

Moody Music Generator

Objective: To automatically generate new random music based on users' selected mood, with as little user input as possible.

Name and UID

Chan Yu Yan, Sam (3035188203)

Joyce Leung (3035569720)

Leung Hiu Ching (3035573628)

1.Highlights of our work in THREE bullet points

- We selected, preprocessed music data (.midi) that are both labeled with (valence: 0/1 and arousal: 0/1) and unlabeled
- We built LSTM and Logistic Regression models (with & without random Gaussian noise) and then trained them with our music data to generate music based on 4 sentiments, which are sent 0=depressed, sent 1=calm, sent 2=angry and sent 3=delighted
- We conducted fine-tuning and experimentation of parameters and modifications to the model in order to improve the music quality generated and classifying performance.

2.Report of tasks achieved

Data Engineering done

Data Acquisition

We have acquired and used the VGMIDI Dataset in our model.

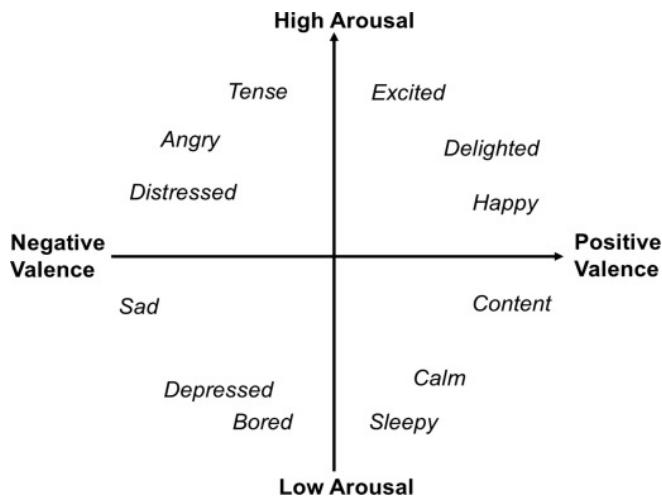
There are 721 non-labelled and 161 labelled MIDI piano pieces from video soundtracks. Video game soundtracks are used, because the music is composed in a way to keep the player in a certain affective state.

For the labelled ones, each piece was annotated by 30 human subjects according to a valence-arousal model, which represents emotion. Valence indicates positive versus negative emotion, and arousal indicates emotional intensity. Data is processed such that each piece has a 1 or 0 valence and arousal values.

Details of the annotation process could be found in Ferreira's and Whitehead's paper.

Data Visualisation and Preprocessing

In the data folder "vgmidi-master", originally, vgmidi_test.csv (test data) and vgmidi_train.csv (training data) were just labelled with valence = 0/1 and arousal = 0/1. As we may need to transform our original trained 1-D model to a 2-D model, we finally decided to make a column of 'feeling', using the formula of $\text{value_valence} + \text{value_arousal} * 2$. Thus, from the diagram below, we got



Sent 0 (valence = 0; arousal = 0) = depressed, sad, bored
 Sent 1 (valence = 1; arousal = 0) = calm, sleepy, content
 Sent 2 (valence = 0; arousal = 1) = angry, tense, distressed
 Sent 3 (valence = 1; arousal = 1) = delighted, happy, excited

Here is our data composition of vgmidi-master with 4 labeled sentiments.

```
Total number of pieces: 161
Proportion of sent 0 (depressed): 14.906832%
Proportion of sent 1 (calm): 28.571429%
Proportion of sent 2 (angry): 19.875776%
Proportion of sent 3 (happy): 36.645963%
```

From this, we can see that while the data is slightly inclined more towards happy and calm music, the imbalance is not severe.

Our data consist of audio in .wav format converted to .mid format. The .mid format contains information such as melody, harmony, tempo and timbre of a music piece.

Features including the pitch, note duration, note velocity, piece tempo are encoded using a sequence of words and punctuations.

A vocabulary of the indices corresponding to each unique word in our music sentences was built

```
["n": 0, "d_16th_0": 1, "d_breve_0": 2, "d_breve_1": 3, "d_breve_2": 4, "d_breve_3": 5, "d_eighth_0": 6, "d_eighth_1": 7,
 "d_half_0": 8, "d_half_1": 9, "d_half_2": 10, "d_half_3": 11, "d_quarter_0": 12, "d_quarter_1": 13, "d_quarter_2": 14,
 "d_whole_0": 15, "d_whole_1": 16, "d_whole_2": 17, "d_whole_3": 18, "d_whole_4": 19, "n_31": 20, "n_34": 21, "n_35": 22, "n_36":
 23, "n_37": 24, "n_38": 25, "n_39": 26, "n_40": 27, "n_41": 28, "n_42": 29, "n_43": 30, "n_44": 31, "n_45": 32, "n_46": 33,
 "n_47": 34, "n_48": 35, "n_49": 36, "n_50": 37, "n_51": 38, "n_52": 39, "n_53": 40, "n_54": 41, "n_55": 42, "n_56": 43, "n_57":
 44, "n_58": 45, "n_59": 46, "n_60": 47, "n_61": 48, "n_62": 49, "n_63": 50, "n_64": 51, "n_65": 52, "n_66": 53, "n_67": 54,
 "n_68": 55, "n_69": 56, "n_70": 57, "n_71": 58, "n_72": 59, "n_73": 60, "n_74": 61, "n_75": 62, "n_76": 63, "n_77": 64, "n_78":
 65, "n_79": 66, "n_80": 67, "n_81": 68, "n_82": 69, "n_83": 70, "n_84": 71, "n_85": 72, "n_86": 73, "n_87": 74, "n_88": 75,
 "n_89": 76, "n_90": 77, "n_91": 78, "n_92": 79, "t_100": 80, "t_102": 81, "t_104": 82, "t_105": 83, "t_107": 84, "t_108": 85,
 "t_110": 86, "t_112": 87, "t_113": 88, "t_115": 89, "t_116": 90, "t_118": 91, "t_120": 92, "t_121": 93, "t_123": 94, "t_124": 95,
 "t_126": 96, "t_128": 97, "t_129": 98, "t_131": 99, "t_132": 100, "t_134": 101, "t_136": 102, "t_137": 103, "t_139": 104, "t_140":
 105, "t_142": 106, "t_144": 107, "t_145": 108, "t_147": 109, "t_148": 110, "t_150": 111, "t_152": 112, "t_153": 113, "t_155": 114,
 "t_156": 115, "t_158": 116, "t_160": 117, "t_24": 118, "t_25": 119, "t_27": 120, "t_28": 121, "t_30": 122, "t_32": 123, "t_33":
 124, "t_35": 125, "t_36": 126, "t_38": 127, "t_40": 128, "t_42": 129, "t_43": 130, "t_44": 131, "t_46": 132, "t_48": 133, "t_49":
 134, "t_51": 135, "t_52": 136, "t_54": 137, "t_56": 138, "t_57": 139, "t_59": 140, "t_60": 141, "t_62": 142, "t_64": 143, "t_65":
 144, "t_67": 145, "t_68": 146, "t_70": 147, "t_72": 148, "t_73": 149, "t_75": 150, "t_76": 151, "t_78": 152, "t_80": 153, "t_81":
 154, "t_83": 155, "t_84": 156, "t_86": 157, "t_88": 158, "t_89": 159, "t_91": 160, "t_92": 161, "t_94": 162, "t_96": 163, "t_97":
 164, "v_99": 165, "v_100": 166, "v_104": 167, "v_108": 168, "v_112": 169, "v_116": 170, "v_121": 171, "v_126": 172, "v_124": 173,
 "v_126": 174, "v_20": 175, "v_24": 176, "v_28": 177, "v_32": 178, "v_36": 179, "v_4": 180, "v_40": 181, "v_44": 182, "v_48": 183,
 "v_52": 184, "v_56": 185, "v_60": 186, "v_64": 187, "v_68": 188, "v_72": 189, "v_76": 190, "v_8": 191, "v_80": 192, "v_84": 193,
 "v_88": 194, "v_92": 195, "v_96": 196, "w_3": 197, "w_10": 198, "w_11": 199, "w_12": 200, "w_126": 201, "w_129": 202, "w_13": 203,
 "w_14": 204, "w_15": 205, "w_16": 206, "w_17": 207, "w_18": 208, "w_19": 209, "w_2": 210, "w_20": 211, "w_21": 212, "w_22": 213,
 "w_23": 214, "w_24": 215, "w_25": 216, "w_26": 217, "w_27": 218, "w_28": 219, "w_29": 220, "w_3": 221, "w_30": 222, "w_31": 223,
 "w_32": 224, "w_33": 225, "w_34": 226, "w_35": 227, "w_36": 228, "w_37": 229, "w_38": 230, "w_39": 231, "w_4": 232, "w_40": 233,
 "w_41": 234, "w_42": 235, "w_44": 236, "w_47": 237, "w_48": 238, "w_49": 239, "w_5": 240, "w_53": 241, "w_54": 242, "w_55": 243,
 "w_61": 244, "w_64": 245, "w_65": 246, "w_7": 247, "w_75": 248, "w_73": 249, "w_8": 250, "w_85": 251, "w_8": 252, "w_93": 253]
```

