Codalab: tyler-romero, zbarnes, FrankCipollone

Machine Comprehension with MMLSTM and Clustering

Tyler Romero
Stanford University
555 Salvatierra Walk
Stanford, CA 94305
tromero1@stanford.edu

Zach Barnes
Stanford University
135 Running Farm Lane
Stanford, CA 94305
zbarnes@stanford.edu

Frank Cipollone
Stanford University
217 Ayrshire Farm Lane
Stanford, CA 94305
fcipollo@stanford.edu

Abstract

We evaluate two new models for machine comprehension. MMLSTM builds on the work of Match-LSTM with Answer Pointer by adding more layers of Match-LSTMs that read both question and context one additional time. Question Clustering is based on the idea that there are many different types of questions, and that there may be a performance benefit based on clustering questions and training a different model for each cluster.

1 Background

1.1 Problem Description

The goal of the reading comprehension task is to enable a machine to understand and answer questions about a passage. This has recently become a feasible task for deep learning due to the creation of the SQuAD dataset, which is orders of magnitude larger than any other human-created question-answer dataset. The SQuAD dataset also has the desirable property that all correct answers are contiguous spans within the context passage.

The two most common ways to evaluate model performance on the reading comprehension task are F1 and Exact Match scores.

1.2 Data

Our model is trained on the SQuAD dataset [1]. We split the SQuAD dataset up into 82k training questions, 5k validation questions, and 10k dev questions. The set of test questions is withheld.

We use 300d Common Crawl GloVe [2] vectors for our word embeddings.

1.3 Related Work

Our baseline model is based upon the work done by Shuohang Wang and Jing Jiang on Machine Comprehension Using Match-LSTM and Answer Pointer [3]. The Match-LSTM model uses LSTMs to encode the question and context, and uses a Match-LSTM in order to gain some understanding of the contextual entailment within the context passage.

2 Approach

We discuss methods we used in order to improve baseline performance. In addition, we propose two new models for the reading comprehension task: Question Clustering (QC) and Multi-Match-LSTM (MMLSTM).

2.1 Improvements to Baseline

We made several significant improvements to the baseline Match-LSTM model in order to improve

performance and generalization.

In order to improve training time, we set a max context (passage) length of 400. Any passages of length less than 400 get padded to a length of 400, and any passages with length greater than 400 get truncated. We chose a length of 400 based on the histogram in figure 1. Similarly, we pad questions to a length of 30 based on the histogram in figure 2.

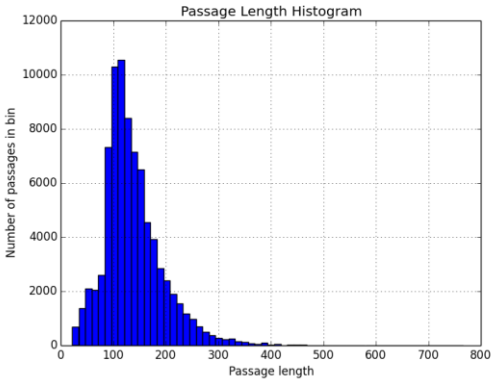

Figure 1: Passage Lengths

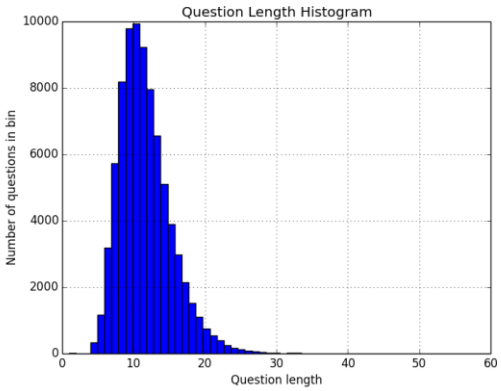

Figure 2: Question Lengths

The decoder of the Match-LSTM model is an Answer-Pointer decoder. It returns a start index and an end index, which are used to slice the context paragraph to produce an answer. Unconstrained, there is a chance that the predicted end index is in front of the start index. Clearly this is not desirable. There are several viable methods of resolving this, such as simply switching the end index and the start index. However, we chose to use a window search in order to find the optimal span, based on the probabilities we have computed. Based on the histogram of answer lengths in figure 3, we limited our window search to spans of length 15.

