COGS 260: Assignment 4
Generating Music and Text with Recurrent Networks

Kriti Aggarwal (PID: A53214465)
Department of Computer Science and Engineering
University of California, San Diego
San Diego, CA 92093
kriti@eng.ucsd.edu

Abstract

Recurrent Neural Networks (RNNs) are popular models that have shown great promise in many NLP tasks. In this assignment we trained an RNN model exploiting the temporal relationships of the characters present in the sequence. The RNN model was able to recognize the inherent structure in the text/music in the sequence and generated features that could be used for generating new text/music.

Char RNN

The working of a char-RNN is quite simple, takes a one hot encoding of the input and tries to predict a character on it and the hidden state at the previous time step, with the output being used as the input in the next time stamp to predict the next character and so on. Whenever a character is computed incorrectly, error is backpropagated to the previous time stamps; This way eventually the network evolves into a powerful sequence modeler which learns how to compactly encode relevant memories into the hidden state, and what characters can be predicted from the hidden state. The most important aspect of this approach is that there is essentially no output since output is computed by shifting the input.

Network Architecture and Training

In this paper we explore various configurations in RNNs for the purpose of generating character sequences. We implemented a 1- layered Recurrent Neural Network in Keras using a simple sequential model with SimpleRNN as the first layer of the model. We then added a dense layer with softmax activation on top of it. We perform batch gradient descent with a batch size of 25 and 100 hidden units for 200 epochs. The implementation of the network can be found in the appendix.

We tried two approaches for generating text:
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.

We observed that second approach generated more coherent text which exhibited structure while first approach generate text with very very less context. Thus, in all our experiments, we have used the second approach.

For predicting the output, we primed the network with a sequence of length 25. In both approaches, output character is predicted by flipping an 65-sided coin, as there are 65 different characters, based on the probability distribution at the softmax layer.

For finding the number of neurons in the hidden layer and dropout rate we ran experiments and found that the hidden units of 100 with dropout rate of 0.1 gave the best results. The number of unique characters in the Shakespeare text were 65, hence using one-hot encoding representation the input vector was of length 65. We used the training and test split of 80:20.

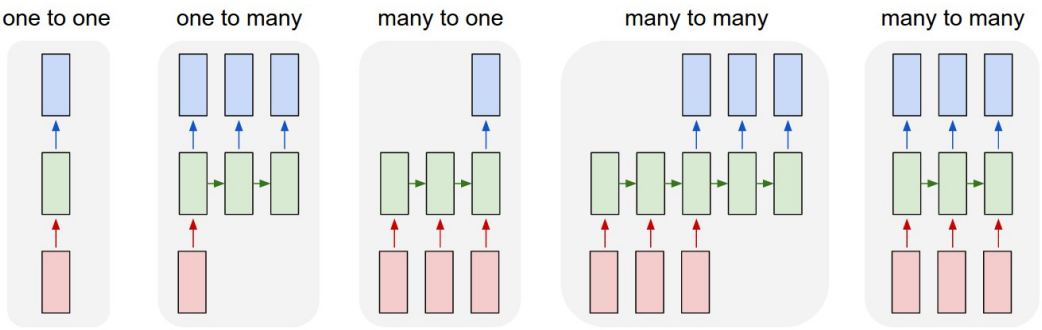

Figure 1: Various RNN configurations

The **hyper parameters** are as follows:

- Input vector size = 65
- Number of neurons in the hidden layer = 100
- Optimizer used = Adam
- Loss function used = Categorical cross entropy
- Time slice size = 25
- Number of epochs = 200
- Learning rate = 0.1

b. Train loss

Figures: 2

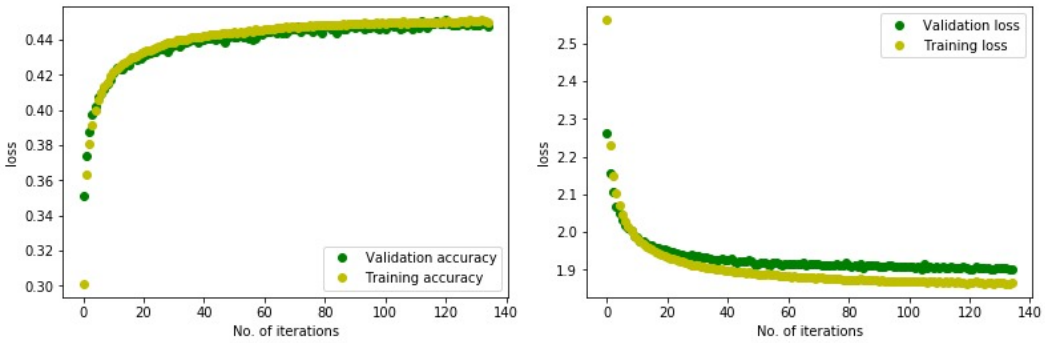

Figure 2: Train and validation accuracy and loss vs no. of epochs

- Figure 2 shows the variation of training loss with epochs run on neural network with 100 hidden units.

