Music Generation using Character-level Recurrent Neural Networks

Aditya Verma
A53219148
arv018@ucsd.edu

Akshaya Purohit
A53215013
akpurohi@ucsd.edu

Vamshi Gudavarthi
A53216604
vgudavar@ucsd.edu

Rishabh Misra
A53205530
r1misra@ucsd.edu

Chetan Gandotra
A53210397
cgandotr@ucsd.edu

Abstract

This report discusses the fourth programming assignment of our course CSE 253: Neural Networks and Pattern Recognition, i.e., generating music in ABC format using Recurrent Neural Networks (RNN). The report in detail describes about the sub-parts of the assignment like the format of the data available, architecture of the model, the way training data is prepared and fed into the model, experimentation, results, inferences, visualization of the activations of neurons etc. Our main observation was that the RNN generated music that is reasonably good.

1 Introduction

In this report, we train a Recurrent Neural Network (RNN) to learn the structure of music files in ABC format and then we generate a music file based on what the model learns. We experiment with different parameters like dropout, changing number of neurons in the hidden layer, optimizers etc. to see their effect on model's learning behaviour and have reported the same.

2 Training the Recurrent Neural Network

The data is read from the text file which has around 1124 songs (music sequences) each enclosed between start and end tags. This file is given as an input to the python script which reads the whole file and splits 80 % of the list of input and output sequences as training data and rest 20 % as validation data, where each input and output sequence has about 25 characters. Also, there is one to one correspondence between input and output sequence but output sequence is one character offset to the right from the corresponding input sequence. This is because we are training the model to predict the next character, given the current input character and the previous context. There are 93 unique characters in the input data file so each character of the sequence is represented through one-hot encoding in 93 dimensional space. The RNN when unrolled in time will remember the context for about 25 characters. Hence, the RNN network will have an input of size 25*93. Since we are predicting the next character, the output will have the same size as well. The number of neurons in the hidden layer is a variable and we experimented with 50, 75, 100 and 150 neurons. A softmax layer is used for predicting the probabilities of the occurrences of the next character, and the cross entropy loss is used for training. We shuffled the training data to train the network for inputs passed in random order. Over the entire training data the cross-entropy loss is minimized and final set of parameters are used to generate the music.

3 Music Generation

After the network has been trained reasonably well on the training data, we had used a sequence of length 25 which is not in the training data as an input to the network. The output corresponding to each character in the input sequence is a softmax probability distribution over all possible characters. A character is selected each time by sampling from the output corresponding to the last character in the input sequence according to the softmax probability distribution. This is added to the input vector and earliest character was removed to make input sequence have only 25 characters. This is then fed into the network again as the input. In this way music is generated one character at a time, with stopping criteria being setting the limit on the length of the generated text to a predefined value or end tag generated in the process (whichever is earlier) which also stops generating music.

4 Experimentation and Corresponding Inferences

4.1 Effect of Temperature

4.1.1 Introduction

In this section we generated music files using three different temperature values, $T = 1$, $T = 2$ and $T = 0.5$. The music quality and length were studied and inferences were made using the same.

4.1.2 Methodology

The optimizer used here is RMSProp with a learning rate of 0.01 and decay of 0. A batch size of 25 was used for training the RNN. The same will be used for later sections unless specified otherwise. After training the model, we gave a sample start token to the model and let it predict the output. The predicted output was then used as part of input for the next iteration and this was continued till the end token was predicted by the model. The output music generated by the model is in ABC format and it is then converted into midi format using the link provided - ABC Converter.

4.1.3 Results - Temperature = 1

```
<start>
X:70
T:Farandole de Cabasse
O:France
A:Provenddeilan-13
M:C|
K:D
zc/d/|dc B>B B>B|BA A>c|df ~gaf|
c2ff dBGA|BG6 BBc d2e|f2B | B2f2| ^ed Be|
Add BdB|G2A GGF|B,DGF GAFG|G(2D CEF|GFG Bdc|BdB GFG|BGB Fd=c|edAd BGFG|FABG defd|
eg~a2 af gedd|~g3B ||
ded BAG | GEE FDD DG:|BAB BBB|BEB dGB|dfe fed|AG~G2 FGF(d3B|(3deg dB|c4c4||
d] (f2ef | e2efefB3f|
defe efge|df~f2f | z4-| g2b bged | B>B A>Bc2|B2c2 | B2B>G | cGcd | c2 F>G GF | BG "C"G3d | BGFG2G2 | C8 B2 g>e | [bA>B
ce|"G"faec2>e|f2g|fed gef|g3 afa|ffg edc|Bdd ~g3|ggf =def|gfdB BedB|~f3d ~g3f|feab afed|cgAd BGcd|eddd AF~A2|ABGF GBdA|ABGD
FFEF|FGDF G>A dBmA|BGBB dBGB|AFBE GGB,|B,2D G,EC[+B,GB, D,|F,4G,>G>G | FABG | B6 | G2BBB|"G3G B/B/d/|=cd de|c2B2 G2Bc | AGGE/G
///~E/B/B/G/A/G|G3|: cde d>B|BAB D3F | Ged2B2|"C"c>FG | dG|GB | GBGF D2 :\
|
de2c dede|fed def|gfg d2B|cBA Bdf|fed cdB|ABA G3|A2B BGG BGB|ddB B2e||
eed ^fAFA ||
V:2
L:1I
V:F
B2B BcB||
<end>
```

Figure 1: T = 1: First Generated Sample Music ABC Format

```
<start>
X: 14
T:Adie? paure Carnava
Z:Transcrit etgr
M:6/8
Q:B/2
L:1/8
K:G
G2Ac cded|Adff dfLed|eAd c2A|AdA ABA|Adc def|d>d de dB|dc AA|AB Ap | cB d2d2|d) f2ff/d/d/B/ | c/)Bce| de}gf/g/e/d/
fd|"E"Am"B2A2|G3B | d2 cd)d2|dd|d/d/e/d/ de| A/f/g/e/ |1 AB |:"C}"EG/A/|G2 A2 A)A ||
P:Variatomezeraiel Me=he Deedynn Ahnd.
R:Danleely dondegeg Ginegou lackautla cofe d oe dan- elanbm
H:Janamain olsd andansy Go  :aransoine Conddernan
Z:id:hn-polkande
Z:id:hn-slinelef lo la/4
D: Ian maad ondeanee ofdel Bed nnenn 2ne bon- ertananey'a oalende
R:pllole y ene Holl Arableneed ofse.fred ohe dellye Brant plane lacho
Wed
Z:ig:hn-polkaneEnalande
ewele
Z:id:hn-pelec lo Guncenne cona ofec
Z:indlleley:|
<end>
```

Figure 2: T = 1: Second Generated Sample Music ABC Format

Figure 3: T = 1: First Generated Sample Music

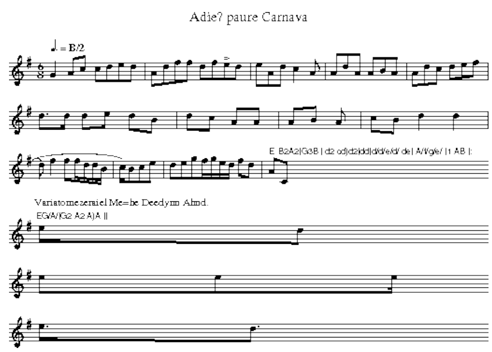

Figure 4: T = 1: Second Generated Sample Music

4.1.4 Results - Temperature = 0.5

```
<start>
X: 11
T:Ungaresca
R:---
C:Pierre Phal?se
C:Trad.
R:jigholka-03
Z:id:hn-slide-2
Z:Pour toute obselvarlathan
Z:id:hn-joloiga
C:Tarntonca-par
Z:id:hn-jig-15
M:C|
K:D
AF|A3B|BBGB BG~A2|AFGA FD|A,2G,2|D,4|G,4 G2Bd | d4 ee|f2 ce|f2 fe||
|: f2e def|ged Bd|
eec edB|ABG GEF|G2B def|ged def|~g3 ged||
<end>
```

