Match LSTM based Question Answering Codalab Username: yyu10

Hershed Tilak
Department of Electrical Engineering
Stanford University
htilak@stanford.edu

Yangyang Yu
Department of Electrical Engineering
Stanford University
yyu10@stanford.edu

Michael Lowney
Department of Electrical Engineering
Stanford University
mlowney@stanford.edu

Abstract

Machine based reading comprehension is an interesting problem in the field of natural language processing. In this project, an end to end neural architecture for this task is implemented based on the Match LSTM and Answer Pointer model described by Wang and Jiang [10]. The system is evaluated on the Stanford Question Answering Dataset. The results demonstrate that the system performs reasonably well.

1 Introduction

Deep learning has been able to solve some of the most advanced machine learning problems, from object recognition, to machine translation. One of the major advantages of deep learning is that we no longer have to hand craft features for each problem, but instead can learn how to apply a series of non linear functions to our data in order to minimize our error. These data driven approaches have led to many breakthroughs in the fields of artificial intelligence and language understanding. In this paper we highlight how these deep learning approaches can be applied to the task of reading comprehension.

Machine based reading comprehension has many practical applications, from automatic text summarization to information extraction to question answering. The Stanford Question Answering Dataset (SQuAD) [7] is a reading comprehension dataset with over one hundred thousand question answer pairs, sourced from over five hundred Wikipedia articles. The question-answer collections are presented in the format of a context paragraph, a related question, and an answer constructed by selecting part of the context paragraph. The question-answer pairs in the dataset were created by human crowdworkers, making the dataset a good candidate for evaluating machine based reading comprehension models. The goal of the system implemented in this paper to highlight the text in a paragraph that answers a given question.

Our system is based heavily on the one developed by Wang and Jiang [10]. Our contributions to their original system include using a bidirectional LSTM cell when encoding the question, applying exponential masking in the loss function, and providing an implementation for our system in Tensorflow. Figure 2 shows a comparison between the performance of our implementation of the original system versus our implementation of the system with our contributions.

2 Related Work

Many traditional approaches to machine based question answering typically involve rule based classifiers [3] or manual feature engineering [1], making use of semantic and syntactic features associated with subsections of the text. Typically such approaches involved a deeper understanding of linguistics, and are unable to utilize information from features other than those chosen by the system designer. Due to these limitations, many modern approaches have focused on utilizing deep learning architectures.

Early deep learning architectures could only be used to a small set of problems with a fixed input size and fixed target size. While these models were powerful and accurate, they were not flexible enough to be applied to the problem of machine comprehension. In 2014, Sutskever et. al proposed a sequence to sequence model that helped overcome these limits [9]. Their model uses a multilayer Long Short-Term Memory (LSTM) units to map, or encode, the input to a fixed size vector. An additional LSTM is used to decode the fixed hidden vector to the final output target. The sequence to sequence model is the backbone of many machine translation and machine comprehension techniques.

3 Method

In this section, we present all the major components in our end-to-end architecture. As mentioned in Introduction, our system takes a context paragraph and a related question as inputs and outputs part of the context paragraph as the answer to the question.

3.1 Data Preprocessing

To represent the words in the input context paragraphs and questions, we used the word embeddings initialized by the GloVe word vectors[5]. GloVe is an unsupervised learning algorithm for obtaining vector representation for words. The word vectors we chose are trained on 6B word corpora from Wikipedia and Gigaword which overlaps with the source of the SQuAD database. The dimensionality of the embeddings is a parameter that implies accuracy and process time trade-off. While larger dimensionality of the embeddings might provide more information about each word, it makes the inputs to our neural network much lager and thus significantly increases the number of parameters in the model.

To be able to batch variable-length paragraphs and questions, we zero-padded all the paragraphs to the same max paragraph length and all the questions to the same max question lengths. The length of each paragraph or question is also passed to the neural networks, so that the paddings can be ignored when necessary.

3.2 Match LSTM and Answer Pointer Network Model

Our model is based heavily on the Mach LSTM and Answer Pointer Network Model proposed by Wang and Jiang [10]. At a high level, the model consists of three major sub components. First, the question and paragraph are passed through separate LSTM cells to incorporate contextual information in their encodings. Next, the encoded question and paragraph are fed to a Match LSTM layer, which uses a word to word attention mechanism to associate each word in the context paragraph with its relavence to the question being asked. Finally an Answer Pointer layer takes the output of the Match LSTM layer and uses an attention mechanism to determine the probability of each word in the paragraph being the start index or the end index of the answer. These three sub components are covered in more detail below.

