Bidirectional attention flow for Question Answering

Ana-Maria Istrate
Department of Computer Science
Stanford University
Stanford, CA 94305
aistrate@stanford.edu

Abstract

This paper tackles the task of Question Answering in Natural Language Processing by using a recurrent neural network with bidirectional attention flow and smart span selection over the probability distributions of start and end words at test time. The paper focuses on the SQuAD dataset, Stanford's Question Answering system Dataset, on which best final scores of F1 = 71.88 and EM = 60.5, were obtained.

1 Introduction

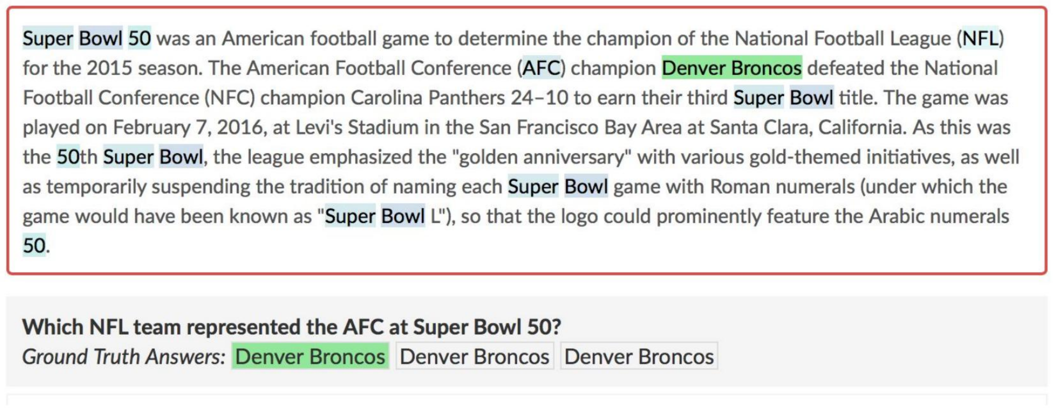
Question Answering is a task in Natural Language Processing concerned with building systems that are able to read and understand information from a piece of text, and then answer questions based on it as accurately as possible. The task has many applications, from summarizing class notes, newspaper articles or academic papers, to being able to provide useful information extracted from a conversation or generating abstracts. The wide variety of intended audiences that could benefit from advances in the area of question answering makes this an active field of research in natural language processing. While answering questions based on a paragraph could involve generating new text, this paper focuses on extracting the span of text in the provided paragraph that answers the question as accurately as possible. Hence, the system is trying to "highlight" the part of the paragraph that best matches the answer to a given question. We use the SQuAD dataset[1], which contains 100K questions on paragraphs taken from Wikipedia. The dataset is a popular one, and teams from different research groups, either in academia or industry are actively creating models to tackle the task[2]. We propose a model that uses a recurrent neural network and bidirectional attention flow with smart span selection over the probability distributions of start and end words at test time. The proposed model achieves a score of F1 = 71.88 and EM = 60.50,on the dev set.

2 Related Work

There are a number of approaches that are known to address the task of question answering fairly well. One such category are the models that rely on having a RNN encoder layer combined with different types of attention between the context and the question. An RNN-based model makes sense in the context of wanting to keep track of the shared information between the context and the question being asked, while attention is useful for combining the representations for the context and the question. The provided baseline for the project used a basic dot-product attention, but more complex types have proved to achieve good results. One such type is co-attention[4], which is a high-performing SQuAD model that involves a two-way attention between the context and the question and attention attending once more over attention outputs. Another high-performing model involves a Context-to-Question attention and a self-attention layer[6]. More attention techniques, such as Fine-Grained Gating[5] or Multi-Perspective Matching[7] are also possible. Other approaches involve using character-level CNNs[3] to learn character embeddings, systems that condition the end prediction on start, or span distributions that concern themselves with learning the joint probability distributions over contexts and questions. At the core of the model proposed in this paper there is an RNN encoder layer using bidirectional attention flow, architecture which has been proven to give good results[3] for the task at hand. Complete details about the architecture of the model are provided in Section 4.2.

