Machine Comprehension Using Robust Rule-Based Features and Multiple-Sentence Enhancing

Wei Chen
`cwind` [†]

Danyang Wang
`danyangw` [†]

Xiaoshi Wang
`xiaoshiw` [†]

Abstract

In this project, we designed multiple featurelizers to extract information and answer multiple choice reading comprehension questions. Given a triple of passage question and answer, the featurelizer will generate a set of features which are designed using robust NLP tools. We then feed generated features into a neural network classifier which gives a probability score for each answer. The features we used are improved sliding window, key word distance, syntax feature, word embeddings, multiple sentences and coreference resolution.

1 Introduction

Machine comprehension (MC) is a raising research field which attracts interest from both industry and academia. There are a number of datasets available for this task, each designed to reflect different challenges in MC. The Facebook bAbI dataset (Weston et al., 2015) contains short examples which requires to derive answers by combining two sentences. The MC500 dataset (Richardson et al., 2013) contains longer passages and various types of multiple choice questions. Wiki QA (Smith et al., 2013) introduces a context where agents are required to read full-length wikipedia articles before answering related questions.

We use MC500 as the guildeline to evaluate our MC system. The dataset provides 500 instances of stories. Each instance contains one passage, four multiple choice questions and each question contains four choices. Instances are designed such that answers to each question can be entailed only using information in the passage.

[†]`@stanford.edu`

A typical question is shown as follows:

> Passage: ...John asked Tim if he could play on the slide. Tim said no. John was very upset and started crying. A girl named Susan saw him crying. Susan told the teacher Ms. Tammy. ...
>
> Question: Who saw John crying and told Ms. Tammy?
> A) Tim
> B) Susan
> C) John
> D) Ms. Tammy

Such tasks are easy for human readers but are hard for machines. Even though answers are designed to be retrievable from the passage, it does not imply that the agent could be knowledge-free. In fact, locating the answer in the passage requires a large knowledge base. For example, in the sample above,an agent has to understand "a girl named Susan" implies "the girl is Susan". In general, Machine Comprehension challenge involves the application of many fields, especially in NLP and machine learning.

2 Previous Works

State-of-art methods usually approach Machine comprehension in two major directions: neural network based and feature based. Both directions incorporates with the observations that the knowledge required to retrieve answers must be learned or hard-coded.

For neural network based methods, (Hermann et al., 2015) suggests word similarities can be used to train LSTM which embed sentences into a vector space which preserves sentence similarities. Similarly, (Kapashi and Shah, 2015) uses word embedding (Mikolov et al., 2013) vectors as input to LSTM to approach Wiki QA task.

On the other hand, feature designed systems represent candidate answers by hand-crafted features to train linear classifiers (regression (Wang et al., 2015), SVM (Narasimhan and Barzilay, 2015), ect.). (Richardson et al., 2013) proposed a simple bag-of-word and word-distance feature as baseline. (Smith et al., 2015) uses enhanced bag-of-word feature to capture cross-sentence matches. (Wang et al., 2015) uses various NLP tools to process the original data and uses matches in semantics, dependencies and work tokens as features. In (Sachan et al., 2015), the system introduce hypothesis as latent variables in the model.

Neural network based systems automatically generate features from a weak feature, whereas feature based systems provide strong features specifically designed and tuned for the dataset. Therefore, neural network based systems have more potential to generalize; however, feature based systems reveals more NLP natures of the problem and have better performance in practice.

3 Approach

3.1 Overview

Our system framework is similar to the one proposed in (Wang et al., 2015). For each multiple choice question, it consists of a passage P, a question Q and a set of answers $A = \{A_1, A_2, A_3, A_4\}$; there is exactly one answer $A^* \in A$ labeled correct. Our goal is to select the answer A^* without knowing the labelings.

Given a multiple question (P, Q, A), our system generates features for each answer candidates A_i. The features are represented in feature vector $f(P, Q, A_i)$. Those features can be used to train a classifier which selects the answer. In training stage, our pipeline includes three steps:

1. Read in each question (P, Q, A) and preprocess the text file using Stanford CoreNLP (Manning et al., 2014) tools.
2. From preprocessed data, generate feature vectors $f(P, Q, A_i)$ for each candidate answer. We also label each candidate answer with a binary label indicating whether it is the correct one.
3. Train a classifier using feature vectors and binary labels.

Similarly, in testing stage we also preprocess the data and generate feature vectors. However, given candidates to a particular question (P, Q, A_i), instead of generating binary labels, the classifier outputs a distribution describing the likelihood that A_i is correct. We select the answer with maximum likelihood.

