Iterative reasoning with bi-directional attention flow for machine comprehension

Anand Dhoot
Stanford University
anandd@stanford.edu

Anchit Gupta
Stanford University
anchitg@stanford.edu

Abstract

In this project we create an end-to-end model for question answering. Our model combines ideas from some of the best performing SQuAD models like BiDAF and iterative self matching attention along with highway networks and extensive tuning. At the time of submission our model stood 2^{nd} on the test leaderboard with a F1 score of **81.12** for the ensemble of 7 of our models and 2^{nd} among single models as well with a F1 of **78.63**

1 Introduction

Machine comprehension is an instance of a Question Answering task. We are given a context, usually a few sentences or paragraphs long and a query related to it, the machine needs to answer the query based on the information in the paragraph. The quality of the dataset was a bottleneck was such systems until very recently. The release of the The Stanford Question Answering Dataset by Rajpurkar et al. [2016] has facilitated rapid progress in this field.The SQuAD dataset contains examples of the type described above with the additional constraint that the answers to the query are exact substrings of the context. This property along with the quality and size ($\approx 100,000$) makes the dataset particularly suitable as a benchmark for Machine Comprehension systems.

Our system combines ideas from some of the best performing models on the SQuAD leaderboard. It consists of 5 main components 1) A fastText based contextual word embedding layer 2) A Bi-directional attention layer between the query, context representations 3) An iterative reasoning layer 4) A Modeling Layer to capture interactions between attention outputs of previous layers 5) Output layer. This combined with model optimization tricks like using CUDNN based RNNs etc. allow us to train much deeper models in time almost equal to that taken by the baseline provided to us. Our best single model is able to attain an Dev F1 score of 78.1 and subsequent ensembling gives us a final F1 score of 81.12 on the hidden test set.

2 Related Work

There has been a large amount of recent work on models benchmarked on this dataset. We outline some of the notable ones. Dynamic co-attention networks proposed by Xiong et al. [2016] were one of the first models to achieve impressive performance on this task, they proposed the co-attention layer which attended to both the question and context simultaneously. Seo et al. [2016] proposed the bi-directional attention layer which generates a representation which combines both C2Q and Q2C attention giving a large performance improvement. Wang et al. [2017] introduced many techniques like self matching layers and pointer networks based output layers. Reinforced-Mnemonic reader Hu et al. [2017] used a iterative self attention layer along with the a policy gradient based training procedure to directly optimize the F1 score. Wang et al. [2016] use a cosine similarity based filtering layer along with multi-perspective attention mechanism.

3 Approach

Our approach involved trying to combine parts of other state of the art SQuAD models in new and novel ways. In section 3.1 we outline the components of our best model. Section 3.3 contains descriptions of other modules that were part of intermediate models but were eventually removed due to varying reasons on the way to the final model.

3.1 Model Architecture

Our best model consists of 5 layers of smaller components (ref: Figure 1).

