TPDS Project

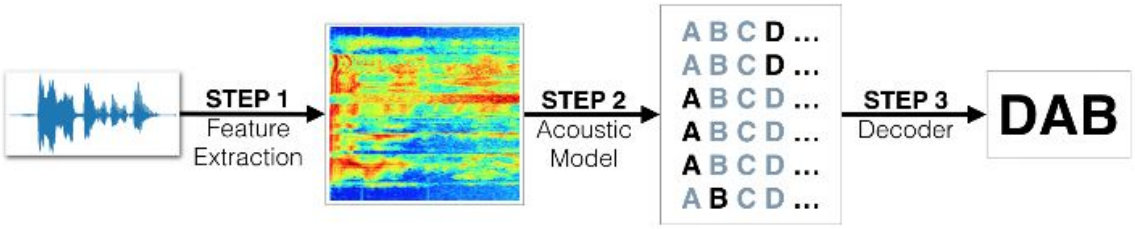
Ashutosh Upadhye, SST Siddhardha, Rahul Dhawan

This report includes the implementation procedures of our Project as part of TPDS course at the Indian Institute of Technology, Palakkad. This report includes the introduction section, literature survey, implementations followed by results and conclusions.

Speech Recognition

Introduction

Speech recognition is one of the most recently developing field of research at both industrial and scientific levels. Until recently, the idea of holding a conversation with a computer seemed pure science fiction. If you asked a computer to “open the pod bay doors”—well, that was only in movies. But things are changing, and quickly. A growing number of people now talk to their mobile smart phones, asking them to send email and text messages, search for directions, or find information on the Web. The first step towards the world of speech recognition problem is digit recognition.



Problem Statement

For a given audio file in which the speaker will speak a sequence of numbers, recognize the number being said.

Literature Survey

Books are available to read and learn about speech recognition ,these enabled us to see what happens beyond the code. "Automatic Speech Recognition: A Deep Learning Approach" (Publisher: Springer) written by D. Yu and L. Deng published near the end of 2014, with highly mathematically-oriented technical detail on how deep learning methods are derived and implemented in modern speech recognition systems based on DNNs and related deep learning methods.This gave us an insight into the conversion algorithm used by Google.

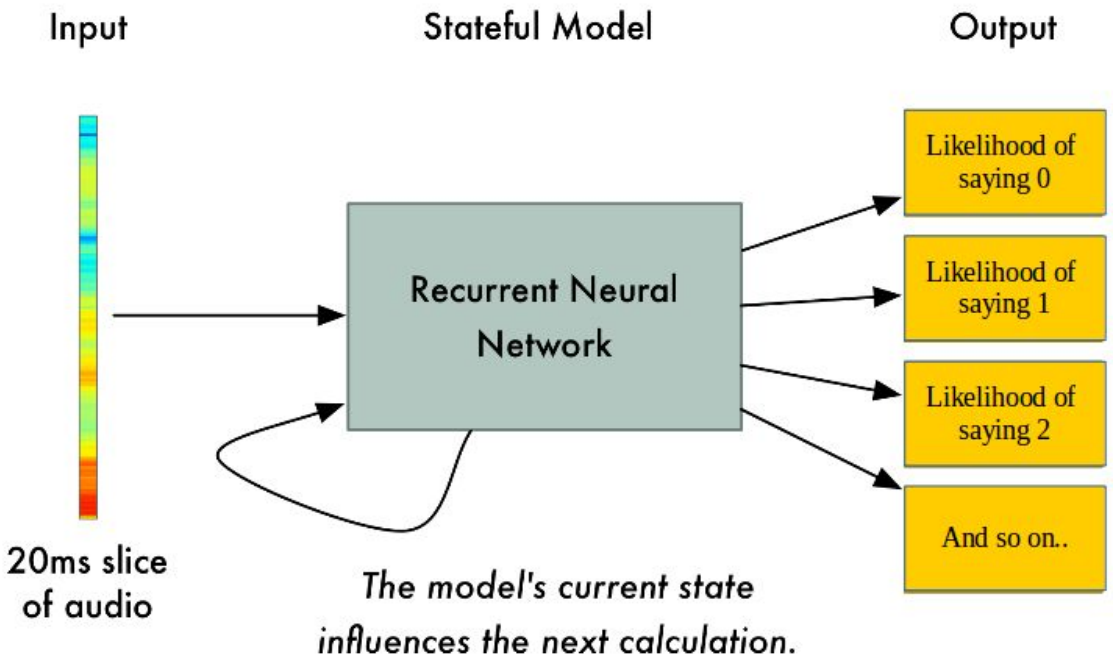
Here are some IEEE and other articles we referred :

[1]Waibel, Hanazawa, Hinton, Shikano, Lang. (1989) "Phoneme recognition using time-delay neural networks. IEEE Transactions on Acoustics, Speech and Signal Processing."

