Question Answering Using Match-LSTM and Answer Pointer

Annie Hu, Cindy Wang, and Brandon Yang
{anniehu, ciwang, bcyang}@stanford.edu
CodaLab: anniehu
March 21, 2016

1 Introduction

Machine comprehension of text is a significant problem in natural language processing today – in this project, we tackle machine reading comprehension as applied to question answering. Our goal is: given a question and a context paragraph, to extract from the paragraph the answer to the question.

As an oracle, on the dataset we used, humans score over 86.8% accuracy (EM) on the test set for this task, while the best models only achieve roughly 75%. Existing approaches to this extractive Question Answering problem typically involve an encoding layer that encodes the question and paragraph into a sequence, some additional layer that accounts for interaction between the question and paragraph, and a final decoding layer that extracts the answer from the paragraph [2][3][4][7]. In this paper, we will follow a similar structure, using LSTMs in our encoding and decoding layers, and calculating attention as our interaction layer.

2 Dataset

The dataset used is the recently released Stanford Question Answering Dataset (SQuAD)[1]. The context paragraphs are extracted from Wikipedia, while questions and answers are human-generated. It consists of around 100K <question, paragraph, answer> triples. For our models, we will represent each question and paragraph as a list of IDs corresponding to each word, then use GloVe embeddings to represent each word as a word vector, and each answer as a span (start, end), marking its start and end indices in the context paragraph. We use a pre-split train set to train our models, and the leftover validation set to tune our parameters.

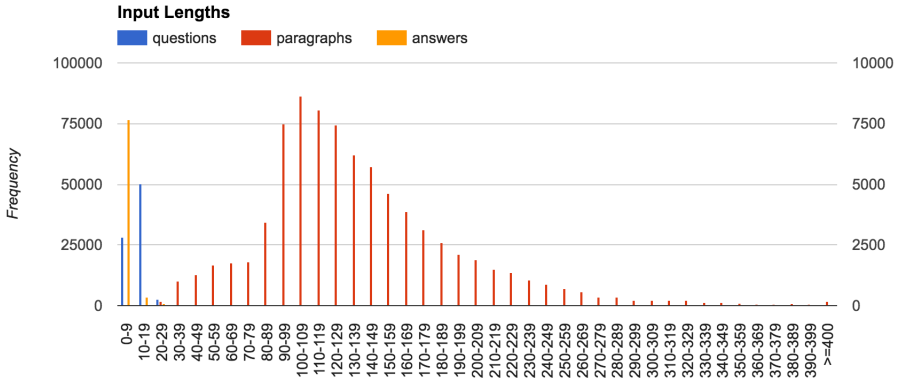

Figure 1: Histogram of Question, Paragraph, and Answer lengths in train dataset

From an initial histogram of our training data, we determined that the large majority of paragraph lengths were below 300 words, and all question lengths were below 70 words. Thus we simply excluded input paragraphs longer than 300 words in our training and validating steps.

3 Approach / Model Formulation

For this task, we implemented two main models. We started with a baseline of a sequence attention mix model. We then implemented Match-LSTM with answer pointer, adapted from Wang & Jiang, 2016 [3]. Finally, we improved upon the original Match-LSTM model by exploring the use of BiLSTMs in the preprocessing layer and different amounts of dropout regularization and hyperparameter tuning.

3.1 Baseline: Sequence Attention Mix

For our baseline, we implemented a three-layer model that utilizes the coattention mechanism described in [6] to encode the question vectors. We give a detailed description of our method below.

