LATTICERNN: Recurrent Neural Networks over Lattices

Faisal Ladhak, Ankur Gandhe, Markus Dreyer
Lambert Mathias, Ariya Rastrow, Björn Hoffmeister

Amazon.com
{faisall,aggandhe,mddreyer,mathiasl,arastrow,bjornh}@amazon.com

Abstract

We present a new model called LATTICERNN, which generalizes recurrent neural networks (RNNs) to process weighted lattices as input, instead of sequences. A LATTICERNN can encode the complete structure of a lattice into a dense representation, which makes it suitable to a variety of problems, including rescoring, classifying, parsing, or translating lattices using deep neural networks (DNNs). In this paper, we use LATTICERNNs for a classification task: each lattice represents the output from an automatic speech recognition (ASR) component of a spoken language understanding (SLU) system, and we classify the intent of the spoken utterance based on the lattice embedding computed by a LATTICERNN. We show that making decisions based on the full ASR output lattice, as opposed to 1-best or n-best hypotheses, makes SLU systems more robust to ASR errors. Our experiments yield improvements of 13% over a baseline RNN system trained on transcriptions and 10% over an n-best list rescoring system for intent classification.

1. Introduction

The output of many spoken language understanding components such as automatic speech recognition (ASR) can be encoded as a lattice: a directed acyclic graph (DAG). Lattices can represent multiple prediction hypotheses efficiently, see Figure 1 for an example speech lattice. The hypotheses in a lattice share substructure, as opposed to the hypotheses in an n-best list. In this way, lattices can represent exponentially many weighted hypotheses efficiently.

It has been shown that processing full lattices, as opposed to 1-best or n-best hypotheses, can improve robustness and task performance since they preserve ambiguity and avoid making hard decisions too early. In machine translation, researchers have translated lattices representing multiple segmentations [1] or paraphrases [2] of the foreign input; in speech recognition, lattices representing the recognition hyopthesis have been rescored [3], parsed [4], translated [5], or classified [6].

However, recurrent neural networks (RNNs) have not been applied to lattices since they originally process sequences only. On sequences, RNNs perform well for tasks like classifying, parsing or translating ([7], [8], [9]), so it is natural to extend RNNs to benefit from increased robustness and task performance that lattice inputs provide.

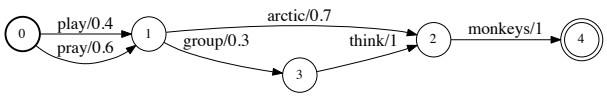


Figure 1: Example ASR lattice: Relying on the 1-best path may lead to intent classification errors.

The LATTICERNN model we present recurrently computes an RNN embedding of each lattice state; the embedding of the final state represents the content and structure of the complete lattice.

In this paper, we focus on the task of intent classification [10], where each utterance in a spoken language understanding (SLU) system is classified as one of several predefined *intents*. An intent is the action that the speaker wishes to evoke from the system, e.g., `PlayMusic`, `GetWeatherForecast`, `SetTimer`, etc. Each spoken utterance is represented as a weighted lattice that encodes the output hypotheses of an automatic speech recognizer (ASR). To perform classification, we compute a dense representation of the weighted lattice using LATTICERNN and use it as input features into a softmax layer; all model weights are trained jointly. We show significant improvements over classifying based on n-best lists.

We describe the formal LATTICERNN model in the next section, then describe experiments on the intent classification task (Section 3) and compare our approach to related work (Section 4).

2. Framework for DNN Training Using Lattices

2.1. RNNs

While feed-forward neural networks treat each input in a sequence independently, recurrent neural networks have feedback connections in the hidden layer, which allow activations to flow through time. This enables RNNs to learn temporal associations and perform predictions over sequences. This means that recurrent networks are well suited for tasks that require a "memory" to be retained over time in order to do predictions over a sequence, making them a great choice for modeling natural language.

