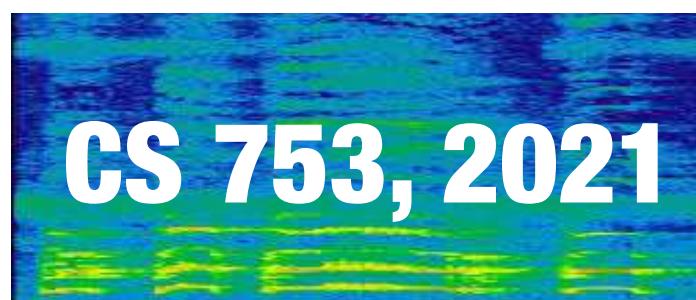


HMMs for Acoustic Modeling

Lecture 1a



Instructor: Preethi Jyothi, IITB

Recall: Statistical ASR

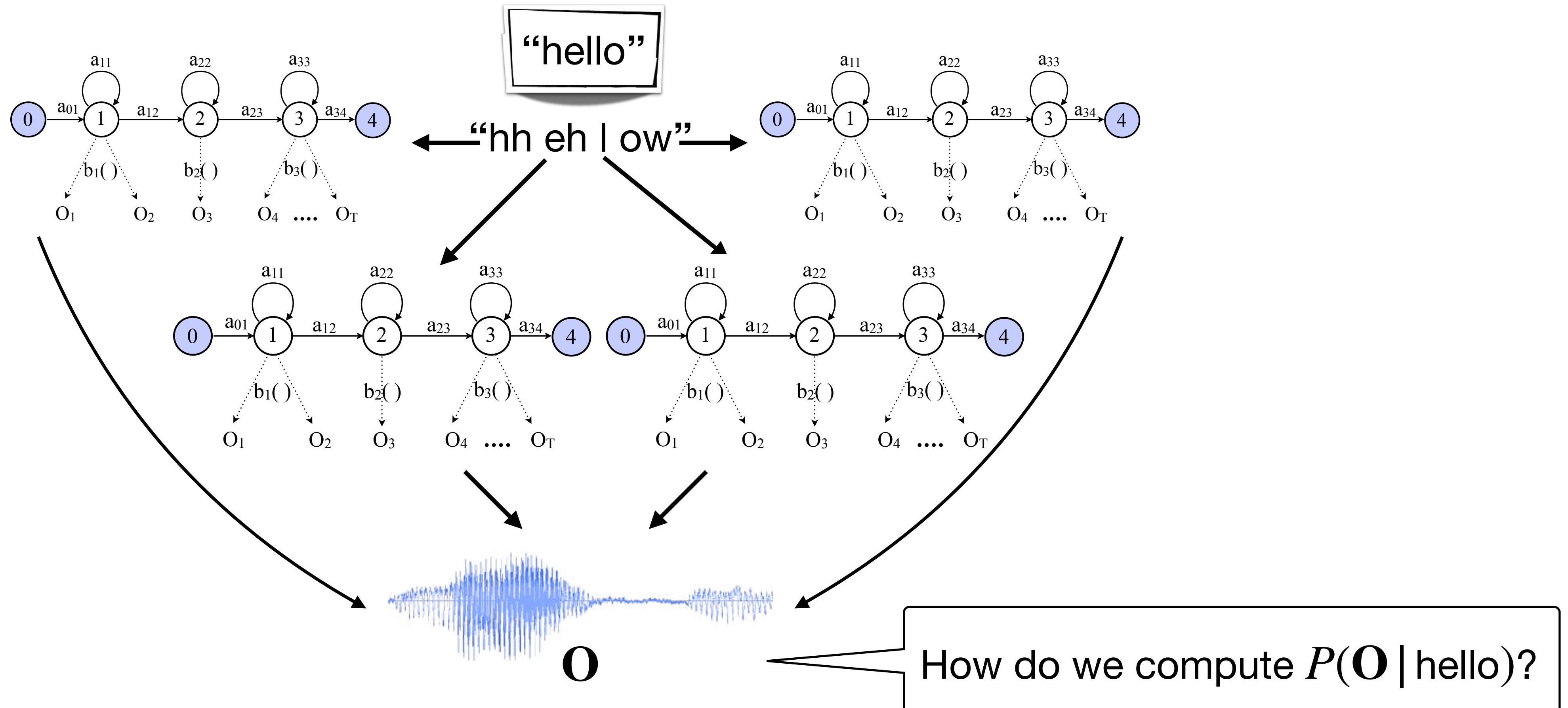
Let \mathbf{O} be a sequence of acoustic features corresponding to a speech signal. That is, $\mathbf{O} = \{O_1, \dots, O_T\}$, where $O_i \in \mathbb{R}^d$ refers to a d -dimensional acoustic feature vector and T is the length of the sequence.

Let \mathbf{W} denote a word sequence. An ASR decoder solves the following problem:

$$\begin{aligned}\mathbf{W}^* &= \arg \max_W \Pr(\mathbf{W} | \mathbf{O}) \\ &= \arg \max_W \Pr(\mathbf{O} | \mathbf{W}) \Pr(\mathbf{W}) \\ &\approx \arg \max_W \sum_Q \Pr(\mathbf{O} | Q) \Pr(Q | \mathbf{W}) \Pr(W)\end{aligned}$$

Acoustic
Model

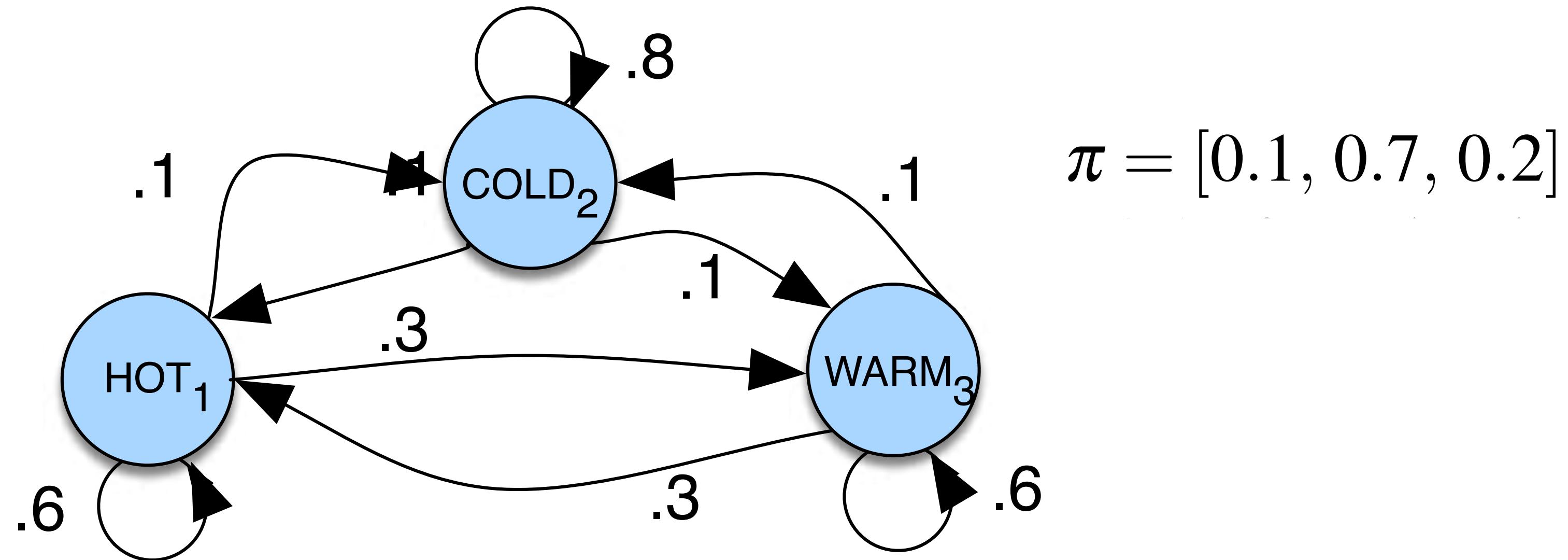
Generative Model of Speech



What are Hidden Markov Models (HMMs)?

Following slides contain figures/material from “Hidden Markov Models”,
“Speech and Language Processing”, D. Jurafsky and J. H. Martin, 2019.
(<https://web.stanford.edu/~jurafsky/slp3/A.pdf>)

Markov Chains



$Q = q_1 q_2 \dots q_N$

a set of N states

$A = a_{11} a_{12} \dots a_{n1} \dots a_{nn}$

a **transition probability matrix** A , each a_{ij} representing the probability of moving from state i to state j , s.t.
 $\sum_{j=1}^n a_{ij} = 1 \quad \forall i$

$\pi = \pi_1, \pi_2, \dots, \pi_N$

an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i . Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^n \pi_i = 1$

Hidden Markov Model

$Q = q_1 q_2 \dots q_N$

a set of N **states**

$A = a_{11} \dots a_{ij} \dots a_{NN}$

a **transition probability matrix** A , each a_{ij} representing the probability of moving from state i to state j , s.t. $\sum_{j=1}^N a_{ij} = 1 \quad \forall i$

$O = o_1 o_2 \dots o_T$

a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, \dots, v_V$

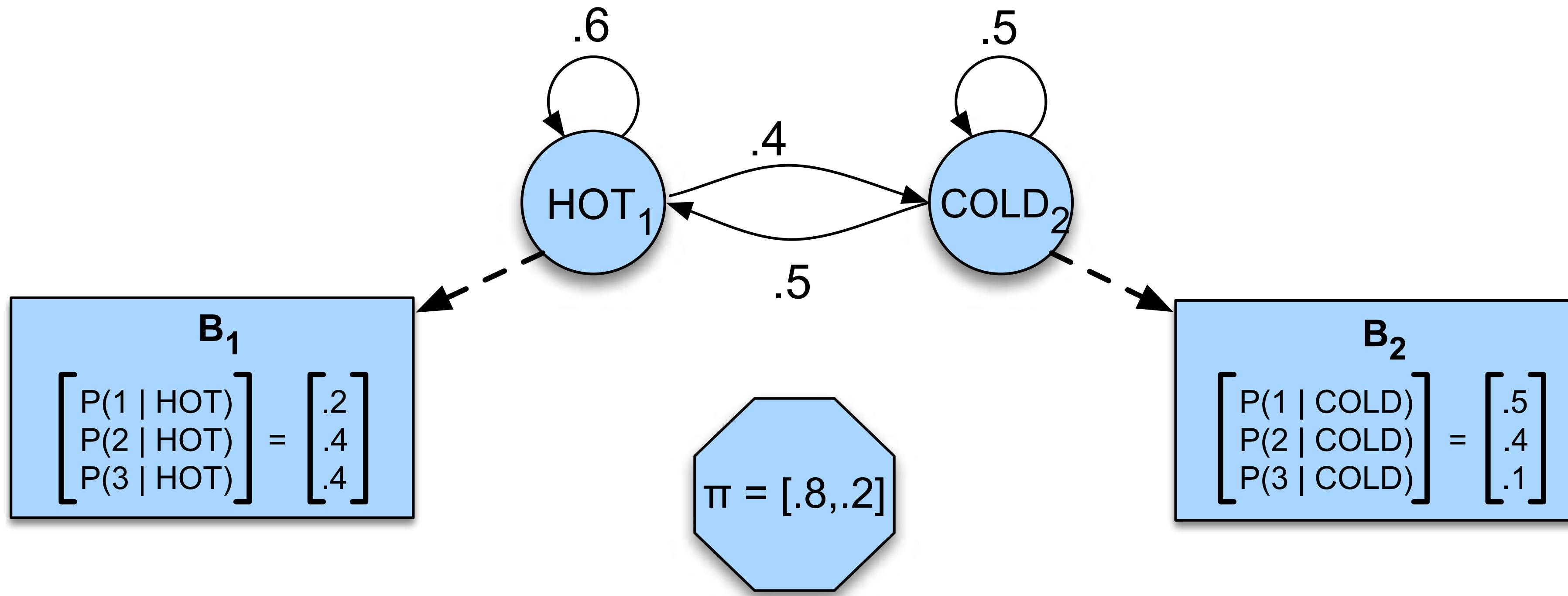
$B = b_i(o_t)$

a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state i

$\pi = \pi_1, \pi_2, \dots, \pi_N$

an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i . Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^n \pi_i = 1$

HMM Assumptions



Markov Assumption: $P(q_i | q_1 \dots q_{i-1}) = P(q_i | q_{i-1})$

Output Independence: $P(o_i | q_1 \dots q_i, \dots, q_T, o_1, \dots, o_{i-1}, o_T) = P(o_i | q_i)$

Three problems for HMMs

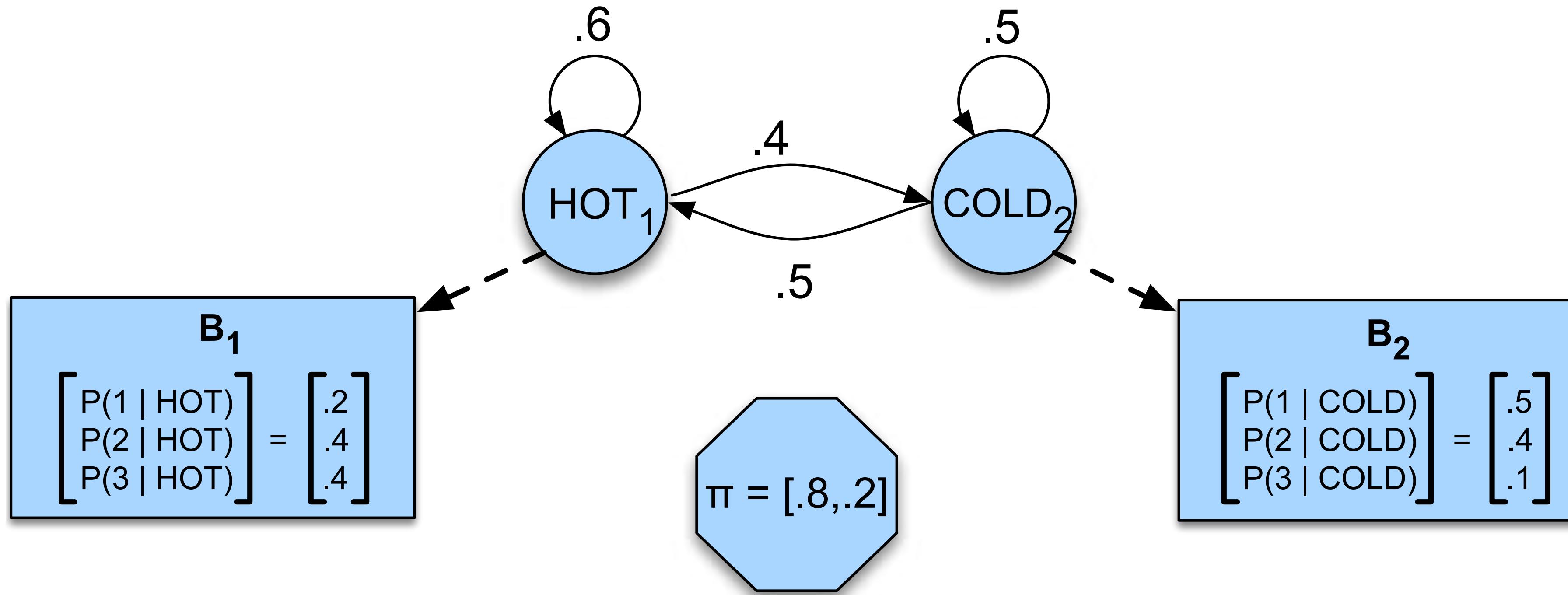
Problem 1 (Likelihood): Given an HMM $\lambda = (A, B)$ and an observation sequence O , determine the likelihood $P(O|\lambda)$.

Problem 2 (Decoding): Given an observation sequence O and an HMM $\lambda = (A, B)$, discover the best hidden state sequence Q .

Problem 3 (Learning): Given an observation sequence O and the set of states in the HMM, learn the HMM parameters A and B .

Computing Likelihood: Given an HMM $\lambda = (A, B)$ and an observation sequence O , determine the likelihood $P(O|\lambda)$.

What's the probability of O="2 1 2"?

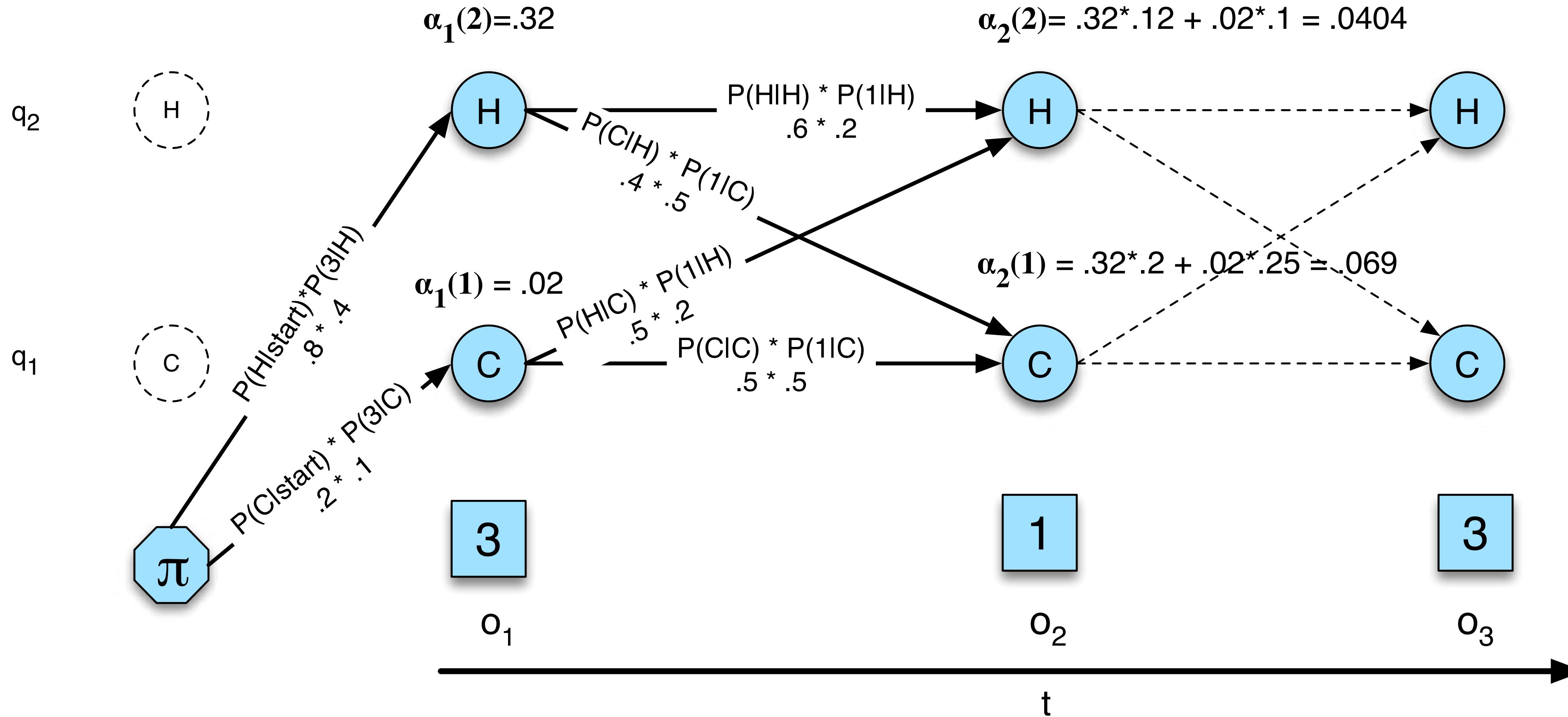


Forward Algorithm

Forward Probability

$$\alpha_t(j) = P(o_1, o_2 \dots o_t, q_t = j | \lambda)$$

$$\alpha_t(j) = \sum_{i=1}^N \alpha_{t-1}(i) a_{ij} b_j(o_t)$$



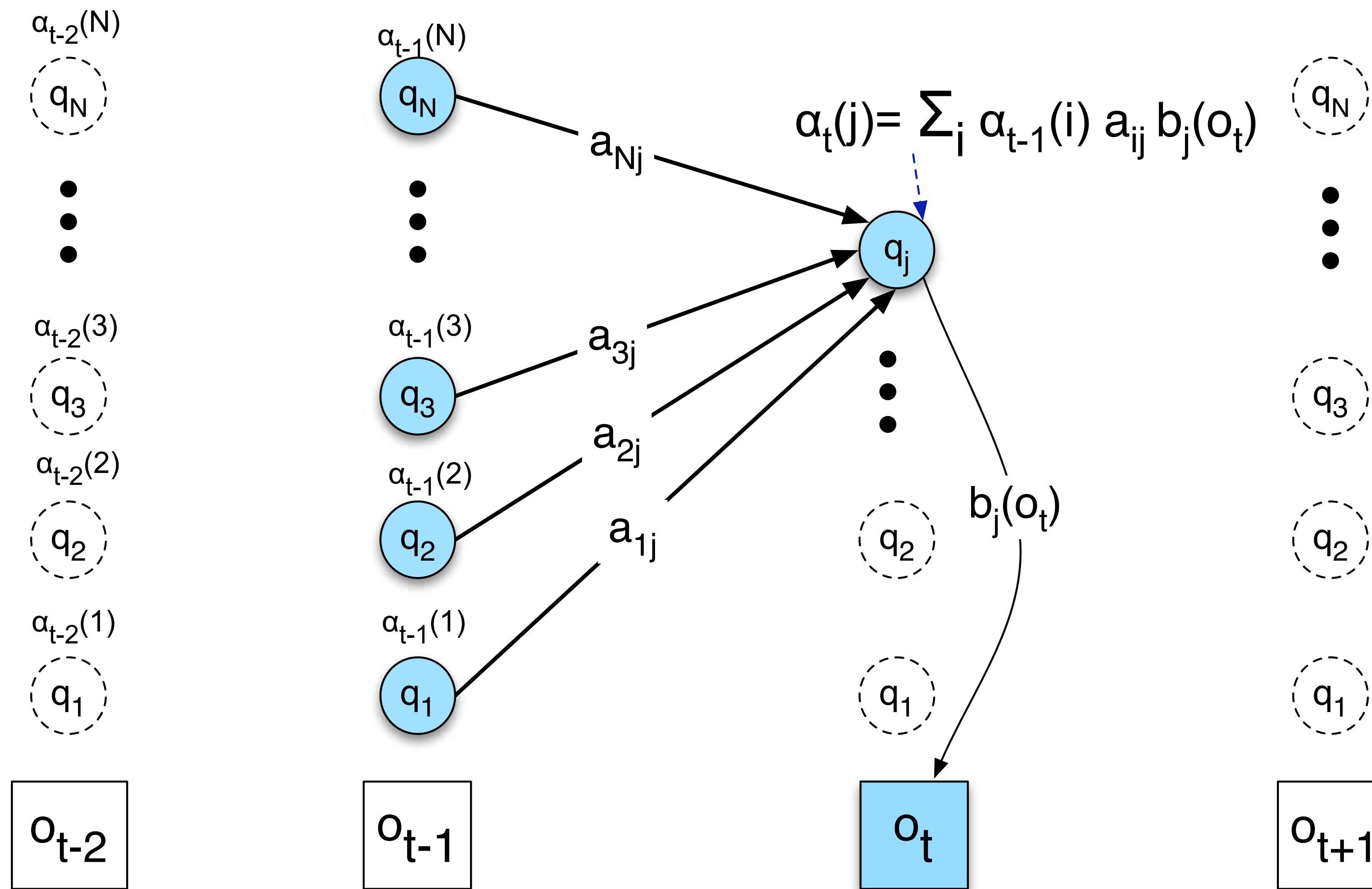
Forward Algorithm

Forward
Probability

$$\alpha_t(j) = P(o_1, o_2 \dots o_t, q_t = j | \lambda)$$

$$\alpha_t(j) = \sum_{i=1}^N \alpha_{t-1}(i) a_{ij} b_j(o_t)$$

Visualizing the forward recursion



Forward Algorithm

1. Initialization:

$$\alpha_1(j) = \pi_j b_j(o_1) \quad 1 \leq j \leq N$$

2. Recursion:

$$\alpha_t(j) = \sum_{i=1}^N \alpha_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \leq j \leq N, 1 < t \leq T$$

3. Termination:

$$P(O|\lambda) = \sum_{i=1}^N \alpha_T(i)$$

Three problems for HMMs

Problem 1 (Likelihood): Given an HMM $\lambda = (A, B)$ and an observation sequence O , determine the likelihood $P(O|\lambda)$.

Problem 2 (Decoding): Given an observation sequence O and an HMM $\lambda = (A, B)$, discover the best hidden state sequence Q .

Problem 3 (Learning): Given an observation sequence O and the set of states in the HMM, learn the HMM parameters A and B .

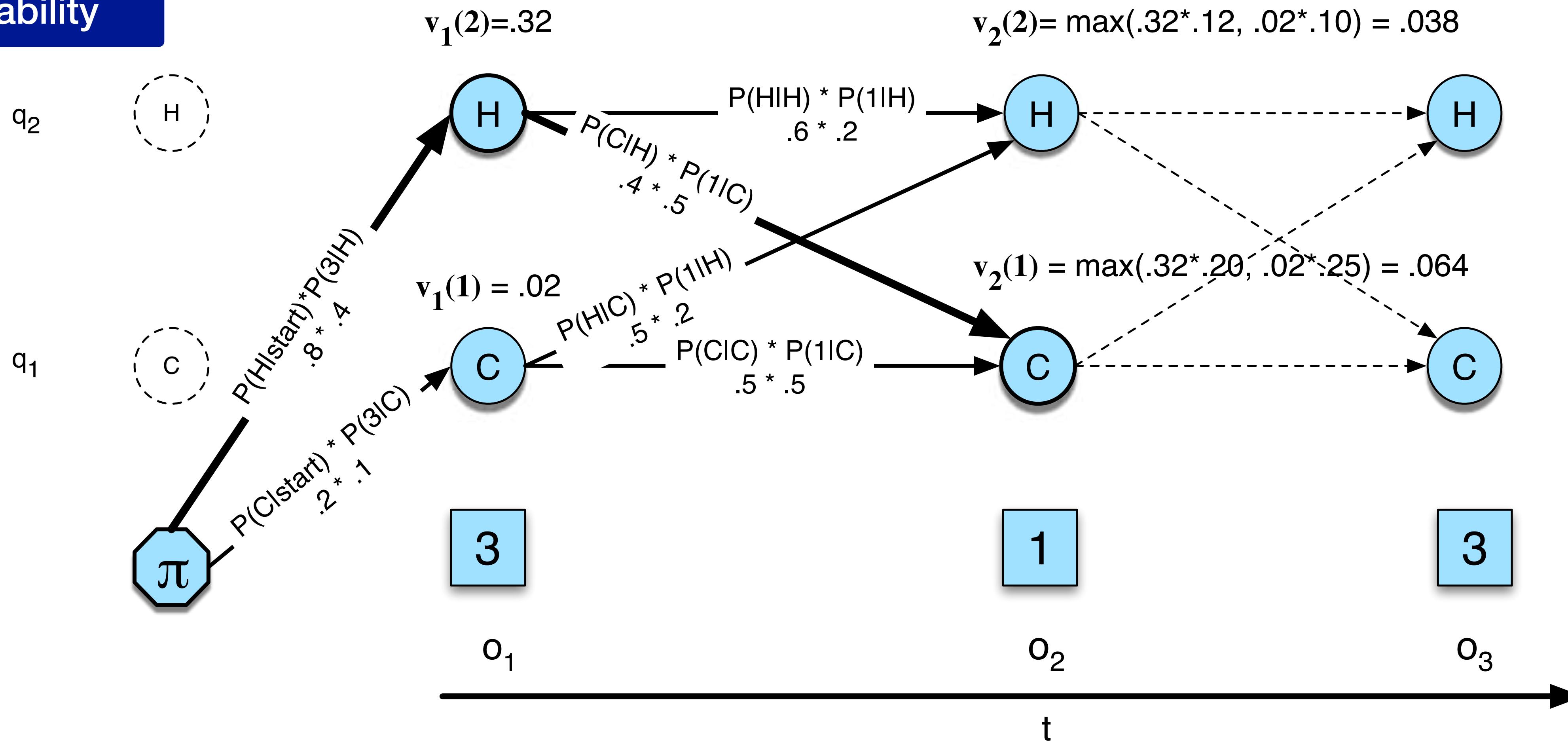
Decoding: Given as input an HMM $\lambda = (A, B)$ and a sequence of observations $O = o_1, o_2, \dots, o_T$, find the most probable sequence of states $Q = q_1 q_2 q_3 \dots q_T$.

Viterbi Trellis

$$v_t(j) = \max_{q_1, \dots, q_{t-1}} P(q_1 \dots q_{t-1}, o_1, o_2 \dots o_t, q_t = j | \lambda)$$

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t)$$

Viterbi Path Probability



Viterbi recursion

1. Initialization:

$$\begin{aligned} v_1(j) &= \pi_j b_j(o_1) & 1 \leq j \leq N \\ bt_1(j) &= 0 & 1 \leq j \leq N \end{aligned}$$

2. Recursion

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \leq j \leq N, 1 < t \leq T$$

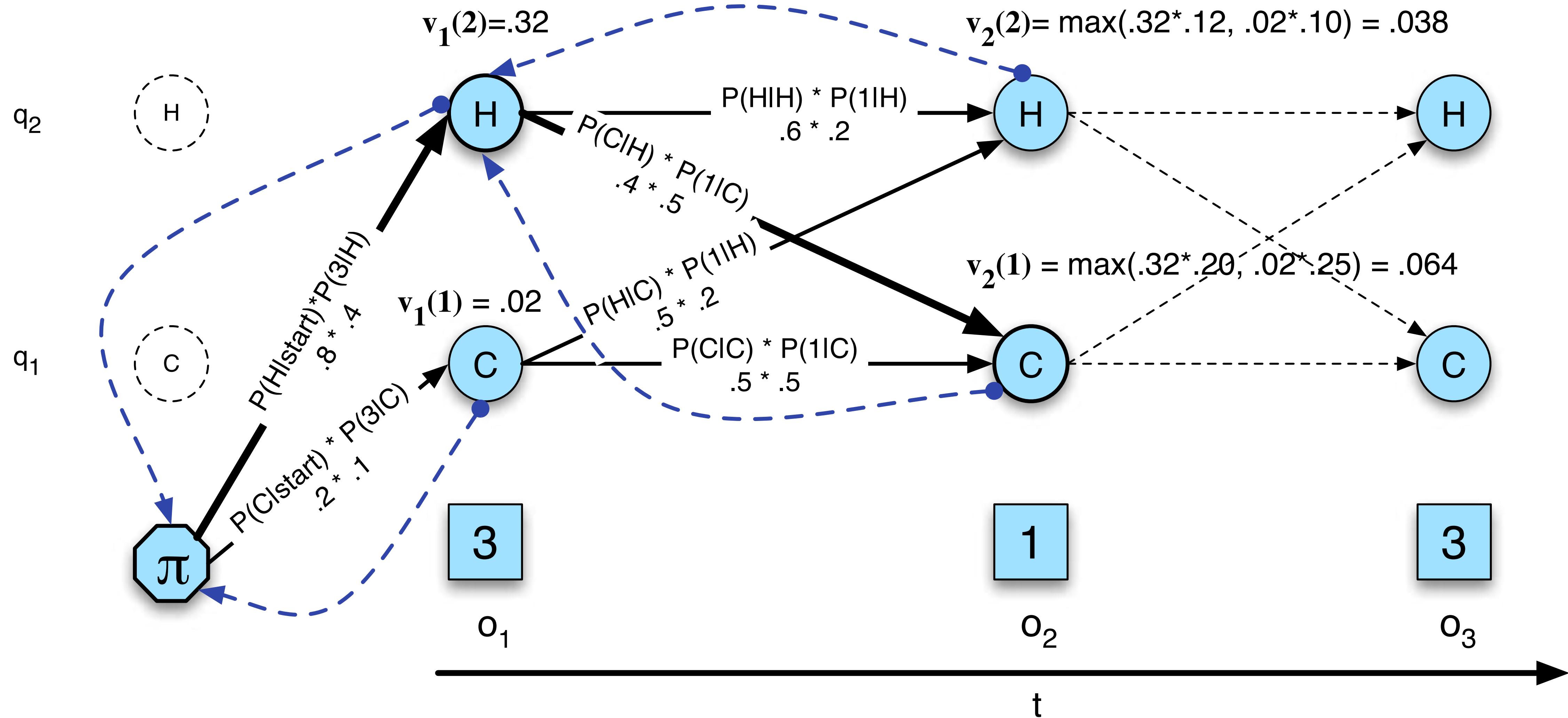
$$bt_t(j) = \operatorname{argmax}_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \leq j \leq N, 1 < t \leq T$$

3. Termination:

The best score: $P* = \max_{i=1}^N v_T(i)$

The start of backtrace: $q_T* = \operatorname{argmax}_{i=1}^N v_T(i)$

Viterbi backtrace



Next Module: Learning in HMMs

Problem 1 (Likelihood): Given an HMM $\lambda = (A, B)$ and an observation sequence O , determine the likelihood $P(O|\lambda)$.

Problem 2 (Decoding): Given an observation sequence O and an HMM $\lambda = (A, B)$, discover the best hidden state sequence Q .

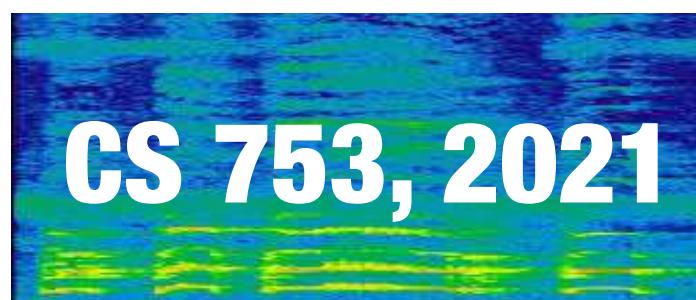
Problem 3 (Learning): Given an observation sequence O and the set of states in the HMM, learn the HMM parameters A and B .

Learning: Given an observation sequence O and the set of possible states in the HMM, learn the HMM parameters A and B .

HMM training: **Forward-backward or Baum-Welch algorithm**

HMMs for Acoustic Modeling

Lecture 1b



Instructor: Preethi Jyothi, IITB

Recap: HMMs

- ✓ What are (first-order) HMMs?
- ✓ What are the simplifying assumptions governing HMMs?
- ✓ What is the forward algorithm? What is it used to compute?
Computing Likelihood: Given an HMM $\lambda = (A, B)$ and an observation sequence O , determine the likelihood $P(O|\lambda)$.
- ✓ What is the Viterbi algorithm? What is it used to compute?
Decoding: Given as input an HMM $\lambda = (A, B)$ and a sequence of observations $O = o_1, o_2, \dots, o_T$, find the most probable sequence of states $Q = q_1 q_2 q_3 \dots q_T$.

Learning in HMMs

Problem 1 (Likelihood): Given an HMM $\lambda = (A, B)$ and an observation sequence O , determine the likelihood $P(O|\lambda)$.

Problem 2 (Decoding): Given an observation sequence O and an HMM $\lambda = (A, B)$, discover the best hidden state sequence Q .

Problem 3 (Learning): Given an observation sequence O and the set of states in the HMM, learn the HMM parameters A and B .

Learning: Given an observation sequence O and the set of possible states in the HMM, learn the HMM parameters A and B .

Forward-backward or Baum-Welch (EM) algorithm

EM for general HMMs: Baum-Welch algorithm (1972)

(predates the general formulation of EM (1977))

What is the EM Algorithm?

Expectation Maximization (EM) Algorithm

EM is an iterative algorithm used to compute Maximum Likelihood (ML) (or Maximum A posteriori MAP) estimates in the presence of missing or hidden data.

Two step iterative algorithm:

E step: Estimate hidden variables given the observations and current estimates of the model parameters.

M step: Estimating model parameters by maximising the likelihood function using estimates of the hidden data from the E step.

EM Algorithm: Fitting Parameters to Data

Observed data: i.i.d samples $x_i, i=1, \dots, N$

Hidden data: Denoted by z

Goal: Find $\arg \max_{\theta} \mathcal{L}(\theta)$ where $\mathcal{L}(\theta) = \sum_{i=1}^N \log \Pr(x_i; \theta)$

Initial parameters: θ^0 (x is observed and z is hidden)

Iteratively compute θ^ℓ as follows that optimises an auxiliary function $Q(\theta, \theta^{\ell-1})$:

$$Q(\theta, \theta^{\ell-1}) = \sum_{i=1}^N \sum_z \Pr(z|x_i; \theta^{\ell-1}) \log \Pr(x_i, z; \theta)$$

$$\theta^\ell = \arg \max_{\theta} Q(\theta, \theta^{\ell-1})$$

Estimate θ^ℓ cannot get worse over iterations because for all θ :

$$\mathcal{L}(\theta) - \mathcal{L}(\theta^{\ell-1}) \geq Q(\theta, \theta^{\ell-1}) - Q(\theta^{\ell-1}, \theta^{\ell-1})$$

EM is guaranteed to converge to a local optimum or saddle points [Wu83]

[Optional] EM: How do we arrive at the auxiliary function?

$$Q(\theta, \theta^{\ell-1}) = \sum_{i=1}^N \sum_z \Pr(z|x_i; \theta^{\ell-1}) \log \Pr(x_i, z; \theta)$$

**How do we use EM for
HMM parameter estimation?**

Forward and Backward Probabilities

Baum-Welch algorithm iteratively estimates transition & observation probabilities and uses these values to derive even better estimates.

Require two probabilities to compute estimates for the transition and observation probabilities:

1. **Forward probability:** Recall $\alpha_t(j) = P(o_1, o_2 \dots o_t, q_t = j | \lambda)$
2. **Backward probability:** $\beta_t(i) = P(o_{t+1}, o_{t+2} \dots o_T | q_t = i, \lambda)$

Backward probability

1. Initialization:

$$\beta_T(i) = 1, \quad 1 \leq i \leq N$$

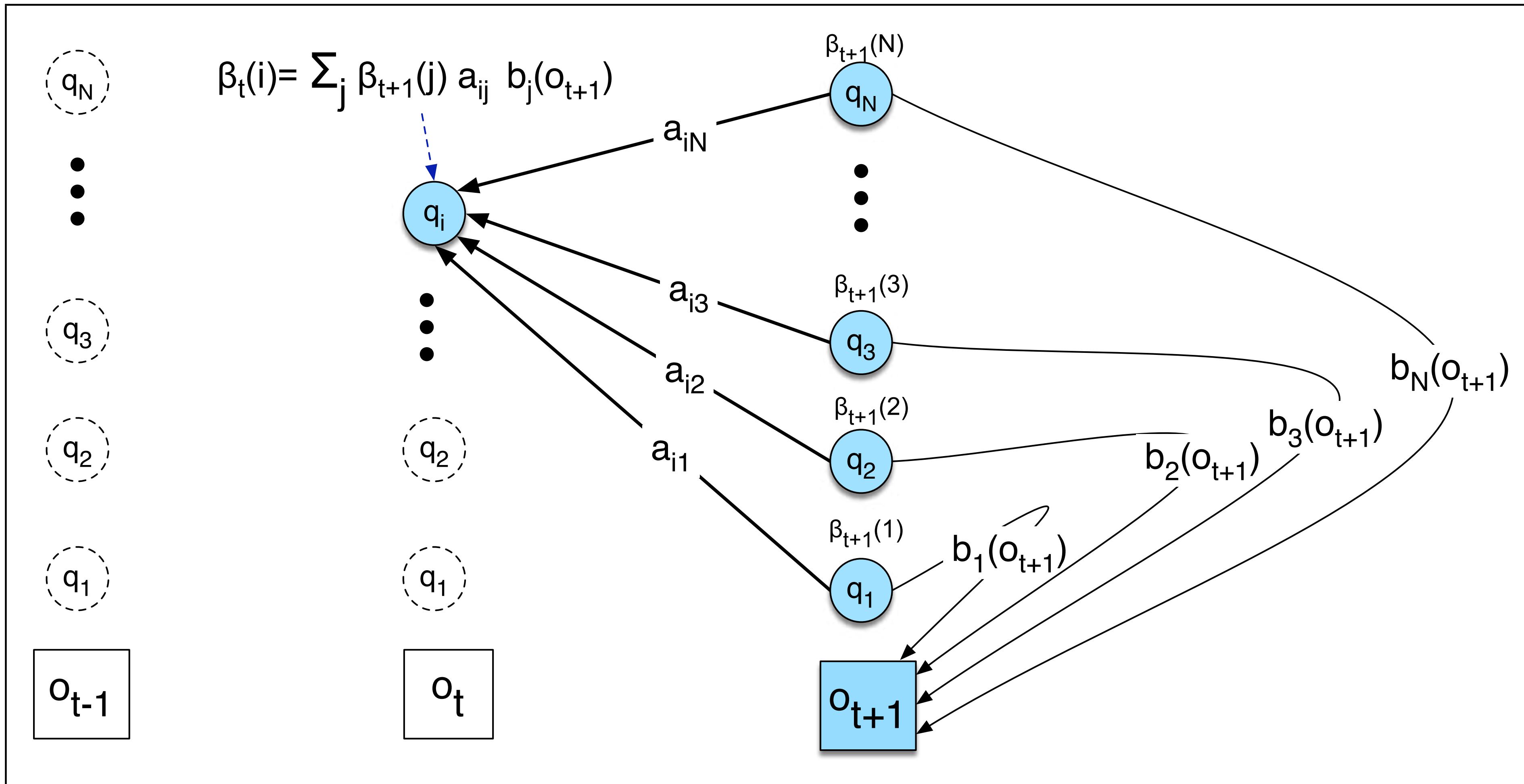
2. Recursion

$$\beta_t(i) = \sum_{j=1}^N a_{ij} b_j(o_{t+1}) \beta_{t+1}(j), \quad 1 \leq i \leq N, 1 \leq t < T$$

3. Termination:

$$P(O|\lambda) = \sum_{j=1}^N \pi_j b_j(o_1) \beta_1(j)$$

Visualising backward probability computation



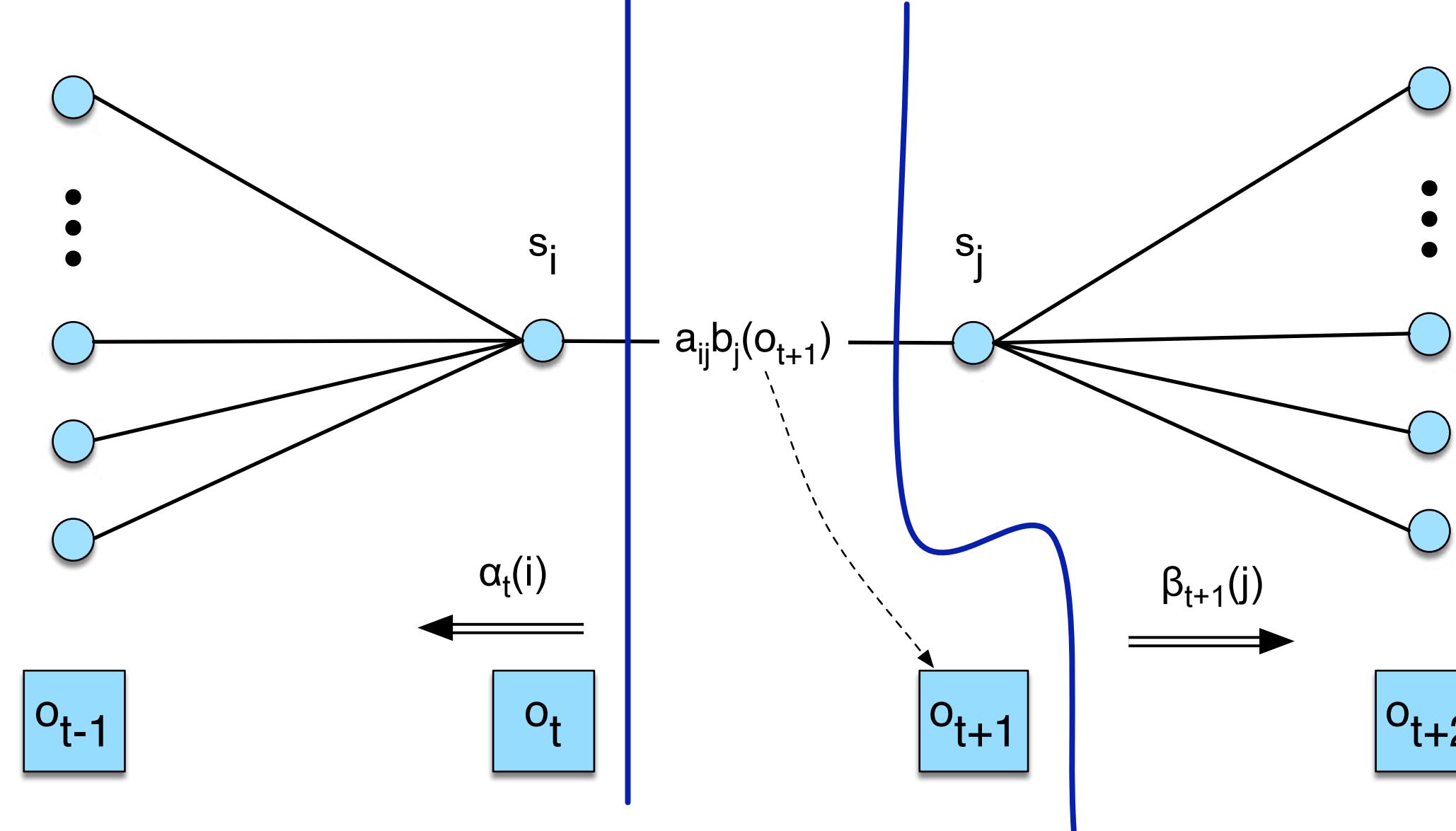
1. Baum-Welch: Estimating a_{ij}

We need to define $\xi_t(i, j)$ to estimate a_{ij}

where $\xi_t(i, j) = P(q_t = i, q_{t+1} = j | O, \lambda)$

which works out to be $\xi_t(i, j) = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{j=1}^N \alpha_t(j) \beta_t(j)}$

Then, $\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \sum_{k=1}^N \xi_t(i, k)}$



2. Baum-Welch: Estimating $b_j(v_k)$

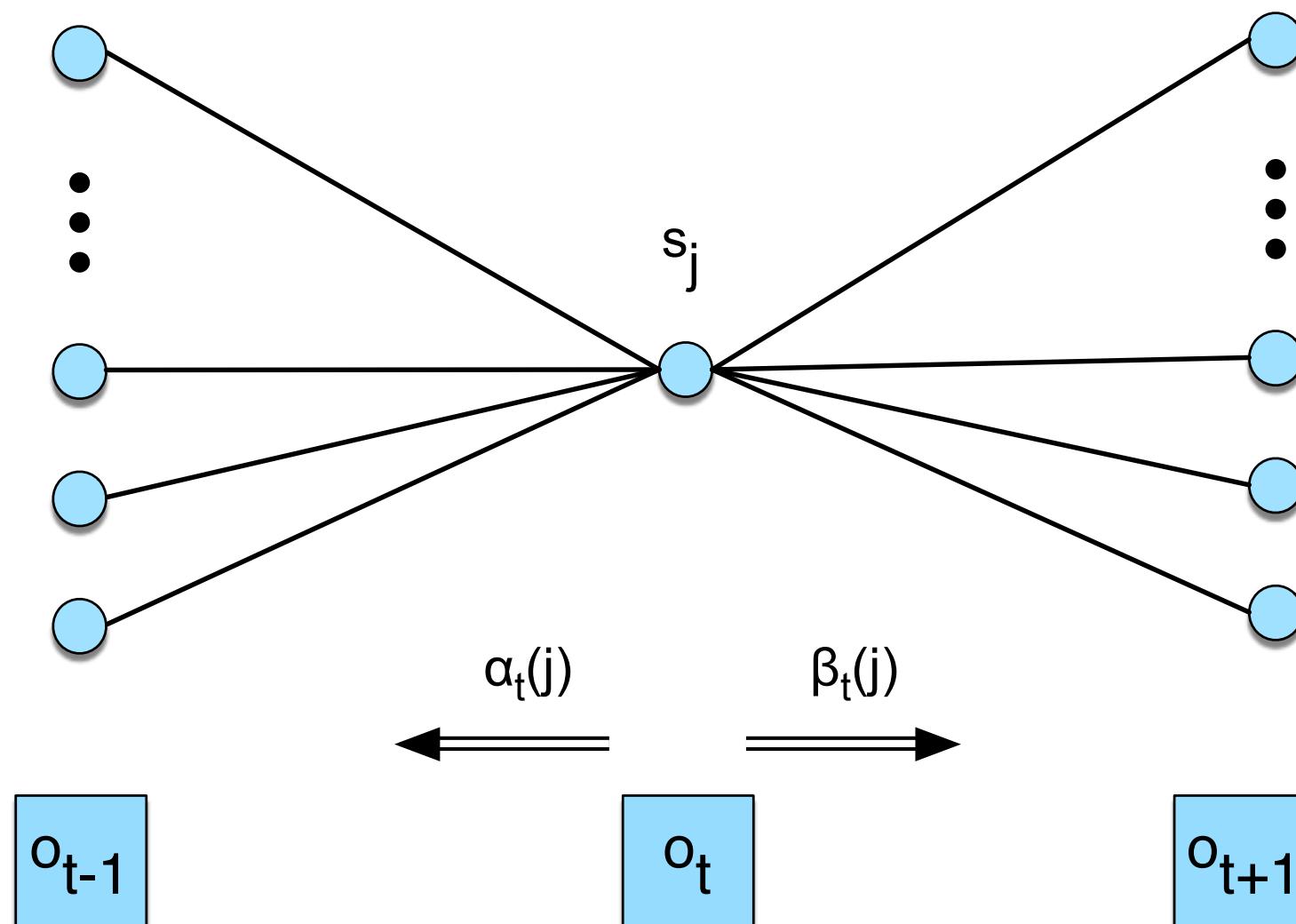
We need to define $\gamma_t(j)$ to estimate $b_j(v_k)$

where $\gamma_t(j) = P(q_t = j | O, \lambda)$

which works out to be $\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)}$

State occupancy probability

Then, $\hat{b}_j(v_k) = \frac{\sum_{t=1}^T \gamma_t(j)}{\sum_{t=1}^T \gamma_t(j)}$ for discrete outputs



Bringing it all together: Baum-Welch

Estimating HMM parameters iteratively using the EM algorithm.

For each iteration, do:

E step: For all time-state pairs, compute the state occupation probabilities $\gamma_t(j)$ and $\xi_t(i, j)$

M step: Reestimate HMM parameters, i.e. transition probabilities, observation probabilities, based on the estimates derived in the E step

Baum-Welch algorithm (pseudocode)

function FORWARD-BACKWARD(*observations* of len T , *output vocabulary* V , *hidden state set* Q) **returns** $HMM=(A,B)$

initialize A and B

iterate until convergence

E-step

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{\alpha_T(q_F)} \quad \forall t \text{ and } j$$

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(q_F)} \quad \forall t, i, \text{ and } j$$

M-step

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{k=1}^N \xi_t(i,k)}$$

$$\sum_{t=1}^T \gamma_t(j)$$

$$\hat{b}_j(v_k) = \frac{\sum_{t=1}^T s.t. O_t=v_k \gamma_t(j)}{\sum_{t=1}^T \gamma_t(j)}$$

return A, B

Baum-Welch Algorithm as EM

Observed data: N sequences, x_i , $i=1..N$ where $x_i \in V$

Parameters θ : transition matrix A , observation probabilities B

[EM Iteration, E-step]

Compute quantities involved in $Q(\theta, \theta^{\ell-1})$

$$\gamma_{i,t}(j) = \Pr(z_t = j \mid x_i; \theta^{\ell-1})$$

$$\xi_{i,t}(j,k) = \Pr(z_t = j, z_{t+1} = k \mid x_i; \theta^{\ell-1})$$

Baum-Welch Algorithm as EM

Observed data: N sequences, x_i , $i=1..N$ where $x_i \in V$

Parameters θ : transition matrix A , observation probabilities B

[EM Iteration, M-step]

Find θ which maximises $Q(\theta, \theta^{\ell-1})$

$$A_{j,k} = \frac{\sum_{i=1}^N \sum_{t=1}^{T_i-1} \xi_{i,t}(j, k)}{\sum_{i=1}^N \sum_{t=1}^{T_i-1} \sum_{k'} \xi_{i,t}(j, k')}$$

$$B_{j,v} = \frac{\sum_{i=1}^N \sum_{t:x_{it}=v} \gamma_{i,t}(j)}{\sum_{i=1}^N \sum_{t=1}^{T_i} \gamma_{i,t}(j)}$$

Discrete to continuous outputs

We derived Baum-Welch updates for discrete outputs.

However, HMMs in acoustic models emit real-valued vectors as observations.

Use probability density functions to define observation probabilities

If x were 1D values, HMM observation probabilities: $b_j(x) = \mathcal{N}(x | \mu_j, \sigma_j^2)$
where μ_j is the mean associated with state j and σ_j^2 is its variance

If $\mathbf{x} \in \mathbb{R}^d$, then we use multivariate Gaussians, $b_j(\mathbf{x}) = \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$
where $\boldsymbol{\Sigma}_j$ is the covariance matrix associated with state j

BW for Gaussian Observation Model

Observed data: N sequences, $x_i = (x_{i1}, \dots, x_{iT_i})$, $i=1..N$ where $x_{it} \in \mathbb{R}^d$

Parameters θ : transition matrix A , observation prob. $B = \{(\mu_j, \Sigma_j)\}$ for all j

[EM Iteration, M-step]

Find θ which maximises $Q(\theta, \theta^{\ell-1})$

A same as with discrete outputs

$$\mu_j = \frac{\sum_{i=1}^N \sum_{t=1}^{T_i} \gamma_{i,t}(j) x_{it}}{\sum_{i=1}^N \sum_{t=1}^{T_i} \gamma_{i,t}(j)}$$

$$\Sigma_j = \frac{\sum_{i=1}^N \sum_{t=1}^{T_i} \gamma_{i,t}(j) (x_{it} - \mu_j)(x_{it} - \mu_j)^T}{\sum_{i=1}^N \sum_{t=1}^{T_i} \gamma_{i,t}(j)}$$

Gaussian Mixture Model

- Assuming that observations associated with a state follow a Gaussian distribution is too simplistic.
- More generally, we use a “mixture of Gaussians” to allow for acoustic vectors associated with a state to be non-Gaussian.
- Instead of $b_j(\mathbf{x}) = \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_j, \Sigma_j)$ in the single Gaussian case, $b_j(\mathbf{x})$ can be an M -component mixture model:

$$b_j(\mathbf{x}) = \sum_{m=1}^M c_{jm} \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_{jm}, \Sigma_{jm})$$

where c_{jm} is the mixing probability for Gaussian component m of state j

$$\sum_{m=1}^M c_{jm} = 1, \quad c_{jm} \geq 0$$

BW for Gaussian Mixture Model

Observed data: N sequences, $x_i = (x_{i1}, \dots, x_{iT_i})$, $i=1..N$ where $x_{it} \in \mathbb{R}^d$

Parameters θ : transition matrix A , observation prob. $B = \{(\mu_{jm}, \Sigma_{jm}, c_{jm})\}$ for all j, m

[EM Iteration, M-step]

Find θ which maximises $Q(\theta, \theta^{\ell-1})$

$$\mu_{jm} = \frac{\sum_{i=1}^N \sum_{t=1}^{T_i} \gamma_{i,t}(j, m) x_{it}}{\sum_{i=1}^N \sum_{t=1}^{T_i} \gamma_{i,t}(j, m)}$$

$$\Sigma_{jm} = \frac{\sum_{i=1}^N \sum_{t=1}^{T_i} \gamma_{i,t}(j, m) (x_{it} - \mu_{jm})(x_{it} - \mu_{jm})^T}{\sum_{i=1}^N \sum_{t=1}^{T_i} \gamma_{i,t}(j, m)}$$

$$c_{jm} = \frac{\sum_{i=1}^N \sum_{t=1}^{T_i} \gamma_{i,t}(j, m)}{\sum_{i=1}^N \sum_{t=1}^{T_i} \sum_{m'=1}^M \gamma_{i,t}(j, m')}$$

Prob. of component m
of state j at time t

Baum Welch: In summary

[Every EM Iteration]

Compute $\theta = \{ A_{jk}, (\mu_{jm}, \Sigma_{jm}, c_{jm}) \}$ for all j, k, m

$$A_{j,k} = \frac{\sum_{i=1}^N \sum_{t=2}^{T_i} \xi_{i,t}(j, k)}{\sum_{i=1}^N \sum_{t=2}^{T_i} \sum_{k'} \xi_{i,t}(j, k')}$$

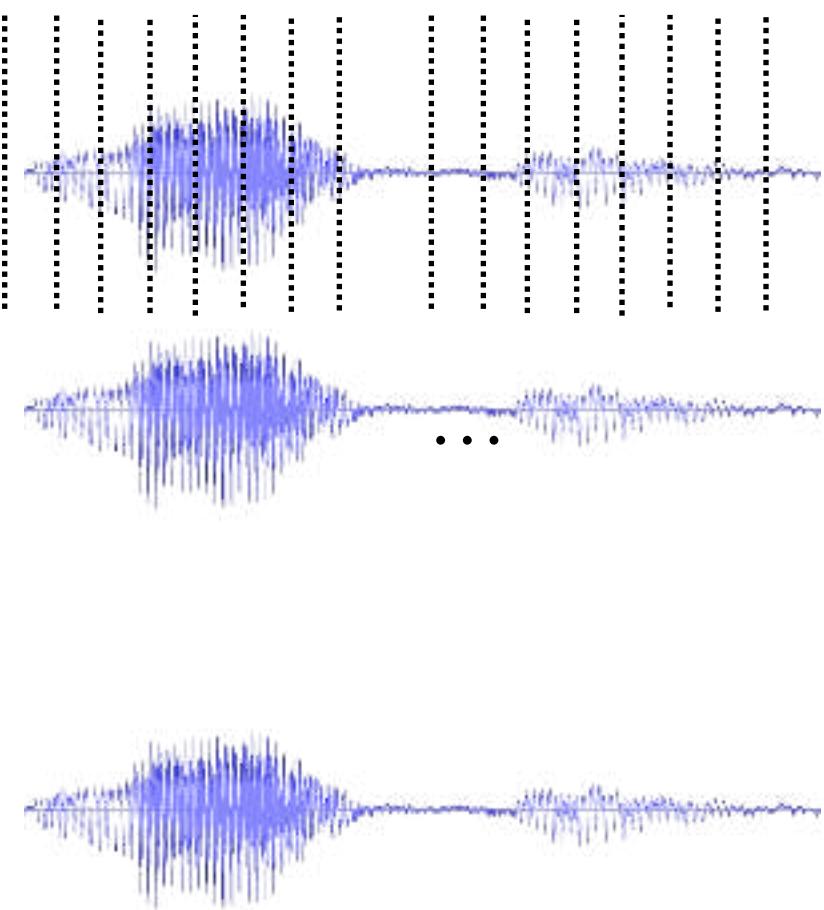
$$\mu_{jm} = \frac{\sum_{i=1}^N \sum_{t=1}^{T_i} \gamma_{i,t}(j, m) x_{it}}{\sum_{i=1}^N \sum_{t=1}^{T_i} \gamma_{i,t}(j, m)}$$

$$\Sigma_{jm} = \frac{\sum_{i=1}^N \sum_{t=1}^{T_i} \gamma_{i,t}(j, m) (x_{it} - \mu_{jm})(x_{it} - \mu_{jm})^T}{\sum_{i=1}^N \sum_{t=1}^{T_i} \gamma_{i,t}(j, m)}$$

$$c_{jm} = \frac{\sum_{i=1}^N \sum_{t=1}^{T_i} \gamma_{i,t}(j, m)}{\sum_{i=1}^N \sum_{t=1}^{T_i} \sum_{m'=1}^M \gamma_{i,t}(j, m')}$$

Overall Summary

Training



$$\begin{aligned} \rightarrow O_1^1, \dots, O_{t1}^1 &\quad \text{and} \quad \mathbf{w}^1 = w_1^1, \dots, w_{\ell 1}^1 \\ \dots \rightarrow O_1^2, \dots, O_{t2}^2 &\quad \text{and} \quad \mathbf{w}^2 = w_1^2, \dots, w_{\ell 2}^2 \\ &\vdots \\ \rightarrow O_1^N, \dots, O_{tN}^N &\quad \text{and} \quad \mathbf{w}^N = w_1^N, \dots, w_{\ell N}^N \end{aligned}$$

Estimate $\theta = \{A_{jk}, (\mu_{jm}, \Sigma_{jm}, C_{jm})\}$ over all phone states. Use Baum-Welch.

HMM of i th training utterance determined by using a word-to-phone mapping applied to $w_1^i, \dots, w_{\ell i}^i$

Train LM using $\mathbf{w}^1, \dots, \mathbf{w}^N$

Test



$$\mathbf{O} = O_1, \dots, O_T$$

$$W^* = \arg \max_W P(W | O)$$

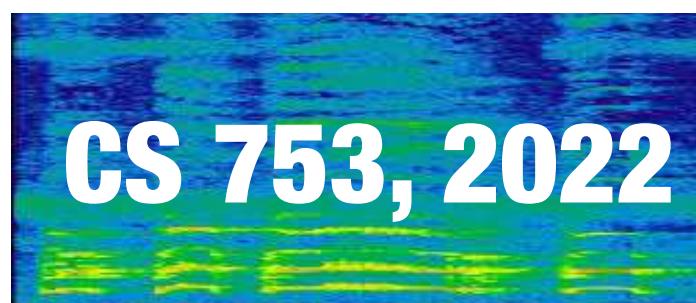
Compute using Viterbi algorithm

Search all possible state sequences arising from all word sequences most likely to have generated \mathbf{O}

Computationally intractable for continuous speech! Efficient algorithms in later classes.

HMMs for Acoustic Modeling

Live Session



Instructor: Preethi Jyothi, IITB

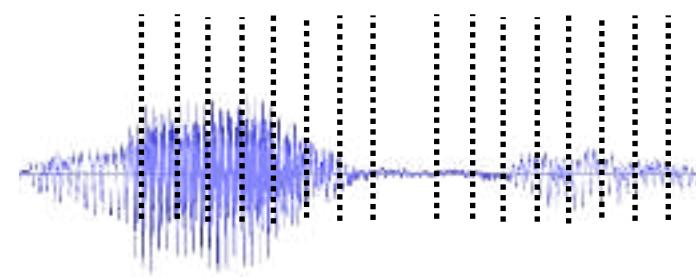
Around the world

wer_are_we				
WER				
LibriSpeech				
(Possibly trained on more data than LibriSpeech.)				
WER test- clean	WER test- other	Paper	Published	Notes
5.83%	12.69%	Humans Deep Speech 2: End-to-End Speech Recognition in English and Mandarin	December 2015	Humans
1.8%	2.9%	HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units	June 2021	CNN-Transformer + Transformer LM (Self-Supervised, Libri-light-60K Unlabeled Data)
1.9%	3.9%	Conformer: Convolution-augmented Transformer for Speech Recognition	May 2020	Convolution-augmented- Transformer(Conformer) + 3-layer LSTM LM (data augmentation, Cross-Augment)

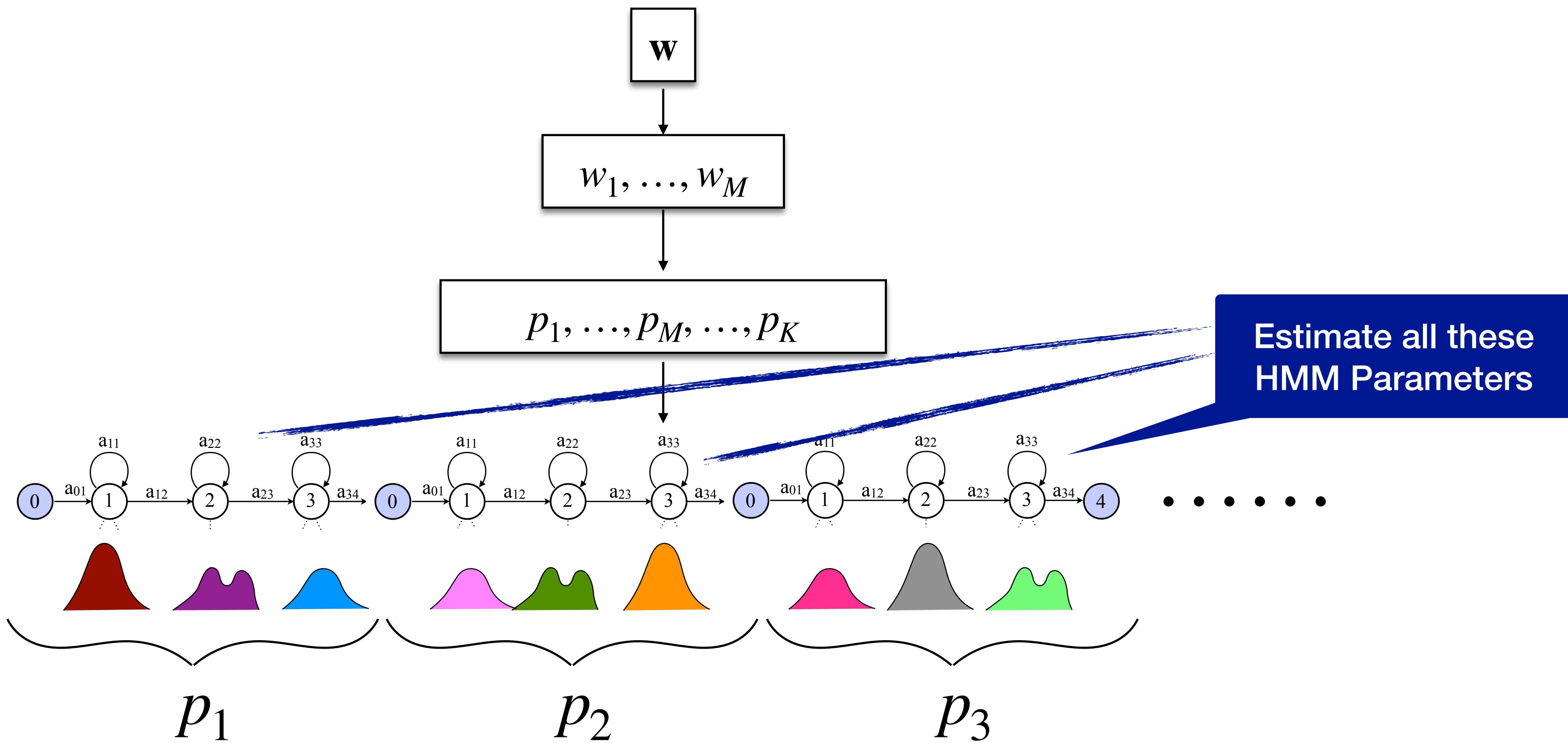
Role-Playing Seminars

ROLE	Description	Evaluation/Due	Score
Talk Presentation	Give a 15-20 min talk (as the author) to clearly explain the main ideas in the paper	According to talk schedule	8
Scientific Reporter	Write a technical article about the paper for a non-specialist but general CS audience	Due towards the end of the semester	5
Peer Reviewer	Provide a full review of the paper	Reviews will be discussed in the same session as the talk	4
Poster Presentation	Prepare a poster visualising the main highlights of the paper	GatherTown poster session at the end of the semester	5
Hacker	Implement a small part/simplified version of the paper	Code/demo submission after the final exams	8

Summary: Training



$$\mathbf{O} = O_1, \dots, O_T \quad \text{and} \quad \mathbf{w} = w_1, \dots, w_M$$



Summary: Test



$$\mathbf{O} = O_1, \dots, O_T$$

$$Q^* = \arg \max_Q P(Q | O)$$

Compute using Viterbi algorithm!

Monophone HMMs — Good enough?

- A phone is affected by its phonetic context.
 - E.g. Coarticulation: Production of a speech sound is affected by adjacent speech sounds. “soon” vs. “seat”. “ten” vs. “tenth”.
- Use phones in context instead of monophones. E.g. diphones or triphones.
- Triphones are commonly used in ASR systems. Phone p with left context l and right context r is written as “l-p+r”
 - “hello world” → sil-h+eh h-eh+l eh-l+ow l-ow+w ow-w-er w-er+l er-l+d l-d+sil

Triphone HMM Models

- Number of observed triphones that appear in training data $\approx 10,000$ s
- If each triphone HMM has 3 states and each state generates m -component GMMs ($m \approx 64$), for d -dimensional acoustic feature vectors ($d \approx 40$) with Σ having d parameters (assuming diagonal covariance matrices)
 - Hundreds of millions of parameters!
- Insufficient data to learn all triphone models reliably. What do we do? Share parameters across triphone models.

More in this week's lecture on tied-state HMMs!

Recall ASR Decoding



$$\mathbf{O} = O_1, \dots, O_T$$

$$Q^* = \arg \max_Q P(Q | O)$$

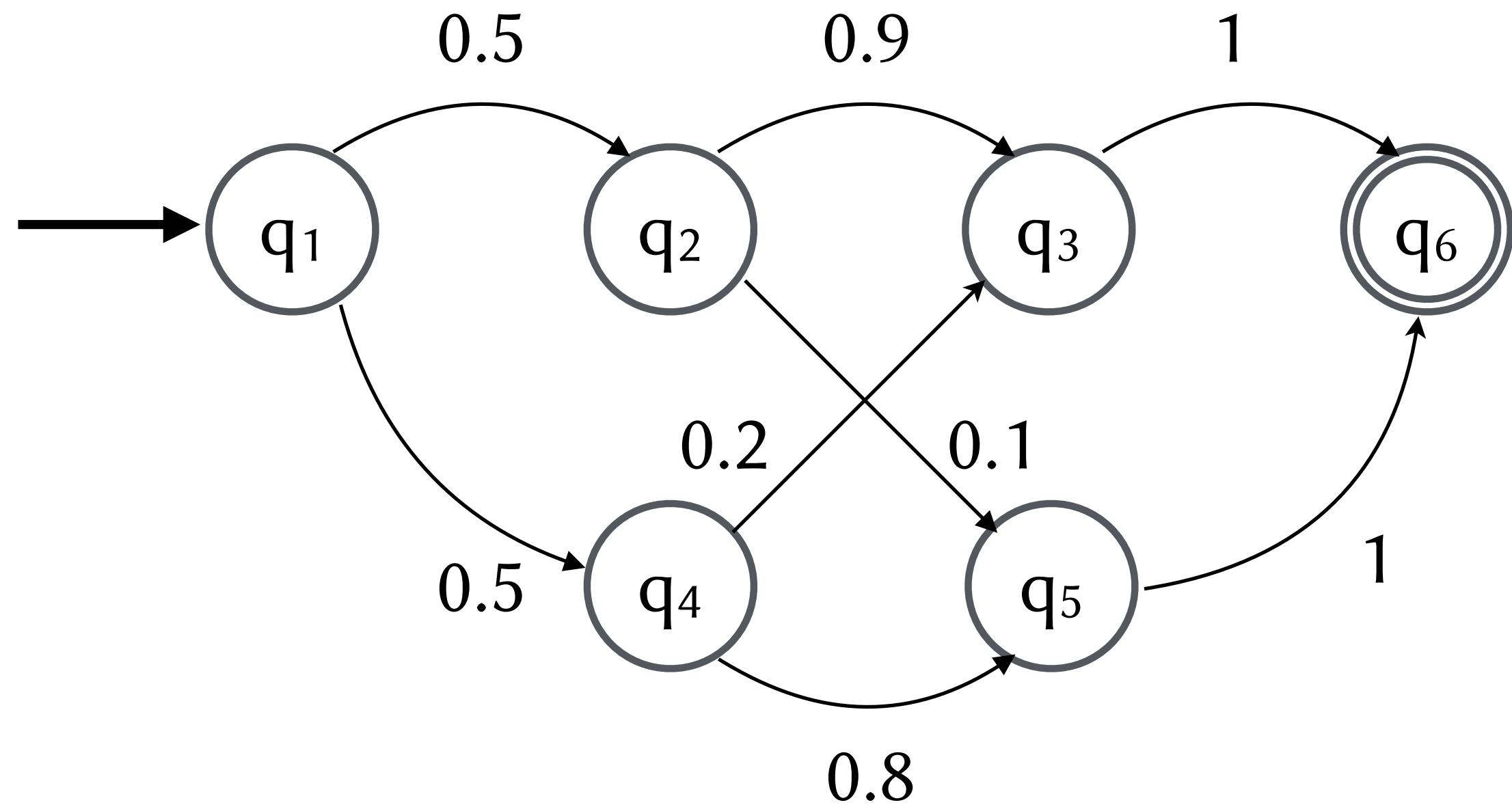
How do we go from the best state sequence to the best word sequence?

Weighted
Finite State
Transducers

Recap: HMM Forward/Viterbi Algorithms

HMMs (I)

This HMM generates hidden state sequences ($S_i \in \{q_1, \dots, q_6\}$) and observation sequences ($O_i \in \{a, b, c\}$) of length 4.



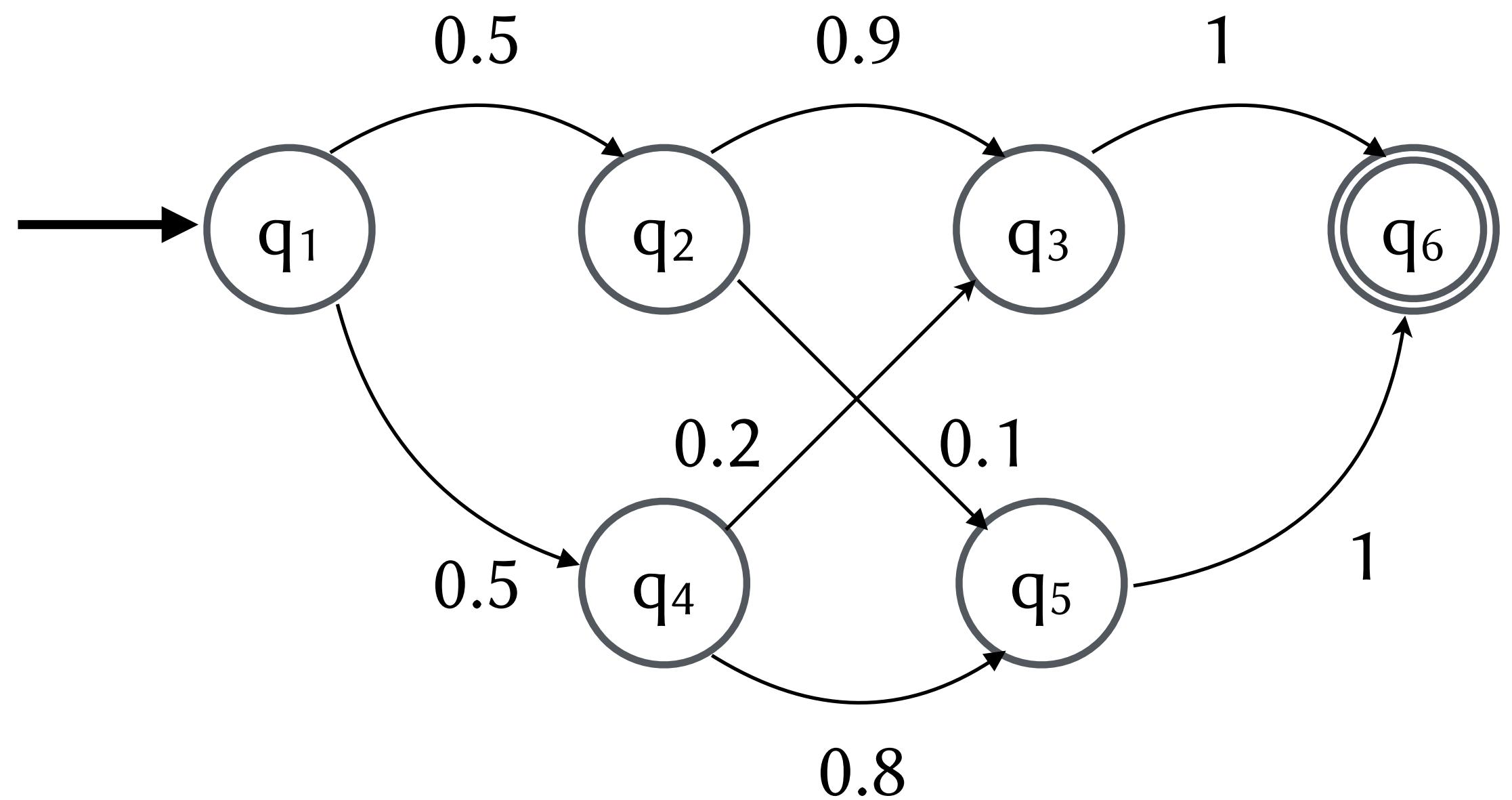
	a	b	c
q1	0.5	0.3	0.2
q2	0.3	0.4	0.3
q3	0.2	0.1	0.7
q4	0.4	0.5	0.1
q5	0.3	0.3	0.4
q6	0.9	0	0.1

True or False?

$$\Pr(O = bbca, S_1 = q_1, S_4 = q_6) = \Pr(O = bbca | S_1 = q_1, S_4 = q_6)$$

HMMs (II)

This HMM generates hidden state sequences ($S_i \in \{q_1, \dots, q_6\}$) and observation sequences ($O_i \in \{a, b, c\}$) of length 4.



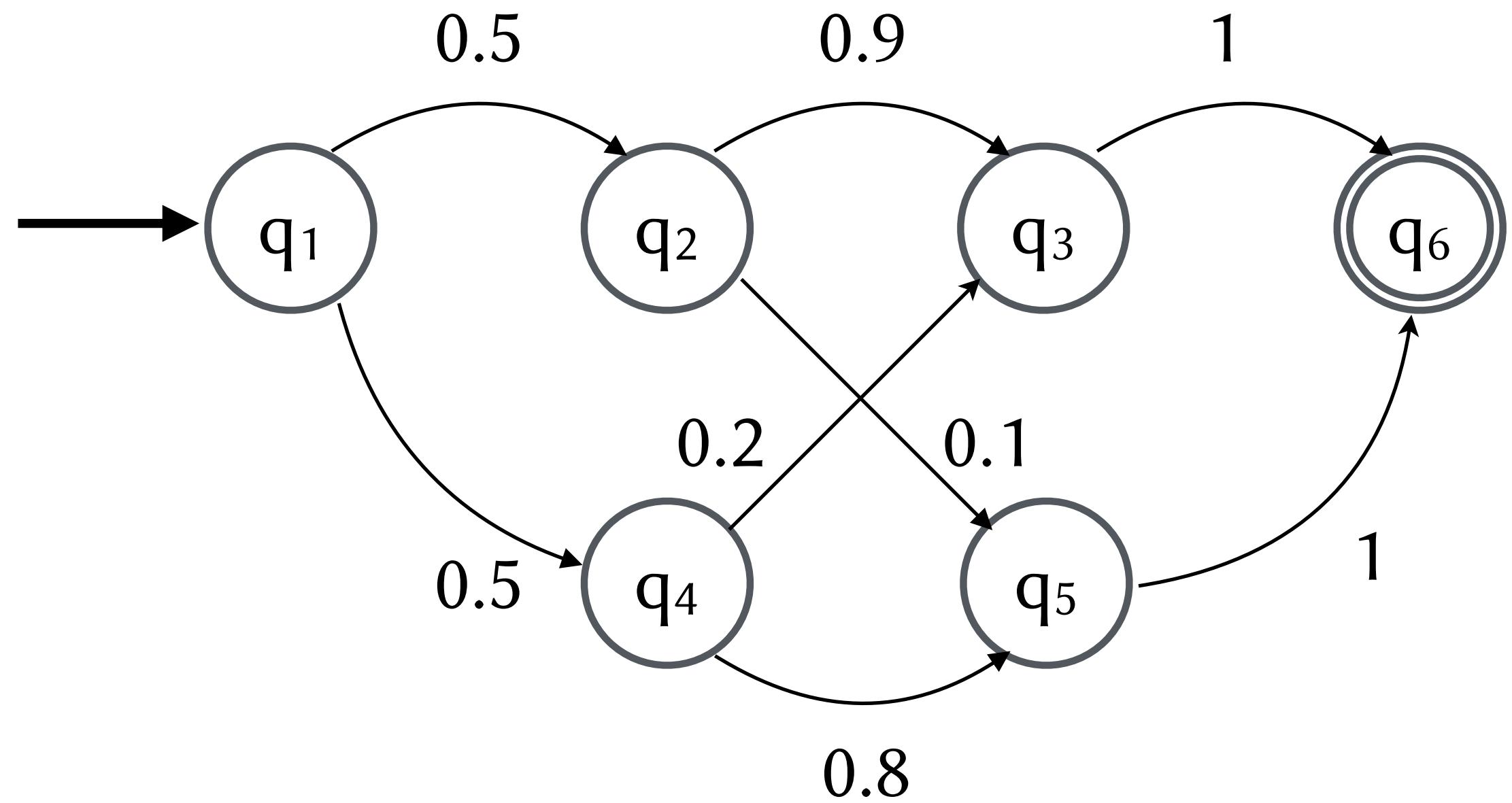
	a	b	c
q ₁	0.5	0.3	0.2
q ₂	0.3	0.4	0.3
q ₃	0.2	0.1	0.7
q ₄	0.4	0.5	0.1
q ₅	0.3	0.3	0.4
q ₆	0.9	0	0.1

True or False?

$$\Pr(O = acac, S_2 = q_2, S_3 = q_5) > \Pr(O = acac, S_2 = q_4, S_3 = q_3)$$

HMMs (III)

This HMM generates hidden state sequences ($S_i \in \{q_1, \dots, q_6\}$) and observation sequences ($O_i \in \{a, b, c\}$) of length 4.



	a	b	c
q ₁	0.5	0.3	0.2
q ₂	0.3	0.4	0.3
q ₃	0.2	0.1	0.7
q ₄	0.4	0.5	0.1
q ₅	0.3	0.3	0.4
q ₆	0.9	0	0.1

True or False?

$$\Pr(O = cbcb | S_2 = q_2, S_3 = q_5) = \Pr(O = baac, S_2 = q_4, S_3 = q_5)$$

Recap: HMM Baum-Welch (EM) Algorithm

Expectation Maximization (EM) Algorithm

EM is an iterative algorithm used to compute Maximum Likelihood (ML) (or Maximum A posteriori MAP) estimates of observed data (denoted by x) in the presence of missing or hidden data (denoted by z). E.g., EM is used to compute:

$$\theta_{\text{ML}} = \arg \max_{\theta} \sum_i \log \sum_z P(x_i, z | \theta)$$

Two step iterative algorithm:

E step: Estimate hidden variables given the observations and current estimates of the model parameters.

M step: Estimating model parameters by maximising an auxiliary function (lower bound of the likelihood function) using estimates of the hidden data from the E step.

EM Algorithm: Fitting Parameters to Data

Observed data: i.i.d samples $x_i, i=1, \dots, N$

Hidden data: Denoted by z

Goal: Find $\arg \max_{\theta} \mathcal{L}(\theta)$ where $\mathcal{L}(\theta) = \sum_{i=1}^N \log \Pr(x_i; \theta)$

Initial parameters: θ^0 (x is observed and z is hidden)

Iteratively compute θ^ℓ as follows that optimises an auxiliary function $Q(\theta, \theta^{\ell-1})$:

$$Q(\theta, \theta^{\ell-1}) = \sum_{i=1}^N \sum_z \Pr(z|x_i; \theta^{\ell-1}) \log \Pr(x_i, z; \theta)$$

$$\theta^\ell = \arg \max_{\theta} Q(\theta, \theta^{\ell-1})$$

EM is guaranteed to converge to a local optimum or saddle points [Wu83]

Coin example to illustrate EM



$$\rho_1 = \Pr(H)$$

$$\rho_2 = \Pr(H)$$

$$\rho_3 = \Pr(H)$$

Repeat:

Toss Coin 1 privately
if it shows H:

Toss Coin 2 twice
else

Toss Coin 3 twice

The following sequence is observed: “HH, TT, HH, TT, HH”

How do you estimate ρ_1 , ρ_2 and ρ_3 ?