3 Problem Statement

In the question-answering task, we are concerned with answering a question about a paragraph as accurately as possible, One particularity of the type of data used in this paper is that the answers to the questions are taken straight from the paragraph. Rather than generating new answers by itself, the model is concerned with finding the span of words in the paragraph that answers the given question the best. For instance, one entry in the dataset and its question and answer could be:



Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL) for the 2015 season. The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion Carolina Panthers 24–10 to earn their third Super Bowl title. The game was played on February 7, 2016, at Levi's Stadium in the San Francisco Bay Area at Santa Clara, California. As this was the 50th Super Bowl, the league emphasized the "golden anniversary" with various gold-themed initiatives, as well as temporarily suspending the tradition of naming each Super Bowl game with Roman numerals (under which the game would have been known as "Super Bowl L"), so that the logo could prominently feature the Arabic numerals 50.

Which NFL team represented the AFC at Super Bowl 50?
Ground Truth Answers: Denver Broncos Denver Broncos Denver Broncos

Figure 1: Example of a {context, question, answer} in the dataset. Picture taken from the SQuAD official website[2]

The model is essentially trying to “highlight” a portion of the paragraph that gives the best answer.

The goal of the model then becomes generating two indices, i_{start} and i_{end} corresponding to the starting and ending indices of the words in the paragraph that give the span of text that best answers the question.

4 Approach

The section below describes the model architecture of the model, built on the provided baseline and the model suggested in the BiDAF paper[3].

4.2 Model architecture

Question answering tasks often involve some attention mechanism that captures which part of the paragraph (context) the question should be focusing on when generating the answer, one word at a time. The model proposed in this paper uses bidirectional attention flow, which is built on the insight that attention should go both ways: both from the question to the context and from the context to the question. The architecture of the final model is inspired by the approach described in the baseline provided and the BiDAF paper[3] and is the following:

