# CLOSED DOMAIN QUESTION ANSWERING

NLP COURSE PROJECT

Mathew George B160008CS

## Aim

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

## Motivation

- There is a gap in technology that aids students to obtain answers to questions directly from a textbook.
- Currently done by scouring online resources and regex search, which is cumbersome.
- To fill this gap, we propose a CDQA system.

### Introduction

- Two types of Question Answering Systems:
  - O <u>Closed Domain Question Answering System:</u> Questions restricted to a particular domain.
  - Open Domain Question Answering System: No restrictions on the domain for questions.
- We aimed to build a CDQA system through this project.
- The specified domain was the Class XII CBSE Physics textbook (Part 1 testing purposes and later extendable to other textbooks)

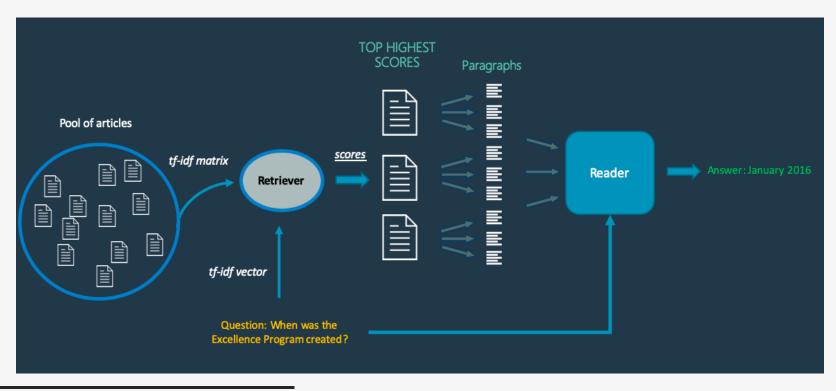
## Software Used

- Used the CDQA suite, a python implementation built upon BERT pretrained on the SQUAD Dataset.
- It has three component parts:
  - O cdqa- python package implementing a QA pipeline
  - cdqa-annotator- a tool built to facilitate the annotation of question-answering datasets for model evaluation and fine-tuning
  - o cdQA-ui: a user-interface that can be coupled to any website and can be connected to the back-end system.

## cdqa

- It is the primary package of the suite.
- It implements a pipeline with the following functions:
  - Fit\_retriever feeds the dataset to the retriever
  - o Fit\_reader feeds the annotated .json file to the reader to train/finetune it
  - Predict predicts the answer to a given question
  - Evaluate\_reader scores the evaluation of the reader
  - Evaluate\_pipeline scores the performance of the reader and the retriever

# Design



### The Retriever

- Main Function: returns a list of documents in the dataset that are most likely to contain the answer.
- Working:
  - O Computes TF-IDF features (Term Frequency Inverse Document Frequency) based on unigrams and bigrams

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

Where  $df_i$  is number of documents containing i, N is total number of documents and  $tf_{i,j}$  is number of occurances of i in j

- O Computes cosine similarity between the question and each document based on those features.
- O After extracting the most probable documents, documents are converted to paragraphs and are sent to the Reader.

### The Reader

- 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).
- Model outputs each probable answer it can find to the question in a particular paragraph.
- These are then scored and the best one is outputted.

## Work Done I

- Phase I
  - Dataset built by converting pdfs to text files using OCR (Optical Character Recognition) and preprocessed to eliminate bad characters.
  - O Done for all eight chapters in the textbook.
  - O The processed chapters combined to create the dataset for the model to be trained on.

#### Phase II

- Basic questions queried using the pretrained model on our dataset. Model performed poorly.
- Concluded that model has to be fine tuned to fit our dataset.
- Dataset was annotated to generate questionanswer pairs to fine tune using the CDQAannotator. Result imported as .json.

## Work Done II

- Phase III
  - o The model was then run on the dataset.
  - Dataset converted to a pandas dataframe using paragraph filters, passed to the Retriever using the fit\_retriever function.
     Output passed to the Reader.
  - O The Reader trained with the fit\_reader function using the annotated dataset, with 3 epochs and 36 iterations.
  - Trained model dumped for testing with new questions.

