Assignment 4: Reading Comprehension

Vishakh Hegde
Stanford University
vishakh@stanford.edu

Abstract

Reading comprehension is the task of understanding a piece of text by a machine. We train an end-to-end neural network that models the conditional distribution of start and end indices, given the question and context paragraph. We build on top of the baseline suggested in the Assignment, and explore new models to implement attention. We also measure the performance of the models and analyse the reason for improved performance over the baseline.

1 Introduction

Natural Language Understanding lies at the core of artificial intelligence and holds the problem of reading comprehension. In recent years, researchers have developed several benchmark datasets to measure and accelerate the progress of machine comprehension technologies. RCTest (Richardson et al., 2013) is one such dataset, which consists of 500 fictional stories and 4 multiple choice questions per story (2,000 questions in total). A variety of machine comprehension methods were proposed based on this dataset.

End-to-end learning systems are now replacing many of the traditional block-by-block learning systems in the fields of Natural Language Processing and Computer vision. The problem of machine comprehension is no different. However, such end to end supervised systems (which typically have millions of parameters) require large amounts of data to model the complexities in the data distribution. However, the limited size of the RCTest dataset prevented researchers from building end-to-end deep neural network models. For the task of machine comprehension, such a large dataset was recently made available by [3] and is called Stanford Question Answering Dataset (SQuAD).

1.1 SQuAD Problem Statement:

In the SQuAD task, answering a question is defined as predicting an answer span within a given context paragraph. (add some more stuff). SQuAD consists of questions posed by crowd-workers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage. There are 100,000 triplets of (question, context, answer) like the example in the table below:

Context Paragraph: On 24 March 1879, Tesla was returned to Gospic under police guard for not having a residence permit. On 17 April 1879, Milutin Tesla died at the age of 60 after contracting an unspecified illness (although some sources say that he died of a stroke). During that year, Tesla taught a large class of students in his old school, Higher Real Gymnasium, in Gospic.
Question: Why was Tesla returned to Gospic? *Answer*: not having a residence permit

1.2 Why is this problem hard?

If we go about solving this problem in the canonical manner, the most naive method would be to consider all possible answers, rank them based on some metric and choose the best answer based on that ranking. The number of possible answers scales quadratically. i.e. if N is the number of passage words, the number of possible answers to consider is in the order of $O(N^2)$.

1.3 Possible solution

Researchers have proposed a plethora of end-to-end neural models to tackle this problem of machine comprehension. The goal is to model the index of start and end of the answer with some probability distribution. Like most problems in language, the problem of memory exists - for a neural system, it is hard to encode information over long sequences. A key ingredient to the advancement of such end-to-end systems has been the use of neural attention mechanism, which guides the system to focus on a targeted area within context (in both text and images) that is most relevant to correctly answer the question [6], [1].

1.4 What to expect in this report?

In this report, we highlight all the methods we've tried and report some performance measures for comparison. Guided by the good advice in the assignment handout, we first worked on implementing the baseline. After some experimentation, we tried more complex RNN cells and finally use a biLSTM to encode both question and context paragraph. Based on the encoding in the final hidden, we predict the start and end labels independently.

After this, we worked on implementing attention, as that is one of the key ingredients missing in the baseline. After using a biLSTM to encode the question and context paragraph, the most canonical method to implement attention is to have a weight matrix which takes a weighted average of all the hidden states seen so far and use that for prediction instead of using only the final hidden state. We tried this as well.

The key insight we had was on the use of convolutional neural networks to implement attention. We explain all these techniques in detail.

2 Related Work (needs more work - replace it completely)

For the SQUAD dataset, the original paper [3] implemented a linear model with sparse features based on n-grams and part-of-speech tags present in the question and the candidate answer. The MatchLSTM paper [5] is very similar to the baseline provided in the assignment. The paper on Multi-Perspective Context Matching for Machine Comprehension [5] describes a layer to filter out irrelevant and redundant information from the passage using a relevancy score for every word in the context paragraph.

In the Bi-Directional Attention Flow (BIDAF) network paper [4], the authors introduce a multi-stage hierarchical process that represents the context at different levels of granularity and uses bidirectional attention flow mechanism to obtain a query-aware context representation without early summarization.

As SQuAD became more popular, more an more people have been competing on the leader-board. All of them have bettered the original model proposed by [3]

3 Performance Metric

The original SQuAD paper introduces two metrics to evaluate the performance of a model: Exact-Match (EM) and F1 score. We use the same metrics to evaluate our model.

Exact match is a metric that measures the percentage of predictions that match one of the ground truth answers exactly.

