Improving Match-LSTM for Machine Comprehension

Kevin Moody
kmoody@stanford.edu

Mike Yu
myu3@stanford.edu

Dennis Xu
dennisx@stanford.edu

Codalab Username
kmoody

Abstract

Machine comprehension is a critical problem that lies on the frontier of natural language processing. The Stanford Question Answering Dataset (SQuAD), offers a set of questions and their answers created by humans through crowdsourcing. We implemented an end-to-end neural architecture for the task based on Match-LSTM and Pointer Net, inspired by previous work done by Wang and Jiang in *Machine Comprehension Using Match-LSTM and Answer Pointer* (2016) as well as Vinyals et al. in *Pointer Net, a sequence-to-sequence model* (2015). We were able to achieve an F1 score of 0.65 and EM score of 0.53 through our implementation, which leverages a dynamic-programming search to extract the final answer.

1 Introduction

Question Answering is an application of NLP in which artificial intelligence is able to process and represent a question, and then respond with an appropriate answer. The Stanford Question Answering Dataset is a set of around 100,000 questions, combined with labeled answers which are part of some reference document - the use of a reference document allows question answering algorithms to use the document as a per-question knowledge base and thus prevents such algorithms from needing overtly extensive knowledge about the world.

In particular, a data point in the Stanford Question Answering Dataset consists of a question, a reference document, and a start and end position within the document which together provide the answer to the question. In this project, we attempt to train a neural network model that, given a question and reference document pair, predicts the start and end positions of the answer within the document.

We approach this problem by using deep learning techniques that lean heavily on Long Short Term Memory neural networks (LSTMs) and attention mechanisms that allow us to encode the responsiveness of words in the reference document to the question, and vice versa.

2 Explored Papers

We explored a few papers before deciding on an approach that blended feasibility with effectiveness and accuracy. We took a deep look first at the dynamic coattention network model presented by Xiong et al in Dynamic Coattention Networks for Question Answering. Because it seems to set a new bar for state-of-the-art, we thought this might be a good place to start. However, it was a bit ambitious; in particular, implementing the affinity matrix calculation and then using it with a Highway Maxout Network seemed challenging for a first attempt at solving this problem in the span of a couple weeks, and watering down the HMN with a biLSTM or something left results unpredictable, so we opted not to implement this model directly.

We also took a look at the bilateral multiperspective approach taken in Wang et al's Bilateral Multi-Perspective Matching for Natural Language Sentences. One big open question with the biMPM approach surrounds how one selects candidate answers to use the multi-perspective matching with the question. Obviously, there are $\frac{D^2}{2}$ possible substrings of a document of length D, and this makes running biMPM on each substring in a document on the test pass infeasible. Then, perhaps biMPM should be used in conjunction with other models, where the other models select some candidates for this pass - future work might move that direction.

Finally, we took a look at a simpler paper, Wang and Jiang's Machine Comprehension using Match-LSTM and Answer Pointer. This paper achieved very solid results, and utilized three layers which were pretty conventional, with the one unique piece being the middle match-LSTM layer, which computes attention vectors for each document token against the entire question. We took a pass at implementing this model for our project.

3 Our Approach and Implementation

As a first step, we need vector representations of words. As provided in the assignment, we used GLoVE vectors that were trimmed, rather than training our own word embeddings. Our word vectors were 100-dimensional GLoVE vectors, which constituted the first word representations fed into the LSTM. We also clipped inputs, capping documents at length 300 and questions at length 30 to increasing training speed of our epochs.

The obvious next step was to incorporate contextual information into the encodings of both the question and the document. In order to do this, we used a simple LSTM encoder - a simple standard one-directional LSTM on the question, and another (with different parameters) on the document. All LSTM states were of size 100 - the same size as the word vectors.

Before feeding this output into the next layer, we apply 0.2 dropout to the outputs of this preprocessing LSTM layer to prevent overfitting.

Once we'd encoded representations of both the document and question via the LSTM encoding layer, we computed attention vectors for each word in the document against the question. The idea here is to see how responsive each word is to the question - words that are more responsive to the question are more likely to answer it. We did this by using a word-by-word attention mechanism to compare each word in the passage to each word in the question. More specifically, we tile the document so that we can combine each word in the question with a full version of the document to create a weighted question, before concatenating that with the present word in the passage.

