Hidden Markov Models

Lecture Notes

Ahmet Ademoglu, *PhD*
Bogazici University
Institute of Biomedical Engineering

Some concepts and illustrations in this lecture are adapted from the textbooks,
Pattern Recognition and Machine Learning, C. M. Bishop, *Springer*, 2006.
Bayesian Reasoning and Machine Learning, D. Barber, *Cambridge*, 2020.

Hidden Markov Model (HMM)

Let's assume we have an observed sequence

$$\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_N], \text{ where } \mathbf{y}_n \in \mathcal{R}^d,\ n = 1, \ldots, N,$$

and a hidden sequence

$$\mathbf{Z} = [\mathbf{z}_1, \ldots, \mathbf{z}_N] \text{ where } n = 1, \ldots, N \text{ and } \mathbf{z}_n \in \{1, 2, \ldots, K\} \text{ where}$$

$\mathbf{z}_n$ is a $K-$binary vector all of whose element but one is zero.
The complete data *i.e.* the observed plus the (hidden) sequence is $(\mathbf{Y}, \mathbf{Z})$

Let's assume we have states as $\{1, 2, \ldots, K\}$
Joint distribution of a sequence of N observations under HMM :

$$p(\mathbf{y}_1, \ldots \mathbf{y}_N) = \prod_{n=1}^{N} p(\mathbf{y}_n | \mathbf{y}_1, \ldots, \mathbf{y}_{n-1})$$

For a first order HMM

$$p(\mathbf{y}_1, \ldots \mathbf{y}_N) = p(\mathbf{y}_1) \prod_{n=2}^{N} p(\mathbf{y}_n | \mathbf{y}_{n-1})$$

Using the latent variables $\mathbf{z}_n$, the joint distribution is

$$p(\mathbf{y}_1, \ldots \mathbf{y}_N, \mathbf{z}_1, \ldots \mathbf{z}_N) = p(\mathbf{z_1}) \prod_{n=2}^{N} p(\mathbf{z}_n | \mathbf{z}_{n-1}) \prod_{n=1}^{N} p(\mathbf{y}_n | \mathbf{z}_n)$$

If we assume a mixture model with K components ϕ_k, then $\mathbf{z}_n$ are discrete multinomial variables describing which component of the mixture is responsible for generating the corresponding observation $\mathbf{y}_n$.
$\mathbf{z}_n$ depends on the state of the previous latent variable $\mathbf{z}_{n-1}$ with probability $p(\mathbf{z}_n|\mathbf{z}_{n-1})$ and is denoted by $\mathbf{A}$ whose entries $A_{jk} = P(z_{nk} = 1|z_{n-1j} = 1)$ are called transition probabilities.

$$p(\mathbf{z}_n|\mathbf{z}_{n-1}, \mathbf{A}) = \prod_{k=1}^{K} \prod_{j=1}^{K} A_{jk}^{z_{n-1j} z_{nk}}$$

The probability of $\mathbf{z}_1$: initial latent node is

$$p(\mathbf{z}_1|\pi) = \prod_{k=1}^{K} \pi_k^{z_{1k}}$$

set by π with $\pi_k = p(z_{1k} = 1)$ and $\sum_{k=1}^{K} \pi_k = 1$.

The probabilities of the observations: also called *emission probabilities* are

$$p(\mathbf{y}_n|\mathbf{z}_n,\phi) = \prod_{k=1}^{K} p(\mathbf{y}_n|\phi_k)^{z_{nk}}$$

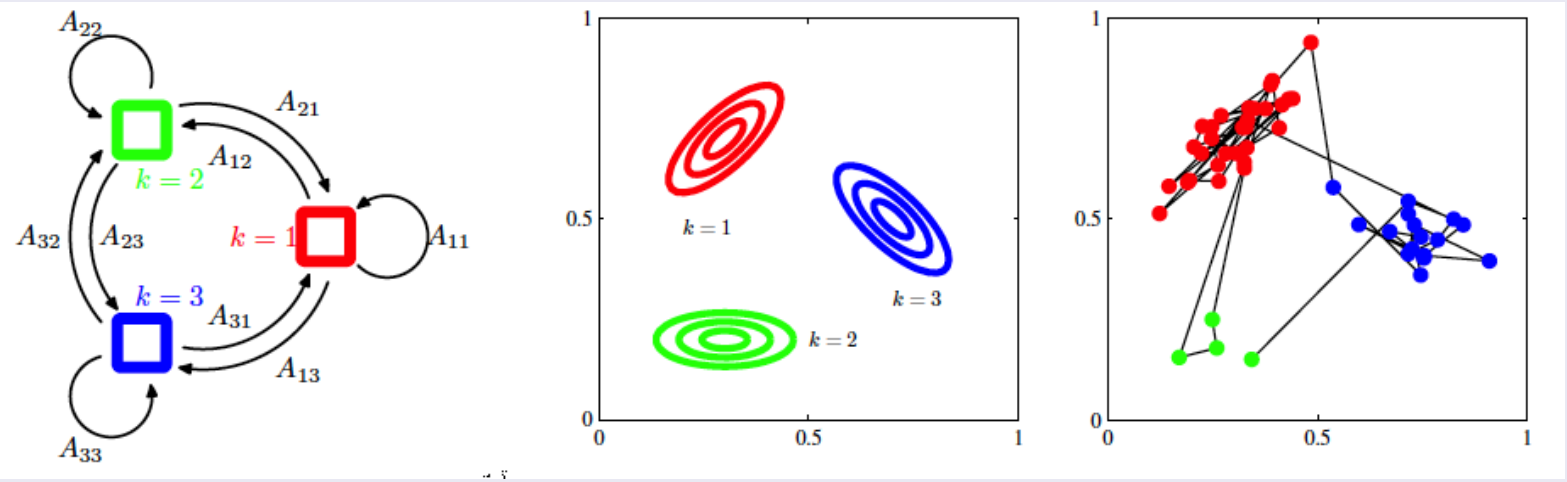
where $\mathbf{z}_n$ determines from which node the observation has been sampled from.