F1 score is a metric that loosely measures the average overlap between the prediction and ground truth answer. We treat the prediction and ground truth as bags of tokens, and compute their F1. We

Cell Name	State Size	Embed Size	Epochs Trained	F1 score	EM score
Basic RNN	200	100	5	0.157	0.094
LSTM	200	100	6	0.176	0.107
bi-LSTM	200	100	5	0.168	0.102

Table 1: Performance of baseline models for different cells used

take the maximum F1 over all of the ground truth answers for a given question, and then average over all of the questions.

4 Models Used

4.1 Baseline Models

We briefly explore three different cell architectures (RNN, LSTM and bidirectional LSTM) for our baseline model. In all three cases, we first encode the question using the cell and use the final hidden state as the initial state of the cell which encodes the paragraph. As is the best practice, we initialize the first hidden state for the question encoder to be a vector of all zeros. The only inputs to any cell are the word and the previous hidden state. Once we obtain the final hidden state of the paragraph, we simply use fully connected layers to output the probability distribution over the start and end indices.

For the bi-LSTM, we obtain two hidden states. We will call it $h_forward$ and $h_backward$. Before using a fully connected layer, we concatenate the two vectors. The index corresponding to the maximum probability is chosen to be the start/end index. Once we get the final hidden state of the context paragraph, the start and end indices are guessed independently.

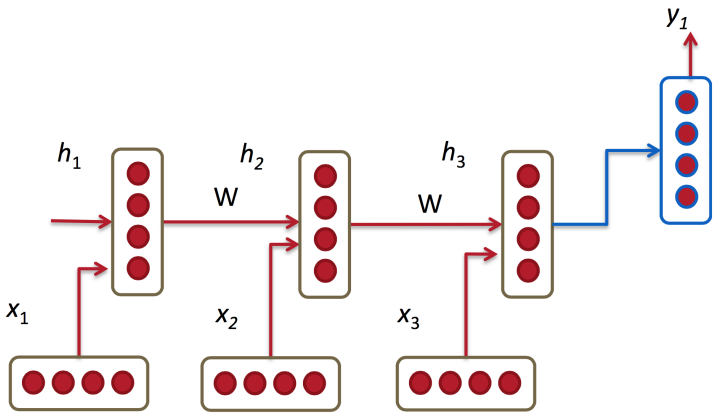

Figure 1: Baseline model overview. We experiment with three different cells: Basic RNN, LSTM and bidirectional LSTM

The loss function is a sum of two loss functions: one representing the error in guessing the start word and the other representing the error in guessing the end word. Each of these loss functions is softmax cross entropy loss on the logits. The ground truth is one hot vector with one at the index where the start/end word occurs in the paragraph.

We found from our performance analysis of different baseline models that the model with the LSTM cell performs best among the three. However, we believe that if we had trained the models over more epochs, bi-LSTM would ultimately triumph since it encodes more information than the LSTM cell model. Therefore, in all of our subsequent experiments, we use a bi-LSTM cell to encode both the question and the context paragraph.

4.2 CNN for Attention (our) Model

Even though LSTMs are better than RNNs for modeling long sequences, the vanishing gradient problem persists. Therefore, researchers have come up with ways to counter this by adding attention mechanisms.

The Concept of Attention

As explained in the blog `http://www.wildml.com`, with an attention mechanism we no longer try encode the full source sentence into a fixed-length vector. Rather, we allow the decoder to "attend" to different parts of the source sentence at each step of the output generation. Importantly, we let the model learn what to attend to based on the input.

We chose to implement the attention mechanism with a convolutional neural network. Convolutional neural networks are used on high-dimensional inputs and use hierarchical features to embed them into a lower dimensional space such that the embedding captures the global correlations in the input. We proceeded with the hunch that this could help identify features that are most important for predicting start and end indices.

