Exploring Techniques for Neural Question Answering

Gabriel Bianconi (bianconi@stanford.edu) & Mahesh Agrawal (mahesha@stanford.edu)

Winter 2018

1 Abstract

We present a number of techniques for neural question answering and evaluate our results on the Stanford Question Answering Dataset (SQuAD). We explore, implement, and evaluate the effectiveness of some of the state-of-the-art models and techniques. In particular, we discuss how Bidirectional Attention Flow (BiDAF) can improve a naive attention layer. We improve the modeling capacity of our system by introducing better recurrent architectures, and a smarter span selection algorithm that improves test time performance. Finally, we expand the input features by leveraging character-level embeddings and a few hand-crafted futures. The model achieves competitive F1 and EM scores on the task.

2 Introduction & Literature Review

Large-scale neural question answering systems have gained more attention since the publication of the Stanford Question Answering Dataset (SQuAD) [1] in 2016. SQuAD is a reading comprehension dataset based on Wikipedia articles built by crowdsourced workers. It is significantly larger than previous datasets, with over 100K question-answer pairs from over 500 Wikipedia articles. Due to the large amount of labeled data, this dataset is particularly interesting for supervised learning problems.

More formally, the reading comprehension task on the SQuAD dataset is as follows. The model is given an input context (a passage from Wikipedia) and a question and is supposed to output the span of the context representing the answer. Note that answers are always explicitly contained in the context, but might have different length.

Since the dataset was published, various novel models and techniques have been proposed for solving this task. Recent publications have achieved incredibly high performance. The highest-scoring model is the Hybrid AoA Reader [2], an ensemble which has achieved an F1 score of 89.2%. Even single-model approaches such as QANet+ (unpublished) have achieved high performances, with an F1 score of 88.6%. By comparison, human performance is slightly higher, with an F1 score of 91.2%.

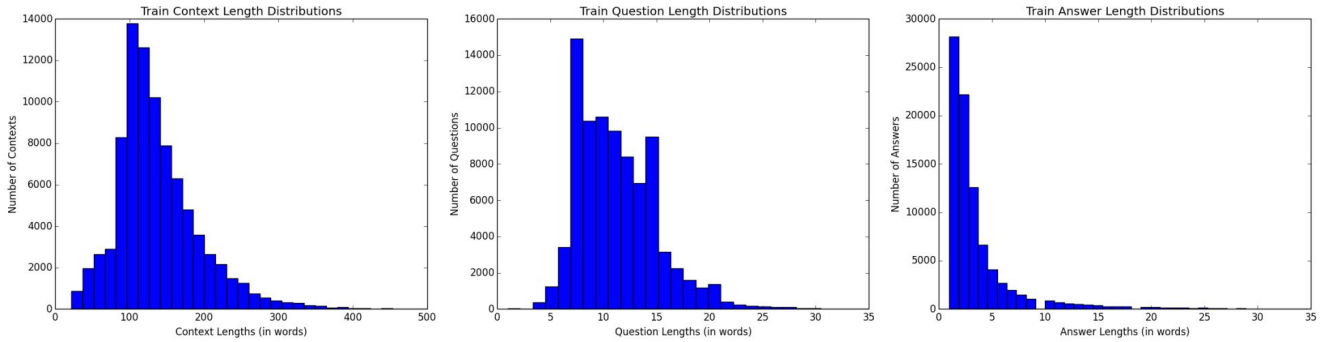
Multiple previous projects focused on different attention mechanisms to improve question answering performance. In [3], the authors introduce the Bidirectional Attention Flow (BiDAF) network, which features two-way attention between questions and contexts to achieve state-of-the-art results in this task. [4] leveraged co-attention, a hierarchical and iterative approach to attention, to achieve similarly strong results. [5] combines a simple context-to-attention layer with a higher-level self-attention layer. Previous research has also explored smarter ways to jointly model start and end positions to achieve better span predictions. In [6], among other contributions, the authors condition the end distribution on the start distribution to achieve higher accuracy. [3] provides a similar approach, which combines two hierarchical RNNs to model the start and end distributions. Finally, multiple projects introduced richer features that augmented token embeddings. [3] leveraged a character-level CNN to capture knowledge about unknown tokens and model intra-word representations. [7] introduces a number of hand-engineered features, such as part-of-speech and named-entity tags, and achieves competitive performance with an otherwise simple model.

