CS224n Assignment 4: Machine Comprehension with Exploration on Attention Mechanism

Chen Guo
Institute of Computational and Mathematical Engineering
Stanford University
Stanford, CA 94305
cguo2@stanford.edu
CodaLab User Name: cguo2

Abstract

This goal of this paper is to perform the prediction task on SQuAD dataset about reading comprehension. Given a pair of context paragraph and a question, we'll output an answer. To do this, a model is built combining the idea of Bidirectional LSTM and attention flow mechanism. The basic architecture and setup details of the model are introduced, so do the summary of performance and error analysis during experiments. Further more, the main exploration part would be focused on how to plug appropriate attention mechanism into the whole neural network architecture.

1 Introduction

In this project, I focused on both achieving the goal of high-performance question answering on SQuAD dataset and also exploring different models to see how do model architecture and initial setup influence the performances of our prediction performance.

As a first step, we have to find a method to represent the context paragraph and question in an appropriate way which can carry as much as the information contained in both of them. We choose pre-trained embedding vectors to represent the words in contexts and questions. Then we apply bidirectional LSTM model on contexts and questions to allow the model to understand them as a whole instead of every single word separately, such that our representation would make more sense of the environment around the target words.

After we got a good representation of both contexts and questions, here comes the crucial part of this project: how to make connections between questions and contexts so that the model can highlight the words that are related to the specific questions. Thats also what my exploration mainly focused on: attention mechanism. We need to map the question representation to the context representation to understand the fusion relations between the two of them.

Based on the attention matrix, the model will further more give a final representation of every word in the context paragraph carrying the information extracted from our attention mechanism. Two probability distribution will then output based on that, one being the probability of every word as the starting token of the correct answer, the other being the probability of every word as the ending token of the correct answer.

I've tried four different models. Two of them has the same architecture but different fusion functions, the other two share the same structure with different hidden state sizes. I compare these models in terms of basic performance (F1 and EM), training speed, convergence rates, stability on gradient norms also the parameters and special setups they separately need.

2 Related Work

For the newly-released SQuAD dataset, a lot of work has been done to produce a Neural QA System, which gives very decent results. Although different models make use of different model architectures, the most variation is located on the attention mechanism, which is the essential module of modeling complex interactions between context paragraphs and questions.

In [1], co-attention encoder is used to attend the question and document simultaneously and finally fuses both attention contexts. After generating the foundation for selecting the best span, they make use of dynamic pointing decoder which takes into account the current estimates of start and end positions and produces new estimates of the start and end positions. Although we didn't use dynamic attention mechanisms at last, we are inspired by the attention mechanisms introduced in this paper.

In [2], the author made use of attention flow and generated query-to-context representation and also context-to-query representation. The attention mechanism is static instead of dynamic, which allows the model separately split the energy on both parts. The model is creative in using element-wise multiplication on many parts of the whole process, proved to be successful to capture the interaction of vectors/matrices. Also the model functionize many specific steps, allowing people to try different functions (like fusion function and final query-aware representation).

3 Approach

3.1 Initial Understanding of the data

To understand better about the dataset, we plot the distribution of context paragraph length, question length and answer length in Figure 1. Telling from the data, truncating the context paragraph length at 500 600 might be a reasonable choice.

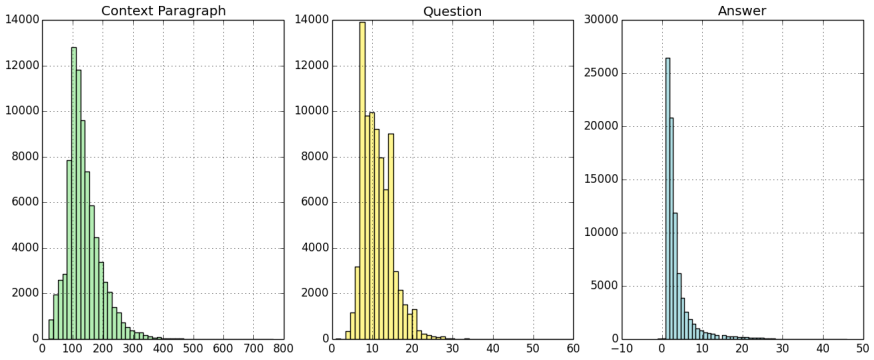

Figure 1: Distribution of Lengths of Context, Question and Answer

3.2 General Approach

The two models I tried are based on the same general approach summarized by [1]. The model structure can be formulated as 2 and briefly described as below:

- Word-level Embedding Layer: Based on GloVe pre-trained embeddings, we can represent every word in the context/question as a vector. In this specific problem, we use 100-dimensional trimmed vectors as embedding reference. From the step, we get $\in \mathbb{R}^{T\times d}, Q \in \mathbb{R}^{J\times d}$ where T and J are the lengths of context and question.
- Contextual Embedding Layer: Pass the word-level embeddings into bi-directional LSTM. This will let the embeddings include the useful information contained in the environment.
- Attention Flow: Based on the valid embeddings, we can get attention matrix with dimension $T \times J$.

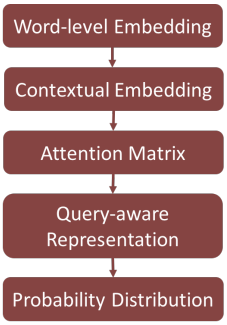

Figure 2: General Structure of the Model

- Query-aware Mixing: Given the attention matrix, we can get a better representation of contexts carrying the information in the question, so-called query-aware mixing. This will lead to an matrix with dimension $T \times D$ where D is defined custom as hidden size.
- Decoding: Using the representation of query-aware mixing, we should be able to get probability distributions (one for starting index, the other for ending index) by making use of more LSTM cells, simple linear transformation and softmax functions.

We'll mainly discuss two models here, which are based on the general approach above. The main difference is located in the last three steps: getting attention matrix, generating query-aware representation and decoding to probabilities.

3.3 Model 1: Sequence Attention Mix Model

Sequence Attention Mix Model is one of the simple baseline models which utilizes mixing attention mechanism. Assuming that the maximum length of context and the maximum length of question is T and J, the embedding size d. Then the model architecture is set up like:

$$
\begin{aligned}
\text{Word-level Embedding:} & X \in \mathbb{R}^{T \times d}, Q \in \mathbb{R}^{J \times d} \\
\text{Contextual Embedding:} & X \xrightarrow{Bi-LSTM} H \in \mathbb{R}^{T \times 2d}, Q \xrightarrow{Bi-LSTM} U \in \mathbb{R}^{J \times 2d} \\
\text{Attention Flow:} & A = \text{softmax}(HU^T) \\
\text{Query-aware Mixing:} & M = [AU, H] \cdot W + b \text{ where } W \in \mathbb{R}^{2d \times D}, b \in \mathbb{R}^D \\
\text{Decoding:} & M \xrightarrow{Bi-LSTM} M' \in \mathbb{R}^{T \times D} \\
& Mw_p = P, M'w'_p = P' \text{ where } w_p, w'_p \in \mathbb{R}^{D \times 1} \\
\text{Loss:} & L = -(\log p_{\text{start}}[\text{true start}] + \log p_{\text{end}}[\text{true end}])
\end{aligned}
$$

where the attention flow is simply the matrix multiplication of the contextual embedding matrices. And the query-aware mixing is basically combining both the original contexts and attention-based representation. This model involves variation as the choice of hidden size D, we'll conduct experiments on different choices D. Suppose $d = 100$ for both embedding size and LSTM output size, we tried $D_1 = 2d$ and $D_2 = 4d$.

3.4 Model 2: Bi-directional Attention Flow Model

Bi-directional Attention Flow Model is the model described in [2], where they achieved the state-of-art results on SQUAD dataset. The model I test is a slightly simplified version of the model in the paper. I removed the character embedding part but only used word embeddings. This model used more complicated query-aware representations. The model setup can be described here:

$$
\begin{aligned}
\text{Word-level Embedding:} & X \in \mathbb{R}^{T\times d}, Q \in \mathbb{R}^{J\times d} \\
\text{Contextual Embedding:} & X \xrightarrow{Bi-LSTM} H \in \mathbb{R}^{T\times 2d}, Q \xrightarrow{Bi-LSTM} U \in \mathbb{R}^{J\times 2d} \\
\text{Attention Flow:} & A = \alpha(H, U) \in \mathbb{R}^{T\times J} \\
\text{Query-aware Mixing:} & b = \text{softmax}(\max_{\text{row}}(A)) \in R^T \\
& \tilde{U} = AU \in \mathbb{R}^{T\times 2d}, \tilde{H} = b \cdot \mathbf{1}^{1\times 2d} \circ H \in \mathbb{R}^{T\times 2d} \\
& G = [H, \tilde{U}, H \circ \tilde{U}, H \circ \tilde{H}] \in \mathbb{R}^{T\times 8d} \\
\text{Decoding:} & G \xrightarrow{Bi-LSTM} M \in \mathbb{R}^{T\times 2d}, M \xrightarrow{Bi-LSTM} M' \in \mathbb{R}^{T\times 2d} \\
& [G, M]w_p = P, [G, M']w'_p = P' \text{ where } w_p, w'_p \in \mathbb{R}^{10d\times 1} \\
\text{Loss:} & L = -(\log p_{\text{start}}[\text{true start}] + \log p_{\text{end}}[\text{true end}])
\end{aligned}
$$

where the fusion function for attention flow is generalized to a function that take two matrices of dimension $T \times 2d$ and $J \times 2d$ and output a matrix with dimension $T \times J$. Further more, the query-aware mixing is more complicated than the baseline model, it generates context-to-query attention flow and query-to-context attention flow at the same time. Then concatenation and element-wise multiplication are used to combine the representations together. For this model, the main variations I tried are located on the attention flow fusion function: $A_1 = \text{softmax}(HU^T)$ and $A_2[i, j] = [H_{i\cdot}, U_{j\cdot}, H_{i\cdot} \circ U_{j\cdot}]$.

3.5 Search for Answers

After we got the probability distribution, we use the following algorithm to search for the best answer span satisfying $(s, e) = \max_{s\leq e} p_{\text{start}}(s) \times p_{\text{end}}(e)$.

Algorithm 1 Answer Span Searching Algorithm

```
From above we get P and P'.
Normalize: p_start = C_mask · softmax(P), p_end = C_mask · softmax(P').
Best start s = 0, best probability p = 0 and best answer a = (0, 0).
for k in 1:n do
    if p_start[k] > p_start[s] then
        s = k.
    end if
    if p_end[k] * p_start[s] > p then
        p = p_end[k] * p_start[s]
        a = (s, k)
    end if
end for
return a
```

4 Experiments

4.1 Summary

For the Sequence Attention Mix Model, we tried two variations $D_1 = 2d$ and $D_2 = 4d$ where $d = 100$. For Bi-directional Attention FLow Model, we tried two variations with $A_1 = \text{softmax}(HU^T)$ and $A_2[i, j] = [H_{i\cdot}, U_{j\cdot}, H_{i\cdot} \circ U_{j\cdot}]$. The summary of the four trials are in Table 1. In terms of the specific hyper-parameters in the training. We used maximum context length as 600 and maximum question length as 60. Dropouts are added after each LSTM layer and also before the final softmax function after linear transformation.

Table 1: Experiments Summary

	Model 1 with D_1	Model 1 with D_2	Model 2 with A_1	Model 2 with A_2
Dropout	0.9	0.85	0.8	0.8
Learning Rate	0.01	0.005	0.01	0.01
Gradients	Exploding	Exploding	Stable	Stable
Num of Parameters	0.5M	1.1M	1.2M	1.2M
Time/Epoch	0.7h	2.5h	3.3h	6.0h
Best F1	28%	30%	52%	35%
Best EM	19%	21%	38%	24%

4.2 Findings and Thoughts

From experiments, we have some findings and intuitive thoughts in terms of the model performance, implementation and hyper-parameter tuning.

