SQuAD Question Answering using Multi-Perspective Matching

Zachary Maurer
zmaurer@stanford.edu

Shloka Desai
shloka@stanford.edu

Sheema Usmani
sheema@stanford.edu

Codalab Submission ID: zmaurer

Abstract

In this paper we explore several approaches to implementing a Question-Answering model using the SQuAD [1] dataset. We explore four neural network architectures for this task: a model that attends over the question representation, a simple model without attention, a model that uses a normalized affinity matrix between paragraph and question representations as an attention layer and a final model that uses a Multi-Perspective Matching layer with filtering based on the IBM Watson Labs paper[2]. We find that the Multi-Perspective model performs the best, with 68% F1 score and 54% EM score on the training set, and 55.06% F1 score and 43.13% EM score on the Codalab test dataset.

1 Introduction

Machine Comprehension (MC) and Question Answering (QA) have long been central problems in the field of Natural Language Processing (NLP). Recent advances in Deep Learning and associated hardware technologies, like GPUs and cloud computing, have facilitated the development of new networked architectures that have made large gains in performance on more generalized QA tasks. New crowdsourcing platforms have also allowed researchers to develop larger datasets that are sufficient training inputs for these more complicated and data-intensive models.

In this project, we surveyed existing literature and implemented neural network architectures that competed on the most recent of these QA datasets: the Stanford Question Answer Database (SQuAD). We developed a series of baselines and created a final model which was based on IBM Watson Labs' Multi-Context Perspective Matching [2] paper. We used TensorFlow [4] for all of our models, with some early experimentation in Keras. Our models implement various forms of attention and relied heavily on the use of Bidirectional LSTMs.

On the train and validation data sets respectively, our final model achieved an F1 score of 68% and 54%, and an EM score of 53% and 39%.

On the Codalab test dataset, our model achieved an F1 score of 55.06% and an EM score of 43.13%.

2 Background and related work

The SQuAD dataset comprises of around 100,000+ question-context-answer tuples. The context paragraphs were extracted from a set of articles from Wikipedia and the questions are generated

using those paragraphs as context by people.[6] This dataset is orders of magnitude larger than all previous hand-annotated datasets and has the desirable quality that answers are multi-word spans in a reference document. This is lacking in the Cloze-style QA datasets where answers were a single word in the paragraph. The presence of multi-word answers is a defining feature of the SQuAD dataset that will be crucial to successful model design.

Multiple other groups have developed high performing models using SQuAD. A description of a subset of these models follows below.

Xiong et al.[3] proposed a dynamic coattention network for QA task. This method uses attention vectors to focus on a small portion of the context and summarizes it with a fixed-size vector. This can be done by taking the weighted average of all the hidden states rather than using only the final hidden state.

Multi-Perspective Context Matching[2] is another approach which has an initial filtering of word embeddings using cosine similarity. Their approach identifies the answer span by matching each time-step of the passage with the question from multiple perspectives using a full matching layer with multiple perspectives. It predicts the beginning and ending points based on globally normalizing probability distributions.

Inspired by the above approaches we tried building an end to end neural network for question answering using simple forms of attention, filtering and Multi-perspective Context Matching.

3 Distribution of Question-Answer Data

To gain more insight into the inputs for our model, we did some exploratory data analysis of the question dataset to get a sense of the sequence lengths that our model would have to take as input.

