
Music VAE를 통한 미디 생성

작성자 : 김재훈

논문리뷰

A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music

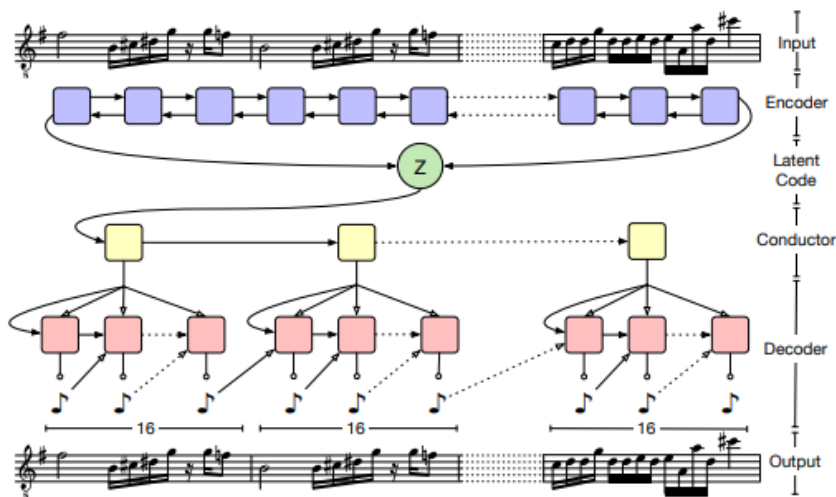
Adam Roberts, Jesse Engel, Colin Raffel,
Curtis Hawthorne, Douglas Eck

❖ 논문 개요

- ▶ Latent vector 들을 차원에 적절하게 분포 시키기 위한 VAE 구조 사용
- ▶ 기존 RNN 사용시 긴 데이터 구조에서 발생하는 Vanishing Gradient 문제를 해결하기 위해 계층적 디코더를 제안함
- ▶ 제안된 아키텍처 적용을 통해 기존보다 Sampling, Interpolation, reconstruction 성능이 향상됨을 발견함

❖ 제안된 네트워크 구조

- ▶ 긴 마디의 멜로디 생성을 위한 네트워크 구조
- ▶ Encoder, Latent Code, Conductor, Decoder



[논문에서 제안한 네트워크 구조]

```
CONFIG_MAP['hierdec-mel_16bar'] = Config(
    model=MusicVAE(
        lstm_models.BidirectionalLstmEncoder(),
        lstm_models.HierarchicalLstmDecoder(
            lstm_models.CategoricalLstmDecoder(),
            level_lengths=[16, 16],
            disable_autoregression=True)),
    hparams=merge_hparams(
        lstm_models.get_default_hparams(),
        HParams(
            batch_size=512,
            max_seq_len=256,
            z_size=512,
            enc_rnn_size=[2048, 2048],
            dec_rnn_size=[1024, 1024],
            free_bits=256,
            max_beta=0.2,
        )),
    note_sequence_augmenter=None,
    data_converter=mel_16bar_converter,
    train_examples_path=None,
    eval_examples_path=None,
)
```

[코드로 구현된 네트워크 구조와 파라미터]

❖ 제안된 네트워크 구조

▶ Encoder

- 2048 크기를 갖는 양방향 2 Layer-LSTM으로 구성
- 순방향 역방향에 대한 LSTM 네트워크 레이어를 각 2개씩 생성
- stack_bidirectional_dynamic_rnn을 통한 두개의 LSTM 레이어를 stacking
- 순방향과 역방향을의 마지막 weight를 concatenate하여 출력값으로 보냄

```
def build_bidirectional_lstm(
    layer_sizes, dropout_keep_prob, residual, is_training):
    """Build the Tensorflow graph for a bidirectional LSTM."""

    cells_fw = []
    cells_bw = []

    for layer_size in layer_sizes:
        cells_fw.append(
            rnn_cell([layer_size], dropout_keep_prob, residual, is_training))
        cells_bw.append(
            rnn_cell([layer_size], dropout_keep_prob, residual, is_training))

    return cells_fw, cells_bw
```

[양방향 LSTM 빌드 코드]

```
def encode(self, sequence, sequence_length):
    cells_fw, cells_bw = self.cells

    .. states_fw, states_bw = contrib_rnn.stack_bidirectional_dynamic_rnn(
        cells_fw,
        cells_bw,
        sequence,
        sequence_length=sequence_length,
        time_major=False,
        dtype=tf.float32,
        scope=self.name_or_scope)

    # Note we access the outputs (h) from the states since the backward
    # outputs are reversed to the input order in the returned outputs.
    last_h_fw = states_fw[-1][-1].h
    last_h_bw = states_bw[-1][-1].h

    return tf.concat([last_h_fw, last_h_bw], 1)
```

[양방향 LSTM을 활용한 인코더 코드]

❖ 제안된 네트워크 구조

▶ Latent Code

- Encoder의 출력과 Dense Layer를 활용한 Latent Vector를 생성
- Dense Layer와 softplus 활성화함수를 통해 Encoder의 출력에 대한 시그마값을 생성
- Latent Vector의 확률분포와 MultiVariateNormalDiag에 의한 확률분포의 KL Divergence 계산을 통한 확률분포 차이 계산

```
def encode(self, sequence, sequence_length, control_sequence=None):
    """
    hparams = self.hparams
    z_size = hparams.z_size

    sequence = tf.to_float(sequence)
    if control_sequence is not None:
        control_sequence = tf.to_float(control_sequence)
        sequence = tf.concat([sequence, control_sequence], axis=1)
    encoder_output = self.encoder.encode(sequence, sequence_length)

    mu = tf.layers.dense(
        encoder_output,
        z_size,
        name='encoder/mu',
        kernel_initializer=tf.random_normal_initializer(stddev=0.001))
    sigma = tf.layers.dense(
        encoder_output,
        z_size,
        activation=tf.nn.softplus,
        name='encoder/sigma',
        kernel_initializer=tf.random_normal_initializer(stddev=0.001))

    return ds.MultivariateNormalDiag(loc=mu, scale_diag=sigma)
```

[Latent Vector 생성 코드]

```
if hparams.z_size: # vae mode:
    q_z = self.encode(input_sequence, x_length, control_sequence)
    z = q_z.sample()

    # Prior distribution.
    p_z = ds.MultivariateNormalDiag(
        loc=[0.] * hparams.z_size, scale_diag=[1.] * hparams.z_size)

    # KL Divergence (nats)
    kl_div = ds.kl_divergence(q_z, p_z)
```

[KL Divergence 계산 코드]

❖ 제안된 네트워크 구조

▶ Hierarchical Decoder (Conductor, Decoder)

- Conductor : 512인풋의 1024출력을 가지는 단방향 LSTM 레이어로 구성
Latent vector를 입력으로 가지지만 학습시에는 Latent vector를 0으로 초기화하여 재귀적으로 업데이트함
- Decoder : 1024 크기를 갖는 단방향 2 Layer-LSTM으로 구성

```
def build(self, hparams, output_depth, is_training=True):
    self.hparams = hparams
    self.output_depth = output_depth
    self.total_length = hparams.max_seq_len
    if self.total_length != np.prod(self.level_lengths):
        raise ValueError(
            'The product of the HierarchicalLstmDecoder level lengths (%d) must '
            'equal the padded input sequence length (%d).' % (
                np.prod(self.level_lengths), self.total_length))
    tf.logging.info('\nHierarchical Decoder:\n'
                    '  input length: %d\n'
                    '  level output lengths: %s\n',
                    self.total_length,
                    self.level_lengths)

    self.hier_cells = [
        lstm_utils.rnn_cell(
            hparams.dec_rnn_size,
            dropout_keep_prob=hparams.dropout_keep_prob,
            residual=hparams.residual_decoder)
        # Subtract 1 for the core decoder level
        for _ in range(len(self.level_lengths) - 1)]

    with tf.variable_scope('core_decoder', reuse=tf.AUTO_REUSE):
        self_core_decoder.build(hparams, output_depth, is_training)
```

