Introduction

The aim of this project is to implement a Multilingual Chatbot i.e a chatbot which is capable of conversing with people of different linguistic backgrounds. As a prototype this chatbot system can interact in both Hindi and English and answer queries on Goods and Services Tax(GST).

Working Concept

This project is divided into two sections :- a Chatbot and a Translator. The Chatbot is capable of understanding and replying to queries in English only. .The Translator converts the query from the user's native language to english, which is fed into the Chatbot who determines the response to the user's query, this response is passed to the translation api which this time reverse translates the english response to the user's native language. This model ensures scalabilty as in the future this project would incorporate various other languages, hence the developers just have to create a new translation model for that language rather than implementing the whole system(Chatbot + Translator) from scratch.

Chatbot

Chatbot Interface

Transformer Model

The aim was to build the chatbot using the generative based model. For this purpose we had used the transformer model with tensorflow 2.0.
Transformer, proposed in the paper Attention is All You Need, is a neural network architecture solely based on self-attention mechanism and is very parallelizable.A Transformer model handles variable-sized input using stacks of self-attention layers instead of RNNs or CNNs.

This general architecture has a number of advantages:

- It makes no assumptions about the temporal/spatial relationships across the data. This is ideal for processing a set of objects.
- Layer outputs can be calculated in parallel, instead of a series like an RNN.
- Distant items can affect each other's output without passing through many recurrent steps, or convolution layers.
- It can learn long-range dependencies.
- The disadvantage of this architecture:
- For a time-series, the output for a time-step is calculated from the entire history instead of only the inputs and current hidden-state. This may be less efficient.
- If the input does have a temporal/spatial relationship, like text, some positional encoding must be added or the model will effectively see a bag of words.
  Note that Transformer is an autoregressive model, it makes predictions one part at a time and uses its output so far to decide what to do next. During training this example uses teacher-forcing. Teacher forcing is passing the true output to the next time step regardless of what the model predicts at the current time step.

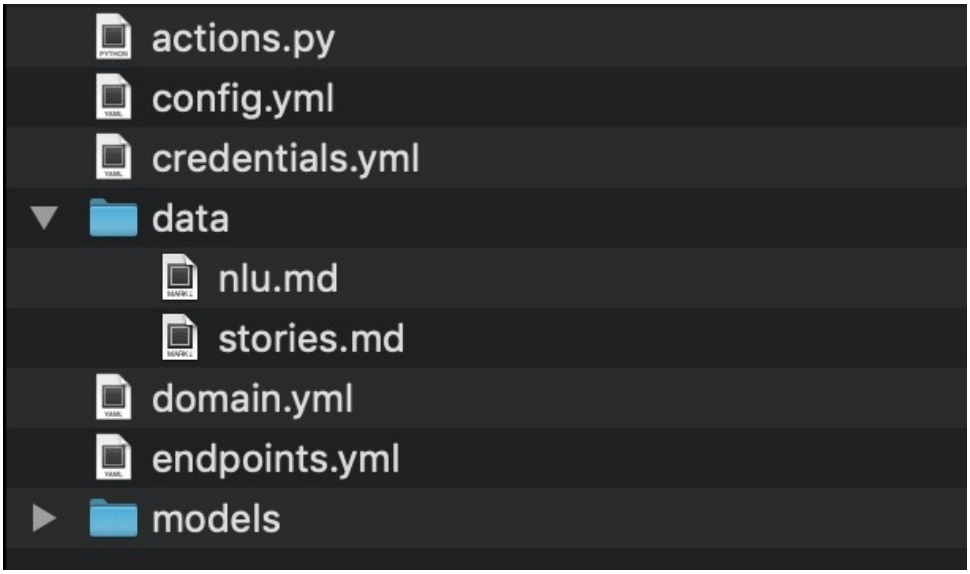
The model requires huge amount of data to achieve satisfactory results. Another disadvantage is that incorporating Named Entity Recognition (NER) with this model is very difficult which requires expertise in this domain.

Due to these disadvantages we had to move from the generative model to a retrival based model and use Rasa.

