CS224N: Natural Language Processing with Deep Learning
Reading Comprehension

Wissam Baalbaki
Institute of Computational & Mathematical University
Stanford University
Stanford, CA 94305
baalbaki@stanford.edu

Dan Zylberglejd
Department of Statistics
Stanford University
Stanford, CA 94305
dzylber@stanford.edu

Codalab username: baalbaki

Abstract

Machine Comprehension is a very interesting task in both natural language processing and artificial intelligent research but extremely challenging. The goal is to enable a machine to understand a given passage and then answer questions related to the passage. In out attempt to implement a neural network architecture for Reading Comprehension, we use the published Stanford Question Answering Dataset (SQuAD) for training our model.

1 Introduction

There are several approaches to NLP tasks in general. With recent recent breakthroughs allowed in algorithms (deep learning), hardware (GPUs) and user friendly APIs (Tensorflow), some tasks have became feasible up to a certain accuracy. In this project we try to explore all of those mentioned tools to solve the question-answering problem. We make use of word embeddings (Glove), a seq2seq framework based on encoder and decoder and built with cell structures such as LSTMs.

2 Literature Review

Using the SQuAD dataset, the original paper from Rajpurkar et al. (2016) implemented a linear model with sparse features based on n-grams and part-of-speech tags present in the question and the candidate answer. They also used syntactic information in the form of dependency paths to extract more general features. They set a strong baseline for following work and also presented an in depth analysis, showing that lexical and syntactic features contribute most strongly to their model's performance.
A recent paper by K. Lee et al. proposed a Recurrent Span Representation (RASOR). The point is to develop an expressive model that computes joint representations of every answer span, which can be globally normalized during learning. They aggregated the passage information along with each question and used recurrent computations for shared substructures (i.e. common passage words) from different spans. They utilized both passage-aligned and passage-independent question representations. Thus, the model constructs question-focused passage word embeddings by concatenating: (a) the original passage word embedding, (b) a passage-aligned representation of the question and (c) a passage-independent representation of the question shared across all passage words. They use a BiLSTM over these concatenated embeddings to efficiently recover embedding representations of all possible spans, which are then scored by the final layer of the model.

In another paper in 2016, Wang & Jiang used an end-to-end neural network method that uses a Match-LSTM to model the question and the passage, and extracted the answer span from the passage using pointer networks. That model worked better than feature engineered model in the original paper (i.e. Rajpukar) so we decided to take a closer look at it. They used an LSTM pre-processing layer, a match-LSTM layer and an Answer Pointer Layer:

- The purpose for the LSTM preprocessing layer is to incorporate contextual information into the representation of each token in the passage and the question.

- The match-LSTM model tries to predict the textual entailment. Given two sentences (a premise and a hypothesis), to predict whether the premise entails the hypothesis, the match-LSTM model goes through the tokens of the hypothesis sequentially.
  At each position of the hypothesis, attention mechanism is used to obtain a weighted vector representation of the premise. This weighted premise is then to be combined with a vector representation of the current token of the hypothesis and fed into an LSTM. This match-LSTM essentially aggregates the matching of the attention-weighted premise to each token of the hypothesis and uses the matching result to make a final prediction.

- Pointer Network is usually used to solve a special kind of problems where we want to generate an output sequence whose tokens must come from the input sequence. Instead of picking an output token from a fixed vocabulary, Pointer Network model uses attention mechanism as a pointer to select a position from the input sequence as an output symbol. Here, Wang & Jiang used a Pointer Network Model to construct answers using tokens from the input text. Two different Pointer Network Models work here: the sequence model and the boundary model. The latter worked better and we followed a similar approach for our algorithm

3 Analyzing Training Dataset

SQuAD is comprised of around 100K question-answer pairs, along with a context paragraph. Previous machine comprehension datasets are either too small to train end-to-end deep learning models, or not difficult enough to evaluate the ability of current machine comprehension techniques.Since SQuAD has been released various researchers have hoped that it could help in alleviating these limitations. We looked at some of the papers who have recently utilized SQuAD in order to gain insight about the potential benefits as well as limitations of the dataset. Prior research focused on extracting answers from paragraphs by picking a specific sentence. Using a Neural architecture was successful in that task but obviously picking an exact answer span is a more challenging problem. In most structured prediction problems (e.g. sequence labeling or parsing), the number of possible output structures is exponential in the input length, and computing representations for every candidate is prohibitively expensive. We ran some first inspection on the data, to understand better what type of questions/answers are in there, and the style of the contexts. We also obtained histograms of the sizes of contexts, questions and answers, in terms of number of tokens, as follows:

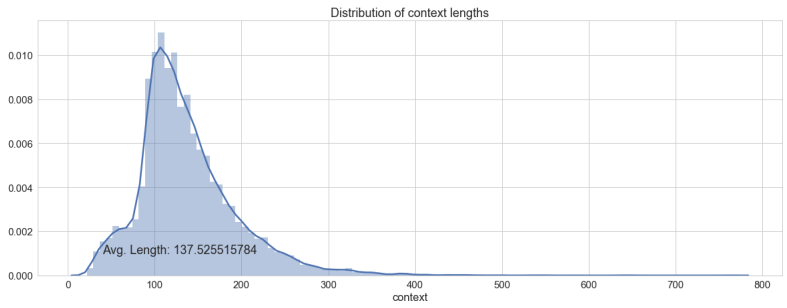

Figure 1: **Histogram of Paragraphs' Length**

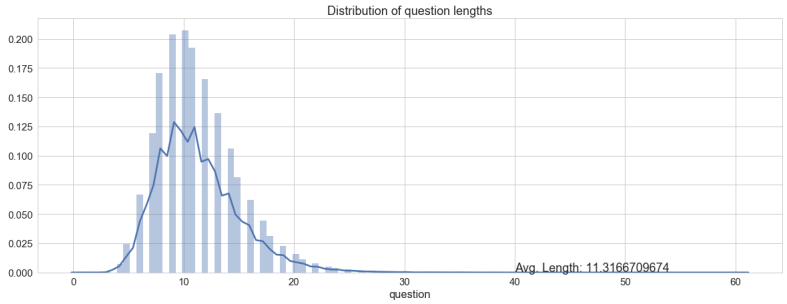

Figure 2: **Histogram of Questions' Length**

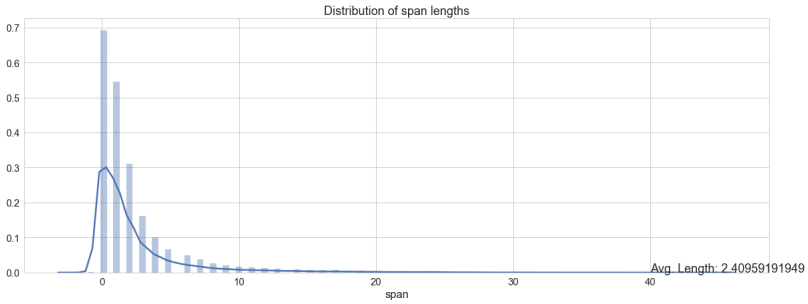

Figure 3: **Histogram of Answers' Length**

In order to get familiar with the dataset, we first run basic statistical analysis of the SQuAD dataset. We did additional analysis looking at the question types and their distribution. Then, we investigated the location of the answer for various question types.