The joint distribution is

$$p(\mathbf{Y},\mathbf{Z}|\theta) = p(\mathbf{z}_1|\pi)\Big[\prod_{n=2}^{N} p(\mathbf{z}_n|\mathbf{z}_{n-1},\mathbf{A})\Big] \prod_{m=1}^{N} p(\mathbf{y}_m|\mathbf{z}_m,\phi)$$

and HMM is defined by $\theta = \{\mathbf{A},\phi,\pi\}$.

HMM with Three States



Sampling Procedure

- We first choose the initial state with $\mathbf{z}_1$ using the distribution π and then we choose the next state by determining $\mathbf{z_2}$ according to the distribution $p(\mathbf{z}_2|\mathbf{z}_1)$ using A_{jk}.
- $\mathbf{y}_2$ is sampled from component with distribution ϕ_k for which $z_{2k} = 1$.
- Once we know $\mathbf{z}_2$ we can determine the next hidden variable $\mathbf{z}_3$ and sample $\mathbf{y}_3$ so on.

Generating Nucleotide Sequence Clusters using a Simple Model

Let's assume we have two clusters each having sequences of 20 symbols from the set $\{A, G, C, T\}$ such as `CATAGGCATTCTATGTGCTG`.

After the number of clusters, their prior probabilities π and state emission probabilities $\{p(y|z = 1), p(y|z = 2)\}$ are specified;

1. Randomly choose a cluster z according to prior distribution π.
2. Generate the 20 nucleotides using $p(y|z)$ belonging to the cluster z.
3. Repeat Steps 1 and 2 until you generate all the sequences.

Generation of Nucleotide Sequence Clusters using HMM

```
N= 20; % sequence length
Ns = 20; % number of gene sequences
Nuc = {'A','G','C','T'}; % nucleotides
pycz = [ 0.4 0.1 0.1 0.4; 0.1 0.4 0.4 0.1]'; % p(y|z)
PI  = [0.5 0.5]; % prior probability for clusters
CPI =  cumsum(PI); % cumulative function for PI
s = 1;
% choose a cluster with prior probability PI
%if rand(1,1) < PI(1),  C(s) = 1; else C(s)=2;   end;
c = sum( (rand()>=CPI))+1; C(s) = c;
% Generate a stationary HMM sequence in cluster C(s)
S(1) = randgen(pycz(:,C(1)));  %randomly determine the initial state based on z_1
for n=2:N,
S(n) = randgen(pycz(:,C(1))); % sample from the appropriate emission distribution
%S(n) =sum( (rand()>=squeeze(CA(:,S(n-1),c))))+1;
end;
Seq(s) = join(Nuc(S));
sequences{1,s}= regexprep(Seq{s}, '\s', '');
for s=2:Ns,
% choose a cluster with prior probability PI
%  if rand(1,1) > 0.4,  c = 1; else   c = 2; end;   C(s) = c;
c = sum( (rand()>=CPI))+1; C(s) = c;
S(1) = randgen(pycz(:,C(s)));% Generate a stationary HMM sequence in cluster c
for n=2:N,   S(n) = randgen(pycz(:,C(s)));   end;
Seq(s) = join(Nuc(S));
sequences{1,s} = regexprep(Seq{s}, '\s', '');
end;
save ('mysequences','sequences','C');
```

Generation of Nucleotide Sequences using HMM

```
% Obtained from Barber's book
function y = randgen(p, m, n, s)
% RANDGEN Generates discrete random variables given the pdf
% Y = RANDGEN(P, M, N, S)
% Inputs :
% P : Relative likelihoods (or pdf) of symbols
% M : Rows    <Default = 1>
% N : Columns <Default = 1>
% S : Symbols <Default 1:length(P)>
% Change History :
% Date Time Prog Note
% 06-Jul-1997  4:36 PM ATC Created under MATLAB 5.0.0.4064
% ATC = Ali Taylan Cemgil,
% Bogazici University, Dept. of Computer Eng. 80815 Bebek Istanbul Turkey
% e-mail : cemgil@boun.edu.tr
if isempty(p); y=[]; return; end % no samples if p is empty (DB)
if (nargin < 2) m = 1; end
if (nargin < 3) n = 1;end
if (nargin < 4) s = 1:length(p);end
c = cumsum(p);
c = c(:)'/c(end);
N = m*n;
u = rand(N,1);
% for i=1:N, y(i) = length(find(c<y(i)))+1; end;
y = sum( c(ones(N,1),:) < u(:, ones(length(c),1)) , 2 ) + 1;
y = reshape(s(y), m, n);
```

EM Algorithm for HMM Classification

Given a set of N_s sequences $\mathbf{Y} = \{y_{1:N}^1, \ldots, y_{1:N}^{N_s}\}$, assuming that they are *i.i.d.*, we can define a mixture model for a single sequence $y_{1:N}$

$$p(y_{1:N}) = \sum_{z=1}^{K} p(z)p(y_{1:N}|z) = \sum_{z=1}^{K} p(z) \prod_{n=1}^{N} p(y_n|y_{n-1}, z)$$

Clustering can be achieved by finding the maximum likelihood parameters, $p(z)$ and $p(y_n|y_{n-1}, z)$ and assigning the clusters according to $p(z|y_{1:N})$.