3.2 Enhanced Sliding Window Features

The baseline system proposed in (Richardson et al., 2013) uses a simple bag-of-words score to evaluate each candidate answer. Concretely, we first concatenate the question and an answer to form a string s. Within a sliding window in P of size k, we count the number of word matches to s and score the answer by the maximum sliding window count. To prevent counts boosted by trivial words, we weighted the count of each word w by its inverse frequency across the passage.

The original baseline uses $k = \text{word\_size}(s)$, which is the total number of words in s. (Narasimhan and Barzilay, 2015) suggests that a simple modification could significantly boost the performance. Instead of using the score of size-k sliding window, we used the sum of sliding window scores from size-2 to size-30. We also notice that weighted sum of sliding windows of various size has even better performance in experiments. We use fixed weights set to be the inverse of the sliding window sizes.

3.3 Distance Features

Another feature used in the baseline system is distance of key words between question and candidate answer. The intuition is that the part of the passage representing the question is usually not far away from the one for the true answer. For a question Q and answer A_i, we calculate the "distance" between the question and answer by:

$$d_i = \min_{q \in S_Q, a \in S_{A,i}} d(q, a),$$

with

$$S_Q = (Q \cap PW) \backslash U,$$

and

$$S_{A,i} = (A_i \cap PW) \backslash (U \cup Q).$$

U provides a dictionary of "stop words" which intend to filter out non-keywords.

3.4 Syntax Features

The sliding window score captures similarities by word matches. However, it does not capture matches in word dependencies. Investigating grammar structures of questions and answers (Wang et al., 2007) provides insights to latent variables which aligns question-answer pair. In syntax features, we represent similarities between statements using dependency tree parsing (Chen and Manning, 2014).

We generate statements mainly following rules proposed by (Wang et al., 2015). For example:

Q: What did he do on Tuesday?
A: He went to school.
Generated: He went to school on Tuesday.

We slightly modified the rules to obtain more accurate results. In general, we denote $arc(u, v)$ be the grammar relationship between word u and word v and $POS(u)$ be the part-of-speech Penn Tree tag for word u. For question q, let c be the wh-word and r_q be the root word. Similarly, for answer a, let r_a be the root word. Rules are described as follows, each with an example in the box:

1. $c = what$, $POS(r_q) = VB$ and $r_q = do$, and $arc(c, r_q) = dobj$. Let u_q be the word such that $arc(u_q, r_q) = nsubj$. If r_a is a verb, let u_a be the word such that $arc(u_a, r_a) = nsubj$, removed u_a from a. In q, remove the first two words as well as r_q, insert the updated a after u_q.

2. $c = what$, $POS(r_q) = VB$ and $r_q \neq do$, and $arc(c, r_q) = dobj$. If r_a is a verb, let u_a be the word such that $arc(u_a, r_a) = nsubj$. Remove u_a and r_a from a. In q, insert the updated a after r_q.

3. $c = what$, $POS(r_q) = NN$, and $arc(c, r_q) = nsubj$. If r_a is a verb, let u_a be the word such that $arc(u_a, r_a) = nsubj$ and replace the entire a by u_a. In q, replace c by the updated a.

4. $c = where$, $arc(c, r_q) = advmod$, and $POS(r_q) = VB$. If there is a word u_q such that $arc(u_q, r_q) = dobj$. In q, insert a after u_q, otherwise insert a after r_q. Also delete first two words in q.

5. $c = where$, $arc(c, r_q) = advmod$, and $r_q = is$. Let u_q be the word such that $arc(u_q, r_q) = nsubj$. In q, delete the first two words, put r_q after u_q, and finally insert a after q.

6. $c = who$, $arc(c, r_q) = nsubj$, and $POS(r_q) = NN$. If r_a is a verb, let u_a be the word such that $arc(u_a, r_a) = nsubj$ and replace the entire a by u_a. In q, replace c by the updated a.

It is also intuitively sounding to construct rules for other types of questions. For example, for "why" questions we can generate the statement by connecting the answer and question using "because of". However, in practice, the passage usually does not explicitly state such relationships. It is more likely that the actual statements in the passage lies in several different logically connected sentences. Therefore, we cannot retrieve dependency matching out of those type of answers. It turns out that adding more such rules rarely improve the result.

