위 내용은 [MusicVAE 논문](https://arxiv.org/pdf/1803.05428.pdf) 및 [yop5292 (김우빈)](https://velog.io/@yop5292) , [Seyong Kim – Medium](https://medium.com/@seyong.dev) 에 내용을 참고하여 작성 되었습니다.

## **Variational AutoEncoder(변분 추론 인코더, 이하 VAE)**

# 입력데이터 X를 잘 설명하는 특징(feature)을 추출하여 Latent vector(잠재백터) Z에 담은후에, 이 Latent vector(잠재 벡터) Z를 통하여, X와 유사하나 완전 새로운 데이터를 생성하는것을 목표로한다.

# 이때 각 특징(feature)이 가우시안 분포를 따른다고 가정하고, Latent vector(잠재 벡터)Z는 각 특징(feature)의 평균과 분산값을 나타낸다.

# VAE를 이용해 음악을 작곡한다고 가정한다면, 악기의 속도, 피치의 평균 과 분산등의 정보등을 (feature)를 Latent vector(잠재 벡터) Z에 담고, 그 Latent vector(잠재 벡터) Z를 이용해 음악을 작곡 해 나가는 것 이다.

### **Latent vector(잠재 벡터)**

# Latent vector(잠재 벡터)는 독립적인 잠재 변수들의 쌍을 Latent vector(잠재 벡터)라고 부르며, GAN(적대적 생성신경망)등의 컴퓨터비전 분야에서도 사용한다.

# 입력 음원(midi데이터) X를 VAE에 통과시켜, Latent vector(잠재 벡터) Z를 다시 Decoder에 통과시켜 입력 음원(midi데이터) X와 비슷하지만 새로운 데이터를 찾아내는 구조다.

# VAE는 입력 음원(midi데이터) X가 들어오면 그 데이터의 다양한 특징을 찾아 각각의 확률 변수가 되는 확률 분포를 만들게 된다. 이런 확률 분포를 잘 찾아, 값이 높은 부분을 이용해 실제 있을법한 음원(midi데이터)를 만들어 낸다.

# 

# **Recurrent VAEs**

MusicVAE에서는 순환 신경망(RNN) 모델을 적용한다. 구체적으로 인코더는 입력 시퀀스를 처리하여 은닉 상태(hidden state)로 생성한다. 잠재 코드 z의 분포 파라미터는 h\_T에 관한 함수로 표현된다. 디코더는 샘플링된 잠재 벡터 z를 RNN 디코더의 초기 상태값으로 설정하고 출력 시퀀스를 생성한다. MusicVAE 모델은 입력 시퀀스를 재구성하고 사후확률(posterior) q\_lambda(z|x)가 사전확률(prior) p(z)에 근사하도록 훈련된다.

이 방식에는 두가지 문제점이 있다. 첫째로 RNN은 강력한 자기회귀 모델이기 때문에 RNN 디코더는 데이터를 학습하기 위해 잠재 코드를 무시할 수 있다. 둘째로 모델은 전체 시퀀스를 단일 잠재 벡터로 압축해야 한다. 이 방식은 짧은 시퀀스에서는 가능하지만 시퀀스 길이가 길어질 경우 실패한다. 이 문제들을 해결하기 위해 **MusicVAE에서는 계층적(hierarchical) RNN을 디코더에 사용**한다.

# **양방향 순환 인코더(Bidirectional Encoder)**

인코더 q\_lambda(z|x)는 2개 층의 양방향 LSTM 층을 사용한다. 2번째 양방향 LSTM 층에서 최종 상태 벡터 h\_T\_fw, h\_T\_bw를 얻는다. 이 두 벡터를 연결(concatenation)하여 h\_T 벡터를 만들어 두 개의 완전 연결 신경망(fully connected layers)를 통과시켜 잠재 분포 파라미터인 mu와 sigma를 생성한다.



W\_{h\mu}, W\_{h\sigma)와 b\_mu, b\_sigma는 각각 가중치 행렬과 편향 벡터다. 이 모델에서는 512개의 층과 256의 잠재 차원을 사용하는데 표준 VAE에서처럼 mu와 sigma를 통해서 잠재 분포를 표현할 수 있다. 양방향 순환 인코더는 긴 입력 시퀀스를 잠재 분포의 파라미터로 이상적으로 표현할 수 있다.

