High Performance SQuAD and Transfer Learning

Jeff Chen
jc1@stanford.edu

Alexandre Gauthier
agau@stanford.edu

Abstract

We use the Stanford Question Answering Dataset [1], derived from Wikipedia, to train a question answering model. We achieved an F1 score of 74.3%+ on the test set with an EM of 64.2%+ (latest results are pending CodaLab completion), which is comparable to the original Bidirectional Attention Flow publication [4] and a significant improvement over the baseline F1 of $\sim$ 43%. Key improvements include bidirectional attention, a stacked bidirectional LSTM in the modeling layer, span representations, feature generation, and parameter tuning. We demonstrate that transfer learning is possible with our trained model by achieving 68-90% correct on our hand-crafted datasets in Top News, Financial News, and Corporate Annual Report categories.

1 Introduction

Answering questions posed in natural language is a difficult and relevant problem in AI, and one in which advances are continuously being made. It is desirable for a computer system to answer questions without having to explicitly parse and extract information from text, making processing much more scalable. Further, complicated queries are not required, opening up access to a larger pool of untrained users.

The Stanford Question Answering Dataset (SQuAD) [1] is a valuable resource for training machine learning models to answer questions. Over the couple of years since the dataset has been released, computer performance increased to the point that it nearly matches human performance on this task.

2 Background & Related Work

The first question answering systems were highly domain restricted, answering only a few questions (encoded on punchcards) relating to very specific datasets [2]. Recently, neural network techniques have allowed for the creation of much more general question answering software, as evidenced by the rapid progress achieved on the SQuAD dataset.

This dataset requires the computer to answer short questions, where the answer is contained within a provided "context" document drawn from Wikipedia. Performance can be quantified using the F1 score, defined as

$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} \in [0, 1]$$

where precision is the fraction of predicted answer words which are true answer words, and recall is the fraction of true answer words that are predicted. State of the art models with increasingly elaborate neural network structures achieve an F1 score of up to 0.89 [3], compared to human performance of 0.91. This represents a significant improvement over $\sim$ 0.4 which can be achieved using a baseline composed of a relatively basic neural network architecture.

This question-context method is highly generalizable - even if a user asks a computer a question without providing a document, it is feasible to guess a Wikipedia page to use as the context. Commercial virtual assistants currently provide rudimentary question answering services; in the future these capabilities will likely greatly expand.

2.1 Dataset Details

The SQuAD dataset consists of a set of over 100,000 (question, context, span) triplets. Contexts are paragraphs up to several hundred words long. Questions are sentences of up to a couple dozen words, with an answer contained verbatim in the context. The span of this answer within the context, in terms of starting and ending indices, is also provided in the dataset. The goal is to predict the span from the context and question. The texts are input to the model as sequences of word vectors pretrained on GloVE.

We performed a statistical analysis on the dataset (Fig. 1). We found that most questions are shorter than 20 tokens, but contexts can reach 300 or more tokens. Ninety percent of answers span fewer than 5 tokens. These values inform us how small we can shrink our model to conserve memory usage and decrease training time without significantly limiting performance.

We also found that answers are evenly distributed throughout their contexts. Answers are equally likely to be found in the beginning, middle, or end of the context. This means we must be careful not to choose a maximum context length shorter than our contexts, as such truncation will likely result in unanswerable questions.

