Hidden Markov Models and Gaussian Mixture Models

Hiroshi Shimodaira and Steve Renals

Automatic Speech Recognition— ASR Lectures 4&5
23&27 January 2014

Overview

HMMs and GMMs

- Key models and algorithms for HMM acoustic models
- Gaussians
- GMMs: Gaussian mixture models
- HMMs: Hidden Markov models
- HMM algorithms
  - Likelihood computation (forward algorithm)
  - Most probable state sequence (Viterbi algorithm)
  - Estimting the parameters (EM algorithm)

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

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

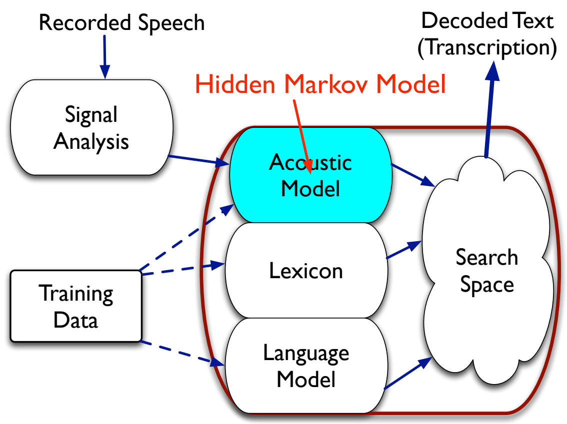
Applying Bayes' Theorem:

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

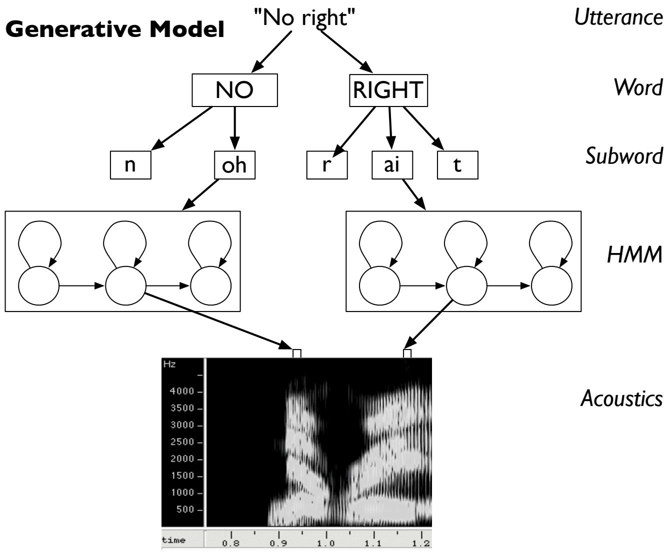
$$\propto p(\mathbf{X} \mid \mathbf{W}) P(\mathbf{W})$$

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

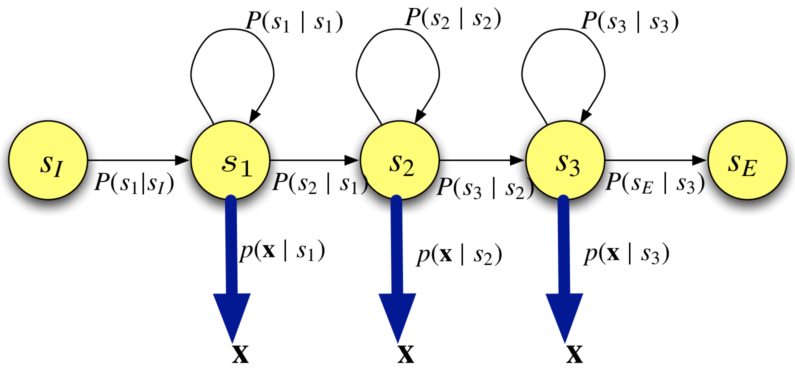
Acoustic Modelling



Hierarchical modelling of speech



Acoustic Model: Continuous Density HMM

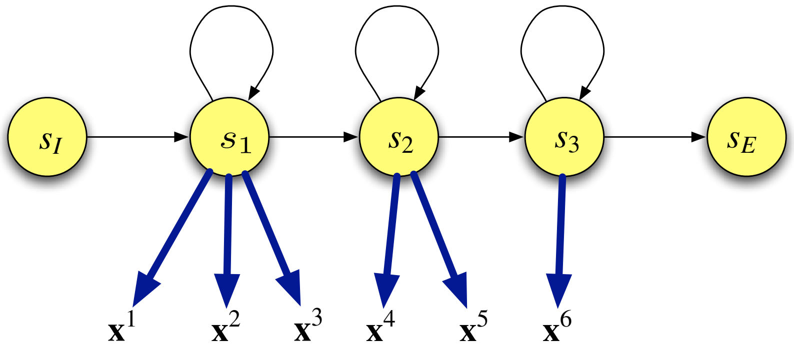


Probabilistic finite state automaton

Paramaters $\boldsymbol{\lambda}$:

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

Acoustic Model: Continuous Density HMM

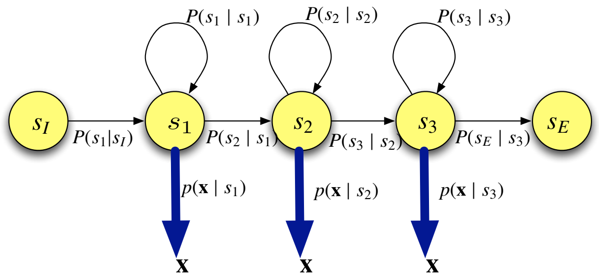


Probabilistic finite state automaton

Paramaters $\boldsymbol{\lambda}$:

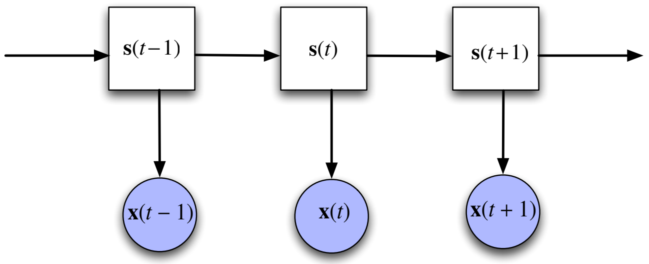
- Transition probabilities: $a_{kj} = P(S = j \mid S = k)$
- Output probability density function: $b_j(\mathbf{x}) = p(\mathbf{x} \mid S = j)$

HMM Assumptions



1. **Observation independence** An acoustic observation $\mathbf{x}$ is conditionally independent of all other observations given the state that generated it
2. **Markov process** A state is conditionally independent of all other states given the previous state