3.2.1 LSTM Preprocessing Layer

The purpose of the LSTM Preprocessing layer is to incorporate contextual information into our encodings of the question and paragraph. We use separate LSTM cells for the question and the paragraph. We choose to do this because words can sometimes have different meanings depending on whether they appear in the context of a question or a paragraph, and we wanted to ensure that our encodings would be able to reflect this. For the paragraph we use a single directional LSTM cell because the directional context of words in the paragraph will later be accounted for in the Match LSTM layer. However, such is not the case for the words in the question, so we decided to use a bidirectional LSTM to encode contextual information for each word in the question in each direction, and concatenated the two directional enconding vectors for each word. The outputs of this layer are are

$$H^p = \overrightarrow{\text{LSTM}}(P) \qquad\qquad H^q = \overleftrightarrow{\text{LSTM}}(Q) \tag{1}$$

Here, $H^p \in \mathbb{R}^{l \times p}$ and $H^q \in \mathbb{R}^{2l \times q}$, where l is the size of the hidden state of the LSTM, p is the length of the paragraph, and q is the length of the question.

3.2.2 Match LSTM Layer

The Match LSTM Layer goes through each word in the paragraph sequentially and computes an attention vector that measures the similarity between it and each word in the question. The attention vector is computed as follows:

$$\overrightarrow{G}_i = \tanh(W^q H^q + (W^p h_i^p + W^r \overrightarrow{h}_{i-1}^r + b^p) \otimes e_q) \tag{2}$$

$$\overrightarrow{\alpha}_i = \text{softmax}(w^T \overrightarrow{G}_i + b \otimes e_q) \tag{3}$$

where $W^q \in \mathbb{R}^{l \times 2l}$, $W^p, W^r \in \mathbb{R}^{l \times l}$, $b^p, w \in \mathbb{R}^l$, and $b \in \mathbb{R}$ are parameters to be learned and $\overrightarrow{G}_i \in \mathbb{R}^{l \times q}$, $\overrightarrow{\alpha}_i \in \mathbb{R}^q$. The $\otimes e_q$ operator indicates that the argument to the left is tiled vertically q times in order to make the dimensions line up correctly. The attention vector $\overrightarrow{\alpha}_i$ is then used to create a weighted representation of the question $H^q \overrightarrow{\alpha}_i^T$, and this weighted representation is concatenated with the initial encoding of the word in the paragraph to create a new vector $\overrightarrow{z}_i$.

$$\overrightarrow{z}_i = \begin{bmatrix} h_i^p \\ H^q \overrightarrow{\alpha}_i^T \end{bmatrix} \tag{4}$$

The vector $\overrightarrow{z}_i \in \mathbb{R}^{3l}$ is then fed through an LSTM conditioned on the previous state of the Match LSTM to generate the next state of the Match LSTM

$$\overrightarrow{h}_i^r = \overrightarrow{\text{LSTM}}(\overrightarrow{z}_i, \overrightarrow{h}_{i-1}^r) \tag{5}$$

where $\overrightarrow{h}_i^r \in \mathbb{R}^l$. The output of the forward Match LSTM is then $\overrightarrow{H}^r = [\overrightarrow{h}_1^r, ..., \overrightarrow{h}_p^r] \in \mathbb{R}^{l \times p}$. We then run a Match LSTM in the backwards direction by reversing the input and feeding it through the same Match LSTM, but using a different LSTM cell when calculating the state vectors $\overleftarrow{h}_i^r$. We use a different LSTM because the context captured in each encoding is dependent on the direction that the paragraph is traversed, and to capture these different contexts we need separate LSTM cells. All of the other learned parameters are reused. After this step, we reverse the output of the Match LSTM to get the matrix $\overleftarrow{H}^r = [\overleftarrow{h}_1^r, ..., \overleftarrow{h}_p^r] \in \mathbb{R}^{l \times p}$. Finally we stack these two matrices vertically to get

$$H^r = \begin{bmatrix} \overrightarrow{H}^r \\ \overleftarrow{H}^r \end{bmatrix} \tag{6}$$

where $H^r \in \mathbb{R}^{2l \times p}$. The reason we stack our two directional H^r vectors is so that the directional dependence can be encompassed in the knowledge representation that will be passed to the Answer Pointer Layer.

