Rolling Deep with the SQuAD: Question Answering

Dhruv Amin
dhruv92
dhruvamin@stanford.edu

Tanuj Thapliyal
tanuj
tanuj@stanford.edu

Reid Westwood
rwestwoo
rwestwood@stanford.edu

cs224n-dtr

Abstract

The problem of question-answering is critical in order to improve data retrieval for text sources. Our group sought to understand and implement current effective models, and test our own modifications. While we were not able to improve on test results with our model, the exercise was effective in gaining some intuition for building deep learning models.

1 Introduction

Question answering systems work by finding answers to questions as substrings within context paragraphs. Recently, multi-layered deep neural networks have been able to converge as strong predictive models to answer such questions. We explored a variety of question answering models, and obtained a best model through an iterative process of model optimizations.

1.1 Background and Related Work

Stanford Question Answering Dataset (SQuAD) is a ”reading comprehension dataset”. It is composed of passages of text, a question about that text, and an answer directly from the text. It allows groups to test their question-answering systems against other submissions as well as a Human Performance benchmark of 82.3 F1, 91.221 EM scores. Our task is to build our own question-answering system using the SQuAD dataset for training and testing.

Our first motivating design example was outlined by the instructors of CS224n as a ’naive baseline’. It is outlined as follows:

1. Run a BiLSTM over the question, concatenate the two end hidden vectors and call that the question representation.
2. Run a BiLSTM over the context paragraph, conditioned on the question representation.
3. Calculate an attention vector over the context paragraph representation based on the question representation.
4. Compute a new vector for each context paragraph position that multiplies context-paragraph representation with the attention vector.
5. Run a final LSTM that does a 2-class classification of these vectors as O or ANSWER.

As we later discuss, we did not find success with the provided GRU attention cell or LSTM decoder. However, we did use this encoder.

We felt that a critical component of our system would be the attention mechanism (Step 3). We decided to follow the mechanism described in ”Bidirectional Attention Flow for Machine Comprehension” [1]. We tried only minor modifications to their design.

2 Approach

2.1 Data Preprocessing

We began the project by analyzing the SQuAD dataset in an effort to understand some basic characteristics. For example, we first plotted histograms of the question lengths and context lengths to get an idea of the distribution. We noted that for each, but especially the context, there were some outliers that were extremely long. This was useful later on - when we ran into GPU memory capacity issues, removing outliers in length relieved significant memory strain while not significantly impacting performance.

2.2 Architecture

At a high level, our system takes in an ordered list of question words and context words, represented by their indices in a vocabulary list. We were provided GloVe word representations for this vocabulary. After fetching the appropriate embeddings, our system was designed to learn how to analyze the ’meaning’ of the context and question. Based on the relationship between the context words and the question, the system gives two probability distributions. These represent the probability that a given context word is the starting word or ending word of the answer, respectively. This representation of the answer is referred to as the ’span’ - extracting all words between the two indices (inclusively) give the text answer to the given question.

The standard characterization of such a question-answering neural network consists of three primary components: Encoder, Attention, and Decoder. The encoder is responsible for taking word embeddings and finding more meaningful representations. The attention mechanism is intended to identify similarity between words in the context and the question being asked. The decoder takes the updated context word representations and calculates the probability that the word is the starting and ending word in the answer.

Below, we will outline the different architectural choices we considered and/or tested. We will detail the design choices made in Section 3. For simplicity, all cells were created with a hidden state size of H. Futhermore, D denotes the word embedding size used.

2.2.1 Encoder

BiLSTM

Our input consisted of question word embeddings $w_t^Q \in \mathbb{R}^{1xD}$ for $1 \leq t \leq T$, and context word embeddings $w_i^C \in \mathbb{R}^{1xD}$ for $1 \leq i \leq N$. As suggested in the problem outline, one standard encoder uses bidirectional RNNs with LSTM cells. First, each word embedding in the question vector is fed through the bidirectional RNN. The purpose of this is to capture relationships between the words within the question. The output state of the forward and backward paths are concatenated to form $h_Q \in \mathbb{R}^{1x2H}$, our question representation vector. We also have available the hidden states corresponding to each word in the forward and backward passes. These are also concatenated to form the question word representations, $h_t^Q \in \mathbb{R}^{1xH}$.

