# Speech Recognition Engine

Hritik Raj (ES17BTECH11009)

December 16, 2020

## Contents

## 1 Introduction

This consists details of speech recognition engine for a voice bot. The model used is based on concepts of Convolution, LSTM and Attention.

## 2 Data Generation

Used Audacity to generate your own voice samples.

1. Set the sampling rate to 16 kHz
2. Record 80 utterances of each command - forward, back, right, left, stop.
3. Trim each utterance to one second.
4. Save samples of each command in different folders as
   Dataset/forward
   Dataset/back
   Dataset/left
   Dataset/right
   Dataset/stop

Use soundfile package to read the .wav. Other package like wavefile, librosa etc could also be used.
Store the loaded data as numpy file for ease and speed of access. Now, the data can be loaded from the numpy file if the experiment is repeated.

## 3 Split dataset

Split the dataset into train and test set with 80% as train samples and 20% as test samples.
Set a random seed for reproducing the split.

## 4 Augment data

Augment each audio sample by time shifting in 25000 length vectors filled with zeros.
Take steps of 500 to create 18 files per sample

## 5 Feature Extraction

MFCCs(Mel-Frequency Cepstral Coefficients) feature extraction is a leading approach for speech feature extraction.Normalizing the MFCCs over the frequency axis is found to reduce effect of noise.

Kapre is a python package that provides layers for audio processing that are compatible with keras and utilize GPU for faster processing. Kapre provides us with a layer basically.

*Melspectrogram (padding = 'same', sr=16000, n_mels = 39, n_dft = 1024, power_melgram = 2.0, return_decibel_melgram = True, trainable_fb = False, trainable_kernal = False, name = 'mel_stft')*

**Arguments to the layer**
**padding** : Padding when convoluting
**sr** : Sampling rate of audio given
**n_mels**: number of coefficients to return
**n_dft**: width
**power_melgram** : exponent to raise log-mel-amplitudes before taking DCT. Using power 2 is shown to increase performance by reducing effect of noise
**return_decibel_melgram** : If to return log over values
**trainable_fb** : If filter bank is trainable
**trainable_kernel** : If the kernel is trainable

## 6 Building Model

### 6.1 Concept

1. LSTMs(Long Short Term Memory Units) are a type of RNNs capable of learning order dependence in sequence prediction problems.
2. Several research papers show that using Convolutional layers ahead of LSTM has shown to improve performance.
3. BatchNormalization layers are added to improve convergence rate.
4. Using Bidirectional LSTM is optimal when complete input is available. But this increases the runtime two-fold.

5. Final output sequence of LSTM layer is used to calculate importance of units in LSTM using a FC layer.

6. Then take the dot product of unit importance and output sequences of LSTM to get Attention scores of each time step.

7. Take the dot product of Attention scores and the output sequences of LSTM to get attention vector.

8. Add an additional FC Layer and then to output Layer with SoftMax Activation.

### 6.2 Parameters

- **sparse_categorical_crossentropy** is used as **Loss** because only output which should be 1 is given instead of One Hot Encoding.

- **sparse_categorical_accuracy** is used as performance **Metric** for the above reason.

- **Adam** is used as **Optimizer**. Adam is adaptive learning rate optimization algorithm. This is shown to achieve a faster convergence because of having all the features of other optimization algorithms.

- Batch size of 20 is used

Overall look of the architecture of the model is shown in Fig 6 and the model diagram is shown in Fig 5

## 7 Training the Model

### 7.1 Back Propagation

Back propagation is the algorithm used to calculate the gradient of loss w.r.t all the parameters in the neural network. Gradient of Loss w.r.t parameter is the direction in which the parameter should move such that decrease in loss will be maximum. It is the practice of fine-tuning the weights of a neural network based on the error rate (i.e. loss) obtained in the previous epoch (i.e. iteration). Proper tuning of the weights ensures lower error rates, making the model reliable by increasing its generalization.
Each layer and function is programmed to calculate the gradient of loss w.r.t it's trainable parameters and input given the input and gradient of loss w.r.t its output.
Gradient of loss w.r.t input of the current layer or function is back propagated as gradient of loss w.r.t output of previous layer or function. Hence, the name back propagation.