3.2.3 Answer Pointer Layer

Finally, the Answer Pointer layer takes in all the information that was encoded in the previous steps. Our implementation uses the boundary model, in which the final output is two probability distributions, $\beta_1, \beta_2 \in \mathbb{R}^p$, where p is the length of the paragraph fed to the system, β_1 is the probability distribution for the start index, and β_2 is the probability distribution for the end index. An attention method is used again on the hidden states H^r from the Match LSTM layer.

$$\mathbf{F}_k = \tanh(\mathbf{V}\mathbf{H}^r + (\mathbf{W}^a\mathbf{h}^a_{k-1} + \mathbf{b}^a) \otimes \mathbf{e}_p) \tag{7}$$

$$\beta_k = \text{softmax}(\mathbf{v}^T\mathbf{F}_k + c \otimes \mathbf{e}_p) \tag{8}$$

$$\mathbf{h}^a_k = \overrightarrow{\text{LSTM}}(\mathbf{H}^r\beta_k^T, \mathbf{h}^a_{k-1}) \tag{9}$$

Where $\mathbf{V} \in \mathbb{R}^{lx2l}$, $\mathbf{W}^a \in \mathbb{R}^{lxl}$, $\mathbf{b}^a, \mathbf{v} \in \mathbb{R}^l$, are all trainable variables, and $\otimes\mathbf{e}_p$ is used as defined in the Match LSTM layer. In this case k only takes on two values, one for the start index, and one for the end index. The index with the greatest probability is chosen as the start of end of the answer.

3.3 Loss Function

We used cross-entropy loss since we modeled the problem as a discrete classification task in which the classes are mutually exclusive and the neural network models we used generate the probability distribution of the start index and the end index. Since we initially pad all of our paragraphs before processing them, our model will output nonzero probabilities for indices beyond the original paragraph length. To prevent such predictions from contributing to our loss, we apply an exponential masking function which sets the probabilites of choosing indices greater than the length of the context paragraph to be negligible.

3.4 Dropout

Srivastava et al. [4] presented dropout as an effective way to reduce overfitting in deep neural networks. By randomly dropping some connections in the network during training, Srivastava et al. reported significant improvement in test accuracy in various deep learning tasks. Therefore we also experimented with dropout in our model. We applied dropout to the outputs of the LSTM cells in the preprocessing layer and the output of LSTM cells in the Match LSTM layer.

4 Results and Analysis

4.1 Hyperparameters

Due to time constraints, we were only able to train our model for 8 epochs. We used a learning rate of 0.001 and applied a learning rate decay factor of 0.9 every half epoch. We used a hidden state size of $l = 150$ and passed training data through our system in 64 sample batches. For our word level encodings, we used length 100 GLoVE vectors. To prevent our model from overfitting the training data, we applied dropout to the matrices H^p, H^q, and H^r, using a keep probability of 0.6. We initialized all of our trainable matrices using the Xavier initializer [6] and all trainable vectors and scalars using the Tensorflow uniform unit scaling initializer.

4.2 Results

Our system trained for 8 epochs, and the learning curve can be seen in Figure 3. The final loss converges to 3.23. The two other metrics we used to evaluate our model were the F1 score and the Exact Match (EM) score. The F1 score shows the average overlap between prediction and ground truth by calculating the harmonic mean of the precision and recall. The EM score shows the percentage of predictions that exactly matched the ground truth. The final F1 score for the train set was 69.023, and the final EM score on the train set was 54. A full list of results can be seen in Table 1.

Table 1: F1 & EM Scores

	F1	EM
Train	69.17	53.667
Dev	61.965	50.539
Test	62.679	51.883

The first few implementations of our model did not have any form of dropout or regularization. This was done to ensure that the core functionality of our model was working. We could tell that our model was learning if it started to overfit the training data. In Figure 1, the results of training our model with and without dropout are shown side by side. The data is from earlier runs of our model, before hyperparameter tuning, and as such the F1 and EM scores are not as high as for our final model. In the case without dropout the F1 score for the validation set is much lower than the training score. When dropout is used we are able to generalize to the unseen validation set more efficiently. One advantage of using dropout over L2 regularization is that it is easy to reason about how the keep probability effects training. In L2 regularization we may have to tune our hyperparameter every time we expand our model or add more variables, but in dropout we can keep applying dropout layers with the same dropout rate hyperparameter.

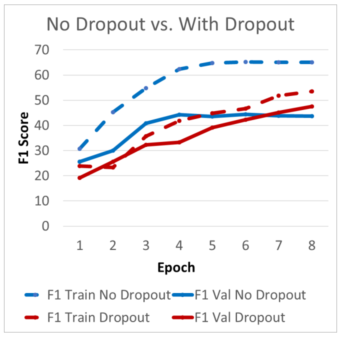

Figure 1: Comparison of F1 scores for train set and validation with and with out dropout.

