An LSTM Attention-based Network for Reading Comprehension

Rafael G. Setra
Department of Electrical Engineering
Stanford University
rsetra@stanford.edu

Abstract

A significant goal of natural language processing (NLP) is to devise a system capable of machine understanding of text. A typical system can be tested on its ability to answer questions based on a given context document. One appropriate dataset for such a system is the Stanford Question Answering Dataset (SQuAD), a crowdsourced dataset of over 100k (question, context, answer) triplets. In this work, we focused on creating such a question answering system through a neural net architecture modeled after the attentive reader and sequence attention mix models.

1 Introduction

Machine understanding of text, or machine reading comprehension, is the ability of a system to read a portion of text and showcase some understanding of its meaning. The existance of such a system has many practical applications including improving current understanding of human language and, one of my personal favorites, improving algorithms used in the financial sector [1]. One particular measure of text comprehension is the ability to answer a question based on the text. A simple question-context example could be ”Who was America’s first president?” coupled with an entry from a history book. For simplicity, we will represent an answer as a subset of the given text.

Systems responsible for reading comprehension tend to consume large databases of examples. One recent dataset is SQuAD [2] which is further explained in Section 3. This dataset consists of 100k (question, context, answer) triplets and will be used in this work. We will first preprocess the SQuAD into a quickly useable format and then train a neural network system with the dataset as an input. The system depends on several statistics of the dataset including question and context lengths, and will be evaluated on a disjoint subset of the SQuAD. The final model should then be able to answer previously unseen questions based on new or old contexts.

2 Background

Question answering systems are a significant and long-studied part of NLP [3]. Existing methods have been based on hand engineering grammars [4] and have made use of relatively small datasets. A recent trend has instead been Neural net systems. Part of this is because many existing datasets, although detailed, were small in size [5]. It is only recently that SQuAD has become available. SQuAD is a relatively large dataset and answers in squad are subsets of the context; this is a very flexible property. An advantage of this large dataset is that neural nets may be more easily trained. As a result, neural nets have become a more viable approach than in the years preceding it.

Two recent uses of neural nets for text comprehension have been based off Dynamic Coattention Networks (DCNs) [6] and other Attentive Reader methods [7]. The DCN captures similarities between the question and the document, and applies a dynamic pointing decoder which alternates in learning the beginning and ending of the answer. These networks have received great results. In general, Attentive Reader models can be viewed as applications of Memory Networks [8] to question asnwering. These models encode the information of the question and context together so that token-level information can be used. The two models that I attempted in this work are based off [7]-[8] and are described in the next section.

3 Approach

3.1 Dataset

As mentioned before, SQuAD contains around 100k (question, context, answer) triplets. The context documents are extracted from Wikipedia and the answers are generated by humans. This crowd-sourcing allows the dataset to be effectively realistic. However, it should be noted that humans are not perfect, and so the dataset contains answers which may not be the perfect [2].

The question lengths in this dataset go up to a length a 60 words, and the context lengths up to 766. However, 99% of the lengths are below 40 and 330 for questions and contexts respectivelly. This is represented below in Figure 1. Thus, in our model we will not consider question and contexts lengths above these values. All longer lengths will be truncated, and shorter lengths will be padded with zeros.

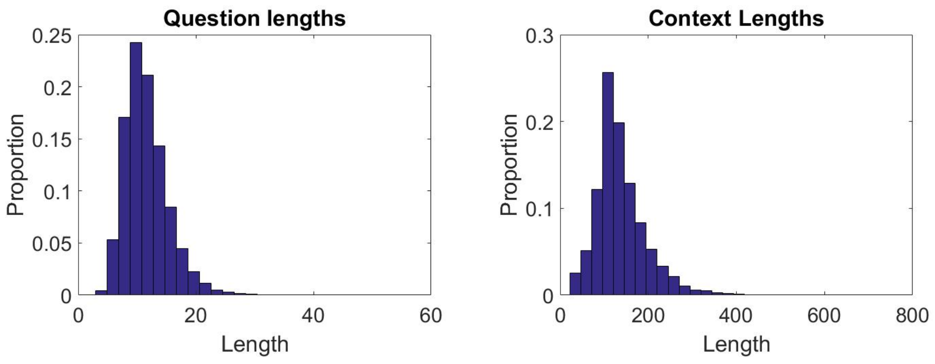

Figure 1: Histograms of lengths from questions(left) and contexts(right) of the SQuAD.