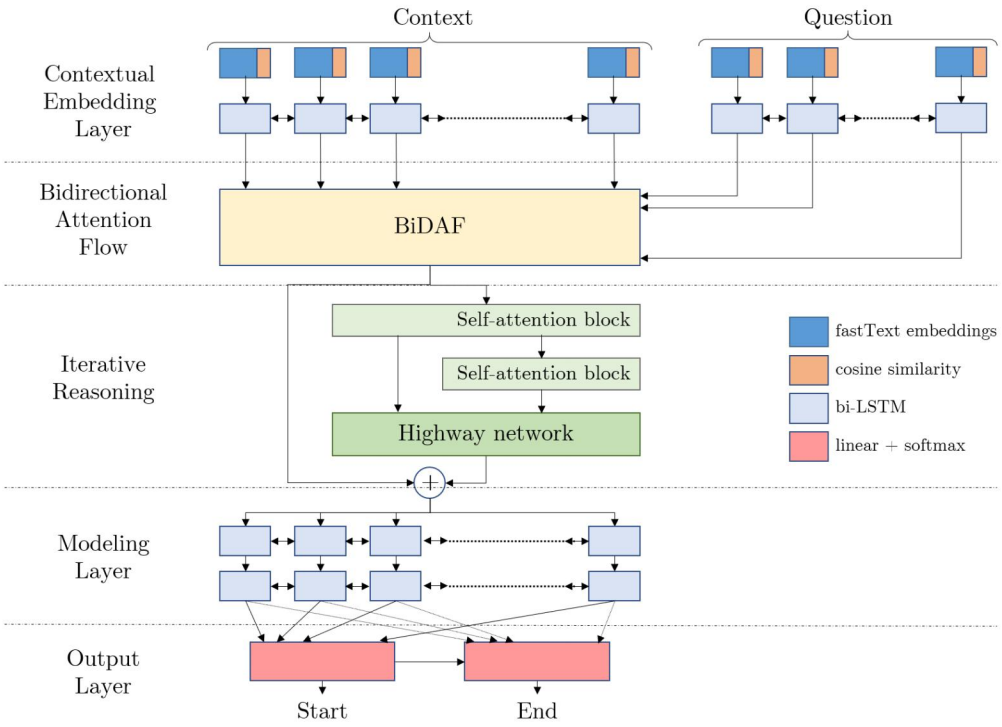

Figure 1: Architecture of best performing model

3.1.1 Contextual Embedding Layer

The first portion of this layer converts symbolic words into a continuous representation. Unlike the baseline, we use fastText word vectors Bojanowski et al. [2016] pre-trained on the common crawl dataset. These vectors are trained to incorporate sub-word information and have been shown in Bojanowski et al. [2016] to give an improvement on SQuAD models. Besides this unlike the baseline model we do not lowercase the dataset and found that using case-sensitive embeddings improved the performance of the model. These are kept fixed while training to prevent overfitting. A comparison of these embeddings can be found in Table 2.

Once these embeddings are computed for the context $\tilde{c} = \{c_1, c_2, .., c_T\}$ and query $\tilde{q} = \{q_1, q_2, .., q_T\}$, inspired by the filtering layer in Wang et al. [2016] we add a cosine similarity based feature to each of the words which is computed as follows-
For each context vectors c_i we compute and append

$$\max_j \frac{c_i^T q_j}{||c_i||_2 ||q_j||_2}$$

similarly for query vectors q_j we append

$$\max_i \frac{c_i^T q_j}{||c_i||_2 ||q_j||_2}$$

This extra feature can be interpreted as a continuous version of the exact match feature used in Chen et al. [2017] it would be 1 for a context word if it is present in the query and closer to 0 if it is really different from the query words, unlike in Wang et al. [2016] we found that appending this feature instead of multiplying gave better results.

A shared bi-directional LSTM is then used to generate contextual representations from these augmented word embeddings.

$$u_t^C = BiLSTM(u_{t-1}^C, c_t)$$
$$u_t^Q = BiLSTM(u_{t-1}^Q, q_t)$$

Sharing the parameters across the question and context results in computing representations in the same manifold which help with the various bi-linear attention modules we apply later. As a final output of this layer of we have a matrix $U^C \in \mathbb{R}^{2h \times T}$ of context representations and $U^Q \in \mathbb{R}^{2h \times T}$ of the query.

3.1.2 Bi-directional Attention flow

We exactly implement the BiDAF layer from Seo et al. [2016]. For brevity we do not go over the detailed math of this layer, good treatments of these can be found in Seo et al. [2016] and in the default project handout. This layer first computes a similarity score between all pairs of context and query words. Subsequently a row-wise softmax is computed to get the context to query attention scores using which a averaged query vector for each context word is computed. This layer differs from standard attention by also computing a query to context attention by calculating a softmax over the rows of the similarity matrix. Finally, the two attention elements are combined into R^{8h} representations for each context word. As a output of this layer we obtain a matrix $U^B \in \mathbb{R}^{8h \times T}$.

