Music Generation using RBM and TensorFlow

Shruti Mehta | Pranav Swaminathan
mehta.shru@husky.neu.edu, swaminathan.p@husky.neu.edu
CSYE 7245, INFO 7390, Fall 2017, Northeastern University

Abstract

Automatic music composition is not a new area of research, the first technique was to create several rules for the computer to follow. This technique can give some appreciable results, but music is more complex than a series of strict rules. A lot of people think that composing music can only be achieved by humans and that a machine is not intelligent enough to create pleasing music. This may be true for now, but it may be that there is still some information about musical structure and repetition that can be learned from all the existing examples by fairly simple models and that can be used to compose more melodic songs.

Introduction

This project involves the generation of music by training the **Restricted Boltzmann Machine** model using TensorFlow on a set of homogeneous instrumental music to produce new music.
Applying machine learning techniques in the domain of music generation is an interesting subject that hasn't been studied in depth. This algorithm will be able to model the melodic repetition that is found in almost all kinds of songs. In this project, we are generating new music files by applying the algorithm on the music files as input data.

Motivation

The datasets in real life are much more complex. We have to first understand it, collect it from various sources and arrange it in a format which is ready for processing. This is even more difficult when the data is in an unstructured format such as image or audio. This is so because you would have to represent image/audio data in a standard way for it to be useful for analysis.
Having inspired by music and having decent understanding of the music theory, we were motivated to work on this project. Music theory (sheets) – notes are related to each other. These notes are just ways to mapping out pitches or sound waves. The common problem to tackle is to fit the model or make the model learn in such a way that it recognizes the relation between notes.

Music sheets and notes

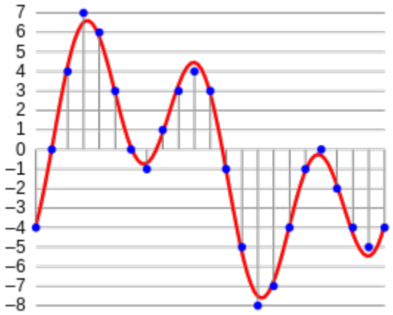
Audio data

Directly or indirectly, we are always in contact with audio. Your brain is continuously processing and understanding audio data and giving you information about the environment. A simple example can be your conversations with people which you do daily. This speech is discerned by the other person to carry on the discussions. Even when you think you are in a quiet environment, you tend to catch much more subtle sounds, like the rustling of leaves or the splatter of rain. This is the extent of your connection with audio.

There are devices built which help you catch these sounds and represent it in computer readable format. Examples of these formats are

- **wav (Waveform Audio File) format**
- **mp3 (MPEG-1 Audio Layer 3) format**
- **WMA (Windows Media Audio) format**
- **MIDI(Musical Instrument Digital Interface)**

If you give a thought on what an audio looks like, it is nothing but a wave like format of data, where the amplitude of audio change with respect to time. This can be pictorial represented as follows.



An audio signal

Applications of Audio Processing

- Indexing music collections according to their audio features
- Recommending music for radio channels
- Similarity search for audio files
- Speech processing and synthesis – generating artificial voice for conversational agents

Dataset

The training data is around 100 MIDI files of pop songs (MIDI is a format that directly encodes musical notes). To keep it simple, we won't label them with emotions, our output will be another pop melody, just like our training files.

