Noam Azulay – 212988232

Yuval Gorodissky – 318963436

Deep Learning Assignment 3 Report

- All our source code, data and readme are available at: https://github.com/yuvalgorodissky/deep-learning
- The zip file includes the generated text for each song from the test set.

Preprocessing Steps Summary

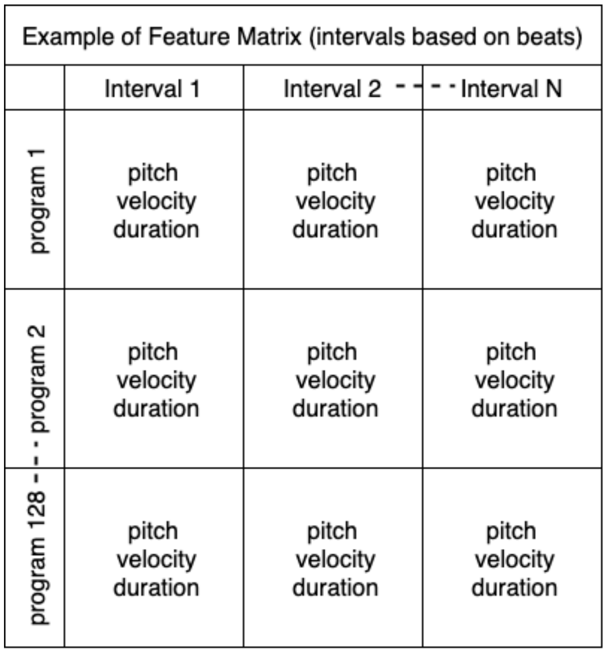
Feature Extraction from MIDI Files

First Extraction Method steps - instrument Feature Extraction Strategy

1. **Load MIDI File:** Use the pretty_midi library to load the MIDI file.
2. **Get Beat Times:** Extract beat times from the MIDI file. These beats serve as intervals for organizing the feature data.
3. **Initialize Feature Matrix:** Create a matrix of size 128 (number of programs) x 3 x number of beats to store features: pitch, velocity, and duration.
4. **Extract Features:**
    - Iterate through each instrument and its notes in the MIDI file.
    - For each note, extract pitch, start time, end time, velocity, and duration.
    - Find the closest beat index before the note starts.
    - Update the feature matrix at the corresponding program number and beat index with the extracted features.

5. **Reshape Features:** Reshape the feature matrix to a 2D tensor of size (128 * 3) x number of beats.

By using beats as intervals, we ensure that the extracted features are organized in a time-aligned manner, capturing detailed note information, including pitch, velocity, and duration, within the context of the music's beat structure.
At the end of the process we got a matrix look like:



Example of Feature Matrix (intervals based on beats)

	Interval 1	Interval 2 - -	- - Interval N
program 1	pitch velocity duration	pitch velocity duration	pitch velocity duration
program 2	pitch velocity duration	pitch velocity duration	pitch velocity duration
program 128 - - -	pitch velocity duration	pitch velocity duration	pitch velocity duration

