Machine Comprehension for Question Answering

Jamie Brandon
Brandeis University
jbrandon@brandeis.edu

Abstract

I present a model architecture for predicting whether or not a question is answeable given its context using the SQuAD dataset as a corpus. Given a question and context from the dataset, my model does not predict the span of the answer found in the context, but only predicts whether or not the question can be answered from the context. This project shows the necessity of finding a possible answer first, then comparing the question and answer to predict whether it is answerable, rather than predicting solely from the context. This work yields a no-answer accuracy of 48% on unseen data, while other works that do (Hu et al., 2018) have no-answer accuracies in the mid-seventies.

1 Introduction

Machine comprehension tasks usually require an evidence text, called a passage, and a question as input. The model predicts the location in the passage that contains a relevant answer to the question. There are many challenges that arise for picking the correct location. For instance, sometimes co-reference resolution must first take place before the machine has a chance to answer the question successfully. For example, *passage: Colorado has beautiful mountains. This makes it a popular location for skiing* and *question: what is a popular location for skiing?* First the model must resolve *it* to refer to *Colorado*, or more refined, *the Colorado mountains*. Other challenges include passages in which the answer to the question is not there.

The models built to solve machine comprehension tasks vary wildly. There are many different techniques that all yielded top results at their respective time of submission. Many employ BiLSTMs, CNNs, RNNs, and nearly all of them use an attention layer to align the question and answer. In order to pick out the relevant portions of the question and answer, attention seems to perform the best for this task.

To evaluate models built to answer questions, two main metrics are used: the standard f1 score and Exact Match (EM). Just as in other machine learning models, the f1 is a weighted average of precision and recall. The EM score "measures the percentage of predictions that match any one of the ground truth answers exactly." (Rajpurkar et al., 2016) f1 scores are always higher, as EM scores need to be perfect to be counted as correct.

For my final project, I hope to use the SQuAD dataset to build a model with an attention layer that predicts the best span to answer the question provided. There are other interesting areas I'd like to explore, but I take a particular interest in passages which do no contain the answer to the question. Getting a model to accurately say "I don't have what I need to answer that question" is an important task that seems like it gets overlooked often. The SQuAD 2.0 data set offers a collection of questions, both unanswerable and answerable to fine-tune a model to. I hope that my work could be used by assistant technologies like Amazon's Alexa or the Google Home.

2 Relevant Work

The papers tackling Machine Reading Comprehension on the SQuAD data set can be divided into two categories: those attempting to find the answer span *assuming* that the answer must be in the context (using SQuAD 1.0 data set) and those attempting to give a "no-answer" response in the cases in which an answer cannot be found in the context.

Of those making the assumption that the answer will be in the context, many approaches use (bi-)LSTMs - including (Wang et al., 2018), (Liu

et al., 2017), (Hu et al., 2018), (Huang et al., 2017). CNNs are employed by (Wang et al., 2018), (Han et al., 2018). Others take a more simple approach. The authors of (Rajpurkar et al., 2016) use a logistic regression model on features including dependency tree paths and POS of the question word.

Those that also ask the model to "know what it doesn't know" include (Hu et al., 2018), though they employ many of the same architectures to find the span of the answer with an additional layer to verify that the answer found does indeed answer the proposed question.

This work directly contrasts and highlights that of (Hu et al., 2018). In this work, the authors first find the likely answer span given both the question and context. After finding the most likely answer span, they then use attention to predict whether or not this likely answer span does answer the question. My work proves how crucial this task is. In my work, I do not predict the answer span, but rather predict whether or not the context can answer the question without first finding the most likely answer span. As expected, this model performs worse than models that have the likely answer span provided.

3 Method

Details for model implementation are found in this section.

3.1 Data Processing

Data from SQuAD is formatted in JSON. First, the data is parsed, then each question gets an object of type $Instance$. This $Instance$ holds exactly one question and one context and a list of possible answers. It also holds the target variable $is - impossible$. Note that the same context has multiple questions, so many $Instance$ objects hold the same context. Since the context is stored in multiple $Instance$ objects and preprocessed multiple times, it is open to improvement. While unfortunate, the time to load and preprocess the data is not crucially affected and completes within 4 minutes consistently.

