Question Answering System using Dynamic Coattention Networks

Bojiong Ni, Adeline Wong, Zhiming Shi
Stanford University
Stanford, CA 94305
{bojiong, adelinew, zhimings}@stanford.edu

Abstract

We tackle the difficult problem of building a question answering system by building an end-to-end recurrent neural using network sequence-to-sequence model. We use the coattention encoder and explore three different decoders: linear, single layer maxout, and highway maxout network. We train and evaluate our model using the recently published Stanford Question and Answering Dataset (SQuAD). Out best result is achieved by using linear decoder with an F1 score of 54.93%.

1 Introduction

Machine comprehension of texts written in human natural language is one of the most important and fundamental areas of research in NLP. Recent research interests in this area focus on the task of question answering - the task of requiring a computer to answer questions based on provided texts.

In this paper, we build a question answering system and train our model using the recently published Stanford Question Answering Dataset[1], consisting of 100k+ human annotated context-question-answer triplets.

1.1 Problem Formulation

Let $\mathbf{X}^Q = (x_1^Q, x_2^Q, \ldots, x_n^Q)$ denote the sequence of words in the question, and $\mathbf{X}^P = (x_1^P, x_2^P, \ldots, x_m^P)$ denote the sequence of words in the context paragraph. We wish to predict a pairs of indices (a_s, a_e) pointing to the start and end positions in the context $\mathbf{X}^P$ as the answer to the question $\mathbf{X}^Q$.

1.2 Evaluation Metric

As in [1], performance of our question answering system is evaluated using two metrics: Exact Match (EM) and F1, which can be interpreted as a measure of overlap between the predicted answer and the true answer. Human performance on the tasks results in a EM of 77.0% and and F1 of 86.8%. The state-of-art question answering system (r-net (ensemble) from Microsoft Research Asia) has an EM of 76.922% and an F1 of 84.01%, which is at nearly human levels of performance[1]. [1] implemented a logistic regression baseline achieving 40.4% EM and 51.0% F1. We hope to beat this baseline with a deep learning approach.

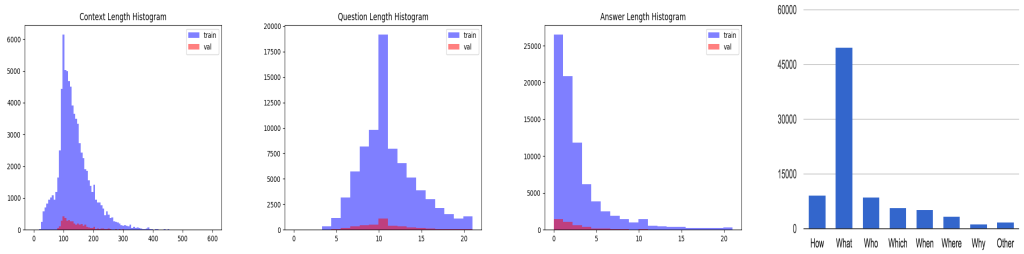
2 Dataset

We use the Stanford Question Answering Dataset (SQuAD)[1], which consists of about 100K human-generated questions and answers, based on text taken from Wikipedia articles. Questions

[1] As of Mar. 21 2017. See the Leaderboard for the latest results.

were generated for 23,215 paragraphs from 536 Wikipedia articles, which were randomly sampled from the top 10K articles in Wikipedia and assumed to be of high quality. The articles cover a variety of topics; and the questions reflect different types of reasoning. Rather than being completely human-generated, answers are phrases from the passages (i.e. a sequence of contiguous words). Figure 1 shows histograms summarizing some characteristics of this dataset. We shall discuss the implication of these characteristics on our model in the later sections.

We split the supplied 87k training dataset into two parts: 95% for training our model and 5% for validation purpose. Our evaluation is performed against a 10k dev set, in which three correct answers are given per question/context, and a prediction is considered correct if it matches any of them. The test set is hidden from the public for fairness in model evaluation. We report our final performance against this test set after submitting to the leaderboard.



