Project by Uddeshya Sinha, Shivank Mishra and Tanmay Dixit

ABSTRACT

Speech recognition has long been a task deemed to be require huge data and dictionary, and even then, working on it was considered to be an end of someone's career. In the past few years however, with the advent of GPUs, fast parallel processing and the availability of huge amounts of open-sourced, processable data, the use of neural networks has grown in pretty much every aspect of information technology, generally.

Our project is the practical implementation of two neural networks — one, a basic (vanilla) seven-layered neural network and two, a recurrent neural network. We have used the same training and testing data for both the networks to maintain uniformity through models and compare results on the same scale. These models are trained to do *speech recognition*, i.e., recognize one-word speech files spoken by thousands of different persons and collected by Google. The models are implemented practically and tested vigorously.

INDEX

Introduction

1. **Scope of the work:** The models that we have created for this project have the potential to be implemented in larger scales. This can definitely be done for the recurrent model, since it gives significantly more stable results for training and testing scenarios alike.

2. **Product Scenarios:** We have tried to deploy both the models on various platforms like Google Cloud Engine and FloydHub, which are both cloud based processing platforms. Even if the dataset increases exponentially, our model will be able to classify all labels at a decent accuracy, considering the fact that it is only a small model, devoid of any intense pre-processing and accumulation or feature extraction of data. We employed only one feature in our one model and four, in the other, to make them comparable.

Requirement Analysis

1. **Functional Requirements:**

   - Rename downloaded files as their word labels.
   - Separate collection into test and training data (95:5).
   - Visualize the audio files as spectrograms and Mel-frequency cepstrum coefficients so that we can obtain an idea of the audio's frequency domain.
   - The model uses "weights and bias" framework to create the perfect function and simulate our classification boundaries.
   - The model uses "Adam" optimizer to adjust gradients.
   - The model uses squashing functions— activations, to calculate loss.
   - Backpropagate to reduce error and adjust weights as much as possible.
   - Use softmax regression to classify labels.
   - Print "Accuracy vs Epoch" graphs for easy comparison between models.

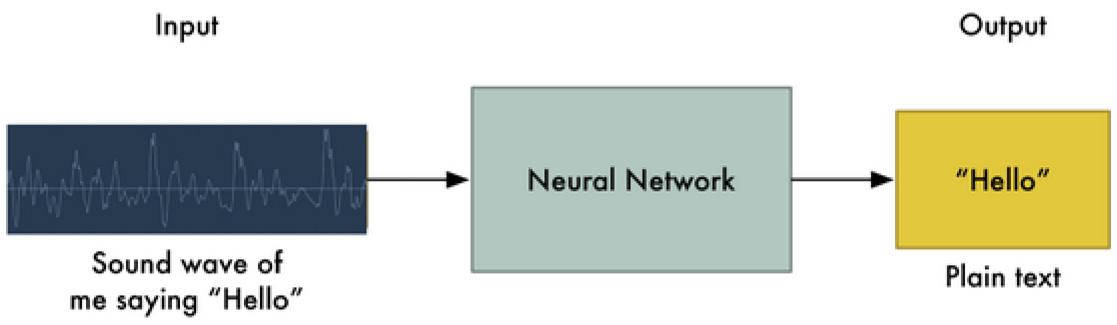
2. **Non-Functional Requirements:**

   - *Performance:* We calculate accuracy on the basis of comparison between the output and the ground truth label by minimizing the loss after every epoch.
   - *Utilization:* The model uses CPU processing power and over the course of numerous iterations reduces the errors obtained and gets closer and closer to the convergence.

- *Response:* The RNN model responds as expected and continues to evaluate both test and training data at similarly high accuracy.
- *Reliability:* The model uses LSTMs and Bidirectional RNNs to provide a stable output without any significant variations.
- *Scalability:* Currently we are working to classify thirty words, but our model can be utilized to efficiently classify upto 300 words with only a standardised decrease in accuracy.
- *Efficiency:* The LSTM model reached a 90/85 % accuracy on training/test data.
- *Portability:* The model can work on any system on even a distributed system, provided it has the environment installed on it.
- *Usability:* The results can be used to correctly choose our choice of model for upscaling towards bigger and better speech recognition systems.