3.1.3 Iterative Reasoning

When humans are tasked with a reading comprehension task usually they do not give the answer in one-shot. The thinking process involves making a candidate solution and then re-reading the question, context iteratively refining the answer. This iterative self-matching layer attempts to simulate this. A similar idea has been used in the iterative self-aligner module by Hu et al. [2017], our approach is a simplified version of theirs. Each self attention layer consists of a Bi-directional LSTM applied to its input vectors v_t.

$$h_t = BiLSTM(h_{t-1}, v_t)$$

Then we compute a self attention logits matrix S where $S_{ij} = h_i^T h_j$. A row-wise softmax is computed using this to get the attention distribution

$$a_{ij} = \frac{e^{S_{ij}}}{\sum_j e^{S_{ij}}}$$

which is used to compute the attention weighted representation $r_t = \sum_j a_{tj} h_j$

The above self attention layer is applied iteratively and the outputs from them are combined using Highway networks Srivastava et al. [2015]. This is a specialized gated network which learns to combine outputs from different layers in very deep models dynamically. After doing a hyper parameter search we use 2 level reasoning in our final model. The output of this layer is a matrix $U^I \in \mathbb{R}^{2h \times T}$

3.1.4 Modeling layer

The modeling layer acts on the concatenated output of the bidaf layer with the final output of the iterative self attention layer. This combined representation is fed into a two-layered stacked Bi-LSTM.

$$b_t = BiLSTM^2(b_{t-1}, [u_t^B; u_t^I])$$

The output of this layer is a matrix $B \in \mathbb{R}^{2h \times T}$.

3.2 Prediction layer

The distribution of starting position of the span p^{start} is computed directly from the modeling layer output by using a linear layer followed by a softmax.

To compute the distribution of the end of the span, the logits computed for the above softmax are concatenated with the output of the modeling layer and fed into another Bi-directional LSTM. This is done so as that the model is able to learn to predict a span end probability distribution which is conditioned on the predicted span start location. As before the output of this LSTM is fed into a linear layer followed by a softmax to get the final distribution p^{end}

3.3 DP for span prediction at test time

During test time given the start, end probabilities we implemented a linear time dynamic programming based method to find the span $i, j, i \geq j$ with the largest possible $p_i^{start} \cdot p_j^{end}$ value. This gave us an improvement of around 2 points on the F1 metric over the greedy approach implemented in the default baseline.

3.4 Other approaches tried

Refer 1 for a performance comparison between our intermediate models. Detailed descriptions of each model are in subsections below.

Model	F1	EM
Baseline	43.81	34.74
Multi-head+Modeling	75.67	64.64
Bidaf+Modeling	76.83	67.2
Multi-headed Bidaf	75.21	65.48
Multi-head+Bidaf concat	76.45	67.03
Multi-head+Bidaf highway	76.49	67.1

Table 1: Performance on the dev set

Model	F1	EM
Uncased GloVe 300D	74.5	64.6
Uncased fastText wiki	74.8	64.8
Cased fastText wiki	75.83	66.2
Cased fastText crawl	76.83	67.2

Table 2: Impact of embeddings used with a constant model architecture (Dev)

3.4.1 Multi-head dot product attention

This attention mechanism was proposed in Vaswani et al. [2017], it involves multiple heads each of which first linearly down-projects the keys and values to a lower dimensional space and then computes a standard attention between them. The output across the heads is then concatenated. The benefits of this approach lie in its ability to compute attention form varying perspectives and it also benefits from parallelizability. The first model we made improved upon the baseline by using a multi-head attention layer with 4 heads and a LSTM modeling layer. As can be seen in Table 1, this model was able to achieve a huge improvement over the baseline and remains within 5% of much more complex models which we implemented later.

3.4.2 Multi-headed BiDAF

After the good results above we replaced the multi-head attention layer in the above model with BiDAF and got a slight improvement in performance. We then attempted to combine these and implemented a multi-headed BiDAF module which first down projected the context and queries in multiple different heads and then applied BiDAF to each of them.

