Machine Comprehension Using Match-LSTM and Answer Pointer

Shuohang Wang
School of Information Systems
Singapore Management University
shwang.2014@phdis.smu.edu.sg

Jing Jiang
School of Information Systems
Singapore Management University
jingjiang@smu.edu.sg

Abstract

Machine comprehension of text is an important problem in natural language processing. A recently released dataset, the Stanford Question Answering Dataset (SQuAD), offers a large number of real questions and their answers created by humans through crowdsourcing. SQuAD provides a challenging testbed for evaluating machine comprehension algorithms, partly because compared with previous datasets, in SQuAD the answers do not come from a small set of candidate answers and they have variable lengths. We propose an end-to-end neural architecture for the task. The architecture is based on match-LSTM, a model we proposed previously for textual entailment, and Pointer Net, a sequence-to-sequence model proposed by Vinyals et al. (2015) to constrain the output tokens to be from the input sequences. We propose two ways of using Pointer Net for our task. Our experiments show that both of our two models substantially outperform the best results obtained by Rajpurkar et al. (2016) using logistic regression and manually crafted features.

1 Introduction

Machine comprehension of text is one of the ultimate goals of natural language processing. While the ability of a machine to understand text can be assessed in many different ways, in recent years, several benchmark datasets have been created to focus on answering questions as a way to evaluate machine comprehension (Richardson et al., 2013; Hermann et al., 2015; Hill et al., 2016; Weston et al., 2016; Rajpurkar et al., 2016). In this setup, typically the machine is first presented with a piece of text such as a news article or a story. The machine is then expected to answer one or multiple questions related to the text.

Different benchmark datasets define the task in slightly different ways. In some datasets, a question is a multiple choice question, whose correct answer is to be chosen from a set of provided candidate answers (Richardson et al., 2013; Hill et al., 2016). Alternatively, in some other datasets, a question does not provide candidate answers and the machine is expected to answer the question using any token or sequence of tokens from the given text (Hermann et al., 2015; Rajpurkar et al., 2016). Presumably, questions without given candidate answers are more challenging because the candidate answer set essentially includes all possible tokens or token sequences from the text and is thus much larger. Furthermore, questions whose answers span multiple tokens are more challenging than those with single-token answers. The Stanford Question Answering Dataset (SQuAD) introduced recently by Rajpurkar et al. (2016) contains such challenging questions, i.e., there are no candidate answers provided and a correct answer to a question can be any sequence of tokens from the given text. Moreover, unlike some other datasets whose questions and answers were created automatically in Cloze style (Hermann et al., 2015; Hill et al., 2016), the questions and answers in SQuAD were created by humans through crowdsourcing, which makes the dataset more realistic. Given these advantages of the SQuAD dataset, in this paper, we focus on this new dataset to study machine comprehension of text. A sample piece of text and three of its associated questions are shown in Table 1.

Traditional solutions to this kind of question answering tasks rely on NLP pipelines that involve multiple steps of linguistic analyses and feature engineering, including syntactic parsing, named entity

In 1870, Tesla moved to Karlovac, **to attend school at the Higher Real Gymnasium**, where he was profoundly influenced by a math teacher **Martin Sekulić**. The classes were held in **German**, as it was a school within the Austro-Hungarian Military Frontier. Tesla was able to perform integral calculus in his head, which prompted his teachers to believe that he was cheating. He finished a four-year term in three years, graduating in 1873.	
1. In what language were the classes given?	German
2. Who was Tesla's main influence in Karlovac?	Martin Sekulić
3. Why did Tesla go to Karlovac?	attend school at the Higher Real Gymnasium

Table 1: A paragraph from Wikipedia and three associated questions together with their answers, taken from the SQuAD dataset. The tokens in bold in the paragraph are our predicted answers while the texts next to the questions are the ground truth answers.

recognition, question classification, semantic parsing, etc. Recently, with the advances of applying neural network models in NLP, there has been much interest in building end-to-end neural architectures for various NLP tasks, including several pieces of work on machine comprehension (Hermann et al., 2015; Hill et al., 2016; Yin et al., 2016; Kadlec et al., 2016; Cui et al., 2016). However, given the properties of previous machine comprehension datasets, existing end-to-end neural architectures for the task either rely on the candidate answers (Hill et al., 2016; Yin et al., 2016) or assume that the answer is a single token (Hermann et al., 2015; Kadlec et al., 2016; Cui et al., 2016), which make these methods unsuitable for the SQuAD dataset. In this paper, we propose a new end-to-end neural architecture to address the machine comprehension problem as defined in the SQuAD dataset.

Specifically, observing that in the SQuAD dataset many questions are paraphrases of sentences from the original text, we adopt a match-LSTM model that we developed earlier for textual entailment (Wang & Jiang, 2016). We further adopt the Pointer Net (Ptr-Net) model developed by Vinyals et al. (2015), which enables the predictions of tokens from the input sequence only rather than from a larger fixed vocabulary and thus allows us to generate answers that consist of multiple tokens from the original text. We propose two ways to apply the Ptr-Net model for our task. Experiments on the SQuAD dataset show that our two models both outperform the best performance reported by Rajpurkar et al. (2016).