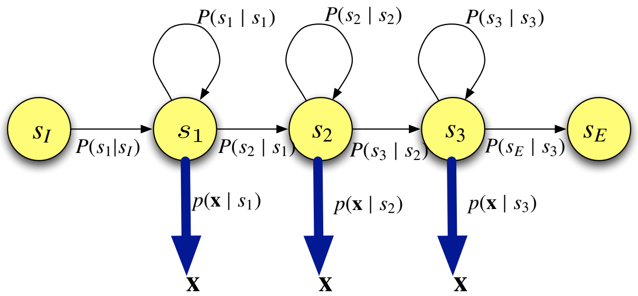
HMM Assumptions



1. **Observation independence** An acoustic observation $\mathbf{x}$ is conditionally independent of all other observations given the state that generated it
2. **Markov process** A state is conditionally independent of all other states given the previous state

HMM OUTPUT DISTRIBUTION

Output distribution



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

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

M-component Gaussian mixture model:

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

Background: cdf

Consider a real valued random variable X

- Cumulative distribution function (cdf) $F(x)$ for X:

$$F(x) = P(X \leq x)$$

- To obtain the probability of falling in an interval we can do the following:

$$\begin{aligned} P(a < X \leq b) &= P(X \leq b) - P(X \leq a) \\ &= F(b) - F(a) \end{aligned}$$

Background: pdf

- The rate of change of the cdf gives us the *probability density function* (pdf), $p(x)$:

$$p(x) = \frac{d}{dx}F(x) = F'(x)$$
$$F(x) = \int_{-\infty}^{x} p(x)\mathrm{d}x$$

- $p(x)$ is **not** the probability that X has value x. But the pdf is proportional to the probability that X lies in a small interval centred on x.
- Notation: p for pdf, P for probability

The Gaussian distribution (univariate)

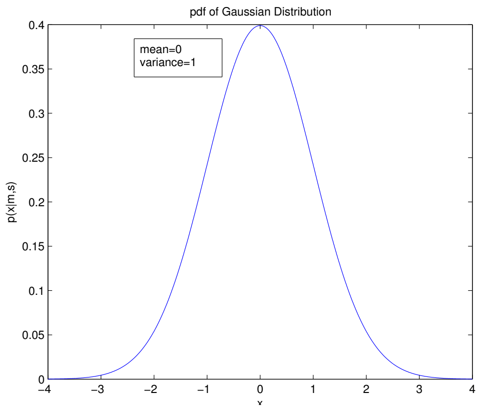
- The Gaussian (or Normal) distribution is the most common (and easily analysed) continuous distribution
- It is also a reasonable model in many situations (the famous "bell curve")
- If a (scalar) variable has a Gaussian distribution, then it has a probability density function with this form:

$$p(x|\mu, \sigma^2) = N(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)$$

- The Gaussian is described by two parameters:
  - the mean μ (location)
  - the variance σ^2 (dispersion)

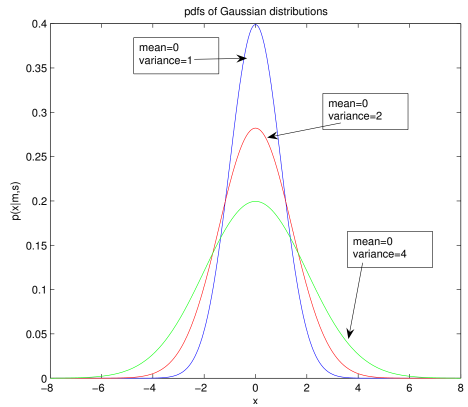
Plot of Gaussian distribution

- Gaussians have the same shape, with the location controlled by the mean, and the spread controlled by the variance
- One-dimensional Gaussian with zero mean and unit variance ($\mu = 0$, $\sigma^2 = 1$):



Properties of the Gaussian distribution

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



Parameter estimation

- Estimate mean and variance parameters of a Gaussian from data $x^{(1)}, x^{(2)}, \ldots, x^{(N)}$
- Use sample mean and sample variance estimates:

$$\mu = \frac{1}{N}\sum_{n=1}^{N} x^{(n)} \qquad \text{(sample mean)}$$

$$\sigma^2 = \frac{1}{N}\sum_{n=1}^{N} (x^{(n)} - \mu)^2 \qquad \text{(sample variance)}$$

Exercise — maximum likelihood estimation (MLE)

Consider the log likelihood of a set of N training data points $\{x^{(1)}, \ldots, x^{(N)}\}$ being generated by a Gaussian with mean μ and variance σ^2:

$$L = \ln p(\{x^{(1)}, \ldots, x^{(n)}\} \mid \mu, \sigma^2) = -\frac{1}{2}\sum_{n=1}^{N}\left(\frac{(x^{(n)} - \mu)^2}{\sigma^2} - \ln \sigma^2 - \ln(2\pi)\right)$$

$$= -\frac{1}{2\sigma^2}\sum_{n=1}^{N}(x^{(n)} - \mu)^2 - \frac{N}{2}\ln \sigma^2 - \frac{N}{2}\ln(2\pi)$$

By maximising the the log likelihood function with respect to μ show that the maximum likelihood estimate for the mean is indeed the sample mean:

$$\mu_{ML} = \frac{1}{N}\sum_{n=1}^{N} x^{(n)}.$$

The multivariate Gaussian distribution

- The d-dimensional vector $\mathbf{x} = (x_1, \ldots, x_d)^T$ follows a multivariate Gaussian (or normal) distribution if it has a probability density function of the following form:

$$p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right)$$

  The pdf is parameterized by the mean vector $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_d)^T$

  and the covariance matrix $\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_{11} & \ldots & \sigma_{1d} \\ \vdots & \ddots & \vdots \\ \sigma_{d1} & \ldots & \sigma_{dd} \end{pmatrix}$.
- The 1-dimensional Gaussian is a special case of this pdf
- The argument to the exponential $0.5(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})$ is referred to as a *quadratic form*.

Covariance matrix

- The mean vector $\boldsymbol{\mu}$ is the expectation of $\mathbf{x}$:

$$\boldsymbol{\mu} = E[\mathbf{x}]$$

- The covariance matrix $\boldsymbol{\Sigma}$ is the expectation of the deviation of $\mathbf{x}$ from the mean:

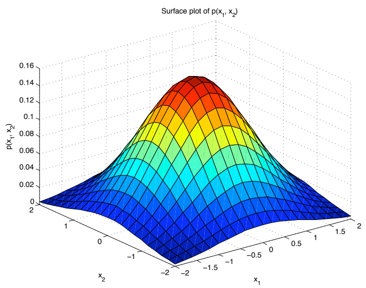
$$\boldsymbol{\Sigma} = E[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T]$$

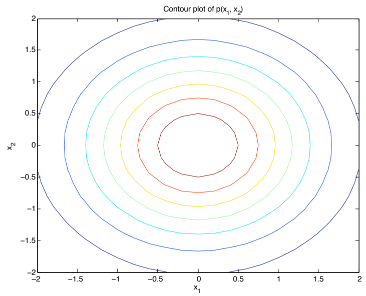
- $\boldsymbol{\Sigma}$ is a $d \times d$ symmetric matrix:

$$\sigma_{ij} = E[(x_i - \mu_i)(x_j - \mu_j)] = E[(x_j - \mu_j)(x_i - \mu_i)] = \sigma_{ji}$$

- The sign of the covariance helps to determine the relationship between two components:
  - If x_j is large when x_i is large, then $(x_i - \mu_i)(x_j - \mu_j)$ will tend to be positive;
  - If x_j is small when x_i is large, then $(x_i - \mu_i)(x_j - \mu_j)$ will tend to be negative.

