Machine Comprehension on SQuAD
BiDAF vs Coattention

Minh-An Quinn
Department of Computer Science
Stanford University
Stanford, CA 94305
minhan@stanford.edu

Ramin Ahmari
Department of Computer Science
Stanford University
Stanford, CA 94305
rahmari@stanford.edu

Abstract

This paper investigates and contrasts the use of a Bi-Directional Attention Flow model as postulated in Seo et al. (2017) [4] and a Dynamic Coattention Network Model as postulated by Xiong et al. (2018) [1] for the purpose of question-answering on the Stanford Question Answering Dataset. Best results were achieved on the test set at a 74.501 F1 score and 64.523 EM score (73.79 F1 and 63.30 EM scores on development set) with a tuned Bi-directional Attention Flow model with Modeling Layer. The Dynamic Coattention Network model without the dynamic pointing decoder performed slightly behind the Bi-Directional Attention Flow model at 71.54 F1 and 60.39 EM scores (development set).

1 Introduction

Machine comprehension is an open and booming task in natural language processing. Machine comprehension is complex, yet one of its faces can be tested via textual question answering, which requires the machine to answer a question based on a passage. The Stanford Question Answering Dataset (SQuAD), released in 2016, has become the benchmark dataset for question answering. The SQuAD task requires the machine to predict an answer span of words within a context paragraph that answers a given question. Performing well on SQuAD requires building an architecture that can understand both the content of the context paragraph and the question given, resulting in an adequate answer to the question without providing extraneous or erroneous information.

Drawing upon the Bidirectional Attention Flow model as advanced in Seo et al. (2017) [4] and a Dynamic Coattention Network model as advanced by Xiong et al. (2018) [1], we tackle the task of machine comprehension by comparing the two models and experimenting with our own variations of them.

2 Background & Literature

Given comprehensive nature of SQuAD, there is currently a vital amount of research and development regarding mastering machine comprehension on SQuAD. Currently, the best performing model on the dataset is the "Hybrid AoA Reader Ensemble", made by the joint laboratory of HIT and iFLYTEK Research, which achieves 89.281 F1 and 82.482 EM scores. This model uses an Attention-over-Attention Neural Network on top of Individual Attention and then N-best re-ranks the predictions to achieve this score [6].

This project focused on slightly older, yet fundamental works of Seo et al. (2017) and Xiong et al. (2018). A brief overview of the works is given below and more details will be given as we discuss our approach in the following section.

2.1 Bi-Directional Attention Flow

Seo et al. (2017) propose a bi-directional attention flow model whose architecture is a hierarchical, multi-stage approach at modeling representations of the context paragraph. Their paper utilizes contextual embeddings created by concatenated character-level encodings and word-level encodings that are fed into a bi-directional attention flow layer to obtain query-aware context representations. These are then fed into a modeling layer of two bi-directional LSTMs and an output layer. This model is then ensembled, resulting in 80.7 F1 and 72.6 EM scores [4].

2.2 Dynamic Coattention Network

Xiong et al. (2018) introduce the Dynamic Coattention Network. This architecture relies on word embeddings only and feeds these into a coattention encoder. This encoder aims at capturing interactions between context and queries. Instead of the modeling layer in the Bi-Directional Attention Flow, the Dynamic Coattention Network utilizes a dynamic pointing encoder that alternatively estimates start and end positions for the answering span. This model is then ensembled, resulting in 80.4 F1 and 71.2 EM scores [1].

3 Approach

This paper's Bi-Directional Attention Flow (BiDAF) model consists of five layers: a word embedding layer, an encoding layer, a bi-directional attention flow layer, a modeling layer, and an output layer. Although the original paper by Seo et al. uses an additional character embedding layer added to the word embedding layer, due to time constraints and as mentioned in section 6.3.1 "Conclusion - Future Works - Attempted", the character embedding layer was not implemented in our final solution in order to spend efforts exploring hyperparamterization of the BiDAF model. Figure 1 showcases the overall architecture of this model.

The implemented Dynamic Coattention Network (DCN) model consists of four layers: a word embedding layer, an encoding layer, a coattention layer, and an output layer. Although the original paper uses an dynamic pointing decoder instead of our simple output layer, we used the simple output later due to issues implementing the dynamic pointing decoder (explained in further detail in section 6.3.1 "Conclusion - Future Works - Attempted"). Figure 1 showcases the overall architecture of this model.