Figure 5: T = 0.5: First Generated Sample Music ABC Format

```
<start>
X:70
T:Farandole de Cabasse
O:France
A:Provence
Z:id:hn-slipjig-16
M:2/4
L:1/8
K:F
|: ed/d/e/f/|g/2g/2g/2|fe/f/ de || fefg fefg|afge gedB|BGBB BGAB|BBGB BDGF|ABAG AFFG|G2GF EFFE|FGEF GFGE|~D3F FG~A2|BcBG
GDFG|BAGF GFFG|AGBA BG~A2|AGB BdB|B2B BGA|BAG G2d|B2B B2B|B2B BBA|1 AGF GFF | BGG GFA|Bcd BdB|BdB ~B3|AGF GFA|BAB BdB|BAB GFG|Bdd
B2A|BBG GFG|BGB Bde|f2e d2B|ABG EFD|F2G GFG|BGB Bde|def edB|BBB Bdf|faf g2f|edB B2G|B2B BGA|B2B BAG|GBd BdB|1 B2Bc | de/f/ ge||
<end>
```

Figure 6: T = 0.5: Second Generated Sample Music ABC Format

Figure 7: T = 0.5: First Generated Sample Music

Figure 8: T = 0.5: Second Generated Sample Music

4.1.5 Results - Temperature = 2

```
<start>
X: 11
T:Ungaresca
R:---
C:Pierre Phal?se
C:Tolar]Th2taies e'gr
DED~G3 d2G:| | x6 |egfog2>A|A8eCA ):D!e/e/ cBA|D"FGd__GAegrfe:c
_AB|B//=B/8Ac ug|dgbeBgeB:|q"eua^f e|d^Bf>fp b2ec :d/e?zpjlag=f Ma)fdBB|"D"cde,f>b cecce | G2 E>GD|BF:|2
cBnAGFG1".D,d=B:y"E|AGc/|g/ggat|f4gf|fzc2) b2as|G2dBGB fd=||
c4fg)}d26 |]|
d2e f4^B,DF:|d,^c'bf G3||
A4~f2ccB4^F|a4 |]
V- f|~E  Bc"F3DG|B3[07"c/g/CBA d/Ato9|
f>gG"Fmg||
"=(ed8 ((3eaf cfgdfg |gma}e2d d3B
JAc|r^+DG]:|
<e) suId R55de
qwMvianBWc'Y& HA^m
Q:ZOs
S:Atd's <aRg, F'>hn, Bua m'ye/2uk>DMGn-
O:2_\
T:A/c|a2f0
fd/e/t.!h],g2nd2 aced[M:EE22FF>=B6"F"c2-|!
L:g=fe
B>B BB bt|:|e4G, GG|d2Az:|
|:z4 |1 EBgE b-fa4i2d=~g2A AAd|ed^Gc=frgd|=e2ag a2c^gefec dfe|Bf:|g !c)d=c>-B2|c>AF>B|b/FLAFF|F/E/G,\#G7"GFGc|"D@gac'e4-c2B4B4 |
c2B2B2 :/AF #(3FAf|f2dd ::
3:-duc fd|BCEFG],A6:
W:
GzG|F,G:EF|\FEE AJ~AF|AG2 E4|]x6(] G>2BA  Fd(e/gABc|BB cB|G>d/va|
~b B>erdeed 1k>AF ^cb|
k2ef c2erAeBF|2GBB
BBBG | d2dcB^cB=b|g4(4 aig|4ge Bef|edcB)\'shcdu feaf
g4f+cfb4g3a3B4:T,96
A>GGL'rc'A4G|x6:hs:
&:w,Dst go/Gu'i?'qugie lad ji
R2e|
wgf
```

Figure 9: T = 2: First Generated Sample Music ABC Format

```
<start>
X:55
T:Lei courdello
O:France
A:Provence
R:jpouic
H:g1iq\'esde=(C///A/G,D,7G,>D/F(g:Be|{A:(B\gu|gbgbndz|e3feAB |BBB=Bdfd>c |5G2EG.d^ge|Lf>f cegg:fefg2d|ebg2E j|: A~A2 |
A6-|3B~A4"gecd|edcd egace/A|A>d ^a2d>e L|d/
1/2
f^f.u-n)g
EarA'tW:,2 iGhl[+e2er^nye
A>D|AD~D>|[G>I}>(dd|gf|ggdB|ec8|!AE:F/B/cu G | B6 |deeff d^H~G2:scAF|\
PBA]1 |
Z:|
S:nets"01 Ietmansochampj
G3BBeA ||BmEC,B B,(Dd>f|edG/^bc/e/y|Q2B )dz|":"2-Af gvee|de dgfc|fcbF ge)(3F/=F)!2fgf|VgeB ~f3a:|T:1Z seat/Tr
Riow\aiecq,GF.=n^D)rlsubeor bol-e ZaThaai2nnp]
qedf]C3g3d|a3 (3f:GMB2|^c{e}~d34B4||
Gg:\
A:g2g4e3nBD|"2\g,f6"c|ie-DG/^F)g|Bg-d>c|Mce2|d4A4PG(G9g4c/(9BfGe|g d8f/2g/at ajze dB|(3~AmGc ABG,2Eb, | T
Gfug fe|B/=e/cFgAGddB:B4f|
B6 Boc2:|E2A[E4].w. J.Drie|{e}eGGF Bc :B>c fF|D>B/Bd|B <a\g:|d~G2.F3F2:|Tm(]\de/d|d2BWAG4G3 :C,2 G,OB,,}C,>G/2-,|
F>"D|qb2|1 cg,fz2ed |   F4 :(AcA=zG}^ FTGeGar|(^2=e ddccf | d3-b'faff|b2=f 4a>f | d, c2|
~e4efFgg2ba4|f>db3fB | e/2f ~D3c|~\7A de/efff |1 S Aed  eFDB2 EFG|[T2dfedd!e2gff e2Bc | A>dcB dFF> :\
a2{:b/[3/2^f/>"g/ef/)A c2B|GBcB G vF|[BcBBFc
```

Figure 10: T = 2: Second Generated Sample Music ABC Format

4.1.6 Discussion

As can be noted from figures 1 to 10, the music file generated for lower temperatures is closer to the input music files. The music generated for $T = 0.5$ seems to be more informative than for $T = 1$. This shows that the model learns to predict better with lower temperature values. Lower temperature will make the difference in probability values more distinguishable than with higher temperature and hence, the predicted character is better for lower temperature. However, if we temperature value is set to be too small, it would probably become biased towards some particular characters and will not give good performance.

From figures 9 and 10 we can see that the output music file generated in the ABC format is not very informative. This results in this case are not getting converted into midi format. This is expected because with higher temperature value the output probability distribution got flattened. When a character sample was taken using the flattened distribution the predicted character was more equally likely to picked from any of the possible characters and thus, is less informative. We reaffirmed our hypothesis by comparing our output with that of BeerMind which generates random and non-comprehend-able reviews at $T = 2$.

4.2 Plot of Training Loss and Validation Loss v/s Number of Epochs

In Fig. 11 plot of training and validation loss is shown. This is obtained using 100 hidden neurons, zero dropout and RMSprop optimizer. The learning process has almost converged within 10 epochs with final training loss settling to 1.64 and validation loss settling to 1.8. The fluctuations in the loss for validation dataset seems to die down towards the end of 10 epochs giving us an indication of the convergence of the learning process. A learning rate of 0.01 was used with 0 decay.

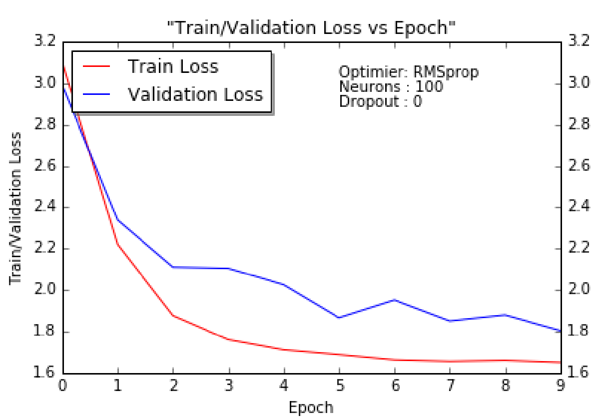

Figure 11: Training and Validation loss vs number of epochs

4.3 Effect of number of hidden neurons on Training and Validation loss

4.3.1 Introduction

In this section we will describe in detail the inferences we can draw from our experiments with changing number of hidden neurons.

