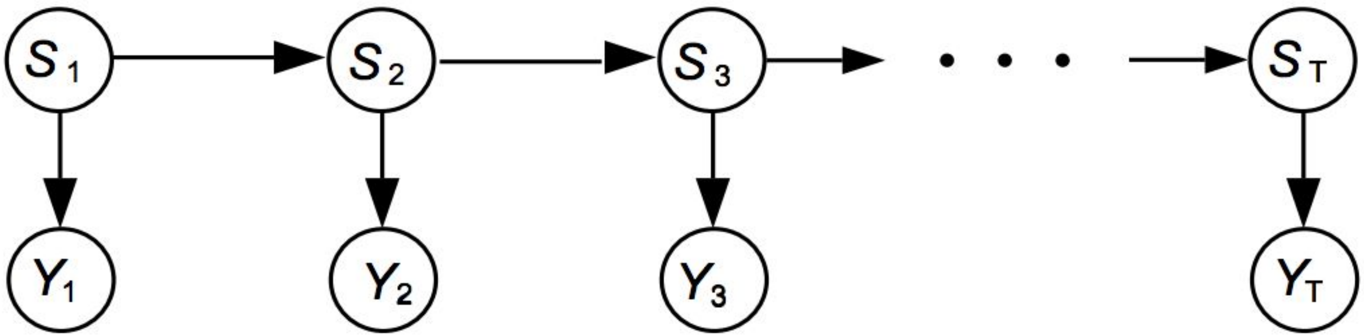
Hidden Markov Models

Gabriela Tavares and Juri Minxha
Mentor: Taehwan Kim
CS159   04/25/2017

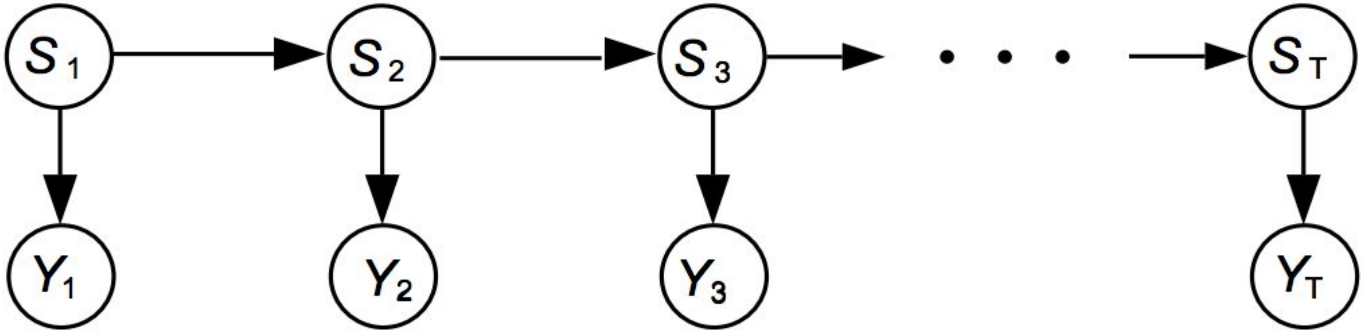
Outline

1. Brief review of HMMs
2. Hidden Markov Support Vector Machines
3. Large Margin Hidden Markov Models for Automatic Speech Recognition
4. Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition

Review of Hidden Markov Models



- A tool for representing probability distributions over sequences of observations
- A type of (dynamic) Bayesian network
- Main assumptions: **hidden states** and **Markov property**



$$P(S_{1:T}, Y_{1:T}) = P(S_1)P(Y_1|S_1)\prod_{t=2}^{T} P(S_t|S_{t-1})P(Y_t|S_t)$$

- a probability distribution over the initial state
- the state transition matrix
- the output model (emission matrix)

Learning in HMMs

- **Generative** setting: model the joint distribution of inputs and outputs
- Obtain the maximum likelihood estimate for the parameters of the HMM given a set of output sequences
- No tractable algorithm to solve this exactly
- **Baum-Welch** (especial case of EM algorithm) can be used to obtain a local maximum likelihood
- Baum-Welch makes use of the **forward-backward** algorithm

Baum-Welch

1. Initialize the model parameters: initial state distribution, transition and emission matrices
2. Compute the probability of being in state i at time t given an observed sequence and the current estimate of the model parameters
3. Compute the probability of being in state i and state j at times t and t+1, respectively, given an observed sequence and the current estimate of the model parameters
4. Use these probabilities to update the estimate of the model parameters
5. Repeat 2-4 iteratively until desired level of convergence

Forward-backward

- Forward pass: recursively compute alpha(t), the joint probability of state S(t) and the sequence of observations Y(1) to Y(t)

$$\alpha_t = P(S_t, Y_{1:t})$$

- Backward pass: compute beta(t), the conditional probabilities of the observations Y(t+1) to Y(T) given the state S(t)

$$\beta_t = P(Y_{t+1:T} | S_t)$$

- These probabilities are used to compute the expectations needed in Baum-Welch

Inference in HMMs

- **Viterbi**: a dynamic programming algorithm which can be used to find the most likely sequence of states given a sequence of observations
- Richer hidden state representations can lead to intractability when inferring hidden states from observations
- **Monte Carlo** and **variational** methods can be used to approximate the posterior distribution of the states given a set of observations

Common applications of HMMs

- Speech/phoneme recognition
- Part-of-speech tagging
- Computational molecular biology
- Data compression
- Vision: image sequence modelling, object tracking

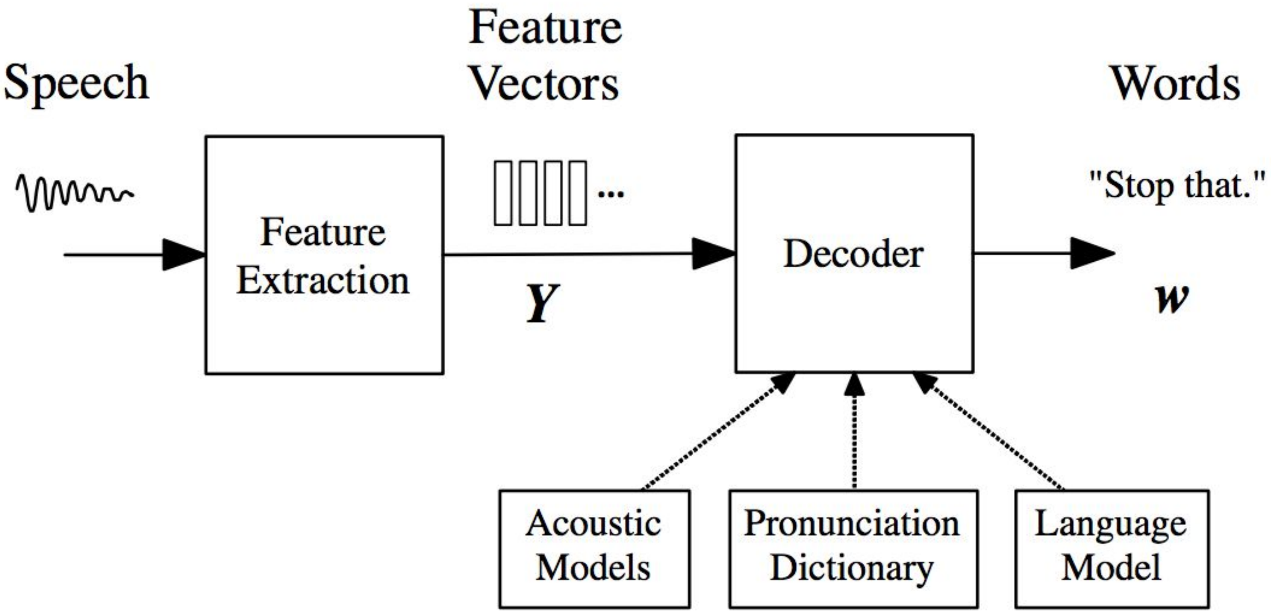
HMM for POS Tagging

- Y = “Fish sleep”
- S = (N, V)

- Y = “The dog ate my homework”
- S = (D, N, V, D, N)

- Y = “The fox jumped over the fence”
- S = (D, N, V, P, D, N)