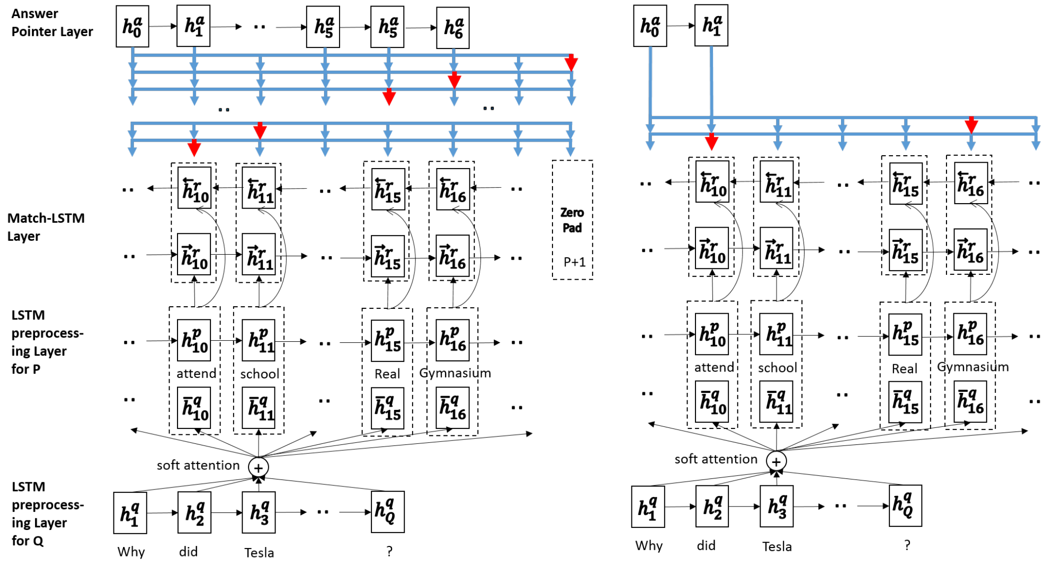
Our contributions can be summarized as follows: (1) We propose two new end-to-end neural network models for machine comprehension, which combine match-LSTM and Ptr-Net to handle the special properties of the SQuAD dataset. (2) We have achieved the state-of-the-art performance of an exact match score of 59.5% and an F1 score of 70.3% on the unseen test dataset. (3) Our further analyses with the models reveal some useful insights for further improving the method.

2 METHOD

In this section, we first briefly review match-LSTM and Pointer Net. These two pieces of existing work lay the foundation of our method. We then present our end-to-end neural architecture for machine comprehension.

2.1 MATCH-LSTM

In a recent work on learning natural language inference, we proposed a match-LSTM model for predicting textual entailment (Wang & Jiang, 2016). In textual entailment, two sentences are given where one is a premise and the other is a hypothesis. To predict whether the premise entails the hypothesis, the match-LSTM model goes through the tokens of the hypothesis sequentially. At each position of the hypothesis, attention mechanism is used to obtain a weighted vector representation of the premise. This weighted premise is then to be combined with a vector representation of the current token of the hypothesis and fed into an LSTM, which we call the match-LSTM. The match-LSTM essentially sequentially aggregates the matching of the attention-weighted premise to each token of the hypothesis and uses the aggregated matching result to make a final prediction.


Answer Pointer Layer
h_0^a
h_1^a
h_5^a
h_6^a
Match-LSTM Layer
$\overleftarrow{h}_{10}^r$
$\overleftarrow{h}_{11}^r$
$\overleftarrow{h}_{15}^r$
$\overleftarrow{h}_{16}^r$
$\overrightarrow{h}_{10}^r$
$\overrightarrow{h}_{11}^r$
$\overrightarrow{h}_{15}^r$
$\overrightarrow{h}_{16}^r$
Zero Pad
P+1
LSTM preprocessing Layer for P
h_{10}^p
h_{11}^p
h_{15}^p
h_{16}^p
attend
school
Real
Gymnasium
$\bar{h}_{10}^q$
$\bar{h}_{11}^q$
$\bar{h}_{15}^q$
$\bar{h}_{16}^q$
soft attention
LSTM preprocessing Layer for Q
h_1^q
h_2^q
h_3^q
h_Q^q
Why
did
Tesla
?

(a) Sequence Model

(b) Boundary Model

Figure 1: An overview of our two models. Both models consist of an LSTM preprocessing layer, a match-LSTM layer and an Answer Pointer layer. For each match-LSTM in a particular direction, $\bar{h}_i^{\mathrm{q}}$, which is defined as $\mathbf{H}^{\mathrm{q}}\alpha_i^{\intercal}$, is computed using the α in the corresponding direction, as described in either Eqn. (2) or Eqn. (5).

2.2 Pointer Net

Vinyals et al. (2015) proposed a Pointer Network (Ptr-Net) model to solve a special kind of problems where we want to generate an output sequence whose tokens must come from the input sequence. Instead of picking an output token from a fixed vocabulary, Ptr-Net uses attention mechanism as a pointer to select a position from the input sequence as an output symbol. The pointer mechanism has inspired some recent work on language processing (Gu et al., 2016; Kadlec et al., 2016). Here we adopt Ptr-Net in order to construct answers using tokens from the input text.

2.3 Our Method

Formally, the problem we are trying to solve can be formulated as follows. We are given a piece of text, which we refer to as a passage, and a question related to the passage. The passage is represented by matrix $\mathbf{P} \in \mathbb{R}^{d \times P}$, where P is the length (number of tokens) of the passage and d is the dimensionality of word embeddings. Similarly, the question is represented by matrix $\mathbf{Q} \in \mathbb{R}^{d \times Q}$ where Q is the length of the question. Our goal is to identify a subsequence from the passage as the answer to the question.