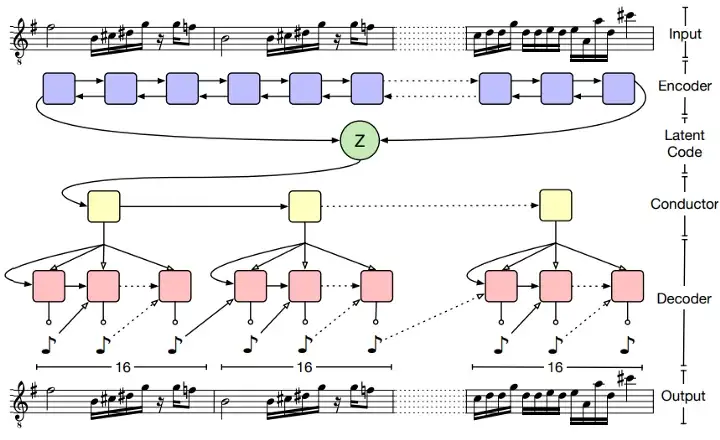
# **계층적 디코더(Hierarchical Decoder)**

예비 실험에서는 단순한 적층(stacked) RNN을 사용하였는데, 긴 시퀀스에 대하여 형편없는 샘플링과 재구성 성능을 확인했다. 이 문제는 출력 시퀀스가 생성됨에 따라 잠재 상태값의 영향력이 줄어들기 때문으로 추측되는데 이를 해결하기 위해 계층적 RNN을 디코더에 적용한다.

전체 입력 시퀀스 X를 중복되지 않도록 U개의 하위시퀀스 y\_u로 나눈다.



위의 식에서 i\_{U+1} = T 로 정의된다. Conductor RNN은 U개의 임베딩 벡터 c={c\_1, c\_2, …, c\_U}를 생성한다. Conductor에는 1024 크기의 은닉 상태와 512의 출력 차원을 가진 2개 층의 단방향(unidirectional) LSTM이 사용된다.



Conductor가 임베딩 벡터 c를 생성하면, 각 벡터는 개별적으로 tanh 활성함수가 적용된 완전 결합 층을 지나면서 디코더의 최종 바닥층의 초기상태를 생성한다. 그리고 RNN 디코더는 softmax 출력층을 통해서 자기회귀적으로 각 하위시퀀스 y\_u에 해당하는 분포를 생성한다. 디코더 바닥층의 각 스텝에서, 현시점의 Conductor 임베딩 c\_u는 이전시점의 출력 토큰과 연결(concatenation)되어 입력으로 사용된다.

계층적 RNN 디코더의 구조를 통해서, 각 하위시퀀스 y\_u가 연결된 각 Conductor로부터 생성된 임베딩 c\_u을 통해서만 영향을 받도록 한다. 이는 기존 단순 RNN 디코더의 문제점을 해소하여 잠재 코드 벡터에 근거하여 모델이 효과적으로 학습하도록 강제한다. 각 하위시퀀스의 끝에서 디코더의 상태값이 Conductor로 다시 들어가는 자기회귀적인 버전의 Conductor 실험에서는 성능이 좋지 않았다고 한다.

**Preprocessing**

입력 데이터로 쓰이는 MIDI 데이터는 음악이 어떻게 연주되어야 하는지나타내는 악보와 유사한 개념으로, 어느 시점에 어떤 악기를 연주하여야 하는지에 대한 정보가 담겨있다.