Introduction To Rasa

Rasa is an open source machine learning framework for automated text and voice-based conversations. Understand messages, hold conversations, and connect to messaging channels and APIs.

Your NLU and dialogue data should be stored in a ./data directory in your project while all other files (domain, custom actions script, credentials, endpoints, nlu configuration file) should be kept in the main directory of your project folder. The image below represents the project structure the Rasa expects to see:

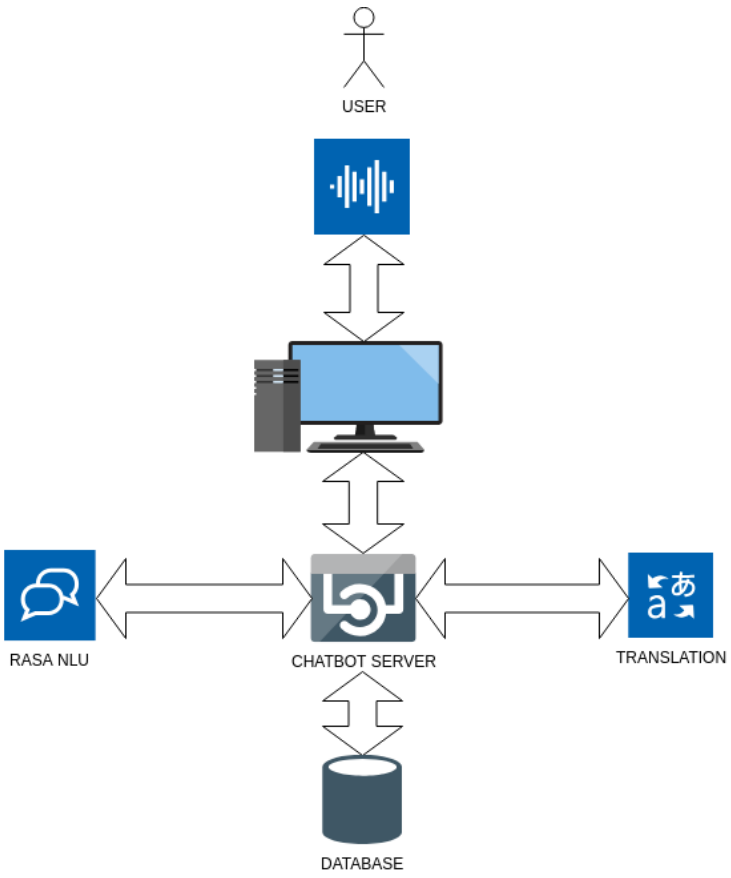


__init__.py	A file that helps python find your actions
actions.py	code for your custom actions
config.yml	configuration of your NLU and Core models
credentials.yml	details for connecting to other services
data/nlu.md	your NLU training data
data/stories.md	your stories or dialogue flow
domain.yml	your assistant's domain
endpoints.yml	details for connecting to channels like fb messenger
models/<timestamp>.tar.gz	your initial model

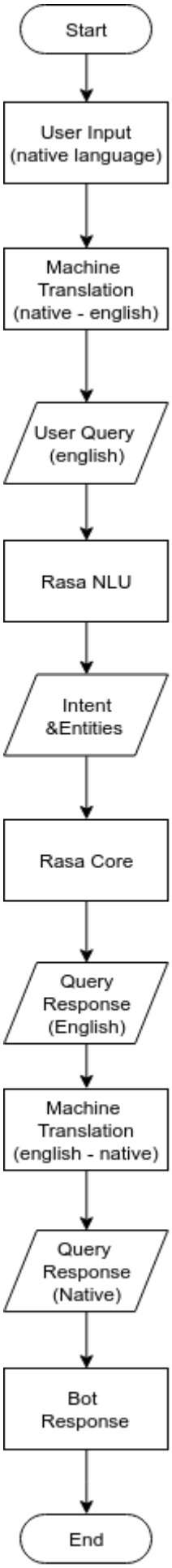
You can read more on https://rasa.com/docs/rasa/ and Rasa Blog.

