Ask BiDAF

Mitchell Douglass, Griffin Koontz, Caelin Tran
{mrdoug95, griffink, cktt13}@stanford.edu
Stanford University | CS 224n | Winter 2018

Abstract

Machine language comprehension stands as a highly researched goal within the field of natural language processing (NLP). In the fundamental question-and-answer (QA) task, a system must appropriately answer a query given contextual information. In this paper, we explore the implementation and extension of one state-of-the-art approach to the QA task on the Stanford Question Answering Dataset (SQuAD) [5]: a recurrent neural network model using bidirectional query-context attention [7]. After testing different word and character embeddings, neural network configurations, hyperparameter settings, ensembling techniques, and other extensions, such as a candidate answer prior based on answer lengths in the training set, we achieved a final F1 score of 79.937 and EM score of 71.562 on the SQuAD test set, demonstrating the utility of bidirectional attention flow (BiDAF) in machine comprehension.

1 Introduction

The exact QA challenge we address can be characterized as follows. Given a human-readable query Q and a context C of natural language text containing the answer to said query, identify the span of text in C that most appropriately answers Q. As an example, a query might be "When was Benjamin Franklin born?" while the context might be an excerpt from the biographical Wikipedia article on Mr. Franklin. The appropriate answer extracted from the context would then be "January 6, 1705".

We used the Stanford Question Answering Dataset to train, develop, and test our model. SQuAD is the result of a massive crowdsourcing project whereby human participants assembled more than 100,000 contexts, queries, and corresponding answers from the English Wikipedia corpus [5]. In the SQuAD Challenge, researchers compete to develop QA models that achieve cutting-edge performance on a private test set curated by the authors of the original dataset. Submissions have far exceeded the performance of the published logistic regression baseline (F1 score of 51.0) [5] and continue to approach human performance (F1 score of 91.2) [5].

In our approach to the SQuAD Challenge, we applied deep learning with neural networks, which has experienced a resurgence in recent years due in large part to the increasing number of large, high-quality datasets that are required for training. These models are attractive when compared to traditional approaches due to their mostly end-to-end nature, i.e., their ability to learn using only raw data as input, often doing away completely with the costly and time-consuming process of feature engineering. Indeed, nearly all top postings on the SQuAD leaderboard were achieved with deep learning models. Our particular model leverages a popular machine comprehension paradigm in Seo et al. [7], utilizing recurrent neural networks (RNNs) and attention to achieve performance close to the state of the art.

2 Background and Related Work

Almost all competitive SQuAD models incorporate three predominant characteristics, the first of which is the use of RNNs. A recurrent neural network is an array of artificial neurons, arranged in series, that can effectively model temporal dependencies between a sequence of inputs, such as a sentence of words. An extension of RNNs called long short term memory networks (LSTMs) are common in practice as they better facilitate the propagation of information along many steps of a sequence and also solve other difficulties with standard, less expressive RNNs [3]. Many models developed for SQuAD, including ours, employ LSTMS as their ability to model contextual relationships of words in sequence has proven crucial to machine comprehension of the semantics

of natural language. They are used to generate quantitative vector representations of a query and its associated context.

The second heavy-lifting characteristic of competitive SQuAD models is attention. Just as a human reader might glance back and forth between a context and a query in order to zero-in on an appropriate answer, attention is a mechanism whereby context and query vector embeddings interact in order to produce a combined, "query-aware" representation. A first approach, called "Context to Query" (C2Q) attention, consists of simply augmenting each context vector by a linear combination of the query vectors, applying weights to query vector according to their similarity to the context vector via some differentiable metric (e.g., dot product). Seo et al. additionally augmented context vectors with a fixed linear combination of the context, weighting context vectors by maximum similarity to any query vector [7]. Another model, dynamic co-attention networks [8], augments the query with an analogous linear combination of the context, before again applying C2Q. Together, these techniques are examples of "Query to Context" (Q2C) attention, and they generally supplement C2Q by allowing the query to attend to important regions of the context.

Finally, the use of dense vector embeddings as input features, as compared to one-hot word representations, is known to provide substantial improvements in many NLP tasks, and SQuAD models are no exception. Global Vectors (GloVe) of words are a popular choice of embeddings, as they have been shown to perform well on a variety of intrinsic and extrinsic tasks [4]. In addition, the use of character embeddings as input features has been shown to improve performance of models on SQuAD and other NLP tasks, often providing the model with actionable information when exposed to words during prediction that were unseen during training [2].