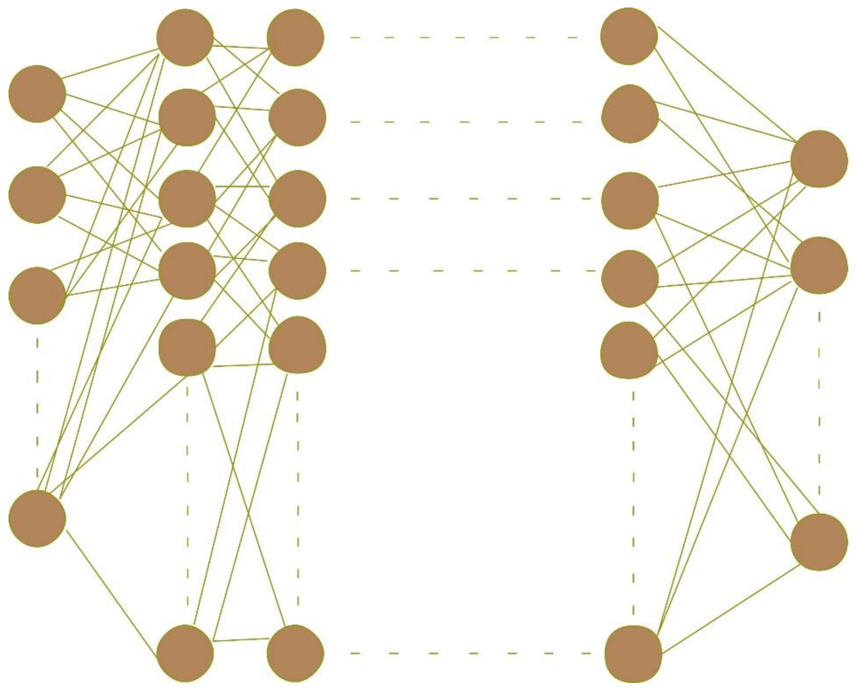
System Design

The high-level flow of the development is as show—



However, we cannot feed the sound wave directly into the network, as we know that any neural network may only accept data in the form of multidimensional vectors, but these dimensions must be programmed beforehand.

7 — Layer Vanilla Neural Network



This neural network uses —

1. **Input layer**, containing 173 different neurons, each a significant part of one of the four extracted audio features—
    - **MFCCs:** Mel-Frequency Cepstral Coefficients enable us to understand the complex sounds generated by a human, which is determined by various factors. MFCCs help us represent this bias in a mathematical fashion.
    - **Chroma:** Chroma is a powerful feature for analysing sound whose pitches can be meaningfully categorised into 12 classes.
    - **Mel:** This feature is also closely related to Mel-log bias imposition.
    - **Contrast:** Minimises the differences in how the same sound is perceived by different listeners.

