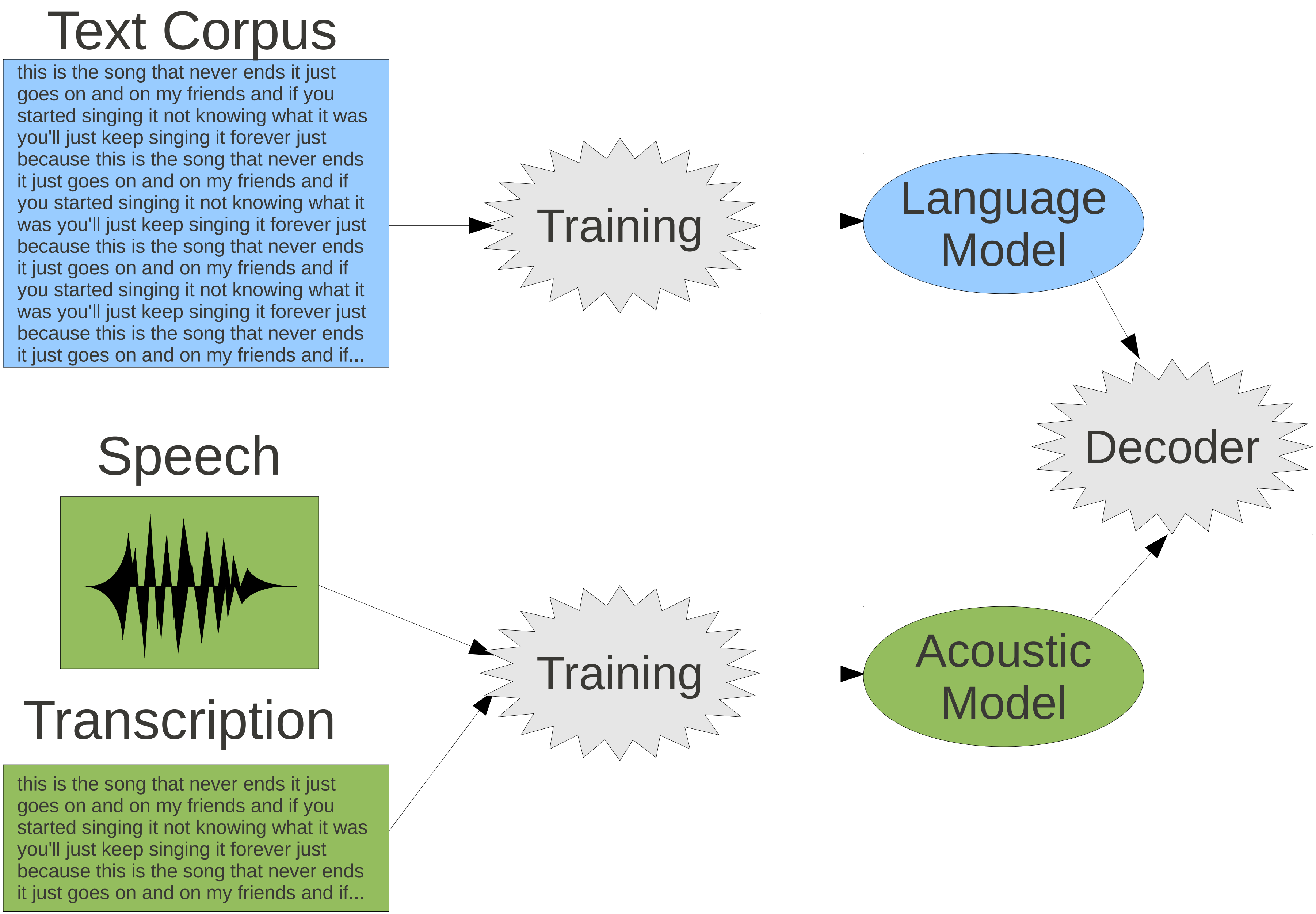
Learning a Language Model from Continuous Speech

Graham Neubig, Masato Mimura,
Shinsuke Mori, Tatsuya Kawahara

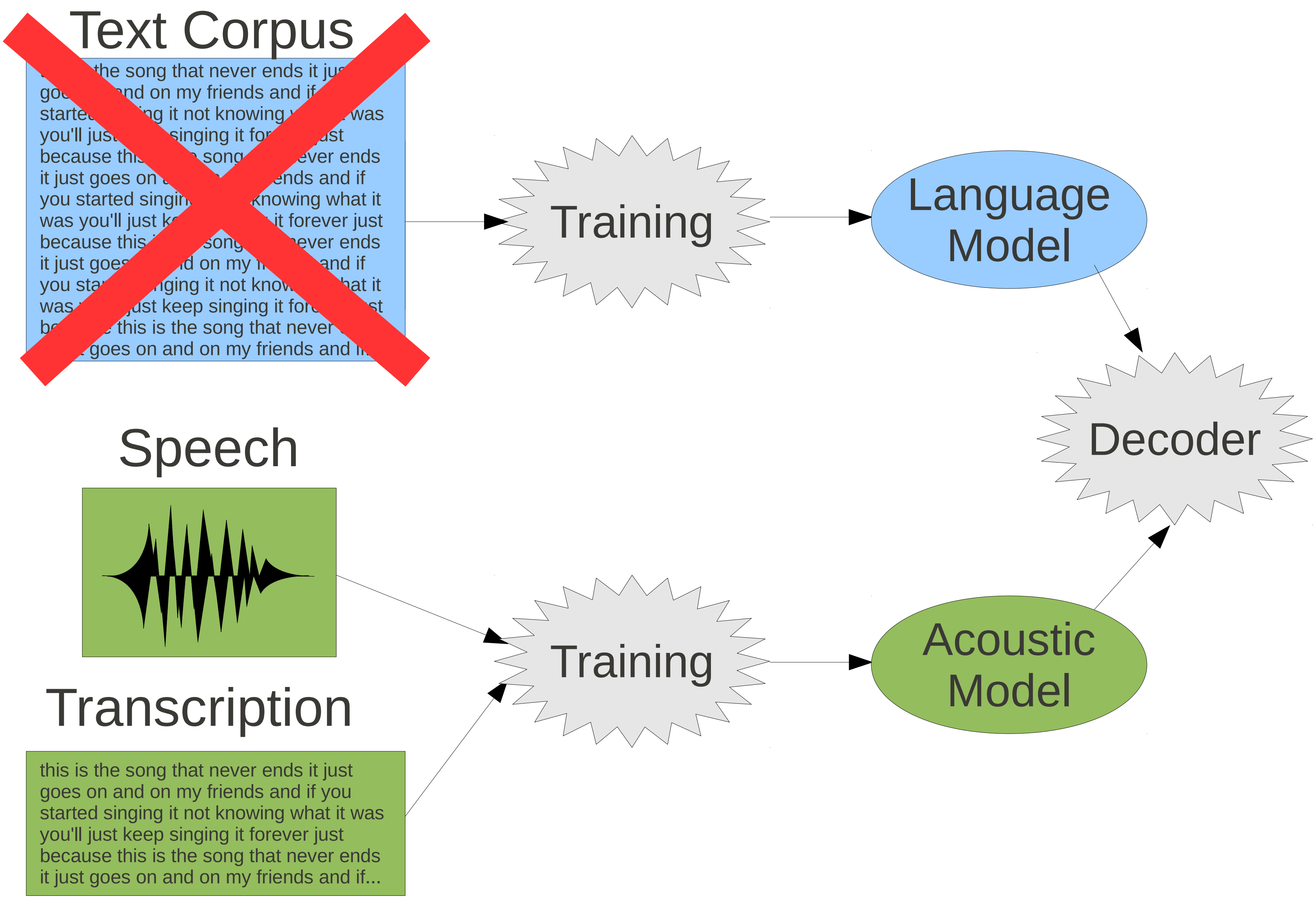
School of Informatics, Kyoto University, Japan

1. Outline

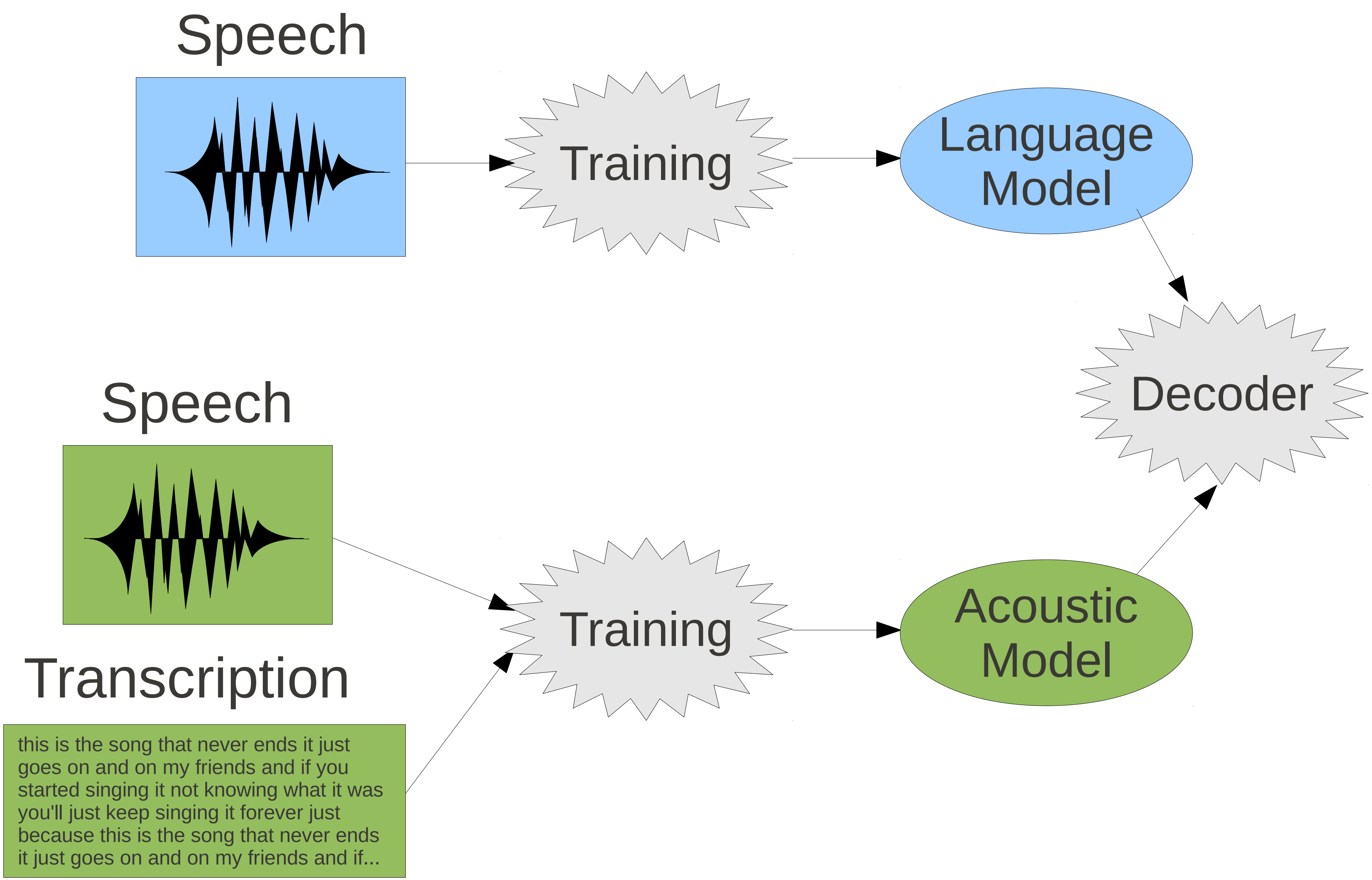
Training of a Speech Recognition System



Training of a Speech Recognition System



Training of a Speech Recongition System



Why Learn a Language Model from Speech?