BiLSTM Preprocessing Layer
The BiLSTM preprocessing layer incorporates contextual information into the representation of each token in the question and the paragraph. We use a standard bidirectional LSTM to process the question and the paragraph separately, as shown below.

$$\mathbf{H}^p = LSTM(\mathbf{P}) \text{ and } \mathbf{H}^q = LSTM(\mathbf{Q})$$

Coattention Layer
This layer incorporates a coattention mechanism that simultaneously attends to the question and the paragraph. We first compute the affinity matrix, which contains affinity scores corresponding to all pairs of question and paragraph words. This is normalized to produce the attention weights A across the paragraph for each word in the question.

$$A = softmax(\mathbf{P} \cdot \mathbf{Q}^T)$$

Using the affinity matrix, we then compute attention contexts of the question in light of each word of the paragraph.

$$C^P = A \cdot \mathbf{Q}$$

We mix the attention context with $\mathbf{P}$ using a linear layer to attain the final encoded representation.

$$\mathbf{P}_{mix} = [C^P; \mathbf{P}]W + b$$

Dynamic Decoding Layer
The decoding layer uses the output from the first two time steps of an LSTM to predict the probability distributions for the start and end indices of the answer, respectively.

$$a^s = \mathbf{P}_{mix} W_1$$

$$a^e = LSTM(\mathbf{P}_{mix}) W_1,$$

3.2 Match-LSTM with Answer Pointer

Iterating upon our baseline, we implemented the match-LSTM with answer pointer model from Wang & Jiang, 2016 [3]. This model incorporates a more complex coattention mechanism, as well as a novel decoder. The Pointer Network model used for decoding is optimal for our task, since it is built for the specific task of generating an output sequence with elements that are discrete tokens corresponding to positions in the input sequence.

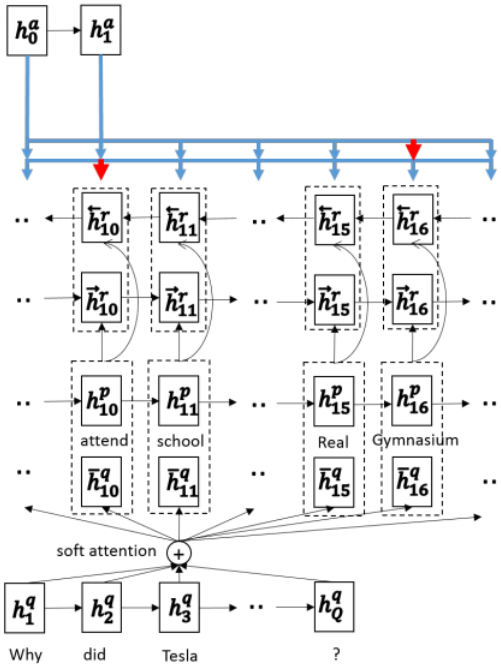

Figure 2: Diagram of Match-LSTM model layers

LSTM Preprocessing Layer
The preprocessing layer is the same as the one in the baseline above, except we use a standard one-directional LSTM rather than a bidirectional LSTM.

$$\mathbf{H}^p = LSTM(\mathbf{P}) \text{ and } \mathbf{H}^q = LSTM(\mathbf{Q})$$

Match-LSTM Layer
The purpose of the Match-LSTM layer is to encode the paragraph vectors with the backward and forward paragraph contexts as well as attention weighted representations of the question. We use the same match-LSTM model proposed by [5] and applied in [3].

We build an LSTM cell that takes as input a token of the paragraph and does the following. First, we compute attention weight vectors corresponding to the degree of matching between the paragraph token and the question. We use this weight to obtain a weighted version of the question and combine it with the paragraph token to form a vector $\overrightarrow{z}_i$:

$$\begin{aligned}
\overrightarrow{\mathbf{G}}_i &= \tanh(\mathbf{W}^q\mathbf{H}^q + (\mathbf{W}^p\mathbf{h}_i^p + \mathbf{W}^r\overrightarrow{\mathbf{h}}_{i-1}^r + \mathbf{b}^p) \otimes \mathbf{e}_Q) \\
\overrightarrow{\alpha}_i &= \text{softmax}(\mathbf{w}^T\overrightarrow{\mathbf{G}}_i + b \otimes \mathbf{e}_Q), \\
\overrightarrow{\mathbf{z}}_i &= \begin{bmatrix} \mathbf{h}_i^p \\ \mathbf{H}^q\overrightarrow{\alpha}_i^T \end{bmatrix}
\end{aligned}$$

