ASR

第二课　声学模型
GMM/DNN-HMM

第二课 声学模型 GMM/DNN-HMM

- 知识点1：单音素声学模型：MONOPHONE GMM-HMM语音模型
- 知识点2：基于EM的模型训练
- 知识点3：考虑上下文的多音素声学模型: CONTEXT-DEPENDENT TRIPHONE GMM-HMM
- 知识点4：神经网络DNN原理
- 知识点5：从传统GMM-HMM到经典CD-DNN-HMM语音模型

REFERENCES

- HTTP://59.80.44.98/WWW.INF.ED.AC.UK/TEACHING/COURSES/ASR/2018-19/ASR03-HMMGMM-HANDOUT.PDF
- HTTP://WWW.INF.ED.AC.UK/TEACHING/COURSES/ASR/2018-19/ASR04-CDHMM-HANDOUT.PDF
- HTTP://WWW.INF.ED.AC.UK/TEACHING/COURSES/ASR/2018-19/ASR07-NNINTRO.PDF
- HTTP://WWW.INF.ED.AC.UK/TEACHING/COURSES/ASR/2018-19/ASR08-HYBRID_HMM_NN.PDF
- HTTP://WWW.INF.ED.AC.UK/TEACHING/COURSES/ASR/2018-19/ASR10-SEQ.PDF

第二课 声学模型 GMM/DNN-HMM

- 知识点1：基于EM的模型训练
- 知识点2：单音素声学模型：MONOPHONE GMM-HMM语音模型
- 知识点3：考虑上下文的多音素声学模型: CONTEXT-DEPENDENT TRIPHONE GMM-HMM
- 知识点4：神经网络DNN原理
- 知识点5：从传统GMM-HMM到经典CD-DNN-HMM语音模型

基础模型框架
HIERARCHICAL MODEL FOR SPEECH RECOGNITION

Fundamental Equation of Statistical Speech Recognition

If $\mathbf{X}$ is the sequence of acoustic feature vectors (observations) and $\mathbf{W}$ denotes a word sequence, the most likely word sequence $\mathbf{W}^*$ is given by

$$\mathbf{W}^* = \arg\max_{\mathbf{W}} P(\mathbf{W}|\mathbf{X})$$

Applying Bayes' Theorem:

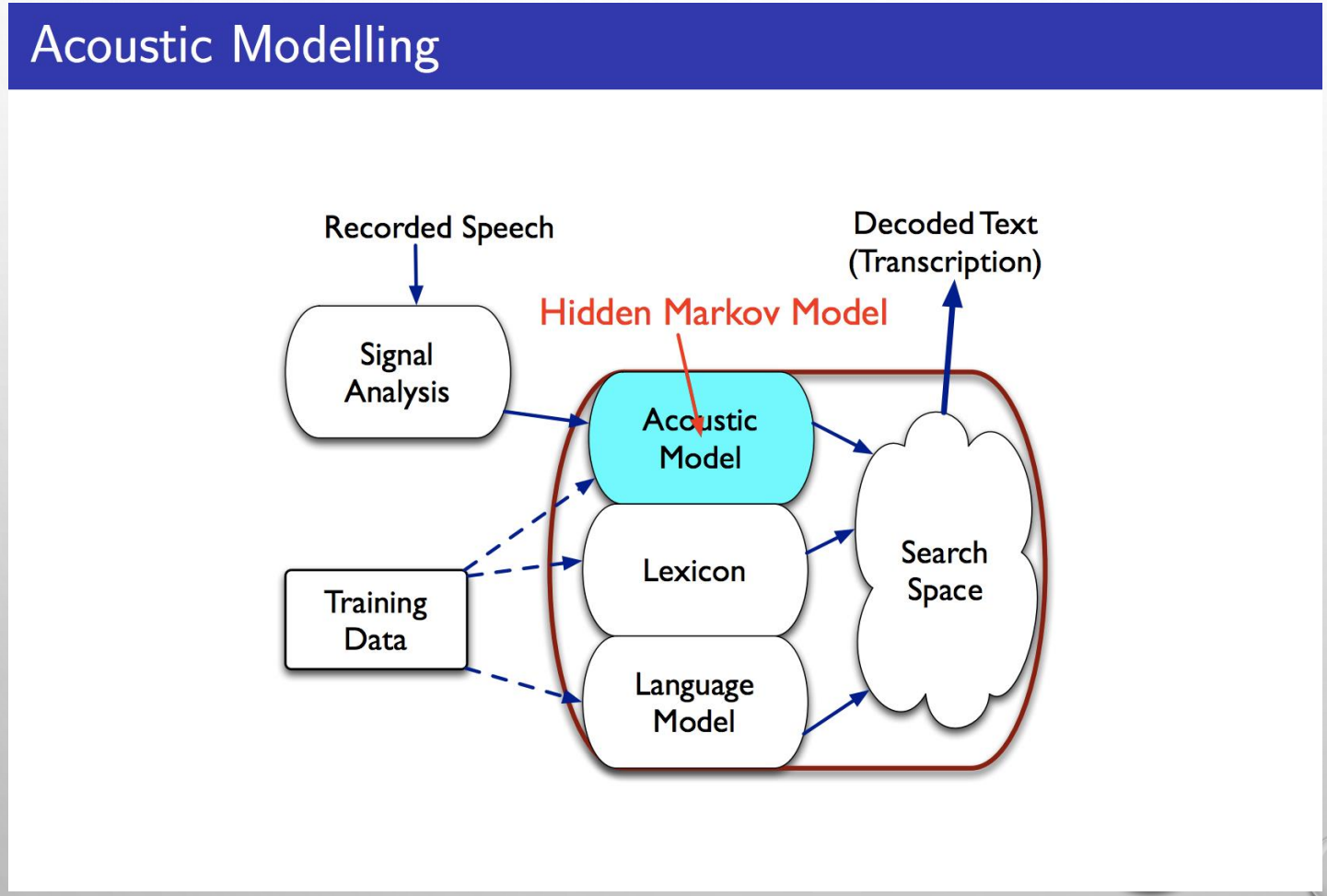
$$P(\mathbf{W}|\mathbf{X}) = \frac{p(\mathbf{X}|\mathbf{W})\,P(\mathbf{W})}{p(\mathbf{X})}$$

$$\propto p(\mathbf{X}|\mathbf{W})\,P(\mathbf{W})$$

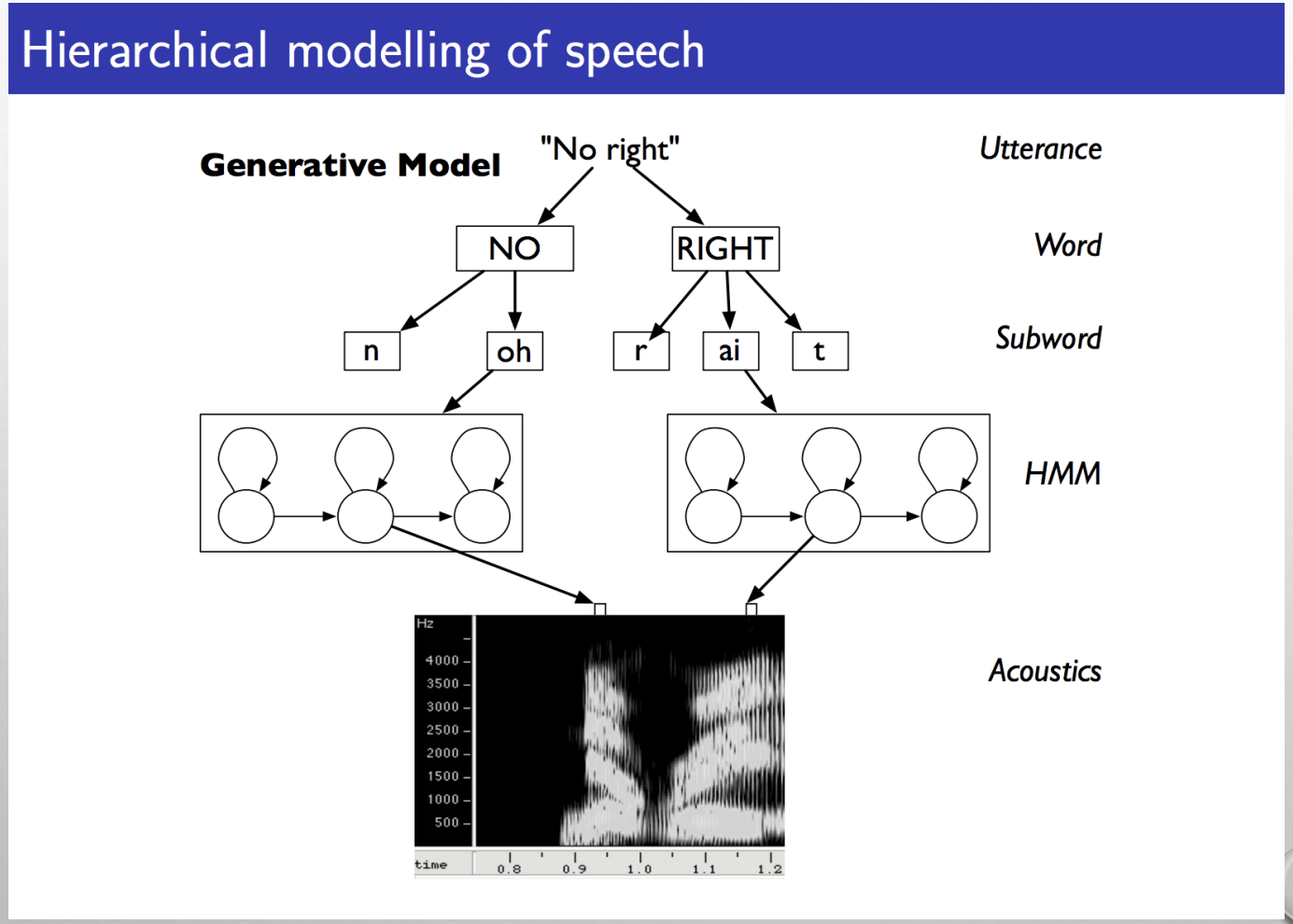
$$\mathbf{W}^* = \arg\max_{\mathbf{W}} \underbrace{p(\mathbf{X}|\mathbf{W})}_{\text{Acoustic model}} \underbrace{P(\mathbf{W})}_{\text{Language model}}$$

NB: $\mathbf{X}$ is used hereafter to denote the output feature vectors from the signal analysis module rather than DFT spectrum.

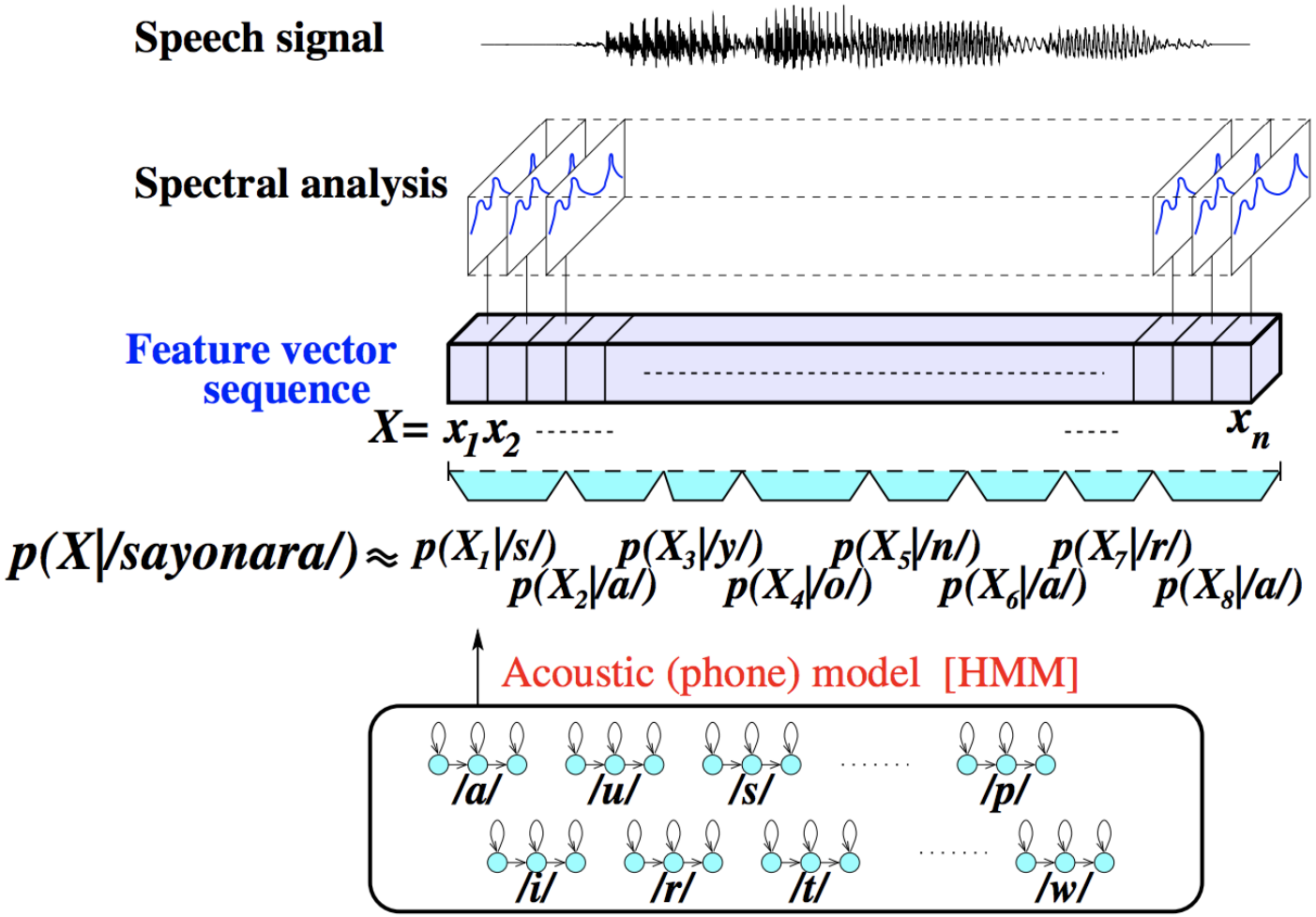
基础模型框架
HIERARCHICAL MODEL FOR SPEECH RECOGNITION



基础模型框架
HIERARCHICAL MODEL FOR SPEECH RECOGNITION

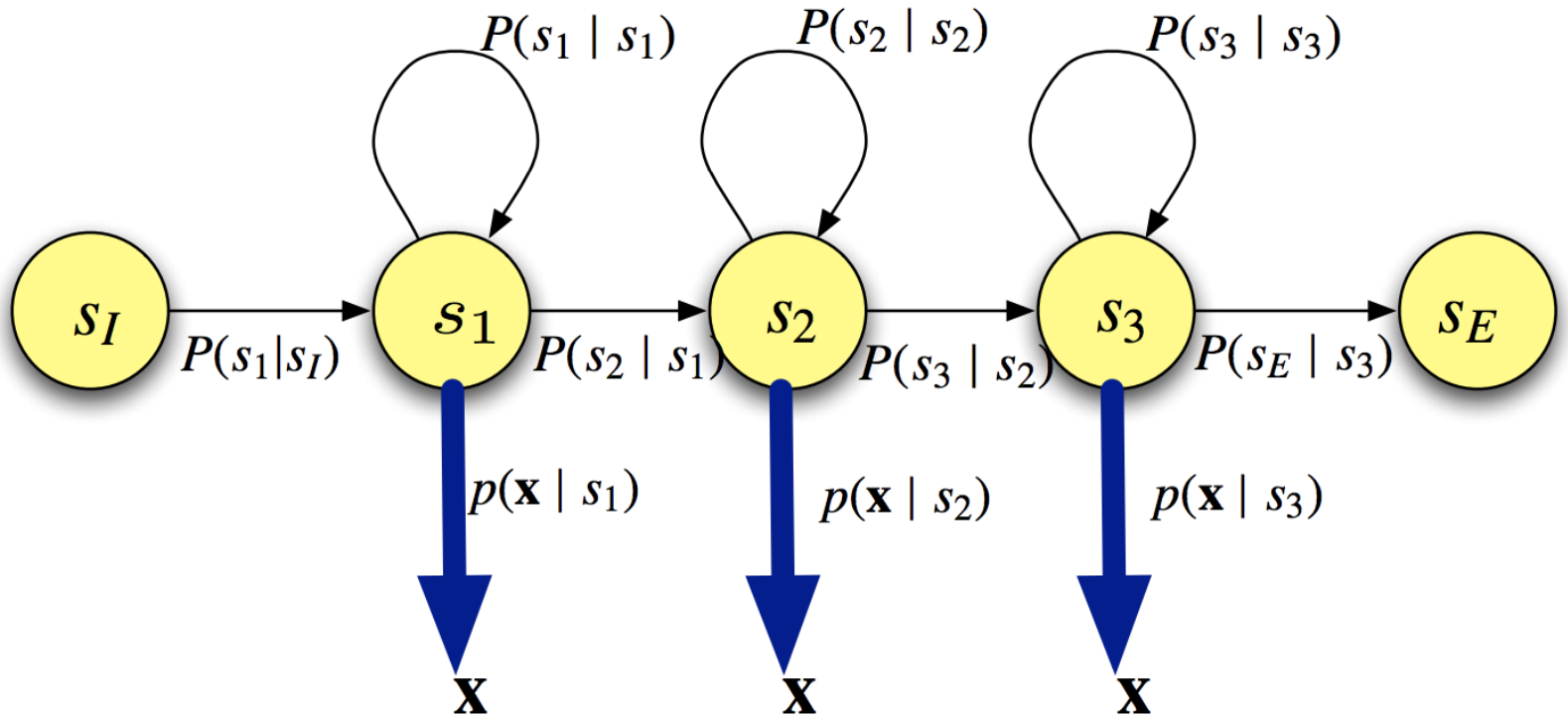


Calculation of $p(X|W)$



NB: some conditional independency is assumed here.

