About our Model

(feat. Listen, Attend and Spell)

Winter Vacation Capstone Study

TEAM Kai.Lib

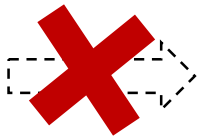
발표자 : 김수환

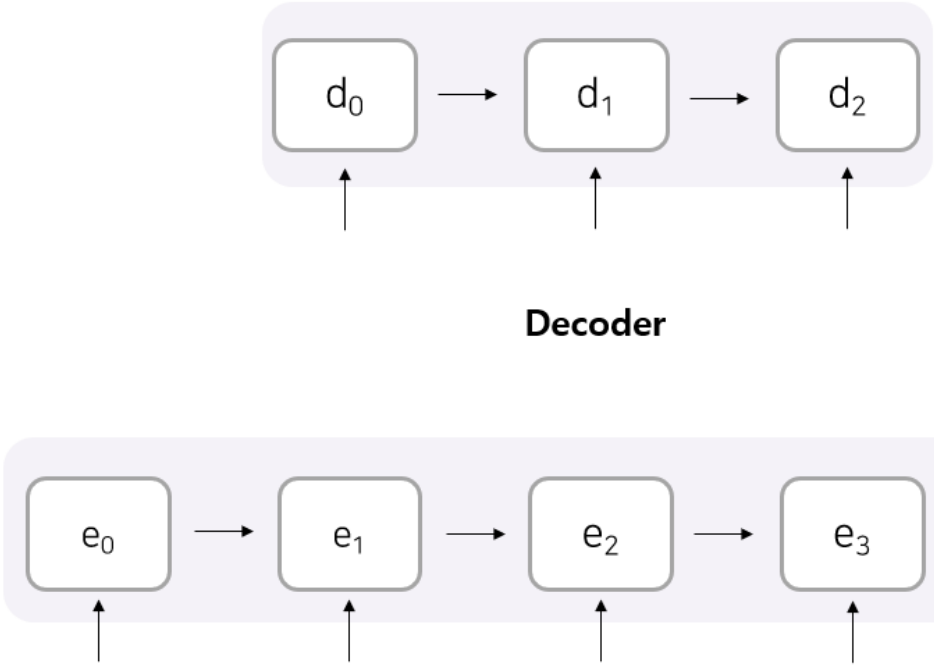
2020.02.03 (MON)

내 머리속 Debug

- 잘못 이해하고 있던 점

```python
def __init__(self):
    self.use_bidirectional = True
    self.use_attention = True
    self.input_reverse = True
    self.use_augmentation = True
    self.use_pickle = True
    self.augment_ratio = 0.3
    self.hidden_size = 256
    self.dropout = 0.5
    self.encoder_layer_size = 5
    self.decoder_layer_size = 3
    self.batch_size = 6
    self.worker_num = 1
    self.max_epochs = 40
    self.lr = 0.0001
    self.teacher_forcing = 0.99
    self.seed = 1
    self.max_len = 80
    self.no_cuda = False
    self.save_name = 'model'
    self.mode = 'train'
    self.load_model = False
    self.model_path = './weight_file/epoch2'
```