We then pass this concatenation into two LSTMs (one forward and one reverse) to incorporate the context (intuitively, if a word is very responsive to a question, and it neighbors other words that are also responsive, that entire sequence is very likely to be the answer, compared to a word that is equally responsive but neighbored by unresponsive words). This way, each token in the document gets a hidden state that reflects the token, the question, and the context before, and a hidden state that reflects the token, the question, and the context after. We combine these two hidden states and feed it as input into the pointer layer.

For our final layer, we train a pointer layer, which basically consists of two fully connected layers which use softmax to assign probabilities to each token in the passage being the start token and the end token of the answer, respectively. By doing this, we are able to get two probability distributions over the document, once which represents each token's probability of being the start of the answer, and the other which represents each token's probability of being the end of the answer.

Finally, we maximize the multiplied probabilities of the start and end tokens, conditioned on the start being before the end token and within some hyperparameter k (constraining the answer length). In our case, we use $k = 15$, and so we just look at every single start-end pairing with start before end and start and end not more than 15 tokens apart, and pick the pair with the highest probability. Brute

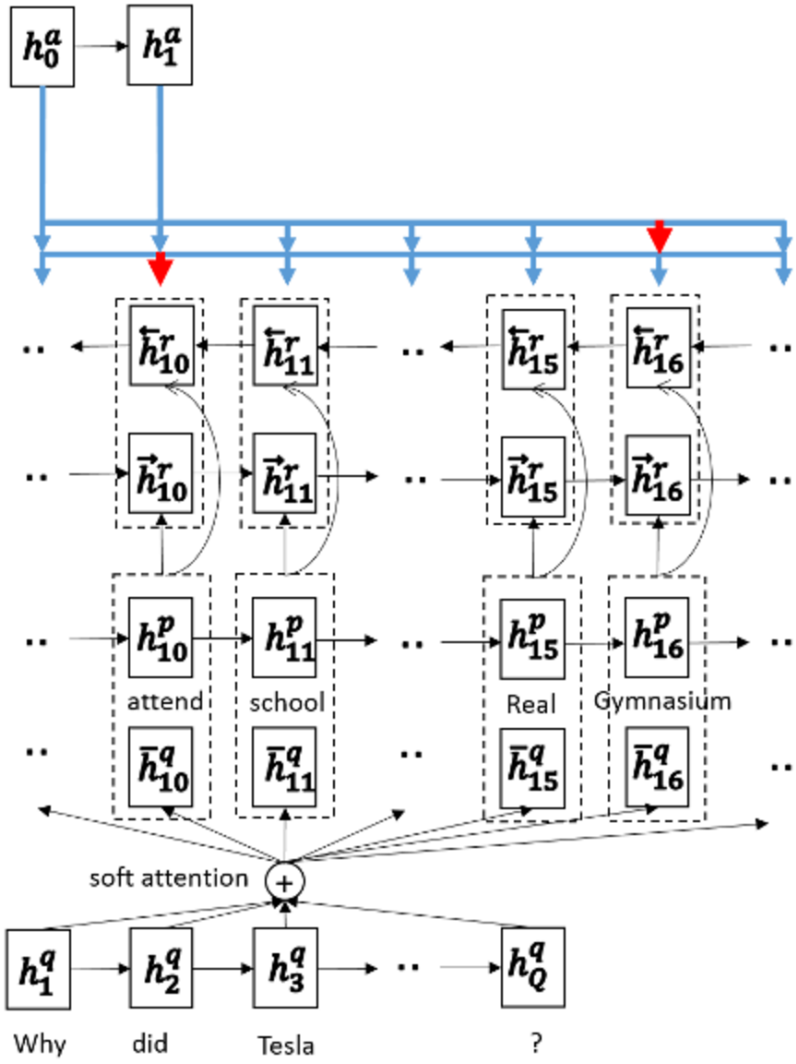
h_0^a
h_1^a
$\overleftarrow{h}_{10}^r$
$\overleftarrow{h}_{11}^r$
$\overleftarrow{h}_{15}^r$
$\overleftarrow{h}_{16}^r$
$\overrightarrow{h}_{10}^r$
$\overrightarrow{h}_{11}^r$
$\overrightarrow{h}_{15}^r$
$\overrightarrow{h}_{16}^r$
h_{10}^p
h_{11}^p
h_{15}^p
h_{16}^p
attend
school
Real
Gymnasium
$\bar{h}_{10}^q$
$\bar{h}_{11}^q$
$\bar{h}_{15}^q$
$\bar{h}_{16}^q$
soft attention
h_1^q
h_2^q
h_3^q
h_Q^q
Why
did
Tesla
?

