University of New Haven Tagliatela college of Engineering

Master’s in Data Science

Course: Natural Language Processing

Title:

QUESTION ANSWERING USING SEQUENCE-TO-SEQUENCE RECURRENT NEURAL NETWORKS

Names:
1.Venkata Harish Gogineni
2.Pavan Kumar Gonna
3. Lavanya Chadalawada

Under the guidance of,
Prof. Vahid Behzadan , Ph.D.
vbehzadan@newhaven.edu

1 Introduction

Question Answering a task in NLP that takes input as a questions of different types and outputs answers for these questions. The questions can be of various types such as Open domain vs closed domain. In open domain, factual and non-factual etc., The answers can also be of different types such as one word answers, selecting answers from a text inputted, yes or no questions, etc. In our project, we have considered answering the Open domain question answering using sequence to sequence encoders and decoders. We have used Bidirectional LSTMS in both encoders and decoders. The open domain question answering has two parts. The first part is the Retriever module that gets documents from the web to answer the questions. The reader module is then used to answer the questions based on the retrieved documents. Both of these tasks have to be trained.

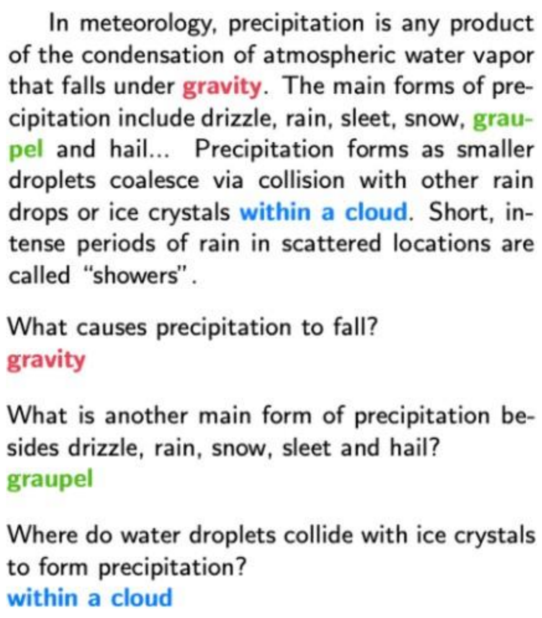
2 Method

1. Data Set

SQUAD dataset has over 100000 questions for Reading comprehension task. This dataset was developed by the Sandford. Here every answer to the question is a segment of text from the corresponding reading passage. In the below example, we can see that the passage is given in which the answers are present. When a question is asked based on the passage, a single answer like the one indicated in red or green is or a spam of answers like the one indicated in blue is returned.

<u>Features of SQuAD:</u>

SQuAD dataset is closed. This means that the answer is a part of the reading comprehension given. 75% of the questions are lesser than 4 words. Most of them are single word answers. In SQUAD version 1, there are no answers for all questions. Only some are answered and others are null.



In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under **gravity**. The main forms of precipitation include drizzle, rain, sleet, snow, **graupel** and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals **within a cloud**. Short, intense periods of rain in scattered locations are called "showers".

What causes precipitation to fall?
gravity

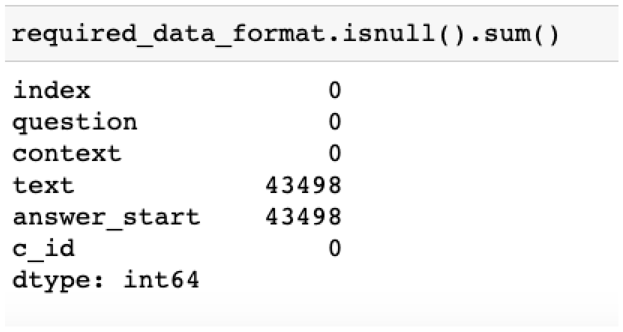
What is another main form of precipitation besides drizzle, rain, snow, sleet and hail?
graupel

Where do water droplets collide with ice crystals to form precipitation?
within a cloud