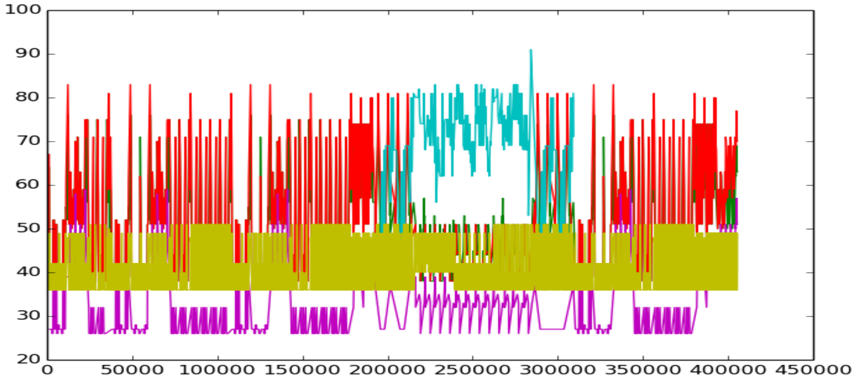
MIDI (Musical Instrument Digital Interface)

MIDI (Musical Instrument Digital Interface) is a protocol designed for recording and playing back music on digital synthesizers that is supported by many makes of personal computer sound cards. Originally intended to control one keyboard from another, it was quickly adopted for the personal computer. Rather than representing musical sound directly, it transmits information about how music is produced. The command set includes note-ons, note-offs, key velocity, pitch bend and other methods of controlling a synthesizer. The sound waves produced are those already stored in a wavetable in the receiving instrument or sound card.

Perhaps the best way to understand what MIDI is to first understand what it is not:

- MIDI isn't music
- MIDI doesn't contain any actual sounds
- MIDI isn't a digital music file format like MP3 or WAV

MIDI is nothing more than data -- a set of instructions. MIDI data contains a list of **events** or **messages** that tell an electronic device (musical instrument, computer sound card, cell phone, et cetera) how to generate a certain sound.



MIDI signal

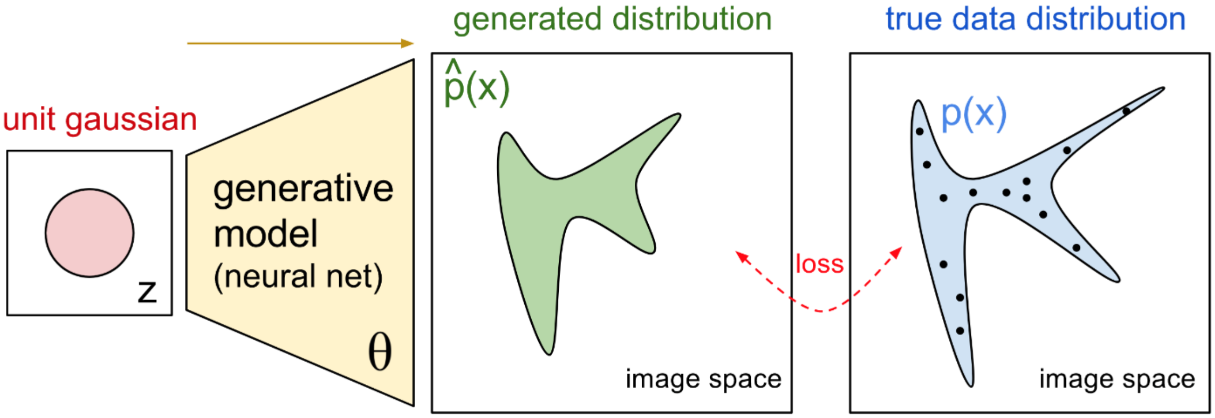
Technical Approach

1. Generative Models

It is a model that allows us to learn simulator of data. They are the models that allow for (conditional) density estimation. It is an approach for unsupervised learning of data.

Generative models are used in machine learning for either modeling data directly (i.e., modeling observations drawn from a probability density function), or as an intermediate step to forming a conditional probability density function. Generative models are typically probabilistic, specifying a joint probability distribution over observation and target (label) values. A conditional distribution can be formed from a generative model through Bayes' rule.

Generative Models specify a probability distribution over a dataset of input vectors. For an unsupervised task, we form a model for P(x)P(x), where xx is an input vector. For a supervised task, we form a model for P(x|y)P(x|y), where yy is the label for xx. Like discriminative models, most generative models can be used for classification tasks. To perform classification with a generative model, we leverage the fact that if we know P(X|Y)P(X|Y) and P(Y)P(Y), we can use bayes rule to estimate P(Y|X)P(Y|X).