[Hierarchical Decoder 빌드 코드]

```
def recursive_decode(initial_input, path=None):
    """Recursive hierarchical decode function."""
    path = path or []
    level = len(path)

    if level == num_levels:
        with tf.variable_scope('core_decoder', reuse=tf.AUTO_REUSE):
            return base_decode_fn(initial_input, path)

    scope = tf.VariableScope(tf.AUTO_REUSE, 'decoder/hierarchical_level_%d' % level)

    num_steps = self.level_lengths[level]

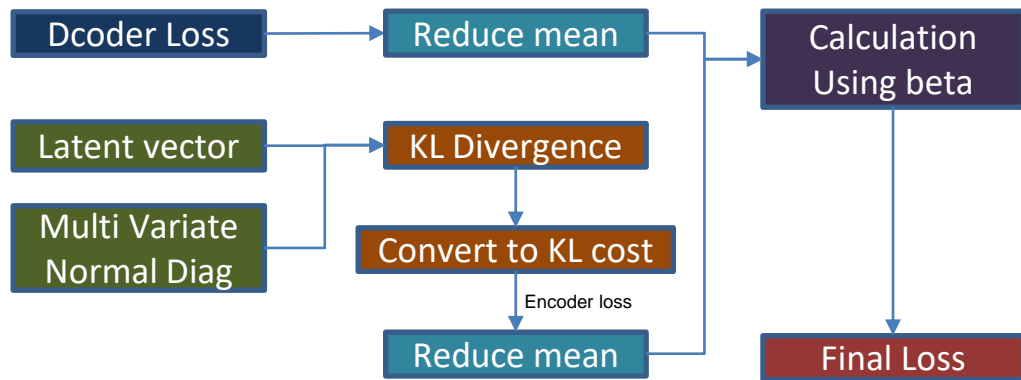
    with tf.variable_scope(scope):
        state = lstm_utils.initial_cell_state_from_embedding(
            self.hier_cells[level], initial_input, name='initial_state')
```

[Conductor 초기화 코드]

❖ 제안된 네트워크 구조

▶ 최종 Loss 계산

- Decoder의 결과 Loss와 Encoder & Latent vector 결과의 KL 확률분포의 차이를 Reduce_mean하여 계산 -> 최종 Loss 산출



[최종 Loss 계산 도식]

```

if hparams.z_size: # vae mode:
    q_z = self.encode(input_sequence, x_length, control_sequence)
    z = q_z.sample()

    # Prior distribution.
    p_z = ds.MultivariateNormalDiag(
        loc=[0.] * hparams.z_size, scale_diag=[1.] * hparams.z_size)

    # KL Divergence (nats)
    kl_div = ds.kl_divergence(q_z, p_z)

    # Concatenate the Z vectors to the inputs at each time step.
    else: # unconditional, decoder-only generation

r_loss, metric_map = self.decoder.reconstruction_loss(
    x_input, x_target, x_length, z, control_sequence)[0:2]

free_nats = hparams.free_bits * tf.math.log(2.0)
kl_cost = tf.maximum(kl_div - free_nats, 0)

beta = ((1.0 - tf.pow(hparams.beta_rate, tf.to_float(self.global_step)))
        * hparams.max_beta)
self.loss = tf.reduce_mean(r_loss) + beta * tf.reduce_mean(kl_cost)

scalars_to_summarize = {
    'loss': self.loss,
    'losses/r_loss': r_loss,
    'losses/kl_loss': kl_cost,
    'losses/kl_bits': kl_div / tf.math.log(2.0),
    'losses/kl_beta': beta,
}
return metric_map, scalars_to_summarize
  
```

[최종 Loss 계산 코드]

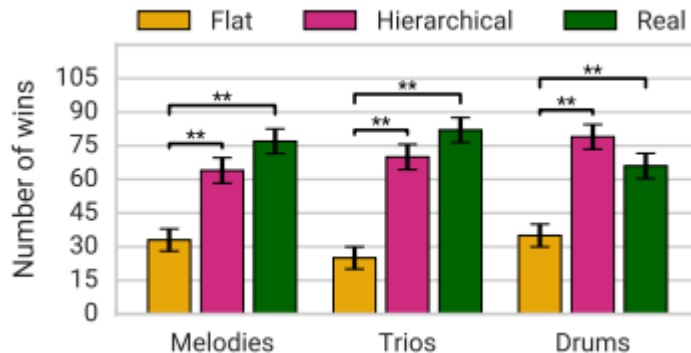
❖ 제안된 네트워크의 실험 결과

▶ 실험결과

- 긴 데이터 생성시 확실한 성능 향상을 보여주며, 생성된 파일의 음악적인 뉘앙스도 향상된것을 확인할 수 있음

Model	Teacher-Forcing		Sampling	
	Flat	Hierarchical	Flat	Hierarchical
2-bar Drum	0.979	-	0.917	-
2-bar Melody	0.986	-	0.951	-
16-bar Melody	0.883	0.919	0.620	0.812
16-bar Drum	0.884	0.928	0.549	0.879
Trio (Melody)	0.796	0.848	0.579	0.753
Trio (Bass)	0.829	0.880	0.565	0.773
Trio (Drums)	0.903	0.912	0.641	0.863

[제안된 네트워크를 활용한 실험 결과]



[청취자를 통한 음악성 실험 결과]

과제 구현

❖ 과제 목표

- ▶ 논문에서 제안한 아키텍처인 Hierarchical Latent Vector Model 구조를 활용한 MusicVAE와 Groove MIDI Dataset을 활용하여 4마디에 해당하는 드럼샘플을 뽑아내는 과정을 수행하는 것



❖ 실험 환경

- ▶ CPU : INTEL CORE i9 - 12900K
- ▶ GPU : Geforce RTX 2080TI X 1EA
- ▶ RAM : 32GB RAM
- ▶ OS : Ubuntu 18.04

❖ 실험 모델 구성

- ▶ MusicVAE 모델을 활용한 드럼샘플 생성기
- ▶ GrooVAE 모델을 활용한 드럼샘플 생성기
- ▶ MusicVAE 모델을 활용한 피아노샘플 생성기
- ▶ 실험 결과에 따른 개선 모델

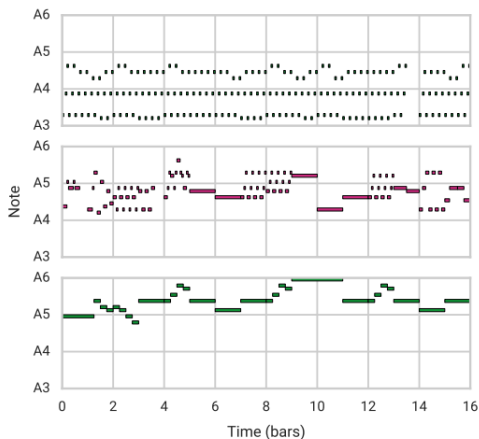
❖ 실험 데이터 구성

- ▶ 사용 데이터 셋 : Groove MIDI Dataset
- ▶ 데이터 형식 : MIDI
- ▶ 총 파일 개수 : 1,150개
- ▶ 총 파일 시간 : 약 814분 분량
- ▶ 곡 스타일 : 18개의 대분류 스타일로 구성
- ▶ 비트 타입 : 2가지로 구성 (Beat, Fill)

