An Approach to a Machine Comprehension Problem

Paritosh Mittal

1 Introduction and Problem Statement

The objective of the Machine Comprehension Problem is to develop a model which can answer natural-language questions from a natural-language context. We work with a provided, **closed** dataset (i.e. we are able to make the assumption that the answer always lies in the context.) So the problem of finding the answer is simplified to finding the start and end index of the appropriate answer in the context, (given the question).

Formally: let $c = \{c_1, c_2, ..., c_n\}$ be the n words in the comprehension, while $q = \{q_1, q_2, ..., q_m\}$ are the m words in the question. Our objective in this project is to develop a model or a function $f(q, c)$, which can return $(start, end)$, where $1 \leq start \leq end \leq n$, and the substring of the context from position $start$ to position end, i.e. $c_{start}...c_{end}$, is the answer to the question.

2 Background and Related Work

Machine Comprehension is a high-impact field of current research, due to its applications in building voice assistants such as Cortana. For instance, Wang et al [5] have demonstrated several different methods of combining question and context, to achieve learning objectives. For this particular project, our focus is on Attention Mechanisms [9][10][11].

Attention Mechanisms in Neural Networks are (very) loosely based on the attention mechanism found in humans. The idea is to focus on a certain region of an image/text with "high resolution" while perceiving the surrounding image in "low resolution", and then adjust the focal point over time. When working with text, the source sentence is encoded into a vector using a Recurrent Neural Network, and then decoded. The weakness of this approach is that the entire information is stored in single final vector – which is hard to store, even for LSTM, when a sentence can have 40 or more words. Other approaches, such as encoding in reverse direction encoding (as done in the Seq2Seq mechanism) or feeding the same sentence twice [12], have been shown to improve performance. But inherently, decoding information for long sentences is still a challenge.

In this context, it is important to note that Bengio et al [13] have recently proposed a new mechanism for attention, which no longer tries to encode the full source sentence into a fixed-length vector. Rather, the decoder is allowed to "attend" to different parts of the source sentence at each step of the output generation. The model learns what to attend to based on the input **and** what it has produced so far. Seo et al [1] also propose a bi-directional attention mechanism, where attention can be considered to "flow" from question to context and vice versa. We make use of all these ideas in attacking the current problem.

A few other research papers, which were of help in this project, are the work of Socher [2] (which proposed an out of word embedding, replacing word embedding with character embedding; this allows encoding to be generated for words that are not seen in the training data), and Yu et al [6] (new encodings that integrate start and end predictions). In particular, we gratefully acknowledge the help provided by the Stanford course CS 224D [7], taught by Richard Socher; in addition to helpful tutorial slides, they provide the skeleton code we used as a starting point (and to which we added our own layers, for bidirectional and smart selection, etc.)