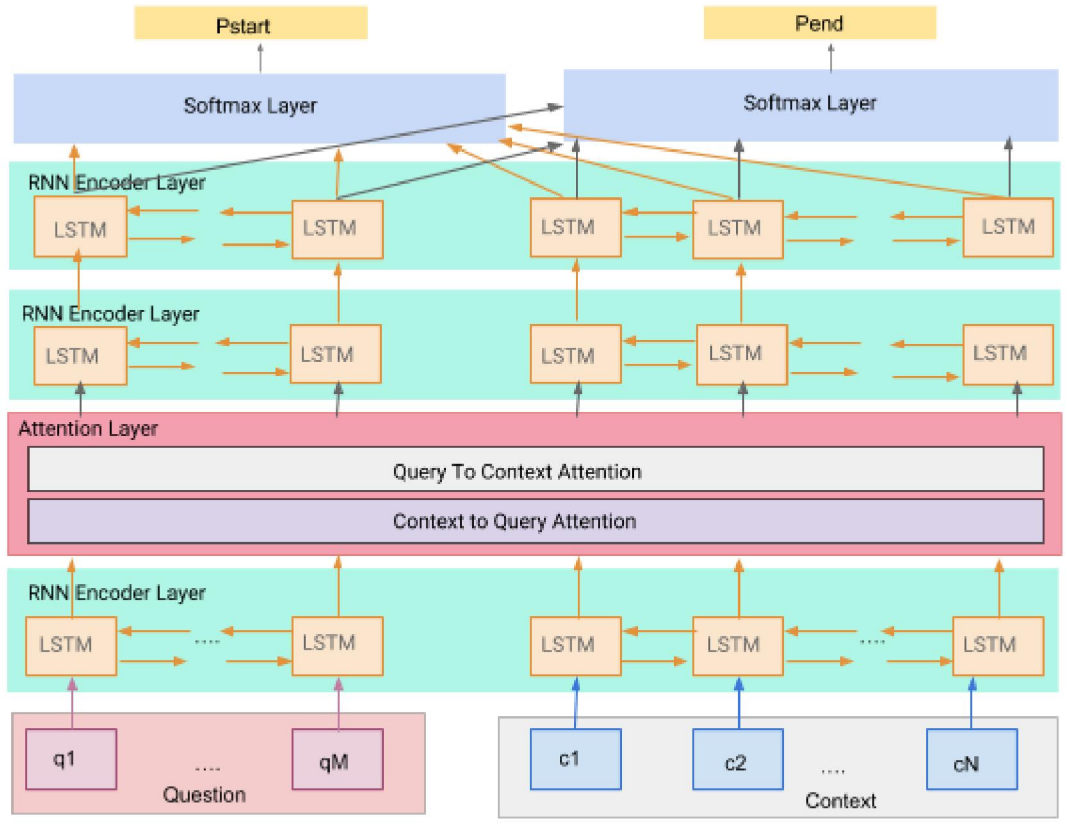


Figure 2: Model architecture

Input: Each context is represented as a sequence of d-dimensional fixed, pretrained GloVe word embeddings $x_1, x_2, \ldots, x_N \in R^d$, where N = context length

Similarly, each question is also represented as a sequence of d-dimensional fixed GloVe word embeddings $y_1, y_2, \ldots, y_M \in R^d$ where M = question length

RNN Encoder Layer: Each of the embeddings are fed into a bidirectional LSTM layer, shared between the context and the question. For both the context and the question, the RNN encoder layer outputs a sequence of forward and backward hidden states, where each $c_{i-forward}, c_{i-backward}, q_{j-forward}, q_{j-backward} \in R^h$, for a predefined size of the hidden states h.

$$[c_{1-forward}, c_{1-backward}, \ldots, c_{N-forward}, c_{N-backward}] = \text{biLSTM}([x_1, x_2, \ldots, x_N])$$

$$[q_{1-forward}, q_{1-backward}, \ldots, q_{M-forward}, q_{M-backward}] = \text{biLSTM}([y_1, y_2, \ldots, y_M])$$

The choice of using an LSTM as opposed to a GRU for the RNN cell was made because it gave better results on the dev set.

The forward and backward states are then concatenated together with their sum, to obtain the context hidden states c_i and the question hidden states q_j:

$$c_i = [c_{i-forward}, c_{i-backward}, c_{i-forward} + c_{i-backward}] \in R^{3h}$$

$$q_j = [q_{j-forward}, q_{j-backward}, q_{j-forward} + q_{j-backward}] \in R^{3h}$$

I experimented with different ways of obtaining the new hidden states c_i and q_j from the backward and forward hidden states, including adding and averaging them, but the concatenation above gave the best results.

Bidirectional Attention Layer: Firstly, we compute the similarity matrix $S \in R^{NxM}$ between the hidden context and question states. Each entry S_{ij} in the matrix corresponds to how similar context c_i and question q_j are and is defined as: $S_{ij} = w^T_{sim}[c_i; q_j; c_i \circ q_j] \in R$, where w_{sim} is a weight vector learned during training. Then, we focus on the two types of attention: Context-to-Question Attention and Question-to-Context Attention

1. **Context-to-Question Attention:** The first step is to compute the Context-to-Question Attention output. In order to do that, we take the row-wise softmax of S, which is then used to weigh the question hidden states q_j to get the Context-to-Question Attention outputs a_i:

   $\alpha^i = softmax(S_{i:}) \in R^M$ for all i = 1, ... N

   $a_i = \sum_{j=1}^{M} \alpha^i q_j \in R^{3h}$ for all i = 1, ... N

   When computing the softmax over the rows of S, a mask is being used, to account for the fact that some context and question entries are being padded with 0s to fit in the maximum context length and question length, respectively. The mask for S is being computed from the given context and question masks.