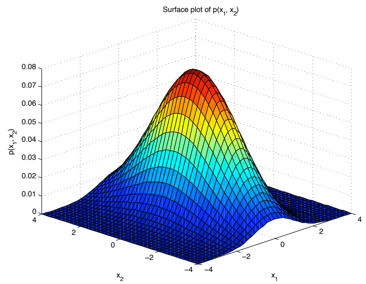
Spherical Gaussian

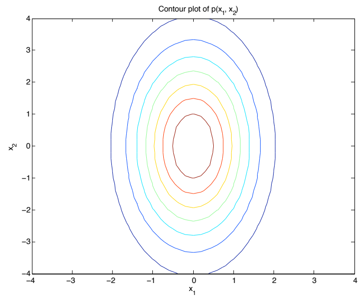




$$\boldsymbol{\mu} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad \boldsymbol{\Sigma} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad \rho_{12} = 0$$

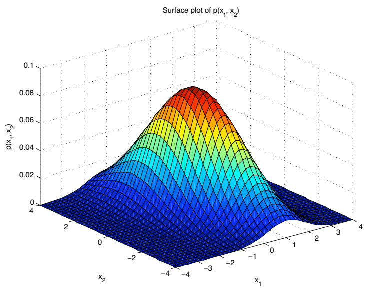
Diagonal Covariance Gaussian

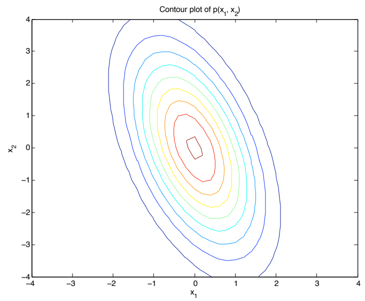




$$\boldsymbol{\mu} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad \boldsymbol{\Sigma} = \begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix} \qquad \rho_{12} = 0$$

Full covariance Gaussian





$$\boldsymbol{\mu} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad \boldsymbol{\Sigma} = \begin{pmatrix} 1 & -1 \\ -1 & 4 \end{pmatrix} \qquad \rho_{12} = -0.5$$

Parameter estimation of a multivariate Gaussian distribution

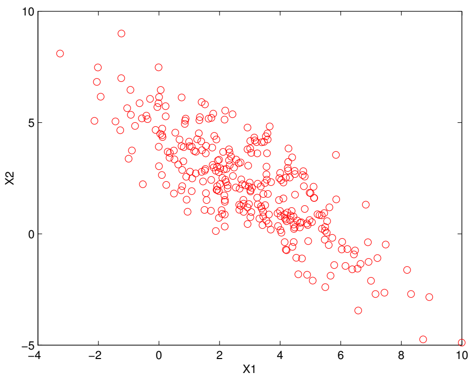
- It is possible to show that the mean vector $\hat{\boldsymbol{\mu}}$ and covariance matrix $\hat{\boldsymbol{\Sigma}}$ that maximize the likelihood of the training data are given by:

$$\hat{\boldsymbol{\mu}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}^{(n)}$$

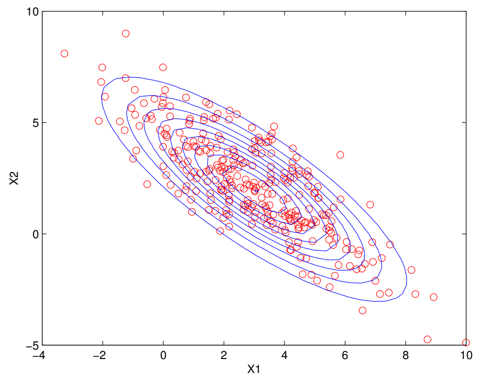
$$\hat{\boldsymbol{\Sigma}} = \frac{1}{N} \sum_{n=1}^{N} (\mathbf{x}^{(n)} - \hat{\boldsymbol{\mu}})(\mathbf{x}^{(n)} - \hat{\boldsymbol{\mu}})^T$$

- The mean of the distribution is estimated by the sample mean and the covariance by the sample covariance

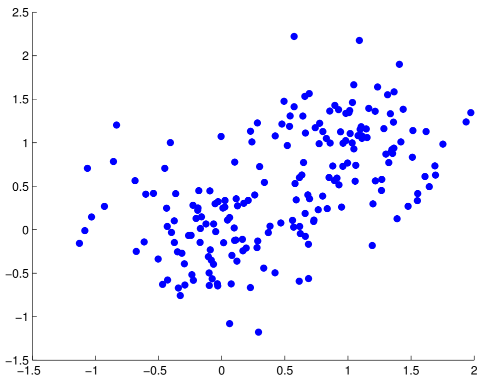
Example data



Maximum likelihood fit to a Gaussian

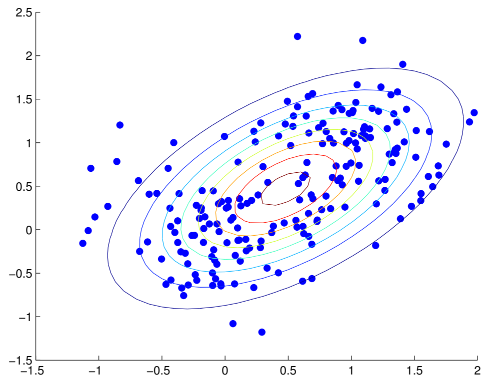


Data in clusters (example 1)



$$\boldsymbol{\mu}_1 = [0 \quad 0]^T \qquad \boldsymbol{\mu}_2 = [1 \quad 1]^T \qquad \boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_2 = 0.2\,\mathbf{I}$$