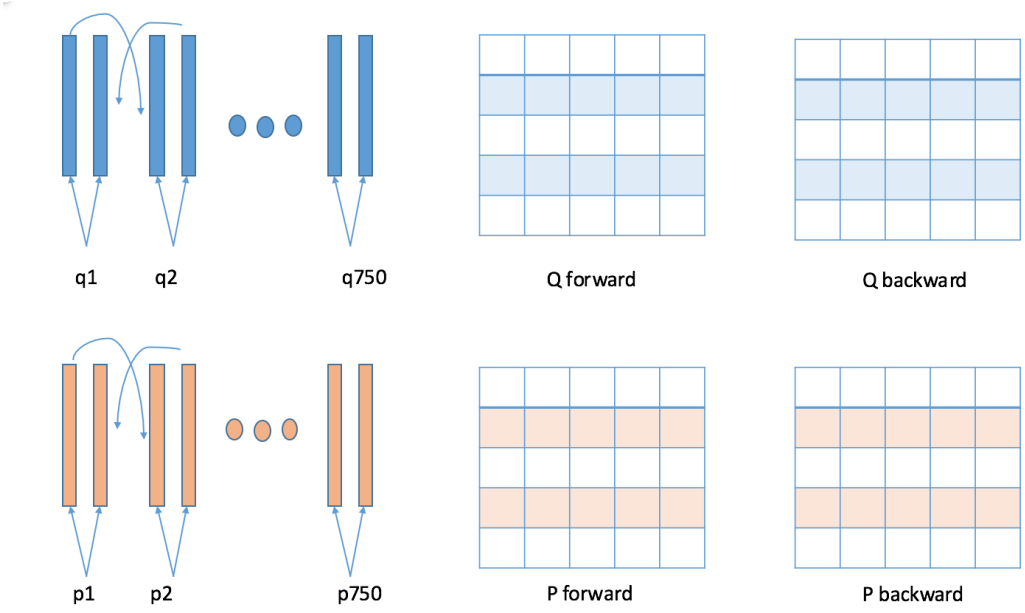

Figure 2: Encoder details. It uses a bi-LSTM which encodes the question first. The final state of the question encoding is fed to the encoder which encodes the paragraph

bi-LSTM Encoder:

We use pre-trained word vectors, GloVe [2], with each word vector having a dimension of 100. Let N be the number of words in a question and paragraph. Let H be the size of the hidden state.

We first use a bi-LSTM cell of hidden state size H to encode the question into two matrices $Q^{H\times N}_{forward}$ and $Q^{H\times N}_{backward}$. Let $q_{forward}$ and $q_{backward}$ respectively be the final hidden states. We feed this to the encoder which now encodes the context paragraph. Let H be the size of the hidden state, same as that for the question. This encoder encodes the paragraph into two matrices $P^{H\times N}_{forward}$ and $P^{H\times N}_{backward}$.

The encoder is illustrated in figure 2. Any attention mechanism built on top of this should make use of these four matrices.

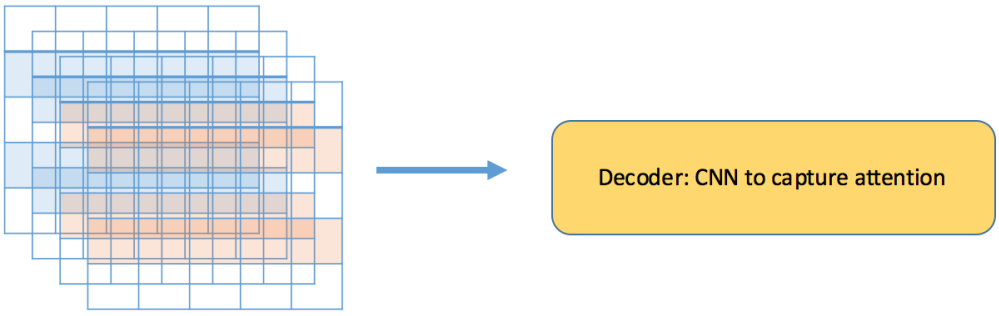

Figure 3: Block diagram of the decoder. It uses all the hidden states from both the question and paragraph encodings. It uses a convolutional neural network to implement the attention mechanism

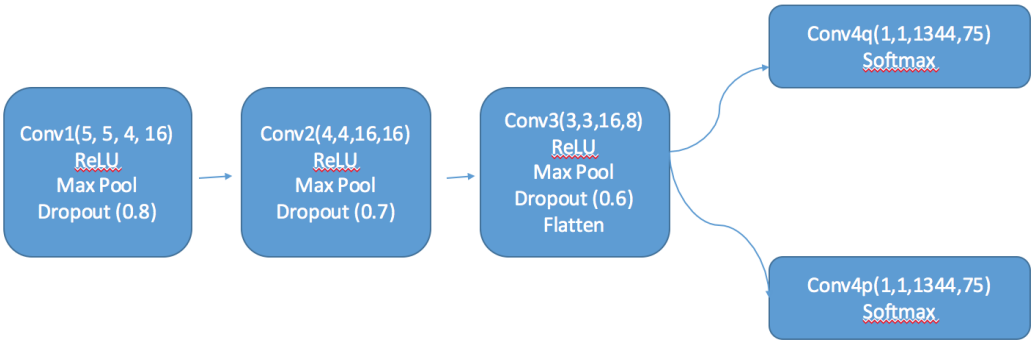

Figure 4: Decoder details. I use a fully convolutional neural network made up of 4 convolutional layers with ReLU non-linearities, max pooling and dropout.

