# CS224N Project: Let Computers Do Reading Comprehension

**Jiyue Wang**
jiyue@stanford.edu

**Ye Tian**
yetian1@stanford.edu

## 1 Introduction

In this work, we want to teach the computers to do reading comprehension. More specifically, the task is to let computers read a short passage and then answer some simple questions based on the passage.[1] proposes an end-to-end, RNN-like approach which, according to them, needs significantly less supervision but performs comparatively good to benchmarks for many applications. However, we found that their Q&A result is based on a small synthetic data set defined in [2]. We could see that the dataset is quite small and simple with a vocabulary size of 177. Here is a sample from the dataset.

> **Article.** Brian is a lion. Julius is a lion. Julius is white. Bernhard is green.
>
> **Question.** What color is Brian?
>
> **Answer.** White

In this work, we try to experiment their method on a much larger and complicated dataset from [2].

## 2 Dataset

We use the question answering corpus from [2]. Although this dataset is generated automatically, it is based on real CNN and Daily Mail articles and thus, more realistic and complicated. Here is a sample from the dataset:

> **Article.** @entity0 ( @entity1 ) – senate majority leader @entity2 announced tuesday the appointment of the first woman parliamentarian in the chamber since that position was created in the 1930s . @entity8 will replace retiring parliamentarian @entity9 ....
>
> **Question.**@placeholder will replace @entity9 , who was parliamentarian for 18 years
>
> **Answer.**@entity8
>
> **Entities.**
>
> @entity31:Frumin
>
> @entity2:Reid
>
> @entity1:CNN
>
> @entity0:Washington
>
> ...

As we move forward, we found that this dataset is much more difficult to process than we expected in the following aspects.

### 2.1 Corpus Size

We only consider the CNN dataset in this work. The CNN dataset alone has 2.27G, with 326,193 training data, 3,537 validation data and 2,803 test data. Morever, the vocabulary size of CNN dataset is around 50k if we exclude words with frequency less than 5.

### 2.2 Entity Number

As is shown above, the question and answer are only related to entities in the article and the same entity could appear in different articles. Also, the distortion from uniform distribution of the answers may well influence the training of an end-to-end model.

## 3 Approach

### 3.1 Data Representation

We represent language as three levels, that is, word, represented as an index in a vocabulary, sentence, represented as a list of words, and article, represented as a list of a sentences. The training data has three parts, the context, which is an article, represented by a list of sentences $x_1, ..., x_n$, the question $q$, a single sentence and the answer $a$ an entity, represented as a word or noun phrase.

### 3.2 The Original Model

Our model basically follows the architecture proposed in [1], as described in Figure 1 (a). Firstly it uses an embedding matrix $B$ to embed the word indices in $q$ as word vectors. Then embed the sentence to a vector $u$ by combining the word vectors in some way. Similarly, every sentence $x_i$ is embedded as a sentence vector $m_i$ in the input matrix, and a sentence vector $c_i$ in the output matrix. Applying SoftMax to the inner product of $u$ and each $m_i$ produces a score for each $x_i$, which represents the "correlation" between $q$ and $x_i$. And then use these scores as weights to give a weighted sum of $c_i$, producing an output vector $o$, which encodes the information in the context correlated with $q$. Finally, it adds $o$ and $q$ together as the input of a fully connected layer, and output a score for every candidate, that is, every word in the vocabulary; the one with largest score is considered to be the answer. In addition, we can also stack the layers, which is called hops, as in Figure 1 (b), and use $u^{i+1} = o^i + u^i$

The logic behind is that it simulates the process of how human beings deal with such problems – first select context information according to question, and then infer the answer with this information and the question.

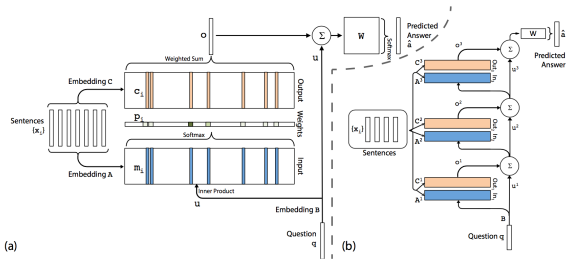


Figure 1: The Original Architecture [1]

There are some points to mention. Firstly, to avoid data sparsity, we filter out the words whose frequency is less than a threshold. Secondly, since we cannot guarantee that the vocabulary generated by our training set will contain all the words in test set as well as the filtered words, we use a special index to represent `UNKOWN`. Finally, because we need to input tensor data to our model, the sentence length and article length must be equal, both in and between data samples. So we set two parameters `sen_len` and `atc_len`. For sentences longer than `sen_len` or articles longer than `atc_len`, we simply cut them off. For sentences shorter, we append a symbol `EOS`, representing "End of Sentence", to the ends. For articles shorter than `atc_len`, we append sentences filled with `EOS` to the ends.

