Ensemble Leaning for Stanford Question Answering Challenge

Yuzhou Liu
yuzliu@microsoft.com

Abstract

In this paper we explored techniques used in machine comprehension and question answering using the Stanford Question Answering Dataset (SQuAD). The SQuAD challenge defines a task where a context passage and a query are given and an answer to the query needs to be identified in the context passage.

Typically, this kind of task is approached using deep recurrent neural network such as Gated Recurrent Unit (GRU), Long-Short Term Memory (LSTM) and Attention mechanism. In this paper, we explored several recently published successful neural network architecture and attention mechanism such as BiDAF, Dynamic Coattention and Self Attention. In the end, this study was able to produce a F1 score of 76.7 and an EM score of 67.1 on the development dataset and a 77.65 F1, 68.1 EM score on the test dataset.

1 Introduction

Machine Comprehension (MC) is a task where an AI agent is given a query about a paragraph of context and is required to predict the answer. Such problems have gained popularity in recent years due to it's vast range of applications. The recent advancement of deep learning techniques has shown good progress in many machine learning areas including machine comprehension and question answering. To solve this problem, the machine learning system is required to understand both the context passage and the query and in most cases the connections between them to be able to accurately predict the location of the answer phrase in the context passage.

The Stanford Question Answering Dataset (SQuAD)[1] is a dataset of 100,000+ question and context pairs for machine comprehension on text. The training and development dataset are publicly available whereas the test dataset is kept secret. There is a public leaderboard recording various model's F1 and EM scores. At the time this report is written, there are many state-of-the-art models producing predictions that are close to human level performance in terms of F1 and EM scores.

Since solving this task involves the understanding and modeling of relationship between context and query, attention mechanisms are usually used in the model and in this paper, I mainly investigate several recently published attention mechanisms and some other techniques.

Specifically, we implemented and evaluated Dynamic Coattention[2], BiDAF Attention[3], Self-Attention[4] and also experimented with a new attention mechanism where the output of Coattention is fed to a Self-Matching Attention network introduced in the RNET[4] paper.

2 The SQuAD Dataset and Problem Definition

In the SQuAD challenge, given a context passage consists of an array of words $C = \{c_1 \dots c_N\}$ and a query about this context $Q = \{q_1 \dots q_M\}$, the system is required to produce the answer to the query as two integer indexes i, j where i indicates the start index of the answer and j indicates the ending index of the answer in the context passage.

Here are several example context, query and answer triplets:

Question: In what year was Nikola Tesla born?
Context Passage: Nikola Tesla (Serbian Cyrillic: Никола Тесла; 10 July 1856 – 7 January 1943) was a Serbian American inventor, electrical engineer, mechanical engineer, physicist, and futurist best known for his contributions to the design of the modern alternating current (AC) electricity supply system.
Answer: 1856

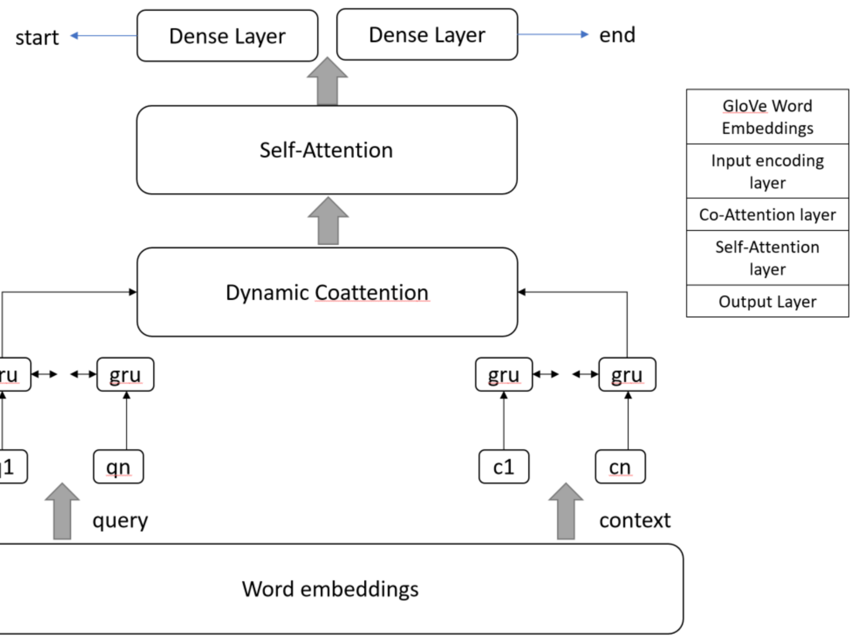
Question: Why was Tesla returned to Gospic?
Context Passage: On 24 March 1879, Tesla was returned to Gospic under police guard for not having a residence permit. On 17 April 1879, Milutin Tesla died at the age of 60 after contracting an unspecified illness (although some sources say that he died of a stroke). During that year, Tesla taught a large class of students in his old school, Higher Real Gymnasium, in Gospic.
Answer: not having a residence permit

3 Experiments

In this section, we describe our experiments in detail. All models we have experimented with can be divided into three layers: RNN encoder layer, attention flow layer, and output layer.

