Recurrency-Free Question Answering with Unanswerable Questions

Michael Hazard cmhazard@stanford.edu
Ian Hodge ihodge@stanford.edu
Daniel Semeniuta dsemeniu@stanford.edu

Abstract

We implement the QANet model for question answering proposed by Yu et al. and adapt it for the SQuAD 2.0 dataset. We propose a verifier architecture for use with the QANet model to improve its performance on the unanswerable questions introduced by SQuAD 2.0. Additionally, we improve upon the baseline Bidirectional Attention Flow (BiDAF) model to achieve an EM score of 60.473 and an F1 score of 64.201 on the test set.

1 Introduction

The past few years have seen a significant increase in the natural language processing community's interest in Machine Comprehension (MC) systems. Systems have reached near human performance at identifying answers within a section of text, but they tend to make unreliable predictions when faced with questions that do not have an answer. The SQuAD 2.0 dataset, which contains unanswerable questions to train on, was created to address this problem. SQuAD 2.0 significantly increases the difficulty of the problem as complex models that achieve very high F1 scores in SQuAD 1.1 see a significant decrease when being evaluated on SQuAD 2.0 [10].

Each system designed for question answering must be able to model complex interactions between the context text and the query in order to effectively perform. One of the biggest advancements in this area has come from neural attention mechanisms. These mechanisms enable systems to focus on specific areas of the context paragraph that is most relevant to the given question [14]. There have been many successful models that combine this neural attention system with a recurrent model to process sequential inputs. The BiDAF model, which we use as our baseline, is a good example of this as it reported very high numbers on the SQuAD dataset [11].

One problem with recurrence based models is that they are slow for both training and for inference due to their sequential nature. This complicates efficient parallelization and experimentation, especially with a large data set. In order to solve this problem, Yu et al created the recurrence free QANet structure for question answering [15]. Their model relies on convolutions and self-attentions in order to encode contexts and questions rather than recurrent networks. This greatly increases training speed and allows for a faster iteration process when tuning hyper parameters. For these reasons, we decided to use the their model as the base of our approach to this problem.

The QAnet model performed very well on the SQuAD 1.1 dataset, but Yu et al wrote their paper before the the release of SQuAD 2.0. As a result, there is no optimization for specifically dealing with unanswerable questions built into the model. In order to combat this, we attempted to adapt the answer verifier system described by Hu et al. in their paper 'Read + Verify: Machine Reading Comprehension with Unanswerable Questions' to a non-PCE setting[5][9]. We hoped that by combining these two approaches we could improve the QAnet model's on unanswerable questions.

2 Related work

A wide array of systems and architectures have been either applied or invented to tackle the question answering problem posed by SQuAD. State of the art at the time of its publication, the Bidirection Attention Flow (BiDAF) model described by Seo et al. in [11] utilizes bidirection Long Short-Term Memory networks (LSTM) for capturing contextual connections and attention flow mechanisms for building the relationship between context and query inputs in its question answering scheme. Due to the recurrent structure of BiDAFs, we were concerned about the ability to quickly run through iterations of model tuning in the debug cycle.

Beyond research done specifically for the task of question answering, other language models have informed the work done in the task. Vaswani et al. [13] propose the transformer, a model which is entirely recurrence-free and instead based on self-attention mechanisms to build global and local dependencies between the model input and output. The transformer utilizes stacked self-attention, point-wise, fully connected layers and positional embeddings in building the encoder and decoder blocks. Additionally, Vaswani et al. propose a method of Scaled Dot-Product Attention with multiple heads to increase the ability of the model to attend to information from different subspaces. These stacked encoder blocks inspired the design of the encoder blocks in QANet. Addtionally, the described self-attention mechanism is utilized in QANet's architecture.