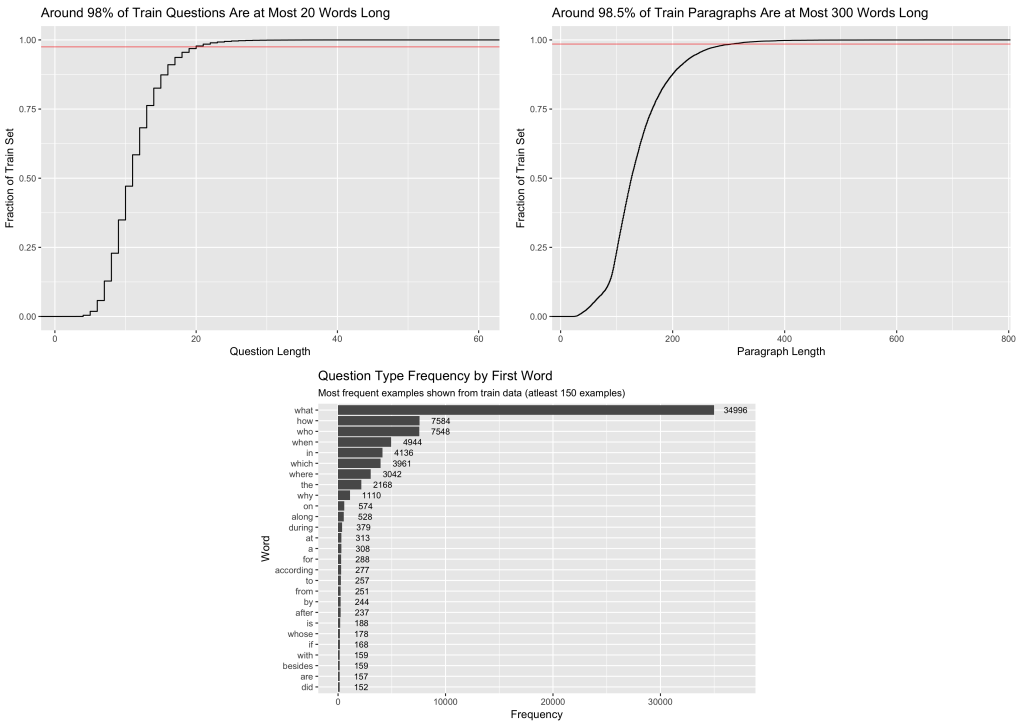


Figure 1: Distribution of question length, paragraph length, and question type in the SQuAD dataset

Based on plotting the cumulative distributions of question and paragraph length in the train data, we observed that roughly 98% of questions are 20 words or less, and 98.5% of paragraphs are words

300 words. We used these boundaries to modify the data to standard lengths through zero-padding or truncation.

Also, we examined the relationship between paragraph length and answer span, and question length and answer span, and found nothing that would suggest a relationship between the different variables. Question length appeared to be unrelated to paragraph and question lengths.

4 Approach

We implemented four different models, as described below. All initial question and paragraph representations are passed through an embedding layer. Let P_e and Q_e be the paragraph and question representation after passing through the embedding layer. All of our models predict probability distributions for the start index and end index of the answer directly. This is referred to as a "boundary" prediction model.

4.1 Model 1: Simple model using attention over the question

We first pass P_e through a BiLSTM and concatenate the outputs at each timestep to get P_v, the paragraph representation. We get the question representation Q_v by passing Q_e through a BiLSTM and concatenating the final states. We then blend these by concatenating the final state of the question to each word in the paragraph and then passing the result through another BiLSTM to get C_v, the question paragraph representation. We calculate an attention vector α over the question by taking the dot product $C_v Q_v^T$. Finally we concatenate α to each word of the paragraph, C_v to get our final paragraph representation P_{final}. We pass this through a LSTM to calculate a probability distribution for the start index, and a probability distribution for the end index.

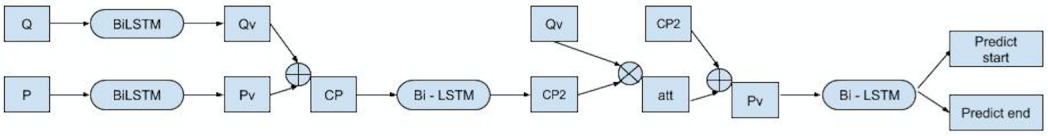


Figure 2: Simple model using attention over the question

4.2 Model 2: Simple model without attention

We pass P_e through a BiLSTM and concatenate the outputs at each timestep to get P_v, the paragraph representation. We get the question representation Q_v by passing Q_e through a BiLSTM and concatenating the final states of the forward and backward pass. We predict probability distribution of the start index by taking the dot product of Q_v with every word in P_v. The end index is computed in a similar manner.