As pointed out earlier, since the output tokens are from the input, we would like to adopt the Pointer Net for this problem. A straightforward way of applying Ptr-Net here is to treat an answer as a sequence of tokens from the input passage but ignore the fact that these tokens are consecutive in the original passage, because Ptr-Net does not make the consecutivity assumption. Specifically, we represent the answer as a sequence of integers $\mathbf{a} = (a_1, a_2, \ldots)$, where each a_i is an integer between 1 and P, indicating a certain position in the passage.

Alternatively, if we want to ensure consecutivity, that is, if we want to ensure that we indeed select a subsequence from the passage as an answer, we can use the Ptr-Net to predict only the start and the end of an answer. In this case, the Ptr-Net only needs to select two tokens from the input passage, and all the tokens between these two tokens in the passage are treated as the answer. Specifically, we can represent the answer to be predicted as two integers $\mathbf{a} = (a_{\mathrm{s}}, a_{\mathrm{e}})$, where a_{s} an a_{e} are integers between 1 and P.

We refer to the first setting above as a *sequence* model and the second setting above as a *boundary* model. For either model, we assume that a set of training examples in the form of triplets $\{(\mathbf{P}_n, \mathbf{Q}_n, \mathbf{a}_n)\}_{n=1}^{N}$ are given.

An overview of the two neural network models are shown in Figure 1. Both models consist of three layers: (1) An LSTM preprocessing layer that preprocesses the passage and the question using LSTMs. (2) A match-LSTM layer that tries to match the passage against the question. (3) An Answer Pointer (Ans-Ptr) layer that uses Ptr-Net to select a set of tokens from the passage as the answer. The difference between the two models only lies in the third layer.

LSTM Preprocessing Layer

The purpose for the LSTM preprocessing layer is to incorporate contextual information into the representation of each token in the passage and the question. We use a standard one-directional LSTM (Hochreiter & Schmidhuber, 1997) to process the passage and the question separately, as shown below:

$$\mathbf{H}^{\mathrm{p}} = \overrightarrow{\mathit{LSTM}}(\mathbf{P}), \quad \mathbf{H}^{\mathrm{q}} = \overrightarrow{\mathit{LSTM}}(\mathbf{Q}). \tag{1}$$

The resulting matrices $\mathbf{H}^{\mathrm{p}} \in \mathbb{R}^{l \times P}$ and $\mathbf{H}^{\mathrm{q}} \in \mathbb{R}^{l \times Q}$ are hidden representations of the passage and the question, where l is the dimensionality of the hidden vectors. In other words, the i^{th} column vector $\mathbf{h}_i^{\mathrm{p}}$ (or $\mathbf{h}_i^{\mathrm{q}}$) in $\mathbf{H}^{\mathrm{p}}$ (or $\mathbf{H}^{\mathrm{q}}$) represents the i^{th} token in the passage (or the question) together with some contextual information from the left.

Match-LSTM Layer

We apply the match-LSTM model (Wang & Jiang, 2016) proposed for textual entailment to our machine comprehension problem by treating the question as a premise and the passage as a hypothesis. The match-LSTM sequentially goes through the passage. At position i of the passage, it first uses the standard word-by-word attention mechanism to obtain attention weight vector $\overrightarrow{\alpha}_i \in \mathbb{R}^Q$ as follows:

$$\begin{aligned}
\overrightarrow{\mathbf{G}}_i &= \tanh(\mathbf{W}^{\mathrm{q}}\mathbf{H}^{\mathrm{q}} + (\mathbf{W}^{\mathrm{p}}\mathbf{h}_i^{\mathrm{p}} + \mathbf{W}^{\mathrm{r}}\overrightarrow{\mathbf{h}}_{i-1}^{\mathrm{r}} + \mathbf{b}^{\mathrm{p}}) \otimes \mathbf{e}_Q), \\
\overrightarrow{\alpha}_i &= \mathrm{softmax}(\mathbf{w}^{\intercal}\overrightarrow{\mathbf{G}}_i + b \otimes \mathbf{e}_Q),
\end{aligned} \tag{2}$$

where $\mathbf{W}^{\mathrm{q}}, \mathbf{W}^{\mathrm{p}}, \mathbf{W}^{\mathrm{r}} \in \mathbb{R}^{l \times l}$, $\mathbf{b}^{\mathrm{p}}, \mathbf{w} \in \mathbb{R}^l$ and $b \in \mathbb{R}$ are parameters to be learned, $\overrightarrow{\mathbf{h}}_{i-1}^{\mathrm{r}} \in \mathbb{R}^l$ is the hidden vector of the one-directional match-LSTM (to be explained below) at position $i-1$, and the outer product $(\cdot \otimes \mathbf{e}_Q)$ produces a matrix or row vector by repeating the vector or scalar on the left for Q times.[1]

Essentially, the resulting attention weight $\overrightarrow{\alpha}_{i,j}$ above indicates the degree of matching between the i^{th} token in the passage with the j^{th} token in the question. Next, we use the attention weight vector $\overrightarrow{\alpha}_i$ to obtain a weighted version of the question and combine it with the current token of the passage to form a vector $\overrightarrow{\mathbf{z}}_i$:

$$\overrightarrow{\mathbf{z}}_i = \begin{bmatrix} \mathbf{h}_i^{\mathrm{p}} \\ \mathbf{H}^{\mathrm{q}}\overrightarrow{\alpha}_i^{\intercal} \end{bmatrix}. \tag{3}$$

This vector $\overrightarrow{\mathbf{z}}_i$ is fed into a standard one-directional LSTM to form our so-called match-LSTM:

$$\overrightarrow{\mathbf{h}}_i^{\mathrm{r}} = \overrightarrow{\mathit{LSTM}}(\overrightarrow{\mathbf{z}}_i, \overrightarrow{\mathbf{h}}_{i-1}^{\mathrm{r}}), \tag{4}$$

where $\overrightarrow{\mathbf{h}}_i^{\mathrm{r}} \in \mathbb{R}^l$.

