Using SSL models for Multilingual ASR

Lucas Ondel & Léa-Marie Lam-Yee-Mui

Who we are…

Léa-Marie:

- Ph. D. student at LISN (ex-LIMSI)
- Working on adaptation of multilingual systems for ASR

Lucas:

- Postdoc at Sonos
- Working on “on-device” ASR

For the workshop

Working with 2 teams:

- Multilingual/Code-Switching ASR:
  - Building ASR systems coping with 2 languages at once
- Leveraging Pre-Training Models :
  - Adapting self-supervised models for speech processing

Research focus for the workshop:

- “Universal” Speech Recognition System

Universal ASR

An ASR system is universal if it usable **for everyone** and **by everyone**:

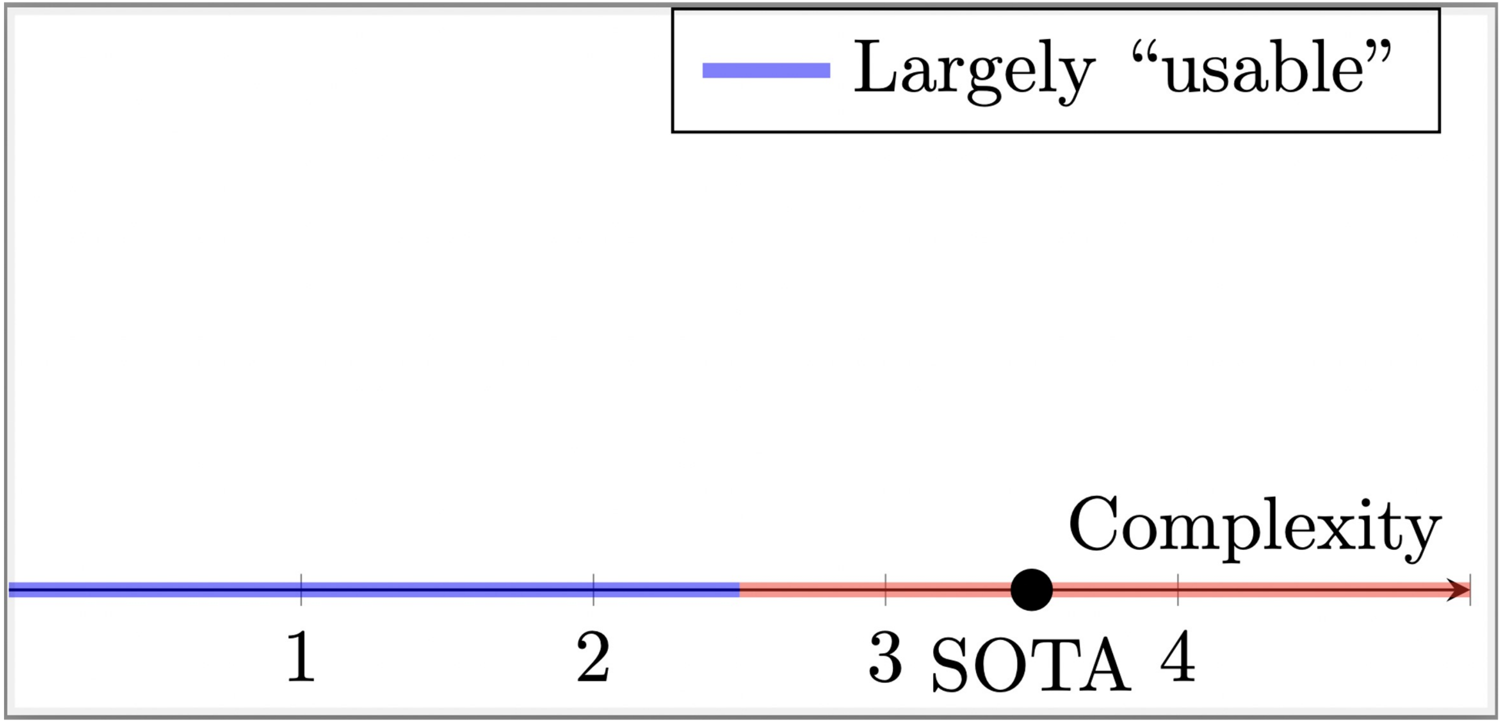
- It can recognize all languages (i.e. usable for everyone)
- It’s construction and deployment is simple enough (i.e. usable by everyone)

Complexity

Complexity of building a production level ASR for a language C(L):