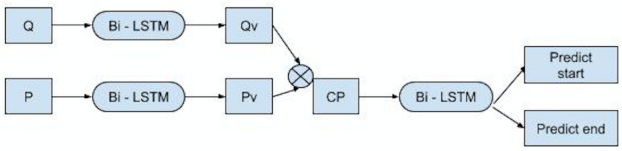


Figure 3: Simple model without attention

4.3 Model 3: Model using affinity matrix between paragraph and question

We pass P_e and Q_e through a BiLSTM and concatenate the outputs at each timestep to get P_v and Q_v the paragraph representation, and the question representation respectively. We then calculate the affinity matrix A by taking the dot product between $P_v Q_v^T$ and softmaxing it. We multiply this with the question Q_v, to calculate the context vector for the question. The result is concatenated to each word in the paragraph. We add a fully connected layer on top of this to calculate the start index. We calculate the probabilities for the end index by passing the start index probabilities through another LSTM and adding a dense layer on top.

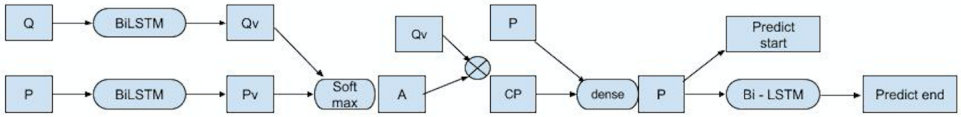


Figure 4: Model using affinity matrix between paragraph and question

4.4 Model 4: Multi perspective context matching

Our Multi perspective context matching model consists of the following layers (in addition to the embedding layer described above):

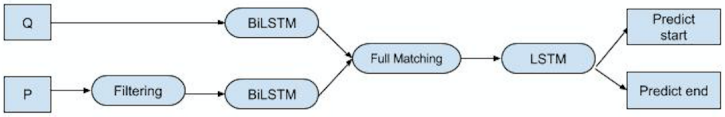


Figure 5: Multi Perspective Context matching layer

Filter layer for the paragraph embeddings: This layer filters out irrelevant information from the paragraph. For each word in the paragraph we calculate a relavancy degree with respect to the question, this is done by taking the max of the cosine distance between the paragraph word and every question word. We multiply each paragraph embedding vector by its relevancy degree to filter out those words that are less relevant.

Context representation layer: In this layer we pass the question embeddings Q_e through a biLSTM to get the question representation, and pass the filtered paragraph embeddings through another BiLSTM to get the paragraph representation.

Multi perspective context matching layer: In this layer we compare each word in the paragraph with the question from multiple perspective. For every word in the paragraph, we calculate an l length vector **m**, where l is the number of perspectives. Let W be a matrix of size $l \times h$, where h is state size of the LSTM in the previous layer. Then define the matching function for two words u and v, $m = f_m(u, v, W)$ where:

$$m_k = cos(u \circ W_k, v \circ W_k)$$

We calculate $m_{forward}$ as $f_m(u, v, W1)$ where u is every forward hidden state of the paragraph representation from the previous layer and v is the the final state of the question representation. Similarly $m_{backward}$ is calculated as $f_m(r, s, W2)$ where r is every backward hidden state of the paragraph representation and s is the final state in the backward LSTM for the question. The output of this layer is the concatenation of $m_{forward}$ and $m_{backward}$.

Prediction layer: This layer takes the output of the full matching layer and passes it through an LSTM. The outputs of the LSTM are passed through two separate fully connected layers (one for

the start, and one for the end) to calculate the probability distribution for the start and end indices respectively.

4.5 Implementation details

For all our models we used the cross entropy loss, with Adam as the optimizer. We used a hidden size and embedding size of 200. All question representations were padded, and masking was applied for all LSTMs. For the Multi perspective context matching layer, we used 50 perspectives. When making predictions, we force the end index to be greater than or equal to the start index. Hyperparameter tuning on smaller train sets of ten to twenty thousand examples indicated that a learning rate of 0.001 with a batch size of 20 produced the best results.

5 Results and Discussion

Initial Efforts

Our three baseline models produced very limited performance, despite utilizing attention. Table 1 shows the F1 and EM scores for the first three models.