Coin example to illustrate EM

Recall, for partially observed data, the log likelihood is given by:

$$\mathcal{L}(\theta) = \sum_{i=1}^N \log \Pr(x_i; \theta) = \sum_{i=1}^N \log \sum_z \Pr(x_i, z; \theta)$$

where, for the coin example:

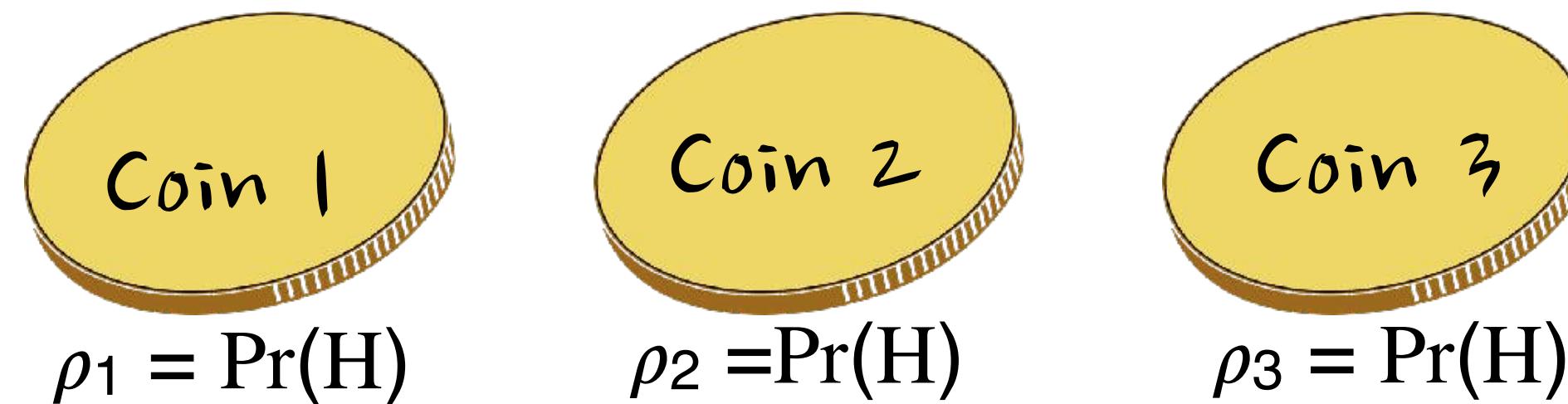
- each observation $x_i \in \mathcal{X} = \{\text{HH}, \text{HT}, \text{TH}, \text{TT}\}$
- the hidden variable $z \in \mathcal{Z} = \{\text{H}, \text{T}\}$

Coin example to illustrate EM

Recall, for partially observed data, the log likelihood is given by:

$$\mathcal{L}(\theta) = \sum_{i=1}^N \log \Pr(x_i; \theta) = \sum_{i=1}^N \log \sum_z \Pr(x_i, z; \theta)$$

$$\Pr(x, z; \theta) = \Pr(x|z; \theta) \Pr(z; \theta)$$



where $\Pr(z; \theta) = \begin{cases} \rho_1 & \text{if } z = \text{H} \\ 1 - \rho_1 & \text{if } z = \text{T} \end{cases}$

$$\Pr(x|z; \theta) = \begin{cases} \rho_2^h (1 - \rho_2)^t & \text{if } z = \text{H} \\ \rho_3^h (1 - \rho_3)^t & \text{if } z = \text{T} \end{cases}$$

h : number of heads, t : number of tails

Coin example to illustrate EM

Our observed data is: {HH, TT, HH, TT, HH}

Let's use EM to estimate $\theta = (\rho_1, \rho_2, \rho_3)$

[EM Iteration, E-step]

Compute quantities involved in

$$Q(\theta, \theta^{\ell-1}) = \sum_{i=1}^N \sum_z \gamma(z, x_i) \log \Pr(x_i, z; \theta)$$

where $\gamma(z, x) = \Pr(z \mid x ; \theta^{\ell-1})$

i.e., compute $\gamma(z, x_i)$ for all z and all i

Suppose $\theta^{\ell-1}$ is $\rho_1 = 0.3, \rho_2 = 0.4, \rho_3 = 0.6$:

What is $\gamma(H, HH)$? **= 0.16**

What is $\gamma(H, TT)$? **= 0.49**

Coin example to illustrate EM

Our observed data is: {HH, TT, HH, TT, HH}

Let's use EM to estimate $\theta = (\rho_1, \rho_2, \rho_3)$

Find θ which maximises $Q(\theta, \theta^{\ell-1}) = \sum_{i=1}^N \sum_z \gamma(z, x_i) \log \Pr(x_i, z; \theta)$

Coin example to illustrate EM

Our observed data is: {HH, TT, HH, TT, HH}

Let's use EM to estimate $\theta = (\rho_1, \rho_2, \rho_3)$

[EM Iteration, M-step]

Find θ which maximises

$$Q(\theta, \theta^{\ell-1}) = \sum_{i=1}^N \sum_z \gamma(z, x_i) \log \Pr(x_i, z; \theta)$$

$$\rho_1 = \frac{\sum_{i=1}^N \gamma(H, x_i)}{N}$$

$$\rho_2 = \frac{\sum_{i=1}^N \gamma(H, x_i) h_i}{\sum_{i=1}^N \gamma(H, x_i) (h_i + t_i)}$$

$$\rho_3 = \frac{\sum_{i=1}^N \gamma(T, x_i) h_i}{\sum_{i=1}^N \gamma(T, x_i) (h_i + t_i)}$$

Baum-Welch Algorithm

Observed data: N sequences, x_i , $i=1..N$ where $x_i \in V$

Parameters θ of an HMM: transition matrix A , observation probabilities B

[EM Iteration, E-step]

Compute quantities involved in $Q(\theta, \theta^{\ell-1})$

$$\gamma_{i,t}(j) = \Pr(z_t = j \mid x_i; \theta^{\ell-1})$$

$$\xi_{i,t}(j,k) = \Pr(z_t = j, z_{t+1} = k \mid x_i; \theta^{\ell-1})$$

Baum Welch Algorithm for GMM-HMMs

[Every EM Iteration]

Compute $\theta = \{ A_{jk}, (\mu_{jm}, \Sigma_{jm}, c_{jm}) \}$ for all j, k, m

$$A_{j,k} = \frac{\sum_{i=1}^N \sum_{t=2}^{T_i} \xi_{i,t}(j, k)}{\sum_{i=1}^N \sum_{t=2}^{T_i} \sum_{k'} \xi_{i,t}(j, k')}$$

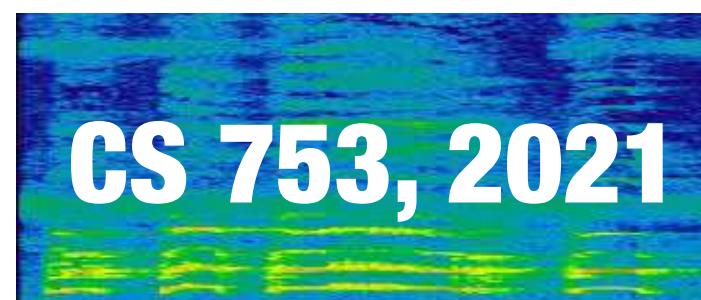
$$\mu_{jm} = \frac{\sum_{i=1}^N \sum_{t=1}^{T_i} \gamma_{i,t}(j, m) x_{it}}{\sum_{i=1}^N \sum_{t=1}^{T_i} \gamma_{i,t}(j, m)}$$

$$\Sigma_{jm} = \frac{\sum_{i=1}^N \sum_{t=1}^{T_i} \gamma_{i,t}(j, m) (x_{it} - \mu_{jm})(x_{it} - \mu_{jm})^T}{\sum_{i=1}^N \sum_{t=1}^{T_i} \gamma_{i,t}(j, m)}$$

$$c_{jm} = \frac{\sum_{i=1}^N \sum_{t=1}^{T_i} \gamma_{i,t}(j, m)}{\sum_{i=1}^N \sum_{t=1}^{T_i} \sum_{m'=1}^M \gamma_{i,t}(j, m')}$$

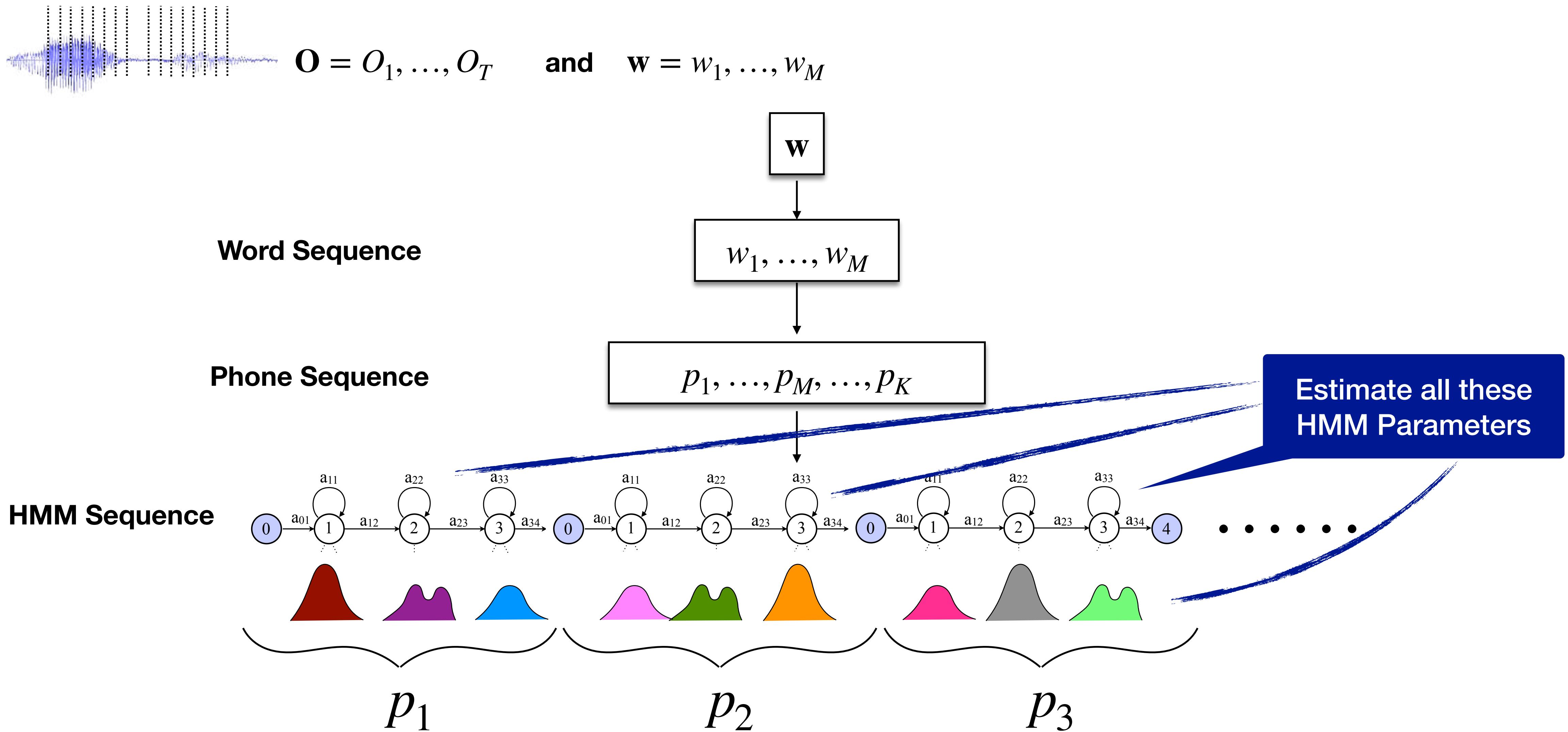
Tied-State HMMs for Acoustic Modeling

Lecture 2a



Instructor: Preethi Jyothi, IITB

Recap: HMM-based Acoustic Models

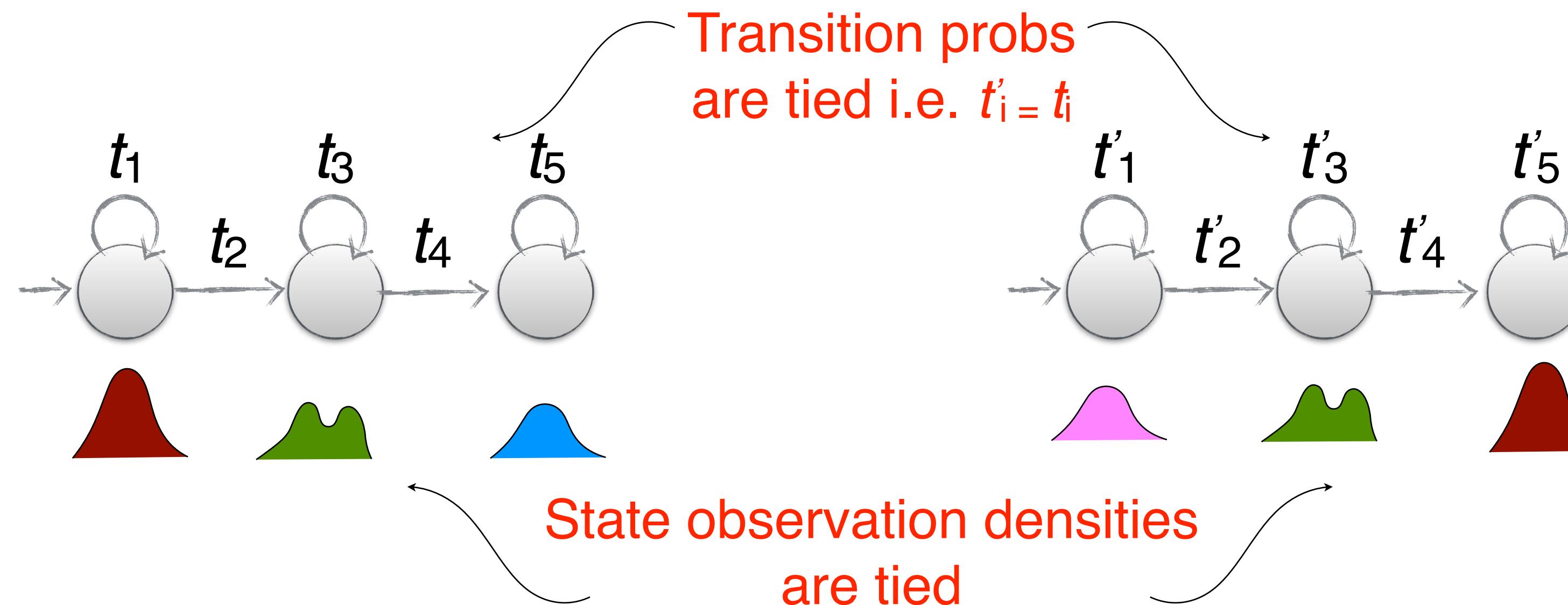


Triphone HMM Models

- Each phone is modelled in the context of its left and right neighbour phones
 - Pronunciation of a phone is influenced by the preceding and succeeding phones.
E.g. The phone [p] in the word “*peek*” : p iy k” vs. [p] in the word “*pool*” : p uw l
- Number of triphones that appear in data $\approx 1000s$ or 10,000s
- If each triphone HMM has 3 states and each state generates m -component GMMs ($m \approx 64$), for d -dimensional acoustic feature vectors ($d \approx 40$) with Σ having d^2 parameters
 - Hundreds of millions of parameters!
- Insufficient data to learn all triphone models reliably. What do we do? Share parameters across triphone models!

Parameter Sharing

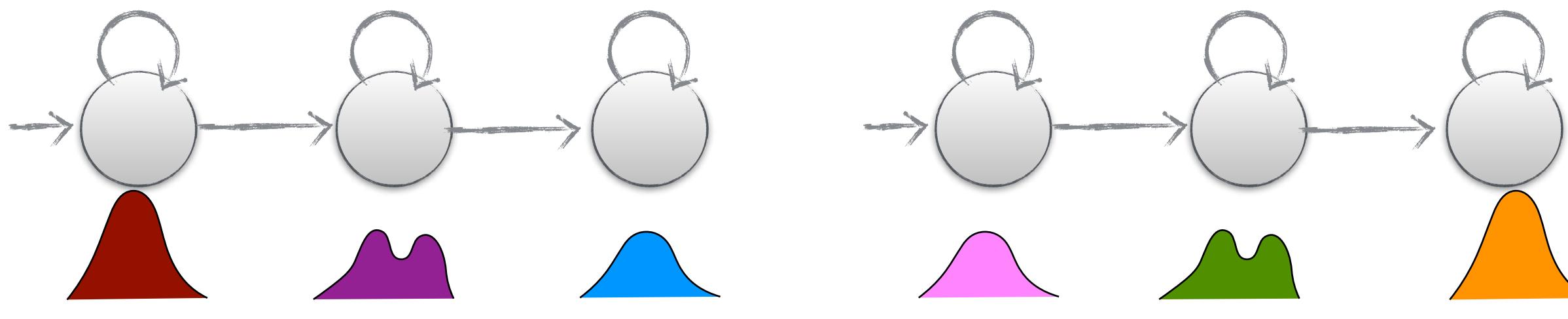
- Sharing of parameters (also referred to as “parameter tying”) can be done at any level:
 - Parameters in HMMs corresponding to two triphones are said to be tied if they are identical



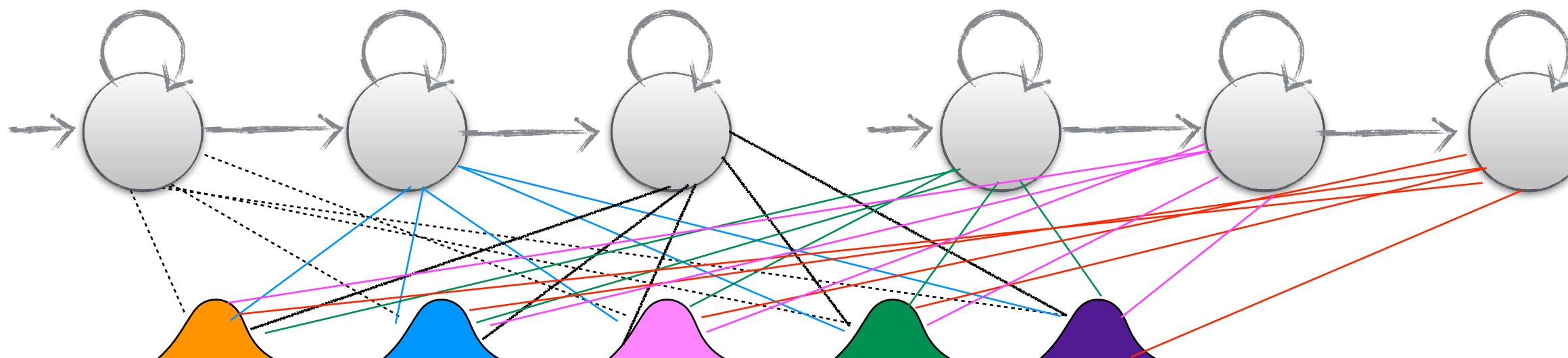
- More parameter tying: Tying variances of all Gaussians within a state, tying variances of all Gaussians in all states, tying individual Gaussians, etc.

1. Tied Mixture Models

- All states share the same Gaussians (i.e. same means and covariances)
- Mixture weights are specific to each state



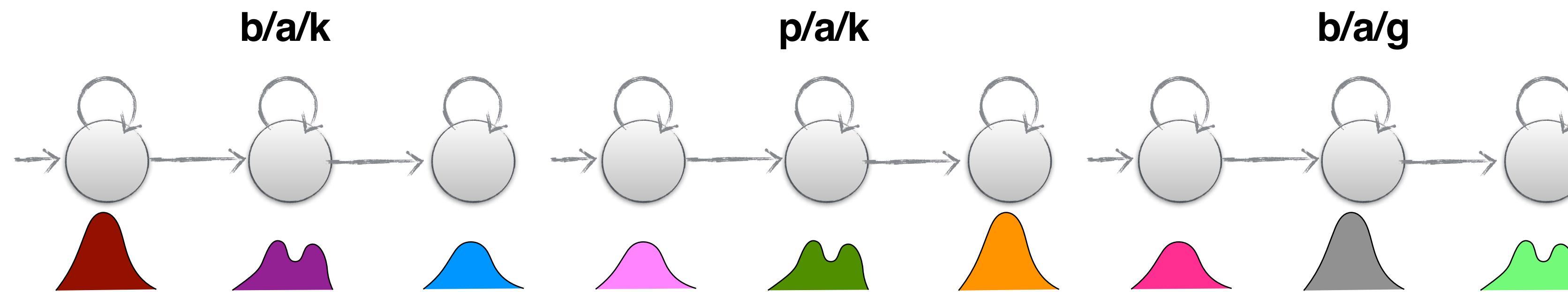
Triphone HMMs (No sharing)



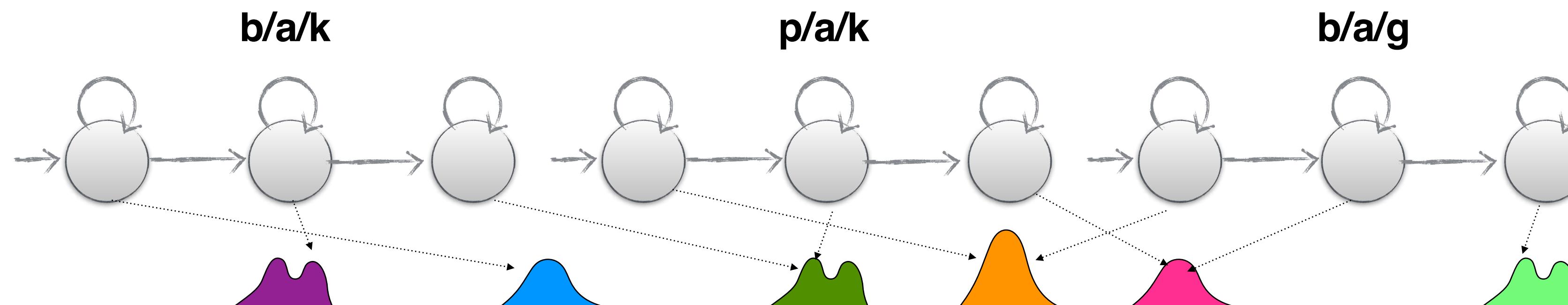
Triphone HMMs (Tied Mixture Models)

2. State Tying

- Observation probabilities are shared across states which generate acoustically similar data



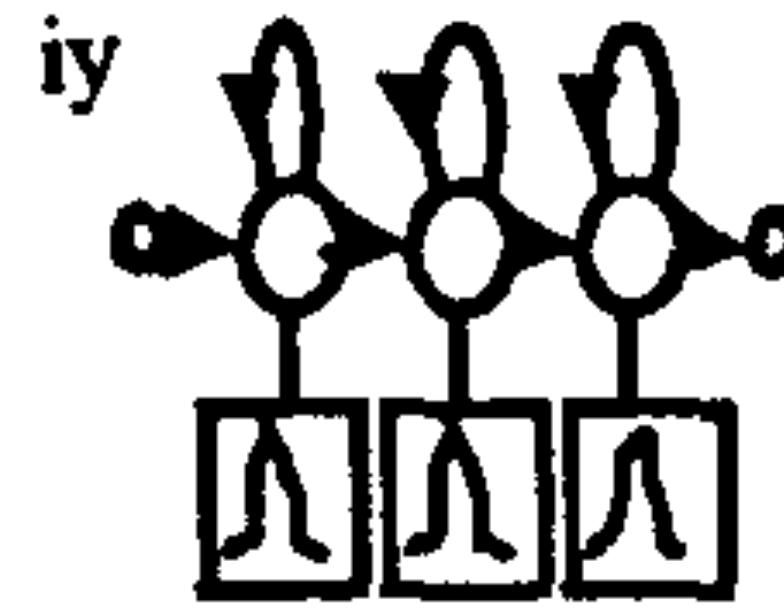
Triphone HMMs (No sharing)



Triphone HMMs (State Tying)

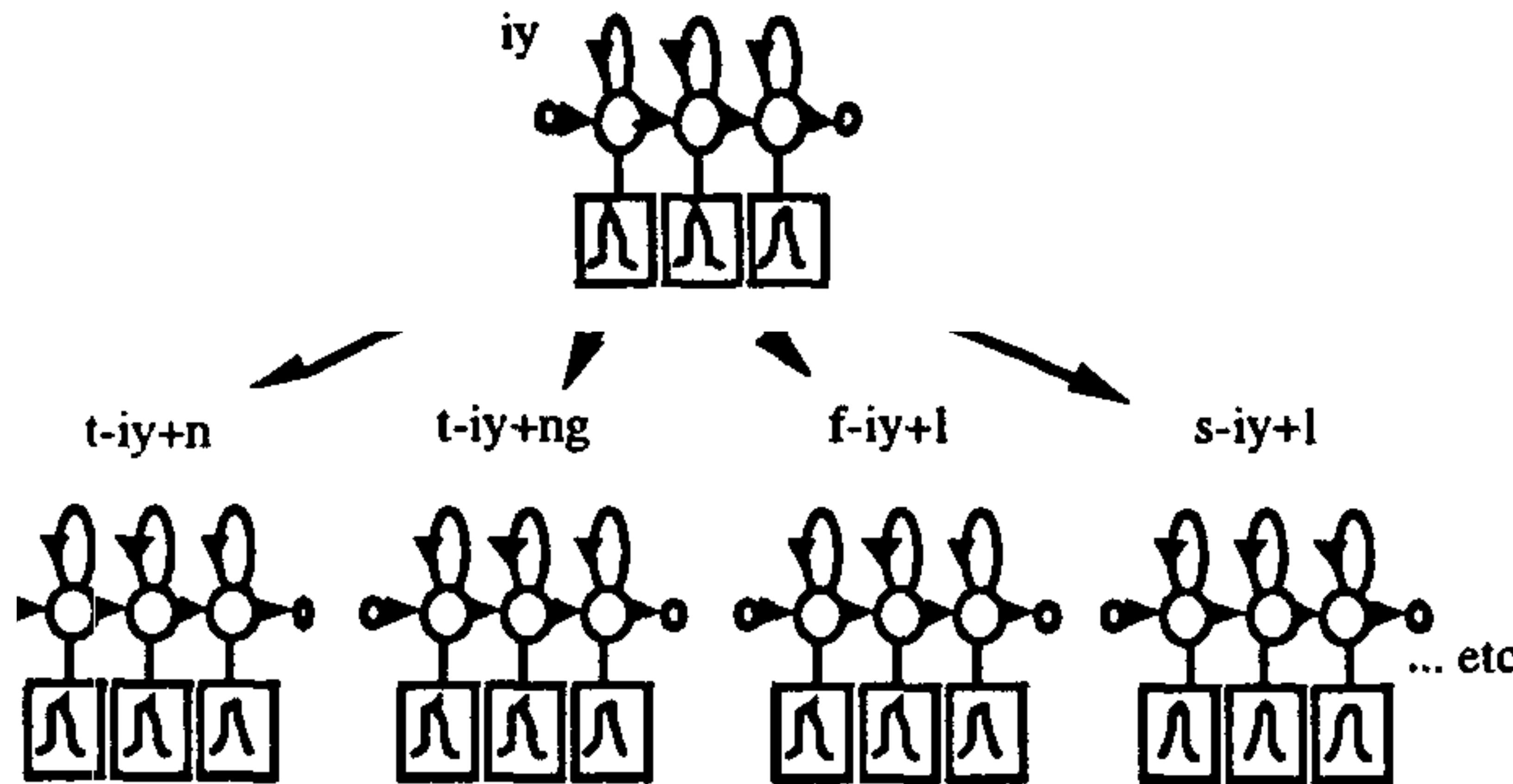
Tied state HMMs: Step 1

Create and train 3-state monophone HMMs with single Gaussian observation probability densities

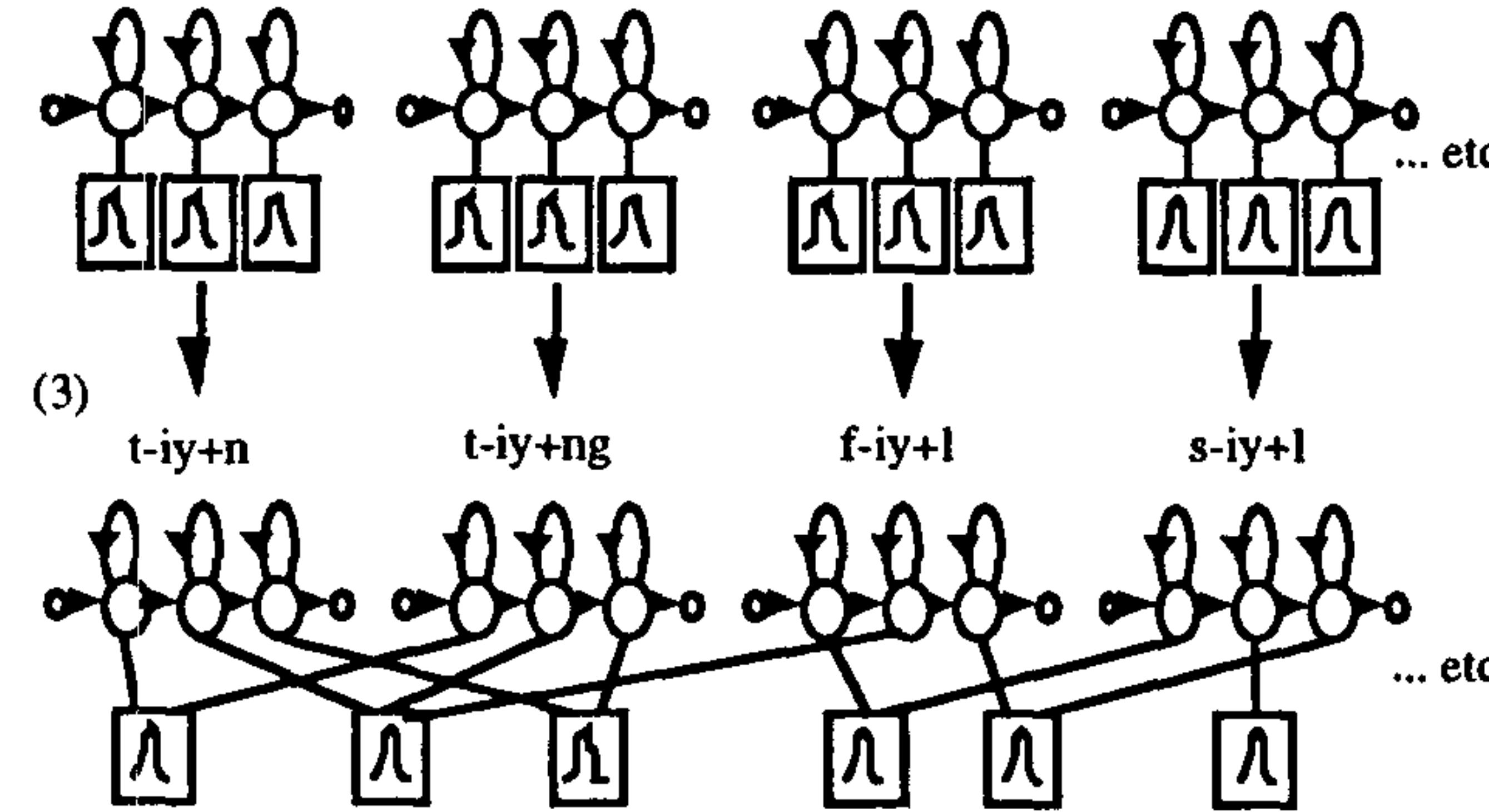


Tied state HMMs: Step 2

Clone these monophone distributions to initialise a set of untied triphone models



Tied state HMMs: Step 3



For all triphones derived from the same monophone, cluster states whose parameters should be tied together.

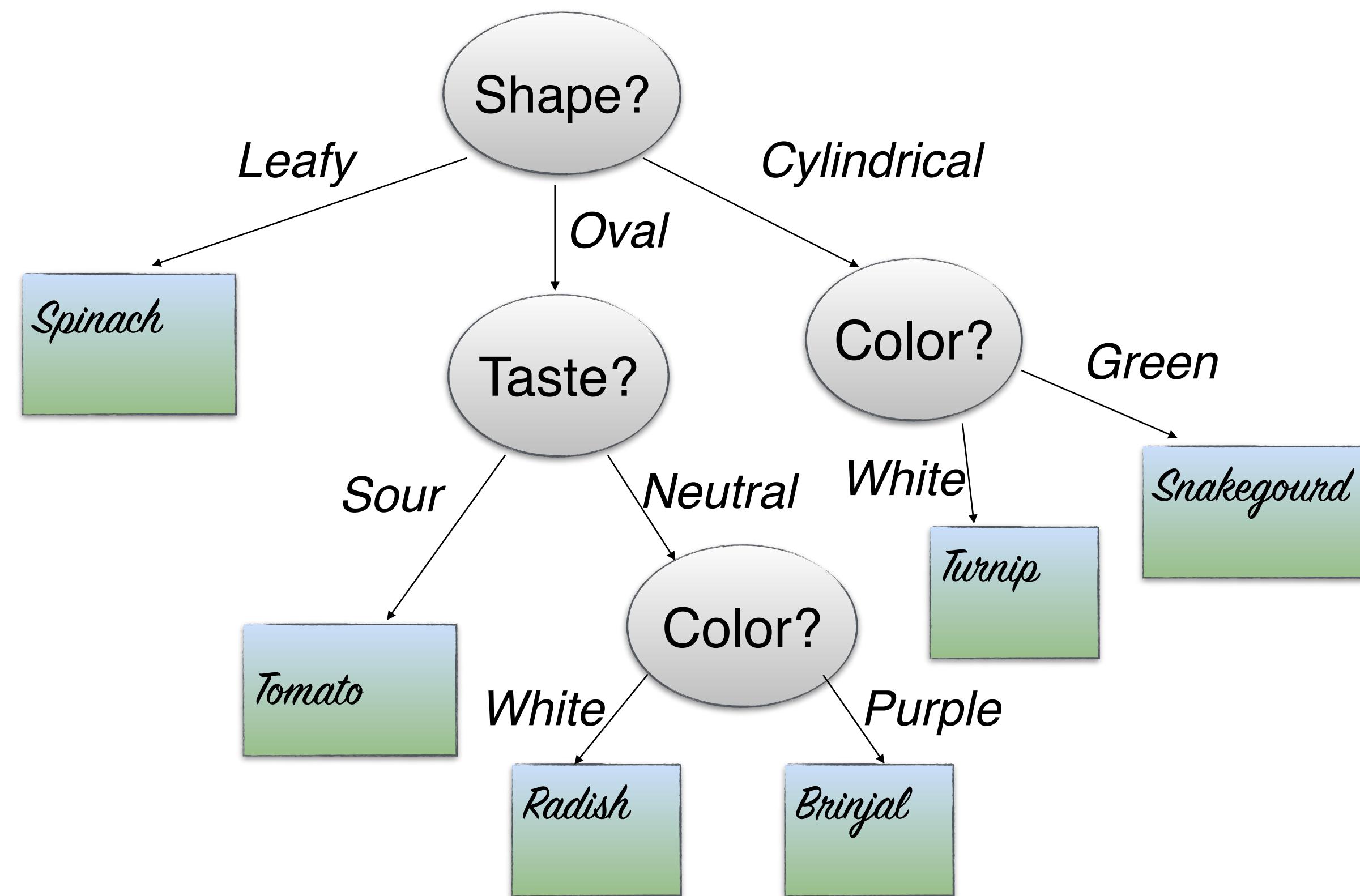
Popular option: Use ***decision trees*** to determine which states should be tied together!

Decision Trees

Classification using a decision tree:

Begins at the root node: What property is satisfied?

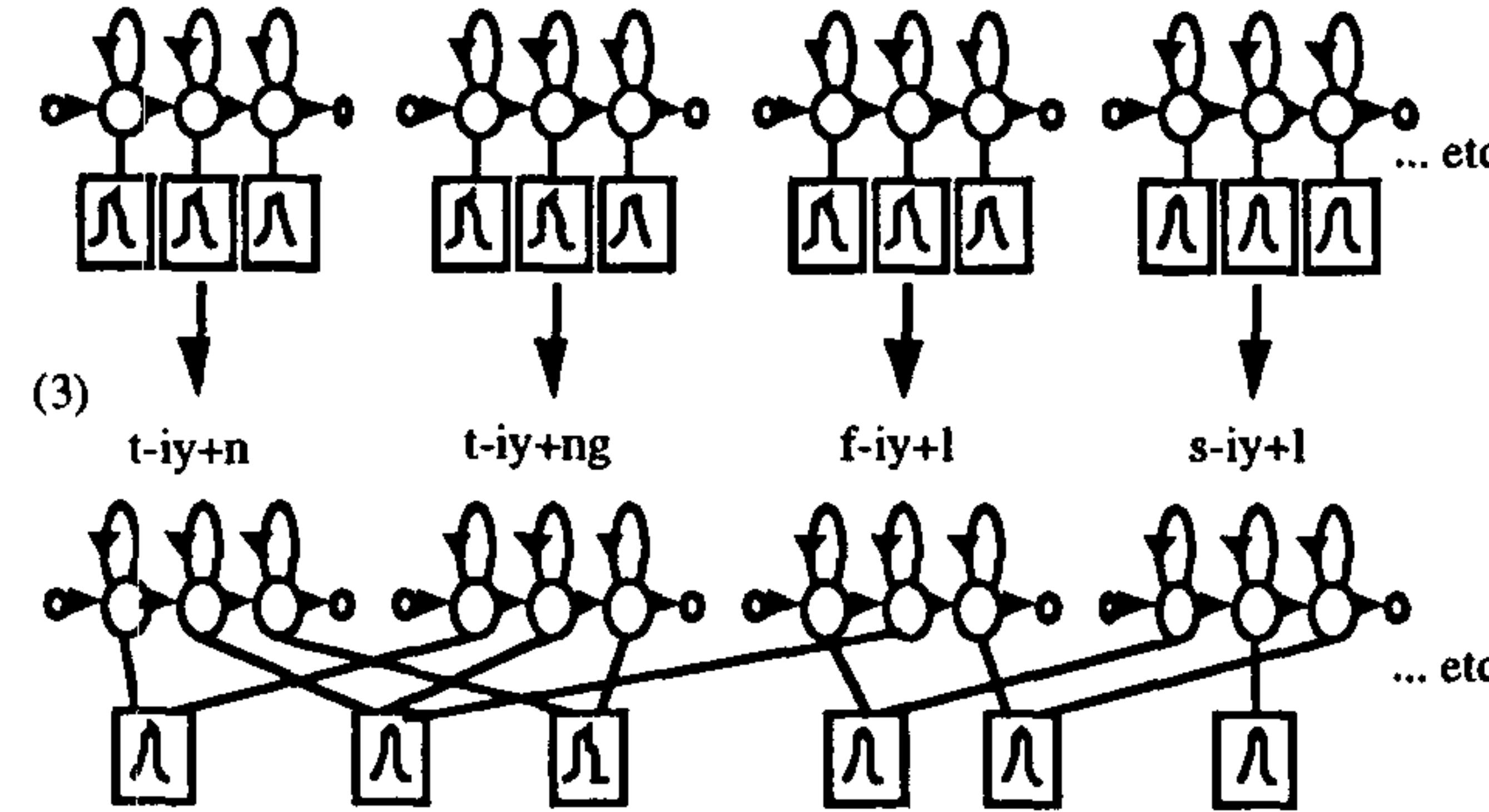
Depending on answer, traverse to different branches



Decision Trees

- Given the data at a node, either declare the node to be a leaf or find another attribute to split the node further.
- Important questions to be addressed for DTs:
 1. How many splits at a node?
Chosen by the user.
 2. Which attribute/question should be used at a node for splitting?
One which decreases “impurity” of nodes as much as possible.
 3. When is a node a leaf?
Set threshold in reduction in impurity

Tied state HMMs: Step 3



For all triphones derived from the same monophone, cluster states whose parameters should be tied together.

Popular option: Use ***decision trees*** to determine which states should be tied together!

Example: Phonetic Decision Tree (DT)

One tree is constructed for each state of each monophone to cluster all the corresponding triphone states

Head node
 $aa_2/ow_2/f_2$, $aa_2/ow_2/s_2$,
 $aa_2/ow_2/d_2$, $h_2/ow_2/p_2$,
 $aa_2/ow_2/n_2$, $aa_2/ow_2/g_2$,
...



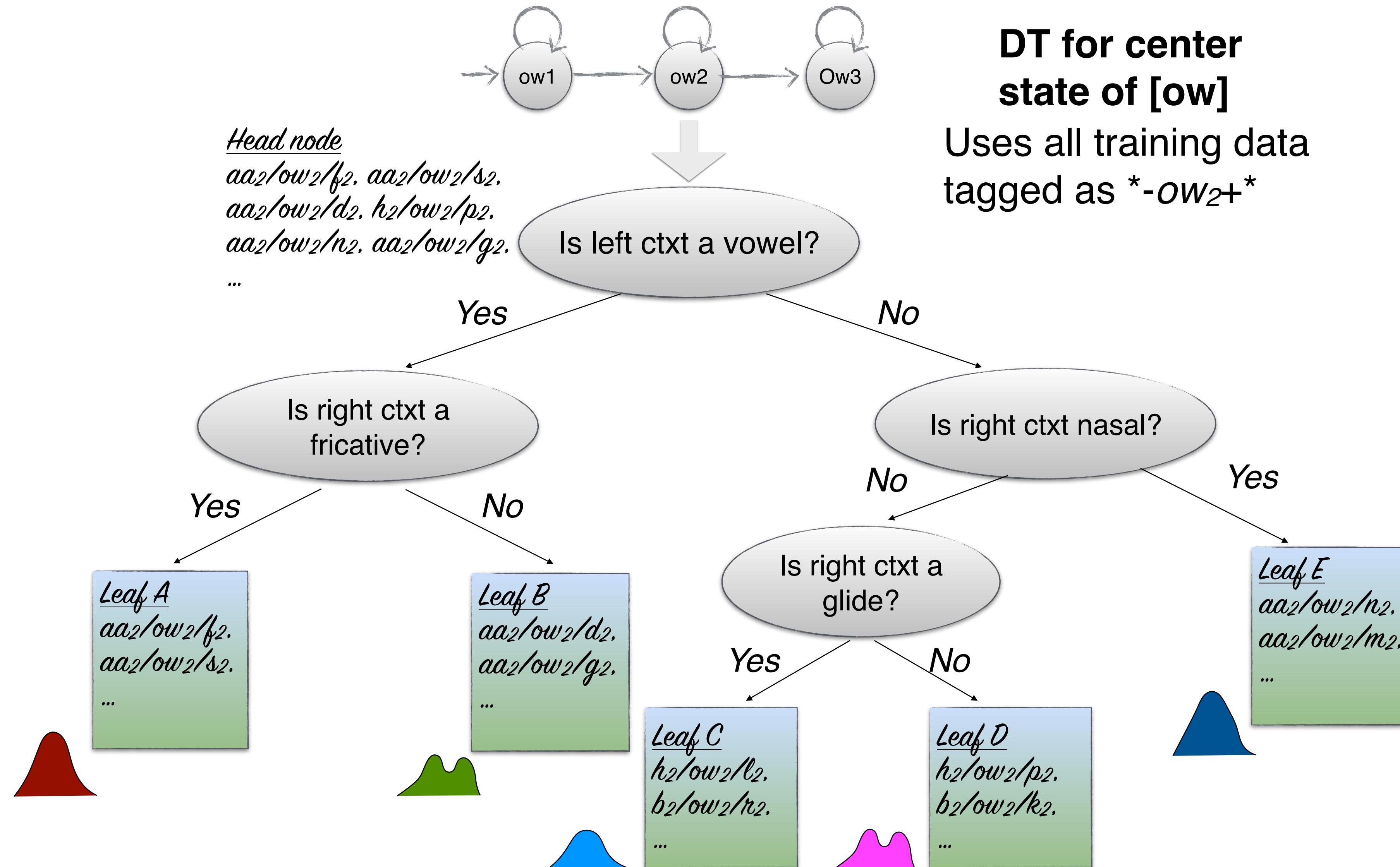
**DT for center
state of [ow]**

Uses all training data
tagged with $*-ow_2+*$

How do we determine this training
data? Coming up in two slides.

Example: Phonetic Decision Tree (DT)

One tree is constructed for each state of each monophone to cluster all the corresponding triphone states



How do we build these phone DTs?

1. What questions are used?

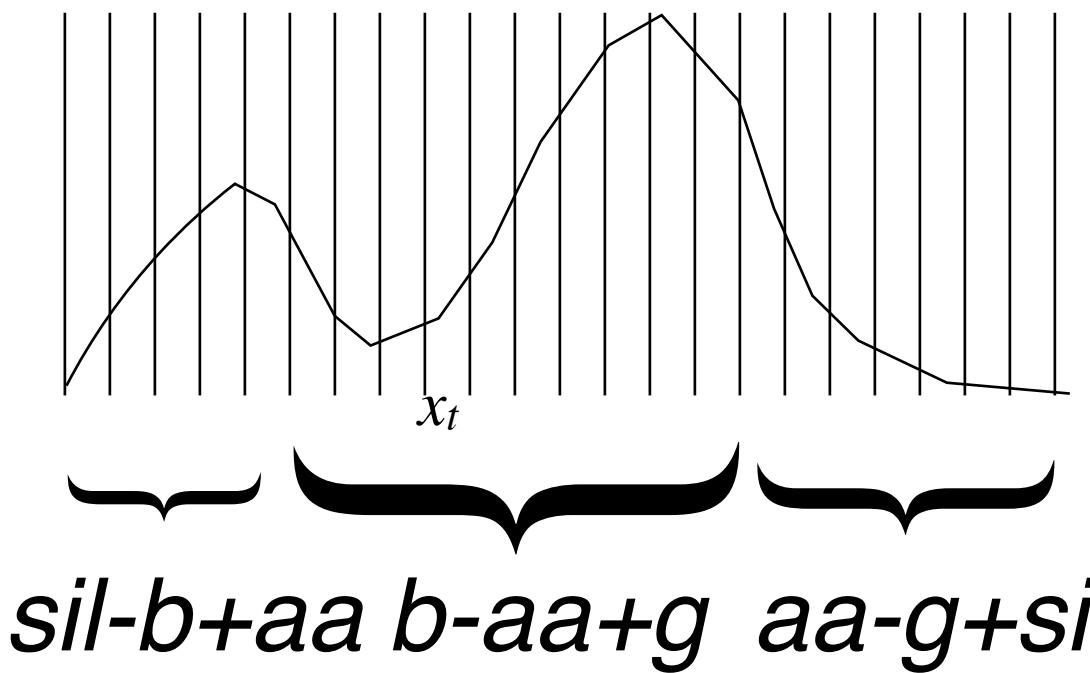
Linguistically-inspired binary questions: “Does the left or right phone come from a broad class of phones such as vowels, stops, etc.?” “Is the left or right phone [k] or [m]?”

2. What is the training data for each phone state, p_j ? (root node of DT)

All speech frames that align with the j^{th} state of every triphone HMM that has p as the middle phone

Training data for DT nodes

- Align training instance $x = (x_1, \dots, x_T)$ where $x_i \in \mathbb{R}^d$ with a set of triphone HMMs
- Use Viterbi algorithm to find the best HMM triphone state sequence corresponding to each x
- Tag each x_t with ID of current phone along with left-context and right-context



x_t is tagged with ID $b_2\text{-}aa_2\text{+}g_2$ i.e. x_t is aligned with the second state of the 3-state HMM corresponding to the triphone b-aa+g

- Training data corresponding to state j in phone p : Gather all x_t 's that are tagged with ID $*\text{-}p_j\text{+}* \text{ }$

How do we build these phone DTs?

1. What questions are used?

Linguistically-inspired binary questions: “Does the left or right phone come from a broad class of phones such as vowels, stops, etc.?” “Is the left or right phone [k] or [m]?”

2. What is the training data for each phone state, p_j ? (root node of DT)

All speech frames that align with the j^{th} state of every triphone HMM that has p as the middle phone

3. What criterion is used at each node to find the best question to split the data on?

Find the question which partitions the states in the parent node so as to give the maximum increase in log likelihood

Likelihood of a cluster of states

- If a cluster of HMM states, $S = \{s_1, s_2, \dots, s_M\}$ consists of M states and a total of K acoustic observation vectors are associated with $S, \{x_1, x_2 \dots, x_K\}$, then the log likelihood associated with S is:

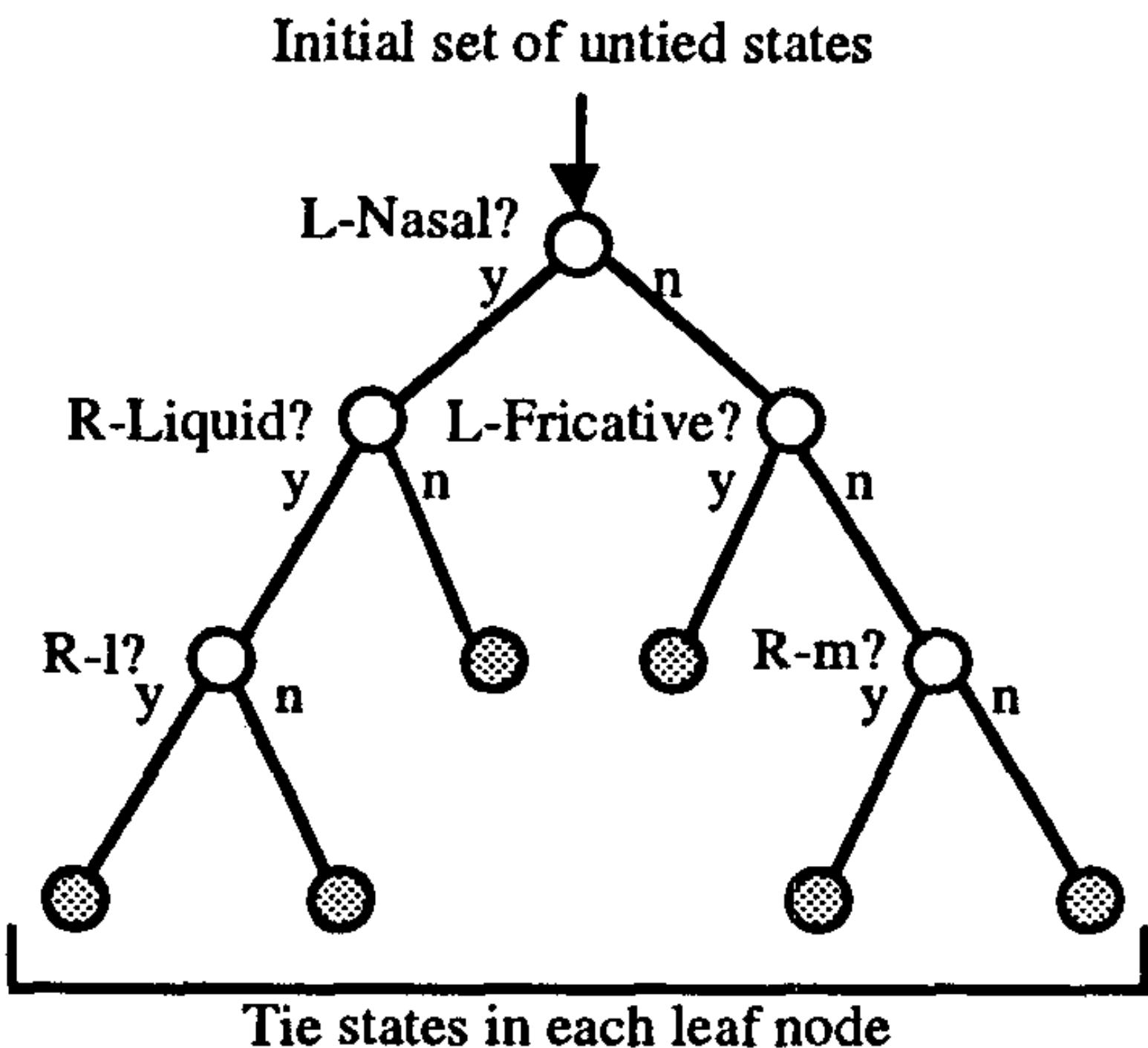
$$\mathcal{L}(S) = \sum_{i=1}^K \sum_{s \in S} \log \Pr(x_i; \mu_s, \Sigma_s) \gamma_s(x_i)$$

- For a question q that splits S into S_{yes} and S_{no} , compute the following quantity:

$$\Delta_q = \mathcal{L}(S_{yes}^q) + \mathcal{L}(S_{no}^q) - \mathcal{L}(S)$$

- Go through all questions, find Δ_q for each question q and choose the question for which Δ_q is the biggest
- Terminate when: Final Δ_q is below a threshold or data associated with a split falls below a threshold

Likelihood criterion

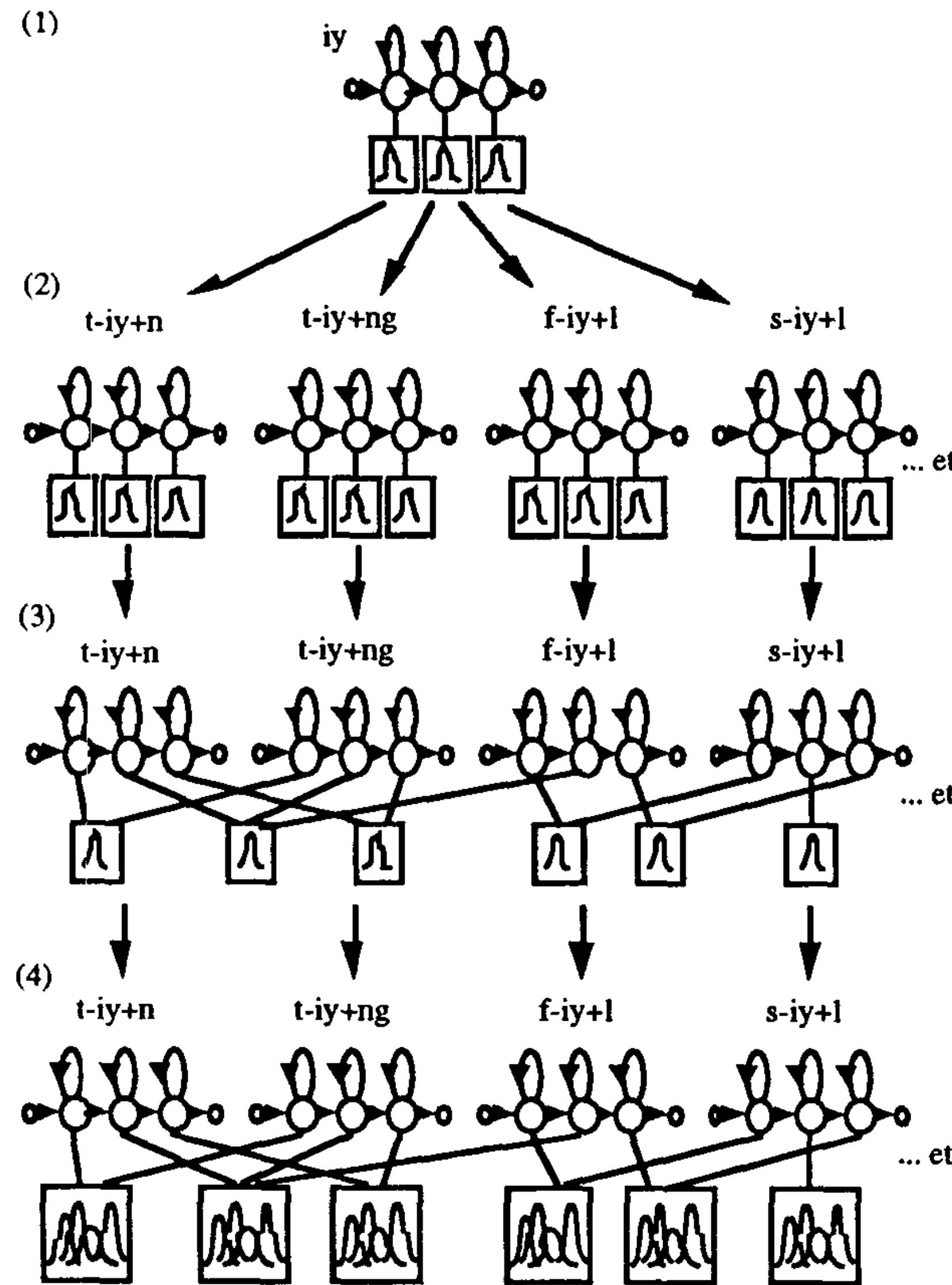


Given a phonetic question, let the initial set of untied states S be split into two partitions S_{yes} and S_{no}

Each partition is clustered to form a single Gaussian output distribution with mean $\mu_{S_{yes}}$ and covariance $\Sigma_{S_{yes}}$

Use the likelihood of the parent state and the subsequent split states to determine which question a node should be split on

Tied state HMMs

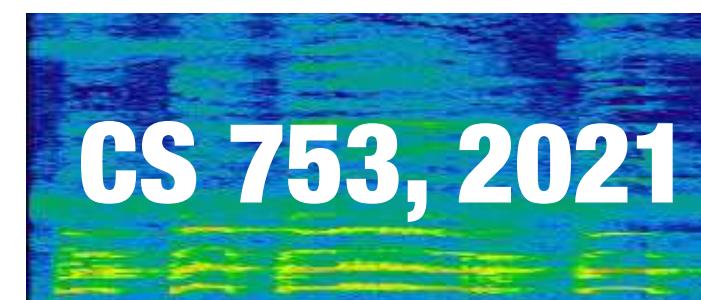


Four main steps in building a tied state HMM system:

1. Create and train 3-state monophone HMMs with single Gaussian observation probability densities
2. Clone these monophone distributions to initialise a set of untied triphone models. Train them using Baum-Welch estimation. Transition matrix remains common across all triphones of each phone.
3. For all triphones derived from the same monophone, cluster states whose parameters should be tied together.
4. Number of mixture components in each tied state is increased and models re-estimated using BW

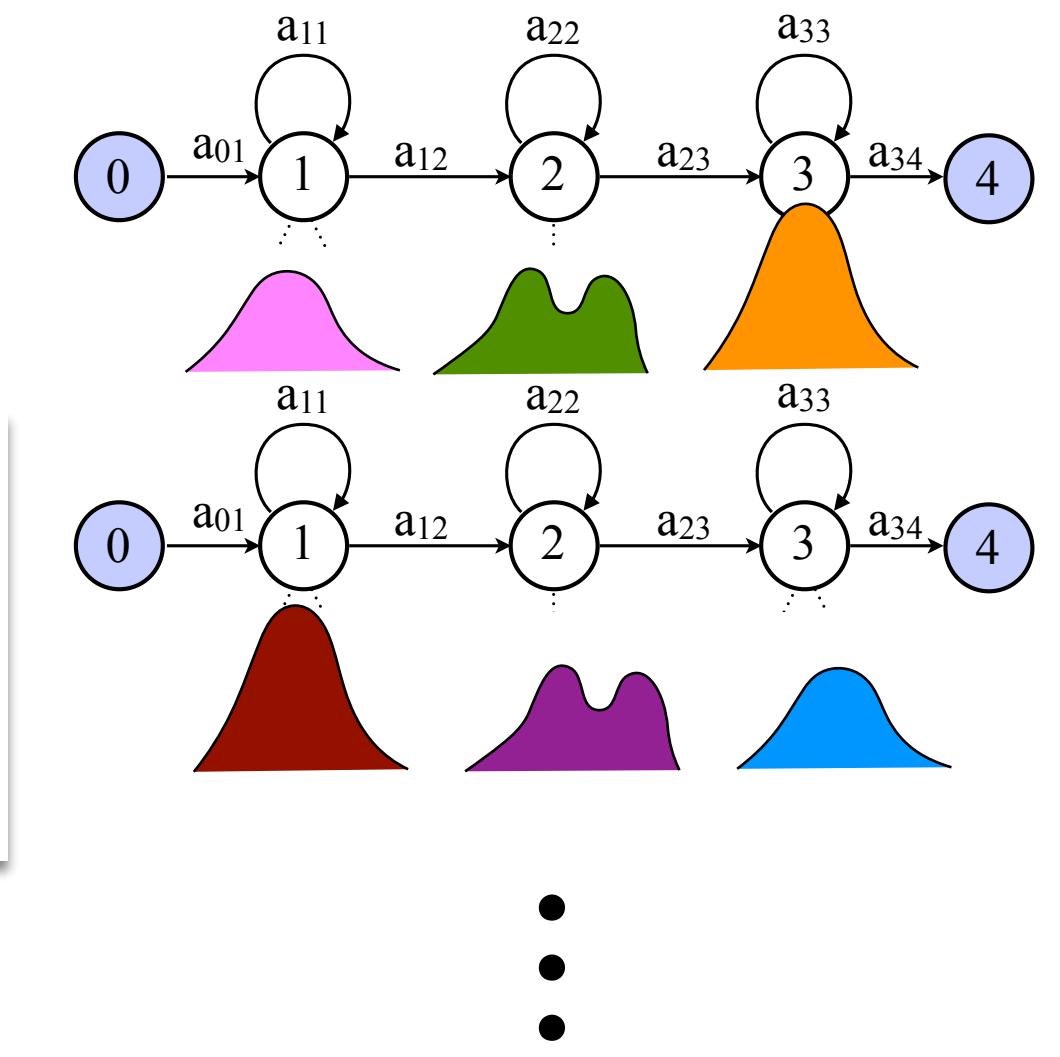
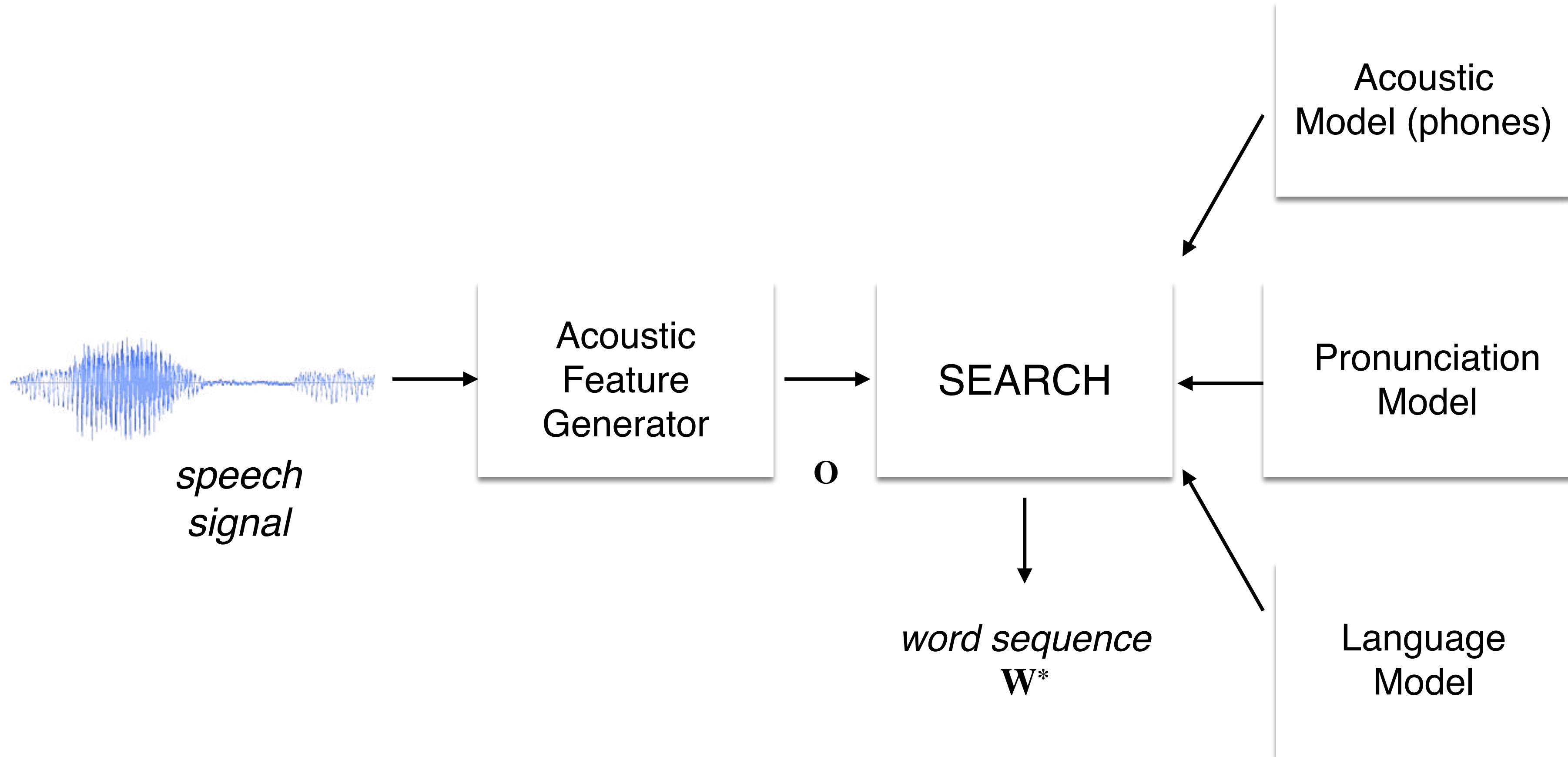
Introduction to WFSTs

Lecture 2b



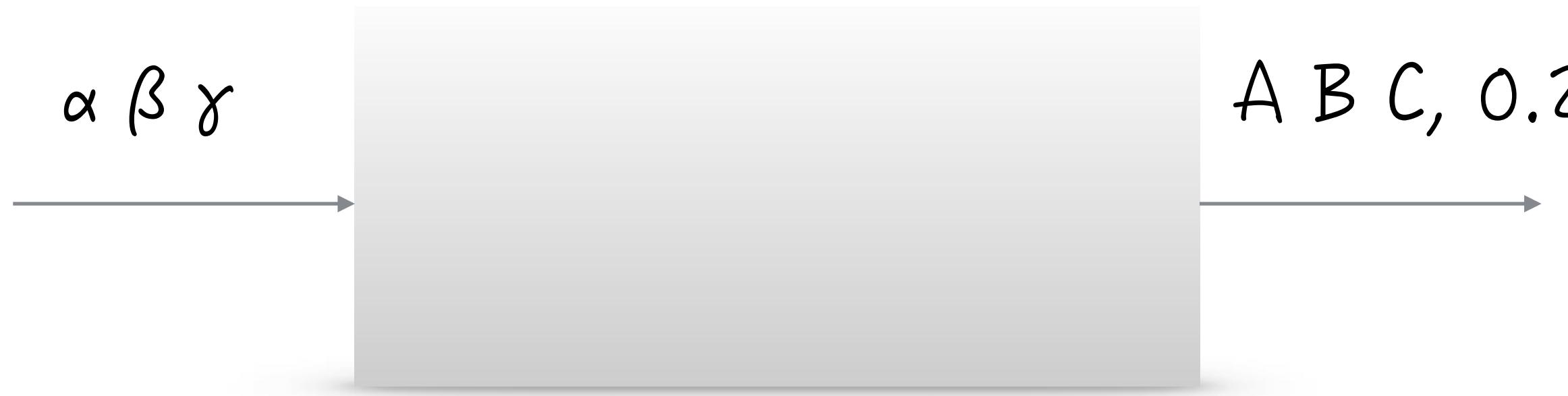
Instructor: Preethi Jyothi, IITB

Architecture of an ASR system



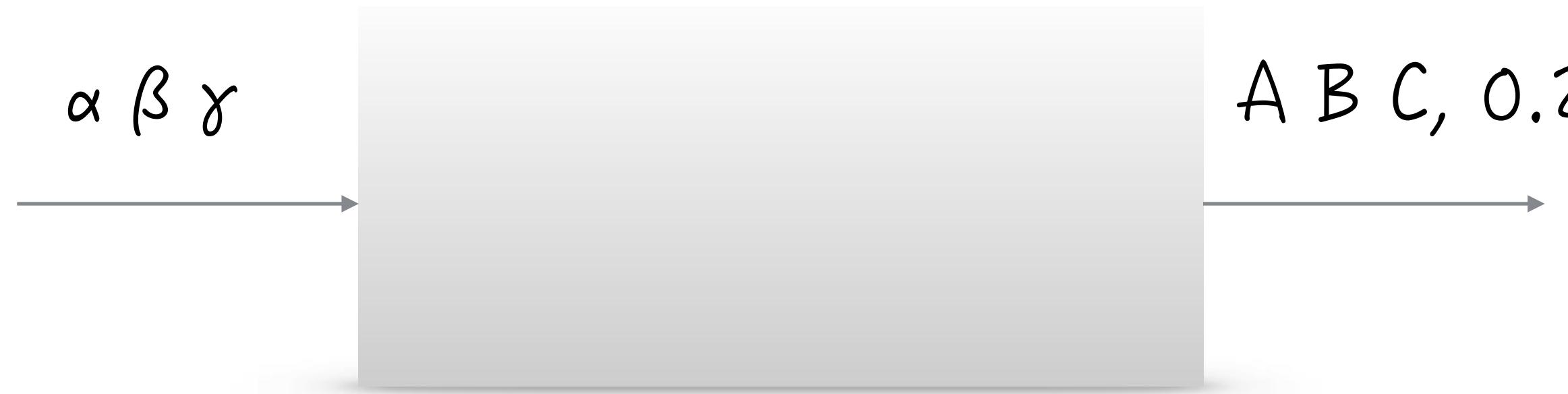
How do we combine the acoustic model HMMs with the remaining components? Weighted finite-state transducers (WFSTs)!

(Weighted) Automaton



- Accepts a subset of strings (over an alphabet), and rejects the rest
 - Mathematically, specified by $L \subseteq \Sigma^*$ or equivalently $f : \Sigma^* \rightarrow \{0,1\}$
- Weighted: outputs a “weight” as well (e.g., probability)
 - $f : \Sigma^* \rightarrow W$
- Transducer: outputs another string (over possibly another alphabet)
 - $f : \Sigma^* \times \Delta^* \rightarrow W$

(Weighted) Finite State Automaton

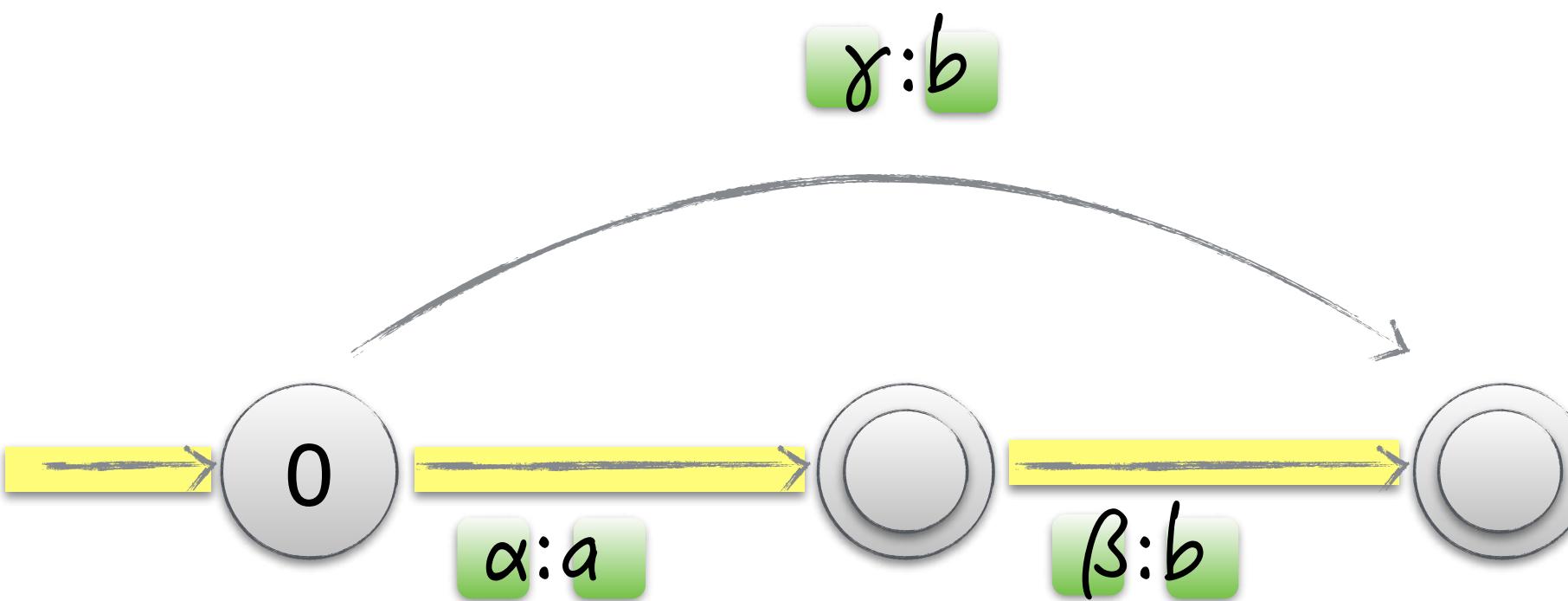


- Functions that can be implemented using a machine which:
 - reads the string one symbol at a time
 - has a fixed amount of memory: so, at any moment, the machine can be in only one of finitely many *states*, irrespective of the length of the input string
- Allows efficient algorithms to reason about the machine
 - e.g., output string with maximum weight for input $\alpha\beta\gamma$

Why WFSTs?

- Powerful enough to (reasonably) model processes in language, speech, computational biology and other machine learning applications
- Simpler WFSTs can be combined to create complex WFSTs, e.g., speech recognition systems
- If using WFST models, efficient algorithms available to train the models and to make inferences
- Toolkits that don't have domain specific dependencies

Structure: Finite State Transducer (FST)

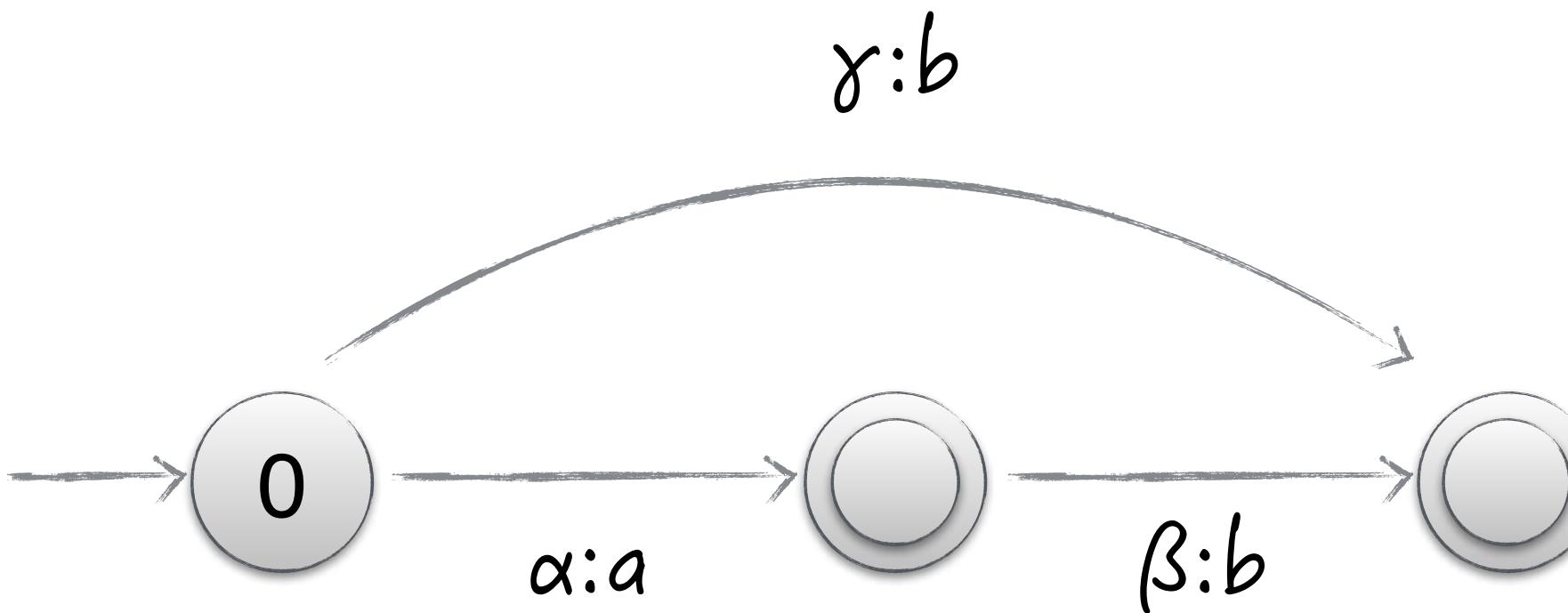


Elements of an FST

- States
- Start state (0)
- Final states (1 & 2)
- Arcs (transitions)
- Input symbols (from alphabet Σ)
- Output symbols (from alphabet Δ)

FST maps input strings to output strings

Path



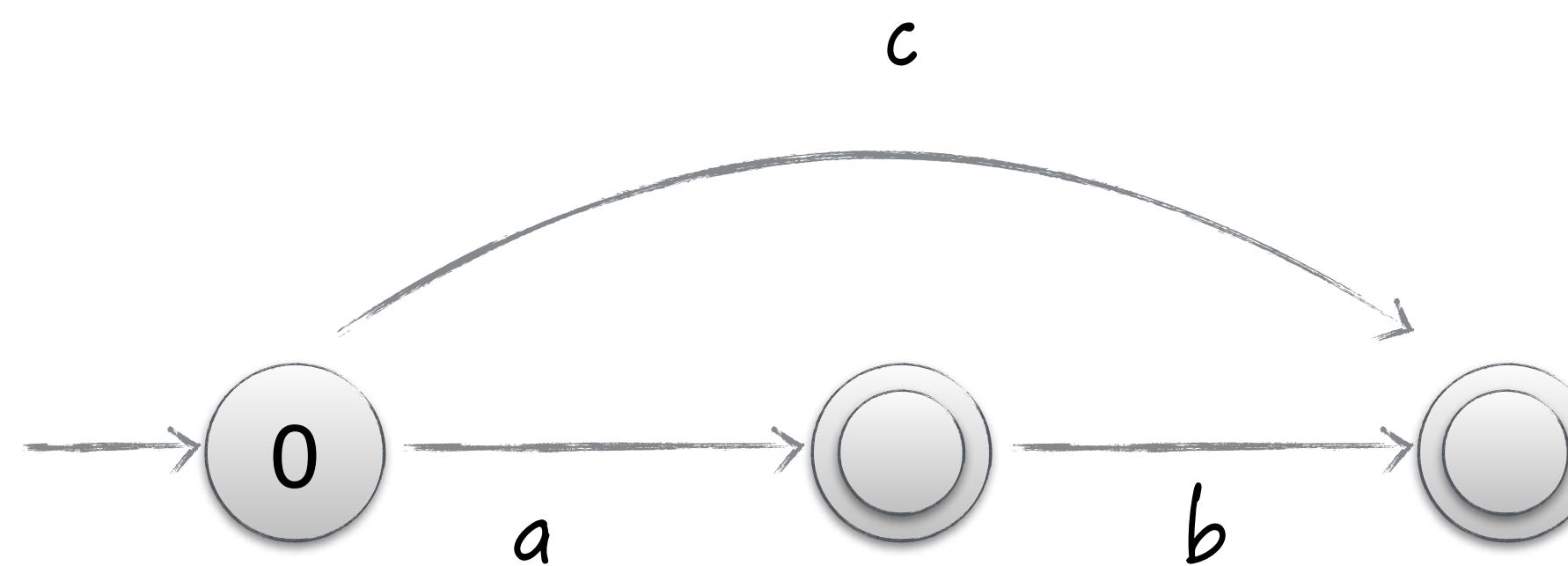
- A successful “path” → Sequence of transitions from the start state to any final state
- Input label of a path → Concatenation of input labels on arcs.
Similarly for output label of a path.

FSAs and FSTs

- Finite state acceptors (FSAs)
 - Each transition has a source & destination state, and **a label**
 - FSA accepts a set of strings, $L \subseteq \Sigma^*$
- Finite state transducers (FSTs)
 - Each transition has a source & destination state, **an input label and an output label**
 - FST represents a relation, $R \subseteq \Sigma^* \times \Delta^*$

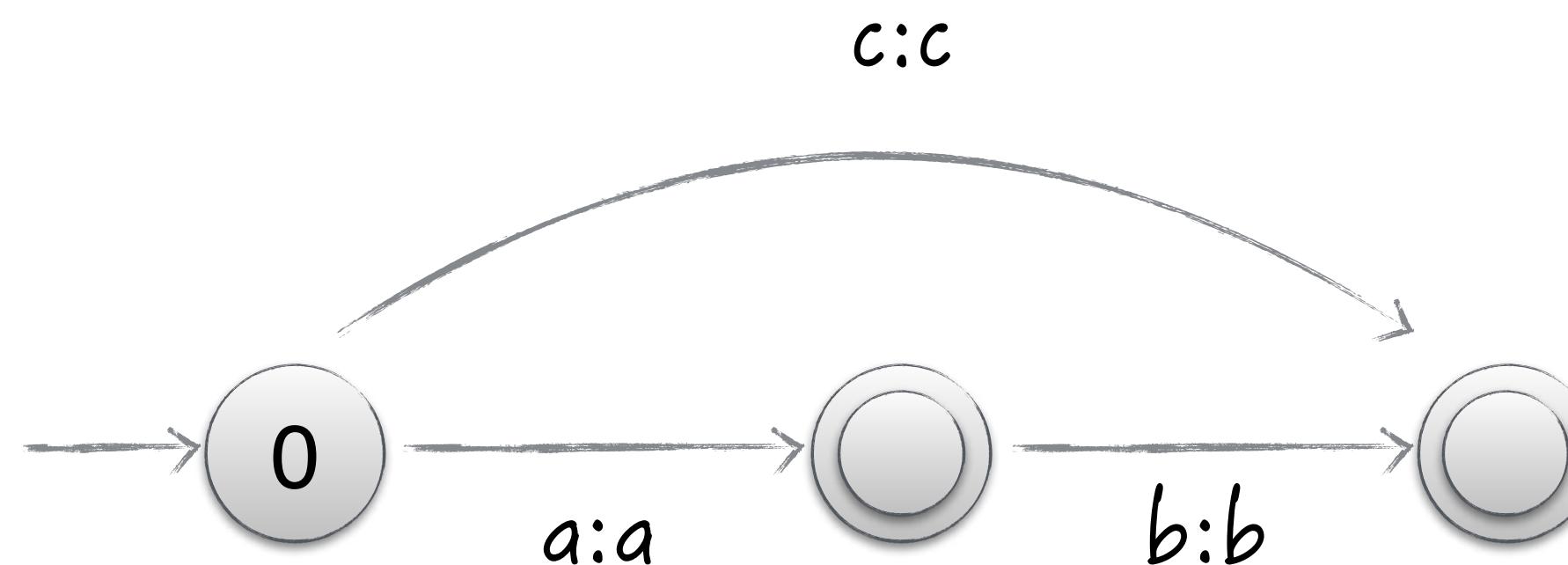
FSA can be thought of as a
special kind of FST

Example of an FSA



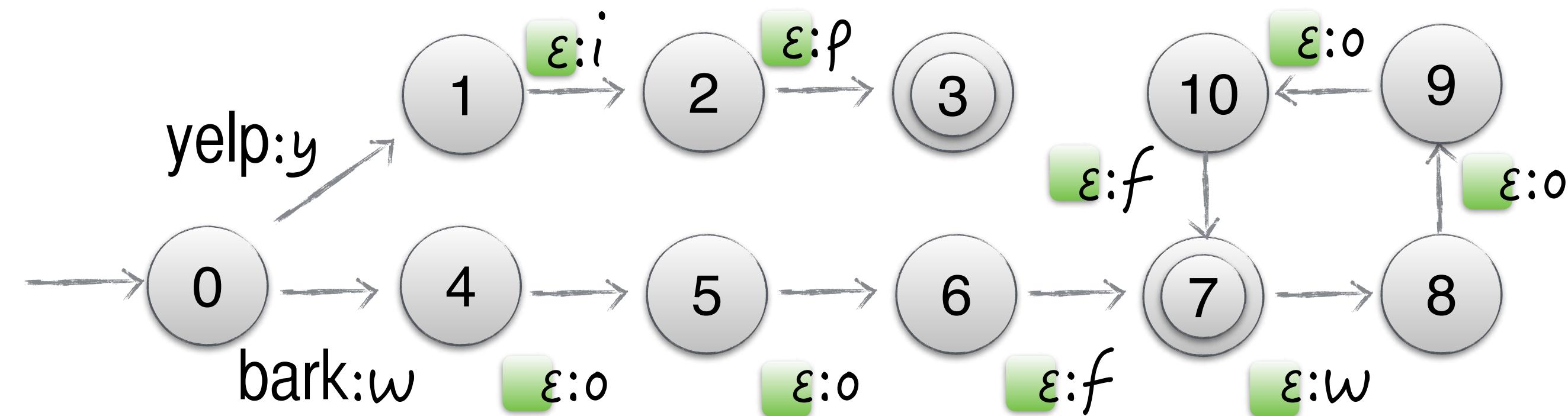
Accepts strings
 $\{c, a, ab\}$

Equivalent FST



Accepts strings
 $\{c, a, ab\}$
and outputs identical strings
 $\{c, a, ab\}$

Barking dog FST



$$\Sigma = \{ \text{yelp}, \text{bark} \}, \Delta = \{ a, \dots, z \}$$

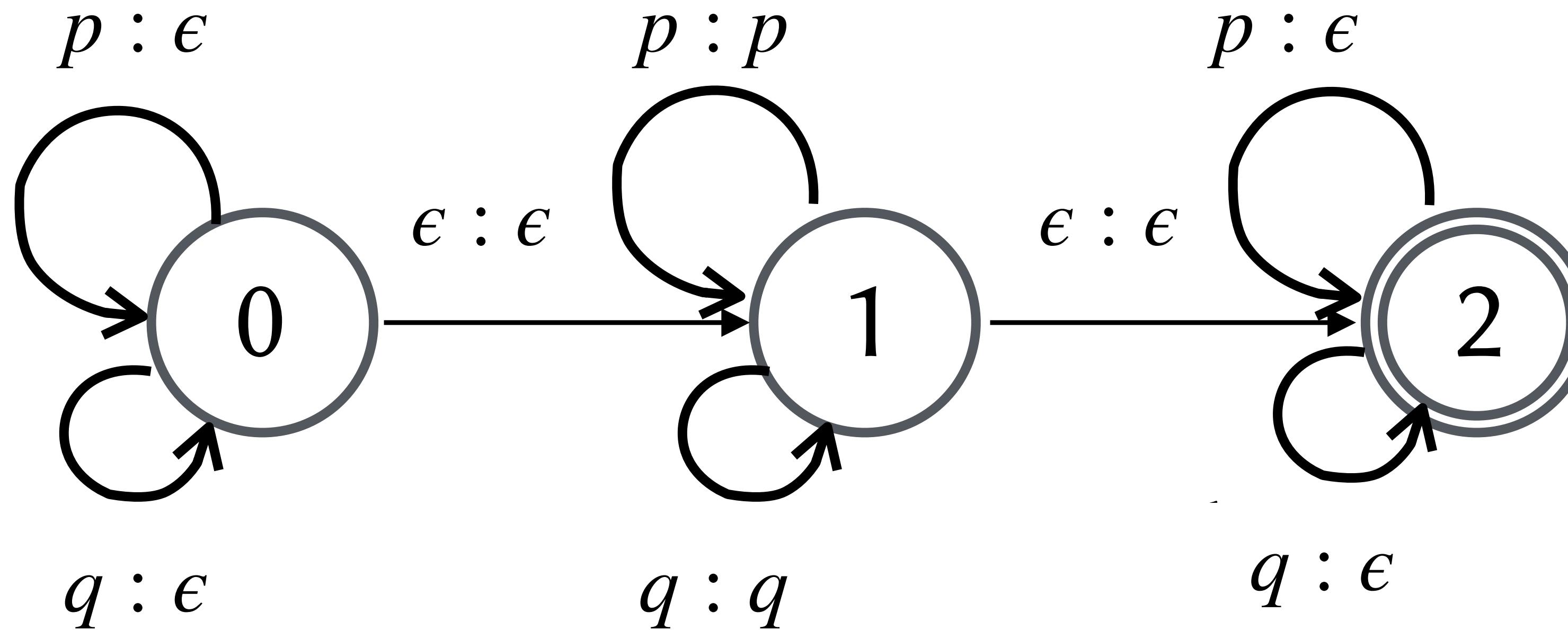
yelp \rightarrow y i ρ. bark \rightarrow w o o f | w o o f w o o f | ...

Special symbol, ϵ (epsilon) : allows to make a move without consuming an input symbol

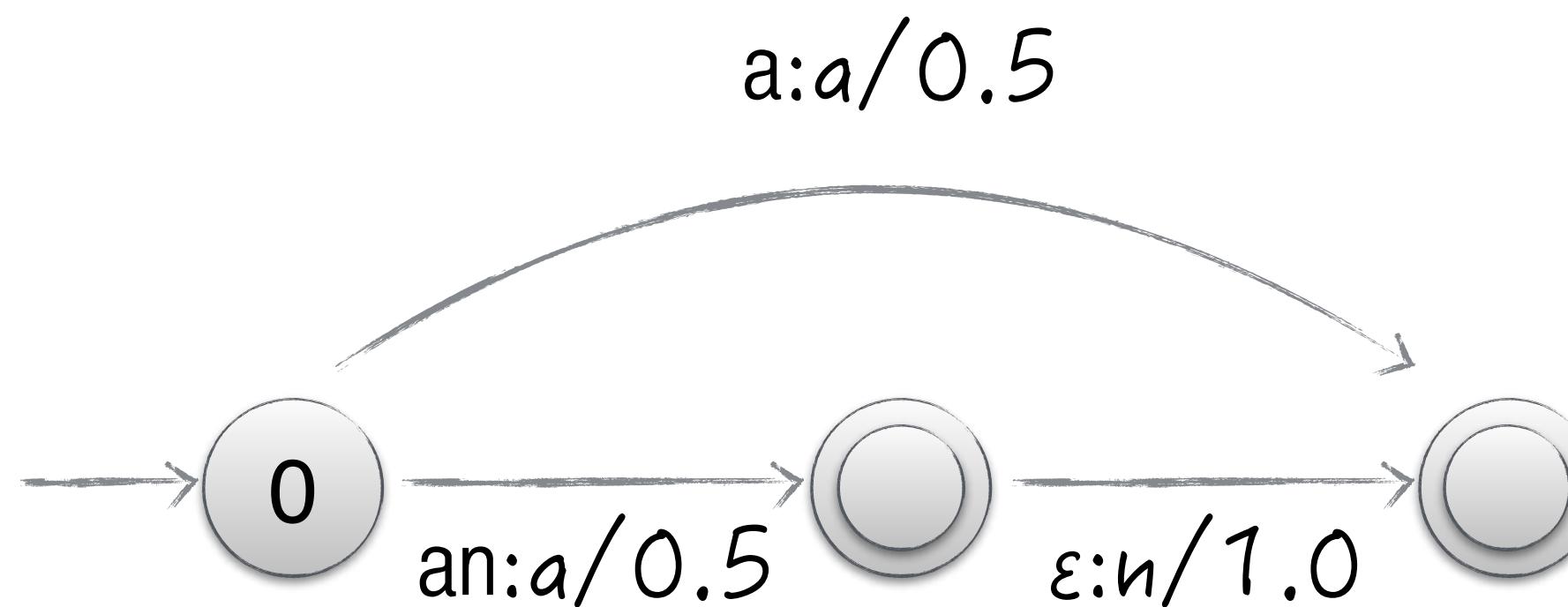
or without producing an output symbol

Problem

Design a finite-state transducer that can map a string in Σ^* to any of its substrings.
 $s' \in \Sigma^*$ is a substring of $S \in \Sigma^*$ if $S = xs'y$ for some $x, y \in \Sigma^*$. Let $\Sigma = \{p, q\}$.

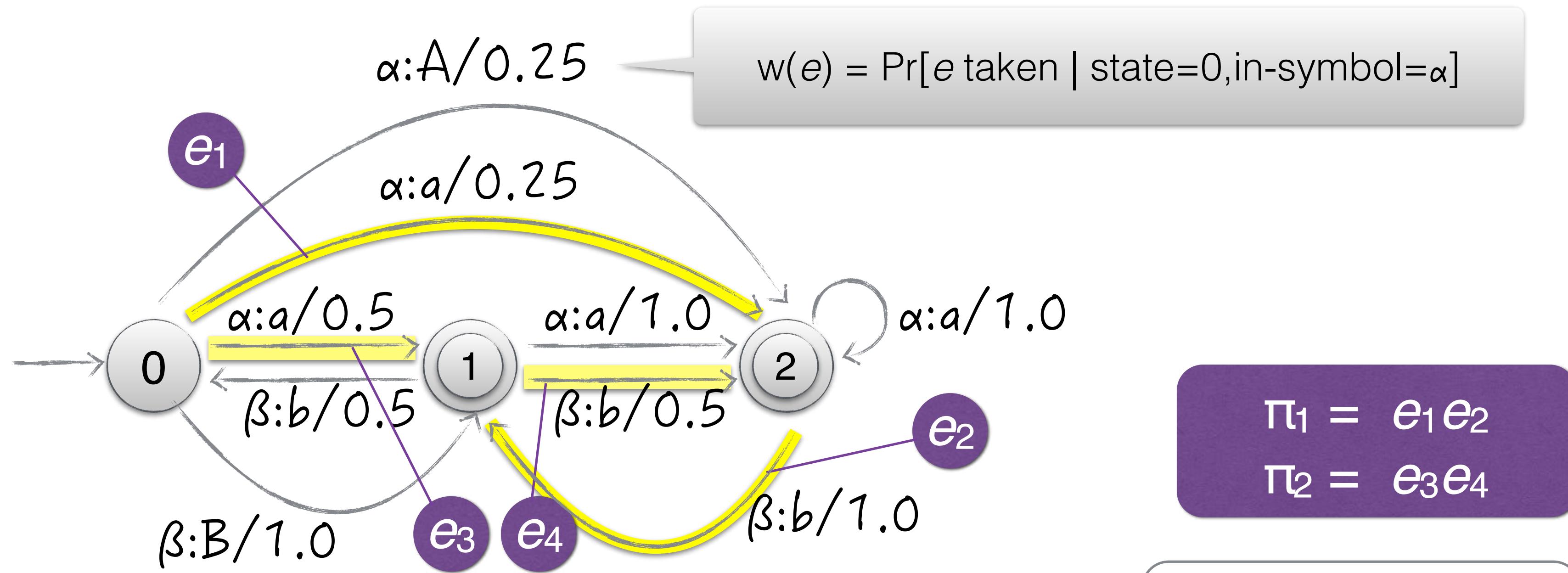


Weighted Path



- “Weights” can be probabilities, negative log-likelihoods, or any cost function representing the cost incurred in mapping an input sequence to an output sequence
- How are the weights accumulated along a path?

