# Building a QA system (IID SQuAD track)

**Siddhartha Jagannath**
Department of Computer Science, Stanford University
kachapah@stanford.edu
https://github.com/Sidu28/BiDAF-Self-Attention.git

## Abstract

Machine comprehension and question answering is one of the most interesting and complex fields in modern NLP. The SQuAD data set, a reading comprehension data set with 100,000+ questions on Wikipedia articles, has risen to prominence as a benchmark for training QA systems. One popular SQuAD-based QA system is BiDAF. In this project, I will present a modified implementation of the provided baseline, that includes character embeddings with an added self-attention layer to enhance the performance over the baseline.

## 1 Introduction

In this paper, I focus on building a QA system that focuses on reading comprehension style of questioning. Given a passage and a question, the goal is to generate an answer from the passage. This paper will focus on the Stanford Question Answering Data set (SQuAD) (Rajpurkar et al., 2016), perhaps the most famous large-scale reading comprehension/question answering data set. SQuAD, unlike cloze-style reading comprehension data sets (Hermann et al. 2015; Hill et al. 2016) which require a single word answer, requires answers to be derived from the space of all possible spans within the reference passage.

In this paper, I make use of the provided baseline for the CS224n SQuAD IID track. This baseline is a slightly modified version of the BiDAF ( bi-directional attention flow networks) proposed by Seo et al. (2016). I added character embeddings and a self attention layer to this baseline. The basic architecture of my modified model can be seen below, in Figure 1.

The model is made up of 6 distinct layers: 1) **Embedding layer**, which takes in character embeddings (which are processed by a basic CNN for standardized dimensions) and word embeddings and concatenates them 2) **Contextual embedding layer** made up of a bidirectional LSTM which utilizes contextual cues from surrounding words to refine the embedding of the words, applied to both query and context embeddings 3) **Attention Layer** which combines the query and context vectors