Noam Azulay – 212988232

Yuval Gorodissky – 318963436

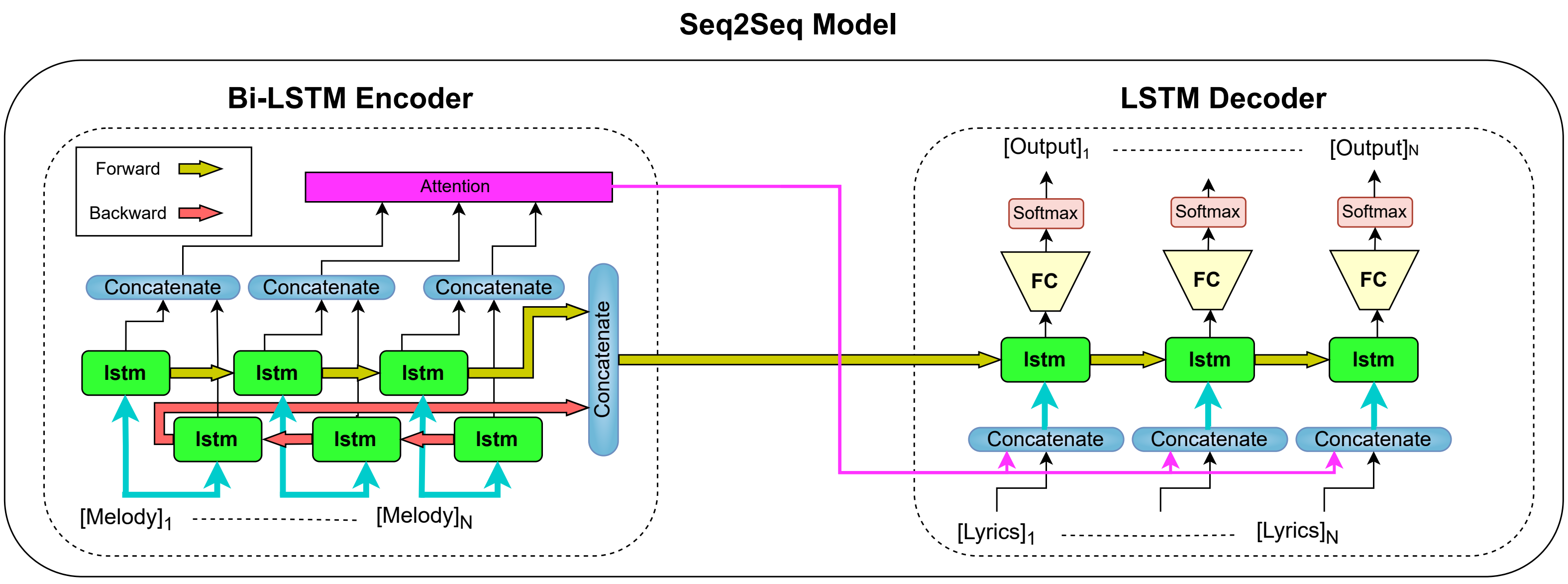
Second Extraction Method steps - Piano Roll Feature Extraction Strategy

This approach uses pretty_midi -get_piano_roll function, which outputs a matrix where rows represent different pitches and columns represent discrete time steps. Each entry in this matrix indicates the intensity (velocity) of a note at a given pitch and time, providing a detailed snapshot of the musical activity over time. By setting a specific sampling rate (**fs=2**).

Lyrics Feature Extraction:

1. **Load Lyrics:** Read lyrics from a CSV file containing columns for artist, song, and lyrics.
2. **Tokenize** Lyrics: Split the lyrics into individual words and convert them to lowercase to ensure consistency.
3. **Generate Word Embeddings:** For each word in the lyrics we retrieve its embedding using the word2vec model we choose 'word2vec-google-news-300' .If the word is not found in the vocabulary, use a predefined embedding for unknown words.
4. **Append EOS**: For each lyric we append (EOS) token to signify the end of the lyrics

Model architecture :



Noam Azulay – 212988232

Yuval Gorodissky – 318963436

1. **LSTM Encoder**: Encodes the input sequence into a fixed-size context vector and extracts sequential features across time, while handling long-range dependencies.
   Key Features:
   - Bi-directional LSTM layers capture information from both past and future contexts effectively.
   - Dropout is used as a regularization method to prevent overfitting.
2. **Attention Mechanism**: Computes the attention weights which indicate the importance of each input sequence element to each output sequence element.
   Key Features:
   - Combines the decoder's hidden state with each encoder output using a linear layer.
   - Applies a non-linear activation function (tanh) and then uses a parameter vector to convert the activated values into attention weights using a dot product.
   - Normalizes these weights across the sequence using the softmax function.

3. **LSTM Decoder**: Decodes the encoded information step-by-step to generate the output sequence. It integrates the attention mechanism to focus selectively on parts of the input sequence.
   Key Features:
   - Combines the output from the attention layer with the decoder's input to form the input for the LSTM layers in the decoder.
   - Utilizes the last hidden state of the decoder and all encoder outputs to compute attention weights.

