NUS National University of Singapore | School of Computing

CS5340
Uncertainty Modeling in AI

Lecture 9: Hidden Markov Models (HMM)

Asst. Prof. Lee Gim Hee

AY 2018/19

Semester 1

Course Schedule

Week	Date	Topic	Remarks
1	15 Aug	Introduction to probabilities and probability distributions	
2	22 Aug	Fitting probability models	Hari Raya Haji*
3	29 Aug	Bayesian networks (Directed graphical models)	
4	05 Sep	Markov random Fields (Undirected graphical models)	
5	12 Sep	I will be traveling	No Lecture
6	19 Sep	Variable elimination and belief propagation	
-	26 Sep	Recess week	No lecture
7	03 Oct	Factor graph and the junction tree algorithm	
8	10 Oct	Parameter learning with complete data	
9	17 Oct	Mixture models and the EM algorithm	
10	24 Oct	Hidden Markov Models (HMM)	
11	31 Oct	Monte Carlo inference (Sampling)	
12	07 Nov	Variational inference	
13	14 Nov	Graph-cut and alpha expansion	

* **Make-up lecture**: 25 Aug (Sat), 9.30am-12.30pm, LT 15

Acknowledgements

- A lot of slides and content of this lecture are adopted from:

1. "Pattern Recognition and Machine Learning", Christopher Bishop, Chapter 13.

2. "Machine Learning – A Probabilistic Perspective", Kevin Murphy, Chapter 17.

3. "An Introduction to Probabilistic Graphical Models", Michael I. Jordan, Chapters 12. http://people.eecs.berkeley.edu/~jordan/prelims/chapter12.pdf

4. "Computer Vision: Models, Learning and Inference", Simon Prince, Chapter 11.

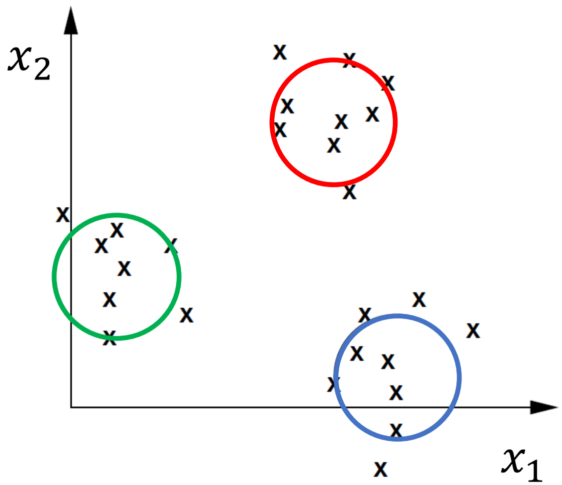
Learning Outcomes

- Students should be able to:

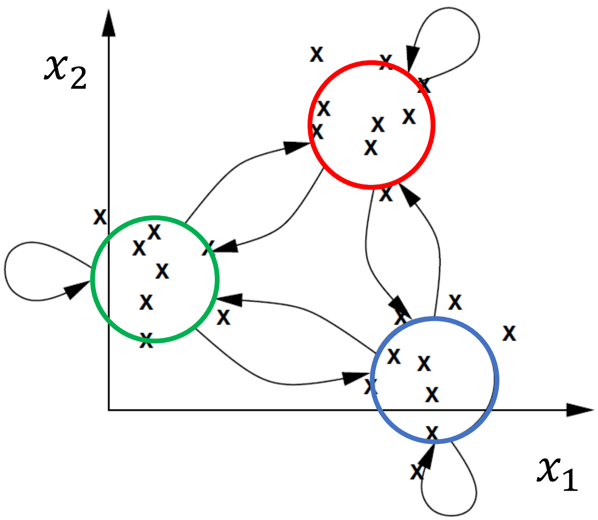
1. Describe the joint distribution of a HMM with the transition and emission probabilities.
2. Use the EM algorithm for maximum likelihood estimation of the latent variables and unknown parameters in the HMM.
3. Use the forward-backward algorithm to evaluate the EM algorithm.
4. Apply the Viterbi algorithm to find the maximal probability and its configuration.

Sequential Data

Independent choice of mixture component



Choice of current mixture component is dependent on the previous observation



- In EM, we focused on mixture models, where the choice of mixture component for each observation is independent.

- HMM is an extension of the mixture model, where choice of mixture component depends on the choice of component for the previous observation.

Image Source: "An introduction to probabilistic graphical models", Michael I. Jordan, 2002.

Hidden Markov Model (HMM)

- An HMM is a natural generalization of a mixture model – can be viewed as a "dynamic" mixture model.
- For each observed variable X_n, there is corresponding latent variable Z_n (which may be of different type or dimensionality to X_n).
- The latent variables $\{Z_1, \dots Z_{n-1}, Z_n \dots\}$ form a Markov Chain, where the current state Z_n is dependent on the previous state Z_{n-1}.

