Decoding Coattention Encodings for Question Answering

John Clow
Stanford University
`jclow@stanford.edu`

Alex Kolchinski
Stanford University
`yakolch@stanford.edu`

Qandeel Tariq
Stanford University
`qandeel@stanford.edu`

Codalab Username: **`kolchinski`**

Abstract

An encoder-decoder architecture with recurrent neural networks in both the encoder and decoder is a standard approach to the question-answering problem (finding answers to a given question in a piece of text). The Dynamic Coattention[1] encoder is a highly effective encoder for the problem; we evaluated the effectiveness of different decoder when paired with the Dynamic Coattention encoder. We found that models leveraging global attention[2] and locally window-based approaches proved effective, but that a simple decoder combining LSTM neural networks, a linear transformation layer, and a post-decoding length-optimized maximum likelihood layer was the most effective.

Contributions

All three of us contributed substantially to the project. We collaborated in pair/triple programming to build the baseline model, and then specialized more after that. Qandeel did the most research in reading papers/finding models for us to explore and implement, with some help from Alex and John. John and Alex did the most implementation work for post-baseline models, with some help from Qandeel.

Introduction

Question answering is a canonical task in NLP. The nature of the problem is, given a textual question and corresponding context text, to identify the part of the text that comprises the answer to the question.

For example, an example of the context can be: The President was asked by a German reporter about why his White House had cited a Fox News report that claimed that the British surveillance agency GCHQ had been used to wiretap Trump Tower during the election campaign.

Q: Who claimed that the Trump Tower was wiretapped during campaign?

A: A Fox News report

While this problem can be attacked with basic approaches like logistic regression (51% F1 score) , approaches that come closer to human performance generally make use of various neural network architectures [3]. A common theme in such architectures is to use recurrent neural networks (RNNs) in an encoder/decoder pair, with the encoder RNN taking (some function of the) vector representations of the question and context text as input, and producing a sequence of state vectors, which (possibly after some transformation) comprise the embedding. The decoder RNN then takes (some function of) those vectors as inputs and, possibly after some additional transformation, produces the predicted answer spans in the context text.

We sought to investigate the performance of different decoder architectures, by pairing different decoders with the highly effective Dynamic Co-attention Network (DCN) encoder for question answering [1].

Background/Related Work

Various models introduced to approach the question answering problem have had significant success. Most of these experiments have been conducted on a dataset produced by Rajpurkar et al. [3], the Stanford Question Answering Dataset (SQuAD). This contains more than 100,000 question-answer pairs on 500+ articles from Wikipedia. What makes this dataset particularly useful is that the answer to every question in it is a segment of text, or span, from the context which makes it possible to directly predict full answer spans for the models instead of predicting distinct words.

One of the contributing factors of the success of the models introduced to solve this problem has been the use of neural attention mechanisms, which enable the model to focus on the relevant areas in the context paragraph with respect to areas of the question, and vice versa. Attention mechanisms generally work by extracting the relevant information from the context by combining a context vector with the question vector. Then, the attention weights in the model at the current time step are a function of the attended vector at the previous time step. This makes the attention weights temporally dynamic. [4]

One of the most recent works in this domain, Bi-Directional Attention Flow (BiDAF) network, a hierarchical multi-stage architecture for modeling the representations of the context paragraph at different levels of granularity which includes character level embeddings using Convolutional Neural Networks (CNNs), word level embeddings (GloVe representation), contextual embeddings and a bi-directional attention flow mechanism to obtain query-aware representation. [4]

Another approach presented by Lee et al. (2017) [5] presents a model architecture that builds fixed length representations of all spans in the context with a recurrent network. It then scores these explicit span representations and shows better performance than approached that factor the prediction into separate predictions about start words and end words markers. This approach advocated magnifying out of making predictions at that level of granularity and instead predicting spans from context and then scoring them.

Wang et al. (2017) [6] propose an end-to-end neural architecture for the task of machine comprehension of question answer in context, which is based on Match-LSTM, a model used for textual entailment, and Pointer Net, a sequence-to-sequence model to constrain the output tokens from the input sequences. Pointer Network (Vinyals et al. 2015) also employs attention mechanism as a pointer to select a position from the context as an output symbol. Wang et al. use this approach in order to construct answers using tokens from the context.

