[AATG015-01] Assignment 4

20200118 조창희

1. MPLEMENT RNN model

   a. 코드 설명

RNN만을 사용하여 transcriber을 구현하는 모델이다.

```
[ ] class Transcriber_RNN(nn.Module):
        def __init__(self, cnn_unit, fc_unit):
            '''
            TODO: Complete the initialization of the model.

            Args:
              cnn_unit: unit for number of channels in CNN Stack (You don't have to use this in Transcriber_RNN)
              fc_unit: unit ofr number of hidden units in LSTM layer
            '''
            super().__init__()
            # Notice: Changing the initialization order may fail the tests.
            self.melspectrogram = LogMelSpectrogram()
            self.frame_lstm = nn.LSTM(input_size = N_MELS, hidden_size = fc_unit, num_layers = 2, batch_first = True, bidirectional = True)
            self.frame_fc = nn.Linear(in_features=fc_unit * 2, out_features=88)

            self.onset_lstm = nn.LSTM(input_size = N_MELS, hidden_size = fc_unit, num_layers = 2, batch_first = True, bidirectional = True)
            self.onset_fc = nn.Linear(in_features=fc_unit * 2, out_features=88)

        def forward(self, audio):
            # TODO: Question 1
            # print(self.melspectrogram(audio).shape[-1])
            frame_out, _ = self.frame_lstm(self.melspectrogram(audio))
            frame_out = self.frame_fc(frame_out)
            onset_out, _ = self.onset_lstm(self.melspectrogram(audio))
            onset_out = self.onset_fc(onset_out)
            return frame_out, onset_out


    model = Transcriber_RNN(cnn_unit=64, fc_unit=256)
```

먼저, LogMelSpectrogram 함수 호출을 통해 forward에서 인자로 주어지는 audio를 melspectrogram으로 변환한다. 그러한 다음 양방향 LSTM을 거친다. 양방향이기 때문에 결과값은 fc_unit * 2가 나온다. 이것을 그대로 입력으로 받아, 최종적인 예측을 위해 88(건반 개수) 크기로 변환해준다. 이러한 과정을 frame과 onset에 대하여 각각 수행한다.

   b. 결과 분석

먼저 100 iteration으로 훈련시켰을 때의 결과이다.

```
metric/loss/frame_loss : 0.1362825632095337
metric/loss/onset_loss : 0.013355893082916737
metric/frame/frame_f1 : 0.0
metric/frame/onset_f1 : 0.0
metric/note/f1 : 0.0
metric/note-with-offsets/f1 : 0.0
                         loss frame_loss               : 0.136 +- 0.049
                         loss onset_loss               : 0.013 +- 0.007
                        frame frame_precision          : 0.000 +- 0.000
                        frame frame_recall             : 0.000 +- 0.000
                        frame frame_f1                 : 0.000 +- 0.000
                        frame onset_precision          : 0.000 +- 0.000
                        frame onset_recall             : 0.000 +- 0.000
                        frame onset_f1                 : 0.000 +- 0.000
                         note precision                : 0.000 +- 0.000
                         note recall                   : 0.000 +- 0.000
                         note f1                       : 0.000 +- 0.000
                         note overlap                  : 0.000 +- 0.000
               note-with-offsets precision             : 0.000 +- 0.000
               note-with-offsets recall                : 0.000 +- 0.000
               note-with-offsets f1                    : 0.000 +- 0.000
               note-with-offsets overlap               : 0.000 +- 0.000
```

Frame_loss가 onset_loss에 비해 훨씬 큰 값을 보인다. (0.1) 또한, Precision과 Recall의 조화평균인 f1 score은 둘다 0으로, 상당히 낮은 성능을 보이고 있다.

