## Harvard Extension School E-63 Big Data Analytics

## 05/10/27

## Final Project Summary

## TensorFlow for Music Generation

## Gerardo Castaneda, Stephen Ford

## Problem Statement

## Build a web app that uses TensorFlow and neural networks to generate music. The web app needs to be usable by anybody, with zero training

## Data Set

## To generate music, we feed TensorFlow a large number of songs encoded using the MIDI format

## Technologies

## Web layer: HTML5, CSS3, Skeleton

## Web server: Apache 2.4.18

## Server: Ubuntu 16.04.2 LTS

## TensorFlow 1.1

## Python 2.7

## Sound synthesizer: Timidity++

## Cloud Hosting AWS

## Benefits

## Creates unlimited songs in very short time . Better than other solutions on the web since use is very simple and songs are generated for different instruments

## Drawbacks

## Subject to the limitations of MIDI. Does not handle human voice, and sounds depend on what is available in the sound synthesizer. Current solution only handles one instrument at a time

## Results

## Live site: http://ec2-34-207-111-44.compute-1.amazonaws.com/tensorweb/index.html

## Click on the icon below to listen to a sample song

## 

## Challenges

## Music manipulation libraries and tools for python have not kept up with the times.

## Video URLs

## Short: <https://youtu.be/-3Ay7-5WHa0>

## Long: <https://youtu.be/QSp_lL30NIg>

**Solution Details**

Our solution is a web app that generates .wav files using recurrent neural networks in TensorFlow. The RNN\_RBM solution is heavily based on Dab Shiebler's [Musical TensorFlow](http://danshiebler.com/2016-08-17-musical-tensorflow-part-two-the-rnn-rbm/) article for generating long sequences of polyphonic music by using an RNN\_RBM in TensorFlow.

The original solution from Dan Shiebler was upgraded to work with Tensorflow 1.1 ( not a trivial task). Other modifications and enhancements include the ability to automatically generate new music, convert the new music from midi to wav, and play the new music online. The solution to convert from midi to wav files uses Timidity++ Synthesizer.

To see a demo of our live solution checkout [TensorWeb Music](http://ec2-34-207-111-44.compute-1.amazonaws.com/tensorweb/index.html)

**High Level Flow**

**The theory behind**

This solution uses a RNN\_RBM model

* Recurrent Neural Network (RNN): a neural network architecture that can handle sequences of vectors (perfect for working with temporal data)
* Restricted Boltzmann Machines (RBM): A neural network with 2 layers, the visible layer and the hidden layer
* RNN\_RBM : A series of RBMs whose parameters are controlled by the RNN

To train the RNN-RBM, we repeat the following procedure:

* Communicate the current state of the song to the RBM
* Create a few musical notes with the RBM
* Calculate the loss and back-propagate to update the network weights and biases
* Use the new information to update the state of the song

The training data set is a series of songs codified using the MIDI protocol. A python utility converts MIDI codes into matrices that Tensor flow can understand

To generate music, we follow these steps

* Initialize an empty array to store our music and initialize RBM with zeros
* Repeat the following
* Communicate the current state of the song to the RBM
* Create a few musical notes with the RBM
* Add to the music array

Note that the dimensions of the music array will change with every step

**Installation of core tensor flow components**

All the code files are hosted in <https://github.com/gc-cloud/cyberbach> You can also copy the code manually from the class website.

The instructions below assume that you have installed anaconda and know how virtual environments work. If you are not familiar with virtual environments it is highly recommended that you familiarize yourself with them first. We also assume you are using Ubuntu 16.04.2 LTS with git and a C compiler preinstalled

1) Clone our github repository

$ git clone git@github.com:gc-cloud/cyberbach.git

2) Move to the cyberbach folder

$ cd cyberbach

3) Create and activate a conda environment with python 2.7

$ conda create -n tf1.1\_py2.7 python=2.7

$ source activate tf1.1\_py2.7

4) Install tensorflow 1.1

$ export TF\_BINARY\_URL=https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-1.1.0-cp27-none-linux\_x86\_64.whl

$ pip install --upgrade $TF\_BINARY\_URL

5) Install dependencies (pandas, tqdm, patplotlib, midi, timidity++)