Acoustic Model: Continuous Density HMM



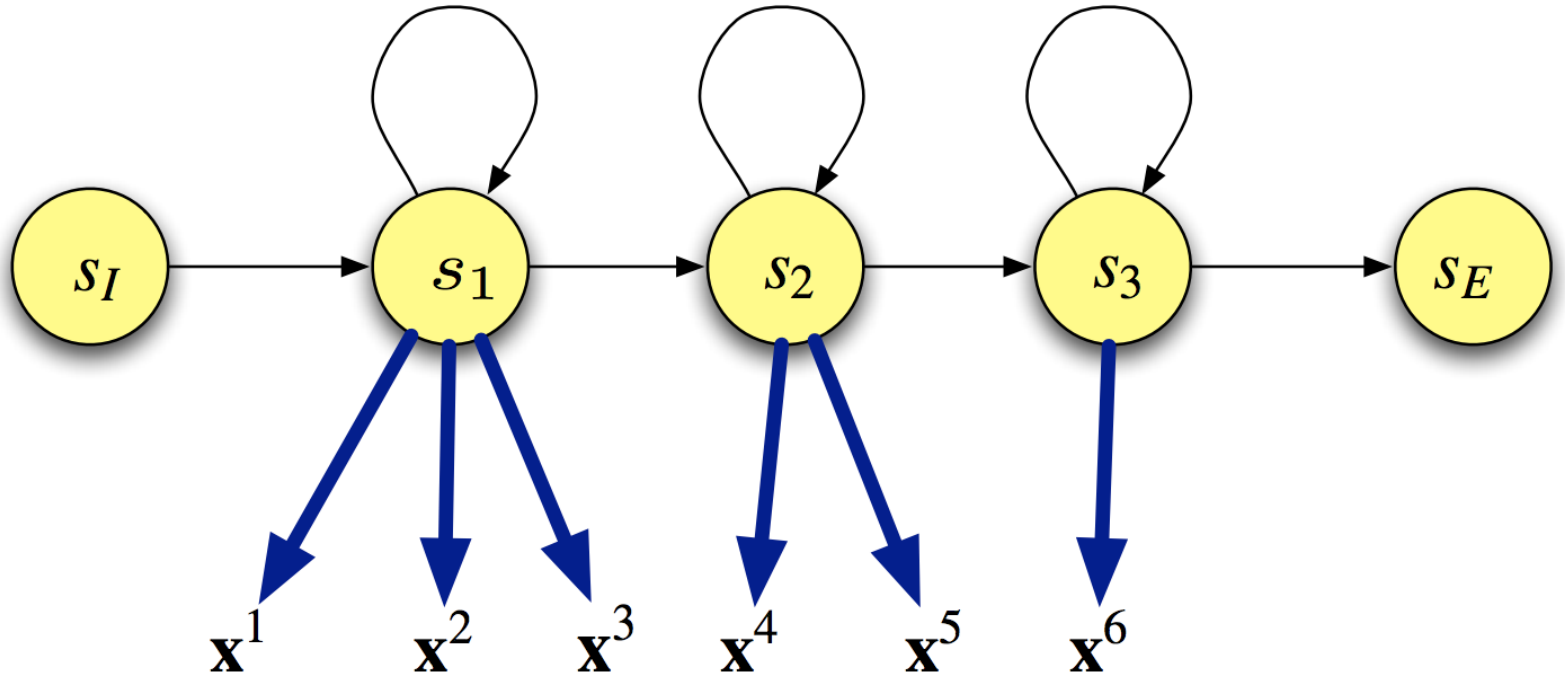
Probabilistic finite state automaton

Parameters $\boldsymbol{\lambda}$:

- Transition probabilities: $a_{kj} = P(S=j \mid S=k)$
- Output probability density function: $b_j(\mathbf{x}) = p(\mathbf{x} \mid S=j)$

NB: Some textbooks use Q or q to denote the state variable S.
$\mathbf{x}$ corresponds to $\mathbf{o}_t$ in Lecture slides 02.

Acoustic Model: Continuous Density HMM



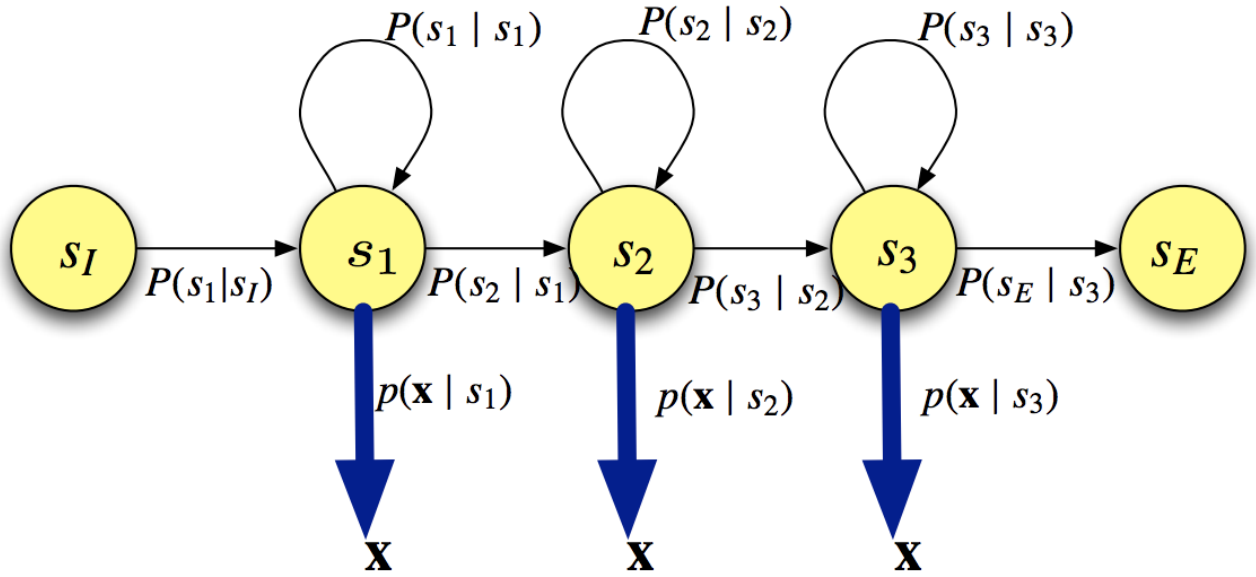
Probabilistic finite state automaton

Parameters $\boldsymbol{\lambda}$:

- Transition probabilities: $a_{kj} = P(S=j \mid S=k)$
- Output probability density function: $b_j(\mathbf{x}) = p(\mathbf{x} \mid S=j)$

NB: Some textbooks use Q or q to denote the state variable S.
$\mathbf{x}$ corresponds to $\mathbf{o}_t$ in Lecture slides 02.

Output distribution



- Single multivariate Gaussian with mean $\boldsymbol{\mu}_j$, covariance matrix $\boldsymbol{\Sigma}_j$:

$$b_j(\mathbf{x}) = p(\mathbf{x}|S=j) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

- M-component Gaussian mixture model:

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

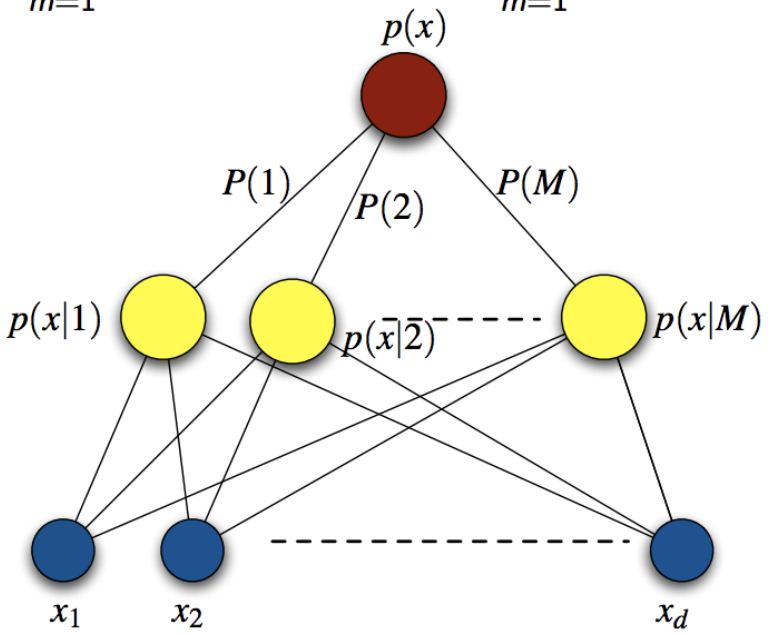
- Neural network:

$$b_j(\mathbf{x}) \sim P(S=j|\mathbf{x}) \,/\, P(S=j)$$ NB: NN outputs posterior probabilities

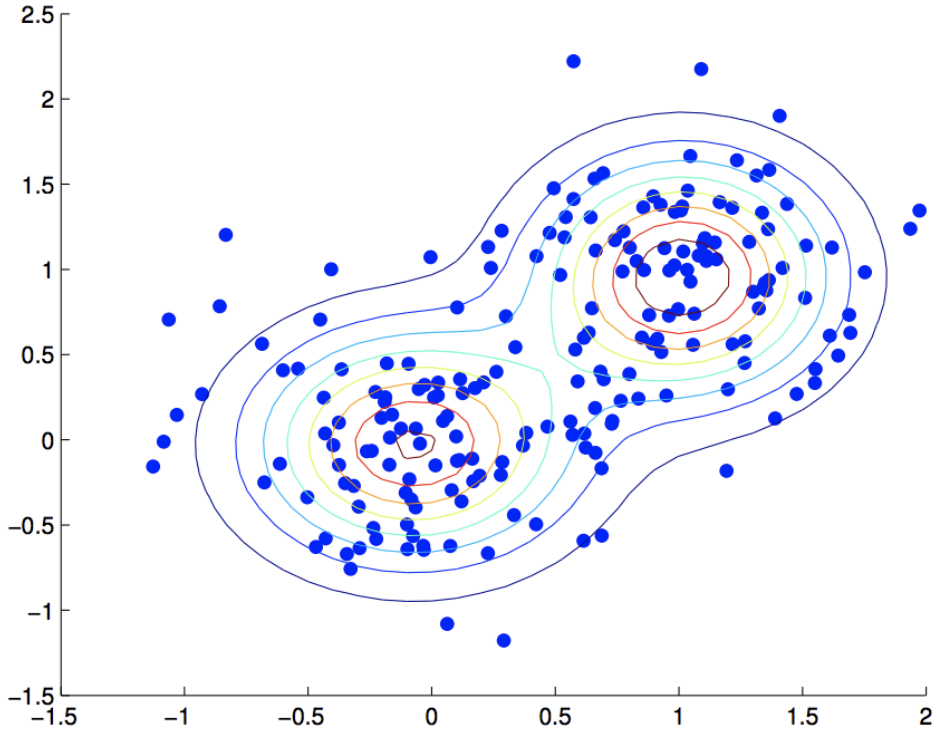
Gaussian mixture model

- The most important mixture model is the *Gaussian Mixture Model* (GMM), where the component densities are Gaussians
- Consider a GMM, where each component Gaussian $\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)$ has mean $\boldsymbol{\mu}_m$ and a spherical covariance $\boldsymbol{\Sigma}_m = \sigma_m^2 \mathbf{I}$

$$p(\mathbf{x}) = \sum_{m=1}^{M} P(m)\, p(\mathbf{x} \mid m) = \sum_{m=1}^{M} P(m)\, \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_m, \sigma_m^2 \mathbf{I})$$



Example 1 fit using a GMM



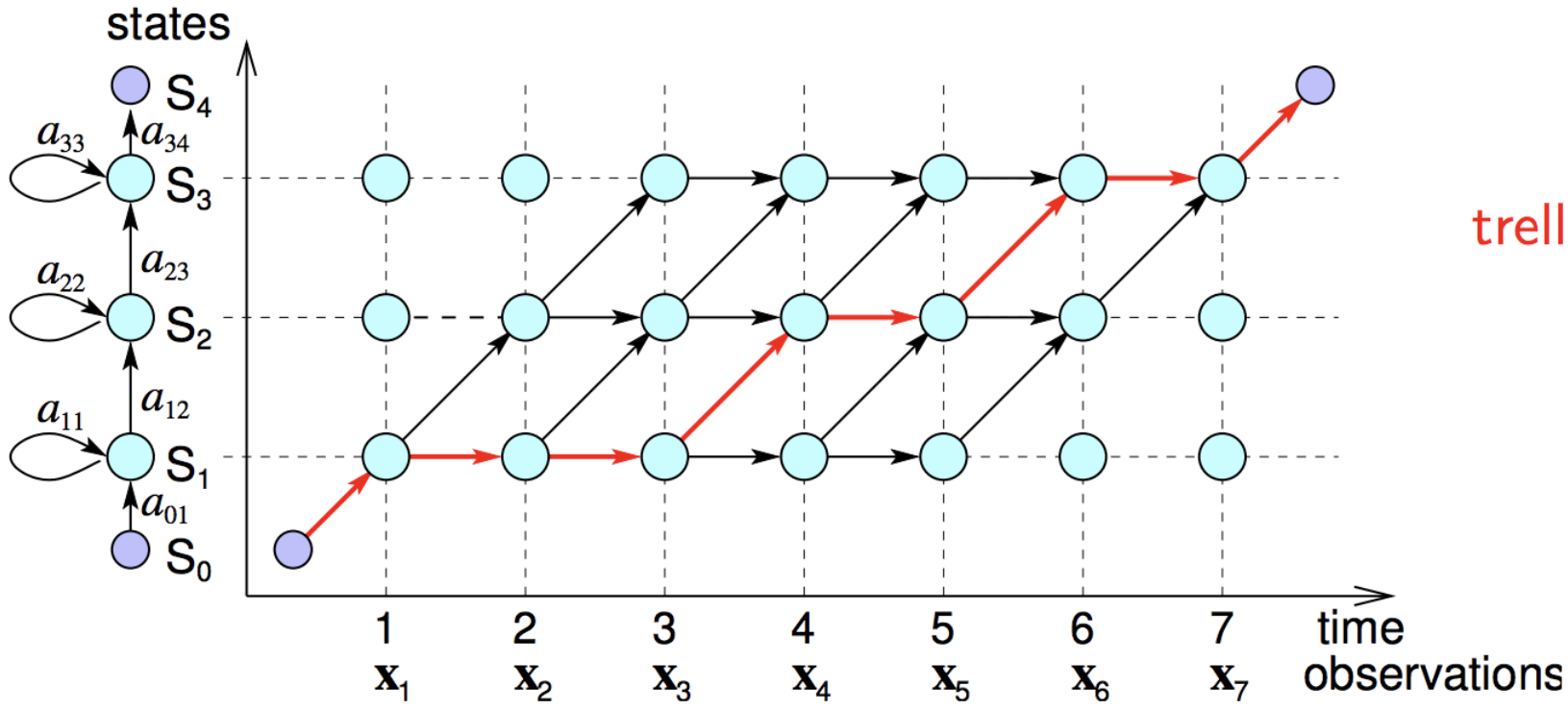
Fitted with a two component GMM using EM

The three problems of HMMs

Working with HMMs requires the solution of three problems:

1. **Likelihood** Determine the overall likelihood of an observation sequence $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_t, \ldots, \mathbf{x}_T)$ being generated by an HMM.
2. **Decoding** Given an observation sequence and an HMM, determine the most probable hidden state sequence
3. **Training** Given an observation sequence and an HMM, learn the best HMM parameters $\boldsymbol{\lambda} = \{\{a_{jk}\}, \{b_j()\}\}$

1. Likelihood: how to calculate?



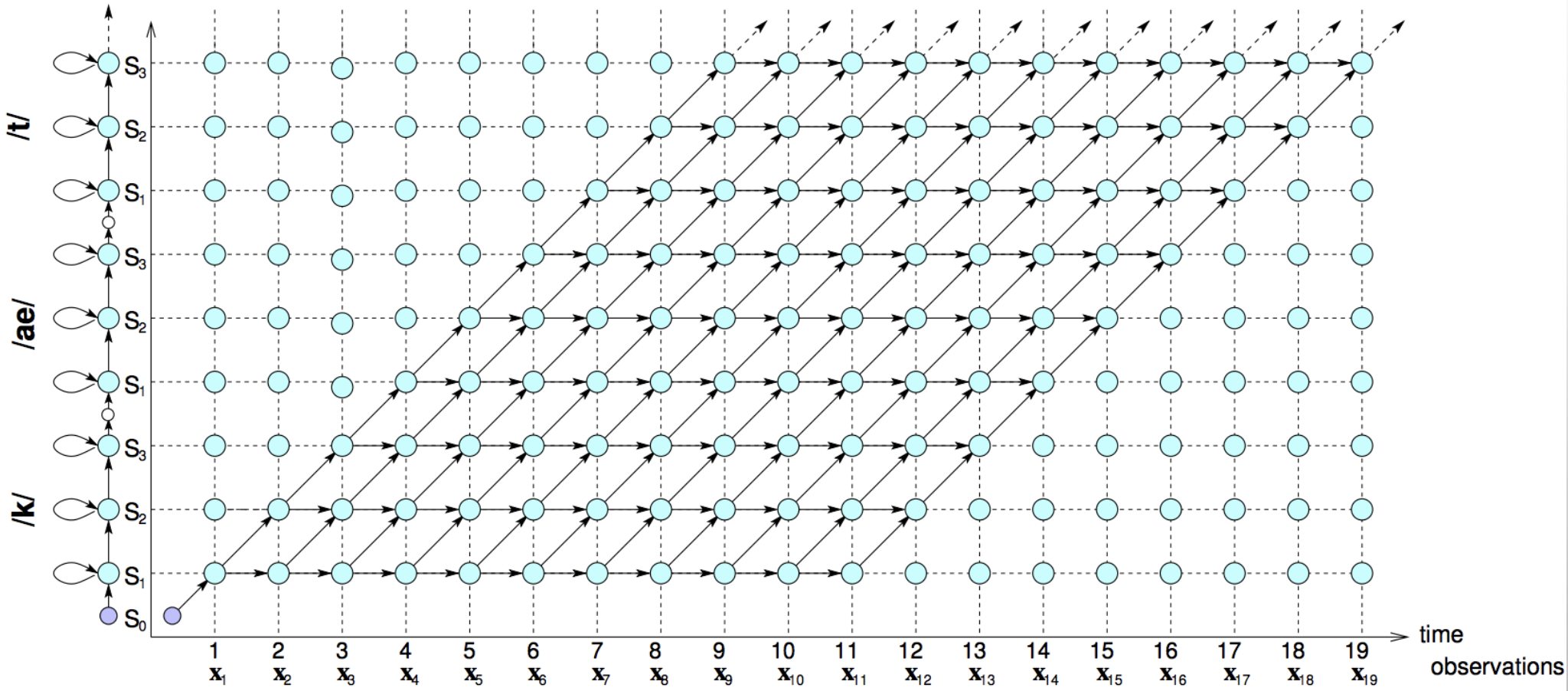
trellis

$$p(\mathbf{X}, \text{path}_\ell \,|\, \boldsymbol{\lambda}) = p(\mathbf{X} \,|\, \text{path}_\ell, \boldsymbol{\lambda}) P(\text{path}_\ell \,|\, \boldsymbol{\lambda})$$
$$= p(\mathbf{X} \,|\, s_0 s_1 s_1 s_1 s_2 s_2 s_3 s_3 s_4, \boldsymbol{\lambda}) P(s_0 s_1 s_1 s_1 s_2 s_2 s_3 s_3 s_4 \,|\, \boldsymbol{\lambda})$$
$$= b_1(\boldsymbol{x}_1) b_1(\boldsymbol{x}_2) b_1(\boldsymbol{x}_3) b_2(\boldsymbol{x}_4) b_2(\boldsymbol{x}_5) b_3(\boldsymbol{x}_6) b_3(\boldsymbol{x}_7) a_{01} a_{11} a_{11} a_{12} a_{22} a_{23} a_{33} a_{34}$$

$$p(\mathbf{X} \,|\, \boldsymbol{\lambda}) = \sum_{\{\text{path}_\ell\}} p(\mathbf{X}, \text{path}_\ell \,|\, \boldsymbol{\lambda}) \simeq \max_{\text{path}_\ell} p(\mathbf{X}, \text{path}_\ell \,|\, \boldsymbol{\lambda})$$

forward(backward) algorithm Viterbi algorithm

Trellis for /k ae t/



1. Likelihood: The Forward recursion

- Initialisation

$$\alpha_0(s_I) = 1$$
$$\alpha_0(j) = 0 \qquad \text{if } j \neq s_I$$

- Recursion

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(\mathbf{x}_t) \qquad 1 \leq j \leq N,\ 1 \leq t \leq T$$