Rasa NLU + Database

NLU stands for Natual Language Understanding. Rasa NLU determines the intent and the entities present in the user's query. Learn More About Rasa NLU : https://rasa.com/docs/rasa/nlu/using-nlu-only/. The response is determined based on the information extracted by NLU model from the query. But the response is predefined hence we have to create a database or a dictionary which holds the response and define the logic which determines the response. This method becomes complex as the number of intents and entites increase hence it is not scalable.



Architecture


Start
User Input
(native language)
Machine
Translation
(native - english)
User Query
(english)
Rasa NLU
Intent
&Entities
Rasa Core
Query
Response
(English)
Machine
Translation
(english - native)
Query
Response
(Native)
Bot
Response
End

Work Flow

Inbuilt pipelines(pipeline is a collection of processes executed one after the other) are available in Rasa NLU, for more information refer : https://rasa.com/docs/rasa/nlu/choosing-a-pipeline/. As Rasa is an open source framework the developers can create custom pipeline tailored to their needs, custom pipelines can be defined in the config.yml file. Rasa NLU also offers different entity extractors from which the developer can choose from, refer https://rasa.com/docs/rasa/nlu/entity-extraction/. The pipeline which has been used in this project is :

```
language: en
pipeline:
- name: WhitespaceTokenizer
- name: RegexFeaturizer
- name: CRFEntityExtractor
- name: EntitySynonymMapper
- name: CountVectorsFeaturizer
- name: EmbeddingIntentClassifier
```

Refer https://rasa.com/docs/rasa/nlu/training-data-format/ for more information on how to create the training data file. Rasa NLU model is hosted as a server. Communication between the master and the Rasa NLU server occurs through API calls. When hosted locally Rasa NLU server has a default IP address: http://localhost:5000. To retrieve the intent and the entities from a query, the query message is sent as a get request to the NLU server with the “q” parameter (Code available in GBOT/rasapi.py). The rasa nlu server responds with a json object like:-

The intents field of this response contains a list of intents along with the confidence values for the intents which have been predicted by the Rasa NLU. Choosing the intent depends on the developer, he can choose the intent with the greatest confidence value. But this logic is not applicable in every situation, if the user queries something that is out of scope for the chatbot Ex: Who is the president of Algeria? then that query should not be mapped to any of the defined intents, in this case the developer should set a minimum threshold for confidence below which the Chatbot gives a fallback response Ex: Sorry, I didn’t get that.
The developer has to define the logic for giving response to a query (refer : GBOT/engine.py).