SQuAD provides a great amount of text, but it is necessary to numeracilly represent each individual word for our chosen neural net design. Thus, the SQuAD will be coupled together with GloVe [9] word representations of dimensionality 100. These word representations are trained on global word-word occurance statistics from a corpus, and showcase linear substructures in the word vector space. The vocabulary size is 400k and it has been pretrained on Wikipedia 2014 and Gigaword 5. However, there are out of vocabulary words and these will be initialized with random values. There was a choice to also initialize these with zeros, but that implies that they should be treated the same way which is a wild assumption.

3.2 Model Framework

As mentioned before, the goal is to predict an answer given a question and context. Questions and context paragraphs are of the form $q = \{q_1, q_2, ..., q_m\}$ and $c = \{c_1, c_2, ..., c_n\}$ respectively; each element is a GloVE word representation so that $q_i, p_i \in \mathbb{R}^{100}$. Answers are represented as span tuples (a_s, a_e) which correspond to the start and ending index of the context; such a tuple is equivalent to an answer of $\{c_{o_s}, ..., c_{o_e}\}$. The model will be trained on the SQuAD (q, c) inputs and (o_s, o_e) outputs. Additionally we have the following constraints: $o_s \leq o_e, 0 \leq o_s, o_e \leq m$.

The two models I have considered to generate these outputs are an Attentive Reader-like model based on [6],[10] and a Sequence Attention Mix model based on [7]. The final output is then an ensemble of the two answers. Ensemble methods such as these help improve the answer by averaging out noise. Additionally, these models make use of deep LSTM networks [11] and are trained through the Adam optimizer [12]. Finally, dropout [13] and L2 regularization are applied to reduce overfitting.

3.2.1 Deep LSTM

Long short-term memory [11] cells are vital to my chosen models. Each cell in a deep LSTM network makes use of a current input, a previous output, and memory coefficients. These memory coeffcients allow the network to represent current and previous inputs in a flexible manner. In other words, the general idea in deep LSTM is that the network takes in an input sequence $\{x_1, x_2, ..., x_k\}$ and outputs another sequence $\{h_1, h_2, ..., h_k\}$. The key fact is that an output vector h_i gives an representation of the input subsequence $\{x_1, ..., x_i\}$. These cells have been successful in NLP tasks [7],[11] and have shown the ability to effectively represent sequences of words into a vector representation. A bidirectional LSTM network basically takes in an input sequence and its reversed version, and outputs two corresponding output sequences. This allows more information with less bias towards the start or end of a sequence.

3.2.2 Attentive Reader-like

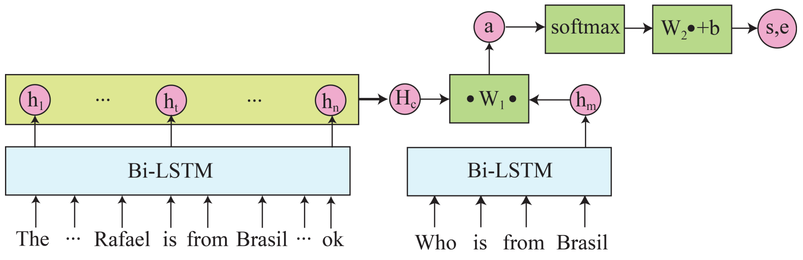

Figure 2: Attentive Reader diagram.

The diagram of the model is shown in Figure 2. The first step of the model is to apply an encoding to the context $c = \{c_1, c_2, ..., c_n\}$ through a birectional LSTM network. The resulting forward and output vectors $\overrightarrow{H_c} = \{\overrightarrow{(h_c)_i}\}$ and $\overleftarrow{H_c} = \{\overleftarrow{(h_c)_i}\}$ are concatenated into a final paragraph representation $(H_c)_i = (\overrightarrow{(H_c)_i}, \overleftarrow{(H_c)_i})$. Each output vector is chosen to be in $\mathbb{R}^h$, where $h = 200$. The concatenation is thus in $\mathbb{R}^{2h}$. Another bidirectional LSTM network is also applied to map the question sequence $q = \{q_1, q_2, ..., q_m\}$ into $h_q = (\overrightarrow{(h_q)_m}, \overleftarrow{(h_q)_m})$. All of the output states are used in the paragraph representation but only the final states are used in the question representation. This choice is because the questions are typically shorter in length and can be adequately represented with less information.