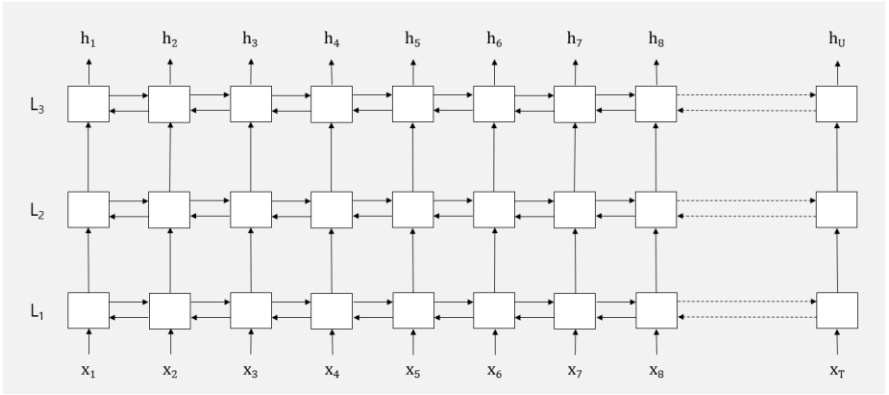




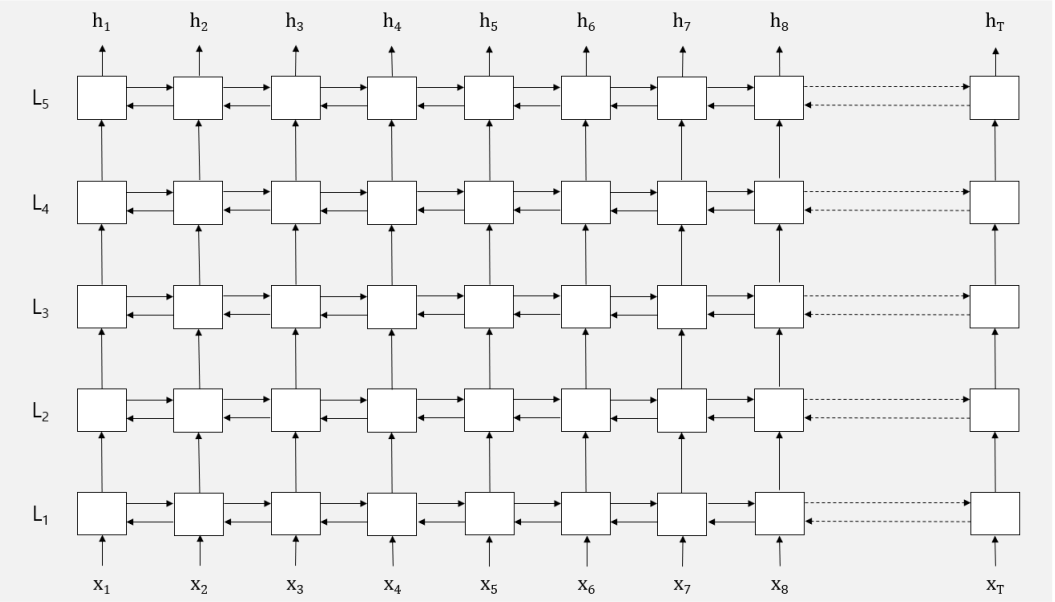
내 머리속 Debug

- 잘못 이해하고 있던 점

```
def __init__(self):
    self.use_bidirectional = True
    self.use_attention = True
    self.input_reverse = True
    self.use_augmentation = True
    self.use_pickle = True
    self.augment_ratio = 0.3
    self.hidden_size = 256
    self.dropout = 0.5
    self.encoder_layer_size = 5
    self.decoder_layer_size = 3
    self.batch_size = 6
    self.worker_num = 1
    self.max_epochs = 40
    self.lr = 0.0001
    self.teacher_forcing = 0.99
    self.seed = 1
    self.max_len = 80
    self.no_cuda = False
    self.save_name = 'model'
    self.mode = 'train'
    self.load_model = False
    self.model_path = './weight_file/epoch2'
```



Decoder RNN Layer



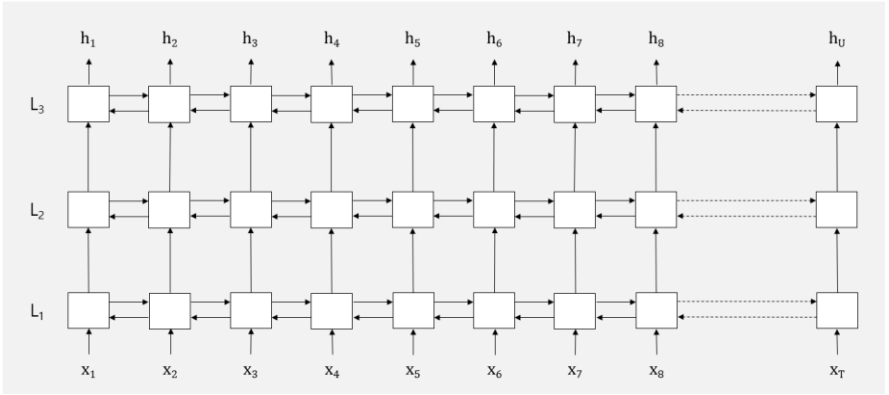
Encoder RNN Layer

내 머리속 Debug

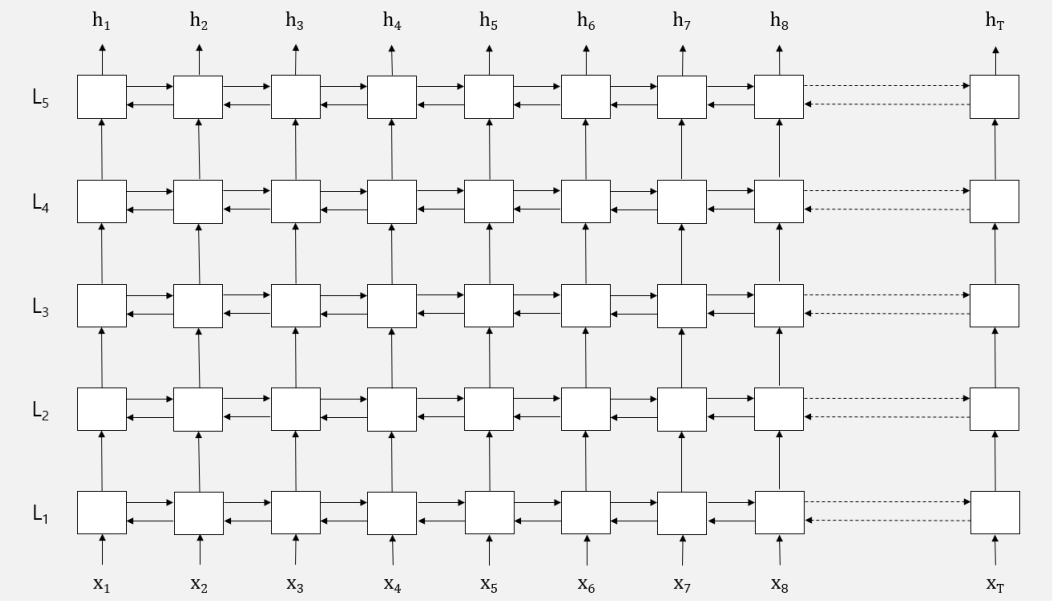
- 잘못 이해하고 있던 점

```
def __init__(self):
    self.use_bidirectional = True
    self.use_attention = True
    self.input_reverse = True
    self.use_augmentation = True
    self.use_pickle = True
    self.augment_ratio = 0.3
    self.hidden_size = 256
    self.dropout = 0.5
    self.encoder_layer_size = 5
    self.decoder_layer_size = 3
    self.batch_size = 6
    self.worker_num = 1
    self.max_epochs = 40
    self.lr = 0.0001
    self.teacher_forcing = 0.99
    self.seed = 1
    self.max_len = 80
    self.no_cuda = False
    self.save_name = 'model'
    self.mode = 'train'
    self.load_model = False
    self.model_path = './weight_file/epoch2'
```

