Neural sequence-to-sequence models for machine translation

CMSC 470
Marine Carpuat
Machine Translation

• Translation system
  • Input: source sentence F
  • Output: target sentence E
  • Can be viewed as a function

\[ \hat{E} = \text{mt}(F) \]

• Modern machine translation systems

\[ \hat{E} = \arg\max_E P(E \mid F; \theta) \]

• 3 problems

  • Modeling
    • how to define P(.)?

  • Training/Learning
    • how to estimate parameters from parallel corpora?

  • Search
    • How to solve argmax efficiently?
The Encoder models the input/source sentence $F=(f_1, \ldots, f_{|F|})$.

The decoder hidden state is initialized with the last hidden state of the encoder.

The Decoder models the output/target sentence $E=(e_1, \ldots, e_{|E|})$. 

The diagram shows the interaction between the encoder and decoder models.
Neural Machine Translation

- Neural language models review

- Sequence to sequence models for MT
  - Encoder-Decoder
  - Sampling and search (greedy vs beam search)
  - How to train?
  - Model variants and practical tricks
  - Attention mechanism
Training

• Same as for RNN language modeling
  • Intuition: a good model assigns high probability to training examples

• Loss function
  • Negative log-likelihood of training data
    • Also called cross-entropy loss
  • Total loss for one example (sentence pair) = sum of loss at each time step (word)

• Backpropagation
  • Gradient of loss at time step $t$ is propagated through the network all the way back to $1^{st}$ time step
Aside: why don’t we use BLEU as training loss?

N-gram overlap between machine translation output and reference translation

Compute precision for n-grams of size 1 to 4

Add brevity penalty (for too short translations)

$$\text{BLEU} = \min \left( 1, \frac{\text{output-length}}{\text{reference-length}} \right) \left( \prod_{i=1}^{4} \text{precision}_i \right)^{\frac{1}{4}}$$

Typically computed over the entire corpus, not single sentences
Training in practice: online

Algorithm 1 A fully online training algorithm

1: procedure ONLINE
2:  for several epochs of training do
3:    for each training example in the data do
4:      Calculate gradients of the loss
5:    Update the parameters according to this gradient
6:  end for
7: end for
8: end procedure
Training in practice: batch

Algorithm 2 A batch learning algorithm

1: procedure BATCH
2: for several epochs of training do
3: for each training example in the data do
4: Calculate and accumulate gradients of the loss
5: end for
6: Update the parameters according to the accumulated gradient
7: end for
8: end procedure
Training in practice: minibatch

• Compromise between online and batch

• Computational advantages
  • Can leverage vector processing instructions in modern hardware
  • By processing multiple examples simultaneously

Operations w/o Minibatching
\[
\begin{align*}
\tanh(Wx_1 + b) & \quad \tanh(Wx_2 + b) & \quad \tanh(Wx_3 + b)
\end{align*}
\]

Operations with Minibatching
\[
\begin{align*}
x_1 x_2 x_3 & \quad \text{concat} \quad \text{broadcast} \quad b \\
\tanh(Wx + b)
\end{align*}
\]
Problem with minibatches: examples have varying length

• 3 tricks (same as for language modeling)
  • Padding
    • Add </s> symbol to make all sentences same length
  • Masking
    • Multiply loss function calculated over padded symbols by zero
  • + sort sentences by length
Neural Machine Translation

- Neural language models review

- Sequence to sequence models for MT
  - Encoder-Decoder
  - Sampling and search (greedy vs beam search)
  - How to train?
  - Model variants and practical tricks
  - Attention mechanism
Other encoder structures: RNN variants

• LSTMs
  • Aim to address vanishing/exploding gradient issue

• Stacked RNNs

• …
Other encoder structures: Bidirectional encoder

\[
\vec{h}_t^{(f)} = \begin{cases} 
\overrightarrow{\text{RNN}}^{(f)}(m_t^{(f)}, \vec{h}_{t+1}^{(f)}) & t \geq 1, \\
0 & \text{otherwise.}
\end{cases}
\]

\[
\vec{h}_t^{(f)} = \begin{cases} 
\overleftarrow{\text{RNN}}^{(f)}(m_t^{(f)}, \vec{h}_{t+1}^{(f)}) & t \leq |F|, \\
0 & \text{otherwise.}
\end{cases}
\]

\[
h_0^{(e)} = \tanh(W_{\vec{f}e} \vec{h}_{|F|} + W_{\vec{f}e} \vec{h}_1 + b_e)
\]

Motivation:
- Help bootstrap learning
- By shortening length of dependencies

Motivation:
- Take 2 hidden vectors from source encoder
- Combine them into a vector of size required by decoder
A few more tricks: ensembling

- Combine predictions from multiple models

Methods
- Linear or log-linear interpolation
- Parameter averaging
Tricks: addressing length bias

• Default models tend to generate short sentences
• Solutions:
  • Prior probability on sentence length

\[
\hat{E} = \arg\max_E \log P(|E| \mid |F|) + \log P(E \mid F).
\]

• Normalize by sentence length

\[
\hat{E} = \arg\max_E \log P(E \mid F)/|E|.
\]
Neural Machine Translation

• Neural language models review

• Sequence to sequence models for MT
  • Encoder-Decoder
  • Sampling and search (greedy vs beam search)
  • How to train?
  • Model variants and practical tricks
  • Attention mechanism