다른 조건은 동일하며, 5000번으로 훈련시켰을 때는 다음과 같다.

```
metric/loss/frame_loss : 0.04459338262677193
metric/loss/onset_loss : 0.004179627634584904
metric/frame/frame_f1 : 0.6003304048014455
metric/frame/onset_f1 : 0.6243114976133135
metric/note/f1 : 0.7469723455038787
metric/note-with-offsets/f1 : 0.2672840924123455
                           loss frame_loss               : 0.045 +- 0.023
                           loss onset_loss               : 0.004 +- 0.003
                          frame frame_precision          : 0.753 +- 0.242
                          frame frame_recall             : 0.544 +- 0.225
                          frame frame_f1                 : 0.600 +- 0.205
                          frame onset_precision          : 0.740 +- 0.127
                          frame onset_recall             : 0.591 +- 0.214
                          frame onset_f1                 : 0.624 +- 0.148
                           note precision                : 0.980 +- 0.032
                           note recall                   : 0.636 +- 0.218
                           note f1                       : 0.747 +- 0.176
                           note overlap                  : 0.424 +- 0.172
                 note-with-offsets precision             : 0.333 +- 0.163
                 note-with-offsets recall                : 0.233 +- 0.149
                 note-with-offsets f1                    : 0.267 +- 0.158
                 note-with-offsets overlap               : 0.710 +- 0.280
```

iteration을 늘렸을 때 loss가 크게 감소하였음을 확인할 수 있다. F1 스코어가 이전에 비해 꽤 증가하였다. note에 대한 f1 스코어는 0.7인 것에 비해, note with offsets의 경우 0.2 정도로 낮은 결과를 보였다.

2. IMPLEMENT CRNN MODEL

a. 구현 코드

CNN과 RNN을 결합하여 transcriber을 구현한 모델이다. 구현 코드는 다음과 같다.

```python
class Transcriber_CRNN(nn.Module):
    def __init__(self, cnn_unit, fc_unit):
        '''
        TODO: Complete the initialization of the model.

        Args:
          cnn_unit: unit for number of channels in CNN Stack
          fc_unit: unit for number of hidden units in FC layer of CNN stack and LSTM layer
        '''
        super().__init__()
        # Notice: Changing the initialization order may fail the tests.
        self.melspectrogram = LogMelSpectrogram()

        self.frame_conv_stack = ConvStack(N_MELS, cnn_unit, fc_unit)
        self.frame_lstm = nn.LSTM(input_size = fc_unit, hidden_size = fc_unit, num_layers = 1, batch_first = True, bidirectional = True)
        self.frame_fc = nn.Linear(in_features=fc_unit * 2, out_features=88)

        self.onset_conv_stack = ConvStack(N_MELS, cnn_unit, fc_unit)
        self.onset_lstm = nn.LSTM(input_size = fc_unit, hidden_size = fc_unit, num_layers = 1, batch_first = True, bidirectional = True)
        self.onset_fc = nn.Linear(in_features=fc_unit * 2, out_features=88)
```

Init에서는 기존 RNN model에서 conv_stack이 추가되었다. ConvStack 레이어 구현 코드를 보면, 마지막에 nn.Linear을 사용하여 계산 결과를 fc_unit의 크기로 변화시킨다. 따라서 ConvStack을 호출한 다음 nn.LSTM은 n_mels가 아닌, fc_unit을 입력 차원으로 받는다. 그 외에는 1번과 동일하다.

```python
def forward(self, audio):
    '''
    TODO: Complete the forward pass of the model,

    Args:
      audio: torch.Tensor of shape (batch_size, sequence_length)

    Returns:
      frame_out: torch.Tensor of shape (batch_size, sequence_length, 88)
      onset_out: torch.Tensor of shape (batch_size, sequence_length, 88)
    '''
    mel = self.melspectrogram(audio)
    x = self.frame_conv_stack(mel)  # (B, T, C)
    frame_out, _ = self.frame_lstm(x)
    frame_out = self.frame_fc(frame_out)

    x = self.onset_conv_stack(mel)  # (B, T, C)
    onset_out, _ = self.onset_lstm(x)
    onset_out = self.onset_fc(onset_out)

    return frame_out, onset_out
```

forward에서 먼저 오디오를 mel 로 변환해주고, conv_stack에 스펙트로그램을 넣어준 다음, 계산된 결과에 대해 blstm을 수행 후 fc를 수행한다. 1번과 마찬가지로 frame과 onset 각각에 대해 위 과정을 적용하였다.