[2]Reynolds, Douglas; Rose, Richard (January 1995). "Robust text-independent speaker identification using Gaussian mixture speaker models" (PDF). IEEE Transactions on Speech and Audio Processing (IEEE) 3 (1): 72–83. doi:10.1109/89.365379. ISSN 1063-6676. OCLC 26108901. Retrieved 21 February 2014.

Model and Algorithm

Overview of the model



Step 1: Acoustic Features for Speech Recognition

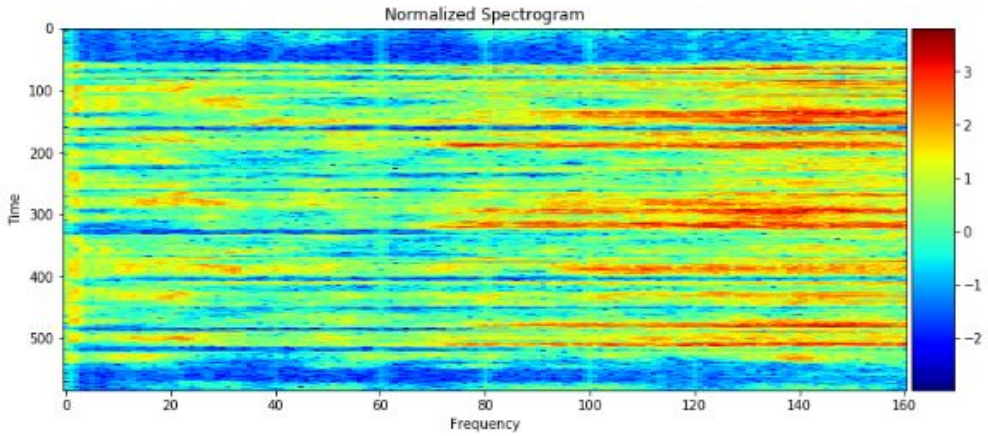
STEP 1 is a pre-processing step that converts raw audio to one of two feature representations that are commonly used for ASR.

Spectrograms:

A spectrogram is a visual representation of the spectrum of frequencies of sound or other signal as they vary with time. Spectrograms are sometimes called spectral waterfalls, voiceprints, or voicegrams.

Spectrograms may created from a time-domain signal in one of two ways: approximated as a filterbank that results from a series of band-pass filters, or calculated from the time signal using the Fourier transform. These two methods actually form two different time–frequency

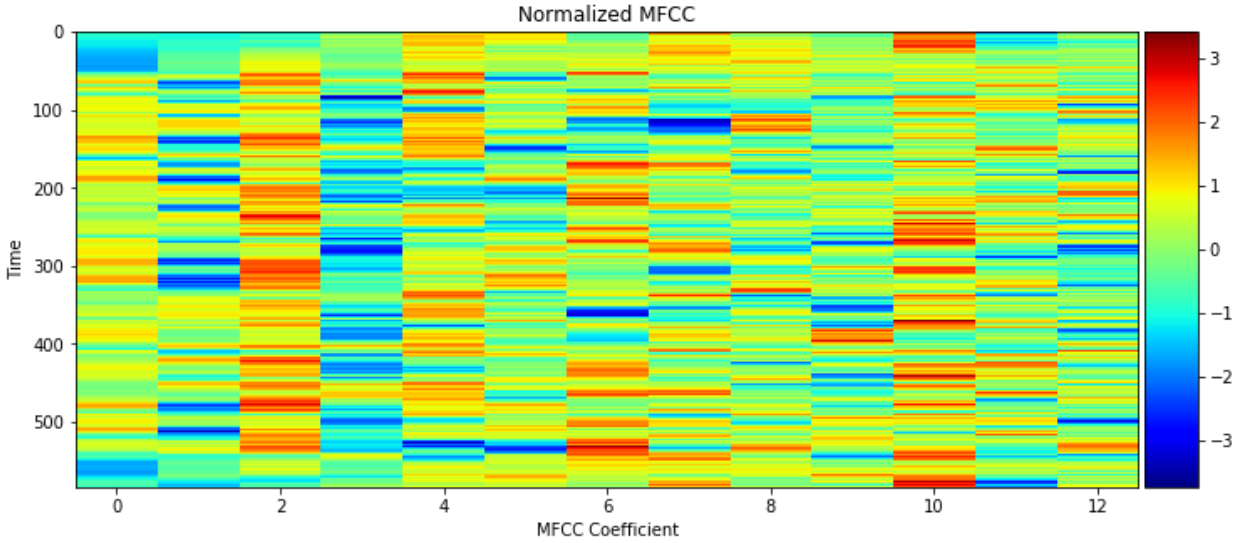
representations, but are equivalent under some conditions.



Mel-Frequency Cepstral Coefficients (MFCCs):

In sound processing, the mel-frequency cepstrum (MFC) is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency. MFCCs are

commonly used as features in speech recognition systems, such as the systems which can automatically recognize numbers spoken into a telephone.