Example 1 fit by a Gaussian

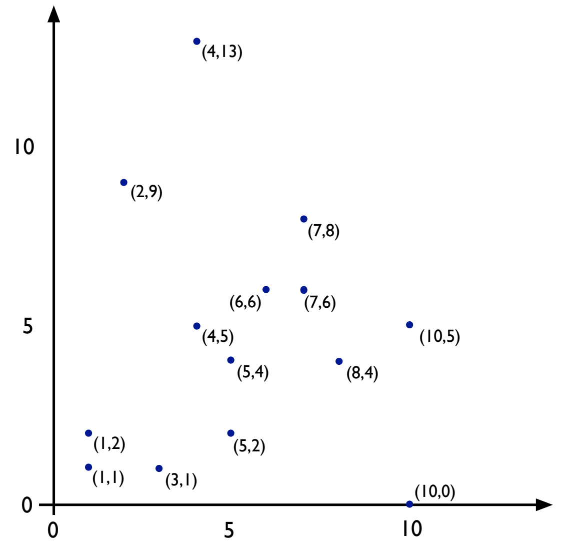


$$\boldsymbol{\mu}_1 = [0 \quad 0]^T \qquad \boldsymbol{\mu}_2 = [1 \quad 1]^T \qquad \boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_2 = 0.2\,\mathbf{I}$$

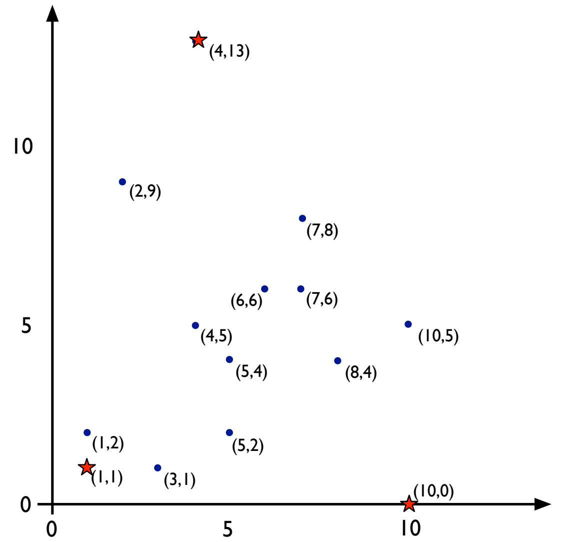
k-means clustering

- k-means is an automatic procedure for clustering unlabelled data
- Requires a prespecified number of clusters
- Clustering algorithm chooses a set of clusters with the minimum within-cluster variance
- Guaranteed to converge (eventually)
- Clustering solution is dependent on the initialisation

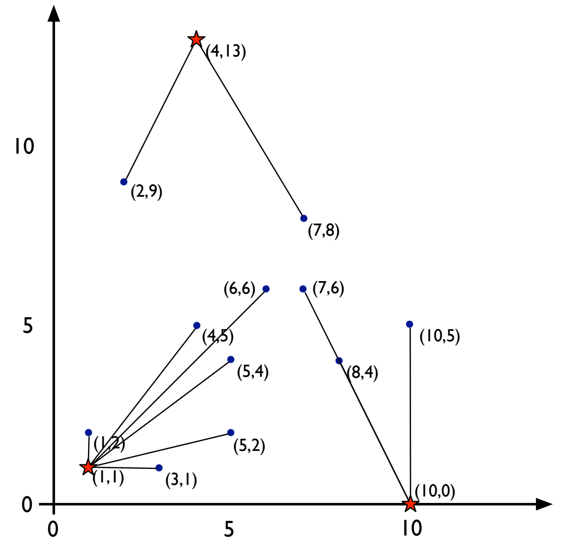
k-means example: data set



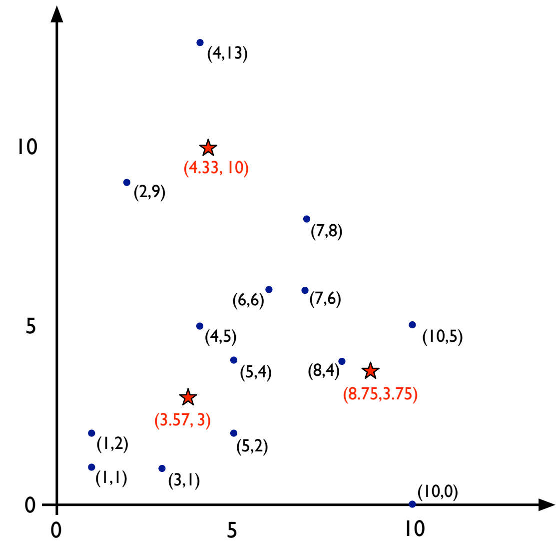
k-means example: initialization



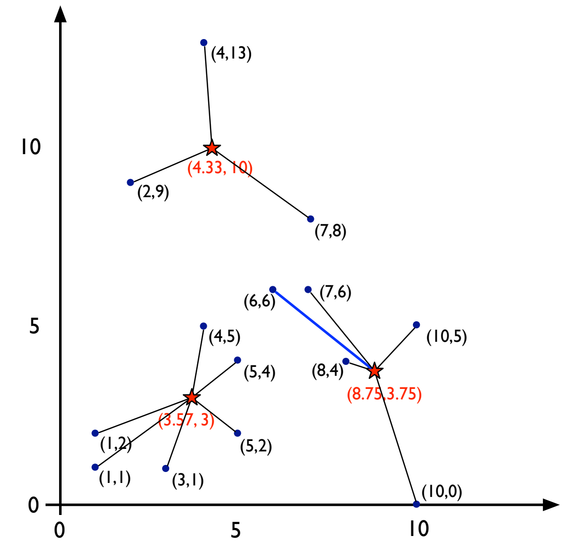
k-means example: iteration 1 (assign points to clusters)



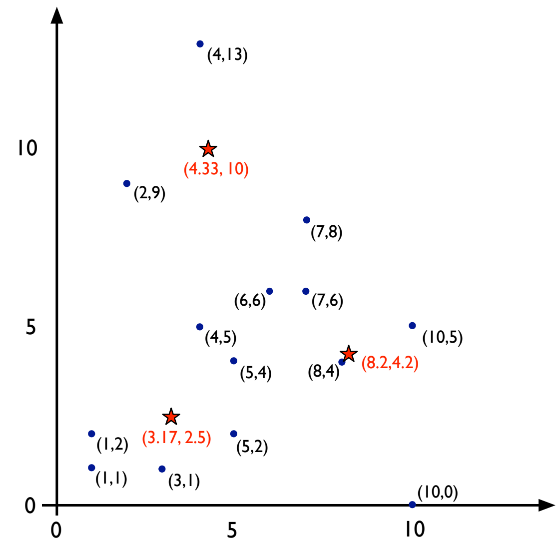
k-means example: iteration 1 (recompute centres)



