Music Generation using Recurrent Neural Network

Shradha Agrawal
A53105993
sha015@eng.ucsd.edu

Sudhanshu Bahety
A53209213
sbahety@eng.ucsd.edu

Jean Choi
A53214201
jsc078@eng.ucsd.edu

Ajitesh Gupta
A53220177
ajgupta@eng.ucsd.edu

Nimish Srivastava
A53222968
n2srivas@eng.ucsd.edu

Abstract

In this assignment we trained a Recurrent Neural Network to learn to generate music by training it a large number of music samples in MIDI format. We used slices of 25 characters while training and mostly used a learning rate of 0.0001 with the Adam optimizer and batches of 64.

In our experiments we are able to achieve training accuracies of around 50-55% while the validation accuracy varies from 40-45%. We see that increasing the length of training slices does not have a profound effect on the accuracy, only increasing marginally for larger slices. Increasing the number of neurons in hidden layer shows a marked improvement in the accuracy. Adding dropout gives a small drop in accuracy which we think might be due to lack of data to generalize very well on. Among optimizers other than Adam, RMSProp seems to learn faster and better than Adagrad. At the end we are also able to visualize which neuron learns what about the music. We can see neurons being able to learn specific notes, length of notes, bar lines and other musical patterns.

1 Description of Data

The given data is a collection of music files in ABC notation. There are a total of 1,124 music files in the input data. The beginning and the end of a music file are marked by <start> and <end>. A music file is mainly composed of tune header and tune body. A music file may only contain one of the two. Tune header specifies reference number, tune title, rhythm, composer, composer, and area. Tune body is the tune itself in ABC notation.

We removed <end> tag as it is redundant and only <start> can be used to split input file into separate tunes. Next we identified 94 different characters used for notation of tunes ignoring <start> tag. We then used '$' character which wasn't among these 94 characters to encode <start> tag. After this, every character was one hot encoded in 95 length vector.

2 Network Architecture and Training

We have used the Keras Simple_RNN implementation to create one RNN layer of 100 hidden units and 95 input units. The output of this RNN layer is passed through a Lambda layer which handles softening by the temperature parameter, before finally being passed to a Softmax output. For most of the experiments, this structure of network is consistent unless mentioned otherwise.
We first divide data into training and validation set by performing an 80-20% split of data. For training this network, we have used Adam optimizers unless mentioned otherwise. Parameters for Adam solver are given where they were used. Another observation we made was although, random

slices of the ABC notations provided for training gave good enough accuracy to generate the tunes, the results were improving if the network was fine-tuned with longer sequences, possibly consisting of entire tune sequences. For this we experimented with appropriately padded sequences upto a length of 1200 due to memory constraints and observed better generated tunes as finetuning helped the network learn the entire ABC notation structure and patterns of repetition in notes which make the tune soothing and continuous.
With this in mind and to mitigate the memory limitation faced for the fine-tuning data, we generated new training data consisting of sequentially sliced data samples and trained our RNN with statefulness. This way the network was able to grasp finer details about the file structure and patterns of repetition in the tunes and hence imitate them better in the generated tunes.

3 Generation of music

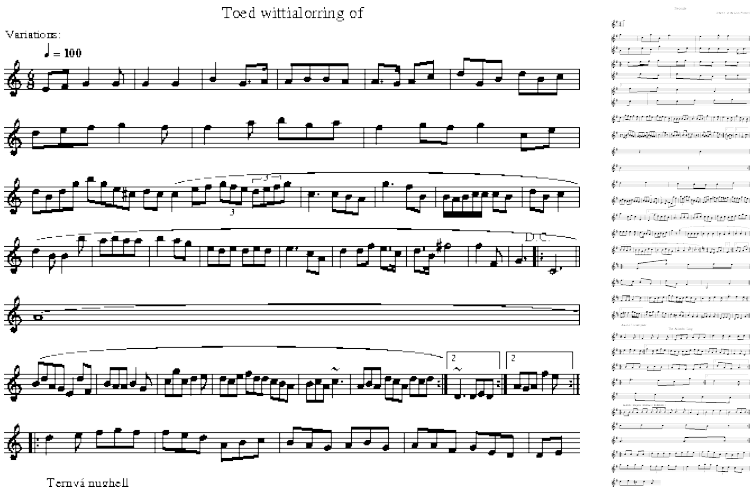
We tried two approaches for generating music:
Char to char: many to many mapping
In this, we passed one character as input to network and asked network to generate next character which will be feed as input on next step.

Sequence to char: many to one mapping
This is similar to first approach except that instead of just feeding the output character, we feed concatenated string of output character and input string as input on next step. However, note that network will generate as many characters as in input. We take only last character as output character.

In both approaches, output character is predicted by flipping an 95-sided coin, as there are 95 different characters, based on the probability distribution at the softmax layer. We observed that second approach generated coherent music which exhibited structure while first approach generate music with very distant notes. Thus, in all our experiments, we have used the second approach.

4 Effect of Temperature

We experimented with different temperature parameters to generate music. The results shown in Figure 1, Figure 2 are corresponding to a Temperature parameter of 0.5, Figure 3, Figure 4 for Temperature = 1 and Figure 5, Figure 6 for Temperature = 2. The network hyperparameters are as listed:
We used Adam optimizer $\beta_1 = 0.9, \beta_2 = 0.999, \eta = 10^{-3}$ with ReLU non-linearity, to train over 100 epochs with a batch size of 64 containing note sequences of length 25. The network architecture had an input layer, followed by a Keras simple_rnn layer with 100 hidden units and a Lambda layer before the output Softmax, on which the Temperature parameter was applied before feeding to the Softmax.