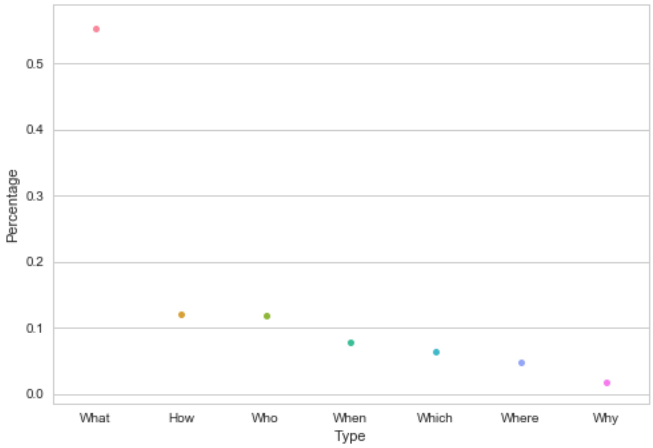

Figure 4: **Questions' Type Distribution**

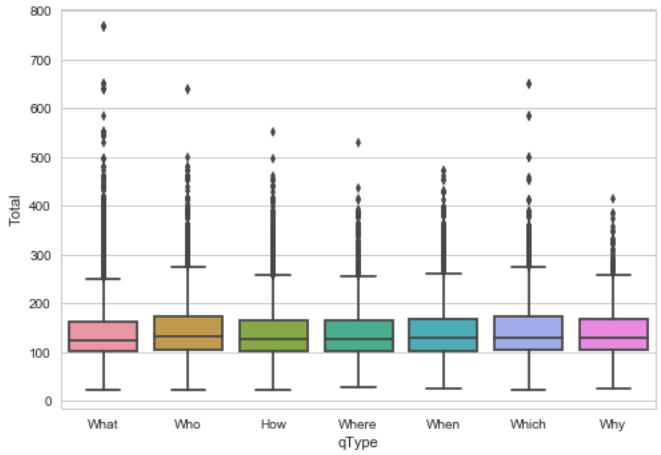

Figure 5: **Question Length per Type**

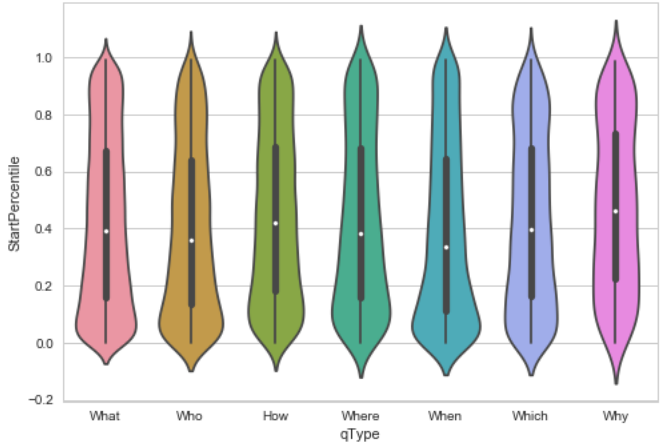

Figure 6: **Starting Index Distribution per Type**

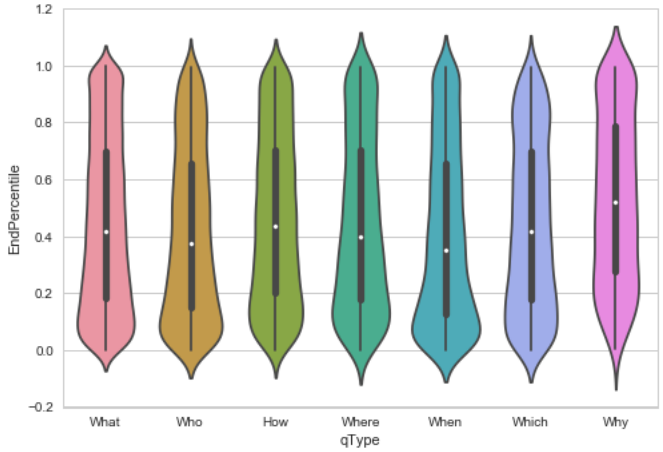

Figure 7: **Ending Index Distribution per Type**

We notice the following:

- The answer can be in arbitrary span within the paragraph
- There is a variety in the types of questions
- The same questions are asked in multiple ways (e.g. a the “What” could appear anywhere in the question , not necessarily as the first word).
- The answers for the questions "Who" & "When" showed up usually earlier in the paragraph than answers for the questions "Where" & "Why". That makes sense since sentences would refer to periods of time more often in the beginning and that is also the natural place for the subject to appear in a sentence, whereas reasoning or explaining about a topic usually comes later in the paragraph.

4 Approach

We take an approach very similar to the one in the Machine Comprehension Using Match-LSTM and Answer Pointer paper (Liang & Wang). We now describe the main settings of our model the intuition and motivation that led us to choose each of them::

- Glove dense representations with 50 dimensions for each word. The choice for 50 (as opposed to 100, 200, etc.) was for computational efficiency and model simplicity. We later tried representations with size 100 without success over the 50 one.
- Mini-batching, with batches of size 100 (which is faster as compared to size 10);
- Adam optimizer, with gradient clipping (max norm size of 5.0) to avoid exploded gradients, and an exponential decay learning rate (starting at 0.005 and decaying at a rate of 0.96 every 100000 steps);
- State size (which will become more clear later) of 50. We wanted as most flexibility as possible, at the same time as keeping the same magnitude for the word embedding dimensions, which seems somehow intuitive;
- Output size of 766 (and we padd our paragraphs with a dummy PAD_{ID} up to this size);
- Question size of 100 (we also padd our questions in the same way).