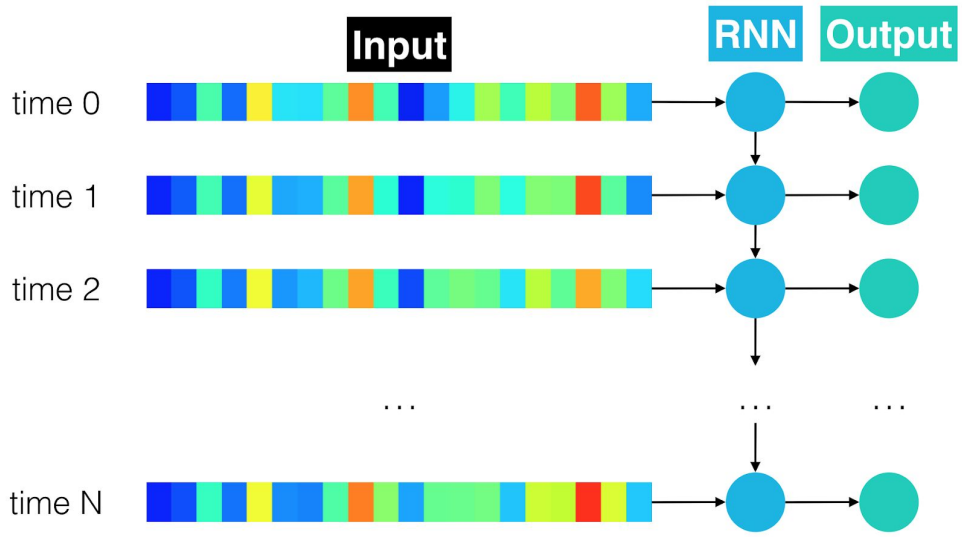
Step 2: Deep Neural Networks for Acoustic Modeling

STEP 2 is an acoustic model which accepts audio features as input and returns a probability distribution over all potential transcriptions.

Recurrent Neural Networks:

A recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed graph along a sequence. This allows it to exhibit dynamic temporal

behavior for a time sequence. Unlike feedfor- ward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition.



Step 3: Obtaining Predictions

STEP 3 in the pipeline takes the output from the acoustic model and returns a predicted digit. This is rather trivial.

Implementation

Data Collection:

The dataset we are going to use for the project is collection of sound clips of spoken digits from 0 to 9 (‘*.wav’ file). We will use 60 x 10 sample points for training and 4 x 10 samples for testing.

Data Preprocessing:

To extract the useful features from sound data, we will use *Librosa* library. It provides several methods to extract different features from the sound clips.

To make the process of feature extraction from sound clips easy, two helper methods are defined. First **parse_audio_files** which takes parent directory name, subdirectories within parent directory and file extension (default is .wav) as input. It then iterates over all the files within subdirectories and call second helper function extract_feature. It takes file path as input, read the file by calling *librosa.load* method, extract and return features discussed above. These two methods are all that is required to convert raw sound clips into informative features (along with a class label for each sound clip) that we can directly feed into our classifier.

Codecs:

Now we have our training and testing set ready , now only things left is to implement the model to classify the test data into predicted labels. We have several methods to implement that:

1. Neural Network
2. KNN
3. SVC
4. LogisticRegression
5. LogisticRegressionCV
6. RandomForestClassifier
7. Naive Bayes

We are going for the Random Forest Classifier because it is giving maximum accuracy for classifying. The others are not giving because they need more training data for classifying but we have very less data and we are trying to generate more data by speaking manually and storing.The approximate accuracy on the current data are:

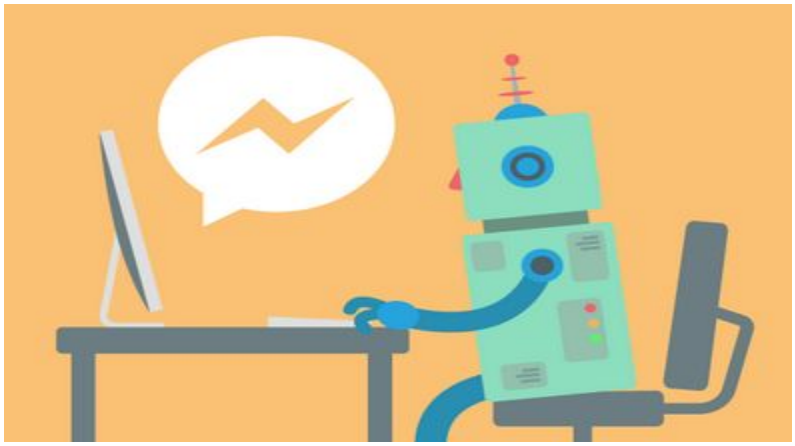
Class	N-N	KNN	SVC	LR	LRCV	RFC	NB
Acc	0.1	0.3	0.1	0.37	0.3	0.575	0.2

The classifiers are showing these much less accuracy because of less training data and we are generating the more data manually. We cannot improve the results because the data generation step was bottleneck.