The BiDAF model was eventually supplanted by models which utilized pre-trained contextual embeddings (PCE) such as Embeddings from Language Models (ELMo) [8] and Bidirectional Encoder Representations from Transformer (BERT) [2]. Pre-trained contextual embeddings go beyond singular word embeddings as in Global Vectors for Word Representation (GloVe) [7]. With PCE models, word embeddings are dependent on the context in which a word appears in the text. PCE models greatly outperform any non-PCE models on the SQuAD 2.0 dataset.

3 Approach

3.1 Baseline

For our baseline model we utilized the provided starter code of the BiDAF model. To extend the capabilities of the BiDAF model and test if the character embeddings we designed for QANet worked, we added character embeddings by the method described in section 3.2.1 below.

3.2 QANet Model

Our approach is modeled on the QANet structure described by Yu et al [15]. With the exception of a few parts noted explicitly, we designed our own PyTorch implementation. At a high level, we utilize five different layers: an embedding layer, an embedding encoding layer, a context query attention layer, a model encoder layer, and an output layer, which can be seen visually in figure 1. The context and the query are encoded separately, but they share the same weights. They are combined during the context query attention layer.

3.2.1 Embedding Layer

We obtain the GloVe embedding $p_w \in \mathbb{R}^{300}$ of each word w. These vectors are fixed during training. We also train character embeddings of dimension $p_c \in \mathbb{R}^{64}$ initialized from a pretrained embedding matrix. We additionally pass these character embeddings through several convolutional layers. The final character embedding representation is the max character value for each word. These character embeddings are then concatenated with the word embeddings to form the embedding of our context and query. We then apply a two layer one-dimensional convolutional highway network to these embeddings [12]. Before passing the output to the next layer of our model we project the hidden dimension from $p_w + p_c$ to the hidden size consistent within our entire model.

3.2.2 Embedding Encoder Layer

The embedding encoder layer consists of two different encoder blocks. Each block consists of three sections: a four-times repeated convolutional block, a self-attention layer, and a feed-forward layer. Each of these operations is preceded by a layer normalization and maintains a residual connection

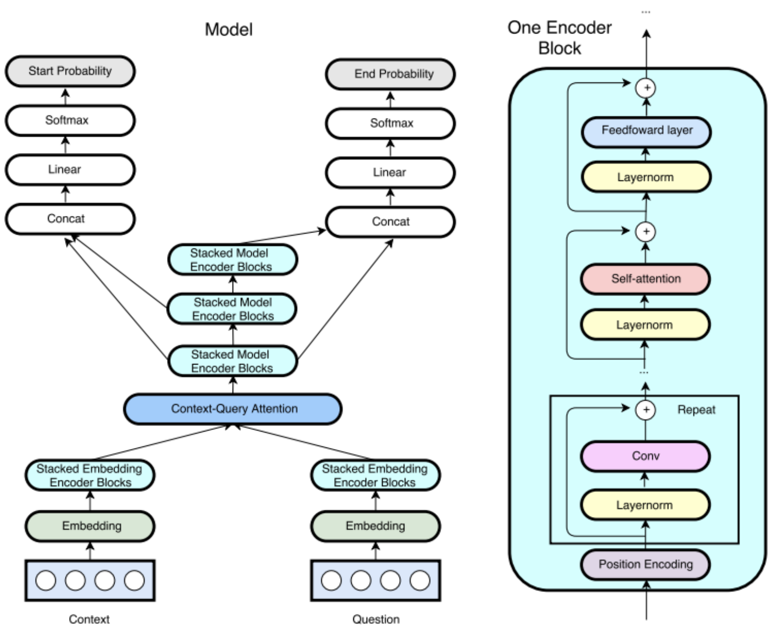

Figure 1: QANet Model Architecture

to the input prior to the normalization. As suggested by the QANet authors, we also add a chance of hidden unit dropout for every other sublayer in the encoder block. This means for each of these functions *f*, the output of the layer is equal to $dropout(f(layernorm(x)) + x)$ for an input x, with $p_{\text{drop}} = 0$ in odd layers, and $p_{\text{drop}} = 0.1$ in even layers. Additionally, within the convolutional blocks, we utilize depth-wise separable convolutions as this has been shown to increase the generalization and parallelization capacities of the model [1]. Similar to Yu et al, we used a filter size of seven with same padding so as to maintain the length of our input sequence in our convolutions.