	F1 score	EM score
Model 1	9%	6%
Model 2	5%	3%
Model 3	15%	9%

Table 1: Initial F1 scores and EM scores for models 1, 2, and 3 on train

The most important finding from this stage of development was that utilizing a boundary prediction model was crucial for encouraging the model to learn and thus perform better. In earlier iterations, we had a sequential prediction layer which made binary classifications on each paragraph word as part of the answer span or not. This caused the model to almost always predict that a word was not in the answer span. Because most paragraph words aren't in the answer, our model would minimize its loss by essentially never making a real prediction. The boundary characterization of the problem forces the model to output a meaningful prediction at each timestep, and forces it to develop an internal representation that reduces the loss.

We believe that Model 1 failed to perform particularly well due to an insufficient aggregation layer that combined the attention vectors and conditioned representations. We believe this to be the case because Model 1 is very similar to Model 3 (which performed better), except for this missing aggregation layer. We believe that the signal encoded by the attention vector in Model 1 was attenuated immediately by the LSTMs, early in training.

Model 2 failed to produce meaningful results simply because its expressive power was too limited as it did not use attention.

Using BiLSTMs caused modest improvements in performance over single direction LSTMs in earlier models. We believe that BiLSTMs assist in making predictions on longer paragraphs.

Final Model

Our implementation of MPCM performed significantly better than our baseline models. This is directly due to the use of filtering using cosine similarity and the usage of the Full Match perspective layer. Overall this method provided a much more sophisticated way of conditioning the paragraph, question representations and encoding attention. Essentially, the number of perspectives used by our model could be thought of as multiplying the attentive-focus of our baseline by a factor 50, since each timestep in a paragraph now was multiplied an entire trained weights matrix, instead of just a vector. In addition, the conditioning using the question at this step was also much more complex because it additionally processed the question representation through another dense layer.

Fig. 6 below shows the F1 and EM scores, training curves and loss curves for the MPCM model.