We further build a similar match-LSTM in the reverse direction. The purpose is to obtain a representation that encodes the contexts from both directions for each token in the passage. To build this reverse match-LSTM, we first define

$$\begin{aligned}
\overleftarrow{\mathbf{G}}_i &= \tanh(\mathbf{W}^{\mathrm{q}}\mathbf{H}^{\mathrm{q}} + (\mathbf{W}^{\mathrm{p}}\mathbf{h}_i^{\mathrm{p}} + \mathbf{W}^{\mathrm{r}}\overleftarrow{\mathbf{h}}_{i+1}^{\mathrm{r}} + \mathbf{b}^{\mathrm{p}}) \otimes \mathbf{e}_Q), \\
\overleftarrow{\alpha}_i &= \mathrm{softmax}(\mathbf{w}^{\intercal}\overleftarrow{\mathbf{G}}_i + b \otimes \mathbf{e}_Q).
\end{aligned} \tag{5}$$

[1] Following Wang & Jiang (2016), we add a special token *NULL* to the end of the question so that words in the passage that do not match any word in the question can be matched to this *NULL*. So Q here is actually the number of tokens in the question plus 1.

Note that the parameters here ($\mathbf{W}^{\text{q}}$, $\mathbf{W}^{\text{p}}$, $\mathbf{W}^{\text{r}}$, $\mathbf{b}^{\text{p}}$, $\mathbf{w}$ and b) are the same as used in Eqn. (2). We then define $\overleftarrow{\mathbf{z}}_i$ in a similar way and finally define $\overleftarrow{\mathbf{h}}^{\text{r}}_i$ to be the hidden representation at position i produced by the match-LSTM in the reverse direction.

Let $\overrightarrow{\mathbf{H}}^{\text{r}} \in \mathbb{R}^{l \times P}$ represent the hidden states $[\overrightarrow{\mathbf{h}}^{\text{r}}_1, \overrightarrow{\mathbf{h}}^{\text{r}}_2, \ldots, \overrightarrow{\mathbf{h}}^{\text{r}}_P]$ and $\overleftarrow{\mathbf{H}}^{\text{r}} \in \mathbb{R}^{l \times P}$ represent $[\overleftarrow{\mathbf{h}}^{\text{r}}_1, \overleftarrow{\mathbf{h}}^{\text{r}}_2, \ldots, \overleftarrow{\mathbf{h}}^{\text{r}}_P]$. We define $\mathbf{H}^{\text{r}} \in \mathbb{R}^{2l \times P}$ as the concatenation of the two:

$$\mathbf{H}^{\text{r}} = \begin{bmatrix} \overrightarrow{\mathbf{H}}^{\text{r}} \\ \overleftarrow{\mathbf{H}}^{\text{r}} \end{bmatrix}. \tag{6}$$

Answer Pointer Layer

The top layer, the Answer Pointer (Ans-Ptr) layer, is motivated by the Pointer Net introduced by Vinyals et al. (2015). This layer uses the sequence $\mathbf{H}^{\text{r}}$ as input. Recall that we have two different models: The *sequence* model produces a sequence of answer tokens but these tokens may not be consecutive in the original passage. The *boundary* model produces only the start token and the end token of the answer, and then all the tokens between these two in the original passage are considered to be the answer. We now explain the two models separately.

The Sequence Model: Recall that in the sequence model, the answer is represented by a sequence of integers $\mathbf{a} = (a_1, a_2, \ldots)$ indicating the positions of the selected tokens in the original passage. The Ans-Ptr layer models the generation of these integers in a sequential manner. Because the length of an answer is not fixed, in order to stop generating answer tokens at certain point, we allow each a_k to take up an integer value between 1 and $P+1$, where $P+1$ is a special value indicating the end of the answer. Once a_k is set to be $P+1$, the generation of the answer stops.

In order to generate the k^{th} answer token indicated by a_k, first, the attention mechanism is used again to obtain an attention weight vector $\beta_k \in \mathbb{R}^{(P+1)}$, where $\beta_{k,j}$ $(1 \le j \le P+1)$ is the probability of selecting the j^{th} token from the passage as the k^{th} token in the answer, and $\beta_{k,(P+1)}$ is the probability of stopping the answer generation at position k. β_k is modeled as follows:

$$\mathbf{F}_k = \tanh(\mathbf{V}\widetilde{\mathbf{H}}^{\text{r}} + (\mathbf{W}^{\text{a}}\mathbf{h}^{\text{a}}_{k-1} + \mathbf{b}^{a}) \otimes \mathbf{e}_{(P+1)}), \tag{7}$$

$$\beta_k = \text{softmax}(\mathbf{v}^{\intercal}\mathbf{F}_k + c \otimes \mathbf{e}_{(P+1)}), \tag{8}$$

where $\widetilde{\mathbf{H}}^{\text{r}} \in \mathbb{R}^{2l \times (P+1)}$ is the concatenation of $\mathbf{H}^{\text{r}}$ with a zero vector, defined as $\widetilde{\mathbf{H}}^{\text{r}} = [\mathbf{H}^{\text{r}}; \mathbf{0}]$, $\mathbf{V} \in \mathbb{R}^{l \times 2l}$, $\mathbf{W}^{\text{a}} \in \mathbb{R}^{l \times l}$, $\mathbf{b}^{a}, \mathbf{v} \in \mathbb{R}^{l}$ and $c \in \mathbb{R}$ are parameters to be learned, $(\cdot \otimes \mathbf{e}_{(P+1)})$ follows the same definition as before, and $\mathbf{h}^{\text{a}}_{k-1} \in \mathbb{R}^{l}$ is the hidden vector at position $k-1$ of an answer LSTM as defined below:

$$\mathbf{h}^{\text{a}}_k = \overrightarrow{\mathit{LSTM}}(\widetilde{\mathbf{H}}^{\text{r}}\beta_k^{\intercal}, \mathbf{h}^{\text{a}}_{k-1}). \tag{9}$$

