Stanford CS224N SQuAD IID Default Project

Tracy Cai
Department of Computer Science
Stanford University
cpcai@stanford.edu

Paul Mure
Department of Computer Science
Stanford University
paulmure@stanford.edu

Ben Rocklin
Department of Computer Science
Stanford University
brocklin@stanford.edu

Abstract

In this project, we attempt to build a state-of-the-art model for question answering on the SQuAD 2.0 dataset via combining several different deep learning techniques. We iterated off of the baseline BiDAF model with various improvements such as feature engineering, character embeddings, coattention, transformer models, and more. We had mixed success in getting all of these methodologies to fully run as anticipated and found many to not work as well as we had hoped, but we still managed to make significant improvements over the baseline by combining some of what we had implemented and performing a hyperparameter search. Our final model was quite successful on this front, achieving an F1 score of 63.517 and an EM score of 59.966 over the baseline's 58 F1 score and 55 EM score.

1 Key Information to include

- Mentor: Gita Krishna
- External Collaborators (if you have any): N/A
- Sharing project: N/A

2 Introduction

SQuAD 2.0 is a well-known and established NLP problem. The original SQuAD dataset is composed of several question-context pairs, and the goal of the problem is to be able to select the start and end locations within the context that answers the question. SQuAD 2.0 is an iteration on top of this original dataset than introduces a new metric: AvNA, or answer vs. no answer. In this new dataset, some contexts may not actually contain the answer to the question, and in the case an answer is not present, the model must learn to predict that there is no answer to the question in the given context.

This evolution to the original SQuAD dataset helped to motivate this as a more realistic problem, as it is quite clear that not every question can be answered by a single context. Moreover, given the incredibly high amount of data and finite amount of time, answering questions correctly and efficiently is an invaluable resource that is still a developing field. Many approaches have managed to make improvements, and some have even managed to create slightly different iterations improving on others. Very few, however, have tried combining these state-of-the-art approaches such as BiDAF, QA-Net, Transformers, and so on to create a hybrid model that outperforms all of these.

Consequently, our key goal was to first reimplement various successful approaches, where each is described further individually in the Approach section, in order to build an effective model for question-answering. Because we were supplied with BiDAF in starter code, we decided to mainly iterate off of this and see what improvements we could make to boost the model's overall performance. Then, once we had reimplemented many of these different improvements individually and evaluated them, we took the most successful of these and combined them to build our main model.

3 Related Work

Previously, existing work in machine reading comprehension has revolved mostly around elaborate forms of recurrent neural networks, with the BiDAF[1] model being at the forefront in terms of LSTM based question answering models. It is this model that we chose to iterate off of for our project. However, many other approaches exist.

More recently, the NLP landscape has seen the promising results obtained in transfer learning approaches involving a pretrained Transformer based network. Newer machine reading comprehension models have thus revolved around taking pretrained models such as ELMo[2], GPT[3], BERT[4], etc., and finetuning it on the SQuAD dataset. More involved models like the Retro-Reader model[5], one of many models to have surpassed human performance on the subject, takes the basic encoder-decoder concept of Transformer and improve it by adding more complex verification modules. Owing to the size of some of these models and the restriction on using pre-trained models for this version of the final project, we do not use these newer state-of-the-art models explicitly. However, we do take concepts from several of these existing models as well as others mentioned below in order to build our state-of-the-art model, and we describe what these related approaches are and how they work in-depth in the Approach section.

4 Approach

Our primary approach involved trying out several different improvements to the baseline model individually before combining the ones we found to work best in order to build an effective model. We will detail each of the individual improvements we made to the BiDAF algorithm, the methods for finetuning hyperparameters we used, and the final model we decided to build out of the individual components. Unless noted otherwise, we coded every single one of the following approaches (except for the baseline) from scratch.

4.1 Baseline

As mentioned above, our baseline is a BiDAF model. The BiDAF model is described in depth in its associate paper [1], so we recommend that readers read over this paper before continuing to read this one, as we will not describe it in detail and make several references to it.

