Question Answering Using Bi-Directional rNN

Aojia Zhao
Computer Science
Stanford University
Stanford, CA 94305
aojia93@stanford.edu

Simon Kim
Computer Science
Stanford University
Stanford, CA 94305
spkim@stanford.edu

Abstract

A sentence is a set of words, each of which can be represented in multiple dimensions using the GloVe model. Hence we can represent questions and context paragraphs as vectors of parametrized dimensions. We can then use different neural network techniques, such as Bidirectional-LSTM or GRU to train a model to give correct answers to given questions. However, until the SQuAD dataset became public, rNN models did not have sufficient amount of data to train their parameters. With SQuAD, the models can be trained so that, given a question and its corresponding context paragraph, they return the predicted indices of where the answer would lie.

This paper will explore past research papers to discuss the optimal architecture of neural network learning for the QA model, and discuss possible implementation approaches. Next, it will elaborate on the model implemented in this paper, and talk about possible improvements in the future.

Team name on Codalab: **das_stnemom**

1 Introduction

Recent improvements in the field of neural network have made rNN models competent in Machine Comprehension (MC) and Question Answering (QA). The rapid growth in computational powers of GPU and cloud computing may have contributed to the success, but more importantly the advent of database of rich supervised data has allowed rNN models to train on more effectively than in the past. Stanford Question Answering Dataset (SQuAD) is one of such database, holding pairs of questions and answers with their corresponding context paragraphs.

SQuAD is a database of questions whose answers are found in exact words from the given context paragraph. The answer parameter is a tuple of two indices that indicate the start and end of the phrase containing the actual expected answer to the question. With an appropriate model to represent the words of questions and context paragraphs as vectors, rNN models can use the SQuAD database to train their parameters for comprehending the context paragraph conditioned to the asked question.

One of the crucial components of using SQuAD database to train our model was how to represent a context paragraph conditioned on the question asked. Even when given the same context paragraph, if the question is different it is expected for the answer to be different under all conditions. We decided to first perform word representation of the questions and the context paragraphs using GloVe vectors, and then concatenate the resulting matrix with embeddings to represent question-context in a single vector. The resulting vector was then fed into an encoder, a bidirectional Long Short-Term Memory model, and then into a decoder that used softmax function to output the predicted indices of the answer. To assess the success of this model, we used F1 score and Exact Match (EM) score to compare against the baseline of 60% F1 score and 50% EM score.

2 Background

2.1 GloVe Word Representation

The GloVe model is effective in obtaining the vector representation of words.[4] It uses the statistics on co-occurrence of words in an unsupervised training dataset to represent words in vectors. For machine comprehension to be viable, it must first translate the words in questions and context paragraphs into vector representations. Hence the model implemented in this paper will use GloVe model for word representation layer.

2.2 Bi-LSTM Model

For our model we are implementing Bi-LSTM (Bidirectional Long Short-Term Memory), one of the architectures in recurrent neural networks (rNN). In traditional rNN, exploding and diminishing gradient issues have made models with long encoding nodes difficult to achieve good results. Nodes in the early stages of the nodes would either be lost or exaggerated as it passed multiple layers. Long Short-Term Memory (LSTM) accounts for these errors by implementing multiple gates: input, output, and forget gates. They would determine how much and far each cell would be propagated down the line, and have proved to answer well to the exploding and diminishing gradient issues.

Bi-LSTM takes LSTM further and implement it in both ways: forward and backward directions. Each vector is fed in both directions of LSTM - forward and backward. In the backward propagation each node receives as input the vector from a previous node as well as from a node in the forward propagation. The output vectors of these two LSTM models are concatenated as one to be used for the LSTM model in decoding part.

3 Related Work

3.1 Use of Attention Mechanism in Model

The paper *Teaching Machines to Read and Comprehend* (K. Herman, et.al.)[1] talks about the significance of having the attention function in the model. Its primary focus is on the possibility of machine comprehension of natural language with minimal prior knowledge of the language structure. It incorporated Deep LSTM to create two models of readers: the Attentive Reader and the Impatient Reader. The Attentive Reader uses bi-directional LSTM to create a composite output of question and context paragraph from both directions, which is then used to derive the answer. The Impatient Reader focuses more on the context paragraph by repeatedly referencing the context paragraph even after the Bi-LSTM has completed for the context to get the answer, which has an effect of recurrently accumulating information from the context paragraph to increase accuracy of the answer. Both of these readers do not have any attention mechanism. The empirical experiments showed that the use of neural network models, such as Bi-LSTM, in machine comprehension and question answering was promising compared to frequency and semantic models, but still underperformed compared to attention-based models that used a single layer of LSTM. This paper proved that the use of attention function is important in increasing the accuracy scores of the results.

