Seq2Seq Models
&
Attention Mechanism

Lili Mou
lmou@ualberta.ca
lili-mou.github.io
Seq2Seq

• Question:

Why do we feed back the generated words?

How can we train Seq2Seq models?

How do we do inference?

Why do we feed back the generated words?

- **First thought:** Feeding back is unnecessary
  - $y_2$ is predicted from $h_2$
  - $h_2$ depends on $h_1$
  - $y_1$ depends on $h_1$

  $\Rightarrow$ $y_1$ brings no information
Multi-Modal Distribution

- Continuous distribution
- Discrete distribution
  - Image in some embedding space, sentences are multi-modal distributed
  - "A beats B" vs. "B is beaten by A"
Training Seq2Seq Models

- Decoder's input layer
  - Attempt #1: Feed in the predicted words
  - Attempt #2: Feed in the groundtruth word
  - Scheduled sampling

- Loss $J = J_1 + J_2 + \cdots + J_T$
  - Suppose we known “groundtruth” target sequence
  - Recall BP with multiple losses
Inference

- Decoder’s input layer
  - Feed in the predicted words

- When do we terminate?
  - Include a special token “EOS” (end of sequence) in training
  - If “EOS” is predicted, the sentence is terminated by def
Caveat

- Batch implementation
  - Padding EOS or 0 vector => Incorrect
  - Masking => Correct

\[ \tilde{h}_t = \text{RNN}(h_{t-1}, x_t) \]
\[ h_t = (1 - m)h_{t-1} + m\tilde{h}_t \]

- Implementation should always be equivalent to math derivations
Inference Criteria

• Single-output classification
  - Max a posteriori inference ⇔ Minimal empirical loss
  - \( y = \text{argmax } p(y | x) \)

• Sentence generation
  - If we want to generate the “best” sentence:
    \( y = \text{argmax } p(y | x) \)
  - Is greedy correct? \( y_i = \text{argmax } p(y_i | y_{<i}, x) \)
  - The cost of exhaustive search?
# Greedy vs. Exhaustive Search

<table>
<thead>
<tr>
<th></th>
<th>Greedy</th>
<th>Beam Search</th>
<th>Exhaustive search</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>For each step</strong></td>
<td>Pick the best word</td>
<td>Try a few best words</td>
<td>Try every word</td>
</tr>
<tr>
<td><strong>Maintain</strong></td>
<td>One sequence</td>
<td>Several good partial sequences</td>
<td>All possible combinations</td>
</tr>
</tbody>
</table>

**Diagram:**

1. **Greedy:**
   - Le site officiel du gouvernement du Canada

2. **Beam Search:**
   - Various possible paths
   - EOS

3. **Exhaustive Search:**
   - All possible combinations
   - EOS
Beam Search

B = 2
Beam Search

B=2

Le site officiel du gouvernement du Canada

The official website of the Government of Canada

argmax

EOS
Beam Search

B = 2

The official website of the Government of Canada
Beam Search

- A list of best partial sequences $S$
- For every decoding step $t$
  - For every partial seq $s \in S$ and every word $w \in \mathcal{V}$
    - Expand $s$ as $(s, w)$
    - $S = \text{top-}B$ expanded subsequences among all $(s, w)$
- Return the most probable sequence in the beam
  (existing a terminated sequence better than all $s \in S$)
Issues with Autoregressiveness

- Error accumulation
  - 1st word good, 2nd worse, 3rd even worse, etc.

- Label bias
  - Not “label imbalance” problem
  - BS bias towards high probable words at the beginning
  - Locally normalized models prefer high probable (but possibly unimportant) words
Information Bottleneck

- Last hidden state
  - Has to capture all source information
- Average/Mean pooling
  - Still loses information
  - Not directly related to the current decoded word
Attention Mechanism

• Dynamically aligning words
  - **Average pooling** => **weighted pooling**
  - Alignment dependents on the current word to be generated
  - Alignment to a particular source word obviously also dependents on that source word itself
Convex vs Linear Weighting

- Average pooling
  \[ c = \frac{1}{N} h_1 + \cdots + \frac{1}{N} h_N \]

- Weighted pooling
  \[ c = \alpha_1 h_1 + \cdots + \alpha_N h_N \]

What are \( \alpha_1, \cdots, \alpha_N \)?
Computing Attention

Score (-Energy)

\[ s_j^{(t)} = s(h_{j}^{(\text{tar})}, h_i^{(\text{src})}) \]

Unnormalized measure

\[ \tilde{\alpha}_j^{(t)} = \exp(s_j^{(t)}) \]

Probability

\[ \alpha_j^{(t)} = \frac{\exp(s_j^{(t)})}{\sum_{j'} \exp(s_{j'}^{(t)})} \]

Denominator: Partition function

Inner-product

\[ s_j^{(t)} = \left( h_{j}^{(\text{tar})} \right)^T h_i^{(\text{src})} \]

Metric learning

\[ s_j^{(t)} = \left( h_{j}^{(\text{tar})} \right)^T W h_i^{(\text{src})} \]

Neural layer

\[ s_j^{(t)} = u^T f(W[h_{j}^{(\text{tar})}; h_i^{(\text{src})}]) \]
Where are we?

Attention: in the convex hull

Average pooling

Max pooling

Linear combination: anywhere in the space

Picking the best (how do you know)

This is not too wrong.