2. **Question-to-Context Attention:** Next, we compute the Question-to-Context Attention. For each context c_i, we take the maximum of the corresponding row in S, which we then use to get a softmax distribution β over all the context locations. The result is used to get the Question-to-Context Attention output c':

   $m_i = max(S_{ij}) \in R$ for all i = 1, ... N

   $\beta = softmax(m) \in R^N$

   $c' = \sum_{i=1}^{M} \beta_i c_i \in R^{3h}$

The final outputs b_i of the bidirectional attention layer are then computed by the following concatenation:

$$b_i = [c_i; a_i; c_i \circ a_i; c_i \circ c'] \in R^{12h} \text{ for all i = 1, ... N}$$

RNN Encoder Layer: The outputs b_i from the Bidirectional Attention Layer are fed into an RNN Encoding Layer that uses a bidirectional LSTM cell to generate forward and backward states for each b_i:

$$[b_{1-forward}, b_{1-backward}, ..., b_{N-forward}, b_{N-backward}] = \text{biLSTM}([b_1, b_2, ..., b_N])$$

Second RNN Encoder Layer: The outputs from the previous layer are fed into another RNN encoding layer to get the final outputs d_i :

$$[d_1, d_2, ..., d_{4N}] = \text{biLSTM}[b_{1-forward}, b_{1-backward}, ..., b_{N-forward}, b_{N-backward}]$$

Output Layer: Finally, each of the previous outputs d_i are being fed into a fully connected layer with N number of outputs and a ReLU activation function to get N outputs d'_i:

$$d'_i = ReLU(Wd_i + q) \text{ for each i = 1, ..., N}$$

Next, we use use the outputs d'_i to get scores s_i^{start}, s_i^{end} for each context location being either a start or an end word for the answer:

$$s_i^{start} = W_{start}^T d_i' + b_{start} \text{ for each i = 1, …, N}$$

$$s_i^{end} = W_{end}^T d_i' + b_{end} \text{ for each i = 1, …, N}$$

Finally, we take a softmax over each of s_i^{start}, s_i^{end} to get the probabilities that the answer starts or ends with the context word c_i.

$$p_i^{start} = softmax\,(s_i^{start}) \quad \text{for each i = 1, …, N}$$

$$p_i^{end} = softmax\,(s_i^{end}) \quad \text{for each i = 1, …, N}$$

Loss: The loss function L is the sum of the cross-entropy loss for both the start and end locations, averaged across the entire batch of size B:

$$L = \Big(\sum_{j=1}^{k} - log\, p^{start}(i_{start,j}) - log\, p^{end}(i_{end,j}) \Big)/k$$

where k is the total number of items in a batch, and $i_{start,j}$ and $i_{end,j}$ are the true start and end locations for the answer on a single item j. During training, we minimize this loss function using the Adadelta optimizer.

Prediction: During test time, for a given context and question pair, we predict two indices i, j, corresponding to the start and end indices respectively, that satisfy $1 \le i \le CL - 5$, $i \le j \le min(i + 10,\ CL)$ and maximize $p_i^{start} p_j^{end}$ where CL is equal to the length of the given context. In contrast with the baseline model, which was taking i = argmax(p_i^{start}), j = argmax(p_j^{end}), predicting the two indices independently from each other, this model is looking to maximize the joint probability of both the start and end indices, and takes into account the fact that the end index has to be after the start index (in the baseline model, there is nothing that prevents the predicted end index to be before the predicted start index).

Start index: The choice of having the start index i go only up until $CL - k$ for some positive value k came after realizing that the answer to a question is not very likely to start towards the end of the paragraph, so there should be some range of words at the end of the paragraph that should not be considered as start indices. The same approach was tried by setting a lower bound $p \le i$ from which the start index could start at, for $p > 1$. The search gave worse results on the dev set, suggesting that it should not be assumed that the answer cannot start at the first indices in the paragraph. Intuitively, the insight makes sense, as by setting the upper bound for the start index to $CL - 5$ we are only restricting answers not to start less than five words before the end of the paragraph, but we are not restricting that the answer itself cannot be contained towards the end of the sentence. However, if we restrict the start index not to start at the first indices in the paragraph, we are completely disregarding information from a part of the paragraph. This is not necessarily a good assumption, as there is nothing that prohibits the answer of a question to start at the very first indices of the paragraph.