3 Model

Our model consists of six hierarchical layers.

1. **Character embedding layer:**
   Generates a fixed-length vector representation for each word in the query and context based upon trainable vector representations for each character.

2. **Word embedding layer:**
   Maps each word in the query and context to a pretrained vector representation [4].

3. **Context awareness layer:**
   Provides query and context word embeddings with context-awareness using bidirectional LSTM A (see Figure 3).

4. **Attention layer:**
   Incorporates bidirectional attention, outputting query-aware context vectors.

5. **Modeling layer:**
   Adds two stacked LSTMs B and C that provide additional processing of the attention layer output. Also adds LSTM D that generates a separate output using activations from C.

6. **Output layer:**
   Generates a final prediction of the answer start and end indices based on outputs from the modeling and attention layers.

Intuitively, layers one, three, and five provide context-awareness at different scales. The character embedding layer provides character embeddings with information about the surrounding characters, while the context awareness layer provides each word representation with knowledge of the neighboring words. Then, the modeling layer provides query-aware context representations with information about the surrounding query-aware context representations.

Character embedding layer:
Using the convolutional approach outlined in Santos et al. [2] [6], this layer generates a fixed-length vector representation for each word based on trainable character embeddings. It aims to capture character-level information by generating local features around each character and running a max operation to condense them into a fixed-size word embedding.

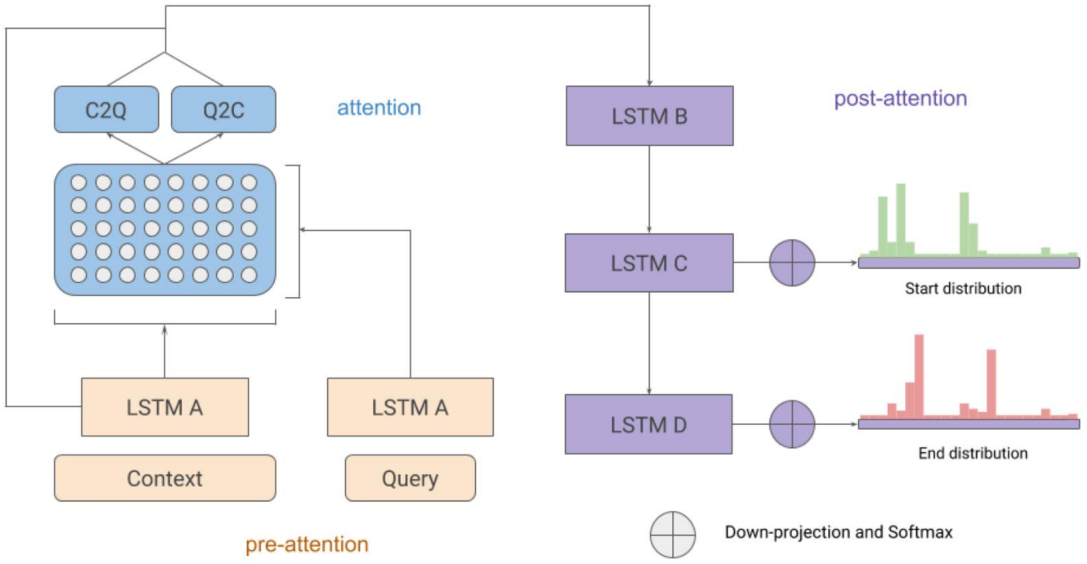


Figure 1: Our model architecture

Suppose we start with a word containing A characters $\{c_1, c_2, ..., c_A\}$. A character embedding lookup operation maps each character to a corresponding vector representation, generating $\{r_{c_1}, r_{c_2}, ..., r_{c_A}\}$, where r_x is the character embedding for character x. Subsequently, the convolutional layer applies a matrix-vector product to each window of k characters, building a feature matrix in $\mathbb{R}^{A \times f}$, where f is the number of filters applied by the convolutional layer. Finally, a max-pool operation takes the maximum value for each feature across all windows, yielding a fixed-size output vector in $\mathbb{R}^f$.

This process is applied to every word in the context and query. Given a context containing N words and query containing M words, this layer outputs $\mathbf{C}' \in \mathbb{R}^{N \times f}$ and $\mathbf{Q}' \in \mathbb{R}^{M \times f}$ for the context and query respectively.

