Two-Step Information Retrieval for Automatic Question Answering

Arjun Panickssery

Abstract

Automatic question-answering through machine learning algorithms has numerous applications in business, public health, and other real-world problems where skilled work has the potential to be supplemented or replaced with AI natural language understanding. This investigation examined the AI2 Reasoning Challenge (ARC) question dataset provides multiple-choice questions to compare standard information retrieval methods, using text span matching from a large corpus of known sentences, with a more sophisticated two-step information retrieval protocol. The hypothesis was that this method could use contextual information to more accurately answer questions.

The results showed that the experimental group, using a two-step method, had a 36.0% accuracy rate compared to 33.1% for the control group. These results were statistically significant ($p = 0.01$) and suggest that the contextual information were more productive than the more straightforward direct text similarity metrics. Future research will focus on how this effect varies by difficulty, question length, and other variables.

Introduction

The goal of developing algorithms to automatically answer questions has been considered important for over two decades, with the American Association for Artificial Intelligence calling it one of the "Grand Challenges of AI" in 2005. However, research into automatic question answering has seen tremendous expansion in just the past five years due to the growth of several large datasets on which to conduct experiments (Brachman, 2005). Artificial intelligence (AI) models that can process information from a background resource of hundreds of thousands of documents and synthesize the data to produce useful knowledge have the potential to supplement and even replace many functions that are now performed by a human being at great expense and after many years of training. The most ambitious of these goals is to eventually build systems that can imitate the basic abilities of skilled professionals in areas of the world too impoverished to afford the high cost of training and paying for services.

The relatively recent development of large datasets consisting of natural-language text resources (such as newspaper articles, science textbooks, or Wikipedia articles) accompanied by questions. The most prominent of these are the Stanford Question Answering Dataset (SQuAD) and TriviaQA (Rajpurkar et al., 2016; Joshi et al., 2017). However, the questions in these datasets are free-response and the answer is guaranteed to be a specific span of text within one of the resource documents. Models are then typically scored based on the text overlap between the span of text they select and the correct answer's text span.

This has encouraged the development of algorithms primarily focused on direct text similarity, taking advantage of the fact that many sentences that contain the same uncommon words tend to be on the same topic. In any attempt to counteract the trend of researchers developing increasingly sophisticated direct similarity models for marginal gains, the AI2 Reasoning Challenge (ARC) dataset was introduced in 2018 (Clarke et al., 2018). The questions consist of 5,197 elementary and middle-school level science questions. To accompany the questions are a corpus of 14 million science-related sentences that the authors predict contain the answers to roughly 95% of the questions. Most importantly, the questions are four-answer multiple-choice; perhaps counterintuitively, this can make the task more difficult since the answer is not guaranteed to exist word-for-word anywhere in the corpus.

Simple word retrieval is significantly less effective in this case and is unlikely to solve the problem well, given the constraints of the format and the corpus (Radford et al., 2018). Instead, more advanced models attempt to incorporate the context or other semantic information in the question and in the corpus's sentences to provide better understanding, with the best models currently performing with about 65% accuracy (Boratko et al., 2019). This experiment tested the effect of applying two-step contextualization, a method which involves a look-up of sentences in the corpus related to the question followed by a second look-up of words and phrases in the nearby context of those sentences. I hypothesized that this would obtain a larger quantity of relevant information and would produce a model that performed competitively with existing methods.

Methodology

A Python script was written to process the training data, test data, and 14-million-sentence scientific fact database. Before applying either the experimental or control algorithm, the data was cleaned of abnormalities and the corpus was preprocessed to filter out stop words, common words such as "the," "an," and "in" that do not provide meaningful data.

For a control group, a generic information retrieval model was designed that uses a single step to identify key terms in the question and look them up in the corpus; this is based on the control model used by Ni et al., 2018 to compare with their distraction-removal algorithm. The control group's algorithm selected its responses by identifying sentences in the corpus that shared words (stop words excluded) with the question. Using this subset of the sentences, the answer choice which had words in common with the most sentences was selected as the prediction.

