# Neural Lattice-to-Sequence Models for Uncertain Inputs

**Matthias Sperber**[1], **Graham Neubig**[2], **Jan Niehues**[1], **Alex Waibel**[1]
[1]Karlsruhe Institute of Technology, Germany
[2]Carnegie Mellon University, USA
matthias.sperber@kit.edu

## 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. We experiment with speech translation lattices and report consistent improvements over baselines that translate either the 1-best hypothesis or the lattice without posterior scores.

## 1 Introduction

In many natural language processing tasks, we will require a down-stream system to consume the input of an up-stream system, such as word segmenters, part of speech taggers, or automatic speech recognizers. Among these, one of the most prototypical and widely used examples is speech translation, where a down-stream translation system must consume the output of an up-stream automatic speech recognition (ASR) system.

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 from a previous step (Ney, 1999; Casacuberta et al., 2004). Several works have shown quality improvements by translating lattices, compared to translating only the single best upstream output. Examples include translating lattice representations of ASR output (Saleem et al., 2004; Zhang et al., 2005; Matusov et al., 2008), multiple word segmentations, and morphological alternatives (Dyer et al., 2008).

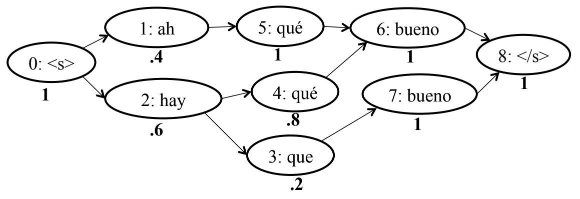

Figure 1: A lattice with 3 possible paths and posterior scores. Translating the whole lattice potentially allows for recovering from errors in its 1-best hypothesis.

Recently, neural sequence-to-sequence (seq2seq) models (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014; Bahdanau et al., 2015) have often been preferred over the traditional methods for their strong empirical results and appealing end-to-end modeling. These models force us to rethink approaches to handling lattices, because their recurrent design no longer allows for efficient lattice decoding using dynamic programming as was used in the earlier works.

As a remedy, Su et al. (2017) proposed replacing the sequential encoder by a lattice encoder to obtain a lattice-to-sequence (lat2seq) model. This is achieved by extending the encoder's Gated Recurrent Units (GRUs) (Cho et al., 2014) to be conditioned on multiple predecessor paths. The authors demonstrate improvements in Chinese-to-English translation by translating lattices that combine the output of multiple word segmenters, rather than a single segmented sequence.

However, this model does not address one aspect of lattices that we argue is critical to obtaining good translation results: their ability to encode the certainty or uncertainty of the paths through the use of posterior scores. Specifically, we postulate that these scores are essential for tasks that require handling lattices with a considerable amount of erroneous content, such as those produced by ASR systems. In this paper, we propose a lattice-to-sequence model that accounts for this uncertainty. Specifically, our contributions are as follows:

- We employ the popular child-sum TreeLSTM (Tai et al., 2015) to derive a lattice encoder that replaces the sequential encoder in an attentional encoder-decoder model. We show empirically that this approach yields only minor improvements compared to a baseline fine-tuned on sequential ASR outputs. This finding stands in contrast to the positive results by Su et al. (2017), and by Ladhak et al. (2016) on a lattice classification task, and suggests higher learning complexity of our speech translation task.

- We hypothesize that lattice scores are crucial in aiding training and inference, and propose several techniques for integrating lattice scores into the model: (1) We compute *weighted child-sums*,[1] where hidden units in the lattice encoder are conditioned on their predecessor hidden units such that predecessors with low probability are less influential on the current hidden state. (2) We bias the TreeLSTM's *forget gates* for each incoming connection toward being more forgetful for predecessors with low probability, such that their cell states become relatively less influential. (3) We *bias the attention mechanism* to put more focus on source embeddings belonging to nodes with high lattice scores. We demonstrate empirically that the third proposed technique is particularly effective and produces strong gains over the baseline. According to our knowledge, this is the first attempt of integrating lattice scores already at the *training stage* of a machine translation model.

- We exploit the fact that our lattice encoder is a strict generalization of a sequential encoder by *pre-training* on sequential data, obtaining faster and better training convergence on large corpora of parallel sequential data.

[1]This is reminiscent of the weighted pooling strategy by Ladhak et al. (2016) for spoken utterance classification.

We conduct experiments on the Fisher and Callhome Spanish–English Speech Translation Corpus (Post et al., 2013) and report improvements of 1.4 BLEU points on Fisher and 0.8 BLEU points on Callhome, compared to a strong baseline optimized for translating 1-best ASR outputs. We find that the proposed integration of lattice scores is crucial for achieving these improvements.

## 2 Background

Our work extends the seminal work on attentional encoder-decoder models (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014; Bahdanau et al., 2015) which we survey in this section.

