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

Fangjun Kuang

June 22, 2021

Contents

Introduction

- Will describe LF-MMI training and decoding using k2
- For the decoding part
  - 1-best decoding
  - n-best decoding
  - (n-gram) LM rescoring
- LF-MMI training shares many things in common with CTC training
  - We first describe CTC training
- All happens in the framework of finite state machines

Introduction

- Will describe LF-MMI training and decoding using k2
- For the decoding part in Part II in the **next meeting**
  - 1-best decoding
  - n-best decoding
  - (n-gram) LM rescoring
- LF-MMI training shares many things in common with CTC training
  - We first describe CTC training
- All happens in the framework of finite state machines
- Part I (this Part) focuses on **training**
- Part II is about **decoding**

Introduction (Continued)

- All happens in the framework of finite state machines

Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks

Alex Graves[1] ALEX@IDSIA.CH
Santiago Fernández[1] SANTIAGO@IDSIA.CH
Faustino Gomez[1] TINO@IDSIA.CH
Jürgen Schmidhuber[1,2] JUERGEN@IDSIA.CH
[1] Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA), Galleria 2, 6928 Manno-Lugano, Switzerland
[2] Technische Universität München (TUM), Boltzmannstr. 3, 85748 Garching, Munich, Germany

(a) CTC was first proposed in this paper

This gives us the following rules for initialisation

$$\alpha_1(1) = y_b^1$$
$$\alpha_1(2) = y_{\mathbf{l}_1}^1$$
$$\alpha_1(s) = 0, \ \forall s > 2$$

and recursion

$$\alpha_t(s) = \begin{cases} \bar{\alpha}_t(s) y_{\mathbf{l}'_s}^t & \text{if } \mathbf{l}'_s = b \text{ or } \mathbf{l}'_{s-2} = \mathbf{l}'_s \\ \left(\bar{\alpha}_t(s) + \alpha_{t-1}(s-2)\right) y_{\mathbf{l}'_s}^t & \text{otherwise} \end{cases} \tag{6}$$

where

$$\bar{\alpha}_t(s) \stackrel{\text{def}}{=} \alpha_{t-1}(s) + \alpha_{t-1}(s-1). \tag{7}$$

(b) Equations used in CTC (extracted from the above paper)

Introduction (Continued)

- All happens in the framework of **finite state machines**

Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks

Alex Graves[1] ALEX@IDSIA.CH
Santiago Fernández[1] SANTIAGO@IDSIA.CH
Faustino Gomez[1] TINO@IDSIA.CH
Jürgen Schmidhuber[1,2] JUERGEN@IDSIA.CH
[1] Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA), Galleria 2, 6928 Manno-Lugano, Switzerland
[2] Technische Universität München (TUM), Boltzmannstr. 3, 85748 Garching, Munich, Germany

(a) CTC was first proposed in this paper

This gives us the following rules for initialisation

$$\alpha_1(1) = y_b^1$$
$$\alpha_1(2) = y_{\mathbf{l}_1}^1$$
$$\alpha_1(s) = 0, \ \forall s > 2$$

and recursion

$$\alpha_t(s) = \begin{cases} \bar{\alpha}_t(s) y_{\mathbf{l}'_s}^t & \text{if } \mathbf{l}'_s = b \text{ or } \mathbf{l}'_{s-2} = \mathbf{l}'_s \\ \left(\bar{\alpha}_t(s) + \alpha_{t-1}(s-2)\right) y_{\mathbf{l}'_s}^t & \text{otherwise} \end{cases} \tag{6}$$

where

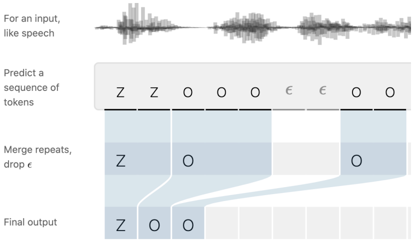
$$\bar{\alpha}_t(s) \stackrel{\text{def}}{=} \alpha_{t-1}(s) + \alpha_{t-1}(s-1). \tag{7}$$

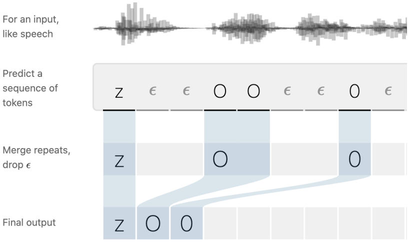
(b) We replace them with FSA operations

Contents

CTC Training

- In CTC training, we need to
  - Merge **repeated contiguous** symbols and drop blanks within a sequence
    - Example: Z Z O O O ϵ ϵ O O $\longrightarrow$ Z O O
    - Example: Z ϵ ϵ O O ϵ ϵ O ϵ $\longrightarrow$ Z O O
  - **Sum** the probabilities of **all correct** sequences



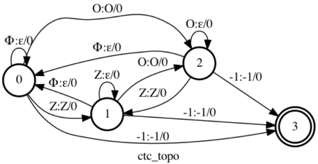


CTC Training (Continued)

- In the following, we show how to
  - Merge repeated contiguous symbols
  - Find all the correct sequences
  - Compute the sum of the probabilities of all the correct sequences

in k2 through FSA operations

CTC Topology

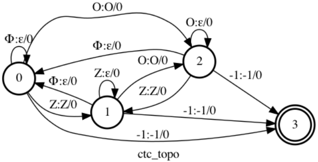


(a) `ctc_topo` that merges repeated contiguous symbols (Φ is the input blank sybmol)

- Input: `Z Z O O O Φ Φ O O`
- Output: `Z` ϵ `O` ϵ ϵ ϵ ϵ `O` ϵ

time	cur_state	next_state	input	output
0	0	1	Z	Z
1	1	1	Z	ϵ
2	1	2	O	O
3	2	2	O	ϵ
4	2	2	O	ϵ
5	2	0	Φ	ϵ
6	0	0	Φ	ϵ
7	0	2	O	O
8	2	2	O	ϵ
9	2	3	EOF	accepted

CTC Topology (Continued)



(a) `ctc_topo` that merges repeated contiguous symbols (Φ is the input blank sybmol.).