(a) Sample 1 (b) Sample 2

Figure 1: Two Sample Music Generated with Temperature = 0.5

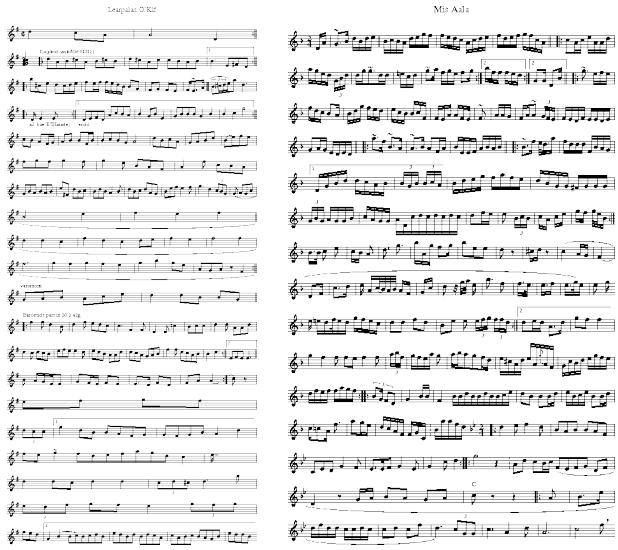
```
X:46
T:Toed wittialorring of
D:John nd diven: Splaneele jigsh we's Slaner
R:Branle d'sfa blave
W:Terny\'a nughell
R:jig
H:See also #9, Cul
R:carolane
D: ef-en
R:jig
Z:id:hn-cardcos -1977-22
G,c|
"Em"G2A|"G7"d^F | DAce d3-fde "D7"d2 FA|"G" dB|c2a2 f2ed||
P:Variations
|: EGF G2E|fef g2g | edB cde | afg fdf | dge ~g2d | ~e3- dBG :|
|: zgfde | c>Bcc | A2G2 zBd.c |
fecc |
cBcd | ~c^c | d2d2 | (cgf "C"c/d/c/c/|B/AG/g/d/|\
GD2 def|e2d ecA|d2e fB|cA A/F/|1G3 GFD|D2E ~E3 | GEF FED AGE |1 GGE E3 :|
E4A,C, | D,2D,A, | D4CE, | D,2D/2 DEE :|1 GGG E2B |
GFG ~F3 :|
P:Variations:
ed ca|(3efg be (3BGd|(3Bcd Ac/B/|AF FE EB|eA Ad|ff|ec de|cA/G/|A/G/F/E/|1 EE F2|F2 F>D FA|B3 FA|B2 G>B|AA G2:|
|:dd|AG (3Bde|fedc AGAB|d2GB c2dcA|EcED fedc|~B3c BAGE|EGAB AGAB|cedc BGgg|fafe fgff|g2ed cABA|BcdB AGAc|G2FG BGGB|(3AFG Ad (3GBd (3GFA (GBcB ~A2D |
(fBcd ^c4-cc.A | B4A2B/2 | B6 z>GB | d2d2 | dc/d/ dc | Ad cA | ABc | B3 :|
V:1
Q:"The Willis lande.
D:Faricat nobman: the D. Y"Lan loud, 1
M:2/4
L:1/8
K:Vanie
A:Provence
C:Trad.|
R:Earant leatimer.ck's
Z:id:hn-jig-31
M:6/8
L:1/8
Q:1/4=100
K:C
EF G2 G | G2 G2 | B2 G>A | ABBA | A>G Ac | dGB dBc | def g2f | f2a bga | f2gf g2ce | dBdg bge^c dc(c2ef (3gfe (3defg | c3cBA | g3fB | B/A/B/c/2c/2 cB | dB c2 |
d2B B2b | abaa | b2ag | ede ded | e3 cA- | d2df e>c | d>B ^f4 | f2F | G3 !D.C.! |: C6 ||
AB|
BdAG E2dF | BAA B2G | cgc d2e | dce fdd cBA | BcA ~c3 | ABA dcA dcd :|2 ~D3 DED:|2 AGA f2e:|
|:d2e fgf|fed ABc|BAG ABG|AAF GED|DE
```

(a) Sample 1

```
X: 2 desternpe
O:France
A:Provecie
C:Johen's  -st the Live
R:Baraloube des in EIN've
R:htghl, Gie la. ig flmoid in & Ssisin Stee dlacome
Z:Thradofes
T:Seorale
L:1/8
K:G
a2|caB g3|e2g f2||
|:ef/g/ af/a/|fe fe|1
fg d2 A2 |]
V:4
P|
Adec a2ce|eddc B<dec|eBA Bcd|edf fgd | agf g2f | f3 fge | gef e2c | BBB BAB | B3 BAB | g3 | e2f | e3 c2d | g2d B2B |
z2z E2B dBAG | G^GG (AF BA)GG | c4 | g4 fz | d2 A>B B2 | B>B d>d | B2z2 | BG AGc | g3 fd :|2 dA cB |1 c3 | F3 c z |
z2 B2  |
B2z2 :|
d2e2 | cB/A/Bd |
BdB ABc AAG|BAA ~F2GE|~E3D FDaf|(3bea ba cB AG|FGc e2^gf ag^fg|affe a2ge|
a4ce c2E2|1 (D2fd)cdB | A3 ABA | ABc ccA | dBc Bcd ege | eef e2e | d>B BA | BB/c/ Bd | g2 z -  (a4 | BBBB | B2B2B2 | B2A2B2 | B2B2z2 | BBAB | BBGBdc | B2BdBG | A3 A2B | B3 :| [P:Ture la Cos chereat, The
D:Carblery
Z:id:hn-polka-93
M:4/4
L:1/8
Q:2/4
L:1/8
K:D
ED FA|1E3:|2 AG Ac||
|:E>f ef/2e/2|
f>d cG | eG F2 | G,4 G4 |]
FG/G/ (A2d) | | d><d>e | a/ge/c/d/ edB | c2d e2d | ~f3 f2e :|2 e^f f2 a/g/e/e/ | : a ga | a2 fe|d2 ^B2BA|B2G2 G4^FGG2G||
DEFC | CE D2 G2 :|
P:variatiogs
W:Au-rl-L7 l cost plier
R|---
T:Allandn: Lng, The
R:hornpipe-7
M:C|
K:G
A2G|F2G G2E|DFG AGA|d3d c2cd||
|:e2f e2d:|2 g2d ege|dcB cce fec|BAB AGE|F2F D2||
|:BAB BcB|AFE EAB|1 dAF A3:|
|:-f3 eca:|2 d2 (3/2c/|
g/eG/AG/2 |
|:- "BeB2f2 | B2g2z2 | B2B2e2 | B2BB2B2 |
B>B BA BB B>BB>B | AB3B | A2E2 | B2d2d2 | B3 BAB :|
cdc dBz :|
G3 G>A|c|
|:BB AB|dcdc defe|fgeB cAce|
agag (3ffe gf|e2(3gba (3gcd cA |
de^cd A|
```

(b) Sample 2

Figure 2: Two Sample Music Text Generated with Temperature = 0.5 in ABC Notation



(a) Sample 1 (b) Sample 2

Figure 3: Two Sample Music Generated with Temperature = 1.0

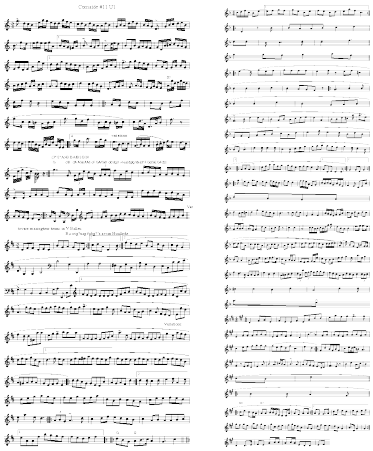
```
X: 114
T:Learpaian O'Kif
Z:id:hn-reel-28
M:C|
K:G
d2cA|A4 D2:|
[G4G2:|2 B2c2|F2c2|e4d4 |]"C\bigdecd cac|eAGF ECD2:|
|:dAB^cA B^cB|d^cAc cBAB|1 B3 E^FD:|
|: b^ce dc/d/e/|fe/f/ f/e/d/B/ BG|FG FE|F/~E3/A/|] (3B^c/d/e/)a d2>2
|:
D E2E:|2 DEFD AcBA|B2 GBE2|GBBB G2B2|B2c2 g4:|
w:a2 bue E'DIninde) ercid.
H:Bes ap
ed ecA d2A |1 G2AG A2Ac |
efg e2g |
faf g2f | fed edc | Bef g2gf | g3dfa d2fe |
d3 dfe | d2e g2g | g2g f2f | (dd)d | A>B GB | cG ze | e2f2 | c2 !fic>e||
|:a2f|1 dded edcA||
Z:Transcrit Ht1is Maksyed que hon Vellowe.
H:Sde aB|e3 g2f||
|:dged e2fr|g2aa gfed|(3efg fedB|cBAB~a3a|ggbg g2ab|d2ed cedf|agag abab|baba | defe d2ce | c2B2 B2B2 | B2B2B2 | B2cBd2 | d3B | d2 BA | F2F2FD | D2D2D2 | D4 EF | B2 B2 |]
dc- B2B | A2A/2G/2|
[P:Iticha11 sctdong
H:Delowe.
G:Matth! |: BA/B/ E2 BB | c2 dB | Bd cB/c/ |
cB AF/G/ | AB BF/B/ | BcBG A4 | dBcA | GABd | (Be)fe |
fgf g2f | g2e BAF | A2f cBA |
GBA BAB | e^F dBc | BdB A2B | dBc dAG | Bef e2f | g3 d2(3cBG | (G AGc |
B4e2 | f2e2f2 ||
d2d2c2 | f2 d2 c2de | f2e2 | f4f2 |
f6 | g2f2g2 f2e2 | g2b2ge | AGA zF2 :|
P:variations.
GAB Ace|d2 dB BG||
P:Baroetid t part in 20 2 a2g
|: f2f d2B | d2f edc | c3 F2F | D2D ~c2 | Bcc | d2d AAd | d2c dcB | efe e2d | edB AAd | dcA GAB | f2f  dcAB |
B2d3 :|2 EFE E3 ||
B A/G/E/E |GF/E/ FE|E>D FD|FA A2:|(3-c)z |
ef gf |
(3dcc |1 g2dB BAFG | GFGB A2d2 |
fe~g2 fdefagdc2 | e692 |
d2dc d2d2 | (3cd^c d3b|
(3edf gf e2 (3gfg |
(33fed |1 G2(B2Bc)BcdB | ed}Bd BdcA | cABc BAGB | eddB
```