(a) Context Length (b) Question Length (c) Answer Length (d) Question Type

Figure 1: Dataset Summary Histograms

3 Related Work

Early approaches tackling question answering problem typically include rule-based algorithms, structured knowledge base querying, multi-step linguistic analysis and heavy feature engineering such as named-entity recognition, semantic parsing and question classification. Recent attempts tackle this problem by building end-to-end neural network models.

3.1 Dynamic Coattention Networks

Xiong et al introduced this model in [2]. Their model captures the interaction between question and context paragraph using a coattentive encoder, and then predicts the start and end of the answer using a dynamic decoder built using highway maxout networks (HMN). They achieved an F1 score of 75.9%. We based our encoder largely on the coattention network, and we attempted to reproduce the result by building the dynamic decoder with HMN as one of our exploratory efforts with different decoders.

3.2 Match-LSTM

In [3], Wang and Jiang proposed a match-LSTM encoder, and two different decoding schemes to predict answers from the context: sequence model and boundary model. This model achieved an F1 score of 71.2. We initially attempted to build an encoder similar to match-LSTM, with a simple linear decoder. However, the result is subpar and we discontinued this attempt.

4 Method

We build an end-to-end sequence-to-sequence recurrent neural network. The high level architecture of the neural network can be divided into encoder, coattention fusion, and decocder. See Figure 2 for an overview.

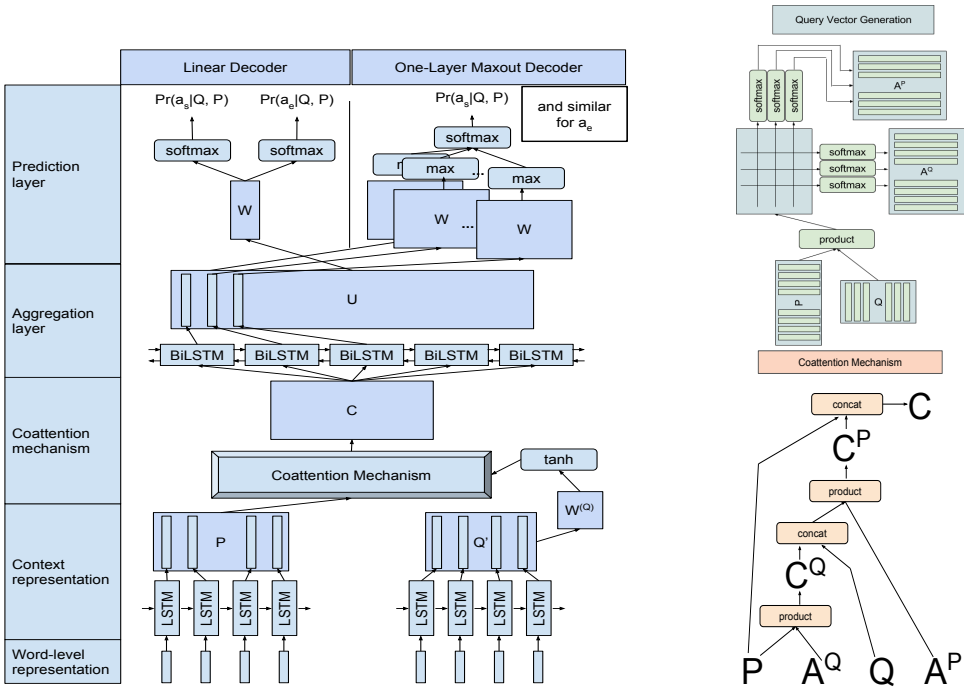


Figure 2: Detailed Network Design

4.1 Context and Question Encoder

For each context, we feed the gloVe embedding of each word into an LSTM to generate a context embedding.

For question encoding, to share the same word representation power with context, we feed the gloVe embedding of each word in the question into the same LSTM as the one for context. In addition, as suggested in [2], to allow variance in question and context encoding space, we further add a non-linear (tanh) transformation to the outputs of question LSTM encoding to generate the final question representation. Unlike [2], we do not append an additional column of sentinel vectors in either question or context encoding, as we did not observe an improvement on model performance after addition of the sentinel vectors.