The experimental model used two-step contextualization but standard protocol otherwise, keeping other variables constant. Whereas the control group simply counted the instances of co-occurrence between the question and answer text spans, the experimental group's algorithm went a step further by analyzing other nearby words in those sentences. It developed a score for each answer choices computed by adding the number of sentences sharing words with both the question and the answer choice to the number of sentence containing words in the context of those overlapping sentences.

Results measured the accuracy using ARC's metric, by which a model can select as many answer choices as it wants for each question, and if any of them is correct the score is increased by a value of 1 divided by the number of answer choices that were selected. In addition to comparing against the control model, performance results were also compared to existing models, the best of which are featured in the ARC website. The top model listed performs at 67.07% accuracy and the 10^{th}-best at 36.6%. The training time and run time of both models were also recorded and compared with each other.

Results

A Python script was written to examine and preprocess the data (see Appendix for full implementation). Of the 5,197 questions, it was found that 24 questions had either three or five answer choices rather than five. These questions were removed from the dataset, leaving 5,173 data points remaining.

Data exploration revealed that 4,954 of these questions contained answers marked A, B, C, and D; the remaining 219 had answered marked 1, 2, 3, and 4. All of the numerically marked multiple-choice questions were from an exams labeled in the dataset as "NYSEDREGENTS." All of the exams are listed in the table below with the grade levels and years of their respective questions:

Table 1: Distribution of ARC Question Data by Exam

Exam Name	**Grade Level(s)**	**Year(s)**
TAKS	8	2009
California Standards Test	8	2009
Alaska Dept. of Education & Early Development	4	2008
MCAS	4, 5, 8, 9	1998, 2000, 2001, 2003, 2005, 2006, 2010
Louisiana Educational Assessment Program	5, 8	2005
Virginia Standard of Learning – Science	3, 5	2007, 2008, 2009, 2010

Mercury	3, 4, 5, 6, 7, 8, 9	2015
ACTAAP	5	2009, 2010, 2015
MSA	5	2013
California Standards Test – Science	5	2008
North Carolina READY End-of-Grade Assessment	8	2013
Maryland School Assessment	8	2008
NYSEDREGENTS	4, 8	2008, 2010, 2012, 2015

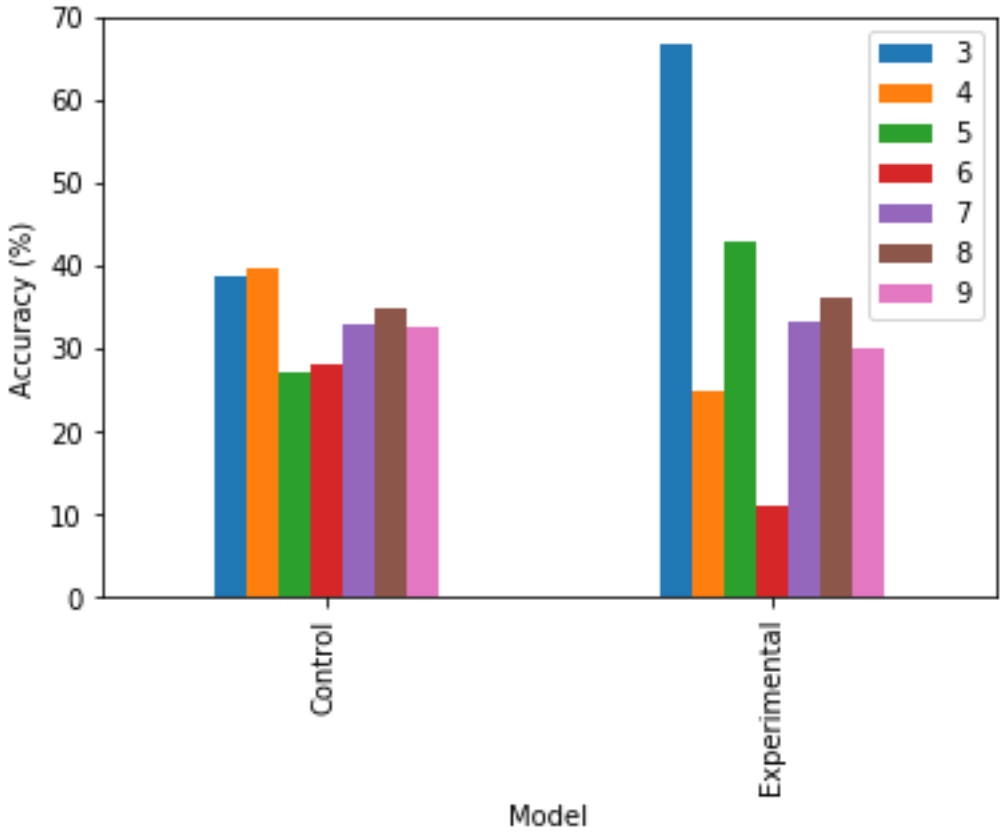
Loading the Python libraries took 0.78 seconds. Loading the question data from its compressed file format took 3.61 seconds. Loading the sentence data from its plain text format took 20.47 seconds. The corpus of sentences was found to contain 14,621,856 sentences. Examples of sentences found include "Large international companies are involved in bauxite, iron ore, diamond, and gold mining operations" and "ORDER ODONATA (Damselflies and Dragonflies) Diagnosis: large, to over three inches long; four wings, transparent and membranous, held vertically (damselflies) or laterally (dragonflies) at rest; chew- ing mouth parts, tooth-like; nymphs aquatic, feeding on mosquito larvae to small fish; adults terrestrial, feeding on other insects (Figure 14.27)." The sentences varied in relevance but were all science-related.

