemble model that averaged the performance three different models which might be good at different types of questions/ Indeed, the bar-plot in Figure 3

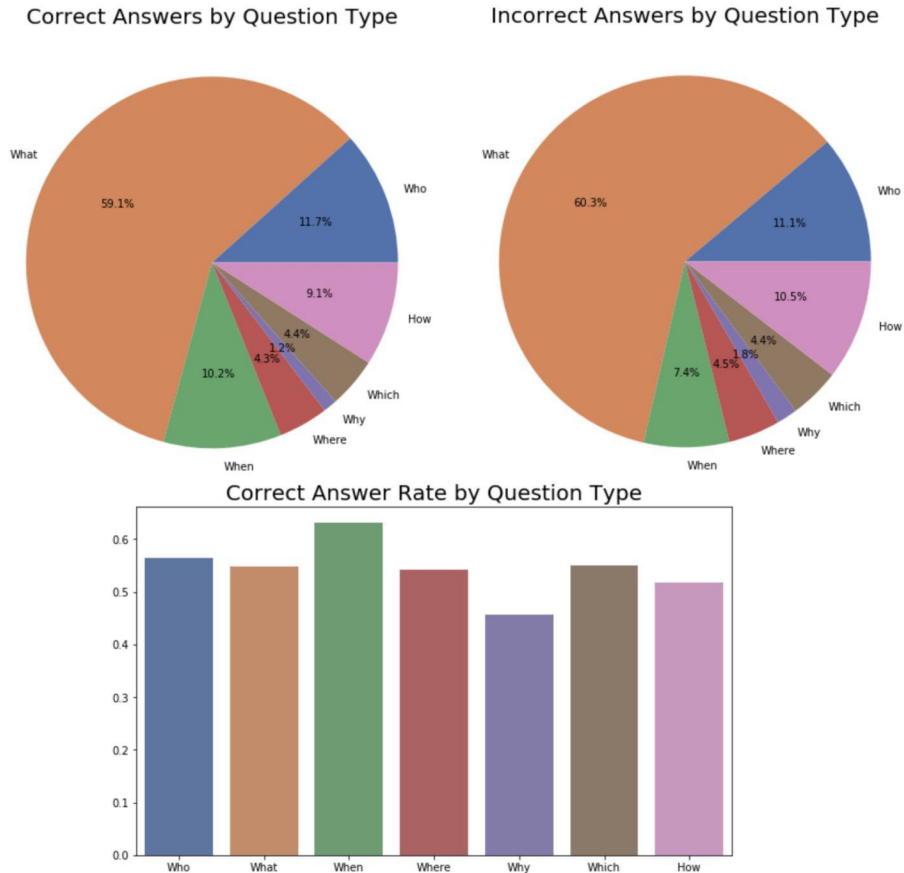


Figure 3: Left: Breakdown of question type for correctly answered questions. Right: Breakdown of question type for incorrectly answered questions.

shows that breaking down correctness by question type. We can see that our model perform slightly better on the "when" questions and slightly worse on the "why" questions.

Of the questions that were un-answerable about 56% of unanswerable questions were correctly not answered by QANet. What is interesting is that a good portion, about 43%, of unanswerable questions were actually given an answer by our model, and so we looked at a few examples to illustrate what might be going on. In Example 1, notice that the context has the answer to the question if it were "What is the force exerted by standard gravity on one kilogram of mass?," to which the answer would be kilogram-force. However, our model did not notice this and indeed predicted that answer, suggesting that the model has a general sense of grasp on the passage, but may not pay attention to detail. Similarly, in Example 2 (same context paragraph), the answer to the question is quite particular, and indeed, even the passage itself says the terms such as "kip" and "sthene" are arcane. We conjectured that our models perform poorly on questions that have answers with rare words because of the use of word embeddings. During the original training process of GLoVe, the relative frequency of each word might have already been encoded into the word vectors. However, the weakness of our model to answer this type of questions might not be a big problem because there might be fewer user cases for a machine to answer arcane questions.

Context	The pound-force has a metric counterpart, less commonly used than the newton: the kilogram-force (kgf) (sometimes kilopond), is the force exerted by standard gravity on one kilogram of mass. The kilogram-force leads to an alternate, but rarely used unit of mass: the metric slug (sometimes mug or hyl) is that mass that accelerates at 1 m·s ⁻² when subjected to a force of 1 kgf. The kilogram-force is not a part of the modern SI system, and is generally deprecated; however it still sees use for some purposes as expressing aircraft weight, jet thrust, bicycle spoke tension, torque wrench settings and engine output torque. Other arcane units of force include the sthène, which is equivalent to 1000 N, and the kip, which is equivalent to 1000 lbf.
Question	What is the force exerted by standard gravity on one ton of mass?
True Answer	NA
Predicted.	kilogram-force

Table 1: Example

Question	What seldom used term of a unit of force equal to 1000 pounds of force?
True Answer	kip
Predicted.	NA

Table 2: Example

6 Conclusion & Future Work

In this project, we have successfully implemented the QANet architecture using the details given in [2]. We have experimented with three additional architecture features that are not in the original QANet paper: attention re-use, gated self-attention, and conditional output. While attention re-use improved the performance of our model the most, conditional output was not very useful for our QANet model.

The next step of our research includes two parts. First, we want to implement the DCN context-query attention which is used in the original model. Second, currently, we solve the position-less property of self-attention by using a positional encoder. We intend to implement the local recurrent neural network approach used in R-Transformer to encode the local positional information [12].

References

- [1] Giancarlo Kerg, Bhargav Kanuparthi, Anirudh Goyal, Kyle Goyette, Yoshua Bengio, and Guillaume Lajoie. Untangling tradeoffs between recurrence and self-attention in neural networks. *arXiv preprint arXiv:2006.09471*, 2020.
- [2] 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.
- [3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- [4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.
- [5] Zhiwei Wang, Yao Ma, Zitao Liu, and Jiliang Tang. R-transformer: Recurrent neural network enhanced transformer. *arXiv preprint arXiv:1907.05572*, 2019.
- [6] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.
- [7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

- [8] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.
- [9] Pranav Rajpurkar. Squad2.0.
- [10] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.
- [11] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.
- [12] Minghao Hu, Yuxing Peng, Zhen Huang, Xipeng Qiu, Furu Wei, and Ming Zhou. Reinforced mnemonic reader for machine reading comprehension. *arXiv preprint arXiv:1705.02798*, 2017.
- [13] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905*, 2016.
- [14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.