4.2 Acoustic Model

- 4.2.1 Hidden Markov Model (overview)
- 4.2.2 GMM-HMM
- 4.2.3 DNN-HMM

Parameters of an HMM

- States: A set of states $S=s_1,\ldots,s_n$
- Transition probabilities: $A= a_{1,1},a_{1,2},\ldots,a_{n,n}$ Each $a_{i,j}$ represents the probability of transitioning from state s_i to s_j.
- Emission probabilities (output probabilities): A set B of functions of the form $b_i(o_t)$ which is the probability of observation o_t being emitted by s_i
- Initial state distribution: π_i is the probability that s_i is a start state

The Three Basic HMM Problems

- Problem 1 (Evaluation): Given the observation sequence $O=o_1,\ldots,o_T$ and an HMM model $\lambda=(A,B,\pi)$, how do we compute the probability of O given the model?

- Problem 2 (Decoding): Given the observation sequence $O=o_1,\ldots,o_T$ and an HMM model $\lambda=(A,B,\pi)$, how do we find the state sequence that best explains the observations?

The Three Basic HMM Problems

- Problem 3 (Learning): How do we adjust the model parameters $\lambda = (A, B, \pi)$, to maximize $P(O \mid \lambda)$?

Problem 1 (Evaluation): Probability of an Observation Sequence

- What is $P(O \mid \lambda)$?
- The probability of an observation sequence is the sum of the probabilities of all possible state sequences in the HMM.
- Naïve computation is very expensive. Given T observations and N states, there are N^T possible state sequences.
- Even small HMMs, e.g. T=10 and N=10, contain 10 billion different paths
- Solution to this and problem 2 is to use dynamic programming

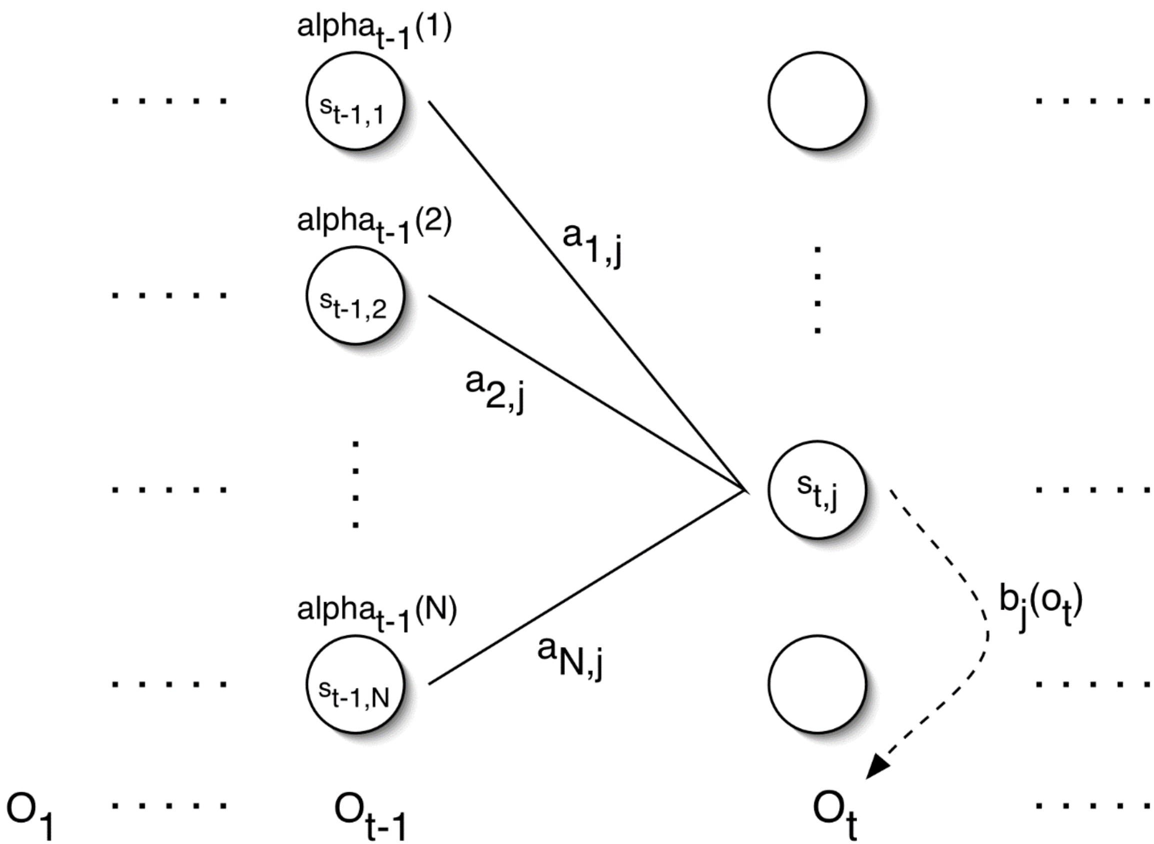
Forward Probabilities

- What is the probability that, given an HMM λ , at time t the state is i and the partial observation $o_1 \ldots o_t$ has been generated?

$$\alpha_t(i) = P(o_1 \ldots o_t, q_t = s_i \mid \lambda)$$

Forward Probabilities

$$\alpha_t(i) = P(o_1 \ldots o_t, q_t = s_i \mid \lambda)$$



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

Forward Algorithm

- Initialization:

$$\alpha_1(i) = \pi_i b_i(o_1) \quad 1 \le i \le N$$

- Induction:

$$\alpha_t(j) = \left[\sum_{i=1}^{N} \alpha_{t-1}(i)\, a_{ij} \right] b_j(o_t) \quad 2 \le t \le T, 1 \le j \le N$$

- Termination:

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

Forward Algorithm Complexity

- In the naïve approach to solving problem 1 it takes on the order of $2T*N^T$ computations
- The forward algorithm takes on the order of N^2T computations

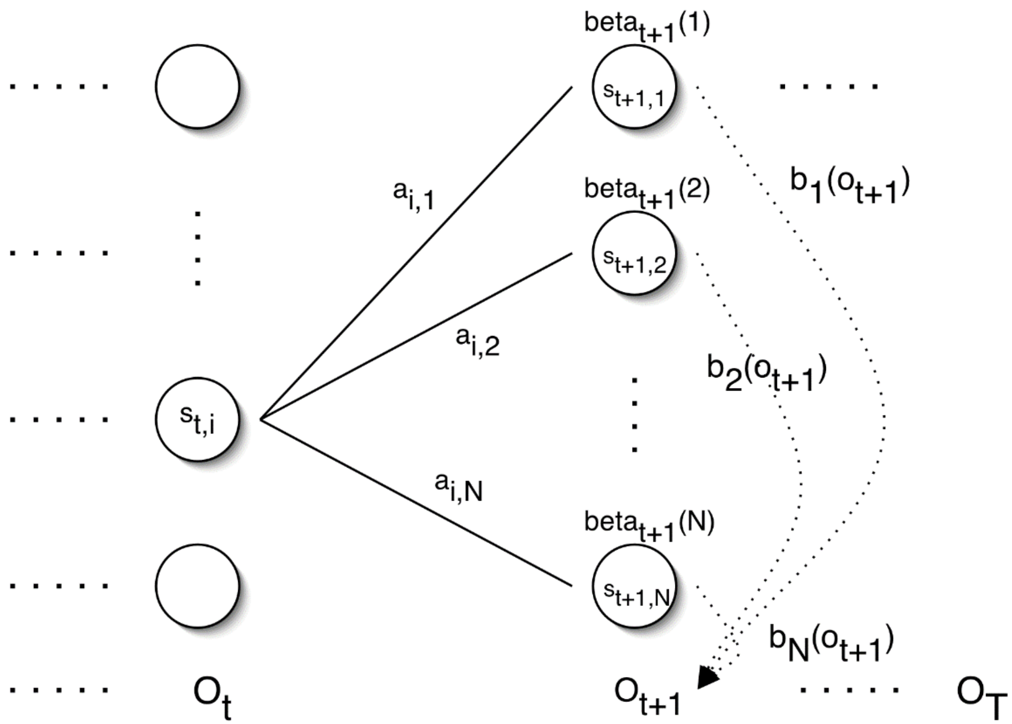
Backward Probabilities

- Analogous to the forward probability, just in the other direction
- What is the probability that given an HMM λ and given the state at time t is i, the partial observation $o_{t+1} \dots o_T$ is generated?

$$\beta_t(i) = P(o_{t+1} \dots o_T \mid q_t = s_i, \lambda)$$

Backward Probabilities

$$\beta_t(i) = P(o_{t+1} \ldots o_T \mid q_t = s_i, \lambda)$$



$$\beta_t(i) = \left[\sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) \right]$$

Backward Algorithm

- Initialization:

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

- Induction:

$$\beta_t(i) = \left[\sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) \right] \quad t = T-1 \ldots 1, 1 \le i \le N$$

- Termination:

$$P(O \mid \lambda) = \sum_{i=1}^{N} \pi_i \beta_1(i)$$

Problem 2 (Decoding)

- The solution to Problem 1 (Evaluation) gives us the sum of all paths through an HMM efficiently.
- For Problem 2, we want to find the path with the highest probability.
- We want to find the state sequence $Q=q_1 \ldots q_T$, such that

$$Q = \underset{Q'}{\arg\max}\, P(Q' \mid O, \lambda)$$

Viterbi Algorithm

- Similar to computing the forward probabilities, but instead of summing over transitions from incoming states, compute the maximum
- Forward: $$\alpha_t(j) = \left[\sum_{i=1}^{N} \alpha_{t-1}(i)\, a_{ij}\right] b_j(o_t)$$
- Viterbi Recursion: $$\delta_t(j) = \left[\max_{1 \le i \le N} \delta_{t-1}(i)\, a_{ij}\right] b_j(o_t)$$

Viterbi Algorithm

- Initialization: $$\delta_1(i) = \pi_i b_j(o_1) \quad 1 \le i \le N$$
- Induction: $$\delta_t(j) = \left[\max_{1\le i\le N} \delta_{t-1}(i)\, a_{ij}\right] b_j(o_t)$$

$$\psi_t(j) = \left\lfloor \arg\max_{1\le i\le N} \delta_{t-1}(i)\, a_{ij} \right\rfloor \quad 2 \le t \le T, 1 \le j \le N$$

- Termination: $$p^* = \max_{1\le i\le N} \delta_T(i) \qquad q_T^* = \arg\max_{1\le i\le N} \delta_T(i)$$
- Read out path: $$q_t^* = \psi_{t+1}(q_{t+1}^*) \quad t = T-1, \ldots, 1$$

Problem 3 (Learning)

- Up to now we've assumed that we know the underlying model $\lambda = (A, B, \pi)$
- Often these parameters are estimated on annotated training data, which has two drawbacks:
  - Annotation is difficult and/or expensive
  - Training data is different from the current data
- We want to maximize the parameters with respect to the current data, i.e., we're looking for a model λ', such that

$$\lambda' = \underset{\lambda}{\arg\max}\, P(O \mid \lambda)$$

Problem 3 (Learning)

- Unfortunately, there is no known way to analytically find a global maximum, i.e., a model λ', such that

$$\lambda' = \arg\max_{\lambda} P(O \mid \lambda)$$

- But it is possible to find a local maximum

- Given an initial model λ, we can always find a model λ', such that

$$P(O \mid \lambda') \geq P(O \mid \lambda)$$

Parameter Re-estimation

- Use the forward-backward (or Baum-Welch) algorithm, which is a hill-climbing algorithm

- Using an initial parameter instantiation, the forward-backward algorithm iteratively re-estimates the parameters and improves the probability that given observation are generated by the new parameters

Expectation Maximization

- The forward-backward (Baum-Welch) algorithm is an instance of the more general EM algorithm
  - The E Step: Compute the forward and backward probabilities for a give model
  - The M Step: Re-estimate the model parameters

Parameter Re-estimation

- Three parameters need to be re-estimated:
  - Initial state distribution: π_i
  - Transition probabilities: $a_{i,j}$
  - Emission probabilities: $b_i(o_t)$

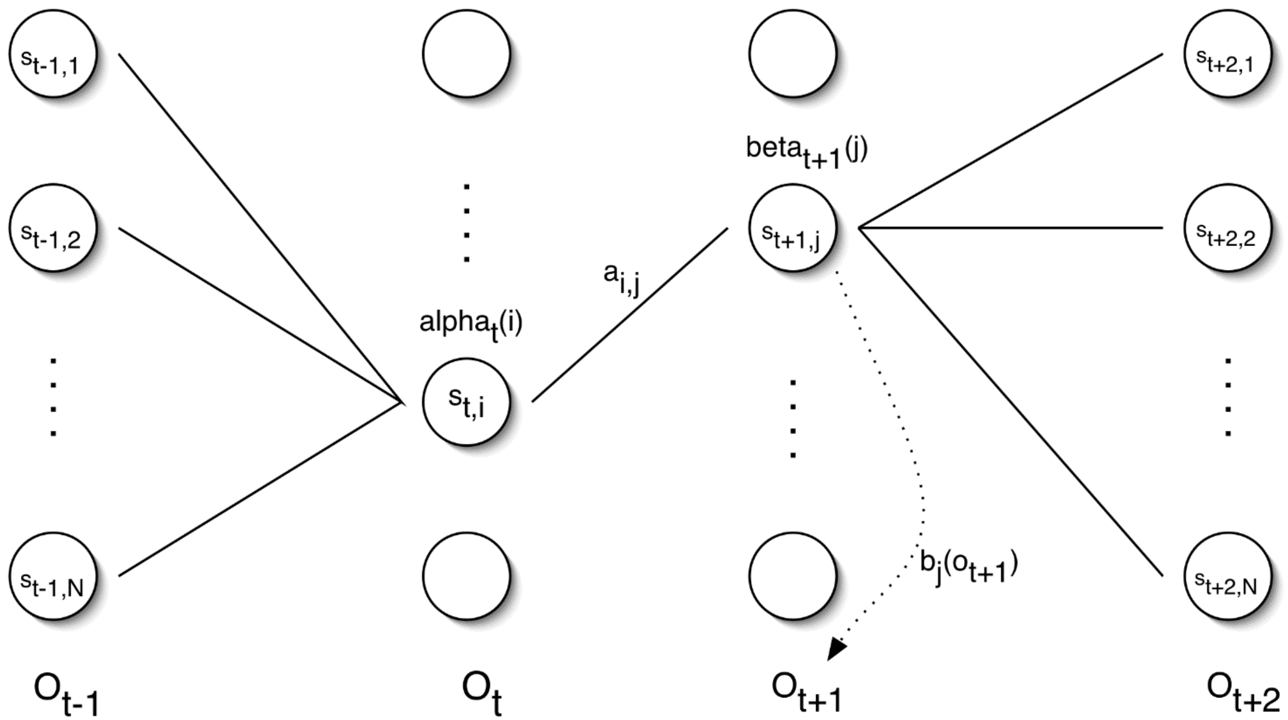
Re-estimating Transition Probabilities

- What's the probability of being in state s_i at time t and going to state s_j, given the current model and parameters?

$$\xi_t(i, j) = P(q_t = s_i, q_{t+1} = s_j \mid O, \lambda)$$

Re-estimating Transition Probabilities

$$\xi_t(i,j) = P(q_t = s_i, q_{t+1} = s_j \mid O, \lambda)$$



$$\xi_t(i,j) = \frac{\alpha_t(i)\, a_{i,j}\, b_j(o_{t+1})\, \beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_t(i)\, a_{i,j}\, b_j(o_{t+1})\, \beta_{t+1}(j)}$$

Re-estimating Transition Probabilities

- The intuition behind the re-estimation equation for transition probabilities is

$$\hat{a}_{i,j} = \frac{\text{expected number of transitions from state } \mathrm{s}_\mathrm{i} \text{ to state } \mathrm{s}_\mathrm{j}}{\text{expected number of transitions from state } \mathrm{s}_\mathrm{i}}$$

- Formally:

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

Re-estimating Transition Probabilities

- Defining $$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j)$$

  as the probability of being in state s_i, given the complete observation O

- We can say:

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

Review of Probabilities

- **Forward probability:** $\alpha_t(i)$

  The probability of being in state s_i, given the partial observation $o_1,\ldots,o_t$

- **Backward probability:** $\beta_t(i)$

  The probability of being in state s_i, given the partial observation $o_{t+1},\ldots,o_T$

- **Transition probability:** $\xi_t(i,j)$

  The probability of going from state s_i, to state s_j, given the complete observation $o_1,\ldots,o_T$

- **State probability:** $\gamma_t(i)$

  The probability of being in state s_i, given the complete observation $o_1,\ldots,o_T$

Re-estimating Initial State Probabilities

- Initial state distribution: π_i is the probability that s_i is a start state

- Re-estimation is easy:

$$\hat{\pi}_i = \text{expected number of times in state } s_i \text{ at time } 1$$

- Formally:

$$\hat{\pi}_i = \gamma_1(i)$$

Re-estimation of Output Probabilities

- Emission probabilities are re-estimated as

$$\hat{b}_i(k) = \frac{\text{expected number of times in state } \mathrm{s_i} \text{ and observe symbol } \mathrm{v_k}}{\text{expected number of times in state } \mathrm{s_i}}$$

- Formally:

$$\hat{b}_i(k) = \frac{\sum_{t=1}^{T} \delta(o_t, v_k)\gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)}$$

where $\delta(o_t, v_k) = 1$, if $o_t = v_k$, and 0 otherwise

Note that δ here is the Kronecker delta function and is not related to the δ in the discussion of the Viterbi algorithm!!