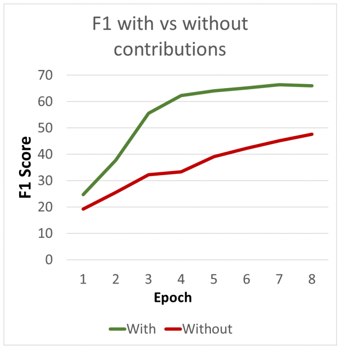

Figure 2: Comparison of F1 scores on validation set for our model implementation with and without our contributions.

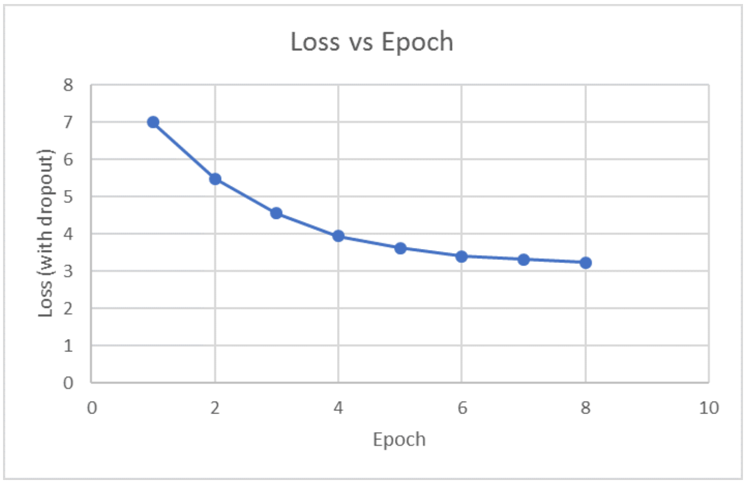

Figure 3: Training loss vs epoch

4.2.1 Attention

Figure 4 shows the Match LSTM attention vectors generated for part of the context in relation to Question 5 from Table 2. On the left side of the figure are the context words; On the top of the figure are the question words. The weights clearly show correlation between the question and context words. For example, exact same word pairs like '(during, during)' and '(time, time)' result in very high attention weights. Also words that are often associated like '(year, during)', '(year, time)' and '(year, period)' have relatively high attention weights. Overall we can see that this attention vector performed well in the example by paying more attention to the question at context words that are related to time, which is what the question asked about.

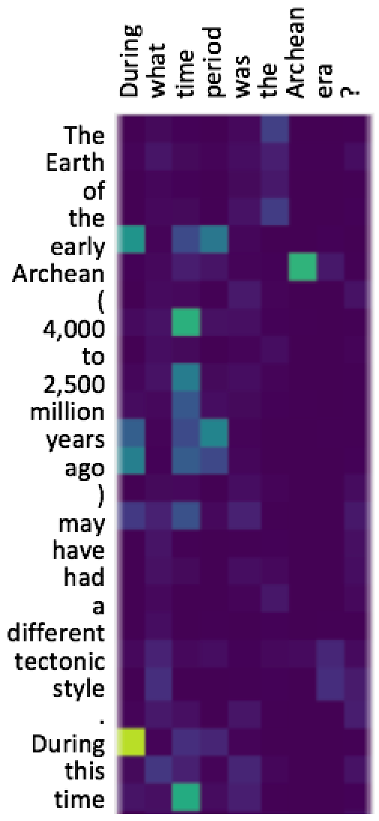

Figure 4: Visualization of subset of α vectors in Match LSTM for Question 5 in Table 2

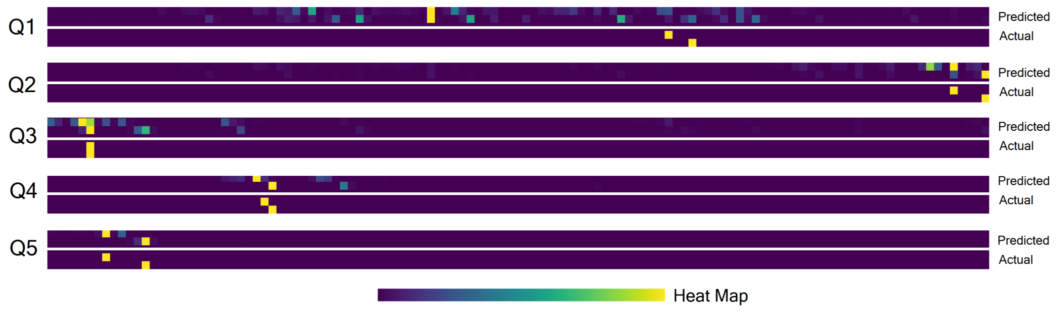

Figure 5: Predicted start and end indices for various questions. For a given question, the first row is the predicted start index, the second row is the predicted end index, the third row is the actual start index, and the fourth row is the actual end index. The context paragraph and questions are provided in Table 2

