CS 224N Final Project Report

Nick Steele
Department of Computer Science
Stanford University
Stanford, CA 94305
nsteele@stanford.edu

Gabriel Voorhis-Allen
Department of Symbolic Systems
Stanford University
Stanford, CA 94305
gvoorhis@stanford.edu

Abstract

Our project examines end-to-end deep learning techniques for question answering on the Stanford Question Answering Dataset (3). To address this problem, we have implemented two primary additional features on a baseline Bidirectional Attention Flow model, namely character-level embeddings and self-attention. At the time of submission, our test leaderboard F1 score was 61.221.

1 Introduction

SQuAD 2.0 is a reading comprehension dataset consisting of paragraphs and questions about those paragraphs. Given a context (several sentences or paragraphs) and a related question, the goal is for the model to answer a question related to the paragraph. This challenge is significant because improved question answering capabilities have the potential to positively impact many fields including academic research, medical services, and education. SQuAD 2.0 is particularly well-suited for evaluating question answering models because of its size (about 150,000 examples), its quality, the fact that each answer is an exact substring of the context, and the fact that it contains about 50,000 unanswerable questions.

Our model combines several features that make it well-suited to question answering. It includes three primary parts:

1. A Bidirectional Attention Flow model
2. Character-level embeddings
3. A Self-Attention layer

These features, in addition to hyperparameter tuning, obtain a F1 score of 61.221 on the test leaderboard.

2 Related Work

There are a wide variety of recent papers on deep learning approaches to question answering on SQuAD. Our baseline model is based off of BiDAF by Seo et al. 2016 with no character embeddings (4). BiDAFcombines context-to question and question-to-context attention, leading to a large performance gain over previous models. Our self-attention layer is inspired by Wei et al. 2017 in RNET (9), which aggregates evidence to answer a question from the entire context to form an answer, and also created large performance gains over other models. The current top of the leaderboard (2) models use pre-trained contextual embeddings such as ELMo and Bert (13) (14). Pre-trained contextual embeddings use use word embeddings that are dependent on the context in which words appear in text, and thus post large performance gains over other models. This represents an alternative approach to the problem.

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

3 Approach

Our baseline model is BiDAF (4). Our final model consists of combining BiDAF with character-level embeddings and self-attention. This model has 6 layers in this order: a character-level embedding layer, an RNN encoder layer, a BiDAF layer, a modeling layer, a self-attention layer, and an output layer.

3.1 Character-Level Embedding Layer

We describe our approach to the embedding layer here:

Let X be the length of the context, let M be the length of the question, let W be the word embedding size, let C be the character embedding size, and let H be the hidden size of the model.

First, we create word level embeddings. Given some input word indices $w_1, ..., w_k \in N$, the embedding layer performs an embedding lookup to convert the indices into word embeddings $v_1, ..., v_k \in R^W$. This is done for both the context and the question, producing embeddings $c_1, ..., c_X \in R^W$ for the context and $q_1, ..., q_M \in R^W$ for the question.

Second, we create character level embeddings. Given a word x of length l, we look up the index of each character in a character vocabulary, and represent x as a vector of integers: $x = [c_1, c_2, \ldots, c_3] \in Z^l$. Using a $< PAD >$ 'character', we pad (or truncate) every word so that it has length $mword$, a hyperparameter for maximum word length: $xpadded = [c_1, c_2, \ldots, c_{mword}] \in Z^{mword}$. We look up a character embedding for each of these characters c_i . This yields an embedding tensor: $x_{emb} \in R^{mword \times C}$.

We combine these character-level embeddings using a 1d convolutional network. For a detailed explanation, see CS224n, Assignment 5, pg. 3 (6). Ultimately, the convolutional network. produces the embedding $x_{conv-out} \in R^C$. This is done for both the context and the question, producing embeddings $c*_1, ..., c*_X \in R^C$ for the context and $q*_1, ..., q*_M \in R^C$ for the question.