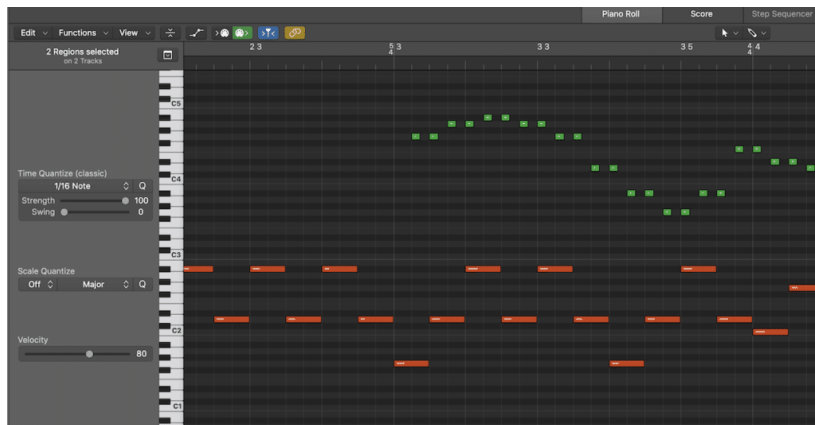
❖ 데이터 구조

▶ MIDI 데이터의 구조

- 곡의 템포 정보 및 악기 정보가 포함됨
- 각 노트의 발생 여부, 피치, 세기, 시작 및 끝 시간 등의 정보가 포함됨



[논문에서의 미디 파일 표기]

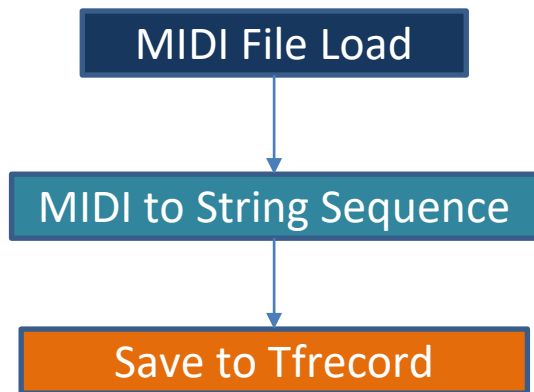


[DAW에서의 미디파일을 불러왔을 때의 예]

❖ 데이터 전처리

▶ MIDI 데이터 전처리 과정 - 1

- MIDI 파일을 스트링 시퀀스로 컨버팅하여 Tfrecored로 저장



[MIDI to String 컨버팅 과정]

```

tempos {
  qpm: 100.0
}
notes {
  pitch: 22
  velocity: 113
  start_time: 0.01625
  end_time: 0.11750000000000001
  is_drum: true
}
notes {
  pitch: 36
  velocity: 65
  start_time: 0.01625
  end_time: 0.11750000000000001
  is_drum: true
}
  
```

[미디 파일을 스트링 시퀀스로 컨버팅한 예]

❖ 데이터 전처리

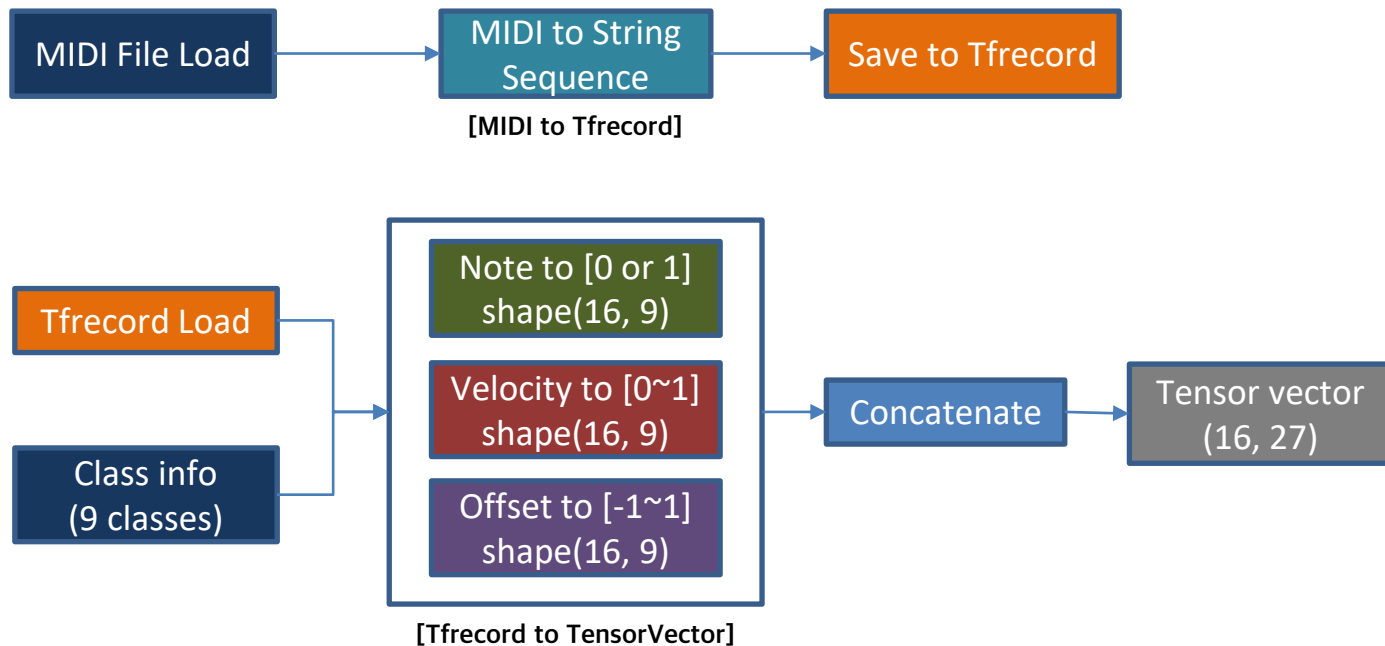
▶ MIDI 데이터 전처리 과정 - 2

- GrooveConverter 사용
- String Sequence를 Tensor Vector로 컨버팅
- 컨버팅 과정 (data.py의 GrooveConverter클래스를 참조함)

1. REDUCED_DRUM_PITCH_CLASSES 옵션을 사용하였기 때문에 9개의 클래스 사용 (kick, snare, closed_hh, open_hh, low_tom, mid_tom, hi_tom, crash, ride)
2. 마디당 16개의 노트와 각 노트당 9개의 클래스로 구성된 스페이스를 사용
3. 피치 정보는 각 9개 클래스에 fit하게 재 맵핑
4. Note 발생 여부를 0 또는 1로 구성 -> Shape(16, 9)
5. Velocity의 경우 0~127 -> 0~1 사이의 구성으로 노멀라이즈 -> Shape(16, 9)
6. Offsets의 경우 발생 노트의 시작타임을 퀀타이제이션하여 제일 가까운 마디로 마디를 맞추고 -0.5 ~ 0.5 -> -1 ~ 1로 리스케일링을 진행함 -> Shape(16, 9)
7. Note 여부, Velocity, Offset을 Concatenate하여 최종 Vector로 구성 -> Shape(16, 27)

❖ 데이터 전처리

▶ MIDI 데이터 전처리 과정 도식



❖ 데이터 생성

▶ 학습된 모델을 활용한 MIDI 데이터 생성 과정 - 1

- MIDI To TensorVector와 정반대의 진행과정

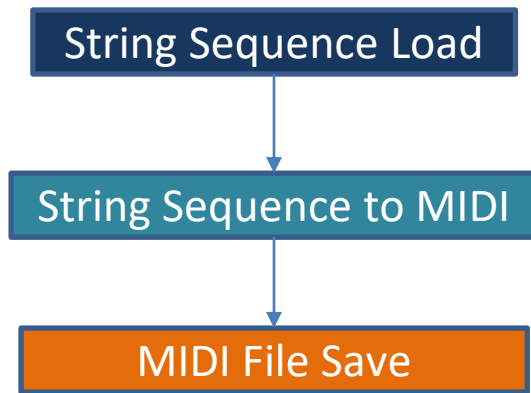
- Tensor Vector Result를 String Sequence로 컨버팅 및 MIDI 파일 생성