Weighted Path: Probabilistic FST



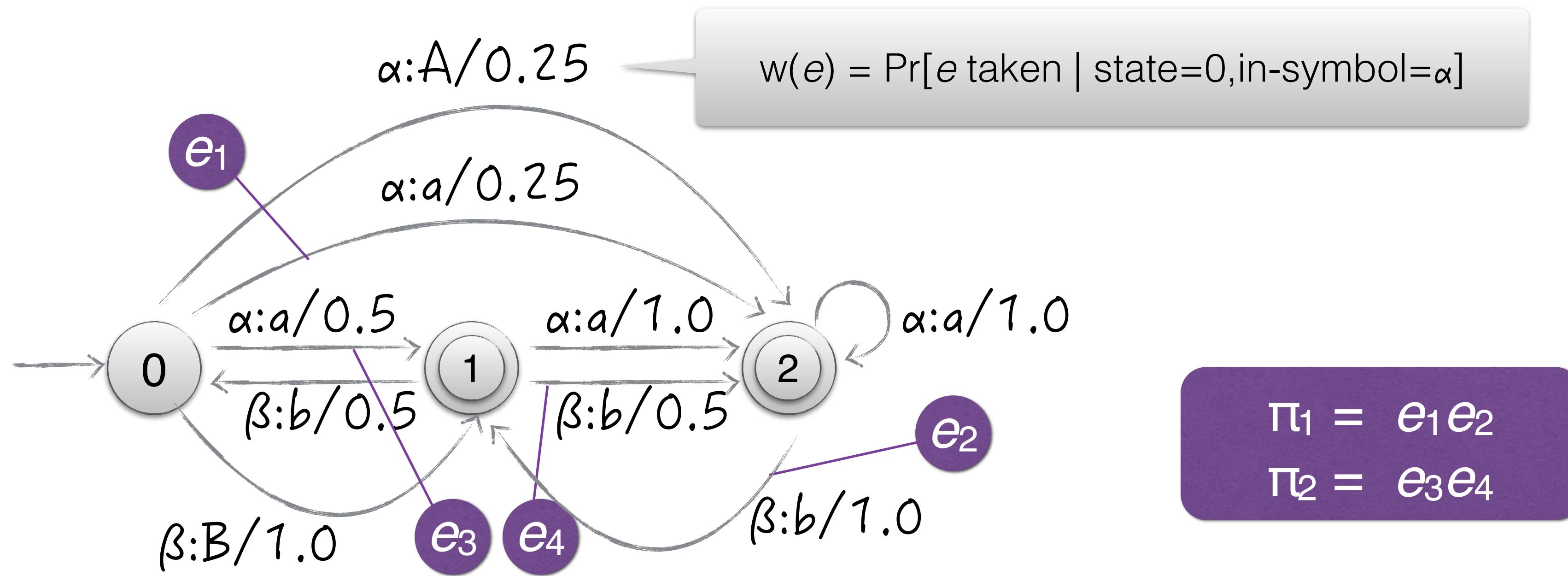
- $T(\alpha\beta, ab) = \Pr[\text{output}=ab, \text{accepts} \mid \text{input}=\alpha\beta, \text{start}]$
 $= \Pr[\Pi_1 \mid \text{input}=\alpha\beta, \text{start}] + \Pr[\Pi_2 \mid \text{input}=\alpha\beta, \text{start}]$

$$= 0.25 + 0.25 = 0.5$$

$$\begin{aligned}
 &= \Pr[e_1 \mid \text{input}=\alpha\beta, \text{start}] \times \Pr[e_2 \mid \text{input}=\alpha\beta, \text{start}, e_1] \\
 &= \Pr[e_1 \mid \text{state}=0, \text{in-symb}=\alpha] \times \Pr[e_2 \mid \text{state}=2, \text{in-symb}=\beta] \\
 &= w(e_1) \times w(e_2) = 0.25 \times 1.0 = 0.25
 \end{aligned}$$

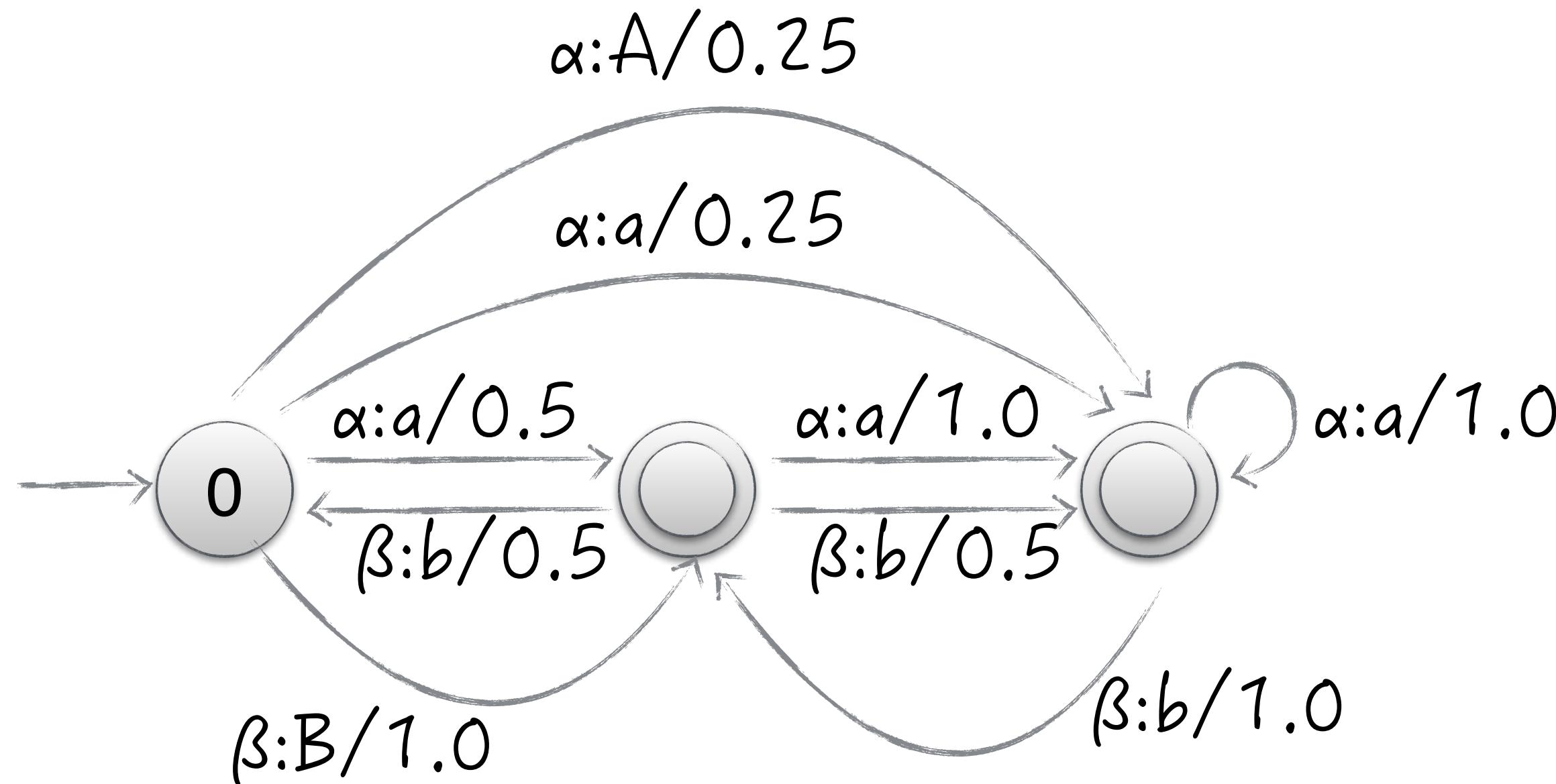
$$\begin{aligned}
 &= w(e_3) \times w(e_4) = 0.5 \times 0.5 = 0.25
 \end{aligned}$$

Weighted Path: Probabilistic FST



- $T(\alpha\beta, ab) = \Pr[\text{output}=ab, \text{accepts} \mid \text{input}=\alpha\beta, \text{start}]$
 $= \Pr[\pi_1 \mid \text{input}=\alpha\beta, \text{start}] + \Pr[\pi_2 \mid \text{input}=\alpha\beta, \text{start}]$
- More generally, $T(x, y) = \sum_{\pi \in P(x, y)} \prod_{e \in \pi} w(e)$
where $P(x, y)$ is the set of all accepting paths with input x and output y

Weighted Path



- But not all WFSTs are probabilistic FSTs
- Weight is often a “score” and maybe accumulated differently
- But helpful to retain some basic algebraic properties of weights: abstracted as **semirings**

Semirings

A semiring is a set of values associated with two operations \oplus and \otimes , along with their identity values $\bar{0}$ and $\bar{1}$

Weight assigned to an input/output pair

$$T(x,y) = \bigoplus_{\pi \in P(x,y)} \bigotimes_{e \in \pi} w(e)$$

where $P(x,y)$ is the set of all accepting paths with input x , output y

(generalizing the weight function for a probabilistic FST)

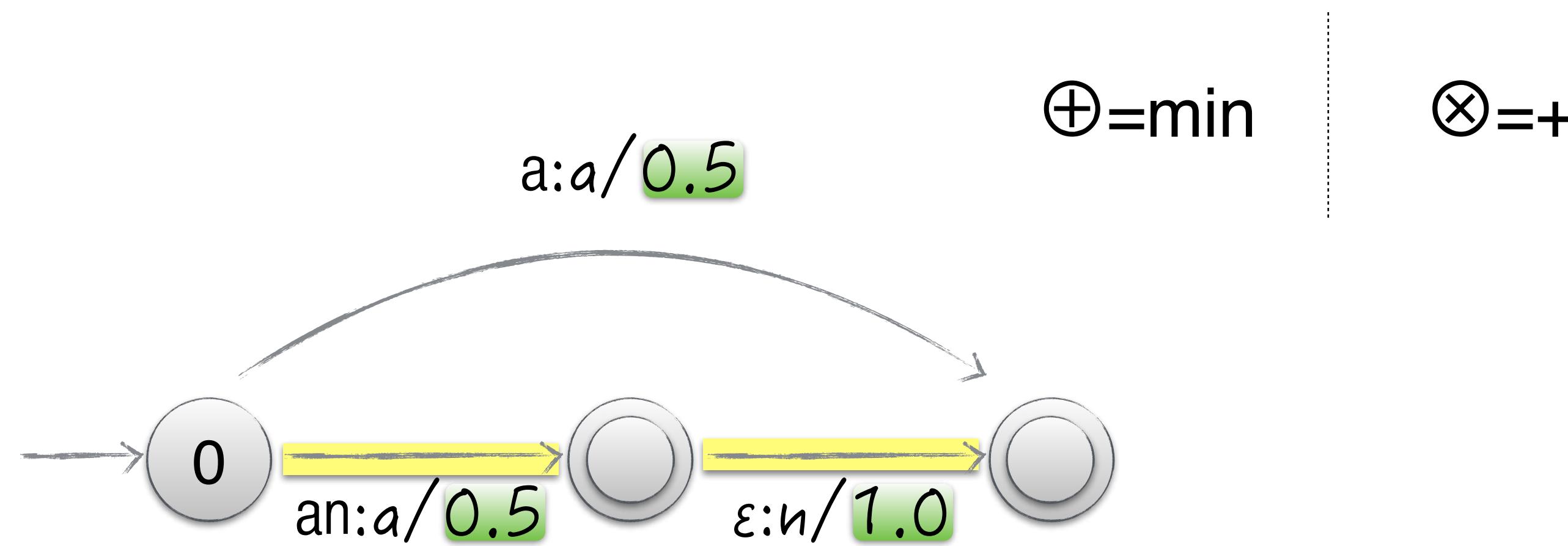
Semirings

Some popular semirings [M02]

SEmiring	SET	\oplus	\otimes	$\bar{0}$	$\bar{1}$	
Boolean	{F,T}	\vee	\wedge	F	T	Is there a path?
Real	\mathbb{R}_+	+	\times	0	1	Log Det Val
Log	$\mathbb{R} \cup \{-\infty, +\infty\}$	\oplus_{\log}	+	$+\infty$	0	
Tropical	$\mathbb{R} \cup \{-\infty, +\infty\}$	min	+	$+\infty$	0	“Viterbi Approx.” of $-\log \Pr[y x]$

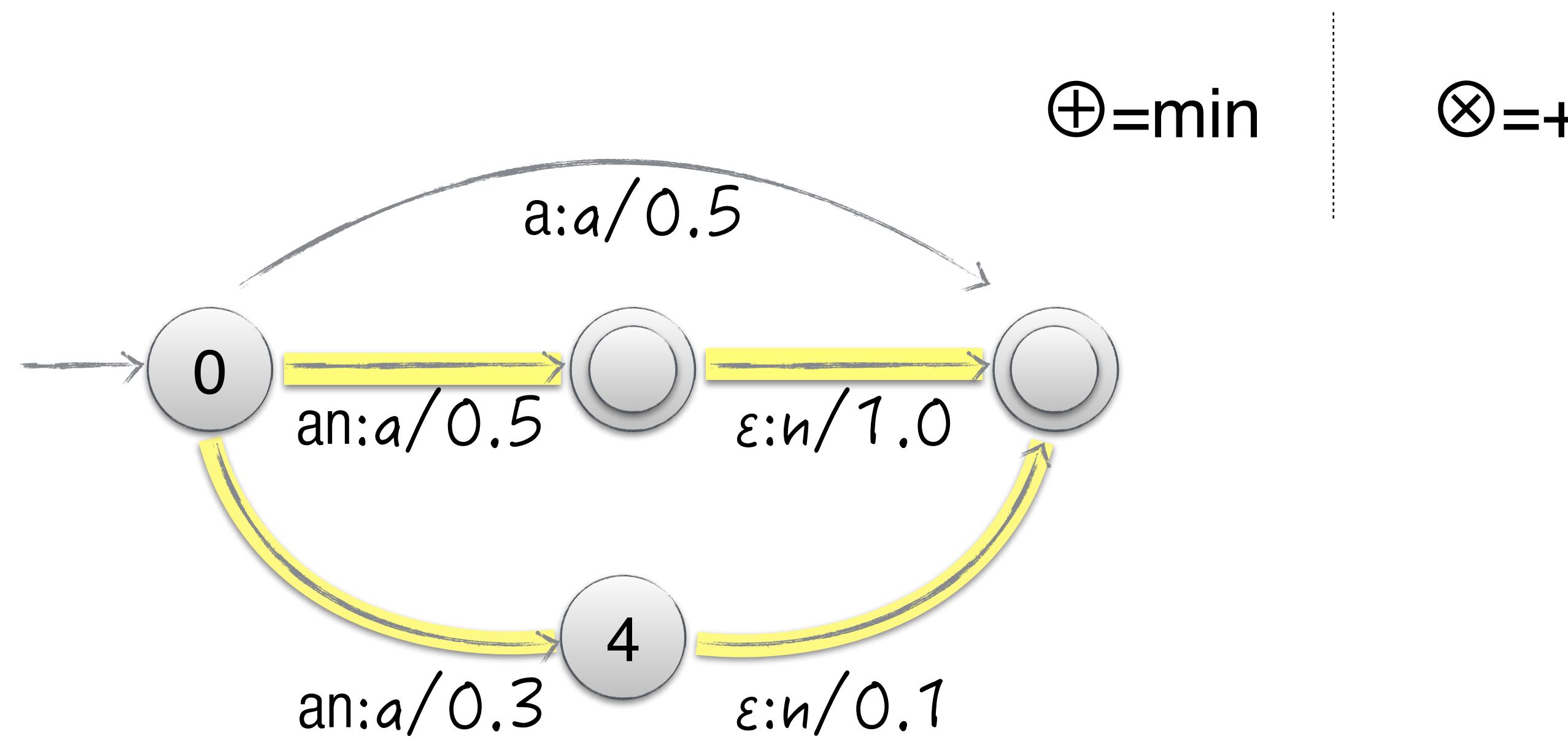
Operator \oplus_{\log} defined as: $x \oplus_{\log} y = -\log (e^{-x} + e^{-y})$

Weighted Path: Tropical Semiring



- Weight of a path π is the \otimes -product of all the transitions in π
 $w(\pi): (0.5 \otimes 1.0) = 0.5 + 1.0 = 1.5$
- Weight of a sequence " x,y " is the \oplus -sum of all paths labeled with " x,y "
 $w((an), (a \ n)) = (1.5 \oplus 0) = \min(1.5, \infty) = 1.5$

Weighted Path: Tropical Semiring

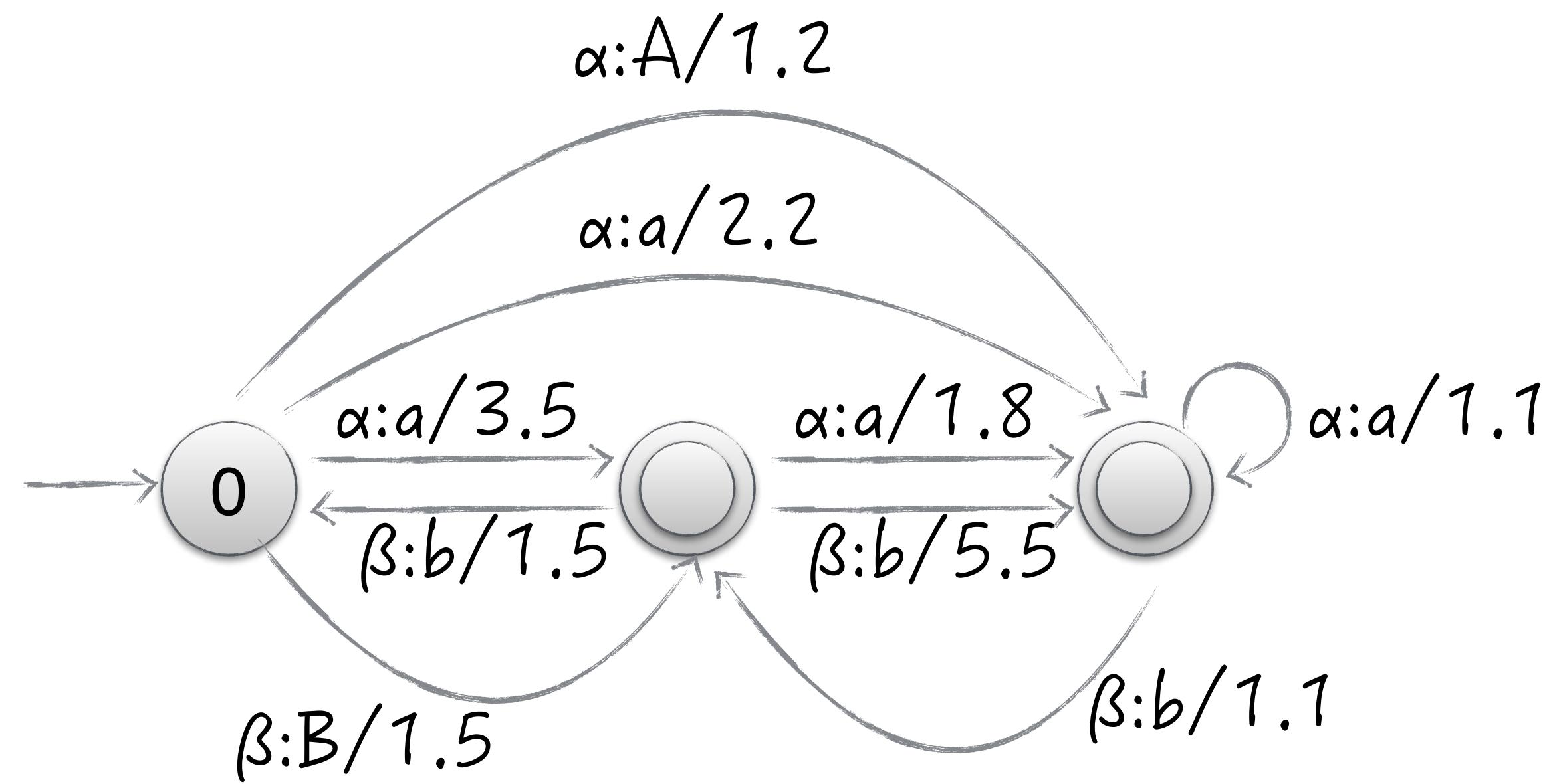


- Weight of a sequence " x,y " is the \oplus -sum of all paths labeled with " x,y "
 $w((an), (a \ n)) = ?$
Path 1: $(0.5 \otimes 1.0) = 1.5$
Path 2: $(0.3 \otimes 0.1) = 0.4$
Weight of " $((an), (a \ n))$ " = $(1.5 \oplus 0.4) = 0.4$

Shortest Path

- Recall $T(x,y) = \bigoplus_{\pi \in P(x,y)} w(\pi)$
where $P(x,y) = \text{set of paths with input/output } (x,y); w(\pi) = \bigotimes_{e \in \pi} w(e)$
- In the tropical semiring \bigoplus is min. $T(x,y)$ associated with a single path in $P(x,y)$: *Shortest Path*
 - Can be found using Dijkstra's algorithm : $\Theta(|E| + |Q| \cdot \log |Q|)$ time

Shortest Path

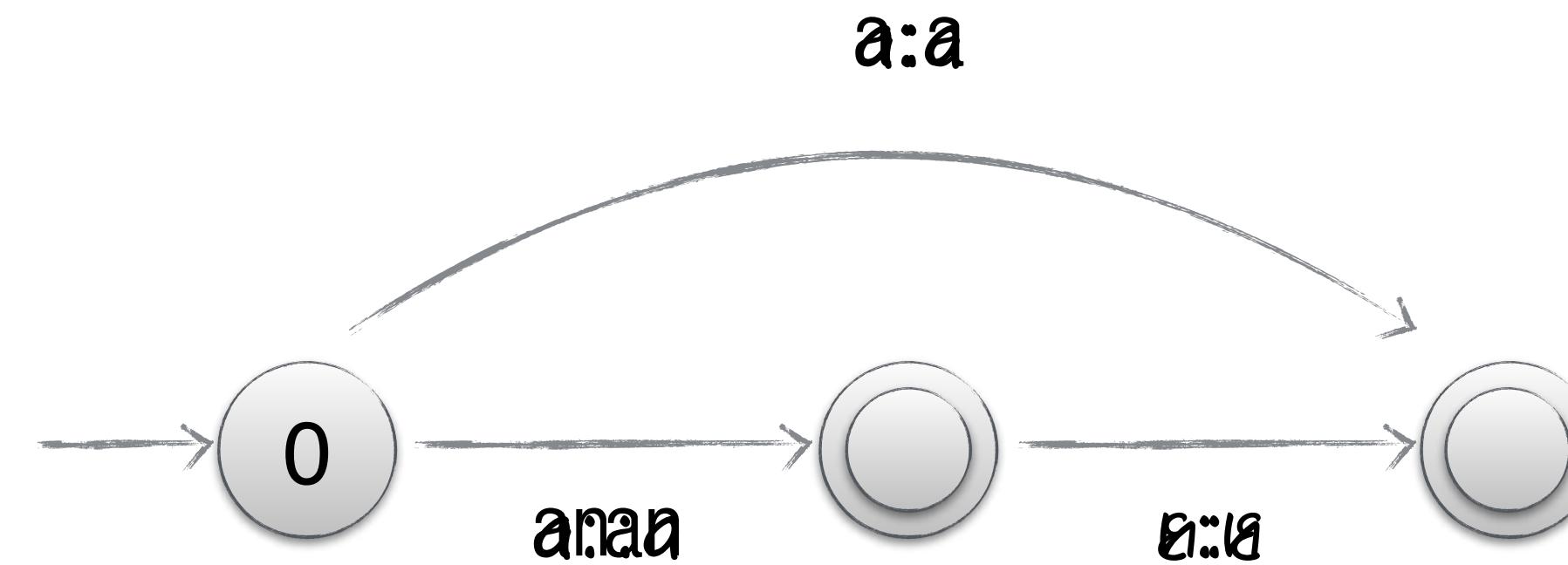


$$T(\alpha, a) = ?$$

$$T(\alpha\alpha, aa) = ?$$

Inversion

Swap the input and output labels in each transition

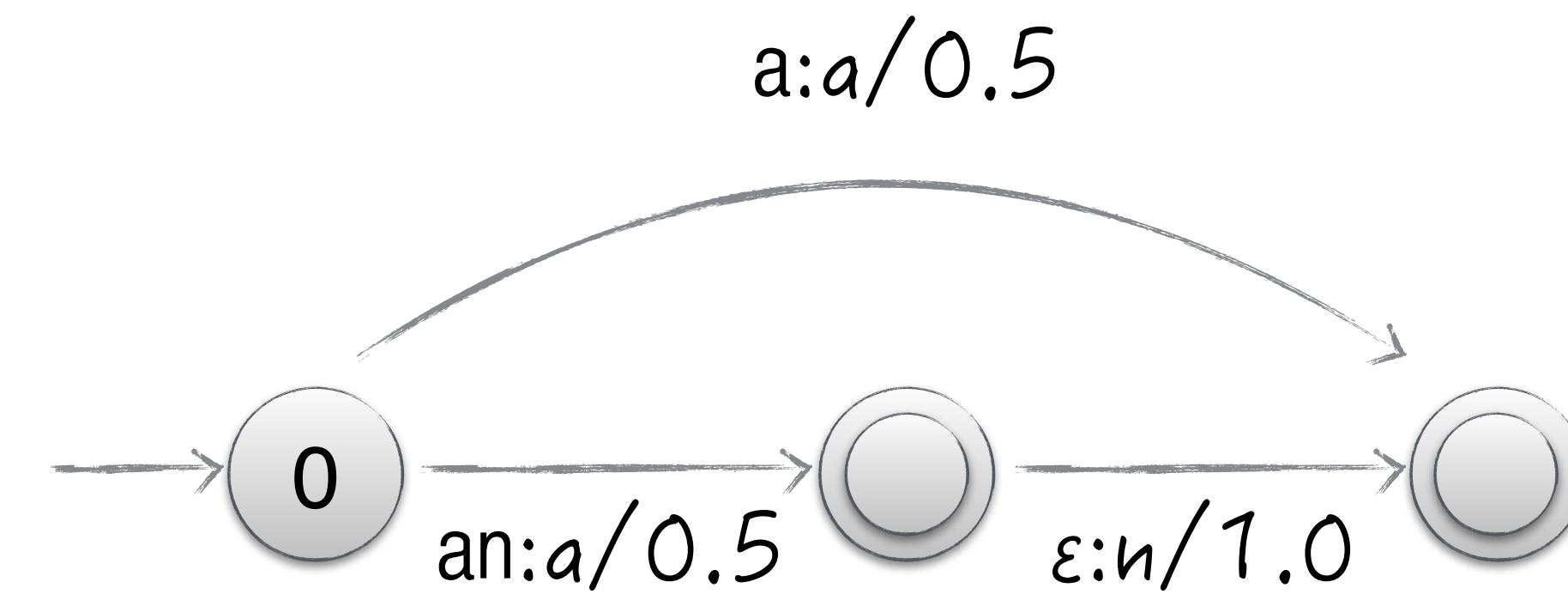


Weights (if they exist) are retained on the arcs

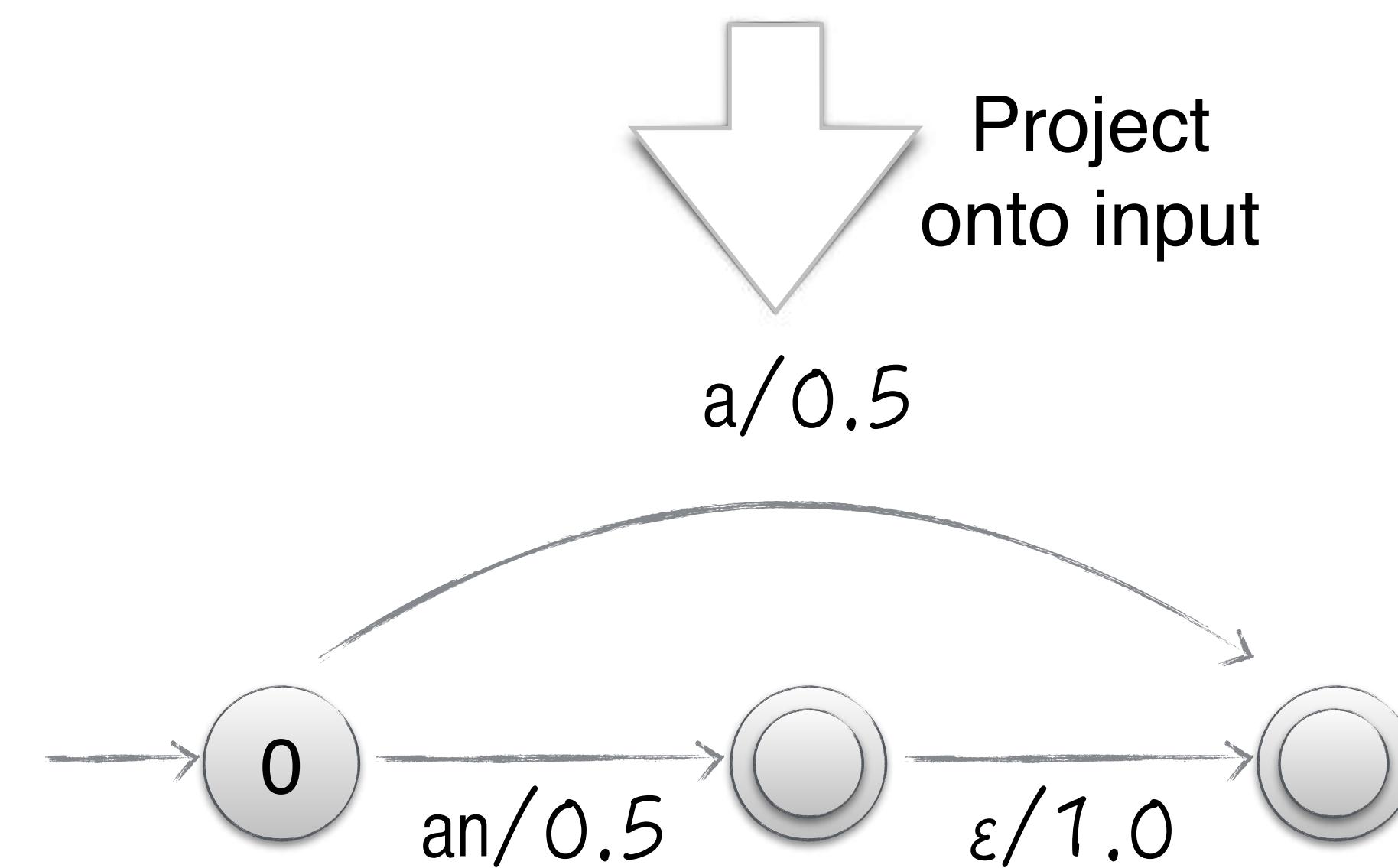
This operation comes in handy, especially during composition!

Projection

Project onto the input or output alphabet



Project
onto input



Basic FST Operations (Rational Operations)

The set of weighted transducers are closed under the following operations [Mohri '02]:

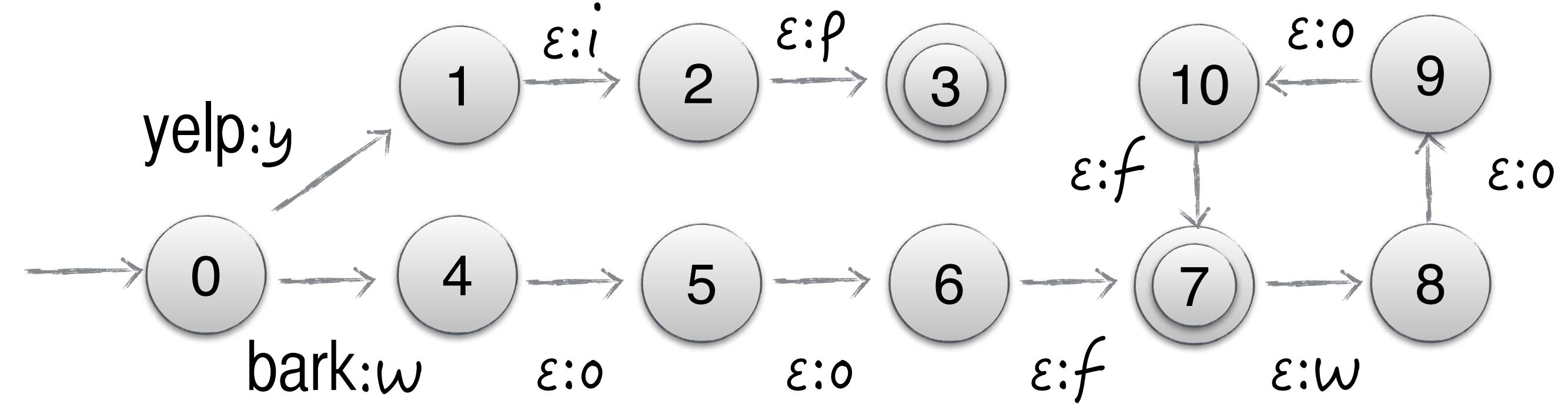
1. Sum or **Union**: $(T_1 \oplus T_2)(x, y) = T_1(x, y) \oplus T_2(x, y)$
2. Product or **Concatenation**: $(T_1 \otimes T_2)(x, y) = \bigoplus_{\substack{x=x_1x_2 \\ y=y_1y_2}} T_1(x_1, y_1) \otimes T_2(x_2, y_2)$
3. Kleene-closure: $T^*(x, y) = \bigoplus_{n=0}^{\infty} T^n(x, y)$

Basic FST Operations (Rational Operations)

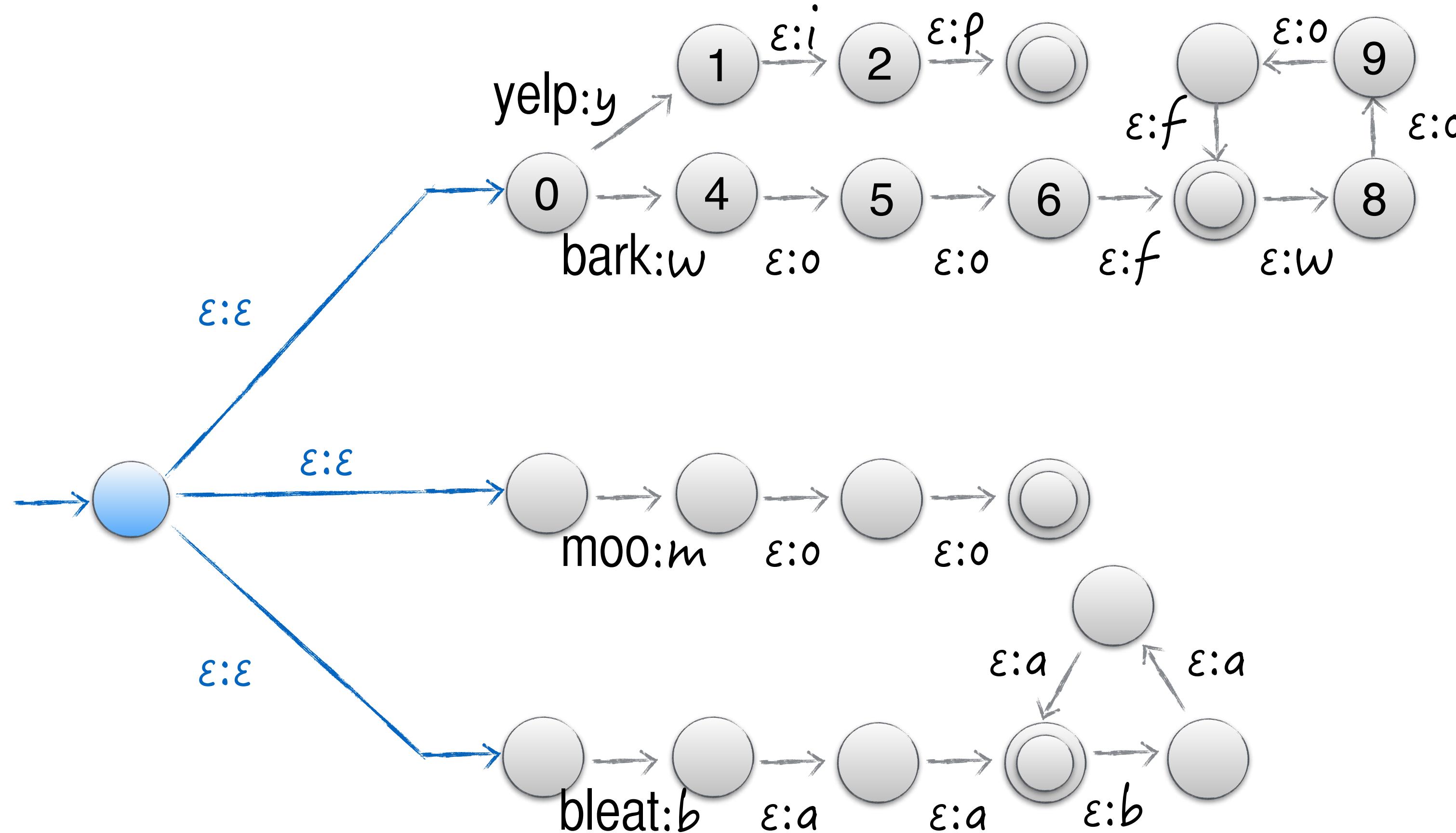
The set of weighted transducers are closed under the following operations [Mohri '02]:

1. Sum or **Union**: $(T_1 \oplus T_2)(x, y) = T_1(x, y) \oplus T_2(x, y)$
2. Product or **Concatenation**: $(T_1 \otimes T_2)(x, y) = \bigoplus_{\substack{x=x_1x_2 \\ y=y_1y_2}} T_1(x_1, y_1) \otimes T_2(x_2, y_2)$
3. Kleene-closure: $T^*(x, y) = \bigoplus_{n=0}^{\infty} T^n(x, y)$

Example: Recall Barking Dog



Example: Union



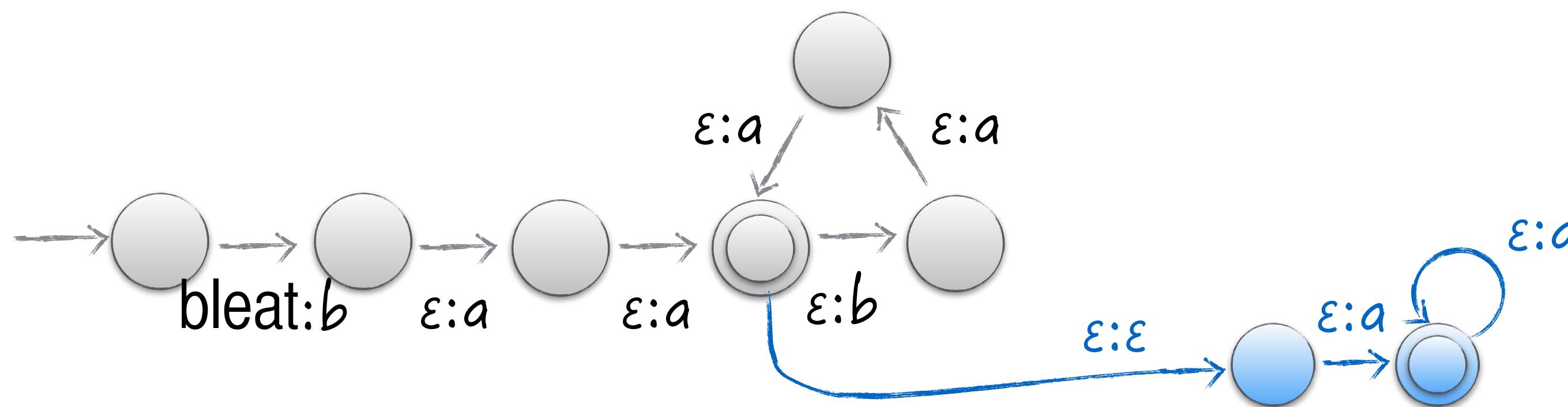
Animal farm

Basic FST Operations (Rational Operations)

The set of weighted transducers are closed under the following operations [Mohri '02]:

1. Sum or **Union**: $(T_1 \oplus T_2)(x, y) = T_1(x, y) \oplus T_2(x, y)$
2. Product or **Concatenation**: $(T_1 \otimes T_2)(x, y) = \bigoplus_{\substack{x=x_1x_2 \\ y=y_1y_2}} T_1(x_1, y_1) \otimes T_2(x_2, y_2)$
3. Kleene-closure: $T^*(x, y) = \bigoplus_{n=0}^{\infty} T^n(x, y)$

Example: Concatenation



Suppose the last “baa” in a bleat should be followed by one or more a's

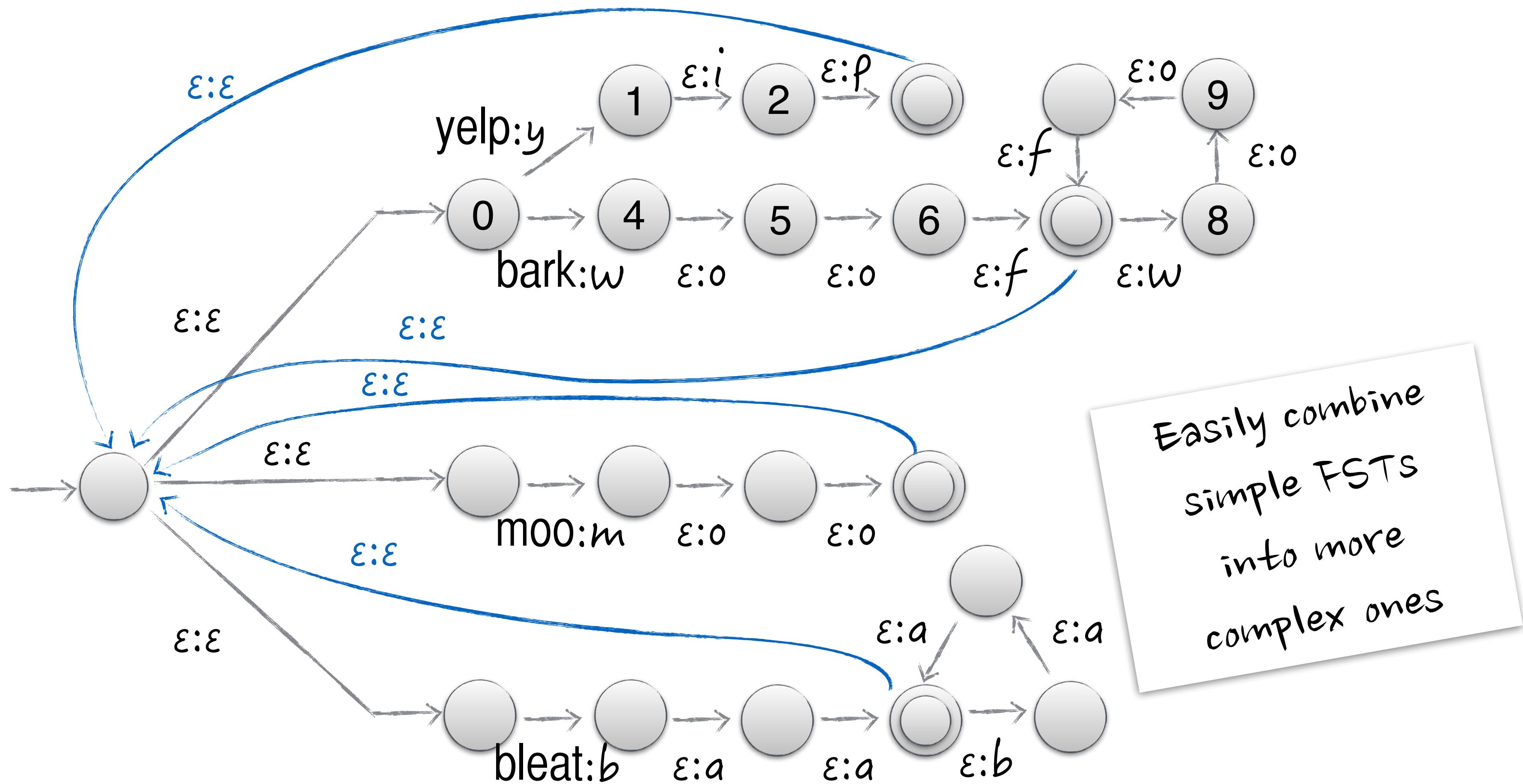
(e.g., “baabaa” is not OK, but “baaa” and “baabaaaaa” are)

Basic FST Operations (Rational Operations)

The set of weighted transducers are closed under the following operations [Mohri '02]:

1. Sum or **Union**: $(T_1 \oplus T_2)(x, y) = T_1(x, y) \oplus T_2(x, y)$
2. Product or **Concatenation**: $(T_1 \otimes T_2)(x, y) = \bigoplus_{\substack{x=x_1x_2 \\ y=y_1y_2}} T_1(x_1, y_1) \otimes T_2(x_2, y_2)$
3. Kleene-**closure**: $T^*(x, y) = \bigoplus_{n=0}^{\infty} T^n(x, y)$

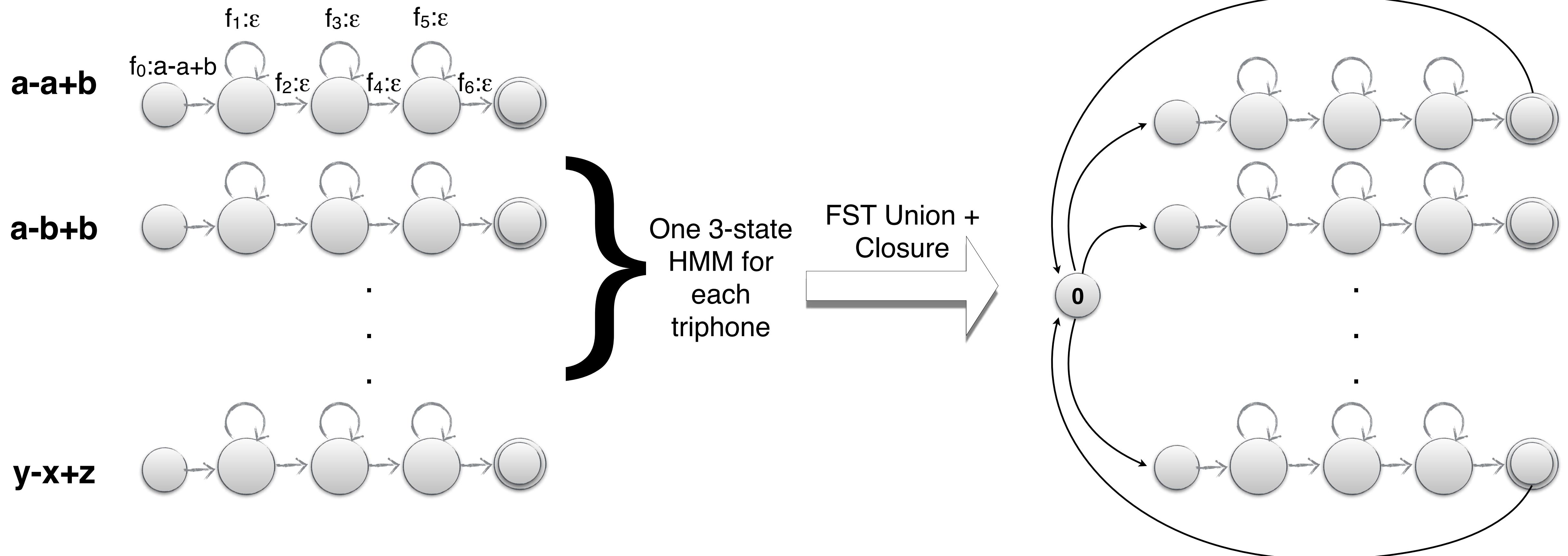
Example: Closure



Animal farm: allow arbitrarily long sequence of sounds!

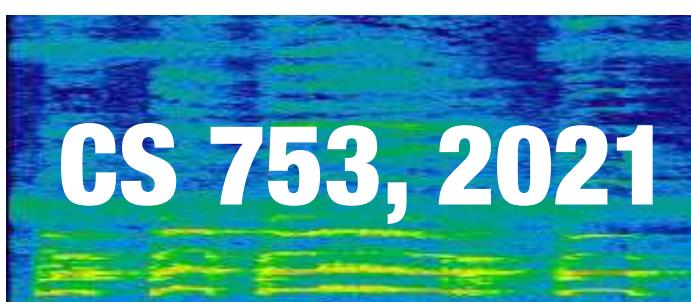
bark moo yelp bleat → woof woof moo yip baa baa

Acoustic Model WFST



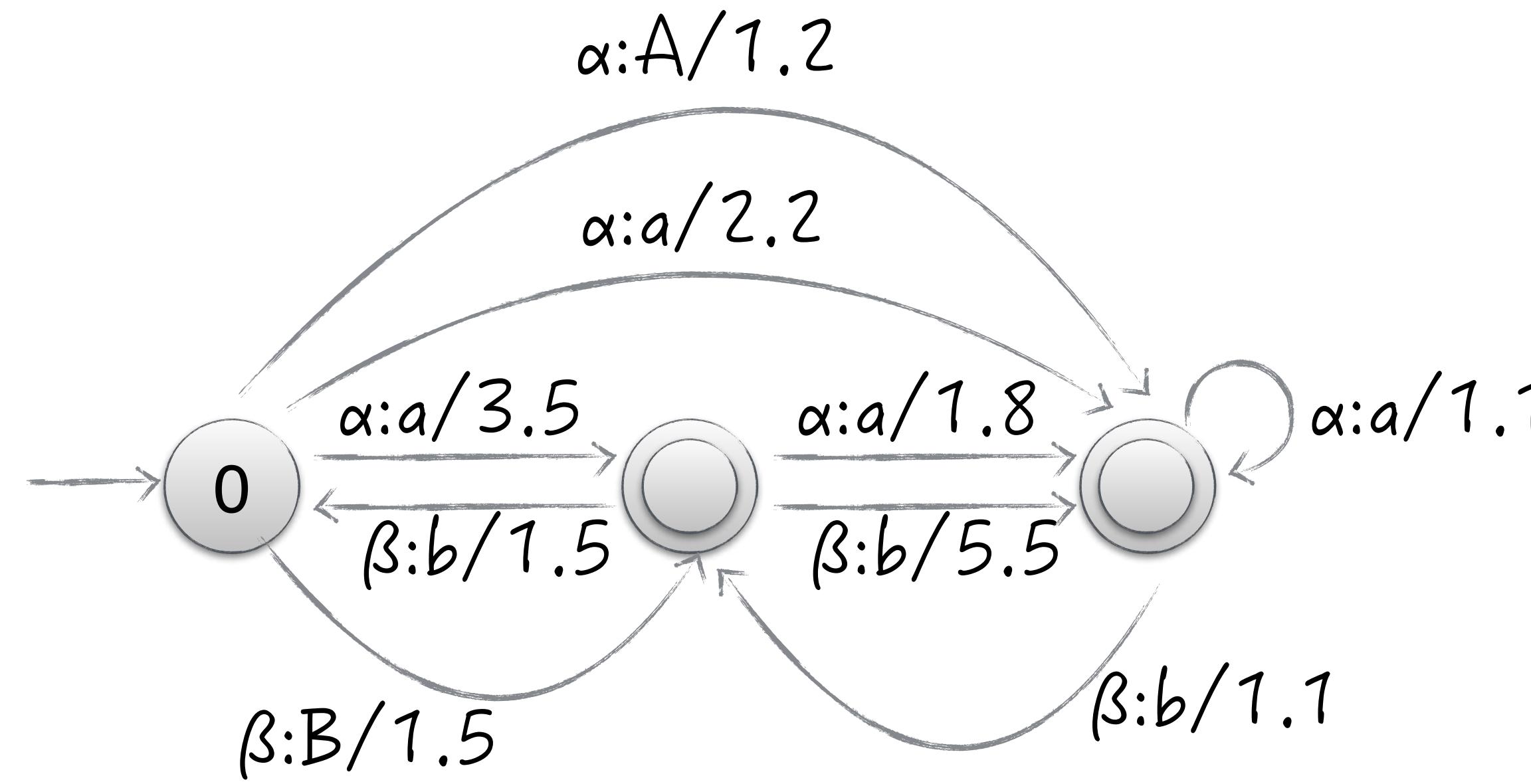
WFST Algorithms

Lecture 3a



Instructor: Preethi Jyothi, IITB

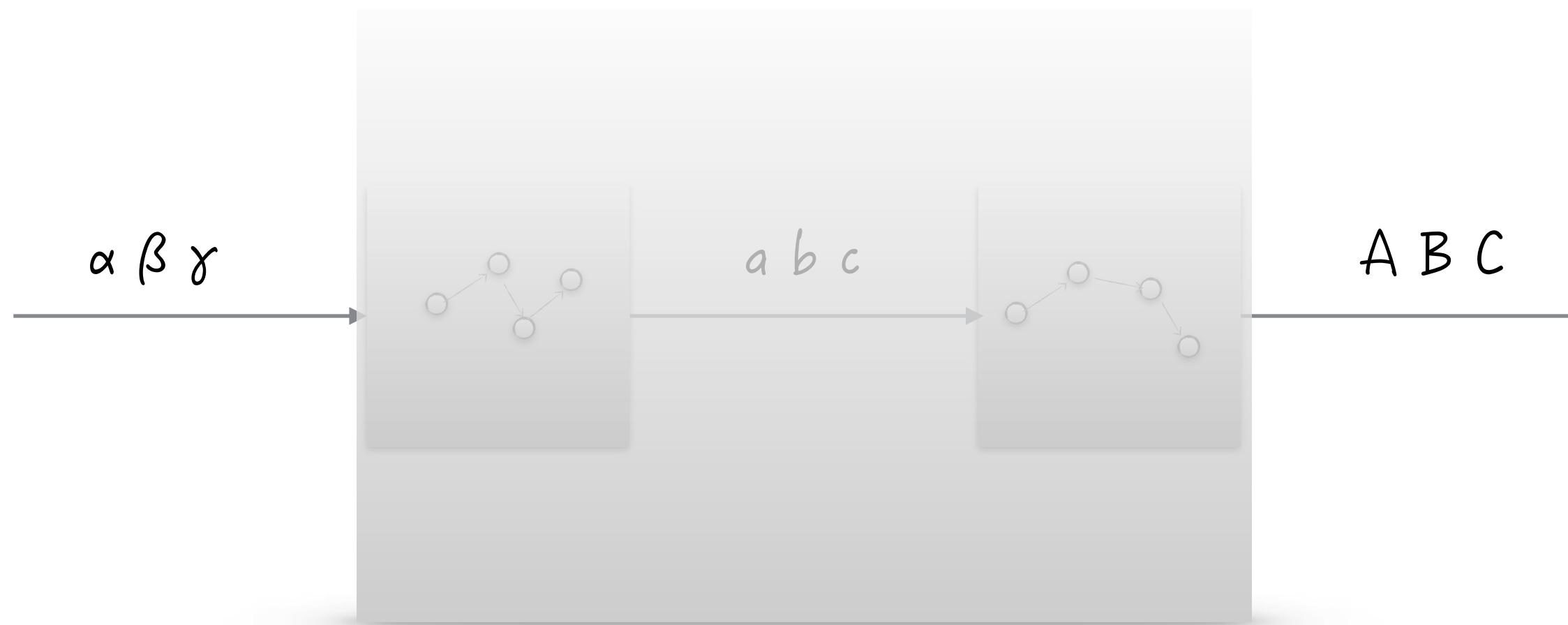
Recall weighted paths in WFSTs



Recall: Path weight with x as input and y as output is $T(x,y) = \bigoplus_{\pi \in P(x,y)} w(\pi)$
where $P(x,y) = \text{set of paths with input/output } (x,y)$; $w(\pi) = \bigotimes_{e \in \pi} w(e)$
 \oplus and \otimes are two operations associated with the semiring on the weights

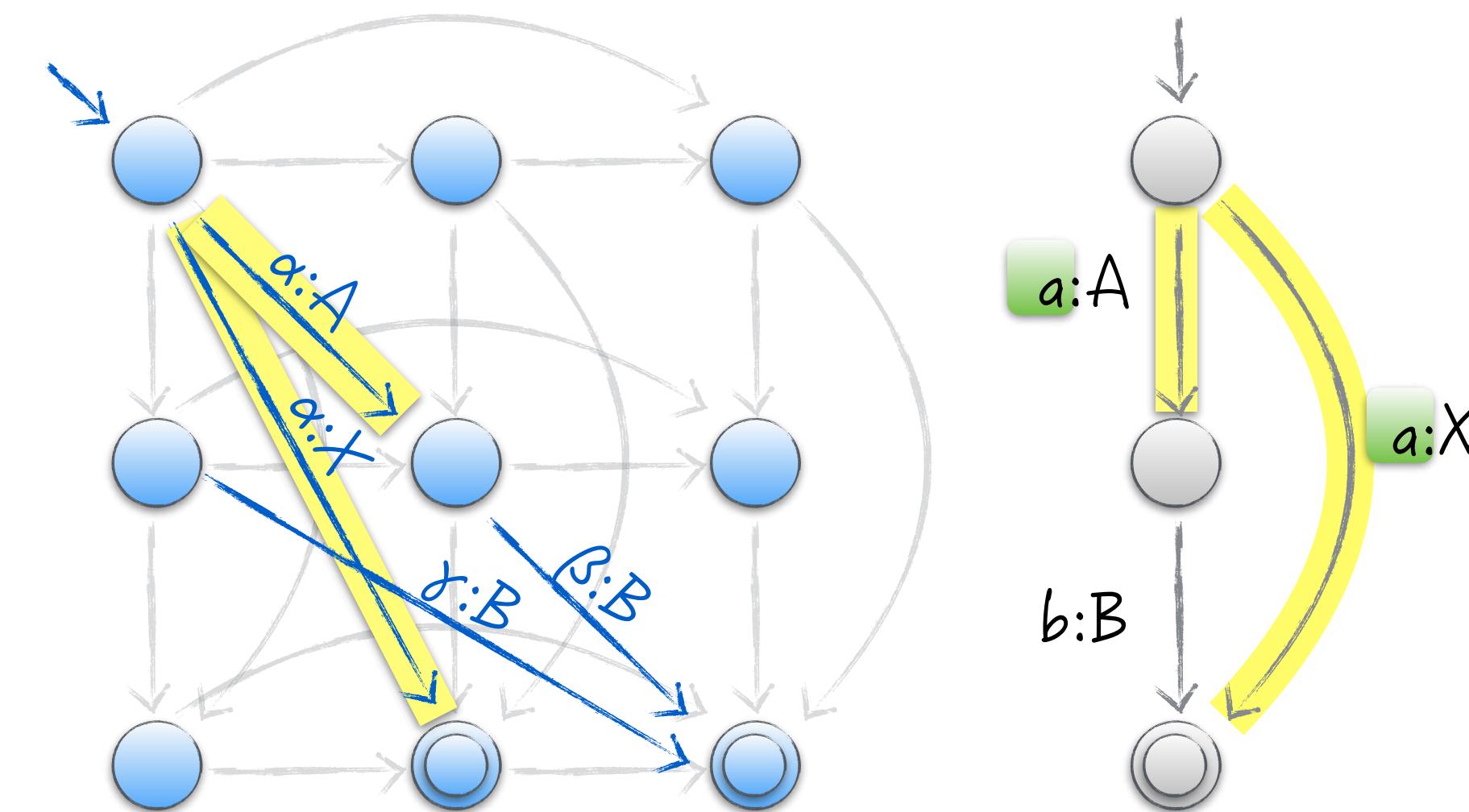
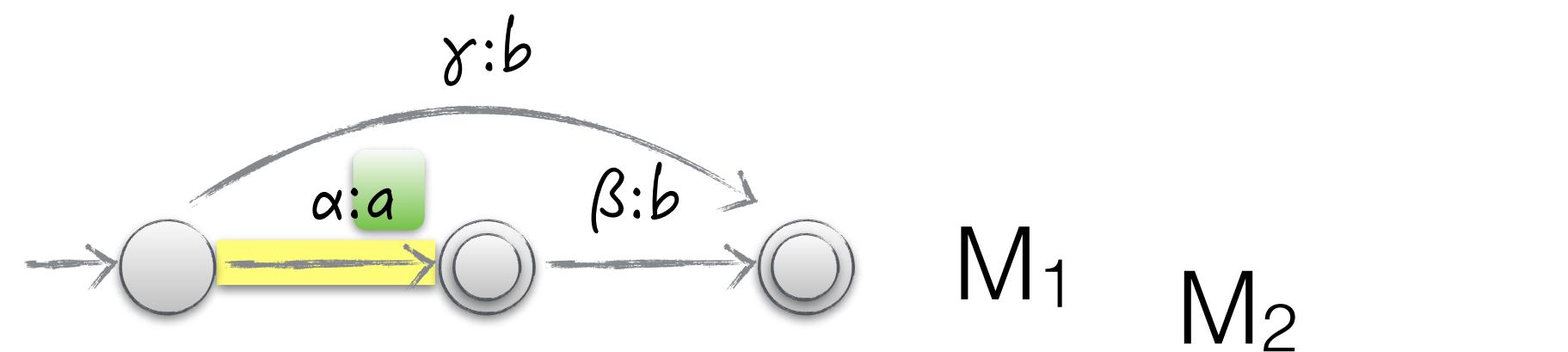
- In the tropical semiring \oplus is *min*.
- $T(x,y)$ associated with a single path in $P(x,y)$: *Shortest Path*

Composition of WFSTs



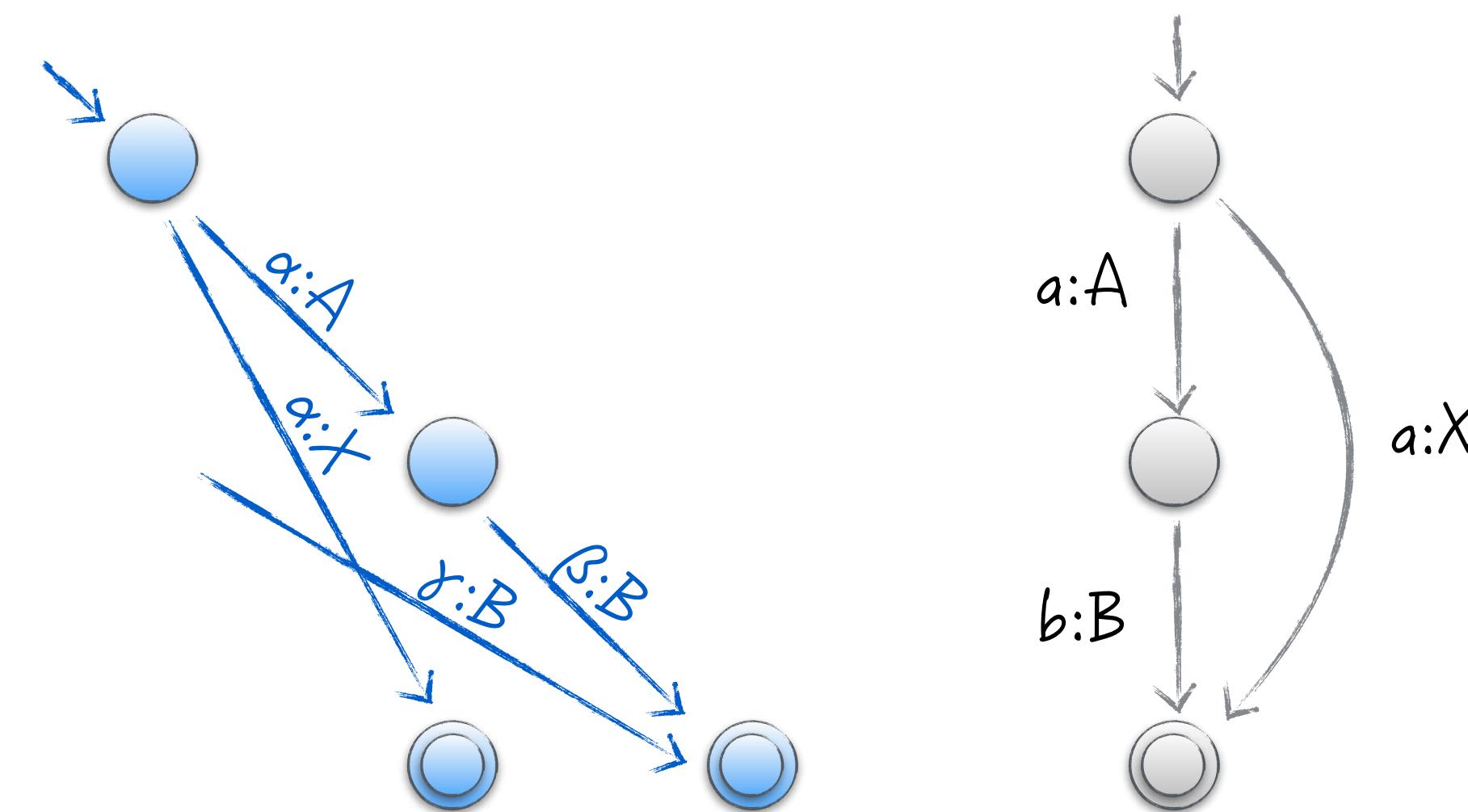
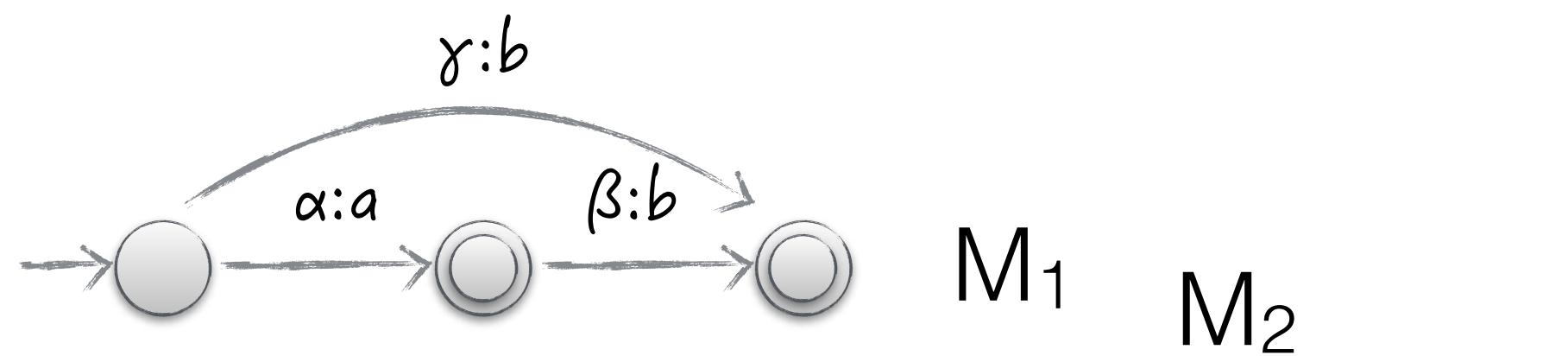
- If T_1 transduces x to z , and T_2 transduces z to y , then $T_1 \circ T_2$ transduces x to y
- $(T_1 \circ T_2)(x, y) = \bigoplus_z T_1(x, z) \otimes T_2(z, y)$

Composition: Construction



$M_1 \circ M_2$

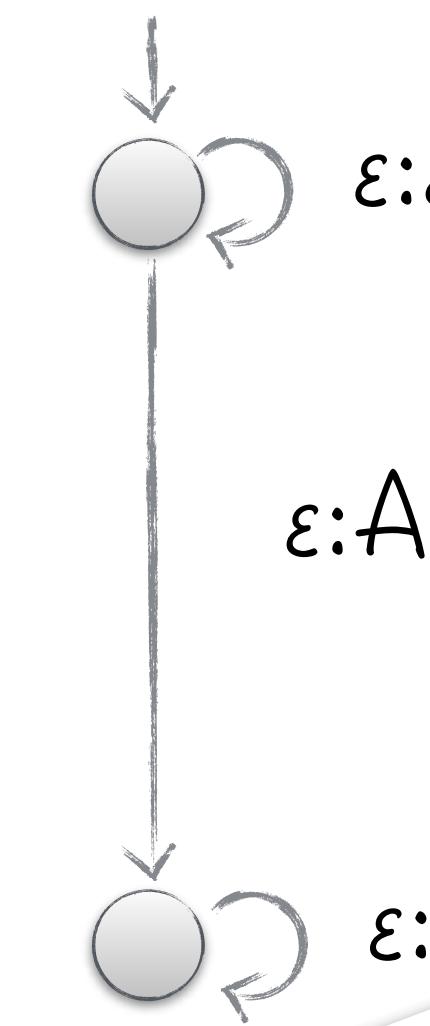
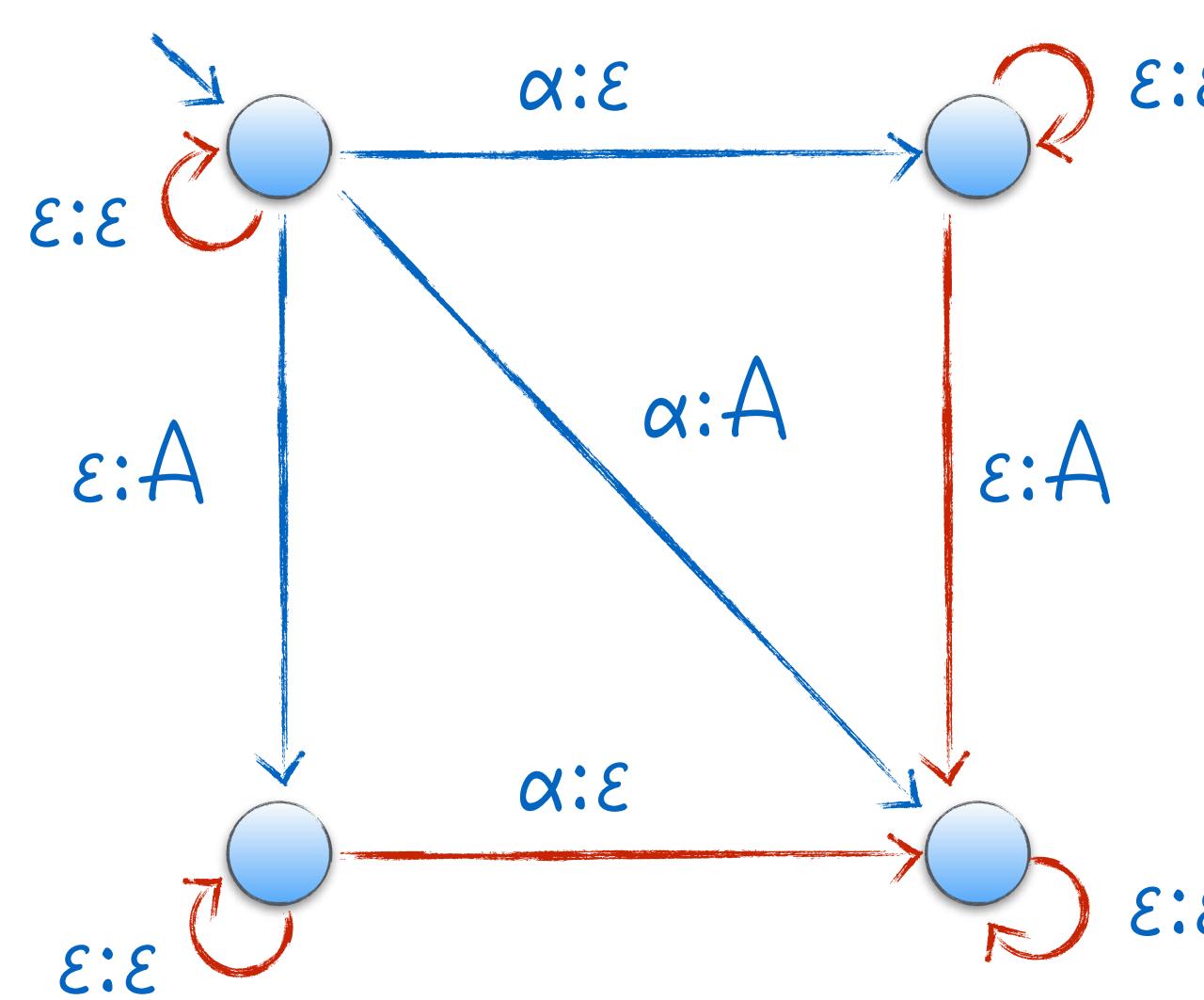
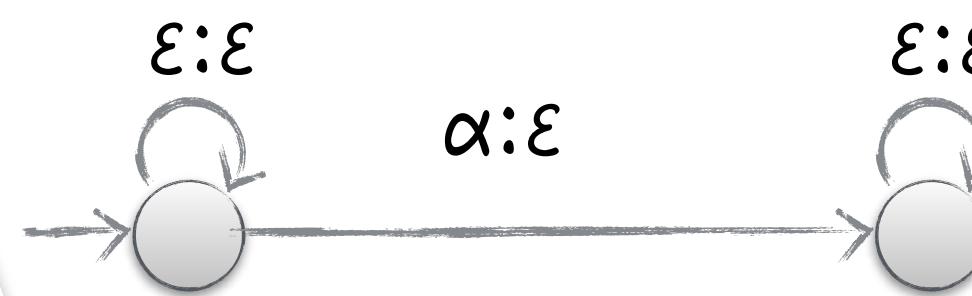
Composition



$M_1 \circ M_2$

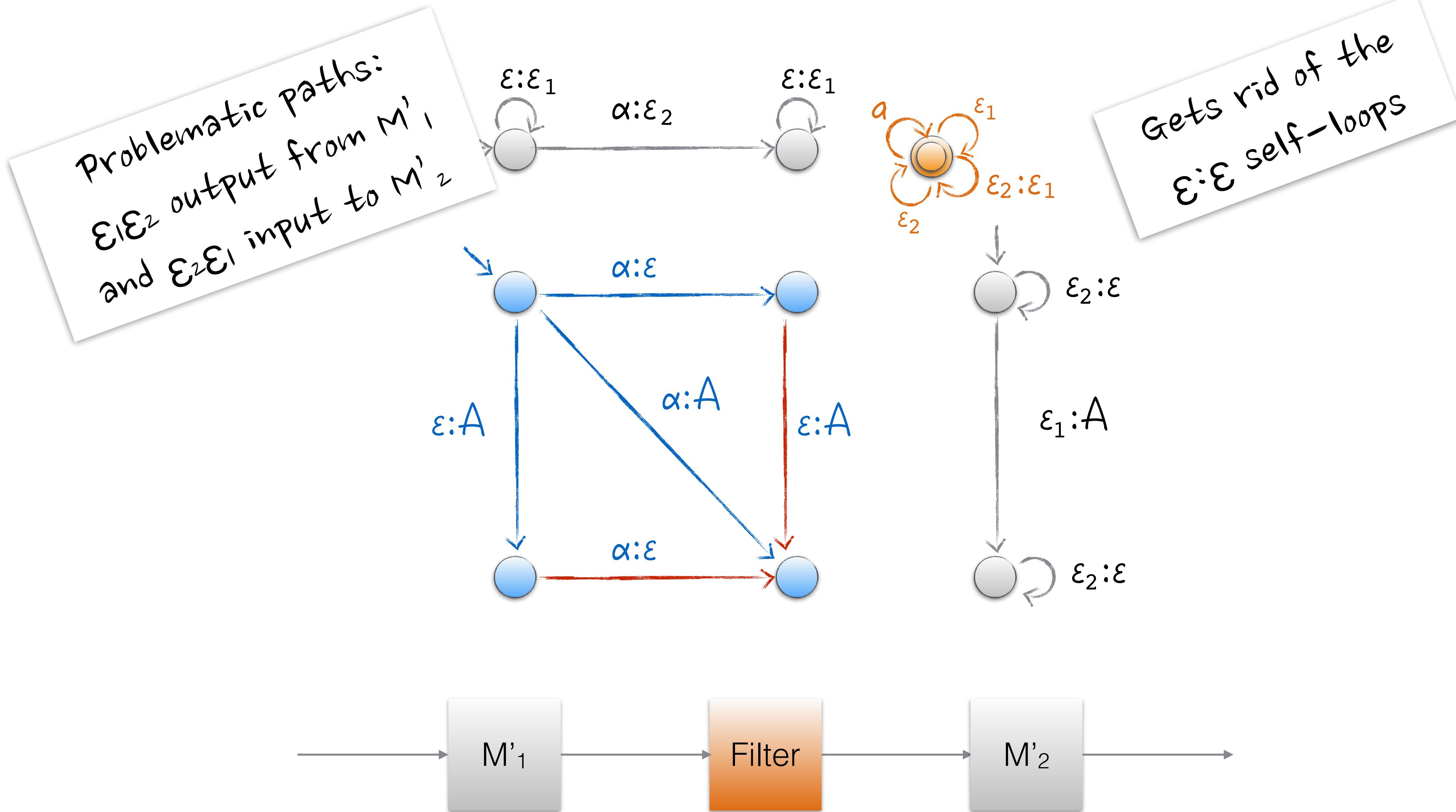
Composition: Handling epsilons

Implicit
 $\epsilon:\epsilon$ self-loops
on every state



Duplicate paths!
Weights can be wrong
(in certain semirings)

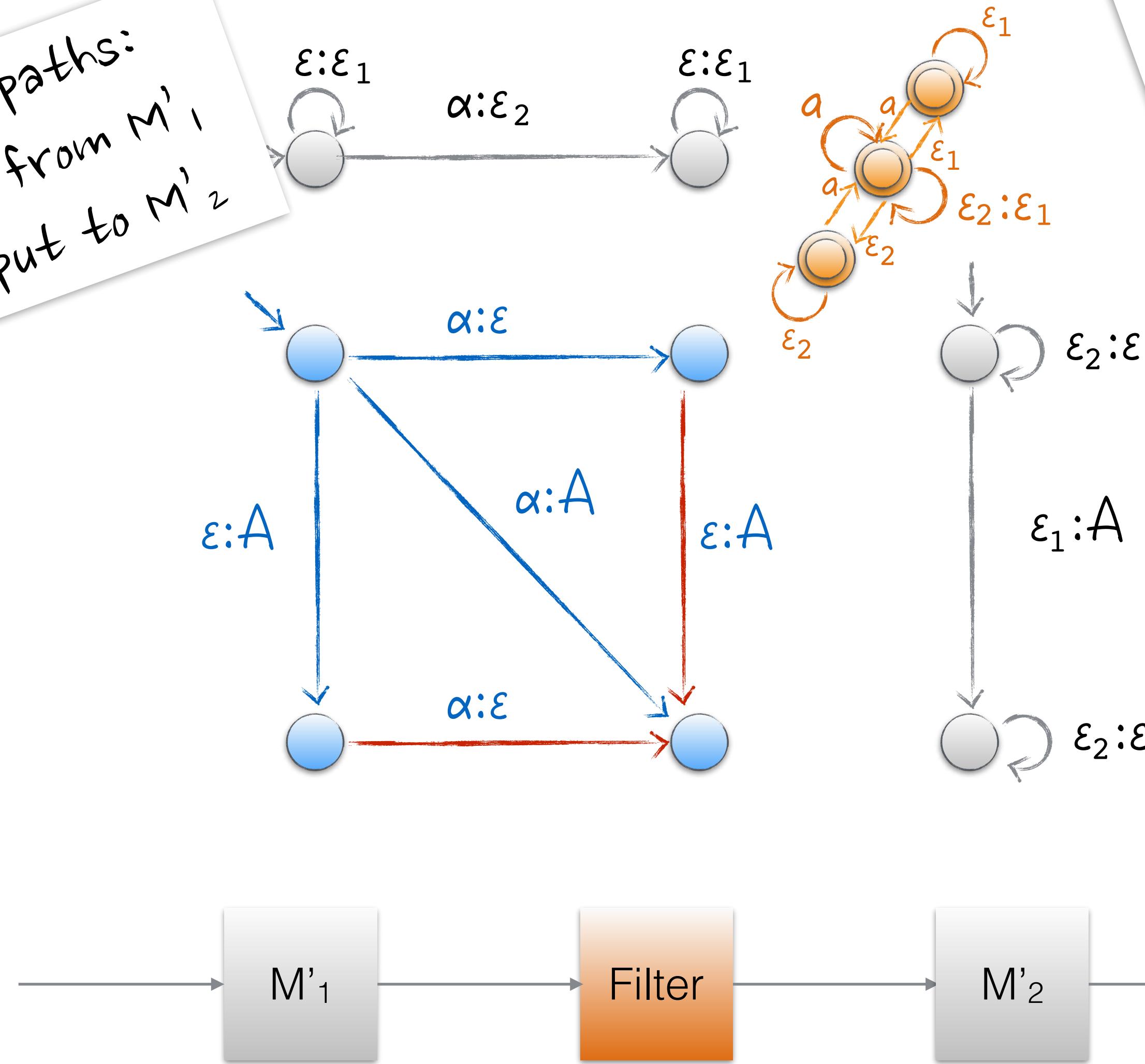
Composition: Filters



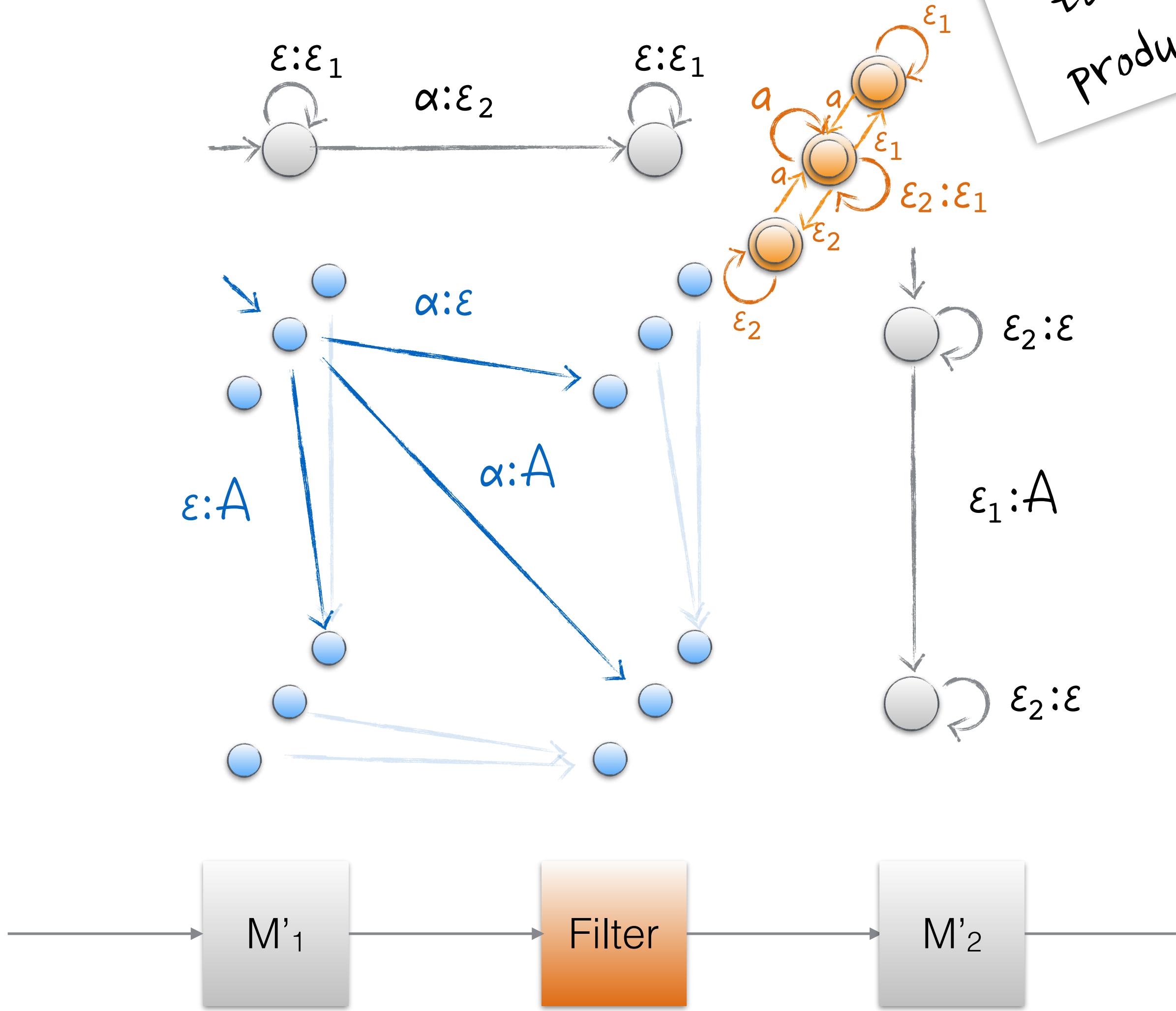
Composition: Filters

Problematic paths:
 $\varepsilon_1\varepsilon_2$ output from M'_1
 and $\varepsilon_2\varepsilon_1$ input to M'_2

Filter that doesn't
 take input $\varepsilon_1\varepsilon_2$ or
 produce output $\varepsilon_2\varepsilon_1$

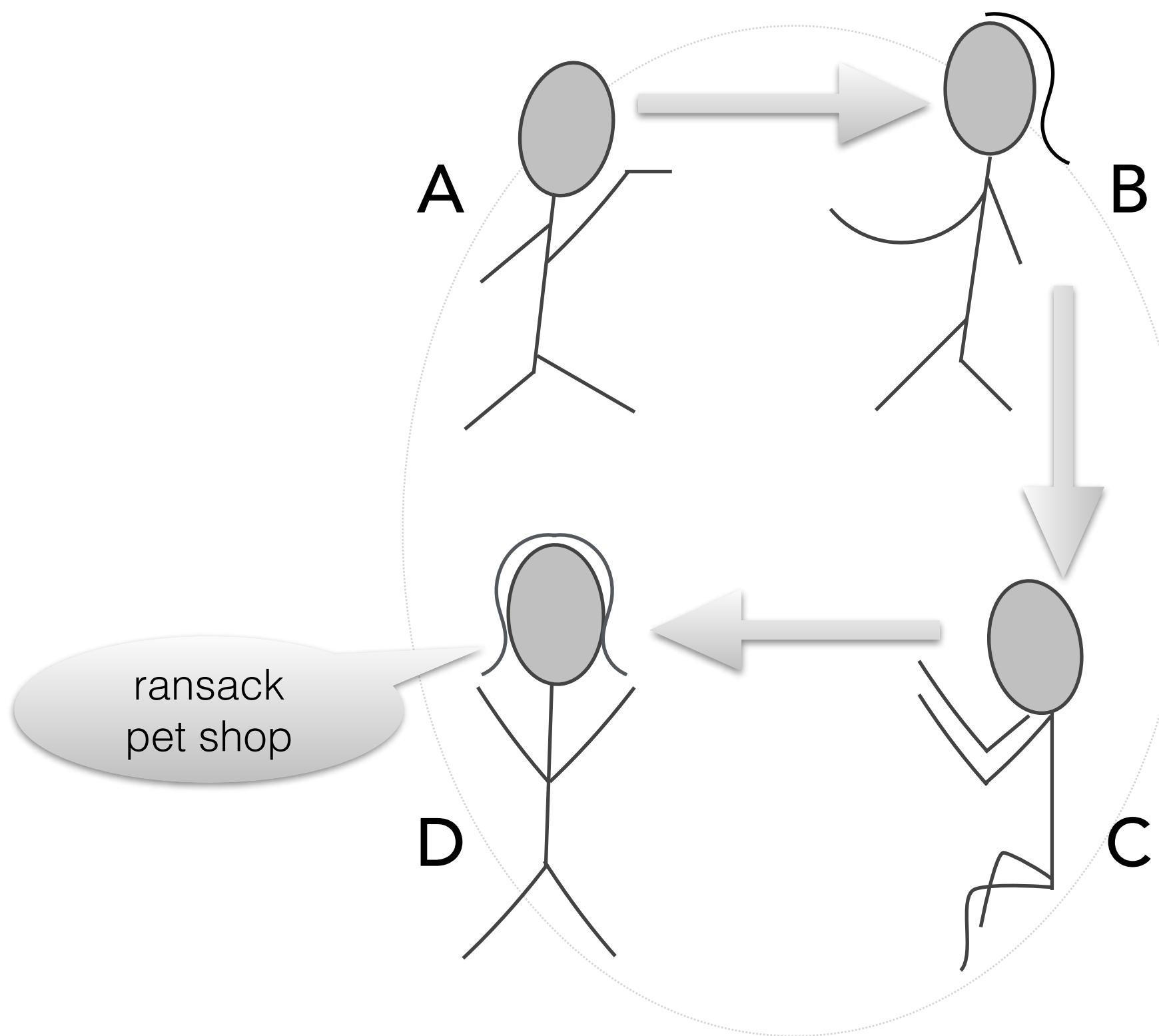


Composition: Filters



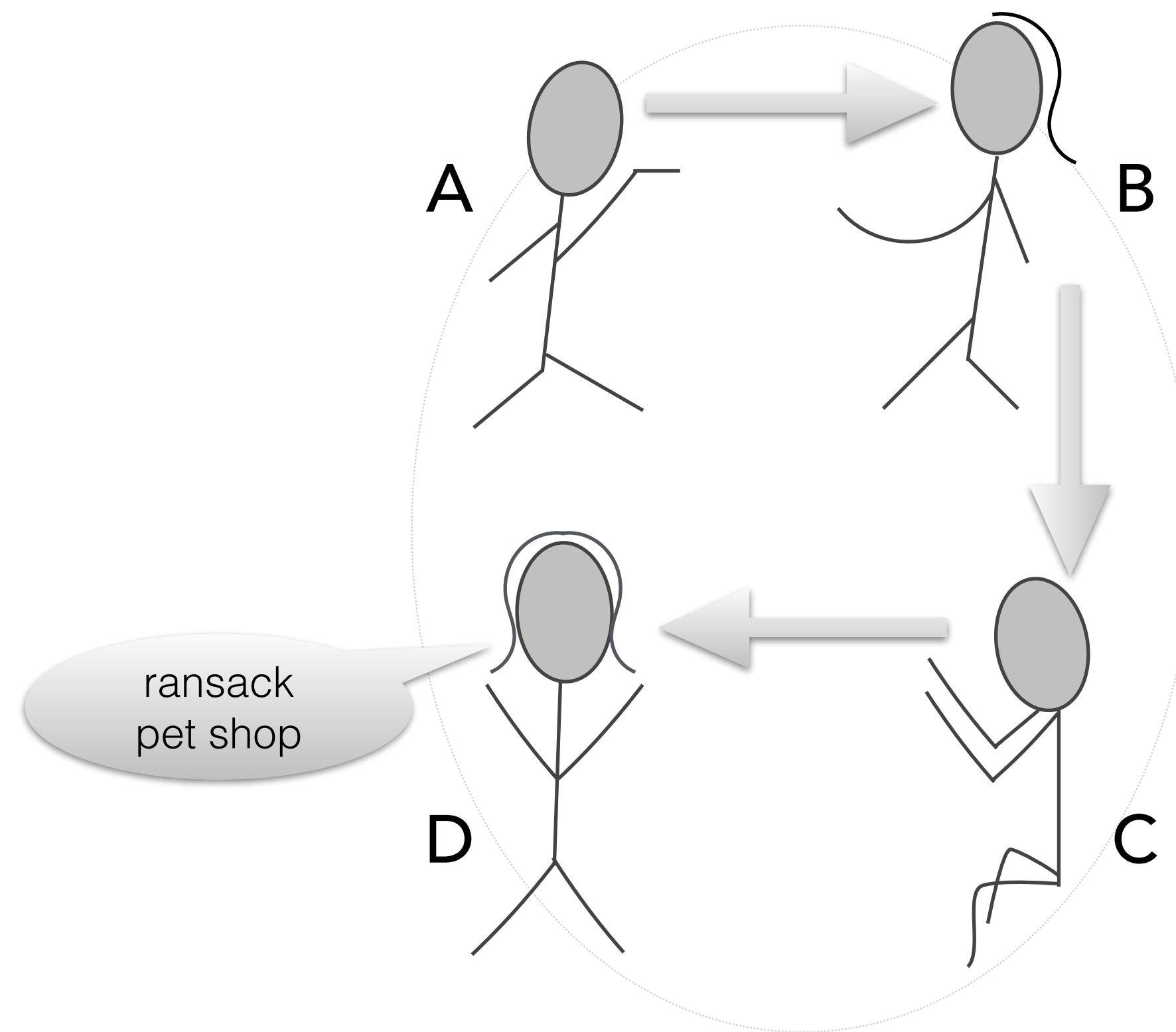
Filter that doesn't
take input $\varepsilon_1\varepsilon_2$ or
produce output $\varepsilon_2\varepsilon_1$

The Telephone Game



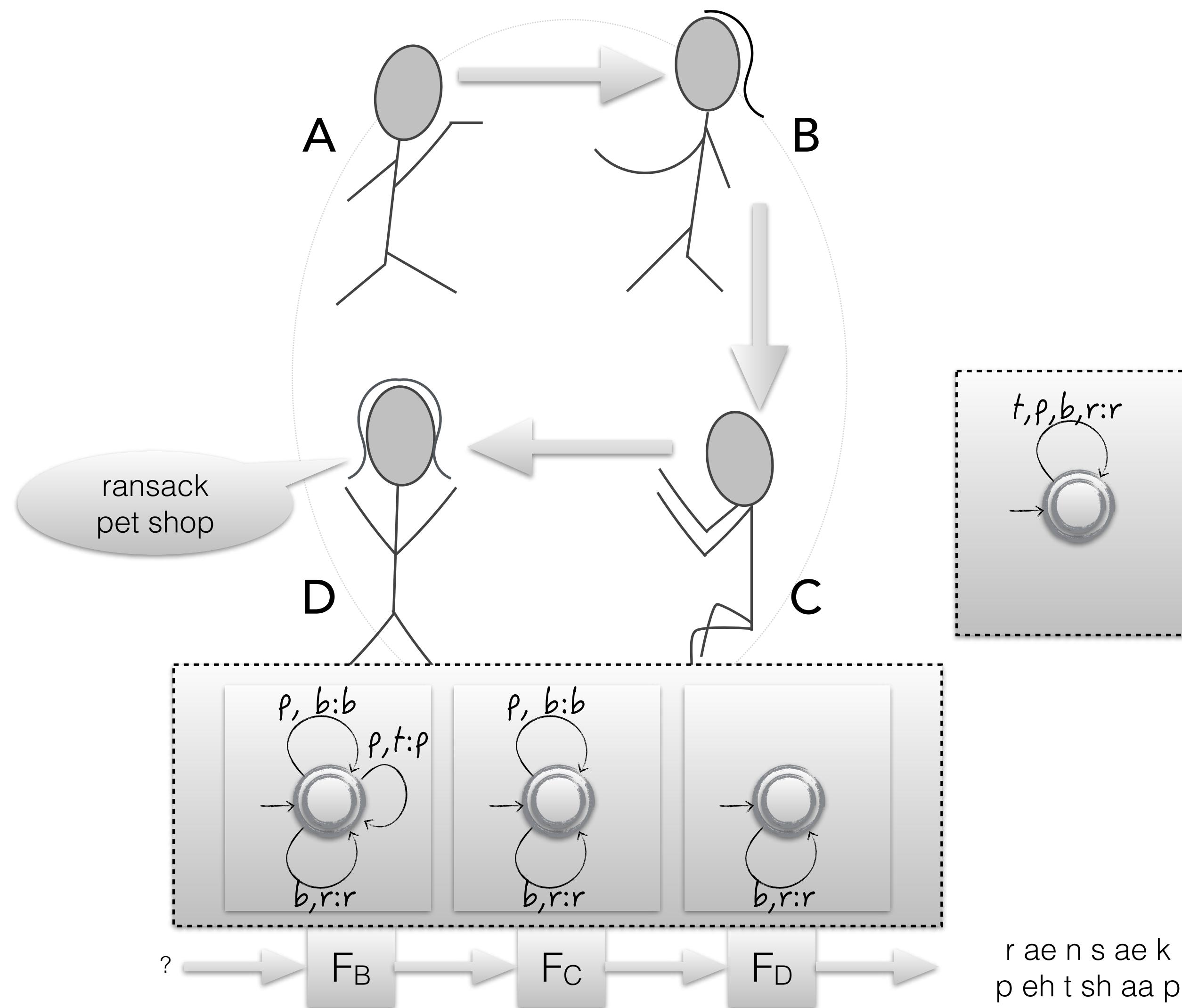
Given what D said,
can we infer the message A started with?

