Deep Learning for Question Answering on the SQuAD

Ramtin Keramati PhD ICME *keramati@stanford.edu*

Kian Katanforoosh MS MS&E *kiank@stanford.edu*

Reza Takapoui PhD EE *takapoui@stanford.edu*

Abstract

This paper describes our project for the Spring 2017 session of CS224N. Question answering has been a hard problem for a long time, and it was originally hard to apply cutting-edge Deep Learning algorithm to this problem because of lack of data. In 2016, the Stanford Question Answering Dataset (SQuAD) was published. It boosted the application of deep learning algorithm to Question Answering problems.

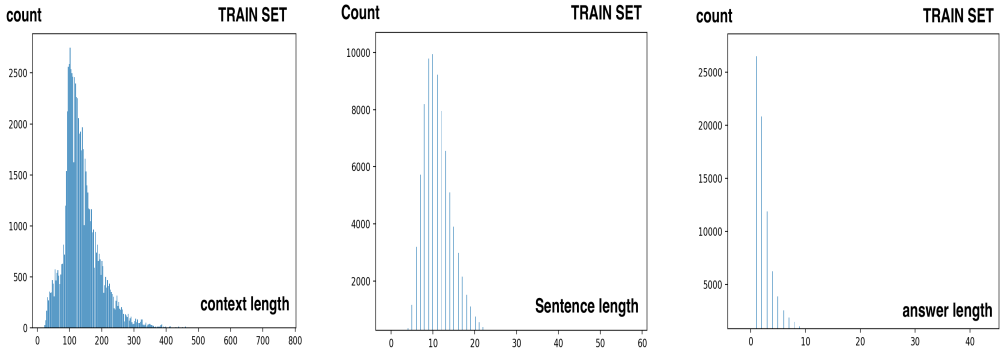
In fact, we will see in this paper two different algorithms to tackle this problem. We will start with a Baseline algorithm that relies on Bidirectional-LSTMs, and later boost its performances by adding an attention mechanism.

1 Data Analysis

Before jumping to the architecture of our model, we would like to see what our dataset look like. It will help us take the right decision and make hypothesis on our model's expected behavior.

2.1 Sequences length

First, let us derive some interesting plots on the frequency distribution of sequences (questions = sentence, context=paragraph and answers) length on the training set.



By looking at these plots, we can assert that:

- The sentence (or question) and context length distributions are quite spread out, and some questions/paragraphs are quite long. Classic Recurrent Neural Networks are known to usually perform poorly on long sentences (it is observed in Machine Translation for instance), because in practice they fail at keeping track of too much information. To resolve this we will use Long Short Term Memory (LSTM) cells in

our model instead of classic RNN cells.

- The answers are often short. To discuss more what are the implications let's look at the types of questions we have in our dataset

2.1 Questions type

Question	Count	Rate
How?	7545	9 %
Where?	3015	4 %
What?	34729	43 %
Who?	7479	9 %
Why?	1102	1 %

Interestingly, most of our train-set's questions are what-type questions. We thus expected to perform quite well on this type of questions on the validation or test set.

Besides, it is a fact that where/what/who-type questions generally lead to shorter answers compared to why-type questions. It makes sense that we observed a pike on short answers-length in our previous graph. We thus expect our model to perform better on short answers than long answers. As most of answers are short, we will try to find a trick so that our model learns to prioritize some specific parts of the input and focus more on these parts.

1.3 Important details to keep track of

During our study, we will constantly keep track of the following behaviors in our model:

- Gradient saturation: We will pay extra attention when initializing the weights of sigmoid/tanh neurons to prevent saturation. Besides, we will constantly print the gradient at the sigmoid/tanh neurons and keep track of its values.
- Exploding gradient: we will use gradient clipping (by ceiling the gradient value at 10) to avoid an exploding gradient to compromise the convergence of our model.
- Size of the hidden state in our LSTM cell: It is very important to test models with different hidden state sizes for the LSTM cell. Indeed, increasing the state size gives more flexibility to the model and better learning capacity (because we have more parameters). However, it increases the complexity of the model, the number of computations grows exponentially and it becomes hard to train the model in a finite amount of time. We have to find the right tradeoff.
- Number of stacked LSTM: Just as we said for the size of the hidden state in our LSTM cell, we will have to find a tradeoff between accuracy and complexity of the model by tuning the number of stacked LSTM layers.
- Dropout probability: We will use dropout and we will train our model with different dropout probabilities to make sure we are not overfitting the data.
- Overfitting the data: We will repeatedly train the model on a small sample dataset and try to overfit the data, to make sure our model has enough capacity to understand complex behavior. This doesn't mean the model is good but it gives us an idea on its flexibility.

