Interactive Machine Translation

Dan Klein, John DeNero
UC Berkeley
System Demonstrations

(Demo)
Mixed-Initiative Experimental Studies

(Spence Green's Dissertation Slides)
Prefix-Constrained Decoding
Prefix Decoding

A user enters a prefix of the translation; the MT system predicts the rest.

*Yemeni media report that there is traffic chaos in the capital.*

Once the user has typed:
*Jemenitische Medien berichten von einem Verkehrschaos*

The system suggests:
*in der Hauptstadt.*

Suggestion is useful when:
• Sentence is completed in a way that a translator accepts,
• The next word suggestion is acceptable,
• Sentence is completed in a way that requires minimal post-editing.
Phrase-Based Prefix-Constrained Decoding

Early work [Barrachina et al. 2008; Ortiz-Martínez et al. 2009]: Standard phrase-based beam search, but discard hypotheses that don't match the prefix.

Better version [Wuebker et al. 2016]: While aligning the prefix to the source, use one beam per target cardinality. While generating the suffix of the translation, use one beam per source cardinality.

Also added:

• Different translation model weights for phrases in the prefix and suffix (lexical features are more relevant for alignment).
• Phrases extracted from the source and prefix to ensure coverage.
Neural Prefix-Constrained Decoding

State-of-the-art neural MT model from 2015 [Luong et al., 2015]:
• 4-layer stacked LSTM with attention.
• Embedding size & hidden unit size of 1000.
• 50-sentence mini-batches of sentences with length 50 or less; trained with SGD.
• Before layer normalization, residual connections, back translation, knowledge distillation, Transformer architecture, subwords, or label smoothing.
• Beam size of 12 for the suffix; beam size of 1 for the prefix (the constrained word).
## Prefix Decoding: Phrase-Based vs Neural

<table>
<thead>
<tr>
<th>En-De</th>
<th>autodesk</th>
<th></th>
<th>newstest2015</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>BLEU</strong></td>
<td><strong>Next word accuracy</strong></td>
<td><strong>BLEU</strong></td>
<td><strong>Next word accuracy</strong></td>
</tr>
<tr>
<td>Phrasal baseline</td>
<td>44.5</td>
<td>37.8</td>
<td>22.4</td>
<td>28.5</td>
</tr>
<tr>
<td>Phrasal improved</td>
<td>44.5</td>
<td>46.0</td>
<td>22.4</td>
<td>41.2</td>
</tr>
<tr>
<td>NMT</td>
<td>40.6</td>
<td>52.3</td>
<td>23.2</td>
<td>50.4</td>
</tr>
<tr>
<td>NMT ensemble</td>
<td>44.3</td>
<td>54.9</td>
<td>26.3</td>
<td>53.0</td>
</tr>
</tbody>
</table>

Wuebker et al., 2016, "Models and Inference for Prefix-Constrained Machine Translation"
Online Adaptation
Online Fine-Tuning for Model Personalization

After sentence $i$ is translated, take a stochastic gradient descent step with batch size 1 on $(x_i, y_i)$ [Turchi et al., 2017].

Evaluation via simulated post-editing [Hardt and Elming, 2010]:
• Adaptation is performed incrementally on the test set.
• Translate $x_i$ using model $\theta_{i-1}$ and compare it to reference $y_i$.
• Then, estimate $\theta_i$ from $(x_i, y_i)$.

E.g., Autodesk corpus results using a small Transformer:
• Unadapted baseline: 40.3% BLEU
• Online adaptation: 47.0% BLEU

Recall of observed words goes up, but unobserved words goes down [Simianer et al., 2019]:
• $R1$ — % of words appearing for the second time in any reference that also appear in the corresponding hypothesis: 44.9% -> 55.0%
• $R0$ — % of words appearing for the first time in any reference that also appear in the corresponding hypothesis: 39.3% -> 35.8%

Turchi et al., 2017, "Continuous learning from human post-edits for neural machine translation."
Simianer et al., 2019, "Measuring Immediate Adaptation Performance for Neural Machine Translation"
Space-Efficient Model Adaptation

Inference for "personalized" (user-adapted) models:
• Load User X’s model from cache or persistent storage
• Apply model parameters to computation graph
• Perform inference

Example production constraint: latency budget of 300ms ⇒ maximum of ~10M parameters for a personalized model

Full (small transformer) model in 2019: ~36M parameters

Solution:
• Store models as offsets from baseline model: \( W = W_b + W_u \)
• Select sparse parameter subset \( W_u \)
### Table: Model Performance

<table>
<thead>
<tr>
<th></th>
<th>batch adaptation</th>
<th>online adaptation</th>
<th># params</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline</td>
<td>33.7</td>
<td></td>
<td>36.2M</td>
</tr>
<tr>
<td>full model</td>
<td><strong>41.7</strong></td>
<td><strong>39.0</strong></td>
<td>25.8M</td>
</tr>
<tr>
<td>outer layers</td>
<td>38.6</td>
<td>37.9</td>
<td><strong>2.2M</strong></td>
</tr>
<tr>
<td>inner layers</td>
<td>38.8</td>
<td>37.8</td>
<td>2.7M</td>
</tr>
<tr>
<td>enc. embeddings</td>
<td>36.3</td>
<td>35.7</td>
<td>5.0M</td>
</tr>
<tr>
<td>dec. embeddings</td>
<td>34.2</td>
<td>34.3</td>
<td>5.5M</td>
</tr>
<tr>
<td>output proj.</td>
<td>38.7</td>
<td>37.5</td>
<td>5.5M</td>
</tr>
</tbody>
</table>