In this paper, we explore a number of state-of-the-art architectures and techniques that are well-suited to deal with the different challenges of this task. Our final approach combines techniques from multiple models and achieves competitive performance. We thoroughly discuss the architecture in the Model section. Finally, we analyze the limitations of our model and present ideas for further improvements that might help achieve even stronger performance.

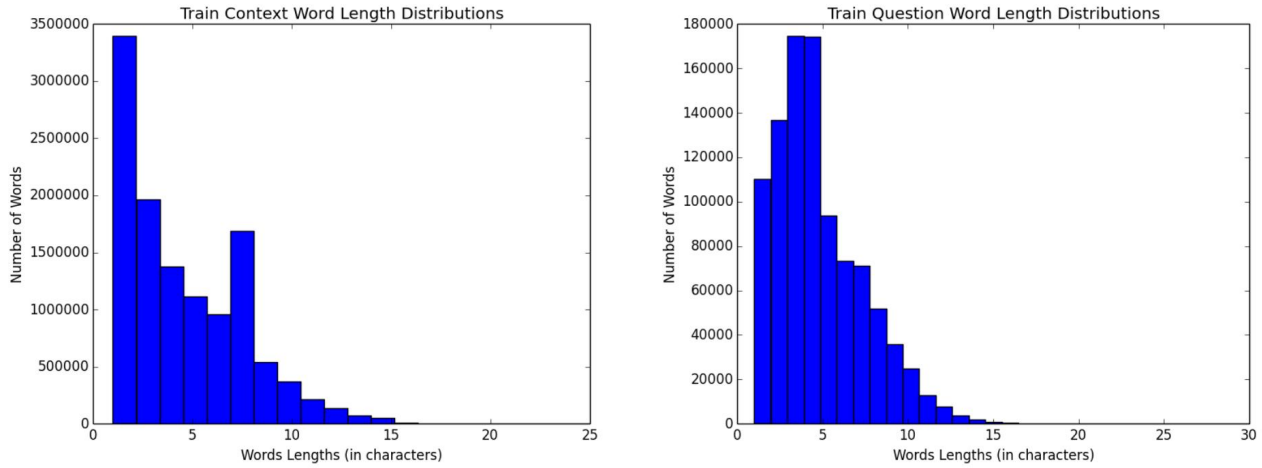
3 Dataset

The provided dataset includes 86K+ training and 10K+ validation contexts from Wikipedia with accompanying questions and answer spans. The answers are always taken directly from the contexts. Below we present several

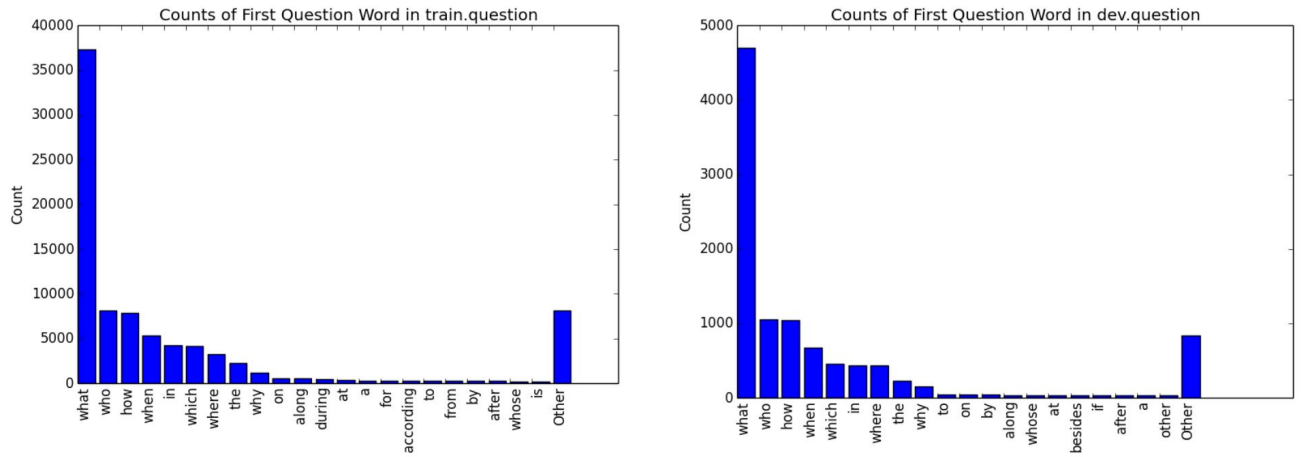
plots to help better visualize the dataset and its features. We used this understanding of our dataset to help us create hand engineered features, fine tune hyperparamaters and improve the model and architecture.



