Paper Number: 4992

A Parallelizable Lattice Rescoring Strategy with Neural Language Models

Ke Li[1], Daniel Povey[2], Sanjeev Khudanpur[1]

[1]The Johns Hopkins University, USA, [2]Xiaomi Corp., China.

Motivation

- Lattice rescoring usually involves lattice expansion
  - The general goal of lattice expansion is to make arcs on highly likely paths have unique histories
  - *n*-gram approximation, which merges histories that share $(n-1)$ most recent words, may sacrifice accuracy and waste computation on less likely paths. **Whether can we do better?**
- A major speedup bottleneck of lattice rescoring with neural LMs is LM evaluation
  - Existing methods such as caching computed LM scores and pruning-based algorithms speedup the process by reducing the number of LM evaluations
  - While the sequential LM evaluation order in a lattice is still inefficient. **How to parallel LM evaluations within a lattice?**

A Parallel Lattice Rescoring Strategy ("Non-iterative")

Rescoring Procedure

- Step 1 - Lattice Expansion: expand a lattice with a posterior-based method (with beam pruning applied beforehand)
- Step 2 - Lattice-to-List Conversion: convert the expanded lattice into a minimal list of hypotheses that cover every arc
- Step 3 - Score computation and estimation: compute LM scores of the lists in parallel and estimate LM scores for each arc
- Step 4 - Integrate scores back to the expanded lattice

Posterior-based Lattice Expansion Algorithm

- Idea - only expand very possible arcs, e.g. arc posteriors $> \epsilon$
- The basic question is whether an incoming arc should be split off from the rest of incoming arcs to its destination-state
- The rule is to allocate a new copy of the destination-state if the arc posterior $> \epsilon$ (a predefined threshold), otherwise transition to the original destination-state

Lattice-to-List Conversion

- Definition of *path cover*: a set of paths such that every arc in the lattice is covered by at least one path
- Lattice-to-List conversion aims to find a minimal path cover with condition that each path is the best one for at least one arc it has

Estimation of Neural LM Scores

- Estimation of neural LM scores for each arc is needed since an arc may be shared by multiple paths
- We experiment with three approximation methods:
  - Simply average neural LM scores from shared paths
  - Perform weighted average with weights as neural LM scores of histories on shared paths
  - Choose the neural LM score from the lowest-cost path among the shared paths (Referred to "Semi-Viterbi" in experiments)

Lattice-to-List Conversion

Algorithm 1 A Constrained Path Cover Algorithm

Input: L: a lattice
Output: O: a list of paths, each is represented as a linear FST.

```
procedure ConstrainedPathCover(L)
    TopologicalSort(L)
    P ← []                                   ▷ A list of pairs of a path and its cost
    α, β ← ViterbiForwardBackward(L)
    for s = 0 : S − 1 do                     ▷ Loop over states
        for e ∈ s.out do                     ▷ Loop over outgoing arcs of s
            if best path including e is not generated then
                p, c ← BestPathForAnArc(α,β,s,e)
                P.append((p, c))
    Sort(P)                                  ▷ Sort paths based on their costs
    O ← ConstructOutputLattice(P)
```

A Refined Parallel Lattice Rescoring Strategy ("Iterative")

- Apply score replacement on top of the introduced non-iterative parallel lattice rescoring strategy
  - Score replacement means replacing *n*-gram scores with neural LM ones for lattices from first-pass decoding
  - It is referred to "iterative" since rescoring happens twice

Experimental Setup

- Data: SWBD with 260h speech
- Acoustic models: Factorized TDNN with LFMMI objective (Kaldi)
- Neural LMs: 2-layer LSTM and a 6-layer Transformer (PyTorch); 2-layer LSTM (Kaldi RNNLM)

Experimental Results

Effect of Estimation Methods

Model	ϵ	Average	Weighted Average	Semi-Viterbi
Transformer	0.5	10.7	10.7	**10.6**
	0.05	10.6	10.6	**10.5**

Table 1: WERs on Hub5'00 (full set) of SWBD from the non-iterative lattice rescoring strategy with three estimation methods.

- Similar observation is observed with the LSTM LM

Analysis of Iterative Rescoring

Rescoring Method	Hub5'00	Swb	Callhm
Score replacement	10.8	6.8	14.6
Non-iterative (ϵ = 0.5)	10.6	6.8	14.3
Score replacement + Non-iterative	**10.3**	**6.6**	**14.0**

Table 2: WERs from proposed lattice rescoring strategies with a Transformer LM.

- The better performance of non-iterative rescoring compared with score replacement alone indicates the value of lattice expansion

Experimental Results (cont.)