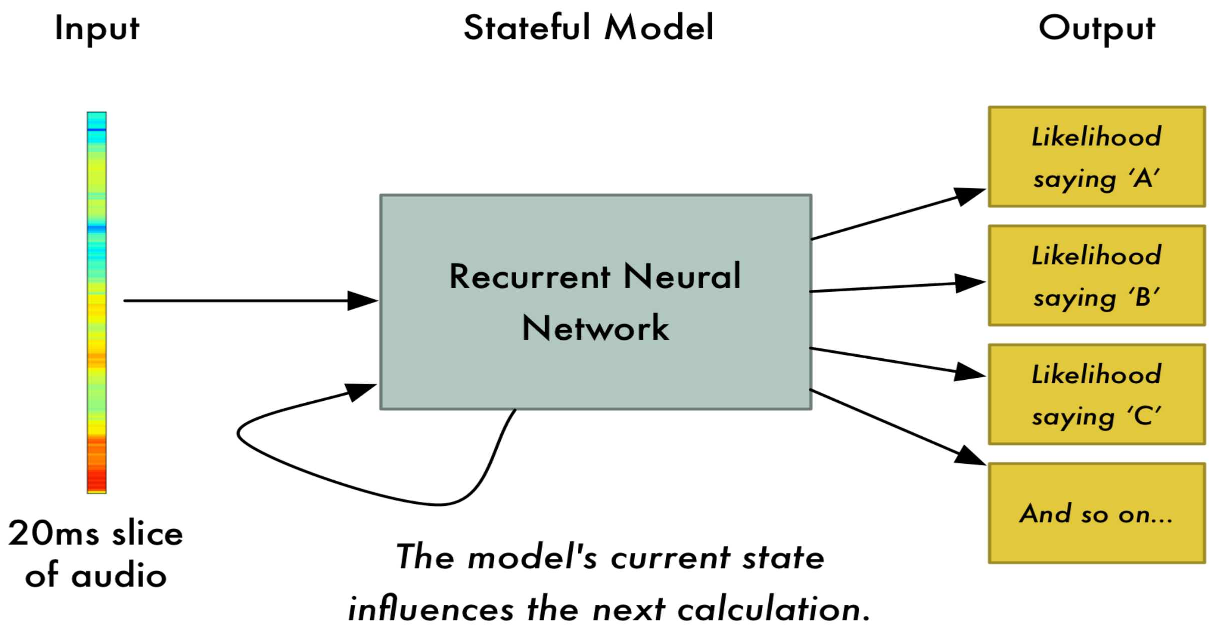


Since, we are going to use a generative model, we can generate new samples directly by sampling from the modelled probability distribution.

Thinking about Generating new music files, we thought of trying Generative Adversarial Networks as a part of music generation process. We started working on GAN with our dataset.

2. Sound from recurrent neural networks

Audio generation seems like a natural application of recurrent neural networks, which are currently very popular (and effective) models for sequence-to-sequence learning. They have been used in speech recognition and speech synthesis



