Question Answering model using BiDAF

Ran Gross
Stanford University
rangross@stanford.edu

Shawn Hu
Stanford University
shawnghu@stanford.edu

Abstract

Language understanding and information extraction is a complicated task for machines to deal with, even after the advent of the deep learning era. In this paper, we tackle a Question Answering (QA) task with the SQuAD dataset. We examine the features of a partial reimplementation of the Bi-Directional attention flow model (BiDAF) along with a few novel experiments and augmentations, including selectively downsampling and different neural net activation functions. Using this model, we were able to achieve an F1 score of .758 and EM score of .659 on the test set.

1 Introduction

Throughout recent years, Neural Networks have grown to be studied and relied on in the field of AI. As with other fields, the field of NLP has also adapted to integrate these Neural Networks into some of its more complex problems. One such problem is the Question answering (QA) challenge where we are given some sort of passage (referred to as a context) and are requested to answer a related question using a span of content from the context. It is pretty easy to see the real-world application of a system that can solve such a task. From a human perspective the task is simple, but upon further observation, it includes a lot of complex tasks for a computer, which ultimately amount to understanding the contextual meaning of each word for both the context and the question, and using an abstract understanding of the question to extract the correct section of the context.

For this task, we implemented multiple neural network models to attempt and solve this problem by mixing together components of different previously successful solutions to this problem, mainly the Bi-Directional attention flow model(BiDAF)[**?**]. One of the recent major drivers for this line of research is the release of the Stanford Question Answering Dataset (SQuAD). The SQuAD dataset attempts to solve two previously existing problems with previously existing datasets, which were either high quality, but not large enough for meaningful neural network training, or very large, but low quality or synthetic. The SQuAD dataset attempts to provide both quality and quantity by having enough data to use for training while being realistic and non-synthetic. The SQuAD dataset contains 107,785 question-answer pairs taken from 536 Wikipedia articles. The answers come in the form of a ”span” (contiguous subsection) from the relevant context that answers the question.

2 Related Work

2.1 Bi-Directional attention flow model[?]

A large portion of our experiments and implementation was inspired by the implementation of the BiDAF paper. The major relevant features of their model are:

- they apply their novel bidirectional attention flow layer to the output of the contextual embedding layer, as we do. The attention flow is called ”bidirectional” since it works with a notion of both ”context-to-query” and ”query-to-context” attention, i.e, their architecture

models which words from the question words are relevant to which context words, and their architecture models which context words are relevant to which question words;

- they use a multi-layer bi-directional RNN, which they call the modeling layer, to thoroughly encode the context-aware representation into the query and the query-aware representation of the context;
- their output layer conditions the end pointer prediction on the start one, in a way which we will detail later.

2.2 Dynamic Coattention Networks For Question Answering [5]

Another paper we reviewed was *Dynamic Coattention Networks For Question Answering*, mainly focusing on its attention layer. In short, coattention also approaches the problem of integrating mutual awareness between the query and context. It does so by generating first-order context-to-query and query-to-context attention outputs, and then using one on the other to generate a second-order attention output, and ultimately representation of the context which encodes the relationships between the two (a "(context-aware query)-aware context representation").

3 Approach

On a high level, we followed a fairly standard neural network engineering approach to the task: we iteratively implemented various augmentations to our task, examined their performance and behavior, and kept the most successful contributions, determined via repeated cross-validation.

In general, our models exhibit the same high-level architecture as either the baseline or the BiDAF model, and we tested augmentations on the individual layers in a modular fashion.

More concretely, our model maps from an input context and question to the start and end pointers using the following modules:

- A word embedding layer separately maps the context tokens and the question tokens into two sequences of pre-trained word vectors;
- A contextual embedding layer uses a recurrent model to separately transform the context sequence and question sequence into two sequences of contextually sensitive representative word vectors;
- An attention layer takes the two sequences, and through some means uses both to output a sequence of representations of the context vectors which (at the very least) encode some awareness of the structure of the question;
- A modeling layer uses more recurrent models to transform this sequence into a sequence which now incorporates contextual awareness on top of the query-awareness;
- An output layer maps the final sequence representations of our context words into start and end-pointer predictions.

A more detailed account of our approach for the individual layers is provided below and can be seen in figure 1.

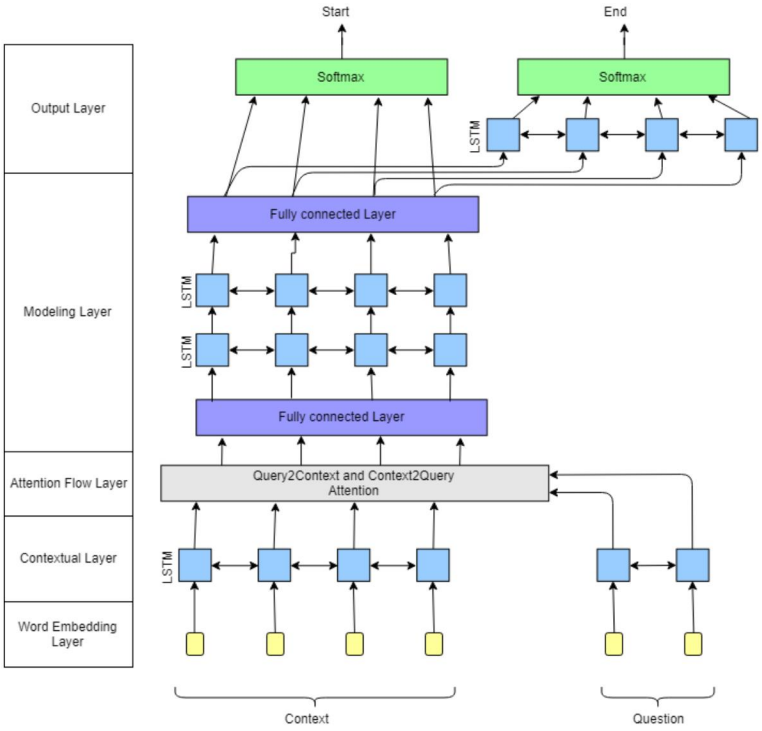

Figure 1: The Neural Network Diagram for our final model.

3.1 Word Embedding Layer

For this initial layer, we take pre-trained word vectors, namely GLoVe, as our initial word representations for the model. This layer converts each word from a discrete form into a continuous high dimensional vector.

$$\text{context embeddings } = x_0, \ldots, x_n$$
$$\text{question embeddings } = y_0, \ldots, y_n$$

3.2 Contextual Layer

Next, we pass the word embeddings into a Bi-directional LSTM, one for the context and one for the query separately. This allows us to learn information about the relation between words within the context or query.

$$c_0, \ldots, c_n = BiGRU(x_0, \ldots, x_n)$$
$$q_0, \ldots, q_n = BiGRU(y_0, \ldots, y_n)$$

3.3 Attention Flow Layer

During this layer, we calculate the bi-directional relevance of each context word on each question word and each question word on each context word. The intuition behind this is that we want to have the information flow in both direction during attention. First, we want to compute a similarity score for each context-query word pair and place it into a similarity matrix.

$$S_{ij} = w_{sim}^T [c_i, q_j, c_i \circ q_j] \epsilon R$$

Next is computing the Context to Query attention.

$$\alpha^i = softmax(S_{i,:}) \epsilon R^M$$

$$a_i = \sum_{j=1}^{M} \alpha_j^i q_j \epsilon R^{2h}$$

After, we compute the Query to Context attention.

$$m_i = max_j S_i j \epsilon R$$

$$\beta = softmax(m) \epsilon R^N$$

$$c' = \sum_{i=1}^{N} \beta_i c_i \epsilon R^{2h}$$

Finally, we combine the two computed attentions into the final bi-directional layer attention output.

$$b_i = [c_i; a_i; c_i \circ a_i; c_i \circ c']$$

3.4 Modeling Layer

The modeling layer is meant to allow the model to do some more processing before reaching the final output layer. Here our model learns contextual information between the context words given all it has learned so far regarding the query and its relation to the context in both directions. We do this by passing the Attention layer output into a fully connected layer in order to first down sample the dimensionality of each representation. We then pass that into two layers of bi-directional LSTMs. We then pass it through another fully connected layer. Using the fully connected layers allows us to shrink the model size with no loss in learning power, discussed more in detail in section 4.4.

$$d = ReLu(Wb + b_{bias})$$

$$e_0, \ldots, e_n = BiGRU(d_0, \ldots, d_n)$$

$$f_0, \ldots, f_n = BiGRU(e_0, \ldots, e_n)$$

$$g = ReLu(Wf + b_{bias})$$

3.5 Output Layer

For the output layer, we make a prediction on the start of the span, and use the result of that prediction as a way to learn where the end of the span should be. We do this by using the output of the modeling layer as the hidden states that represent the start pointer. We take that through a linear layer and pass it into a softmax to get the start pointer distribution.

$$start_{dist} = softmax(Wg + b_{bias})$$

The hidden layers of the start pointer, aka the output of the modeling layer, are then given as input to another LSTM whose hidden states are passed through a linear layer into a softmax to gain the end pointer distribution. We then take the maximum join probability of the start and end pointer conditioned on that the start pointer comes before the end pointer.

$$h_0, \ldots, h_n = BiGRU(g_0, \ldots, g_n)$$

$$end_{dist} = softmax(Wh + b_{bias})$$

3.6 Project-External Considerations

Some of the components we decided to not implement include the following:

- Larger and better-trained word vectors;
- Character-level CNNs;
- Manually-engineered features;
- Model ensembling, in which we take the average or max entry-wise over the predicted probability distributions of many separately-trained models.

Many of these would most likely give some small improvement to the F1 score, but we felt that they were less important than some of the other components or not as generalizable we did decide to experiment with. Moreover, since we our main educational focus was on how we could improve the architecture, changing the input vectors or ensembling multiple models was of lower priority.

4 Experiments

4.1 Analysis of the Performance of the Model with Respect to the Dataset

One of the very first things we did was to take the advice of the handout and generate a histogram of the training data lengths. We found that over 95 percent of the training contexts were of length less than 250, so we capped our context training length to 250. This allowed us to avoid spending memory and time computing masked parameters and zero-gradients, and allowed us to increase our batch size by a factor of two (notably, even with the decreased batch size, batches with longer context lengths took longer to compute because of the inherently sequential nature of our recurrent components). We found that removing the longest five percent of our training data had little to no effect on performance, both in terms of the shape of the training curve and in terms of the dev-set performance of the resulting model.

Moreover, we found that at for some of our first models, training on even a dataset of the first 10,000 examples resulted in very similar training and dev performance curves, while running much faster. (More specifically, we found that the training performance was about the same on a per-iteration basis, which suggests that it's possible that the higher context-length dataset, which used a smaller batch size, was slightly more data-efficient at the cost of per-sample training time.) As a result of these early observations, we trained and cross-validated our models on the truncated data set to improve our model iteration time. (We later ensured that the difference between performances was small, even for our later, more expressive models, of dev performance up to .7 F1.)

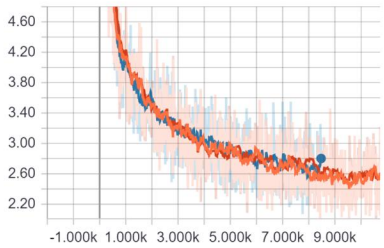

Figure 2: Training curves of a model with .7 F1 with context lengths of 250 and 600, and with only 10,000 of the training samples. For 8300 training iterations, the 10,000 training samples took 4 hours, the length-250 contexts took 5, and the full-length contexts took 6.25.

4.2 Attention Mechanisms

We tried both the BiDAF paper's form of attention and coattention. Both are explained quite thoroughly between the project handout and the Related Works section, but a succinct motivation is that

both are ways of handling the important task of modeling ways for the context to pertain to the question and vice versa.

We observed that BiDAF's attention did substantially better than coattention via simple cross-validation, so admittedly, not much deeper analysis was required.

4.3 Output Layer Behavior

We tested three different changes to the output layer:

- As a baseline for other changes, we simply modeled the start and end pointer distributions separately, as in the starter code, but required that the end pointer come after the start pointer at test time;
- We tested an experimental implementation inspired by the Match-LSTM paper, with the main motivation of conditioning the end pointer on the start one. Our module predicts the start pointer distribution using a simple softmax layer, then passes this distribution through a fully-connected layer, inputs the result and the context representations into another layer, and then uses the results to output the end pointer distribution.
- We reimplemented the output layer from BiDAF, which conditions the end pointer on the start pointer as follows: first, it predicts a distribtuion for the start pointer by taking the softmax of a linear transformation of context representations. Then, it uses both this distribution and the same set of context representations as input to another RNN, whose outputs are used to output an end pointer distribution.

4.4 Multi-Layer RNNs and Downsampling

We directly implemented the "modeling layer" module of the BiDAF paper (this was really easy, as computationally it's basically the exact same thing as two copies of our contextual embedding layer) to great effect. However, with the default hidden vector size of 100, after the BiDAF attention layer (which outputs vectors of size 1000), adding these two layers increased the parameter count of our model four-fold, forcing us to shrink our batch size. One of our contributions is to add a fully-connected layer per-context vector which downsamples the input and output of this layer. By downsampling the inputs, we were able to reduce the growth of parameter size to about 20 percent, with no observable decrease in performance.

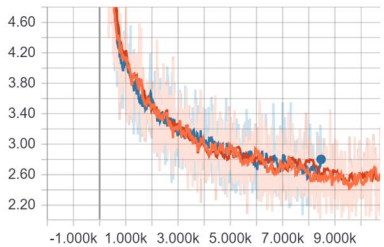

Figure 3: Dev performance during training of a model with and without the added FC layers. The added FC layers run slightly faster on a per-iteration basis.

4.5 Neural Network Activations

Tensorboard histograms and "distributions" suggest that for our experiments that used ReLU activations, by the end of 7,000 iterations, around *seventy percent* of the neurons after our "modeling layer" were dead. This motivates the use of an activation function designed not to saturate or die easily. In particular, we tested the effects of the ELU [1] and [2] activation functions. Both of these are described in more detail in the Supplementary Notes section.

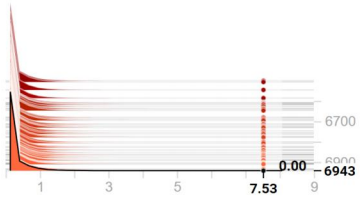

Figure 4: Histogram of our activations before the output layer.

5 Results

Our model was able to achieve an F1 of 0.749 and EM of 0.648 on the three answer dev set, as well as an F1 of 0.7 on the one-answer dev set, and an F1 of 75.75 and an EM of 65.9 on the test set. Moreover, many, many variations of our model, including modules with a simple output layer, models with different FC layer output sizes, and other models with smaller hidden layers or SELU/ELU activations both in the fully connected layers and recurrent cells were able to achieve F1's of 0.72 - 0.735, as well as one-answer dev F1 scores of 0.68-0.69.

Augmentation	F1 Score (one example)
Baseline	0.4
Bidirectional Attention	0.45
Modeling Layer	0.65
Basic Conditional Output Layer	0.67
BiDAF Output Layer	0.68
Activation and Hyperparameter Tuning	0.7
F1 (three examples)	0.75

5.1 Hyperparameters of the Final Model

Parameter	Setting
Contextual Hidden Layer Size	400
Modeling Hidden Layer Size	200
Learning Rate	$0.001 \rightarrow 0.00025$
Training Context Length	250
First Downsampling Layer Output Size	100
Second Downsampling Layer Output Size	100
Activation Function	ReLU ☺
Dropout Rate	$0.2 \rightarrow 0.32$

5.2 Other Notable Remarks

- We noted that ELU and SeLU seemed to result in higher F1 by about .01 frequently; it just so happens our best model was one that didn't have this change, possibly due to variance in random initialization.
- The SELU activation models were able to overfit to a smaller dataset faster and better than any other model by a wide margin, suggesting that they have serious merits that we were unable to fully exploit.
- Across all of our models, there was a substantial gap between training set performance and dev set performance- frequently as high as .2 F1, even on the full dataset, and even after adjustments to the dropout rate. It's therefore unclear whether a more or less complex

model is desired for the task, but likely other forms of regularization would be helpful, or a more appropriate architecture or possibly a different distribution of input data would help to bridge the gap.

- Increasing the size of the output of the FC layers from 100 didn't have a substantial effect on performance, though this may be due to other limiting features of the model.

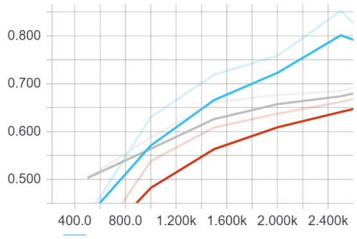

Figure 5: SELU version of the model overfits to .85 F1 on 10k samples in only 2,400 iterations.

5.3 Error Analysis

Some notable weak points of our final model include:

- Our model frequently seems to construct its answer directly in response to prolific entries. For example, in the entry:

  ```
  QUESTION: what position did rivera play in super bowl xx ?
  TRUE ANSWER: linebacker
  PREDICTED ANSWER: chicago bears
  ```

  the answer is no longer even a "position". Issues like this can arise because our model has a sort of naive level of contextual awareness by the time it reaches the attention layer, so that the activations which relate "super bowl xx" to "chicago bears" can be very high before an understanding that the question is about "what position" is reached. Of course, issues like this can also arise due to biases in the dataset (as there likely wasn't too much information about the Super Bowl). For similar reasons, the model performs especially well on "who", "when", and "how many" questions, but can fail when more than one viable answer (i.e, more than one numerical quantity or human name) appears.

- On a related note, in the absence of highly prolific entries, or in the presence of multiple, the model seems to discard lots of relevant context information:

  ```
  QUESTION: before the formation of which planet , did sol lose oxygen 16 ?
  TRUE ANSWER: earth
  PREDICTED ANSWER: moon , mars , and meteorites
  ```

  This might be addressed by having more recurrent, context-encoding structures surrounding the major focuses of attention, or by letting the model have the freedom to make predictions based on context-encoding modules separately from the attention outputs.

- Like human children, the model also isn't very good at providing answers to questions that are only alluded to, and not very explicitly exhibited, and it struggles with incorporating long introductory clauses (i.e, it has difficulty modeling very-long term dependencies). For example, for the question

  ```
  what president is credited with the original notion of putting
  americans in space ?
  ```

  , we are supposed to surmise that the question can be answered by noticing the "first" in the phrase

  ```
  first conceived during dwight d. eisenhower's...
  (15 words before the next comma),
  ```

Notably, the model also answers incorrectly here because another president name is present in the passage, so its attention cannot overcome the error. This might be addressed by incorporating information with something which models sentence syntax, or otherwise directly handles syntactic structures.

- The model very frequently predicts the correct start pointer, but will predict a seemingly nonsensical end pointer, sometimes including over 15 unrelated words. This might also be mitigated with more context-encoding structures surrounding proposed output start pointers.

6 Conclusion

In this project, we implemented a question-answering model based on the BiDAF paper architecture which achieved reasonably strong results on the SQuAD dataset. We exhibit a few features of the cross-validation behavior of basic models on the dataset. We were also able to introduce some minor novel augmentations to the model, and we examined the effects of a few different modules and parameters, including output layers and activation functions. Some of the flaws of our model are relatively easy to operationally interpret, motivating further exploration.

References

[1] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1606.05250, 2016.

[2] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self normalizing neural networks. *CoRR*, abs/1606.05250, 2017.

[3] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016.

[4] S. Wang and J. Jiang. Machine comprehension using match-lstm and answer pointer. *CoRR*, abs/1608.07905, 2016.

[5] C. Xiong, V. Zhong, and R. Socher. Dynamic coattention networks for question answering. *CoRR*, abs/1611.01604, 2016.

7 Supplementary Notes

7.1 ELU and SeLU Activations

The ELU activation (applied elementwise) is defined as

$$ELU(x) \begin{cases} \alpha(exp(x)) - 1 & x \leq 0 \\ x & x > 0 \end{cases}$$

, where α is a parameter that controls the degree of saturation of the negative activations.

A short motivation of the ELU function is that

- It has at least nonzero gradient everywhere, so neurons cannot completely die, and its gradient is not highly saturated except at very negative values;
- ELU activations can be negative, which has the net effect of making the mean activation across a layer closer to zero, which can lead to a better approximation of the natural gradient.

In a similar vein, the SELU activation is defined as

$$SELU(x)\lambda * \begin{cases} \alpha(exp(x)) - \alpha & x \leq 0 \\ x & x > 0 \end{cases}$$

where α = 1.6732, λ = 1.0507.

The SELU function purportedly fixes activation distributions on layers in a deep neural network near a mean-zero, variance-one normal distribution (given the weight initialization distribution meets

some conditions), which improves the rate and performance of training in a similar fashion to batch-normalization. The paper famously has a 98-page appendix explaining the mathematical justification, so unfortunately it is difficult to give a short explanation as to how this works.

We tested replacing ReLU with both activations, in both just the fully-connected downsampling layers, and in the FC layers + the activations.

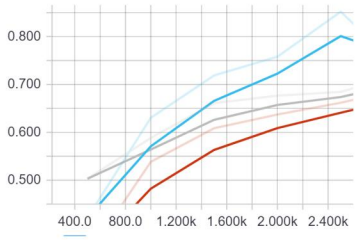

Figure 6: SELU version of the model overfits to .85 F1 on 10k samples
in only 2,400 iterations.

7.2 Other notes on cross-Validation: tuning subject to time and resource constraints; overfitting

In addition to the above, we tried to tune basic parameters including learning rate, hidden vector size, downsampling layer size, and dropout. It was not possible to find the optimal hyperparameter configuration for each augmentation we added to our model, so the figures for the improvements they provided can only be approximate. In particular, a much larger train than dev performance indicates some degree of overfitting, and our general approach to this problem was to increase dropout until the difference between the values became small- in general, we gradually decreased the learning rate and increased the dropout by small amounts as the dev F1 curve began to flatten out in order to obtain the highest possible dev F1, but notably, this was not possible for all except our final models. In the general case, there were times at which we were unable to pursue training multiple models with our resources, and we chose the models with higher dev performance on the default parameters for these decisions, even when the other model had significantly higher train performance, suggesting that it had more potential. Subsequently, we may have slightly underestimated the potential of vanilla BiDAF (without the downsampling layers) and alternative activation functions.