VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELAGAVI – 590 018

An Internship Report on

Chat Bot

Submitted in partial fulfillment of the requirements during VII Semester for the degree of **Bachelor of Engineering in Information Science and Engineering** of Visvesvaraya Technological University, Belagavi

Bachelor of Engineering
in
Information Science and Engineering

Submitted by

Abhishek Choudhary 1RN18IS004

UNDER THE GUIDANCE OF

Dr. Prakasha S.
Assistant Professor
Dept. of ISE, RNSIT

ESTD:2001
An Institute with a Difference

Department of Information Science and Engineering
RNS Institute of Technology
Channasandra, Dr. Vishnuvardhan Road, RR Nagar Post
Bengaluru – 560 098
2021 – 2022

RNS Institute of Technology

Channasandra, Dr. Vishnuvardhan Road, RR Nagar Post
Bengaluru – 560 098

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

CERTIFICATE

Certified that the internship work entitled ***Chat Bot*** has been successfully completed by **Abhishek Choudhary (1RN18IS004)**, a bonafide student of **RNS Institute of Technology, Bengaluru** in partial fulfillment of the requirements for the award of degree in **Bachelor of Engineering in Information Science and Engineering** of **Visvesvaraya Technological University, Belagavi** during academic year **2021-2022**. The mini project report has been approved as it satisfies the academic requirements in respect of mini project work for the said degree.

Dr. Prakasha S
Faculty Incharge

Dr. Suresh L
Professor and HoD

DECLARATION

I, **Abhishek Choudhary [USN: 1RN18IS004]** am a student of VII Semester BE, in Information Science and Engineering, RNS Institute of Technology hereby declare that the internship project entitled ***Chat Bot*** has been carried out by me and submitted in partial fulfillment of the requirements for the *VII Semester degree of* ***Bachelor of Engineering in Information Science and Engineering*** *of Visvesvaraya Technological University, Belagavi* during academic year 2021-2022.

Place : Bengaluru

Date :

ABHISHEK CHOUDHARY
(1RN18IS004)

ABSTRACT

Dialogue Generation or Intelligent Conversational Agent development using Artificial Intelligence or Machine Learning technique is an interesting problem in the field of Natural Language Processing. In many research and development projects, they are using Artificial Intelligence, Machine Learning algorithms and Natural Language Processing techniques for developing conversation/dialogue agent. Their research and development is still under progress and under experimentation. Dialogue/conversation agents are are predominately used by businesses, government organizations and non-profit organizations. They are frequently deployed by financial organizations like bank, credit card companies, businesses like online retail stores and start-ups. These virtual agents are adopted by businesses ranging from very small start-ups to large corporations. There are many chatbot development frameworks available in market both code based and interface based. But they lack the flexibility and usefulness in developing real dialogues. Among popular intelligent personal assistants includes Amazon’s Alexa, Microsoft’s Cortana and Google’s Google Assistant. The functioning of these agents are limited, are retrieval based agent and also they are not aimed at holding conversations which emulate real human interaction. Among current chatbots, many are developed using rule based techniques, simple machine learning algorithms or retrieval based techniques which do not generate good results. In this project, I have developed intelligent conversational agent using state of the art techniques proposed in recently published research papers. For developing intelligent chatbot, I have used Google’s Neural machine Translation(NMT) Model which is based on Sequence to Sequence(Seq2Seq) modeling with encoder-decoder architecture. This encoder-decoder is using Recurrent Neural Network with bi-directional LSTM (Long-Short-Term-Memory) cells. For performance optimization, I applied Neural Attention Mechanism and Beam Search during training.

ACKNOWLEDGEMENT

At the very onset I would like to place our gratefulness to all those people who helped me in making the Internship a successful one.

Coming up, this internship to be a success was not easy. Apart from the sheer effort, the enlightenment of the very experienced teachers also plays a paramount role because it is they who guided me in the right direction.

First of all, I would like to thank the **Management of RNS Institute of Technology** for providing such a healthy environment for the successful completion of internship work.

In this regard, I express sincere gratitude to our beloved Principal Dr. M K Venkatesha, **for providing us all the facilities.**

We are extremely grateful to our own and beloved Professor and Head of Department of Information science and Engineering, **Dr. Suresh L**, for having accepted to patronize me in the right direction with all her wisdom.

We place our heartfelt thanks to **DR. SURESH L,** Professor and HOD, Department of Information Science and Engineering for having guided internship and all the staff members of the department of Information Science and Engineering for helping at all times.

I thank **Mr. R Rajkumar,** Associate Professor, Department of Information Science and Engineering, for providing the opportunity to be a part of the Internship program and having guided me to complete the same successfully.

I also thank our internship coordinator **Mr. R Rajkumar,** Associate Professor, Department of Information Science and Engineering. I would thank my friends for having supported me with all their strength and might. Last but not the least; I thank my parents for supporting and encouraging me throughout. I have made an honest effort in this assignment.

Date : 9th January, 2022
Place : Bangalore

ABHISHEK CHOUDHARY
USN-1RN18IS004

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

OVERVIEW

Conversational agent or Chatbot is a program that generates response based on given input to emulate human conversations in text or voice mode. These applications are designed to simulate human-human interactions. Chatbots are predominantly used in business and corporate organizations including government, non-proﬁt and private ones. Their functioning can range from customer service, product suggestion, product inquiry to personal assistant. Many of these chat agents are built using rule based techniques, retrieval techniques or simple machine learning algorithms. In retrieval based techniques, chat agents scan for keywords within the input phrase and retrieves relevant answers based on the query string. They rely on keyword similarity and retrieved text is pulled from internal or external data sources including world wide web or organizational database. Some other advanced chatbots are developed with natural language processing(NLP) techniques and machine learning algorithms. Also, there are many commercial chat engines available, which help build chatbots based on client data input.