Due to limitations on processing power, one-tenth of the sentences (i.e. 1,462,186) were selected randomly to be used for training the model. Using the NLTK library list of English language stop words, each of the filtered sentences was converted into a list of meaningful words. This manipulation took 813.06 seconds of processing time. The resulting word-lists contained an average of 9.33 words.

Following the preprocessing steps, and control group and experimental models were run and their results recorded. The control group had an accuracy of 33.1% and the experimental group had an accuracy of 36.0%. This difference was statistically significant with a p-value of 0.001 based on a 2-population proportion z test. The control group's calculations ran faster, at 2,006.45 seconds compared to 2,755.02 seconds for the experimental group.

These results are not comparable to current state-of-the-art neural network models. The experimental group's algorithm narrowly scores alongside the top twenty algorithms listed by the Allen Institute for AI. However, the overall performance, which is affected by limitations in processing power and algorithm complexity, does not affect the comparison between the particular explanatory variable—the two-step information retrieval—since the other variables are being kept constant.

Figure 1: Model Performance by Grade Level



Also important is the relative performance across various grade levels. Figure 1 compares the performance of the two models for each of the seven possible grade levels with which each question is labeled. It is evident that the experimental group had much more variability, with a standard deviation of 17.17 percentage points of accuracy compared to the control group's 4.86 percentage points of accuracy. Furthermore, the experimental algorithm saw a modest performance increase on the earlier grade levels, in which the questions were presumably easier in difficulty.

Conclusions

Overall, the implementation of two-step information retrieval was conducive to statistically significant ($p < 0.01$) improvements in model accuracy. In terms of sources of error, the limitations on processing power prevented the maximization of general accuracy, but since this source of error was equivalent in both the control and experimental groups it should not affect the comparison between the one-step and two-step retrieval algorithms. On the other hand, having a larger corpus of pre-trained sentence data available is a limitation that may have changed the results. The evidence suggests that the use of context provides measurable improvements to question-answering machine learning models. Future research could potentially investigate the relationship between question difficulty and the improvements gained through added contextual measures, a reasonable hypothesis being that the effects of context-based methods would increase with higher-complexity question material.