### 7.2 Adam Optimizer

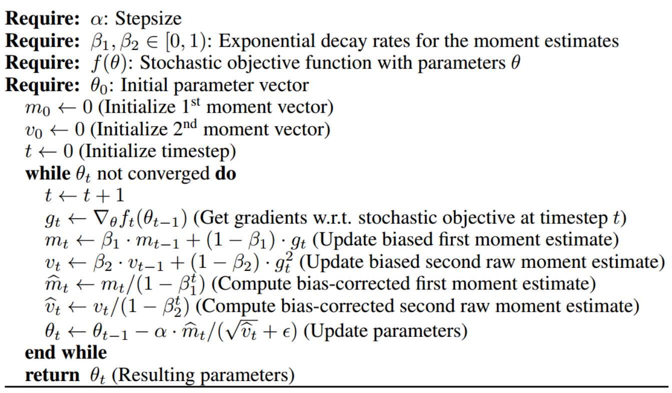
Back propagation calculates the gradient of loss w.r.t each parameter. But an adaptive learning rate algorithm will be used to update the weights in a more calculated manner.
Adam is an optimization algorithm that can be used instead of the classical SGD procedure to update network weights iterative based in training data.Adam maintains few training parameters independently for each network parameter while SGD only has a single learning rate parameter for all the weights of the network. Adam is a combination of AdaGrad and RMSProp (Two other optimization algorithms).
RMSProp uses only Average first moment (running average of the gradient) while Adam also uses Average second moment (running average of the square of gradient ) in the learning rate calculation.
For each network parameter, Adam maintains

- **Alpha** Also referred to as the learning rate or step size. Proportion of the update to be added to the weight.

- **Beta1** The exponential decay rate for the first moment estimates (e.g. 0.9).

- **Beta2** The exponential decay rate for the second-moment estimates (e.g. 0.999)