HMM: State-Space Model

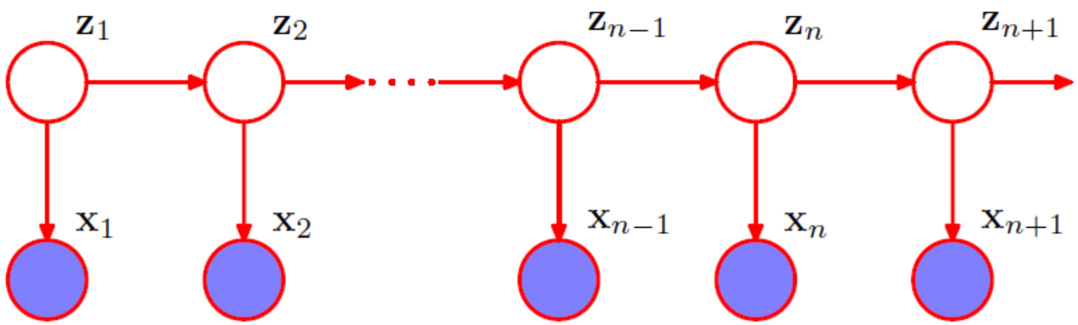
- The Markov chain of latent variables gives rise to the graphical structure known as a state space model.

- It satisfies the key conditional independence property that Z_{n-1} and Z_{n+1} are independent given Z_n, so that:

$$\mathbf{z}_{n+1} \perp\!\!\!\perp \mathbf{z}_{n-1} \mid \mathbf{z}_n.$$

HMM: Joint Distribution

1st order Markov Chain

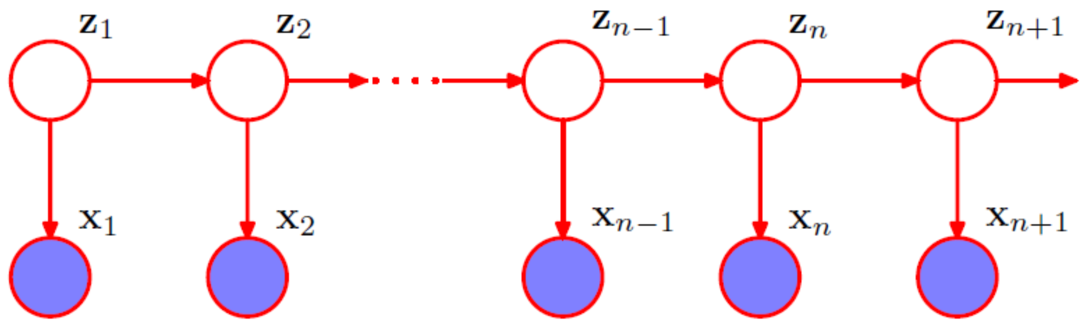


- The joint distribution is given by:

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_N, \mathbf{z}_1, \ldots, \mathbf{z}_N) = p(\mathbf{z}_1) \left[\prod_{n=2}^{N} p(\mathbf{z}_n | \mathbf{z}_{n-1}) \right] \prod_{n=1}^{N} p(\mathbf{x}_n | \mathbf{z}_n).$$

Image source: “Pattern recognition and machine learning”, Christopher Bishop

HMM: Joint Distribution



- **HMM** (covered in this lecture): Latent variables must be discrete, observed variable can be either discrete or continuous.

- **Linear dynamic system** (not covered in this course): Latent and observed variables are both continuous; linear-Gaussian if both are Gaussian.

Image source: "Pattern recognition and machine learning", Christopher Bishop

Hidden Markov Model (HMM)

Example:
Speech Recognition

- Given an audio waveform, the goal is to robustly extract and recognize any spoken words

