Question Answering with the SQuAD

Wayne Lu
Computer Science
Stanford University
waynelu@stanford.edu

Abstract

The reading comprehension problem involves training a machine system to process text such that it is able to accurately answer queries about the text. In this project, we investigate, train, and evaluate recurrent neural network architectures to address this problem. Architecture modules are largely drawn from the Bi-Directional Attention Flow (BiDAF) model introduced by Seo et al. [4]. The performance of the networks are evaluated on the Stanford Question Answering Dataset (SQuAD), with results similar to the original non-ensembled BiDAF model.

1 Introduction

The reading comprehension problem is a form of natural language understanding which involves training a machine system to answer questions about a body of text. That is, given an query and context pair, the system should return an answer either directly quoted from the context or generated using a language model. In this way, the reading comprehension problem serves as a measure of how well machine systems are able to process and understand human text and has practical applications in quickly extracting answers to human queries from large bodies of text. In the case of this project, our systems extract portions of the context as answers, which simplifies the problem.

In this project, we begin with a basic baseline neural network model and iteratively upgrade different portions of the model to improve performance. The basic components of the model consist of a token embedding layer, which converts tokens to dense vector representations better suited for neural networks; an encoding layer, which processes the query and context; an attention layer, which allows the system to focus on portions of the context which are most relevant; and finally an output layer, which produces probability distributions for the start and end of the answer span.

2 Background/Related Work

SQuAD is a publicly available dataset with a public leaderboard displaying test F1 and exact match scores. Human performance is quantified as an exact match (EM) score of 82.304 and a F1 score of 91.221. The leaderboard has attracted model submissions from both academic and industry research departments, with many systems producing near-human performance.

Notably, many models introduce novel attention mechanisms such BiDAF's context-to-query and query-to-context attention [4], the Hybrid AoA Readeer's attention-over-attention mechanism [1], and the Dynamic Coattention Network's coattention mechanism [5].

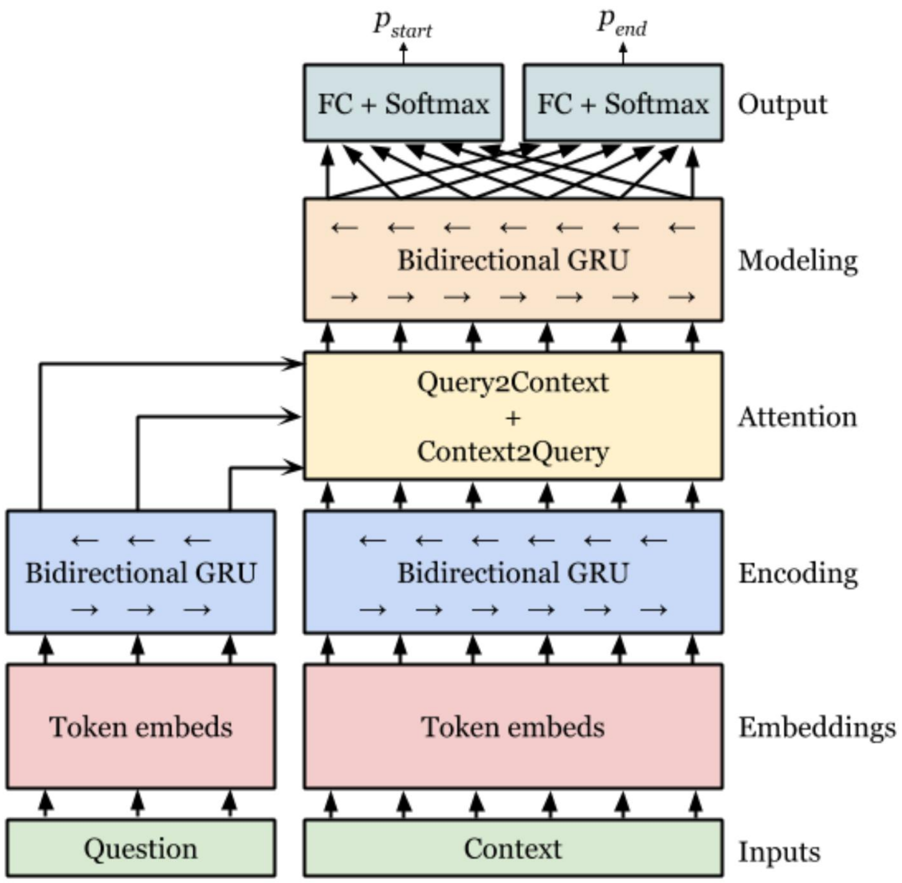

Figure 1: Diagram of the neural network architecture used in this project.