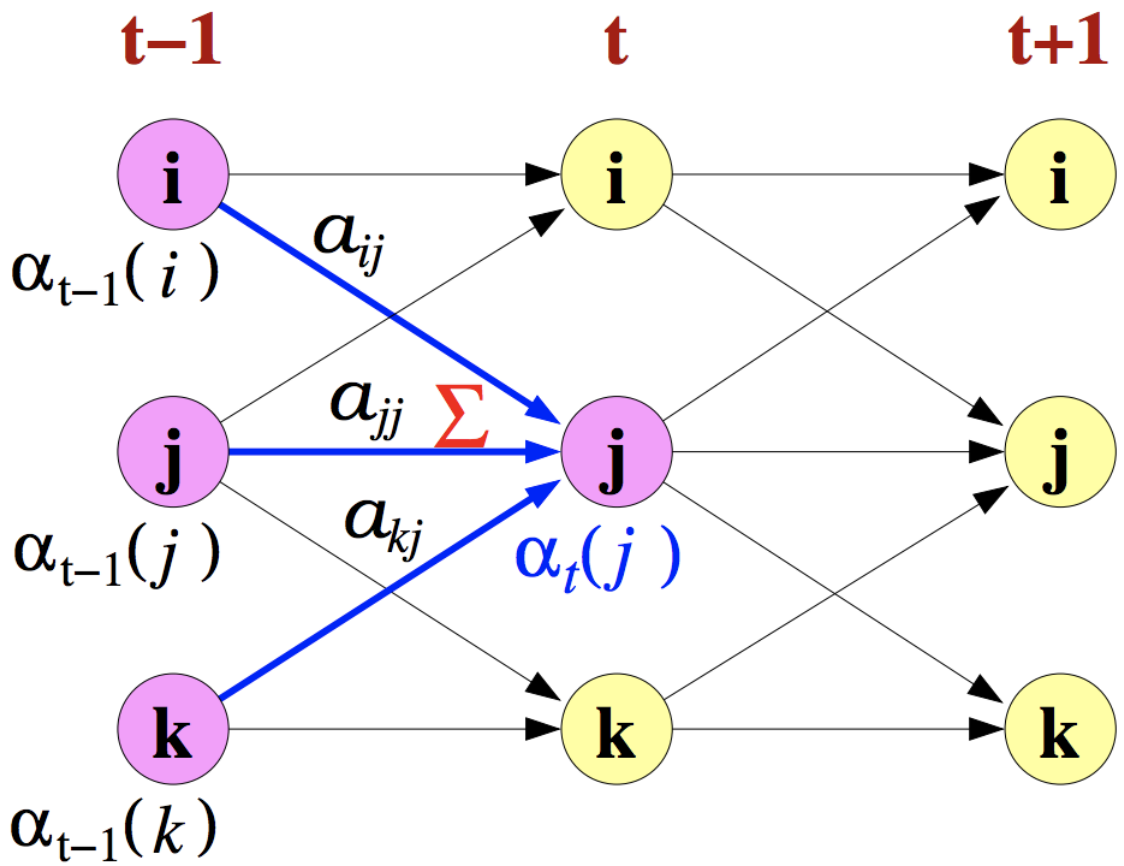
- Termination

$$p(\mathbf{X}|\boldsymbol{\lambda}) = \alpha_T(s_E) = \sum_{i=1}^{N} \alpha_T(i) a_{iE}$$

s_I: initial state, s_E: final state

1. Likelihood: Forward Recursion

$$\alpha_t(j) = p(\mathbf{x}_1, \ldots, \mathbf{x}_t, S(t)=j \mid \lambda) = \sum_{i=1}^{N} \alpha_{t-1}(i)\, a_{ij}\, b_j(\mathbf{x}_t)$$



Viterbi approximation

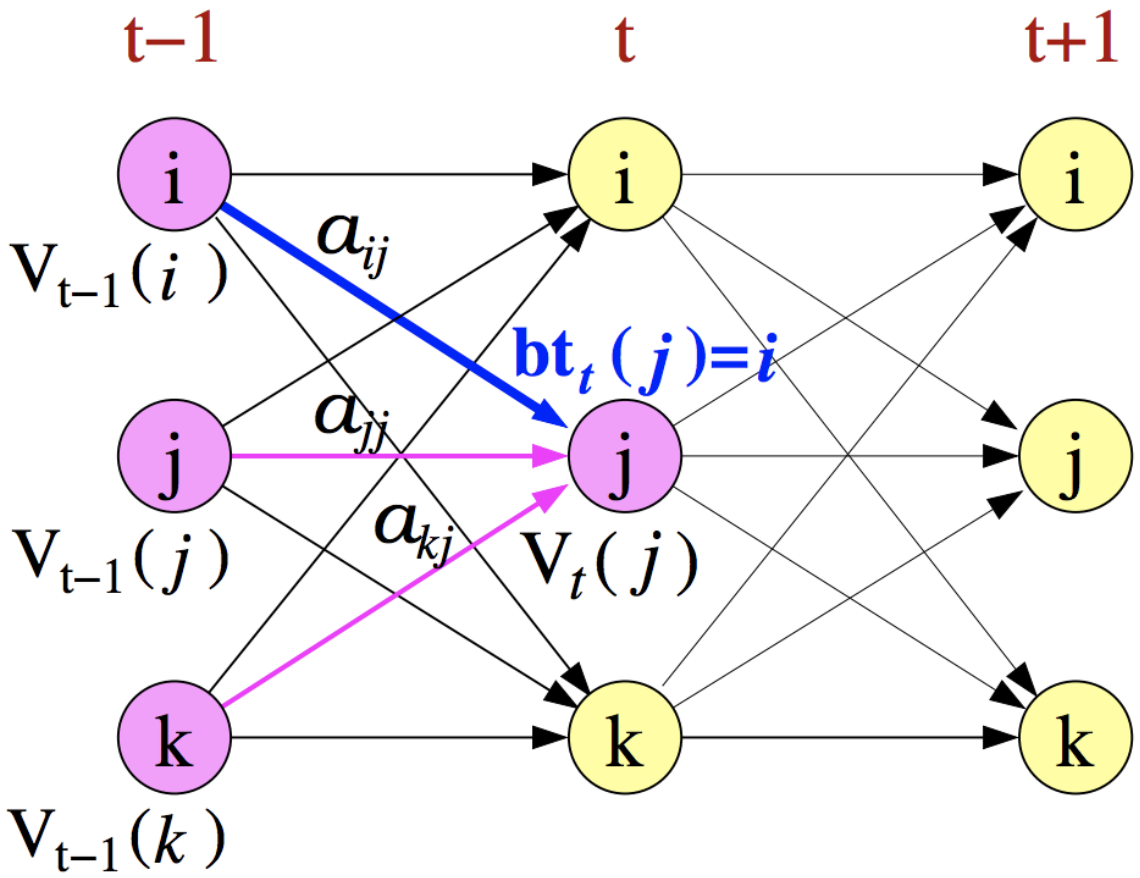
- Instead of summing over all possible state sequences, just consider the most likely
- Achieve this by changing the summation to a maximisation in the recursion:

$$V_t(j) = \max_i V_{t-1}(i) a_{ij} b_j(\mathbf{x}_t)$$

- Changing the recursion in this way gives the likelihood of the most probable path
- We need to keep track of the states that make up this path by keeping a sequence of *backpointers* to enable a Viterbi *backtrace*: the backpointer for each state at each time indicates the previous state on the most probable path

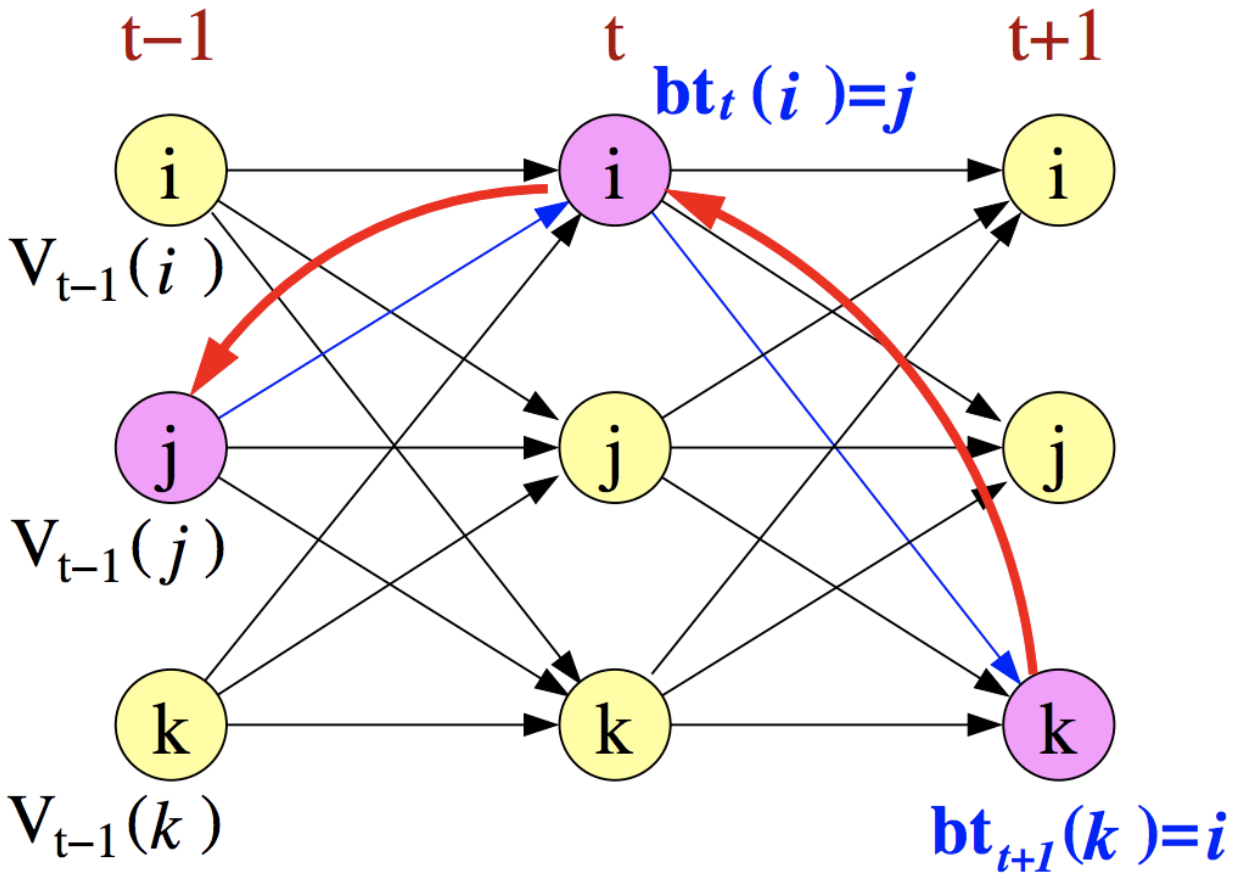
Viterbi Recursion

Backpointers to the previous state on the most probable path



Viterbi Backtrace

Backtrace to find the state sequence of the most probable path



2. Decoding: The Viterbi algorithm

- Initialisation

$$V_0(i) = 1$$
$$V_0(j) = 0 \qquad \text{if } j \neq i$$
$$\text{bt}_0(j) = 0$$

- Recursion

$$V_t(j) = \max_{i=1}^{N} V_{t-1}(i) a_{ij} b_j(\mathbf{x}_t)$$
$$\text{bt}_t(j) = \arg\max_{i=1}^{N} V_{t-1}(i) a_{ij} b_j(\mathbf{x}_t)$$

- Termination

$$P^* = V_T(s_E) = \max_{i=1}^{N} V_T(i) a_{iE}$$
$$s_T^* = \text{bt}_T(q_E) = \arg\max_{i=1}^{N} V_T(i) a_{iE}$$

3. Training: Forward-Backward algorithm

- Goal: Efficiently estimate the parameters of an HMM $\boldsymbol{\lambda}$ from an observation sequence
- Assume single Gaussian output probability distribution

$$b_j(\mathbf{x}) = p(\mathbf{x} \mid j) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

- Parameters $\boldsymbol{\lambda}$:
  - Transition probabilities a_{ij}:

$$\sum_j a_{ij} = 1$$

  - Gaussian parameters for state j:
    mean vector $\boldsymbol{\mu}_j$; covariance matrix $\boldsymbol{\Sigma}_j$

Viterbi Training

- If we knew the state-time alignment, then each observation feature vector could be assigned to a specific state
- A state-time alignment can be obtained using the most probable path obtained by Viterbi decoding
- Maximum likelihood estimate of a_{ij}, if $C(i \rightarrow j)$ is the count of transitions from i to j

$$\hat{a}_{ij} = \frac{C(i \rightarrow j)}{\sum_k C(i \rightarrow k)}$$

- Likewise if Z_j is the set of observed acoustic feature vectors assigned to state j, we can use the standard maximum likelihood estimates for the mean and the covariance:

$$\hat{\boldsymbol{\mu}}_j = \frac{\sum_{\boldsymbol{x} \in Z_j} \boldsymbol{x}}{|Z_j|}$$

$$\hat{\boldsymbol{\Sigma}}_j = \frac{\sum_{\boldsymbol{x} \in Z_j} (\boldsymbol{x} - \hat{\boldsymbol{\mu}}_j)(\boldsymbol{x} - \hat{\boldsymbol{\mu}}_j)^T}{|Z_j|}$$

EM Algorithm

- Viterbi training is an approximation—we would like to consider *all* possible paths
- In this case rather than having a hard state-time alignment we estimate a probability
- *State occupation probability*: The probability $\gamma_t(j)$ of occupying state j at time t given the sequence of observations.
  Compare with component occupation probability in a GMM
- We can use this for an iterative algorithm for HMM training: the EM algorithm (whose adaption to HMM is called 'Baum-Welch algorithm')
- Each iteration has two steps:

  E-step estimate the state occupation probabilities (Expectation)

  M-step re-estimate the HMM parameters based on the estimated state occupation probabilities (Maximisation)

Backward probabilities

- To estimate the state occupation probabilities it is useful to define (recursively) another set of probabilities—the *Backward probabilities*

$$\beta_t(j) = p(\mathbf{x}_{t+1}, \ldots, \mathbf{x}_T \,|\, S(t)=j, \boldsymbol{\lambda})$$

