Introduction to Deep Learning

22. Encoder-Decoder, Seq2seq

STAT 157, Spring 2019, UC Berkeley

Alex Smola and Mu Li

courses.d2l.ai/berkeley-stat-157
Encoder-Decoder
Rethink about CNN

• Encoder: encode inputs into intermediate presentation (features)
• Decoder: decode the presentation into outputs
Rethink about RNN

• Encoder: present a piece of text as a vector
• Decoder: decode the presentation into outputs
The Encoder-decoder Architecture

- A model is partitioned into two parts
  - The encoder process inputs
  - The decoder generates outputs
The Base Class for an Encoder

class Encoder(nn.Block):
    def __init__(self, **kwargs):
        super(Encoder, self).__init__(**kwargs)

    def forward(self, X):
        raise NotImplementedError
The Base Class for a Decoder

• Create state with the encoder outputs and any other infos

```python
class Decoder(nn.Block):
    def __init__(self, **kwargs):
        super(Decoder, self).__init__(**kwargs)

    def init_state(self, enc_outputs, *args):
        raise NotImplementedError

    def forward(self, X, state):
        raise NotImplementedError
```
class EncoderDecoder(nn.Block):
    def __init__(self, encoder, decoder, **kwargs):
        super(EncoderDecoder, self).__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder

    def forward(self, enc_X, dec_X, *args):
        enc_outputs = self.encoder(enc_X)
        dec_state = self.decoder.init_state(enc_outputs, *args)
        return self.decoder(dec_X, dec_state)
Seq2seq
Machine Translation

• Given a sentence in a source language, translate into a target language
• These two sequences may have different lengths
Seq2seq

- The encoder is a RNN to read input sequence
- The decoder uses another RNN to generate output
Encoder/Decoder Details

- The encoder is a standard RNN model without the output layer.
- The encoder’s hidden state in last time step is used as the decoder’s initial hidden state.
Training

• The decoder is feed with the targeted sentence during training

Encoder

Decoder

Predict:
Code...
Beam Search
Greedy Search

- We used greedy search in the seq2seq model during predicting
- It could be suboptimal

<table>
<thead>
<tr>
<th>Time step</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.5</td>
<td>0.1</td>
<td>0.2</td>
<td>0.0</td>
</tr>
<tr>
<td>B</td>
<td>0.2</td>
<td>0.4</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>C</td>
<td>0.2</td>
<td>0.3</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>&lt;eos&gt;</td>
<td>0.1</td>
<td>0.2</td>
<td>0.2</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Greedy search: $0.5 \times 0.4 \times 0.4 \times 0.6 = 0.048$

<table>
<thead>
<tr>
<th>Time step</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.5</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>B</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
<td>0.2</td>
</tr>
<tr>
<td>C</td>
<td>0.2</td>
<td>0.3</td>
<td>0.2</td>
<td>0.1</td>
</tr>
<tr>
<td>&lt;eos&gt;</td>
<td>0.1</td>
<td>0.2</td>
<td>0.1</td>
<td>0.6</td>
</tr>
</tbody>
</table>

A better choice: $0.5 \times 0.3 \times 0.6 \times 0.6 = 0.054$
Exhaustive Search

• For every possible sequence, compute its probability and pick the best one
• If output vocabulary size is \( n \), and max sequence length \( T \), then we need to examine \( n^T \) sequences
  • It’s computationally infeasible

\[
n = 10000, \quad T = 10 : \quad n^T = 10^{40}
\]
Beam Search

- We keep the best $k$ (beam size) candidates for each time step.
- Examine $kn$ sequences by adding a new item to a candidate, and then keep the top-$k$ ones.

![Diagram showing Beam Search](image-url)
Beam Search

• Time complexity is $O(knT)$

$$k = 5, \quad n = 10000, \quad T = 10 : \quad knT = 5 \times 10^5$$

• The final score for each candidate is

$$\frac{1}{L^\alpha} \log \mathbb{P}(y_1, \ldots, y_L) = \frac{1}{L^\alpha} \sum_{t' = 1}^{L} \log \mathbb{P}(y_{t'} \mid y_1, \ldots, y_{t'-1}, c)$$

• Often $\alpha = 0.75$