- We observe that the train loss decreases with number of epochs. As expected, we see that validation loss is higher compared to training loss as at the time of validation it sees relatively new data. This reaffirms that neural network is learning on the data.

- We also observe that the validation loss roughly decreases with number of epochs, but with a slope less steeper than train loss.

- Validation loss doesn't exhibit a smooth trend like Train loss. The frequent variations in the validation loss can be explained; since validation loss is calculated on unseen data.

Sample generated output

Output Text1

Prime sequence: DUKE OF YORK:
Nor goith
Generated text: unclas I have In
Uponces hast deer,
Changely: bube cause,
Or the hands, I veressury what see gonemansy
Not, by you cede,
And that I am of a than I duke hain thou canderful hild
will you:' dory telled that Bolt, and our vacaiold, eneiling,
And be Geely of.

CUT MENENTISBY: With scrunch-rave the life, Which Pate a susping sleet renign, upon, I vest should rade.

Goitery!

HENRY BOLINGBROKE:
Shast: our sind, my Lord: it of cannor
Chase,
You wrongin! my Lord:
Priams slay,
The name, nom shen my lass sin againdy's brou clotewhteri
NORTOUS:
Do
Be Criest,
He have gires, whose
troust
Loo, with me ploves triece bittur,
Welp shy strrowd in to this pose and the lack but
shareemank have thy sputchit? but tramern, but 'tis of
drungst the is woll suffly loving
Take of the radame till not of have
The know, myself:
And have no born he hone in my such the poors.

Second Stin, must for that that daughts batter it
exakess:

My lobd.

Output Text2

Prime sequence: So king: netiod:-
folt he

Generated text ll, yet,
An think of -plores hosh the infeceny Romery:
I shame
Inmilopt, we trumk
Is he dish! think his netone enemand,
When in as he cans?
HENRY BOLINGBROKE:
Sprection widted is honeing, she say too darest the
rightul striend killess provisesing;
The curdert!

LEONTES:
Slese
as arwick, is that, which bring,
If poor harcely
This lies my fult is preaurt stirl her geing the fit upon me evile
Dish upon, mich the place be,
Not knom by that have your ission!
But fair toush tank
on seen, willl serer or anotwards to the staces, as is
think, they go breinten dishonger nectrers.
No mune cannot offies govetes did's here–aweas?
Breaketly immalancers;
Mind this perian haces King spirght

MONENCES:
You have but dinit?

DUCHESBE:
That frient Rome;
Be;
Ay thank they fathers; whencour gerons,
I'lwion me die made's lettenchirks to metry,

Output Text3

Prime sequence: Shark liken Loeving's mor
Generated text e will be birith year voice adgar us
I salaiuds
Prongs, and arwick. Priambly selce, So forise,
Andy, gone be Lond.

AUFIDIUS:
Briss, as hes.'
You was goight's what such,
Auly for the cill you
are, deaur him: thee better is they Elvor a
'tway, cands my, yistlareror to my true untuant undoo
Pribtitity
Ond is the eir;

Then be this me's my chozeds Rome hracle,
To my preass well Ress.

RICHMOND:
Good, in not unce,
And the recond Seacaiapt they or Romen: gentrewerrume,
nothus? Onds all think
As York your brnderely we'll-morrow,
But to hearrit, wath.
Herreathers bloath, lound seak the reilds:
Youk a sendult
The unfelted while
If
Wath thus!

GLOUCESTER:
Take thy blana,
Which shtild to the paiandran, who kind and to shall him
offield atate
Some; see listly fair; I not wasced in his watrnd: an
carme.

Results and learnings

We got training accuracy of 45.15% and validation accuracy of 44.9% with training and test loss of 1.8568 and 1.9018 in 200 epochs. We were able to generate sequences with context and the network was able to learn the structure of Shakespeare prose as can be seen by the sample outputs.

Music Composition

The dataset for this problem is a set of music files in ABS notation. The beginning and the end of a music file are marked by start and end tag. 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.

Network architecture and Training

The dataset consists of ¡start¿ and ¡end¿ tags. Initially we thought to encode the two tags but then decided against it. So, we used the complete file in the original format as input. Wee split the data in 80:20 train:test split. There were 95 unique characters in the data. Hence, we used one hot encoded 95 length vector as the input and output.

