Question Answering with SQuAD: Variations on Multi-Perspective Context Matching

Jason Freeman, Raine Hoover
jasonf2@stanford.edu, raine@stanford.edu
CodaLab: JasonF, rainehoover
Stanford University
CS 224N, Winter 2016-17

Abstract

We implement multi-perspective context matching for the task of question-answering on the SQuAD dataset and explore a variety of modifications to this core architecture. In our first modification, we compare the performance of GRUs with that of LSTMs in the original model. Next we attempt to predict the answer's start index and length rather than its start and end indices. Finally, we introduce a variation of attention which we call "question summaries" into the model. This last modification proves most fruitful.

1 Introduction

Question answering is a key language competency that humans acquire early in their development. Beyond the compelling theoretical implications of succeeding in modelling this competency in machines, automating question answering has many practical applications. Machine comprehension has already helped improve the web search experience via direct answer display [2], and taken home the winning prize on Jeopardy [1].

In this work, we build on existing research addressing the traditional question answering task: given a passage and a question referencing that passage, find the segment (or span) of the passage that answers the question. Our dataset for this task is described in section 2. Next we discuss the base model and our extensions to it in section 3. We explain our evaluation metrics in section 4, compare and contrast our efforts with previous works in section 5, and present our results in section 6. Finally, we describe common errors and propose directions for future work in section 7.

2 Dataset

We use the SQuAD dataset [4], a reading comprehension resource obtained through crowdsourcing question and answer pairs over Wikipedia articles. The questions are framed in the context of paragraphs, and the answers to these questions consist of spans within their corresponding paragraphs.

Our training set consists of 81,381 question, context, and answer triplets, while our validation set consists of 4,284 such triplets. We test on a development set with approximately 87,000 triplets, and report our results on this set for intermediate models. We report our final results on the hidden test set of 9,533 triplets.

An answer in this dataset consist of a pair of indices that correspond to the beginning and end of the answer span in the paragraph. Therefore, there is an opportunity for malformed labels, when the end index is before the start index. In training, we choose to ignore such examples. There are approximately 2,914 such malformed examples in the training set used.

In order to correctly set our models hyperparameters and appropriately balance accuracy with memory consumption, we examined the lengths of the questions and paragraphs (see figures 1 and 2).

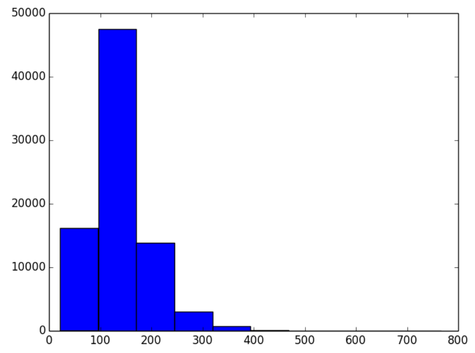


Figure 1: Lengths of paragraphs in SQuAD.

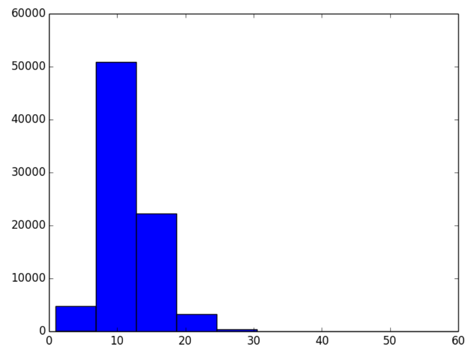


Figure 2: Lengths of questions in SQuAD.

Given these statistics, we set our maximum paragraph length to be 250 and our maximum question length to be 40. Examples with spans that have endings after this maximum paragraph length are tossed out during training as well (355 example triplets).