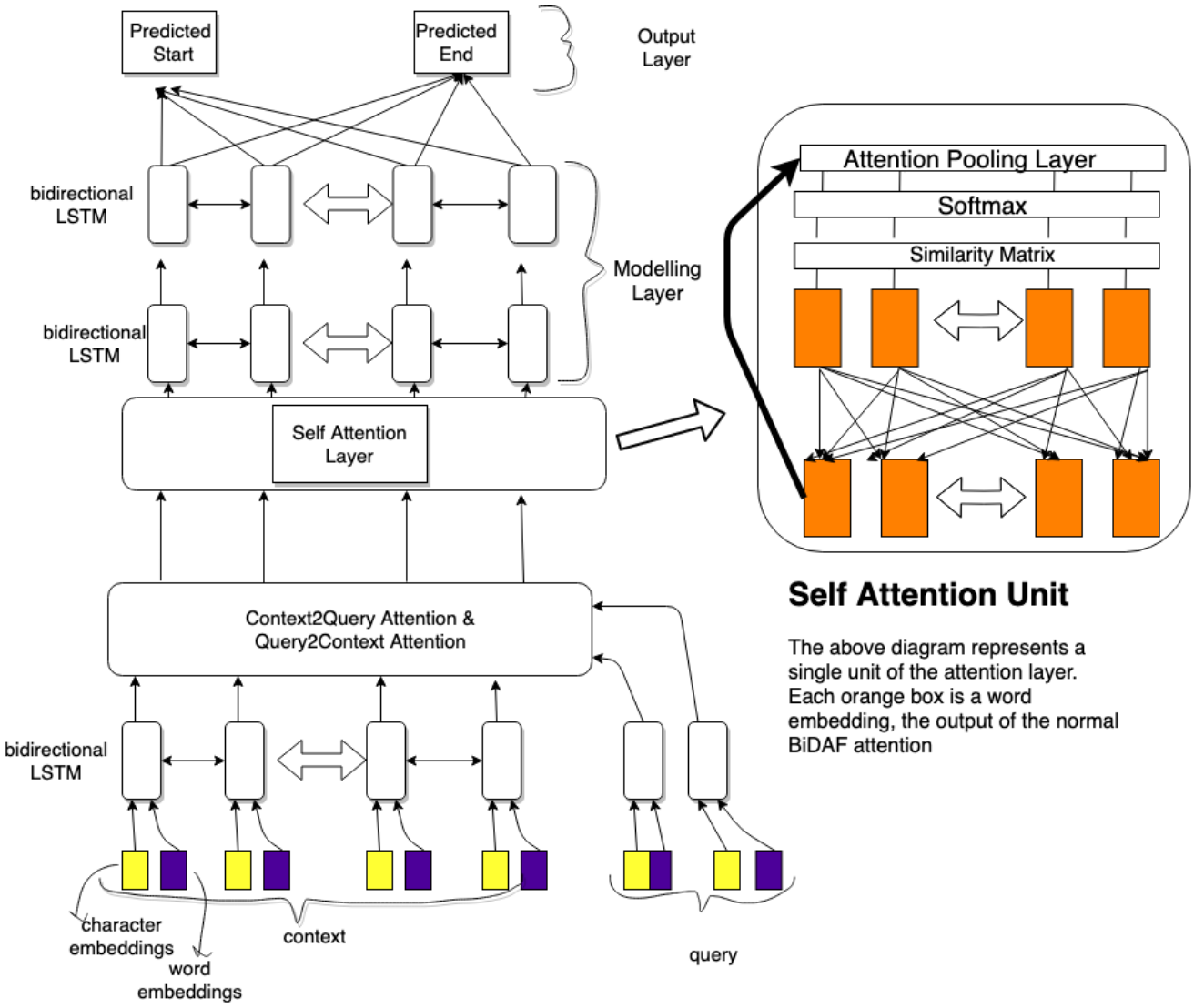



Figure 1: BiDAF w/ a Self Attention Layer

and produces a set of query-aware feature embedding vectors for each word in the context 4) **Self-attention layer** which takes in the output of the normal attention layer, and captures relationships between each word embedding and every other word embedding in a given context (refer to *Self Attention Unit* in Figure 1) 5) **Modeling layer** that uses a bi-directional RNN to scan and model the output from the self-attention layer 6) **Output layer** that provides an answer to the query.

My contributions in this paper are two-fold. First, I implemented a convolutional neural network to process and concatenate character embeddings to the provided word embeddings. Second, I designed a self attention layer that takes in input from the normal context-to-query attention layer, enriches it with additional information from the context and passes it on to the modeling and output layers. The passage representation from the normal attention layer effectively encodes question information for each passage word. However, it only captures limited passage context in practice, which is where the

self-attention layer comes in - with the self attention layer, we can capture broader relationships that exist between all different words in the passage.

Finally, my results are as follows. The baseline achieves the following scores: EM=56.646 and F1 = 60.101 on the validation set. My model as proposed had an improved score of EM = 59.81, F1 = 62.87 on the validation set and EM = 58.952, F1 = 62.459 on the test set.

## 2 Related Work

Since the release of the SQuAD data set in 2016 (Wang  Jiang (2016b)), many developments in have been made in terms of modelling the SQuAD data. Here are several notable models that tackle this task using the famous SQuAD data set:

- Vinyals et al., (2015) predicts answer boundaries in the passage with pointer networks
- Wang  Jiang, (2016a) builds question-aware passage representation with match-LSTM
- Lee et al. (2016) and Yu et al. (2016) predict answers by ranking continuous text spans within passages.
- Xiong et al. (2016) proposed dynamic co-attention networks which attend the question and passage simultaneously and iteratively refine
- Seo et al. (2016) used bi-directional attention flow networks (BiDAF) to model question-passage pairs at multiple levels of granularity.
- R-Net (2017) proposes an end-to-end neural network (R-Net) that uses a self-matching attention mechanism and pointer networks to locate the positions of answers from the context passages.

My paper is heavily inspired by the BiDAF paper and the R-Net paper. The baseline implementation, provided to me by the CS224n teaching team, is inspired by the BiDAF paper with some simplifications (including the exclusion of a character embedding layer). My implementation of a self attention layer modeled after the self-matching layer proposed in the R-Net paper. My model, in many ways, seeks to combines what I see are the strong points of the BiDAF model and the R-Net model, to create a model that represents a novel approach to the problem of building a SQuAD-based QA system.

## 3 Approach

The main approach presented here is simple. I have taken the baseline implementation of the project and added character-level embeddings as well as a self attention. Here are each of the layers of my model in detail:

- **Character embedding layer** maps each word to high dimension vector space. The addition of character-level embeddings is to allow our model to pick up additional nuances in sequences of words. I use a simple CNN with a single convolution layer - the char-embedding

vectors associated with each word are input, the output is max-pooled, fixed-length vector that represents a combination of the char_vectors (using the maxpooling). That output is then passed into the embedding layer after being concatenated to the word embeddings.

