# CS 523 Project 2: Chat Bot Using RNN in Tensor Flow

## Introduction:

For the second project we decided to implement the option of developing a Chat bot using RNN (Recurrent Neural Network) in Tensor Flow. We searched for multiple repositories which we could modify to develop the chat bot which outputs data in our own style.

Following two repositories were reused and modified to implement our chat bot.

1. <https://github.com/b0noI/dialog_converter/tree/simaple_input_generator> (Repository 1)
2. <https://github.com/b0noI/tensorflow/tree/r0.11/tensorflow/models/rnn/translate> (Repository 2)
3. <https://github.com/suriyadeepan/easy_seq2seq/tree/master/ui> (Repository 3)

We first got these repositories to run on their own without making any changes.

## Explanation of Functionality of Original Repositories:

### Repository 1:

This is the converter of the dialog data, that is used in order to prepare the data. The repository converts the dialog data in the Cornell Movie Dialog Corpus in a way that can be used for training the input of the recurrent Neural Network. The repository will produce 4 files train.a and train.b which can be used for training and test.a and test.b which can be used for testing.

### Repository 2:

The second repository is provided by Tensor Flow which implements a sequence to sequence RNN which uses the attention mechanism to for translation from English to French.

### Repository 3:

The third repository is owned by Suriyaeepan Ram and we have referenced the Web Interface part from this repository. The Web Interface is built using flask.

## Implementing the repository for English to French translations:

Original repository was run as a deliverable for part 1 of the project submission to get any chat bot to run. This was due in class after week 1 of the project. For English to French Translations only the repository 2 was used. No code changes were made to the repository. The repository 2 was downloaded and the translate.py file was run in a python 3.5 Conda environment. The model was trained for 100000 iterations. Once the model was trained the English sentences were translated to French at a good accuracy.

(Initially we trained the model by providing English data as input to the encoder. The vector provided by the encoder is passed as an input to the decoder. At each step t while decoding the data the output at t-1 was passed as input to the next step of the decoder).

## Implementing the chat bot for English to English movie dialog conversations:

The code for the project is found in the folder ‘English to English Conversations’ in our repository. This folder has a user interface which can be used to interact with the model to get outputs in the style of a movie dialog corpus. If the user wants to train the model on their own no UI is present.

In Order to implement the chat bot for generating the output in the style of movie dialog corpus both the repository 1 and repository 2 were used.

Repository 1 was used to prepare the training and testing data and repository 2 was modified to train on the generated training data produced by repository 1.

### Process:

#### Creation of The Data from Repository 1

Repository 1 was downloaded and the Converter.py was run in a Conda environment having Tensor Flow package in python 3.5.

The output produced, generate 4 files from the Cornell Movie Dialog Corpus which are as follows:

1. Train.a (Input for Training)
2. Train.b (Expected Outputs During Training)
3. Test.a (Input for Testing)
4. Test.b (Expected Outputs During Testing)

The above 4 files were renamed as

1. Train.en (Input for Training)
2. Train.en1 (Expected Outputs During Training)
3. Test.en (Input for Testing)
4. Test.en1 (Expected Outputs During Testing)

#### Training The Chat Bot Repository 2:

Once these files were renamed, they were copied into a ‘tmp’ folder which was explicitly created in the root of the directory where repository 2 was downloaded. The ‘tmp’ directory path was configured by modifying the below two flags in the translate.py file of the downloaded repository 2. We modified the path from “/tmp” to ‘’temp”. Below are the two changed flags.

tf.app.flags.DEFINE\_string("data\_dir", "tmp", "Data directory")

tf.app.flags.DEFINE\_string("train\_dir", "tmp", "Training directory.")

The original Repository 2 is set up such that it trains the RNN on English to French datasets. We modified the code to allow the repository to train on our datasets. Below is the modified code in repository 2.

#### Modified Code:

In File data\_utils.py:

#train\_path = get\_wmt\_enfr\_train\_set(data\_dir) # Commented this Line

#dev\_path = get\_wmt\_enfr\_dev\_set(data\_dir) # Commented this Line

train\_path = os.path.join(data\_dir, "train") # Added this Line

dev\_path = os.path.join(data\_dir, "test") # Added this Line

In function create\_vocabulary, add the following:

for line in f:

line = tf.compat.as\_bytes(line) # added by Us

counter += 1

In function sentence\_to\_token\_ids, make the following change

# return [vocabulary.get(re.sub(\_DIGIT\_RE, b"0", w), UNK\_ID) for w in words] # Commented This Line

return [vocabulary.get(w.decode('utf-8'), UNK\_ID) for w in words] # added by Us

In Function data\_to\_token\_ids

for line in data\_file:

line = tf.compat.as\_bytes(line) # added by Us

counter += 1

In File Translate.py:

from tensorflow.models.rnn.translate import data\_utils

from tensorflow.models.rnn.translate import seq2seq\_model

import data\_utils

import seq2seq\_model

#### Further Changes Made for UI:

In order to create a Web Interface, we used flask which can be imported by using the command:

“conda install -c anaconda flask=0.12” or “pip install Flask”. Following are the folders used for web interface (present in English to English Conversation)

* Static – this contains a folder for javascript file –js which has the following files

1. Index.js – contains the main logic of interacting with interface. The button clicks for Chat is present in this file which fetches the message and then via AJAX request, it sends the message to decode\_line method present in translate.py to get the output from seq2seq\_model. The reply is appended in a div.
2. Other jquery files present in this director is to support the jquery operations.

* Template – this contains the main html file: index.html which contains a text box where user can enter messages and click on Chat to get output. The html to be rendered must be present in template folder as flask expects the html file in template folder. We embed index.js file in this html and functions with respect to sending message(chat), receiving response are handled in index.js

**Webui.py** – this file contains the code to interact with the UI. The constructor is initialized with index.js present in static folder which is embedded in index.html.

app = Flask(\_\_name\_\_,static\_url\_path="/static")

This file receives the AJAX request from index.js and replies back with the output for message.

@app.route('/message', methods=['POST'])

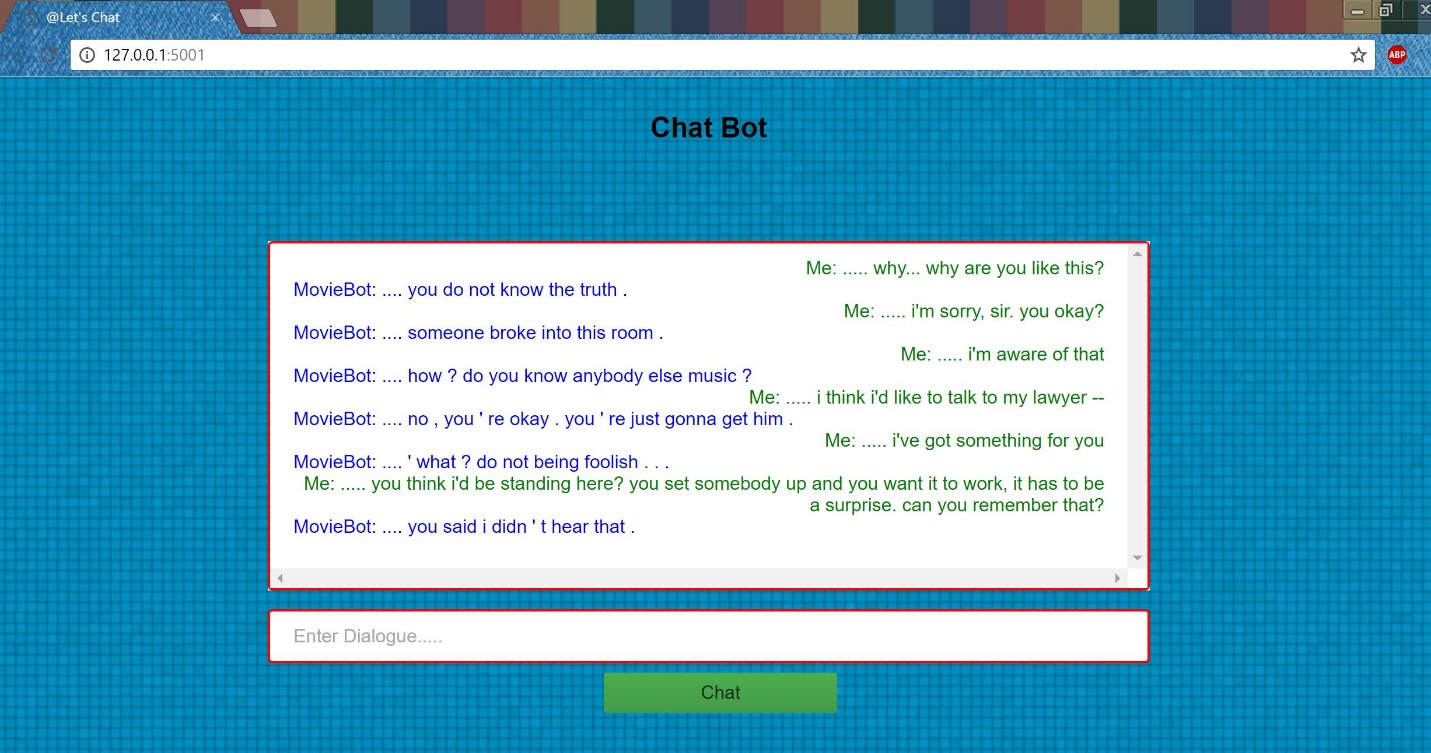
def reply():

return jsonify( { 'text': translate.decode\_line(sess, model, en\_vocab, rev\_en1\_vocab, request.form['msg'] ) } )