References

Boratko, M., Padigela, H., Mikkilineni, D., Yuvraj, P., Das, R., McCallum, A., … Witbrock, M. (2018). A Systematic Classification of Knowledge, Reasoning, and Context within the ARC Dataset. *Proceedings of the Workshop on Machine Reading for Question Answering*, 60–70. https://doi.org/10.18653/v1/W18-2607

Boratko, M., Padigela, H., Mikkilineni, D., Yuvraj, P., Das, R., McCallum, A., … Witbrock, M. (2018). An Interface for Annotating Science Questions. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 102–107. https://doi.org/10.18653/v1/D18-2018

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., & Tafjord, O. (2018). Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *arXiv:1803.05457 [Cs]*. Retrieved from http://arxiv.org/abs/1803.05457

Conneau, A., Kiela, D., Schwenk, H., Barrault, L., & Bordes, A. (2017). Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. *EMNLP*. https://doi.org/10.18653/v1/d17-1070

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [Cs]*. Retrieved from http://arxiv.org/abs/1810.04805

Musa, R., Wang, X., Fokoue, A., Mattei, N., Chang, M., Kapanipathi, P., … Witbrock, M. (2018). Answering Science Exam Questions Using Query Rewriting with Background Knowledge. *arXiv:1809.05726 [Cs]*. Retrieved from http://arxiv.org/abs/1809.05726

Ni, J., Zhu, C., Chen, W., & McAuley, J. (2018). Learning to Attend On Essential Terms: An Enhanced Retriever-Reader Model for Open-domain Question Answering. *arXiv:1808.09492 [Cs]*. Retrieved from http://arxiv.org/abs/1808.09492

Pan, X., Sun, K., Yu, D., Chen, J., Ji, H., Cardie, C., & Yu, D. (2019). Improving Question Answering with External Knowledge. *arXiv:1902.00993 [Cs]*. Retrieved from http://arxiv.org/abs/1902.00993

Radford, A. (2018). *Improving Language Understanding by Generative Pre-Training*.

Sun, K., Yu, D., Yu, D., & Cardie, C. (2018). Improving Machine Reading Comprehension with General Reading Strategies. *arXiv:1810.13441 [Cs]*. Retrieved from http://arxiv.org/abs/1810.13441

Zhang, Y., Dai, H., Toraman, K., & Song, L. (2018). KG^2: Learning to Reason Science Exam Questions with Contextual Knowledge Graph Embeddings. *arXiv:1805.12393 [Cs, Stat]*. Retrieved from http://arxiv.org/abs/1805.12393

Zhong, W., Tang, D., Duan, N., Zhou, M., Wang, J., & Yin, J. (2018). Improving Question Answering by Commonsense-Based Pre-Training. *arXiv:1809.03568 [Cs]*. Retrieved from http://arxiv.org/abs/1809.03568