- **Word embedding** is part of the baseline and is accomplished by passing pre-trained GLoVE embedding vectors. I then concatenate the word and character embeddings into a single vector and pass them into a Highway Network. This is done for both the context and queries, so we have two outputs, $C \in R^{d \times C}$ (context) and $Q \in R^{d \times Q}$ (query) where T and J are the dimensions of context input and query input respectively.
- **Contextual Embedding Layer** Here a bidirectional LSTM is used, and the outputs of both directions are concatenated. The inputs for context and query are $C \in R^{d \times C}$ and $Q \in R^{d \times Q}$ respectively. The outputs (concatenated for both directions) is $H \in R^{2d \times C}$ (context) and $U \in R^{2d \times Q}$ (query).
- **Attention Layer** This is the step where I combine the context and query information using attention. There are two parts, Context to query attention and query to context attention. To calculate both attentions, I use the similarity matrix $S = \alpha(H_{:t}, U_{:j})$ where $\alpha(h, u) = w_{(S)}^T[h; u; h \odot u]$ is a scalar function that encodes the similarity between the two matrices by multiplying a simple concatenation by a weight vector.
  - **Context to Query** We calculate the attention weight $\hat{U}_{:t} \in R^{2d \times C}$ where:

$$\hat{U}_{:t} = \sum_j a_{tj} U_{:j} \in R^{2d} \tag{1}$$

  - **Query to Context** We calculate the attention weight $\hat{H}_{:t} \in R^{2d \times C}$ where:

$$b = \texttt{softmax}(max_{col}(S)) \in R^C$$

$$\hat{H}_{:t} = \sum_j b_t H_{:j} \in R^{2d} \tag{2}$$

  Thus $\hat{U}_{:t} \in R^{2d \times C}$ and $\hat{H}_{:t} \in R^{2d \times C}$. Finally the output of the entire layer is a combination of the two attentions $G_{:t} = \beta(H_{:t}, \hat{U}_{:t}, \hat{H}_{:t})$, where $\beta(h, \hat{h}, \hat{u}) = [h; \hat{u}; h \odot \hat{u}; h \odot \hat{h}] \in R^{8d \times C}$.
- **Self-attention Layer** serves to better inform the main Attention layer about the context itself. It does this by directly matching or attending the context representation against itself, to produce a passage representation $h$ using a bidirectional RNN, the passage embeddings $p_t$ and the attention mechanism $c_t$. Below is the math behind my self-attention layer. Note that $d$ refers to the hidden size, and $C$ refers to the number of word embeddings in a example context.

$$s_j^t = v^T \tanh(W_t p_j + W_j p_t) \in R \tag{3}$$

$$s = v^T \tanh(pW_t + pW_j) \in R^{C \times d} \tag{4}$$

$$a = \texttt{softmax}(s) \in R^{C \times C} \tag{5}$$

$$c_t = \sum_{i=1}^{n} a_i^t p_i \tag{6}$$

$$c = [c_0, c_1, ....., c_{n\_embed}] \in R^{C \times d} \tag{7}$$

For reference, $p \in R^{S \times d}$, $W_t \in R^{d \times d}$, $W_j \in R^{d \times d}$ and $v \in R^d$. I begin by calculating a similarity matrix $s$, that captures the relationship between all possible pairings of words in the context. In equation (3), we can calculate each element of the similarity matrix, $s_j^t$. Equation (4) represents a vectorized form of this computation, to calculate the similarity matrix $s$. Once we have $s$, in equation (5) we take the softmax across each row of $s$. Finally, in equation (6) and (7) we compute the attention pooling vectors for each word embedding $p_t$. The output of this layer is the tensor $c \in R^{C \times d}$ which represents a stacked matrix of all the attention pooling vectors ($c_t \in R^d$) for every word in the context embedding.

- **Modeling Layer** Just as in BiDAF, I used the baseline's two layers of bidirectional LSTMs, which has an output $M \in R^{2d \times C}$. Each column vector of M is expected to contain contextual information about the word with respect to the entire context.
- **Output Layer** The output of the model is supposed to be the start and end index of the phrase that answers our question in the paragraph. So the output is two probability distributions, one for the start and one for the end index. First we have $p_{start} = softmax(w_{p_{start}}^T [G; M])$ and then we have $p_{end} = softmax(w_{p_{end}}^T [G; M^2])$. The loss, Negative Log Likelihood, is defined as:

$$Loss_{example_i}(\theta) = \frac{-1}{N} \sum_{i}^{N} log(p_{start}) + log(p_{end}) \tag{8}$$

Where $\theta$ is the set of all trainable weights to be updated by backpropagation, *start* refers the true start index, and *end* represents the true end index.

