Neural Machine Comprehension with BiLSTMs and Handcrafted Features

Kevin Wu
Columbia University
kjw2157@columbia.edu

Tze-Hsiang Wei
Columbia University
tw2623@columbia.edu

Sahil Manocha
Columbia University
sm4389@columbia.edu

Abstract

Since its creation 2016, the Stanford Question Answering Dataset (SQuAD) [4] has spawned the rapid development of deep neural network models for reading comprehension. In this project, we build a deep learning system for extractive question answering from the ground up. Using word embeddings and hand-crafted context features as the inputs to a simple bidirectional LSTM, we are able to achieve reasonably high accuracy with remarkably few parameters and model components.

1. Introduction

Neural question answering is one of the most promising areas of current deep learning research. From a machine learning perspective, question answering (QA) systems lie at the intersection of Natural Language Processing and Information Retrieval. Solving the question answering problem successfully would pave the way for a variety of high-impact technologies, including autonomous customer service agents that answer queries, email AI assistants, and general CRM systems. In this paper, we deal with the extractive QA task, in which answers to a particular query are fetched directly from the provided textual context.

Though complex attention-based models have achieved state-of-the-art results on machine comprehension tasks, we decided to start with the basic building blocks of a QA system to find out what types of baseline accuracies could be achieved with a simple bidirectional LSTM model without attention. For our approach, we followed closely the approach of the FastQA model [8], with several ablations and modifications.

2. Related Work

The release of SQuAD in 2016 has spurred the development of neural machine comprehension systems of ever-increasing accuracy and complexity. The current first place on the public leaderboard for the challenge, a QANet ensemble model jointly produced by Google Brain and Carnegie Mellon [9], has achieved near-human performance with an F1 score of 89.7 (compared to humans' F1 score of 91.2). Many of the top-performing results on the SQuAD leaderboard are the results of ensembled predictions, often of underlying base learners with different approaches to the machine comprehension problem. We briefly highlight some of the state-of-the-art architectures below:

- *R-NET:* Published by Microsoft Research in 2017, R-NET uses a gated attention-based recurrent neural network (based on LSTMs) to learn question-aware passage representations, each of which is then "self-matched" against a copy of itself to obtain an attention mechanism over the passage [7]. For the output, the authors use pointer networks to learn the probability of answer spans in the context [6].
- *BiDAF:* Released in 2017 by the Allen Institute is the Bi-Directional Attention Flow (BiDAF) model. BiDAF contains two layers of LSTM recurrent units; the top-most layer encodes the context and question into a single embedding over which two layers of attention are passed (Context2Query and Query2Context) [5]. The outputs from the attention layers are then passed through another LSTM layer, followed by the output layer.
- *QANet:* Published in 2018, Google Brain's QANet architecture discards recurrent architectures entirely, using convolutional neural networks to learn local question-context interactions and self-attention to learn global patterns. [9]

While most state-of-the-art advances in machine comprehension consist of highly complex attention-based models, in our project we tried to see how much performance we could obtain through a simple BiLSTM-based model with word embeddings and a small number of hand-engineered features. In doing so, we largely followed the example of Weissenborn et al. in their 2017 paper "Making Neural QA as Simple as Possible but not Simpler" [8]. In the paper, the authors achieve a 78.9% F1 score on SQuAD using a recurrent neural network architecture without attention. The

architecture of this question answering model, known as FastQA, is discussed in greater detail in Section 4.

3. SQuAD Challenge: Dataset and Evaluation

The Stanford Question Answering Dataset (SQuAD), consists of 100,000 question-answer pairs on over 500 passages taken from Wikipedia.

3.1. Training Data

The training data contains 87,599 context-question-answer pairs. An example of a particular question-context pair is given below, with the ground truth answer underlined.

> **Question:** *Where is the headquarters of the Congregation of the Holy Cross?*
> **Context:** *The university is the major seat of the Congregation of Holy Cross (albeit not its official headquarters, which are in Rome). Its main seminary, Moreau Seminary...*

3.2. Testing Data

The testing data consists of 10,750 context-question pairs, each with up to 3 unique ground truth answers from the context passage.

> **Question:** *Which region of California is Palm Springs located in?*
> **Context:** *Many locals and tourists frequent the southern California coast for its popular beaches, and the desert city of Palm Springs is popular for its resort feel and nearby open spaces.*
> **Answers:** *"southern' , "the desert", "southern"*

3.3. Evaluation Metrics

The SQuAD competition uses the following two metrics for evaluating answers:

- *Exact match:* The percentage of predicted answers that match exactly with any one of the ground truth answers.