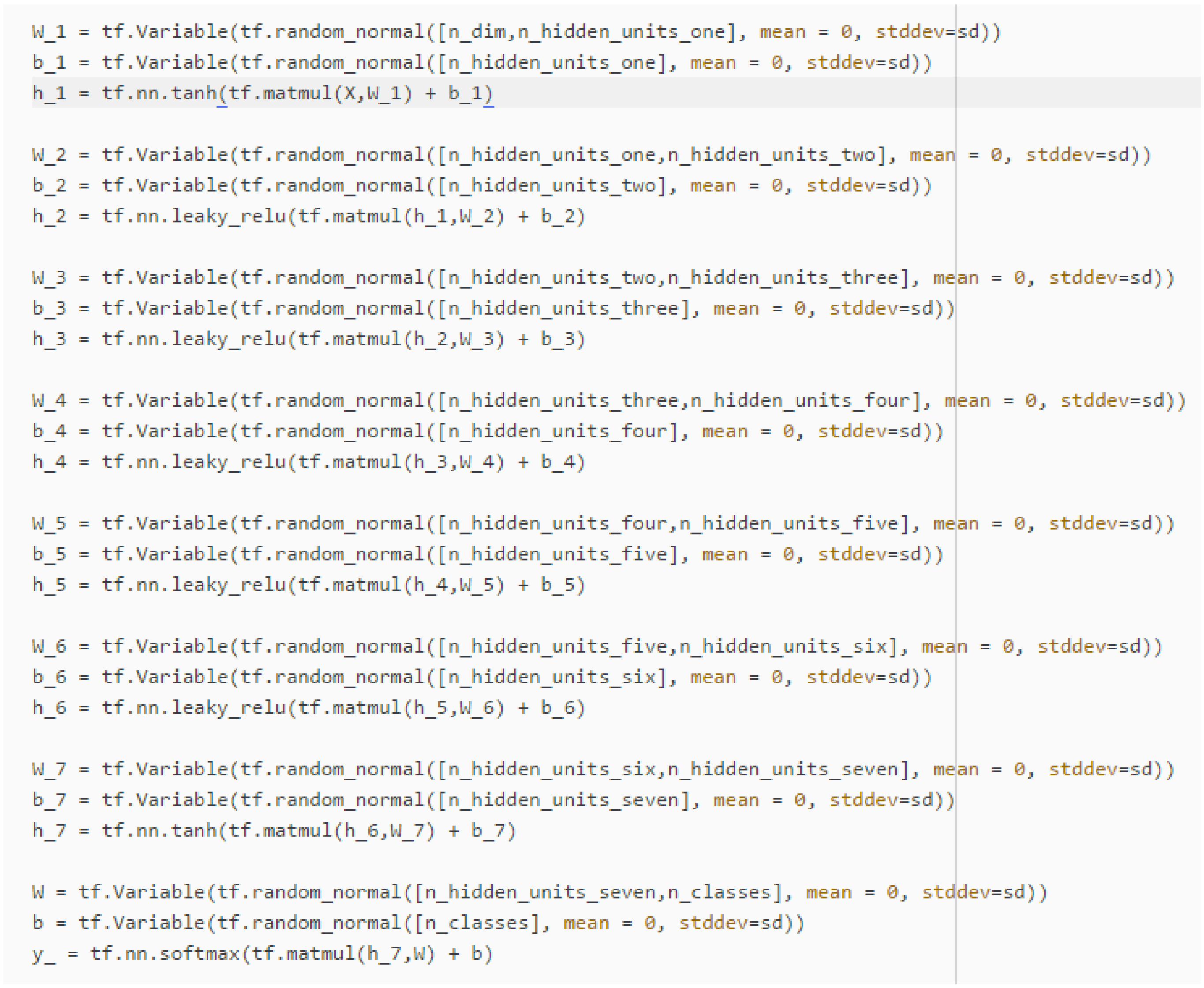
2. After extracting the features in a spectrogram, the network learns progressively to recognize label patterns in it and for this we use **seven hidden layers** to compute the loss and update weights and biases using backpropagation.

    We used *"leaky ReLU"* and *"tanh"* activation functions in the model. The first and last layers were *"tanh"* and the remaining were fixed with *"leaky ReLU".*

3. The **output layer** is a 30-unit layer for the classification of thirty labels and uses the Tensorflow softmax cross-entropy to obtain the probabilities for each class. After this, the layer chooses the one with the most probability as the predicted output.

4. The optimizer used for this network is **Nadam Optimizer**, which is the **Adam Optimizer**, coupled with **Nesterov Momentum.** The optimizer uses a learning rate which is an algorithm that modifies the parameters of the network in order for a given input to the network to produce a favoured output.