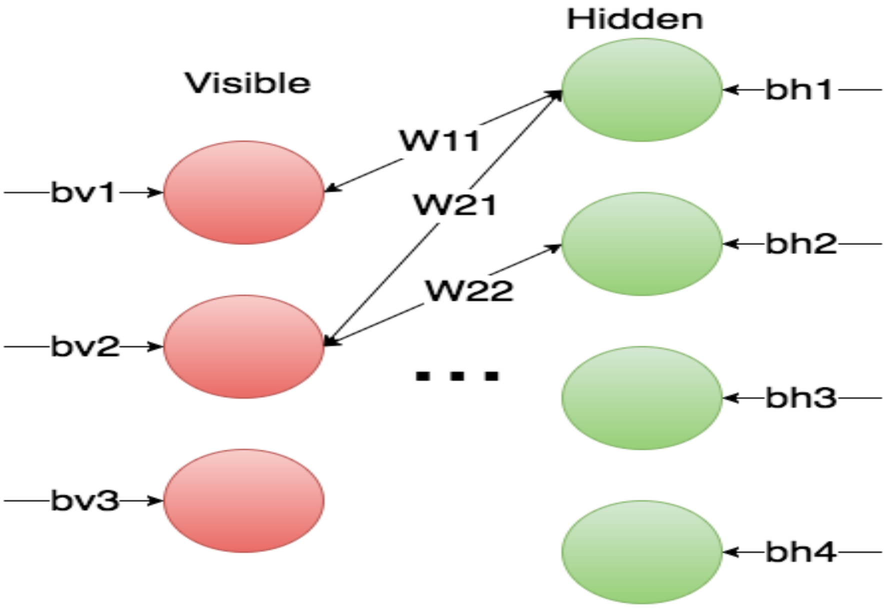
3. Restricted Boltzman Machine(RBM)

A restricted Boltzmann machine (RBM) is a generative stochastic artificial neural network that can learn a probability distribution over its set of inputs.

Architecture

The RBM is a neural network with 2 layers, the visible layer and the hidden layer. Each visible node is connected to each hidden node (and vice versa), but there are no visible-visible or hidden-hidden connections (the RBM is a complete bipartite graph). Since there are only 2 layers, we can fully describe a trained RBM with 3 parameters:

- The weight matrix WW:
  - WW has size nvisible x nhiddennvisible x nhidden. WijWij is the weight of the connection between visible node ii and hidden node jj.
- The bias vector bvbv:
  - bvbv is a vector with nvisiblenvisible elements. Element ii is the bias for the ithith visible node.
- The bias vector bh:
  - bhbh is a vector with n_hidden elements. Element jj is the bias for the jthjth hidden node.
  - nvisible nvisible is the number of features in the input vectors. Nhidden nhidden is the size of the hidden layer. Each visible node takes one chord. Each chord is multiplied by a weight and then the node's output at the hidden layer. Unlike mostof the neural networks, RBMs are generative models that directly model the probability distribution of data.



RBM network

Challenges

GAN (Generative Adversarial Networks)

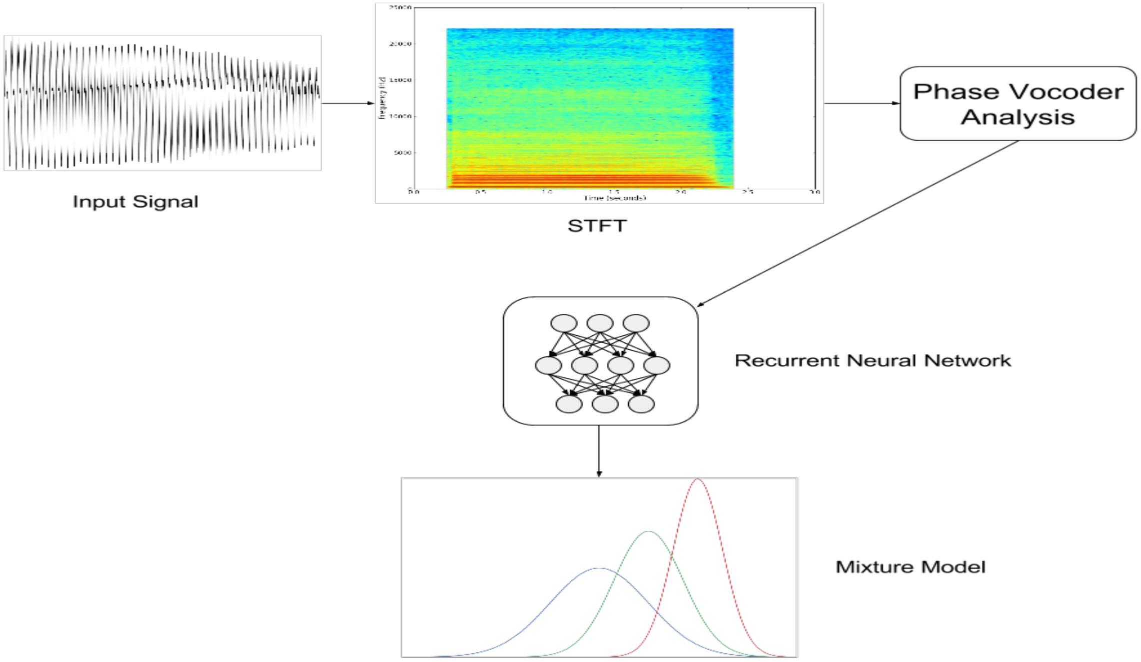
Generative adversarial networks (GANs) provide a way to learn deep representations without extensively annotated training data. They achieve this through deriving backpropagation signals through a competitive process involving a pair of networks. The representations that can be learned by GANs may be used in a variety of applications, including image synthesis, semantic image editing, style transfer, image super-resolution and classification.