- **Epsilon** Is a very small number to prevent any division by zero in the implementation (e.g. 10E-8)



```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

Figure 1: Adam algorithm

.

## 8 Testing

1. Augment the test set same as training set.

2. Extract MFCCs same as above

3. Test set is passed as validation set to fit method of model.

4. The performance of model on test set is calculated after every epoch.

## 9 Visualize Attention

1. Now build a sub model from the trained model. Take same input layer but add ‘AttentionSoftmax’ layer as additional output layer.
2. Pass MFCCs of test samples to predict method.
3. Plot log of Attention Scores and corresponding input vector before taking MFCCs on different axes.
4. By looking at Fig 2 and Fig 3, We observe that Attention Scores are high on informative part.

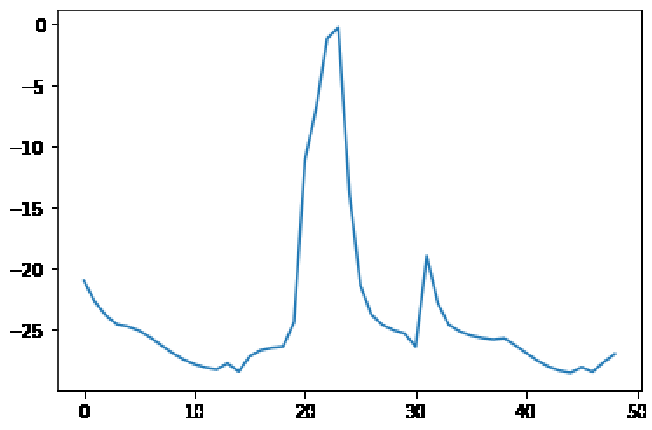


Figure 2: log(Attention Scores)

.

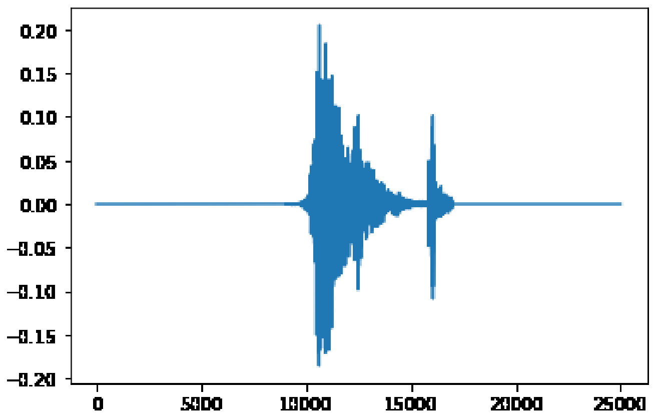


Figure 3: Raw sample

.

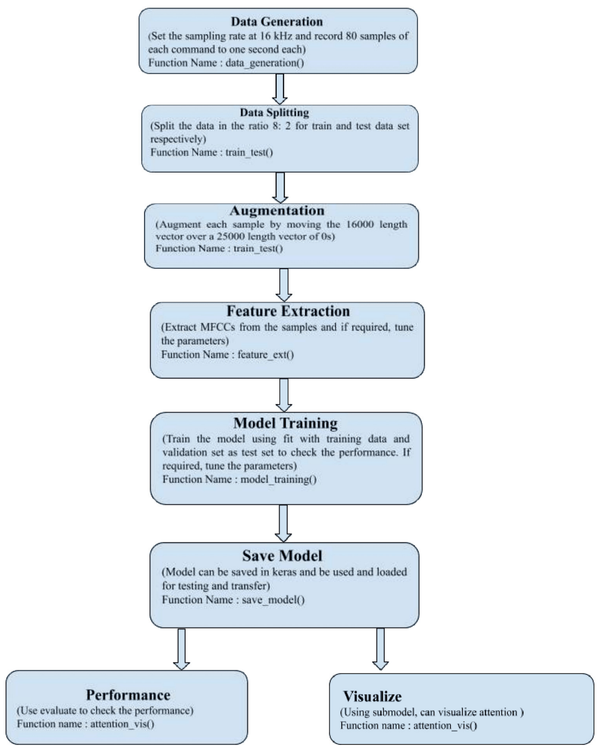
## 10 Observations

- Smaller batch size is prefferable
- Setting power_melgram=2 of Melspectogram gave faster convergence.

## 11 Functions

Each block in the flowchart can be executed through a function mentioned below

- *data_generation*() : Generates the data
- *train_test*() : Splits the data into train and test and augments the data