Since $p(\mathbf{Y}) = \prod_{s=1}^{N_s} p(y_{1:N}^s)$, we can maximize the log likelihood of

$$\log p(\mathbf{Y}) = \sum_{s=1}^{N_s} \log \sum_{z=1}^{K} p(z) \prod_{n=1}^{N} p(y_n^s|y_{n-1}^s, z)$$

where z is the hidden variable that determines from which cluster the sequence is sampled.

For the M-Step, we first maximize the energy E with respect to $p(z)$

$$E = \sum_{s=1}^{N_s} \mathrm{E}_{p^{old}(z|y^s_{1:N})} [\log p(y^s_{1:N}, z)]$$

$$= \sum_{s=1}^{N_s} \left\{ \underbrace{\mathrm{E}_{p^{old}(z|y^s_{1:N})}[\log p(z)]}_{\text{Contribution to } E \text{ from parameter } p(z)} + \sum_{n=1}^{N} \mathrm{E}_{p^{old}(z|y^s_{1:N})}[\log p(y^s_n|y^s_{n-1}, z)] \right\}$$

The first term in E can be expressed as

$$\sum_{s=1}^{N_s} \mathrm{E}_{p^{old}(z|y^s_{1:N})}[\log p(z)] = \int \underbrace{\sum_{s=1}^{N_s} p^{old}(z|y^s_{1:N})}_{\propto p^{old}(z)} \log(p(z))dz$$

$$\propto -KL(p^{old}(z)|p(z)) + \int p^{old}(z) \log p^{old}(z)dz$$

Maximizing the above expression is achieved by minimizing

$$KL(p^{old}(z)|p(z))$$

which can be achieved when

$$p^{old}(z) = p(z) = p^{new}(z) \propto \sum_{s=1}^{N_s} p^{old}(z|y^s_{1:N})$$

The energy should also be maximized with respect to

$$p(y_n^s | y_{n-1}^s, z).$$

with a similar reasoning that we used in maximizing E with respect to $p(z)$.

For the M-step maximizing E with respect to $p(y_n^s | y_{n-1}^s, z)$ we can choose

$$p^{new}(y_n^s = i | y_{n-1}^s = j, z = k)$$
$$\propto \sum_{s=1}^{Ns} p^{old}(z = k | y_{1:N}^s) \sum_{n=2}^{N} \mathbb{I}[y_n^s = i] \mathbb{I}[y_{n-1}^s = j]$$

The inital term $p(y_1|z)$ is updated as

$$p(y_1 | z = k) \propto \sum_{s=1}^{Ns} p^{old}(z = k | y_{1:N}^s) \mathbb{I}[y_1^s = i]$$

The E-Step is

$$p^{old}(z|y^s_{1:N}) \propto p(z)p(y^s_{1:N}|z) = p(z)\prod_{n=1}^{N} p(y^s_n|y^s_{n-1}, z)$$

For long sequences, to overcome numerical overflow, it is better to work with log values *i.e.*