HMM for Speech Recognition



Challenges and Limitations

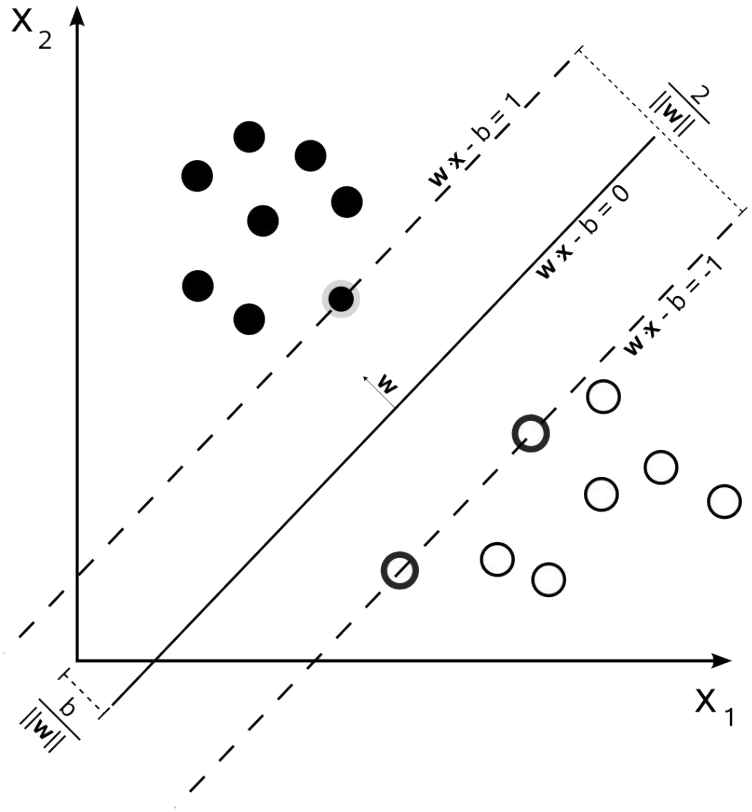
- HMMs model the joint distribution of states and observations; with a (traditionally) generative learning procedure, we lose predictive power
- Number of possible sequences grows exponentially with sequence length, which is a challenge for large margin methods
- The conditional independence assumption is too restrictive for many applications
- HMMs are based on explicit feature representations and lack the ability to model nonlinear decision boundaries
- HMMs cannot account for overlapping features

Hidden Markov Support Vector Machines

Y Altun, I Tsochantaridis and T Hoffman (ICML 2003)

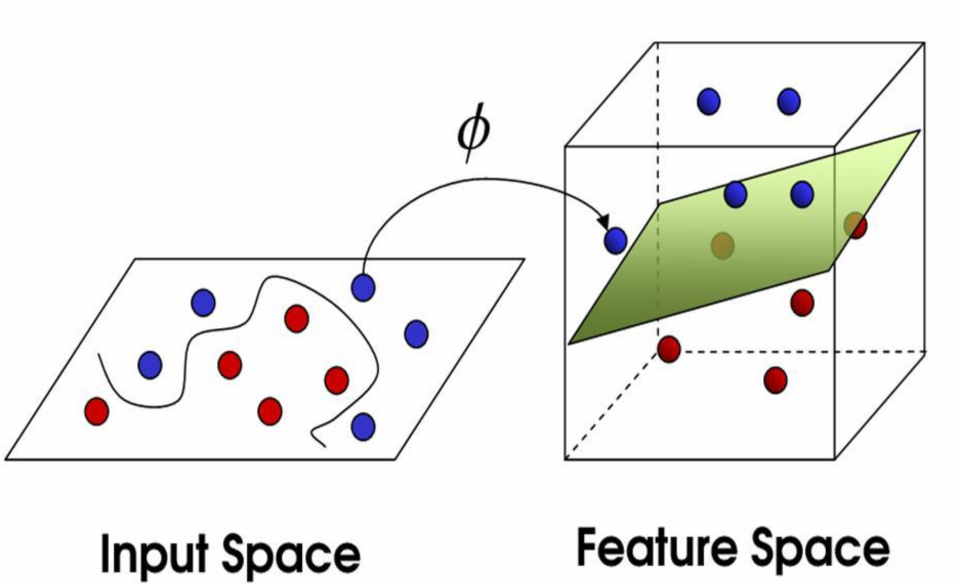
Quick Review of SVMs

- Non-probabilistic binary linear classifier
- Find the hyperplane which maximizes the margins
- Samples on the margin are called **support vectors**
- Soft margins can be used (with slack variables)
- Nonlinear classification can be achieved through the kernel trick (mapping inputs into high dimensional feature spaces)



Quick Review of SVMs

- Non-probabilistic binary linear classifier
- Find the hyperplane which maximizes the margins
- Samples on the margin are called support vectors
- Soft margins can be used (with slack variables)
- Nonlinear classification can be achieved through the **kernel trick** (mapping inputs into high dimensional feature spaces)



Limitations of Traditional HMMs

- Typically trained in non-discriminative manner
- Based on explicit feature representations and lack the power of kernel-based methods
- The conditional independence assumption is often too restrictive

Advantages of HM-SVMs

- Discriminative approach to modeling
- Can account for overlapping features (labels can depend directly on features of past or future observations)
- Maximum margin principle
- Kernel-centric approach to learning nonlinear discriminant functions

Inherited from HMMs:

- Markov chain dependency structure between labels
- Efficient dynamic programming formulation

Input-Output Mappings via Joint Feature Functions

$$f(\mathbf{x}) = \arg\max_{\mathbf{y} \in \mathcal{Y}} F(\mathbf{x}, \mathbf{y}; \mathbf{w})$$

$$F(\mathbf{x}, \mathbf{y}; \mathbf{w}) = \langle \mathbf{w}, \Phi(\mathbf{x}, \mathbf{y}) \rangle$$ ⇨ discriminant function

$$K((\mathbf{x}, \mathbf{y}), (\bar{\mathbf{x}}, \bar{\mathbf{y}})) = \langle \Phi(\mathbf{x}, \mathbf{y}), \Phi(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \rangle$$ ⇨ kernel trick

Key idea: extract features not only from the input pattern (as in binary classification), but also jointly from input-output pairs

Hidden Markov Chain Discriminants

Problem description

$$\mathbf{x} = (x^1, x^2, \ldots, x^t, \ldots)$$
$$\mathbf{y} = (y^1, y^2, \ldots, y^t, \ldots)$$
$$y^t \in \Sigma$$
$$\mathcal{X} \equiv \{(\mathbf{x}_i, \mathbf{y}_i) : i = 1, \ldots, n\}$$

Feature representation

$$\phi^{st}_{r\sigma}(\mathbf{x}, \mathbf{y}) = [\![y^t = \sigma]\!]\psi_r(x^s), \; 1 \leq r \leq d, \; \sigma \in \Sigma$$
$$\bar{\phi}^{st}_{\sigma\tau} = [\![y^s = \sigma \wedge y^t = \tau]\!], \quad \sigma, \tau \in \Sigma$$
$$\Phi(\mathbf{x}, \mathbf{y}) = \sum_{t=1}^{T} \Phi(\mathbf{x}, \mathbf{y}; t)$$

Hidden Markov Chain Discriminants

$$\phi^{st}_{r\sigma}(\mathbf{x},\mathbf{y}) = [\![y^t = \sigma]\!]\psi_r(x^s)\,,\ 1 \le r \le d,\ \sigma \in \Sigma$$

$$\bar{\phi}^{st}_{\sigma\tau} = [\![y^s = \sigma \wedge y^t = \tau]\!]\,,\quad \sigma,\tau \in \Sigma$$

$$\Phi(\mathbf{x},\mathbf{y}) = \sum_{t=1}^{T} \Phi(\mathbf{x},\mathbf{y};t)$$

In HMMs, we use only $\phi^{tt}_{r\sigma}$ and $\bar{\phi}^{t(t+1)}_{\sigma\tau}$

POS tagging example:

- $\psi_r(x^s)$ denotes the input feature of "rain" occurring at position *s*
- $[\![y^t = \sigma]\!]$ encodes whether the word at *t* is a noun or not
- $\phi^{st}_{r\sigma}$ = 1 indicates the conjunction of these two predicates (a sequence where the word at *s* is "rain" and the word at *t* is a noun)

Hidden Markov Chain Discriminants

$$\phi^{st}_{r\sigma}(\mathbf{x},\mathbf{y}) = [\![y^t = \sigma]\!]\psi_r(x^s)\,,\; 1 \le r \le d,\; \sigma \in \Sigma$$

$$\bar{\phi}^{st}_{\sigma\tau} = [\![y^s = \sigma \wedge y^t = \tau]\!]\,, \quad \sigma,\tau \in \Sigma$$

$$\Phi(\mathbf{x},\mathbf{y}) = \sum_{t=1}^{T} \Phi(\mathbf{x},\mathbf{y};t)$$

Rewriting the inner product between feature vectors for different sequences:

$$\langle \Phi(\mathbf{x},\mathbf{y}), \Phi(\bar{\mathbf{x}},\bar{\mathbf{y}})\rangle = \sum_{s,t} [\![y^{s-1} = \bar{y}^{t-1} \wedge y^s = \bar{y}^t]\!] + \sum_{s,t} [\![y^s = \bar{y}^t]\!] k(x^s, \bar{x}^t),$$

$$k(x^s, \bar{x}^t) = \langle \Psi(x^s), \Psi(\bar{x}^t)\rangle$$

The similarity between sequences depends on the number of **common two-label fragments** and on the inner product between the **feature representation of patterns with common labels**.