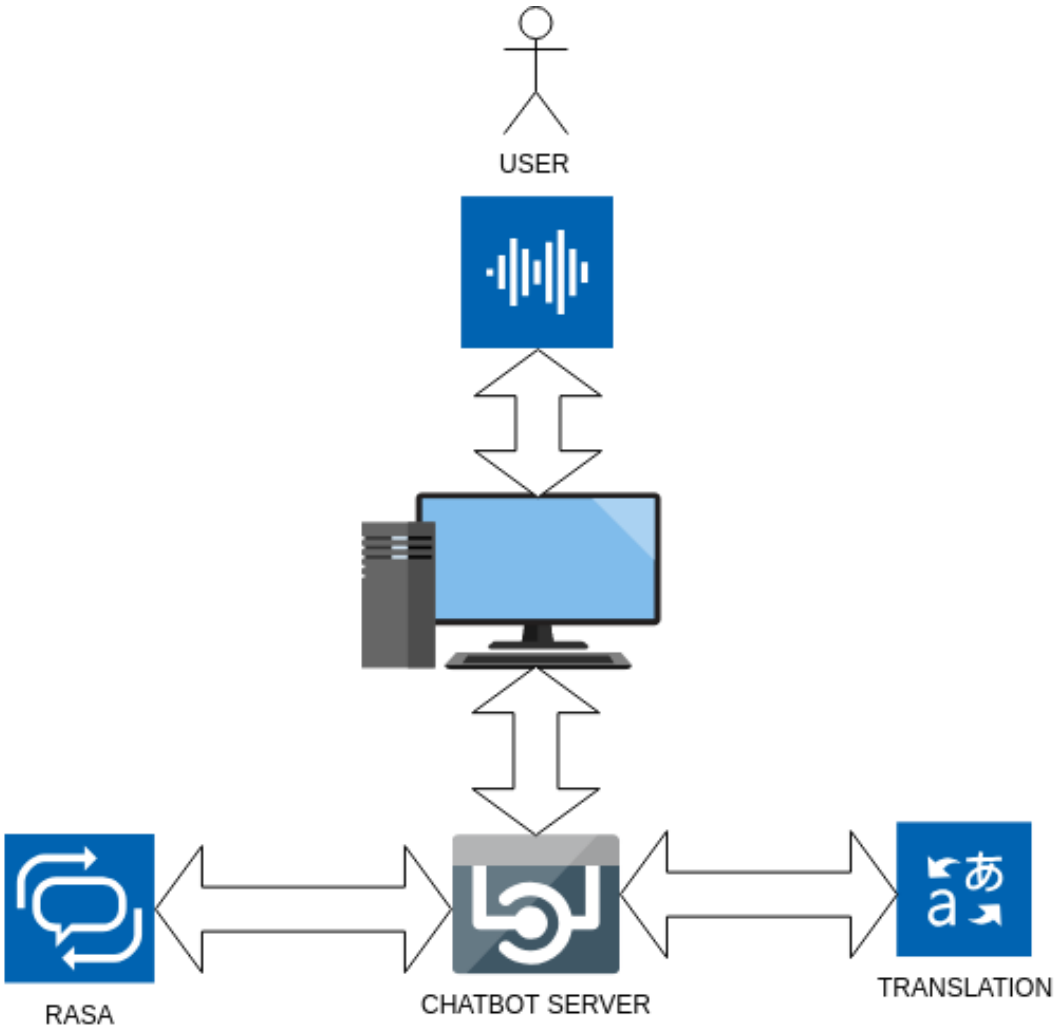
RASA STACK

Due to the scalability issues in Rasa NLU + Database model ,Rasa Stack was used. Rasa stack consists of two parts : Rasa NLU + Rasa Core. Rasa Core utilises Deep Learning models which determines the response (refer: https://rasa.com/docs/rasa/core/about/). The Rasa Core is a collection of policies which is used to choose the output. There are various policies to choose from refer :- https://rasa.com/docs/rasa/core/policies/ for more information. The developer has to specify the stories in the data/stories.md file, any lookup table that has been used should also be present in the data directory(https://rasa.com/docs/rasa/nlu/training-data-format/).

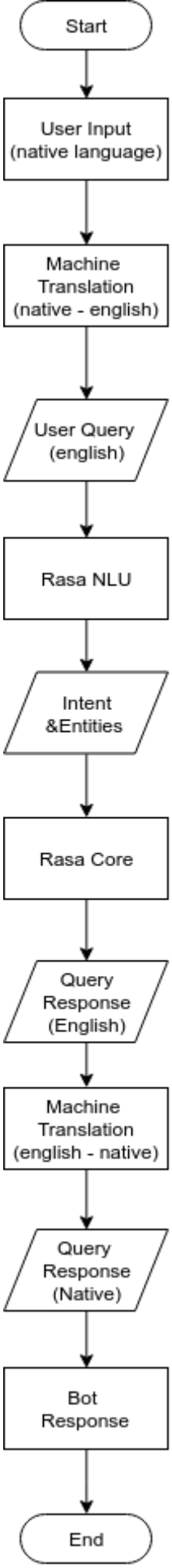
The policies used in this project are :-

```
policies:
- name: "FallbackPolicy"
  nlu_threshold: 0.7
  core_threshold: 0.3
  fallback_action_name: 'utter_fallback'
```

```
- name: EmbeddingPolicy
  epochs:  500
  attn_shift_range: 5
  featurizer:
  - name: FullDialogueTrackerFeaturizer
    state_featurizer:
    - name: LabelTokenizerSingleStateFeaturizer
- name: AugmentedMemoizationPolicy
- name: MappingPolicy
```