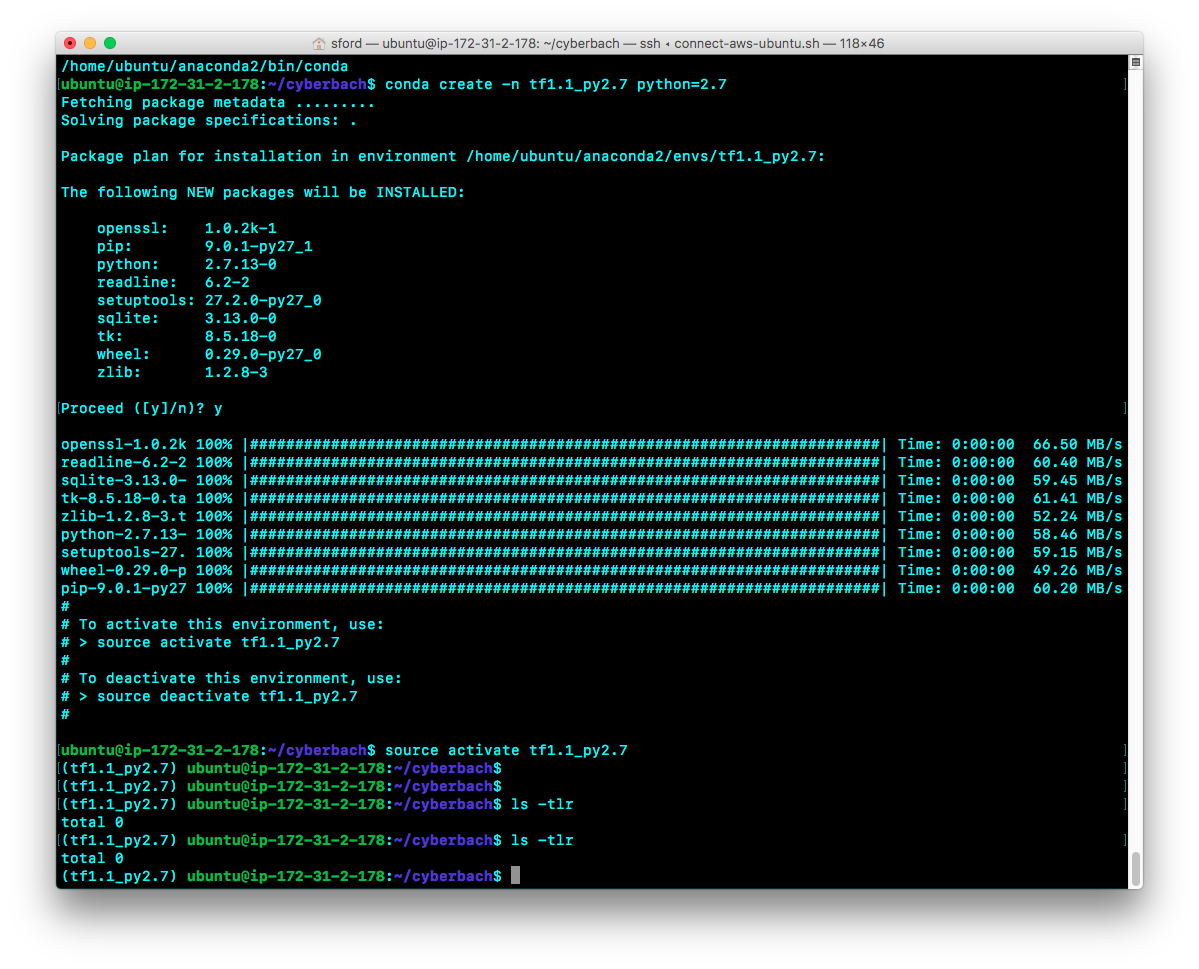
$ conda install pandas

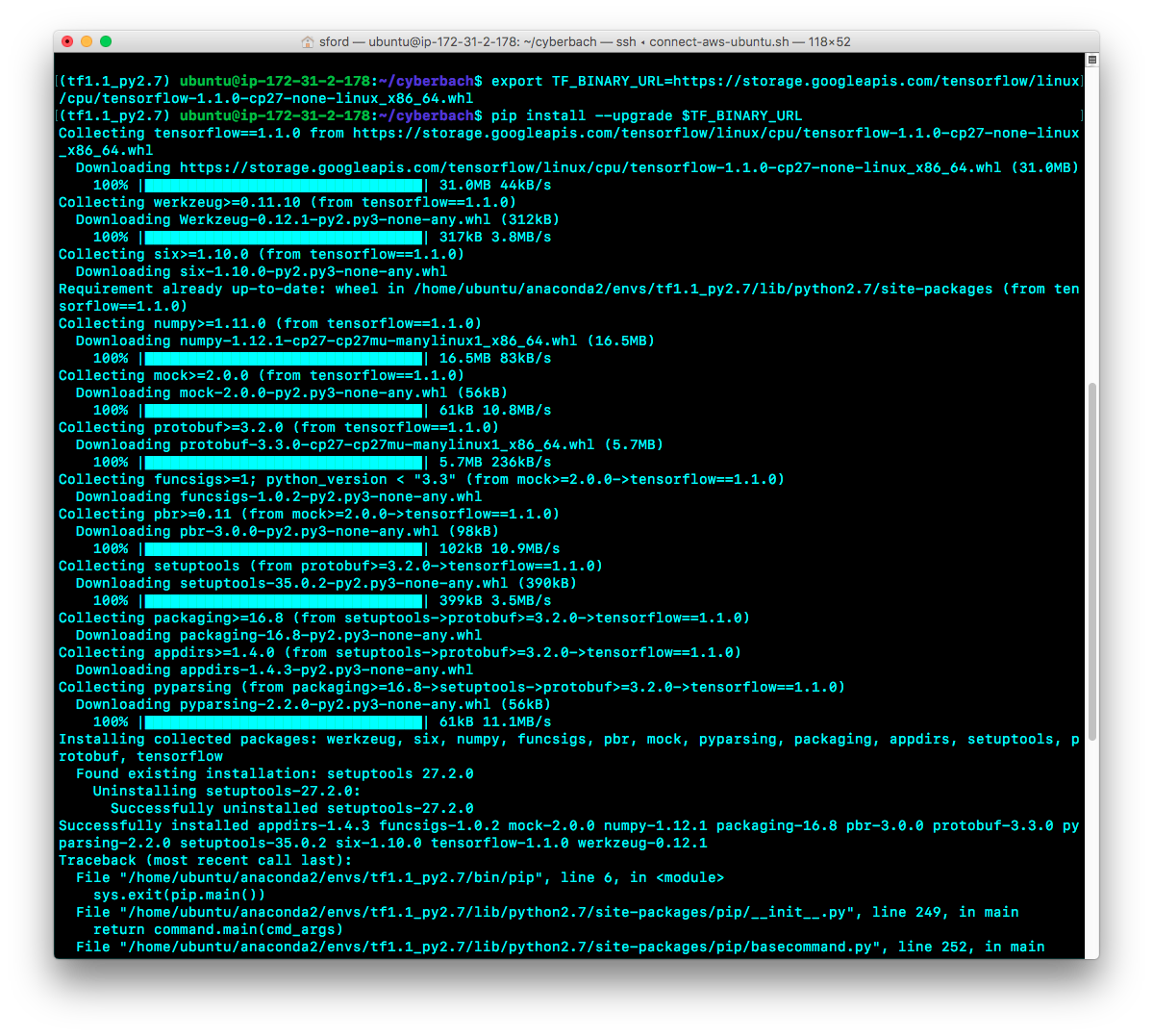
$ conda install -c conda-forge tqdm

