SQuAD with LSTM and BiDAF

Ethson Villegas
Department of Computer Science
Stanford University
Stanford, CA 94305
ethson@stanford.edu

Danielle Siy
Department of Computer Science
Stanford University
Stanford, CA 94305
beasiy@stanford.edu

Abstract

For the final project of CS224N, we tested a variety of modifications to the baseline model in terms of embeddings, attention, regularization, layer additions, as well as ensembling. Our final model comprises of Character-level CNN encoding, word embeddings, exact match feature search, BiDAF attention, a modeling component comprising of 2 bi-directional LSTM layers and a modification on the output layer which adds a bi-directional LSTM in order to find the end position . Our final F1 and EM scores on the test set are 0.57 and 0.61 respectively. We also have a model with different hyper-parameters and improved variable scoping but needs to run more iterations at this point.

1 Introduction

Our model tackles machine comprehension (MC), a specific task that falls under question answering (QA). These tasks have seen a rise in popularity over the past few years, especially in the Natural Language Processing (NLP) community, because of a wider availability of data to train on such as Stanford University's SQuAD dataset, which we use for this project. MC as an exploration in the QA space has livened in recent times as the surge of popularity using neural network architectures has led to advances in scalability and robustness that give them superior performance over traditional NLP methods. Our goal is, given a question and a paragraph of context surrounding, to predict the answer span, that is, the start and end indexes of the answer from within the context data.

For this project, our final model uses: BiDAF model attention, Character-level CNN, Bi-directional 2-layer LSTM, Exact Match feature embedding, and an End Position LSTM layer. We started with the provided baseline model which uses basic attention and bi-directional GRU for embeddings. We then implemented the BiDAF model attention described in *Bi-Directional Attention Flow for Machine Comprehension* [3] followed by the Character-level CNN. Next, we gave our model increased non-linearity through the 2-layer LSTM and then added Dropout to all LSTM layers. Some other features we tried but did not include in our final model were "smarter" answer spans, attenuated learning rates, and exact matching [2].

2 Related Work

As touched upon prior, neural network based approaches to MC tasks are relatively novel, with newer developments notably centered on research around the SQuAD dataset. Delving into developing our own model, we explored several academic papers, which developed and implemented architectures that were previously unexplored. Notable variants, some of which we borrowed or tested features from are discussed below.

Perhaps the paper that had the greatest influence on our project was Farhadi et al.'s *Bi-Directional Attention Flow for Machine Comprehension* [3]. Their approach involved a multi-stage hierarchical process that involved bi-directional attention flow to obtain query-aware context representations. This model was named the BiDAF model. Key features of this model besides the attention model was the use of Character-level CNN, a highway network, and a modeling layer that used two layers of bi-directional LSTM's. This model achieved state-of-the-art F1 and EM scores, outperforming all prior approaches for the SQuAD test set leaderboard at the time of publication.

Another approach that we explored was Bordes et al.'s *Reading Wikipedia to Answer Open-Domain Questions* [2]. Here, they introduce the DrQA model which comprises of two components: a Document Retriever and Document Reader. The Document Retriever uses a non-machine learning system, however, so we focus on the Document Reader. This second component is a multi-layer recurrent neural network that uses a collection of enhancements such as answer span restrictions and exact matching (discussed in Section 4).

3 Model

In the following we describe our system which can be split into five main components: (1) the embedding layer, (2) the encoder layer, (3) the attention model, for which we use Bi-Directional Attention Flow, (4) the modeling layer, which is comprised of a dual layer Bi-Directional LSTM, and (5) the output layer.

3.1 Embedding layer

The embedding layer is split into three components: (1) a character level embedding, (2) the GloVe word embeddings, (3) exact match embedding.

3.1.1 Character Level Embedding - CNN

The Character level embedding is tasked with mapping each character to a vocabulary embedding matrix. As per Kim [5], we use Convolutional Neural Networks (CNN) to get a temporal-spatial relation among the characters in each word per batch. This was founded upon exploration of the data we are training on and on the queries we examined. We found characters that were sometimes used which were niche to a specific language and could not be modeled properly by an English only embedding. If a sentence contains a couple characters that are common in Spanish but not in English, then it's likely that the answer is related to some extent to this subspace of the data (Spanish related). With this intuition we followed Kim's model and added a CNN embedding layer.