Third, we concatenate the word-level and character-level embeddings, producing embeddings $c_1^{cat}, ..., c_X^{cat} \in R^{C+W}$ for the context and $q_1^{cat}, ..., q_M^{cat} \in R^{C+W}$ for the question. As in the baseline model, we then project each embedding to have dimensionality H, and then apply a Highway Network to refine the embedded representation. See the CS 224n Default final project handout, page 7 for further details (7). This completes our approach to the embedding layer: the other layers were left unchanged.

3.2 Encoder Layer and BiDAF Attention Layer

A good treatment of these layers, which are based directly on BiDAF, is given in the default project handout (7).

3.3 Modeling Layer

A good treatment of this layer is given in the default project handout. We substituted a bidirectional GRU for the bidirectional LSTM in this layer because we noticed a moderate improvement in training speed without a decrease in F1 score.

3.4 Self-Attention Layer

The self-attention layer is based off of the self-matching layer in RNET. Self-attention gathers evidence from the whole context according to the current passage word and question information, and encodes this information in the passage representation hP. The representation of the passage that serves as input to the self-attention layer is v Pt t=1,...,n where n is the context length. Then, we compute:

$$h_t^P = \text{BiRNN}(h_{t-1}^P, [v_t^P, c_t])$$

where $c_t = att(v^P, v_t^P)$ is an attention-pooling vector of the whole passage (v^P):

$$s_j^t = \text{v}^\text{T}\tanh(W_v^P v_j^P + W_v^{\tilde{P}} v_t^P)$$
$$a_i^t = \exp(s_i^t)/\Sigma_{j=1}^n \exp(s_j^t)$$
$$c_t = \Sigma_{i=1}^n a_i^t v_i^P$$

3.5 Output Layer

The output layer is almost exactly the same as described in the default project handout, taking in as input the outputs of the self attention layer $g_1, ..., g_N \in R^{8H}$, and the modeling layer outputs $m_1, ..., m_N \in R^{2H}$. One change we made was substituting a bidirectional GRU for the bidirectional LSTM because of the training speed benefits.

4 Experiments

4.1 Data

All of our models are trained on the official The Stanford Question Answering Dataset 2.0 (SQuAD 2.0). Refer to (1) for a detailed description of this dataset, but in brief: it consists of short passages taken from Wikipedia articles, and crowdsourced questions and answers about these passages. Each question either can be answered by a span of text in the given passage, or is unanswerable (a new feature of SQuAD 2.0 that attempts to test systems' ability to acknowledge when they *don't* know the answer to a question). Thus, the dataset models the task of reading comprehension.

The dataset has approximately 150,000 examples and exhibits a roughly 92%/4%/4% train/dev/test split, where the dev and test sets are obtained by splitting the official SQuAD 2.0 dev set in half.

4.2 Evaluation Method

The high-level goal is to maximize the accuracy of the model's answers so that, as much as possible, it 'matches' the target answer when there is one, and otherwise acknowledges that the passage does not contain the answer to the question. One complicating factor is that not all of the answers are perfectly deterministic; the dataset actually contains three crowdsourced answers for each question, many of which match, but some of which don't. There are two primary evaluation metrics with which we are judging the performance of our models, Exact Match (EM) score and F1 score. These are the official SQuAD evaluation metrics.

In short, Exact Match requires that the model's output answer *exactly* matches a given ground truth answer, yielding a score of 1 for an exact match, and 0 for any divergences. F1, at a high level, attempts to balance evaluating the precision of the model (penalizing any divergence from the ground truth answer) with the recall of the model (how completely it captures the ground truth answer). The exact equation for F1 score is given here (8). We take the maximum F1 score and maximum EM score across the three crowdsourced answers to yield the overall F1 and EM score. These scores will be entered on a leaderboard of our classmates in CS 224N, allowing us to compare the performance of our model to our peers in a standardized way.