Appendix: Python Model Implementations

```
import time
start = time.process_time()
import pandas as pd
import nltk

import numpy as np
from matplotlib import pyplot as plt

import re
regex = re.compile('[^a-zA-Z ]')

from nltk.corpus import stopwords
stops = set(stopwords.words("english"))
print('Loaded Libraries: ' + str(time.process_time() - start))

def max_index(choices):
    best = 0
    for i in range(1, 4):
        if choices[best] < choices[i]:
            best = i

    return best

def important_words(text):
    return [i for i in nltk.word_tokenize(text) if i not in stops and len(i) > 1]

def answerkey_to_number(answerkey):
    if answerkey in ['A', '1']:
        return 0

    if answerkey in ['B', '2']:
        return 1

    if answerkey in ['C', '3']:
        return 2

    if answerkey in ['D', 'E', '4', '5']:
        return 3

def analyze_questiontext(questiontext):
    letters = ['(A)', '(B)', '(C)', '(D)']
    numbers = ['(1)', '(2)', '(3)', '(4)']

    dividers = letters if '(A)' in questiontext else numbers

    sections = []
    for divider in dividers:
        split = questiontext.split(divider)
        sections.append(split[0].strip())
        questiontext = split[1]
    sections.append(questiontext.strip())

    return {'questiontext':sections[0],
            'choices': [sections[1], sections[2], sections[3], sections[4]]}

start = time.process_time()

dev_df = pd.read_csv('ARC-V1-Feb2018-2\ARC-Easy\ARC-Easy-Dev.csv')
```

```
train_df = pd.read_csv('ARC-V1-Feb2018-2\ARC-Easy\ARC-Easy-Train.csv')
test_df = pd.read_csv('ARC-V1-Feb2018-2\ARC-Easy\ARC-Easy-Test.csv')


dev = []
for i in range(len(dev_df)):
    q = dict(dev_df.iloc[i])
    questiontext = q['question']

    # Check if question has 4 choices
    if (all(i in questiontext for i in ['(A)', '(C)', '(B)', '(D)']) and '(E)' not in q
uestiontext) or (all(i in questiontext for i in ['(1)', '(2)', '(3)', '(4)']) and '(5)'
 not in questiontext):
        question_data = analyze_questiontext(questiontext)
        q['questiontext'] = question_data['questiontext']
        q['choices'] = question_data['choices']
        q['answer'] = answerkey_to_number(q['AnswerKey'])
        dev.append(q)

train = []
for i in range(len(train_df)):
    q = dict(train_df.iloc[i])
    questiontext = q['question']

    # Check if question has 4 choices
    if (all(i in questiontext for i in ['(A)', '(C)', '(B)', '(D)']) and '(E)' not in q
uestiontext) or (all(i in questiontext for i in ['(1)', '(2)', '(3)', '(4)']) and '(5)'
 not in questiontext):
        question_data = analyze_questiontext(questiontext)
        q['questiontext'] = question_data['questiontext']
        q['choices'] = question_data['choices']
        q['answer'] = answerkey_to_number(q['AnswerKey'])
        train.append(q)

test = []
for i in range(len(test_df)):
    q = dict(test_df.iloc[i])
    questiontext = q['question']

    # Check if question has 4 choices
    if (all(i in questiontext for i in ['(A)', '(C)', '(B)', '(D)']) and '(E)' not in q
uestiontext) or (all(i in questiontext for i in ['(1)', '(2)', '(3)', '(4)']) and '(5)'
 not in questiontext):
        question_data = analyze_questiontext(questiontext)
        q['questiontext'] = question_data['questiontext']
        q['choices'] = question_data['choices']
        q['answer'] = answerkey_to_number(q['AnswerKey'])
               test.append(q)

       print('Loaded Question Data: ' + str(time.process_time() - start))

       answers = set([i['AnswerKey'] for i in test])
       categories = set([i['category'] for i in test])
       exams = set([i['examName'] for i in test])
       has_diagram = set([i['includesDiagram'] for i in test])
       is_multiple_choice = set([i['isMultipleChoiceQuestion'] for i in test])
       grades = set([i['schoolGrade'] for i in test])
       subjects = set([i['subject'] for i in test])
       possible_points = set([i['totalPossiblePoint'] for i in test])
       years = set([i['year'] for i in test])