# 4 Experiments

## 4.1 Data

My model focused on modelling a QA system based solely on the Stanford Question and Answering Data set (or SQuAD). SQuAD (Rajpurkar et al., 2016) is a machine comprehension dataset on a large set of Wikipedia articles, with more than 100,000 questions. The answer to each of these question is always contained within the provided context. The input to the model is a context passage and a query about that context. The output of my model is the start and end indices (within the context) of the predicted answer to the question.

## 4.2 Evaluation method

Since I am doing the default project, I am primarily evaluating my model's performance on the baseline F1/EM scores that the provided implementation produces, using the leaderboard. The EM is

the Exact Match score and the F1 score measures the weighted average of the precision and recall rate at the character level of the predictions. I will also be examining the Negative Log Liklehood (NLL) and the AvNA score (Answer vs. No Answer). NLL is the computed loss at the end of training, which we are trying to maximize. AvNA measures the classification accuracy of the model based on it's final predictive ability. It measures the model's answer (any span predicted) against its no-answer predictions. I will be using these four metrics to characterize the efficacy of each of my experimental models.

### 4.3 Experimental details

Over the course of this project, I trained a total of four different iterations of my model, each with some variations. I have listed each one below with their details:

- The first model I trained was the provided baseline. This was very straightforward and took about 10 hours to run.
- The next iteration of the model was to concatenate the character embeddings to the word embeddings. I implemented a simple CNN with a single convolution layer. The purpose of that was to standardize the lengths of the character embeddings, so I could easily concatenate them to the word embeddings and maintain uniform dimensions. Once this was implemented, I retrained the model and saw an improvement in performance, as expected. Training of this model took about 16 hours.
- My next iteration of the model was my initial implementation of the self attention layer - let's call this self attention v1. This was the first version of my attention layer, and it involved an implementation that was slightly different than the implementation provided in the Section 4 of this paper. There were several things that did not work out very well including limited GPU, memory, etc. For a more thorough analysis of why this did not work, please refer to the Analysis section, **Section 5**.
- My final step was to configure a self-attention layer that feeds into the current attention layer. I was able to fix most of the issues with the previous version of my self attention. I calculated my similarity matrix using standard matrix multiplication as opposed to element-wise multiplication. With an upgraded GPU, I could accommodate a hidden size of 75 with the default learning rate. I also passed the self attention to the modeling layer and the normal attention to the output layer yielded the best results. For my analysis of why this did work, please refer to the Analysis section, **Section 5**.

### 4.4 Results

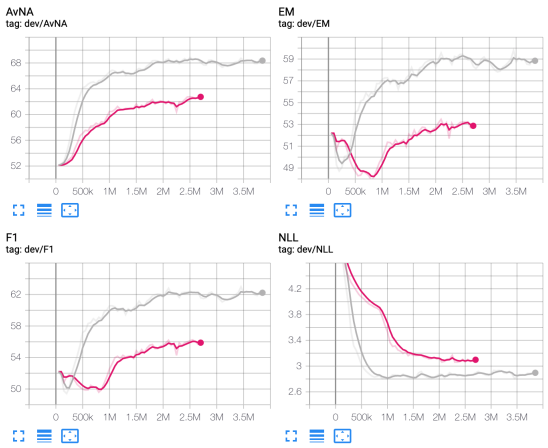
In the table below are all the results of my different experiments. Please refer to the second table for a description of each "Model":

| Validation Set Results | | | | |
|---|---|---|---|---|
| Model Iteration | EM | F1 | NLL | AvNA |
| Model 1 | 56.646 | 60.101 | 3.21 | 64.46 |
| Model 2 | 58.141 (+1.496) | 61.262 (+1.161) | 2.97 | 66.78 |
| Model 3 | 59.81 (+3.164) | 62.97 (+2.869) | 2.81 | 68.73 |

| Test Set Results for Model 3 | | |
|---|---|---|
| Model Iteration | EM | F1 |
| Model 3 | 58.952 | 62.459 |

| Model Descriptions | |
|---|---|
| Model Iteration | Description |
| Model 1 | Baseline implementation of the BiDAF model provided by CS224n Teaching Team |
| Model 2 | Addition of character embeddings to the baseline BiDAF model provided by the CS224n Teaching Team |
| Model 3 | Full model with Character Embeddings and an added Self Attention Layer |

Here are some charts generated by TensorBoard that plot the F1, EM, NLL and AvNA scores of Model 3 (in gray) against Model 1 (in pink) for the validation data set. We can clearly see that the Self-Attention plots from Model 3 outperform the baseline in all metrics.