4.2 Character-Level Embeddings

To improve the baseline model, we implemented character-level embeddings in addition to the word-level embeddings in the baseline model, as mentioned in the original BiDAF paper[1]. Specifically, we first modify the model to get the character indices in the forward function and applied an embedding layer to each index. Next, we apply a one-dimensional CNN on each of these character embeddings. We then apply a maxpooling layer over the width of the word to get a fixed size, one-dimensional output for each word. Finally, we concatenate the character embedding vectors and the pretrained GloVe word vectors to be the final input to the model.

4.3 Coattention

Owing to some of the vagueness and different definitions of coattention, we decided to implement two different varieties of coattention, detailed in the next two subsections.

4.3.1 DCN

We implemented a coattention layer as described by the authors of the Dynamic Coattention Network for Question Answering [6]. A command line argument allows this layer to be substituted in for the attention layer of the BiDAF model. A question representation $Q \in \mathbb{R}^{q \times 2H}$ and context representation $C \in \mathbb{R}^{c \times 2H}$ are passed in as arguments. We first obtain an updated version of the question representation via passing it through a tanh nonlinearity:

$$Q = \tanh(\text{Linear}(Q))$$

Addtionally, we append two trainable sentinel vectors of length $\mathbb{R}^2 H$ to Q and C, where H is the hidden size, yielding $Q \in \mathbb{R}^{(q+1)\times 2H}$ and $C \in \mathbb{R}^{(c+1)\times 2H}$, and then build an affinity matrix:

$$L = CQ^\top \in \mathbb{R}^{(c+1)\times(q+1)}$$

From here, we get two attention weights α and β and multiply by matrices to get context-to-question attention outputs a and question-to-context attention outputs b:

$$\alpha = \text{softmax}(L) \in \mathbb{R}^{(c+1)\times(q+1)}$$
$$\beta = \text{softmax}(L^\top) \in \mathbb{R}^{(q+1)\times(c+1)}$$
$$a = \alpha * Q \in \mathbb{R}^{(c+1)\times 2H}$$
$$b = \beta * C \in \mathbb{R}^{(q+1)\times 2H}$$

Finally, we obtain second-level attention output s:

$$s = \alpha * B \in \mathbb{R}^{(c+1)\times 2H}$$

We then concatenate s and a and pass this into a bidirectional LSTM, popping off the sentinel vector at this step to yield $s, a \in \mathbb{R}^{c\times 2H}$, yielding our final output:

$$out = \text{BiLSTM}([s; a]) \in \mathbb{R}^{c\times 8H}$$

4.3.2 LSTM BiDAF Attention

Owing to the similarity between the attention mechanism in the baseline code and the coattention layer as described in the DCN paper [6], we also decided to create our own novel approach to coattention. Namely, we noted that the main difference between the baseline's attention and standard coattention was the choice of using a bidirectional LSTM instead of a tensor concatenation for the attention layer output. Consequently, we created a hybrid layer by simply replacing the output of the BiDAF model ($[c, a, c*a, c*b]$) with a bidirectional LSTM output ($\text{BiLSTM}([c*a; c*b]) \in \mathbb{R}^{c\times 8H}$). This layer is, for the most part, identical to standard BiDAF attention with the exception of the last layer; thus, we redirect the reader to read through the associated section 2.4 of the BiDAF paper [1] for more detail on how the rest of this layer works.