We can then model the probability of generating the answer sequence as

$$p(\mathbf{a}|\mathbf{H}^{\text{r}}) = \prod_k p(a_k|a_1, a_2, \ldots, a_{k-1}, \mathbf{H}^{\text{r}}), \tag{10}$$

and

$$p(a_k = j|a_1, a_2, \ldots, a_{k-1}, \mathbf{H}^{\text{r}}) = \beta_{k,j}. \tag{11}$$

To train the model, we minimize the following loss function based on the training examples:

$$-\sum_{n=1}^{N} \log p(\mathbf{a}_n|\mathbf{P}_n, \mathbf{Q}_n). \tag{12}$$

The Boundary Model: The boundary model works in a way very similar to the sequence model above, except that instead of predicting a sequence of indices $a_1, a_2, \ldots$, we only need to predict two indices a_{s} and a_{e}. So the main difference from the sequence model above is that in the boundary model we do not need to add the zero padding to $\mathbf{H}^{\text{r}}$, and the probability of generating an answer is simply modeled as

$$p(\mathbf{a}|\mathbf{H}^{\text{r}}) = p(a_{\text{s}}|\mathbf{H}^{\text{r}})p(a_{\text{e}}|a_{\text{s}}, \mathbf{H}^{\text{r}}). \tag{13}$$

	d	$\|\theta\|$	Exact Match		F1	
			Dev	Test	Dev	Test
Random Guess	-	0	1.1%	1.3%	4.1%	4.3%
Logistic Regression	-	-	40.0%	40.4%	51.0%	51.0%
Match-LSTM with Ans-Ptr (Sequence)	150	882K	54.4%	53.5%	68.1%	67.4%
Match-LSTM with Ans-Ptr (Boundary)	150	882K	**59.1%**	**59.5%**	**70.0%**	**70.3%**

Table 2: Experiment Results

3 Experiments

In this section, we present our experiment results and perform some analyses to better understand how our models works.

3.1 Data

We use the Stanford Question Answering Dataset (SQuAD) v1.0[2] to conduct our experiments. Passages in SQuAD come from 536 articles from Wikipedia covering a wide range of topics. Each passage is a single paragraph from a Wikipedia article, and each passage has around 5 questions associated with it. In total, there are 23,215 passages and 107,785 questions. The data has been split into a training set (with 87,636 question-answer pairs), a development set (with 10,600 question-answer pairs) and a test set that is not released. To evaluate our model on the test set, we had to submit our trained model to Rajpurkar et al. (2016), who helped us obtain the performance on the test set.

3.2 Experiment Settings

We first tokenize all the passages, questions and answers. The resulting vocabulary contains 116,576 unique words. We use word embeddings from GloVe (Pennington et al., 2014) to initialize the model. Words not found in GloVe are initialized as zero vectors. The word embeddings are not updated during the training of the model.

The dimensionality l of the hidden layers is fixed to be 150. We use ADAMAX (Kingma & Ba, 2015) with the coefficients $\beta_1 = 0.9$ and $\beta_2 = 0.999$ to optimize the model. Each update is computed through a minibatch of 30 instances. We do not use L2-regularization. We use the development set to help tune the hyper-parameters, which include the learning rate and dropout.

The performance is measured by two metrics: percentage of exact match with the ground truth answers, and word-level F1 score when comparing the tokens in the predicted answers with the tokens in the ground truth answers. Note that in the development set and the test set each question has around three ground truth answers. F1 scores with the best matching answers are used to compute the average F1 score.

3.3 Results

The results of our models as well as the results of two baselines given by Rajpurkar et al. (2016) are shown in Table 2. We can see that both of our two models have clearly outperformed the logistic regression model by Rajpurkar et al. (2016), which relies on carefully designed features. Furthermore, our boundary model has outperformed the sequence model, achieving an exact match score of 59.5% and an F1 score of 70.3%. In particular, in terms of the exact match score, the boundary model has a clear advantage over the sequence model. The improvement of our models over the logistic regression model shows that our end-to-end neural network models without much feature engineering are very effective on this task and this dataset.