For the self attention layer, we utilized a multihead attention mechanism as described by Vaswani et al [13] with 4-8 heads depending on the set of hyperparameters utilized. For this part of the code, we adapted a PyTorch implementation of the model as described in 'Attention is all you need' and clarified in Harvard's The Annotated Transformer.

In addition to traditional dropout at the end of each sublayer, we apply stochastic depth dropout to the sublayers of the encoder block [6]. Each layer has a survival probability of $p_l = 1 - \frac{\ell}{L}(1 - p_L)$ where, L is the total number of sublayers across all of the blocks, ℓ is the current layer number, and $p_L = 0.9$. Stochastic depth dropout prevents co-dependence of sublayers. If a sublayer does not survive, then the sublayer returns an identity of its input.

3.2.3 Context Query Attention Layer

For this layer, we adapted the code implemented in the BiDAF baseline for computing context-to-query attention. It starts by computing the similarity between each pair of words in the context and query and storing the results in a similarity matrix S. This S is computed as a trilinear similarity function between the query and context, where element-wise similarity is $f(Q, c) = W_0[q, s, q \odot c]$ and W_0 is a learnable parameter. We then normalize the rows of S with the softmax function and denote it as $\bar{S}$ and normalize the columns of S with the softmax function and denote it as $\bar{\bar{S}}$. The context-to-query attention is then computed as $A = \bar{S} \cdot Q^T$, and the query-to-context attention is computed as $B = \bar{S} \cdot \bar{\bar{S}}^T \cdot C^T$. This layer combines the separate encodings of the query and context into a single representation.

3.2.4 Model Encoder Layer

The input of this layer is built from a concatenation of the respective attentions calculated in the previous layer, $[c, a, c \odot a, c \odot b]$ where a and b are the rows of the attention matrices A and B. This concatenation of size $4d$ is then projected with a pointwise one-dimensional convolution back down to our hidden size, d. This layer is built from the same encoding blocks described in the embedding encoding layer. The differences are in the number of repetitions of the encoder block in each stack, 7, and the number of times we repeat the convolution, 2 instead of 4.

We construct three of these seven-times stacked encoder blocks, which share weights. The first and second of the three encoder blocks are used to calculate the probabilities of the beginning of the answer, while the first and third calculate the probabilities of the end of the answer in the context. The outputs of these encoder blocks by design maintain the same shape as the input.

3.2.5 Output Layer

The output layer is task specific to the SQuAd dataset, as the architecture of QANet can be adapted to various machine comprehension tasks. We try to predict the probability of each position being either the start or the end of the answer span. More specifically, given outputs M_0, M_1, M_2 from the three model encoders and two trainable weight matrices W_0 and W_1 we use softmax to calculate the start and end probability distributions p_1 and p_2 with the equations:

$$p_1 = softmax(W_0[M_0; M_1]), p_2 = softmax(W_0[M_0; M_2]).$$

We concatenate the output of the first and second encoder block for the start position and the first and third block for the end position along the projection dimension. We then apply a pointwise convolution to project each word's embedding down to a scalar value. Thus, each positional word in the context is represented by a single value. We then apply a softmax to assign probabilities of any given word being at the start and end positions of the answer.

The score of a candidate answer is defined as the product of the probabilities of its start and end positions. In predicting an answer, the span with the highest score is chosen. By default, our baseline for predicting an unanswerable question is an additional sentinel position in the input context. If the highest score is assigned to a span beginning and ending at this sentinel position then question is predicted to be unanswerable.

If using the answer verifier and an answer for the question is found, the model proceeds to the Verifier Embedding Layer described in 3.3 below. Otherwise the results of this layer are returned.