The input to all of our models are the pre-trained GloVe[5] word embeddings. We will first discuss the baseline model whose implementation was given by the course staff, then we will discuss various improvements we have implemented and their effectiveness on top of the baseline model.

Here is the block diagram of our final model:



3.1 Baseline Model

The baseline model uses the GloVe word embeddings pretrained from Wikipedia corpus. For each context and question tokens/words, the corresponding word embeddings are looked up from the word embedding matrix of the entire vocabulary.

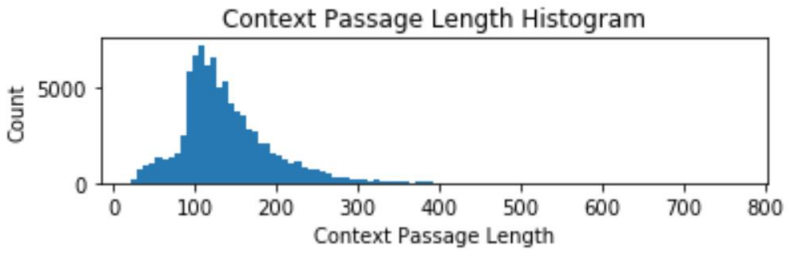
The model first encodes the context and query embeddings using bidirectional GRU and the hidden states of forward and backward GRU are concatenated. This input encoding layer captures the forward and backward sequence information for both the context and the query.

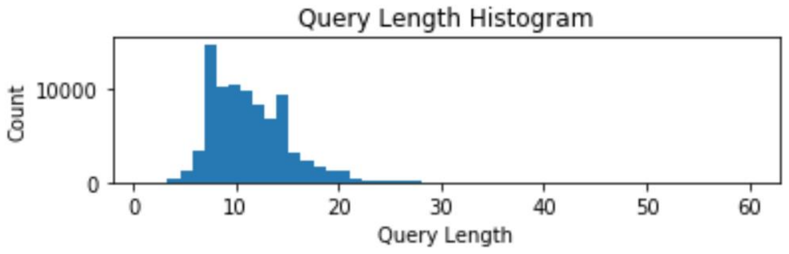
The attention layer of the baseline implementation uses the context hidden states attending to the query hidden states by a simple dot product and produce an attention score. Then the attention score is taken dot product with the query hidden states to produce the attention output which is a weighted sum of the query vectors. Before sending to the output layer, the attention output is concatenated with the original context hidden states to produce the blended representation.

The output layer of the baseline model is a simple feed-forward network with a SoftMax output over all positions in the context passage for both the start and end indexes respectively.

3.2 Input Data Analysis

The baseline codebase used several default values of hyper-parameters such as context length, query length (600 and 30 respectively). These dimensions will greatly affect the computation time for the model in both training and inference.
Therefore, we first conducted some analysis on the entire training dataset to understand statistically what the SQuAD dataset looks like in terms of context and query length.





As shown in the above figure, in the training data the context passage length in the number of words is tightly distributed around the range of 100 to 200 and rarely have any context passage longer than 500 words. Therefore, we set the max context length to be 500 in our model. With the same analysis on query length, we set the max query length to be 30 in our model. When processing the input data, any context or query that is longer than their respective max length is truncated to the max length.

