Exploring Attention Mechanisms for Reading Comprehension

Noah Makow
Department of Computer Science
Stanford University
Stanford, CA 94301
nmakow@stanford.edu

Abstract

Reading comprehension has long enticed the natural language processing community as a task that requires a deep understanding of the text at hand. In this paper we explore several recent developments in reading comprehension, focusing on varying attention mechanisms. We observe that while these attention mechanisms can greatly boost performance in themselves, we see that they are not the only improvements required to approach human-level performance on the SQuAD dataset. We conclude that future work on reading comprehension should consider both implementing novel attention mechanisms as well as new ways to combine distinct extensions into new, high-performing hybrid architectures.

1 Introduction

Reading comprehension has long enticed the natural language processing community as a task that requires a deep understanding of the text at hand. In order to answer questions, one must derive a representation (or "understanding") of both the question and relevant passage, and then be able to relate the two pieces of text to one another to extract an answer. As a more complex task than simple sequence to sequence generation, reading comprehension systems are built from the same building blocks but require more specialized techniques and architectures to perform well. The publication of the Stanford Question Answering Dataset (SQuAD) [4] in 2016 opened the gates, welcoming a wealth of research into designing more complex neural architectures specialized to accomplish this task.

To define the reading comprehension task formally, we are presented with a context (passage) and a question. The answer to the question is represented as a span (or substring) from the context.

In this paper we explore several recent developments in reading comprehension, focusing on varying attention mechanisms. In particular, we investigate self-attention [2] and bidirectional attention flow [5], comparing the two methods and inspecting their results. We also explore a number of architectural designs and hyperparameter settings and report performance.

2 Background & Related Work

The Stanford Question Answering Dataset (SQuAD) is a recently presented reading comprehension dataset consisting of 100,000+ questions posed by crowdworkers on a set of Wikipedia articles. The answer to each question is a segment of text from the original passage. The SQuAD dataset serves as a large, realistic dataset that has driven research into the task of reading comprehension, or the ability to read texts and then answer questions about it. Simple baselines achieve in the range of 50-60% F1 score, while humans are able to achieve a much higher F1 score of 86.8%. [4]

The SQuAD dataset has inspired a wealth of research on developing various neural architectures to solve the reading comprehension task. Central towards this task are deep recurrent neural networks incorporating various forms of attention. Recent work has explored different types of attention to be used at different stages in the reading comprehension pipeline. Three of these ideas explored in this paper include bidirectional attention flow, self-attention, and answer pointer networks.

Bidirectional attention flow (BiDAF) extends basic attention mechanisms that typically focus on small portions of the context by introducing a bidirectional attention process by which both the context and query can attend to one another. This bidirectional flow of attention enables query-aware context representations. Coupled with other extensions, BiDAF still achieves state-of-the-art results on the SQuAD leaderboard. [5]

Self-attention is a concept introduced in Microsoft Research's R-Net. The self-attention mechanism attempts to refine the context representation by matching the passage against itself, effectively encoding information from the entire passage. R-Net employs other techniques that are less the focus of this paper such as gated attention-based recurrent networks. Finally, R-Net utilizes pointer networks in the output layer which we discuss below. [2]

Pointer networks were first applied to the SQuAD task in Match-LSTM and Answer Pointer by Wang et al. The core idea of pointer networks is to condition the probability distribution of the answer's end location on the answer's start location probability distribution. The details of the model are described further below. This output layer is used in most top performing SQuAD models. [6]

3 Approach

In this paper we implement several of the core components of a number of different state-of-the-art SQuAD systems. The general structure of our architecture is described in five layers.

3.1 Embedding Layer

We implement both word-level and character-level embeddings. Word embeddings map each word to a high-dimensional vector space. We use fixed pre-trained GloVe [3] word vectors to obtain a fixed-size vector for each word. Character embeddings also map each word to a high-dimensional space. Let $\{x_1, \ldots, x_N\}$ and $\{y_1, \ldots y_M\}$ represent the words in the input context and question, respectively. We use convolutional neural networks to obtain character-level embeddings. [1] We represent each character as a vector, the dimension of which can be considered as the input channels to a one-dimensional CNN. The outputs are max-pooled elementwise over the number of filters to obtain a fixed-size vector for each word.

For a sequence of word embeddings $\{x_w^{(1)}, \ldots, x_w^{(N)}\} \in \mathbb{R}^l$ and corresponding character embeddings $\{x_c^{(1)}, \ldots, x_c^{(N)}\} \in \mathbb{R}^k$, we concatenate the corresponding embeddings to result in the final embeddings $\{[x_w^{(1)}; x_c^{(1)}], \ldots, [x_w^{(N)}; x_c^{(N)}]\} \in \mathbb{R}^{l+k}$. Note that we do not include character embeddings in all of the experiments conducted – see the results table for the details of each model.