- Input: `Z Φ Φ O O Φ Φ O Φ`
- Output: `Z` ϵ ϵ `O` ϵ ϵ ϵ `O` ϵ

time	cur_state	next_state	input	output
0	0	1	Z	Z
1	1	0	Φ	ϵ
2	0	0	Φ	ϵ
3	0	2	O	O
4	2	2	O	ϵ
5	2	0	Φ	ϵ
6	0	0	Φ	ϵ
7	0	2	O	O
8	2	0	Φ	ϵ
9	0	3	EOF	accepted

CTC Training (Continued)

- In the following, we show how to
  - Merge repeated contiguous symbols ✓
  - Find all the correct sequences
  - Compute the sum of the probabilities of all the correct sequences

  in k2 through FSA operations

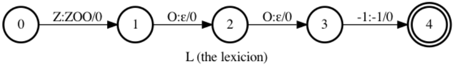
CTC Training (Continued)

- In the following, we show how to
  - Merge repeated contiguous symbols ✓
  - Find all the correct sequences
  - Compute the sum of the probabilities of all the correct sequences

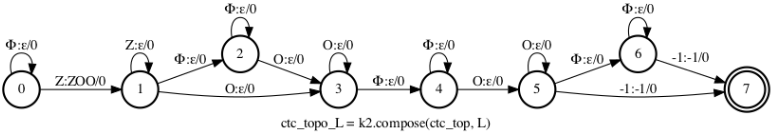
  in k2 through FSA operations

Find All Correct Sequences

- First, let us introduce a lexicon to constrain the possible paths



(a) Lexicon `L` that converts input sequence `Z O O` to `ZOO`.



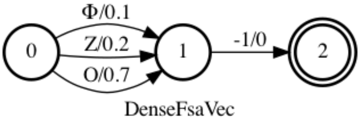
(b) `ctc_topo_L`, the composition of `ctc_topo` and `L`
(A path in it is **always** correct, i.e., transduces to `ZOO`;
It also merges repeated contiguous symbols!)

Find All Correct Sequences (Continued)

- Second, convert the neural network output to an FSA.
  - Assume there are **five** frames

```
nnet_output = torch.tensor(
    [[0.1, 0.2, 0.7],
     [0.3, 0.4, 0.3],
     [0.8, 0.1, 0.1],
     [0.2, 0.2, 0.6],
     [0.9, 0.08, 0.02],
    ]).requires_grad_(True)
```

  - The first frame contains $(0.1, 0.2, 0.7)$. Suppose 0.1 is the probability of the blank Φ, 0.2 the probability of symbol Z, and 0.7 the probability of symbol O



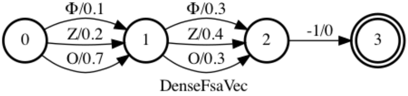
(a) Convert the output of frame 1 to a DenseFsa

Find All Correct Sequences (Continued)

- Second, convert the neural network output to an FSA.
  - Assume there are **five** frames

```
nnet_output = torch.tensor(
    [[0.1, 0.2, 0.7],
     [0.3, 0.4, 0.3],
     [0.8, 0.1, 0.1],
     [0.2, 0.2, 0.6],
     [0.9, 0.08, 0.02],
    ]).requires_grad_(True)
```

  - The first frame contains $(0.1, 0.2, 0.7)$. Suppose 0.1 is the probability of the blank Φ, 0.2 the probability of symbol `Z`, and 0.7 the probability of symbol `O`



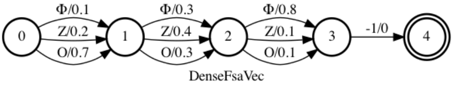
(a) Convert the output of frames 1–2 to a DenseFsa

Find All Correct Sequences (Continued)

- Second, convert the neural network output to an FSA.
  - Assume there are **five** frames

```
nnet_output = torch.tensor(
    [[0.1, 0.2, 0.7],
     [0.3, 0.4, 0.3],
     [0.8, 0.1, 0.1],
     [0.2, 0.2, 0.6],
     [0.9, 0.08, 0.02],
    ]).requires_grad_(True)
```

  - The first frame contains $(0.1, 0.2, 0.7)$. Suppose 0.1 is the probability of the blank Φ, 0.2 the probability of symbol `Z`, and 0.7 the probability of symbol `O`



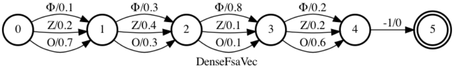
(a) Convert the output of frames 1–3 to a DenseFsa

Find All Correct Sequences (Continued)

- Second, convert the neural network output to an FSA.
  - Assume there are **five** frames

```
nnet_output = torch.tensor(
    [[0.1, 0.2, 0.7],
     [0.3, 0.4, 0.3],
     [0.8, 0.1, 0.1],
     [0.2, 0.2, 0.6],
     [0.9, 0.08, 0.02],
    ]).requires_grad_(True)
```

  - The first frame contains $(0.1, 0.2, 0.7)$. Suppose 0.1 is the probability of the blank Φ, 0.2 the probability of symbol `Z`, and 0.7 the probability of symbol `O`