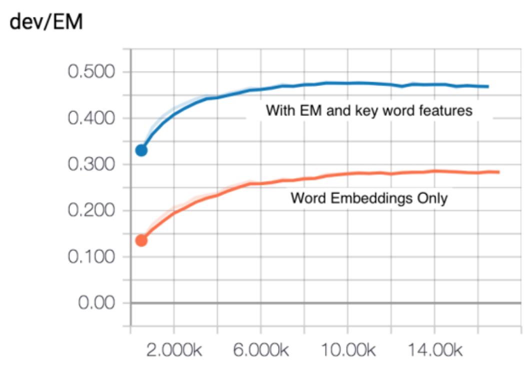
3.3 Input Encodings

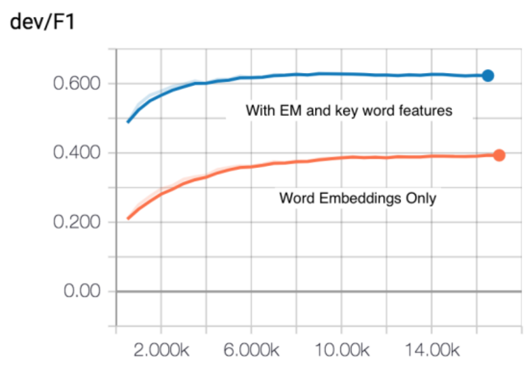
As apposed to just using the word embeddings of the words in the context and query, our input feature vectors have additional features like what is described in Chen el al. [5] For both the context passage and the query, a bidirectional recurrent neural network is used for encoding the input into hidden states. The input feature vector is consisting of the following parts:

- *Word Embeddings and Query Key Words*: We use the GloVe word embeddings pre-trained on 6B Wikipedia data. (Pennington et al., 2014). These pre-trained word embeddings are fixed during training with the following exceptions: words such as *what, how, who, when, which, where, why* are obviously high-level features that indicates what "type" of query it is. Therefore, these word embeddings are trainable variables in our model.

- *Exact match*: We use two simply binary features, indicating whether a word in the context passage c_i is an exact match of any words in the query, either in it's original form or the lower-case form. We append these indicator feature

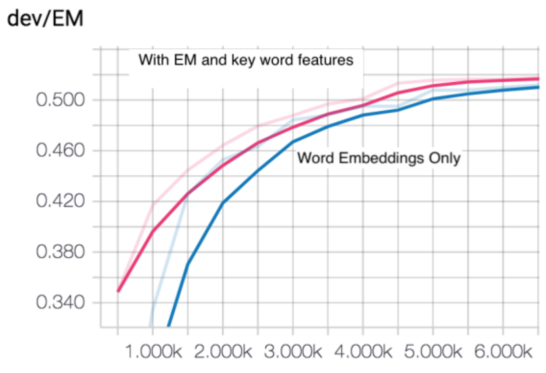
Surprisingly, these simple addition of hand-engineered input features significantly increased the performance on top of the baseline model, as shown in the F1/EM score plot below:

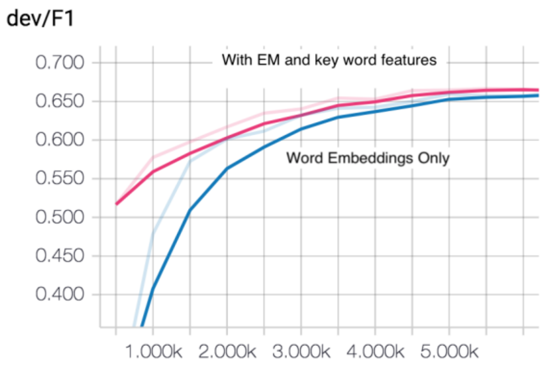




In more complex models with more advanced attention mechanisms, this input feature addition to word embeddings also provided a boost in performance however the boost is not as significant. This is possibly due to the fact that advanced attention mechanisms are already capable of capturing the exact match and query key word features in the model itself and adding additional input features like these does not help as much.

The following plot shows the self-matching attention we implemented from the Rnet paper [5] with and without the help of the hand-engineered input features. The performance gap between the two cases is significantly smaller.





The input vectors are then encoded with a bidirectional GRU to be consumed with later stages. We have experimented with using LSTM however the final model used GRU since they have similar performance and GRU is less computationally expensive.

3.4 Attention Layer

We have experimented with various attention techniques in recently published papers on the SQuAD challenge. The best performing attention layer in our experiments is a combination of Co-Attention and Self-Attention as explained below:

3.4.1 Dynamic Co-attention

The dynamic co-attention is similar to the baseline attention mechanism but with a few add-ons for performance.
With the context and query hidden states encoded by the input layer RNN, we add a sentinel vector concatenated with the context and query hidden states. This allows the attention mechanism to not attend to any particular word in the input.

$$Q = [q_1 \dots q_m; q_{sentinel}] \;\; \mathbb{R}\,(m+1 \;\times h)$$

$$C = [c \dots c_n; c_{sentinel}] \;\; \mathbb{R}\,(n+1 \;\times h)$$

Where $m, n \; and \; h$ represents query length, context length and hidden size respectively.
For the query encoding, a non-linear projection layer is added on top of the original query encoding with $tanh$ non-linearity.

