Language Modeling in Automatic Speech Recognition

ZRE lecture

Karel Beneš

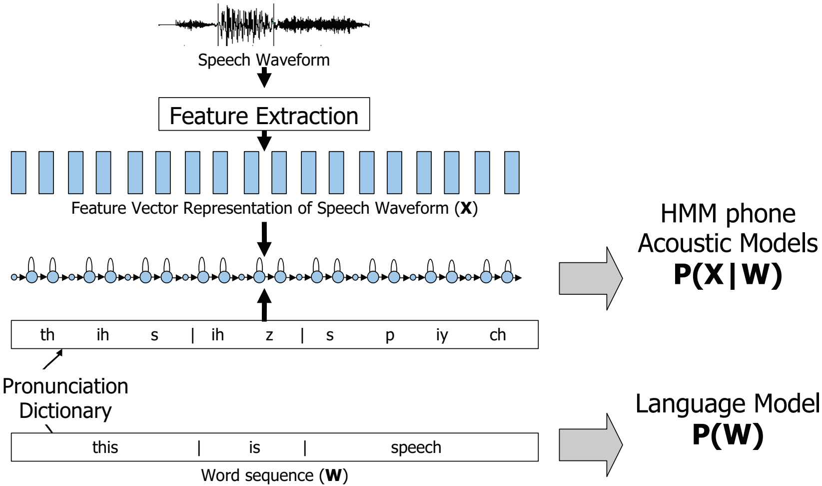
Slides obtained from Mirko Hannemann,
as based on other slides from Daniel Povey
and some slides from T. Schultz, M. Mohri, M. Riley and S. Renals

15.04.2015

Roadmap of the Lecture

1. ASR Intro
2. Language Modeling
3. Reminder: HMMs, ASR, Viterbi
4. Reminder: FSA, FST, WFST
5. Encoding Knowledge in WFSTs
6. Decoding with WFSTs

Automatic speech recognition



Automatic speech recognition

- Determine the most probable word sequence $\tilde{W}$ given the observed acoustic signal Y

$$\tilde{W} = argmax\, P(W|Y) = \frac{argmax\, P(W)\, P(Y|W)}{P(Y)}$$

- Search for word sequence $\tilde{W}$ that maximizes P(W) and P(Y|W)
  - P(W) = language model (likelihood of word sequence)
  - P(Y|W) = acoustic model

    (likelihood of observed acoustic signal given word sequence)

Roadmap of the Lecture

Language model

Estimate probability of a word sequence W:

$$P(W) = P(w_1, w_2, \ldots, w_N) \quad (1)$$

$$P(W) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1, w_2) \cdot \cdots \cdot P(w_N|w_1, \ldots, w_{N-1}) \quad (2)$$

Markov assumption, n-grams counting

Approximate by only considering 1-word history, e.g.:

$$P(W) \approx P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_2) \cdot \cdots \cdot P(w_N|w_{N-1}) \tag{3}$$

Gives raise to *2-gram / bigram* language model. Probabilities $P(w_a|w_b)$ are estimated as:

$$P(w_a|w_b) = \frac{C(w_b, w_a)}{C(w_b)} \tag{4}$$

Markov assumption, n-grams counting

Approximate by only considering 2-word history, e.g.:

$$P(W) \approx P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1, w_2) \cdot \dots \cdot P(w_N|w_{N-2}, w_{N-1}) \quad (5)$$

Gives raise to *3-gram / trigram* language model. Probabilities $P(w_a|w_b, w_c)$ are estimated as:

$$P(w_a|w_b, w_c) = \frac{C(w_b, w_c, w_a)}{C(w_b, w_c)} \quad (6)$$

Problem of n-grams

1. limited context, e.g. "...to Paris, we couldn't wait to see the ??? ". Unfortunate, but acceptable.
2. unseen n-grams, e.g. 2M words from Wikipedia -> 2.6 % of unseen words in other wiki texts. Cca 1/3 of trigrams not seen since 1970s. So called *Curse of Dimensionality*, severe.

Katz's backoff

If we have enough data ($C(w_{i-n+1}, \dots, w_{i-1}, w_i) > k$):

$$P(w_i | w_{i-n+1}, \dots, w_{i-1}) = d_{w_{i-n+1}, \dots, w_{i-1}, w_i} \frac{C(w_{i-n+1}, \dots, w_{i-1}, w_i)}{C(w_{i-n+1}, \dots, w_{i-1})} \quad (7)$$

Here, $d_{C(w_{i-n+1}, \dots, w_{i-1}, w_i)}$ is a discount factor, estimated typically with Good-Turing algorithm.

If $C(w_{i-n+1}, \dots, w_{i-1}, w_i) \leq k$:

$$P(w_i | w_{i-n+1}, \dots, w_{i-1}) = \alpha_{w_{i-n+1}, \dots, w_{i-1}, w_i} P(w_i | w_{i-n+2}, \dots, w_{i-1}) \quad (8)$$

Here, we distribute the discounted probability from (7) through $\alpha_{w_{i-n+1}, \dots, w_{i-1}, w_i}$.

The threshold k for "enough" can in practice be set to 0.

Kneser-Ney smoothing

For bigrams:

$$P(w_i|w_{i-1}) = \frac{\max(C(w_{i-1}w_i) - \delta, 0)}{C(w_{i-1})} + \lambda \frac{N_{1+}(\bullet w_i)}{N_{1+}(\bullet\bullet)} \tag{9}$$