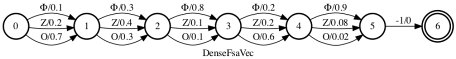
(a) Convert the output of frames 1–4 to a DenseFsa

Find All Correct Sequences (Continued)

- Second, convert the neural network output to an FSA.
  - Assume there are **five** frames

```
nnet_output = torch.tensor(
    [[0.1, 0.2, 0.7],
     [0.3, 0.4, 0.3],
     [0.8, 0.1, 0.1],
     [0.2, 0.2, 0.6],
     [0.9, 0.08, 0.02],
    ]).requires_grad_(True)
```

  - The first frame contains $(0.1, 0.2, 0.7)$. Suppose 0.1 is the probability of the blank Φ, 0.2 the probability of symbol Z, and 0.7 the probability of symbol O



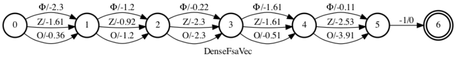
(a) Convert the output of frames 1–5 to a DenseFsa

Find All Correct Sequences (Continued)

- Second, convert the neural network output to an FSA.
  - Assume there are **five** frames

```
nnet_output = torch.tensor(
    [[0.1, 0.2, 0.7],
     [0.3, 0.4, 0.3],
     [0.8, 0.1, 0.1],
     [0.2, 0.2, 0.6],
     [0.9, 0.08, 0.02],
    ]).requires_grad_(True)
```

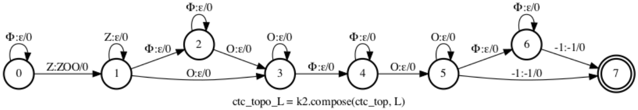
  - The first frame contains $(0.1, 0.2, 0.7)$. Suppose 0.1 is the probability of the blank Φ, 0.2 the probability of symbol Z, and 0.7 the probability of symbol O



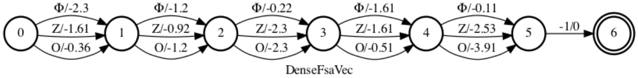
(a) Convert the output of frames 1–5 to a DenseFsa in **log** space

- **HINT:** $\log(0.1) = -2.3$, $\log(0.2) = -1.61$, $\log(0.7) = -0.36$

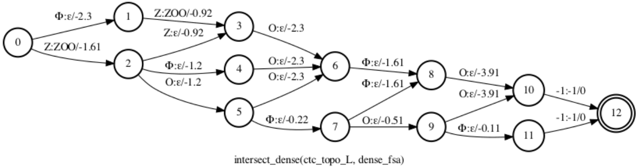
Find All Correct Sequences (Continued)



(a) `ctc_topo_L`, the result of `compose(ctc_topo, L)`



(b) `DenseFsa`, from the neural network output



(c) The decoding lattice, the result of `intersect(ctc_topo_L, DenseFsa)`.

- The decoding lattice contains **all** the **correct** paths (sequences).

CTC Training (Continued)

- In the following, we show how to
  - Merge repeated contiguous symbols ✓
  - Find all the correct sequences ✓
  - Compute the sum of the probabilities of all the correct sequences

  in k2 through FSA operations

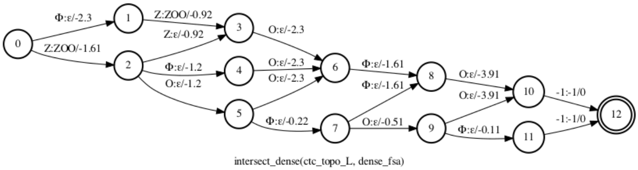
CTC Training (Continued)

- In the following, we show how to
  - Merge repeated contiguous symbols ✓
  - Find all the correct sequences ✓
  - Compute the sum of the probabilities of all the correct sequences
    - Also known as total scores

  in k2 through FSA operations

Compute Total Scores

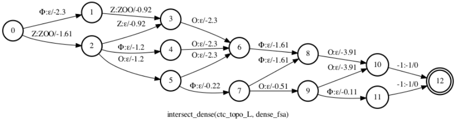
- Compute the sum of the probabilities of all the correct sequences
  - Using the decoding lattice via the forward algorithm in log-semiring



(a) The decoding lattice, the result of `intersect(ctc_topo_L, DenseFsa)`.

- Note the decoding lattice is topologically sorted
  - If not, `k2.top_sort()` can ensure that

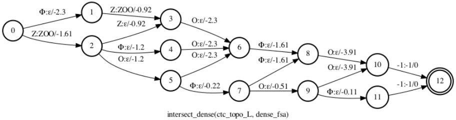
Compute Total Scores (Continued)



a) The decoding lattice, the result of `intersect(ctc_topo_L, DenseFsa)`.

`state`	`forward_score`	description
0	0	score of start state is always 0

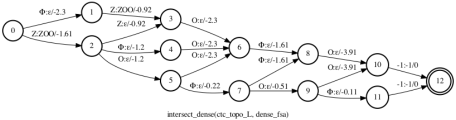
Compute Total Scores (Continued)



a) The decoding lattice, the result of `intersect(ctc_topo_L, DenseFsa)`.

`state`	`forward_score`	description
0	0	score of start state is always 0
1	-2.3	

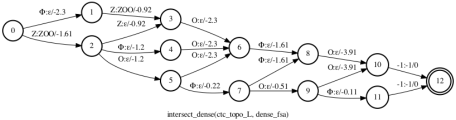
Compute Total Scores (Continued)



a) The decoding lattice, the result of `intersect(ctc_topo_L, DenseFsa)`.