3 Approach

3.1 Token embedding layer

Dense vector representations of the input tokens are generated at both the word level and the character level. Word-level vectors use 100-dimensional fixed pretrained GloVe embeddings [2]. Character-level vectors are generated by first converting the token characters to trainable 20-dimensional embeddings. The character embeddings are then fed through two 1-D convolutional layers with 100 filters, window size 11, and stride 1. The first convolutional layer applies a ReLU non-linearity while the second layer maintains a linear activation. The outputs of the second layer are then passed through a max pooling layer to create a 100-dimensional dense representation of the token at the character level. The word-level vector and the character-level vectors are then concatenated to create a 200-dimensional dense representation of the token.

Character vectors are only assigned to the set of printable ASCII characters (accessed via `string.printable` in Python 2.7). No normalization of Unicode characters is performed, such as converting ”è” to ”e” or Romanization of non-Latin scripts. Characters outside of the set of printable ASCII characters are all assigned the same UNK vector, similar to how unknown tokens are handled with word-level vectors.

All queries, contexts, and tokens are padded or truncated to a fixed size, as is required for batch processing. Queries are fixed to $M = 30$ tokens, contexts are fixed to $N = 400$ tokens, and tokens are fixed to 20 characters. These limits allow approximately 99% of the training examples to remain untruncated. A visual representation of this system can be found in Figure 2.

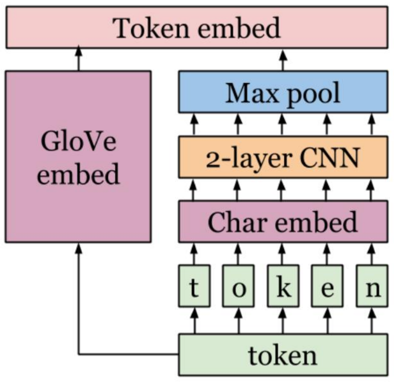

Figure 2: Diagram of the architecture of generating token embeddings.

3.2 Encoding layer

An initial encoding of the query and context dense representations is performed by feeding the query and contexts independently through a bidirectional GRU. The forward and backward GRU cells each have a hidden state size $h = 200$, resulting in a query encodings $q_1, \ldots, q_M \in \mathbb{R}^{2h}$ and context encoding $c_1, \ldots, c_N \in \mathbb{R}^{2h}$.

3.3 Bidirectional attention layer

With the query and context encodings, we then apply the bidirectional attention mechanism described by Seo et al. [4]. As described in the project handout, given context encodings $c_1, \ldots, c_N \in \mathbb{R}^{2h}$ and query encodings $q_1, \ldots, q_M \in \mathbb{R}^{2h}$, we calculate a similarity matrix $S \in \mathbb{R}^{N \times M}$ such that

$$S_{ij} = w_{\text{sim}}^T [c_i; q_j; c_i \circ q_j]$$

where $w_{\text{sim}} \in \mathbb{R}^{6h}$ is a trainable weight vector.