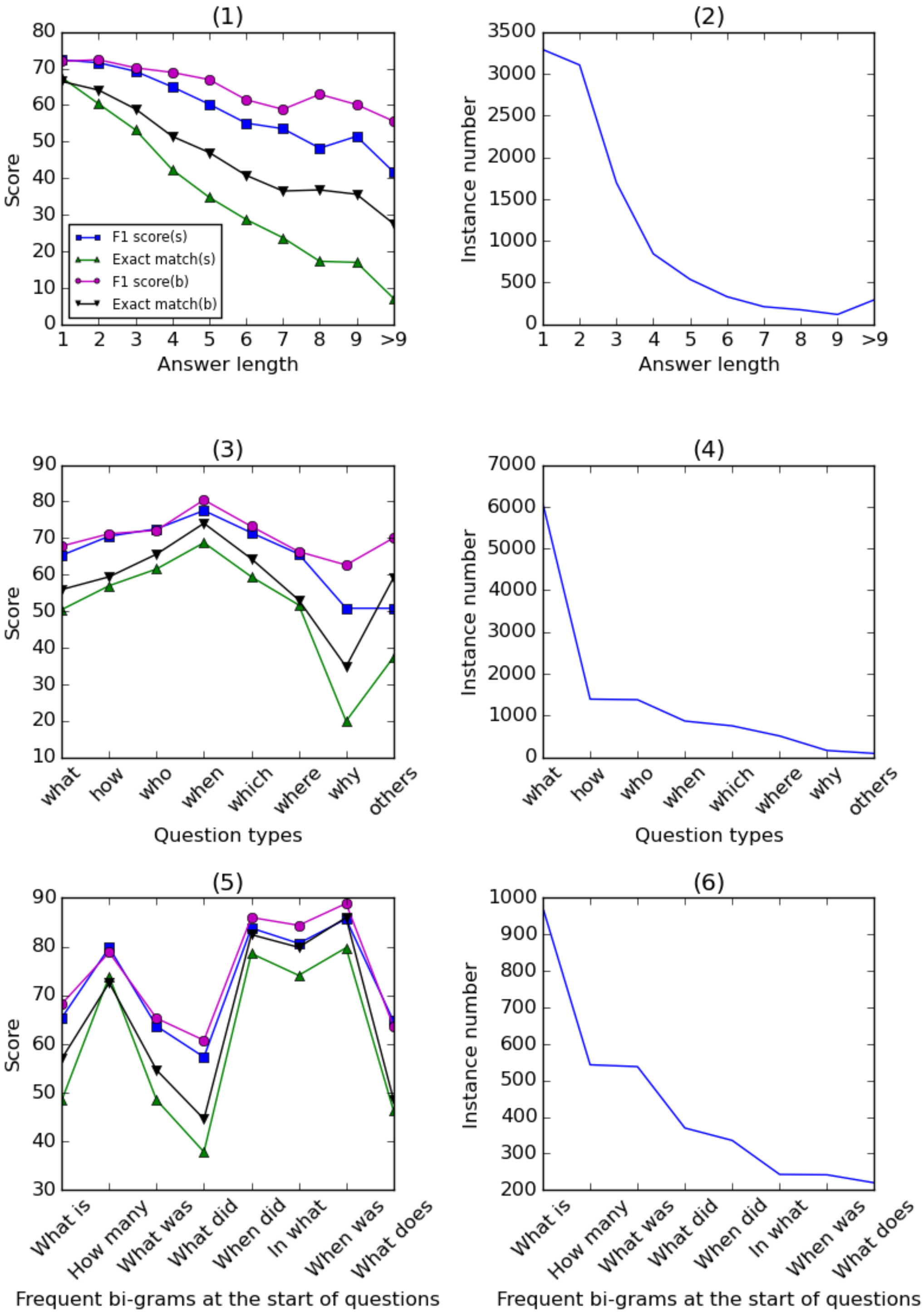


Figure 2: Performance breakdown by answer lengths and answer types. Top: Plot (1) shows the performance of our two models (where *s* refers to the sequence model and *b* refers to the boundary model) over answers with different lengths. Plot (2) shows the numbers of answers with different lengths. Middle: Plot (3) shows the performance our the two models on different types of questions. Plot (4) shows the numbers of different types of questions. Bottom: Plot (5) uses frequent bi-grams at the start of questions to define question types and shows the performance of different question types. Plot (6) shows the numbers of questions in each question type defined this way.

3.4 Further Analyses

To better understand the strengths and weaknesses of our models, we perform some further analyses of the results below.

First, we suspect that longer answers are harder to predict. To verify this hypothesis, we plot out the performance in terms of both exact match and F1 score with respect to the answer length on the development set. We also show the number of question-answer pairs with different lengths of answers. The two plots are shown in the first row of Figure 2. We can see from Plot (2) on the upper right corner of Figure 2 that there are more short answers than long answers in the dataset. And from Plot (1) on the upper left corner, we can see that our models work much better for short answers than for long answers. For example, for questions whose answers contain more than 9 tokens, the F1 score of the boundary model drops to around 56% and the exact match score drops to only around 30%, compared to the F1 score and exact match score of close to 72% and 66%, respectively, for questions with single-token answers.

Next, we analyze the performance of our models on different groups of questions. We use a crude way to split the questions into different groups based on a set of question words we have defined, including "what," "how," "who," "when," "which," "where," and "why." These different question words roughly refer to questions with different types of answers. For example, "when" questions look for temporal expressions as answers, whereas "where" questions look for locations as answers. We plot the performance on different groups of questions in the middle row of Figure 2. We also plot the numbers of different groups of questions in the middle row. As we can see from Plot (3) in Figure 2, our models work the best for "when" questions. This may be because in this dataset temporal expressions are relatively easier to recognize. Other groups of questions whose answers are noun phrases, such as "what" questions, "which" questions and "where" questions, also get relatively better results. On the other hand, "why" questions are the hardest to answer. This is not surprising because the answers to "why" questions can be very diverse, and they are not restricted to any certain type of phrases.