Chatbot

Introduction

A **chatbot** (also known as a **talkbot**, **chatterbot**, **Bot**, **IM bot**, **interactive agent**, or **Artificial Conversational Entity**) is a computer program which conducts a conversation via auditory or textual methods. Such programs are often designed to convincingly simulate how a human would behave as a conversational partner, thereby passing the Turing test. Chatbots are typically used in dialog systems for various practical purposes including customer service or information acquisition. Some chatterbots use sophisticated natural language processing systems, but many simpler systems scan for keywords within the input, then pull a reply with the most matching keywords, or the most similar wording pattern, from a database.



Problem Statement

a. Given a natural language sentence try to predict a suitable natural language response.
b. Implementing Sequence to Sequence encoder decoder

Literature Survey

What is sequence-to-sequence learning?

Sequence-to-sequence learning (Seq2Seq) is about training models to convert sequences from one domain (e.g. sentences in English) to sequences in another domain (e.g. the same sentences translated to French).

```
"the cat sat on the mat" -> [Seq2Seq model] -> "le chat etait assis sur le tapis"
```

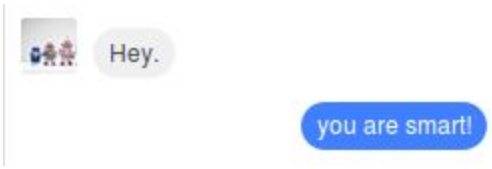
This can be used for machine translation or for free-from question answering (generating a natural language answer given a natural language question) -- in general, it is applicable any time you need to generate text.

There are multiple ways to handle this task, either using RNNs or using 1D convnets. Here we will focus on RNNs.

The general case: canonical sequence-to-sequence

In the general case, input sequences and output sequences have different lengths (e.g. machine translation or chatbot) and the entire input sequence is required in order to start predicting the target. This requires a more advanced setup, which is what people commonly refer to when mentioning "sequence to sequence models" with no further context. Here's how it works:

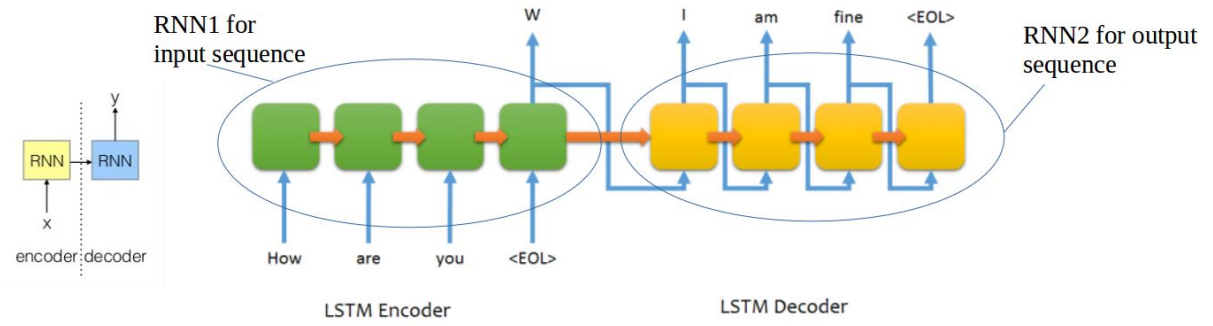
- A RNN layer (or stack thereof) acts as "encoder": it processes the input sequence and returns its own internal state. Note that we discard the outputs of the encoder RNN, only recovering the state. This state will serve as the "context", or "conditioning", of the decoder in the next step.
- Another RNN layer (or stack thereof) acts as "decoder": it is trained to predict the next characters of the target sequence, given previous characters of the target sequence. Specifically, it is trained to turn the target sequences into the same sequences but offset by one timestep in the future, a training process called "teacher forcing" in this context. Importantly, the encoder uses as initial state the state vectors from the encoder, which is how the decoder obtains information about what it is supposed to generate. Effectively, the decoder learns to generate targets[t+1...] given targets[...t], *conditioned on the input sequence*.



Deep Neural Networks (DNNs) are powerful models that have achieved excellent performance on difficult learning tasks. Although DNNs work well whenever large labeled training sets are available, they cannot be used to map sequences to sequences. So our method uses a multilayered Long Short-Term Memory (LSTM) to map the input sequence to a vector of a fixed dimensionality, and then another deep LSTM to decode the target sequence from the vector.

http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf?utm_campaign=DonanimHaber&utm_medium=referral&utm_source=DonanimHaber

Overview:



Step 1: Preprocessing

We have decided to use Cornell Movie Dialogue Corpus dataset. This corpus contains a large metadata-rich collection of fictional conversations extracted from raw movie scripts.

Vocaboulary Extraction:

We have to extract the vocablulary from the training dataset and map each word to an integer.

