Question Answering on the SQuAD Dataset

Do-Hyoung Park
Department of Chemical Engineering
Stanford University
Stanford, CA 94305
dpark027@stanford.edu
Codalab: dhpark

Vihan Lakshman
ICME
Stanford University
Stanford, CA 94305
vihan@stanford.edu
Codalab: vl29

Abstract

We develop a deep learning framework for question answering on the Stanford Question Answering Dataset (SQuAD), blending ideas from existing state-of-the-art models to achieve results that surpass the original logistic regression baselines. Using a dynamic coattention encoder and an LSTM decoder, we achieved an F1 score of 55.9% on the hidden SQuAD test set. In this paper, we present the methodology governing our question answering model. In addition, we critically analyze potential shortcomings and limitations of our algorithm and others in the literature and propose extensions for future work to push the boundaries of question answering further.

The SQuAD Task

Machine question answering remains one of the most important problems in artificial intelligence and natural language processing for both its rich potential in numerous applications requiring information synthesis and knowledge retrieval as well as its inherent difficulty, testing the boundaries of computers to process human language. Because of the added difficulty in parsing text to draw meaningful conclusions, question answering has emerged as a very fertile area for advancements in end-to-end deep learning models and innovations in natural language processing techniques.

In the past decade, several datasets for question answering tasks have been proposed. These resources, while valuable towards the end-goal of training question-answering systems, all experience a considerable tradeoff between data quality and size. Some older datasets, such as those of Berant, et al. and Richardson, et al. use human curated questions and effectively capture the nuances of natural language, but – due to the labor involved in generating the questions– are often insufficient for training robust machine learning models for the task. Conversely, datasets generated via automation, such as those of Hermann, et al. and Hill, et al. lack the structure of authentic human language and, as a result, lose the ability to test for the core skills involved in reading comprehension. Moreover, many of these older datasets employ multiple choice or single-word formats for the ground-truth answers which inherently limits the ability of models to learn linguistic structure when formulating answers.

In response to the shortcomings of these previous question answering datasets, Rajpurkar, et al. released the Stanford Question Answering Dataset (SQuAD) in 2016. Utilizing questions generated from 536 Wikipedia articles by a team of crowdworkers, SQuAD consists of over 100,000 rows of data – far exceeding the size of similar datasets – in the form of a question, an associated Wikipedia context paragraph containing the answer to the question, and the answer. The ground-truth answer labels are represented in the form of two indices, a *start* index a_s and an *end* index a_e, which represent words in the context paragraph.

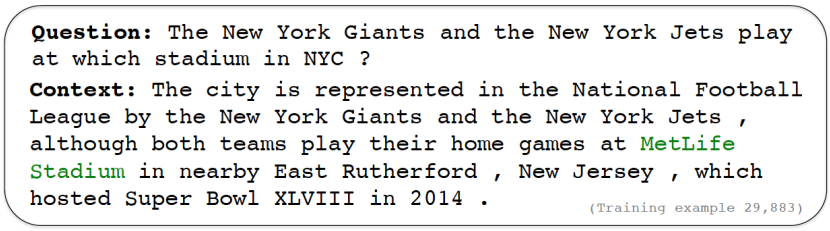


Figure 1: A training example from the SQuAD dataset, consisting of a question, context paragraph, and answer span (in green)

Formally, we define the **SQuAD Question Answering Task**: Given a three-tuple $(Q, P, (a_s, a_e))$ consisting of a question Q, context paragraph C, and start and end indices a_s and a_e, the goal of the problem is to predict the *answer span* $\{a_s, a_{s+1}, \ldots, a_{e-1}, a_e\}$ where each a_i is an index of the context paragraph corresponding to the answer.

In this paper, we develop an end-to-end deep learning model for the SQuAD task. We utilized 80% of the dataset for training the model, 10% for validation and hyperparameter tuning. The final 10% of the dataset is reserved for testing and kept private by the creators of SQuAD for the purpose of preserving the integrity of question answering models. For our final evaluation, we submitted our fully trained model to Codalab to test on the withheld data.

