Stanford CS224N Winter 2017 Final Project Reading Comprehension on SQuAD Dataset

SUNet IDs:	adeclerc & krake
Names:	Andrew DeClerck & Kevin Rakestraw
Codalab Group:	cs224n-krakeadeclerc
Codalab Name:	krake

By turning in this assignment, we agree by the Stanford Honor Code and declare that all of this is our own work.

1. Overview

In this project we seek to explore NLP and Deep Learning with what we have learned and used throughout the quarter. In this project we tackled Reading Comprehension using Stanfords recently published Stanford Question Answering Dataset (SQuAD).

The SQuAD dataset is comprised of three elements: a question, a context, and an answer. The question, represented by a string, is an open-ended question whose answer can be found within the context paragraph. The context, again represented by a string, is a paragraph that presents various information about a topic of which the question will target a subsection of. The answer, which can be represented as the answer substring or start and end word indicator for the context, is the result that we seek to predict with our task.

Our tried two different baseline models, and variations of each model. We were able to demonstrate improvement over training, however each model was limited in their own way due to oversimplification of the problem, and were both ultimately capped at a very low accuracy of prediction. Throughout our project we developed a strong understanding for the weaknesses of our models, and, vice versa, strengths in other models proposed in research papers. Unfortuneately, due to our time constraints we were unable to implement further comprehensive techniques on top of our baseline. Despite low performance we are very proud of our ability to tackle a real world problem, develop experience using a widely used ML/NLP tool in tensorflow, and gain an understanding of the leading research in this field.

2. SQuAD Dataset

As touched on by the overview, the SQuAD dataset is a dataset that contains question-context-answer pairs. An example pairing is as follows.

Question: Which company is directly above Universal Studios?

Context: Universal Studios Inc. (also known as Universal Pictures) is an American film studio, owned by Comcast through its wholly owned subsidiary NBCUniversal, and is one of Hollywood's "Big Six" film studios. Its production studios are at 100 Universal City Plaza Drive in Universal City, California. Distribution and other corporate offices are in New York City. Universal Studios is a member of the Motion Picture Association of America (MPAA). Universal was founded in 1912 by the German Carl Laemmle (pronounced "LEM-lee"), Mark Dintenfass, Charles O. Baumann, Adam Kessel, Pat Powers, William Swanson , David Horsley, Robert H. Cochrane, and Jules Brulatour.

Answer: NBCUniversal

In order to make our problem easier to solve, we use data not in this format, but rather a simpler format. For this we want to take the text and map it to an index in a vocabulary of words. However, one major change is with the answer. As the problem defines the answer as a substring of the context paragraph, we do not need to know the words contained, but rather just the start and end index of its placement within the context. The following is the same SQuAD pairing with our simpler format.

Question: 253 315 12 862 686 1483 4715 18
Context: 1483 4715 5215 17 41 86 15 1483 4700 16 12 33 121 286 1839 4 1591 20 1813 111 42 8807 1591 3986 9557 4 7 12 50 5 3620 23 11 2975 7135 11 286 5747 6 980 382 5747 26 31 621 1483 169 4772 5899 8 1483 169 4 1609 6 12457 7 47 3887 1946 26 8 84 231 169 6 1483 4715 12 10 653 5 3 10148 5736 974 5 309 17 22761 16 6 1483 13 439 8 3668 20 3 199 4288 7688 17 4093 11 43358 11 16 4 3131 28923 4 787 12894 24727 4 3775 28691 4 11784 5068 4 627 28590 4 1256 28648 4 1190 3576 22913 4 7 14308 28564 6
Answer: 24 24

It is important to keep in mind that glove embeddings are not the only way to represent the text, and should be by no means the only representation. We wanted to incorporate parsing or other types of word classifications. We were impressed by the IBM Watson team's Chunk Extraction and Ranking[1], and wanted to incorporate part-of-speech on top of the glove embeddings, and have our output be determined by the two separately.

Anyways, we ultimately decided it would be best to avoid complicating our model, and focus on getting something up and running first. While we have an understanding of what our input data will look like format-wise, it would be very useful to have some more knowledge about our Training set and Val set.