2 Baseline model

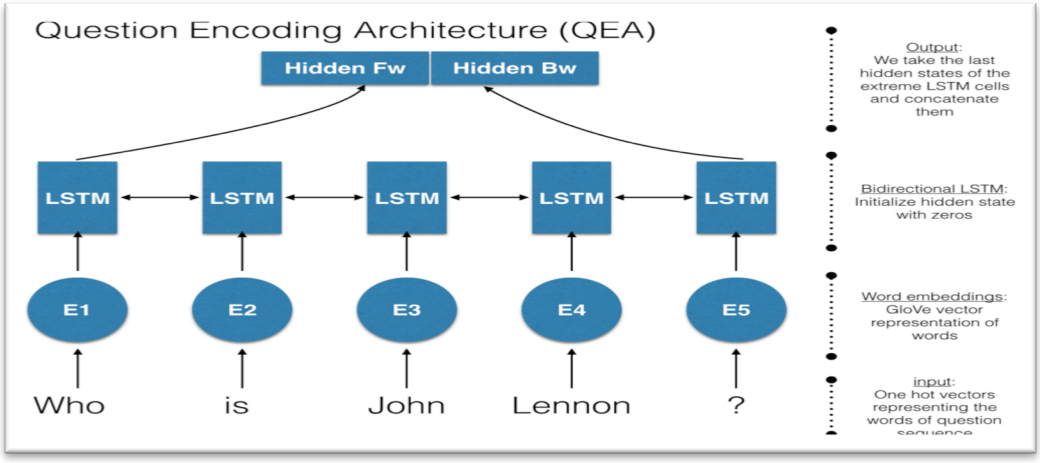
It is now time to jump on the detail of our first model. Our goal is to:

- Build a simple model that runs correctly in tensorflow. Once we get such a model, we can explore further and build more complicated features upon it.
- Evaluate this simple model with specific metrics, namely F1 and EM scores

2.1 Question Encoding Architecture (QEA)

To understand our Baseline model, we need to understand each of its

components, starting with the Question Encoding Architecture (QEA).



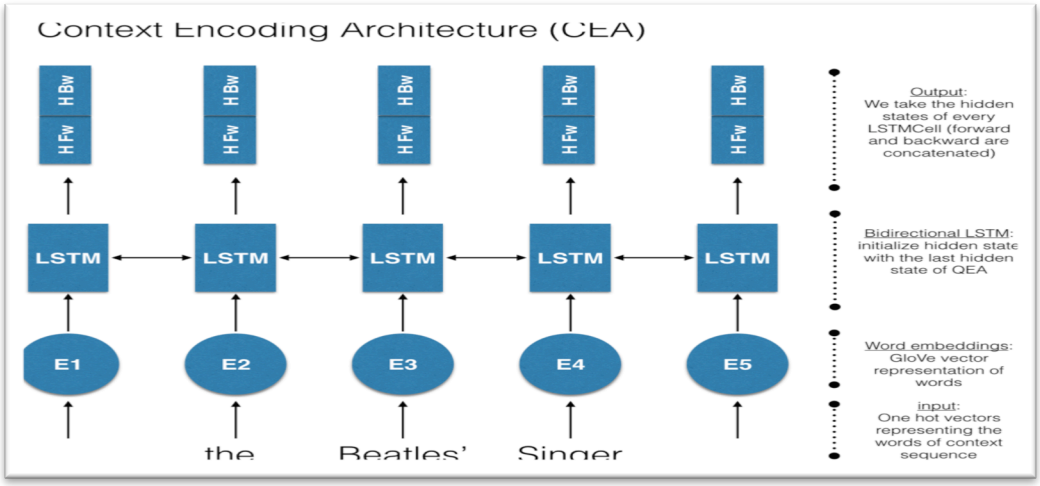
The input of the QEA is a sequence of one-hot vectors corresponding to the words of a given question. We first convert these words in their GloVe representation and give that as input to a Bidirectional LSTM where the hidden states are initialized as zero-vectors.