Things I have learnt by completing this project:

- How to apply machine learning techniques on Time series Data.
- How to perform statistical analysis of Time series Data.
- How to collect and preprocess given data.
- How to ensemble different machine learning models and analyze model's performance.
- How to optimize Machine Learning models to increase accuracy and reduction in error.

HARDWARE REQUIREMENTS

- **Hardware** : Processor Intel dual core and above
- **Operating System** : Windows 7, Windows 8, Windows 10
- **Internet Connection** : Existing telephone lines, Data card or Any Wireless Network
- **Browser**: Google chrome latest version, IExplorer 10;
- **Performance**: The turn-around time of the project will be medium.

CHAPTER 2

WORKING MODEL OF PROJECT

Sequence to Sequence (Seq2Seq)

Some of the state of the art techniques involve using Deep Neural Network and it's architectural variations. Sequence to Sequence (Seq2Seq) model based on encoder-decoder architecture is such an architecture which is very popular for dialogue generation, language modeling and machine translation. Seq2Seq uses Recurrent Neural Network(RNN) which is a popular Deep Neural Network architecture specially for Natural Language Processing tasks. In Sequence to Sequence (Seq2Seq) model, many to many RNN architecture is used for decoder. In this, encoder-decoder architecture, input sequence is fed as a vector representation of text to encoder. Then, encoder produces some intermediate representation of information or thought vectors. Consequently, the thought vector generated by encoder is fed into decoder as input. Finally, decoder processes the thought vector and converts the sequence one by one word and produces multiple output from the decoder in form of target sequence. Though, vanilla RNN is default in Seq2Seq and works well for many NLP problems yet, due to higher complexity of language modeling problem, vanilla recurrent neural network cells often fails, specially, where long sequence of information needs to be remembered, as this information frequently becomes large for bigger datasets and turns to information bottleneck for the RNN network. Therefore, researchers uses variations of recurrent neural network to handle such problem

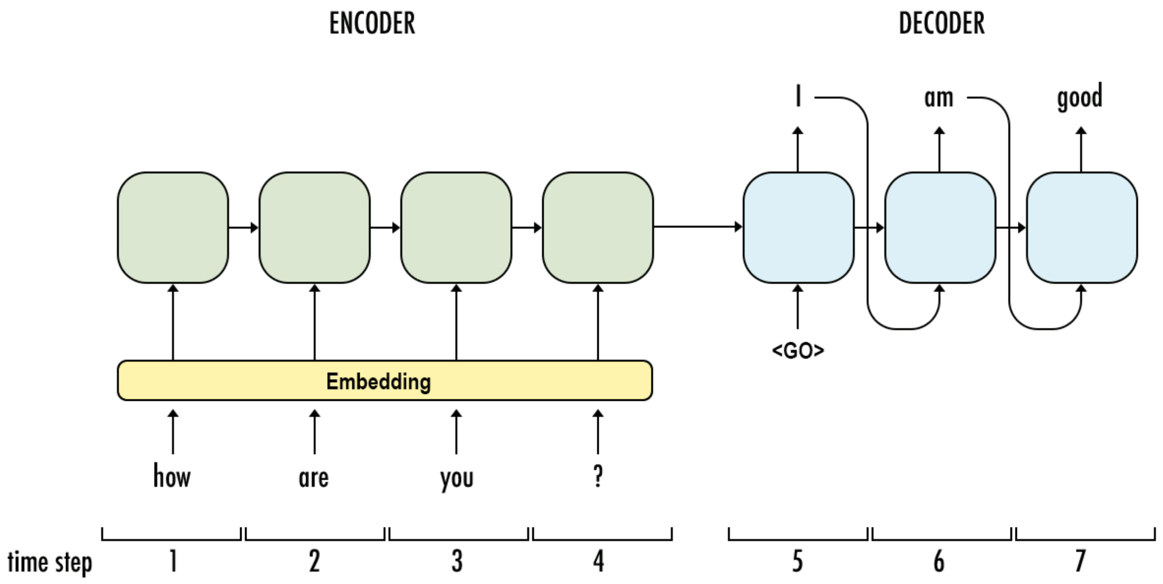


Fig 2.1 Architecture Diagram

CHAPTER 3

ANALYSIS

DATA EXPLORATION

In this project, the dataset ”Cornell Movie Subtitle Corpus” has been primarily used for 6inal model training. Few other dialogue corpus based on movie subtitles have been preprocessed and cleaned for GNMT training including ”Open Movie Subtitle Corpus” and ”Movie Subtitle Corpus”. But due to lack of data quality, they have been eliminated from 6inal training. For future work, more robust and real life conversation based corpus can be incorporated for dialogue generation model building if available. Other possible candidates for dialogue corpus includes, twitter data, reddit dataset and relay chat engine data corpus.
For developing final chatbot, popular movie subtitle corpus ”Cornell movie subtitle corpus” has been used. This corpus contains metadata-rich large collection of conversations extracted from raw movie scripts from popular movies. The following are found in corpus- - 220,579 conversational exchanges between 10,292 pairs of movie characters. - involves 9,035 characters from 617 movies - in total 304,713 utterances Other movie meta-data included genres, release year, IMDB rating, number of IMDB votes, IMDB rating The data corpus can be found in https://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.ht

TECHNICAL INDICATORS

Using Seq2Seq model, we have used softmax normalization on the key and values. We have used adam optimizer for optimization.