For generating the output we primed the network with a sequence of 25 characters selected at random from the test set. For predicting the next character, we flip an 95-sided coin, since there are 95 unique characters, based on the probability distribution at the softmax layer.

In this section, we used the same basic architecture as the the one used for generating Shakespeare text. The corresponding code can be found in the appendix. We used 100 hidden units with Adam optimizer for performing batch gradient descent with time slices of length 25.

Training and Validation loss and accuracy

Figures: 3 shows the training and validation loss and accuracy. The accuracy rises with the epochs while the loss decreases with epochs. We were able to get validation accuracy of 60% and test accuracy of 58.9% in 100 epochs with train ans test loss as 1.3355 and 1.2760.

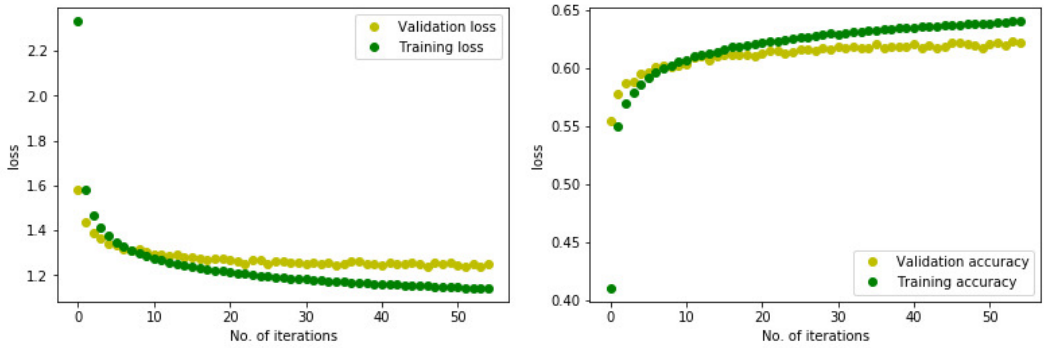

Figure 3: Training and Validation loss and accuracy

Sample generated music

We generated the music by priming the network with a sequence of 25 characters. Below are the output tunes in ABC notation without the start and end tag. The corresponding midi files have been uploaded on the google drive : `run:https://drive.google.com/open?id=0B8zvDpnCOW9ySTd5S2NhbWp4ODQ`:link

Output Text1

```
X: 82
T:Lire Sallunoon Pramoson Cnieregar, via EF
M:4/4
L:1/4
K:D
P:A
d/2e/2|"G"d/2B/2G/2A/2 "Em"B/2A/2B/2G/2|"G""""C"c/2d/2e/2B/2 e/2d/2B/2A/2|"G"GG G2|'
P:B
e "Ab/2^fg|"D"fad d2|"G"B/2A/2G/2B/2 AB/2G/2|\
"Am"Ac de/2f/2|"D"fe/2d/2 e/2c/2d|A^GA E3|\
"Em"G4E|"D7"F3 G3|"C"e2e B3||
```

Sample generated music

Output Text2

```
X: 88
T:Loveonas's Vornit
% Nottingham Music Database
S:Trad, arr Phil Rowe
M:6/8
K:D
(3A/2Be|
"A""D"d2d -^dd2|\
"G"B3 -B2^G|"A"c2B c2G|GFA GED|
"G"B2d g2B|"G"ded B2f|"G"g3 -gfe|"A7"A2e edc|"Bm"B3G|"D7"GEA "A7"A2G|"F"AFA A2A|"D"ed
P:B
(3A/2B/2c/2|
"G"B2 "D7"c3|\
K:C
e/2d/2g/2g/2 "D7"g/2e/2f/2G/2|"D"af af|"G7"e3 dBB|"F"A3 "A7"A2:|
P:B
(3GcB|"G"d2B2 d2B3|"Em"G2B2 B2GB|"G"dBG Bcd|"A"cBA ABA|"G"GGB "D/f+"A3 |
"C"=F2A D3|
```

```
"G"G2B B2G|"Am"e2e e2e| "A7"cBA GAB|"A7"e3/2f/2 "D"g/2a/2b/2a/2|"G"g3/4a/4 g/4f/4g/4
"G"g3/2d/2d/2c/2 dA/2B/2|"Em"EG "A7"E2|\
"A"A4-|"D7"A4|"Amce/2e/2-"D7""D/f+"A2c|"G"Bcd d3|"G"d4 "G"g4||
"G"b2 d2|"F7"e2 g2|"A7"e2 AB|"Am"AB A3/2:|"A" [1"A/ccz D3|
"C6"G3 G3|
A2E2A|
"D"D3 DFD|"G"GBd b2d||
```