An example of our vocabulary, built from the training and testing datasets before splitting.

AI/ML Model Implemented & Fine-tuned

As we have encoded and represented the music pieces as a sequence of words and punctuation marks, music composition can be treated like a language modelling problem.

Our model uses the combination of mLSTM and logistic regression to generate music based on mood. We believe that implementing this approach would work well because generative mLSTM is capable of learning in an unsupervised way, and it would be a better representation of sentiment in music pieces. Also, unlabelled music is less expensive than labelled ones and they are easier to obtain, this approach allows the use of more unlabelled music to improve the model performance.

For the mLSTM, it consists of 4 LSTM layers, an embedding layer to compress the input feature space into a smaller one and a dense layer. We use the unlabelled MIDI pieces to train a generative mLSTM, in order to predict the next word in a sequence. There is also a Gaussian Noise Layer, to reduce overfitting and to introduce some randomness to prevent model being stacked into a local loop.

Layer (type)	Output Shape	Param #
embedding (Embedding)	(100, None, 256)	65024
lstm (LSTM)	(100, None, 512)	1574912
gaussian_noise (GaussianNoise)	(100, None, 512)	0
lstm_1 (LSTM)	(100, None, 512)	2099200
lstm_2 (LSTM)	(100, None, 512)	2099200
lstm_3 (LSTM)	(100, None, 512)	2099200
dense (Dense)	(100, None, 254)	130302
Total params: 8,067,838		
Trainable params: 8,067,838		
Non-trainable params: 0		

Experiments carried out

Model Initial Training

We have trained the original LSTM model for 18 epochs, using the original model fine tuned in the Interim Report. Hyperparameter: Embedding size:256; LSTM units:512; LSTM layers:4; Batch size:100; learning rate:0.00001; Sequence length:256; Dropout: 0.05

The structure of that LSTM model:

Model: "sequential_1"

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(100, None, 256)	65024
lstm_4 (LSTM)	(100, None, 512)	1574912
lstm_5 (LSTM)	(100, None, 512)	2099200
lstm_6 (LSTM)	(100, None, 512)	2099200
lstm_7 (LSTM)	(100, None, 512)	2099200
dense_1 (Dense)	(100, None, 254)	130302

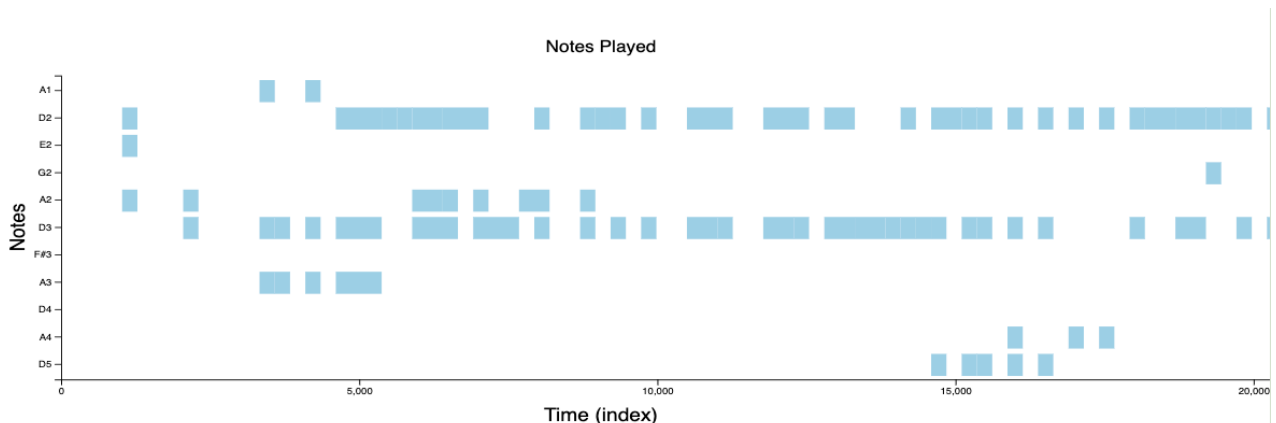
Epoch	Loss	val_loss
1	5.1705	3.7945
2	4.1699	3.3207
3	3.6451	3.0722
4	3.3849	3.0466
5	2.9457	2.751
6	2.6473	2.5737
7	2.4396	2.4402
8	2.2914	2.3641
9	2.198	2.3142
10	2.1199	2.2911
11	2.0571	2.2727
12	2.0013	2.2634
13	1.951	2.2576
14	1.9049	2.2551
15	1.8609	2.2569
16	1.8197	2.2614
17	1.7807	2.2656
18	1.7432	2.2747