Structured Perceptron Learning

- $w^1 = 0$
- For t = 1 ….
  - Receive example (x,y)
  - If $h(x|w^t) = y$
    - $w^{t+1} = w^t$
  - Else
    - $w^{t+1} = w^t + \Psi(y,x)$

Only thing that changes!

$$h(x) = \text{argmax}_{y'} w^T \Psi(y', x)$$

Training Set:

$S = \{(x_i, y_i)\}$
y_i structured

Go through training set
in arbitrary order
(e.g., randomly)

Hidden Markov Perceptron Learning

To avoid explicit evaluation of feature maps and direct representation of the discriminant function, we derive the **dual of the perceptron algorithm**:

$$F(\mathbf{x},\mathbf{y}) = \sum_i \sum_{\bar{\mathbf{y}}} \alpha_i(\bar{\mathbf{y}}) \langle \Phi(\mathbf{x}_i, \bar{\mathbf{y}}), \Phi(\mathbf{x},\mathbf{y}) \rangle$$

Decompose F into two contributions: $F(\mathbf{x},\mathbf{y}) = F_1(\mathbf{x},\mathbf{y}) + F_2(\mathbf{x},\mathbf{y})$

$$F_1(\mathbf{x},\mathbf{y}) = \sum_{\sigma,\tau} \delta(\sigma,\tau) \sum_s [\![y^{s-1}=\sigma \wedge y^s=\tau]\!] \,,$$

$$\boxed{\delta(\sigma,\tau) = \sum_{i,\bar{\mathbf{y}}} \alpha_i(\bar{\mathbf{y}}) \sum_t [\![\bar{y}^{t-1} = \sigma \wedge \bar{y}^t = \tau]\!]}$$

Transition matrix

$$F_2(\mathbf{x},\mathbf{y}) = \sum_{s,\sigma} [\![y^s = \sigma]\!] \boxed{\sum_{i,t} \beta(i,t,\sigma) k(x^s, x_i^t),}$$

Emission matrix

Viterbi

$$\beta(i,t,\sigma) = \sum_{\mathbf{y}} [\![y^t = \sigma]\!] \, \alpha_i(\mathbf{y}) \,.$$

Hidden Markov Perceptron Learning

Algorithm 1 Dual perceptron algorithm for learning via joint feature functions (naive implementation).

initialize all $\alpha_i(\mathbf{y}) = 0$

repeat

  for all training patterns $\mathbf{x}_i$ **do**

    compute $\hat{\mathbf{y}}_i = \arg\max_{\mathbf{y} \in \mathcal{Y}} F(\mathbf{x}_i, \mathbf{y})$, where $F(\mathbf{x}_i, \mathbf{y}) = \sum_j \sum_{\bar{\mathbf{y}}} \alpha_j(\bar{\mathbf{y}}) \langle \Phi(\mathbf{x}_i, \mathbf{y}), \Phi(\mathbf{x}_j, \bar{\mathbf{y}}) \rangle$

Viterbi decoding

    if $\mathbf{y}_i \neq \hat{\mathbf{y}}_i$ **then**

      $\alpha_i(\mathbf{y}_i) \leftarrow \alpha_i(\mathbf{y}_i) + 1$

      $\alpha_i(\hat{\mathbf{y}}_i) \leftarrow \alpha_i(\hat{\mathbf{y}}_i) - 1$

    end if

“Perceptron-style” update

  end for

until no more errors

Hidden Markov SVM

Define the **margin** of a training example with respect to F:

$$\gamma_i = F(\mathbf{x}_i, \mathbf{y}_i) - \max_{\mathbf{y} \neq \mathbf{y}_i} F(\mathbf{x}_i, \mathbf{y})$$

We want to find the weight vector **w** which maximizes $\min_i \gamma_i$.

Add constraint to prevent data points from falling into the margins: $\max_i \gamma_i \geq 1$

We get an optimization problem with a quadratic objective:

$$\min \frac{1}{2}\|\mathbf{w}\|^2, \text{ s.t. } F(\mathbf{x}_i, \mathbf{y}_i) - \max_{\mathbf{y} \neq \mathbf{y}_i} F(\mathbf{x}_i, \mathbf{y}) \geq 1, \forall i.$$

Hidden Markov SVM

$$\min \frac{1}{2}\|\mathbf{w}\|^2, \text{ s.t. } F(\mathbf{x}_i, \mathbf{y}_i) - \max_{\mathbf{y} \neq \mathbf{y}_i} F(\mathbf{x}_i, \mathbf{y}) \geq 1, \forall i.$$

Replace each linear constraint with an equivalent set of linear constraints:

$$F(\mathbf{x}_i, \mathbf{y}_i) - F(\mathbf{x}_i, \mathbf{y}) \geq 1, \ \forall i \text{ and } \forall \mathbf{y} \neq \mathbf{y}_i$$

Rewrite constraints by introducing an additional threshold theta for every example:

$$z_i(\mathbf{y})\left(F(\mathbf{x}_i, \mathbf{y}) + \theta_i\right) \geq \frac{1}{2}, \ z_i(\mathbf{y}) = \begin{cases} 1 & \text{if } \mathbf{y} = \mathbf{y}_i \\ -1 & \text{otherwise.} \end{cases}$$

Obtain dual formulation:

$$\begin{aligned} \max W(\alpha) &= -\frac{1}{2} \sum_{i,\mathbf{y}} \sum_{j,\bar{\mathbf{y}}} \alpha_i(\mathbf{y}) \alpha_j(\bar{\mathbf{y}}) z_i(\mathbf{y}) z_j(\bar{\mathbf{y}}) k_{i,j}(\mathbf{y}, \bar{\mathbf{y}}) \\ &\quad + \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y}) \\ \text{s.t.} \quad & \alpha_i(\mathbf{y}) \geq 0, \quad \forall i = 1, \dots, n, \quad \forall \mathbf{y} \in \mathcal{Y} \\ & \sum_{\mathbf{y} \in \mathcal{Y}} z_i(\mathbf{y}) \alpha_i(\mathbf{y}) = 0, \forall i = 1, \dots, n \end{aligned}$$

$$k_{i,j}(\mathbf{y}, \bar{\mathbf{y}}) = \langle \Phi(\mathbf{x}_i, \mathbf{y}), \Phi(\mathbf{x}_j, \bar{\mathbf{y}}) \rangle$$

HM-SVM Optimization Algorithm

- Although we have a large set of possible label sequences, the actual solution might be extremely sparse (only a few negative pseudo-examples will become support vectors)
- We want to design an algorithm that exploits the anticipated **sparseness** of the solution
- Optimize W iteratively: at each iteration, optimize over the subspace spanned by all alpha_i(y) for a fixed i (i-th subspace)
- Use a **working set** approach to optimize over the i-th subspace, adding at most one negative pseudo-example to the working set at a time

HM-SVM Optimization Algorithm

Lemma 1. *If α^* is a solution of the Lagrangian dual problem in Eq. (16), then $\alpha_i^*(\mathbf{y}) = 0$ for all pairs $(\mathbf{x}_i, \mathbf{y})$ for which $F(\mathbf{x}_i, \mathbf{y}; \alpha^*) < \max_{\bar{\mathbf{y}} \neq \mathbf{y}_i} F(\mathbf{x}_i, \bar{\mathbf{y}}; \alpha^*)$.*

Proposition 2. *Assume a working set $S \subseteq \mathcal{Y}$ with $\mathbf{y}_i \in S$ is given, and that a solution for the working set has been obtained, i.e. $\alpha_i(\mathbf{y})$ with $\mathbf{y} \in S$ maximize the objective W_i subject to the constraints that $\alpha_i(\mathbf{y}) = 0$ for all $\mathbf{y} \notin S$. If there exists a negative pseudo-example $(\mathbf{x}_i, \hat{\mathbf{y}})$ with $\hat{\mathbf{y}} \notin S$ such that $-F(\mathbf{x}_i, \hat{\mathbf{y}}) - \theta_i < \frac{1}{2}$, then adding $\hat{\mathbf{y}}$ to the working set $S' \equiv S \cup \{\hat{\mathbf{y}}\}$ and optimizing over S' subject to $\alpha_i(\mathbf{y}) = 0$ for $\mathbf{y} \notin S'$ yields a strict improvement of the objective function.*

Objective for the i-th subspace, to be maximized over the alpha_i while keeping all other alpha_j fixed:

$$W_i(\alpha_i; \{\alpha_j : j \neq i\})$$

HM-SVM Optimization Algorithm

Algorithm 2 Working set optimization for HM-SVMs.