## 5 Analysis

During my development process, there were many iterations of my model - I modified hyper parameters, etc. However, there were ultimately two main versions of self attention in my model. In this section I will go over why I believe the first version did not work so well and why the second one did work.

### 5.1 Baseline + Character Embeddings

For this iteration of the model, I took the baseline implementation and added character embeddings to the existing word embeddings before initial processing by the LSTM layer. The addition of character-level embeddings is to allow our model to pick up additional nuances in sequences of words, including sub-word relationships. As expected, the addition of these character embeddings raised the F1 and EM scores by 1.161 and 1.496 respectively.

### 5.2 First Iteration of Self Attention

For this version, the first design choice I made was creating my similarity matrix based on element-wise multiplication, to save on compute time. This wasn't the best design choice in retrospect, because doing so severely limited the scope of relationships that could be captured between every different word embedding in the context. Here's an example as to why this is: Element-wise multiplication on two $N \times N$ matrices can only capture $N^2$ relationships, each between only two elements. Matrix multiplication can also capture $N^2$ relationships but each of those relationships is between $2N$ elements. Thus the relationships are much more broad, which is critical for self attention. The goal is to capture as much information about how each element in the input relates to all other elements in the input.

It also happened that I was training on a NC6 GPU, with limited memory. With a hidden size of 100 and a batch size of 64, my weight matrices had roughly 12,288,000,000 elements in them for any given sample. After experimenting, I found that the GPU was able to handle a batch size of 32 and only a hidden size of 30. The reduced hidden size meant the attention layers could pick up on less detail in the encodings themselves.

Finally, I was passing the self attention to both the modeling layer and the output layer. With some experimentation, I found that passing the self attention to the modeling layer and then passing the modeling output and the normal attention to the output layer yielded the best results. While I cannot be certain as to why this is the case, I hypothesize that using the regular attention as a direct input to the output layer (instead of the self attention) helped the output layer because self attention can tend to over fit, as it operates on itself. By including the normal attention in the calculation of the outputs, I'm allowing the model more information about the query as well as the context to make better, more generalized decisions. For all of these reasons, I believe that this version of self attention performed poorly compared to my revision (see below). The training for this version took approximately 18 hours. This model had the following results: **EM**: 55.25, **F1**: 57.39, **NLL**: 3.48. **AvNA**: 63.28.

### 5.3 Self Attention: The Improved Version

This iteration of self attention basically involved fixing all the issues that were present in the above section. I got access to an upgraded NCS6_v2 machine with higher GPU memory. With some additional memory management (including deleting the tensors and clearing the GPU cache after each forward pass), I was able to train my model with a hidden size of 70. I also implemented the

similarity matrix with matrix multiplication, the benefits of which I have described in the previous section. The final change I made was to pass the output of the regular attention to the final output layer and the self attention output only to the modeling layer. All of these resulted in a better F1 and EM score than the baseline, with F1: 59.81, EM: 62.97 on the validation set.

### 5.4 Some Interesting Predictions

In this section, I wanted to take a look at some interesting and telling examples of predictions my model made and include some rationalization about why these might be the case. The first of these examples, Example 1 below was really interesting to me because it seemed as though my model did a better job than the actual answer. The question here is "When did Germany invade Poland and in doing so start World War II?" and the given answer is September 1939. However it seems my self attention seemed to pick up on the relationship between the number "1" and the phrase "September 1939", and predicted the answer "1 September 1939", which can be considered a correct and perhaps more specific answer.

- **Question:** When did Germany invade Poland and in doing so start World War II?
- **Context:** After the German Invasion of Poland on 1 September 1939 began the Second World War, Warsaw was defended till September 27. Central Poland, including Warsaw, came under the rule of the General Government, a German Nazi colonial administration. All higher education institutions were immediately closed and Warsaw's entire Jewish population – several hundred thousand, some 30% of the city – herded into the Warsaw Ghetto. The city would become the centre of urban resistance to Nazi rule in occupied Europe. When the order came to annihilate the ghetto as part of Hitler's "Final Solution" on 19 April 1943, Jewish fighters launched the Warsaw Ghetto Uprising. Despite being heavily outgunned and outnumbered, the Ghetto held out for almost a month. When the fighting ended, almost all survivors were massacred, with only a few managing to escape or hide.
- **Answer:** September 1939
- **Prediction:** 1 September 1939