CHAPTER 4

ALGORITHMS AND TECHNIQUES

ALGORITHMS

There have been many recent development and experimentation in conversational agent system. Apart from traditional chatbot development techniques that use rule based techniques, or simple machine learning algorithms, many advanced chatbots are using advanced Natural Language Processing (NLP) techniques and Deep Learning Techniques like Deep Neural Network (DNN) and Deep Reinforcement Learning (DRL).

- **Adam Optimizer**

Adam is a replacement optimization algorithm for stochastic gradient descent for training deep learning models. Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems

- **Seq2Seq model**

Neural Attention Mechanism embedded in Seq2Seq module has signiϐicantly improved performance in dialogue generation system and other NLP tasks and thus become industry standard practice. In Neural attention mechanism, each hidden target compares with source hidden state, generates attention vector by calculating score and preserves the attention vector in memory to choose over other candidate. Also, other techniques like, Beam Search can help improve decoding performance further by choosing top candidates. Seq2Seq have also been applied for other NLP tasks including machine translation, text summarization and question-answering and image captioning

- **Stochastic Gradient Descent**

SGD stands for Stochastic Gradient Descent: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate). The regularizer is a penalty added to the loss function that shrinks model parameters towards the zero vector using either the squared euclidean norm L2 or the absolute norm L1 or a combination of both (Elastic Net). I have used following parameters : max_iter=1000, the maximum number of passes over the training data (aka epochs). tol=1e-3, the stopping criterion. If it is not None, the iterations will stop when (loss > previous_loss - tol), loss='squared_epsilon_insensitive', ‘epsilon_insensitive’ ignores errors less than epsilon and is linear past that; this is the loss function used in SVR. ‘squared_epsilon_insensitive’ is the same but becomes squared loss past a tolerance of epsilon. penalty='l1', the penalty (aka regularization term) to be used, alpha=0.1, Constant that multiplies the regularization term.

CHAPTER 5
IMPLEMENTATION

LANGUAGE USED

Python programming language

Machine learning is the kind of programming which gives computer the capability to automatically learn from data without being explicitly programmed. In simple words, ML is a type of artificial intelligence that extract patterns out of raw data by using an algorithm or method. Python is a perfect choice for the field of machine learning and data science. It is a minimalistic and intuitive language with a full-featured libraryline (also called frameworks) which significantly reduces the time required to get the results.

LIBRARIES USED

NumPy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, alongwith a large collection of high- level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open- source software and has many contributors.

Pandas

Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license.

Seaborn

Seaborn is a library for statistical graphics. It builds on top of matplotlib and integrates closely with pandas data structures. Its plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset-oriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them.

Matplotlib

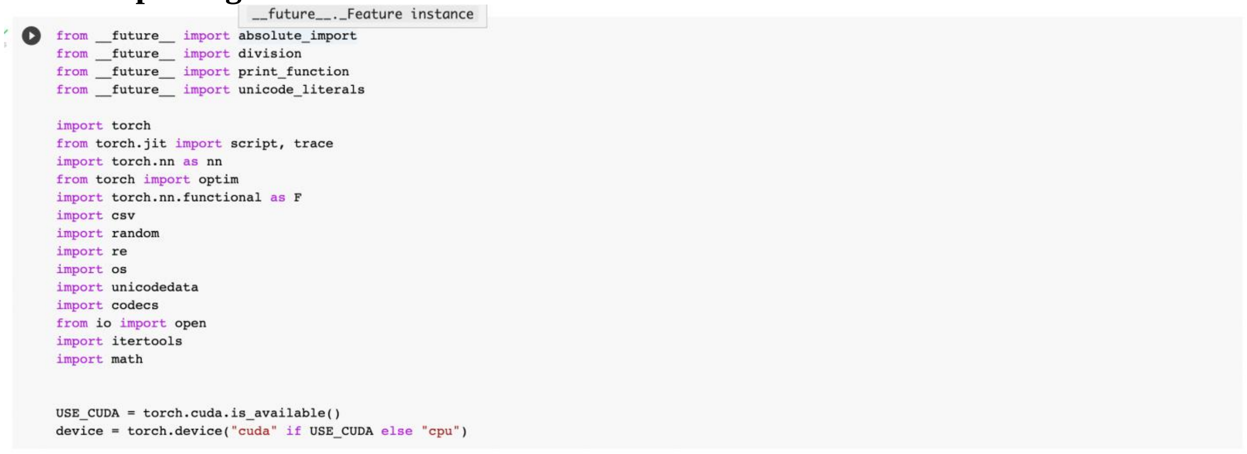
Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits. Pyplot is a Matplotlib module which provides a MATLAB-like interface. Matplotlib is designed to be as usable as MATLAB, with the ability to use Python, and the advantage of being free and open-source.

Scikit Learn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. The vision for the library is a level of robustness and support required for use in production systems. This means a deep focus on concerns such as ease of use, code quality, collaboration, documentation and performance. The library is focused on modeling data. It is not focused on loading, manipulating and summarizing data.