Epoch	Generated soundtracks
2	Positive1 Negative1
3	Positive1 Positive2
8	Positive1 Positive2 Negative1 Negative2
10	Sent3_1 Music_2
13	Positive1 Negative1
15	Positive1 Negative1
18	NoSentConrol1 Negative1

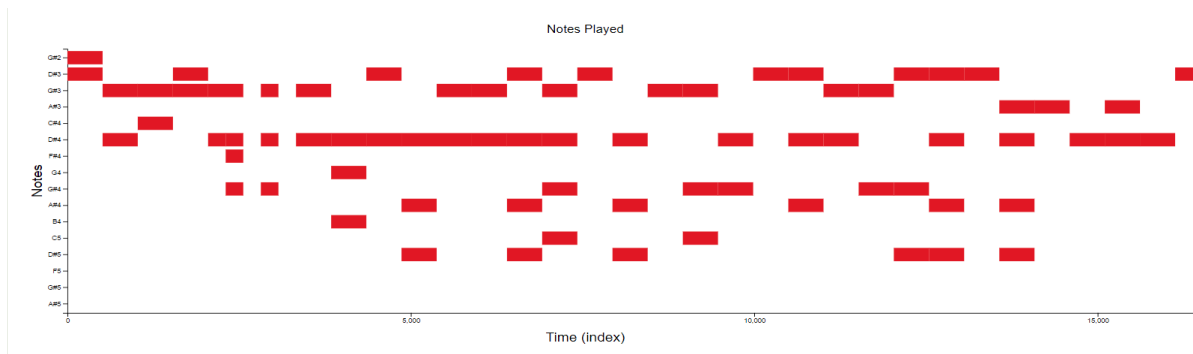
Evaluation of the generated soundtracks throughout different epoch

Originally, there were just a few notes every 5-10 seconds and it did not sound like a music track but just random music notes. Some generated music could even last for only 3 seconds. After 8 epochs, the music had fewer gaps and there was more difference between notes but some notes were repeatedly used for a few consecutive seconds.

Epoch 8 music, with low note count and has a lot of emptiness, the number of different pitches is very small:



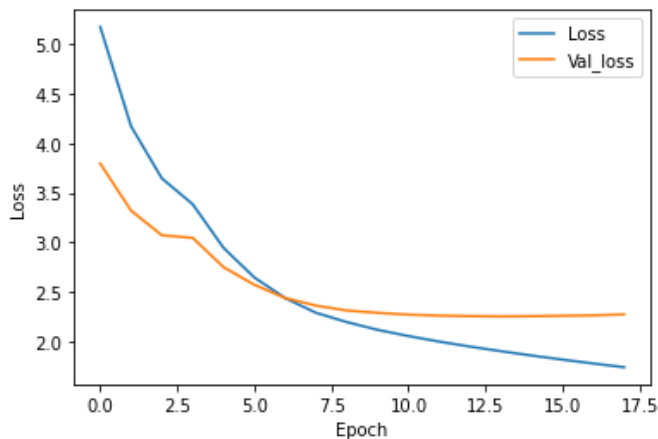
Epoch 18 music, with higher note count, and have more different pitches:



However, the negative music was noisier than the positive music. Besides, the negative music tended to be positive while the positive music still sounded positive.

As the music is still monotonous, it seems that the model is stucked into a local loop. Hence, more noise is needed to be introduced into the model to increase the randomness.

Also, we have plotted a graph to represent the loss on train and test set. It can be observed that our model has overfitting, as loss is divergent. While validation loss decreases till epoch 14, it has been increasing since then. It has the shape of an overfit model.

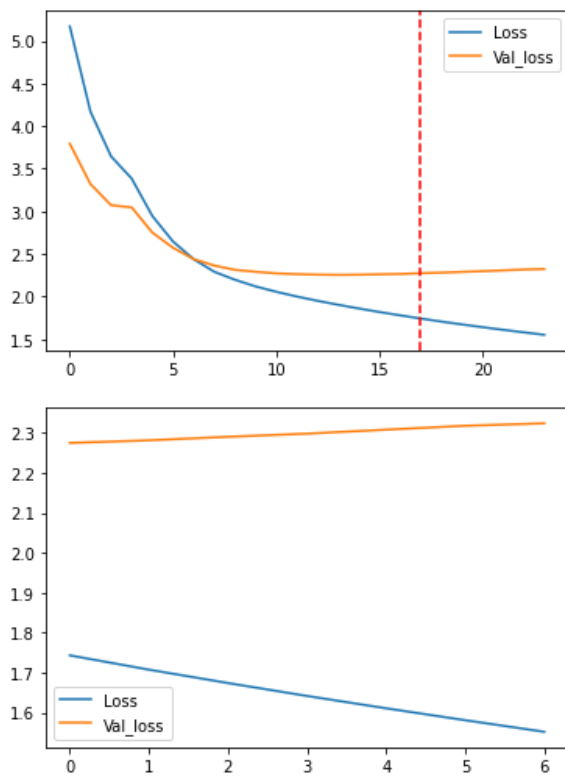
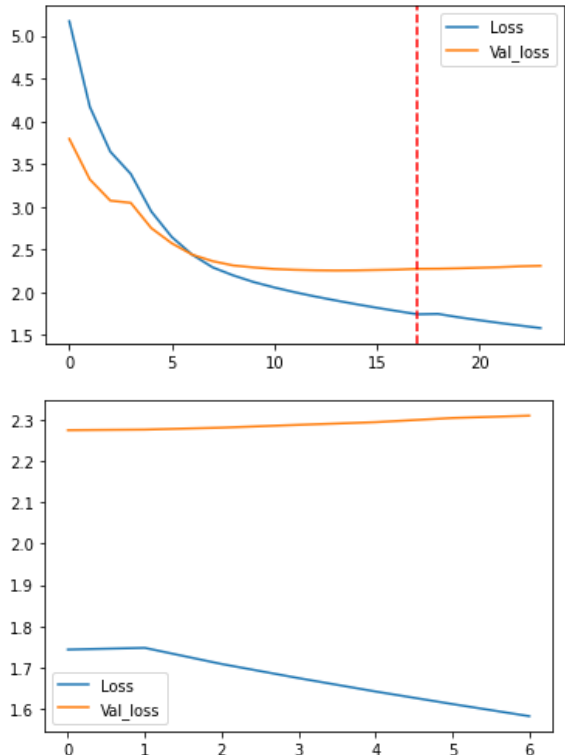


This shows that the model performs well on the training dataset and poorly on unseen data. To modify our learning algorithm to reduce its generalization error but not its training error, we decided to perform generalisation. Hence, we have done experiments on adding dropout and introducing stochastic noise into the training process, to see if they can reduce the problem of overfitting and repetitive music.