Related Work

Although the SQuAD dataset was released less than 12 months ago, the NLP and deep learning communities have already achieved very impressive results on a rather challenging machine learning task. In the original June, 2016 paper introducing SQuAD, Rajpurkar, et al. developed a logistic regression model based on detailed linguistic features that achieved an F1 score of 51%, a significant improvement over a naive sliding window baseline that recorded 20% F1.

In November of the same year, the state-of-the-art was pushed significantly with the development of sophisticated neural network architectures for the question answering task. Xiong, et al. introduced a dynamic coattention network for encoding and attention modeling, a framework that allowed for a fine-grained weighting of each question work on the associated context paragraph. In addition, they devised a highly intricate dynamic pointer decoder to iterate over potential answer spans to achieve an F1 score of 75.9%.

Our final model was heavily influenced by this prior work. In particular, we implemented the authors' encoder and attention network exactly and replaced the pointer decoder with a simpler, more efficient variant that utilized a single linear transform and an LSTM to learn differentiation between the question and context. We will discuss the particulars of our model design and implementation in the next section.

Also in November 2016, Seo, et al. utilize bidirectional attention flow (BiDAF) encoding and attention schemes, combining question and context influence via a bidirectional LSTM, along with multistage decoding decoding, in a model that achieves 77.3% F1.

We also implemented the overall structure of this model, with a few small simplifications in the encoding step. While our model showed great promise during training, we ultimately found our implementation to be infeasible given our time constraints for this project. In particular, we calculated that a single epoch under our BiDAF algorithm would take approximately 9 hours to complete, meaning that a full 10-epoch training would take approximately four days, which we deemed too costly given that we had approximately seven days to submit the project at the time. We will discuss our implementation of this model in more detail in a subsequent section and

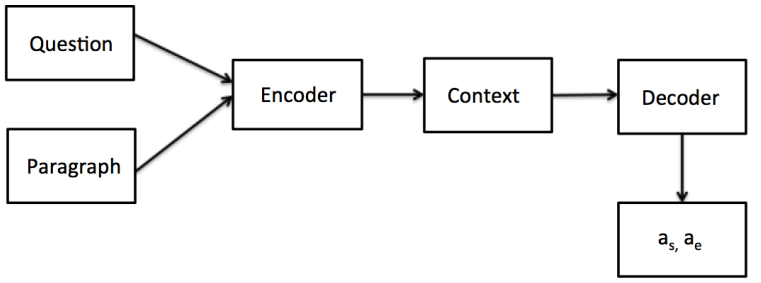


Figure 2: Sequence-to-sequence model for SQuAD task

hypothesize why it proved to be computational intractable for our purposes.

In December of 2016, Wang, et al. introduced multi-perspective context matching which aggregates words in the context with relevancy weights to compute "matching vectors, which are then aggregated to predict the beginning and ending points of the answer. This neural network structure achieved an F1 of 77.8%. This algorithm continues to rank as one of top single-model entries on the global SQuAD leaderboard [1].

With the SQuAD dataset only released within the last year, new advancements continue to emerge with rapid frequency. The current top model (currently secret), developed by Microsoft Research Asia, was submitted to the leaderboard five days prior to the publication of this paper. This model recorded an F1 score of 84%, which pushes the state of the art even closer to the accuracy human performance on the SQuAD reading comprehension task, which stands at 91% F1.

The Model: Coattentive Encoding with LSTM Decoder

Overview

In this section, we recap the general framework and philosophy behind sequence-to-sequence modeling in NLP and explain, at a high level, how we apply this approach to solving the SQuAD task. Originally developed for solving the machine translation problem, a sequence-to-sequence model is an end-to-end model consisting of two separate algorithms: an *encoder* and *decoder*. Both the encoder and decoder utilize recurrent neural networks (RNNs) as the backbone behind their respective computations. The encoder is responsible for receiving the input sequence of the model and returning a "context" which succinctly captures the necessary information about the input for the purposes of achieving the desired goal. The context is often represented as a vector or a matrix. The decoder, in turn, takes in the context as input and proceeds to generate an output sequence, which is returned as the final answer.