With a representation for the word, we now want to find representations for the words our context paragraph. Because we will be analyzing each context word specifically with the question in mind, we will condition our context embeddings with the question representation. A standard method for doing this is concatenation. Therefore, each input into our context BiLSTM can be written as

$[w_i^C; h_Q] \in \mathbb{R}^{1xD+2H}$. The output of the forward and backward passes are also concatenated to form conditioned context word representations, $h_i^C \in \mathbb{R}^{1xH}$.

2.2.2 Attention

Although we did use conditioning of the context words on the question representation, the Attention mechanism is used to more directly find the relevance of context words to the question. In general, it will come up with a score, usually a scalar, for each word. This can then be used to weight each word representation before decoding the results.

GRU
The first attention mechanism was a modified GRU cell. The pseudocode was provided by instructors, but our implementation did not seem to work. The concept was to run the input context representations h_i^C through a GRU cell before taking the cosine similarity with the final question representation h_Q.

Simple Dot
Because we suspected an implementation error with the GRU, our next approach was a very basic one. The second attention mechanism used a basic cosine-similarity scoring. We use these scores to do a scaling of our context word representations.

$$a_i = h_i^C \cdot h_Q$$
$$\hat{a} = softmax(a)$$
$$\hat{h}_i^C = \hat{a}_i h_i^C$$

Bidirectional
Our final attention mechanism was much more complex, as it followed a recent publication by Seo et. al [1]. Instead of just using the final question representation, it finds similarity between each context word and each question word, forming a matrix. However, each similarity is not a scalar but instead a full state. In the paper, a direct Hadamard product was used. We decided that this would require the question and context representations to be 'parallel'. Therefore we tried adding an intermediate transformation matrix $W^l \in \mathbb{R}^{2Hx2H}$. For context word i and question word t, we calculate $L_{it} = h_i^C \odot (h_t^Q W^l)$. This similarity is concatenated with the input representations to form a full representation $S_{it} = [h_i^C; h_t^Q; L_{it}]$. This is projected down to a scalar using a trainable vector W^s to give values a_{it}. The idea here is that we have made available a question and context word pair as well as a measure of the similarity between the pair. Our projection then has the ability to find which parts are most relevant in coming up with a scalar similarity score

Once we have this matrix, we can use two mechanisms to use the similarity measures. The first is Context-to-query, which takes the softmax of a for a given context word across the question words.

$$\hat{a}_{i:} = softmax(a_{i:})$$

It uses the result to create a representation for the context word that is a convex combination of the question word representations: $U_i = \sum_t \hat{a}_{it} h_t^Q$. The second is Query-to-Context, which finds the highest similarity across context words to each query word, takes the softmax of the resulting array, and applies the result as weights to the context representation.

$$\tilde{a}_i = max(a_{i:})$$
$$V = softmax(\tilde{a}) \odot h^C$$

Finally, we concatenate these two new representations to the input context word representations. For dimensionality purposes, we pre-multiplied this by a trainable matrix to get back to a state size of $2H$.

$$\hat{h}^C = W^s[h^C; U; V]$$

2.2.3 Decoder

BiLSTM
We first implemented a bidirectional LSTM RNN for the output of the attention mechanism. We continued to use a hidden layer size of H, so we then projected down to a single value using a trainable vector. The idea here was to again find the importance of a given word by considering the relevance of surrounding words.

Dense
We also implemented a simple dense layer. The output score $p_s = W\hat{h}_i^C$. This method no longer considers other words, and instead relies on the encoder and attention for this.

Dense with ReLU
Because our bidirectional attention mechanism had no nonlinear components, we were curious whether we could use another nonlinear layer in our system. We decided to implement a dense decoder with a ReLU hidden layer: $p_s = U \cdot ReLU(W\hat{h}_i^C + b)$ where U, W are trainable. We knew that this would be fed into the softmax function for our loss, so feeding a ReLU directly would be a bad idea. However, we were curious whether the product with a trainable matrix would work. Furthermore, by multiplying the context words by a full matrix, we do have the ability to capture some information about relatedness between context words.