- A straightforward way to handle spoken language
  - Fillers, colloquial expressions, and pronunciation variants are included in the model
- A way to learn models for resource-poor languages
  - LMs can be learned even for languages with **no** digitized text
  - Use with language-independent acoustic models? [Schultz & Waibel 01]
- Semi-supervised Learning
  - Learn a model from newspaper text, update it with spoken expressions or new vocabulary from speech

Our Research

- Goal: Learn a LM using no text
- Two problems:
  - Word boundaries are not clear → use unsupervised word segmentation
  - Acoustic ambiguity→Use a phoneme lattice to absorb acoustic model errors
- Method: Apply a Bayesian word segmentation method [Mochihashi+ 09] to phoneme lattices
  - Implementation using weighted finite state transducers (WFST)
- Result: An LM learned from continuous speech was able to significantly reduce the ASR phoneme error rate on test data

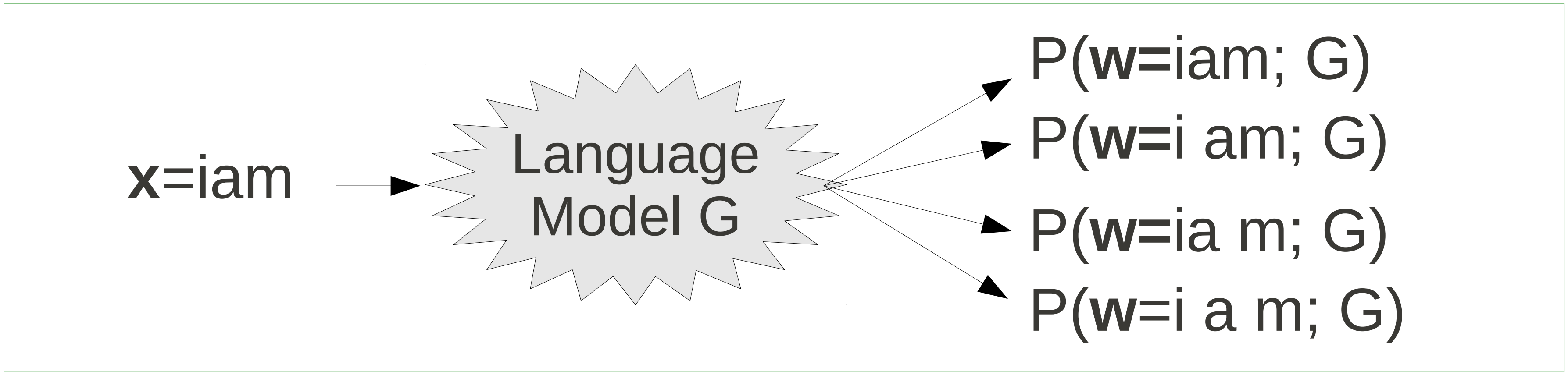
Previous Research

- Learning words from speech
  - Using audio/visual data and techniques such as MMI or MDL, learn grounded words [Roy+ 02, Taguchi+ 09]
  - Find similar audio segments using dynamic time warping and acoustic similarity scores [Park+ 08]
- Learning language models from speech
  - Use standard LM learning techniques on 1-best AM results [de Marcken 95, Gorin+ 99]
  - Multigram model from acoustic lattices [Driesen+ 08]
- No research learning n-gram LMs with acoustic uncertainty
- Most work handles small vocabulary (infant directed speech, digit recognition)

2. Unsupervised word segmentation

LM-based Supervised Word Segmentation

- Training: Use corpus W that is annotated with word boundaries to train model G
- Decoding: for character sequence **x**, treat all word sequences **w** as possible candidates
  - The probability of a candidate is proportional to its LM probability



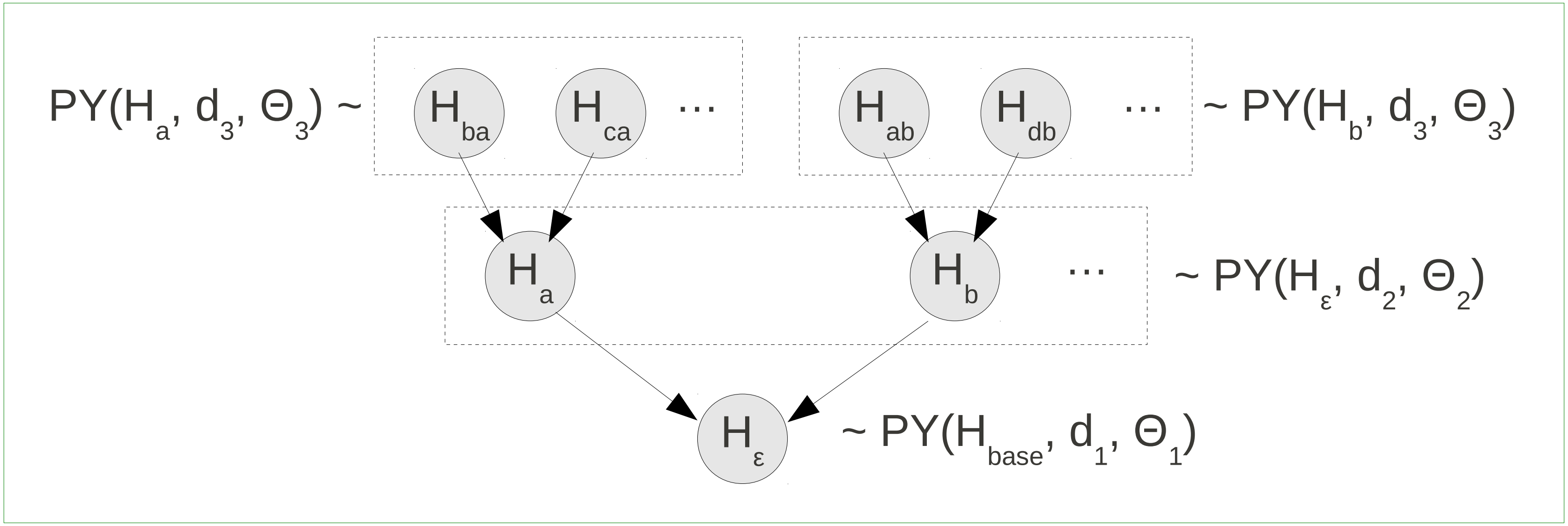
LM-Based Unsupervised Word Segmentation

- Estimate an unobserved word sequence W of unsegmented corpus X, train language model G over W
- We desire a model that is highly expressive, but simple
  - Likelihood P(W|G) prefers expressive (complex) models
  - Add a prior P(G) that prefers simple models
  - Find a model with high joint probability P(G,W)=P(G)P(W|G)

Simple Model	Ideal Model	Complex model
P(G) high	P(G) mid	P(G) low
P(W|G) low	P(W|G) mid	P(W|G) high
P(G)P(W|G) low	P(G)P(W|G) mid	P(G)P(W|G) low

Hierarchical Pitman-Yor Language Model (HPYLM) [Teh 06]

- An n-gram language model based on non-parametric Bayesian statistics
- Has a number of attractive traits
  - Language model smoothing is realized through prior P(G)
  - Parameters can be learned using Gibbs sampling



Unsupervised Word Segmentation using HPYLMs [Mochihashi+ 09]

- The model G is separated into a word-based language model LM and a character-based spelling model SM
  - Words and spellings are connected in a probabilistic framework (unknown words can be modeled)

i am in chiba now

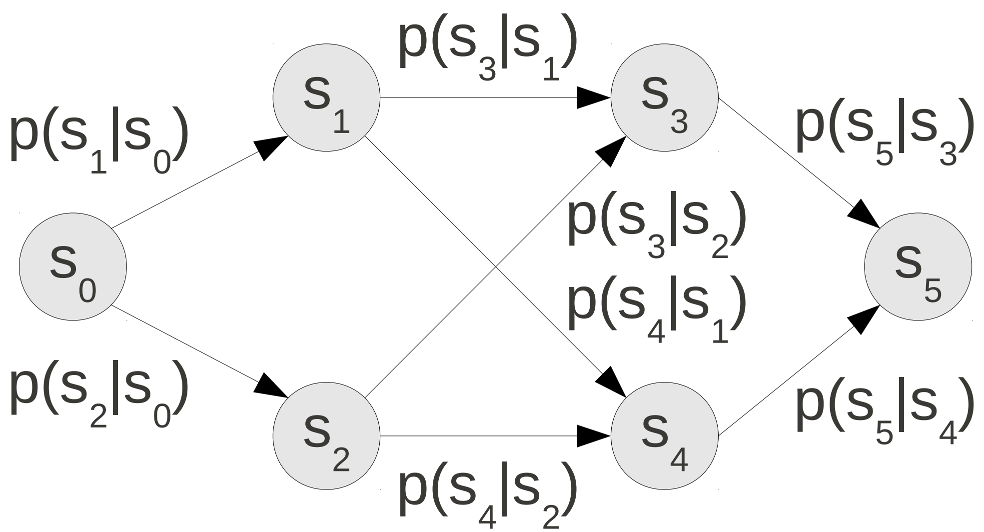
$P_{LM}(i|<s>)$ $P_{LM}(am|i)$ $P_{LM}(in|am)$ $P_{LM}(<unk>|in)$ $P_{LM}(now|<unk>)$ $P_{LM}(</s>|now)$

$P_{SM}(c|<s>)$ $P_{SM}(h|c)$ $P_{SM}(i|h)$ $P_{SM}(b|i)$ $P_{SM}(a|b)$ $P_{SM}(</s>|a)$

- It is possible to sample word boundaries using a technique called forward-filtering/backward-sampling
  - Can be used with any (non-cyclic) finite-state automaton
  - Very similar to the forward-backward algorithm for HMMs

Forward Filtering

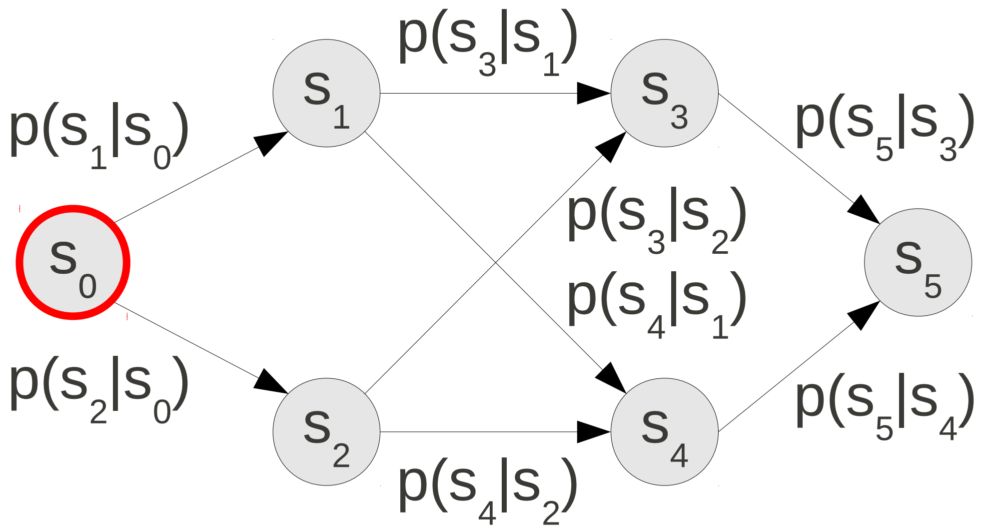
- Forward filtering is identical to the forward step in the forward-backward algorithm



forward filtering
add forward probabilities in order

Forward Filtering

- **Forward filtering** is identical to the forward step in the forward-backward algorithm