3.2 Bi-Directional Attention Flow

We also looked at *Bi-Directional Attention Flow for Machine Comprehension* (M. Seo, et. al.)[2], which tries to answer to the tasks of question answering in machine comprehension by using Bi-Directional Attention Flow(BiDAF). BiDAF has advantage over previous models in that the attention vector representation flows from one layer to another instead of being summarized, which reduces the loss from early summarizing of the attention vector. The network uses character-level, word-level, and contextual embeddings to represent the context conditioned on the question by using bi-directional attention flow.

We observed that the paper used GloVe to obtain word embeddings of each word, and LSTM for the contextual embedding layer. For the modeling layer, it used the query-aware representations of context words as inputs to the model, which is a bi-directional LSTM. The output from the Bi-LSTM would be fed into the output layer to get the predicted answer. On a single model, this

network achieved 77.3 F1 score and 68.0 EM (exact match) score; on an ensemble it achieved 81.1 F1 score and 73.3 EM score.

3.3 Multi-Perspective Context Matching

We then looked at *Multi-Perspective Context Matching for Machine Comprehension* (Z. Wang, et.al.)[3] This network also has word-level and context-level representation layer, but there is a filter layer between word- and context-representation that filters out redundant information from the passage. The relevancy degree is the cosine similarity of the question and the passage vector, and it uses the value to put more weight on relevant words in passage. The context representation layer uses Bi-LSTM to encode contextual embeddings for question and context paragraph vectors. The output of the context representation layer is fed into the multi-perspective context matching layer.

The multi-perspective context matching layer processes the input by three procedures: Full-Matching, Maxpooling-Matching, and Meanpooling-Matching. Full-Matching compares each forward or backward contextual embeddings to that of the question; Maxpooling-Matching keeps only the maximum value of the comparison; Meanpooling-Matching keeps the mean of all the comparison values. The resulting vector is concatenated into a single vector. These different matching are crucial in answering questions with different span of answer in the context paragraph. The single model of MPCM achieved 75.1 F1 score and 65.5 EM score, and the ensemble achieved 77.2 F1 score and 68.2 EM score.

4 Approach

4.1 Vector Representation of Words

Each word in the question and the corresponding context paragraph are parametrized into multi-dimensional vectors using GloVe for word representations. For training data, each question and context vectors are concatenated into a single vector, and the output would be a tuple of two integers indicating where in the context paragraph the answer would lie.

From observing the distribution of the length of questions and context paragraphs from the SQuAD data, we found out that it diminishes almost completely after length of 30 and 400 words, respectively. Figures 1 and 2 shows the distributions of length. To use these data in a rNN model, the length of these vectors must be a fixed number to be processed by encoders and decoders, as discussed in the next section. Therefore, we fixed the length of question and context vectors to 30 and 400. All vectors longer would be cut off, and those shorter would be embedded with zeros. We are also using masks to prevent these values from being taken into account in training and testing.

As mentioned above, we are using GloVe model for representing the words as vectors. We can parametrize on the number of dimensions to represent the words. By default in the model, we will put the number of dimension as 100.

4.2 Encoder - Bi-Directional LSTM

For the input of the encoder, we are feeding in the embedded questions and the context paragraphs obtained by processing the database, and also supplying the corresponding masks. The model then feeds the embedded question vector into bidirectional dynamic rNN model using tensorflow. As default, we defined the model for Bi-rNN to have number of states as 200, and the size of dimension as 100. The output of the Bi-LSTM is two vectors from both forward and backward flow through the nodes. These two vectors are concatenated into a single vector. Thus, we are setting up the encoder so that it runs two Bi-LSTM on question and context paragraph embedding vectors. The output of the encoder is two vectors, one for each of the question and context paragraph vectors.

The idea behind using Bi-LSTM for both the question and the context paragraph is to have vectors that first read in the direction of question to context, and also that read the context and then the question. This would have an effect of reading the context paragraph conditioned on the question and vice versa.