상당히 깊은 구조였다...



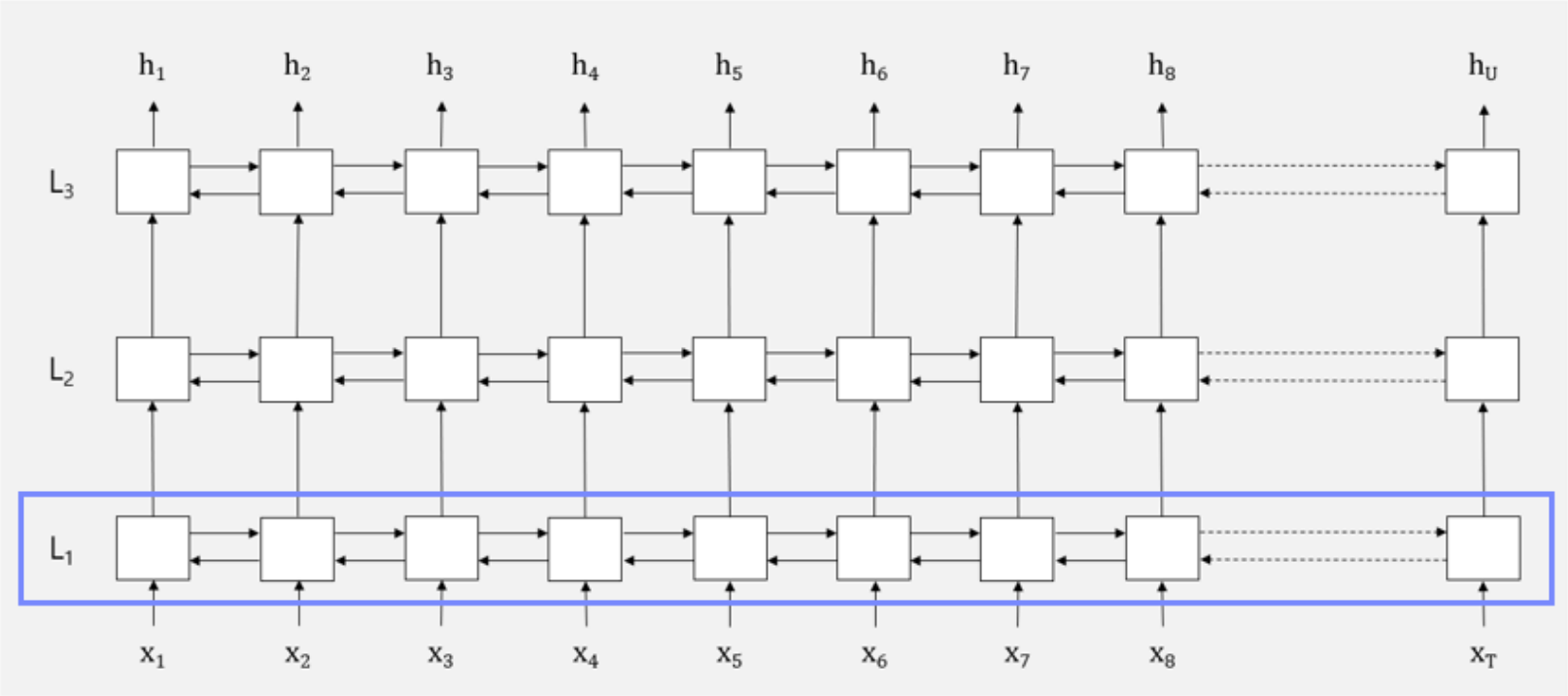
Decoder RNN Layer

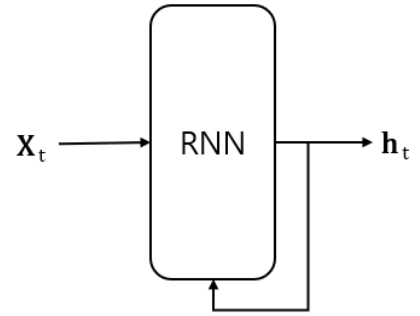


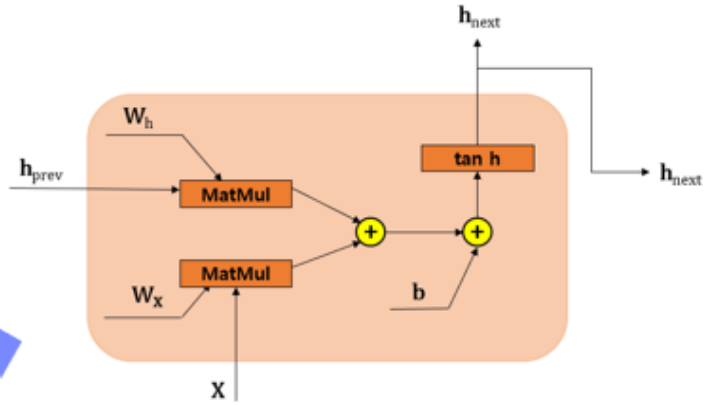
Encoder RNN Layer

