Neural Lattice-to-Sequence Models for Uncertain Inputs

Matthias Sperber\textsuperscript{1}, Graham Neubig\textsuperscript{2}, Jan Niehues\textsuperscript{1} and Alex Waibel\textsuperscript{1}

\textsuperscript{1}Karlsruhe Institute of Technology, Germany

\textsuperscript{2}Carnegie Mellon University, USA

speaker: Alex Hung
Outline

1. Introduction
2. Word-Lattice based RNN Encoders
3. Experiments
Previous research on traditional phrase-based or tree-based statistical machine translation have used word lattices (e.g. Figure 1) as an effective tool to pass on uncertainties.

Recently, neural sequence-to-sequence models have often been preferred over the traditional methods for their strong empirical results and appealing end-to-end modeling.
Abstract

The input to a neural sequence-to-sequence model is often determined by an up-stream system, e.g. a word segmenter, part of speech tagger, or speech recognizer. These up-stream models are potentially error-prone. Representing inputs through word lattices allows making this uncertainty explicit by capturing alternative sequences and their posterior probabilities in a compact form. In this work, we extend the TreeLSTM (Tai et al., 2015) into a LatticeLSTM that is able to consume word lattices, and can be used as encoder in an attentional encoder-decoder model. We integrate lattice posterior scores into this architecture by extending the TreeLSTM’s child-sum and forget gates and introducing a bias term into the attention mechanism.

Figure 1: A lattice with 3 possible paths and posterior scores. Translating the whole lattice potentially allows for recovering from errors in its 1-best hypothesis.
Word-Lattice based RNN Encoders


At time step $t$, we first identify all edges pointing to $v_t$, each of which covers different input words with preceding hidden states.

for $k$th edge:

- $x_t^{(k)}$ = input word vector
- $h_{pre}^{(k)}$ = the preceding hidden state
We combine all possible word embeddings \( \{x_t^*\} \) into a compressed \( x_t \).

Similarly, the hidden state vectors \( \{h_{\text{pre}}^*\} \) of preceding time steps are also compressed into \( h_{\text{pre}} \).

(a) Standard GRU

(b) Shallow Word-Lattice Based GRU
Gated Recurrent Unit

σ: sigmoid function (gate)
z_t: update gate
r_t: reset gate

\[ r_t = \sigma \left( W^{(r)} x_t + U^{(r)} h_{t-1} \right) \]  \hspace{1cm} (1)
\[ z_t = \sigma \left( W^{(z)} x_t + U^{(z)} h_{t-1} \right) \]  \hspace{1cm} (2)
\[ \tilde{h}_t = \phi \left( W x_t + U (r_t \odot h_{t-1}) \right) \]  \hspace{1cm} (3)
\[ h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t \]  \hspace{1cm} (4)

- Note: sequential RNNs are conditioned on only one predecessor state (h_{t-1}).

**SWL-GRU**

\[ x_t = g(x_t^{(1)}, x_t^{(2)}, ...) \]  \hspace{1cm} (5)
\[ h_{pre} = g(h_{pre}^{(1)}, h_{pre}^{(2)}, ...) \]  \hspace{1cm} (6)
\[ r_t = \sigma \left( W^{(r)} x_t + U^{(r)} h_{pre} \right) \]  \hspace{1cm} (7)
\[ z_t = \sigma \left( W^{(z)} x_t + U^{(z)} h_{pre} \right) \]  \hspace{1cm} (8)
\[ \tilde{h}_t = \phi \left( W x_t + U (r_t \odot h_{pre}) \right) \]  \hspace{1cm} (9)
\[ h_t = z_t \odot h_{pre} + (1 - z_t) \odot \tilde{h}_t \]  \hspace{1cm} (10)
Deep Word-Lattice Based GRU

- "Shallow": 先合併
- "Deep": 後合併

\[
\begin{align*}
    r_t^{(k)} &= \sigma \left( W^{(r)} x_t^{(k)} + U^{(r)} h_{\text{pre}}^{(k)} \right) \quad (11) \\
    z_t^{(k)} &= \sigma \left( W^{(z)} x_t^{(k)} + U^{(z)} h_{\text{pre}}^{(k)} \right) \quad (12) \\
    \tilde{h}_t^{(k)} &= \phi \left( W x_t^{(k)} + U (r_t \odot h_{\text{pre}}^{(k)}) \right) \quad (13) \\
    h_t^{(k)} &= z_t \odot h_{\text{pre}}^{(k)} + (1 - z_t) \odot \tilde{h}_t^{(k)} \quad (14) \\
    h_t &= g(h_t^{(1)}, h_t^{(2)}, \ldots) \quad (15)
\end{align*}
\]
Composition function $g(\cdot)$

Types:

1. (Max) Pooling Operation Function
2. Gating Operation Function

\begin{align}
\alpha_t^{(k)} &= \frac{\sigma(U(g)h_t^{(k)} + b(g))}{\sum_i \sigma(U(g)h_t^{(i)} + b(g))} \\
h_t &= \sum_j \alpha_t^{(j)} h_t^{(j)}
\end{align} 

(16)  
(17)
The input to a neural sequence-to-sequence model is often determined by an up-stream system. A word segmenter, part of speech tagger, or speech recognizer.