This LSTM cell is used to run a bidirectional LSTM for $|\mathbf{P}|$ time steps. We concatenate the forward and backward LSTM outputs to produce the final encoded representation:

$$\begin{aligned}
\overrightarrow{\mathbf{h}}_i^r &= \overrightarrow{LSTM}(\overrightarrow{\mathbf{z}}_i, \overrightarrow{\mathbf{h}}_{i-1}^r), \\
\mathbf{H}^r &= \begin{bmatrix} \overrightarrow{\mathbf{H}}^r \\ \overleftarrow{\mathbf{H}}^r \end{bmatrix}
\end{aligned}$$

Answer Pointer Decoding Layer
The answer pointer model adds significant complexity to the decoding layer. It uses an attention mechanism to generate output consisting of tokens from $\mathbf{P}$. We implement the boundary model as specified in [3], which predicts the indices within the paragraph of the start and end tokens of the answer.

We use the attention mechanism again to produce the attention vectors β_s and β_e. These are probability distributions where $\beta_{s,j}$ and $\beta_{e,j}$ are the probabilities of selecting token j of the paragraph as the start and end tokens of the answer, respectively. We calculate β by retrieving the outputs from the first two time steps of the following LSTM:

$$\begin{aligned} \mathbf{F}_k &= \tanh(\mathbf{V}\widetilde{\mathbf{H}}^{\mathrm{r}} + (\mathbf{W}^{\mathrm{a}}\mathbf{h}^{\mathrm{a}}_{k-1} + \mathbf{b}^{a}) \otimes \mathbf{e}_{(P+1)}) \\ \beta_k &= \mathrm{softmax}(\mathbf{v}^{\intercal}\mathbf{F}_k + c \otimes \mathbf{e}_{(P+1)}) \\ \mathbf{h}^{\mathrm{a}}_k &= \overrightarrow{LSTM}(\widetilde{\mathbf{H}}^{\mathrm{r}}\beta_k^{\intercal}, \mathbf{h}^{\mathrm{a}}_{k-1}) \end{aligned}$$

3.3 Model Extensions, Regularization, and Hyperparameter Tuning

The preprocessing layer in the Wang & Jiang, 2016 Match-LSTM model [3] encodes both the question and paragraph using a forwards LSTM to create $\mathbf{H}^p$ and $\mathbf{H}^q$ for the Match-LSTM layer. We extend the pre-processing layer for both question and paragraph to encode hidden states in both directions with a bi-directional LSTM and concatenate the hidden states: $\mathbf{H}_q = \begin{bmatrix} \overleftarrow{\mathbf{H}_q} \\ \overrightarrow{\mathbf{H}_q} \end{bmatrix}$ and $\mathbf{H}_p = \begin{bmatrix} \overleftarrow{\mathbf{H}_p} \\ \overrightarrow{\mathbf{H}_p} \end{bmatrix}$ before passing into the Match-LSTM model. This allows the pre-processed question and paragraph representations to capture information from both directions.

Our original Match-LSTM implementation had no regularization and suffered from overfitting, as our performance dropped from 81% F1 and 66% EM on the train set to 61% F1 and 51% EM on the validation set. To address this problem, we considered adding Dropout and L2-regularization to our model. Based on our experimentation and existing literature on machine comprehension, we chose to apply a Dropout of 0.2 to all LSTM layers in the encoder with no L2-regularization to regularize our model.

We then tuned our hyperparameters to achieve better performance. Our original implementations had been trained with the Adam optimizer, and we explore the use of the Adamax optimizer. We further experimented with varying batch sizes, hidden state sizes, and gradient clipping values and trained our final model with batch size 30, state size 100, and a gradient clipping value of 5.

3.4 Evaluation

We mainly used two metrics to evaluate our model: F1 and EM.
F1 roughly measures the average overlap (over all questions) between the prediction and ground truth answer. We treat the prediction and ground truth as bags of tokens for these calculations. F1 is the harmonic mean of precision and recall, which can be calculated as

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

EM (exact match) measures the percentage of predictions that match the ground truth answer exactly.