b. 결과 분석

100 iterations일 때 결과는 아래와 같다.

```
metric/loss/frame_loss : 0.13778850436210632
metric/loss/onset_loss : 0.013313904404640198
metric/frame/frame_f1 : 0.0
metric/frame/onset_f1 : 0.0
metric/note/f1 : 0.0
metric/note-with-offsets/f1 : 0.0
                          loss frame_loss               : 0.138 +- 0.044
                          loss onset_loss               : 0.013 +- 0.007
                         frame frame_precision          : 0.000 +- 0.000
                         frame frame_recall             : 0.000 +- 0.000
                         frame frame_f1                 : 0.000 +- 0.000
                         frame onset_precision          : 0.000 +- 0.000
                         frame onset_recall             : 0.000 +- 0.000
                         frame onset_f1                 : 0.000 +- 0.000
                          note precision                : 0.000 +- 0.000
                          note recall                   : 0.000 +- 0.000
                          note f1                       : 0.000 +- 0.000
                          note overlap                  : 0.000 +- 0.000
               note-with-offsets precision              : 0.000 +- 0.000
               note-with-offsets recall                 : 0.000 +- 0.000
               note-with-offsets f1                     : 0.000 +- 0.000
               note-with-offsets overlap                : 0.000 +- 0.000
```

RNN 100번 돌렸을 때와 유사한 loss값이며 마찬가지로 낮은 성능을 보인다.

5000 iteration일 때의 결과는 다음과 같다.

```
metric/loss/frame_loss : 0.02627275511622429
metric/loss/onset_loss : 0.002397937700152397
metric/frame/frame_f1 : 0.7849880082225225
metric/frame/onset_f1 : 0.7773933935638053
metric/note/f1 : 0.927747834829036
metric/note-with-offsets/f1 : 0.5710584050865846
                           loss frame_loss               : 0.026 +- 0.016
                           loss onset_loss               : 0.002 +- 0.002
                          frame frame_precision          : 0.856 +- 0.195
                          frame frame_recall             : 0.727 +- 0.149
                          frame frame_f1                 : 0.785 +- 0.168
                          frame onset_precision          : 0.792 +- 0.118
                          frame onset_recall             : 0.769 +- 0.094
                          frame onset_f1                 : 0.777 +- 0.098
                           note precision                : 1.000 +- 0.000
                           note recall                   : 0.870 +- 0.093
                           note f1                       : 0.928 +- 0.054
                           note overlap                  : 0.624 +- 0.125
                note-with-offsets precision              : 0.614 +- 0.142
                note-with-offsets recall                 : 0.537 +- 0.143
                note-with-offsets f1                     : 0.571 +- 0.141
                note-with-offsets overlap                : 0.850 +- 0.124
```

loss값은 RNN 5000번 돌렸을 때보다 두배정도 더 큰 값을 보인다. 다만 f1 스코어가 각각 0.2 정도 더 높은 수치를 보였다. note의 f1 스코어는 0.92로 상당히 높은 점수를 보인다.

