Question Answering with Attention and Boundary Pointers

Tal Stramer
Department of Computer Science
Stanford University
Stanford, CA 94503
tstarmer@stanford.edu

Abstract

Answering questions automatically based on context has been a long-standing problem in Machine Learning and Natural Language Processing. In recent years, end-to-end neural architectures have led to new, powerful models that achieve ground-breaking performance on question answering tasks. The Stanford Question Answering Dataset (SQuAD) provides a dataset of real questions and answers. In this paper, we build an end-to-end neural network architecture for the SQuAD task that combines ideas from several recently proposed architectures: Match-LSTM, Dynamic Co-attention, and Pointer Networks.

1 Introduction

Text comprehension has been central to many areas of research in Natural Language Processing, such as machine translation, sentiment analysis, entity extraction, co-reference resolution, and question answering. Until recently, state-of-the-art models to solve these tasks has required building large systems with many hand-engineering features and modules. In recent years, neural architectures have been shown to be highly successful at solving problems involving text comprehension with little to no manual feature-engineering.

One such task, question answering, can be formulated and evaluated in many ways. The SQuAD dataset provides one such formulation. Specifically, it provide a dataset of (question, context, answers) examples, where a correct answer to the question is a contiguous span of text within the context. The task is then to pick one of the correct answer spans given the question and context. The correct answer spans for a given question and context were crowd-sourced and so are of high quality.

Text comprehension algorithms using neural networks are usually based on recurrent neural networks (RNNs). Specifically, sequence-to-sequence models were first introduced by Sutskever et al. [1] which use an encoder and decoder RNN to solve the problem of machine translation with variable-length source and target language sentences. These models commonly use Long Short-Term Memory (LSTM) cells [2] in order to model long-term dependencies in sentences.

Based on sequence-to-sequence models, Wang et al. [3] introduced a new model for the SQuAD task. Specifically, they propose the following approach: 1) Encode the question and context independently using a RNN with LSTM cells, using Glove word embeddings to represent each word 2) build a question-aware representation of the context using the Match LSTM algorithm and 3) decode the context representation into pointers to the correct answer within the context using a Pointer Network. In the next section, we describe the Match-LSTM algorithm and Pointer Networks.

Xiong et al. [4] use a similar approach, but instead of using Match-LSTM they use a dynamic co-attention mechanism in order to create a representation of the question and context that captures

interactions between the two, which they run a bidirectional LSTM through to encode additional temporal information. To decode the answer spans, they use a LSTM-based state machine to iteratively alternate between predicting the start span and end span, which helps recover from local maxima predictions of the span.

Building on these ideas, we experiment with combining ideas proposed by Wang et al. [3] and Xiong et al. [4]. Specifically, we experiment with using the dynamic co-attention mechanism along with answer pointers. We hypothesis that DCN will perform better than Match-LSTM due to combining both a question to context representation and a context to question representation. We also experiment with different training sources and sizes for word embeddings.

2 Method

At a high-level, we build a network that receives as input a question and context, represented as a list of ids corresponding to indices in a vocabulary list, and output two numbers, a start index and end index, representing the start and end indices in the context paragraph corresponding to the answer. In the following sub-sections, we describe the various layers of the neural network.

2.1 Embeddings Layer

Each word id in the question and context is converted to a word embedding vector. We initialize each embedding vector using pre-trained GloVe [5] vectors. Let Q be the representation of the question and P be the representation of the context, where $Q \in R^{dxQs}$ and $P \in R^{dxPs}$, Qs is the number of tokens in the question, Ps is the number of tokens in the passage, and d is the size of an embedding vector.

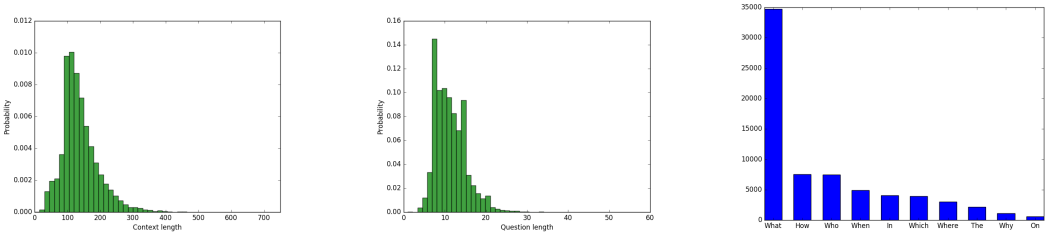
2.2 Encoder Layer

Each question and context is encoded using a bidirectional LSTM. Specifically, let H^p and H^q be defined as follows: $H^p = biLSTM(P)$, $H^q = biLSTM(Q)$, where $H^p \in R^{2lxPs}$, $H^q \in R^{2lxQs}$, and l is the length of a hidden vector outputted by the LSTM. Note that the vectors for each context word from the forward and backwards layers of the LSTM are concatenated to form the vectors in H^p and H^q.