4.3.2 Methodology

The models were retrained from scratch for this experimentation part by changing the number of neurons in the hidden layer of RNN.

4.3.3 Results

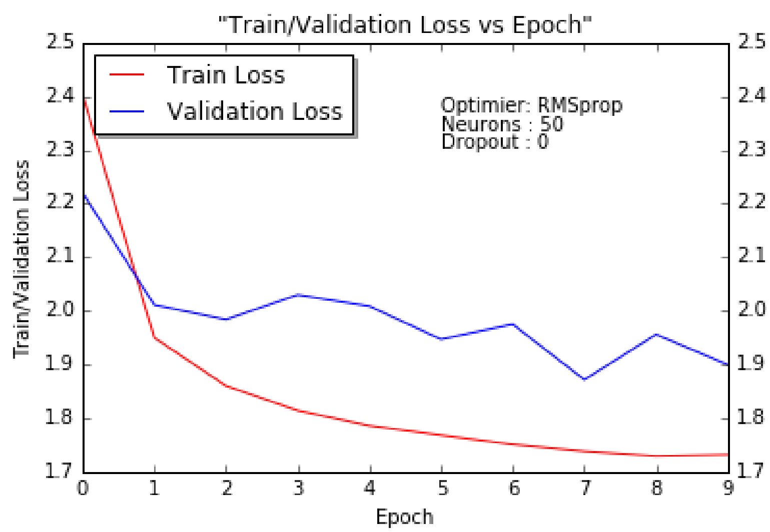

Figure 12: Training and Validation loss vs number of epochs for 50 Hidden Neurons

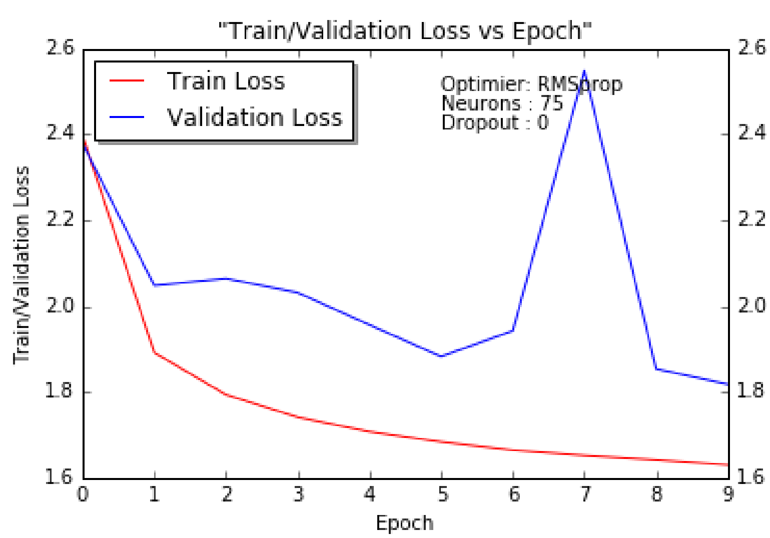

Figure 13: Training and Validation loss vs number of epochs for 75 Hidden Neurons

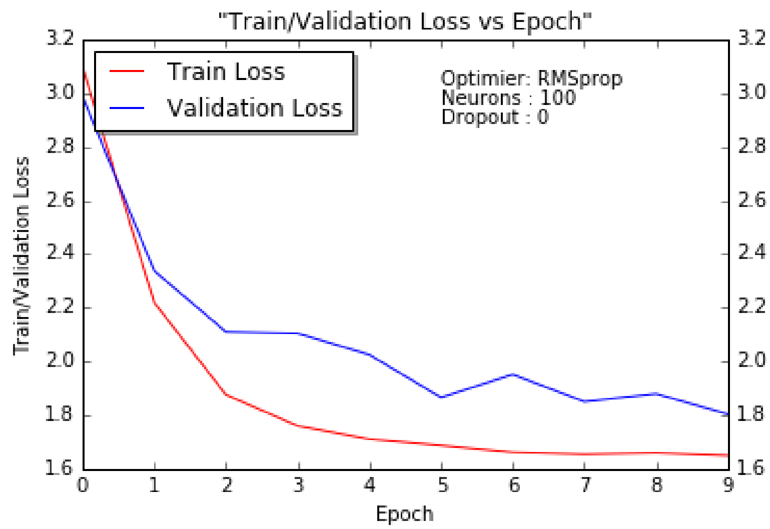

Figure 14: Training and Validation loss vs number of epochs for 100 Hidden Neurons

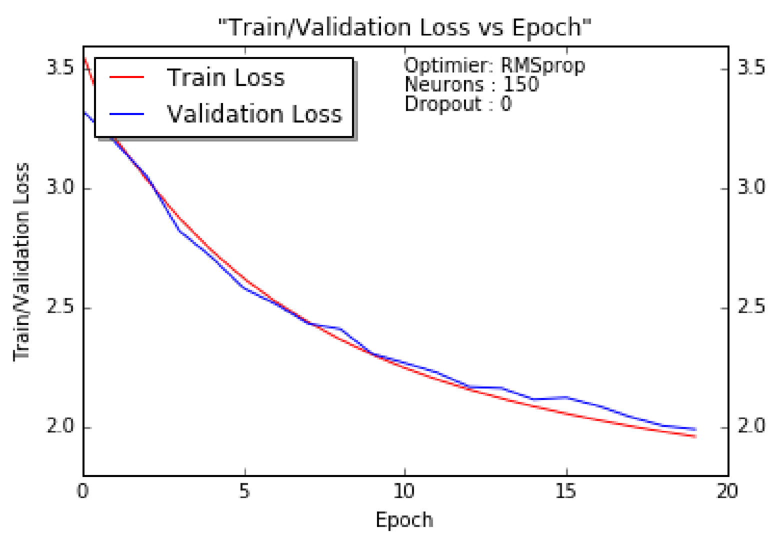

Figure 15: Training and Validation loss vs number of epochs for 150 Hidden Neurons

4.3.4 Discussion

As we increase the number of hidden neurons from 50 in Fig.12 to 100 in Fig.14, the loss decreases but further increasing it to 150 hidden neurons (Fig.15)the loss increased showing signs of over-fitting. For 150 hidden neurons the learning process with $learningrate = 0.01$ was not satisfactory, hence we changed the learning rate to 0.0001. This also shows the necessity of using dropout as described in the next section. Note that for comparative study, the number of epochs in each were set to be the same (ten).

4.4 Effect of the optimizer on the model performance

4.4.1 Introduction

In this section we describe the performance of different optimizers like RMSprop and Adagrad.

4.4.2 Methodology

The model was compiled and trained from scratch after changing the optimizer. Up till this point, we had been using only the RMSprop optimizer.