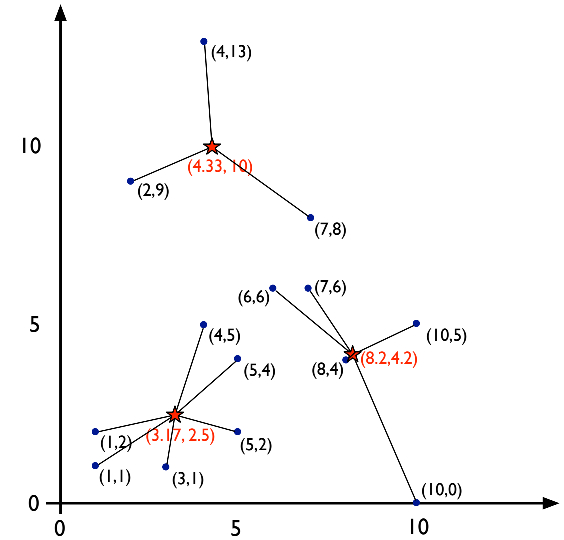
k-means example: iteration 2 (assign points to clusters)



k-means example: iteration 2 (recompute centres)



k-means example: iteration 3 (assign points to clusters)



No changes, so converged

Mixture model

- A more flexible form of density estimation is made up of a linear combination of component densities:

$$p(\mathbf{x}) = \sum_{j=1}^{M} p(\mathbf{x}|j)P(j)$$

- This is called a *mixture model* or a *mixture density*
- $p(\mathbf{x}|j)$: component densities
- $P(j)$: mixing parameters
- Generative model:
  1. Choose a mixture component based on $P(j)$
  2. Generate a data point $\mathbf{x}$ from the chosen component using $p(\mathbf{x}|j)$

Component occupation probability

- We can apply Bayes' theorem:

$$P(j|\mathbf{x}) = \frac{p(\mathbf{x}|j)P(j)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|j)P(j)}{\sum_{j=1}^{M} p(\mathbf{x}|j)P(j)}$$

- The posterior probabilities $P(j|\mathbf{x})$ give the probability that component j was responsible for generating data point $\mathbf{x}$
- The $P(j|\mathbf{x})$s are called the *component occupation probabilities* (or sometimes called the *responsibilities*)
- Since they are posterior probabilities:

$$\sum_{j=1}^{M} P(j|\mathbf{x}) = 1$$

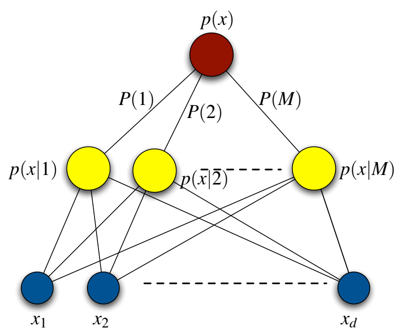
Parameter estimation

- *If* we knew which mixture component was responsible for a data point:
  - we would be able to assign each point unambiguously to a mixture component
  - and we could estimate the mean for each component Gaussian as the sample mean (just like k-means clustering)
  - and we could estimate the covariance as the sample covariance
- *But* we don't know which mixture component a data point comes from...
- Maybe we could use the component occupation probabilities $P(j|\mathbf{x})$?

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 $N_j(\mathbf{x}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$ has mean $\boldsymbol{\mu}_j$ and a spherical covariance $\boldsymbol{\Sigma}_j = \sigma_j^2 \mathbf{I}$

$$p(\mathbf{x}) = \sum_{j=1}^{M} P(j)p(\mathbf{x}|j) = \sum_{j=1}^{M} P(j)N_j(\mathbf{x}; \boldsymbol{\mu}_j, \sigma_j^2 \mathbf{I})$$



GMM Parameter estimation when we know which component generated the data

- Define the indicator variable $z_{jn} = 1$ if component j generated component $\mathbf{x}^{(n)}$ (and 0 otherwise)
- If z_{jn} wasn't hidden then we could count the number of observed data points generated by j:

$$N_j = \sum_{n=1}^{N} z_{jn}$$

- And estimate the mean, variance and mixing parameters as:

$$\hat{\boldsymbol{\mu}}_j = \frac{\sum_n z_{jn}\mathbf{x}^{(n)}}{N_j}$$

$$\hat{\sigma}_j^2 = \frac{\sum_n z_{jn}||\mathbf{x}^{(n)} - \hat{\boldsymbol{\mu}}_j||^2}{N_j}$$

$$\hat{P}(j) = \frac{1}{N}\sum_n z_{jn} = \frac{N_j}{N}$$

Soft assignment

- Estimate "soft counts" based on the component occupation probabilities $P(j\,|\mathbf{x}^{(n)})$:

$$N_j^* = \sum_{n=1}^{N} P(j\,|\mathbf{x}^{(n)})$$

- We can imagine assigning data points to component j weighted by the component occupation probability $P(j\,|\mathbf{x}^{(n)})$
- So we could imagine estimating the mean, variance and prior probabilities as:

$$\hat{\boldsymbol{\mu}}_j = \frac{\sum_n P(j\,|\mathbf{x}^{(n)})\mathbf{x}^{(n)}}{\sum_n P(j\,|\mathbf{x}^{(n)})} = \frac{\sum_n P(j\,|\mathbf{x}^{(n)})\mathbf{x}^{(n)}}{N_j^*}$$

$$\hat{\sigma}_j^2 = \frac{\sum_n P(j\,|\mathbf{x}^{(n)})||\mathbf{x}^{(n)} - \boldsymbol{\mu}_j||^2}{\sum_n P(j\,|\mathbf{x}^{(n)})} = \frac{\sum_n P(j\,|\mathbf{x}^{(n)})||\mathbf{x}^{(n)} - \boldsymbol{\mu}_j||^2}{N_j^*}$$

$$\hat{P}(j) = \frac{1}{N}\sum_n P(j\,|\mathbf{x}^{(n)}) = \frac{N_j^*}{N}$$

EM algorithm

- *Problem!* Recall that:

$$P(j\,|\mathbf{x}) = \frac{p(\mathbf{x}|\,j\,)P(j\,)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|\,j\,)P(j\,)}{\sum_{j=1}^{M} p(\mathbf{x}|\,j\,)P(j\,)}$$

  We need to know $p(\mathbf{x}|\,j\,)$ and $P(j\,)$ to estimate the parameters of $p(j\,|\mathbf{x})$, and to estimate $P(j\,)$....
- Solution: an iterative algorithm where each iteration has two parts:
  - Compute the component occupation probabilities $P(j\,|\mathbf{x})$ using the current estimates of the GMM parameters (means, variances, mixing parameters) (E-step)
  - Computer the GMM parameters using the current estimates of the component occupation probabilities (M-step)
- Starting from some initialization (e.g. using k-means for the means) these steps are alternated until convergence
- This is called the *EM Algorithm* and can be shown to maximize the likelihood