Characters are embedded into vectors that we can essentially think of as 1D inputs the CNN. We use a filter size of 100, a maximum characters per word length of 20, padded if it goes under and truncated if it goes over that value, and a embedding dimension of 30, which is lower than our character vocabulary size of 50 which we use as the embedding matrix. The outputs of the CNN are then max-pooled over its width to obtain a fixed-size vector for each word. This is then reshaped and concatenated with the GloVe embeddings. This is done for both questions and context.

It adds the maximum characters per word length to the dimensionality of each word, which using our above values means an addition to the dimensionality by 20 per context/question. This is m_w.

3.1.2 GloVe Word Embedding

The word embedding layer maps each word to a high dimensional vector. For this we used pre-trained GloVe word vectors to obtain the embeddings for each word. This is done for both questions and context.

For our model we use the 300d GloVe vectors, adding a dimensionality of 300 per context/question.

3.1.3 Exact match embedding

For exact match we use two simple binary features for the context based on the questions. The features indicate whether the word can be exactly matched to another word in the question in either its original form or in its lowercase form. This was only done for the context and we padded the questions to match the dimensionality of both vectors, which is necessary for the encoder layer.

After the embedding is completed, each vector has an added dimensionality of 2, for each feature, per context/question. This is $\mathrm{e_m}$.

3.2 Encoder layer - RNN GRU encoder

After both the questions and context have passed through the embedding layers they both have dimensionality of $d + m_{\mathrm{w}} + e_{\mathrm{m}}$, for a total of $300 + 20 + 2 = 322$ dimensionality per embedding for both context and questions.

The embeddings are then fed to a 1-layer bidirectional GRU, which shares weights between context and question which yield the hidden states for our model. This layer remains largely untouched from the original implemented in the handout.

3.3 Attention Model - BiDAF

The Bi-Directional Attention Flow model is the core part of the model, and represents a novel approach from more simple attention mechanisms. The core idea behind it is that attention should flow both from the question to the context and from the context to the question, hence its name as Bi-Directional. For this model we followed the specifications delineated in the handout

3.4 Modeling layer

The modeling layer is arguably one of the most important pieces of the model, as it results in the highest independent improvement. In the handout we are given a rudimentary, fully connected layer as the model but this fails to take into account temporal relations and cannot pass information from neighboring states efficiently. As proposed by Farhadi et al. [3] we opted for using a 2 layer LSTM model with dropout in order to help capture the intertwined interactions between multiple contexts, as it saves states per run, conditioned on the multitude of queries available. We feel that this ability to retain a sense of context over time, on top of the attention, allows the model to discern questions better.

3.5 Output layer

The output layer is almost the same as the one defined in the handout, with the inclusion of a 1 layer bidirectional LSTM RNN to predict the end position after the start position has been calculated. By doing this, we obtain further contextual interactions which can help the ending position be more accurate. The remainder of the output remains largely untouched

4 Experiments

4.1 Dataset

As the majority of advances in MC are reliant on the availability of a large dataset for training, we decided to use the SQuAD dataset. This dataset is ideal not only in that it has over 100,000 question-answer pairs, but also in that it is manually-labeled by crowd-sourcing. This is significant in that human performance still outperforms all models for our QA task. Further, the SQuAD dataset is amenable to our constraints of needing the answer within the given context, as we return an answer span.

4.2 Methodology

In order to enhance the model we aimed to follow a method that would allow the least amount of uncertainty to go into every feature test. Every iteration would tell us about how different features that we tested on our model would affect our performance individually and, once they passed a certain number of iterations with improved performance, how they would act in unison with the rest of the model. It was in this way that we tested the performance benefits of exact matching and smarter span selection, accepting one into our model while rejecting the other despite promising benefits in theory.