Metric (Length)	Train-Questions	Train-Context	Val-Question	Val-Context
Total Number	81381	81381	4284	4284
Min	1	22	4	78
Max	60	766	30	499
Mean	11.3	137.5	10.8	142.1
Std Dev	3.7	56.9	3.3	50.1
Median	11	126	10	129

SQuAD Question and Context Statistics

Metric (Value)	Train-Ans-Start	Train-Ans-End	Val-Ans-Start	Val-Ans-End
Total Number	81381	81381	4284	4284
Min	0	0	0	0
Max	605	605	331	332
Mean	57.8	60.3	60.0	61.9
Std Dev	48.7	49.0	46.7	46.8
Median	47	50	52	54

SQuAD Answer Span Statistics

We analyzed the length of each question and context within the train and val sets. The vast majority of questions and contexts are much shorter than their maximum lengths. We also saw that our answer ranges were also even shorter than the average context length. By knowing this we could trim down our representation of the context, and essentially ignore the ends of longs contexts, as we would still be able to perform well all the majority of contexts that are shorter. However, for our purposes, and given our models (we used baseline models that do not require much computation compared to more advanced models) the

difference in speed was not large enough to convince us. Instead we prioritized a complete knowledge of context and questions, so we made our input sizes for questions and context based on the maximum question and context sizes respectively. We also made use of masks, so that padding did not factor into our encoding of both questions and contexts. While our output predictions are still unavoidably represented by the a probability across the max context size, this is not an issue with our later baseline, as it is based on the dot products.

3. Our Implementation

For our model, we ended up trying variations of two different concepts. Our first attempts, which we later revisited, were based with a GRU Attention Model. We later tried variations of a model based with a LSTM encoding with the Question final state being incorporated into each state of the context to make a prediction.

Our goal with each model is to extract enough information from our question and context to make a reasonable prediction. In both models, we don't do anything to limit the inputs of the problem, outside of limiting the length of the question and context placeholder variables (However we also have ensured that nothing we will train on will be too long). We do end up oversimplifying our problem, as discussed in the Reflection sections of each model. Due to time constraints, we were forced to remain with simpler models that account solely on the word embeddings of the question and context. We wanted to, but were unable to, explore parts-of-speech to provide more information than glove word embeddings, as well as improving our context and question analysis with a Match-LSTM like model as shown in S. Wang & J. Jiang's research[2].

For both models, we set up the model to take in batches of question-context-answer data for training and validation, and batches of question-context pairs if our goal is strictly to predict. Upon receiving the data, we make sure to pad the question and context to their respective max sizes as well as create a mask for each. After we read in our batches of this preprocessed data, we can add the glove embeddings to our question and context and use our model to generate an output.

For this we broke up our problem into sub-problems encoding and decoding as outlined to us by the initial model skeleton given by the cs224n staff. For each of distinct models we do the encodings and Decoding differently but the ideas are the same in both. In our encoding we want to process the question and context embeddings, to gain an understanding of each element as a whole. In our decoding we want to take what we learned through the Encoding step and utilize it to make our prediction.

Each of our implementations performed roughly around the same. We evaluated F1 and Exact Match scores, after both training and validation, for each of the two datasets. We saw roughly the same ranges of scores being produced from epoch to epoch for all of our models. For each evaluation we did, we expanded the evaluation to 250 pairs that were randomly selected from the train or dev set. The highest F1 scores were around 4% while the highest EM score was 1.2% (3 out of the 250 yielding an exact match).

There isn't evidence to say either model outperformed the other, as they were roughly the same in results, although it's safe to say that neither does a particularly good job of prediction.

3. Baseline Model: GRU Attention

Implementation

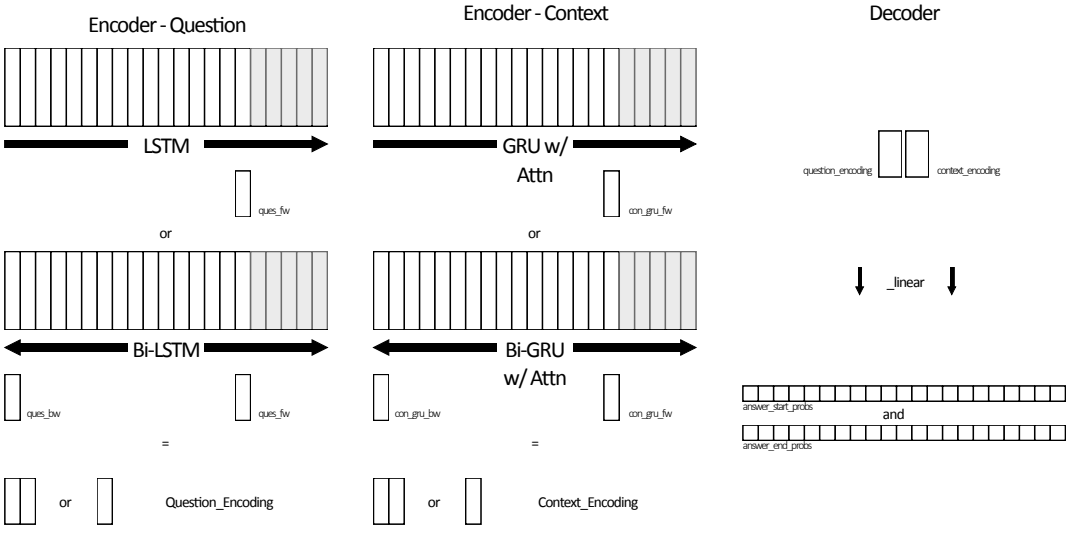
Our first attempt at implementing a model came with the GRU attention model. With this model, we broke our encoding down into two steps. For encoding our question we performed a straight-forward LSTM and used the final state(s) as our question encoding. Next we used a GRU RNN with attention based on the Question Encoding from before. This provides us a way to encode our context in respect to the question. We then used the final state(s) of the GRU as our context encoding. We then combined our context encoding and question encoding and created used a linear function followed by a sigmoid in order to output probabilities of each index in the output size being the start or end indexes of the answer. Note