- *F1:* This is the average maximum F1 score between the predicted answer and any of the ground truth answers. In this case, precision and recall are both calculated using a "bag-of-words" representation of both predictions and ground truth answers. Casing, punctuation, extra whitespaces, and articles ("a", "an", "the") are removed from the text snippets before calculating F1.

4. Model

In this section we describe in mathematical detail the different parts of our RNN model, highlighting any deviations we made from the original FastQA paper. A high-level overview of the architecture described below is shown in Figure 1.

4.1. Embedding

Word embeddings have become an indispensable technique in most state-of-the-art deep NLP models. An alternative to one-hot token embeddings, learned word embeddings allow for a deep distributed representation of word meaning in a low-dimensional space [2].

In the first step of our model, preprocessed and tokenized text data was transformed into a sequence of d-dimensional word embedding vectors. Call $\mathbf{X_p}$ an $n \times d$ dimensional input matrix representing a sequence of word embeddings corresponding to a passage of token length n, and $\mathbf{X_q}$ an $m \times d$ input matrix representing a sequence of word embeddings corresponding to a question of token length m.

For this step, we used pretrained word vectors (see Section 5.2 for details); out-of-vocabulary and zero-padding tokens were assigned a vector of 0.

4.2. "Word-in-Question" Features

In addition to the word embedding features, we augment our input data with two additional "word-in-question" features to assist our neural network model in matching questions to relevant text spans in the context. These features were first introduced by Weissenborn et al. in the FastQA model [8].

4.2.1 Binary "Word-in-Question"

For each context word x_j, we define the binary word-in-question feature, wiq_j^b to be 1 if the context word x_j is present in the question, and 0 otherwise. Formally, we have the following:

$$wiq_j^b = \mathbb{1}(\exists i : x_j = q_i) \tag{1}$$

To ensure that all textual inputs have the same dimensionality, we define word-in-question features for each question word q_i as well, and we set $wiq_i^b = 1$ for all question tokens q_i.

4.2.2 Weighted "Word-in-Question"

Again following the technique and notation laid out in [8], we next define a weighted word-in-question feature wiq_j^w for each context word x_j. To compute the weighted word-in-question feature for a given context word x_j, we measure

its similarity with each question word q_i using an elementwise multiplication of the word embeddings for q_i and x_j. We then take the dot product of the multiplied embeddings with a learned d-dimensional similarity vector v_{wiq}.

$$sim_{ij} = v_{wiq}(x_j \odot q_i) \tag{2}$$

$$wiq_j^w = \sum_i \text{softmax}(sim_{ij}) \tag{3}$$

Just as the binary word-in-question features, question inputs are assigned weighted word-in-question features of 1 in order to ensure that all textual inputs are of the same dimensionality.

After concatenating the two hand-engineered word-in-question features to the word embedding representations, we have $\tilde{\mathbf{X}}_\mathbf{p}$, an $n \times (d+2)$ dimensional input matrix representing the passage text, and $\tilde{\mathbf{X}}_\mathbf{q}$, an $m \times (d+2)$ input matrix representing the question text.

4.3. Recurrent Unit Architecture

4.3.1 LSTM Cells

First published in 1997, the Long Short-Term Memory (LSTM) cell has been one of the pioneering breakthroughs in RNN architectures, providing a complex recurrent unit that allows more efficient backpropagation of gradients through time [1]. For a neural network made up of a series of recurrent LSTM cells, for each element in the input sequence x_t, the network calculates a hidden state s_t, a candidate hidden state g_t, and an internal memory c_t. The equations for an LSTM cell are as follows:

$$\begin{aligned} i_t &= \sigma(x_t W^i + s_{t-1} U^i) \\ f_t &= \sigma(x_t W^f + s_{t-1} U^f) \\ o_t &= \sigma(x_t W^o + s_{t-1} U^o) \\ g_t &= \tanh(x_t W^g + s_{t-1} U^g) \\ c_t &= c_{t-1} \odot f_t + g_t \odot i_t \\ s_t &= \tanh(c_t) \odot o_t \end{aligned}$$

The LSTM cell consists of an input gate i_t, a forget gate f_t, and an output gate o_t, each of which is the result of passing an affine transformation of the input and previous hidden state through a sigmoid nonlinearity. We also similarly compute a candidate hidden state g_t using a $\tanh$ activation instead of a sigmoid.