In the context of machine translation, the application of the sequence-to-sequence paradigm is natural: words of one language are fed into the encoder as input and the decoder outputs a sequence of words in the target language to be translated. In the context of question answering, we apply a sequence-to-sequence model by treating the question and paragraph words as inputs to the encoder, generating a blended context between the two, and having the decoder output the start and end indices of the answer span (or, equivalently, the sequence of answer tokens). This approach is the dominant paradigm in the literature and has already led to outstanding results for question answering.

[1] https://rajpurkar.github.io/SQuAD-explorer/

Preprocessing

Before presenting the full details of our neural network architecture, we will first discuss the preprocessing steps we took to prepare our inputs for the encoding step. We read in the raw SQuAD dataset and represented each question and paragraph as a list of indices corresponding to the position of each word in an external dictionary. Next, we utilized pre-trained GloVe vectors (Pennington, et al.) to obtain a word vector for each word in the question and paragraph. Thus, we ultimately represented each question and paragraph as a matrix where the ith row corresponds to the word embedding for the ith word in either the paragraph or question. Initially, we used 100-dimensional word embeddings pretrained on the Wikipedia corpus to train our model before fine-tuning our system by switching to 300-dimensional GloVe vectors trained on the Common Crawl corpus.

One additional hurdle we had to overcome before deploying our question and paragraph representations into the encoder was the issue of non-uniformity in sentence lengths. Both our encoder and decoder utilize recurrent neural network structures that required inputs of equal length to maintain the same number of RNN time steps. We resolved this problem by setting the question and paragraph lengths as hyper-parameters in our model. After visualizing the data via histograms, we decided to set the question length to be 25 and the paragraph length to be 250. We then clipped any question or paragraph longer than these respective prescribed lengths. Conversely, we padded shorter questions and paragraphs with the buffer token <PAD> which we then masked out in the encoding phase.

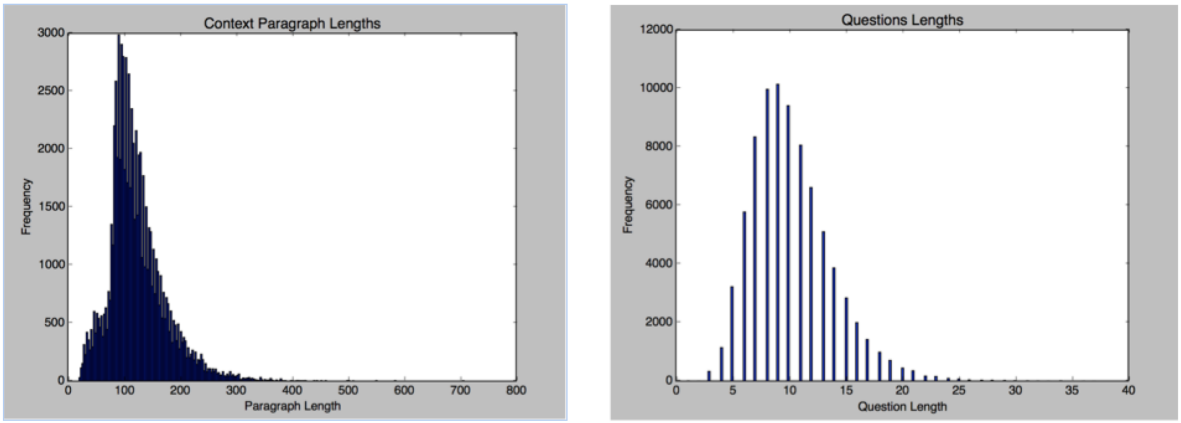


Figure 3: Frequency of question and paragraph lengths in SQuAD training set. This visualization influenced where we clipped question and paragraph lengths.

Identifying good hyper-parameter values for the paragraph length and question length proved to be extremely influential in determining the performance of our model in terms of efficiency. Originally, we naively set our question and context paragraph lengths to be loose upper bounds on the length of any sentence in the dataset, choosing values of 45 and 715, respectively. This design choice ensured that we would not have to clip any sentences. Our rationale was that such a choice would guarantee that our model would have a chance of predicting any answer since we eliminated the possibility of a paragraph being clipped at an index before the answer start token. However, this choice meant that, on average, (as indicated by the histogram), we were heavily masking most questions and paragraphs in the dataset, which increased the time required for training considerably (since each RNN included many more additional layers). However, once we tuned these parameters to their current values, we were able to train our model in a reasonable amount of time and achieve much-improved results.