This model was quite computationally expensive due to the fact that the BiDAF module was used multiple times which does in-efficient concat operations. Moreover the performance of this model was slightly worse than the original. We opine that this was due the fact that the BiDAF module is using a extensively hand tuned mechanism combining it with multi-head makes it harder to optimize and it interacts in a conflicting way with some of the constituents on BiDAF. The performance of this can be seen in Table 1

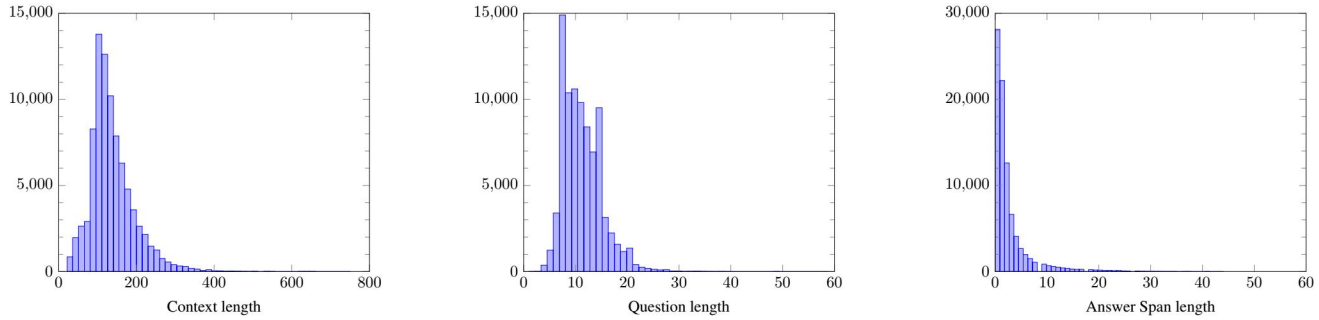

Figure 2: Distributions of context and question length for the training set.

3.4.3 Combine output of BiDAF with Dot product attention

We also tested out a model which runs both BiDAF and multi-headed dot product attention and concatenates their output. This is in row 4 of Table 1.

As an alternative to concatenating we tried using highway networks instead to combine them giving a performance of. Both of these gave performance very similar to that of just using a BiDAF layer and hence they did not justify the increased computational cost

3.4.4 Layer Normalization

Layer norm recurrent networks have been shown to improve convergence rate and generalization similar to what batch norm does for standard neural networks. We tried implementing these using the LSTM Layer norm cell in tensorflow it was around 2 times slower than the normal cell and also failed to perform as well. Using layer norm between intermediate layers of our model also failed to give a significant improvement.

4 Experiments

4.1 Dataset

We use the SQuAD dataset Rajpurkar et al. [2016] for training & evaluating the performance of our models. The training fold of the dataset has a little more than 86,000 examples.

The lengths of the contexts & questions vary significantly between examples (see Fig 2) – the largest context is more than 750 words long and the largest question has 60 words. However, from the distribution of the lengths over all examples, we observed less than 0.1% of contexts had length greater than 450 words and less than 0.07% of questions had length greater than 30 words. We set maximum threshold for the length of context to 450 & questions to 30 words in any example and pad examples with smaller context/questions to the threshold lengths. The padded values eventually get ignored during computation of attention or loss for the entire model, and hence, doesn't affect the performance.

4.2 Implementation Details

We used pre-trained case sensitive 300-dimensional pre-trained word embeddings trained on Common Crawl and Wikipedia using fastText Bojanowski et al. [2016] in the first layer of our network. These embeddings have a vocabulary size of 2 million words and were around 4.7GB in size, owing to the limited amount of GPU memory we configured our model so that the embedding lookup operation happened on the CPU, this gave huge savings in GPU ram with minimal impact on run-time.

The network was coded up using Tensorflow-1.6 and Cudnn LSTM cells were used as recurrent units to speed up training. These optimizations resulted in roughly a **4X** improvement in running time and also a improvement in GPU memory usage. These were crucial in our ability to rapidly test out models and train larger models.

Finally, an ensemble of 7 of our best single models was created by training with different random seeds and on different slices of the dataset made available to us. This was done so as to increase the diversity of the models and give a better ensemble. Their outputs were combined using majority voting and ties broken in favor of spans with higher confidence scores. Our code is publicly available at `https://github.com/Anzhit/224n-project`.