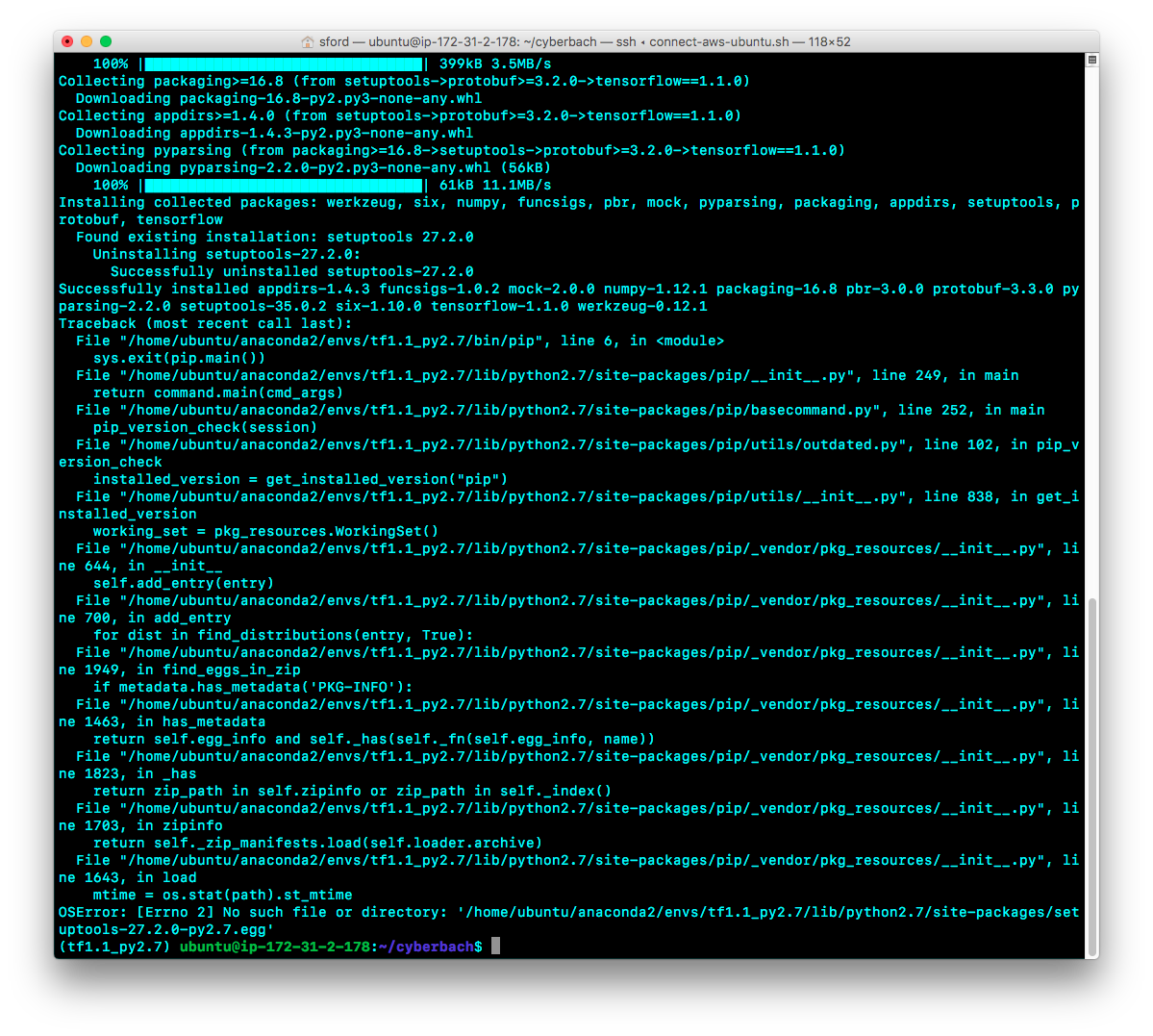
$ conda install matplotlib

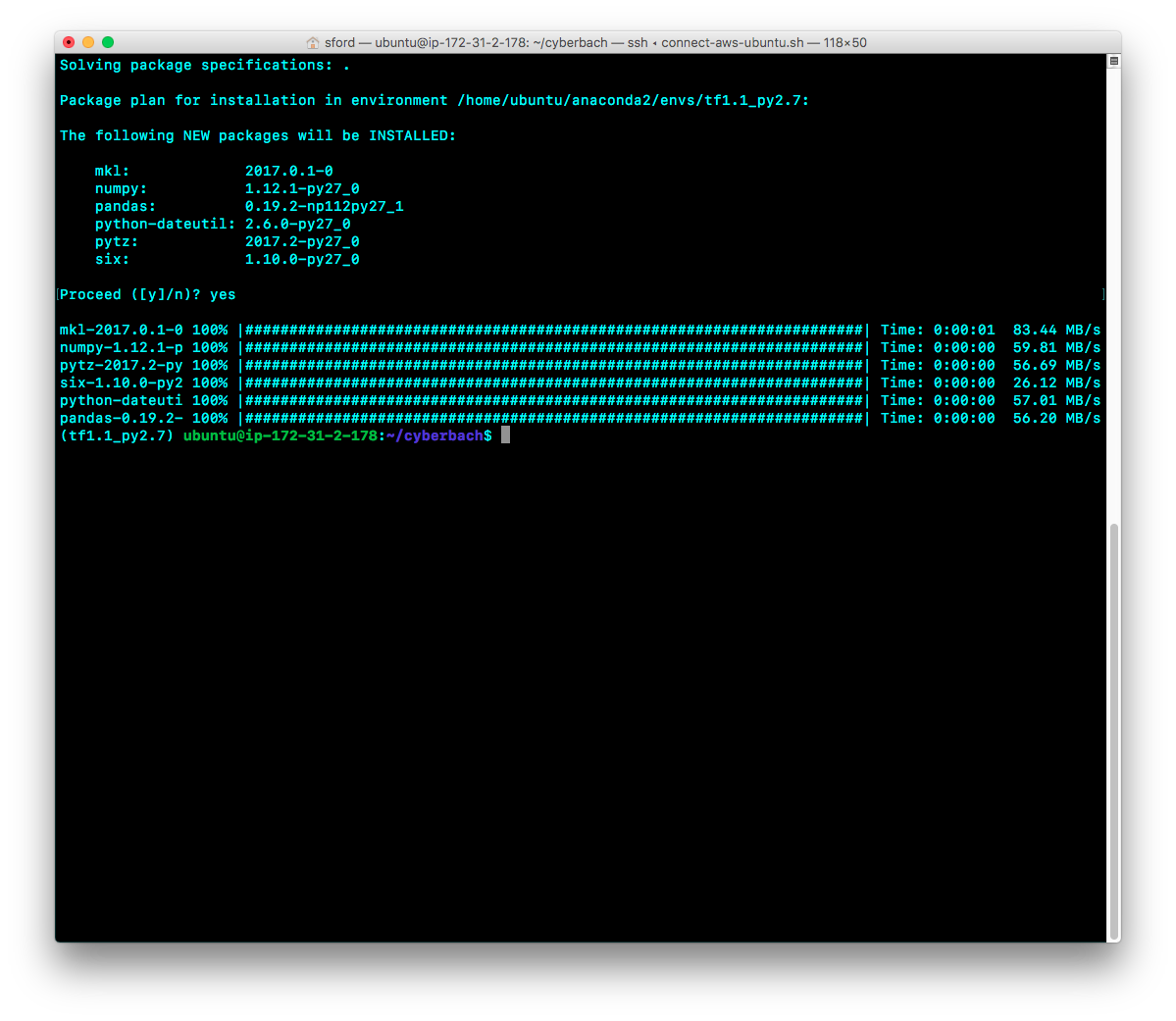