Limitations with GAN

The big disadvantage is that these networks are very hard to train. The function of these networks try to optimize is a loss function that essentially has no closed form (unlike standard loss functions like log-loss or squared error). Thus, optimizing this loss function is very hard and requires a lot of trial-and-error regarding the network structure and training protocol. Since RNNs are generally more fickle than CNNs, this is likely why very few (if any) people have been able to apply GANs to anything more complex than images, such as text or speech.

Excellent knowledge about the arguments we are passing through the model. Python version 3 doesn't have many libraries for implementing GAN.

So, we decided to apply Restricted Boltzmann Machine using RNN.



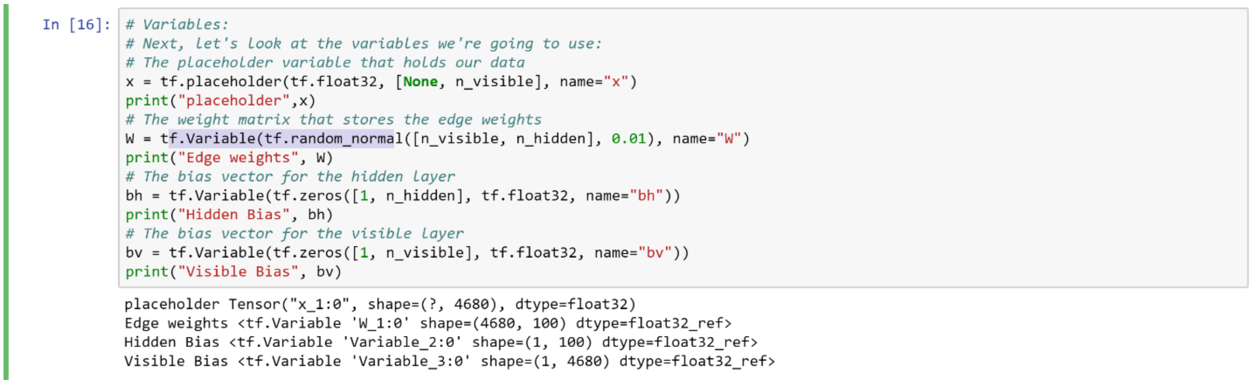
Python Library

Tensorflow:

TensorFlow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API.