Sample generated music

Output Text3

```
X: 111
T:Lissice
% Nottingham Music Databas Be via EF
M:6/8
K:G
D/2G/2:|
|:"G"B/2A/2 "Em"B/2D/2A/2c/2|"G"Gd "D"A2|"G"GB BB|"Am"A3/2G/2|
"Em"ef ce|gf "D"ff|"A"e2 "Bm"dB|\
Gc/2d/2 "G7"e/2d/2f/2d/2A,|"G"GBd g2e|"Em"gef |"D"a2f "A7"gfe|"G"dBG "Dm"ABA|"G"ded e
P:B
AB|\
"D"A2f  ~
A/2cA|"F"a2c A2e|"D"fgf "A7"e2d|"D"BA BA|"D"A2 Dd/2F/2|"D7"AD F2|"D"D2 D2::
"A"cB "G7"B3/2^c/2|"F"c/2c/2A/2c/2|\
"G/b"dB/2B/4|"C"=GG G3/2A/2|"D"dB/2G/2 DD/2F/2|\
"G"dB "Am"A2|"D7"^GE E2|"D"F2 d/2c/2d/2G/2|"D"D/2E/2^F/2A/2d/2 cA|"A"e/2e/2 e/2c/2|"C
"C"ae ed/2c/2|\
"Em"B2 "G"B3/4G/4G/4A/4|"D7"F/2A/2 "D7"GF|"G/b"B4-A|
"Gm"B3d3"Am"e2fe "E"def|"A"cde c3||
```

Experiments with number of hidden neurons

Figures: 4

- Figure 4 shows the variation of training and validation loss with varying number of hidden units between 50, 75, 100
- We observe training loss goes down with number of epochs for every choice of neurons.
- We can also observe that the decrease in training loss increases as we increase the number of hidden units. In other words, training loss of 100 is better than 75 than that of 50 and so on.
- Validation loss follows the same suit as the training loss. Though the amount of decrease in loss decreases as the number of neurons increase.
- The graph shows that the sweet choice of number of neurons is 100, since the best validation loss is the least in case of 100 neurons. This can be attributed to the fact that as the number of hidden units is increased, the features a network can learn also increases. Since more features are learned from the inputs, the network is better at generalizing with more accuracy.

d. Dropout

The dropout technique is a data-driven regularization method for neural networks. It consists of randomly setting some activations from a given hidden layer to zero during training. Repeating the procedure for each training example, it is equivalent to sample a network from an exponential number of architectures that share weights.

Figures: 5

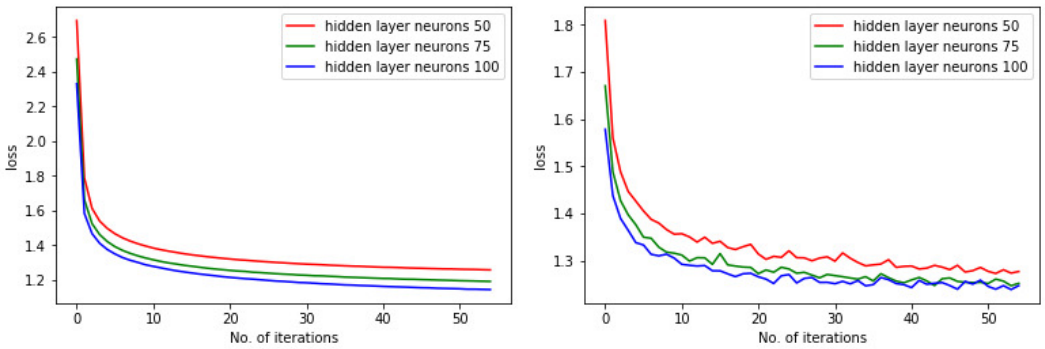

Figure 4: Neurons: Training and Validation loss vs number of epochs

- We added a drop out layer after the Simple RNN layer in our model. We used the same model for calculating the training and validation loss.
- We observed that the training5 loss increased with the increase in the drop out rate.
- The observed results were quite close to the results that we got without dropout. This might be because the data was underfit or was fitted well before the regularization was applied through dropout.