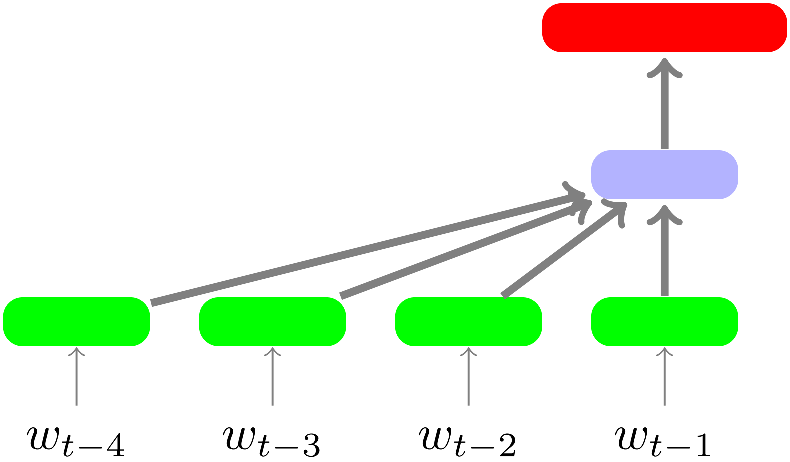
The mysterious $N_{1+}(\cdot)$ just counts how many unique bigrams have we seen:

$$N_{1+}(\bullet w_i) = |\{w_{i-1}|C(w_{i-1}, w_i)\}| \tag{10}$$

$$N_{1+}(\bullet\bullet) = |\{w_{i-1}, w_i|C(w_{i-1}, w_i)\}| \tag{11}$$

So effectively, we only believe the unigram w_i, if we have seen it in many different contexts.

Battling the Curse of Dimensionality using NNs



Battling the Curse of Dimensionality using NNs

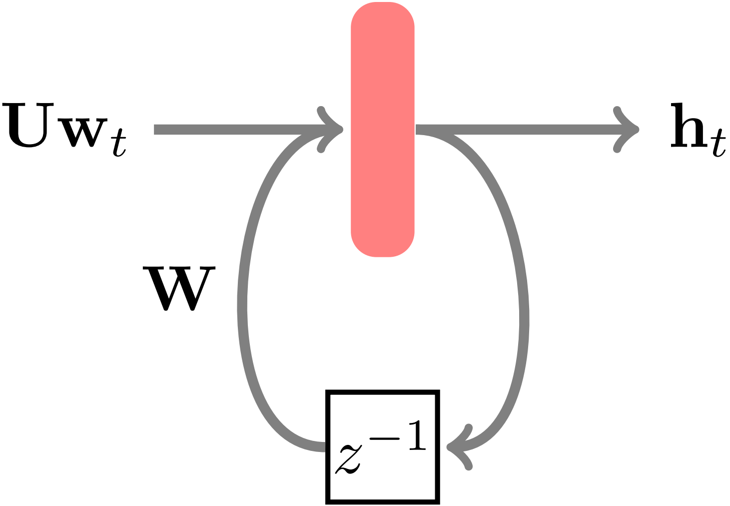
$$\mathbf{e}_t = \mathbf{E}[w_t] \tag{12}$$

$$\mathbf{h}_t = \tanh(\mathbf{W}(\mathbf{e}_{t-2} || \mathbf{e}_{t-1}) + \mathbf{b}_h) \tag{13}$$

$$\mathbf{y}_t = \mathrm{softmax}(\mathbf{V}\mathbf{h}_t + \mathbf{b}_y) \tag{14}$$

First, words are replaced by embeddings (12). Then, several embeddings (two) are concatenated and serve as input to a hidden layer (13). Finally, the next word is predicted (14).

Battling the Curse of Dimensionality using RNNs



Battling the Curse of Dimensionality using RNNs

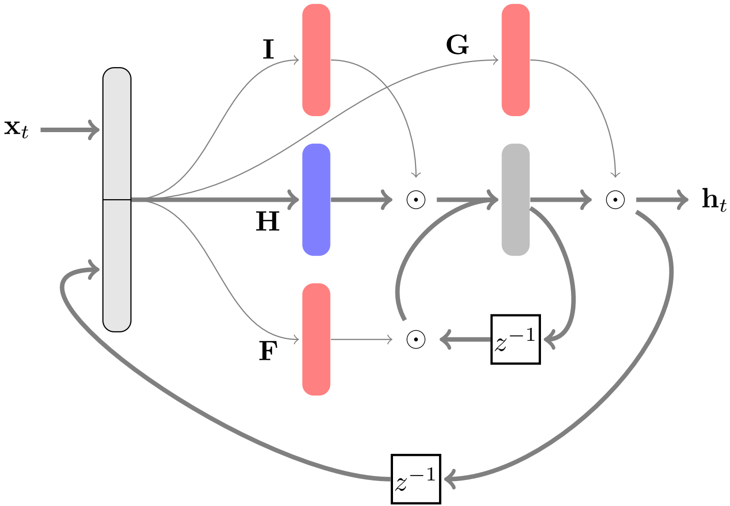
$$\mathbf{e}_t = \mathbf{E}[w_t] \tag{15}$$

$$\mathbf{h}_t = \tanh(\mathbf{e}_{t-1} + \mathbf{W}\mathbf{h}_{t-1} + \mathbf{b}_h) \tag{16}$$

$$\mathbf{y}_t = \mathrm{softmax}(\mathbf{V}\mathbf{h}_t + \mathbf{b}_y) \tag{17}$$

Only the neural magic in the middle changes!

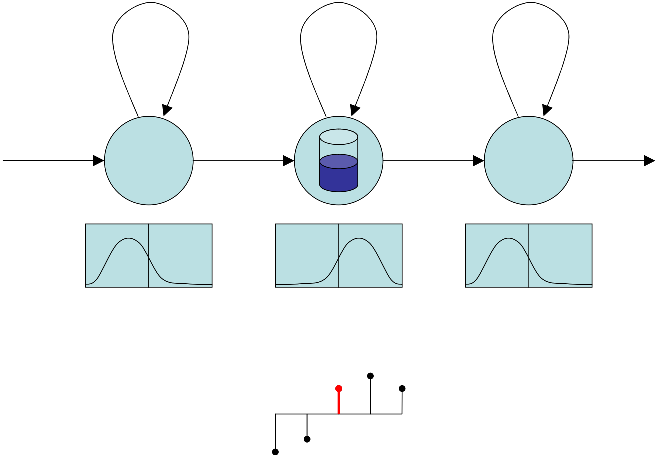
A Glimpse of State-of-the-Art: Long Short-Term Memory Architecture