CODE SEGMENT

Importing Libraries

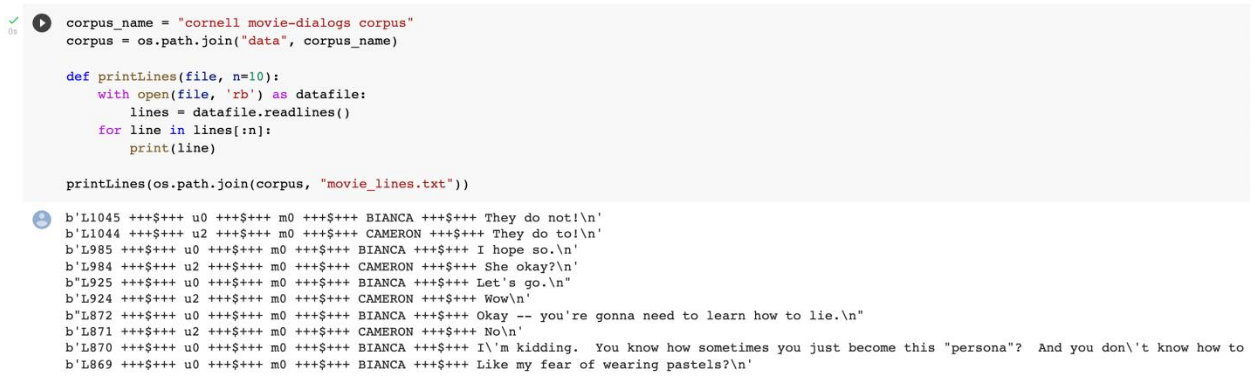


```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals

import torch
from torch.jit import script, trace
import torch.nn as nn
from torch import optim
import torch.nn.functional as F
import csv
import random
import re
import os
import unicodedata
import codecs
from io import open
import itertools
import math


USE_CUDA = torch.cuda.is_available()
device = torch.device("cuda" if USE_CUDA else "cpu")
```

Fig 5.1 Importing Libraries

Load and Preprocess data

The data is to be preprocessed so that it can be inserted as training input.



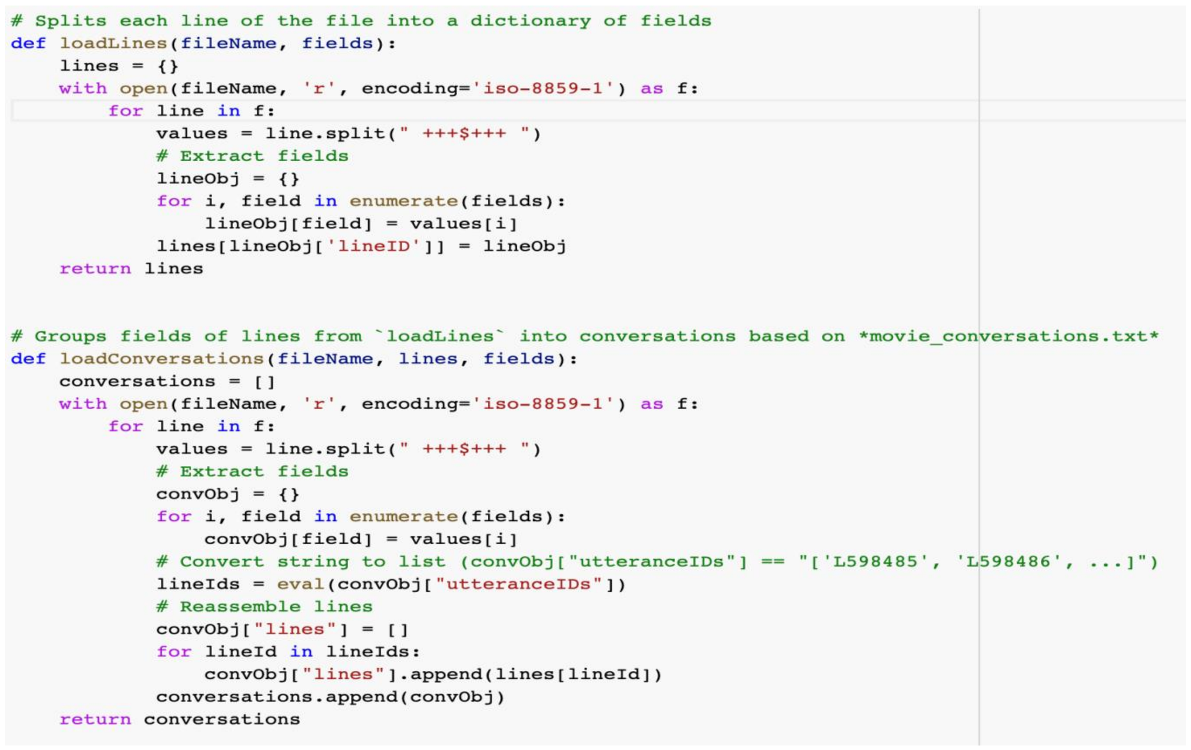
```
corpus_name = "cornell movie-dialogs corpus"
corpus = os.path.join("data", corpus_name)

def printLines(file, n=10):
    with open(file, 'rb') as datafile:
        lines = datafile.readlines()
    for line in lines[:n]:
        print(line)

printLines(os.path.join(corpus, "movie_lines.txt"))
```

```
b'L1045 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ They do not!\n'
b'L1044 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ They do to!\n'
b'L985 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ I hope so.\n'
b'L984 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ She okay?\n'
b"L925 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ Let's go.\n"
b'L924 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ Wow\n'
b"L872 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ Okay -- you're gonna need to learn how to lie.\n"
b'L871 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ No\n'
b'L870 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ I\'m kidding.  You know how sometimes you just become this "persona"?  And you don\'t know how to
b'L869 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ Like my fear of wearing pastels?\n'
```

Fig 5.3 Data preprocessing and loading

Create formatted data file

The following code creates a data file from the formatting which has been done.