In this paper, we propose a lattice-to-sequence model that accounts for the uncertainty of the paths.
Lattices and the TreeLSTM

- the TreeLSTM composes multiple child states into a parent state

\[
\tilde{h}_i = \sum_{k \in C(i)} h_k
\]  

(18)

Figure 2: Network structure of a bidirectional lattice encoder with one layer.
when integrating scores into the lat2seq framework, it is desirable to maintain flexibility over how strongly they should impact the model.

\[ \tilde{h}_i = \sum_{k \in C(i)} w_k h_k \]  \hspace{1cm} (19)

Let \( w_{f,i}, w_{m,i}, w_{b,i} \) denote forward-normalized, marginal, and backward-normalized scores for node \( i \) respectively

\[ w_{m,i} = \sum_{k \in C(i)} w_{m,k} w_{f,i} \] \hspace{1cm} (20)

\[ w_{b,i} = \frac{w_{m,i}}{\sum_{k \in C'(i)} w_{m,k}} \] \hspace{1cm} (21)

\( C'(i) \) denotes the successors of node \( i \).
Figure 3: Lattice with forward-normalized, marginal, and backward-normalized scores.
Lattice Score Normalization

- forward direction:

\[ w_i = \frac{w_{f,i}^S}{Z_i} \]  \hspace{2cm} (22)

\[ Z_i = \sum_{k \in C'(i)} w_{f,k} \]  \hspace{2cm} (23)

- S: peakiness coefficient
  - Setting \( S = 0 \) amounts to ignoring the scores by flattening their distribution
Integration Approaches

1. using lattice scores as weights when composing the hidden state

\[ \tilde{h}_i = \sum_{k \in C(i)/C'(i)} \frac{w_{b/f,i}^{S_h}}{Z_{h,k}} h_k \] (24)

2. biases the forget gate \( f_{i,k} \) for each predecessor cell state
   - predecessors with high lattice score are more likely to pass through the forget gate

\[ f_{i,k} = \sigma \left( W_f x_i + U_f h_k + \left[ \log \frac{w_{b/f,k}^{S_f}}{Z_{f,k}} \right] + b_f \right) \] (25)
<table>
<thead>
<tr>
<th></th>
<th>Sequential LSTM</th>
<th>TreeLSTM</th>
<th>Proposed LatticeLSTM</th>
</tr>
</thead>
<tbody>
<tr>
<td>recurrence</td>
<td>$\tilde{h}<em>i = h</em>{i-1}$</td>
<td>$\tilde{h}<em>i = \sum</em>{k \in C(i)} h_k$</td>
<td>$\tilde{h}<em>i = \sum</em>{k \in C(i)} \frac{w^S_{h_k}}{Z_{h,k}} h_k$ (5)</td>
</tr>
<tr>
<td>forget-gate</td>
<td>$f_i = \sigma (W_f x_i + U_f \tilde{h}_i + b_f)$</td>
<td>$f_{ik} = \sigma (W_f x_i + U_f h_k + b_f)$</td>
<td>$f_{ik} = \sigma (W_f x_i + U_f h_k + [\ln w_{b,f,k} S_f - Z_{f,k}] + b_f)$ (6)</td>
</tr>
<tr>
<td>input-gate</td>
<td>$i_i = \sigma (W_{in} x_i + U_{in} \tilde{h}<em>i + b</em>{in})$</td>
<td>as sequential</td>
<td>as sequential</td>
</tr>
<tr>
<td>output gate</td>
<td>$o_i = \sigma (W_{o} x_i + U_{o} \tilde{h}<em>i + b</em>{o})$</td>
<td>as sequential</td>
<td>as sequential</td>
</tr>
<tr>
<td>update</td>
<td>$u_i = \tanh (W_u x_i + U_u \tilde{h}_i + b_u)$</td>
<td>as sequential</td>
<td>as sequential</td>
</tr>
<tr>
<td>new cell</td>
<td>$c_i = i_i \odot u_i + f_i \odot c_{i-1}$</td>
<td>$c_i = i_i \odot u_i + \sum_{k \in C(i)} f_{ik} \odot c_k$</td>
<td>as TreeLSTM</td>
</tr>
<tr>
<td>new hidden</td>
<td>$h_i = o_i \odot \tanh(c_i)$</td>
<td>as sequential</td>
<td>as sequential</td>
</tr>
<tr>
<td>attention</td>
<td>$\alpha_{ij} \propto \exp (s (\cdot))$</td>
<td>$\alpha_{ij} \propto \exp [s (\cdot) + S_a \ln w_{m,i}]$ (7)</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Formulas for sequential and TreeLSTM encoders according to Tai et al. (2015), the proposed LatticeLSTM encoder, and conventional vs. proposed integration into the attention mechanism (bottom row). Inputs $x_j$ are word embeddings or hidden states of a lower layer. $W.$ and $U.$ denote parameter matrices, $b.$ bias terms, other terms are described in the text.
In the third and final method, we bias the attentional weights (BATT) to put more focus on lattice nodes with high lattice scores. This can potentially mitigate the problem of having multiple contradicting lattice nodes that may confuse the attentional decoder. BATT is computed by introducing a bias term to the attention as in (7). Attentional weights are scalars, so here the peakiness $S_a$ is also a scalar. We drop the normalization term, relying instead on the softmax normalization. Both BFG and BATT use the logarithm of lattice scores so that values will still be in the probability domain after the softmax or sigmoid is computed.