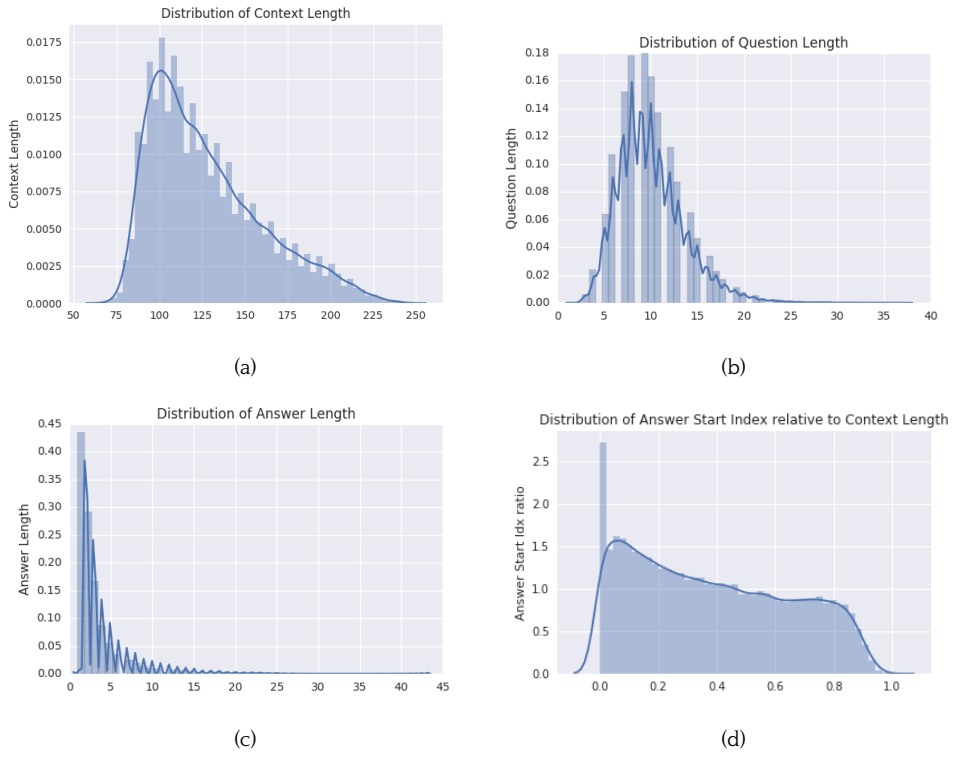

Figure 1: Plots for training split

3 Approach

The previous section mentions the ideas we took from the existing literature and adapted to this project; in this section, we explain the details of our method. Implementation and experimental results follow in the next section.

1. Data Understanding and Cleaning: Our first step is to understand the data, for pre-processing. We therefore begin with a manual inspection of the dataset.

   We frame the problem as finding the indexes that demarcate the answer, i.e. to extract the indices of the answer in the context.

   - We remove all punctuation and extra spaces from the data.
   - We only consider those examples whose answer is an exact match in the context. In case where there are multiple matches for the answer, we pick the first match as the answer location.

   After preprocessing, we are left with 84080 samples, which we split into training, validation and test sets. Training has 65000 samples, validation (dev) 10000 samples, and test has 9080 samples. Fig 1 shows the distribution of the data in training split.

   Next, statistics of the training data are used to fix the length of the context and question. We set the maximum length of context to 225, and maximum length of question to 35. If the length is smaller, they are suitably padded; if the length is larger, they are truncated. Table 1 shows some statistics from the training data.

2. Embedding Layer: Our first step, in processing, is to convert raw words into a word embedding. We started by using the GLOVE [4] word embedding of size 100 on the raw data. A word embedding is created with the training data, using two methods – CBOW and the skip gram Model.

Stats	Context	Question	Answer	Answer start wrt to context
Min	20	2	1	0.0
Max	240	40	43	0.98
Mean	120	11	3.4	0.38
Median	100	10	2	0.35

Table 1: Stats on training data

(a) CBOW: One approach is to treat "The", "cat", "over", "the", "puddle" as input and, from these words, be able to predict or generate the (missing) center word "jumped". This type of model is called a Continuous Bag of Words [7].

(b) Skip gram: Another approach is to create a model such that given the center word "jumped", the model will be able to predict or generate the surrounding words "The", "cat", "over", "the", "puddle" [7].

In this case, the input is a single word, coded as one hot encoding. This means we use a vector with z components (one for every word in vocabulary); we place a "1" in the position corresponding to the seen word, and "0" elsewhere. The target is words around that window both in past and future. Dean et al [8] further extend this approach, using a dynamic window size with more weight given to words nearer the center word. Unseen and rare words are replaced with a single UNKNOWN category. Other strategies exist as well.

Rush [2] and Socher [3] have proposed character embeddings which can refine results, with the advantage of being able to generate embeddings for words which have not been seen in the training data.

3. Encoder Layer: The next layer we add in the model is a RNN based Encoder layer. The task is for each word to be aware of its neighbors. We note that network parameters are shared – the question and context are passed through the same networks. (We use a bi-directional GRU/LSTM.)

   The hidden parameter h is set to 150, and the output of the RNN is a series of hidden vectors (in both the forward and the backward direction), which we concatenate to get a vector of size $2h$ (i.e. 300). Similarly, we can use the same RNN Encoder to create hidden vectors for the question. In addition, we also use a highway hidden layer, to further reach the neighborhood words which are far away. The output of this layer is $R_c \in R^{N \times 2h}$ for context, and $R_q \in R^{M \times 2h}$ for question.

4. Attention Layer: The objective of this layer is to decide which part of the context, and question, attention should be focused on.

   The input to this layer is the hidden vector generated from the Encoder layer above (making use of both the question and context). We use two attention mechanism, as follows:

   (a) Dot Product:

   To compute dot product attention, we multiply a context vector c_i with each question vector q_j to get vector e_i. Next, we take a softmax over e_i to get α_i. (Softmax is used to normalize the sum of e_i to 1.) Finally, we calculate a_i as the product of the attention distribution α_i and the corresponding question vector.

   (b) Bidirectional:

   Bidirectional attention [1] is an improved idea, that models how attention can flow both from question to context and context to questions.