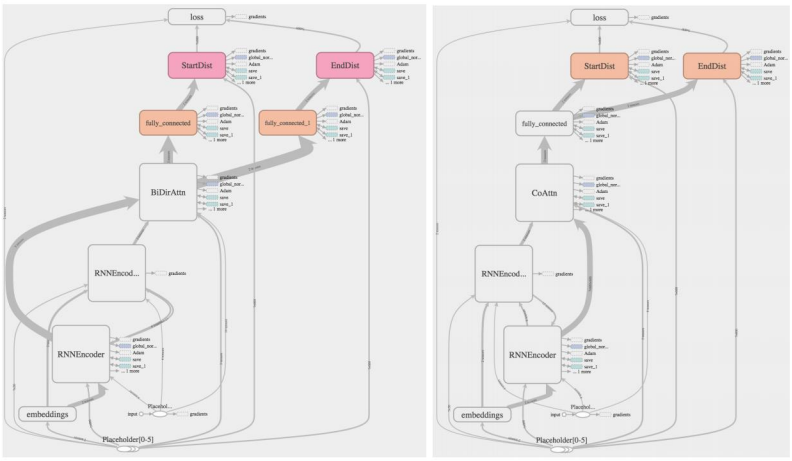

Figure 1: Overall Architectures of BiDAF Model (left) and DCN Model (right)

3.1 Word Embedding Layer

The word embedding layer maps context and question tokens into a higher dimensional space. We used 100-dimensional pre-trained GLoVe word vectors (Pennington et al. 2014) [3], which preserves the meaning of the word, from their co-occurrence information.

3.2 Encoding Layer

The encoding layer captures the interactions between words and their surrounding words. Our encoding layer utilizes a single layer bi-directional LSTM with dropout, to capture the interaction of a specific word with words both before and after it. We encode both our context paragraphs and questions separately, although the weights for the context paragraphs and questions are shared.

3.3 Bi-Directional Attention Flow Layer

The bi-directional attention layer creates two attention matrices: the context-to-query (C2Q) attention matrix and the query-to-context (Q2C) attention matrix. The C2Q attention matrix shows which words in the context are relevant to each query word, while the Q2C attention matrix shows which words in the query are relevant to each context word. We compute the bi-directional Attention Flow Layer as specified in the paper by Seo et al. (2017) [4]. Given context hidden states $c_1, ..., c_N$ and question hidden states $q_1, ..., q_M$, we first compute a similarity matrix $S_{i,j}$, which calculates the similarity between each pair of context and question hidden states.

$$S_{i,j} = w_{sim}^T[c_i : q_j : c_i \circ q_j]$$

We then compute the C2Q attention matrix by taking the row-wise softmax of the similarity matrix and then computing the weighted sum of the attended question hidden states.
We compute the Q2C attention matrix similarly by taking the softmax of the column-wise max of the similarity matrix. Next, we compute the weighted sum of the attended context hidden states.
Finally, to get the bidirectional attention output, we simply concatenate the context hidden state, the C2Q attention distribution, the element-wise multiplication of the context hidden state and the C2Q attention distribution, and the element-wise multiplication of the context hidden state and the weighted sum of the attended context hidden states [4].

3.4 Coattention Layer

The coattention layer, similar to the BiDAF paper, computes two attention matrices for both the context paragraph and the question (C2Q and Q2C). The coattention layer then combines information from both attention matrices. Given context hidden states $c_1, ..., c_N$ and question hidden states $q_1, ..., q_M$, we first compute the projected question hidden states by applying a tanh-non-linearity to the question hidden states.

$$q'_j = tanh(Wq_j + b)$$

Next, we add trainable sentinel vectors to the question and context hidden states and compute the affinity scores for every pair of question and context words.

$$L_{i,j} = c_i^T q'_j$$

We then compute the C2Q attention matrix by taking the row-wise softmax of the affinity matrix and doing a weighted sum of the attended projected question hidden states.
Similarly, we compute the Q2C attention matrix by taking the column-wise softmax of the affinity matrix and doing a weighted sum of the attended context hidden states.
We then combine the C2Q attention matrix (s) and the Q2C attention matrix by doing a weighted sum of the Q2C output with the context-to-query attention distribution. To get the coattention output, we simply concatenate the combined C2Q attention matrix and the Q2C attention matrix and C2Q attention matrix (s) the and feed it through a bidirectional LSTM. The bidirectional LSTM serves to capture the temporal relationships of the coattention context [1].

