Machine Translation 2
Neural Machine Translation

CS 287
Review: Simple One-to-One Model

1. Language Model; words depend on previous word

\[ p(y) = \prod_{i=1}^{n} p(y_i | y_{i-1}) \]

2. Translation Model; source word depends on current position

\[ p(x|y) = \prod_{i=1}^{n} p(x_i | x_{i-1}) \]
Review: Alignment

- a; alignment mapping each target word to a source word
- Assuming one-to-one

Mary golpeo la bruja verde

Mary slapped the green witch

Mary golpeo la bruja verde

Mary X
slapped X
the X
green X
witch X
Review: Using Alignments

\[ p(y|x) \propto \sum_{a} p(y) p(a|y) p(x|a, y) \]

With alignment,

\[ p(x|a, y) = \prod_{i=1}^{n} p(x_{a_i}|y_i) \]

Sum-over-alignment approximated with a max-over-alignment,

\[ \arg \max_{j, w_{i:n}^t} \prod_{i=1}^{n} p(x_{a_i}|y_i = w_i^t) p(y_i = w_i^t|y_{i-1} = w_{i-1}^t) p(a_i = c_i|a_{i-1} = c_{i-1}, i) \]
Quiz: CRF

Note: Nothing in our definition of CRFs relied on $y_i$ to align with $x_i$ (conditioned on full sequence).

For this quiz, imagine we have an input sequence $x$, and we want to find the optimal output sequence $y$ but we do not fix $n < N$. For instance finding the best word segmentation of an unsegmented input $x$.

- How would you find
  $ \arg \max_{n,c_{1:n}} f(x, c_{1,n}) $?

- How would you train $f(x, c_{1,n}; \theta)$?
Preliminaries

- How alignment is used in SMT.
- Language Model RNNs
- Bidirectional RNNs
- Stacked RNNs
Bit-Set Beam Search

[Describe on board]
Stacked RNN
Recent popularization of RNNs has been based on language modeling (Mikolov, 2012).

In particular RNNs allow for non-Markovian models:

\[ p(w_i|w_1, \ldots, w_{i-1}; \theta) = O(s_i) \]

Compare this to the feed-forward windowed approach:

\[ p(w_i|w_{i-n+1}, \ldots, w_{i-1}; \theta) = O(s_i) \]
RNN as Transducer

Can reuse hidden state each time

\[
p(w_i|w_1, \ldots, w_{i-1}; \theta) = O(s_i) = O(R(s_{i-1}, x_i))
\]

\[
p(w_{i+1}|w_1, \ldots, w_i; \theta) = O(R(s_i, x_{i+1}))
\]
Bidirectional Models

- For all $i \in \{1, \ldots, n\}$
  \[
  s_i^f = R^f(s_{i-1}, x_i)
  \]

- For all $i \in \{1, \ldots, n\}$
  \[
  s_i^b = R^b(s_{i+1}, x_i)
  \]

- For all $i \in \{1, \ldots, n\}$
  \[
  \hat{y}_i = O([s_i^b, s_i^f]) = [s_i^b, s_i^f]W + b
  \]
Today’s Lecture

1. Sequence-to-Sequence Model (Sutskever et al 2014)

2. Attention-Based Model (Bahdanau et al 2014)
Neural Machine Translation

- \( x = [w_1^s, w_2^s, w_3^s, \ldots, w_n^s] \)
- \( y = [w_1^t, w_2^t, w_3^t, \ldots, w_n^t] \)

\[
p(w^t|w^s) = \prod_{i=1}^{n} p(w_i^t|w^s, \theta)
\]

Intuition:

1. Start with bag-of-source words
2. Pick a source word, translate it to a target word that “fits”
3. Update source bag representation.
Encoder-Decoder Idea

Compute a single vector $s^t$ encoding the source.

$p(w_i^t|w_1^t, \ldots, w_{i-1}^t, s) = \hat{y}_{w_i}$

Intuition:

- Start with encoded source
- Use encoded vector to translate to a target word that “fits”
- Update source representation
Encoder

- How do you get a source encoding? LSTM (or bi-LSTM)
Decoder

- How do you use this source encoding? LSTM

![Diagram of LSTM decoder with words "the", "red", and "dog" as inputs and symbols $\hat{y}_1$, $\hat{y}_2$, $\hat{y}_3$, $s^t_1$, $s^t_2$, $s^t_3$, $\hat{x}_1$, $\hat{x}_2$, $\hat{x}_3$, and $\langle s \rangle$ as outputs.]
Sequence-to-Sequence

The red dog

\[
\begin{align*}
\mathbf{s}_1^s &\rightarrow \mathbf{s}_2^s &\rightarrow \mathbf{s}_3^s &\rightarrow \mathbf{s}_1^t \\
\mathbf{x}_1 &\rightarrow \mathbf{x}_2 &\rightarrow \mathbf{x}_3 &\rightarrow \mathbf{x}_1 \\
\text{the} &\rightarrow \text{red} &\rightarrow \text{dog} &\rightarrow \langle s \rangle \\
\hat{\mathbf{y}}_1 &\rightarrow \hat{\mathbf{y}}_2 &\rightarrow \hat{\mathbf{y}}_3 \\
\text{the} &\rightarrow \text{red} &\rightarrow \text{dog} &\rightarrow \langle s \rangle
\end{align*}
\]
Stacked Sequence-to-Sequence