The figure above shows that most contexts were smaller than 400 words in length. Similarly, most questions were under 25 words in length and most answers under 15 words. We used these as the maximum values in our model (reducing the higher default values used in the baseline) which helped reduce over fitting and improved training time.



The figure above shows the distribution of the word lengths (in characters) across the contexts and questions. We can see from the figure that most words were under 15 characters in length. Understanding this about our dataset helped us choose the appropriate size for the maximum word length allowed, which was a key hyperparameter for our character level model.



The figure above visualizes the distribution of the start words for questions in the training and validation set. We can see that ‘what’, ‘who’, ‘when’ and ‘how’ were the highest occurring question start words in that order in both

sets. This provides insight into the nature of the answers our model should be looking for when working with this dataset.

4 Model

4.1 Word Embeddings

We extract token features from the contexts and questions using pre-trained GloVe [8, 9]. GloVe has been found to be the state-of-the-art word representations that combines both semantic and syntactic meaning and thus we leverage these word embeddings for our task. The GloVe authors provide pre-trained word embeddings trained on different corpora sizes, ranging from 6 billion to 840 billion tokens, and with different dimensions.

The baseline model uses 100-dimensional GloVe embeddings trained on 6 billion tokens, which includes 400K word embeddings. However, using this vocabulary size, there is a massive number of unknown tokens (UNKs) encountered during training and testing which hurts the model's performance in the relevant questions. As a result, we replace these embeddings with the GloVe embeddings trained on 840 billion tokens, which includes 2.2 million word embeddings. This significantly reduced the number of UNKs we encountered in the training and validation set as can been seen in the table below.

	Unique UNKs (Training)	**Total UNKs (Training)**	**Unique UNKs (Validation)**	**Total UNKs (Validation)**
Baseline Model (GloVe 6B)	28,929	158,341	3,103	19,408
Final Model (GloVe 840B)	21,601	119,545	2,325	14,913
Difference	−25%	−24%	−25%	−26%

The table above shows the count of unique and total UNKs encountered in both the contexts and questions combined in the training and validation set for the baseline and our final model. We see that using the 2.2 million GloVe-840B embeddings helped reduce the number of UNKs encountered, both unique and total, by about 25% for both the training and validation set.

In addition, we also used the 300-dimensional word embeddings for GloVe (as compared to the 100-dimensional in baseline) since we found that this gave us further representational power which helped the model better capture relationships between words.

4.2 Character-level CNN

In addition to using dense word embeddings to represent words, we also augment each word embedding with its corresponding character embedding using a convolutional neural network (CNN). While CNNs are traditionally used for computer vision, they have been shown to be effective in various language modeling tasks [10, 11, 12, 13, 14]. Most recently, [3] uses a character-level CNN to augment the word embeddings and finds an F1 performance boost of up to 3%. Character level CNNs work by performing a one-dimensional convolution over the character embeddings for a given word and then applying either a max or average pooling after the convolution to obtain one final vector representation for the word. They have shown to help better deal with UNKs and also better capture the internal structure of the words.

More formally, suppose a word w consists of characters $c_1, c_2, \ldots, c_l$, where l is a hyperparameter for the maximum word length. We then convert each character c_i into a character embedding e_i using a character embedding lookup table that can either be pre-trained or randomly initialized (we experimented with both and discuss our findings below). We then transform the character level embeddings into a sequence of hidden representation $h_1, h_2 .... h_l \in \mathbb{R}^f$ using a one-dimensional CNN that has f filters. Then, we apply element-wise max-pooling to obtain the character level encoding: $\text{emb}_{char}(w) = \max_i h_i \in \mathbb{R}^f$. Finally, we concatenate this to the word embedding to get the final word representation.