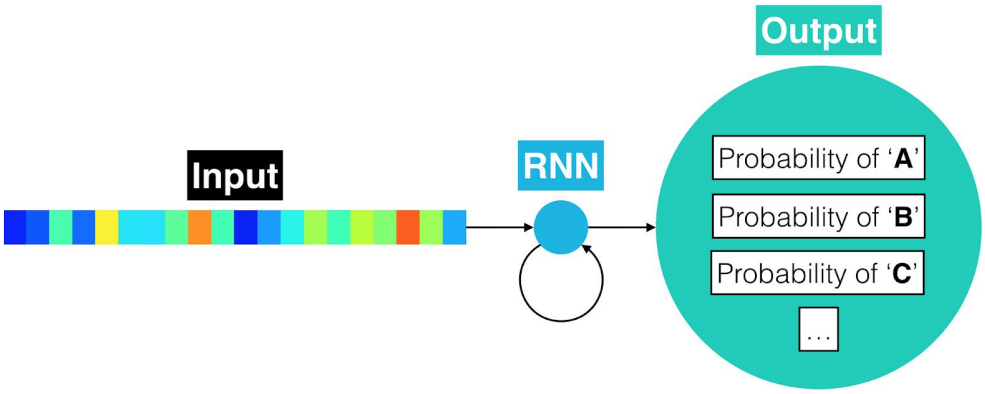
Tokenizer:

All the words in the speech have to be tokenised in order to generate a sequence of words as input.

Step 2: Model

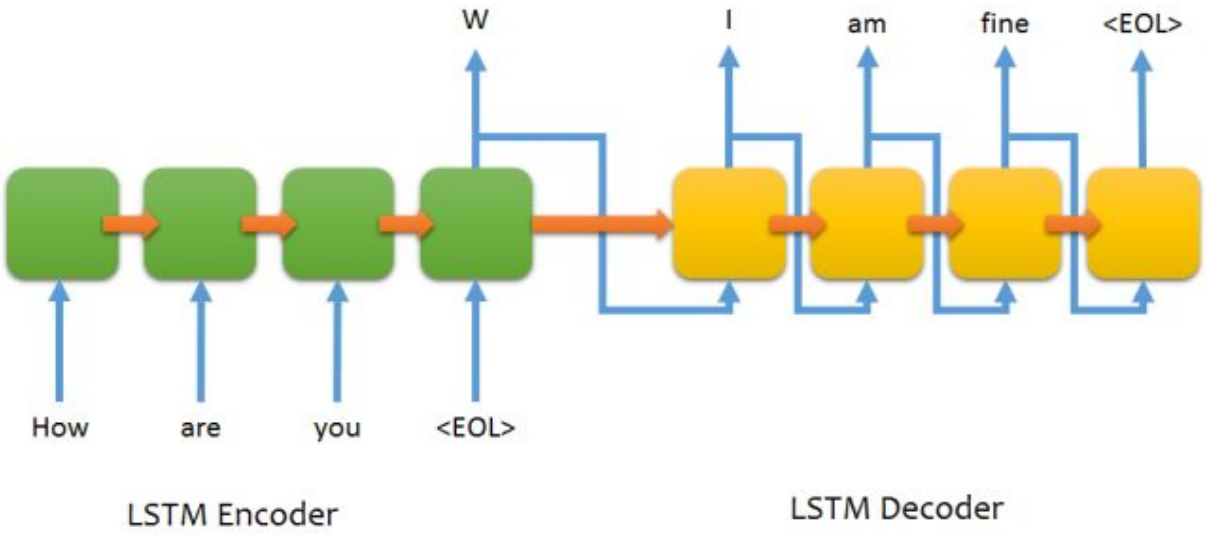
Recurrent Neural Networks:

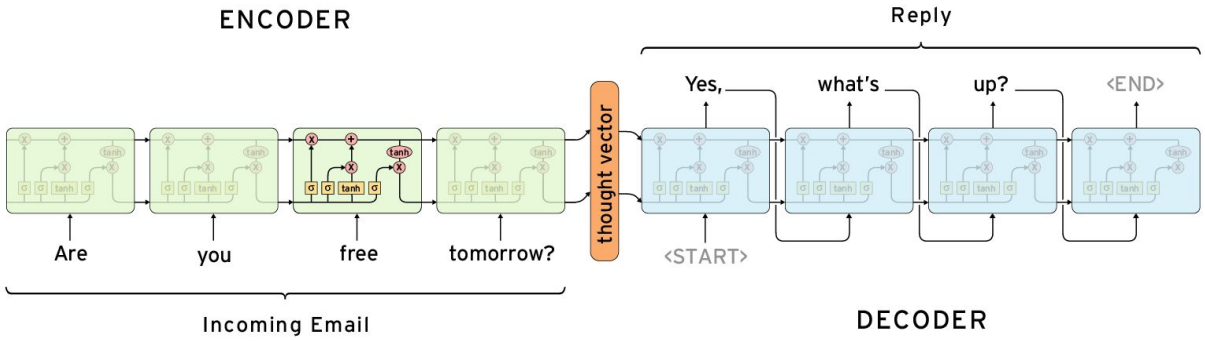
A recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed graph along a sequence. This allows it to exhibit dynamic temporal behavior for a time sequence. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition.