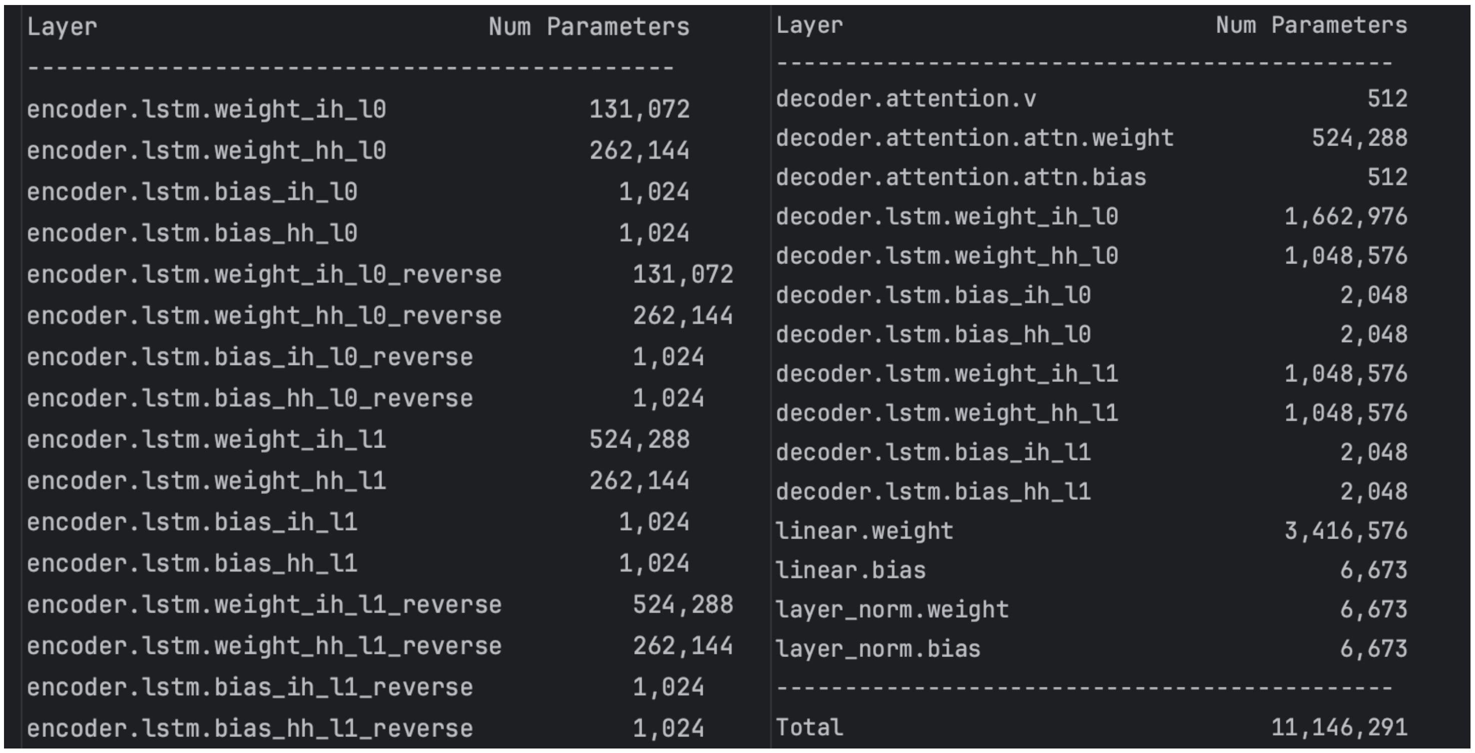
4. **LSTM Seq2Seq** - Manages the overall process of encoding the input sequence and decoding it into the output sequence, using the encoder and decoder components.
   Key Features:
   - Implements **teacher forcing** as a strategy to speed up training and improve the model’s ability to converge. Teacher forcing works by using the actual target outputs as each next input, instead of using the decoder’s guess as the next input.
   - Uses a linear transformation followed by layer normalization on the decoder's outputs.
   - Allows for different strategies for output selection during inference (greedy, probabilistic, top k).