Next we compute the cell state c_t by passing the previous cell state c_{t-1} through the forget gate f_t, and then add in the new candidate hidden state after passing it through the input gate i_t.

The final hidden state s_t is the elementwise product between the cell state c_t and the output gate o_t.

4.3.2 Bidirectional Flow

To augment the amount of data available to the recurrent neural network, we set up a bidirectional RNN architecture in which information is passed both in the forward-time and backward-time direction in two layers of LSTM cells (BiLSTM). Context inputs and question inputs, $(\tilde{\mathbf{X}}_\mathbf{p})$ and $(\tilde{\mathbf{X}}_\mathbf{q})$, are passed through the same recurrent layer (BiLSTM).

$$\begin{aligned} \mathbf{H}'_\mathbf{p} &= \text{BiLSTM}(\tilde{\mathbf{X}}_\mathbf{p}) \\ \mathbf{H}'_\mathbf{q} &= \text{BiLSTM}(\tilde{\mathbf{X}}_\mathbf{q}) \end{aligned} \tag{4}$$

The output of the two BiLSTMs are matrices corresponding to the encoded context and question inputs, of size $2(d+2) \times n$ and $2(d+2) \times m$, respectively.

4.4. Output Layers

Following the recurrent layer, the outputs from the BiLSTMs are passed through a fully-connected layer with a $\tanh$ activation.

$$\begin{aligned} \mathbf{H}_\mathbf{p} &= \tanh\left(\mathbf{B}_\mathbf{p} \mathbf{H}'_\mathbf{p}\right) \\ \mathbf{H}_\mathbf{q} &= \tanh\left(\mathbf{B}_\mathbf{q} \mathbf{H}'_\mathbf{q}\right) \end{aligned} \tag{5}$$

B_p and B_q are both matrices of size $(d+2) \times 2(d+2)$. Again following the example of Weissenborn et al.'s FastQA model [8], we initialize B_p and B_q to be the concatenation of two identity matrices of size $d+2$, so initially, $B_p = B_q = [I_{d+2}; I_{d+2}]$.

Next, we compute a weighted sum of the projected question representation, $\mathbf{H}_\mathbf{q}$, using as weights the outputs of a softmax activation layer over the m column vectors of $\mathbf{H}_\mathbf{q}$, $h_{q_1}, ..., h_{q_m}$. We define the transformed question representation, $\tilde{h}_q$, as:

$$\alpha = \text{softmax}(v_q H_q) \tag{6}$$

$$\tilde{h}_q = \sum_i \alpha_i h_{q_i} \tag{7}$$

This weighted sum $\tilde{h}_q$, is interacted with each element in the encoded and transformed context sequence in order to calculate predicted start and end indices for the answer span.

4.4.1 Start Index Network

To predict the start index, we use a feed-forward neural network containing a single hidden layer with a ReLU activation. The input to this feed-forward neural network, H_s, is a $n \times 3d$ matrix where each row h_{s_j} consists of a combination

of the projected context encoding for the jth element, h_{p_j}, and the weighted sum of the question encodings:

$$h_{s_j} = [h_{p_j}; \tilde{h}_q; \tilde{h}_q \odot h_{p_j}] \tag{8}$$

We pass each of these concatenated context representations through a two layer feed-forward neural network.

$$s_j = \text{ReLU}(W_{s_j} h_{p_j}) \tag{9}$$
$$p(s_j) = \text{softmax}(v_s S)_j \tag{10}$$

The probability of a given location j in the context being the start index of the answer span, $p(s_j)$, is computed by taking the softmax over all s_j.

4.4.2 End Index Network

Unlike in the FastQA paper, we compute the end index probability distribution separately from the start index probability distribution; Weissenborn et al. compute a separate conditional probability distribution over end indices, $p(e|s_j)$ for each possible output index s_j. To increase computational efficiency, the authors also implement a beam search within the network, only computing conditional probabilities for the 5 most likely start indices.

To simplify our architecture, we compute the end index probability distribution as a marginal distribution over end indices, and instead implement a beam search architecture at prediction time (see Section 5.4).