3.3.1 Context-to-query attention (C2Q)

Given the similarity matrix S, we take a row-wise softmax to produce attention distributions $\alpha^1, \ldots, \alpha^N \in \mathbb{R}^M$. We then use the C2Q attention distributions to take a weighted sum across the query encodings, which produces C2Q attention outputs $a_1, \ldots, a_N \in \mathbb{R}^{2h}$. That is,

$$a_i = \sum_{j=1}^{M} \alpha_j^i q_j \quad \forall i \in \{1, \ldots, N\}$$

3.3.2 Query-to-context attention (Q2C)

Taking the row-wise max over the similarity matrix S, we obtain a vector $m \in \mathbb{R}^N$. Taking the softmax of m, we then obtain a attention distribution $\beta \in \mathbb{R}^N$. Using β, we then take a weighted sum across the context encodings to produce the Q2C attention output $c' \in \mathbb{R}^{2h}$. That is,

$$c' = \sum_{i=1}^{N} \beta_i c_i$$

Finally, we combine the context encodings, the C2Q attention outputs, and the Q2C attention output to produce query-aware context encodings

$$b_i = [c_i; a_i; c_i \circ a_i; c_i \circ c'] \quad \forall i \in \{1, \ldots, N\}$$

3.4 Modeling layer

As with the BiDAF model, we pass the query-aware context encodings $b_1, \ldots, b_N$ through a second bidirectional GRU in order to encode the context after it has taken the query into account. As with the first bidirectional GRU, each cell has a hidden size of $h = 200$, resulting in modeling output $m_1, \ldots, m_N \in \mathbb{R}^{2h}$.

3.5 Output and prediction layer

The modeling outputs are flattened and concatenated, then fed through two independent fully connected layers with softmax to generate p_{start} and p_{end}, which are probabilities distributions over the context for the start and end of the predicted answer. At test time, the actual predicted start and end are chosen as the pair

$$(i, j) = \underset{i \le j \le i+15}{\arg\max}\, p_{\text{start}}(i) p_{\text{end}}(j)$$

3.6 Optimization and regularization

Loss is calculated as the summed cross-entropy loss of the start and end predictions. Trainable weights are optimized using the Adam optimizer. The initial learning rate is set to 0.001 and decayed by 0.1 every 10,000 steps, which correspond to times when the development loss has plateaued.

Regularization of the model is achieved by applying dropout with dropout rate of 0.15 to the outputs of the encoding and attention layers.

4 Experiments

4.1 Dataset

The networks are trained and evaluated on the Stanford Question Answering Dataset (SQuAD), which consists of queries, paragraphs from Wikipedia, and human-generated answers crowdsourced via Amazon Mechanical Turk [3]. To account for variation in human answers, each example has three answers. After pre-processing and cleaning of the data, there are 86,326 training examples and 10,391 validation examples.

4.2 Model components

The final submitted for testing was created by iteratively adding improvements to the provided baseline model. In order of implementation, the improvements consist of: the BiDAF attention mechanism, character-level token embeddings, smarter test-time answer span selection, the BiDAF modeling layer, and decayed learning rate. Each additional architecture model produced noticeable improvements in performance, with the BiDAF layers having the most impact.

Table 1: Dev F1 and EM scores of different models at training time

BiDAF attention	Char-level embed	Span selection	BiDAF modeling	LR decay	F1	EM
No	No	No	No	No	40.375	29.352
Yes	No	No	No	No	46.168	34.081
Yes	Yes	No	No	No	49.241	36.307
Yes	Yes	Yes	No	No	54.424	40.572
Yes	Yes	No	Yes	No	67.654	52.902
Yes	Yes	Yes	Yes	No	69.566	54.048
Yes	Yes	Yes	Yes	Yes	70.839	55.713

4.2.1 Character-level token embeddings

Interestingly, the character-level token embeddings improved the dev F1 and EM scores by only 3.1 and 2.2 points, respectively. However, this is not surprising considering that character-level embeddings are meant to act as a fallback for when word-level embeddings are not available. Given that the SQuAD dataset consists of paragraphs taken from Wikipedia and the pre-trained GloVe vectors are pretrained on the Wikipedia 2014 dataset, it is likely that there is a large amount of overlap between the vocabularies.

4.2.2 Test-time span selection

Smarter test-time span selection greatly improved performance for the model which does not include the BiDAF modeling layer, increasing F1 and EM by 5.1 and 4.3, respectively. However, the effect becomes less pronounced after the BiDAF modeling layer is implemented, resulting in F1 and EM increases of 1.9 and 1.1, respectively. This suggests that the BiDAF modeling layer is better able to capture the relationship between the start and end indices than directly passing the attention outputs to the softmax prediction layers. However, as smarter span selection still leads to improved performance, the relationship is not completed captured.