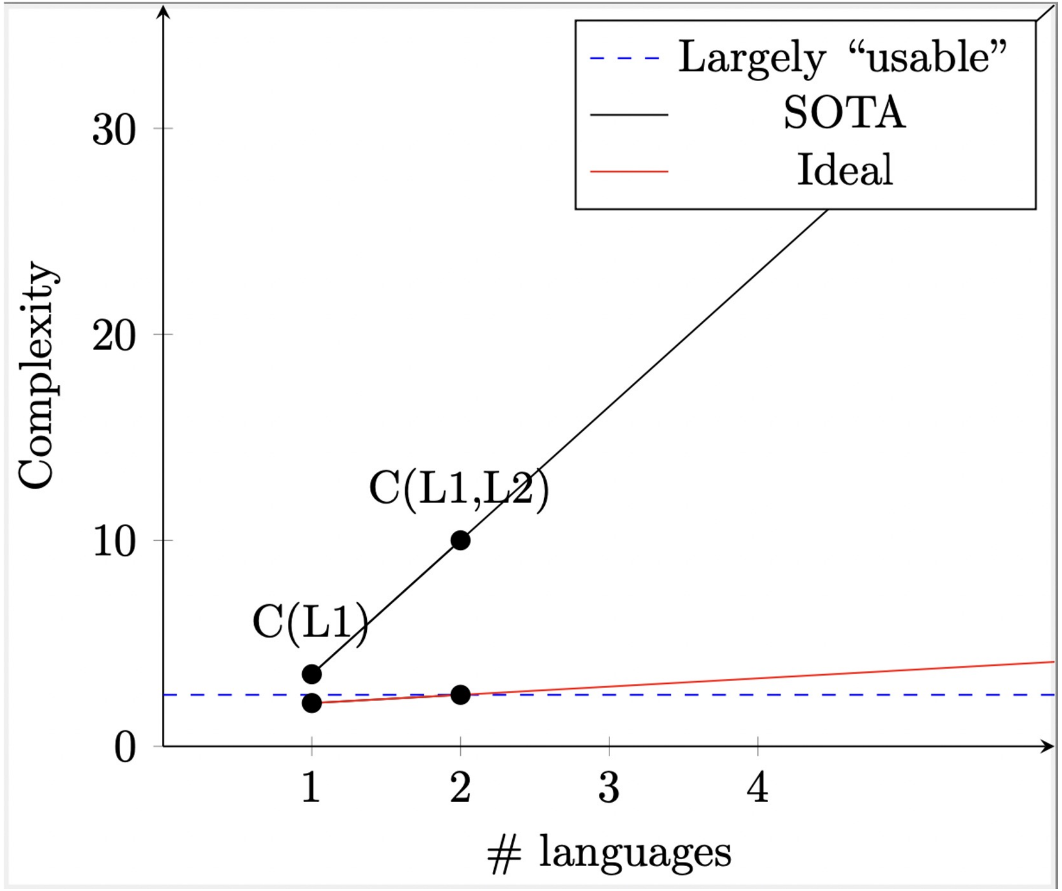
- Data requirements
- Software complexity
- Memory requirements
- Computation requirements
- …

- 1 - handyman: no expert knowledge & no infrastructure needed
- 2 - non-expert company: no expert knowledge but some infrastructures needed
- 3 - dedicated company/university: expert knowledge & dedicated infrastructures
- 4 - big tech: expert knowledge and large infrastructures


Largely "usable"
Complexity
1
2
3
SOTA
4

Complexity of building a 2-languages ASR

$$C(L1, L2) > C(L1) + C(L2)$$


Largely "usable"
SOTA
Ideal
30
20
10
0
Complexity
C(L1,L2)
C(L1)
1
2
3
4
languages

Using SSL models

SSL models:

- Strong improvements on multilingual ASR 🙂
- Ease of use: easily adapted less target data 🙂
- Huge memory and computation requirements 🙁
- Decoding several languages is still a big issue 🙁

Towards Universal Speech Recognition…

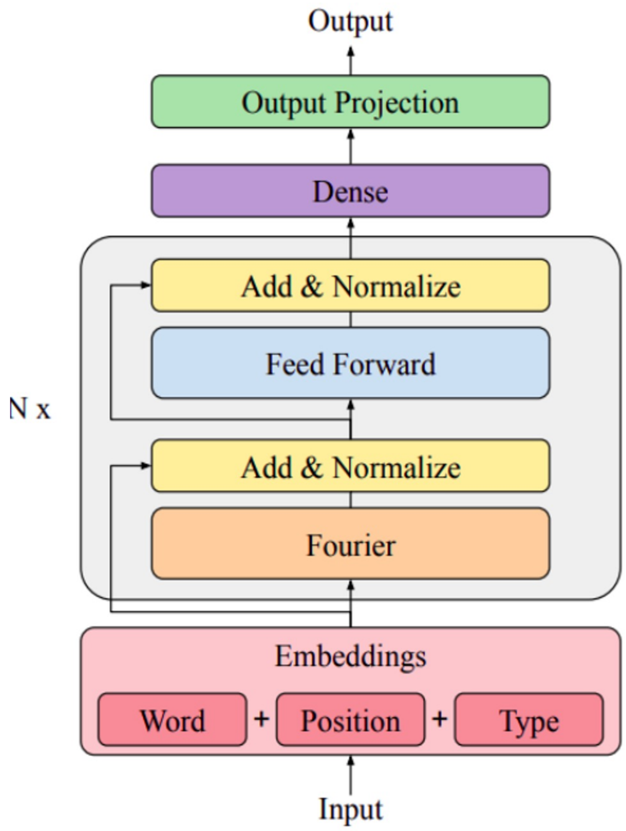
- **Lightweight SSL** models
  - Using FNet architecture for pre-training on speech
- Using **semiring algebra** for adaptation and inference in SSL models for ASR
  - Efficient adaptation of SSL models with LF-MMI
  - Decoding speech