4.4.3 Results - Using RMSprop

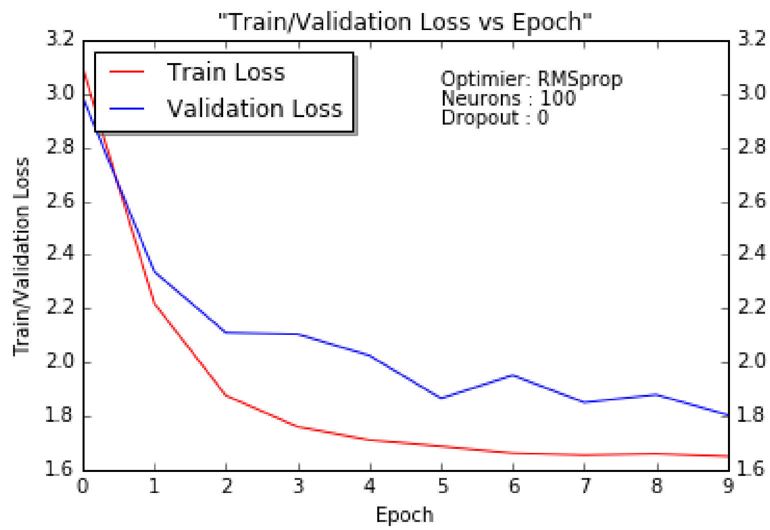

Figure 16: Training and Validation loss vs number of epochs using RMS-prop optimizer

4.4.4 Results - Using Adagrad

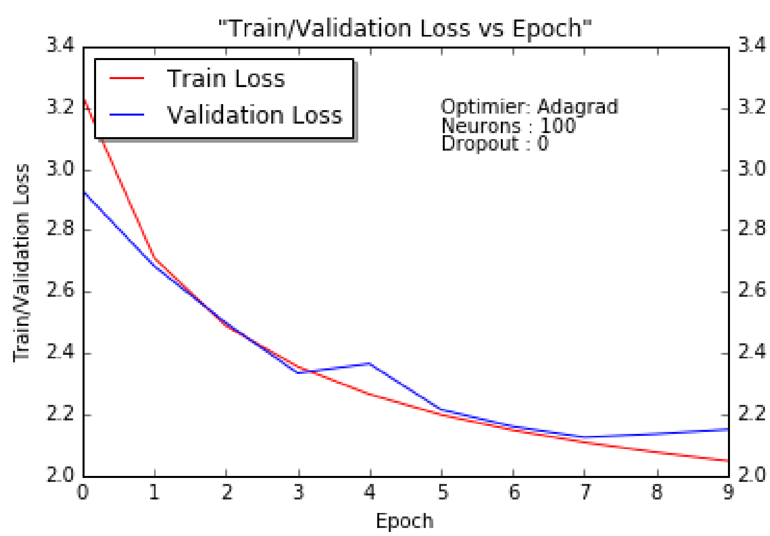

Figure 17: Training and Validation loss vs number of epochs using Adagrad optimizer

4.4.5 Discussion

Fig.16 shows the results obtained with RMSprop and Fig.17 shows the results obtained with Adagrad. From the plots we can conclude that loss curve is steeper in case of RMSprop than in Adagrad. Also, the final loss is lower in RMSprop than in Adagrad. This is expected because RMSprop was developed to solve the radically diminishing learning rates problem present with Adagrad optimizer and hence, RMSprop converges to the minima quickly. Correspondingly, the music generated in case of RMSprop was better. To generate music of comparable quality with Adagrad, we had to train the network for longer and let it reach a training loss value that was comparable to that of RMSprop. Midi files for music generated from both Adagrad and RMSprop have been attached.

4.5 Dropout

4.5.1 Introduction

In this subsection, we observe both the qualitative as well as quantitative effects of dropping out certain fraction of hidden layer neurons randomly during the training time. This allows the rest of the neurons to be more independent and robust, thus (ideally) improving the performance up to a certain extent.

4.5.2 Methodology

We experiment with dropout values of 0, 0.05, 0.1, 0.2, and 0.3 and analyze the results. For comparative study, we choose the learning rate to be 0.01, number of neurons to be 100, optimizer as RMSprop and decay of 0.02. Even though these parameters are not tuned to give the best results, it doesn't matter for this section as we need to analyze the relative performance.

4.5.3 Results

Dropping out more neurons has been seen to increase the training time. The average training time without any dropout is 95s, with 0.1 dropout it's 100s, with 0.2 dropout it's 160s and with 0.3 it's 165s.

Plots for training loss and validation loss for each of the dropout configuration are as follows:

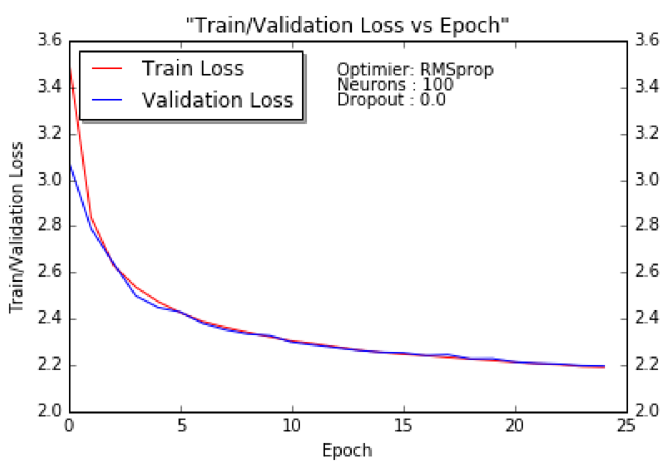

Figure 18: Training and Validation loss vs number of epochs for 0 dropout

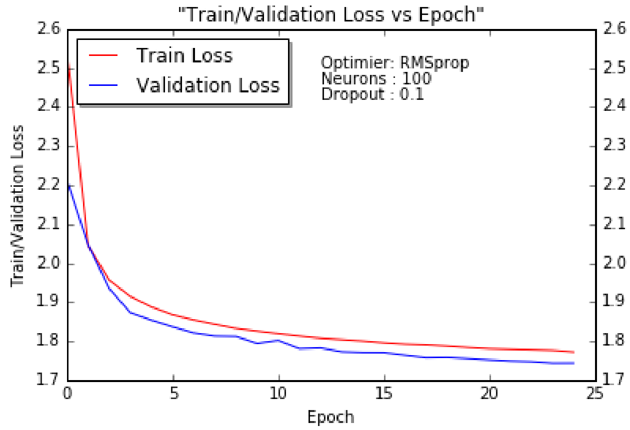

Figure 19: Training and Validation loss vs number of epochs for 0.1 dropout

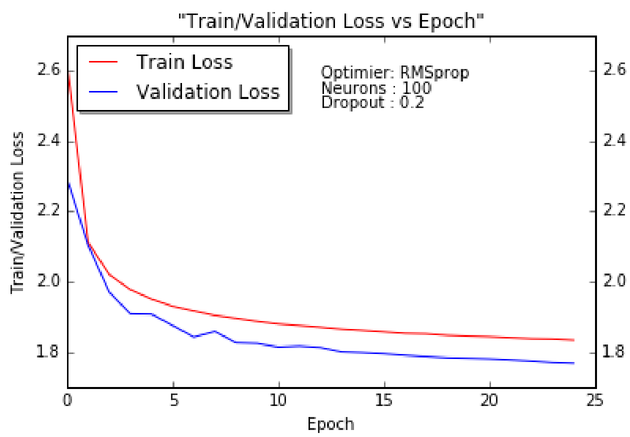

Figure 20: Training and Validation loss vs number of epochs for 0.2 dropout

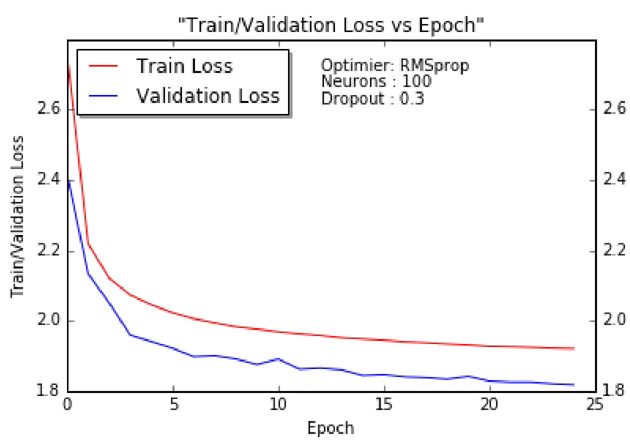

Figure 21: Training and Validation loss vs number of epochs for 0.3 dropout

Music generated from these experiments with different dropout fractions is as shown below:

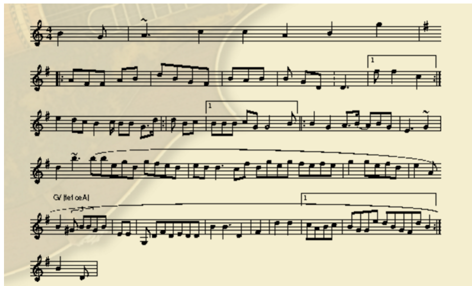

Figure 22: Music notes generated with 0 dropout

```
<start>
X: 14
T:Adie? paure Carnava
Z:Transcrit et+on
Z:Trhn-edehe an BEC F2 G, L:j/8/8
K:hn-s in-.
Z:id:FE/ c/2A/2| ge cgd|e =cg | fed|afc'fgf/fd | d3
| B2 z/2A/ : 24| ed/)g | efe dcD | gFf|
DdB|g2Aef|gfefee|ggf |dgfa feg| ef BE G |EFD G2 G2A
D DFFE|BF FAFA|
B2G|~A3c2c2A2 B2g2 |
K:Gd A f2_ GAGA:|2 FAB2 c2F2|1]dBAAB cFBGAd|
|:AFF AB|dBGF |BAB|B GD:D3|1 f f2c2:|
e2dc B2c/ BB G>d |:|d Bc |1 BBB cG G4B||:B2G2 A>B -
ABG | E3 ~G2 |
d2~b3 (bbegd gfed|ed d3cf dgeg fefe|g2gefd|e2A "G/
|fef ceA|
B2 (3^Gn BBGB |EE|G, DFDEDD |B2B3 D2| F2 | d2d2 |1
cABc|BGGe dBGFd B2:|
B2D|
<end>
```

Figure 23: ABC format notes generated with 0 dropout

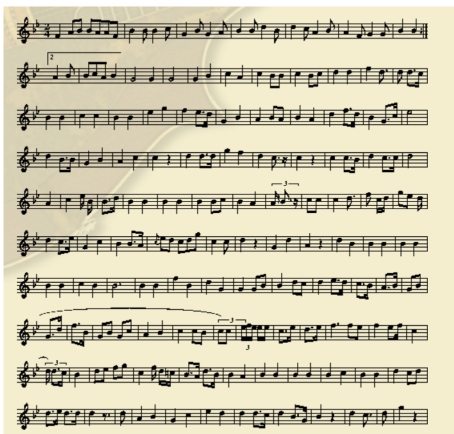

Figure 24: Music notes generated with 0.1 dropout

```
<start>
X: 14
T:Adie? paure Carnava
Z:Transcrit et/ou corrig? par Michel BELLON -
2005-03-04
Z:Pour toute observetion mailto:galouvielle@free.fr
M:2/4
L:1/8
K:Gm
F2AB BAAF| B2B B2c | G2B G2A | B2B d2B | d2c A2B |
A2F G2G | B2 B2 :|2 A2B | BcA B2 | G4 G2 | G4 | G2
B2 | c2 A2 | c2 BB | c2 Bc | d2 d2 | f2 d> | d2 d>c
| B2B2 | c2c2 | B2 B2 | e2 g2 | f2 e>d | G2 B2 | A2
BA | B2 A2 | d2 f>d | B2 G>c | e4 | d2 B>B | G2 B2 |
A2 c2 | c2 z2 | d2 d>d | g2 f2 | d2 c>z | c2 z2 | c2
c>B | d2c>c | d4 | A2 | c2 e/ B/ | B>d | B2 B2 | B2
B2 | B2 Bc | B2 A2 | (3/2A/2 B>z | cc | c2 d> | f2
c/d/ | ge d/ | d2 c>e | G2c2 | B2 B>A | {c}}ed cdg |
c2c | d2z2 | G2d2 | A2z2 | d2B2 | B2B2 | B2B2 | B2B2
| c2B2 | B6 | B2 B2 | f2 B2 | d2 G2 | G2 GB | B2 dc
| d2 e>d | c>B | A>B | GB |  G>d | f>B | GB Gc | A2
B2 | c2 cB | (3c)c| (3e/f/e/ e/2e/2 | c>e | d>e | f3
fe | f2 ef/ | c2 | (3d/) d>c | B2 | de fg | c2 f/
d/2=c/2 | B>A d>B | B2A2 | B2 B2 | B2 B2 | B2 c2 |
B2 B2 | d2 cd | d>e d>d | d2 z> | d2 | A2 B2 | G2 c2
| e2 d2 | d2 d>c | d2 B>G | B2 z2 | d2 c> | d2 |
g2z2 |
<end>
```

Figure 25: ABC format notes generated with 0.1 dropout

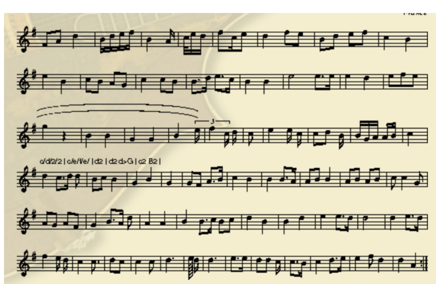

Figure 26: Music notes generated with 0.2 dropout

```
<start>
X:70
T:Farandole de Cabasse
O:France
A:Provinse
Z:id:hn-herley Mallan Rerither se che the lae Harlet
pure ch cherthe Mic ate She the The Tha sore erore
thurolse tre the surels ela Soan ando tond (1919)
Z:id:hn-realar athourioriraitio seriethon corrige
~a3y| cdd fed frassine (1975)
Z:id:hn-heranan
Z:id:hn-ellef-07-05-07
M:|/8
L:1/8
K:G
FA d2 | B/d/e/f/ | B2A/2 | c/2/e/d/ | fc/2e | d2 fe
| Bd ef | c2 B2 | e2 B2 | cB AG | c2 cB | B>d e>c |
B2B2 | e4 e>e | e2 d2 |  efe | g2 z2 | B2 B2 | G2 G2
| B2 (3/e/)) | f2 c/ d/2 | c | e2 e/ c/ | cd B/ |
B/G/A/ A/2B/2 | c2 | "c/d/2/2 | c/e/f/e/ | d2 | d2
d>G | c2 B2 |
d2 c>d  d | Bc B2 | G2 B2 | G2 | GF A>B | c2 B2 | B2
B>A | AB B2 | AB AB | c c2G | AG AF | G2 B>A d | A2
A2 | B2 B>c Bc | d2B2 | B2d2 | e2 e>d | d4 | d2 e>d
| f2 e/ d/ | c2 c> | d2c | c2c | d3 | e/2/2/2/ d/2 |
d3 | e>e | dd c/ | c>B | c2 d>e | e2f d2 | d2 A2 :|
<end>
```

Figure 27: ABC format notes generated with 0.2 dropout

4.5.4 Discussion

From the experimentation, we notice that the time taken for convergence increases as the fraction of neurons dropped increase. This could be because in each iteration our program has to randomly choose which neurons to drop and this adds an extra overhead. Also, the reduced number of parameters take longer to train (compared to a full network).

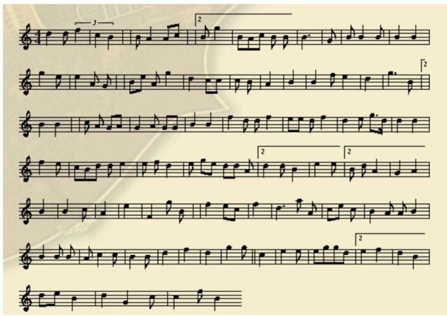

Figure 28: Music notes generated with 0.3 dropout

```
<start>
X:20
T:La festo vierginenco
T:Introduciounlkilloor:s
l:o
N:-ain
r:-oreeso
t:ns ooreltote cou d iosr ga tors eiutt ue toui mos
hos easau el e r: ton thi iou hon ssirelnirlonn
L:1/8
K:no42 A2 | G2 B2 | |2 e cc | c3   Ac | B2 2 A
|2Bc   c2 | B2A d2 | e3 d2 | d2   | A2  a e | e2 | c2
| B2 2 A| |eA d2 | c c2 | d2 B e | d2 2 | B2 | c e
d2 | e  B e B2 | A2 | | a2 c A | ce e | f2 c | e2 e
A2 | E2e c fe fe | d2e d2 | 2BB | A2 | B2 | A2 f e |
d2 e c2 | |A F2 A | A2 | G A2 | d2 A A B2B | F d B
A2 B2 | A d2 | e2B | B2 d | A2 | d2 | d2 g dcdf |
ffee ffef| e2f| |
d2 d (3f2 | c2 B2 | |BA2 Ac | |2 B g2 | BAc  B B |
B3 2  | G| B B2 B | B2   B2 | g2e  | e2A  G | | Be A
g2 | d2   cc| c B | A2 | B2 | B2 f e2 | d2 | g3  B
|2 | B2 B2 | |c A GA |G2 A  GG | B2  B2 | f2 d d f2
| ee d f2 | d2e g>d | d2 d2 | f2e | cBd d e2| e f d2
|e ge dd A |2d2 d B2 | e2 2 e |2B d A2 | G2 A2 | B2
| B2c | A2  | e2 | F2 g B | | f2 ec | f2  | d3 a A |
ce c | B2 A A B2 | B2 B B | A c2c | B2B | d2 | f2|
d2 | g2 g ||c2| e2e |eggd |2e2 f2 | d2 2 B2 |
de B2 | d2  G2c | c2f B2
<end>
```