2.3 Attention Layer

We experiment with two different attention layers:

1. Match LSTM: Originally proposed in Wang et al. [6], it's goal is to output a premise-aware representation of a hypothesis by performing word-by-word matching between the two. For the case of the SQuAD task the premise and hypothesis are the the question and context, respectively. It works by executing an LSTM over the context vectors, and at each step using an attention mechanism to obtain weights over the question vectors based on the current context vector and current hidden state of the LSTM. Then, it concatenates the weighted representation of the question vectors with the current context vector and feeds it into another LSTM, called the Match LSTM. The output of this LSTM represents our question-aware encoding of the context. Note that the question and context vectors in this case are H^q and H^p.
2. Dynamic Coattention: Originally proposed in Xiong et al. [4], it similarly has a goal of producing a question-aware representation of the context. It works by attending to the question and context simultaneously. To do so, it first creates an affinity matrix containing affinity scores for all pairs of words in the context and question. It then computes summaries of the question and context by weighting them based on the affinity matrix L, and combines them to form the co-dependent representation of the question and context. Finally, the representation is passed through a bi-directional LSTM. One potential advantage of this approach over Match LSTM is that it directly combines both a context-aware question representation and a question-aware context representation.



(a) SQuAD context lengths **(b)** SQuAD question lengths **(c)** SQuAD question types

Figure 1: Distribution of context and question lengths (a-b) and distribution of top 10 questions by first word in question (c)

2.4 Prediction Layer

For the prediction layer, we build a pointer network as described by Vinyals et al. [7]. The idea is to use an attention mechanism to select which word vectors in the context to select. Our input into this layer is the question-aware representation of the context from the previous layer. In a pointer network, one can predict either a sequence of symbols from the input sequence or just the boundaries of the sequence. We choose to predict the boundaries only, as it produced better results in our experiments. The output of the pointer network are two vectors β_0 and β_1, where $\beta_{0,i}$ represents the probability that the start of the answer span is at index i, and $\beta_{1,i}$ represents the probability that the answer span is at index i.

During training, we use cross-entropy loss to maximize the probability of seeing the given training start and end indices. During prediction, we first use the network to get β_0 and β_1 and then perform a linear search through the two arrays to compute the most probable start and end indices, enforcing that the end index is at least is large as the start index.

3 Experiments

3.1 Data

Our model is evaluated using the SQuAD dataset, which consists of 81,381 (question, context, answer) pairs used for training, and 4,284 for validation. There is also a held-out development set of 10,570 (question, answer) pairs for testing, as well as a hidden test set on CodaLab which determines our final competitive score. We use a simple word tokenizer based on a regular expression to tokenize the question and contexts. We experimented with using the Stanford tokenizer instead, which is a more sophisticated tokenizer model trained based on the Penn-Treebank corpora, but found it made no discernible difference in performance. We experiment with two word embeddings sources: pre-trained Glove vectors based on Wikipedia 2014 and Gigaword 5, which consists of 6 billion tokens and 400 thousand vocabulary size, and pre-trained Glove vectors based on Common Crawl, which consists of 840 billion tokens and 2.2 million vocabulary size. For out-of-vocabulary words, we simply initialize the vectors randomly. Note that the vocabulary consists of all words from the training, validation, and development set, but not the hidden test set on CodaLab. Figure 1 shows the the distribution of context / question lengths and question types in the training set.

3.2 Parameters

We experiment with various configuration of the parameters as summarized below.