Simplification of models

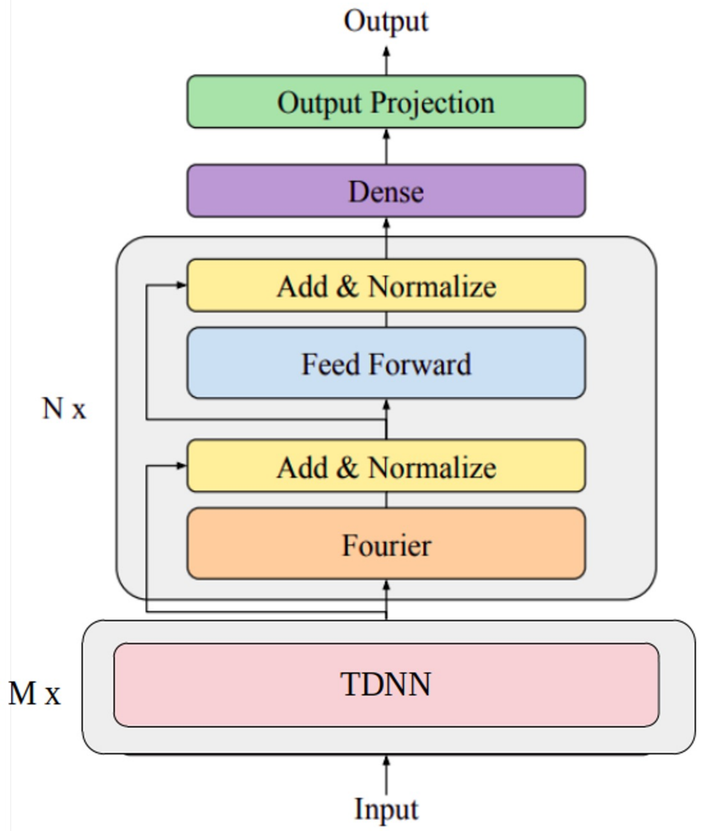
- Transformers need lots of computation/memory

- Can we simplify the network architecture:
  - Use FNet[1] instead of Transformer

[1]"FNet: Mixing Tokens with Fourier Transforms" https://arxiv.org/pdf/2105.03824.pdf



FNet architecture



TDNN-FNet architecture

Progress

As a first try, finished Pytorch implementation of FNET architecture

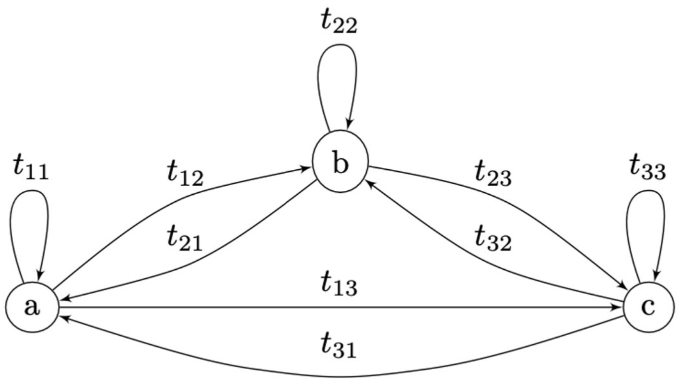
Added models to PyChain for an ASR pipeline on mini-Librispeech

Next step:

- Pre-training a FNet on LibriSpeech using classical pre-training loss

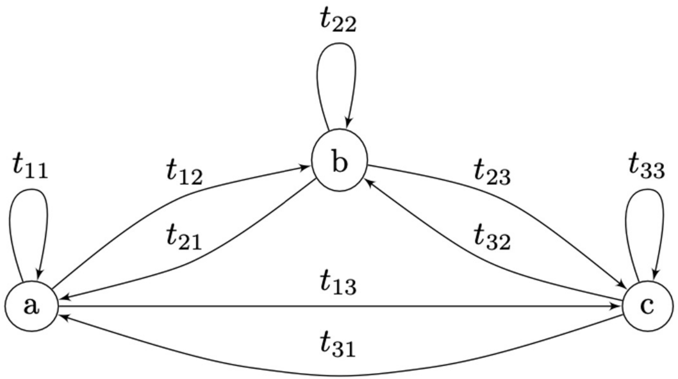
Graph and neural networks

- In ASR training and inference are operations on (probabilistic) graphs



- Not very friendly for neural networks

Probabilistic Graph



$$\mathbf{T} = \begin{bmatrix} p(z_n = a|z_{n-1} = a) & p(z_n = b|z_{n-1} = a) & p(z_n = c|z_{n-1} = a) \\ p(z_n = a|z_{n-1} = b) & p(z_n = b|z_{n-1} = a) & p(z_n = c|z_{n-1} = b) \\ p(z_n = a|z_{n-1} = c) & p(z_n = b|z_{n-1} = c) & p(z_n = c|z_{n-1} = c) \end{bmatrix}$$

T is usually sparse

Operations on graphs are linear operations

$$\boldsymbol{\alpha}_n = \mathbf{v}_n \circ (\mathbf{T}^\top \boldsymbol{\alpha}_{n-1})$$
$$\boldsymbol{\beta}_n = \mathbf{T}(\boldsymbol{\beta}_{n+1} \circ \mathbf{v}_n)$$
$$p(z_n|\mathbf{x}) = \frac{\alpha_n(z_n)\beta_n(z_n)}{\sum_{z_N} \alpha_N(z_N)}.$$

Using Semiring Algebra

$$\boldsymbol{\alpha}_n^{\log} = \mathbf{v}_n^{\log} \circ (\mathbf{T}^{\log \top} \boldsymbol{\alpha}_{n-1}^{\log})$$

$$\boldsymbol{\beta}_n^{\log} = \mathbf{T}^{\log} (\boldsymbol{\beta}_{n+1}^{\log} \circ \mathbf{v}_n^{\log})$$

$$\log p(z_n | \mathbf{x}) = \frac{\alpha_n^{\log}(z_n) \beta_n^{\log}(z_n)}{\sum_{z_N} \alpha_N^{\log}(z_N)}.$$

Used for CTC, LF-MMI, …

GPU-Accelerated Forward-Backward algorithm with Application to Lattice-Free MMI

Manipulating graph with semiring linear algebra

- Easy integration with neural networks (training / inference)
- Simpler code
- Better optimization

System	Dataset	B/F	Duration	WER (%)
PyChain	MiniLS	128/1	0h42	27.17
proposed	MiniLS	64/2	**0h22**	**21.21**
PyChain	WSJ	128/1	6h48	4.74
proposed	WSJ	64/2	**3h20**	**4.37**

For the workshop…

- Efficient adaptation of SSL models with LF-MMI loss function for ASR

- Matrix-based decoder (Multilingual team)

Progress: theoretical development

Derivation of matrix-based algorithm on graph (in progress)

- Some interestings results:
  - Speech decoder

$$\mathbf{T} = \mathbf{S} + \sum_{k}^{K} \boldsymbol{\nu}_k \boldsymbol{\delta}_k^\top ,$$

  - Graph composition

$$\mathbf{T}_2 = \begin{bmatrix} \mathbf{T}^1 & & \\ & \ddots & \\ & & \mathbf{T}^d \end{bmatrix} + (\mathbf{M}_K \mathbf{T}_1 \mathbf{M}_K^\top) \odot \left(\begin{bmatrix} \boldsymbol{\omega}_1 \\ \vdots \\ \boldsymbol{\omega}_d \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}_1 \\ \vdots \\ \boldsymbol{\alpha}_1 \end{bmatrix}^\top \right)$$

Progress: implementation

- Main code in **Julia** (it is necessary to use sparse semiring matrices)
- Creating a python wrapper to integrate it with pytorch

https://github.com/FAST-ASR/Semirings.jl

https://github.com/FAST-ASR/MarkovModels.jl

Please add a star 😉 !!

Timeline

Before the workshop:

- Implement FNet - DONE
- Wav2Vec 2.0 pipeline with FNet architecture on subset of LibriSpeech / WSJ (IN PROGRESS)
- LF-MMI adaptation of end-to-end models :
  - PyChain (DONE)
  - Matrix-based (IN PROGRESS)
- Matrix-based training / decoding:
  - Theoretical development (IN PROGRESS)
  - Implementation (IN PROGRESS)

Workshop period:

- Full scale LibriSpeech (FNET)
- Adaptation on multilingual data
- Comparison of adaptation / decoding with Kaldi / ESPNET / K2