Architecture


Start
User Input
(native language)
Machine
Translation
(native - english)
User Query
(english)
Rasa NLU
Intent
&Entities
Rasa Core
Query
Response
(English)
Machine
Translation
(english - native)
Query
Response
(Native)
Bot
Response
End

Work Flow

The entites which are discovered by Rasa NLU can be used to choose the output by the core by using slots for more information refer :- https://rasa.com/docs/rasa/core/slots/. The developer is not required to create all the combinations for the story instead he can create separate stories for each intent and while training he can provide the –augmentation flag which create new stories by randomly mixing present stories. This makes this project more scalable as even though the number of intents increate the developer just has to create single new intents for each one of them rather than creating lengthy stories, thereby saving time and effort.

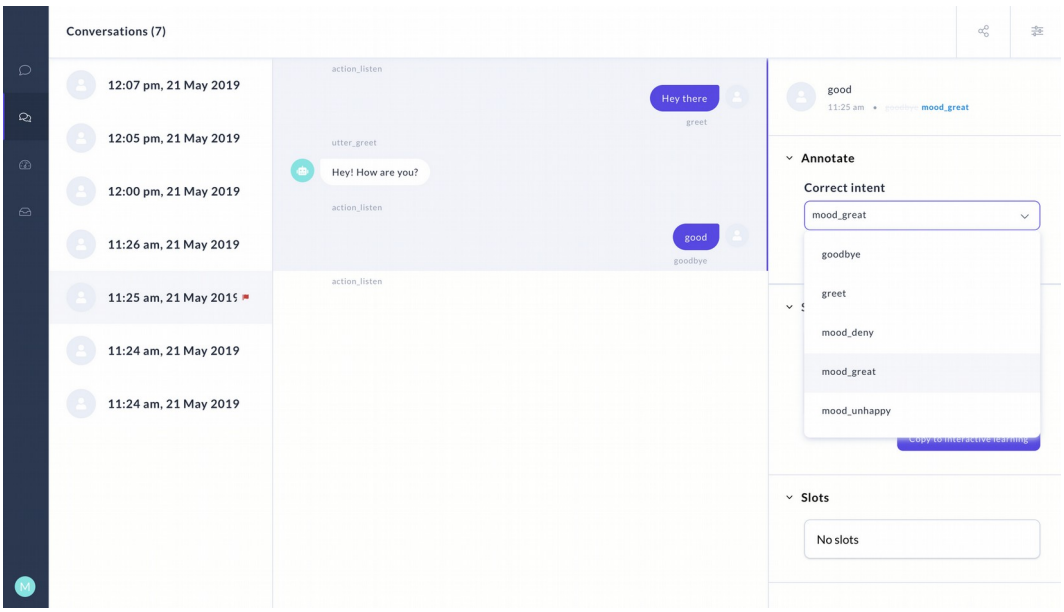
Rasa server can be started locally by using the command :-
rasa run -m models --enable-api --cors “*” --debug (Debug Mode)
For more information Refer : https://rasa.com/docs/rasa/user-guide/running-the-server/

This server uses webhooks to communicate with the master server. The webhook is hosted as http://localhost:5005/webhooks/rest/webhook and a post request with “sender” field which contains the sender’s name(any string) and “message” field which contains the query is sent to the server. The server response is converted to json (Parsing response example: GBOT_Server/rasapi.py).

