Final Project:
Listen, Attend and Spell (LAS)

Nguyen Huu Kim
2020.06.24

Contents

- Introduction
- Implementation and Results

Introduction

- Automatic Speech Recognition Model (ASR)
- Input: Acoustic features $\boldsymbol{x} = (x_1, \dots, x_T)$

  Output: Characters or phonemes $\boldsymbol{y} = (< sos >, y_1, \dots, y_S, < eos >)$
- Objective: maximize likelihood

$$P(\boldsymbol{y}|\boldsymbol{x}) = \prod_{\boldsymbol{i}} \boldsymbol{P}(y_i|\boldsymbol{x}, y_{<i})$$

Introduction

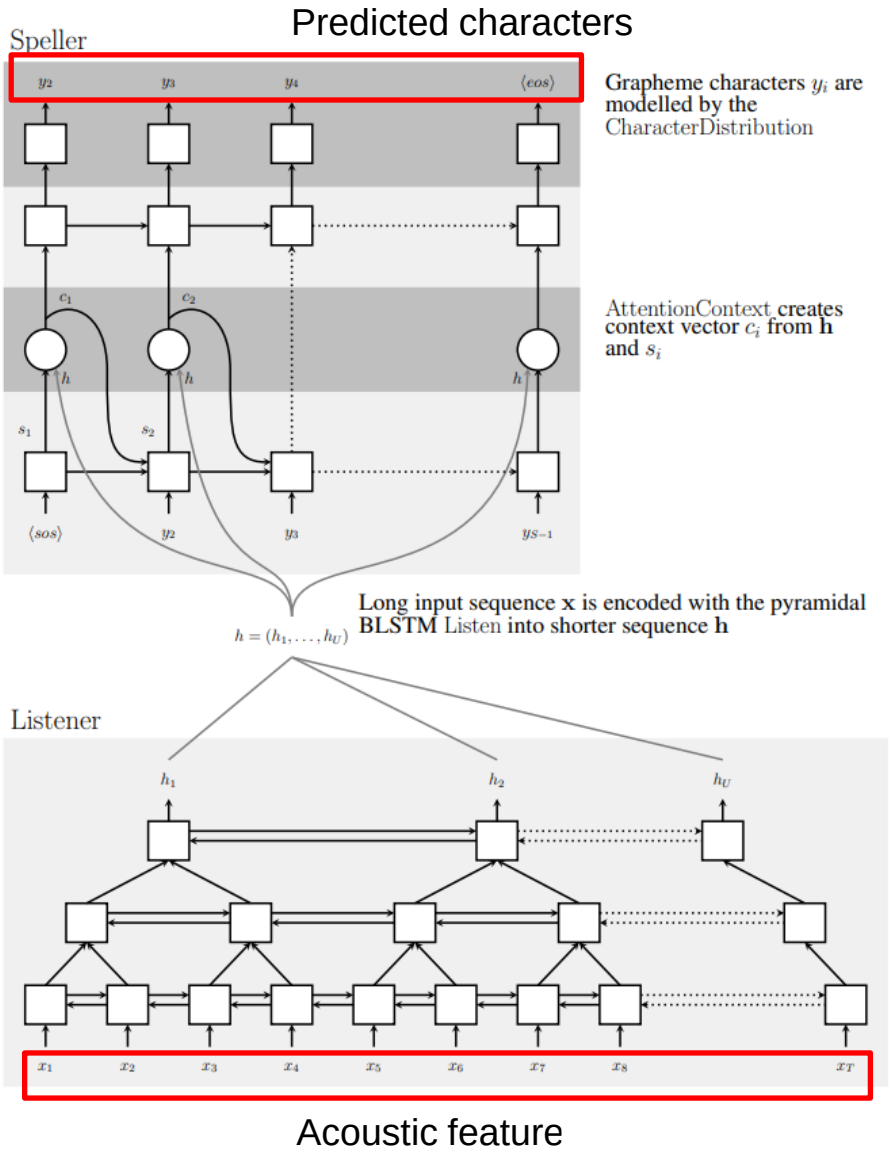
- Speller:
  - Attention-based LSTM transducer

$$c_i = \text{AttentionContext}(s_i, \mathbf{h})$$
$$s_i = \text{RNN}(s_{i-1}, y_{i-1}, c_{i-1})$$
$$P(y_i|\mathbf{x}, y_{<i}) = \text{CharacterDistribution}(s_i, c_i)$$