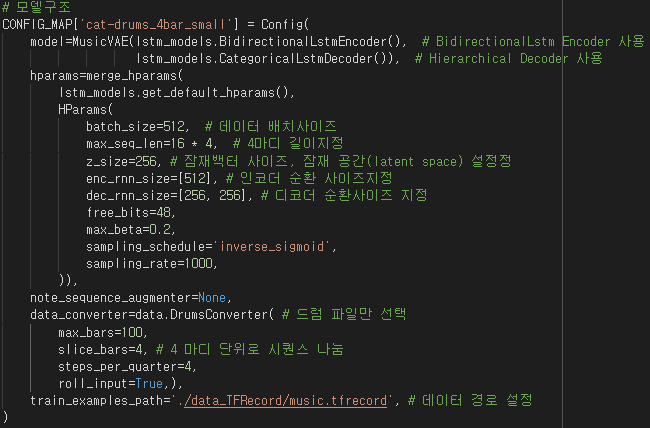
Magenta의 note-seq 패키지를 이용하여 MIDI (.mid, .midi) MusicXML (.xml, .mxl) 파일을 TFRecord 포맷의 NoteSequences으로 변환할 수 있다.

TFRecord 포맷은 바이너리 레코드를 저장하는 단순한 포맷이다. TFRecord 포맷은 더 적은 공간으로 동일한 데이터를 저장할 수 있고, TPU와 같은 병렬 I/O 연산을 사용할 수 있다.

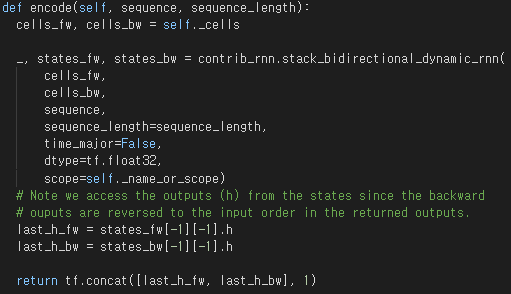
**Training**

TFRecord 데이터를 입력 시퀀스로 하여 VAE 모델을 학습시킬 수 있다. MusicVAE의 configs.py에는 다양한 설정이 있고, 직접 모델을 훈련시키거나 Magenta에서 제공하는 각 설정별 기학습(pre-trained) 체크포인트를 이용할 수 있다.

이번에는 Groove MIDI Dataset으로 MusicVAE 모델을 학습시키고 4마디에 해당하는 드럼 샘플을 생성하는 과정을 진행하였다. 4마디 드럼 샘플을 생성하기 위해서는 configs.py의 ‘cat-drums\_2bar\_small’ 의 하이퍼 파라미터를 기반으로 변형 사용했다.

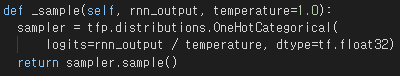


* 인코더로는 BidirectionalLstmEncoder 클래스를 사용하고 디코더로는 CategoricalLstmDecoder 클래스를 사용한다. 인코더의 경우는 512개의 은닉 노드를 가진 양방향 LSTM 레이어 1개가 사용되었고, 디코더의 경우에는 2048개의 은닉 노드를 가진 Categorical LSTM 레이어가가 사용되었다.
* 16bar 정도의 길이가 긴 시퀀스의 경우에는 Posterior Collapse(RNN 모델이 잠재 벡터를 무시하고 학습에 집중하는 현상)를 해결하기 위해 hierarchical decoder를 적용하지만, 해당 과제에서는 4bar 의 비교적 짧은 시퀀스를 다루기 때문에 conductor RNN 없이 MusicVAE 모델을 구성한다.
* max\_seq\_len = 16 \* 4 는 1bar 당 16 steps 기준으로 4bar 길이를 의미한다.
* z\_size=256 는 잠재 벡터(latent vector) z의 크기가 256임을 뜻한다.
* slice\_bars=4 로 정하여 4마디 단위로 시퀀스를 나눈다.



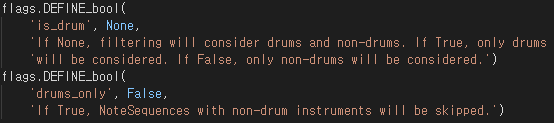
lstm\_models.py 의 BidirectionalLstmEncoder 클래스의 encode 함수다.

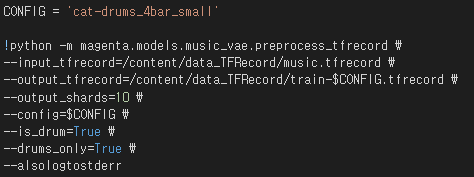
마지막에 forward 은닉 상태값과 backward 은닉 상태값을 tf.concat 메소드로 결합시킨다.