The next step is to compare the question and context embeddings so that we may select the most relevant information. This is done by computing an output vector a:

$$a_i = \text{softmax}(q^T W_1 (H_c)_i), i = 1, ..., n.$$

Here we use $W_1 \in \mathbb{R}^{2h \times 2h}$ for a bilinear term which provides more flexibility in computing the similarity between c and q than a direct product. The softmax is applied to provide nonlinear normalization. For the toy example in Figure 2, a sample attention vector is shown in Figure 3. Note that the attention values are high when at time values where the paragraph words ”Rafael is from Brasil” have similarity with the question ”Who is from Brasil”.

Then, we calculate the likelihood for our output tuple based on each entry in the context:

$$p_s = W_{2s} a + b_s$$

$$p_e = W_{2e} a + b_e$$

Where $W_{2s}, W_{2e} \in \mathbb{R}^{n \times n}$ and $b_s, b_e \in \mathbb{R}^n$. Different matrices are used for each output to allow disparity.

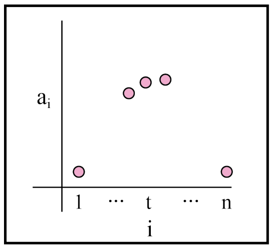

Figure 3: Attentive Reader sample attention vector.

3.2.3 Sequence Attention Mix

The diagram of this model is shown in Figure 4. The model is similar to the previous in the initial stages. The first step is to encode the context and question through the same unidirectional LSTM network so that we may share representation power. The resulting representation of all of the outputs are then $(H_c)_i = (\overrightarrow{(H_c)_i}, \overleftarrow{(H_c)_i})$ and $(H_q)'_i = (\overrightarrow{(H_q)_i}, \overleftarrow{(H_q)_i})$. Note that now we use all of the question output vectors unlike the previous model. This helps reduce too much generalization from reusing the network.

To allow for more variation between the question and context we apply a nonlinear transformation to H_q:

$$H_q = \tanh(W_1 H'_q + b_1) \in \mathbb{R}^{h \times m}.$$

Where $W_1 \in \mathbb{R}^{h \times m}, b_1 \in \mathbb{R}^h$. In the previous problem we allowed for variation by simply using two LSTM networks. This difference method was chosen so that the two models would be dissimilar and provide more benefit when combined together through an ensemble.

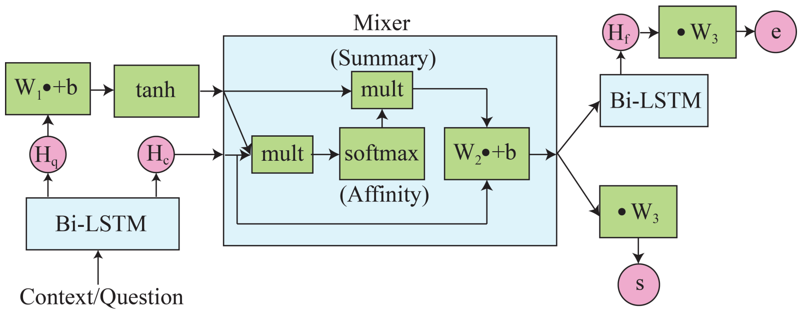

Figure 4: Sequence Attention Mix diagram.

The next step is to mix the question and context information together. First we compute an affinity matrix $A = \text{softmax}(H_q^T H_c) \in \mathbb{R}^{m \times n}$. The affinity matrix contains scores corresponding to all pairs of document and question words, and are normalized row-wise. An affinity matrix for the same toy example in the previous part is shown in Figure 5. Note that scores are higher between context words which are similar to question words such as "from" and "from". This allows us to focus in on relevant portions of the context document.

Next we compute a summary of the questions in light of each word of the document as $Q = H_q A$. The mixing is then complete when we concatenate the question summaries with the context information and apply it through a linear filter:

$$M = W_2(Q, H_c) + b_2 \in \mathbb{R}^{2h \times n}.$$

Here $W_2 \in \mathbb{R}^{2h \times 2h}, b_2 \in \mathbb{R}^n$. We now have a mixed representation of the context. The likelihood for the start entries are then:

$$p_s = MW_{3s}$$

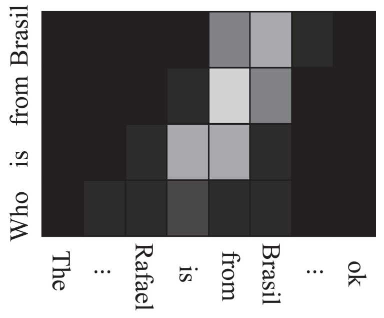

Figure 5: Sequence Attention Mix sample affinity matrix. Brighter colors correspond to larger magnitudes.