Sequence to Sequence model: RNN Encoder-Decoder

RNN Encoder-Decoder that consists of two recurrent neural networks. One RNN encodes a sequence of symbols into a fixed-length vector representation, and the other decodes the representation into another sequence of symbols.





There are various proposals made in various papers. Reversing the input sequence is proven to be better than the ordinary one. Padding the input supposedly yields better output and so on. These could be tried out. Sequence to Sequence model is the best for chatbot kind of applications.

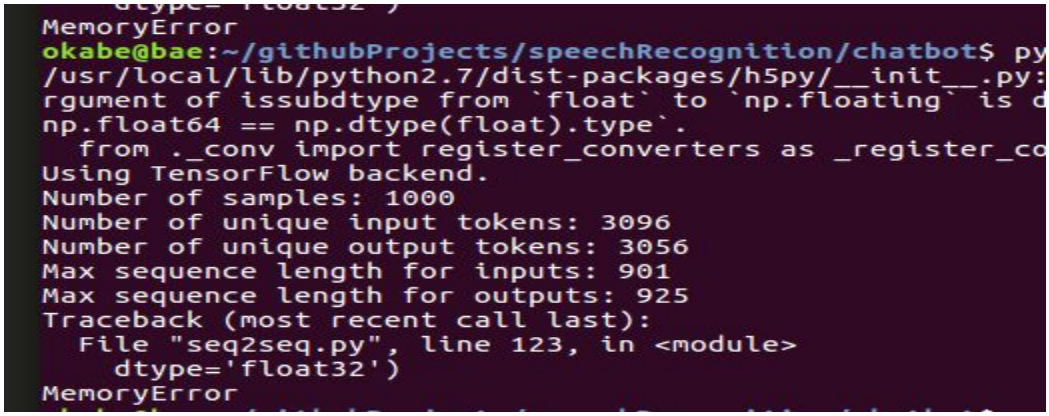
Outputs:

Overview:

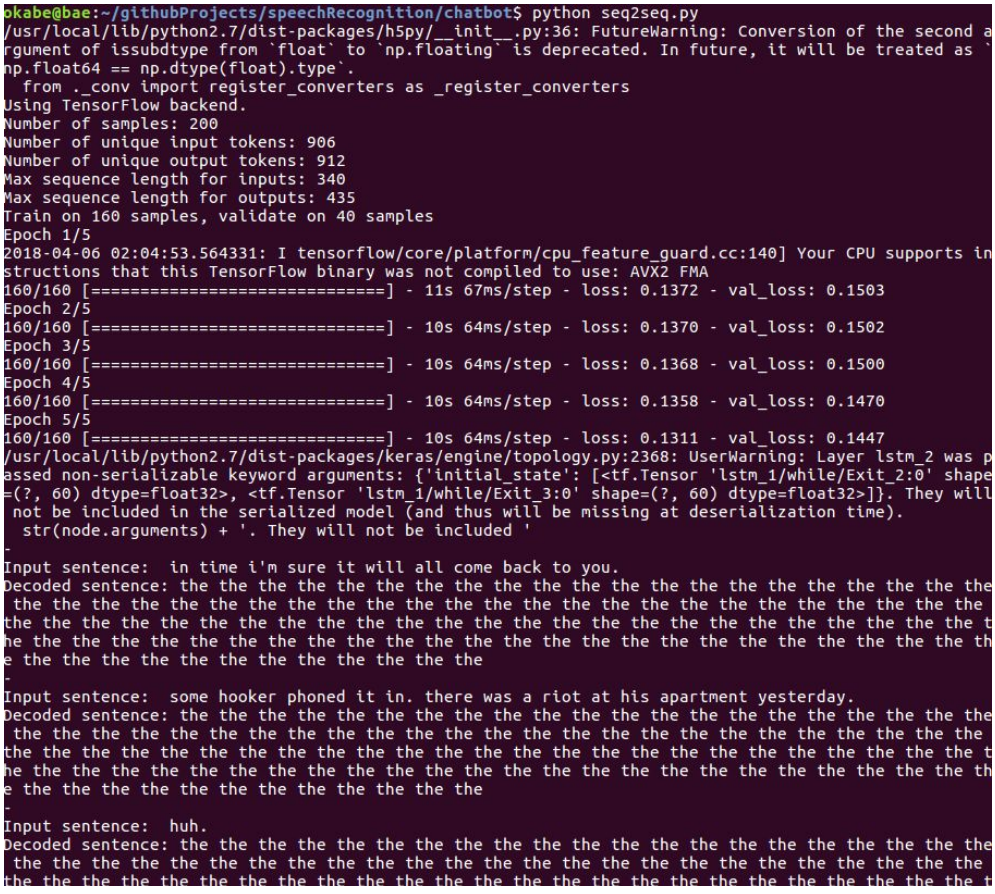
The accuracy scores are as listed below.