The formulation of the forward pass of an RNN is as follows:

$$\boldsymbol{h}_t = g(\boldsymbol{W}_x^T \boldsymbol{x}_t + \boldsymbol{W}_r^T \boldsymbol{h}_{t-1} + \boldsymbol{b}) \quad (1)$$

where $\boldsymbol{x}_t$ is the input at time t, $\boldsymbol{W}_x$ is the weight matrix for the input, $\boldsymbol{W}_r$ is the recurrent weight matrix, $\boldsymbol{h}_t$ is the RNN state at time t, and g is an activation function. The affine transform can be abstracted out from the recurrence, since it does not have temporal dependencies:

$$\boldsymbol{z}_t = \boldsymbol{W}_x^T \boldsymbol{x}_t + \boldsymbol{b} \quad (2)$$

$$\boldsymbol{h}_t = g(\boldsymbol{z}_t + \boldsymbol{W}_r^T \boldsymbol{h}_{t-1}) \quad (3)$$

The corresponding gradient computations for Equation 3

are as follows:

$$\delta_z^t = (\delta_h^t + \boldsymbol{W}_r^T \delta_z^{t+1}) * g'(\boldsymbol{z}_t + \boldsymbol{W}_r^T \boldsymbol{h}_{t-1}) \quad (4)$$

$$\delta_{W_r} = \sum_t \delta_z^t \boldsymbol{h}_{t-1} \quad (5)$$

where δ_z^t and δ_h^t are the gradients at time t of the input transformation and recurrent activation respectively.

2.2. Extending RNNs to Lattices

A lattice $L = (\Sigma, S, E, s_i, s_f)$ is defined by a label set Σ, a finite set of states S, a finite set of arcs E, an initial state $s_i \in S$, and a final state $s_f \subseteq S$.[1] An arc $e = (p[e], l[e], w[e], n[e]) \in E$ is a transition from a source state $p[e]$ to a destination state $n[e]$ (both in S), with label $l[e]$ and weight $w[e]$. We also use $\boldsymbol{x}[e]$ to denote the input vector associated with arc e. For example, $\boldsymbol{x}[e]$ can be the *one-hot* representation corresponding to $l[e]$. The challenge to extend the RNN framework to encode a lattice as a dense vector is that a lattice is not a single sequence – it is a compact representation of multiple overlapping sequences. Ideally, information about the structure of the lattice and the distribution over the hypotheses space (represented by the lattice) must be retained by the encoding framework.[2]

We propose to traverse an input lattice in *topological order*, compute RNN hidden states for each visited arc and lattice state and use the RNN hidden state of the lattice final state as the *dense vector* representing the entire lattice. We describe the details of the proposed LATTICERNN framework in the following section.

2.3. Computation Over Lattices

Neural network components can be broken down into two general types: temporal and non-temporal. A temporal components have temporal dependencies in their computation, such as RNNs, LSTMs, windows, etc. Non-temporal components have no temporal dependencies, such as affine transforms and activation functions. In order to perform computations over lattices, we modify the temporal components, while the non-temporal components remain unchanged. Below is the reformulation of RNNs for lattices, but this general idea could be extended to any temporal component.

The RNN formulation presented in Section 2.1 works for simple linear-chain lattices, but it does not generalize to lattices that have states with multiple incoming and outgoing arcs. We propose the formulation for LATTICERNNs shown in Algorithm 1, as a generalization of RNNs to work with any arbitrary DAG. The basic idea involves recurrence across the topology of the lattice, by pooling over the arc representations of the incoming arcs into a state, and propagating the same hidden representation to the outgoing arcs from a state (see Fig. 2 for a visualization). Here, $\boldsymbol{h}[e]$ and $\boldsymbol{h}[s]$ denote RNN hidden representations for arc e ($\in E$) and state s ($\in S$), respectively. The representation (RNN hidden state) of a lattice state is computed by applying a pooling function, f_{pool}, over the set of representations for all incoming arcs into that state (line 13). The RNN hidden state for the start state s_i is set to the null vector (line 1). Recurrence comes into play when computing the arc representation (RNN hidden state) in line 9. Note that this is similar to

[1] In case of multiple final states, a super-final state is created by connecting each final state to it with an ϵ-arc.

[2] Competing hypothesized words in a specific region of an input speech lattice carry significant information which can be leveraged in a downstream application, e.g., intent classification.