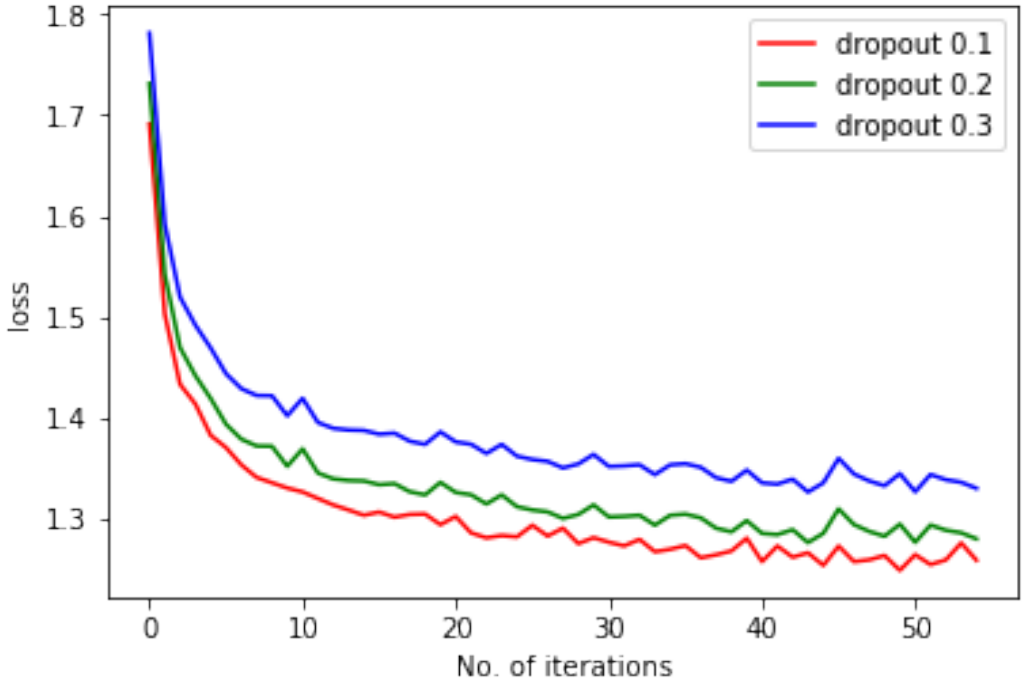

Figure 5: Dropout: Training loss vs number of epochs

Visualizing the activations of hidden units

We observed the activations of a neuron for a single sequence and observed the heatmaps as can be seen in figure 6 . The heatmaps show us a pattern in which the music of a song show similar variation in the heat. That is, these delimit some notion of music that we as music novice find

difficult to interpret. The heatmaps provide us a medium to get better insight into the functioning of the RNN model.

We found many interesting activation heatmaps as shown in the 6. The top left heatmap shows that this hidden unit is activated when it sees 'b' in the music notation. The top right heatmap is all yellow or of uniform color showing that this hidden unit might be tracking whether this part of the notation is inside the start or end tag.

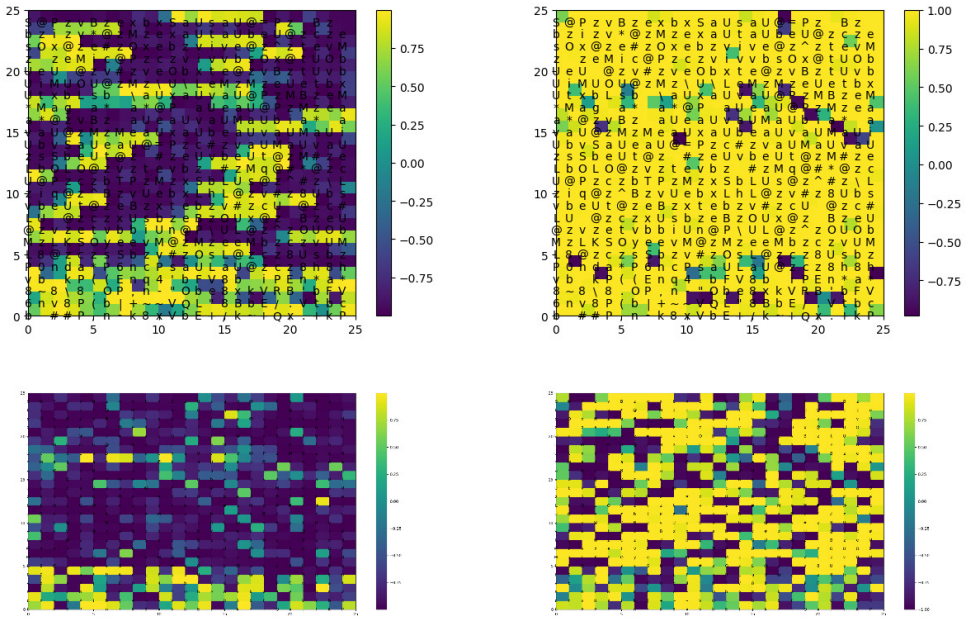

Figure 6: Heat maps for a various hidden neurons

Results and learnings

We were able to get 60.12% test set. The best part of the training was that we did not pre-process the data in any form. We even used the $< start >$ and $< end >$ tags to be present in the file. Our model could learn this structure , thereby ensuring proper formats of our target output file. Needless to say , it could generate midi files with proper headers.