5. **Backpropagation** is the algorithm which, when combined with the learning rate, adjusts weights and biases to reduce loss and minimize error.



```
W_1 = tf.Variable(tf.random_normal([n_dim,n_hidden_units_one], mean = 0, stddev=sd))
b_1 = tf.Variable(tf.random_normal([n_hidden_units_one], mean = 0, stddev=sd))
h_1 = tf.nn.tanh(tf.matmul(X,W_1) + b_1)

W_2 = tf.Variable(tf.random_normal([n_hidden_units_one,n_hidden_units_two], mean = 0, stddev=sd))
b_2 = tf.Variable(tf.random_normal([n_hidden_units_two], mean = 0, stddev=sd))
h_2 = tf.nn.leaky_relu(tf.matmul(h_1,W_2) + b_2)

W_3 = tf.Variable(tf.random_normal([n_hidden_units_two,n_hidden_units_three], mean = 0, stddev=sd))
b_3 = tf.Variable(tf.random_normal([n_hidden_units_three], mean = 0, stddev=sd))
h_3 = tf.nn.leaky_relu(tf.matmul(h_2,W_3) + b_3)

W_4 = tf.Variable(tf.random_normal([n_hidden_units_three,n_hidden_units_four], mean = 0, stddev=sd))
b_4 = tf.Variable(tf.random_normal([n_hidden_units_four], mean = 0, stddev=sd))
h_4 = tf.nn.leaky_relu(tf.matmul(h_3,W_4) + b_4)

W_5 = tf.Variable(tf.random_normal([n_hidden_units_four,n_hidden_units_five], mean = 0, stddev=sd))
b_5 = tf.Variable(tf.random_normal([n_hidden_units_five], mean = 0, stddev=sd))
h_5 = tf.nn.leaky_relu(tf.matmul(h_4,W_5) + b_5)

W_6 = tf.Variable(tf.random_normal([n_hidden_units_five,n_hidden_units_six], mean = 0, stddev=sd))
b_6 = tf.Variable(tf.random_normal([n_hidden_units_six], mean = 0, stddev=sd))
h_6 = tf.nn.leaky_relu(tf.matmul(h_5,W_6) + b_6)

W_7 = tf.Variable(tf.random_normal([n_hidden_units_six,n_hidden_units_seven], mean = 0, stddev=sd))
b_7 = tf.Variable(tf.random_normal([n_hidden_units_seven], mean = 0, stddev=sd))
h_7 = tf.nn.tanh(tf.matmul(h_6,W_7) + b_7)

W = tf.Variable(tf.random_normal([n_hidden_units_seven,n_classes], mean = 0, stddev=sd))
b = tf.Variable(tf.random_normal([n_classes], mean = 0, stddev=sd))
y_ = tf.nn.softmax(tf.matmul(h_7,W) + b)
```

Fig. Code for the Feedforward Neural Network.

Recurrent Neural Network

A Recurrent Neural Network is a class of Artificial Neural Networks where connections between units form a directed cycle. Unlike the Vanilla Neural Network, RNNs can use their internal memory to process arbitrary sequence of inputs.

For memory management, we incorporated LSTMs(Long Short -Term Memory) into the RNN. LSTMs help preserve the error that can be backpropagated through time and layers. By maintaining a more constant error, they allow recurrent nets to continue to learn over many time steps (over 1000), thereby opening a channel to link causes and effects remotely.

LSTMs contain information outside the normal flow of the recurrent network in a gated cell. Information can be stored in, written to, or read from a cell, much like data in a computer's memory. The cell makes decisions about what to store, and when to allow reads, writes and erasures, via gates that open and close. Unlike the digital storage on computers, however, these gates are analog, implemented with element-wise multiplication by sigmoids, which are all in the range of 0–1. Analog has the advantage over digital of being differentiable, and therefore suitable for backpropagation.