다시 정리

- **RNN Layer 개념 다시 정리**



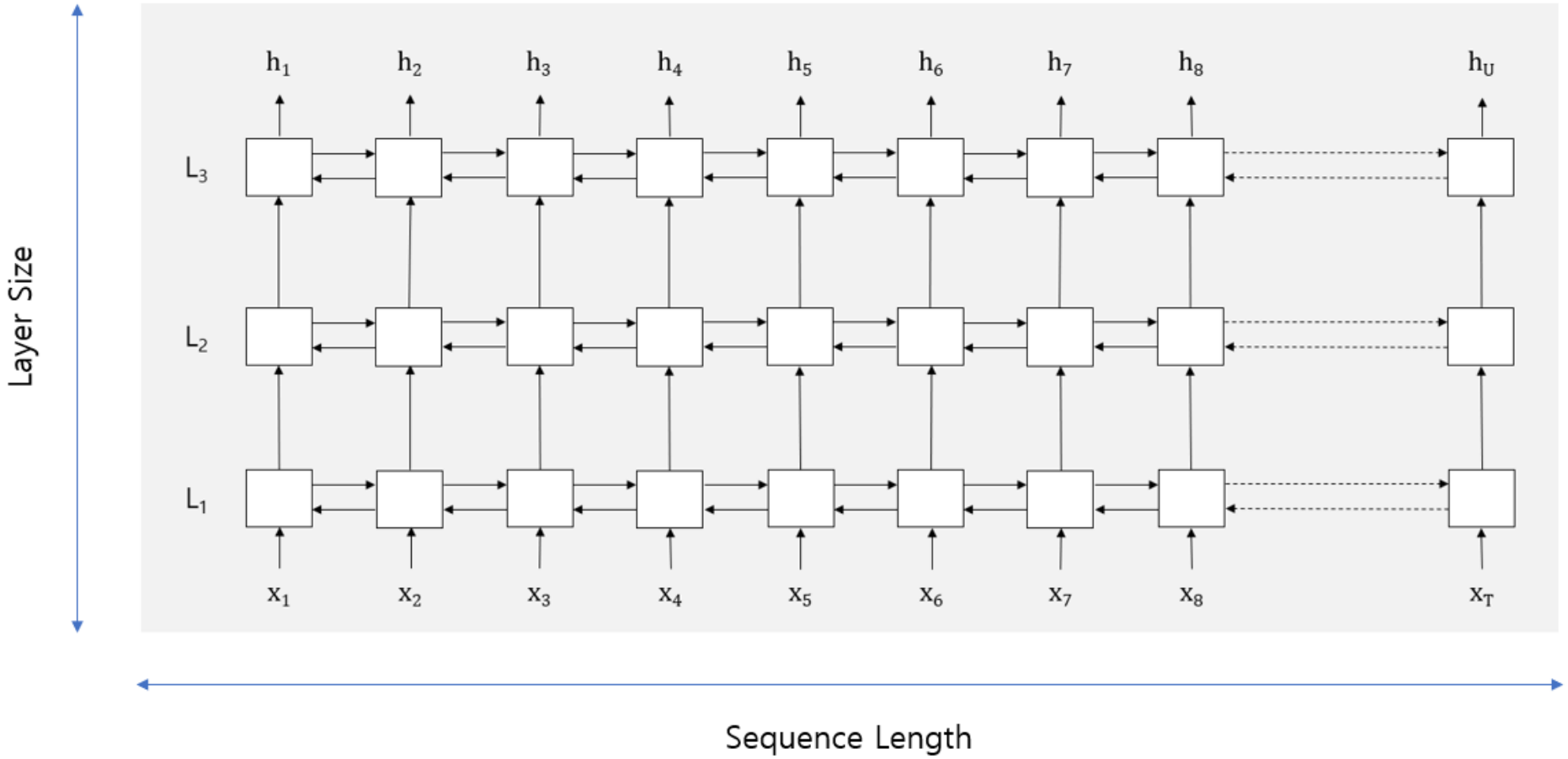




W_h, W_x는 고정된 채로,
X와 h_{prev}만 바뀌면서 Hidden State를 생성
(입력이 끝날 때까지 반복)
(가변 길이의 입력이 가능한 이유)