Noam Azulay – 212988232

Yuval Gorodissky – 318963436

Model Parameters and Configuration

- **Input Size (Encoder):** The input size for the encoder is set to 384 (calculated as 128 * 3) or 128 depend on the melody strategy .
- **Hidden Size (Encoder):** The encoder's hidden size is 256. This parameter determines the dimensionality of the hidden state in the LSTM layers of the encoder we are using bidirectional so after concat is 512.
- **Input Size (Decoder):** For the decoder, the input size is set at 300. This size generally corresponds to the dimensionality of the embeddings used for the decoder inputs.
- **Hidden Size (Decoder):** Similar to the encoder, the decoder's hidden size is 512. This size impacts the decoder's ability to process and generate the output sequence based on the learned context.
- **Output Size (Decoder):** The vector size for the decoder outputs is determined by the length of the vocabulary, denoted as `vect_size_decoder`. This size is crucial as it defines the output space of the decoder, where each dimension corresponds to a word in the vocabulary.



```
Layer                               Num Parameters
---------------------------------------------
encoder.lstm.weight_ih_l0                  131,072
encoder.lstm.weight_hh_l0                  262,144
encoder.lstm.bias_ih_l0                      1,024
encoder.lstm.bias_hh_l0                      1,024
encoder.lstm.weight_ih_l0_reverse            131,072
encoder.lstm.weight_hh_l0_reverse            262,144
encoder.lstm.bias_ih_l0_reverse                1,024
encoder.lstm.bias_hh_l0_reverse                1,024
encoder.lstm.weight_ih_l1                  524,288
encoder.lstm.weight_hh_l1                  262,144
encoder.lstm.bias_ih_l1                      1,024
encoder.lstm.bias_hh_l1                      1,024
encoder.lstm.weight_ih_l1_reverse            524,288
encoder.lstm.weight_hh_l1_reverse            262,144
encoder.lstm.bias_ih_l1_reverse                1,024
encoder.lstm.bias_hh_l1_reverse                1,024
```

```
Layer                               Num Parameters
---------------------------------------------
decoder.attention.v                            512
decoder.attention.attn.weight              524,288
decoder.attention.attn.bias                    512
decoder.lstm.weight_ih_l0                1,662,976
decoder.lstm.weight_hh_l0                1,048,576
decoder.lstm.bias_ih_l0                      2,048
decoder.lstm.bias_hh_l0                      2,048
decoder.lstm.weight_ih_l1                1,048,576
decoder.lstm.weight_hh_l1                1,048,576
decoder.lstm.bias_ih_l1                      2,048
decoder.lstm.bias_hh_l1                      2,048
linear.weight                            3,416,576
linear.bias                                  6,673
layer_norm.weight                            6,673
layer_norm.bias                              6,673
---------------------------------------------
Total                                   11,146,291
```

Noam Azulay – 212988232

Yuval Gorodissky – 318963436

Training Configuration:

- **Number of Layers:** Both the encoder and decoder consist of 2 layers. This setting offers a balance between model complexity and computational efficiency.
- **Batch Size:** The batch size is set to 4
- **Learning Rate:** The learning rate is 0.001. This parameter controls the speed at which the model learns by defining the step size at each iteration while moving toward a minimum of the loss function.
- **Epochs:** The model is configured to train for 200 epochs.
- **Optimizer :** To optimize the model, we use the Adam optimizer
- **Teacher Forcing Ratio:** Set to 1, indicating that teacher forcing is used in every step of training.

Vocabulary Construction and Management - Data Processing and Tokenization:

1. **Reading Lyrics:** Each line of lyrics is tokenized into individual words. This tokenization process breaks down the lyrical content into its basic linguistic units.
2. **Filtering Words:** During the vocabulary construction, each tokenized word is checked against the available Word2Vec model. This ensures that the vocabulary only includes words that have corresponding embeddings in Word2Vec, which is critical for leveraging these pre-trained embeddings during model training.
3. **Special Tokens Addition:** To facilitate effective model training and generation, special tokens such as PAD_TOKEN (padding), EOS_TOKEN (end-of-sequence), SOS_TOKEN (start-of-sequence), and UNK_TOKEN (unknown) are added to the vocabulary. These tokens serve several essential functions:
    - **PAD_TOKEN:** Used to equalize the lengths of sequences within a batch.
    - **EOS_TOKEN and SOS_TOKEN:** Mark the beginning and end of sequences, aiding the model in recognizing the boundaries of lyrical content.
    - **UNK_TOKEN :** Represents words encountered during inference that are not present in the training vocabulary, allowing the model to handle unknown words gracefully.