3.5 Bi-Directional Attention Flow Modeling Layer

Our BiDAF model has an additional modeling layer inserted before the final output layer. This consists of two bi-directional LSTMs. The modeling layer uses the query-aware representations of the context words output by the BiDAF layer to capture interactions between the context words conditioned on the query.

3.6 Output Layer

The output layer produces the answer span by predicting the answer start and end indices within the context paragraph. The output layer applies a linear downward projection, then a softmax of the sequence to gain a probability distribution of the start / end index over the entire paragraph.

3.7 Implementation Details

Though the layers were implemented as specified in the papers, our experiments showed that the best hyperparameters were different than those mentioned in the papers. Both of our models used an Adam Optimizer (over the Adadelta optimizer as suggested by Seo et al. (2017) [4] with a learning rate of 0.001. Our best BiDAF model had a dropout of 0.2, hidden size of 200, and a batch size of 60. The best DCN model had a dropout of 0.15, hidden size of 300, and a batch size of 60. Both models were trained to 20 epochs (about 27.5K iterations).

4 Experimentation

4.1 Stanford Question Answering Dataset

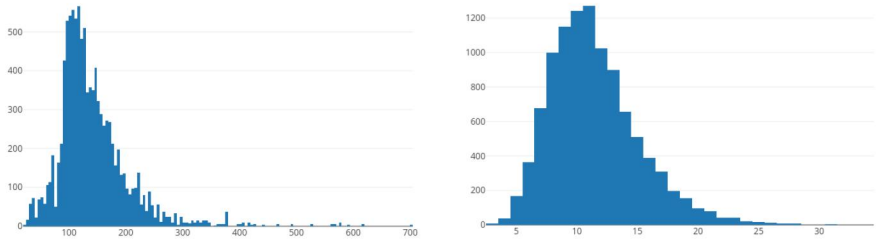

Figure 2: Histograms of Context Word Length (left) and Question Word Length (right)

The Stanford Question Answering Dataset (SQuAD) is comprised of 536 Wikipedia articles and over 100,000 question-answer pairs, making it a significantly large reading comprehension dataset. Questions and answers were set up so that answers would be spans within the relevant Wikipedia article. Therefore, answering questions does not require text generation, but rather selecting the "start" and "end" of a section in the context paragraph. The dataset was partitioned into a training set (80%), a development set (10%) and a testing set (10%) [5].

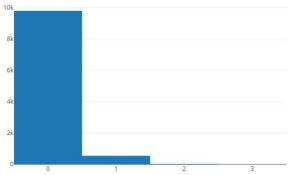

Figure 3: Histogram of Context Word Length in Buckets of Size 150 Words

Looking at the distribution of context and question lengths (Figure 2), it is evident that using a maximum context length of 600 and question length of 30, one is able to encompass almost all examples. Because of that, this project ran most of the experiments at those lengths. However, one could dramatically reduce the context length and question length to 150 and 20, respectively, and still encompass more than 95% of examples (see Figure 3). Therefore, this project also experimented with these greatly reduced context and question lengths. As further explained in section 6.2, these reduced lengths allowed the models to maintain similar F1 and EM scores with significantly reduced training times - training in less than 1/5 of the time.

4.2 Parameter Tuning

We experimented with various parameters including: optimizer, hidden size, batch size, learning rate, dropout rate, and the number of layers in our encoding and modeling layers.

In addition to the Adam Optimizer, we tried a plethora of other optimizers including: AdaDelta, Gradient Descent, Proximal Gradient Descent, Adagrad, Proximal Adagrad, "Follow the Regularized Leader" (Ftrl), and RMS prop. Though we read literature suggesting that these optimizers could be promising, we did not have enough time (and resources) to tune the optimal learning rates for these optimizers and therefore, results were limited at learning rates of 0.01. RMS prop performed quite well and warrants more investigation and learning rate tuning. Surprisingly, the AdaDelta optimizer, which was used in the original BiDAF paper, performed poorly compared to the Adam Optimizer, even when used with the learning rate cited in the paper (0.5 [4]). The added smoothing on Adadelta might lead to vastly more smoothed and slower convergence and the amount of iterations / epochs ran might not have been enough to experience improvements by the Adadelta optimizer.