We observe that for many questions, not only the first question word but also the second question word is very important in characterizing the nature of the question. For example, the word "many" in questions starting with "how many" indicates that the answers should be numbers. Therefore, we further select a set of frequent bigrams that we have observed in the beginnings of questions, which presumably characterize the nature of the questions at a finer granularity. Again we plot the performance on different groups of questions based on these bigrams as well as the numbers of questions in these different groups in the bottom row of Figure 2. We can see that "how many" questions have relatively good performance, which makes sense because their answers are mostly numbers and thus easier to recognize.

Finally, we would like to check whether the attention mechanism used in the match-LSTM layer is effective in helping the model locate the answer. We show the attention weights α in Figure 3. In the figure the darker the color is the higher the weight is. We can see that some words have been well aligned based on the attention weights. For example, the word "German" in the passage is aligned well to the word "language" in the first question, and the model successfully predicts "German" as the answer to the question. For the question word "who" in the second question, the word "teacher" actually receives relatively higher attention weight, and the model has predicted the phrase "Martin Sekulic" after that as the answer, which is correct. For the last question that starts with "why", the attention weights are more evenly distributed and it is not clear which words have been aligned to "why".

4 Related Work

Machine comprehension of text has gained much attention in recent years, and increasingly researchers are building data-drive, end-to-end neural network models for the task. We will first review the recently released datasets and then some end-to-end models on this task.

[2] Rajpurkar et al. (2016) just released a v1.1 of the same dataset. We have yet to use the new version of the data to train our models.

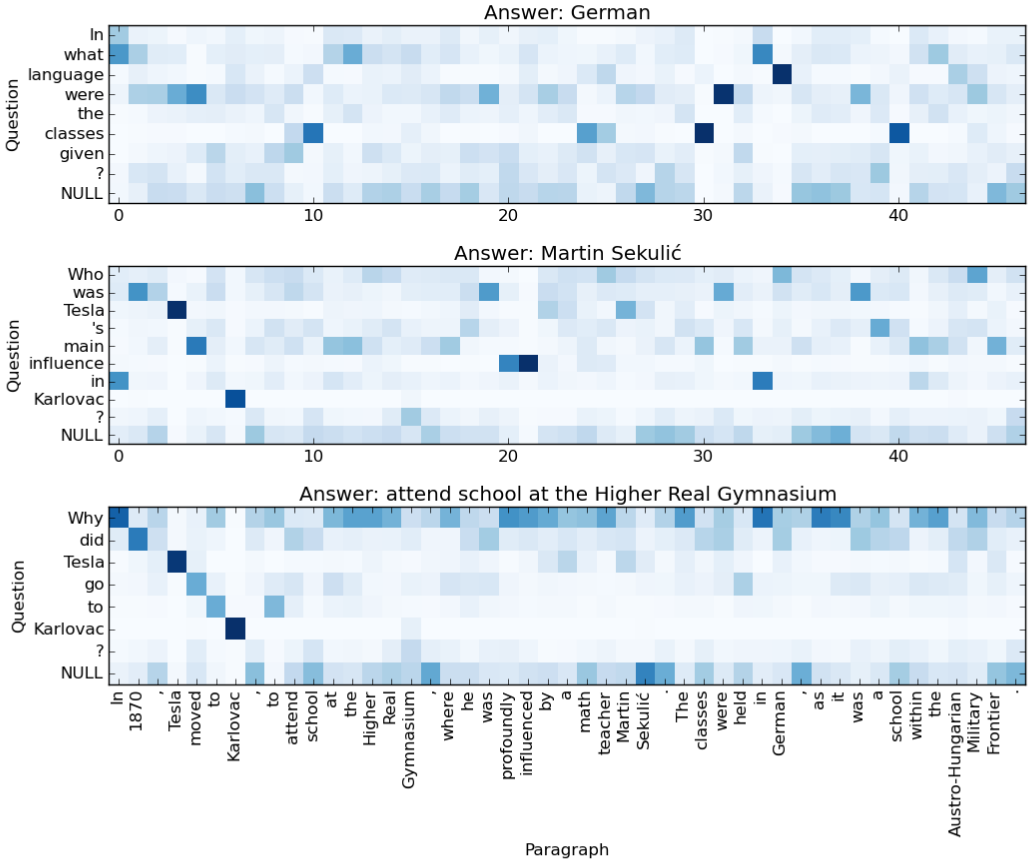


Figure 3: Visualization of the attention weights α for three questions associated with the same passage.

4.1 Datasets

A number of datasets for studying machine comprehension were created in Cloze style by removing a single token from a sentence in the original corpus. Questions created in this way are not real questions that start with “when,” “where,” etc. For example, Hermann et al. (2015) created questions in Cloze style from CNN and Daily Mail highlights, which are short summaries of the major points of news articles from CNN and Daily Mail. The task is to predict a missing token removed from a highlight based on the corresponding news article. Hill et al. (2016) created the Children’s Book Test dataset, which is based on children’s stories. Each query is the 21st sentence from a story with one word removed. The task is to use the first 20 sentences to predict the missing word in the query. Cui et al. (2016) released two similar datasets in Chinese, the People Daily dataset and the Children’s Fairy Tale dataset, following the work by Hermann et al. (2015) and by Hill et al. (2016).

Instead of creating questions in Cloze style, a number of other datasets rely on human annotators to create real questions. Richardson et al. (2013) created the well-known MCTest dataset, which is based on short fictional stories and has candidate answers provided for each question. Tapaswi et al. (2016) created the MovieQA dataset, which uses multiple sources of information including video clips, plots, subtitles, scripts and DVS of movies for question answering. In this dataset, there are also candidate answers provided. Similar to these two datasets, the SQuAD dataset (Rajpurkar et al., 2016) was also created by human annotators. Different from the previous two, however, the SQuAD dataset does not provide candidate answers, and thus all possible subsequences from the given passage have to be considered as candidate answers.