Input : $L = (\Sigma, S, E, s_i, s_f)$, $f_{pool}()$
Output: $\boldsymbol{h}[s_f]$ // lattice representation

1 $\boldsymbol{h}[s_i] = \mathbf{0}$
2 **for** $s \in$ TopSort$(S \setminus \{s_i\})$ **do**
  // incoming arc representations
3 $\mathcal{I}(s) \leftarrow \{\}$ // empty set
4 **for** $e \in$IncomingArcs(s) **do**
5 **if** $l[e] == \epsilon$ **then**
6 $\boldsymbol{h}[e] = \boldsymbol{h}[p[e]]$
7 **else**
8 $\boldsymbol{z}[e] = \boldsymbol{W}_x^T \boldsymbol{x}[e] + \boldsymbol{b}$
9 $\boldsymbol{h}[e] = g(\boldsymbol{z}[e] + \boldsymbol{W}_r^T \boldsymbol{h}[p[e]])$
10 **end**
11 Insert$(\mathcal{I}(s), \boldsymbol{h}[e])$
12 **end**
13 $\boldsymbol{h}[s] = f_{pool}(\mathcal{I}(s))$
14 **end**

Algorithm 1: LATTICERNN forward pass

Equation 3 above, except instead of multiplying the recurrent weights with $\boldsymbol{h}_{t-1}$ we multiply them with the representation of the source state of the arc, $\boldsymbol{h}[p[e]]$. Another thing to note is that for epsilon arcs, we simply propagate the representation of the source state, as shown in line 6.

The gradient update (δ_s) for a state s can be computed as shown below:

$$\delta_s = \sum_{e \in \mathcal{O}(s)} f_{pool}^*(\delta_{\boldsymbol{z}[e]}) \quad (6)$$

where f_{pool}^* is backprop through the pooling function, and $\mathcal{O}(s)$ is the set of all outgoing arcs from state s. The gradients propagated out ($\delta_{z[e]}$) of the LATTICERNN for each arc are as follows:

$$\delta_{z[e]} = (\delta_e + \boldsymbol{W}_r^T \delta_{n[e]}) * g'(\boldsymbol{z}[e] + \boldsymbol{W}_r^T \boldsymbol{h}[p[e]]) \quad (7)$$

where $\delta_{n[e]}$ is the gradient of the destination state for arc e. Note that δ_e is simply the gradient for arc e that is propagated down from the next layer. The gradients for the recurrent weights are calculated as shown below:

$$\delta_{W_r} = \sum_{e \in E} \delta_{\boldsymbol{z}[e]} \boldsymbol{h}[p[e]] \quad (8)$$

3. Experiments

Our experiments evaluate the LATTICERNN model for the task of intent classification of ASR utterances.

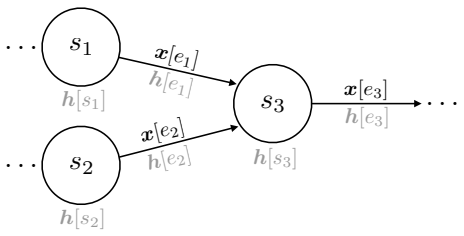
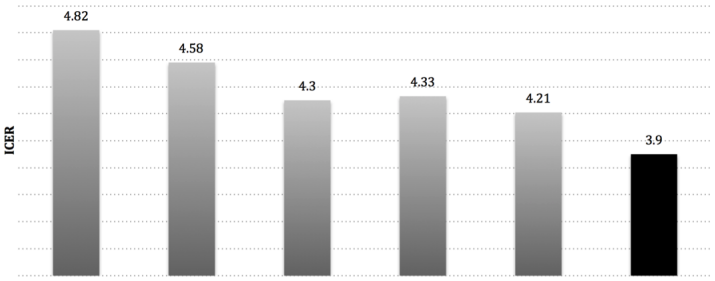


Figure 2: This example shows part of a lattice, with the RNN hidden representations shown in gray. $\boldsymbol{h}[e_1]$ is computed based both on local arc features $\boldsymbol{x}[e_1]$ and the representation $\boldsymbol{h}[s_1]$ of the arc's source state (see Alg. 1, line 9); likewise for $\boldsymbol{h}[e_2]$. State representation $\boldsymbol{h}[s_3]$ is computed by pooling the incoming arc representations $\boldsymbol{h}[e_1]$ and $\boldsymbol{h}[e_2]$ (see Alg. 1, line 13).



Training	Manual transcription (Baseline)	ASR 1-best (*1-best*)	ASR 5-best (*5-best lattice*)	ASR 15-best (*15-best lattice*)	ASR lattice (*Full Lattice*)	Manual transcription **(Oracle)**
Evaluation	ASR 1-best	ASR 1-best	ASR 5-best	ASR 15-best	ASR lattice	Manual transcription

Figure 3: ICER for different training and evaluation conditions.