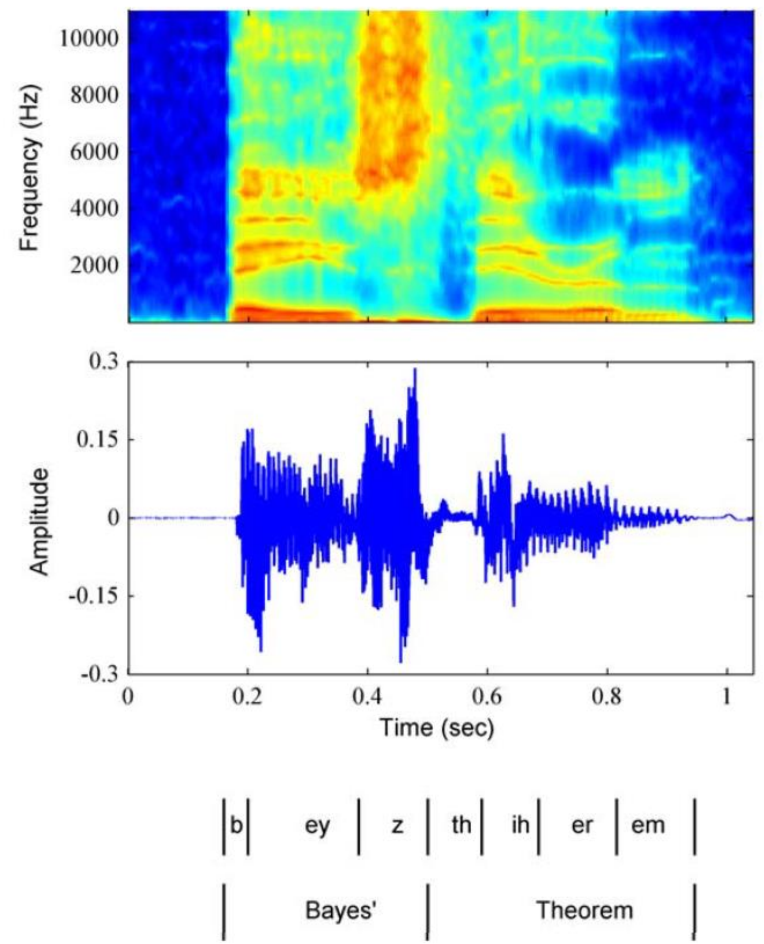


Image source: “Pattern recognition and machine learning”, Christopher Bishop

Hidden Markov Model (HMM)

Example: Target Tracking

- Estimate motion of targets in 3D world from indirect, potentially noisy measurements

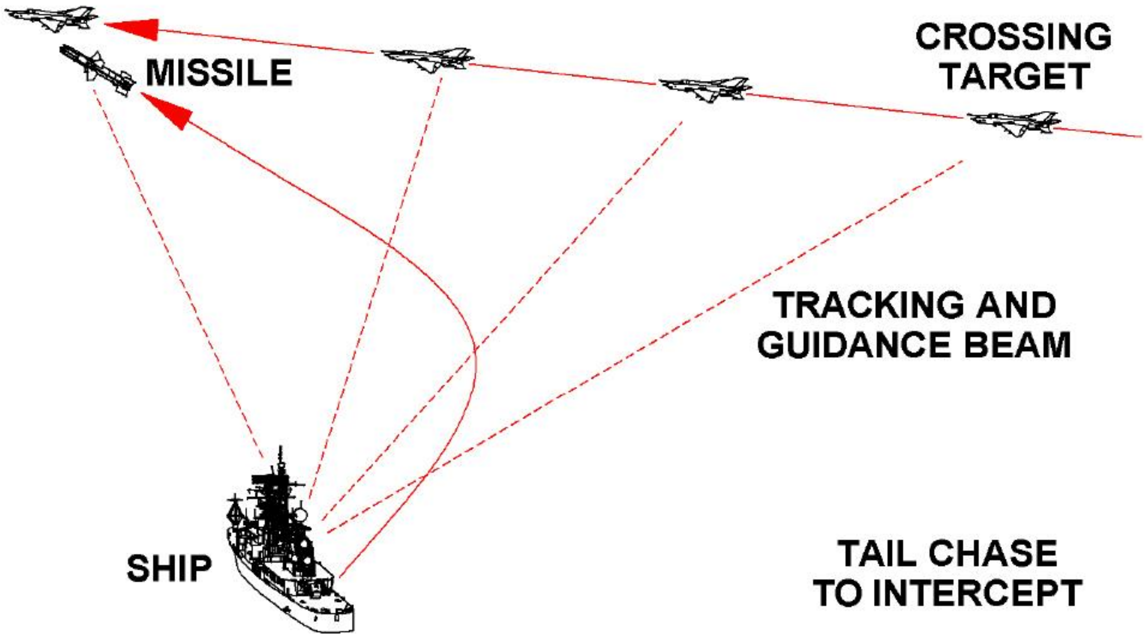


Image source: http://www.okieboat.com/History%20guidance%20and%20homing.html

Hidden Markov Model (HMM)

Example: Robotic SLAM

Simultaneous Localization and Mapping (SLAM) – Pose and world geometry estimation as robot moves.