A significant amount of time was also spent exploring the effects of different hidden sizes. In general, increased performance was achieved with increased hidden size. However, we were limited in our ability to increase hidden size due to two constraints. The first and most evident constraint was memory. To counter this, we would reduce the batch size. However, we had to balance having a larger hidden size and smaller batch size, resulting in very noisy gradient updates and highly fluctuating loss, or having a smaller hidden size and larger batch size, rendering the model less expressive. Furthermore, while increasing the hidden size adds to the complexity and expressiveness of the model, it also increases the chance of over-fitting and thus had to be changed with forethought.

4.3 Evaluation Methodology

To evaluate our models, we compared the F1 and EM scores of the models, as well as quantitative results, as described further in the following section.

5 Results

5.1 Quantitative Evaluation

For the implemented Bi-Directional Attention Flow model, the biggest boost in performance was observed from the introduction of the modeling and output layers as postulated in Seo et al. (2017) [4]. Since the modeling layer learns the interactions between query-aware context representations, it is able to attend to parts of the context that are most relevant to answering the question. The introduction of LSTM cells rather than GRU cells for the RNN encoding further enhanced results. After reading several research papers such as Chung et al. (2014) (which could not conclude beneficial scenarios of using LSTMs over GRUs) [2] we are inconclusive as to why this change boost scores. However, we hypothesize that the absence of a second non-linearity in the GRUs could be the cause of an inadequate account for non-linearity which could be the reason for the increase in performance when switching to LSTM cells.

Furthermore, the addition of an LSTM layer in the modeling layer of the BiDAF lead to a decreased performance. This must have been due to an overt amount of non-linearities that could cause over-fitting.

Moreover, the dropout rate of 0.2 as suggested by Seo et al. (2017) [4] was adequate. A higher dropout rate of 0.3, while aimed to regularize more, lead to a decreased performance - the effect of the dropout must have been to strong at that point and rather than avoiding over-fitting, the model failed to learn accurately. Table 1 showcases these results.

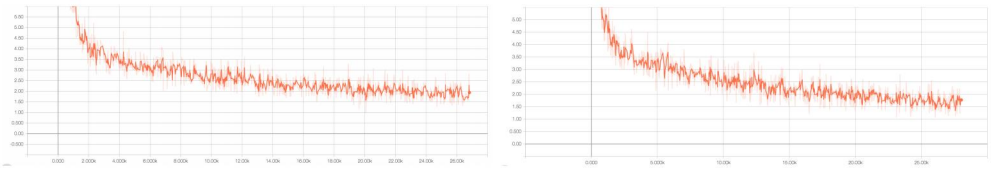

Figure 4: Loss Progression for BiDAF Model (left) and DCN Model (right)

For the Dynamic Coattention Network (DCN) model, a marked increase in performance was observed with a switch from GRU to LSTM encoder. The added non-lineary in LSTMs for encoders seems to

Model	F1	EM
Baseline - Simple Attention (Default project)	43.38	34.55
Baseline - Linear Regression (Rajpurkar et al., 2016)	51.00	40.00
BiDAF + GRU Encoding (hidden size 100)	50.32	40.88
BiDAF + LSTM Encoding (hidden size 100)	51.82	42.04
BiDAF + LSTM Encoding + Modeling (+1 LSTM) & Output (Hidden Size 200)	60.79	49.68
BiDAF + LSTM Encoding + Modeling & Output (Hidden Size 200) + 0.3 Drop	62.57	51.43
BiDAF + LSTM Encoding + Modeling & Output (Hidden Size 100) + 0.2 Drop	71.98	61.79
BiDAF + LSTM Encoding + Modeling & Output (Hidden Size 200) + 0.2 Drop	73.79	63.30

Table 1: Bi-Directional Attention Flow Model Deconstruction on the SQuAD Development Set

Model	F1	EM
Baseline - Simple Attention (Default project)	40.00	51.00
Baseline - Linear Regression (Rajpurkar et al., 2016)	51.00	40.00
DCN + biGRU encoder	55.35	43.55
DCN + biLSTM encoder + BiDAF Modeling Layer (Hidden Size 200)	58.27	45.61
DCN + biLSTM encoder (Hidden Size 200)	71.42	59.90
DCN + biLSTM encoder (Hidden Size 300)	71.54	60.39

Table 2: Dynamic Coattention Network Model Deconstruction on the SQuAD Development Set

be of even larger benefit to DCNs than BiDAFs (see previous discussion on GRUs and LSTMs). With just the coattention layer, this model is able to perform almost to the same caliber as the full BiDAF model with modeling layer, showing how powerful the coattention layer is as this model has no running implementation of the dynamic pointing decoder. An increase in hidden size past the 200 suggested units of the original paper [1] to 300 also seemed profitable.