We now analyze our main architecture, which has its main insights coming from the mentioned paper, but implemented with some modifications. Our implementation is more closely related with the Boundary Model described by the author, but we also don’t implement the search step in the end, which could provide great enhancements to our model at a future step. Let’s jump into our model:

- First, we run a BiLSTM over the question (with state size of 50);
- Within the same BiLSTM, we loop through the paragraph itself, with initial states as the end states of the previous BiLSTM. Call it our question-context-representation;
- We then compute an attention vector for each position of the question-context-representation based on the question embedding words, in a relatively simple way. Motivated by the fact that a high dot product indicates that 2 vectors point in the same direction and, therefore, represent similar contents, we compute the attention element $\alpha_i j$ for the $j_t h$ question-context-representation and the $i_t h$ question word as the dot product between their vector representations. To allow for some more flexibility in here, and to ensure we could have different dimensionalities between those vectors, we first map the $question - context - representation$ vectors (a tensor with shape [`batch_size`,`paragraph_size` `question_context_representation_size`] to one with dimensions [`batch_size`, `paragraph_size`, `question_representation_size`], by multiplying it by a matrix W to be trained as well). After we have the attention vector, we multiply the question representation by the attention vector to get a weighted representation of the question, in a similar fashion as described in the mentioned paper. Again, performing a similar step as in the paper, we concatenate this weighted representation with the question-context-representation we had before, and call it the "final inputs".

- Now, differently from the original paper, we don't run an LSTM over those final inputs. We set a new vector W of shape [final inputs size, 1] (to be trained), and we multiply each final input (associated to each position in the paragraph) by W (dot product), and obtain a score for each position in the paragraph. Note that we design a W_s and a W_e, one for the start position, one for the end position.

- We finally perform a softmax operation over the outputs of multiplication by W_s and W_e.

- We use Cross Entropy loss to evaluate our predictions.

Please, refer to the below sketch as a representation of our model.

