Question Answering from Wikipedia articles

A Reading Comprehension task

Garima Gupta
Department of Computer Science
University of Illinois at Chicago, Illinois, USA
ggupta22@uic.edu

ABSTRACT

This project is an attempt to answer questions posed to a set of articles from Wikipedia using a machine learning model. The model I chose uses layers of bidirectional long-short-term memory and attention mechanism on the paragraphs and questions to capture the relationships among different words which then helps locate the answer to a factoid question in the text span of the Wikipedia article. This is a part implementation of the paper by Danqi Chen et al. [1] which describes a state-of-the-art model to tackle open domain questions using Wikipedia articles. The paper proposes a two-part model, first being a document retriever which locates all articles relevant to a particular question, and second part being a document reader which is RNN and attention machine learning model to identify answer to the questions. This project only tries to implement the document reader assuming the relevant article is already known. With a few missing components, this implementation achieves a 0.184 F1 score.

CONCEPTS

• Deep Learning • Neural Networks • Natural Language Processing

KEYWORDS

Bi-LSTM, Attention Mechanism, Machine Comprehension, Question Answering

INTRODUCTION

We are surrounded by data, be it textual, visual, audio, sensor, tabular or statistical. New data is created every second and the need to understand it evidently increasing in every business, industry, science and research. One example is the enormous collection of articles on Wikipedia which are written in natural language. An internet search for a question most probably gets at least a few hits in the Wikipedia repository which just proves that the information that constitutes these articles has proven to be immensely valuable pertaining to the huge variety of subject matters.

Question answering is a natural language processing discipline which aims at answering questions posed in natural language. The system takes a natural language question as input and converts it to a query which represents not just the keywords in the question, but also the 'meaning' of the question as understood by the system.

Most of the current question answering models today approach the task as a reading comprehension wherein the model tries to 'understand' the question and the paragraph it is posed from, which means that the model tries to capture the context of the question and search it in the paragraph. When it finds the relevant context, it then tries to extract the span of the text in that paragraph which is returned as the most probable answer to the question.

Wikipedia has been a source of information to wide variety of topics and has become a very common reference with up-to-date knowledge. But it has been written for humans to read rather than machines. This paper attempts to leverage the power of Wikipedia to be used by machines in answering open-domain questions which, with high probability, have been answered in articles on the platform. A lot of question answering systems today rely heavily on the redundancy of information in their knowledge bases which is leveraged in extracting answer to the questions. But using a source without redundancy inspires high precision in the system because if not, the system would miss the only location an answer could be found. The paper proposes a two-step model to overcome this issue and perform the task of question answering. First is that of a Document Retriever which uses bigram hashing and TF-IDF which is used to identify a set of relevant articles for a question. Second is a Document Reader which serves a machine comprehension task model to locate the spans of answers in a relevant article. This implementation attempts to study the Document Reader part of the proposed model and test the machine comprehension on the SQuAD dataset. When all the components in the paper are used, including Document Retriever and Document Reader, the model achieves an F1 score of 0.79. Although, this implementation tries to focus on certain main components of the Reader and is able to get an F1 score of 0.18.

RELATED WORK

In the recent years, attention model has proven to be a success as a neural network model for machine comprehension tasks, similar to the Attentive Reader in (Hermann et al., 2015) and (Chen et al., 2016). It has also been used in extractive question answering tasks as an alignment tool (Lee et al., 2016) to capture soft similarities between words used in documents and questions. The model implemented here uses similar ideas for the Document Reader model. It uses linear attention at three different levels to gather information about word similarities and dependencies in the context, uses them as encodings to be sent to a recurrent neural network as input. The output of

RNN is then fed to two separate bi-directional linear attention models which scores each word as a start and end of the answer span.

TECHNICAL APPROACH

The approach used here is that of reading comprehension of articles in which the answers to certain question are to be located. The data is in the json format which are read into dictionaries and extracted level by level. Each article forms the root of the example with multiple paragraphs. Each paragraph is posed multiple questions. Each question then has a list of answers, where each answer object consists of the answer text, which is a segment of the paragraph, along with an answer start tag indication the position in the paragraph where the answer to that question starts.