The Updated Model

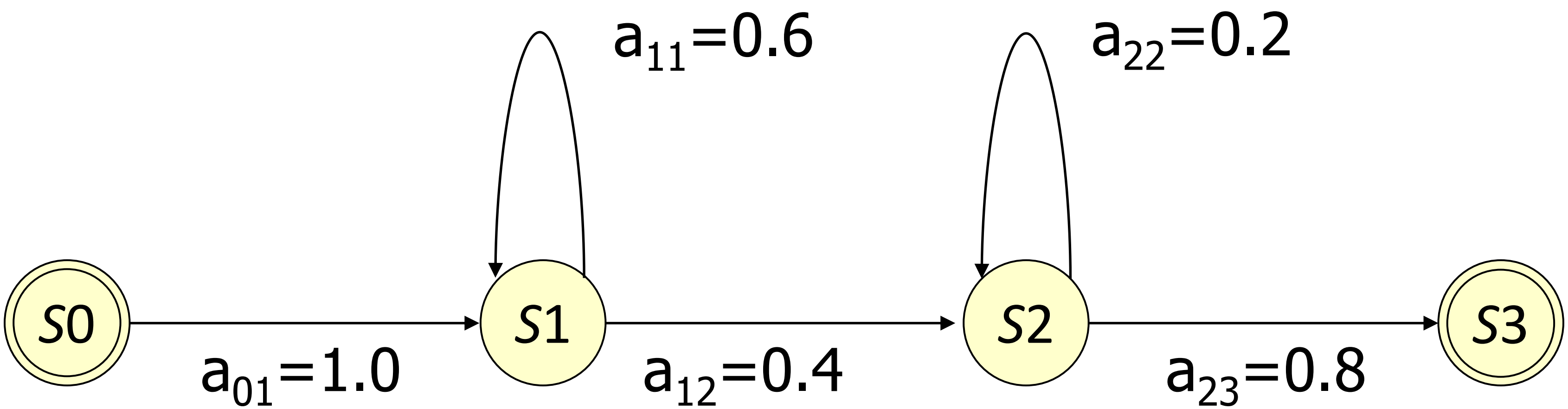
- Coming from $\lambda = (A, B, \pi)$, we get to $\hat{\lambda} = (\hat{A}, \hat{B}, \hat{\pi})$ by the following update rules:

$$\hat{a}_{i,j} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \qquad \hat{b}_i(k) = \frac{\sum_{t=1}^{T} \delta(o_t, v_k)\gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)} \qquad \hat{\pi}_i = \gamma_1(i)$$

4.2 Acoustic Model

- 4.2.1 Hidden Markov Model
- 4.2.2 GMM-HMM
- 4.2.3 DNN-HMM

How to model the output probability distribution of HMM?



$b_1(v_1) = 0.7$ $\quad$ $b_2(v_1) = 0.6$

$b_1(v_2) = 0.3$ $\quad$ $b_2(v_2) = 0.4$

Discrete Distribution

Left-to-right HMM

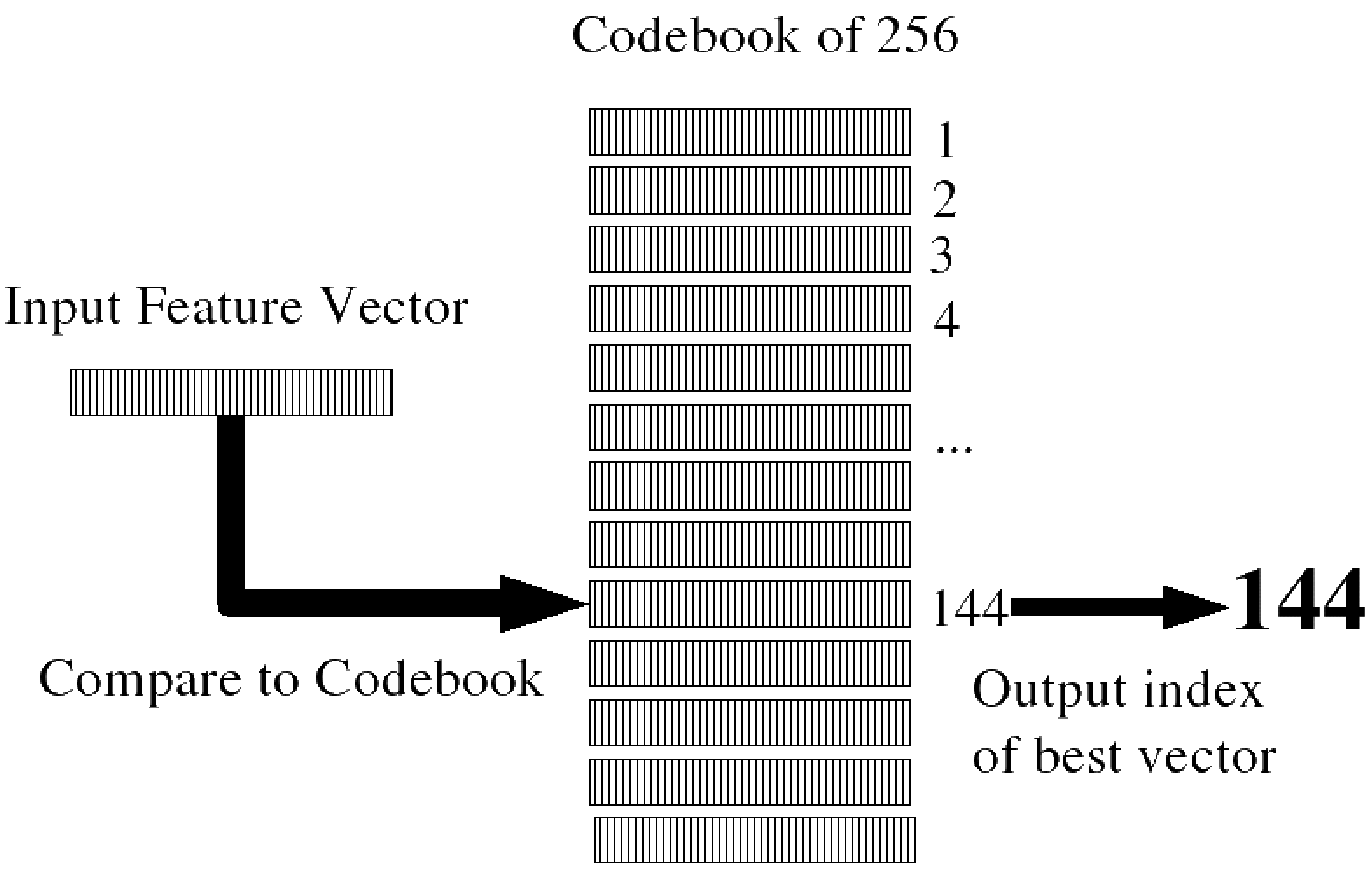
Vector Quantization

- Create a training set of feature vectors
- Cluster them into a small number of classes
- Represent each class by a discrete symbol
- For each class v_k, we can compute the probability that it is generated by a given HMM state using Baum-Welch as above

Vector Quantization

- We'll define a
  - Codebook, which lists for each symbol
  - A prototype vector, or codeword
- If we had 256 classes ('8-bit VQ'),
  - A codebook with 256 prototype vectors
  - Given an incoming feature vector, we compare it to each of the 256 prototype vectors
  - We pick whichever one is closest by some 'distance metric'
  - And replace the input vector by the index of this prototype vector

Vector Quantization



VQ requirements

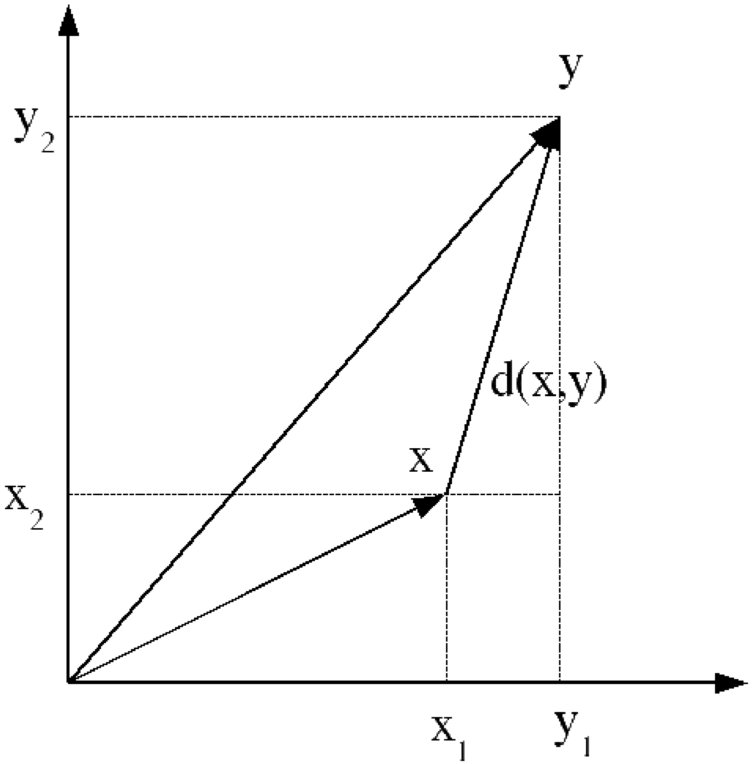
- A distance metric or distortion metric
  - Specifies how similar two vectors are
  - Used:
    - to build clusters
    - To find prototype vector for cluster
    - And to compare incoming vector to prototypes
- A clustering algorithm
  - K-means, etc.

Distance metrics

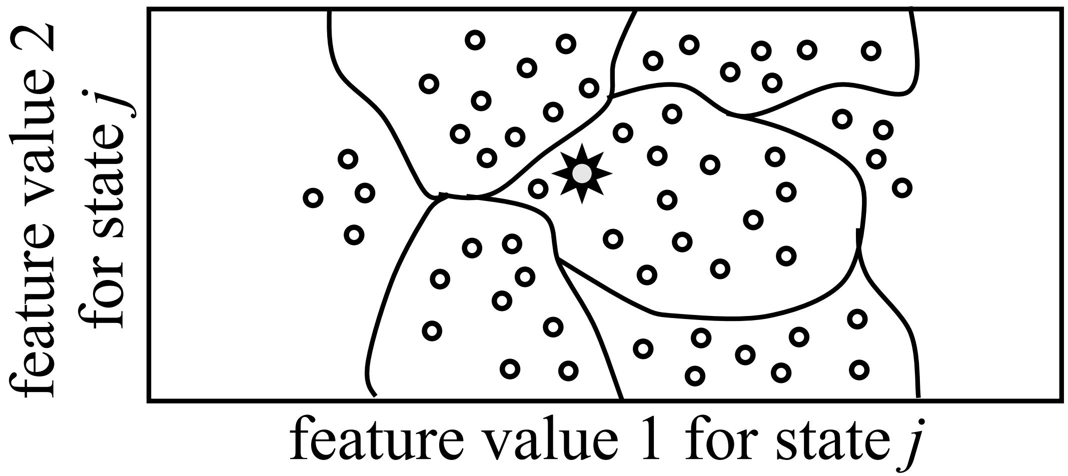
- Simplest:
  - (square of) Euclidean distance

$$d^2(x,y) = \sum_{i=1}^{D} (x_i - y_i)^2$$

  - Also called ‘sum-squared error’



Computing $b_j(v_k)$



- $b_j(v_k) = \dfrac{\text{number of vectors with codebook index } k \text{ in state } j}{\text{number of vectors in state } j} = \dfrac{14}{56} = \dfrac{1}{4}$

Problem: how to apply HMM to continuous observations?

- We have assumed that the output alphabet V has a finite number of symbols
- But spectral feature vectors are real-valued!
- How to deal with real-valued features?
  - Decoding: Given o_t, how to compute $P(o_t|q)$
  - Learning: How to modify EM to deal with real-valued features

Directly Modeling Continuous Observations

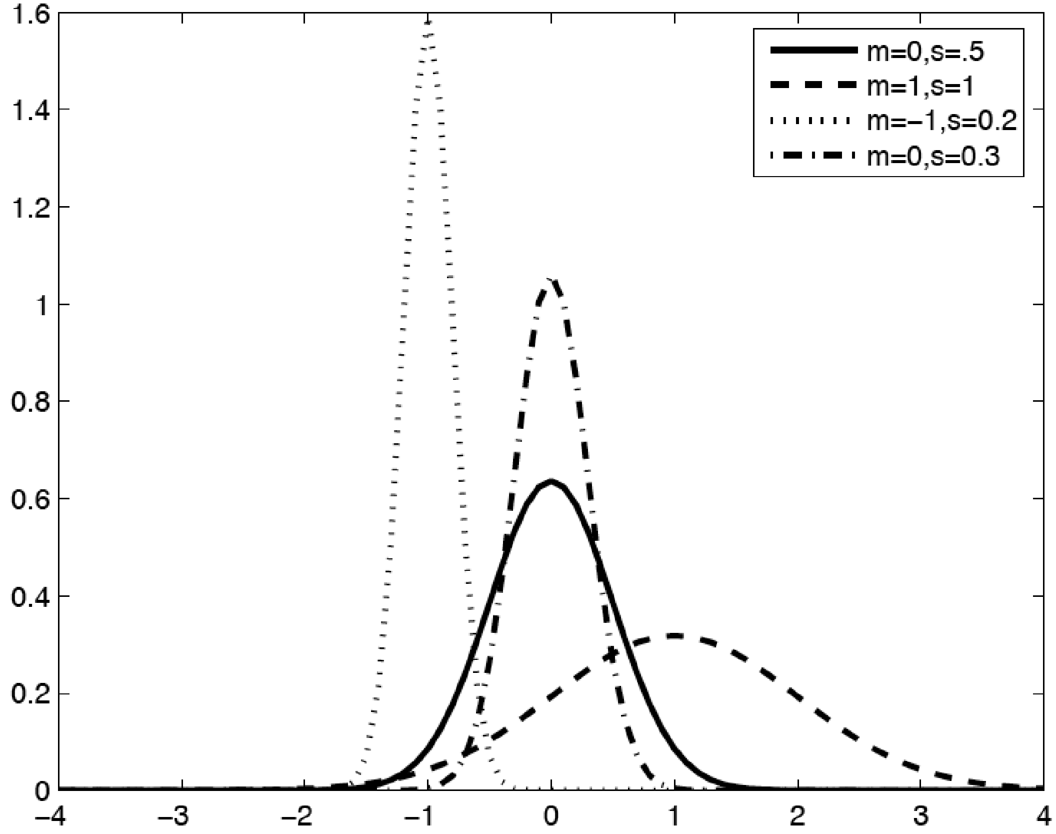
- Assume the possible values of the observation feature vector o_t are normally distributed.
- Represent the observation likelihood function $b_j(o_t)$ as a Gaussian with mean μ_j and variance σ_j^2

$$f(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

- Gaussians
  - Univariate Gaussians
    - Baum-Welch for univariate Gaussians
  - Multivariate Gaussians
    - Baum-Welch for multivariate Gaussians
  - Gaussian Mixture Models (GMMs)
    - Baum-Welch for GMMs

Gaussians with mean and variance

$$f(x|m,s) = \frac{1}{s\sqrt{2\pi}} \exp(-\frac{(x-m)^2}{2s^2})$$



Review: means and variances

- For a discrete random variable X
- Mean is the expected value of X
  - Weighted sum over the values of X

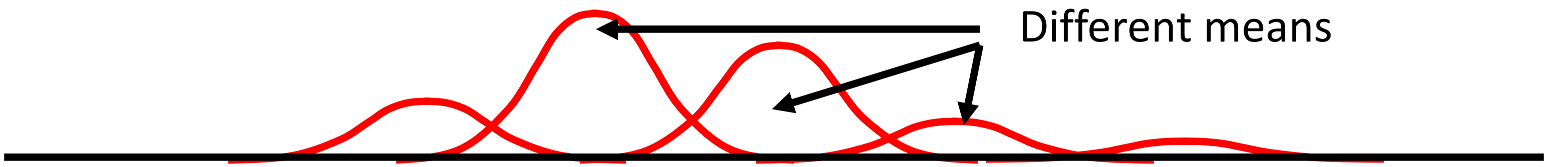
$$\mu = E(X) = \sum_{i=1}^{N} p(X_i) X_i$$

- Variance is the squared average deviation from mean

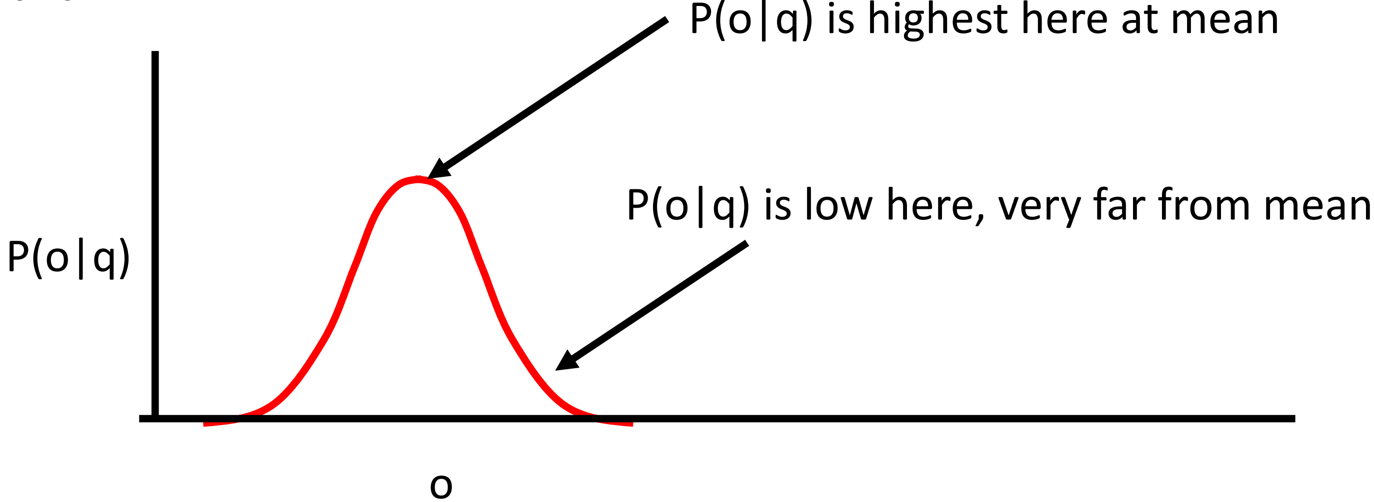
$$\sigma^2 = \sum_{i=1}^{N} p(X_i)(X_i - E(X))^2$$

Gaussians for Acoustic Modeling

A Gaussian is parameterized by a mean and a variance:



- P(o|q):



Using a (univariate Gaussian) as an acoustic likelihood estimator

- Let's suppose our observation was a single real-valued feature (instead of 39D vector)
- Then if we had learned a Gaussian over the distribution of values of this feature
- We could compute the likelihood of any given observation o_t as follows:

$$b_j(o_t) = \frac{1}{\sqrt{2\pi\sigma_j^2}} exp\left(-\frac{(o_t - \mu_j)^2}{2\sigma_j^2}\right)$$

Training a Univariate Gaussian

- A (single) Gaussian is characterized by a mean and a variance
- Imagine that we had some training data in which each state was labeled
- We could just compute the mean and variance from the data:

$$\mu_i = \frac{1}{T}\sum_{t=1}^{T} o_t \quad s.t. \; o_t \;\; is \;\; state \;\; i$$

$$\sigma_i^{\,2} = \frac{1}{T}\sum_{t=1}^{T} (o_t - \mu_i)^2 \quad s.t. \; o_t \;\; is \;\; state \;\; i$$

Training Univariate Gaussians

- But we don't know which observation was produced by which state!
- What we want: to assign each observation vector o_t to every possible state *i*, prorated by the probability the HMM was in state *i* at time t.
- The probability of being in state i at time t is $\xi_t(i)$!!

$$\bar{\mu}_i = \frac{\sum_{t=1}^{T} \xi_t(i) o_t}{\sum_{t=1}^{T} \xi_t(i)} \qquad \bar{\sigma}^2{}_i = \frac{\sum_{t=1}^{T} \xi_t(i)(o_t - \mu_i)^2}{\sum_{t=1}^{T} \xi_t(i)}$$

Multivariate Gaussians

- Instead of a single mean μ and variance σ:

$$f(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

- Vector of means μ and covariance matrix Σ

$$f(x \mid \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} \mid \Sigma \mid^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

- Defining μ and Σ: $\mu = E(x)$ $\Sigma = E\left[(x-\mu)(x-\mu)^T\right]$

- So the i-jth element of Σ is: $\sigma_{ij}^2 = E\left[(x_i-\mu_i)(x_j-\mu_j)\right]$

But: assume diagonal covariance

- Assume that the features in the feature vector are uncorrelated. This isn't true for FFT features, but is almost true for MFCC features.
- Computation and storage much cheaper if diagonal covariance, i.e. only diagonal entries are non-zero.
- Diagonal contains the variance of each dimension σ_{ii}^2
- So this means we consider the variance of each acoustic feature (dimension) separately.