4 Results

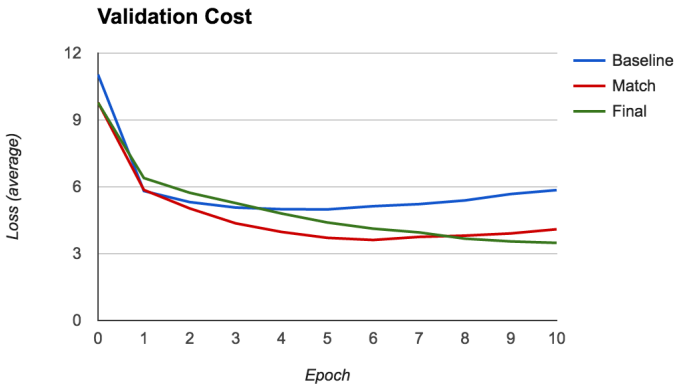

Figure 3: Comparison of model validation cost curves

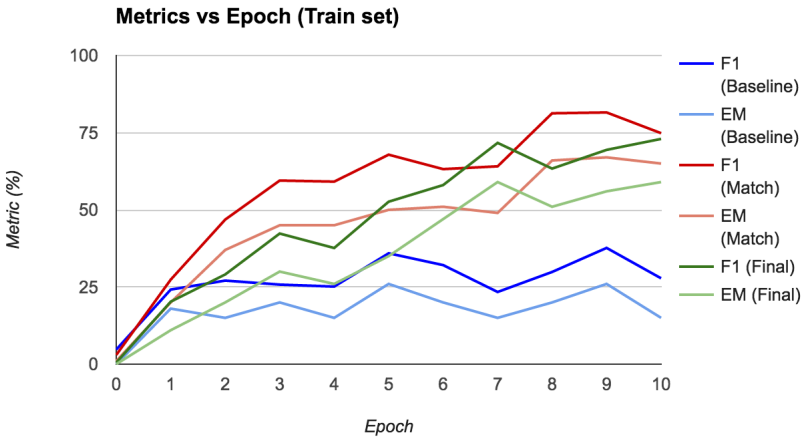

Figure 4: Comparison of model results on train set

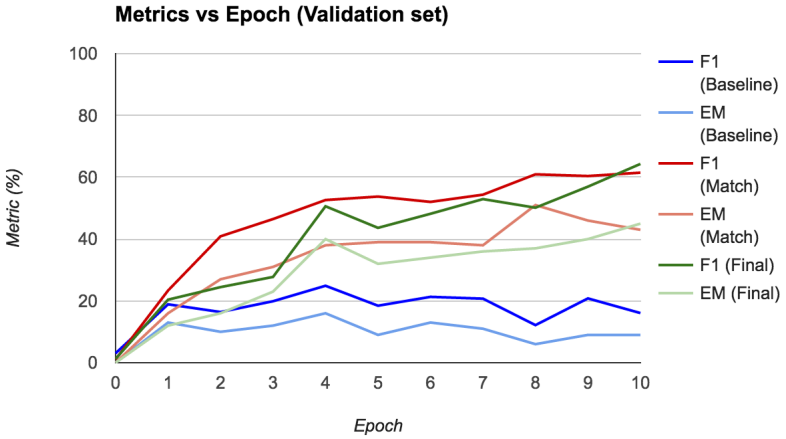

Figure 5: Comparison of model results on validation set

Model	F1	EM
Baseline (validation set)	24.9	16
Match-LSTM (train set)	81.3	66
Match-LSTM (validation set)	61.0	51
Match-LSTM (dev set)	53.7	40
Match-LSTM (test set)	54.0	41
Final Match-LSTM (train set)	73.0	59
Final Match-LSTM (validation set)	64.3	45
Final Match-LSTM (dev set)	54.0	40
Final Match-LSTM (test set)	54.6	42