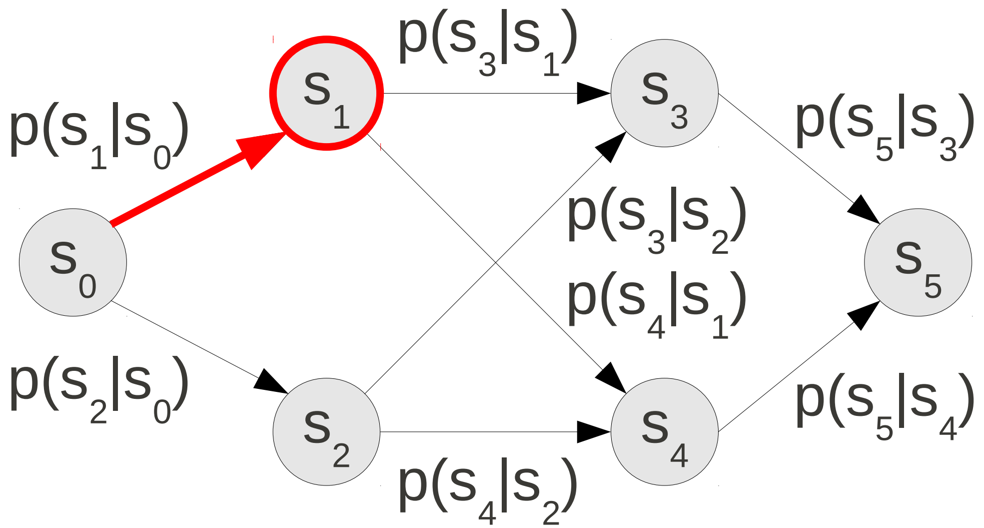


forward filtering
<u>add forward probabilities in order</u>

$f(s_0) = 1$

Forward Filtering

- Forward filtering is identical to the forward step in the forward-backward algorithm



forward filtering

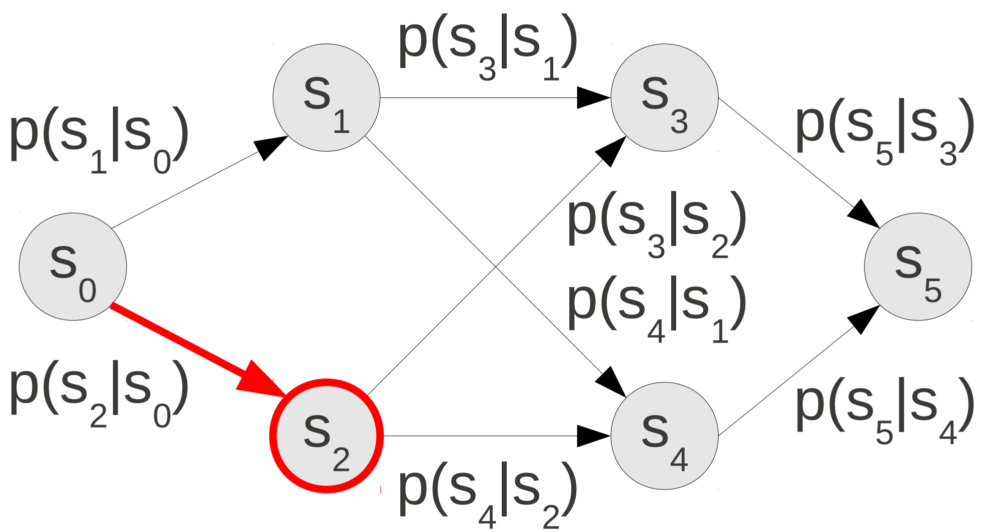
add forward probabilities in order

$f(s_0) = 1$

$f(s_1) = p(s_1|s_0)*f(s_0)$

Forward Filtering

- Forward filtering is identical to the forward step in the forward-backward algorithm



forward filtering

add forward probabilities in order

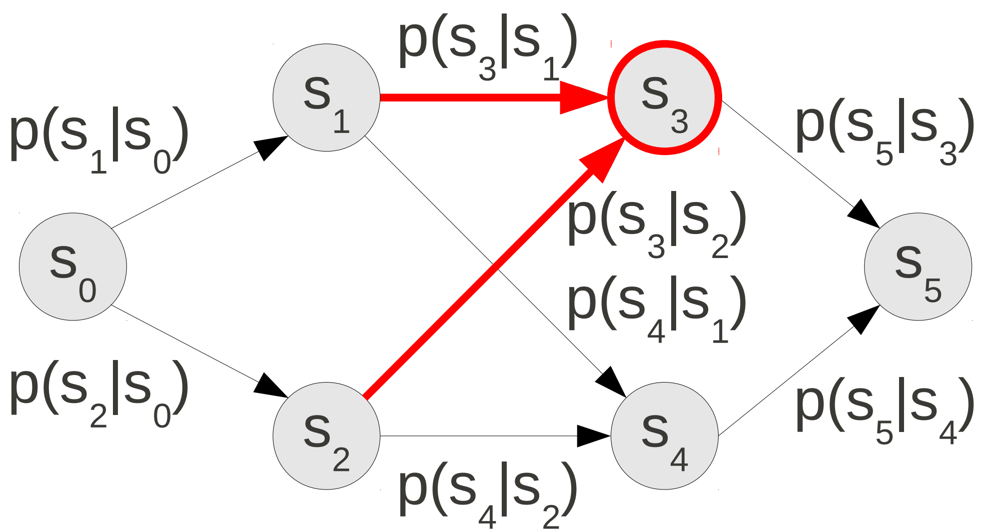
$f(s_0) = 1$

$f(s_1) = p(s_1|s_0) * f(s_0)$

$f(s_2) = p(s_2|s_0) * f(s_0)$

Forward Filtering

- Forward filtering is identical to the forward step in the forward-backward algorithm



forward filtering

add forward probabilities in order

$f(s_0) = 1$

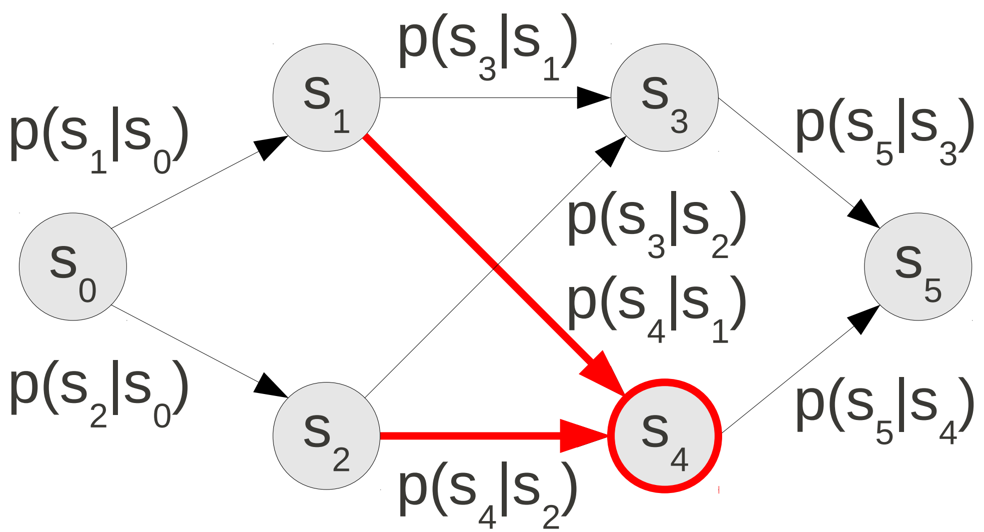
$f(s_1) = p(s_1|s_0)*f(s_0)$

$f(s_2) = p(s_2|s_0)*f(s_0)$

$f(s_3) = p(s_3|s_1)*f(s_1) + p(s_3|s_2)*f(s_2)$

Forward Filtering

- Forward filtering is identical to the forward step in the forward-backward algorithm



forward filtering

add forward probabilities in order

$f(s_0) = 1$

$f(s_1) = p(s_1|s_0)*f(s_0)$

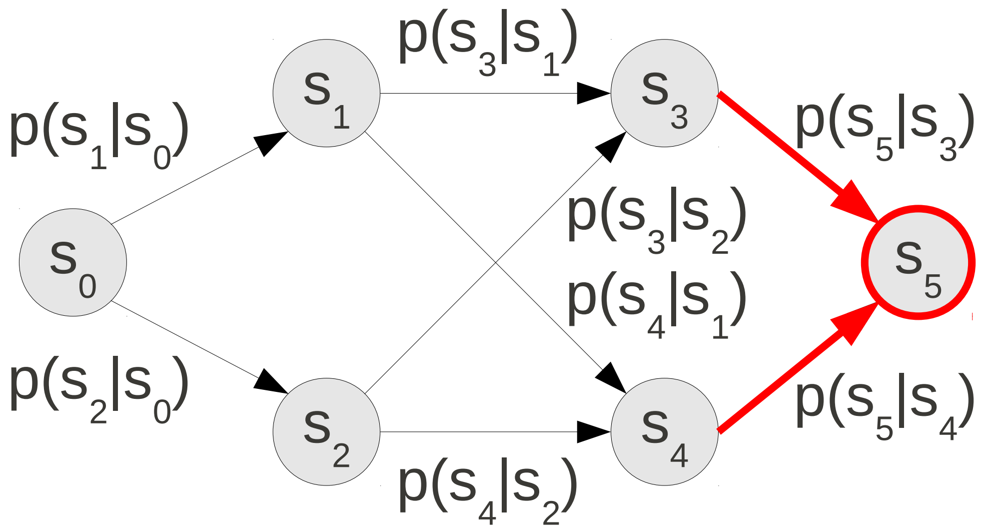
$f(s_2) = p(s_2|s_0)*f(s_0)$

$f(s_3) = p(s_3|s_1)*f(s_1) + p(s_3|s_2)*f(s_2)$

$f(s_4) = p(s_4|s_1)*f(s_1) + p(s_4|s_2)*f(s_2)$

Forward Filtering

- Forward filtering is identical to the forward step in the forward-backward algorithm



forward filtering

add forward probabilities in order

$f(s_0) = 1$

$f(s_1) = p(s_1|s_0)*f(s_0)$

$f(s_2) = p(s_2|s_0)*f(s_0)$

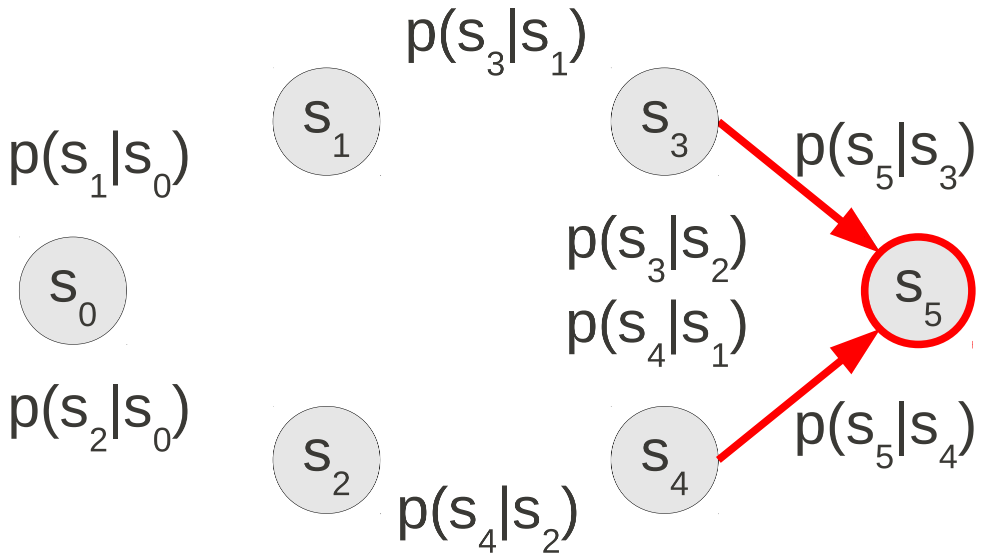
$f(s_3) = p(s_3|s_1)*f(s_1) + p(s_3|s_2)*f(s_2)$