- Word embedding size: 100, 200, and 300
- Word embedding source: Wikipedia / Gigaword, Common Crawl.
- Word embeddings parameter type: fixed, model variable
- Hidden state size: 100, 200 and 300
- Learning rate: Fixed at .0007 (based on manual tuning)

Table 1: Summary of F1 and EM scores on the development set for various configurations of parameters

Word Embed Size	Word Embed Source	Hidden State Size	Attention Encoder	F1 Score	EM
100	Wiki	100	Match LSTM	41.7	32.8
200	Wiki	200	Match LSTM	55.3	38.5
200	Wiki	200	DCN	54.4	36.4
300	Wiki	300	Match LSTM	59.4	45.9
300	Common	200	Match LSTM	59.2	44.3
300	**Common**	**300**	**Match LSTM**	**66.4**	**55.7**
300	Common	300	DCN	62.7	51.0

- Dropout rate: Fixed at .3 (based on manual tuning)
- Optimizer: ADAM optimizer [8]
- Number of epochs: Fixed at 10

3.3 Evaluation

We evaluate our model using two standard metrics: F1 and exact match (EM). Exact match measures the percentage of predictions that match the ground truth answers exactly, and F1 measures the average overlap between the prediction and ground truth answers.

After each epoch during training, we compute the F1 and EM scores of 100 random (question, answer) pairs from the validation set. We then choose the model from the epoch corresponding to the best F1 score on the validation samples.

3.4 Results

Table 1 shows the F1 and EM scores on the development set for models trained based on various configurations of our hyper-parameters using word embeddings fixed to their pre-trained Glove values. Our best F1 and EM scores on the development set are 66.4 and 55.7, respectively. This corresponds to a model that uses a word embedding size of 300, state size of 300, Glove vectors trained on Common Crawl, and Match LSTM for the attention layer. Our best F1 and EM scores on the hidden test set are 57.06 and 44.69, respectively, using the same model. One potential explanation for the drop in performance between development and test set is out-of-vocabulary words on the test set.

We observe only a small drop in performance when using dynamic co-attention for the attention layer, which may simply be noise. In general, we do not observe a big difference between the two, as both encode a question-aware representation of the context in a similar way.

The use of Common Crawl Glove vectors is especially helpful. We see a 11.7% increase in F1 score and a 21.3% increase in EM score using Common Crawl versus Wikipedia / Gigaword (fixing the word embedding size and hidden state size to 300).

We further analyze the effect of making the word embedding vectors variables in the model rather than fixing them. We run this experiment on our best model, which has a word embedding size and hidden state size of 300 and Glove vectors trained on Common Crawl. Note that when fixing the word embedding vectors our model has 5,317,501 parameters, and when making them variables it has 42,309,001 parameters, so it greatly increases the number of parameters in our model to make them variables. Figure 2 shows the F1 scores on the training and validation set for both of these models after each epoch. The best F1 score achieved on the validation set for the model with variable word embeddings is 64.91 (epoch 2), and is 66.4 for the model with fixed word embeddings (epoch 5). Figure 2 makes clear that the model with variable word embeddings is much more prone to overfitting as can be seen by the fact that the training set and validation set performance diverges for this model sooner and more sharply then for the model with fixed word embeddings.

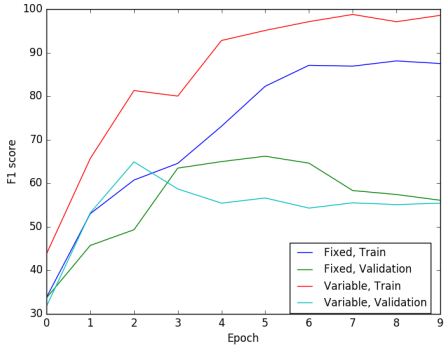
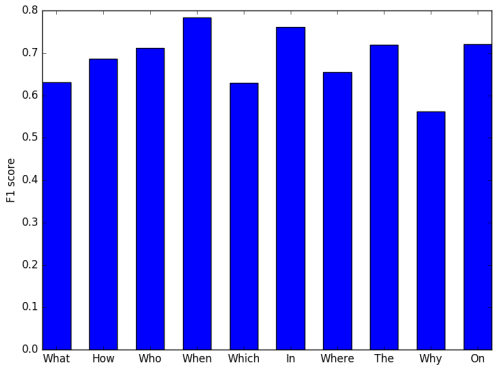
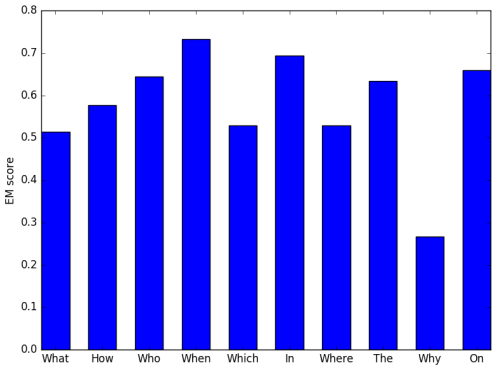