The Telephone Game

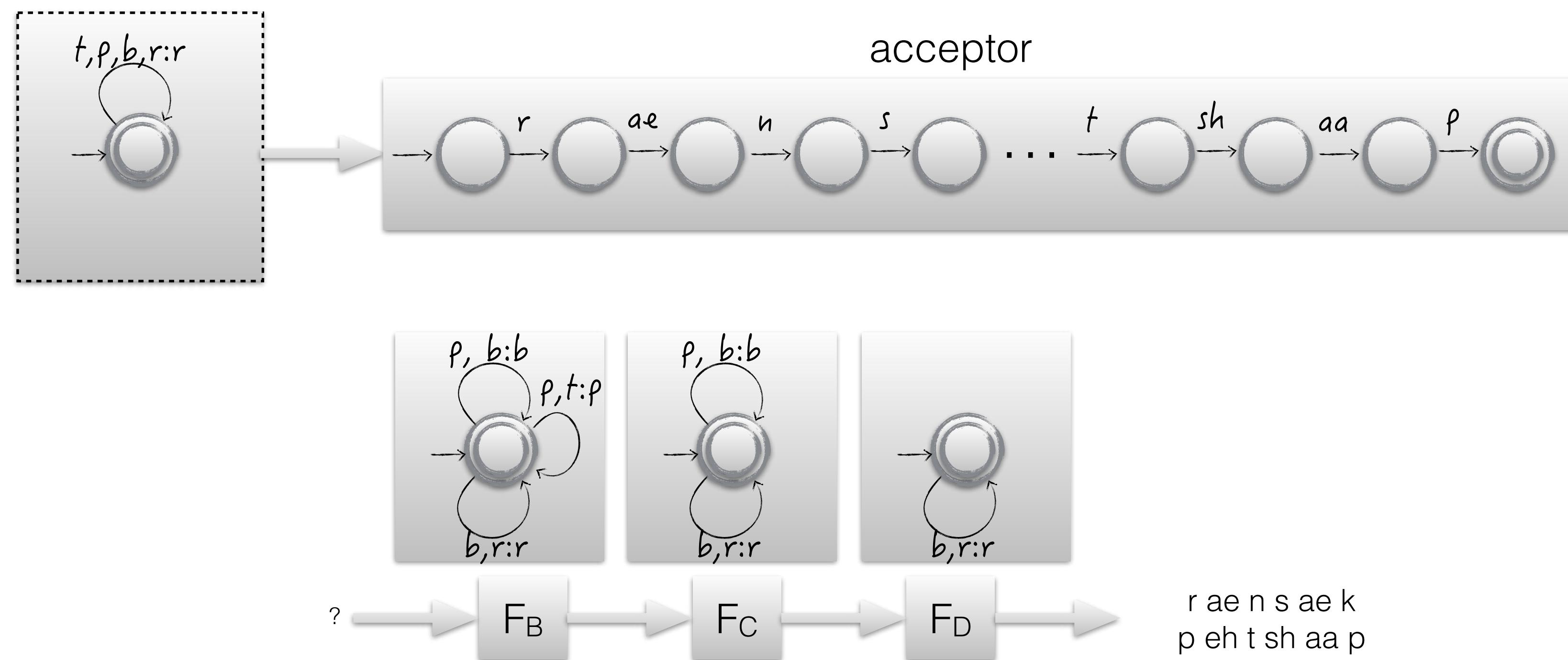


Model the errors made by each player using an FST

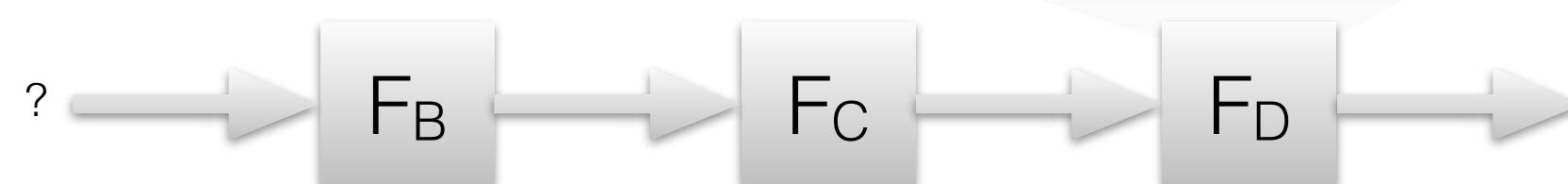
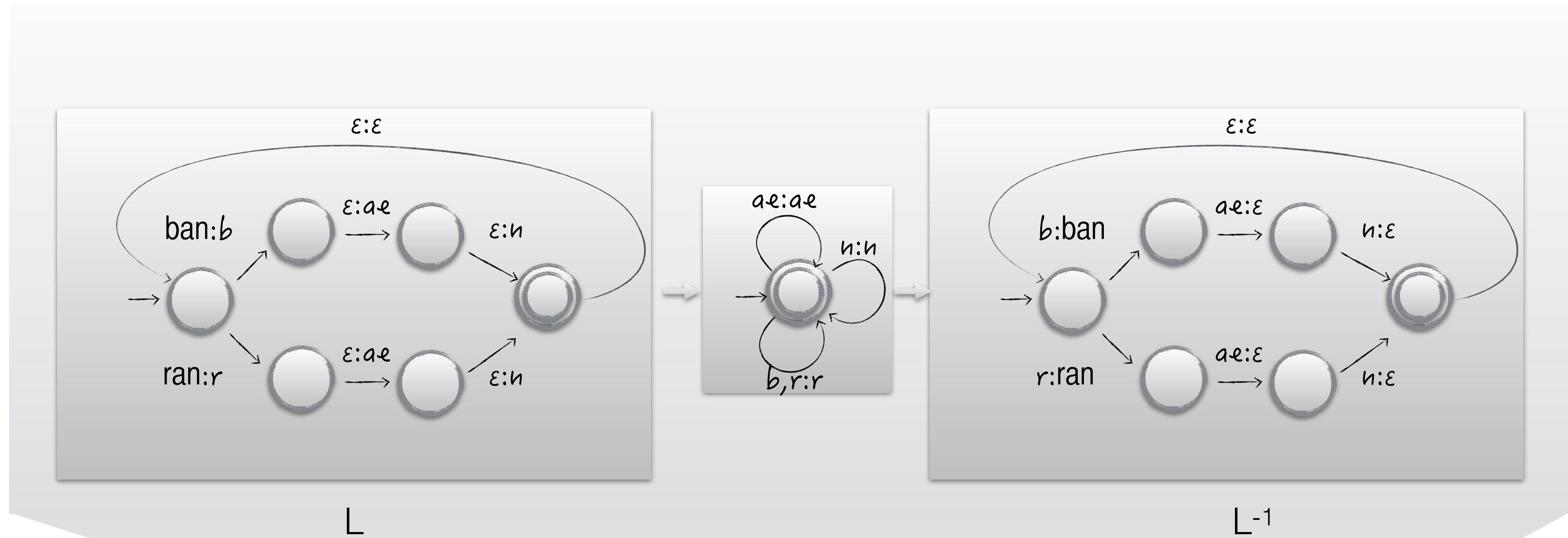
The Telephone Game



The Telephone Game

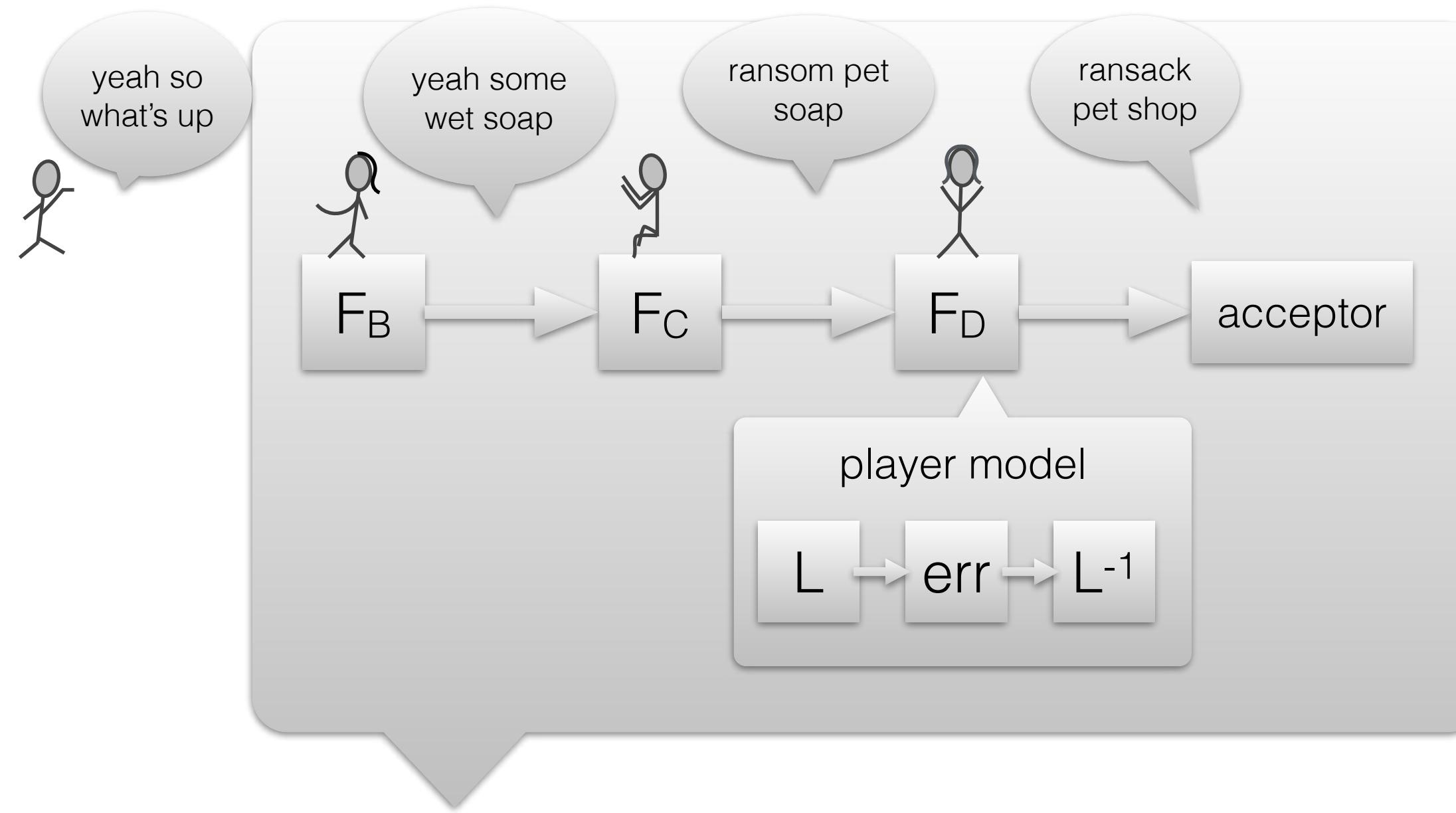


The Telephone Game



ransack
pet shop

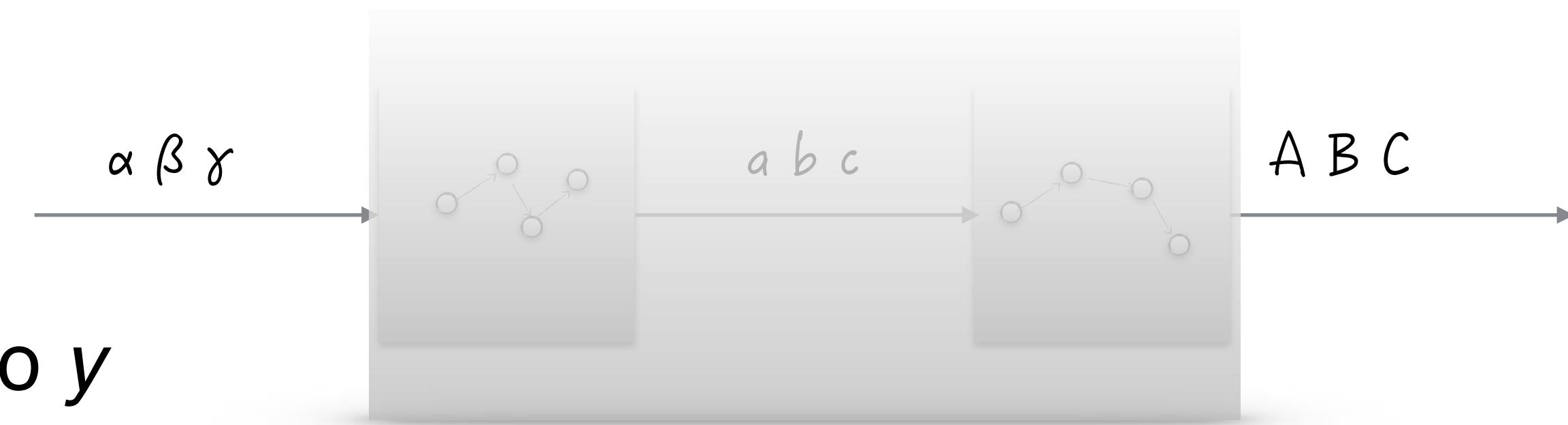
The Telephone Game



Find the best path in this FST
Read off the input words on the arcs
Can also find the best combination of paths in each player FST

Composition: Recap

- If T_1 transduces x to z ,
and T_2 transduces z to y ,
then $T_1 \circ T_2$ transduces x to y



- Note: output alphabet of $T_1 \subseteq$ input alphabet of T_2
- E.g. If T_1 removes punctuation symbols from a string, and T_2 changes uppercase letters to lowercase letters, then $T_1 \circ T_2$ brings about both changes

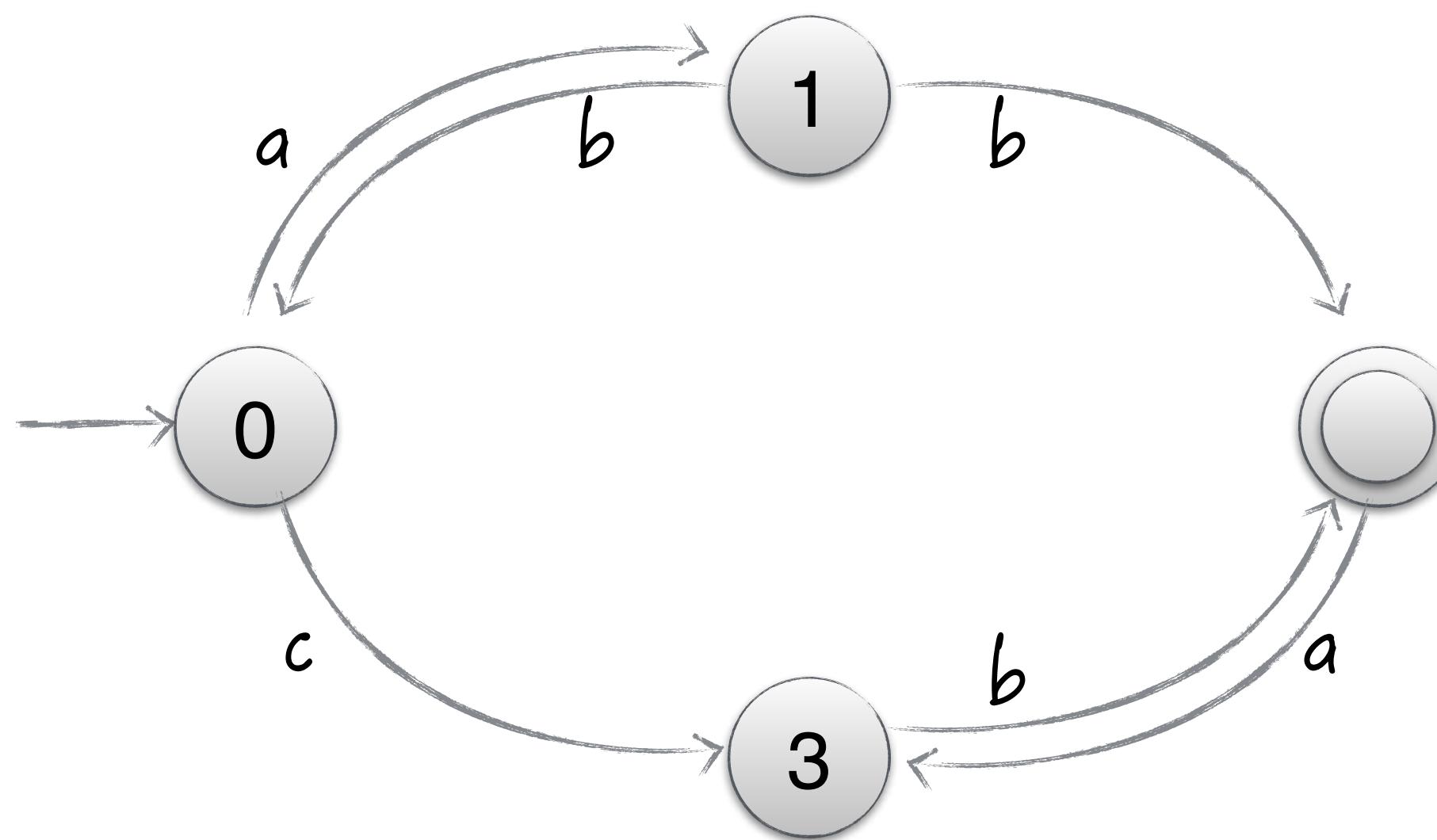
Determinization and Minimization

- WFSTs constructed using various operations (or designed by hand) may have several redundancies
 - Affects the efficiency of subsequent operations
- Determinization and minimization seek to remove redundancies
 - Determinization can expand a WFST, but makes it faster to process an input string
 - Minimization results in the smallest number of states
- Will discuss WFSAs here. Extends to WFSTs.

Deterministic FSAs

- An FSA is **deterministic** if:
 - Unique start state
 - No two transitions from a state share the same label
 - No epsilon labels

Any input sequence yields a unique path (if at all)

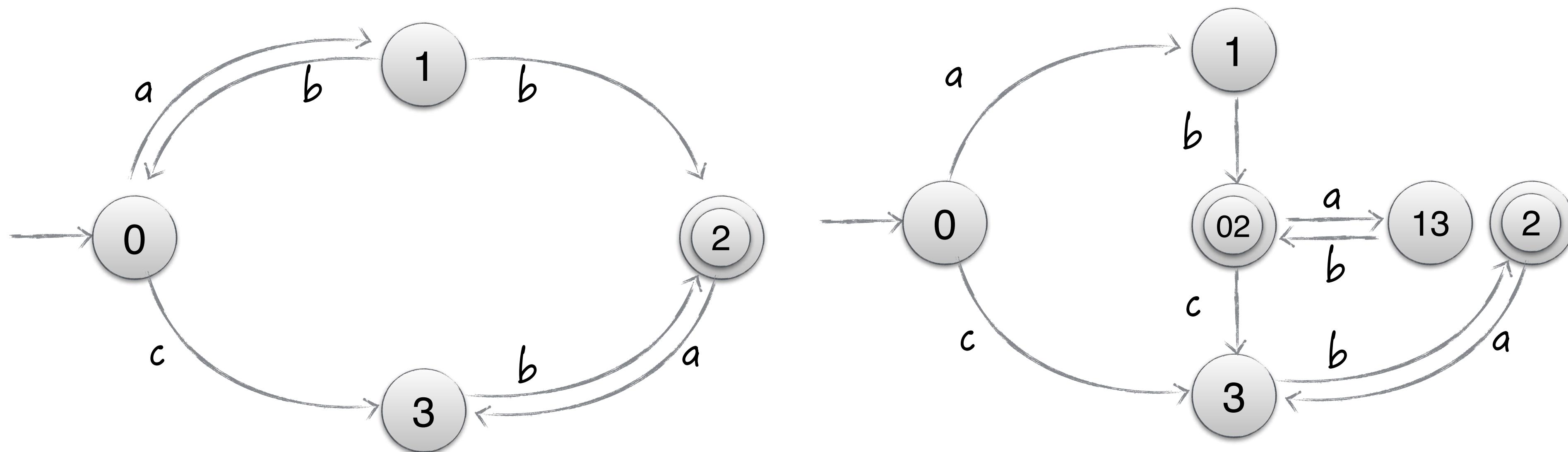


Deterministic or non-deterministic?

Determinization

Construct an equivalent deterministic FSA

States correspond to
subsets of states in
the original FSA

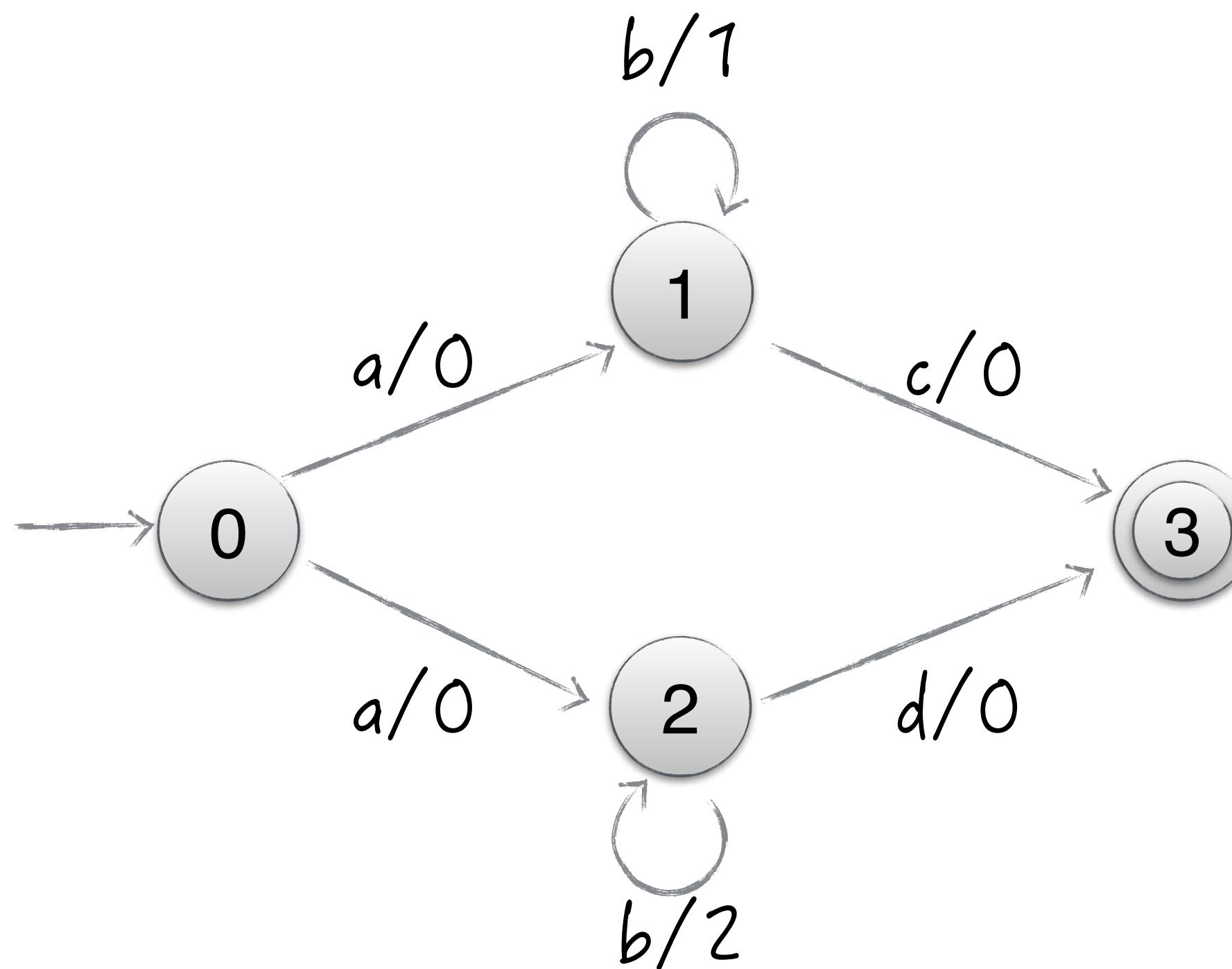


non-deterministic FSA

equivalent deterministic FSA

Determinization: Weighted FSA

Some *Weighted-FSAs* are not determinizable! [M97]

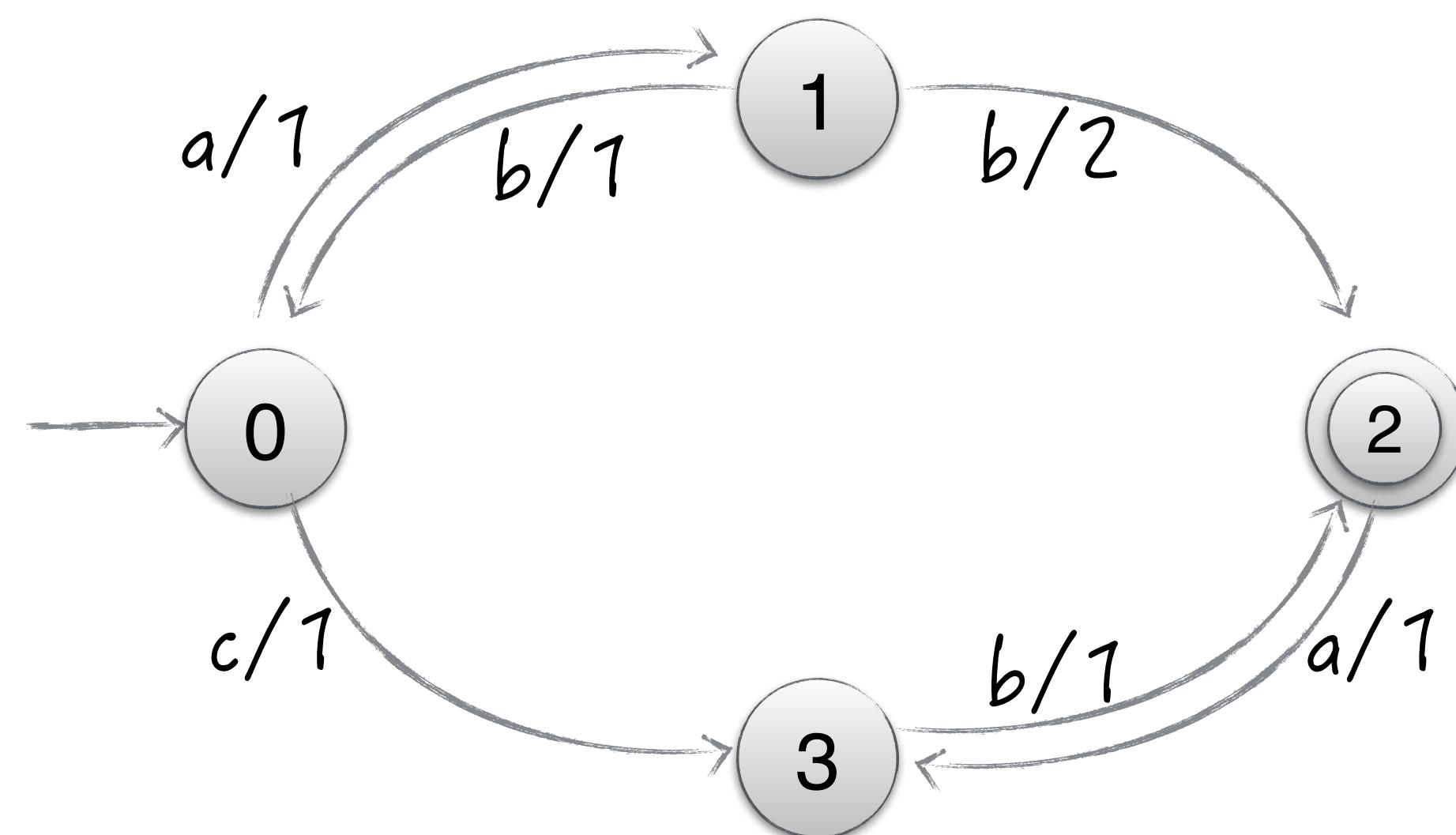


Weight of string $ab^n c = n$ and weight of $ab^n d = 2n$

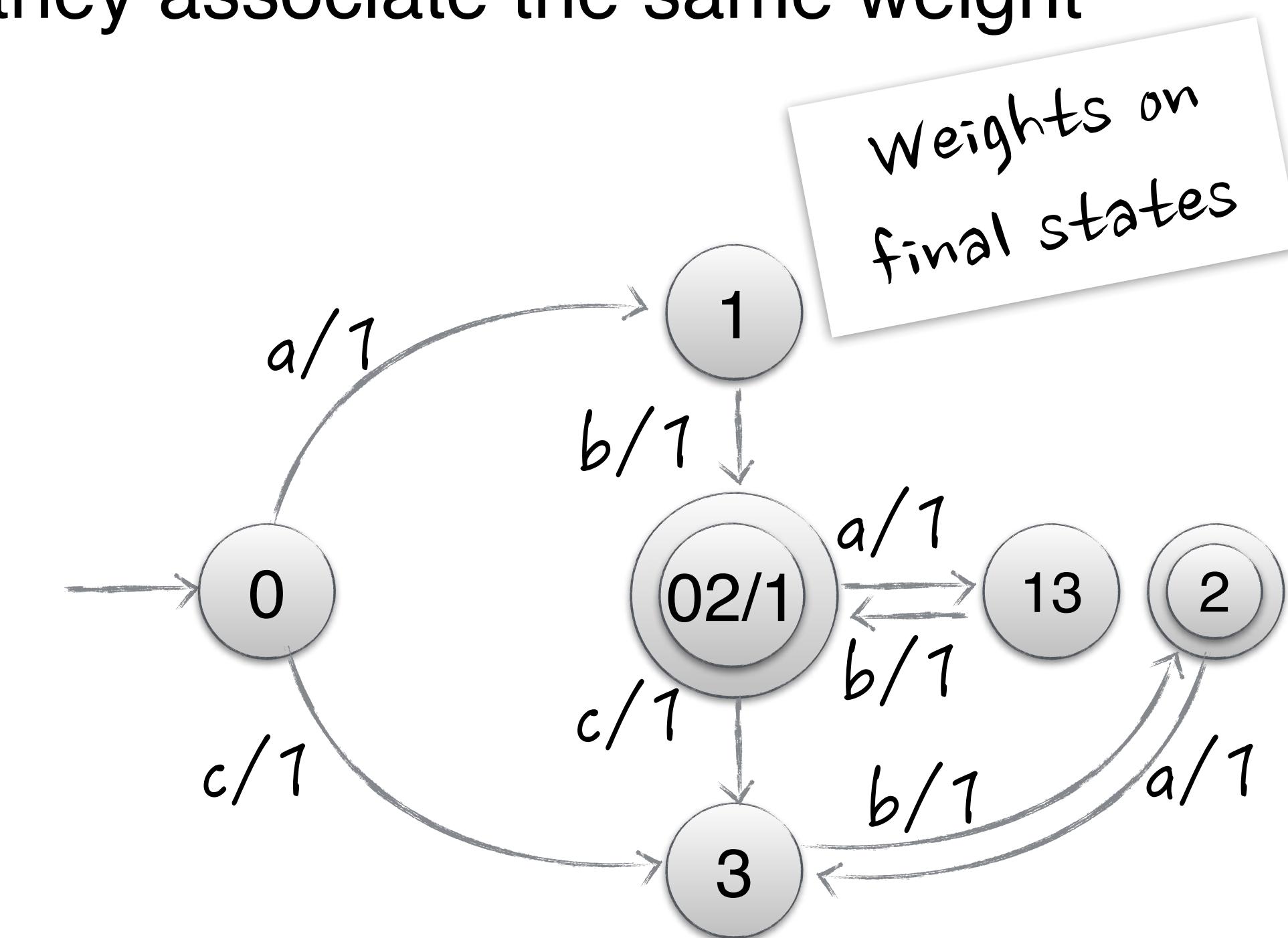
After seeing ab^n an FSA can't remember n

Determinization: Weighted FSA

Two WFSAs are equivalent if they associate the same weight to each input string



non-deterministic WFSA

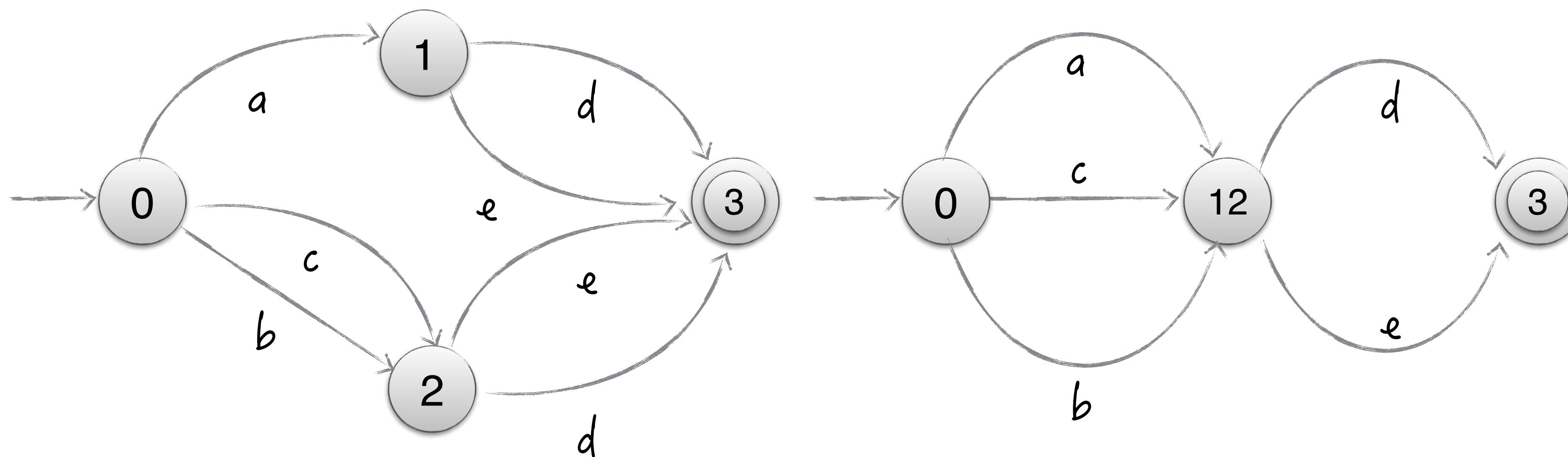


equivalent deterministic WFSA

Minimization

Minimization: find an equivalent deterministic FSA with the least number of states (and transitions)

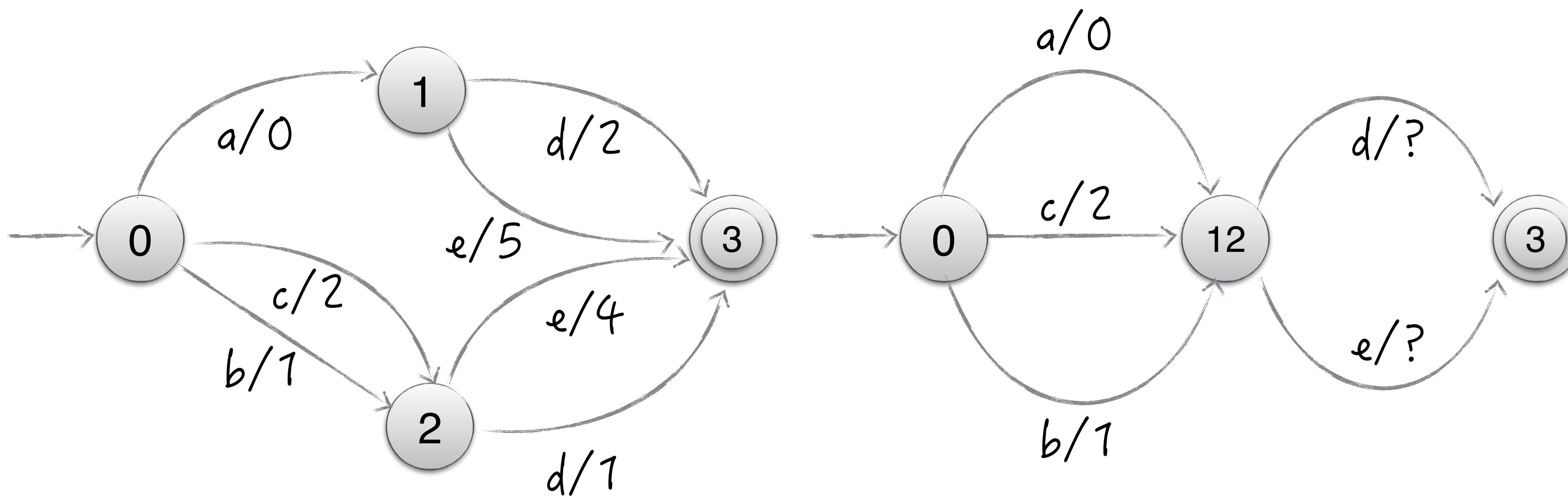
Unweighted FSAs have a unique minimal FSA [Aho74]



Obtained by identifying and merging *equivalent states*

Minimization: Weighted FSA

Two states are equivalent only if for every input string, the outcome — weight assigned to the string, if accepted — starting from the two states are the same

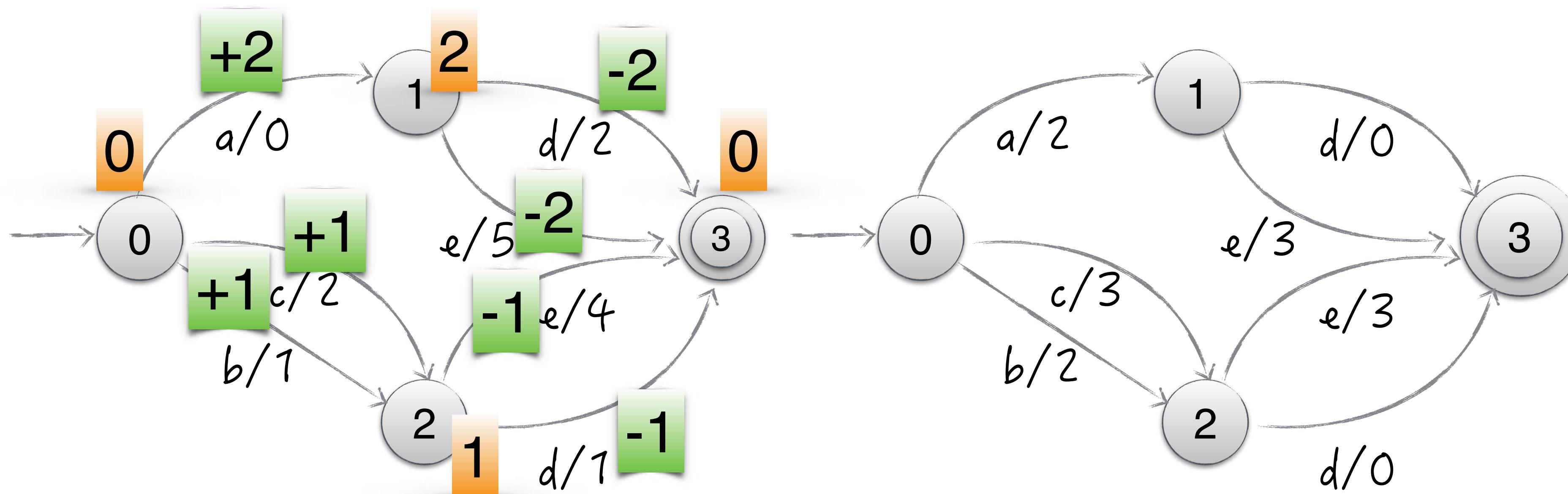


Redistribute weights before identifying equivalent states

Minimization: Weighted FSA

Reweighting OK as long as resulting WFSAs are equivalent

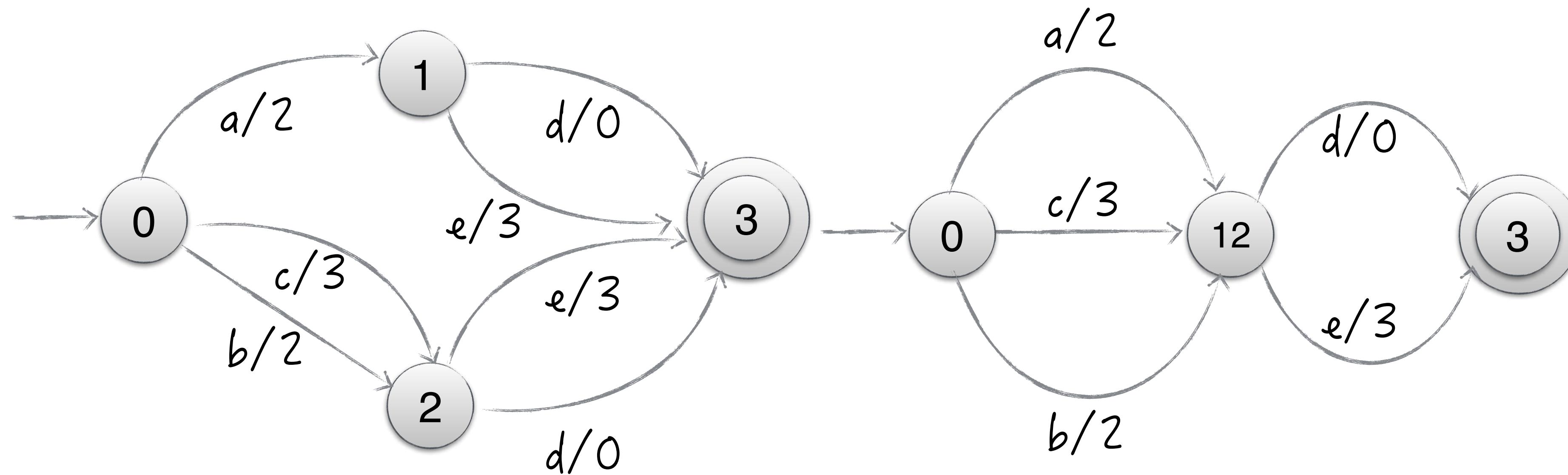
Can reweight using a “potential function” on states



“Weight pushing”: Reweighting using a potential function that optimally moves weights towards the start state

Minimization: Weighted FSA

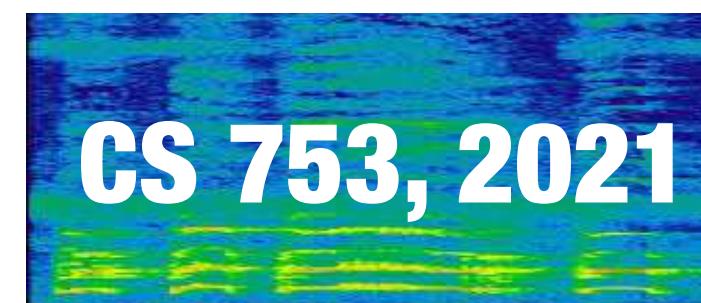
After weight-pushing, can simply apply unweighted FSA minimization (treating label/weight as label)



Guaranteed to yield a minimal WFSA (under some technical conditions required for weight-pushing)

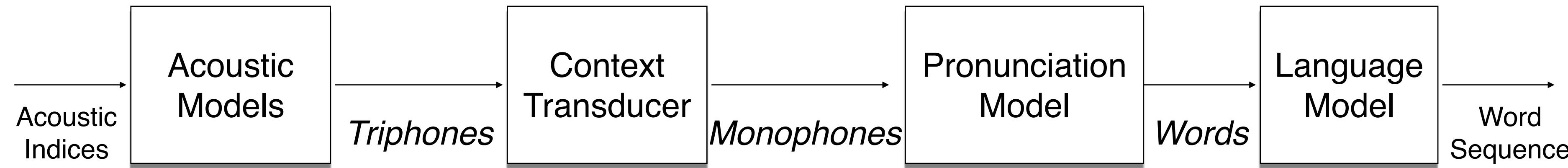
WFSTs in ASR

Lecture 4b

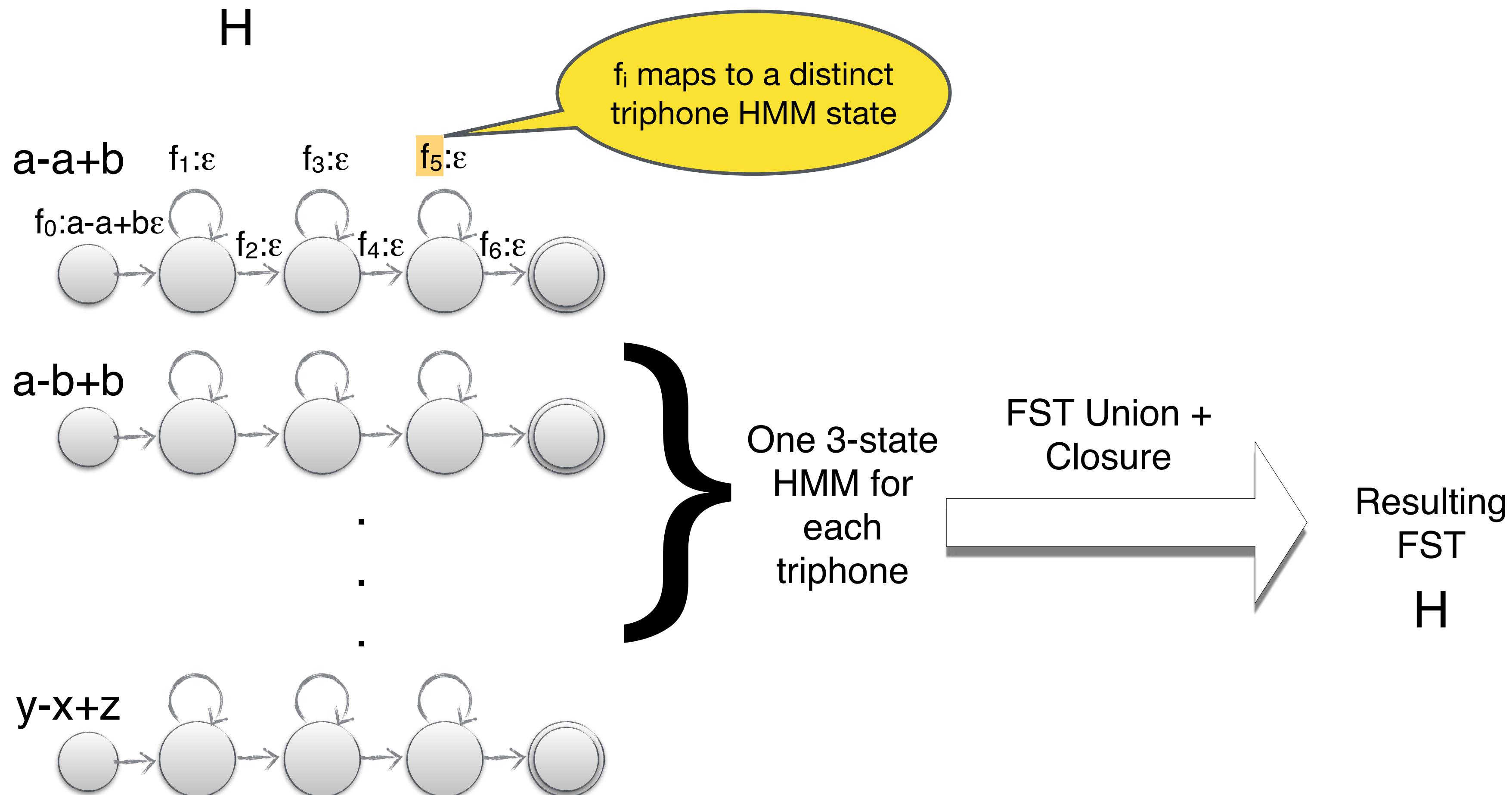


Instructor: Preethi Jyothi, IITB

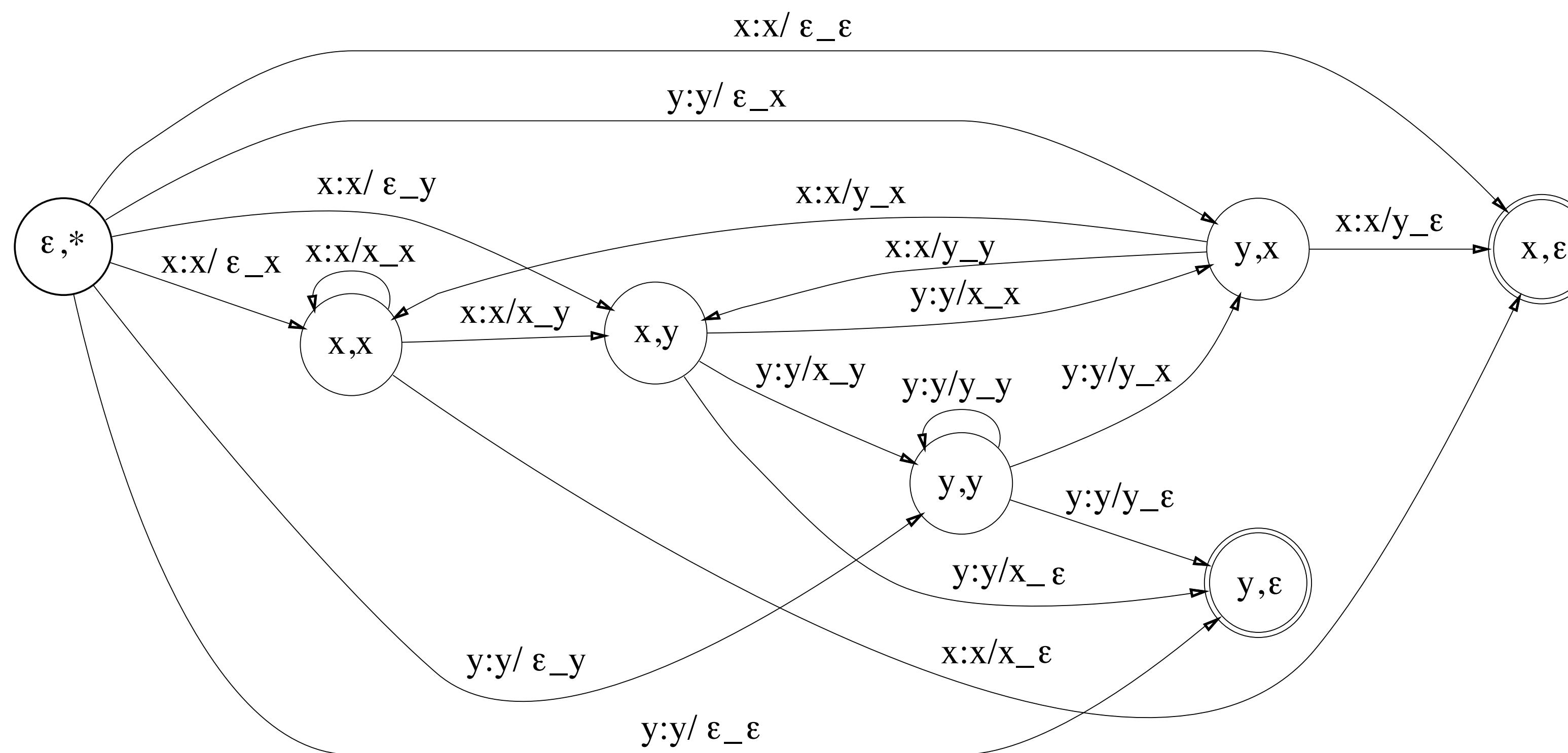
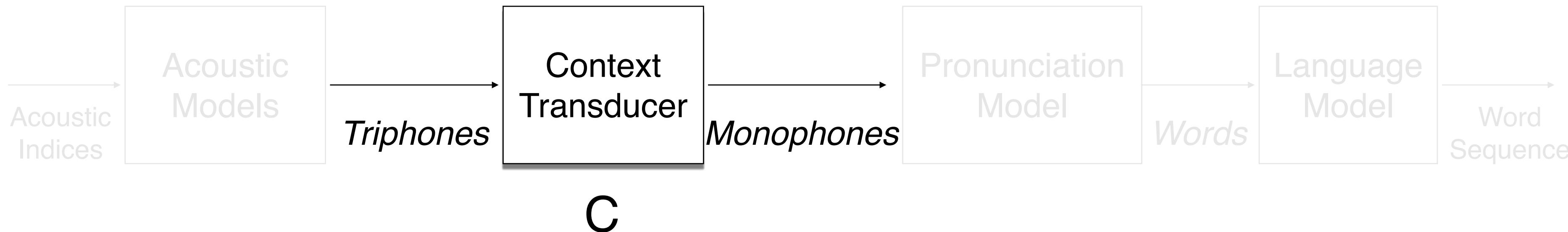
WFST-based ASR System



WFST-based ASR System

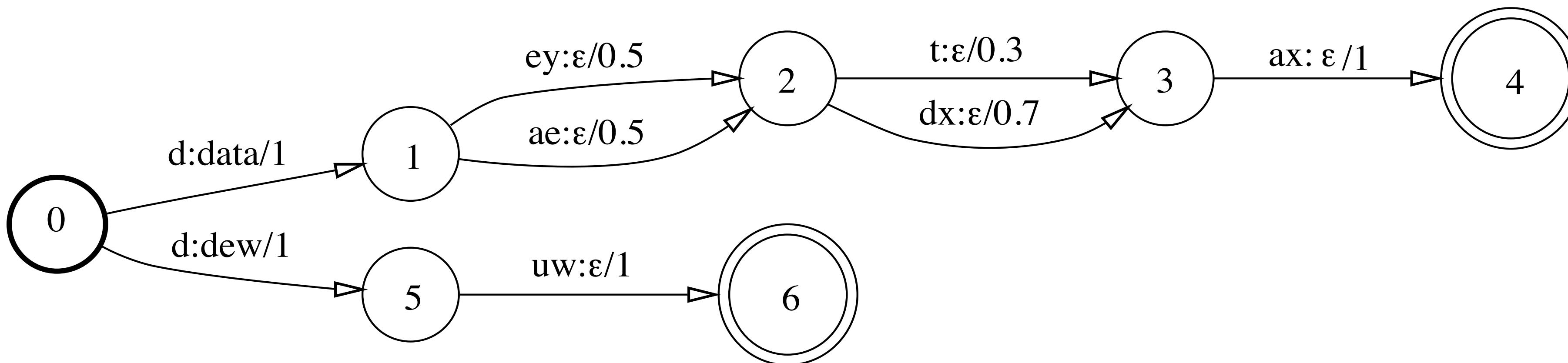
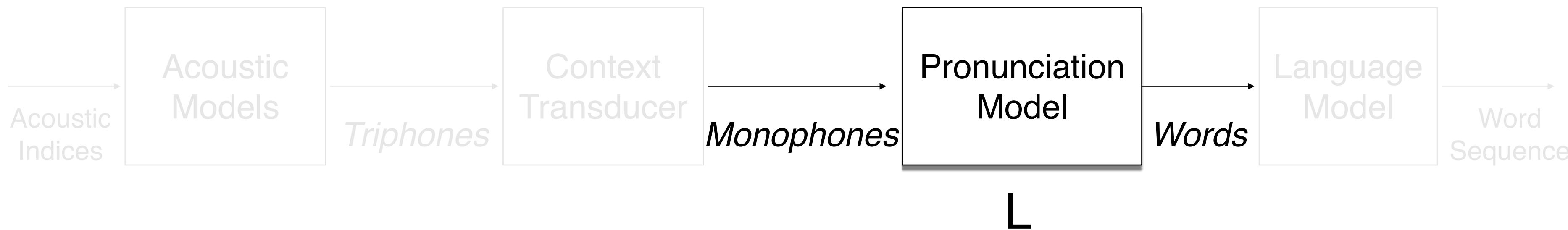


WFST-based ASR System

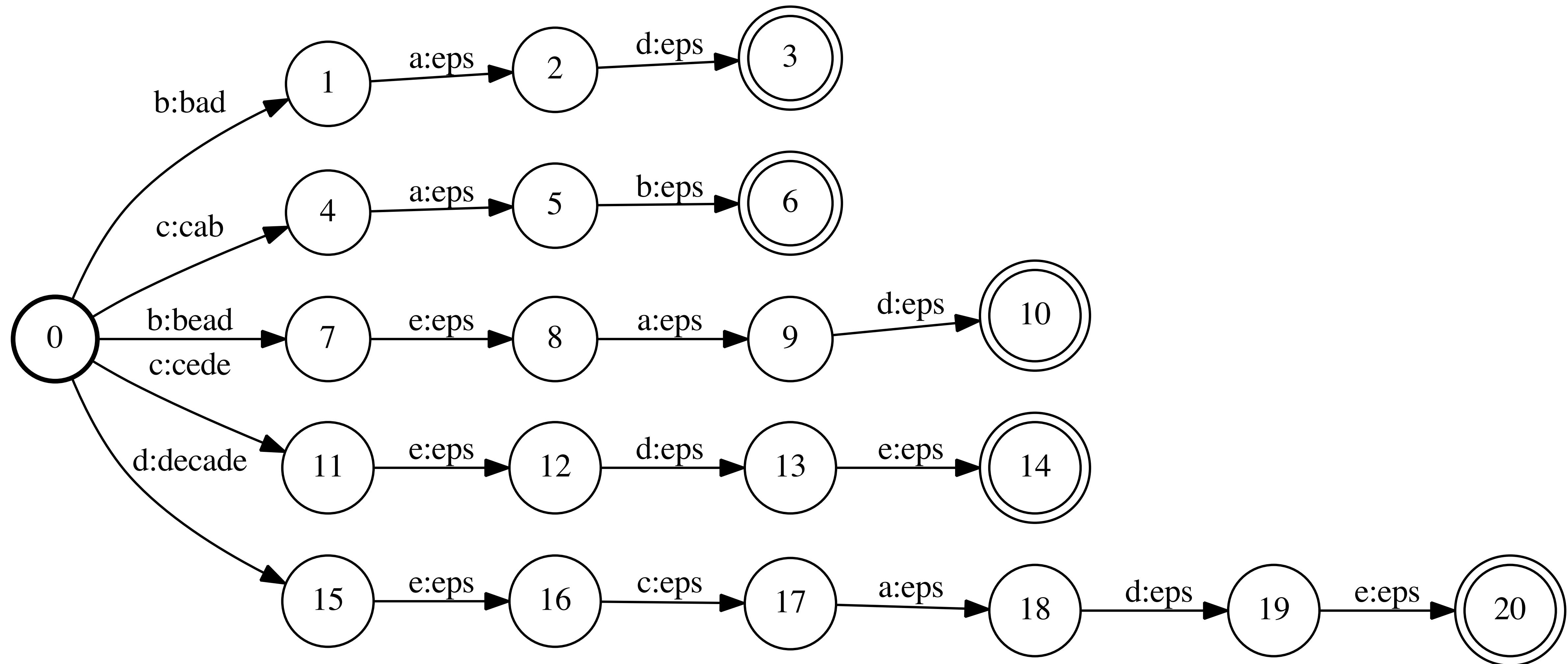


C⁻¹: Arc labels: “monophone : phone / left-context_right-context”

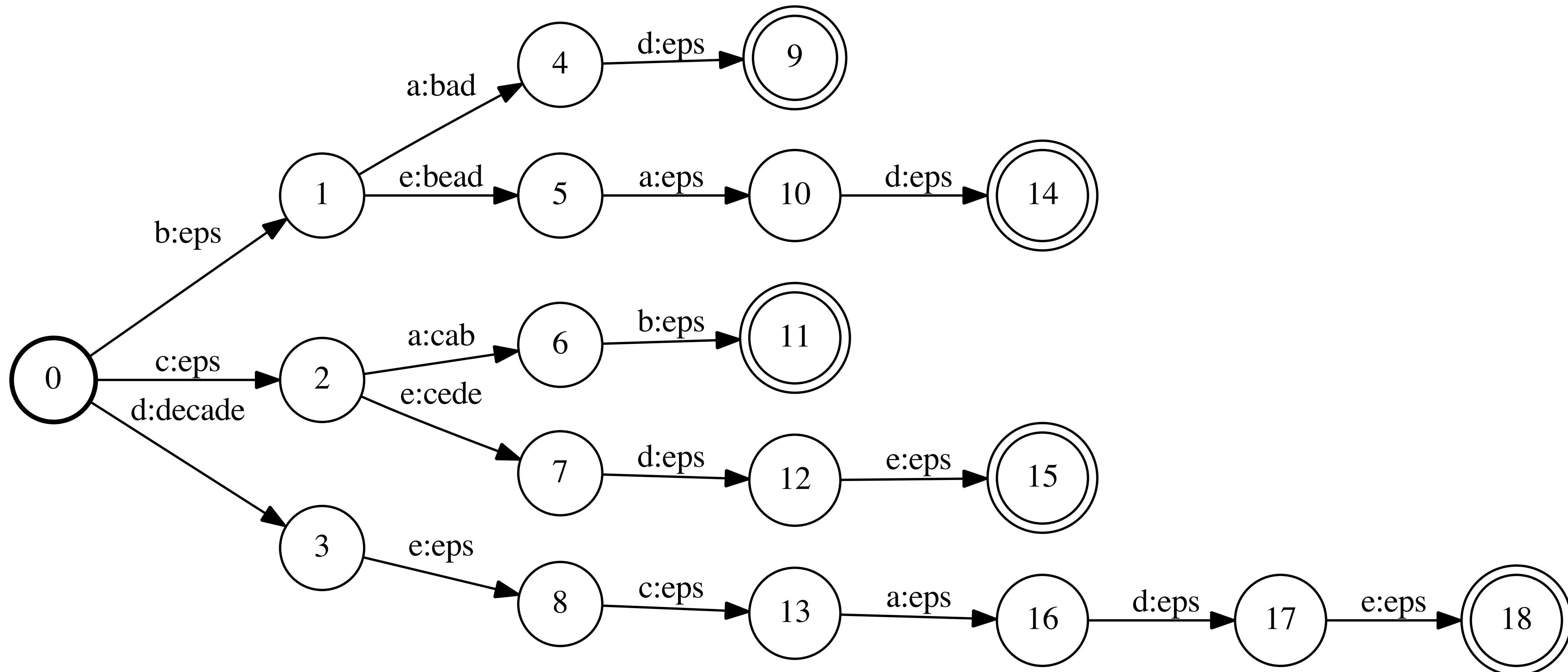
WFST-based ASR System



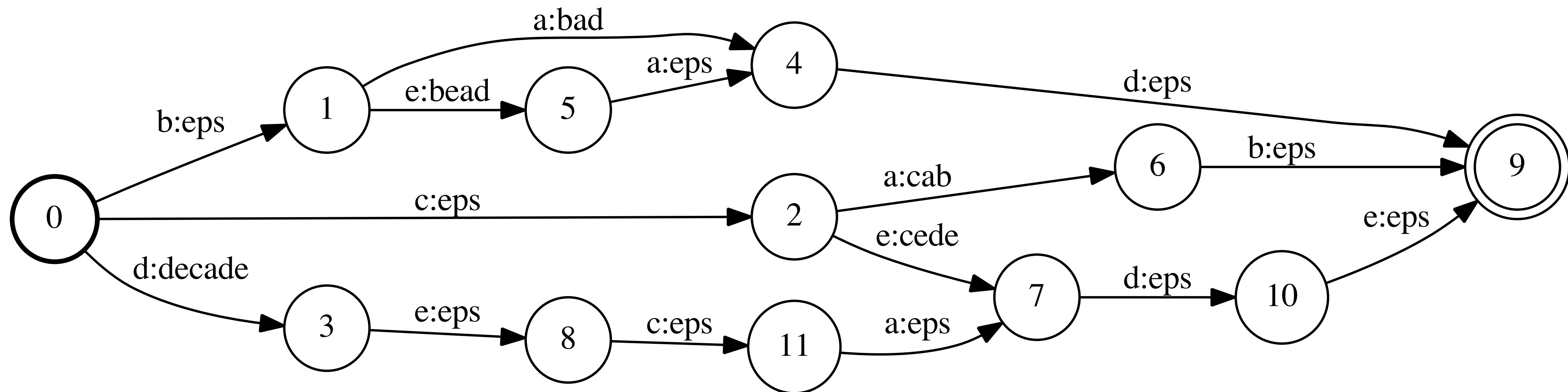
Example: Dictionary WFST



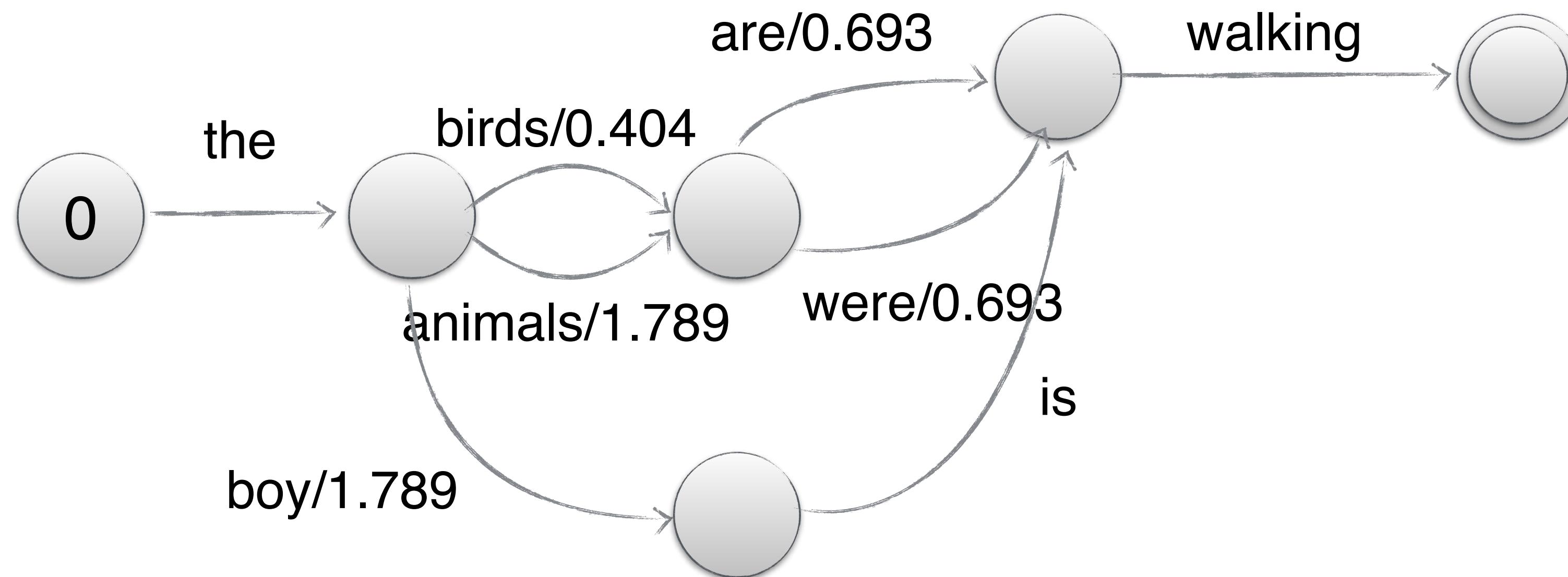
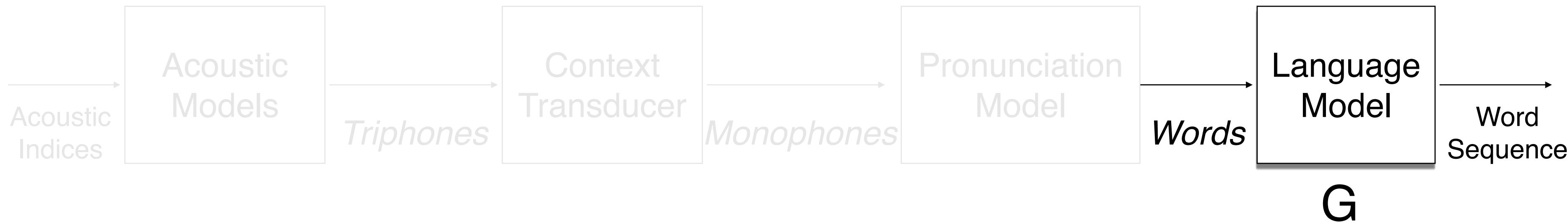
Determinized Dictionary WFST



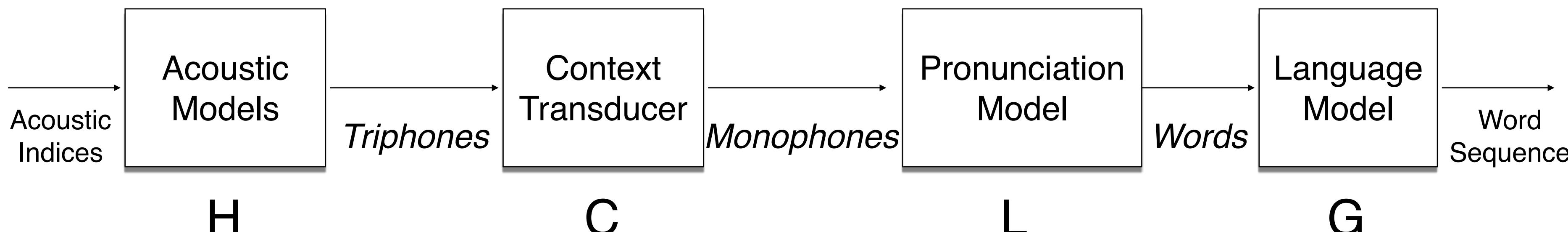
Minimized Dictionary WFST



WFST-based ASR System



Decoding



Carefully construct a decoding graph D using optimization algorithms:

$$D = \min(\det(H \circ \det(C \circ \det(L \circ G))))$$

Need to add *disambiguation symbols* to L in order to make $L \circ G$ determinizable!

When are disambiguation symbols needed?

1. When you deal with homophones i.e., same pronunciation, different words.
2. When pronunciation of a word is a prefix of another.

book b uh k #1

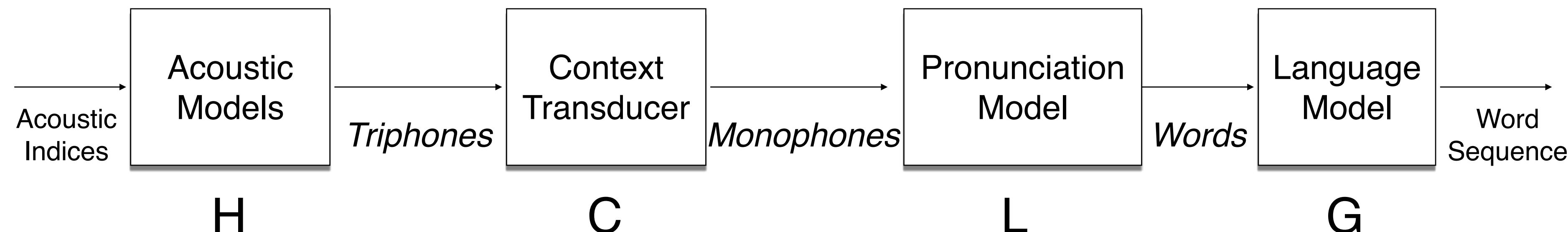
books b uh k s

right r ay t #1

write r ay t #2

:

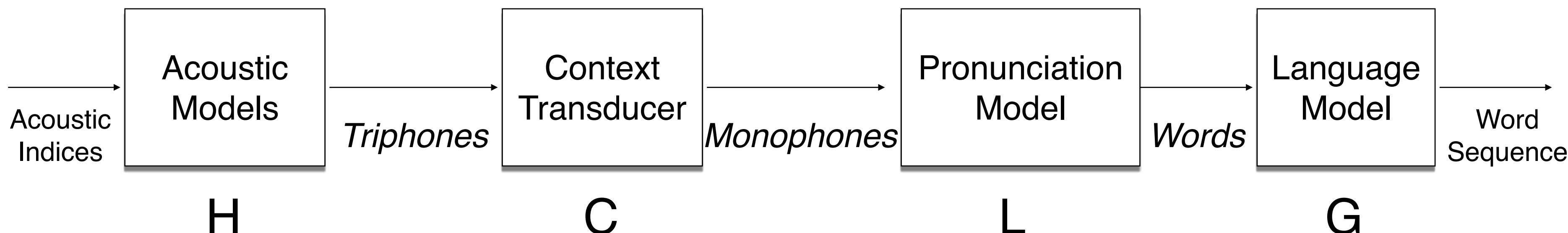
Decoding



Carefully construct a decoding graph D using optimization algorithms:

$$D = \min(\det(H \circ \det(C \circ \det(L \circ G))))$$

Decoding

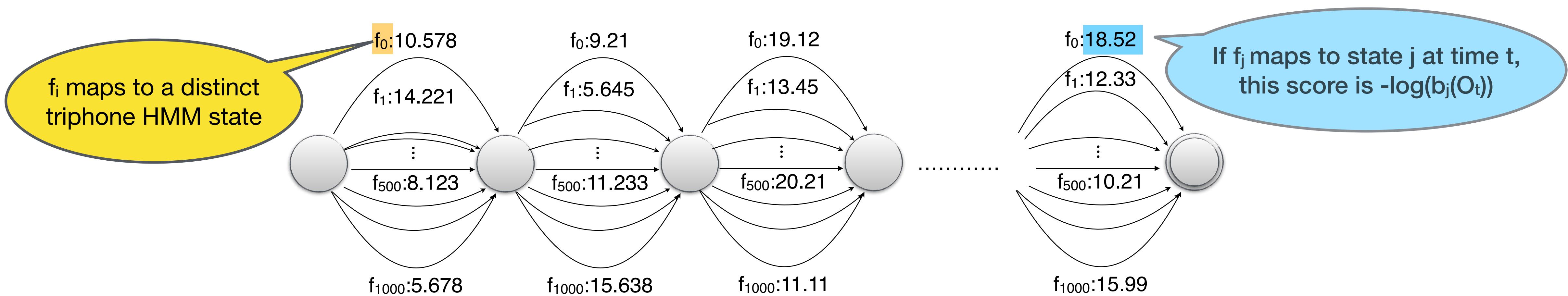


Carefully construct a decoding graph D using optimization algorithms:

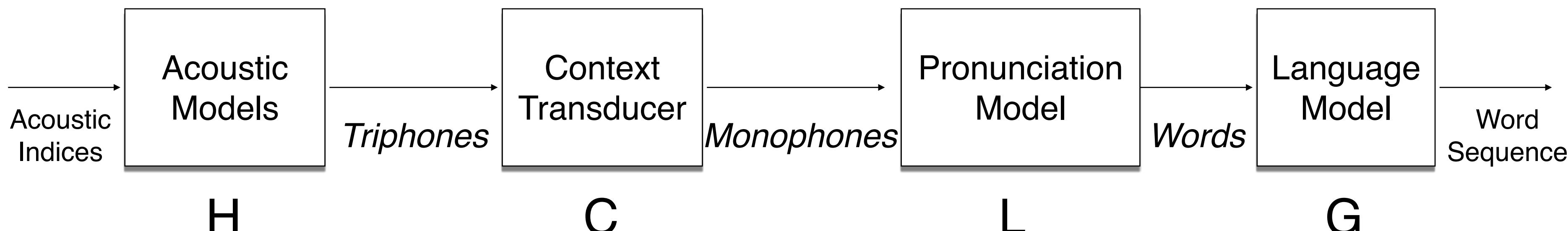
$$D = \min(\det(H \circ \det(C \circ \det(L \circ G))))$$

Given a test utterance O , how do I decode it?

Assuming ample compute, first construct the following machine X from O .



Decoding

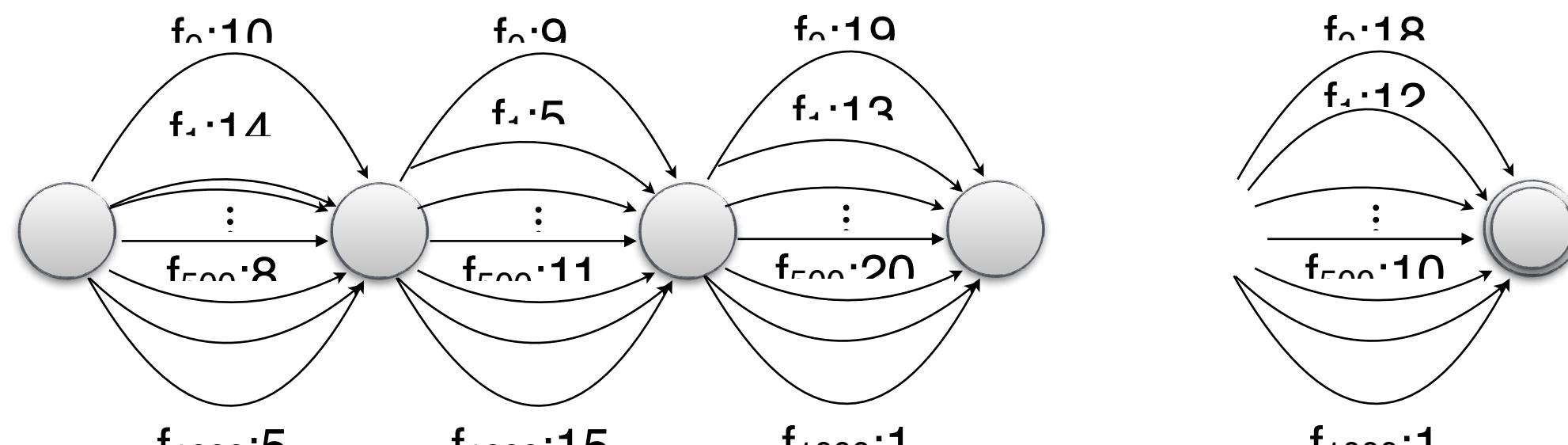


Carefully construct a decoding graph D using optimization algorithms:

$$D = \min(\det(H \circ \det(C \circ \det(L \circ G))))$$

Given a test utterance O , how do I decode it?

Assuming ample compute, first construct the following machine X from O .



X

$$W^* = \arg \min_{W=\text{out}[\pi]} X \circ D$$

where π is a path in the composed FST
 $\text{out}[\pi]$ is the output label sequence of π

X is never typically constructed;
 D is traversed dynamically using approximate search algorithms
 (discussed later in the semester)

Impact of WFST Optimizations

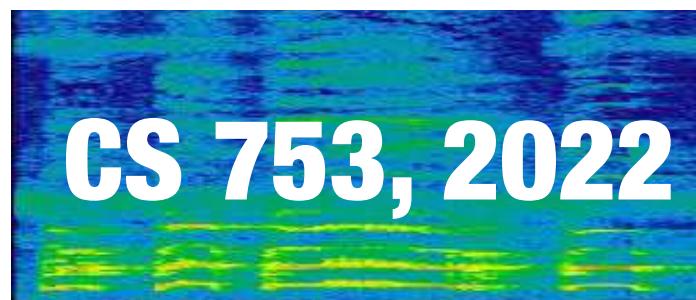
40K NAB Evaluation Set '95 (83% word accuracy)

network	states	transitions
G	1,339,664	3,926,010
$L \circ G$	8,606,729	11,406,721
$det(L \circ G)$	7,082,404	9,836,629
$C \circ det(L \circ G))$	7,273,035	10,201,269
$det(H \circ C \circ L \circ G)$	18,317,359	21,237,992

network	x real-time
$C \circ L \circ G$	12.5
$C \circ det(L \circ G)$	1.2
$det(H \circ C \circ L \circ G)$	1.0
$push(min(F))$	0.7

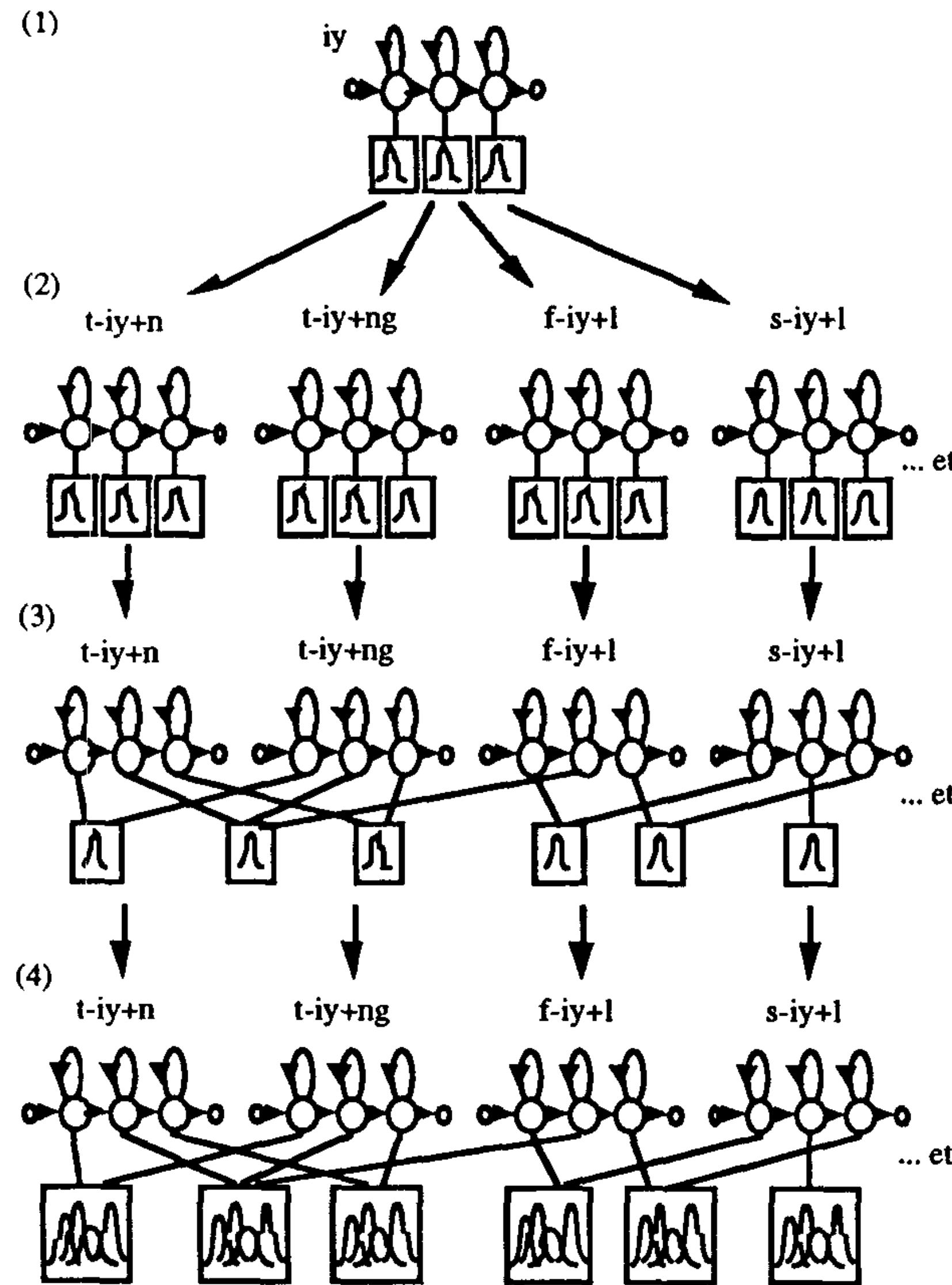
Tied-state HMMs + WFSTs (Live Session)

Lecture 3b



Instructor: Preethi Jyothi, IITB

Tied state HMMs



Four main steps in building a tied state HMM system:

1. Create and train 3-state monophone HMMs with single Gaussian observation probability densities
2. Clone these monophone distributions to initialise a set of untied triphone models. Train them using Baum-Welch estimation. Transition matrix remains common across all triphones of each phone.
3. For all triphones derived from the same monophone, cluster states whose parameters should be tied together.
4. Number of mixture components in each tied state is increased and models re-estimated using BW

WFST: Problem 1A

In the following problems, the input and output alphabets are both $\{a, b, c, A, B, C\}$.

Design a deterministic FST M_1 which takes a string of characters and converts them all to uppercase or lowercase, to match the case of the first character. As an example, the input $aABBc$ will be converted to $aabbC$.

WFST: Problem 1B

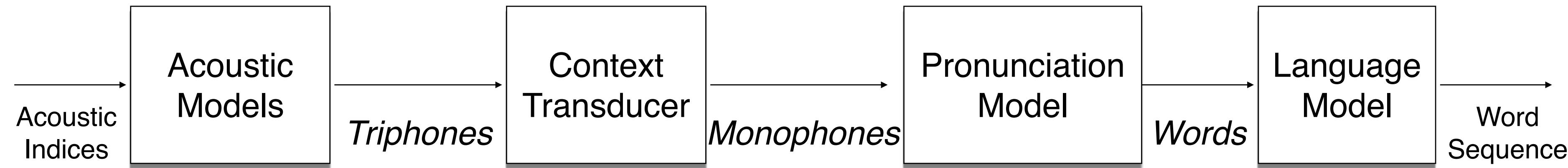
In the following problems, the input and output alphabets are both $\{a, b, c, A, B, C\}$.

Design a non-deterministic FST M_0 which takes a string and outputs it as it is, but changes only the first character as follows: If the string is of (non-zero) even length, then the first character of the output is lowercase and if the string is of odd length, then the first character is uppercase. For example, abC is converted to AbC while input $abCB$ results in the same string as the output. (The empty string should not result in a successful output.)