`state`	`forward_score`	description
0	0	score of start state is always 0
1	-2.3	
2	-1.61	

Compute Total Scores (Continued)

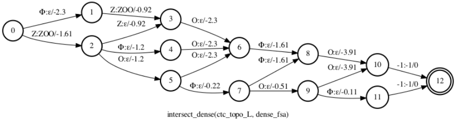


a) The decoding lattice, the result of `intersect(ctc_topo_L, DenseFsa)`.

`state`	`forward_score`	description
0	0	score of start state is always 0
1	-2.3	
2	-1.61	
3	-2.12	$\log(e^{-2.3-0.92} + e^{-1.61-0.92})$ `log_add`

- $\text{log\_add}(a, b) = \log(e^a + e^b)$
- $\text{log\_add}(a, b, c) = \text{log\_add}(\text{log\_add}(a, b), c) = \log(e^a + e^b + e^c)$

Compute Total Scores (Continued)

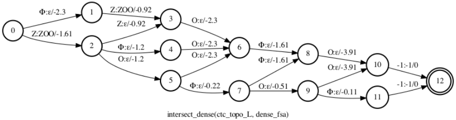


a) The decoding lattice, the result of `intersect(ctc_topo_L, DenseFsa)`.

`state`	`forward_score`	description
0	0	score of start state is always 0
1	-2.3	
2	-1.61	
3	-2.12	$\log(\mathrm{e}^{-2.3-0.92} + \mathrm{e}^{-1.61-0.92})$ `log_add`
4	-2.81	$-1.61 - 1.2$

- $\text{log\_add}(a, b) = \log(\mathrm{e}^a + \mathrm{e}^b)$
- $\text{log\_add}(a, b, c) = \text{log\_add}(\text{log\_add}(a, b), c) = \log(\mathrm{e}^a + \mathrm{e}^b + \mathrm{e}^c)$

Compute Total Scores (Continued)

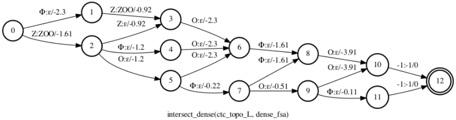


a) The decoding lattice, the result of `intersect(ctc_topo_L, DenseFsa)`.

`state`	`forward_score`	description
0	0	score of start state is always 0
1	-2.3	
2	-1.61	
3	-2.12	$\log(\mathrm{e}^{-2.3-0.92} + \mathrm{e}^{-1.61-0.92})$ `log_add`
4	-2.81	$-1.61 - 1.2$
5	-2.81	$-1.61 - 1.2$

- $\mathrm{log\_add}(a, b) = \log(\mathrm{e}^a + \mathrm{e}^b)$
- $\mathrm{log\_add}(a, b, c) = \mathrm{log\_add}(\mathrm{log\_add}(a, b), c) = \log(\mathrm{e}^a + \mathrm{e}^b + \mathrm{e}^c)$

Compute Total Scores (Continued)

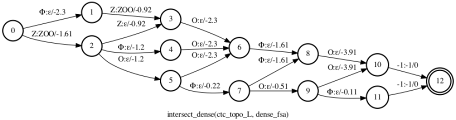


a) The decoding lattice, the result of `intersect(ctc_topo_L, DenseFsa)`.

`state`	`forward_score`	description
0	0	score of start state is always 0
1	-2.3	
2	-1.61	
3	-2.12	$\log(\mathrm{e}^{-2.3-0.92} + \mathrm{e}^{-1.61-0.92})$ `log_add`
4	-2.81	$-1.61 - 1.2$
5	-2.81	$-1.61 - 1.2$
6	-3.73	$\log(\mathrm{e}^{-2.12-2.3} + \mathrm{e}^{-2.81-2.3} + \mathrm{e}^{-2.81-2.3})$

- $\mathrm{log\_add}(a, b) = \log(\mathrm{e}^{a} + \mathrm{e}^{b})$
- $\mathrm{log\_add}(a, b, c) = \mathrm{log\_add}(\mathrm{log\_add}(a, b), c) = \log(\mathrm{e}^{a} + \mathrm{e}^{b} + \mathrm{e}^{c})$

Compute Total Scores (Continued)

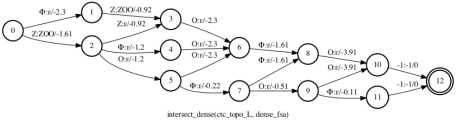


a) The decoding lattice, the result of `intersect(ctc_topo_L, DenseFsa)`.

`state`	`forward_score`	description
0	0	score of start state is always 0
1	-2.3	
2	-1.61	
3	-2.12	$\log(\mathrm{e}^{-2.3-0.92}+\mathrm{e}^{-1.61-0.92})$ `log_add`
4	-2.81	$-1.61-1.2$
5	-2.81	$-1.61-1.2$
6	-3.73	$\log(\mathrm{e}^{-2.12-2.3}+\mathrm{e}^{-2.81-2.3}+\mathrm{e}^{-2.81-2.3})$
7	-3.03	$-2.81-0.22$

- $\log\_\mathrm{add}(a, b) = \log(\mathrm{e}^a + \mathrm{e}^b)$
- $\log\_\mathrm{add}(a, b, c) = \log\_\mathrm{add}(\log\_\mathrm{add}(a, b), c) = \log(\mathrm{e}^a + \mathrm{e}^b + \mathrm{e}^c)$

Compute Total Scores (Continued)

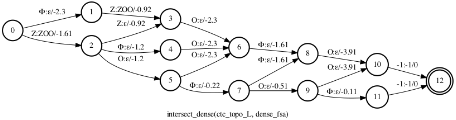


a) The decoding lattice, the result of `intersect(ctc_topo_L, DenseFsa)`.

