Closed Domain Question Answering System

Mathew George
B160008CS
mathewg98@gmail.com

AIM

To create a Closed Domain Question Answering System for 12th grade NCERT physics.

I. Introduction

There are primary two broad categories of Question-Answering sytems - CDQA and ODQA. CDQA stands for closed domain question answering system. In this construct, a deep learning model (in this case BERT) is trained with a dataset of documents annotated with question and answers in a closed domain so as to answer any question related to a topic which falls under that particular domain. This differs from ODQA (Open Domain Question Answering system) which does not have a fixed domain to which the questions should belong.

II. Motivation

There is a gap in technology that aids students to obtain answers to questions directly from a textbook, without having to scour through multiple pages on the internet. To fill this gap, we propose a CDQA system.

III. Literature Survey

There are multiple libraries for closed domain question answering, all differing in architecture and deep learning layers used. On evaluating our priorities, we ended up with a manual which can help us build a basic CDQA system in python [1]. Written by the creators of CDQA, the documentation gave us detailed instructions on how to create such a system. The tutorial also explained how the annotator worked and how we can annotate our dataset and create a json file which would be used to train our dataset.

IV. Design

The CDQA architecture consists of two main components: the Retriever and the Reader. When a question is sent to the system, the Retriever returns a list of documents in the dataset that are most likely to contain the answer. It works by computing TF-IDF features (Term Frequency Inverse Document Frequency) based on unigrams and bigrams and computing cosine similarity between the question and each document based on those features. After extracting the most probable documents, the documents are converted to paragraphs and are sent to the Reader.

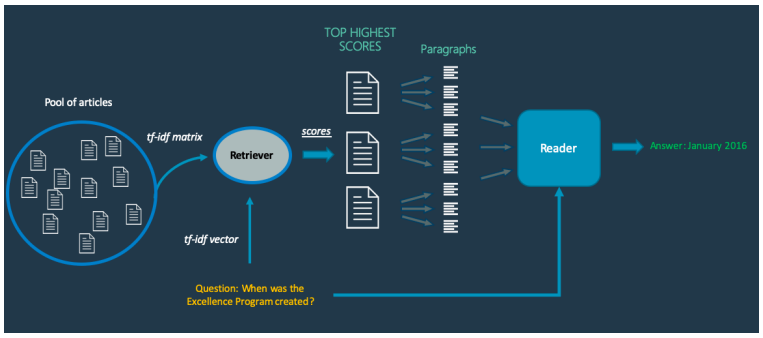

Fig. 1. Design of the CDQA system

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

tf_{ij} = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

The Reader is a PyTorch version BERT model based on HuggingFace's transformer implementation [3]. The model comes pretrained on the popular SQUAD 1.1 dataset [4](Stanford Question Answering Dataset). The model outputs each probable answer it can find to the question in a particular paragraph. This is then sent to another layer that internally scores the answers and outputs the most likely one based upon the scoring.

The CDQA suite consists of:

1) cdqa: python package implementing a QA(Question-Answer) pipeline

2) cdqa-annotator: a tool built to facilitate the annotation of question-answering datasets for model evaluation and fine-tuning

3) cdQA-ui: a user-interface that can be coupled to any website and can be connected to the back-end system.

V. Work done

Our goal was to build a Closed Domain Question Answering system with the 12th grade physics textbook (Part A), and eventually scale the system for other textbooks since the format of data is similar.

A. Phase I

Initially, the dataset had to be built. The model needed textual input and all online sources contained the textbook only in .pdf format. To solve this, we had to built OCR (Optical Character Recognition) code to convert the pdfs into a text file and then preprocess the data to get rid of unicode characters, stray hex values cause by translation of images and other unnecessary characters. This process was done for all eight chapters in the textbook. The processed chapters we then combined to create the dataset for the model to be trained on.