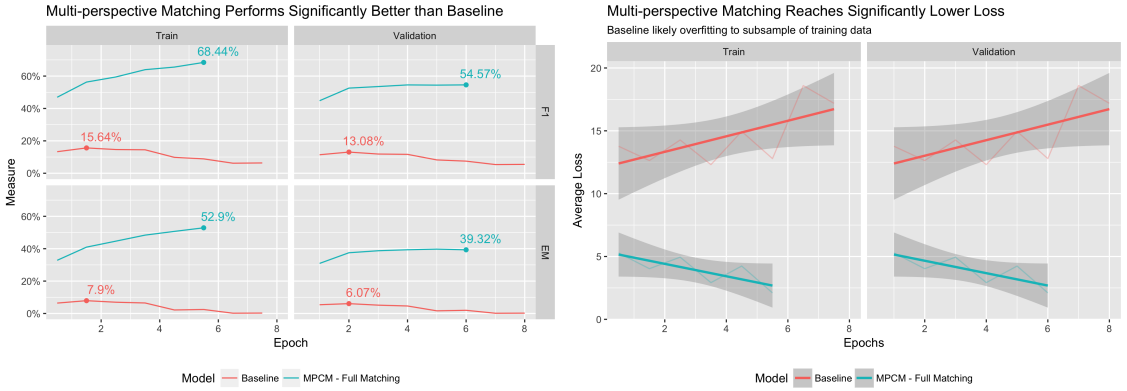


Figure 6: Model performance: F1 score and loss over time

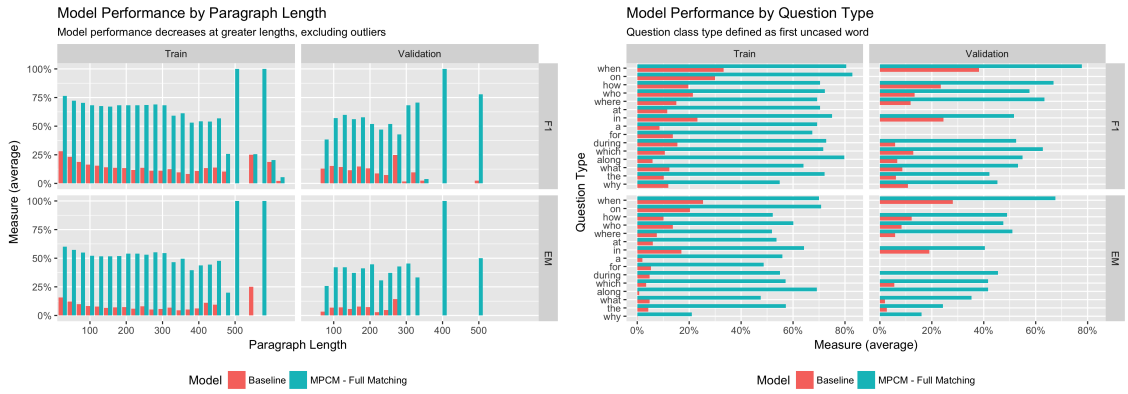
	F1 score	EM score
Train	68%	54%
Val	53%	39%
Test	55.06%	43.13%

Table 2: F1 and EM scores for Model 4 on the train, validation, and test sets

Analysis of Final Model Results

We spent some time analyzing the training data to understand what types of errors our model was making.

We see in Fig. 7i that our models performance tends to drop off slightly on longer paragraphs around 300-450 words long. (The spike around 500-600 is due to the fact that there are only a couple training examples of this length in the data set, and thus getting one example correct would skew the average F1/EM extremely high). This could likely be fixed by adding the additional Max-pooling and Mean-pooling layers specified in the MPCM model paper. We had implemented this code, but unfortunately the entire model could not run due to Out-of-Memory errors on Azure.



In Fig 7ii, we tried to determine what type of questions our model performed well at. We see that questions starting with ”when” performed consistently well across train and validation this is likely due to the fact that questions starting with ”when” have predictable qualities about their answers such as the presence of dates or times. The performance of more conceptual questions, like ”what” and ”why” questions suffered most heavily in the validation evaluation, likely because the features that define an answer are more nuanced.

