Seq2seq Translation model

CSC401 / 2511 tutorial
Feb 26, 2020
Zining Zhu
Agenda

● Recap: Recurrent models: RNN, LSTM, GRU
● Seq2seq model
  ○ Setup
  ○ Train
  ○ Run
● Beam Search example
● Misc. reminders for A2
● Questions
RNN vs. LSTM vs. GRU

https://colah.github.io/posts/2015-08-Understanding-LSTMs/
RNN models in PyTorch

- For A2, consider RNN, LSTM, and GRU
- The forward() method:
  - Input: hidden_state, x
  - For the first step, hidden_state contains all 0 by default
  - hidden_state for LSTM vs. RNN / GRU
  - x is a tensor of shape (seq_len, batch_size, dim), unless you want batch_first=True
  - Training and eval difference. module.train(), module.eval()

- Check out pytorch docs for details.
Seq2seq model: Setup

- Encoder + Decoder
- Both nn.Modules

Image source:
https://towardsdatascience.com/sequence-to-sequence-model-introduction-and-concepts-44d9b41cd42d
Seq2seq model: Encoder

- Encoder
  - Consists of an `nn.Embedding` and a bidirectional RNN (one of LSTM, GRU, and RNN)
  - Input: F (sequence len S, batch size N)
  - The `nn.Embedding` maps input F onto the rnn input, x (S, N, RNN dimension I)
  - Encoder outputs all hidden states: h (S, N, hidden size H*2)
  - Same for both training and running.
Seq2seq model: Setup

- **Decoder**
  - Consists of an RNN / LSTM / GRU and a feed-forward Linear layer.
  - Input / output dependent on training and running status.
- **Two versions: without and with attention.**
  - DecoderWithoutAttention: Encoder -> h -> first_hidden_state
  - DecoderWithAttention: Attend to the h (encoder outputs).
  - Nevertheless, each decoder step computes log prob of current step.
Seq2seq model: Train

- “Teacher forcing”
- For each step: given the input and a first hidden state, should learn the correct output (i.e., next token).
  - The input comes from the current token.
  - See forward() in a2_abcs.py for details.
- Maximum likelihood training
- Train the EncoderDecoder nn.Module end-to-end!
  - Gradient passed through both decoder and encoder
  - Remember to optim.zero_grad() after loss.backward()
  - If computing total loss, use loss.item() not loss (which preserves the massive graph)
Sidenote: computation graph in PyTorch

- At each operation, a graph node is created
  - Each tensor contains reference to the graph
- Gradients are computed in `.backward()`
- Some intermediate nodes are discarded
  - BackProp 2nd time error
  - `optim.zero_grad()`
  - See [this notebook](https://bit.ly/2HAXlQP) for examples.
- The graphs could use up your memory.
  - Graph is gc’ed with tensors.
  - `tensor.detach()` or `tensor.item()`
Seq2seq model: Run

- “Beam Search Decoding”
- Maintain a “beam” of K partially decoded sentences
- At each step:
  - Compute the likelihood of candidates.
  - Preserve the K candidates with highest likelihood, discard the rest.
- When to stop decoding?
  - When your top candidate comes to an <eos>
- Go over example in slides P55-60
Misc.

- Make sure the shape / types of tensors are correct.
  - Especially in broadcasting / slicing.
  - CPU / GPU tensor types mismatch.
- Debug on small datasets
  - E.g., setting `--max-vocab 100` in building vocabulary. See Appendix A.4 on handout.
- Use a GPU to train the large model.
  - BA GPU labs / AWS EC2 / Google Cloud / etc.,
  - Make sure the code will run on teach.cs -- otherwise will receive 0 marks.
- Ask and answer others’ questions on piazza
  - But don’t post your solution codes.
- Start early! Due March 9 @ 7pm.
Labs with GPU

- See teaching lab availabilities at [https://www.teach.cs.toronto.edu/faq.html#ABOUT4](https://www.teach.cs.toronto.edu/faq.html#ABOUT4)
- Sometimes labs are occupied for classes: (dot means “booked”)

<table>
<thead>
<tr>
<th>Time</th>
<th>BA2200</th>
<th>BA3175</th>
<th>BA3185</th>
<th>BA3195</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monday</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10-11am</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>12-2pm</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>2-3pm</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>8-9pm</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Tuesday</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-3pm</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>6-8pm</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>8-9pm</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Wednesday</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12-2pm</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>2-4pm</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>4-5pm</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>6-8pm</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Thursday</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9-9pm</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Friday</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10-12pm</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>1-4pm</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>4-6pm</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>