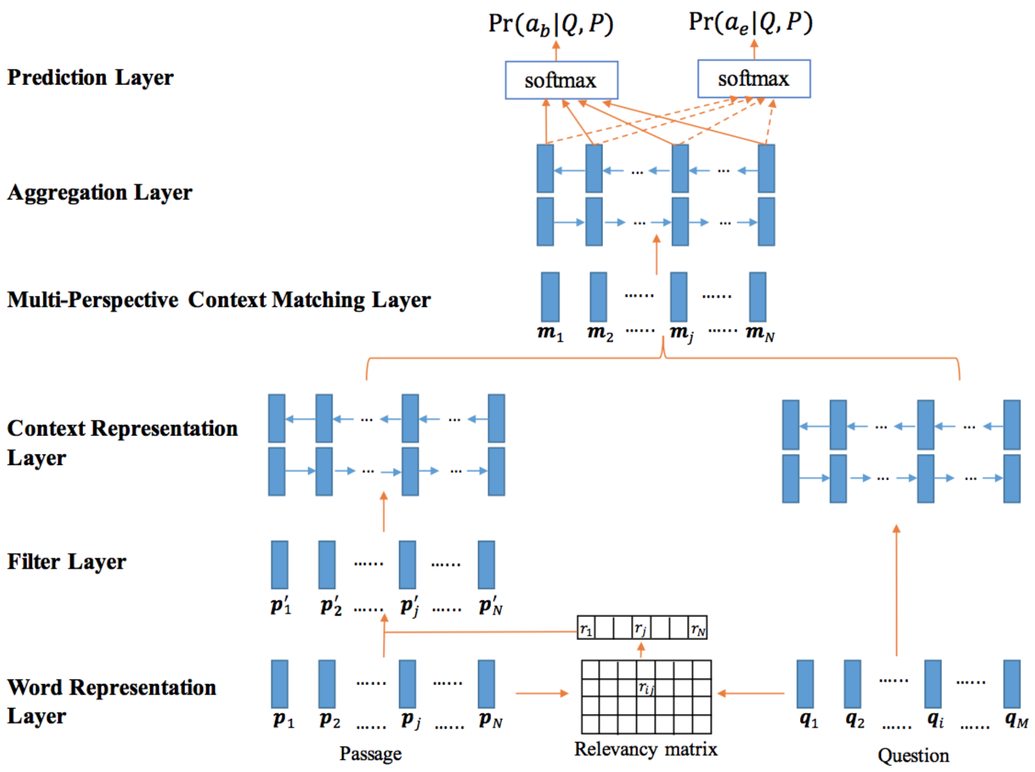
3 Models

Our primary model is an implementation of the multi-perspective context matching model (MPCM), utilizing the same architecture described in Wang, et. al [5]. In the following sections we describe our version of this base model as well as our extensions.

3.1 Base Multi-Perspective Context Matching Model (MPCM)

Multi-Perspective Context Matching (MPCM) is composed of a number of layers, depicted in figure 3. Given a paragraph P and a question Q, the model passes these representations through six total layers, ultimately outputting the probability that each index of the paragraph is the start and end index of the answer span.

Word representation layer. In this first layer, the authors use character and GloVe or Word2Vec embeddings to represent the words in both the paragraph and the question. In our implementation, we use only GloVe embeddings [3] truncated at 300 dimensions and choose not to include character



$Pr(a_b|Q,P)$
$Pr(a_e|Q,P)$
Prediction Layer
softmax
softmax
Aggregation Layer
Multi-Perspective Context Matching Layer
m_1 m_2 ...... m_j ...... m_N
Context Representation Layer
Filter Layer
p'_1 p'_2 ...... p'_j ...... p'_N
r_1 r_j r_N
r_{ij}
Word Representation Layer
p_1 p_2 ...... p_j ...... p_N
q_1 q_2 ...... q_i ...... q_M
Passage
Relevancy matrix
Question

Figure 3: Base MPCM model as implemented in Wang, et. al [5]

.

embeddings . At the end of this layer, we have $Q = [\mathbf{q_1}, \mathbf{q_2}, ...\mathbf{q_M}]$ and $P = [\mathbf{p_1}, \mathbf{p_2}, ...\mathbf{p_N}]$.

Filter layer. Words in the paragraph are weighted by the maximum relevance to any word in the question. This is done through the calculation of a relevancy matrix R, where $r_{ij} = \frac{\mathbf{q_i}^\mathbf{T}\mathbf{p_j}}{||\mathbf{q_i}||\cdot||\mathbf{p_j}||}$. Each $\mathbf{p_i}$ is weighted by its relevancy degree r_j where $r_j = \max_{i \in M} r_{i,j}$, to give us $\mathbf{p'_j} = \mathbf{r_j} \cdot \mathbf{p_j}$. We pass $P' = [\mathbf{p'_1}, \mathbf{p'_2}, ...\mathbf{p'_N}]$ and $Q = [\mathbf{q_1}, \mathbf{q_2}, ...\mathbf{q_M}]$ into the next layer.

Context representation layer. Time-steps of the paragraph and question are mixed together in order to capture contextual information in the paragraph and question. We use the same bi-LSTM to encode P' and Q.