Encoder and Attention

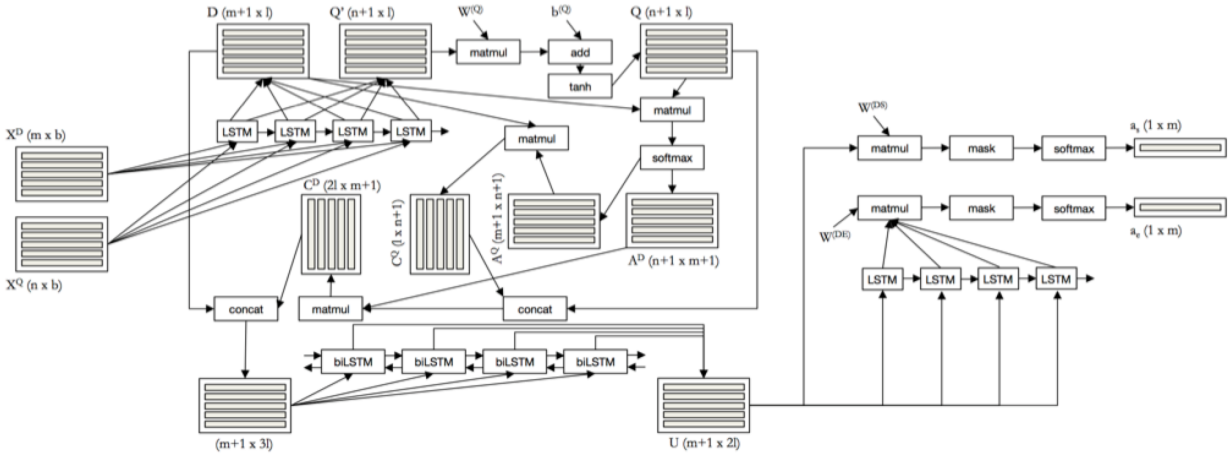


Figure 4: Diagram of Full Model with encoder and decoder

Now, we will discuss the full details of our model. Our encoder segment is a direct implementation of the dynamic coattention encoder originally proposed by Xiong, et al. We implemented our entire neural network architecture using the Python TensorFlow libraries, and trained our system for five epochs with a batch size of 10.

As a matter of notation, let m be the number of words in the question and n the number of words in the paragraph. The encoding step begins by taking the word embeddings for every word in the paragraph $x_1^P, x_2^P, x_3^P, \ldots, x_n^P$ and the embeddings for every word in the question $x_1^Q, x_2^Q, \ldots, x_m^Q$ and passing each of these sets of vectors through the *s*ame LSTM. Mathematically, we can represent this operation as

$$\mathbf{p_t} = \mathbf{LSTM}(\mathbf{p_{t-1}}, \mathbf{x_t^P}) \quad \mathbf{q_t} = \mathbf{LSTM}(\mathbf{q_{t-1}}, \mathbf{x_t^Q})$$

In our model, we made the design choice to keep the original authors' decision of maintaining one LSTM with one set of weights to train as opposed to two since we postulated that a commonality in the weights would, in essence, allow us to condition the context paragraph with respect to the question. This conditioning feature is extremely important on the SQuAD dataset since the same paragraphs are used as context for multiple questions, so it is crucial for our encoder to take the specific question into account.