4.3 Experimental Details

We designed and tested a number of modifications and extensions to the baseline model over the course of our experiments, which we will detail in the following section. At a high level, the most significant extensions were the incorporation of a character-level embedding layer into the model (a feature which is present in the official BiDAF implementation but which was not included in the baseline), and the implementation of a self-attention layer, inspired by Microsoft Research Asia's

R-NET paper (9). Beyond this, we made numerous tweaks to the existing modeling and output layers, as well as tuning hyperparameters to combat training problems and optimize performance. The details of these implementations are explained in the Approaches section, and the performance of the resulting models is described in the Results section; this section focuses on explaining our experimentation process, motivating our choices and justifying the steps we took.

4.3.1 Character Level Embeddings

The first model we attempted to implement was an addition of convolutional character-level embeddings to the baseline model. Char embeddings have a potentially better understanding of word variants than strict word-based embeddings since they are based on the internal structure of words (morphology). Previous experiments conducted in this class (namely in Assignment 5, which involved adding a character level embedding layer to the model built in Assignment 4) demonstrated the potential performance advantages of sub-word modeling in an NLP task. Given that character level embeddings were present in the original implementation of BiDAF, but excluded from the baseline model, we believed this would be a natural place to start.

Going into this experiment, we were uncertain as to whether the substantial performance improvements from sub-word modeling in the domain of Neural Machine Translation (NMT) would translate to the task of Question Answering (QA). However, given the similarities between the two tasks in relevant areas – i.e. the need to process text from input with a wide variety of (potentially specialized) language from a large vocabulary – we believed that there was a strong possibility of out-of-vocabulary (OOV) words inhibiting the model's overall 'understanding' of the paragraph, and thus its performance. This motivated our decision to incorporate sub-word modeling.

4.3.2 Partially Vectorized Additive Self-Attention (No RNN) + Character Level Embeddings

We found the results of character level embedding alone to be underwhelming, as it did not substantially improve on the score of the baseline model. After reading the paper 'R-Net: Machine Reading Comprehension with Self-Matching Networks' (9), we decided that the addition of a Self-Attention layer would be a worthy upgrade to our model. In particular, we felt that because "recurrent networks can only memorize limited passage context in practice," the sizeable length of many of the context paragraphs in the dataset made it likely that the baseline model would not be able to adequately consider "clues" throughout the whole passage when answering questions. Self-attention attempts to capture these clues: for each word in the context passage, self-attention examines every other passage word to attempt to aggregate evidence that may be relevant to it.

In our first attempt to implement self-attention, we coded a variation of the self-attention calculations in the R-Net paper, shown in Approaches. As written in R-Net, the algorithm is not at all 'vectorized,' meaning a direct implementation of it would be extremely slow. Seeking to limit computational cost and training time, we vectorized much of the computation in our implementation. However, the paper calls for a variation of self-attention called additive attention, and the nature of this formula made it very difficult to fully vectorize. As such, our initial implementation required iterating over every word in the context paragraph and performing the attention calculations separately on each one; this led to an extremely slow model, and precluded us from getting meaningful results.

4.3.3 Fully Vectorized Multiplicative Self-Attention (No RNN) + Character Level Embeddings

Seeking to speed up our model, we pivoted to multiplicative self-attention, as "multiplicative attention is faster and more space-efficient [than additive attention] in practice as it can be implemented more efficiently using matrix multiplication" (10). As an added benefit of this switch, the math behind multiplicative attention better lends itself to vectorization, and after switching, we were able to fully vectorize our code, removing all for loops.

Also of note is that at this stage, we decided not to include a bidirectional RNN inside the self-matching attention layer. The R-Net paper calls for the results of the self-attention calculations to be concatenated with the output from the previous layer and encoded in a bidirectional RNN before being passed to the output layer. Examining the baseline implementation, we believed that if we inserted the self-matching attention layer before the modeling layer (and immediately after the BiDAF attention layer), this would serve the same purpose and eliminate the need for an extra RNN inside the

self-attention layer. This intuition turned out to be incorrect, and as shown in the results section, this model produced very poor scores – worse than the baseline model, and increasingly bad as training went on. Clearly, at this stage, our self-attention was not functioning properly and was hindering learning more than helping it.