Diagonal covariance

- Diagonal contains the variance of each dimension σ_{ii}^2
- So this means we consider the variance of each acoustic feature (dimension) separately

$$f(x \mid \mu, \sigma) = \prod_{d=1}^{D} \frac{1}{\sigma_d \sqrt{2\pi}} \exp\left(-\frac{1}{2} \left(\frac{x_d - \mu_d}{\sigma_d} \right)^2 \right)$$

$$f(x \mid \mu, \sigma) = \frac{1}{(2\pi)^{D/2} \prod_{d=1}^{D} \sigma_d} \exp\left(-\frac{1}{2} \sum_{d=1}^{D} \frac{(x_d - \mu_d)^2}{\sigma_d^{\,2}}\right)$$

Baum-Welch reestimation equations for multivariate Gaussians

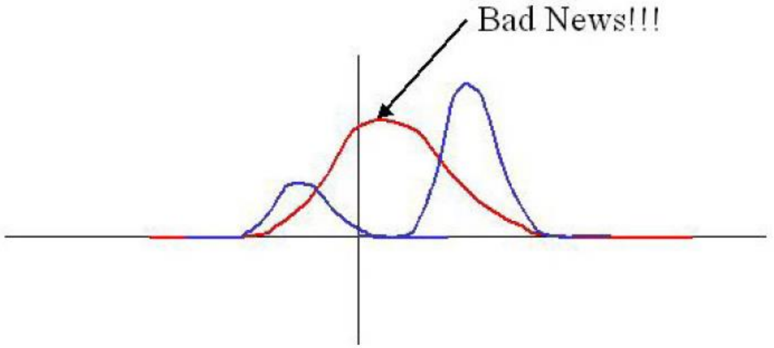
- Natural extension of univariate case, where now μ_i is mean vector for state i:

$$\bar{\mu}_i = \frac{\sum_{t=1}^{T} \xi_t(i) o_t}{\sum_{t=1}^{T} \xi_t(i)}$$

$$\bar{\Sigma}_i = \frac{\sum_{t=1}^{T} \xi_t(i)(o_t - \mu_i)(o_t - \mu_i)^T}{\sum_{t=1}^{T} \xi_t(i)}$$

A bad modeling of distribution

- Single Gaussian may do a bad job of modeling distribution in any dimension:



- Solution: Mixtures of Gaussians

Mixtures of Gaussians

- M mixtures of Gaussians:

$$f(x \mid \mu_{jk}, \Sigma_{jk}) = \sum_{k=1}^{M} c_{jk} N(x, \mu_{jk}, \Sigma_{jk})$$

$$b_j(o_t) = \sum_{k=1}^{M} c_{jk} N(o_t, \mu_{jk}, \Sigma_{jk})$$

- For diagonal covariance:

$$b_j(o_t) = \sum_{k=1}^{M} \frac{c_{jk}}{2\pi^{D/2} \prod_{d=1}^{D} \sigma_{jkd}^{\ 2}} \exp\left(-\frac{1}{2} \sum_{d=1}^{D} \frac{(x_{jkd} - \mu_{jkd})^2}{\sigma_{jkd}^{\ 2}}\right)$$

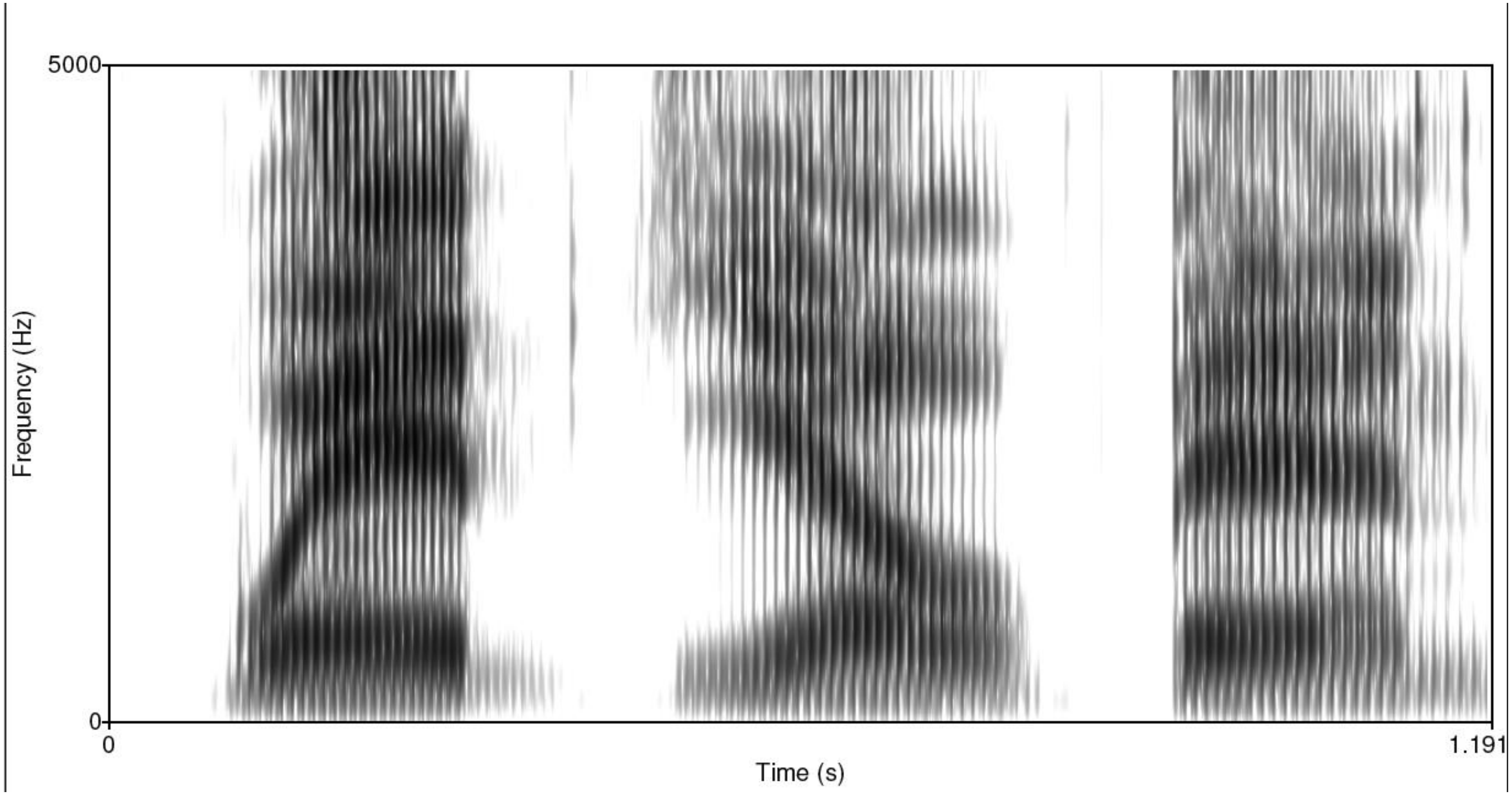
Summary: GMMs

- Each state has a likelihood function parameterized by:
  - M Mixture weights
  - M Mean Vectors of dimensionality D
  - Either
    - M Covariance Matrices of DxD
  - Or more likely
    - M Diagonal Covariance Matrices of DxD

    which is equivalent to
    - M Variance Vectors of dimensionality D

Modeling phonetic context: different "eh"s

- w eh d y eh l b eh n

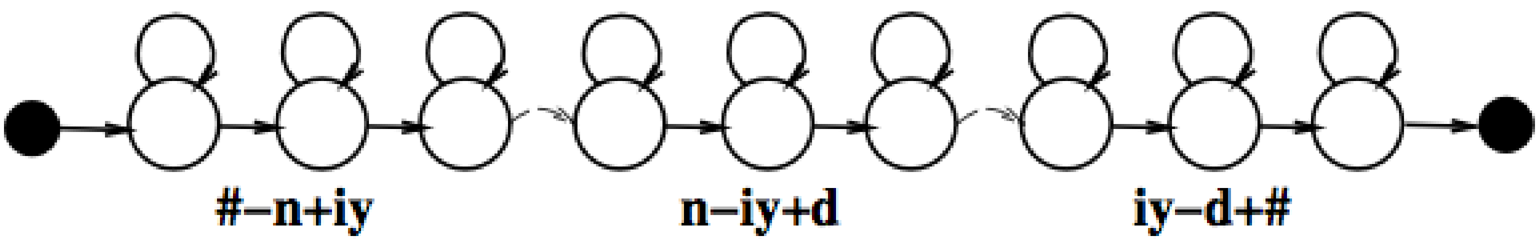


Modeling phonetic context

- The strongest factor affecting phonetic variability is the neighboring phone
- How to model that in HMMs?
- Idea: have phone models which are specific to context.
- Instead of Context-Independent (CI) phones
- We'll have Context-Dependent (CD) phones

CD phones: triphones

- Triphones
- Each triphone captures facts about preceding and following phone
- Monophone:
  - p, t, k
- Triphone:
  - iy-p+aa
  - a-b+c means "phone b, preceding by phone a, followed by phone c"
- "Need" with triphone models



Word-Boundary Modeling

- Word-Internal Context-Dependent Models

  'OUR LIST':

  SIL AA+R AA-R L+IH L-IH+S IH-S+T S-T

- Cross-Word Context-Dependent Models

  'OUR LIST':

  SIL-AA+R AA-R**+L** **R-**L+IH L-IH+S IH-S+T S-T**+SIL**

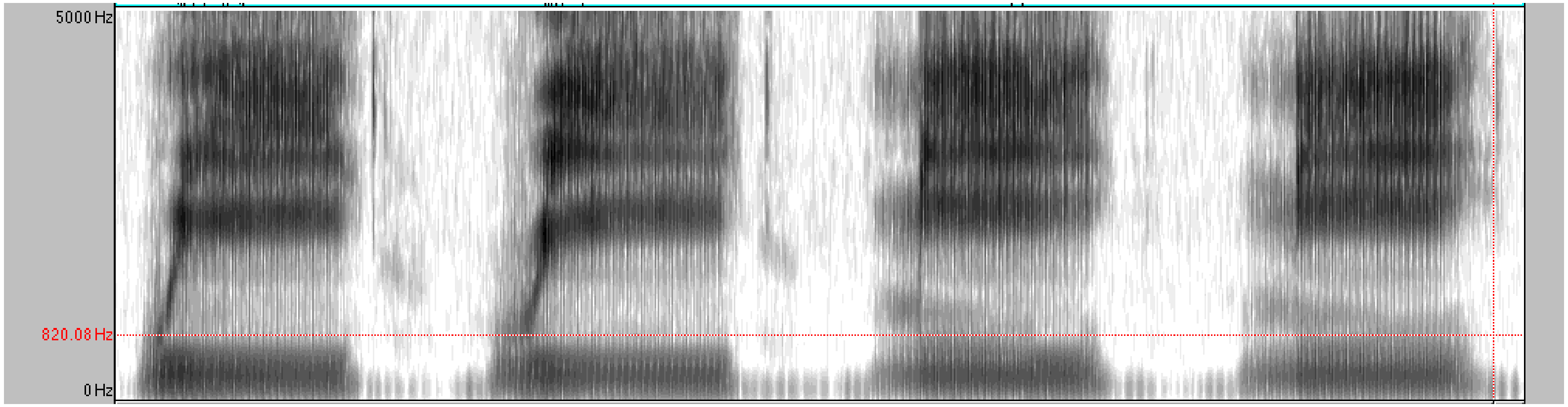
- Dealing with cross-words makes decoding harder! We will return to this.

Implications of Cross-Word Triphones

- Possible triphones: 50x50x50=125,000
- How many triphone types actually occur?
- 20K word WSJ Task, numbers from Young et al
- Cross-word models: need 55,000 triphones
- But in training data only 18,500 triphones occur!
- Need to generalize models.

Modeling phonetic context

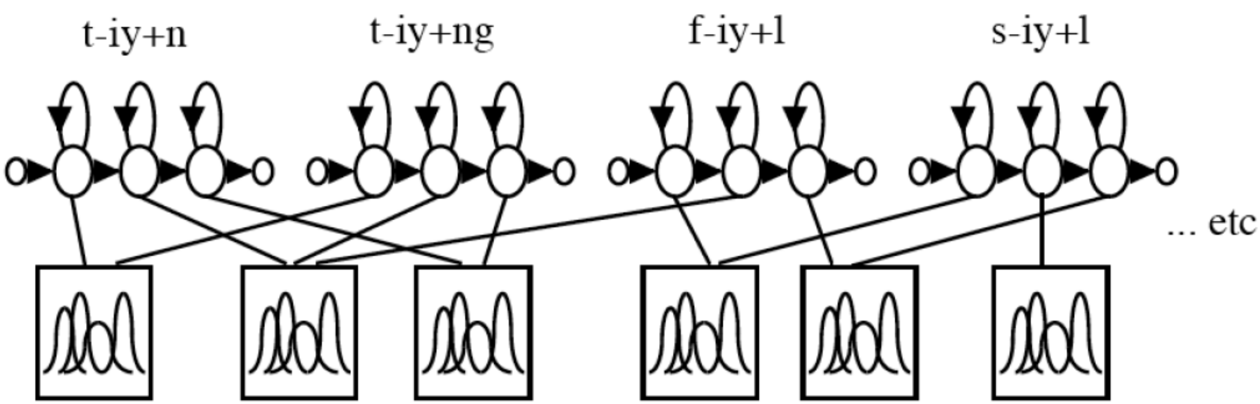
some contexts look similar



w iy r iy m iy n iy

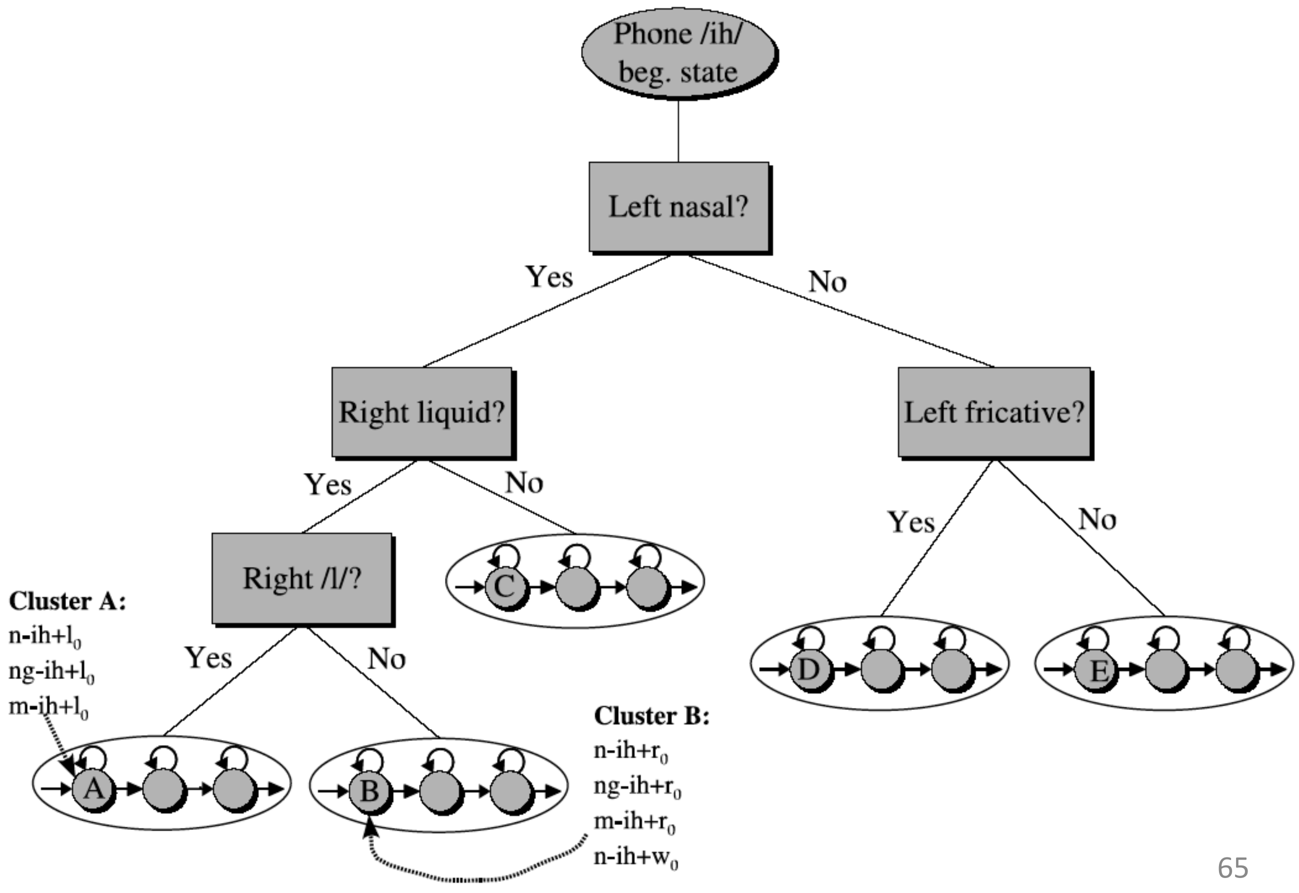
Solution: State Tying (Young, Odell, Woodland, 1994)