\[ s_1^s \rightarrow s_2^s \rightarrow s_3^s \rightarrow s_1^t \rightarrow s_2^t \rightarrow s_3^t \]

\[ x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow \hat{x}_1 \rightarrow \hat{x}_2 \rightarrow \hat{x}_3 \]

\[ y_1 \rightarrow y_2 \rightarrow y_3 \]

\[ \text{the} \]

\[ \text{red} \]

\[ \text{dog} \]
While the LSTM is capable of solving problems with long term dependencies, we discovered that the LSTM learns much better when the source sentences are reversed (the target sentences are not reversed). By doing so, the LSTMs test perplexity dropped from 5.8 to 4.7, and the test BLEU scores of its decoded translations increased from 25.9 to 30.6.
We found that the LSTM models are fairly easy to train. We used deep LSTMs with 4 layers, with 1000 cells at each layer and 1000 dimensional word embeddings, with an input vocabulary of 160,000 and an output vocabulary of 80,000. We found deep LSTMs to significantly outperform shallow LSTMs, where each additional layer reduced perplexity by nearly 10%, possibly due to their much larger hidden state. We used a naive softmax over 80,000 words at each output. The resulting LSTM has 380M parameters of which 64M are pure recurrent connections (32M for the encoder LSTM and 32M for the decoder LSTM). The complete training details are given below:
We search for the most likely translation using a simple left-to-right beam search decoder which maintains a small number $B$ of partial hypotheses, where a partial hypothesis is a prefix of some translation. At each timestep we extend each partial hypothesis in the beam with every possible word in the vocabulary. This greatly increases the number of the hypotheses so we discard all but the $B$ most likely hypotheses according to the model’s log probability. As soon as the EOS symbol is appended to a hypothesis, it is removed from the beam and is added to the set of complete hypotheses. While this decoder is approximate, it is simple to implement. Interestingly, our system performs well even with a beam size of 1, and a beam of size 2 provides most of the benefits of beam search (Table 1).
<table>
<thead>
<tr>
<th>Method</th>
<th>test BLEU score (ntst14)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline System [29]</td>
<td>33.30</td>
</tr>
<tr>
<td>Cho et al. [5]</td>
<td>34.54</td>
</tr>
<tr>
<td>State of the art [9]</td>
<td>37.0</td>
</tr>
<tr>
<td>Rescoring the baseline 1000-best with a single forward LSTM</td>
<td>35.61</td>
</tr>
<tr>
<td>Rescoring the baseline 1000-best with a single reversed LSTM</td>
<td>35.85</td>
</tr>
<tr>
<td>Rescoring the baseline 1000-best with an ensemble of 5 reversed LSTMs</td>
<td>36.5</td>
</tr>
<tr>
<td>Oracle Rescoring of the Baseline 1000-best lists</td>
<td>~45</td>
</tr>
</tbody>
</table>
Contents

Sequence-To-Sequence

Attention-Based
Sequence-To-Sequence

- True Encoder-Decoder stores full sentence in single vector.
- In practice we have access to the original at decoding time.
Alignment

1. Start with bag-of-source words
2. Pick a **aligned** source word, translate it to a target word that “fits”
3. Update source bag representation.

Mary golpeo la bruja verde

Mary X
slapped X
the X
green X
witch X
Hard Alignments

- Have source vectors $s^t_j$
- Could find greedy alignment at each step.
- Idea: learn scoring MLP, and use best aligned vector.
- At generation position $i$, For all source positions $j$,

$$z_j = \tanh([s^t_i, s^s_j]W + b)$$

$$j = \arg\max_j z_j$$

$$w^t_{i+1} = \arg\max_w O(w, s^t_{i+1}, s^s_j)$$

What is the issue here?
Attention: Soft Alignments

At generation position $i$, For all source positions $j$,

$$z_j = \tanh([s_t^i, s_j^s]W + b)$$

$$\alpha = \text{softmax}(z)$$

$$c = \sum_j \alpha_j s_j^s$$

$$w_{i+1}^t = \arg \max_w O(w, s_{i+1}^t, c)$$

$\alpha_j$ is the “attention” paid to source position $j$
La destruction de l'équipement signifie que la Syrie ne peut plus produire de nouvelles armes chimiques.
While the LSTM is capable of solving problems with long term dependencies, we discovered that the LSTM learns much better when the source sentences are reversed (the target sentences are not reversed). By doing so, the LSTMs test perplexity dropped from 5.8 to 4.7, and the test BLEU scores of its decoded translations increased from 25.9 to 30.6.
We use a minibatch stochastic gradient descent (SGD) algorithm together with Adadelta (Zeiler, 2012) to train each model. Each SGD update direction is computed using a minibatch of 80 sentences. We trained each model for approximately 5 days. Once a model is trained, we use a beam search to find a translation that approximately maximizes the conditional probability (see, e.g., Graves, 2012; Boulanger-Lewandowski et al., 2013). Sutskever et al. (2014) used this approach to generate translations from their neural machine translation model.
WMT 14 contains the following English-French parallel corpora: Europarl (61M words), news commentary (5.5M), UN (421M) and two crawled corpora of 90M and 272.5M words respectively, totaling 850M words. Following the procedure described in Cho et al. (2014a), we reduce the size of the combined corpus to have 348M words using the data selection method by Axelrod et al. (2011). We do not use any monolingual data other than the mentioned parallel corpora, although it may be possible to use a much larger monolingual corpus to pretrain an encoder.