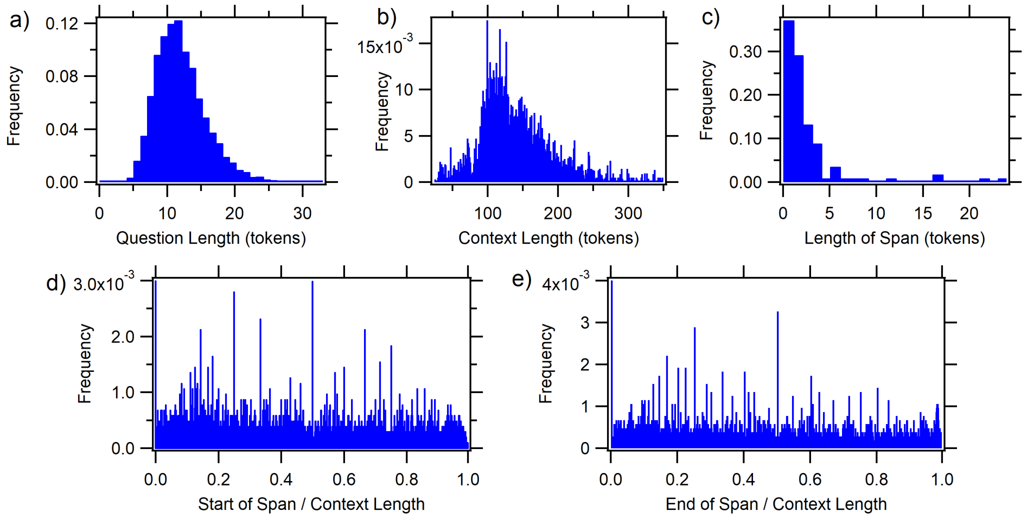


Figure 1: Dataset statistics. (a-c) show histograms of the length in tokens of the question, context and answer, respectively. (d-e) show the start and end indices of the correct answer's span within the context. Answers are evenly distributed within the contexts.

2.2 Baseline

The baseline model provided by SQuAD consists of two bidirectional GRUs which take the question and contexts as inputs. Attention is then applied so that the context hidden states attends to the question hidden states question hidden states; the attention outputs are concatenated with the context hidden states to form a "blended representation." After an additional fully-connected layer, the blended representations are fed into two separate fully-connected layers and Softmax predictors. The two sets of Softmax outputs are then taken as probabilities for the start and end positions of the answer.

3 Approach

Some of the initial tweaks we tried were: using LSTMs instead of GRUs for our bidirectional RNNs, and optimizing our hyperparameters. In addition, we implemented several more elaborate modifications to our question-answering model.

3.1 BiDAF Attention

The basic attention layer implemented in the baseline goes through the context word by word, and extracts the most relevant parts of the question for each context word. This informs the model which parts of the question are most relevant to the context. Intuitively, the model should also be able to learn the reverse - which parts of the context are most relevant to the question. This is implemented in Bidirectional Attentional Flow (BiDAF) [4] (Fig. 2(a)).

Given context and attention hidden states c_i and q_j, respectively, BiDAF starts with a similarity matrix $S_{ij} = w^T[c_i; q_j; c_i \cdot q_j]$ where w is a learned weight vector. To compute the importance of each question word to the context words, take $\alpha_i = softmax(S_{i,:})$, with the attention output being $a_i = \Sigma(\alpha_i)_j q_j$.

To compute the importance of each context word to the question words, let $\beta_i = \max_j S_{ij}$, and $\gamma = softmax(\beta)$. Then $b = \Sigma\gamma_i c_i$. The final blended states are then the concatenation $[c_i; a_i; b]$. This blended representation is then propagated to further layers.

The new b term added by BiDAF allows for the model to overweight context words that are strongly associated with the entire question in a way that the baseline attention model does not allow.

3.2 Post-Attention Stacked Bidirectional LSTM (BiLSTM)

Now that we have attention flowing both ways, it's reasonable to imagine that the blended representations could contain attention scores that indicate important words. This could mean that nearby words are also important.

To implement this intuition, we add an additional bidirectional stacked LSTM in the post-attention layer (Fig. 2(a)), also known as a model layer. This layer takes the blended representations generated by the attention layer as its inputs [4]. The outputs of this layer are then passed through the final fully-connected and Softmax layers from the baseline model. Note that this is similar to the additional RNN implemented in the span representation architecture (below); however, in that case a GRU was used to save memory.

3.3 Spans

One of the shortcomings of the baseline model is that it predicts the start and end position of the answer within the context independently. This means that occasionally the predicted end comes before the predicted start, in which case the prediction is necessarily incorrect. This sort of error represented a substantial fraction of false predictions by the baseline model.