this is not a true probabilistic distribution, as the final vector does not need add up to 1, however for our purposes it represents the likelihood of the answer span.

We tried two variations of this problem and doing single directional RNNs as well as bi-directional RNNS, and both are reflected in the figure. This variation does not vary the encoding or decoding much, however it does involve reflecting the different sizes in code.

One-Directional
1) Read the question Embeddings into a LSTM and use the final state as the question encoding.
2) Read the context embeddings into a GRU RNN and use the question encoding with the GRU Attention Cell to apply our knowledge of the question to the context. Take the final state and use this as the context encoding.
3) Use the concatenation of these encodings in a simple neural net layer consisting of a linear function within a sigmoid function. The output is a vector of our context size that represents the likelihood of answer span indices.

Bi-Directional
1) Read the question embeddings into a Bi-Directional LSTM and concatenate the final states as the question encoding.
2) Read the context embeddings into a bi-directional GRU RNN and use the question encoding with the GRU Attention Cell, to apply our knowledge of the Question to the context. Take the final state and use this as the context encoding.
3) Use the concatenation of these encoding in a simple neural net layer consisting of a linear function within a sigmoid function. The output is a vector of our context size that represents the likelihood of answer span indices.



Visual Breakdown of the GRU Encoding Decoding Steps

Results

We found that this model did a poor job of solving our problem, however showed better results than our second form of model. This model proved very time-demanding, having Epoch training times of about 7 hours for the one-directional and training time of about 10 hours for the 81k train set data tuples. We also found that it would approximate its minimum loss value of around 10 within the first epoch. After this the resulting evaluation would vary (likely due to sampling different data), but our average loss would not. We achieved the following maximum F1 and Exact-Match Scores from this model. The disparity between our dev score and the CodaLab scores was caused by human error in using the id files and converting back to text which caused all the words too rare to be covered by the vocabulary to be called "¡UNK¿" for unknown. We were unable to retest, as by the time we discovered this issue, we were unable to train our model any considerable amount.

Max Scores (Dev)	F1 Score	Exact-Match Score
GRU	**3.85%**	**0.8%**
Bi-GRU	**3.50%**	0.4%
	2.82%	**1.2%**

Best Coda Lab Dev Score – F1: 1.386% – EM: 0%

Coda Lab Test Score – F1: 1.38% – EM: 0%

Reflection

The shortcomings from this model stem from its oversimplification of the problem inputs. Encoding the Question into the final hidden state(s) of a LSTM run is not a terrible way to represent the Question. In this we are taking what averages to be around a 10 word question and representing it as one or two 200-size state vectors. However its major shortcoming comes from the Encoding of the Context. We encode what averages to be a 137 word context into one or two 200-size state vectors, with the goal of outputting two indices within an output range of 766. We don't allow ourselves to encapsulate enough information about the context to provide a strong indicator of answer span.

An improvement could be made to our model, if instead of using the the final state(s) output for our decoding, we utilized the full output of the GRU RNN run. Performing a linear function on the output would likely be too memory-demanding as it would require nearly 1000x the memory for the parameters to support the matrix multiplication. However, we could incorporate what we did in our LSTM batch multiplication model, and multiply each state in the context output by a new variable of size (state-size, 1) to reduce the RNN output to a likelihood distribution.

We limit our ability to understand other ways an answer can relate to a question.