Multi-perspective matching layer. Three matching vectors are calculated for each forward and backward state in the paragraph, for a total of six matching vectors per paragraph position. The elements of the resulting matching vectors are called perspectives, and are denoted with the subscript k. Each of the six matching vectors is computed using a different weight matrix W. The matching vectors all are generated using the same basic function:

$$m_k^z(v_1, v_2) = \cos(W_k^z \odot v_1, W_k^z \odot v_2)$$

where cos is the cosine similarity function and z corresponds to a type of matching vector (of the three options described below) and a particular direction (forward or backward).

Full matching: For each token in the paragraph p_j, compute

$$m^z(p_j, q^*)$$

where q^* is the final and initial hidden state of the question words, for the forward $(z = 1)$ and backward $(z = 2)$ direction respectively.

Mean-pooling matching: For each position in the paragraph p_j, compute

$$\frac{1}{M}\sum_{q_i} m^z(p_j, q_i)$$

This is done for the forward $(z = 3)$ and the backward $(z = 4)$ directions.

Maxpooling-matching: For each position in the paragraph p_j, compute

$$\max_{q_i} m^z(p_j, q_i)$$

This is done for the forward $(z = 5)$ and the backward $(z = 6)$ directions.

The result of the entire matching layer is the concatenation of all these results $(z = 1 \ldots 6)$ for each paragraph token p_j. The shape of this output is a vector of length six times the number of perspectives for each p_j.

Aggregation layer. Similar to the context representation layer, the aggregation layer uses a bi-LSTM to mix the outputs of the matching layer. In this way, it allows the model to further capture interactions between different matching vectors.

Prediction layer. The aggregation vector for each time step is fed through two different two-layer neural networks that each end in a softmax. The hidden layer in each network uses tanh for its nonlinearity. One of these neural networks produces the probability distribution over all paragraph positions for the start index, and the other for the end index.

Since we have padded paragraphs, there is the possibility to predict indices that are past the end of the paragraph. In order to prevent this, we add an exponential mask to the outputs before applying the softmax. The mask adds 0 for each position in the paragraph, and $-\infty$ for each position past the end of the paragraph. This results in a probability of zero for each index that is out of bounds of the paragraph.

3.2 MPCM Using GRUs

For our first experiment, we replace the LSTM cells in the context representation layer and the aggregation with GRU cells. This approach is motivated by the comparable performance of GRUs and LSTMs.

3.3 MPCM With Length Prediction

In this variation, we change the interpretation of the final prediction for the end index. Instead of predicting the start and end indices of the answer in the paragraph, we predict the start index and the length. The intuition behind this change is that the length seemed to be in some sense more predictable. Our implementation further incorporates this intuition by introducing a new hyperparameter $.5 < \alpha < 1$. The intended result of α is to weight the loss generated from the start prediction more than the loss generated from the length prediction.

3.4 MPCM With Question Summaries

We replace the filter layer with a different relevancy-based modification, inspired by co-attention [6]. In this scheme we take the same relevancy matrix R, but we apply a softmax operator, normalizing each column to produce R_q. This can be interpreted as a conditional distribution where $(R_q)_{i,j} = P(q_i \mid p_j)$. We then take the expectation over the question tokens for each paragraph token $\mathbb{E}[q_i \mid p_j] = QR_q$. We then concatenate this matrix with P. The result is that each element in the paragraph is accompanied by a corresponding summary of the words in the question.

3.5 Learning and Loss Across All Models

We use batched stochastic gradient descent with the Adam optimizer and a learning rate of 0.0001. Our batch size is 32 examples, and we run 10 total epochs. Our loss is the cross entropy loss on the softmax predictions for the start and end (or length) indices, using one hot vectors of paragraph end indices (or lengths) as the labels. We also apply dropout at each layer with a drop rate of 0.2.

4 Evaluation

To evaluate the performance, we use two metrics: F1 and EM (exact match). F1 is defined as the harmonic mean of precision (P) and recall (R):

$$F1 = \frac{2PR}{P+R}$$

Precision is the ratio of the number of correctly predicted tokens in the span divided by the number of total predicted tokens in the span. Recall is the ratio of the number of correctly predicted tokens in the span divided by the number of total tokens in the ground truth span. EM is a stricter metric that simply assigns 1 to an example if the span matches perfectly, and 0 otherwise.

We report the average of both of these metrics over the entire development/test dataset.