  The probability of future observations given a the HMM is in state j at time t
- These can be recursively computed (going backwards in time)
  - Initialisation

$$\beta_T(i) = a_{iE}$$

  - Recursion

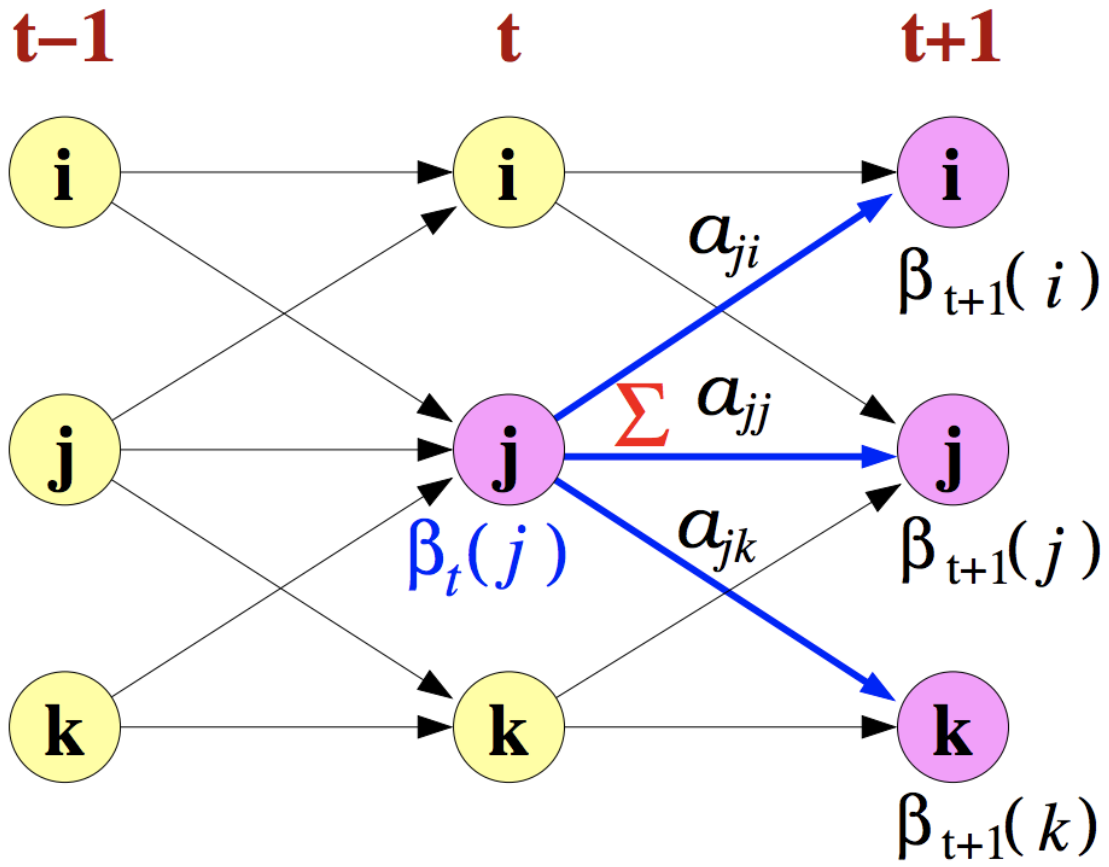
$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(\mathbf{x}_{t+1}) \beta_{t+1}(j) \quad \text{for } t = T-1, \ldots, 1$$

  - Termination

$$p(\mathbf{X}|\boldsymbol{\lambda}) = \beta_0(I) = \sum_{j=1}^{N} a_{Ij} b_j(\mathbf{x}_1) \beta_1(j) = \alpha_T(s_E)$$

Backward Recursion

$$\beta_t(j) = p(\mathbf{x}_{t+1}, \ldots, \mathbf{x}_T \,|\, S(t)=j, \boldsymbol{\lambda}) = \sum_{j=1}^{N} a_{ij} b_j(\mathbf{x}_{t+1}) \beta_{t+1}(j)$$



State Occupation Probability

- The **state occupation probability** $\gamma_t(j)$ is the probability of occupying state j at time t given the sequence of observations
- Express in terms of the forward and backward probabilities:

$$\gamma_t(j) = P(S(t)=j|\mathbf{X},\boldsymbol{\lambda}) = \frac{1}{\alpha_T(s_E)}\alpha_t(j)\beta_t(j)$$

recalling that $p(\mathbf{X}|\boldsymbol{\lambda}) = \alpha_T(s_E)$

- Since

$$\begin{aligned}\alpha_t(j)\beta_t(j) &= p(\mathbf{x}_1,\ldots,\mathbf{x}_t,S(t)=j|\boldsymbol{\lambda}) \\ &\qquad p(\mathbf{x}_{t+1},\ldots,\mathbf{x}_T|S(t)=j,\boldsymbol{\lambda}) \\ &= p(\mathbf{x}_1,\ldots,\mathbf{x}_t,\mathbf{x}_{t+1},\ldots,\mathbf{x}_T,S(t)=j|\boldsymbol{\lambda}) \\ &= p(\mathbf{X},S(t)=j|\boldsymbol{\lambda})\end{aligned}$$

$$P(S(t)=j|\mathbf{X},\boldsymbol{\lambda}) = \frac{p(\mathbf{X},S(t)=j|\boldsymbol{\lambda})}{p(\mathbf{X}|\boldsymbol{\lambda})}$$

Re-estimation of Gaussian parameters

- The sum of state occupation probabilities through time for a state, may be regarded as a "soft" count
- We can use this "soft" alignment to re-estimate the HMM parameters:

$$\hat{\boldsymbol{\mu}}_j = \frac{\sum_{t=1}^{T} \gamma_t(j)\boldsymbol{x}_t}{\sum_{t=1}^{T} \gamma_t(j)}$$

$$\hat{\boldsymbol{\Sigma}}_j = \frac{\sum_{t=1}^{T} \gamma_t(j)(\boldsymbol{x}_t - \hat{\boldsymbol{\mu}}_j)(\boldsymbol{x} - \hat{\boldsymbol{\mu}}_j)^T}{\sum_{t=1}^{T} \gamma_t(j)}$$

Re-estimation of transition probabilities

- Similarly to the state occupation probability, we can estimate $\xi_t(i, j)$, the probability of being in i at time t and j at $t+1$, given the observations:

$$\begin{aligned}\xi_t(i, j) &= P(S(t)=i, S(t+1)=j \mid \mathbf{X}, \boldsymbol{\lambda}) \\ &= \frac{p(S(t)=i, S(t+1)=j, \mathbf{X} \mid \boldsymbol{\lambda})}{p(\mathbf{X} \mid \boldsymbol{\lambda})} \\ &= \frac{\alpha_t(i) a_{ij} b_j(\mathbf{x}_{t+1}) \beta_{t+1}(j)}{\alpha_T(s_E)}\end{aligned}$$

- We can use this to re-estimate the transition probabilities

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

Pulling it all together

- Iterative estimation of HMM parameters using the EM algorithm. At each iteration

  E step For all time-state pairs

  1. Recursively compute the forward probabilities $\alpha_t(j)$ and backward probabilities $\beta_t(j)$
  2. Compute the state occupation probabilities $\gamma_t(j)$ and $\xi_t(i, j)$

  M step Based on the estimated state occupation probabilities re-estimate the HMM parameters: mean vectors $\boldsymbol{\mu}_j$, covariance matrices $\boldsymbol{\Sigma}_j$ and transition probabilities a_{ij}

- The application of the EM algorithm to HMM training is sometimes called the Forward-Backward algorithm or Baum-Welch algorithm

EM training of HMM/GMM

- Rather than estimating the state-time alignment, we estimate the component/state-time alignment, and component-state occupation probabilities $\gamma_t(j, m)$: the probability of occupying mixture component m of state j at time t.

  ($\xi_{tm}(j)$ in Jurafsky and Martin's SLP)

- We can thus re-estimate the mean of mixture component m of state j as follows

$$\hat{\boldsymbol{\mu}}_{jm} = \frac{\sum_{t=1}^{T} \gamma_t(j, m)\boldsymbol{x}_t}{\sum_{t=1}^{T} \gamma_t(j, m)}$$

  And likewise for the covariance matrices (mixture models often use diagonal covariance matrices)

- The mixture coefficients are re-estimated in a similar way to transition probabilities:

$$\hat{c}_{jm} = \frac{\sum_{t=1}^{T} \gamma_t(j, m)}{\sum_{m'=1}^{M} \sum_{t=1}^{T} \gamma_t(j, m')}$$

第二课 声学模型 GMM/DNN-HMM

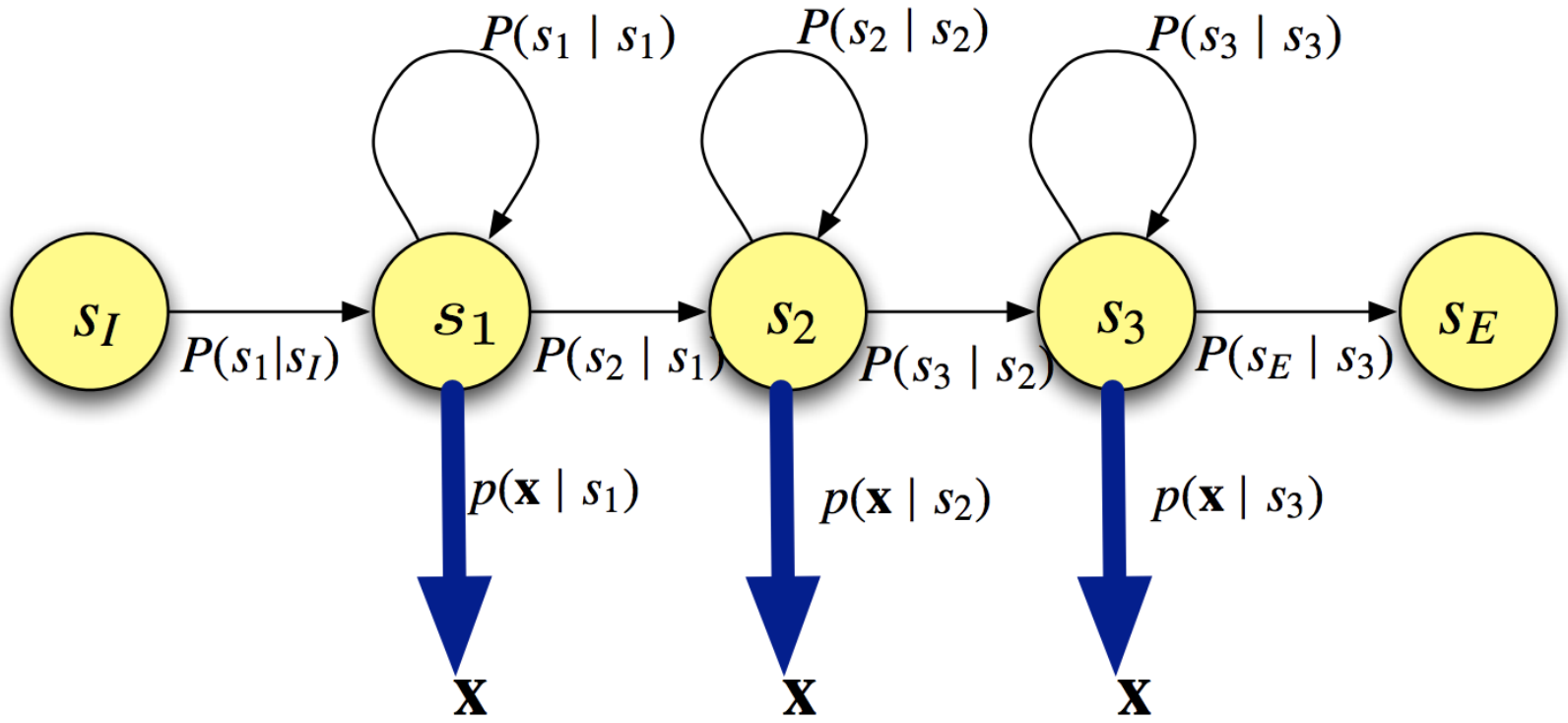
- 知识点1：单音素声学模型：MONOPHONE GMM-HMM语音模型
- 知识点2：考虑上下文的多音素声学模型: CONTEXT-DEPENDENT TRIPHONE GMM-HMM
- 知识点4：神经网络DNN原理
- 知识点5：从传统GMM-HMM到经典CD-DNN-HMM语音模型

- 知识点2：基于EM的模型训练

第二课 声学模型 GMM/DNN-HMM

- 知识点1：基于EM的模型训练
- 知识点2：单音素声学模型：MONOPHONE GMM-HMM语音模型
- 知识点3：考虑上下文的多音素声学模型: CONTEXT-DEPENDENT TRIPHONE GMM-HMM
- 知识点4：神经网络DNN原理
- 知识点5：从传统GMM-HMM到经典CD-DNN-HMM语音模型

Recap: Continuous Density HMM

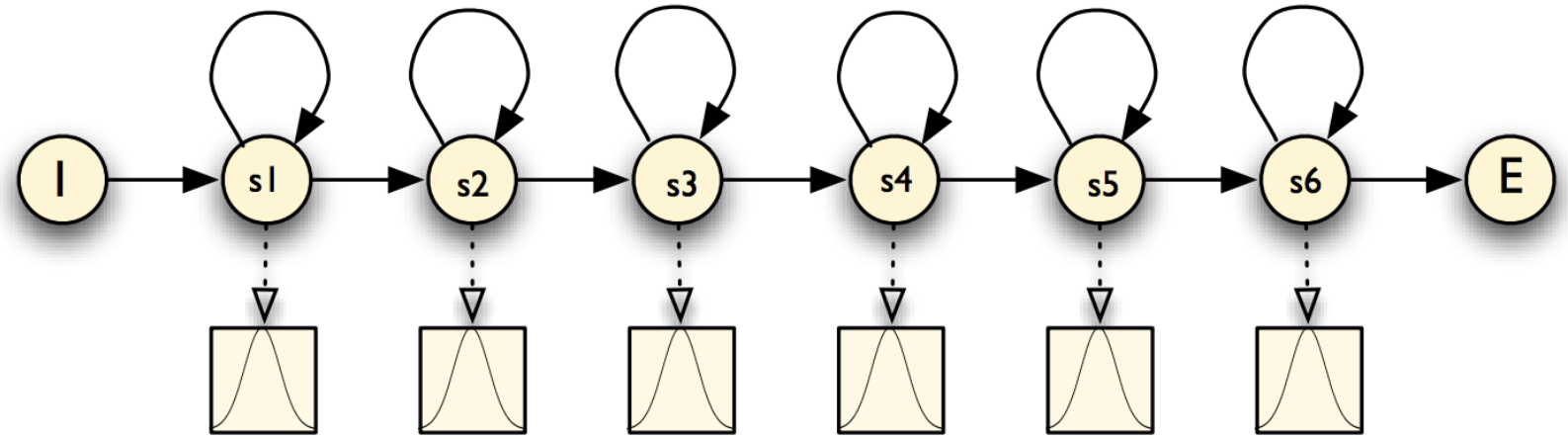


Probabilistic finite state automaton

Paramaters $\boldsymbol{\lambda}$:

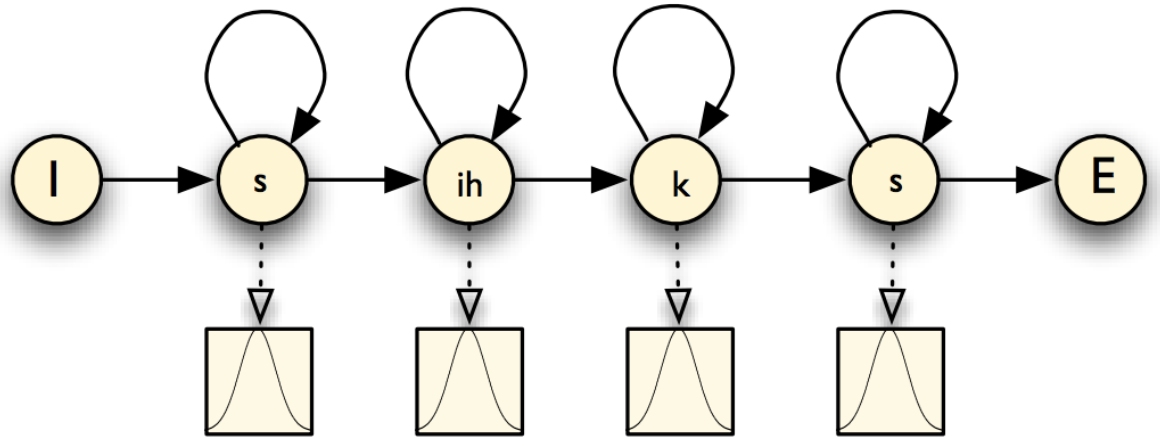
- Transition probabilities: $a_{kj} = P(s_j \mid s_k)$
- Output probability density function: $b_j(\mathbf{x}) = p(\mathbf{x} \mid s_j)$

Whole word models



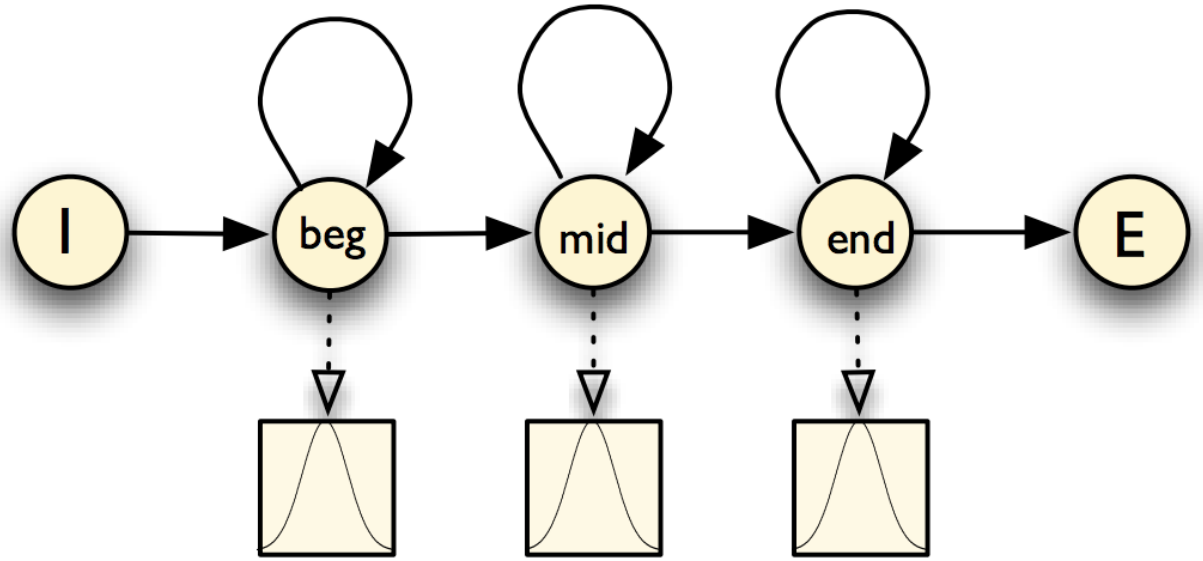
"six"