$S \leftarrow \{\mathbf{y}_i\}$, $\alpha_i = 0$
loop
  compute $\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \neq \mathbf{y}_i} F(\mathbf{x}_i, \mathbf{y}; \alpha)$
  if $F(\mathbf{x}_i, \mathbf{y}_i; \alpha) - F(\mathbf{x}_i, \hat{\mathbf{y}}; \alpha) \geq 1$ **then**
    return α_i
  else
    $S \leftarrow S \cup \{\hat{\mathbf{y}}\}$
    $\alpha_i \leftarrow$ optimize W_i over S
  end if
  for $\mathbf{y} \in S$ **do**
    if $\alpha_i(\mathbf{y}) = 0$ **then**
      $S \leftarrow S - \{\mathbf{y}\}$
    end if
  end for
end loop

Initialize working set

Viterbi decoding

Return current solution when constraint is broken

Add negative pseudo-example to working set and optimize in the i-th subspace

Remove from the working set the sequences for which alpha_i is zero

Soft Margin HM-SVM

- In the non-separable case, we can introduce **slack variables** to allow margin violations

$$\begin{aligned} \min \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i \\ \text{s.t.} \quad & z_i(\mathbf{y})(\langle \mathbf{w}, \Phi(\mathbf{x}_i, \mathbf{y})\rangle + \theta_i) \geq 1 - \xi_i, \quad \xi_i \geq 0 \\ & \forall i = 1, \ldots, n, \quad \forall \mathbf{y} \in \mathcal{Y} \end{aligned}$$

Lagrangian:

$$\begin{aligned} L \quad = \quad & \frac{1}{2}\|\mathbf{w}\|^2 + \sum_i (C - \rho_i)\xi_i \\ & - \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y}) \left[z_i(\mathbf{y}) \left(F(\mathbf{x}_i, \mathbf{y}) + \theta_i \right) - 1 + \xi_i \right] \end{aligned}$$

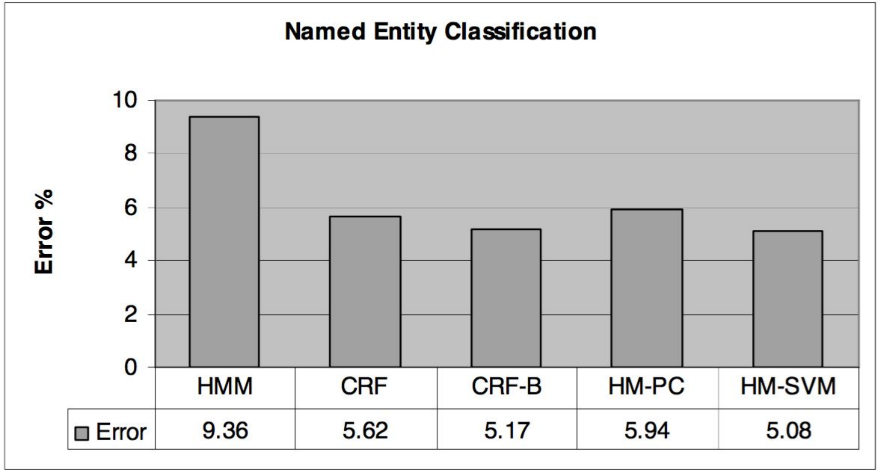
- Use same working set approach from Algorithm 2, but with different constraints in the quadratic optimization (step 8)

Results for Named Entity Classification

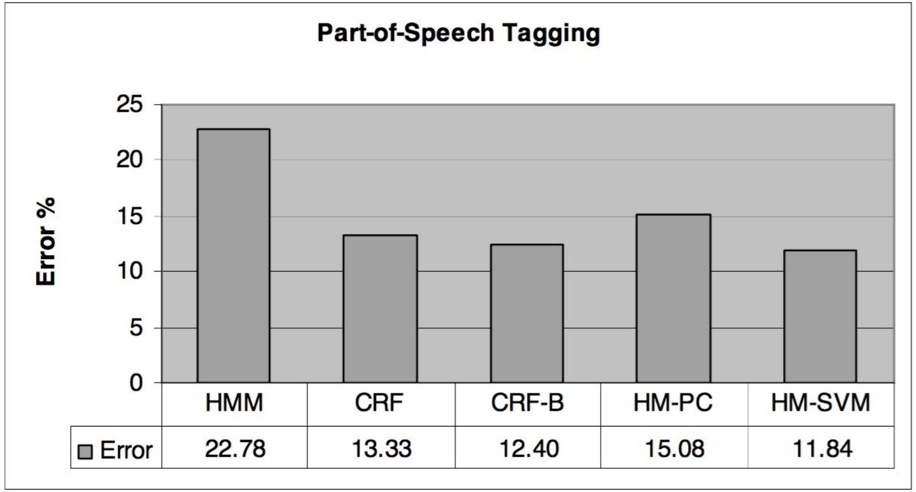
```
PP ESTUDIA YA PROYECTO LEY TV REGIONAL REMITIDO
O  N       N  N        M   m  m        N

POR LA JUNTA Merida ( EFE ) .
N   N  O     L        N  O  N N

ONNNMmmNNNOLNONN
--------M-------
------N---------
----------P-----
----N-----------
N---P-----------
-------m--------
-----------o----
```



Results for Part-of-Speech Tagging



Large Margin Hidden Markov Models for Automatic Speech Recognition

F Sha and L K Saul (NIPS 2007)

What are we trying to do?

- Infer correct hidden state sequence $\boldsymbol{y} = [y_1,y_2,\ldots,y_T]$ given observation sequence $\boldsymbol{X} = [\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_T]$
- In automatic speech recognition (ASR), **y** can be words, phonemes, etc. In this instance **y** is a set of 48 phonetic classes, each represented by a state in the HMM
- **X** is 39-dimensional real-valued acoustic feature vector (MFCCs)
- Continuous density is needed to model emissions (we will use gaussian mixture models)