Word embedding layer:
This layer concatenates pretrained word vectors (using GloVe vectors trained on 840 billion tokens from Common Crawl) [4] for the context and query words with $\mathbf{C}'$ and $\mathbf{Q}'$ from the previous layer, outputting $\mathbf{C} \in \mathbb{R}^{N \times (d_g + f)}$ and $\mathbf{Q} \in \mathbb{R}^{M \times (d_g + f)}$, where d_g is the GloVe vector dimensionality.

Contextual embedding layer:
This layer passes the context and query embedding matrices through bidirectional LSTM A, sharing weights for both the query and context. Output from the forward direction is concatenated with output from the reverse direction, yielding $\mathbf{H} \in \mathbb{R}^{N \times 2d_A}$ for the context and $\mathbf{Q} \in \mathbb{R}^{M \times 2d_A}$ for the query, where d_A is the output dimensionality of LSTM A in a single direction.

Attention flow layer:
Following Seo et al. [7], we utilize a bidirectional attention flow where the context attends to the query, and the query attends to the context. Both computations are derived from a similarity matrix $\mathbf{S} \in \mathbb{R}^{N \times M}$, such that

$$\mathbf{S}_{ij} = \alpha(H_{i:}, Q_{j:}) \in \mathbb{R}$$

where $\alpha(h, u) = w^T[h; u; h \circ u]$, for some trainable weight vector $w \in \mathbb{R}^{6d}$.

The context-to-query (C2Q) attention computes, for each context word, a weighted sum of query word representations from the previous layer. For the t^{th} context word vector, this is defined as:

$$\tilde{U} = \sum_i \text{softmax}(S_{t:})_i U_{i:}$$

where $\tilde{U} \in \mathbb{R}^{N \times 2d}$ contains the attended query vectors for the entire context. Intuitively, $\text{softmax}(S_{t:})_i$ is a scalar that captures the similarity between context word t and query word i.

Similarly, the query-to-context (Q2C) attention computes, for the entire query, a weighted sum of context-aware context word representations, which is tiled N times to build $\tilde{H} \in \mathbb{R}^{N \times 2d}$. First we compute the weights b to be applied to each context word:

$$b = \text{softmax}(\max_{row}(\mathbf{S}))$$

This determines the weight for a context word t based on the maximum value in $\mathbf{S}_{t:}$, which intuitively captures how similar t is to *any* word in the query. Finally we compute $\tilde{h}$:

$$\tilde{h} = \sum_i b_i H_{i:}$$

which simply takes a weighted sum using the weights we defined above. In the final step, we combine C2Q, Q2C, and $\tilde{\mathbf{h}}$ from the previous layer to build $\mathbf{G} \in \mathbb{R}^{N \times 8d}$:

$$\mathbf{G}_{t:} = \beta(\mathbf{H}_{t:}, \tilde{\mathbf{U}}_{t:}, \tilde{\mathbf{h}})$$

where $\beta(h, \tilde{u}, \tilde{h}) = [h, \tilde{u}, h \circ \tilde{u}, h \circ \tilde{h}]$.

Modeling layer:
This layer runs $\mathbf{G}$ through two additional bidirectional LSTM layers, to provide each query-aware context word representation with information about the other query-aware context word representations. It generates $\mathbf{M} \in \mathbb{R}^{N \times 2d_C}$, where d_C is the output dimensionality of LSTM C. In the output layer, $\mathbf{M}$ is used to predict the probability distribution of start indices for the answer.

To predict the probability distribution of end indices, we run $\mathbf{M}$ through an additional LSTM layer D to obtain $\mathbf{M_2} \in \mathbb{R}^{N \times 2d_D}$, where d_D is the output dimensionality of LSTM D. This step allows the model to extract different features needed to compute the end index probability distribution given information about the start distribution.

Output layer:
To predict the start and end index, we compute:

$$p^{start} = \text{softmax}(w_{start}^T[\mathbf{M}; \mathbf{G}]) \in \mathbb{R}^N, \quad p^{end} = \text{softmax}(w_{end}^T[\mathbf{M_2}; \mathbf{G}]) \in \mathbb{R}^N$$

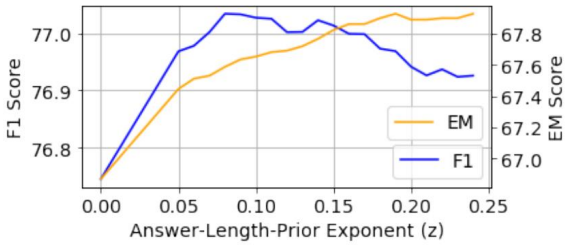
where $w_{start}, w_{end} \in \mathbb{R}^{10d}$ are trainable weight vectors. Note that we concatenate the appropriate modeling layer output with the attention layer output when predicting both start and end indices, instead of solely using $\mathbf{M}$ or $\mathbf{M_2}$. This allows the model to make the final prediction with all the contextual information from the attention layer at hand.