Note that the data had to be trimmed in order for experiments to complete in a timely manner. For this set of experiments, I used 10,000 instances of data. That is, I used 10,000 different questions that may have come from the same contexts. With more data, I expect better performance. Without access to GPU enabled machines, this was the that could be done within the time contraints.

```
def load_json_to_array(data_file: str):
    """
    load the data into memory
    :param data_file: the file path of the data to load in
    :return: a list of Instance objects
    """
    arr = []
    with open(data_file) as f:
        dataset_json = json.load(f)
        dataset = dataset_json['data']
        for wiki_article in dataset:
            title = wiki_article['title']
            for paragraph in wiki_article['paragraphs']:
                context = paragraph['context']
                qas = paragraph['qas']
                for qa in qas:
                    i = Instance(answers=qa['answers'],
                                 id=qa['id'],
                                 is_impossible=qa['is_impossible'],
                                 question=qa['question'],
                                 context=context,
                                 title=title,
                                 )
                    arr.append(i)
    return arr
```

Figure 1: Method to load data from JSON format in SQuAD to an array

After loading the data in, a Vocabulary object is built from MXNet using pre-trained embeddings from Wikipedia. Random vectors are initialized for special tokens like $< unk >, < pad >, < bos >, < eos >$. I opted to use the 100 dimensional vectors, as they worked well in my previous projects with future work plans to explore other dimensions or even using context-specific embeddings like ELMO or BERT.

I tokenize the context and question by splitting on white space and taking the word in an a case-insensitive format. The context is then padded with a maximum length of 300, the question with a maximum length of 40.

```
def process_text(words, vocab, max_len):
    indices = vocab[words]  ## map tokens (strings) to unique IDs
    indices = indices[:max_len]  ## truncate to max_len
    # pad if necessary
    while len(indices) < max_len:
        indices.append(vocab['<pad>'])
    assert len(indices) == max_len
    return mx.nd.array(indices)
```

Figure 2: Text preprocessing before the Neural Network

Whether or not the question is answerable is encoded with integers as 0 for unanswerable and 1 for answerable.

3.2 Model Architecture

Following many of the academic publications achieving state-of-the-art performance on this

task, the model employs a Bi-LSTM on both the question and the context. The model has one attention layer, and finally a fully connected layer mapping to two classes for the outputs.

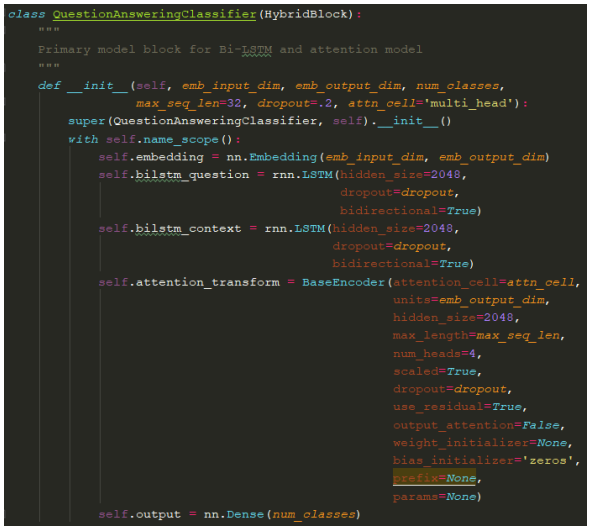

Figure 3: Model Architecture Definition

When calling the attention layer, note that two encoded representations are concatenated together before being fed into the output layer. First, the question is used as the query and the context as the key, and vice versa. Using these two representations together will provide more information to the model than exclusively one or the other.

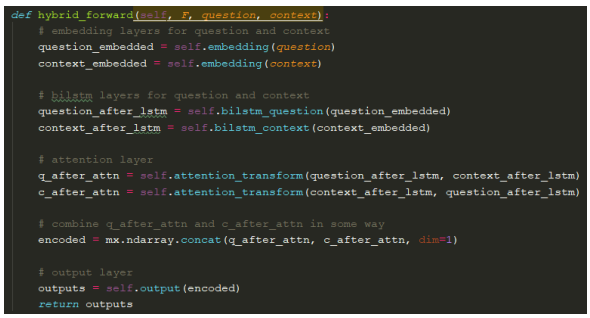

Figure 4: The Hybrid Forward Method

Unfortunately, even with this relatively simple architecture, the model needed a significant amount of time to run, even on the truncated 10,000 instances. To get through one epoch, the model needed about 8 hours. It's possible that GPU enabled machines would speed up this run time significantly.

4 Results

After about 24 hours of training with only 3 epochs, I present the following results.

Epoch	Train loss	Train Acc	Val Acc
1	377.6983	0.8188	0.4976
2	280.7023	0.8438	0.4988
3	207.3620	0.8322	0.4972

Note that in the first 10,000 instances, only 1756 cannot be answered. While the classifier isn't necessarily performing well, it is not the case that it classifies all instances as the majority class.

It is unfair to compare directly with (Hu et al., 2018), as I was unable to run my experiment with the entirety of the data set. However, as a quick comparison, the no-answer accuracy scores for their models were in the mid-seventies, with the highest accuracy 76.2% on a hidden test set.