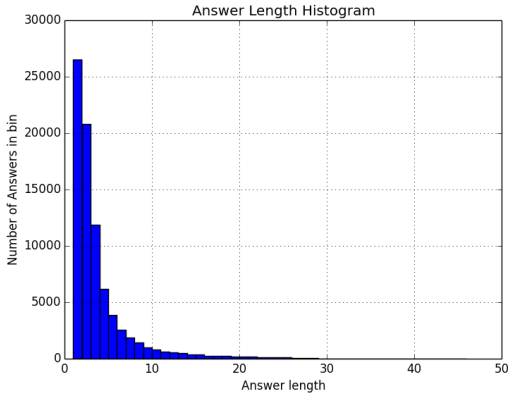

Figure 3: Answer Lengths

We added several improvements to our baseline's infrastructure as well. We made our Answer-Pointer decoder bi-direction, which helped our model by allowing it to analyze the encoded question/context both forwards and backwards. In addition, we deepened our model by adding two generic LSTM cells prior to the Match-LSTM layer.

In order to combat overfitting, we added dropout regularization to our input question and context representation, as well as to our encoder output.

2.2 Multi-Match-LSTM (MMLSTM)

Human tend to read passages in a different manner than our baseline model does. Instead of reading the question, and then reading the passage, humans tend to read the question and then

the passage repeatedly until they can fully comprehend the relationship between them.

With Multi-Match-LSTM we take a step towards simulating that behavior.

We created a multi-layered network that used 3 Match-LSTMs to handle "rereading". In this extension of Match-LSTM, the first layer reads the encoded question and paragraph and generates a representation of the entailment with the context as the premise and the question as the hypothesis. We use this representation in the second layer to then "reread" the question by now using the hidden representation of the first layer as the hypothesis and the encoded question as the premise. In the last layer, we then use the entailed context (from the first layer) as the premise and then the entailed question (from the second layer) as the hypothesis in a final Match-LSTM layer. This layer's output is then connected to the answer pointer layer, similar to vanilla Match-LSTM. This architecture can be seen in figure 4.

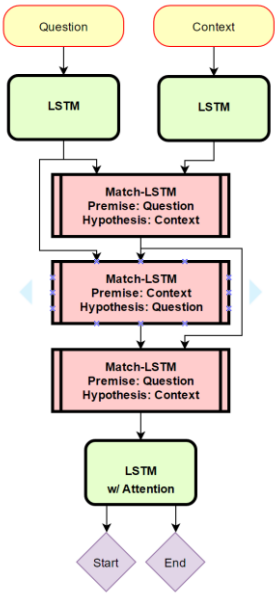

Figure 4: MMLSTM architecture

Our intention was to allow for the model to gain a more nuanced understanding of the question/context pair, as well as to allow for better generalization due to the increased depth of the model.

2.3 Question Clustering

In the Stanford Attentive Reader paper, the authors examine the CNN/Daily mail data for the reading comprehension task [4]. In the paper, the authors note that the example contexts and questions given for the task are not all of the same type. In fact, the authors sort the data into different groups, based on how the question should be answered. For example, some of the questions are almost exactly matched to the sentence containing the answer in the context paragraph, while other questions require multiple different sentences to answer.

We took a look into the SQuAD dataset in the hopes of finding a similar split. After calculating various statistics over our training questions and contexts, we noted some differences in the ways the different questions should be answered.

We decided to attempt to cluster our data into groups, and to learn a separate model for each group. We began with our original baseline Match-LSTM model, and defined the following framework:

1. Cluster our training data into the groups $\{C_1, C_2, \ldots, C_n\}$
2. Retrain our baseline model n times with training data $\{C_1, C_2, \ldots, C_n\}$ in order to

generate models $\{M_1, M_2, \ldots, M_n\}$

3. Make predictions by assigning a new data point p to cluster C_i and predict on p using M_i

The two remaining decisions are the choice for the number of clusters and the choice of method of clustering.