$ pip install python-midi

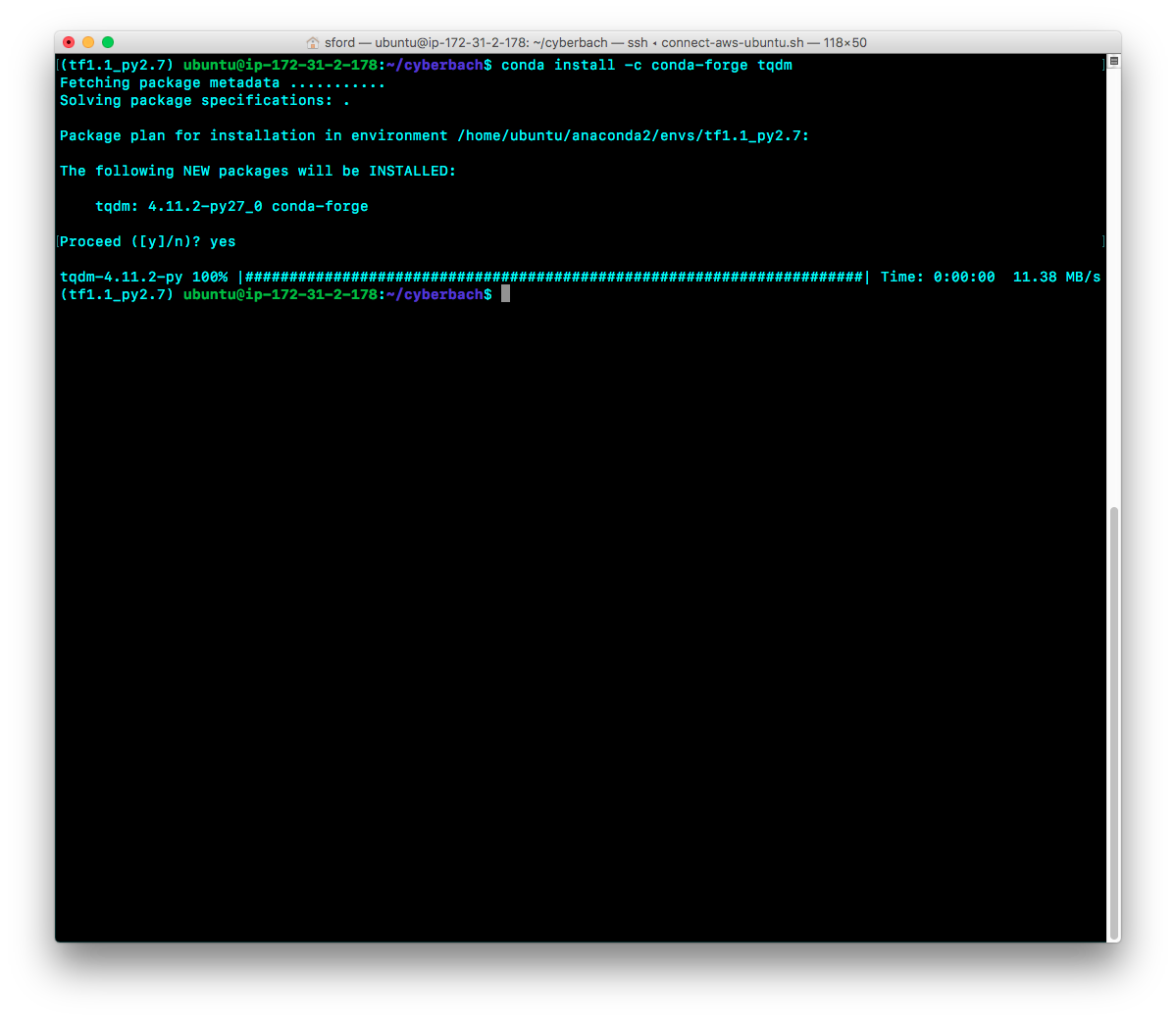
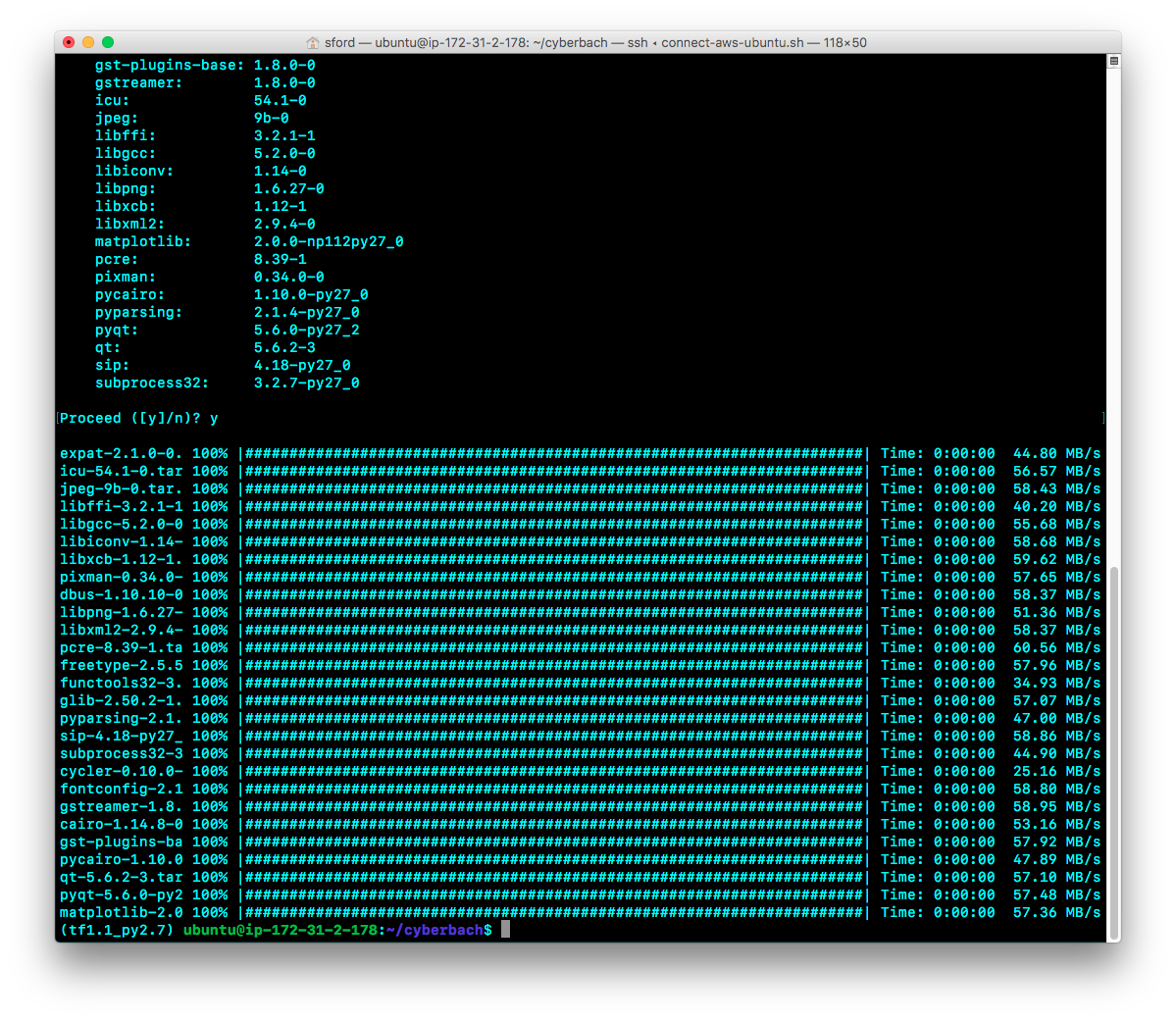
$ sudo apt install timidity