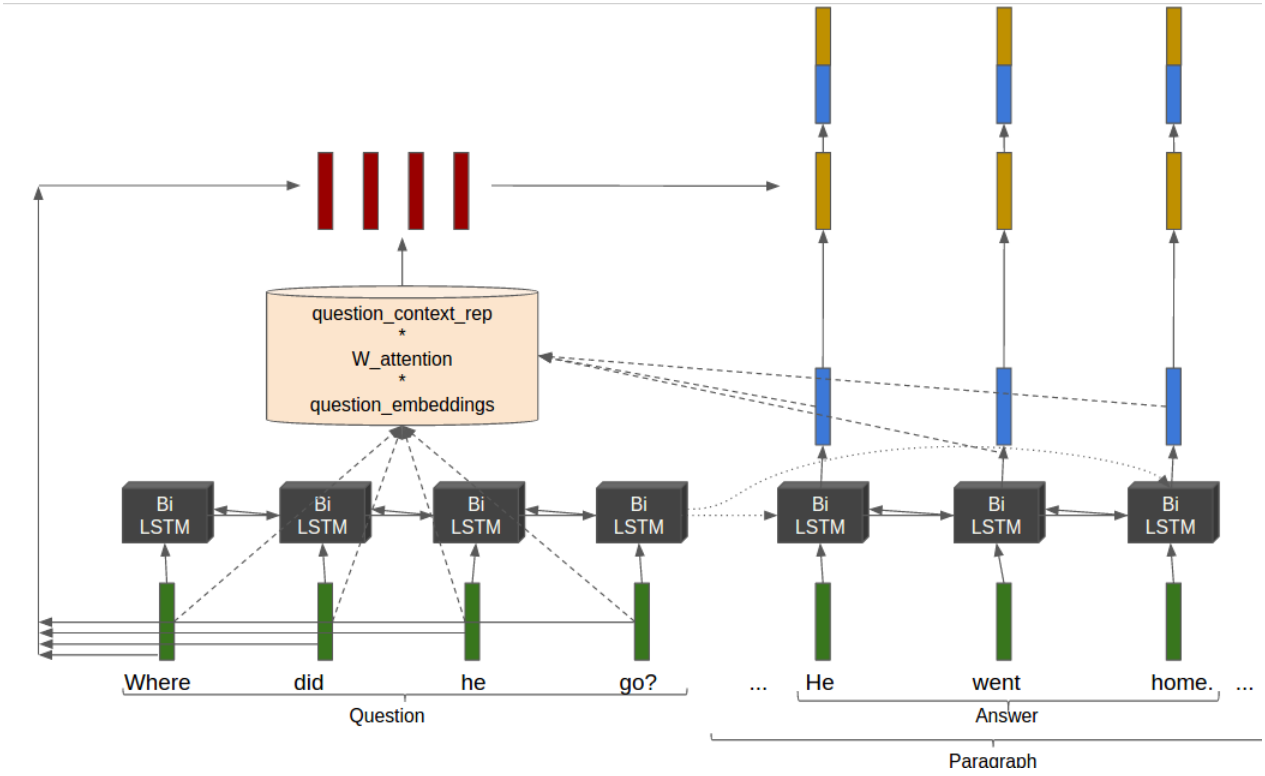

Figure 8: **Model Architecture I: generating the "final inputs"**

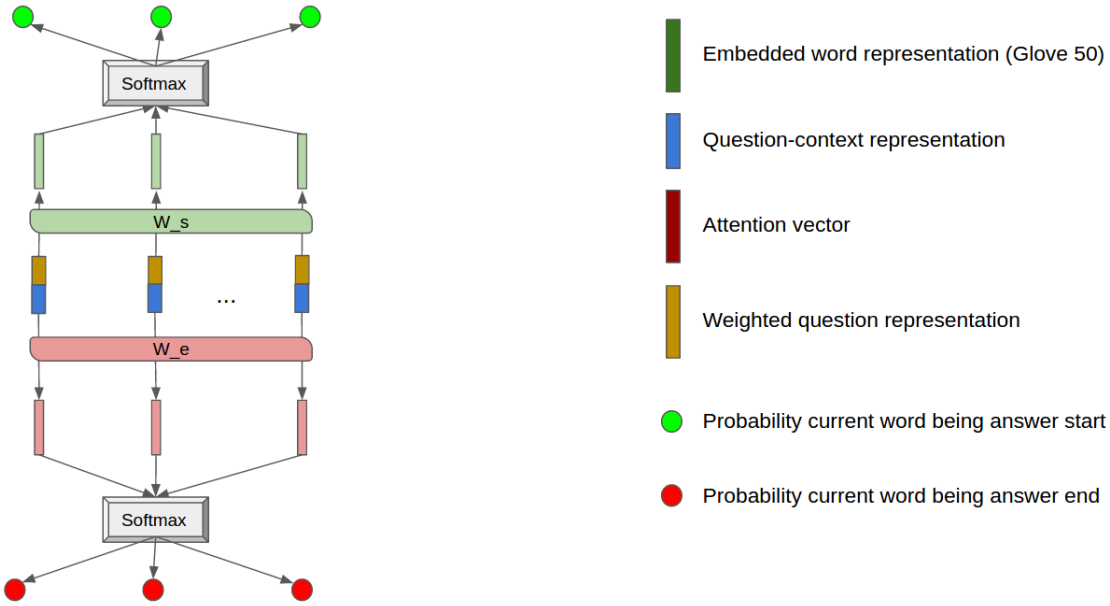

Figure 9: **Model Architecture II: from "final inputs" to predictions**

5 Analysis and Experiments

In this section we present the analysis of our Reading and Comprehension system and how it performs from time and accuracy perspectives.

5.1 Accuracy

	Train	Development	Test
F1	29.710	22.587	23.532
EM	19.000	12.015	12.745

5.2 Run Time

Our running time on a CPU is of about 35 minutes per epoch, or around 12 hours in total. On the GPU, we were able to run it in nearly half the time, i.e., slightly less than 6 hours.

5.3 Algorithmic convergence

After around 16 epochs our model has pretty much converged, as we can see from the learning curve below. Note that for Glove vectors of size 50 the models performs better, and that loss is computed for all examples, whereas F1 and EM just for a random subset of 100 examples at each epoch:

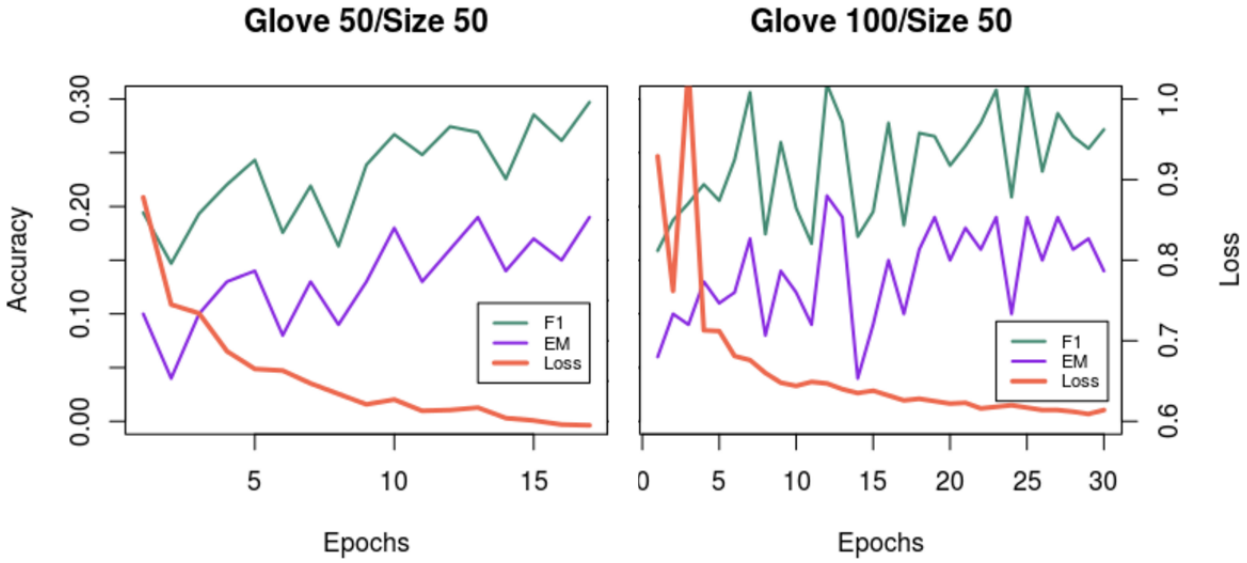

Figure 10: **Convergence of the algorithm per epoch**

5.4 Scalability

Training Dataset Size	Time Per Epoch (size 50)	Time Per Epoch (size 100)
50	1.5s	3.2s
100	2.6s	6.4s
500	12.0s	33.7s
1000	24.0s	62.9s

Our algorithm scaled linearly in terms of size of the training dataset on a CPU.

5.5 Conclusion

We were able to perform reasonably well given the limited size and relatively short time that our model takes to train (which can become an important advantage in case new datasets get eventually generated). We achieve a test performance slightly lower than the in-sample one, so there is some evidence that we are overfitting by some amount and in some parts of the model, and incurring in some Variance, but not as much as another issue: since even on training our performance is not as high as desired (state-of-the art or human baseline), we are facing a larger Bias issue, with an indication of underfitting on most parts of the model. We currently train over 45700 parameters for the 50 size model, and 181400 for the 100 size one, which is arguably low. Next steps to improve the model include increasing the dimensionality of the word vectors and the size of the hidden states. But since going from sizes 50 to 100 didn't improve the model, we could add complexity through different ways: adding fully connecting layers between the final outputs and the softmax operations, another LSTM over the "final inputs" (as in the mentioned paper), or more sophisticated attention functions. To control for the Variance, we tried a dropout rate of 0.15 over the "final inputs" before they are multiplied by W_s and W_e, without much success. So we can perform other types of techniques such as L1 or L2 regularization of the parameters, or an ensemble with different initializations.

Contributions

Dan worked on establishing the code pipeline, data processing, modelling, tuning, writing the source code (train.py, qa_model.py and qa_answer.py) and data exploration. Wissam worked on setting up the VM machine (Azure), Codalab submission pipeline, literature review and data exploration.

Acknowledgments

By: Wissam Baalbaki
I want to thank Dan for all the effort he put into this project. It could have never been done without him. Dan spent tens of hours grasping the concepts and implementing the model and he deserves more credit than me. Thank you

References

[1] Kenton Lee, Tom Kwiatkowski, Ankur Parikh, Dipanjan Das (2017); *"LEARNING RECURRENT SPAN REPRESENTATIONS FOR EXTRACTIVE QUESTION ANSWERING"*;

[2] Shuohang Wang, Jing Jiang (2017); *"MACHINE COMPREHENSION USING MATCH-LSTM AND ANSWER POINTER"*;

[3] Christopher Manning, Richard Socher; *http://web.stanford.edu/class/cs224n/*