- Decision-Tree based clustering of triphone states
- States which are clustered together will share their Gaussians
- We call this "state tying", since these states are "tied together" to the same Gaussian.
- Previous work: generalized triphones
  - Model-based clustering ('model' = 'phone')
  - Clustering at state is more fine-grained



Decision tree for clustering triphones for tying

- How do we decide which triphones to cluster together?
- Use <u>phonetic features</u> (or 'broad phonetic classes')
  - Stop
  - Nasal
  - Fricative
  - Sibilant
  - Vowel
  - lateral



Decision tree for clustering triphones for tying

Feature	Phones
Stop	b d g k p t
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 ey ih ix iy ow oy uh uw
Front Vowel	ae eh ih ix iy
Central Vowel	aa ah ao axr er
Back Vowel	ax ow uh uw
High Vowel	ih ix iy uh uw
Rounded	ao ow oy uh uw w
Reduced	ax axr ix
Unvoiced	ch f hh k p s sh t th
Coronal	ch d dh jh l n r s sh t th z zh

State Tying:
Young, Odell, Woodland 1994

- The steps in creating CD phones.
- Start with monophone, do EM training
- Then clone Gaussians into triphones
- Then build decision tree and cluster Gaussians
- Then clone and train mixtures (GMMs)

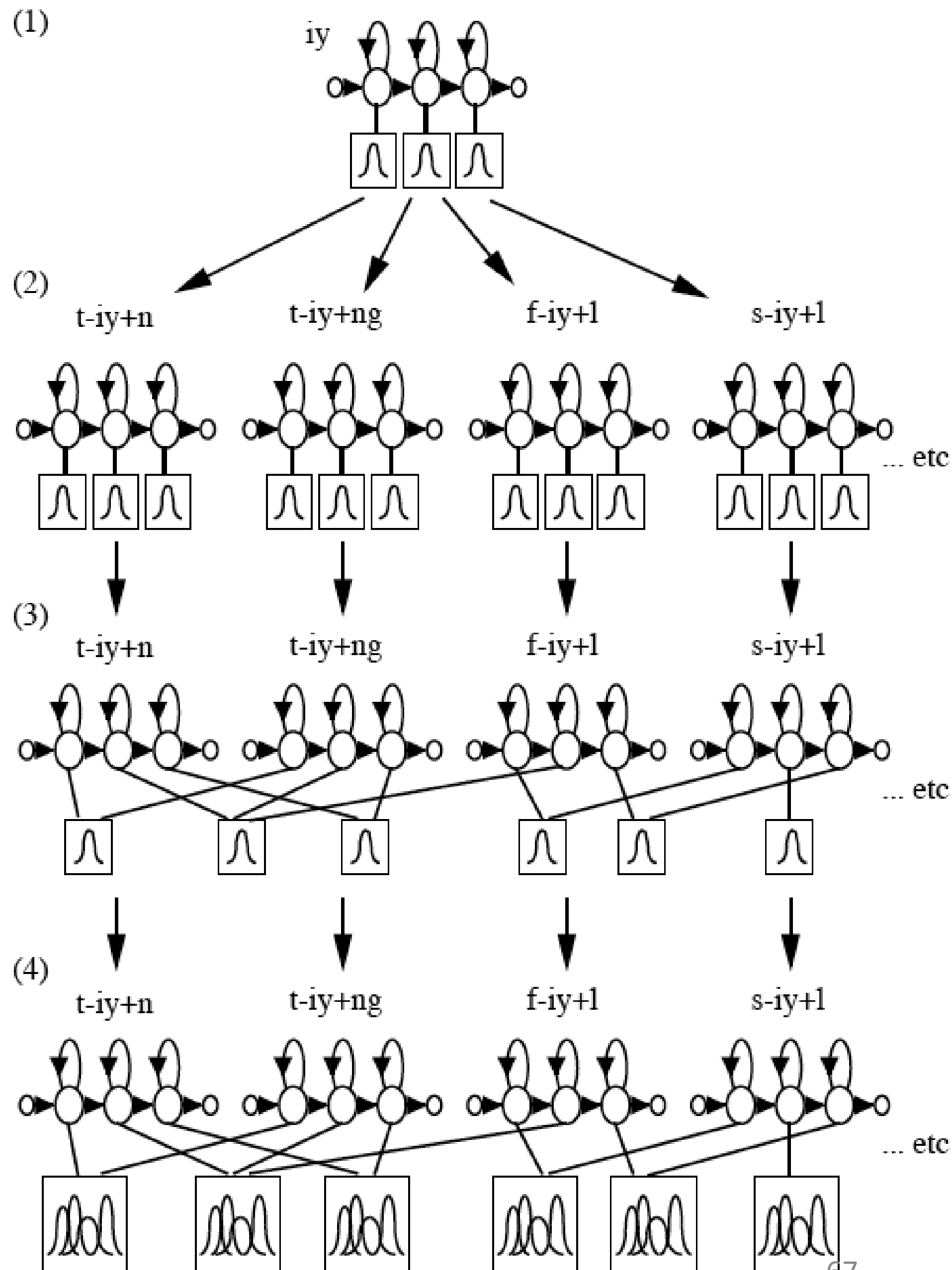
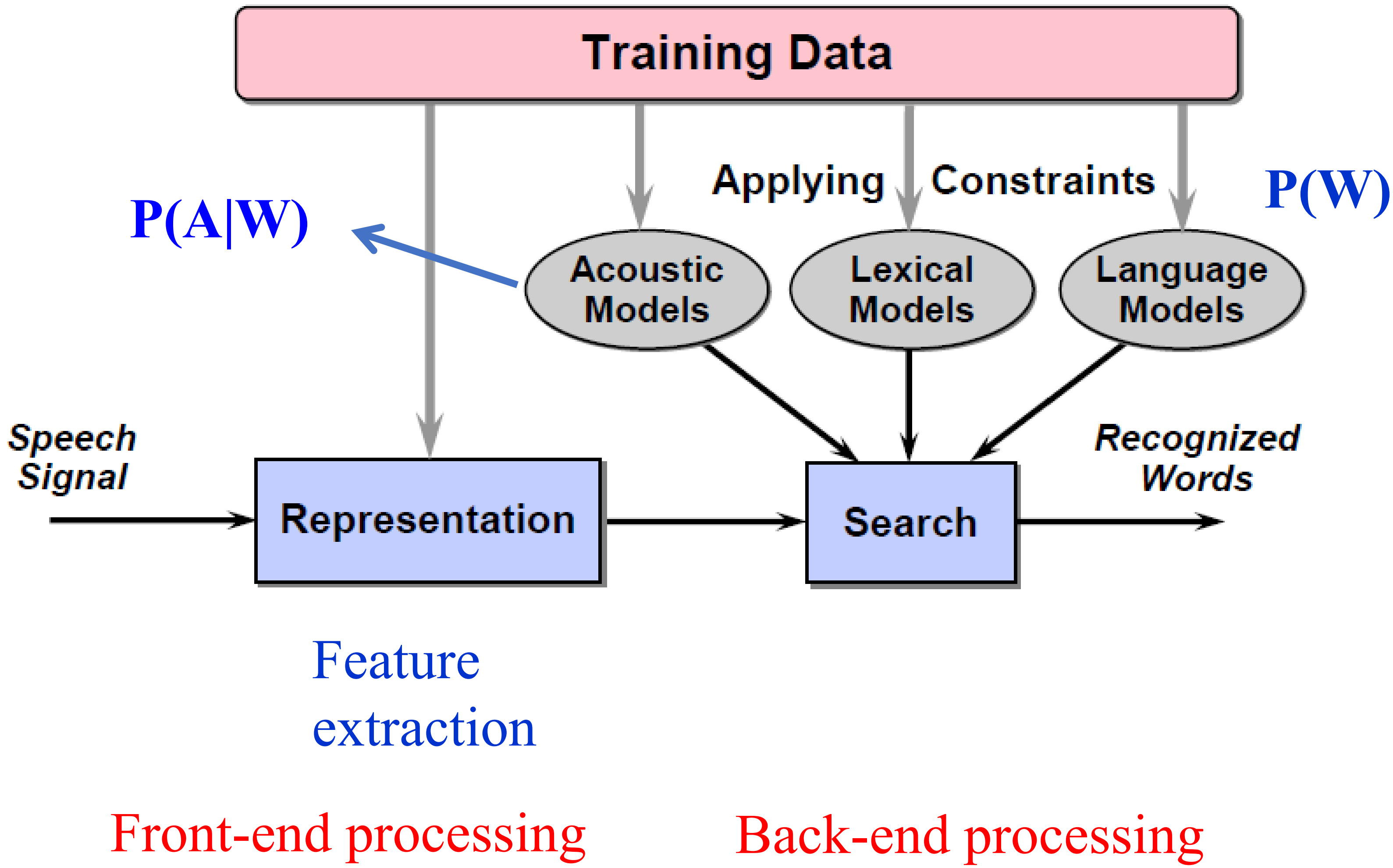
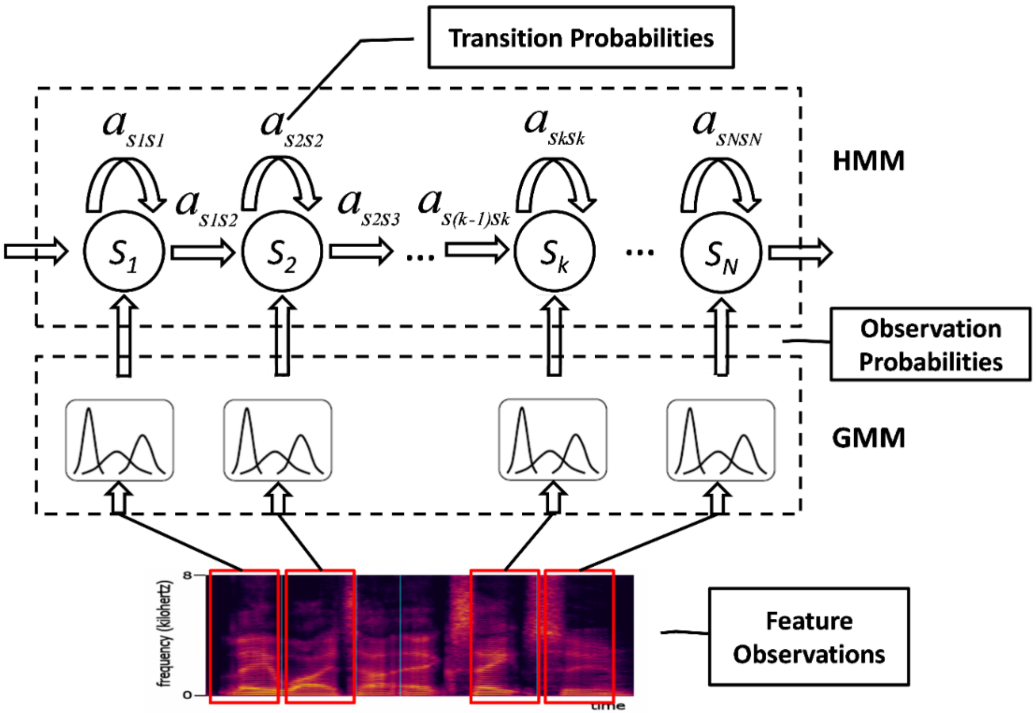


Diagram of ASR system (Review)



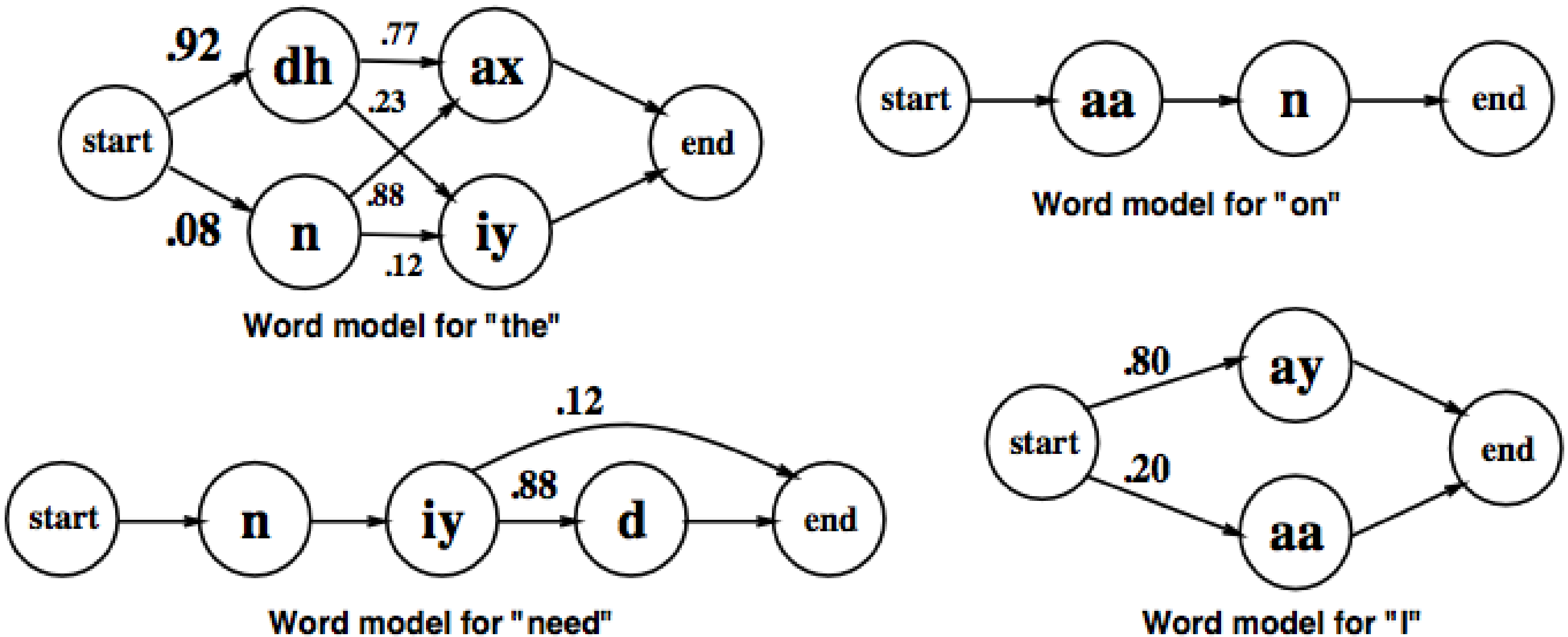
Acoustic Model (Review)

- GMM-HMM



- L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. Proc. IEEE, 77(2):257–286, 1989.

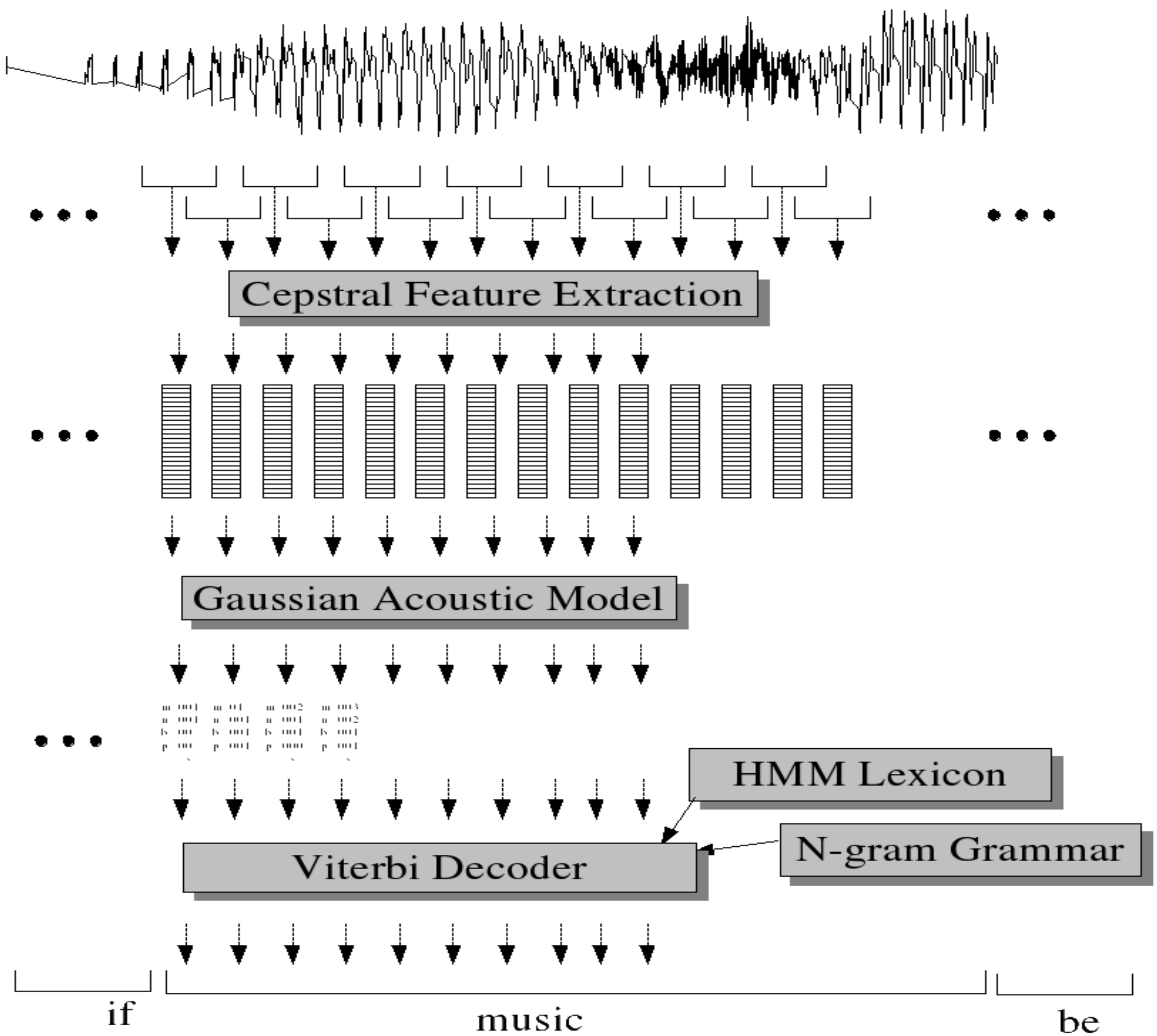
ASR Lexicon: Markov Models for pronunciation



Summary: ASR Architecture

- Five easy pieces: ASR Noisy Channel architecture
  1) Feature Extraction:
     39 “MFCC” features
  2) Acoustic Model:
     Gaussians for computing p(o|q)
  3) Lexicon/Pronunciation Model
     HMM: what phones can follow each other
  4) Language Model
     N-grams for computing p(wi|wi-1)
  5) Decoder
     Viterbi algorithm: dynamic programming for combining all these to get word sequence from speech!