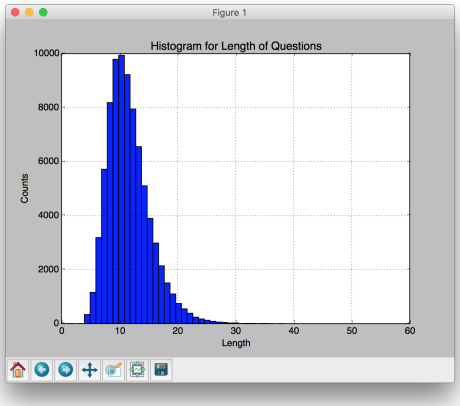


Figure 1: Length of questions

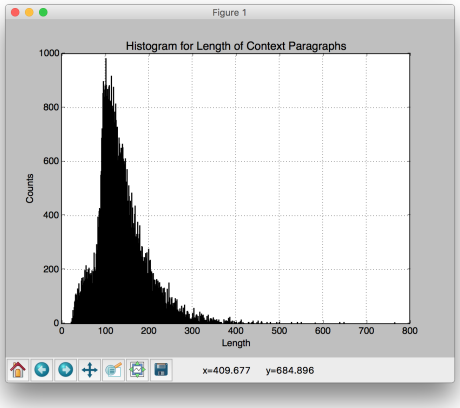


Figure 2: Length of context paragraphs

4.3 Decoder - LSTM

The decoder is run in minibatches of size of 100. It receives vectors for the question and the context paragraphs from the hidden layer of the encoder. It first processes the question vector by looking into the masks to determine the last node from the hidden layer. It then slices only the last node and tiles it by 400 to get a vector of dimensions (100, 400, 400) composed of the repeating last node of the question vector. This vector is then concatenated with the context vector from the hidden layer. The idea behind this is that the model can first read from the context paragraph and then the question, which has the effect of reading the context conditioned on the question to provide the answer.

Each question and context paragraph has different length. The model accounts for this by taking a logarithmic on the elements of the mask, which has 1s in the actual indices and 0s on the zero embeddings. The resulting vector would have a format $(0, 0, ..., -NaN, -NaN)$, where the values are 0 for the actual indices and $-NaN$ in others. The model then adds this vector to the concatenated vector above to account for the masking before calculating the probabilities of the output.

The resulting vector after the concatenation and the masking is fed into LSTM to predict the indices of the answer.

4.4 QASystem

4.4.1 Initial Values

The QA System has defined initial values as follows: the lengths of a question and a context paragraph are 400 and 30, respectively. The batch size is 100, and the dimension of a hidden state, which is used in the encoder, is 100. The dimension of a mask is equal to that of the batch size, 100, and is used with the embedding vectors of question and context paragraph in the encoder. The size of span is 2.

4.4.2 Loss Function

For the loss function of the QA System, we referenced *Bi-Directional Attention Flow for Machine Comprehension*[2], which declared the loss function as follows:

$$L(\theta) = -\frac{1}{N}\sum_{i}^{N}\log(p^1_{y^1_i}) + log(p^2_{y^2_i}) \tag{1}$$

where θ is the set of all trainable weights. N is the batch-size, which is 32. y^1_i and y^2_i are true start and end indices for the i-th entry, and p_a is the a-th entry for the vector p. For the probability vector it uses the output from the decoder. As the span size is 2, the loss is the negative sum of all the probability distribution for span of 2 divided by the batch size. For training we are using Minibatch Stochastic Gradient Descent, which will be discussed further in the next section.

4.4.3 Optimize Functions

For the optimizer, we started with Stochastic Gradient Descent, and looked for the optimal learning rate using exponential decay with the starter learning rate set as 0.5, decay step as 1000, decay rate as 0.99, and staircase set as false. In the experiments section, we will discuss which optimizer function yields the best results under which parameters.

The loss function implements minibatch SGD. The size of the minibatch is 32, and for each minibatch processed the global step is incremented by 1. After all minibatches are processed it prints out the resulting loss value and saves the model.

5 Experiments

5.1 Different Models Attempted

For the encoder, we first experimented with the baseline model, which had only a single one-directional LSTM. The LSTM model certainly showed decreasing loss function value after each iteration, but the convergence was really slow despite the adaptive learning rate and did not show good numbers even after one complete run of epoch. We decided we needed to implement a stronger model, and thus went on to implement multiple Bi-LSTM with our own configurations as follows.