$$\log p^{old}(z|y^s_{1:N}) = \log p(z) + \sum_{n=1}^{N} \log p(y^s_n|y^s_{n-1}, z) + const$$

```
function [C,c]=hmm_mix_markov
% Output C: real class belongings, c:estimated class belongings  for sequences
load mysequences.mat
Ns = length(sequences);
for n=1:Ns % convert the characters into (arbitrary) numerical values (from 1 to 4)
Y{n}(findstr(sequences{n},'A'))=1;
Y{n}(findstr(sequences{n},'C'))=2;
Y{n}(findstr(sequences{n},'G'))=3;
Y{n}(findstr(sequences{n},'T'))=4;
end
D=4; % number of states of observable data
K=2; % number of classes
opts.maxit=50; opts.plotprogress=1;
[pz,py1cz,pycy_1z,loglikelihood,pzcy]=mixMarkov1(Y,D,K,opts);
c=ones(1,20);
class2sequences=[]; class1sequences=[];
for n=1:Ns
if pzcy{n}(1)<0.5; c(n)=2;
class2sequences=vertcat(class2sequences,sequences{n});
else
class1sequences=vertcat(class1sequences,sequences{n});
end
end
fprintf(1,' Sequences in group 1:\n'); disp(class1sequences)
fprintf(1,'\n Sequences in group 2:\n'); disp(class2sequences)
fprintf(1,'\n labels of each sequence:\n'); disp(c);
fprintf(1,'\n log likelihood for this run = %g\n',loglikelihood);
```

```
function [pz,py1cz,pycy_1z,loglikelihood,pzcy]=mixMarkov1(Y,D,K,opts);
% EM training for a mixture of Markov Models
% Adapted from Barber's ML code MixMarkovDemo.m
% [pz,py1cz,pycy_1z,loglikelihood,pzcy]=mixMarkov1(Y,D,K,opts);
% Inputs:
% Y : cell array of the data. Y{1} contains the first data sequence, Y{2} the next etc.
% D : the number of visible states of the data
% K : number of mixture components
% opts.maxit : the number of iterations of the EM algorithm
% Outputs:
% pz : learned p(z)
% py1cz : p(y(1)|z)
% pycy_1z : p(v(n)|v(n-1),z)
% loglikelihood : log likelihood of all the sequences (assuming iid)
% pzcy : posterior probability p(z|y) for each sequence
pz= condp(rand(K,1)); % random parameter initialisation
py1cz=condp(rand(D,K));
pycy_1z=condp(rand(D,D,K));
for emloop=1:opts.maxit
% E-step
pz_stat=zeros(K,1);
py1cz_stat =zeros(D,K);
pycy_1z_stat =zeros(D,D,K);
loglik=0;
for ns=1:length(Y)
N = length(Y{ns});
lpz_old =log(pz)+log(py1cz(Y{ns}(1),:)');
```

```
for n=2:N
lpz_old=lpz_old+squeeze(log(pycy_1z(Y{ns}(n),Y{ns}(n-1),:)));
end
pz_old{ns}=condexp(lpz_old);
loglik = loglik + logsumexp(lpz_old,ones(K,1)); % avoids numerical underflow
% collect statistics for M-step:
pz_stat=pz_stat + pz_old{ns};
py1cz_stat(Y{ns}(1),:)=py1cz_stat(Y{ns}(1),:) + pz_old{ns}';
for n=2:N
pycy_1z_stat(Y{ns}(n),Y{ns}(n-1),:) = squeeze(pycy_1z_stat(Y{ns}(n),Y{ns}(n-1),:))+ pz_old{ns};
end
end
llik(emloop)=loglik;
if opts.plotprogress; plot(llik); title('log likelihood'); drawnow; end
% M-step
pz = condp(pz_stat);
pv1gh = condp(py1cz_stat);
pycy_1z = condp(pycy_1z_stat);
end
loglikelihood=llik(end); pzcy=pz_old;
```

```
function pnew=condp(pin,varargin)
%CONDP Make a conditional distribution from an array
% pnew=condp(pin,<distribution indices>)
% Input : pin  -- a positive matrix pin
% Output:  matrix pnew such that sum(pnew,1)=ones(1,size(p,2))
% The optional specifies which indices form the distribution variables.
% For example:
% r=rand([4 2 3]); p=condp(r,[3 1]);
% p is now an array of the same size as r, but with sum(sum(p,3),1) equal
% to 1 for each of the dimensions of the 2nd index.
m=max(pin(:));
if m>0    p = pin./m;   else
p=pin+eps;% in case all unnormalised probabilities are zero
end
if nargin==1
pnew=p./repmat(sum(p,1),size(p,1),1);
else
allvars=1:length(size(pin));
sizevars=size(pin);
distvars=varargin{1};
condvars=setdiff(allvars,distvars);
newp=permute(pin,[distvars condvars]);
newp=reshape(newp,prod(sizevars(distvars)),prod(sizevars(condvars)));
newp=newp./repmat(sum(newp,1),size(newp,1),1);
pnew=reshape(newp,sizevars([distvars condvars]));
pnew=ipermute(pnew,[distvars condvars]);
end
```

```
function pnew=condexp(logp)
%CONDEXP  Compute p\propto exp(logp);
pmax=max(logp,[],1); P =size(logp,1);
pnew = condp(exp(logp-repmat(pmax,P,1)));
```

```
function anew=logsumexp(a,varargin)
%LOGSUMEXP Compute log(sum(exp(a).*b)) valid for large a
% logsumexp(a,<b>)
% if b is missing it is assumed to be 1
% example: logsumexp([-1000 -1001 -998],[1 2 0.5])
if nargin==1
b=ones(size(a));
else
b=varargin{1};
if isscalar(b); b=b*ones(size(a)); end
end
amax=max(a); A =size(a,1);
anew = amax + log(sum(exp(a-repmat(amax,A,1)).*b));
```

EM Algorithm on HMM

Complete log-likelihood function

$$Q(\theta|\theta^{old}) = \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{Y}, \theta^{old}) \log p(\mathbf{Y}, \mathbf{Z}|\theta)$$

$$p(\mathbf{Y}, \mathbf{Z}|\theta) = p(\mathbf{z}_1|\pi)\Big[\prod_{n=2}^{N} p(\mathbf{z}_n|\mathbf{z}_{n-1}, \mathbf{A})\Big] \prod_{n=1}^{N} p(\mathbf{y}_n|\mathbf{z}_n, \phi_k)$$

$$p(\mathbf{z}_1|\pi) = \prod_{k=1}^{K} \pi_k^{z_{1k}} \ , \ p(\mathbf{z}_n|\mathbf{z}_{n-1}, \mathbf{A}) = \prod_{k=1}^{K}\prod_{j=1}^{K} A_{jk}^{z_{n-1j}z_{nk}}$$

and $p(\mathbf{y}_n|\mathbf{z}_n, \phi) = \prod_{k=1}^{K} p(\mathbf{y}_n|\phi_k)^{z_{nk}}$

We define the marginal and joint distributions of $\mathbf{z}_n$ and $\mathbf{z}_{n-1}$

$$p(\mathbf{z}_n|\mathbf{Y}, \theta^{old}) = \gamma(\mathbf{z}_n)$$
$$p(\mathbf{z}_{n-1}, \mathbf{z}_n|\mathbf{Y}, \theta^{old}) = \xi(\mathbf{z}_{n-1}, \mathbf{z}_n)$$

For a discrete multinomial random variable $\mathbf{z}_n$, the expected value of one of its components is the probability of that component to have the value of 1 as

$$E[z_{nk}] = \gamma(z_{nk}) = \sum_{\mathbf{z}_n} \gamma(\mathbf{z}_n) z_{nk}$$

$$\xi(z_{n-1j}, z_{nk}) = E[z_{n-1j} z_{nk}] = \sum_{\mathbf{z}_n} \gamma(\mathbf{z}_n) z_{n-1j} z_{nk}$$

The E-step yields

$$Q(\theta|\theta^{old}) = \sum_{k=1}^{K} \gamma(z_{1k}) \log \pi_k + \sum_{n=2}^{N} \sum_{j=1}^{K} \sum_{k=1}^{K} \xi(z_{n-1j}, z_{nk}) \log A_{jk} + \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_{nk}) \log p(\mathbf{y}_n|\phi_k)$$