We compute the output of the QEA by concatenating the hidden states of the extremities of the Bi-LSTM. This output (output_QEA) will be used two times:

- to initialize the hidden states of the Bi-LSTM of the context words
- as an input to the decoder

2.2 Context Encoding Architecture (CEA)

As we said, we have another Bi-LSTM based model that takes the context words (GloVe Vectors) as inputs and has its hidden states initialized using the output_QEA.

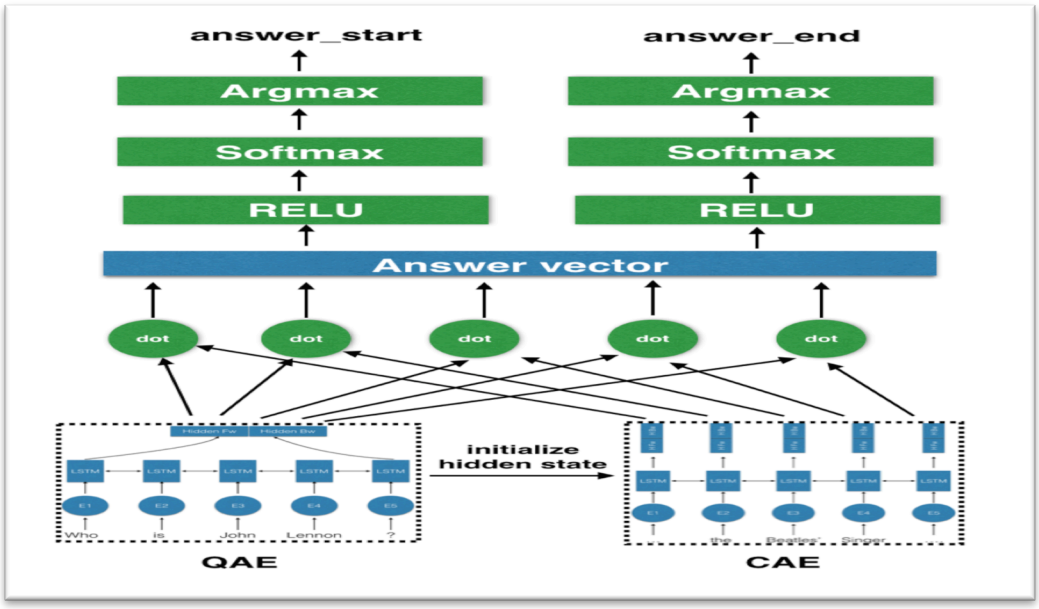


The CEA is very similar to the QAE, apart from the fact that we record the hidden-state vector of every LSTM cell.

2.3 Baseline Model Architecture (BMA)

After encoding our question sequence (QEA) and our context paragraph (CEA), we will use these new representation to find the answer. To do that, we will perform several layers of

computations as explained in the following scheme.



To keep track of the information coming from both the question and the context, we will first perform a dot product between QEA output and each CEA output. This is how we extract a feature answer vector. On this vector we successively apply a fully-connected layer with ReLU, Softmax and finally taking the argmax to find the indexes answer_start and answer_end.

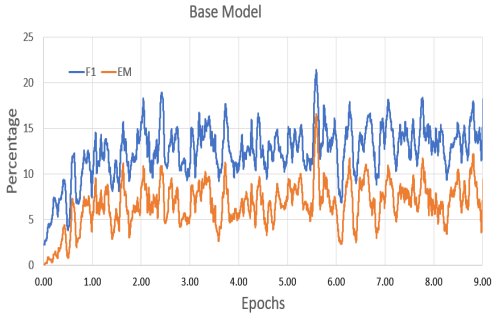
2.4 Important details

In this Baseline Model Architecture, we also looked at the following issues and tricks:

- Exponential decay of the learning rate: We wanted the learning rate to be fast at the beginning of the training, because we start from random, we rapidly want get to a smart model and then optimize it more and more slowly.
- Masks: We want to keep the right vector/matrix size for our representation given any (question, context) pair. Without a mask, our LSTM trains on some random input figures and the loss is wrongly evaluated.
- Dropout layers: to avoid overfitting we added a dropout layer in the fully connected layer with ReLU.
- Gradient clipping: We ceiled the gradients in all the backpropagation so that there is no exploding gradients that would fail our learning.
- Batch size tuning: The higher the batch size, the lower the training time and the higher the memory needed. We tuned this batch size until we found the right tradeoff given the time/memory constraints.