After generating the answer statement, we parse the statement and compare its dependency tree to each sentence s in the passage. The score $sy(P, Q, A_i, s)$ to each sentence is given by the number of exact dependency matches. In other words, let $e_s(u_s, v_s)$ be an edge for s and $e_a(u_a, v_a)$ be one for a, we increment sy by 1 if $u_s = u_a$, $v_s = v_a$ and $arc(u_s, v_s) = arc(u_a, v_a)$. We select the sentence with maximum score as the syntax feature:

$$sy(P, Q, A_i) = \max_{s \in P}(P, Q, A_i, s).$$

3.5 Word Embeddings

In previous features, we compare word matches with direct string match. (Mikolov et al., 2013) suggests that it is possible to embed words into a vector space in which similarities between words is measured by inner products. Furthermore, (Wang et al., 2015),(Hermann et al., 2015) and (Kapashi and Shah, 2015) suggests that in linear combination of word vectors is representative for phrases and sentences. In addition, as the embedding provided by (Mikolov et al., 2013) is trained over a dataset containing over 30 billion training words, this embedding implicitly covers abundant amount of knowledge which we might not able to retrieve if training merely on the MC500 dataset.

The word embedding we feature measures similarities between concatenated Q-A pairs $(Q + A_i)$ and a sentence (s) in the passage. Concretely, we measure the cosine of the angle between the vector of the concatenated statement and that of the sentence. Let $v(w)$ be the embedding of a given word w:

$$v_{QAi} = \sum_{w \in Q + A_i} v(w),$$

and

$$v_s = \sum_{w \in s} v(w),$$

we use the word embedding feature:

$$we(P, Q, A_i) = \max_s \frac{v_{QAi}^T v_s}{\|v_{QAi}\| \|v_s\|}.$$

3.6 Multiple Sentences

Our previous features, sw, sy, and we, each represent the triple (P, Q, A_i) using a score of a particular portion of the passage (a sliding window or a sentence). Although the score is obtained by comparing to those obtained from other portions of the passage, the feature itself only reflects a local statistics of the passage, but failed to retrieve information across the whole passage.

For a feature $F(P, Q, A_i)$ selected from individual sentence scores $f(P, Q, A_i, s)$, we previously have:

$$F(P, Q, A_i) = \max_{s \in P} f(P, Q, A_i, s).$$

We enhance the score selection process so that features previously taken on single sentences are improved to incorporate the overall score of the passage. For a sentence s, we compute the weighted sum of all sentences in the passage, taking nearby sentences with higher weights. Specifically, we generate new score g for each sentence by:

$$\begin{aligned} &g(P, Q, A_i, s) \\ &= \sum_{s' \in P} \exp\left\{ -\frac{d(s, s')^2}{\tau^2} \right\} f(P, Q, A_i, s). \end{aligned}$$

Then we select the maximum of the enhanced feature:

$$F'(P, Q, A_i) = \max_{s \in P} g(P, Q, A_i, s).$$

3.7 Coreference Resolution

Our previously used exact-string match and word vector similarities would fail on coreferences. In other words, if two different phrases (e.g. “Mr.Obama” and “the president”) actually refer to the same entity, our feature generator should regard them as matched phrases.

Stanford CoreNLP (Manning et al., 2014) provides a robust tool for coreference resolution. Given a passage, CoreNLP pipelines generates a set of entities and for each entities a set of tokens refering to this entity. It also selects the most representative name string for each entity. For each token in the passage, if it refer to certain entity, we replace the token by the name of the entity.

4 Classifier

Since the data contains 75% 0’s and 25% 1’s, we balance the data by replicating each correct candidate three times. We select the best model by selecting the hyperparameters which optimize the k-fold cross validation result on the union of training and developing set. In practice we choose $k = 10$. Our final system uses a shallow neural network as the classifier. The neural network contains one hidden layer with 12 nodes. The learning rate is 0.3 and the momentum is 0.09.

5 Performance

Our system achieved results comparable to the baseline system. Although the full implementation did not yield the desired accuracy, further investigation does reveal many interesting NLP natures of the problem.

Table 1 shows accuracies using a single feature. Word embedding we turns out to have the best single-feature improvement. It implies that word embedding is a representative feature having the potential to support more generalized models as suggested by (Hermann et al., 2015) and (Kapashi and Shah, 2015).

Feature	Accuracy
Baseline(SW+D)	0.551
Enhanced SW+D	0.565
SW+D+Coref	0.555
SW+D+Syntax	0.570
SW+D+WordEmbed	0.595

Table 1: Performance of Single Features. The first line uses baseline features SW+D. Each other row is obtained by adding one feature/improvement over the baseline SW+D features.