Within the DCN model, we attempted to add a modified modeling layer (instead of a dynamic pointing decoder) as postulated by Seo et al. (2017) [4] after the Coattention layer. However, this resulted in a marked decrease in performance. We hypothesize that this is due to the strong added non-linearity given by the two LSTMs of the BiDAF modeling layer was too much for coattention's already specific single model, leading to over-specialization and uncessary inter-correlations. Table 2 showcases these results.

5.2 Qualitative Evaluation

Examining the examples and answers produced by the baseline model of simple attention, we can see that the model often makes the mistake of including extraneous information, even when the question asks specifically for a single entity or word.

```
QUESTION: what is the only district in the cbd to not have
" downtown " in it 's name ?
TRUE ANSWER: south coast metro
PREDICTED ANSWER: central business districts ( cbd ) include
downtown los angeles , downtown san diego , downtown san bernardino
, downtown bakersfield , south coast metro and downtown riverside
```

The output above shows that the model clearly misses the the meaning of the sentence, instead simply listing all the districts in the CBD. Deeper contextualization and a more complex model would be helpful. Another naive mistake that the baseline model made was that it would sometimes select the same start and end position, rendering the answer span completely blank.

Though our best model (BiDAF) still struggled on some specific questions, examining the output of the model shows that it avoided these naive mistakes and generally struggled due to more nuanced reasons:

```
QUESTION: what was the idealized value of imperialism ?
TRUE ANSWER: philanthropy
```

```
PREDICTED ANSWER: idealism and philanthropy
```

The output above shows that the model does not fully understand the fine-grained nature of asking for a singular answer (value) or several answers (values). Getting our character embeddings to work might help here as morphemes can be accounted for and thus the absence of the ”s” could be realized into the algorithm more significantly.

```
QUESTION: what network showed a doctor who film ?
TRUE ANSWER: the fox network
PREDICTED ANSWER: fox
```

The output above shows that the model still struggles with established names made up of very common words. In this case, more specialization is necessary. But it also is important to generalize, which it successfully does here, yet that breaks ”fox network” into ”fox” and ”network”. Named Entity Recognition tagging could help her as well.

```
QUESTION: when did luther write a german mass ?
TRUE ANSWER: early 1526
PREDICTED ANSWER: 1526
```

The output above shows that the model still struggles to correctly associate adjectives with nouns, dates, etc. This can only be accounted for if the model learns some form of syntactic grammar. Augmenting our data by adding tags to works that identify them as a certain type (adjective, noun, etc.) could be helpful here.

6 Conclusion

6.1 Insights

Our experimentation has clarified to us that the Bi-Directional Attention Flow model is very powerful despite it’s rather simple approach of bi-directional attention. At the same time, it is evident that the coattention layer is in itself a powerful tool for natural language understanding as it performs almost as good as the BiDAF model despite not having its dynamic pointing decoder layer.

The addition of LSTMs in the RNN encoder boosted performance for both the BiDAF model and (significantly) the DCN model. In addition, an increase in the hidden size by about 100 units increased performances across both as well. This showcases the benefit of increased complexity for the deeply complex task of machine understanding of natural language. Adding non-linearities and complexities; however, should be treated carefully as we have seen that by adding LSTM layers we can also reduce the overall performance due to overfitting.

Moreover, our best model still struggles to identify syntactic grammar rules, understand entities, respond to plurals / morpheme structures and understand multiple answer options. As discussed, character embeddings and data set augmentation with type and entity tagging could be helpful in addressing these issues in the future.