While we tried using pre-trained character embeddings from [15], they did not perform as well as training our own character level embedding. We hypothesize this is because training our own embeddings allows the model to learn character level representations relevant to the task at hand as opposed to general purpose pre-trained character embeddings.

4.3 Hand-Engineered Features

While the model is able to achieve high performance without any explicit features other than the embeddings discussed about, there have been strong results based on hand-engineered features. [7] employed various such

features and found a significant performance improvement. In particular, we augment the representation of each context token with three additional binary features, namely, whether the context token appears somewhere in the question, whether the lowercase context token appears somewhere in the question and whether the stemmed context token matches any stemmed token from the question. We then concatenate these binary features to the word representation.

In order to check whether the stemmed token appears anywhere in the question, we stem both the current context token p_i and all the question tokens q. We use the Porter stemming algorithm [16], which is largely regarded as the most accurate stemming algorithm with significant gains for NLP tasks [17].

4.4 Attention Layer

For our attention layer, we implement bidirectional attention flow (BiDAF) [3]. First, for every context token c_i and question token q_j, we compute a similarity matrix

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

Second, we perform context-to-question attention, using

$$\alpha^i \text{softmax}(S_{i*}), \quad \forall i \in \{1, \ldots, N\},$$

$$a_i = \sum_j a_j^i q_j, \quad \forall i \in \{1, \ldots, N\}.$$

Finally, question-to-context attention, using

$$m_i = \max_j S_{ij}, \quad \forall i \in \{1, \ldots, N\},$$

$$\beta = \text{softmax}(m), \quad c' = \sum_{i=1}^{N} \beta_i c_i.$$

The output G of this layer, combined with the context features, is given by

$$G = [c_i; a_i; c_i \circ a_i; c_i \circ c'].$$

In addition, we experimented with using a self-attention layer [18] both independently and in conjunction with a BiDAF layer. However, the performance was deteriorated, so we did not include it in the final model.

4.5 Modeling & Output Layer

The baseline model simply added a fully connected layer after the attention layer's output, and fed the outputs to two independent softmax layers. This approach is highly problematic since the end position distribution p_{end} is calculated independently of the p_{start}. In addition, the fully-connected layer does not appropriately model the sequential relationship of this layer's input. This is particularly problematic since most contexts have several hundred tokens.

Instead, we implement the approach used in [3]. The layer is composed of the two hierarchical bidirectional RNNs (BiRNNs)

$$M^{(1)} = \text{BiRNN}(G), \qquad M^{(2)} = \text{BiRNN}(M).$$

Finally, we model the position distributions using

$$p_{\text{start}} = \text{softmax}(\text{FC}([G; M^{(1)}])), \qquad p_{\text{end}} = \text{softmax}(\text{FC}([G; M^{(2)}])).$$

with two independent fully connected (FC) layers.

We experimented with inserting both an additional basic attention and BiDAF layer between $M^{(1)}$ and $M^{(2)}$, but these approaches did not improve the results and greatly reduced performance. They were not included in the final model.

4.6 Span Selection Algorithm

The baseline model selected the start and end positions for the answer span independently of each other by maximizing the individual position distributions p_{start} and p_{end}. However, this naive approach is problematic. The algorithm does not ensure that the end position does not occur before the start position. In cases like that, the prediction is clearly garbage, and the solution is wrong. Instead, the span selection algorithm should ensure the span is valid and reasonable and model the join probability distribution of p_{start} and p_{end}.

We based our approach on [7]. The span selection algorithm ensures that the spans are valid and have a reasonable length. The authors approximate the joint distribution of the span with

$$p_{\text{span}}(i, j) = p_{\text{start}}(i)\ p_{\text{end}}(j)$$

We propose an original extension to this approach that penalizes longer spans. As seen in section 3, the answer length distribution is highly skewed to shorter answers. As a result, we introduce a penalty term that ensures that longer answers are only picked when the model has high conviction. This approach can be implemented efficiently with dynamic programming using the relationship:

$$\underset{(i,j)\ :\ i \le j \le i+\Delta}{\arg\max} p_{\text{start}}(i)\ p_{\text{end}}(j)\ \alpha^{j-i}$$

where Δ is the maximum answer length and α is the length penalty parameter. This is the span that is finally outputted by the model.