- 컨버팅 과정 (data.py의 GrooveConverter클래스를 참조함)

1. 학습된 모델의 결과로 발생하는 Tensor Vector Result -> Shape (16, 27)
2. 결과에 따른 Timesteps, Tempo, 비트길이, 스텝길이 생성 (Tempo는 120으로 고정됨)
3. 결과 텐서를 각각 Hit, Velocity, Offsets으로 분리 -> Shape (16, 9) 3개로 분리됨
4. Velocity의 경우 0~127로 리스케일링
5. Offset의 경우 -0.5~0.5로 리스케일링하여 시작시간과 끝시간을 계산함, 끝시간은 시작시간 + 스텝길이를 계산함
6. 9개의 클래스 결과를 MIDI Pitch로 리맵핑 진행
7. Hit의 결과가 0.5 이상의 경우 Note Sequence에 Note를 추가하며, Velocity, Offset, Pitch정보를 같이 저장

❖ 데이터 생성

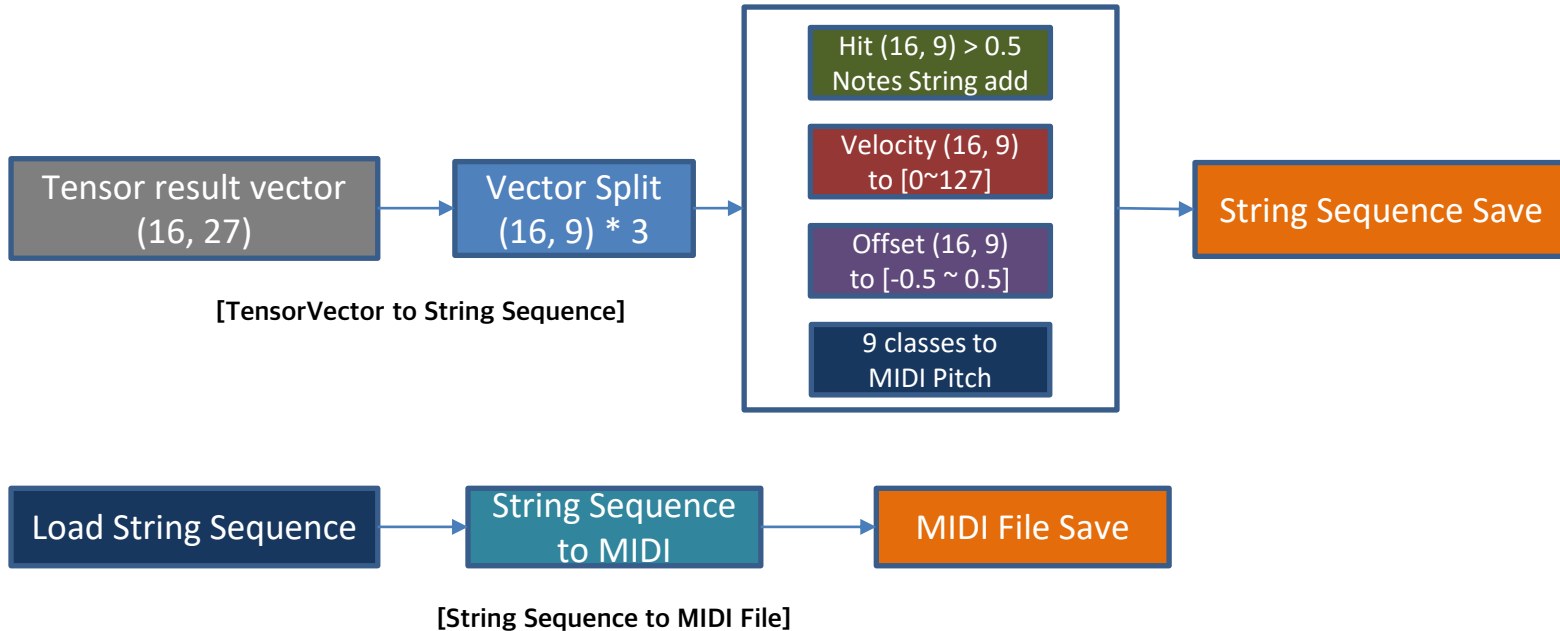
- ▶ 학습된 모델을 활용한 MIDI 데이터 생성 과정 - 2
 - 스트링 시퀀스를 MIDI 파일로 컨버팅하여 저장



[String to MIDI 컨버팅 과정]

❖ 데이터 생성

▶ 학습된 모델을 활용한 MIDI 데이터 생성 과정 도식



❖ MusicVAE 모델을 활용한 드럼샘플 생성기

▶ 모델 구성

- 논문에서 제안한 Hierarchical LSTM Decoder을 사용함
- 논문에서 요구하는 하이퍼 파라미터를 적용하여 학습시 OOM문제 발생
- 따라서 GrooVAE 하이퍼 파라미터 구성을 참조, 파라미터를 재조정하여 학습함

```
CONFIG_MAP['musicvae_4bar_paper_org'] = Config(
    model=MusicVAE(lstm_models.BidirectionalLstmEncoder(),
        lstm_models.HierarchicalLstmDecoder(
            core_decoder = lstm_models.CategoricalLstmDecoder(),
            level_lengths=[16, 16],
            disable_autoregression=True)),
    hparams=merge_hparams(
        lstm_models.get_default_hparams(),
        HParams(
            batch_size=512,
            max_seq_len=256, # 4 bars w/ 16 steps per bar ### 256
            z_size=512,
            enc_rnn_size=[2048, 2048], ### 2048, 2048
            dec_rnn_size=[1024, 1024], ### 1024, 1024
            max_beta=0.2,
            free_bits=256, ### 256
        )),
    note_sequence_augmenter=None,
    data_converter=data.GrooveConverter(
        split_bars=4, steps_per_quarter=4, quarters_per_bar=4,
        max_tensors_per_notesequence=20,
        pitch_classes=data.ROLAND_DRUM_PITCH_CLASSES,
        inference_pitch_classes=data.REDUCED_DRUM_PITCH_CLASSES),
    tfds_name='groove/groovae_4bar_paper',
)
```

[논문에서 명시된 하이퍼 파라미터 적용모델]



```
CONFIG_MAP['musicvae_4bar_paper_reduceunit'] = Config(
    model=MusicVAE(lstm_models.BidirectionalLstmEncoder(),
        lstm_models.HierarchicalLstmDecoder(
            core_decoder = lstm_models.CategoricalLstmDecoder(),
            level_lengths=[8, 8],
            disable_autoregression=True)),
    hparams=merge_hparams(
        lstm_models.get_default_hparams(),
        HParams(
            batch_size=512,
            max_seq_len=16 * 4, # 4 bars w/ 16 steps per bar ### 256
            z_size=256,
            enc_rnn_size=[512], ## 2048, 2048
            dec_rnn_size=[256, 256], ### 1024, 1024
            max_beta=0.2,
            free_bits=48, ## 256
        )),
    note_sequence_augmenter=None,
    data_converter=data.GrooveConverter(
        split_bars=4, steps_per_quarter=4, quarters_per_bar=4,
        max_tensors_per_notesequence=20,
        pitch_classes=data.ROLAND_DRUM_PITCH_CLASSES,
        inference_pitch_classes=data.REDUCED_DRUM_PITCH_CLASSES),
    tfds_name='groove/groovae_4bar_paper',
)
```

[하이퍼 파라미터 조절 후의 모델]