$f(s_4) = p(s_4|s_1)*f(s_1) + p(s_4|s_2)*f(s_2)$

$f(s_5) = p(s_5|s_3)*f(s_3) + p(s_5|s_4)*f(s_4)$

Backward Sampling

- Backward sampling samples a path, starting at the final state, using the edge and forward probabilities



backward sampling
<u>sample edges from the final state</u>

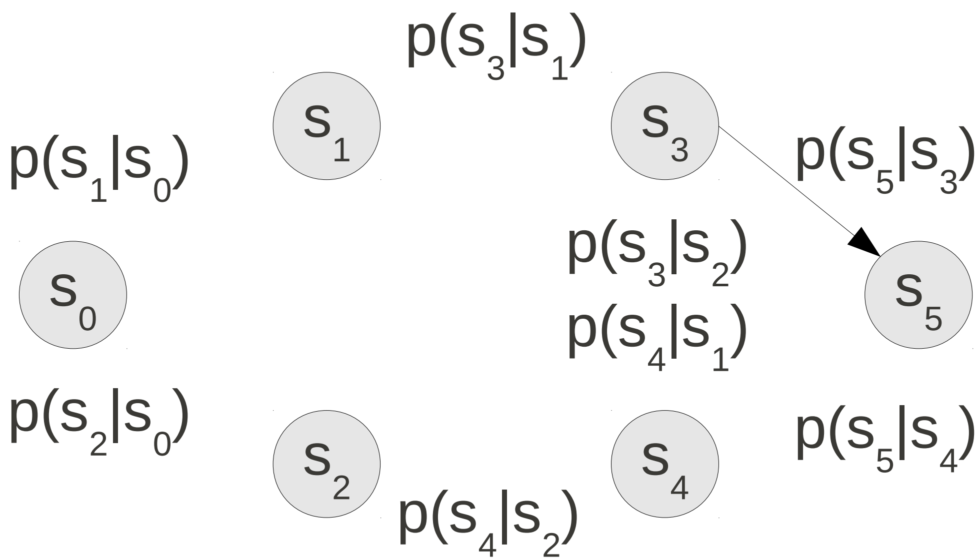
$e(s_5 \rightarrow x)$

$p(x=s_3) \propto p(s_5|s_3)*f(s_3)$

$p(x=s_4) \propto p(s_5|s_4)*f(s_4)$

Backward Sampling

- Backward sampling samples a path, starting at the final state, using the edge and forward probabilities



backward sampling

<u>sample edges from the final state</u>

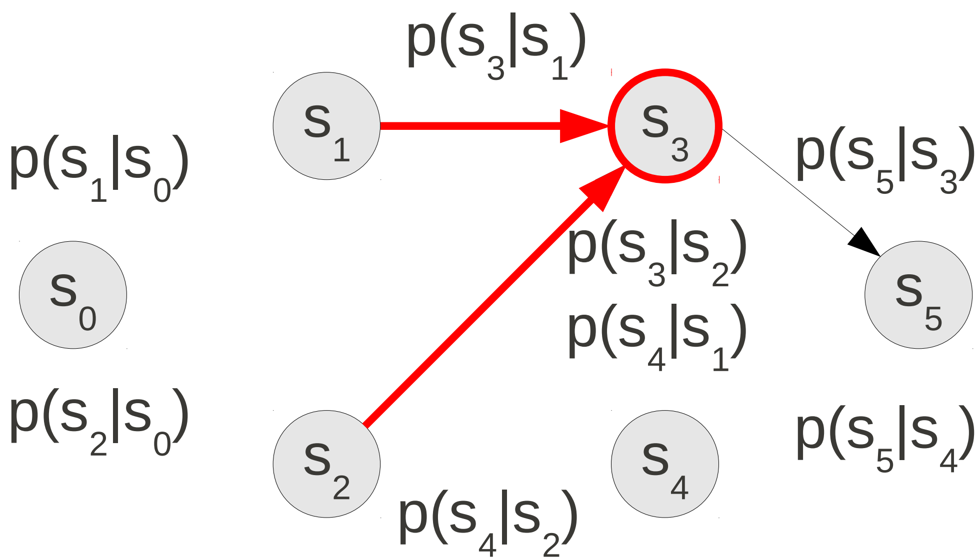
$e(s_5 \rightarrow x)$

$p(x=s_3) \propto p(s_5|s_3)*f(s_3)$

$p(x=s_4) \propto p(s_5|s_4)*f(s_4)$

Backward Sampling

- Backward sampling samples a path, starting at the final state, using the edge and forward probabilities



backward sampling

<u>sample edges from the final state</u>

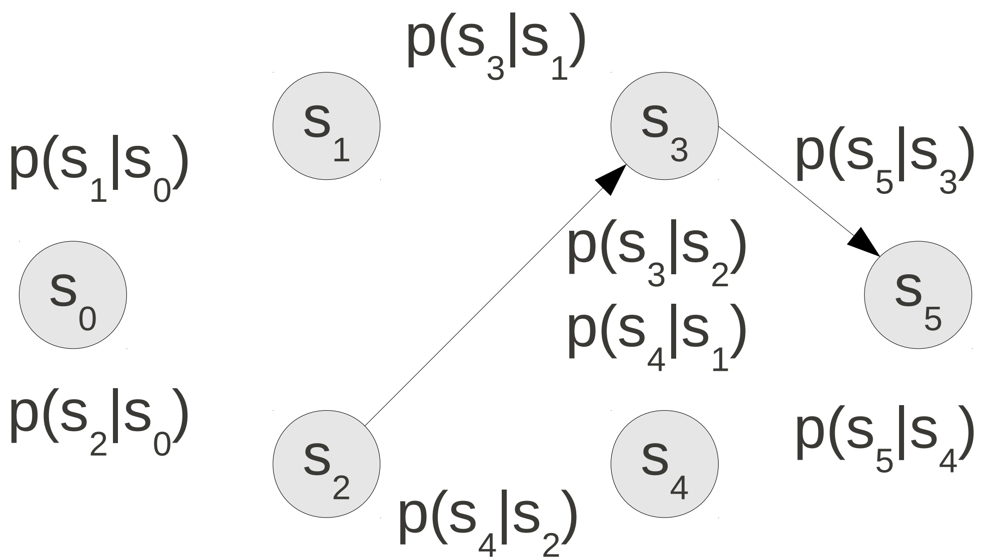
$e(s_3 \rightarrow x)$

$p(x=s_1) \propto p(s_3|s_1)*f(s_1)$

$p(x=s_2) \propto p(s_3|s_2)*f(s_2)$

Backward Sampling

- Backward sampling samples a path, starting at the final state, using the edge and forward probabilities

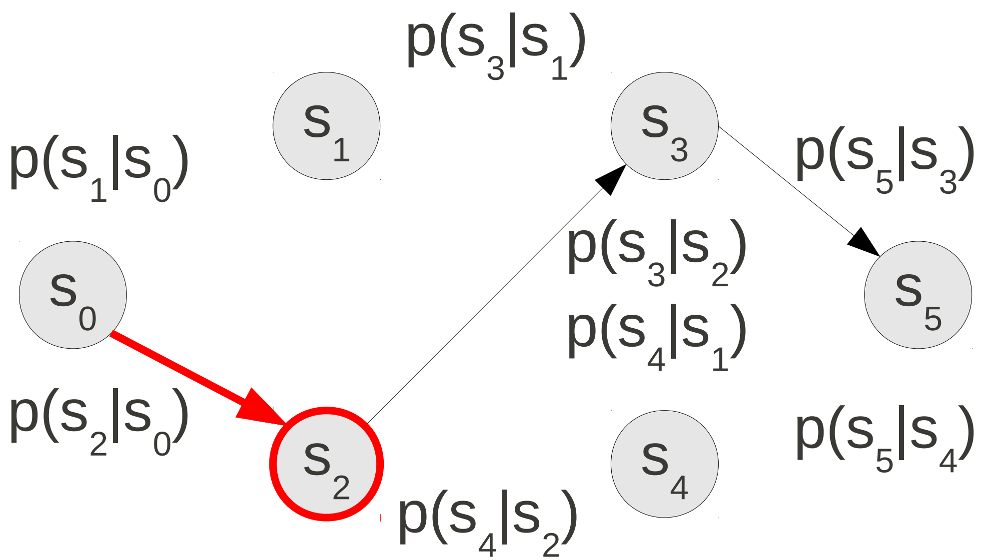


backward sampling

sample edges from the final state

Backward Sampling

- Backward sampling samples a path, starting at the final state, using the edge and forward probabilities

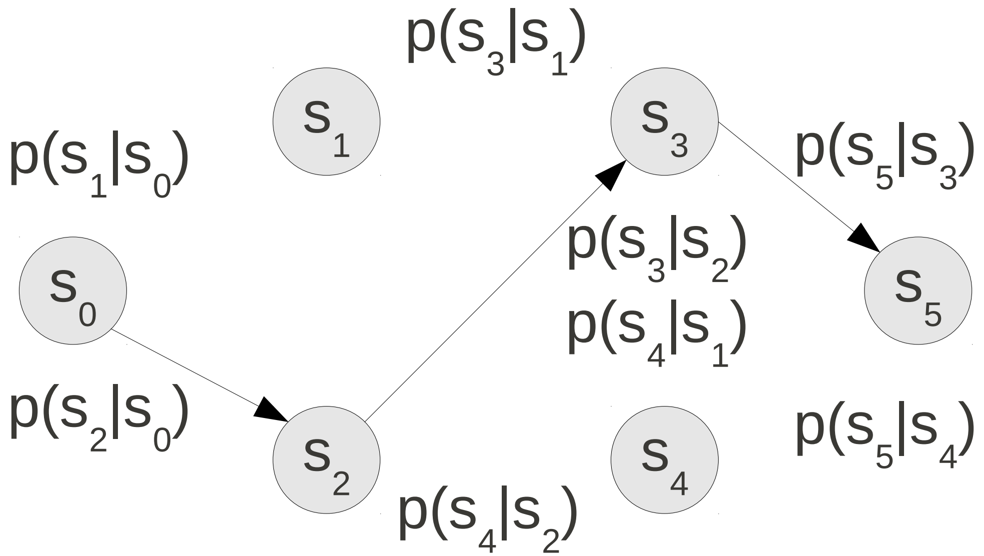


backward sampling

sample edges from the final state

Backward Sampling

- Backward sampling samples a path, starting at the final state, using the edge and forward probabilities



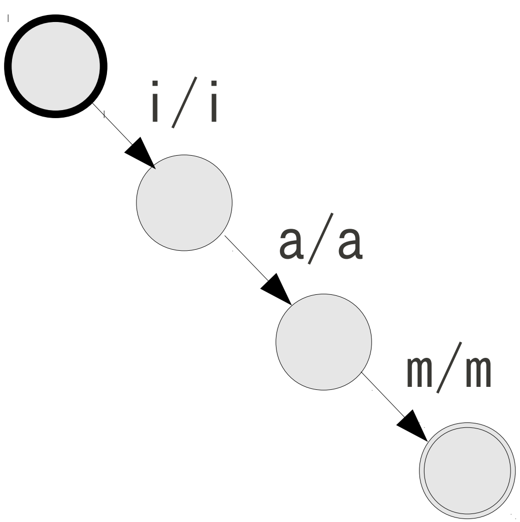
backward sampling
sample edges from the final state

3. WFST Implementation and Learning from Speech

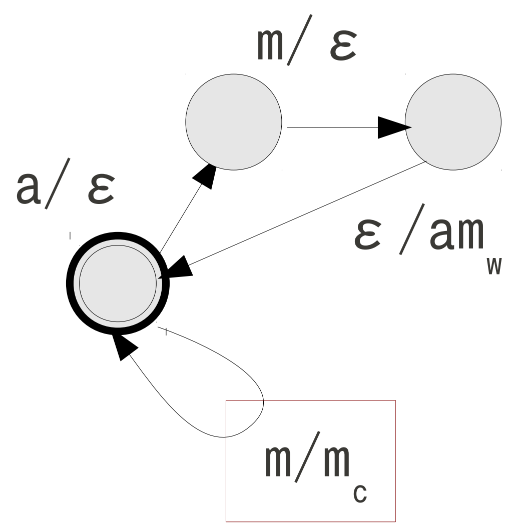
Generating Word Segmentation Candidates with WFSTs

- We propose a simple way to generate word segmentation candidates using WFSTs
- The WFSTs are quite similar to those used in ASR