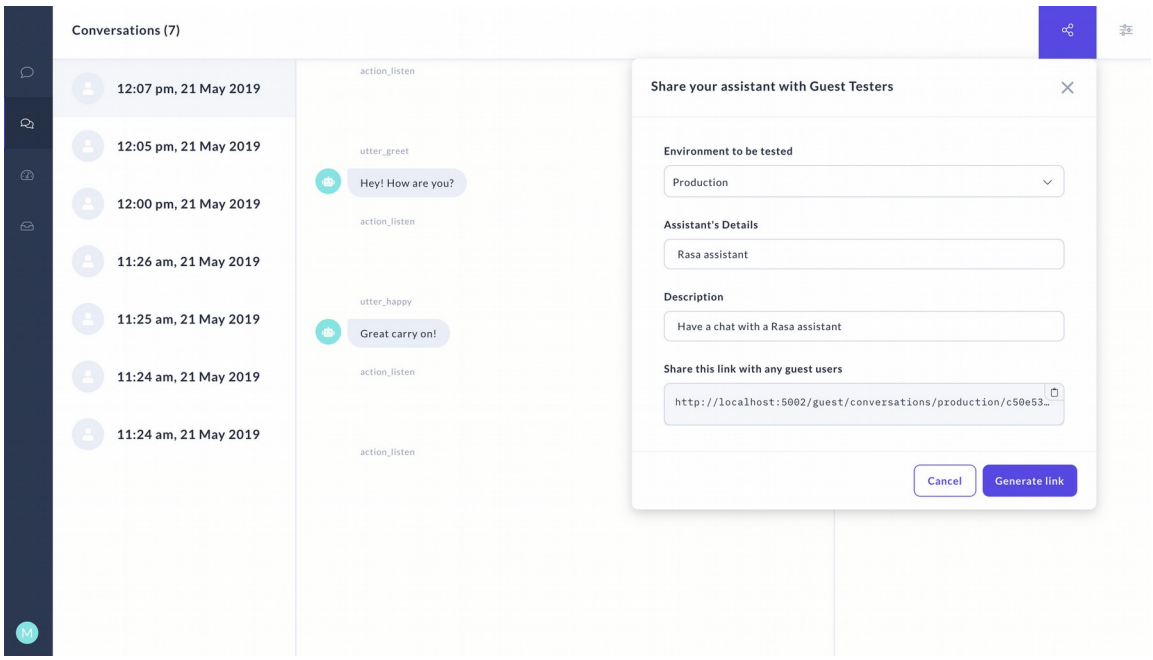
Tool Used

Rasa X

Rasa X is a tool designed to make it easier to deploy and improve Rasa-powered assistants by learning from real conversations. Rasa X is packed with new UI features which allow you to: View and annotate conversations: Filter, flag, and fix conversations that didn't go well to continually improve your assistant.



It make easier to work with nlu data by adding and modifying intents and entities.With Rasa X, it’s very easy to let others test your assistant - all you have to do is generate a link to your bot by clicking on ’Share’ icon in a *Conversations* tab and share it with your friends:



GBOT Server

Gitlab Branch : Multilingual Chatbot
Directory: GBOT_Server

This directory contains the programs which are required to host the server for the chatbot. This project uses a flask server, hence flask directory structure is maintained.

Important File Descriptions:-

Tapi.py

This file contains a function which acts as a user friendly interface to the Translation API. The localhost Translation server IP address is provided as a default argument to the function. The user just needs to provide the query to this function, this function will invoke the Translation API and provide the translated query back to the caller.

rasapi.py

This file contains a function which acts as a user friendly interface to the Rasa API. The localhost Rasa Server IP address is provided as a default argument to the function. The user just needs to provide the query to this function, this function will invoke the Rasa API and provide the response to the query back to the caller.

app.py

This is the server program, it is the master of this system. It receives queries from the user, translates it to english by invoking the Translation API, feeds this output to the Rasa Server which determines the suitable response, translates the response to the Native language.

This server can be started by using the command:-
python app.py

Rasa API Server

Gitlab Branch : Multilingual Chatbot
Directory: Rasa_API_Server

This directory contains the files which are reqired to host the Rasa API Server. The directory structure is the same as for a Rasa project.

To start the Rasa API Server use the command:-
rasa run -m models --enable-api --cors “*” --debug

Translation API Server

Gitlab Branch : Multilingual Chatbot
Directory: TRANSLATION_API_Server

This directory contains the files which are reqired to host the Translation API Server. The directory structure is the same as specified in the Translation Section below.

Important File Descriptions:-

final.py
Contains functions translate and translate_rev which translates text in native language to english and vice versa.

api_server.py
This is the API server program.
The Translation API Server can be started using the command :-
python api_server.py

Translation

Objective :
To translate from local langauges to english and viceversa.
Machine translation is used so as to enable the chatbot to percieve information in english.
Chatbot is designed to operate in english and hence all native languages have to converted to english for chatbot to operate.