Test accuracy	13.11
Validation accuracy	14.47

We have tried running the encoder decoder program with larger data samples but there was insufficient memory exception.



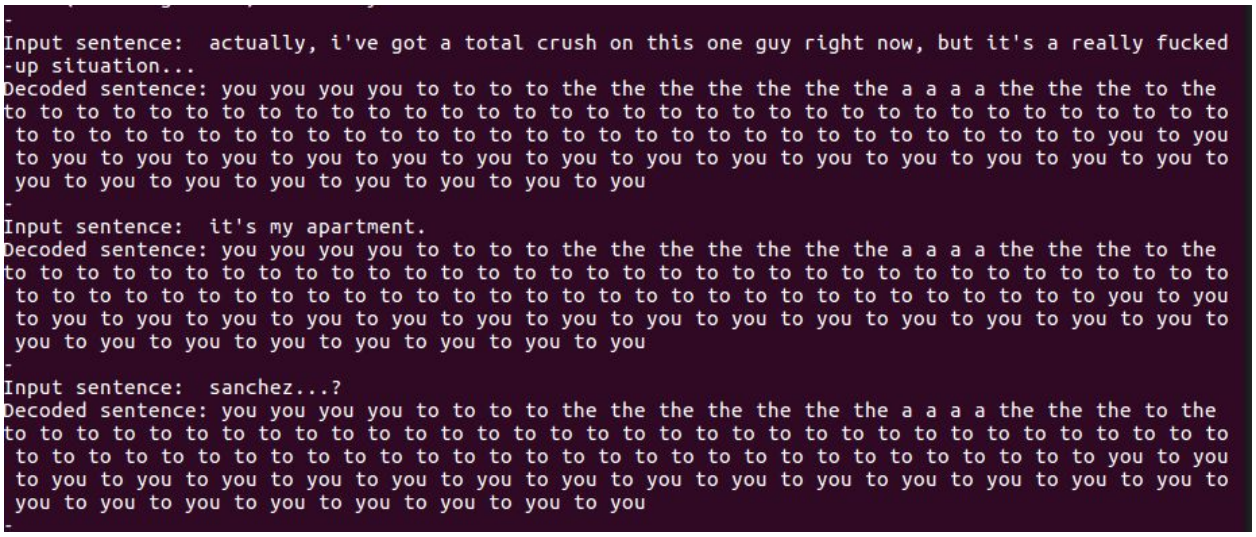
On running with smaller value of data sample, the program takes off, however the results are not very appealing. The output is as mentioned below.



On running with slightly more data set, and 20 epochs, we get the following output:

```
okabe@bae:~/githubProjects/speechRecognition/chatbot$ python seq2seq.py
/usr/local/lib/python2.7/dist-packages/h5py/__init__.py:36: FutureWarning: Conversion of the second a
rgument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `
np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
Number of samples: 500
Number of unique input tokens: 1859
Number of unique output tokens: 1900
Max sequence length for inputs: 442
Max sequence length for outputs: 437
Train on 400 samples, validate on 100 samples
Epoch 1/20
2018-04-06 02:09:26.740507: I tensorflow/core/platform/cpu_feature_guard.cc:140] Your CPU supports in
structions that this TensorFlow binary was not compiled to use: AVX2 FMA
400/400 [==============================] - 97s 242ms/step - loss: 0.1609 - val_loss: 0.1461
Epoch 2/20
400/400 [==============================] - 97s 242ms/step - loss: 0.1425 - val_loss: 0.1464
Epoch 3/20
400/400 [==============================] - 104s 260ms/step - loss: 0.1382 - val_loss: 0.1490
Epoch 4/20
400/400 [==============================] - 106s 265ms/step - loss: 0.1369 - val_loss: 0.1504
Epoch 5/20
400/400 [==============================] - 105s 264ms/step - loss: 0.1361 - val_loss: 0.1530
Epoch 6/20
400/400 [==============================] - 105s 264ms/step - loss: 0.1351 - val_loss: 0.1540
Epoch 7/20
400/400 [==============================] - 105s 261ms/step - loss: 0.1344 - val_loss: 0.1565
Epoch 8/20
400/400 [==============================] - 110s 275ms/step - loss: 0.1338 - val_loss: 0.1595
Epoch 9/20
400/400 [==============================] - 106s 265ms/step - loss: 0.1331 - val_loss: 0.1616
Epoch 10/20
400/400 [==============================] - 106s 266ms/step - loss: 0.1325 - val_loss: 0.1629
Epoch 11/20
400/400 [==============================] - 105s 264ms/step - loss: 0.1319 - val_loss: 0.1651
Epoch 12/20
400/400 [==============================] - 105s 262ms/step - loss: 0.1312 - val_loss: 0.1648
Epoch 13/20
400/400 [==============================] - 108s 270ms/step - loss: 0.1306 - val_loss: 0.1658
Epoch 14/20
400/400 [==============================] - 106s 266ms/step - loss: 0.1300 - val_loss: 0.1662
Epoch 15/20
400/400 [==============================] - 106s 266ms/step - loss: 0.1293 - val_loss: 0.1684
Epoch 16/20
400/400 [==============================] - 112s 280ms/step - loss: 0.1289 - val_loss: 0.1678
Epoch 17/20
400/400 [==============================] - 105s 263ms/step - loss: 0.1281 - val_loss: 0.1705
Epoch 18/20
400/400 [==============================] - 106s 264ms/step - loss: 0.1278 - val_loss: 0.1689
Epoch 19/20
400/400 [==============================] - 109s 272ms/step - loss: 0.1272 - val_loss: 0.1692
Epoch 20/20
400/400 [==============================] - 110s 276ms/step - loss: 0.1269 - val_loss: 0.1701
/usr/local/lib/python2.7/dist-packages/keras/engine/topology.py:2368: UserWarning: Layer lstm_2 was p
assed non-serializable keyword arguments: {'initial_state': [<tf.Tensor 'lstm_1/while/Exit_2:0' shape
=(?, 256) dtype=float32>, <tf.Tensor 'lstm_1/while/Exit_3:0' shape=(?, 256) dtype=float32>]}. They wi
ll not be included in the serialized model (and thus will be missing at deserialization time).
```