Testing the effects of adding Gaussian Noise to the generative model

Layer (type)	Output Shape	Param #
embedding (Embedding)	(100, None, 256)	65024
lstm (LSTM)	(100, None, 512)	1574912
gaussian_noise (GaussianNois	(100, None, 512)	0
lstm_1 (LSTM)	(100, None, 512)	2099200
lstm_2 (LSTM)	(100, None, 512)	2099200
lstm_3 (LSTM)	(100, None, 512)	2099200
dense (Dense)	(100, None, 254)	130302
Total params: 8,067,838		
Trainable params: 8,067,838		
Non-trainable params: 0		

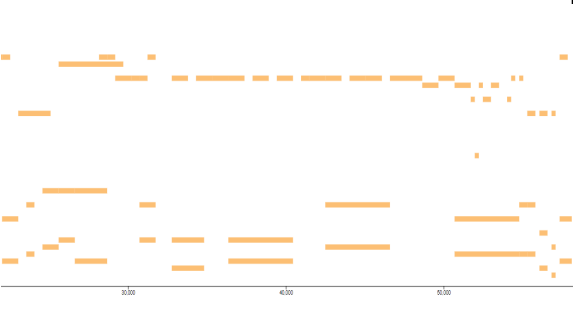
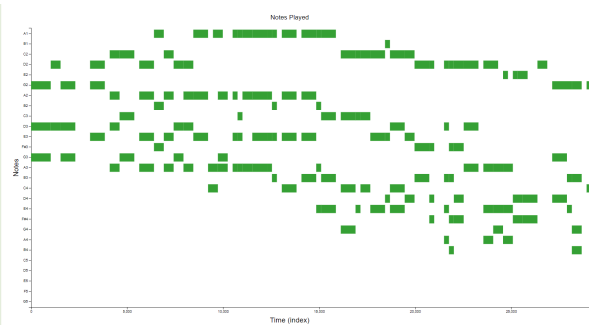
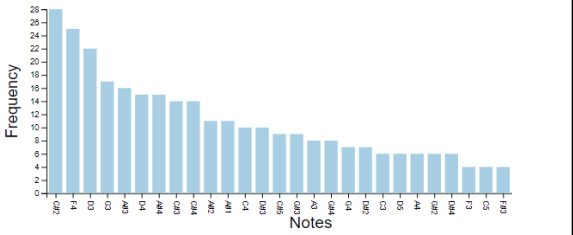
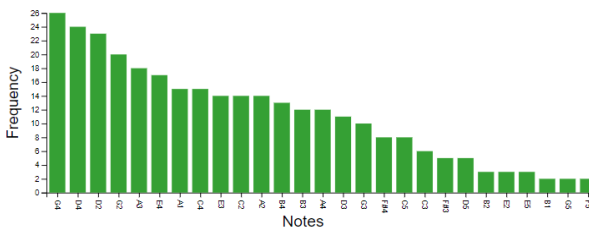
To see if adding a noise layer can alleviate overfitting, we have trained the same model with and without a gaussian noise layer for 6 epochs, starting from epoch 18 of the initial model. Below shows the loss and validation loss of epoch 18-24 training.

Tuning	Loss and Validation Loss Graph (red line indicates when we start training each respective model)	Data	Audio Track																
Without the Gaussian Noise Layer		<table><tr><th>Loss</th><th>val_loss</th></tr><tr><td>1.7432</td><td>2.2747</td></tr><tr><td>1.7074</td><td>2.2812</td></tr><tr><td>1.6739</td><td>2.29</td></tr><tr><td>1.6416</td><td>2.2979</td></tr><tr><td>1.6107</td><td>2.3079</td></tr><tr><td>1.5809</td><td>2.3176</td></tr><tr><td>1.5519</td><td>2.3237</td></tr></table>	Loss	val_loss	1.7432	2.2747	1.7074	2.2812	1.6739	2.29	1.6416	2.2979	1.6107	2.3079	1.5809	2.3176	1.5519	2.3237	Sent0_1 Sent0_2 Sent0_3 Sent1_1 Sent1_2 Sent1_3
Loss	val_loss																		
1.7432	2.2747																		
1.7074	2.2812																		
1.6739	2.29																		
1.6416	2.2979																		
1.6107	2.3079																		
1.5809	2.3176																		
1.5519	2.3237																		
With Gaussian Noise Layer with std of of the noise distribution=0.2		<table><tr><th>Loss</th><th>Validation Loss</th></tr><tr><td>1.7432</td><td>2.2747</td></tr><tr><td>1.7473</td><td>2.2761</td></tr><tr><td>1.7082</td><td>2.2808</td></tr><tr><td>1.674</td><td>2.2876</td></tr><tr><td>1.6416</td><td>2.2941</td></tr><tr><td>1.6112</td><td>2.3042</td></tr><tr><td>1.5818</td><td>2.3098</td></tr></table>	Loss	Validation Loss	1.7432	2.2747	1.7473	2.2761	1.7082	2.2808	1.674	2.2876	1.6416	2.2941	1.6112	2.3042	1.5818	2.3098	Sent0_1 Sent0_2 Sent0_3 Sent1_1 Sent1_2 Sent2_1 Sent2_2 Sent3_1 Sent3_2
Loss	Validation Loss																		
1.7432	2.2747																		
1.7473	2.2761																		
1.7082	2.2808																		
1.674	2.2876																		
1.6416	2.2941																		
1.6112	2.3042																		
1.5818	2.3098																		

Evaluation of the testing result

There is no significant difference between the two tuning cases. However, the one with Gaussian noise layer has its validation loss increased at a slower rate (+1.543%) than the model without a noise layer (+2.154%). The loss is also less divergent than the model without a noise layer. This shows that Gaussian noise layer can decrease the overfitting to a small extent only.