Table 1: Comparison of model best performances

4.1 Model Analysis

Our implementation of the Match-LSTM model greatly out-performed our attention-mix baseline. We attribute this largely due to the more powerful attention model in Match-LSTM, by sequentially aggregating backward and forward paragraph contexts with attention weighted representations of the question and Answer Pointer layer. Our Final Match-LSTM model, using Bi-LSTMs for the paragraph and question preprocess encodings, regularization, hyperparameter tuning, and the ADAMAX optimizer outperformed our initial implementation by 1% F1 and 1% EM on the test set. However, as shown in Figure 6, the F1 and EM scores of our final Match-LSTM model are still increasing with every epoch while those of our original implementation peaked at Epoch 7, indicating that our final Match-LSTM scores could be further improved with more training time. Using Bi-LSTMs for the paragraph and question preprocess encodings may not have greatly impacted performance, since the Match-LSTM layer already encodes the paragraph contexts with forward and backward LSTMs and uses the entire weighted representation of the question in the attention calculations. Adding dropout helped significantly with overfitting, particularly with the F1 score: there was a drop of 20% F1 between training and validation for our original implementation, but only a drop of 8% F1 between training and validation. This contributed to higher validation and test scores, and further increasing the amount of dropout could improve our score on the test set. All of our models saw a significant drop in performance between the validation set scores and the test set scores, of 8 - 10 % F1 and 5 - 10 % EM. Our embeddings have been trimmed to those in the training set and validation set, which resulted in a lot of unknown tokens in the dev set and test set, and this could have been a major source of error on the dev and test sets.

4.2 Error Analysis

Our model was able to successfully predict some surprisingly complex answers:
Paragraph: While experimenting, Tesla inadvertently faulted a power station generator, causing a power outage...which caused heavy sparks to jump through the windings and destroy the insulation!" (parts omitted here to save space)
Question (1): What did the sparks do to the insulation?
Answer: jump through the windings and destroy the insulation

Figure 6 shows our model's fitted probabilities of each paragraph index as a start answer (blue points) or an end answer (green points). The red and black vertical lines indicate the final predicted start and end indices, respectively. We can see that the model is much more confident in its results for the simple Question (2), as the final predicted indices have much higher probabilities than the rest, than the more complex Question (1), where many points are close to the final indices.

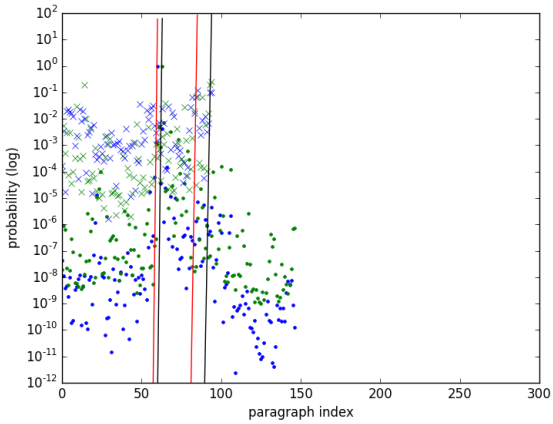

Figure 6: Probability distribution for Questions (1) and (2)

Unsurprisingly, the vast majority of our model's correct answers were simple questions with simple answers, such as the following:
Paragraph: ...The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion Carolina Panthers 24-10 to earn their third Super Bowl title. The game was played on February 7, 2016, at Levi's Stadium...
Question (2): What day was the game played on?
Answer: February 7, 2016
Question (3): What team was the NFC champion?
Answer: Carolina Panthers

Interestingly, using the same paragraph with a slightly different question gave a wrong result:
Question (4): What team was the AFC champion?
Answer: Carolina Panthers
True Answer: Denver Broncos

A possible explanation for this is that, when running our initial LSTM to calculate the question encoding, we place heavier emphasis on later tokens in the question. In this case, this emphasizes "champion" over the specific conference title (AFC or NFC), so we again predict "Carolina Panthers". These types of errors happened fairly often, suggesting that our final model could be improved by using a more advanced coattention mechanism.

