Speech Command Model

Pradeep Moturi (es16btech11016)

May 2020

Contents

1 Introduction

This is the report of my work done under Dr. G V V Sharma on building a speech command recognition model for a voice bot. The model used is based on concepts of Convolution, LSTM and Attention and is derived from [1].

2 Create Data

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

Used Audacity to do this.

3 Loading Data

I've used soundfile package to read the .wav. You may choose to use any other package like wavefile, librosa etc to do the same job.
As this is one of the slowest part, I've stored the loaded data as a numpy file for ease and speed of access. Now, I can load the data from npy file if repeating the experiment.

4 Split dataset

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

5 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

6 Feature Extraction

MFCCs are most prominent features used in audio processing. 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_kernel=False, name='mel_stft')

Arguments to the layer
padding: Padding when convoluting
sr: Sampling rate of audio provided
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 trainable
trainable_kernel: If the kernel is trainable

7 Building Model

7.1 Concept

1. Using Convolutional layers ahead of LSTM is shown to improve performance in several research papers.
2. BatchNormalization layers are added to improve convergence rate.
3. Using Bidirectional LSTM is optimal when complete input is available. But this increases the runtime two-fold.
4. Final output sequence of LSTM layer is used to calculate importance of units in LSTM using a FC layer.
5. Then take the dot product of unit importance and output sequences of LSTM to get Attention scores of each time step.
6. Take the dot product of Attention scores and the output sequences of LSTM to get attention vector.
7. Add an additional FC Layer and then to output Layer with SoftMax Activation.

7.2 Hyper 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 15 is used

7.3 Notations

Operators:

- $\times$ indicate matrix multiplication
- $*$ denote convolution (0 padding to same size)
- . denote dot product
- +,- can expand dimensions of their arguments

Format:
Layer Name (Layer Type) (Output Size).
Parameters
Equations
Output
$= equation$

Layer name indicates output of the corresponding layer.
Let us understand the maths behind the model.

7.4 Math

You can have a overall look at the architecture of the model in Fig [6]. Lets observe the math in each layer below.

0. Input (InputLayer) (49, 39, 1)
1. Conv1 (Conv2D) (49, 39, 10)
   Parameters:
   Kernel = (5, 1, 1, 10), Bias = (10)
   Conv1[:,:,i]
   $= Kernel[:,:,:,i] * Input + Bias[i]$
2. BN1 (BatchNormalization) (49, 39, 10)
   Parameters:
   Trainable: γ = (10), β = (10),
   Non-Trainable: Mean = (10), Std = (10)
   Equations:
   Mean[i] $= mean(Conv1[:,:,i])$
   Std[i] $= std(Conv1[:,:,i])$
   BN1[i]
   $= (Conv1[:,:,i] - Mean[i])\frac{\gamma[i]}{Std[i]} + \beta[i]$
3. Conv2 (Conv2D) (49, 39, 1)
   Parameters:
   Kernel = (5, 1, 10, 1), Bias = (1)
   Conv2[:,:,1]
   $= Kernel[:,:,:,1] * BN1 + Bias$
4. BN2 (BatchNormalization) (49, 39, 1)
   Parameters:
   Trainable: γ = (1), β = (1),
   Non-Trainable: Mean = (1), Std = (1)
   Equations:
   Mean[i] $= mean(Conv2[:,:,i])$
   Std[i] $= std(Conv2[:,:,i])$
   BN2[i]
   $= (Conv2[:,:,i] - Mean[i])\frac{\gamma[i]}{Std[i]} + \beta[i]$
5. Squeeze (Reshape) (49, 39)
   Squeeze
   $= BN2.reshape(49, 39)$
6. LSTM_Sequences (LSTM) (49, 64)
   Parameters:
   $U^i = U^f = U^o = U^g = (39, 64)$,
   $W^i = W^f = W^o = W^g = (64, 64)$,
   $B^i = B^f = B^o = B^g = (64)$
   Equations:
   $i_t = \sigma(Squeeze[:,t] \times U^i + h_{t-1} \times W^i + B^i)$
   $f_t = \sigma(Squeeze[:,t] \times U^f + h_{t-1} \times W^f + B^f)$

$o_t = \sigma(Squeeze[:,t] \times U^o + h_{t-1} \times W^o + B^o)$
$\widetilde{C}_t = tanh(Squeeze[:,t] \times U^g + h_{t-1} \times W^g + B^g)$
$C_t = \sigma(f_t * C_{t-1} + i_t * \widetilde{C}_t)$
$h_t = tanh(C_t) * o_t$
LSTM_Sequences[t]
$= h_t$

7. FinalSequence (Lambda) (64)
FinalSequence
$= LSTM\_Sequences[-1,:]$

8. UnitImportance (Dense) (64)
Parameters:
Weights = (64,64), Bias = (64)
UnitImportance
$= Weights \times FinalSequence + Bias$

