

Implementation and Analysis of Match-LSTM for SQuAD

Michael Graczyk

Abstract—Wang and Jiang recently proposed an end-to-end neural system solving the Stanford Question Answering Dataset (SQuAD) task [1]. The model consists of their textual entailment model called Match-LSTM, and a new sequence model for finding answers in context paragraphs. We reimplemented the model using Keras and Tensorflow [2][3]. Although we were not able to reproduce the original system’s accuracy, here analyze our system’s performance on the dataset and describe the process used to develop our implementation. We also offer suggestions and lessons learned concerning the practical training of deep sequential models.

1 INTRODUCTION

For decades, question answering (QA) has been an open research topic and area of great commercial interest within natural language processing (NLP). Although approaches based on computational linguistics and hand built structural models have had some success at the task for many years, the recent explosion in usefulness of deep neural networks has allowed computer scientists to greatly improve performance of automated QA system.

The recently released SQuAD dataset for QA models has attracted many NLP researchers [4]. Seemingly countless papers have been authored since its release proposing new models for accomplishing the task. Much of the difficulty in SQuAD, and perhaps the source of its popularity, is that the answers are selected freely from the context paragraph, with no additional structure or constraints imposed on the learning problem. The central difficulty is to somehow combine the question and arbitrarily long context paragraph in a way that exposes the answer. Most models use some form of attention, effectively using the context and question to jointly decide which sections of the context are relevant to the answering task.

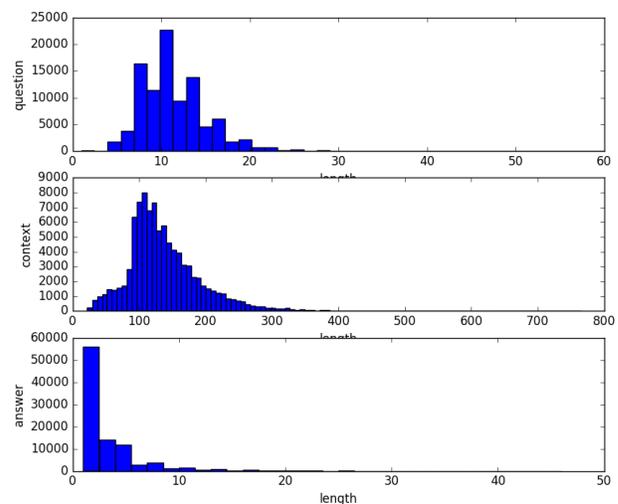
One such attention based approach is Match-LSTM, the topic of this paper. Match-LSTM attempts to solve this task by layering a basic attention mechanism on top of a simple sequential feature representation for both the context and question. The system then uses another sequential model to “point” to the boundaries of the answer within the context. We will explain how the model combines the question and answer, and how its specifics could lead to deficiencies.

We attempted to reimplement Match-LSTM using Keras and Tensorflow in order to verify its creator’s results and to expand it in a nontrivial way. Unfortunately, we were not able to create an implementation that achieved performance anywhere close to the results claimed in the Match-LSTM paper. This is almost certainly due to bugs in our implementation. Our implementation failed to learn almost any long term context. It was only able to correctly answer questions in which the answer and question were lexically similar. We will describe our implementation and its problems throughout the rest of the document.

2 SQUAD DATASET

Before we explored possible models and eventually decided to implement Match-LSTM, we analyzed the SQuAD dataset for size and structure. We noticed that the sizes of the questions and paragraphs varied wildly, so we decided to implement a model which could be trained or tested with arbitrary size inputs. We also found that the vast majority of context paragraphs were shorter than 300 words, and that typical answers were 1-2 words long. The bottleneck in training sequential models with LSTMs is typically in processing the LSTM layers sequentially. In response, we decided to truncate paragraphs at train time to 300 words. Later, when time became short and it became apparent that our implementation was not going to perform well, we further reduced the maximum size paragraph input to 150 words. We

Fig. 1. SQuAD Data Set Length Distribution



SQuAD Size Statistics

```

***** question *****
count: 95932
mean: 11.3092503023
var: 13.8884467256
median: 11.0
min: 1
max: 60

***** context *****
count: 95932
mean: 138.194637868
var: 3314.18656675
median: 127.0
min: 22
max: 766
\% <= 300: 98.2383354876

***** answer *****
count: 95932
mean: 3.36944919318
var: 13.9848223919
median: 2.0
min: 1
max: 46

```

3 MATCH-LSTM

Here we will describe Match-LSTM and our implementation at a high level. Please refer to the paper [1] for more detailed information.