4.4 Answer Pointer

Inspired by the paper on Answer Pointer Net[7], we seek to improve the model's performance by conditioning the end probabilities on the start probabilities. We tried two approaches to this: the first one is taking the concatenation of the outputs from the modeling and attention layers and run it through a RNN layer to produce the start probabilities; it then takes the RNN output of that layer through another RNN layer to produce the end probabilities. The second approach is to take the same concatenated output through two layers of self-attention, predicting the start and end probabilities respectively. This approach first takes the output of the modeling layers $M \in \mathbb{R}^{2H\times c}$, and the BiDAF attention layers $G \in \mathbb{R}^{8H\times c}$, and calculate the start probabilities as follows:

$$C_{start} = [G; M]$$
$$A_{start} = W_{start} \cdot \left(\text{softmax}\left(\frac{C_{start} \cdot C_{start}^\top}{\sqrt{8H}}\right) \cdot C_{start}\right)$$
$$p_{start} = \text{softmax}\left(A_{start}\right)$$

where $W_{start} \in \mathbb{R}^{1\times 8H}$ is a learnable weight parameter. Similarly, for the end probability, we have

$$M' = \text{LSTM}(M)$$
$$C_{end} = [G; M']$$
$$A_{end} = W_{end} \cdot \left(\text{softmax}\left(\frac{C_{end} \cdot C_{end}^\top}{\sqrt{8H}}\right) \cdot C_{end}\right)$$
$$p_{end} = \text{softmax}\left(A_{end}\right)$$

where again, $W_{end} \in \mathbb{R}^{1\times 8H}$ is a learnable weight parameter. This approach is inspired by the Scaled Dot-Product Self Attention described in the original Transformer architecture[8]. We ran two experiments using this Self Attention setup, the difference being that for one we take the output of the first layer (start probabilities) to be the key vectors for the second layer; and for the other, we take the output of the first layer to be the value vectors for the second layer. We kept all training parameters constant.

4.5 Self-Attention

Once again, self-attention has many definitions, so we decided to implement two versions of this. The first, named self-matching attention, stems from Microsoft's R-Net [9], while the second is the same self-attention mechanism used in transformer models. Both are implemented as an optional additional layer that may be used immediately after the standard BiDAF attention layer or a variant of coattention and are usable via command line arguments. Let the input from the previous attention layer be defined as $v_P \in \mathbb{R}^{c \times 8H}$.

4.5.1 R-Net Self-Matching Attention

Here, we use a modified version of self-matching attention as presented in Microsoft's R-Net [9]. We begin with some passage representation $v_P \in \mathbb{R}^{c \times 8H}$. We then compute a similarity matrix and attention weights as follows, where $v^\top \in \mathbb{R}^8 H$, $W1 \in \mathbb{R}^{8H \times 8H}$, $W2 \in \mathbb{R}^{8H \times 8H}$ are trainable network parameters:

$$S = v^\top * \tanh(W1 * v_p + W2 * v_p)$$
$$a = \text{softmax}(S)$$
$$c = a * v_p \in \mathbb{R}^{c \times 8H}$$

Now, the original R-Net paper uses a bidirectiona RNN over the concatenation of v_p and c (BiRNN$([v_P, c]) \in \mathbb{R}^{c \times 32H}$). However, this did not work for our output dimensionality, as we need the output hidden size to match the input size. Consequently, we instead use $out = \text{LSTM}(c) \in \mathbb{R}^{c \times 8H}$ as our output for this layer.

4.5.2 Transformer Multiheaded Self-Attention

Similar to the Transformer architecture [10], we also implemented the same multi-head attention mechanism from Attention is all you need [11]. We pass in the result of standard BiDAF attention/coattention, represented as $v_P \in \mathbb{R}^{c \times 8H}$. By definition, multihead attention is given by

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \cdots, \text{head}_h) W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Here, we pass in v_P for Q, K, and V, so $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d_{mod} \times 8H}$ and $d_{mod} = 8H/h$ and h is the number of heads to use, a tunable hyperparameter. Now, all that remains is to define the attention operation itself.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^\top}{\sqrt{8H}})V$$

Doing all of this as described above will then yield the attention output for this variant of self-attention.