9. AttentionScores (Dot) (49)
AttentionScores[i]
$= UnitImportance.LSTM\_Sequences[i,:]$

10. AttentionSoftmax (Softmax) (49)
AttentionSoftmax[i]
$= \frac{exp(AttentionScores[i])}{\sum_j exp(AttentionScores[j])}$

11. AttentionVector (Dot) (64)
AttentionVector[i]
$= AttentionSoftmax.LSTM\_Sequences[:,i]$

12. FC (Dense) (32)
Parameters:
Weights = (64,64), Bias = (64)
FC
$= Weights \times AttentionVector + Bias$

13. Output (Dense) (5)
Parameters:
Weights = (32,5), Bias = (5)
Output
$= SoftMax(Weights \times FC + Bias)$

After an input passes through the layers, the training happens by principle of Back Propagating Loss or Gradients calculated by Sparse Categorical Cross-Entropy and updating weights using the Adam Optimizer update equations.

8 Training

8.1 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. This algorithm uses the property of differentiation called chain rule. This algorithm is based on dynamic programming hence removes any redundant calculations of intermediate gradients.
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.

8.2 Adam Optimizer

Back propagation calculates the gradient of loss w.r.t each parameter. But a adaptive learning rate algorithm will be used to update the weights in a more calculated manner.
Unlike SGD, 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: α: Stepsize
Require: $\beta_1, \beta_2 \in [0,1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_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 θ_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 θ_t (Resulting parameters)

Figure 1: Adam algorithm

.

9 Testing

1. Augment the test set same as training set.
2. Extract MFCCs using same method as training set
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.

10 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. Now 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.

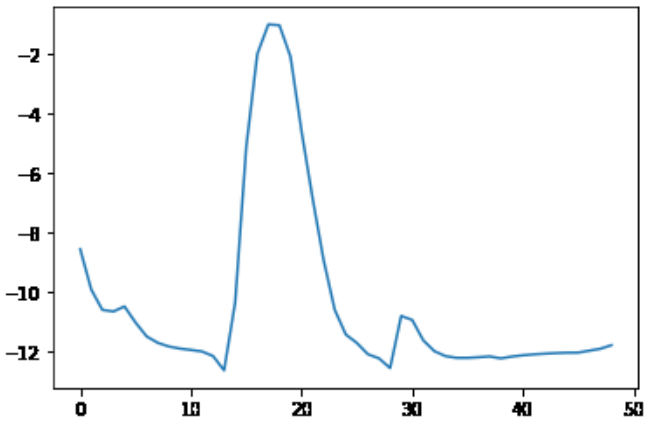

Figure 2: log(Attention Scores)

.

11 Observations

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

12 Files

- Src/DataGenerator.py: Augments the data
- Src/FeatureExtractor.py: Extracts MFCC coefficients

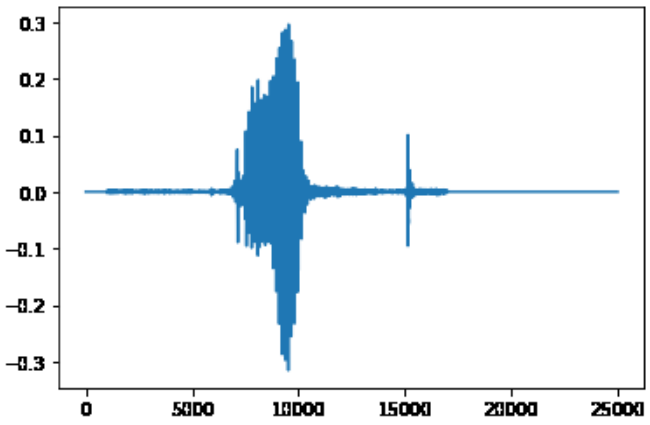

Figure 3: Raw sample

.

- Src/TrainModel.py: Trains model and saves it in h5 file
- ColabNotebook.ipynb: Use this for experimental purpose

13 Further

- Different augmentation techniques like adding noise, changing pitch, speed etc.
- Given the low complexity of our dataset, We can replace LSTM with GRU which is little less complex but shown to outperform LSTM in many scenarios.
- We can tune the arguments to Melspectrogram
- Changing the model architecture like layers and units in layers.
- Further the scope of project to check performance on Google's Speech Command Datasets (v1 and v2) and participate in Kaggle challenge by google [https://www.kaggle.com/c/tensorflow-speech-recognition-challenge/]

References

[1] Douglas Coimbra de Andrade, Sabato Leo, Martin Loesener Da Silva Viana, Christoph Bernkopf. *A neural attention model for speech command recognition.*