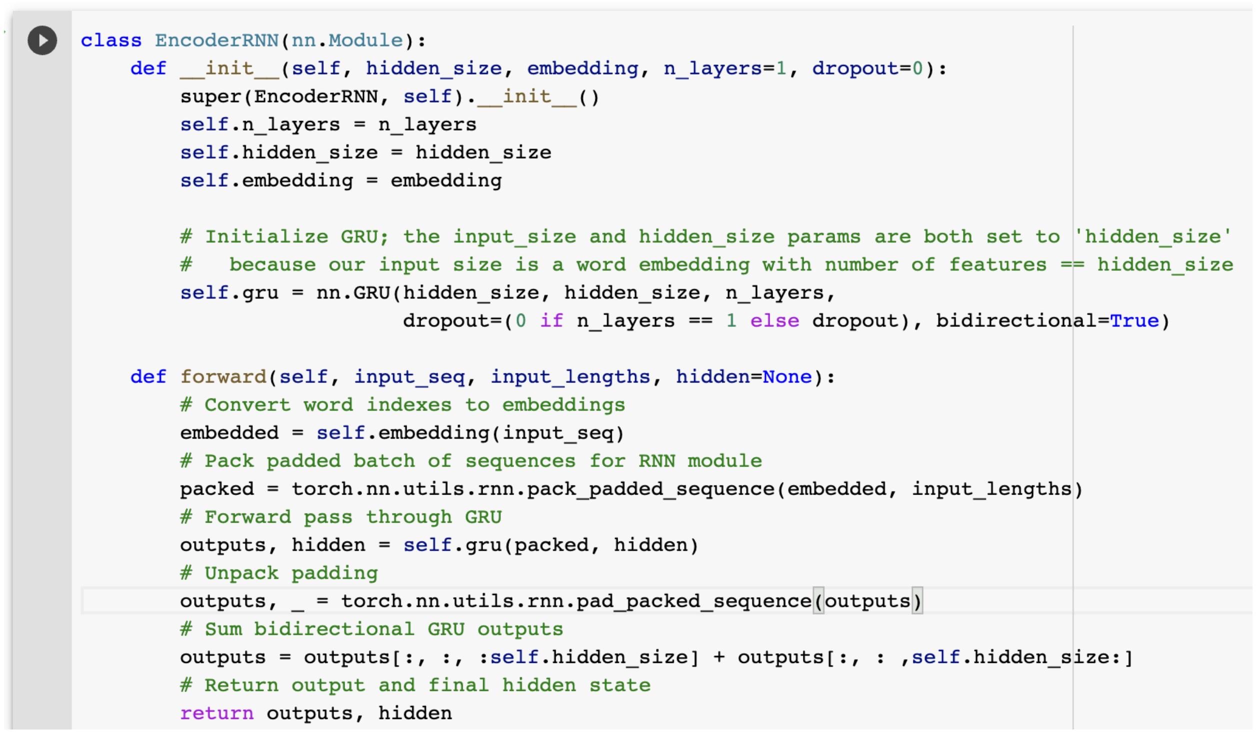


```
# Splits each line of the file into a dictionary of fields
def loadLines(fileName, fields):
    lines = {}
    with open(fileName, 'r', encoding='iso-8859-1') as f:
        for line in f:
            values = line.split(" +++$+++ ")
            # Extract fields
            lineObj = {}
            for i, field in enumerate(fields):
                lineObj[field] = values[i]
            lines[lineObj['lineID']] = lineObj
    return lines


# Groups fields of lines from `loadLines` into conversations based on *movie_conversations.txt*
def loadConversations(fileName, lines, fields):
    conversations = []
    with open(fileName, 'r', encoding='iso-8859-1') as f:
        for line in f:
            values = line.split(" +++$+++ ")
            # Extract fields
            convObj = {}
            for i, field in enumerate(fields):
                convObj[field] = values[i]
            # Convert string to list (convObj["utteranceIDs"] == "['L598485', 'L598486', ...]")
            lineIds = eval(convObj["utteranceIDs"])
            # Reassemble lines
            convObj["lines"] = []
            for lineId in lineIds:
                convObj["lines"].append(lines[lineId])
            conversations.append(convObj)
    return conversations
```