4.3 Hyper parameters

Cross entropy loss was used between the predicted start & end probabilities of the answer and the true span for training the complete network. The Adam optimizer was used for training with a learning rate of 0.001 which was halved whenever the dev-loss increased upto a minimum value of 0.0001 to ensure stable convergence. Gradients clipping was applied with a value of 5.

We used a batch-size of 32, larger values gave faster convergence but worse generalization, smaller values had excessively noisy training curves and slower convergence. A dropout of 0.15 was applied, varying this seemed to have negligible change on performance. A hidden size of 200 was used for all LSTM layers.

4.4 Results

As can be seen in Table 3, our model performs competitively with the various published models talked about in this paper.

Model(Single if not specified)	F1	EM
Multi perspective matching Wang et al. [2016]	75.1	65.5
Dynamic Coattention Xiong et al. [2016]	75.9	66.2
Bidaf Seo et al. [2016]	77.3	68
R-net Wang et al. [2017]	80.7	72.3
Reinforced Mnemonic Hu et al. [2017]	81.8	73.2
IR-Bidaf Single Model (Ours)	78.63	69.4
IR-Bidaf Ensemble (Ours)	81.12	73.17

Table 3: Performance comparison with published models on test set

5 Output Analysis & Potential future improvements

5.1 Qualitative analysis of the model errors

5.1.1 Imprecise span boundaries

In most of the incorrect cases where our model makes an error, it predicts the answer which is roughly accurate, with a few words of mis-match at either end of the span. These kinds of errors are likely to even be made by most humans, due to ambiguity about where the span boundary should lie. For instance:

Context:	Water (H2O) and carbon dioxide (CO2) are used in photosynthesis, and sugar and oxygen (O2) is made, using light energy.
Question:	What are the molecular inputs for photosynthesis ?
True Answer:	Water (H2O) and carbon dioxide (CO2)
Prediction:	Water (H2O) and carbon dioxide

5.1.2 Incorrect inference

In some cases our model completely misses the question and ends up attending to a different part of the sentence. An example where this happened:

Context:	The IPCC receives funding through the IPCC Trust Fund, established in 1989 by the UNEP and WMO, Costs of the Secretary and of housing the secretariat are provided by the WMO, while UNEP meets the cost of the **Depute** Secretary
Question:	Who funds the IPCC 's Deputy Secretary ?
True Answer:	United Nations Environment Programme
Predicted:	the IPCC Trust Fund

In the above example our model seems to get confused by the fact that the first sentence has the word ”funding” in it which is also part of the question, and ends up focusing completely on this. Interestingly if we modify the context to ”...UNEP meets the funding of the **Depute** Secretary” our model is able to make the correct prediction. This suggests that it is unable to draw a correlation between the phrase ”meet the cost” and funding. We feel using deep contextual embeddings based on language models like ELMO Peters et al. [2018] would be able to help with such cases as the SQuAD dataset is not big enough to be able to learn such language nuances.

5.1.3 Ambiguities

In some cases the question does not have a clear unambiguous answer.

Context:	”They cling to and creep on surfaces by everting the pharynx and using it as a muscular foot.”
Question:	What do platyctenida use their pharynx for ?
True Answer:	cling to and creep on surfaces
Predicted:	a muscular foot

In this case even though most humans would probably give an answer close to the true one, the predicted answer which implies that ”platyctenida use their pharynx as a muscular foot” also makes a lot of sense.

5.2 Visualizing attention weights

Figure 3 below plots the attention weights of our BiDAF layer, the first self-attention layer, the second self-attention layer and the distributions of start/end probabilities of our model respectively from left to right for one context-question pair.