Figure 1: Match-LSTM with Answer Pointer - credit to Wang and Jiang for the diagram

forcing this problem takes a while (just linear time, but with a multiplier of 15 and for each example), but we were able to implement a dynamic programming search which sped this up significantly.

4 Results

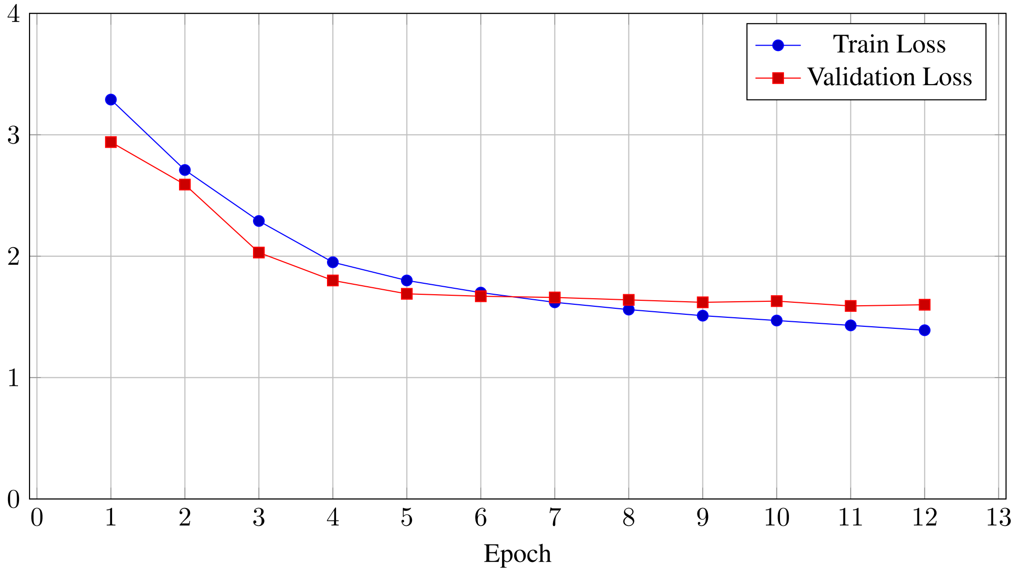

Figure 2: Training and Validation Loss

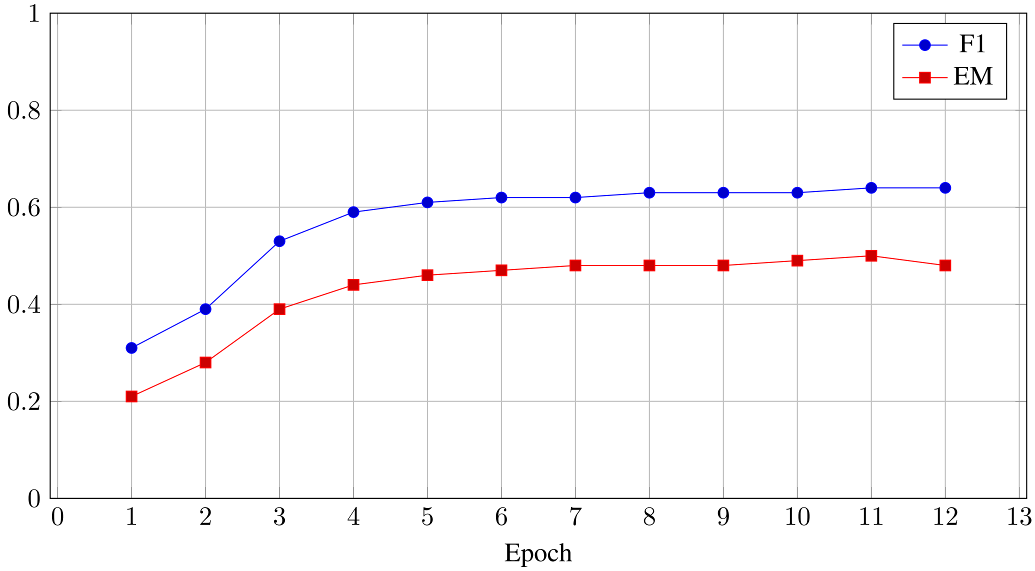

Figure 3: Performance on Validation Set

Our results for the first model we tried (with only 100 dimensional word vectors, and accordingly 100 dimensional LSTM layers) are graphed in Figure 3. This model was also trained without the dynamic search (rather just picking the start and end - the dynamic search boosted F1 by about 3%.