The architecture for the end index prediction network is thus exactly the same as before.

$$h_{e_j} = [h_{p_j}; \tilde{h}_q; \tilde{h}_q \odot h_{p_j}]$$
$$e_j = \text{ReLU}(W_{e_j} h_{e_j})$$
$$p(e_j) = \text{softmax}(v_e E)_j$$

4.5. Loss Function

The objective function for the network is sum of the cross entropy loss for the start index network and the end index network. Given the ground truth start and end index locations, with n-dimensional one-hot representations $\mathbf{y^s}$ and $\mathbf{y^e}$, the loss function is defined as:

$$J(\cdot) = -(\sum_{i=1}^{n} y_i^s \log p(s_i) + \sum_{i=1}^{n} y_i^e \log p(e_i)) \tag{11}$$

5. Experimental Setup

In this section we describe in detail how we implemented a neural comprehension system end-to-end, including the preprocessing of the raw text data, the training of the RNN model, and then the prediction of answer spans on the test set.

We used the entire training dataset provided by the creators of SQuAD to train our model (`train-v1.1.json`), and we used the accompanying development dataset (`dev-v1.1.json`) as both our validation and test dataset.

5.1. Preprocessing

Keras's `Tokenizer` class was used for processing the raw text data; all characters were converted to lowercase, and punctuation characters were stripped from the text. All tokenized passage sequences were right (post) zero-padded to a length of 700, and all tokenized question sequences were zero-padded to a length of 50; both input lengths were set to be greater than the maximum input length of the longest passage and question in the training data.

In a later iteration of the experiment, the BiLSTM model was modified to allow for different input lengths at training vs. test time. In particular, in FastQA and other implementations, training passage lengths were truncated to 400 (ensuring that the answer span was within the truncated sub-passage) and testing passage lengths were zero-padded to 1000; however, we found that this preprocessing modification failed to yield a significant improvement in F1 or exact match scores.

Ground truth answers in the training dataset were provided in the form of the index of the start character of the answer span, and the total character length of the answer span. These indices were converted into the start and end indices of the corresponding token sequence for the passage, using the same tokenizer as the passage preprocessing in order to ensure consistent input and output alignments.

5.2. Implementation

The code for the neural networks was written from scratch using Keras (with Tensorflow backend). Models were trained using an NVIDIA Tesla K80 GPU.

For the embedding layer, we used pretrained 300-dimensional GloVe word vectors covering 840 billion tokens from the Common Crawl corpus [3].

In mapping the preprocessed texts to the pretrained GloVe embeddings, out-of-vocabulary words and "padding" tokens were mapped to a 300-dimensional zero vector.

Concatenating the wiq^b and wiq^w features, to the word embeddings resulted in an dimensionality of 302 for each token, which we used as the number of hidden units in the Bidirectional LSTM model as well.

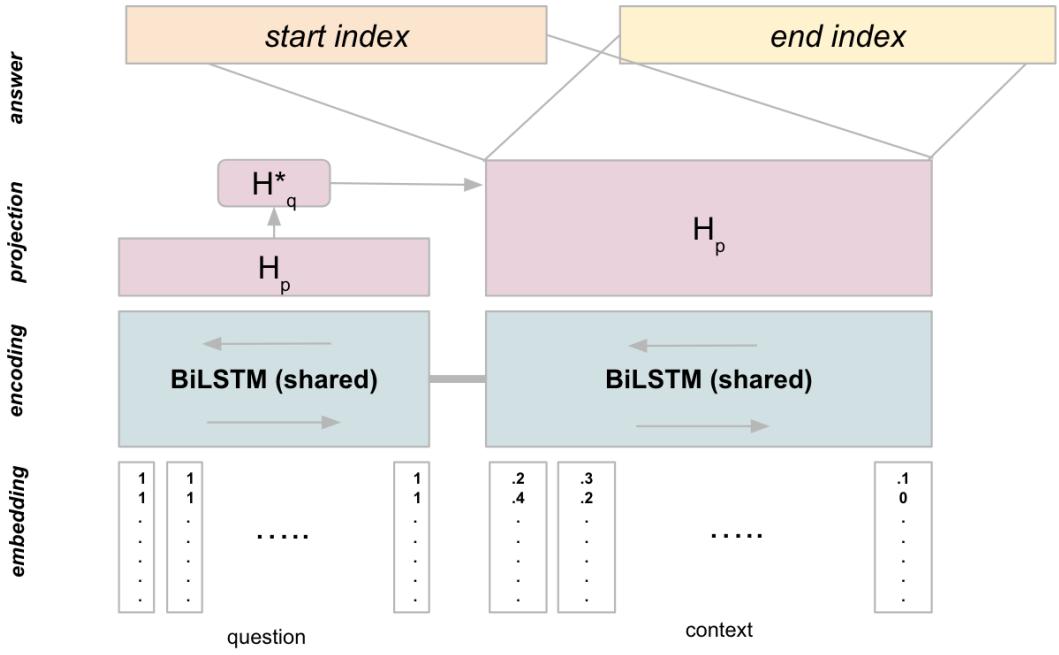

Figure 1. High-level overview of the model architecture used for machine comprehension task.