4.6 Transformer

We also experimented with Transformer-based architectures. The main model we tried is implementing the original QANet architecture as it is described in the paper[10]. The only difference between our model and the one described in the paper is the omission of the convolutional layers due to hardware constraints. A reduction in the overall size of the model was also needed due to limited memory in the machines used for training. Furthermore, we experimented with an original design for combining the context and query attention output. Let N be the sequence length and d_{model} be the model dimension which is a hyperparameter; our model involves a custom decoder block that takes as inputs the output of the context encoder $C \in \mathbb{R}^{N \times d_{model}}$, the output of the query encoder $Q \in \mathbb{R}^{N \times d_{model}}$, and the output of the previous decoder layer $X_{i-1} \in \mathbb{R}^{N \times d_{model}}$. We then compute the output of the ith decoder layer $X_i \in \mathbb{R}^{N \times d_{model}}$ as follows:

$$A_i = \text{MultiHead}\left(\text{Query} = Q, \text{Keys} = X_{i-1}, \text{Values} = X_{i-1}\right)$$
$$X_i = \text{MultiHead}\left(\text{Query} = C, \text{Keys} = A_i, \text{Values} = A_i\right)$$

The rest of this novel model is the same as the QANet architecture. Due to hardware constraints, we ran one model with our custom decoder with $d_{model} = 128$ and $d_{hidden} = 128$, with 4 attention heads and 3 layers in both the encoder and decoder stack. As for the QANet, we ran three different

models, first, a small one with $d_{model} = 64$, $d_{hidden} = 64$, 4 attention heads and 3 layers; a medium one with $d_{model} = 128$, $d_{hidden} = 128$, 4 attention heads and 4 layers; and finally, a large one with $d_{model} = 256$, $d_{hidden} = 512$, 8 attention heads and 4 layers. Unfortunately, due to limited memory in our machines, we had to drastically decrease the batch size for these Transformer based models to fit, which cause significant increases in training time.

4.7 Feature Engineering

To further improve our model's performance, we used feature engineering [12]. To be more specific, we first created a word frequency map, and then for each word vector, we tagged the word's frequency at the end. However, there are tradeoffs between feature engineering and end-to-end learning. While feature engineering can be make training the model easier with the additional human guidance; it requires more human intervention and is thus less adaptable. For end-to-end learning, more emphasis is put on model autonomy; even though this makes the model harder to train, it is arguably more adaptable.

4.8 Putting It All Together

Unfortunately, not all our of experiments yielded results that actually surpassed the baseline model or were efficient enough to train in a practical amount of time for this project, as shown below in section 5.4. We took the experiments that achieved superior performance than the baseline and combined them together to form an ultimate BiDAF-based model, which we dubbed UltimateBiDAF. This involves combining the character-level embedding, the RNN variation of the Answer Pointer Net output, feature engineering, and some hyperparameter tuning. We also trained a model with the Transformer style Self Attention, but it did not produce favorable results.

5 Experiments

5.1 Data

As mentioned above, we use the SQuAD 2.0 dataset for our data, as built by Rajpurkar et. al [13]. However, we used the modified dev set as specified by the teaching team to avoid breaking the honor code. The task of the SQuAD dataset is to, given a context passage and a question, mark the indices of the starting and ending positions within the context passage that provide the answer to the associated question. If the answer to the question is not present in the context passage, the model should instead predict 0 and 0 for the start and end locations, which correlates to a prediction of "no answer." Therefore, the input of the model, concretely, will be a string representing a context passage and a string representing a question, while the output will be a pair of indices, where (0, 0) maps to no answer and (i, j) maps to an answer to the question starting at location i in the context passage and ending at location j.

5.2 Evaluation method

We use the four metrics supplied by the default project as our metrics. Our first metric is NLL, which stands for negative log-likelihood and is our loss function; here, it is the log likelihood of the start and end locations as predicted by the model. Our second metric, AvNA, stands for answer vs. no answer, and provides the model's accuracy in predicting no answer vs. an answer being present as described in section 5.1. Our third metric, EM, stands for exact match and represents the frequency in which the model outputs the perfect label verbatim as a binary outcome. Our final metric, F1, is a looser metric than EM that represents the harmonic mean of recall and precision. For instance, if our model only predicted part of the true correct answer or predicted more parts of the context outside of the true answer, F1 would be non-zero as part of the model's answer was correct, whereas EM would output 0 as a binary label. We primarily used F1 and EM to evaluate our models, although we still factored in NLL and AvNA in our experiments.