Input X



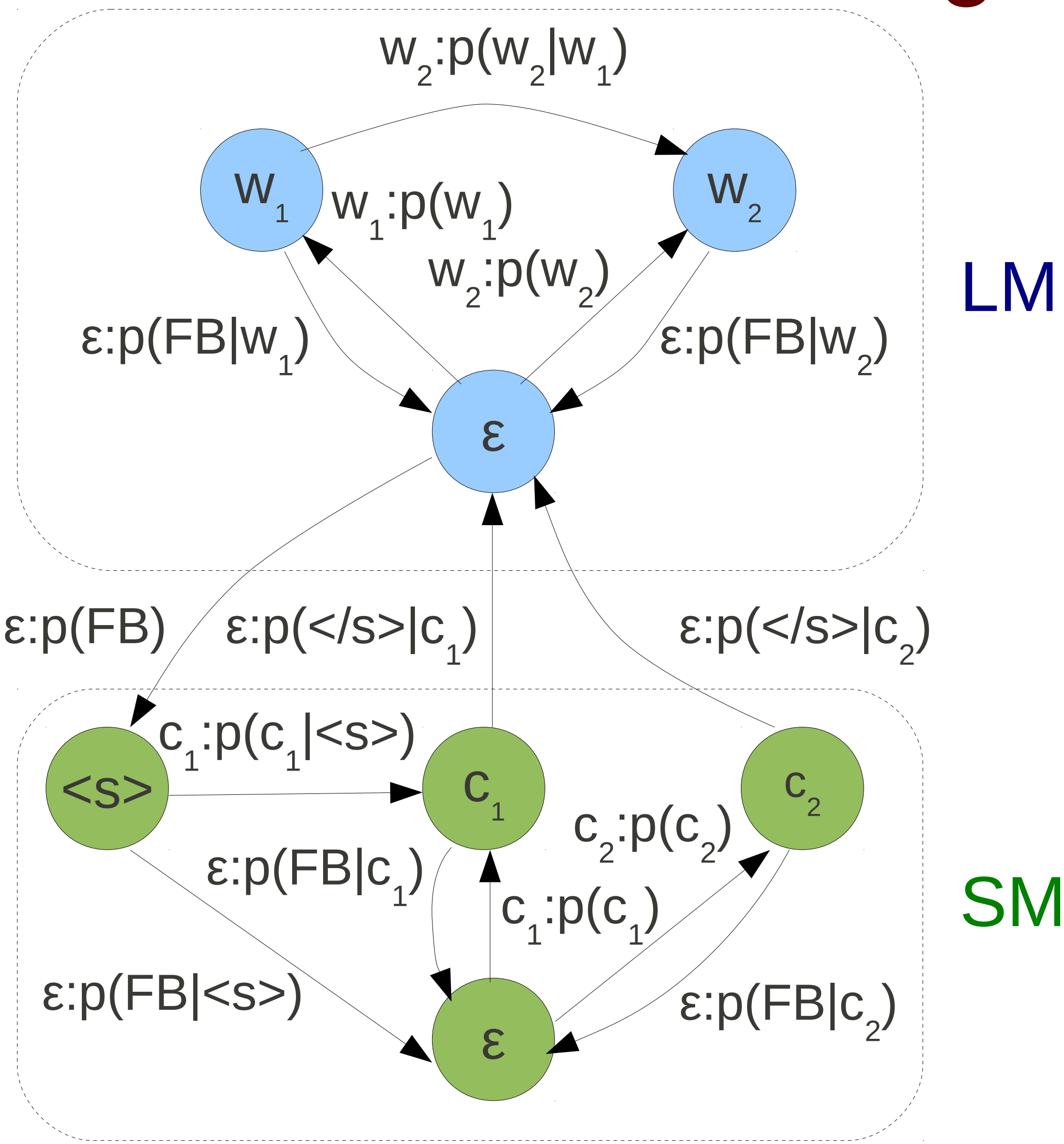
Dictionary L



LM + SM

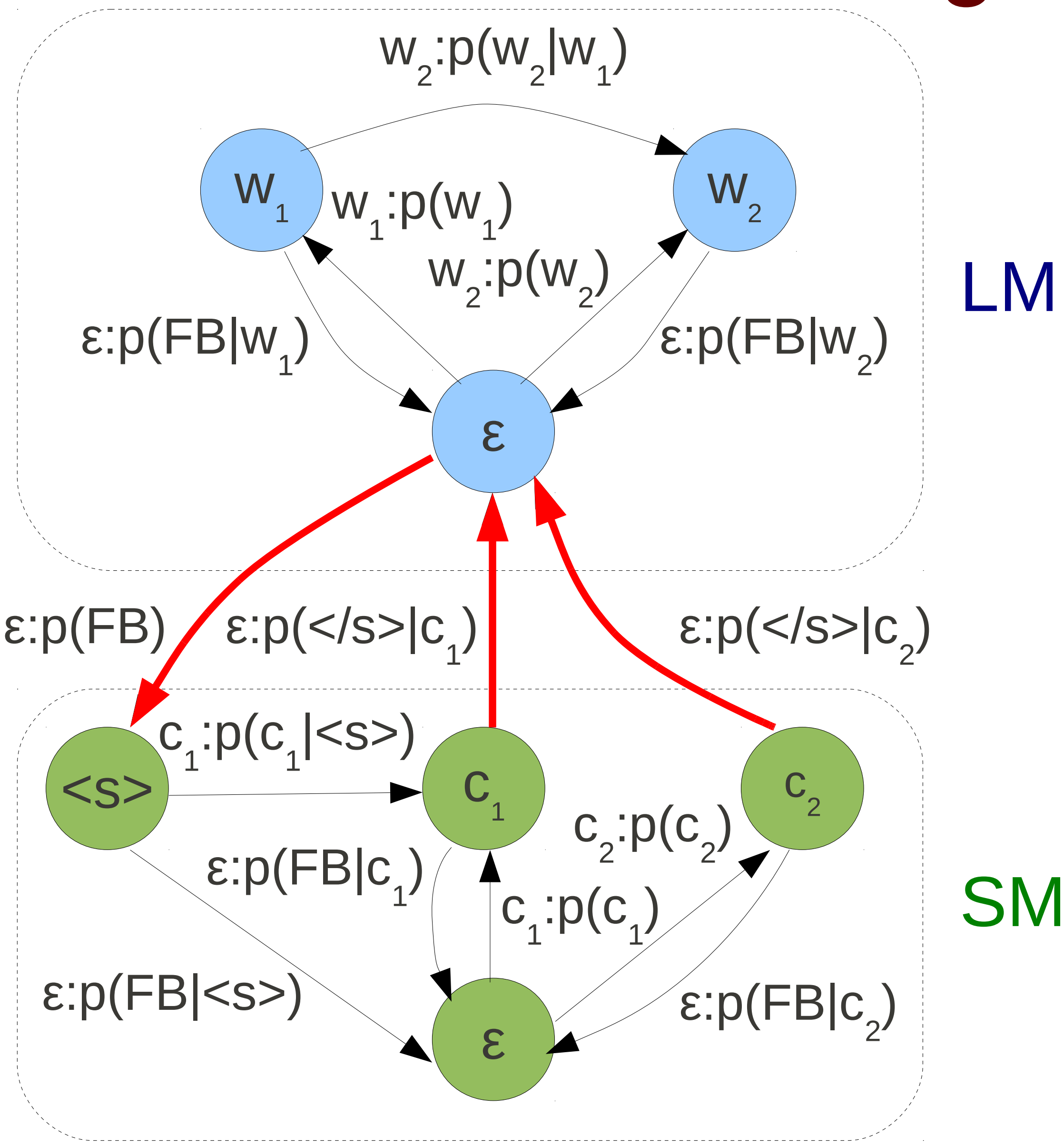
Next →

A Language Model WFST for Word Segmentation



- Express both the Language Model LM and Spelling Model SM as a single WFST

A Language Model WFST for Word Segmentation

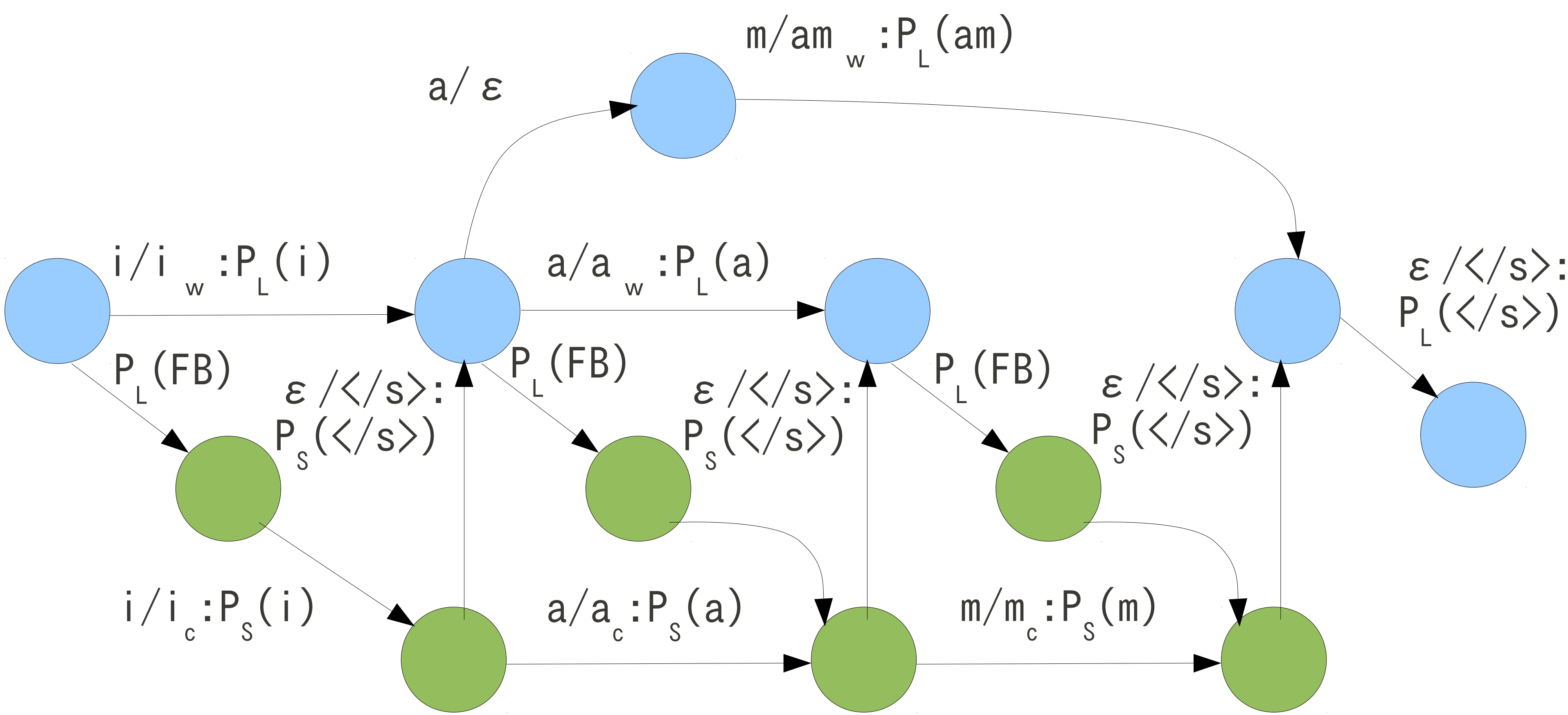


- Express both the Language Model LM and Spelling Model SM as a single WFST

The key is the weighted edges connecting the two models

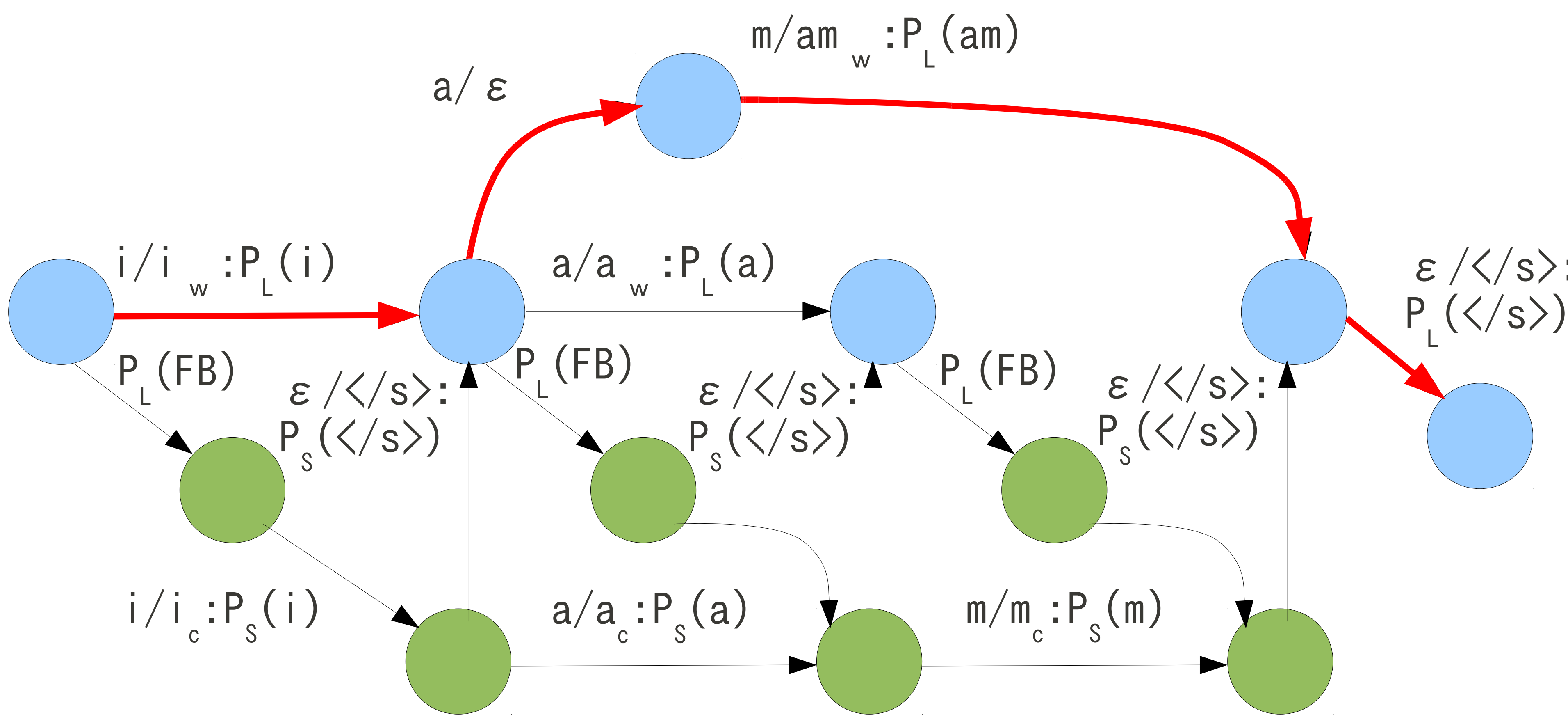
Word Segmentation Candidates as a WFST

- Vocabulary “i, a, am”, unigram model



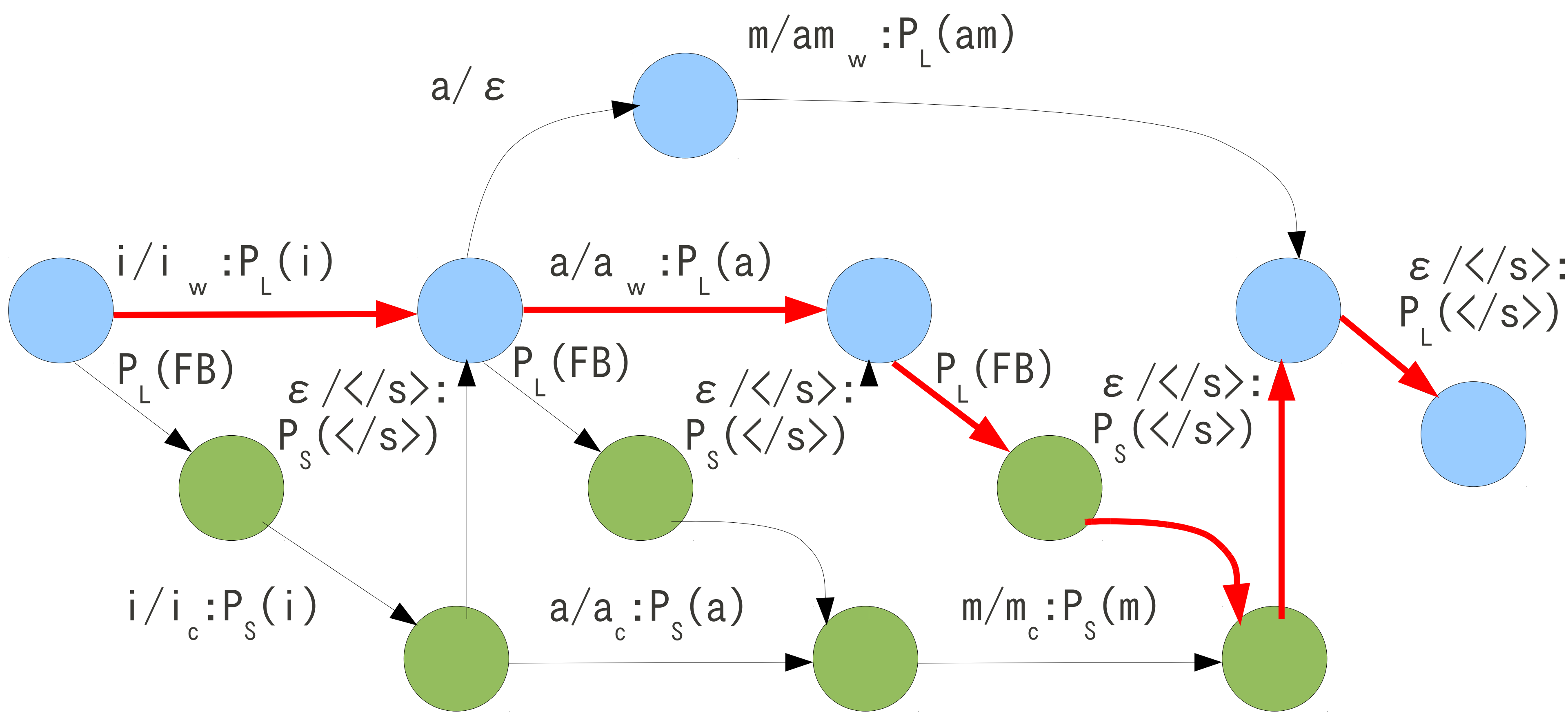
Word Segmentation Candidates as a WFST