Given an input sequence $\boldsymbol{x} = (x_1, x_2, \ldots, x_N)$, the goal is to generate an appropriate output sequence $\boldsymbol{y} = (y_1, y_2, \ldots, y_M)$. The conditional probability $p(\boldsymbol{y} \mid \boldsymbol{x})$ is estimated using parameters trained on a parallel corpus, e.g. of sentences in the source and target language in a translation task. This probability is factorized as the product of conditional probabilities of each token to be generated: $p(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{t=1}^{M} p(y_t \mid \boldsymbol{y}_{<t}, \boldsymbol{x})$. The training objective is to estimate parameters $\theta$ that maximize the log-likelihood of the sentence pairs in a given parallel training set $D$: $J(\theta) = \sum_{(\boldsymbol{x},\boldsymbol{y})\in D} \log p(\boldsymbol{y} \mid \boldsymbol{x}; \theta)$.

### 2.1 Encoder

In our baseline model, the encoder is a bi-directional recurrent neural network (RNN), following (Bahdanau et al., 2015). Here, the source sentence is processed in both the forward and backward directions with two separate RNNs. For every input $x_i$, two hidden states are generated as

$$\overrightarrow{\mathbf{h}}_i = \text{LSTM}\left(E_{\mathit{fwd}}(x_i), \overrightarrow{\mathbf{h}}_{i-1}\right) \quad (1)$$

$$\overleftarrow{\mathbf{h}}_i = \text{LSTM}\left(E_{\mathit{bwd}}(x_i), \overleftarrow{\mathbf{h}}_{i+1}\right), \quad (2)$$

where $E_{\mathit{fwd}}$ and $E_{\mathit{bwd}}$ are source embedding lookup tables. We opt for long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) recurrent units because of their high performance and in order to later take advantage of the TreeLSTM extension (Tai et al., 2015). We stack multiple LSTM layers and concatenate the final layer into the final source hidden state $\mathbf{h}_i = \overrightarrow{\mathbf{h}}_i \mid \overleftarrow{\mathbf{h}}_i$, where layer indices are omitted for simplicity.

### 2.2 Attention

We use an attention mechanism (Luong et al., 2015) to summarize the encoder outputs into a fixed-size representation. At each decoding time step $j$, a context vector $\mathbf{c}_j$ is computed as a weighted average of the source hidden states: $\mathbf{c}_j = \sum_{i=1}^{N} \alpha_{ij} \mathbf{h}_i$. The normalized attentional weights $\alpha_{ij}$ measure the relative importance of the source words for the current decoding step and are computed as a softmax with normalization factor $Z$ summing over $i$:

$$\alpha_{ij} = \frac{1}{Z} \exp\left(\mathrm{s}\left(\mathbf{s}_{j-1}, \mathbf{h}_i\right)\right) \tag{3}$$

$\mathrm{s}(\cdot)$ is a feed-forward neural network with a single layer that estimates the importance of source hidden state $\mathbf{h}_i$ for producing the next target symbol $y_j$, conditioned on the previous decoder state $\mathbf{s}_{j-1}$.

## 2.3 Decoder

The decoder creates output symbols one by one, conditioned on the encoder states via the attention mechanism. It contains another LSTM, initialized using the final encoder hidden state: $\mathbf{s}_0 = \mathbf{h}_N$. The decoding at step $j$ assumes a special start-of-sequence symbol $y_0$ and is computed as $\mathbf{s}_j = \mathrm{LSTM}\left(E_{trg}(y_{j-1}), \mathbf{s}_{j-1}\right)$, and then $\tilde{\mathbf{s}}_t = \tanh(W_{hs}[\mathbf{s}_j; \mathbf{c}_j] + \mathbf{b}_{hs})$ The conditional probability that the $j$-th target word is generated is: $p(y_j \mid \boldsymbol{y}_{<j}, \boldsymbol{x}) = \mathrm{softmax}(W_{so}\tilde{\mathbf{s}}_t + \mathbf{b}_{so})$. Here, $E_{trg}$ is the target embedding lookup table, $W_{hs}$ and $W_{so}$ are weight matrices, and $\mathbf{b}_{hs}$ and $\mathbf{b}_{so}$ are bias vectors.

During decoding beam search is used to find an output sequence with high conditional probability.

## 3 Attentional Lattice-to-Sequence Model

The seq2seq model described above assumes sequential inputs and is therefore limited to taking a single output of an up-stream model as input. Instead, we wish to consume lattices to carry over uncertainties from an up-stream model.

### 3.1 Lattices

Lattices (e.g. Figure 1) represent multiple ambiguous or competing sequences in a compact form. They are a more efficient alternative to enumerating all hypotheses as an $n$-best list, as they allow for avoiding redundant computation over subsequences shared between multiple hypotheses. Lattices can either be produced directly, e.g. by an ASR dumping its pruned search space (Post et al., 2013), or can be obtained by merging several $n$-best sequences (Dyer et al., 2008; Su et al., 2017).