End index: The first restriction on the end index is that it should be as least as large as the start index. This is because we can’t have an answer that ends at index smaller than the start index. Secondly, the fact that the end index j should go only up to $min(i + k',\ CL)$ for some positive integer k’ was motivated by the insight that answers that come from a paragraph are not very likely to be of very large length, so their length must have some bound k’. After trying different values for k’, k’ = 10 gave the best results on the dev set. We are taking the minimum between i + 10 and CL to account for the fact that some answers that start towards the end of the paragraph can only span until CL, not 10. Below are the F1/EM scores for different models for i, j, which motivated the choice above.

	F1 score	**EM score**
$1 \le i \le CL - 5$, $i \le j \le min(i + 10,\ CL)$, $\text{argmax}(p_i^{start} p_j^{end})$	71.8820	60.5014
$1 \le i \le CL - 5$, $i \le j \le min(i + 15,\ CL)$, $\text{argmax}(p_i^{start} p_j^{end})$	71.3774	60.0378
$1 \le i \le CL$, $i \le j \le min(i + 10,\ CL)$, $\text{argmax}(p_i^{start} p_j^{end})$	71.0703	59.9526
$1 \le i \le CL$, $i \le j \le min(i + 15,\ CL)$, $\text{argmax}(p_i^{start} p_j^{end})$	71.3406	59.9526
$1 \le i \le CL$, $i \le j \le CL$, $\text{argmax}(p_i^{start} p_j^{end})$	71.1302	59.3945
i = $\text{argmax}(p_i^{start})$, j = $\text{argmax}(p_j^{end})$ (baseline model)	69.5170	58.4295

Table 1: Comparison of different models for choosing the start and end indices at test time

5 Experiments

Below there is a description of the experiments I performed during training.

5.1 Dataset

The dataset used the SQuAD[1] dataset, Stanford's Question Answering Dataset, proposed and maintained by Stanford's Natural Language Processing group in the Computer Science Department. The dataset contains over 100K paragraphs from Wikipedia, along with questions and answers crowdsourced using Amazon Mechanical Turk. Each entry in the dataset contains a context (a paragraph from Wikipedia), a question, and an answer as a span from the paragraph, all tokenized and lowercased.

4.2 Training

Below are the experiments I tried during training. For each experiment, I assessed its performance using the F1 and EM scores on the dev set from the Tensorboard graphs.

4.2.1 Training Specifications that were kept in the final model

The following training specifications were used on the best model:

Optimizer: During training, I experimented with different optimizers (RMSProp, Adam, Adadelta, Adagrad, Momentum), and learning rates. I achieved the best results with the Adadelta optimizer with a learning rate of 0.5.

Training time: I trained my best model for 95.5K iterations, which took a little over three days.

Regularization: In order to avoid overfitting, I used dropout on all the LSTM cells. I experimented with various values for the dropout, and the best results were obtained with dropout 0.15.

Batch size: I used a batch size of 50. The choice of 50 was made such that the model would fit into memory.

Context length: I used a context length of 400. The initial context length was of size 600 and encompassed more context information. However, due to the dimensions of the model, I had to reduce both the batch size and the context length in order to be able to fit the model in memory. I experimented with reducing the batch size even more, but context length of 600 still proved worse performing than context length 400 in terms of training time: ~9s/batch for context length of 600, in comparison to ~2s/batch for context length of 400. Because of the significantly faster training, I kept context length of 400. I also tried decreasing context length even more, but I observed significantly worse performing results than context length 400, so I decided to keep 400.

Types of RNN cells used: The baseline code was using GRU cells in the RNN encoder layer. I experimented with LSTM cells, which provided better results, so were kept in the final model.