Fig 5.4 Create formatted data file

Defining models

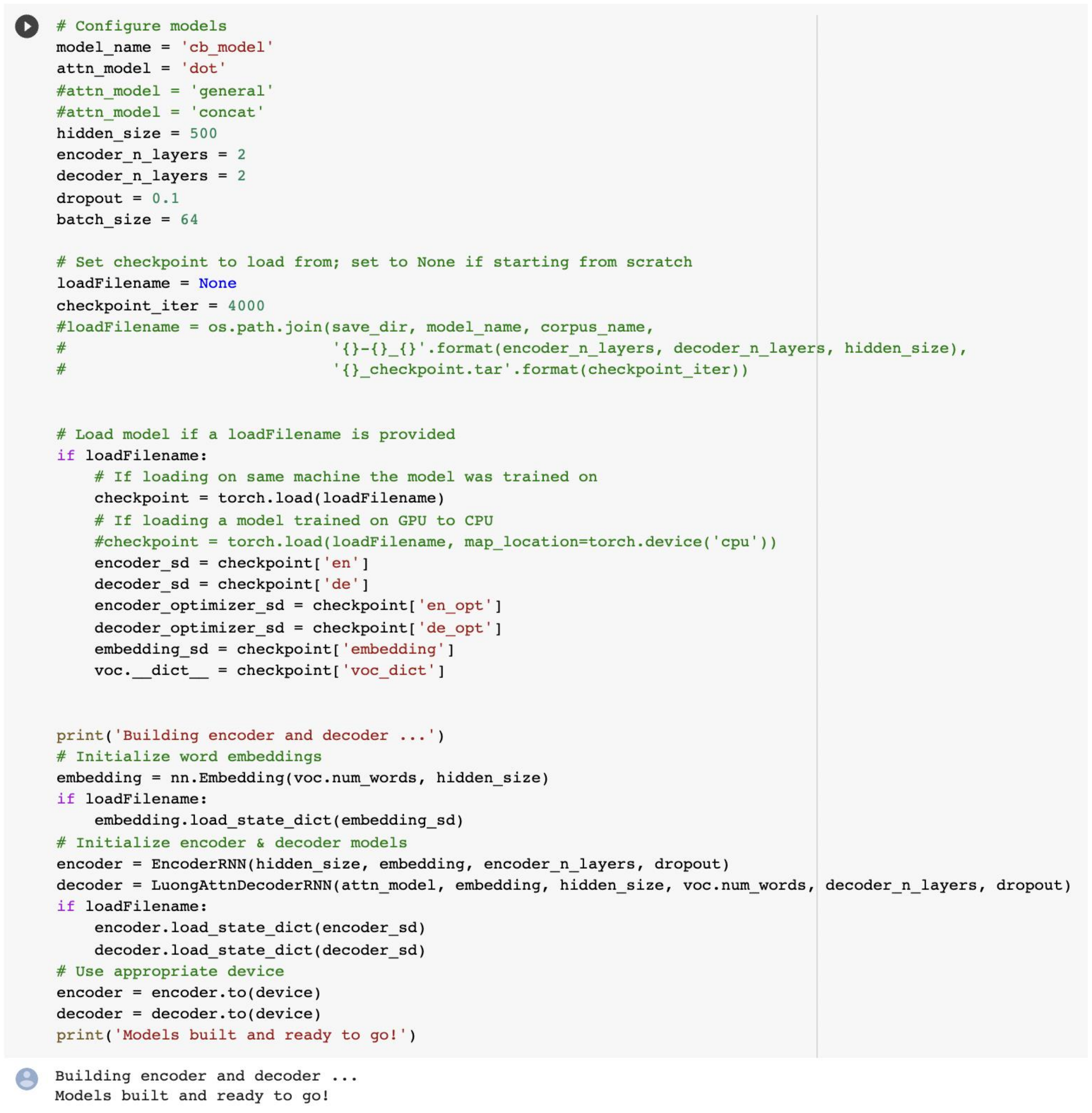
We have defined the models as encoder and decoders for seq2 seq models. The model is made of 2 encoder, 3 decoder, 500 hidden layers and 0.1 dropout ratio.



```
class EncoderRNN(nn.Module):
    def __init__(self, hidden_size, embedding, n_layers=1, dropout=0):
        super(EncoderRNN, self).__init__()
        self.n_layers = n_layers
        self.hidden_size = hidden_size
        self.embedding = embedding

        # Initialize GRU; the input_size and hidden_size params are both set to 'hidden_size'
        #   because our input size is a word embedding with number of features == hidden_size
        self.gru = nn.GRU(hidden_size, hidden_size, n_layers,
                          dropout=(0 if n_layers == 1 else dropout), bidirectional=True)

    def forward(self, input_seq, input_lengths, hidden=None):
        # Convert word indexes to embeddings
        embedded = self.embedding(input_seq)
        # Pack padded batch of sequences for RNN module
        packed = torch.nn.utils.rnn.pack_padded_sequence(embedded, input_lengths)
        # Forward pass through GRU
        outputs, hidden = self.gru(packed, hidden)
        # Unpack padding
        outputs, _ = torch.nn.utils.rnn.pad_packed_sequence(outputs)
        # Sum bidirectional GRU outputs
        outputs = outputs[:, :, :self.hidden_size] + outputs[:, : ,self.hidden_size:]
        # Return output and final hidden state
        return outputs, hidden
```