A word lattice $\mathcal{G} = \langle V, E \rangle$ is a directed, connected, and acyclic graph with nodes $V$ and edges $E$. $V \subset \mathbb{N}$ is a node set, and $(k,i) \in E$ denotes an edge connecting node $k$ to node $i$. $C(i)$ denotes the set of predecessor nodes for node $i$. We assume that all nodes follow a topological ordering, such that $k < i \; \forall \; k \in C(i)$. Each node $i$ is assigned a word label $w(i)$. [2] Every lattice contains exactly one start-of-sequence node with only outgoing edges, and exactly one end-of-sequence node with only incoming edges.

### 3.2 Lattices and the TreeLSTM

One thing to notice here is that lattice nodes can have multiple predecessor states. In contrast, hidden states in LSTMs and other sequential RNNs are conditioned on only one predecessor state ($\tilde{h}_j$ in left column of Table 1), rendering standard RNNs unsuitable for the modeling of lattices. Luckily Tai et al. (2015)'s TreeLSTM, which was designed to compose encodings in trees, is also straightforward to apply to lattices; the TreeLSTM composes multiple child states into a parent state, which can also be applied to lattices to compose multiple predecessor states into a successor state. Table 1, middle column, shows the TreeLSTM in its child-sum variant that supports an arbitrary number of predecessors. Conditioning on multiple predecessor hidden states is achieved by simply taking their sum as $\tilde{\mathbf{h}}_i$. Cell states from multiple predecessor are each passed through their own forget gates $\mathbf{f}_{jk}$ and then summed.

Encoding a lattice results in one hidden state for each lattice node. Our lat2seq framework uses this network as encoder, computing the attention over all lattice nodes.[3] In other words we replace (1) by the following:

$$\overrightarrow{\mathbf{h}}_i = \mathrm{LatticeLSTM}\left(\mathbf{x}_i, \{\overrightarrow{\mathbf{h}}_k \mid k \in C(i)\}\right) \tag{4}$$

Similarly, we encode the lattice in backward direction and replace (2) accordingly. Figure 2 illustrates the result. The computational complexity of the encoder is $\mathcal{O}(|V| + |E|)$, i.e. linear in the number of nodes plus number of edges in the graph. The complexity of the attention mechanism is $\mathcal{O}(|V|M)$, where $M$ is the output sequence

[2] It is perhaps more common to think of each edge representing a word, but we will motivate why we instead assign word labels to nodes in §3.3.

[3] This is similar in spirit to Eriguchi et al. (2016) who used the TreeLSTM in an attentional tree-to-sequence model.

|  | Sequential LSTM | TreeLSTM | Proposed LatticeLSTM |
|---|---|---|---|
| recurrence | $\tilde{\mathbf{h}}_i = \mathbf{h}_{i-1}$ | $\tilde{\mathbf{h}}_i = \sum_{k\in C(i)} \mathbf{h}_k$ | $\tilde{\mathbf{h}}_i = \sum_{k\in C(i)} \frac{w_{b/f,k}^{\mathbf{S}_h}}{\mathbf{Z}_{h,k}} \mathbf{h}_k$ (5) |
| forget gt. | $\mathbf{f}_i = \sigma\left(W_f \mathbf{x}_i + U_f \tilde{\mathbf{h}}_i + \mathbf{b}_f\right)$ | $\mathbf{f}_{ik} = \sigma(W_f \mathbf{x}_i + U_f \mathbf{h}_k + \mathbf{b}_f)$ | $\mathbf{f}_{ik} = \sigma(W_f \mathbf{x}_i + U_f \mathbf{h}_k + \left[\ln w_{b/f,k} \mathbf{S}_f - \mathbf{Z}_{f,k}\right] + \mathbf{b}_f)$ (6) |
| input gt. | $\mathbf{i}_i = \sigma\left(W_{in} \mathbf{x}_i + U_{in} \tilde{\mathbf{h}}_i + \mathbf{b}_{in}\right)$ | as sequential | as sequential |
| output gt. | $\mathbf{o}_i = \sigma\left(W_o \mathbf{x}_i + U_o \tilde{\mathbf{h}}_i + \mathbf{b}_o\right)$ | as sequential | as sequential |
| update | $\mathbf{u}_i = \tanh\left(W_u \mathbf{x}_i + U_u \tilde{\mathbf{h}}_i + \mathbf{b}_u\right)$ | as sequential | as sequential |
| cell | $\mathbf{c}_i = \mathbf{i}_i \odot \mathbf{u}_i + \mathbf{f}_i \odot \mathbf{c}_{i-1}$ | $\mathbf{c}_i = \mathbf{i}_i \odot \mathbf{u}_i + \sum_{k\in C(i)} \mathbf{f}_{ik} \odot \mathbf{c}_k$ | as TreeLSTM |
| hidden | $\mathbf{h}_i = \mathbf{o}_i \odot \tanh(\mathbf{c}_i)$ | as sequential | as sequential |
| attention | $\alpha_{ij} \propto \exp\left(s\left(\cdot\right)\right)$ |  | $\alpha_{ij} \propto \exp\left[s\left(\cdot\right) + S_a \ln w_{m,i}\right]$ (7) |

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 $\mathbf{x}_j$ are word embeddings or hidden states of a lower layer. $W_{\cdot}$ and $U_{\cdot}$ denote parameter matrices, $\mathbf{b}_{\cdot}$ bias terms, other terms are described in the text.