Loss Function - Custom Cross-Entropy Loss Function

Loss Components:

1. **Standard Cross-Entropy Loss:** This component directly addresses the accuracy of the predicted tokens against the target tokens, emphasizing the model's performance in generating contextually and semantically correct words.
2. **Length Loss:** This is computed as the average absolute difference between the lengths of predicted and target sequences up to the <EOS> token. This component ensures that the model not only predicts accurate words but also generates outputs of appropriate lengths, aligning closely with the structure of the target sequence.
3. **Line Loss**: This measures the discrepancy in the count of `next_line` tokens between the predictions and the targets, reflecting how well the model learns and replicates the structural composition of the text(the number of lines).

Noam Azulay – 212988232

Yuval Gorodissky – 318963436

Experiment Setup

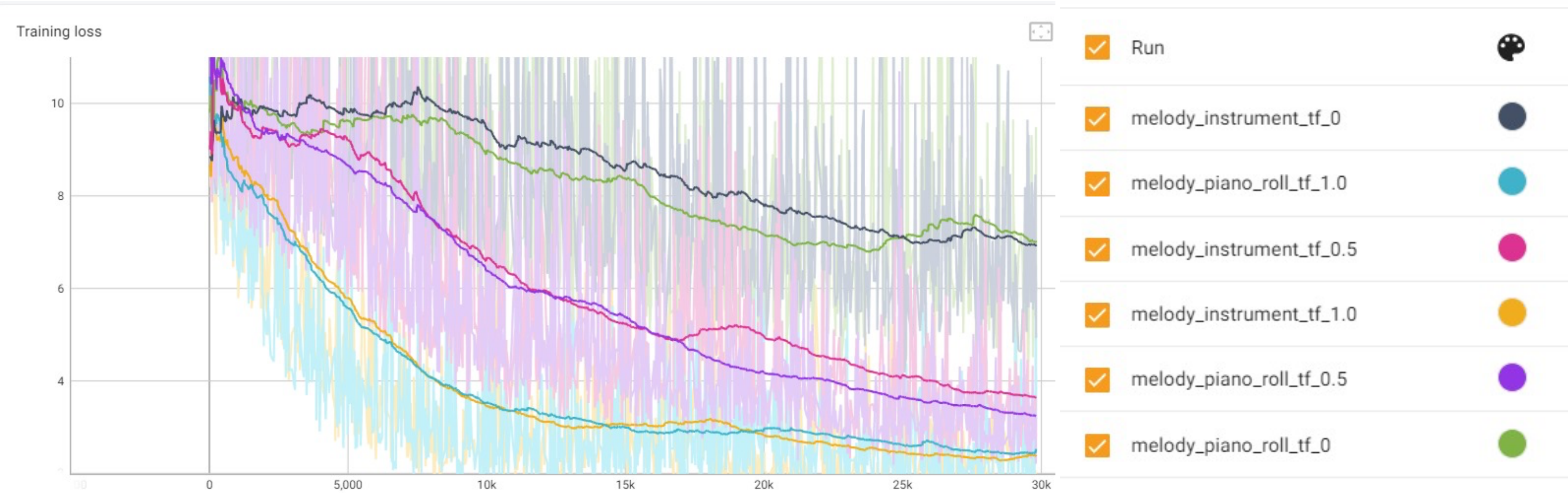
In the experiment to evaluate the effectiveness of different melody feature extraction strategies, we focused on two distinct approaches: extracting features via the piano roll representation and directly using instrument-specific data from MIDI files. This comparison aims to identify which strategy better captures the nuances necessary for accurate lyrics generation in a sequence-to-sequence modeling context.