`state`	`forward_score`	description
0	0	score of start state is always 0
1	-2.3	
2	-1.61	
3	-2.12	$\log(e^{-2.3-0.92} + e^{-1.61-0.92})$ log_add
4	-2.81	$-1.61 - 1.2$
5	-2.81	$-1.61 - 1.2$
6	-3.73	$\log(e^{-2.12-2.3} + e^{-2.81-2.3} + e^{-2.81-2.3})$
7	-3.03	$-2.81 - 0.22$
8	-4.24	$\log(e^{-3.73-1.61} + e^{-3.03-1.61})$

- $\text{log\_add}(a, b) = \log(e^a + e^b)$
- $\text{log\_add}(a, b, c) = \text{log\_add}(\text{log\_add}(a, b), c) = \log(e^a + e^b + e^c)$

Compute Total Scores (Continued)

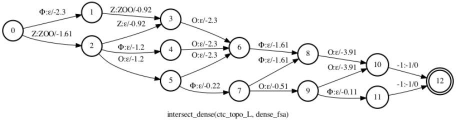


a) The decoding lattice, the result of `intersect(ctc_topo_L, DenseFsa)`.

`state`	`forward_score`	description
0	0	score of start state is always 0
1	-2.3	
2	-1.61	
3	-2.12	$\log(e^{-2.3-0.92} + e^{-1.61-0.92})$ `log_add`
4	-2.81	$-1.61 - 1.2$
5	-2.81	$-1.61 - 1.2$
6	-3.73	$\log(e^{-2.12-2.3} + e^{-2.81-2.3} + e^{-2.81-2.3})$
7	-3.03	$-2.81 - 0.22$
8	-4.24	$\log(e^{-3.73-1.61} + e^{-3.03-1.61})$
9	-3.54	$-3.03 - 0.51$

- $\text{log\_add}(a, b) = \log(e^a + e^b)$
- $\text{log\_add}(a, b, c) = \text{log\_add}(\text{log\_add}(a, b), c) = \log(e^a + e^b + e^c)$

Compute Total Scores (Continued)

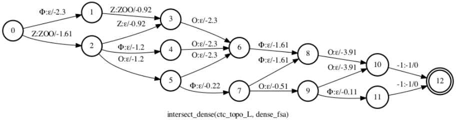


a) The decoding lattice, the result of `intersect(ctc_topo_L, DenseFsa)`.

`state`	`forward_score`	description
0	0	score of start state is always 0
1	-2.3	
2	-1.61	
3	-2.12	$\log(e^{-2.3-0.92} + e^{-1.61-0.92})$ `log_add`
4	-2.81	$-1.61 - 1.2$
5	-2.81	$-1.61 - 1.2$
6	-3.73	$\log(e^{-2.12-2.3} + e^{-2.81-2.3} + e^{-2.81-2.3})$
7	-3.03	$-2.81 - 0.22$
8	-4.24	$\log(e^{-3.73-1.61} + e^{-3.03-1.61})$
9	-3.54	$-3.03 - 0.51$
10	-7.05	$\log(e^{-4.24-3.91} + e^{-3.54-3.91})$

- $\text{log\_add}(a, b) = \log(e^a + e^b)$
- $\text{log\_add}(a, b, c) = \text{log\_add}(\text{log\_add}(a, b), c) = \log(e^a + e^b + e^c)$

Compute Total Scores (Continued)

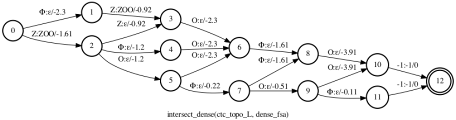


a) The decoding lattice, the result of `intersect(ctc_topo_L, DenseFsa)`.

`state`	`forward_score`	description
0	0	score of start state is always 0
1	-2.3	
2	-1.61	
3	-2.12	$\log(e^{-2.3-0.92} + e^{-1.61-0.92})$ `log_add`
4	-2.81	$-1.61 - 1.2$
5	-2.81	$-1.61 - 1.2$
6	-3.73	$\log(e^{-2.12-2.3} + e^{-2.81-2.3} + e^{-2.81-2.3})$
7	-3.03	$-2.81 - 0.22$
8	-4.24	$\log(e^{-3.73-1.61} + e^{-3.03-1.61})$
9	-3.54	$-3.03 - 0.51$
10	-7.05	$\log(e^{-4.24-3.91} + e^{-3.54-3.91})$
11	-3.65	$-3.54 - 0.11$

- $\text{log\_add}(a, b) = \log(e^a + e^b)$
- $\text{log\_add}(a, b, c) = \text{log\_add}(\text{log\_add}(a, b), c) = \log(e^a + e^b + e^c)$

Compute Total Scores (Continued)

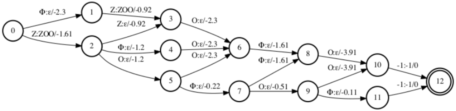


a) The decoding lattice, the result of `intersect(ctc_topo_L, DenseFsa)`.

`state`	`forward_score`	description
0	0	score of start state is always 0
1	-2.3	
2	-1.61	
3	-2.12	$\log(e^{-2.3-0.92} + e^{-1.61-0.92})$ `log_add`
4	-2.81	$-1.61 - 1.2$
5	-2.81	$-1.61 - 1.2$
6	-3.73	$\log(e^{-2.12-2.3} + e^{-2.81-2.3} + e^{-2.81-2.3})$
7	-3.03	$-2.81 - 0.22$
8	-4.24	$\log(e^{-3.73-1.61} + e^{-3.03-1.61})$
9	-3.54	$-3.03 - 0.51$
10	-7.05	$\log(e^{-4.24-3.91} + e^{-3.54-3.91})$
11	-3.65	$-3.54 - 0.11$
12	-3.62	$\log(e^{-7.05} + e^{-3.65})$

- $\mathrm{log\_add}(a, b) = \log(e^a + e^b)$
- $\mathrm{log\_add}(a, b, c) = \mathrm{log\_add}(\mathrm{log\_add}(a, b), c) = \log(e^a + e^b + e^c)$

Compute Total Scores (Continued)



intersect_dense(ctc_topo_L, dense_fsa)

a) The decoding lattice, the result of `intersect(ctc_topo_L, DenseFsa)`.