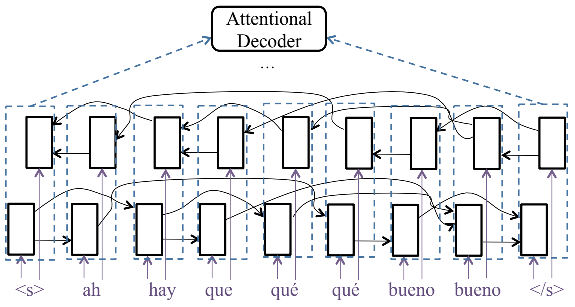

Figure 2: Network structure of a bidirectional lattice encoder with one layer.

length. $|V|$ depends on both the expected input sentence length and the lattice density.

### 3.3 Node-labeled Lattices

At this point we take a step back to motivate our choice of assigning word labels to lattice nodes, which is in contrast to the prior work by Ladhak et al. (2016) and Su et al. (2017) who assign word labels to edges. Recurrent states in edge-labeled lattice encoders are conditioned not only on multiple predecessor states, but must also aggregate words from multiple incoming edges. This implies that hidden units may represent more than one word in the lattice. Moreover, in the edge-labeled case hidden units that are in the same position in forward and backward encoders represent different words, but are nevertheless concatenated and attended to jointly. For these reasons we find our approach of encoding word-labeled lattices more intuitively appealing when used as input to an attentional decoder, although empirical justification is beyond the scope of this paper. We also note that it is easy to convert an edge-labeled lattice into a node-labeled lattice using the line-graph algorithm (Hemminger and Beineke, 1978), which we utilize in this work.

## 4 Integration of Lattice Scores

This section describes the key technical contribution of our work: integration of lattice scores encoding input uncertainty into the lat2seq framework. These lattice scores assign different probabilities to competing paths, and are often provided by up-stream statistical models. For example, an ASR may attach posterior probabilities that capture acoustic evidence and linguistic plausibility of words in the lattice. In this section, we describe our method, first explaining how we normalize scores to a format that is easily usable in our method, then presenting our methods for incorporating these scores into our encoder calculations.

### 4.1 Lattice Score Normalization

Lattice scores that are obtained from upstream systems (such as ASR) are typically given in forward-normalized fashion, interpreted as the probability

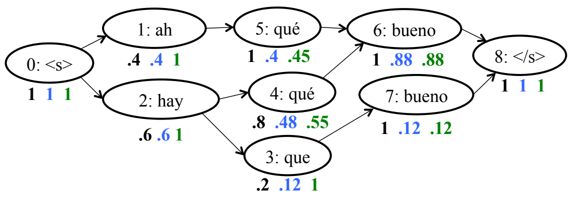

Figure 3: Lattice with forward-normalized, marginal, and backward-normalized scores.

of a node given its predecessor. Here, outgoing edges sum up to one, as illustrated in Figure 1. However, in some of our methods it will be necessary that scores be normalized in the backward direction, so that the weights from incoming connections sum up to one, or globally normalized, so that the probability of the node is the marginal probability of all the paths containing that node.

Let $w_{f,i}$, $w_{m,i}$, $w_{b,i}$ denote forward-normalized, marginal, and backward-normalized scores for node $i$ respectively, illustrated in Figure 3. Given $w_{f,i}$, we can compute marginal probabilities recursively as $w_{m,i} = \sum_{k \in C(i)} w_{m,k} \cdot w_{f,i}$ by using the forward algorithm (Rabiner, 1989). Then, we can normalize backward using $w_{b,i} = \frac{w_{m,i}}{\sum_{k \in C'(i)} w_{m,k}}$, where $C'(i)$ denotes the successors of node $i$. All 3 forms are employed in the sections below.

Furthermore, when integrating these scores into the lat2seq framework, it is desirable to maintain flexibility over how strongly they should impact the model. For this purpose, we introduce a peakiness coefficient $S$. Given a lattice score $w_{b,i}$ in backward direction, we compute $w_{b,i}^S / Z_i$. $Z_i = \sum_{k \in C(i)} w_{b,k}$ is a re-normalization term to ensure that incoming connections still sum up to one. In the forward direction, we compute $w_{f,i}^S / Z_i$ and normalize analogously over outgoing connections. Setting $S$=0 amounts to ignoring the scores by flattening their distribution, while letting $S \to \infty$ puts emphasis solely on the strongest nodes. $S$ can be optimized jointly with the other model parameters via back-propagation during model training.