Two other common errors we ran into were 1) seeing or predicting words that were not in our training or validation datasets, and 2) predicting a start index greater than our end index:
Question (5): Who was the first quarterback to take two teams to more than one Super Bowl?
Answer: <unk> <unk>
True Answer: Peyton Manning
Since neither of the tokens "Peyton" or "Manning" had been seen previously, our model predicted <unknown> for both. The first error should be fixed by training on the full GLoVE vectors, instead of the trimmed vocabulary. The second error we could fix by forcing the model to choose the next-best index until we have a valid end index.

Our exact-match metric was also not very forgiving of small differences in output. For example, we counted as wrong:
Question (6): Whose works helped Tesla recover from illness?
Answer: Mark Twain's
True Answer: Mark Twain
We could relax our evaluation metric to account for this.

Finally, there were some errors where our predicted answer was technically different from

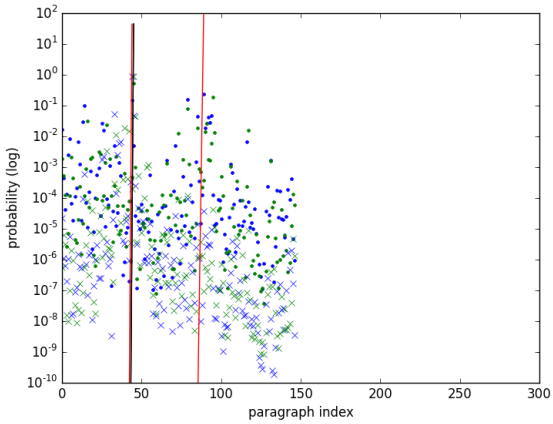

Figure 7: Probability distribution for Question (4)-dots and invalid indices-crosses. Note the incorrect start index (red line) to the right of its corresponding end index (black line), and that the maximum probability indices have much closer neighbors than in our correct predictions.

the true answer, but in meaning almost the same:
Paragraph: ...Controlled, experimental studies exploring intrinsic motivation of college students has shown that nonverbal expressions of enthusiasm, such as demonstrative gesturing, dramatic movements which are varied, and emotional facial expressions, result in college students reporting higher levels of intrinsic motivation to learn...
Question (7): What is dramatic gesturing an example of?
Answer: emotional facial expressions
True Answer: nonverbal expressions of enthusiasm

These two responses seem very close, even to a human reader – these errors are very exciting indicators of comprehension, but also show the difficulty of building a more nuanced model to capture these differences.

5 Conclusion and Future Work

Overall, our implementation of the adapted Match-LSTM model attains 54.6 % F1 and 42% EM scores on the test set, improving upon the Logistic Regression baseline [8]. The original Wang & Jiang, 2016 Match-LSTM model [3] attains 73.7 % F1 and 64.7 % EM on the test set, which indicates that our Match-LSTM implementation could be improved through more extensive hyperparameter tuning, longer training time, using full word embeddings, and additional investigation of our model and preprocessing. Moreover, we could further improve our test scores through implementing multiple different models and ensembling. This Match-LSTM model has been effective for textual entailment and machine comprehension, and we are interested in investigating other applications of the model. In addition, we would like to investigate other models that out-perform Match-LSTM, like the BiDAF model [7] and its memory-less attention mechanism, and how these insights can be applied to new machince comprehension models and data sets.

References

[1] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. CoRR, abs/1606.05250, 2016.

[2] Dirk Weissenborn, Georg Wiese, and Laura Seiffe. FastQA: A Simple and Efficient Neural Architecture for Question Answering. arXiv preprint arXiv:1703.04816, 2017.

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

[4] Zhiguo Wang, Wael Hamza, and Radu Florian. Bilateral multi-perspective matching for natural language sentences. arXiv preprint arXiv:1702.03814, 2017.

[5] Shuohang Wang and Jing Jiang. Learning natural language inference with LSTM. In Proceedings of the Conference on the North American Chapter of the Association for Computational Linguistics, 2016.

[6] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. arXiv preprint arXiv:1611.01604, 2016.

[7] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. arXiv preprint arXiv:1611.01603, 2016.

[8] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In ICCV, 2015.