4.7 Miscellaneous

4.7.1 RNN Cell

The baseline model used GRU cells [19] for its RNNs. We also experimented with LSTM cells [20] for our models, but did not find any significant change in performance. Our final model included GRUs.

4.7.2 Activation Function

The baseline model used ReLU activations [21] in multiple parts of the model. We experimented with a novel activation function, Swish,

$$a(x) = x \cdot \text{sigmoid}(\beta x),$$

which has been shown to improve overall performance in many existing models due to better gradient flow and avoiding “dead units" [22]. However, we did not see any significant changes in performance, so we did not include it for our final model. Unlike the models discussed in its original paper, our model does not heavily depend on ReLU activations, so we hypothesize this greatly reduced the impact of this change.

5 Experimental Results

5.1 Implementation Details

In this section, we report the implementation details for our final model. These were selected after multiple architecture experiments and extensive hyperparameter search.

Each component of the model had a single layer of units. The RNN cells were GRU cells [19]. The contexts and questions were truncated to 400 and 25 tokens respectively based on our analysis of the dataset (see Dataset section).

Similarly, based on our analysis of the training and validation datasets, we found that over 90% of the words were under 15 characters in length and thus we fixed this as our max word when generating character embeddings for words. Shorter words were padded with a special `PAD` character while longer words were truncated. In addition, we choose a character embedding size, $d_c = 20$, and for the 1 dimensional CNN we chose a kernel size, $k = 5$, number of filters, $f = 100$, and stride of 1.

We selected a hidden size of 128, learning rate of 0.0005, dropout rate of 0.25, and weight decay of 0.0001. The span selection algorithm had a maximum answer length of $\Delta = 15$ and penalty $\alpha = 1$ (no penalty; the penalty was only helpful in weaker models). The loss was minimized with an Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$. We manually reduced the learning rate by half after 8K and 13K iterations once the loss had plateaued. The model trained for 20K iterations over 60 hours. All weights were initialized using Xavier initialization [23].

We use EM (Exact Match) and F1 scores as our performance metric. EM is a binary measure of whether the model output matches the ground truth exactly while F1 is the harmonic mean of the precision and recall. In addition, since SQuAD has three human answers for each question, we use the maximum F1 and EM scores across all three answers.

5.2 Performance

The baseline achieved an F1 score of 43.7% and an EM score of 35.6% (validation). Our final model considerably improved on this result with an F1 score of 75.8% and an EM score of 65.6% (validation). Our model is competitive with cutting-edge single-model results, falling short by only a few percentage points. In particular, the most comparable model, presented in [3], achieved an F1 score of 77.3% and an EM score of 68.0% (test). Refer to the table below for a comparison to other models.

Selected Experiments	**F1**	**EM**
Baseline (ours)	43.7%	35.6%
Final Model (ours)	75.8%	65.6%
Human Performance [1]	91.2%	82.3%
QANet+ (unpublished)	88.6%	82.2%
BiDAF [3]	77.3%	68.0%
Dynamic Chunk Reader [24]	71.0%	62.5%

5.3 General Mistakes

To better understand the limitations of our final model, we manually examined 100 predictions that did not exactly match the solution. The majority of mistakes fell into seven groups: wrong named entity, very long solutions, number formatting errors, date formatting errors, prediction too succinct, prediction not succinct enough, and lack of general knowledge. Refer to table below for details and examples.