## 4.2 Integration Approaches

We suggest three methods to integrate these scores into our lat2seq model, with equations shown in the right column of Table 1. These methods can optionally be combined, and we conduct an ablation study to assess the effectivity of each method in isolation (§5.3).

The first method consists of computing a weighted child-sum (WCS), using lattice scores as weights when composing the hidden state $\tilde{\mathbf{h}}_i$. This is based on the intuition that predecessor hidden states with high lattice weights should have a higher influence on their successor than states with low weights. The precise formulas for WCS are shown in (5).

The second method biases the forget gate $\mathbf{f}_{ik}$ for each predecessor cell state such that predecessors with high lattice score are more likely to pass through the forget gate (BFG). The intuition for this is similar to WCS; the composed cell state is more highly influenced by cell states from predecessors with high lattice score. BFG is implemented by introducing a bias term inside the sigmoid as in (6).

In the cases of both WCS and BFG, all hidden units have their own independent peakiness. Thus $\mathbf{S}_h$ and $\mathbf{S}_f$ are vectors, applied element-wise after broadcasting the lattice score. The renormalization terms $\mathbf{Z}_{h,k}$ and $\mathbf{Z}_{f,k}$ are also vectors and are applied element-wise. We use backward-normalized scores $w_{b,i}$ for the forward-directed encoder, and forward-normalized scores $w_{f,i}$ for the backward-directed encoder.

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, note that our lattice encoder is a strict generalization of a sequential encoder. To reduce the computational burden, we exploit this fact and perform a two-step training process where the model is first *pre-trained* on sequential data, then *fine-tuned* on lattice data.[4] 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 lat-

[4]For the sequential data, we set all confidence scores to 1.

tices available. The fine-tuning is then performed on parallel data with lattices on the source side. This is much slower[5] than the pre-training because the network structure changes from sentence to sentence, preventing us from using efficient mini-batched 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 (Post et al., 2013), a corpus of Spanish telephone conversations that includes automatic transcripts and lattices. The Fisher portion consists of telephone conversations between strangers, while the Callhome portion contains telephone conversations between relatives or friends. The training data size is 138,819 sentences (Fisher/Train), and 15,000 sentences (Callhome/Train). Held-out testing data is shown in Table 2. ASR word error rates (WER) are relatively high, due to the spontaneous speaking style and challenging acoustics. Lattices contain on average 3.4 (Fisher/Train) or 4.5 (Callhome/Train) times more words than the corresponding reference transcripts.

For preprocessing, we tokenized and lowercased source and target sides. We removed punctuation from the reference transcripts on the source side for consistency with the automatic transcripts and lattices. All models are pre-trained and fine-tuned on Fisher/Train unless otherwise noted. Our source-side vocabulary contains all words from the automatic transcripts for Fisher/Train, replacing singletons by an unknown word token, totaling 14,648 words. Similarly, on the target side we used all words from the reference translations of Fisher/Train, replacing singletons by the unknown word, yielding 10,800 words in total.

| | 1-best WER | oracle WER | # sent. |
|---|---|---|---|
| Fisher/Dev | 41.3 | 19.3 | 3,979 |
| Fisher/Dev2 | 40.0 | 19.4 | 3,961 |
| Fisher/Test | 36.5 | 16.1 | 3,641 |
| Callhome/Devtest | 64.7 | 36.4 | 3,966 |
| Callhome/Evltest | 65.3 | 37.9 | 1,829 |

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

Our implementation is based on lamtram (Neubig, 2015) and the DyNet (Neubig et al., 2017a) toolkit. We use the implemented attentional model with default parameters: a layer size of 256 per encoder direction and 512 for the decoder. Word embedding size was also set to 512. We used two encoder layers and two decoder layers for better baseline performance. For the sequential baselines, the LSTM variant in the left column of Table 1 was employed. We initialized the forget gate biases to 1 as recommended by Jozefowicz et al. (2015).

We used Adam (Kingma and Ba, 2014) for training, with an empirically determined initial learning rate of 0.001 for pre-training and 0.0001 for fine-tuning. We halve the learning rate when the dev perplexity (on Fisher/Dev) gets worse. Pre-training and fine-tuning on 1-best sequences is performed until convergence, and training on lattices is performed for 2 epochs to keep experimental effort manageable. On Fisher/Train, this took 3-4 days on a fast CPU.[6] Minibatch size was 1000 target words for pre-training, and 20 sentences for lattice training. Unless otherwise noted, we employed all three proposed lattice score integration approaches, and optimized peakiness coefficients jointly during training. We repeat training 3 times with different random seeds for parameter initialization and data shuffling, and report averaged results. We set the decoding beam size to 5.

