# Standard Attention versus Attention-Based Recurrent Networks for Reading Comprehension

Stanford CS224N Default Project

**Eric Hatch**
Department of Computer Science
Stanford University
enchatch@stanford.edu

## Abstract

This paper describes an Attention-Based Recurrent Network model containing the self-attention mechanism from Microsoft Research Asia's R-Net (2017) with the encoding and output layers similar to Seo et al.'s Bi-Directional Attention Flow (BiDAF) model. Given an input context paragraph and question about the context paragraph, the model locates the answer to the question within the context paragraph by predicting the start token and end token of the answer string within the context paragraph.

The paper compares the attention mechanisms of BiDAF and R-Net, discussing their upsides and drawbacks. This paper also analyzes the challenges of implementing an Attention-Based Recurrent Network model with minimal information loss and quick training time. The model attains an EM score of 55.790 and an F1 score of 59.537 on the Stanford Question Answering Dataset 2.0 (SQuAD 2.0) (Rajpurkar et al., 2016). This metric shows that the Attention-Based Recurrent Network model does not perform much better than the BiDAF baseline, although there are specific improvements that could be made to the model as discussed in section 6.

## 1 Key Information to include

- Mentor: Vincent Li
- External Collaborators (if you have any): N/A
- Sharing project: N/A

## 2 Introduction

The goal of question answering (QA) is to build systems that automatically answer questions posed by humans in a natural language. QA systems are ubiquitous in society today. Examples of QA systems include search engines and virtual assistants both of which have revolutionized life as we know it.

The Information-Retrieval (IR) question answering (QA) task is as follows: given an input question, the task is to (1) locate a context paragraph containing information containing the input question, and (2) predict an answer span from the context paragraph. Task (2), also known as reading comprehension, is the focus of this paper. Formally, given a question $Q$ and a context paragraph $C$, our task is to select $q_1$ and $q_2$, the start and endpoints (respectively) of the answer $A$ to $Q$ inside context paragraph $C$.

Breakthroughs have been made in the field of neural-network based reading comprehension models. Attention-based models have been further refined by RNN-based models, which have been since been succeeded by transformer and pretrained models. R-Net (2017) is the state of the art RNN-based

model that uses self-attention over a question-aware context representation to predict the correct answer span in the context paragraph. The paper reports that Microsoft Research Asia's R-Net model achieves an EM score of 72.3 and an F1 score of 80.7 on the Stanford Question Answering Dataset (SQuAD), the premier dataset used for models on the reading comprehension task.

While R-Net performs very well over SQuAD, it does come with a major flaw. In particular, the use of many RNNs with inputs that are dependent on the previous timestep slows down training time considerably. In this paper, I will compare attention components of a Bi-Directional Attention Flow (BiDAF) model (Seo et al., 2016) against R-Net in terms of their effects on training time, Exact Match (EM), and F1 scores.

## 3 Related Work

Neural Network-based question answering systems have pervaded the field since the mid 2010s and have achieved very promising results on the reading comprehension task. Primitive neural-based attention mechanisms were used by models to detect spans within the context paragraph that are most relevant to words within the question string (Weston et al., 2015). Bidirectional Attention Flow Network (BiDAF) improved upon this attention mechanism by introducing both Context-to-Query and Query-to-Context attention (Seo et al., 2016), and a version of the BiDAF model serves as the baseline for our exploration into R-Net.

RNN based question-aware context representations have been explored in match-LSTM (Wang & Jiang, 2016), which performs word-by-word matching of and encoding the question and context. Pointer Networks had previously been explored in Vinyals et al. (2015), and are a system that uses attention over the final hidden state representation of the context paragraph to select the tokens that correspond to the start and end tokens of answer $A$. We do not use pointer networks in our final model, but it is useful to know that pointer networks are the output mechanism used in R-Net.

## 4 Approach

My model takes inspiration from R-Net and BiDAF. It includes a word and character embedding layer followed by a Bidirectional RNN encoding layer. Next, the model uses an attention-based RNN layer to create a question-aware context representation. The model makes use of self-attention in the following RNN layer to match the question-aware context representation against itself to attend to the parts of the context representation that are important to answering the question. Finally, we pass our self-attention representation through an output layer similar to BiDAF to predict the start and end of the answer span using a learned linear transformations.