`state`	`forward_score`	description
0	0	score of start state is always 0
1	-2.3	
2	-1.61	
3	-2.12	$\log(e^{-2.3-0.92} + e^{-1.61-0.92})$ `log_add`
4	-2.81	$-1.61 - 1.2$
5	-2.81	$-1.61 - 1.2$
6	-3.73	$\log(e^{-2.12-2.3} + e^{-2.81-2.3} + e^{-2.81-2.3})$
7	-3.03	$-2.81 - 0.22$
8	-4.24	$\log(e^{-3.73-1.61} + e^{-3.03-1.61})$
9	-3.54	$-3.03 - 0.51$
10	-7.05	$\log(e^{-4.24-3.91} + e^{-3.54-3.91})$
11	-3.65	$-3.54 - 0.11$
12	-3.62	$\log(e^{-7.05} + e^{-3.65})$

- The `forward_score` of the final state (i.e, state 12), is the total score of the lattice
  - That is, the log of the sum of probabilities of all correct paths (sequences)

Compute Total Scores (Continued)

- k2 provides
  - `k2.Fsa.get_forward_scores()`
  - `k2.Fsa.get_total_scores()`
- for log-semiring as well as tropical semiring, in a **differentiable** manner

```
In [75]: lats.get_forward_scores(log_semiring=True, use_double_scores=True)

Out[75]: tensor([ 0.0000, -2.3026, -1.6094, -2.1203, -2.8134, -2.8134, -3.7297, -3.0366,
                 -4.2405, -3.5474, -7.0539, -3.6527, -3.6200], dtype=torch.float64,
                grad_fn=<_GetForwardScoresFunctionBackward>)

In [76]: lats.get_tot_scores(log_semiring=True, use_double_scores=True)

Out[76]: tensor([-3.6200], dtype=torch.float64, grad_fn=<_GetTotScoresFunctionBackward>)
```

(a) Examples of `k2.Fsa.get_forward_scores()` and `k2.Fsa.get_total_scores()`
(**HINT:** It supports **autograd**.)

CTC Training (Continued)

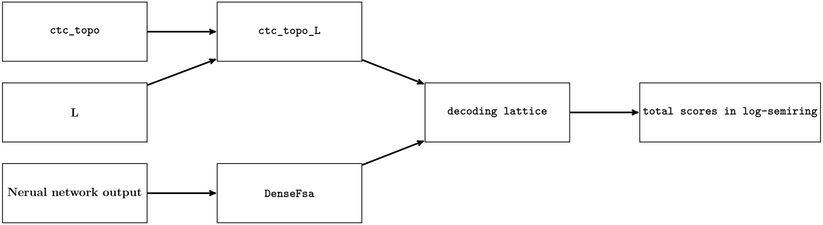
- In the following, we show how to
  - Merge repeated contiguous symbols ✓
  - Find all the correct sequences ✓
  - Compute the sum of the probabilities of all the correct sequences ✓

  in k2 through FSA operations
- The objective function of CTC training is
  - To maximize `lats.get_tot_scores(log_semiring=True)`
- Super easy in PyTorch

```
optimizer.zero_grad()
(-tot_score).backward()
clip_grad_value_(model.parameters(), 5.0)
optimizer.step()
```

(a) See https://github.com/k2-fsa/snowfall/blob/master/egs/librispeech/asr/simple_v1/ctc_train.py#L121

CTC Training (Summary)



(a) CTC training using FSA operations with k2

This gives us the following rules for initialisation

$$\alpha_1(1) = y_b^1$$
$$\alpha_1(2) = y_{\mathbf{l}_1}^1$$
$$\alpha_1(s) = 0, \ \forall s > 2$$

and recursion

$$\alpha_t(s) = \begin{cases} \bar{\alpha}_t(s) y^t_{\mathbf{l}'_s} & \text{if } \mathbf{l}'_s = b \text{ or } \mathbf{l}'_{s-2} = \mathbf{l}'_s \\ \left(\bar{\alpha}_t(s) + \alpha_{t-1}(s-2)\right) y^t_{\mathbf{l}'_s} & \text{otherwise} \end{cases} \tag{6}$$

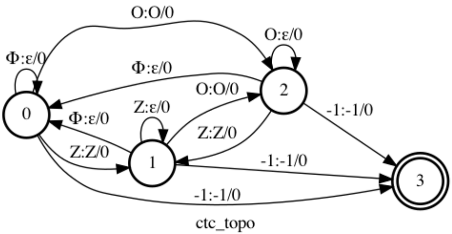
where

$$\bar{\alpha}_t(s) \stackrel{\text{def}}{=} \alpha_{t-1}(s) + \alpha_{t-1}(s-1). \tag{7}$$

(b) No need to know the above equations

ctc_topo Notes

- The topology to merge repeated contiguous symbols is not unique.