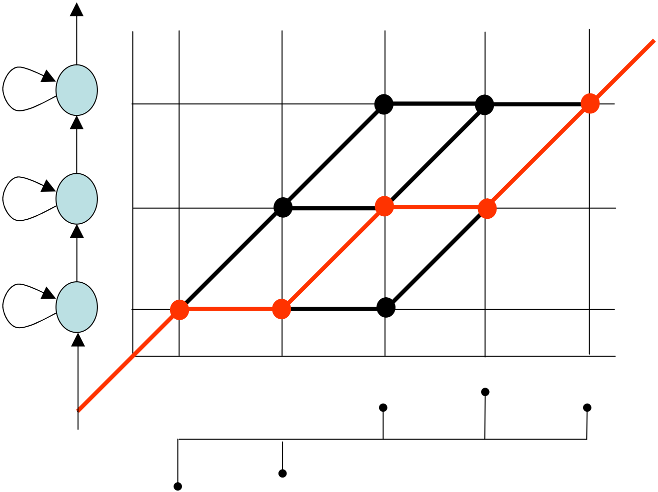
Rather effective in processing longer contexts (ca. 30 words).

Roadmap of the Lecture

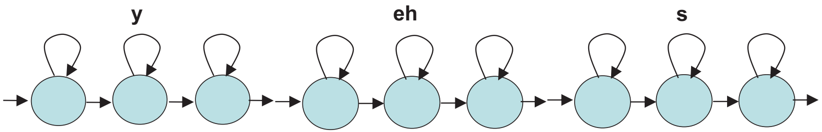
Hidden Markov Model, Token passing



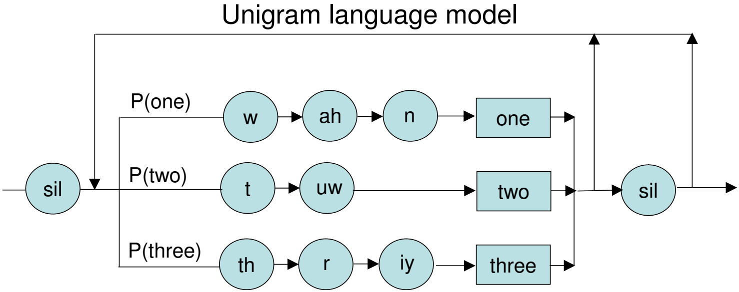
Viterbi algorithm, trellis



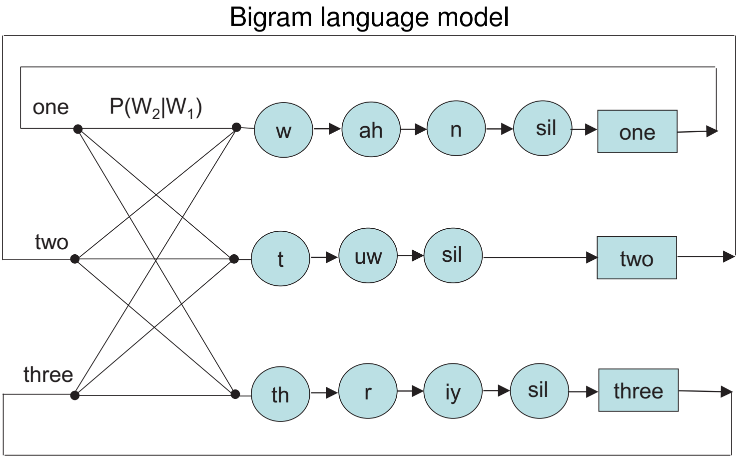
Phoneme based models — re-usable acoustic units