Following are operations performed in this file:

* Initiate a tensorflow session and load the model, vocab for English-English conversations using “**init\_session**” function in translate.py
* It has an AJAX POST method to accept request from user. This method calls “**decode\_line**” method in translate.py with the loaded model, vocab to fetch the output from seq2seqmodel.

#### Screenshots:



## Implementing the chat bot for English to German translations:

For the purpose of the demo we in class we decided to show the sequence to sequence RNN model trained on English to German.

The code for the project is found in the folder ‘English to German Conversations’ in our repository. This folder has a user interface which can be used to interact with the model to get outputs in the style of a movie dialog corpus. If the user wants to train the model on their own no UI is present.

In Order to implement the chat bot for generating the translation from English to German, we used only Repository 2 and the data set was downloaded from <http://www.statmt.org/wmt15/translation-task.html> - English to German corpus – Europarl v7 which contains two files: europarl-v7.de-en.de (German translations) and europarl-v7.de-en.en (English translations). We simply renamed the file from europarl-v7.de-en.en – train.en and europarl-v7.de-en.de – train.en1. The test files were created from these training files by reading the starting 5000 lines from each of the file and were saved in respective files: test.en and test.en1.

#### Training The Chat Bot Repository 2:

Once these files were renamed, they were copied into a ‘tmp’ folder which was explicitly created in the root of the directory where repository 2 was downloaded. The ‘tmp’ directory path was configured by modifying the below two flags in the translate.py file of the downloaded repository 2. We modified the path from “/tmp” to ‘’temp”. Below are the two changed flags.

tf.app.flags.DEFINE\_string("data\_dir", "tmp", "Data directory")

tf.app.flags.DEFINE\_string("train\_dir", "tmp", "Training directory.")

Further both the training files and testing files were copied in a new folder eng-german present under temp.

The original Repository 2 is set up such that it trains the RNN on English to French datasets. We modified the code to allow the repository to train on our datasets. Below is the modified code in repository 2.

#### Modified Code:

In File data\_utils.py:

#train\_path = get\_wmt\_enfr\_train\_set(data\_dir) # Commented this Line

#dev\_path = get\_wmt\_enfr\_dev\_set(data\_dir) # Commented this Line

train\_path = os.path.join(data\_dir, "train") # Added this Line

dev\_path = os.path.join(data\_dir, "test") # Added this Line

In function create\_vocabulary, add the following:

for line in f:

line = tf.compat.as\_bytes(line) # added by Us

counter += 1

In function sentence\_to\_token\_ids, make the following change

# return [vocabulary.get(re.sub(\_DIGIT\_RE, b"0", w), UNK\_ID) for w in words] # Commented This Line

return [vocabulary.get(w.decode('utf-8'), UNK\_ID) for w in words] # added by Us

In Function data\_to\_token\_ids

for line in data\_file:

line = tf.compat.as\_bytes(line) # added by Us

counter += 1

In File Translate.py:

from tensorflow.models.rnn.translate import data\_utils

from tensorflow.models.rnn.translate import seq2seq\_model

import data\_utils

import seq2seq\_model

#### Further Changes Made for UI:

In order to create a Web Interface, we used flask which can be imported by using the command:

“conda install -c anaconda flask=0.12” or “pip install Flask”. Following are the folders used for web interface (present in English to English Conversation)

* Static – this contains a folder for javascript file –js which has the following files

1. Index.js – contains the main logic of interacting with interface. The button clicks for **Translate** is present in this file which fetches the message and then via AJAX request, it sends the message to “**tdecode\_line**” method present in translate.py to get the output from seq2seq\_model. The reply is appended in a div.
2. Other jquery files present in this director is to support the jquery operations.

* Template – this contains the main html file: index.html which contains a text box where user can enter messages and click on Translate to get the translation. The html to be rendered must be present in template folder as flask expects the html file in template folder. We embed index.js file in this html and functions with respect to sending message(translate), receiving response are handled in index.js

**Webui.py** – this file contains the code to interact with the UI. The constructor is initialized with index.js present in static folder which is embedded in index.html.

app = Flask(\_\_name\_\_,static\_url\_path="/static")

This file receives the AJAX request from index.js and replies back with the output for message.