Code in the implementation of 4.3, and 4.4 is original while code in the implementation of 4.1, 4.2, 4.5 is adapted from the CS224 Winter 2022 IID SQuAD final project baseline.

### 4.1 Character and Word Embeddings

We begin with word embeddings of size *d = 300* to represent our word embeddings, and character embeddings of size *d = 45*. We receive each question and context representation in terms of its word indices and character indices, where each word maps to a particular index and each character maps to a particular index.

We convert each sequence (context and question) into both word embedding representations $\{w_t^C\}_{t=1}^n$ and $\{w_t^Q\}_{t=1}^m$ where $w_t^C, w_t^Q \in \mathbf{R}^{300}$ and character embedding representations $\{c_t^C\}_{t=1}^n$ and $\{c_t^Q\}_{t=1}^m$ where $c_t^C, c_t^Q \in \mathbf{R}^{45\times\text{char limit}}$. Our character embedding representations are further refined through a 2D CNN, and we apply max-pooling to refine each 2D character representation of a word in $\mathbf{R}^{45\times\text{char limit}}$ to a 1D character representation for our word in $\mathbf{R}^{45}$. Lastly, we concatenate our word and character representations, getting $\{[w_t^C, c_t^C]\}_{t=1}^n$ and $\{[w_t^Q, c_t^Q]\}_{t=1}^m$ where $[w_t^C, c_t^C], [w_i^Q, c_i^Q] \in \mathbf{R}^{300+45}$. Unlike BiDAF baseline, we do not project our embedding to our hidden size in this step to maximize information flow our of this layer.

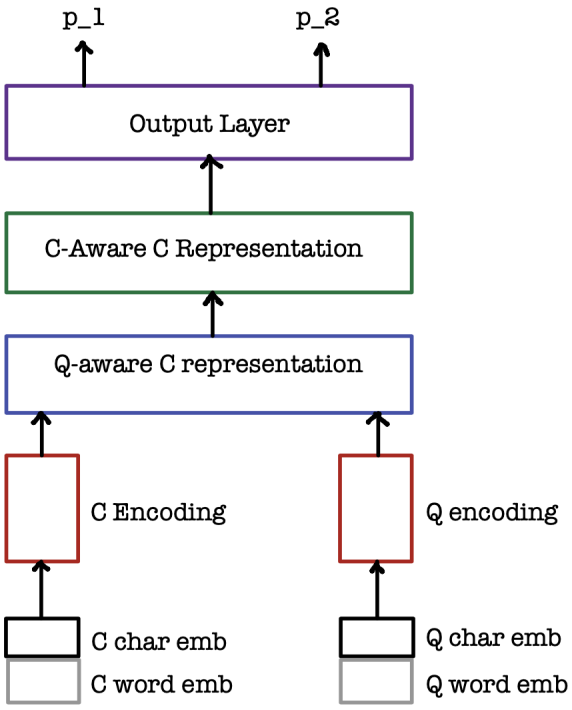


Figure 1: Model structure overview.

### 4.2 Question and Context Encoding Layer

We use a BiDirectional RNN to encode question and context representations $\{u_t^C\}_{t=1}^n$ and $\{u_t^Q\}_{t=1}^m$ using our concatenated word and character representations:

$$u_t^C = \text{BiGRU}(u_{t-1}^C, [w_t^C, c_t^C]) \tag{1}$$

$$u_t^Q = \text{BiGRU}(u_{t-1}^Q, [w_t^Q, c_t^Q]). \tag{2}$$

We choose GRU because it performs similarly to LSTM but is computationally cheaper (Hochreiter & Schmidhuber, 1997). Our BiGRU takes in input vectors in $\mathbf{R}^{345}$ and outputs our sequences of hidden states $u_t^Q, u_t^C \in \mathbf{R}^{2\times\text{hidden size (HS)}}$, which are concatenated representation of forward and backward GRU encodings of our word-character embeddings.