Methodology
Code Implementation

1. Initializing the parameters in the neural network of hidden and visible layer

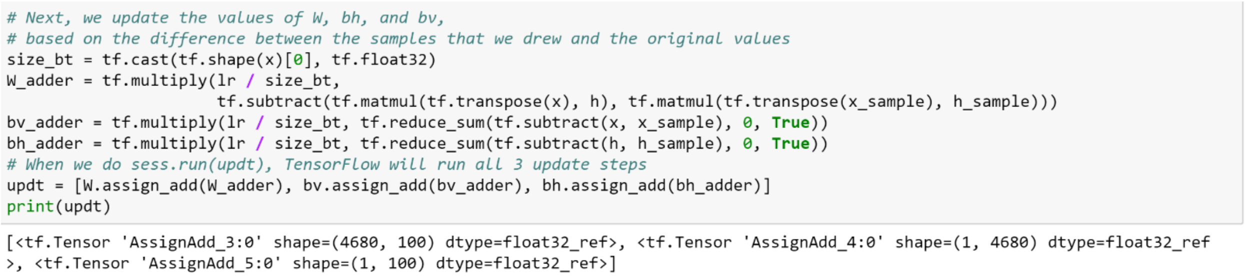


In [16]:

```
# Variables:
# Next, let's look at the variables we're going to use:
# The placeholder variable that holds our data
x = tf.placeholder(tf.float32, [None, n_visible], name="x")
print("placeholder",x)
# The weight matrix that stores the edge weights
W = tf.Variable(tf.random_normal([n_visible, n_hidden], 0.01), name="W")
print("Edge weights", W)
# The bias vector for the hidden layer
bh = tf.Variable(tf.zeros([1, n_hidden], tf.float32, name="bh"))
print("Hidden Bias", bh)
# The bias vector for the visible layer
bv = tf.Variable(tf.zeros([1, n_visible], tf.float32, name="bv"))
print("Visible Bias", bv)
```

```
placeholder Tensor("x_1:0", shape=(?, 4680), dtype=float32)
Edge weights <tf.Variable 'W_1:0' shape=(4680, 100) dtype=float32_ref>
Hidden Bias <tf.Variable 'Variable_2:0' shape=(1, 100) dtype=float32_ref>
Visible Bias <tf.Variable 'Variable_3:0' shape=(1, 4680) dtype=float32_ref>
```