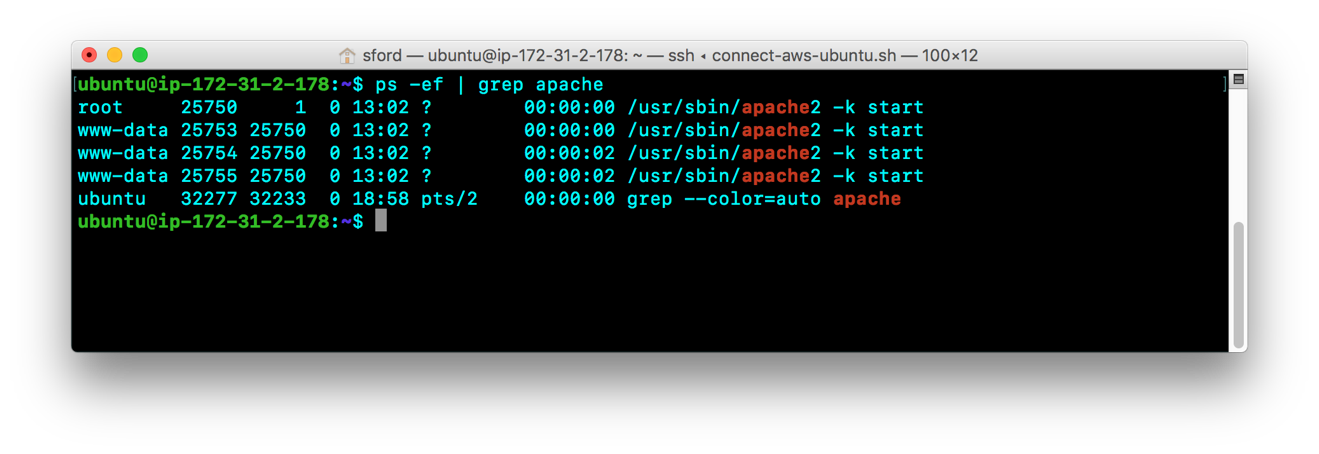




**Installation of web server components**

**Apache2 Server Configuration**

Ensure Apache2 is installed and running on Ubuntu. Apache2 comes installed on Ubuntu versions.



Go to the directory **/etc/apache2** and edit the file **apache2.conf**

Add the following entry to **apache2.conf** :

|  |
| --- |
| <Directory /home/ubuntu/cyberbach/tensorweb>  Options Indexes FollowSymLinks  AllowOverride None  Require all granted  </Directory> |

This configures the web server to access the /**home/ubuntu/tensorweb** directory for html content.