CategoricalLstmDecoder 클래스의 \_sample 함수다.

tfp.distributions.OneHotCategorical 메소드는 categorical distribution을 사용하여 샘플링을 수행한다. categorical distribution은 1부터 k까지의 정수 중 하나가 나올 확률 변수의 분포다. 이 방식은 bernoulli distribution의 일반화된 버전이다.



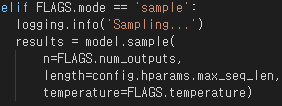


preprocess\_tfrecord.py 의 설정 중 드럼 시퀀스만 뽑아내기 위한 설정이다.

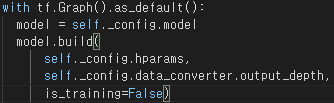
preprocess 과정에서 drum의 notesequences만 남도록 ‘is\_drum’과 ‘drums\_only’ flags들을 True로 설정한다.(적용 실패)

# **Generation**

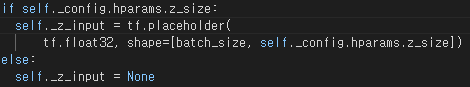
훈련된 체크포인트를 이용해서 새로운 MIDI 데이터를 생성할 수 있다. 생성 방식에는 ‘Sample’ 방식과 ‘Interpolate’ 방식이 있다. 이번에는 ‘Sample’ 방식으로 진행하였다.



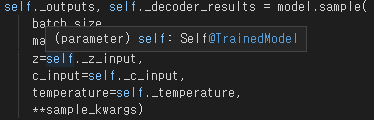
trained\_model.py의 TrainedModel 클래스에서 tf.Graph()를 통해 그래프를 생성한다.



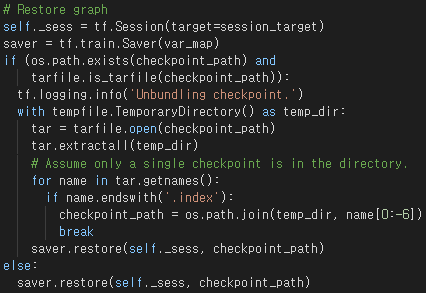
batch\_size와 z\_size 크기의 플레이스홀더를 정의한다.



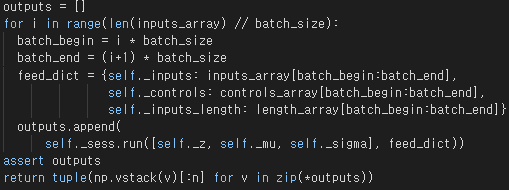
디코더 모델의 sample 함수에 batch\_size, \_max\_length, \_z\_input, \_c\_input, \_temperature 등의 하이퍼파라미터를 입력값으로 넣어 \_outputs를 출력한다. \_outputs는 이후의 sess.run() 함수에서 플레이스홀더 역할을 하게 된다.



아래의 코드로 체크포인트의 모델 가중치를 로드한다.



TrainedModel 클래스의 sample 함수에서 np.random.randn()으로 표준정규분포에서 샘플링한 난수를 잠재 코드(latent code)로 삼고 feed\_dict에 넣는다. same\_z=False이고, 샘플별 잠재 코드를 서로 다르게 샘플링한다. \_sess.run()을 통해서 연산 그래프를 실행하여 tensor 형식의 결과를 리턴받아서 outputs에 추가한다. 그 이후에는 data\_converter.from\_tensors() 함수를 통해서 tensor 형식의 샘플들을 NoteSequence 형식으로 변환한다.



위 코드에서 \_c\_input은 시퀀스를 제어(control)하는 입력값이다. 아래 코드에서 \_c\_input은 controls 변수의 위치에 해당하고, 멜로디, 베이스, 드럼 등 여러가지 악기로 연주된 데이터에서 멜로디를 제어하는 역할을 한다.

NoteSequence 형식으로 반환된 샘플들은 아래 코드를 통해서 MIDI 파일로 변환된다.

