Co-Attention with Answer-Pointer for SQuAD Reading Comprehension Task

Tommy Fan, Mindy Yang, Chenyao Yu
Stanford University
`{t0msta, mindyang, chenyaoy}@stanford.edu`
`Codalab username: chenyaoy`

Abstract

The recently published Stanford Question Answering Dataset (SQuAD) dataset provides NLP researchers an interface to evaluate machine reading comprehension models. We explore a variety of previously proposed architectures and implement our own in an attempt to maximize performance on the SQuAD dataset. Our most successful architecture combined ideas from two research papers, Machine Comprehension Using Match-LSTM and Answer Pointer and Dynamic Coattention for Question Answering. We use the encoding and coattention layers from Dynamic Coattention Networks and the Pointer Net layer from Match-LSTM for prediction. Our model achieved 60.096% FM and 47.131% EM.

1 Introduction

The Stanford Question Answering Dataset (SQuAD) is a dataset that contains question, context paragraph and answer triples. Each question is paired up with a relevant paragraph which contains what the answer should be. These questions and answers were generated by crowd-workers on Wikipedia articles and SQuAD has around 100k such pairs. Our goal is to build a neural network architecture that, when fed a question-paragraph pair, generates an answer span within that context paragraph that answers the question.

Our general approach to solving the problem is as follows. We first convert words to vectors using pretrained GloVe vectors [2]. Then the question and paragraph vectors are each fed through a Long Short-Term Memory (LSTM) network to obtain hidden encodings of both. To create an effective reading comprehension system, the paragraph needs to be read along with the question. We replicate this behavior by using an attention model to combine the question and context vectors. This gives us how relevant each word in the context paragraph is to the question. Lastly, we run the result through another layer of LSTMs to make a final prediction on the answer span (specifically the starting and ending indices of the span within the paragraph).

2 Background and Related Work

Our research began with Chen et al's 2016 paper on a similar reading comprehension task, which involved reading Daily Mail/CNN articles and filling in a missing word in their summaries [6]. In the paper, the authors proposed an architecture that had three layers: encoding, attention, and prediction. In the encoding layer, they used a recurrent neural network (RNN) to encode the passages and questions. For the attention layer, the authors used a simple bilinear term to compute the similarity score between the question q and the ith word of the passage, p_i. The scoring function they considered was $\alpha_i = qW_sp_i$ where W_s is a trainable matrix. Finally for prediction, Chen et al performed a simple linear transformation of the attention vector and picked the index with the highest value as the answer choice. Although this paper's implementation details were simple and the problem was different, we drew ideas from it, such as separating our architecture into encoding, attention and prediction layers.

We followed up by exploring another paper that uses bidirectional attention flow for machine comprehension (Seo et al). This paper extends ideas mentioned in Chen's paper by creating two different attention layers, one for context-to-query attention and another for query-to-context attention. They also run the attention vectors through an additional LSTM that further encodes them. The outputs of the LSTM are encodings of the paragraph words that are conditioned on the question - they are context-aware of both the question and paragraph words. As for the prediction layer, this paper performs a similar linear transformation to get the starting index of the answer span. To predict the answer's ending index, however, the result is fed into another LSTM and a final linear transformation is applied. Our baseline model was primarily based off of Seo et al's bidirectional attention flow architecture.

Our final model was inspired by the ”Match-LSTM with Answer Pointer” structure. In their design, Wang et al. used match-LSTM to encapsulate attention scores. Whereas Seo et al.'s model multiplies the question and paragraph encoding matrices, match-LSTM applies a non-linearity (tanh) after combining question and paragraph encodings. It also feeds the result into a LSTM cell for each word in the paragraph during computation of the attention vector, instead of simply feeding the end result into one like what Seo et al did. To make predictions, this model uses answer pointers. We opted for the boundary model in this case, which creates only two probability vectors, corresponding to the starting and ending indices of the answer.

3 Approach

For our final model, we combined the ”Machine Comprehension Using Match-LSTM and Answer Pointer” model with the ”Dynamic Coattention Networks for Question Answering” model [1][2]. Specifically, we took the encoding and coattention layers from the second paper and used the boundary model prediction layer from the first paper.