We compare the following different systems configurations:[3]

1. *Baseline*: Train on manual transcriptions
2. *1-best*: Train on 1-best path in the ASR lattice
3. *n-best lattice*: Use LATTICERNN to train and test on n-best paths in the form of n parallel paths in an ASR lattice, merging only at the final state
4. *Full lattice*: Use LATTICERNN to train and test on the entire ASR lattice
5. *Oracle*: Train and evaluate on manual transcription. This is an upper bound on performance, assuming that the ASR system was as good as human annotators.

For evaluation, we use intent classification error rate (ICER), i.e., the percentage of classification errors made by the model.

3.1. Training Setup

We use 300-dimensional GloVe vectors [11] as the feature representation of words on the lattice arcs, with 100 hidden units, i.e., $\boldsymbol{W}_x \in \mathbb{R}^{300\times100}$, $\boldsymbol{W}_r \in \mathbb{R}^{100\times100}$, and $\boldsymbol{b} \in \mathbb{R}^{100}$. We use the logistic sigmoid function as activation function g. We connect the embedding of the lattice final state to a softmax layer over all intents. We use cross entropy as the training objective.

In all our experiments, we set the initial learning rate to 0.05 and reduce it by 0.5 after each epoch if the loss on a held-out development set does not improve. For the experiments we used 26 intents, a training set of 806K utterances, and a development and test set each of 80K. All the experiments are performed using our in-house DNN toolkit [12] and lattices were extracted from our in-house ASR system based on [13, 14].

3.2. Results with LATTICERNN

Figure 3 shows the ICER results on all the models described in Section 3. Using the same baseline model, we find that noise due to ASR errors results in a ICER on ASR 1-best that is 19% relative worse than the same system tested on manual transcriptions. Training and testing on the ASR n-best, allows the model to recover from ASR errors as seen in the improvements in ICER at $n = 1, 5, 15$. The best *Full lattice* model improves over the baseline system ICER by 13% relative. Training the LATTICERNN on a larger input hypothesis space allows the model to recover from ASR errors, with significant gains over the *1-best* or the *n-best lattice* systems. Furthermore, even though the *15-best* model has more hypothesis paths in the input than the *5-best* model, many paths are incorrect; combining their representation only at the final state in LATTICERNN leads to a slightly worse performance. Possibly, encoding the lattice topology is important in order to efficiently recover from ASR errors.

3.3. Effect of Pooling

In LATTICERNN, the pooling function is responsible for determining how incoming arcs into a state are combined. We investigate two different pooling functions:

1. MeanPool: The representations of incoming arcs are averaged into a single vector.
2. WeightedPool: The representations of incoming arcs are weighted by arc scores and summed into a single vector.

In case of lattices from the ASR system, each arc has an acoustic score and a language model score associated with it, which is combined using an acoustic scale factor. Using raw combined scores does not improve performance over MeanPool, because the raw scores vary a lot from one lattice to another. Instead, we calculate the posterior score of each arc by running the forward backward algorithm [15] and then normalizing the scores over incoming arcs for each state.

	Pooling	ICER	Rel. impr.
Baseline		4.82	
5-best lattice	MeanPool	4.34	10%
	WeightedPool	4.3	11%
15-best lattice	MeanPool	4.61	4%
	WeightedPool	4.33	10%
Full lattice	MeanPool	4.37	9%
	WeightedPool	4.21	13%

Table 1: Effect of pooling on different lattice inputs

As the depth of the lattice increases, performance with MeanPool gets worse, indicating that the noisy (high error rate)

[3] All configurations are evaluated on ASR 1-best output, unless stated otherwise.

paths in the lattice are dominating the encoding. Instead, incorporating the lattice posteriors to weight each incoming arc in WeightedMeanPool, each state encoding is biased towards the more likely arcs and hence the final encoding towards the more likely paths in the lattice.

3.4. Comparison to RNN rescoring

Another way of incorporating a larger hypothesis space in the input is to predict the intent by rescoring the n-best paths extracted from the ASR lattice. The *baseline* model is used to get the probability distribution over intents for each path and the final intent is obtained using maximum a posteriori (MAP) estimation:

$$p(\text{Intent}|\text{nbest}) = \sum_{w \in nbest} p(\boldsymbol{w}|\text{nbest}) * p(\text{Intent}|\boldsymbol{w}) \quad (9)$$

where $p(\boldsymbol{w}|\text{nbest})$ is the posterior probablity of the hypothesis $\boldsymbol{w}$.