- Listener
  - Pyramid BLSTM structure

$$h_i^j = \text{BLSTM}(h_{i-1}^j, h_i^{j-1})$$

$$h_i^j = \text{pBLSTM}(h_{i-1}^j, \left[h_{2i}^{j-1}, h_{2i+1}^{j-1}\right])$$



Implementation

- Experiment settings:
  - Dataset: VCTK Corpus
  - Sampling rate: 16kHz
  - Frame length: 50 ms
  - Hop length: 12.5 ms
  - Window: Hann

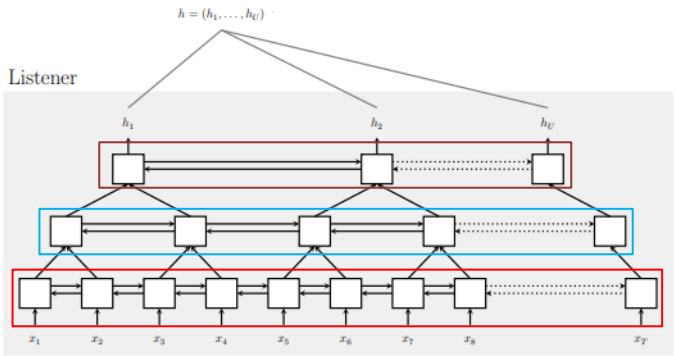
Hyperparameter setup

Transcription features	Phonemes
Acoustic features	80-dim log mel-spectrogram
Optimizer	Adam
Learning rate	10^{-3}

Implementation

- Different with original paper:
  - 2 layer pyramid BLSTM vs. 3 layer pyramid BLSTM
  - VCTK corpus vs. private Google dataset

Implementation



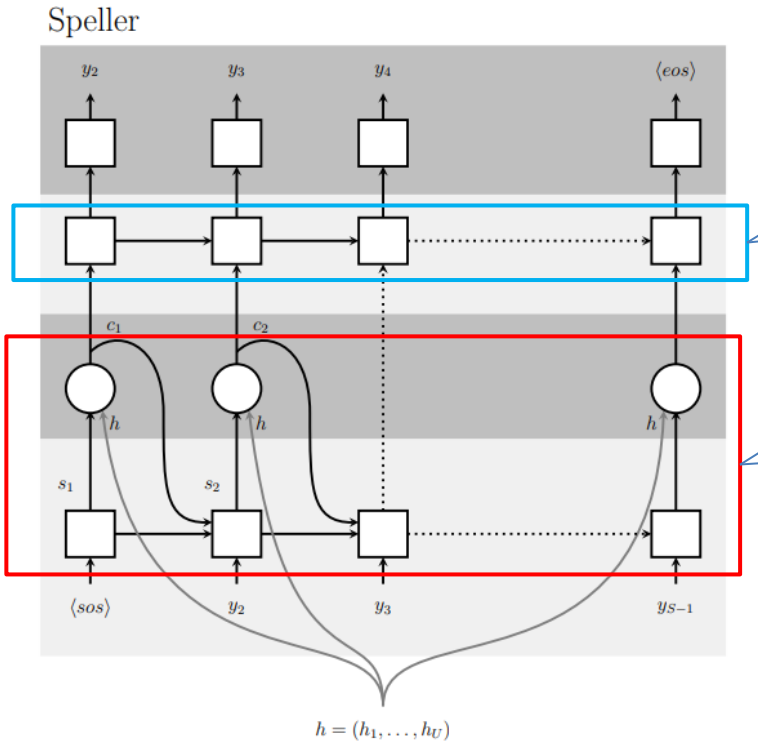
removed

lstm 2

lstm 1

```python
class Listener(nn.Module):
    '''
    - Simple 2 layer bidirectional LSTM

    '''
    def __init__(self, hparams):
        super(Listener, self).__init__()

        if hparams.spemb_input:
            input_dim = hparams.n_mel_channels + hparams.speaker_embedding_dim
        else:
            input_dim = hparams.n_mel_channels

        self.lstm1 = nn.LSTM(input_dim, int(hparams.audio_encoder_hidden_dim / 2),
                             num_layers=1, batch_first=True, bidirectional=True)
        self.lstm2 = nn.LSTM(hparams.audio_encoder_hidden_dim*hparams.n_frames_per_step_encoder,
                             int(hparams.audio_encoder_hidden_dim / 2),
                             num_layers=1, batch_first=True, bidirectional=True)

        self.concat_hidden_dim = hparams.audio_encoder_hidden_dim*hparams.n_frames_per_step_encoder
        self.n_frames_per_step = hparams.n_frames_per_step_encoder

    def forward(self, x, input_lengths):
        '''
        x  [batch_size, mel_bins, T]

        return [batch_size, T, channels]
        '''
        x = x.transpose(1, 2)

        x_sorted, sorted_lengths, initial_index = sort_batch(x, input_lengths)

        x_packed = nn.utils.rnn.pack_padded_sequence(
            x_sorted, sorted_lengths.cpu().numpy(), batch_first=True)

        self.lstm1.flatten_parameters()
        outputs, _ = self.lstm1(x_packed)

        outputs, _ = nn.utils.rnn.pad_packed_sequence(
            outputs, batch_first=True, total_length=x.size(1)) # use total_length make sure the recovered

        outputs = outputs.reshape(x.size(0), -1, self.concat_hidden_dim)

        output_lengths = torch.ceil(sorted_lengths.float() / self.n_frames_per_step).long()
        outputs = nn.utils.rnn.pack_padded_sequence(
            outputs, output_lengths.cpu().numpy() , batch_first=True)

        self.lstm2.flatten_parameters()
        outputs, _ = self.lstm2(outputs)

        outputs, _ = nn.utils.rnn.pad_packed_sequence(
            outputs, batch_first=True)

        return outputs[initial_index], output_lengths[initial_index]
```