3.1 Question Paragraph Encoding

To represent the question and the paragraph, we encode both separately using two single direction LSTMs. We then applied a nonlinearity onto the question representation in order to allow for variation between the question encoding space and the document encoding space [2]. W^Q and b^q are parameters to be trained.

$$\begin{aligned}
H^{q\prime} &= \overrightarrow{LSTM}(Q) \\
H^q &= tanh(H^{q\prime}W^Q + b^q) \\
H^p &= \overrightarrow{LSTM}(P) \\
H^q &\in \mathbb{R}^{Q\times l} \\
H^p &\in \mathbb{R}^{P\times l}
\end{aligned}$$

Next we created an affinity matrix L to represent scores corresponding to both question and context paragraphs. Using the affinity matrix, we generate attention weights A^q and A^p. Then we compute contexts C^q and C^p . The contexts are created in order to represent both the attention layer and the

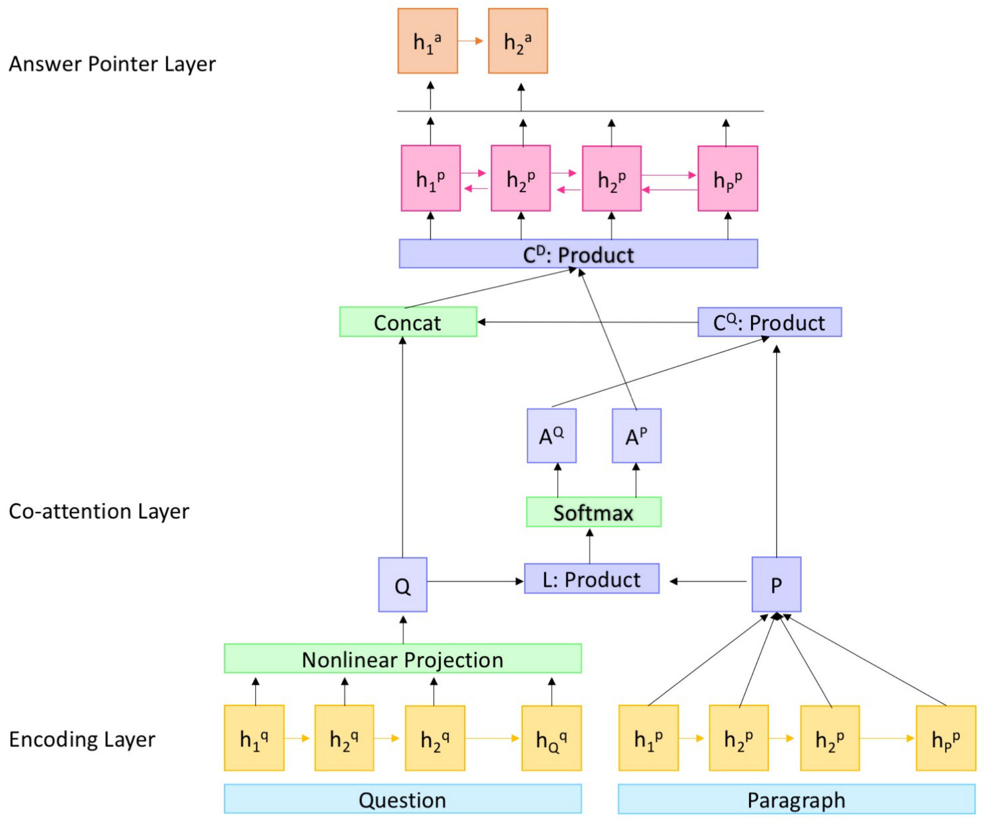

Figure 1: Graphical representation of our model architecture

encoded question and paragraphs.

$$\begin{aligned}
L &= H^p(H^q)^T, & L &\in \mathbb{R}^{P\times Q}\\
A^q &= softmax(L), & A^q &\in \mathbb{R}^{P\times Q}\\
A^p &= softmax(L^T), & A^p &\in \mathbb{R}^{Q\times P}\\
C^q &= H^{p^T} A^q, & C^q &\in \mathbb{R}^{l\times Q}\\
C^p &= [H^q; C^q]A^p, & C^p &\in \mathbb{R}^{2l\times P}
\end{aligned}$$

We then feed a combination of the context matrices with the document into a bilateral LSTM to incorporate the co-attention and temporal information.

$$\begin{aligned}
&H^r = BiLSTM([H^p; C^p])\\
&H^r \in \mathbb{R}^{2l\times P}
\end{aligned}$$

