LF-MMI training and decoding in k2 (Part II)

Fangjun Kuang

July 20, 2021

Contents

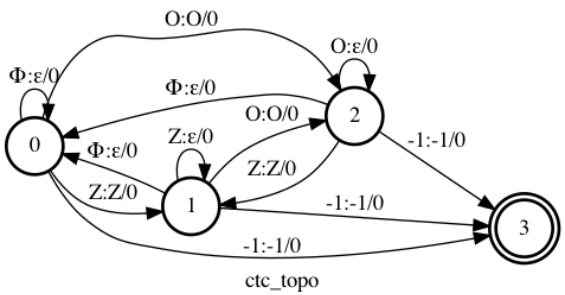
Introduction

- We described LF-MMI training with k2 in the last meeting
- Will talk about decoding in k2

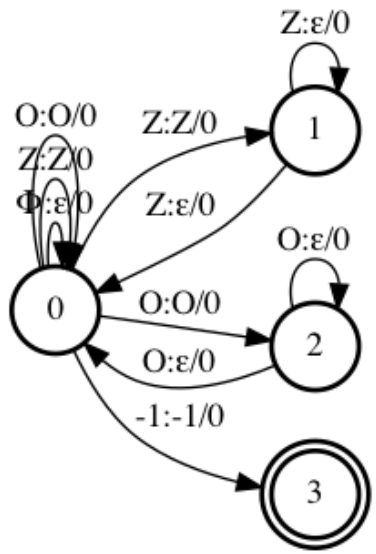
Contents

H - HLG (1/2)

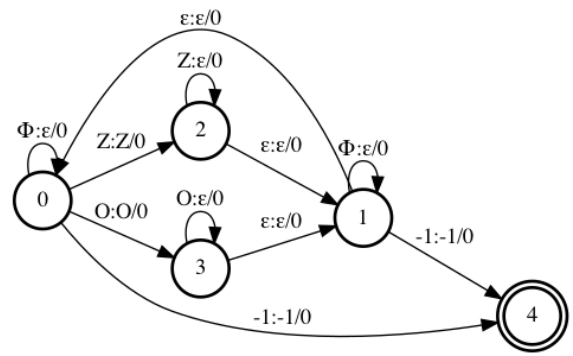
- H is the CTC topology
  - Merge repeated contiguous symbols
- We talked about the following three kinds of topologies in the last meeting
  - Pros and cons ?



ctc_topo



ctc_topo_modified



ctc_topo from EESEN

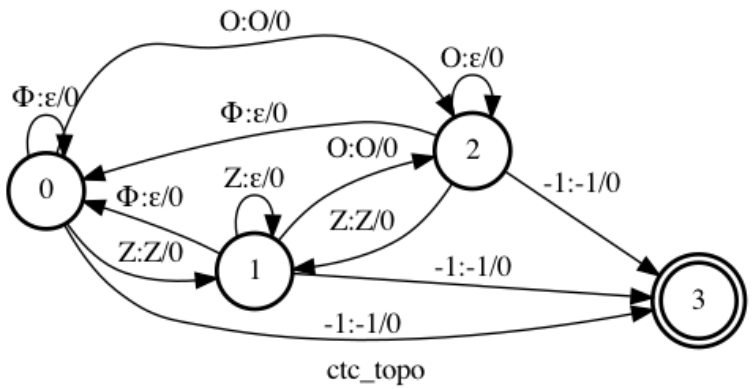
- You can do decoding with only H (i.e., without LG)

H - HLG (2/2)

- H in snowfall

```python
def build_ctc_topo(tokens: List[int]) -> k2.Fsa:
    '''Build CTC topology.
    A token which appears once on the right side (i.e. olabels) may
    appear multiple times on the left side (ilabels), possibly with
    epsilons in between.
    When 0 appears on the left side, it represents the blank symbol;
    when it appears on the right side, it indicates an epsilon. That
    is, 0 has two meanings here.
    Args:
      tokens:
        A list of tokens, e.g., phones, characters, etc.
    Returns:
      Returns an FST that converts repeated tokens to a single token.
    '''
    assert 0 in tokens, 'We assume 0 is ID of the blank symbol'
```

https://github.com/k2-fsa/snowfall/blob/master/snowfall/training/ctc_graph.py#L13



L - HLG

- L is the lexicon **FST**
  - It uses the `prepare_lang.sh` from Kaldi
  - Its output is in **OpenFST format**
  - k2 will convert it **automagically**

```
374  local/make_lexicon_fst.py $grammar_opts \
375    --sil-prob=$sil_prob --sil-phone=$silphone --sil-disambig='#'$ndisambig \
376    $tmpdir/lexiconp_disambig.txt | \
377    local/sym2int.pl -f 3 $dir/phones.txt | \
378    local/sym2int.pl -f 4 $dir/words.txt | \
379    local/fstaddselfloops.pl $wdisambig_phone $wdisambig_word > $dir/L_disambig.fst.txt || exit 1;
```

https://github.com/k2-fsa/snowfall/blob/master/egs/librispeech/asr/simple_v1/local/prepare_lang.sh#L374