### 4.3 Question-Aware Context Representation Layer

This step uses an attention-based RNN to encode a context representation with information about the question incorporated into it. We take in our question and context encodings $\{u_t^Q\}_{t=1}^m$ and $\{u_t^C\}_{t=1}^m$. We define our question-aware context representation to be

$$v_t^C = \text{GRU}(v_{t-1}^C, [u_t^C, c_t]^*) \in \mathbf{R}^{2H} \tag{3}$$

where $[u_t^C, c_t]^* \in \mathbf{R}^{2H}$ is a gated concatenation of $u_t^C$ and attention pooling vector $c_t$. We define $c_t$ to be

$$s_j^t = \text{v}^T \tanh(W_u^Q u_j^Q + W_u^C u_t^C) \tag{4}$$

$$a_i^t = \exp(s_i^t) / \sum_{j=1}^{m} \exp(s_j^t) \tag{5}$$

$$c_t = \sum_{i=1}^{m} a_i^t u_i^Q \tag{6}$$

where v $\in \mathbf{R}^H$, $W_u^Q, W_u^C \in \mathbf{R}^{2H \times 2H}$ are learnable parameters.

To build our gated representation, we have

$$g_t = \text{sigmoid}(W_g[u_t^C, c_t]) \tag{7}$$

$$[u_t^C, c_t]^* = g_t \odot [u_t^C, c_t] \tag{8}$$

where $W_g \in \mathbf{R}^{2H \times 2H}$ is a learnable parameter, and $[u_t^C, c_t] \in \mathbf{R}^{4H}$ is first projected down to $\mathbf{R}^{2H}$.

### 4.4 Context-Aware Context Representation Layer (Self-Matching Attention)

This step uses a attention-based RNN to encode the question-aware context representation against itself. Without this step, the question-aware context representation would not contain information from the surrounding context. We use a Bidirectional RNN (BiGRU) to encode the context information against itself so that the resulting context representation contains information over the entire passage. To build our self-matching attention representation, we take in our question-aware context representation $\{v_t^C\}_{t=1}^m$. We define our context-aware self-attention representation to be

$$h_t^C = \text{BiGRU}(h_{t-1}^C, [v_t^C, c_t]^*) \in \mathbf{R}^{2H} \tag{9}$$

where $[v_t^C, c_t]^*$ is a gated concatenation of $v_t^C$ and attention pooling vector $c_t$. We define $c_t$ to be

$$s_j^t = \text{v}^T \tanh(W_v^C v_j^C + W_v^{\tilde{C}} v_t^C) \tag{10}$$

$$a_i^t = \exp(s_i^t) / \sum_{j=1}^{m} \exp(s_j^t) \tag{11}$$

$$c_t = \sum_{i=1}^{m} a_i^t v_i^C \tag{12}$$

where v $\in \mathbf{R}^H$, $W_v^C, W_v^{\tilde{C}} \in \mathbf{R}^{2H \times 2H}$ are learnable parameters. We use the same gating mechanism used in (7) and (8) over $[v_t^C, c_t]$ to get $[v_t^C, c_t]^*$.

### 4.5 Output Layer

We use an output layer similar to the output layer from BiDAF, where we are calculating $p_1$ and $p_2$, the log softmax probabilities of the words in the context paragraph being the start and endpoints of the answer, respectively. We receive question-aware context representation $v^C = \{v_t^C\}_{t=1}^m \in \mathbf{R}^{n \times 2H}$ and self-matching attention representation $h^C = \{h_t^C\}_{t=1}^m \in \mathbf{R}^{n \times 2H}$ as input. We have

$$p_1 = \text{LogSoftmax}((v^C W_{v_1}^T + b_{11}) + (h^C W_{h_1}^T + b_{12})) \in \mathbf{R}^{n \times 1}$$

and

$$p_2 = \text{LogSoftmax}((v^C W_{v_2}^T + b_{21}) + (r^C W_{h_2}^T + b_{22})) \in \mathbf{R}^{n \times 1}$$

Where each $W_i \in \mathbf{R}^{2H \times 1}$ and $b_{ij} \in \mathbf{R}^{n \times 1}$ is a learnable parameter, and $r_t^C \in \mathbf{R}^{n \times 2H}$ is defined by

$$r_t^C = \text{BiGRU}(r_{t-1}^C, h_t^C)$$

## 5 Experiments

### 5.1 Data

The Stanford Question Answering Dataset 2.0 (SQuAD 2.0) (Rajpurkar et al., 2016) contains roughly 150,000 (question, context paragraph, answer) triples and is split into **train, dev** and **test** sets. Because the SQuAD 2.0 test set is not publicly available, we will be splitting the official dev set into two, using the first half of the official dev set as our dev set and the second half of the official dev set as our test set. This preprocessing will be done for us by setup.py. Each context paragraph is an excerpt from Wikipedia, and each question and answer pair were crowdsourced using Amazon Mechanical Turk. The training set only has one answer per question while the dev and test sets have three answers per question.