One state per phone models



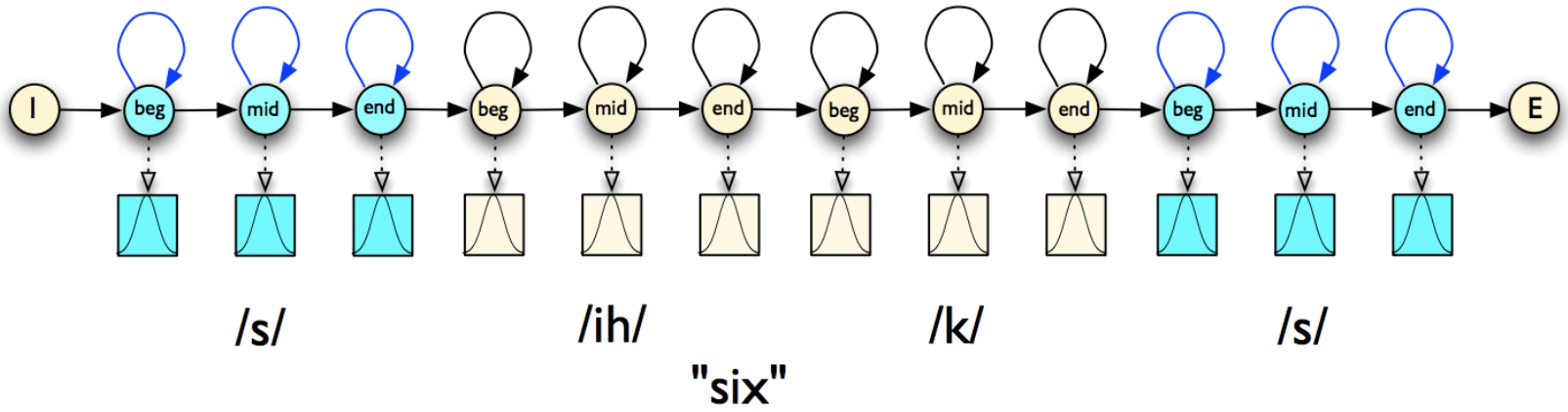
"six"

Three-state phone models

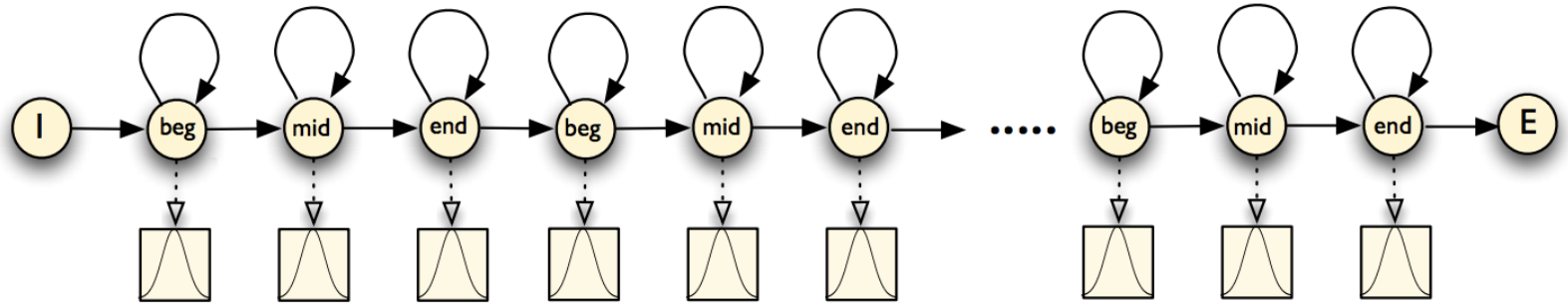


/ih/

Word model made of phone models



Word sequence models



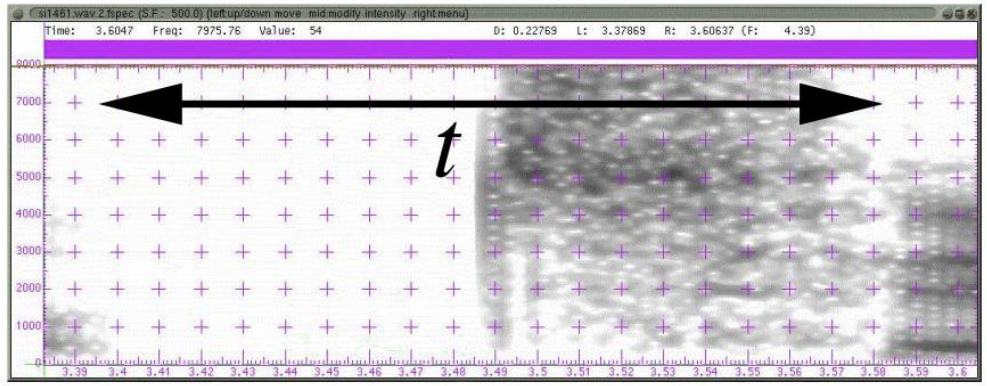
/s/ /ih/ ..... /d/

"six quid"

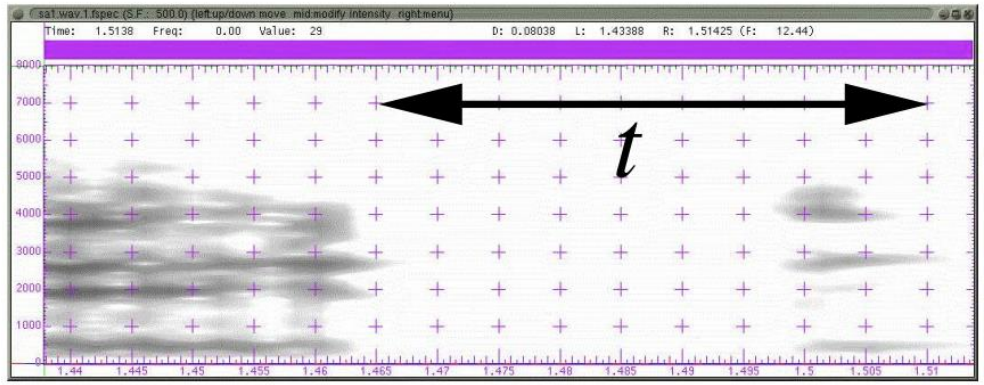
Phonetic Context

- Context The acoustic phonetic context of a speech unit has an effect on its acoustic realization
- Coarticulation the place of articulation for one speech sound depends on a neighbouring speech sound.
- Consider /n/ in ten and tenth
  - alveolar in ten
  - dental in tenth

Phonetic Context Example



"tube"



"suit"

Context-dependent phone models

- Triphones Each phone has a unique model for each left and right context. Represent a phone `x` with left context `l` and right context `r` as `l-x+r`
- Word-internal triphones Only take account of context within words, so "don't ask" is represented by:
  `sil d+oh d-oh+n oh-n+t n-t ah+s ah-s+k s-k sil`
  Word internal triphones result in far fewer models than cross-word models, and enable the subword sequence for a word to be known independent of the neighbouring words.
  But: context is not well-modelled at word boundaries.
- Cross-word triphones "don't ask" is represented by:
  `sil sil-d+oh d-oh+n oh-n+t n-t+ah t-ah+s ah-s+k s-k+sil sil`
  Note that triphone context extends across words (eg unit `n-t+ah`)

Triphone models

- How many triphones are there? Consider a 40 phone system. $40^3 = 64\,000$ possible triphones. In a cross-word system maybe 50 000 can occur
- Number of parameters:
  - 50 000 three-state HMMs, with 10 component Gaussian mixtures per state: 1.5M Gaussians
  - 39-dimension feature vectors (12 MFCCs + energy), deltas and accelerations
  - Assuming diagonal Gaussians: about 790 parameters/state
  - Total about 118 million parameters!
- We would need a very large amount of training data to train such a system
  - to enable robust estimation of all parameters
  - to ensure that all possible triphones are observed (more than once) in the training data

Modelling infrequent triphones

The number of possible triphone types is much greater than the number of observed triphone tokens.

- Smoothing – combine less-specific and more-specific models
- Parameter Sharing – different contexts share models
  - Bottom-up – start with all possible contexts, then merge
  - Top-down – start with a single context, then split
- All approaches are data driven
  NB: knowledge is used to make it work effectively

Parameter Sharing

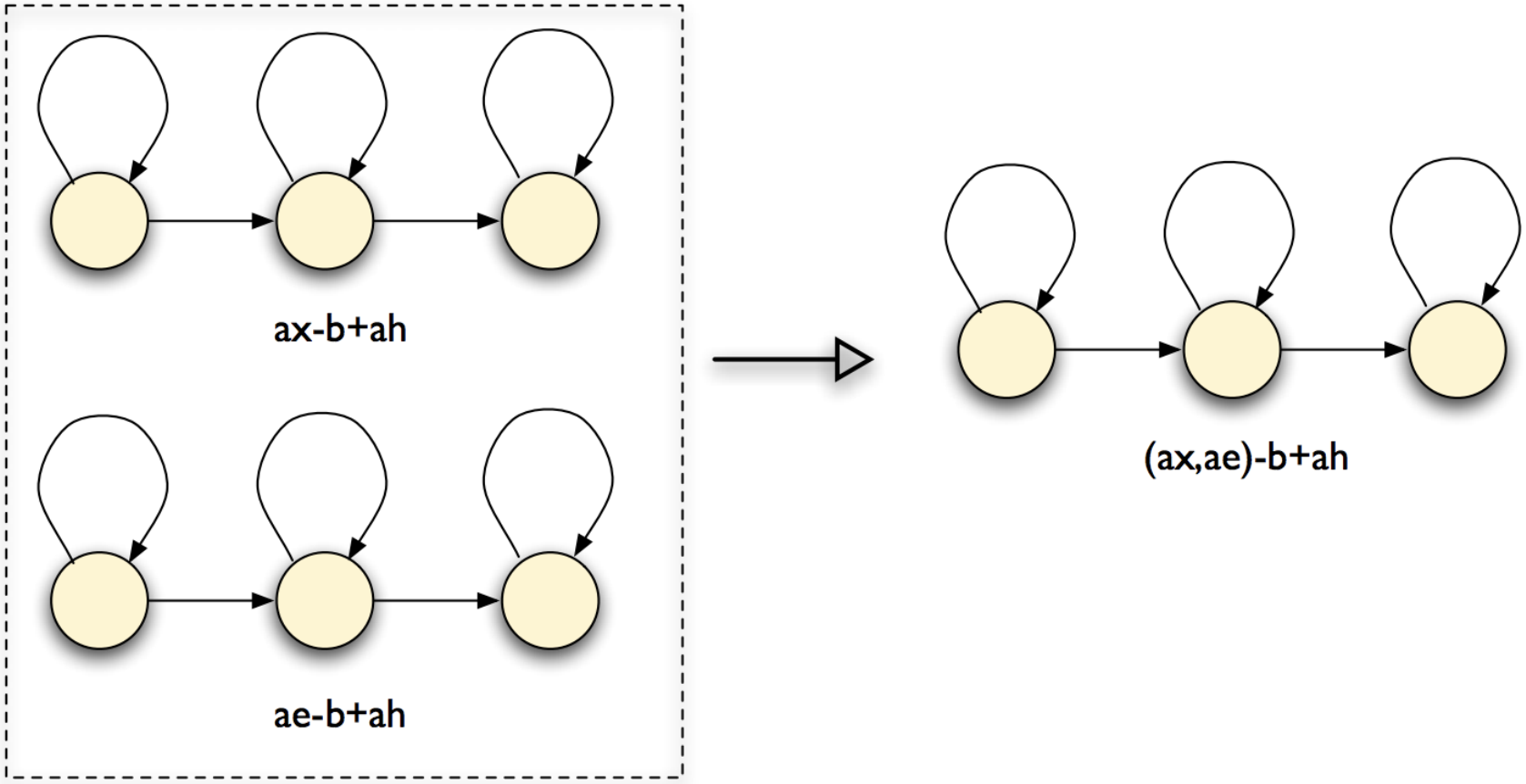
- Basic idea Explicitly share models or parameters between different contexts
  - enables training data to be shared between the models
  - enables models to share parameters
- Sharing can take place at different levels

1. Sharing Gaussians: all distributions share the same set of Gaussians but have different mixture weights (tied mixtures)
2. Sharing states: allow different models to share the same states (state clustering)
3. Sharing models: merge those context-dependent models that are the most similar (generalised triphones)

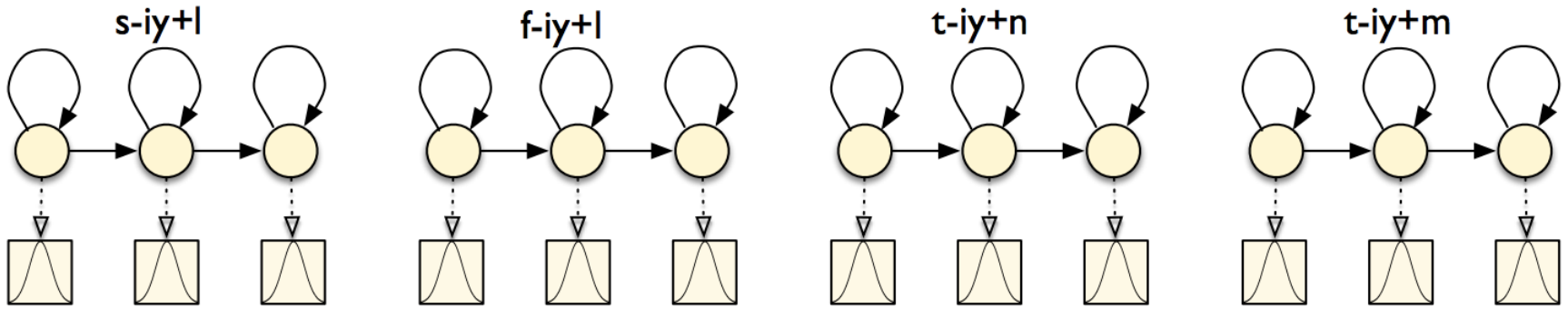
Sharing Models: Generalized triphones

- Basic idea Merge similar context-dependent models
- Bottom-up merging: Compare allophone models with different triphone contexts and merge those that are similar
- Merged models will be estimated from more data than individual models: more accurate models, fewer models in total
- The resultant merged models are referred to as generalized triphones

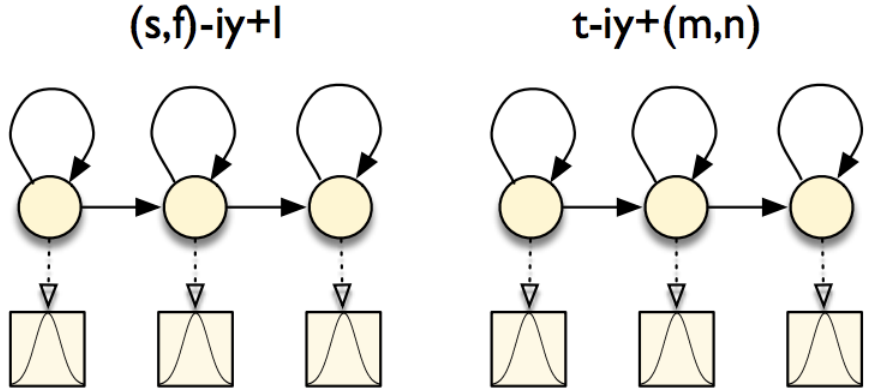
Example: Generalized Triphones



Example : Generalized triphones

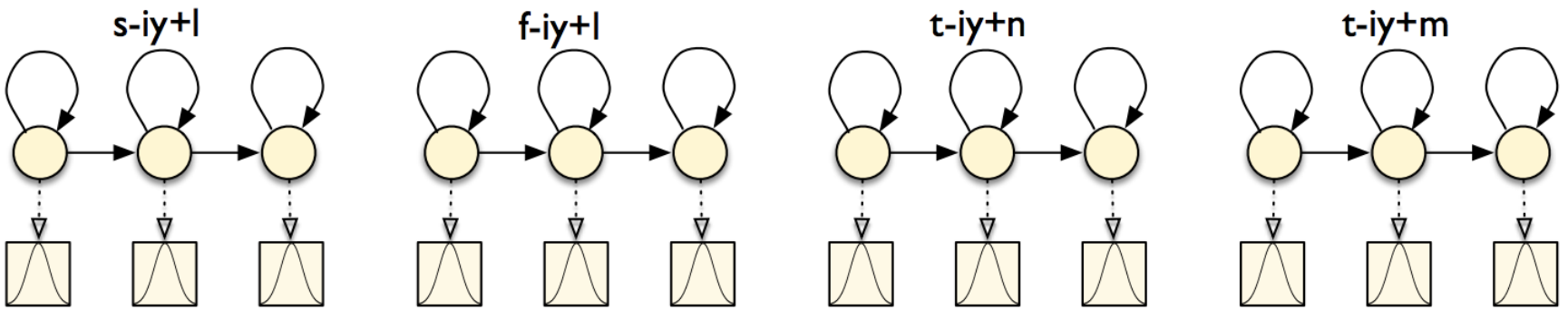


Simple triphones (no sharing)

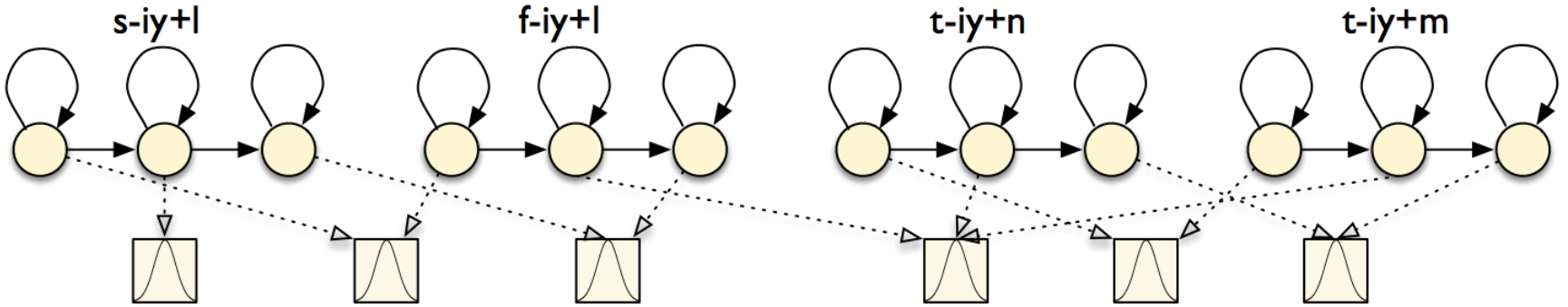


Generalized triphones (model sharing)