Besides the datasets above, there are also a few other datasets created for machine comprehension, but they are quite different from the datasets above in nature. Hewlett et al. (2016) created a WikiReading dataset based on Wikipedia articles and the open knowledge base Wikidata. Although the dataset was also created for natural language understanding, the queries in this dataset are not complete natural language sentences but properties such as "gender," "country" and "date of birth," and in total the dataset has 884 unique properties. It is therefore quite different from those datasets in which the questions are long sentences and have much more variations. Weston et al. (2016) created a question answering dataset that contains questions requiring different levels of intelligence such as fact chaining, counting, deduction and induction. The text was generated automatically based on a simple automated grammar and a small vocabulary, but the questions require higher levels of intelligence to answer than the aforementioned datasets.

4.2 End-to-end Neural Network Models for Machine Comprehension

There have been a number of studies proposing end-to-end neural network models for machine comprehension. A common approach is to use recurrent neural networks (RNNs) to process the given text and the question in order to predict or generate the answers (Hermann et al., 2015). Attention mechanism is also widely used on top of RNNs in order to match the question with the given passage (Hermann et al., 2015; Chen et al., 2016). Given that answers often come from the given passage, Pointer Network has been adopted in a few studies in order to copy tokens from the given passage as answers (Kadlec et al., 2016; Trischler et al., 2016). Compared with existing work, we use match-LSTM to match a question and a given passage, and we use Pointer Network in a different way such that we can generate answers that contain multiple tokens from the given passage.

Memory Networks (Weston et al., 2015) have also been applied to machine comprehension (Sukhbaatar et al., 2015; Kumar et al., 2016; Hill et al., 2016), but its scalability when applied to a large dataset is still an issue. In this work, we did not consider memory networks for the SQuAD dataset.

5 Conclusions

In this paper, we proposed a three-layer neural network architecture to solve the machine comprehension problem defined in the Stanford Question Answering (SQuAD) dataset. We developed two models for the problem, both making use of match-LSTM and Pointer Network. Experiments on the SQuAD dataset showed that both of our models could substantially outperform the best performance achieved by (Rajpurkar et al., 2016). In particular, our second model, the boundary model, could achieve an exact match score of 59.5% and an F1 score of 70.3% on the test dataset. Our further analyses showed that our models work better for questions with short answers and certain types of questions such as "when" questions.

In the future, we plan to look further into the different types of questions and focus on those questions which currently have low performance, such as the "why' questions. We also plan to test how our models could be applied to other machine comprehension datasets.

6 Acknowledgments

We thank Pranav Rajpurkar for testing our model on the hidden test dataset and Percy Liang for helping us with the Dockerfile for Codalab.

References

Danqi Chen, Jason Bolton, and Christopher D. Manning. A thorough examination of the CNN/Daily Mail reading comprehension task. In *Proceedings of the Conference on Association for Computational Linguistics*, 2016.

Yiming Cui, Ting Liu, Zhipeng Chen, Shijin Wang, and Guoping Hu. Consensus attention-based neural networks for chinese reading comprehension. In *arXiv preprint arXiv:1607.02250*, 2016.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the Conference on Association for Computational Linguistics*, 2016.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*, pp. 1693–1701, 2015.

Daniel Hewlett, Alexandre Lacoste, Llion Jones, Illia Polosukhin, Andrew Fandrianto, Jay Han, Matthew Kelcey, and David Berthelot. WIKIREADING: A novel large-scale language understanding task over wikipedia. In *Proceedings of the Conference on Association for Computational Linguistics*, 2016.

Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. The Goldilocks principle: Reading children's books with explicit memory representations. In *Proceedings of the International Conference on Learning Representations*, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Rudolf Kadlec, Martin Schmid, Ondrej Bajgar, and Jan Kleindienst. Text understanding with the attention sum reader network. In *Proceedings of the Conference on Association for Computational Linguistics*, 2016.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*, 2015.

Ankit Kumar, Ozan Irsoy, Jonathan Su, James Bradbury, Robert English, Brian Pierce, Peter Ondruska, Ishaan Gulrajani, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. In *Proceedings of the International Conference on Machine Learning*, 2016.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global vectors for word representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2014.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2016.

Matthew Richardson, Christopher JC Burges, and Erin Renshaw. MCTest: A challenge dataset for the open-domain machine comprehension of text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2013.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Proceedings of the Conference on Advances in neural information processing systems*, 2015.

Makarand Tapaswi, Yukun Zhu, Rainer Stiefelhagen, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. MovieQA: Understanding stories in movies through question-answering. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Adam Trischler, Zheng Ye, Xingdi Yuan, and Kaheer Suleman. Natural language comprehension with the EpiReader. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2016.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*, 2015.

Shuohang Wang and Jing Jiang. Learning natural language inference with LSTM. In *Proceedings of the Conference on the North American Chapter of the Association for Computational Linguistics*, 2016.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. In *Proceedings of the International Conference on Learning Representations*, 2015.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards AI-complete question answering: A set of prerequisite toy tasks. In *Proceedings of the International Conference on Learning Representations*, 2016.

Wenpeng Yin, Sebastian Ebert, and Hinrich Schütze. Attention-based convolutional neural network for machine comprehension. *arXiv preprint arXiv:1602.04341*, 2016.