6.2 Alternative Goals

We explored ways to make the algorithm train faster while achieving comparable performance. When analyzing the dev set for the distribution of context paragraph length, question length, and answer span within the context paragraph, we noticed that all of the distributions were skewed to the right. Most context paragraphs were 300 words or less. Furthermore, most question lengths were smaller than 20 words and most answer spans fell within the first 150 words of the context paragraph. While we initially created long context paragraph and question lengths to encompass all training and development examples, we concluded that we could significantly decrease the maximum lengths of context paragraphs and questions, while still encompassing the majority of the needed context paragraph, question, and answer spans. Having shorter maximum lengths would reduce our training time because the data would be significantly smaller. Additionally, shorter lengths would make it easier for the model to create intercorrelations and deductions from the reduced data set. On our best model, we made reduced training

time by 80% and achieved an F1 score of 67.43 and an EM score of 57.30. This model ran to 20 epochs in 7.5 hours, compared to our other BiDAF and DCN models which took about 36 hours for 20 epochs.

7 Future Work

7.1 Attempted

Along with our implemented architectures, we also implemented additional features that did not fully work. If given additional time, we would want to continue working on these features.

7.1.1 Character Embedding Layer

In addition to word embeddings, we attempted to train our own character embeddings to augment our encoder. We used an alphabet containing all alphanumeric characters, commonly used special characters and the pad and unknown tokens. We randomly initialized the character embeddings and trained them using a convolutional neural network (CNN). We attempted to concatenate our character embeddings to the word embeddings in both our BiDAF and DCN models, however, our character embeddings led to a decrease in performance for both models. Since the literature indicates that character embeddings should improve our BiDAF model by 2-3 percentage points, we believe the performance of our models decreased because we were incorrectly padding in our data_batcher.py file. If given more time, we would look into fixing our character embedding layer as we believe the added granularity allowed by the morphology of the words could be beneficial to the performance of the algorithms.

7.1.2 Dynamic Pointing Decoder

We implemented the Dynamic Pointing Decoder, as specified in Xiong et al. (2018) [1]; however, the paper was quite vague on the specifics of the decoder implementation, specifically the initialization of the estimated start and end indices (u_{s_0} and u_{e_0}). We attempted to randomly initialize these but ran into dimension issues. If given more time, we are very interested in trying to correctly implement this portion of the model, as we know that it would give us a significant increase in performance for our DCN model, given that the spanning is currently non-dynamic. We also aim to incorporate a modified Dynamic Pointing Decoder into the BiDAF model for increased performance as we believe it could also benefit from more advanced, dynamic span realizations.

7.1.3 L2 Regularization

We attempted to add L2 regularization to our loss in order to prevent over-fitting. However, we believe that we implemented our regularization incorrectly, as it lead to a significant reduction in performance overall.

7.2 Hypothesized

In addition to the attempted changes as explained above, we also realized that, as evident in Figure 5, the answer spans are limited to around 10 words with a heavy right skew, leaving most answer spans to be below three words. We would like to incorporate this rather manual information into a dynamic span estimation procedure that re-establishes itself if a predicted answer span exhausted the answer span preset manually (such as in this case 10) as we believe this could avoid answers that are too long and vastly improve the EM score.

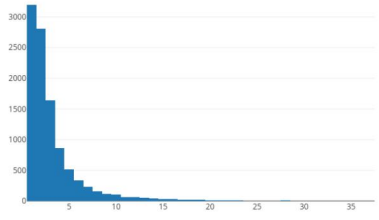

Figure 5: Histogram of Answer Word Length

Furthermore, we would like to investigate the correlation between the answer span of certain types of questions (e.g. "why" or "what" questions). If there is a correlation, we would also like to feed that into our span prediction and re-orient the span size if it goes astray based on our manually inputted features that we derived from looking at the data.

References

[1] R. Socher C. Xiong V. Zhong. “Dynamic Coattention Networks For Question Answering”. In: *ICLR* ().

[2] K. Cho J. Chung C. Gulcehre and Y. Bengio. “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling”. In: *NIPS* ().

[3] R. Socher J. Pennington and C. Manning. “GloVe: Global Vectors for Word Representation”. In: *ACL* ().

[4] A. Farhadi M. Seo A. Kembhavi and H. Hajishirzi. “Bidirectional Attention Flow for Machine Comprehension”. In: *ICLR* ().

[5] K. Lopyrev P. Rajpurkar J. Zhang and P. Liang. “SQuAD: 100,000+ Questions for Machine Comprehension of Text”. In: *EMNLP* ().

[6] S. Wei S. Wang T. Liu Y. Cui Z. Chen and G. Hu. “Attention-over-Attention Neural Networks for Reading Comprehension”. In: *ACL* ().