Final Prediction:
To output a final prediction, we generate matrix $\mathbf{F}$ of probabilities of all potential answer spans $(i_{start}, i_{end}) = (i, j)$, where $\mathbf{F}_{ij} = p_i^{start} p_j^{end}$, setting to zero all locations where $j < i$. We then predict range (i, j) which gives the maximal value of $\text{P}(j-i)^z \cdot \mathbf{F_{ij}}$. Here, the function $P(\ell)$ is a distribution which models the prior belief about answer length distributions, and it is computed from the distribution of answer lengths from the training set, using Laplace smoothing for unseen answer lengths. The exponent $z \geq 0$ is a tunable hyper-parameter for controlling the influence of this prior belief on final range prediction (see figure 2 for impact of this prior on our models).

Table 1: Test set performance comparison

Model	*F1*	*EM*
Human Performance [5]	91.221	82.30
State of the art [1]	89.28	82.48
BiDAF (single) [7]	77.3	68.0
BiDAF (ensemble) [7]	81.1	73.3
Our model (ensemble)	79.94	71.56

Figure 2: single-model dev-set performance



4 Results

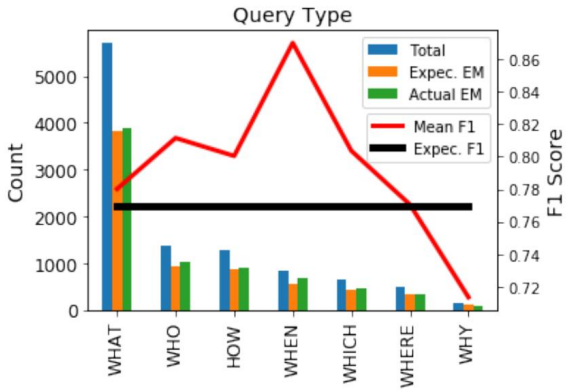
We trained our complete model on a Microsoft Azure NV6 VM with an M60 GPU, for a total of 14 epochs or roughly 12,000 batches of 100 examples each. Upon inspection of the training set, we found that more than 98% of contexts contain 300 or fewer tokens, while more than 99% of queries contain 30 or fewer tokens, so we truncate contexts and queries to these respective lengths in order to significantly reduce training time without sacrificing performance. We use a hidden state size of 150 for each LSTM except for end prediction (LSTM D), where we use a hidden size of 100. Intuitively, increasing the hidden state size of these LSTMs allows for a more expressive model, and our ablation tests confirm this. Using a model ensembling technique (detailed below), our model achieved an F1 score of 79.94 and EM score of 71.56 on the official SQuAD test set (see Table 1). Our single model, incorporating an answer-length prior exponent $z = 0.08$, achieved an F1 score of 77.15 and EM of 67.73 on the development set.

We performed extensive development set ablation tests on a number of subcomponents of our model in order to understand the marginal contribution of each component; the results are in Table 2. Our "Full model" baseline consists of a single neural-network model with all components, however the answer-length prior is ignored to avoid dependence on a tuned value of the exponent z for each ablation. We notice that all components contribute to the high performance of the full model. Experimenting with embeddings trained using the English Wikipedia corpus (roughly 6B tokens) as well as the Common Crawl corpus (roughly 840B tokens), we saw that the Common Crawl embeddings exhibited improved performance over the Wikipedia embeddings. However, ablating only Common Crawl word embeddings or only character embeddings resulted in a decrease of no more than 1.1 in F1 and 1.6 in EM. Ablating both resulted in a decrease of more than 2.5 F1 and 3.5 in EM. This result is not surprising, as character embeddings and Common Crawl similarly address out-of-vocab words as well as capture additional semantic structure between words. Based on ablation test results, another key feature of our model is the combination of two bidirectional LSTMs prior to start prediction, feeding into a final bidirectional LSTM for end prediction; ablating these two features resulted in a F1 decrease of 1.27 / 1.62 and an EM decrease of 2.39 / 2.25, respectively. Ablating bidirectional attention and replacing with standard C2Q attention resulted in a modest decrease of 1.1 F1 and 1.87 EM, showing that Q2C does contribute to model performance. Finally, we performed an ablation replacing the 150-dimensional LSTMs