Comparison with n-gram Expansion

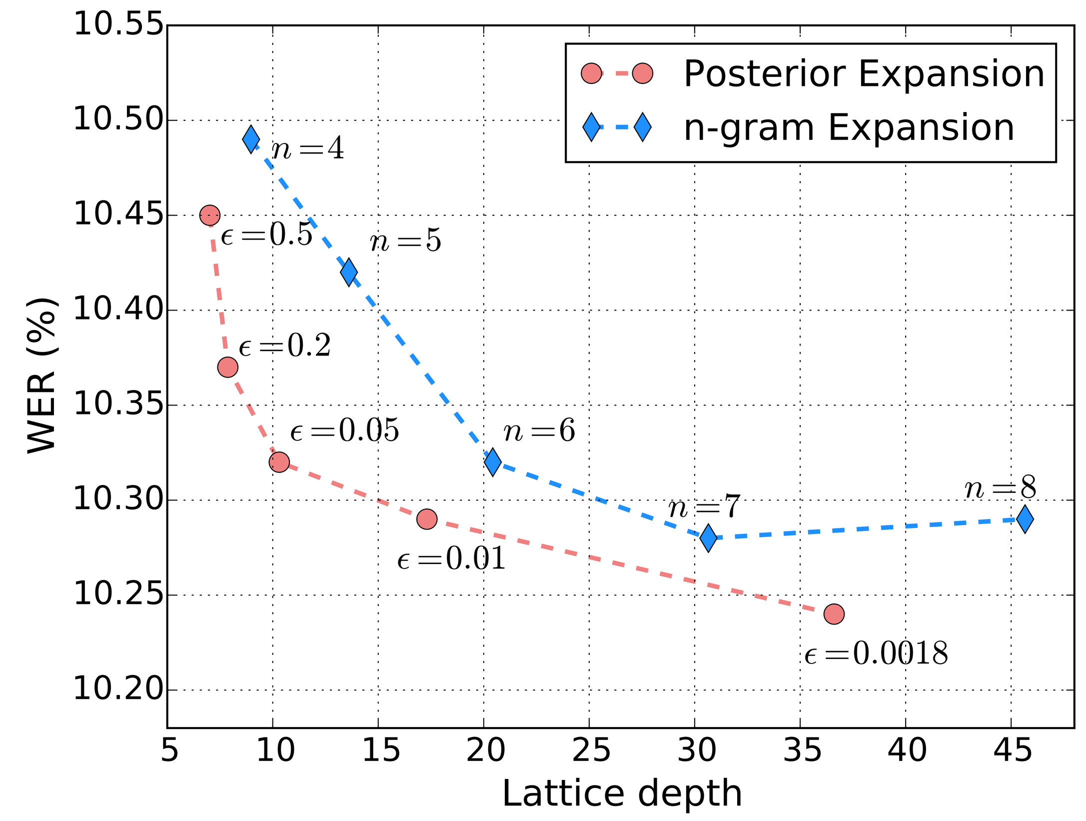

Figure 1: WERs and lattice depths for different ϵ values and *n*-gram orders.

- Posterior-based expansion generates more compact lattices with better recognition accuracy than *n*-gram expansion

Comparison with Pruned Lattice Rescoring

Method	WER			Lattice Depth
	Hub5'00	Swb	Callhm	
20-best	11.3	7.5	15.0	-
Pruned (4-gram approx.)	11.2	**7.3**	15.0	15.1
Non-iterative (ϵ = 0.5)	**11.1**	7.4	**14.9**	**6.4**

Table 3: WERs and lattice depths from pruned lattice rescoring and the proposed non-iterative lattice rescoring.

- The non-iterative rescoring strategy obtains competitive performance and generates smaller lattices

WERs on SWBD

Method	Hub5'00	Swb	Callhm
4-gram KN	12.8	8.6	17.0
20-best (LSTM)	10.9	7.1	14.6
20-best (Transformer)	10.8	7.2	14.4
Non-iterative (ϵ = 0.005)	10.4	6.8	14.0
Iterative (ϵ = 0.001)	**10.2**	**6.5**	**13.9**

Table 4: WERs from proposed lattice rescoring strategies with a Transformer LM.

Conclusions

- Lattice-to-list conversion enables parallel LM evaluations within a lattice and fully takes advantage of the parallel computation cross words of Transformer LMs for speedup.
- Posterior-based lattice expansion outperforms *n*-gram expansion.
- The proposed rescoring strategy makes it easier and more flexible to perform lattice rescoring with PyTorch LMs in Kaldi `kaldi/egs/swbd/s5c/local/pytorchnn/`

ICASSP 2021, June 6-11 — Toronto, Canada

E-Mail: kli26@jhu.edu