### Diagram: Encoder and Decoder

- **Encoder**
  - Embedding lookup
  - Enc. embeddings (10.3M (5.0M))
  - 4x self-attention layers (526K)
  - Outer layers (10.3M (5.5M))

- **Decoder**
  - Embedding lookup
  - Dec. embeddings (10.3M (5.5M))
  - 4x self-attention layers (788K)
Group Lasso Regularization for Sparse Adaptation

Simultaneous regularization and tensor selection

Regularize offsets Wu, define each tensor as one group g for L1/L2 regularization

\[ R_{\ell_1,2}(W_u) = \sum_{g \in W_u} \sqrt{|g|} \|g\|_2 \]

Total loss: \[ \mathcal{L} = \mathcal{L}_{\text{seq}}(W_b + W_u) + \lambda R_{\ell_1,2}(W_u) \]

Cut off all tensors g with: \[ \frac{1}{|g|} \sum_{w \in g} |w| < \theta \]

Define a group for each hidden layer and each embedding column

<table>
<thead>
<tr>
<th></th>
<th>en&gt;fr</th>
<th>fr&gt;en</th>
<th>en&gt;ru</th>
<th>ru&gt;en</th>
<th>en&gt;zh</th>
<th>zh&gt;en</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>28.8</td>
<td>35.8</td>
<td>10.7</td>
<td>29.7</td>
<td>19.9</td>
<td>18.9</td>
</tr>
<tr>
<td>Full Adaptation</td>
<td>36.6</td>
<td>49.6</td>
<td>21.0</td>
<td>42.1</td>
<td>40.6</td>
<td>46.6</td>
</tr>
<tr>
<td>Sparse Adapt. (# params)</td>
<td>36.2 (16.5%)</td>
<td>49.2 (15.9%)</td>
<td>21.2 (16.1%)</td>
<td>42.2 (15.8%)</td>
<td>42.0 (15.6%)</td>
<td>46.5 (15.2%)</td>
</tr>
</tbody>
</table>

Wuebker et al., 2018, "Compact Personalized Models for Neural Machine Translation"
Bottleneck Adapter Modules

Adapter modules:

- Add offsets to activations by a combination of new adapter layers and residual connections.
- Initialize adapter layer weights near zero.
- During adaptation, freeze all model parameters except the adapter layers.

Houlsby et al., 2019, "Parameter-Efficient Transfer Learning for NLP"
Tag Projection
Simple terminology-constrained inference:
• Users often specify termbases, which act as restrictions on the target translation.
• When attention focuses on a source term, add the corresponding target term to the translation hypothesis.

Tag projection:
• Strip markup tags before translation
• Project tags to final target sentence using word alignments

From Wikipedia: <span><b>Translation</b> is the communication of the <a1>meaning</a1> of a <a2>source-language</a2> text by means of an <a3>equivalent</a3> <a4>target-language</a4> text.<sup><a5>[1]</a5></sup></span>

Google Translate: <span><b>Übersetzung</b> ist die Übermittlung der <a1>Bedeutung</a1> eines <a2>quellsprachlichen</a2> Textes mittels eines <a3>äquivalenten</a3> <a4>zielsprachlichen</a4> Textes.<sup><a5>[1]</a5></sup></span>
Alignment by Attention in a Transformer

**Train** alignment attention to predict the next word.

**Inference:** find attention activations that maximize the likelihood of the observed next word.

**Retrain:** toward a self-supervised loss that maximizes the likelihood of inferred word alignments.

**Ensemble:** inference under a bidirectional objective.

**Bias** alignment configurations to have adjacent source words aligned to adjacent target words.

---

**Zenkel et al., 2020. "End-to-End Neural Word Alignment Outperforms GIZA++"**
Some Open Questions
**Insertion Transformer**

How can a translation model make suggestions about mid-sentence edits?

\[ p(c, \ell| x, \hat{y}_t) = \text{InsertionTransformer}(x, \hat{y}_t) \]

- \( c \in C \): a word to be inserted
- \( \ell \in [0, |\hat{y}_t|] \): an insertion location
- \( x \): Input sequence
- \( \hat{y}_t \): A sequence of output words inserted so far

---

<table>
<thead>
<tr>
<th>( t )</th>
<th>Canvas</th>
<th>Insertion</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[]</td>
<td>(ate, 0)</td>
</tr>
<tr>
<td>1</td>
<td>[ate]</td>
<td>(together, 1)</td>
</tr>
<tr>
<td>2</td>
<td>[ate, together]</td>
<td>(friends, 0)</td>
</tr>
<tr>
<td>3</td>
<td>[friends, ate, together]</td>
<td>(three, 0)</td>
</tr>
<tr>
<td>4</td>
<td>[three, friends, ate, together]</td>
<td>(lunch, 3)</td>
</tr>
<tr>
<td>5</td>
<td>[three, friends, ate, lunch, together]</td>
<td>(EOS, 5)</td>
</tr>
</tbody>
</table>

Stern et al., 2019. "Insertion Transformer: Flexible Sequence Generation via Insertion Operations"
Reformer

Should document context be incorporated through learning or inference? With some optimizations, a 64k-length sequence can be encoded on a GPU.

Kitaev et al., 2020. "Reformer: The Efficient Transformer"