Model	ICER	Rel. impr.
Baseline	4.82	
5-best rescore	4.78	1%
5-best lattice	4.3	11%
15-best rescore	4.67	3%
15-best lattice	4.33	10%

Table 2: Results of rescoring on ICER

Table 3.4 shows the result of the rescoring experiments. n-best rescoring improves the ICER by 3% over baseline. However, it is expensive to compute the intent probabilities since the model needs to be run n times over the paths in the list, whereas the LATTICERNN can process the n-best lattice in a single step. Another advantage of the *n-best lattice* model is that the final prediction is computed from the combined representation of all the paths. This is perhaps the most important feature of the LATTICERNN as the performance is 7-10% better than the MAP rescoring.

3.5. Effect of Domain

The accuracy of the intent classifier is predicated on getting certain indicative keywords right for that particular intent. Complex domains are characterized by the number of keywords indicative of the particular intent, as well as the number of intents that have to predicted for that domain. For example, the word "weather" is an important keyword for the *WeatherForecast* intent, whereas, the words "play", "album", "music" etc. are indicative keywords for the Music domain intents. The ASR system is much more likely to make errors on the more complex domains, as it needs to get many more of the indicative keywords right in order to classify correctly into one of several possible intents. Table 3 below, shows the ICER for various domains with increasing complexity - Music, Shopping, Notifications and Weather.

	Music	Shopping	Notification	Weather
# intents	9	6	10	1
Baseline	5.66	6.95	4.68	1.58
Full lattice	4.67	5.42	4.2	1.29
Rel. impr.	17%	22%	10%	18%

Table 3: ICER and relative improvement on different domains

We get larger improvements on more complex domains, using the LATTICERNN. This indicates that the lattice encoding is more robust to ASR errors for the intent classification task.

4. Related Work

There has been previous work in using word lattices and n-best hypotheses output by speech recognition systems for the spoken language understanding task [16, 17, 18, 10]. In [10], the authors show improvements by rescoring n-best hypotheses for joint intent classification and speech recognition. Beyond n-best lists, [18, 17] use confidence scores from a confusion network to improve intent classification. However, these works do not exploit the structure of the richer hypothesis space provided by the confusion network. [19, 20] use confusion networks to assign semantic tags to each bin in the network. They extract n-gram features over the bins and use them in a conditional random field. However, this requires aligning the words into bins, and requires feature engineering to extract various n-gram combinations over the confusion networks.

More recently, there has been work on moving to neural network architectures for slot-filling and intent classification tasks [21, 22, 23, 24]. Using recurrent architectures have been shown to improve upon traditional conditional random fields and maximum entropy classifiers for slot-filling and intent classification tasks. However, these works are focused on training recurrent architectures on reference text and applied to 1-best ASR hypotheses. [25] presents approaches for fast rescoring of lattices using a recurrent architecture. However, the recurrent network is trained on reference transcription and does not leverage the richer hypothesis space during training. Our work differs in that we train our neural network on the entire lattice so as to better capture the context and the structure.

The closest approach to our work is [26], which presents a method to recursively model tree structure for a sentiment classification task. In order to score a given tree under their model, the tree is traversed from the leaves to the root, successively combining children representations into parent representation, much like we traverse from the start state to the final state, but we account for a variable number of incoming arcs as well as arc labels. Another key difference is that the lattices we encode may contain multiple different strings.

5. Conclusion and Future Work

We have presented LATTICERNN, a novel approach for computing dense fixed length representations of weighted lattices. We have used these lattice representations for intent classification of spoken utterances, where we preserved the ambiguities of the ASR recognition and allowed the classification to be influenced by multiple paths in the lattice representing multiple ASR hypotheses of the utterance.

There are several avenues for future work. Adapting LSTMs to lattice input may perform better than the presented basic LATTICERNN; adding an attention mechanism may further help to focus decisions on certain states or arcs in the lattice; different applications of LATTICERNN may require different pooling strategies for incoming arcs. Finally we are working on extending this formulation to solve structured prediction tasks such as part-of-speech tagging, and entity recognition.

Lattices are ubiquitous in language understanding and processing tasks. We hope that the presented formalism will spark the community's interest in further applications and extensions of this work.

6. References

[1] C. Dyer, S. Muresan, and P. Resnik, "Generalizing word lattice translation," in *Proceedings of ACL-08: HLT*. Columbus, Ohio: Association for Computational Linguistics, June 2008, pp. 1012–1020. [Online]. Available: http://www.aclweb.org/anthology/P/P08/P08-1115

[2] T. Onishi, M. Utiyama, and E. Sumita, "Paraphrase lattice for statistical machine translation," *IEICE TRANSACTIONS on Information and Systems*, vol. 94, no. 6, pp. 1299–1305, 2011.

