Machine Learning Optimizations for SQuAD

Julian Villalpando
Department of Computer Science
Stanford University
Stanford, CA 94305
jvilla32@stanford.edu

Abstract

The baseline codebase offered by the CS224 instructing team offers a solid foundation to build upon. By adding simple, yet powerful upgrades, I was able to improve upon the baseline model in meaningful ways. In this paper, I will detail the implementation and impact of my added features.

1 Introduction

Information retrieval has come a long way – especially in the last few years. Where there used to be teams of 100 hand-crafting feature sets, there are now end-to-end models producing stronger, more robust results. With the recent introduction of attention mechanisms, performances have reached all-time highs. Below, I will detail the techniques that I used to improve upon the baseline model – which include: bidirectional attention flow, feature engineering, and improved span selection

2 Dataset

The Stanford Question Answering Dataset (SQuAD) was generated from over 500 Wikipedia articles and contains over 100,000 question, context, span, and answer pairs. The context represents the body of text, from which the question can be answered. The question has a well defined answer, with the span being the start and ending indices of the answer.

Before developing, I generated several histograms that gave me insight into the general trends of the SQuAD dataset. The most telling feature of the dataset that I found was that most of the answers were relatively short:

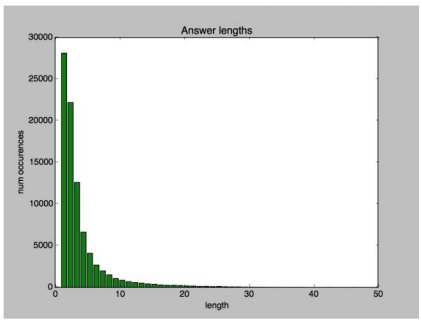

Figure 1: SQuAD Answer lengths

It seemed seems that ~90% of the answers spanned less than 15 indices, which corroborates well with the DrQA paper [1], who capped their answers at 15 in length.

The next thing I noted was that the context lengths hardly, if ever, reached 600 words in size:

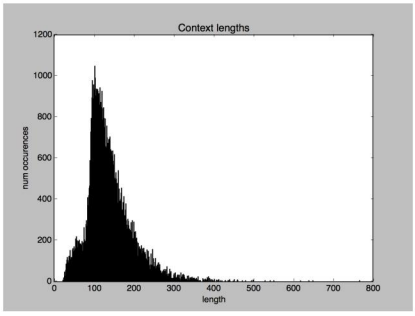

Figure 2: SQuAD Context Lengths

Lastly, I measured both the question lengths and the index of the true start position in the context.

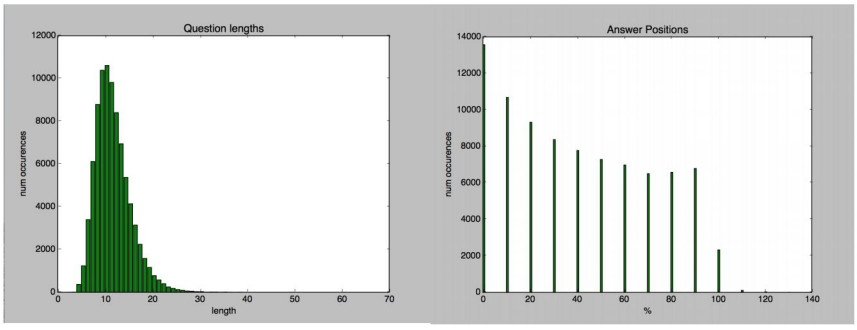

Figure 3: SQuAD Question Lengths and Answer Positions

3 Architecture

3.1 Overview

My model, like the baseline, is an encoder-decoder neural network. Every word is converted into a vector representation, and then passed into the encoder. This encoder then produces an attention output and distribution, which feeds the decoder. Finally, given the final output distribution, start and end positions are predicted.

3.2 Word Representations

Every context and question is represented as a matrix of word representations. Every word is converted to a pre-trained, static glove vector, which is then concatenated with a feature vector. The feature vector is quite simple – it represents whether a context word appears in the question.

3.3 Encoder

The word representations are then passed through a bidirectional recurrent neural net. The forward and backward states are concatenated together, to give the final encodings. The encoder is shared between the context and question encodings.