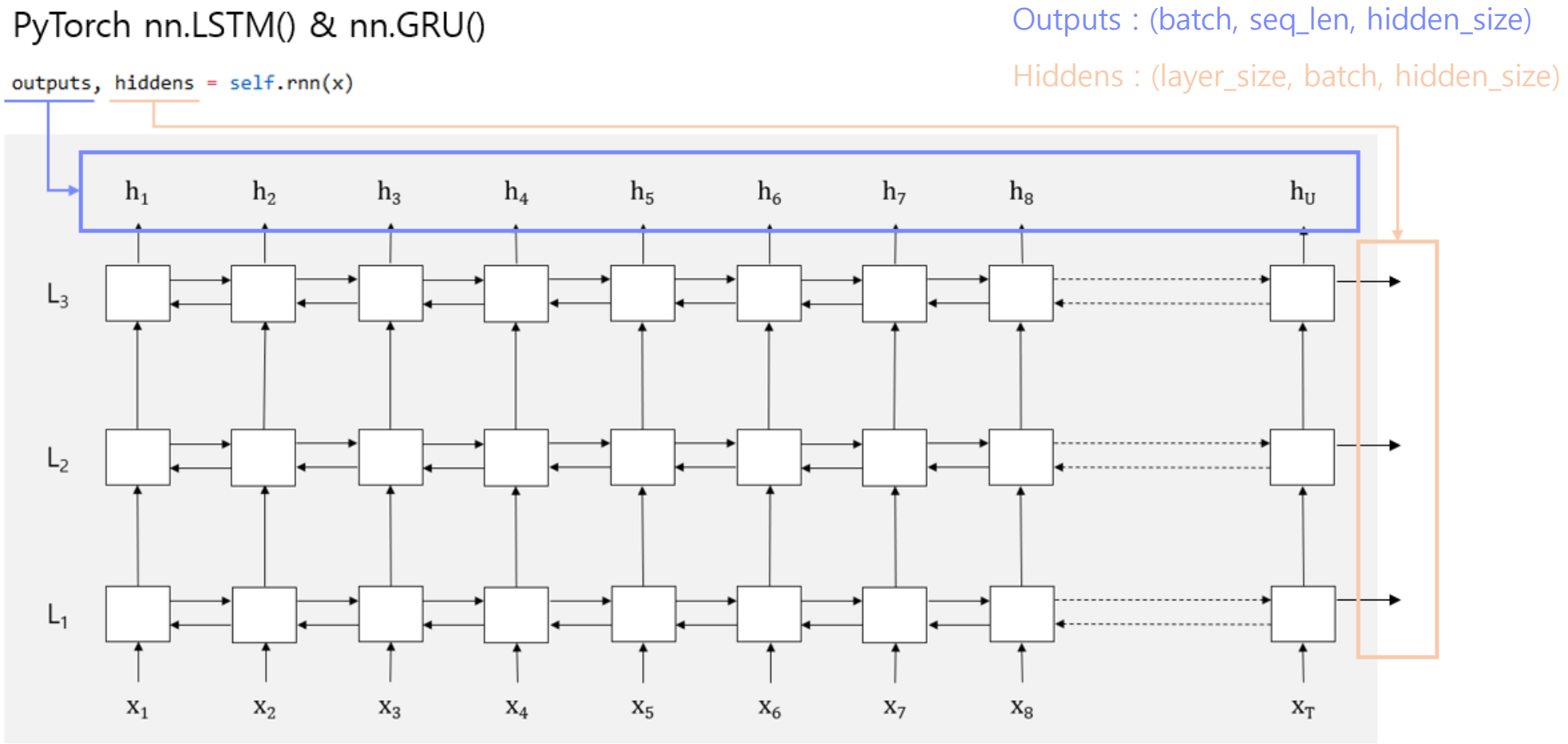
다시 정리

- **layer_size & seq_len**



다시 정리

- **PyTorch nn.LSTM() & nn.GRU()**



LAS Model

- **Listen, Attend and Spell Model**

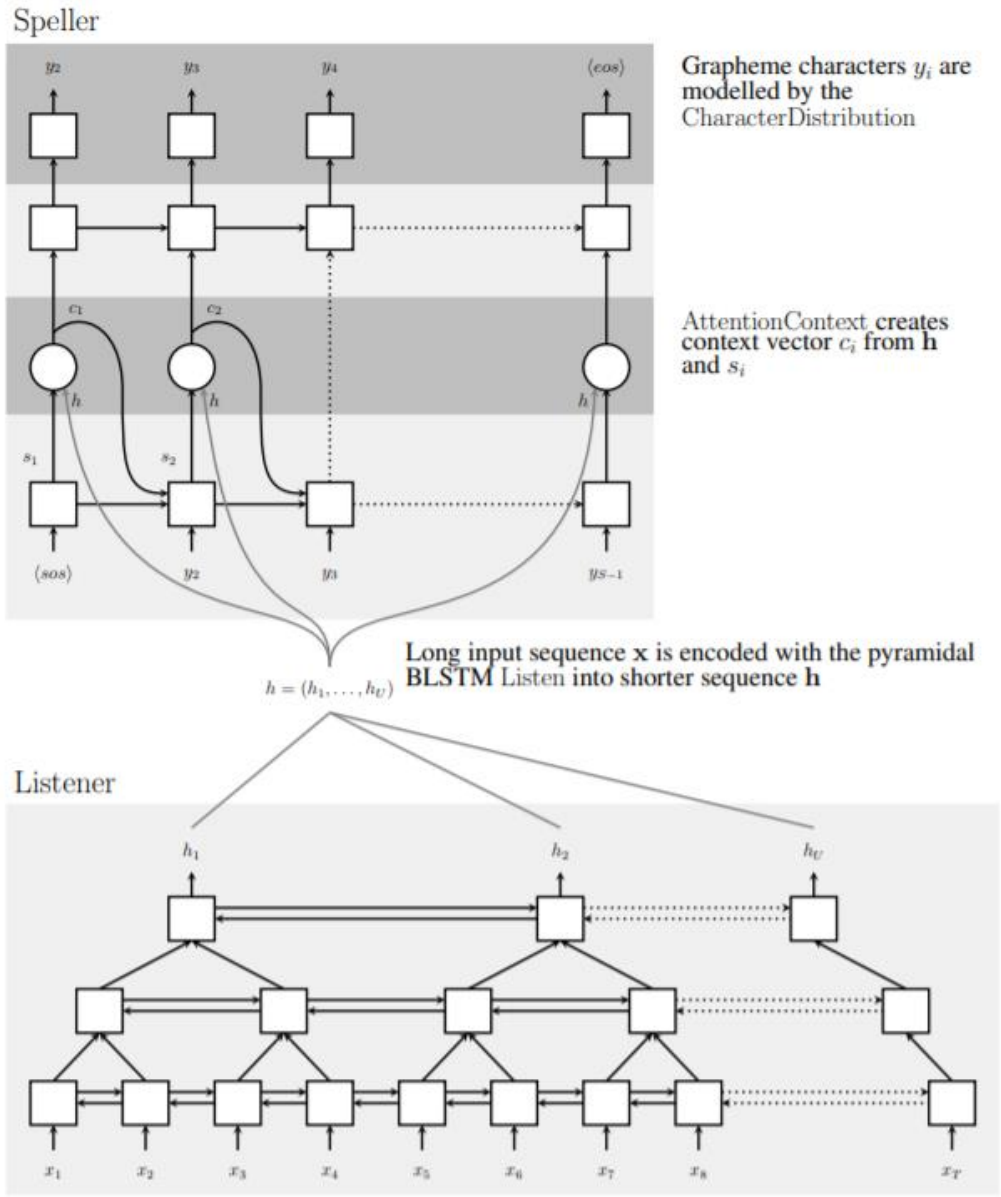