Summary


Cepstral Feature Extraction
Gaussian Acoustic Model
HMM Lexicon
N-gram Grammar
Viterbi Decoder
if
music
be

4.2 Acoustic Model

- 4.2.1 Hidden Markov Model
- 4.2.2 GMM-HMM
- 4.2.3 DNN-HMM

Output probability distribution of HMM

- Discrete Distribution HMM
  - ✓Vector Quantization (VQ)

- Continues Distribution HMM (CD-HMM)
  - ✓Gaussian Mixture Model (GMM)
  - ✓Deep Neural Network (DNN)

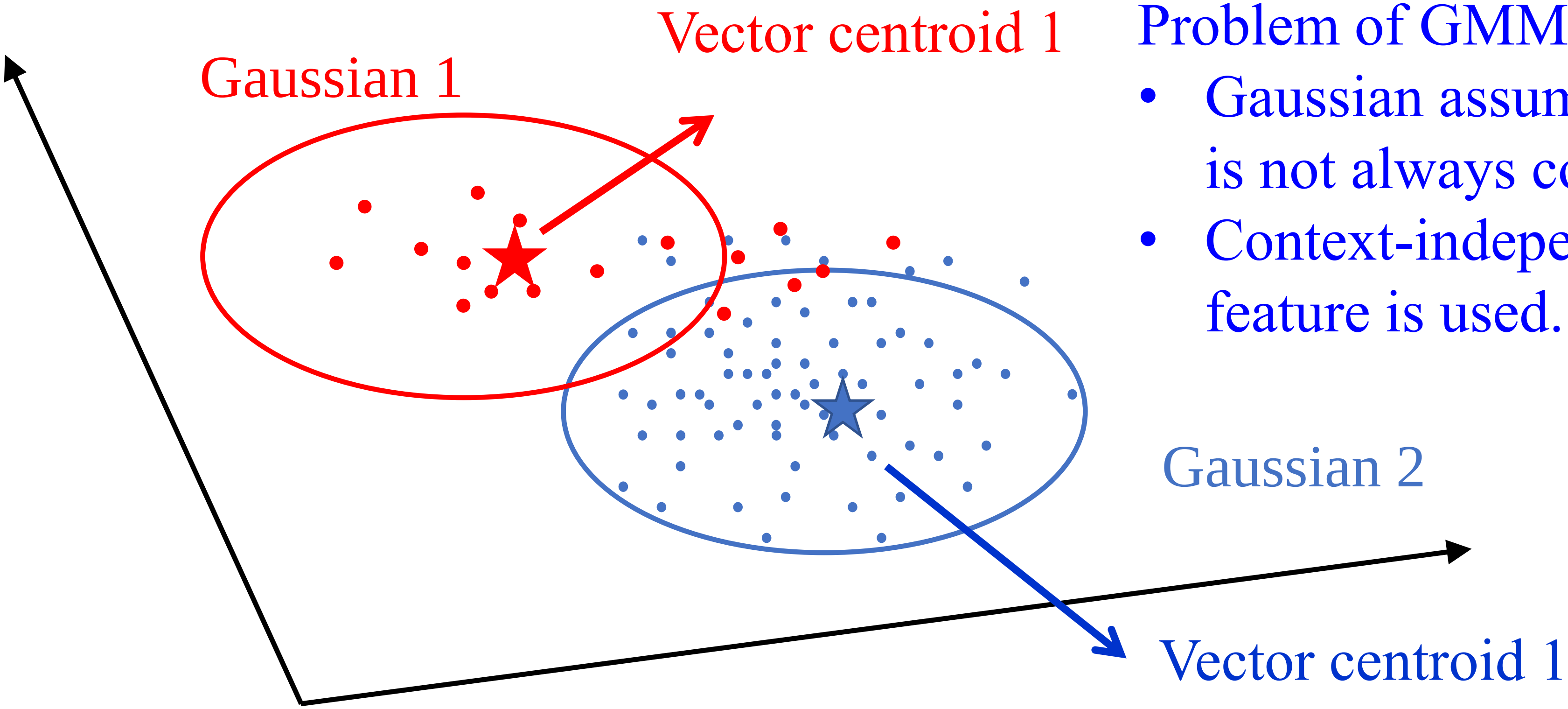
GMM

A Gaussian mixture model (GMM) is a weighted sum of M component Gaussian densities as given by the equation,

$$p(\mathbf{x}|\lambda) = \sum_{i=1}^{M} w_i \, g(\mathbf{x}|\boldsymbol{\mu}_i, \mathbf{\Sigma}_i),$$

$$g(\mathbf{x}|\boldsymbol{\mu}_i, \mathbf{\Sigma}_i) = \frac{1}{(2\pi)^{D/2}|\Sigma_i|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)' \, \Sigma_i^{-1} \, (\mathbf{x}-\boldsymbol{\mu}_i)\right\},$$

Problem of VQ and GMM



Problem of GMM:
- Gaussian assumption is not always correct.
- Context-independent feature is used.

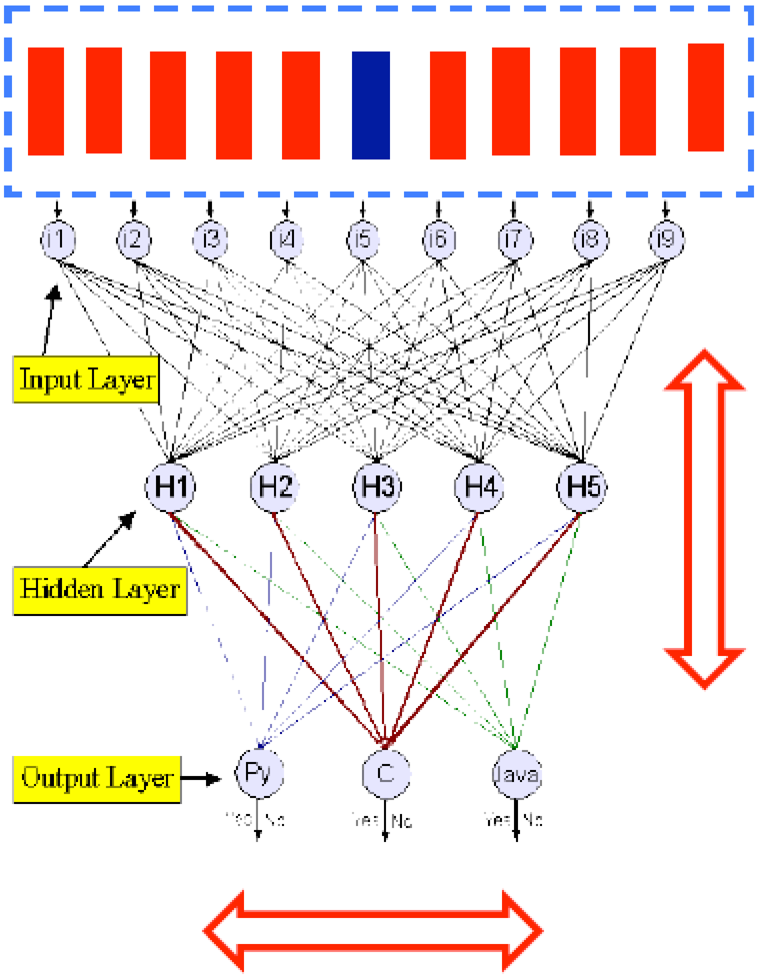
- 2D features
- Single Gaussian model: **Assumption of Gaussian distribution**

Neural Network for ASR

Neural Network (NN) with any distribution assumption was applied to ASR (1990s).

- **1990s: MLP for ASR** *(Bourlard and Morgan, 1994)*
  - **NN/HMM hybrid model (worse than GMM/HMM)**
- **2000s: TANDEM** *(Hermansky, Ellis, et al., 2000)*
  - **Use MLP as Feature Extraction (5-10% rel. gain)**
- **2006: DNN for small tasks** *(Hinton et al., 2006)*
  - **RBM-based pre-training for DNN**
- **2010: DNN for small-scale ASR** *(Mohamed, Yi, et al. 2010)*
- **2011: DNN for large-scale ASR**
  - **Over 30% rel. gain in Switchboard** *(Seide et al., 2011)*

Why DNN is success for ASR? NN for ASR: old and new

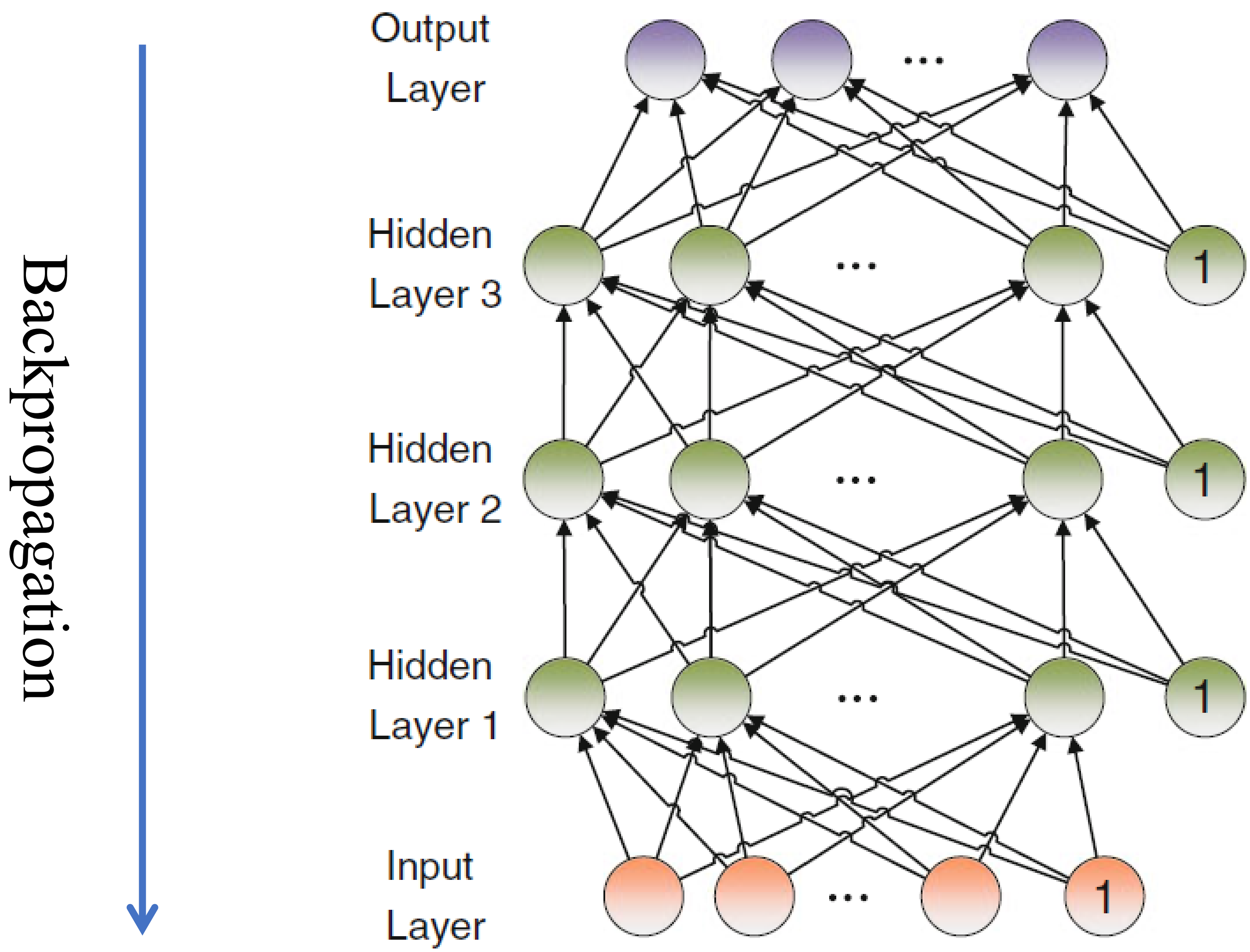


- Deeper network
  - more hidden layers
  - (1 → 6-7 layers)
- Wider network
  - More hidden nodes
  - More output nodes
  - (100 → 5-10 K)
- More data
  - 10-20 hours → 2-10 k hours training data
- **Longer context**

Key points of success

- More data
- Faster computer
- Good pre-training algorithm

Deep Neural Network (DNN)



Training criteria

- Classification task

Cross-entropy (CE) criterion

$$J_{\mathrm{CE}}(\mathbf{W}, \mathbf{b}; \mathbb{S}) = \frac{1}{M} \sum_{m=1}^{M} J_{\mathrm{CE}}\left(\mathbf{W}, \mathbf{b}; \mathbf{o}^m, \mathbf{y}^m\right)$$

$$J_{\mathrm{CE}}(\mathbf{W}, \mathbf{b}; \mathbf{o}, \mathbf{y}) = -\sum_{i=1}^{C} y_i \log v_i^L,$$

$$y_i = P_{\mathrm{emp}}(i|\mathbf{o})$$

$$v_i^L = P_{\mathrm{dnn}}(i|\mathbf{o})$$

To minimize the cost function

Training criteria

- Regression task

Mean square error (MSE) criterion

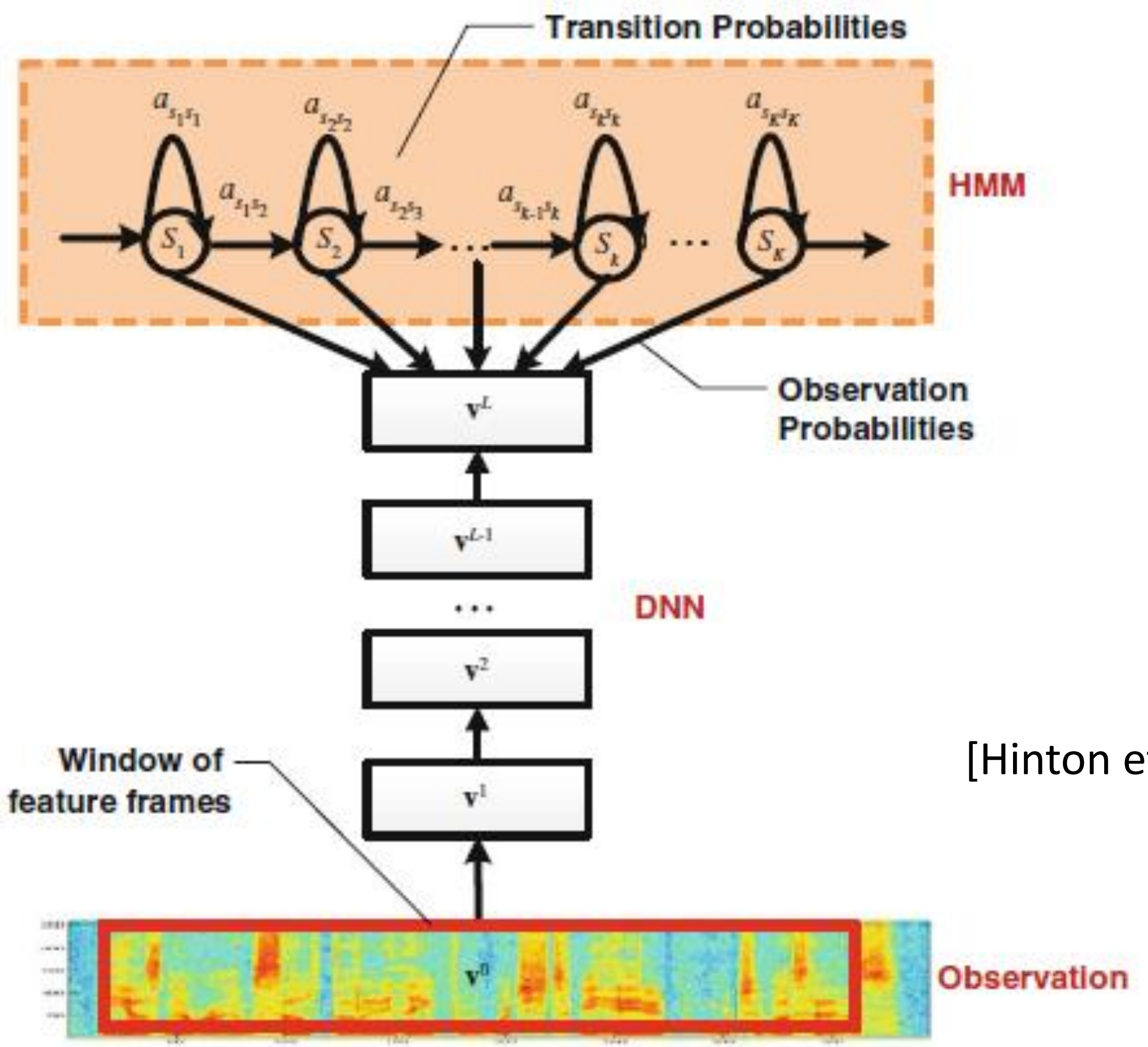
$$J_{\text{MSE}}(\mathbf{W}, \mathbf{b}; \mathbb{S}) = \frac{1}{M} \sum_{m=1}^{M} J_{\text{MSE}}\left(\mathbf{W}, \mathbf{b}; \mathbf{o}^m, \mathbf{y}^m\right)$$

$$J_{\text{MSE}}(\mathbf{W}, \mathbf{b}; \mathbf{o}, \mathbf{y}) = \frac{1}{2} \left\| \mathbf{v}^L - \mathbf{y} \right\|^2 = \frac{1}{2} \left(\mathbf{v}^L - \mathbf{y}\right)^{\mathrm{T}} \left(\mathbf{v}^L - \mathbf{y}\right).$$