3.4 Bidirectional Attention

My architecture uses the bidirectional attention mechanism proposed in [2]. Given the context and question encodings, the bidirectional attention layer computes a question-to-context (Q2C) and context-to-question (C2Q) attention. It outputs a blended representation of the encoded states as a concatenation of the encoded context, the C2Q output, the encoded context * C2Q output, and the encoded context * Q2C output.

Equation 1: BiDAF output

$$\boldsymbol{b}_i = [\boldsymbol{c}_i; \boldsymbol{a}_i; \boldsymbol{c}_i \circ \boldsymbol{a}_i; \boldsymbol{c}_i \circ \boldsymbol{c}'] \in \mathbb{R}^{8h} \quad \forall i \in \{1, \ldots, N\}$$

3.5 Decoder

The decoder layer takes the blended representations and feeds it through a fully connected layer, followed by a RELU non-linearity. That output is then passed through a downprojecting linear layer, at which point the scores for each context location are softmaxed.

3.6 Loss

A cross-entropy loss is taken of both the start and end locations to calculate the loss, and is minimized by the Adam optimizer.

3.7 Prediction

At test time, the model predicts the start and end indices – s and e – that maximize: $p_s * p_e$ where e <= s + 15

3.8 Evaluation

F1 and EM scores are used to evaluate the model – where F1 measures precision and recall, while the EM measure exact matching.

3.9 Hyper parameters

Context lengths were capped to 400, which showed no significant difference in performance:

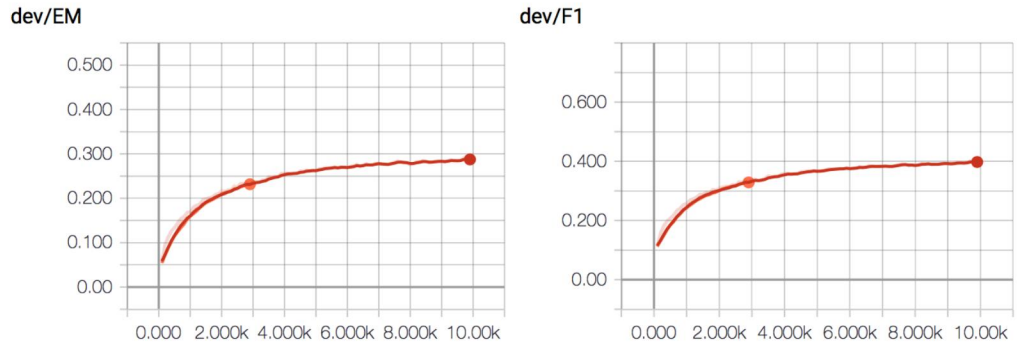

Figure 4: Context lengths 400 vs 600

Batch sizes were reduced to 32 to accommodate the Microsoft Azure memory limits, which the bidirectional attention flow tended to surpass.

All else was held constant.

3.10 Results

The final results for the model were:

Table 1: Results

	F1	EM
Train	64.472	54.938
Dev	67.643	56.197
Test	67.632	56.572

4 Improvements and Analysis

4.2 Featurization

The simplistic addition of featuring whether or not a context word appears in the question made a significant impact in performance:

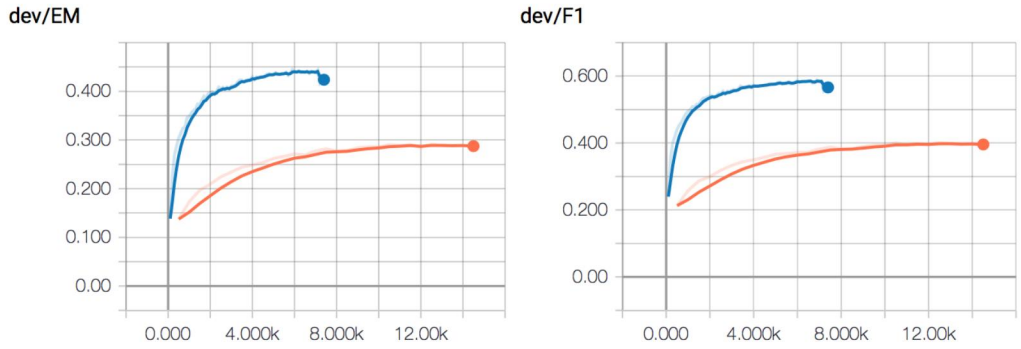

Figure 5: Baseline vs Featurization

This makes intuitive sense; many of the words in the question are in close vicinity to the answer span in a context question, so the simple feature makes a pronounced difference.

I think that end-to-end models are significantly more powerful than feature-engineered models. We've seen this in class, with regards to the models built by big teams that tried to handcraft all the rules of the English language. End-to-end models have proven not only more powerful, but also significantly easier to develop. However, as seen here, Supplementing features on top of an end-to-end model can give the best of both worlds, and a simple feature can make a pronounced difference.

4.3 Bidirectional Attention

I anticipated that bidirectional attention flow would be my largest performance improvement. That didn't turn out to be the case:

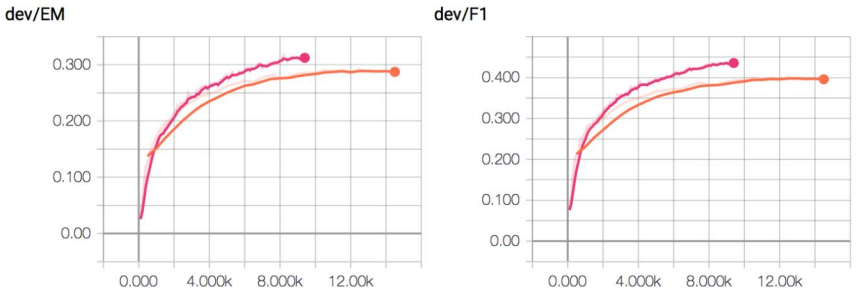

Figure 6: Baseline vs Bidirectional Attention Flow

Upon consulting with peers, I was told that adding a LSTM layer at the end of attention layer, as in the paper, significantly boosted performance. I find this surprising, but understandable. I suppose the attention outputs given in the paper were optimized with the final layer in mind. Thus, their model was able to make better sense of my blended representation detailed in Equation 1.

4.4 Test-time Prediction

One of the biggest problems I noticed in the baseline model, when generating examples, was that the end index would appear before the start index. Given how frequently I saw it, I had guessed that constraining: start < end < start + 15 would be a significant improvement. I got the following results:

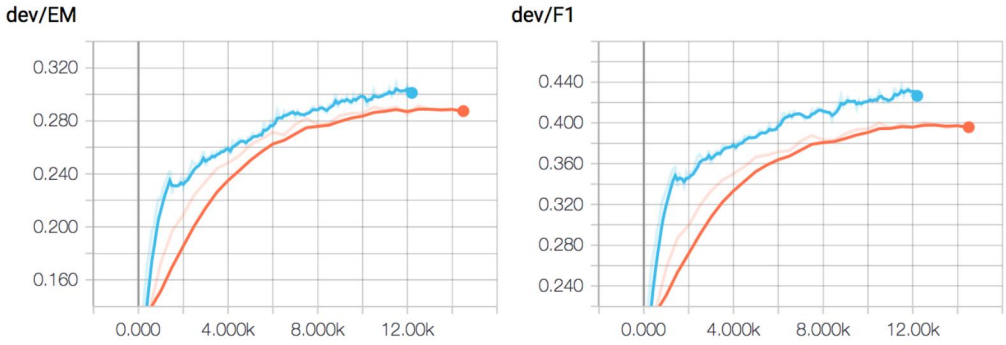

Figure 7: Baseline vs Smarter Span Selection

While the performance improved consistently over all iterations, the improvement was not as large as I thought it would be. I suppose might occur because the start and end distributions are still calculated separately, and the loss + gradient is determined from them. Meanwhile, the F1 and EM scores, are calculated from the joint probability, so the weights were not directly optimizing for F1 and EM. Nevertheless, the improvements were significant and thus kept for the final implementation. I think conditioning the end distribution on the start distribution – as in the Answer Pointer implementation [4] - would have paired nicely with this heuristic, but I was unable to successfully implement it as described in the handout.

4.5 Self Attention

In my attempt to improve the model, I implemented self-attention as proposed in [3]. However, given the memory constraints, I had to reduce batch sizes to 20, context lengths to 350, and hidden sizes to 100 to run the model on Microsoft Azure. I believe that given these constraints, I was unable to produce a self-attention model that surpassed the less memory-intensive baseline

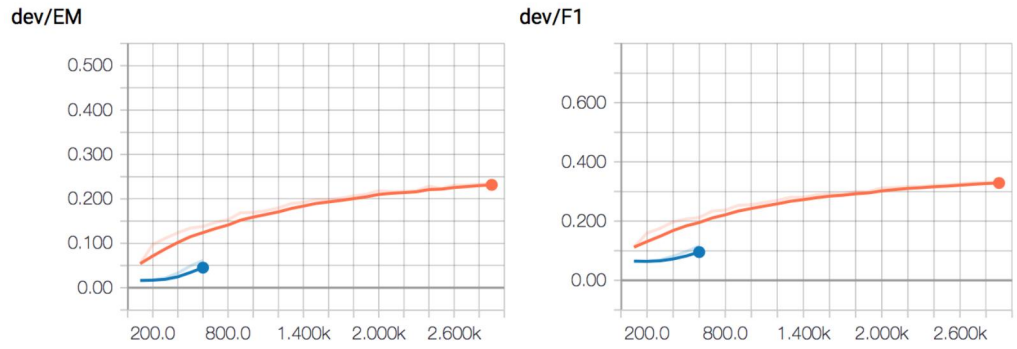

Figure 8: Baseline vs Self Attention

4.6 Final Product

As detailed in the Table 1, my final performance was:

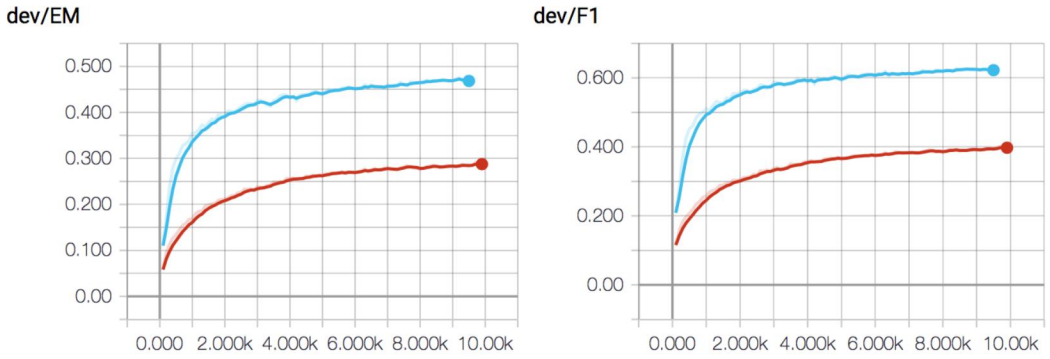

Figure 9: Baseline vs Final Model

My features seemed to pair well on top of each other. The performance of the final model suggests that the improvements all added on to each other, with little overlap – as in output final = improvement from A + improvement from B + improvement from C. I would wager that the improvement of the start and end predictions did not overlap with the improvements of the word embedding and the attention mechanism, and that most of the overlap occurred between the latter two. This is because both the word embedding and attention mechanism affect the final blended representation of the hidden and context states.

5 Conclusion

Neural nets have pushed the limits of question answering – with several leading models now surpassing human answering in F1 scoring. That being said there is still a long way to go. For my model in particular, I'd like to spend more time making the start and end positions dependent upon each other – as alluded to previously. I think I'd start with a simpler model than the Answer Pointer described in [4], to first verify that it would pair well with my current model, and then proceed to implement more advanced iterations of it.

References

[1] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. arXiv preprint arXiv:1704.00051, 2017.

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

[3] Natural Language Computing Group, Microsoft Research Asia. "R-NET: Machine: Reading Comprehension With Self-Matching Networks." Microsoft.

[4] ShuohangWangandJingJiang.Machinecomprehensionusingmatch-lstmandanswerpointer. arXiv preprint arXiv:1608.07905, 2016.