5.3 Experimental details

As described above, we ran our models above individually, tweaking hyperparameters for each subcomponent to optimize performance, and then combined the parts that worked best into Ulti-

mateBiDAF and tweaked hyperparameters once more. It would be too long to list the value of each hyperparameter we chose individually, so we will simply list the hyperparameters for the final model and list our general methodology below.

The main hyperparameters we tweaked were batch size, dropout rate, number of epochs, and hidden size. We ran hyperparameter searches for each model below by tweaking each individually, using different "scales" for each value we tried for all of them, and then ran them for enough epochs to the point where we had a relatively clear idea where it would converge and would then cancel each manually. We repeated this process for every combination of hyperparameters on a large scale and then narrowed our search to converge to the best possible hyperparameters for each. Finally, we would also note the number of epochs where the model would seem to converge without overfitting for each model by analyzing the dev NLL in Tensorboard (a rising dev NLL indicated overfitting) and adjust accordingly. Some models had extra hyperparameters, such as the number of heads to use in Multiheaded Attention, so we tuned these in much the same way.

Overall, our ideal values for each seemed to be a batch size somewhere between 16-128, depending on the model, dropout rate between 0.1-0.35, roughly 25 epochs, and the default hidden size of 100. Note that there were some experiments we could not try, such as raising the hidden size to very high values, due to time and memory constraints. For our UltimateBiDAF model, we settled on a batch size of 64, a dropout rate of 0.3, 25 epochs of training, and a hidden size of 100.

5.4 Results

Model	Dev NLL	F1	EM	Dev AvNA
Baseline	3.20	58.02	55.02	64.64
Character Level Embedding	2.80	63.72	60.70	69.82
Character Level Embedding with Feature Engineering	3.14	64.92	61.49	72.09
DCN Coattention	3.34	54.34	51.07	63.23
BiDAF Coattention	3.43	54.73	52.13	62.59
Answer Pointer: RNN	3.11	60.94	57.67	67.82
Answer Pointer: Key Vectors Self-Attention	3.10	60.72	57.37	67.22
Answer Pointer: Value Vectors Self-Attention	3.37	58.58	54.58	65.92
Self-Matching Attention	3.45	52.96	50.31	60.75
QANet Small, 4 heads	5.46	52.19	52.19	52.14
QANet Medium, 4 heads	4.84	48.19	47.40	54.16
QANet Large, 8 heads	9.87	04.96	02.00	47.84
QANet Custom Decoder, 4 heads	5.46	52.19	52.19	52.14
UltimateBiDAF with Self Attention	3.03	61.42	58.09	67.72
UltimateBiDAF without Self Attention	2.69	66.03	62.73	72.11

Note that we ran far more experiments than in the table above for hyperparameter tuning, these were just the ones we ran in optimal configuration for each all the way to convergence. We did not train multiheaded self-attention in isolation to completion, as it was clearly being outperformed by self-matching attention and QANet by a large margin.

Our UltimateBiDAF that we chose (the one without self attention) achieved an F1 score of 63.517 and an EM score of 59.966 for the Gradescope test leaderboard. It achieved 66.026 F1 and 62.729 EM on the dev leaderboard as well, reflected in the table above. The leaderboards we used were the IID SQuAD leaderboards.