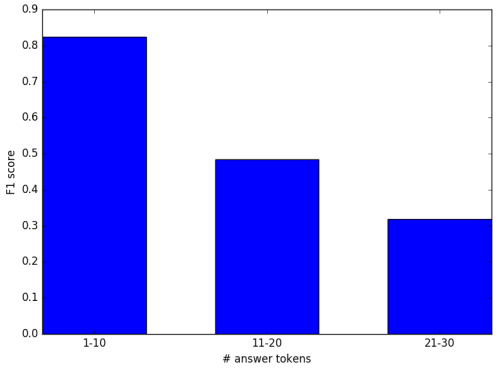
Figure 2: F1 scores on the training and validation set for fixed and variable word embeddings



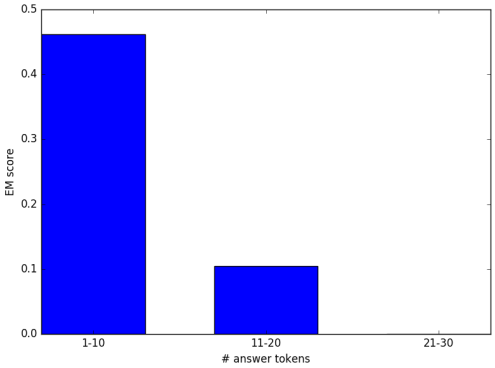
(a) F1 score by question type



(b) EM score by question type



(c) F1 score by # answer tokens



(d) EM score by # answer tokens

Figure 3: Break-down of F1 and EM scores by question type and number of answer tokens

3.5 Analysis

To understand where our model is making mistakes, we further break down the F1 and EM scores by answer length and question type, as shown in figure 3.

Figure 3.a and 3.b show that some questions are much easier to answer than others. For example, the F1 score for questions starting with "When" is ~8, while the F1 score for questions starting with "What" is ~6, a 33% difference. We hypothesize that this is due to the fact that questions starting with the word "What" are more ambiguous / general. For example, the SQuAD dataset contains the following two questions: "What was South Vietnam called in 1955?" and "What three things are Pentecostals committed to?". The first is asking for a single answer while the second is actually asking for a list of things as a response.

Figure 3.c and 3.d show that prediction accuracy is highly dependent on answer length. When the number of tokens in the answer is between 1 and 10, the average F1 score is almost .8, but when it is 11-20 tokens it drops below .5, and between 21-30 tokens to almost .3. This intuitively makes sense, as a longer answer requires a more complex model to be able to comprehend the greater amount of tokens in the answer and its relationship to the question. Figure 3 overall indicates potentially fruitful places to look for improvements to the model.

4 Conclusions

In this paper, we developed a neural attention model capable of answering questions from the SQuAD dataset. Our model is built based on word embeddings, Match LSTM, and Pointer Networks. We explored various approaches to building the model, and found that various parameters make a large difference in performance, such as state size and word embedding source. We achieved a F1 score of 66.4 and an EM score of 55.7 on a development set, which is much better than a logistic regression baseline model with many hand-engineered features, which has a F1 score of 51 and EM score of 40.4. EM. Our main take-away is that a relatively simple neural architecture can achieve high performance on question-answering task using SQuAD.

In the future, we would like to integrate character-level embeddings into word representations which will help with out-of-vocabulary words on the test set, and train an ensemble model rather than using the model from just one training run. In addition, we would like to look further into the breakdown of performance by question type and # answer tokens, which may lead to insights into better architectures for our model.

5 References

[1] Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." Advances in neural information processing systems. 2014.

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

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

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

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

[6] Wang, Shuohang, and Jing Jiang. "Learning natural language inference with LSTM." arXiv preprint arXiv:1512.08849 (2015).

[7] Vinyals, Oriol, Meire Fortunato, and Navdeep Jaitly. "Pointer networks." Advances in Neural Information Processing Systems. 2015.

[8] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).