- Vocabulary “i, a, am”, unigram model



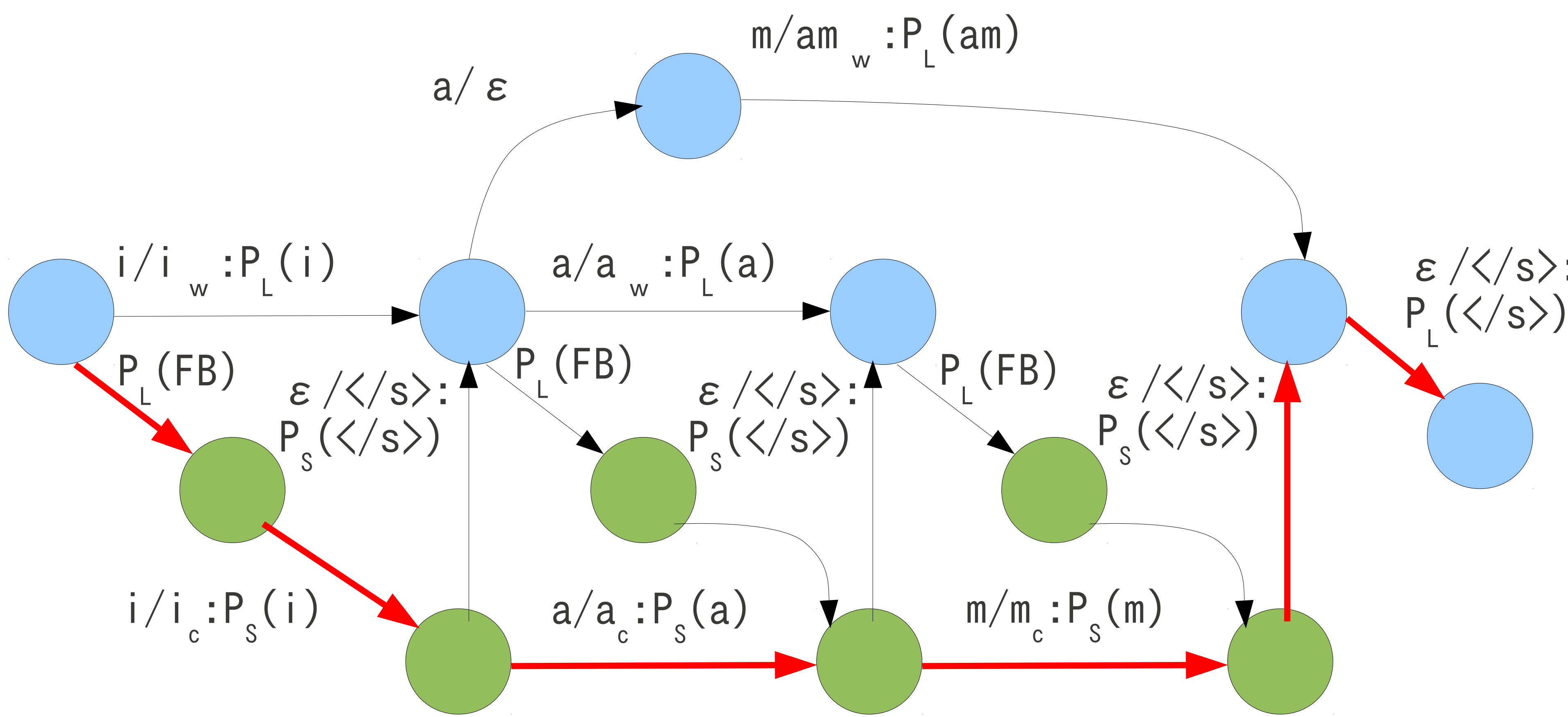
Word Segmentation Candidates as a WFST

- Vocabulary “i, a, am”, unigram model



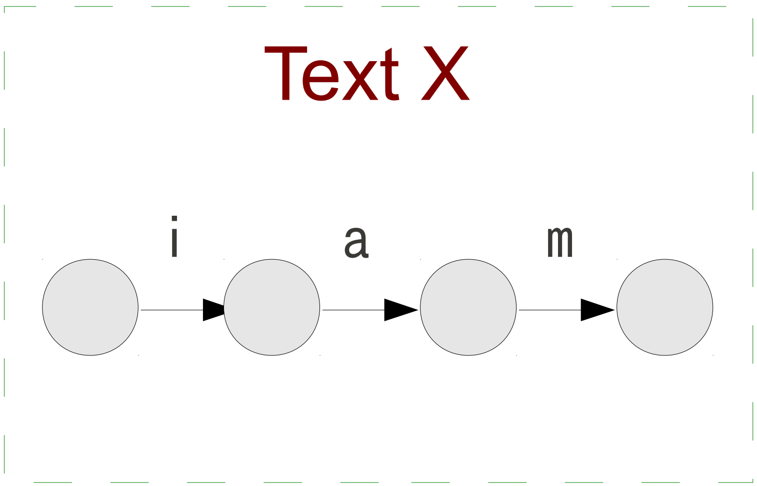
Word Segmentation Candidates as a WFST

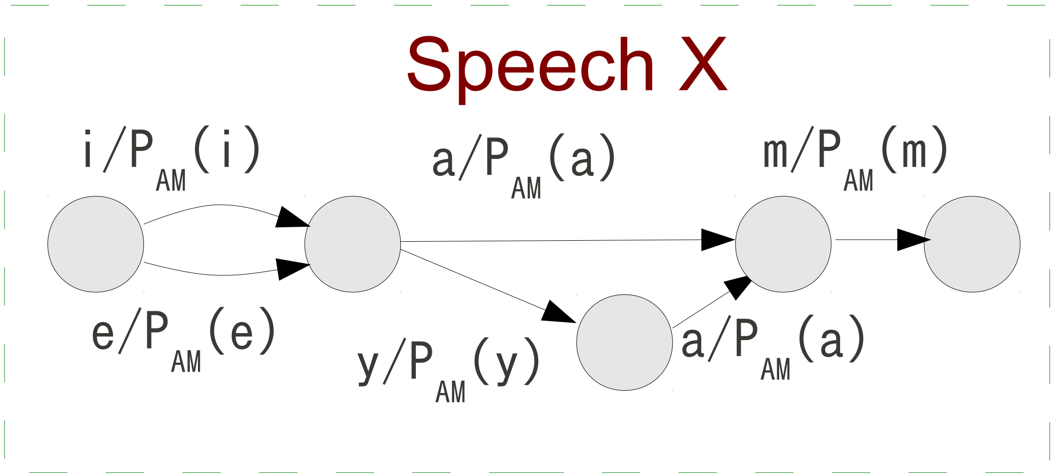
- Vocabulary “i, a, am”, unigram model



Adaptation to Speech

- When using WFSTs, adaptation to speech is simple
- Replace input X with a HMM-based acoustic model
  - Forward-filtering = creation of a recognition lattice
- However, full expansion using HMMs is impossible
  - Instead, we use a trimmed phoneme lattice with acoustic model scores