Match-LSTM attempts to recover an sequence of positions within a context paragraph using the contents of the context and an associated question. The model begins by preprocessing both the context and the question with LSTM layers. The authors claimed to remove output gates from their LSTM layers, which reduces the number of parameters in each LSTM by 25%. The author's also experimented with including an additional "backwards" LSTM layer for each input, but found that the backwards layer had little affect on performance.

The input LSTMs are followed by a so-called "Match-LSTM" layer. The layer consists of a modified LSTM whose inputs are a nonlinear function of the input LSTM outputs and the previous Match-LSTM state. At each timestep, the Match-LSTM computes an attention vector over the question, using its previous state and the input layers. It uses this attention vector to collapse the question into a single feature vector, effectively associating with each context paragraph token a linear combination of all words in the question. The attention-weighted question representation is concatenated with the paragraph representation to form the Match-LSTM input. In this way, the Match-LSTM should be able to learn a sequential representation of the paragraph that is semantically relevant to the particular question.

The authors provide two final model layers, both of which learn to represent the answer sequence in some way. Their first layer, called the "sequence model" in the paper, is more complicated and performs worse than the second. Their second layer, the only one of the two we implemented, simply predicts the beginning and end position of the question using a single-step LSTM. The output LSTM uses an attention mechanism almost

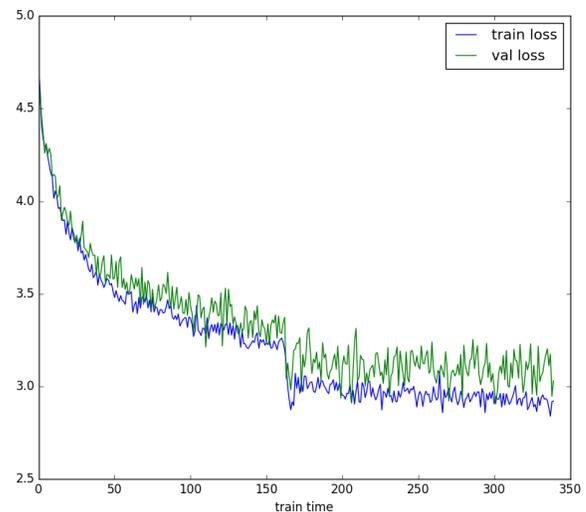
identical to Match-LSTM attention, except that it computes an attention vector over the Match-LSTM output rather than the input layer. This final attention vector is passed through a softmax activation and used as a distribution over answer boundaries. The authors describe several mechanisms to convert these soft boundaries into hard decisions, claiming that a global search gives them significantly better results than simply taking the *argmax* of both boundary probabilities.

Our implementation followed the paper almost exactly. Our of convenience, we did not remove the output gates from the input LSTM layers as was done in the original paper. In addition, we added dropout with probability 0.5 to the LSTMs and large dense layers despite dropout not being explicitly mentioned in the paper. We used the Common Crawl 840B Glove vectors with 300 dimensions as our input embedding, and like the paper we did not train these vectors. Finally, we used softsign instead of tanh for a substantial improvement in training performance with no obvious change in metrics. We used Keras for the entire model, and only had to write special purpose code for the Match-LSTM inputs. Even then, Keras abstractions made it easy to implement Match-LSTM as a small tweak to the existing LSTM layer.

4 ANALYSIS OF RESULTS

Our implementation did not perform nearly as well as the system in the original paper. Our test set performance was 32.014% F1 21.829% EM, compared to approximately 70% and 60% in the paper. At this time, we do not know exactly why our model performs so poorly. As shown in the figure below, it was trained until convergence on the entire train dataset several times. The drop near epoch 150 was caused by learning rate annealing.