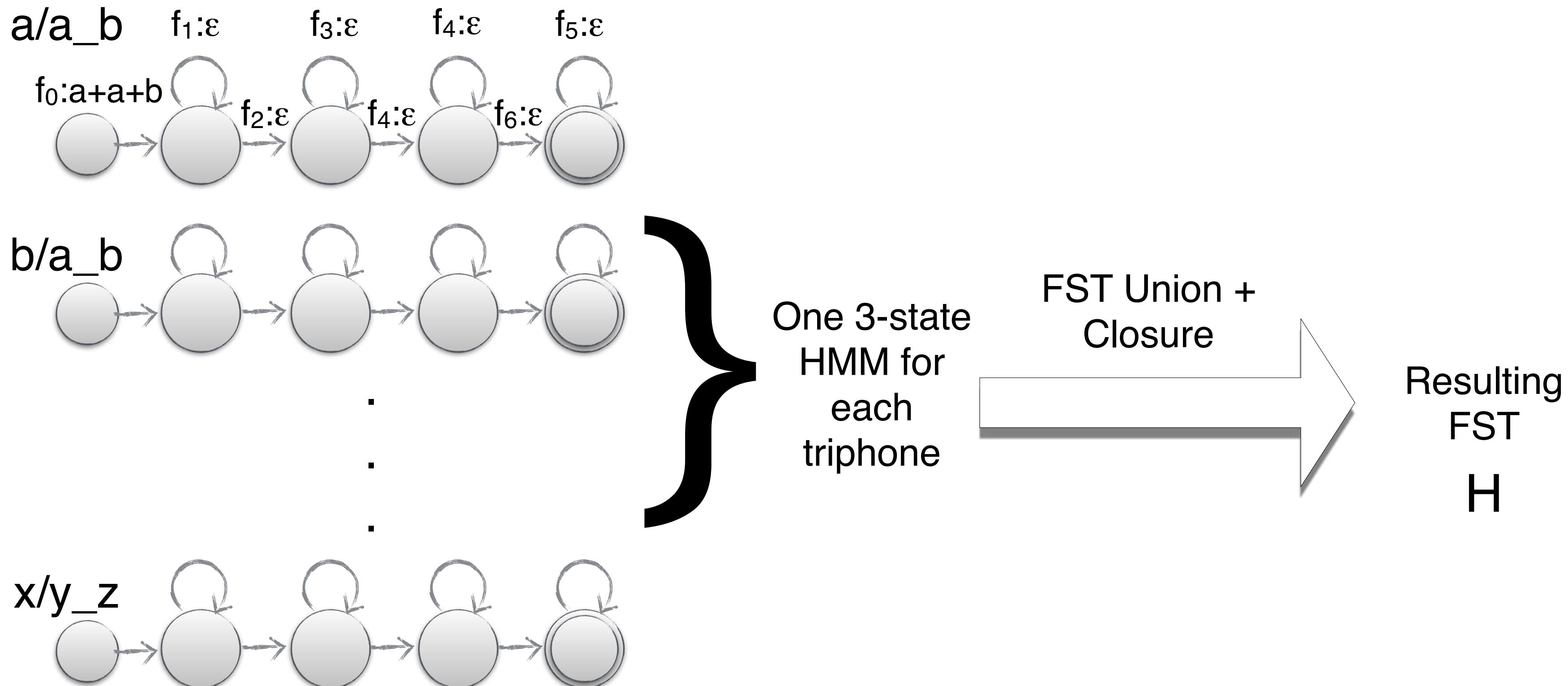
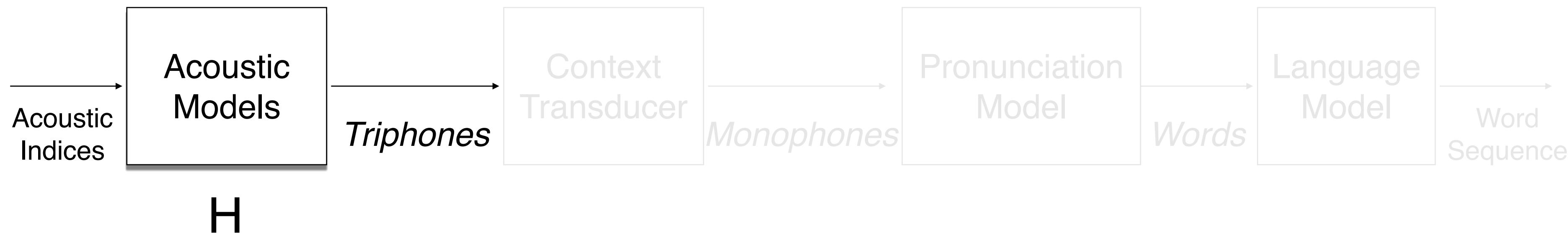
WFST: Problem 1C

Describe, in English, the behaviour of the composed FST $M_0 \circ M_1$.

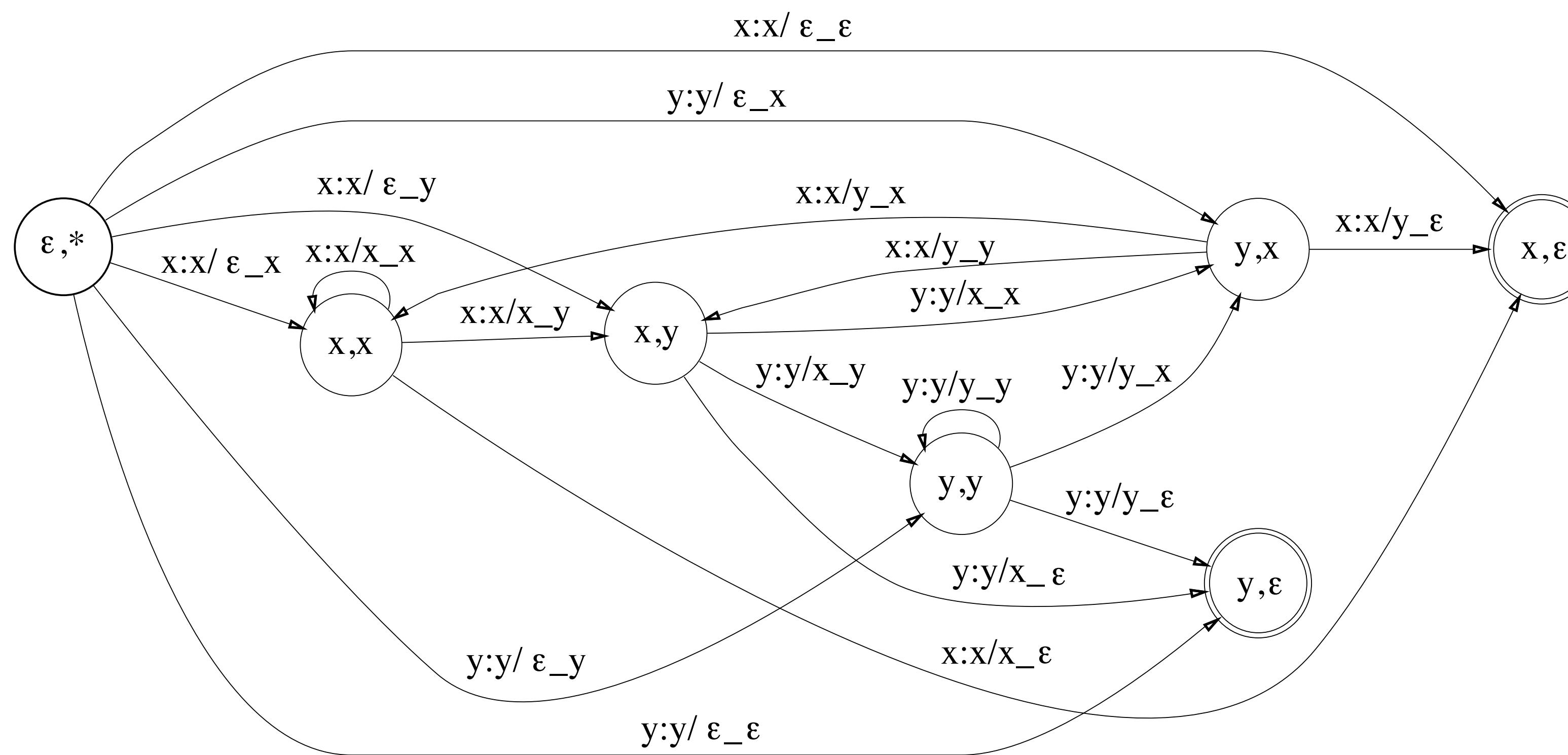
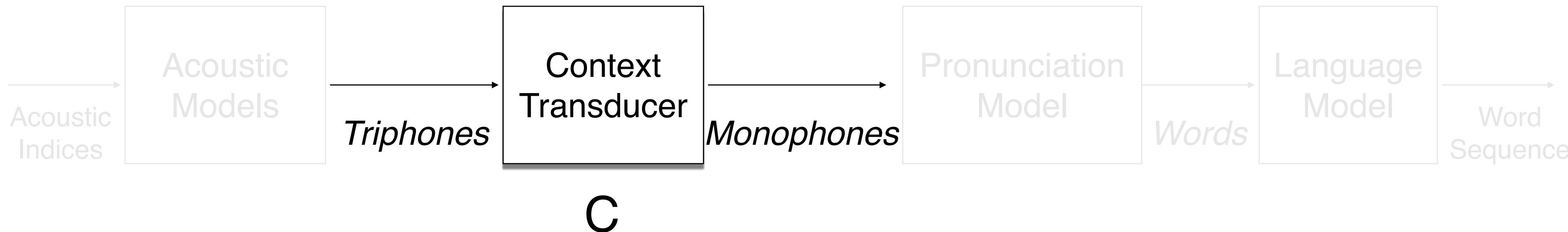
WFST-based ASR System



WFST-based ASR System

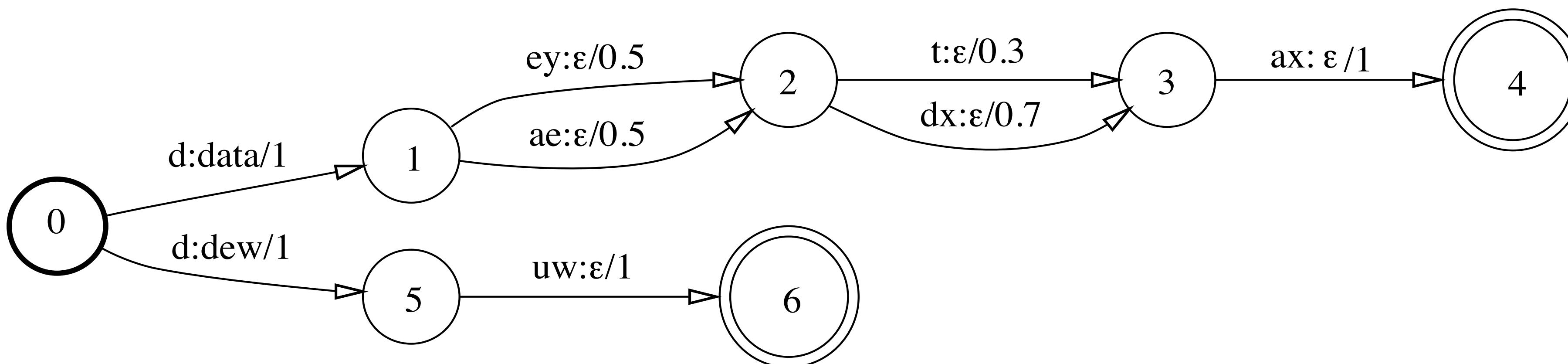
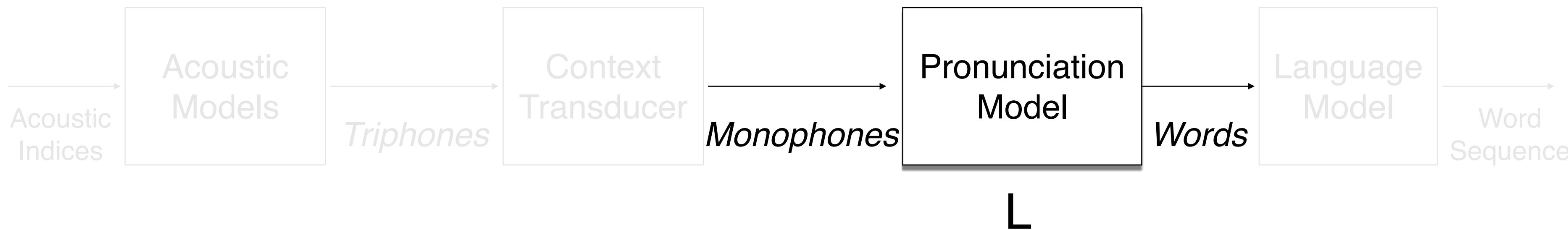


WFST-based ASR System

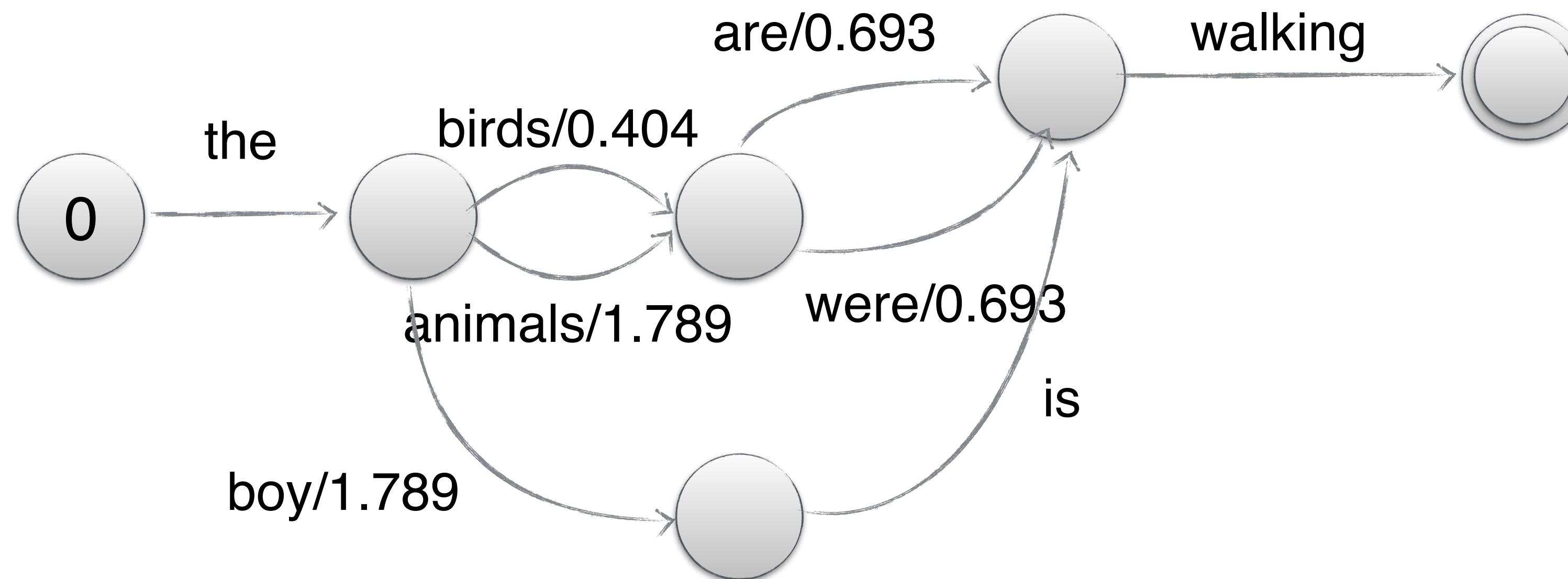
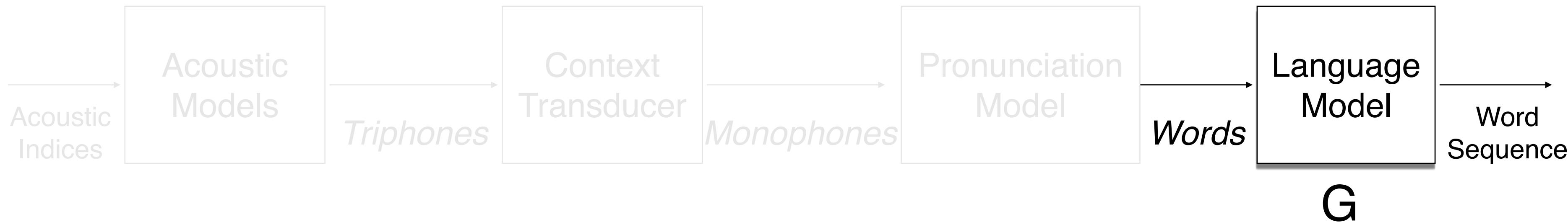


C⁻¹: Arc labels: “monophone : phone / left-context_right-context”

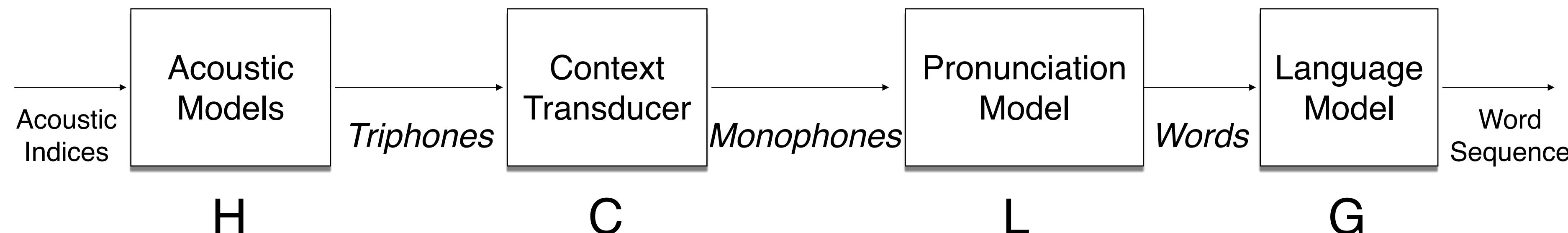
WFST-based ASR System



WFST-based ASR System



Decoding

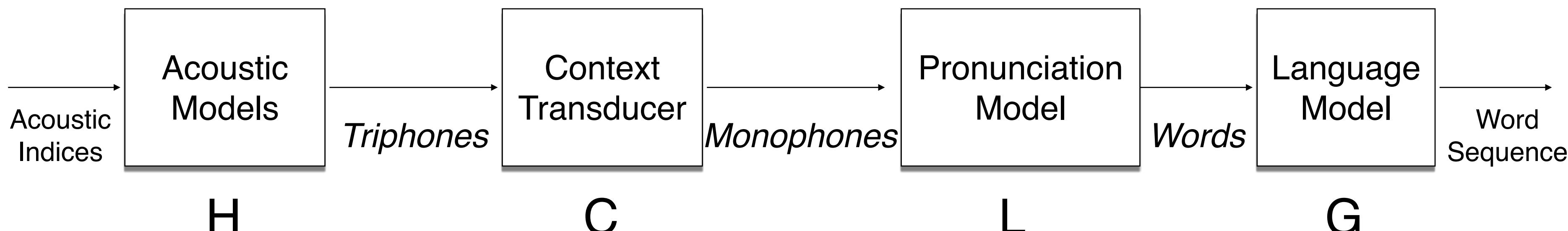


Carefully construct a decoding graph D using optimization algorithms:

$$D = \min(\det(H \circ \det(C \circ \det(L \circ G))))$$

Given a test utterance O , *how do I decode it?*

Decoding

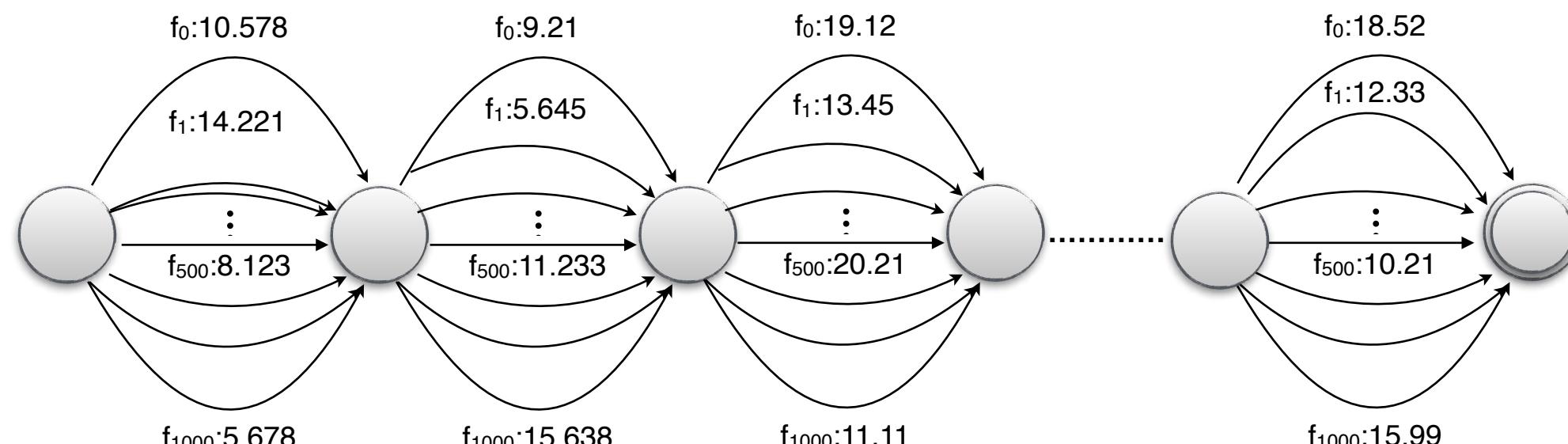


Carefully construct a decoding graph D using optimization algorithms:

$$D = \min(\det(H \circ \det(C \circ \det(L \circ G))))$$

Given a test utterance O , how do I decode it?

Assuming ample compute, first construct the following machine X from O .



X

$$W^* = \arg \min_{W=\text{out}[\pi]} X \circ D$$

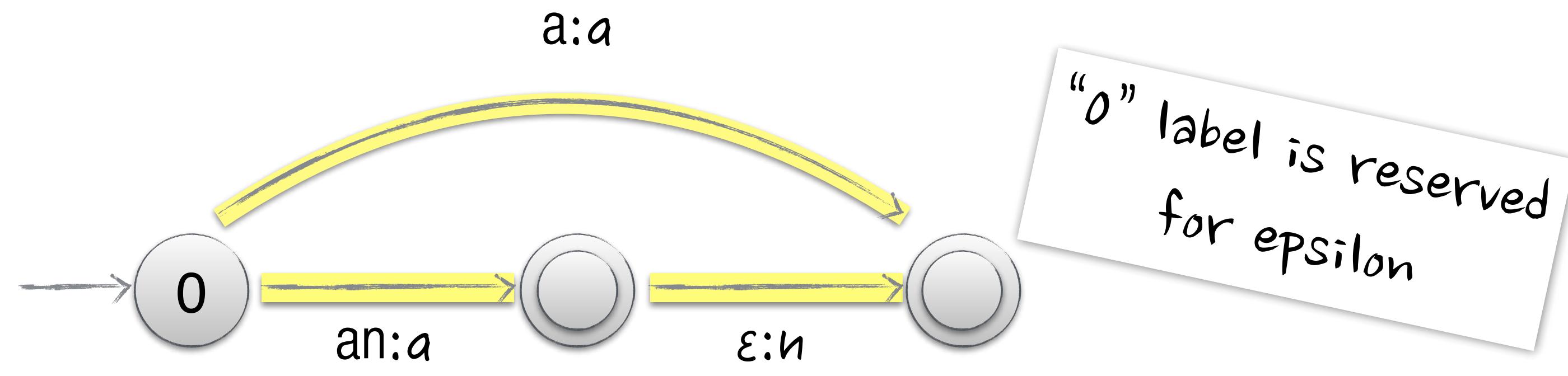
where π is a path in the composed FST
 $\text{out}[\pi]$ is the output label sequence of π

X is never typically constructed;
 D is traversed dynamically using approximate search algorithms
 (discussed later in the semester)

Toolkits to work with finite-state machines

- AT&T FSM Library (no longer supported)
<http://www3.cs.stonybrook.edu/~algorith/implement/fsm/implement.shtml>
- RWTH FSA Toolkit
<https://www-i6.informatik.rwth-aachen.de/~kanthak/fsa.html>
- Carmel
<https://www.isi.edu/licensed-sw/carmel/>
- MIT FST Toolkit
<http://people.csail.mit.edu/ilh/fst/>
- OpenFST Toolkit (actively supported, used by Kaldi)
<http://www.openfst.org/twiki/bin/view/FST/WebHome>

Quick Intro to OpenFst (www.openfst.org)



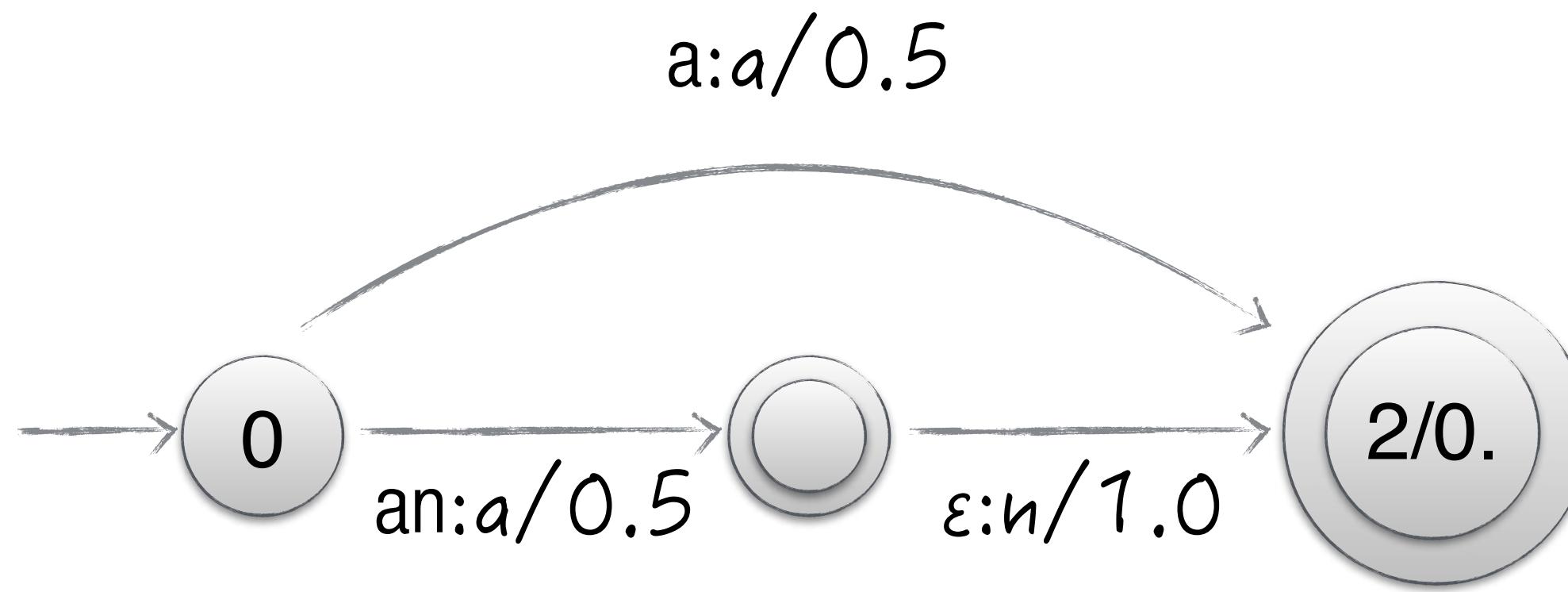
0	1	an	a
1	2	<eps>	n
0	2	a	a
1			
2			

A.txt

<eps>	0	Input alphabet (in.txt)
an	1	
a	2	

<eps>	0	Output alphabet (out.txt)
a	1	
n	2	

Quick Intro to OpenFst (www.openfst.org)



0	1	an	a	0.5
1	2	<eps>	n	1.0
0	2	a	a	0.5
1				
2	0.1			

Compiling & Printing FSTs

The text FSTs need to be “compiled” into binary objects before further use with OpenFst utilities

- Command used to compile:

```
fstcompile --isymbols=in.txt --osymbols=out.txt A.txt A.fst
```

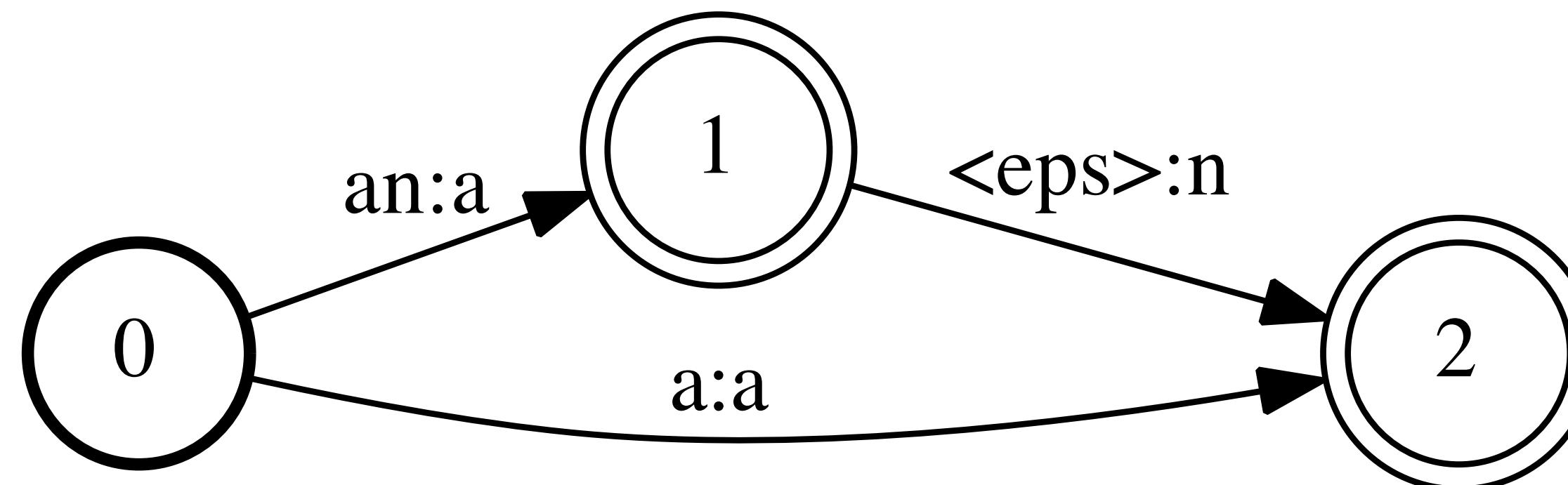
- Get back the text FST using a print command with the binary file:

```
fstprint --isymbols=in.txt --osymbols=out.txt A.fst A.txt
```

Drawing FSTs

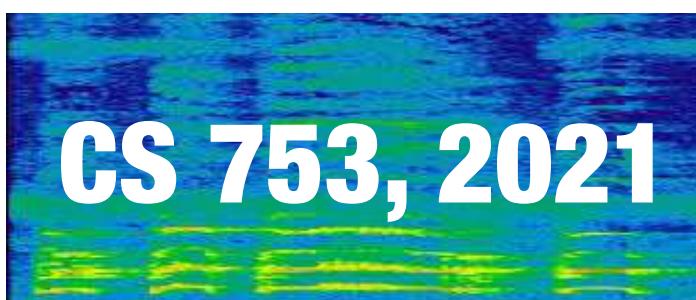
Small FSTs can be visualized easily using the draw tool:

```
fstdraw --isymbols=in.txt --osymbols=out.txt A.fst |  
dot -Tpdf > A.pdf
```



Language Models (I)

Lecture 4c



Instructor: Preethi Jyothi, IITB

Next, language models



- Language models
 - provide information about word reordering

$$\Pr(\text{"she class taught a"}) < \Pr(\text{"she taught a class"})$$

- provide information about the most likely next word

$$\Pr(\text{"she taught a class"}) > \Pr(\text{"she taught a speech"})$$

Application of language models

- Speech recognition
 - $\Pr(\text{"she taught a class"}) > \Pr(\text{"sheet or tuck lass"})$
- Machine translation
- Handwriting recognition/Optical character recognition
- Spelling correction of sentences
- Summarization, dialog generation, information retrieval, etc.

Popular Language Modelling Toolkits

- SRILM Toolkit:

<http://www.speech.sri.com/projects/srilm/>

- KenLM Toolkit:

<https://kheafield.com/code/kenlm/>

- OpenGrm NGram Library:

<http://opengrm.org/>

Introduction to probabilistic LMs

Probabilistic or Statistical Language Models

- Given a word sequence, $W = \{w_1, \dots, w_n\}$, what is $\Pr(W)$?
- Decompose $\Pr(W)$ using the chain rule:

$$\Pr(w_1, w_2, \dots, w_{n-1}, w_n) = \Pr(w_1) \Pr(w_2|w_1) \Pr(w_3|w_1, w_2) \dots \Pr(w_n|w_1, \dots, w_{n-1})$$

- Sparse data with long word contexts: How do we estimate the probabilities $\Pr(w_n|w_1, \dots, w_{n-1})$?

Estimating word probabilities

- Accumulate counts of words and word contexts
- Compute normalised counts to get next-word probabilities
- E.g. $\text{Pr}(\text{"class I she taught a"})$
$$= \frac{\pi(\text{"she taught a class"})}{\pi(\text{"she taught a"})}$$
where $\pi(\dots)$ refers to counts derived from a large English text corpus
- What is the obvious limitation here? We'll never see enough data

Simplifying Markov Assumption

- Markov chain:
 - Limited memory of previous word history: Only last m words are included
 - 1-order language model (or bigram model)
$$\Pr(w_1, w_2, \dots, w_{n-1}, w_n) \approx \Pr(w_1 | < s >) \Pr(w_2 | w_1) \Pr(w_3 | w_2) \dots \Pr(w_n | w_{n-1})$$
 - 2-order language model (or trigram model)
$$\Pr(w_1, w_2, \dots, w_{n-1}, w_n) \approx \Pr(w_2 | w_1, < s >) \Pr(w_3 | w_1, w_2) \dots \Pr(w_n | w_{n-2}, w_{n-1})$$
 - Ngram model is an N-1th order Markov model

Estimating Ngram Probabilities

- Maximum Likelihood Estimates
 - Unigram model

$$\Pr_{ML}(w_1) = \frac{\pi(w_1)}{\sum_i \pi(w_i)}$$

- Bigram model

$$\Pr_{ML}(w_2|w_1) = \frac{\pi(w_1, w_2)}{\sum_i \pi(w_1, w_i)}$$

Example

The dog chased a cat
The cat chased away a mouse
The mouse eats cheese

What is $\Pr(\text{"The cat chased a mouse"})$ using a bigram model?

$\Pr(\text{"<s> The cat chased a mouse </s>"}) =$

$\Pr(\text{"The|<s>"}) \cdot \Pr(\text{"cat|The"}) \cdot \Pr(\text{"chased|cat"}) \cdot \Pr(\text{"a|chased"}) \cdot \Pr(\text{"mouse|a"}) \cdot \Pr(\text{"</s>|mouse"}) =$

$$3/3 \cdot 1/3 \cdot 1/2 \cdot 1/2 \cdot 1/2 \cdot 1/2 = 1/48$$

Example

The dog chased a cat
The cat chased away a mouse
The mouse eats cheese

What is $\Pr(\text{"The dog eats cheese"})$ using a bigram model?

$\Pr(\text{"<s> The dog eats cheese </s>"}) =$

$\Pr(\text{"The|<s>"}) \cdot \Pr(\text{"dog|The"}) \cdot \Pr(\text{"eats|dog"}) \cdot \Pr(\text{"cheese|eats"}) \cdot \Pr(\text{"</s>|cheese"}) =$

$3/3 \cdot 1/3 \cdot 0/1 \cdot 1/1 \cdot 1/1 = 0!$ Due to unseen bigrams

How do we deal with unseen bigrams? We'll come back to it.

Open vs. closed vocabulary task

- Closed vocabulary task: Use a fixed vocabulary, V . We know all the words in advance.
- More realistic setting, we don't know all the words in advance. Open vocabulary task. Encounter out-of-vocabulary (OOV) words during test time.
- Create an unknown word: <UNK>
 - Estimating <UNK> probabilities: Determine a vocabulary V . Change all words in the training set not in V to <UNK>
 - Now train its probabilities like a regular word
 - At test time, use <UNK> probabilities for words not in training

Evaluating Language Models

- Extrinsic evaluation:
 - To compare Ngram models A and B, use both within a specific speech recognition system (keeping all other components the same)
 - Compare word error rates (WERs) for A and B
 - Time-consuming process!

Intrinsic Evaluation

- Evaluate the language model in a standalone manner
- How likely does the model consider the text in a test set?
- How closely does the model approximate the actual (test set) distribution?
- Same measure can be used to address both questions – perplexity!

Measures of LM quality

- How likely does the model consider the text in a test set?
- How closely does the model approximate the actual (test set) distribution?
- Same measure can be used to address both questions – perplexity!

Perplexity (I)

- How likely does the model consider the text in a test set?
 - $\text{Perplexity}(\text{test}) = 1/\Pr_{\text{model}}[\text{text}]$
 - Normalized by text length:
 - $\text{Perplexity}(\text{test}) = (1/\Pr_{\text{model}}[\text{text}])^{1/N}$ where $N = \text{number of tokens in test}$
 - e.g. If model predicts i.i.d. words from a dictionary of size L , per word perplexity = $(1/(1/L)^N)^{1/N} = L$

Intuition for Perplexity

- Shannon's guessing game builds intuition for perplexity
 - What is the surprisal factor in predicting the next word?

- At the stall, I had tea and _____
 - biscuits* 0.1
 - samosa* 0.1
 - coffee* 0.01
 - rice* 0.001
 - :
 - tree* 0.00000001

- A better language model would assign a higher probability to the actual word that fills the blank (and hence lead to lesser surprisal/perplexity)

Measures of LM quality

- How likely does the model consider the text in a test set?
- How closely does the model approximate the actual (test set) distribution?
- Same measure can be used to address both questions – perplexity!

Perplexity (II)

- How closely does the model approximate the actual (test set) distribution?
 - KL-divergence between two distributions X and Y
$$D_{KL}(X||Y) = \sum_{\sigma} Pr_X[\sigma] \log (Pr_X[\sigma]/Pr_Y[\sigma])$$
 - Equals zero iff $X = Y$; Otherwise, positive
- How to measure $D_{KL}(X||Y)$? We don't know X!
 - $D_{KL}(X||Y) = \sum_{\sigma} Pr_X[\sigma] \log(1/Pr_Y[\sigma]) - H(X)$ where $H(X) = -\sum_{\sigma} Pr_X[\sigma] \log Pr_X[\sigma]$
 - Empirical cross entropy:

$$\frac{1}{|test|} \sum_{\sigma \in test} \log\left(\frac{1}{Pr_y[\sigma]}\right)$$

Cross entropy
between X and Y

Perplexity vs. Empirical Cross Entropy

- Empirical Cross Entropy (ECE)

$$\frac{1}{|\#sents|} \sum_{\sigma \in test} \log\left(\frac{1}{\Pr_{model}[\sigma]}\right)$$

- Normalized Empirical Cross Entropy = ECE/(avg. length) =

$$\begin{aligned} & \frac{1}{|\#words/\#sents|} \frac{1}{|\#sents|} \sum_{\sigma \in test} \log\left(\frac{1}{\Pr_{model}[\sigma]}\right) \\ &= \frac{1}{N} \sum_{\sigma} \log\left(\frac{1}{\Pr_{model}[\sigma]}\right) \end{aligned}$$

where $N = \#words$

- How does $\frac{1}{N} \sum_{\sigma} \log\left(\frac{1}{\Pr_{model}[\sigma]}\right)$ relate to perplexity?

Perplexity vs. Empirical Cross Entropy

$$\begin{aligned}\log(\text{perplexity}) &= \frac{1}{N} \log \frac{1}{\Pr[\text{test}]} \\ &= \frac{1}{N} \log \prod_{\sigma} \left(\frac{1}{\Pr_{\text{model}}[\sigma]} \right) \\ &= \frac{1}{N} \sum_{\sigma} \log \left(\frac{1}{\Pr_{\text{model}}[\sigma]} \right)\end{aligned}$$

Thus, perplexity = $\exp^{(\text{normalized cross entropy})}$

Example perplexities for Ngram models trained on WSJ (80M words):

Unigram: 962, Bigram: 170, Trigram: 109

Introduction to smoothing of LMs

Recall example

The dog chased a cat
The cat chased away a mouse
The mouse eats cheese

What is $\text{Pr}(\text{"The dog eats cheese"})$?

$\text{Pr}(\text{"<S> The dog eats cheese </S>"}) =$

$\text{Pr}(\text{"The|<S>"}) \cdot \text{Pr}(\text{"dog|The"}) \cdot \text{Pr}(\text{"eats|dog"}) \cdot \text{Pr}(\text{"cheese|eats"}) \cdot \text{Pr}(\text{"</S>|cheese"}) =$

$3/3 \cdot 1/3 \cdot 0/1 \cdot 1/1 \cdot 1/1 = 0!$ Due to unseen bigrams

Unseen Ngrams

- Even with MLE estimates based on counts from large text corpora, there will be many unseen bigrams/trigrams that never appear in the corpus
- If any unseen Ngram appears in a test sentence, the sentence will be assigned probability 0
- Problem with MLE estimates: maximises the likelihood of the observed data by assuming anything unseen cannot happen and overfits to the training data
 - **Smoothing methods:** Reserve some probability mass to Ngrams that don't occur in the training corpus

Add-one (Laplace) smoothing

Simple idea: Add one to all bigram counts. That means,

$$\Pr_{ML}(w_i|w_{i-1}) = \frac{\pi(w_{i-1}, w_i)}{\pi(w_{i-1})}$$

becomes

$$\Pr_{Lap}(w_i|w_{i-1}) = \frac{\pi(w_{i-1}, w_i) + 1}{\pi(w_{i-1}) + V} \quad \checkmark$$

where V is the vocabulary size

Example: Bigram counts

**No
smoothing**

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

**Laplace
(Add-one)
smoothing**

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Example: Bigram probabilities

**No
smoothing**

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

**Laplace
(Add-one)
smoothing**

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Laplace smoothing moves too much probability mass to unseen events!

Add- α Smoothing

Instead of 1, add $\alpha < 1$ to each count

$$\Pr_{\alpha}(w_i|w_{i-1}) = \frac{\pi(w_{i-1}, w_i) + \alpha}{\pi(w_{i-1}) + \alpha V}$$

Choosing α :

- Train model on training set using different values of α
- Choose the value of α that minimizes cross entropy on the development set

Smoothing or discounting

- Smoothing can be viewed as **discounting** (lowering) some probability mass from seen Ngrams and redistributing discounted mass to unseen events
- i.e. probability of a bigram with Laplace smoothing

$$\Pr_{Lap}(w_i|w_{i-1}) = \frac{\pi(w_{i-1}, w_i) + 1}{\pi(w_{i-1}) + V}$$

- can be written as

$$\Pr_{Lap}(w_i|w_{i-1}) = \frac{\pi^*(w_{i-1}, w_i)}{\pi(w_{i-1})}$$

- where discounted count $\pi^*(w_{i-1}, w_i) = (\pi(w_{i-1}, w_i) + 1) \frac{\pi(w_{i-1})}{\pi(w_{i-1}) + V}$

Example: Bigram adjusted counts

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Smoothing or discounting

- Smoothing can be viewed as **discounting** (lowering) some probability mass from seen Ngrams and redistributing discounted mass to unseen events
- i.e. probability of a bigram with Laplace smoothing

$$\Pr_{Lap}(w_i|w_{i-1}) = \frac{\pi(w_{i-1}, w_i) + 1}{\pi(w_{i-1}) + V}$$

- can be written as

$$\Pr_{Lap}(w_i|w_{i-1}) = \frac{\pi^*(w_{i-1}, w_i)}{\pi(w_{i-1})}$$

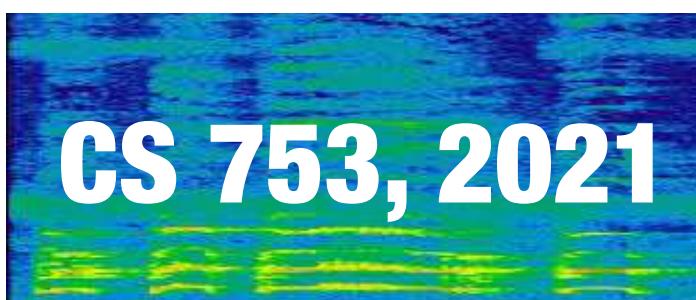
- where discounted count $\pi^*(w_{i-1}, w_i) = (\pi(w_{i-1}, w_i) + 1) \frac{\pi(w_{i-1})}{\pi(w_{i-1}) + V}$
- More sophisticated smoothing/discounting schemes in the next lecture!

Ngram models as WFSAs

- With no optimizations, an Ngram over a vocabulary of V words defines a WFSA with V^{N-1} states and V^N edges.
- Example: Consider a trigram model for a two-word vocabulary, A B.
 - 4 states representing bigram histories, A_A, A_B, B_A, B_B
 - 8 arcs transitioning between these states
- Clearly not practical when V is large.
- Resort to backoff language models (More in the next lecture)

Language Models (II)

Lecture 5a



Instructor: Preethi Jyothi, IITB

Unseen Ngrams

- By using estimates based on counts from large text corpora, there will still be many unseen bigrams/trigrams at test time that never appear in the training corpus
- If any unseen Ngram appears in a test sentence, the sentence will be assigned probability 0
- Problem with MLE estimates: Maximises the likelihood of the observed data by assuming anything unseen cannot happen and overfits to the training data
 - **Smoothing methods:** Reserve some probability mass to Ngrams that don't occur in the training corpus

Add-one (Laplace) smoothing

Simple idea: Add one to all bigram counts. That means,

$$\Pr_{ML}(w_i|w_{i-1}) = \frac{\pi(w_{i-1}, w_i)}{\pi(w_{i-1})}$$

becomes

$$\Pr_{Lap}(w_i|w_{i-1}) = \frac{\pi(w_{i-1}, w_i) + 1}{\pi(w_{i-1}) + V}$$

where V is the vocabulary size

Example: Bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Example: Bigram probabilities

**No
smoothing**

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

**Laplace
(Add-one)
smoothing**

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Laplace smoothing moves too much probability mass to unseen events!

Add- α Smoothing

Instead of 1, add $\alpha < 1$ to each count

$$\Pr_{\alpha}(w_i|w_{i-1}) = \frac{\pi(w_{i-1}, w_i) + \alpha}{\pi(w_{i-1}) + \alpha V}$$

Choosing α :

- Train model on training set using different values of α
- Choose the value of α that minimizes cross entropy on the development set

Smoothing or discounting

- Smoothing can be viewed as **discounting** (lowering) some probability mass from seen Ngrams and redistributing discounted mass to unseen events
- i.e. probability of a bigram with Laplace smoothing

$$\Pr_{Lap}(w_i|w_{i-1}) = \frac{\pi(w_{i-1}, w_i) + 1}{\pi(w_{i-1}) + V}$$

- can be written as
- where discounted count $\pi^*(w_{i-1}, w_i) = (\pi(w_{i-1}, w_i) + 1) \frac{\pi(w_{i-1})}{\pi(w_{i-1}) + V}$

Example: Bigram adjusted counts

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Problems with Add- α Smoothing

- What's wrong with add-a smoothing?
- Assigns too much probability mass away from seen Ngrams to unseen events
- Does not discount high counts and low counts correctly
- Also, α is tricky to set
- Is there a more principled way to do this smoothing?
A solution: Good-Turing estimation

Advanced Smoothing Techniques

- Good-Turing Discounting
- Backoff and Interpolation
 - Katz Backoff Smoothing
 - Absolute Discounting Interpolation
- Kneser-Ney Smoothing

Good-Turing estimation (uses held-out data)

r	N_r	r^* in heldout set	add-1 r^*
1	2×10^6	0.448	2.8×10^{-11}
2	4×10^5	1.25	4.2×10^{-11}
3	2×10^5	2.24	5.7×10^{-11}
4	1×10^5	3.23	7.1×10^{-11}
5	7×10^4	4.21	8.5×10^{-11}

r = Count in a large corpus & N_r is the number of bigrams with r counts
 r^* is estimated on a *different* held-out corpus

- Add-1 smoothing hugely overestimates fraction of unseen events
- Good-Turing estimation uses observed data to predict how to go from r to the heldout- r^*

Good-Turing Estimation

- Intuition for Good-Turing estimation using leave-one-out validation:
 - Let N_r be the number of words (tokens,bigrams,etc.) that occur r times
 - Split a given set of N word tokens into a training set of $(N-1)$ samples + 1 sample as the held-out set; repeat this process N times so that all N samples appear in the held-out set
 - In what fraction of these N trials is the held-out word unseen during training?
 N_1/N
 - In what fraction of these N trials is the held-out word seen exactly k times during training? $(k+1)N_{k+1}/N$
 - There are (\approx) N_k words with training count k .
 - Probability of each being chosen as held-out: $(k+1)N_{k+1}/(N \times N_k)$
 - Expected count of each of the N_k words in a corpus of size N : $k^* = \theta(k) = (k+1) N_{k+1}/N_k$

Good-Turing Estimates

r	N _r	r*-GT	r*-heldout
0	7.47×10^{10}	.0000270	.0000270
1	2×10^6	0.446	0.448
2	4×10^5	1.26	1.25
3	2×10^5	2.24	2.24
4	1×10^5	3.24	3.23
5	7×10^4	4.22	4.21
6	5×10^4	5.19	5.23
7	3.5×10^4	6.21	6.21
8	2.7×10^4	7.24	7.21
9	2.2×10^4	8.25	8.26

Table showing frequencies of bigrams from 0 to 9
In this example, for $r > 0$, $r^*\text{-GT} \approx r^*\text{-heldout}$ and $r^*\text{-GT}$ is always less than r

Good-Turing Smoothing

- Thus, Good-Turing smoothing states that for any Ngram that occurs r times, we should use an adjusted count $r^* = \theta(r) = (r + 1)N_{r+1}/N_r$
- Good-Turing smoothed counts for unseen events: $\theta(0) = N_1/N_0$
- Example: 10 bananas, 5 apples, 2 papayas, 1 melon, 1 guava, 1 pear
 - How likely are we to see a guava next? The GT estimate is $\theta(1)/N$
 - Here, $N = 20$, $N_2 = 1$, $N_1 = 3$. Computing $\theta(1)$: $\theta(1) = 2 \times 1/3 = 2/3$
 - Thus, $\text{Pr}_{\text{GT}}(\text{guava}) = \theta(1)/20 = 1/30 = 0.0333$

Good-Turing Estimation

- One issue: For large r , many instances of $N_{r+1} = 0$!
 - This would lead to $\theta(r) = (r + 1)N_{r+1}/N_r$ being set to 0.
- Solution: Discount only for small counts $r \leq k$ (e.g. $k = 9$) and $\theta(r) = r$ for $r > k$
- Another solution: Smooth N_r using a best-fit power law once counts start getting small
- Good-Turing smoothing tells us how to discount some probability mass to unseen events. Could we redistribute this mass across observed counts of lower-order Ngram events?

Advanced Smoothing Techniques

- Good-Turing Discounting
- Backoff and Interpolation
 - Katz Backoff Smoothing
 - Absolute Discounting Interpolation
- Kneser-Ney Smoothing

Backoff and Interpolation

- General idea: It helps to use lesser context to generalise for contexts that the model doesn't know enough about
- **Backoff:**
 - Use trigram probabilities if there is sufficient evidence
 - Else use bigram or unigram probabilities
- **Interpolation**
 - Mix probability estimates combining trigram, bigram and unigram counts

Interpolation

- Linear interpolation: Linear combination of different Ngram models

$$\begin{aligned}\hat{P}(w_n | w_{n-2} w_{n-1}) &= \lambda_1 P(w_n | w_{n-2} w_{n-1}) \\ &\quad + \lambda_2 P(w_n | w_{n-1}) \\ &\quad + \lambda_3 P(w_n)\end{aligned}$$

where $\lambda_1 + \lambda_2 + \lambda_3 = 1$

How to set the λ 's?

Interpolation

- Linear interpolation: Linear combination of different Ngram models

$$\begin{aligned}\hat{P}(w_n | w_{n-2} w_{n-1}) &= \lambda_1 P(w_n | w_{n-2} w_{n-1}) \\ &\quad + \lambda_2 P(w_n | w_{n-1}) \\ &\quad + \lambda_3 P(w_n)\end{aligned}$$

where $\lambda_1 + \lambda_2 + \lambda_3 = 1$

1. Estimate N-gram probabilities on a training set.
2. Then, search for λ 's that maximises the probability of a held-out set

Advanced Smoothing Techniques

- Good-Turing Discounting
- Backoff and Interpolation
 - Katz Backoff Smoothing
 - Absolute Discounting Interpolation
- Kneser-Ney Smoothing

Katz Smoothing

- Good-Turing discounting determines the volume of probability mass that is allocated to unseen events
- Katz Smoothing distributes this remaining mass proportionally across “smaller” Ngrams
 - i.e. no trigram found, use backoff probability of bigram and if no bigram found, use backoff probability of unigram

Katz Backoff Smoothing

- For a Katz bigram model, let us define:
 - $\Psi(w_{i-1}) = \{w: \pi(w_{i-1}, w) > 0\}$
- A bigram model with Katz smoothing can be written in terms of a unigram model as follows:

$$P_{\text{Katz}}(w_i | w_{i-1}) = \begin{cases} \frac{\pi^*(w_{i-1}, w_i)}{\pi(w_{i-1})} & \text{if } w_i \in \Psi(w_{i-1}) \\ \alpha(w_{i-1}) P_{\text{Katz}}(w_i) & \text{if } w_i \notin \Psi(w_{i-1}) \end{cases}$$

where $\alpha(w_{i-1}) = \frac{\left(1 - \sum_{w \in \Psi(w_{i-1})} \frac{\pi^*(w_{i-1}, w)}{\pi(w_{i-1})}\right)}{\sum_{w_i \notin \Psi(w_{i-1})} P_{\text{Katz}}(w_i)}$

Katz Backoff Smoothing

$$P_{\text{Katz}}(w_i | w_{i-1}) = \begin{cases} \frac{\pi^*(w_{i-1}, w_i)}{\pi(w_{i-1})} & \text{if } w_i \in \Psi(w_{i-1}) \\ \alpha(w_{i-1}) P_{\text{Katz}}(w_i) & \text{if } w_i \notin \Psi(w_{i-1}) \end{cases}$$

$$\text{where } \alpha(w_{i-1}) = \frac{\left(1 - \sum_{w \in \Psi(w_{i-1})} \frac{\pi^*(w_{i-1}, w)}{\pi(w_{i-1})}\right)}{\sum_{w_i \notin \Psi(w_{i-1})} P_{\text{Katz}}(w_i)}$$

- A bigram with a non-zero count is discounted using Good-Turing estimation
- The left-over probability mass from discounting for the unigram model ...
- ... is distributed over $w_i \notin \Psi(w_{i-1})$ proportionally to $P_{\text{Katz}}(w_i)$

Advanced Smoothing Techniques

- Good-Turing Discounting
- Backoff and Interpolation
 - Katz Backoff Smoothing
 - Absolute Discounting Interpolation
- Kneser-Ney Smoothing

Recall Good-Turing estimates

r	N_r	$\theta(r)$
0	7.47×10^{10}	.0000270
1	2×10^6	0.446
2	4×10^5	1.26
3	2×10^5	2.24
4	1×10^5	3.24
5	7×10^4	4.22
6	5×10^4	5.19
7	3.5×10^4	6.21
8	2.7×10^4	7.24
9	2.2×10^4	8.25

For $r > 0$, we observe that $\theta(r) \approx r - 0.75$ i.e. an absolute discounting

Absolute Discounting Interpolation

- Absolute discounting motivated by Good-Turing estimation
- Just subtract a constant d from the non-zero counts to get the discounted count
- Also involves linear interpolation with lower-order models

$$\Pr_{\text{abs}}(w_i | w_{i-1}) = \frac{\max\{\pi(w_{i-1}, w_i) - d, 0\}}{\pi(w_{i-1})} + \lambda(w_{i-1}) \Pr(w_i)$$

Advanced Smoothing Techniques

- Good-Turing Discounting
- Backoff and Interpolation
 - Katz Backoff Smoothing
 - Absolute Discounting Interpolation
- Kneser-Ney Smoothing

Kneser-Ney discounting

$$\Pr_{\text{KN}}(w_i|w_{i-1}) = \frac{\max\{\pi(w_{i-1}, w_i) - d, 0\}}{\pi(w_{i-1})} + \lambda_{\text{KN}}(w_{i-1}) \Pr_{\text{cont}}(w_i)$$

c.f., absolute discounting

$$\Pr_{\text{abs}}(w_i|w_{i-1}) = \frac{\max\{\pi(w_{i-1}, w_i) - d, 0\}}{\pi(w_{i-1})} + \lambda(w_{i-1}) \Pr(w_i)$$

Kneser-Ney discounting

$$\Pr_{\text{KN}}(w_i|w_{i-1}) = \frac{\max\{\pi(w_{i-1}, w_i) - d, 0\}}{\pi(w_{i-1})} + \lambda_{\text{KN}}(w_{i-1})\Pr_{\text{cont}}(w_i)$$

Consider an example: “*Today I cooked some yellow curry*”

Suppose $\pi(\text{yellow}, \text{curry}) = 0$. $\Pr_{\text{abs}}[w \mid \text{yellow}] = \lambda(\text{yellow})\Pr(w)$

Now, say $\Pr[\text{Francisco}] \gg \Pr[\text{curry}]$, as *San Francisco* is very common in our corpus.

But *Francisco* is not as common a “continuation” (follows only *San*) as *curry* is (*red curry*, *chicken curry*, *potato curry*, ...)

Moral: Should use probability of being a continuation!

c.f., absolute discounting

$$\Pr_{\text{abs}}(w_i|w_{i-1}) = \frac{\max\{\pi(w_{i-1}, w_i) - d, 0\}}{\pi(w_{i-1})} + \lambda(w_{i-1})\Pr(w_i)$$

Kneser-Ney discounting

$$\Pr_{\text{KN}}(w_i|w_{i-1}) = \frac{\max\{\pi(w_{i-1}, w_i) - d, 0\}}{\pi(w_{i-1})} + \lambda_{\text{KN}}(w_{i-1})\Pr_{\text{cont}}(w_i)$$

$$\Pr_{\text{cont}}(w_i) = \frac{|\Phi(w_i)|}{|B|} \quad \text{and} \quad \lambda_{\text{KN}}(w_{i-1}) = \frac{d}{\pi(w_{i-1})} |\Psi(w_{i-1})|$$

where $\Phi(w_i) = \{w_{i-1} : \pi(w_{i-1}, w_i) > 0\}$
 $B = \{(w_{i-1}, w_i) : \pi(w_{i-1}, w_i) > 0\}$

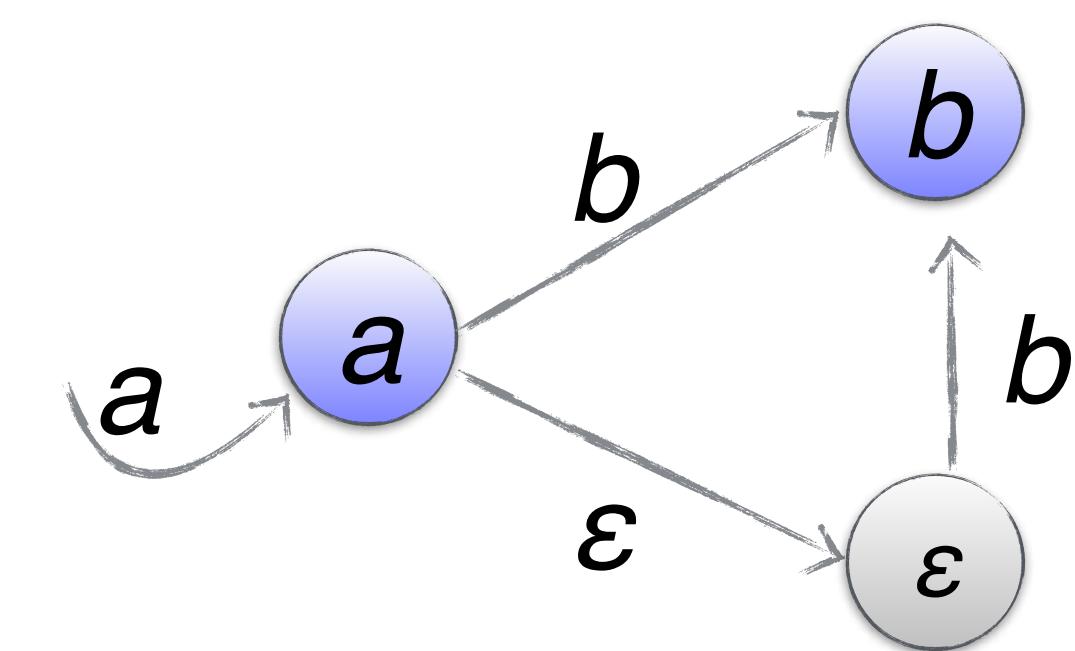
$$\frac{d \cdot |\Psi(w_{i-1})| \cdot |\Phi(w_i)|}{\pi(w_{i-1}) \cdot |B|}$$

c.f., absolute discounting

$$\Pr_{\text{abs}}(w_i|w_{i-1}) = \frac{\max\{\pi(w_{i-1}, w_i) - d, 0\}}{\pi(w_{i-1})} + \lambda(w_{i-1})\Pr(w_i)$$

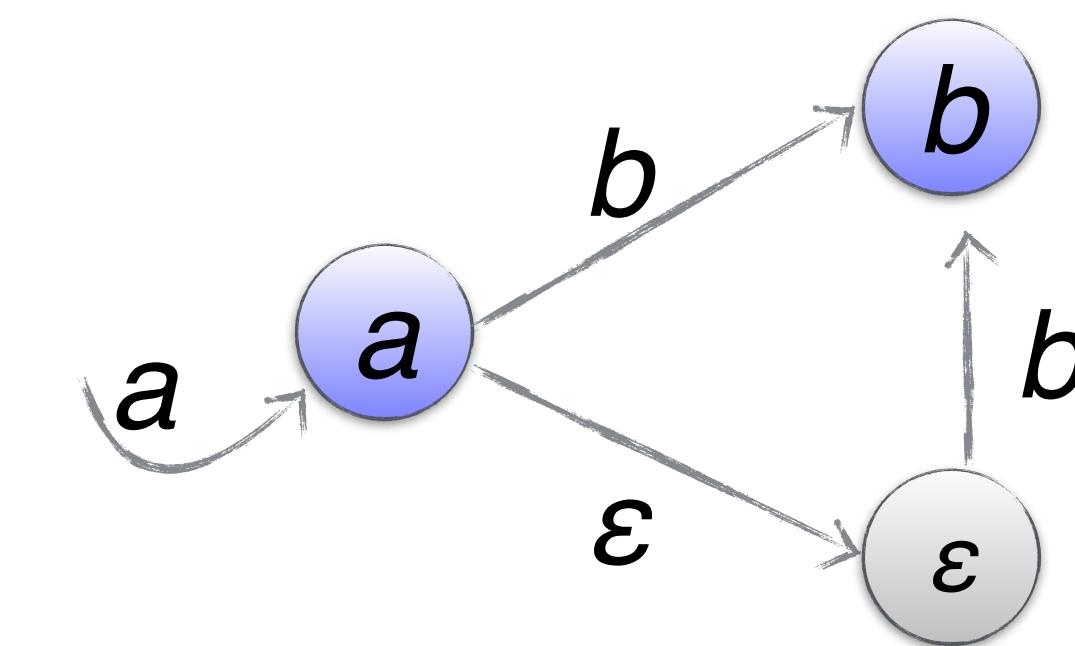
Kneser-Ney: An Alternate View

- A mix of bigram and unigram models
- A bigram ab could be generated in two ways:
 - In context a , output b , or
 - In context a , forget context and then output b (i.e., as “ $a\epsilon b$ ”)
- In a given set of bigrams, for each bigram ab , assume that d_{ab} of its occurrences were produced in the second way
- Will compute probabilities for each transition under this assumption



Kneser-Ney: An Alternate View

- Assuming $\pi(a,b) - d_{ab}$ occurrences as “ ab ”, and d_{ab} occurrences as “ $a\varepsilon b$ ”
- $\Pr[b|a] = [\pi(a,b) - d_{ab}] / \pi(a)$
 - $\Pr[\varepsilon |a] = [\sum_y d_{ay}] / \pi(a)$
 - $\Pr[b |\varepsilon] = [\sum_x d_{xb}] / [\sum_{xy} d_{xy}]$
 - $\Pr_{\text{KN}}[b | a] = \Pr[b|a] + \Pr[\varepsilon |a] \cdot \Pr[b |\varepsilon]$
- Kneser-Ney: Take $d_{xy} = d$ for all bigrams xy that do appear (assuming they all appear at least d times — kosher, e.g., if $d = 1$)
- Then $\sum_y d_{ay} = d \cdot |\Psi(a)|$, $\sum_x d_{xb} = d \cdot |\Phi(b)|$, and $\sum_{xy} d_{xy} = d \cdot |B|$ where $\Psi(a) = \{y : \pi(a,y) > 0\}$, $\Phi(b) = \{x : \pi(x,b) > 0\}$, $B = \{xy : \pi(x,y) > 0\}$

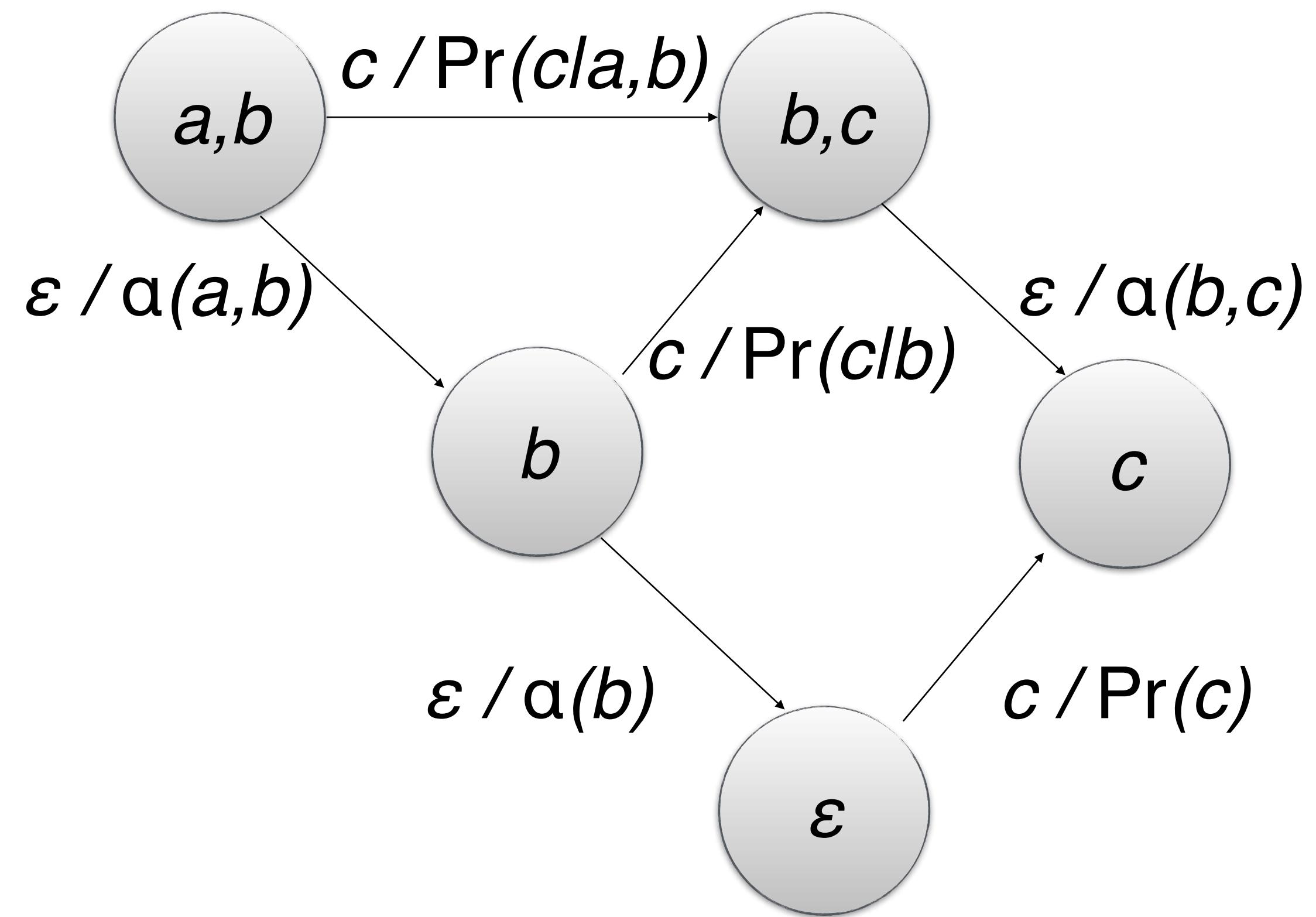


$$\Pr_{\text{KN}}(b|a) = \frac{\max\{\pi(a,b) - d, 0\}}{\pi(a)} + \frac{d \cdot |\Psi(a)| \cdot |\Phi(b)|}{\pi(a) \cdot |B|}$$

Ngram models as WFSAs

- With no optimizations, an Ngram over a vocabulary of V words defines a WFSA with V^{N-1} states and V^N edges.
- Example: Consider a trigram model for a two-word vocabulary, A B.
 - 4 states representing bigram histories, A_A, A_B, B_A, B_B
 - 8 arcs transitioning between these states
- Clearly not practical when V is large.
 - Resort to backoff language models

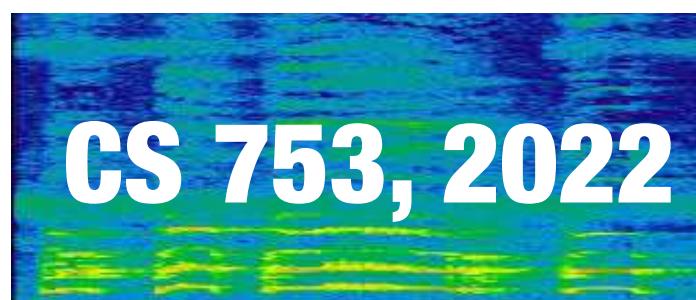
WFSA for backoff language model



Live Session

(Basics of Speech Production)

Lecture 5a



Instructor: Preethi Jyothi, IITB

Cascaded ASR Systems: Putting it all together

- A : speech utterance
- O_A : acoustic features corresponding to the utterance A

$$W^* = \arg \max_W \Pr(O_A|W) \Pr(W)$$

- Return the word sequence that jointly assigns the highest probability to O_A
- How do we estimate $\Pr(O_A|W)$ and $\Pr(W)$?
- How do we decode?

Acoustic Model

$$W^* = \arg \max_W \Pr(O_A|W) \Pr(W)$$

$$\begin{aligned} \Pr(O_A|W) &= \sum_Q \Pr(O_A, Q|W) \\ &= \sum_{q_1^T, w_1^N} \prod_{t=1}^T \Pr(O_t|O_1^{t-1}, q_1^t, w_1^N) \Pr(q_t|q_1^{t-1}, w_1^N) \end{aligned}$$

First-order HMM assumptions

$$\approx \sum_{q_1^T, w_1^N} \prod_{t=1}^T \Pr(O_t|q_t, w_1^N) \Pr(q_t|q_{t-1}, w_1^N)$$

Viterbi approximation

$$\approx \max_{q_1^T, w_1^N} \prod_{t=1}^T \Pr(O_t|q_t, w_1^N) \Pr(q_t|q_{t-1}, w_1^N)$$

Acoustic Model

$$\Pr(O_A|W) = \max_{q_1^T, w_1^N} \prod_{t=1}^T \Pr(O_t|q_t, w_1^N) \Pr(q_t|q_{t-1}, w_1^N)$$

Modeled using a mixture of Gaussians

$$\Pr(O_t|q_t) = \sum_{m=1}^M c_{qm} \mathcal{N}(O|\mu_{qm}, \Sigma_{qm})$$

Transition probabilities

Emission probabilities

Language Model

$$W^* = \arg \max_W \Pr(O_A|W) \Pr(W)$$

$$\begin{aligned}\Pr(W) &= \Pr(w_1, w_2, \dots, w_N) \\ &= \Pr(w_1) \dots \Pr(w_N | w_{N-m+1}^{N-1})\end{aligned}$$

m-gram language model

- Further optimized using smoothing and interpolation with lower-order Ngram models

Decoding

$$W^* = \arg \max_W \Pr(O_A|W) \Pr(W)$$

$$W^* = \arg \max_{w_1^N, N} \left\{ \left[\prod_{n=1}^N \Pr(w_n | w_{n-m+1}^{n-1}) \right] \left[\sum_{q_1^T, w_1^N} \prod_{t=1}^T \Pr(O_t | q_t, w_1^N) \Pr(q_t | q_{t-1}, w_1^N) \right] \right\}$$

$$\text{Viterbi} \approx \arg \max_{w_1^N, N} \left\{ \left[\prod_{n=1}^N \Pr(w_n | w_{n-m+1}^{n-1}) \right] \left[\max_{q_1^T, w_1^N} \prod_{t=1}^T \Pr(O_t | q_t, w_1^N) \Pr(q_t | q_{t-1}, w_1^N) \right] \right\}$$

- Search space still very huge for LVCSR tasks! Use approximate decoding techniques (A* decoding, beam-width decoding, etc.) to visit only promising parts of the search space

How are ASR systems evaluated?

- Word/Phone error rate (ER) uses the Levenshtein distance measure: What are the minimum number of edits (insertions/deletions/substitutions) required to convert W^* to W_{ref} ?

On a test set with N instances:

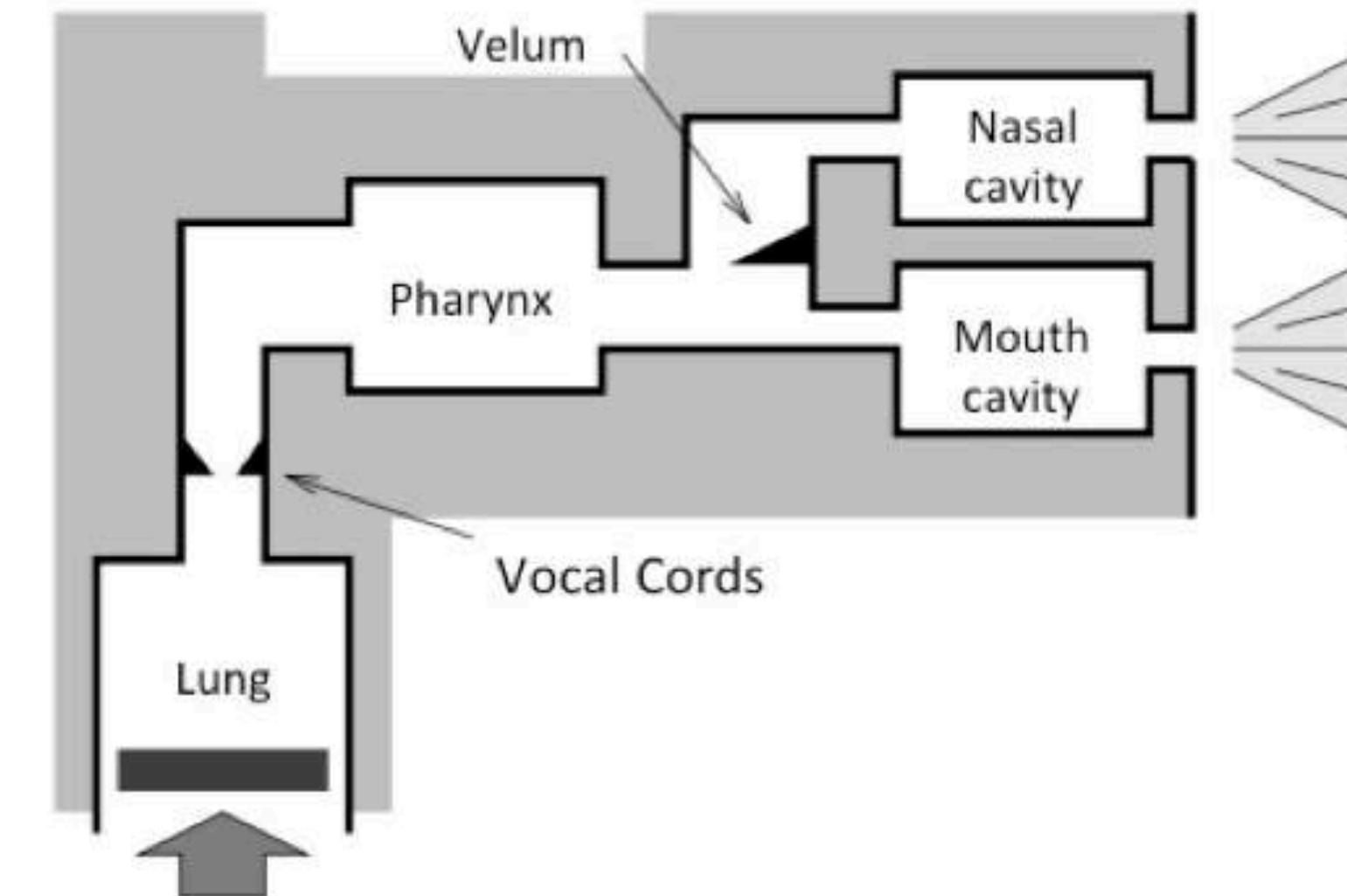
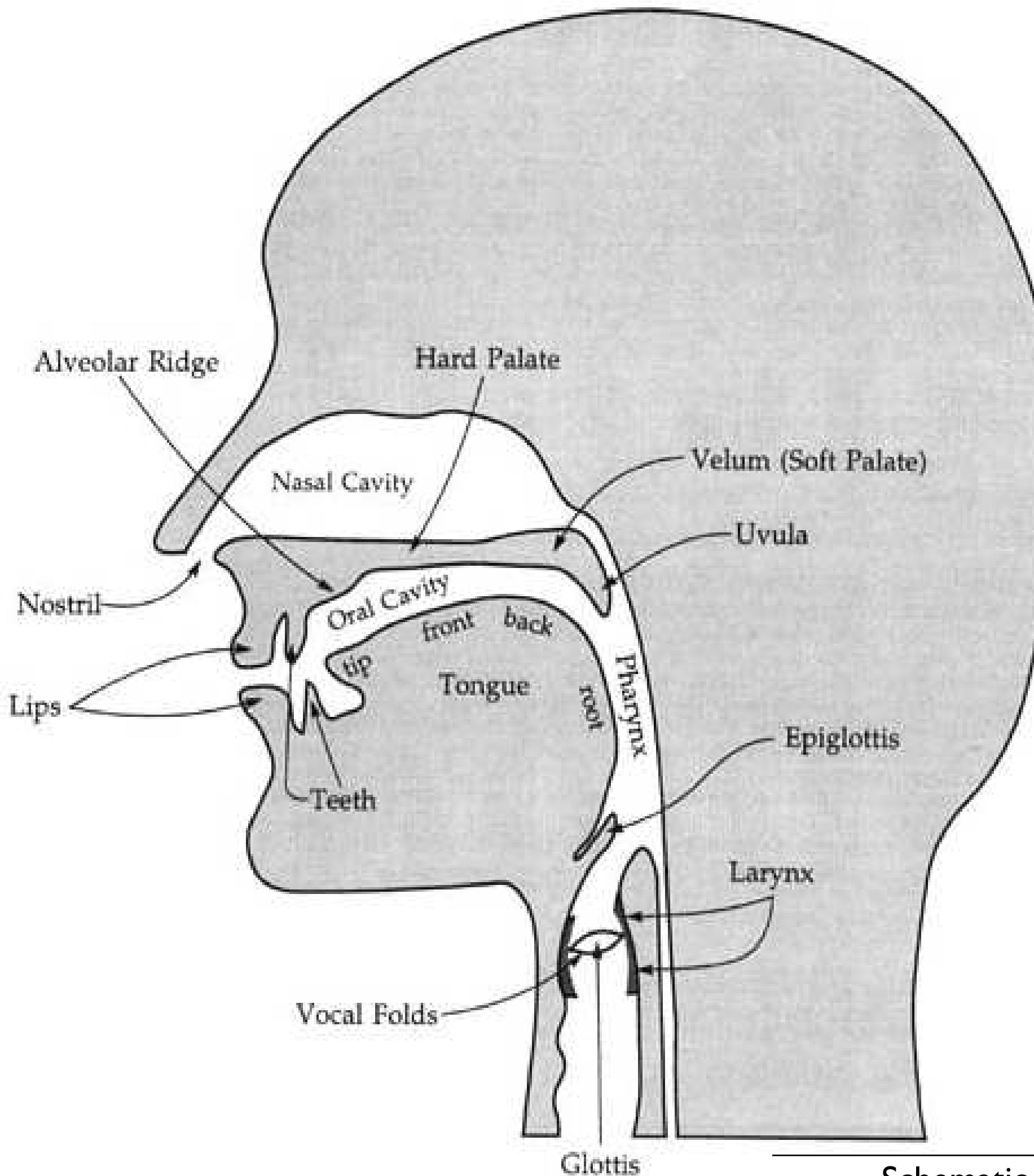
$$ER = \frac{\sum_{j=1}^N Ins_j + Del_j + Sub_j}{\sum_{j=1}^N \ell_j}$$

Ins_j , Del_j , Sub_j are number of insertions/deletions/substitutions in the j^{th} ASR output

ℓ_j is the total number of words/phones in the j^{th} reference

Basics of Speech Production

Speech Production

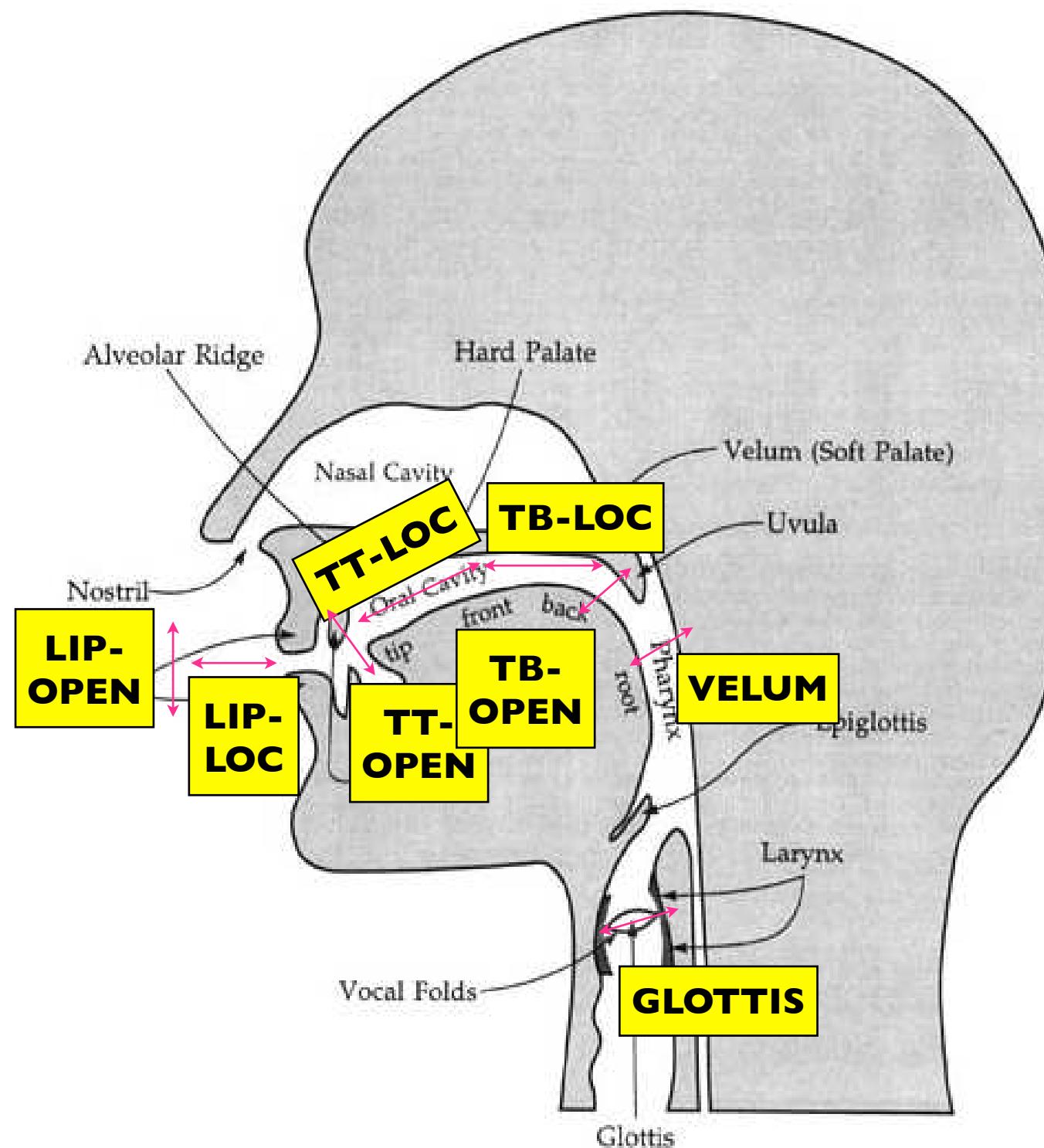


Schematic representation of the vocal organs

Schematic from L.Rabiner and B.-H.Juang , Fundamentals of speech recognition, 1993

Figure from <http://www.phon.ucl.ac.uk/courses/spsci/iss/week6.php>

Pronunciation Model



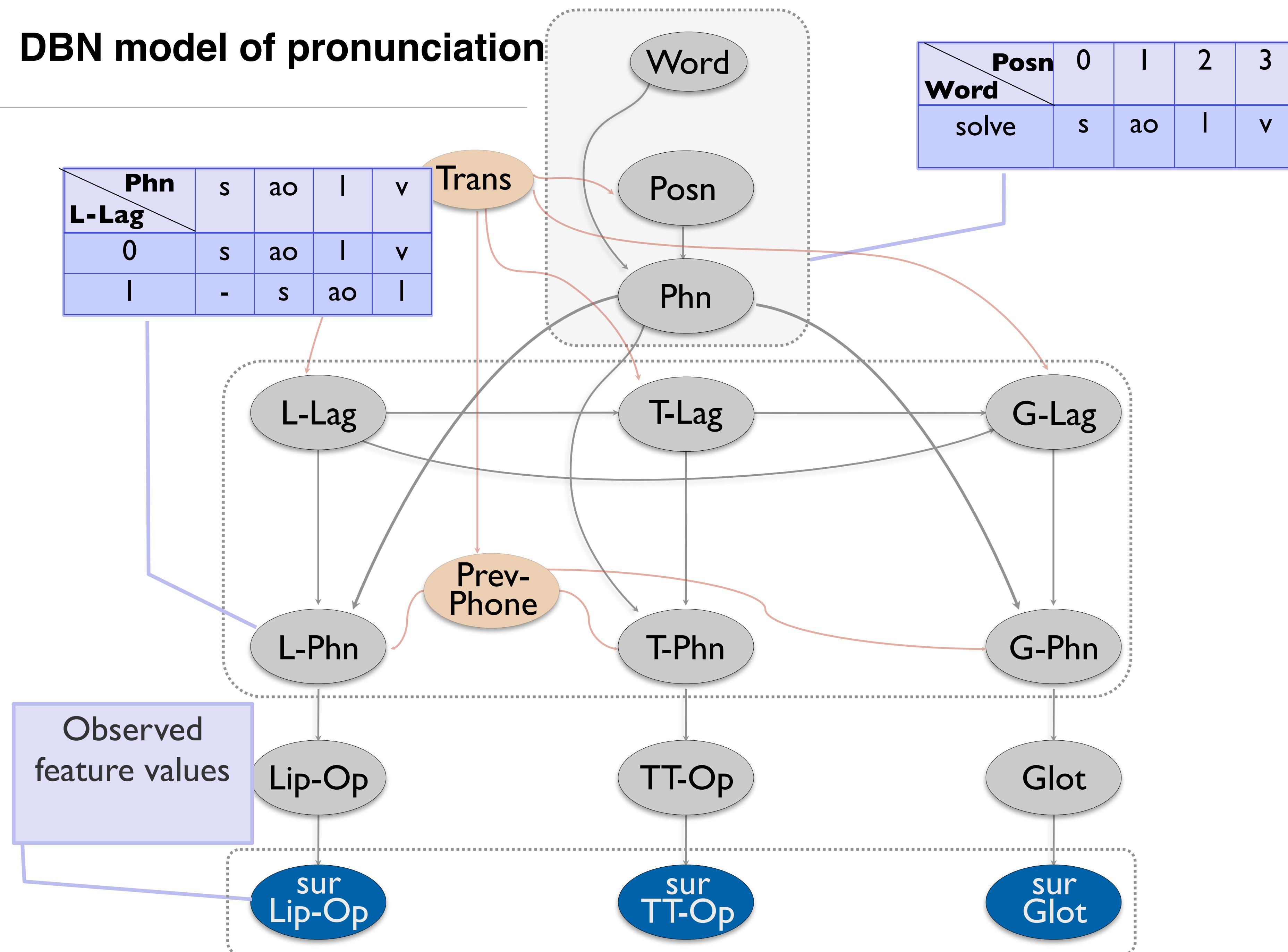
Articulatory Features

Parallel streams of articulator movements

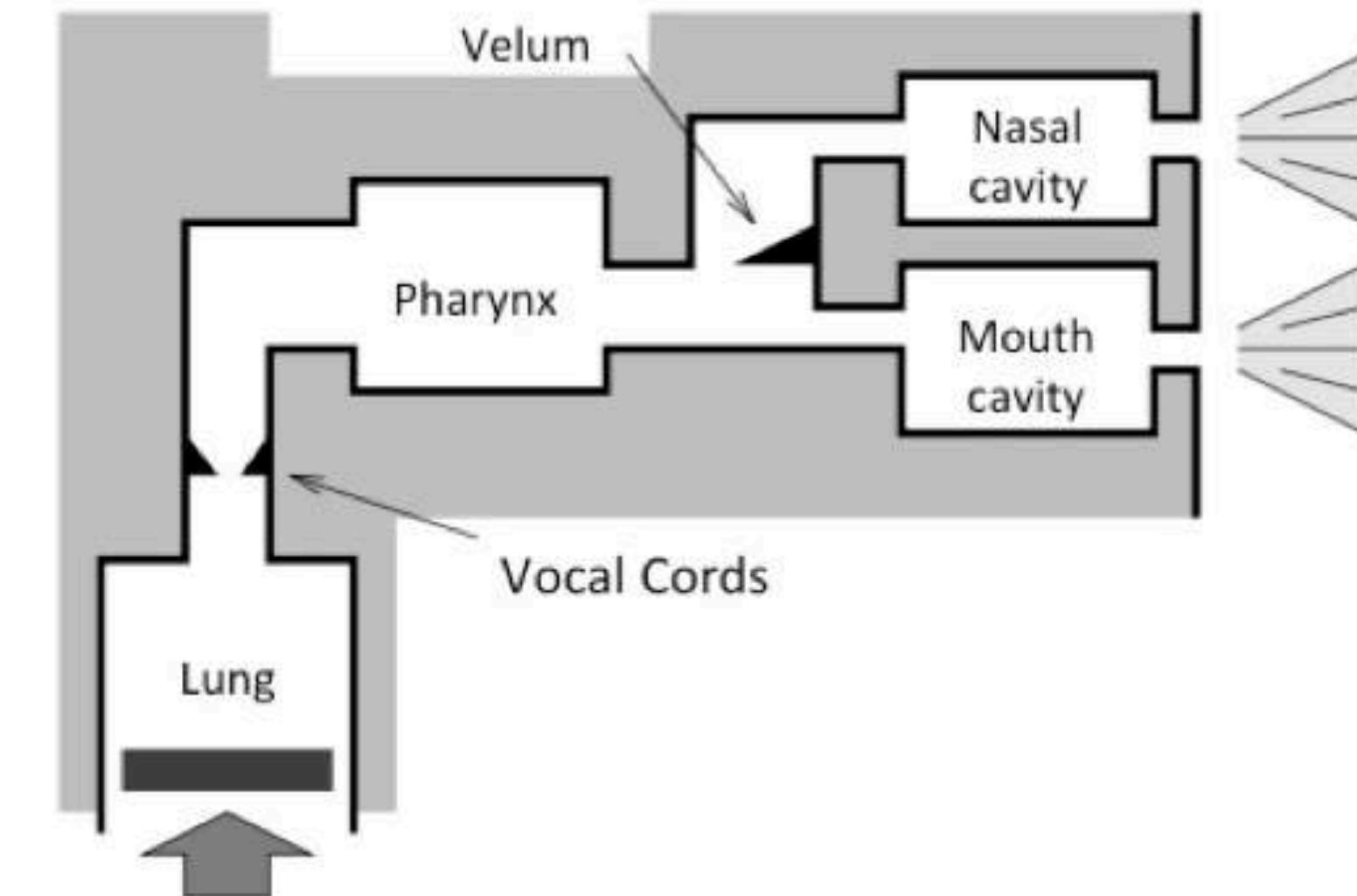
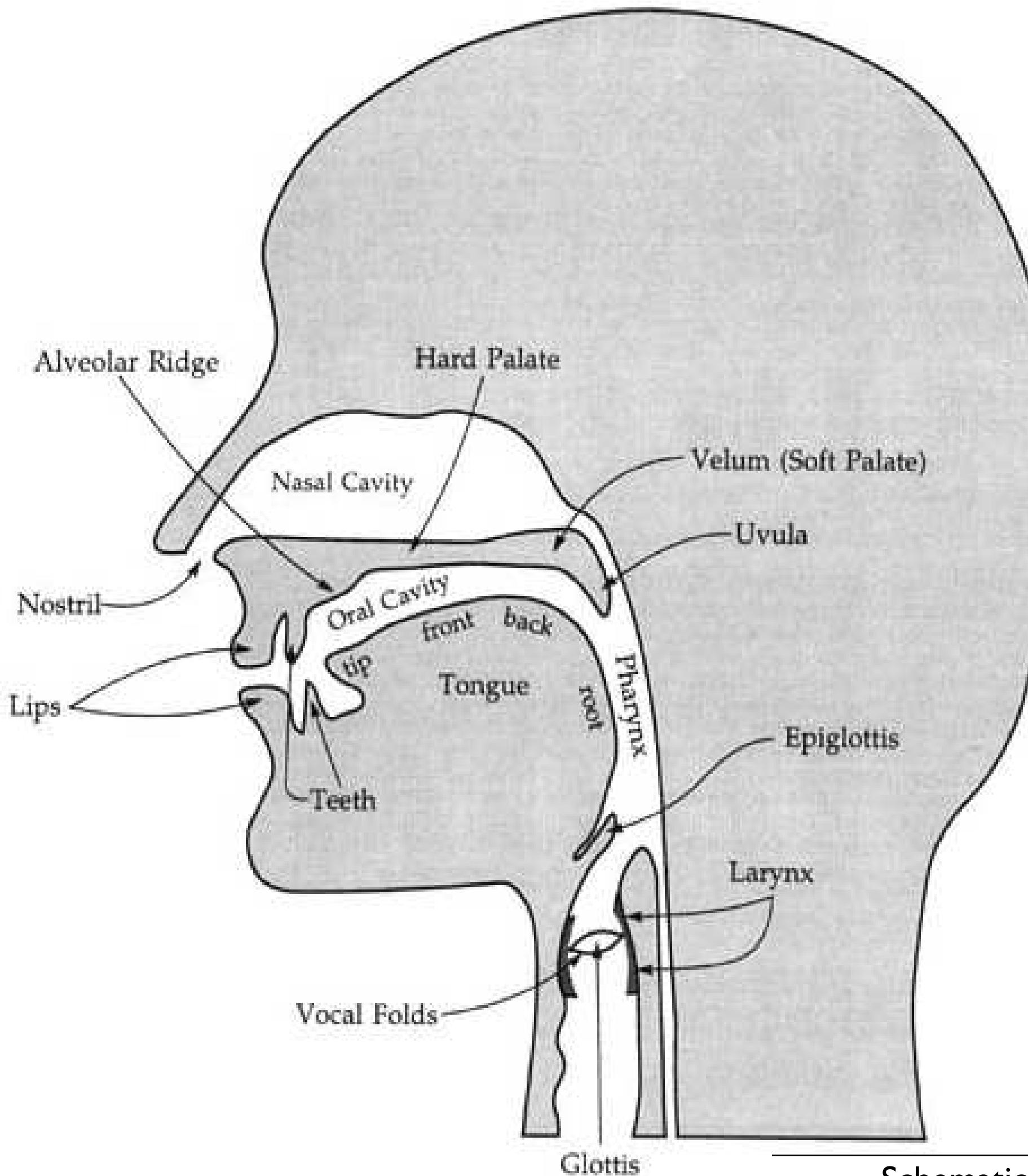
Based on theory of articulatory phonology

PHONE	s	eh	n	s
LIP	open/labial			
TON.TIP	critical/alveolar	mid/alveolar	closed/alveolar	critical/alveolar
TON.BODY	mid/uvular	mid/palatal	mid/uvular	
GLOTTIS	open	critical		open
VELUM	closed		open	closed

DBN model of pronunciation



Speech Production



Schematic representation of the vocal organs

Schematic from L.Rabiner and B.-H.Juang , Fundamentals of speech recognition, 1993

Figure from <http://www.phon.ucl.ac.uk/courses/spsci/iss/week6.php>

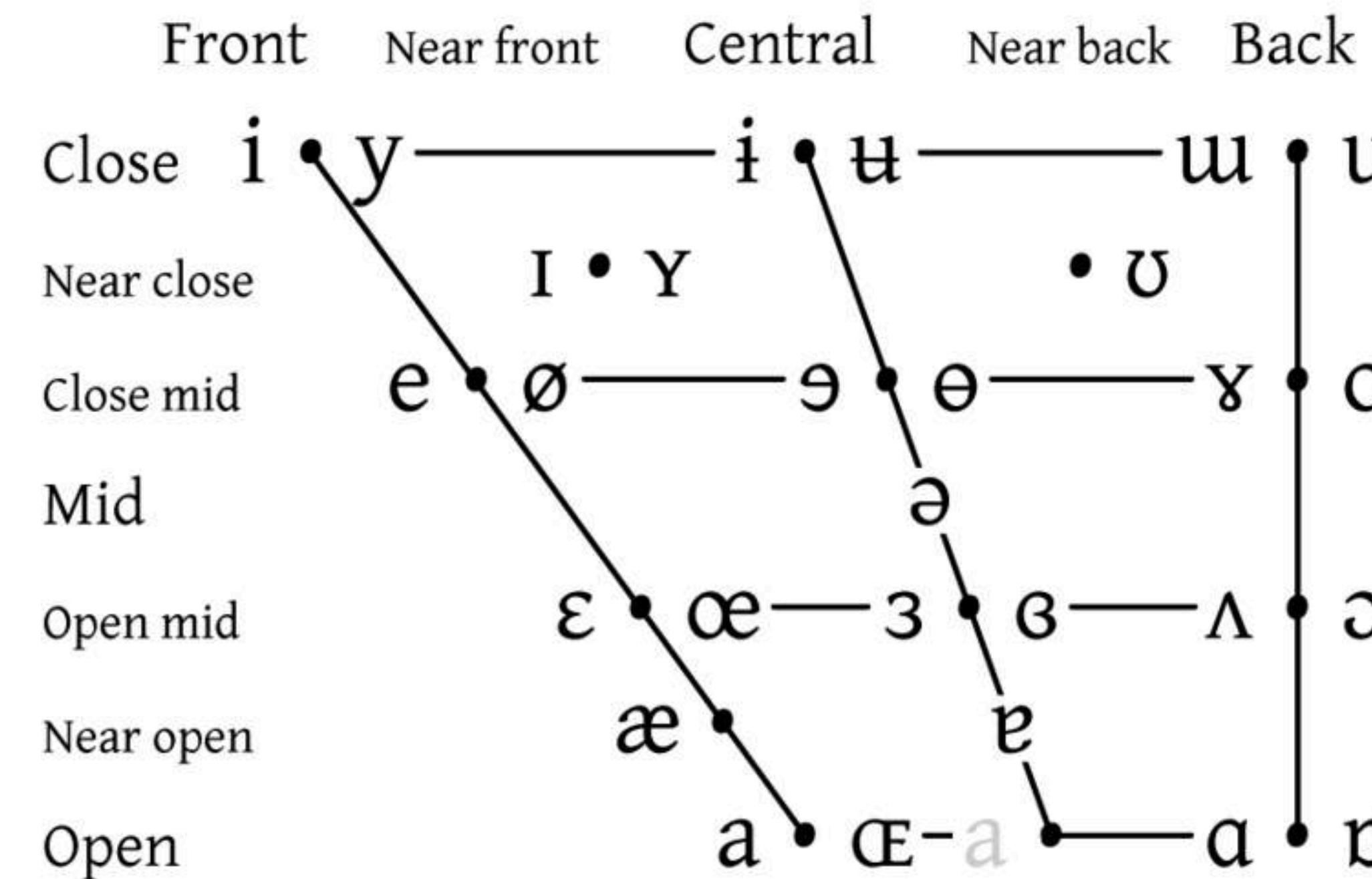
Sound units

- **Phones** are acoustically distinct units of speech
- **Phonemes** are abstract linguistic units that impart different meanings in a given language
 - Minimal pair: pan vs. ban
- **Allophones** are different acoustic realisations of the same phoneme
- **Phonetics** is the study of speech sounds and how they're produced
- **Phonology** is the study of patterns of sounds in different languages

Vowels

- Sounds produced with no obstruction to the flow of air through the vocal tract

VOWEL QUADRILATERAL



Spectrogram

- Spectrogram is a sequence of spectra stacked together in time, with amplitude of the frequency components expressed as a heat map
- Spectrograms of certain vowels:
<http://www.phon.ucl.ac.uk/courses/spsci/iss/week5.php>
- Praat (<http://www.fon.hum.uva.nl/praat/>) is a good toolkit to analyse speech signals (plot spectrograms, generate pitch contours, etc.)