Where $W_{3s} \in \mathbb{R}^{1\times 2h}$. To further allow variation between the start and ending probabilities, so that the model may better learn to produce the constraint $o_s \leq o_e$, we apply M through a bi-LSTM to produce the concatenated output $H_f \in \mathbb{R}^2 h \times n$. Finally, the probabilities of the ending indices are:

$$p_e = H_f W_{3e}$$

Where $W_{3e} \in \mathbb{R}^{1\times 2h}$.

3.2.4 Final Output

In the final result, the likelihoods from the two models are averaged together, and the final answer tuple is:

$$(o_s, o_e) = (\text{argmax}_i(p_s), \text{argmax}_i(p_e)).$$

Furthermore, loss for the two methods is calculated by:

$$L = \text{sce}(p_e, e_t) + \text{sce}(p_s, s_t) + \sum_i 0.001 * \text{L2}(W_i)$$

Where sce is the softmax cross entropy between the generated probabilities and the one hot vector with a one at index e_t or s_t. These indices represent the true values of the end and start indices. L2 is the L2 loss of the non-bias matrices.

4 Experiments

The two models above were trained with a piece of the given training dataset from SQuAD which contains roughly 80k (question, context, answer) triplets. The model was then validated through a smaller dataset with roughly 8k triples, and the model with the best score on this dataset was selected. To finish it off, the final model was run on a dev dataset of roughly 10k triplets, and a hidden test dataset.

The score of a model is determined through two metrics: F1 and exact match. The F1 score makes use of the recall r and precision p of the output answer. These are defined as:

$$r = \frac{\text{true positive}}{\text{true positive} + \text{false negative}}$$

$$p = \frac{\text{true positive}}{\text{true positive} + \text{false positive}}$$

$$F1 = \frac{2rp}{r+p}$$

It can be thought of as a measure of the average overlap between the prediction and ground truth answer. We treat the prediction and ground truth as a bag of tokens, and compute the F1 based off that.

Table 1: Performance on various datasets

Dataset	F1 Score	Exact Match
Train	17.8	4.1
Validation	15.6	2.4
Dev	6.2	0.5
Test	6.8	0.7

Exact match is a percentage of predictions that match one of the grount truth answers exactly. The scores from my resultant model on the training, validation, dev, and hidden test datasets are shown in Table 1.

Overall, the results are terrible. This makes it difficult to interpret the model's behavior. These results were for the mixer model. The ensemble and the attentive reader model have very similar results. There were about 5% unanswered questions, due to the start index being larger than the ending index. Adding in a constraint into the code to avoid this should slightly boost the score. However, in theory the mixer model should help prevent this behavior.

As far as the exact match (EM) goes, there were roughly 50 exact matches. All but 1 of these had answers with lengths below 4. This makes sense because as seen in Figure 6, most of the trainable answers were of very small length; 80% of the answers were of length below 4. So if my model learned anything, it was how to locate these small answers.

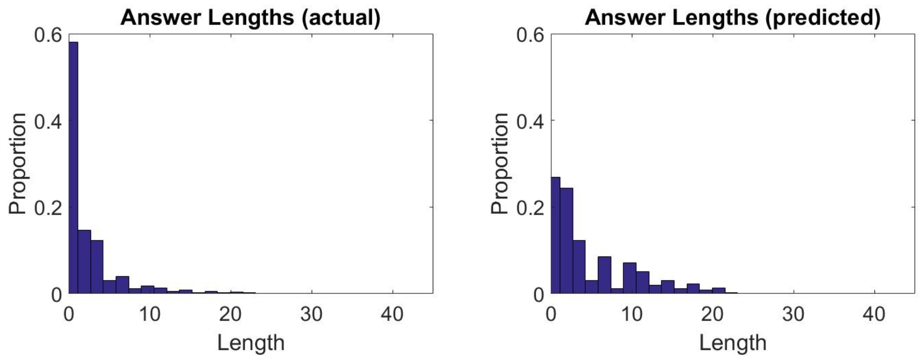

Figure 6: Histograms of lengths from answers of the SQuAD (left) and predicted (right).

The one long answer that was correct was the (question, answer) pair (”What was report P-2626” ,”a large-scale, distributed, survivable communications network”). I have no other data on longer questions so this one match was likely due to randomness/luck. In theory, bi-LSTMs should be able to deal with longer answers than these.