Set Up:

Implemented in python v3.7
Using conda for creating virtual environments is preferrable(hassle free and easy to export the environment files).

Uses the following tools
- tensorflow version 2.0.0-beta1
- scikitlearn
- matplotlib(if graphical plots are necessary)
- numpy(it it doesn't already come with tensorflow)
- indic-trans

For cpu only machine the beta version is supposed to be run on newer cpus and hence the parallelisation is not effective for old cpu architectures.
Therefore running on cpus alone may prove time consuming(upto 8X the time required for gpu)

For gpu:
- Seperate installation is required for gpu rendering.
- Allows for high gpu usage and therefore faster execution.
- Cuda version 10 is required for using the same. Doesn't seem to be compatible for otherversions.
- In case supporting libraries are not present it might render it using cpus else it mayraise error.

Prefferable H/W specs:
- If higher amount of data is to be used for training:
  - Ram : Atleast 16GB
    Prefferable 64GB
  - GPU memory : Atleast 8GB
    Prefferable 16GB
  - CPU : Atleast gen8 i5
    Prefferable gen8 i7
  - Harddisk memory : a minimum of 10GB is to be made available(not a performance criteria).
  - Swap Memory: Atleast 2GB
    Prefferable 8-16GB(optional if high ram is available)

Observations:

- Doesn't seem viable to run on cpu alone.
- Memory requirement increases exponentially with increase in dataset size,embedding dimension or policy,datatypes used,...
- A maximum of 35000 lines in dataset(may vary according to length of lines) can be handled using 16GB RAM memory.

Machine Translation model

The model is an encoder-decoder GRU with attention and teacher forced learning.
This is one of the most effective models available in the internet at the moment of writing this documentation.

Pros

- Faster execution.
- Seems to handle much more data for training.
- Seems to learn to isolate individual words if the dataset is good enough.
- Able to create new grammatic structure even if it is not available in the dataset.
- GRU cells give better result in the case of shorter strings and is easier to implement than LSTM.
- Uses Bahdanau attention mechanism.

Cons

- Execute in beta versions which maybe unstable.
- Only a basic model; further modifications are possible to be incorporated.
- Seems to lose precision when longer strings are considered.
- Harder to implement new layers and features as the layers and functions are manually defined.
- Built-in library for cells(GRU) lack the feature to change the data type for individual memory units(in case of numpy arrays).
- Cannot handle words not in the vocabulary of dataset.

Work Arounds

For improving skill of the model:

Use LSTM cells rather than GRU
Include Bidirectional Encoder layes: upto 4 layers maximum(2 layers prefferable)
Include Decoder layers: upto 4 layers maximum(2 layers prefferable)
Embedding dimension to 512 from 256. *** warning : Might consumes a lot of memory***
Include beam search.

If memory becomes too much of an issue:

Use glove embedding or word2vec for embedding the words as vectors.
Both allow for retraining the words to match the dataset , however glove doesn't allow for addition of new vocabulary.

Observations:

- Seems to be able to handle medium to large sentences.
- Translation at a time is thus limited to a single sentence at a time.
- Fails when completely new grammar is to be generated.

Dataset

For acquiring already available datasets:

IITBOMBAY

http://www.cfilt.iitb.ac.in/iitb_parallel/

Wouldn't prefer to use this dataset unless millions of line can be accomodated in the model.
Data contains technology related terms rather than generic talks.

for parallel corpus on varied language pairs

http://www.manythings.org/anki/

manythings.org is prefferable as its dataset are configured to be used in machine translation.

http://opus.nlpl.eu/

contains pairs from multiple sources. However, it contains context which might not be acceptable based on the theme of the movies. Hence, it is not prefferable.

for custom building of datasets:

Acquire english sentences of the words.

For Translation:

Manual Translation:

Time consuming ; may not be viable.

Automated Translation:

Use google translate(options are available for uploading as a doc)
Character limit is 5000 per translation.
No daily quota on translation is imposed however, at the moment of writing this.