3.2 Encoding Layer

For each SQuAD example (context, question, answer), the context is represented as the sequence of embeddings $\{x_1, \ldots, x_N\} \in \mathbb{R}^d$ and the question is represented as the sequence of embeddings $\{y_1, \ldots, y_M\} \in \mathbb{R}^d$. These embeddings are fed into a shared 1-layer bidirectional GRU:

$$\{c_1, \ldots, c_N\} = \text{biGRU}(\{x_1, \ldots, x_N\})$$
$$\{q_1, \ldots q_M\} = \text{biGRU}(\{y_1, \ldots, y_M\})$$

The bidirectional GRU produces a sequence of states the same length as the input sequence, where *hidden states* $c_i, q_i \in \mathbb{R}^{2h}$ are the concatenation of the forward and backward hidden state at timestep i for the context and question, respectively.

3.3 Attention Layer

After encoding the context and question as hidden states, the next step is the attention layer. We use two alternative attention mechanisms mentioned above – bidirectional attention flow and self-

attention. At its core, this layer enables each context word to pay selective attention to each query word, producing a set of query-aware feature vectors for each word in the context.

3.3.1 Bidirectional Attention Flow (BiDAF)

The central idea of bidirectional attention flow is that attention should flow both from the context to the question and from the question to the context. [5] BiDAF serves as a substitute for a more basic context to query attention model such as the one mentioned above. In the BiDAF layer, we have context hidden states $c_1, \ldots, c_N \in \mathbb{R}^{2h}$ and question hidden states $q_1, \ldots, q_M \in \mathbb{R}^{2h}$. First we compute the *similarity matrix* $S \in \mathbb{R}^{n \times m}$ which contains the similarity score S_{ij} for each pair (c_i, q_j):

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

The first step is context-to-question attention:

$$\alpha_i = \text{softmax}(S_{i,:}) \in \mathbb{R}^M$$

$$a_i = \sum_{j=1}^{M} \alpha_j^i q_j \in \mathbb{R}^{2h}$$

Then we perform question-to-context attention:

$$m_i = \max_j S_{ij} \in \mathbb{R}$$

$$\beta = \text{softmax}(m) \in \mathbb{R}^N$$

$$c' = \sum_{i=1}^{N} \beta_i c_i \in \mathbb{R}^{2h}$$

The final output for each context location i is $b_i = [c_i; a_i; c_i \circ a_i; c_i \circ c'] \in \mathbb{R}^{8h}$.

3.3.2 Self-Attention

Suppose we have some sequence of representations $v_1, \ldots, v_N \in \mathbb{R}^\ell$ with v_i corresponding to context location i. Each v_i attends to *all* the $\{v_1, \ldots, v_N\}$:

$$e_j^i = v^T \tanh(W_1 v_j + W_2 v_i) \in \mathbb{R}$$

$$\alpha^i = \text{softmax}(e^i) \in \mathbb{R}^N$$

$$a^i = \sum_{j=1}^{N} \alpha_j^i v_j \in \mathbb{R}^\ell$$

Note that W_1, W_2, and v are trainable weight tensors. We then concatenate the self attention output a_i to v_i and pass these as input into a bidirectional GRU to obtain a new set of hidden states:

$$\{h_1, \ldots, h_N\} = \text{biGRU}(\{[v_1; a_1], \ldots, [v_N; a_N]\})$$

The representations $\{h_1, \ldots, h_N\}$ are the output of the self-attention layer. This layer is inserted after the basic or bidirectional attention layer.

3.4 Modeling Layer

The modeling layer is borrowed from the BiDAF model, and in our experiments is only used in conjunction with the BiDAF layer. The input to the modeling layer is the query-aware representations of the context words from the attention layer. The output of the modeling layer captures the interaction among context words conditioned on the query. We use a bidirectional GRU and pass the output to the output layer.

3.5 Output Layer

A basic output layer simply takes the softmax over each position in the context, producing a probability distribution for both the start and end positions (which we denote p_{start} and p_{end}. Note that for the actual prediction we limit the argmax such that the end position must be greater than the start position and within 20 positions.

3.5.1 Pointer Networks

In a number our experiments we use the more complex answer pointer network as our output layer. [6] The answer pointer is an RNN run for exactly two timesteps. On the first timestep, the answer pointer attends to $h_1^r, \ldots, h_N^r$ (the representations for each context location), producing an attention distribution $\beta_s \in \mathbb{R}^N$ and an attention output $a_s \in \mathbb{R}^\ell$. We use β_s as p_{start}. We then feed the attention output a_s as input to the second step and use the RNN hidden state to attend to $h_1^r, \ldots h_N^r$, producing an attention distribution $B_e \in \mathbb{R}^N$ which we use as p_{end}.