3.2 Answer Pointer Layer

We use the answer pointer layer to predict the start and indices as the answer to a question. This layer uses the H^r we produced in the attention step to produce the answer span. We use the boundary model described in Wang et. al's paper to predict the starting and ending indices. The LSTM is created with these equations. W^a, b^a, V, v, c are all parameters to be learned. The answer pointer layer adds the nonlinearity tanh and normalizes to report a score.

$$\begin{aligned}
F_1 &= \tanh(H^r V + (h_0^a W^a + b^a) \otimes e_p) \\
\beta_{start} &= softmax(F_1 v + c \otimes e_p) \\
h_1^a &= LSTM(H^r \beta_1, h_0^a) \\
F_2 &= \tanh(H^r V + (h_1^a W^a + b^a) \otimes e_p) \\
\beta_{end} &= softmax(F_2 v + c \otimes e_p) \\
V &\in \mathbb{R}^{2l \times l} \\
W^a &\in \mathbb{R}^{l \times l} \\
b^a, v &\in \mathbb{R}^l \\
c &\in \mathbb{R}
\end{aligned}$$

The β's give us a probability distribution for the indices. For example, to get the starting index, we would output $\text{argmax}_i \beta_{start_i}$. To train the model, we used cross entropy loss.

4 Experiments

First we implemented the baseline described in the assignment handout. However, our baseline had poor results, far below a logistic regression baseline. The poor results were mostly due to an overly simplified model.

Then we started out using the Match-LSTM architecture for the attention layer, but soon ran into some obstacles. After training our model for 5 epochs, the highest F1 score we achieved on the validation set was only 38.1% and the EM score was 27.4%. Aside from the poor performance, Match-LSTM also took very long to train since there is an additional bidirectional LSTM instead of just matrix multiplication.

Although it was replaced by the co-attention approach described above, we still present this technique here:
For the forward direction

$$\begin{aligned}
\overrightarrow{\mathbf{G}}_i &= \text{tanh}(\mathbf{H}^q \mathbf{W}^q + (\mathbf{h}_i^p \mathbf{W}^p + \overrightarrow{\mathbf{h}}{}_{i-1}^{r^T} \mathbf{W}^r + \mathbf{b}^{p^T}) \otimes \mathbf{e}_Q) \\
\overrightarrow{\alpha_i} &= \text{softmax}(\overrightarrow{\mathbf{G}}_i \mathbf{w} + b \otimes \mathbf{e}_Q) \\
\overrightarrow{z}_i &= \begin{bmatrix} h_i^{p^T} \\ \overrightarrow{\alpha_i}^T \mathbf{H}^q \end{bmatrix}
\end{aligned}$$

To get the next state $\overrightarrow{\mathbf{h}}{}_i^r$, we feed it into a LSTM:

$$\overrightarrow{\mathbf{h}}{}_i^r = \overrightarrow{LSTM}(\overrightarrow{z}_i, \overrightarrow{\mathbf{h}}{}_i^r)$$

The parameters are of the following shape:

$$\begin{aligned}
\mathbf{W}^q, \mathbf{W}^p, \mathbf{W}^r &\in \mathbb{R}^{l \times l} \\
\mathbf{b}^p, \mathbf{w} &\in \mathbb{R}^l \\
b &\in \mathbb{R}
\end{aligned}$$

For the backward direction

$$\begin{aligned}
\overleftarrow{\mathbf{G}}_i &= \text{tanh}(\mathbf{W}^q \mathbf{H}^q + (\mathbf{W}^p \mathbf{h}_i^p + \mathbf{W}^r \overleftarrow{\mathbf{h}_{i-1}^r} + \mathbf{b}^p) \otimes \mathbf{e}_Q) \\
\overleftarrow{\alpha_i} &= \text{softmax}(\mathbf{w}^T \overleftarrow{\mathbf{G}}_i + b \otimes \mathbf{e}_Q)
\end{aligned}$$

The output that we feed into the next layer is

$$H^r = \begin{bmatrix} \overrightarrow{\mathbf{h}}{}_1^r, \overrightarrow{\mathbf{h}}{}_2^r, \ldots, \overrightarrow{\mathbf{h}}{}_P^r \\ \overleftarrow{\mathbf{h}}{}_1^r, \overleftarrow{\mathbf{h}}{}_2^r, \ldots, \overleftarrow{\mathbf{h}}{}_P^r \end{bmatrix}$$