5.3. Training

Adam optimization was used with an initial learning rate of 10^{-3}. The batch size was set to 64; this was consistent with FastQA and other comparable methods, and we found that in practice, trying batch sizes lower than 64 simply slowed down training without a noticeable improvement in accuracy.

For regularization, we followed the example in the FastQA paper of introducing dropout at the embedding layer with $p = 0.5$. Dropout was applied column-wise rather than element-wise; in other words, entire tokens were dropped out during the training, rather than dropping out individual embedding elements in the tokens.

Prediction accuracies and losses on the validation dataset for the start index and end index were evaluated after each epoch. The Adam learning rate was halved every time the validation loss failed to decrease after an epoch.

5.4. Prediction

The output from our RNN model consists of two probability distributions over the tokens of the context (padded to length 700).

To predict the answer span from the context, we employed a beam search over start and end index distributions. First we chose the top K most likely start indices; then for each of the K start indices, s_j, we take an argmax of the end index probabilities for indices $s_j \leq e_j \leq s_j + L$. The final predicted answer span was chosen to be the start-end index pair giving the highest joint probability, $s_j * e_j$.

We found that at prediction time, using $K = 10$ and $L = 12$ performed best; and that utilizing this beam search technique instead of independently choosing and start and end index yielded an improvement in F1 score of around 1-2 percentage points.

6. Results and Discussion

A comparison of our model's performance on the SQuAD dev dataset with existing baselines and the reference FastQA paper is provided in Table 1. The following model results are used as a comparison:

- Logistic regression baseline: The logistic regression baseline model described in the original SQuAD paper uses a mix of categorical and continuous features from questions and context spans, including unigram and bigram frequencies, part-of-speech features, and dependency parse tree structures [4].
- Neural BoW baseline: Weissenborn et al. [8] train a feed-forward neural network which takes as input a mix of word embeddings, binary and weighted word-in-question features, and a speical "type matching" feature that is dependent on the question word. This

Model	F1	Exact Match
Logistic Regression baseline	51.0	40.0
Neural BoW baseline	56.2	43.8
BiLSTM + wiq^b + wiq^w	74.9	65.5
Our Model	*64.6*	*50.6*

Table 1. SQuAD Results on Dev dataset.

model considers all possible answer spans in the context passage for the training data and minimizes cross-entropy loss.

- BiLSTM + wiq^b + wiq^w: This is the FastQA implementation described in [8] without character embeddings and beam search.

As Table 1 shows, our model significantly outperforms both a naive logistic regression baseline and the more sophisticated Neural Bag-of-Words baseline introduced in [8], achieving a final F1 score of 64.6 and an exact match score of 50.6 on the dev dataset. However, the results achieved in our implementation are markedly different than the results reported in [8] with a comparable model (BiLSTM + wiq^b + wiq^w). Having followed closely the model training setup (i.e. optimization algorithm, learning rate, batch size, learning rate schedule) described in the original paper, the difference in results is most likely attributable to a different method of text preprocessing, or a discrepancy in the actual model architecture implemented.

7. Conclusion

Although falling short of state-of-the-art results for machine comprehension, in this work we demonstrate that attention is not necessary in order to achieve good results. A recurrent neural network based on bidirectional flows through LSTM cells, using only word vectors and a small number of hand-crafted features, can still outperform baseline models by a large margin. Future work will focus on refining these results, including matching the performance achieved by Weissenborn et al. [8] on a comparable model, and then adding in the additional components of FastQA, including character embeddings, highway layers, and beam-search.

References

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

[2] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[3] 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.

[4] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016.

[5] M. J. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016.

[6] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2692–2700, 2015.

[7] W. Wang, N. Yang, F. Wei, B. Chang, and M. Zhou. Gated self-matching networks for reading comprehension and question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 189–198, 2017.

[8] D. Weissenborn, G. Wiese, and L. Seiffe. Fastqa: A simple and efficient neural architecture for question answering. *CoRR*, abs/1703.04816, 2017.

[9] A. W. Yu, D. Dohan, M. Luong, R. Zhao, K. Chen, M. Norouzi, and Q. V. Le. Qanet: Combining local convolution with global self-attention for reading comprehension. *CoRR*, abs/1804.09541, 2018.