Consonants (voicing/place/manner)

- “Consonants are made by restricting or blocking the airflow in some way, and may be voiced or unvoiced.” (J&M, Ch. 7)

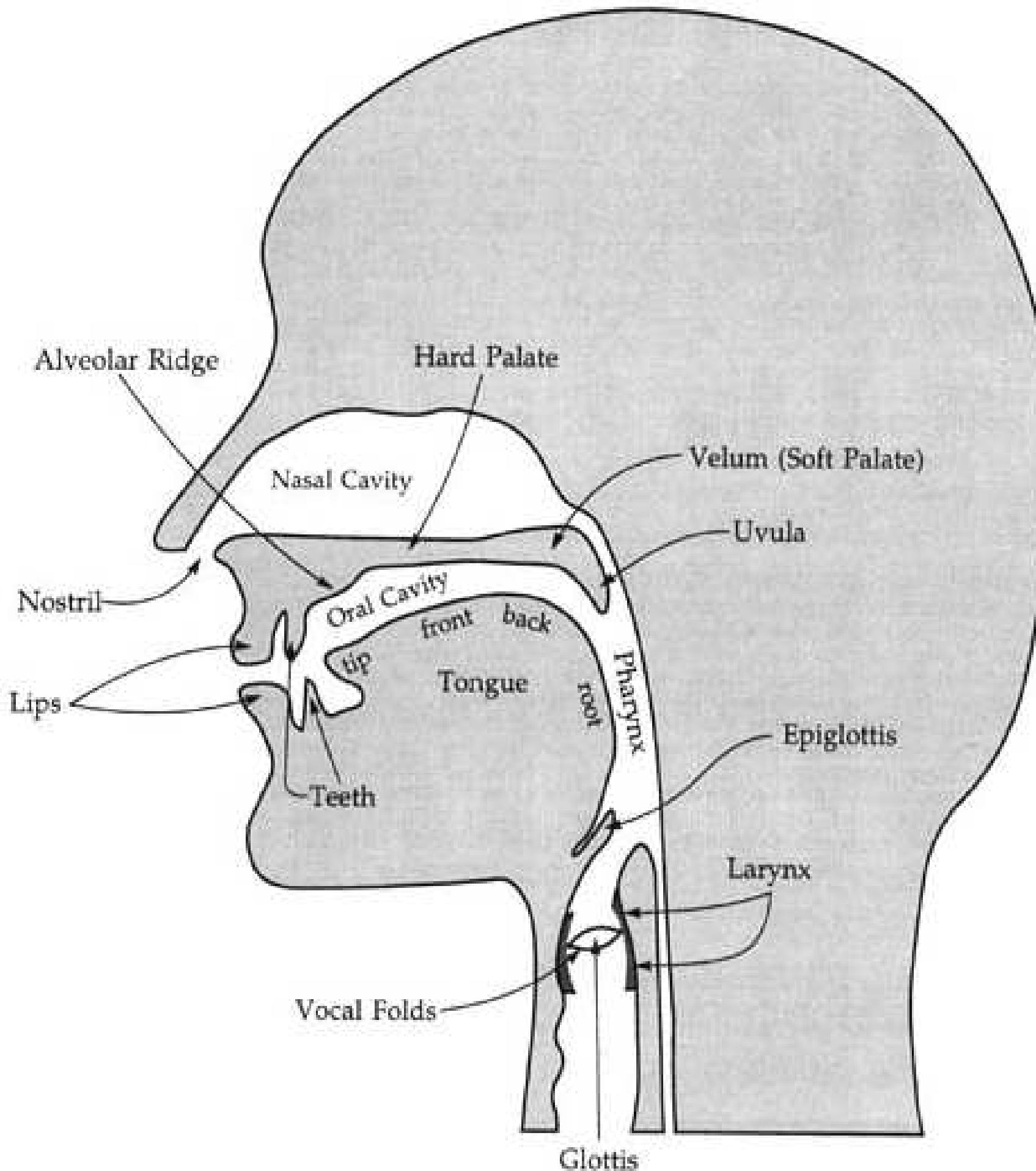
Voiced/Unvoiced Sounds

- Sounds made with vocal cords vibrating: **voiced**
 - E.g. /g/, /d/, etc.
 - All English vowel sounds are voiced
- Sounds made without vocal cord vibration: **voiceless**
 - E.g. /k/, /t/, etc.

Consonants (voicing/place/manner)

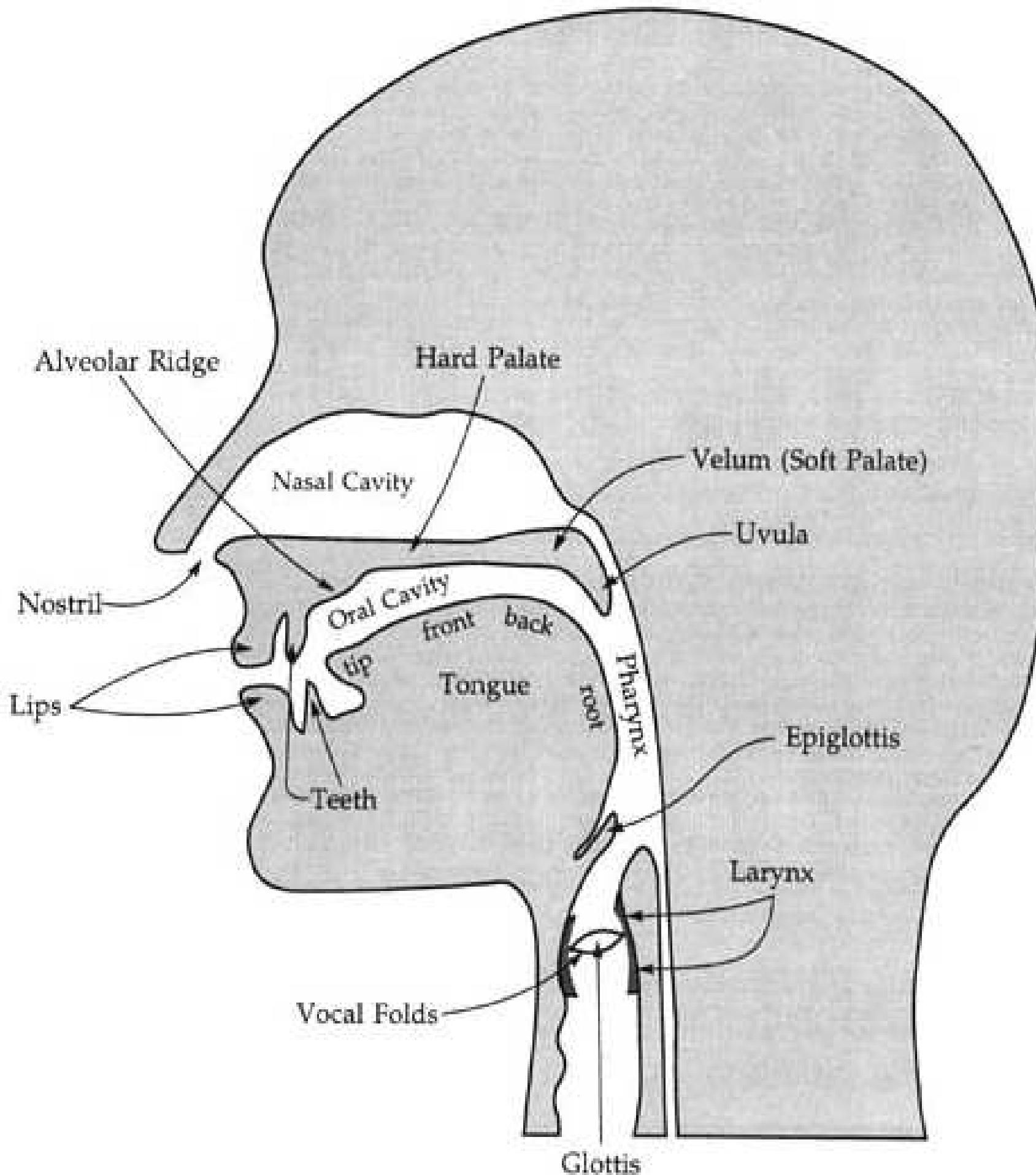
- “Consonants are made by restricting or blocking the airflow in some way, and may be voiced or unvoiced.” (J&M, Ch. 7)
- Consonants can be labeled depending on
 - *where* the constriction is made (place of articulation)
 - *how* the constriction is made (manner of articulation)

Place of articulation



- Bilabial (both lips)
[b],[p],[m], etc.
- Labiodental (with lower lip and upper teeth)
[f], [v], etc.
- Interdental (tip of tongue between teeth)
[θ] (thought), [δ] (this)

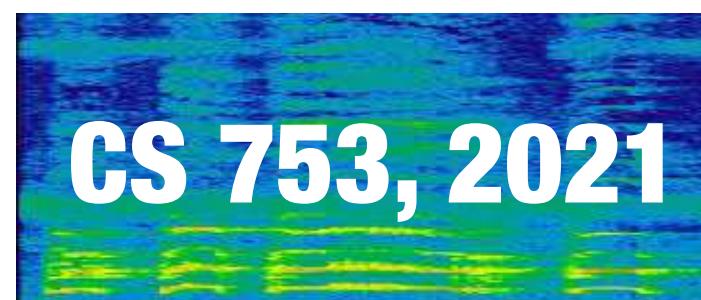
Manner of articulation



- Plosive/Stop (airflow completely blocked followed by a release)
[p], [g], [t], etc.
- Fricative (constricted airflow)
[f], [s], [th], etc.
- Affricate (stop + fricative)
[ch], [jh], etc.
- Nasal (lowering velum)
[n], [m], etc.

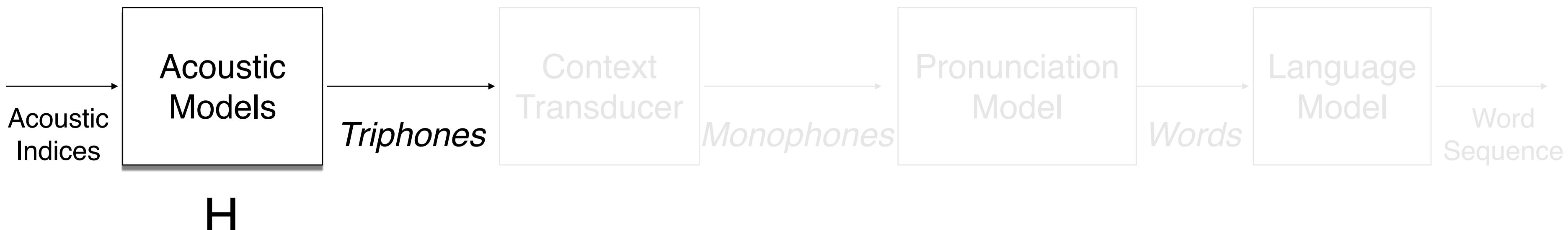
Feedforward Neural Network Acoustic Models

Lecture 6a

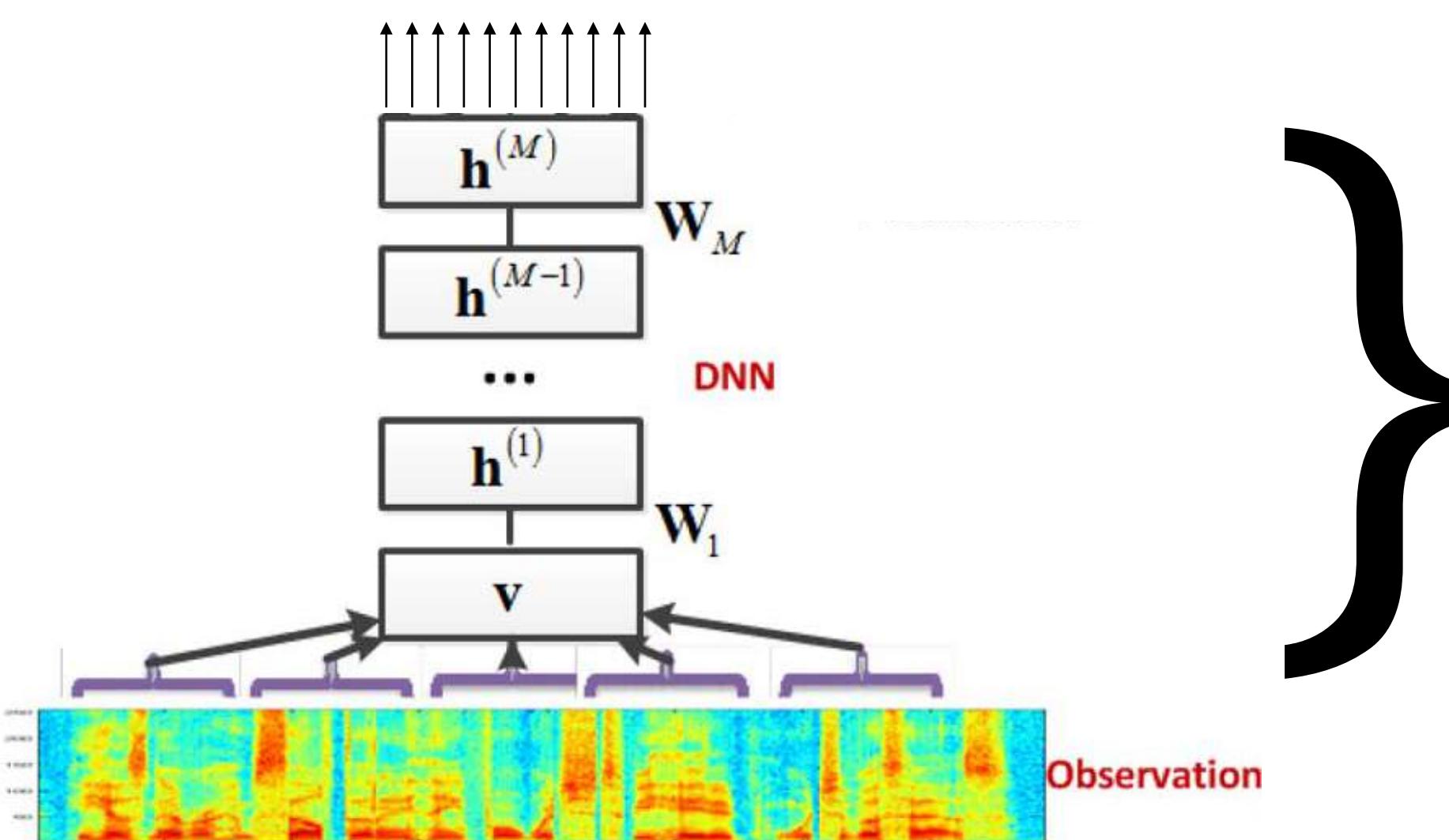


Instructor: Preethi Jyothi, IITB

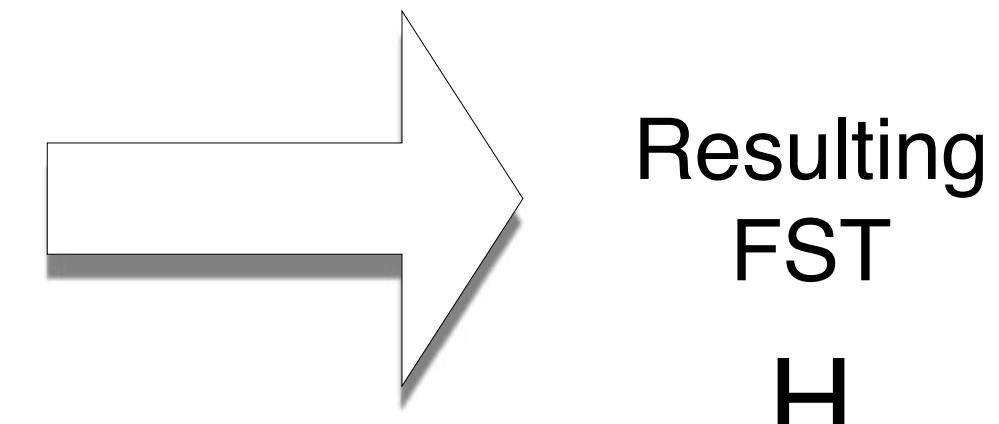
DNN-based acoustic models?



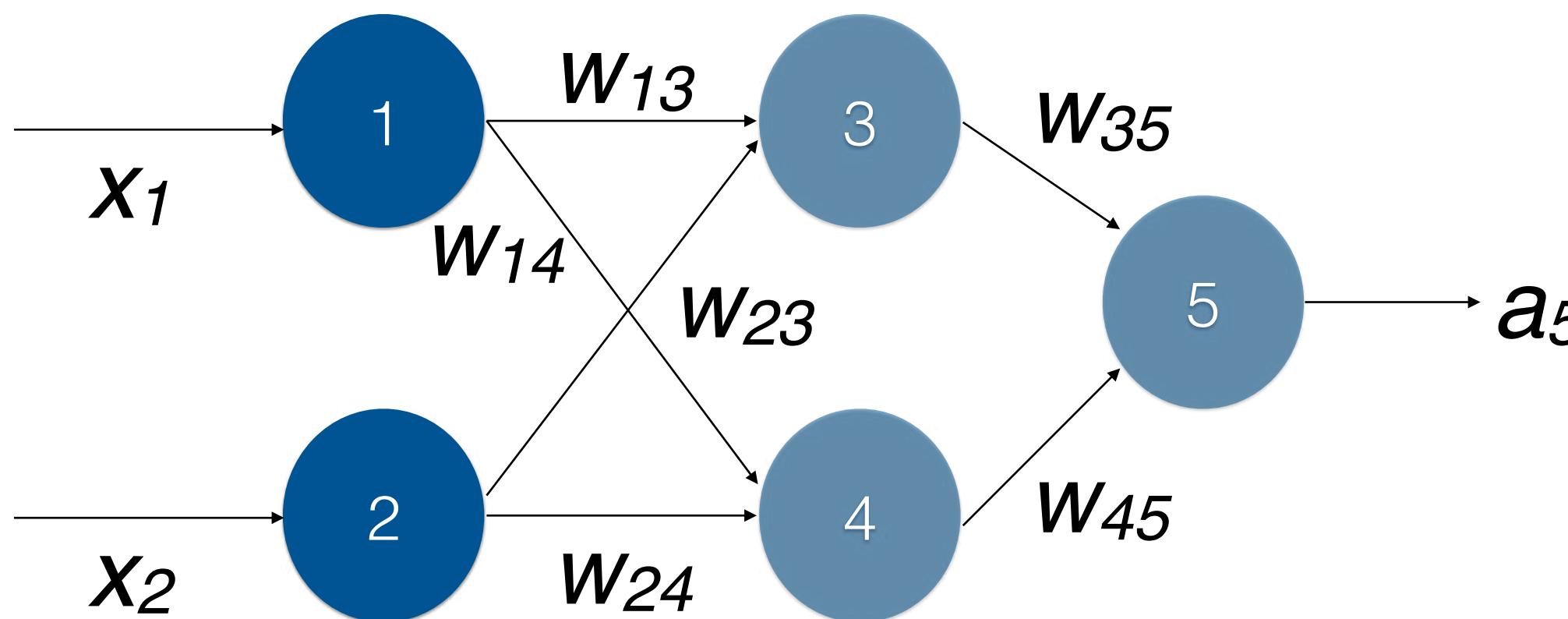
Phone posteriors



Can we use
deep neural networks
instead of HMMs to
learn mappings
between acoustics
and phones?



Feed-forward Neural Network Parameterized Model



$$\begin{aligned}a_5 &= g(w_{35} \cdot a_3 + w_{45} \cdot a_4) \\&= g(w_{35} \cdot (g(w_{13} \cdot a_1 + w_{23} \cdot a_2)) + \\&\quad w_{45} \cdot (g(w_{14} \cdot a_1 + w_{24} \cdot a_2)))\end{aligned}$$

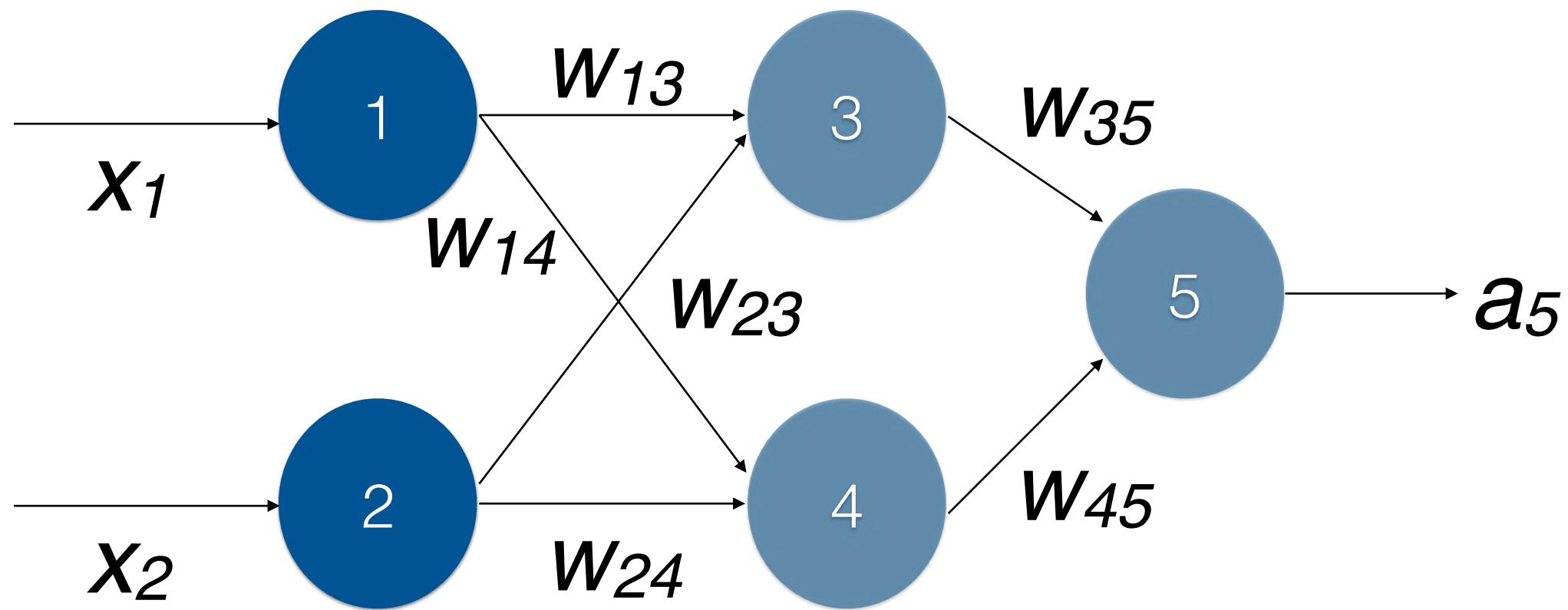
Parameters of
the network: all w_{ij}
(and biases not
shown here)

If \mathbf{x} is a 2-dimensional vector and the layer above it is a 2-dimensional vector \mathbf{h} , a fully-connected layer is associated with:

$$\mathbf{h} = \mathbf{x}\mathbf{W} + \mathbf{b}$$

where w_{ij} in \mathbf{W} is the weight of the connection between i^{th} neuron in the input row and j^{th} neuron in the first hidden layer and \mathbf{b} is the bias vector

Feed-forward Neural Network Parameterized Model



$$\begin{aligned}a_5 &= g(w_{35} \cdot a_3 + w_{45} \cdot a_4) \\&= g(w_{35} \cdot (g(w_{13} \cdot a_1 + w_{23} \cdot a_2)) + \\&\quad w_{45} \cdot (g(w_{14} \cdot a_1 + w_{24} \cdot a_2)))\end{aligned}$$

The simplest neural network is the perceptron:

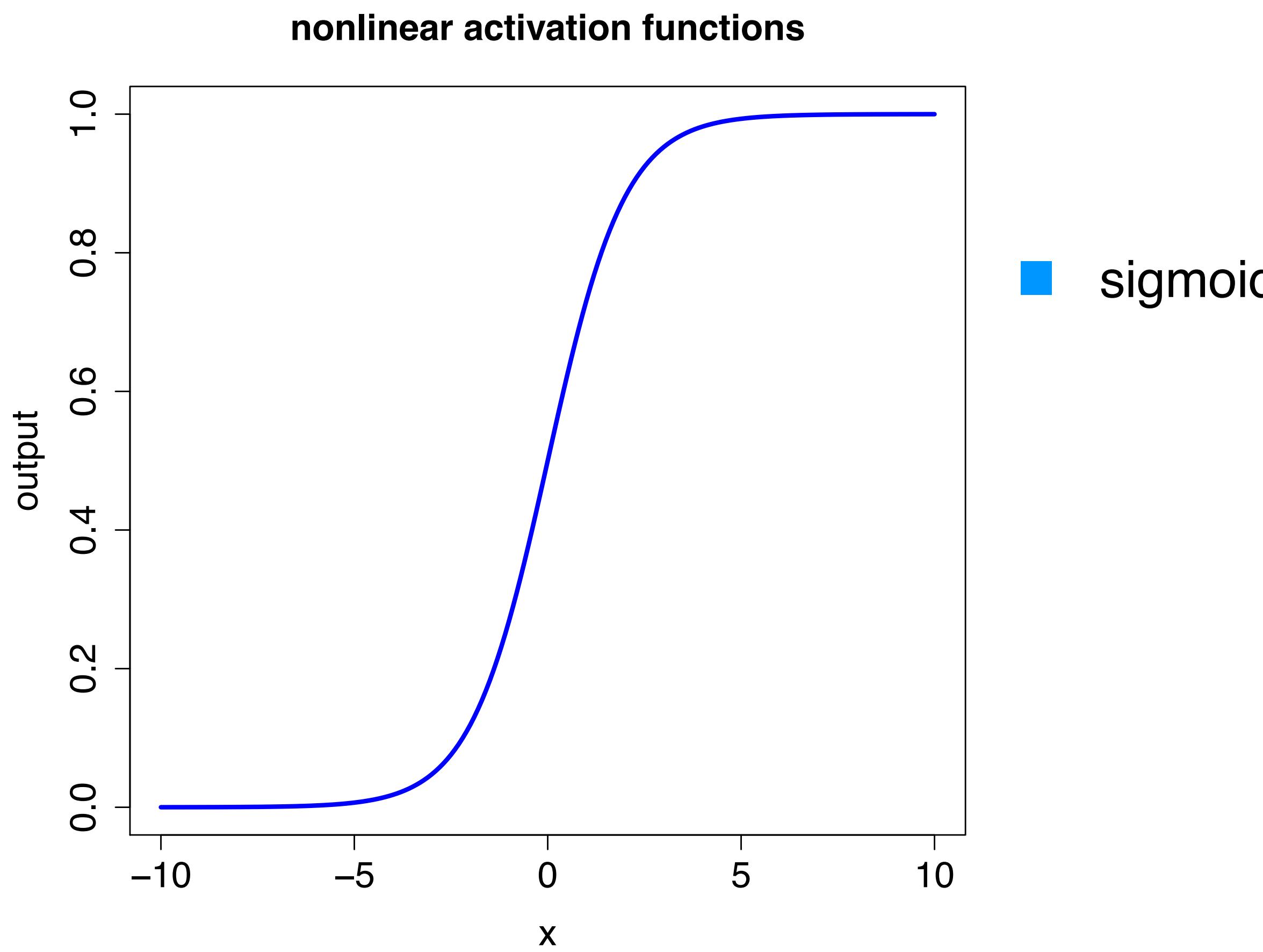
$$\text{Perceptron}(x) = \mathbf{x}\mathbf{W} + \mathbf{b}$$

A 1-layer feedforward neural network has the form:

$$\text{MLP}(x) = g(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1) \mathbf{W}_2 + \mathbf{b}_2$$

Common Activation Functions (g)

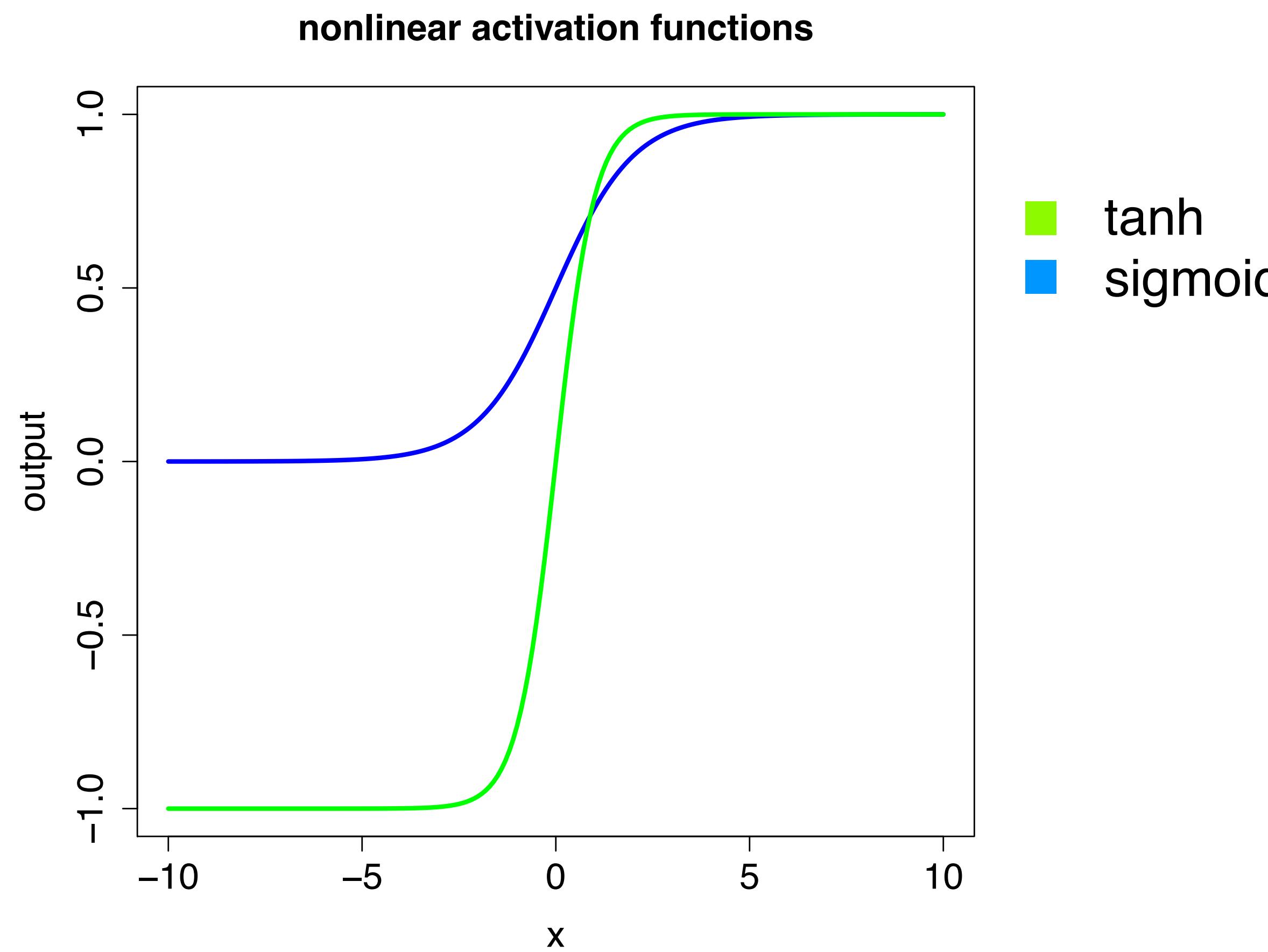
Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$



Common Activation Functions (g)

Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$

Hyperbolic tangent (tanh): $\tanh(x) = (e^{2x} - 1)/(e^{2x} + 1)$

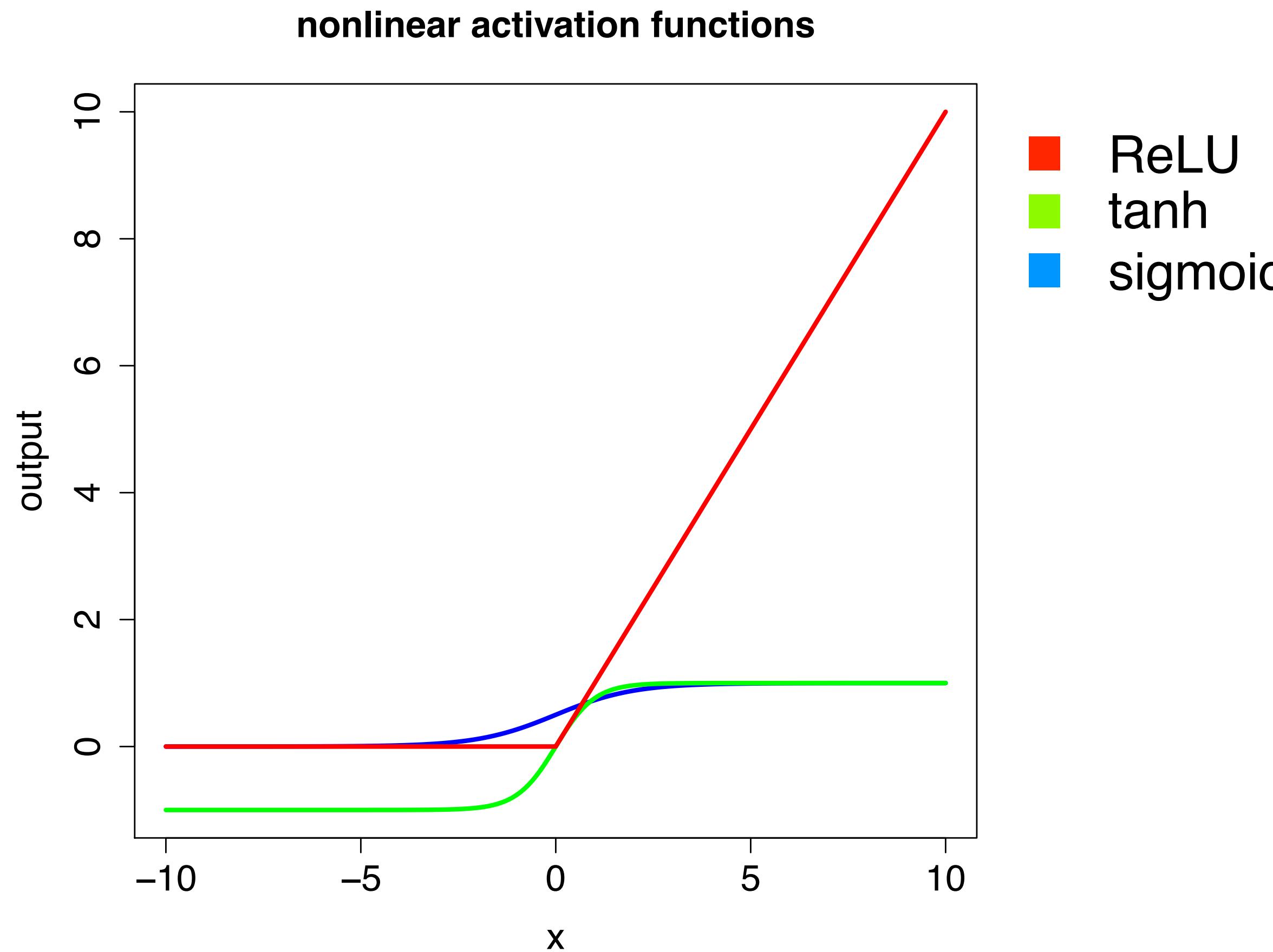


Common Activation Functions (g)

Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$

Hyperbolic tangent (tanh): $\tanh(x) = (e^{2x} - 1)/(e^{2x} + 1)$

Rectified Linear Unit (ReLU): $\text{ReLU}(x) = \max(0, x)$



Optimization Problem

- To train a neural network, define a loss function $L(y, \tilde{y})$: a function of the true output y and the predicted output \tilde{y}
- $L(y, \tilde{y})$ assigns a non-negative numerical score to the neural network's output, \tilde{y}
- The parameters of the network are set to minimise L over the training examples (i.e. a sum of losses over different training samples)
- L is typically minimised using a *gradient-based method*

Stochastic Gradient Descent (SGD)

SGD Algorithm

Inputs:

Function $NN(x; \theta)$, Training examples, $x_1 \dots x_n$ and outputs, $y_1 \dots y_n$ and Loss function L .

do until stopping criterion

 Pick a training example x_i, y_i

 Compute the loss $L(NN(x_i; \theta), y_i)$

 Compute gradient of L , ∇L with respect to θ

$\theta \leftarrow \theta - \eta \nabla L$

done

Return: θ

Training a Neural Network

Define the **Loss function** to be minimised as a node L

Goal: Learn weights for the neural network which minimise L

Gradient Descent: Find $\partial L / \partial w$ for every weight w , and update it as
 $w \leftarrow w - \eta \partial L / \partial w$

How do we efficiently compute $\partial L / \partial w$ for all w ?

Will compute $\partial L / \partial u$ for every node u in the network!

$\partial L / \partial w = \partial L / \partial u \cdot \partial u / \partial w$ where u is the node which uses w

Backpropagation

$$\partial L / \partial u = \sum_{v \in \Gamma(u)} \partial L / \partial v \cdot \partial v / \partial u$$

Backpropagation

Base case: $\partial L / \partial L = 1$

For each u (top to bottom):

For each $v \in \Gamma(u)$:

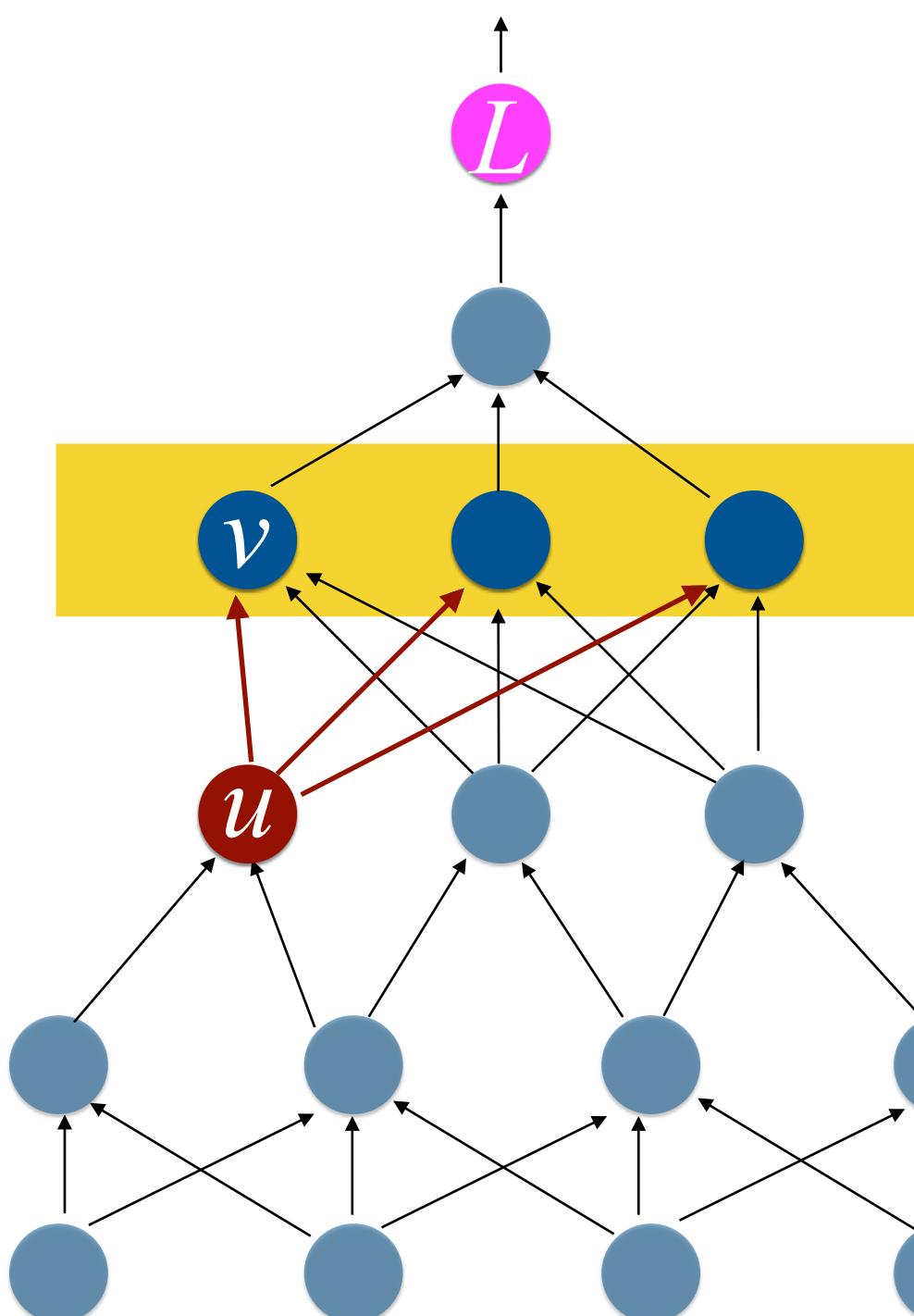
Inductively, have computed $\partial L / \partial v$

Directly compute $\partial v / \partial u$

Compute $\partial L / \partial u$

Compute $\partial L / \partial w$

where $\partial L / \partial w = \partial L / \partial u \cdot \partial u / \partial w$



Forward Pass

First, in a forward pass, compute values of all nodes given an input
(The values of each node will be needed during backprop)

Where values computed in the forward pass may be needed

History of Neural Networks in ASR

- Neural networks for speech recognition were explored as early as 1987
- Deep neural networks for speech
 - Beat state-of-the-art on the TIMIT corpus [M09]
 - Significant improvements shown on large-vocabulary systems [D11]
 - Dominant ASR paradigm [H12]

[M09] A. Mohamed, G. Dahl, and G. Hinton, “Deep belief networks for phone recognition,” NIPS Workshop on Deep Learning for Speech Recognition, 2009.

[D11] G. Dahl, D. Yu, L. Deng, and A. Acero, “Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition,” TASL 20(1)

[H12] G. Hinton, et al., “Deep Neural Networks for Acoustic Modeling in Speech Recognition”, IEEE Signal Processing Magazine, 2012.

Feedforward Neural Networks for ASR

- Two main categories of approaches have been explored:
 1. Hybrid neural network-HMM systems: Use DNNs to estimate HMM observation probabilities
 2. Tandem system: NNs used to generate input features that are fed to an HMM-GMM acoustic model

Feedforward Neural Networks for ASR

- Two main categories of approaches have been explored:
 1. Hybrid neural network-HMM systems: Use DNNs to estimate HMM observation probabilities
 2. Tandem system: DNNs used to generate input features that are fed to an HMM-GMM acoustic model

Decoding an ASR system

- Recall how we decode the most likely word sequence W for an acoustic sequence O :

$$W^* = \arg \max_W \Pr(O|W) \Pr(W)$$

- The acoustic model $\Pr(O|W)$ can be further decomposed as (here, Q represents a triphone state sequence):

$$\begin{aligned}\Pr(O|W) &= \sum_Q \Pr(O, Q|W) \\ &= \sum_Q \Pr(O|Q, W) \Pr(Q|W) \Pr(W) \\ &\approx \sum_Q \Pr(O|Q) \Pr(Q|W) \Pr(W)\end{aligned}$$

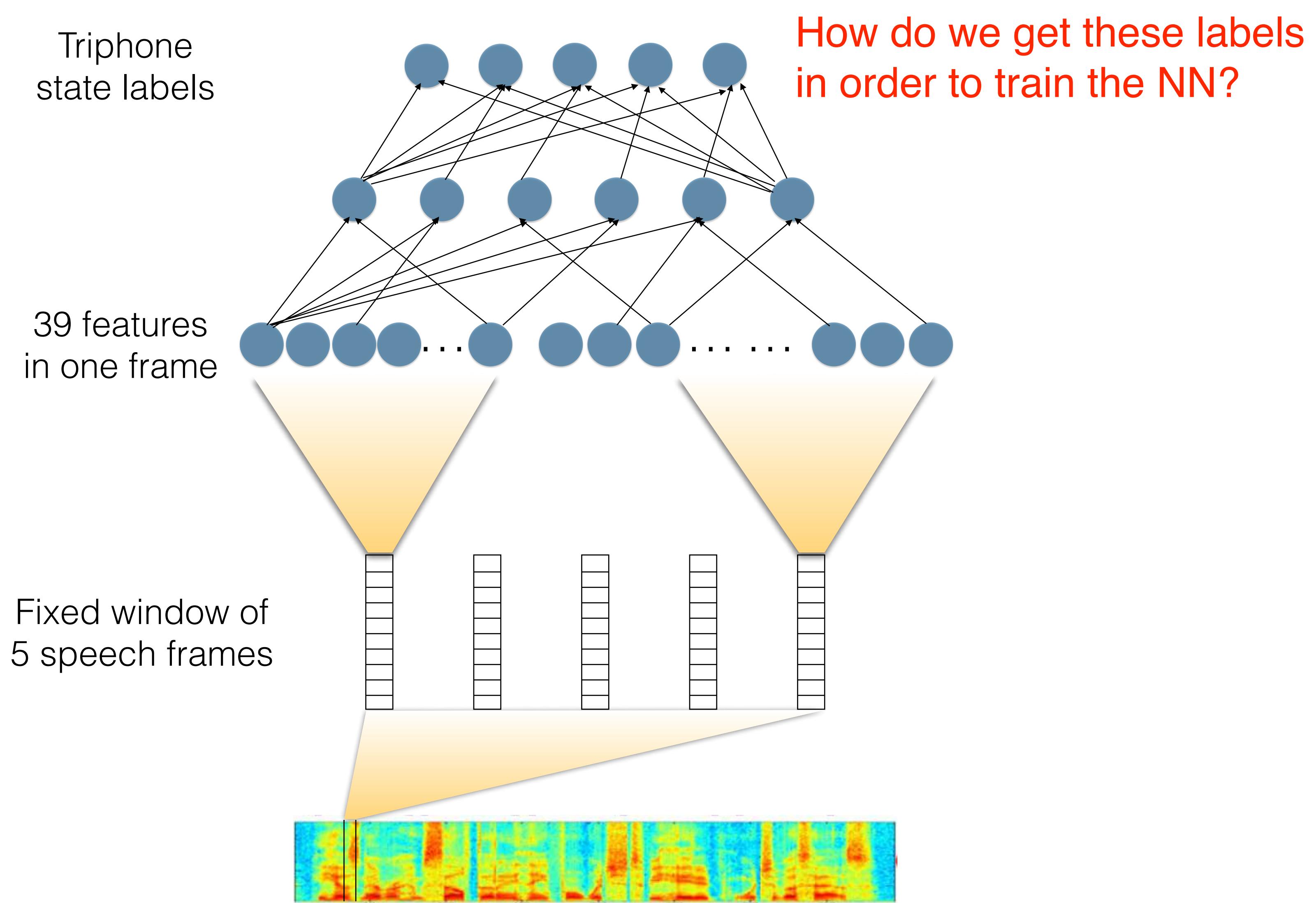
Hybrid system decoding

We've seen $\Pr(O|Q)$ estimated using a Gaussian Mixture Model.
Let's use a neural network instead to model $\Pr(O|Q)$.

$$\begin{aligned}\Pr(O|Q) &= \prod_t \Pr(o_t|q_t) \\ \Pr(o_t|q_t) &= \frac{\Pr(q_t|o_t) \Pr(o_t)}{\Pr(q_t)} \\ &\propto \frac{\Pr(q_t|o_t)}{\Pr(q_t)}\end{aligned}$$

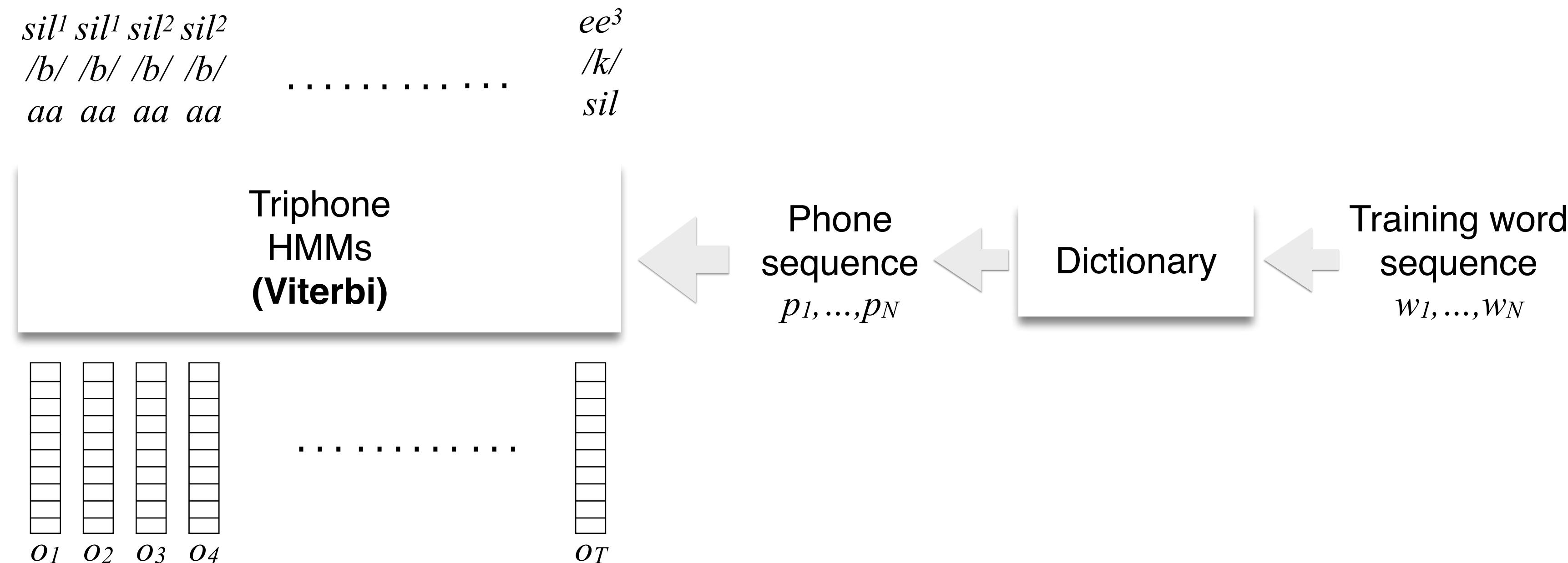
where o_t is the acoustic vector at time t and q_t is a triphone HMM state
Here, $\Pr(q_t|o_t)$ are posteriors from a trained neural network.
 $\Pr(o_t|q_t)$ is then a scaled posterior.

Computing $\Pr(q_t|o_t)$ using a deep NN



Triphone labels

- Forced alignment: Use current acoustic model to find the most likely sequence of HMM states given a sequence of acoustic vectors. (**Algorithm to help compute this?**)
- The “Viterbi paths” for the training data, are also referred to as forced alignments



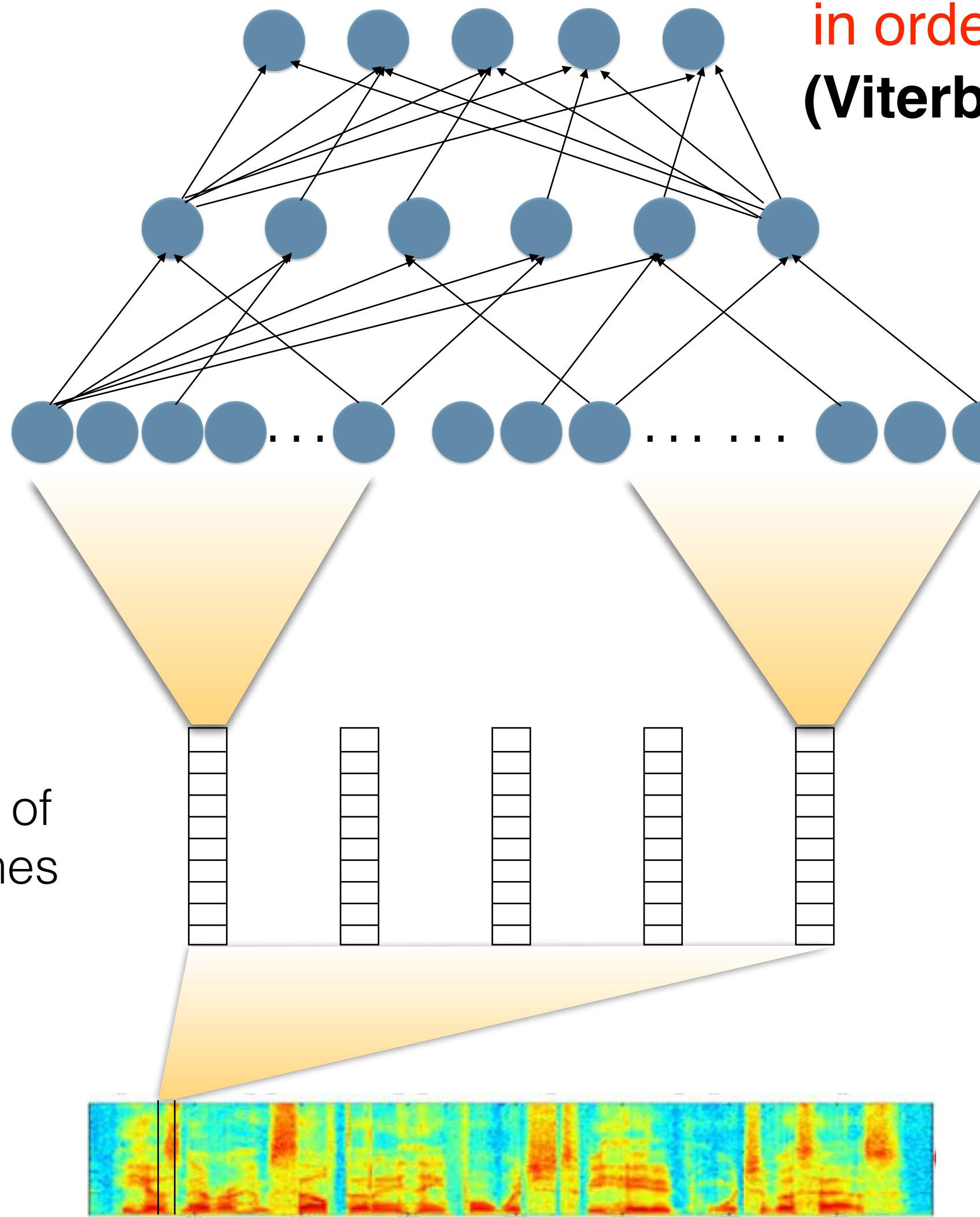
Computing $\Pr(q_t | o_t)$ using a deep NN

Triphone
state labels

39 features
in one frame

Fixed window of
5 speech frames

How do we get these labels
in order to train the NN?
(Viterbi) Forced alignment



Computing priors $\Pr(q_t)$

- To compute HMM observation probabilities, $\Pr(o_t|q_t)$, we need both $\Pr(q_t|o_t)$ and $\Pr(q_t)$
- The posterior probabilities $\Pr(q_t|o_t)$ are computed using a trained neural network
- $\Pr(q_t)$ are relative frequencies of each triphone state as determined by the forced Viterbi alignment of the training data

Hybrid Networks

- The networks are trained with a minimum cross-entropy criterion

$$L(y, \hat{y}) = - \sum_i y_i \log(\hat{y}_i)$$

- Advantages of hybrid systems:
 1. Fewer assumptions made about acoustic vectors being uncorrelated: Multiple inputs used from a window of time steps
 2. Discriminative objective function used to learn the observation probabilities

Summary of DNN-HMM acoustic models

Comparison against HMM-GMM on different tasks

[TABLE 3] A COMPARISON OF THE PERCENTAGE WERs USING DNN-HMMs AND GMM-HMMs ON FIVE DIFFERENT LARGE VOCABULARY TASKS.

TASK	HOURS OF TRAINING DATA	DNN-HMM	GMM-HMM WITH SAME DATA	GMM-HMM WITH MORE DATA
SWITCHBOARD (TEST SET 1)	309	18.5	27.4	18.6 (2,000 H)
SWITCHBOARD (TEST SET 2)	309	16.1	23.6	17.1 (2,000 H)
ENGLISH BROADCAST NEWS	50	17.5	18.8	
BING VOICE SEARCH (SENTENCE ERROR RATES)	24	30.4	36.2	
GOOGLE VOICE INPUT	5,870	12.3		16.0 (>> 5,870 H)
YOUTUBE	1,400	47.6	52.3	

Hybrid DNN-HMM systems consistently outperform GMM-HMM systems (sometimes even when the latter is trained with lots more data)

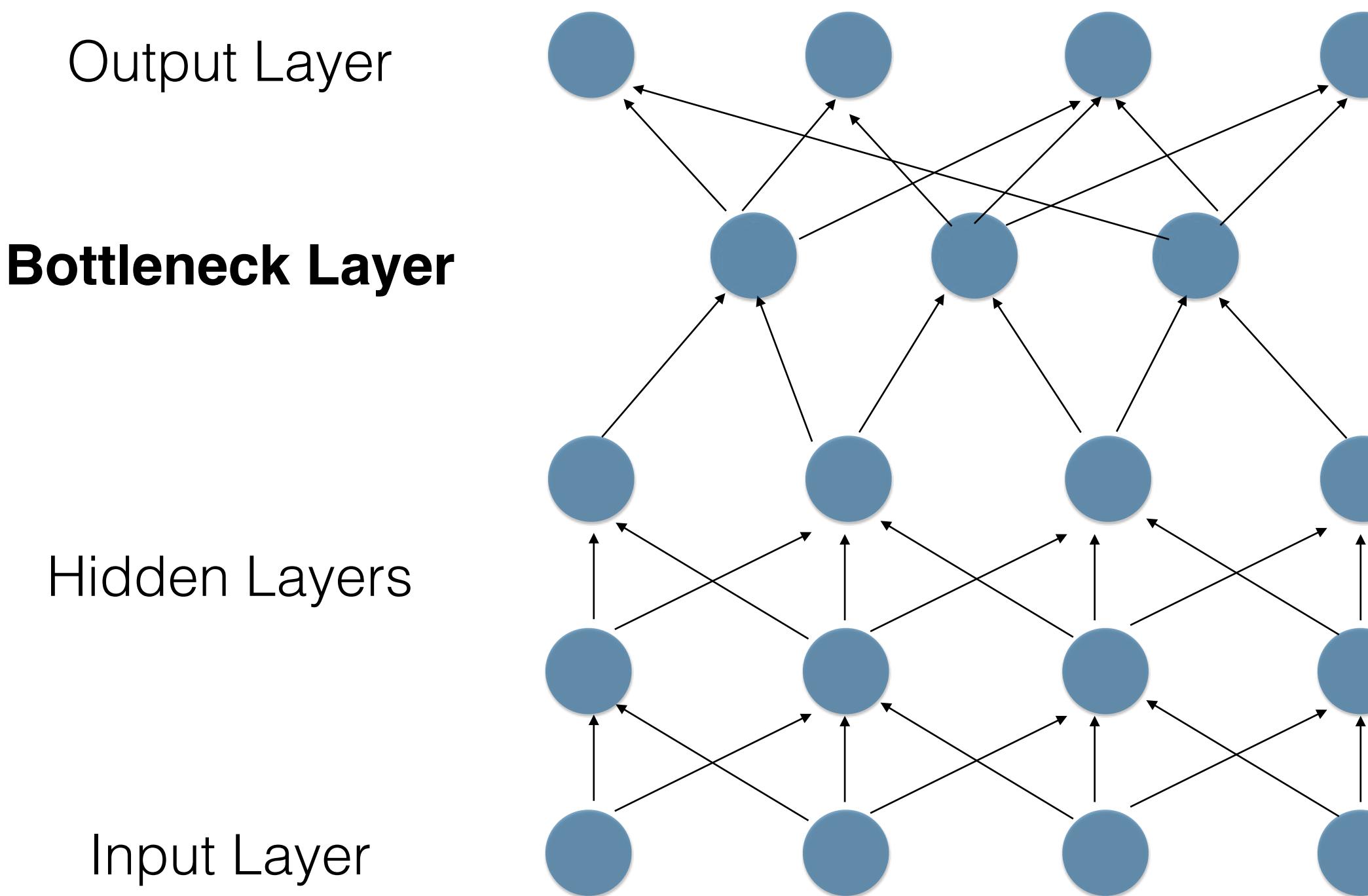
Neural Networks for ASR

- Two main categories of approaches have been explored:
 1. Hybrid neural network-HMM systems: Use DNNs to estimate HMM observation probabilities
 2. Tandem system: NNs used to generate input features that are fed to an HMM-GMM acoustic model

Tandem system

- First, train a DNN to estimate the posterior probabilities of each subword unit (monophone, triphone state, etc.)
- In a hybrid system, these posteriors (after scaling) would be used as observation probabilities for the HMM acoustic models
- In the tandem system, the DNN outputs are used as “feature” inputs to HMM-GMM models

Bottleneck Features

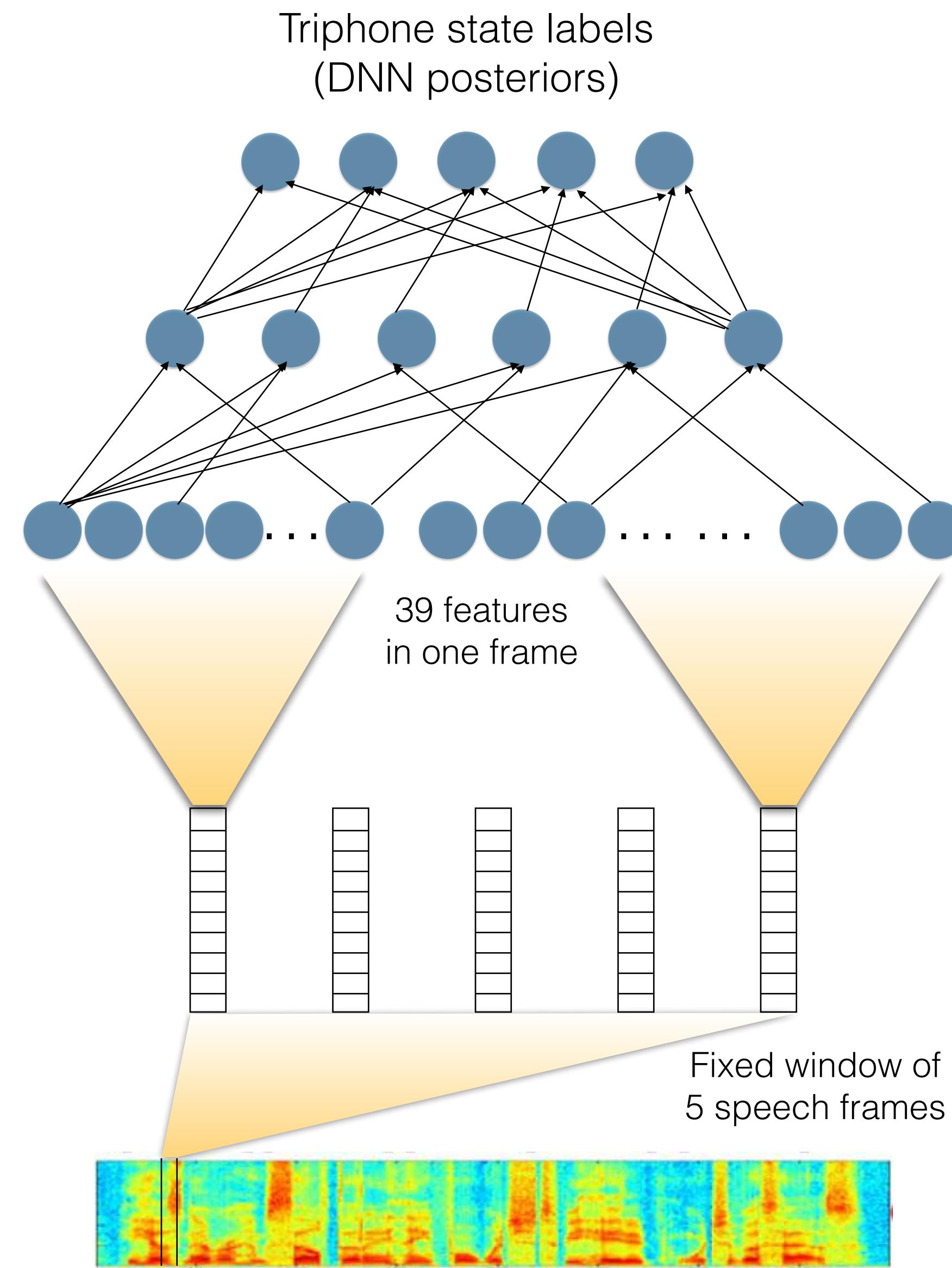


Use a low-dimensional *bottleneck* layer representation to extract features

These bottleneck features are in turn used as inputs to HMM-GMM models

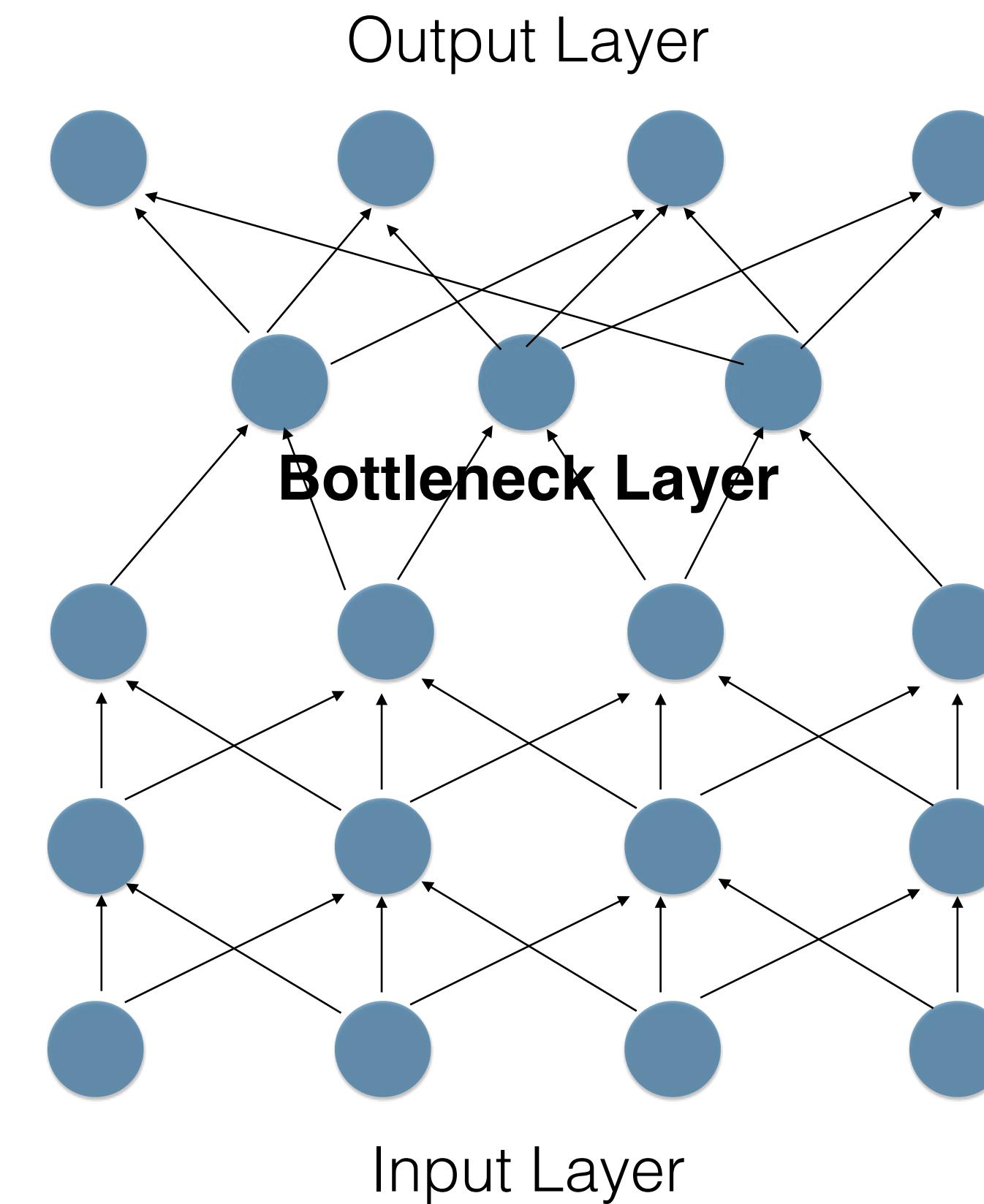
Recap: Hybrid DNN-HMM Systems

- Instead of GMMs, use scaled DNN posteriors as the HMM observation probabilities
- DNN trained using triphone labels derived from a forced alignment “Viterbi” step.
- Forced alignment: Given a training utterance $\{O, W\}$, find the most likely sequence of states (and hence triphone state labels) using a set of trained triphone HMM models, M . Here M is constrained by the triphones in W .



Recap: Tandem DNN-HMM Systems

- Neural networks are used as “feature extractors” to train HMM-GMM models
- Use a low-dimensional *bottleneck* layer representation to extract features from the bottleneck layer
- These bottleneck features are subsequently fed to GMM-HMMs as input

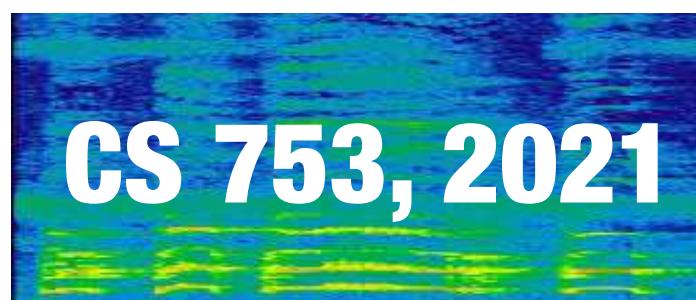


Feedforward DNNs we've seen so far...

- Assume independence among the training instances (modulo the context window of frames)
 - Independent decision made about classifying each individual speech frame
 - Network state is completely reset after each speech frame is processed
- This independence assumption fails for data like speech which has temporal and sequential structure
- Two model architectures that capture longer ranges of acoustic context:
 - 1. Time delay neural networks (TDNNs)**
 - 2. Recurrent neural networks (RNNs)**

TDNNs and Introduction to RNN Models

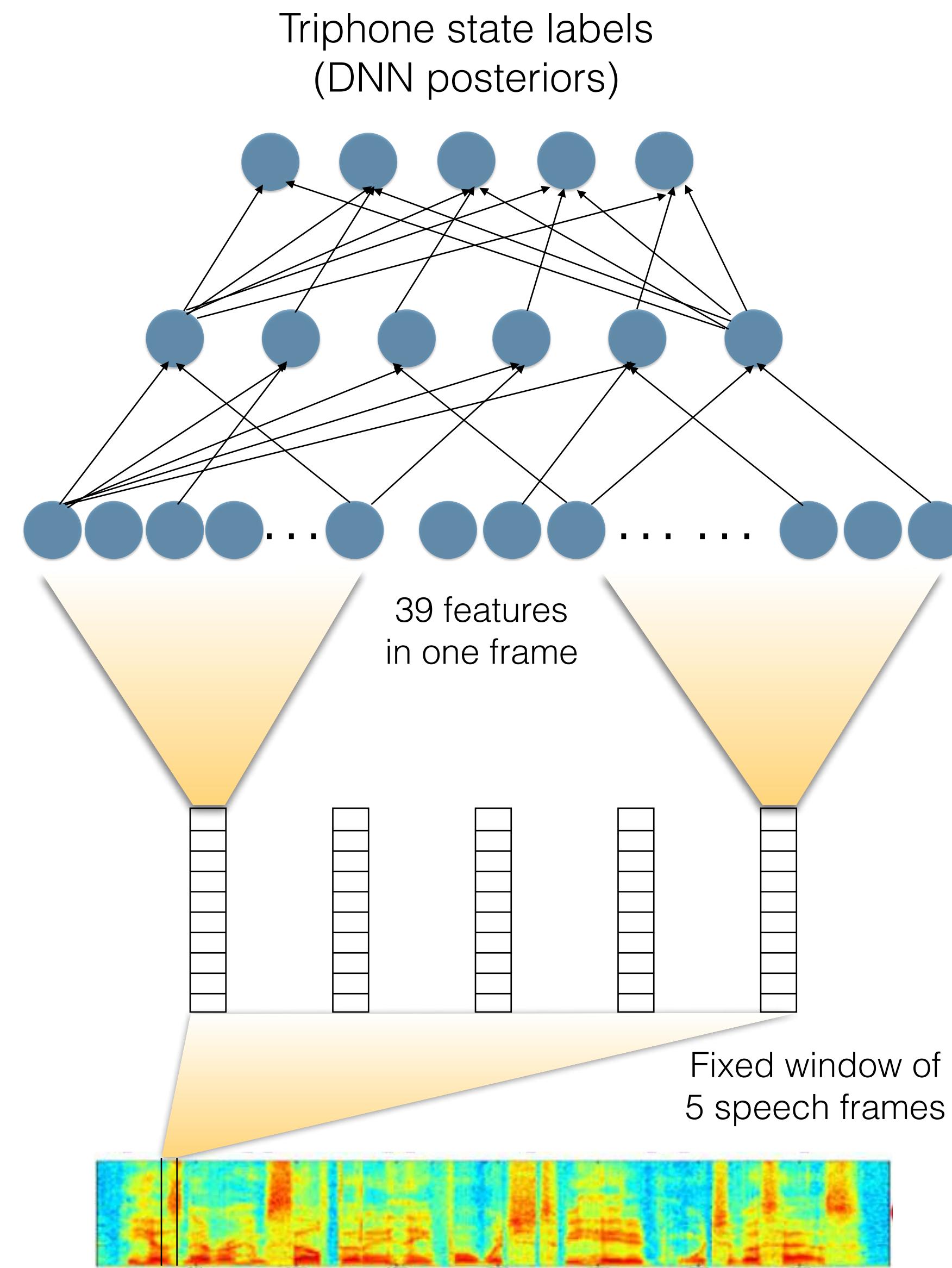
Lecture 6b



Instructor: Preethi Jyothi, IITB

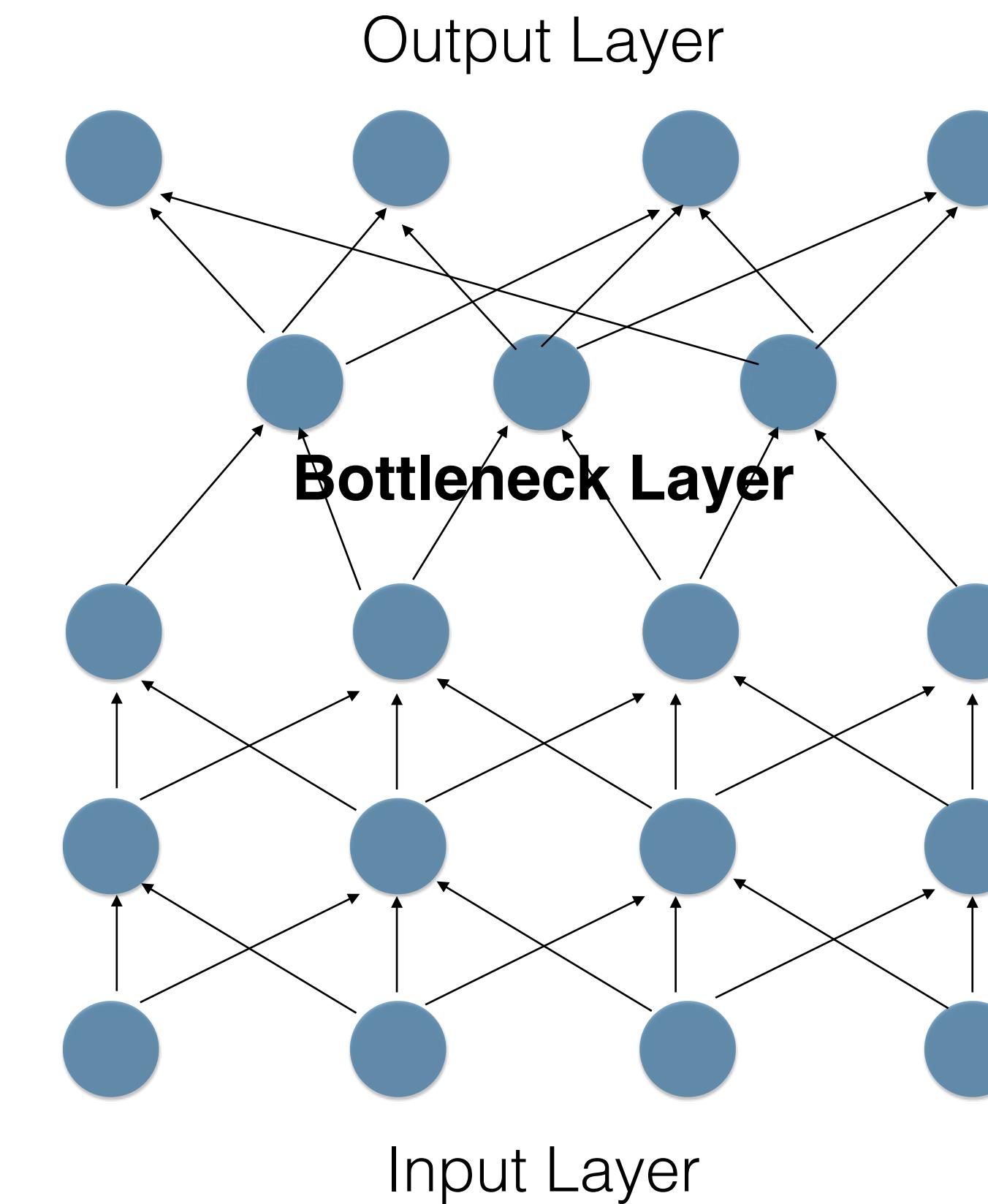
Recap: Hybrid DNN-HMM Systems

- Instead of GMMs, use scaled DNN posteriors as the HMM observation probabilities
- DNN trained using triphone labels derived from a forced alignment “Viterbi” step.
- Forced alignment: Given a training utterance $\{O, W\}$, find the most likely sequence of states (and hence triphone state labels) using a set of trained triphone HMM models, M . Here M is constrained by the triphones in W .



Recap: Tandem DNN-HMM Systems

- Neural networks are used as “feature extractors” to train HMM-GMM models
- Use a low-dimensional *bottleneck* layer representation to extract features from the bottleneck layer
- These bottleneck features are subsequently fed to GMM-HMMs as input

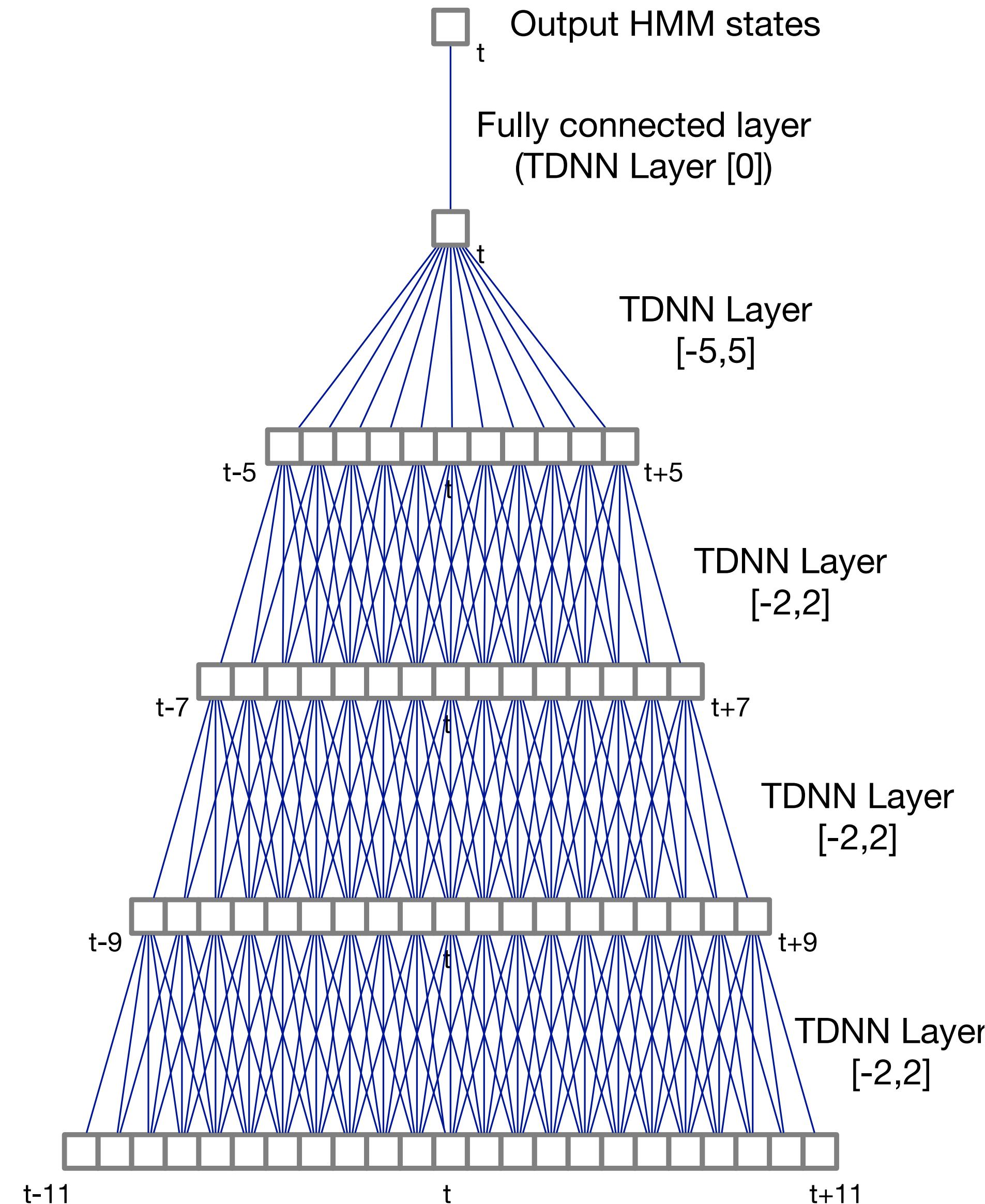


Feedforward DNNs we've seen so far...