State Clustering



Simple triphones (no sharing)



State-clustered triphones (state sharing)

Sharing States: State clustering

- Basic idea States which are responsible for acoustically similar data are shared
- By clustering similar states, the training data associated with individual states may be pooled together – results in better parameter estimates for the state
  1. Create a set of context dependent models for a parent phone
  2. Cluster and tie similar states, ensuring that each resultant clustered state is responsible for "enough" training data (ie setting a minimum state occupation count)
- More flexible than clustering whole models: left and right contexts may be clustered separately

Good contexts to share

- Which states should be clustered together?
- Bottom-up clustering, for triphones of the same parent phone
  1. Create raw triphone models for each observed triphone context
  2. Cluster states as before

  Drawback: unable to solve unseen triphone problem
- Top-down clustering: start with a parent context independent model then successively split models to create context dependent models

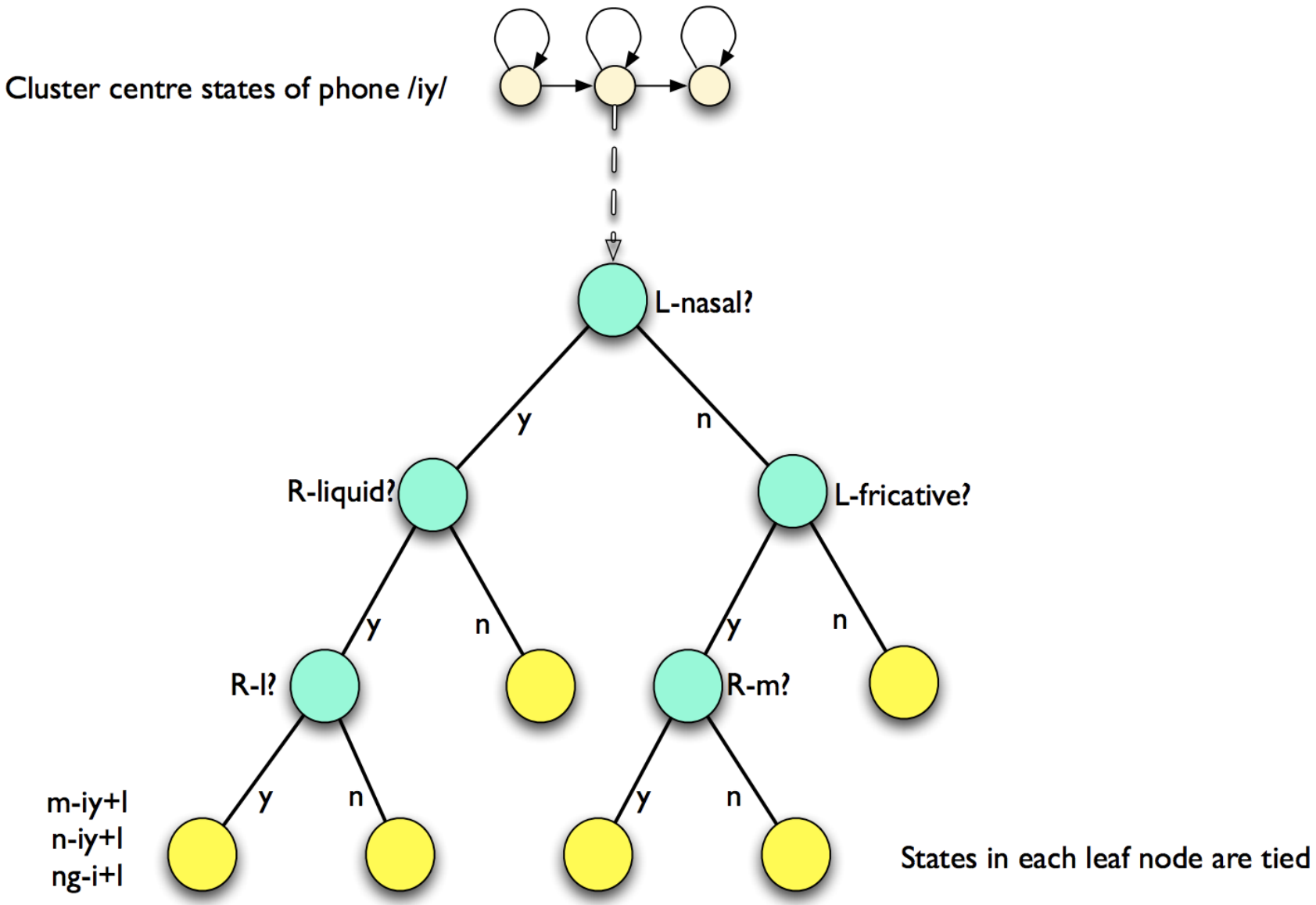
$$\text{Gain} = (L(\mathbf{S}_1) + L(\mathbf{S}_2)) - L(\mathbf{S})$$

Phonetic decision trees

Phonetic Decision Trees

- Basic idea Build a decision tree for each state of each parent phone, with yes/no questions at each node
- At the root of the tree, all states are shared
- Questions split the pool of states, the resultant state clusters are given by the leaves of the tree
- Example questions:
  - Is the left context a nasal?
  - Is the right context a central stop?
- The questions at each node are chosen from a large set of predefined questions
- Choose the question which maximizes the likelihood of the data given the state clusters
- Stop splitting if either: (a) the likelihood does not increase by more than a predefined threshold; or (b) the amount of data associated with a split node would below a threshold

Phonetic Decision Tree



Phonetic questions

- Ask questions of the form: does phone at offset s have feature f?
- Offsets are $+/-1$ for triphone context
- Example general questions:
  - Stop: `b d g p t k`
  - Nasal: `m n ng`
  - Fricative: `ch dh f jh s sh th v z zh`
  - Liquid: `l r w y`
  - Vowel: `aa ae ah ao aw ax axr ay eh er ...`
- Example consonant questions: Un/voiced, front/central/back, voiced stop, ...
- Example vowel questions: front, central, back, long, short, diphthong, rounded, ...
- Kaldi – generates all questions automatically using a top down binary clustering

Likelihood of a state cluster (1)

- Basic idea Compute the log likelihood of the data associated with a pool of states
- All states pooled in a single cluster at the root
- All states have Gaussian output pdf
- Let $\mathbf{S} = \{s_1, s_2, \ldots, s_K\}$ be a pool of K states forming a cluster, sharing a common mean $\boldsymbol{\mu}_S$ and covariance $\boldsymbol{\Sigma}_S$
- Let $\mathbf{X}$ be the set of training data
- Let $\gamma_s(\mathbf{x})$ be the probability that $\mathbf{x} \in \mathbf{X}$ was generated by state s (i.e. state occupation probability)
- The log likelihood of the data associated with cluster $\mathbf{S}$ is:

$$L(\mathbf{S}) = \sum_{s \in \mathbf{S}} \sum_{\mathbf{x} \in \mathbf{X}} \log P(\mathbf{x} | \boldsymbol{\mu}_S, \boldsymbol{\Sigma}_S) \gamma_s(\mathbf{x})$$

State splitting (1)

- Basic idea Use the likelihood of the parent state and of the split states to choose the splitting question
- Split $\mathbf{S}$ into two partitions $\mathbf{S}_y$ and $\mathbf{S}_n$ using a question about the phonetic context
- Each partition is now clustered together to form a single Gaussian output distribution with mean $\boldsymbol{\mu}_{S_y}$ and covariance $\boldsymbol{\Sigma}_{S_y}$) (for partition S_y)
- The likelihood of the data after partition is given by $L(\mathbf{S}_y) + L(\mathbf{S}_n)$
- The total likelihood of the partitioned data will increase by

$$\Delta = L(\mathbf{S}_y) + L(\mathbf{S}_n) - L(\mathbf{S})$$

“Mixing up”

- Basic idea Transforming an HMM-based system based on Gaussian distributions to one based on mixtures of Gaussians
- The above methods for state clustering assume that the state outputs are Gaussians – this makes the computations much simpler
- BUT: Gaussian mixtures offer much better acoustic models than Gaussians
- Solution:
  - Perform state clustering using Gaussian distributions
  - Split the Gaussian distributions in the clustered states, by cloning and perturbing the means by a small fraction of the standard deviation, and retrain.
  - Repeat by splitting the dominant (highest state occupation count) mixture components in each state

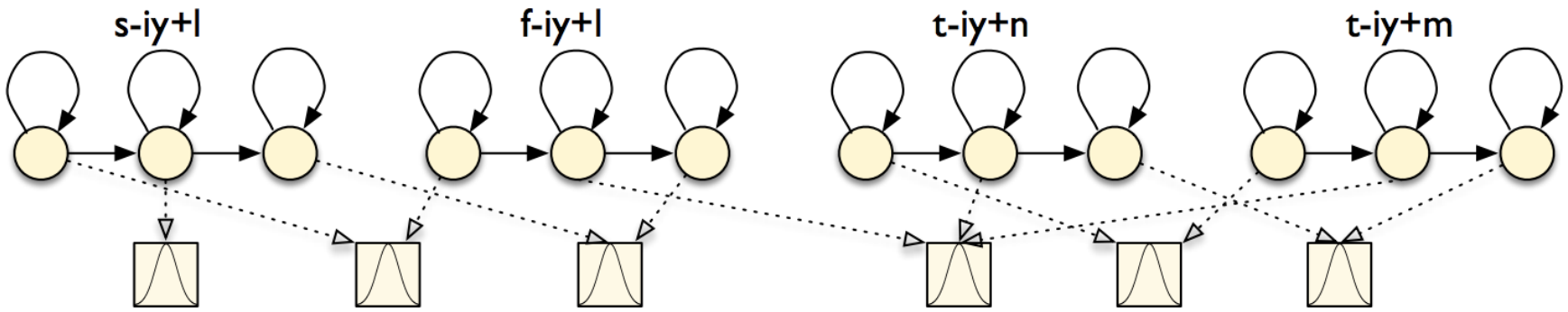
State splitting (2)

- Basic idea Use the likelihood of the parent state and of the split states to choose the splitting question

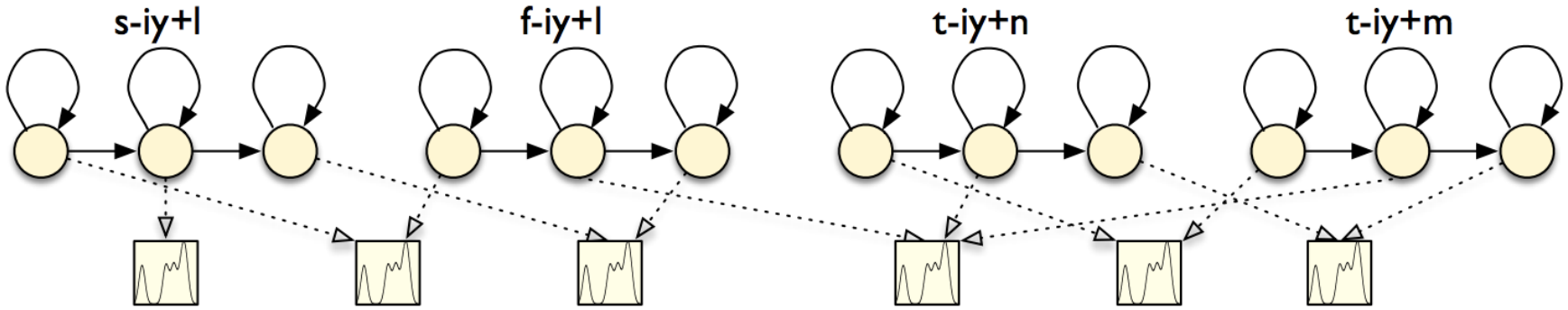
$$\Delta = L(\mathbf{S}_y) + L(\mathbf{S}_n) - L(\mathbf{S})$$

- Cycle through all possible questions, compute Δ for each and choose the question for which Δ is biggest
- Continue by splitting each of the new clusters $\mathbf{S}_y$ and $\mathbf{S}_n$
- Terminate when
  1. Maximum Δ falls below a threshold
  2. The amount of data associated with a split node falls below a threshold
- For a Gaussian output distribution: State likelihood estimates can be estimated using just the *state occupation counts* (obtained at alignment) and the parameters of the Gaussian – no need to use the acoustic data
- State occupation count: sum of state occupation probabilities for a state over time

“Mixing up”



State-clustered triphones (Gaussians)



State-clustered triphones (GMMs)

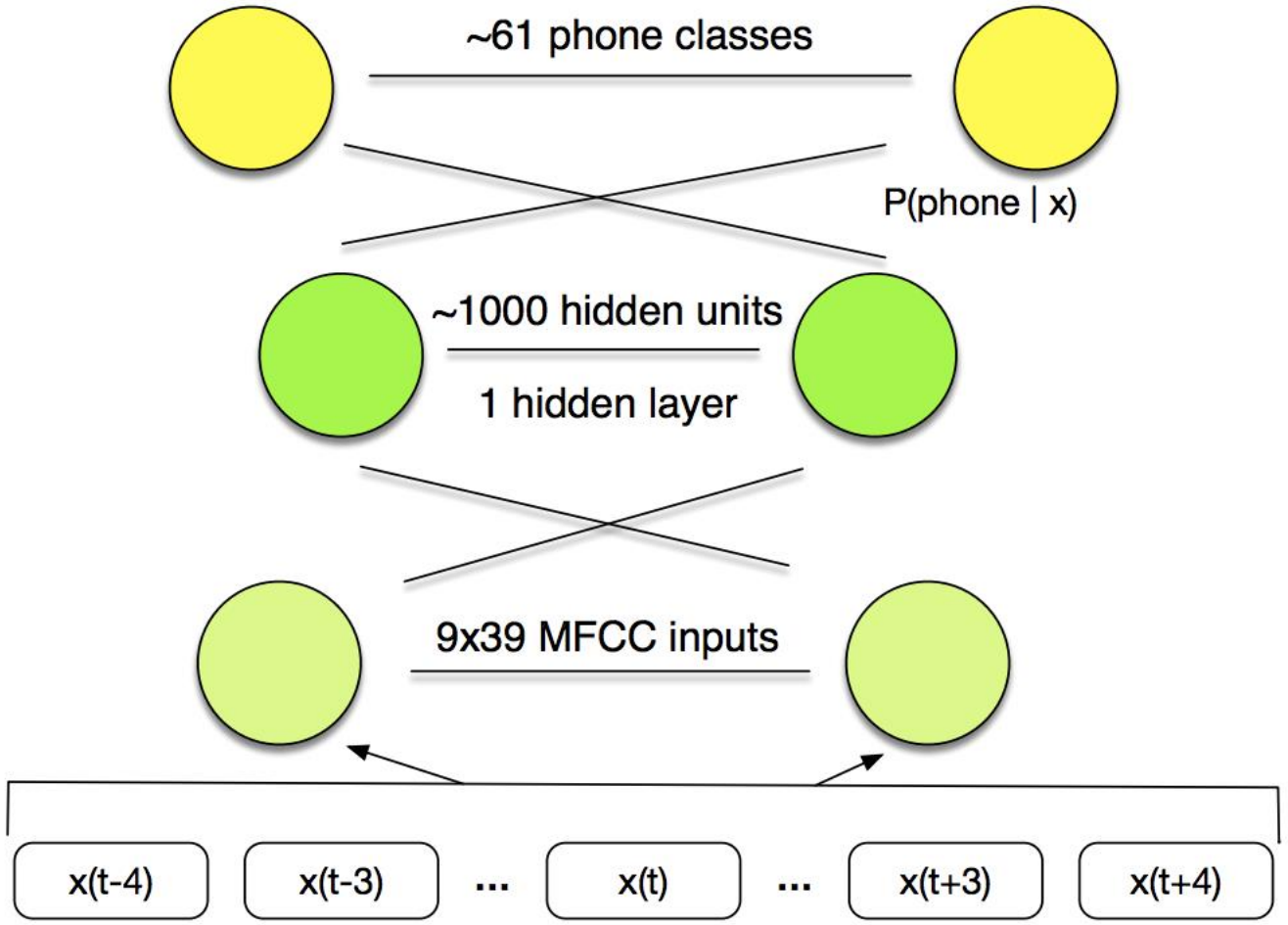
Summary: Context-dependent phone models

- Share parameters through state clustering
- Cluster states using phonetic decision trees for each state of parent phone
- Use Gaussian distributions when state clustering
- Then split Gaussians and retrain to obtain a GMM state clustered system

第二课 声学模型 GMM/DNN-HMM

- 知识点1：基于EM的模型训练
- 知识点2：单音素声学模型：MONOPHONE GMM-HMM语音模型
- 知识点3：考虑上下文的多音素声学模型: CONTEXT-DEPENDENT TRIPHONE GMM-HMM
- 知识点4：神经网络DNN原理
- 知识点5：从传统GMM-HMM到经典CD-DNN-HMM语音模型

Simple neural network for phone classification



Interim conclusions

- Neural networks using cross-entropy (CE) and softmax outputs give us a way of assigning the probability of each possible phonetic label for a given frame of data
- Hidden layers provide a way for the system to learn representations of the input data
- All the weights and biases of a network may be trained by gradient descent – back-propagation of error provides a way to compute the gradients in a deep network
- Acoustic context can be simply incorporated into such a network by providing multiples frame of acoustic input

Neural networks for phone recognition

- Train a neural network to associate a phone label with a frame of acoustic data (+ context)
- Can interpret the output of the network as P(phone | acoustic-frame)
- Hybrid NN/HMM systems: in an HMM, replace the GMMs used to estimate output pdfs with the outputs of neural networks
- One-state per phone HMM system:
  - Train an NN as a phone classifier (= phone probability estimator)
  - Use NN to obtain output probabilities in Viterbi algorithm to find most probable sequence of phones (words)