4. Baseline Model: LSTM Batch Multiplication

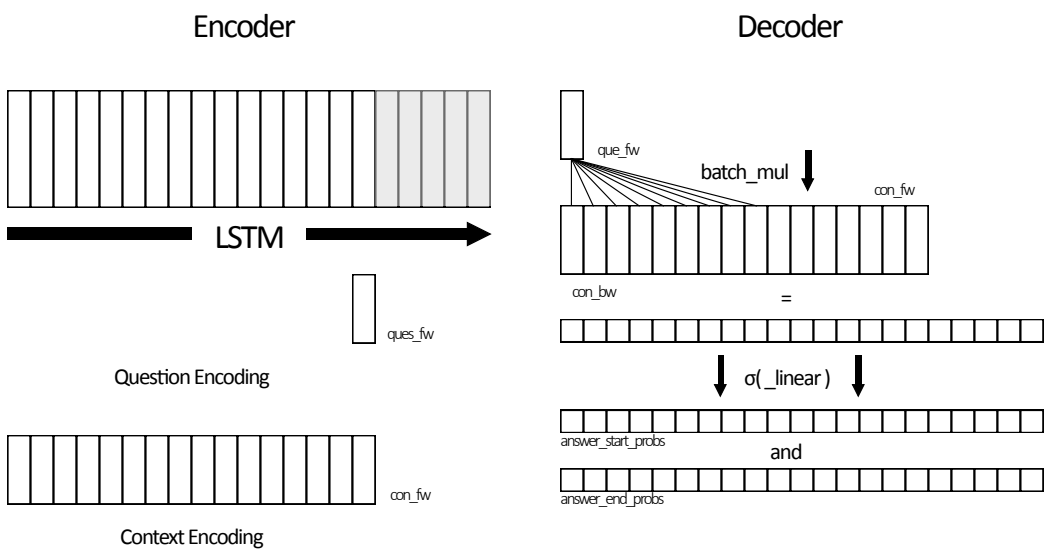
Implementation

Our second attempt at implementing a model came with a LSTM multiplication model. With this model, we broke our encoding down into two nearly identical steps. For encoding our question we performed a straight-forward LSTM and used the final state(s) as our question encoding. For encoding our context we again performed a straight-forward LSTM and used the entire output as our context encoding. We then matrix multiplied our question encoding by each state of the context encoding to get an Output-sized vector of scalar values that roughly represents our answer indices. We take this Vector output from the multiplication and use two separate straight-forward neural nets to output the probabilities of the span start

and end respectively. Again, this is not a true probability as it is not guaranteed to sum to 1, but for our purposes indicates the likelihood of certain indices being the start or end indices.

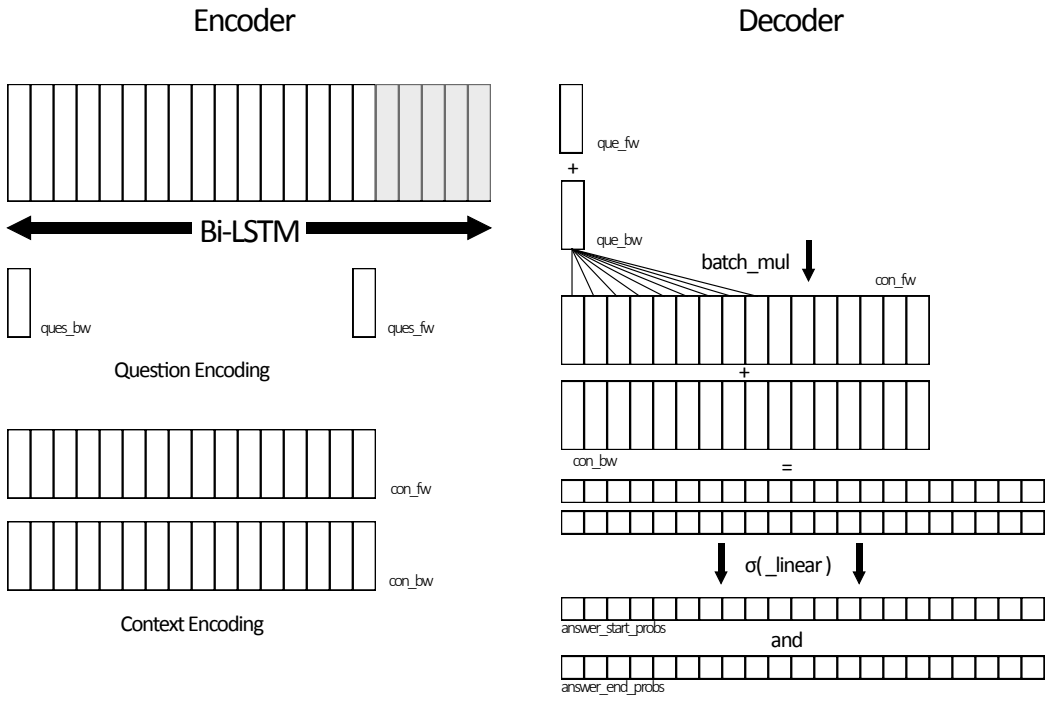
We tried two variations of this model as we did for the last model, again by varying use of single directional RNNs and bi-directional RNNS. For the bi-directional version, we used combinations of intermediate representations to come to our predictions. We did this, as we felt this will increase the information available for our prediction.