The chat output for this chatbot looks something like the following.



Future Works:

The chatbot can be trained with ‘pre existing’ seq2seq encoder decoder libraries to make it more efficient and usable.

The chatbot can be trained based on feedback from users on the decoded sequence.

The chatbot can be trained on customer service dialogue(call center call recordings), which can be recorded and the speech can be converted to text using ‘**speech recognition**’ program.

Code snippets:

1. Pre processing the data: converting the raw movie corpus to usable dialogue.

```
data_path = 'movie_lines.txt'
with open(data_path, 'r') as f:
    lines = f.read().split('\n')
op = open ('data.txt', 'w')
op2 = open('data1.txt','w')
i = 0
while i < len(lines):
    print lines[i].split('+++$+++')[4].split('\t')
    try:
        line = []
        line = lines[i].split('+++$+++')[4].split('\t')
        st = ""
        for j in range(0,len(line)):
            st = st+line[j]+" "
        line1 = []
        line1 = lines[i+1].split('+++$+++')[4].split('\t')
        st1 = ""
        for j in range(0,len(line1)):
            st1 = st1+line1[j]+" "
        op.write(st+'\t'+st1+'\n')
        #op1.write(lines[i].split('+++$+++')[4].split('\t')[0]
    except:
        break
    i += 2
op.close()
```

2. Defining the layers and linking them to implement seq2seq encoding decoding. Please note that here we have not used any pre existing library function for seq2seq encoding decoding.

```
encoder_inputs = Input(shape=(None, num_encoder_tokens))
encoder = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_inputs)
# We discard `encoder_outputs` and only keep the states.
encoder_states = [state_h, state_c]

# Set up the decoder, using `encoder_states` as initial state.
decoder_inputs = Input(shape=(None, num_decoder_tokens))
# We set up our decoder to return full output sequences,
# and to return internal states as well. We don't use the
# return states in the training model, but we will use them in inference.
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_inputs,
                                     initial_state=encoder_states)
decoder_dense = Dense(num_decoder_tokens, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)

# Define the model that will turn
# `encoder_input_data` & `decoder_input_data` into `decoder_target_data`
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)

# Run training
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
model.fit([encoder_input_data, decoder_input_data], decoder_target_data,
          batch_size=batch_size,
          epochs=epochs,
          validation_split=0.2)
# Save model
model.save('s2s.h5')

```

References:

[1]http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf?utm_campaign=DonanimHaber&utm_medium=referral&utm_source=DonanimHaber

[2]https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in- keras.html

[3] Waibel, Hanazawa, Hinton, Shikano, Lang. (1989) "Phoneme recognition using time-delay neural networks. IEEE Transactions on Acoustics, Speech and Signal Processing."

[4] Reynolds, Douglas; Rose, Richard (January 1995). "Robust text-independent speaker identification using Gaussian mixture speaker models" (PDF). IEEE Transactions on Speech and Audio Processing (IEEE) 3 (1): 72–83. doi:10.1109/89.365379. ISSN 1063-6676. OCLC 26108901. Retrieved 21 February 2014.

[5] Sequence to Sequence Learning with Neural Networks by Ilya Sutskever, Oriol Vinyals, Quoc V. Le.