As the model approached the end of the first epoch, we noticed that our training losses were inconsistent across mini-batches. After printing the global gradient norm of all our trainable variables, we saw that large gradient norms (>10.0) heavily increased the training loss. This would essentially reset much of the learning done in previous mini-batches. In order to fix this, we implemented gradient clipping.

```
$\hat{g} \leftarrow \frac{\partial E}{\partial W}$;
**if** $||\hat{g}|| \geq threshold$ **then**
    $\hat{g} \leftarrow \frac{threshold}{||\hat{g}||}\hat{g}$;
**end**
```

Algorithm 1: Pseudo-code for norm clipping in the gradients whenever they explode

We also experimented with different optimizers. We started with Adam and then switched to the Adamax optimizer because it adjusts learning rates for each variable. Adamax is similar to Adam except that the second order momentum is replaced by an infinite order momentum. We chose to experiment with Adamax because Adamax is said to be more stable with sparse data such as embeddings.

$$\begin{aligned} g_t &= \nabla_\theta f_t(\theta_{t-1}) \\ m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \max(\beta_2 v_{t-1}, |g_t|) \\ \theta_t &= \theta_{t-1} - \frac{\alpha}{1-\beta_1^t} \cdot \frac{m_t}{v_t} \end{aligned}$$

To add on to this, we would also switch to standard Stochastic Gradient Descent (SGD) during later epochs because SGD provides more predictable and steady changes to parameters. For example we saved the model parameters after 7 epochs and switched to SGD with a small learning rate of 0.0001.

With these changes, we reran the Match-LSTM with Answer-Pointer model and achieved a highest F1 score of 38.1% and EM score of 27.4%. Even with these optimizations, our Match-LSTM model still proved to perform poorly.

At this point, our team decided to scrap the Match-LSTM attention layer entirely and replace it with a co-attention layer from Xiong et al's paper. This attention layer is much more direct in its combination of paragraph and question encodings because we multiply the two matrices together and then concatenate. We decided to keep the answer-pointer prediction layer because we thought that the one described in Xiong et al's paper, a Highway Maxout Network, was unnecessarily complicated and took too long to train.

The figure above shows a visualization of the matrix A^q in our attention layer. In this heatmap, the darker the shade, the more correlation there is between the word in the question and word in the paragraph. For example, the darkest square in the figure corresponds to the word 'universities' in the paragraph and 'institutions' in the question which makes sense considering the answer to this question is 'national universities'.

The new model, using co-attention and answer-pointer, performed the best so far. It achieved a F1 score of 57.0% and EM score of 43.9%. Another observation was that this model began to over-fit near the end. After the ninth epoch, the F1 score on the validation set actually dropped down to 54.8% and the EM score decreased to 41.5%. To fix this, we tried using L2-regularization with a regularization weight of 0.001.

We ran `qa_answer.py` on the dev set with this model to actually see what our predictions were. We noticed that a lot of our incorrect predictions (in terms of exact match) were very close to the ground truth, but had a ⟨UNK⟩ token in it, referring to a word that does not have an embedding in our GloVe file. The starter code stripped away all the GloVe embeddings that are not in the training