Go to the directory **/etc/apache2/sites-available** and edit the file **000-default.conf**

|  |
| --- |
| ServerAdmin webmaster@localhost  **DocumentRoot /home/ubuntu/cyberbach** |

This configures the document root for the web server to be **/home/ubuntu/cyberbach**

Go to the directory **/etc/apache2/config-available** and edit the file **serve-cgi-bin.conf**

|  |
| --- |
| <IfDefine ENABLE\_USR\_LIB\_CGI\_BIN>  ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/  **<Directory "/home/ubuntu/cyberbach/tensorpy">**  **AllowOverride None**  **Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch**  **Require all granted**  **AddHandler cgi-script .py**  **</Directory>**  </IfDefine> |

This configures the web server to access the /**home/ubuntu/tensorpy** directory for executing python files as CGI.

Once these configurations are made, restart apache2.

|  |
| --- |
| $ sudo systemctl restart apache2.service |

To enable python running CGIs in apache2, run the following command.

|  |
| --- |
| $ sudo a2enmod cgi |

This will automatically enable mod\_cgid

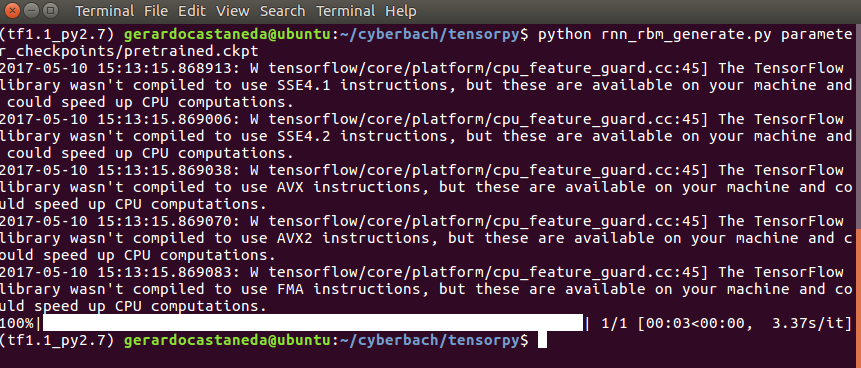
Based on this configuration all web content will successfully be accessed from **/home/ubuntu/cyberbach/tensorweb** and sub-directories.

In addition, all python programs will reside and be successfully executed as CGI from the **/home/ubuntu/cyberbach/tensorpy** directory**.**

**How to use the server based components with the command line**

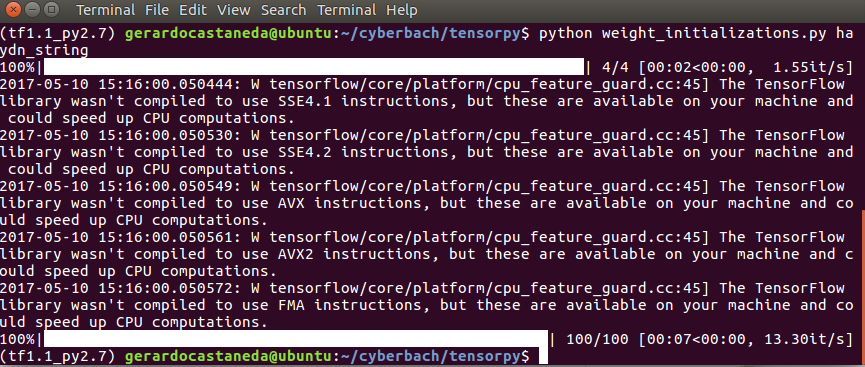
You can generate music with a provided pre-trained file by running:

$python rnn\_rbm\_generate.py parameter\_checkpoints/pretrained.ckpt



To train the model, first run the following command to initialize the parameters of the RBM.

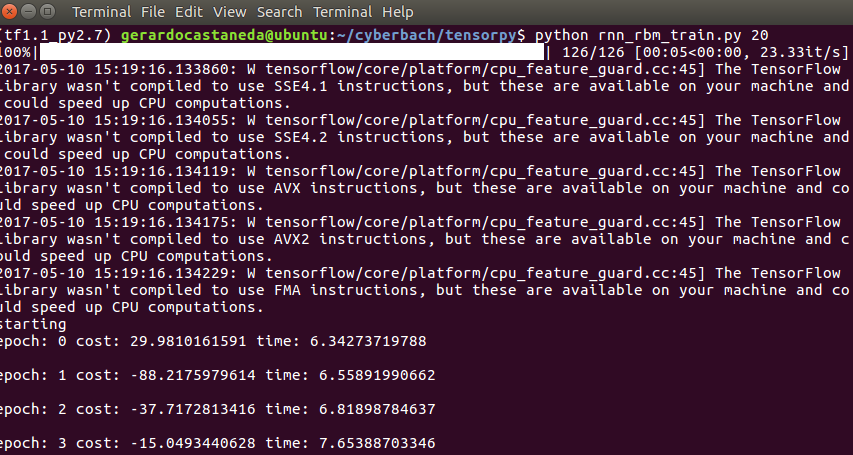
$python weight\_initializations.py <source\_music\_folder>

In the example, we are using **haydn\_string** as the source folder because that is where we have the midi data set for haydn 

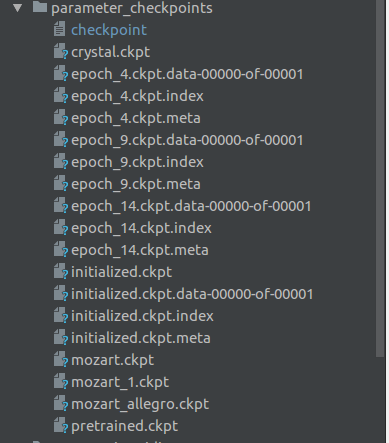
Then, run the following command to train the RNN\_RBM model:

$python rnn\_rbm\_train.py <num\_epochs>

num\_epochs can be any integer. In our experience 20-50 tend to converge quickly. The training will save .cpkt files in the parameter\_checkpoints folder