Learning from Text, Learning from Speech

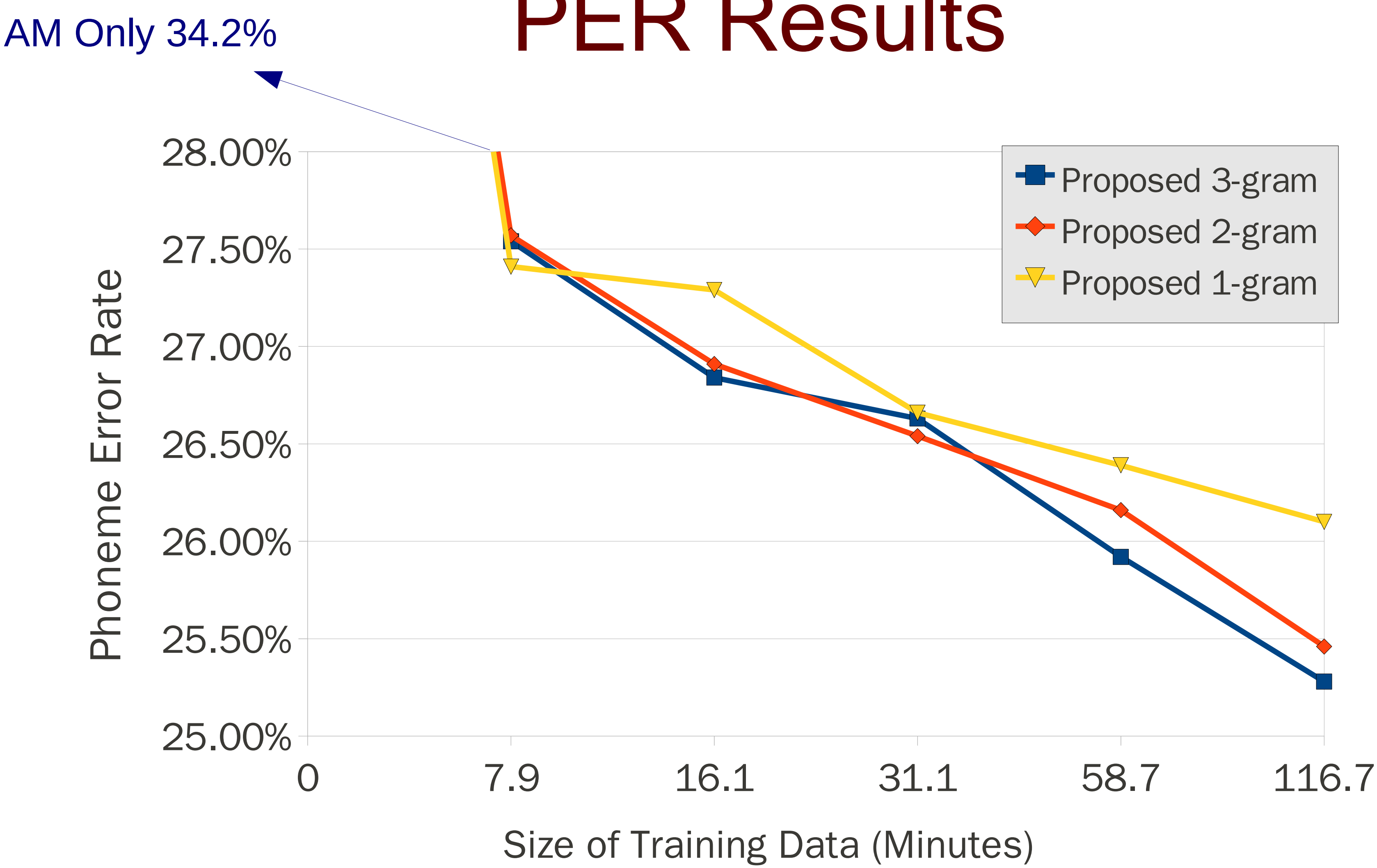
	Text	Speech
Input	Character String	Phoneme Lattice
Technique	WFST Composition, Sampling	WFST Composition, Sampling
Probability	$P(W\|G)P(G)$ (LM Likelihood, Prior)	$P(X\|W)P(W\|G)P(G)$ (AM, LM Likelihoods, Prior)
Samples	Segmentation, LM	Phoneme String for Each Utterance, Segmentation, LM

4. Evaluation

Experimental Setting

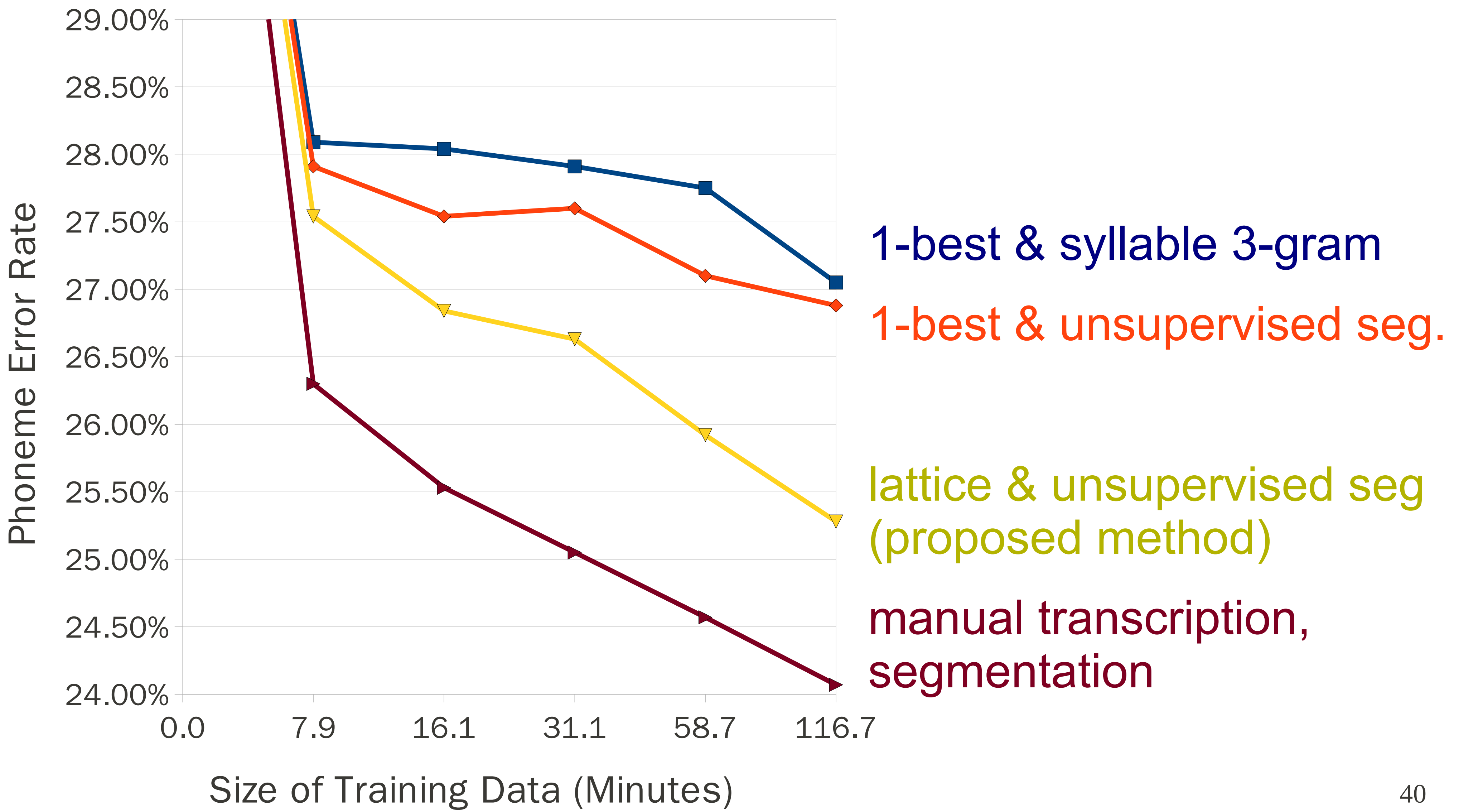
- Target: Speech from meetings of the Japanese diet
  - Fluent, large-vocabulary speech
  - Actual vocabulary size is 2858 words
- Data preparation: triphone acoustic model
  - PER: one-best 34.2%, oracle 8.1%
  - Used syllable lattices, not phoneme lattices (due to requirements of the decoder)
  - 8-117 minutes of training data, 27 minutes of test data
- Evaluation standard:
  - Phoneme error rate over the test data using language model learned from training speech

PER Results



- An LM learned from continuous speech reduced the PER by 8.92%
- 3-gram is better than 1-gram: learned contextual info

Other Training Methods



5. Conclusion

Conclusion

- We demonstrated that it is possible to learn a language model from continuous speech

Released open source

http://www.phontron.com/latticelm

- A number of potential applications
  - Learning language models and dictionaries for resource-poor languages
  - Elegant handling of spoken language
  - Semi-supervised learning

Thank You

Extra Slides

Vocabulary/Model Complexity

	1-gram	2-gram	3-gram	Gold Standard 3-gram
Vocabulary	4480	1351	708	2858
Average Word Length (Syl.)	2.03	1.37	1.18	1.73
Language Model States	4480	16150	38759	34073
Spelling Model States	9624	3869	2426	8378

Words learned

Particles

word	English	# (rank)
no	*possessive*	1052 (1)
ni	*positional*	830 (2)
to	and	685 (5)

Colloquial Expressions

word	English	# (rank)
yu:	say (colloq)	324 (19)
e:	*filler*	202 (28)
desune	*discourse marker*	94 (65)

rimasukeredomo, mo:shiage, yu:fu:ni

Subwords

word	English	# (rank)
ka	*particle, subword*	713 (3)
to:	*subword*	204 (27)
sai	*subword*	94 (65)

Content/Function Words

word	English	# (rank)
koto	thing	189 (32)
omo	think (stem)	56 (109)
hanashi	speak	23 (242)

jo:kyo:, kangae, chi:ki, toki, shiteki

Experimental Setup (2)

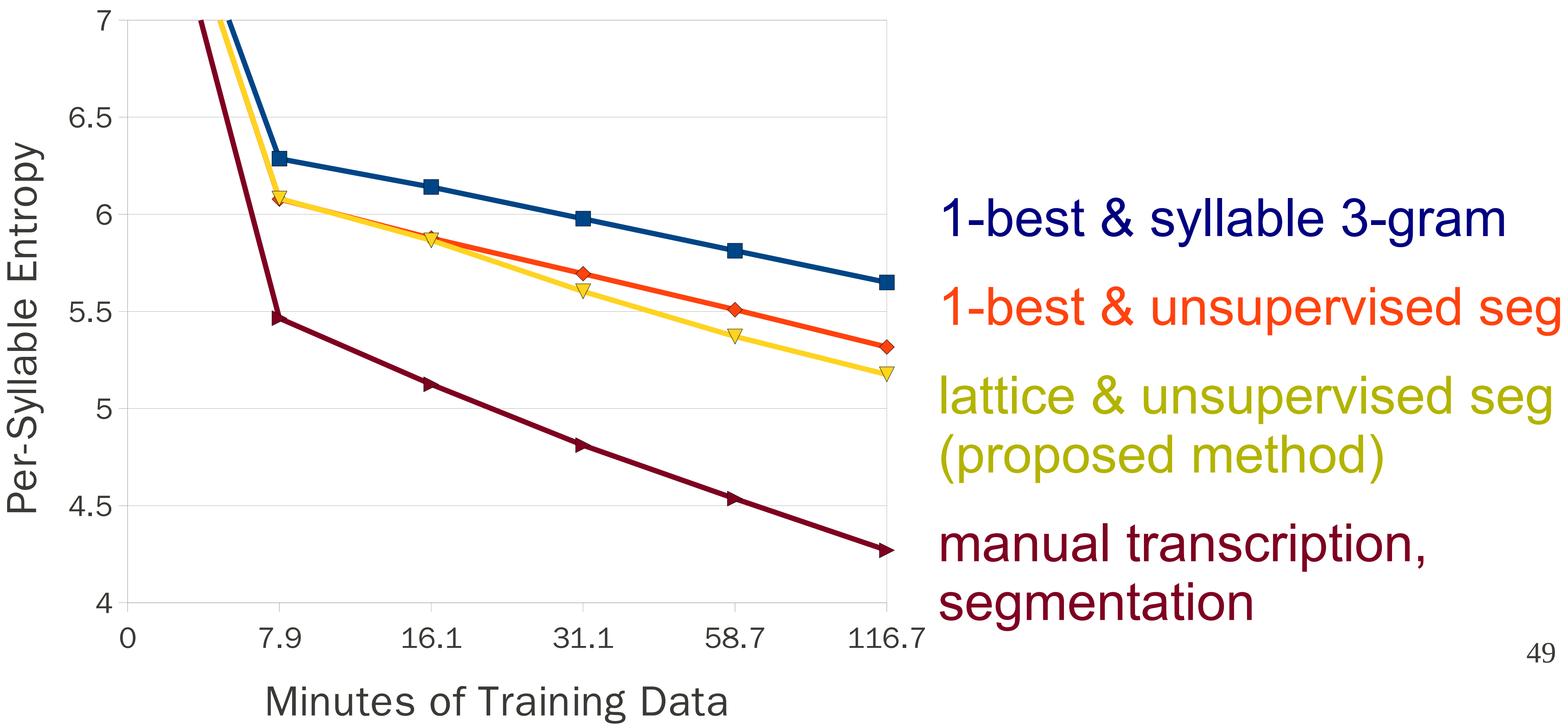
- Training:
  - 8-117 minutes of continuous speech as training data
  - 0.5-20 second utterances
  - Flat priors on hyperparameters, little influence
  - 20 samples burn-in, 50 LM samples
- Testing:
  - 27 minutes of speech separate from the training data
  - Lattice rescoring (not speech recognition)
  - Viterbi phoneme strings for each LM sample combined using ROVER