Figure 29: ABC format notes generated with 0.3 dropout

In terms of quantitative measure (loss value), we see (from Fig.18, 19, 20, and 21) that as we increase the dropout fraction, the loss decreases up to a certain point (which is 0.1 in our case) and further increase in dropout fraction does not decrease the loss any further. Instead, it starts to increase again. The reason is that dropping certain fraction of the neurons make others to be more independent in predicting the output thus avoiding over-fitting. However, by dropping neurons further, we are restricting the learning capability of our network. This result is in line with previous experimentation where the network with 100 neurons perform better than network with 50 neurons as in the latter case they just aren't enough neuron to learn the complex relationship between input and output.

The qualitative aspect of dropout could be gauged from Fig.22, 24, 26, and 28. When the dropout is 0, though the music generated is decent, the music notes are somewhat sparse in some places and dense in other. Also, there seems to be some repetition of notes. For dropout fraction 0.1, we see that we have gotten an elaborate tune with a lots of variation and music notes appear to be more regular (or there's less aberrant behavior). Music quality also seemed to have improved. Between 0.1 and 0.2 dropout fraction, there doesn't seem to be much difference in terms of output quality. There are sufficient variations in this case as well. However, for dropout fraction 0.3, the music quality seems to have deteriorated relatively. The music notes are regular though there does not seem to be much variation. The repetition of notes can also be noticed. Thus, empirically, it is safe to say that dropout helps only when applied in the correct amount.

4.6 Feature Evaluation

4.6.1 Introduction

In this subsection, we use one of our generated music sample and analyzed the output of the activations of hidden neurons by producing its heat-map.

4.6.2 Methodology

For the purpose of plotting the heat-maps, we trained our RNN on the training data for 50 epochs using the RMSprop optimizer with learning rate 0.01 and decay 0.02. Then, we used the weights of the trained model to build a truncated model which doesn't have the final output layer. This allowed us to capture the output on each hidden neuron when the input was fed to it. For generating the heat-map, we chose one of our generated music samples and fed one character at a time into the truncated model to record the output at each of the neurons. In results section, we show a few outputs which seems to be doing something understandable.

4.6.3 Results

The activations for few of the neurons at each character of the generated music sample is as shown below:

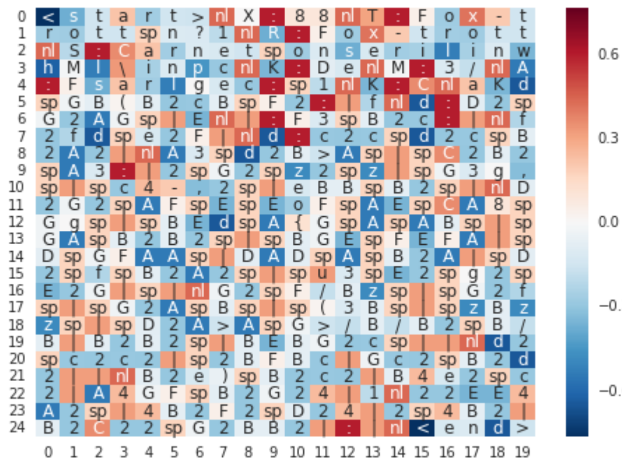

Figure 30: 12th neuron detecting colon

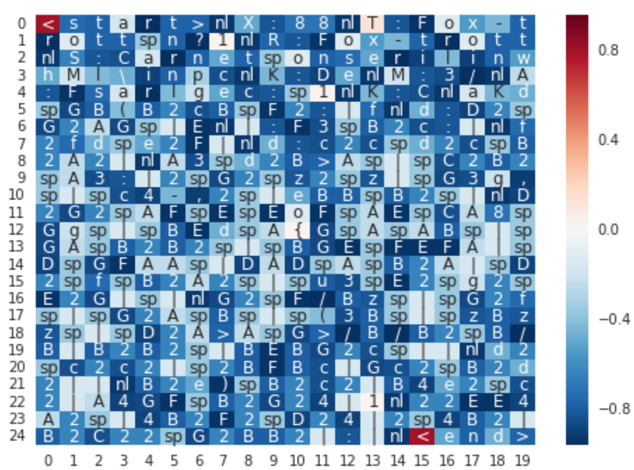

Figure 31: 15th neuron detecting opening angular bracket

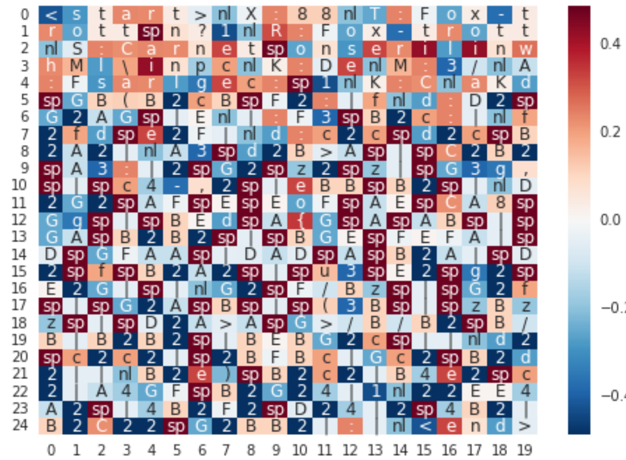

Figure 32: 3rd neuron detecting space

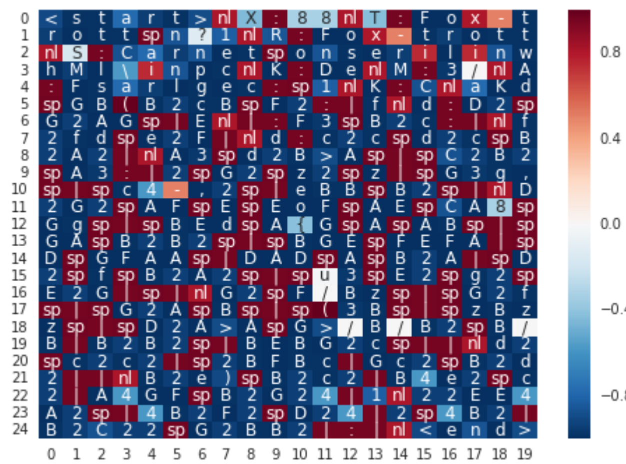

Figure 33: 11th neuron detecting delimiters like space, newline, colon, pipe

4.6.4 Discussion

In RNN, for predicting and generating sequence, each of the neuron learns to do a specific task over time. In this section, we seek to understand which neuron serves what purpose. To that end, we recorded the activations of 4 neurons (3rd, 11th, 12th, and 15th) and visualize their activations when fed with the one of the generated music samples. In figure 30, neuron seem to be detecting colon which is a separator in the header of the music file. In figure 31, the neuron seem to be sensitive to opening angular bracket which is actually an indication of start or end of the music. In figure 32, the neuron seem to be detecting space characters. At last, in Figure 33, the neuron seem to detect various delimiters/separators like new line, space, colon, and pipe.

5 RESULTS AND LEARNINGS

This programming assignment gave us a good insight into Recurrent Neural Networks and their working. We were able to generate music of good quality using this assignment and it shows the strength of RNNs. A number of other inferences were made, such as those related to effect of temperature, dropout, number of neurons in the hidden layer and the effect of optimizers. A loss value close to 1.6 was obtained in around 10 iterations with 100 neurons in the hidden layer and RMSprop optimizer.