Figure 1: Example of SQUAD

2. Pre-Processing and preparation

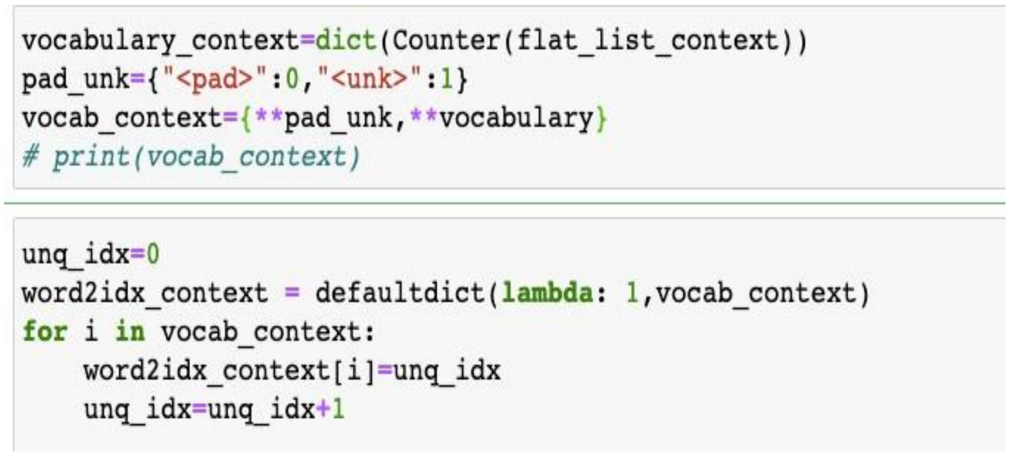
Firstly the squad dataset has a lot of Null values. These null values were simply removed as a part of Data Pre-processing.



```
required_data_format.isnull().sum()

index               0
question            0
context             0
text            43498
answer_start    43498
c_id                0
dtype: int64
```

Figure 2: Null values in SQUAD

Then, the information data is divided into list of questions and also list of paragraphs. From these list of paragraphs and contexts, are converted into simple vocabulary. In the vocabulary of both the Questions and contexts, we add padding and unknown tags. With the obtained paragraph and question vocabulary, we build a word to index mapping using a simple incrementing indexing method. Here each word in the dataset is encoded into a number. The below is an example of how the paragraph/context words are transformed into a dictionary with the pad and unknown tags included. Every word in the dictionary as keys has values as its index.



```
vocabulary_context=dict(Counter(flat_list_context))
pad_unk={"<pad>":0,"<unk>":1}
vocab_context={**pad_unk,**vocabulary}
# print(vocab_context)
```

```
unq_idx=0
word2idx_context = defaultdict(lambda: 1,vocab_context)
for i in vocab_context:
    word2idx_context[i]=unq_idx
    unq_idx=unq_idx+1
```

Figure 3: Implementation of word to index

3. Implementation

- **Model**

  Open domain question answering has two parts:

  1) **Retriever module:**

     In the paper we have considered, we aim to develop a recurrent graph-based retrieval model which is trained to retrieve documents as reasoning paths. In this approach a history of retrieved documents are stored. With this information new and relevant documents are obtained. Here, a Wikipedia paragraph graph is constructed. This graph is obtained from creating hyperlinks from one paragraph to another. The links are symmetric that means, it can hop from one paragraph to another. The graphs are also densely connected which helps in retrieving many relevant information.