Table 2: Sample Context, Question, and Ground Truth Answer Data for Analysis

Context	The Earth of the early Archean (4,000 to 2,500 million years ago) may have had a different tectonic style. During this time , the Earth 's crust cooled enough that rocks and continental plates began to form. Some scientists think because the Earth was hotter, that plate tectonic activity was more vigorous than it is today, resulting in a much greater rate of recycling of crustal material. This may have prevented cratonisation and continent formation until the mantle cooled and convection slowed down. Others argue that the subcontinental lithospheric mantle is too buoyant to subduct and that the lack of Archean rocks is a function of erosion and subsequent tectonic events.
Q1	What might have a very hot earth stopped from occurring?
A1	cratonisation and continent formation
Q2	What do some believe accounts for the small amount of Archean rocks?
A2	erosion and subsequent tectonic events
Q3	During what period did the earths crust cooling allow the creation of plates?
A3	Archean
Q4	It is believed that a very warm earth would lead to more recycling of what?
A4	crustal material
Q5	During what time period was the Archean era?
A5	4,000 to 2,500 million years ago

4.2.2 Start and End Index Prediction

Figure 5 shows the predicted, actual start, and end indices for the answers to five questions, all sharing the same context paragraph. The context and questions are listed in Table 2 for reference. For Question 1, the model has a number of potential candidates for the start and end indices, including the actual correct answer. However, it is most confident in choosing indices that correspond to the answer “hotter”, which is incorrect. We believe the confusion is partly due to the strange wording of the question. Even as a human reader, it is difficult to identify the correct answer from the context. For Question 2, our model predicts the correct response of “erosion and subsequent tectonic events”. The second most probable answer according to our model weights would have been ”a function of erosion and subsequent tectonic events”, which is also a very similar and reasonable answer. For Question 3, our model predicts that the answer is “early Archean”, with the second most likely answer being the correct answer of ”Archean”. Once again, the answer predicted by our model is very reasonable, even though it is incorrect. For Question 4, our model predicts ”of crustal material” with a pretty high confidence when the correct answer is just “crustal material”. For Question 5, our model correctly predicts the correct indices and does so with a high confidence. The trend in these five samples is that our model does well at choosing the correct end indices of questions. Also, when it incorrectly predicts the start indices, it is usually pretty close to the correct answer, suggesting that it may be capable of predicting them correctly with some more fine tuning of our model. Finally, our model has difficulty with some questions, possibly due to strange wording.

5 Conclusion and Future Work

Overall, our system was able to achieve decent F1 and EM scores on the Stanford SQuAD test dataset. The scores illustrate that our model is able to do reasonably well at the specified task. Looking forward, the main extension we would like to make to the model is to utilize character level embeddings. This would provide our model with a mechanism for dealing with out of vocabulary words, allowing it to generalize better to unseen texts. Other possible extensions include implementing the search algorithm on top of the Match LSTM model as described by Wang and Jiang [10] or implementing a secondary model, such as the Bidirecional Attention Flow model proposed by Seo et al. [8] and using ensembling to combine what it learns with our current model. Finally, we would like to spend some more time performing hyperparameter tuning to see how well our current model can perform as is.

6 Contributions

The work was split evenly and we all kind of worked on everything. There were a ton of group debugging sessions.

References

[1] Hai Wang, Mohit Bansal, Kevin Gimpel, and David McAllester. Machine comprehension with syntax, frames, and semantics. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers), pp. 700706. Association for Computational Linguistics, 2015.

[2] Hermann, Karl Moritz, Kocisky, Tomas, Grefenstette, Edward, Espeholt, Lasse, Kay, Will, Suleyman, Mustafa, and Blunsom, Phil. Teaching machines to read and comprehend. In Advances in Neural Information Processing Systems (NIPS), 2015. URL http://arxiv.org/abs/1506.03340.

[3] Matthew Richardson, Christopher JC Burges, and Erin Renshaw. Mctest: A challenge dataset for the open-domain machine comprehension of text. In EMNLP, volume 3, pp. 4, 2013.

[4] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. JMLR, 2014.

[5] Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global Vectors for Word Representation." EMNLP. Vol. 14. 2014.

[6] Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." Aistats. Vol. 9. 2010.

[7] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, 2016.

[8] Seo, Minjoon, et al. "Bidirectional Attention Flow for Machine Comprehension." arXiv preprint arXiv:1611.01603 (2016).

[9] Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." Advances in neural information processing systems. 2014.

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