2.5 Baseline model results

Let's run our baseline model on the training set and observe the learning as well as the predictions.



We clearly see a good learning capability on the Train and Validation cross-entropy loss plot. The learning capability echoes the second plot as well with a growing F1 score and a oscillation around 14%.

Having a working baseline model makes the next steps a lot easier, and we only have to work on the model parts (encoding + decoding). Let's now try to improve this model.

3 Baseline model explorations

In this part, we are going to underline some explorations that we did.

3.1 Stacked-LSTM Depth

As said in part (1.3), adding several layers of Bi-LSTM one after each other makes our model more complex and capable of extracting more information. However, it also makes the time complexity increase. Let's look at our analysis for different number of Bi-LSTM layers.

#Stacks	1	2	3
Performance	Correct	Better	overcomplexity in time

We decided to keep 1 Stack because of the time constraint. Even if the results were better using 2 Stacks, it was taking about twice the time to train our model on an epoch for 1 Stack. See plots in (3.3).

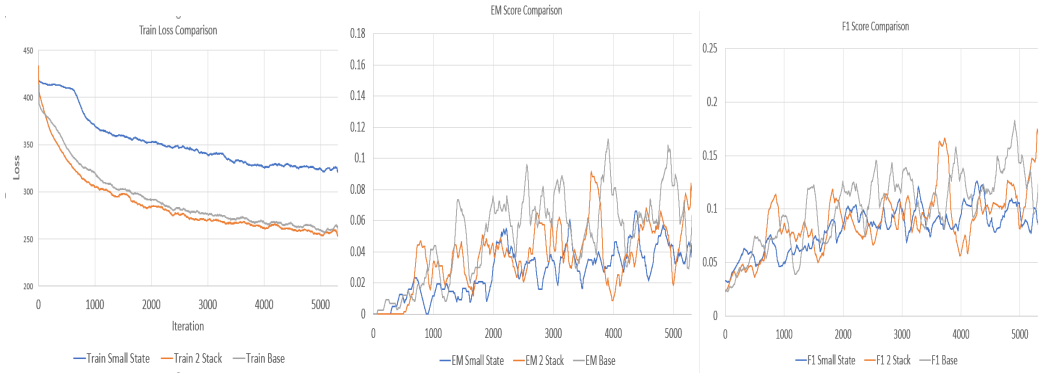
3.2 Hidden state size of the LSTM cell

As said in part (1.3), it is very important to choose accurately the state size for the LSTM cells as it can change the performances. However, it has the negative effect to drain a lot of memory. Here's our observation:

State size	64	128	256
Performance	Low	Correct	Out of memory

We decided to move to a state size of 128 for our LSTM cell because we still had enough memory and were getting better results as you can see on the plot in (3.3). For 256, we unfortunately ran out of memory as the number of parameters in our model increased exponentially.