Conclusion

We learned that RNN possesses great power to not only predict next character in the sequence but can also generate music at its own behest without much triggering. This shows us the enormous capacity of RNNs to do feature engineering across time. The natural algorithms like BPTT takes care of the shared weights learned across time. And hence provide us a learned model.

References

http://hjweide.github.io/char-rnn
http://karpathy.github.io/2015/05/21/rnn-effectiveness

Appendix

```
## Generate text using char RNN
import matplotlib.pyplot as plt
import numpy as np
import keras
from keras.models import Sequential
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM,SimpleRNN
from keras import backend as K
import random
import os.path

class LossHistory(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        data_file.write(str(epoch) + " " + str(logs.get('loss')) + " " + str(logs.get


def throwNSidedDice(massDist):
    randRoll = random.random()
    su = 0
    result = 0
    for mass in massDist:
        su += mass
        if randRoll < su:
            return result
        result+=1

def onehot(index,n_input):
        a = np.zeros(n_input)
        a[index] = 1
        return a

def findone(pred):
        for i in range(len(pred)):
                if pred[i] == 1:
                        return i

def getdata():
        lst = []
        with open("tinyshakespeare.txt") as f:
                for line in f:
                        for ch in line:
                                lst.append(ch)
        slst = set(lst)
        dic1 = {}
        dic2 = {}
        count = 0
        for i in slst:
                dic1[count] = i # given index, gives character
                dic2[i] = count
                count+=1
        ilst = []
        for i in lst:
                ilst.append(dic2[i])
        l = len(ilst)
        a = np.array(ilst)
        b = np.zeros((l,len(dic2)))
        b[np.arange(l),a] = 1
```

```
        return dic1,dic2,b


dic1,dic2,train = getdata()
n_hidden = 75
sample_size = 25
n_input = len(dic2)


train_data = train[:int(len(train)*0.8)]
test_data = train[int(len(train)*0.8):]
n_samples = int(len(train_data)/sample_size)
n_val = int(len(test_data)/sample_size)

# print(len(train))
# print(train[:25*int(len(train)/25)].shape)
train_data = train_data[:n_samples*sample_size]
val_data = test_data[:n_val*sample_size]

print(len(val_data))
train_data = np.reshape(train_data, (n_samples, sample_size, n_input))
print(train_data.shape)
val_data = np.reshape(val_data, (n_val, sample_size, n_input))
print(val_data.shape)
inp = train_data[:, :-1, ]
outp = train_data[:, 1:, ]

val_inp = val_data[:,:-1, ]
val_outp = val_data[:, 1:, ]
nepochs = 200

dropping_arr = [0.1]
for dropping_rate in dropping_arr:
    model = Sequential()
    model.add(SimpleRNN(n_hidden,input_dim = 65, return_sequences = True, activation
#     model.add(Dropout(dropping_rate))
    model.add(Dense(n_input, activation='softmax'))

    data_file = open('Output_epoch_' + str(dropping_rate) +  '.txt', 'w')
    history = LossHistory()

    newpath = 'generated_text'
    if not os.path.exists(newpath):
        os.makedirs(newpath)

    newpath = 'models'
    if not os.path.exists(newpath):
        os.makedirs(newpath)


    weights_file = "models/model_text_" + str(nepochs) + str(dropping_rate)+ ".h5"
    if not os.path.exists(weights_file):
            model.compile(optimizer = 'adam',loss= 'categorical_crossentropy', metri
            model.fit(inp,outp,batch_size = sample_size,nb_epoch = nepochs,callbacks=
            model.save_weights(weights_file)
    else:
            model.load_weights(weights_file)

    count = 0
```

```
    for x in [0, random.randint(1,len(test_data)),random.randint(1,len(test_data)),ra
            song_len = 20000
            prime_seq = test_data[x:x+25]
            prime_seq_reshape = np.reshape(prime_seq, (1, sample_size, n_input))
            # print (prime_seq)
            target = open("generated_text/generated_text_" + str(count)+ str(droppin
            print(target)
            count += 1
            print ("==============================")
            for i in range(25):
                    target.write(dic1[findone(prime_seq[i])])
                    print (dic1[findone(prime_seq[i])])

            for i in range(song_len):
                    prediction = model.predict(prime_seq_reshape,verbose=2)
                    #print (prediction.shape)
                    prediction = prediction[0]
                    #print(prediction)
                    index = throwNSidedDice(prediction[-1])
                    #print(i,index)
                    oh = onehot(index, n_input)