$$Q_{proj} = \tanh(WQ + b) \in \mathbb{R}(m + 1 \times h)$$

An affinity matrix is then computed by:

$$\mathrm{L} = \mathrm{C}^{\mathrm{T}} Q \in \mathbb{R}\,(n+1 \;\times m+1)$$

The affinity matrix captures the similarity between each context and query words.
Then the affinity matrix is used to compute Context-to-Query attention and Query-to-Context attention.

$$CtoQ = Q\; softmax(L_{i:}) \in \mathbb{R}\,(m+1)$$

$$QtoC = C\; softmax(L_{:j}) \in \mathbb{R}\,(n+1)$$

The softmax function on the affinity matrix is taken row-wise and column-wise respectively for the Context-to-Query and Query-to-Context attention to produce the attention weights across the query for every word in the context passage and vice versa. The attention weight/score is them dot product with the context and query hidden states to produce the attention vector.

Then a second-level attention output is computed as follows:

$$S = [Q_{proj}; QtoC]\; softmax(L_{i:}) \in \mathbb{R}\,(n+1 \;\times 2h)$$

This gives us the co-dependent representation of the query and the context passage. As the last step, another bi-directional LSTM was used to encode the blended representation of $[C; S]$ as the final output of this dynamic co-attention layer.

3.4.2 Self Attention

In the R-NET paper, the authors proposed computing the attention against attention stage output itself. We employed a similar stage after the dynamic co-attention output described above and it has shown improvement in performance.
However, the original R-NET paper used a more complex additive attention mechanism, which consumes huge amount of memory and result in very slow training time. Instead, we applied a much simpler multiplicative attention calculation mechanism described below:

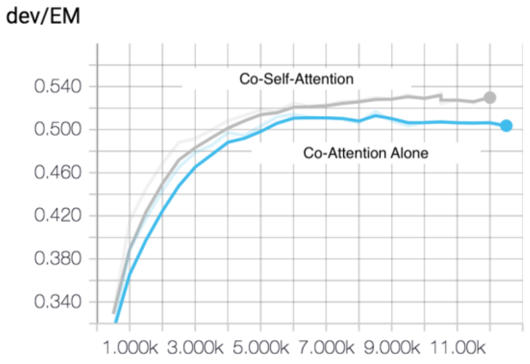
$$T = relu(WA + b)$$

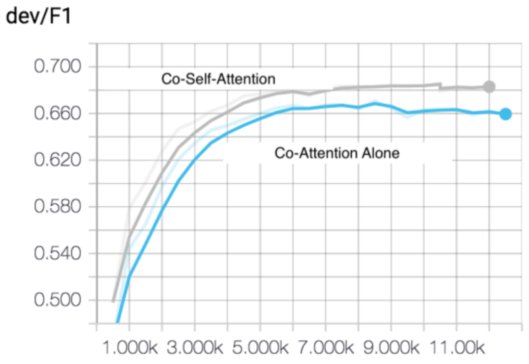
Where W and b are trainable parameters and A is the input to this layer, in this case the output of previous co-attention layer.

$$Attn = (A\, softmax(T^t T))$$

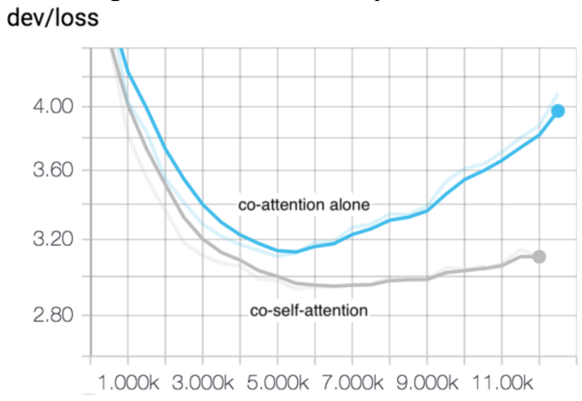
Lastly, the Attn output is concatenated with T and processed with another bi-directional LSTM layer that produce the final output of the attention layer.