#### Phase IV

- Pipeline and the Reader evaluated separately, found to have an improved performance after fine-tuning.
- Phase V
  - Entire model bundled into a python FLASK web app using a REST api, built with the CDQA-ui package.
  - Web app tested on localhost:5050

## The Code I

```
!pip install cdqa #installs the cdqa library using pip3
from google.colab import drive
drive.mount('/content/drive')
import os
import pandas as pd
from ast import literal eval
from cdga.utils.filters import filter paragraphs
from cdga.pipeline import QAPipeline
from cdqa.utils.download import download model
download model(model='bert-
squad 1.1', dir='./models')
                                 #downloads the pretrained model that was built on that news articles dataset
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
```

## The Code II

```
cdga pipeline = QAPipeline(reader='./models/bert ga.joblib')
cdqa_pipeline.fit_retriever(df=df)
#calls the reader and retriever part of the cdga 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')
query = "What is current density?"
prediction = cdga pipeline.predict(query, n predictions=5)
print('query: {}'.format(query))
from cdqa.utils.evaluation import evaluate reader
evaluate reader(cdqa pipeline, 'dataset.json')
from cdga.utils.evaluation import evaluate pipeline
evaluate pipeline(cdqa pipeline, 'dataset.json')
```

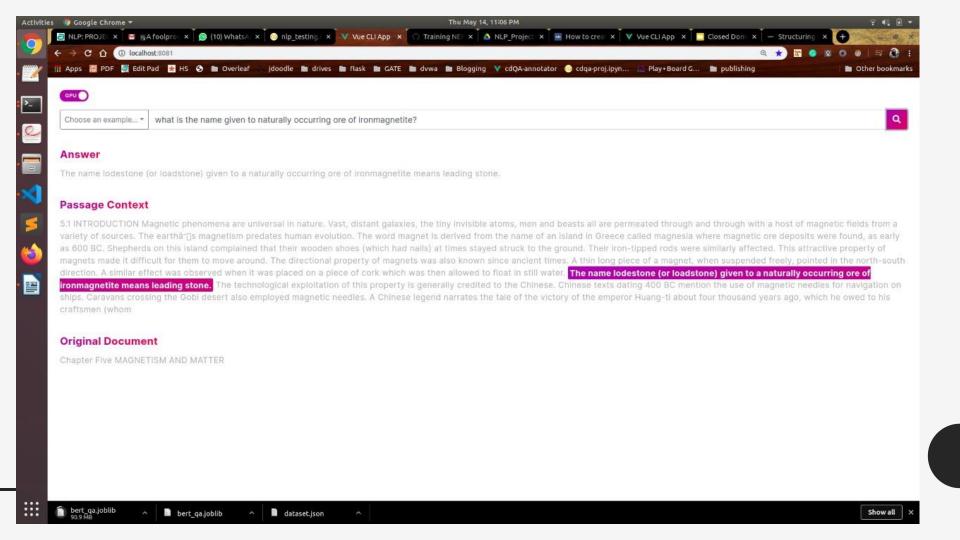
## Result

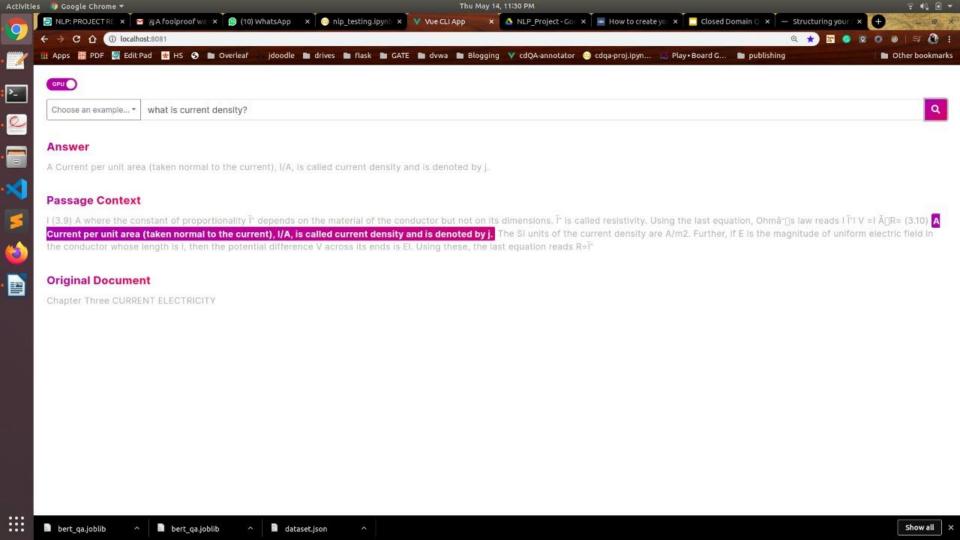
The trained system tested with an f1-score of 0.7599