(a) Sample 1

```
X:46
T:Mis Aala
M:2/4
K:F
DA|~G3B dBde||
BcBA degf|fedc Acde|fdef gfcA||
|:c2Ac eAAd|aged (3GBd|dg~g3dd |
=ecd|~g2a fge f2g:|2 bff fed:|2 G2G D2B||
|:~g3 efe|fdc B2A|Bdg fed|ecB BcF|FEA AFE|~F3|AAG|FA~A2 (3GFE|FEDF EEDE|
G2AG EDED||
|:~a3f ~B3A|~A3B A2FA|BEEF ABAB||
|:~B3f e<da}|
fddd FBAG|1 DGBd dcAB|(3BGB (3BcA|Bdeg fafe|dBGG ^FGGG|(3GBG AGGB|cAGG ADc=|d=^cdcdc|d2fe|e2 (3BcB|cA ce|gf g/e/B:|
B:| AdBc|BcBA F2EF|EFAB AAGA|(3BBc dc dB| d2e dBd | deg gfe | fec d efc|1 G3 G2:|
B>B c2 e2 | cB A2 | d3 g3 | bAg f2g | b2e2e2 | d2z2 | ^cdz2 | (c4 AG | F4- | D2F | c3d | e A(c/B/)A/B/ AF | Gc |
d2 z/2g/2a/2 e|b3 f2 | f>e> | e3f | e>g)e | e2 g!wentt
=edf/ed gfec|d2d2 f2d2|B2e2 e2 (3efg (3ffd :|2
d2ed A3d|g2f2 f4|
g4f4f2 f2e2 f4f2|
(3abf|gfea g2|aba2g2fd|e^cAd eA{g}AF|GcBd egfe | dfef fagf :|
|:(3B2G F2D2|
G4BB|F4BBdBBdBBce|d2B2 B2Bd|eBce B2Bd|c=c2A f3|a2ge|dcAF FGC^T:Braillafd
O:France
Z:Transcoit et/ou maryis
R:hornpipe
Z:id:hn-polka-36
M:2/4
L:1/8
K:Bb
|: d2e f2d | BAB BAF | CED G2F | E>EF G2E | G2c d2 :|
|: g4 z2 : Ad cB/c/ (FEFG DF | D2 zG/A/ BA GA | "C"cz z2 |:
A 2|
B>e | d/f/d/d/ c/c/d/c/ | cd cf | (32d3) (3cGG|
c3 A2d | A6 | c4!breath!f|
```

(b) Sample 2

Figure 4: Two Sample Music Text Generated with Temperature = 1.0 in ABC Notation



(a) Sample 1 (b) Sample 2

Figure 5: Two Sample Music Generated with Temperature = 2.0

```
<start>
X:93
T:Cornaide #11 U1
M:3/4
L:1/16
K:cd d4 d2 | gfeg |
~g3 dcB|eag edd afe|cdB dBA AdB|GBA dfd|BAG ~D3|GBc deB|B3 f2d|efe efA|1 BAG ^FA|~B3 AGB|~B3d/ed|B2EB GAGF|GDB^c dBAG|
EGDE A2ge|1 d2dg edcB|A2Bc FAcA|
~f3g afed|B2AF GB BG|ABcd GA Bc|G3G dcBc|d4 cdBG|
D3FGA|F2 A>G|FE/G/ FE|"D7"Ag/d/ e/g/e/d/ | df ez | edc GGG |
|:AG dc | c2B d2f | g2c c=B ABc | g3 gdf | B>cB =Bc) | cd/>A D>AB | f3 z2=+!dd/c/ | e/e/B/A/ F/B/A/d/ A/E/E:|
|:4 gf | b3 eag |
P:variations
e>B c>c | ((G>B)GG | .B/.c/d/) fe | d>c | (F/^=e/2g/2 c/2e/2 B/2/2/2 B)z :|
P:(3"G"A>B BA|B2 B2||
"G"g7"dB "G"dA AG|(AA/) cF GA/G/|1 dd dg|1 bf ea|dgfg fde(d^F G2GE GFd2|
(3GBe dB (3GAG fAed|cdFA ABAG|~D3D FAAF|d2AG AB|AGFA D2EE|fdfd cdBA|AAGE E2D2:|
|:AB, AA, D,D E2~G2|GA BBdd cBAA|BAGF GAGB||
P:Variations: 2 ae~l.
W: tovare maaingtour tounc in V Balleu
O:France
A:Provench
C:hne Belly
O:France
B:Dellanses hongil Cono
T:B,cwig?uag fubg?'s arvan Huullede
M:3/4
L:1/8
K:Edor
DFF DED|BGA GDE:|2 cdA D2B:|2 (fede BA dB|cB G2|DG,2G,2B,4|D4G,2D,,2|A,4A,4|^F,D,G,C, | B,G,2D,2E,C,CCE|~G,3G,2B,,4B,,
2C,2G,A, D,, C,E |1 FEDE DDDE DFAB|BAFD DEFE|ECcB|~A3 gfe|gfg agf|geg ~a3|ede g2d|cAA dAB|c2A FED|Efd|f2d d^Faa|
eg~g2 baya||
AAGA GABd|~a3e AGBA BGAB||
P:Variations:
g2e agf|dfe gdB|1 d3f ed|dfe^faf fedg|fded dfed|edge fedB|A^GABA|GEFG E2:|
|:ddBG GEDB|AGAF d2:|2 d2d c2d|e2f A2e|deg age|feg efa|dBd cBA|cee fed|e a2e d3:|
|:(3ABG ~A3d|c2df edcB|c2AB AABc|1 eAAf bagf|fede f4 |:
|: "G"dBG "D"AB>cA2|G3c2 ||
|:
```