❖ MusicVAE 모델을 활용한 드럼샘플 생성기

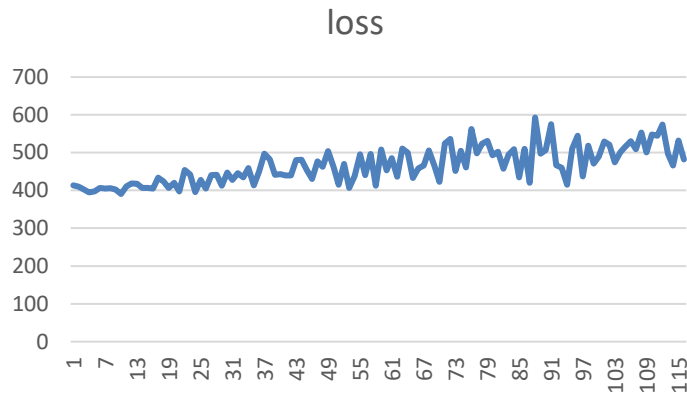
▶ 모델 학습 결과 (Loss)

- Loss가 정상적으로 수렴하지 않는 현상이 발생함

```

2022-12-08 19:37:45.499785 {'global_step': 1500, 'loss': 404.56555}
2022-12-08 19:38:16.078867 {'global_step': 1600, 'loss': 433.5907}
2022-12-08 19:38:41.952162 {'global_step': 1700, 'loss': 423.32022}
2022-12-08 19:39:12.814201 {'global_step': 1800, 'loss': 406.69147}
2022-12-08 19:39:38.184088 {'global_step': 1900, 'loss': 420.4788}
2022-12-08 19:40:08.420905 {'global_step': 2000, 'loss': 397.63693}
2022-12-08 19:40:34.406170 {'global_step': 2100, 'loss': 453.93857}
2022-12-08 19:41:05.136708 {'global_step': 2200, 'loss': 442.0014}
2022-12-08 19:41:31.138173 {'global_step': 2300, 'loss': 395.34402}
2022-12-08 19:42:01.856173 {'global_step': 2400, 'loss': 427.68744}
2022-12-08 19:42:27.639652 {'global_step': 2500, 'loss': 404.72083}
2022-12-08 19:42:57.671283 {'global_step': 2600, 'loss': 440.6816}
2022-12-08 19:43:23.177979 {'global_step': 2700, 'loss': 441.5909}
2022-12-08 19:43:53.550373 {'global_step': 2800, 'loss': 412.12387}
2022-12-08 19:44:19.237989 {'global_step': 2900, 'loss': 446.85895}
2022-12-08 19:44:45.189608 {'global_step': 3000, 'loss': 427.8056}
2022-12-08 19:45:15.514716 {'global_step': 3100, 'loss': 445.20685}
2022-12-08 19:45:41.206355 {'global_step': 3200, 'loss': 434.45682}
  
```

[학습간 Loss 결과 로그]



[학습간 Loss 결과 그래프]

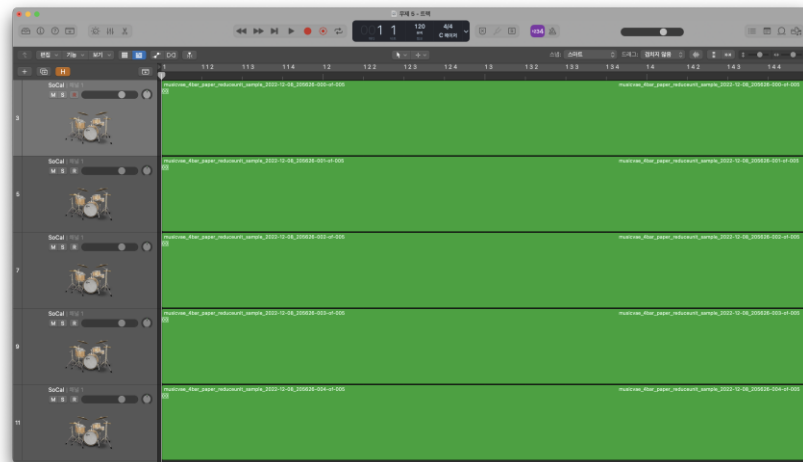
❖ MusicVAE 모델을 활용한 드럼샘플 생성기

▶ 모델 학습 결과 (MIDI 생성 및 재생)

- 정상적으로 미디파일이 생성되었으며, DAW에 정상적으로 로드 하였으나 4마디 드럼샘플이 제대로 생성되지 않은 것을 확인함

🎵 musicvae_4bar_paper_reduceunit_sample_2022-12-08_205626-000-of-005.mid
 🎵 musicvae_4bar_paper_reduceunit_sample_2022-12-08_205626-001-of-005.mid
 🎵 musicvae_4bar_paper_reduceunit_sample_2022-12-08_205626-002-of-005.mid
 🎵 musicvae_4bar_paper_reduceunit_sample_2022-12-08_205626-003-of-005.mid
 🎵 musicvae_4bar_paper_reduceunit_sample_2022-12-08_205626-004-of-005.mid

[생성된 MIDI파일 리스트]



[DAW에서의 MIDI파일 로드]

❖ GrooVAE 모델을 활용한 드럼샘플 생성기

▶ 모델 구성

- HierarchicalLstmDecoder 구성이 GrooveLstmDecoder로 변경되었으며, Dropout이 추가됨
- 하이퍼 파라미터의 경우 이전 실험과 최대한 동일하게 적용하여 학습 진행

```
CONFIG_MAP['musicvae_4bar_paper_reduceunit'] = Config(
    model=MusicVAE(lstm_models.BidirectionalLstmEncoder(),
        lstm_models.HierarchicalLstmDecoder(
            core_decoder=lstm_models.CategoricalLstmDecoder(),
            level_lengths=[8, 8],
            disable_autoregression=True)),
    hparams=merge_hparams(
        lstm_models.get_default_hparams(),
        HParams(
            batch_size=512,
            max_seq_len=16 * 4, # 4 bars w/ 16 steps per bar ## 256
            z_size=256,
            enc_rnn_size=[512], ## 2048, 2048
            dec_rnn_size=[256, 256], ## 1024, 1024
            max_beta=0.2,
            free_bits=48, ## 256
        )),
    note_sequence_augmenter=None,
    data_converter=data.GrooveConverter(
        split_bars=4, steps_per_quarter=4, quarters_per_bar=4,
        max_tensors_per_notesequence=20,
        pitch_classes=data.ROLAND_DRUM_PITCH_CLASSES,
        inference_pitch_classes=data.REDUCED_DRUM_PITCH_CLASSES),
    tfds_name='groove/groovae_4bar_paper',
)
```

[실험용 MusicVAE 모델]



```
CONFIG_MAP['groovae_4bar'] = Config(
    model=MusicVAE(lstm_models.BidirectionalLstmEncoder(),
        lstm_models.GrooveLstmDecoder()),
    hparams=merge_hparams(
        lstm_models.get_default_hparams(),
        HParams(
            batch_size=512,
            max_seq_len=16 * 4, # 4 bars w/ 16 steps per bar
            z_size=256,
            enc_rnn_size=[512],
            dec_rnn_size=[256, 256],
            max_beta=0.2,
            free_bits=48,
            dropout_keep_prob=0.3,
        )),
    note_sequence_augmenter=None,
    data_converter=data.GrooveConverter(
        split_bars=4, steps_per_quarter=4, quarters_per_bar=4,
        max_tensors_per_notesequence=20,
        pitch_classes=data.ROLAND_DRUM_PITCH_CLASSES,
        inference_pitch_classes=data.REDUCED_DRUM_PITCH_CLASSES),
    tfds_name='groove/4bar-midionly',
)
```

[GrooVAE 모델]