3.3 Planned Verifier Architecture

Given both QANet's prior status as a state of the art system for SQuAD 1.1 and our relative underperformance on unaswerable questions, we believe that the QANet architecture could be improved upon by adding a more robust system to distinguish between answerable and unanswerable questions. As such, we propose the following architecture for an additional verifier layer on top of QANet. As QANet's model architecture is largely agnostic to the structure of the output layer, this verifier can easily slot into our model.

3.3.1 Embedding Layer

The inputs of this verifier layer are the context indices S, question indices Q, and answer indices A from the QANet model. These three inputs are concatenated to produce a new input $I = [S, Q, \langle\text{DELIMETER}\rangle, A]$. As in the QANet model we retrieve a GLoVe embedding $p_w \in \mathbb{R}^{300}$ for each word $w \in I$, and inject positional embedding as the model is not sequential. These embeddings are then projected down to the hidden size of the model.

3.3.2 Transformer Layer

As depicted in Figure 2 above, this layer is composed of a standardized transformer block repeated 12 times. The standard transformer block is composed of a masked multi-head self attention followed by a feed-forward layer. Each of the two sub-components feature a residual connection and layer normalization.

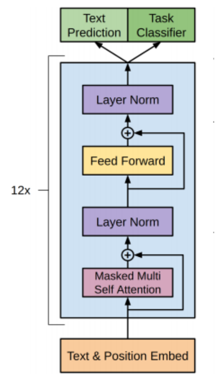

Figure 2: Verifier Architecture

3.3.3 Output Layer

The output of the last transformer block is projected down from the hidden dimension to 2 dimensions. A softmax is then taken over these two dimensions generating the probability of the answer being yes or no respectively. Given a higher "no" probability the system returns no answer, and given a higher "yes" it returns the answer pointed to by the output of the QANet system.

3.3.4 Verifier Pre-Training

As noted in 'Read + verify:Machine reading comprehension with unanswerable questions' [5] the verifier performs best when pre-trained as a language model. We first wanted to test our verifier architecture without pre-training to see if this is this is possible given our different architecture and usage of GLoVe embeddings as opposed to BERT embeddings. Our QANet implementation never beat the baseline so we unfortunately did not have time to do this testing.

4 Experiments

4.1 Data

We train and evaluate our model on the SQuAD 2.0 dataset.

4.2 Evaluation Method

Our evaluation methods follow the methods laid out in the default project handout. We utilize negative log likelihood for our training loss. Our model's answer predictions are evaluated using EM and F1 scores.

4.3 Experimental Details

To ensure the correctness of our model, we designed sanity checks which checked the shape and values of the output of individual layers and as well as the complete model. Due to the modularity of PyTorch, we were able to treat single layers as full modules and test that each layer was keeping the shape of our output consistent with expectations. Additionally, it allowed for easier debugging in checking for NaNs in our output probabilities as we were able to narrow down which layers were the source of overflow/underflow bugs. Once we passed all of the sanity checks we proceeded to train on a 30 example subset of SQuAD 2.0 until the model was able to over fit this dataset.

We evaluated the performance of our model on the dev set on Tensorboard as it trained. We utilized an Adam optimizer with $\beta_1 = 0.8, \beta_2 = 0.999$ as this was the proposed solution by the QANet rather than the Adadelta optimizer provided by the baseline. We utilized this same Adam optimizer when running our tuned BiDAF model. As part of our optimizer when training QANet, we utilized an inverse exponential learning rate warmup scheme for the first 1,000 training batches until reaching a constant learning rate of $\alpha = 10^{-3}$.

In our first experiments, we trained the model with a simplified word-level only embedding layer, prior to implementing character-level embeddings. Due to memory issues, we had to experiment with various architectural changes which ended up limiting performance. We found that the number of heads did not have a large impact on overall performance, but did have a large memory footprint. In order to run the model with the eight heads specified in the paper, for example, we had to cut the hidden size of the model from 128 to 96.