Embedding layer - GloVe

The model does not understand natural language. It cannot directly process the English language words and hence text needs to be converted so that our system can process it. The words in the paragraphs and questions need to be replaced with numerical values which correctly capture the meanings and relative context it should be used in. For this purpose, I use the GloVe embedding with 300 dimensions to convert each single word to a vector of size 300. The GloVe dataset used here has been pre-trained on a set of 6 billion words coming from Wikipedia using an unsupervised learning algorithm by the same name which results in the vector representation of those words. This forms the embedding layer of the model

The model we use here is a combination of Bidirectional Long Short Term Memory layer and linear and bilinear attention layers.

Bi-LSTM

Long Short-Term Memory (LSTM) is a type of recurrent neural network. This was introduced to tackle the problem of vanishing and exploding gradients. Its structure involves –

- Forget Gate (ft)
- Input Gate (it)
- Memory Cell (ct)
- Hidden Cell (ht)
- Output Gate (ot)

These are defined as follows:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \quad (1)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \quad (2)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad (3)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t1} + W_{co}c_t + b_o) \quad (4)$$

$$h_t = o_t \odot \tanh(c_t) \quad (5)$$

A memory cell allows for the transition of information from previous time steps to future time steps. Bi LSTM takes this one step forward by allowing us to do the same step but in the reverse direction - from future timesteps to previous timesteps. Together these help in capturing the context of the current word in both directions of a sentence.

Linear Attention

Attention mechanism form a very important component of the natural language machine learning. They are inherently one of the defining features of the deep learning models solving NLP problems. Even though they started off as being proposed for NLP, their characteristics and simply the capability to capture details have made it possible for them to be used in a wide range of machine learning problems, ranging from not only natural language but also vision and audio.

The basic idea behind attention is simple. It tries to weigh the extent to which different input features contribute towards learning some target. For example, a linear self-attention for natural language describes how closely the words in a sentence describe each other. Previously, seq2seq models were used to capture these details. But they failed to do so for longer documents because they tried to cram all the details in a single vector at the end of the encoder sequence, which might miss out on important information. The decoder for attention mechanism however, apart from the final output vector of the encoder, also takes into account the intermediate states of the encoder. A linear attention takes a linear weighted sum of these hidden states before sending it to the decoder.

We use linear attention for three different purposes in this implementation.

- Attention layer for alignment of question embedding over the context
- Self-Attention layer as question encoder
- Bi-linear attention layer as decoder which acts as an answer pointer decoder, which means that this layer is responsible in pointing the location and span of the answer to that particular question.

DOCUMENT READER MODEL

The paper proposes different encodings for the paragraph and question.

Paragraph Encoding : Each token p_i in the paragraph is first represented by 4 different features which are concatenated together to get one large vector $p_i\tilde{}$ representing token features. All such vectors are then passed through a recurrent neural network to get the final paragraph encodings $\mathbf{p_i}$.

1. Word Embeddings (f_{emb}) – These form the first set of token features. The embedding layer is uses pre-trained GloVe embeddings as explained above. To capture some crucial words, 1000 most frequent words vectors are fine-tuned, and all other vectors remain fixed.
2. Exact-match (f_{em}) – Represents whether a token can be exactly to one of the tokens in the question. This project implementation does not implement this feature.

3. Token Features (f_{token}) – Trained tag for part of speech (POS), named entity recognition tag (NER) and the term frequency (TF) of the token. This project implementation does not implement this feature.
4. Aligned Question Embedding (f_{align}) – This is one of the most important features for this task. This feature tries to represent soft alignments between a context word and a question token which maybe similar but are not exactly the same. To get this feature, we first calculate the attention scores $a_{i,j}$ which captures similarities between p_i and q_j

$$a_{i,j} = \frac{\exp\left(\alpha(\mathbf{E}(p_i)) \cdot \alpha(\mathbf{E}(q_j))\right)}{\sum_{j'} \exp\left(\alpha(\mathbf{E}(p_i)) \cdot \alpha(\mathbf{E}(q_{j'}))\right)},$$

A single dense layer with ReLU is first applied to paragraph and question word embedding and a softmax is taken to get attention scores. These scores are then used as weights to the question embeddings to get a linear attention for each paragraph token.

All these vectors are then concatenated to get a single vector representing the encoding of each paragraph token.