[5]Our implementation processed sequential inputs about 75 times faster than lattice inputs during training, and overall fine-tuning convergence was 15 times faster. Decoding was only 1.2 times slower when using lattice inputs. Note that recently proposed approaches for autobatching (Neubig et al., 2017b) may considerably speed up lattice training.

[6]For comparison, we tried training on lattices from scratch, which took 9 days (6 epochs) to converge at a dev perplexity that was 10% worse than with the pre-training plus fine-tuning strategy. We also confirmed BLEU scores to be much inferior without pretraining.

## 5.2 Main Results

We compare 4 systems: Performing pre-training on the sequential reference transcripts only (R), fine-tuning on 1-best transcripts (R+1), fine-tuning on lattices without scores (R+L), and fine-tuning on lattices including lattice scores (R+L+S). At test time, we try references, lattice oracles,[7] 1-best transcripts, and lattices as inputs to all 4 systems. The former 2 experiments give upper bounds on achievable translation accuracy, while the latter 2 correspond to a realistic setting. Table 3 shows the results on Fisher/Dev2 and Fisher/Test.

Before even considering lattices, we can see that 1-best fine-tuning boosted BLEU scores quite impressively (1-best/R vs. 1-best/R+1), with gains of 1.3 and 0.7 BLEU points. This stands in contrast to Post et al. (2013) who find the 1-best transcripts not to be helpful for training a hierarchical machine translation system. Possible explanations are learning from repeating error patterns, and improved robustness to erroneous inputs. On top of these gains, our proposed set-up (lattice/R+L+S) improve BLEU scores by another 1.4. Removing the lattice scores (lattice/R+L) diminishes the results and performs worse than the 1-best baseline (1-best/R+1), indicating that the proposed lattice score integration is crucial for good performance. This demonstrates a clear advantage of our proposed method over that of Su et al. (2017).

As can be seen in the table, models fine-tuned on lattices show reasonable performance for both lattice and sequential inputs (1-best/R+L, lattice/R+L, 1-best/R+L+S, lattice/R+L+S). This is not surprising, given that the lattice training data includes lattices of varying density, including lattices with very few paths or even only one path. On the other hand, without fine-tuning on lattices, using lattices as input performs poorly (lattice/R and lattice/R+1). A closer look revealed that translations were often too long, potentially because implicitly learned mechanisms for length control were not ready to handle lattice inputs.

Table 3 reports perplexities for Fisher/Dev2. Unlike the corresponding BLEU scores, the lattice encoder appears stronger than the 1-best baseline in terms of perplexity even without lattice scores (lattice/R+L vs. 1-best/R+1). To understand this better, we computed the average entropy of the decoder softmax, a measure of how much confusion there is in the decoder predictions, independent of whether it selects the correct answer or not. Over the first 100 sentences, this value was 2.24 for 1-best/R+1, 2.39 for lattice/R+L, and 2.15 for lattice/R+L+S. This indicates that the decoder is more confused for lattices without scores, while integrating lattice scores removes this problem. These numbers also suggest that it may be possible to obtain further gains using methods that stabilize the decoder.

[7]The path through the lattice with the best WER.

## 5.3 Ablation Experiments

Next, we conduct an ablation study to assess the impact of the three proposed extensions for integrating lattice scores (§4.2). We train models with different peakiness coefficients $S$, either ignoring lattices scores by fixing $S{=}0$, using lattice scores as-is by fixing $S{=}1$, or optimizing $S$ during training. Table 4 shows the results. Overall, joint training of $S$ gives similar results as fixing $S{=}1$, but both clearly outperform fixing $S{=}0$. Removing confidences (setting $S{=}0$) in one place at a time reveals that the attention mechanism is clearly the most important point of integration, while gains from the integration into child-sum and forget gate are smaller and not always consistent.

We also analyzed what peakiness values were actually learned. We found that all 3 models that we trained for the averaging purposes converged to $S_a{=}0.62$. $\mathbf{S}_h$ and $\mathbf{S}_f$ had per-vector means between 0.92 and 1.0, at standard deviations between 0.02 and 0.04. We conclude that while the peakiness coefficients were not particularly helpful in our experiments, stable convergence behavior makes them safe to use, and they might be helpful on other data sets that may contain lattice scores of higher or lower reliability.