As for how we chose to implement features before testing, we based it on qualitative analysis of our data along with comparisons against previous works and past results. One salient example comes from our choice of use of a CNN, which was not only based on the BiDAF [3] paper, but also on our manual classification of errors on the baseline for sentences that used characters that were not standard in the English language, as these would often bias heavily towards non standard English context as well. Our main model uses BiDAF, which we chose since it displays one of the most promising end to end models. Using this as an experimental baseline we tested features that had not been implemented on the original paper which we thought had been overlooked, such as exact matching and smarter span selection.

As far as our final model goes, we used 600D context length along with a restricted 30D question length, the batch size remained the same at 100D with padding and masking to maintain consistency throughout. Our character level encoding used a 30D embedding size, to reduce the dimensionality from our 50D embedding matrix/vocabulary, the max word length per character was capped at 20 and padded and masked as needed and for the convolutions we used a 100D filter size along with a width of 5. Our word vectors followed standard GloVe dimensionality of 100D with an embedding size of 300D. We trained our model on a M60 NVIDIA GPU.

4.3 Evaluation Metrics

Our evaluation on the success of features was holistic and involved an amalgamation of quantitative and qualitative measures. On the quantitative end, we looked at F1 and EM scores which are the harmonic mean of precision and recall and a binary measure of whether the model output exactly matches the ground truth answer respectively. We also took into account iteration runtime and number of parameters (storage) although optimizing for these was not the focus of our project.

In terms of qualitative results, we did error analysis on our models to see what types of questions they were getting wrong. We did this by manual classification of errors of models run with the –model=show_examples flag. We looked at a several hundred examples per model.

4.4 Results and Evaluation

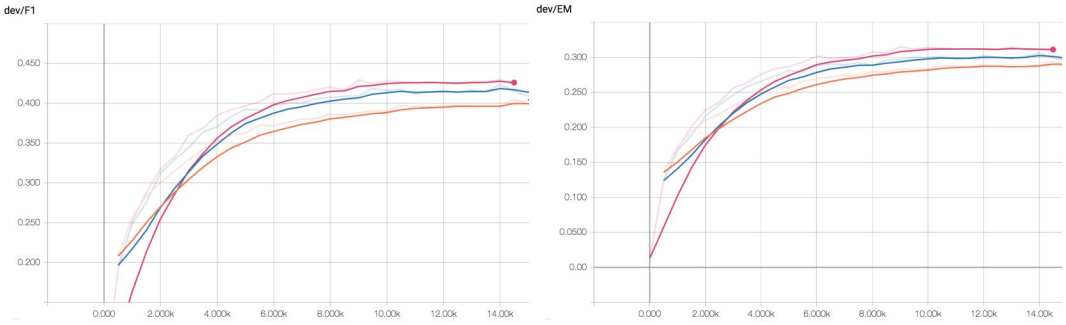
In this section we discuss the results of all our implemented features. The first subsection discusses the main elements that comprise our final model, and the second looks at some of the features that we implemented but decided not to include.

4.4.1 Final Model Features

BiDAF Attention Layer and Character Level CNN

The first feature we implemented was the BiDAF Attention Layer. Our implementation is described in section 3.3 and comes from the BiDAF paper's description [3]. Like the paper, our baseline model saw a sure, but modest improvement of about 2% for both F1 and EM.

Likewise, the second enhancement we implemented, the Character-level CNN also taken from the same paper, described in section 3.1.1 when run with the enhanced BiDAF Attention model also led to an approximate 2% increase in both measurements. Although we would have liked to see greater differentiation, that is, more drastic effects in terms of improvement, these two features were consistently net positive even when more complex models were implemented. The same could not be said for other features such as exact matching or some regularization.