A recurrent neural network is used to learn the reasoning path. The first hidden state of the RNN does not depend on any questions or answers. The network selects the relation between the many paragraphs by conditioning on the history of the selections. When a terminate symbol is encountered, this process comes to an end. Beam search is later used to select the right paragraphs from the outputted probability distribution

$$\begin{aligned} w_i &= \mathrm{BERT}_{[\mathrm{CLS}]}(q, p_i) \in \mathbb{R}^d, \\ P(p_i|h_t) &= \sigma(w_i \cdot h_t + b), \\ h_{t+1} &= \mathrm{RNN}(h_t, w_i) \in \mathbb{R}^d, \end{aligned}$$

Figure 4: Implementation of recurrent retriever

2) **Reader module:**
Here, the question and the contexts are encoded. These are encoded using two BiLSTMs. The outputs of the BiLSTMs are two hidden states. These states are put in another encoder using sequence to sequence attention. The attention is performed on the hidden states of the question and the contexts or the paragraphs. The product is sent to another layer of LSTM to generate the weighted context. The output is concatenated with the paragraph hidden sate. Now we have the information about which parts which are most important between context and paragraphs.

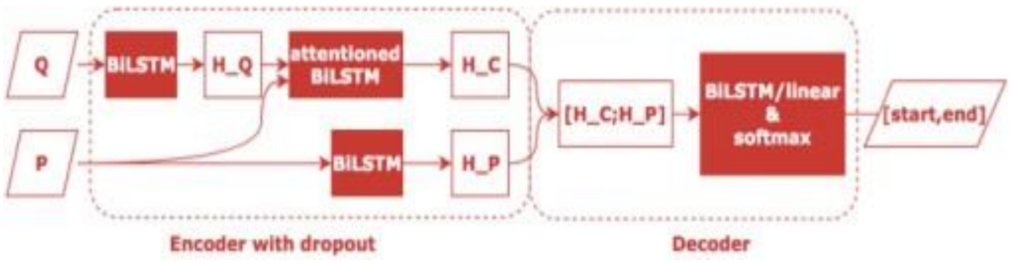


Figure 5: Diagram of the Reader architecture

Firstly, the questions and paragraphs are divided into batches. Each batch is a set of sentences. Every batch first goes into a padding function that returns the length of the questions and length of the paragraphs after padding. Padding is done because, every question or paragraph in a batch will not be of same length. In order to make them also into a vectors of equal length, we make all them all equal size by appending zeros for the shorter sentences. Every batch also returns the start index of the answer found in the training dataset.

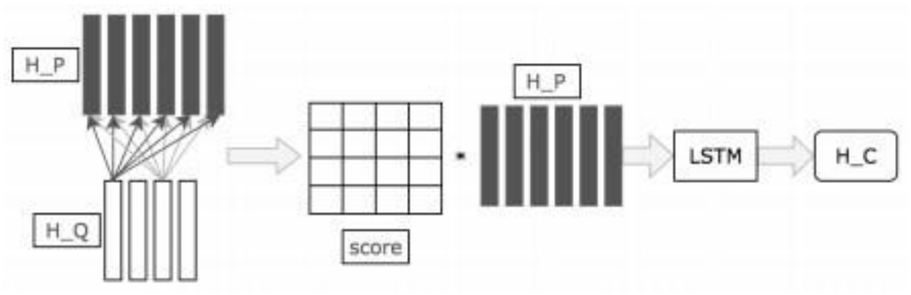


Figure 6: Attention encoding

The next step would be to concatenate the hidden outputs from the encoder. We then feed this concatenated input into a decoder. A decoder has 2 bidirectional LSTM networks, one for the start index and one for the end index. We take the largest element in each predicted index. We then use softmax and followed by cross-entropy loss to find the loss. Then do the gradient descent to minimise the loss between the predicted and actual value.

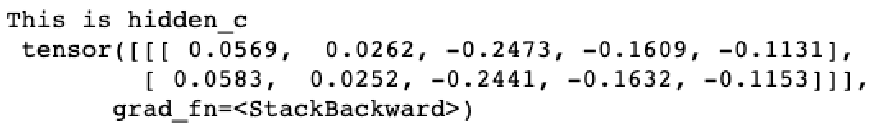
For example,

SENTENCE	**SEQUENCE LENGTH**	**PADDED**
This is a dog	4	This is a dog 0 0 0
The girl has long hair	5	The girl has long hair 0 0
Good cat	2	Good cat 0 0 0 0 0
This is the longest of all sentences	7	This is the longest of all sentences

Every batch obtained also goes through a get_Converted_Sentence_And_Labels function that converts every word that was encoded using the incremental indexes.