While it is possible that my model would increase its accuracy with more data, more would need to be done to optimize the training process. Access to GPUs may make the runtime for this model more reasonable, though perhaps a more efficient setup may help as well.

5 Conclusion

Many different approaches are used to tackle the question answering that data sets like SQuAD provide. The best approaches use attention in combination with BiLSTMs to properly align the question and passage so that the model is able to extract the relevant information instead of overloading with unimportant words. From this work, it is apparent that first extracting the possible answer improves this no answer accuracy significantly. Making a model that can correctly predict answers is incredibly useful for user experience in searching for information, and it lets users do this faster and more effectively.

6 Future Work

With more time, I would like to expand this work by training my own word embeddings on the SQuAD Dataset. In other projects, I have found that training my own word embeddings increases the performance of the model, even if only barely. I have not used context-specific embeddings like BERT or ELMO, but this avenue also seems advantageous for this task.

Additionally, predicting the span first then using that span to predict answerabiilty is a natural next

step following the work of (Hu et al., 2018). Measuring how much the model changes with those advances will provide invaluable insight.

Lastly, it is absolutely necessary to have GPU support for tasks this large. The dataset is massive, and it's a shame to have to throw away data in order to get results in time. In the future, having someone look over the code for efficiencies and setting up for GPU effectiveness would greatly benefit this effort.

7 Acknowledgements

I give full credit to Ben Wellner for the source code provided in a previous assignment that set up the attention layers. Without this starter code, it would've taken me much longer to achieve results (and understand the attention mechanism).

References

Xu Han, Pengfei Yu, Zhiyuan Liu, Maosong Sun, and Peng Li. 2018. Hierarchical relation extraction with coarse-to-fine grained attention. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2236–2245.

Minghao Hu, Yuxing Peng, Zhen Huang, Nan Yang, Ming Zhou, et al. 2018. Read+ verify: Machine reading comprehension with unanswerable questions. *arXiv preprint arXiv:1808.05759*.

Hsin-Yuan Huang, Chenguang Zhu, Yelong Shen, and Weizhu Chen. 2017. Fusionnet: Fusing via fully-aware attention with application to machine comprehension. *arXiv preprint arXiv:1711.07341*.

Xiaodong Liu, Yelong Shen, Kevin Duh, and Jianfeng Gao. 2017. Stochastic answer networks for machine reading comprehension. *arXiv preprint arXiv:1712.03556*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Wei Wang, Ming Yan, and Chen Wu. 2018. Multi-granularity hierarchical attention fusion networks for reading comprehension and question answering. *arXiv preprint arXiv:1811.11934*.

A Appendices

Find code snippets in larger context below.

Column intentionally left blank

```
def load_json_to_array(data_file: str):
    """
    load the data into memory
    :param data_file: the file path of the data to load in
    :return: a list of Instance objects
    """
    arr = []
    with open(data_file) as f:
        dataset_json = json.load(f)
        dataset = dataset_json['data']
        for wiki_article in dataset:
            title = wiki_article['title']
            for paragraph in wiki_article['paragraphs']:
                context = paragraph['context']
                qas = paragraph['qas']
                for qa in qas:
                    i = Instance(answers=qa['answers'],
                                 id=qa['id'],
                                 is_impossible=qa['is_impossible'],
                                 question=qa['question'],
                                 context=context,
                                 title=title,
                                 )
                    arr.append(i)
    return arr
```

Figure 5: Load Data

Column intentionally left blank

```
def process_text(words, vocab, max_len):
    indices = vocab[words]  ## map tokens (strings) to unique IDs
    indices = indices[:max_len]  ## truncate to max_len
    # pad if necessary
    while len(indices) < max_len:
        indices.append(vocab['<pad>'])
    assert len(indices) == max_len
    return mx.nd.array(indices)
```

Figure 6: Text preprocessing before the Neural Network

```
def hybrid_forward(self, F, question, context):
    # embedding layers for question and context
    question_embedded = self.embedding(question)
    context_embedded = self.embedding(context)

    # bilstm layers for question and context
    question_after_lstm = self.bilstm_question(question_embedded)
    context_after_lstm = self.bilstm_context(context_embedded)

    # attention layer
    q_after_attn = self.attention_transform(question_after_lstm, context_after_lstm)
    c_after_attn = self.attention_transform(context_after_lstm, question_after_lstm)

    # combine q_after_attn and c_after_attn in some way
    encoded = mx.ndarray.concat(q_after_attn, c_after_attn, dim=1)

    # output layer
    outputs = self.output(encoded)
    return outputs
```

Figure 7: The Hybrid Forward Method