Wang et al. (2016) [7] uses the SQuAD dataset to propose a Multi-Perspective Context Matching (MPCM) model, which is an end-to-end system that directly predicts the start and end span in the context. They first adjust the word embedding vectors by multiplying relevancy weight computed against question. They encode the question and weighted context representations by using BiLSTMs. It goes over each point in the context and matches the context to the encoded question from multiple perspectives and produces a matching vector which is then passed through another BiLSTM to collect the information and predict beginning and end points in the context.

Approach

The first step in our model encodes the question and context tokens. We map each word in the two categories into a high-dimensional vector space. For this purpose, we use pre-trained word vectors, Global Vectors for Word Representations to obtain the fixed word embedding for each word. Using an LSTM, we then encode temporal information into the context and then compute the question representation using the same LSTM. For the context embeddings, we also append a sentinel vector to allow the model to not attend to any particular word in the input. We also introduce a simple neural network layer with a tanh nonlinearity on top of the question encoding in order to allow the system to transform the question encodings into the context encoding space.

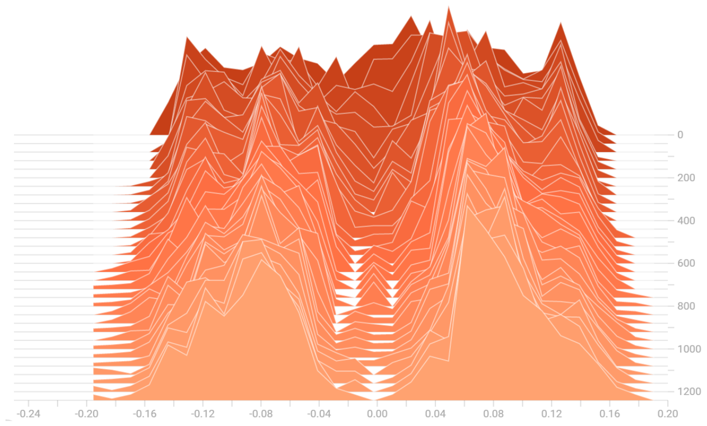


Figure 1: Weights for the start token final Naive layer

The next step is to combine question and context encodings through a coattention layer. For this purpose, we first build an unnormalized affinity matrix by multiplying the context and the question encodings which gives us a measure of similarity between the two. We then add normalization in two different dimensions: first, the affinity matrix is normalized row-wise, i.e. for each context position, weights for corresponding question positions are calculated through a softmax operation; second, the affinity matrix is normalized column-wise, i.e. for each question position, weights for corresponding context positions are calculated. After this, the attention contexts of the context paragraphs are calculated with respect to each word in the question. In other words, for each question index, a weighted sum of context word representations are calculated, weighted by the attention paid to that.

Once the attention contexts of the context paragraph with respect to each word in the question and the question with respect to each word in the context have been calculated, the last step in the encoder is to merge the temporal information with the coattention context by using a bidirectional LSTM. The output of this LSTM, referred to as the coattention encoding, then provides in input for our decoder that helps establish the span (start to end index) that can be the potential answers.

Decoder

The final step of our model is the decoder. We tried three different architectures for the decoder.

Naive Decoder

The first and simplest (which we called the naive decoder) uses two separate pipelines to predict the start position of the answer and the end position of the answer. Both pipelines take the output of the encoder and run it through an LSTM, and then run the outputs of that LSTM through a linear layer, such that we take the dot product of each time step's output vector with a vector of weights. The resulting scalar values, one per context index, are taken as logit inputs into a softmax function, which generates the discrete probability distribution that a given index is the start/end position.

Global Decoder

A more sophisticated decoder which we also tried used all of the same steps as the naive decoder, but with the addition of a global attention mechanism [2]. The start-position decoder functions exactly as in the naive decoder, but now the outputs of the end LSTM are taken through an attention function which calculates a weighted attention-ified average of start LSTM outputs. This average is then combined with the output of the end LSTM for the time step in question, and then that combined vector is used as the input to the linear layer, instead of just the end LSTMs output.

Window Decoder

Our final decoder was a window-based model. This model also took the outputs of the naive decoder, but now the final layer was a single window vector, which took a linear combination of both start and end output probabilities around a target index to calculate the start or logit probability for that index. (One vector each for start and end probabilities)

We also experimented with a post-decoder layer, which picked out the most likely pair of (start, end) indices based on the probability output by the decoder, and weighted by inverse answer length to discourage overly long answers.