After passing both the question and paragraph embeddings through the LSTM, we construct encoding matrices Q and P where each the ith rows of each matrix corresponds to the LSTM output at the ith time step when Q and P are respectively given as input. In addition, we append "sentinel vectors" as the final row of each matrix, giving us

$$Q' = \begin{bmatrix} —q_1— \\ —q_2— \\ \vdots \\ —q_m— \\ —q_\emptyset— \end{bmatrix} \qquad P = \begin{bmatrix} —p_1— \\ —p_2— \\ \vdots \\ —p_n— \\ —p_\emptyset— \end{bmatrix}$$

Where the sentinel vectors $p_\emptyset$ and $q_\emptyset$ are randomly initialized vectors whose values are learned by the model. We use Xavier initialization to populate the values at the start. The purpose of the sentinel vectors is to introduce some irregularity in the representations question and context

paragraph. Originally, we considered omitting the sentinel vectors from our implementation, but we realized that they would help us in overcoming a problem we experienced in our preliminary baseline approach of the model consistently predicting one of three indices as both the start and end answer tokens.

We then pass the matrix Q' through an additional hyperbolic tangent layer of nonlinearity to further this separation.

$$\mathbf{Q} = \tanh(\mathbf{W}^{(\mathbf{Q})}\mathbf{Q}' + \mathbf{b}^{(\mathbf{Q})})$$

We then construct our coattention matrix between the paragraph and context by utilizing matrix multiplication and a softmax normalization across both the rows and columns.

$$\mathbf{A}^{\mathbf{Q}} = \text{softmax}(\mathbf{P}\mathbf{Q}^{\mathbf{T}}) \quad \mathbf{A}^{\mathbf{P}} = \text{softmax}(\mathbf{Q}\mathbf{P}^{\mathbf{T}})$$
$$\mathbf{C}^{\mathbf{Q}} = \mathbf{P}^{\mathbf{T}}\mathbf{A}^{\mathbf{Q}} \quad \text{(coattention matrix)}$$

The attention matrix, as we discovered, was one of the most critical pieces to the model – it added a boost of approximately 10-15% to our F1 score. From analyzing the question answering problem, we can see the value of attention since the essence of reading comprehension is in knowing which words of the question are most pertinent to formulating an answer.

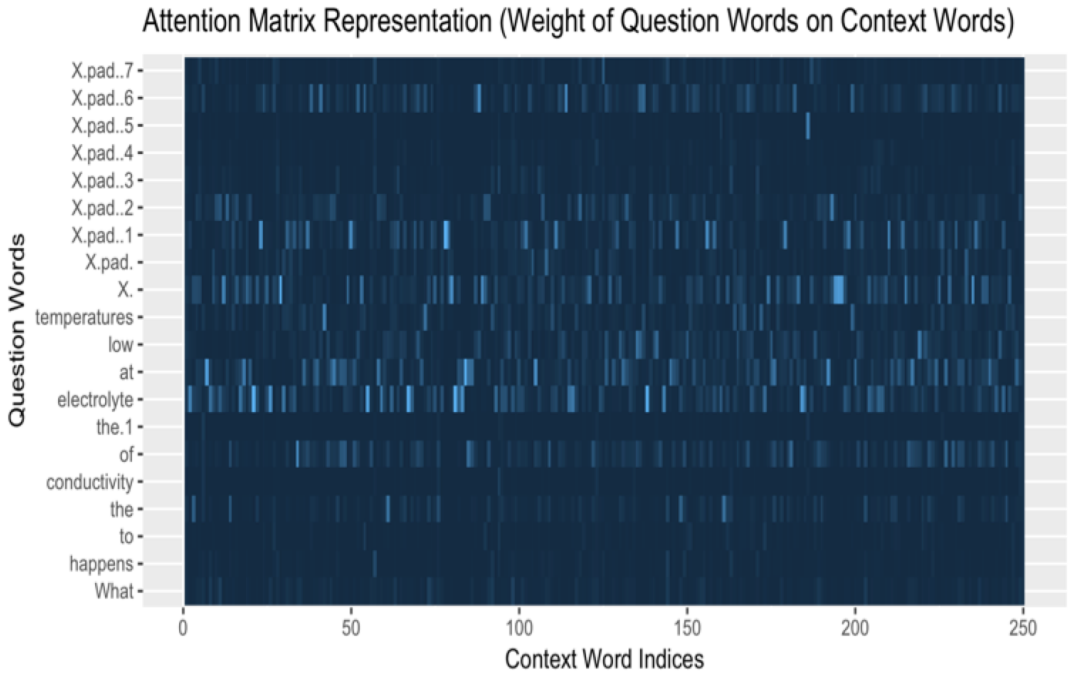


Figure 5: Heat map of weight of question words on each context word

Finally, with out coattention matrix C^Q, we combine information from the question and paragraph again (where we hypothesize the model uses this redundancy to enforce that the context is conditioned on the question) to produce our final context U returned by the encoder.