2.2.4 Answering

The output of a decoder was an unscaled score vector for the context words for each the start index and the end index. For training, we use a softmax and then cross-entropy loss function. For validation, we used the argmax for our predicted start and end indices, respectively.

2.3 Methodology

Our first goal was to establish working baseline model. The inspiration for this model was provided by the instructors. It consisted of the BiLSTM encoder with conditioning[2.2.1], the GRU Attention Cell [2.2.2], and the BiLSTM decoder [2.2.3]. However, we struggled to get this model to converge. We employed several techniques in order to debug the model. First, we trained on only 10 samples in order to intentionally overfit. By observing the loss drop, we were able to establish a very basic confidence in overall code health. We then proceeded to increase the training set size until there were no signs of model convergence. Because this occurred at only 5,000-10,000 samples (out of a full corpus of about 80,000 samples), we decided that some component of our architecture was likely not working.

We decided to further simplify the model by using a simple attention mechanism [2.2.2] and simple dense decoder [2.2.3]. This model was successful in reaching a baseline performance. We then proceeded to upgrade the attention mechanism to the bidirection one described. Here, we saw a significant bump in performance.

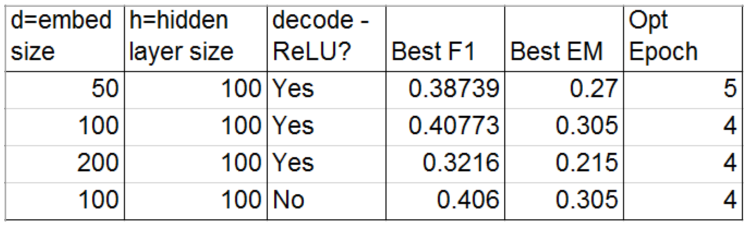
Our second goal was to test major hyperparameters and architectural choices, as outlined in Section 2.2. These hyperparameters included the hidden state size H used throughout the architecture and the word embedding size D. The focus of our architecture testing was the decoder. As discussed previously, we wanted to understand whether having a ReLU close to, but not directly feeding, the output scores would be a poor idea. The results are given in the following section.

The metrics used for this stage of testing consisted of validation loss, F1 score, and Exact-Match (EM). These values were tracked as the model trained. If the results were clearly worse, we would cut off the run. If at all promising, the run would be left to completion of 10 epochs. After running our tests, the highest performing model was kept through the remainder of the process.

Finally, we wanted to fine tune the minor hyperparameters. This included the learning rate and optimizer, gradient clipping value, dropout, and batch size. Due to time constraints, not all of these were fully tuned nor were all combinations attempted. The experiments chosen were based on trends observed between tests.

3 Experiments

As previously discussed, once we had obtained a working model, our goals were to 1) experiment with hyperparameters such as hidden layer and word vector dimensionality, and 2) test our different decoders. Below is a summary of our results.



d=embed size	h=hidden layer size	decode - ReLU?	Best F1	Best EM	Opt Epoch
50	100	Yes	0.38739	0.27	5
100	100	Yes	0.40773	0.305	4
200	100	Yes	0.3216	0.215	4
100	100	No	0.406	0.305	4

Figure 1: Summary of Results

We indicate in this table the epoch at which our test evaluation was optimal. We found it noteworthy that this occurred before we expected. To investigate this further, we plotted our learning curve with the training and test loss for a typical model. One item for further research into the model is whether this early convergence is suboptimal. For example, is this an indication of overfitting? Would more carefully choosing a rage of annealing reach a lower test error?

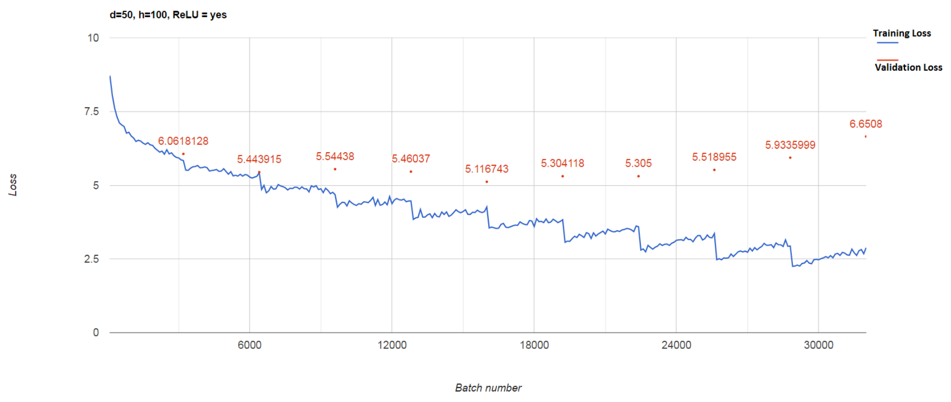


Figure 2: Learning Curve

4 Conclusion

The experience of designing an encoder-decoder system with only guidance from state-of-the-art research papers was very challenging but extremely rewarding. We quickly developed a much deeper understanding of TensorFlow and its nuances. We spent a significant amount of our time debugging an early model, requiring a complete disection of each feature in our model. By the time we fixed our bug, we were familiar enough with TensorFlow and the architecture to write out new features in very little time. In the process, we had many ideas we wanted to test but did not have sufficient time to. In this section, we will summarize our key findings, analyze our systems performance, and talk a bit about these ideas.

4.1 Findings

Our primary observation was that the more complex bidirectional attention mechanism improved the model performance. Regarding the decoder, we did not obtain distinctive results to make any conclusion about the impacts of including the ReLU near the output. We believe this is due to one of two things. First, the added complexity in the decoder may not have any value, as the model is already beginning to overfit. The second possibility is just that our model's bottleneck is elsewhere, and therefore the difference in the decoder is not visible.

4.2 Test Set

To understand how our system is working, we run inference on our development set and examine our predictions in order to gain insights. Our analysis is detailed below.

Our model performs particuarly well when identifying answers that are proper nouns, dates, or numbers with question words nearby in the context. This behavior indicates the model's attention mechanism is working as expected. Where the model struggles more is determining how to select the right noun, date or number from a cluster of such information based upon references farther in the context. In these scenarios, the model gets confused, often outputting the wrong noun, number, or date from a set of figures in close proximity. One extension we might attempt to achieve better performance in this realm is to incorporate the dependency structure of the context paragraph in our learning so as to pick up on references and dependencies that could improve selection accuracy.

Two other areas that our model needs to improve are domain knowledge and similarities. If our model encouters a domain noun (e.g. ”Unified Diagram Protocol”) and then later answer information in regards to this question is referenced by another name (e.g. ”network”), our model has no way to assess the two and understand with so few training examples and that information is lost, resulting in an empty answer prediction. Similarly, subtle synonyms between the question formulation (e.g. ”What do people say about X?”) and the context formulation (e.g. ”people believe X”) can lead to lost answers. To improve here, we might experiment with introducing the larger 6B vocabulary introduced by the assignment handout or pretrain our word vectors on domain texts that are likely to model these similarities.

Finally, our last major set of issues fall into overelaboration mistakes. There are many times where a shorter, more direct answer from the context would suffice but our model prefers a longer winded one. At times, these choices are similar in meaning to the true answer; often though, they are more a reflection of the system trying to match question and answer similarity (e.g. Q: ”Who has limited productive potential when faced with less access to education?” Pred: ”those who are unable to afford an education , or choose not to pursue optional education” when a more simple ans is ”poor”) We should add a component to our model that better learns these question context similarity emphasis weights, and perhaps penalizes longer answers.

4.3 Future Ideas

We had explored and developed many possible model features, including modifications to the encoder and decoder. However, we did not have time to fully explore the inter-relatedness of the components. We hypothesize that a simple test of changing one component while holding all else equal is not sufficient in making good design decisions. Each component has a performance-to-complexity trade-off. In other words, a component may add a significant number of parameters while improving, but not significantly, out-of-sample predictions. It may be more efficient to remove this marginal improvement for another component, while using both leads to a surplus of parameters and thus over-fitting. In summary, while we were able to test the inclusion of each individual component, we were not able to test all combinations of components which could have significant performance improvements.

We also still believe that a more intelligent decoder would help. One idea was to use the prediction vector for the start index to inform the ending index and vice versa. This could be done by adding another layer, or an LSTM with conditioning.

Finally, we had wanted to experiment further with dropout and regularization.

5 References

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