3. Implement Onsets and Frames model

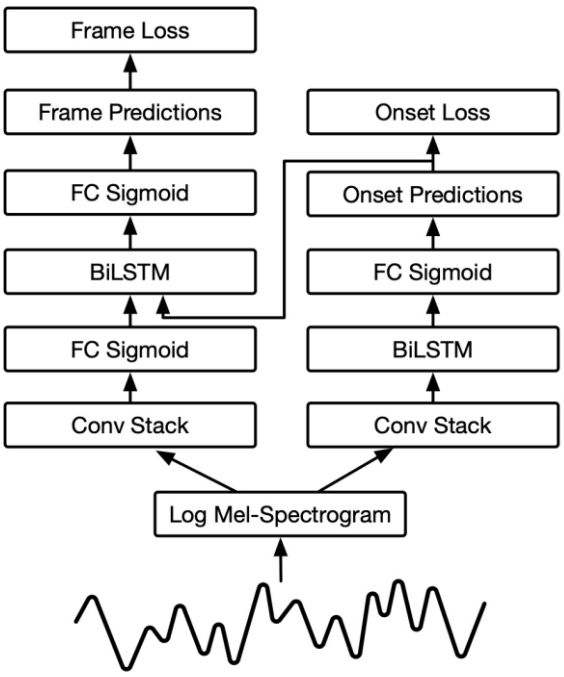


Figure 1. Diagram of Network Architecture

1, 2번에서는 frame과 onset에 대해 각각 신경망을 적용하기 때문에 두 정보가 교류되지 않았고, 두 모델이 독립적으로 학습되었다. 하지만 3번 모델에서는 onset 결과가 나온 후 frame의 BLSTM에 입력값으로 함께 들어가게 된다는 차이점이 있다.

a. 구현 코드

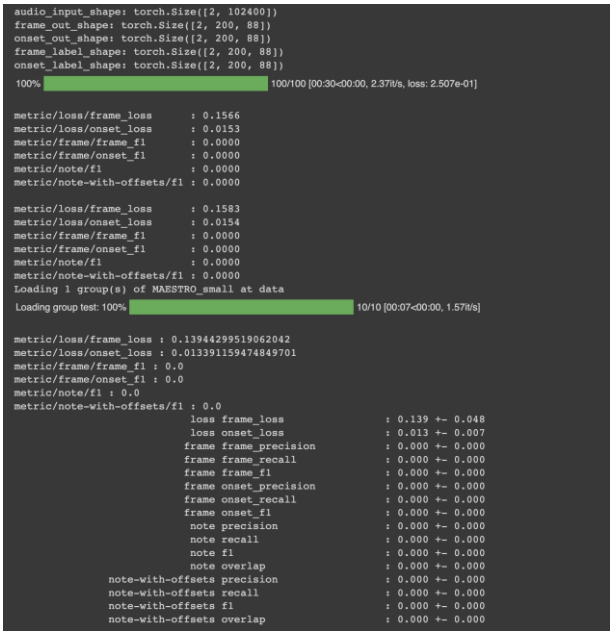
```
class Transcriber_ONF(nn.Module):
    def __init__(self, cnn_unit, fc_unit):
        super().__init__()
        '''
        TODO: Complete the initialization of the model.

        Args:
          cnn_unit: unit for number of channels in CNN Stack
          fc_unit: unit for number of hidden units in FC layer of CNN stack and LSTM layer
        '''

        # Notice: Changing the initialization order may fail the tests.
        self.melspectrogram = LogMelSpectrogram()

        self.frame_conv_stack = ConvStack(N_MELS, cnn_unit, fc_unit)
        self.frame_fc = nn.Linear(fc_unit, 88)

        self.onset_conv_stack = ConvStack(N_MELS, cnn_unit, fc_unit)
        self.onset_lstm = nn.LSTM(input_size = fc_unit, hidden_size = fc_unit, num_layers = 1, batch_first = True, bidirec
        self.onset_fc = nn.Linear(in_features=fc_unit * 2, out_features=88)
        self.combined_lstm = nn.LSTM(input_size = 88 * 2, hidden_size = fc_unit, num_layers = 1, batch_first = True, bidir
        self.combined_fc = nn.Linear(in_features=fc_unit * 2,  out_features=88)
```