Implementation

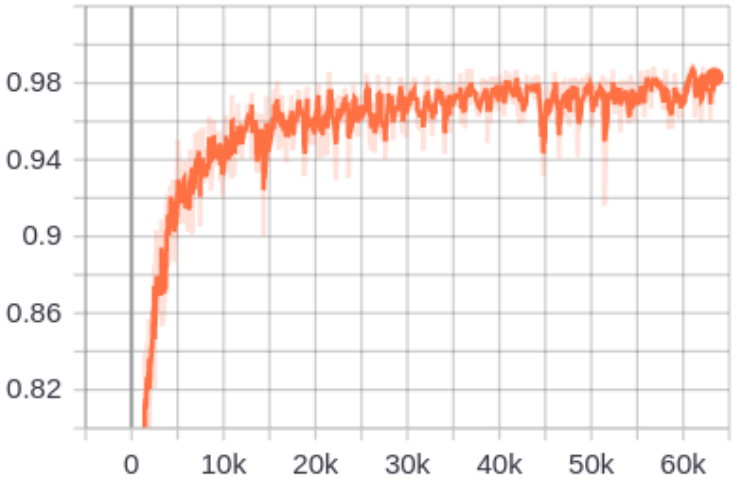



```python
def decode(self, decoder_input):

    # import pdb
    # pdb.set_trace()
    cell_input = torch.cat((decoder_input, self.attention_context), -1)
    self.decoder_hidden, self.decoder_cell = self.decoder_rnn(
        cell_input,
        (self.decoder_hidden,
        self.decoder_cell))

    attention_weigths_cat = torch.cat(
        (self.attention_weigths.unsqueeze(1),
        self.attention_weigths_cum.unsqueeze(1)),dim=1)

    self.attention_context, self.attention_weigths = self.attention_layer(
        self.decoder_hidden,
        self.memory,
        self.processed_memory,
        attention_weigths_cat,
        self.mask)

    self.attention_weigths_cum += self.attention_weigths

    hidden_and_context = torch.cat((self.decoder_hidden, self.attention_context), -1)

    hidden = self.project_to_hidden(hidden_and_context)

    # dropout to increasing g
    logit = self.project_to_n_symbols(F.dropout(hidden, 0.5, self.training))

    return hidden, logit, self.attention_weigths
```

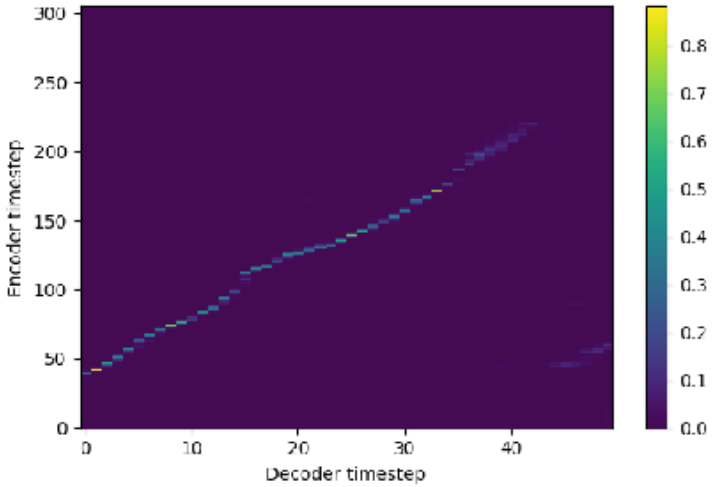
FC layers

Results

Identification task: Accuracy 98%



Training accuracy



Attention plot