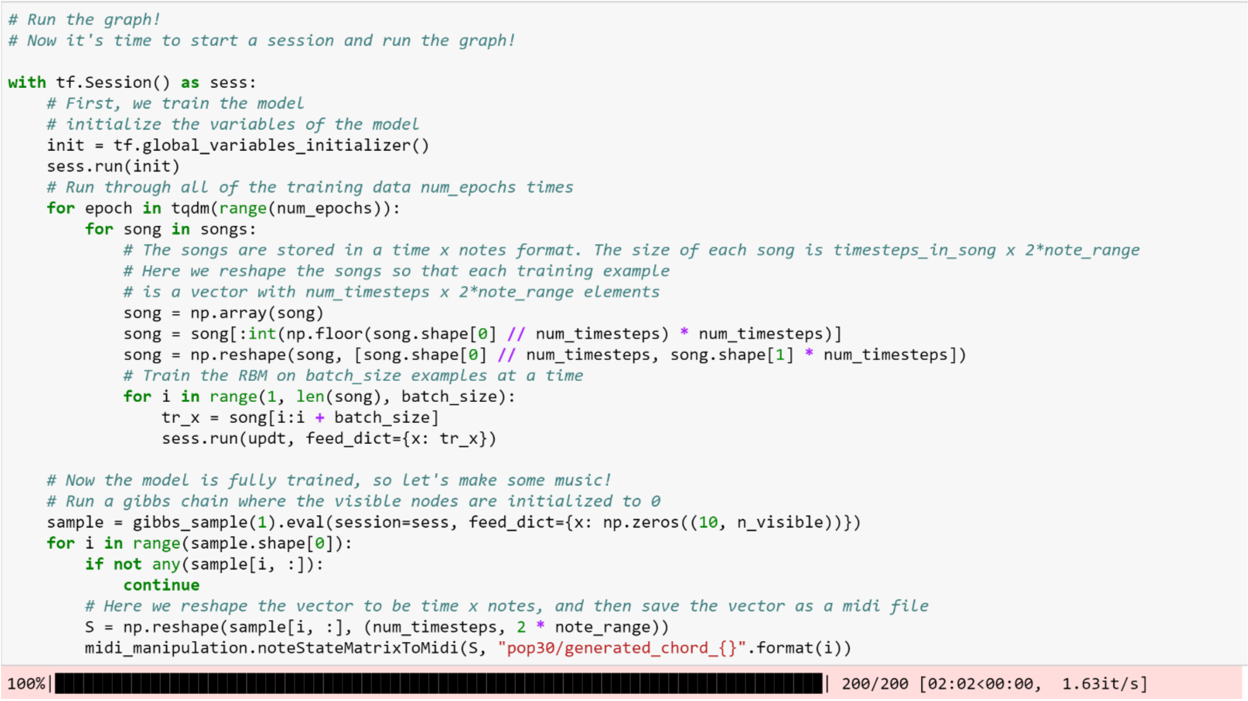
2. **Updating the input parameters based on the difference between the samples that we drew and original values**



```
# Next, we update the values of W, bh, and bv,
# based on the difference between the samples that we drew and the original values
size_bt = tf.cast(tf.shape(x)[0], tf.float32)
W_adder = tf.multiply(lr / size_bt,
                      tf.subtract(tf.matmul(tf.transpose(x), h), tf.matmul(tf.transpose(x_sample), h_sample)))
bv_adder = tf.multiply(lr / size_bt, tf.reduce_sum(tf.subtract(x, x_sample), 0, True))
bh_adder = tf.multiply(lr / size_bt, tf.reduce_sum(tf.subtract(h, h_sample), 0, True))
# When we do sess.run(updt), TensorFlow will run all 3 update steps
updt = [W.assign_add(W_adder), bv.assign_add(bv_adder), bh.assign_add(bh_adder)]
print(updt)
```

```
[<tf.Tensor 'AssignAdd_3:0' shape=(4680, 100) dtype=float32_ref>, <tf.Tensor 'AssignAdd_4:0' shape=(1, 4680) dtype=float32_ref
>, <tf.Tensor 'AssignAdd_5:0' shape=(1, 100) dtype=float32_ref>]
```

3. **Training the model by using each music file (.midi) as input and generating the midi files as output**