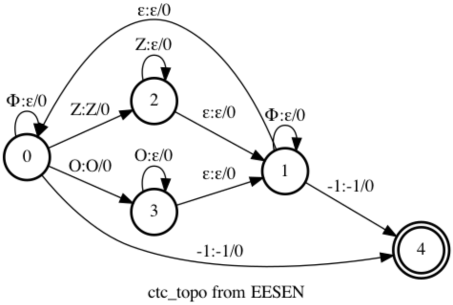


(a) Standard `ctc_topo`

- Assume there are n tokens
  - The number of arcs in the above topology is $\mathcal{O}(n^2)$, i.e., quadratic in n
  - NOTE: There are no epsilon transitions
  - NOTE: It is deterministic

ctc_topo Notes (Continued)

- The topology to merge repeated contiguous symbols is not unique.

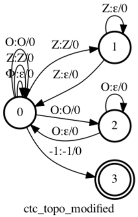


(a) ctc_topo from EESEN `https://github.com/srvk/eesen`

- Assume there are n tokens
  - The number of arcs in the above topology is $\mathcal{O}(n)$, i.e., linear in n
  - NOTE: There are lots of epsilon transitions
  - NOTE: It is deterministic

ctc_topo Notes (Continued)

- The topology to merge repeated contiguous symbols is not unique.

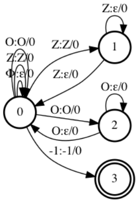


(a) `ctc_topo_modified` from Daniel Povey `https://github.com/k2-fsa/k2/issues/746`

- Assume there are n tokens
  - The number of arcs in the above topology is $\mathcal{O}(n)$, i.e., linear in n
  - NOTE: There are no epsilon transitions
  - NOTE: It is non-deterministic.

ctc_topo Notes (Continued)

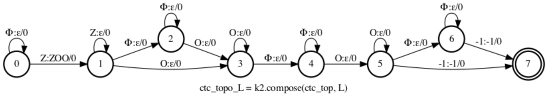
- The topology to merge repeated contiguous symbols is not unique.



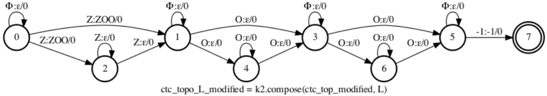
(a) `ctc_topo_modified` from Daniel Povey `https://github.com/k2-fsa/k2/issues/746`

- Assume there are n tokens
  - The number of arcs in the above topology is $\mathcal{O}(n)$, i.e., linear in n
  - NOTE: There are no epsilon transitions
  - NOTE: It is non-deterministic.
  - CAUTION: No mandatory blanks between consecutive repeated symbols
    - See next page for an example

ctc_topo Notes (Continued)



(a) `k2.compose(ctc_topo, L)`



(b) `k2.compose(ctc_topo_modified, L)`

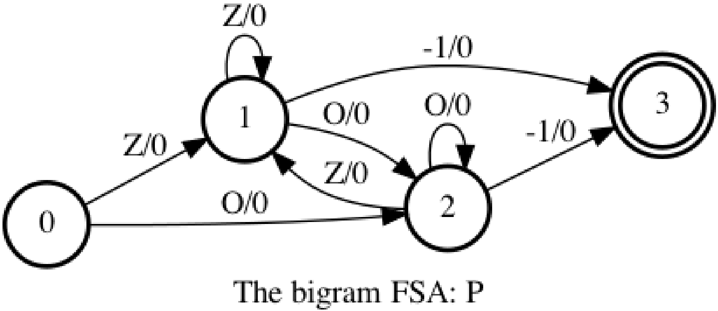
- In (a), there is a state 4, indicating a blank separating the two consecutive symbols `O` in the transcript `Z O O`
- In (b), there are no such mandatory blanks
- **TODO:** We have not compared the WER between `ctc_topo` and `ctc_topo_modified`

Contents

LF-MMI Training

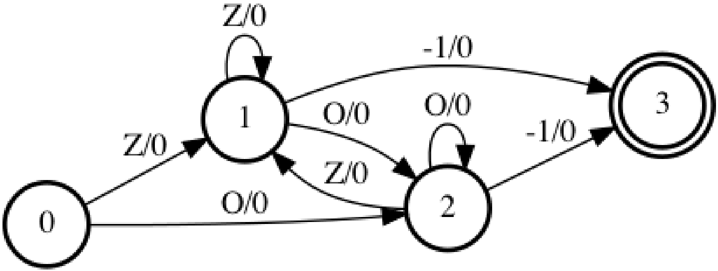
- LF-MMI training reuses the same FSA operations from CTC training
  - Differs only in numbers/types of FSA.

The bigram P



(a) The bigram FSA: `P`

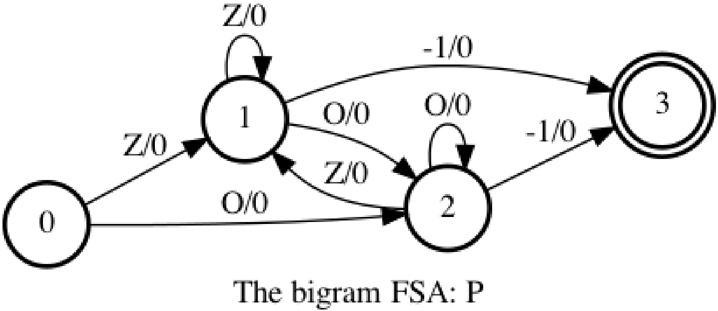
The bigram P



(a) The bigram FSA: P

- The scores on every arc are learnable parameters
  - They are trained together with the neural networks