@app.route('/message', methods=['POST'])

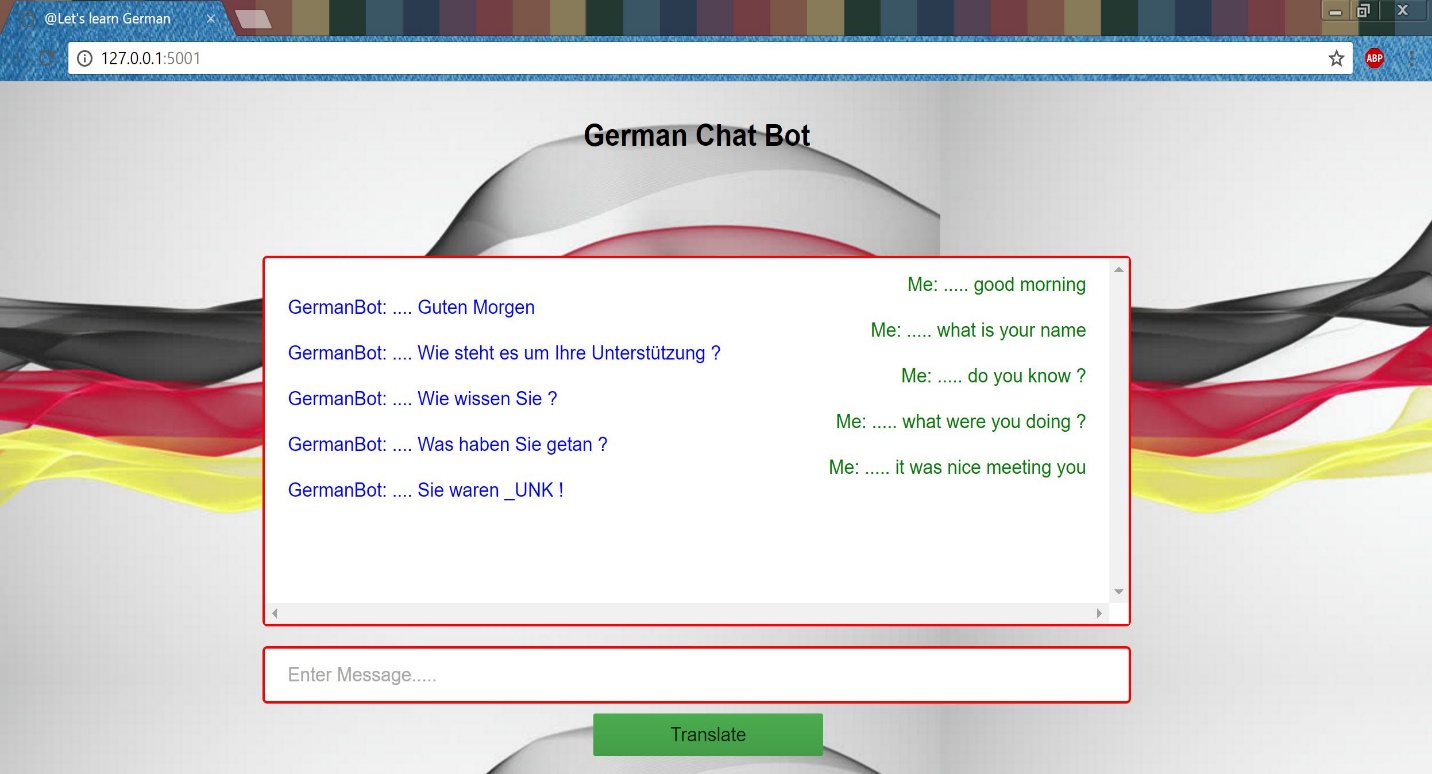
def reply():

return jsonify( { 'text': translate.decode\_line(sess, model, en\_vocab, rev\_en1\_vocab, request.form['msg'] ) } )

Following are operations performed in this file:

* Initiate a tensorflow session and load the model, vocab for English-German translations using “**tinit\_session**” function in translate.py
* It has an AJAX POST method to accept request from user. This method calls “**tdecode\_line**” method in translate.py with the loaded model, vocab to fetch the output from seq2seqmodel.

#### Screenshots:



## Conclusion:

The chat bots were implemented using recurrent neural network with LSTM (Long short term memory model) with sequence to sequence models and transfer learning. The English to German Translations chat bot was trained for 216800 iterations on a windows machine with GPU while the English to English conversations chat bot was trained on the same machine for 142200 iterations. The outputs of the chat bots varied and were accurate for some inputs and not all. We presume if we limit each line in the training data set to only 5 words and train for the same number of iterations we may get more accuracy in the outputs for both the chat bots for the datasets we have used.