Maximum likelihood parameter estimation

- The likelihood of a data set $\mathbf{X} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}\}$ is given by:

$$\mathcal{L} = \prod_{n=1}^{N} p(\mathbf{x}^{(n)}) = \prod_{n=1}^{N}\sum_{j=1}^{M} p(\mathbf{x}^{(n)}|\,j\,)P(j\,)$$

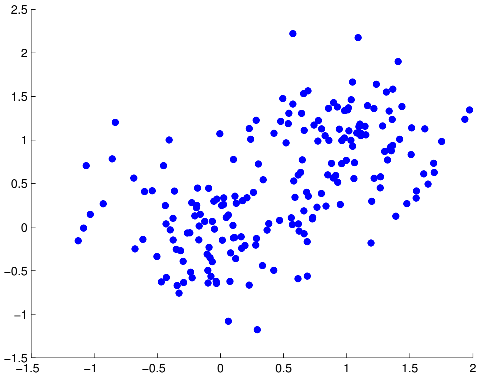
- We can regard the *negative log likelihood* as an error function:

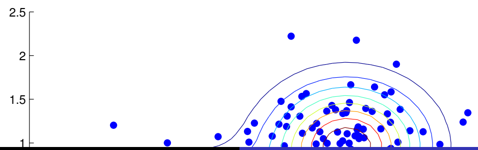
$$E = -\ln\mathcal{L} = -\sum_{n=1}^{N}\ln p(\mathbf{x}^{(n)})$$

$$= -\sum_{n=1}^{N}\ln\left(\sum_{j=1}^{M} p(\mathbf{x}^{(n)}|\,j\,)P(j\,)\right)$$

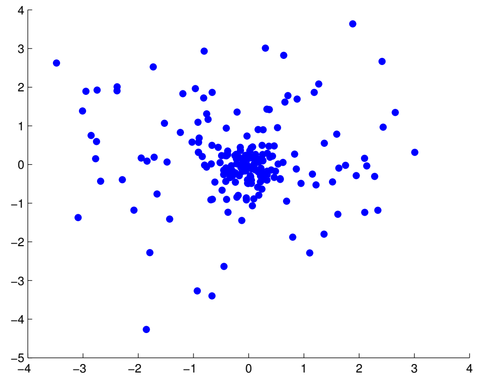
- Considering the derivatives of E with respect to the parameters, gives expressions like the previous slide

Example 1 fit using a GMM



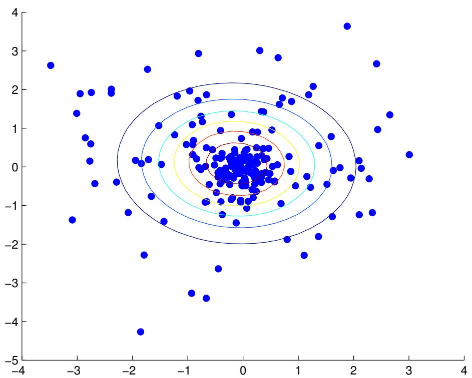


Peakily distributed data (Example 2)



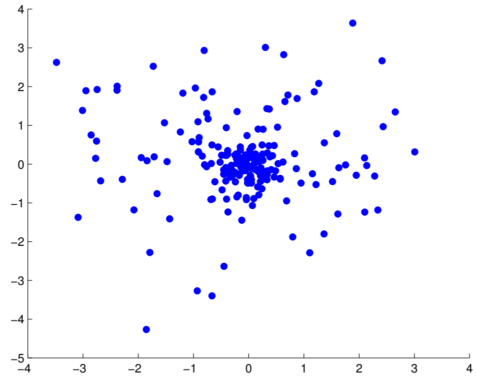
$$\boldsymbol{\mu}_1 = \boldsymbol{\mu}_2 = [0 \quad 0]^T \qquad \boldsymbol{\Sigma}_1 = 0.1\mathbf{I} \qquad \boldsymbol{\Sigma}_2 = 2\mathbf{I}$$

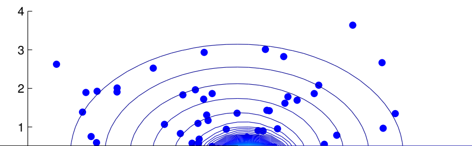
Example 2 fit by a Gaussian



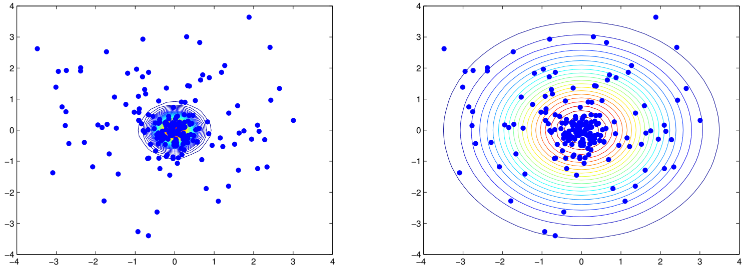
$$\boldsymbol{\mu}_1 = \boldsymbol{\mu}_2 = [0 \quad 0]^T \qquad \boldsymbol{\Sigma}_1 = 0.1\mathbf{I} \qquad \boldsymbol{\Sigma}_2 = 2\mathbf{I}$$

Example 2 fit by a GMM





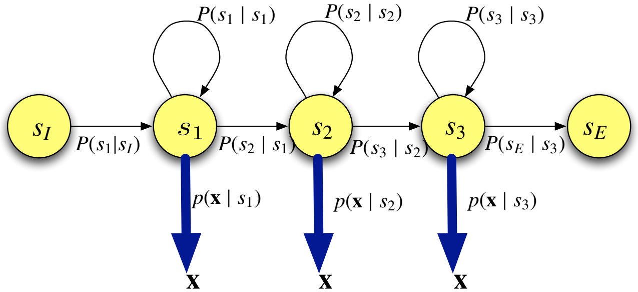
Example 2: component Gaussians



Comments on GMMs

- GMMs trained using the EM algorithm are able to self organize to fit a data set
- Individual components take responsibility for parts of the data set (probabilistically)
- Soft assignment to components not hard assignment — "soft clustering"
- GMMs scale very well, e.g.: large speech recognition systems can have 30,000 GMMs, each with 32 components: sometimes 1 million Gaussian components!! And the parameters all estimated from (a lot of) data by EM

Back to HMMs...