G - HLG (1/2)

- G is an n-gram language model
  - 3-gram for decoding
  - 4-gram for rescoring

```
-rw-r--r-- 1 kuangfangjun root 4.1G Apr 23 14:40 lm_fglarge.arpa
-rw-r--r-- 1 kuangfangjun root  94M Apr 23 14:39 lm_tgmed.arpa
```

Size of the LM. tg - trigram, fg - four gram

- You can use kaldi's `arpa2fst` to convert an arpa formatted n-gram G to an OpenFST style FST. (We have wrapped it to Python, i.e., no Kaldi dependencies)

G - HLG (2/2)

- `pip install kaldilm`

```
55    python3 -m kaldilm \
56      --read-symbol-table="data/lang_nosp/words.txt" \
57      --disambig-symbol='#0' \
58      --max-order=4 \
59      data/local/lm/lm_fglarge.arpa >data/lang_nosp/G_4_gram.fst.txt
```

https://github.com/k2-fsa/snowfall/blob/master/egs/librispeech/asr/simple_v1/run.sh#L78

HLG

- We provide a function to get `HLG` from `H`, `L`, and `G`

```
 8 def compile_HLG(
 9     L: Fsa,
10     G: Fsa,
11     H: Fsa,
12     labels_disambig_id_start: int,
13     aux_labels_disambig_id_start: int
14 ) -> Fsa:
```

https://github.com/k2-fsa/snowfall/blob/master/snowfall/decoding/graph.py#L8

- You can also reuse the existing graph, e.g., from Kaldi
  - **CAUTION**: transition ID vs pdf ID

HLG (Summary)

1. What are H, L, and G in HLG ?
2. How to build H, L, G, and HLG ?

Contents

Decoding in Kaldi (1/3)

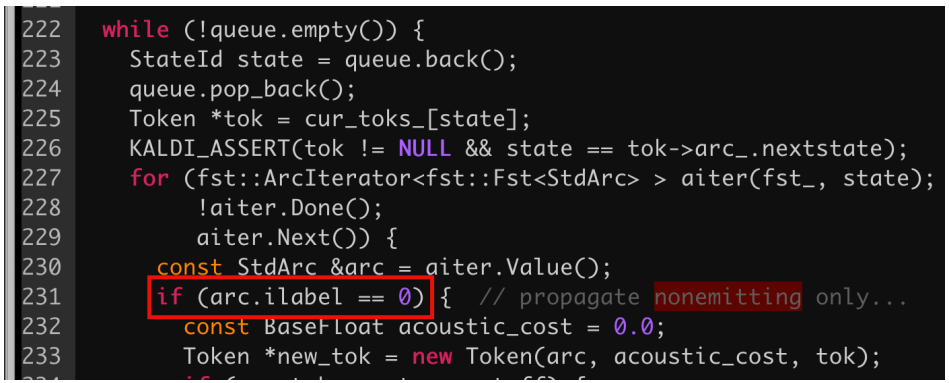
- Let us first revisit the decoding procedure in Kaldi using its `simpile-decoder.cc`

```
66  while (num_frames_decoded_ < target_frames_decoded) {
67    // note: ProcessEmitting() increments num_frames_decoded_
68    ClearToks(prev_toks_);
69    cur_toks_.swap(prev_toks_);
70    ProcessEmitting(decodable); 1
71    ProcessNonemitting(); 2
72    PruneToks(beam_, &cur_toks_); 3
73  }
74 }
```

https://github.com/kaldi-asr/kaldi/blob/master/src/decoder/simple-decoder.cc#L66

Decoding in Kaldi (2/3)

- ProcessNonemitting - epsilon transitions



```
222  while (!queue.empty()) {
223    StateId state = queue.back();
224    queue.pop_back();
225    Token *tok = cur_toks_[state];
226    KALDI_ASSERT(tok != NULL && state == tok->arc_.nextstate);
227    for (fst::ArcIterator<fst::Fst<StdArc> > aiter(fst_, state);
228         !aiter.Done();
229         aiter.Next()) {
230      const StdArc &arc = aiter.Value();
231      if (arc.ilabel == 0) {  // propagate nonemitting only...
232        const BaseFloat acoustic_cost = 0.0;
233        Token *new_tok = new Token(arc, acoustic_cost, tok);
```

https://github.com/kaldi-asr/kaldi/blob/master/src/decoder/simple-decoder.cc#L222

Decoding in Kaldi (3/3)