Table 2: Development set performance of ablated models

Ablated model component(s)	*Replacement component(s)*	*F1*	*EM*
Full model (single)	-	76.98	67.22
150-dim hidden states	100-dim hidden states	76.43	66.04
Character embeddings	-	75.97	65.62
840B GloVe word embeddings	300D Wikipedia GloVe	75.89	65.70
Bidirectional attention flow	Basic attention	75.88	65.35
Linked start and end LSTMs in modeling layer	Separate LSTMs	75.71	64.83
Two-layer start LSTM in modeling layer	One-layer LSTM	75.36	64.97
840B GloVe and character embeddings	Only 300D GloVe	74.47	63.85
Modeling layer, LSTMs, BiDAF, char, 840B	GRU baseline with 100D GloVe	43.95	34.67

Figure 3: Model performance under different conditions



Answer length

Ground truth length (ℓ)	*Count*	*Mean F1*	*Mean EM*	*Median pred. length*
$\ell = 1$	3654	79.51	75.59	1
$\ell = 2$	2881	78.59	71.40	2
$2 < \ell \leq 4$	2455	78.51	67.01	3
$4 < \ell \leq 8$	1163	68.70	49.36	4
$8 < \ell \leq 16$	378	56.56	34.92	4
$16 < \ell \leq 30$	39	54.64	25.64	6

with 100-dimensional counterparts, showing that 150 dimensions provides a small advantage of 0.55 F1 and 1.18 EM.

Ensembling is a technique whereby the predictions from multiple independent instances of a model are combined to produce final predictions, the idea being that the biases of any particular instance are negated during the process of combining. One approach, called "max sum ensembling", extracts a probability for each non-empty range (i_{start}, i_{end}) by taking the outer product of start and end distributions and then selecting the range with the greatest sum, summing over all models. We applied max sum ensembling to 9 checkpoints taken from 3 independent training sessions, from which we took checkpoints after 8,000, 10,000, and 12,000 batch iterations. With this configuration, we achieved our best performance of 79.94 F1 and 71.56 EM given in table 1. Interestingly, we found that the answer-length prior did not contribute to the performance of the ensemble as it did for the single model.

5 Error Analysis

Even our best model fails to match a human answer for roughly 30% of examples and exhibits a 20% inefficiency in F1 score. Understanding for which contexts and/or queries our model performs poorly (and for which it performs well) gives valuable insight into both possible model improvements as well as the inherent complexity of the reading comprehension problem.

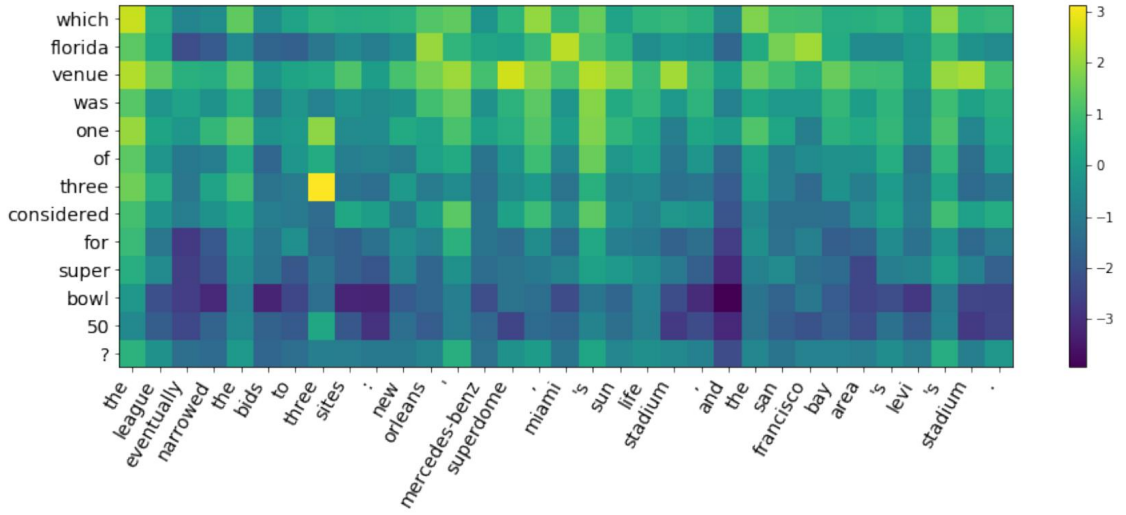
Figure 3 shows the breakdown of the dev dataset by query type, showing our model's performance on each type compared to mean overall model performance. We note that a majority of queries in the dataset are "what" questions, and further that our model under-performs on this query type. We find this intuitive, as directives such as "who" or "when" unambiguously require a person or time, whereas the "what" directive is more general and requires further parsing of the query to understand its requirement. The low performance of "where" queries is surprising, and could be caused by the low proportion of queries in the dataset with this directive.