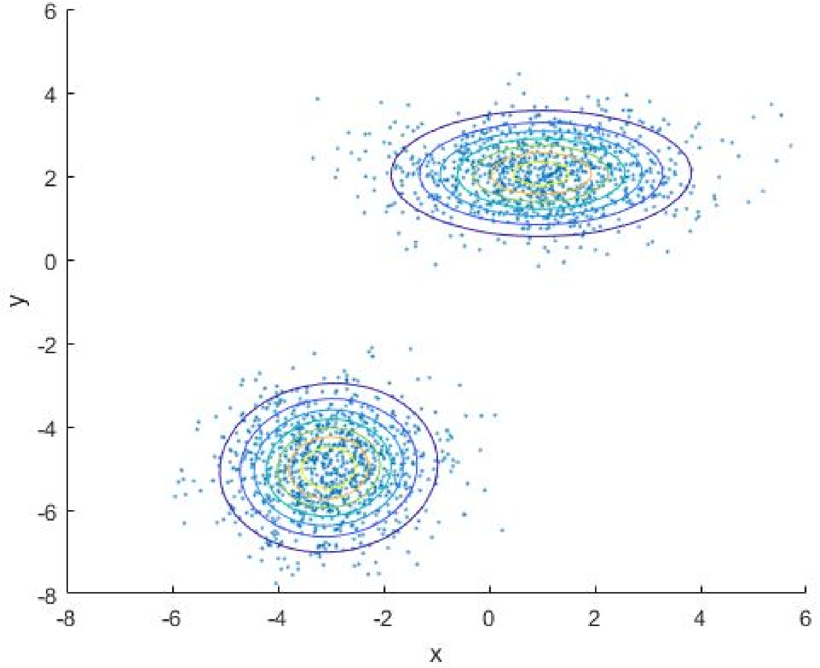
GMMs for multiway classification

General form of mixture model

$$p(x) = \sum_{i=0}^{k} \pi_i f_i(x)$$

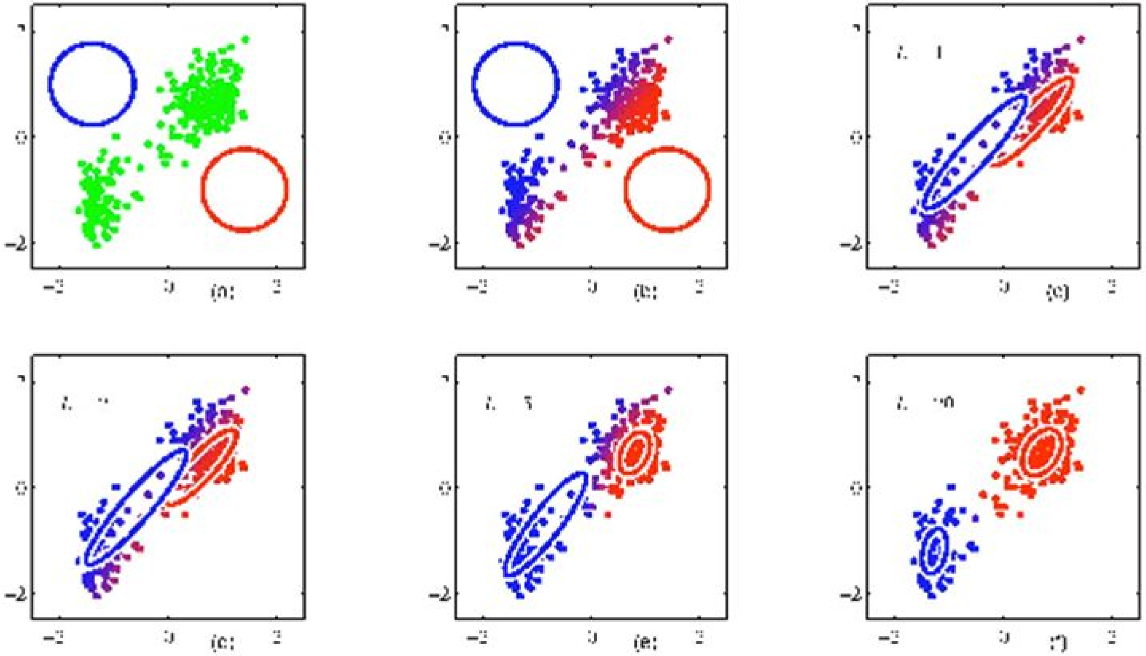
Gaussian mixture model

$$p(x) = \sum_{i=0}^{k} \pi_i N(x|\mu_k, \Sigma_k)$$



Learning Parameters for GMM

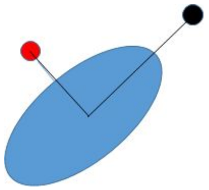
- Initialize parameters $\boldsymbol{\theta} = \{\boldsymbol{\tau},\boldsymbol{\mu}_1,\boldsymbol{\mu}_2,\boldsymbol{\Sigma}_1,\boldsymbol{\Sigma}_2\}$
- Given current parameters, compute membership probability (i.e. soft clustering) for each data point (E-step)
- Adjust $\boldsymbol{\theta}$, such that it best explains the points assigned to each cluster (M-step)



Large Margin GMMs

- In EM, we seek to maximize the joint likelihood of observed feature vectors and label sequences
- This however does not minimize phoneme or word error rates, which are more relevant for automatic speech recognition
- Unlike EM, we seek to maximize the distance between labeled examples
- Decision rule for single ellipsoid (i.e. N($\boldsymbol{\mu}_i$,$\boldsymbol{\Sigma}_i$))

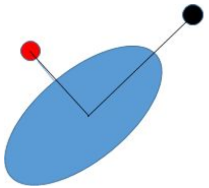
$$y = \underset{c}{\operatorname{argmin}} \left\{ (\mathbf{x}-\boldsymbol{\mu}_c)^{\mathrm{T}} \boldsymbol{\Psi}_c (\mathbf{x}-\boldsymbol{\mu}_c) + \theta_c \right\}$$



Large Margin GMMs

- Decision rule for single ellipsoid (i.e. N($\boldsymbol{\mu}_i$,$\boldsymbol{\Sigma}_i$))

$$y = \operatorname*{argmin}_c \left\{ (\mathbf{x} - \boldsymbol{\mu}_c)^{\mathrm{T}} \boldsymbol{\Psi}_c (\mathbf{x} - \boldsymbol{\mu}_c) + \theta_c \right\}$$



which can be reformulated as

$$y = \operatorname*{argmin}_c \left\{ \boldsymbol{z}^{\mathrm{T}} \boldsymbol{\Phi}_c \, \boldsymbol{z} \right\}$$

$$\boldsymbol{\Phi}_c = \begin{bmatrix} \boldsymbol{\Psi}_c & -\boldsymbol{\Psi}_c \boldsymbol{\mu}_c \\ -\boldsymbol{\mu}_c^{\mathrm{T}} \boldsymbol{\Psi}_c & \boldsymbol{\mu}_c^{\mathrm{T}} \boldsymbol{\Psi}_c \boldsymbol{\mu}_c + \theta_c \end{bmatrix} \qquad \boldsymbol{z} = \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix}$$

Large Margin GMMs

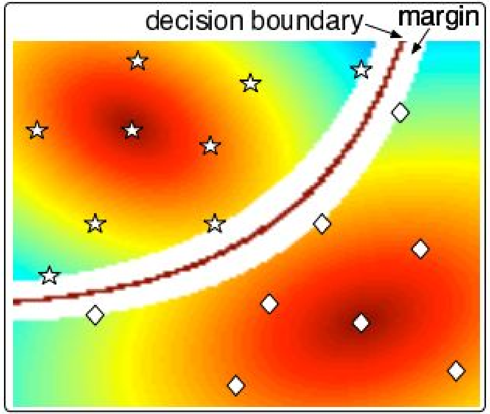
- Hard-margin maximization for single ellipsoid per class

$$
\begin{array}{ll}
\min & \sum_c \operatorname{trace}(\boldsymbol{\Psi}_c) \\
\text{s.t.} & 1 + \boldsymbol{z}_n^{\mathrm{T}}(\boldsymbol{\Phi}_{y_n} - \boldsymbol{\Phi}_c)\boldsymbol{z}_n \leq 0, \quad \forall c \neq y_n, n = 1, 2, \ldots, N \\
& \boldsymbol{\Phi}_c \succ 0, \quad c = 1, 2, \ldots, C
\end{array}
$$

Regularizes the scale of ellipsoids

Enforces margin condition

Restricts the matrix to be positive semidefinite



Large Margin GMMs

- Soft-margin (i.e. with slack variables)

$$
\begin{array}{ll}
\min & \sum_{nc} \xi_{nc} + \gamma \sum_{c} \operatorname{trace}(\boldsymbol{\Psi}_c) \\
\text{s.t.} & 1 + \boldsymbol{z}_n^{\mathrm{T}} (\boldsymbol{\Phi}_{y_n} - \boldsymbol{\Phi}_c) \boldsymbol{z}_n \leq \xi_{nc}, \\
& \xi_{nc} \geq 0, \quad \forall c \neq y_n, n = 1, 2, \ldots, N \\
& \boldsymbol{\Phi}_c \succ 0, \quad c = 1, 2, \ldots, C
\end{array}
$$

Large Margin GMMs

- How does this margin maximization criteria generalize to case where each class is modeled as a mixture?
- Generate a "proxy label" for each data point ($\mathbf{x}_n$,y_n,m_n), where m_n represents the mixture component label

$$
\begin{aligned}
\min \quad & \textstyle\sum_{nc} \xi_{nc} + \gamma \sum_{cm} \mathrm{trace}(\mathbf{\Psi}_{cm}) \\
\text{s.t.} \quad & 1 + \boldsymbol{z}_n^{\mathrm{T}} \mathbf{\Phi}_{y_n m_n} \boldsymbol{z}_n + \log \textstyle\sum_m e^{-\boldsymbol{z}_n^{\mathrm{T}} \mathbf{\Phi}_{cm} \boldsymbol{z}_n} \leq \xi_{nc}, \\
& \xi_{nc} \geq 0, \quad \forall c \neq y_n, n = 1, 2, \ldots, N \\
& \mathbf{\Phi}_{cm} \succ 0, \quad c = 1, 2, \ldots, C, \; m = 1, 2, \ldots, M
\end{aligned}
$$

Sequential classification with CD-HMMs

- **Reminder:** HMM states are phonemes, observations are low-level spectral features of the recording
- Model emission densities with gaussian mixture models
- Compute a score over a sequence of observations and states (note that number of incorrect sequences grows as $O(C^T)$)

$$\mathcal{D}(\boldsymbol{X}, \boldsymbol{s}) = \sum_t \log a(s_{t-1}, s_t) - \sum_{t=1}^{T} \boldsymbol{z}_t^{\mathrm{T}} \boldsymbol{\Phi}_{s_t} \boldsymbol{z}_t$$