The effect of this additional self-attention layer can be shown in the following graph:





One way to explain the performance gain with the self-attention layer is that it delayed overfitting and the rise of development set loss as illustrated in the following plot:

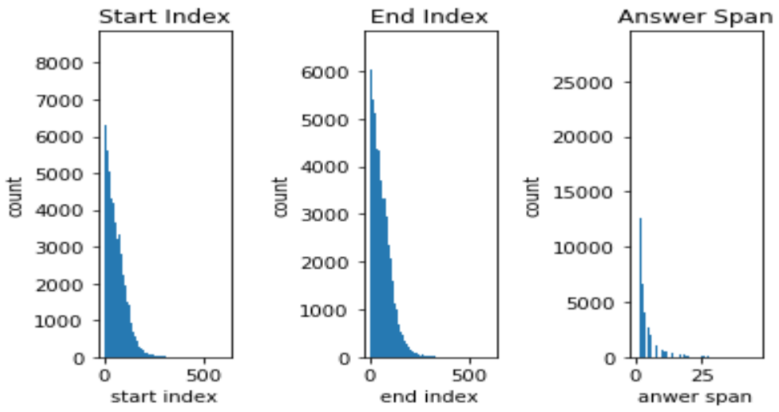


3.4.3 Output Layer

We have attempted to implement PointerNet output layer described in the BiDaf paper [3] but was unsuccessful. Out output layer is a straight forward feed-forward dense network with a softmax layer at the output to predict probability of p_{start} *and* p_{end} for the answer in the context passage respectively.

One short-coming of the independent prediction of the simple output layer is that it predicts start and end index completely independently and thus may result in cases where the start index is bigger than the end index.

Upon analysis of the training data and the ground truth answer span of the training data, we found that most of the answers have a span of 30. Therefore, similar to Chen et al.[5], when computing the final answer, we chose the start and end pair that maximize the following expression: $P_{start}(i) \times P_{end}(j)$ where $i \leq j \leq i + 30$.



3.4.4 Other Experiments

We have also experimented and implemented the BiDAF attention mechanism and it's modeling layer and have used that as a substitute of the attention layer in our model. This implementation resulted in similar but slightly worse results in terms of EM/F1 scores compared to the previous discussed Co-attention and Self-attention mechanism.
The BiDAF implementation is used in the final ensemble model.

In the ensemble model, we first run each model separately on the dev dataset to produce their answers. Then a simple voting mechanism where the most common answer is picked as the final answer for the ensemble model.

4 Analysis

In this section we perform some analysis of our model.

4.1 Hyper-parameter tuning

There are a lot of hyper-parameters in any deep neural network architecture. In this model, these parameters include max context length, max query length, learning rate, word-embedding size per word, hidden layer size for attention and RNNs etc.

We did not perform an systematic or exhaustive hyper-parameter search instead we picked these parameters for the co-attention and self-attention combo model after some experiments and have shown relatively good performance:

Parameter	Value
learning rate	0.001
max context length	500
max query length	30

hidden size	100
dropout rate	0.15

To prevent the model from overfitting we used dropout technique at each layer of the neural network.

4.2 Error Analysis

Our single model achieved a F1 score of 74.053 and a EM score of 63.415.

Our ensemble model achieved a F1 score of 75.278 and a EM score of 65.193

For difference categories of queries on the development dataset, our model performs like the following table:

Query key word	F1	EM	Count
What	73	62.06	5846
Who	80.28	73.74	1093
When	86.61	81.61	707
How	76.70	66.91	1224
Which	76.67	68.05	698
Where	73.55	60.54	474
Why	66.33	66.33	155

From the per query category analysis, we can see that the model performs the best on "when" category queries and perform the worst on "why" category queries. This is understandable since 'why' category typically demands more "understanding" or "reasoning" of the context and query, whereas the 'when' category's answer typically is clearly identifiable in the context passage and require less 'understanding' of the rest of the context and the query.

5 Conclusion

In this report, we presented a model with Co-attention and Self-attention mechanisms as well as other models experimented in order to tackle the Stanford Question Answering Challenge. After experiments with various attention mechanisms, we discovered that a combination of the dynamic co-attention and self-attention mechanism result in the best performance for a single model.

The tensorflow code for this project is available on Github: https://github.com/yuzhoul1991/squad_challenge

Acknowledgements

We would like to thank the course CA for their help on answering questions on the course platform and supporting of technical issues. Also would like to thank the course CA for providing a working baseline model that we can implement improvements on. We also appreciate the lectures and advises given by the instructor Richard Socher which helped a lot in understanding the research papers that we attempted to implement.

References

[1] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. CoRR, abs/1606.05250, 2016. [2] Bower, J.M. & Beeman, D. (1995) *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System*. New York: TELOS/Springer-Verlag.

[2] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. arXiv preprint arXiv:1611.01604, 2016.

[3] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. arXiv preprint arXiv:1611.01603, 2016.

[4] Natural Language Computing Group, Microsoft Research Asia. R-NET: Machine Reading Comprehension with Self-Matching Networks

[5] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. arXiv preprint arXiv:1704.00051, 2017.