4.2 Coattention Encoder

We base this entirely on [2]. Let the output matrices of the question LSTM encoding be $Q \in \mathbb{R}^{\ell \times n}$, whose column vectors are the output encoding corresponding to each word in the question sequence; and that of the context paragraph encoding be $P \in \mathbb{R}^{\ell \times m}$. We obtain the affinity matrix $L = P^T Q \in \mathbb{R}^{m \times n}$ whose entries consist of the affinity scores of all pairs of document words and question words. We then normalize the rows of the affinity matrix to obtain an attention matrix $A^Q = \text{softmax}(L) \in \mathbb{R}^{m \times n}$ whose rows are attention weights for each word in question corresponding to the document words (at the row); similarly, we normalize the columns of L to obtain another attention matrix $A^P = \text{softmax}(L^T) \in \mathbb{R}^{n \times m}$ whose rows are attention weights for each word in context across the question. Therefore, the product $C^Q = PA^Q \in \mathbb{R}^{\ell \times n}$ has the interpretation of the summary of context with attention to each words in question; the product QA^P has the corresponding interpretation of summary of question attending to words in context. We concatenate that with the attention-over-attention summary $C^Q A^P$ (projection of the previous question attention based summary to context attention space) to obtain the final coattention representation question and context: $C^D = [Q; C^Q]A^P \in \mathbb{R}^{2\ell \times m}$. This, concatenated with context encoding again is fed into a bidirectional LSTM to output the final encodings. This scheme is illustrated in detail in Figure 2.

4.3 Decoder

We explored three different methods of implementing decoders, single layer linear decoder, single layer max-out decoder and the highway max-out decoder presented in [2] papar.

4.3.1 Single Layer Linear Decoder

The first method we tried is to project the output vector $u_i \in U$ of Coattention Encoder at each context index i into a single value α using linear transformation. We can think of this value as the confidence score of index i being the start index. The index a_s with the confidence score get selected as the prediction for answer start position.

$$\alpha = W_s U + b_s$$
$$a_s = argmax([\alpha_0, \alpha_1, ..., \alpha_n])$$

We apply the same mechanism for answer end index a_e predictions. The prediction for start and end positions are applied independently. I.e. the linear transformation matrix and bias parameters are not shared between start index and end index prediction.

$$\beta = W_e U + b_e$$
$$a_e = argmax([\beta_0, \beta_1, ..., \beta_n])$$

For training efficiency, we operates each training step in batches and vectorize all operations. A key caveat here is that vectorized operations requires each batch to be padded to the same context length and but contexts in each batch may have different lengths. Thus we need to find a way to inform the model explicitly not to predit start and end indices outside the context actual length. We introduces exponential masking to solve the problem. For example, if a context has length l and the max context length in the batch is l_{max}, where $l_{max} > l$, we add -10^{30} to each of the α_i where $i > l$. I.e.

$$a_s = argmax([\alpha_0, \alpha_1, ...\alpha_l - 10^{30}, \alpha_{l+1} - 10^{30}\alpha_n - 10^{30}])$$
$$a_e = argmax([\beta_0, \beta_1, ...\beta_l - 10^{30}, .\beta_{l+1} - 10^{30}\beta_n - 10^{30}])$$

By using this technique, we can prevent predict our of range index without passingh the context length information into the hidden parameters.

In order to calculate the loss function, we then apply softmax normalization and cross entropy.

4.3.2 Single Layer Max-out Decoder

As suggested in [2] paper, there could be different types of questions in the SQuAD data set. We did an analysis and found that the questions can be divided into the following types:*what, who, which, why, when, where, how, others*. The distribution of these question types is illustrated in Figure 1. One intuitive way to address this characteristic is to generate a confidence score for each type of question.