## 5.4 Callhome Experiments

In this experiment, we test a situation in which we have a reasonable amount of sequential data available for pre-training, but only a limited amount of lattice training data for the fine-tuning step. This may be a more realistic situation, because speech translation corpora are scarce. To investigate in this scenario, we again pre-train our models on Fisher/Train, but then fine-tune them on the 9 times smaller Callhome/Train portion of the corpus. We fine-tune for 10 epochs, all other settings are as before. We use Callhome/Evltest for testing. Table 5 shows the results. The trends are consistent to §5.2: The proposed model (lattice/R+L+S) outperforms the 1-best baseline (1-best/R+1) by

| test-time | Trained on | | | | Trained on | | | |
|---|---|---|---|---|---|---|---|---|
| inputs | R | R+1 | R+L | R+L+S | R | R+1 | R+L | R+L+S |
| reference | 53.9 *(7.1)* | 53.8 *(6.5)* | 53.7 *(6.8)* | 54.0 *(6.7)* | 52.2 | 51.8 | 52.2 | 52.7 |
| oracle | 44.9 *(13.4)* | 45.6 *(9.5)* | 45.2 *(10.6)* | 45.2 *(10.6)* | 44.4 | 44.6 | 44.6 | 44.8 |
| 1-best | 35.8 *(24.7)* | 37.1 *(13.7)* | 36.2 *(16.4)* | 36.2 *(16.3)* | 35.9 | 36.6 | 36.2 | 36.4 |
| lattice | 25.9 *(23.4)* | 25.8 *(15.7)* | 36.9 *(13.0)* | **38.5** *(12.6)* | 26.2 | 25.8 | 36.1 | **38.0** |
| | Fisher/Dev2 | | | | Fisher/Test | | | |

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.

| BATT $S_a$ | WCS $\mathbf{S}_h$ | BFG $\mathbf{S}_f$ | Fisher /Dev2 | Fisher /Test |
|---|---|---|---|---|
| 0 | 0 | 0 | 36.9 | 36.1 |
| 1 | 1 | 1 | **38.2** | **37.4** |
| * | * | * | **38.5** | **38.0** |
| 0 | 1 | 1 | 37.2 | 36.2 |
| 1 | 0 | 1 | **37.9** | **37.5** |
| 1 | 1 | 0 | **38.2** | **37.8** |
| 0 | * | * | 37.0 | 36.3 |
| * | 0 | * | **38.3** | **37.9** |
| * | * | 0 | **38.1** | **37.5** |
| 1-best/R+1 | | | 37.2 | 36.6 |

Table 4: BLEU scores (4 references) for differently configured peakiness coefficients $S_a, \mathbf{S}_h, \mathbf{S}_f$. 0/1 means fixing to that value, * indicates optimization during training. Statistically significant improvement over 1-best/R+1 is in bold.

| test-time | Trained on | | | |
|---|---|---|---|---|
| inputs | R | R+1 | R+L | R+L+S |
| reference | 24.7 | 24.3 | 24.8 | 24.4 |
| oracle | 15.8 | 16.8 | 16.3 | 15.9 |
| 1-best | 11.8 | 13.3 | 12.4 | 12.0 |
| lattice | 9.3 | 7.1 | **13.7** | **14.1** |

Table 5: BLEU scores on Callhome/Evltest (1 reference). All models are pre-trained on Fisher/Train references (R), and potentially fine-tuned on Callhome/Train. The best result using 1-best or lattice inputs is in bold. Statistically significant improvement over 1-best/R+1 is in bold.

0.8 BLEU points, which in turn beats the pre-trained system (1-best/R) by 1.5 BLEU points. Including the lattice scores is clearly beneficial, although lattices without scores also improve over 1-best inputs in this experiment.

### 5.5 Impact of Lattice Quality

Next, we analyze the impact of using lattices and lattice scores as the ASR WER changes. We concatenate all test data from Table 2 and divide the result into bins according to the 1-best WER. We sample 1000 sentences from each bin, and compare BLEU scores between several models.

The results are shown in Figure 4. For very good WERs, lattices do not improve over 1-best inputs, which is unsurprising. In all other cases, lattices are helpful. Lattice scores are most beneficial for moderate WERs, and not beneficial for very high WERs. We speculate that for high WERs, the lattice scores tend to be less reliable than for lower WERs.

## 6 Conclusion

We investigated translating uncertain inputs from an error-prone up-stream component using a neu-

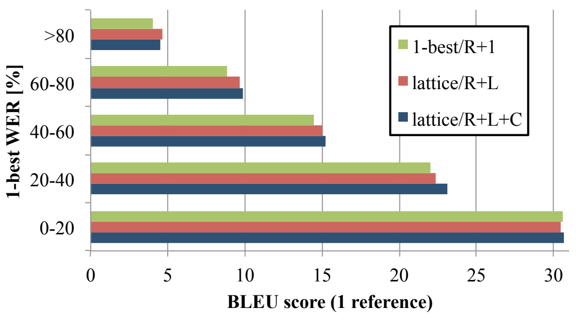

Figure 4: BLEU score over varying 1-best WERs.

ral lattice-to-sequence model. Our proposed model takes word lattices as input and is able to take advantage of lattice scores. In our experiments in a speech translation task we find consistent improvements over translating 1-best transcriptions and that consideration of lattice scores, especially in the attention mechanism, is crucial for obtaining these improvements.