Smarter test-time span selection also has the disadvantage of adding a non-neural component to the network. That is, the actual predictions require extra processing and the network's loss function does not accurately model the dependence between the start and end predictions. This is produces the interesting effect of improving evaluation metrics while not changing the training or dev loss curves of the networks.

4.2.3 BiDAF attention and modeling layers

The BiDAF attention and modeling layers produced the largest improvements in evaluation metrics. Implementing the attention layer improved F1 and EM by 5.8 and 4.7 points, respectively, while implementing the modeling layer then improved the scores by 15.1 and 13.5 points, respectively. These results suggest the importance of discovering attention and encoding mechanisms which are able to properly capture the relationship between the query and the context.

4.2.4 Decayed learning rate

The final boost in performance is produced by using learning rate decay, leading to slight increases in F1 and EM scores of 1.3 and 1.7, respectively.

4.3 Final model evaluation

As noted in the project handout, the training time F1 and EM scores are different than official evaluation scores, as official evaluation takes all three human-generated answers into account. Test evaluation of our final model produced an F1 score of 77.082 and an EM score of 67.408. Looking at sample predictions of the model suggests a few common reasons for prediction errors:

1. Phrases which are similar to the question, but semantically ambiguous without additional context. For example, the question ”What was produced at Tesla?s company?” results the answer ”Tesla Electric Light & Manufacturing” likely because the system fails to recognize the sequence as a named entity rather than a product.
2. Query/context pairs with many unknown tokens. For example, in the question ”What do platyctenida use their pharynx for?”, the system did not recognize the token ”platyctenida” and many other tokens in the context were also unknown. This can degrade the performance of the system because it is not able to properly process the context.
3. Queries which require deeper inference about the context. For example, for the question ”What was the protest in Antigone about?”, the context never directly mentions a protest but instead discusses how Antigone ”defies Creon, the current king of Thebes, who is trying to stop her from giving her brother Polynices a proper burial.” Thus, answering this question requires resolving the connection that Antigone is protesting against Creon because of ”giving her brother Polynices a proper burial,” which is the correct answer. Instead, the system predicts ”civil disobedience,” as that is closely related to protesting.

4. Predictions which are slightly too long or too short. This is likely because the system does not explicitly have a notion of what constitutes a complete semantic unit causing it to either extend past the unit or truncating it. Because ground truth and predicted answers are both typically very short, being off by even one token can have a large impact on the F1 score.

5 Conclusions and Future Work

The results of the evaluation suggest that performance could be improved by injecting more semantic information into the dense representations of the tokens. For example, using a named entity recognition system to annotate the query and context could lead to better disambiguation (such as in the case of the first error reason). The system would also benefit from a larger vocabulary, as shown by the second error reason. Though character-level embeddings can act as a stopgap measure, they are not able to capture as much semantic information as word vectors trained on large corpuses. The third and fourth error reasons are less obvious to address, and as modeling these complex semantic relationships in text is the core problem of natural language understanding. Finding ways to encode sentence- and paragraph-level information such as parse trees and coreference resolution would likely be useful in allowing the system to learn more complex relationships in the text and warrants more investigation.

It should also be noted that minimal hyperparameter tuning was performed in this project, as we were more interested in the effects of broad architecture and training changes on the performance of the system rather than trying to push a fixed architecture to its limits. Experimenting with stronger dropout or other forms of regularization would help mitigate the overfitting problem. Increasing embedding and hidden sizes throughout the network would likely also improve performance, though at the cost of higher runtimes and increased memory demands.

References

[1] Yiming Cui, Zhipeng Chen, Si Wei, Shijin Wang, Ting Liu, and Guoping Hu. Attention-over-attention neural networks for reading comprehension. *arXiv preprint arXiv:1607.04423*, 2016.

[2] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[3] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

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

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