We can think of the problem statement as, given question type T, what is the confidence for context position i to be the start / end index. Thus at each context index position, we have eight different confidence scores. We choose the highest score from the eight as the final confidence score α / β for position i:

$$\alpha_i = max(Score\_start_{ij})$$
$$\beta_i = max(Score\_end_{ij})$$

where $j \in \{1, 2, ..., 8\}$.

We then apply the same trick of exponential masking as introduced in Single Layer Linear Decoder section to prevent model choosing out of bound indices. The loss function is also computed same way to α and β as the Single Layer Linear Decoder.

4.3.3 Highway Max-out Network

In addition to the above to different decoders, we also tried the Highway Max-out Network (referred as HMN below) decoder introduced by [2] paper. This is an iterative method to predict multiple pairs of start and end indices for each context. At each iteration i, the coatnetion vector corresponding to the prediction of previous iteration $i-1$ will be fed into an LSTM:

$$h_i = LSTM(h_{i-1}, [u_{si-1}, u_{ei-1}])$$

The result of LSTM will be further feed into the HMN together with coattention vectors for previous prediction of start and end index u_{si-1}, u_{ei-1} and the coattention vector per context position u_t to give current estimate of α and β for current iteration.

$$\alpha_i = HMN_\alpha(h_i, u_t, u_{si-1}, u_{ei-1})$$
$$\beta_i = HMN_\beta(h_i, u_t, u_{si-1}, u_{ei-1})$$

Then the same $argmax$ method is used to select the current prediction of start and end index. If the current prediction is the same as the previous iteration, the system short cuts and output the current prediction as final output. Otherwise, it continues to iterate until the max number of iteration is reached. The rational here is that for a given question, multiple answer candidates can appear in the context, the model shall pick the best guess. The paper does not specify what is the initial start and end index fed into the iterative method. Since our single layer linear decoder gives ¿50% F1 result, we decided to use it as an initial guess for the iterative method.

The structure of HMN is the following. These equatioins are referenced from [2] paper:

$$r = tanh(W^{(D)}[h_i; u_{si-1}; u_{ei-1}]) \tag{1}$$
$$m_t^{(1)} = max(W^{(1)}[u_t; r] + b^{(1)}) \tag{2}$$
$$m_t^{(2)} = max(W^{(2)}m_t^{(1)} + b^{(2)}) \tag{3}$$
$$HMN_{output} = max(W^3[m_t^{(1)}; m_t^{(2)}] + b^{(3)}) \tag{4}$$

The rationale for using max-out network is the same as the rationale described in the single layer max-out decoder part: to capture different question and context types. One important difference here is that [2] paper's method uses equation (2) to project h_i, u_{si-1}, u_{ei-1} into the same vector space and use $tanh$ function to normalize the scale.

5 Experiments

5.1 Implementation and Hyperparameter Tuning

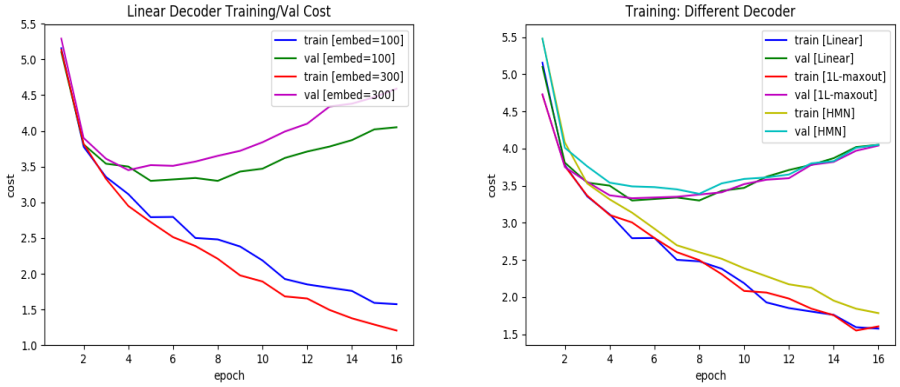
From observation, we found that the model's training loss will continue to decrease ove epochs, but its validation cost will first decrease together with the training loss then starts to increase (Figure (3)). This indicates our model starts to overfit the training data set. Thus, our best model is chosen for the one with lowest validation cost.