To minimize the cost function

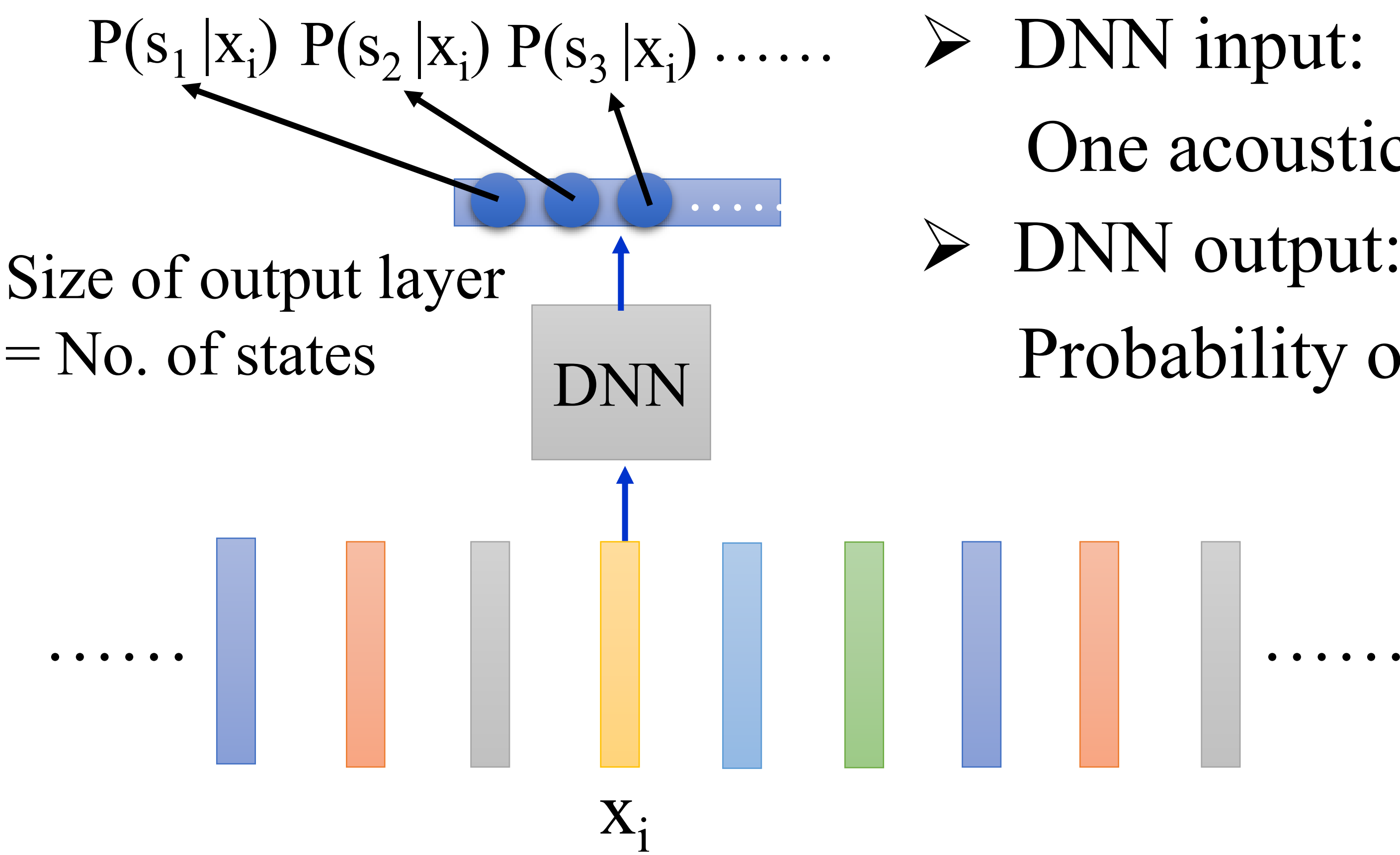
DNN-HMM for ASR

Architecture of DNN-HMM system



[Hinton et al. 2012]

What DNN can do ?



- DNN input:
  One acoustic feature
- DNN output:
  Probability of each state

GMM-HMM Based Acoustic Model

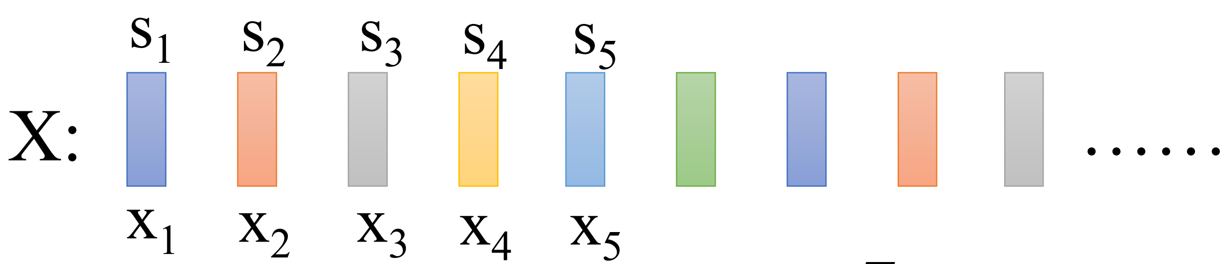
$$\widetilde{W} = arg \max_{W} \boxed{P(X|W)} P(W)$$

P(X|W) = P(X|S)

W: What's your name?

S: s_1 s_2 s_3 s_4 s_5 ……

Actually, we don't know the alignment.



$$P(X|S) \approx \max_{s_1 \cdots s_T} \prod_{t=1}^{T} P(s_t|s_{t-1}) P(x_t|s_t)$$

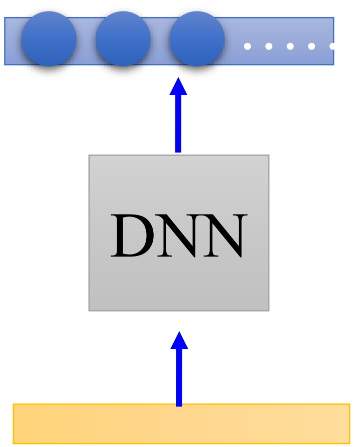
(Viterbi algorithm)

DNN-HMM Hybrid Acoustic Model

$$\tilde{W} = arg \max_{W} P(W|X) = arg \max_{W} P(X|W)P(W)$$

$$P(X|W) \approx \max_{s_1 \cdots s_T} \prod_{t=1}^{T} P(s_t|s_{t-1}) \underline{P(x_t|s_t)}$$

From DNN



$P(s_t|x_t)$

x_t

$$P(x_t|s_t) = \frac{P(x_t, s_t)}{P(s_t)}$$

$$= \frac{P(s_t|x_t)\cancel{P(x_t)}}{P(s_t)}$$

Count from training data

DNN-HMM Hybrid Acoustic Model

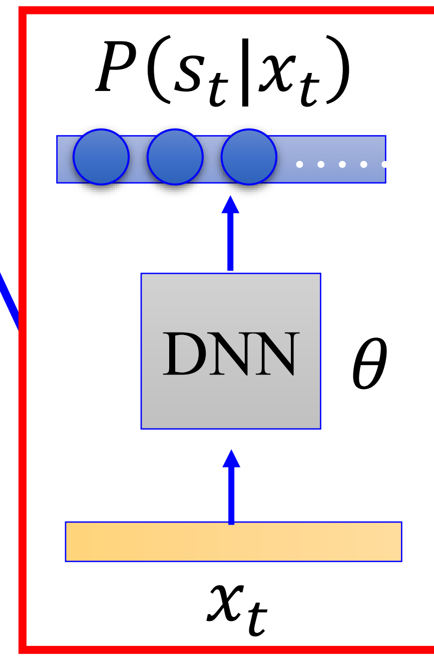
$$P(X|W) \approx \max_{s_1 \cdots s_T} \prod_{t=1}^{T} P(s_t|s_{t-1})\, P(x_t|s_t)$$

$$\cancel{P(X|W)}\ P(X|W;\theta) \approx \max_{s_1 \cdots s_T} \prod_{t=1}^{T} P(s_t|s_{t-1})\, \frac{P(s_t|x_t)}{P(s_t)}$$

From original HMM

Count from training data

DNN



Decoding with DNN-HMM

$$\hat{w} = \arg\max_w p(w|\mathbf{x}) = \arg\max_w p(\mathbf{x}|w)\,p(w)/p(\mathbf{x})$$
$$= \arg\max_w p(\mathbf{x}|w)\,p(w),$$

Same as GMM-HMM

[TABLE 1] COMPARISONS AMONG THE REPORTED SPEAKER-INDEPENDENT (SI) PHONETIC RECOGNITION ACCURACY RESULTS ON TIMIT CORE TEST SET WITH 192 SENTENCES.

METHOD	PER
CD-HMM [26]	27.3%
AUGMENTED CONDITIONAL RANDOM FIELDS [26]	26.6%
RANDOMLY INITIALIZED RECURRENT NEURAL NETS [27]	26.1%
BAYESIAN TRIPHONE GMM-HMM [28]	25.6%
MONOPHONE HTMS [29]	24.8%
HETEROGENEOUS CLASSIFIERS [30]	24.4%
MONOPHONE RANDOMLY INITIALIZED DNNs (SIX LAYERS) [13]	23.4%
MONOPHONE DBN-DNNs (SIX LAYERS) [13]	22.4%
MONOPHONE DBN-DNNs WITH MMI TRAINING [31]	22.1%
TRIPHONE GMM-HMMs DT W/ BMMI [32]	21.7%
MONOPHONE DBN-DNNs ON FBANK (EIGHT LAYERS) [13]	20.7%
MONOPHONE MCRBM-DBN-DNNs ON FBANK (FIVE LAYERS) [33]	20.5%
MONOPHONE CONVOLUTIONAL DNNs ON FBANK (THREE LAYERS) [34]	20.0%

DNN vs. GMM

- **Table:** Voice Search SER (24-48 hours of training)

Features	Setup	Error Rates
Pre-DNN	GMM-HMM with MPE	36.2%
DNN	5 layers x 2048	30.1%

~20% relative improvement

- **Table:** SwitchBoard WER (309 hours training)

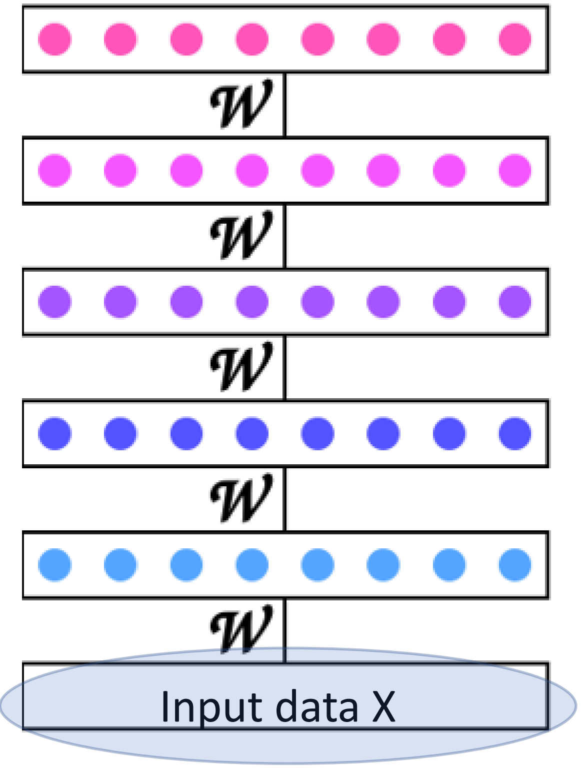
Features	Setup	Error Rates
Pre-DNN	GMM-HMM with BMMI	23.6%
DNN	7 layers x 2048	15.8%

~30% relative Improvement

For DNN, the more data, the better!

Deep neural nets (DNN) & many of the variants

Innovation: Towards Raw Inputs



- **Bye-Bye MFCCs (no more cosine transform, Mel-scaling?)**

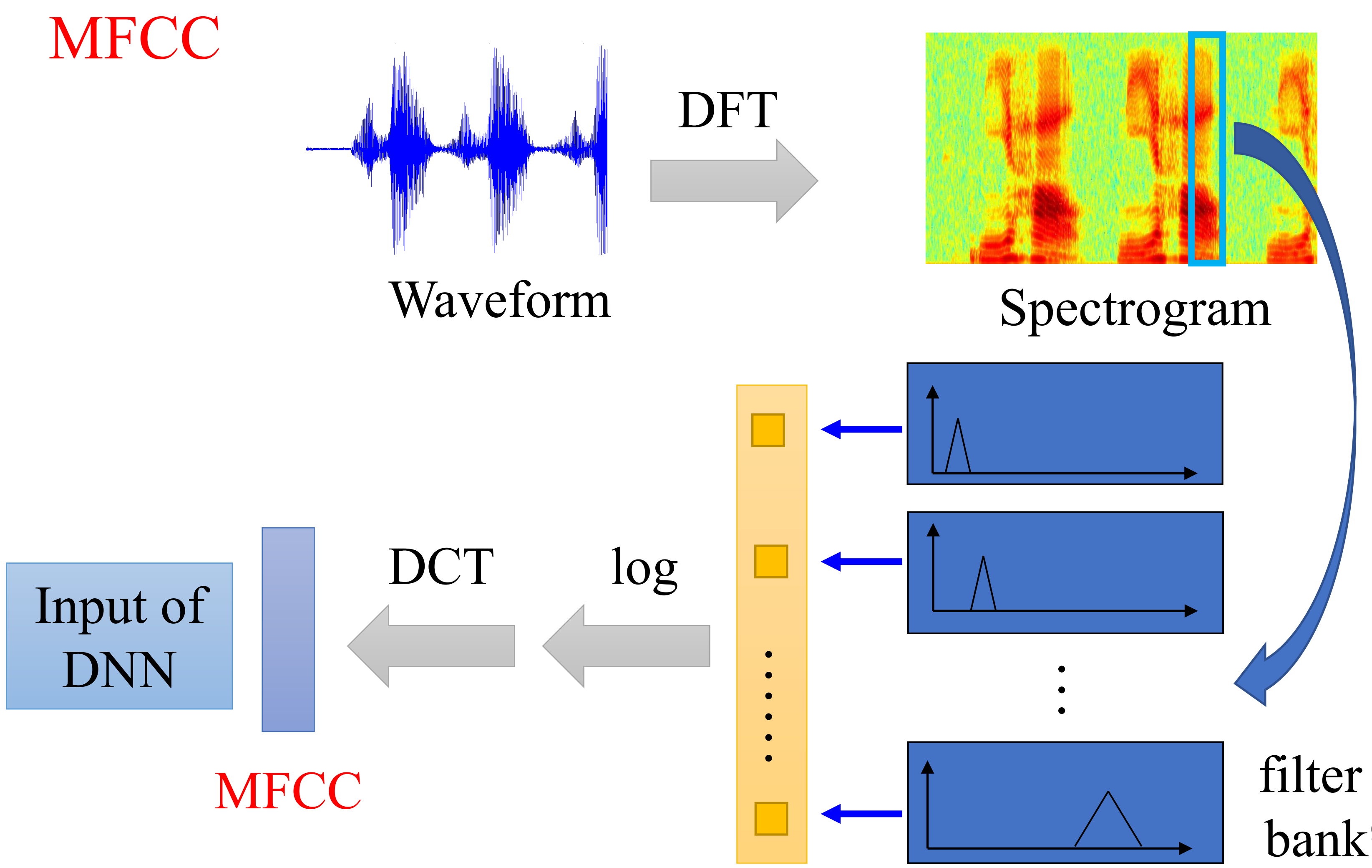
- Deng, Seltzer, Yu, Acero, Mohamed, Hinton. "Binary coding of speech spectrograms using a deep auto-encoder," **Interspeech, 2010**.
- Mohamed, Hinton, Penn. "Understanding how deep belief networks perform acoustic modeling," ICASSP, 2012.
- Li, Yu, Huang, Gong, "Improving wideband speech recognition using mixed-bandwidth training data in CD-DNN-HMM" SLT, 2012
- Deng, J. Li, Huang, Yao, Yu, Seide, Seltzer, Zweig, He, Williams, Gong, Acero. "Recent advances in deep learning for speech research at Microsoft," ICASSP, 2013.
- Sainath, Kingsbury, Mohamed, Ramabhadran. "Learning filter banks within a deep neural network framework," ASRU, 2013.