(a) Sample 1

```
<start>
X:530
K:FA
c/d/f/c/ Be~|1 ef gf|ge ce|fe cd|eaf c3:|
|:c2zB | BAAB c2d2 |
dd|Bc3d | d4f3||
g2gf g2af||
|: (e2d) (gef | gfff | e2fz ef^c2 |
B2B2 B2c2 | B2AGFz |: G3 B2B |
B2B2d2 | B4 GAB |
~F3B d2dB | B2GB Acde|d2G2 | BBd^c a<s : Lowmathrel
R:slide
H:See also #5
Z:Transcrit et/ou corrig? par Mich lefr
R:Blann
C:Tradr?r
Z:Pola trin Firre,2B Me2A |
DFD GAB cAd|e2g e2d BdB|
BG B3|e2e e2e|c2c deg|gab agf|g2e d2B|1
|: ABA ~A3 | dfa afg | gfd efe d2A |1 ADE D2D:|2 G2GE G2:|
|:guied g>e|ed f/e/d/c/ | fed2 | f/f/e/A/ FG | c/B/c/B/GB/G/E/2 | F2E CEB |
KdBGA |
d2 (3Bc^c | d2g2 edcB | B6 zc dc | E2 G2 F2 | B/A/G/E/ GG/B/ | c>B dB dc | GB/A/B | A>F G2 dc | Bc dz | c2c2 | A4 c/c/
A/B/ c/B/d/g/ a | dBA B2 | e>f f>e | d>d e>A | c>c c>A | B>c cB | Ac ce | de d>e | dc d/d/e/e/) | (). A/) | g) BG E/
BG/G/ | BG A2 | AB>ce | d>e f>d | cA B/A/A | FEF G/F/e/ | dB Bc/2 | ef g2e | b>e g2 | f2 d3B ABAB | e2c2d2 | A=Bcd |
(3dBb (3efg fggf|edcA BdcB c2(3c8c|
^c2c2B2d2|A4d2d2 |
<savre
R:See
S:Camachain the Eile, Theds tou loutrethra
R:Leacl-10
M:6/8
K:Edor
C:Orin.cornpipe-89
M:6/8
K:A
D2GAE | GEEG EAdd|eafg ~g3g|
E2GG GABA|EFAF G2:D2|B3-A2GA | AGFA ABGA | c2ef gaag | (gefdg c3 | f2 d2 :: GdB/G/B/G/ | ce/d//2e/2=B A3|F2z EFD|E2A !
breate! | B2A3/ =c/B/A | ~83A | ABGF E>CD | F3 Ez | B2 d2 A>c | B2
(cB | e4 ||
|:B2B A2F||
|:"Em"ce cB/A/|"D7"AA B/B/G/E/ DD/A/|AF A>A:|2 AG G>A|df ed|ed ef|gf ge|af ge|gc de|fg/e/ dB|AG|df e>e|dB G2|B>D G2|
G2:|
D2 D>E|FE AB/B/|AB/A/ EDD|EF
```

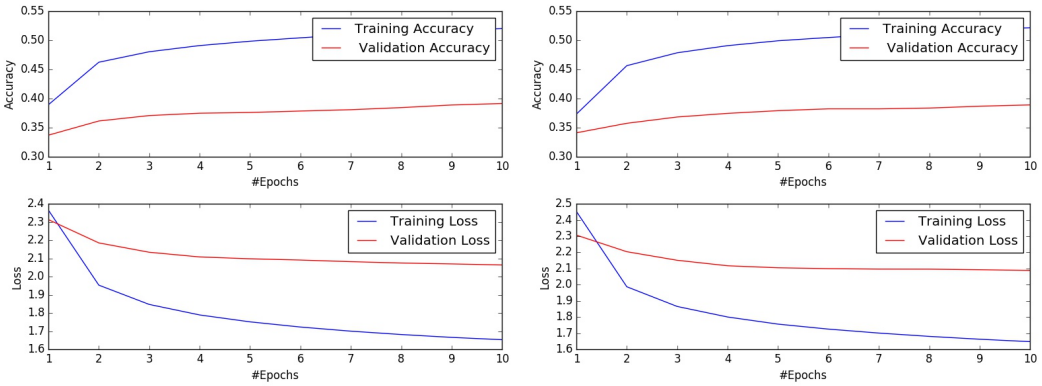
(b) Sample 2

Figure 6: Two Sample Music Text Generated with Temperature = 2.0 in ABC Notation

We observe that for lower temperature parameter we generated notes which were in smaller, less continuous sequences and these were repeated after very small intervals of times, like the chords

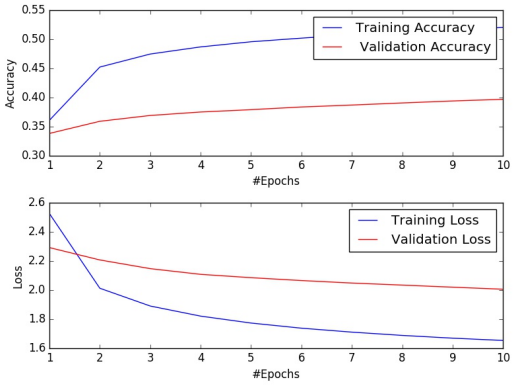
being repeated for T=0.5. Lower temperature also results in most of the predicted outputs being spaces or delimiters as they occur in higher frequency in the training data comparative to each note. As higher temperature softens the peaks of Softmax, we get much softer, regular tones which do not have as many gaps or delimiters. Also the music produced is softer with repetitions only happening after long stretches which only add to the feel of continuity in the tune generated.
Another discrete observation is the increasing length of note sequences with temperature. This is because for a fixed length of generated music string, the Softmax with lower temperature will give output closer to what it has already seen in the training data, which had more frequency of delimiters than any specific note. Thus, the Softmax with higher temperature will generate more notes as compared to the one with lower temperature and hence an increase in number of valid notes with temperature.
All these observations are theoretically coherent too; as they support the fact that decreasing the temperature makes the RNN more conservative in its predictions which are made with higher confidence, i.e.: peaked-out softmax for delimiters in training data. Conversely with higher temperature we get more diversity, i.e. more notes, in the predicted outputs although at the cost of lower confidence of prediction.

5 Effect of Length of Slice



(a) The length is 25 characters

(b) The length is 35 characters



(c) The length is 45 characters

Figure 7: Accuracy and loss with varied lengths of a slice

Length	Train Acc(%)	Validation Acc(%)	Train Loss	Validation Loss
25	52.07	39.35	1.65	2.04
35	52.17	38.91	1.64	2.08
45	52.04	39.71	1.65	2.00

Table 1: Experiments with slice length parameter

Denote the length of a slice as timesteps. We experimented with timesteps as 25, 35, and 45. As we can observe from Table 1, training accuracy doesn't change much with length of timesteps. However, validation accuracy improves with increase in length of timesteps. It might be due to the reason that increased length of timesteps allows network to remember larger context and hence generalize better.