B. Phase II

The CDQA model comes pretrained on the popular SQUAD 1.1 dataset (Stanford Question Answering Dataset) [4]. On querying a few basic questions using the pretrained model on our dataset, we found that the model performed poorly. Thus the model had to be fine tuned to fit our dataset. To achieve this, question-answer pairs had to be generated. Thus the dataset was annotated using the CDQA-annotator for all the chapters and the result was imported as a json file.

C. Phase III

The model was then run on the dataset. First, the dataset was converted to a pandas dataframe using paragraph filters and was passed to the retriever using the fit_retriever function. The reader was then trained with the fit_reader function using the annotated dataset, with 3 epochs and 36 iterations. The trained model was then dumped for testing with new questions.

D. Phase IV

The entire pipeline and the reader were then evaluated separately, and we found to have an improved performance after fine-tuning.

E. Phase V

The entire model was bundled into a python FLASK web app using a REST api. The entire UI was built using the CDQA-ui package.

VI. CODE

The primary code is given below:

```
pip install cdqa #installs the cdqa library
    using pip3

import os
import pandas as pd
from ast import literal_eval

from cdqa.utils.filters import
    filter_paragraphs
from cdqa.pipeline import QAPipeline

from cdqa.utils.download import download_model

download_model(model='bert-squad_1.1',
    dir='./models') #downloads the pretrained
    model

from google.colab import files
uploaded = files.upload() #calls ui to upload
    the dataset as csv

df = pd.read_csv('./dataset.csv',
    converters={'paragraphs':
    literal_eval},encoding="Latin-1",
    names=['title','paragraphs'], header=None)
df = filter_paragraphs(df)
df.head()
#dataset is read and converted into a pandas
    dataframe

cdqa_pipeline =
    QAPipeline(reader='./models/bert_qa.joblib')
cdqa_pipeline.fit_retriever(df=df)
                    #calls the reader and
    retriever part of the cdqa lib that
    parses the dataset

#retriever takes a pool of paragraphs as
    input and ranks and scores them base

#training the reader

cdqa_pipeline.fit_reader('dataset.json')
cdqa_pipeline.dump_reader('./models/bert_qa.joblib')


from cdqa.utils.evaluation import
    evaluate_reader

evaluate_reader(cdqa_pipeline, 'dataset.json')

from cdqa.utils.evaluation import
    evaluate_pipeline

evaluate_pipeline(cdqa_pipeline,
    'dataset.json')
```

VII. RESULT

The trained system tested with an f1-score of 0.7599 .

VIII. WORK TO BE DONE

The retriever currently uses tf-idf features that take into account the frequency of a word in a document to reflect its importance in said document. While this is an efficient solution for a dataset of independent documents, it doesn't scale well for our dataset where the documents are dependent. Further, since increased occurance of a word may not signify the presence of an answer in such a dataset, the retriever sometimes presents the reader with an incorrect list of probable documents. Hence, a new retriever architecture taking order of the documents into account may be created to solve this issue.

CONCLUSION

While our preliminary conclusion is that the project fails to deliver completely, the project offers a variety of opportunities to learn more about how a modern AI works and provides answers to our questions. While the online resources available for use is large, the project definitely involves a considerable

learning curve. With the improvement of retriever architecture, we believe such a project will be feasible for wide-scale release.

REFERENCES

[1] Farias, André Macedo. "How to Create Your Own Question-Answering System Easily with Python." Medium, 21 Oct. 2019, towardsdatascience.com/how-to-create-your-own-question-answering-system-easily-with-python-2ef8abc8eb5.

[2] "Cdqa-Suite Documentation." GitHub, 14 May 2020, github.com/cdqa-suite/cdQA.

[3] "Huggingface/Transformers." GitHub, 15 May 2020, github.com/huggingface/transformers.

[4] Rajpurkar, Pranav, et al. "SQuAD: 100,000+ Questions for Machine Comprehension of Text." ArXiv.Org, 2016, arxiv.org/abs/1606.05250.

SCREENSHOTS