Using Translate from pypi.org

Translation is offered by MyMemory and microsoft.
Daily limit of 1000 words and 10000 words if registered with valid email adress.

Pros of automated translation:

- Easier to generate dataset on varied languages.
- Unable to provide manual translation for all langauges.

Cons of automated translation:

- Aggregate Errors:
  Existing translation mechanism have inherent error associated with it.
  Training model on this errenous data might lead to aggregate error
  eg : If transalted dataset has 90% accuracy and training model has 90% accuracy , aggregate accuracy might be 81%(90% of 90%).

- Translation limit imposed by packages.
- Google doesn't provide translation api for free hence, must be translated using their interfaces only.

Criterias to look for while creating dataset:

Try to include more vocabulary in fewer sentences.
Dataset size must be optimum i.e. not too much lines and not too less lines.
Sentences containing few too words(<5 words) must be kept to a minimum : maximum of 10-20% of dataset maybe allocated for this.
Sentences with single lines may be completely avoided unless essential.
A dataset working in one translation scheme might not be effective for the reverse translation.

Many to one relationships in sentences is encouraged though it can harm translation in reverse if the same dataset is used for trainig the reverse translation model.

Additional:

For transliteration indic-trans library is used. It can convert english representation to native(Indian) language represention.

Installation:

clone the git directory of indic trans library
git repository: https://github.com/libindic/indic-trans
Type the following commands:

```
cd indic-trans
pip install -r requirements.txt
python setup.py install
```

if import issues arise the use this command

```
python setup.py build_ext --inplace
```

Issues faced and possible resolution:

Issue:

There is no transliteration standard:
This implies that local language words may have multiple representations in english.
Indictrans and google translate seems to have different word representations.

Resolution:

Generate the translation in native representation.
Use a standard(eg:indictrans) to generate the training dataset.
Use another layer for transliteration at both input and output ends.

Issue:

Punctuation marks generates different meaning for same sentences.

Resolution:

Either take out punctuation completely or ensure that all the sentences have punctuations.
Preferrably take out all punctuations as speech to text does not take into consideration the punctuations.
Use regular expressions extensively to accomodate all exceptions(not preffered).

Issue:

Cannot isolate phrases from complete sentences.

Resolution:

Switch to LSTM cells for a lift in skill and try implementing the other criterias also.

Issue:

Cannot handle large datasets.
This might be due to the datatype used in numpy arrays eg. for one hot encoding.

Resolution:

Try alternate embedding schemes(eg:glove,word2vec).
Try parallesing the input dataset. Might affect the performance due to memory swap time.
Run on a machine with higher RAM.

In case none of the above works do the following:

**** Warning: Use this technique only if u have a highly constrained domain for translation ****
Generate the dataset with only the essential translation schemes.
Take out the test or validation dataset and put the entire sentences to train data.
Increase the number of epochs to overfit the model.
In this way the model is familiar with all the required data lines.
The model is now "critically overfit" , however it will generate translation for any data line in the train data set with near 100% accuracy, but, fails with new sentences.
This can be done when the queries and responses are limited.

Package Details:

Gitlab branch: Translation
Contains 3 directories and 1 file

File: transliterator.py
Description : Used for transliterating from native language to english or vice versa.
Run the script and input the data in the specified langauge eg: hindi in Devnagari.

Directories:

hin_to_english and **english_to_hin**
These are for training the model for hindi to english and english to hindi translations respectively.
The dataset used for training is final_dataset.txt and final_dataset_rev.txt for hindi to english and english to hindi respectively. The files must be tab separated formats in either cases. In both the directories run the attention_model.py for training the model. The models are saved in the corrsespondind folders in the respective directories. “translation.py” is the python package for transalation. It is accessed by pkg_trial.py . It gives an interactive script for testing the translation of the models. Ensure that the same dataset and the parameters are used in both translation.py and attention_model.py.

final
This folder contains the python package to be called in the API.
For checking the package run pkg_trial.py.

References:

tensorflow.org : https://www.tensorflow.org/beta/tutorials/text/nmt_with_attention
anaconda.org (for packages)
machinelearningmastery.com
medium.com
towardsdatascience.com