Table 2 shows the final performance with full implementation. We are able to achieve accuracy comparable to baseline system reported in (Richardson et al., 2013) on test set and dev set. Since our train accuracy is significantly higher than others, we add one more set of experiments to ensure that the classifier is not overfitted. We swap the train set and test set. It turns out that we still get high accuracy on train set, which implies that questions in the test set is harder than those in the train set.

	dev	test	train	train*
Single	0.512	0.577	0.658	0.622
Multi	0.614	0.548	0.613	0.560
All	0.570	0.561	0.633	0.588

Table 2: Performance of Full Implementation. We train the dataset using “train” and record the accuracy over “dev”, “test” and “train”. The last column is obtained using “test” as training data but “train” as test data.

To further investigate the performance, we record accuracy on different types of questions. Notice that the accuracy is high on ”why”, ”when” and ”how” questions. We are able to outperform (Smith et al., 2015) on ”how” type questions by 12.0%.
As shown in table 3, the most challenging type of questions we found in the dataset are those required to summarize a particular aspect of the passage:

Type	Correct	Total	Accuracy
How	15	24	0.625
When	5	7	0.714
Which	8	24	0.333
Why	39	61	0.639
Count	7	17	0.412
What	178	307	0.580
Where	29	57	0.509
Who	43	77	0.558
Other	13	26	0.500
All	337	600	0.561

Table 3: Accuracy on Different Types of Questions. The statistics are taken over test set.

Q: How many rooms did I say I checked.

In those questions, neither the answer statements nor their paraphrases are explicitly embedded in the passage. It requires us to generate specified matches (“I say I checeked”) as we did for other features, and then reduce those matches (“How many”) to the answer. The major challenge requires us to parse the question into two parts and solve two sub-questions cooperatively.

6 Future Considerations

We could extend features which require match counting to incorporate with word embedding space to better reflect wordwise similarities. For syntax feature, our analysis show that dependency graph match should be weighted to reflect relationship between a question and an answer candidate. For multiple sentence enhancing, we can view our approach as applying a convolution filter over the score of each sentence, which reveals the potential of applying signal processing or CNN techniques.

7 Conclusion

In this project we developed a MC system using sliding windows, syntax, and word embedding as features. We improved those features through coreference resolution and multiple sentence enhancing. Our full system performance reveal interesting NLP natures of the MC task, which implies possible furture research direction.

References

J.Weston, A.Bordes, S.Chopra, A.M.Rush, B.Merrienboer, and T.Mikolov *Towards AI Complete Question Answering: A Set of Prerequisite Toy Tasks* arXiv:1502.05698

N.A.Smith, M.Heilman, and R.Hwa *Question Generation as a Competitive Undergraduate Course Project* In *Proceedings of the NSF Workshop on the Question Generation Shared Task and Evaluation Challenge*, Arlington, VA, 2008

C.D.Manning, M.Surdeanu, J.Bauer, J.Finkel, S.J.Bethard, and D.McClosky *The Stanford CoreNLP Natural Language Processing Toolkit* ACL, 2015

M.Richardson, C.J.C.Burgers, and E.Renshaw *MCTest: A Challenge Dataset for the Open-Domain Machine Comprehension of Text*, EMNLP, 2013.

H.Wang, M.Bansal, K.Gimpel, and D.McAllester *Machine Comprehension with Syntax, Frames, and Semantics*, ACL, 2015

K.Narasimhan and R.Barzilay *Machine Comprehension with Discourse Relations* MCDR, 2015

M.Sachan, A.Dubey, E.P.Xing, and M.Richardson *Learning Answer-Entailing Structures for Machine Comprehension* IJCNLP, 2015

E.Smith, N.Greco, M.Bosnjak, and A.Vlachos *A Strong Lexical Matching Method for the Machine Comprehension Test* EMNLP, 2015

T.Mikolov, I.Sutskever, K.Chen, G.S.Corrado, and J.Dean *Distributed Representations of Words and Phrases and Their Compositionality* In *Advances in Neural Information Processing Systems 26*, pages 3111-3119. Curran Associates, Inc.

K.M.Hermann, T.Kocisky, E.Grefenstette, Lasse Espeholt, W.Kay, M.Suleyman, and P.Blunsom *Teaching Machines to Read and Comprehend* arXiv:1506.03340v3, 2015

D.Chen and C.Manning *A fast and accurate dependency parser using neural networks* EMNLP, 2014

D.Kapashi and P.Shah *Answering Reading Comprehension Using Memory Networks* cs224d report, Stanford University. 2015

M.Wang, N.Smith, and T.Mitamura *What is the Jeopardy Model? A Quasi-Synchronous Grammar for QA* EMNLP, 2007