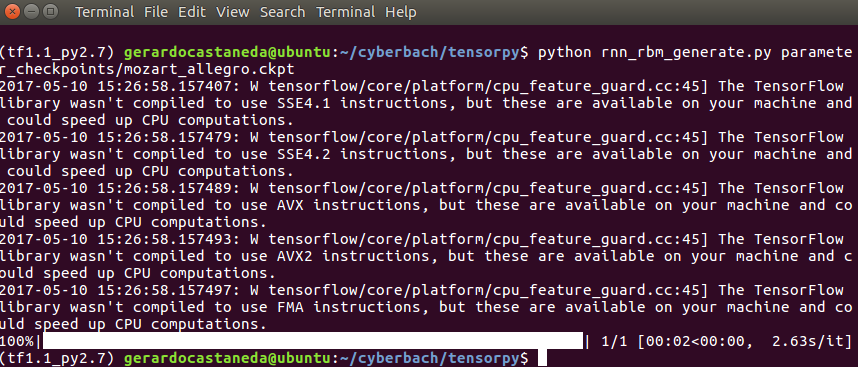
Training generates .ckpt files in the **parameter\_checkpoints** directory every few epochs. If you want to save a particular training result, the best is to rename the file. In the screenshot below we can see for example **mozart\_allegro.ckpt** which is one of the files we produced listening to Mozart MIDI files.



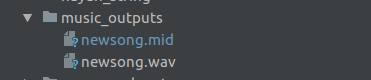
You can generate music with your newly trained files using the following command:

$python rnn\_rbm\_generate.py <path\_to\_ckpt\_file>

To document the outcome, we will train using our **mozart\_allegro.ckpt** file



That’s it! We generated a new song! The song is called newsong.mid and is available in the **music\_outputs** directory



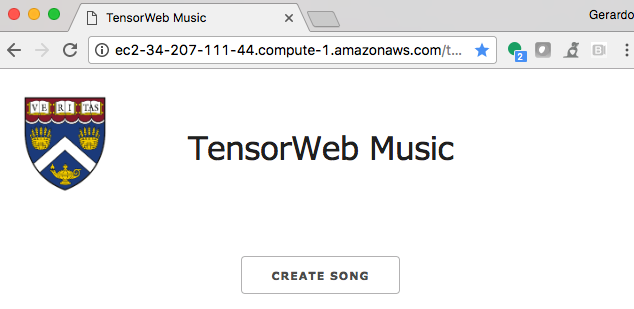
If you want to continue manually you can open the file with a midi synthesizer such as garage band



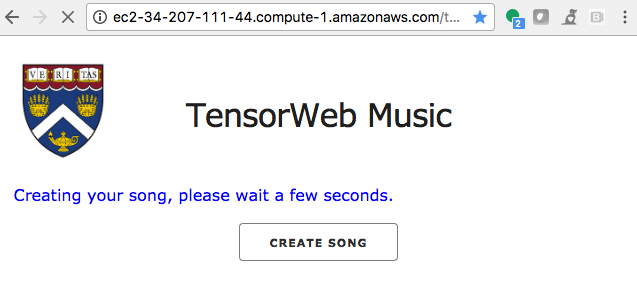
**Using the web app**

According to our design objective, the web app should require zero training…

Open a browser and go to <http://ec2-34-207-111-44.compute-1.amazonaws.com/tensorweb/index.html>

There is only one thing to do. Click on **Create Song**

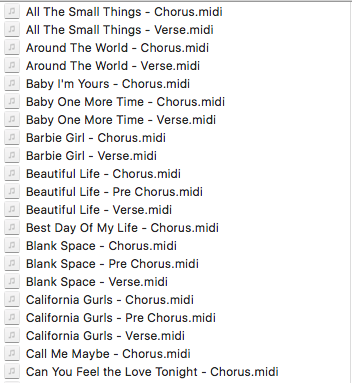
After 5-20 seconds, your song will play automatically.





**Data set**

To train the model we used a large sample of midi songs. The original solution from Dan Shiebler has several pop songs in the Pop\_Music\_Midi direcdtory



As part of experimenting with the solution, we also built other data sets of (open source) songs that had a more cohesive style. The goal was to check if the trained file produced significantly different results. The outcome of this testing was inconclusive. The additional MIDI files can be found on the following directories

* classical\_allegro
* classical\_moderato
* classical\_two
* haydn\_string

**What worked**

* Working on this project added knowledge to the programming community since we could not find a solution for RNN\_RBM that was working on TensorFlow 1.1
* Our solution really hides the complexity of TensorFlow, Python, MIDI manipulation, and audio conversion. When in doubt, push the only button on the screen!

**What did not work**

* Upgrading the code to TensorFlow 1.1 was very painful. The main issue was due to calls to deprecated experimental functions

control\_flow\_ops.while\_loop -> tf.while\_loop

* Calling python scripts and Timidity++ from the web server required quite a bit of manipulation. Main challenges were 500 errors due to malformed CGI headers and access issues to run programs, and modify files
* The original plan was to produce mp3 files but Timidity++ does not support that format. In general, it seems that Python libraries for music creation are falling behind

**Lessons learned**

* How recurrent neural networks work
* Music generation in TensorFlow is fun but it definitely has complexities
* Working with the latest and greatest technologies comes with challenges finding established libraries
* There is a need to share learnings with the programming community

**Submission Instructions:**

Submit final assignment on the course site. Upload your reports and slides separately. Upload your zip file that contains your software and data file. Include your visualization files: html, JavaScript, Python, R code and data file within a separate folder in the zip file. Ensure your visualization files and all dependent files run! TAs will deduct points if they have to edit your code to run it. Name your zip file: **YourLastNameYourFirstName\_Final.zip**

**Grading criteria:**

Project Report and practical software code example     50%

PowerPoint Slides                  20%

15 minute YouTube video               15%

2 minute YouTube video 10%

One page summary          5%

If you fail to provide practical software code example, you will lose at least 30% of the final project grade.