The M-step maximizes $Q(\theta|\theta^{old})$ with respect to θ subject to $\sum_{k=1}^{K} \pi_k = 1$, $\pi_i \geq 0$ and $\sum_{k=1}^{k} A_{ik} = 1$, $A_{ij} \geq 0$ to yield

$$\pi_k = \frac{\gamma(z_{1k})}{\sum_{j=1}^{K} \gamma(z_{1j})}$$

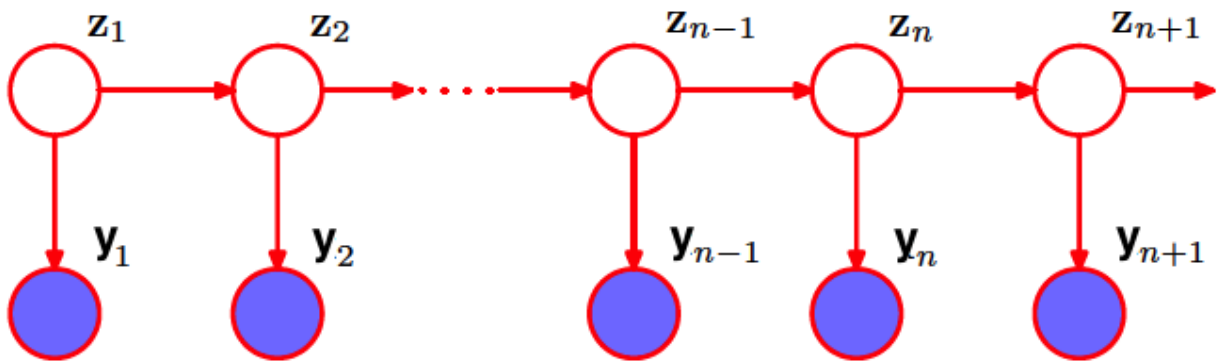
$$A_{jk} = \frac{\sum_{n=2}^{N} \xi(z_{n-1j}, z_{nk})}{\sum_{l=1}^{K} \sum_{n=2}^{N} \xi(z_{n-1j}, z_{nl})}$$

If we choose $p(\mathbf{y}|\phi_k) = \mathcal{N}(\mathbf{y}|\mu_k, \Sigma_k)$

$$\mu_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk})\mathbf{y}_n}{\sum_{n=1}^{N} \gamma(z_{nk})} \text{ and } \Sigma_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk})(\mathbf{y}_n - \mu_k)(\mathbf{y}_n - \mu_k)^T}{\sum_{n=1}^{N} \gamma(z_{nk})}$$

Efficient Computation of $\gamma(z_{nk})$ and $\xi(z_{n-1j}, z_{nk})$

$$
\begin{aligned}
p(\mathbf{Y}|\mathbf{z}_n) &= p(\mathbf{y}_1, \ldots, \mathbf{y}_n|\mathbf{z}_n)p(\mathbf{y}_{n+1}, \ldots, \mathbf{y}_N|\mathbf{z}_n) \\
p(\mathbf{y}_1, \ldots, \mathbf{y}_{n-1}|\mathbf{y}_n, \mathbf{z}_n) &= p(\mathbf{y}_1, \ldots, \mathbf{y}_{n-1}|\mathbf{z}_n) \\
p(\mathbf{y}_1, \ldots, \mathbf{y}_{n-1}|\mathbf{z}_{n-1}, \mathbf{z}_n) &= p(\mathbf{y}_1, \ldots, \mathbf{y}_{n-1}|\mathbf{z}_{n-1}) \\
p(\mathbf{y}_{n+1}, \ldots, \mathbf{y}_N|\mathbf{z}_n, \mathbf{z}_{n+1}) &= p(\mathbf{y}_{n+1}, \ldots, \mathbf{y}_N|\mathbf{z}_{n+1}) \\
p(\mathbf{y}_{n+2}, \ldots, \mathbf{y}_N|\mathbf{z}_{n+1}, \mathbf{y}_{n+1}) &= p(\mathbf{y}_{n+2}, \ldots, \mathbf{y}_N|\mathbf{z}_{n+1}) \\
p(\mathbf{Y}|\mathbf{z}_{n-1}, \mathbf{z}_n) &= p(\mathbf{y}_1, \ldots, \mathbf{y}_{n-1}|\mathbf{z}_{n-1})p(\mathbf{y}_n|\mathbf{z}_n) \\
&\quad p(\mathbf{y}_{n+1}, \ldots, \mathbf{y}_N|\mathbf{z}_n) \\
p(\mathbf{y}_{n+1}|\mathbf{Y}, \mathbf{z}_{N+1}) &= p(\mathbf{y}_{N+1}|\mathbf{z}_{N+1}) \\
p(\mathbf{z}_{N+1}|\mathbf{z}_N, \mathbf{Y}) &= p(\mathbf{z}_{N+1}|\mathbf{z}_N)
\end{aligned}
$$



Efficient Computation of $\gamma(z_{nk})$ and $\xi(z_{n-1j}, z_{nk})$

For a discrete multinomial random variable $\mathbf{z}_n$ the expected value of one of its components is the probability of that component to have the value of 1.