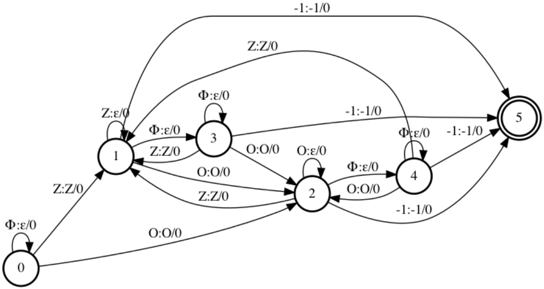
The bigram P



(a) The bigram FSA: `P`

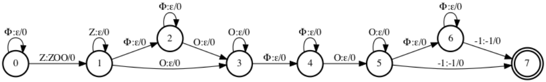
- **CAUTION:**
  - Number of arcs is $\mathcal{O}(n^2)$, i.e., quadratic in n
  - Where n is the number tokens

The Numerator Graph



k2.compose(ctc_topo, P)

(a) `ctc_topo_P = k2.compose(ctc_topo, P)`

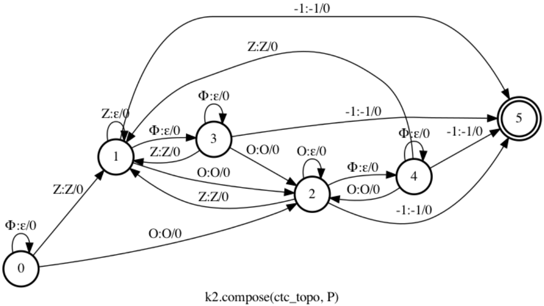


k2.compose(ctc_topo_P, L)

(b) `num_graph = k2.compose(ctc_topo_P,L)`

- CAUTION: Arc scores are not 0s in practice

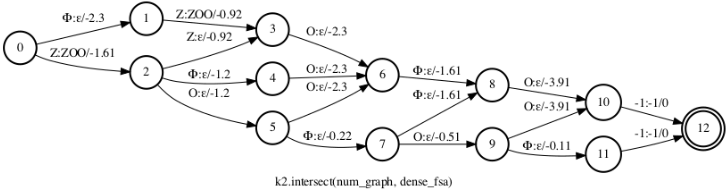
The Denominator Graph



(a) `ctc_topo_P = k2.compose(ctc_topo, P)`

- `ctc_topo_P` is the denominator graph, `den_graph`
- CAUTION: Arc scores are not 0s in practice

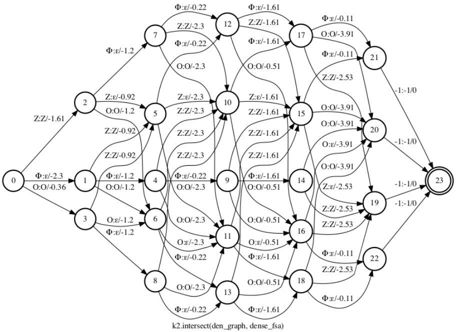
The Numerator Lattice



(a) `num_lats = k2.intersect(num_graph, dense_fsa)`

- It is identical to the decoding lattice in CTC training when the scores of P are all 0s

The Denominator Lattice



(a) `den_lats = k2.intersect(den_graph, dense_fsa)`

- **HINT:** `num_lats` is a subgraph of `den_lats`
  - That is, `den_lats` contains all the paths that are in `num_lats`
  - `den_lats` also contains extra paths that are not present in `num_lats`

Objective Function of LF-MMI Training

- To maximize `num_scores - den_scores`
  - `num_scores` = `num_lats.get_tot_scores(log_semiring=True)`
  - `den_scores` = `den_lats.get_tot_scores(log_semiring=True)`

Objective Function of LF-MMI Training

- To maximize `num_scores - den_scores`
  - `num_scores = num_lats.get_tot_scores(log_semiring=True)`
  - `den_scores = den_lats.get_tot_scores(log_semiring=True)`

```
    num = k2.intersect_dense(num, dense_fsa_vec, 10.0)
    den = k2.intersect_dense(den, dense_fsa_vec, 10.0)

    num_tot_scores = num.get_tot_scores(
        log_semiring=True,
        use_double_scores=True)
    den_tot_scores = den.get_tot_scores(
        log_semiring=True,
        use_double_scores=True)
    tot_scores = num_tot_scores - den_scale * den_tot_scores
```

```
      optimizer.zero_grad()
      (-tot_score).backward()
```

(a) See `https://github.com/k2-fsa/snowfall/blob/master/egs/aishell/asr/simple_v1/mmi_bigram_train.py#L126`

Objective Function of LF-MMI Training (Continue)

- To maximize `num_scores - den_scores`
  - `num_scores = num_lats.get_tot_scores(log_semiring=True)`
  - `den_scores = den_lats.get_tot_scores(log_semiring=True)`

- Explanation:
  - `num_scores`: It is the log of the sum of all **correct** paths
  - `den_scores`: It is the log of the sum of all **possible** paths
  - `num_scores` is always less than `den_scores`
  - Aim to
    - Increase the probabilities of correct paths
    - Decrease the probabilities of incorrect paths

LF-MMI Training Summary

- `ctc_topo`
- The bigram LM `P`
- The lexicon `L`
- `num_graph`
- `den_graph`
- `num_lats`
- `den_lats`
- `get_tot_scores` in log-semiring

- NOTE
  - k2 is a very **generic framework** supporting **FSA** operations, in a **differentiable** manner
  - We don't specify the underlying neural network model
    - You can use any types of network you like

Contents

Summary

- This talk has covered
  - How to implement CTC training and LF-MMI training
    - with **FSA** operations in **k2**
- Some terms
  - `ctc_topo`
  - The bigram LM `P`
  - The lexicon `L`
  - `num_graph`
  - `den_graph`
  - `num_lats`
  - `den_lats`
  - `get_tot_scores`
  - log-semiring
  - decoding lattice

Thank you!