6 Effect of Number of Epochs

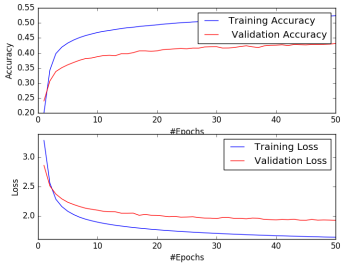
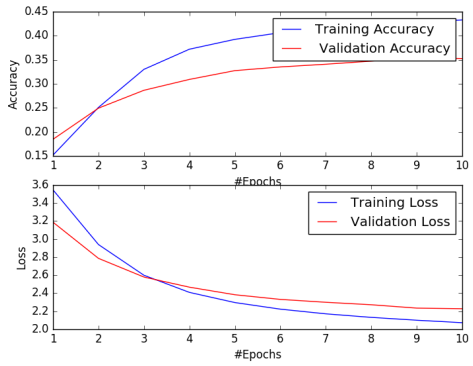


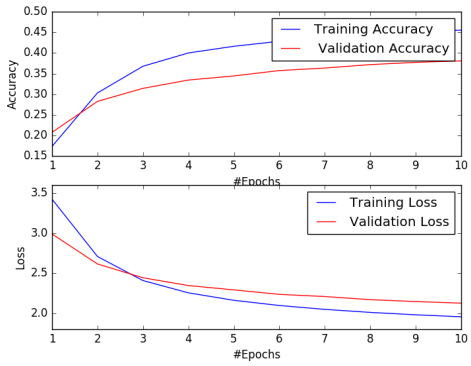
Figure 8: Accuracy and loss of training and test with respect to the number of epochs

For this experiment, the maximum number of epochs was 50, learning rate was 0.001, and the optimizer used was adam. As number of epochs increase, both the training and test accuracies increase and loss decrease. However the rate of increase is greater for first 5 epochs, and then the accuracies increase in a slower rate. This is because in the first few epochs, the network is learning to generalize so the accuracies are low but increasing fast. After those epochs, the network is better at generalizing so the accuracies get better but due to generalization the rate of increase is slower.

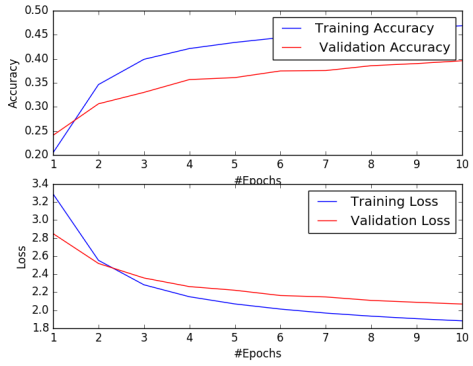
7 Effect of Number of Neurons in the Hidden Layer



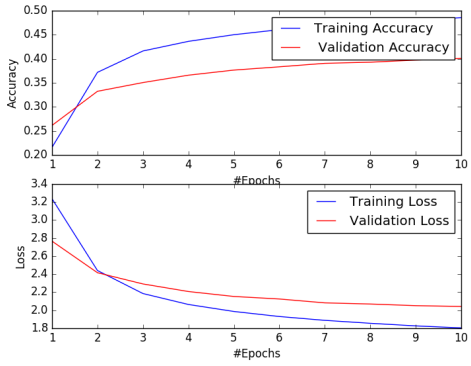
(a) 50 hidden units



(b) 75 hidden units



(c) 100 hidden units



(d) 150 hidden units

Figure 9: Accuracy and loss with varied number of hidden units

No. of Hidden Units	Train Acc(%)	Validation Acc(%)	Train Loss	Validation Loss
50	43.29	35.27	2.07	2.22
75	45.53	38.07	1.95	2.12
100	46.87	38.86	1.88	2.08
150	48.55	40.07	1.80	2.04

Table 2: Experiments with number of hidden units parameter

For this experiment, the number of hidden units was varied from 50 to 75 to 150. The maximum number of epochs was 10, learning rate was 0.001, and the optimizer used was adam. As the number of hidden units increase, the accuracies increase and the loss decrease. At 10th epoch, the test accuracy for 50 hidden units was around 35%, for 75 hidden units, around 38%, and for 150 hidden units, around 40%. When the number of hidden units is increased, the features a network can learn increase as well. Since more features are learned from the inputs, the network is better at generalizing with more accuracy. The plot of 100 hidden units is shown for reference.

8 Effect of Dropout

Dropout is a regularization technique for reducing overfitting in neural network by preventing complex co-adaptations on training set. With dropout, the learned weights of the nodes become

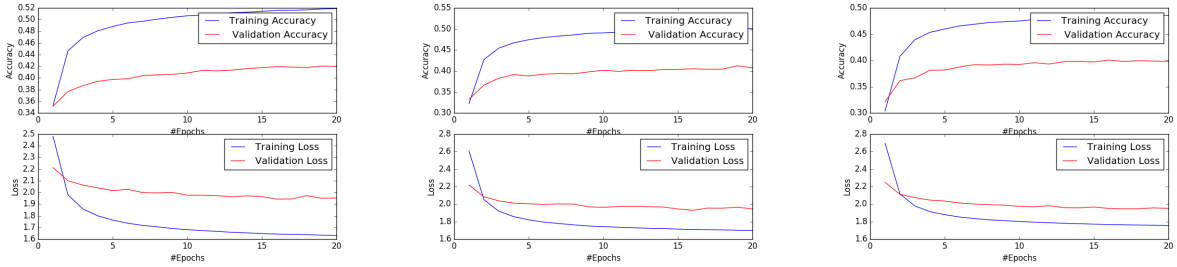
more insensitive to the weights of the other nodes and learn to decide more by their own. In general, dropout helps the network to generalize better and increase accuracy since the influence of a single node is decreased by dropout. On the contrary, we found that increasing dropout did not improve the accuracy in training and validation set. A possible explanation might be that our initial model was able to generalize well on the training data and was not overfitting the data. Hence, the dropout technique was not effective in our case.

In our experiments, the hidden units were dropped with a probability of 0.1, 0.2, 0.3. It can be seen from Figure 10 and Table 3 that training and validation set accuracy decreases as we increase the dropout rate, whereas the training and validation loss increases as we increase the dropout. The model is able to converge faster with increase in dropout rate. Also, it can be seen from Table 3 that the model takes almost similar training time for different values of dropout. In order to measure a significant difference in the training time, the network should be large enough. For larger network dropping out certain chunk will drop out many parameters which were being backpropagated, which may result in lesser updates and thereby reducing the training time for the reduced network. As we are not using large network, we don't see major difference in the training time for different values of dropout.

The music notes in Figure 11 shows parts of notes generated using different dropout. As the value of dropout increases, the regularities in the music notes decreases. For example in Figure11(a), the notes occur at more regular intervals as compared to 11(b). For Figure 11(c), the notes are not regular and there are long pauses of varying lengths.

