SQuAD Reading Comprehension with Coattention

Austin Hou
Department of Electrical Engineering
Stanford University
Stanford, CA 94305
austhou@stanford.edu

Abstract

Reading comprehension is an important task in NLP, which involves teaching a machine to understand text enough to answer questions. The Stanford Question Answering Dataset (SQuAD) is a dataset consisting of 100,000 question-context-answer datapoints. Here, deep learning methods are used to answer questions based on context data. A model based on the Attentive Reader [1,2] model is used as a baseline, with elements of a Dynamic Coattention Network [3] applied. Co-dependent attention representations that combine the individual representations of the question and context paragraph are implemented. The model is evaluated using F1 and exact match (EM) scores.

1 Introduction

Teaching a machine to understand text is a difficult task – especially when human performance is not perfect [4]. Traditionally, machine comprehension has relied on multi-step processes that include steps such as linguistic and feature analysis, syntactic or dependency parsing, named entity recognition, an many others. With recent advances in deep learning, there have been many studies done that concentrates on applying neural network models to this task.

Recently, the SQuAD dataset was released, which consists of 100,000 question-context-answer datapoints[4]. SQuAD is useful because each answer is contained in the context, and it is significantly larger than any previous quality datasets – though other large datasets exist, they have been machine-generated and are of a different nature [1].

Tesla gained experience in telephony and electrical engineering before emigrating to the United States in 1884 to work for Thomas Edison in New York City. He soon struck out on his own with financial backers, setting up laboratories and companies to develop a range of electrical devices. His patented AC induction motor and transformer were licensed by George Westinghouse, who also hired Tesla for a short time as a consultant. His work in the formative years of electric power development was involved in a corporate alternating current/direct current "War of Currents" as well as various patent battles.
Q: In what area of the United States did Tesla move to?
A: New York City

Table 1: A sample from the SQuAD dataset. The context paragraph is at top, with the question and answer at bottom.

Here, I base the model off of the Attentive Reader[1,2], with parts of the Dynamic Coattention Network applied [3]. Coattention is used to capture the interaction/relationship between the question and the context paragraph, which is not captured in a baseline model.

2 Background

A number of deep learning-based studies applied to reading comprehension have been conducted. Some of them are detailed below.

3.1 Deep LSTM Reader

LSTMs are commonly used in many NLP applications [5]. Deep LSTMs have the capability of handling much longer sequences due to their ability to encode data while preserving significant amounts of information. The Deep LSTM reader concatenates the context and the question, and feeds it into a uni- or bidirectional LSTM.

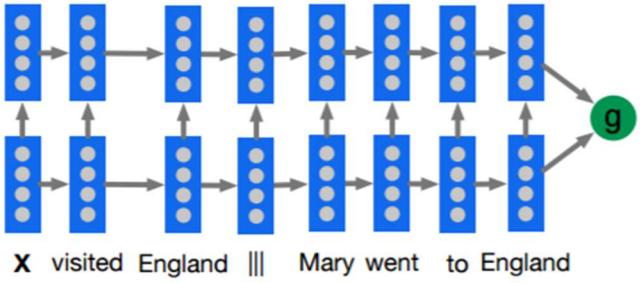

Figure 1: A Deep LSTM reader (two layer). Figure from [2]

3.1 Attentive Reader

The Attentive Reader model attempts to improve on the shortcomings of the Deep LSTM Reader [6]. Deep LSTMs are good at embedding data into vectors, but to answer questions they must propagate question-answer dependencies through potentially long spans. The Attentive Reader model introduces an attention vector based on the question that can be multiplied with the context representation.

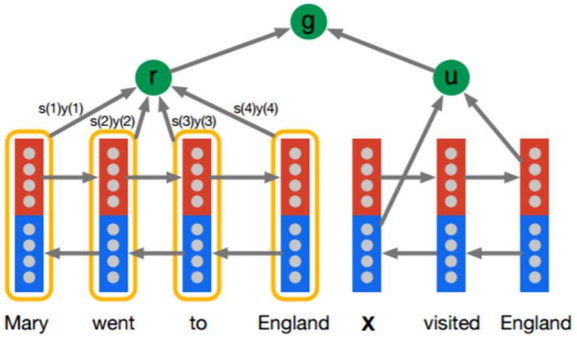

Figure 2: an example of one Attentive Reader model. Here, r represents the attended context representation, and u represents the question representation. Figure from [2]

3.2 Dynamic Coattention Networks

An attention mechanism can help a lot with reading comprehension, but does not fully encapsulate the interactions between the context and the question. Dynamic Coattention

networks [3] aim to solve this by introducing a coattention mechanism. In the DCN, an affinity matrix consisting of the product of the embedded question and context representations is used to generate a coattention mechanism. In addition to the coattention mechanism, the authors propose a Highway Maxout Network (HMN) that combines the strengths of Maxout Networks [7] and Highway Networks [8].