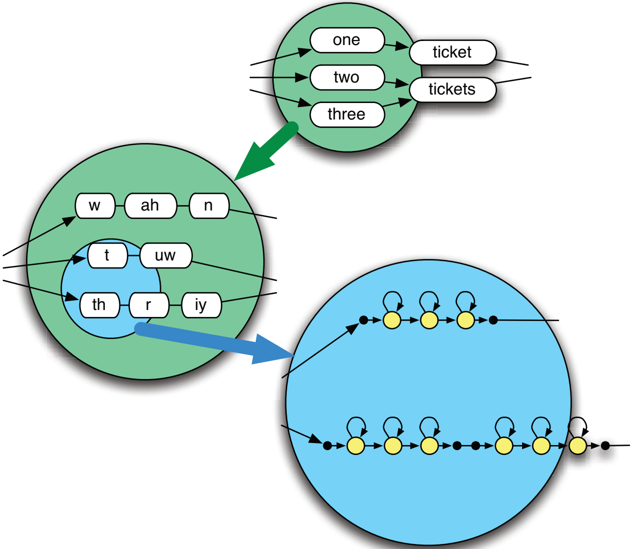
Decoding graph/recognition network



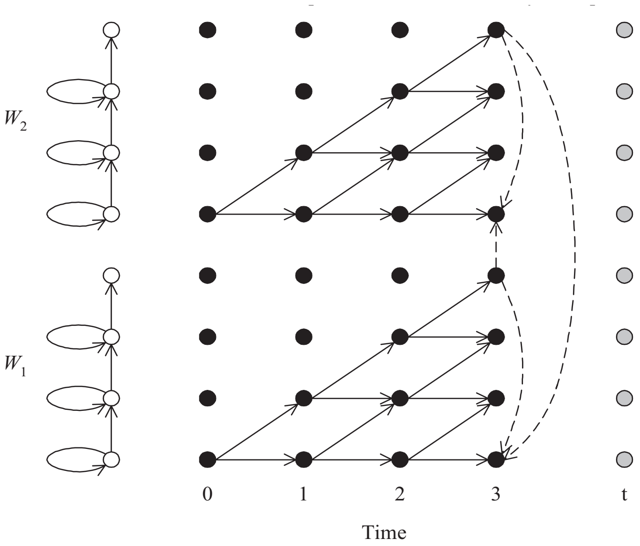
Decoding graph/recognition network



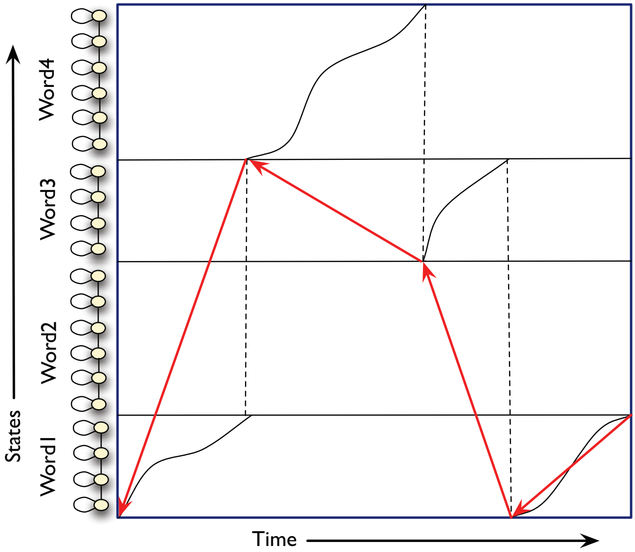
Decoding graph/recognition network



Viterbi path with complex models



Viterbi path with back-tracking



Roadmap of the Lecture

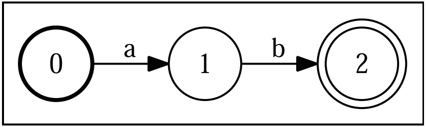
Why Finite State Transducers?

Motivation:

- most components (LM, lexicon, lattice) are finite-state
- unified framework for describing models
- integrate different models into a single model via composition operations
- improve search efficiency via optimization algorithms
- flexibility to extend (add new models)

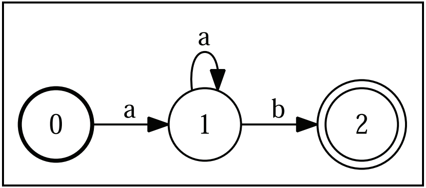
$\rightarrow$ *speed*: pre-compiled search space, near realtime performance on embedded systems

$\rightarrow$ *flexibility*: same decoder used for hand-held devices and LVCSR



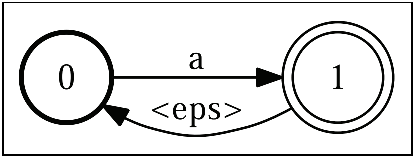
- An FSA “accepts” a set of strings
- (a string is a sequence of symbols).
- View FSA as a representation of a possibly infinite set of strings.
- This FSA accepts just the string *ab*, i.e. the set $\{ab\}$
- Numbers in circles are state labels (not really important).
- Labels are on arcs are the symbols.
- Start state(s) bold; final/accepting states have extra circle.
  - Note: it is sometimes assumed there is just one start state.

A less trivial FSA



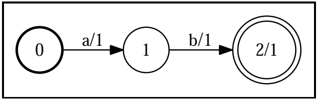
- The previous example doesn't show the power of FSAs because we could represent the set of strings finitely.
- This example represents the infinite set $\{ab, aab, aaab, \ldots\}$
- Note: a string is "accepted" (included in the set) if:
  - There is a path with that sequence of symbols on it.
  - That path is "successful' (starts at an initial state, ends at a final state).

The epsilon symbol



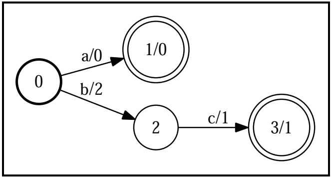
- The symbol ϵ has a special meaning in FSAs (and FSTs)
- It means "no symbol is there".
- This example represents the set of strings $\{a, aa, aaa, \ldots\}$
- If ϵ were treated as a normal symbol, this would be $\{a, a\epsilon a, a\epsilon a\epsilon a, \ldots\}$
- In text form, ϵ is sometimes written as <eps>
- Toolkits implementing FSAs/FSTs generally assume <eps> is the symbol numbered zero

Weighted finite state acceptors



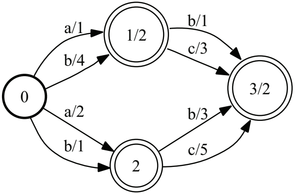
- Like a normal FSA but with costs on the arcs and final-states
- Note: cost comes after "/". For final-state, "2/1" means final-cost 1 on state 2.
- View WFSA as a function from a string to a cost.
- In this view, unweighted FSA is f : string $\rightarrow \{0, \infty\}$.
- If multiple paths have the same string, take the one with the lowest cost.
- This example maps *ab* to $(3 = 1 + 1 + 1)$, all else to ∞.

Weights vs. costs



- Personally I use "cost" to refer to the numeric value, and "weight" when speaking abstractly, e.g.:
  - The acceptor above accepts *a* with unit weight.
  - It accepts *a* with zero cost.
  - It accepts *bc* with cost $4 = 2 + 1 + 1$
  - State 1 is final with unit weight.
  - The acceptor assigns zero weight to *xyz*.
  - It assigns infinite cost to *xyz*.

Probability or tropical semi-ring



Probability semiring $(\mathbb{R}_+, +, \times, 0, 1)$	Tropical semiring $(\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$
$[\![A]\!](ab) = 14$	$[\![A]\!](ab) = 4$
$(1 \times 1 \times 2 + 2 \times 3 \times 2 = 14)$	$(\min(1 + 1 + 2, 3 + 2 + 2) = 4)$

Semirings

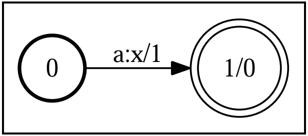
- The semiring concept makes WFSAs more general.
- A semiring is
  - A set of elements (e.g. $\mathbb{R}$)
  - Two special elements $\bar{1}$ and $\bar{0}$ (the identity element and zero)
  - Two operations, $\oplus$ (plus) and $\times$ (times) satsifying certain axioms.