Output distribution:

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

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

- M-component Gaussian mixture model:

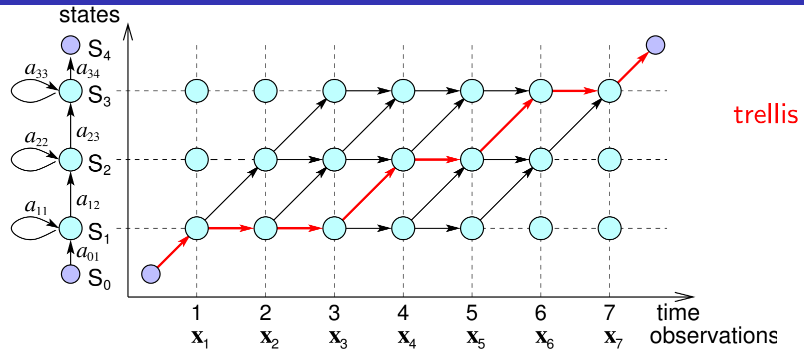
$$b_j(\mathbf{x}) = p(\mathbf{x} \mid S = j) = \sum_{m=1}^{M} c_{jm} \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm})$$

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. (NB: $\mathbf{x}_t$ is used to denote $\mathbf{x}^{(t)}$ hereafter)
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?



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

$$P(X|\Lambda) = \sum_{\{\mathrm{path}_j\}} P(X, \mathrm{path}_j|\Lambda) \quad \simeq \quad \max_{\mathrm{path}_j} P(X, \mathrm{path}_j|\Lambda)$$

forward(backward) algorithm — Viterbi algorithm

1. Likelihood: The Forward algorithm

- Goal: determine $p(\mathbf{X} \mid \boldsymbol{\lambda})$
- Sum over all possible state sequences $s_1 s_2 \ldots s_T$ that could result in the observation sequence $\mathbf{X}$
- Rather than enumerating each sequence, compute the probabilities recursively (exploiting the Markov assumption)
- Hown many paths calculations in $p(\mathbf{X} \mid \boldsymbol{\lambda})$?

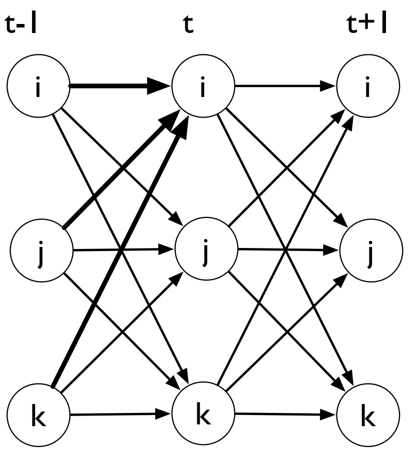
$$\sim \underbrace{N \times N \times \cdots N}_{T \text{ times}} = N^T \qquad \begin{array}{ll} N: & \text{number of HMM states} \\ T: & \text{length of observation} \end{array}$$

e.g. $N = 3,\ T = 20 \ \rightarrow \approx\ 10^{10}$

- Computation complexity for multiplication: $O(2T\,N^T)$
- The Forward algorithm reduces this to $O(TN^2)$

Recursive algorithms on HMMs

Visualize the problem as a *state-time trellis*



1. Likelihood: The Forward algorithm

- Goal: determine $p(\mathbf{X} \mid \boldsymbol{\lambda})$
- Sum over all possible state sequences $s_1 s_2 \ldots s_T$ that could result in the observation sequence $\mathbf{X}$
- Rather than enumerating each sequence, compute the probabilities recursively (exploiting the Markov assumption)
- *Forward probability*, $\alpha_t(j)$: the probability of observing the observation sequence $\mathbf{x}_1 \ldots \mathbf{x}_t$ and being in state j at time t:

$$\alpha_t(j) = p(\mathbf{x}_1, \ldots, \mathbf{x}_t, S(t) = j \mid \boldsymbol{\lambda})$$

1. Likelihood: The Forward recursion

- Initialization

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

- Recursion

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

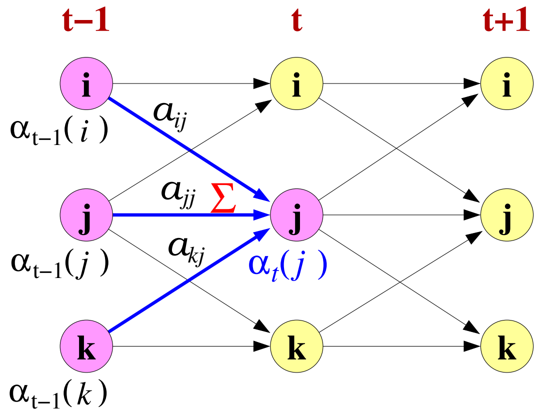
- Termination

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

s_I: initial state, s_E: final state

1. Likelihood: Forward Recursion

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



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:

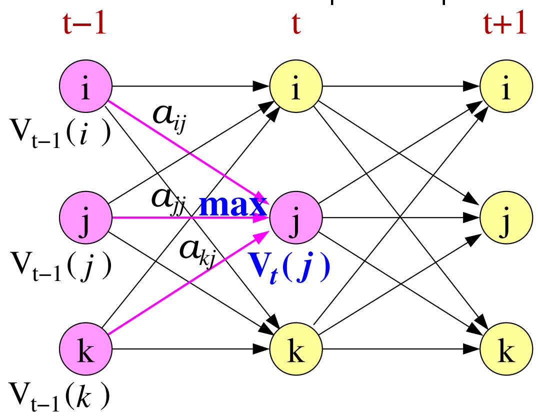
$$V_t(j) = \max_i V_{t-1}(i) a_{ij} b_j(\mathbf{x}_t)$$

- 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

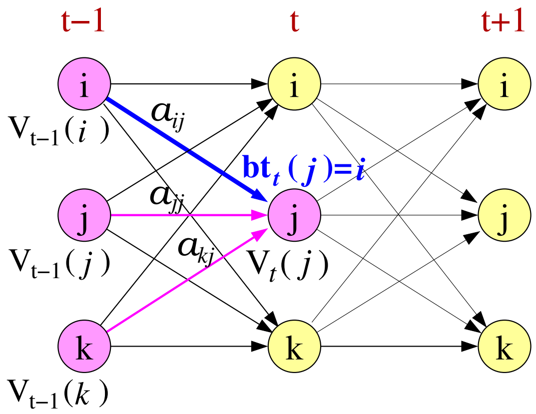
$$V_t(j) = \max_i V_{t-1}(i) a_{ij} b_j(\mathbf{x}_t)$$