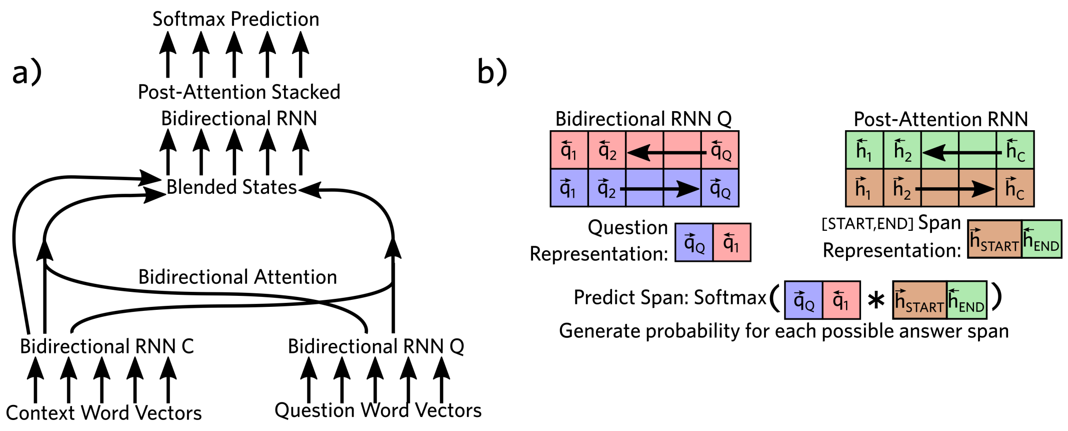
To address this problem, we can consider each possible answer span independently, generating a Softmax probability for each one (up to a specified length) [5] (Fig. 2(b)).

This "full" span representation takes the post-attention blended representations, and passes them through an additional bidirectional GRU. Each span (up to a reasonable length) is then represented by the concatenation of $[\overrightarrow{h_i}, \overleftarrow{h_j}]$ where h_i and h_j are the hidden representations at the starting and ending indices of the span, respectively. The question is represented by the concatenation of the final forward and first backward hidden states of the initial bidirectional RNN whose inputs are the question embeddings.

Un-normalized scores are generated for each possible span by taking the scalar product of the question representation with each span representation. These scores are sent through a Softmax layer to turn them into normalized probabilities. Cross-entropy loss is used during training. During testing, the span with the highest probability is predicted.

The “full” span representation method described above is very effective, but it is very slow to compute probabilities for every possible span. For spans of length S and a context of length C, this takes on the order of SC to compute - compared to O(C) for the baseline. GPU memory usage is also a concern; we are forced to reduce the hidden size, context length and batch size to conserve memory, which hurts performance.

A simpler solution is to only allow the model to consider potential answer spans (start, end) such that end > start. This requires a very minor modification to the baseline model which affects only prediction, not training. When predicting an answer span, we first choose the start position as in the baseline. We then choose end position as the argmax of the Softmax outputs *such that end > start*. The length of the predicted span is also limited to a reasonable number, e.g. 6. We call this the “simple” span representation.



a)
Softmax Prediction
Post-Attention Stacked Bidirectional RNN
Blended States
Bidirectional Attention
Bidirectional RNN C
Bidirectional RNN Q
Context Word Vectors
Question Word Vectors
b)
Bidirectional RNN Q
$\overleftarrow{q}_1$ $\overleftarrow{q}_2$ $\overleftarrow{q}_Q$
$\overrightarrow{q}_1$ $\overrightarrow{q}_2$ $\overrightarrow{q}_Q$
Post-Attention RNN
$\overleftarrow{h}_1$ $\overleftarrow{h}_2$ $\overleftarrow{h}_C$
$\overrightarrow{h}_1$ $\overrightarrow{h}_2$ $\overrightarrow{h}_C$
Question Representation: $\overrightarrow{q}_Q$ $\overleftarrow{q}_1$
[START,END] Span Representation: $\overrightarrow{h}_{START}$ $\overleftarrow{h}_{END}$
Predict Span: Softmax($\overrightarrow{q}_Q$ $\overleftarrow{q}_1$ * $\overrightarrow{h}_{START}$ $\overleftarrow{h}_{END}$)
Generate probability for each possible answer span