Next, we investigate the performance of our model with respect to varying context length, query length, and ground-truth answer length. We find that context length and query length have significant impact on our model's performance only for extreme values, with a modest 12%

Table 3: Model prediction error examples

#	*Query*	*Ground truth*	*Prediction*
1	How did Tesla know he was being struck by the particle?	he could feel a sharp stinging pain	stinging pain
2	What was Tesla's idea regarding Earth?	that the earth had a resonant frequency	resonant frequency
3	According to Lenin why must capitalistic countries have an imperialistic policy?	to constantly expand investment, material resources and manpower	constantly expand investment
4	What is grown in the fertile highlands?	Tea, coffee, sisal, pyrethrum, corn, and wheat	wheat

Figure 4: Heat map of query-context attention example



performance drop for 150 very-long contexts ($>$ 300 tokens) and a 10% performance drop for 220 very-short queries ($\leq$ 4 tokens). However, as summarized in Table 3, we notice considerable performance degradation for examples with ground-truth answers longer than 4 tokens, where our model is often too succinct. By observing a handful of these mistakes, we believe there are two components. First, our model struggles to identify which surrounding words can be included to produce a natural and complete response, as demonstrated by examples 1 and 2 of Table 3. Second, our model often fails to predict the entire contents of a list, even when all items are relevant to answering the query, as evidenced by examples 3 and 4 of Table 3. Apart from simply training on more data, we believe these problems might be alleviated by including features from a dependency parse on sentences from the context and query (e.g., appending embedding of parent under dependency parse), as this might inform the model on which additional words to include.

We also sought to better understand how our query and context interacted in our attention layer. Figure 4 shows our models attention given an example query about the venue of Super Bowl 50 and its associate context. Since our model computes softmax and max-pooling along these rows, a large positive value indicates stronger attention. We notice several interesting features within this attention matrix. First, query words "which", "venue", and even "one" attend strongly to the word "the" in the context as they realize the following noun may be important. The query word "florida" attends strongly to "miami", but also to other locations such as "san francisco" and "new orleands". Of course, the query word "venue" attends strongly to "superdome" or "stadium", but also "'s" (apostrophe s), apparently picking up on the tendency of venue names to end with these two characters. Also, quantities such as "one", "three", and "50" attend strongly to another quantity "three" within the context. Finally, we notice that the attention of query vectors fades towards the end of the query, and this possibly occurs as a result of these vectors being distant from the subject of the query.

6 Conclusion

We explored the application of bidirectional context-to-query/query-to-context attention and an advanced LSTM-based neural network architecture to the question-and-answer task on the Stanford Question Answering Dataset. Our extensions, especially the answer length prior, demonstrated the adaptability and potential of this approach driven by deep learning. We learned that although the QA problem is extremely difficult – our model occasionally exceeded our own reading comprehension abilities – it is tractable with sufficient data and appropriate computational techniques. In the future, we would like to try more architectural adjustments to our system (such as additional LSTM layers), partial retraining of word embeddings, additional features (such as tags from a dependency parser), more hyperparameter tuning (such as separate hidden size adjustments for each layer), and compact representation of the final model for prediction. Regarding the last point, our model in its full ensembled glory consists of nine separate checkpoints with over a million pa-

rameters each, which takes up quite a bit of space and limits its portability. Overall, our project's success is illustrative of incremental steps towards effective machine comprehension within the field of NLP.

Acknowledgements: We graciously thank Professor Socher and the CS224N teaching staff for their assistance and for developing an excellent course project, and Microsoft Azure and CodaLabs for providing training and evaluation infrastructure.

References

[1] Attention-over-attention neural networks for reading comprehension. 2016.

[2] B. Z. Cícero Nogueira dos Santos. Learning character-level representations for part-of-speech tagging. *PMLR*, 2014.

[3] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.

[4] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[5] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. SQuAD: 100,000+ Questions for Machine Comprehension of Text. *ArXiv e-prints*, June 2016.

[6] A. See, R. Socher, and TAs. Cs 224n default final project. 2018.

[7] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi. Bi-Directional Attention Flow for Machine Comprehension. *ICLR*, 2017.

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