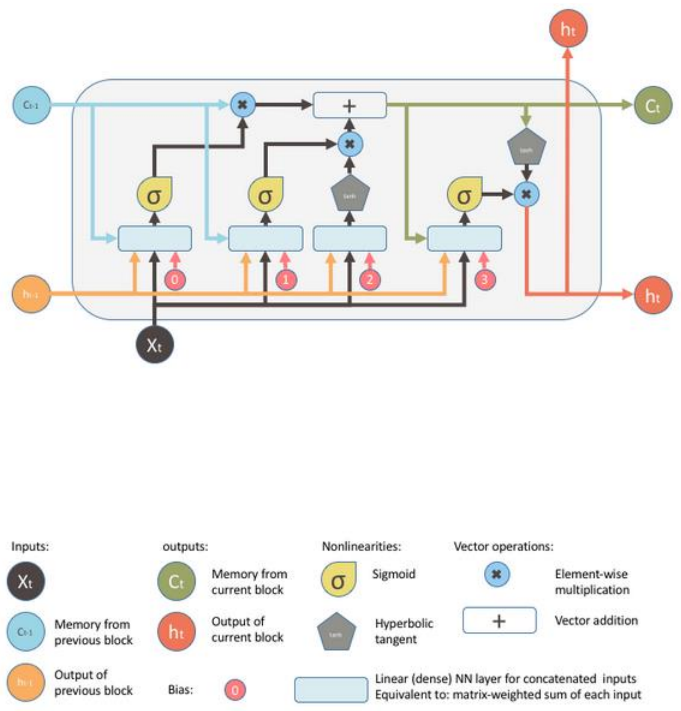
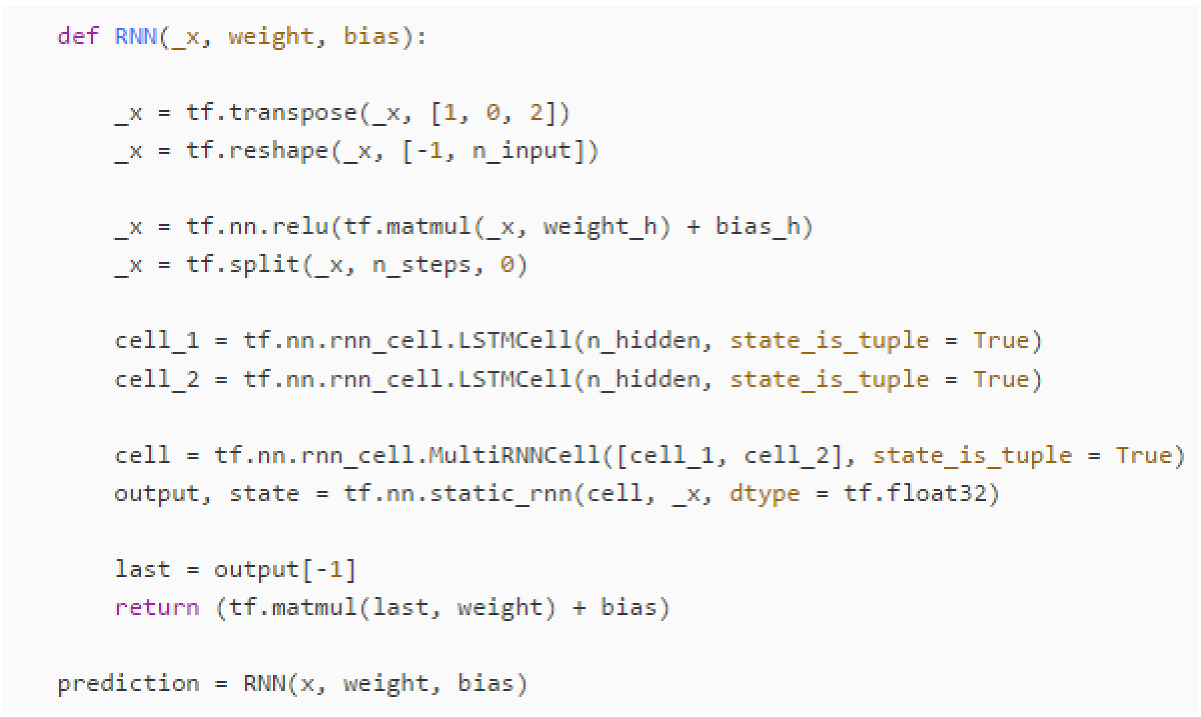

Inputs:
Xt
Ct-1 Memory from previous block
ht-1 Output of previous block
outputs:
Ct Memory from current block
ht Output of current block
Bias:
Nonlinearities:
σ Sigmoid
Hyperbolic tangent
Vector operations:
Element-wise multiplication
Vector addition
Linear (dense) NN layer for concatenated inputs
Equivalent to: matrix-weighted sum of each input

Fig. LSTM Neuron.