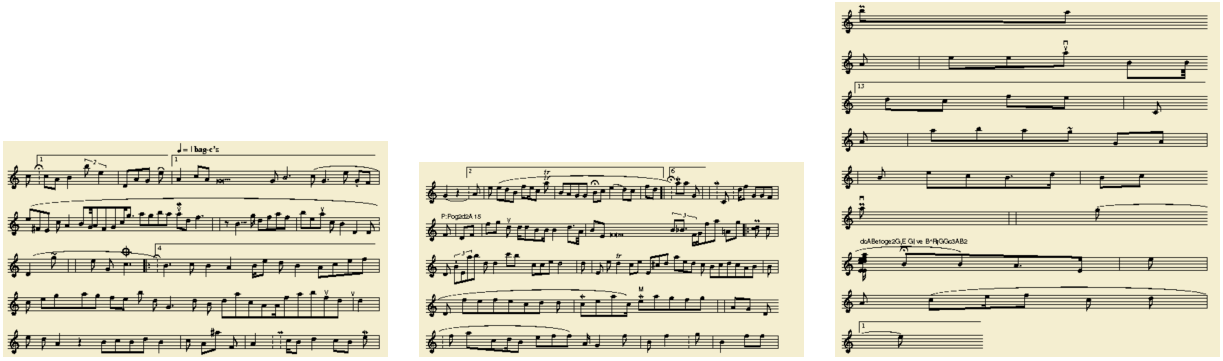
Dropout	**Train Acc(%)**	**Validation Acc(%)**	**Train Loss**	**Validation Loss**	**Time(in sec)**
0.1	51.89	41.98	1.63	1.95	1281
0.2	50.04	40.72	1.70	1.95	1278
0.3	48.58	39.82	1.75	1.96	1283

Table 3: Experiments with dropout parameter. Sequence length = 25, Adam for numerical optimization, $\beta_1 = 0.9, \beta_2 = 0.999, \eta = 10^{-3}$, epoch = 20 and batch size = 64 was used for all experiments.



(a) dropout = 0.1 (b) dropout = 0.2 (c) dropout = 0.3

Figure 10: Experiments with dropout parameter. Sequence length = 25, Adam for numerical optimization, $\beta_1 = 0.9, \beta_2 = 0.999, \eta = 10^{-3}$, epoch = 20 and batch size = 64 was used for all experiments. Figure shows how accuracy and loss changes with change in dropout.



(a) dropout = 0.1 (b) dropout = 0.2 (c) dropout = 0.3

Figure 11: Experiments with dropout parameter. Sequence length = 25, Adam for numerical optimization, $\beta_1 = 0.9, \beta_2 = 0.999, \eta = 10^{-3}$, epoch = 20 and batch size = 64 was used for all experiments. Figure parts of notes generated using different dropout parameter.

9 Effect of different Optimization Techniques

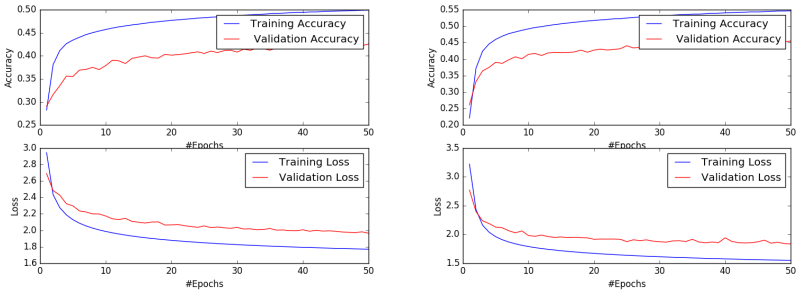
In this experiment we compared the performance of 2 different optimizers namely RMSProp and Adagrad. We ran both optimizers for 50 epochs.

We observed that RMSProp converges much faster than adagrad. This can be seen from the learning rate required for the two algorithms to converge to similar accuracies. In case of RMSProp the learning rate required was 0.001 whereas Adagrad required it to be 0.01. Even then Adagrad was not able to completely match the accuracy achieved by RMSProp. This is because Adagrad accumulates squared gradients over all epochs and normalizes over them. This sum goes on increasing and hence at one point the learning rate becomes extremely low, thus preventing learning. RMSProp on the other hand takes a decaying average over the squared gradients along iterations, leading to a comparatively slower decay of learning rate.

Apart from that both Adagrad and RMSProp keep a separate learning rate for each feature but Adagrad needs an initial period to tune the learning rates to good values so it is quite unstable in the initial stages as visible from the loss and accuracy plots.

Algorithm	LR	Train Acc(%)	Validation Acc(%)	Train Loss	Validation Loss
Adagrad	0.01	49.92	42.57	1.77	1.96
RMSProp	0.001	54.65	45.51	1.55	1.83

Table 4: Experiments with different optimizers. Sequence length = 25, batch size = 64, Epochs = 50 was used for all experiments.



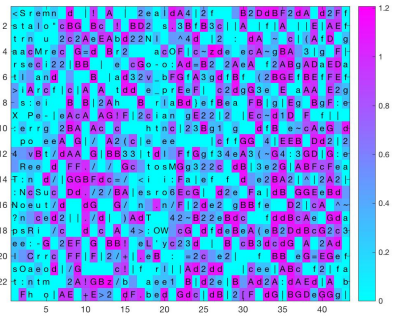
(a) Adagrad (b) RMSProp

Figure 12: Accuracy and loss with different optimizers

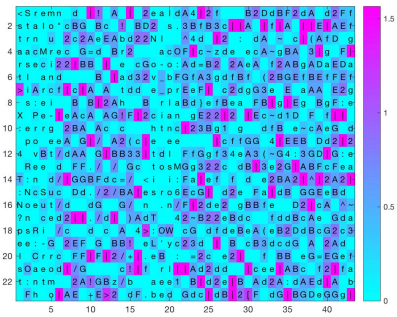
10 Feature Evaluation and Heatmap

In this experiment we generated a sequence of music using our trained RNN model. The sequence was of length 989 characters. We do a forward pass on all the characters in the sequence one-by-one and record the activations of each hidden neuron.

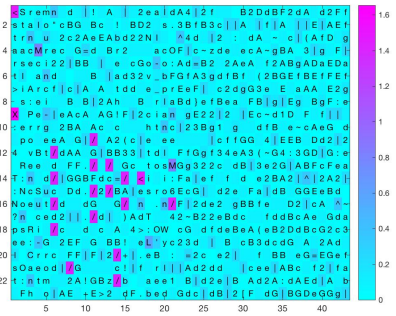
We can clearly see from figure 13 that different neurons activate to identify different parts of the music. Neuron 39 in 13a fires for all kinds of musical notes (A-G,a-g). Neuron 89 in figure 13b activates for bar lines 'l'. Neuron 60 in figure 13c identifies half length notes denoted by '/'. Apart from identifying all musical notes some neurons even activate for specific notes like in Neuron 16 in figure 13d activates specifically for lower 'F' notes (as per the c-major scale).



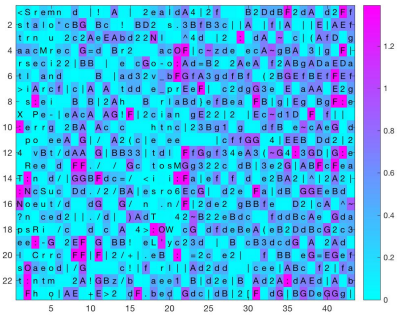
(a) Neuron 39



(b) Neuron 89



(c) Neuron 60



(d) Neuron 16

Figure 13: Activations of different neurons for the same sequence of generated music.

Team Contribution

All the team members wrote the code individually and then the best methods / practices among the group were discussed and submitted for the final report.