Figure 2: (a) Architecture of BiDAF + BiLSTM (post-attention bidirectional stacked LSTM) model (b) Schematic of the full span representation method

3.4 Features

We tried adding several additional features to the input context and question word vectors. Core Features include an Exact Match of the question word to the context word, Parts of Speech, and Named Entity Recognition. Dependency parsing includes various techniques of incorporating output from a parser.

3.4.1 Core Features: Exact Match, POS, NER

As we manually evaluated the performance of our model, we saw that the model missed some answers where the context phrasing is exactly the same as that of the question. Furthermore, predicted answers did not match the parts of speech requested by the questions (e.g. predicting a verb as an answer to a “What...” question).

We added these three features to try to address these shortcomings:

- **Exact Match**: if a context word exactly matches a word in the question, this feature is assigned a 1; else it is 0. This feature is appended to the word vectors as a single number (1 or 0).
- **Part of Speech (POS)** - using the Spacy package [6], we assigned each context and question word to one of 21 parts of speech labels. This part of speech tagging is then encoded into a 21-dimensional one-hot vector, which is appended to the input word vectors.
- **Named Entity Recognition (NER)** - the Spacy package also generates one of 19 possible NER labels for each word, which is encoded as a one-hot vector and appended to the input word vectors.

Together these core features add 41 more features to each word in the question and context.

3.4.2 Dependency Parsing

We noticed that the model struggled with questions involving lists. We speculated that a dependency parser might allow the model to recognize that all items in the list are related.

We implemented a dependency parser in two different ways - first, we append the parent word's vector to each input embedding. Second, we appended a scalar representing the number of children to each input embedding. Unfortunately, these measures hurt the model's performance. We believe this is because the model got confused between the actual input word vector, and the appended parent word vector.

4 Results

First we noticed the baseline model was overfitting, so we added L2 regularization and increased the dropout rate. We found performance peaked when we increased the dropout parameter to $p \sim 0.2$ (See Fig. 3). Increasing L2 regularization did not materially change the performance; for further tests we left the L2 hyperparameter at 0.

During tuning of regularization hyperparameters, we used default parameters given by the baseline, using batch size 100 and running for 18 epochs. It's possible that Batch Size can be increased further to achieve better results, though we were compute-limited so did not spend more time conducting hyperparameter searches.

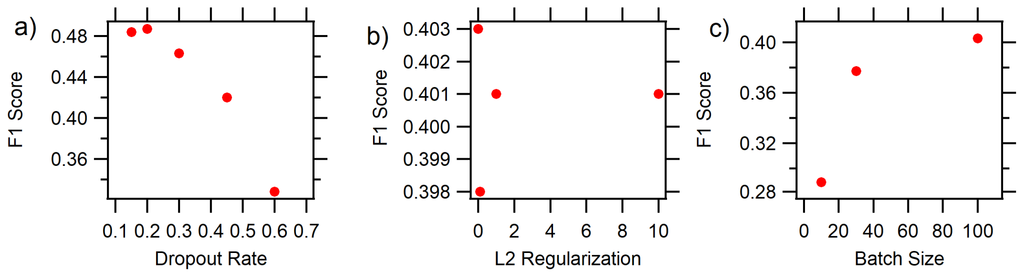


Figure 3: (a-b) Optimizing dropout and L2 regularization hyperparameters on F1 score. (c) Comparing performance on trained models as a function of batch size. All F1 scores taken after 18 epochs.