Semiring examples. $\oplus_{\log}$ *is defined by:* $x \oplus_{\log} y = -\log(e^{-x} + e^{-y})$.

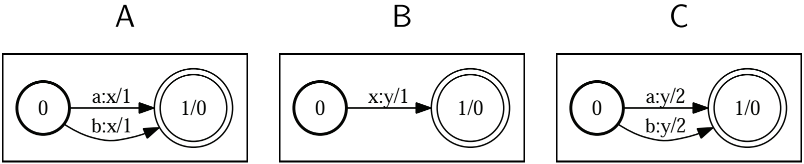
SEMIRING	SET	$\oplus$	$\otimes$	$\bar{0}$	$\bar{1}$
Boolean	$\{0,1\}$	$\vee$	$\wedge$	0	1
Probability	$\mathbb{R}_+$	$+$	$\times$	0	1
Log	$\mathbb{R} \cup \{-\infty, +\infty\}$	$\oplus_{\log}$	$+$	$+\infty$	0
Tropical	$\mathbb{R} \cup \{-\infty, +\infty\}$	$\min$	$+$	$+\infty$	0

- In WFSAs, weights are multiplied along paths
- summed over paths with identical symbol-sequences

Weighted finite state transducers (WFST)



- Like a WFSA except with two labels on each arc.
- View it as a function from a (pair of strings) to a weight
- This one maps (a, x) to 1 and all else to ∞
- Note: view 1 and ∞ as costs. ∞ is $\bar{0}$ in semiring.
- Symbols on the left and right are termed "input" and "output" symbols.



- Notation: $C = A \circ B$ means, C is A composed with B.
- In special cases, composition is similar to function composition
- Composition algorithm "matches up" the "inner symbols"
  - i.e. those on the output (right) of A and input (left) of B

Composition algorithm

- Ignoring ϵ symbols, algorithm is quite simple.
- States in C correspond to tuples of (state in A, state in B).
  - But some of these may be inaccessible and pruned away.
- Maintain queue of pairs, initially the single pair $(0, 0)$ (start states).
- When processing a pair (s, t):
  - Consider each pair of (arc a from s), (arc b from t).
  - If these have matching symbols (output of a, input of b):
    - Create transition to state in C corresponding to (next-state of a, next-state of b)
    - If not seen before, add this pair to queue.
- With ϵ involved, need to be careful to avoid redundant paths...

Roadmap of the Lecture

Construction of decoding network

- WFST approach [Mohri et al.]
- exploit several knowledge sources (lexicon, grammar, phonetics) to find most likely spoken word sequence

$$HCLG = H \circ C \circ L \circ G \tag{18}$$

G probabilistic grammar or language model acceptor (word)

L lexicon (phones to words)

C context-dependent relabeling (ctx-dep-phone to phone)

H HMM structure (PDF labels to context-dependent phones)

Create H, C, L, G separately and compose them together

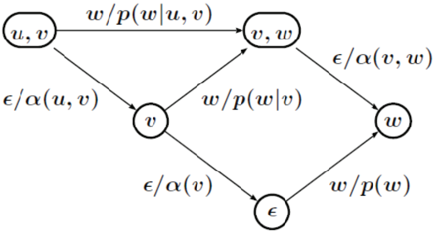
Language model acceptor G

- **G:** Grammar Transducer

Backing-off language model:

$$p(w|h) = \begin{cases} f(w|h) & : \text{if } N(w,h) > 0 \\ \alpha(h) \cdot f(w|\overline{h}) & : \text{if } N(w,h) = 0 \end{cases}$$

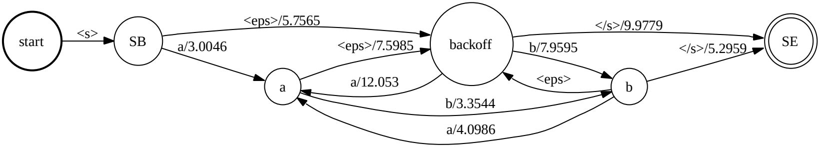
- **Input:** word
- **Weight:** history dependent word probability



Language models (ARPA back-off)

```
\1-grams:
-5.2347  a -3.3
-3.4568  b
 0.0000  <s> -2.5
-4.3333  </s>

\2-grams:
-1.4568  a b
-1.3049  <s> a
-1.78    b a
-2.30    b </s>
```

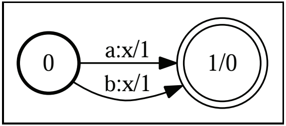


Pronunciation lexicon L

A	ax
ABERDEEN	ae b er d iy n
ABOARD	ax b r ao dd
ADD	ae dd
ABOVE	ax b ah v

Non-determinism: the same phone sequence can output different words ("I scream for ice cream.")

Deterministic WFSTs



- Taken to mean "deterministic on the input symbol"
- I.e., no state can have > 1 arc out of it with the same input symbol.
- Some interpretations (e.g. Mohri/AT&T/OpenFst) allow ϵ input symbols (i.e. being ϵ-free is a separate issue).
- I prefer a definition that disallows epsilons, except as necessary to encode a string of output symbols on an arc.
- Regardless of definition, not all WFSTs can be determinized.