Params:

- **Feature Extraction Strategies :** [Piano Roll, Instrument]
- **Teacher Forcing Rates:** [0 , 0.5, 1]
- **Epochs:** 200
- **Learning Rate:** 0.001
- **Batch Size:** 4
- **Optimizer :** Adam

These experiment settings are designed to rigorously test the two feature extraction strategies under varied training conditions, providing a comprehensive analysis of their effectiveness in enhancing the model's ability to generate lyrics accurately.

Tensorboard Loss Curves:



The TensorBoard graph illustrates three pairs of curves, each representing different models trained with teacher forcing values of 1.0, 0.5, and 0. The model with teacher forcing set to 1.0 shows the lowest training loss, followed by the model with 0.5, and finally the model with 0. This pattern indicates a clear reduction in training loss with higher teacher forcing values. Additionally, the feature extraction method for melodies did not significantly impact the model's training loss.

Noam Azulay – 212988232

Yuval Gorodissky – 318963436

Evolutions:

In this section, we evaluate the performance of our six models derived from experiments with:

- Varying teacher forcing rates -1, 0.5, 0
- Two melody feature extraction strategies - "piano roll", "instruments"

Each model generates lyrics using three prediction methods:

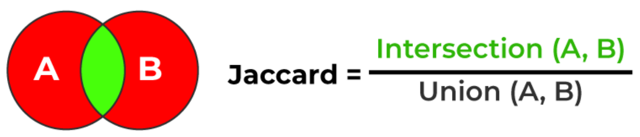
- Prob sampling - Samples from the predicted probability distribution of the next word.
- Argmax - Selects the word with the highest predicted probability.
- Top-k sampling - Samples from the top-k most likely words.

Furthermore, the models are initialized with three different starting tokens:

- Real first token - The actual first token of the target sequence.
- SOS_token - A special token representing the start of the sequence.
- Random token - A randomly selected token from the vocabulary.

The generated text is evaluated using Jaccard similarities :

- Jaccard similarity with n-gram=1
- Jaccard similarity with n-gram=2



Results:

Model	Strategy	Start_Token	Song-1_jacard_1	Song-1_jacard_2	Song-2_jacard_1	Song-2_jacard_2	Song-3_jacard_1	Song-3_jacard_2	Song-4_jacard_1	Song-4_jacard_2	Song-5_jacard_1	Song-5_jacard_
model instrument tf 0.5	prob	first_word	0.1267	0.0346	0.1553	0.0361	0.1345	0.0374	0.1132	0.0303	0.1059	0.0272
model instrument tf 0.5	prob	SOS	0.14	0.0207	0.1359	0.0152	0.1206	0.0357	0.1346	0.0352	0.1081	0.0293
model instrument tf 0.5	prob	random	0.1218	0.0264	0.1367	0.0157	0.1239	0.0422	0.0935	0.0267	0.1509	0.0208
model instrument tf 0.5	argmax	first_word	0.1972	0.0204	0.1042	0.0167	0.0674	0.0229	0.1163	0.038	0.0972	0.016
model instrument tf 0.5	argmax	SOS	0.1605	0.0258	0.1327	0.0251	0.1064	0.0338	0.0795	0.0186	0.1412	0.0292
model instrument tf 0.5	argmax	random	0.2078	0.0452	0.1273	0.032	0.069	0.0118	0.1333	0.0294	0.1159	0.0236
model instrument tf 0.5	topk	first_word	0.1771	0.0385	0.1714	0.0299	0.117	0.0317	0.1413	0.0343	0.1205	0.0241
model instrument tf 0.5	topk	SOS	0.1828	0.0421	0.1402	0.031	0.1296	0.0428	0.2021	0.03	0.1087	0.0215
model instrument tf 0.5	topk	random	0.2065	0.0577	0.1552	0.0327	0.1346	0.0424	0.1458	0.0219	0.1047	0.0245
model instrument tf 0	prob	first_word	0.1646	0.0392	0.1789	0.0217	0.1714	0.0377	0.1359	0.031	0.1444	0.0196
model instrument tf 0	prob	SOS	0.1139	0.013	0.1571	0.0378	0.1827	0.0433	0.1619	0.0383	0.1266	0.0321
model instrument tf 0	prob	random	0.1098	0.0187	0.1364	0.0288	0.1402	0.0295	0.1415	0.0493	0.1383	0.0296
model instrument tf 0	argmax	first_word	0.1034	0.033	0.0632	0.0113	0.0581	0.006	0.0833	0.0336	0.0469	0.0093
model instrument tf 0	argmax	SOS	0.0847	0.0337	0.0312	0.0057	0.0617	0.0252	0.0588	0.0197	0.0484	0.0196
model instrument tf 0	argmax	random	0.0678	0.0222	0.0538	0.012	0.0706	0.0179	0.093	0.0253	0.0323	0.0096
model instrument tf 0	topk	first_word	0.125	0.0273	0.1089	0.027	0.1348	0.0354	0.1667	0.0408	0.0959	0.0197
model instrument tf 0	topk	SOS	0.1061	0.0336	0.099	0.0218	0.186	0.0513	0.1099	0.0314	0.1029	0.0226