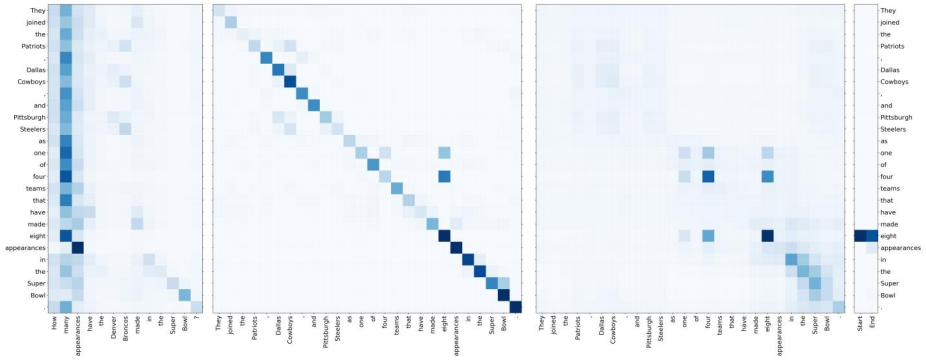

Figure 3: Visualizations of the attention values of our model on a sample context-question pair

It can be observed that in our BiDAF layer, a very high attention is paid from the question word ‘many’ to the context word ‘eight’. In general, the question word ‘many’ seems to be the most relevant in determining what the answer should be. However, the interpretability of these attention weights is low due to several layers of complex interactions till the final span prediction.

In our iterative reasoning layer, we observe strong attention weights from a word to itself in the first layer. In hindsight, we should have trained our model by not allowing our model to do this. However, we observe that the model learns this behavior (of not attending to the same word significantly) in the second deeper attention layer. In fact, we observe that similar words like ‘one’, ‘four’ and ‘eight’ attend to each other very strongly. Therefore, we leave it upto the highway network to determine which of the two attention values to choose for the final predictions.

Finally, it is observed that the output probability (start & end) is extremely high for the word ‘eight’ which is the prediction of our model and the correct answer. More plots & examples are left out of this report for brevity.

5.3 Conclusions and future improvements

In this report we propose a model architecture that combines features of some of the state of the art models like BiDAF Seo et al. [2016], Reinforced Mnemonic Reader Hu et al. [2017] and R-Net Wang et al. [2017]. This combined with fastText word embeddings Bojanowski et al. [2016] gives us competitive performance with other published state-of-the-art architectures and amongst the top models on the class leaderboard.

There are multiple other things that can be implemented to further improve the model which we could not due to lack of time. Character level CNNs to enrich the embeddings and better handle out-of-vocabulary words are used by most state-of-the-art models. Using contexualized ELMO Peters et al. [2018] embeddings to help deal with issues similar to ones highlighted in 5.1.2 have shown to give a huge improvement of 4-5 F1 points. It would be very interesting to evaluate changes in model behavior if no word is allowed to attend to itself in the self-attention layers of our model. As done in Hu et al. [2017] we can also further fine tune our converged models using policy gradient methods to directly optimize for F1 score which could be expected to give a modest improvement of around 0.5-1 F1.

Acknowledgments

We thank the course staff for all their help and Microsoft Azure for providing compute to train our models.

References

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading Wikipedia to answer open-domain questions. In *Association for Computational Linguistics (ACL)*, 2017.

Minghao Hu, Yuxing Peng, and Xipeng Qiu. Mnemonic reader for machine comprehension. *CoRR*, abs/1705.02798, 2017. URL `http://arxiv.org/abs/1705.02798`.

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

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016. URL `http://arxiv.org/abs/1606.05250`.

Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016. URL `http://arxiv.org/abs/1611.01603`.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015. URL `http://arxiv.org/abs/1505.00387`.

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. URL `http://arxiv.org/abs/1706.03762`.

Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, and Ming Zhou. Gated self-matching networks for reading comprehension and question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 189–198, 2017. doi: 10.18653/v1/P17-1018. URL `https://doi.org/10.18653/v1/P17-1018`.

Zhiguo Wang, Haitao Mi, Wael Hamza, and Radu Florian. Multi-perspective context matching for machine comprehension. *CoRR*, abs/1612.04211, 2016. URL `http://arxiv.org/abs/1612.04211`.

Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. *CoRR*, abs/1611.01604, 2016. URL `http://arxiv.org/abs/1611.01604`.