- *feature_ext*(): Extracts MFCC coefficients
- *model_training*(): Trains model.
- *save_model*(): Saves the model in h3 file
- *attention_vis*() : Checks Performance and visualize attention
- ColabNotebook.ipynb: Use this for experimental purpose



Data Generation
(Set the sampling rate at 16 kHz and record 80 samples of each command to one second each)
Function Name : data_generation()
Data Splitting
(Split the data in the ratio 8: 2 for train and test data set respectively)
Function Name : train_test()
Augmentation
(Augment each sample by moving the 16000 length vector over a 25000 length vector of 0s)
Function Name : train_test()
Feature Extraction
(Extract MFCCs from the samples and if required, tune the parameters)
Function Name : feature_ext()
Model Training
(Train the model using fit with training data and validation set as test set to check the performance. If required, tune the parameters)
Function Name : model_training()
Save Model
(Model can be saved in keras and be used and loaded for testing and transfer)
Function Name : save_model()
Performance
(Use evaluate to check the performance)
Function name : attention_vis()
Visualize
(Using submodel, can visualize attention )
Function name : attention_vis()

Figure 4: Flowchart

.

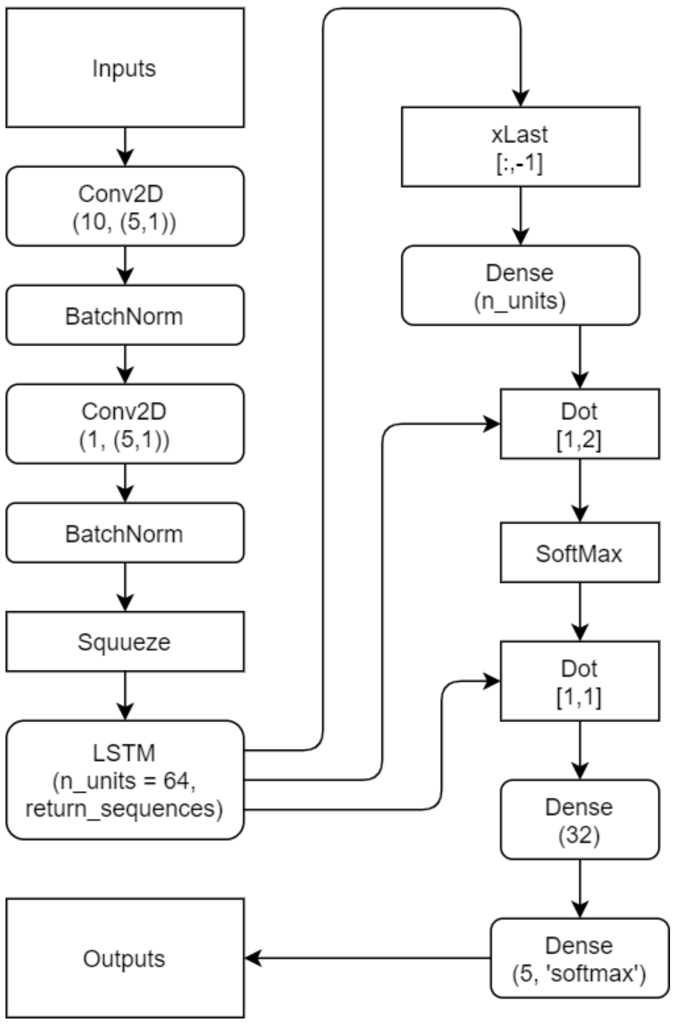
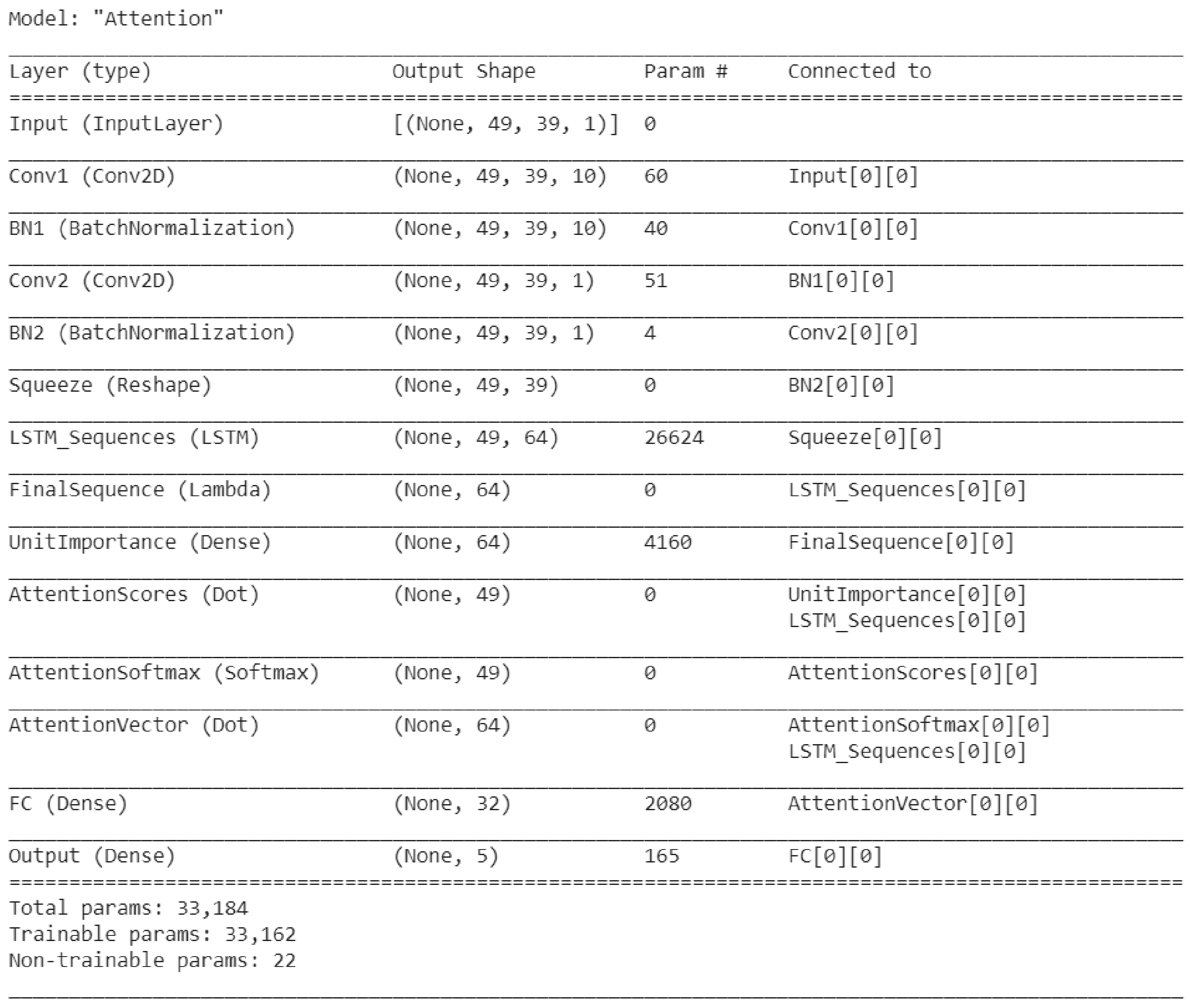


Figure 5: Model Diagram

.