- ProcessEmitting - non-epsilon transitions, taking into account the neural network output

```
182      if (arc.ilabel != 0) {  // propagate..
183        BaseFloat acoustic_cost = -decodable->LogLikelihood(frame, arc.ilabel);
184        double total_cost = tok->cost_ + arc.weight.Value() + acoustic_cost;
185
```

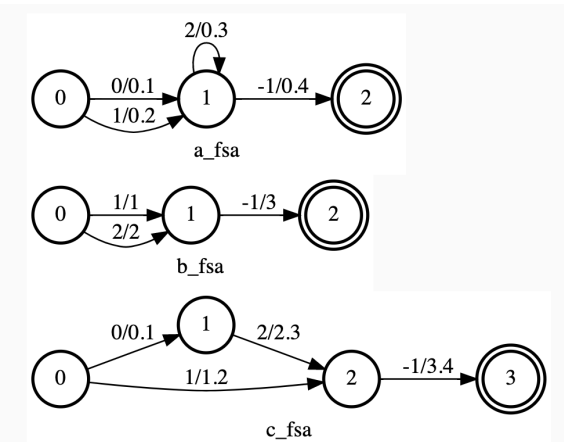
https://github.com/kaldi-asr/kaldi/blob/master/src/decoder/simple-decoder.cc#L182

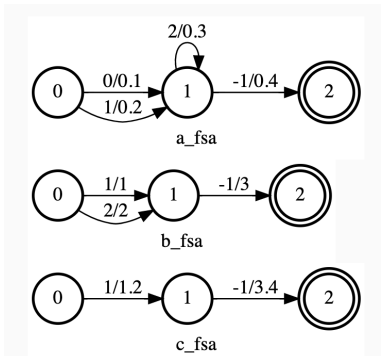
- Note
  - `acoustic cost` is from the neural network (am score)
  - `arc.weight.value` is from HLG (lm score)

Contents

Decoding in k2 (1/3)

- k2 implements the above decoding process by FSA intersections
  - However, there is **no** `ProcessNonemitting()`
  - We treat epsilon as a normal symbol by adding epsilon self-loops





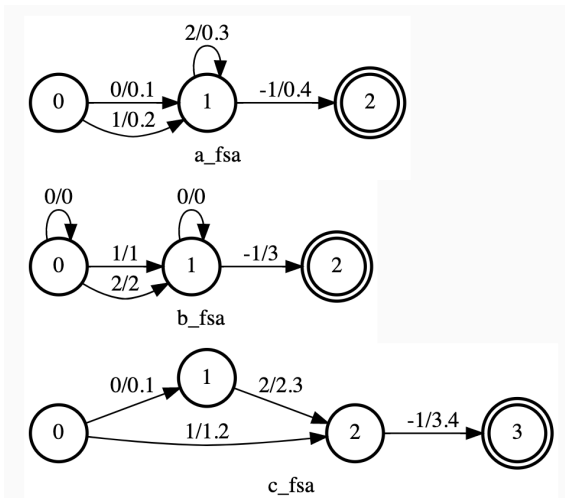


Table: c_fsa is the intersection of a_fsa and b_fsa. **Left**: Treat epsilon specially. **Middle**: Treat epsilon as a normal symbol without adding epsilon self-loops (Result is incorrect). **Right**: Treat epsilon as a normal symbol by adding epsilon self-loops.

Decoding in k2 (2/3)

- Decoding in k2. The returned lattice is also an FSA

```
lattices = k2.intersect_dense_pruned(HLG, dense_fsa_vec, 20.0, output_beam_size, 30, 10000)
```

- We can replace HLG with H, without any extra effort.
- It supports batch decoding and runs on GPU
  - i.e., decoding multiple waves in parallel

Decoding in k2 (2/3)

- Decoding in k2. The returned lattice is also an FSA

```
lattices = k2.intersect_dense_pruned(HLG, dense_fsa_vec, 20.0, output_beam_size, 30, 10000)
```

- We can replace HLG with H, without any extra effort.
- It supports batch decoding and runs on GPU
  - i.e., decoding multiple waves in parallel
- **How** to get the decoding result from the decoding lattice ?

Decoding in k2 (3/3)

- There are several techniques:
  1. Use the 1-best path from the lattice, `k2.shortest_path()`
  2. Extract n paths from the lattice and rescore it with a larger LM
     - n-best rescoring
  3. Rescore the lattice with a larger LM and get the 1-best path from the resulting lattice
     - whole lattice rescoring
  4. Rescore it with a larger LM, extract n paths from it and rescore them with
     - an transformer attention decoder
     - a Transformer LM

- We provide implementations for above techniques in snowfall (transformer LM is still an on-going work by Liyong)

Contents

Summary

- HLG and how it is created in k2
- Reviewed the simple decoder from kaldi
  - process emitting, process non-emitting, prunning
- The decoding techniques supported by k2

Thank you!