Overall, our results weren't quite what we were hoping. Namely, we found it strange that our self-attention layers were achieving such low results (F1 and EM scores beneath the baseline). We ran these, transformers, and coattention past various instructors who were unable to find any bugs, so perhaps it was not our implementation but rather how we applied them that did not fit. For instance, self-matching attention might not work well in a BiDAF model, as its input and output were both different compared to the R-net model from where it came. The QANet poor performance likely stemmed from us drastically reducing the model size in order to be able to run it; due to time and space constraints, as discussed above, it was completely intractable for us to train the full QANet architecture from start to finish to convergence, so we believe this shrinking of the model may have impacted its performance as well.

Overall, our approach may have been slightly optimistic, as this project taught us that these machine learning techniques are not simple things to "slap together" in order to improve on an existing model; but rather are carefully calculated improvements that rely on certain inputs and outputs to truly be effective. Nevertheless, our improvement over the baseline using some parts of what we implemented such as character level embeddings and feature engineering shows that some small tweaks can certainly still improve an existing model by a decent margin as evidenced by the table above.

6 Analysis

Our UltimateBiDAF model appeared to be quite successful on the questions we analyzed overall. However, we did notice the model appeared to routinely "overpredict" the correct answer, yielding high recall scores but low precision and, in some cases, reducing its AvNA performance. For instance, when asked "what is the active form of vitamin D known as?", our model predicted "steroid hormone calcitriol" instead of simply "calcitriol." This pattern repeats in many other questions as well, and one notable thing is that our model never seemed to underpredict the correct output (that is, it didn't appear to ever have high precision but low recall for any output). Furthermore, we noticed that it has a tendency to predict an answer when there truly is no answer in the context passage. Thus, our model appears to be "overly optimistic" when finding an answer, yielding much information that isn't necessarily relevant to the true answer.

On the other hand, however, our model appeared to not miss the answer often when an answer was present, and it very rarely yielded passages completely unrelated to the true answer (with the exception of yielding an answer when no answer actually existed in the context). Consequently, we believe our model, qualitatively speaking, to be effective at question answering. No particular type of question seemed to give it more issues than any other. The majority of missed questions did begin with "what", but most of the questions present that we analyzed began with "what", so we believe this phenomenon was simply due to the relatively higher frequency of "what" questions rather than any particular issue the model had with these.

7 Conclusion

Through this project, we learned that while many excellent and powerful NLP tools exist, these tools cannot necessarily be used in all contexts. Some require very specific network structures to truly be effective, and trying to use them in other contexts can often times lead to lower performance than if they had not been used at all. That said, question answering is a growing field, and many established solutions can clearly be combined to greater effect, as shown by our UltimateBiDAF model. By combining different unrelated methodologies used for question-answering, we were able to build a model that surpassed standard BiDAF in performance; we believe that given more time, we may have been able to achieve even greater results. Namely, the Transformer/QANet model was held back by severe time and resource constraints. Given more effective GPUs that could hold the model in full and the time needed to train one of these to completion, we believe that we may have been able to train this model in conjunction with feature engineering and character embeddings to outperform our UltimateBiDAF model that we submitted. This is most likely the avenue we would investigate next, along with perhaps implementing R-Net from scratch to explore self-matching attention's potential further.

References

[1] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension, 2016. arXiv:1611.01603.

[2] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018. arXiv:1802.05365.

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz

Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. 2020.

[4] 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.

[5] Zhuosheng Zhang, Junjie Yang, and Hai Zhao. Retrospective reader for machine reading comprehension. *arXiv preprint arXiv:2001.09694*, 2020.

[6] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604*, 2016.

[7] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer, 2016. arXiv:1608.07905.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, page 5998–6008, 2017.

[9] Natural Language Computing Group of Microsoft Research Asia. R-net: Machine reading comprehension with self-matching networks. 2017.

[10] 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, 2018. arXiv:1804.09541.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, and Aidan N. Gomez. Attention is all you need. *CoRR, abs/1706.03762, http://arxiv.org/abs/1706.03762.*, 2017.

[12] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. 2017. arXiv:1704.00051.

[13] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Association for Computational Linguistics (ACL)*, 2018.