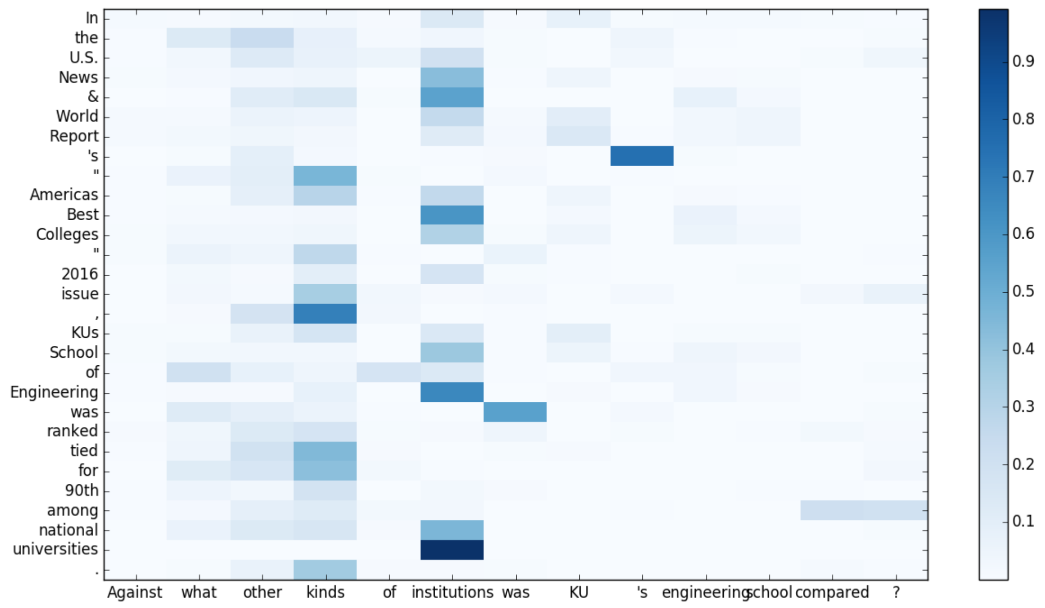

Figure 2: Heatmap of co-attention matrix for an example question

and validation sets, which means that it throws away words that might appear in the dev and test sets. To solve this, we had to edit our starter code files like `qa_data.py` to include in our vocabulary the union of the GloVe vocabulary, and the train and validation vocabularies. For words not found in the GloVe embeddings, we randomly initialized the word vectors. Lastly, we swapped the indices of our predicted start and end answer indices if the end index was greater than the start index, bumping our performance by a final 1% to over 60% F1.

	EM			F1		
	Val	Dev	Test	Val	Dev	Test
Random Guess		1.1	1.3		4.1	4.3
Logistic Regression		40	40.4		51.0	51.0
Match-LSTM w/ Answer Pointer	27.41			38.07		
DCN w/ Answer Pointer	42.23	39.04		56.08	52.90	
DCN w/ Answer Pointer w/ larger vocab	47.73	46.85	47.131	61.06	60.029	60.096

Table 1: Results using hidden size of 150 and embedding size of 100

4.1 Analysis of Predictions

For the remainder of this section, we only refer to exact matches as our metric of correctness. We begin our analysis of predictions by breaking down the questions into one of six categories: ‘who’, ‘what’, ‘which’, ‘why’, ‘when’ and ‘where’. Shown below is a table that breaks down our correct and incorrect predictions into these categories:

From these results, it is pretty clear that our model is best at predicting ‘when’ questions. Even for the incorrect predictions, our model’s guess is very close to the right answer span. For example, one of the questions is ‘When did Luther broaden his attacks to include core Church doctrines?’. The correct answer is ‘1521’ while our prediction is ‘summer of 1521’, which is arguably even more precise than the ground truth answer.

Our model struggles with ‘why’ questions, only achieving 4.4% accuracy on the dev set. One

	Correct	Incorrect	% Accurate
who	449	1641	21.5%
what	2067	8108	20.3%
which	150	554	21.3%
why	17	373	4.4%
when	383	670	36.4%
where	146	849	14.7%

Table 2: Break down of correct and incorrect predictions

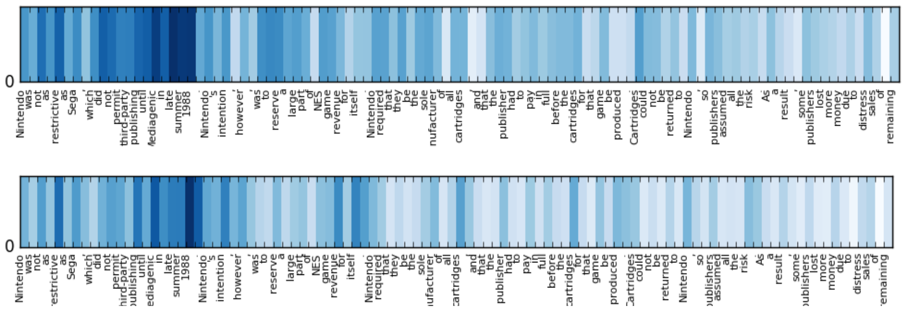

Figure 3: Visualization of prediction probabilities of start (top) and end (bottom) indices. This example predicts correctly. The question is a ”when” question