- **How does dropout help? Is it helpful for all methods?** Not Really. For Model 1 with D_1, the model does not suffer from over-fitting problem because the model only has 0.5 million parameters. Considering that the variables in each sample is roughly $100 \times (600 + 60)$ (embedding size is 100 and context length and question length are separately 600 and 60) which is around 0.6 million. For this situation, the model is more under-fitting than over-fitting. However, for more complex models like Model 2, dropouts do help with over-fitting problems.
- **Observation about gradients.** We observed exploding gradients phenomenon for Model 1, we must use gradient norm clipping for the purpose of stability. But for model 2, the gradient norm is pretty stable, basically around 1.
- **Model Capacity and Performance.** If we use the number of model parameters to represent the model capacity, we want to know if there are any significant relationship between model capacity and performance. For model 1, when we increase the hidden size from 200 to 400, the number of model parameters also doubled. The training time also doubled. However, the model performance didn't see any drastic increasing. My conjecture is that under this case, the model structure is the bottle neck of the performance but not the model capacity. Simply increasing the number of parameters doesn't help for model structure that is not complicated enough. We haven't got the time to train Model 2 with larger hidden state size. My guess is that it would help more because the model is more complicated.
- **Model Implementation.** The models I tried include a lot of matrix transformations, especially Model 2. For example, if we want to multiply matrix with size `[batch_size, max_context_length, state_size]` and another matrix with size `[state_size, max_question_length]`. We need to reshape first then do multiplication then reshape back. This might be computationally expensive, because the use of fusion function A_2 for Model 2 increases the training time a lot because I didn't find a good way to implement it. Also the steps for calculating query-aware representation are not really well implemented. Because it involved a lot of transpose and reshape. The biggest regret for this project might be this: It would be better if I could find a better implementation of the model.

4.3 Analysis of the Best Model

During the training of the best model among the above four trials, we plot the changing of loss and gradient norms as training in Figure 3. We can see that the loss got decreased drastically in the first epoch and then got better slightly in the future training. At the same time, the gradient norm kept very stable around the region of [1, 2].

For different categories of questions, we did the error analysis on validation set. We can see the number of questions, F1 and EM score for each category in Table 2. We can see that, as expected, our model performed better on questions with `When`, `Where` and `Who`, which are intuitively easier to locate the exact answers. However, for questions like `Why`, we have relatively worse F1 but still acceptable. However, the gap between and F1 and exact match score is very high. Intuitively, the answers for `Why` questions are more vague and subtle so it's hard to capture the exact answer

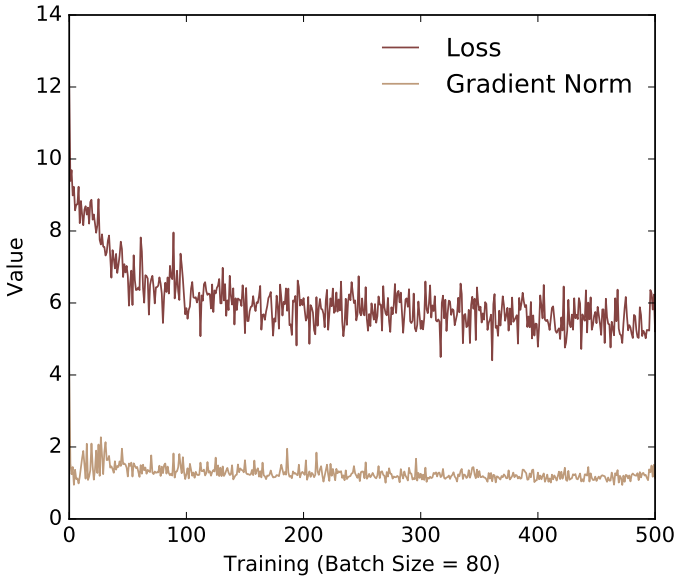

Figure 3: Changing of Loss and Gradient Norm during Training

span. Another observation is about the difference between `Where` and `Who`. They have similar F1 score but the EM scores have larger difference. This is probably because the name of person is easier to recognize, although the names of places are more subtle.

Table 2: Error Analysis

Category	Number	F1	EM
When	349	61.7	43.3
Where	173	55.5	37.0
Who	424	56.5	42.2
Why	85	40.2	14.1
Which	98	48.5	31.6
How	382	57.0	40.1
What	2019	51.0	32.1

5 Conclusion

References

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

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