Noam Azulay – 212988232

Yuval Gorodissky – 318963436

model instrument tf 0	topk	random	0.125	0.0254	0.0962	0.0222	0.1136	0.0474	0.129	0.0309	0.1268	0.0265
model piano roll tf 1.0	prob	first_word	0.1067	0.0179	0.13	0.0269	0.0759	0.0191	0.1166	0.0218	0.0845	0.0177
model piano roll tf 1.0	prob	SOS	0.066	0.0085	0.0797	0.0172	0.0764	0.0037	0.0596	0.0124	0.0826	0.0174
model piano roll tf 1.0	prob	random	0.0455	0.0106	0.1058	0.0127	0.066	0.0106	0.1163	0.0136	0.1197	0.0159
model piano roll tf 1.0	argmax	first_word	0.1413	0.0258	0.2	0.0478	0.1984	0.0537	0.1215	0.0262	0.125	0.0182
model piano roll tf 1.0	argmax	SOS	0.0333	0.0119	0.0745	0.0435	0.1538	0.0343	0.101	0.0178	0.1314	0.008
model piano roll tf 1.0	argmax	random	0.1429	0.0341	0.051	0.0061	0.0889	0.0291	0.1204	0.0368	0.1304	0.0079
model piano roll tf 1.0	topk	first_word	0.1301	0.0446	0.1689	0.0662	0.1769	0.0565	0.176	0.0373	0.1053	0.0116
model piano roll tf 1.0	topk	SOS	0.1161	0.0158	0.1349	0.0442	0.1441	0.0417	0.0568	0.0068	0.1406	0.0178
model piano roll tf 1.0	topk	random	0.126	0.0226	0.1915	0.0405	0.1148	0.0346	0.1212	0.0114	0.1458	0.0127
model instrument tf 1.0	prob	first_word	0.1269	0.0314	0.127	0.0181	0.1126	0.0216	0.1355	0.0176	0.0941	0.0143
model instrument tf 1.0	prob	SOS	0.0959	0.027	0.1194	0.0407	0.0976	0.0314	0.1102	0.0202	0.0593	0.0159
model instrument tf 1.0	prob	random	0.0855	0.0123	0.1053	0.0164	0.0812	0.0215	0.0702	0.0159	0.0423	0.0091
model instrument tf 1.0	argmax	first_word	0.1724	0.0228	0.0938	0.0116	0.1188	0.0312	0.1261	0.0144	0.0896	0.0094
model instrument tf 1.0	argmax	SOS	0.0965	0.0105	0.07	0.0237	0.1443	0.0423	0.104	0.0188	0.1522	0.0364
model instrument tf 1.0	argmax	random	0.0758	0.0206	0.1465	0.0401	0.1721	0.0574	0.0714	0.0255	0.0949	0.016
model instrument tf 1.0	topk	first_word	0.1489	0.0245	0.0938	0.012	0.1538	0.0452	0.1078	0.0226	0.0909	0.0113
model instrument tf 1.0	topk	SOS	0.0984	0.0227	0.1346	0.0217	0.1475	0.0519	0.1111	0.0211	0.0333	0.0
model instrument tf 1.0	topk	random	0.1014	0.0097	0.1009	0.0262	0.05	0.0132	0.0916	0.0213	0.0959	0.025
model piano roll tf 0.5	prob	first_word	0.1231	0.0187	0.1667	0.0524	0.1545	0.0506	0.145	0.0403	0.1485	0.0266
model piano roll tf 0.5	prob	SOS	0.0857	0.0216	0.1382	0.0489	0.1321	0.0415	0.1389	0.0283	0.1188	0.0052
model piano roll tf 0.5	prob	random	0.1163	0.0232	0.082	0.0135	0.1351	0.038	0.1324	0.034	0.1207	0.0133
model piano roll tf 0.5	argmax	first_word	0.1304	0.0256	0.0978	0.0182	0.1059	0.0294	0.0581	0.0067	0.1205	0.0186
model piano roll tf 0.5	argmax	SOS	0.1667	0.036	0.086	0.0305	0.122	0.0359	0.069	0.0	0.0658	0.0138