- **Bye-Bye Fourier transforms?**

- Jaitly and Hinton. "Learning a better representation of speech sound waves using RBMs," ICASSP, 2011.
- Tuske, Golik, Schluter, Ney. "Acoustic modeling with deep neural networks using raw time signal for LVCSR," Interspeech, 2014.
- Golik et al, "Convolutional NNs for acoustic modeling of raw time signals in LVCSR," Interspeech, 2015.
- Sainath et al. "Learning the Speech Front-End with Raw Waveform CLDNNs," Interspeech, 2015

Innovation: Towards Raw Inputs

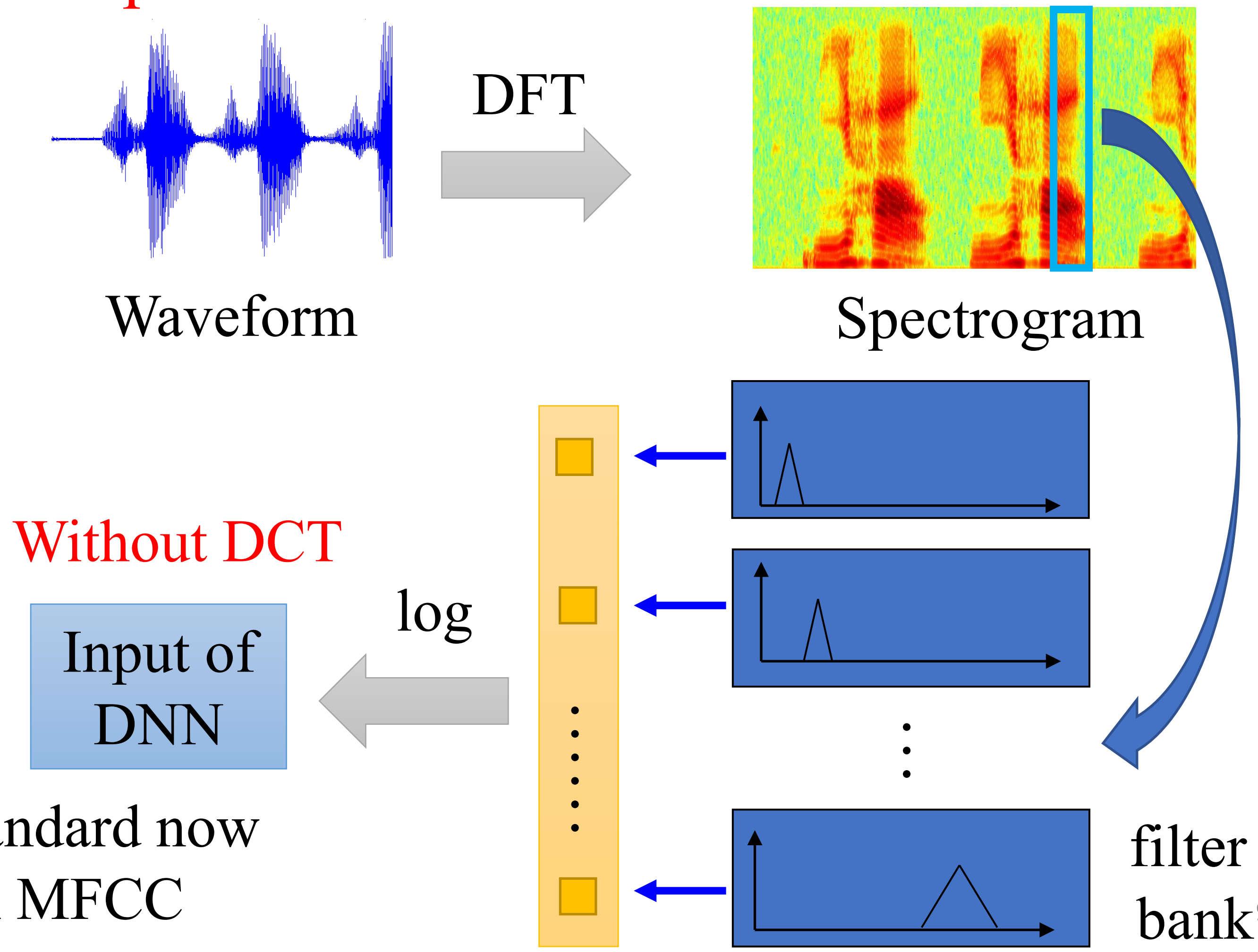
- **Bye-Bye MFCCs (no more cosine transform, Mel-scaling?)**



Innovation: Towards Raw Inputs

- **Bye-Bye MFCCs (no more cosine transform, Mel-scaling?)**

Filter-bank Output

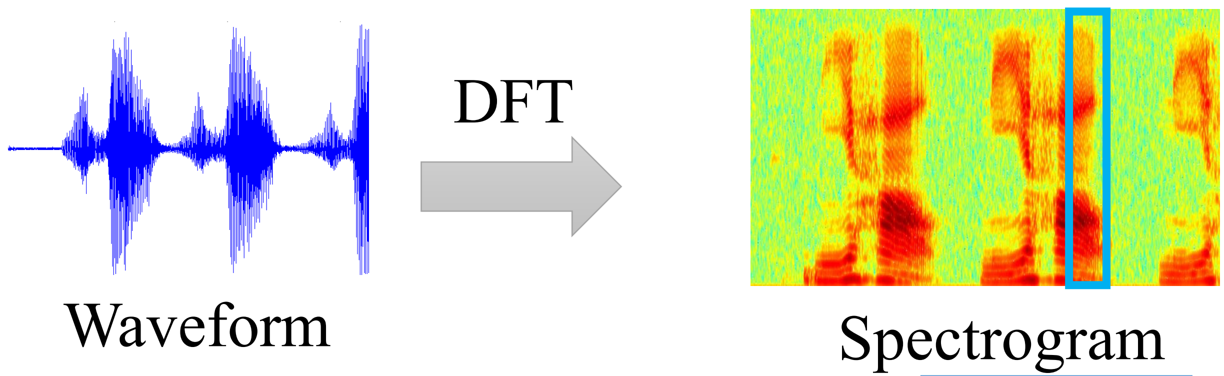


- ➢ Kind of standard now
- ➢ Better than MFCC

Innovation: Towards Raw Inputs

- **Bye-Bye MFCCs (no more cosine transform, Mel-scaling?)**

Spectrogram



Input of DNN

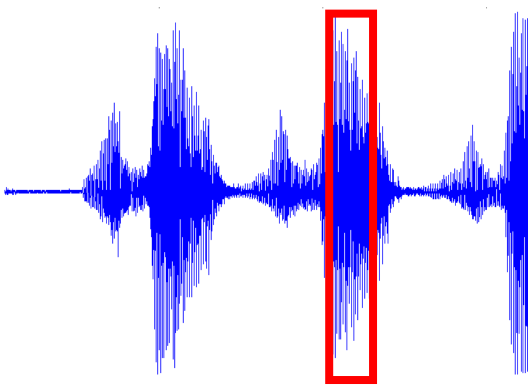
➢ common today

➢ 5% relative improvement over filter-bank output

Innovation: Towards Raw Inputs

- **Bye-Bye Fourier transforms?**

Waveform?



Waveform

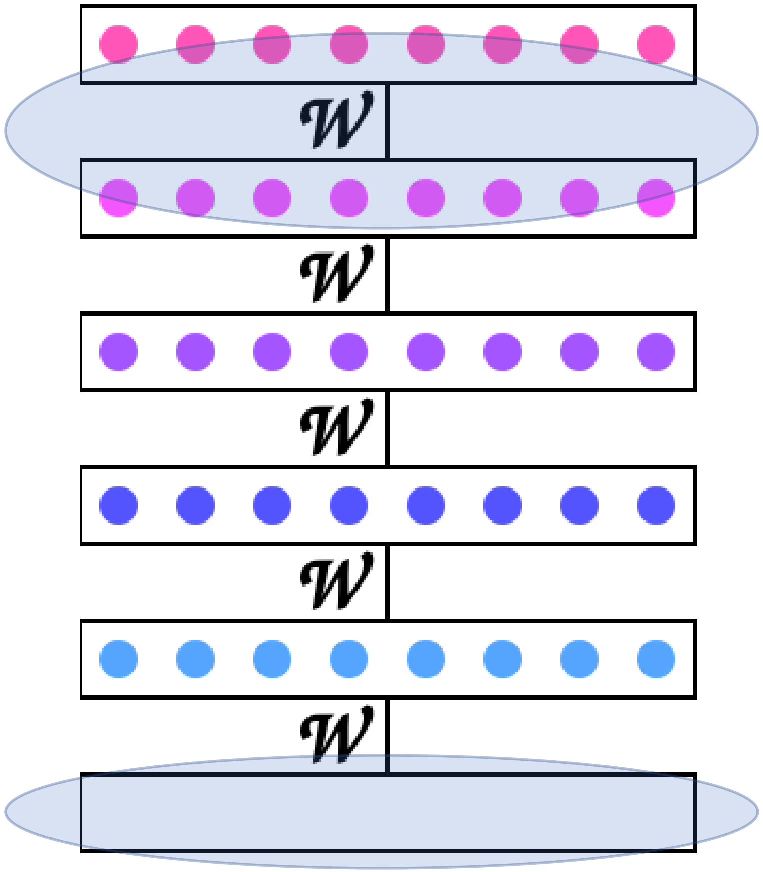
Input of DNN

➢ If success, no Signal & Systems

➢ Researchers tried, but not better than spectrogram yet

➢ Still need to take Signal & Systems ……

Innovation: Transfer/Multitask Learning & Adaptation



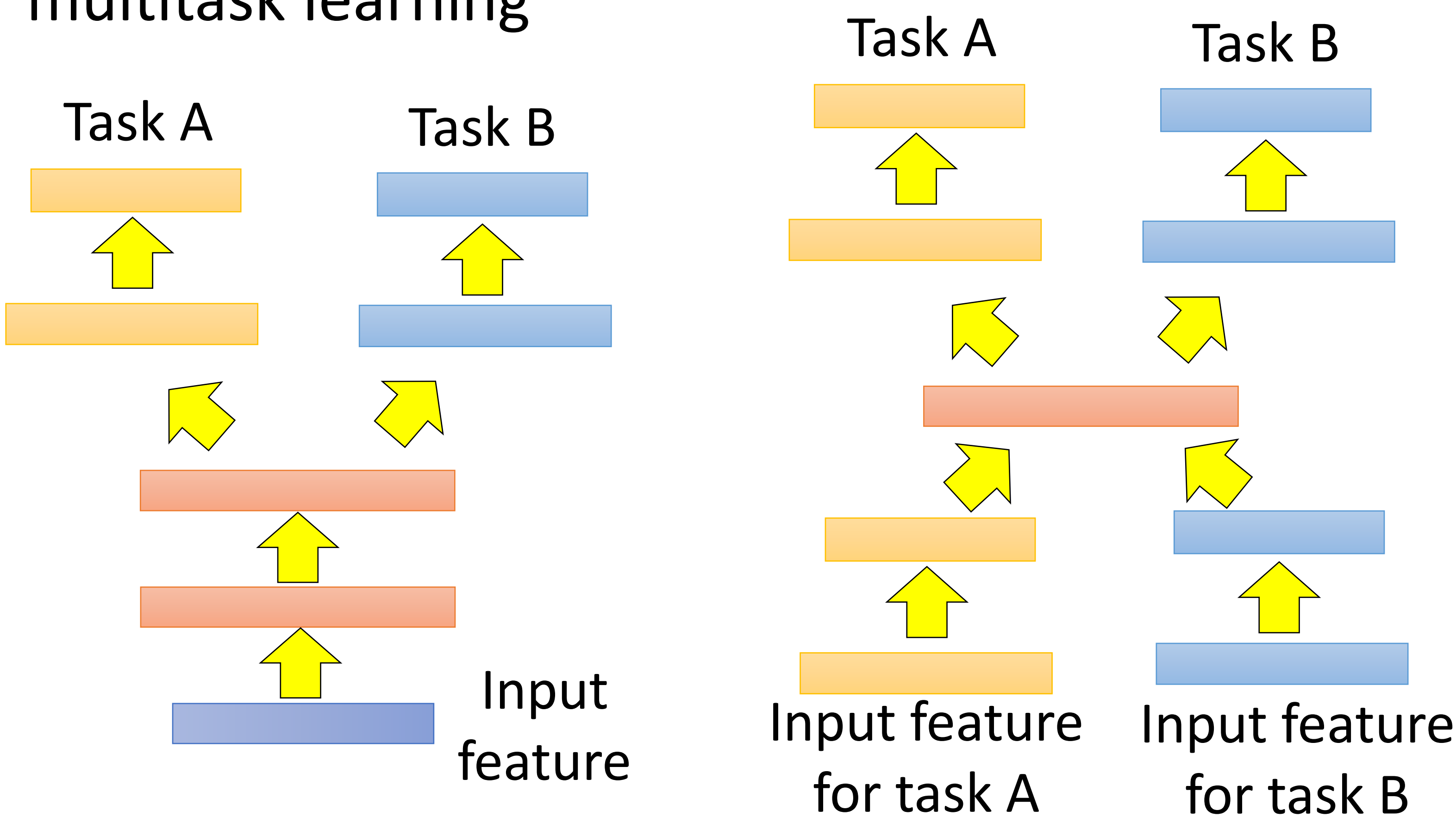
Multi-lingual acoustic modeling

Adaptation to speakers & environments (i-vectors)

Mixed-bandwidth acoustic modeling

Innovation: Multitask Learning

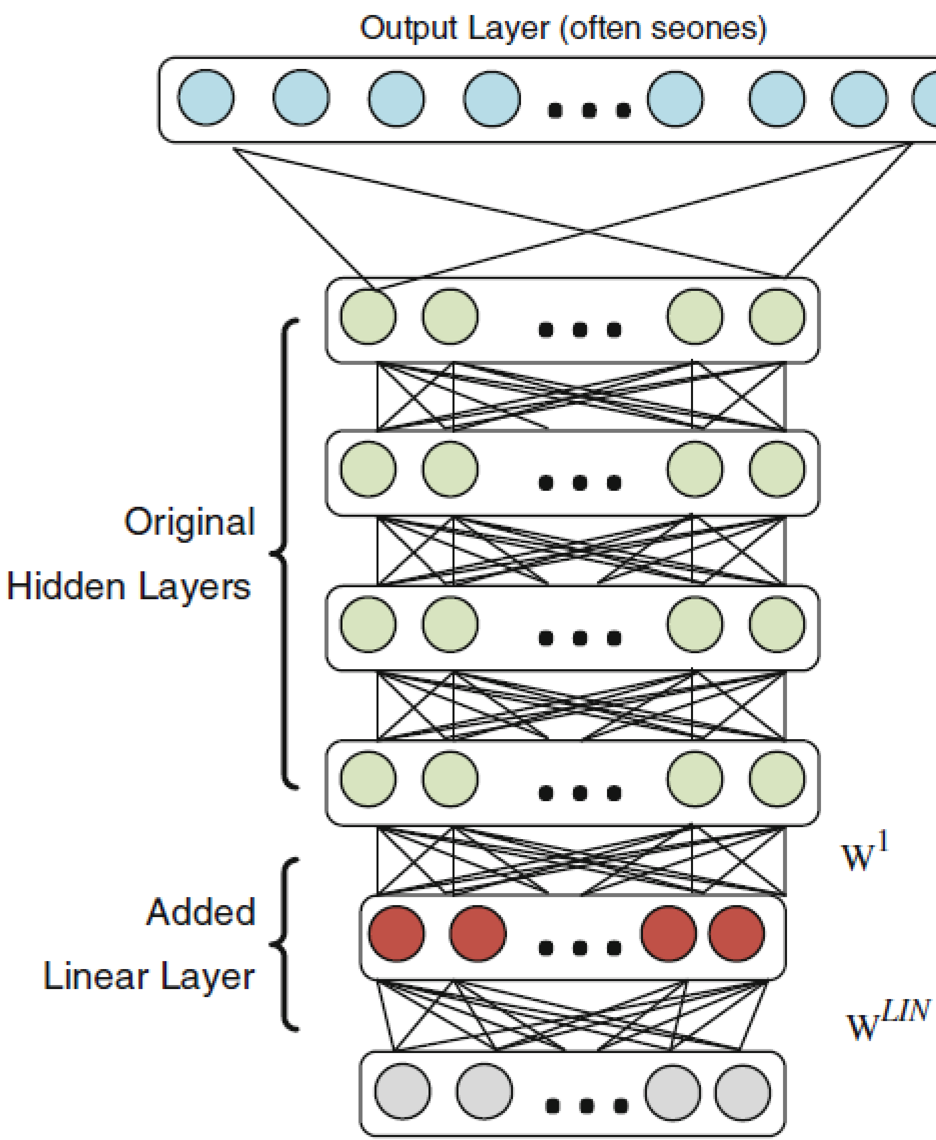
- The multi-layer structure makes DNN suitable for multitask learning



Innovation: Adaptation for DNN

- Linear Transformation
- Conservative Training
- Training with augment information