현재 우리가 사용하는 모델과 거의 같다고 해도 무방.

Seq2seq with Attention을 음성 인식에 적용한 모델.

※ 차이점 ※

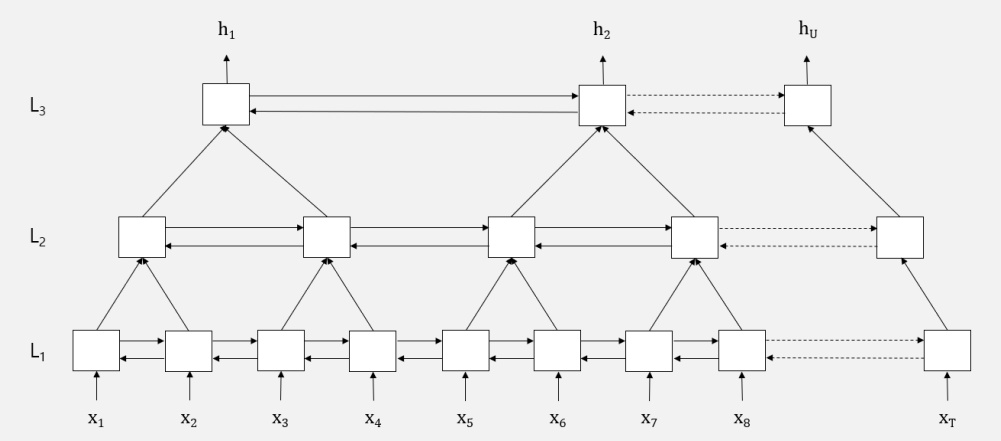
1. LAS는 인코더에서 pBLSTM(Pyrimidal LSTM)를 사용
2. 우리는 인코더 LSTM 레이어에 넣기 전 Convolution Layer 선행

(Deep Speech Style)