4.3.4 Self-Attention with BiRNN and with Modeling Layer as Input + Character Level Embeddings

At the previous step of our experiment, there were two primary flaws with our assumptions: first, we failed to realize the importance of passing the results of the Self-Attention calculations through an RNN to encode them; and second, we assumed we could feed the output of BiDAF attention directly into the self-attention layer, rather than first encoding it in the modeling layer.

We realized the first of these two mistakes immediately, and added a bidirectional RNN to our self-attention layer; this RNN encoded the concatenated output of BiDAF attention with the self-attention matrix, and this encoding was then fed into the output layer. The resulting model achieved better performance than the previous one (about 52 F1 score), but still noticeably worse than the baseline. At this point, we realized that our intuitions surrounding the ordering of the modeling layer were incorrect, and changed the model so that instead of coming after the self-attention layer, the modeling layer fed the encoded output of BiDAF attention to the self-attention layer as input.

Making this adjustment to the order of the layers required changing many of the parameters within the affected layers, most notably the dimensions; the input size of the self-attention layer doubled in size. We were concerned about the effects that the increased number of parameters would have on the training time; knowing that Gated Recurrent Unit (GRU) RNNs are less complex than LSTMs while generally achieving comparable performance (11) (12), we modified both the modeling layer and the output layer to utilize GRU encoders rather than LSTMs. We also reduced the batch size, as we began running into memory issues on our virtual machine; after experimenting with a number of values, we settled on batch size 16 (instead of the default 64) as the optimal value for training time.

The resulting model was a small breakthrough for us in that it did not exhibit the poor performance of all prior self-attention equipped networks; instead, the performance roughly matched baseline BiDAF.

4.3.5 Self-Attention + Character Embeddings with Reduced Dimensionality and Dropout and Learning Rate Decay Tuning

At this point, we had not achieved the desired performance gains with our self-attention algorithm. While we wanted to make further structural tweaks to the model, we also noticed instances where decreases in training error did not result in corresponding improvements in our dev error, and as a result hypothesized that overfitting the training set may have been holding back our models' performance. As such, we sought to perform experiments with hyperparameter tuning to combat this overfitting. We identified dropout and learning rate decay as two important and oft-adjusted parameters which could help address this issue, and set about training our existing model with different rates of dropout (0.2, 0.35, and 0.5) and learning rate decay (0, 0.001, and 0.005).

We quickly realized that our model was not efficient enough to train on each of these different values in a reasonable amount of time, and sought to make the model run faster. To do so, we decided to reduce the number of learnable parameters in the model; we changed the hidden size from 100 to 64, and reduced the output size of the GRU in the modeling layer to 50% of its original size. This yielded substantial improvements to performance (over a 200% increase in training iterations per second) without significantly compromising model performance, and allowed us to test each hyperparameter value of interest. Ultimately, we found that increasing the dropout parameter above 0.2 actually resulted in more volatile training (with frequent spikes and drops) and inferior performance overall. Meanwhile, adding learning rate decay also decreased performance; a decay value of .001 reduced the F1 score by 2 points, while a decay of .005 proved to be far too large, decreasing the F1 score by 8 points.

4.3.6 Ablation Experiment: Self-Attention without Character Embeddings

We wanted further insight into why our full self-attention model was not exceeding the performance of the baseline model, even after hyperparameter tuning to combat overfitting. We turned to an

ablation experiment: we wanted to isolate the effect of adding self-attention to our network, as all the self-attention models we had tested to date were built on top of our character embedding model.

We took our 'best' self-attention model and removed the sub-word modeling functionality from the initial encoding layer. The results were illuminating: the self-attention-only model scored slightly worse than our original extension to the baseline – a character embedding model with no self-attention, as the latter achieved a F1 score of 61.1, while the former (self-attention only) had an F1 score of 59.49. The results of this ablation experiment showed that our self-attention layer, as implemented, was actually not contributing much value at all to the performance of the overall model.