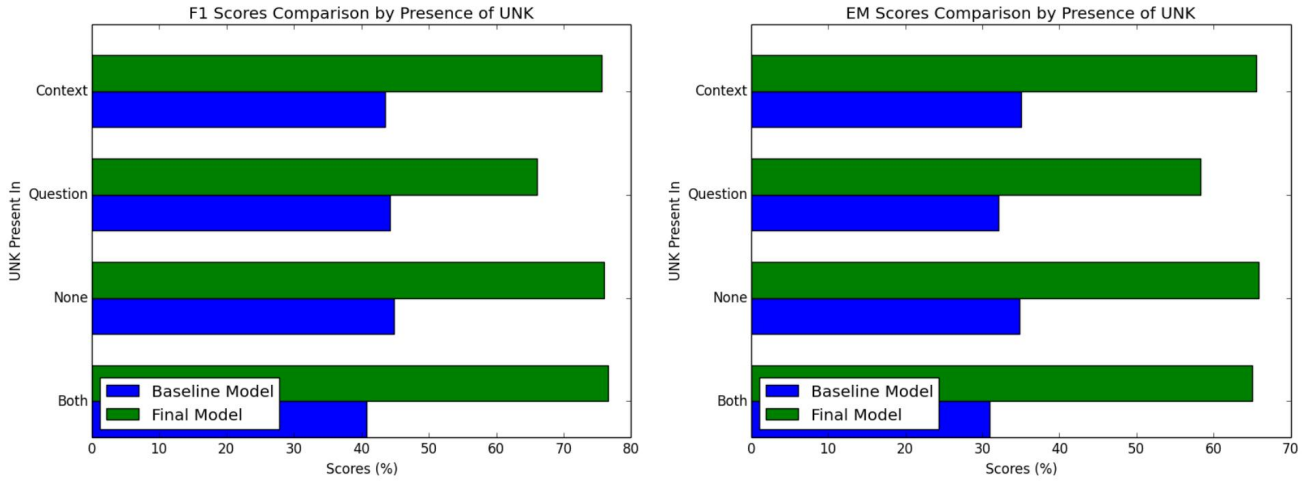
Mistake Class	**Example**
Wrong Named Entity	Question: `Who else did Tesla make the acquaintance of in 1886?` Prediction: `alfred s. brown` Solution: `Charles F. Peck`
Very Long Solutions	Question: `How much did Westinghouse pay to license Tesla's designs?` Prediction: `$60,000` Solution: `$60,000 in cash and stock and a royalty of $2.50 per AC horsepower produced by each motor`
Number Formatting Errors	Question: `When is the first reference in history to Warsaw?` Prediction: `year 1313` Solution: `1313`
Date Formatting Errors	Question: `What is the estimated death toll for Polish civilians?` Prediction: `150,000 and 200,000` Solution: `between 150,000 and 200,000`
Not Succinct Enough	Question: `Which plateau is the left part of Warsaw on?` Prediction: `moraine plateau` Solution: `moraine`
Lack of General Knowledge	Question: `Which Florida venue was one of three considered for Super Bowl 50?` Prediction: `san francisco bay area's levi's stadium` Solution: `Miami's Sun Life Stadium`

In future iterations, a few improvements could help address these issues. A Named Entity Recognition (NER) system could be added to help classify the correct entity. A Part-of-Speech (PoS) tagging system could help with problems related to succinctness, date, and numbers. Finally, handcrafted features to represent numeric values and dates could also help solve some of these issues. Solving the lack of general knowledge mistake is a more involved task and we address this in the future work.

5.4 Unknown Tokens

As discussed in section 3, there was a very large number of unknown tokens (UNKs) in the dataset. We were able to reduce this number by using GloVe-840B, but the number is still substantial. In this section, we investigate the impact of these tokens.

In the figure below, we see that these tokens are especially problematic when they occur in the question. We also see that our final model deals with UNKs much better than the baseline in all situations shown below.