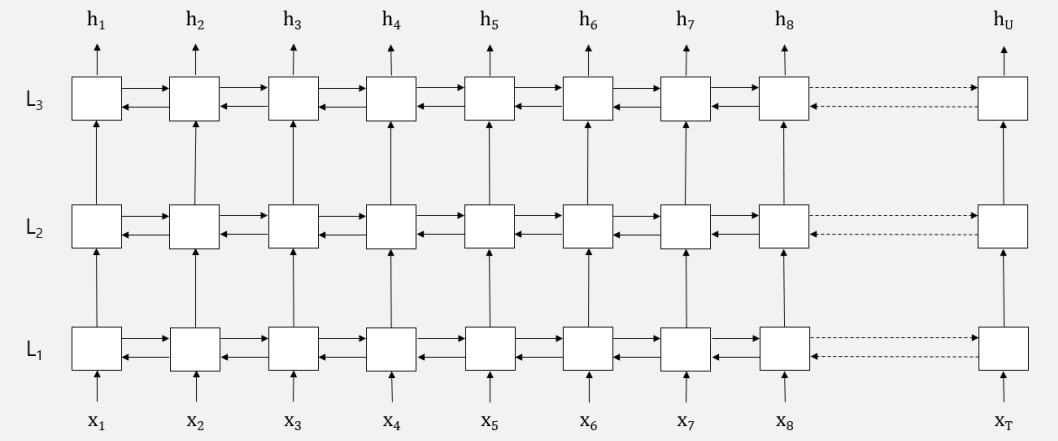


LAS Model

- **Pyrimidal LSTM**



pBLSTM

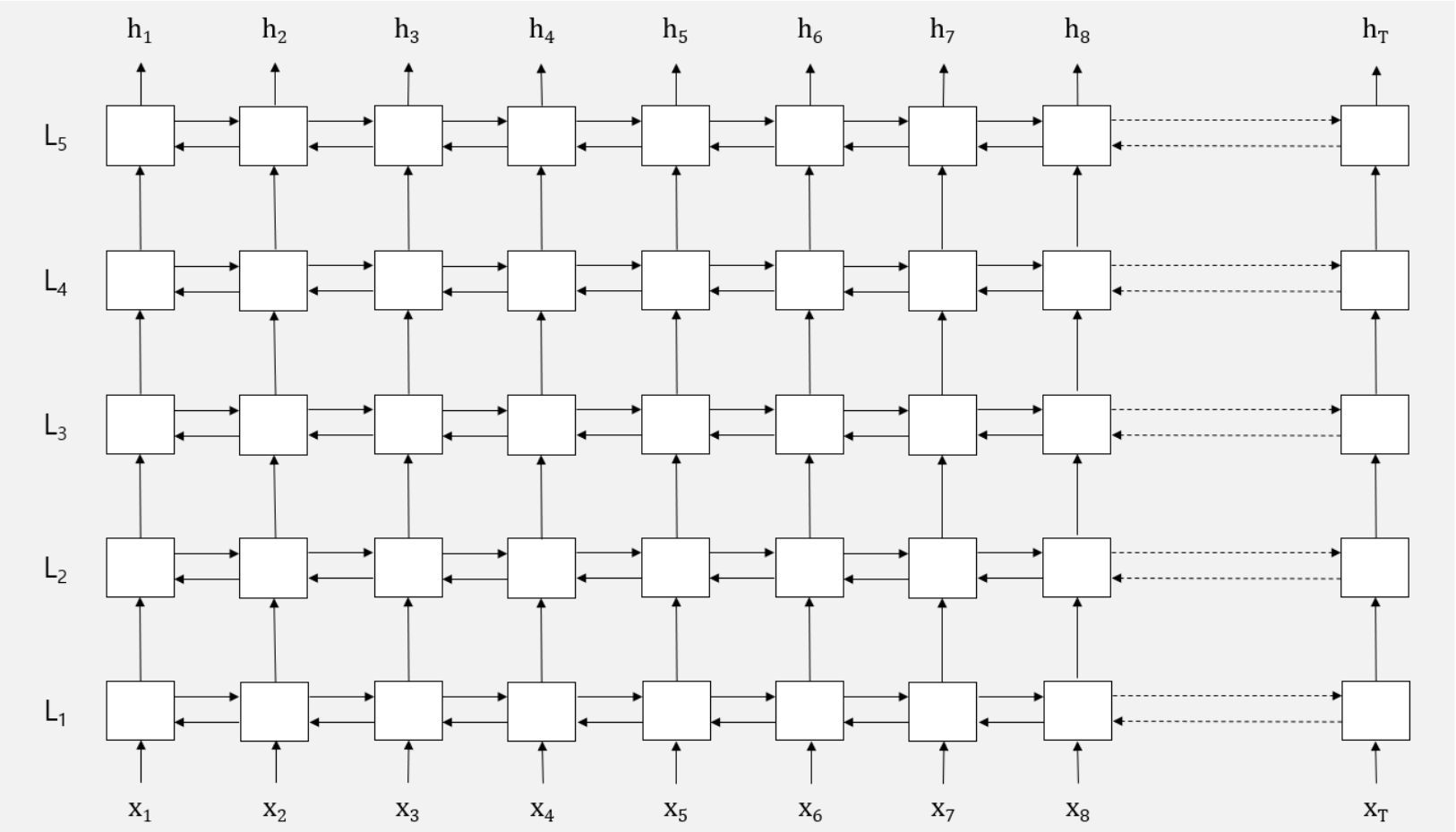


BLSTM

이전 레이어 2i,2i+1을 concatenate하여 다음 레이어의 i번째 RNN 셀의 입력으로 넣는 구조.
=> 더 압축하여 표현함으로써 Sequence Length를 줄일 수 있다.
이는 인코더 & 디코더 & 어텐션 메커니즘 모두에서 연산량 감소를 가능하게 한다.