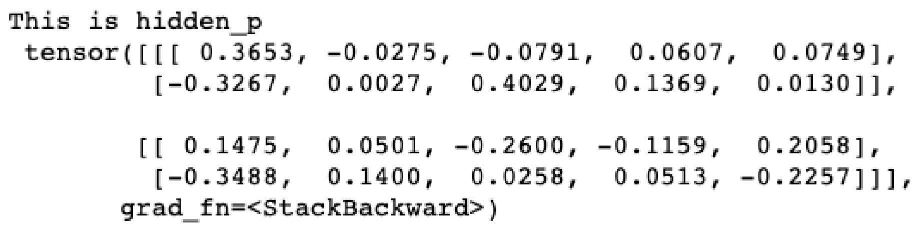
3 Results

The models produce the following results. The following results are obtained by training the encoder and decoder modules with hidden layer size 2 and the batch size 2.



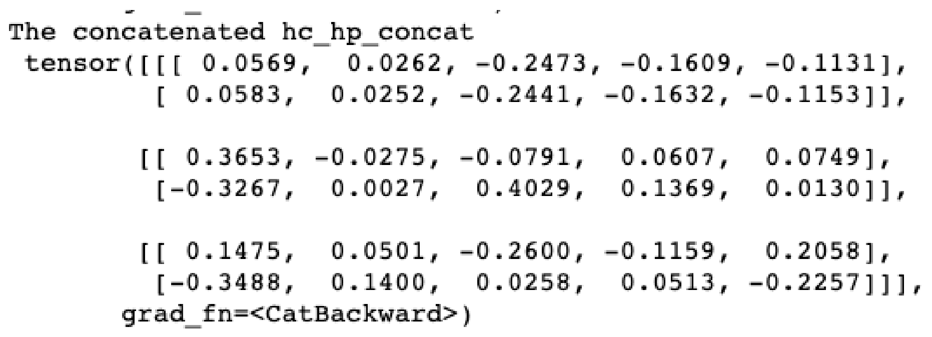
```
This is hidden_c
 tensor([[[ 0.0569,  0.0262, -0.2473, -0.1609, -0.1131],
          [ 0.0583,  0.0252, -0.2441, -0.1632, -0.1153]]],
       grad_fn=<StackBackward>)
```

Figure 6: Output from the encoder model – hidden_c



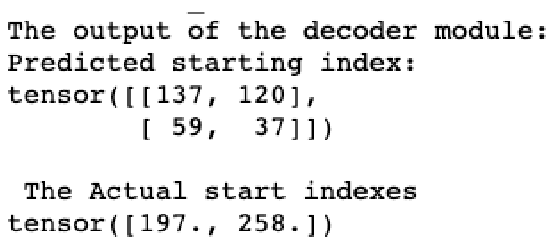
```
This is hidden_p
 tensor([[[ 0.3653, -0.0275, -0.0791,  0.0607,  0.0749],
          [-0.3267,  0.0027,  0.4029,  0.1369,  0.0130]],

         [[ 0.1475,  0.0501, -0.2600, -0.1159,  0.2058],
          [-0.3488,  0.1400,  0.0258,  0.0513, -0.2257]]],
       grad_fn=<StackBackward>)
```

Figure 7: Output from the encoder module – hidden_p



```
The concatenated hc_hp_concat
 tensor([[[ 0.0569,  0.0262, -0.2473, -0.1609, -0.1131],
         [ 0.0583,  0.0252, -0.2441, -0.1632, -0.1153]],

        [[ 0.3653, -0.0275, -0.0791,  0.0607,  0.0749],
         [-0.3267,  0.0027,  0.4029,  0.1369,  0.0130]],

        [[ 0.1475,  0.0501, -0.2600, -0.1159,  0.2058],
         [-0.3488,  0.1400,  0.0258,  0.0513, -0.2257]]],
       grad_fn=<CatBackward>)
```

Figure 8: Before inputting the output of encoder to the decoder



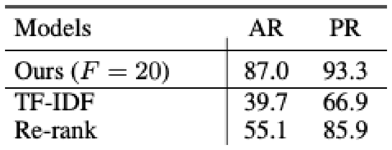
```
The output of the decoder module:
Predicted starting index:
tensor([[137, 120],
        [ 59,  37]])

 The Actual start indexes
tensor([197., 258.])
```

Figure 9: Predicted start indexes and Actual indexes. The batch size in this output is 2.