- We can then define our margin constraints as

$$\forall \boldsymbol{s} \neq \boldsymbol{y}, \quad \mathcal{D}(\boldsymbol{X}, \boldsymbol{y}) - \mathcal{D}(\boldsymbol{X}, \boldsymbol{s}) \geq \boxed{\mathcal{H}(\boldsymbol{s}, \boldsymbol{y})}$$

Hamming Distance

Sequential classification with CD-HMMs

- Number of constraints grows exponentially with the sequence length, there is 1 constraint for each incorrect sequence ***s***

$$\forall \boldsymbol{s} \neq \boldsymbol{y}, \quad \mathcal{D}(\boldsymbol{X}, \boldsymbol{y}) - \mathcal{D}(\boldsymbol{X}, \boldsymbol{s}) \geq \mathcal{H}(\boldsymbol{s}, \boldsymbol{y})$$

$$-\mathcal{D}(\boldsymbol{X}, \boldsymbol{y}) + \max_{\boldsymbol{s} \neq \boldsymbol{y}} \{\mathcal{H}(\boldsymbol{s}, \boldsymbol{y}) + \mathcal{D}(\boldsymbol{X}, \boldsymbol{s})\} \leq 0$$

i.e. log-likelihood of target sequence must be at least as good as next best one + handicap

- Collapse the constraints

$$-\mathcal{D}(\boldsymbol{X}, \boldsymbol{y}) + \log \sum_{\boldsymbol{s} \neq \boldsymbol{y}} e^{\mathcal{H}(\boldsymbol{s}, \boldsymbol{y}) + \mathcal{D}(\boldsymbol{X}, \boldsymbol{s})} \leq 0$$

Softmax upper bound (why? differentiable with respect to model params)

Sequential classification with CD-HMMs

- Full convex optimization problem:

$$
\begin{array}{ll}
\min & \sum_n \xi_n + \gamma \sum_{cm} \mathrm{trace}(\mathbf{\Psi}_{cm}) \\
\text{s.t.} & -\mathcal{D}(\boldsymbol{X}_n, \boldsymbol{y}_n) + \log \sum_{\boldsymbol{s} \neq \boldsymbol{y}_n} e^{\mathcal{H}(\boldsymbol{s}, \boldsymbol{y}_n) + \mathcal{D}(\boldsymbol{X}_n, \boldsymbol{s})} \leq \xi_n, \\
& \xi_n \geq 0, \quad n = 1, 2, \ldots, N \\
& \mathbf{\Phi}_{cm} \succ 0, \quad c = 1, 2, \ldots, C, m = 1, 2, \ldots, M
\end{array}
$$

Experiments

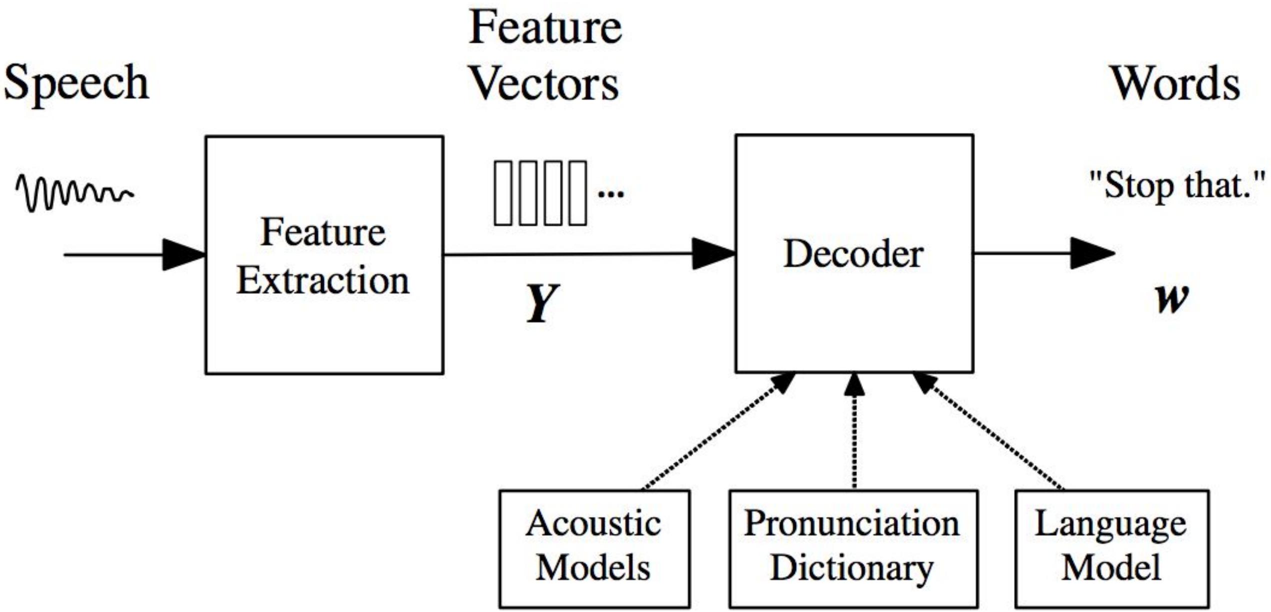
- Used TIMIT speech corpus for phonetic recognition
- Error rate using hamming distance, compared to EM baseline
- Utterance-based training is better than frame-based training

mixture (per state)	baseline (EM)	margin (frame)	margin (utterance)
1	45%	37%	30%
2	45%	36%	29%
4	42%	35%	28%
8	41%	34%	27%

Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition

G Dahl, D Yu, L Deng and A Acero (2012)

HMM for Speech Recognition



CD-DNN-HMM

- Key Concepts
  - **Context-dependent states in HMM**
  - **Acoustic model as a deep belief network**
    - **Using restricted boltzmann machines**
  - **Pre-training of deep neural network**
  - **Deep neural network HMM hybrid acoustic model**

Let's have a look at what these things mean!

Context Dependence

- Large vocabulary systems do not use words as units of sound
  - Vocabularies can consist of tens of thousands of words
  - It's difficult to find enough examples of every word even in large training datasets
  - Words not seen in training cannot be learned
- Use sub-word units
  - There are many more instances of sub-word units in a corpus than of words and therefore HMM parameters can be better estimated
  - Sub-word units can be combined to form new words
  - Usually called phones

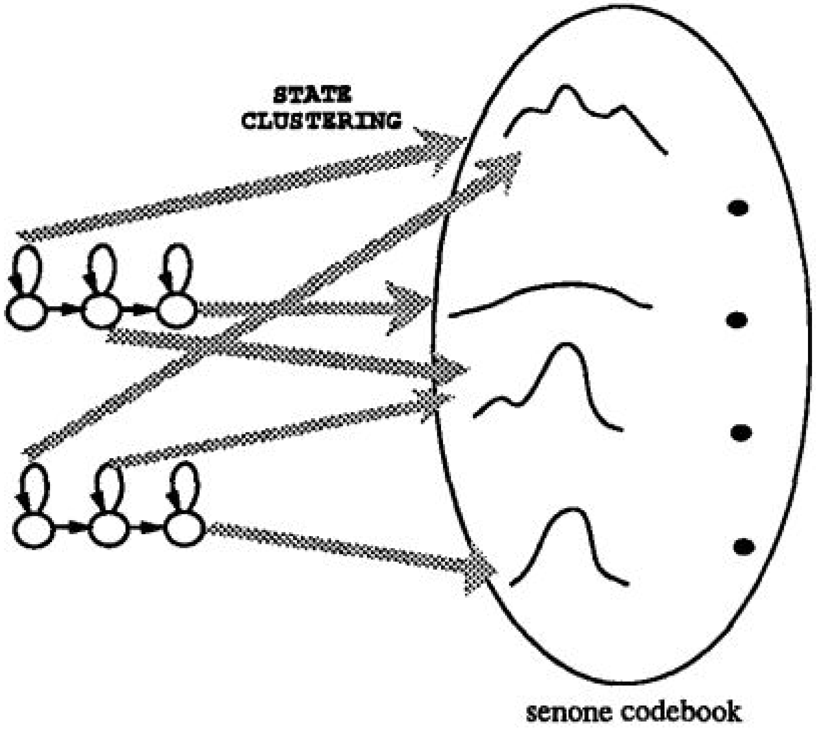
Context Dependence

- Consider the word ROCK for example. Phonetically, we can write that as R-AO-K
- An HMM where states are *context-independent* phonemes is plausible
- Phonemes are however very coarse units
  - When /AO/ is preceded by /R/ and followed by /K/, it has a different spectral signature than when it is preceded by /B/ and followed by /L/ as in the word ball
- We try to capture this variability, by considering phonemes *in context*