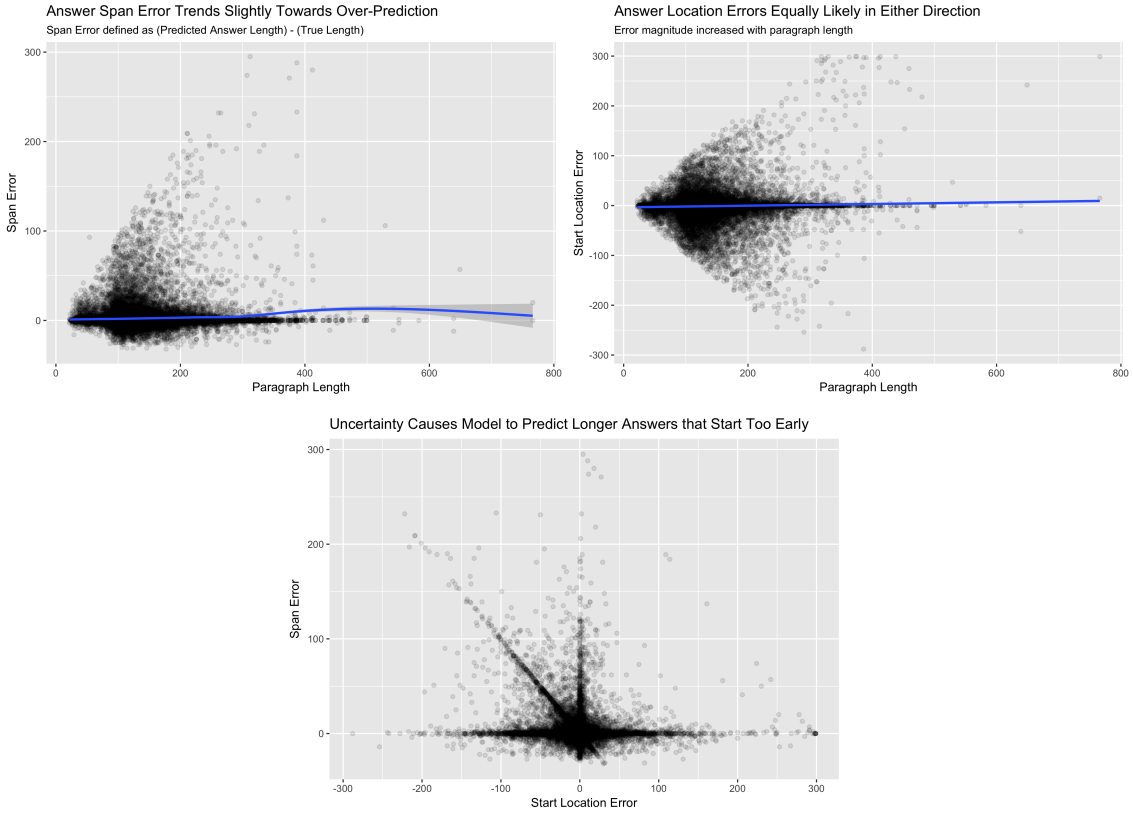


Figure 7: Plots for span error versus paragraph length and start location error

After reviewing the performance across different question classes, we wanted to dig deeper into the types of errors our model made. Specifically we investigated the start index and answer span length and tried to determine how they were related.

We accomplished this by plotting in Fig. 8i and 8ii the relative error (predicted - true) in answer start index (referred to as "location error") and answer length (referred to as "span error") by paragraph length. We found our final MPCM model made uniform errors in either direction around true start index of an answer in a paragraph in the vast majority of cases. This indicates that our model did not have a strong bias towards predicting the start of an answer too early or late. Predictably, our model demonstrated more uncertainty for larger paragraph lengths, as the location error tends to grow in both directions. For span error, we that the model generally made the vast majority of errors close to the true value, on either side, producing slightly too long or slightly too short answers. However, its clear the in uncertain situations, the model tended to classify larger sequences of the paragraph as the answer in attempt to include as much information as possible.

We finally plotted the joint distribution of span error and location error (see Figure 8iii). Here, we find that most errors are symmetrically clustered around the origin, which indicates small errors from the true value. However, we see an interesting trend line that shoots off as location error decreases (namely the model predicts a start index before the true answer start), the model over predicts the answer span in order to compensate. The slope of this line is approximately -1. Thus for every time it predicts n words too early, the model will compensate by picking n extra words to add to the answer span. This is not surprising and just perhaps reflects the need for more attention mechanisms which would hone in on the relevant text. Another possible way of dealing with this behavior in our model would be to penalize longer incorrect answers more severely than shorter incorrect answers. However, this solution introduces more rules into our model and reduces generality to datasets with larger answer spans.

6 Conclusion and future work

Overall, we found that the MPCM model performed the best because of the filter layer and the new ways in which it encoded attention. We saw that the model performed best for "when" questions, and that the performance dropped slightly with paragraph length. Our model did not have a strong bias toward predicting well before the start of the answer, or after the start of the answer, but it tended to predict a longer answer than was necessary in some cases.

In terms of future work to improve on our models, we can use 840B Common Crawl GloVe word vectors rather than the Glove word vectors pretrained on Wikipedia 2014 and Gigaword5. Given additional computational resources, we can train more perspective matching layers for our fourth model, specifically the Mean-pooled and Max-pooled perspective layers. Also, working on an ablative analysis would be useful in gauging the impact of the different perspective layers.

Finally, we could generate an ensemble of models that trained on subsets of question type and bucketed question length. We can implement an answer span penalty as well to counter the fact that the model tends to predict longer answers.

Acknowledgements

We would like to thank the teaching staff for their kind support throughout a very challenging project, the team at Codalab for troubleshooting any leaderboard issues and the staff at Microsoft Azure who generously donated computing resources to this class.

References

1. Bi-Directional Attention Flow for Machine Comprehension. Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, Hananneh Hajishirzi.
2. Multi-Perspective Context Matching for Machine Comprehension. ZhiguoWang, Haitao Mi, Wael Hamza and Radu Florian.
3. Dynamic Coattention Networks for Question Answering. Caiming Xiong , Victor Zhong , Richard Socher.
4. https://www.tensorflow.org/
5. https://nlp.stanford.edu/projects/glove/
6. Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text