Legend: orange=baseline, blue=bidaf, pink=bidaf+cnn

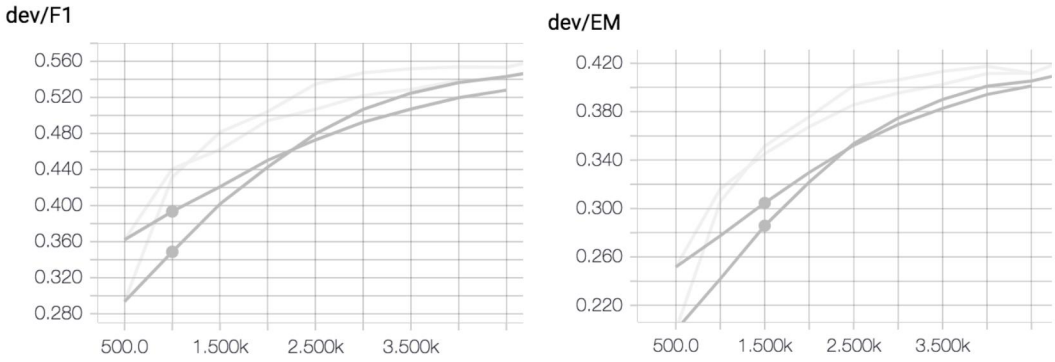
In terms of effect to iteration speed, we found that having both these features added to the baseline model made each iteration take 1.6x the original time. However, since we weren't optimizing for time, we decided to leave them in the final implementation, although we did turn them off for training more complex models to speed up training time.

In terms of error analysis, we saw that the model often guessed longer spans than necessary, especially with the one-word answers. Thus, our response to this was to implement Smarter Spans, described in section 4.4.2.

As for error analysis for CNN, as previously discussed, we found that a large number of questions that the model didn't score as well on were due to the nature of the characters. We inferred that the model would lump every word that it couldn't recognize as an UNK, even though some of the characters in it can clearly tell it's from a specific linguistic context, such as another language. An example of this would be question "In which tude of neumes rythmiques do the primes 41 , 43 , 47 and 53 appear in ?". For this, the model would interpret the unknown characters, and thus unknown word, the same as an English mispelled word and lead to errors. To fix it we implemented the CNN layer and it was able to better classify these kinds of examples.

LSTM Layers

We also implemented dual bi-directional LSTM layers for our modeling layer. This showed the most drastic improvement in terms of F1, EM, and dev loss. It increased our F1 scores and EM scores by about 20 percentage points. This, however, came at a significant reduction on speed as each layer essentially doubled the number of parameters that needed to be modelled. This is due to our choice of outputs layers that are double our hidden size, along with feeding the attention model's results with a dimensionality of 8 times the hidden size, all for the context length and per batch. Due to this, some reductions in the model parameters were necessary, and a choice of unloading some memory into the CPU was chosen in order to retain a higher dimensionality of encoding while compromising speed at a minimum. Overall, the layers saw an increase on our iteration time of almost 3 times as much unhindered, and only 2 times as much with reduction on parameters. By cutting context length in half we were able to fit all parameters in GPU memory while taking a minimal hit in performance due to the LSTM layer weight matrices being shared across, essentially simulating a larger context area without the need to load it all on memory



Legend: starts lower=lstm, starts higher=lstm + exact matching

4.4.2 Other Tested Features

Regularization: L2

Early on, we identified our model over-fitting to the training data as an issue. With or without our other enhancements to the baseline model such as the BiDAF Attention Layer or CNN, the model would over-fit within the first 4,000 - 15,000 iterations, depending on the implementation. As such, we wanted to counteract this over-fitting through the use of regularization on the trainable parameters. We tried this through applying L2 regularization across the whole model using tf.trainable_variables and tf.contrib.layers.l1_l2_regularizer, remembering to apply the penalty term to the loss.

This actually killed our model's performance, however, increasing loss. Upon further research we discovered that L2 regularization might kill RNN performance. The Bengio et al. paper *On the difficulty of training recurrent neural networks* explains that L1/L2 regularizing RNN cells hinders the cells' ability to learn and retain information through time [1]. This was the case with our model and so we decided to use Dropout instead.

Smarter Spans

From the DrQA paper, we explored the idea of smarter answer spans [2]. In the paper, they describe that for some start and end indexes *i* and *j*,

$$i \leq j \leq i + 15 \quad \text{and} \quad P_{start}(i)P_{end}(j) \quad \text{is maximized}$$

In the baseline model provided, the predicted span is the argmaxes of the start and end indexes taken separately. This means that it is possible that the end location returned comes before the start location. Printing out these indexes, we found this to occur fairly often, many times for almost 20% of the batch. Thus, we thought that implementing this feature would be an asset to our F1 score, even if it might cost some of our EM.