The posterior distribution $p(\mathbf{z}_n|\mathbf{y}_1, \ldots, \mathbf{y}_N)$ which is a vector of length K whose entries are the expected values of z_{nk}.

$$\gamma(z_{nk}) = p(\mathbf{z}_n|\mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{z_n})p(\mathbf{z}_n)}{p(\mathbf{Y})}$$

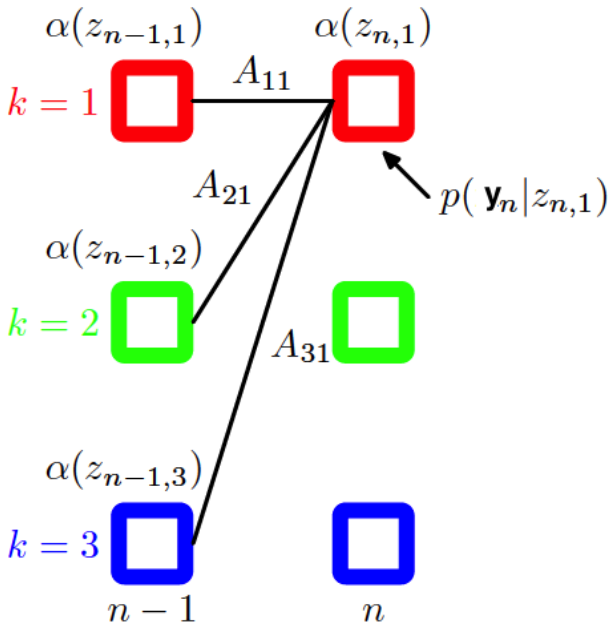
$$\gamma(z_{nk}) = \frac{\overbrace{p(\mathbf{y}_1, \ldots, \mathbf{y}_n, \mathbf{z}_n)}^{\alpha(\mathbf{z}_n)}\overbrace{p(\mathbf{y}_{n+1}, \ldots, \mathbf{y}_N|\mathbf{z}n)}^{\beta(\mathbf{z}_n)}}{p(\mathbf{Y})}$$

$\alpha(z_{nk})$ denotes the value of $\alpha(\mathbf{z}_n)$ when $z_{nk} = 1$.

Recursion Relation for $\alpha(\mathbf{z}_n)$

$$
\begin{aligned}
\alpha(\mathbf{z}_n) &= p(\mathbf{y}_1, \ldots, \mathbf{y}_n, \mathbf{z}_n) \\
&= p(\mathbf{y}_1, \ldots, \mathbf{y}_n | \mathbf{z}_n) p(\mathbf{z}_n) \\
&= p(\mathbf{y}_n | \mathbf{z}_n) p(\mathbf{y}_1, \ldots, \mathbf{y}_{n-1} | \mathbf{z}_n) p(\mathbf{z}_n) \\
&= p(\mathbf{y}_n | \mathbf{z}_n) p(\mathbf{y}_1, \ldots, \mathbf{y}_{n-1} | \mathbf{z}_n) p(\mathbf{z}_n) \\
&= p(\mathbf{y}_n | \mathbf{z}_n) p(\mathbf{y}_1, \ldots, \mathbf{y}_{n-1}, \mathbf{z}_n) \\
&= p(\mathbf{y}_n | \mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} p(\mathbf{y}_1, \ldots, \mathbf{y}_{n-1}, \mathbf{z}_n | \mathbf{z}_{n-1}) p(\mathbf{z}_{n-1}) \\
&= p(\mathbf{y}_n | \mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} p(\mathbf{y}_1, \ldots, \mathbf{y}_{n-1} | \mathbf{z}_{n-1}) p(\mathbf{z}_n | \mathbf{z}_{n-1}) p(\mathbf{z}_{n-1}) \\
&= p(\mathbf{y}_n | \mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} p(\mathbf{y}_1, \ldots, \mathbf{y}_{n-1}, \mathbf{z}_{n-1}) p(\mathbf{z}_n | \mathbf{z}_{n-1})
\end{aligned}
$$

$$
\alpha(\mathbf{z}_n) = p(\mathbf{y}_n | \mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} \alpha(\mathbf{z}_{n-1}) p(\mathbf{z}_n | \mathbf{z}_{n-1})
$$



Initalization of the Recursion

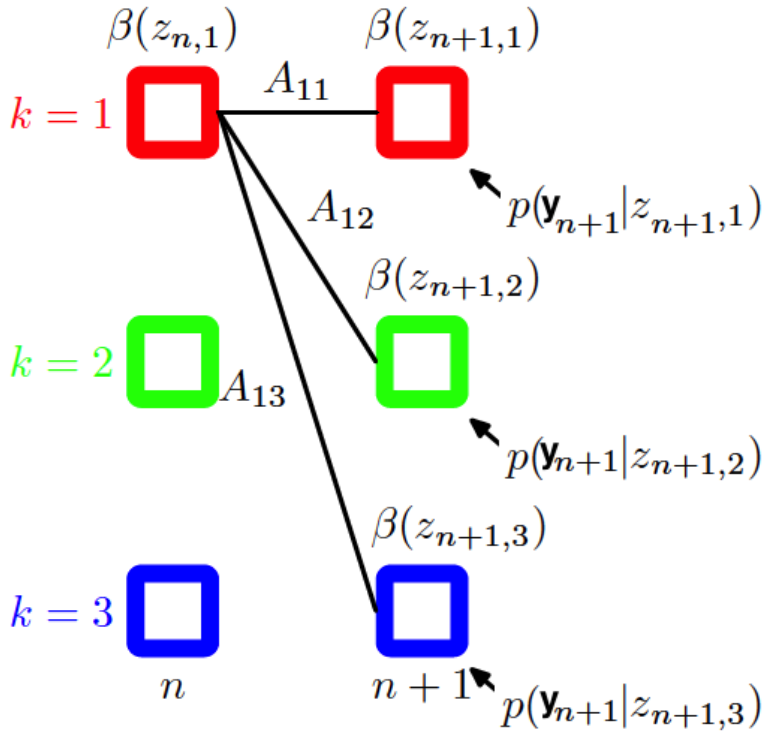
$$\alpha(\mathbf{z}_1) = p(\mathbf{y}_1, \mathbf{z}_1) = p(\mathbf{z}_1)p(\mathbf{y}_1|\mathbf{z}_1) = \prod_{k=1}^{K} \{\pi_k p(\mathbf{y}_1|\phi_k)\}^{z_{1k}}$$