3.3 Exploration plots



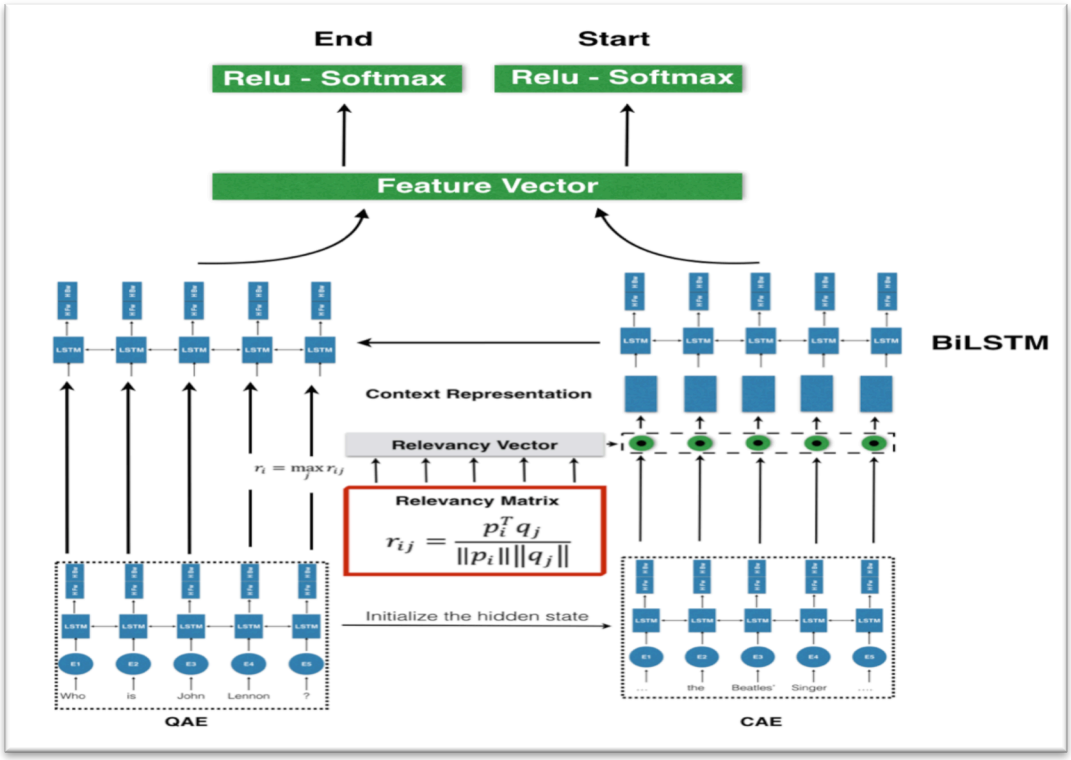
We can compare the performances between the baseline model (#State=128, 1 Stack) to a model with (#State=128, 2 Stack) and another with (#State=64, 1 Stack). This is one of the graph we used to tune these hyperparameters.

4 Personalized attention model

Armed with the knowledge based on Baseline model training, we propose a new network to predict answer of a given question our model is based a paper from IBM Research ([1]), with some improvement that we think would make the model better.

4.1 Model description

The following figure shows the abstract version of our model, given a question $\{q_1, \dots, q_m\}$ and a paragraph $\{p_1, \dots, p_m\}$, we want to predict two number a_s and a_e as the index of start and end of the question.



Word embedding Layer, in this layer for each word in the question and paragraph we choose their corresponding word vector from Glove data set. This layer use embedding look up table to retrieve the vector.

Summarization layer, we believe that given a question, each word of the context can be summarized in a better way, to capture its importance the context. So, we first initialize the BiLSTM with state size 128, to zero and feed a word embedding of question to that, to that we obtain:

$$\overrightarrow{h_i^q} = \overrightarrow{LSTM}\left(\overrightarrow{h_{i-1}^q}, q_i\right) \quad and, \quad \overleftarrow{h_i^q} = \overleftarrow{LSTM}\left(\overleftarrow{h_{i-1}^q}, q_i\right)$$

The same LSTM layer is applied to the paragraph initializing by the final backward and forward hidden state of question, so we obtain the same for paragraph:

$$\overrightarrow{h_i^p} = \overrightarrow{LSTM}\left(\overrightarrow{h_{i-1}^p}, p_i\right) \quad and, \quad \overleftarrow{h_i^p} = \overleftarrow{LSTM}\left(\overleftarrow{h_{i-1}^p}, p_i\right)$$

One of our major contribution to the main paper, is using this summarization layer before amplification layer.