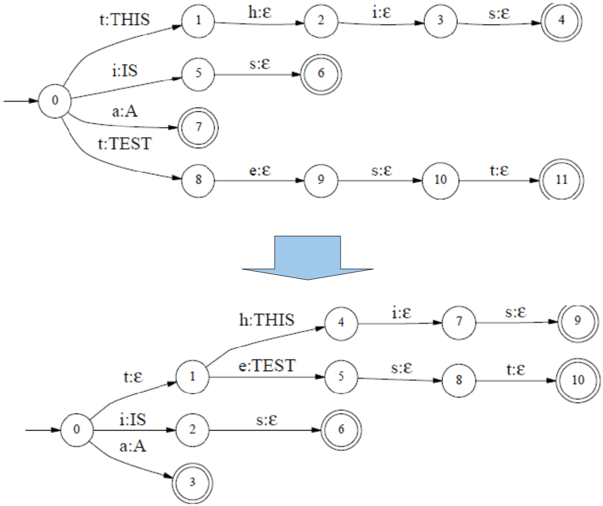
Pronunciation lexicon L with disambiguation symbols

```
A          ax #1
ABERDEEN   ae b er d iy n
ABOARD     ax b r ao dd
ADD        ae dd #1
ABOVE      ax b ah v
```

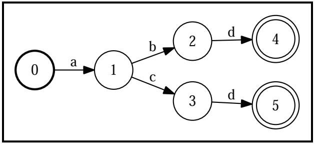
Added disambiguation symbols:

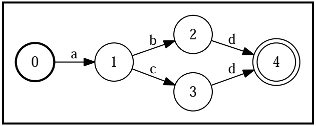
- if a phone sequences can output different words ("I scream for ice cream.")
- non-determinism: introduce disambiguation symbols, remove at last stage

Determinization (like making tree-structured lexicon)



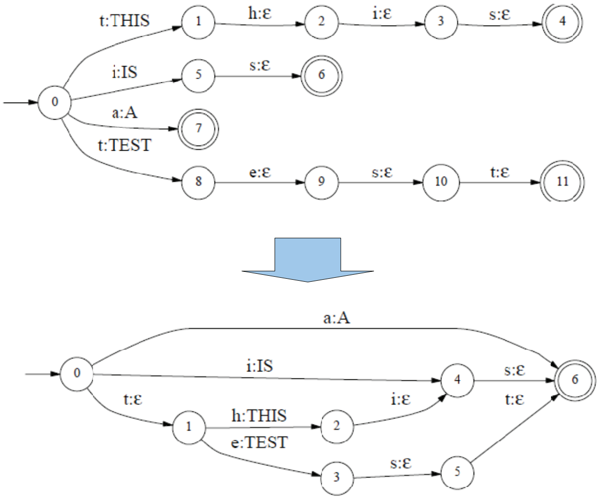
Minimal deterministic WFSTs



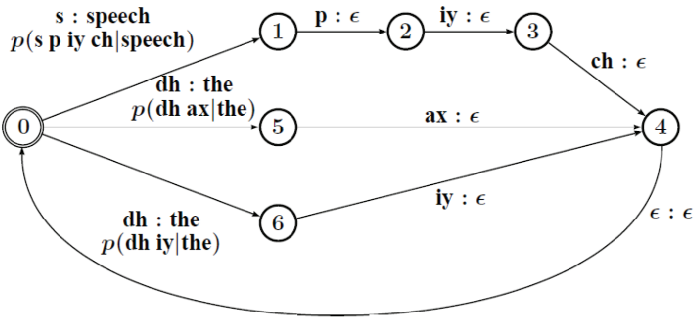


- Here, the left FSA is not minimal but the right one is.
- "Minimal" is normally only applied to *deterministic* FSAs.
- Think of it as suffix sharing, or combining redundant states.
- It's useful to save space (but not as crucial as determinization, for ASR).

Minimization (like suffix sharing)



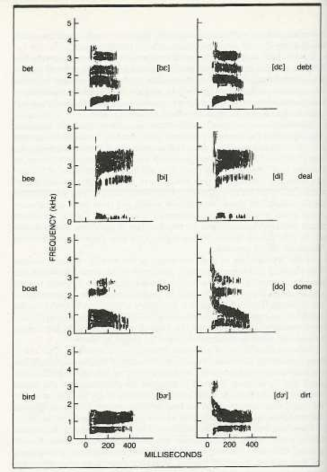
- **L:** Context-Dependency Transducer
  - **Input:** context-independent phone (phoneme)
  - **Output:** word
  - **Weight:** pronunciation probability



Context-dependency of phonemes

So far we use phonemes independent of context, but:

- co-articulation: pronunciation of phones changes with surrounding phones



Context-dependency transducer C

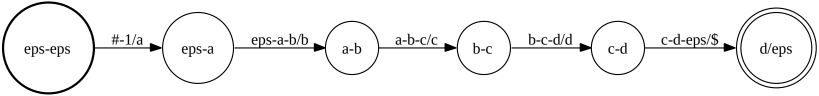
Introduce context-dependent phones:

- tri-phones: each model depends on predecessor and successor phoneme: a-b-c (b/a_c)
- implemented as context-dependency transducer C

Input: context-dependent phone (triphone)

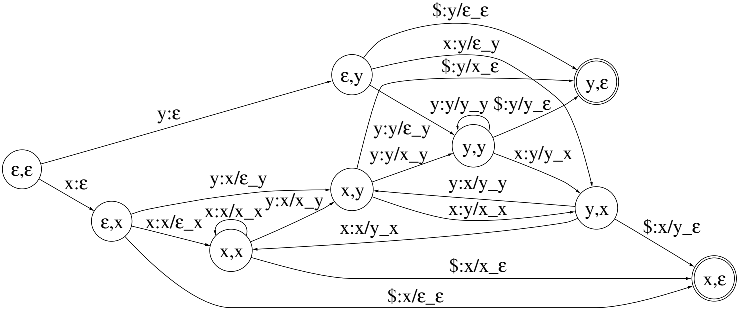
Output: context-independent phone (phone)

- shown is one path of it



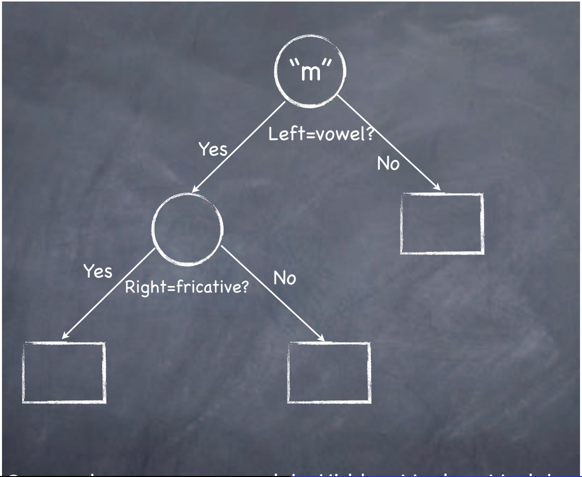
Context-dependency transducer C

Figure 6: Context-dependent triphone transducer transition: (a) non-deterministic, (b) deterministic.