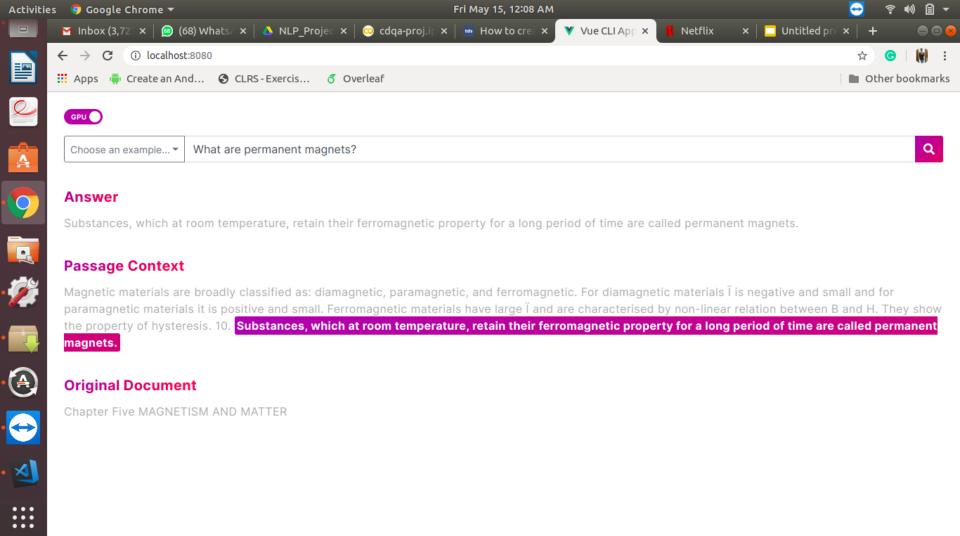
## Further Work

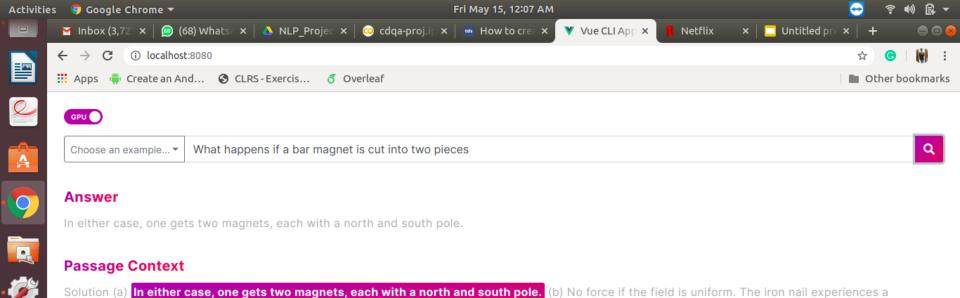
- Retriever currently uses tf-idf features that works based on frequency of a word in a document. This doesn't scale well for our dataset where the documents are dependent.
- Since increased occurrence of a word may not signify presence of an answer, the Retriever sometimes presents the Reader with an incorrect list of probable documents.
- A new Retriever architecture based on order of documents may solve this issue.

