# CS224n Final Project: Question Answering

**Jacob E. Perricone**
Institute for Computational and Mathematical Engineering
Stanford, CA 94305
`jacobp2@stanford.edu`

**Blake M. Jennings**
Institute for Computational and Mathematical Engineering
Stanford, CA 94305
`bmj@stanford.edu`

## Abstract

Machine comprehension, correctly responding to a query given a context paragraph, is a canonical task in Natural Language processing which requires the modeling of complex dependencies between the paragraph and the query. Recurrent deep-neural network models that utilizing attention have been successful in Machine Comprehension. Among other attributes, attention mechanisms allow the network to focus on a particular section of the context paragraph, synthesize it in a smaller feature space, capture temporal dependencies. Recently,[**?**] dynamic attention networks, which make multiple passes over the data to overcome an initial local, but not optimal, minimum. We embark on an attempt of Machine Comprehension on the Stanford Question Answering Dataset, implementing a baseline sequence to sequence model, which we then expand to a dynamic co-attention network outlined in []. Our efforts, though powered by the impetus of inquiry and interest, did not yield spectacular results. We achieved a ..

## 1 Introduction

The task of question answering has become increasingly popular in recent years since it requires both language understanding and global knowledge. Recently, [**?**] released the Stanford Question Answering dataset (SQuAD) which is orders of magnitude larger than previously hand-annotated datasets. The SQuAD dataset is particularly useful since the answer spans are within the context, thereby constraining the answer space significantly. We explore the use of a sequence to sequence attention model and an attempt at [] Dynamic Coattention network, as end-to-end networks for question answering.

The Dynamic Coattention Network (DCN), in theory and in practice, is a far more powerful model than one-pass deep learning models, since it allows the network to take multiple passes in the estimation of the answer span. After literary review, the DCN network, not only appears the most intuitively correct, but also achieves the outstanding results on the leader-board. Our journey to implement a deep-learning model for question answering has proven arduous– more so in the details than in the theory– but fruitful. Although we have strong reason to believe that our model has implementation errors, the underlying basis of those errors is not in our grasp of the theory but rather in the of the pedantry of the tensorflow implementation.

The model first maps the words to word vectors, in our case 100 dimensional GloVE vectors. The questions are padded at a max length of 60 and the context are padded at a max length of 600. The word embeddings of each of our contexts and questions are then used as inputs to an encoder, which collapses the context-question space into a feature representation to be processed by the decoder. The decoder uses the encoded representation to learn the span of the answer within the document. The span is represented by a start index, $s$, and end index $e$.

## 1.1 Encoder

As recommended by the teaching staff, we set out to implement a baseline model initially, which proved to be more challenging than expected given, especially since care needs to be taken to ensure the model is bug free. To introduce some notation, let $(x_1^Q, \ldots, x_n^Q)$ denote the sequence of word vectors corresponding to words in a specific question and let $(x_1^C, \ldots, x_n^C)$ correspond to the words in context. Let $L$ be the dimension of our word vectors. We chose to follow along the word of [**?**] to encode the context and question:

```
Step 1: Encoder
1: The encoder uses an LSTM to encode the context, such that:
   $d_t = \text{LSTM}_{enc}(d_{t+1}, [x_t^D]), D = [d_1, \ldots, d_n, d_\phi] \in \mathbb{R}^{2L \times m+1}$ is the encoded representation of the context and
   the final question column $d_\phi$ is a sentinel vector.

2: We use the same LSTM to encode the question $Q' = [q_1, \ldots, q_m, q_\phi] \in \mathbb{R}^{L \times (n+1)}$. To increase the distance
   between the encoding state of the question and the answer, a nonlinearity is introduced to yield
   $Q = \tanh(W^Q Q' + b^Q) \in \mathbb{R}^{l \times (n+1)}$.

3: The attention mechanism defines affinity scores between all pairs of document words and questions words:
   $L = D^\top Q \in \mathbb{R}^{m+1 \times n+1}$. In order to find the relative affinity, the affinity matrix is normalized row wise to
   produce attention weights for the question, $A^Q$, across the document for each question word, and column wise to
   produce $A^D$ across the question for each document word. Softmax normalization is used. Although the dot
   product provides a decent measure of the distance between the words coattention, a mores sophisticated method
   may yield superior results.

4: The attention contexts of the documents by weighting the document words by their respective attention score.
   $C^Q = DA^Q \in \mathbb{R}^{L \times (n+1)}$.

5: We similarly compute attention contexts of the question for each word of the document and the summaries of the
   previous attention summaries for each word of the document. Mathematically:
   $$C^D = \begin{bmatrix} Q \\ C^Q \end{bmatrix} A^D \in \mathbb{R}^{2L \times (m+1)}$$

6: Finally to capture temporal dependency a BiLSTM is used encode as the question-document representation:
   $$u_t = \text{Bi-LSTM}(u_{t-1}, u_{t+1}, [d_{t+1}, c_t^D]) \in \mathbb{R}^{3L}$$
   $U = [u_1, \ldots, u_m] \in \mathbb{R}^{2L \times m}$
```