First, in the encoder we implemented two separate Bi-LSTM architectures that each read the question and context paragraph vectors separately. This insured there would be no exploding or diminishing gradient issues that would have been prominent with the long vectors. We concatenated each of the two vectors from a single Bi-LSTM into one, producing total of two embedded vectors for question and context.

Next, the decoder was unique in processing the vectors it received from the encoder. From the mask vectors, it looked at where the last node of the question, sliced it from the rest, and tiled that last node to match the dimension of the hidden layer. Next, the tiled vector was concatenated with the context paragraph, which was then fed into LSTM to get the indices of the answer. The resulting probability matrix was used to find and evaluate the predicted answer.

5.2 Adaptive Learning Rate

One of the critical parameters in rNN model is the learning rate. A learning rate too high would hinder the model from converging to optima, and that too low would make it likely to fall into a local optima, unable to find the global optimum value. However, finding the appropriate learning rate via ablation tests was too costly given the number of hyper-parameters we had to train under the limited amount of time and resources. Therefore, in our model we implemented the adaptive learning rate, which started training with a relatively high value that diminished gradually as the number of iterated epoch grew.

The advantages of using adaptive learning rate is that in the early stages of training, when the parameter values are far from optimal, the model can move quickly to get near to the global optimum value. After sufficient number of iterations, the learning rate diminishes to allow the model to converge to the optimal value. With the massive database and multiple layers of rNN models were incorporating with, it was appropriate to use the adaptive learning rate for training.

5.3 Sequenced Attention Mix Model

We implemented sequenced attention mix model for our decoder. After the Bi-LSTMs in the encoder returned question and context vectors, we computed the context vector for Q←P and concatenated it with P. For computing the starting and ending indices, we used different weights with the concatenated matrices of PQ to generate two unnormalized probability distributions, which we used softmax function for normalizing. This yielded two probability vectors for each context, from which we chose the highest probability for starting and ending indices.

5.4 Match-LSTM

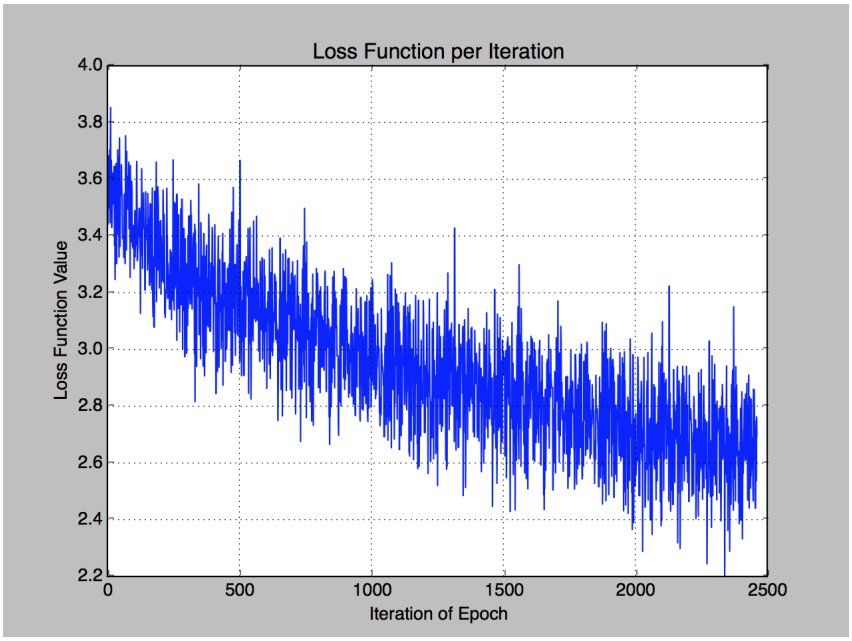
After our Bi-LSTM model yielded promising results, we moved on to implement another model in the decoder: Matching-LSTM. One of the reasons we wanted to implement Matching-LSTM is because the model can train on every word in the output from the Bi-LSTM encoder. The output vector from the encoder has dimensionality of 400. Each column, which corresponds to each word, would be multiplied by the entire question encoding matrix. After the resulting vector is reshaped to satisfy dimensionality, it would be fed into a forward-LSTM and backward-LSTM to predict the starting and the ending indices from the context paragraph.