Fig. 2. Model Losses During Training



We examined some of the cases in which our model failed. Most of the failures involved sentences longer than 50 words, or answers longer than 2 words. In particular, our seemed to almost always predict that an answer was 2 words or fewer. The figures below show the distributions of our model's answers, and how those answers are small and close the beginning of the paragraph compared to typical answers in the dataset. We are

Fig. 3. Actual Answer Lengths vs. Those Our Model Correctly Predicted

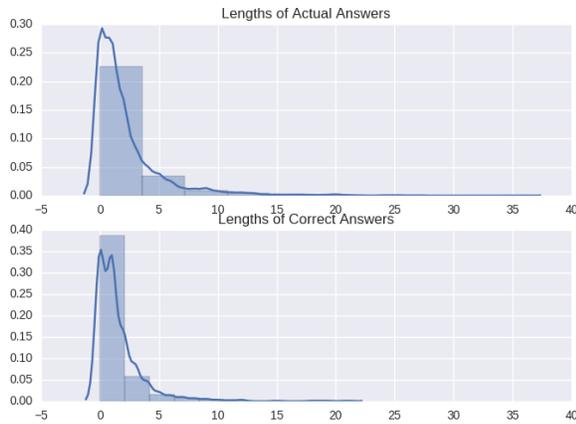
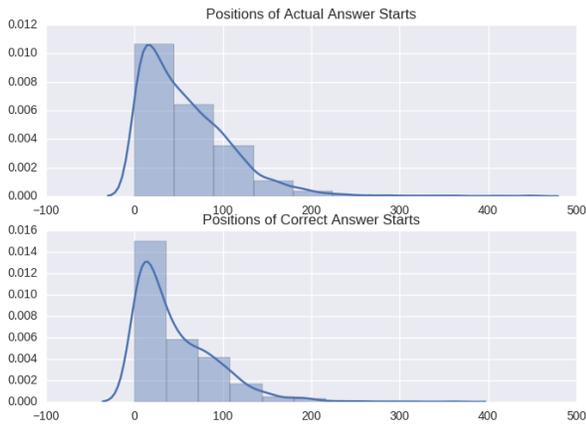


Fig. 4. Actual Answer Start Positions vs. Those Our Model Correctly Predicted



hoping that this fact could help us determine what is wrong with the implementation.

In addition, we found that our model made many mistakes in confusing words which were similar to the answer, but could be disambiguated with context. For example, in one case it mistook one number in the paragraph for the answer:

Question: "What was the population Jacksonville city as of 2010?"

Predicted Positions: (67, 67) wp (0.910357356071, 0.274364680052)

Actual Positions: (109, 109) from [109 109]

Answer Words: 1,345,596

Predicted Words: 853,382

Error Context: "with an estimated population of 853,382 in 2014"

Actual Context: "Jacksonville is the principal city in the Jacksonville metropolitan area, with a population of 1,345,596 in 2010."

It seems that our model is not able to handle even small dependencies backwards in time, even when the context clearly disambiguates otherwise similar answers.

5 FUTURE WORK

Most of all, we would like to determine why our model does not work. We believe that the next steps in debugging would be to examine the code associated with the original Match-LSTM paper, as it is available online. It is possible that there is some discrepancy between their implementation and what we have implemented. Regardless, it should also be possible to implement a simpler model with better performance. We were able barely able to achieve results as good as a simple single layer recurrent network, which uses almost no context. It would appear that our network may also not be using context, although it is unclear how we would go about debugging that.

We would also like to implement a network which takes less time to train on modern GPUs. LSTMs have become exceedingly popular in recent years, especially for NLP. However, they are notoriously difficult to train, in part because doing so does not parallelize well on GPUs. Convolutional networks, on the other hand, are much easier to run efficiently on GPUs and have the potential to model temporal structure just as well as LSTMs. We would have liked to explore the possibility of using a convolutional layer instead of the paragraph representation and Match-LSTM layer. We even implemented such a model, but did not have time to fully debug it or train it.

6 CONCLUSION

We described the Match-LSTM model and our attempt to replicate it using Keras and Tensorflow. Unfortunately, our implementation did not work, for as yet unknown reasons. We believe that our model is incapable of learning long term temporal context, probably because of a bug in the software. We believe that with the right fix, it should at least be possible to replicate the results in the original Match-LSTM paper. Once those results are replicated, it should be possible to improve on them quickly using the efficiency of the Keras and Tensorflow environments.

REFERENCES

- [1] Shuohang Wang and Jing Jiang, "Machine Comprehension Using Match-LSTM and Answer Pointer". arxiv.org/abs/1608.07905
- [2] Francois Chollet, Keras. github.com/fchollet/keras
- [3] Google Inc. Tensorflow. tensorflow.org
- [4] SQuAD The Stanford Question Answering Dataset. rajpurkar.github.io/SQuAD-explorer
- [5] Jeffrey Pennington, Richard Socher, Christopher D. Manning. GloVe: Global Vectors for Word Representation nlp.stanford.edu/projects/glove/