Deep Question Answering using Bi-directional Attention based LSTMs and Weighted Distribution Penalization

Akhil Prakash
Department of Statistics

Pranav Sriram
Department of Computer Science

Vineet Ahluwalia
Institute for Computational and Mathematical Engineering

Abstract

Question Answering and Reading Comprehension lie at the core of Natural Language Processing. We build a question answering model on the SQUAD dataset of 100,000+ questions and answers for training and building machine comprehension. Our model uses bidirectional LSTMs, attention modeling, and a novel loss function based on weighting an output probability distribution over indices by a weight function capturing distances between indices and correct labels.

1 Introduction

The recent resurgence of deep learning research has led to a rapid growth in novel models and algorithms for Natural Language Processing (NLP) tasks. At the heart of these methods lie neural network architectures that seek to map complex natural language constructs, such as words, sentences, and paragraphs, into semantically meaningful latent embedding spaces. The key building blocks of such architectures are Recurrent Neural Networks (RNNs) and their variants (LSTMs, BiLSTMs, and GRUs, to name a few).

NLP has a breadth of applications and well defined research problem areas. However, the core of NLP, influenced in part by the famous Turing Test, aims at addressing one very easily stated but difficult problem: question answering. In other words, we seek models capable of formulating well structured answers to questions, taking into account context, grammatical structure, and knowledge pertaining to the question. In the past, both communities of computational linguists and computer scientists have tackled the problem with theory from various domains including information extraction and retrieval, semantic parsing, and synthetically engineered grammar [2]. The methods from the earlier days of NLP have been outpaced by advances in neural networks, which are more scalable, modular, and robust.

In addition to breakthroughs in neural deep learning, IBMs development of Watson represented another significant catalyst to research in NLP. Open Domain question answering gained a lot of popularity after IBM Watsons successful attempt at beating humans in Jeopardy! In question answering type NLP tasks, domain specific models may be more successful. An analogy would be that of a person specializing in a specific skill versus learning multiple skills. Question Answering tasks are also context and question dependent. In an industry setting, a robust model would be able to handle errors in the type of questions too.

1.1 Problem Statement

Our problem mainly aims at building a novel neural network architecture for Reading Comprehension. The task has been simplified such that it lies at the intersection of closed domain and open domain of QA problems but due to expansiveness of the context paragraph corpus which is Wikipedia it enters the realm of open domain problems (3). We formulate the problem mathematically as follows: we are given a set of triples of ¡paragraph:p, question:q, answer:a¿. The answer a contains a start index s and an end index e, indicating that the correct answer to the q is the set of words in p beginning at index s and ending at index e. Our goal is to build a model capable of returning correct start and end indices given new paragraphs and questions. Note that this problem is more simplified than a general question-answer setting in that the answers are contiguous subsets of words in the paragraphs, rather than model-generated natural language constructs. We experiment by modelling the problem in multiple ways, including a classification setting (assign probabilities of being a start/end word to each word in the paragraph), a regression setting (predicting real numbers for start/end indices and using a distance-based loss), and other frameworks described in detail later on.

2 Literature Review: Background and Relevant Work

As mentioned above, recent advances in computational power and neural network based approaches to model problems like machine translation, voice recognition, image captioning, conversational modelling, or text parsing yield sophisticated architectures and variants. A very simple example comes from the specific problem of attacking long term dependencies or sentences with long sequences of words. Recurrent Neural Networks are very potent at being able to learn from the past information while trying to predict the next word in the sequence based on the previous words. They struggle with long term dependencies in practice [4]. This has given rise to variants of RNN specifically LSTMs which resolve the problem of remembering to information retention over different states by using gates. A more advanced feature augmentation to LSTMs comes about with attention modelling and using convolutional networks. We discuss in the following research papers in Question/Answering and Reading Comprehension that inspired our approach towards modelling our unique architecture and framework.

In context of the Question Answering NLP task, a recent paper by Yin et al[5] introduces a technique called Hierarchical Attention based convolutional network (HABCNN) which uses two approaches to solving this task. One approach is based on attention modelling and the other is based on textual entailment. The first approach basically uses attention mechanisms to generate projections of the question and the answer. The comparison of the projected representations yields the contextual similarity of the question to the answer. The attention mechanism models such that the key snippets, phrases, and sentences are emphasized in the representations. The novelty of the HABCNN model lies in the following three areas:

- **Granularity:** Hierarchical CNN models the document representation from word to sentence, and then from sentence to a level of precise summarized version.
- **Attention Mechanism for dynamic context representation:** The model generates a question specific representation of the context using Attention for enhanced Hierarchical CNN learning.
- **Combination of representations of sentence and snippets:** The corresponding projection/representations carry more information for the question instead of just one.

The above work inspires our thought process towards modeling in the following areas:

- Attention
- Neural Network Module
- Combination of question and context

Most of the deep learning approaches towards question answering frameworks have an underlying assumption that questions and answers have very similar representations [7]. In this paper, the authors approach the problem of question answering as an answer sentence binary classification

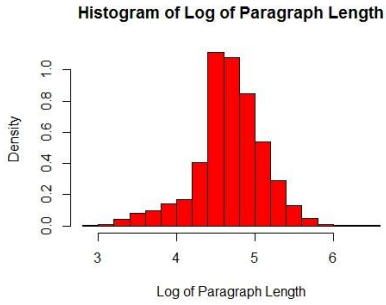

Figure 1

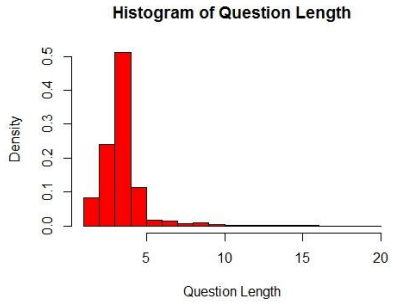

Figure 2

problem. Assuming a list of answer sentences associated with a question and the answer (judgements), they build a classifier over the triplet of ¡question,answer,judgement¿ to predict judgement of other unobserved Question Answer pairs. This work could inspire our model in the following ways: Developing above mentioned triplets with each question and context pair paragraph given to us, and in turn augmenting our dataset. Also, a similar but reverse approach could be to generate a question from an answer representation and trying to equate the similarity of generated question with the given question.

3 Dataset

Most of the advances in Natural Language Processing specific to Reading Comprehension recently have inclined towards supervised learning with the availability of more labeled data. We use the SQuAD data [1] set which is a reading comprehension dataset built using crowdsourcing on a Wikipedia data set. Some of the most important characteristics of the dataset are as follows: Contains about 107,785 question answer pairs with context. Larger than any previously available manual labeled data sets eg: MCTest Most of the neural network models still have not been able to achieve the human performance on this data set. Dataset: https://rajpurkar.github.io/SQuAD-explorer/ Given the above characteristics, SQuAD dataset provides an exciting modeling and machine learning challenge in machine comprehension. It is important to note that all the answers to the questions exist directly in the paragraph. Thus, when answering the questions we should not be generating words from a bag of words. The SQuAD dataset is unique in that the answers are not multiple choice like several other hand-crafted question and answer datasets.

3.1 Preliminary data analysis

Before we start running any models it is important for us to be able to visualize the data so we know what values for hyperparameters that we should try and what models we should try out.The first thing we do is plot the distribution of paragraph lengths. This is highly important for the RNN parts of our model because we know that LSTM cells can only remember the pasts approximately 80 words. Thus, the longer the paragraph the harder the questions will be to answer.

The paragraph lengths are skewed to the right even after taking the logarithm. This makes us believe that a max paragraph length of 200 should be sufficient. Doing so makes us only lose about 5% of the data.

Min	Q1	Median	Mean	Q3	Max
20	86.75	107	116	139	653

Figure 3: Summary Statistics for the question length

Next, we look at the length of the questions. The distribution of question lengths has a long right tail (Figure 2). Most questions are pretty short. Thus, our max paragraph length can be pretty short around 10.

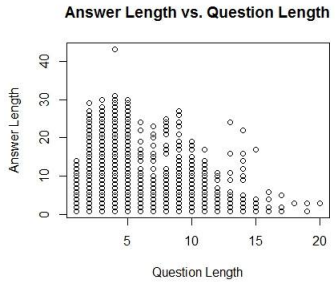

Figure 4: Answer Length vs Question Length. We see that longer questions tend to have shorter answers.

We observed that there were about 4 to 5 questions per paragraph. Thus, we know that we do not have to deal with the issue of one paragraph being super long and difficult to comprehend and then there being several question associated with it. If this were to happen then we may get a lower F1 score than what is expected. We note that most answers are fairly short. For example, 75% of them are less than 3 words. Roughly 25% of the answers are 1 word. Thus, if the model is even off by one word, the F1 score will be 0. From Figure 3, we note that longer questions have shorter answers. The other interpretation of the graph is that there could just be some sparsity for the longer question and thus just by random chance they had short answers.

The most common question words in order of frequency from highest to lowest are What, Who, How, When, . In comparison it is easier to answer the When questions because dates and years come in very specific formats. The Who questions can be answered by looking for noun phrases or named entities.

4 Approach:

4.1 Initial Experiments and Ideas

To get a feel for the data, we built an N-gram model that can answer questions with a quantitative answer, particularly questions with keywords such as when or how much. We did so simply by matching n grams of the question to n grams in the paragraph and then picking the closest number in the paragraph to that n gram. This worked fairly well but we could not generalize this to other questions where we could easily subset the possible answers just by the form of the words. We have taken several approaches to solving this problem. We have both viewed it as a regression problem and a classification problem. For the classification, we output a probability distribution for each index being the start index and a separate probability distribution for each index being the end index. We also tried various loss functions. Initially, we used an ordinary softmax cross entropy function. Our intuition was that cross entropy does not capture differences between the model being far or close to the correct start and end indices. For instance, if the correct start index is 4 and the model places most of the probability mass on index 5, cross entropy would not distinguish that case from a case where the model places most of the probability mass at 100. Clearly, we would rather have a model that thinks the answer starts at index 5 when the correct answer starts at 4 than a model that thinks the answer starts at index 100. To ensure that we reward our model for at least getting close to the right answer, we experimented with other loss functions. One approach was regression. The model, instead of outputting a probability distribution, would output a real number indicating the position of the start index, and another real number for the end position. We first used a Euclidean loss for this model, but then switched to using a power less than 1 to discourage the model for simply learning the mean of the distribution and shooting for the middle (this is explained more in the challenges section). Another approach was using the classification framework but smudging the

true distribution. When one ordinarily computes softmax cross entropy loss, the true distribution for a particular data point is a one-hot vector, that is, the probability mass is concentrated at one particular index (the correct index). We convert this one-hot vector to a smudged distribution that places mass both on the correct index and on indices close to the correct index (that is, within a fixed window size). The model then gets rewarded for placing probability mass on the correct label as well as nearby. We tried using the second model recommended by the course staff in a Piazza post, which we call baseline 2. However, this did not end up working very well. We note that for several models, the model learns to predict one word answers even for questions that do not have one word answers. Thus, we introduce a size penalty where we penalize for the size of the predicted answer being different than the size of the correct answer. This helped the model start to predict two and three word answers. Our final model is inspired by baseline 2, and described in the next section.

4.2 Approach: The Final Model

Here we present a description and visual representation of our final architecture.

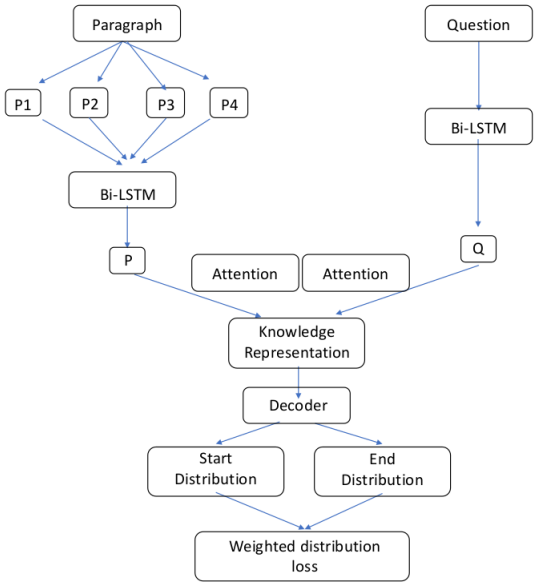

Figure 5: Model Architecture

$$
\begin{aligned}
P &= LSTM(paragraph) \\
Q &= LSTM(paragraph) \\
A &= softmax(PQ^T) \\
C_P &= AQ \\
P_{concat} &= concat(C_P, P) \\
K &= reshape((P_{concat}W + b), 2dReshape) \\
start\_dist, end\_dist &= \\
normalize(affine(lrelu(affine(K))))
\end{aligned}
$$

Figure 6: Model Equations

Encoder and Decoder

First, we chunk the paragraph into 4 pieces. We run each of the four pieces of the paragraph through a Bidirectional LSTM. (The chunking helps avoid vanishing gradients from long paragraphs.) We

combine the four outputs to form P, the paragraph encoding. We use a separate bidirectional LSTM to encode the question as Q.

We then use attention to create a knowledge representation K that combines information about the paragraph and the question. The attention is calculated by taking the outer product of P and Q. This means that we are taking all combinations of words in the paragraph with all the words in the question. Thus, we can figure out which words in the paragraph are similar to which words in the question. K is passed through the decoder, which is a neural network with affine functions, leaky relu, and a normalization output layer that outputs two distributions: start_dist and end_dist.

Below is a summary of equations governing the encoder and decoder: ¡INSERT EQUATIONS HERE (from the sheet I wrote them on)¿

Loss and Predictions
The loss functions for start_dist and end_dist are analogous, so here we explain the loss for start_dist. Consider a particular ¡paragraph, question, answer¿ triplet. Suppose the true start index is s_t . The model outputs a vector of length L, whose ith position represents the probability that index i is the start index. For each index i, we compute a function f(i, s_t) that represents the distance of i to the true start index. We choose f to be f(i, s_t) = abs(i - s_t) to the power of z, where z was a hyperparameter that we varied from 0.3 to 0.8. (Note that f is not a true metric so we put distance in quotes.) The model is then penalized for placing probability mass at i proportional to the distance f(i, s_t). Thus the loss function is the sum of p_i times f(i, s_t) over each index i, which is just the dot product of the probability distribution vector and the vector f. We call this the **weighted distribution loss**, since the loss is based on the distribution and weighted based on proximity to the correct label. If the model places all the probability mass at the correct index, the loss is 0; if the model places mass far from the correct index, the loss is high. The loss for the end distribution is analogous, and we sum the start and end loss to obtain the full distribution_loss. (We later add various regularizers, including a penalty for getting the answer length wrong, but these are not central to the model.)

Make sure the figure caption does not get separated from the figure. Leave sufficient space to avoid splitting the figure and figure caption.

The final prediction for the start index based on the distribution is sigma(pi * i). In other words, it is the expected index, or a weighted mean of indices at which probability mass is placed. We do the same for the end index. We prefer calculating the expected value versus just finding the mode of the distribution because this function is easily differentiable. Also if we just found the mode and there were two peaks, then small perturbations could make us flip flop between the peaks.

5 Results

Following we compare different values of encoding length (the size of the hidden states for the lstm for the paragraph and the question). We see that changing the encoding length does not change the F1 score that much. It could just be that we need to train the model for more time to see a bigger difference.

The graph below shows how the f1 score changes as a function of time. We see that the model is learning and there is some randomness in the F1 score.

Also, we show the graph of the loss as a function of time. We see that the model is learning as the loss function is steadily decreasing. It looks like the loss is plateauing and reaching a global optima but if we had more time we would want to train the model for longer. Most of the models had a around an F1 score of 5% after a couple hours of training. The EM score was about 3%. It seems that the model mainly learns the length of answers as that is the easiest. But doing so, causes the loss function to reach a local optima that it cannot get out of. To combat this we tried several initializations like Xavier, random normal, and random uniform. We thought that the numbers from the Xavier initialization were too small and had too little variance that they did not break the symmetry of the problem. Occasionally, the model would also get stuck and predict the same start indices and end indices for every paragraph. Thus, we wanted more variance in the initializers and tried random normal and random uniform. However, these did not help that much. There is probably a bigger issue and these minor changes in initialization probably only made a few percentage points difference in F1 and loss function.

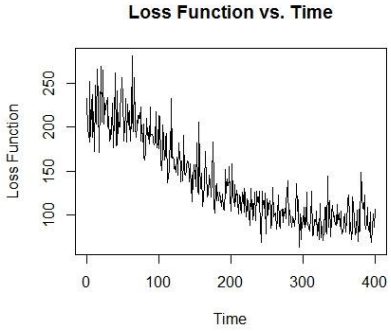

Figure 7

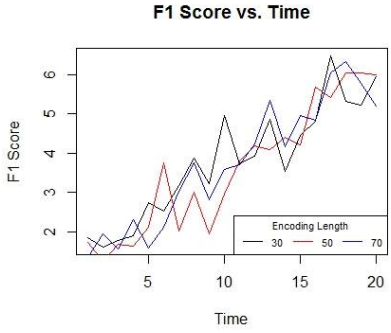

Figure 8

6 Challenges

Setting up the data pipeline was challenging, particularly standardizing data formats (npz files, numpy arrays, tensors, python lists) and ensuring consistency in numerical data types (float32 vs. float64 and int32 vs int64 because of some idiosyncrasies in the older version of tensorflow we were using). However, we were able to overcome these obstacles by diving into the provided starter code and making changes where necessary, and adding additional preprocessing code padding and keeping track of variable lengths of questions and paragraphs. The bigger issue we encountered was vanishing / saturating gradients and hyperparameter tuning. In many cases, our model would simply not learn anything during training, and the error could have been in any of a dozen places (a bug in the code, bad learning rate, bad weight initialization, or an intrinsic lack of model capability stemming from the encoder, decoder, or saturating gradients to name a few). Eventually, we realized by tracking the global grad norm over time that saturating gradients were significantly impeding learning. We hence got rid of various mathematical transformations that could lead to dying gradients, such as a softmax function for normalization and ReLu (which we replaced by L2-normalization and leaky relu respectively). We also chunked our paragraphs into four pieces to prevent saturating gradients that may arise from running long paragraphs through the BiLSTM.

Another significant issue we encountered was that our models would often get stuck in some sort of local optimum. For instance, one of our models based on regression learned to simply predict the same answer positions for every single question and paragraph, by predicting roughly the mean index positions across the dataset. The reason for this was that the loss function encouraged models at the beginning of training to avoid being very far from the right answer, so always shooting for the middle was better than getting some answers right and being far off at other times. The model would then get stuck in this mode and fail to recover. Other models often started predicting all answers to be length 1, that is, end index = start index. The model would get stuck on this and not learn to predict longer answers. Avoiding these training degeneracies was very challenging, and fixing one problem often led to another.

7 Conclusion and Future Ideas

We built a model using bidirectional LSTMs, attention, and a novel loss function that attempts to combine the benefits of classification and regression loss functions. While our model never got higher than a 10% F1 score, we believe that with further fine-tuning and engineering to avoid degeneracies during training, our approach to classification using weighted distributions could prove fruitful. One thing that we did not get to doing was creating batches such that all the paragraphs in the batch had roughly the same length and all the questions in the batch had roughly the same length. Then there would be less padding that would have to be done. This would save some computation time. For the longer paragraphs we might not have to truncate them as much. This may be helpful if there are answers that straddle the bounds of where we define the max paragraph length to be. Given that several of the words do not have GLoVe word embeddings, possibly training new word embeddings on a similar task like sentiment analysis may be helpful. Then, we do not have all these

unknown word vectors. We note that through more hyperparameter searching we may be able to get a better model.

8 Contributions

Akhil and Pranav wrote all of the code. All three authors collaborated on the paper write-up and poster.

References

[1] Rajpurkar. & Zheng et.Al *SQuAD 100,000+ Questions for Machine Comprehension of Text*

[2] Karl Moritz Hermann & Tom Koisk & Edward Grefenstette & Lasse Espeholt & Will Kay & Mustafa Suleyman & Phil Blunsom *Teaching Machines to Read and Comprehend*

[3] Terry Winograd. Proceedings of the NAACL HLT 2010 First International Workshop on Formalisms and Methodology for Learning by Reading *Understanding Natural Language*

[4] Yoshua Bengio & Patrice Simard & Paolo Frasconi (1994) *Learning Long term dependencies with gradient descent is difficult*

[5] Wenpeng Yin & Sebastian Ebert & Hinrich Schutze *Attention-Based Convolutional Neural Network for Machine Comprehension*

[6] Alvaro Morales & Varot Premtoon & Cordelia Avery & Sue Felshin & Boris Katz *Learning to Answer Questions from Wikipedia Infoboxes*

[7] Lei Yu & Karl Moritz Hermann & Phil Blunsom & Stephen Pulman (2014) *Deep Learning for Answer Sentence Selection*