In addition to the model introduced by [**?**], we implemented a baseline encoder that ran a BiLSTM over the question, concatenated the two end hidden vectors as a question representation. A second BiLSTM was then used to calculate the context representation, using the question representation as initial states to the network.

### Decoder

We implemented two decoders, one significantly more complex, yet incredibly trickier to asses. The first decoder, uses the context representation generated by the encoder to first predict the start index, and then the end index.

We use exponential masking and the sum of the average cross entropy loss across batches as our loss.

Following [**?**], we attempt to implement a Dynamic decoder, which is almost like a Hidden Markov Model where the state is maintained by a LSTM-based sequence model. The decoder updates its predictions at each step using the context-paragraph representations of the current estimated of the start and end predictions.

```
Step 2: Encoder Baseline
1: The start prediction computes the scores of the start index using the the inner product of the matrix $W_s \in \mathbb{R}^{2L\times 1}$ and the context representation $U$.
       $$S_s = UW_s$$
2: The end prediction maps $U \to U'$, using an LSTM. The idea here passing the encoded representation to an LSTM will allow the model learn which positions of the paragraph are important and thus capture the sequential dependency that the end index must follow the start index.
3: The score for the end index is then computed as
       $$S_e = U'W_e$$
```

```
Step 3: Dynamic Pointing Decoder
1: $s_0, e_0 = 0$, initial start and end guess.
2: For $i = 1, \ldots,$ MAX ITER
    1. Maintain the state with recurrent LSTM.
           $$h_i = \text{LSTM}_{dec}(h_{i-1}, [u_{s_{i-1}}, u_{e_{i-1}}])$$
       where $u_{s_{i-1}}, u_{e_{i-1}}$ are the encoded representation of the previous iteration
    2. Estimate the starting and ending positions of the previous iterations using a highway max out network.
    3. Let $HMN_{start}, HMN_{end}$ correspond to the two Highway Maxout networks used to predict the start and end index. The highway maxout network allows the model to pool across multiple model variations, thereby providing a mechanism to predict the indices. For each word in the document,$u_t$ the Highway Maxout network computes the scores, $\alpha_t, \beta_t$ (start, end), using the current word, the LSTM hidden state, and the previous start/ end estimates' encoding $u_{s_{i-1}}, u_{e_{i-1}}$. Mathematically:
           $$\alpha_t = HMN_{start}(u_t, h_i, u_{s_{i-1}}, u_{e_{i-1}})$$
    4. The starting and ending indices are then computed using $s_i = \text{argmax}_t(\alpha_1, \ldots, \alpha_m)$,
       $e_i = \text{argmax}_t(\alpha_1, \ldots, \alpha_m)$.
```

The decoder, implemented, naively, takes incredibly long time to run, up to 6 hours per epoch, if not properly vectorized. We vectorized the code so that the Maxout network computes the $\alpha_t, t = 1, \ldots, m$, in one pass, which sped up our work significantly. However, the results are not as good as one would expect, indicating that our model may suffer from unknown errors. A iterative model was also constructed, but we did not have time to fully train the network. Overall, the trickiest aspect overall of the task was how to write efficient code, how to mask padded inputs, and how to deal with variable size data. The network minimizes the cumulative softmax cross entropy across the $\alpha_t, \beta_t$ for all iterations.