### 3.3 Sentence Encoding

There are several methods to encode word vectors into a sentence vector. The most simple way is to add word vectors together. Two problems with this method are

1. It cannot capture the information of word order and when sentence length varies in a large range.

2. There will be a lot of `EOS` at the tail of some sentences. As `EOS` is also encoded as a word vector, this method will introduce a lot of noise into the model.

To capture word order information, [1] proposed a position encoding method, which applies an element wise production between the word vectors and a position vector before they are added together. To eliminate the `EOS` noise, we add a mask for each sentence to indicate its end, and we can only add up the vectors before the end.

### 3.4 Improve with RNN layer

RNN is widely used to encode sequential information and deal with varied sequence length. Thus we think it suitable for our sentence encoding problem. So we try to improve the model by adding a RNN layer after the output word vectors from the embedding layer, as shown in Figure 2. To restrict model complexity, the RNNs are designed to share weights. Except for the inner-sentence order information, RNN can also capture inter-sentence order information, which is an advantage to the position encoding method in [1].

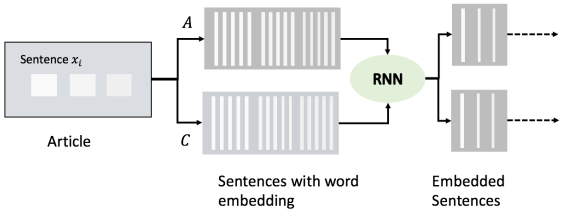


Figure 2: Improve with RNN

### 3.5 Shrink Candidate Categories

When producing the final output, the original model tries to give scores to every member in the

vocabulary. This is a huge work for a large vocabulary size. Moreover, we already know that the true answers are all entities, which implies that the other labels will never be learnt. So we modify the last fully connected layer to produce scores only for entity candidates.

### 3.6 Randomize Entities

Even though we shrink the class number from the vocabulary size (around 20k) to the entity number (around 300), the answers are restricted to a small portion of the categories. This heterogeneity nature of data makes the model hard to train. We noticed that what we really care about is the structure of the sentence, not the actual word behind the `@entity` mark. Specifically, we could replace an `@entity` representing `Mark` with `Jack` and the computer should give an answer of `Jack`. Thus, it is reasonable to restrict the entity number to a smaller set as long as we can distinguish entities within one article. We first manually set a threshold (equals 100) for the maximum entity `MAX_ENTITY_NUMBER` in one article. Then, for each article, we encode the entity with a random number drawn from 1 to `MAX_ENTITY_NUMBER`. For most of the dataset, the entity number is smaller than `MAX_ENTITY_NUMBER`; however, we do find that very few test cases contain an extremely large entity number (over 200) and we drop these test case for simplicity.

## 4 Analysis

A layer end-to-end neural network model is hard to tune.

In this section, we compare our work with [1] in several aspects.

### 4.1 Corpus Size

[1] gives result on both 1k data and 10k data, while we have a much larger dataset. At the very beginning, we planned to train on the whole CNN data set. Thus we first implemented a parallelized program to collect the vocabulary of the whole corpus. However, we soon found that we could not feed the whole corpus into memory to train the model. Due to limited time, we decided to experiment on 1k data to avoid the memory problem. A model trained on 1k data is certainly less powerful than the one on the full corpus, but we think that it should be a good indicator.

### 4.2 Vocabulary Size

During the experiment, we found that the vocabulary size is critical. The vocabulary size in [1] is only 171, while we have a large vocabulary size (50k). We found that the model can hardly learn even without regularization term because the embedding matrix would be very sparse with the increase of vocabulary size. This raised a question: does the information in 1k training data sufficient to infer the embedding matrix? To reduce the sparsity of the embedding matrix, we use a vocabulary collected from the 1k training data with frequency $\geq 5$.

### 4.3 Downside of End-to-End Architecture

The advantage of end-to-end structure is that it requires significantly less supervision. However, this is exactly what makes training hard. For the model, it only takes a series of number as input and a single number as output. The price of this simplicity is a complicated neural network architecture, which requires more data and fine tuning. As both of us are new to deep learning, we only managed to overfit the data set with 0 regularization and cannot get a satisfactory result with regularization.