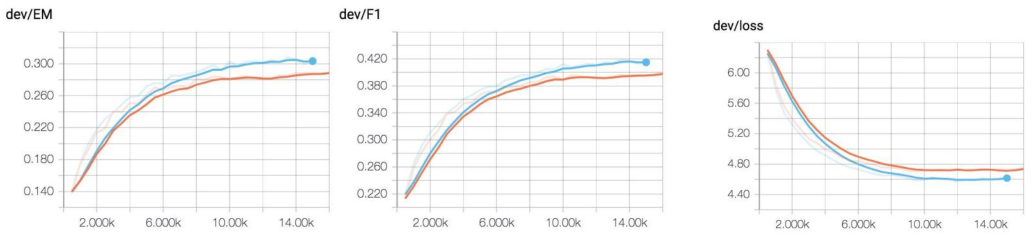


Figure 3: Tensorboard graphs showing the EM, F1 and loss of the baseline model on the dev set of using GRU (orange) versus LSTM (blue) cells

4.2.2 Other training specifications tried that were not kept in the final model

There have been a number of other things I tried during training that did not bring major improvement, so were not kept in the final model:

Changing the embedding size of the GloVe vectors: I experimented with using GloVe vectors of size 200 and 300, but I did not see any improvements. Hence, I kept the size of the GloVe pre-trained vectors to 100, for memory purposes.

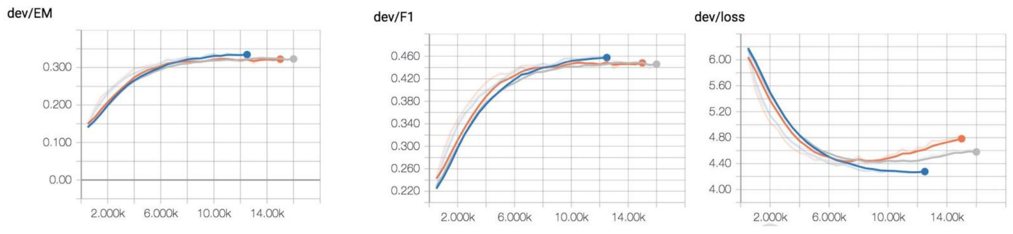


Figure 4: Tensorboard graphs showing the EM, F1 and loss of the model in initial stages on the dev set by using GloVe vectors of size 100 (blue), 200 (gray) and 300(orange)

Training the word vectors: I experimented with training the word vectors embeddings, but did not see any improvement. While the F1 and EM values on the training set increased significantly, the values on the dev set decreased, suggesting that the system was overfitting. Because training the word vectors was also computationally expensive, besides giving worse performance on the dev set, it has not been used in the final model.

4.2 Metrics

Performance was assessed by the F1 and the EM scores, as well as the value of the loss on the dev set. The F1 score is a harmonic mean of precision and recall, while EM score is a binary metric assessing whether or not the predicted output matches the true output exactly.

5 Results

This section presents the results I obtained with my model.

4.2 Quantitative

Below, the performance of the model can be assessed in contrast with the performance of the baseline model. Both the F1 and EM on the dev score increased, where

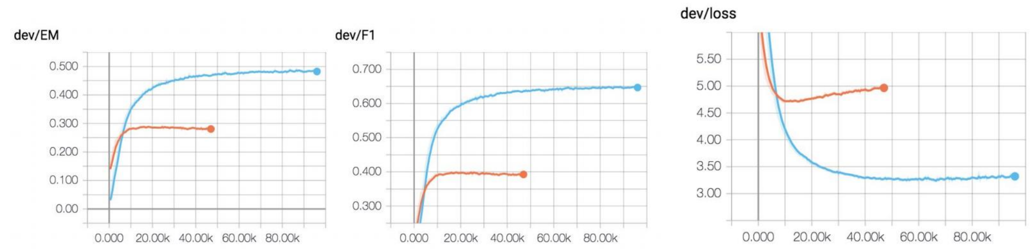


Figure 5: Tensorboard graphs showing the EM, F1 and loss of the final model (blue) compared to the baseline model (orange) on the dev set

The best model achieved a score of F1 = 71.8820 and EM = 60.5014, above the provided baseline, which was only able to achieve scores of F1 = 0.39 and EM = 0.28.