One of the difficulties we faced is that we did not have our own custom rNN-cells, which prevented us from using the tensorflow library for the Match-LSTM. Therefore we tried using the basic LSTM cells with looping, but this did not go well due to the number of parameters we were passing in. The runtime cost was far too costly, and could not satisfy the time constraints we had.

6 Results

6.1 Quantitative Analysis

	Sanity	Dev	Test
F1 Score	31.777	29.747	30.231
EM Score	23.827	19.130	19.322



From the loss graph, the model follows a steady decline until iteration 2500, where it converges to around 2.6. This shows that while the performance in F1 and EM scores are not the best, the model has reached its limits in expressive power. Compared to other models like the logistic regression baseline by Rajpurkar[6], we fell short by around 20 F1 and 20 EM. Nevertheless, this model does perform moderately well when comparing its EM score to those of models with similar F1 scores, taken from the dev/test leaderboard. This result implies that most outputs are either exact matches, or totally off on the substring spanning. Below are some cases of output vs gold standard answers.

6.2 Qualitative Analysis

Case 1: Short Exact Match
Examples: *House of Commons, Leonardo Bruni, 3973 hectares , May 1967*
This represents the majority of EM contribution. Most items of this case are entity names like House of Commons and Learnardo Bruni, dates like May 1967, and adjective-noun pairings like 3973 hectares. We can interpret this as the model having learned basic grammatical components from training such that it knows not to split compound entities.

Case 2: Long Exact Match
Example: *cells of the offspring contain copies of the genes in their parents ' cells*
This is the rest of the EM outputs. Here, it probably results from overfitting to these specific training samples from a previous iteration as the answers do not constitute simple units learnable by this model's architecture.

Case 3: Partial Match
Examples: **Yagi-Uda** *antenna where the solid rod, (near) St. Sophias Cathedral*
This was about one-fifth of the none EM outputs. Here, most omissions or extra words are descriptions surrounding the answer span. Though sometimes short in length, these descriptions do change the meaning, e.g. missing the "near" for the second example changes location to object type.

Case 4: Complete Blunder
This is the rest of the non EM outputs. Here, the answers tend to require higher reading comprehension capabilities not possessed by this current model. Common answers tend to be long and multi-part in entities encompassed instead of single objects.

7 Future Steps

Current model in this paper does not have any character embedding layer. As suggested by the paper *Words or Characters? Fine-Grained Gating for Reading Comprehension*[5], it uses a fine-grained gating to combine the character-level representation with the word-level to create an embedding vector that incorporates both character and word. Another possible character embedding would be to have a LSTM layer on the characters that would then be fed into the word embedding layer.

This model requires the question and the context paragraph expected to hold the answer as inputs. Next step could also look at increasing the length of the context paragraph. In SQuAD the number of context paragraphs longer than 400 words diminish rapidly, and hence this model could not be tested on how long the paragraphs can be before the accuracy is affected. Bi-directional LSTM would take care of the diminishing or exploding gradient issues, and getting the right values of parameters would take careful adjusting and considerations as well. Hence testing the model with different datasets with Bi-LSTM would also yield interesting results as well.

8 Conclusion

The architecture of the model introduced in this paper uses GloVe vectors to produce word embedding vectors of the questions and context paragraphs found in SQuAD, which is then fed into the hidden layers of bi-directional rNN models of the encoder to produce vector representations of the data. In the decoder, the vectors are concatenated, and the resulting probability matrix is fed into the softmax to print the expected indices in the context paragraph.

References

[1] Hermann, K., Kočiský, T., Grefenstette, E. (2015) *Teaching Machines to Read and Comprehend*, Google DeepMind.

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

[3] Wang, Z., Mi, H., Hamza, W., Florian, R. (2016) *Multi-Perspective Context Matching for Machine Comprehension*, New York: IBM T.J. Watson Research Center.

[4] Pennington, J., Socher, R., Manning, C. *GloVe: Global Vectors for Word Representation* Stanford: Stanford University

[5] Yang, Z., Dhingra, B., Yuan, Y., Hu, J., Cohen, W., Salakhutdinov, R. *Words or Characters? Fine-Grained Gating for Reading Comprehension* Carnegie Mellon University

[6] Rajpurkar, P., Zhang J., Lopyrev K., and Liang, P. *SQuAD: 100,000+ questions for machine comprehension of text*. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, 2016.