   We start by generating a similarity matrix $S \in R^{N \times M}$ which has a similar score between context and questions $\forall i \in N, j \in M$.

$$S_{i,j} = w_{sim}^T[(R_c)_i; (R_q)_j; (R_c)_i \cdot (R_q)_j] \tag{1}$$

   For each word j in the question, we estimate attention from the context, using a method similar to the dot product attention.

$$\alpha_i = softmax(S_{i,}) \in R^M \tag{2}$$

$$a_i = \sum_{j=1}^{M} \alpha_j^i (R_q)_j \in R^{2h} \tag{3}$$

Next, we estimate attention for the context, making use of the question. For each context word, we take the max of the corresponding row of the similarity matrix. Then we take the softmax over the resulting vector, giving us an attention distribution β over context locations.

$$m_i = \max_j S_{i,j} \forall i [1..N] \tag{4}$$

$$\beta = softmax(m) \in R^N \tag{5}$$

$$\dot{c} = \sum_{i=1}^{N} \beta_i c_i \tag{6}$$

Finally, both the attention vectors are combined into a single vector.

$$(h_a)_i = [(R_c)_i; a_i; (R_c)_i \cdot a_i; (R_c)_i \cdot (\dot{c})] \in R^{8h} \tag{7}$$

5. Modeling Layer:The task of this layer is is to use the attention output to model the probability distribution on the context which has been conditioned on the question. A single layer bidirectional LSTM is used to model the probability distribution from attention layer. We use forward and backward passes of the LSTM to generate a probability distribution. For each word i in context the bidirectional LSTM outputs $\overrightarrow{b^i}$ corresponding to the forward pass of the data and $\overleftarrow{b^i}$ corresponding to backward pass of the data. They are both concatenated and passed to the output layer.

6. Output Layer: The output layer gives the final answer, i.e. the start and end index of the answer in the context.

   For each word i in context $1 \leq i \leq N$ we have:

$$L_i^s = w_T^1 [\overrightarrow{b^i} \overleftarrow{b^i}] \text{ and } L_i^e = w_T^2 [w_T^1 [\overrightarrow{b^i} \overleftarrow{b^i}]] \tag{8}$$

   These become the input to a fully connected layer which uses softmax to create a p_start vector with probability for start index and a p_end vector with probability for end index.

   Two methods are used to get $start$ and end positions. The first is to directly calculate the positions over soft-max, thus estimating each of them independently.

$$p^s = softmax(L_i^s) \text{ and } p^e = softmax(L_i^e) \tag{9}$$

   However, it is possible to improve the score further, using the fact that start and end are not truly independent. For example, end is certainly greater than (or equal to) start index; this fact can be used in a dynamic-programming approach. We use a combined cross-entropy loss function for the start and end locations, following Yu et al [6], who propose to assign joint probabilities to each possible start and end state.

   Further, since we know that for most answers, the start and end index are at most K words apart, we can look for start and end indexes that maximize $p^s * p^e$. The downside of this approach is that it increases the search space from O(N) to O(N $\times$ K).

The optimizer used in this project is Adam Optimizer. Our loss function is the sum of the cross-entropy loss for the start and end locations. Training is conducted using a dropout keep fraction of 0.4, a momentum value of 0.99, and a learning rate of 0.001. The batch size is adjusted so as to best utilize the available memory.

Embedding	EM	F1
GLOVE	9.1	13.4
CWOB (GENSIM)	9.5	13.6
SKIP GRAM (Gensim)	14.2	17.6
SKIP GRAM (TF)	14.3	17.9

Table 2: Different Embedding on Basic Models on test

4 Experimental Protocol

4.1 Dataset

As mentioned in the above sections, this project makes use of a custom dataset. We have already explained our process for preprocessing and splitting data; we simply note that the same split is used across experiments, to maintain consistency.

4.2 Metric

To evaluate the performance of the proposed model we use two metrics: $F1$ score and Exact Matching.

- $F1$-score: **Precision** is the proportion of items that are actually correct, while **recall** indicates the proportion of the items that should have been found that are actually found. Both these metrics are summarised in the $F1$-score, the harmonic mean of precision and recall.

  We calculate the $F1$ score on the indexes of each test sample.

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{10}$$

- Exact Matching: This is a binary measure, which returns a 1 if there is a **perfect** match between the calculated answer indexes and actual answer indexes, and 0 otherwise.

4.3 Results

Our results, with the different frameworks we use, are as follows.

1. The Gensim CWOB embedding on the dataset performs slightly, but not significantly, better than the GLOVE embedding. We note that the GLOVE embedding is not trained on this data; we suggest that this is because it has been trained on a large corpus (wikipedia).

   We also use tensor flow, with the Skip gram model, to compare with Gensim. Both gave similar results, demonstrating that Gensim is quite an effective library, and provides fast processing.

2. Bidirectional attention performs better on the given dataset. (This shows that attention in both directions does indeed help.)

3. We try to maximize the joint probability of start and end indexes, with smart selection. This approach also successfully increases performance, but the difference is marginal; one reason for this is that LSTM modeling already does a selection in a different way.