- Assume independence among the training instances (modulo the context window of frames)
 - Independent decision made about classifying each individual speech frame
 - Network state is completely reset after each speech frame is processed
- This independence assumption fails for data like speech which has temporal and sequential structure
- Two model architectures that capture longer ranges of acoustic context:
 1. **Time delay neural networks (TDNNs)**

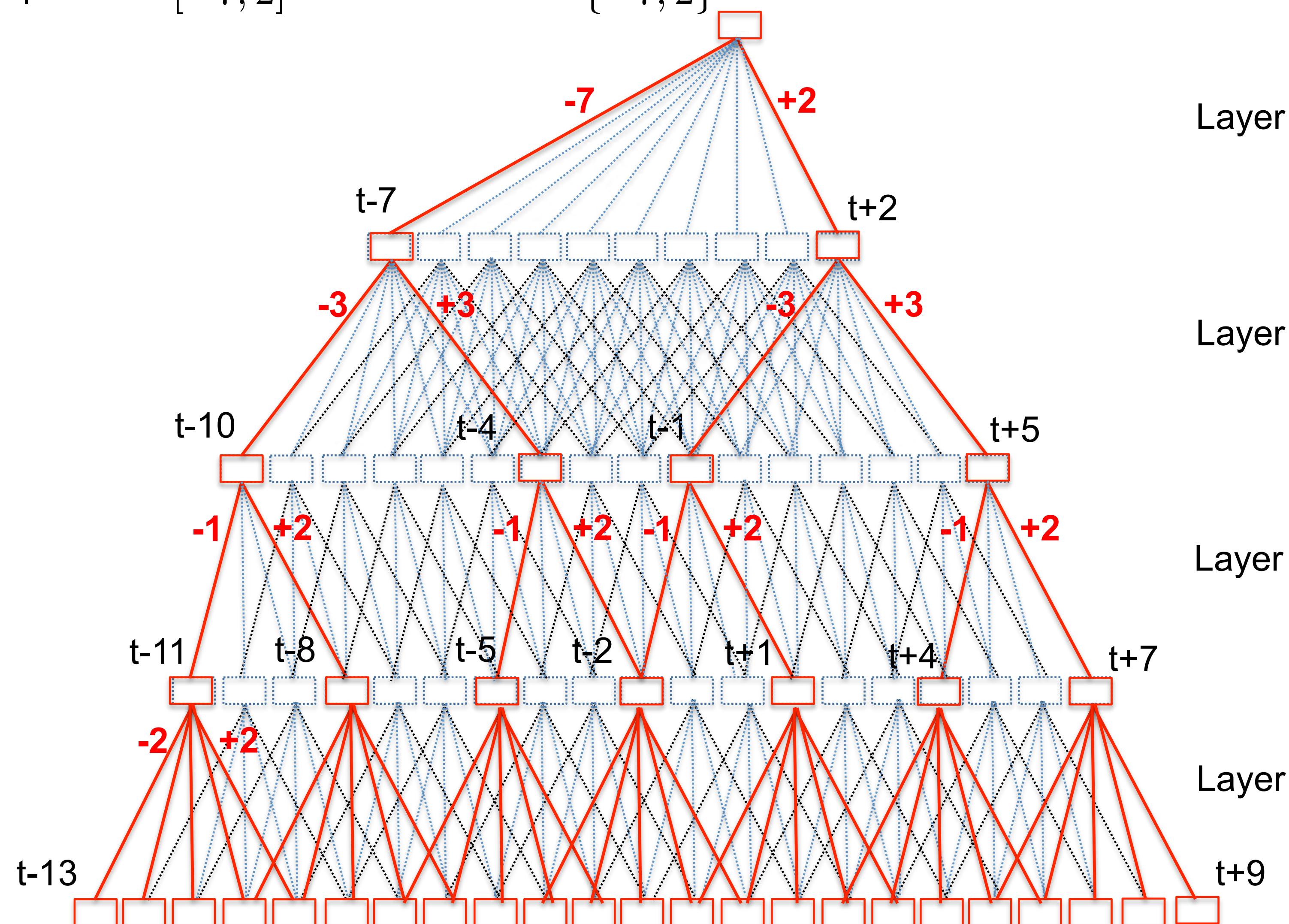
Time Delay Neural Networks

- Each layer in a TDNN acts at a different temporal resolution
 - Processes a context window from the previous layer
 - Higher layers have a wider receptive field into the input
- However, a lot more computation needed than DNNs!



Time Delay Neural Networks

Layer	Input context	Input context with sub-sampling
1	$[-2, +2]$	$[-2, 2]$
2	$[-1, 2]$	$\{-1, 2\}$
3	$[-3, 3]$	$\{-3, 3\}$
4	$[-7, 2]$	$\{-7, 2\}$



- Large overlaps between input contexts computed at neighbouring time steps
- Assuming neighbouring activations are correlated, how do we exploit this?
- Subsample by allowing gaps between frames.
- Splice increasingly wider context in higher layers.

Time Delay Neural Networks

Model	Network Context	Layerwise Context					WER	
		1	2	3	4	5	Total	SWB
DNN-A	[-7, 7]	[-7, 7]	{0}	{0}	{0}	{0}	22.1	15.5
DNN-A ₂	[-7, 7]	[-7, 7]	{0}	{0}	{0}	{0}	21.6	15.1
DNN-B	[-13, 9]	[-13, 9]	{0}	{0}	{0}	{0}	22.3	15.7
DNN-C	[-16, 9]	[-16, 9]	{0}	{0}	{0}	{0}	22.3	15.7
TDNN-A	[-7, 7]	[-2, 2]	{-2, 2}	{-3, 4}	{0}	{0}	21.2	14.6
TDNN-B	[-9, 7]	[-2, 2]	{-2, 2}	{-5, 3}	{0}	{0}	21.2	14.5
TDNN-C	[-11, 7]	[-2, 2]	{-1, 1}	{-2, 2}	{-6, 2}	{0}	20.9	14.2
TDNN-D	[-13, 9]	[-2, 2]	{-1, 2}	{-3, 4}	{-7, 2}	{0}	20.8	14.0
TDNN-E	[-16, 9]	[-2, 2]	{-2, 2}	{-5, 3}	{-7, 2}	{0}	20.9	14.2

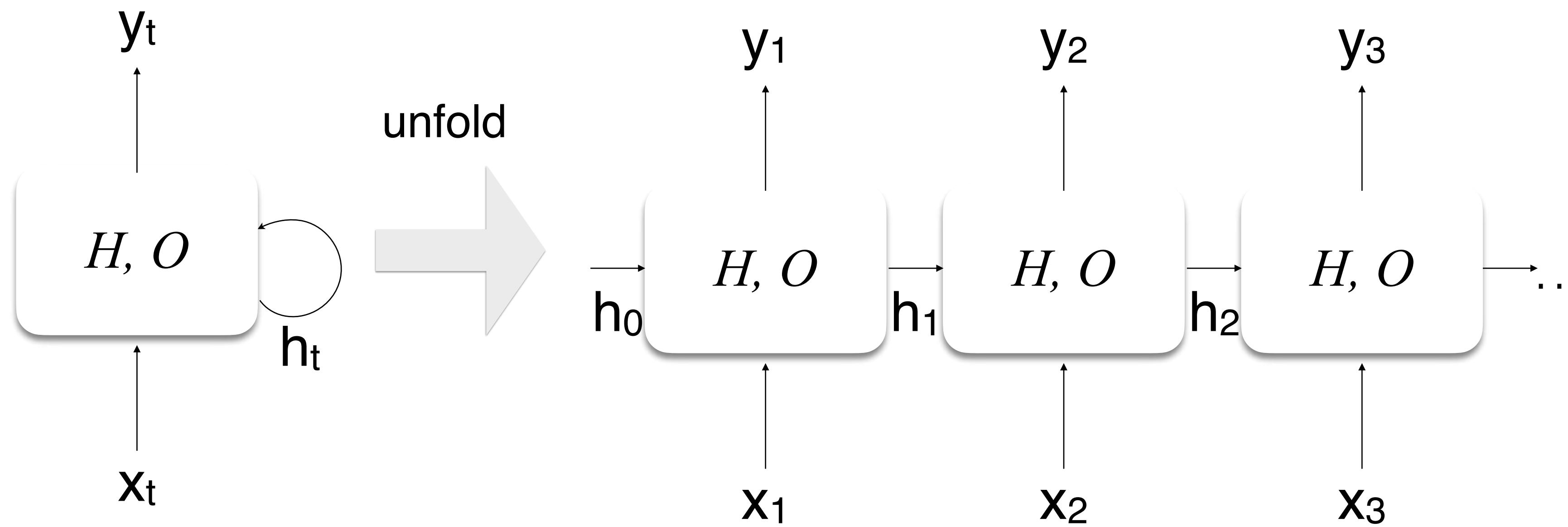
Feedforward DNNs we've seen so far...

- Assume independence among the training instances
 - Independent decision made about classifying each individual speech frame
 - Network state is completely reset after each speech frame is processed
- This independence assumption fails for data like speech which has temporal and sequential structure
- Two model architectures that capture longer ranges of acoustic context:
 1. Time delay neural networks (TDNNs)
 2. **Recurrent neural networks (RNNs)**

Recurrent Neural Networks

- Recurrent Neural Networks (RNNs) work naturally with sequential data and process it one element at a time
- HMMs also similarly attempt to model time dependencies.
How's it different?
- HMMs are limited by the size of the state space. Inference becomes intractable if the state space grows very large!
- What about RNNs?

RNN definition



Two main equations govern RNNs:

$$h_t = H(Wx_t + Vh_{t-1} + b^{(h)})$$

$$y_t = O(Uh_t + b^{(y)})$$

where W, V, U are matrices of input-hidden weights, hidden-hidden weights and hidden-output weights resp; $b^{(h)}$ and $b^{(y)}$ are bias vectors and H is the activation function applied to the hidden layer

Recurrent Neural Networks

- Recurrent Neural Networks (RNNs) work naturally with sequential data and process it one element at a time
- HMMs also similarly attempt to model time dependencies.
How's it different?
 - HMMs are limited by the size of the state space. Inference becomes intractable if the state space grows very large!
 - What about RNNs? RNNs are designed to capture long-range dependencies unlike HMMs: Network state is exponential in the number of nodes in a hidden layer

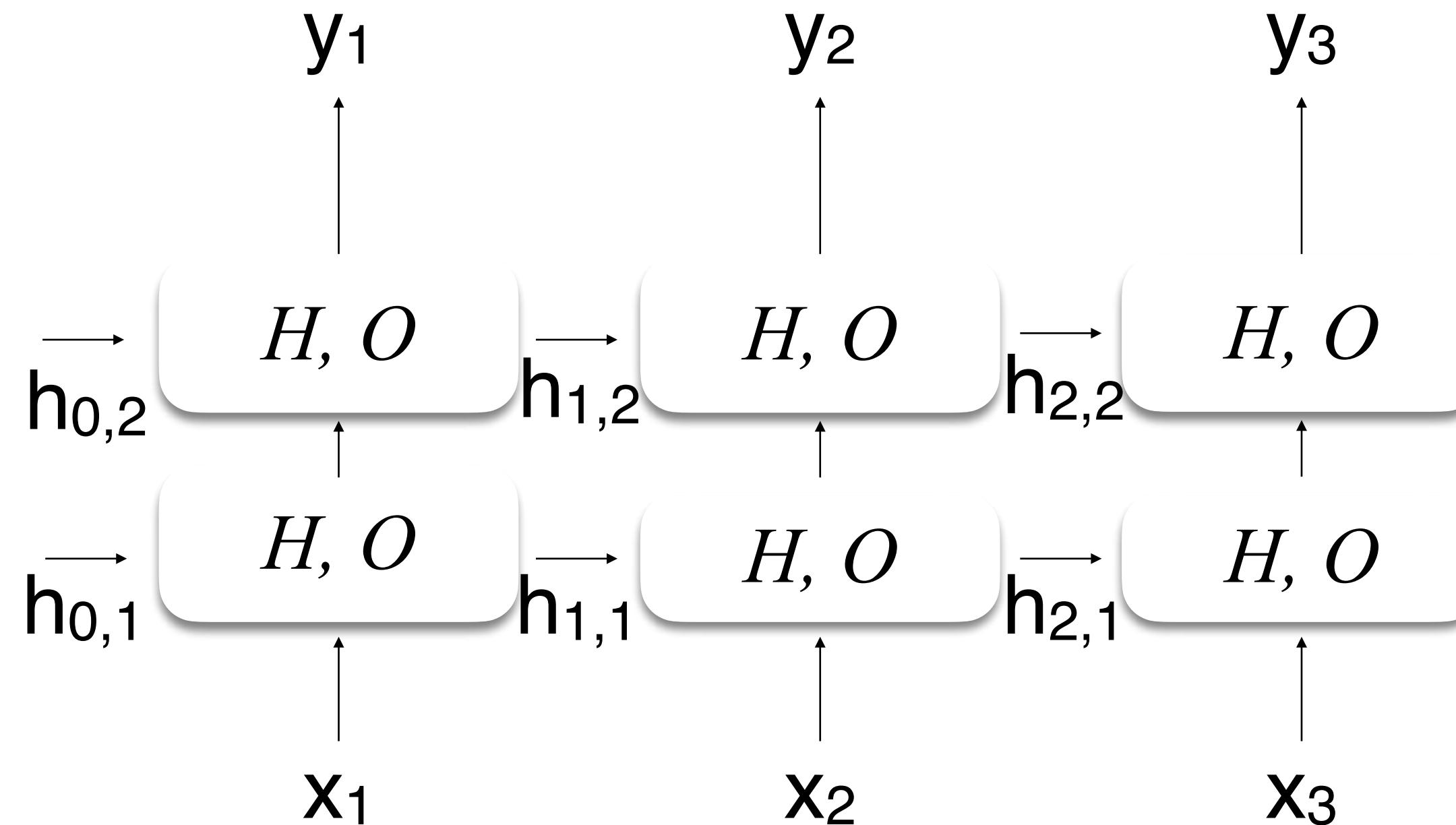
Training RNNs

- An unrolled RNN is just a very deep feedforward network
- For a given input sequence:
 - create the unrolled network
 - add a loss function node to the network
 - then, use backpropagation to compute the gradients

Training RNNs

- An unrolled RNN is just a very deep feedforward network
- For a given input sequence:
 - create the unrolled network
 - add a loss function node to the network
 - then, use backpropagation to compute the gradients
- This algorithm is known as backpropagation through time (BPTT)

Deep RNNs



- RNNs can be stacked in layers to form deep RNNs
- Empirically shown to perform better than shallow RNNs on ASR [G13]

Vanilla RNN Model

$$h_t = H(Wx_t + Vh_{t-1} + b^{(h)})$$

$$y_t = O(Uh_t + b^{(y)})$$

H : element wise application of the sigmoid or tanh function

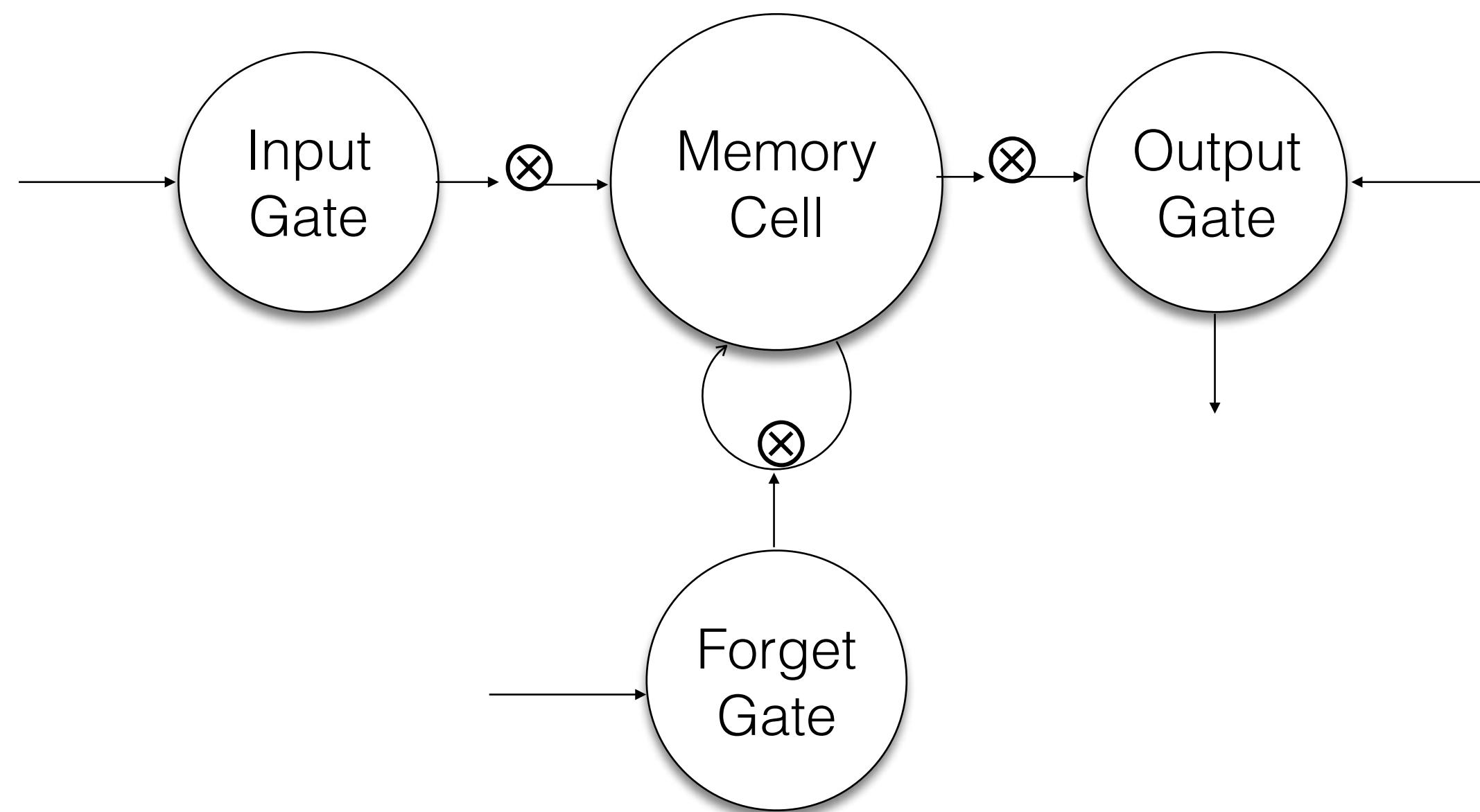
O : the softmax function

Run into problems of exploding and vanishing gradients.

Exploding/Vanishing Gradients

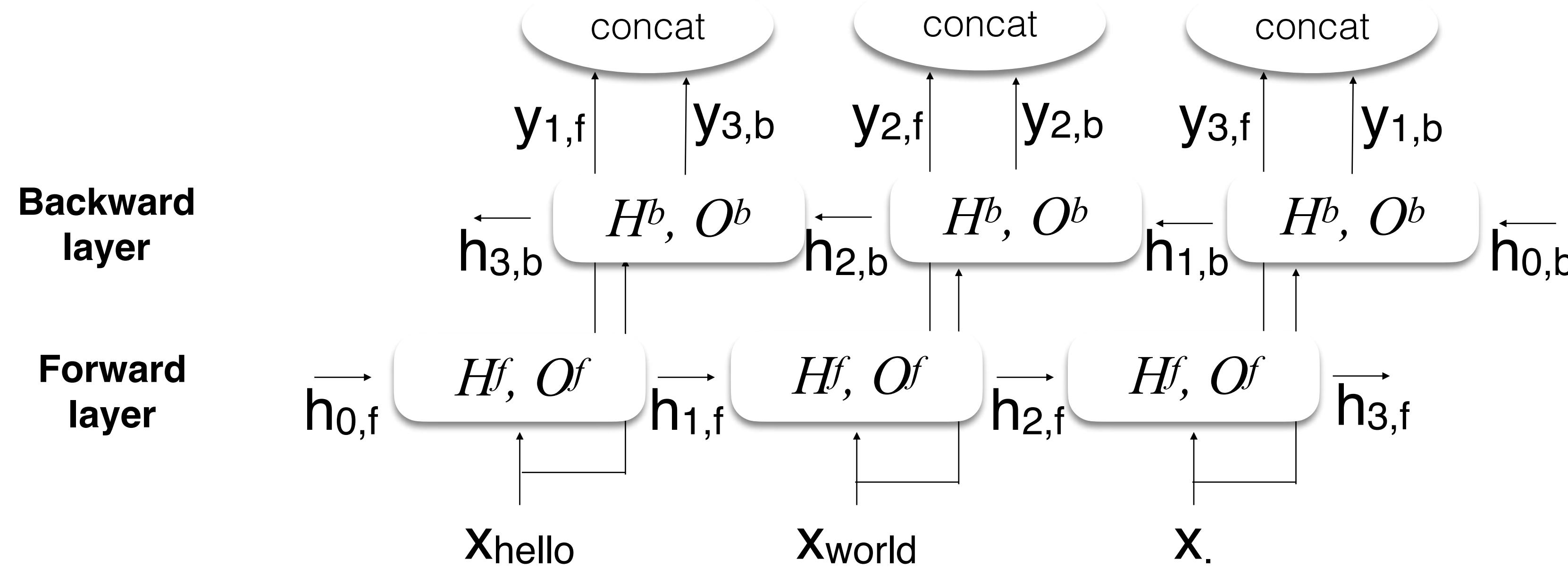
- In deep networks, gradients in early layers are computed as the product of terms from all the later layers
- This leads to unstable gradients:
 - If the terms in later layers are large enough, gradients in early layers (which is the product of these terms) can grow exponentially large: *Exploding gradients*
 - If the terms in later layers are small, gradients in early layers will tend to exponentially decrease: *Vanishing gradients*
- To address this problem in RNNs, Long Short Term Memory (LSTM) units were proposed [HS97]

Long Short Term Memory Cells



- Memory cell: Neuron that stores information over long time periods
- Forget gate: When on, memory cell retains previous contents.
Otherwise, memory cell forgets contents.
- When input gate is on, write into memory cell
- When output gate is on, read from the memory cell

Bidirectional RNNs

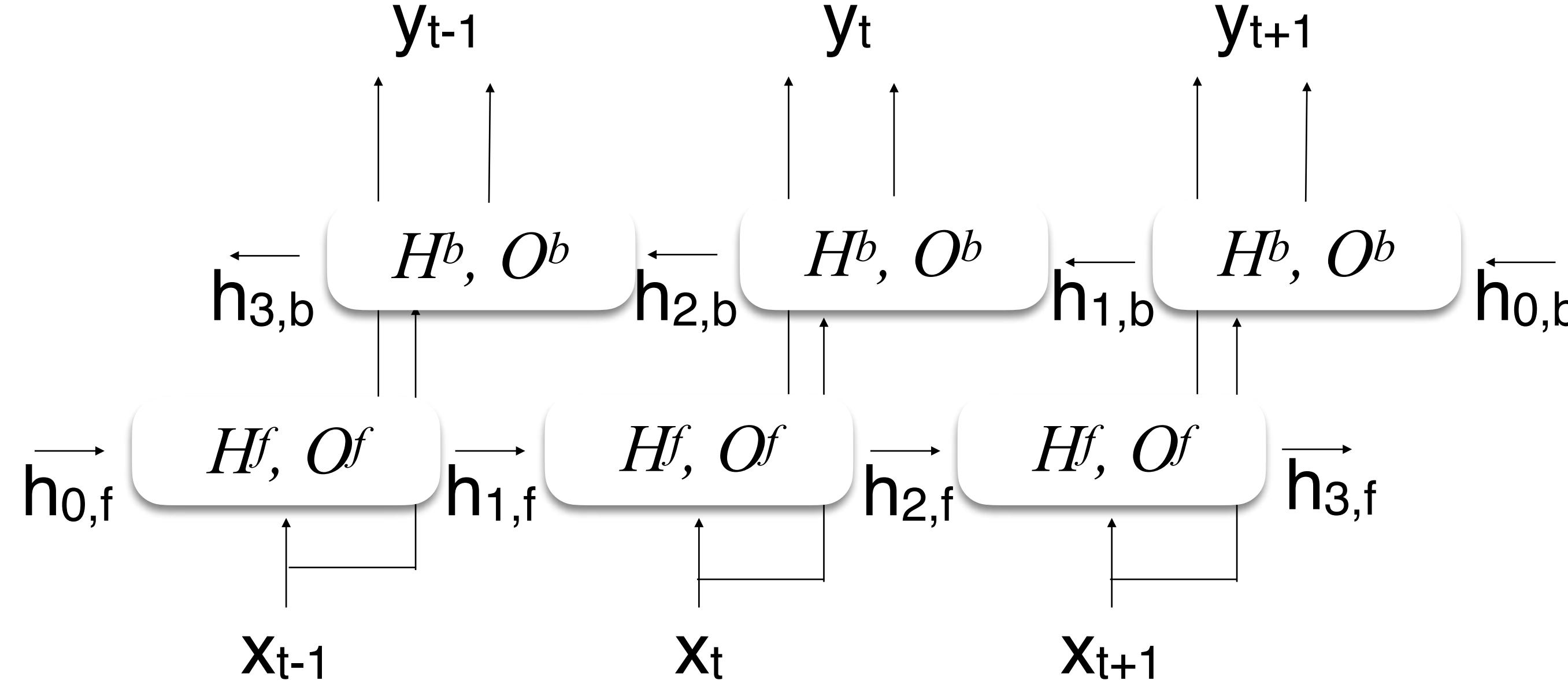


- BiRNNs process the data in both directions with two separate hidden layers
- Outputs from both hidden layers are concatenated at each position

ASR with RNNs

- We have seen how neural networks can be used for acoustic models in ASR systems
- Main limitation: Frame-level training targets derived from HMM-based alignments
- Goal: Single RNN model that addresses this issues and does not rely on HMM-based alignments [G14]

RNN-based Acoustic Model



- H was implemented using LSTMs in [G13]. Input: Acoustic feature vectors, one per frame; Output: Characters + space
- Deep bidirectional LSTM networks were used to do phone recognition on TIMIT
- Trained using the Connectionist Temporal Classification (CTC) loss [covered in later class]

RNN-based Acoustic Model

NETWORK	WEIGHTS	EPOCHS	PER
CTC-3L-500H-TANH	3.7M	107	37.6%
CTC-1L-250H	0.8M	82	23.9%
CTC-1L-622H	3.8M	87	23.0%
CTC-2L-250H	2.3M	55	21.0%
CTC-3L-421H-UNI	3.8M	115	19.6%
CTC-3L-250H	3.8M	124	18.6%
CTC-5L-250H	6.8M	150	18.4%

TIMIT phoneme recognition results

RNNs For Language Models?

Word representations in Ngram models

- In standard Ngram models, words are represented in the discrete space involving the vocabulary
- Limits the possibility of truly interpolating probabilities of unseen Ngrams
- Can we build a representation for words in the continuous space?

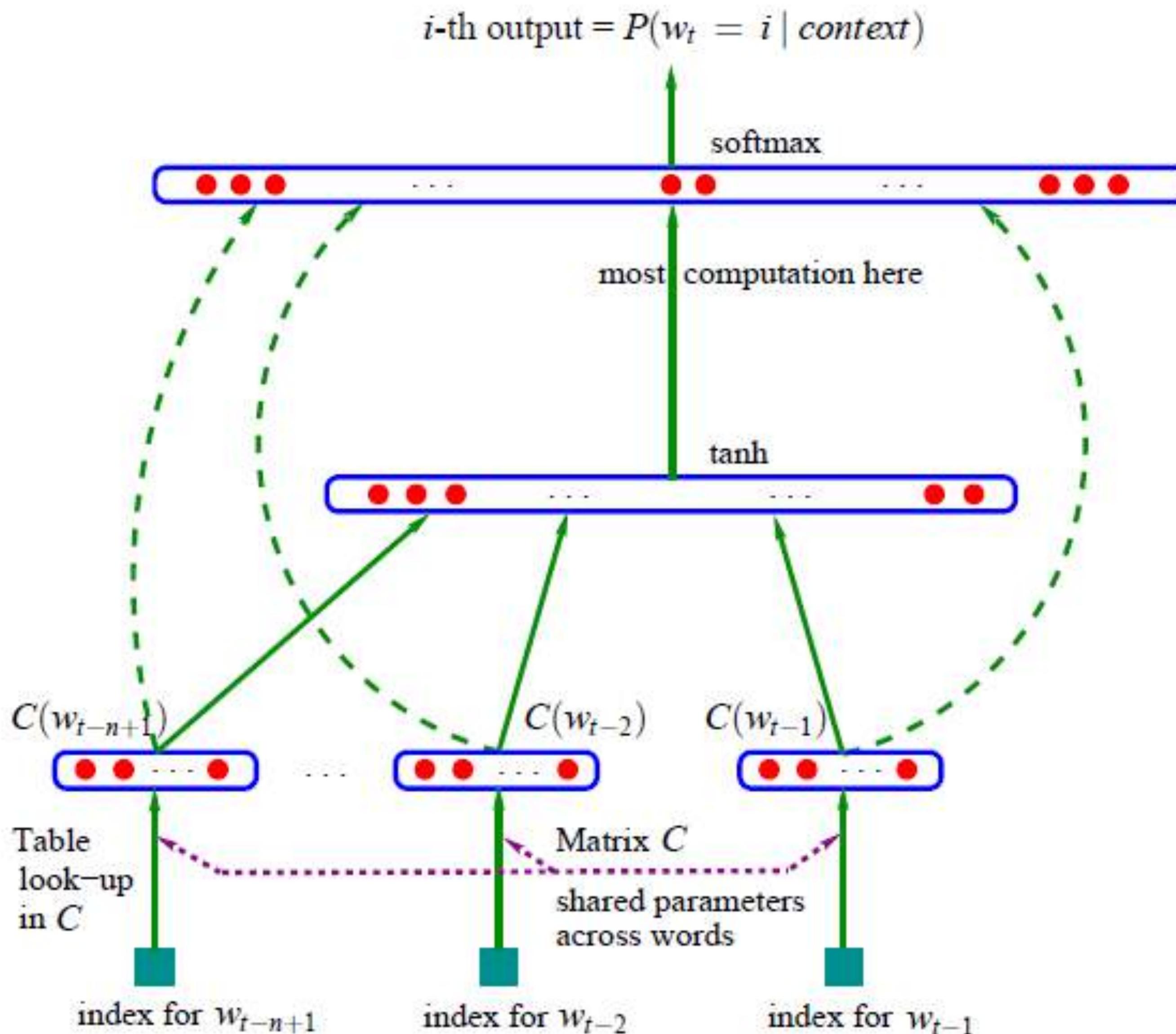
Word representations

- 1-hot representation:
 - Each word is given an index in $\{1, \dots, V\}$. The 1-hot vector $f_i \in R^V$ contains zeros everywhere except for the i^{th} dimension being 1
 - 1-hot form, however, doesn't encode information about word similarity
 - Distributed (or continuous) representation: Each word is associated with a dense vector. Based on the “distributional hypothesis”.
E.g. dog $\rightarrow \{-0.02, -0.37, 0.26, 0.25, -0.11, 0.34\}$

Word embeddings

- These distributed representations in a continuous space are also referred to as “word embeddings”
 - Low dimensional
 - Similar words will have similar vectors
- Word embeddings capture semantic properties (such as *man* is to *woman* as *boy* is to *girl*, etc.) and morphological properties (*glad* is similar to *gladly*, etc.)
- The word embeddings could be learned via the first layer of a neural network [B03].

Word embeddings

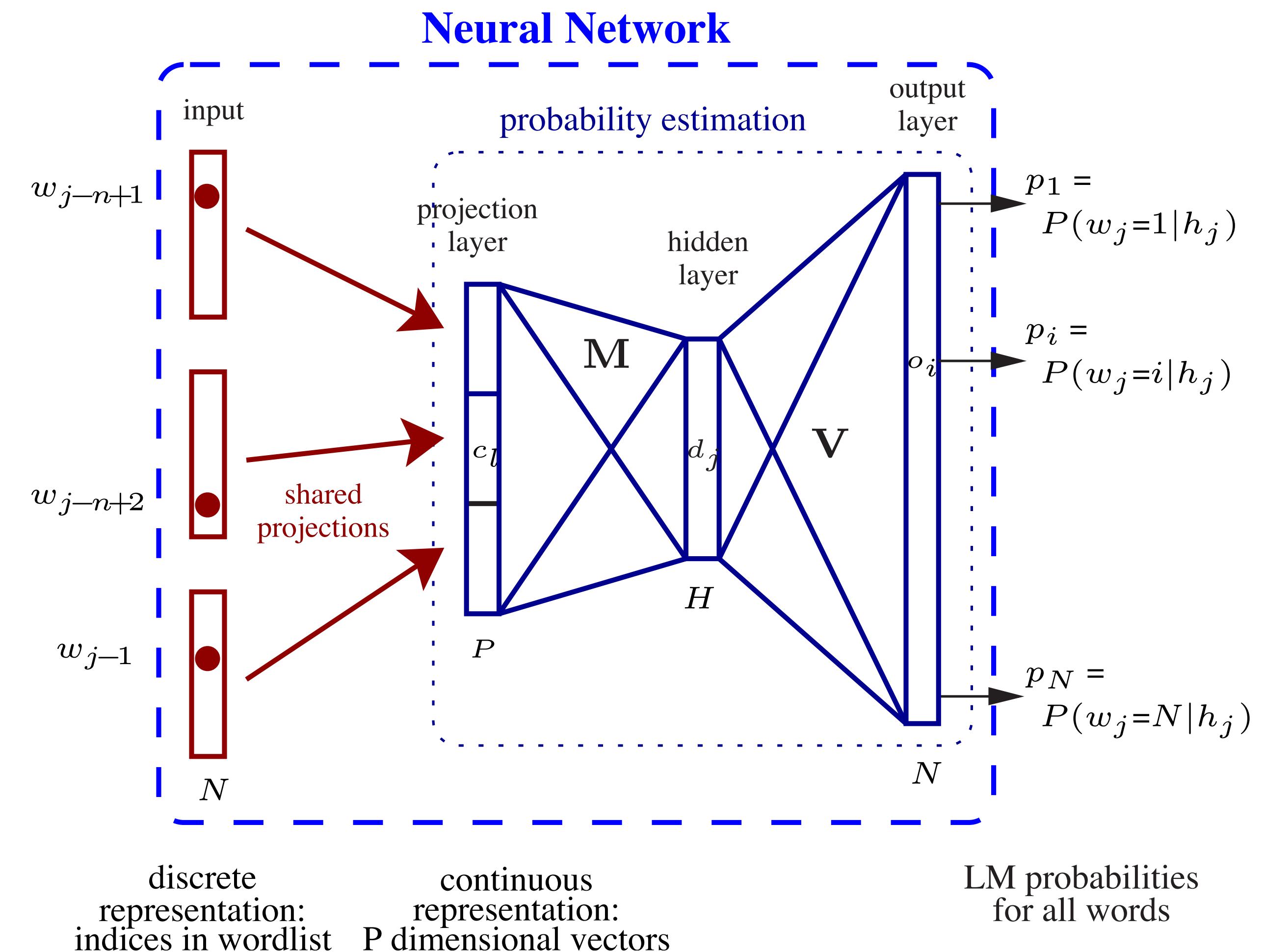


Introduced the architecture that forms the basis of all current neural language and word embedding models

- Embedding layer
- One or more middle/hidden layers
- Softmax output layer

NN language model

- Project all the words of the context $h_j = w_{j-n+1}, \dots, w_{j-1}$ to their dense forms
- Then, calculate the language model probability $\Pr(w_j = i | h_j)$ for the given context h_j



NN language model

- Dense vectors of all the words in context are concatenated forming the first hidden layer of the neural network

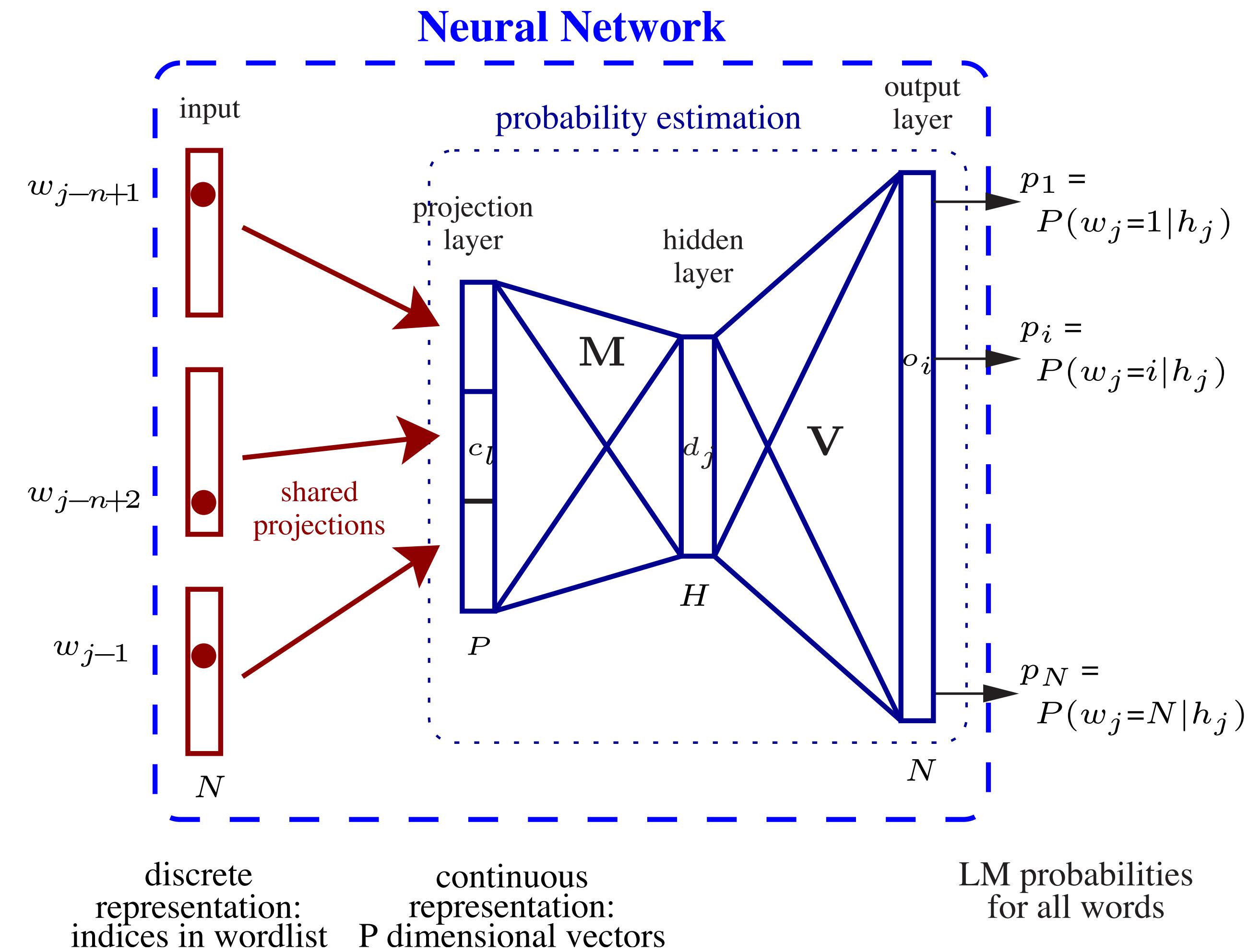
- Second hidden layer:

$$d_j = \tanh\left(\sum_l m_{jl} c_l + b_j\right) \quad \forall j = 1, \dots, H$$

- Output layer:

$$o_i = \sum_j v_{ij} d_j + b'_i \quad \forall i = 1, \dots, N$$

- $p_i \rightarrow$ softmax output from the i^{th} neuron $\rightarrow \Pr(w_j = i | h_j)$



NN language model

- Model is trained to minimise the following loss function:

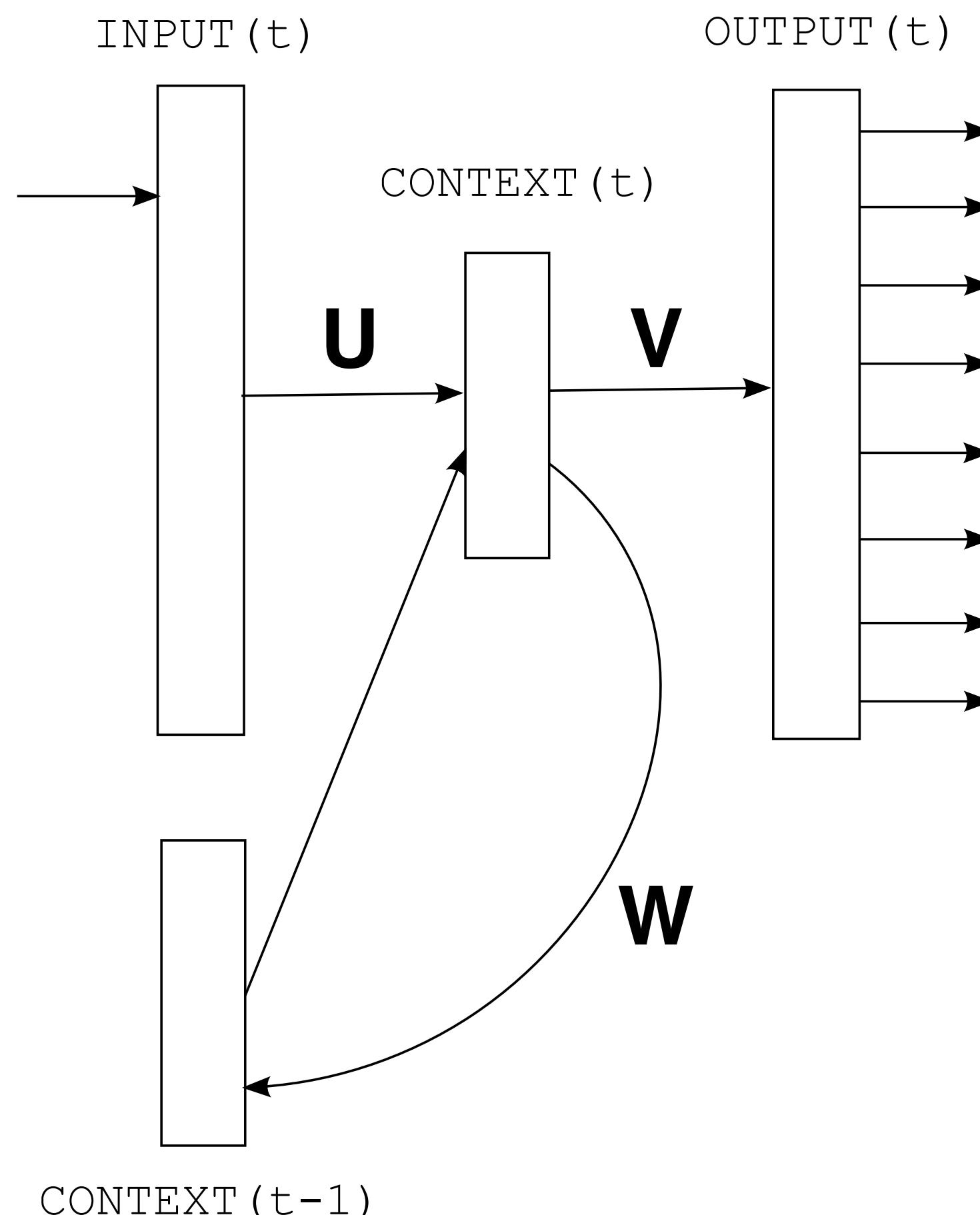
$$L = \sum_{i=1}^N t_i \log p_i + \epsilon \left(\sum_{kl} m_{kl}^2 + \sum_{ik} v_{ik}^2 \right)$$

- Here, t_i is the target output 1-hot vector (1 for next word in the training instance, 0 elsewhere)
- First part: Cross-entropy between the target distribution and the distribution estimated by the NN
- Second part: Regularization term

Longer word context?

- What have we seen so far: A feedforward NN used to compute an Ngram probability $\Pr(w_j = i|h_j)$ (where h_j encodes the Ngram history)
- We know Ngrams are limiting:
Alice who had attempted *the assignment asked* the lecturer
- How can we predict the next word based on the entire sequence of preceding words? Use recurrent neural networks (RNNs)

Simple RNN language model



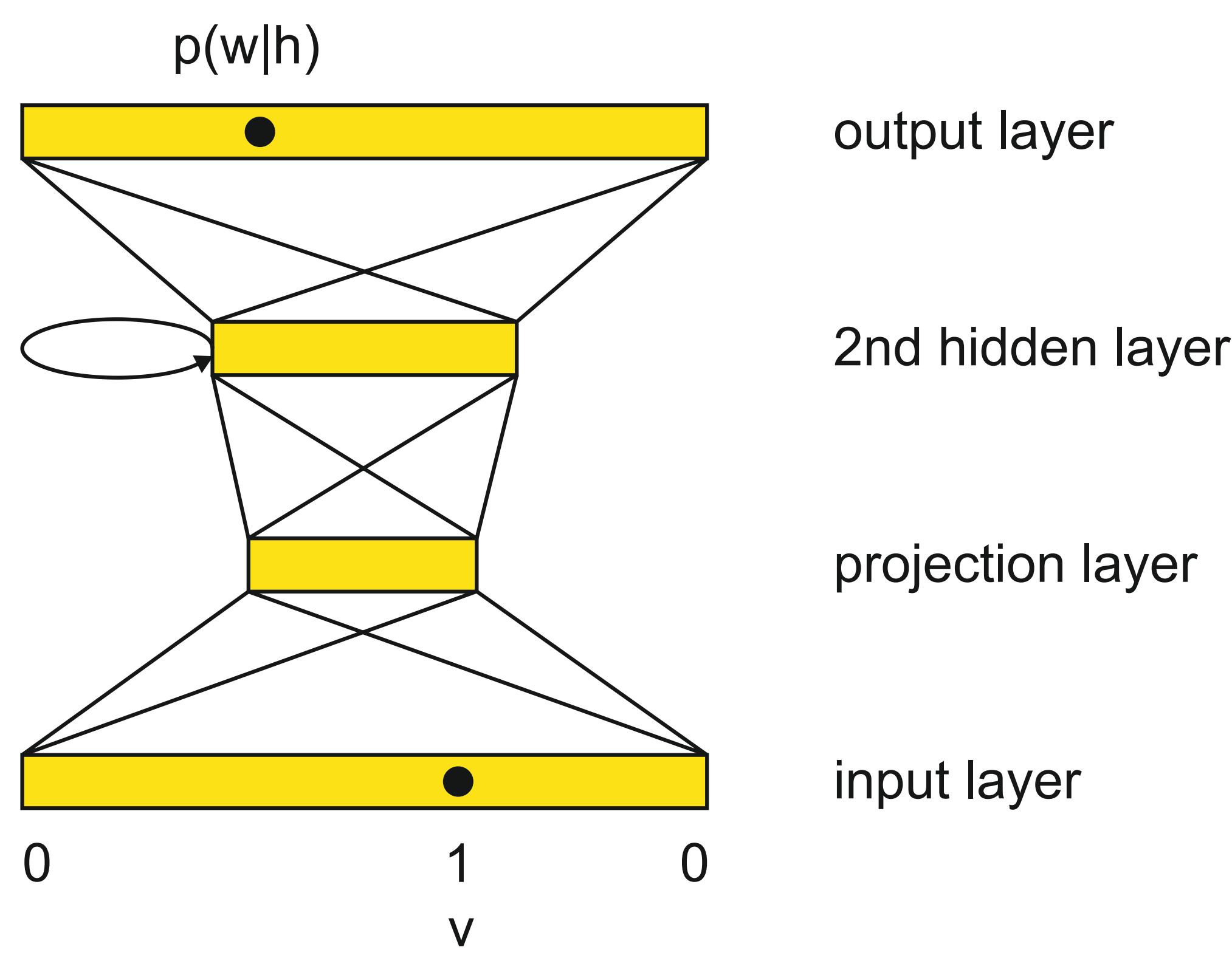
- Current word, x_t
Hidden state, s_t
Output, y_t

$$s_t = f(Ux_t + Ws_{t-1})$$

$$o_t = \text{softmax}(Vs_t)$$

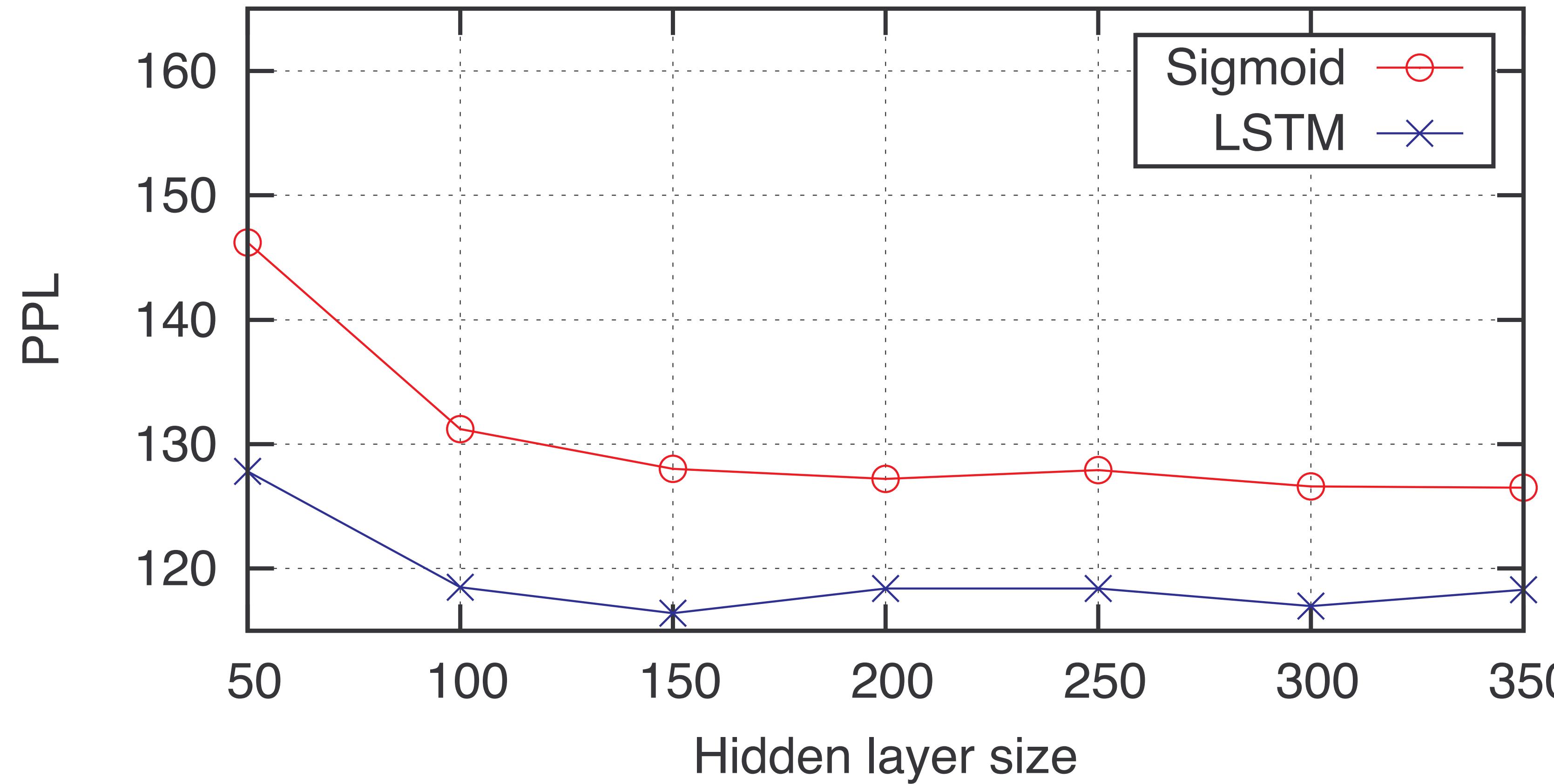
- RNN is trained using the cross-entropy criterion

LSTM-LMs



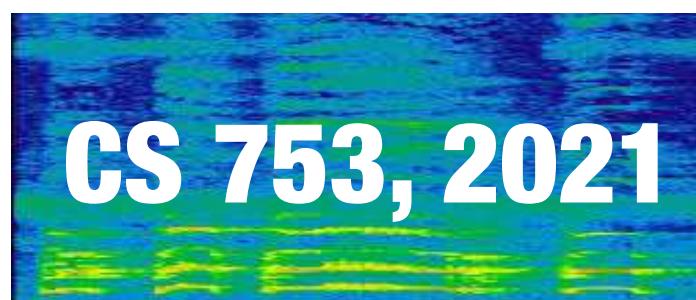
- Vanilla RNN-LMs unlikely to show full potential of recurrent models due to issues like vanishing gradients
- LSTM-LMs: Similar to RNN-LMs except use LSTM units in the 2nd hidden (recurrent) layer

Comparing RNN-LMs with LSTM-LMs



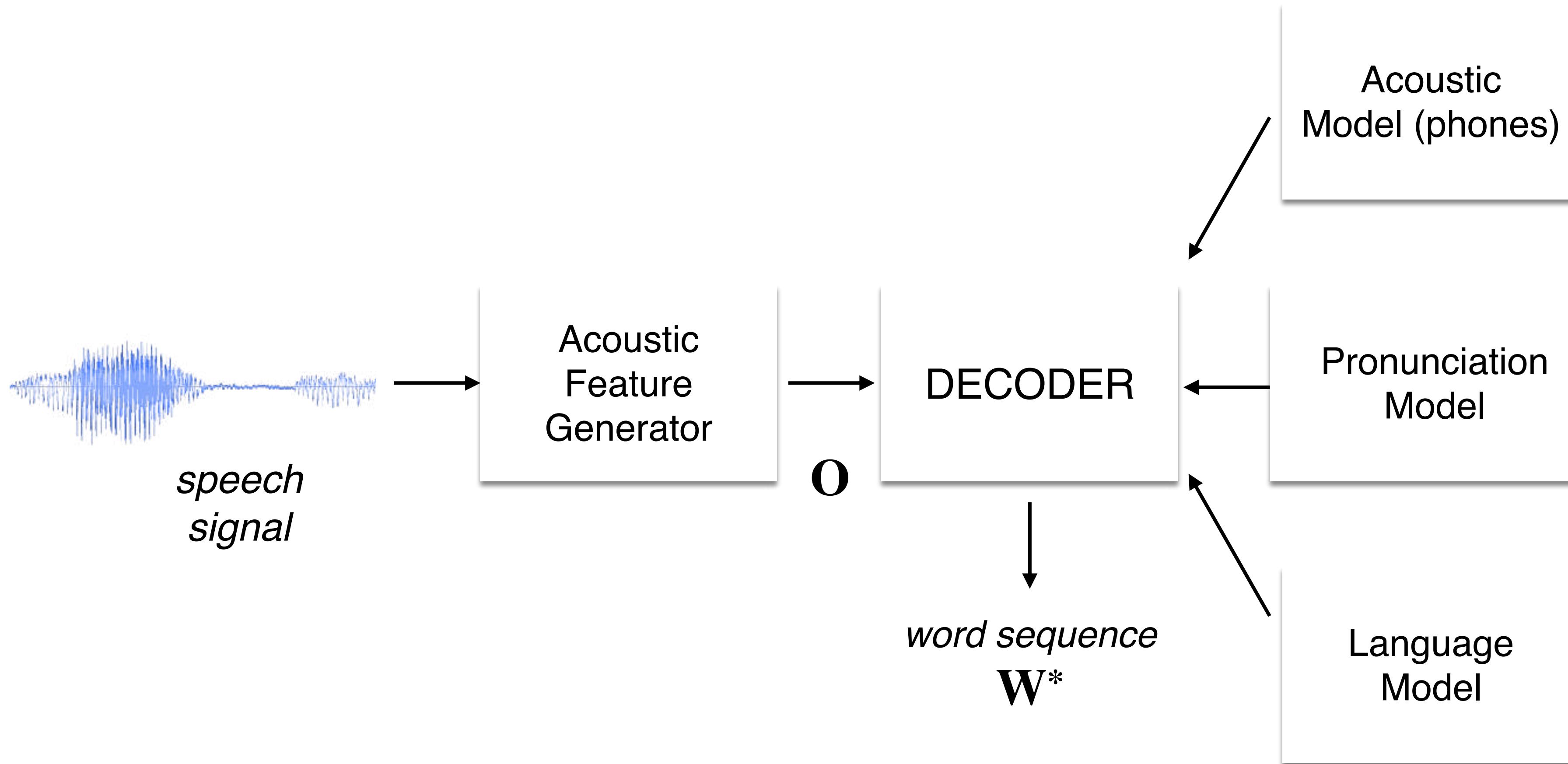
End-to-End Neural ASR Systems (I)

Lecture 7a



Instructor: Preethi Jyothi, IITB

Architecture of a Cascaded ASR system

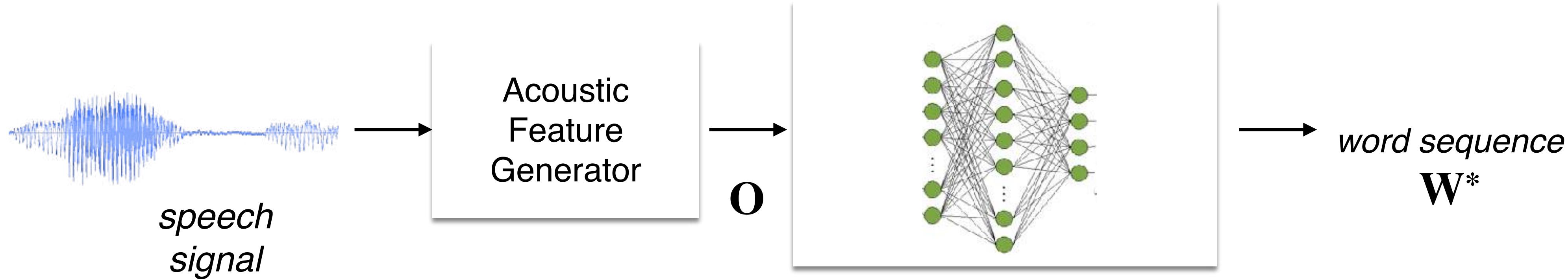


Limitations of Cascaded ASR Systems

- Frame-level training targets derived from HMM-based alignments
- Pronunciation dictionaries are used to map from words to phonemes; expensive resource to create
- Complex training process, and difficult to globally optimise
- [Limitation not specific to cascaded systems per se]
Objective function optimized in neural networks very different from final evaluation metric (i.e. word transcription accuracy)

Cascaded ASR \Rightarrow End-to-end ASR

$$\mathbf{W}^* = \arg \max_W \Pr(\mathbf{W} | \mathbf{O})$$



Single end-to-end model that directly learns a mapping from speech to text

End-to-End ASR Systems

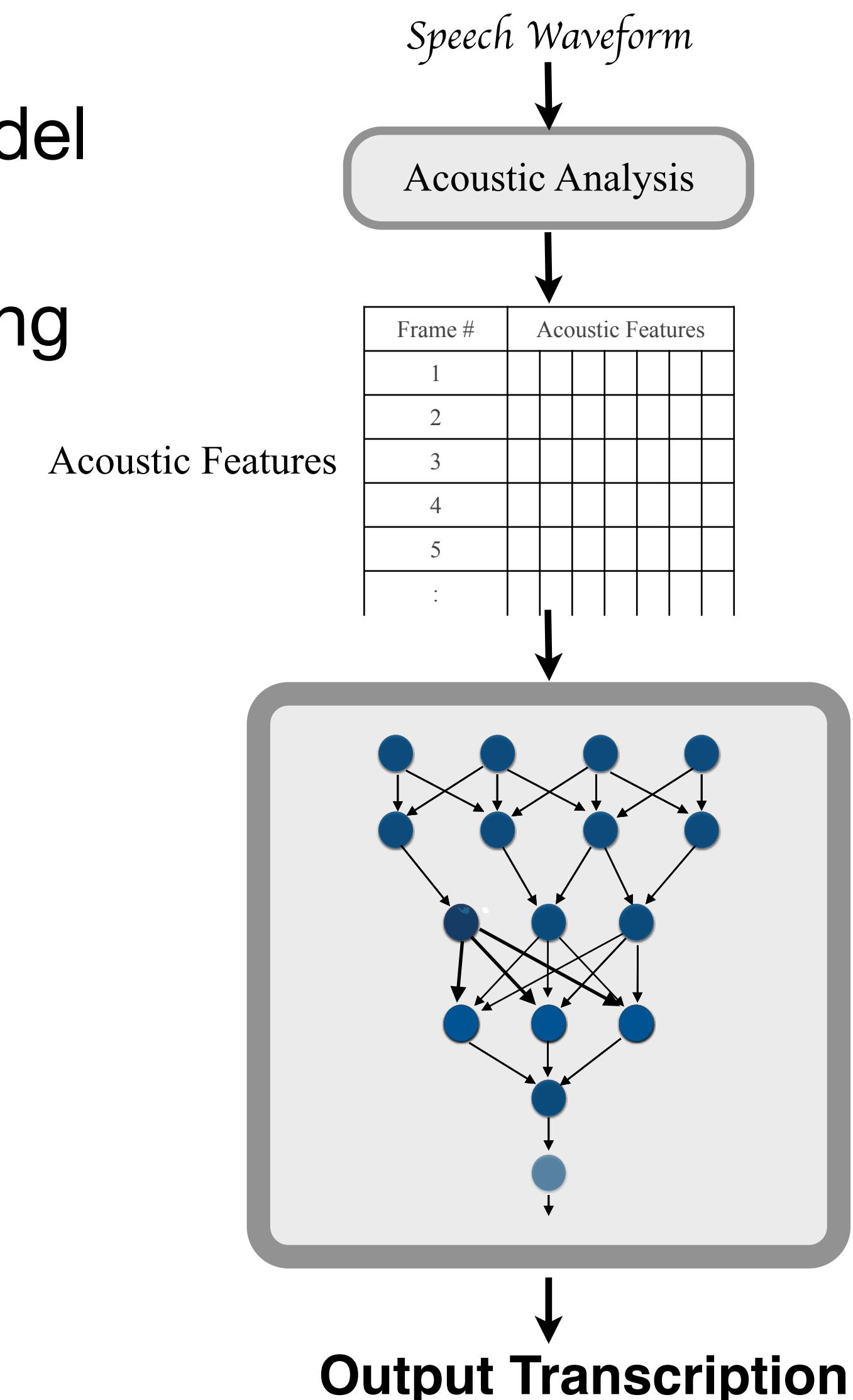
- All components trained jointly as a single end-to-end model
- Trained using pairs of speech clips and their corresponding text transcripts
- End-to-end models, with sufficient data, sometimes outperform conventional ASR systems [1,2]

	dev	test
DNN-HMM	4.0	4.4
E2E (Attention)	4.7	4.8

Librispeech-960

	dev	test
DNN-HMM	5.0	5.8
E2E (Attention)	14.7	14.7

Librispeech-100



[1] “State-of-the-art speech recognition with sequence-to-sequence models”, Chiu et al., 2018

[2] “RWTH ASR Systems for LibriSpeech: Hybrid vs Attention”, Lüscher, Christoph, et al. , 2019

A fully end-to-end system

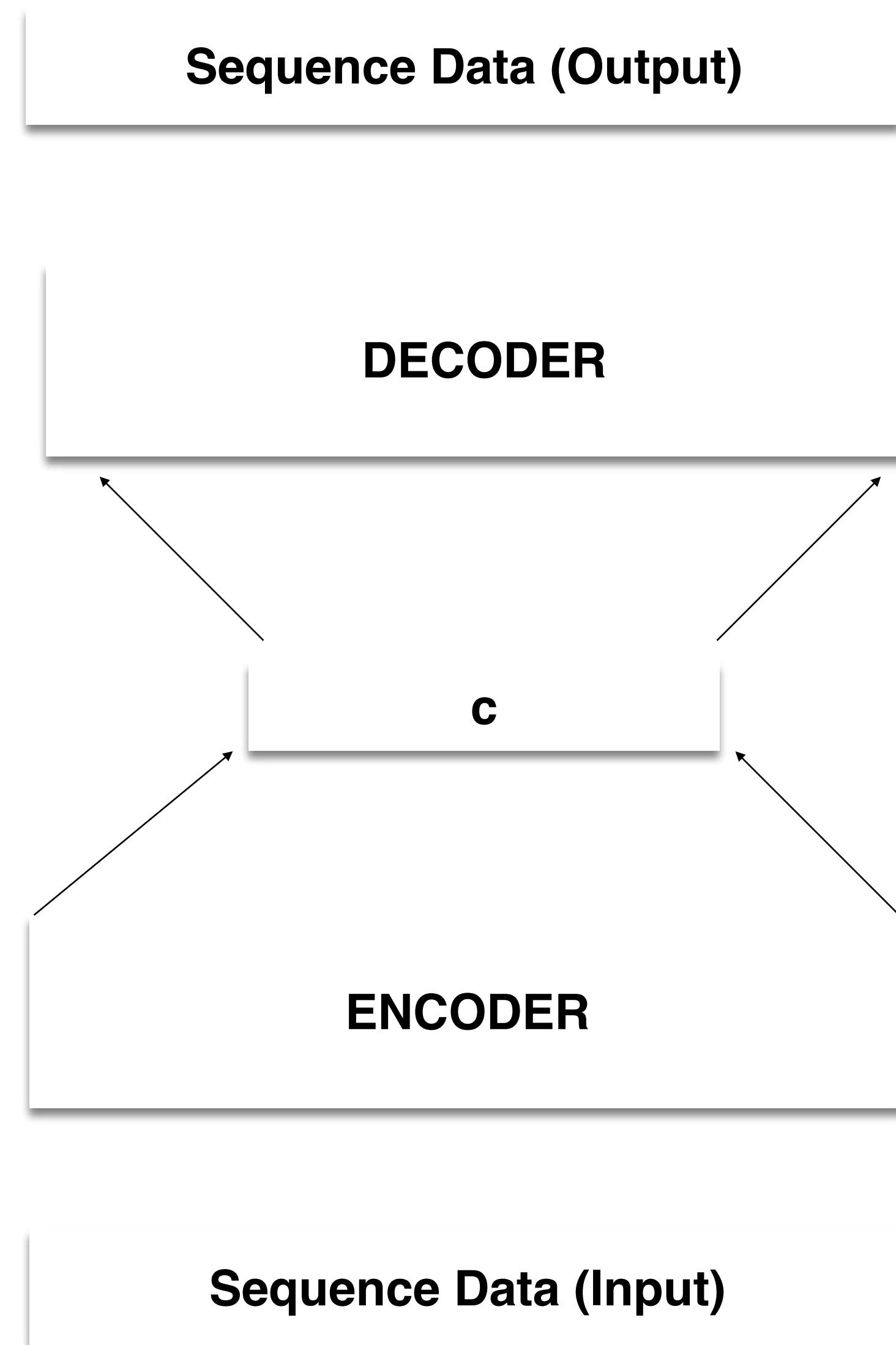
- Build an end-to-end (e2e) model that subsumes all the standard ASR components (ideally, without any additional language model during decoding)
- Attention-based e2e model: Makes *no independence assumptions* (unlike the CTC models) about the probability distribution of the output sequences given the input

$$P(\mathbf{y}|\mathbf{x}) = \prod_i P(y_i|\mathbf{x}, y_{<i})$$

- Based on the sequence-to-sequence learning framework with attention

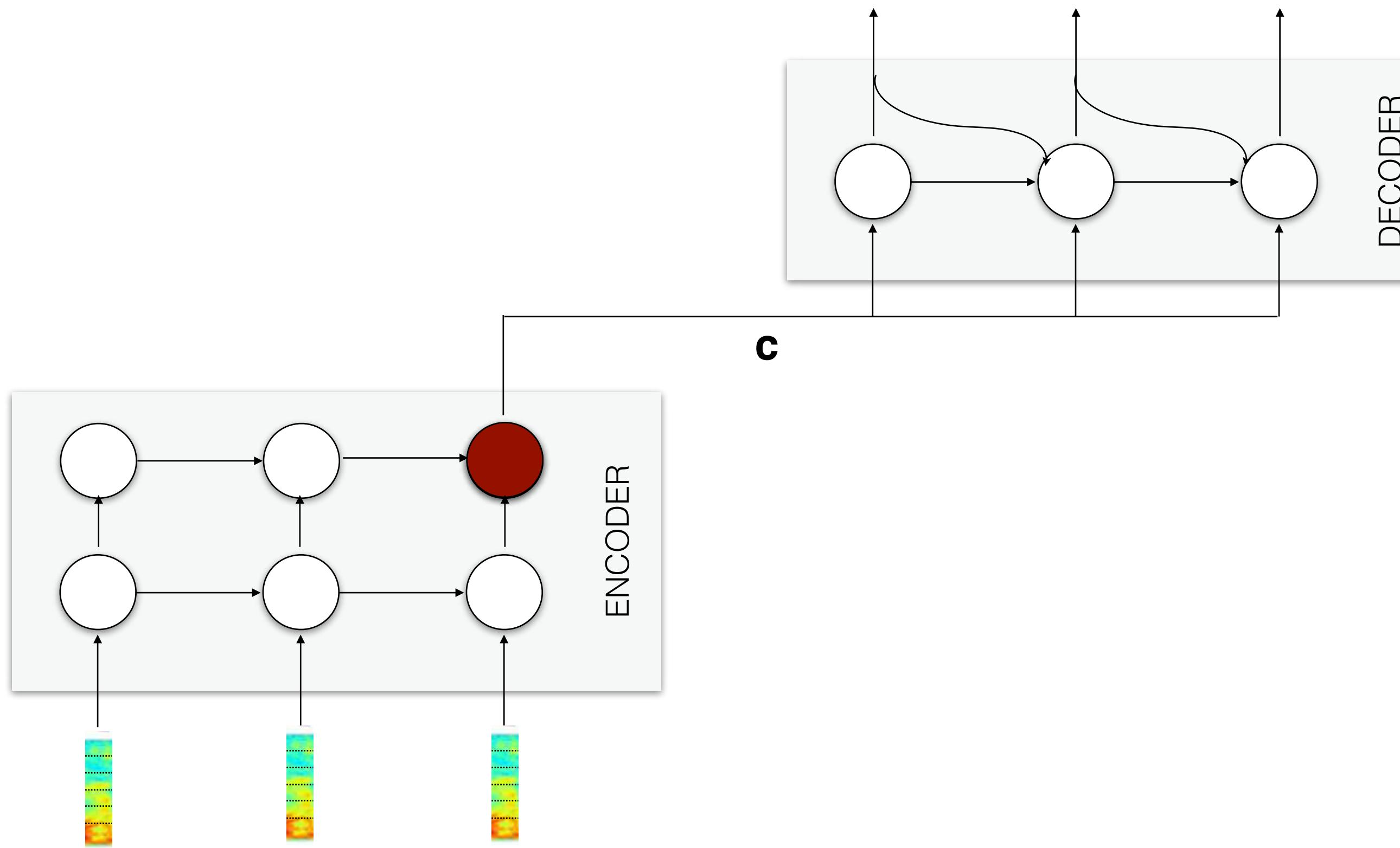
Sequence to sequence models

Encoder-decoder architecture

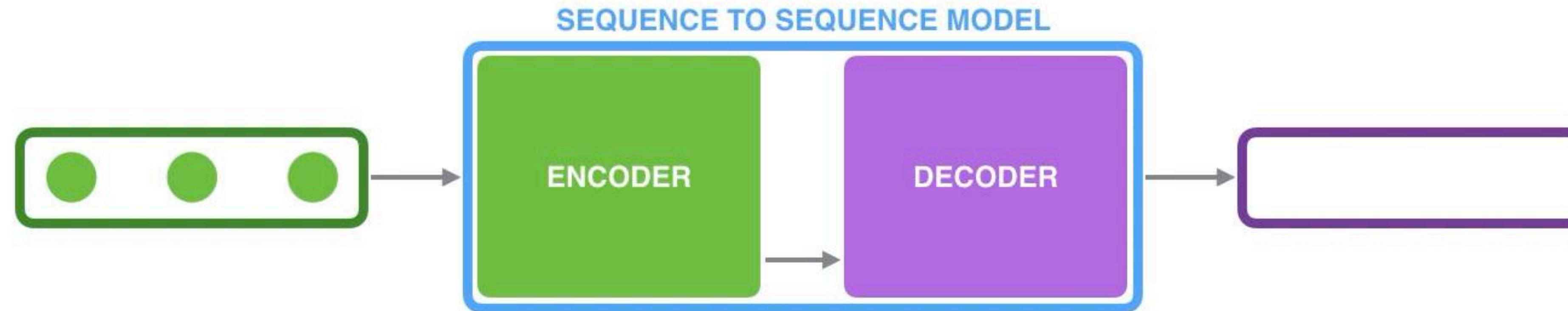


Sequence to sequence models

Encoder-decoder architecture

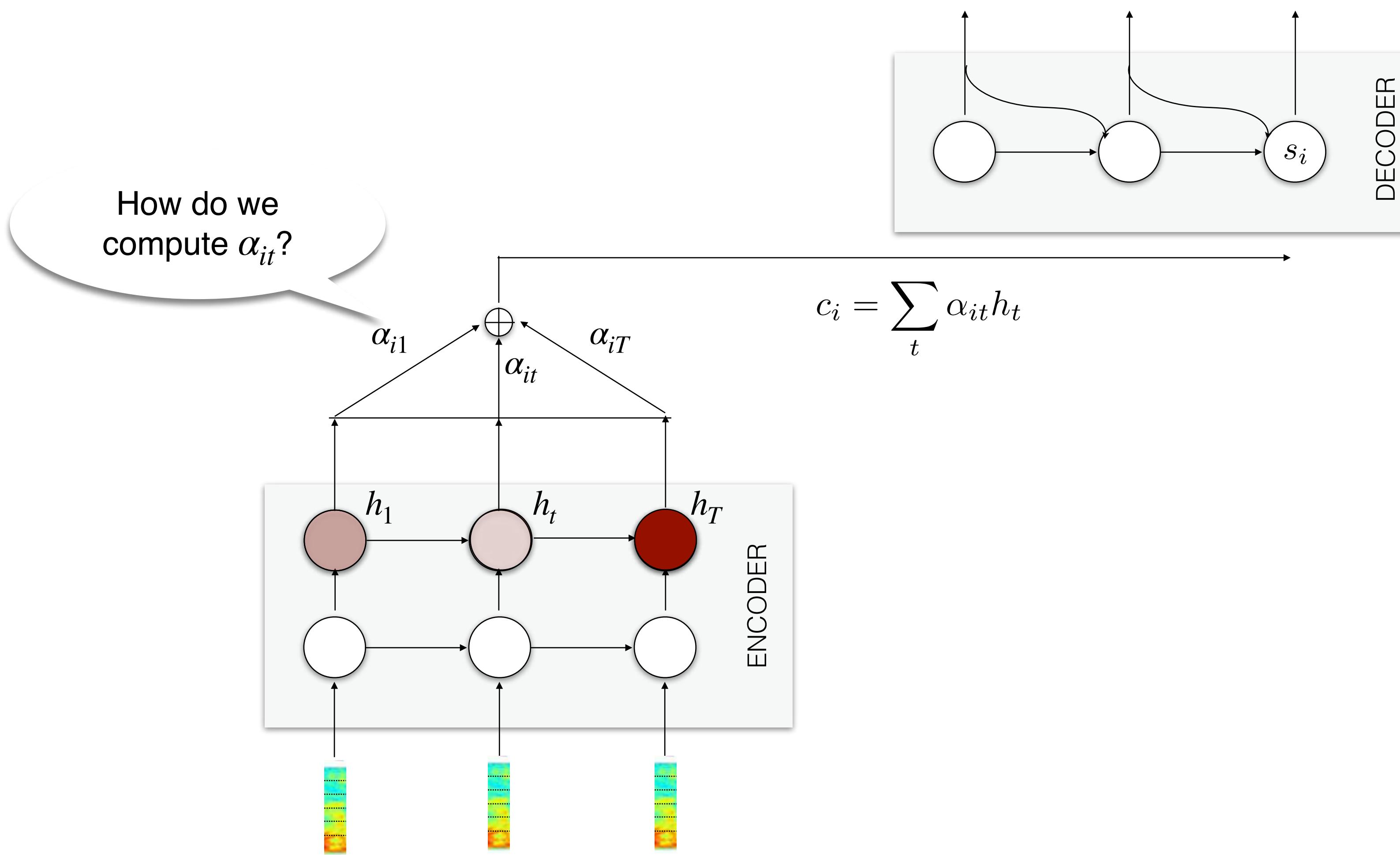


Sequence to sequence model



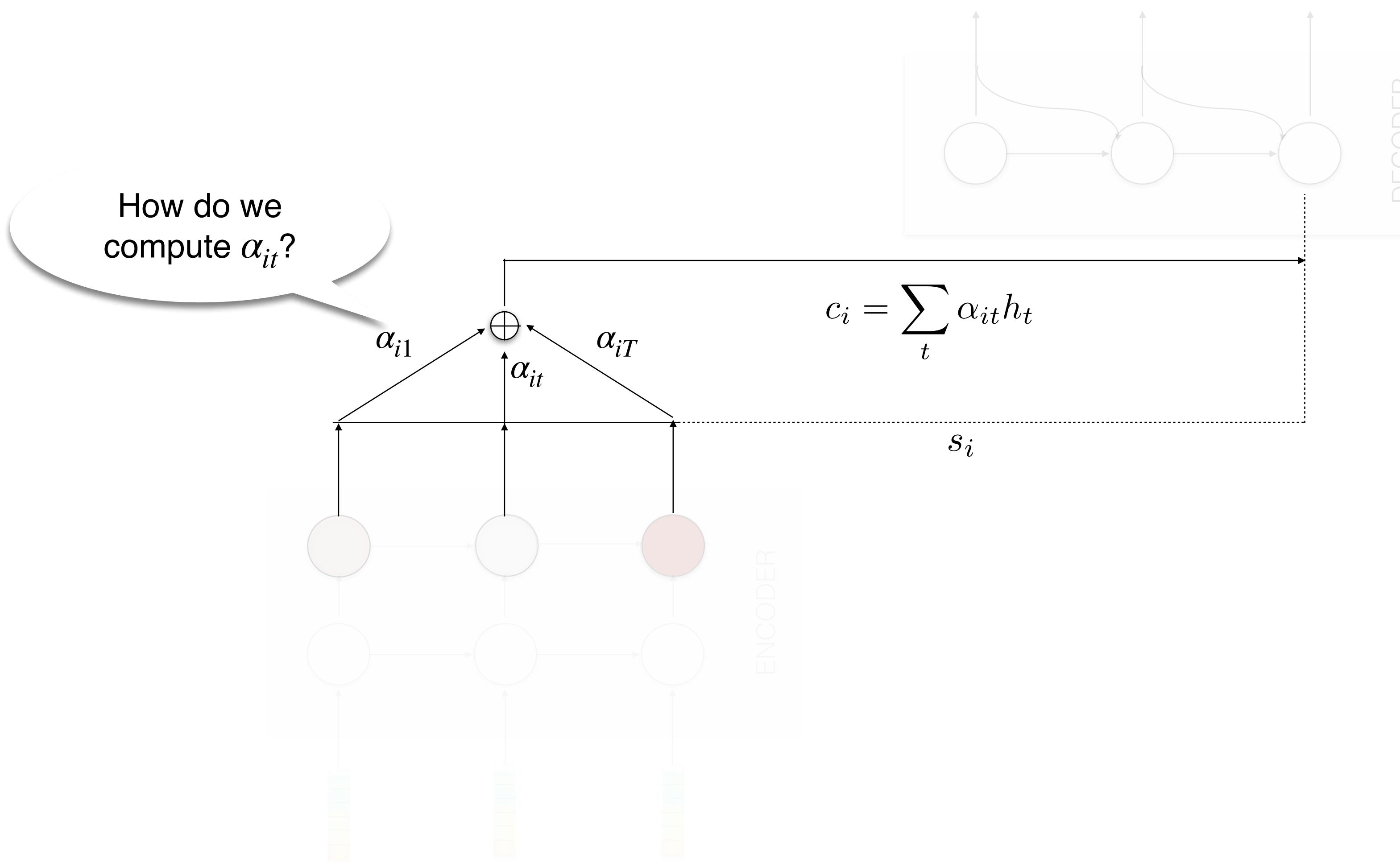
Sequence to sequence models

Encoder-decoder architecture with attention

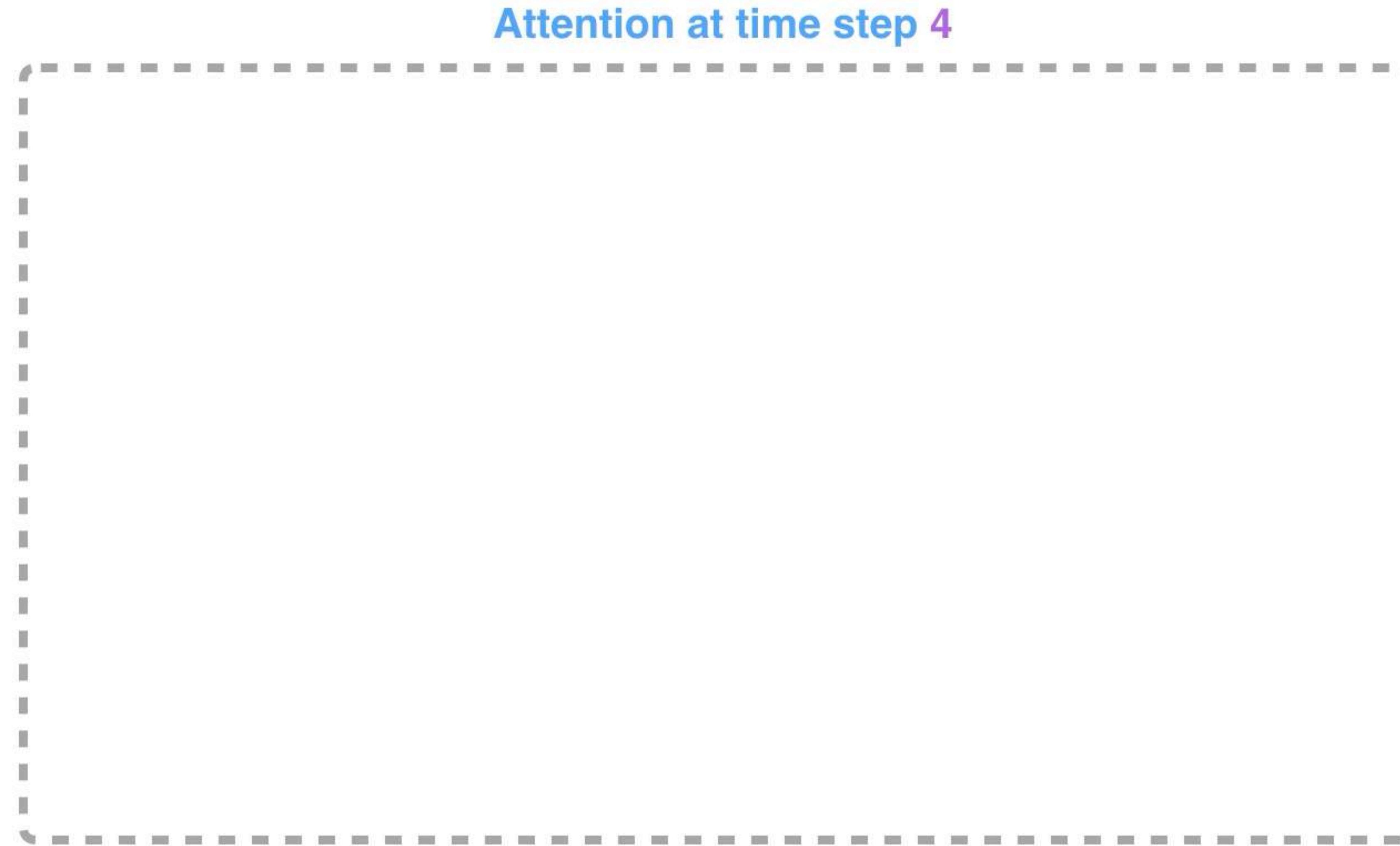


Sequence to sequence models

Encoder-decoder architecture with attention

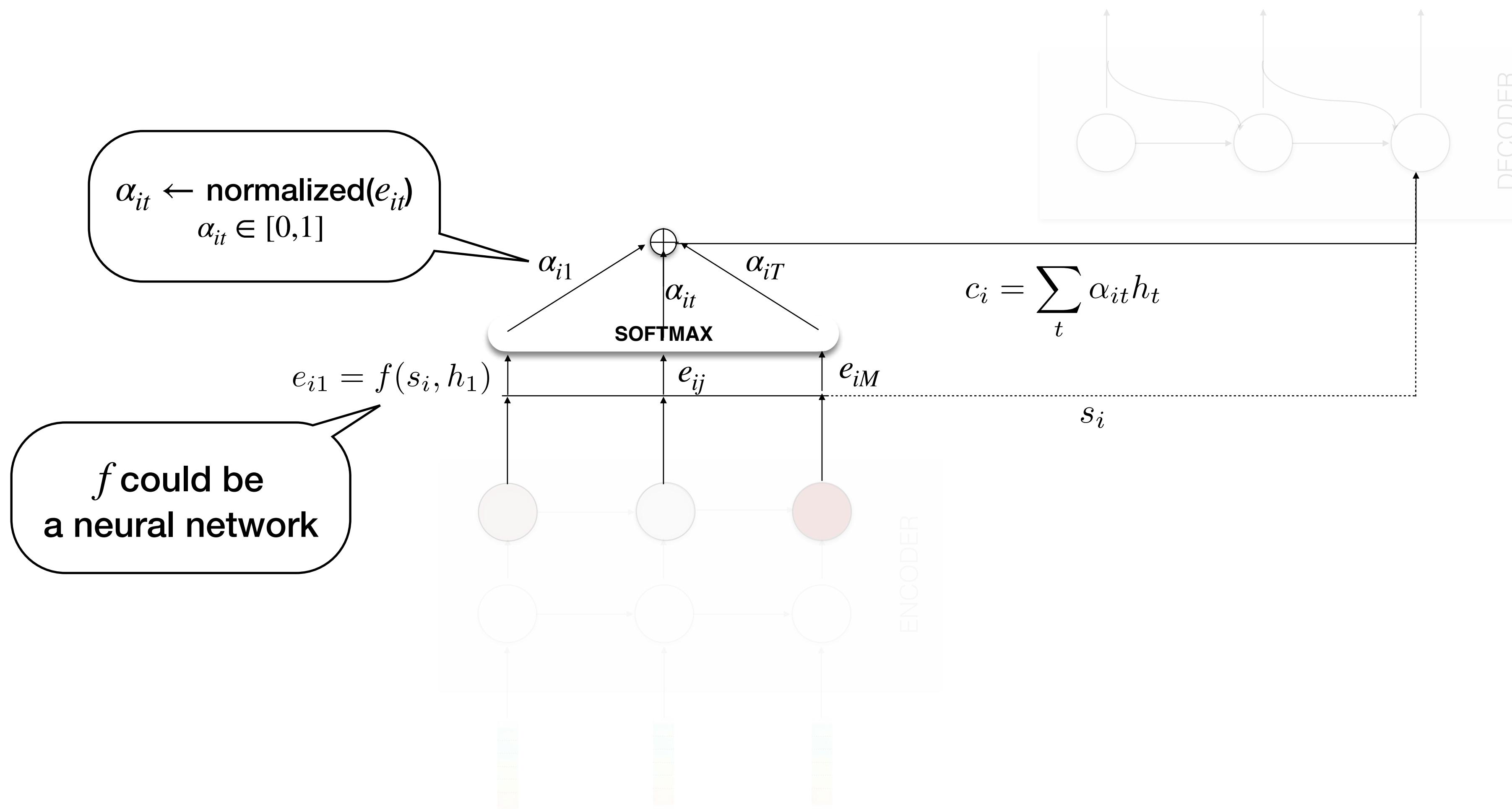


Sequence to sequence model with Attention

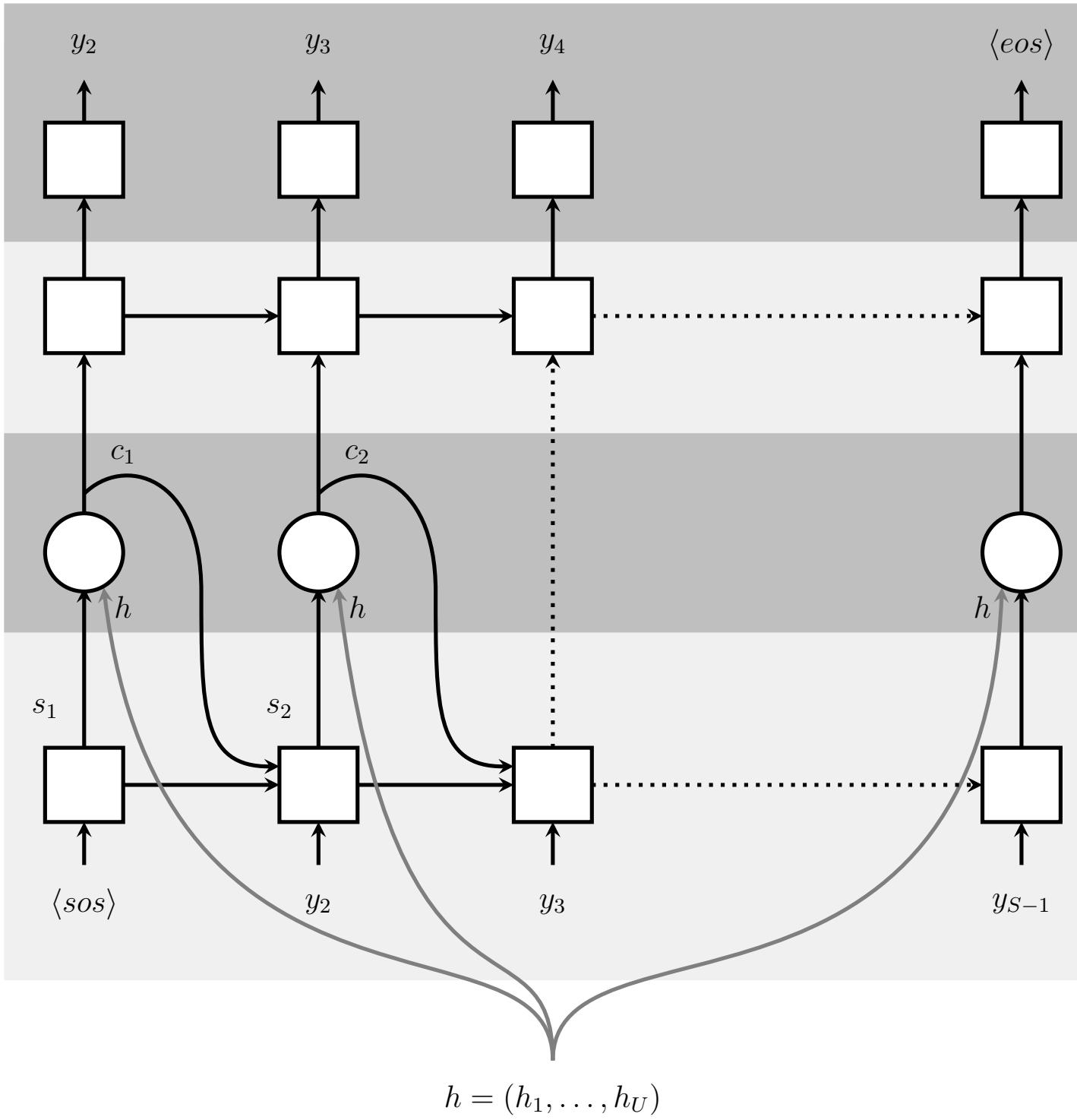


Sequence to sequence models

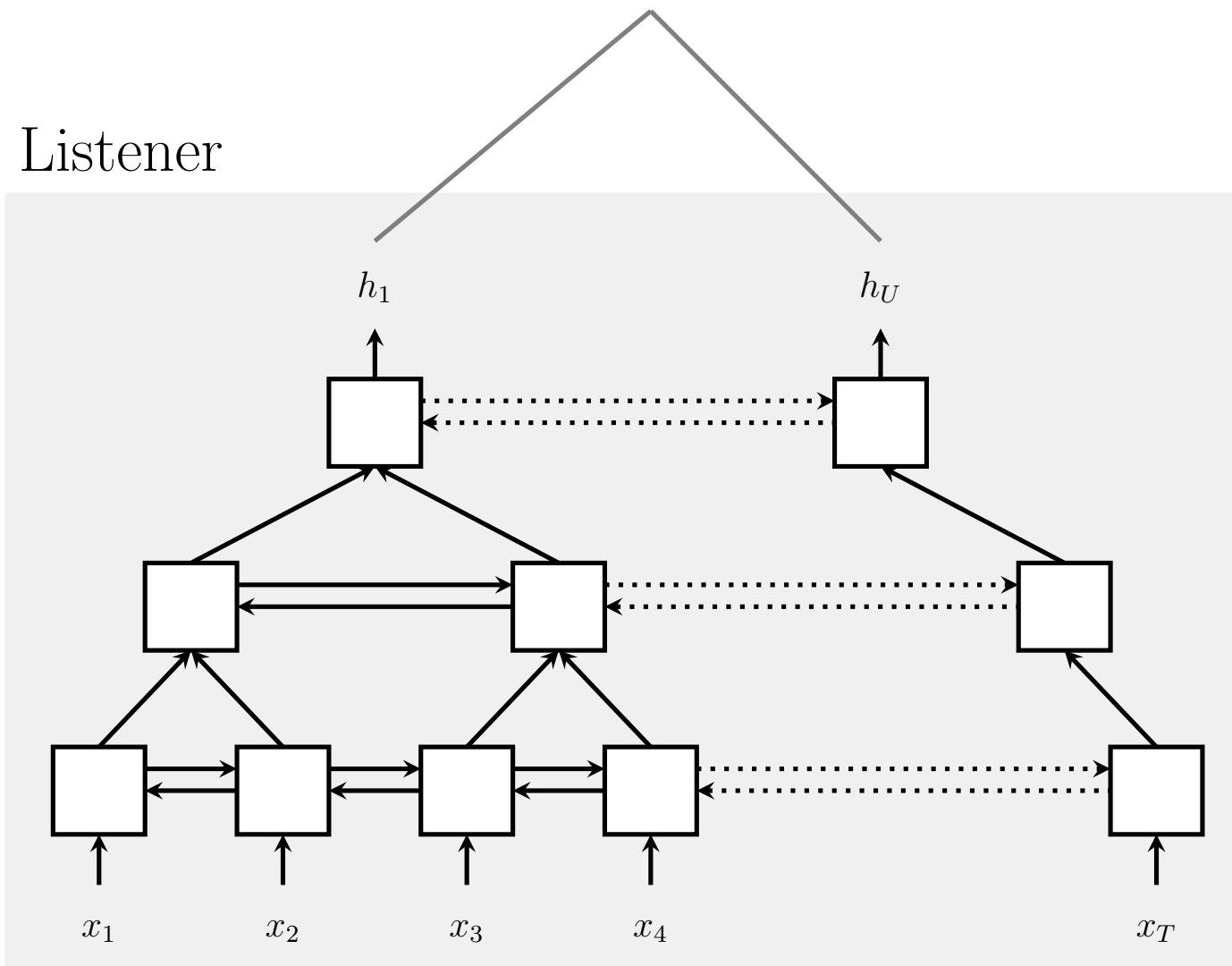
Encoder-decoder architecture with attention



Speller



Listener



The Model

- The Listen, Attend & Spell (LAS) architecture is a sequence-to-sequence model consisting of
 - a Listener (Listen): An acoustic model encoder. Deep BLSTMs with a pyramidal structure: reduces the time resolution by a factor of 2 in each layer.
 - a Speller (AttendAndSpell): An attention-based decoder. Consumes \mathbf{h} and produces a probability distribution over characters.

$$\mathbf{h} = \text{Listen}(\mathbf{x})$$

$$P(y_i | \mathbf{x}, y_{<i}) = \text{AttendAndSpell}(y_{<i}, \mathbf{h})$$

Attend and spell

- Produces a distribution over characters conditioned on all characters seen previously

$$c_i = \text{AttentionContext}(s_i, \mathbf{h})$$

$$s_i = \text{RNN}(s_{i-1}, y_{i-1}, c_{i-1})$$

$$P(y_i | \mathbf{x}, y_{<i}) = \text{CharacterDistribution}(s_i, c_i)$$

- At each decoder time-step i , `AttentionContext` computes a score for each encoder step u , which is then converted into softmax probabilities that are linearly combined to compute c_i

$$e_{i,u} = \langle \phi(s_i), \psi(h_u) \rangle$$

$$\alpha_{i,u} = \frac{\exp(e_{i,u})}{\sum_{u'} \exp(e_{i,u'})}$$

$$c_i = \sum_u \alpha_{i,u} h_u$$

Training and Decoding

- Training
 - Train the parameters of the model to maximize the log probability of the training instances

$$\tilde{\theta} = \max_{\theta} \sum_i \log P(y_i | \mathbf{x}, \tilde{y}_{<i}; \theta)$$

- Decoding
 - Simple left-to-right beam search
 - Beams can be rescored with a language model

Experiments

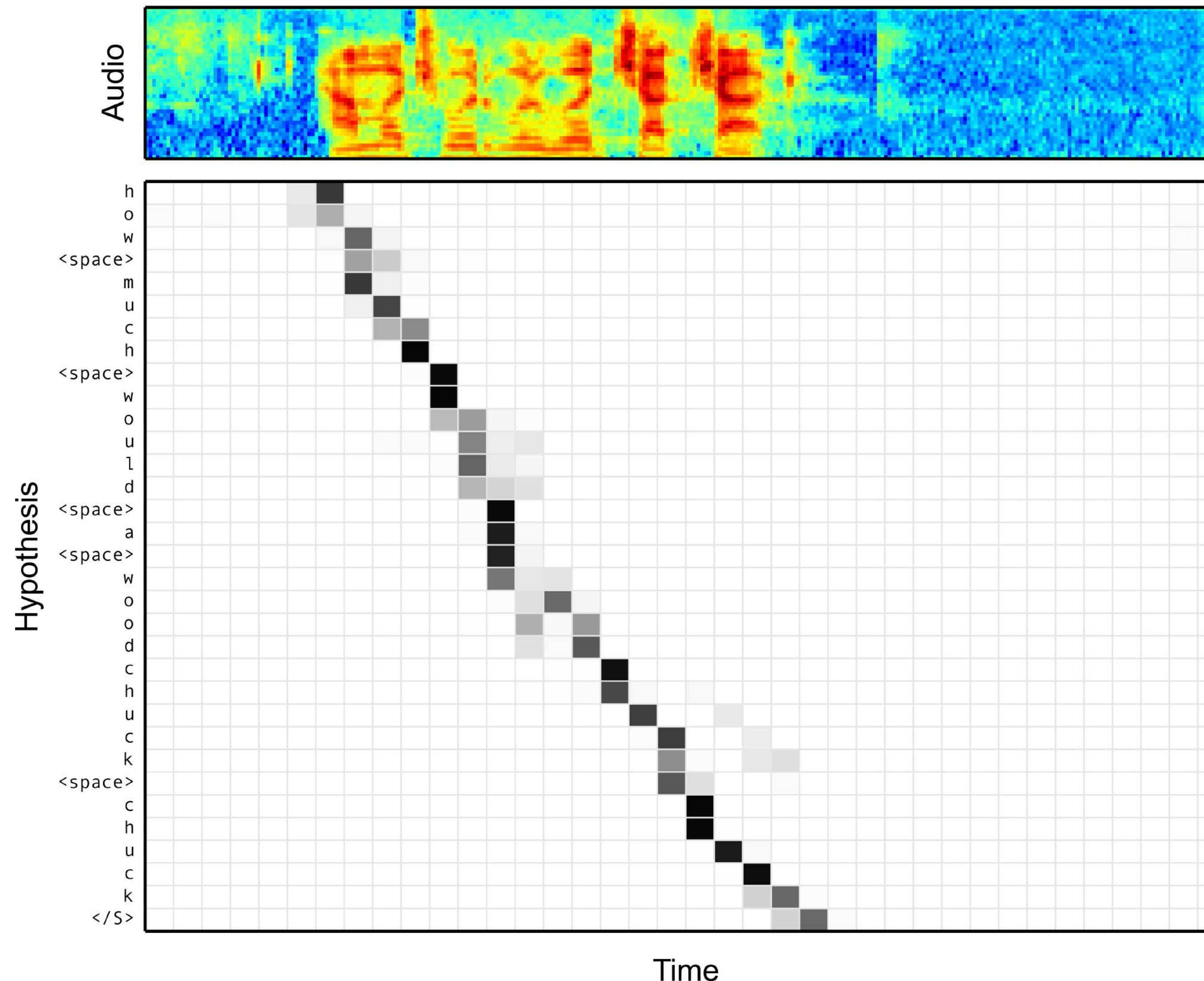
Table 1: WER comparison on the clean and noisy Google voice search task. The CLDNN-HMM system is the state-of-the-art, the Listen, Attend and Spell (LAS) models are decoded with a beam size of 32. Language Model (LM) rescoring can be beneficial.