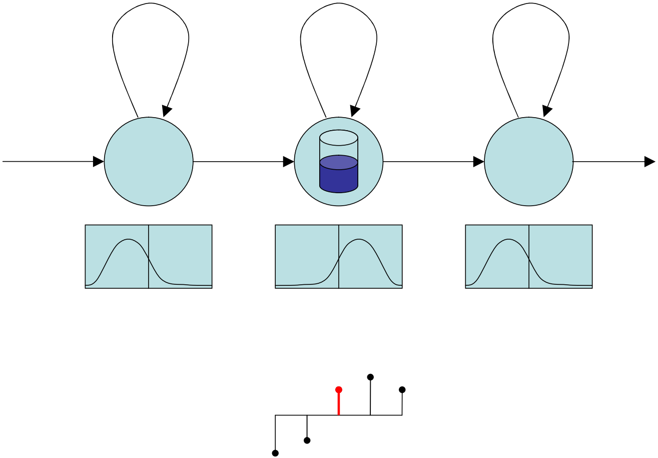


Phonetic decision tree

- too many context-dependent models (N^3) $\rightarrow$ clustering
- determine model-id (gaussian) based on phoneme context and state in HMM
- using questions about context (sets of phones)

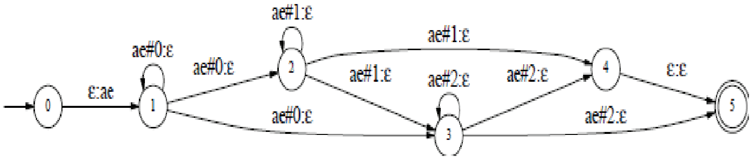


HMM as transducer

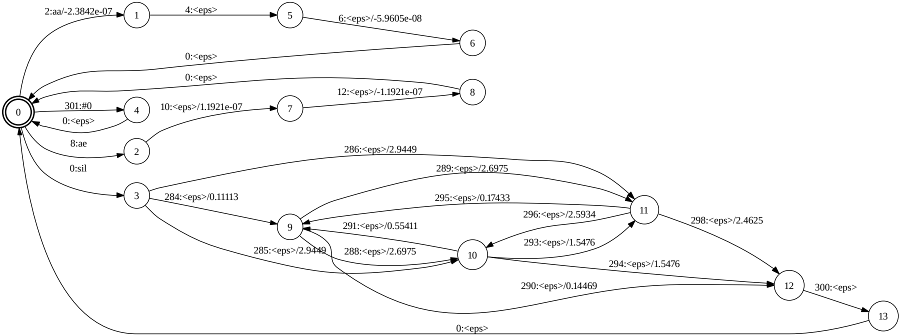


HMM as transducer (monophone)

- H: HMM Topology Transducer (maps states to phonemes)
  - **Input:** state
  - **Output:** context-dependent phone (triphone)
  - **Weight:** HMM transition probability



HMM transducer H_a



(here shown for monophone case, without self-loops)

Construction of decoding network

Let's put all together:

$$HCLG = H \circ C \circ L \circ G \tag{19}$$

H HMM: input PDF labels, output context-dependent phones

C context-dependency: input ctx-dep-phones, output phones

L lexicon: input phones, output words

G language model: input/output words

Construction of decoding network

WFST approach by [Mohri et al.]

$$HCLG = rds(min(det(H \circ det(C \circ det(L \circ G))))) \tag{20}$$

rds — remove disambiguation symbols
min — minimization, includes weight pushing

det — determinization

Kaldi toolkit [Povey et al.]

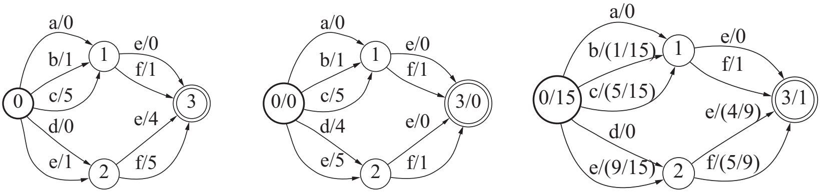
$$HCLG = asl(min(rds(det(H_a \circ min(det(C \circ min(det(L \circ G)))))))) \tag{21}$$

asl — add self loops

rds — remove disambiguation symbols

Weight and label pushing

- two WFSAs are equal, if they accept the same label sequences with the same weights
- local distribution of weights along the path can be different
- same holds for output labels in WFSTs



- for pruning: apply costs as early as possible
- make outgoing arcs stochastic distribution

$\rightarrow$ output labels not synchronized anymore in WFST

Decoding graph construction (complexities)

- Have to do things in a careful order or algorithms “blow up”
- Determinization for WFSTs can fail
  - need to insert “disambiguation symbols” into the lexicon.
  - need to “propagate these through” H and C.
- Need to guarantee that final HCLG is stochastic:
  - i.e. sums to one, like a properly normalized HMM
  - needed for optimal pruning (discard unlikely paths)
  - usually done by weight-pushing, but standard algorithm can fail, because FST representation of back-off LMs is non-stochastic
- We want to recover the phone sequence from the recognized path (words)
  - sometimes also the model-indices (PDF-ids) and the HCLG arcs that were used in best path

Roadmap of the Lecture

Decoding with WFSTs (finding best path)

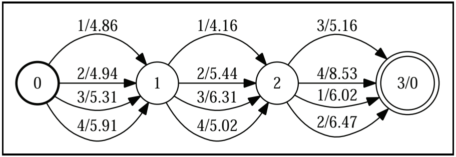
- Solve

$$W' = argmax_W P(X|W) P(W)$$

- Compose recognizer as (H o C o L o G) which maps states to word sequences
- Decode by aligning the feature vectors X with HCLG "