In deciding our method of clustering, we wanted to capture the idea that there are distinct groups of questions that require different methods of question answering. The best representation we had of the way in which a question should be answered was the intermediate step of our Match-LSTM model. We wanted to cluster the encoding created by our model in order to find groups of questions that should be answered similarly. We refer to this encoded matrix as IM from here on.

As IM did not have a human-readable meaning, and was a very large matrix of size $(300\ x\ 300)$, we ran two autoencoders (one in each dimension of the matrix) in order to reduce the dimensionality to size $(1\ x\ 1000)$. The autoencoders worked by first encoding the data in a series of steps:

$$M_t = M_{t-1} * T_t$$

Where T_t is a trainable matrix of size (R, C) with R equal to the second dimension of M_{t-1}, and C equal to some smaller number.

The decoding step worked by reversing this operation with different trainable parameters. In order to train one autoencoder per dimension, we ran the first autoencoder to reduce dimensions to $(300\ x\ 20)$, then took the transpose of the matrix and ran the second autoencoder to reduce dimension down to $(20\ x\ 50)$. Finally, we flattened this matrix to be $(1\ x\ 1000)$ As is typical with autoencoders, we used a dropout rate of $.5$ on each layer and used the same matrix for the input and output.

From there, we used the k-means algorithm to create $k = 3$ clusters on these resulting encodings. Finally, to evaluate the quality of our clusters, we first visualized our data by running PCA to further reducing the dimensionality of each question representation to 3, and graphed the result (figure 5).

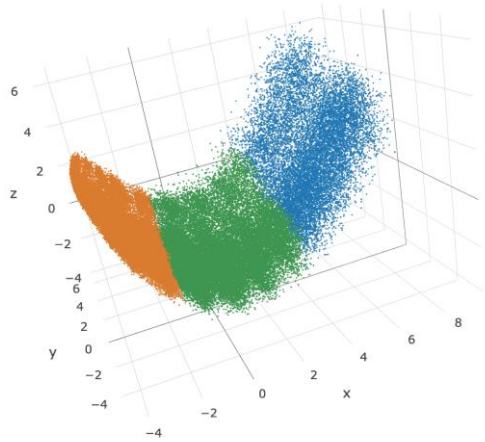

Figure 5: Clustering of Question Representations

This result is interesting. While we were hoping for more defined cluster shapes, we were encouraged by the mapping into some recognizable shape, and the splitting of data into three sections of this hopefully meaningful space.

We further examined the data in each cluster and found most notably that each cluster contained significantly different context lengths (table 1).

Table 1: Cluster Context Lengths

Cluster Number	Avg. Length	Min Length	Max Length
1	323.7	174	766
2	147.80	116	196
3	94.42	22	134

While this indicates that the clustering was to some extent successful, we analyzed the data and were not able to find many more similarities among each cluster. We hypothesize that because of the nature of our matrix EM, the context length feature overpowered other features in the clustering, and that we mainly captured differences in context length.

Regardless, we decided to move forward. We took the data from each cluster and trained three new models. During testing, we took each test data point, ran it through the two autoencoders, assigned it to the nearest cluster, and predicted using the model associated with that cluster.

3 Results and Analysis

3.1 Hyperparameters and Experimentation Details

After spending a significant amount of time tuning hyperparameters, our values are as follows: $learning\ rate = 0.001$, $learning\ decay\ rate = 0.95$, $dropout\ rate = 0.4$, $state\ size = 150$, $embedding\ size = 300$, $epochs = 30$.

We used the Adam Optimizer in order to allow our model train with an adaptive learning rate. In addition, we used early stopping, based on F1 score, in order to help prevent overfitting.

3.2 Results

We present our results in Table 2.