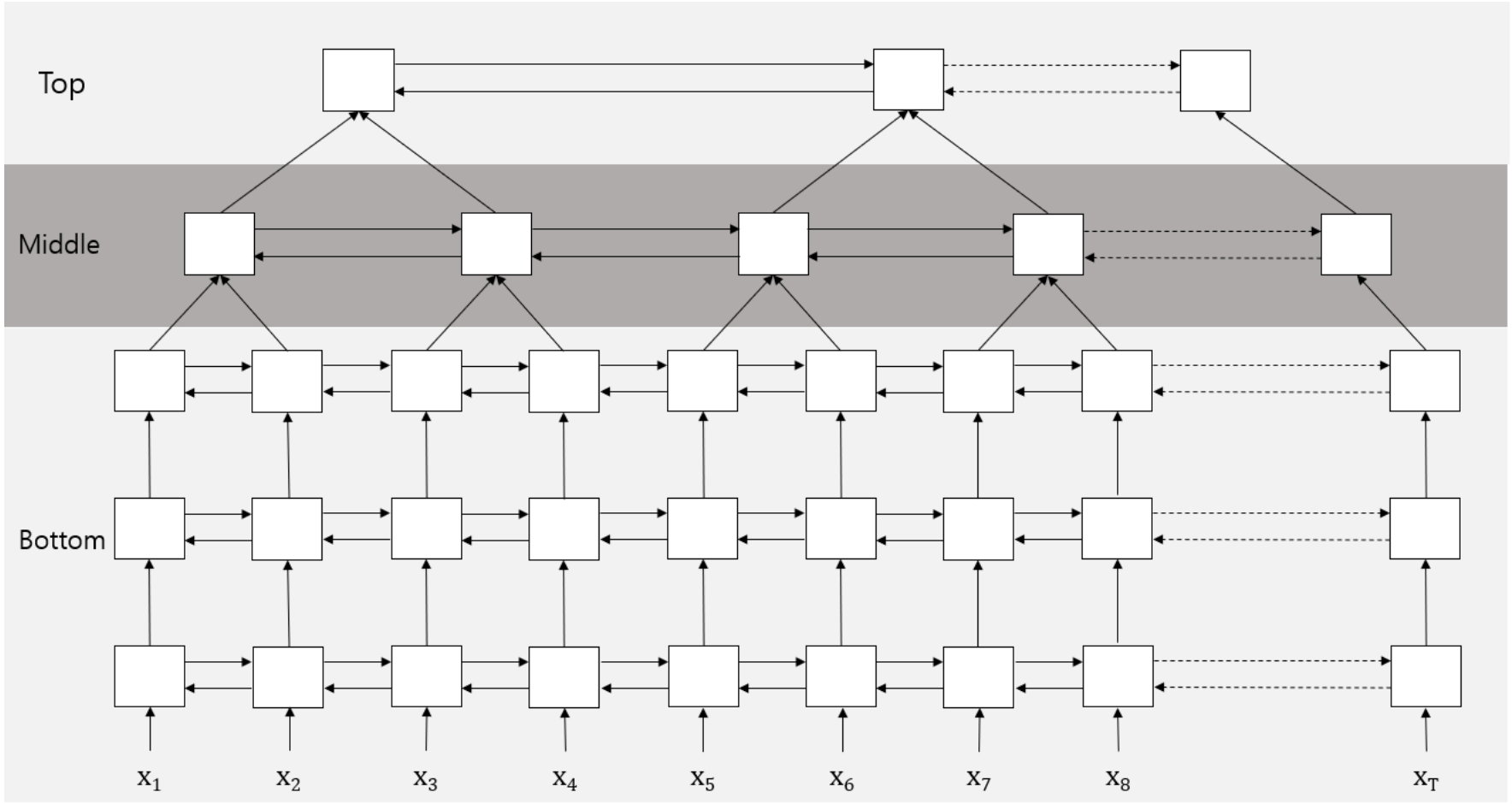
Proposal

- **Option: use_pyramidal == False**



Proposal

- **Option: use_pyramidal == True**



Proposal

- **Implement**

```
if use_pyramidal:
    self.bottom_layer_size = layer_size - 2
    self.bottom_rnn = self.rnn_cell(feature_size, hidden_size, self.bottom_layer_size, batch_first=True, bidirectional=bidirectional, dropout=dropout_p)
    self.middle_rnn = self.rnn_cell(hidden_size * 4, hidden_size, 1, batch_first=True, bidirectional=bidirectional, dropout=dropout_p)
    self.top_rnn = self.rnn_cell(hidden_size * 4, hidden_size, 1, batch_first=True, bidirectional=bidirectional, dropout=dropout_p)
else:
    self.rnn = self.rnn_cell(feature_size, hidden_size, layer_size, batch_first=True, bidirectional=bidirectional, dropout=dropout_p)
```

[use_pyrimidal시, bottom, middle, top으로 RNN 셀을 나누어 생성]

```
if self.use_pyramidal:
    bottom_outputs, _ = self.bottom_rnn(x)
    middle_inputs = self._make_pyramid(bottom_outputs)
    middle_outputs, _ = self.middle_rnn(middle_inputs)
    top_inputs = self._make_pyramid(middle_outputs)
    outputs, hiddens = self.top_rnn(top_inputs)
```

[Bottom → Middle → Top → outputs, hiddens]

```
def _make_pyramid(self, h_outputs):
    if h_outputs.size(1) % 2:
        zeros = torch.zeros((h_outputs.size(0), 1, h_outputs.size(2)))
        h_outputs = torch.cat([h_outputs, zeros], 1)
    return torch.cat([h_outputs[:, 0::2], h_outputs[:, 1::2]], 2)
```

[concatenate 2 layer]