$$\mathbf{C}^{\mathbf{P}} = \left[\mathbf{Q}; \mathbf{C}^{\mathbf{Q}\,\mathbf{T}}\right]^{\mathbf{T}} \cdot \mathbf{A}^{\mathbf{P}}$$
$$\left[\mathbf{P}; \mathbf{C}^{\mathbf{P}\,\mathbf{T}}\right] \rightarrow \text{Bi-LSTM} = \mathbf{U}$$

Where in the final step, we define U such that each row is the concatenation of the forward and backward outputs at each time step of a bidirectional LSTM.

Decoder

For our decoder, we deviate from the model implemented by Xiong, et al. From our experience implementing the bidirectional attention flow decoder, we observed that a complex decoding step, while helpful to a model's performance, can increase training time considerably. With that in mind, we opted for simpler decoder that learns two weight matrices $W^{(DS)}$ and $W^{(DE)}$ which correspond to decoding the start word and the end word, respectively.

We begin by multiplying $W^{(DS)}$ by our encoding context U and then taking the softmax of the resulting matrix to produce a probability distribution over the paragraph indices of each word being the start index. We then pass the context U through an LSTM to introduce some separation between the start and end indices. Then we repeat the same process of taking a matrix multiplication and then applying a softmax to obtain a probability distribution over potential end answer words.

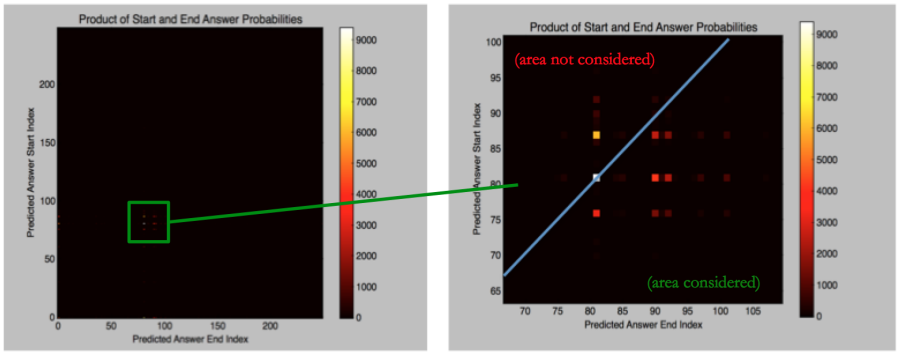
$$\begin{aligned} \text{softmax}\left(W^{(DS)}U\right) &= p_s \\ U \rightarrow LSTM &= U' \\ \text{softmax}\left(W^{(DE)}U'\right) &= p_e \end{aligned}$$

where A_s is the probability distribution over the paragraph words of each word being the start index and A_e is defined analogously for the end index.

Post-Processing

For the final step of our algorithm, we take the probability distributions over potential start and end context words p_s and p_e and choose the paragraph indices i and j such that the product $p_s(i) \cdot p_e(j)$ is maximized subject to the constraint that $i \leq j$ (which enforces that the start must come before the end). Naively, this can be computed in quadratic time by checking all possible pairs of indices, but we devised a more efficient dynamic programming algorith, that runs in linear time. The algorithm works as follows: Fix the start index at 0 and then make a pass over the end token distribution to find the index j that maximizes $p_s(0) \cdot p_e(j)$. Then, fix j and make a linear pass over the start distribution and find the index i such that $p_s(i) \cdot p_e(j)$ is maximized subject to $i \leq j$. We then return i and j as our a_s and a_e predictions.

Visually, we can see this post-processing algorithm in action via a heat map.



Question: What happens to the conductivity of the electrolyte at low temperatures?

75: .
76: The
77: conductivity
78: of
79: the
80: electrolyte
81: drops
82: at
83: low
84: temperatures
85: ,
86: which
87: increases
88: equivalent
89: series
90: resistance
91: .