Additionally, we had to lower our batch size to anywhere between 8 and 16 examples when using the base GPU and full model architecture, which drastically hurt how fast the model was able to train. To alleviate the issues introduced by these smaller batch sizes, we implemented gradient accumulation in our training code apparatus. Gradient accumulation led to reduced noise in the training loss as we trained as it increased our effective batch size for the gradient optimizer. In addition to QANet being a large model on its own, we believe that there are certain bugs in our implementation which led to greater than expected memory usage, as we were only able to build a model with the teaching staff's suggested hyperparameters when using a 16GB GPU, when the staff was able to utilize a 12GB GPU.

4.4 Results

Our QANet implementation was never able to surpass the baseline. Table 1 visualizes our best results. Not shown are the multiple QANet implementations or tunings which failed. Our error analysis describes a few misclassified examples and attempts to understand where the QANet model fails.

We achieved results which beat the baseline by tuning several hyperparameters of the provided BiDAF model and adding character embeddings. At the time of submission our model is ranked 24th of 54 submissions on the Test Non-PCE leaderboard. The model achieved respectable performance in classifying unanswerable questions with an AvNA score of 72.31, which still leaves room for improvement.

We additionally tested the effects that Xavier [3] and He [4] initialization of our model weights had on our model. Surprisingly, our model performed significantly worse with initialized weights. Table 2 displays our results.

4.4.1 Character Embedding Ablation

Character-level embeddings provided different levels of improvement between the QANet model and the BiDAF. They were much more effective when included in the BiDAF model. As seen in table 1 we improved performance in both models when adding character embeddings. Our hyperparameter-tuned BiDAF model received bumps of almost 3 points in both Dev F1 and EM scores. Consequently, the QANet model received relatively small increases in performance from character-level embeddings: just over a single point and just under a single point for F1 and EM respectively. This indicates that QANet may not be as dependent on the character level meanings of words.

5 Error Analysis

Here we provide a few specific examples of common errors in our model when answering questions.

5.1 Misidentified Specifier

As shown in the example below, the model was often unable to discern the difference between the names specifying people. This results in the model confusing the actions of two different subjects within a text and often led to the model providing incorrect answers for unanswerable questions.

Question: What sort of motion did Newcomen's steam engine continuously produce?
Context: In 1781 James Watt patented a steam engine that produced continuous rotary motion.
Answer: N/A
Prediction: rotary

[1]QANet + Char Embeds was trained using gradient accumulation with an actual batch size of 16 but an effective batch size of 32.

Model	Epochs	Batch Size	Learning Rate	L2 Decay	Hidden Size	Attention Heads	Dev F1	Dev EM
Baseline BiDAF	30	64	0.5	0	100	N/A	58	55
Tuned BiDAF	30	32	0.001	3×10^{-7}	128	N/A	62.814	59.267
Tuned BiDAF + Char Embeds	30	64	0.001	3×10^{-7}	128	N/A	65.731	62.225
QANet	10	8	0.001	3×10^{-7}	64	8	55.64	52.26
QANet + Char Embeds	20	32[1]	0.001	3×10^{-7}	96	4	56.91	53.18

Table 1: Results

Weight Initialization	Training Iterations	Number of Heads	Hidden Size	Dev F1	Dev EM
Random	800,000	4	96	56.43	53.64
Xavier and He	800,000	4	96	50.3	47.61
Xavier and He	400,000	8	128	50.3	49.59

Table 2: Weight initialization ablation

A similar phenomenon was seen with numbers, as shown below, where the question would ask about something which occurred in a given year, and the model would provide an answer that is specified as occurring in a different year.

Question: The 2002 Act granted further fiscal devolution including what?
Context: The 2012 Act conferred further fiscal devolution including borrowing powers and some other unconnected matters such as setting speed limits and control of air guns.
Answer: N/A
Prediction: borrowing power

We theorize that these problems may be caused by <UNK> word embeddings for names or numbers which make the words look equivalent. We hoped that our character embeddings would allow the model to discern between <UNK>s, but they did not fix all of the errors. With numbers this is likely due to 2002 and 2012 being off by only one digit which seems relatively insignificant to the model. This error may also be due to the similarity of GloVe representations of the confused words, as we saw one question on which the model confused South and North with East and West.