Likelihood of the most probable path



Viterbi Recursion

Backpointers to the previous state on the most probable path



2. Decoding: The Viterbi algorithm

- Initialization

$$V_0(i) = 1$$
$$V_0(j) = 0 \qquad \text{if } j \neq i$$
$$bt_0(j) = 0$$

- Recursion

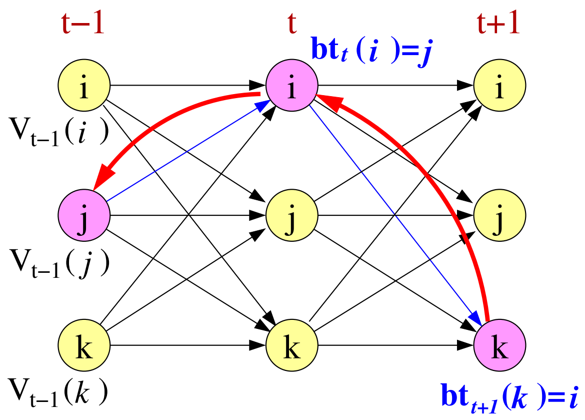
$$V_t(j) = \max_{i=1}^{N} V_{t-1}(i) a_{ij} b_j(\mathbf{x}_t)$$
$$bt_t(j) = \arg\max_{i=1}^{N} V_{t-1}(i) a_{ij} b_j(\mathbf{x}_t)$$

- Termination

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

Viterbi Backtrace

Backtrace to find the state sequence of the most probable path



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:

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

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

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

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

  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

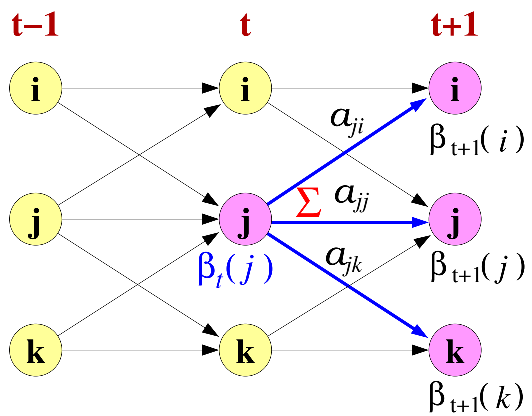
$$\beta_t(\, i\,) = \sum_{j=1}^{N} a_{ij} b_j(\mathbf{x}_{t+1}) \beta_{t+1}(\, j\,) \quad \text{for } t = T-1, \ldots, 1$$

  - Termination

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

Backward Recursion

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



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:

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

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

- Since

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

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

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:

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

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

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:

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

- 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

Extension to a corpus of utterances

- We usually train from a large corpus of R utterances
- If $\mathbf{x}_t^r$ is the tth frame of the rth utterance $\mathbf{X}^r$ then we can compute the probabilities $\alpha_t^r(j)$, $\beta_t^r(j)$, $\gamma_t^r(j)$ and $\xi_t^r(i,j)$ as before
- The re-estimates are as before, except we must sum over the R utterances, eg:

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

Extension to Gaussian mixture model (GMM)

- The assumption of a Gaussian distribution at each state is very strong; in practice the acoustic feature vectors associated with a state may be strongly non-Gaussian
- In this case an M-component Gaussian mixture model is an appropriate density function:

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

  Given enough components, this family of functions can model any distribution.
- Train using the EM algorithm, in which the component estimation probabilities are estimated in the E-step

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

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

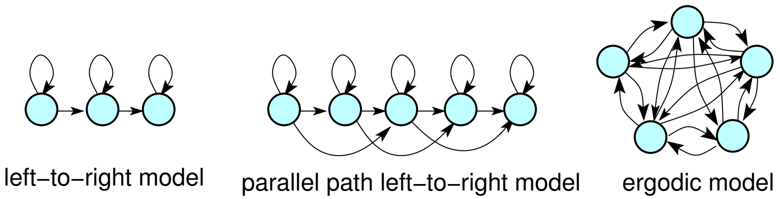
  And likewise for the covariance matrices (mixture models often use diagonal covariance matrices)
- The mixture coefficients are re-estimated in a similar way to transition probabilities:

$$\hat{c}_{jm} = \frac{\sum_{t=1}^{T}\gamma_t(j,m)}{\sum_{\ell=1}^{M}\sum_{t=1}^{T}\gamma_t(j,\ell)}$$

Doing the computation

- The forward, backward and Viterbi recursions result in a long sequence of probabilities being multiplied
- This can cause floating point *underflow* problems
- In practice computations are performed in the log domain (in which multiplies become adds)
- Working in the log domain also avoids needing to perform the exponentiation when computing Gaussians

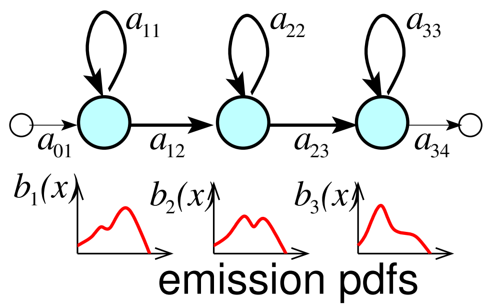
A note on HMM topology



$$\begin{pmatrix} a_{11} & a_{12} & 0 \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{pmatrix} \quad \begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 \\ 0 & a_{22} & a_{23} & a_{24} & 0 \\ 0 & 0 & a_{33} & a_{34} & a_{35} \\ 0 & 0 & 0 & a_{44} & a_{45} \\ 0 & 0 & 0 & 0 & a_{55} \end{pmatrix} \quad \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{pmatrix}$$

Speech recognition: left-to-right HMM with $3 \sim 5$ states
Speaker recognition: ergodic HMM

A note on HMM emission probabilities



	Emission prob.	
Continuous (density) HMM	continuous density	GMM, NN/DNN
Discrete (probability) HMM	discrete probability	VQ
Semi-continuous HMM (tied-mixture HMM)	continuous density	tied mixture

Summary: HMMs

- HMMs provide a generative model for statistical speech recognition
- Three key problems
  1. Computing the overall likelihood: the Forward algorithm
  2. Decoding the most likely state sequence: the Viterbi algorithm
  3. Estimating the most likely parameters: the EM (Forward-Backward) algorithm
- Solutions to these problems are tractable due to the two key HMM assumptions
  1. Conditional independence of observations given the current state
  2. Markov assumption on the states

References: HMMs

- Gales and Young (2007). "The Application of Hidden Markov Models in Speech Recognition", *Foundations and Trends in Signal Processing*, **1** (3), 195–304: section 2.2.
- Jurafsky and Martin (2008). *Speech and Language Processing* (2nd ed.): sections 6.1–6.5; 9.2; 9.4. (Errata at http://www.cs.colorado.edu/~martin/SLP/Errata/SLP2-PIEV-Errata.html)
- Rabiner and Juang (1989). "An introduction to hidden Markov models", *IEEE ASSP Magazine*, **3** (1), 4–16.
- Renals and Hain (2010). "Speech Recognition", *Computational Linguistics and Natural Language Processing Handbook*, Clark, Fox and Lappin (eds.), Blackwells.