For the generated soundtracks, we compared sent0_1 of model without a Gaussian layer, to sent0_3 of the model with a Gaussian layer:

	A (Sent0_1) (without tuning)	B (Sent0_3) (with noise layer)
Note count	612	610
Notes range	As2-F6	A2-G6
Tempo	112	104
Note distribution over time		
Note frequency		



In scientific pitch notation, C4 denotes the middle C,

notes B3 or below are usually the bass.

The note count, note range and tempo are similar. Yet, track A is relatively more monotonous and repetitive than track B.

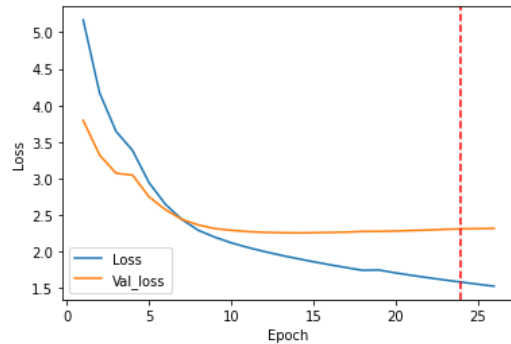
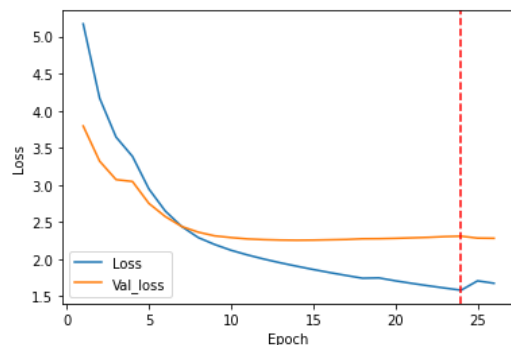
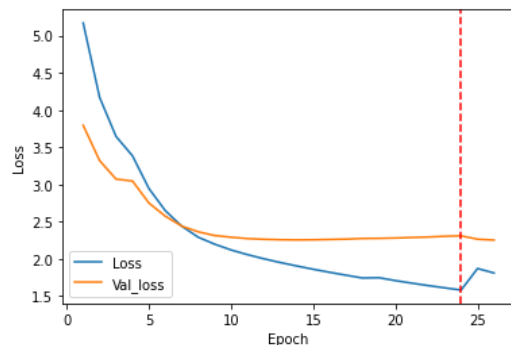
From the note distribution over time graph, A is more monotonous than B, and A keeps having certain notes repeatedly played with low involvement of other notes being played together.

From the note frequency graph, the number of different pitches of A is less than B. Also, for A, the music is mostly dominated by two or three pitch, C#2, F4, D3. However, for B, the situation improved and the music was less dominated by one or two pitches, the frequencies of different notes are relatively similar when compared to A.

Hence, the Gaussian noise layer can slightly improve the generalisation, introduce randomness to the music, and make the data/note distribution smoother. That is why we will have the noise layer in our final model.

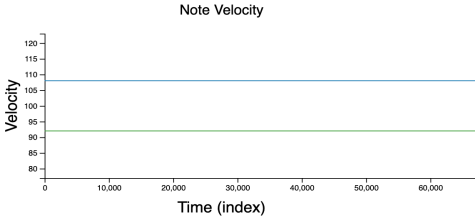
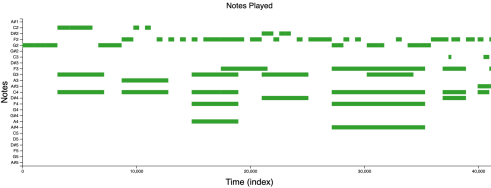
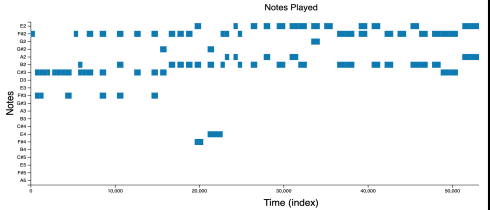
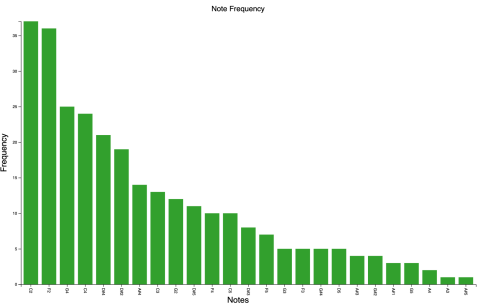
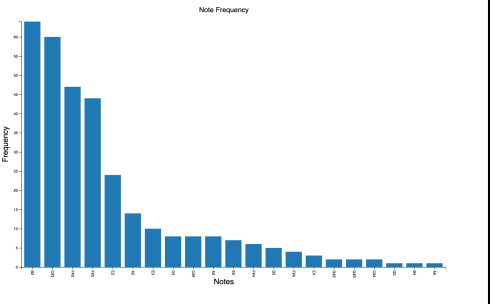
Testing the effects of changing dropout rate

In our model, a dropout rate is applied to every mLSTM layer. Changing this value would change the dropout value of every mLSTM layer. To find the optimal dropout rate, we have trained the model with different dropout rates for 2 epochs, starting from epoch 24 of the model with the Gaussian layer. (loss: 1.5818 and val_loss: 2.3098 for epoch 24)

Dropout rate	Loss and Validation Loss Graph	Data	Audio Track									
0.05		<table><tr><th>Epoch</th><th>Loss</th><th>Val_loss</th></tr><tr><td>25</td><td>1.5534</td><td>2.3132</td></tr><tr><td>26</td><td>1.5256</td><td>2.3165</td></tr></table>	Epoch	Loss	Val_loss	25	1.5534	2.3132	26	1.5256	2.3165	
Epoch	Loss	Val_loss										
25	1.5534	2.3132										
26	1.5256	2.3165										
0.15		<table><tr><th>Epoch</th><th>Loss</th><th>Val_loss</th></tr><tr><td>25</td><td>1.7080</td><td>2.2840</td></tr><tr><td>26</td><td>1.6755</td><td>2.2816</td></tr></table>	Epoch	Loss	Val_loss	25	1.7080	2.2840	26	1.6755	2.2816	sent0_1 sent0_2 sent0_3
Epoch	Loss	Val_loss										
25	1.7080	2.2840										
26	1.6755	2.2816										
0.25		<table><tr><th>Epoch</th><th>Loss</th><th>Val_loss</th></tr><tr><td>25</td><td>1.8712</td><td>2.2647</td></tr><tr><td>26</td><td>1.8105</td><td>2.2525</td></tr></table>	Epoch	Loss	Val_loss	25	1.8712	2.2647	26	1.8105	2.2525	sent0_1 sent0_2 sent0_3
Epoch	Loss	Val_loss										
25	1.8712	2.2647										
26	1.8105	2.2525										

Evaluation of the testing result

For dropout rate 0.05, validation loss continues to increase while loss is decreasing. With higher dropout rates of 0.15 and 0.25, the problem of overfitting is alleviated. There is less divergence between loss and validation loss value, and validation loss decreases.