11 Appendix

The code below includes all the experiments used.

```
import numpy as np
import pickle
import utility as util
import rnn_model as model
import pdb
from sys import argv
```

```
import argparse

def command_line_initialization():
        ap = argparse.ArgumentParser()
        default_arguments = {'-hid': '100', # number of hidden units
                                                '-af': 'tanh', #
                                                    activation-function
                                                    -> "tanh", "sigmoid
                                                    ", "relu", "
                                                    hard_sigmoid", "
                                                    linear", "softplus"
                                                '-opt': 'adam', #
                                                    optimizer -> "adam",
                                                     "sgd", "rmsp", "
                                                    adagrad"
                                                '-ts': '25', # timesteps
                                                     -> [25-30]
                                                '-bs': '5', # batch-size
                                                     -> [5 - 8] (not
                                                    sure)
                                                '-e': '10', # number of
                                                    epochs
                                                '-T': '1.0', # T -> 0.5,
                                                     1, 2
                                                '-dr': '0.0', # dropout
                                                    -> 0.1, 0.2, 0.3
                                                '-lr': '0.001', #
                                                    learning rate
                                                '-dec': '0.000001', #
                                                    decay parameter
                                                '-b_1': '0.9', # beta_1
                                                    for adam
                                                '-b_2': '0.999', #
                                                    beta_2 for adam
                                                '-m': '0.5', # momentum
                                                    for sgd
                                                '-rho': '0.9', # rho for
                                                     rmsp
                                                '-saveas': 'ts_25'} #
                                                    name of filename to
                                                    save image

        for current_argument in default_arguments:
                ap.add_argument(current_argument, default =
                    default_arguments[current_argument])

        pa = ap.parse_args()
        return int(pa.hid), pa.af, pa.opt, int(pa.ts), int(pa.bs), int(pa
            .e), float(pa.T), float(pa.dr), float(pa.lr), float(pa.dec),
            float(pa.b_1), float(pa.b_2), float(pa.m), float(pa.rho), pa.
            saveas


def load_data():
        char_dict = pickle.load(open('../data/char_dict.p', 'rb'))
        all_music = np.load('../data/all_music.npy')

        all_music_encoded = util.one_hot_encoding(all_music, len(
            char_dict.keys()))

        return char_dict, all_music_encoded

def load_finetune():
        ft = np.load('../data/1hot_tunes.npy')
        #X = ft[:,:-1]
```

```
        #y = ft[:,1:]
        X = []
        y = []

        for f in ft:
                X.append(f[:-1])
                y.append(f[1:])
                #pdb.set_trace()
        return X,y


# Call python file as python main.py -hid <#num hidden units> [so on and
    so forth]

if __name__ == '__main__':

        hidden, af, optimizer, timesteps, batch_size, nb_epochs, T, p,
            l_rate, decay, b_1, b_2, mom, rho, fname =
            command_line_initialization()

        print hidden, af, optimizer, timesteps, batch_size, nb_epochs, T,
             p, l_rate, decay, b_1, b_2, mom, rho, fname

        #load data
        char_dict, all_music_encoded = load_data()

        input_dim = len(char_dict.keys())

        #get model
        rnn = model.build_simplernn_model( input_dim, hidden, af,
            optimizer, T, p, l_rate, decay, b_1, b_2, mom, rho )

        #Train model
        hist = model.train_rnn( rnn, all_music_encoded, timesteps,
            batch_size, nb_epochs )
        util.plot_result( hist.history, fname )

        #Fine tuning

        rnn.load_weights("../model/trained_weights_0d_1t.h5")
        seqX,seqy = load_finetune()
        ft_hist,ft_model = model.finetune_model(rnn,seqX,seqy)
        ft_model.save_weights("../model/ft_weights_0d_1t.h5")

        #Generation of music

        #load trained weights from file to model
        #rnn.load_weights("../model/weights.09-2.05.hdf5")

        #pick a random input pattern as our seed sequence, then generate
            music character by character
        prime_text = '$' #K:F\r\n X:2\r\n'

        generated_music = model.generate_music_soft(rnn, prime_text,
            char_dict)

        text_file = open("../music/music_soft.txt", "w")
        text_file.write("%s" % generated_music)
        text_file.close()

        #plot heatmap for activation of neurons
        model.monitor_neurons(rnn, char_dict)
```