## SCREENSHOTS





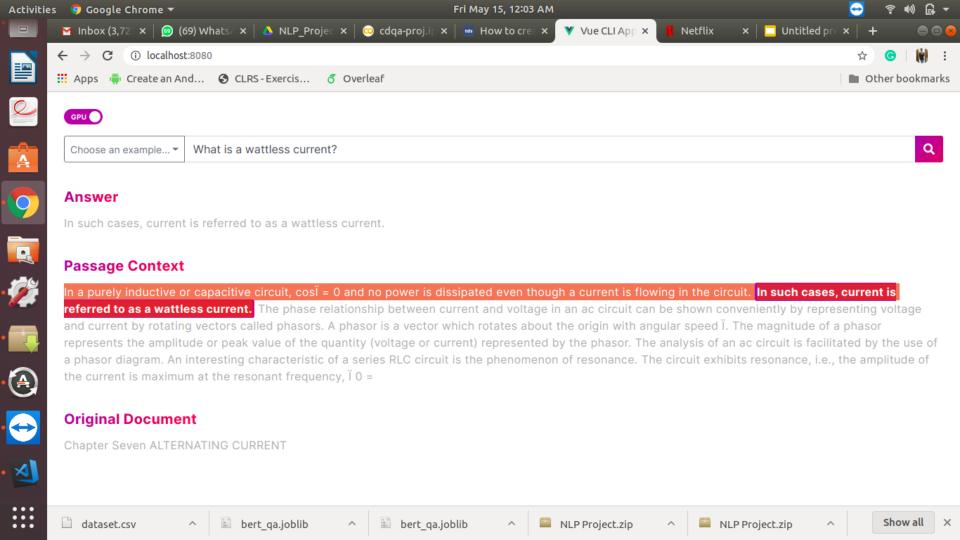


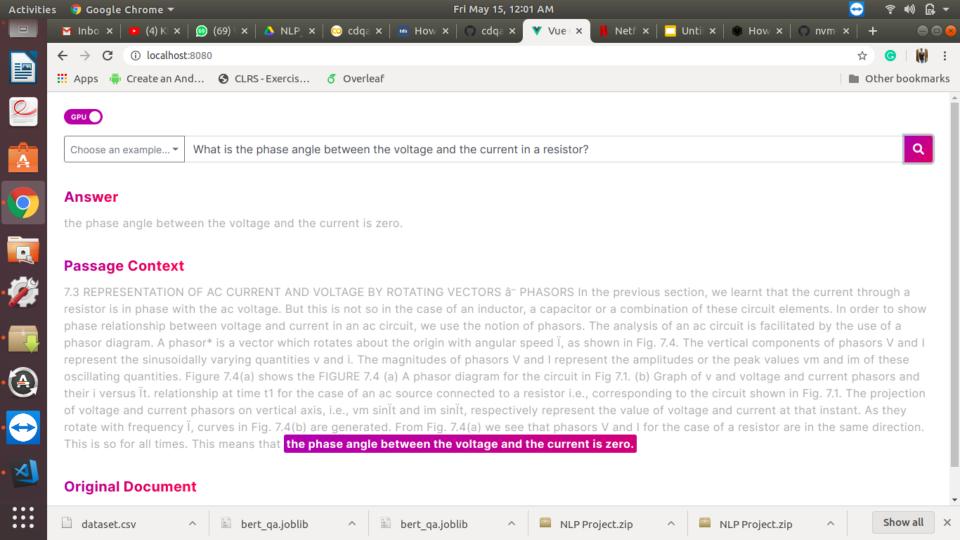


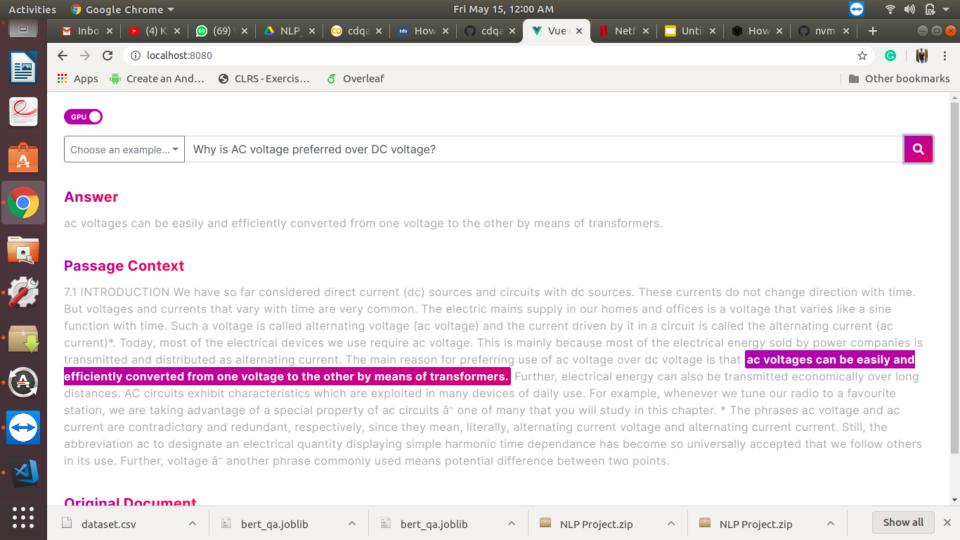
nonuniform field due to the bar magnet. There is induced magnetic moment in the nail, therefore, it experiences both force and torque. The net force is attractive because the induced south pole (say) in the nail is closer to the north pole of magnet than induced north pole. (c) Not necessarily. True only if the source of the field has a net nonzero magnetic moment. This is not so for a toroid or even for a straight infinite conductor. (d) Try to bring different ends of the bars closer. A repulsive force in some situation establishes that both are magnetised. If it is always attractive, then one of them is not magnetised. In a bar magnet the intensity of the magnetic field is the strongest at the two ends (poles) and weakest at the central region. This fact may be used to determine whether A or B is the magnet. In this case, to see which