3 Methods

Here, I create a baseline inspired by the Deep LSTM and Attention models, but implementing a simplified coattention mechanism. Instead of feeding the encoded coattention context into a HMN, it is fed into a feedforward network as a decoder. The structure is as follows:

3.1 Data Encoding

To encode the question and context paragraph data, a BiLSTM [] is used for each, with the word vectors x_i as input:

$$d_t = \text{BiLSTM}_{enc}(d_{t-1}, d_{t+1}, x_t^D)$$

$$q_t = \text{BiLSTM}_{enc}(q_{t-1}, q_{t+1}, x_t^Q)$$

Where $[x_1^D, x_2^D, \ldots x_n^D]$ are the context word vectors and $[x_1^Q, x_2^Q, \ldots x_m^Q]$ are the question word vectors. The hidden states d_t and q_t are concatenated to form D and Q, of dimension lxn and lxm, respectively, which are the encoded representations of the data and question. This is the baseline encoder.

3.1 Coattention

It is desirable to create an attention mechanism that can capture the relationships between the words in the question and context simultaneously. I follow the logic of [3], creating and affinity matrix L:

$$L = D^T Q$$

L is an m by n matrix, that is the product of the hidden state representations of each word in the context and question. Next, the softmax is used on each row and column to produce attention weights across the context document D for each question word in Q, and vice-versa:

$$A^Q = \text{softmax}(L)$$

$$A^D = \text{softmax}(L^T)$$

We then use these to find the attention contexts:

$$C^Q = DA^Q$$

$$C^D = [Q;C^Q]A^D$$

In the second term, C^D is referred to as the coattention context in [3]. The next step combines the temporal information from the context paragraph with the coattention context, and feeds it into a BiLSTM:

$$u_t = \text{BiLSTM}(u_{t-1}, u_{t+1}, [d_t;c_t^D])$$

3.2 Decoding

Finally, the two end states from this BiLSTM are concatenated. I call this r_{in}:

$$r_{in} = [u_s;u_e]$$

Two matrices of weights, W_{D1} and W_{D2}, both of dimension lx4l, are introduced:

$$r_1 = W_{D1}\, r_{in}$$

$$r_2 = W_{D2}\, r_{in}$$

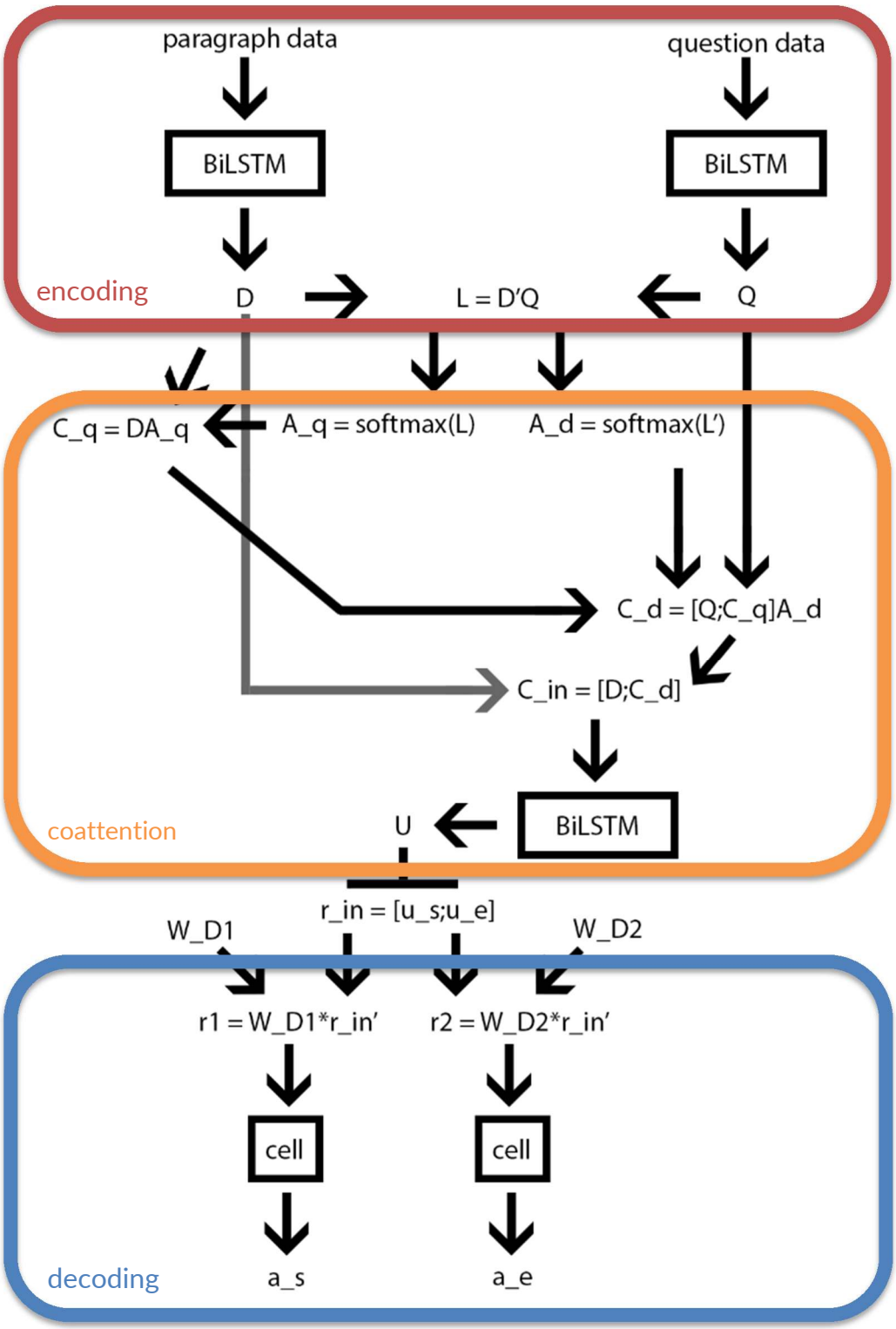

Figure 3: a schematic of the neural network architecture. Shown are the encoding, coattention, and decoding schemes in red, orange, and blue, respectively. At the bottom, a_s and a_e represent the start and end pointers/locations of the answer in the context paragraph, respectively.

Both of these are of dimension lx4l. In the last step, each of these is passed through a single feed-forward layer to produce probabilities for the start and end locations for the answer to the question.

The Adam optimizer [9] is used in training, and the two encoding BiLSTMs incorporate a dropout function with a parameter of 0.7. In training, we minimize the softmax cross entropy loss [10]. We train over a dataset of ~81,000 datapoints over 20 epochs with a learning rate of 0.0015, and test on a dev dataset consisting of 10,570 datapoints.

4 Results and Discussion

4.1 Evaluation Metrics

The model was evaluated using two metrics: the F1 score and the Exact Match (EM) score. F1 is based on the precision (p) and recall (r):

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

Where p and r are defined as:

$$p = \frac{tp}{tp+fp}, \quad r = \frac{tp}{tp+fn}$$

Both metrics are used for a couple reasons. EM is a valuable metric for obvious reasons – it measures how many answers were exactly correct. However, if an answer is not exactly correct, such as if the answer start and/or end are off by a couple words, it would not be captured by EM alone. Thus we need another measure of overlap, which is F1.

4.2 Evaluation Results

Evaluation was done locally on 'val' and 'dev', and on CodaLab on 'dev' and 'test. Scores for dev and test are as follows:

From evaluating locally on the dev dataset, an F1 of 8.918 was obtained. An exact match score of 2.374 was obtained. On CodaLab, results were very different. On the dev dataset, an F1 of 0.811 and em of 0.038 was obtained. This is a strange result and one that I did not have enough time to resolve. This is possibly due to a bug in my online submission and evaluation scripts.

From evaluating on the test dataset, an F1 of 0.775 was obtained. An exact match score of 0.094 was obtained.

4.3 Error Analysis and Discussion

These are extremely low, so it is rather difficult to conduct a detailed analysis. The higher F1 score implies that some answers contained parts of the ground truth, but may have contained more or less words on either side. Examining successful predictions, the vast majority of successful exact matches were one or two words long, suggesting that the model is better at predicting short answers (see Fig. 4).

Looking at the results, there were a significant number of answers that were empty. This could have been due to a suboptimal model, or because the answer start pointer was after the answer end pointer.

An interesting phenomenon was that the majority of correctly answered questions consisted of numbers, commonly a date (examples include 'June 1978', '2nd century BCE', and '1996') or a number (examples include 'eleven', '60 minutes', and '900,000'). This suggests that the model did learn how to answer numerical questions better than others.