Interesting Pronunciation Variants

- nippon (Japan) → nippo:n
  - Learned with a long vowel not in the transcription
  - Extra emphasis is put on the name of the country, particularly when using nippon instead of nihon
- shiteorimasu (is doing)→ shitorimasu
  - There are many places where the speakers skip vowels
- N → *nothing*
  - Many word-final Ns are not recognized by the AM
- Perhaps taking these into account would improve AM training?

Entropy Evaluation

- Gain over 1-best is much lower, why?
  - Different pronunciations than the transcription
    shiteorimasu→shitorimasu
  - Large effect on entropy, small on PER

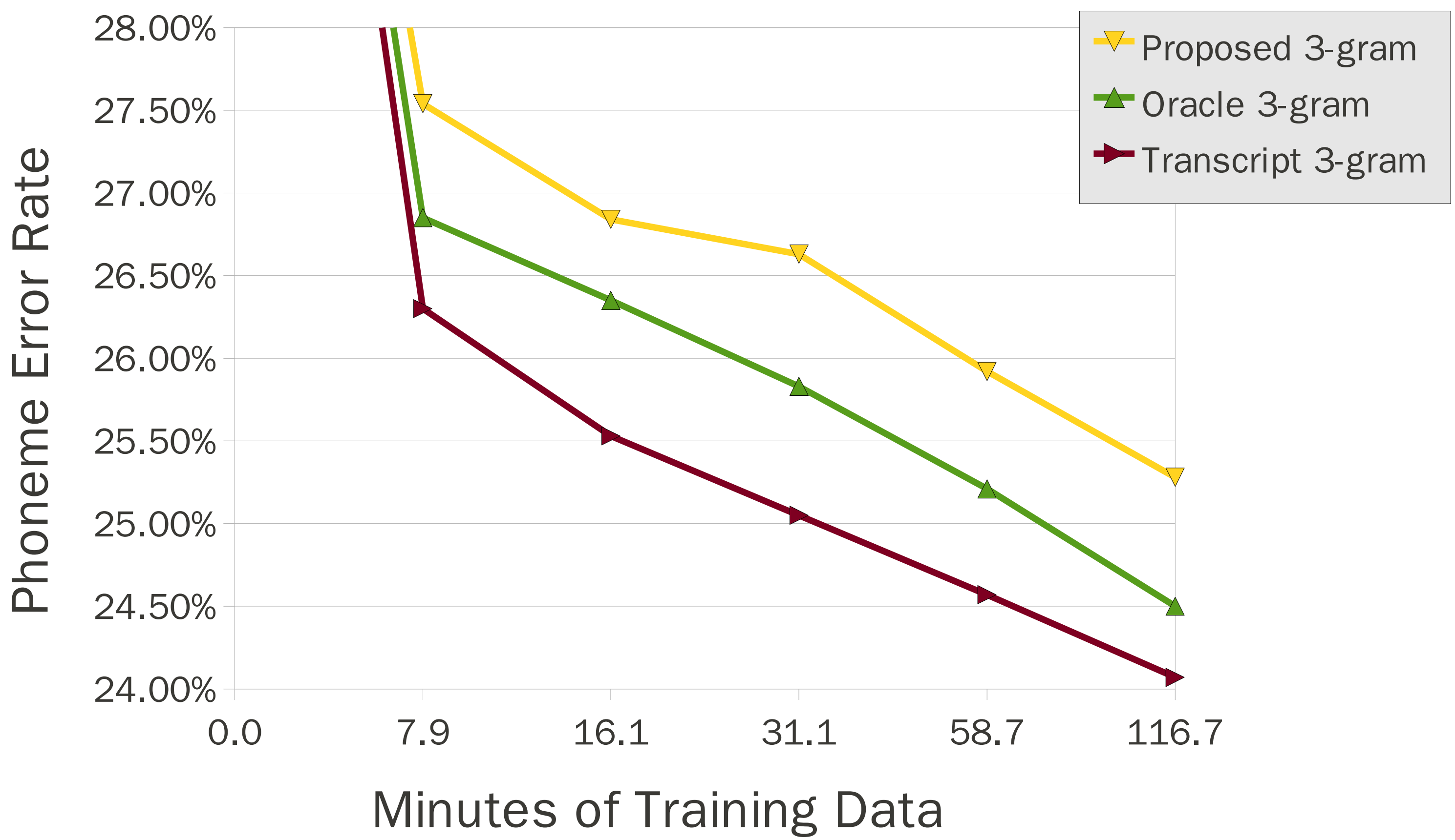


Future Work: Grounding

- The model learns a segmented phoneme string
- For transcription, use actual text
  - Grounding with a grapheme string without pronunciations (subtitles?)
  - In semi-supervised learning, phonetic pronunciations of unknown words is often sufficient
- For dialog, use semantic grounding
  - Use a robot with cameras, match images to words

Future Work: Integration with HMM

- Currently working on lattices, direct integration with HMM will give better results (for both training, testing)



Future Work: Implementation

- Speed
  - Expanding FST lattice and forward filtering take a fair amount of time
    - 0.5-1 times real time
  - Several ways for improvement
    - Perform beam-search trimming during forward filtering
    - Parallel sampling
- Open-source
  - Will be made open-source pending code clean-up
  - Goal: mid-September

Formal Modeling

- For text word segmentation, $P(X|W) = 1$, but for speech this is not the case
  - Our new objective is the joint probability of the model and acoustic features

$$P(X,W,G)=\underbrace{P(X|W)}_{\text{Acoustic Model}}\underbrace{P(W|G)}_{\text{Language Model}}\underbrace{P(G)}_{\text{Prior}}$$

- Use an acoustic model scaling factor

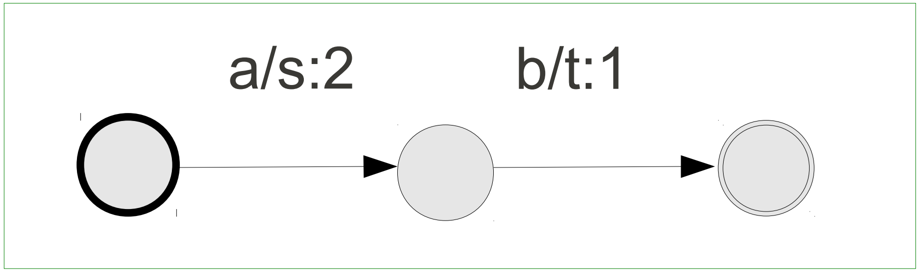
$$P(X,W,G)=P(X|W)^{\lambda}P(W|G)P(G)$$

  - Set to .2 (values between .1-.2 produced similar results)

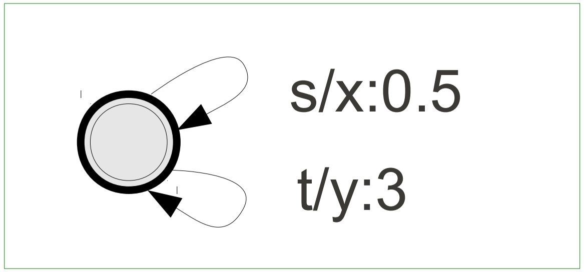
Weighted Finite State Transducers (WFSTs)

- Finite state automata with input/output/weight

A

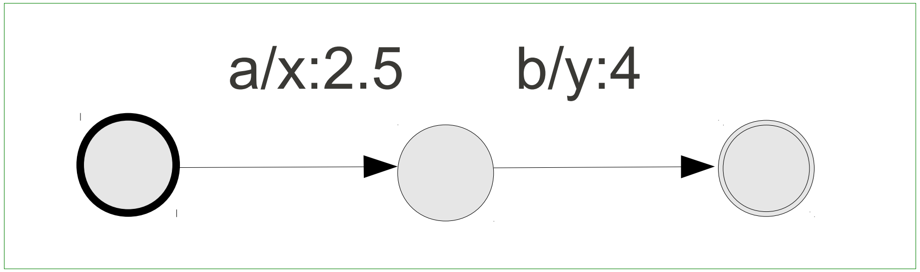


B

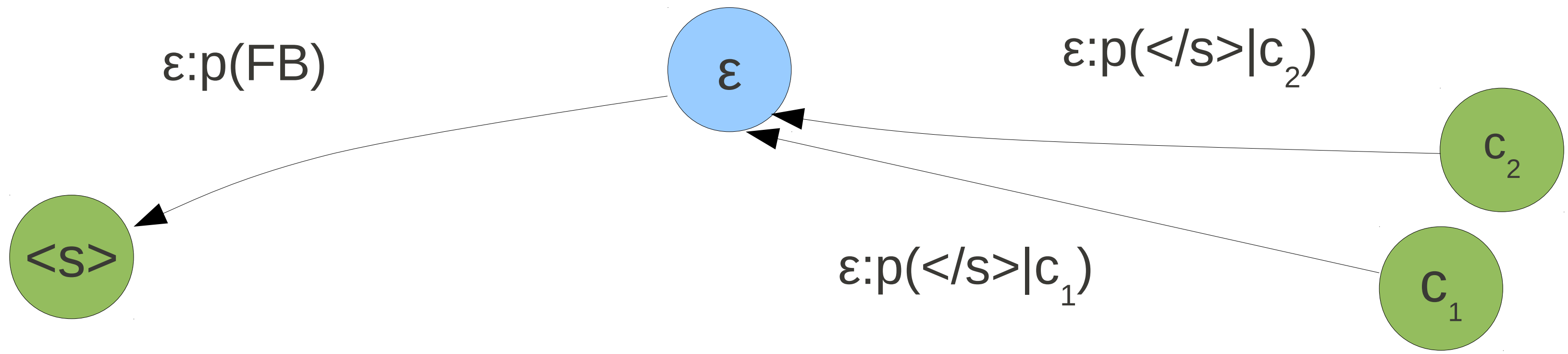


- Define weighted relations over strings
  - If weights are probabilities, probabilistic relations
- Transducers combined through composition

A○B



Connecting Edges in Detail



- To the SM from the base state
  - Equal to the probability of generating a symbol from the base distribution

- In HPYLM, n-grams with an unknown word as w_{i-1} are equal to base probabilities*

$$P(w_i|w_{i-2}, \mathrm{UNK}) = P(w_i|\mathrm{UNK}) = P(w_i)$$

- OK to make edges from the SM only to base state

* technically not true if the same word appears twice in a single sentence

Difference from Mochihashi's Method

	Mochihashi	Neubig
Spelling Model	∞-gram+ Poisson Distribution Explicit Length Limit	Character 3-gram No Length Limit
Implementation	Algorithmic Faster?	WFST-Based Simpler?, Lattice Possible
Worst-Case Complexity	$O(ML^n)$ M=Sentence length L=Max word length n=*n*-gram length	$O(M^{n+1})$
Expected Complexity	$O(ML^n)$	$O(kM+E)$ E=Number of existing word n-grams