	A (sent0_3, dropout rate 0.15)	B (sent0_3, dropout rate 0.25)
Note count	570	642
Notes range	As2-As6	E3-A6
Tempo	112	112
Note velocity		
Note distribution over time (key lowers along y-axis)		
Note frequency		

An increase in dropout rate does not translate to a less repetitive and monotonic bass. Both pieces have repetitions of the same note over time. The piece with drop rate 0.15 has more chords and is less focused on the lower key. In the piece with drop rate 0.25, we see the frequency of notes are more right-skewed with a longer tail, meaning more notes are close to the bass end. There is little improvement of repetitiveness when using a higher dropout rate, contrary to what we expected.

As we want to improve the overfitting problem while having less repetitions in our music, we will use a dropout rate of 0.15 for our model training.

Testing the effects of adding dropout layer

Currently, we only have dropout applied to the recurrent input signal on every LSTM layer and the input connection within the LSTM nodes.

In this experiment, we would like to see if adding a dropout layer before the dense layer prevents overfitting in our model. The dropout layer can randomly set input units to 0 with a frequency of rate at each step during training time, and we have set rate=0.5.

The structure of the tested model:

Model: "sequential_1"

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(100, None, 256)	65024
lstm_4 (LSTM)	(100, None, 512)	1574912
lstm_5 (LSTM)	(100, None, 512)	2099200
lstm_6 (LSTM)	(100, None, 512)	2099200
lstm_7 (LSTM)	(100, None, 512)	2099200
dropout_1 (Dropout)	(100, None, 512)	0
dense_1 (Dense)	(100, None, 254)	130302
Total params: 8,067,838		
Trainable params: 8,067,838		
Non-trainable params: 0		

We have trained the model with the dropout layer for 3 epochs, starting from epoch 18 of the original model. (loss: 1.7432 and val_loss: 2.2747 for epoch 18)

Epoch	Loss	Validation Loss
19	1.9182	2.2815
20	1.8193	2.3034
21	1.7609	2.3254
22	1.6688	2.3620

Evaluation of the testing result

The validation loss increases at a faster rate than training a model without the dropout layer, the divergence between loss and validation loss actually increases with more training. The problem of overfitting worsens.

Therefore, we will not have the dropout layer in our finalized model.

Testing the effects of changing the value of C, the inverse of regularization strength, and the regularizations l1 and l2

Grid search with 10 folds in Stratified KFold.

Tuning	param_grid	Best parameters	Accuracy
1st tuning	<pre>{'C': [0.00390625, 0.0078125, 0.015625, 0.03125, 0.0625, 0.125, 0.25, 0.5, 1], 'penalty': ['l1', 'l2']}</pre>	<pre>{'C': 0.0078125, 'penalty': 'l2'}</pre>	0.5404411764705882

Fix l2, refine search area around C=0.0078125	<pre>param_grid = {'C': [0.005, 0.006, 0.007, 0.0078125, 0.008, 0.009, 0.010, 0.011, 0.012, 0.013, 0.014], 'penalty': ['l2']}</pre>	<pre>{'C': 0.0078125, 'penalty': 'l2'}</pre>	0.5404411764705882
Further narrowing down around C=0.0078125	<pre>param_grid = {'C': [0.0075000, 0.0076000, 0.0077000, 0.0078000, 0.0078125, 0.0079000, 0.0079500], 'penalty': ['l2']}</pre>	<pre>{'C': 0.0077, 'penalty': 'l2'}</pre>	0.5404411764705882

Test accuracy using {'C': 0.0077, 'penalty': 'l2'}: 50.0

Grid Search using

`RAND_SEED = 3359`

```
cv = StratifiedShuffleSplit(n_splits=10, test_size=0.2,
random_state=RAND_SEED)
```

as stratification strategy

Tuning	param_grid	Best parameters	Accuracy
1st tuning	<pre>{'C': [0.00390625, 0.0078125, 0.015625, 0.03125, 0.0625, 0.125, 0.25, 0.5, 1], 'penalty': ['l1', 'l2']}</pre>	<pre>{'C': 0.03125, 'penalty': 'l2'}</pre>	0.4878787878787879
Fix l2, refine search around C=0.03125	<pre>{'C': [0.02000, 0.02500, 0.03125, 0.03500, 0.04000, 0.04500, 0.05000, 0.0600], 'penalty': ['l2']}</pre>	<pre>{'C': 0.04, 'penalty': 'l2'}</pre>	0.490909090909091
Refine search around C=0.04	<pre>{'C': [0.0360, 0.0370, 0.0380, 0.0390, 0.0400, 0.0410, 0.0420, 0.0430, 0.0440], 'penalty': ['l2']}</pre>	<pre>{'C': 0.042, 'penalty': 'l2'}</pre>	0.49696969696969706
Refine search around C=0.042	<pre>{'C': [0.0415, 0.0420, 0.0425], 'penalty': ['l2']}</pre>	<pre>{'C': 0.0415, 'penalty': 'l2'}</pre>	0.49696969696969706

Test accuracy using {'C': 0.0415, 'penalty': 'l2'}: 45.0

Grid search with 10 folds in Stratified KFold with l1 fixed

Tuning	param_grid	Best parameters	Accuracy
--------	------------	-----------------	----------

1st tuning	<code>{'C': [0.00390625, 0.0078125, 0.015625, 0.03125, 0.0625, 0.125, 0.25, 0.5, 1]}</code>	<code>{'C': 0.25}</code>	<code>0.5091911764705882</code>
Refine at 0.25	<code>{'C': [0.15, 0.20, 0.25, 0.30, 0.35]}</code>	<code>{'C': 0.3}</code>	<code>0.5150735294117647</code>
Refine at 0.3	<code>{'C': [0.27, 0.28, 0.29, 0.30, 0.31, 0.32, 0.33, 0.34]}</code>	<code>{'C': 0.27}</code>	<code>0.5213235294117646</code>
Refine at 0.27	<code>{'C': [0.255, 0.260, 0.265, 0.270, 0.275]}</code>	<code>{'C': 0.27}</code>	<code>0.5213235294117646</code>
Further narrow at 0.27	<code>{'C': [0.2695, 0.2700, 0.2705]}</code>	<code>{'C': 0.2695}</code>	<code>0.5213235294117646</code>

Test accuracy using `{'C': 0.2695, 'penalty': 'l1'}`: 50.0

3. Results on model evaluation and issue analysis

Generated music using the finalized model, with a Gaussian Noise Layer and dropout=0.15 according to questionnaire order: [Q1](#), [Q2](#), [Q3](#), [Q4](#), [Q5](#), [Q6](#), [Q7](#), [Q8](#)

To evaluate our generated music pieces, we devised a survey, asking respondents to choose an emotional category from sent 0 to sent 3 that best fits the music piece in the link of each question. The [link](#) will direct respondents to an MP3 file which has a name that does not leak the sentiment of the music piece. We sent the survey to 23 different individuals, each survey containing 8 music pieces pre-shuffled into an order that matches the answer key below.