Init에서는 기본적으로는 1, 2번과 거의 유사하지만, frame_lstm 대신 combined_lstm이 존재한다는 차이점이 있다. combined_lstm 에서는 88건반으로 크기가 조정된 두 값 (frame, onset)이 들어오기 때문에 88 * 2로 설정하였다.

```python
def forward(self, audio):
    '''
    # TODO: implement this function based on the given diagram

    Args:
      audio: torch.Tensor of shape (batch_size, sequence_length)

    Returns:
      frame_out: torch.Tensor of shape (batch_size, sequence_length, 88)
      onset_out: torch.Tensor of shape (batch_size, sequence_length, 88)

    CAUTION: You should not use Sigmoid or Softmax.
    '''
    mel = self.melspectrogram(audio)
    frame_out = self.frame_conv_stack(mel)
    frame_out = self.frame_fc(frame_out)

    onset_out = self.onset_conv_stack(mel)
    onset_out, _ = self.onset_lstm(onset_out)
    onset_out = self.onset_fc(onset_out)
    combined_input = torch.cat((frame_out, onset_out), dim=-1)
    combined_out, _ = self.combined_lstm(combined_input)
    combined_out = self.combined_fc(combined_out)

    frame_out = combined_out
    return frame_out, onset_out
```

forward는 다이어그램과 같은 형식으로 진행되도록 하였다. Combined_input에서 torch.cat을 이용하여 두 입력값을 합쳐주었고, 이것을 그대로 combined_lstm 에 넣어주었다.

다음은 100번 돌렸을 때의 결과값이다.



2번 결과와 거의 유사하며, 마찬가지로 성능이 그리 높지 않다.

다음은 5000번 돌렸을 때의 결과이다.

```
metric/loss/frame_loss : 0.043266661465168
metric/loss/onset_loss : 0.0029081508982926607
metric/frame/frame_f1 : 0.7254237693240256
metric/frame/onset_f1 : 0.7687686390963082
metric/note/f1 : 0.8650366263656133
metric/note-with-offsets/f1 : 0.51251872206589
                          loss frame_loss               : 0.043 +- 0.032
                          loss onset_loss               : 0.003 +- 0.002
                         frame frame_precision          : 0.765 +- 0.196
                         frame frame_recall             : 0.706 +- 0.226
                         frame frame_f1                 : 0.725 +- 0.197
                         frame onset_precision          : 0.825 +- 0.107
                         frame onset_recall             : 0.729 +- 0.129
                         frame onset_f1                 : 0.769 +- 0.105
                          note precision                : 0.977 +- 0.038
                          note recall                   : 0.786 +- 0.129
                          note f1                       : 0.865 +- 0.083
                          note overlap                  : 0.601 +- 0.132
              note-with-offsets precision               : 0.572 +- 0.156
              note-with-offsets recall                  : 0.470 +- 0.174
              note-with-offsets f1                      : 0.513 +- 0.166
              note-with-offsets overlap                 : 0.801 +- 0.158
```

두번째 모델과 유사하나, 오히려 loss값이 조금 더 높았고, f1스코어가 조금 떨어지는 모습을 보였다. 전반적으로는 준수한 성능을 보인다.

4. 결과 분석

두 가지 솔로 피아노 음악을 기준으로 테스트하였다. 첫 번째는 팀프로젝트에서 진행했던 학우분의 연주로 테스트하였고, 두번째는 조성진이 연주한 드뷔시의 달빛을 테스트하였다. 곡 선정 목적은 다음과 같다. 첫 번째 곡은 실제 우리 프로젝트의 결과물인 만큼, transcription이 잘 이루어지면 이후 과제 변형에 있어 도움이 될 수 있을 것 같다는 생각이었다. (입력값을 midi 파일로 안받고 소리 자체로 받는 등) 두 번째 곡은 드뷔시의 달빛이 워낙 섬세하고 소리가 작은 곡이다보니, 이러한 유형의 곡도 잘 전사가 이루어질지 궁금하였다. 첫 번째 곡은 원래 길이(25초)이며, 두번째 곡은 앞 1분을 잘라서 테스트하였다.

두 곡 모두 실제로 결과를 들어보았을 때, 음의 전개, 속도 면에서는 원곡과 거의 유사한 흐름을 따른다. 그러나 두 곡 모두 전반적인 음 옥타브가 낮다. 특히 첫 번째 곡의 경우 음의 옥타브가 너무 낮은데, 소리에 노이즈가 섞여 있기 때문이라고 추측한다. 또한 드뷔시 곡의 경우 초반에 등장하는 페달이 감지되지 않고, 스타카토 형태로 처리가 된다.