One-Directional
1) Read the question embeddings into a LSTM RNN and use the final state as the question encoding.
2) Read the context embeddings into a LSTM RNN and use the full output as the context encoding.
3) Multiply the question encoding into each state of the context embeddings, resulting in a vector of scalars.
4) Use simple neural nets consisting of a linear function and a sigmoid function on this vector to output our answer start and end probabilities.



Visual Breakdown of the LSTM Multiplication Encoding Decoding Steps

Bi-Directional
1) Read the question embeddings into a bi-directional LSTM RNN and use the two final states as the question encoding.
2) Read the context embeddings into a bi-directional LSTM RNN and use the two full outputs as the context encoding.
3) Multiply the question encodings into each state of the two context encodings separating the combined results of the two context encodings. This results in two vectors of scalars that are based on combinations of the two question encodings.
4) For each vector use separate simple neural nets consisting of a linear function and a sigmoid function to output our answer start and end probabilities. Combine the two answer start probabilities and the two end probabilities as the our final probabilities for both.



Visual Breakdown of the Bi-Directional LSTM Multiplication Encoding Decoding Steps

Results

We found that this model also did a poor job of solving our problem, yet showed slightly better results than our first model. This Model was also much quicker, having epoch training times of about 3.5 hours for each variation for the 81k Train set data tuples. One interesting thing with our results is that the loss was significantly higher than our first model, staying around 13 compared to our first model's 10. However, the loss is not our concern as it does not equate to how well we are predicting but is more of a indicator of learning. Again evaluation varied through our epochs (likely due to our sampling different data), but our average loss remained roughly the same. We achieved the following maximum F1 and Exact-Match Scores from this model. The initial disparity between our dev score and the CodaLab scores was caused by human error in using the id files and converting back to text which caused all the words too rare to be covered by the vocabulary to be called "¡UNK¿" for unknown. We believe we were able to resolve the issue, however we were unable to fully finish train our model.

Max Scores (Dev)	F1 Score	Exact-Match Score
LSTM	**4.05%**	**1.2%**
Bi-LSTM	**5.18%**	**2.0%**

Best Coda Lab Dev Score – F1: 0.586% – EM: 0%

Coda Lab Test Score – F1: 0.654% – EM: 0%

Reflection

The shortcomings from this model stem from an oversimplification of the problem, limiting the prediction to a dot product similarity. We use the same Question Encoding as in the GRU model, yet we use the entire Context output as the Context Encoding. However, the oversimplification comes more from our decode than from our encode. In our Decode, we output our vector that will be used for our indicies by doing a dot product between each state of the Context Encoding with the question encoding. This limits our results to be based on how similar the overall meaning of the question is to the words in the contexts (granted the RNN makes each state account for more than just that associated word). Our model is crippled by the assumption that the question needs to be similar to the answer.

One potential improvement to this model would be to find a better way to apply this information. If part of a context is similar to the question, then there is a good chance that the answer is preceding or following. Even, if we use an RNN to try to learn what's nearby, we have limited ourselves to scalar values, and so there could not be a determination on a preceding or following answer with any degree of certainty. This model hits a wall with its simplicity, and thus instead of looking to improve this model, we should instead look to create another model.

5. Main Takeaways

Throughout this project we ultimately discovered that the vast majority of our disappointment in our models was due to the fact that we over-simplified the problem. We focused on only one aspect of the problem, and did so at a very basic level. However, there is a lot to take away from our struggles.

1) Reading comprehension is difficult. There are a lot of ways to take on reading comprehension, and limiting yourself to only one or a few ways limits your model's comprehension.

2) What humans do with little effort takes a ton of effort for a computer. It takes millions of parameters and complex calculations to come close to what humans are capable of.

6. Citations

Yang Yu, Wei Zhang, Kazi Hasan, Mo Yu, Bing Xiang, and Bowen Zhou. End-to-End Answer Chunk Extraction and Ranking for Reading Comprehension. 2016.

Shouhang Wang and Jing Jiang. Machine comprehension Using Match-LSTM and answer Pointer. In Proceedings of the International Conference on Language Recognition, 2017.