5 Related Works

Our experiments build heavily off of previous work in the question answering space. We use Wang et. al [5] as a spring board for our various experiments. One shortcoming of that model– and indeed any model that predicts the start and end indices directly– is its ability to predict invalid "backwards spans", where the end index is before the start index. Another potential pitfall of the model is the loss of paragraph information early on, in the filter layer: the original paragraph word embeddings are mutated according to their relevance to the question word. Furthermore, in taking the maximum over all question words for each paragraph word, the filter layer only captures the relationship of a paragraph word with one question word, rather than with all of them. The coattention model proposed in Xiong et. al [6] does not suffer from either of these problems, as it calculates a version of this relevancy (the dot product instead of cosine distance) for each question word and paragraph word and then concatenates it with the original paragraph representation.

6 Results

The results from each experiment on the development set can be seen in table 1. The MPCM with Question Summaries model performed best, followed by the base MPCM model. The training loss for this model is displayed in figure 4. On the test set MPCM with Question Summaries achieved an F1 of 45.604 and an EM of 32.781.

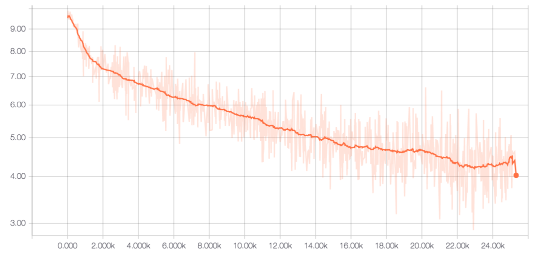


Figure 4: Training loss for MPCM with Question Summaries model.

7 Error Analysis and Future Work

Our initial experiments extracted answer spans using the GloVe vocabulary [3]. For unknown tokens, this sometimes resulted in examples where the model predicted the span correctly, but the preprocessed paragraph had an UNK id corresponding to the unknown token. This penalized the

Model	F1	EM
MPCM with Question Summaries	45.865	33.273
MPCM	44.472	30.719
MPCM with GRU	43.537	29.537
MPCM with Length	33.711*	18.117*

Table 1: Performance on the dev set for each model implemented. Metrics marked with * were polluted with <unk> tokens during evaluation (though the ranking shown here held when all metrics were thus polluted).

performance. We corrected for this in subsequent experiments where we used the indices to take the excerpt from the original paragraph. This boosted our performance by about 5% in F1 and EM.

As we said, in experiments where the model predicted the start and end indices (instead of the length), there was the potential to predict backwards spans. This resulted in empty answers. This occurred on average in 12.4% of examples during evaluation. Better tuning of the length prediction model could help address this issue in the future.

Our model also made some more linguistically interesting errors. Many of the errors were spans that were not valid grammatical constituents, and therefore would not be an appropriate answer for any question. For example, when answering the question *Who stripped the ball from Cam Newton while sacking him on this drive?* regarding a football match, the model answers with *linebacker Von Miller knocked*. This suggests that the model could benefit from more grammatical guidance, say from a constituency or dependency parser.

The ground truth answers given in the dataset tend to be concise. For example, in a piece about complexity theory, the phrase *complexity classes* is always preceded by the phrase *important*. The model therefore has no reason to distinguish between the phrases *complexity classes* and *important complexity classes*, despite the fact that the former would normally constitute a better answer, as it is more concise. This again suggests that parser guidance might be helpful, along with a bias to prefer shorter constituents if they are similar in meaning to longer constituents.

One idea for further augmentation of the model builds on the success of the questions summaries that we added to the paragraph representations. It is conceivable that adding paragraph summaries to each question word would also be helpful in relating the paragraph to the question. One could compute the expected paragraph vector for each question word in the same way that we did the reverse, and augment the questions vectors with this information.

References

[1] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, et al. Building watson: An overview of the deepqa project. *AI magazine*, 31(3):59–79, 2010.

[2] C. Kwok, O. Etzioni, and D. S. Weld. Scaling question answering to the web. *ACM Trans. Inf. Syst.*, 19(3):242–262, July 2001. ISSN 1046-8188. doi: 10.1145/502115.502117. URL `http://doi.acm.org/10.1145/502115.502117`.

[3] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

[4] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[5] Z. Wang, H. Mi, W. Hamza, and R. Florian. Multi-perspective context matching for machine comprehension. *arXiv preprint arXiv:1612.04211*, 2016.

[6] C. Xiong, V. Zhong, and R. Socher. Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604*, 2016.

Contributions of team members: We pair programmed the vast majority of coding tasks, and evenly divided up what we did not pair program. We co-wrote the paper and poster as well.