4.2 Qualitative analysis

Looking at some of the {context, question} pairs that the model did not perform well on, some categories of mistakes can be drawn:

1. **Time periods**

The majority of the mistakes of the model were made when the answer of the question was a year or a time period, such as “2010”, or “2015”, or “1910-1940”. This suggests that while the model has good understanding of the overall context of the question, it still has trouble capturing the temporal relation between elements in the context.

2. **Words towards the end of a sentence**

Another pattern that came across was that the model seemed to be biased towards predicting words that were towards the end of the sentence (either the last or close). Often, the model would predict just one word towards the end of the sentence, when the answer would be somewhere in the middle of the sentence. This suggests that the model is giving higher weight to the words in the end of the sentence and still has trouble remembering information from previous parts of the sentence that could be more relevant.

3. **Names of people**

One other category of mistakes that the model made were those involving names of people, especially when more than one person would be present in the context, or when the name of the person involved two different words (first and last name, for instance). This suggests that the model has trouble distinguishing between different people in a context or realizing that a person’s name should involve more than one word.

4. One-word answers

Most of the mistakes being made were answers that predicted just one word, when the true answer would have been longer. This suggests that the model is being biased towards putting too much weight on one word in a sentence, in contrast to distributing the probabilities across the sentence. It suggests that more complexity could be added to the model or that better mechanisms for using the probabilities of the start and end indices at test time should be employed.

5. Money

Some of the mistakes that the model made involved money: “$1.2 billion allston science complex”, “$4.093 million available for disbursement”, suggesting that the model doesn’t deal well with comprehending financial information and putting it into context to understand its meaning (for instance, it should be able to know to stop after the “billion” tag).

6. Numbers

It seemed that the model made a lot of mistakes that involved numbers. Either time periods, percentages or money, the model seems to have a hard time grasping numbers. This could suggest that the model has more difficulty grasping numbers rather than characters in a word. It would be interesting to study further why that might be the case.

7 Conclusion and future work

The bi-directional attention flow model combined with the smart selection span over the start and end indices provided good results for the question answering task and significantly improved the performance of the provided baseline. Both the training and the dev set had similar final values for F1 and EM scores, suggesting that the model was not overfitting and that was applying everything it was learning from the training set to the dev set. Even on the training set, however, the model did not seem to be able to hit more than ~80% F1 and ~65% EM, which could suggest that the model was still missing some complexity in being able to overfit on the training set. Thus, future work would include adding more layers to the network and increasing its complexity, in order to be able to get larger F1 and EM values. Some ideas include adding a self-attention or co-attention layer, or another RNN Encoder layer. Other possible additions would be adding more features (Part of Speech, Named Entity or whether the context token appears in the context), training the character level CNNs and ensembling different methods. Another suggestion would be to find ways to improve the span selection at test time. This could be achieved, for instance, by using the distribution of answer lengths or positions in the paragraph of the start and end answer indices on the training set to generate a better system for predicting start and end indices on the test set.

References

[1] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. CoRR , abs/1606.05250, 2016.

[2] SQuAD Official Website: https://rajpurkar.github.io/SQuAD-explorer/

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

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

[5] Zhilin Yang, Bhuwan Dhingra, Ye Yuan, Junjie Hu, William W Cohen, and Ruslan Salakhutdinov. Words or characters? fine-grained gating for reading comprehension. arXiv preprint arXiv:1611.01724 , 2016.

[6] Bhuwan Dhingra, Hanxiao Liu, Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Gated-attention readers for text comprehension. arXiv preprint arXiv:1606.01549 , 2016.

[7] Zhiguo Wang, Wael Hamza, and Radu Florian. Bilateral multi-perspective matching for natural language sentences. arXiv preprint arXiv:1702.03814 , 2017.