                    target.write(dic1[index])
                    #print(prime_seq.shape)
                    prime_seq = list(prime_seq)
                    prime_seq.append(oh)
                    prime_seq = prime_seq[1:]
                    prime_seq = np.array(prime_seq)
                    #print(prime_seq.shape)
                    prime_seq_reshape = np.reshape(prime_seq, (1, sample_size, n_inpu

## Generate music
## Test no. of neurons

import matplotlib.pyplot as plt
import numpy as np
import keras
from keras.models import Sequential
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM,SimpleRNN
from keras import backend as K
import random
import os.path

class LossHistory(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        data_file.write(str(epoch) + " " + str(logs.get('loss')) + " " + str(logs.get


def throwNSidedDice(massDist):
    randRoll = random.random()
    su = 0
    result = 0
    for mass in massDist:
        su += mass
        if randRoll < su:
            return result
        result+=1
```

```
def onehot(index,n_input):
        a = np.zeros(n_input)
        a[index] = 1
        return a

def findone(pred):
        for i in range(len(pred)):
                if pred[i] == 1:
                        return i

def getdata():
        lst = []
        with open("music-dataset.txt") as f:
                for line in f:
                        for ch in line:
                                lst.append(ch)
        slst = set(lst)
        dic1 = {}
        dic2 = {}
        count = 0
        for i in slst:
                dic1[count] = i # given index, gives character
                dic2[i] = count
                count+=1
        ilst = []
        for i in lst:
                ilst.append(dic2[i])
        l = len(ilst)
        a = np.array(ilst)
        b = np.zeros((l,len(dic2)))
        b[np.arange(l),a] = 1
        return dic1,dic2,b


dic1,dic2,train = getdata()
n_hidden = 75
sample_size = 25
n_input = len(dic2)


train_data = train[:int(len(train)*0.8)]
test_data = train[int(len(train)*0.8):]
n_samples = int(len(train_data)/sample_size)
n_val = int(len(test_data)/sample_size)

# print(len(train))
# print(train[:25*int(len(train)/25)].shape)
train_data = train_data[:n_samples*sample_size]
val_data = test_data[:n_val*sample_size]

print(len(val_data))
train_data = np.reshape(train_data, (n_samples, sample_size, n_input))
print(train_data.shape)
val_data = np.reshape(val_data, (n_val, sample_size, n_input))
print(val_data.shape)
inp = train_data[:, :-1, ]
outp = train_data[:, 1:, ]
```

```
val_inp = val_data[:,:-1, ]
val_outp = val_data[:, 1:, ]
nepochs = 55

neuron_arr = [50,75,100,125]
for n_hidden in neuron_arr:
    model = Sequential()
    model.add(SimpleRNN(n_hidden,input_dim = 93, return_sequences = True, activation
#     model.add(Dropout(dropping_rate))
    model.add(Dense(n_input, activation='softmax'))

    data_file = open('Kriti_Output_epoch_neuron' + str(n_hidden) +  '.txt', 'w')
    history = LossHistory()

    newpath = 'generated_music'
    if not os.path.exists(newpath):
        os.makedirs(newpath)

    newpath = 'models'
    if not os.path.exists(newpath):
        os.makedirs(newpath)


    weights_file = "models/model_neuron" + str(nepochs) + str(n_hidden)+ ".h5"
    if not os.path.exists(weights_file):
            model.compile(optimizer = 'adam',loss= 'categorical_crossentropy', metric
            model.fit(inp,outp,batch_size = sample_size,nb_epoch = nepochs,callbacks=
            model.save_weights(weights_file)
    else:
            model.load_weights(weights_file)

    count = 0


    for x in [0, random.randint(1,len(test_data))]:
            song_len = 2000
            prime_seq = test_data[x:x+25]
            prime_seq_reshape = np.reshape(prime_seq, (1, sample_size, n_input))
            # print (prime_seq)
            target = open("generated_music/generated_music_" + str(count)+ str(droppi
            count += 1
            print ("==============================")
            for i in range(25):
                    target.write(dic1[findone(prime_seq[i])])
                    print (dic1[findone(prime_seq[i])])

            for i in range(song_len):
                    prediction = model.predict(prime_seq_reshape,verbose=2)
                    #print (prediction.shape)
                    prediction = prediction[0]
                    #print(prediction)
                    index = throwNSidedDice(prediction[-1])
                    #print(i,index)
                    oh = onehot(index, n_input)
                    target.write(dic1[index])
                    #print(prime_seq.shape)
                    prime_seq = list(prime_seq)