This feature was implemented using np.einsum to take the outer product of the start and end index probability matrices row-wise, reshaped to a 2d matrix and np.argsort to get the highest probability combination. The code for this is submitted in Supplementary Materials.

We ultimately decided not to use this feature because the marginal gain in F1 (about 2%) more than halved our EM score. Furthermore, this F1 gain diminished with further iterations. As disappointing as these results were, they make sense because printing out the resultant prediction matrices showed us that a significant number of new predictions replacing formerly empty answer spans were 1 word long. While this was good for the 25% of SQuAD answers that were indeed 1 word long, this was also indicative of incomplete answers, thus causing out EM to suffer and only marginally improve our F1 score.

We also tried variations of "smarter" span selection such as making this constraint only when $j \geq or > i$, having a minimum answer span, etc... These attempts did not perform notably better. A

sample result where we allowed spans of up to length 30 is shown below where the first matrix is the start indexes and the second matrix is the end indexes.

```
[ 52  35 130  18  91 111 135 108  39  86  12  98  78  40   5  79 106  72
  17  71  38 242  72   0  13  48  10 105  22  50  24   8  24  13  54  93
  96  73   0  45   9  30 100  79  24 141  64   8  41 151   5  40  13  61
  88  88  53  44  73  79  64  43  20  53  56  24  56  92 127  12  46 117
   7 103  62  57  50  97  44 114 100  21 111  11   2  45  49  11  78  42
 101  77   3 120  25  34  45  16  85  66]
[ 66  36 138  19  97 112 138 112  42  87  28  99  79  41  11  80 108  73
  18  80  53 243  73   1  16  60  40 135  50  66  26  10  25  36  55 108
  97  85  15  47  11  31 115 106  51 167  75  12  52 177  22  41  14  89
  89  94  69  62  76  80  75  59  22  55  66  27  66 112 128  22  47 118
  32 107  88  70  66 106  62 115 101  22 112  14  22  46  61  32  80  45
 102  78  27 129  47  35  46  17 104  72]
```

5 Conclusion

Our model implemented BiDAF model attention, Character-level CNN, Bi-directional 2-layer LSTM, Exact Match feature embedding, and an End Position LSTM layer. This implementation achieved a 0.58 F1 score along with a 0.44 EM score on the dev set.

From component analysis, the most effective feature added was the dual-layer bi-directional LSTM's. Adding this component gave us 20% increase in F1 and EM from the baseline model. A qualitative improvement from Character-level CNN was that words from other languages were better represented and thus answer and context spans containing them saw an increased correctness. The other features we implemented gave much more modest improvements in terms of scores but sometimes markedly improved other measures. For example, exact matching had negligible effects on our final quantitative scores but allowed these values to converge at much earlier iterations, dividing it by factor 4.

Overall, this project was a great exploration into the sphere of novel neural network architectures for MC and beyond. The principles that were applied to the models in our implementation as well as in related work are portable to other problems in the QA space. For further research, we would like to further tune our hyper-parameters since we feel that this was the drawback of our exploration. In this project, we focused on implementation and feature-adding rather than fine-tuning these parameters which could give us a much better model. Note that as mentioned in the abstract, we also have an enhanced model with different hyper-parameters and improved variable scoping but did not finish training in time to write this paper (in supplementary materials).

6 References

[1] Pascanu, R., Mikolov, T., and Bengio, Y. (2013b). On the difficulty of training recurrent neural networks. In Proceedings of the 30th International Conference on Machine Learning (ICML 2013).

[2] Danqi Chen and (2017). Reading Wikipedia to Answer Open-Domain Questions. CoRR, abs/1704.00051.

[3] Farhadi Ali and (2016). Bidirectional Attention Flow for Machine Comprehension. CoRR, abs/1611.01603.

[4] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever & Ruslan Salakhutdinov (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research, 15, 1929-1958.

[5] Yoon Kim (2014). Convolutional Neural Networks for Sentence Classification. CoRR, abs/1408.5882.