"Meaning is use" —Wittgenstein

In machine learning, how you train is how you predict
Traditional Phrase-Based MT

\[
\Pr(e|f) = \frac{\Pr(e) \Pr(f|e)}{\Pr(f)}
\]

\[
\hat{e} = \arg\max_e \Pr(e) \Pr(f|e)
\]

Explicitly modeling alignment

\[
\Pr(f|e) = \sum_a \Pr(f, a|e)
\]

\[
\Pr(f, a|e) = \Pr(m|e) \prod_{j=1}^{m} \Pr(a_j|a_1^{j-1}, f_1^{j-1}, m, e) \Pr(f_j|a_1^j, f_1^{j-1}, m, e)
\]

IBM Models 1—5: A spectrum of simplifications

Attention is all you need

- Information processed by multi-head attention
- Sinusoidal position embedding
  - BERT uses learned position embedding

Transformer
(Horrible terminology)

Attention beyond MT

- Attention probability is essentially a softmax
  - with varying # of target classes

- Attention content is aggregating information by weighted sum
  - Especially # of entities may change
More Applications

- Dialogue systems
- Summarization
- Paraphrase generation
- etc.

Encoder does not have to be a sequence model

- Table-to-test generation
- Graph-to-text generation

Memory-Based Network

Question answering (synthetic dataset)

Sam walks into the kitchen.
Sam picks up an apple.
Sam walks into the bedroom.
Sam drops the apple.

Q: Where is the apple?
A. Bedroom

Brian is a lion.
Julius is a lion.
Julius is white.
Bernhard is green.

Q: What color is Brian?
A. White

Mary journeyed to the den.
Mary went back to the kitchen.
John journeyed to the bedroom.
Mary discarded the milk.

Q: Where was the milk before the den?
A. Hallway

Memory-Based Network

Basically a multi-layer attention network

\[ p_i = \text{Softmax}(u^T m_i) \quad o = \sum_i p_i c_i \]

# Neural Turing Machine

<table>
<thead>
<tr>
<th>Chomsky hierarchy</th>
<th>Grammar</th>
<th>Language</th>
<th>Automata</th>
<th>Neural analog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type-3</td>
<td>$A \rightarrow aB \mid a$</td>
<td>Regular expression</td>
<td>Finite state machine</td>
<td>RNN</td>
</tr>
<tr>
<td>Type-2</td>
<td>$A \rightarrow a$</td>
<td>Context-free</td>
<td>ND Pushdown automata</td>
<td></td>
</tr>
<tr>
<td>Type-1</td>
<td>$\alpha A \beta \rightarrow \alpha \gamma \beta$</td>
<td>Context-sensitive</td>
<td>Esoteric</td>
<td></td>
</tr>
<tr>
<td>Type-0</td>
<td>$\alpha A \beta \rightarrow \gamma$</td>
<td>Recursive enumerable</td>
<td>Turing machine</td>
<td>NTM</td>
</tr>
</tbody>
</table>
A Rough Thought on RNN Computational Power

• If states are discrete, RNN is FSM
• # distinct states $\propto \exp(\text{units})$
• However, they are not free states
  - Transitions subject to a parametric function
• Unknown (at least to me) how real-valued states add to computational power
• At least, using denseness of real numbers to express potentially infinite steps of recursion is inefficient
Neural Turing Machine

- Augment RNN with a memory pad
- Read & write by memory addressing
- Attention-based memory addressing
  - Content-based addressing
  - Allocation-based addressing
  - Link-based addressing

Neural Turing Machine

Content-based memory addressing

$$C(M, k, \beta)[i] = \frac{\exp\{D(k, M[i, \cdot])/\beta\}}{\sum_j \exp\{D(k, M[j, \cdot])/\beta\}}$$

Dynamic memory allocation

$$\psi_t = \prod_{i=1}^{R} (1 - f_t^i w_t^{R,i})$$

Temporal linkage-based memory addressing

$$L_0[i,j] = 0 \quad \forall i,j$$
$$L_t[i,i] = 0 \quad \forall i$$
$$L_t[i,j] = (1 - w_t^w[i] - w_t^w[j]) L_{t-1}[i,j] + w_t^w[i] p_{t-1}[j]$$

Problems

- Memory addressing is purely hypothetical
- May not learn true “programs”
- Thoughts for future work
  - Learn from restricted class of automata (e.g., PDA)
  - Make intermediate execution results Markov blanket
**Theorem 1.** Let RNN have vanilla transition with the linear activation function, and let the RNN state at the last step $h_{i-1}$ be fixed. For a particular data point, if the memory attention satisfies $\sum_{j=N+1}^{N+M} \alpha_{i,j} \leq \sum_{j=1}^{N} \alpha_{i,j}$, then memory expansion yields a lower expected mean squared difference in $h_i$ than RNN state expansion. That is,

$$\mathbb{E} \left[ \|h_i^{(m)} - h_i\|^2 \right] \leq \mathbb{E} \left[ \|h_i^{(s)} - h_i\|^2 \right]$$

(9)

where $h_i^{(m)}$ refers to the hidden states if the memory is expanded. $h_i^{(s)}$ refers to the original dimensions of the RNN states, if we expand the size of RNN states themselves. Here, we compute the expectation by assuming weights and hidden states are iid from a zero-mean Gaussian distribution (with variance $\sigma^2$).

Conclusion & Take-Home Msg

• Sequence-to-sequence training
  - Training: Step-by-step supervised learning
  - Inference: Greedy, Beam search, sampling

• Attention
  - Adaptive weighted sum of source information
  - Alignment in MT
  - Aggregated information
Suggested Reading

- Automata theory


More References