[3] A. Rastrow, M. Dreyer, A. Sethy, S. Khudanpur, B. Ramabhadran, and M. Dredze, "Hill climbing on speech lattices: A new rescoring framework," in *ICASSP*, 2011.

[4] K. Hall and M. Johnson, "Attention shifting for parsing speech," in *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 2004, p. 40.

[5] L. Mathias and W. Byrne, "Statistical phrase-based speech translation." in *ICASSP (1)*, 2006, pp. 561–564.

[6] C. Cortes, P. Haffner, and M. Mohri, "Rational kernels: Theory and algorithms," *The Journal of Machine Learning Research*, vol. 5, pp. 1035–1062, 2004.

[7] G. Arevian, "Recurrent neural networks for robust real-world text classification," in *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*. IEEE Computer Society, 2007, pp. 326–329.

[8] O. Vinyals, Ł. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton, "Grammar as a foreign language," in *Advances in Neural Information Processing Systems*, 2015, pp. 2755–2763.

[9] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[10] S. Yaman, L. Deng, D. Yu, Y.-Y. Wang, and A. Acero, "An integrative and discriminative technique for spoken utterance classification," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 16, no. 6, pp. 1207–1214, 2008.

[11] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543.

[12] N. Ström, "Scalable distributed DNN training using commodity GPU cloud computing," in *INTERSPEECH*, 2015.

[13] S. H. K. Parthasarathi, B. Hoffmeister, S. Matsoukas, A. Mandal, N. Strom, and S. Garimella, "fMLLR based feature-space speaker adaptation of DNN acoustic models," in *Proceedings Interspeech*, 2015.

[14] S. Garimella, A. Mandal, N. Strom, B. Hoffmeister, S. Matsoukas, and S. H. K. Parthasarathi, "Robust i-vector based adaptation of DNN acoustic model for speech recognition," in *Proceedings Interspeech*, 2015.

[15] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[16] D. Hakkani-Tür, F. Béchet, G. Riccardi, and G. Tur, "Beyond ASR 1-best: Using word confusion networks in spoken language understanding," *Computer Speech & Language*, vol. 20, no. 4, pp. 495–514, 2006.

[17] T. J. Hazen, S. Seneff, and J. Polifroni, "Recognition confidence scoring and its use in speech understanding systems," *Computer Speech & Language*, vol. 16, no. 1, pp. 49–67, 2002.

[18] G. Tur, J. Wright, A. Gorin, G. Riccardi, and D. Hakkani-tür, "Improving spoken language understanding using word confusion networks," in *in Proceedings of the ICSLP*, 2002.

[19] G. Tür, A. Deoras, and D. Hakkani-Tür, "Semantic parsing using word confusion networks with conditional random fields." in *INTERSPEECH*, 2013, pp. 2579–2583.

[20] M. Henderson, M. Gasic, B. Thomson, P. Tsiakoulis, K. Yu, and S. Young, "Discriminative spoken language understanding using word confusion networks," in *Spoken Language Technology Workshop (SLT), 2012 IEEE*. IEEE, 2012, pp. 176–181.

[21] S. Ravuri and A. Stolcke, "Recurrent neural network and LSTM models for lexical utterance classification," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[22] Y. Shi, K. Yao, H. Chen, Y.-C. Pan, M.-Y. Hwang, and B. Peng, "Contextual spoken language understanding using recurrent neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 5271–5275.

[23] G. Mesnil, Y. Dauphin, K. Yao, Y. Bengio, L. Deng, D. Hakkani-Tur, X. He, L. Heck, G. Tur, D. Yu *et al.*, "Using recurrent neural networks for slot filling in spoken language understanding," *Audio, Speech, and Language Processing, IEEE/ACM Transactions on*, vol. 23, no. 3, pp. 530–539, 2015.

[24] K. Yao, B. Peng, Y. Zhang, D. Yu, G. Zweig, and Y. Shi, "Spoken language understanding using long short-term memory neural networks," in *SLT*, 2014.

[25] A. Deoras, T. Mikolov, and K. Church, "A fast re-scoring strategy to capture long-distance dependencies," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2011, pp. 1116–1127.

[26] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, vol. 1631. Citeseer, 2013, p. 1642.