We'll note that we were able to get an F1 of around 62% with just this simpler model - we unfortunately didn't log the model with the search algorithm as comprehensively, so we were unable to draw the pretty graph. This, while not quite matching the paper we modeled after, is pretty solid!

One of the first conclusions we can draw from this graph is that our learning rate was a little high. We saw big improvements in the 5 epochs, and then no more progress - dividing the learning rate by two could have helped here and possibly improved our score if we had more training cycles.

We tried multiple permutations of differing-dimensional GLoVE vectors and LSTM hidden states (as shown in Table 1) before ultimately arriving at optimal 100-dimensional GLoVE vectors with 100-dimensional hidden states.

GLoVE	LSTM hidden state	Dropout	F1	EM
100	**100**	**0.2**	**0.65**	**0.53**
300	150	0.5	0.64	0.53
200	200	0.2	0.64	0.52
100	100	0	0.62	0.47
200	200	0	0.6	0.47
300	150	0	0.47	0.35
300	300	0	0.48	0.36

Table 1: Hyperparameter Dimensionality Search Results

4.1 Examples

We also can learn a lot by looking at examples of misclassifications we had. This can guide our future work, as well as suggest places our model can improve and some inherently hard parts of the problem

We seem to drop leading ”the” often. For instance:

Q: Who established the Theotokos Paregoritissa Church in 1294-96 ?
Our answer: Despot of Epirus
Label: the Despot of Epirus

Q: What college was influenced by Puritan beliefs ?
Our answer: University of Cambridge
Label: the University of Cambridge

However, we also note that there are times that we add an extraneous ”the”, such as:

Q: What is the UK definition of game is governed by ?
Our answer: the Game Act 1831
Label: Game Act 1831

For humans, whether or not to add ”the” also seems ambiguous, so this seems to be an inherently hard part of the problem. Fortunately, it doesn’t have too detrimental an effect on F1 score.

There also were answers that were just longer than 15 words, and for these answers, we tend to predict nonsense:

Q: Why did some landlords burn their own buildings ?
Our answer: low property-value buildings
Label: it was more lucrative to get insurance money than to refurbish or sell a building in a severely distressed area

Q: How are floor leaders kept informed of legislative status ?
Our answer: constantly
Label: contacts with his party ’s members serving on House committees , and with the members of the party ’s whip organization

We might improve these by relaxing that hyperparameter, or better, by learning that some types of questions (perhaps such as ”why” and ”how” questions) might have longer answers.

5 Future Direction

The first option that we were excited about was the idea of using something like match-LSTM with answer pointer as a candidate pruner for bilateral multiperspective matching (biMPM). biMPM seems to be a high potential model, performing very well against datasets such as WikiQA, which are multiple choice style datasets. Obviously, the ”multiple choice” in the SQuAD consists of all roughly $15 * |D|$ possible sets of answers, which is a little much to run through biMPM - picking a network architecture that assigns ”probabilities” to each answer is interesting as a way to narrow these choices down. For instance, we could only offer the top 5 candidates based on the probabilities that come out of match-LSTM with answer pointer, or we could only offer candidates with a cumulative probability of at least .01 to be the answer, or something of the sort.

Another possibility is attempting to learn answer length from the question. Obviously, we capped answer length with hyperparameter 15 - one can imagine that different types of questions (”who” questions vs. ”how” questions, for instance) might have very different max typical lengths. Rather than use the hyperparameter of 15 across the board, we would like to try learning the cap for answer length from the question as another prediction, probably via some LSTM.

References

[1] Wang, Shuohang, and Jing Jiang. ”Machine comprehension using match-lstm and answer pointer.” arXiv preprint arXiv:1608.07905 (2016).

[2] Xiong, Caiming, Victor Zhong, and Richard Socher. ”Dynamic Coattention Networks For Question Answering.” arXiv preprint arXiv:1611.01604 (2016).

[3] Wang, Zhiguo, Wael Hamza, and Radu Florian. ”Bilateral Multi-Perspective Matching for Natural Language Sentences.” arXiv preprint arXiv:1702.03814 (2017).