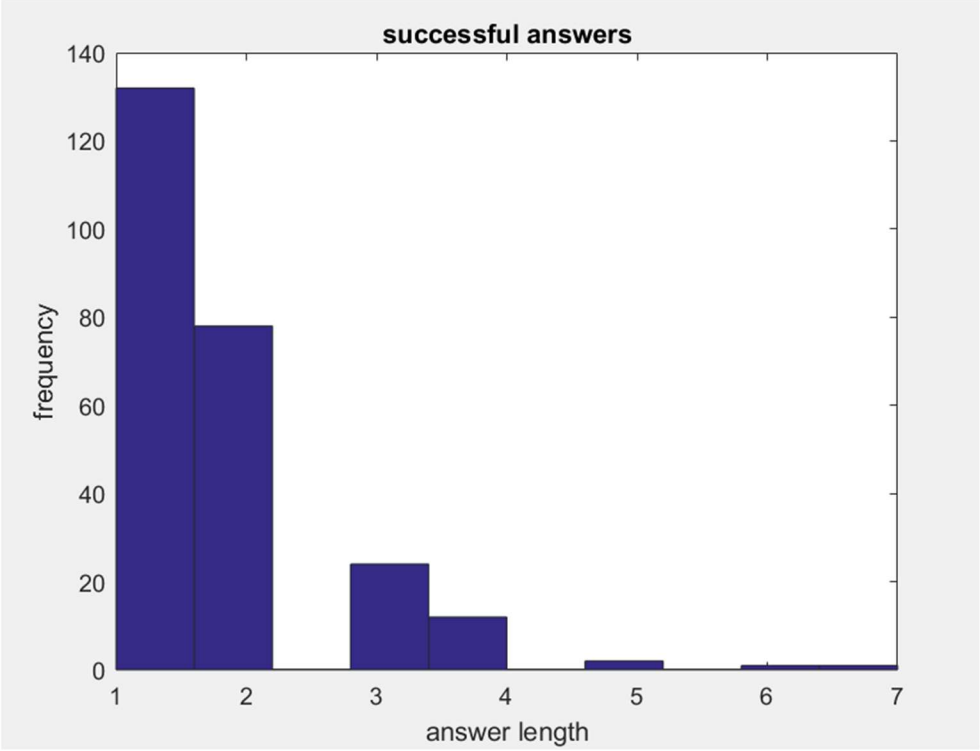

Figure 4: a histogram of the answer length of successful exact matches.

Figure 5: A plot of f1 and exact match scores, vs. epoch evaluated locally on 'val'. Note that the scores were still rising when training was ended due to time constraints.

It is clear that the results are not optimal. This could have been due to a number of reasons, but a significant reason could have been model training. It took a long time to train the model, and though the F1 and EM scores more or less rose every epoch, they were still rather low when training stopped. Additionally, and relatedly, the parameters were not optimal.

The parameters were set as default at first. It was difficult to tune due to the long training times involved, and I was only able to adjust them a handful of times. Finer hyperparameter tuning could have made a difference.

Looking at a graph of training scores, it looks like the model did not converge by the 17th epoch (see Fig. 5), and that the training scores were still rising steadily. Running the training for a larger number of epochs would have helped.

Also, overfitting is a possibility. Training took a long time, and though the loss steadily fell, the F1 and EM scores did not always rise. Given that the hyperparameters were not exhaustively tuned, this is a significant possibility. Dropout was added to the baseline model, but adding dropout to more parameters, such as W_D1 and W_D2, or adding L2 regularization could help avoid overfitting.

Additional possible reasons are a bug in the model, or the model is just not powerful enough for the task, due to the simplifications that were made in implementing the Coattention mechanism.

5 Conclusion

The model was unsuccessful. The F1 and EM scores were very low. The model did show that it could predict numerical answers and short answers at a higher rate than other answers, however. There are many improvements that could be made to improve the model in the future, including training for longer, tuning hyperparameters more finely, experimenting with adding more regularization, and changing the model.

6 References

[1] Danqi Chen, Jason Bolton: “A Thorough Examination of the CNN/Daily Mail Reading Comprehension Task”, 2016

[2] Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman: “Teaching Machines to Read and Comprehend”, 2015; http://arxiv.org/abs/1506.03340 arXiv:1506.03340.

[3] Caiming Xiong, Victor Zhong: “Dynamic Coattention Networks For Question Answering”, 2016

[4] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev: “SQuAD: 100,000+ Questions for Machine Comprehension of Text”, 2016; http://arxiv.org/abs/1606.05250 arXiv:1606.05250

[5] Ilya Sutskever, Oriol Vinyals, and Quoc V. V Le. Sequence to sequence learning with neural networks. In Advances in Neural Information Processing Systems 27

[6] Alex Graves. Supervised Sequence Labelling with Recurrent Neural Networks, volume 385 of Studies in Computational Intelligence. Springer, 2012.

[7] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron C Courville, and Yoshua Bengio. Maxout networks. ICML (3), 28:1319–1327, 2013.

[8] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1):1929–1958, 2014.

[9] Diederik P. Kingma: “Adam: A Method for Stochastic Optimization”, 2014; [http://arxiv.org/abs/1412.6980 arXiv:1412.6980].

[10] Peter Sadowski. "Notes on backpropagation", https://www.ics.uci.edu/~pjsadows/notes.pdf, 2016. Online.