The F1 score is significantly larger than the exact match score. This suggest some amount of overlap between predicted and ground truth answers. The reason for this can be seen in Figure 6. Basically, the model predicted very large lengths overall even on the training dataset, when the distribution should have focused on smaller answers. This cause overlap to be more likely which is why the F1 score is much higher than EM. However, the lengths are too large, which is why the F1 score is still not large enough to be called acceptable.

In the next subsection I will spend some time documenting how I attempted to diagnose my errors.

5 Error Diagnosis

The first test I looked at was whether the model would be able to overfit a small dataset. With inputs of lengths up to 5000, I was able to achieve an F1 and EM score of near 100 after 5 epochs. This means that my model, at the very least, had the ability to learn. Since the model could learn, I next changed my hyperparameters; perhaps the model was simply learning too slowly with a larger training input. However, results did not change with learning rates varying from 0.0001 to 0.1.

Next I looked at what most of the answers looked like. At first, 40% of them were empty and so I fixed the masking mechanism. This dropped empty answers to near 5%. Afterwards, answers were very large and this has not been remedied. One attempt to remedy large answers was to try a different model. In the end the two models I attempted did not produce desireable no results. I also considered unidirectional LSTM networks.

Other small tweaks were to the maximum considered question and context lengths, state sizes and levels of regularization. None were effective. The gradient norms also seemed normal with vales around 1-4. The loss did steadily go down to a value of around 7 from an initial value close to 30. A further diagnosis which may prove effective would be to print out a few of the affinity matrices and attention vectors to determine if they make sense. Unfortunately there was not much time for this.

In conclusion, the error diagnosis was inconclusive.

6 Conclusions

The model did not perform well. The first step I would do would be to figure out what exactly is wrong with the basic implementation. However, if I had gotten the model working, I would have liked to expand on it to include a more complex decoder. For the two models considered, I ended up with a naive linear decoder. However, a more involved method such as a dynamic pointing decoder [7] should produce better results. Furthermore, implementing the Impatient Reader [6] allows the model to effectively reread the context document several times; this should also improve results.

I would also have liked to test the effects of sharing LSTM networks between question and context inputs. For example, the Attentive Reader did not share a network, but the Sequence Mixer did. If the two models had worked appropriately I would have liked to see the effects of changing this aspect. My guess is that sharing a network would be effective on questions which match very closely with the context. A small example would be the question of ”What is your name” coupled with context ”My name is Rafael”.

Although my model did not work, I learned a lot through this project and course.

Acknowledgments

This final assignment was very difficult for me. While I did not accomplish as much as I had wanted, I still learned a great amount. I would like to thank the TAs and instructors for all of the help throughout this quarter; it was much appreciated.

References

[1] Saret, J. N. (2016) An AI Approach to Fed Watching. *Two Sigma*, https://www.twosigma.com/insights/an-ai-approach-to-fed-watching.

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

[3] Norvig, P. (1978) A Unified Theory of Inference for Text Understanding. Ph.D. thesis, University of California, Berkeley.

[4] Riloff, E. & Thelen, M. (2000) A rule-based question answering system for reading comprehension tests. *Proceedings of the ANLP/NAACL Workshop on Reading Comprehension Tests As Evaluation for Computer-based Language Understanding Sytems*.

[5] Berant, J. & et. al. (2014) Modeling biological processes for reading comprehension. *EMNLP*.

[6] Hermann, K. M. et. al. (2015) Teaching Machines to Read and Comprehend. *arXiv preprint arXiv:1506.03340*.

[7] Xiong, C. & Zhong, V. & Socher R. (2017) Dynamic Coattention Networks for Question Answering. *arXiv preprint arXiv:1611.01604*.

[8] Weston, J. & Chopra S. & and Bordes, A. (2014) Memory networks. *CoRR*, abs/1410.3916,

[9] Pennington, J. & Socher, R. & Manning C. D. (2014) GloVe: Global Vectors for Word Representation. https://nlp.stanford.edu/projects/glove/

[10] Chen, D. & Bolton, J. & Manning, C. D. (2016) A Thorough Examination of the CNN/Daily Mail Reading Comprehension Task. *arXiv preprint arXiv:1606.02858*.

[11] Hochreiter, S. & Schmidhuber, J. (1997) Long short-term memory. *Neural Computation*, **9**(8):17351780.

[12] Kingma, D. P. & Ba, J. (2014) Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*.

[13] Srivastava, N. et. al. (2014) Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* **15**:1929-1958.