As temperature grows, the music generated becomes more random and less reliable. In fact, no satisfactory music was generated at T=2. Dropout proved out to be an asset as it improves the music quality substantially. This can be attributed to the fact that it regularizes the input and makes sure that the predictions are not an over-fitted version of the input. Also, we noticed that RMSprop performed better than Adagrad and had lower loss value for the same number of iterations. Also, we reaffirmed our hypothesis that more neurons in general help us achieve less loss. In case the number of neurons is too large (around 150 in this case), the number of parameters become huge and the learning rate needs to be lowered so that they can be assigned appropriate weights that help convergence. The convergence may take longer in this case. Lastly, the loss value for both training and validations sets decrease over time and stabilize pretty quickly with a learning rate of around 0.01.

6 TEAM MEMBER CONTRIBUTION

6.1 Aditya Raj Verma

1. Reading-in data, conversion to input to one-hot form and splitting data into training-validation sets
2. Basic implementation of Simple RNN model in Keras
3. Preparing report for the above point(s)

6.2 Chetan Gandotra

1. Added code to predict the next character in sequence
2. Music generation at different temperatures (T = 0.5, 1, 2)
3. Preparing report for the above point(s)

6.3 Vamshi Gudavarthi

1. Tuning of hyper-parameters
2. Added code for plotting graphs and for the Adagrad optimizer
3. Preparing report for the above point(s)

6.4 Rishabh Misra

1. Added code for dropout functionality and experimented with different dropout configurations
2. Implementation of heatmap generation functionality with Akshaya
3. Preparing report for the above point(s)

6.5 Akshaya Purohit

1. Music generation at varying number of neurons
2. Implementation of heatmap generation functionality with Rishabh and analysis of activations at various neurons
3. Preparing report for the above point(s)

This is just a high level attribution of the contributions. Everyone was involved in all the aspects of the assignment (coding, debugging, analysis of generated samples, experimentation etc.) and added their perspective, which led to successful completion of the assignment. This was made possible because everyone decided to sit down together and help each other out with their tasks.

7 APPENDIX

7.1 Code for Generating Music files by varying parameters - RNN.py

```
# In[ ]:

import numpy as np

from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt

from keras.models import Sequential
from keras.layers import Dense,SimpleRNN,Dropout
from keras.optimizers import RMSprop, Adagrad

# In[ ]:
def generateData(path_to_dataset='input.txt',batch_Size=25):

    print('Loading Data ..............................................\n')

    # Create List of Unique Characters in the Music
    fHandle = open('input.txt')
    text = fHandle.read()
    chars=sorted(list(set(text)))
    print('Number of Different Characters in Music:\t',len(chars))
    split_lines = text.split("<end>\n")
    split_result = ['{}{}'.format(a,'<end>\n') for a in split_lines]
    fHandle.close()

    # Create index number for all the characters
    char_indices = dict((c, i) for i, c in enumerate(chars))
    indices_char = dict((i, c) for i, c in enumerate(chars))

    # Create training Data X and Y
    sentences = [];     next_chars = [];
    for i in range(len(split_result)):
        text = split_result[i]
        for j in range(len(text) batch_Size 1):
            sentences.append(text[j:j+batch_Size])
            next_chars.append(text[j+batch_Size])

    print('Total number of batches: \t',len(sentences))

    print('Vectorization .............')
    X = np.zeros((len(sentences), batch_Size, len(chars)), dtype=np.bool)
    y = np.zeros((len(sentences), len(chars)), dtype=np.bool)
    for i, sentence in enumerate(sentences):
        for t, char in enumerate(sentence):
            X[i, t, char_indices[char]] = 1
        y[i,char_indices[next_chars[i]]] = 1;

    [X_train, X_test, y_train, y_test] = train_test_split(X, y, test_size=0.2,
                                                          random_state=1)

    print('Number of Training Examples: \t',X.shape[0])
    print('Number of Test Examples: \t',X_test.shape[0])

    print('\nComplete.')
    return(X_train,y_train,X_test,y_test,char_indices,indices_char, len(chars),
           split_result)


# In[ ]:
```

```
def buildModel(batch_Size, uniqueChar, nHiddenNeuron=100, percentDropout=0,
               optimizerUsed='RMSprop'):
    print('\nBuilding model.......................................')
    model = Sequential()
    model.add(SimpleRNN(nHiddenNeuron, input_shape=(batch_Size, uniqueChar),
                        return_sequences=False))
    model.add(Dropout(percentDropout))
    model.add(Dense(uniqueChar, activation='softmax'))

    if(optimizerUsed == 'RMSprop'):
        model.compile(loss='categorical_crossentropy',
                      optimizer=RMSprop(lr=0.01, decay=0.01), metrics=['acc'])
    if(optimizerUsed == 'Adagrad'):
        model.compile(loss='categorical_crossentropy',
                      optimizer=Adagrad(lr=0.01, epsilon=1e 05, decay=0.0),
                        metrics=['acc'])

    print('Dropout Percentage: ', percentDropout, '%')
    print('Optimizer Used: ', optimizerUsed)
    print('Complete.')
    model.summary()
    return(model)

# In[ ]:
def generateSequence(fHandle, model, batch_Size, uniqueChar, seedIndex,
                     char_indices, indices_char, temp, maxLength,
                     split_sequence, count):

    seedSentence = split_sequence[seedIndex 1]
    seedSentence = seedSentence[0:batch_Size]
    generatedSequence = seedSentence

    fHandle.write(str(count)+'. \n\n')
    fHandle.write('Temperature: '+str(temp)+'\n')
    fHandle.write('Seed Sentence: '+str(seedSentence)+'\n\n')
    for i in range(maxLength):
        if(seedSentence[batch_Size 5:batch_Size] == '<end>'):
            break
        predict_next_char = predictNextChar(model, batch_Size, uniqueChar,
                                            seedSentence, char_indices,
                                            indices_char, temp);
        generatedSequence = generatedSequence + predict_next_char
        seedSentence = seedSentence[1:] + predict_next_char
    fHandle.write('Generated Sequence: \n'+str(generatedSequence)+'\n\n\n')

def predictNextChar(model, batch_Size, uniqueChar, sentence,
                    char_indices, indices_char, temp):
    X = np.zeros((1, batch_Size, uniqueChar))

    for i, c in enumerate(sentence):
        X[0, i, char_indices[c]] = 1

    pred = model.predict(X, verbose = 0)[0]
    preds = np.asarray(pred).astype('float64')
    preds = np.log(preds) / temp
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    probas = np.random.multinomial(1, preds, 1)
    char_predict = indices_char[np.argmax(probas)]
```

```python
    return(char_predict)

# In[ ]:

def plotGraph(history, percentDropout, nHiddenNeuron,optimizerUsed):
    plt.plot(history.history['loss'],'r ', label='Train Loss')
    plt.plot(history.history['val_loss'],'b ', label='Validation Loss')
    plt.tick_params(labelright = True)
    plt.title('"Train/Validation Loss vs Epoch"')
    plt.ylabel('Train/Validation Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train Loss', 'Validation Loss'], loc='upper left',
               shadow=True)

    xCoord = int(0.5*len(history.history['loss']));
    ran = (max(history.history['loss']+history.history['val_loss'])
           min(history.history['loss']+history.history['val_loss']))
    st = min(history.history['loss']+history.history['val_loss'])

    plt.text(xCoord,st+ran*0.85, 'Dropout : '+str(percentDropout))
    plt.text(xCoord,st+ran*0.9,'Neurons : '+str(nHiddenNeuron))
    plt.text(xCoord,st+ran*0.95, 'Optimier: '+optimizerUsed )

    fileName = ('trainPlot_Dropout_'+str(percentDropout)
                +'_Neuron_'+str(nHiddenNeuron)+'_'+optimizerUsed)
    plt.savefig(fileName)
    plt.show()

# # Load Data

# In[ ]:
np.random.seed(1)
batch_Size = 50
[X_train,y_train,X_test,y_test,char_indices,indices_char,
 uniqueChar,split_sequence] = generateData('input.txt',batch_Size)

# # Initialize Model

# In[ ]:

nHiddenNeuron = 100
percentDropout = 0
optimizerList = ['RMSprop','Adagrad']
optimizerUsed = optimizerList[0]

model = buildModel(batch_Size,uniqueChar,nHiddenNeuron,
                   percentDropout,optimizerUsed)

# # Train Model

# In[ ]:

history = model.fit(X_train,y_train, batch_size=1024, nb_epoch=45,
                    verbose=1,validation_data=(X_test, y_test))
plotGraph(history, percentDropout, nHiddenNeuron,optimizerUsed)

# # Generate Music

# In[ ]:
```

```
temp = 0.5;
tempList = [0.5]#[0.5, 1, 2]
maxLength = 1000
seedIndex = [12,15,21,71,89,53, 55, 22, 42,11,8,1,2,3,4,5,6,7]
count = 1

fHandle = open('GeneratedMusic.txt','w')
for temp in tempList:
    for i in range(100):
        generateSequence(fHandle,model,batch_Size,uniqueChar,i,#seedIndex[i],
                         char_indices,indices_char, temp,maxLength,
                         split_sequence,count)
        count = count+1
fHandle.close()

print('Music Generated in File: GeneratedMusic.txt')
```