Training the model

The seq2seq model is trained with 40,000 epochs



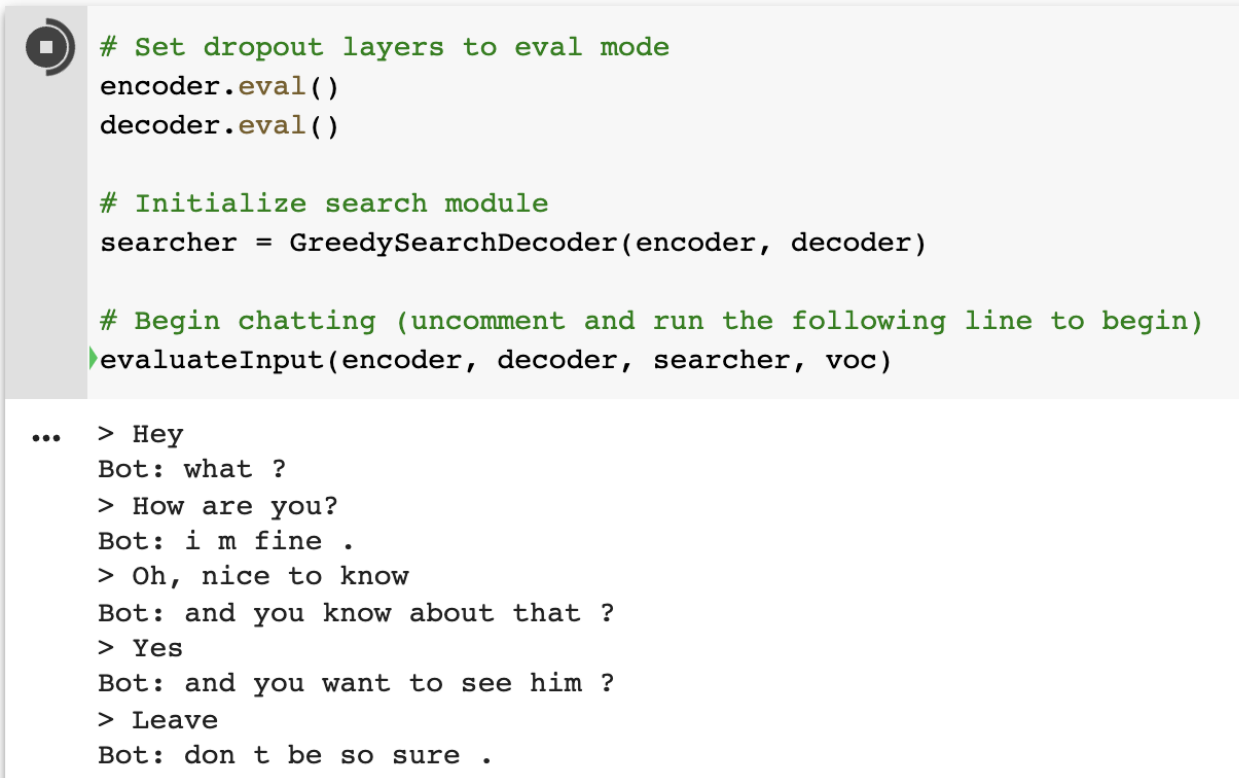
```
# Configure models
model_name = 'cb_model'
attn_model = 'dot'
#attn_model = 'general'
#attn_model = 'concat'
hidden_size = 500
encoder_n_layers = 2
decoder_n_layers = 2
dropout = 0.1
batch_size = 64

# Set checkpoint to load from; set to None if starting from scratch
loadFilename = None
checkpoint_iter = 4000
#loadFilename = os.path.join(save_dir, model_name, corpus_name,
#                            '{}-{}_{}'.format(encoder_n_layers, decoder_n_layers, hidden_size),
#                            '{}_checkpoint.tar'.format(checkpoint_iter))


# Load model if a loadFilename is provided
if loadFilename:
    # If loading on same machine the model was trained on
    checkpoint = torch.load(loadFilename)
    # If loading a model trained on GPU to CPU
    #checkpoint = torch.load(loadFilename, map_location=torch.device('cpu'))
    encoder_sd = checkpoint['en']
    decoder_sd = checkpoint['de']
    encoder_optimizer_sd = checkpoint['en_opt']
    decoder_optimizer_sd = checkpoint['de_opt']
    embedding_sd = checkpoint['embedding']
    voc.__dict__ = checkpoint['voc_dict']


print('Building encoder and decoder ...')
# Initialize word embeddings
embedding = nn.Embedding(voc.num_words, hidden_size)
if loadFilename:
    embedding.load_state_dict(embedding_sd)
# Initialize encoder & decoder models
encoder = EncoderRNN(hidden_size, embedding, encoder_n_layers, dropout)
decoder = LuongAttnDecoderRNN(attn_model, embedding, hidden_size, voc.num_words, decoder_n_layers, dropout)
if loadFilename:
    encoder.load_state_dict(encoder_sd)
    decoder.load_state_dict(decoder_sd)
# Use appropriate device
encoder = encoder.to(device)
decoder = decoder.to(device)
print('Models built and ready to go!')
```

```
Building encoder and decoder ...
Models built and ready to go!
```

Fig 5.7 Training the model

Testing the model

The model is tested using the GreedySearchDecoder algorithm.



```
# Set dropout layers to eval mode
encoder.eval()
decoder.eval()

# Initialize search module
searcher = GreedySearchDecoder(encoder, decoder)

# Begin chatting (uncomment and run the following line to begin)
evaluateInput(encoder, decoder, searcher, voc)
```

```
> Hey
Bot: what ?
> How are you?
Bot: i m fine .
> Oh, nice to know
Bot: and you know about that ?
> Yes
Bot: and you want to see him ?
> Leave
Bot: don t be so sure .
```