model piano roll tf 0.5	argmax	random	0.1311	0.0202	0.0319	0.0121	0.0886	0.0261	0.0345	0.0066	0.0843	0.0189
model piano roll tf 0.5	topk	first_word	0.141	0.034	0.1842	0.0608	0.1529	0.0517	0.0989	0.012	0.1538	0.0316
model piano roll tf 0.5	topk	SOS	0.1176	0.0309	0.1553	0.0591	0.1765	0.0649	0.1413	0.0335	0.1176	0.0114
model piano roll tf 0.5	topk	random	0.1579	0.0194	0.0792	0.038	0.1087	0.0421	0.14	0.0209	0.1263	0.026
model piano roll tf 0	prob	first_word	0.1429	0.0175	0.2075	0.044	0.1667	0.0352	0.1261	0.0297	0.0957	0.0216
model piano roll tf 0	prob	SOS	0.15	0.0116	0.1636	0.0283	0.0901	0.0126	0.1635	0.028	0.0957	0.0052
model piano roll tf 0	prob	random	0.1481	0.0142	0.1304	0.0446	0.1143	0.0221	0.1495	0.0209	0.0889	0.0234
model piano roll tf 0	argmax	first_word	0.127	0.028	0.1236	0.0412	0.0633	0.0065	0.1084	0.0314	0.0769	0.0175
model piano roll tf 0	argmax	SOS	0.0984	0.037	0.087	0.0309	0.0375	0.0065	0.0909	0.0176	0.0806	0.0286
model piano roll tf 0	argmax	random	0.0769	0.0364	0.0538	0.0241	0.0247	0.0065	0.0909	0.0123	0.0781	0.0179
model piano roll tf 0	topk	first_word	0.1111	0.0126	0.1538	0.0392	0.1023	0.0103	0.1136	0.0267	0.1159	0.027
model piano roll tf 0	topk	SOS	0.1176	0.0329	0.1633	0.0318	0.0879	0.0259	0.1196	0.0202	0.1	0.0207
model piano roll tf 0	topk	random	0.1286	0.0316	0.1354	0.0457	0.069	0.0233	0.1505	0.0348	0.0845	0.0147

	n=1	n=2
avg tf 0	0.09806905	0.0225381
avg tf 0.5	0.11852037	0.02645185
avg tf 1.0	0.14437407	0.03957963

	n=1	n=2
avg piano roll	0.11442714	0.02554714
avg insrument	0.11654	0.02641926

Noam Azulay – 212988232

Yuval Gorodissky – 318963436

Conclusion

The evaluation reveals that models trained with higher teacher forcing rates tend to perform better across both metrics.

The choice of melody feature extraction (piano roll vs. instruments) does not significantly affect the performance, suggesting that either method can be effective depending on other model parameters and training settings.

The results are not as high as we had hoped, possibly due to several factors. Here are the most relevant ones:

- **Small Dataset:** The generative model was trained on a relatively small dataset, whereas high-quality text generation typically requires training on millions of files.
- **Complex Encoding:** Even for humans, generating text from a melody is a challenging task, which makes it difficult for an encoder to handle.
- **Similarity Metrics:** Evaluating the quality of generated text requires human assessment or embedding comparison for better understanding.