4.1 Model Modifications

The "full" span representation increased performance by 12 percentage points over the baseline model. However, it took several days to finish training. It was also extremely memory intensive; we had to decrease batch size from 100 to 10 in order to use reasonable values for other parameters (context length: 300; batch size: 10). Adding further complications such as BiDAF only worsens the memory burden. It was not practical to use this span representation method given our budget and timescale. (It is important to note that the "full" span representation also includes a post-attention RNN (a GRU), so some of its performance could be attributed to this rather than just preventing the prediction of impossible spans.)

Our final model (which combines several improvements) used the "simple" span representation method. Although this method only increased performance by 2 percentage points over baseline, it trains in a matter of hours rather than days, and is not memory intensive. In fact, this model does modify training code, it only changes test-time operations.

Adding BiDAF improved F1 scores over baseline by 8 percentage points. Adding the post-attention stacked LSTM further improved F1 by 19 percentage points. The fact that a post-attention RNN, which takes blended representations as input, works so well suggests that there's a lot of spatial structure left over in the blended representations after the attention layer. The BiDAF attention, in particular, is adding structure to the blended representations indicating which context words are most relevant to the question as a whole.

METHOD	TEST	DEV	LOCAL DEV
Bi-LSTM + BiDAF + Core Features + Simple Spans	**Pending**	0.754	0.702
Bi-LSTM + BiDAF + Simple Spans	**0.743**	0.735	0.686
Bi-LSTM + BiDAF + Dropout			0.673
BiDAF + Dropout Tuning			0.486
BiDAF			0.484
“Full” Span Representation			0.521
“Simple” Span Representation			0.442
Baseline		0.430	0.403

Table 1: Summary of Improvements and F1 scores. All experiments were run for 18 epochs. ”Local” refers to the locally run evaluation as opposed to the official online leaderboards. Official Test and Dev scores are higher than ”Local” scores because official evaluation uses 3 possible answers versus 1. Due to submission limits, not all models were submitted for all evaluations.

Core Features further improved F1; we achieved an approximately 2% improvement on F1 with Context to Question Exact Match implemented, along with applying POS and NER to both question and context.

Surprisingly, several other techniques did not work. Bi-directional direct match between Question and Context performed worse; we suspect this was due to occasional convoluted question phrasing. Also, using separate RNN encoders for the question and context hurt performance, revealing the importance of being able to recognize when the question is in the same structure as the context. Perhaps the most unexpected finding of all was that dependency parsing caused the model to perform worse in every implementation we tried. We believe appending the head word embedding to the input vectors was confusing the model because it no longer knew which word to focus on. Counting the number of children likely did not help the model understand the meaning of the key head words.

Our final model combines all four of our main optimizations - simple span representation, BiDAF, a post-attention layer stacked LSTM (BiLSTM), and core features - along with hyperparameter tuning. After 18 epochs, this model achieved an F1 score of 75.4% on the official dev set. Results on the Codalab test set are pending. By comparing to previous results, we believe the official test score will be $\sim 75\%$, slightly higher than the model without core features. (We hope to report these results in the Statement of Additional Information.) This performance is a remarkable improvement over the baseline model at 40.3%. See Table 1 for a quantitative summary of model performance.

4.1.1 Effect of Batch Size on Training

We always used a batch size of 100 during training, except on the “full” span representation model, where we had to decrease the batch size to 10 for memory conservation.

To determine whether this is expected to impact model performance, we trained the baseline model for 18 epochs using batch sizes of 100, 30 and 5 (see Fig. 3(c)). After 18 epochs, batch size 5 had an F1 score 30% lower than batch size 100. However, the models with smaller batch size were continuing to train after 18 epochs, while the batch size 100 model’s performance had stabilized.

This suggests that decreasing batch size is expected to impact performance, at least if you don’t train for weeks. You should train with the largest batch size possible given memory and time constraints. This make the “full” span representation model even more impressive, since it displays significant performance gains despite its decreased batch size.