Figure 6: Left: Full decoding heat map. Right: Zoomed-in map on high-probability area

Results

Ultimately, we achieved the following results, where "Test" below refers to the hidden test set maintained by the SQuAD creators.

	Training	Validation	Dev	Test
F1	76.25%	59.76%	54.91%	55.92%
EM	62.2%	44.7%	40.3%	41.4%

Figure 7: Results

Evaluation Metrics

In the SQuAD task, we keep track of two relevant metrics: F1 and exact match. The F1 metric, which we have already referenced many times before, is the measure of average overlap between a model's prediction and ground-truth answer span. Formally, $F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$ where precision is defined as the ratio of correctly predicted words in the answer span to the total number of predicted answer span words. Recall, meanwhile, is the ratio of correctly predicted answer span words to total number of words in answer span. The exact match metric is a binary value that takes on a value of 1 if the predicted answer and true answer are exactly equal (not counting punctuation and articles) and zero otherwise.

Analysis of Results

As a whole, our model performed reasonably well, surpassing the logistic regression baseline established by Rajpurkar, et al. In particular, our model did extremely well at predicting answers in the form of dates, names, and numbers as shown below.

	Dates/Times/Stats	People/Places	Other
Total #	24	19	56
EM	18	13	17
EM%	**75%**	**68%**	**30%**

Figure 8: Results Sampled from first 100 validation data points

In addition, we also observed that our model was much more effective at predicting shorter answer spans versus longer ones. We hypothesize that the reason for this discrepancy stems from the fact that longer answer spans require greater understanding of the nuances of language versus simply identifying a target word or pair of words. Our (as with almost any question answering model) likely still lacks the ability to learn more complex linguistic structure. In short, our model would likely perform very well on a dataset of questions consisting exclusively of very short answers, but still struggles on handling the more involved examples in SQuAD – a testament to the challenges of this specific task and a signal of where we need to turn our attention to make improvements.

Conclusions, Discussion, and Future Work

We successfully implemented an end-to-end deep learning model for the SQuAD Question Answering task that replicates a state-of-the-art encoder and utilizes a simple, but efficient decoder to achieve promising preliminary results for this very challenging problem.

In fact, one other point of discussion we wish to raise is that the limitations of the SQuAD dataset actually underplays the efficacy of our model. Since every answer to a question in SQuAD is a fixed pair of indices, the question answering task leaves no room for nuance and can mark other technically correct answers as incorrect. As a concrete example, from the dataset, consider the following output from our model compared to the ground-truth answer for the question "As what did the law of 81 BC view human sacrifice?"

```
True answer: murder
Predicted answer: murder committed for magical purposes
F1: 0.33333; EM: false
```

One of the other major takeaways from implementing our model was the absolutely critical role of hyper-parameter tuning. As discussed earlier, our decision to change the length at which we clipped the paragraphs and questions improve our effieciency by a factor of three and made training over multiple epochs before the deadline a tractable reality. With more time, we would love to investigate more hyperparameter decisions. We also changed our word embeddings to 300-dimensional GloVe embeddings to make a significant improvement in performance, but many of our other attempted tweaks, such as introducing dropout with drop probability 0.5 to curb overfitting, annealing the learning rate did not produce any improvements or, in some cases, degraded our performance. With more time, we would like to run a proper hyperparameter search algorithm over other parameters (e.g. batch size, LSTM hidden layer dimension, and hopefully converge on values that would boost our performance.

In short, our performance on the Question Answering task appears to be a good start, but our model certainly has limitations as described above along with the fact that there seems to be a theoretical limit on just how useful SQuAD can be as a proxy for measuring reading comprehension given the lack of nuance in answer choices.

Other models implemented

In addition to the model we described in detail in this paper, we also implemented two other models: a simple baseline to arrive at a working system as soon as possible, and a modified implementation of the bidirectional attention flow scheme proposed by Seo, et al. Our first model (the baseline) consisted of a simple linear encoder that multiplied the question and answer word embeddings by weight matrices W^Q and W^P before then taking the matrix multiplication of the resulting product. We ignored attention. For the decoding step, we used the same procedure as we described in the paper with the omission of the additonal LSTM to separate the question and context futher. This model achieves an F1 score of approximately 10%.