For more on SQuAD, visit https://rajpurkar.github.io/SQuAD-explorer/.

### 5.2 Evaluation method

The main evaluation methods we are using on our models are Exact Match (EM) and F1. EM is a binary measure of whether the system output matches the true answer (the answer provided by the Mechanical Turk crowdworker) exactly. F1 is defined as:

$$F1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

where precision is the percentage of output words that are contained in the correct answer, and recall is the percentage of the words in the correct answer that are contained in the output. The EM and F1 scores are averaged across the entire evaluation dataset to get the final reported scores.

On top of using EM and F1 for evaluation, we are also using Answer vs. No Answer (AvNA) and Negative Log Likelihood (NLL). AvNA scores the model's accuracy on predicting an answer versus a non-answer, while NLL is the loss function metric that use to train our model.

Lastly, we will be evaluating our models in terms of their training time using constant batch sizes and hidden state sizes described further in (5.3). This metric will be in terms of approximate number of iterations per second.

### 5.3 Experimental details

As mentioned earlier, the Gated Recurrent Unit (GRU) is used throughout the model. We use case sensitive pretrained GloVe embeddings for our word embeddings, and randomly initialized character embeddings. Character embeddings for each word are limited to a maximum length of 16.

We use 2 layers of Bidirectional GRU to encode word and character embeddings, and 1 layer GRU for all other RNNs (these are specified in section 4 as either unidirectional or bidirectional). Hidden vector size is set to 64, and batch size is set to 64 as well. We apply dropout between layers (including between layers of LSTM encodings) with a dropout rate of 0.2. We use an AdaDelta optimizer (Zeiler, 2012) with learning rate 0.3, $\rho$ of 0.95, and $\epsilon$ of $1e^{-6}$.

We train for 30 epochs over 129941 SQuAD (question, context paragraph, answer) triples. Training our model averaged approximately 80 iterations per second.

Our baseline contains the original BiDAF model with character embeddings. Unlike our model, it uses question-to-context and context-to-question without RNN encodings. The baseline uses the same hyperparameters used in the final model. Training our baseline averaged approximately 400 iterations per second.

### 5.4 Results

| | **EM** | **F1** |
|---|---|---|
| Baseline Test Score | 55.976 | 59.388 |
| Final Model Test Score | 55.790 | 59.537 |
| Baseline Dev Score | 56.696 | 59.895 |
| Final Model Dev Score | 57.184 | 60.043 |

Figure 1: Test and Dev Scores for Baseline and Final Models.

**Posted on the IID SQuAD leaderboard under name ench**.

The results are not what I expected, and they are worse than I expected. I expected that the extra learned weighted matrix multiplications used in my model would have helped the model create more meaningful representations of the context paragraph—representations that would have resulted in better than +.149 in F1 score and -.186 in the EM score over the test set. Also, I expected that the additional information propagation resulting from the lack of a projection out of the embedding layer would have created more meaningful representations in the original context and question encodings.

These results tell me that my approach either (1) did not benefit from the question-aware context encoding and context-aware context encoding layers over the baseline context-to-question and question-to-context attention layers or (2) resulted in too much information loss when calculating $p_1$ and $p_2$ in the output layer. I hypothesize the latter in the next section.

# 6 Analysis

## 6.1 Training Speed

The final model is 5x slower compared to the baseline model (80 it/s compared to 40 it/s). This drop in speed is due to the presence of an additional RNN in the attention layer compared to the baseline model. Each forward computation through the RNN requires dependence on the previous timestep to compute, removing the possibility for parallelism during these computations.