Results

Scoring

We score the resulting predictions of our models against the development set by calculating the F1 score and the Exact Match score. The F1 score can be interpreted as a weighted average of the recall and precision, both of which allow us to see how our model is able to retrieve relevant answers and what fraction of the answers retrieved was relevant. The calculation of F1 score is described as follows:

F1 Scoring

$$Precision = P = \frac{True\ Positives}{True\ Positives + False\ Positives}$$
$$Recall = R = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$
$$F1 = 2 \cdot \frac{P \cdot R}{P + R}$$

Exact match score calculates how our predictions fare against the answers given by human annotators and what percentage of our predictions was an exact match with them. However, this score can be slightly unreliable if one takes the inaccuracy of responses produced by the annotators.

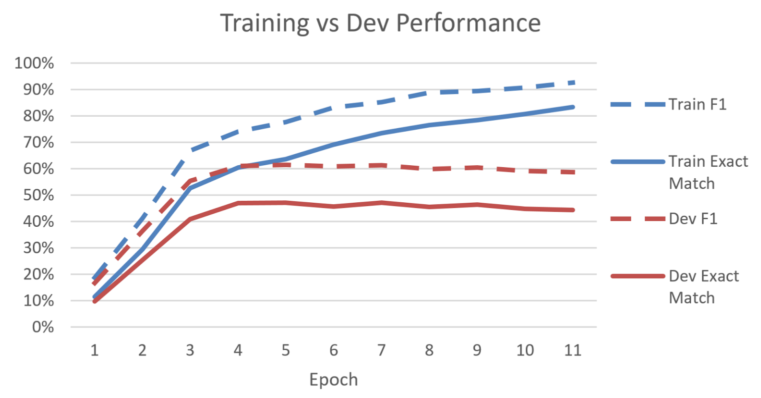
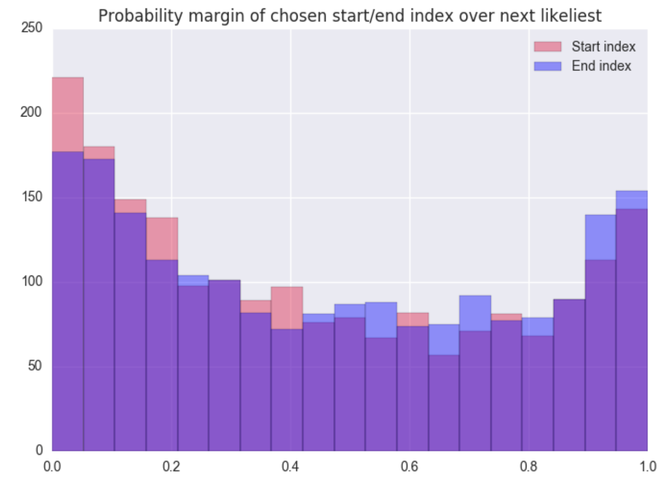


Figure 2: F1 and EM scores for our 300 state size Naive Decoder. It plateus at an F1 of 61%



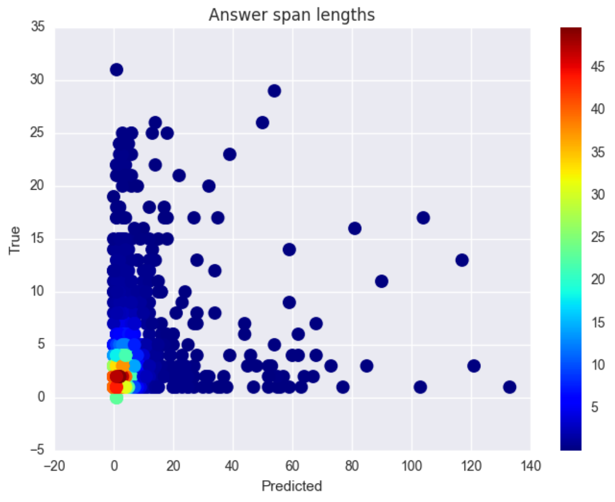


Figure 3: Question vs Answer Spans Lengths. It can be observed that the predictor has a large tail with respect to its span length.

0.1 Decoder Experiments

Analysis

Analysis of Model Performance

Going over our models results, following are some insights we were able to gather from them: 1. Among the partially correct, most have the correct answer as a subset within them and have unnecessary information attached to the answer such as the, title of the person. E.g.:

Our Answer: Grandmaster Boris Gelfand

Correct Answer: Boris Gelfand

2. Some of them have long sequences attached to them which have the correct answer in them but the span is too long and contains irrelevant information. E.g.