Fig 5.8 Testing the model

SGD

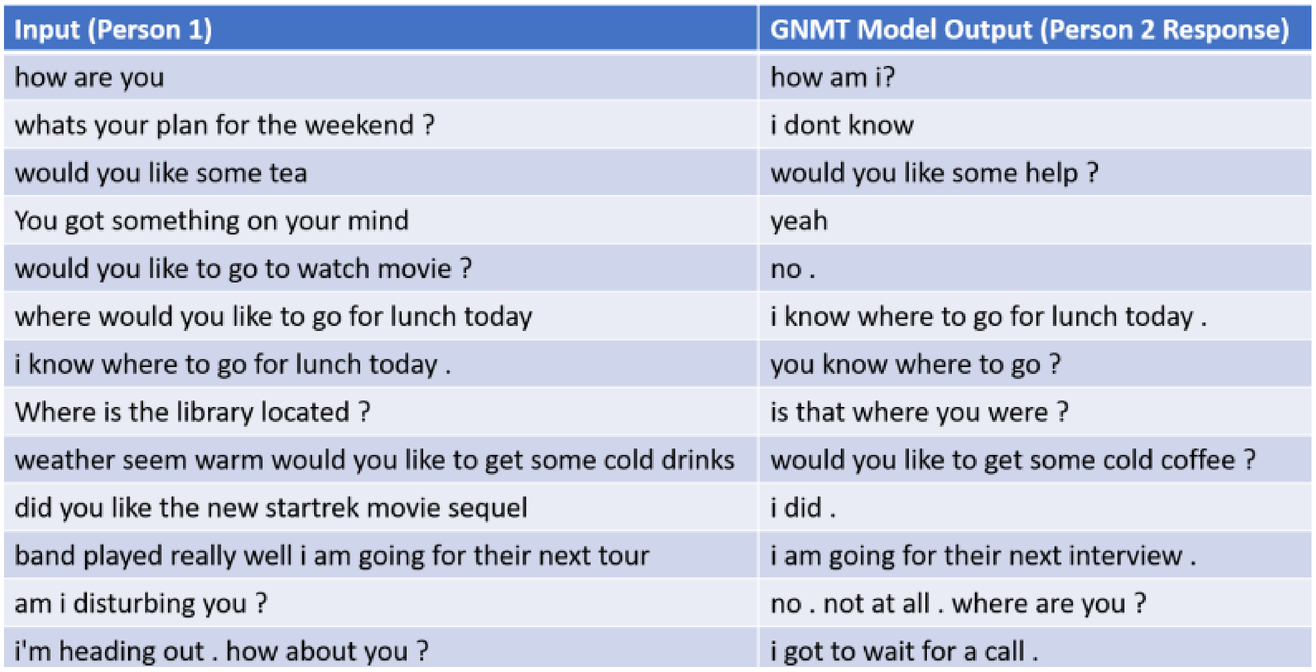
SGD stands for Stochastic Gradient Descent: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate). The regularizer is a penalty added to the loss function that shrinks model parameters towards the zero vector using either the squared euclidean norm L2 or the absolute norm L1 or a combination of both (Elastic Net). I have used following parameters : max_iter=1000, tol=1e-3, loss='squared_epsilon_insensitive', penalty='l1' and alpha=0.1

CHAPTER 7

FUTURE ENHANCEMENT AND CONCLUSIONS

Results

Following are some response derived after training on full dataset with 34MB of training text with 225000+225000 utterance. The initial test result produced moderately coherent sentences. The following responses were generated after inference from trained model. In inference, trained model produced 10-30 candidate response for each input. Hence, the following examples are most suited from 10-30 candidate responses during inference.



Input (Person 1)	GNMT Model Output (Person 2 Response)
how are you	how am i?
whats your plan for the weekend ?	i dont know
would you like some tea	would you like some help ?
You got something on your mind	yeah
would you like to go to watch movie ?	no .
where would you like to go for lunch today	i know where to go for lunch today .
i know where to go for lunch today .	you know where to go ?
Where is the library located ?	is that where you were ?
weather seem warm would you like to get some cold drinks	would you like to get some cold coffee ?
did you like the new startrek movie sequel	i did .
band played really well i am going for their next tour	i am going for their next interview .
am i disturbing you ?	no . not at all . where are you ?
i'm heading out . how about you ?	i got to wait for a call .

Fig 7.1 Results

Challenges

After training, chatbot produced results with moderate relevancy. But many of the output were repetitive and generic. Also, due to lack of real-life quality data the chatbot performed somehow below optimum for imitating human interaction. Also, much utterance was discarded due to longer length or discrepancy. And, number of training utterance was much less than required and test and development dataset was quite larger in comparison which might have caused the model to under-perform. Also, as data was limited, longer period of training may not have suited the dialogue generation problem.

Reflection Future Scope of the Project

The chatbot developed using Google’s Neural Machine Translation Model(GNMT) can be further improved with more robust, high quality real-life conversational datasets which could better emulate human interaction. Also, hyper-parameters of the GNMT model can be further fine-tuned and optimized for performance enhancement. Based on available opportunity to further advance the project, Deep Reinforcement Learning(RL) can be applied that could significantly improve performance as shown empirically in Dufarsky’s paper. Reinforcement learning algorithm can be applied after the initial training using Google’s Neural Machine Translation.

REFERENCES

- Udacity
- Kaggle
- Github
- Coursera
- Youtube
- Medium
- Wikipedia, the free encyclopedia