❖ GrooVAE 모델을 활용한 드럼샘플 생성기

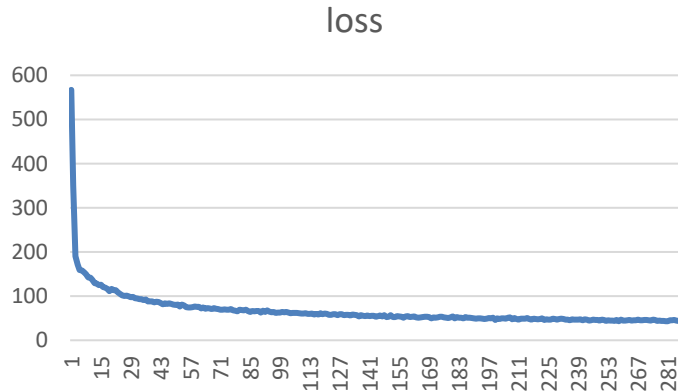
▶ 모델 학습 결과 (Loss)

- Loss가 적절히 수렴하는 그래프를 보임

```

2022-12-08 21:38:08.696601 {'global_step': 26700, 'loss': 45.156902}
2022-12-08 21:38:25.598927 {'global_step': 26800, 'loss': 45.022232}
2022-12-08 21:38:41.906256 {'global_step': 26900, 'loss': 45.54123}
2022-12-08 21:38:58.036237 {'global_step': 27000, 'loss': 46.0045}
2022-12-08 21:39:15.090613 {'global_step': 27100, 'loss': 44.75186}
2022-12-08 21:39:31.136044 {'global_step': 27200, 'loss': 44.045895}
2022-12-08 21:39:47.150473 {'global_step': 27300, 'loss': 46.870502}
2022-12-08 21:40:03.310964 {'global_step': 27400, 'loss': 46.0117}
2022-12-08 21:40:20.294549 {'global_step': 27500, 'loss': 43.349464}
2022-12-08 21:40:36.579751 {'global_step': 27600, 'loss': 44.245388}
2022-12-08 21:40:52.775200 {'global_step': 27700, 'loss': 43.594513}
2022-12-08 21:41:08.845664 {'global_step': 27800, 'loss': 43.92478}
2022-12-08 21:41:25.790991 {'global_step': 27900, 'loss': 42.92893}
2022-12-08 21:41:41.807794 {'global_step': 28000, 'loss': 43.23259}
2022-12-08 21:41:57.792563 {'global_step': 28100, 'loss': 45.258076}
2022-12-08 21:42:14.715215 {'global_step': 28200, 'loss': 45.0357}
2022-12-08 21:42:30.777812 {'global_step': 28300, 'loss': 45.65116}
2022-12-08 21:42:46.880809 {'global_step': 28400, 'loss': 44.540585}
2022-12-08 21:43:02.935018 {'global_step': 28500, 'loss': 43.96586}
2022-12-08 21:43:19.997351 {'global_step': 28600, 'loss': 42.42533}
  
```

[학습간 Loss 결과 로그]



[학습간 Loss 결과 그래프]

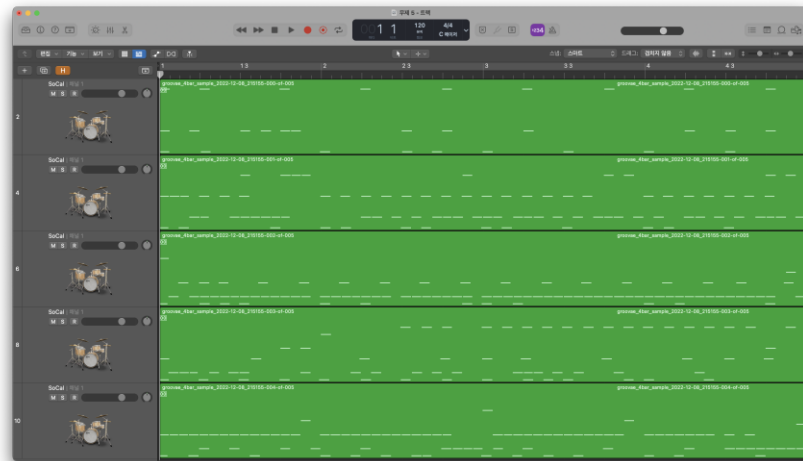
❖ GrooVAE 모델을 활용한 드럼샘플 생성기

▶ 모델 학습 결과 (MIDI 생성 및 재생)

- 정상적으로 미디파일이 생성되었으며, DAW에 로드 및 재생하여 4마디 드럼샘플이 제대로 생성되는 것을 확인함

🎵 groovae_4bar_sample_2022-12-08_215155-000-of-005.mid
 🎵 groovae_4bar_sample_2022-12-08_215155-001-of-005.mid
 🎵 groovae_4bar_sample_2022-12-08_215155-002-of-005.mid
 🎵 groovae_4bar_sample_2022-12-08_215155-003-of-005.mid
 🎵 groovae_4bar_sample_2022-12-08_215155-004-of-005.mid

[생성된 MIDI파일 리스트]



[DAW에서의 MIDI파일 로드]

❖ MusicVAE 모델을 활용한 피아노샘플 생성기

▶ 개요

- 논문에서 제안한 모델이 Groove Dataset에서 좋지 않은 성능을 보여줌
- 성능 저하의 원인 파악을 위해 위해 동일한 파라미터로 4마디 피아노 멜로디 생성 실험을 진행함
- Groove데이터셋과 비슷한 양의 데이터를 구성하여 실험 진행

▶ 실험 데이터 구성

- 사용 데이터 셋 : Maestro v3.0.0 Dataset 2015
- 데이터 형식 : MIDI
- 총 파일 개수 : 129개
- 총 파일 시간 : 857분 분량

```
CONFIG_MAP['musicvae_4bar_paper_reduceunit'] = Config(
    model=MusicVAE(lstm_models.BidirectionalLstmEncoder(),
        lstm_models.HierarchicalLstmDecoder(
            core_decoder = lstm_models.CategoricalLstmDecoder(),
            level_lengths=[8, 8],
            disable_autoregression=True)),
    hparams=merge_hparams(
        lstm_models.get_default_hparams(),
        HParams(
            batch_size=512,
            max_seq_len=16 * 4, # 4 bars w/ 16 steps per bar ## 256
            z_size=256,
            enc_rnn_size=[512], ## 2048, 2048
            dec_rnn_size=[256, 256], ## 1024, 1024
            max_beta=0.2,
            free_bits=48, ## 256
        )),
    note_sequence_augmenter=None,
    data_converter=data.GrooveConverter(
        split_bars=4, steps_per_quarter=4, quarters_per_bar=4,
        max_tensors_per_note_sequence=20,
        pitch_classes=data.ROLAND_DRUM_PITCH_CLASSES,
        inference_pitch_classes=data.REDUCED_DRUM_PITCH_CLASSES),
    tfds_name='groove/groovae_4bar_paper',
)
```

[Groove Data 활용 모델 구성]

```
mel_4bar_converter = data.OneHotMelodyConverter(
    skip_polyphony=False,
    max_bars=100, # Truncate long melodies before slicing.
    slice_bars=4,
    steps_per_quarter=4)

CONFIG_MAP['hierdec-mel_4bar'] = Config(
    model=MusicVAE(
        lstm_models.BidirectionalLstmEncoder(),
        lstm_models.HierarchicalLstmDecoder(
            lstm_models.CategoricalLstmDecoder(),
            level_lengths=[8, 8],
            disable_autoregression=True)),
    hparams=merge_hparams(
        lstm_models.get_default_hparams(),
        HParams(
            batch_size=512,
            max_seq_len=4 * 16,
            z_size=256,
            enc_rnn_size=[512],
            dec_rnn_size=[256, 256],
            free_bits=48,
            max_beta=0.2,
        )),
    note_sequence_augmenter=None,
    data_converter=mel_4bar_converter,
    train_examples_path=None,
    eval_examples_path=None,
)
```

[Maestro Data 활용 모델 구성]