Model	Clean WER	Noisy WER
CLDNN-HMM [22]	8.0	8.9
LAS	14.1	16.5
LAS + LM Rescoring	10.3	12.0

- Listen function used 3 layers of BLSTM (512 nodes); AttendAndSpell used a 2-layer LSTM (256 nodes)
- Constraining the beam search with a dictionary had no impact on WER

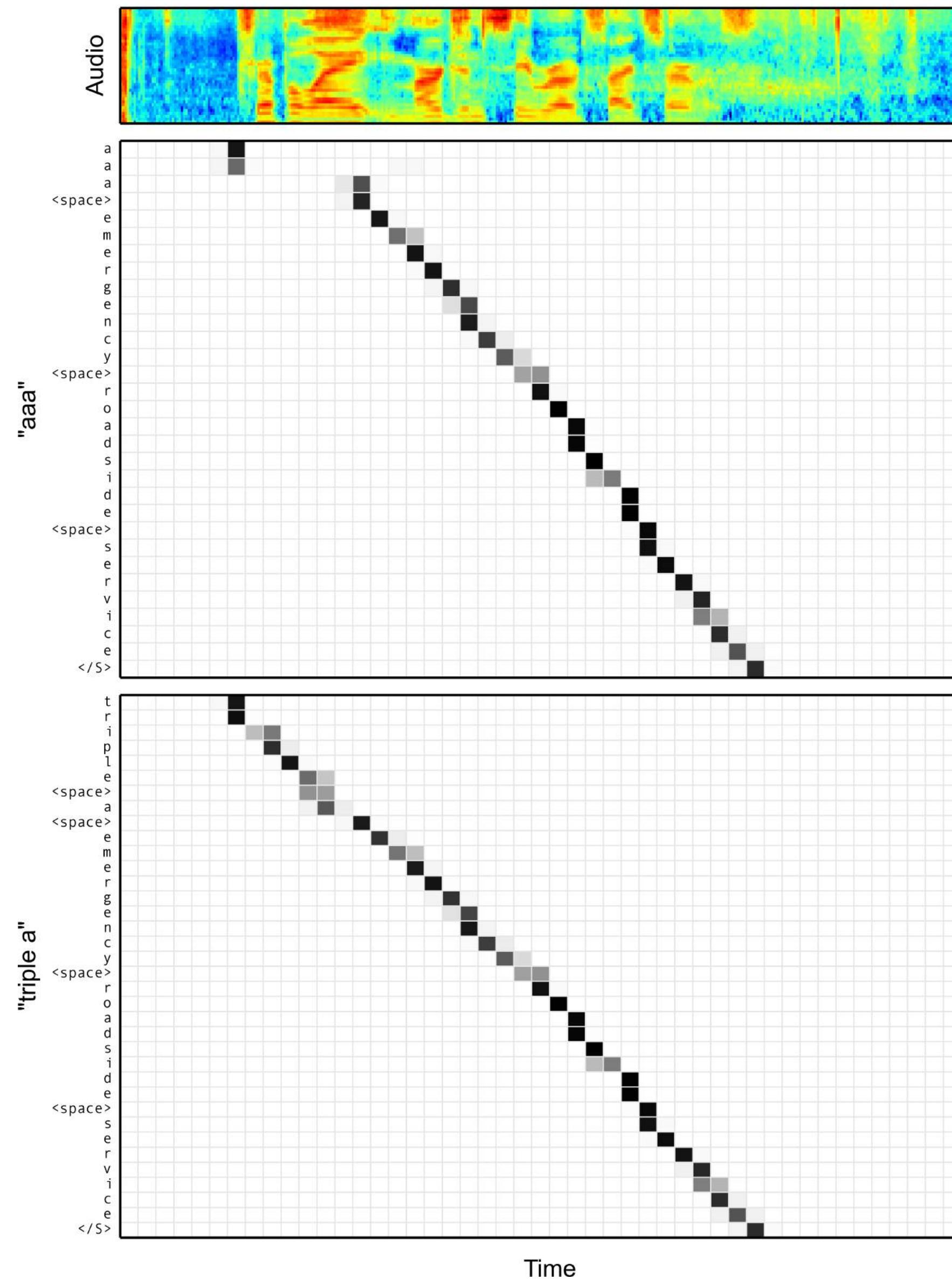
Analysis

Alignment between the Characters and Audio



Attention Distributions

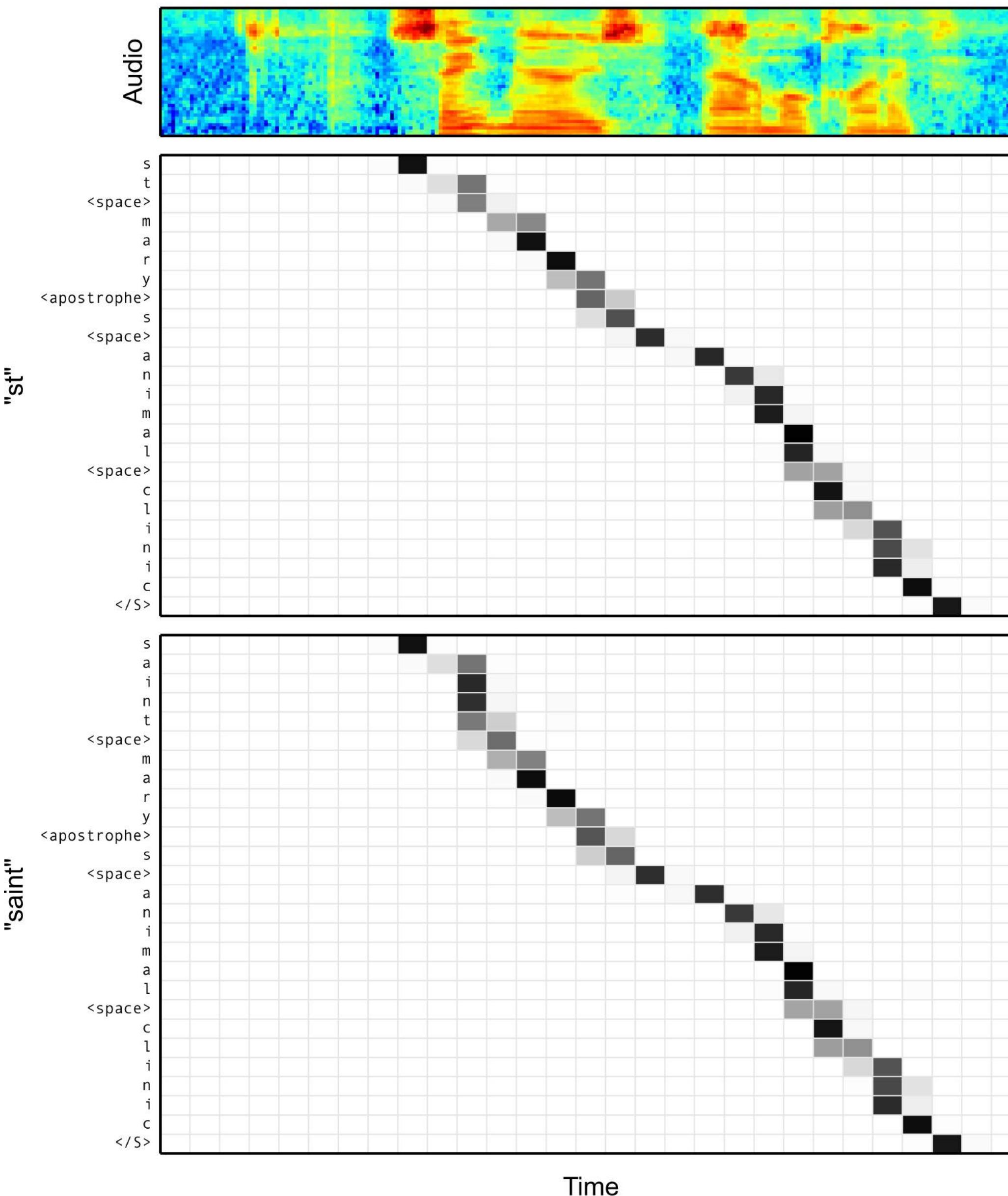
Spelling Variants of "aaa" vs. "triple a"



Beam	Text	$\log P$	WER
Truth	call aaa roadside assistance	-	-
1	call aaa roadside assistance	-0.57	0.00
2	call triple a roadside assistance	-1.54	50.00
3	call trip way roadside assistance	-3.50	50.00
4	call xxx roadside assistance	-4.44	25.00

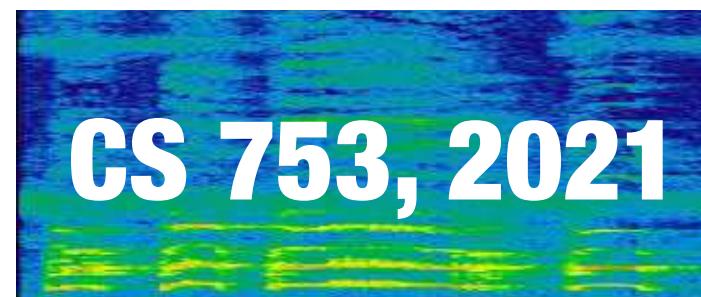
Attention Distributions

Spelling Variants of "st" vs. "saint"



End-to-End Neural ASR Systems (II)

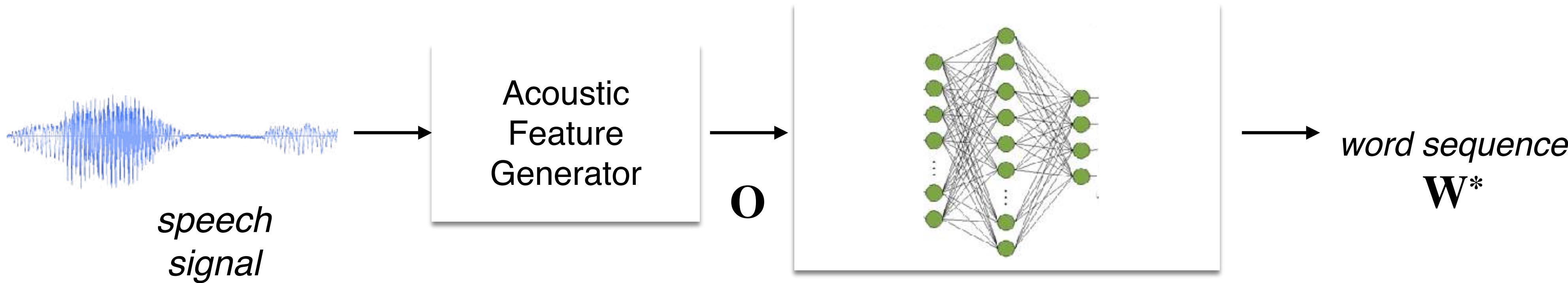
Lecture 8a



Instructor: Preethi Jyothi, IITB

Recap: Cascaded ASR \Rightarrow End-to-end ASR

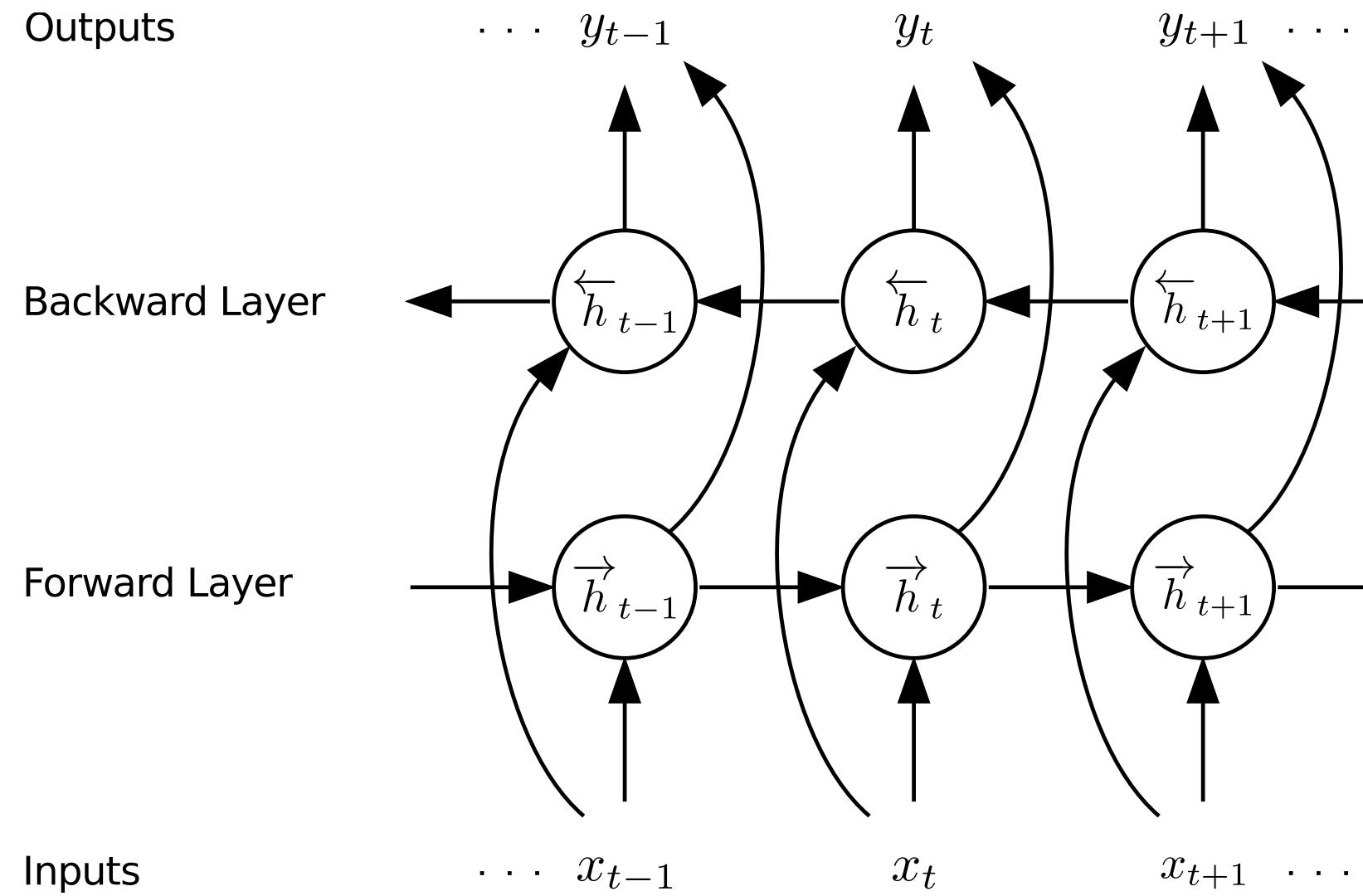
$$\mathbf{W}^* = \arg \max_W \Pr(\mathbf{W} | \mathbf{O})$$



Single end-to-end model that directly learns a mapping from speech to text

1. Encoder-decoder models with attention
2. **CTC-based models**

Network Architecture



$$\begin{aligned}\vec{h}_t &= \mathcal{H} \left(W_{xh}^{\rightarrow} x_t + W_{hh}^{\rightarrow} \vec{h}_{t-1} + b_h^{\rightarrow} \right) \\ \overleftarrow{h}_t &= \mathcal{H} \left(W_{xh}^{\leftarrow} x_t + W_{hh}^{\leftarrow} \overleftarrow{h}_{t+1} + b_h^{\leftarrow} \right) \\ y_t &= W_{hy}^{\rightarrow} \vec{h}_t + W_{hy}^{\leftarrow} \overleftarrow{h}_t + b_o\end{aligned}$$

- Input: Acoustic feature vectors. Output: Characters
- Long Short-Term Memory (LSTM) units (with in-built memory cells) are used to implement \mathcal{H} (in eqns above)
- Deep bidirectional LSTMs: Stack multiple bidirectional LSTM layers

Connectionist Temporal Classification (CTC)

- RNNs in ASR, if trained at the frame-level, will typically require alignments between the acoustics and the word sequence during training telling you which label (e.g. phone or character) should be output at each timestep.
- A new loss function, Connectionist Temporal Classification (CTC) tries to get around this.
- This is an objective function that allows RNN training without an explicit alignment step: CTC considers all possible alignments.

CTC Objective Function

- CTC objective function is the probability of an output label sequence y given an utterance x

$$\text{CTC}(x, y) = \Pr(y|x) = \sum_{a \in B^{-1}(y)} \Pr(a|x)$$

- Here, we sum over all possible alignments for y , enumerated by $B^{-1}(y)$

CTC: Mapping alignments to an output sequence

- Augment the output vocabulary with an additional “blank” (_) label
- For a given label sequence, there can be multiple alignments: (x, y, z) could correspond to (x, _, y, _, _, z) or (_, x, x, _, y, z)
- Define a 2-step operator B that reduces a label sequence by: first, removing repeating labels and second, removing blanks.
 $B("x, _, y, _, _, z") = B("_, x, x, _, y, z") = "x, y, z"$

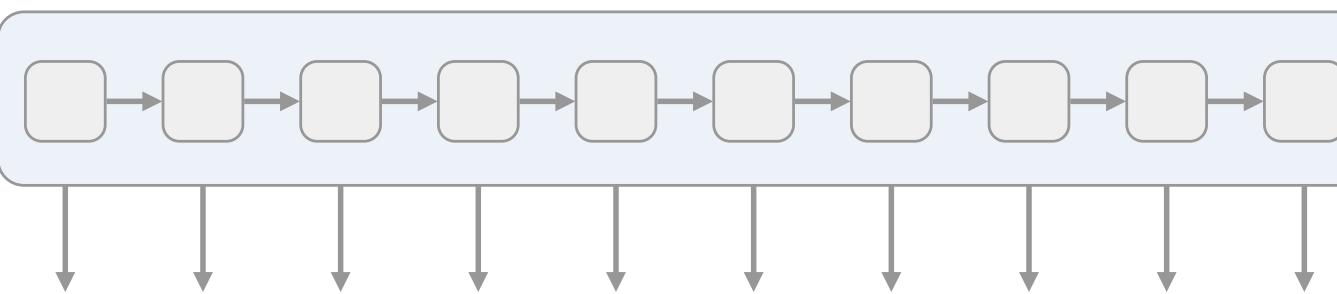
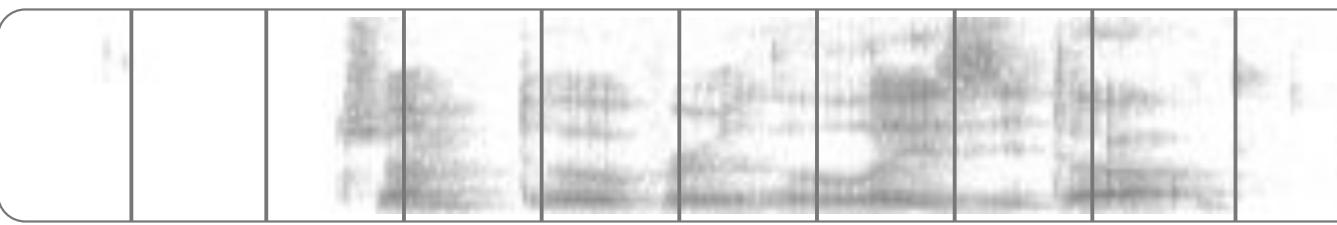
CTC Objective Function

- CTC objective function is the probability of an output label sequence y given an utterance x

$$\text{CTC}(x, y) = \Pr(y|x) = \sum_{a \in B^{-1}(y)} \Pr(a|x)$$

- Here, we sum over all possible alignments for y , enumerated by $B^{-1}(y)$
- CTC assumes that $\Pr(a|x)$ can be computed as $\prod_{t=1}^T \Pr(a_t|x)$
 - i.e. CTC assumes that outputs at each time-step are conditionally independent given the input
 - Efficient dynamic programming algorithm to compute this loss function and its gradients

Connectionist Temporal Classification: Overview



h	h	h	h	h	h	h	h	h	h	h
e	e	e	e	e	e	e	e	e	e	e
l	l	l	l	l	l	l	l	l	l	l
o	o	o	o	o	o	o	o	o	o	o
€	€	€	€	€	€	€	€	€	€	€

h	e	€			€			o	o
h	h	e			€	€		€	o
€	e	€			€	€		o	o

h	e			o
e			o	
h	e		o	

- CTC objective function is the probability of an output label sequence y given an utterance x (by summing over all possible alignments for y provided by $B^{-1}(y)$):

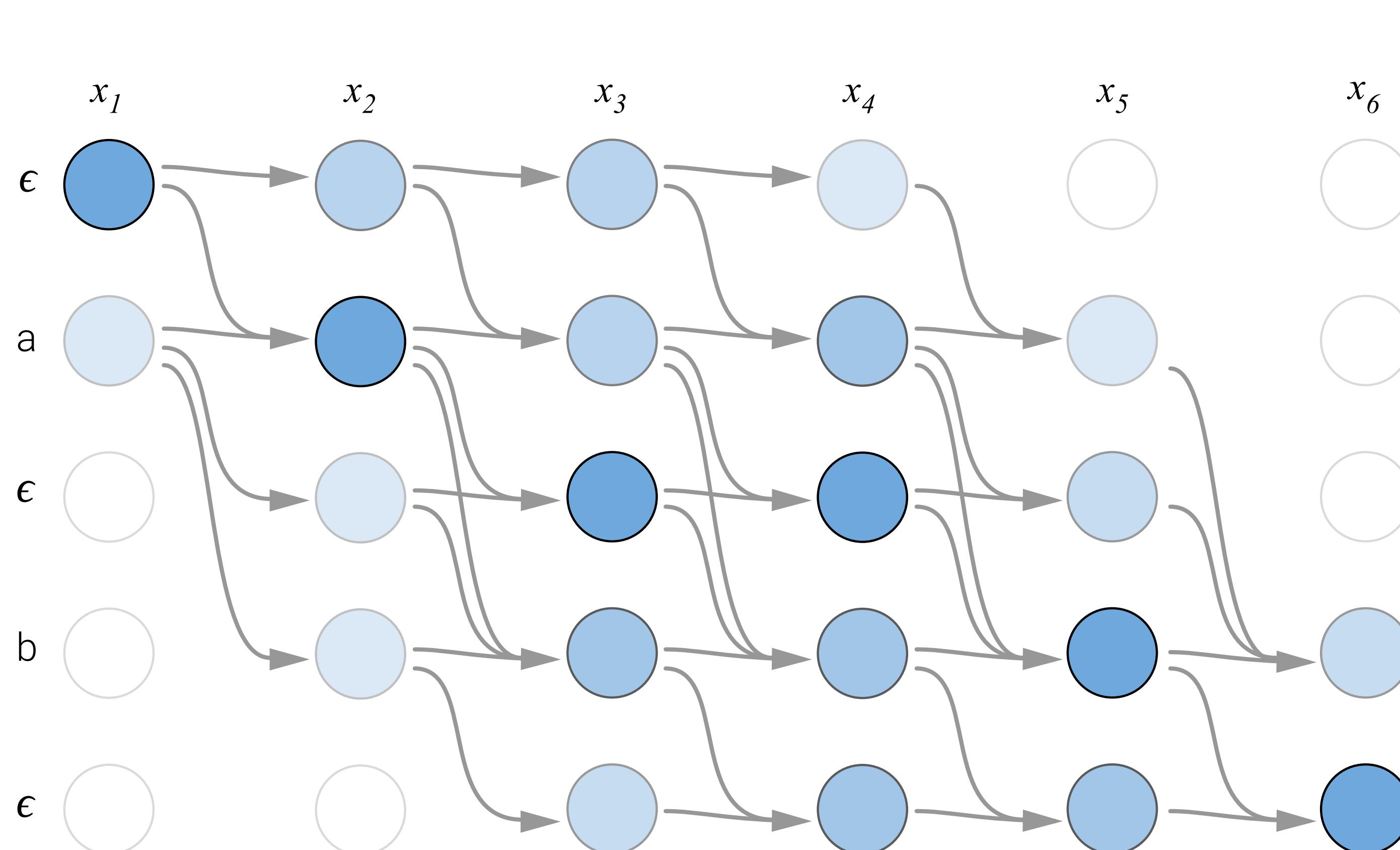
$$\text{CTC}(x, y) = \Pr(y | x) = \sum_{a \in B^{-1}(y)} \Pr(a | x)$$

$$= \sum_{a \in B^{-1}(y)} \prod_{t=1}^T \Pr(a_t | x)$$

- Efficient forward+backward algorithm to compute this loss function and its gradients [GJ14]

[GJ14] Towards End-to-End Speech Recognition with Recurrent Neural Networks, ICML 14

Illustration: Forward Algorithm to compute $\alpha_t(j)$



$$\alpha_t(j) = \sum_{i=j-2}^j \alpha_{t-1}(i)a_{ij}b_t(y'_j)$$

where

$b_t(y'_j)$ is the probability given by NN to the symbol y'_j for $t = 1 \dots T$, when $|x| = T$

$$y'_j = \begin{cases} y_{j/2} & \text{if } j \text{ is even} \\ \epsilon & \text{otherwise} \end{cases} \quad (j = 1 \dots 2l + 1 \text{ when } |y| = l)$$

$$a_{ij} = \begin{cases} 1 & \text{if } i = j \text{ or } i = j - 1 \\ 1 & \text{if } i = j - 2 \text{ and } y'_j \neq y'_{j-2} \\ 0 & \text{otherwise} \end{cases}$$

$$CTC(x, y) = \sum_{a \in B^{-1}(y)} \Pr(a \mid x) = \alpha_T(2l) + \alpha_T(2l + 1)$$

Decoding

- Pick the single most probable output at every time step

$$\arg \max_y \Pr(y|x) \approx B(\arg \max_a Pr(a|x))$$

- Use a beam search algorithm to integrate a dictionary and a language model

Experimental Results

Table 1. Wall Street Journal Results. All scores are word error rate/character error rate (where known) on the evaluation set. ‘LM’ is the Language model used for decoding. ‘14 Hr’ and ‘81 Hr’ refer to the amount of data used for training.

SYSTEM	LM	14 HR	81 HR
RNN-CTC	NONE	74.2/30.9	30.1/9.2
RNN-CTC	DICTIONARY	69.2/30.0	24.0/8.0
RNN-CTC	MONOGRAM	25.8	15.8
RNN-CTC	BIGRAM	15.5	10.4
RNN-CTC	TRIGRAM	13.5	8.7
BASELINE	NONE	—	—
BASELINE	DICTIONARY	56.1	51.1
BASELINE	MONOGRAM	23.4	19.9
BASELINE	BIGRAM	11.6	9.4
BASELINE	TRIGRAM	9.4	7.8
COMBINATION	TRIGRAM	—	6.7

Sample Character-level Transcripts

target: *TO ILLUSTRATE THE POINT A PROMINENT MIDDLE EAST ANALYST IN WASHINGTON RECOUNTS A CALL FROM ONE CAMPAIGN*

output: *TWO ALSTRAIT THE POINT A PROMINENT MIDILLE EAST ANALYST IM WASHINGTON RECOUNCACALL FROM ONE CAMPAIGN*

target: *T. W. A. ALSO PLANS TO HANG ITS BOUTIQUE SHINGLE IN AIRPORTS AT LAMBERT SAINT*

output: *T. W. A. ALSO PLANS TOHING ITS BOOTIK SINGLE IN AIRPORTS AT LAMBERT SAINT*

target: *ALL THE EQUITY RAISING IN MILAN GAVE THAT STOCK MARKET INDIGESTION LAST YEAR*

output: *ALL THE EQUITY RAISING IN MULONG GAVE THAT STACRK MARKET IN TO JUSTIAN LAST YEAR*

target: *THERE'S UNREST BUT WE'RE NOT GOING TO LOSE THEM TO DUKAKIS*

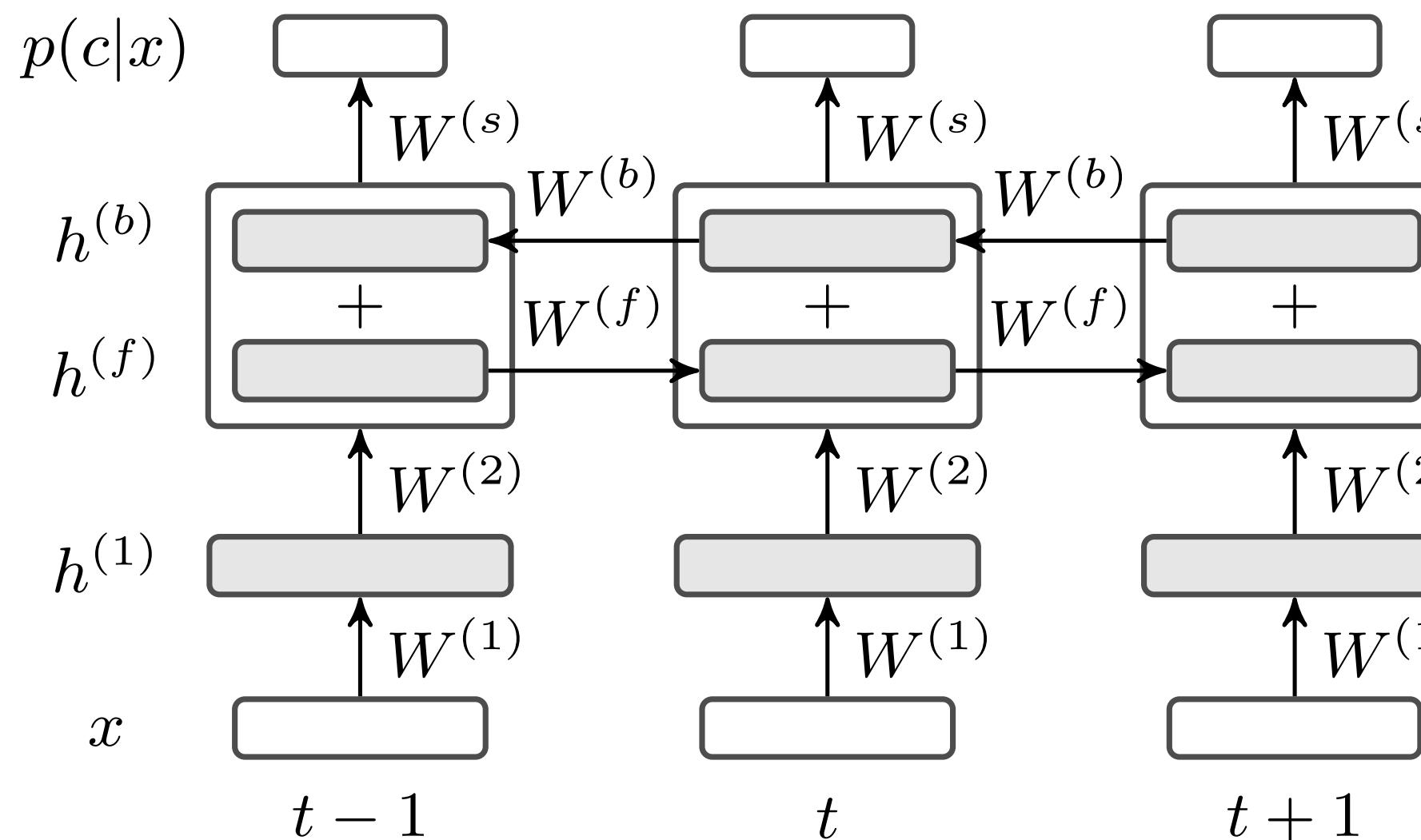
output: *THERE'S UNREST BUT WERE NOT GOING TO LOSE THEM TO DEKAKIS*

Another end-to-end system

- Decoding is at the word level if we use a dictionary+LM.
Out-of-vocabulary (OOV) words cannot be handled.
- Build a system that is trained and decoded entirely at the character-level [Maas et al.]
- This would enable the transcription of OOV words, disfluencies, etc.
- Shows results on the Switchboard task. Matches a GMM-HMM baseline system but underperforms compared to an HMM-DNN baseline.

Model Specifics

- Approach consists of two neural models:
 - A deep bidirectional RNN (DBRNN) mapping acoustic features to character sequences (Trained using CTC.)
 - A neural network character language model



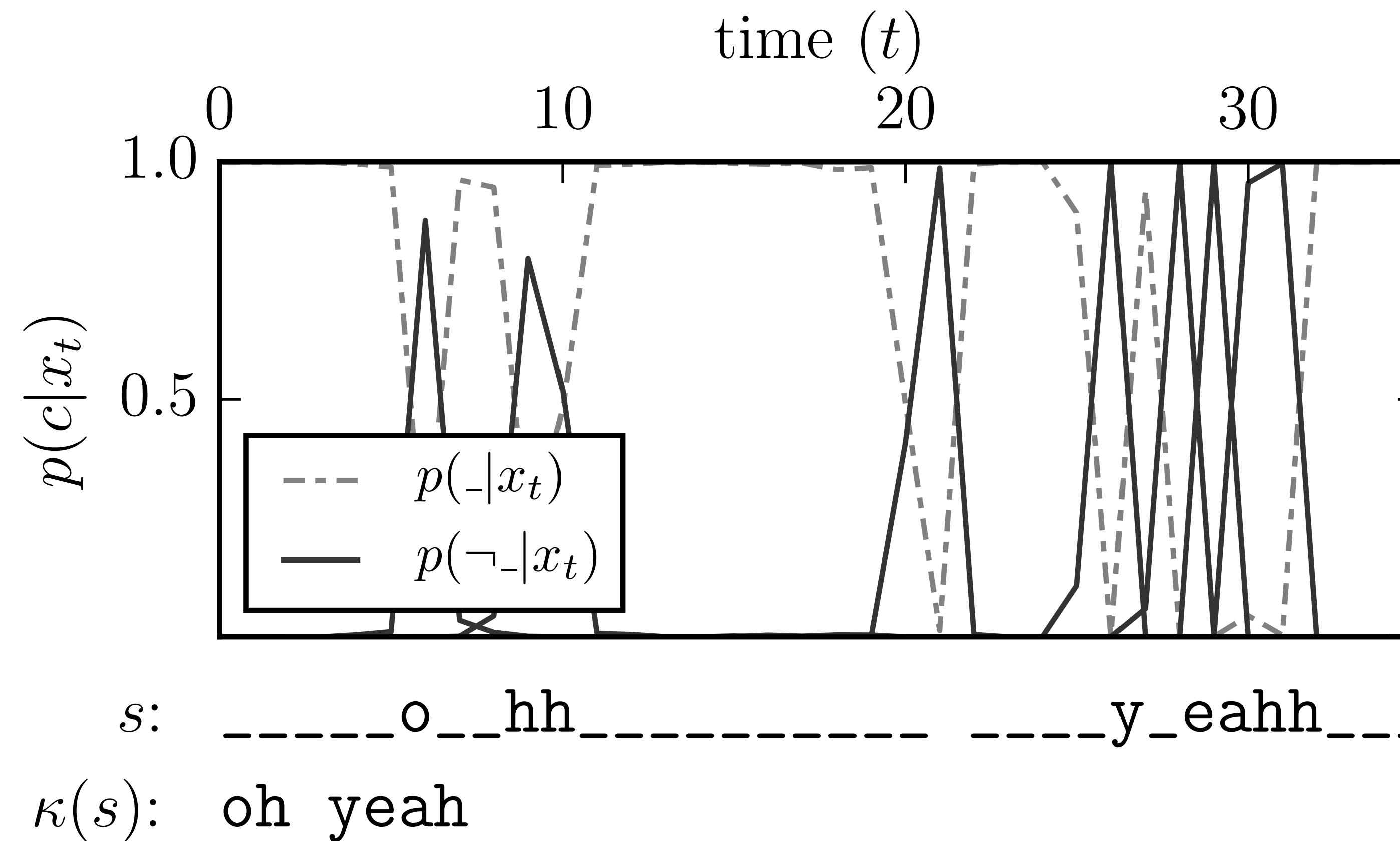
Decoding

- Simplest form: Decode without any language model
- Beam Search decoding:
 - Combine DBRNN outputs with a char-level language model
 - Char-level language model applied at every time step (unlike word models)
 - Circumvents the issue of handling OOV words during decoding

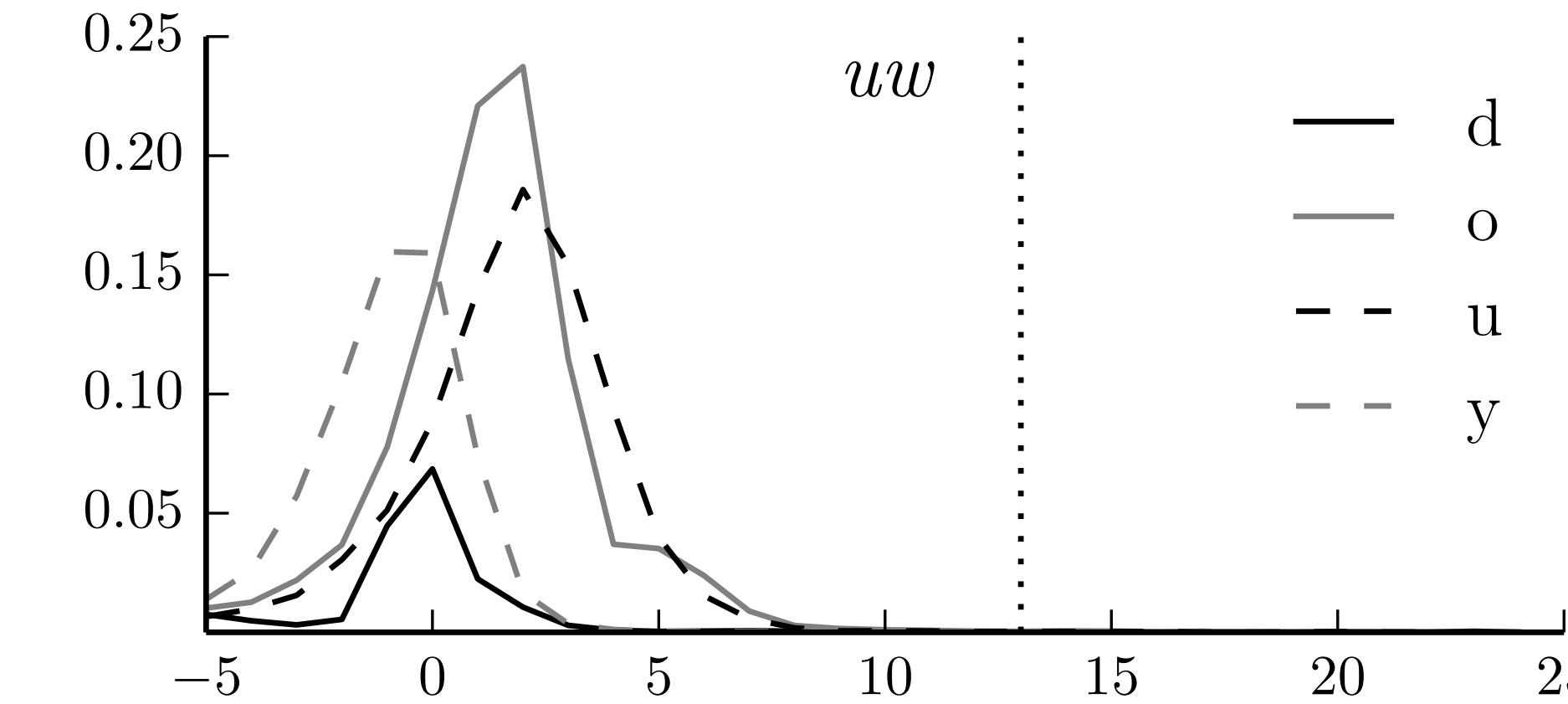
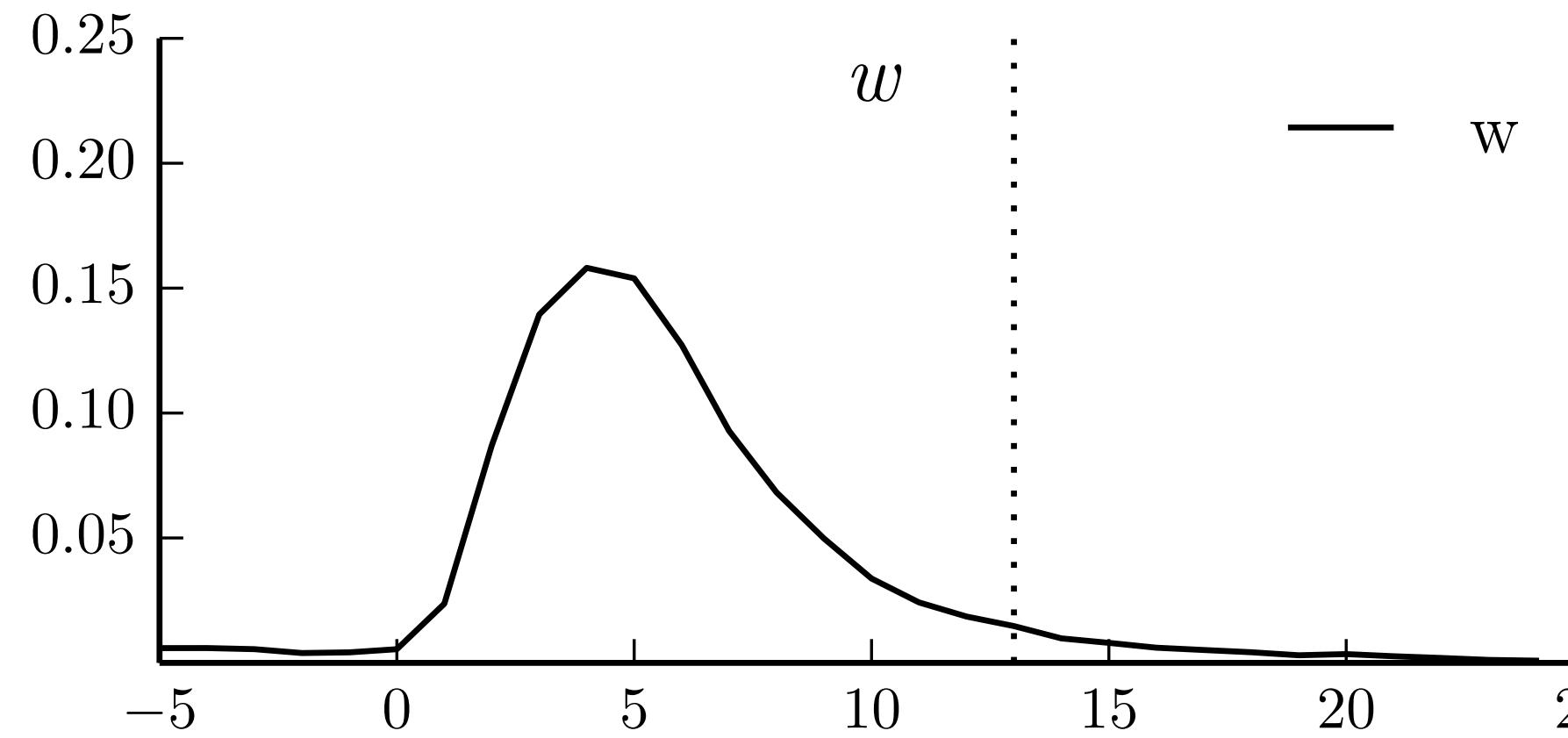
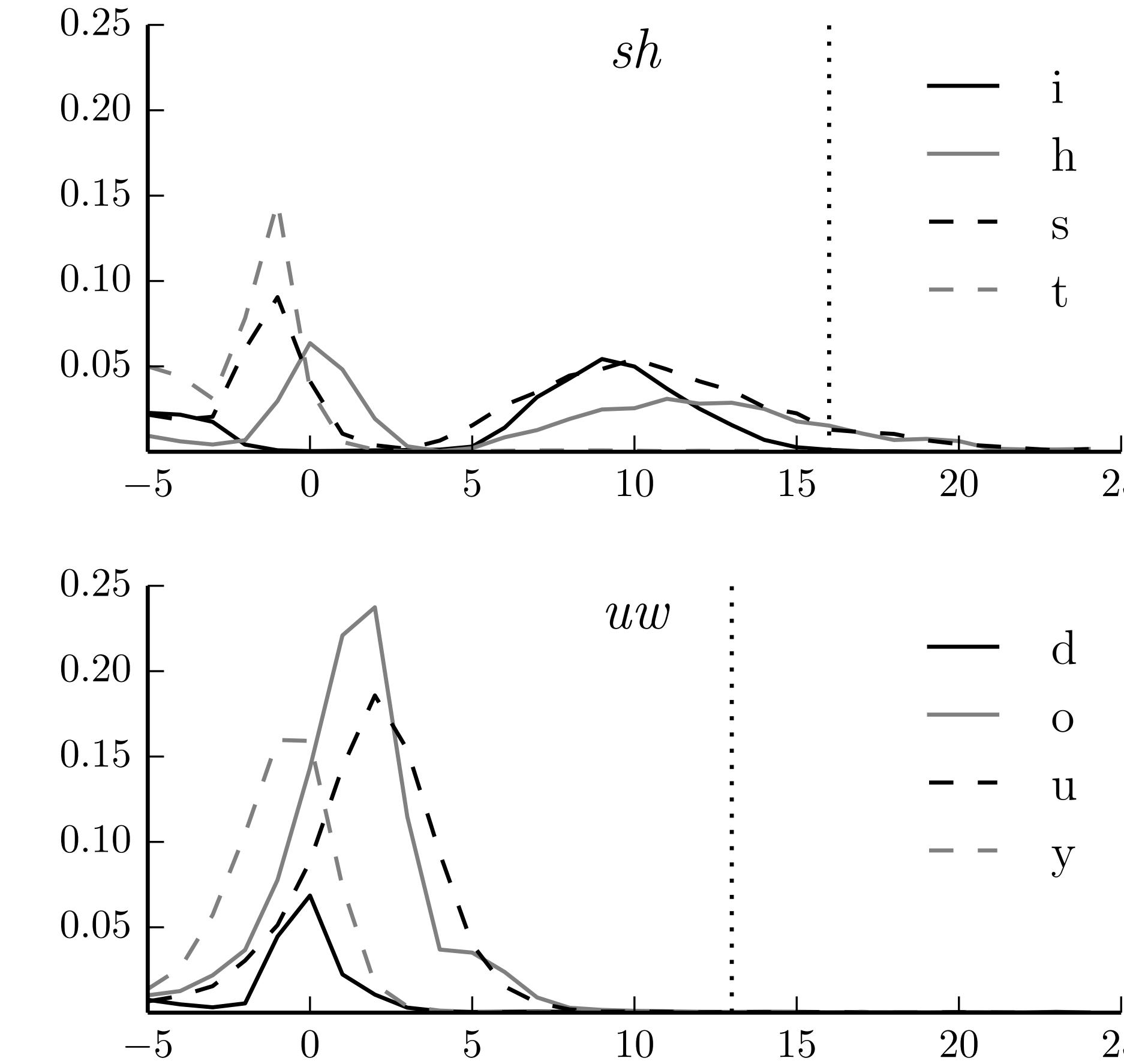
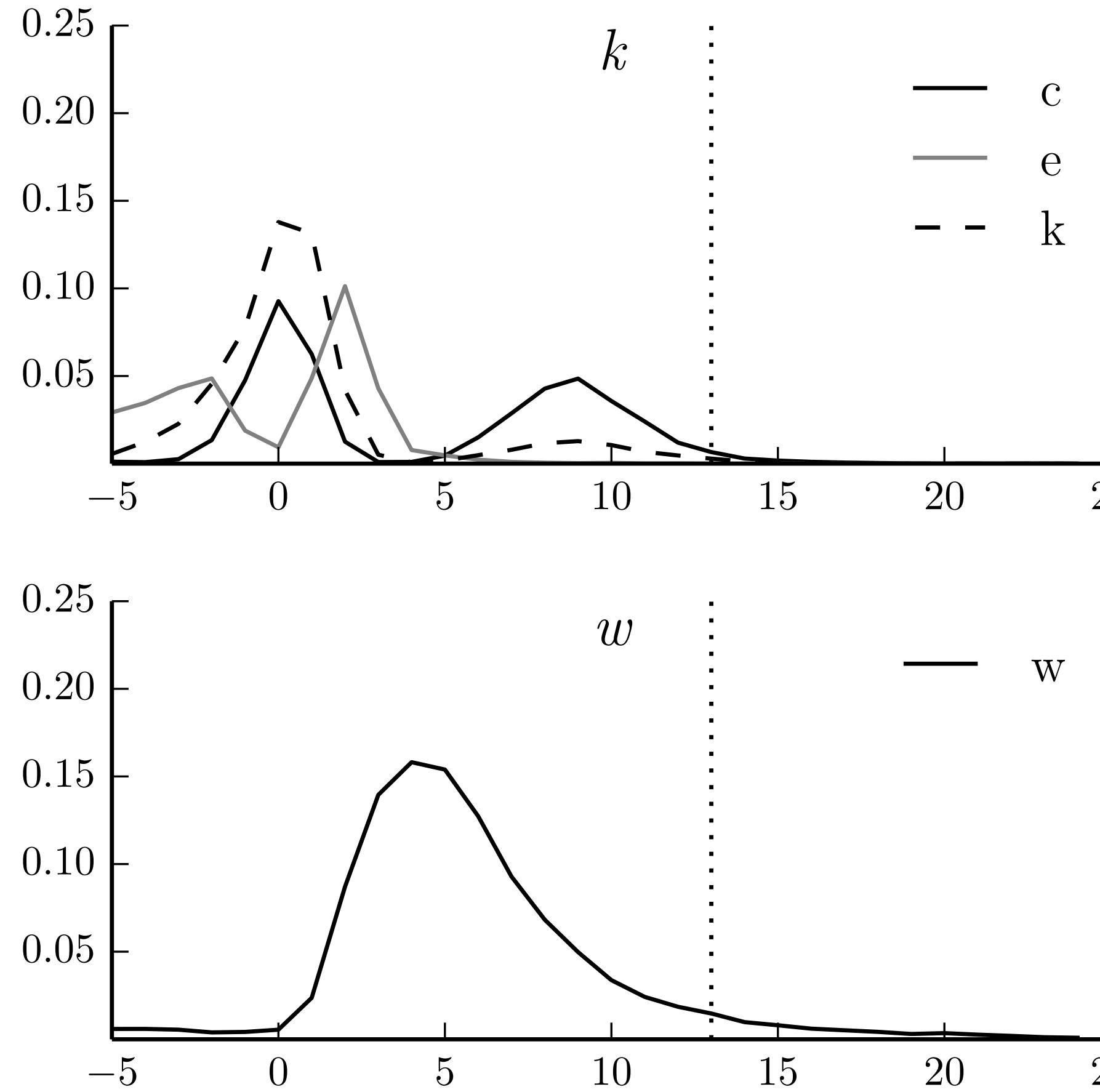
Sample Test Utterances

#	Method	Transcription
(1)	Truth	yeah i went into the i do not know what you think of <i>fidelity</i> but
	HMM-GMM	yeah when the i don't know what you think of fidel it even them
	CTC+CLM	yeah i went to i don't know what you think of fidelity but um
(2)	Truth	no no speaking of weather do you carry a altimeter slash <i>barometer</i>
	HMM-GMM	no i'm not all being the weather do you uh carry a uh helped emitters last brahms her
	CTC+CLM	no no beating of whether do you uh carry a uh a time or less barometer
(3)	Truth	i would ima- well yeah it is i know you are able to stay home with them
	HMM-GMM	i would amount well yeah it is i know um you're able to stay home with them
	CTC+CLM	i would ima- well yeah it is i know uh you're able to stay home with them

Analysing character probabilities



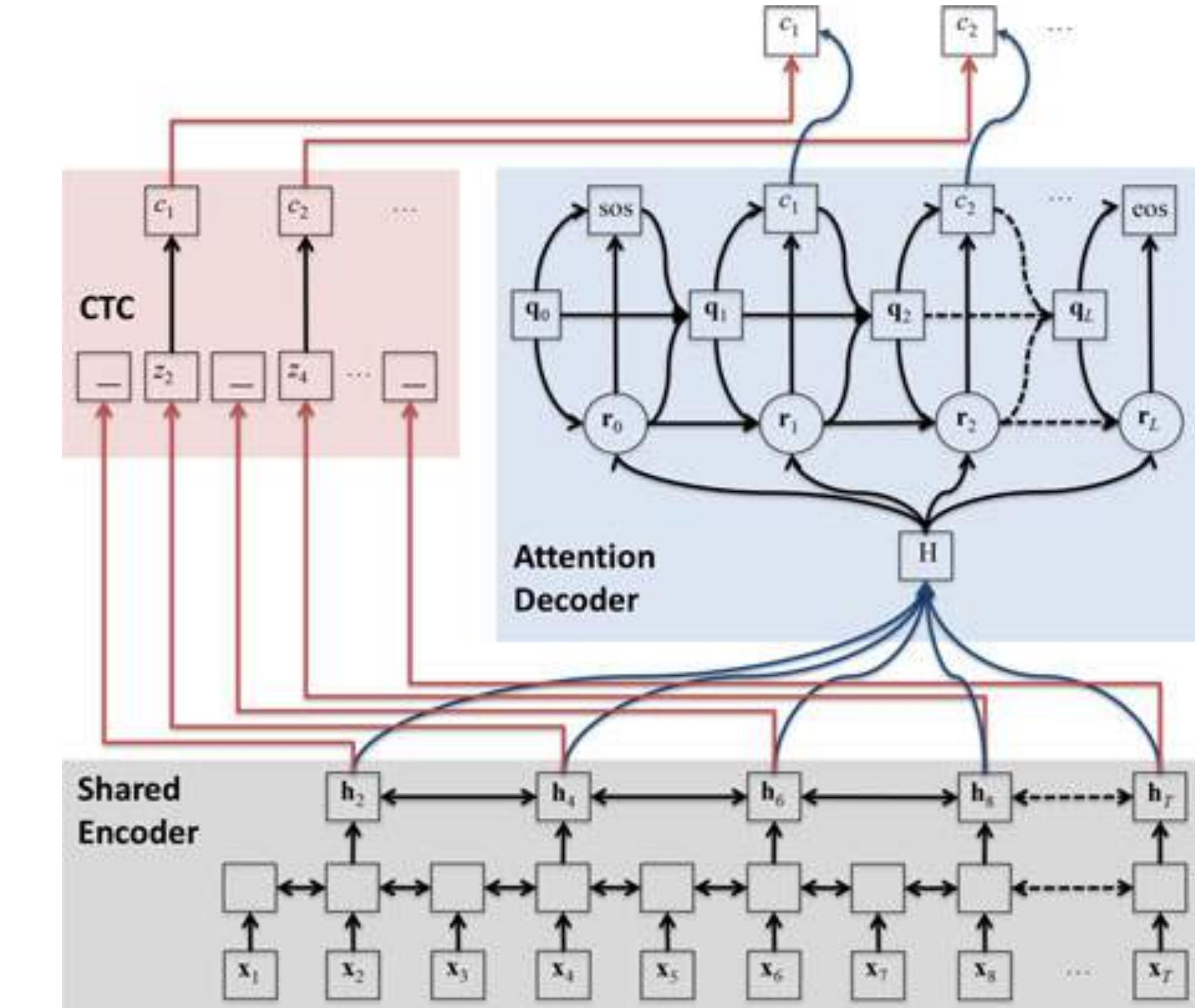
Character probabilities for various monophones



Hybrid CTC/Attention-based End-to-End ASR

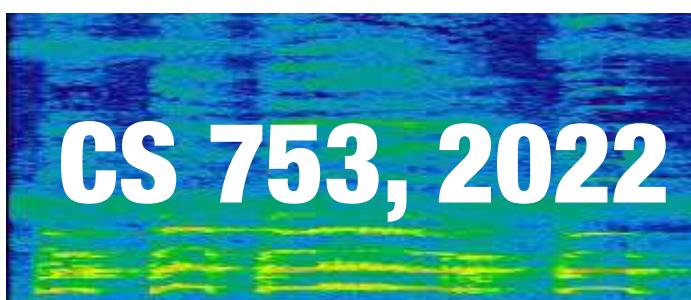
- The objective to be maximised is a linear combination of the CTC and attention objective functions.

	Task1	Task2	Task3
Cascaded GMM/ HMM Model	11.2	9.2	12.1
Cascaded HMM/ DNN Model	9.0	7.2	9.6
Hybrid CTC/ Attention	8.4	6.1	6.9



Live Session (Pre-midsem)

Lecture 7a



Instructor: Preethi Jyothi, IITB

Question 1: WFSTs for ASR

Recall the WFST-based framework for ASR that was described in class. Given a test utterance x , let D_x be a WFST over the tropical semiring (with weights specialized to the given utterance) such that decoding the utterance corresponds to finding the shortest path in D_x . Suppose we modify D_x by adding $\gamma (> 0)$ to each arc in D_x that emits a word. Let's call the resulting WFST D'_x .

A) Describe informally, what effect increasing γ would have on the word sequence obtained by decoding D'_x .

$\uparrow \text{ing } \gamma \rightarrow \text{result in decoded hyps}$
 $\text{being of smaller length}$

B) Recall that decoding D_x was used as an approximation for $\underset{W}{\text{\textbackslash argmax}} \Pr(x|W) \Pr(W)$. What would be the analogous expression for decoding from D'_x ?

$$-\log \Pr(x|w) - \log \Pr(w) + \boxed{\gamma|w|} \\ \Pr(x|w) \Pr(w) e^{-\gamma|w|}$$

Question 2: Morpheme-based ASR

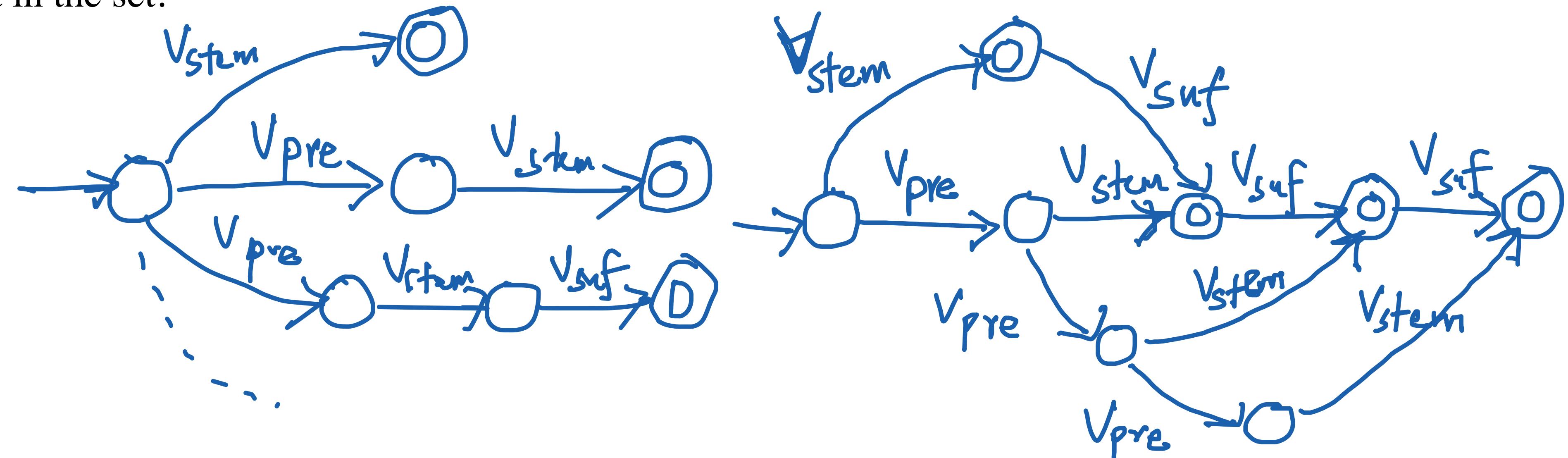
Words in a language can be composed of sub-word units called morphemes. For simplicity, in this problem, we consider there to be three sets of morphemes, V_{pre} , V_{stem} and V_{suf} – corresponding to prefixes, stems and suffixes. Further, we will assume that every word consists of a single stem, and zero or more prefixes and suffixes. That is, a word is of the form $w = p_1 \dots p_k \sigma s_1 \dots s_l$ where $k, l \geq 0$, and $p_i \in V_{\text{pre}}$, $s_i \in V_{\text{suf}}$ and $\sigma \in V_{\text{stem}}$. For example, a word like fair consists of a single morpheme (a stem), whereas the word unfairness is composed of three morphemes, un + fair + ness, which are a prefix, a stem and a suffix, respectively.

- A) Suppose we want to build an ASR system for a language using morphemes instead of words as the basic units of language. Which WFST(s) in the $H \circ C \circ L \circ G$ framework should be modified in order to utilize morphemes?

Question 2: Morpheme-based ASR

Words in a language can be composed of sub-word units called morphemes. For simplicity, in this problem, we consider there to be three sets of morphemes, V_{pre} , V_{stem} and V_{suf} – corresponding to prefixes, stems and suffixes. Further, we will assume that every word consists of a single stem, and zero or more prefixes and suffixes. That is, a word is of the form $w = p_1 \dots p_k \sigma s_1 \dots s_l$ where $k, l \geq 0$, and $p_i \in V_{\text{pre}}$, $s_i \in V_{\text{suf}}$ and $\sigma \in V_{\text{stem}}$. For example, a word like fair consists of a single morpheme (a stem), whereas the word unfairness is composed of three morphemes, un + fair + ness, which are a prefix, a stem and a suffix, respectively.

(B) Draw an FSA over morphemes ($V_{\text{pre}} \cup V_{\text{stem}} \cup V_{\text{suf}}$) that accepts only words with at most four morphemes. Your FSA should not have more than 10 states. You may draw a single arc labeled with a set to indicate a collection of arcs, each labeled with an element in the set.



Question 2: Morpheme-based ASR

Words in a language can be composed of sub-word units called morphemes. For simplicity, in this problem, we consider there to be three sets of morphemes, V_{pre} , V_{stem} and V_{suf} – corresponding to prefixes, stems and suffixes. Further, we will assume that every word consists of a single stem, and zero or more prefixes and suffixes. That is, a word is of the form $w = p_1 \dots p_k \sigma s_1 \dots s_l$ where $k, l \geq 0$, and $p_i \in V_{\text{pre}}$, $s_i \in V_{\text{suf}}$ and $\sigma \in V_{\text{stem}}$. For example, a word like fair consists of a single morpheme (a stem), whereas the word unfairness is composed of three morphemes, un + fair + ness, which are a prefix, a stem and a suffix, respectively.

(B) Draw an FSA over morphemes ($V_{\text{pre}} \cup V_{\text{stem}} \cup V_{\text{suf}}$) that accepts only words with at most four morphemes. Your FSA should not have more than 10 states. You may draw a single arc labeled with a set to indicate a collection of arcs, each labeled with an element in the set.

Question 3: Viterbi Variant

Modify the original Viterbi algorithm such that it returns the best state sequence among all sequences that never stay in a state for more than k consecutive transitions.

$V_t(j)$: Viterbi path probability

$$V_t(j) = \max_{d=1}^k V_{t,d}(j)$$

$$d > 1 : V_{t,d}(j) = V_{t-1,d-1}(j) a_{jj} b_j(o_t)$$

$$d = 1 : V_{t,1}(j) = \max_{i \neq j} \max_{d=1}^k V_{t-1,d}(i) a_{ij} b_j(o_t)$$

Question 4: CTC

The CTC training criterion is defined using a function $\underline{\mathcal{B}}$ that maps a per-frame output sequence $\mathbf{a} = (a_1, \dots, a_T)$ to a final output sequence $\mathbf{y} = (y_1, \dots, y_N)$, by first compressing each run of an identical character in \mathbf{a} to a run of length 1, and then removing all occurrences of the special blank symbol $\boxed{\epsilon}$. Here $y_j \in \underline{V}$ and $a_i \in \underline{V} \cup \{\boxed{\epsilon}\}$, where V is the output vocabulary.

Given an input sequence \mathbf{x} of length T and an output character sequence \mathbf{y} of length N , the CTC objective function is given by:

$$\Pr_{CTC}(\mathbf{y}|\mathbf{x}) = \sum_{\mathbf{a}: \underline{\mathcal{B}}(\mathbf{a})=\mathbf{y}} \frac{\Pr(\mathbf{a}|\mathbf{x})}{\uparrow}$$

Now suppose we would like to avoid the use of the blank symbol. Towards this we use a “contextualized” CTC. Here, each a_i is either a single symbol in \underline{V} or a pair of symbols in \underline{V} (a bigram). That is, the alphabet of a_i is now $W := V \cup V \times \underline{V}$. We define \mathcal{B} such that $\mathcal{B}(a_1, \dots, a_T) = (y_1, \dots, y_N)$ iff

$$\mathbf{a} = (\underbrace{y_1, \dots, y_1}_{u_1 \text{ times}}, \underbrace{(y_1, y_2), \dots, (y_1, y_2)}_{b_1 \text{ times}}, \underbrace{y_2, \dots, y_2}_{u_2 \text{ times}}, \underbrace{(y_2, y_3), \dots, (y_2, y_3)}_{b_2 \text{ times}}, \dots, \underbrace{(y_{N-1}, y_N), \dots, (y_{N-1}, y_N)}_{b_{N-1} \text{ times}}, \underbrace{y_N, \dots, y_N}_{u_N \text{ times}})$$

where the “unigram” counts $u_i > 0$ for all $i = 1, \dots, N$ and the “bigram” counts $b_i > 0$ for all $i = 1, \dots, N - 1$ (if $N = 1$, no bigrams can be present). If \mathbf{a} does not have this form, where two consecutive runs of unigrams are separated by a run of matching bigrams, then $\mathcal{B}(\mathbf{a}) = \perp$ (an error symbol). For example, $\mathcal{B}(a, a, (a, b), (a, b), b, (b, c), c, c) = (a, b, c)$. On the other hand, $\mathcal{B}(a, b, c) = \perp$, $\mathcal{B}(a, (a, b), b, (b, b), (b, c), c) = \perp$ and $\mathcal{B}(a, (a, b), (b, c), c) = \perp$.

Given a T frame input \mathbf{x} , suppose a neural network trained using CTC gives probabilities $\Pr(a|\mathbf{x}, t)$ for each $a \in W$ and $t = 1, \dots, T$. Describe a dynamic programming algorithm which, given these probabilities and $\mathbf{y} \in V^N$ finds $\sum_{\mathbf{a}: \mathcal{B}(\mathbf{a})=\mathbf{y}} \Pr(\mathbf{a}|\mathbf{x})$ under the independence assumption used in CTC. (Defining a new recurrence for the CTC forward algorithm, as we’ve seen in class, will suffice.)

Question 4: CTC

The CTC training criterion is defined using a function \mathcal{B} that maps a per-frame output sequence $\mathbf{a} = (a_1, \dots, a_T)$ to a final output sequence $\mathbf{y} = (y_1, \dots, y_N)$, by first compressing each run of an identical character in \mathbf{a} to a run of length 1, and then removing all occurrences of the special blank symbol ϵ . Here $y_j \in V$ and $a_i \in V \cup \{\epsilon\}$, where V is the output vocabulary.

Given an input sequence \mathbf{x} of length T and an output character sequence \mathbf{y} of length N , the CTC objective function is given by:

$$\Pr_{CTC}(\mathbf{y}|\mathbf{x}) = \sum_{\mathbf{a}: \mathcal{B}(\mathbf{a})=\mathbf{y}} \Pr(\mathbf{a}|\mathbf{x}).$$

Now suppose we would like to avoid the use of the blank symbol. Towards this we use a “contextualized” CTC. Here, each a_i is either a single symbol in V or a pair of symbols in V (a bigram). That is, the alphabet of a_i is now $W := V \cup V \times V$. We define \mathcal{B} such that $\mathcal{B}(a_1, \dots, a_T) = (y_1, \dots, y_N)$ iff

$$\mathbf{a} = (y_1, \dots, y_1, \underbrace{(y_1, y_2), \dots, (y_1, y_2)}_{u_1 \text{ times}}, \underbrace{y_2, \dots, y_2, \underbrace{(y_2, y_3), \dots, (y_2, y_3)}_{b_2 \text{ times}}, \dots, \underbrace{(y_{N-1}, y_N), \dots, (y_{N-1}, y_N)}_{b_{N-1} \text{ times}}, \underbrace{y_N, \dots, y_N}_{u_N \text{ times}})$$

where the “unigram” counts $u_i > 0$ for all $i = 1, \dots, N$ and the “bigram” counts $b_i > 0$ for all $i = 1, \dots, N-1$ (if $N = 1$, no bigrams can be present). If \mathbf{a} does not have this form, where two consecutive runs of unigrams are separated by a run of matching bigrams, then $\mathcal{B}(\mathbf{a}) = \perp$ (an error symbol). For example, $\mathcal{B}(a, a, (a, b), (a, b), b, (b, c), c, c) = (a, b, c)$. On the other hand, $\mathcal{B}(a, b, c) = \perp$, $\mathcal{B}(a, (a, b), b, (b, b), (b, c), c) = \perp$ and $\mathcal{B}(a, (a, b), (b, c), c) = \perp$.

Given a T frame input \mathbf{x} , suppose a neural network trained using CTC gives probabilities $\Pr(a|\mathbf{x}, t)$ for each $a \in W$ and $t = 1, \dots, T$. Describe a dynamic programming algorithm which, given these probabilities and $\mathbf{y} \in V^N$ finds $\sum_{\mathbf{a}: \mathcal{B}(\mathbf{a})=\mathbf{y}} \Pr(\mathbf{a}|\mathbf{x})$ under the independence assumption used in CTC. (Defining a new recurrence for the CTC forward algorithm, as we've seen in class, will suffice.)

$$\alpha_1(1) = b_1(y'_1)$$

$b_t(y'_j)$ is the probability given by the NN to the symbol y'_j when $|\mathbf{x}| = T$ for $t = 1, \dots, T$

$$\alpha_t(j) = b_t(y'_j) [\alpha_{t-1}(j-1) + \alpha_{t-1}(j)]$$

$$P_{CTC}(\mathbf{y}|\mathbf{x})$$

$$= \alpha_T(2N-1)$$

where

$$y'_j = \begin{cases} y_{\frac{j+1}{2}} & \text{if } j \text{ is odd } (j=1, \dots, 2N-1 \text{ when } |\mathbf{y}|=N) \\ (y_{\frac{j}{2}}, y_{\frac{j}{2}+1}) & \text{otherwise} \end{cases}$$

// unigram
when $|\mathbf{y}|=N$

// bigram

Question 5: Kneser-Ney Variant

Recall that a bigram language model with Kneser-Ney smoothing sets the probability of a word b to follow a word a as:

$$\Pr_{\text{KN}}[b | a] := \frac{\max\{\pi(a, b) - d, 0\}}{\pi(a)} + \frac{d \cdot |\Psi(a)| \cdot |\Phi(b)|}{\pi(a) \cdot |B|}.$$
Pr($\epsilon | a$) \cdot Pr($b | \epsilon$)
(1)

Here $\pi(a, b)$ denotes the count of the bigram ab in the data and $\pi(a) = \sum_y \pi(a, y)$; the sets involved are $\Psi(a) := \{y \mid \pi(a, y) > 0\}$, $\Phi(b) := \{x \mid \pi(x, b) > 0\}$, and $B := \{(x, y) \mid \pi(x, y) > 0\}$. d is an “absolute discount” parameter that was subtracted from the counts $\pi(a, b)$. That is, it was assumed that d occurrences of ab in fact occurred as $a\epsilon b$ (i.e., the context was “forgotten” after producing a , and b was produced with an empty context), and the remaining $\pi(a, b) - d$ occurrences corresponded to the bigram.

- (I) Instead of an absolute discounting d from the counts $\pi(a, b)$, one could use an adaptive discounting where each count $\pi(a, b)$ is discounted by $\mathcal{D}(\pi(a, b))$ where \mathcal{D} is a discounting function such that $0 \leq \mathcal{D}(n) \leq n$. Then the first term in the expression for $\Pr_{\text{KN}}[b | a]$ changes to

$$\frac{\pi(a, b) - \mathcal{D}(\pi(a, b))}{\pi(a)}.$$

What should the second term be in this case? Explain how you arrived at your answer.

$$\Pr(\epsilon | a) \Pr(b | \epsilon) = \frac{\sum_y \mathcal{D}(\pi(a, y))}{\pi(a)} \cdot \frac{\sum_x \mathcal{D}[\pi(x, b)]}{\sum_{x,y} \mathcal{D}[\pi(x, y)]}$$

Question 5: Kneser-Ney Variant

Recall that a bigram language model with Kneser-Ney smoothing sets the probability of a word b to follow a word a as:

$$\text{Pr}_{\text{KN}}[b \mid a] := \frac{\max\{\pi(a, b) - d, 0\}}{\pi(a)} + \frac{d \cdot |\Psi(a)| \cdot |\Phi(b)|}{\pi(a) \cdot |B|}. \quad (1)$$

Here $\pi(a, b)$ denotes the count of the bigram ab in the data and $\pi(a) = \sum_y \pi(a, y)$; the sets involved are $\Psi(a) := \{y \mid \pi(a, y) > 0\}$, $\Phi(b) := \{x \mid \pi(x, b) > 0\}$, and $B := \{(x, y) \mid \pi(x, y) > 0\}$. d is an “absolute discount” parameter that was subtracted from the counts $\pi(a, b)$. That is, it was assumed that d occurrences of ab in fact occurred as $a \in b$ (i.e., the context was “forgotten” after producing a , and b was produced with an empty context), and the remaining $\pi(a, b) - d$ occurrences corresponded to the bigram.

(II)

Question 5: Kneser-Ney Variant

Recall that a bigram language model with Kneser-Ney smoothing sets the probability of a word b to follow a word a as:

$$\Pr_{\text{KN}}[b \mid a] := \frac{\max\{\pi(a, b) - d, 0\}}{\pi(a)} + \frac{d \cdot |\Psi(a)| \cdot |\Phi(b)|}{\pi(a) \cdot |B|}. \quad (1)$$

Here $\pi(a, b)$ denotes the count of the bigram ab in the data and $\pi(a) = \sum_y \pi(a, y)$; the sets involved are $\Psi(a) := \{y \mid \pi(a, y) > 0\}$, $\Phi(b) := \{x \mid \pi(x, b) > 0\}$, and $B := \{(x, y) \mid \pi(x, y) > 0\}$. d is an “absolute discount” parameter that was subtracted from the counts $\pi(a, b)$. That is, it was assumed that d occurrences of ab in fact occurred as $a \epsilon b$ (i.e., the context was “forgotten” after producing a , and b was produced with an empty context), and the remaining $\pi(a, b) - d$ occurrences corresponded to the bigram.

- (II) In the case of absolute discount from above, the discounting function is given by $\underline{\mathcal{D}(n)} = \underline{\min\{d, n\}}$. Then, your general expression from the previous part would simplify to the expression in (1) under a certain assumption on the counts. What is this assumption?

All bigrams x, y that have a non-zero
count should appear at least d times

Question 5: Kneser-Ney Variant

Recall that a bigram language model with Kneser-Ney smoothing sets the probability of a word b to follow a word a as:

$$\Pr_{\text{KN}}[b \mid a] := \frac{\max\{\pi(a, b) - d, 0\}}{\pi(a)} + \frac{d \cdot |\Psi(a)| \cdot |\Phi(b)|}{\pi(a) \cdot |B|}. \quad (1)$$

Here $\pi(a, b)$ denotes the count of the bigram ab in the data and $\pi(a) = \sum_y \pi(a, y)$; the sets involved are $\Psi(a) := \{y \mid \pi(a, y) > 0\}$, $\Phi(b) := \{x \mid \pi(x, b) > 0\}$, and $B := \{(x, y) \mid \pi(x, y) > 0\}$. d is an “absolute discount” parameter that was subtracted from the counts $\pi(a, b)$. That is, it was assumed that d occurrences of ab in fact occurred as $a \epsilon b$ (i.e., the context was “forgotten” after producing a , and b was produced with an empty context), and the remaining $\pi(a, b) - d$ occurrences corresponded to the bigram.

- (III) Simplify your expression for $\Pr_{\text{KN}}[b \mid a]$ from part (i) above, when the discount function is defined as $D(n) = \delta n$, where $\delta \in [0, 1]$.

$$\Pr_{\text{KN}}[b \mid a] = \frac{\pi(a, b)(1 - \delta)}{\pi(a)} + \frac{\delta \sum_x \pi(x, b)}{\sum_x \pi(x)}$$

Question 6: LM Perplexity

Consider a simple language with vocabulary $\{A, B, C\}$ and bigram probabilities as follows:

	A	B	C	$\langle /s \rangle$
A	0.5	0.01	0.4	0.09
B	0.6	0.01	0.3	0.09
C	0.7	0.01	0.2	0.09
$\langle s \rangle$	0.33	0.33	0.33	0.01

$$P(c|A)$$

$$P(A|\langle s \rangle)$$

What is the perplexity for the sentence $\langle s \rangle CAB \langle /s \rangle$ in this model? While normalizing, you should omit $\langle s \rangle$ and $\langle /s \rangle$ in counting the total number of words.

You need not evaluate the expression numerically.

$$\Pr[\langle s \rangle CAB \langle /s \rangle] = \frac{1}{3} (P(A|\langle s \rangle) P(C|A) P(B|C)) = \frac{1}{3} (0.33 \cdot 0.7 \cdot 0.33) = \frac{1}{3} (0.33)^2 \cdot 0.7 = \frac{1}{3} (0.11)^2 \cdot 0.7 = \frac{1}{3} (0.0121) \cdot 0.7 = \frac{1}{3} (0.00847) = 0.002823333$$

Question 6: LM Perplexity

Consider a simple language with vocabulary $\{A, B, C\}$ and bigram probabilities as follows:

	A	B	C	$\langle /s \rangle$
A	0.5	0.01	0.4	0.09
B	0.6	0.01	0.3	0.09
C	0.7	0.01	0.2	0.09
$\langle s \rangle$	0.33	0.33	0.33	0.01

Find a sentence obtained by applying a single substitution to the above sentence $\langle s \rangle CAB \langle /s \rangle$ (and not changing $\langle s \rangle$ or $\langle /s \rangle$) such that the resulting sentence has the minimum possible perplexity. Show your work.

$\langle s \rangle, x$ where $x \in \{A, B, C\}$

Keep C intact : bigram prob of C,A is highest

Bigram prob of A,x is the highest for $x = A$

$\langle s \rangle CAA \langle /s \rangle$