Word	Phones	Triphones
Rock	R AO K	R,AO(R,K),K

Context Dependence

- Number of triphones can be very large
- Realizing the amount of overlap between triphones, can we create a “codebook” by clustering triphone **states** that are similar?
- Each cluster called a **senone**
- In the model under consideration, these are the HMM states

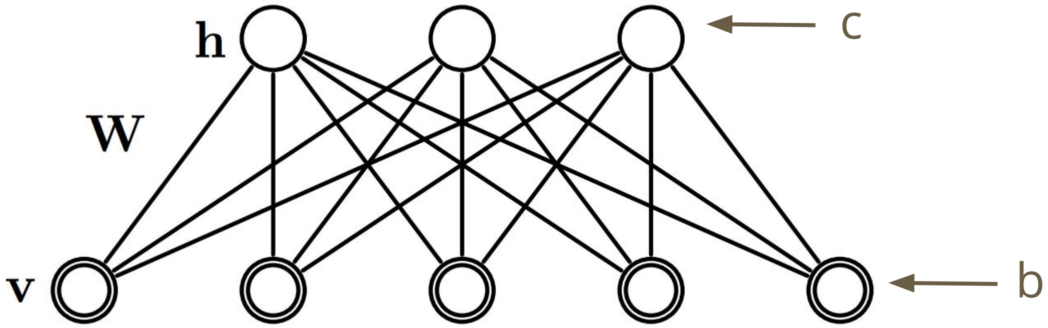


CD-DNN-HMM

- Key Concepts
  - **Context-dependent states in HMM**
  - **Acoustic model as a deep belief network**
    - **Using restricted boltzmann machines**
  - **Pre-training of deep neural network**
  - **Deep neural network HMM hybrid acoustic model**

Restricted Boltzmann Machines

- Undirected graphical model, where **v** = visible units (our data) and **h =** the hidden units



$$\boxed{E(\mathbf{v},\mathbf{h}) = -\mathbf{b}^{\mathrm{T}}\mathbf{v} - \mathbf{c}^{\mathrm{T}}\mathbf{h} - \mathbf{v}^{\mathrm{T}}\mathbf{W}\mathbf{h}}$$

Energy for (v,h) pair, where c and b are bias terms (for binary data)

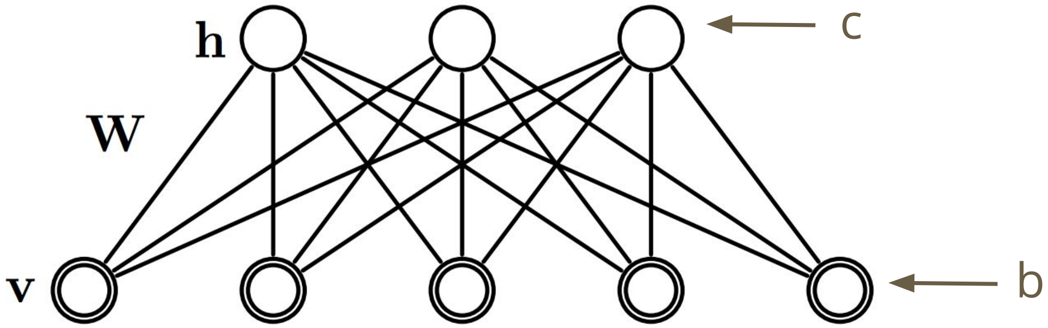
$$\boxed{P(\mathbf{v},\mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{Z}}$$

Joint probability over (v,h), where

$$Z = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}$$

Restricted Boltzmann Machines

- Undirected graphical model, where **v** = visible units (our data) and **h** = the hidden units



$$\boxed{E(\mathbf{v},\mathbf{h}) = \frac{1}{2}(\mathbf{v}-\mathbf{b})^{\mathrm{T}}(\mathbf{v}-\mathbf{b}) - \mathbf{c}^{\mathrm{T}}\mathbf{h} - \mathbf{v}^{\mathrm{T}}\mathbf{W}\mathbf{h}}$$

Energy for (v,h) pair, for real-valued feature vectors

$$\boxed{P(\mathbf{v},\mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{Z}}$$

Joint probability over (v,h), where

$$Z = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}$$

Restricted Boltzmann Machines

- We can define a per-training-case log likelihood function as

$$\boxed{\ell(\boldsymbol{\theta}) = -F(\mathbf{v}) - \log\left(\sum_{\boldsymbol{\nu}} e^{-F(\boldsymbol{\nu})}\right)}$$

perform stochastic gradient descent on this

- Where F(V) is known as the free energy and defined as

$$F(\mathbf{v}) = -\log\left(\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}\right)$$

- In practice, gradient of log likelihood of data in RBM is hard to compute, so use MCMC methods (e.g. Gibbs sampling)

Restricted Boltzmann Machines

- Because there are no intra-layer connections, given **v**, we can easily infer the distribution over hidden units (and vice versa)

$$P(\mathbf{h} = \mathbf{1} \mid \mathbf{v}) = \sigma(\mathbf{c} + \mathbf{v}^{\mathrm{T}}\mathbf{W})$$
$$P(\mathbf{v} = \mathbf{1} \mid \mathbf{h}) = \sigma(\mathbf{b} + \mathbf{h}^{\mathrm{T}}\mathbf{W}^{\mathrm{T}})$$

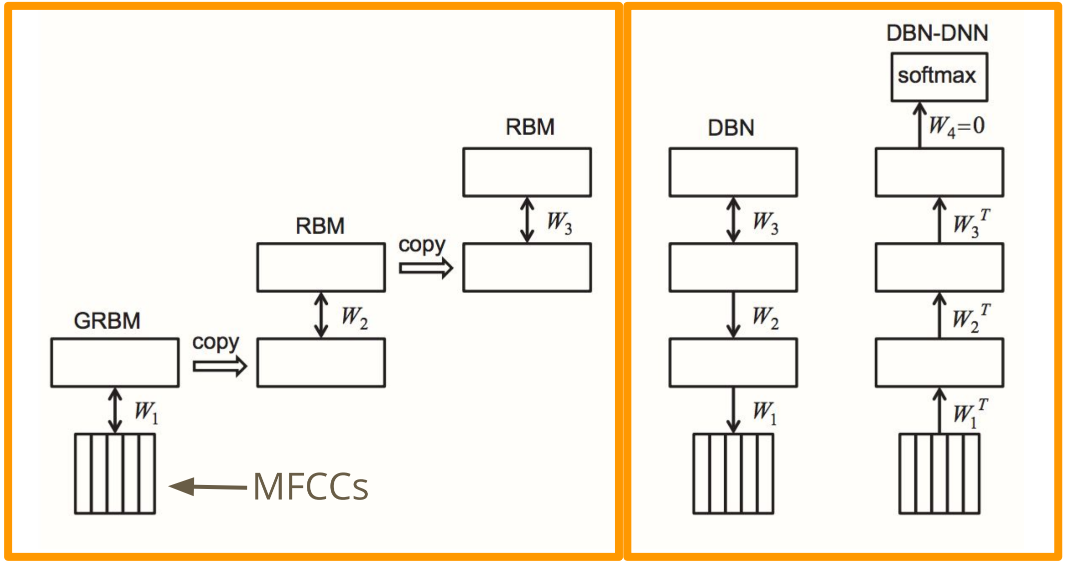
- This looks a lot like feedforward propagation in a neural network. Later this will allow us to use the weights of an RBM to initialize a feed-forward network.