which means that

$$\alpha(z_{1k}) = \pi_k p(\mathbf{y}_1|\phi_k) \text{ for } k = 1, \ldots, K.$$

Recursion Relation for $\beta(\mathbf{z}_n)$

$$\beta(\mathbf{z}_n) = \sum_{\mathbf{z}_{n+1}} \beta(\mathbf{z}_{n+1}) p(\mathbf{y}_{n+1}|\mathbf{z}_{n+1}) p(\mathbf{z}_{n+1}|\mathbf{z}_n)$$



Initalization of the Recursion

Setting $n = N$, using the definitions of $\alpha(\mathbf{z}_N)$ and $\gamma(\mathbf{z}_N)$ *i.e.*
$\alpha(\mathbf{z}_N) = p(\mathbf{Y}, \mathbf{z}_N)$ and $\gamma(\mathbf{z}_N) = p(\mathbf{z}_N|\mathbf{Y}) = \alpha(\mathbf{z}_N)\beta(\mathbf{z}_N)/p(\mathbf{Y})$

we can get

$$p(\mathbf{z}_N|\mathbf{Y}) = p(\mathbf{Y}, \mathbf{z}_N)\beta(\mathbf{z}_N)/p(\mathbf{Y})$$

which also requires $\beta(\mathbf{z}_N)$ to be equal to 1.

We can also find the likelihood distribution $p(\mathbf{Y})$ by summing $\gamma(\mathbf{z}_N)$ over $\mathbf{z}_N$ and assuming that it is a normalized distribution

$$\sum_{\mathbf{z}_N} \gamma(\mathbf{z}_N) = \sum_{\mathbf{z}_N} \alpha(\mathbf{z}_N)/p(\mathbf{Y}) = 1$$

we can obtain

$$p(\mathbf{Y}) = \sum_{\mathbf{z}_N} \alpha(\mathbf{z}_N)$$

Recursion Relations for $\xi(\mathbf{z}_n, \mathbf{z}_{n-1})$

By definition,

$$\xi(\mathbf{z}_n, \mathbf{z}_{n-1}) = p(\mathbf{z}_{n-1}, \mathbf{z}_n|\mathbf{Y})$$

$$= \frac{p(\mathbf{Y}|\mathbf{z}_{n-1}, \mathbf{z}_n)p(\mathbf{z}_{n-1}, \mathbf{z}_n)}{p(\mathbf{Y})}$$

$$= \frac{p(\mathbf{y}_1, \ldots, \mathbf{y}_{n-1}|\mathbf{z}_{n-1})p(\mathbf{y}_n|\mathbf{z}_n)p(\mathbf{y}_{n+1}, \ldots, \mathbf{y}_N|\mathbf{z}_n)p(\mathbf{z}_{n-1}|\mathbf{z}_n)p(\mathbf{z}_{n-1})}{p(\mathbf{Y})}$$

$$= \frac{\alpha(\mathbf{z}_{n-1})p(\mathbf{y}_n|\mathbf{z}_n)p(\mathbf{z}_{n-1}|\mathbf{z}_n)\beta(\mathbf{z}_n)}{p(\mathbf{Y})}$$

EM Algorithm on HMM

• Initialize $\theta = \{\pi, \mathbf{A}, \phi\}$ to θ^{old} using uniform or random distributions. In case of Gaussian ϕ, $K - means$ algorithm can be used to identify the initial μ_k and Σ_k by using cluster means and covariances, respectively.

• **E-Step**

Compute

$$\alpha(\mathbf{z}_n) = p(\mathbf{y}_n|\mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} \alpha(\mathbf{z}_{n-1}) p(\mathbf{z}_n|\mathbf{z}_{n-1}),\ \beta(\mathbf{z}_n) = \sum_{\mathbf{z}_{n+1}} \beta(\mathbf{z}_{n+1}) p(\mathbf{y}_{n+1}|\mathbf{z}_{n+1}) p(\mathbf{z}_{n+1}|\mathbf{z}_n)$$

$$p(\mathbf{Y}) = \sum_{\mathbf{z}_N} \alpha(\mathbf{z}_N)$$

and evaluate

$\gamma(z_{nk}) = \alpha(\mathbf{z}_n)\beta(\mathbf{z}_n)/p(\mathbf{Y})$ and $\xi(\mathbf{z}_n, \mathbf{z}_{n-1}) = \alpha(\mathbf{z}_{n-1}) p(\mathbf{y}_n|\mathbf{z}_n) p(\mathbf{z}_{n-1}|\mathbf{z}_n) \beta(\mathbf{z}_n)/p(\mathbf{Y})$.

• **M-Step**

Determine θ^{new} using

$$\pi_k = \gamma(z_{1k}) / \sum_{j=1}^{K} \gamma(z_{1j}) \qquad A_{jk} = \sum_{n=2}^{N} \xi(z_{n-1j}, z_{nk}) / \sum_{l=1}^{K} \sum_{n=2}^{N} \xi(z_{n-1j}, z_{nl})$$

and the parameter update equations for distribution ϕ_k.