❖ MusicVAE 모델을 활용한 피아노샘플 생성기

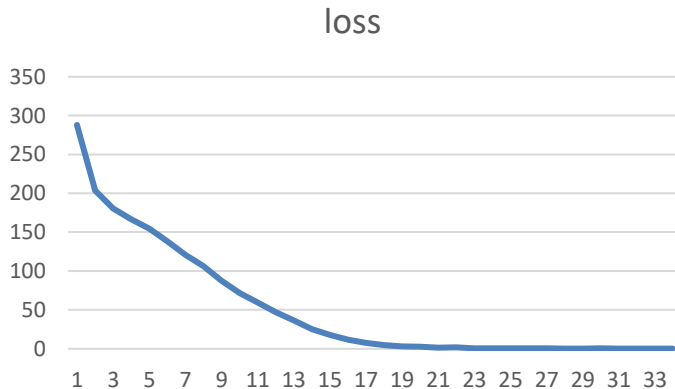
▶ 모델 학습 결과 (Loss)

- Groove 데이터 실험때와 동일한 파라미터임에도 불구하고 Loss가 빠르게 수렴하는 모습을 보임

```

2022-12-09 01:23:34.772502 {'global_step': 1300, 'loss': 25.372885}
2022-12-09 01:24:00.382810 {'global_step': 1400, 'loss': 17.933247}
2022-12-09 01:24:26.091646 {'global_step': 1500, 'loss': 11.584067}
2022-12-09 01:24:56.636984 {'global_step': 1600, 'loss': 7.661615}
2022-12-09 01:25:22.156637 {'global_step': 1700, 'loss': 4.5177584}
2022-12-09 01:25:52.515166 {'global_step': 1800, 'loss': 3.077307}
2022-12-09 01:26:17.909475 {'global_step': 1900, 'loss': 2.3420713}
2022-12-09 01:26:48.057723 {'global_step': 2000, 'loss': 1.1190193}
2022-12-09 01:27:13.771985 {'global_step': 2100, 'loss': 1.6619911}
2022-12-09 01:27:44.286829 {'global_step': 2200, 'loss': 0.57212454}
2022-12-09 01:28:09.735194 {'global_step': 2300, 'loss': 0.593472}
2022-12-09 01:28:40.178178 {'global_step': 2400, 'loss': 0.39969653}
2022-12-09 01:29:05.794799 {'global_step': 2500, 'loss': 0.4452215}
2022-12-09 01:29:35.901383 {'global_step': 2600, 'loss': 0.33789277}
2022-12-09 01:30:01.506219 {'global_step': 2700, 'loss': 0.21799786}
2022-12-09 01:30:27.105100 {'global_step': 2800, 'loss': 0.23725453}
2022-12-09 01:30:57.735892 {'global_step': 2900, 'loss': 0.6474892}
2022-12-09 01:31:23.544674 {'global_step': 3000, 'loss': 0.24804246}
2022-12-09 01:31:53.807016 {'global_step': 3100, 'loss': 0.10720755}
2022-12-09 01:32:19.188726 {'global_step': 3200, 'loss': 0.14033175}
2022-12-09 01:32:49.182710 {'global_step': 3300, 'loss': 0.12702069}
  
```

[학습간 Loss 결과 로그]



[학습간 Loss 결과 그래프]

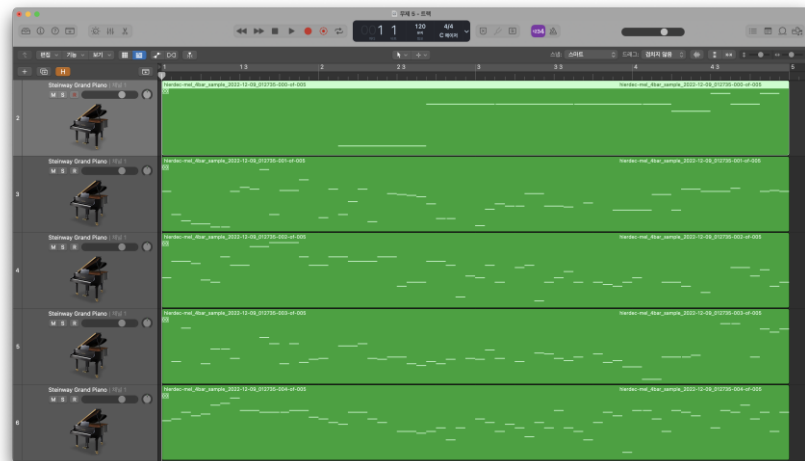
❖ MusicVAE 모델을 활용한 피아노샘플 생성기

▶ 모델 학습 결과 (MIDI 생성 및 재생)

- 정상적으로 미디파일이 생성되었으며, DAW에 로드 및 재생하여 4마디 피아노 샘플이 제대로 생성되는 것을 확인함

🎵 hierdec-mel_4bar_sample_2022-12-09_012735-000-of-005.mid
🎵 hierdec-mel_4bar_sample_2022-12-09_012735-001-of-005.mid
🎵 hierdec-mel_4bar_sample_2022-12-09_012735-002-of-005.mid
🎵 hierdec-mel_4bar_sample_2022-12-09_012735-003-of-005.mid
🎵 hierdec-mel_4bar_sample_2022-12-09_012735-004-of-005.mid

[생성된 MIDI파일 리스트]



[DAW에서의 MIDI파일 로드]

❖ 실험 결과

▶ 결론

- GrooVAE 모델에 Groove dataset을 사용한 결과 MIDI파일이 정상적으로 생성되는 것을 확인함
- MusicVAE 모델에 Groove dataset을 사용한 결과 MIDI파일이 비정상적으로 생성되는 것을 확인함
- MusicVAE 모델을 사용한 실험과 동일한 조건으로 Maestro dataset을 테스트한 결과 MIDI파일의 정상적 생성을 확인함
- 따라서 현재 모델구조에서 하이퍼 파라미터나, 데이터의 양은 모델성능을 크게 저하시키는 원인이 아님을 확인
- MusicVAE 모델과 GrooVAE 모델의 차이점은 모델의 구조 차이도 있지만 Decoder의 Loss function 차이에서 성능이 갈리는 것으로 보임
- Groove 실험과 Maestro 실험에서의 차이점은 DataConverter가 다르며, 해당 알고리즘에 따라서 성능이 갈리는 것으로 보임
- 따라서 MusicVAE 모델을 활용하여 Groove Dataset에서 좋은 성능을 확보하기 위해서는 Decoder의 Loss function의 개선이나 DataConverter 부분을 개선하는 방법이 필요해 보임

❖ 실험 결과에 따른 모델 개선

▶ 모델구성

- 실험 결과에 따라서 빠르게 실험 할 수 있는 DataConverter 부분을 GrooveConverter에서 DrumsConverter로 변경하여 실험 진행
- 이 부분에서 MusicVAE의 모델구조와 파라미터는 동일하게 진행함

```
CONFIG_MAP['musicvae_4bar_paper_reduceunit'] = Config(
    model=MusicVAE(lstm_models.BidirectionalLstmEncoder(),
        lstm_models.HierarchicalLstmDecoder(
            core_decoder = lstm_models.CategoricalLstmDecoder(),
            level_lengths=[8, 8],
            disable_autoregression=True)),
    hparams=merge_hparams(
        lstm_models.get_default_hparams(),
        HParams(
            batch_size=512,
            max_seq_len= 16 * 4, # 4 bars w/ 16 steps per bar ### 256
            z_size=256,
            enc_rnn_size=[512], ### 2048, 2048
            dec_rnn_size=[256, 256], ### 1024, 1024
            max_beta=0.2,
            free_bits=48, ### 256
        )),
    note_sequence_augmenter=None,
    data_converter=data.GrooveConverter(
        split_bars=4, steps_per_quarter=4, quarters_per_bar=4,
        max_tensors_per_notesequence=20,
        pitch_classes=data.ROLAND_DRUM_PITCH_CLASSES,
        inference_pitch_classes=data.REDUCED_DRUM_PITCH_CLASSES),
    tfds_name='groove/groovae_4bar_paper',
)
```