Neural networks and posterior probabilities

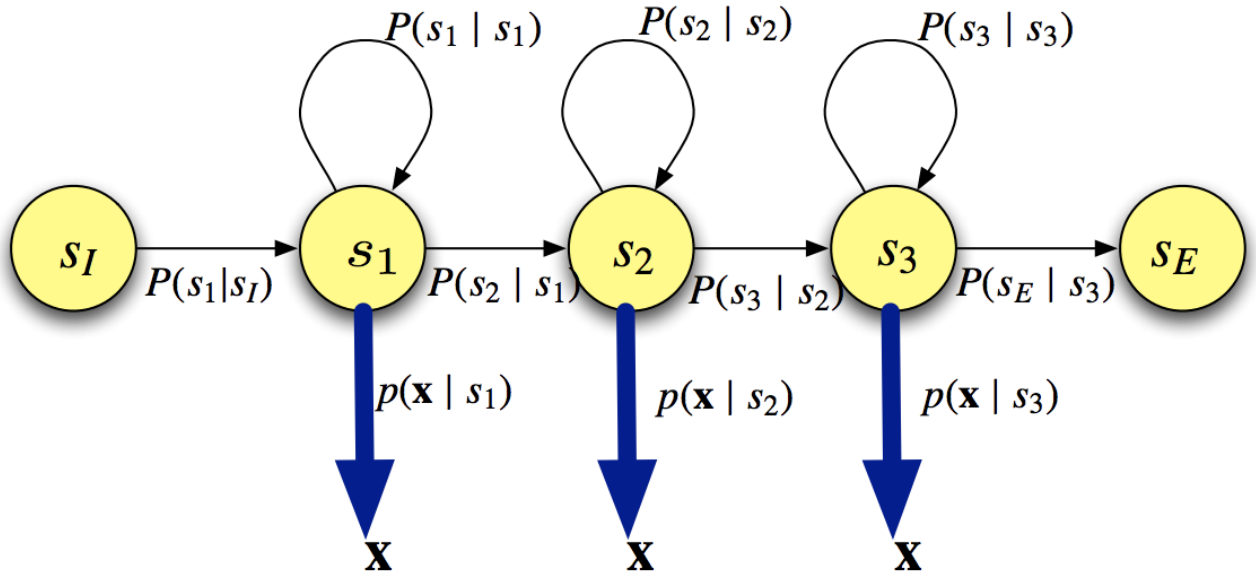
Posterior probability estimation

- Consider a neural network trained as a classifier – each output corresponds to a class.
- When applying a trained network to test data, it can be shown that the value of output corresponding to class q given an input $\mathbf{x}$, is an estimate of the posterior probability $P(q|\mathbf{x})$. (This is because we have softmax outputs and use a cross-entropy loss function)
- Using Bayes Rule we can relate the posterior $P(q|\mathbf{x})$ to the likelihood $p(\mathbf{x}|q)$ used as an output probability in an HMM:

$$P(q|\mathbf{x}) = \frac{p(\mathbf{x}|q)P(q)}{p(\mathbf{x})}$$

(this is assuming 1 state per phone q)

Output distribution



- Single multivariate Gaussian with mean $\boldsymbol{\mu}_j$, covariance matrix $\boldsymbol{\Sigma}_j$:

$$b_j(\mathbf{x}) = p(\mathbf{x}|S=j) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

- M-component Gaussian mixture model:

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

- Neural network:

$$b_j(\mathbf{x}) \sim P(S=j|\mathbf{x}) \,/\, P(S=j)$$ NB: NN outputs posterior probabilities

Scaled likelihoods

- If we would like to use NN outputs as output probabilities in an HMM, then we would like probabilities (or densities) of the form $p(\mathbf{x}|q)$ – likelihoods.
  We can write *scaled likelihoods* as:

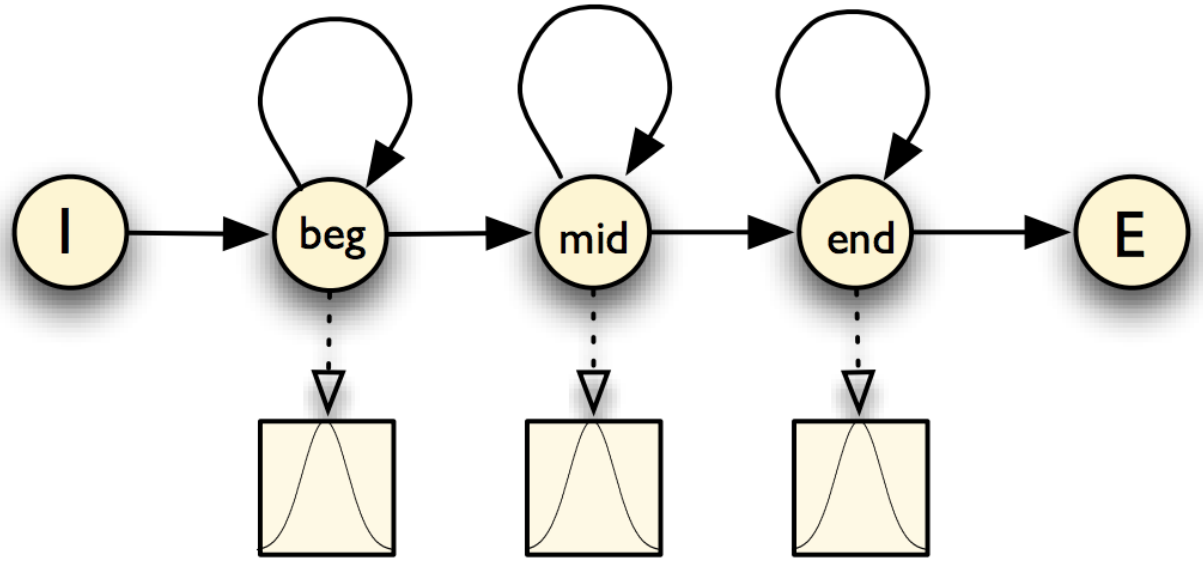
$$\frac{P(q|\mathbf{x})}{p(q)} = \frac{p(\mathbf{x}|q)}{p(\mathbf{x})}$$

- Scaled likelihoods can be obtained by "dividing by the priors" – divide each network output $P(q|\mathbf{x})$ by $P(q)$, the relative frequency of class q in the training data
- Using $p(\mathbf{x}|q)/p(\mathbf{x})$ rather than $p(\mathbf{x}|q)$ is OK since $p(\mathbf{x})$ does not depend on the class q
- Use the scaled likelihoods obtained from a neural network in place of the usual likelihoods obtained from a GMM

Hybrid NN/HMM

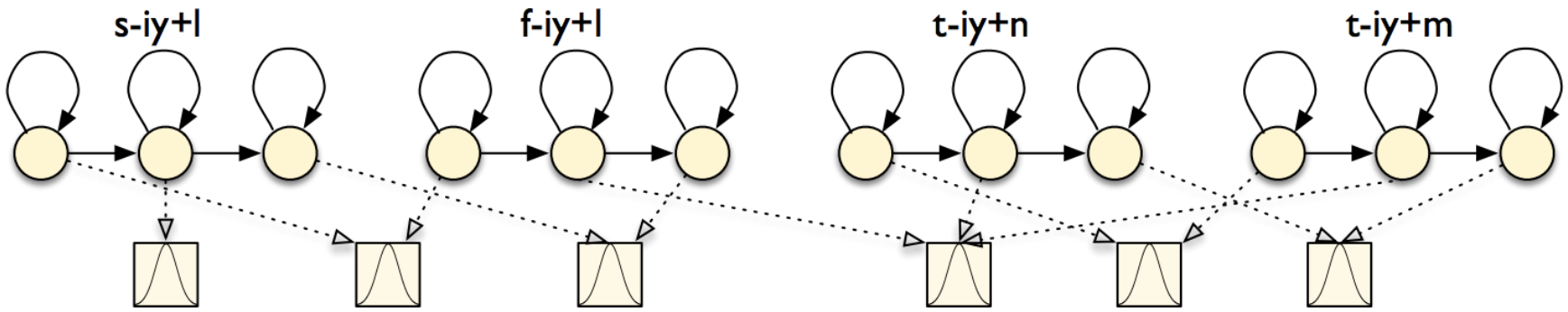
- If we have a K-state HMM system, then we train a K-output NN to estimate the scaled likelihoods used in a hybrid system.
- For TIMIT, using a 1 state per phone systems, we obtain scaled likelihoods from a NN trained to classify phones.
- For continuous speech recognition we can use:
  - 1 state per phone (61 NN outputs, if we have 61 phone classes)
  - 3 state CI models ($61 \times 3 = 183$ NN outputs)
  - State-clustered models, with one NN output per tied state (this can lead to networks with many outputs!) (next lecture)
- Scaled likelihood and dividing by the priors
  - Computing the scaled likelihoods can be interpreted as factoring out the prior estimates for each phone based on the acoustic training data. The HMM can then integrate better prior estimates based on the language model and lexicon.

Three-state phone models

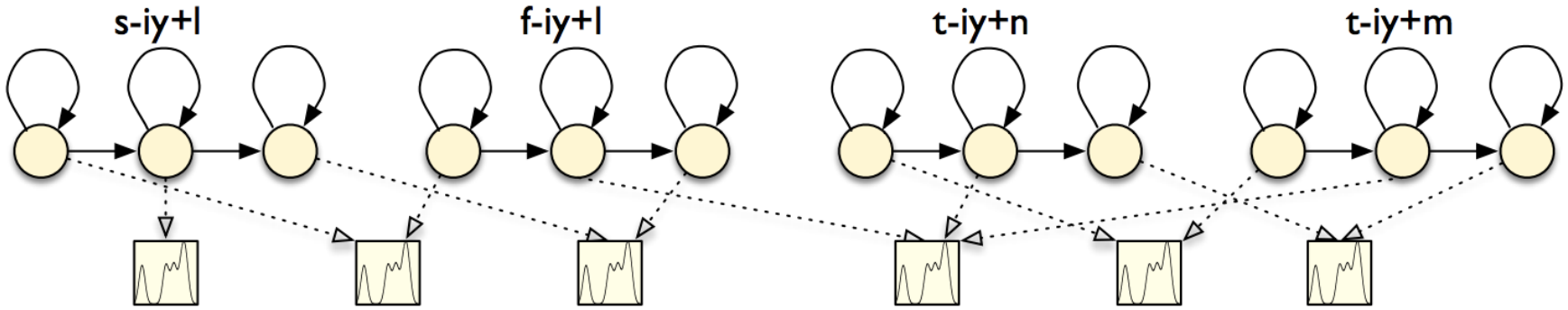


/ih/

“Mixing up”

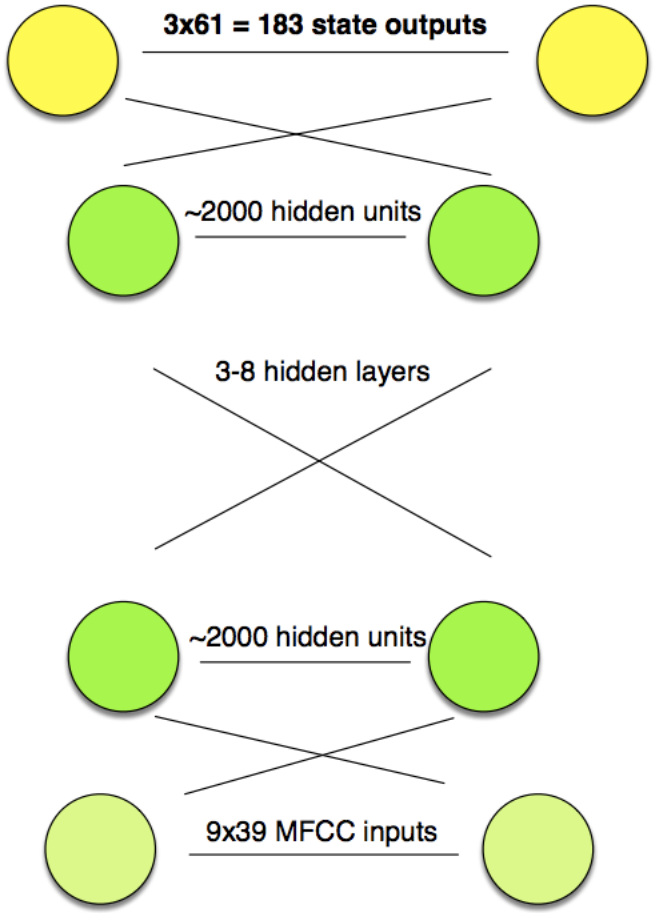


State-clustered triphones (Gaussians)



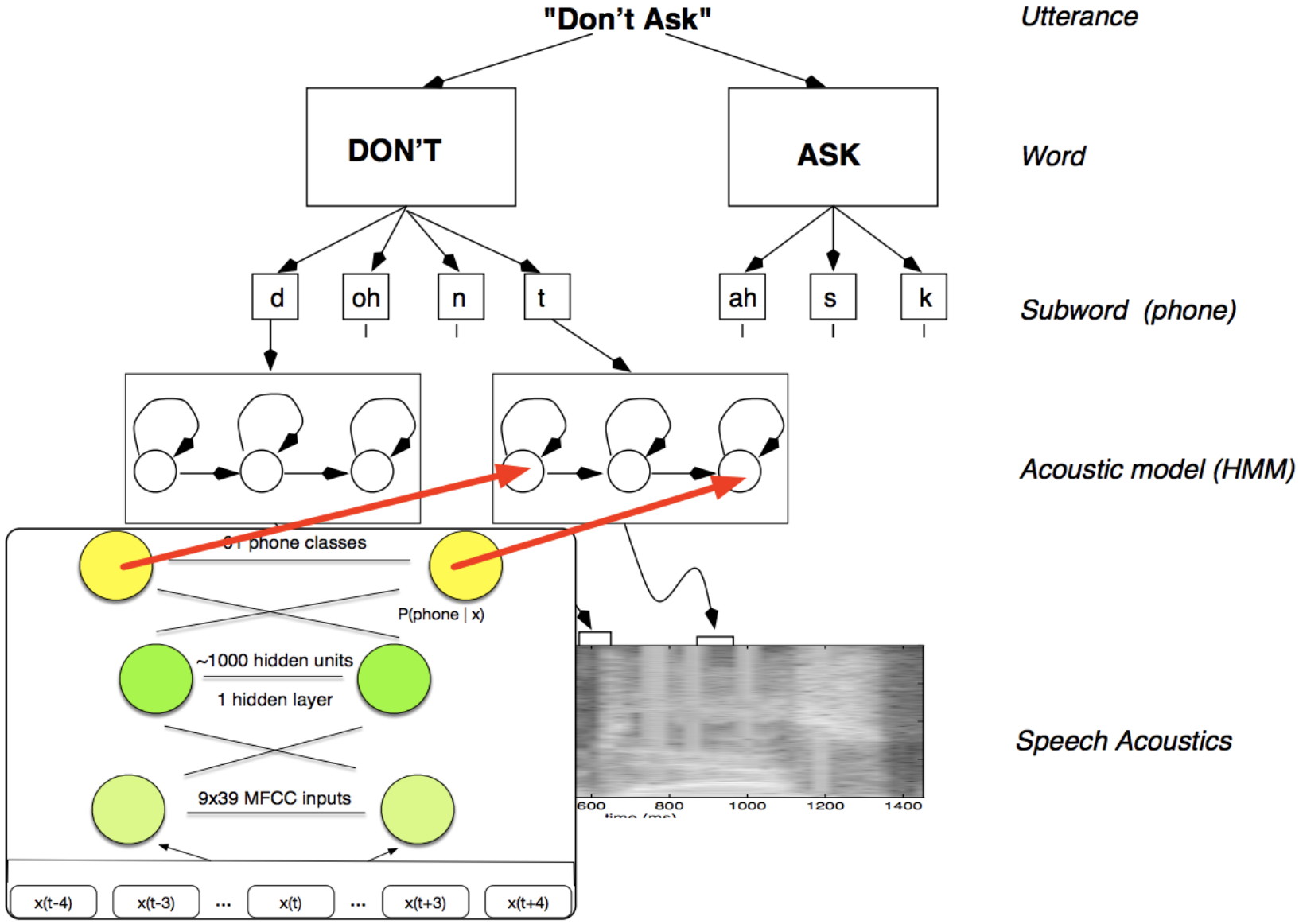
State-clustered triphones (GMMs)