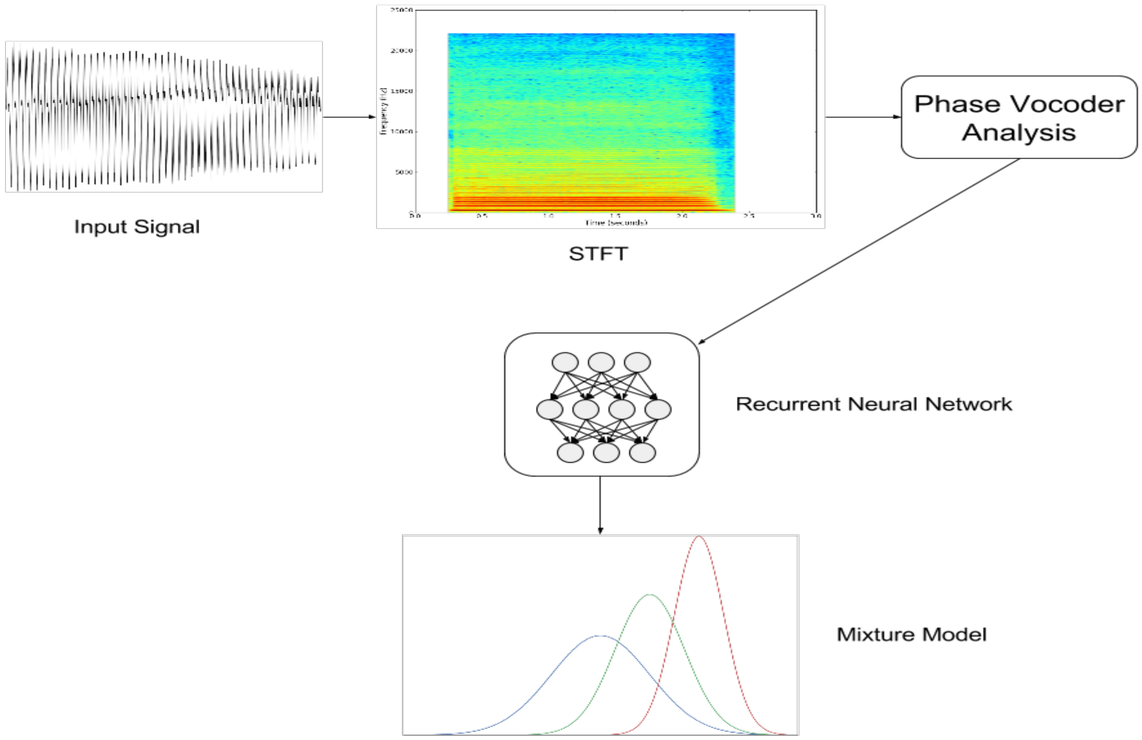


```
# Run the graph!
# Now it's time to start a session and run the graph!

with tf.Session() as sess:
    # First, we train the model
    # initialize the variables of the model
    init = tf.global_variables_initializer()
    sess.run(init)
    # Run through all of the training data num_epochs times
    for epoch in tqdm(range(num_epochs)):
        for song in songs:
            # The songs are stored in a time x notes format. The size of each song is timesteps_in_song x 2*note_range
            # Here we reshape the songs so that each training example
            # is a vector with num_timesteps x 2*note_range elements
            song = np.array(song)
            song = song[:int(np.floor(song.shape[0] // num_timesteps) * num_timesteps)]
            song = np.reshape(song, [song.shape[0] // num_timesteps, song.shape[1] * num_timesteps])
            # Train the RBM on batch_size examples at a time
            for i in range(1, len(song), batch_size):
                tr_x = song[i:i + batch_size]
                sess.run(updt, feed_dict={x: tr_x})

    # Now the model is fully trained, so let's make some music!
    # Run a gibbs chain where the visible nodes are initialized to 0
    sample = gibbs_sample(1).eval(session=sess, feed_dict={x: np.zeros((10, n_visible))})
    for i in range(sample.shape[0]):
        if not any(sample[i, :]):
            continue
        # Here we reshape the vector to be time x notes, and then save the vector as a midi file
        S = np.reshape(sample[i, :], (num_timesteps, 2 * note_range))
        midi_manipulation.noteStateMatrixToMidi(S, "pop30/generated_chord_{}".format(i))
```

```
100%|████████████████████████████████████████████████████████████████████████████████| 200/200 [02:02<00:00,  1.63it/s]
```

Conclusion

In this project, we have generated the audio files after training the model using RBM and tensorflow. The code and the generated audio files can be found at the link below:



Acknowledgement

We would like to acknowledge Professor Nik Bear Brown for his constant motivation and knowledge throughout the course. We would also like to thank all the teaching assistants for helping us with the code implementation.

References

[1] https://medium.com/@oktaybahceci/generate-music-with-tensorflow-midis-4bf928a35c3a
[2]https://www.reddit.com/r/datasets/comments/3akhxy/the_largest_midi_collection_on_the_internet/
[3] https://www.analyticsvidhya.com/blog/2017/08/audio-voice-processing-deep-learning/
[4] https://arxiv.org/pdf/1709.01620.pdf
[5] https://www.youtube.com/watch?v=pg9apmwf7og&t=721s
[6]https://www.reddit.com/r/datasets/comments/3akhxy/the_largest_midi_collection_on_the_internet/