Promising avenues for future work are investigating consensus networks (Mangu et al., 2000) for potential gains in terms of speed or quality as compared to lattice inputs, explicitly dealing with rare or unknown words in the lattice, and facilitating GPU training via autobatching (Neubig et al., 2017b).

## Acknowledgments

We thank Paul Michel and Austin Matthews for their helpful comments on earlier drafts of this paper.

## References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *International Conference on Representation Learning (ICLR)*, San Diego, USA.

F. Casacuberta, H. Ney, F. J. Och, E. Vidal, J. M. Vilar, S. Barrachina, I. Garcia-Varea, D. Llorens, C. Martinez, S. Molau, F. Nevado, M. Pastor, D. Picco, A. Sanchis, and C. Tillmann. 2004. Some approaches to statistical and finite-state speech-to-speech translation. *Computer Speech and Language*, 18(1):25–47.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv preprint arXiv:1406.1078*.

Christopher Dyer, Smaranda Muresan, and Philip Resnik. 2008. Generalizing Word Lattice Translation. Technical Report LAMP-TR-149, University of Maryland, Institute For Advanced Computer Studies.

Akiko Eriguchi, Kazuma Hashimoto, and Yoshimasa Tsuruoka. 2016. Tree-to-Sequence Attentional Neural Machine Translation. In *Association for Computational Linguistic (ACL)*, pages 823–833, Berlin, Germany.

R.L. Hemminger and L.W. Beineke. 1978. Line graphs and line digraphs. In *Selected Topics in Graph Theory*, pages 271–305. Academic Press Inc.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural computation*, 9(8):1735–1780.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. 2015. An Empirical Exploration of Recurrent Network Architectures. In *International Conference on Machine Learning (ICML)*, Lille, France.

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent Continuous Translation Models. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1700–1709, Seattle, Washington, USA.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Faisal Ladhak, Ankur Gandhe, Markus Dreyer, Lambert Mathias, Ariya Rastrow, and Björn Hoffmeister. 2016. LatticeRnn: Recurrent Neural Networks over Lattices. In *Annual Conference of the International Speech Communication Association (InterSpeech)*, pages 695–699, San Francisco, USA.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1412–1421, Lisbon, Portugal.

Lidia Mangu, Eric Brill, and Andreas Stolcke. 2000. Finding consensus in speech recognition: word error minimization and other applications of confusion networks. *Computer Speech & Language*, 14(4):373–400.

Evgeny Matusov, Björn Hoffmeister, and Hermann Ney. 2008. ASR word lattice translation with exhaustive reordering is possible. In *Annual Conference of the International Speech Communication Association (InterSpeech)*, pages 2342–2345, Brisbane, Australia.

Graham Neubig. 2015. lamtram: A Toolkit for Language and Translation Modeling using Neural Networks. http://www.github.com/neubig/lamtram.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. 2017a. DyNet: The Dynamic Neural Network Toolkit. *arXiv preprint arXiv:1701.03980*.

Graham Neubig, Yoav Goldberg, and Chris Dyer. 2017b. On-the-fly Operation Batching in Dynamic Computation Graphs. *arXiv preprint arXiv:1705.07860*.

Hermann Ney. 1999. Speech Translation: Coupling of Recognition and Translation. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 517–520, Phoenix, USA.

Matt Post, Gaurav Kumar, Adam Lopez, Damianos Karakos, Chris Callison-Burch, and Sanjeev Khudanpur. 2013. Improved Speech-to-Text Translation with the Fisher and Callhome Spanish–English Speech Translation Corpus. In *International Workshop on Spoken Language Translation (IWSLT)*, Heidelberg, Germany.

Lawrence R. Rabiner. 1989. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*.

Shirin Saleem, Szu-Chen (Stan) Jou, Stephan Vogel, and Tanja Schultz. 2004. Using Word Lattice Information for a Tighter Coupling in Speech Translation Systems. In *International Conference of Spoken Language Processing (ICSLP)*, pages 41–44, Jeju Island, Korea.

Jinsong Su, Zhixing Tan, Deyi Xiong, Rongrong Ji, Xiaodong Shi, and Yang Liu. 2017. Lattice-Based Recurrent Neural Network Encoders for Neural Machine Translation. In *Conference on Artificial Intelligence (AAAI)*, pages 3302–3308.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3104–3112, Montréal, Canada.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In *Association for Computational Linguistic (ACL)*, pages 1556–1566, Beijing, China.

Ruiqiang Zhang, Genichiro Kikui, Hirofumi Yamamoto, and Wai-Kit Lo. 2005. A Decoding Algorithm for Word Lattice Translation in Speech Translation. In *International Workshop on Spoken Language Translation (IWSLT)*, pages 23–29, Pittsburgh, USA.