Innovation: Adaptation-Linear Transformation



Innovation: Adaptation-Conservative Training

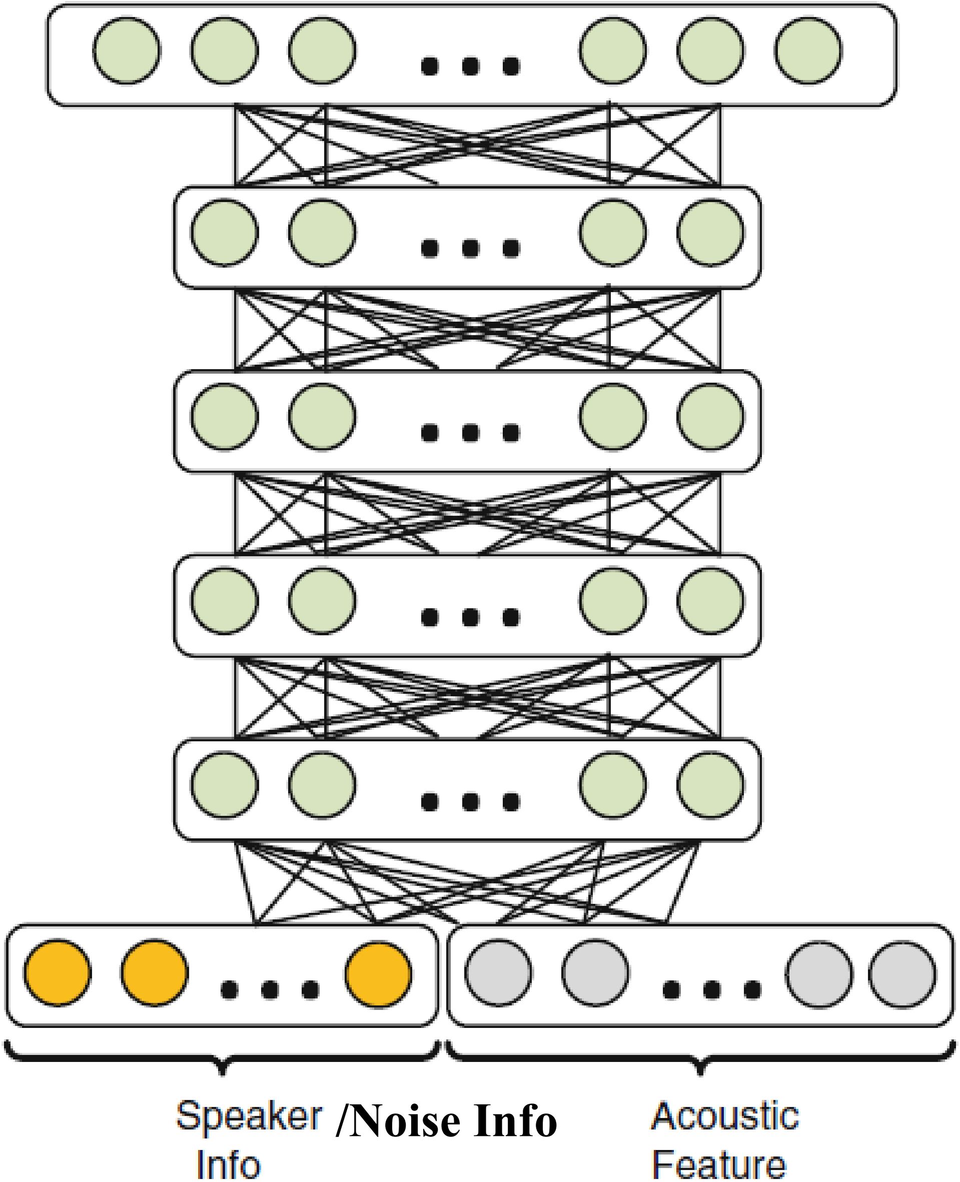
The basic idea of the L_2 regularized CT is to add the L_2 norm of the model parameter difference

$$R_2\left(\mathbf{W}_{\mathrm{SI}}-\mathbf{W}\right)=\left\|\operatorname{vec}\left(\mathbf{W}_{\mathrm{SI}}-\mathbf{W}\right)\right\|_2^2$$
$$=\sum_{\ell=1}^{L}\left\|\operatorname{vec}\left(\mathbf{W}_{\mathrm{SI}}^{\ell}-\mathbf{W}^{\ell}\right)\right\|_2^2$$

$$J_{L_2}\left(\mathbf{W},\mathbf{b};\mathbb{S}\right)=J\left(\mathbf{W},\mathbf{b};\mathbb{S}\right)+\lambda R_2\left(\mathbf{W}_{\mathrm{SI}},\mathbf{W}\right)$$

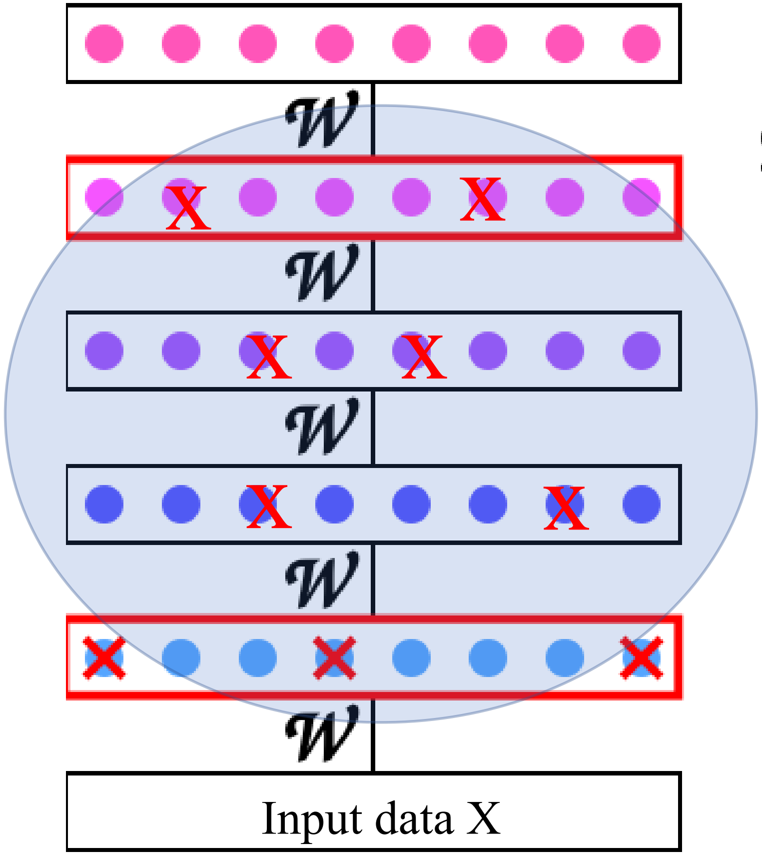
L2 Regularization

Innovation: Adaptation-
Training with augment information



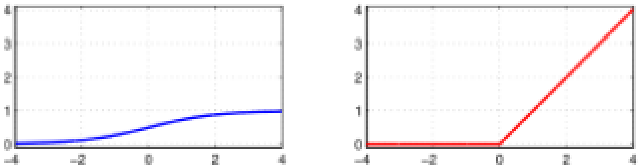
Speaker-aware, Noise-aware Training

Innovation: Better regularization & nonlinearity



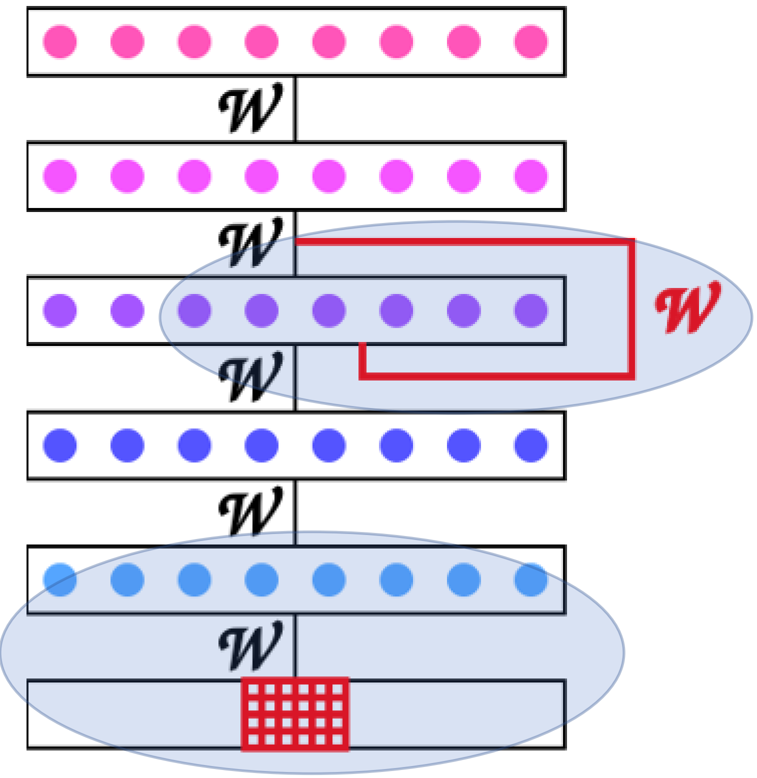
Sparsity in hidden representations

logistic ⟶ ReLU , MaxOut,



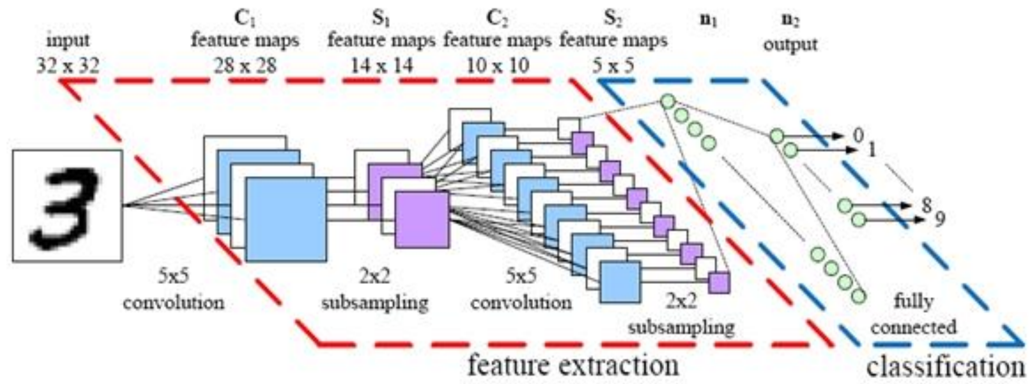
Dropout

Innovation: Better architectures



- **Recurrent Nets (bi-directional RNN/LSTM), Conv Nets (CNN) and Time Delay NN (TDNN) are superior to fully-connected DNNs**
- Sak, Senior, Beaufays. “LSTM Recurrent Neural Network architectures for large scale acoustic modeling,” Interspeech, 2014.
- Soltau, Saon, Sainath. ”Joint Training of Convolutional and Non-Convolutional Neural Networks,” ICASSP, 2014.

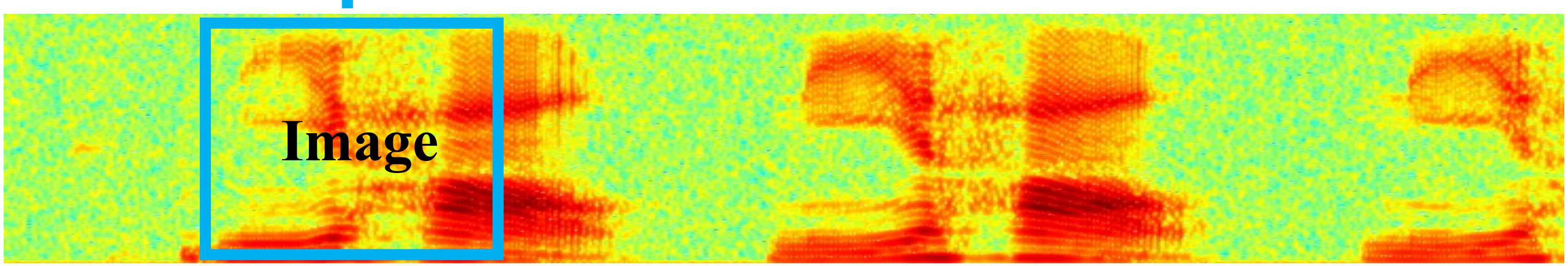
Innovation: Better architectures



Probabilities of states

CNN

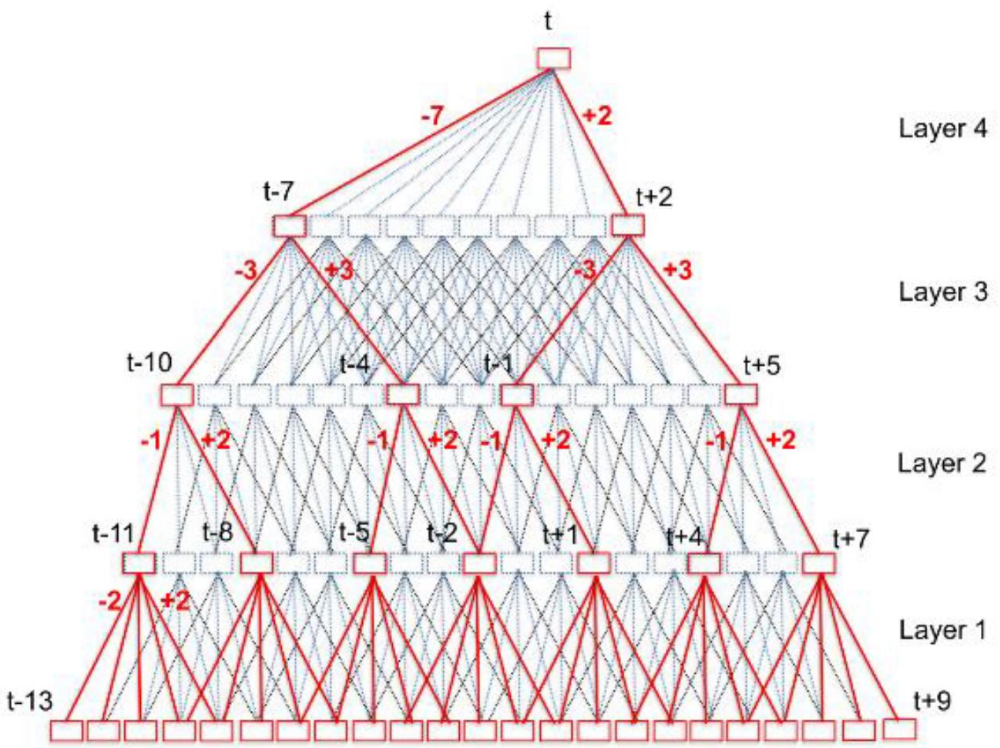
Replace DNN by CNN



Spectrogram

Innovation: Better architectures

Time Delay Neural Network (TDNN)



Innovation: Ensemble Deep Learning

- **Ensembles of RNN/LSTM, DNN, & Conv Nets (CNN) give huge gains (state of the art):**
- T. Sainath, O. Vinyals, A. Senior, H. Sak. “Convolutional, Long Short-Term Memory, Fully Connected Deep Neural Networks,” ICASSP 2015.
- L. Deng and John Platt, Ensemble Deep Learning for Speech Recognition, Interspeech, 2014.
- G. Saon, H. Kuo, S. Rennie, M. Picheny. “The IBM 2015 English conversational telephone speech recognition system,” arXiv, May 2015. (8% WER on SWB-309h)

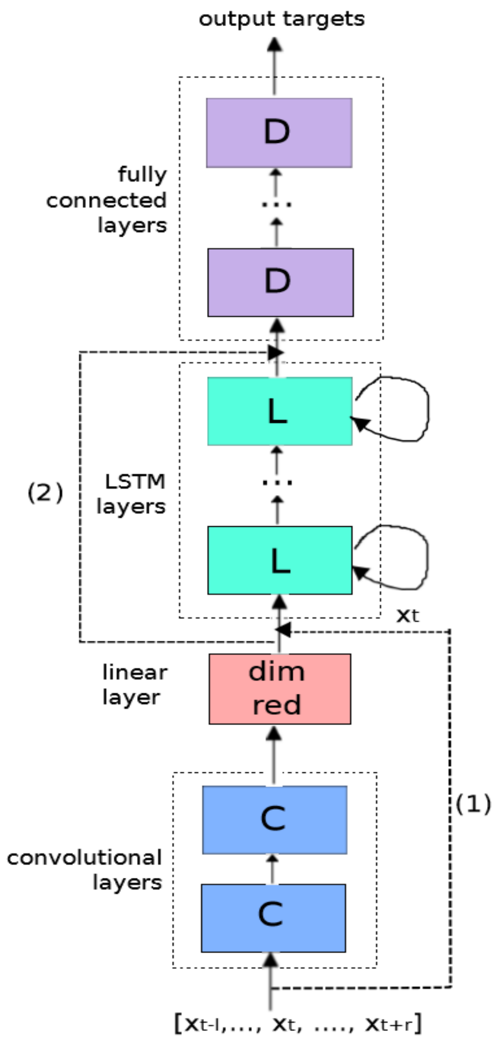
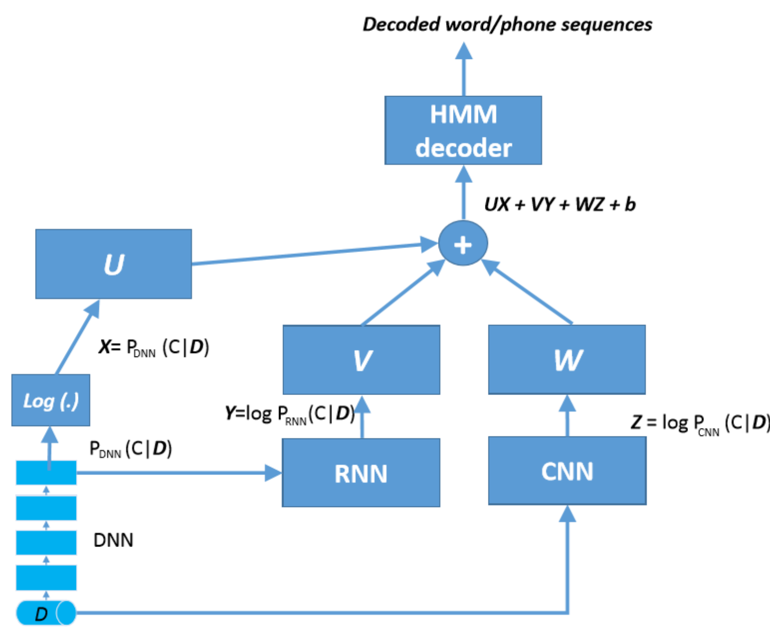
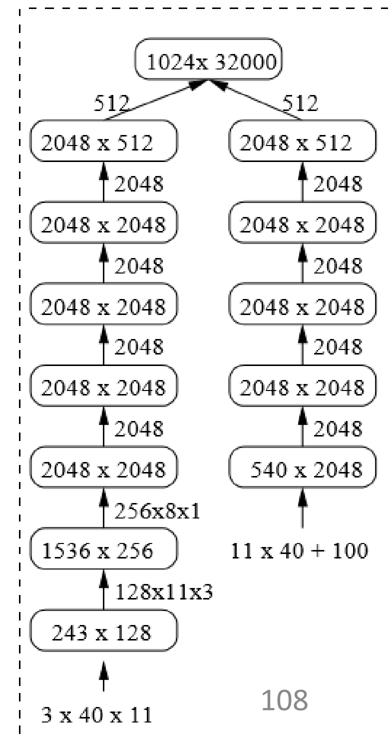


Fig. 1. CLDNN Architecture





Joint CNN/DNN