CNN Decoder:

The four matrices obtained from the encoder is concatenated and fed as a four channel input to the convolutional neural network as depicted in figure 3. The details of the CNN is illustrated in figure 4. It consists of three convolutional layers followed by ReLU non-linearities and dropout. The kernel size decreases as we go deeper into the network. This was done owing to the fact that the receptive field increases with layer depth. We use dropout for regularization and decrease the dropout rates as we go deeper into the network. We also use a fully convolutional network with the hopes of reducing the number of parameters in the CNN.

In order to further reduce the number of parameters and to avoid over-fitting, we share the network for start and end index predictions up-to the third convolution as depicted in figure 4.

5 Results and Discussions

I managed to train the CNN for Attention model over 20 epochs and recorded the training loss. It kept decreasing over all 20 epochs, suggesting that the model has enough capacity to train more on the training data. Even the performance on the test set increased over the 20 epochs, suggesting that we have not yet overfit on the training set.

Here is how the F1 and EM scores changed as a function of number of epochs of training:

Type	Filter size	Stride	Output size	Number of parameters
Convolution	5×5	2	$375 \times 100 \times 16$	1616
ReLU	–	–	$375 \times 100 \times 16$	–
Max pool	2×2	2	$188 \times 50 \times 16$	–
Dropout	0.8	–	$188 \times 50 \times 16$	–
Convolution	4×4	2	$94 \times 25 \times 16$	4112
ReLU	–	–	$94 \times 25 \times 16$	–
Max pool	2×2	2	$47 \times 13 \times 16$	–
Dropout	0.7	–	$47 \times 13 \times 16$	–
Convolution	3×3	1	$47 \times 13 \times 8$	1160
Max pool	2×2	2	$24 \times 7 \times 8$	–
Dropout	0.6	–	$24 \times 7 \times 8$	–
Convolution	1×1	1	$1 \times 1 \times 750$	1,008,750

Table 2: Details of CNN. It has about 1M parameters.

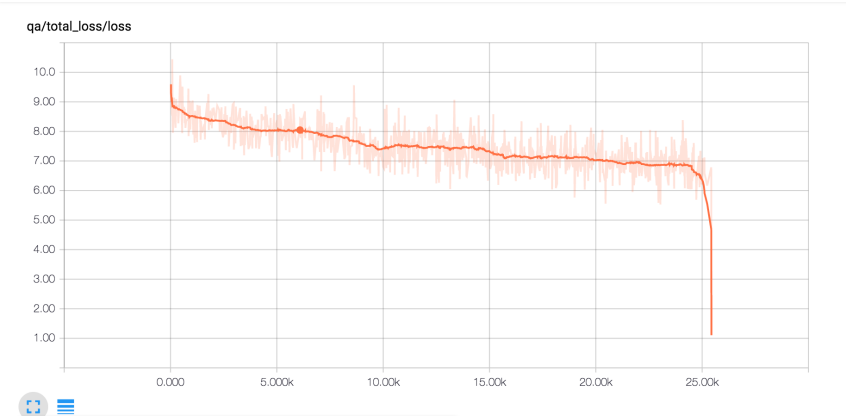

Figure 5: Loss as a function of the number of iterations. I only plot it for training over first 10 epochs

5.1 Discussions:

It can be seen that adding attention to the model performs better than the baseline without any attention. However, the performance of the CNN for Attention model is not as good as we expected it to be. There are many possible reasons for this:

- Since the model can potentially train for more number of epochs and the performance can potentially increase, it is yet to be seen how the model actually performs finally, where the learning gets stalled.
- The start and end tokens are predicted independently, while we know that end token must always be greater than the start token. If we can model this, then the model might perform better.

Number of training epochs	F1 score	EM score
1	0.091	0.000
5	1.678	0.109
10	2.675	0.153
15	3.181	0.170
20	fill	fill

Table 3: Performance of CNN for Attention model over number of training epochs

- The average length of each context paragraph is much smaller than the number of indices we use (750). If we had reduced this to around 400 and had trained the system, it might have performed better, since the algorithm only has to model 400×400 possible outcomes rather than 750×750 outcomes.

References

[1] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh. VQA: visual question answering. *CoRR*, abs/1505.00468, 2015. URL `http://arxiv.org/abs/1505.00468`.

[2] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation.

[3] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016. URL `http://arxiv.org/abs/1606.05250`.

[4] M. J. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016. URL `http://arxiv.org/abs/1611.01603`.

[5] S. Wang and J. Jiang. Machine comprehension using match-lstm and answer pointer. *CoRR*, abs/1608.07905, 2016. URL `http://arxiv.org/abs/1608.07905`.

[6] J. Weston, S. Chopra, and A. Bordes. Memory networks. *CoRR*, abs/1410.3916, 2014. URL `http://arxiv.org/abs/1410.3916`.