To alleviate the overfiting problem, we used dropout rate of 0.15 for single layer linear and maxout decoder model, 0.2 for HMN decoder model.

When model approaches the (local) minima, it needs to move with smaller step to further get cloase to (local) minima. Therefore we started our learning rate from 0.001 and used annealing steps of 5000 for linear and max-out decoder model, 3000 for HMN decoder model. The exponential backoff rate is 0.96.

Given the deep structure of LSTM, gradient vanishing / explosion could be a problem, so we used gradient clipping technique to limit the maximum global gradient norm to 5. However, during experiments, the global gradient norm rarely reaches 5; nor does it vanishing. It is stable between 2 and 4. This could be the result of using Adam optimizer, where gradient vanishing / explosion problem has alleviated.

Our hidden state size is 200 for all LSTM. The pool size is 16, maximum iteration is two for HMN decoder.



(a) Effect of embedding

(b) Effect of decoder

Figure 3: Learning Curve

5.2 Results and Evaluation

The evaluation results of our various models are tabulated below:

Decoder	Embedding	EM		F1	
		Val	Dev	Val	Dev
Human		80.3	77.0	90.5	86.8
Random Guess		1.1	1.3	4.1	4.3
Logistic Regression		40.3	40.4	51.0	51.0
Linear	100	41.0	43.4	55.1	54.5
Linear	300	34.0	39.1	47.8	50.3
1-Layer Maxout	100	48.0	41.4	55.3	52.9
Linear + HMN	100	48.0	43.0	58.1	54.6
Linear + HMN	300	45.0	42.1	60.0	53.1

The first half of the table is the baseline reported in [1]. After submitting to leaderboard, our best performance is achieved by the model with linear decoder using 100 dimensional word embeddings, with an F1 score of 54.93 and EM of 44.40.

5.3 Analysis and Discussion

From the above, we can see that the results are comparable between different implementation of decoders and embeding sizes. Our single layer linear decoder with 100 gloVe embedding works best among all different decoders and embeddings.

5.3.1 Softmax Output analysis

We first examine how certain our system is of its predictions by manually inspecting a sample of predictions in heat map Figure 4. The first four examples are correct, and the last is incorrect. In the first example, all human answers are the same, and the system also makes a single strong prediction for start and end. This is the most common case. The second example is similar, but what is interesting is that the context paragraph discusses various engine configurations and in multiple places mentions 90° angles, yet the system is able to select the right answer of 180°. The question happens to begin, "In a 4-cylinder compound engine," which suggests that our attention mechanism is working admirably at locating relevant information. In the third example, the system has trouble displays some uncertainty about whether adjacent punctuation should be part of the answer. Postprocessing of answers to eliminate unmatched paired punctuation may be of some help. In the fourth question, humans disagree over whether the answer should be "the Gaulish name Rënos," "Gaulish name Rënos," or simply "Rënos," and the system is about equally uncertain about the first two options. Probably human respondents are themselves inconsistent about including or excluding

”the” in the training data. In the last case, there are two loci that the system thinks are possible answers, but unfortunately, the system combines them into an extremely long span. The question asks where Nikola Tesla lived most of his adult life, and the predicted answer ”finance” is definitely implausible. This exemplifies inadequacy in our neural network model, most likely the decoder.

Figure 4: Sampled Softmax Output. White and red correspond to strong and weak predictions, resp., with yellow in between. For each example, the left column is the start index prediction, the right is for the end index. The rightmost prediction is the only incorrect one.

5.3.2 Single Layer Linear decoder

Although the single layer linear decoder achieves non-trivial performance as shown above, it may still be too simple in struccture to capture the contextual information. One common error is that our predicted end index is smaller than the predicted start index. This is possibly due to the fact that the two indices are predicted independently. Feeding the start index prediction into the end index prediction may leads to better performance in eliminating such errors.