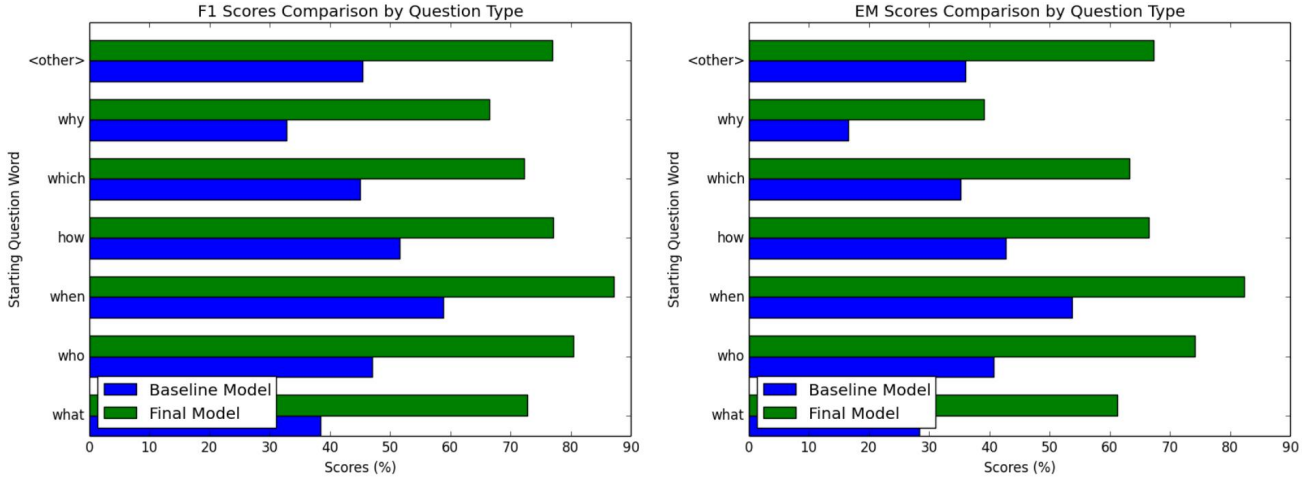
To better understand the problem, we sampled 50 of such questions and manually classified the issues arising from UNKs. The findings are in the table below.

UNK Class	Example
Unknown Named Entity	What position does Demaryius Thomas play?
Unusual Spelling	Which Super Bowl halftime show did Beyoncé headline?
Spelling Mistake	How many intercpetions did Newton have in Super Bowl 50?
Compounded Words	How long was Warsaw the capital of the Polish-Lithuanian Commonwealth?
Numeric Token	For what invention was U.S. Patent 1,655,114 granted?
Unknown Concept	Who fumbled the ball on 3rd-and-9?

Most of these UNKs represent proper nouns that would not be handled by general character or word embedding model and thus dealing with these UNKs remains an open research question.

5.5 Question Types

We also present the performance of our final model in the different question types we observed.



We see that the performance is comparable in most question types, indicating these are not a limitation for our model. The main exception is 'why' questions, which tends to require more reasoning, and therefore is harder to answer accurately. On the converse, the model performs the best on 'when' questions, which makes sense give the objective and factual nature of the answers to such questions.

5.6 Ablations

In this section, we present the performance of our model as we implemented more features. However, unlike the scores reported in the other sections, the scores we report for the ablations only use one human provided answer out of the three possible solutions, so they are lower in aggregate.

Selected Experiments	F1	EM
Baseline	40.0%	29.3%
+ BiDAF Attention Layer	47.8%	35.3%
+ Smart Span Selection	53.0%	39.4%
+ Output Modelling	67.3%	52.2%
+ Char CNN & Hand Engineered Features	68.9%	53.6%
+ Hyper-parameter Search & Regularization	69.2%	54.2%
+ GloVe-840B 300D	70.1%	55.3%

We notice that two major improvements to the model were the addition of the BiDAF attention layer, and the output modelling layer.

6 Conclusion & Future Work

We presented a system based on a number of papers that achieves competitive neural question answering performance using the SQuAD dataset. We identify a number of limitations in the baseline model and challenges of the dataset, and present techniques that address them. Our final model achieves high performance in the validation set (F1: 75.8%, EM: 65.6%).

In particular, we have been able to greatly improve on the baseline performance by upgrading the attention, modeling, and output layers. We observed that BiDAF outperformed Self-Attention and a combination of both techniques. Leveraging RNNs for modeling greatly increased the performance of the model since we are dealing with sequential data and were able to jointly model the start and end positions. Using a more powerful and larger embedding matrix along with character-level embeddings enabled us to alleviate the issue of unknown tokens (UNKs) by learning inter and intra-word representations in the contexts and questions. Finally, a better span selection algorithm gave a final boost to our F1 and EM scores.