We even omit the multiplication of $W_v^C v_{t-1}^C$ in the original R-Net paper from our calculation of $s_j^t$ in the Question-Aware Context Representation Layer (4.3) for training-time purposes. When building our GRU input, the dependence of $v_{t-1}^C$ on the previous timestep forces us to run a single forward iteration of a GRUCell in order to get the input to the next timestep. Without depending on $v_{t-1}^C$, we are free to build our entire GRU input at once allowing us to use GRU, which is heavily optimized using cuDNN, over GRUCell.

The main factor that slows down the final model is the presence of many matrix multiplications in my model compared to the baseline. We learn only one weight vector in the baseline attention mechanism, whereas my model learns two weight vectors and six weight matrices in its attention mechanism. These added matrix multiplications during training make training very computationally expensive.

## 6.2 Information Loss

Although the model feeds in non-projected, size 345 embedding representations into the encoding layer while the baseline feeds in embeddings projected down to the hidden size, the model performs poorly in terms of keeping past information when sending the question-aware and context-aware context representations into the output layer.

The BiDAF baseline output layer takes

$$g_t^C = [u_t^C, a_t^C, u_t^C \odot a_t^C, u_t^C \odot b_t^C] \in \mathbf{R}^{8H}$$

as input (along with a temporal encoding of $g_t^C$), where $a_t^C$ and $b_t^C$ are the context-to-question and question-to-context attention representations, respectively. We can see that this concatenation contains $u_t^C$, our unedited context encoding, as well as the weighted attention representations of $u_t^C$, $u_t^C \odot a_t^C$ and $u_t^C \odot b_t^C$. The unedited and weighted representations of $u_t^C$ preserve the information from the encoding and attention layer in the output layer.

The context representations sent into the output layer of our model, $v_t^C$ and $h_t^C$, are heavily modified temporal representations of $u_t^C$, meaning that they lose useful information from the original $u_t^C$. A solution to this problem would be to add a linear transform of $u_t^C$ to our transformed representations of $v_t^C$ and $h_t^C$ when calculating $p_1$ and $p_2$.

## 6.3 Learning Rate

During final model training, I noticed that the F1 score stabilized around approximately 54 with a learning rate of 1, and approximately 57 with a learning rate of 0.5. These scores indicate that larger learning rates result in overshooting the minimum, hindering training when gradients are small in magnitude. For future experiments, testing a learning rate of $< 0.3$ or using adaptive learning rates would be essential for model training.

## 6.4 Output Layer

Standard R-Net uses pointer networks (Vinyals et al., 2015) in the output layer instead of a simple linear transformation and softmax operation. Pointer networks use neural attention over question-aware and context-aware context representations to predict start and end positions of our answer span. Using this approach would have resulted in using more high-dimensional matrix multiplications within the calculation, but it otherwise could have had the potential of choosing a better span due to the refined representations that these extra matrix transformations could have created.

## 7 Conclusion

This goal of this project was to compare and contrast attention-based recurrent networks against question-to-context and context-to-question attention in the reading comprehension task by comparing the mechanisms specified in BiDAF (Seo et al., 2016) and R-Net (Microsoft Research Asia, 2017). The final model describes an implementation that achieves a slight improvement +.149 in F1 and a slight drop of -.186 in the EM score. We see that the implementation of attention-based recurrent networks performs about as good as that of question-to-context and context-to-question attention, and we see that there some adjustements that can be made to the current model (described in section 6) that could make attention-based recurrent networks perform even better.

The training speed of attention-based recurrent networks remains a cause for concern as the final implementation runs 5x slower than BiDAF attention, even without including components like pointer networks that would have resulted in even slower training due to the use of additional RNNs and time-dependent calculations. This bottleneck showed that RNN based reading comprehension systems should be left in the past in favor of faster transformer-based architectures such as QANet that leverage heavily the use of parallelism and do not rely on the temporal encodings of RNNs.

## References


Meraldo Antonio. Word Embedding, Character Embedding and Contextual Embedding in BiDAF — an Illustrated Guide. Towards Data Science (2019). Retrieved from https://towardsdatascience.com.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, 2016.

Natural Language Computing Group, Microsoft Research Asia. R-Net: Machine Reading Comprehension with Self-Matching Networks. (2017) Retrieved from https://www.microsoft.com/en-us/research/

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

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada, pp. 2692–2700, 2015.

Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. arXiv preprint arXiv:1608.07905, 2016.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. In ICLR, 2015.

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