Our model contains the following layers—

1. **Input Layer:** It contains 26 units, one each for the features extracted. We are using MFCCs only this model, since we have distributed these 26 features across 32 timesteps for each file.
2. **Hidden layer:** We have used one ReLU activated hidden layer that serves the purpose of normalizing out input data for the RNN.
3. **LSTM layer:** We have used two hidden LSTM cells with 100 units each for the long-term learning of the model.
4. **Output layer:** It is a softmax based layer that classifies the words into their respective classes/ predictions.



```
def RNN(_x, weight, bias):

    _x = tf.transpose(_x, [1, 0, 2])
    _x = tf.reshape(_x, [-1, n_input])

    _x = tf.nn.relu(tf.matmul(_x, weight_h) + bias_h)
    _x = tf.split(_x, n_steps, 0)

    cell_1 = tf.nn.rnn_cell.LSTMCell(n_hidden, state_is_tuple = True)
    cell_2 = tf.nn.rnn_cell.LSTMCell(n_hidden, state_is_tuple = True)

    cell = tf.nn.rnn_cell.MultiRNNCell([cell_1, cell_2], state_is_tuple = True)
    output, state = tf.nn.static_rnn(cell, _x, dtype = tf.float32)

    last = output[-1]
    return (tf.matmul(last, weight) + bias)

prediction = RNN(x, weight, bias)
```

Fig. Code for the RNN.

Work Done

Development Environment

- **Python (3.5.4)** : Python is a widely used high-level programming language, which is most preferable for deep learning due to its simple and easy to remember syntax.
- **Tensorflow** : Tensorflow is a library for dataflow programming which we have used as a backend.
- **Matplotlib** : We use Matplotlib for the plotting of various graphs.

- **Librosa :** It is a python package for audio analysis that we use to analyse the input words.
- **Scipy :** A package in python which we used to perform multiple complex algebraic computations.
- **Numpy :** The library in python which we use to deal with arrays, 2d, 3d matrices etc.
- **Anaconda (4.4) :** It is a distribution of python and R, which comes equipped with a wide array of core packages integral for data science.
- **FloydHub :** It is a deep learning platform which we tried to implement (allows us to use the cloud computations which our laptop isn’t capable of), but only succeeded partially.

Results

Neural Network Type	Train Accuracy(%)	Test Accuracy(%)
Feed Forward ANN	99.8	26.8
Recurrent NN	90	85

Observations

- Feed Forward Network achieves a near- perfect accuracy on the training set but it falters on the training set and we can easily observe that this is because of overfitting of data on the training set. It is not generalizing well on the data and hence is not a good model for Speech Recognition as it cannot be scaled.

- We tested two models as Feed Forward ANN, on different combinations of learning rates, optimizers, hidden layer number, and hidden units, before finally choosing the 7-layer one. We observed that the 11-layer model was much too slow to compute the accuracies obtained by the 7-layer one.

- Recurrent Neural Network succeeds in achieving a decent accuracy on both the training and test data.

- We realise that Recurrent Neural Networks are the way to move forward in Speech Recognition as they retain data from previous time-steps and are hence great at recognizing patterns across temporal and spatial differences.

Conclusion & Future Work

We were able to correctly classify the thirty audio files into their respective labels on the basis of the words spoken in them, using both a **Feed Forward Artificial Neural Network** and a **Recurrent Neural Network**, with the former being rejected because of its low generalization behaviour on the training data, as proved by the extremely high accuracy on it, but an equally embarrassing low on the test set, leading us to accept the RNN model which gave much more promising results on both training and test sets. The greater accuracy on the RNN motivated us to experiment with different optimizers like *Nadam, Adam, RMSProp*— with the results being inclined greatly towards *Adam.*

Working within these constraints, we realised we could not improve our models to scale. But, achieving such a high accuracy on Google's dataset has given us the confidence we need to tackle such a deep learning problem on a much larger scale, head on. We intend to pursue our careers in this field as it has been a rewarding journey through the volatile landscape of neural networks.