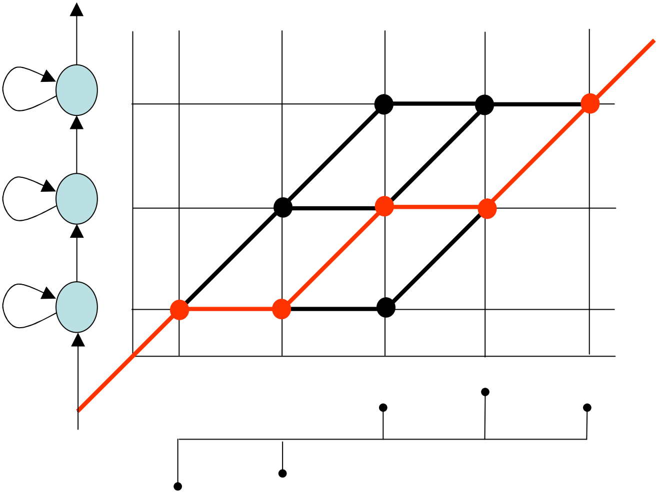
i.e.,

$$W' = argmax_W X \circ (H \circ C \circ L \circ G)$$

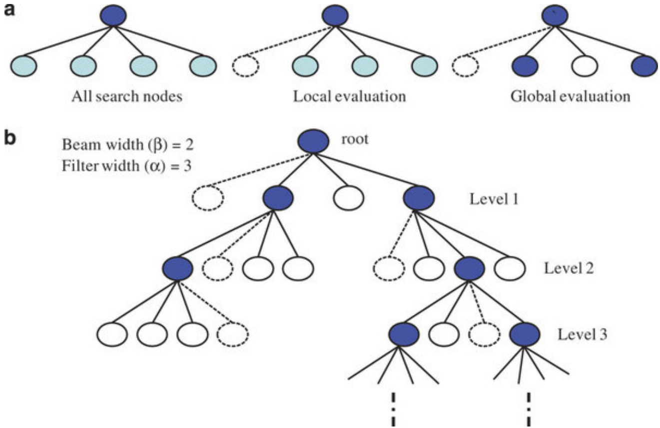


- First– a "WFST definition" of the decoding problem.
- Let U be an FST that encodes the acoustic scores of an utterance (as above).
- Let $S = U \circ HCLG$ be called the search graph for an utterance.
- Note: if U has N frames (3, above), then
  - #states in S is $\leq (N + 1)$ times #states in $HCLG$.
  - Like $N + 1$ copies of $HCLG$.

Viterbi algorithm, trellis



Beam search


a
All search nodes
Local evaluation
Global evaluation
b
Beam width (β) = 2
Filter width (α) = 3
root
Level 1
Level 2
Level 3

Decoding with WFSTs

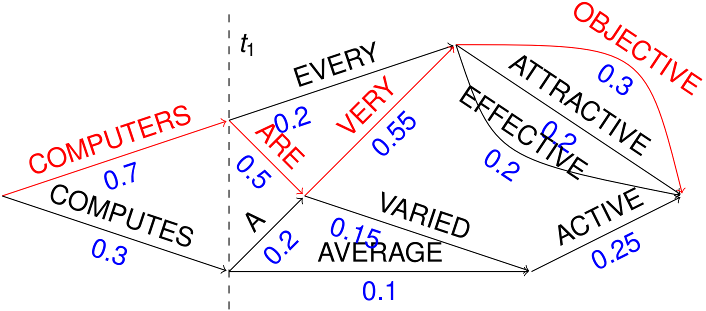
- With beam pruning, we search a subgraph of S.
- The set of "active states" on all frames, with arcs linking them as appropriate, is a subgraph of S.
- Let this be called the *beam-pruned subgraph* of S; call it B.
- A standard speech recognition decoder finds the best path through B.
- In our case, the output of the decoder is a linear WFST that consists of this best path.
- This contains the following useful information:
  - The word sequence, as output symbols.
  - The state alignment, as input symbols.
  - The cost of the path, as the total weight.

Decoding output

```
utt1  [ 2 6 6 6 6 10 ] [ 614 613 613 613 711 ] [ 122 123
utt1  SIL              th                        ax
utt1  <s>              THE
```

Word Lattice / Word Graph

Word Lattice: a compact representation of the search space



Lattices as WFSTs

The word “lattice” is used in the ASR literature as:

- Some kind of compact representation of the alternate word hypotheses for an utterance.
- Like an N-best list but with less redundancy.
- Usually has time information, sometimes state or word alignment information.
- Generally a directed acyclic graph with only one start node and only one end node.

Lattice Generation with WFSTs [Povey12]

Basic process (not memory efficient):

- Generate beam-pruned subgraph B of search graph S
- The states in B correspond to the active states on particular frames.
- Prune B with beam α to get pruned version P.
- Convert P to acceptor and lattice-determinize to get A (deterministic acceptor)

$\rightarrow$ No two paths in L have same word sequence (take best)

- Prune A with beam α to get final lattice L (in acceptor form).

Finite State Transducers for ASR

Pro's:

Fast: compact/minimal search space due to combined minimization of lexicon, phonemes, HMM's

Simple: easy construction of recognizer by composition from states, HMMs, phonemes, lexicon, grammar

Flexible: whatever new knowledge sources, the compose/optimize/search remains the same

Con's:

- composition of complex models generates a huge WFST
- search space increases, and huge memory is required
- esp. how to deal with huge language models

Resources

OpenFST `http://www.openfst.org`
Library, developed at Google Research (M. Riley, J. Schalkwyk, W. Skut) and NYU's Courant Institute (C. Allauzen, M. Mohri)

Mohri08 M. Mohri et al., *"Speech Recognition with weighted finite state transducers."*

Kaldi `http://kaldi.sourceforge.net`
Open source toolkit in C++ with recipes (D. Povey and others)

Povey11 D. Povey et al., *"The Kaldi Speech Recognition Toolkit."*

Povey12 D. Povey et al., *"Generating exact lattices in the WFST framework."*