4.2 Qualitative Analysis of Model Performance

Our best F1 score recorded on the test set was 74.3% - the local dev score reported above was lower because it does not account for multiple possible answers to each question, unlike official eval on the test set. After examining predicted and true answers on individual questions, we identified a few types of questions the computer had trouble answering:

DATASET	# CORRECT
Financial News	17/25 = 68%
General News	27/30 = 90%
Corporate Annual Reports	22/30 = 73%

Table 2: Transfer Learning Results. We hand created 3 small datasets of 25-30 examples and hand-graded the predicted answers as Correct or Incorrect as a proxy to the F1 score.

- Questions phrased as "Besides X...," where the correct answer must be extracted from a list of two. This problem could be addressed by adding a dependency parser, which could help the model learn which items are part of a list.
- Questions requiring "common sense" or some other background knowledge. For instance, the model could not answer the question "Which July 4th holiday movie....?" paired with a list of movies "... X-Men, Independence Day, James Bond ..."
- When there is a long distance in the context between the answer span, and the words that tell you where the answer is. For instance, given the context "Putin, born in 1952, used to work for the KGB and enjoys taking his shirt off. This man is also the President of Russia" the model might have a hard time answering "Who is the President of Russia?"

See the supplementary material (Appendix A) for examples of correctly and incorrectly predicted answers as described above.

4.3 Transfer Learning

To check if our model is translatable to other datasets, we assembled micro datasets of 25-30 questions on articles taken from the financial news column *Lex* [7], corporate annual reports, and general news (news.google.com) [8]. Since these sources are often written in a different style from the Wikipedia SQuAD questions, good performance here would indicate that the model is highly generalizable. See the supplementary material (Appendix B) for examples of questions, contexts and answers from these datasets.

Table 2 shows the performance of our model on these datasets. We evaluated the answers manually, accepting a wide range of logical answers; as such, these figures will be slightly higher than they would be using automatic evaluation. On such a small dataset, we cannot meaningfully quote any quantitative scores such as F1. However, this reasonably good performance suggests that our model is potentially transferable to other genres of data. Note that the general news questions were all very easy questions compared to SQuAD, explaining its anomalously high performance.

5 Conclusion

Adding bidirectional attention and a post-attention bidirectional stacked LSTM to the baseline model significantly improved performance as quantified by the F1 score. Furthermore, using span representation (or a simple approximation) solved the issue of the model predicting impossible spans which end before they begin, boosting our final test F1 score by two additional percentage points to 74.3%. Finally, adding core features of exact match, POS, and NER improved the F1 score by an additional $\sim$2%. Comparison with three other micro datasets shows that this model is quite transferable to slightly different styles of text.

Hand-crafting features can improve performance, though we understand there are drawbacks. First, we now introduce another source of error because our model is no longer end-to-end. Spacy's NER accuracy is $\sim$85%; POS $\sim$97%. Further, our features require two other models, so this technique will not work for languages where POS and NER models do not exist.

We found memory limitations and training speed to be severe limiting factors in natural language processing, which was surprising given modern computing resources. Our models frequently took 10+ hours to train. When we tried large models such as Full Span, the 840B Glove Vectors, or larger hidden sizes, TensorFlow crashed due to Out of Memory Errors.

Issues which remain include correctly predicting when the answer is one element inside of a list and the computer needs to understand the relationship of all list elements, and dealing with complicated sentence structure in the context. We expect to get a further boost in performance by using the full span representation method rather than its simplification, but this would require excessive amount of training time and memory.

Finally, creating an ensemble would likely improve our model F1 performance by an additional 3-5%, this can be done by running 5-7 of the best models we trained and then adding then performing an heuristic to select the best start and end positions of the answer.

References

[1] P. Rajpurkar *et al.*, CoRR, abs/1606.05250 (2016)

[2] B. F. Green *et al.*, web.stanford.edu/class/linguist289/p219-green.pdf (1961)

[3] M. Hu *et al.*, arXiv:1705.02798v3 (2017)

[4] M. Seo *et al.*, arXiv:1611.01603v5 (2017)

[5] Y. Yu *et al.*, arXiv:1610.09996v2 (2016)

[6] pypi.python.org/pypi/spacy

[7] ft.com/lex

[8] Annual report and general news datasets assembled by K. Sill