                    prime_seq.append(oh)
                    prime_seq = prime_seq[1:]
```

```
                    prime_seq = np.array(prime_seq)
                    #print(prime_seq.shape)
                    prime_seq_reshape = np.reshape(prime_seq, (1, sample_size, n_inpu

## Test dropout
import matplotlib.pyplot as plt
import numpy as np
import keras
from keras.models import Sequential
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM,SimpleRNN
from keras import backend as K
import random
import os.path

class LossHistory(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        data_file.write(str(epoch) + " " + str(logs.get('loss')) + " " + str(logs.get


def throwNSidedDice(massDist):
    randRoll = random.random()
    su = 0
    result = 0
    for mass in massDist:
        su += mass
        if randRoll < su:
            return result
        result+=1

def onehot(index,n_input):
        a = np.zeros(n_input)
        a[index] = 1
        return a

def findone(pred):
        for i in range(len(pred)):
                if pred[i] == 1:
                        return i

def getdata():
        lst = []
        with open("music-dataset.txt") as f:
                for line in f:
                        for ch in line:
                                lst.append(ch)
        slst = set(lst)
        dic1 = {}
        dic2 = {}
        count = 0
        for i in slst:
                dic1[count] = i # given index, gives character
                dic2[i] = count
                count+=1
        ilst = []
        for i in lst:
                ilst.append(dic2[i])
        l = len(ilst)
        a = np.array(ilst)
```

```
        b = np.zeros((l,len(dic2)))
        b[np.arange(l),a] = 1
        return dic1,dic2,b


dic1,dic2,train = getdata()
n_hidden = 75
sample_size = 25
n_input = len(dic2)


train_data = train[:int(len(train)*0.8)]
test_data = train[int(len(train)*0.8):]
n_samples = int(len(train_data)/sample_size)
n_val = int(len(test_data)/sample_size)

# print(len(train))
# print(train[:25*int(len(train)/25)].shape)
train_data = train_data[:n_samples*sample_size]
val_data = test_data[:n_val*sample_size]

print(len(val_data))
train_data = np.reshape(train_data, (n_samples, sample_size, n_input))
print(train_data.shape)
val_data = np.reshape(val_data, (n_val, sample_size, n_input))
print(val_data.shape)
inp = train_data[:, :-1, ]
outp = train_data[:, 1:, ]

val_inp = val_data[:,:-1, ]
val_outp = val_data[:, 1:, ]
nepochs = 55

dropping_arr = [0.1,0.2,0.3]
for dropping_rate in dropping_arr:
    model = Sequential()
    model.add(SimpleRNN(n_hidden,input_dim = 93, return_sequences = True, activation
#     model.add(Dropout(dropping_rate))
    model.add(Dense(n_input, activation='softmax'))

    data_file = open('Kriti_Output_epoch_' + str(dropping_rate) +  '.txt', 'w')
    history = LossHistory()

    newpath = 'generated_music'
    if not os.path.exists(newpath):
        os.makedirs(newpath)

    newpath = 'models'
    if not os.path.exists(newpath):
        os.makedirs(newpath)


    weights_file = "models/model_" + str(nepochs) + str(dropping_rate)+ ".h5"
    if not os.path.exists(weights_file):
            model.compile(optimizer = 'adam',loss= 'categorical_crossentropy', metric
            model.fit(inp,outp,batch_size = sample_size,nb_epoch = nepochs,callbacks=
            model.save_weights(weights_file)
    else:
            model.load_weights(weights_file)
```

```
count = 0


for x in [0, random.randint(1,len(test_data))]:
        song_len = 2000
        prime_seq = test_data[x:x+25]
        prime_seq_reshape = np.reshape(prime_seq, (1, sample_size, n_input))
        # print (prime_seq)
        target = open("generated_music/generated_music_" + str(count)+ str(droppi
        count += 1
        print ("==============================")
        for i in range(25):
                target.write(dic1[findone(prime_seq[i])])
                print (dic1[findone(prime_seq[i])])

        for i in range(song_len):
                prediction = model.predict(prime_seq_reshape,verbose=2)
                #print (prediction.shape)
                prediction = prediction[0]
                #print(prediction)
                index = throwNSidedDice(prediction[-1])
                #print(i,index)
                oh = onehot(index, n_input)
                target.write(dic1[index])
                #print(prime_seq.shape)
                prime_seq = list(prime_seq)
                prime_seq.append(oh)
                prime_seq = prime_seq[1:]
                prime_seq = np.array(prime_seq)
                #print(prime_seq.shape)
                prime_seq_reshape = np.reshape(prime_seq, (1, sample_size, n_inpu
```