4.4 Results

Model	F1 Dev Score (High to Low)	EM Score
Char Embedding, No Attention, Batch Size 16	61.13	57.67
Baseline	61.01	58.04
Char Emb. + Self Att. (Full Model): 1 GRU Modeling Layer Before Self-Attention Layer	60.17	57.30
Char Embedding, No Attention, Batch Size 64	59.94	56.86
Char Emb. + Self Att. (Reduced Dimensionality, Default Hyperparameters): Hidden Size = 64, Modeling Output Size = 8 * Hidden Size	59.71	56.91
Self Attention, no Char Embedding	59.49	57.12
Char Emb. + Self Att. (Reduced Dimensionality): Learning Rate Decay = 0.001	57.62	54.15
Char Emb. + Self Att. (Reduced Dimensionality): Dropout = 0.5	52.21	52.21
Char Emb. + Self Att. (Reduced Dimensionality): Learning Rate Decay = 0.005	51.08	50.76
Char Emb. + Self Att. (Reduced Dimensionality): Dropout = 0.35	50.92	48.48
RNET Single Model (2017)	84.265	76.461

5 Analysis

In this section we will perform error analysis, inspecting key characteristics and outputs of our model to gain qualitative insight on its performance and areas for improvement.

5.1 Inaccurate answer length boundaries

In same cases of our model's errors, the model predicts an approximately correct answer, but leaves out a word in the ground truth answer or includes an extra word. These errors are not a significant concern for us as humans likely make them quite often as well (it is unclear exactly how long the answer should be to match the ground truth answer). For example:
Context: ...The Islamic Republic has also maintained its hold on power in Iran in spite of US economic sanctions, and has created or assisted like-minded Shia terrorist groups in Iraq, Egypt, Syria, Jordan (SCIRI) and Lebanon (Hezbollah) (two Muslim countries that also have large Shiite populations)...

Question: What Republic has maintained its control of Iran?
Answer: Islamic
Prediction: Islamic Republic

5.2 Predicting an answer when there is none

In many cases, our model predicts some answer, when the ground truth answer is "N/A". For example:
Context: .... In 2006, Internet2 announced a partnership with Level 3 Communications to launch a brand new nationwide network, boosting its capacity from 10 Gbit/s to 100 Gbit/s. In October, 2007, Internet2 officially retired Abilene and now refers to its new, higher capacity network as the Internet2 Network. . . .
Question: Who did Internet2 partner with to boost their capacity from 100 Gbit/s to 1000 Gbit/s?
Answer: N/A
Prediction: Level 3 Communications

Here, the model is attending to the right part of the passage, but is predicting an answer, when in reality there is no answer that exactly matches the specifications of the question. This can be addressed by testing different values of a decreased null threshold (making it easier for the model to predict "N/A"). In this case, the model likely interpreted "10 Gbit/s to 100 Gbit/s" as very close to "100 Gbit/s to 1000 Gbit/s", and thus deemed the answer to be under the null threshold for N/A.

5.3 Not remembering relevant context information

Our model at times fails to remember relevant passage information, assigning higher weight to words that appear later in the context. For example:
Context: ..Ogedei's grandson Kaidu refused to submit to Kublai and threatened the western frontier of Kublai's domain. The hostile but weakened Song dynasty remained an obstacle in the south. Kublai secured the northeast border in 1259 by installing the hostage prince Wonjong as the ruler of Korea, making it a Mongol tributary state. Kublai was also threatened by domestic unrest. Li Tan, the son-in-law of a powerful official, instigated a revolt against Mongol rule in 1262....
Question: Who was Kaidu's grandfather?
Answer: Ogedei
Prediction: Li Tan

Here, the model seems to associate "son in law" with grandfather, instead of "grandson" with grandfather. One potential reason for this is that the model is unable to draw a correlation between "grandson" and "grandfather", or draws a higher correlation between "grandfather" and "son in law". A potential solution for this problem is using deep contextual embeddings such as ELMo or BERT, if the squad dataset alone is not large enough for the model to learn relationships between words like this.