While our final model achieved impressive results, we believe we can further build on these techniques to improve our results. Due to time and computational constrains, we were unable to perform a more extensive hyperparameter search or anneal at a slower pace for a longer period. In addition, while we managed to reduce overfitting significantly with L2 regularization and dropout, we still believe this is an area that deserves further attention. From our analysis, we also believe that the system would benefit from an augmented dataset with a few more features such as such as part-of-speech (POS) and named-entity recognition (NER) tags as in [7] along with features such as aligned question embedding [25]. Handcrafted features to manage numeric values and dates might also be helpful. There is promise in exploring multiple other model architectures; we're particularly intrigued about the possibility about using convolutional architectures, or even using reinforcement learning to explore novel architectures as in [26]. Finally, we focused on single-model systems, but ensembles traditionally perform exceptionally well on such tasks, so it is an interesting avenue for further performance boost.

References

[1] Pranav Rajpurkar et al. "Squad: 100,000+ questions for machine comprehension of text". In: *arXiv preprint arXiv:1606.05250* (2016).

[2] Yiming Cui et al. "Attention-over-attention neural networks for reading comprehension". In: *arXiv preprint arXiv:1607.04423* (2016).

[3] Minjoon Seo et al. "Bidirectional attention flow for machine comprehension". In: *arXiv preprint arXiv:1611.01603* (2016).

[4] Caiming Xiong, Victor Zhong, and Richard Socher. "Dynamic coattention networks for question answering". In: *arXiv preprint arXiv:1611.01604* (2016).

[5] Wenhui Wang et al. "Gated self-matching networks for reading comprehension and question answering". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vol. 1. 2017, pp. 189–198.

[6] Shuohang Wang and Jing Jiang. "Machine comprehension using match-lstm and answer pointer". In: *arXiv preprint arXiv:1608.07905* (2016).

[7] Danqi Chen et al. "Reading wikipedia to answer open-domain questions". In: *arXiv preprint arXiv:1704.00051* (2017).

[8] Robert Parker et al. "English gigaword fifth edition, linguistic data consortium". In: *Google Scholar* (2011).

[9] Jeffrey Pennington, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[10] Wen-tau Yih, Xiaodong He, and Christopher Meek. "Semantic parsing for single-relation question answering". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Vol. 2. 2014, pp. 643–648.

[11] Yelong Shen et al. "Learning semantic representations using convolutional neural networks for web search". In: *Proceedings of the 23rd International Conference on World Wide Web*. ACM. 2014, pp. 373–374.

[12] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. "A convolutional neural network for modelling sentences". In: *arXiv preprint arXiv:1404.2188* (2014).

[13] Ronan Collobert et al. "Natural language processing (almost) from scratch". In: *Journal of Machine Learning Research* 12.Aug (2011), pp. 2493–2537.

[14] Yoon Kim. "Convolutional neural networks for sentence classification". In: *arXiv preprint arXiv:1408.5882* (2014).

[15] *minimaxir/char-embeddings*. Apr. 2017. URL: `https://github.com/minimaxir/char-embeddings`.

[16] Martin F Porter. *Snowball: A language for stemming algorithms*. 2001.

[17] Peter Willett. "The Porter stemming algorithm: then and now". In: *Program* 40.3 (2006), pp. 219–223.

[18] Bhuwan Dhingra et al. "Gated-attention readers for text comprehension". In: *arXiv preprint arXiv:1606.01549* (2016).

[19] Junyoung Chung et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (2014).

[20] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[21] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[22] Prajit Ramachandran, Barret Zoph, and Quoc V Le. "Swish: a Self-Gated Activation Function". In: *arXiv preprint arXiv:1710.05941* (2017).

[23] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.

[24] Yang Yu et al. "End-to-end answer chunk extraction and ranking for reading comprehension". In: *arXiv preprint arXiv:1610.09996* (2016).

[25] Kenton Lee et al. "Learning recurrent span representations for extractive question answering". In: *arXiv preprint arXiv:1611.01436* (2016).

[26] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. "An empirical exploration of recurrent network architectures". In: *International Conference on Machine Learning*. 2015, pp. 2342–2350.