```

```
114.
115.       start = time.process_time()
116.
117.       sentences = [line.rstrip('\n') for line in open('ARC-V1-Feb2018-
    2\ARC_Corpus.txt', encoding='utf8')]
118.
119.       print('Loaded Sentences: ' + str(time.process_time() - start))
120.
121.       def fast_tokenize_sentences():
122.           start = time.process_time()
123.           for i in range(len(sentences)):
124.               if i % 146218 == 0:
125.                   print(i * 100.0 / 14621856)
126.                   print('Time: ' + str(time.process_time() - start))
127.               sentences[i] = set([i for i in regex.sub('', sentences[i]).split() if i
    not in stops and len(i) > 1])
128.           print('Tokenized Sentences: ' + str(time.process_time() - start))
129.
130.       def tokenize_sentences():
131.           start = time.process_time()
132.           for i in range(len(sentences)):
133.               if i % 14622 == 0:
134.                   print(i * 100.0 / 14621856)
135.                   print('Time: ' + str(time.process_time() - start))
136.               sentences[i] = important_words(sentences[i])
137.           print('Tokenized Sentences: ' + str(time.process_time() - start))
138.
139.       fast_tokenize_sentences()
140.
141.       # Control Group
142.       start = time.process_time()
143.
144.       predictions = []
145.       answers = [i['answer'] for i in test]
146.
147.       counter = 0
148.       for question in test[:1000]:
149.           if counter % 100 == 0:
150.               print(counter)
151.           counter = counter + 1
152.           scores = [0 for i in range(4)]
153.           question_tokens = set(important_words(question['questiontext']))
154.           choice_tokens = [set(important_words(x)) for x in question['choices']]
155.
156.           for sentence in sentences[:1000000]:
157.               for i in range(4):
158.                   if (not sentence.isdisjoint(question_tokens)) and (not sentence.isdi
    sjoint(choice_tokens[i])):
159.                       scores[i] = scores[i] + 1
160.
161.           predictions.append(max_index(scores))
162.
163.       accuracy = 0
164.       for i in range(len(predictions)):
165.           if predictions[i] == answers[i]:
166.               accuracy = accuracy + 1
167.
168.       print('Control Accuracy: ' + str(accuracy) + ' out of ' + str(len(predictions)))

169.       print('Control Time: ' + str(time.process_time() - start))
170.
```

```
171.        # Experimental Group
172.        start = time.process_time()
173.
174.        exp_predictions = []
175.        exp_answers = [i['answer'] for i in test]
176.
177.        counter = 0
178.        for question in test[:100]:
179.            if counter % 10 == 0:
180.                print(counter)
181.            counter = counter + 1
182.            scores = [0 for i in range(4)]
183.            question_tokens = set(important_words(question['questiontext']))
184.            choice_tokens = [set(important_words(x)) for x in question['choices']]
185.
186.            for sentence in sentences[:5000000]:
187.                for i in range(4):
188.                    if (not sentence.isdisjoint(question_tokens)) and (not sentence.isdi
    sjoint(choice_tokens[i])):
189.                        scores[i] = scores[i] + 1
190.                        for y in sentence.union(question_tokens):
191.                            scores[i] = scores[i] + 1
192.                        for z in sentence.union(choice_tokens[i]):
193.                            scores[i] = scores[i] + 1
194.
195.            exp_predictions.append(max_index(scores))
196.
197.        exp_accuracy = 0
198.        for i in range(len(exp_predictions)):
199.            if exp_predictions[i] == exp_answers[i]:
200.                exp_accuracy = exp_accuracy + 1
201.
202.        print('Exp_Accuracy: ' + str(exp_accuracy) + ' out of ' + str(len(exp_prediction
    s)))
203.        print('Exp Time: ' + str(time.process_time() - start))
204.
205.        def histogram(data, title, x_label, y_label):
206.            # fixed bin size
207.            #bins = np.arange(-100, 100, 5) # fixed bin size
208.
209.            #plt.xlim([min(data)-5, max(data)+5])
210.
211.            plt.hist(data, alpha=0.5) # bins=bins
212.            plt.title(title)
213.            plt.xlabel(x_label)
214.            plt.ylabel(y_label)
215.
216.            plt.show()
217.
218.
219.        for name in tests:
220.            print('--------')
221.            print(name)
222.
223.            count = 0
224.            total = 0
225.            for i in range(1000):
226.                if test[i]['examName'] == name:
227.                    total = total + 1
228.                    if predictions[i] == answers[i]:
229.                        count = count + 1
```

```
230.
231.           print('CTRL: ' + str(count*100.0/total) + '% (' + str(count) + ' out of ' +
    str(total) + ')')
232.
233.           count = 0
234.           total = 0
235.           for i in range(100):
236.               if test[i]['examName'] == name:
237.                   total = total + 1
238.                   if exp_predictions[i] == exp_answers[i]:
239.                       count = count + 1
240.
241.           print('EXP: ' + str(count*100.0/total) + '% (' + str(count) + ' out of ' + s
    tr(total) + ')')
```