Example 1

- **Question:** What is the Chinese name for the Yuan dynasty?
- **Context:** The Yuan dynasty (Chinese: 元朝; pinyin: Yuán Cháo), officially the Great Yuan (Chinese: 大元; pinyin: Dà Yuán; Mongolian: Yehe Yuan Ulus[a]), was the empire or ruling dynasty of China established by Kublai Khan, leader of the Mongolian Borjigin clan. Although the Mongols had ruled territories including today's North China for decades, it was not until 1271 that Kublai Khan officially proclaimed the dynasty in the traditional Chinese style. His realm was, by this point, isolated from the other khanates and controlled most of present-day China and its surrounding areas, including modern Mongolia and Korea. It was the first foreign dynasty to rule all of China and lasted until 1368, after which its Genghisid rulers returned to their Mongolian homeland and continued to rule the Northern Yuan dynasty. Some of the Mongolian Emperors of the Yuan mastered the Chinese language, while others only used their native language (i.e. Mongolian) and the 'Phags-pa script.
- **Answer:** Yuán Cháo
- **Prediction:** 元朝

Example 2

- **Question:** What was AUSTPAC
- **Context:** AUSTPAC was an Australian public X.25 network operated by Telstra. Started by Telecom Australia in the early 1980s, AUSTPAC was Australia's first public packet-switched data network, supporting applications such as on-line betting, financial applications — the Australian Tax Office made use of AUSTPAC — and remote terminal access to academic institutions, who maintained their connections to AUSTPAC up until the mid-late 1990s in some cases. Access can be via a dial-up terminal to a PAD, or, by linking a permanent X.25 node to the network. [citation needed]
- **Answer:** AUSTPAC was an Australian public X.25 network operated by Telstra
- **Prediction:** Australia's first public packet-switched data network

Example 3

The next example, example 2 is an instance where my model predicted the one other correct answer. The answer to the question, "What is the Chinese name for the Yuan dynasty", is "Yuan Chao" (in pinyin)but there is an equivalent answer provided in chinese script. My model picked up on the Chinese example whereas the labeled answer was the pinyin script version. Finally in example 3, the question is "What was AUSTPAC?" In the passage, the phrase "AUSTPAC was" shows up two times. This is probably the phrase the model is looking for to find the definition of AUSTPAC. While the correct answer is "AUSTPAC was an Australian public X.25 network operated by Telstra", my model chose the other instance of "AUSTPAC was", which was "AUSTPAC was Australia's first public packet-switched data network." In this case, I hypothesize that the self attention layer found that the relationship was stronger between the defining phrase "AUSTPAC was" and "Australia's first public packet-switched data network" than the phrase "an Australian public X.25 network operated by Telstra." Ultimately it chose the less specific phrase to define AUSTPAC - this indicates that in the future, some research ought to be done into how I can better distinguish between which phrases are "better" answers to a question. These are some interesting situations in which my model went wrong and give us insight into how the model behaves in the wild and how we might improve its performance in future work

## 6 Conclusion

In this paper, I introduce an implementation of the BiDAF with character embeddings and a self attention layer, additions which help capture nuances within words as well as the relationships between different words, to create a QA system trained on SQuAD. The experimental evaluations show that my model slightly outperforms the baseline implementation's performance. The visualizations and subsequent analysis and discussion show that my modified version of the baseline is suited to tackle the reading comprehension-style Question Answering task. Future work will involve extending the self attention layer, ablation studies and hyper-parameter tuning.

## References


[1] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Association for Computational Linguistics (ACL)*, 2018.

[2] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension, 2018.

[3] Yichen Gong and Samuel R. Bowman. Ruminating reader: Reasoning with gated multi-hop attention, 2017.

[4] Kenton Lee, Shimi Salant, Tom Kwiatkowski, Ankur Parikh, Dipanjan Das, and Jonathan Berant. Learning recurrent span representations for extractive question answering, 2017.

[5] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[6] Shuohang Wang and Jing Jiang. Learning natural language inference with LSTM. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1442–1451, San Diego, California, June 2016. Association for Computational Linguistics.

[7] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering, 2018.

[8] CS224n Teaching Team. Cs 224n default final project: Building a qa system (iid squad track), 2021.

[9] Microsoft Research Asia Natural Language Computing Group. R-net: Machine reading comprehension with self-matching networks, 2017.

[1] [2] [3] [4] [5] [6] [7] [8] [9]