[MusicVAE + GrooveConverter]



```
CONFIG_MAP['musicvae_4bar_paper_drumconv'] = Config(
    model=MusicVAE(lstm_models.BidirectionalLstmEncoder(),
        lstm_models.HierarchicalLstmDecoder(
            core_decoder = lstm_models.CategoricalLstmDecoder(),
            level_lengths=[8, 8],
            disable_autoregression=True)),
    hparams=merge_hparams(
        lstm_models.get_default_hparams(),
        HParams(
            batch_size=512,
            max_seq_len= 16 * 4, # 4 bars w/ 16 steps per bar ### 256
            z_size=256,
            enc_rnn_size=[512], ### 2048, 2048
            dec_rnn_size=[256, 256], ### 1024, 1024
            max_beta=0.2,
            free_bits=48, ### 256
        )),
    note_sequence_augmenter=None,
    data_converter=data.DrumsConverter(
        slice_bars=4, steps_per_quarter=4, quarters_per_bar=4,
        max_tensors_per_notesequence=20,
        pitch_classes=data.ROLAND_DRUM_PITCH_CLASSES),
    tfds_name='groove/musicvae_4bar_paper_drumconv',
)
```

[MusicVAE + DrumsConverter]

❖ 실험 결과에 따른 모델 개선

▶ 모델 학습 결과 (Loss)

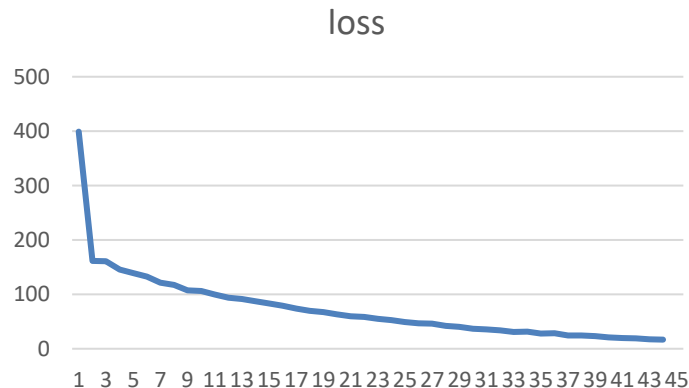
- Loss가 정상적으로 수렴하는 모습을 보이며, GrooVAE 모델보다 Loss가 더 빠르게 수렴하는 모습을 보임

```

2022-12-09 09:53:54.378200 {'global_step': 1600, 'loss': 74.1522}
2022-12-09 09:54:23.911575 {'global_step': 1700, 'loss': 70.050285}
2022-12-09 09:54:58.358131 {'global_step': 1800, 'loss': 67.77874}
2022-12-09 09:55:27.712484 {'global_step': 1900, 'loss': 63.473057}
2022-12-09 09:56:02.062357 {'global_step': 2000, 'loss': 59.81144}
2022-12-09 09:56:36.239632 {'global_step': 2100, 'loss': 58.829933}
2022-12-09 09:57:05.648411 {'global_step': 2200, 'loss': 55.182186}
2022-12-09 09:57:39.905466 {'global_step': 2300, 'loss': 52.64509}
2022-12-09 09:58:09.471392 {'global_step': 2400, 'loss': 49.555206}
2022-12-09 09:58:44.168635 {'global_step': 2500, 'loss': 46.774426}
2022-12-09 09:59:13.720721 {'global_step': 2600, 'loss': 46.198853}
2022-12-09 09:59:48.172193 {'global_step': 2700, 'loss': 42.103794}
2022-12-09 10:00:17.490336 {'global_step': 2800, 'loss': 40.600525}
2022-12-09 10:00:51.764039 {'global_step': 2900, 'loss': 37.019104}
2022-12-09 10:01:21.340044 {'global_step': 3000, 'loss': 35.98981}
2022-12-09 10:01:55.741228 {'global_step': 3100, 'loss': 33.767326}
2022-12-09 10:02:25.086914 {'global_step': 3200, 'loss': 31.317118}
2022-12-09 10:02:59.326642 {'global_step': 3300, 'loss': 31.525696}
2022-12-09 10:03:29.045254 {'global_step': 3400, 'loss': 28.058249}
2022-12-09 10:04:03.211414 {'global_step': 3500, 'loss': 28.542894}
2022-12-09 10:04:37.795651 {'global_step': 3600, 'loss': 24.475431}
2022-12-09 10:05:07.244037 {'global_step': 3700, 'loss': 24.38422}

```

[학습간 Loss 결과 로그]



[학습간 Loss 결과 그래프]

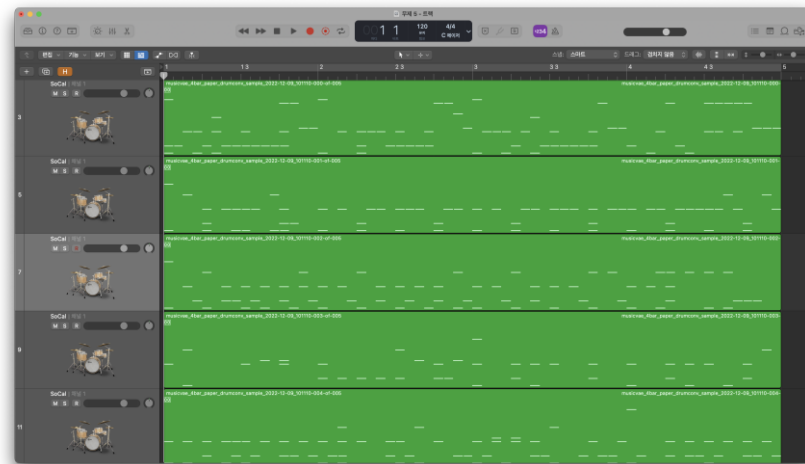
❖ 실험 결과에 따른 모델 개선

▶ 모델 학습 결과 (MIDI 생성 및 재생)

- 정상적으로 midi파일이 생성되었으며, DAW에 로드 및 재생하여 4마디 드럼샘플이 제대로 생성되는 것을 확인함
- 재생시 음악적 느낌이 이전 GrooVAE 보다 향상되어 보임

🎵 musicvae_4bar_paper_drumconv_sample_2022-12-09_101110-000-of-005....
 🎵 musicvae_4bar_paper_drumconv_sample_2022-12-09_101110-001-of-005....
 🎵 musicvae_4bar_paper_drumconv_sample_2022-12-09_101110-002-of-005....
 🎵 musicvae_4bar_paper_drumconv_sample_2022-12-09_101110-003-of-005....
 🎵 musicvae_4bar_paper_drumconv_sample_2022-12-09_101110-004-of-005....

[생성된 MIDI파일 리스트]



[DAW에서의 MIDI파일 로드]

❖ 추가 실험 결과

▶ 결론

- 기존에 실패하였던 MusicVAE 실험에서 Data Converter를 변경하여 실험
- MusicVAE + DrumsConverter 조합으로 성능 확보에 성공하였으며, 기존보다 더 좋은 Loss결과와 음악성을 보임
- GrooVAE + GrooveConverter 조합에서도 괜찮은 성능을 보였으므로 GrooveConverter 자체는 유의미한것으로 보임
- DrumsConverter와 GrooveConverter의 차이점에 대한 확인이 필요함
- MusicVAE + GrooveConverter 조합으로 성능을 내기 위해서는 Decoder의 Loss function의 개선을 통하여 좋은성능을 이끌어낼수 있을것으로 보임

감 사 합 니 다