Amplification layer, using the output of summarization layer, by concatenating the results of backward and forward hidden state, we can find a new representation each question word and paragraph word as $\{q_1', \dots, q_m'\}$ and $\{p_1', \dots . , p_2'\}$, then by using cosine similarity measure we can identify which words in the context are more important, it's kind of variant to attention:

$$r_{ij} = \frac{{q_i'}^T p_j'}{\|{q_i'}^T\| \|p_j'\|}$$

By this similarity matrix we will amplify each word of the context by $r_j = \max_i r_{ij}$, that is we build new representation by

$$p_i^{amp} = r_i * p_i$$

Abstraction Layer, after obtaining new representation of context words, based on their importance, we use another BiLSTM layer to encode the paragraph and question, this time we use the other way of feeding, that is we first encode the paragraph, using $\{p_1^{amp}, \dots, p_n^{amp}\}$, then feed the final backward and forward hidden state to encode the question with the same BiLSTM with hidden state size of 128, so we obtain:

$$\overrightarrow{h_i^{p*}} = \overrightarrow{LSTM}\left(\overrightarrow{h_{i-1}^{p*}}, p_i^{amp}\right) \quad and, \quad \overleftarrow{h_i^{p*}} = \overleftarrow{LSTM}\left(\overleftarrow{h_{i-1}^{p*}}, p_i^{amp}\right)$$

$$\overrightarrow{h_i^q} = \overrightarrow{LSTM}\left(\overrightarrow{h_{i-1}^q}, q_i\right) \quad and, \quad \overleftarrow{h_i^q} = \overleftarrow{LSTM}\left(\overleftarrow{h_{i-1}^q}, q_i\right)$$

Now we can use this hidden state representation of each word (by adding them together, namely $P_i = \frac{1}{2}(\overrightarrow{h_i^{p*}} * \overleftarrow{h_i^{p*}})$) to obtain some feature for each word in paragraph.

Feature Vector, We define six different feature for each word of paragraph as follows:

$$m_1^j = \frac{1}{M}\sum_i \overrightarrow{h^q}_i^T P_j, \quad m_2^j = \frac{1}{M}\sum_i \overleftarrow{h^q}_i^T P_j, m_3^j = \max_i \overrightarrow{h^q}_i^T P_j$$

$$m_4^j = \max_i \overleftarrow{h^q}_i^T P_j, m_5^j = \overrightarrow{h^{end}}_i^T P_j, m_6^j = \overleftarrow{h^{start}}_i^T P_j$$

These 6 features for each word can be used to figure out whether a word is the start or end of an answer.

Classification, to capture the dependency of neighbor words, we pass this feature vector to another BiLSTM with state size 8, and we used the maximum of hidden state value to pass to fully connected layer.

$$\overrightarrow{h_i^m} = \overrightarrow{LSTM}\left(\overrightarrow{h_{i-1}^m}, m_i\right) \quad and, \quad \overleftarrow{h_i^m} = \overleftarrow{LSTM}\left(\overleftarrow{h_{i-1}^m}, m_i\right)$$

$$s_i = \max H_i^m[j], \qquad H_i^m = [\overrightarrow{h_i^m}, \overleftarrow{h_i^m}]$$

Classification, Now, having a number s for each word of context we classify this using a

fully connected layer with ReLu activation:

$$p_1 = ReLU(sW_2 + b_2), p_2 = ReLU(sW_1 + b_1)$$

And then using a SoftMax layer to predict the probability of each word to be the start or the end of an answer.

$$a_s = argmax(p_1), a_e = argmax(p_2)$$

4.2 Results

After hours and hours of work on this model, we finally managed to get to an F1 score of 24% and EM score of 19%. But we are still stuck on some bugs that we weren't able to figure out yet. We are looking forward to big improvement on this model that adds an attention mechanism to our baseline.

Things that we noticed:

- It is important to define all the variable inside the qa scope. Otherwise, at each step we initialize the variables again.
- Dropout should take as argument "keep probability" instead of "drop probability".
- To make sure the LSTM will use the previous variables, we need to reuse the scope to share variables.

5 Conclusions

This study has led us to build a baseline model that predicts the answer in a context paragraph given a specific question. We have looked at many channel of improvement on this baseline model and ended up tuning several hyperparameters and catching bugs. We then undertook the built of another model, more complex, based on an attention mechanism. We tuned it as much as we could and observed results. We believe that with more time, computing power and help we can get very good results with our second model.

References

[1] Wang, Z., Mi, H., Hamza, W., & Florian, R. (2016). Multi-Perspective Context Matching for Machine Comprehension. *arXiv preprint arXiv:1612.04211*.

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

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