## 1.2 Results

Much to our dismay, we were unable to train the DCN model for more than two epochs on the full dataset. This is largely due to the fact that we switched from the baseline to a more complicated model without fully testing our results and were riddled with debugging. We were unable to do comprehensive hyperparamter search, but did notice that with a large learning rate, the loss spiked quickly in the first epoch and gradually descended as time went on. The $F_1$ achieved on the train and dev set typically hovered around $4 - 10\%$, with an EM of $2\%$. The baseline model would get stuck predicting either only a single word for a span or one would get stuck at the length of the paragraph. The DCN had better results, but not nearly up to par with the expected performance, indicating that there is most likely an error in our tensor flow, or variable. The weights submitted to test leader board are the results of training the DCN model on a maximum context size to 100, hidden state size of 200, a learning of rate of .075.

Our issues mainly revolved around code implementation details. Although we are both experienced coders, we struggled on debugging the model. The plots below show the loss per iteration and a histogram of the coattention weights.

Figure 1: Histogram of Coattention Weights

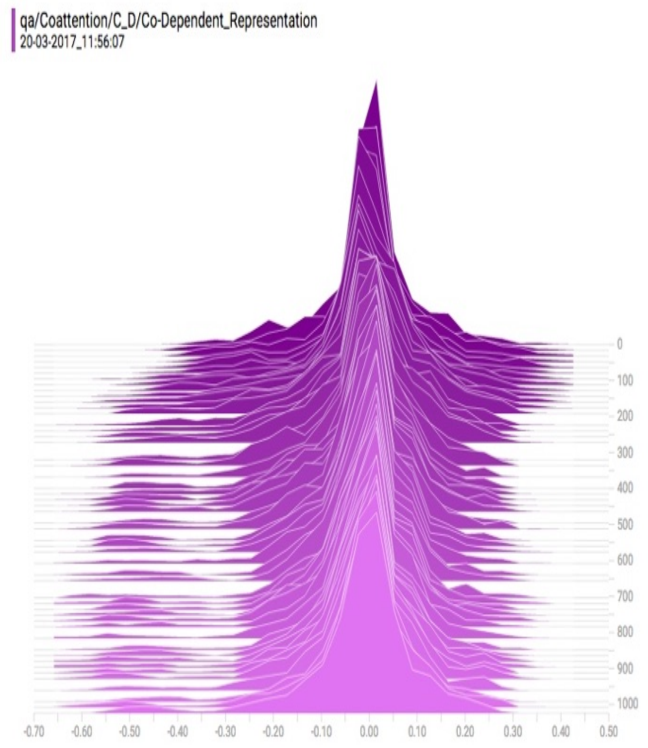
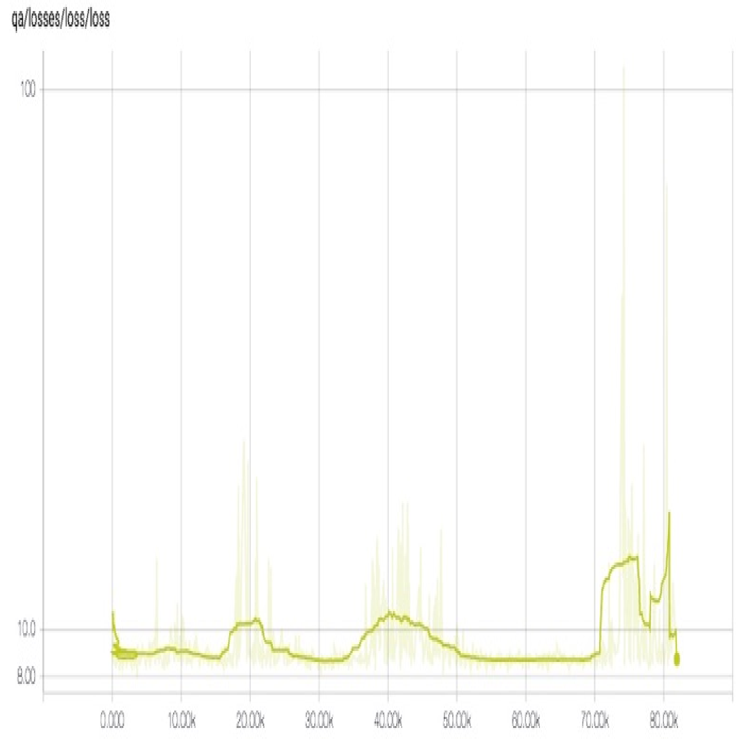


Figure 2: Loss Over Time



### 1.3 Conclusion

### Acknowledgments

Use unnumbered third level headings for the acknowledgments. All acknowledgments go at the end of the paper. Do not include acknowledgments in the anonymized submission, only in the final paper.

### References