4 Experiments

4.1 Dataset

Towards our task of reading comprehension, we make use of the Stanford Question Answering Dataset (SQuAD) [4]. The SQuAD dataset consists of questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text, or *span*, from the corresponding article. SQuAD contains 107,785 question-answer pairs on 536 articles.

In the sections that follow, we summarize the SQuAD dataset with a number of statistics collected in experiments detailed in the supplementary materials attached with this report. We provide Jupyter notebooks for extracting and visualizing these statistics in the `notebooks/` directory of our repository in our Supplementary Materials.

4.1.1 Question, Context, and Answer Lengths

First we inspect the distributions of question, context, and answer span lengths. We notice that the vast majority of context are under length 400 and all questions are under length 30. For certain model extensions the context length becomes an import factor in terms of memory usage, and limiting this value to 400 helps to keep memory usage low while sacrificing little performance.

Next, we visualize the distributions of start, end and answer positions. Most start and end positions fall within the first 250 tokens of the context, and the majority of answer spans are under length 20. The first observation confirms our idea that we can limit the context length without eliminating many of true answer spans. The second observation is used in our output layer when predicting the end position. Instead of taking the argmax over the entire context, we take it over only the 20 tokens that follow the predicted start position.

4.1.2 Character-Level Statistics

Character embeddings are a useful addition to our model. In order to build effective embeddings, it is necessary to understand the set of characters that are most commonly used in the dataset as well as the distribution of the lengths of words. We observe that most words contain less than 15 characters. Further, we select the set of top 50 most common characters as the vocabulary of our character embeddings.

4.2 Model Details

Unless otherwise specified, we use single-layer bidirectional GRUs as our choice of recurrent neural network cell in all layers. We choose GRUs for increased computational efficiency with comparable performance to LSTMs. We use the Adam optimizer beginning at a learning rate of 0.001. We constrain the predicted end position to be within 20 tokens of the predicted start position. We begin with a simple baseline architecture consisting of a word embedding layer, an encoding layer, a basic context-to-query attention layer, and a simple softmax output layer for the start and end position prediction. We use dropout in between layers with dropout probabilities ranging from 0.15 to 0.25. In the results section below, we replace or add only the particular features mentioned to this baseline model.

Model	Dev F1	Dev EM	Train F1	Train EM
Baseline	43.4	34.3	54.3	64.5
Self/CE	62.0	47.2	73.7	60.3
Self/Pointer/CE	57.5	42.1	60.9	45.3
BiDAF/Model/CE	72.3	61.4	**78.8**	**65.6**
BiDAF/Model/CE (deep)	**74.8**	**64.2**	77.9	62.5

Figure 1: Summary of F1 and EM scores for top performing models. Each model contains only the mentioned extensions in addition to our baseline. Note that the deep BiDAF/Model/CE has the same components as the shallow version, but with an extra bidirectional RNN layer in the encoding and modeling layers.

4.3 Evaluation

We use two quantitative evaluation metrics to measure the performance of our model: *exact match (EM)* and *F1 score*. EM is a binary measure of whether the model output exactly matches the ground truth. F1 is a less strict metric – the harmonic mean of precision and recall of the tokens in the predicted answer and ground truth. Since the SQuAD dataset has three human-provided answers for each question, the official evaluation metrics take the maximum F1 and EM scores across the human-provided answers. The final EM and F1 scores are the average scores over the entire evaluation dataset.

4.4 Results

4.4.1 Quantitative Results Summary

In Figure 1, we present the performance on train, dev, and test splits of our top performing models. We are able to achieve the best performance with a BiDAF model that contained a modeling layer and character embeddings with a dev F1 score of 74.8 and dev EM score of 64.2. Note that these results should not suggest that BiDAF strictly outperforms self-attention – rather, self-attention may require other extensions described from the R-Net paper (such as the gated RNN) to achieve comparable performance.

Figure 1 illustrates the clear impact of adding more complex neural attention mechanisms to a simple baseline model. Compared to our baseline which includes a simple attention layer, self attention and bidirectional attention flow each account for around a 20% increase in F1 score. Certainly, further extensions are required to be on par with state-of-the-art SQuAD systems, however these two examples of attention serve as a solid starting point and are actually the foundation of many current systems.

4.4.2 Inspecting Attention Distributions

From our experiments, it is evident that complex attention mechanisms are the key for enabling effective neural reading comprehension systems. These attention mechanisms allow for complex interactions among the context words, question words, and between them (with mechanisms like BiDAF). While obtaining state-of-the-art results will generally require implementing extensions outside of attention, the attention mechanisms remain as one of the foundational components and core drivers of strong performance.