5.3.3 Single Layer Max-out decoder

One possible reason of our single layer max-out decoder not performing better could be that we didn't normalize the confidence scores for each type question before doing max-out. For example, the confidence score given question type *why* can be at range of $[0, 100]$ but the confidence scores for type *what* can be at range of $[100, 200]$, thus we will always choose the most confident position prediction for type *why*. One possible improvement is to do a softmax cross all position for each question type before max-out operation.

5.3.4 Highway Max-out Network

Our highway max-out network implementation also does not work as good as [2] paper described. There are several hypothesis for this result:

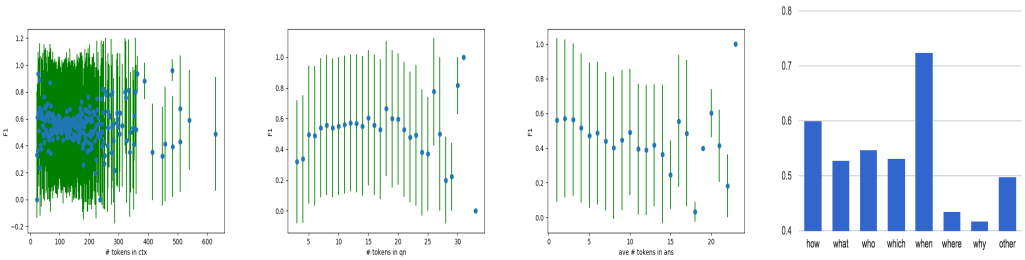
- Pool size and maximum iteration too small. [2] mentions that empirically the model's performance consistently improves with deeper structure when the maximum number of iterations increases. In our implemntation, due to runtime concern, we limited our maximum iteration to two. For the same reason, we limited our pool size to four. The combination of these two detour from [2] might be a reason for performance degrade.
- Initial linear decoder guess masks out the ability for the iterative methods to guess a better answer. The linear decoder has reasonable performance of over 50 F1. This may initiate the iterative process from a local minimum and make it harder to walk out to find the global minimum.
- Not truncating the longer sentence. The paper truncates the max context length to 600, but we kept all context at its original length. Our hypothesis was that the exponential masking

shall handle the case of out of boundary prediction. However, it may have deeper impact on hidden parameter convergence.

- Our code may contain some bug leads to lower performance.

5.3.5 Breakdown of F1 distribution

Despite our less-than-satisfactory model performance, we are able to reproduce some meaningful analysis results also reported in other attempts on SQuad ([2], [3]). We breakdown the F1 distribution according to the length of context, question, average answer length, and question type, as in Figure 5. Note that in the first 3 plots, the error bar is the standard deviation of the F1 scores. We could conclude that the performance of sequence-to-sequence model gets worse as the length of any of the text sequence increases. Moreover, some semantic meaning is harder to capture with such model, as we see that the *why* question is the hardest to answer correctly.



(a) Context Length (b) Question Length (c) Answer Length (d) Question Type

Figure 5: Breakdown of F1 distribution

6 Conclusion and Future Work

We implemented the sequence-to-sequence model by building the coattention encoder from [2], and explored different decoders such as linear, 1-layer maxout, and highway maxout. Our best result is achieved by the network with linear decoder with 54.93 F1 score.

Clearly our implementation of the decoder has a large room of improvement. Given more time, we would like to attempt more decoder architectures, or correctly implement the highway maxout decoder.

Further more, we'd like to do more control-variable experiments to study the effect and contribution of each component of our architecture. For example, the effect of different GloVe embedding, or some of the architecture design choice like LSTM hidden layer size, decoder iteration number, etc. This would gain more insights to the functioning of the model.

By analyzing errors, improved pre- or post-processing or augmenting the system with some "common sense" reasoning may be useful for improving performance.

References

[1] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. CoRR, abs/1606.05250, 2016.

[2] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. arXiv preprint arXiv:1611.01604, 2016.

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

[4] Yiming Cui, Zhipeng Chen, Si Wei, Shijin Wang, Ting Liu, and Guoping Hu. Attention-overattention neural networks for reading comprehension. arXiv preprint arXiv:1607.04423, 2016.