Table 2: Results. s = search, do = dropout, dp = deep

Model	**|Θ|**	**Exact Match**		**F1**	
		Dev	**Test**	**Dev**	**Test**
Baseline	1.5M	33.4	-	44.1	-
Baseline + Question Clustering	4.5M	34.4	-	44.1	-
Baseline + s + do	2.1M	43.2	-	52.4	-
Baseline + s + do + dp	2.5M	**46.2**	**42.5**	**56.1**	**53.04**
MMLSTM + s + do	3.0M	41.8	-	51.5	-

3.3 Analysis

We note that our test performance is a significant amount lower than our dev performance. We hypothesize that this is due to a larger percentage of vocabulary without a mapping to a GloVe word vector. We build our vocabulary by including all words with GloVe representations from both the train and dev sets. We believe that test performance could easily be at the same level as dev performance for our model if we used a much larger vocabulary.

After spending a significant amount of time tuning our Match-LSTM baseline, we have concluded that it is a very difficult model to optimize. Minor changes to our hyperparameters caused significant changes in the performance of our model.

Tuning MMLSTM proved even more difficult than tuning our baseline. It required an extremely low learning rate in order to make progress. Ultimately, MMLSTM did not outperform our baseline. However, we were not able to spend as much time tuning hyperparameters for MMLSTM as we would have liked; we believe that with more time to tune MMLSTM, it could outperform our baseline.

Question clustering did not offer any performance improvements either. We believe that this is, in a large part, due to the training set size. By splitting the training set into several clusters, we severely limit the amount of training data available for each cluster model. We hypothesize that if the overall training set were several times larger, this method would have a greater chance of success. Also, in addition to the hyperparameters required to tune our other models, there are extra hyperparameters required to tune question clustering. Namely, number of clusters and encoding size. We used three clusters, but it is not hard to imagine that there could be more clusters. For example, one for who, what when, where, why, how.

We did try clustering on a much simpler feature set. We looked at the first word of each question, and found that around half of all questions began with the word "What," while the rest began with other words, most notably "Who," "When," "Where," and "How." Because of the limited amount of data, we decided to train only two separate models, one for the "What" questions, and one for everything else. This model was not able to improve upon the baseline either.

3.4 Future Work

Due to the time sensitive nature of this project, we are left with many ideas for future work. First and foremost, we believe that further hyper-parameter tuning could bring our model to its full potential, and increase our F1 and EM scores on the test set. Secondly, we spent much of our time focused on the Match-LSTM solution, but we believe that combining this idea with other recent literature on the topic could allow for further exploration of solutions to the problem of question answering.

In particular, we are interested in extending our attempt to provide a method which reads the question and the context more than once. We believe that combining our Match-LSTM model with a model with a more intelligent extension could offer a large improvement in the ability of our model to answer more general and arbitrary questions. An example of a model which implements this improvement very successfully is ReasoNet [5]. ReasoNet not only uses multiple layers of attention to solve the problem, but it also uses Reinforcement Learning to decide how many layers to use. This method allows the model to generalize to many different ways of answering questions, and we would like to experiment with it in combination with our current model.

Lastly, we were dismayed at the lack of success we saw from our clustering method. We would like to spend more time examining the examples from the dataset to arrive at a better way to cluster. We would also like to optimize our hyper-parameters and test new and different things. We think that it is possible that our data does not actually contain distinct enough clusters to meaningfully boost the performance of our model, but we are not yet ready to reject our hypothesis that it does.

Acknowledgments

We would like to thank Richard Socher and Christopher Manning, the instructors for this course, as well as all of the TA's. We would also like to thank Microsoft for the access to Azure instances to train our models. Finally, we would like to thank Shuohang Wang, an author of the paper "Machine Comprehension Using Match-LSTM and Answer Pointer," who we reached out to for advice on training our model.

References

[1] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text.

[2] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

[3] Shuohang Wang and Jing Jiang.2016. Machine Comprehension using Match-LSTM and Answer Pointer.

[4] Danqi Chen, Jason Bolton, and Christopher D. Manning. 2016. A Thorough Examination of the CNN / Daily Mail Reading Comprehension Task. In Association for Computational Linguistics (ACL).

[5] Shen, Y., Huang, P.-S., Gao, J., and Chen, W. (2016). ReasoNet: Learning to Stop Reading in Machine Comprehension. ArXiv e-prints