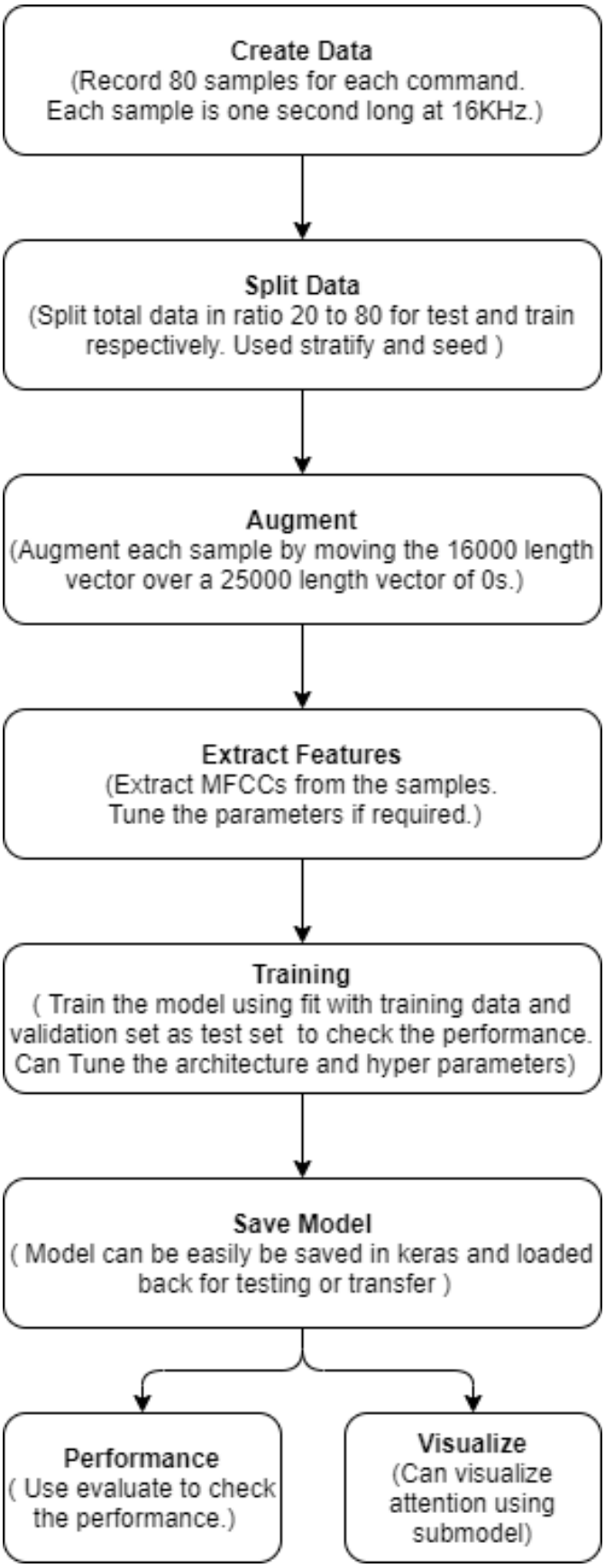

Figure 4: Data Flow Diagram

.

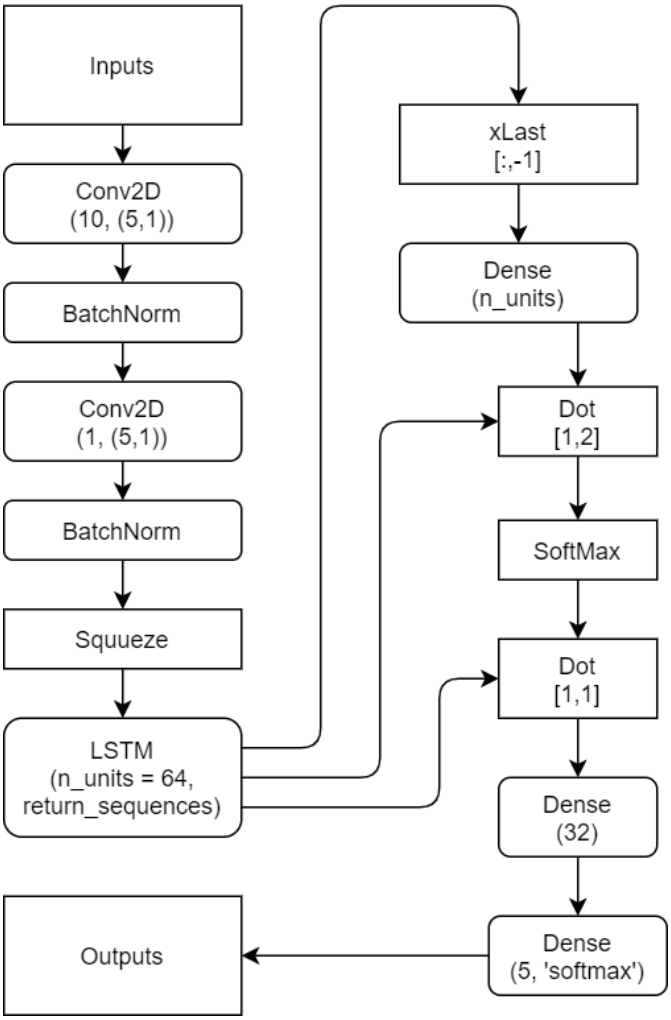

Figure 5: Model Diagram

.

```
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

.