4. We get a high EM score of 23.5% and F1 of 29.6% when we use the bidirectional attention with LSTM modeling and smart selection. Currently, we are able to match single-word answer questions more frequently than long answers; one possible explanation is limited training time. As the model gets complex, our rate of finding long answers also increases.

Models	EM	F1
Basic	14.3	17.9
Bidirectional Attention Addition	19.5	25.1
LSTM modeling Addition	23.4	29.2
Smart Selection addition	23.5	29.6

Table 3: Accuracy of Different models on test

5 Discussion and Concluding Remarks

This section begins with a discussion of remaining errors, and the reasons behind them, to help move forward in future work.

As explained in the previous section, we make use of a LSTM encoder layer, bidirectional attention LSTM modeling layer and smart selection to develop the network. Due to resource/time constraints, we were not able to run the networks as much as we would have liked; we expect that using more epochs and a few extra layers will improve results. We would also want to tune the hyperparameters, especially the drop out parameter. The following analysis applies to the current observations.

- In general, we overshoot the boundaries of end span. We note that the proposed algorithm is able to match smaller answers more effectively than large answers. Either adding another layer, or multi layer LSTM, should lead to improved results.
- The answer to given questions can have imprecise boundaries. For example:

  Context: as a young child, bell, like his brothers, received his early schooling at home from his father. at an early age, however, he was enrolled at the royal high school, edinburgh, scotland ...

  Question: what school did bell leave at 15?

  Predicted Answer: royal high school, edinburgh

  Actual Answer: royal high school

  This is a more general problem, and will be trickier to address.

Some possible extensions to the current work are as follows.

1. We would like to explore different embedding dimension and hidden dimension parameters to compare how they perform. Also will transfer learning from different datasets help in learning better representation along the lines of Life Long Learning.
2. Addition of a pruning layer, after modeling layer can help reduce the scope. Alternatively, using multilayer LSTM or R-net based modeling layer as proposed by Microsoft natural processing group [15].
3. It may also be possible to develop a CNN-based encoding which will be faster. Generating a character level encoding can provide better results and generate all the words [1]. For unseen words, we may replace the vector with zero and random vectors, to see how the network performs.
4. Exploring dynamic co-attention[14] and self attention[15] networks for the attention mechanism.
5. We can explore approaches that rely on ensembles, or that fuse multiple networks to a single network for higher levels of accuracy like R-Net with Bidiaf.

In conclusion, our results on the given (custom) dataset are promising, but warrant more exploration, especially with regard to robustness and scalability. We have suggested some mechanisms for further study, and also expect our results to improve with more epochs and processing power.

References

1. Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, Hannaneh Hajishirzi, **Bidirectional Attention Flow for Machine Comprehension**, International Conference on Learning Representations, 2017
2. Yoon Kim, Yacine Jernite, David Sontag, Alexander M. Rush, **Character-Aware Neural Language Models**, arxiv preprint, 2015
3. Caiming Xiong, Victor Zhong, Richard Socher, **Dynamic Coattention Networks For Question Answering**, arxiv preprint, 2016
4. Jeffrey Pennington, Richard Socher, Christopher D. Manning, **GloVe: Global Vectors for Word Representation**, 2014
5. Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, Ming Zhou, **Gated Self-Matching Networks for Reading Comprehension and Question Answering**, Annual Meeting of the Association for Computational Linguistics, 2017
6. Yang Yu, Wei Zhang, Kazi Hasan, Mo Yu, Bing Xiang, Bowen Zhou, **End-to-End Answer Chunk Extraction and Ranking for Reading Comprehension**, arxiv preprint, 2016
7. Richard Socher, $https://cs224d.stanford.edu/lecture_notes/notes1.pdf$, 2018
8. Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, **Efficient Estimation of Word Representations in Vector Space**, arxiv preprint, 2013
9. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, **Attention Is All You Need**, Neural Information Processing Systems (NIPS), 2017
10. **Transformer: A Novel Neural Network Architecture for Language Understanding**, $ai.googleblog.com$, 2017
11. Zhou, X.; Wan, X.; and Xiao, J. **Attention-based LSTM network for cross-lingual sentiment classification**, Empirical Methods in Natural Language Processing (EMNLP), 2016
12. Wojciech Zaremba, Ilya Sutskever, **Learning to Execute**, arxiv preprint, 2014
13. Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio, **Neural Machine Translation by Jointly Learning to Align and Translate**, arxiv preprint, 2014
14. Andre Jonasson, $https://github.com/andrejonasson/dynamic-coattention-network-plus$, 2018
15. Natural Language Computing Group, Microsoft Research Asia, **R-net: Machine Reading Comprehension with Self-matching Networks**, 2017