The code below consists of snippet for creating rnn model used in the project.

```
import keras
from keras.layers.recurrent import *
from keras.layers import *
from keras.models import Sequential, Model
from keras.optimizers import *
from keras.callbacks import *
import utility as util
from scipy.io import savemat
import pdb

def get_optimizer(name = 'adagrad', l_rate = 0.0001, dec = 0.0, b_1 =
    0.9, b_2 = 0.999, mom = 0.5, rh = 0.9):
        eps = 1e-8

        adam = Adam(lr = l_rate, beta_1 = b_1, beta_2 = b_2, epsilon =
            eps, decay = dec)
        sgd = SGD(lr = l_rate, momentum = mom, decay = dec, nesterov =
            True)
        rmsp = RMSprop(lr = l_rate, rho = rh, epsilon = eps, decay = dec)
        adagrad = Adagrad(lr = l_rate, epsilon = eps, decay = dec)

        optimizers = {'adam': adam, 'sgd':sgd, 'rmsp': rmsp, 'adagrad':
            adagrad}

        return optimizers[name]

def build_simplernn_model(input_units, hidden_units = 100, af = 'tanh',
    optimizer = 'adam', T = 1, p = 0.0, l_rate = 0.001, dec = 0.0, b_1 =
    0.9, b_2 = 0.999, mom = 0.5, rh = 0.9):
        #define network architecture
        model = Sequential()
        model.add(SimpleRNN(input_dim = input_units, output_dim =
            hidden_units, activation = af, return_sequences = True,
            dropout_U = p))
        model.add(Lambda(lambda x: x * 1.0/ T))
        model.add(Dense(output_dim = input_units, activation = 'softmax')
            )

        #compile model
        opt = get_optimizer(optimizer, l_rate, dec, b_1, b_2, mom, rh)
        model.compile(optimizer=opt, metrics=['accuracy'], loss='
            categorical_crossentropy')

        print model.summary()

        return model

def train_rnn(model, train_data, timesteps = 25, b_size = 5, n_epoch =
    10):
        #lets make length of train data to be a multiple of timesteps
        #don't shuffle, as sequence matters
        l = len(train_data)
        train_data = train_data[:l - l%timesteps,]

        train_data = np.reshape(train_data, (-1, timesteps, len(
            train_data[0])))

        #make length of train_data to be multiple of b_size
        l = len(train_data)
        train_data = train_data[:l - l%b_size]
        print l, len(train_data)
        #Predict the next character in the sequence
        X = train_data[:,:-1,]
        Y = train_data[:,1:,]
```

```
        #setting stateful to true
        model.layers[0].stateful = True
        model.layers[0].batch_input_shape = np.shape(X)

        #Callbacks
        hist = History()
        checkpoint = ModelCheckpoint('../model/weights.{epoch:02d}-{
            val_loss:.2f}.hdf5', monitor='val_loss', verbose=1,
            save_best_only=True, mode='auto', period=1)
        callbacks_list = [hist, checkpoint]


        history = model.fit(X, Y, batch_size=b_size, nb_epoch=n_epoch,
            verbose=1, callbacks=callbacks_list,
                validation_split=0.2, validation_data=None, shuffle=False
                    , class_weight=None, sample_weight=None,
                    initial_epoch=0)

        print hist.history
        return hist

def generate(model, pattern, char_dict, total_chars=1500):
        inv_char_dict = {v:k for k,v in char_dict.iteritems()}
        music_str = pattern

        #setting stateful to true
        model.layers[0].stateful = True

        pattern = [inv_char_dict[c] for c in pattern]
        #generate characters
        for i in range(0, total_chars):
                pattern = util.one_hot_encoding(pattern, len(char_dict.
                    keys()))
                pattern = np.reshape(pattern, (1,)+pattern.shape)
                prediction = model.predict(pattern, verbose = 0)

                index = np.random.choice(range(len(char_dict)), p =
                    prediction[0,np.size(prediction,axis=1)-1,:])

                pattern = index
                music_str += char_dict[index]

        return music_str


def generate_music_soft(model, pattern, char_dict, total_chars = 1500):
    inv_char_dict = {v:k for k,v in char_dict.iteritems()}
    music_str = pattern

    for i in range(total_chars):
        enc_pat = [inv_char_dict[c] for c in pattern]
        oneHot_pattern = util.one_hot_encoding(enc_pat,len(char_dict.keys
            ()))
        prediction = model.predict(np.reshape(oneHot_pattern, (1,) +
            oneHot_pattern.shape),verbose =0)

        index = np.random.choice(range(len(char_dict)), p = prediction[0,
            np.size(prediction,axis=1)-1,:])

        pattern = (char_dict[index])
        music_str = music_str + pattern
        pattern = music_str

    return music_str
```

```
def finetune_model(model,X,y,batchSz=4,epoch = 1,lr=0.0001, dec = 0.0):
        #pdb.set_trace()
        orig_lr = model.optimizer.lr
        orig_dec = model.optimizer.decay
        model.optimizer.lr = lr
        model.optimizer.decay = dec
        model.layers[0].stateful = True
        orig_batch_input_shape = model.layers[0].batch_input_shape
        model.layers[0].batch_input_shape = np.shape(X)
        hist = History()
        pdb.set_trace()

        max_len = 1200
        X = pad_sequences(X, maxlen=max_len)
        y = pad_sequences(y, maxlen=max_len)
        #pdb.set_trace()
        model.fit(X,y,batch_size = batchSz, nb_epoch = epoch, verbose =1,
             callbacks=[hist],
                        validation_split=0.0, validation_data=None,
                            shuffle=True, class_weight=None,
                            sample_weight=None, initial_epoch=0)

        model.optimizer.lr = orig_lr
        model.optimizer.decay = orig_dec
        model.layers[0].batch_input_shape = orig_batch_input_shape
        model.layers[0].stateful = False
        return hist,model


def monitor_neurons(model, char_dict):
    inv_dict = {v: k for k, v in char_dict.iteritems()}

    #model.load_weights('../model/weights_ft.h5')

    hidden_out = model.get_layer('simplernn_1').output
    act_model = Model(input = model.input, output = hidden_out)
    all_activations = np.zeros((100,1))
    gen_sequence = []
    output = open('../data/output_acts.npy','w')
    music = open('../music/music_soft.txt','r')
    music_mat = open('../data/output_acts','w')
    sequence = music.read()
    sequence = sequence.split('\n')
    sequence = ''.join(sequence)
    #generate characters
    for i in range(0, len(sequence)):
        index = inv_dict[sequence[i]]
        pattern = util.one_hot_encoding(index, len(char_dict.keys()))
        pattern = np.reshape(pattern, (1,)+pattern.shape)

        current_activation = np.reshape(act_model.predict(pattern,
            verbose=0),(-1,1))
        if i==0:
            all_activations = current_activation
        else:
            all_activations = np.hstack((all_activations,
                current_activation))

        print char_dict[index],sequence[i]
        #pattern = pattern[1:,]
    np.save(output,(all_activations.reshape(100,-1)))
    savemat(music_mat,mdict={'sequence':np.asarray(list(sequence)),'
        activations':all_activations})
    print all_activations.shape
```

The code below consists of snippet for plotting heatmap.

```
load('output_acts.mat')
for i=1:100
    act = activations(i,:);
    act = reshape(act, [23, 43]);
    f = figure('visible','off');
    imagesc(act),colormap(cool);
    hold on;

    [X,Y] = meshgrid(1:43,1:23);
    X = reshape(X,[43*23,1]);
    Y = reshape(Y,[23*43,1]);
    text(X,Y,sequence,'VerticalAlignment','middle','HorizontalAlignment',
        'center');
    colorbar;
    hold off;
    name = sprintf('./activations/neuron_%d',i);
    saveas(f,name,'jpg');
end
```

The code below includes all the utility functions used.

```
import numpy as np
from keras.utils.np_utils import *
import matplotlib.pyplot as plt

def one_hot_encoding(labels, n_classes):
        return to_categorical(labels, nb_classes=n_classes)

def plot_result( hist, fname ):

        epochs = np.array(range(len(hist['acc']))) + 1


        fig = plt.figure(1)

        plt.subplot(211)
        plt.plot(epochs, hist['acc'], 'b-', label = "Training␣Accuracy")
        plt.plot(epochs, hist['val_acc'], 'r-', label = "␣Validation␣
            Accuracy")
        plt.ylabel('Accuracy')
        plt.xlabel('#Epochs')
        plt.legend()

        plt.subplot(212)
        plt.plot(epochs, hist['loss'], 'b-', label = "Training␣Loss")
        plt.plot(epochs, hist['val_loss'], 'r-', label = "Validation␣Loss
            ")
        plt.legend()
        plt.ylabel('Loss')
        plt.xlabel('#Epochs')

        plt.tight_layout()
        plt.show()

        fig.savefig("../results/" + fname + ".jpg")
```

The code below consists of snippet for reading music data.

```
import numpy as np
import pdb
```

```
import pickle

def convert_chars_nums(abc_input):
        char_dict = {}
        all_music = []

        i = 0
        for char in abc_input:

                if char not in char_dict:
                        char_dict[char] = i
                        i += 1

                all_music.append(char_dict[char])


        inv_char_dict = {v: k for k, v in char_dict.iteritems()}
        return inv_char_dict, np.array(all_music)

def read_file(filename):
        raw_text = open(filename).read()

        #remove <end> tag because we have <start> tag too so <end> tag is
             redundant
        raw_text = raw_text.replace('<end>', '')

        #hard code: replace <start> tag with one character which is not
            used by input music
        raw_text = raw_text.replace('<start>', '$')

        return convert_chars_nums(raw_text)

def read_tunes(filename):
        raw_text = open(filename).read()
        raw_tunes = []
        raw_text = raw_text.split('<end>')
        for tune in raw_text:
                if(tune!=''):
                        tune += '<end>'
                        tune.replace('<start>','%')
                        tune.replace('<end>','$')
                        raw_tunes.append(tune)
        raw_tunes = chars_to_num(raw_tunes)
        return np.asarray(raw_tunes)

def onehot_Enc(data,char_dict):
        oneH_chunks = []
        for chunk in data:
                oneH_chunks.append(to_categorical(chunk,nb_classes=len(
                    char_dict.keys())))
        return np.asarray(oneH_chunks)


if __name__ == '__main__':
        char_dict, all_music = read_file('../data/input.txt')

        np.save('../data/all_music.npy', all_music)
        pickle.dump(char_dict, open('../data/char_dict.p', 'wb'))

        #for finetuning
        raw_tunes = read_tunes('../data/input.txt')
        oneH_tunes = oneHot_Enc(raw_tunes,char_dict)
        np.save('../data/1hot_tunes.npy',oneH_tunes)
```