Deep neural network for TIMIT

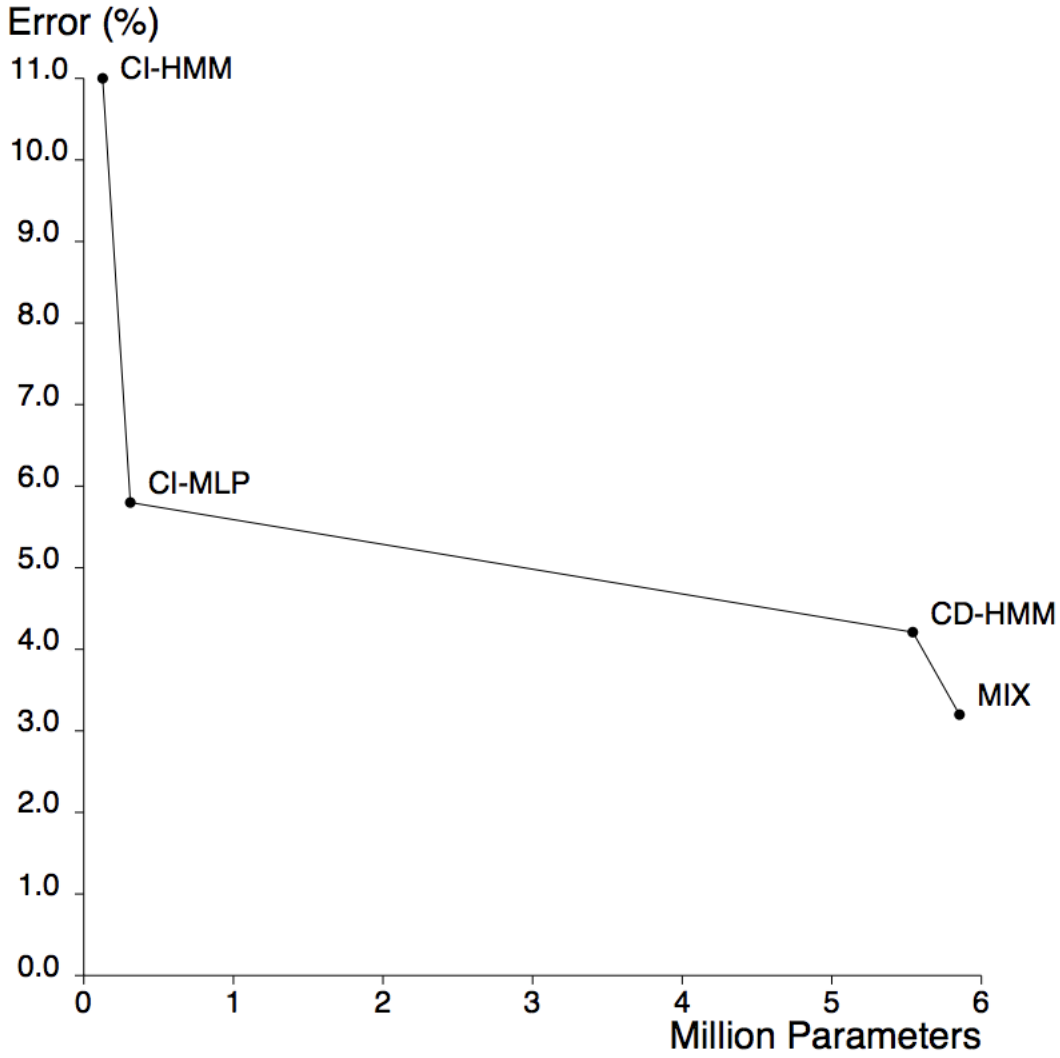


- **Deeper**: Deep neural network architecture – multiple hidden layers
- **Wider**: Use HMM state alignment as outputs rather than hand-labelled phones – 3-state HMMs, so 3×61 states
- Training many hidden layers is computationally expensive – use GPUs to provide the computational power

Hybrid NN/HMM



Monophone HMM/NN hybrid system (1993)



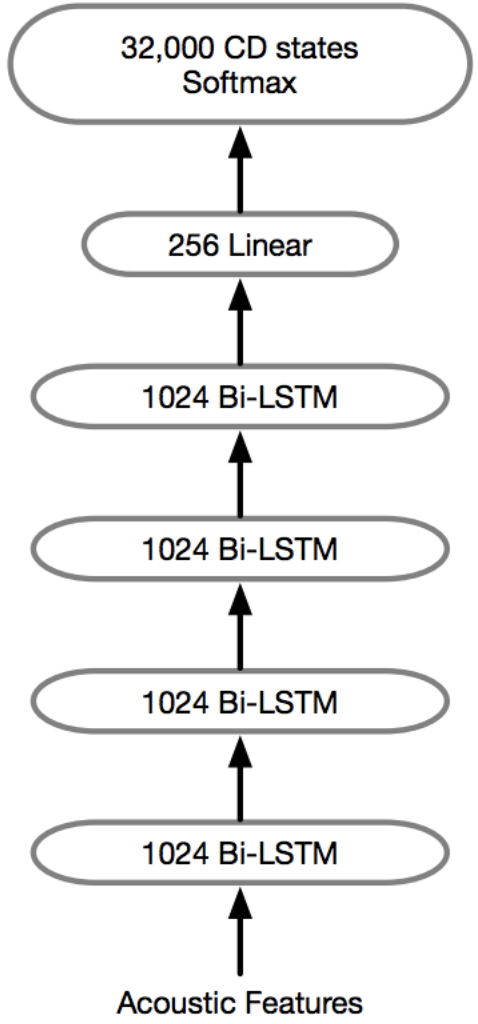
Renals, Morgan, Cohen & Franco, ICASSP 1992

HMM/NN vs HMM/GMM

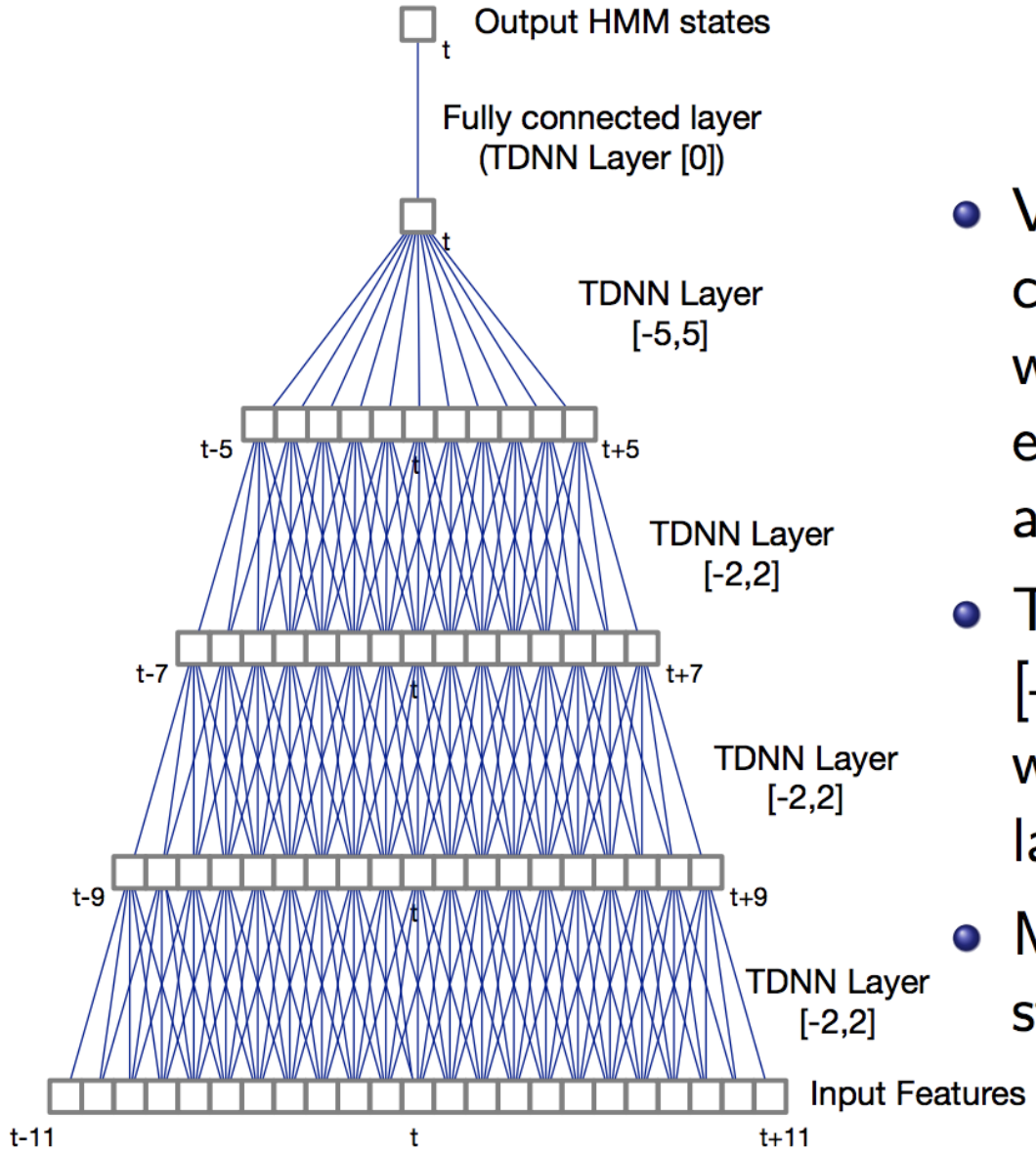
- Advantages of NN:
  - Can easily model **correlated features**
    - Correlated feature vector components (eg spectral features)
    - Input context – multiple frames of data at input
  - **More flexible** than GMMs – not made of (nearly) local components); GMMs inefficient for non-linear class boundaries
  - NNs can **model multiple events** in the input simultaneously – different sets of hidden units modelling each event; GMMs assume each frame generated by a single mixture component.
  - NNs can **learn richer representations** and learn 'higher-level' features (tandem, posteriorgrams, bottleneck features)

Example: Deep Bidirectional LSTM Acoustic Model (Switchboard)

- LSTM has 4-6 bidirectional layers with 1024 cells/layer (512 each direction)
- 256 unit linear bottleneck layer
- 32k context-dependent state outputs
- Input features
  - 40-dimension linearly transformed MFCCs (plus ivector)
  - 64-dimension log mel filter bank features (plus first and second derivatives)
  - concatenation of MFCC and FBANK features
- Training: 14 passes frame-level cross-entropy training, 1 pass sequence training (2 weeks on a K80 GPU)

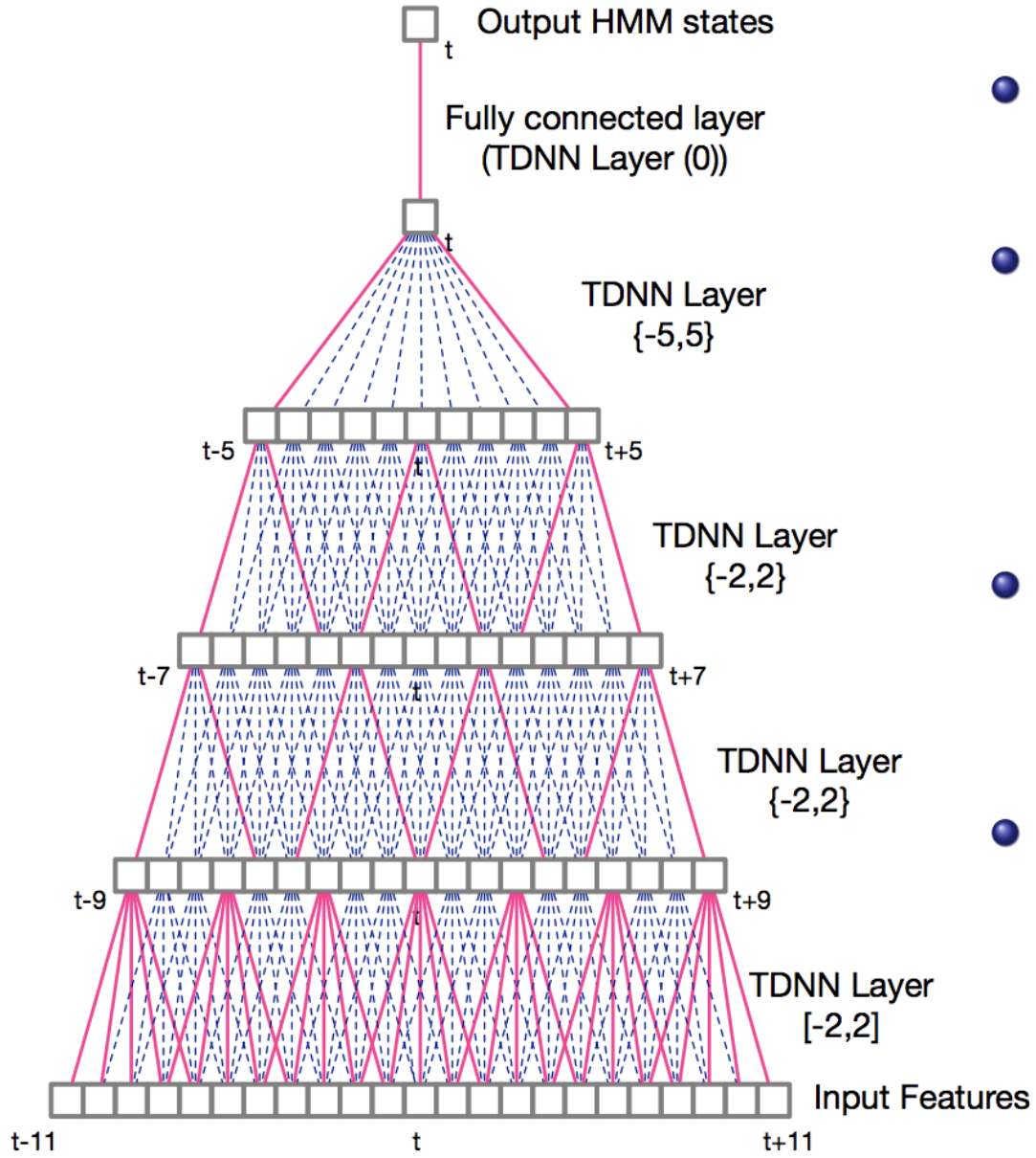


Example TDNN Architecture



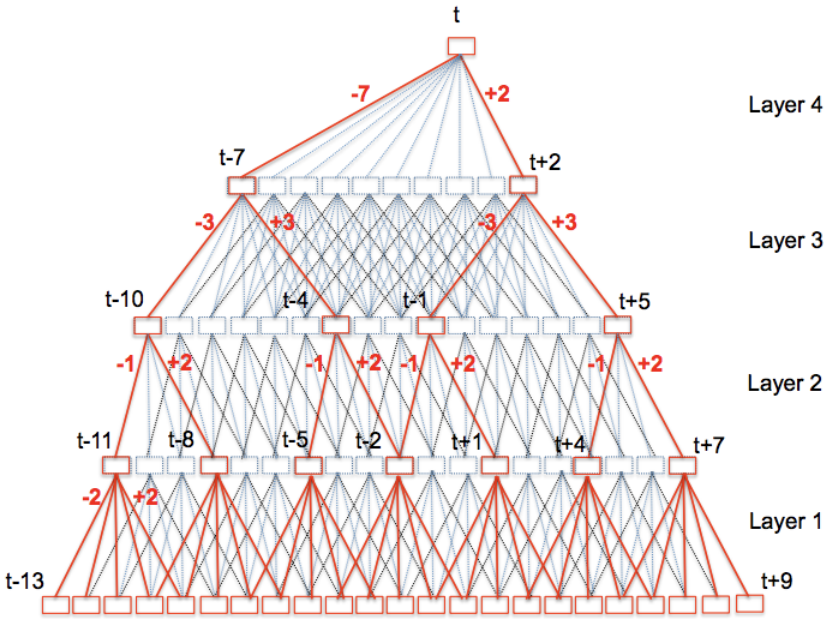
- View a TDNN as a 1D convolutional network with the transforms for each hidden unit tied across time
- TDNN layer with context [-2,2] has 5x as many weights as a regular DNN layer
- More computation, more storage required!

Sub-sampled TDNN



- Sub sample window of hidden unit activations
- Large overlaps between input contexts at adjacent time steps – likely to be correlated
- Allow gaps between frames in a window (cf. dilated convolutions)
- Sub-sampling saves computation and reduces number of model size (number of weights)

Example sub-sampled TDNN



Peddinti (2015)

Layer	Context	Sub-sampled Context
1	[-2,2]	[-2,2]
2	[-1,2]	{-1,2}
3	[-3,3]	{-3,3}
4	[-7,2]	{-7,2}
5	{0}	{0}

- Increase the context for higher layers of the network
- Subsampled so that difference between sampled hidden units is multiple of 3 to enable "clean" sub-sampling
- Asymmetric contexts
- MFCC features in this case

Switchboard Results

Network Architecture	Test Set WER/% Switchboard	CallHome
GMM (ML)	21.2	36.4
GMM (BMMI)	18.6	33.0
DNN (7x2048) / CE	14.2	25.7
DNN (7x2048) / MMI	12.9	24.6
TDNN (6x1024) / CE	12.5	
TDNN (6x576) / LF-MMI	9.2	17.3
LSTM (4x1024)	8.0	14.3
LSTM (6x1024)	7.7	14.0
LSTM-6 + feat fusion	7.2	12.7

GMM and DNN results - Vesely et al (2013); TDNN-CE results - Peddinti et al (2015); TDNN/LF-MMI results - Povey et al (2016); LSTM results - Saon et al (2017)

Combining models, and with multiple RNN language models, WER reduced to 5.5/10.3% (Saon et al, 2017)

博客推荐（KALDI）

- KALDI单音素GMM学习笔记
  - HTTPS://BLOG.CSDN.NET/U010731824/ARTICLE/DETAILS/69668765
- KALDI三音素GMM学习笔记
  - HTTPS://BLOG.CSDN.NET/U010731824/ARTICLE/DETAILS/70161677
- KALDI决策树状态绑定学习笔记
  - HTTPS://BLOG.CSDN.NET/U010731824/ARTICLE/DETAILS/69668647
  - HTTPS://BLOG.CSDN.NET/U010731824/ARTICLE/DETAILS/69667017
  - HTTPS://BLOG.CSDN.NET/U010731824/ARTICLE/DETAILS/69666398
  - HTTPS://BLOG.CSDN.NET/U010731824/ARTICLE/DETAILS/69666560
- 编译 & 命令行工具
  - HTTPS://WWW.TWBLOGS.NET/A/5B8B2E102B717718832DE330/ZH-CN

```
./path.sh

gmm_bin/gmm-copy --binary=false $mono_dir/0.mdl mono_mdl_0.txt
gmm_bin/gmm-copy --binary=false $mono_dir/final.mdl mono_mdl_final.txt
gmm_bin/gmm-copy --binary=false $tri_dir/final.mdl tri_mdl_final.txt

gmm_bin/gmm-info $mono_dir/final.mdl
gmm_bin/gmm-info $tri_dir/final.mdl


bin/copy-tree --binary=false $mono_dir/tree - | less
bin/draw-tree $mono_dir/phones.txt $mono_dir/tree | dot -Gsize=8,10.5 -Tps | ps2pdf - tree.pdf

bin/copy-int-vector "ark:gunzip -c $mono_dir/ali.1.gz|" ark,t:- | head -n 2

bin/show-transitions $lan_dir/phones.txt $mono_dir/0.mdl
bin/show-alignments $lan_dir/phones.txt $mono_dir/0.mdl "ark:gunzip -c $mono_dir/ali.1.gz |" | less
```

THANKS