4.3 Pre-training

Finally, to reduce the computational burden, we perform a two-step training process where the model is first pre-trained on sequential data, then fine-tuned on lattice data. The pre-training, like standard training for neural machine translation (NMT), allows for efficient training using mini-batches, and also allows for training on standard text corpora for which we might not have lattices available. The fine-tuning is then performed on parallel data with lattices on the source side. This is much slower than the pre-training because the network structure changes from sentence to sentence, preventing us from using efficient mini-batch calculations. However, fine-tuning for only a small number of iterations is generally sufficient, as the model is already relatively accurate in the first place. In practice we found it important to use minibatches when fine-tuning, accumulating gradients over several examples before performing parameter updates. This provided negligible speedups but greatly improved optimization stability.

At test time, the model is able to translate both sequential and lattice inputs and can therefore be used even in cases where no lattices are available, at potentially diminished accuracy.

5 Experiments

5.1 Setting

We conduct experiments on the Fisher and Callhome Spanish–English Speech Translation Corpus.

<table>
<thead>
<tr>
<th></th>
<th>1-best WER</th>
<th>oracle WER</th>
<th># sent.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fisher/Dev</td>
<td>41.3</td>
<td>19.3</td>
<td>3,979</td>
</tr>
<tr>
<td>Fisher/Dev2</td>
<td>40.0</td>
<td>19.4</td>
<td>3,961</td>
</tr>
<tr>
<td>Fisher/Test</td>
<td>36.5</td>
<td>16.1</td>
<td>3,641</td>
</tr>
<tr>
<td>Callhome/Devtest</td>
<td>64.7</td>
<td>36.4</td>
<td>3,966</td>
</tr>
<tr>
<td>Callhome/Evltest</td>
<td>65.3</td>
<td>37.9</td>
<td>1,829</td>
</tr>
</tbody>
</table>

Table 2: Development data statistics. Average sentence length is between 11.8 and 13.1.
**BLEU Score**

- 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

---

Experiment 1: Fisher

<table>
<thead>
<tr>
<th>test-time Trained on</th>
<th>Trained on</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputs</td>
<td>R</td>
</tr>
<tr>
<td>reference</td>
<td>53.9 (7.1)</td>
</tr>
<tr>
<td>oracle</td>
<td>44.9 (13.4)</td>
</tr>
<tr>
<td>1-best</td>
<td>35.8 (24.7)</td>
</tr>
<tr>
<td>lattice</td>
<td>25.9 (23.4)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fisher/Dev2</th>
<th>Fisher/Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>52.2</td>
</tr>
<tr>
<td>R+1</td>
<td>51.8</td>
</tr>
<tr>
<td>R+L</td>
<td>52.2</td>
</tr>
<tr>
<td>R+L+S</td>
<td>52.2</td>
</tr>
</tbody>
</table>

Table 3: BLEU scores (4 references) and perplexities (*in brackets*). Models are pre-trained only (R), fine-tuned on either 1-best outputs (R+1), lattices without scores (R+L), or lattices with scores (R+L+S). Statistically significant improvement (paired bootstrap resampling, \( p < 0.05 \)) over 1-best/R+1 is in bold.
Experiment 2: Callhome

Scenario:
- we have a reasonable amount of sequential data available for pre-training
- only a limited amount of lattice training data

Training phase:
1. pre-train models on Fisher/Train
2. fine-tune them on the 9 times smaller Callhome/Train portion of the corpus

Analyzing
- Concatenate all test sets and divide the result into bins according to 1-best WER.
- sample 1000 sentences from each bin
Figure 4: BLEU score over varying 1-best WERs.

- For very low WERs, lattices do not improve over 1-best inputs.
- In all other cases, lattices are helpful.
We investigated translating uncertain inputs from an error-prone up-stream component using a neural lattice-to-sequence model.

Our proposed model takes word lattices as input and is able to take advantage of lattice scores.