Attention also has the side-effect of capturing useful linguistic mechanisms without explicitly modeling them. In Figure 2 we analyze the context-to-question output of the BiDAF layer in our top-performing model. Interestingly, the attention mechanism seems to perform automatic coreference resolution between the context and the question. These types of unintended side-effects highlight the immense representative ability that attention mechanisms provide.

4.4.3 Common Pitfalls

Before concluding, we highlight a number of common pitfalls our models fall into. While there are many ways to categorize such errors, we highlight three common cases that appeared repeatedly

QUESTION: on what streets is the abc headquarters located?

Context word	...	the	network	is	headquartered	on	...
Most attended query word	...	the	**abc**	is	headquarters	on	...

Figure 2: An illustration of the context-to-query attention mechanism. For each word in the context, we visualize the most-attended query word. We notice that there seems to be automatic coreference resolution between nouns in the context and those in the query – for example, our model correctly identifies that "the network" refers to "ABC".

throughout our experiments. We should note, of course, that these categories are not exhaustive and cover only a small, readily-interpretable subset of the errors encountered by our model.

4.4.4 External Knowledge

Occasionally, certain questions posed in the SQuAD dataset refer to or require external knowledge that is not presented within the context passage. As our model is limited in its knowledge representation to the context and questions fed to it as training data, it struggles to extrapolate or derive logical conclusions and/or understandings without directly training on that information.

For example, consider the question "which architect, *famous for designing London's St. Paul Cathedral,* is represented in the Riba Collection?" The context passage contains a list of artists in the Riba Collection without mentioning their other accomplishments. As a result, our model picks a random name rather than the correct one. Again, unless the fact that the architect designed London's St. Paul Cathedral was explicitly contained in our training data, it would be very difficult for our model to successfully reason about this.

4.4.5 Token-Level Logical Reasoning

Some questions are difficult in the sense that they require higher level ("meta"-level) reasoning about the context or tokens in the context. Consider the following question: "what is the only district in the cbd *to not have 'downtown' in it's name*?" Our model fails to answer this question as it is unable to reason about which locations have "downtown" in their name and which do not. As a result, our model reverts to what is likely the most represented location found in the context, Southern California, rather than the true answer. These types of questions prove tricky as the answer is not fully contained in the text itself but requires some level of reasoning regarding the text.

4.4.6 Phrase-Boundary Ambiguities

Our model struggles in certain cases when it is ambiguous where a phrase truly ends. For example, consider the question "what can orthogonal forces be when there are three components with two at right angles to each other?" with the true answer of "three-dimensional". Our model outputs "three-dimensional with the third component being at right-angles". That is, our model runs on past the end of the phrase, despite the answer that it output being one coherent phrase in itself. The model struggles to determine when the phrase that is being asked for actually ends, as it continues naturally in the context.

For another example of this effect, consider the question: "what happened to the apollo program in for the rest of 1970 after the incident regarding apollo 13?". The correct response is "grounded", but our model answers "grounded again", adding an extra word.

Note that these mistakes do not suggest that the model is incredibly far off from the correct answer, however they do penalize performance in terms of F1 and EM. We should note that some of these phrase ambiguities may exist even among the human annotaters. As a result, this problem may be less significant when we take the max F1/EM score over all three human-provided answers.

5 Conclusion

Great progress has been made towards building automated reading comprehension systems. In this paper, we have reviewed some of the fundamental neural components – particularly, attention

mechanisms – that have enabled great improvements in performance on this task. Such attention mechanisms can allow the model to represent complex interactions between the context and query words and among themselves. Future work in this space will include developing novel attention mechanisms, such as the recently introduced dynamic coattention. [7] While these attention mechanisms can greatly boost performance in themselves, we see that they are not the only improvements required to approach human-level performance on the SQuAD dataset. The highest performing SQuAD models today use a combination of approaches from BiDAF, R-Net, and many other high-performing models, such as novel RNN structures, deeper hidden layers, feature engineering, and model ensembling. Future work should look at the benefits and pitfalls of each of these extensions independently and consider novel ways to combine them into new, high-performing hybrid architectures.

References

[1] Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882.*

[2] Natural Language Computing Group, Microsoft Research Asia (2017). R-Net: Machine Reading Comprehension with Self-Matching Networks.

[3] Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).*

[4] Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250.*

[5] Seo, M., Kembhavi, A., Farhadi, A., & Hajishirzi, H. (2016). Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603.*

[6] Wang, S., & Jiang, J. (2016). Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905.*

[7] Xiong, C., Zhong, V., & Socher, R. (2016). Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604.*