Question Encoding : Question encoding is much simpler than the paragraph encoding. First layer is the embedding layer to convert each word to a word vector using pre-trained glove embeddings. These embeddings are then passed through a recurrent neural network whiles saving its hidden states. These states are then used as inputs for a linear attention layer to get attention scores b_j.

$$b_j = \frac{\exp(\mathbf{w} \cdot \mathbf{q}_j)}{\sum_{j'} \exp(\mathbf{w} \cdot \mathbf{q}_{j'})},$$

A question encoding is calculated as the weighted sum of hidden units of the RNN layer with attention score b_j as the weights.

Predictions : The prediction layer is a pair of bi-directional linear attention layers, one each for the start and end tag prediction.

$$\begin{aligned} P_{start}(i) &\propto \exp\left(\mathbf{p}_i \mathbf{W}_s \mathbf{q}\right) \\ P_{end}(i) &\propto \exp\left(\mathbf{p}_i \mathbf{W}_e \mathbf{q}\right) \end{aligned}$$

DATASET

The dataset used for this project is the Stanford Question Answering Dataset (SQuAD) [2].
This dataset is used for machine comprehension and consists of questions posed on a set of Wikipedia articles. The answers to these questions are a span in the corresponding article, which means that each answer indicates a position in the paragraph where is starts and ends which makes it a segment of text. The dataset has been created by crowdsourcing the task to actual humans using Amazon Mechanical Turk.

With a total of more than 100 thousand question answer pairs on 536 uniformly randomly chosen Wikipedia articles, the dataset itself is divided into training, development and test subsets. Only training and dev sets have been used in this experiment as the test set is hidden and only accessible to the dataset curators.

The training set comprises of around 80% of the examples. Each example is composed of a pair of question and answer over a paragraph from an article.
Development set has about 10% of such examples.

EXPERIMENTS AND RESULTS

The dataset is loaded, read and preprocessed to for the training and validation data-frames. We form a data loader so as to get the data in the form of batches of size 32 as suggested in the paper. After reading the words from the data, I collect them in a dictionary which will be used as a vocabulary for training and then for validation. We also collect the start and end tags of each example. Different context paragraphs and questions have different lengths, so all the context and questions are padded to get a uniform set for processing.

As the paper suggests, the RNN layer for both paragraph and question encoding is a 3 layer bidirectional LSTM with 128 hidden units. We gather all the hidden states to be used with the weights for linear attention.

The Reader model uses the Stanford CoreNLP Toolkit for tokenization and getting the token features for paragraph encoding. This includes the part-of speech tags, named entity recognition tags and lemmatization. In this implementation, I used the spacy toolkit only for tokenization without including the token features.

We also do not implement the exact match features of the paragraph encoding.

We use Adamax optimizer for this implementation.

The result of this experiment is an F1 score of 0.185 which is very less ac compared to the baseline model for this dataset along with the original results of the authors of this paper. This difference could be due to some model omissions in this implementation such as missing features in paragraph encoding, and missing Document retriever. These features play an important role in context matching of words in paragraph and question which could in turn affect the prediction of start and end tags for answer span.

FUTURE SCOPE

This project can be extended to include the missing features and the Document retriever model. We can also experiment with different RNN layer structure or adding another CNN layer to further improve learning.

REFERENCES

[1] Danqi Chen,Adam Fisch, Jason Weston, Antoine Bordes. 2017. Reading Wikipedia to Answer Open Domain Questions (ACL). arXiv:1704.00051 **[cs.CL]**

[2] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Empirical Methods in Natural Language Processing (EMNLP)*. arXiv:1606.05250 **[cs.CL]**

[3] Karl Moritz Hermann, Toma´sˇ Kocˇisky´, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching ma- chines to read and comprehend. In *Advances in Neu- ral Information Processing Systems (NIPS)*

[4] Danqi Chen, J. Bolton, Christopher D. Manning. 2016. A Thorough Examination of the CNN/Daily Mail Reading Comprehension Task (ACL) arXiv:1606.02858 **[cs.CL]**

[5] Kenton Lee, Tom Kwiatkowski, Ankur Parikh, and Di- panjan Das. 2016. Learning recurrent span representations for extractive question answering. *arXiv preprint arXiv:1611.01436*

[6] Kushal, Pytorch Question Answering (2020). This Repository helped me debug and understand a lot of intricate details of the model and implementation. In particular, it helped in extraction of the final predictions in terms of text for comparison with the ground truth answers.
Github Repository - https://github.com/kushalj001/pytorch-question-answering