```
Model: "Attention"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
Input (InputLayer)              [(None, 49, 39, 1)]  0
__________________________________________________________________________________________________
Conv1 (Conv2D)                  (None, 49, 39, 10)   60          Input[0][0]
__________________________________________________________________________________________________
BN1 (BatchNormalization)        (None, 49, 39, 10)   40          Conv1[0][0]
__________________________________________________________________________________________________
Conv2 (Conv2D)                  (None, 49, 39, 1)    51          BN1[0][0]
__________________________________________________________________________________________________
BN2 (BatchNormalization)        (None, 49, 39, 1)    4           Conv2[0][0]
__________________________________________________________________________________________________
Squeeze (Reshape)               (None, 49, 39)       0           BN2[0][0]
__________________________________________________________________________________________________
LSTM_Sequences (LSTM)           (None, 49, 64)       26624       Squeeze[0][0]
__________________________________________________________________________________________________
FinalSequence (Lambda)          (None, 64)           0           LSTM_Sequences[0][0]
__________________________________________________________________________________________________
UnitImportance (Dense)          (None, 64)           4160        FinalSequence[0][0]
__________________________________________________________________________________________________
AttentionScores (Dot)           (None, 49)           0           UnitImportance[0][0]
                                                                 LSTM_Sequences[0][0]
__________________________________________________________________________________________________
AttentionSoftmax (Softmax)      (None, 49)           0           AttentionScores[0][0]
__________________________________________________________________________________________________
AttentionVector (Dot)           (None, 64)           0           AttentionSoftmax[0][0]
                                                                 LSTM_Sequences[0][0]
__________________________________________________________________________________________________
FC (Dense)                      (None, 32)           2080        AttentionVector[0][0]
__________________________________________________________________________________________________
Output (Dense)                  (None, 5)            165         FC[0][0]
==================================================================================================
Total params: 33,184
Trainable params: 33,162
Non-trainable params: 22
__________________________________________________________________________________________________
```

Figure 6: Model Architecture

.