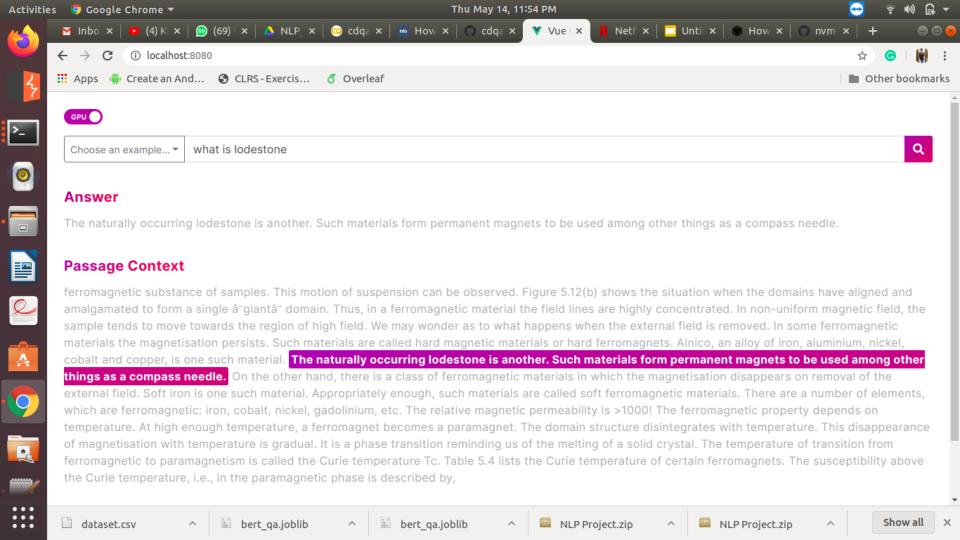
### Original Document

Chapter Five MAGNETISM AND MATTER









## ANNOTATING

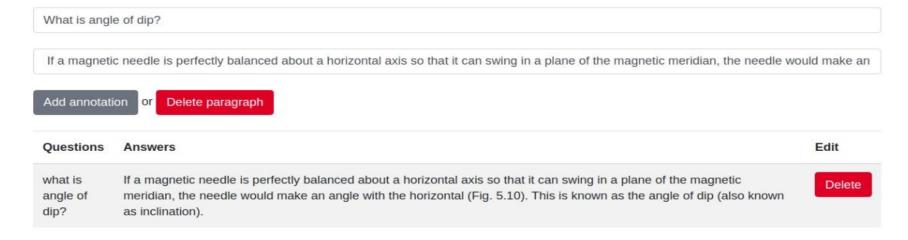
Previous

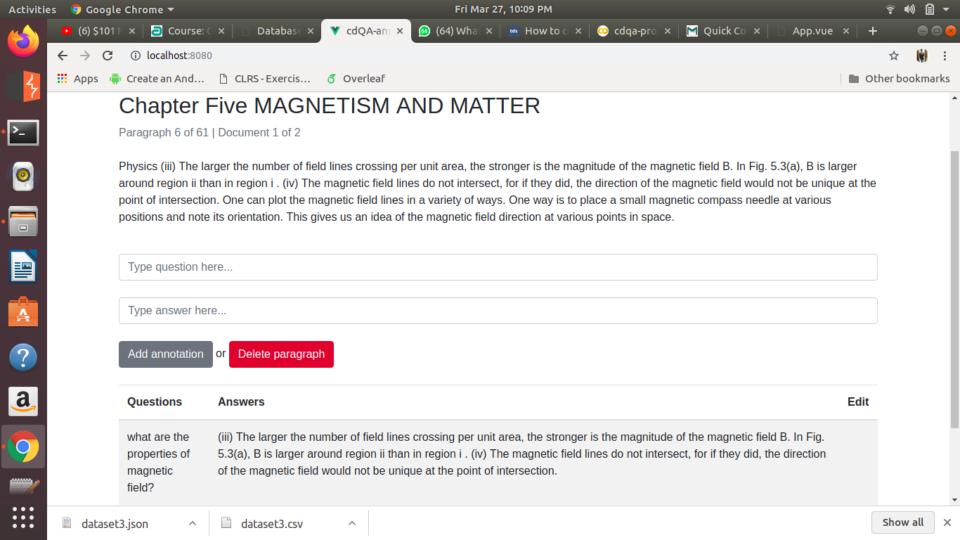
Next

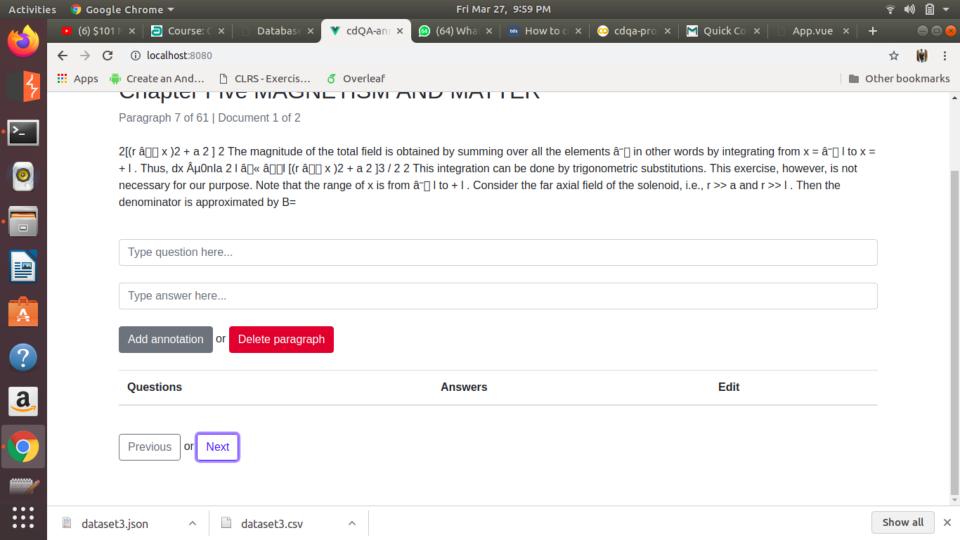
### Chapter Five MAGNETISM AND MATTER

Paragraph 31 of 61 | Document 1 of 2

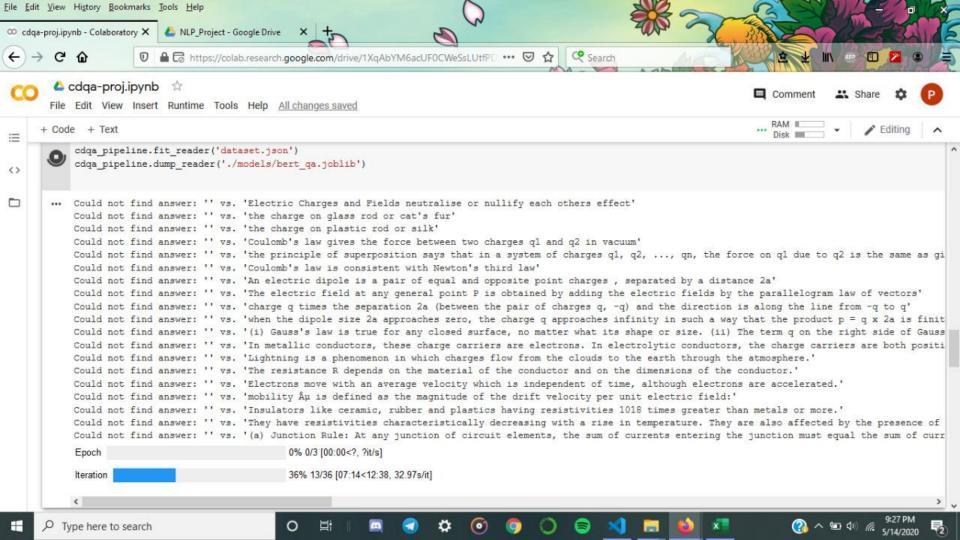
Magnetism and Matter  $0\hat{A}^041\hat{a}^{-2}$  E at Delhi and  $0\hat{A}^058\hat{a}^{-2}$  W at Mumbai. Thus, at both these places a magnetic needle shows the true north quite accurately. There is one more quantity of interest. If a magnetic needle is perfectly balanced about a horizontal axis so that it can swing in a plane of the magnetic meridian, the needle would make an angle with the horizontal (Fig. 5.10). This is known as the angle of dip (also known as inclination). Thus, dip is the angle that the total magnetic field BE of the earth makes with the surface of the earth. Figure 5.11 shows the magnetic meridian plane at a point P on the surface of the earth. The plane is a section through the earth. The total magnetic field at P can be resolved into a horizontal component H E and a vertical component ZE. The angle that BE makes with HE is the angle of dip, I.

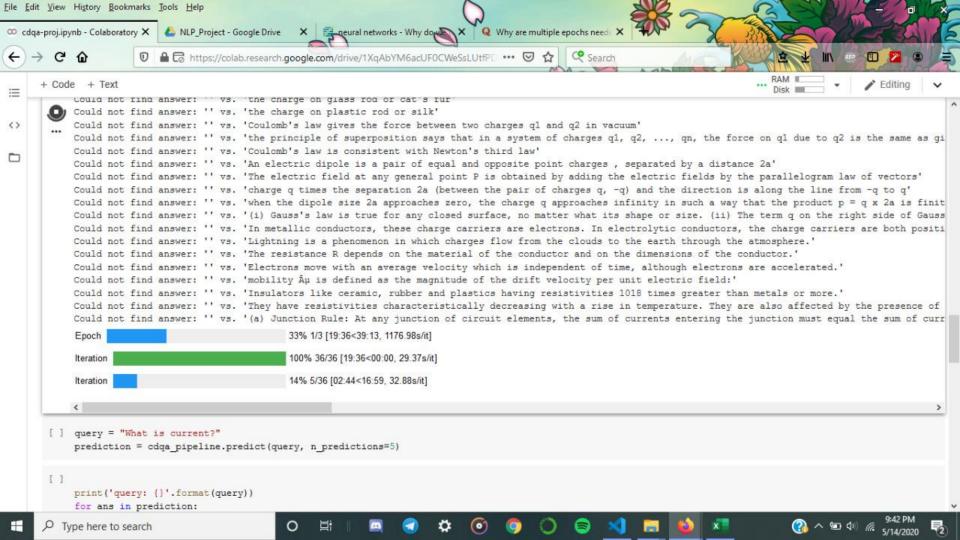






## **TRAINING**





## THE END