The answer is: 0,3,2,2,1,0,3,1 (where 0 represents the first choice and 3 the fourth choice)

The results are as shown

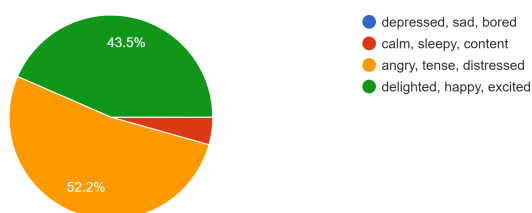
Q1. Link to	Q2. Link to the	Q3. Link to the	Q4. Link to the	Q5. Link to the	Q6. Link to the	Q7. Link to the	Q8. Link to the	score
0	3	2	2	1	2	3	0	6/8
0	2	1	1	0	0	2	1	3/8
0	3	2	2	1	0	3	1	8/8
0	2	1	1	1	0	2	0	3/8
2	3	2	2	0	3	3	2	4/8
2	2	3	2	2	3	3	3	2/8
2	2	2	2	2	3	1	3	2/8
2	3	3	1	0	1	1	1	2/8
0	2	1	0	0	2	2	0	1/8
0	2	3	1	0	0	2	0	2/8
0	3	3	3	0	3	0	1	3/8
2	2	0	2	1	0	2	1	4/8
1	2	3	0	1	3	2	1	2/8
0	3	1	2	1	0	3	1	6/8
0	2	2	2	1	0	2	1	6/8
0	3	2	2	1	0	3	1	8/8
0	2	2	2	1	0	3	1	7/8
1	3	2	1	1	0	3	1	6/8
0	3	1	2	1	1	3	1	6/8
1	2	1	2	2	2	2	2	1/8
2	2	2	2	2	2	2	2	2/8
2	1	2	2	2	2	2	2	2/8
3	3	2	2	0	3	3	3	4/8

Averaging the scores of all respondents gives around 48.91%. It can be due to the subjective nature of mood determination, and high overlapping of mood within and between music pieces. It is also possible that the music generated may not perfectly match the sentiment we used to generate it.

In question 2 and 7, the number of respondents who chose excited (green in the pie chart) and angry (yellow) are similar, while both questions' answers are excited instead. This shows that our generated music with the excited mood is very ambiguous, having both angry and excited measures. To understand why, we will analyze the two music pieces.

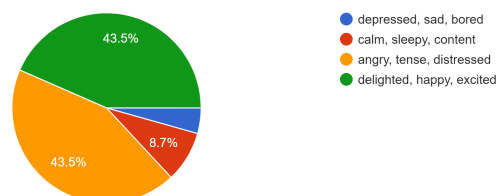
Q2. Link to the soundtrack:

https://drive.google.com/file/d/1FeZLY1zrSsfCMkEfTDko9sTv_7J1myF1/view?usp=sharing
23 responses

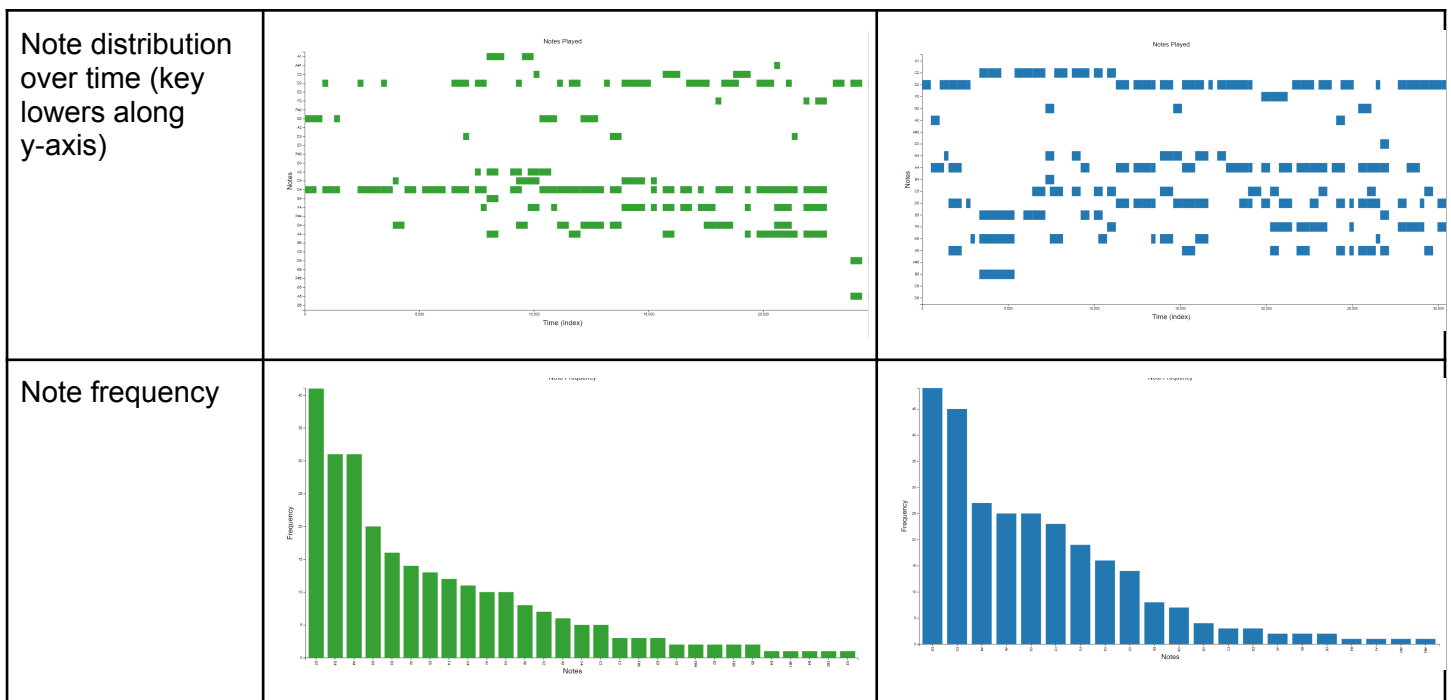


Q7. Link to the soundtrack:

<https://drive.google.com/file/d/1nhybtP27vcp3LLyQZ9Y3a7UBJq2TESC6/view?usp=sharing>
23 responses



	Q3 (mood2-angry)	Q2 (mood3-excited)
Note count	528	556
Notes range	A2-B6	A2-D7
Tempo	N/A (couldn't get the tempo for this midi)	160
Note velocity (green for the angry music, blue for the excited one)	<p>Velocity</p> <p>Time (index)</p>	



Both angry and excited music should have a high note velocity and high tempo to emphasize the mood. The note frequency, range and note count are also very similar. This shows that there is ambiguity in the labelled excited and angry music data.