possible reason is that the model confuses the ‘why’ question with a ‘when’ question. For example the model predicts the answer ‘May 1888’ to the question ‘Why was AC electricity gaining popularity?’. Another reason is that our model fails to understand what the ending index to the answer should be for a ‘why’ question since these types of questions are more open-ended. For instance, the question: ‘Why has the Muslim Brotherhood facilitated inexpensive mass marriage ceremonies?’ has the ground-truth answer ’avoid prohibitively costly dowry demands’. However our model predicted the span ‘to avoid prohibitively costly dowry demands , legal assistance , sports facilities , and women ’s groups’. In the future, we can perhaps train our model separately on ‘why’ questions so it does not confuse it with a ‘when’ question, which it is much more successful at predicting. Also we should impose limits on how long the answer span should be so the model does not produce rambling answers; it is usually the case that the ground truth answers to ‘why’ questions are fairly short. Another alternative is to have a cost function associated with the length of the answer because when our model produces answers with 15+ words, it is almost always incorrect.

Aside from ‘why’ questions, the model is also mediocre at predicting ‘where’ questions. It seems to confuse these with ‘when’ questions as well since both types of questions ask about context; the model just cannot distinguish whether it is asking for temporal or geographical context. For example it answers the question ’Where did scientists find their Y. pestis sample?’ with ‘1998’.

5 Conclusion

Over the course of this assignment, we learned about what it is like to be a natural language processing researcher. To be completely frank, it was way more difficult than we had anticipated. We ran into technology issues that we never expected such as memory exhaustion and power outages. This forced our team to overhaul our code and rewrite it in a more memory efficient way, which was a great learning experience. For example, we created our custom RNN cell which had a LSTM cell embedded in it, and fed it into the library dynamic RNN function. Before this change, we unrolled the time-steps ourselves in a for loop which was grossly inefficient in terms of both computation and GPU memory usage. This assignment also required the longest training times our team has ever encountered which made us think critically about planning the appropriate experiments and running them every night.

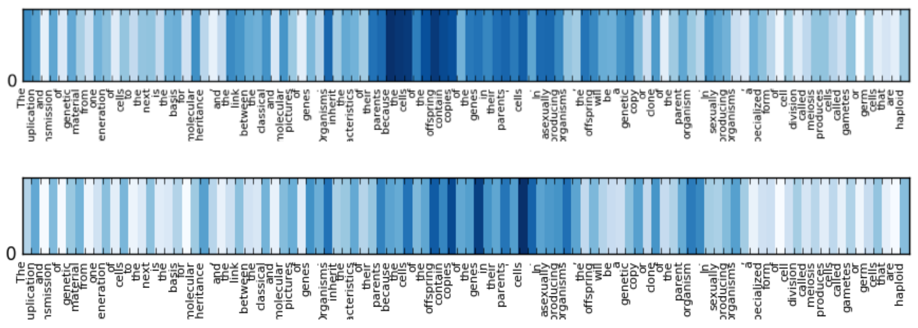

Figure 4: Visualization of prediction probabilities of start (top) and end (bottom) indices. This example predicts incorrectly. The question is a ”why” question

The difficulty of this assignment also taught us more advanced neural network architectures. One benefit of attempting to create a state of the art system was understanding how to read published papers and then implement them. This challenge pushed us to design a complex model, something that might not have happened if we pursued our own project. On a similar note, we also learned more about constructing a model from scratch whereas in class we only had to fill in specific functions. In particular, we learned about the intricacies of TensorFlow and we even stepped through TensorFlow source code in order to really understand our bugs.

For future work, we can combine model architectures from multiple papers. For example, Xiong et al proposed a Highway Maxout Network for forming the predictions of a certain answer index. We can combine our current architecture with using a Highway Maxout Network for the prediction steps.

6 References

[1] Wang, S., & Jiang, J. (2016). Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905*.

[2] Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global Vectors for Word Representation. In *EMNLP (Vol. 14, pp. 1532-1543)*.

[3] Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[4] Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. *ICML (3), 28, 1310-1318* .

[5] Xiong, C., Zhong, V., & Socher, R. (2016). Dynamic Coattention Networks For Question Answering. *arXiv preprint arXiv:1611.01604*.

[6] Danqi Chen, Jason Bolton, & Christopher D Manning. A thorough examination of the cnn/daily mail reading comprehension task. *arXiv preprint arXiv:1606.02858, 2016*.