**Upgrade Log**

These are the main changes performed to upgrade Dan Shiebler’s original solution

1) Upgrade scripts to tensorflow 1

--------------------------------------------------------------------------------

Processing file 'RBM.py'

outputting to 'RBM.py'

--------------------------------------------------------------------------------

'RBM.py' Line 64

--------------------------------------------------------------------------------

Renamed function 'tf.mul' to 'tf.multiply'

Renamed function 'tf.sub' to 'tf.subtract'

Old: bv\_ = tf.mul(lr/size\_bt, tf.reduce\_sum(tf.sub(x, x\_sample), 0, True))

~~~~~~ ~~~~~~

New: bv\_ = tf.multiply(lr/size\_bt, tf.reduce\_sum(tf.subtract(x, x\_sample), 0, True))

~~~~~~~~~~~ ~~~~~~~~~~~

'RBM.py' Line 65

--------------------------------------------------------------------------------

Renamed function 'tf.mul' to 'tf.multiply'

Renamed function 'tf.sub' to 'tf.subtract'

Old: bh\_ = tf.mul(lr/size\_bt, tf.reduce\_sum(tf.sub(h, h\_sample), 0, True))

~~~~~~ ~~~~~~

New: bh\_ = tf.multiply(lr/size\_bt, tf.reduce\_sum(tf.subtract(h, h\_sample), 0, True))

~~~~~~~~~~~ ~~~~~~~~~~~

'RBM.py' Line 46

--------------------------------------------------------------------------------

Renamed function 'tf.sub' to 'tf.subtract'

Old: cost = tf.reduce\_mean(tf.sub(F(x), F(x\_sample)))

~~~~~~

New: cost = tf.reduce\_mean(tf.subtract(F(x), F(x\_sample)))

~~~~~~~~~~~

'RBM.py' Line 63

--------------------------------------------------------------------------------

Renamed function 'tf.mul' to 'tf.multiply'

Renamed function 'tf.sub' to 'tf.subtract'

Old: W\_ = tf.mul(lr/size\_bt, tf.sub(tf.matmul(tf.transpose(x), h), tf.matmul(tf.transpose(x\_sample), h\_sample)))

~~~~~~ ~~~~~~

New: W\_ = tf.multiply(lr/size\_bt, tf.subtract(tf.matmul(tf.transpose(x), h), tf.matmul(tf.transpose(x\_sample), h\_sample)))

~~~~~~~~~~~ ~~~~~~~~~~~

--------------------------------------------------------------------------------

Processing file 'rnn\_rbm.py'

outputting to 'rnn\_rbm.py'

--------------------------------------------------------------------------------

'rnn\_rbm.py' Line 74

--------------------------------------------------------------------------------

Added keyword 'concat\_dim' to reordered function 'tf.concat'

Added keyword 'values' to reordered function 'tf.concat'

Old: music = tf.concat(0, [music, x\_out])

New: music = tf.concat(axis=0, values=[music, x\_out])

--------------------------------------------------------------------------------

Processing file 'rnn\_rbm\_generate.py'

outputting to 'rnn\_rbm\_generate.py'

--------------------------------------------------------------------------------

'rnn\_rbm\_generate.py' Line 35

--------------------------------------------------------------------------------

Renamed function 'tf.initialize\_all\_variables' to 'tf.global\_variables\_initializer'

Old: init = tf.initialize\_all\_variables()

~~~~~~~~~~~~~~~~~~~~~~~~~~~

New: init = tf.global\_variables\_initializer()

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

--------------------------------------------------------------------------------

Processing file 'rnn\_rbm\_train.py'

outputting to 'rnn\_rbm\_train.py'

--------------------------------------------------------------------------------

'rnn\_rbm\_train.py' Line 38

--------------------------------------------------------------------------------

Renamed function 'tf.initialize\_all\_variables' to 'tf.global\_variables\_initializer'

Old: init = tf.initialize\_all\_variables()

~~~~~~~~~~~~~~~~~~~~~~~~~~~

New: init = tf.global\_variables\_initializer()

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

--------------------------------------------------------------------------------

Processing file 'weight\_initializations.py'

outputting to 'weight\_initializations.py'

--------------------------------------------------------------------------------

'weight\_initializations.py' Line 46

--------------------------------------------------------------------------------

Renamed function 'tf.initialize\_all\_variables' to 'tf.global\_variables\_initializer'

Old: init = tf.initialize\_all\_variables()

~~~~~~~~~~~~~~~~~~~~~~~~~~~

New: init = tf.global\_variables\_initializer()

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

2) Upgrade functions and replace deprecated or non-supported calls

In RBM.py

------------

# replace deprecated control\_flowOps.While

# the original parameters are While(condition, body, [initial\_accumulator])

#[\_, \_, x\_sample] = control\_flow\_ops.While(lambda count, num\_iter, \*args: count < num\_iter,gibbs\_step, [ct, tf.constant(k), x], 1, False)

# while\_loop(cond, body, loop\_vars, shape\_invariants=None, parallel\_iterations=10, back\_prop=True, swap\_memory=False, name=None)

[\_, \_, x\_sample] = tf.while\_loop(lambda count, num\_iter, \*args: count < num\_iter, gibbs\_step,

[ct, tf.constant(k), x], parallel\_iterations=1, back\_prop=False)

In rnn\_rbm\_train

--------------

if \_\_name\_\_ == "\_\_main\_\_":

#add variable num\_epochs

num\_epochs = 50

main(int(num\_epochs))

In rnn\_rbm\_generate

--------------------

# Gerardo Castaneda April 2017, replace sys.argv[1] with path\_to\_ckpt\_file

# so we can run without need for console input

#main(sys.argv[1])

path\_to\_ckpt\_file = 'parameter\_checkpoints/pretrained.ckpt'

main(path\_to\_ckpt\_file)

In midi\_manipulation.py

# Gerardo Castaneda April 2017 - Cast to integer to avoid error

#song = song[:np.floor(song.shape[0]/num\_timesteps)\*num\_timesteps]

song = song[:int(np.floor(song.shape[0]/num\_timesteps)\*num\_timesteps)]

**References and credits**

* A Beginner’s Tutorial for Restricted Boltzmann Machines <https://deeplearning4j.org/restrictedboltzmannmachine>
* How to build an RNN-RBM for longer musical compositions in TensorFlow <http://danshiebler.com/2016-08-17-musical-tensorflow-part-two-the-rnn-rbm/>
* All images labeled for reuse. Sources: Wikimedia, Flickr, Pixabay