Since "grandson" and "grandfather" are very similar, the above explanation perhaps seems less likely than the fact that the model is assigning more weight to words later in the context ("son in law" comes much later than "grandson". This could be because our self-attention mechanism is not working properly: it is assigning enough importance to words it does not "remember" as well in the passage. A potential solution to this problem is adding an additional gated attention-based recurrent network in the attention layer before ct is passed into the biRNN (as done in RNET). Gated attention-based recurrent networks assigns different levels of importance to parts of the passage based on their relevance to the question, giving the model more accurate passage context to infer the answer. Additionally, perhaps using a GRU instead of an RNN in the self-attention layer would help the model remember long-range dependencies better (as it is adapted specifically for long-range dependencies).

5.4 Not interpreting complex questions correctly

Sometimes the model seems to misinterpret complex questions with multiple parts or conditions. Often, the model answers in a way that makes sense under a likely misinterpretation of such a question. For example:
Context: The quick and decisive defeat of the Arab troops during the Six-Day War by Israeli troops constituted a pivotal event in the Arab Muslim world. The defeat along with economic stagnation in the defeated countries, was blamed on the secular Arab nationalism of the ruling regimes. A steep and

steady decline in the popularity and credibility of secular, socialist and nationalist politics ensued....
Question: Secular Arab nationalism was blamed for both the defeat of Arab troops as well as what type of stagnation?
Answer: economic
Prediction: secular Arab

Here, the model seems to interpret the questions as "What was blamed...", instead of realizing that the question was actually asking for a type of stagnation (at the end of the question). In that case, the model simply answers "economic", which would make sense. This might be because the model has too few parameters or is not complex enough to understand complex relationships such as "both...as well as", in which case increasing the number and/or size of layers (or training on more data that includes more of these types of questions) might be beneficial.

This could also be an attention problem: the model is focusing on the wrong part of the question ("was blamed" versus "what type of stagnation"). This might be addressed by adding a gated-based recurrent network as described in 6.3, or experimenting with more complex forms of attention with more parameters, such as additive attention or other forms. However, in this project we prioritized speed due to a limited number of credits and time constraints.

6 Conclusion and Future Work

We implemented a deep learning model for question answering on the SQuAD 2.0 dataset using a BiDAF model with character embeddings and self-attention. The character embeddings improved marginally on the baseline, but self-attention (under our implementation) did not. As discussed earlier, lack of inclusion of a second GRU in self-attention and using simpler models in the name of speed likely lead to this underperformance. Going forward, we would like to add a second GRU to our attention function, and experiment with more complex models (more/larger layers, and more complex versions of attention such as additive attention).

There are several other things that we could implement to improve the model that we have not implemented due to time constraints. Our first focus will be to implement pre-trained contextual embeddings such as ELMO or BERT, which are used by the majority of state-of-the-art models on the SQuAD 2.0 leaderboard, and have been shown to increase F1 by 4-5 points. As discussed in the analysis section, we would also like to experiment with the null threshold to potentially fix wrong answers when the correct answer is "N/A".

References

[1] Rajpurkar, Pranav. "The Stanford Question Answering Dataset 2.0," GitHub.io, https://rajpurkar.github.io/SQuAD-explorer/

[2] https://rajpurkar.github.io/SQuAD-explorer/

[3] SQuAD 2.0, Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. arXiv preprint arXiv:1806.03822, 2018.

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

[5] https://github.com/chrischute/squad

[6] http://web.stanford.edu/class/cs224n/assignments/a5.pdf

[7] http://web.stanford.edu/class/cs224n/project/default-final-project-handout.pdf

[8] https://en.wikipedia.org/wiki/F1-score

[9] Natural Language Computing Group, Microsoft Research Asia, R-Net, https://www.microsoft.com/en-us/research/wp-content/uploads/2017/05/r-net.pdf, 2016

[10] http://ruder.io/deep-learning-nlp-best-practices/

[11] Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio, https://arxiv.org/pdf/1412.3555.pdf, 2014

[12] http://proceedings.mlr.press/v37/jozefowicz15.pdf

[13] Embeddings from Language Models, Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. arXiv preprint arXiv:1802.05365, 2018.

[14] Bidirectional Encoder Representations from Transformers, Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova.