Neural networks for text-based answer extraction

Brian Hicks
CodaLab: bhick9
Department of Computer Science
Stanford University
Stanford, CA 94305
bhicks2@stanford.edu

Abstract

In this paper we propose a simple neural network architecture for solving the problem of text-based answer extraction, using the SQuAD dataset to train. Although our model fails to perform as well as desired, it does better than random guess, and could potentially serve as a useful starting point should we choose to pursue further research in the field.

1 Introduction

Although natural language processing, as a sub-field of computer science, has seen widespread growth both in industry and as an area of research in academia, there are many problems that have yet to be solved with human-level or higher accuracy. One such area of research, and the focus of this paper, is that of text-based answer extraction, more generally referred to as the problem of question answering.

On the surface, the problem posed in question answering is a relatively simple one. Given a passage of text, as well as one or more questions about it, our goal is to find the sub-sequence of tokens *from within the passage* that correctly answers the question. Here, it is useful to emphasize an important facet of the problem: the goal is not simply to provide a factually correct answer, but rather to return a specific series of words from within the passage that can accurately answer the given question. It is for this reason that, throughout this paper, we will prefer the term *answer extraction* over the more general *question answering*.

2 Background

Historically, one of the most significant problems in developing answer extraction models is the lack of training data. Prior to 2016, the largest available data set of non-cloze question-and-answer data was WikiQA, with only 3,047 data points. Recently however, researchers at Stanford University have made available a much larger dataset entitled the "Stanford Question and Answer Dataset", or SQuAD, with more that 100,000 separate data points [1]. By making this dataset freely available to the public, the researchers have made it much simpler to build robust and high performing answer extraction models. We will be making use of this datatset ourselves in the development and training of our own model.

Although there are a wide variety of approaches to the problem of answer extraction (as is readily apparent in the literature), there is an important insight that is common to all of them: the importance of the relationship between the question and the answer [2, 3]. Specifically, and intuitively, for any given passage, there are many potential questions that could be asked, each of which can require or elicit a different answer. As a result, it is important, in building an answer extraction model, to ensure that we take into account not only the content of the passage, but the content of the question.

This insight is apparent in all of the previous research referenced in the development of our own system, and will play an important role in our system as well.

Over the course of our development of our own model, two particular papers have been of great interest to us, and a significant source of inspiration. Specifically, we used many of the ideas advanced by [2] and [3], especially those relating to the merging of the question embedding and the context paragraph representation. Although we often ended up designing simpler models than those advocated in these papers, they served as a useful background and reference point in developing our own architecture.

3 Approach and model

Architecturally, we favored a relatively simple approach to this problem, in contrast to some of the more complicated models attempted in previous papers on the subject. As a foundation for our model, we drew heavily from the suggestions made by course staff regarding baseline models, and subsequently augmented the model using techniques that had been used to great success in previous researchers' work.

3.1 Neural network architecture

From a high level perspective, our model consists of three primary layers: contextual processing, similarity weighting, and prediction. The first of these layers serves to encode both the question and context paragraph in such a way as to incorporate the full context of the individual tokens into their vector representations. Next, we pass these context-aware question and context representations into a similarity weighting layer, which we weight each portion of the context paragraph based on its relevance to the question at hand. Finally, we use this weighted representation of the context in a simple feed-forward network to derive a final prediction as the to the answer's start and end indices in the context paragraph. Each of these layers is discussed further in the corresponding sections below, but it is prudent to first define some notation before delving deeper into the underlying architecture.

Let the input context paragraph of length m be denoted $C = [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_m]$, with $\mathbf{c}_i \in \mathbb{R}^d$, where d is the length of the vector representation of the ith word. Similarly, we denote the question of length n as $Q = [\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n]$ with $\mathbf{q}_i \in \mathbb{R}^d$.

3.2 Contextual processing

The first of our three layers serves to take into account the fact that the question and context representations Q and C are abstract representations of the individual tokens of the overall sequences. That is to say, that each element $\mathbf{q}_i$ and $\mathbf{c}_j$ is a generic representation of the ith or jth word of the input, and only that word. As a direct consequence of this, the original form of Q and C are not suitable for our purposes; the representation of each individual token in isolation is not of interest, but rather how the word fits into the larger context of the question or paragraph.

In order to account for the contextual information available to us, we use the first layer of our neural network to create a context-aware encoding of Q and C, denoted $\mathbf{q}'$ and C'. In order to do so, we use a bidirectional LSTM as an encoder for the raw inputs. Both C and Q are processed using the same LSTM (i.e., the same parameters), as inspired by [3], but the post-processing of the outputs differ.

3.2.1 Encoding of $\mathbf{q}'$

When we feed the question encoding Q into the LSTM, we generate the following outputs at each time step t:

$$\overrightarrow{\mathbf{h}}_t^Q = \overrightarrow{LSTM}\left(\mathbf{q}_t, \overrightarrow{\mathbf{h}}_{t-1}^Q\right) \qquad \overleftarrow{\mathbf{h}}_t^Q = \overleftarrow{LSTM}\left(\mathbf{q}_t, \overleftarrow{\mathbf{h}}_{t+1}^Q\right)$$

Once we have done the above encoding process, we are left with $2n$ vectors, $\overrightarrow{\mathbf{h}}_1^Q, \ldots, \overrightarrow{\mathbf{h}}_n^Q$ and $\overleftarrow{\mathbf{h}}_1^Q, \ldots, \overleftarrow{\mathbf{h}}_n^Q$, all of which are in $\mathbb{R}^\xi$, where ξ is the size of the hidden layer in the LSTM. The

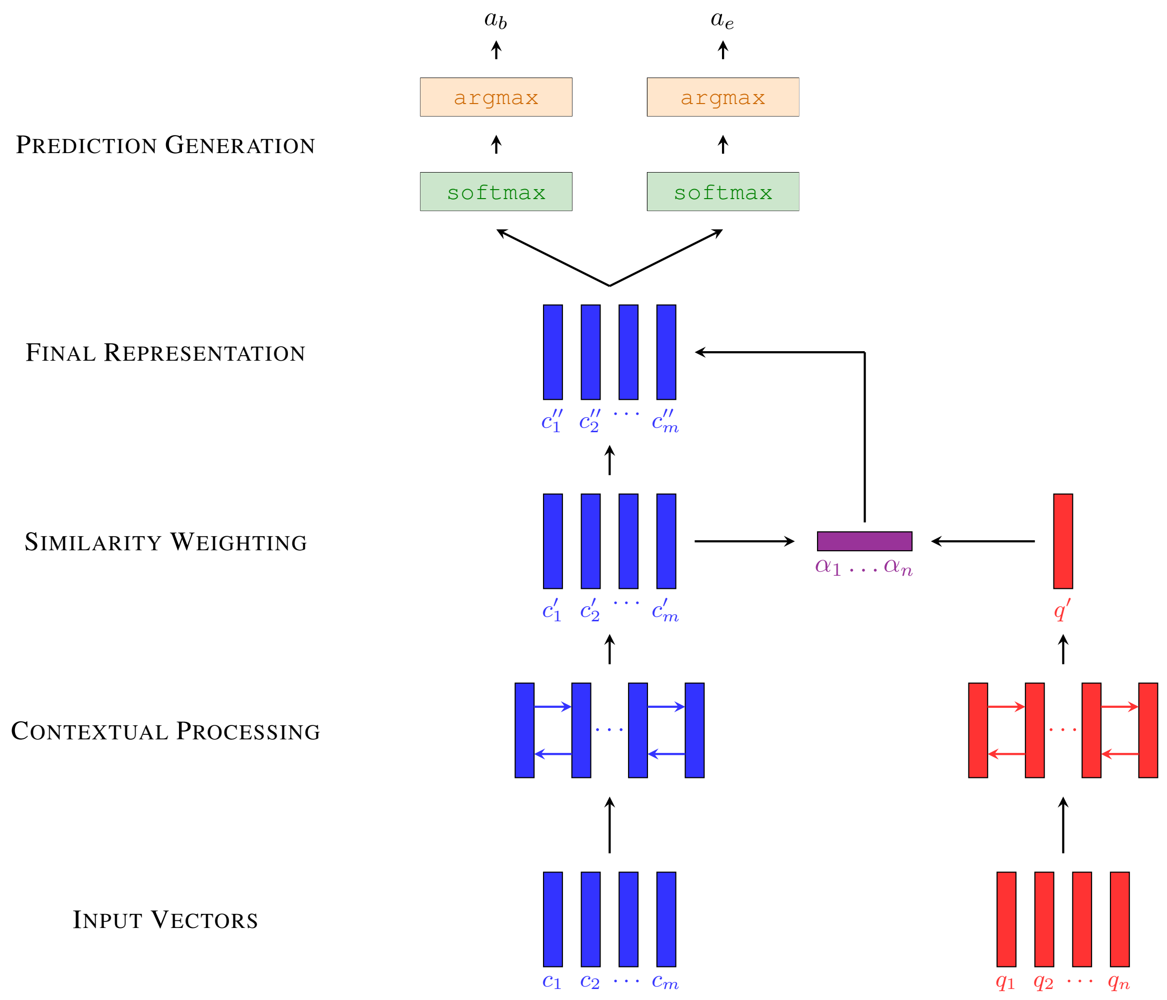

a_b
a_e
argmax
argmax
PREDICTION GENERATION
softmax
softmax
FINAL REPRESENTATION
c''_1 c''_2 $\cdots$ c''_m
SIMILARITY WEIGHTING
$\alpha_1 \ldots \alpha_n$
c'_1 c'_2 $\cdots$ c'_m
q'
CONTEXTUAL PROCESSING
INPUT VECTORS
c_1 c_2 $\cdots$ c_m
q_1 q_2 $\cdots$ q_n

Figure 1: Answer prediction network architecture

next step then is to create a single vector representation of the question, which we will use later to determine the similarity of each element of C'. To generate this single vector representation, we concatenate the final vectors from each direction of the LSTM above. Formally, we define $\mathbf{q}'$, the final, context-aware representation of the question as

$$\mathbf{q}' = \left[\overrightarrow{\mathbf{h}}^Q_n; \ \overleftarrow{\mathbf{h}}^Q_1 \right] \in \mathbb{R}^{2\xi}$$

where $[\,\cdot\,;\,\cdot\,]$ is the operator for concatenation.

By concatenating the final vectors from each direction of the LSTM encoding over the input Q, we can create a single 2ξ-dimensional vector that represents the question as a whole, taking into account contextual considerations from both directions (left-to-right and right-to-left).

3.2.2 Encoding of C'

Just as we did in the above encoding of $\mathbf{q}'$, the first step in generating the context-aware context encoding C' is to feed C into an LSTM. As mentioned above, we use the same LSTM, with identical parameters, that we used to generate $\mathbf{q}'$. By maintaining the same parameters in the model, we ensure that the manner of converting a context-unaware vector to a context-aware one is consistent across the question and answer (as referenced in [3]), which will be important when it comes time to determine the similarity scores, as we describe in the next section.

Mathematically, the outputs of our LSTM over C at time step t can be written as

$$\overrightarrow{\mathbf{h}}_t^C = \overrightarrow{LSTM}\left(\mathbf{c}_t, \overrightarrow{\mathbf{h}}_{t-1}^C\right) \qquad \overleftarrow{\mathbf{h}}_t^C = \overleftarrow{LSTM}\left(\mathbf{c}_t, \overleftarrow{\mathbf{h}}_{t+1}^C\right)$$

With these $2m$ vectors $\overrightarrow{\mathbf{h}}_1^C, \dots, \overrightarrow{\mathbf{h}}_m^C$ and $\overleftarrow{\mathbf{h}}_1^C, \dots, \overleftarrow{\mathbf{h}}_m^C$, all of which are in $\mathbb{R}^\xi$, we then define a combined vector $\mathbf{c}_i' \in \mathbb{R}^{2\xi}$ as the concatenation of $\overrightarrow{\mathbf{h}}_i^C$ and $\overleftarrow{\mathbf{h}}_i^C$. Formally, then, we write that

$$\mathbf{c}_i' = \left[\overrightarrow{\mathbf{h}}_i^C;\ \overleftarrow{\mathbf{h}}_i^C\right] \in \mathbb{R}^{2\xi}$$

Although this appears to be very similar to the definition of $\mathbf{q}'$, there are two important differences to note. First, whereas there is only a single $\mathbf{q}'$, there are m different $\mathbf{c}_i'$—one for all $i \in \{1, \dots, m\}$. Additionally, we created $\mathbf{q}'$ by concatenating the final vector produced by the LSTM in each direction, as we sought a single vector that would represent the entire question, as explained above. When it comes to the context-aware context paragraph vectors, however, we want to maintain them as distinct entities. This is due to the fact that, at the end of the neural network, we will be attempting to predict which of them is the beginning of the answer span and which is the end.For that purpose, the differences between $\mathbf{c}_i'$ and $\mathbf{c}_j'$ will be important, and thus cannot be discarded in the same way as we did in defining the context-aware question encoding.

Nonetheless, the concatenation in this stage serves the same purpose as it did in the encoding of $\mathbf{q}'$: to produce an encoding for the ith word of C that takes into account the surrounding context. This will allow us, in turn, to consider the importance of a word, with respect to the question, as a part of the whole context passage, rather than in isolation.

Once we have defined $\mathbf{c}_i'$ for all i, we then can intuitively define the entire context-aware encoding C' of the context paragraph as

$$C' = [\mathbf{c}_1', \mathbf{c}_2', \dots, \mathbf{c}_m'] \in \mathbb{R}^{2\xi \times m}$$

3.3 Similarity weighting

After we've run the context processing steps that we've defined above, we have context-aware representations of the question and context passage, denoted $\mathbf{q}'$ and C', respectively. Unfortunately, these encodings do not provide us with any information, at the moment, about which elements of C' are important, with respect to the question being asked. That is why the next layer of our model is relevant, and similar constructions are favored in both [2, 3]. In the similarity weighting layer of the architecture, we combine the question encoding and the context passage encoding in order to figure out which of the words are most important to the question at hand.

The problem then becomes how to determine similarity. Since we are dealing with vectors, the most obvious solution is to use the cosine similarity between the question vector and the context passage's individual word vectors, as inspired by the affinity matrices of [3] and the relevancy matrix of [2]. Under this metric, we would be considering the similarity of an individual word to the question has a whole.

In order to create greater flexibility in the model, however, we also added an additional weighting matrix W as a parameter, which we used before calculating the dot product, producing a bilinear similarity score. Mathematically, we can write that the similarity between word i and the question, denoted α_i, is represented as the modified cosine similarity function

$$\alpha_i = \frac{(\mathbf{q}')^T W \mathbf{c}_i'}{\|\mathbf{q}'\| \|\mathbf{c}_i'\|}$$

Once we have our set of $\alpha_1, \dots, \alpha_m$ in hand, we can then use these values to produce a weighted representation of the context passage, which we will denote C''. Here, we want to weight a given word vector in C' by its similarity to the question vector, which we presume, for our purposes, is a reflection of how relevant it is to the question (under the assumption that similar word vectors have similar meaning). To do so, we define C'' as follows:

$$C'' = [\alpha_1 \mathbf{c}_1', \alpha_2 \mathbf{c}_2', \dots, \alpha_m \mathbf{c}_m'] \in \mathbb{R}^{2\xi \times m}$$

3.4 Prediction

Now that we have create C'', which, as we discussed above, is a representation of the context paragraph, weighted to account for a given word's similarity to the question, we need to actually determine what that answer to the question actually is. To do that, we create a two layer feed-forward neural network according to these specifications:

$$\mathbf{h}_i = \tanh\left(W\mathbf{c}''_i + \mathbf{b}_1\right)$$
$$\hat{y}_i = \text{softmax}\left(U\mathbf{h}_i + b_2\right)$$

where $W \in \mathbb{R}^{\xi \times 2\xi}$, $U \in \mathbb{R}^{1 \times \xi}$, $\mathbf{b}_1 \in \mathbb{R}^{\xi}$ and $b_2 \in \mathbb{R}$.

We then run each vector $\mathbf{c}''$ through two such feed-forward networks, each with its own parameters. From one neural network, we interpret $\hat{y}_i$ as the probability of word i being the start of the answer span, while the output of the other gives us the probability of it being the end. Intuitively, then, our prediction for the beginning and end of the answer span will be the most likely candidate in each case.

3.5 Heuristic result management

In addition to the neural network layers that were described above, there were also two important layers that overlay the entire network, and served to "sanity check" the output of the model. The first of these was what we referred to internally as multiplication by a "sanity mask". Concretely, this was used a mask of the same shape as the context paragraph's input, filled with only ones and zeros, where the ith element was 1 only if there was an ith token in the unpadded input sequence. The purpose of this mask was to "restore sanity" to our answer, by which we mean prevent the prediction of indices that were outside of the bounds of the actual input. For example, it doesn't make sense to predict that the tenth word of a five word sequence was the correct answer, and our sanity mask was in place to prevent that very situation from occurring.

Interestingly, we found that this mask was most effective was applied *after* cost was calculated. The prevailing theory in our analysis of this was that applying the mask before calculating cost prevents the model from learning why those inputs were wrong, and artificially inflates the score of the correct answer, and thus generating a fictitiously low cost. By postponing the masking until after the cost of calculated, we were able to reap the benefit to F1 and EM that came from avoiding impossible indices, but also provide as much learning opportunity for the model as possible.

The second overlay on our network was a index flip operation. Specifically, it is inherent in the properties of an answer span in the context of answer extraction that the start index must precede the end index. However, at times the model's output violated this condition. Rather than attempt to train the model to avoid such behavior, we opted to overlay a heuristic measure that would improve the overall F1 performance, without adding complexity to the neural layers of the model. This heuristic simply consisted of checking to see whether the start index was greater than the end index, and flipping them if they were.

4 Evaluation

Having developed the model that we have described above, it is important to evaluate its performance on actual training, development, and testing data. Using an implementation of the model we have advanced thus far, we trained it on 80,000+ tuples of question, context paragraph, and answers, along with a smaller set of validation data.

4.1 Training process

4.1.1 Learning parameters

In order to train the model, we implemented cross-entropy loss on the results of our model, calculating the loss with respect to the starting index of the answer span and the ending index separately, before ultimately adding them to arrive at our total loss. In order to adjust the model in response to the loss, we used TensorFlow's `AdamOptimizer`, with what we eventually found to be the optimal learning rate of 0.001. Higher learning rates ultimately led to the model settling at a relatively high loss of 7.5 to 8.5.

After each epoch we ran, we also evaluated our model on a withheld validation set in order to test the model's success in generalizing to unseen data and thereby prevent overfitting. By observing on performance on this data, we were able to eventually determine the optimal values for our embedding size $d = 100$ and hidden state size $\xi = 100$. Additionally, to prevent our model from overfitting our training data, we included dropout layers on the feed-forward network and the LSTM encoders, using a dropout rate of 15%.

4.1.2 Training performance

Over the course of training our model, we monitored how the loss changed over time, to ensure that it was actually able to learn in response to new data points. If we turn to Figure 2, it is apparent, despite the noise, that the loss did drop dramatically over the first several thousand batches, before eventually settling to a plateau at approximately 5.8 total loss (corresponding to approximately 2.9 loss each for both the starting and ending indices), though even then, it continues to drop, albeit slowly.

Although far from state-of-the-art performance, this model certainly did better than random. Using cross entropy loss, a loss of 2.9 for the start and ending indices implies a probability of 5.5% assigned to the correct answer. While not ideal, with a padded context paragraph size of 750, as used in our training, a completely uniform probability across all possible indices would produce a probability of only 0.13%—an entire order of magnitude worse than the likelihoods assigned by our model. Thus, we can see that, although the model is certainly doing poorly relative to many of the top-performing models, (and indeed, even robust logistic regression models as in [4]), it still performs better than a simple random guessing algorithm, which is promising.

4.2 Overall performance

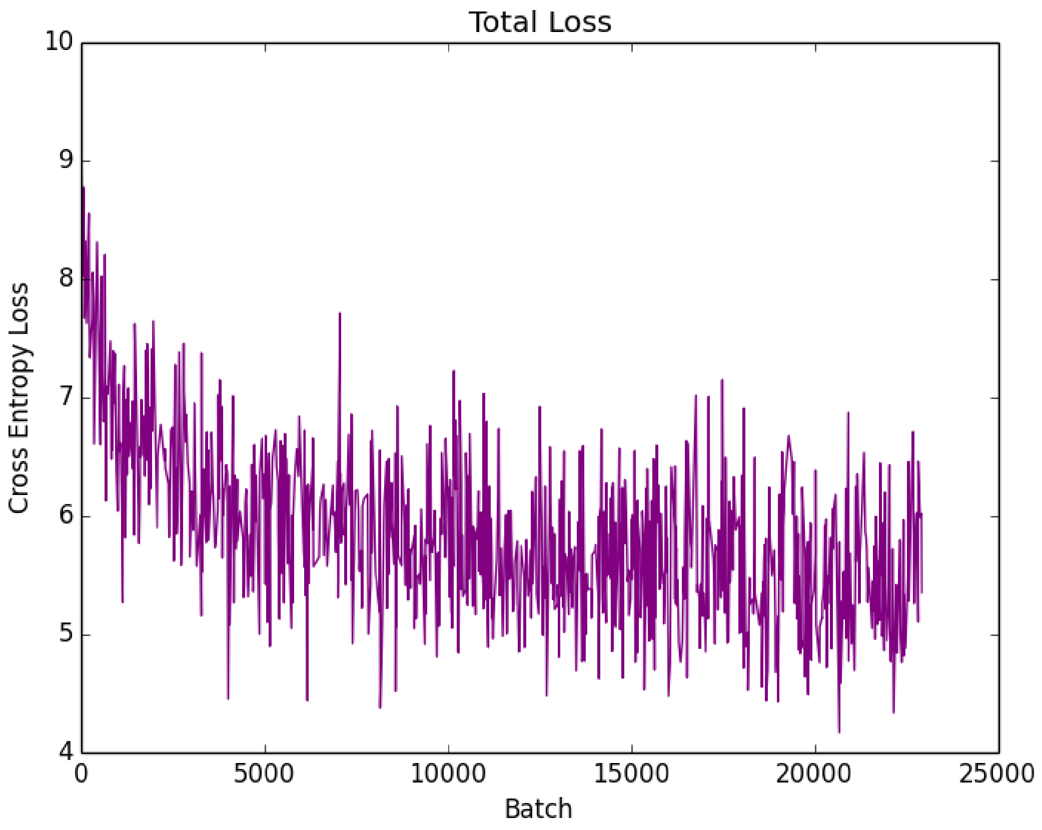
After training our model on the 80,000+ data points in our dataset, it was necessary to see how well the model generalizes to unseen data, such as the privately held SQuAD test set. Ultimately, our performance on these was on par with our performance during training. During each of the training epochs, our model performed a validation step by randomly selecting ≈ 100 data points from a withheld validation set and determining the overall performance of the model on this new data. In general, by the third or fourth epoch , we had achieved an F1 score of approximately 25% and produced the exact answer about 15% of the time. This did, of course, fluctuate over the training process, especially as we moved into the eighth and ninths epochs and the model began to slightly overfit out training data, but overall, our performance was, on average, about 25% F1 and 15% exact match.

With this in mind, out performance on the test and development set leaderboards was unsurprising. On the development set we acheived an F1 score of 23.633% and an exact match score of 15.838%—exactly in line with the performance we had achieved during the training process. In Table 1, we compare the our own model's performance against those of both human performance on the dataset, as well as the current five best performing models, according to the SQuAD leaderboard. As we can see in examining this table, there is significant room for growth in our model's performance.

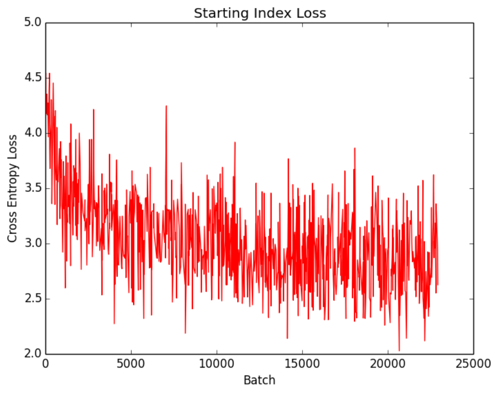
4.3 Other areas of exploration

Although many of our initial ideas eventually proved ineffective, there were several additional techniques that we considered over the course of designing our model that had potential, but were ultimately not implemented in the final version. Here, we discuss some of these ideas and reflect on their potential for use in future models.

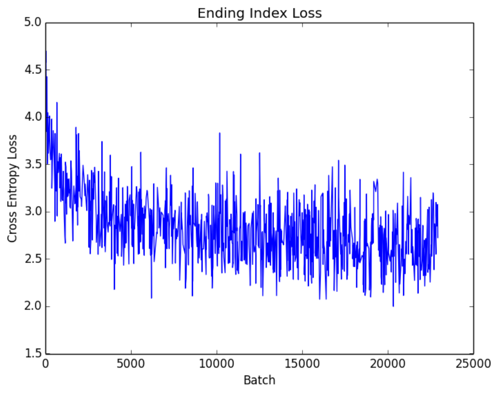
One of the first ideas we considered did not actually relate to the structure of the neural network at all, but rather the process by which it was trained. Although we eventually settled on the simple cross-entropy loss as the driver of learning in our model, we also considered some additional parameters that would factor, with varying weight, into the overall cost. One idea that we considered for a while was to penalize the model based on the distance between the correct index and the predicted index, as opposed to only considering the probability that was assigned to the correct answer. Our theory



(a) Total Loss



(b) Start index loss



(c) End index loss

Figure 2: Loss graphs

Table 1: Model Performance Comparison [1]

	Score	
Model	F1	EM
R-Net (Ensemble)	84.006	76.922
ReasoNet (Ensemble)	82.552	75.034
BiDAF (Ensemble)	81.525	73.744
Multi-Perspective Matching (Ensemble)	81.257	73.76
R-Net (Single Model)	80.751	72.401
Human Performance	91.221	82.304
Our Model	**23.633**	**15.383**

in formulating this idea was that being off by a single position in the context paragraph is preferable, and far less inaccurate, than if we were off by over 100, as happened in our predictions, especially in the early batches. Thus, by penalizing the model for the degree of inaccuracy, as opposed simple cross-entropy loss, we could potentially train it to at least be close to the correct answer, even if it was not entirely correct.

Unfortunately, this idea proved ineffective, as were many of the other loss function ideas that we considered. The problem was that, at the root, the probability assigned to each of the indices in the context paragraph was the output that determined the loss across the board. However, because the values of the loss under other metrics, such as that described above, had larger magnitudes, these secondary loss functions ultimately drowned out the cross-entropy loss and the information it contained therein. Nonetheless, it would be an interesting investigation to see if there are ways of more heavily penalizing distant incorrect answers as compared to adjacent or nearby answers so as to encourage the model to minimize the distance between the predicted and correct answers, even when it cannot get the answer perfectly correct. The question here would be whether or not we can optimize the approximation, even when the precise answer eludes us.

An additional area in which I would have liked to be able to make greater progress is in different methods of combining the question representation with the context so as to better establish vital relationship between the question and the paragraph in determining the correct answer. Many of the papers that I have referenced, such as [2] developed a complex and intricate, but effective, model for combining the question and context paragraph representations. In future work, it would be interesting, and likely advantageous to explore ways of combining these representations in ways that preserve the relationship between the question and passage, especially if we were to build off of the models suggested in the literature.

5 Conclusion

Over the course of this paper, we have advanced a relatively simple model that attempts to solve the problem of text-based answer extraction. Unfortunately, despite our best efforts, our model was unable to perform even remotely as well as the top of the line models, and indeed, failed to even beat robust logistic regression models [4]. Nonetheless, the results were better than random guessing, which suggests that although not as high-performing as we would like at the moment, the basic architecture could likely serve as a useful starting point should we wish to continue to research the problem of answer extraction in the future.

References

[1] SQuAD: The Stanford Question Answering Data Set. `https://rajpurkar.github.io/SQuAD-explorer/`, 2017. Accessed: 20 Mar 2017.

[2] Zhiguo Wang, Haitao Mi, Wael Hamza, and Radu Florian. Multi-perspective context matching for machine comprehension. *arXiv.org*, arXiv1612.04211v1, 2016.

[3] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks. *arXiv.org*, arXiv1611.0604v3, 2017.

[4] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ Questoins for Machine Comprehension of Text. *arXiv.org*, arXiv1606.05250v3, 2017.

[5] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.