Our second model closely followed the bidirectional attention flow model of Seo, et al. We implemented the decoder described in the paper exactly and made one simplifying modification to the encoding/attention phase; when computing the similarity matrix S described in the authors' original paper, we ignored taking a Hadamard product between the question and context paragraph. Our rationale was that this step seemed overly complex and unnecessary since the paragraph and question embeddings were later blended together. However, given our previous discussion in this paper over the utility of reduncy in the dynamic coattention network, it is possible that this simplication was costly. Utimately, as we mentioned, we were only able to train this model for less than an epoch due to its inefficiency and our time constraints, but in the future – with more time and perhaps a more streamlined implementation – we would be interested in training this model as well and possible creating an ensemble with the model outlined in this paper.

Future Work

As we have alluded to throughout this section, the SQuAD task is a very challenging problem in natural language processing and there are many avenues for improvement and further exploration. In addition to more hyper-parameter tuning and training more sophisticated models for the purposes of creating an ensemble (goals that are almost strictly time-related), we would love to explore novel neural network architectures in solving this problem such as perhaps applying convolutional neural networks for character-level embeddings and for attention, exploiting the parallel between attention and several computer vision tasks. We would also be interested in incorporating ideas from other areas of artificial intelligence, most notably reinforcement learning. One idea we thought about but did not have time to explore in any depth is applying rules governing natural language, such as grammar and syntactic structure, to create a set of Markov decision process-like rules and view question answering as a task of navigating a set of rules (while incorporating information about the question) to arrive at an answer.

Acknowledgments

We would like to thank Profs. Chris Manning and Richard Socher for an excellent course and the CS224N teaching staff for their helpful feedback and troubleshooting advice (at all hours of the day). We would also like to acknowledge Microsoft Azure for providing us with the necessary computing resources and The Stanford Daily, LLC for use of their facilities as we battled through the inner workings of TensorFlow (which we would also like to thank for existing). Finally, we are deeply indebted to the hours of 4-5 and 5-6am for countless moments of inspiration.

References

[1] Abadi, Martn, et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." arXiv preprint arXiv:1603.04467 (2016).

[2] J. Berant, V. Srikumar, P. Chen, A. V. Linden, B. Harding, B. Huang, P. Clark, and C. D. Manning. 2014. Modeling biological processes for reading comprehension. In Empirical Methods in Natural Language Processing (EMNLP).

[3] K. M. Hermann, T. Kocisk y, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom. 2015. Teaching machines to read and comprehend. In Advances in Neural Information Processing Systems (NIPS).

[4] F. Hill, A. Bordes, S. Chopra, and J. Weston. 2015. The goldilocks principle: Reading childrens books with explicit memory representations. In International Conference on Learning Representations (ICLR).

[5] Hochreiter, Sepp, and Jrgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

[6] N. Kushman, Y. Artzi, L. Zettlemoyer, and R. Barzilay. 2014. Learning to automatically solve algebra word problems. In Association for Computational Linguistics (ACL).

[7] Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global Vectors for Word Representation." EMNLP. Vol. 14. 2014.

[8] Rajpurkar, Pranav, et al. "Squad: 100,000+ questions for machine comprehension of text." arXiv preprint arXiv:1606.05250 (2016).

[9] Seo, Minjoon, et al. "Bidirectional Attention Flow for Machine Comprehension." arXiv preprint arXiv:1611.01603 (2016).

[10] Wang, Shuohang, and Jing Jiang. "Machine comprehension using match-lstm and answer pointer." arXiv preprint arXiv:1608.07905 (2016).

[11] Xiong, Caiming, Victor Zhong, and Richard Socher. "Dynamic Coattention Networks For Question Answering." arXiv preprint arXiv:1611.01604 (2016).

[12] Wang, Shuohang, and Jing Jiang. "Machine comprehension using match-lstm and answer pointer." arXiv preprint arXiv:1608.07905 (2016).