7.2 Code for Heat-map Generation - heatmap.py

```
# In[ ]:

import numpy as np
import seaborn as sns; sns.set()

from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
import pylab as pl
from keras.models import Sequential
from keras.layers import Dense,SimpleRNN,Dropout
from keras.optimizers import RMSprop, Adagrad

# In[ ]:
def generateData(path_to_dataset='input.txt',batch_Size=25):

    print('Loading Data ............................................\n')

    # Create List of Unique Characters in the Music
    fHandle = open('input.txt')
    text = fHandle.read()
    chars=sorted(list(set(text)))
    print('Number of Different Characters in Music:\t',len(chars))
    split_lines = text.split("<end>\n")
    split_result = ['{}{}'.format(a,'<end>\n') for a in split_lines]
    fHandle.close()

    # Create index number for all the characters
    char_indices = dict((c, i) for i, c in enumerate(chars))
    indices_char = dict((i, c) for i, c in enumerate(chars))

    # Create training Data X and Y
    sentences = [];    next_chars = [];
    for i in range(len(split_result)):
        text = split_result[i]
        for j in range(len(text) batch_Size 1):
            sentences.append(text[j:j+batch_Size])
            next_chars.append(text[j+batch_Size])

    print('Total number of batches: \t',len(sentences))
```

```
    print('Vectorization.............')
    X = np.zeros((len(sentences), batch_Size, len(chars)), dtype=np.bool)
    y = np.zeros((len(sentences), len(chars)), dtype=np.bool)
    for i, sentence in enumerate(sentences):
        for t, char in enumerate(sentence):
            X[i, t, char_indices[char]] = 1
        y[i,char_indices[next_chars[i]]] = 1;

    [X_train, X_test, y_train, y_test] = train_test_split(X, y, test_size=0.2,
                                                         random_state=1)

    print('Number of Training Examples: \t',X.shape[0])
    print('Number of Test Examples: \t',X_test.shape[0])

    print('\nComplete.')
    return(X_train,y_train,X_test,y_test,char_indices,indices_char, len(chars),
           split_result)


def readDataFromGeneratedMusic(path_to_dataset,batch_Size, char_indices,
                               indices_char, uniqueChar):

    print('Loading Data ..........................................\n')

    # Create List of Unique Characters in the Music
    fHandle = open(path_to_dataset)
    text = fHandle.read()
    split_lines = text.split("<end>\n")
    split_result = ['{}{}'.format(a,'<end>\n') for a in split_lines]
    fHandle.close()


    # Create training Data X and Y
    sentences = [];     next_chars = [];
    for i in range(len(split_result)):
        text = split_result[i]
        for j in range(len(text) batch_Size 1):
            sentences.append(text[j:j+batch_Size])
            next_chars.append(text[j+batch_Size])

    print('Total number of batches: \t',len(sentences))

    print('Vectorization.............')
    X = np.zeros((len(sentences), batch_Size, uniqueChar), dtype=np.bool)
    y = np.zeros((len(sentences), uniqueChar), dtype=np.bool)
    for i, sentence in enumerate(sentences):
        for t, char in enumerate(sentence):
            X[i, t, char_indices[char]] = 1
        y[i,char_indices[next_chars[i]]] = 1;

    print('Number of Training Examples: \t',X.shape[0])
    print('Number of Test Examples: \t',X_test.shape[0])

    print('\nComplete.')
    return(X,y,char_indices,indices_char, uniqueChar,
           split_result)

# In[ ]:
```

```
def buildModel(batch_Size,uniqueChar,nHiddenNeuron=100,percentDropout=0,
               optimizerUsed='RMSprop'):
    print('\nBuilding model.......................................')
    model = Sequential()
    model.add(SimpleRNN(nHiddenNeuron,input_shape=(batch_Size, uniqueChar),
                        return_sequences=False))
    model.add(Dropout(percentDropout))
    model.add(Dense(uniqueChar,activation='softmax'))

    if(optimizerUsed == 'RMSprop'):
        model.compile(loss='categorical_crossentropy',
                      optimizer=RMSprop(lr=0.01,decay=0.02),metrics=['acc'])
    if(optimizerUsed == 'Adagrad'):
        model.compile(loss='categorical_crossentropy',
                      optimizer=Adagrad(lr=0.01, epsilon=1e 08, decay=0.02),
                        metrics=['acc'])

    print('Dropout Percentage: ',percentDropout,'%')
    print('Optimizer Used: ',optimizerUsed)
    print('Complete.')
    model.summary()
    return(model)

# In[ ]:
def buildTruncatedModel(original_model, batch_Size,uniqueChar,
                        nHiddenNeuron=100):
    print('\nBuilding model.......................................')
    model = Sequential()
    model.add(SimpleRNN(nHiddenNeuron,input_shape=(batch_Size, uniqueChar),
                        weights = original_model.layers[0].get_weights(),
                        return_sequences=False))
    model.compile(loss='categorical_crossentropy',
                      optimizer=RMSprop(lr=0.01,decay=0))
#     model.summary()
    return(model)

# In[ ]:
def reshape_into_3d(data):
    return np.reshape(data, (1,data.shape[0], data.shape[1]))

def reshape_into_2d(data):
    return np.reshape(data, (1,data.shape[0]))


## Load Data
# In[ ]:
np.random.seed(1)
batch_Size = 50
[X_train,y_train,X_test,y_test,char_indices,indices_char,
 uniqueChar,split_sequence] = generateData('input.txt',batch_Size)

# # Initialize Model

# In[ ]:

nHiddenNeuron = 100
percentDropout = 0
optimizerList = ['RMSprop','Adagrad']
optimizerUsed = optimizerList[0]
```

```
model = buildModel(batch_Size, uniqueChar, nHiddenNeuron,
                   percentDropout, optimizerUsed)

# # Train Model
# In[ ]:

history = model.fit(X_train, y_train, batch_size=1024, nb_epoch=25,
                    verbose=1, validation_data=(X_test, y_test))

## Generating heat map
# In[ ]:
trunc_model = buildTruncatedModel(model, 1, uniqueChar, nHiddenNeuron)

np.random.seed(1)
batch_Size = 1
[X_train_t, y_train_t, char_indices, indices_char,
 uniqueChar, split_sequence] = readDataFromGeneratedMusic('generated_input.txt',
                             batch_Size, char_indices, indices_char, uniqueChar)

inputs = []
outputs = []

for n in range(nHiddenNeuron):
    for i in range(len(X_train_t) 4):
        char=indices_char[np.argmax(X_train_t[i].flatten().astype(int))];
        if char=='\n':
            char='nl'
        elif char==' ':
            char='sp'
        elif char=='\t':
            char='tb'
        elif char=='\r':
            char='rt'

        inputs.append(char)
        outputs.append(trunc_model.predict(reshape_into_3d(X_train_t[i]))
        .flatten()[n])
        sns.heatmap(np.array(outputs).reshape((25,20)),
                    annot=np.array(inputs).reshape((25,20)), fmt='',
                    cmap='RdBu_r')
```