5.2 Incorrect Answer Length

In many cases where our model makes an error, it predicts an answer that is almost correct, but the answer is either too long or two short compared to the given one. This means that it is struggling to identify the correct span boundaries of a given answer, an error that would be common even for humans. Below we show two different examples of this, one where our prediction is too specific and one where it is not specific enough. In the case of the first example, the additional specifics end up being irrelevant to the question as they refer to another English name. In the second example we see that the prediction, which is shorter than the given answer, identifies the main party in question but does not realize that there are multiple possible answers.

Question: What is the Chinese name for the Yuan dynasty?
Context: The Yuan dynasty (Chinese: 元朝; pinyin: Yuán Cháo), officially the Great Yuan (Chinese: 大元; pinyin: Dà Yuán; Mongolian: Yehe Yuan Ulus[a])...
Answer: Yuán Cháo
Prediction: 元朝; pinyin: Yuán Cháo), officially the Great Yuan

Question: What tribes supported British?
Context: Further south the Southeast interior was dominated by Siouan-speaking Catawba, Muskogee-speaking Creek and Choctaw, and the Iroquoian-speaking Cherokee tribes ... The British were supported in the war by the Iroquois Six Nations, and also by the Cherokee – until differences sparked the Anglo-Cherokee War in 1758...
j**Answer:**Iroquois Six Nations, and also by the Cherokee
Prediction: Iroquois Six Nations

This error is in part due to the inconsistency of the provided answers. For example, when referring to a noun the answer sometimes includes “the” and sometimes does not.

6 Conclusion

Given the multiple QANet implementations on the Dev leaderboard which have beaten the baseline on SQuAD 2.0, we believe there are either several bugs in our implementation, or issues with our hyper-paramaters and weight initialization. As illustrated by our experiments, the QANet architecture is extremely fragile. Small changes massively effect performance, especially in comparison to the BiDAF which performed well even with the hyperparameters specified for the QANet. Given more time, we may have been able to build a successful model if we had started from scratch instead of attempting to debug a model with an issue which was very difficult to find.

Beyond debugging the model and further tuning hyper-parameters, we believe that adding our planned verifier mechanism to a working QANet would improve the model’s recognition of and response to unanswerable questions. Our verifier architecture was the simplest of those suggested by Hu et al. [5], so an increase of performance using this architecture without PCE would justify exploring the more complex architectures. One negative of adopting the more complex approach is the addition of sequential model components that would detract from the purported benefit of the QANet architecture.

Along with changing the number of layers we are also interested in using half precision floats. This would allow us to increase the size of the model or batch size, possibly resulting in better results or faster training. We experimented with gradient accumulation to the same end.

Our work in improving upon the baseline BiDAF model achieved surprising results. Simply adding character-level embeddings, along with tuning several hyperparameters, vastly improved its performance as measured by EM and F1 scores by almost 3 points in both. Had we realized the difficulties in debugging our QANet implementation earlier, we may have focused more attention in improving upon the BiDAF model. With more time, we would include the earlier discussed verifier system in the BiDAF architecture to improve its ability on unanswerable questions. We would also incorporate the BiDAF model into various ensembling techniques, especially if we were to achieve a workable QANet model.

References

[1] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.

[2] 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.

[3] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[5] Minghao Hu, Furu Wei, Yuxing Peng, Zhen Huang, Nan Yang, and Ming Zhou. Read + verify: Machine reading comprehension with unanswerable questions. *CoRR*, abs/1808.05759, 2018.

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

[7] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. Association for Computational Linguistics, 2014.

[8] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *CoRR*, abs/1802.05365, 2018.

[9] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training.

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

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

[12] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015.

[13] 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.

[14] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *CoRR*, abs/1608.07905, 2016.

[15] 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.