CD-DNN-HMM

- Key Concepts
  - **Context-dependent states in HMM**
  - **Acoustic model as a deep belief network**
    - **Using restricted boltzmann machines**
  - **Pre-training of deep neural network** ←
  - **Deep neural network HMM hybrid acoustic model**

Pre-training a Deep Neural Network

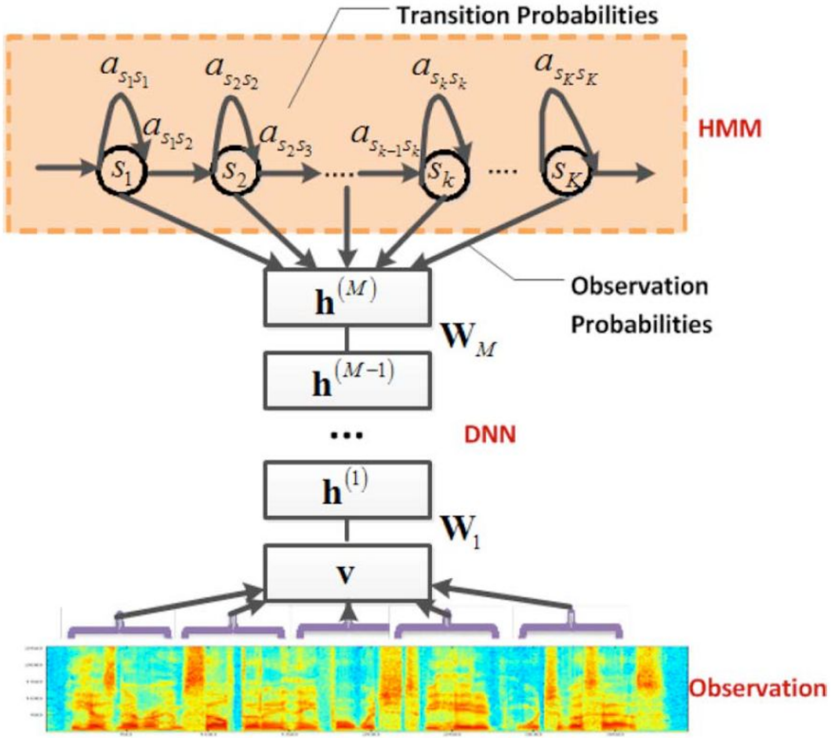
- Stack a series of RBMs
- Transfer learned weights to a feedforward deep neural network and add softmax output layer
- Refine weights of DNN with labeled data
- Output of DNN are treated as "senones"
- **Advantages:**
  - Can use large set of unsupervised data for pretraining, smaller one to further refine pre-trained DNN
  - Often achieves lower training error
  - Sort of data dependent regularization



CD-DNN-HMM

- Key Concepts
  - **Context-dependent states in HMM**
  - **Acoustic model as a deep belief network**
    - **Using restricted boltzmann machines**
  - **Pre-training of deep neural network**
  - **Deep neural network HMM hybrid acoustic model**

Model Architecture



- The decoded word sequence $\hat{w}$ is determined as

$$\hat{w} = \operatorname*{argmax}_{w} p(w \mid \mathbf{x}) = \operatorname*{argmax}_{w} p(\mathbf{x} \mid w) p(w) / p(\mathbf{x})$$

where $p(w)$ is the language model probability and the acoustic model is

$$p(\mathbf{x} \mid w) = \sum_{q} p(\mathbf{x}, q \mid w) p(q \mid w)$$

$$\cong \max \pi(q_0) \prod_{t=1}^{T} a_{q_{t-1}q_t} \prod_{t=0}^{T} p(\mathbf{x}_t \mid q_t)$$

Experimental Results

- Bing mobile voice search application: ex. “Mcdonald's”,”Denny’s restaurant”
- Sampled at 8kHz
- Collected under real usage scenarios, so contains all kinds of variations such as noise, music, side-speech, accents, sloppy pronunciation
- Language Model: 65K word unigrams, 3.2 million word bi-grams, and 1.5 million word trigrams
- Sentence length is 2.1 tokens

	Hours	Number of Utterances
Training Set	24	32,057
Development Set	6.5	8,777
Test Set	9.5	12,758

Experimental Results

- They computed sentence accuracy instead of word accuracy
  - Difficulties with word accuracy
    - “Mc-Donalds”, “McDonalds”
    - “Walmart”, “Wal-Mart”
    - “7-eleven”, “7 eleven”, “seven-eleven”
  - Users only care if find the business or not, so the will repeat whole phrase if one if the words is not recognized
- Maximum 94% accuracy

Experimental Results

- Baseline Systems
  - Performance of best CD-GMM-HMM summarized in table below

TABLE II
CD-GMM-HMM BASELINE RESULTS

Criterion	Dev Accuracy	Test Accuracy	
ML	62.9%	60.4%	Maximum likelihood
MMI	65.1%	62.8%	Maximum mutual information
MPE	65.5%	63.8%	Minimum phone error

Experimental Results

- Context independent vs. context dependent state labels

TABLE IV
COMPARISON OF CONTEXT-INDEPENDENT MONOPHONE STATE LABELS AND CONTEXT-DEPENDENT TRIPHONE SENONE LABELS

# Hidden Layers	# Hidden Units	Label Type	Dev Accuracy
1	2K	Monophone States	59.3%
1	2K	Triphone Senones	68.1%
3	2K	Monophone States	64.2%
3	2K	Triphone Senones	69.6%

Experimental Results

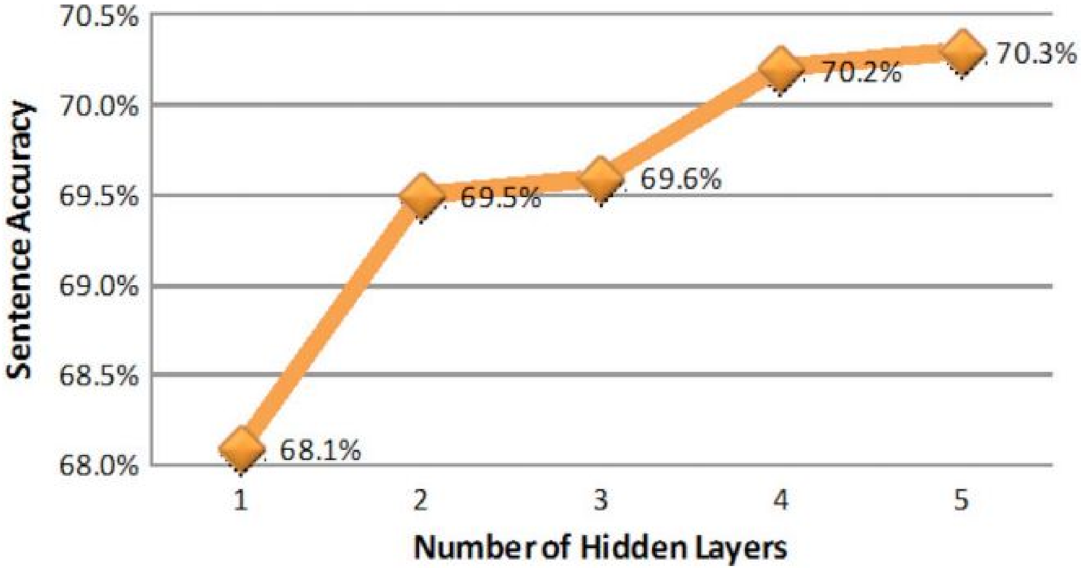
- Pre-training improves accuracy

TABLE V
CONTEXT-DEPENDENT MODELS WITH AND WITHOUT PRE-TRAINING

Model Type	# Hidden Layers	# Hidden Units	Dev Accuracy
without pre-training	1	2K	68.0%
without pre-training	2	2K	68.2%
with pre-training	1	2K	68.1%
with pre-training	2	2K	69.5%

Experimental Results

- Accuracy as a function of the number of layers in DNN



Experimental Results

- Training time

TABLE VII
SUMMARY OF TRAINING TIME USING 24 HOURS OF TRAINING DATA AND 2 K HIDDEN UNITS PER LAYER

Type	# of Layers	Time Per Epoch	# of Epochs
Pre-train	1	0.2 h	50
Pre-train	2	0.5 h	20
Pre-train	3	0.6 h	20
Pre-train	4	0.7 h	20
Pre-train	5	0.8 h	20
Fine-tune	4	1.2 h	12
Fine-tune	5	1.4 h	12

Experimental Results

- Training time
  - So, to train a 5-layer CD-DNN-HMM, pre-training takes about
    (0.2 x 50) + (0.5 x 20) + (0.6 x 20) + (0.7 x 20) + (0.8 x 20) = **62 hours**
  - Fine-tuning takes about 1.4 x 12 = **16.8 hours** (for presented results 33.6 hours)

Experimental Results

- Decoding time

TABLE VIII
SUMMARY OF DECODING TIME

Processing Unit	# of Layers	DNN Time Per Frame	Search Time Per Frame	Real-time Factor
CPU	4	4.3 ms	1.5 ms	0.58
GPU	4	0.16 ms	1.5 ms	0.17
CPU	5	5.2 ms	1.5 ms	0.67
GPU	5	0.20 ms	1.5 ms	0.17

Conclusions

- CD-DNN-HMM performs better than its rival, the CD-GMM-HMM
- It is however more computationally expensive
- Bottlenecks
  - The bottleneck in the training process is the mini-batch stochastic gradient descent (SGD) algorithm.
  - Training in the study used the embedded Viterbi algorithm, which is not optimal for MFCCs