It is also similar for the calm/depressing music. That is because both mood have music that have low note velocity and low tempo.

4. Possible limitations and remaining issues

Possible limitations:

1. Unavoidable Bias for the music classifier
According to Module 3, unavoidable Bias refers to the source of error that is intrinsic to the problem. It is considered as “almost impossible” for one’s solution to overcome this issue, hence the name “unavoidable”. Unavoidable Bias is usually measured by Optimal Error Rate (a.k.a. Bayes Error, Irreducible Error). Sometimes, Optimal Error Rate is determined according to the performance of state-of-the-art solutions. Currently, the best accuracy (in 2019) achieved on a Multi-Class Image Classification task (not Multi-Label) is 0.844, so if we are to build a multi-class image classifier model, it would be very hard for us to beat this performance. This might even be less good than our multi-class music classification task.
2. Inaccurate and Mislabeled Data
Music pieces often contain parts that represent different emotions. For example, the “Final Fantasy 7 Continue” piece would be labeled sent 1 (calm, sleepy) according to our valence-arousal measure. Yet, we find it contains parts that can be regarded as sent 0 (depressed, sad). Another example is “BanjoKazooie Motzhand”, which is labeled as sent 3 (happy, excited). While the general theme matches sent 3, it contains melodies that can qualify as sent 0. It is an inherent problem that many overlapping emotions coexist in a music piece at the same time. One way to mitigate this issue, is to extract music parts that

are very pure in one emotion. Long music tracks often involve many emotions mixed together, and will require a lot of work to carefully extract the parts out for labeling. This is a huge limitation to the performance of the classifier and neurons evolved.

3. Subjective Nature of Evaluating Music

Different people will have different interpretations of the quality and the mood of music, and hence, it is difficult to generate music with a mood that could be agreed by everyone, making it difficult to develop a metric to quantitatively evaluate the result. As it is hard to evaluate the quality of music, it is difficult to tune the model according to the generated music results.

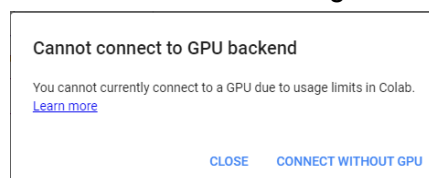
Also, the label of the labelled data could be not exactly accurate too.

4. Ambiguous Nature of the Mood of Music

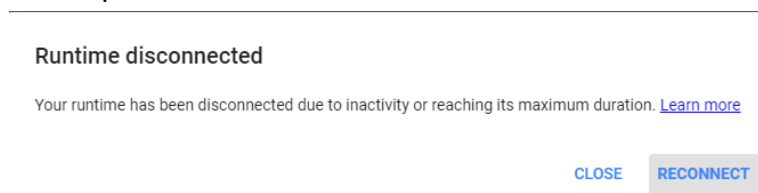
Most of the labelled music pieces are very ambiguous in nature, for example, a sad music piece is similar to a calm and content piece, and an exciting piece is similar to an angry one. It is hard for the classifier to identify the pattern between different moods, making the performance of the classifier quite poor.

5. Unstable GPU

When we used the GPU on Google Colab too much, the following would be shown.



When we trained our model or evolved the neurons during sleep, we might have the problem of runtime disconnected.



These issues made us not able to save time for training models. In our case, it took around 2 hours to train one epoch and another 2 hours to evolve one neuron. However, we had 4 sentiments in total and needed to try to optimise the model. In other words, it caused us some time, especially when the model was not successfully trained due to runtime disconnected..

We, nonetheless, could not use HKU CS GPU Farm as our dataset took up more than 2 GB and the updates between different accounts for things like the codes seemed to be not shared. As a result, we needed to rely on google colab.

Remaining issues:

1. Not enough training for the experiments

One training takes 1-2 hours to complete. As we have a lot of experiments, some test cases have not been trained with enough epochs. The trend could change with more epochs and our evaluation of the test result could be not accurate enough.

2. Accuracy of Classifier Model remained at 52.5

<https://drive.google.com/file/d/1CjYQB4o0r3Z-RLWBurGYBxC-O36-7AJY/view?usp=sharing>

This file was written when there was a new record of Test Accuracy when we trained the classifier. At first, the classifier kept on having Test Accuracy: 45. Then it increased to 52.5 after one more training. However, it has not improved the accuracy after that. Our data composition of vgmidi-master with 4 labeled sentiments shows that we did not have a balanced dataset for this multiclass classification problem, around (Sent 0: Sent 1: Sent 2: Sent 3 = 1:1.92:1.33:2.46). This introduced biases during data collection. As the model is not provided with the similar amount of data for each class, it may be more likely for the model to classify one class rather than another. Besides, the train set only contained 161 pieces in total, which may make the model underfitting, while the test set only contained 40 pieces..

References

- Radford's mLSTM+LR methodology: <https://arxiv.org/abs/1704.01444>
- Ferreira, Lucas N., Whitehead, Jim. (2019). Learning to Generate Music with Sentiment. Proceedings of the Conference of the International Society for Music Information Retrieval. ISMIR'19. Retrieved from: <http://www.lucasnferreira.com/papers/2019/ismir-learning.pdf>
- The Github page of the VGMIDI Dataset: <https://github.com/lucasnfe/vgmidi>
- More detailed implementation of music generation using LSTM: <https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5>
- Medium article on LSTM music generation (.wav files used): <https://medium.com/intel-student-ambassadors/music-generation-using-lstms-in-keras-9ded32835a8f>
- LSTM doc: https://keras.io/api/layers/recurrent_layers/lstm/#lstm-class
- Yang, Lurch. (2018). On the evaluation of generative models in music https://musicinformatics.gatech.edu/wp-content_nondefault/uploads/2018/11/postprint.pdf