The loss function to be implemented is a simple cross entropy loss between the actual and predicted values of start indexes which is the point in the passage where the answer begins.

4 Conclusion

This method of using Wikipedia graph to retrieve the documents is much better than the other methods like TF-IDF which simply selects top two paragraphs. The re-rank method selects top 2 sentences after ranking. The recurrent retrieval has shown significant improvement than the semantic retrieval. The work has been conducted and reported https://arxiv.org/abs/1911.10470. Different methods of evaluation are used in the paper – Answer recall, Paragraph recall and others which showed significant increase in the scores. We can see that the answer and precision recalls were very less for TF-IDF and Re-rank methods.



Models	AR	PR
Ours ($F = 20$)	87.0	93.3
TF-IDF	39.7	66.9
Re-rank	55.1	85.9

Figure 10: AR: Answer recall and PR: Precision recall

REFERENCES:

- Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. arXiv:1607.06450, 2016.
- Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading Wikipedia to answer open-domain questions. In ACL, 2017.
- Christopher Clark and Matt Gardner. Simple and effective multi-paragraph reading comprehension. In ACL, 2018.
- Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, and Andrew McCallum. Multi-step retriever-reader interaction for scalable open-domain question answering. In ICLR, 2019.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In NAACL, 2019.
- Ming Ding, Chang Zhou, Chang Zhou, Qibin Chen, Hongxia Yang, and Jie Tang. Cognitive graph for multi-hop reading comprehension at scale. In ACL, 2019.
- Yair Feldman and Ran El-Yaniv. Multi-hop paragraph retrieval for open-domain question answering. In ACL, 2019.
- Paolo Ferragina and Ugo Scaiella. Fast and accurate annotation of short texts with wikipedia pages. IEEE software, 29(1):70–75, 2011.
- Ameya Godbole, Dilip Kavarthapu, Rajarshi Das, Zhiyu Gong, Abhishek Singhal, Xiaoxiao Yu, Mo Guo, Tian Gao, Hamed Zamani, Manzil Zaheer, and Andrew McCallum. Multi-step entity- centric information retrieval for multi-hop question answering. In Proceedings of the 2nd Work- shop on Machine Reading for Question Answering, 2019.
- Minghao Hu, Yuxing Peng, Zhen Huang, and Dongsheng Li. Retrieve, read, rerank: Towards end- to-end multi-document reading comprehension. In ACL, 2019.
- Hakan Inan, Khashayar Khosravi, and Richard Socher. Tying word vectors and word classifiers: A loss framework for language modeling. In ICLR, 2017.
- Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In ICLR, 2015. Bernhard Kratzwald and Stefan Feuerriegel. Adaptive document retrieval for deep question answer-ing. In EMNLP, 2018.
- Tom Kwiatkowski, Jennimaria Palomaki, Olivia Rhinehart, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, et al. Natural ques- tions: a benchmark for question answering research. TACL, 2019.
- Jinhyuk Lee, Seongjun Yun, Hyunjae Kim, Miyoung Ko, and Jaewoo Kang. Ranking paragraphs for improving answer recall in open-domain question answering. In EMNLP, 2018.
- Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. Latent retrieval for weakly supervised open domain question answering. In ACL, 2019.
- Yankai Lin, Haozhe Ji, Zhiyuan Liu, and Maosong Sun. Denoising distantly supervised open- domain question answering. In ACL, 2018.
- Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized bert pretraining approach. arXiv:1907.11692, 2019.