If ϕ_k is Gaussian *i.e.* $\phi_k = \mathcal{N}(\mu_k, \Sigma_k)$

$$\mu_k = \sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{y}_n / \sum_{n=1}^{N} \gamma(z_{nk}) \qquad \Sigma_k = \sum_{n=1}^{N} \gamma(z_{nk}) (\mathbf{y}_n - \mu_k)(\mathbf{y}_n - \mu_k)^T / \sum_{n=1}^{N} \gamma(z_{nk})$$

If ϕ_k is multinomial $\mu_{ik} = \sum_{n=1}^{N} \gamma(z_{nk}) y_{ni} / \sum_{n=1}^{N} \gamma(z_{nk})$

• Iterate upon E and M steps until convergence.

```
function [sequences,Z]=hmm_sequence_1
% generates nucleotide sequence using stationary hidden markov model for clusters
N= 30; % sequence length
Ns = 30; % number of gene sequences
Nuc = {'A','G','C','T'}; % nucleotides
pycz = [ 0.4 0.1 0.1 0.4; 0.1 0.4 0.4 0.1]'; % p(y|z)
PI  = [0.3 0.7]; % prior probability for clusters p(z_1)
CPI =  cumsum(PI); % cumulative function for PI
pzcz_1 = [0.3 0.7 ; 0.5 0.5]'; %p(z_n|z_n-1)
s = 1;
C(1)= sum( (rand()>=CPI))+1;   % choose a cluster with prior probability PI
% Generate a stationary HMM sequence in cluster c
S(1) = randgen(pycz(:,C(1)));  %randomly determine the initial state based on z_1
for n=2:N,   C(n) = randgen(pzcz_1(:,C(n-1))); S(n) = randgen(pycz(:,C(n)));  end;
Seq(s) = join(Nuc(S));
sequences{1,s}= regexprep(Seq{s}, '\s', '');
Z(s,:) = C;
for s=2:Ns,
% choose a cluster with prior probability PI
%  if rand(1,1) > 0.4,  c = 1; else   c = 2; end;   C(s) = c;
C(s)  = sum( (rand()>=CPI))+1;
% Generate a stationary HMM sequence in cluster c
S(1) = randgen(pycz(:,C(s))); %randomly determine the initial stat
for n=2:N,   C(n) = randgen(pzcz_1(:,C(n-1))); S(n) = randgen(pycz(:,C(n)));  end;
Seq(s) = join(Nuc(S));  sequences{1,s} = regexprep(Seq{s}, '\s', '');
Z(s,:) = C;
end;
save ('mysequences_1','sequences','C','pycz','PI','pzcz_1');
```

```
% code for HMM clustering for mutinomially distributed observed variables
% suitable for nucleotide sequences  {'A','G','C','T'}
function [ loglikelihood,gamma] = hmm_a_b
load mysequences_1
Ns= length(sequences);
N = length(sequences{1});
D = 4 ;
K =2;
Max_Iter = 100;
%PI_0  = [0.5 0.5]; % prior probability for clusters
for n=1:Ns % convert the characters into (arbitrary) numerical values (from 1 to 4)
y{n}(findstr(sequences{n},'A'))=1;
y{n}(findstr(sequences{n},'G'))=2;
y{n}(findstr(sequences{n},'C'))=3;
y{n}(findstr(sequences{n},'T'))=4;
end;
PI, pzcz_1, pycz
% random initialisation:
pzcz_1= condp(rand(K,K));  % transition distribution p(z_n|z_n-1)
%pzcz_1 = eye(K,K);
pycz= condp(rand(D,K)); % emission distribution p(y_n|z_n)
pz1= condp(rand(K,1)); %   p(z_1)
for iter =1:Max_Iter,
A=zeros(K,K); PI=zeros(K,1); MU=zeros(D,K);
loglikelihood(iter)=0;
```

```
for ns=1:Ns,
% E-STEP
% computing alpha
logalpha(:,1) = log(pycz(y{ns}(1),:)'.*pz1 ) ; % pycz => p(y|z), pz1 => p(z_1)
for n=2:N,
logalpha(:,n)=logsumexp(repmat(logalpha(:,n-1),1,K),repmat(pycz(y{ns}(n),:),K,1).*pzcz_1');
end;
loglik = logsumexp(logalpha(:,N),ones(K,1)); % log likelihood
loglikelihood(iter) = loglikelihood(iter) + loglik;
% computing beta
logbeta(:,N)=zeros(K,1); % beta(z_N) =1;
for n=N:-1:2
logbeta(:,n-1)=logsumexp(repmat(logbeta(:,n),1,K),repmat(pycz(y{ns}(n),:)',1,K).*pzcz_1);
end
for n=1:N,  % computing gamma(z_n)  = p(z_n|Y)
loggamma(:,n)=logalpha(:,n)+logbeta(:,n); % alpha-beta approach
gamma(:,n)=condexp(loggamma(:,n));
end
for n=1:N-1,% computing xi
alpha_n1=condexp(logalpha(:,n)) ; beta_n = condexp(logbeta(:,n+1));
xi_un_norm =  repmat(alpha_n1,1,K).*pzcz_1'.*repmat(pycz(y{ns}(n+1),:).*beta_n',K,1);
xi(:,:,n) = xi_un_norm./sum(sum(xi_un_norm));
end
% M-Step Statistics Accumulation
A = A + sum(xi,3); PI = PI + gamma(:,1); for n=1:N,MU(y{ns}(n),:)=MU(y{ns}(n),:)+gamma(:,n)'; end
end; % ns loop
% M-step
pzcz_1=condp(A'); pycz=condp(MU); pz1=condp(PI);
end; % iter loop
pz1, pzcz_1, pycz,plot(loglikelihood)
```