### 4.4 Problems and (Potential) Solutions during Training

The first problem we met is that our learning rate causes either a loss explosion or a extremely low learning rate. So we use a relatively large learning and keeps saving snapshots of weights, and interrupt training, reset learning rate, load weights and go o to train. Then we found we cannot overfit even with a small data set of 100 samples. The average sentence length in the small data set is more than 40, and the largest is more than 100. So we set `sen_len = 40, 50, 60`, etc. However, whatever inputed, the model always produces the same answer, as shown in Figure 4.

```
training set
predicted answer:  [63 63 63 63 63 63 63 63 63 63]
true answer:  [29  2 72 51 78 98 68 38  9  6]
```

Figure 3: Different-input-same-output phenomenon

But if we set $sen\_len = 10$, the model will be able to overfit. This is obviously not what we want, since we threw away too much information.

But it noticed us that $EOS$ noise may be a problem. So we added word mask to tell the model which word vectors to be included in the sentence vectors. After doing this, the model overfit a small data set with 100 samples. And at this time, we can set $sen\_len$ larger, like 100 or so. One thing to be noticed is that, whether we use $sen\_len = 10$ or $sen\_len = 100$ with word mask, we need to stack the hops to at least 5 to overfit the data.

When applied it to a data set with 1000 samples, it cannot overfit again, and phenomenon is the same as before, that is, same output with any input. Interestingly, when we use the vocabulary of the former 100-sample data set, it overfit again. So we assume it is a problem of low parameter/data ratio. Moreover, some of the entities appear much more frequently than others. This bias may also be a reason of the different-input-same-output phenomenon. Therefore, to settle this problem, we first constrain the output to be only entities, and then shuffle and reduce the number of entities, this makes our model overfit the 1000 dataset, as shown in Figure 4.

```
training set
predicted answer:  [59 80 77 44 34 59 24 26 88 12]
true answer:  [59 80 77 44 34 59 24 26 88 12]
```

Figure 4: Different-input-same-output phenomenon

The log of loss and accuracy is shown in Figure 5

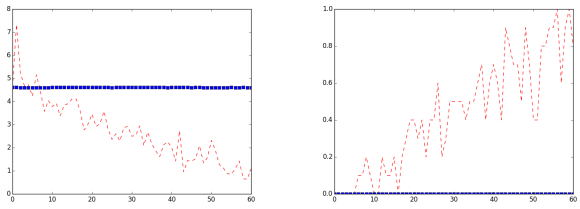

Figure 5: loss log and accuracy log

The hyperparameters at this time are shown in the following table:

*Table 1: Hyperparameters to overfit 1000 dataset*

| | |
|---|---|
| optimization method | nesterov momentum |
| learning rate | 0.0001 |
| momentum | 0.95 |
| batch_size | 10 |
| sen_len | 100 |
| atc_len | 100 |
| embedding_size | 40 |

## 5 Future Work

We spent a lot of time tuning the model but did not get a satisfactory result. This is partially because we are both new to neural network. However, we do have learnt a lot during the process. We found that for an end-to-end task as complicated as this one, we need more hidden layers and more computing power as well as time to train the model. We would try transfer learning to give a better initialization to the embedding layers. Moreover, after overfitting, we utilize L1 and L2 regularization, but these only produce a near 0 training accuracy and near 0 test accuracy. We can fine tune the penalty parameter or use drop-out to regularize our model in the future. If possible, we could further refine the embedding of sentence. Instead of a plain RNN layer, we could try LSTM-RNN model[3] and even take advantage of the parsing tree like the recursive auto-encoder [6].

## References


[1] Sukhbaatar, Sainbayar, et al. *End-to-end memory networks.* arXiv preprint arXiv:1503.08895 (2015).

[2] Hermann, Karl Moritz, et al. *Teaching machines to read and comprehend.* arXiv preprint arXiv:1506.03340 (2015).

[3] Palangi, Hamid, et al. *Deep Sentence Embedding Using Long Short-Term Memory Networks.*

[4] J. Weston, A. Bordes, S. Chopra, and T. Mikolov. *Towards AI-complete question answering: A set of prerequisite toy tasks.* arXiv preprint: 1502.05698, 2015.

[5] Bottou, Lon. *From machine learning to machine reasoning.* Machine learning 94.2 (2014): 133-149.

[6] Socher, Richard, et al. *Semi-supervised recursive autoencoders for predicting sentiment distributions.* Proceedings of the Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, 2011.