Table 1: Decoder Experiment Performances

Decoder Type	State Size	Dropout	F1 (local)
Naive Decoder (no Sentinel)	100	15%	52%
Naive Decoder (no Sentinel)	200	15%	60.5%
Naive Decoder	200	30%	59.8%
Naive Decoder	300	30%	64% , 61.8% *
Naive Decoder with span restriction	300	30%	64.0% *
Global Attention Decoder	200	30%	61%
Global Attention Decoder	300	30%	59%
Window Decoder	200	30%	58%

Our Answer: 23,597 students in fall 2014 ; an additional 3,371 students were enrolled at the KU Medical Center

Correct Answer: 3,371

3. At some points, our system could not understand repetition of information. If the correct answer appeared twice in the context, or system predicted the whole span, starting from the first correct answer to the second correct answer. E.g.

Our Answer: October 2001 , U.S. forces (with UK and coalition allies) invaded Afghanistan to oust the Taliban regime . On 7 October 2001

Correct Answer: 7 October 2001

4. In some cases, our model was in the relevant part of the context to get to the correct answer, but failed to filter down to the correct answer. For example:

Our Answer: Brains are most simply compared in terms of their size . The relationship between brain size , body size and other variables has been studied across a wide range of vertebrate species . As a rule , brain size increases with body size , but not in a simple linear proportion . In general , smaller animals tend to have larger brains , measured as a fraction of body size . For mammals , the relationship between brain volume and body mass essentially follows a power law with an exponent of about 0.75

Correct Answer: larger

5. Our system was able to make a prediction for every question and there was no question for which an answer span was not predicted. However, oftentimes, because it is unaware of the separator tokens, it predicts just a separator token as the result.

6. 49 out of 152 questions were not answered correctly at all and quoted completely irrelevant parts of the context to predict answers. However, some of these were questions which could have had alternative answers. For example:

Question: What is another word to describe a Jew? Our Answer: yahd (sg .) , al-yahd (pl .) , and ban isrl in Arabic , " Jude " in German , " judeu " in Portuguese , " juif " in French , " jde " in Danish and Norwegian , " judo " in Spanish , " jood

Correct Answer: Ebreo

In this case, the answer predicted by our system is also correct, however it could not predict the second possible answer, perhaps due to span restrictions.

Reflection

Firstly, we have learned that it take a lot of iteration and extensive testing to build a model that performs well. Our initial models did not work at all, and we needed to reconsider many of our design choices to get even a decent performance. Also, intuition is useful, but deep learning is not intuitive. We would think that adding a window or global attention layer would make the decode layer better. However, we failed to account for the problems of overfitiing, and failed to notice that the BiLSTMs already do the local/global information tranformations that we were looking for in those models.

We were able to diagnose some of our problems using the result tensors we collected from Tensorflow. However, we could have utilized Tensorboard and other debugging tools to a higher extent earlier in the development process as it would've helped us notice places where we were getting ridiculous values such as for our losses when we were still predicting tokens. Also, looking at our prediction data would have similarly helped us with improving our model. Late in the development process, we realized that we were predicting some absurdly long spans as well as negative length spans. This allowed us to create a very simplistic but quite effective filter to only allow plausable spans, which in turn increased our F1 score by 2%. Had we looked at our prediction results earlier, we would have been able to make a more sophisticated model to accomodate for that.

Future Improvements

Some future improvements we can make are add a convolutional layer that we could then run coattention on top of. Additionally, we could work on creating better decoders (ones that take into account the plasuability of words between the start and end tokens).

References

[1] Xiong, C., Zhong, V.& Socher, R. (2016). Dynamic Coattention Networks For Question Answering. *arXiv preprint arXiv:1611.01604.*

[2] Luong, M. T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025.*

[3] Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250.*

[4] Seo, M., Kembhavi, A., Farhadi, A., & Hajishirzi, H (2016) Bidirectional Attention Flow for Machine Comprehension. *arXiv preprint arXiv:1611.01603.*

[5] Lee, K., Kwiatkowski, T., Parikh, A., & Das, D. (2016). Learning Recurrent Span Representations for Extractive Question Answering. *arXiv preprint arXiv:1611.01436.*

[6] Wang, Z., Mi, H., Hamza, W., & Florian, R. (2017). Multi-Perspective Context Matching for Machine Comprehension. *arXiv preprint arXiv:1612.04211.*

[7]Wang, S., & Jiang, J. (2016). Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905*

[8] Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global Vectors for Word Representation. In EMNLP (Vol. 14, pp. 1532-1543).

[9] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron C Courville, and Yoshua Bengio. Max- out networks. ICML (3), 28:13191327, 2013.