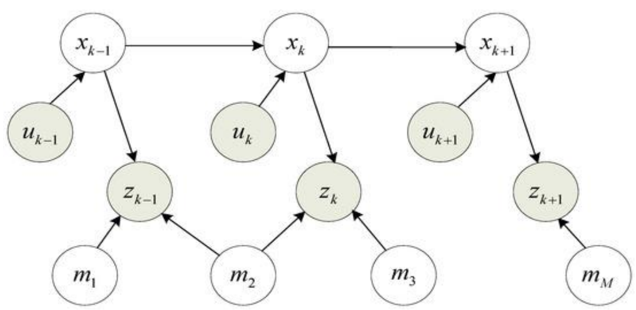
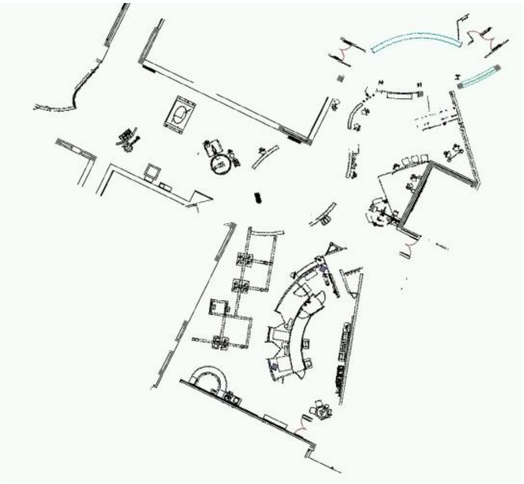
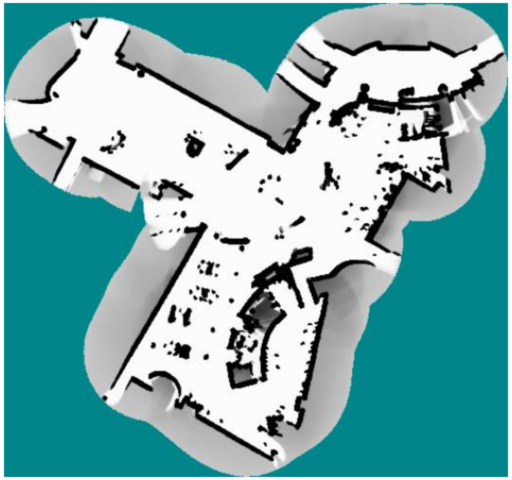


Image source: http://www.mdpi.com/1424-8220/17/5/1174

CAD Map



Estimated Map



(S. Thrun, San Jose Tech Museum)

Hidden Markov Model (HMM)

Example: Financial Forecasting

- Predict future market behavior from historical data.

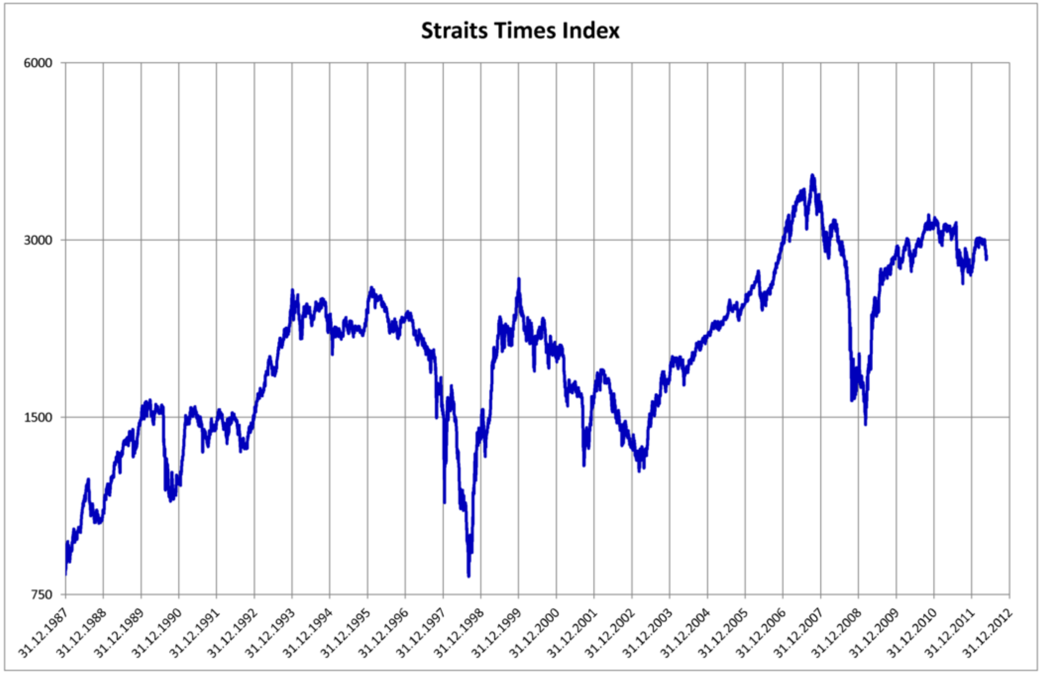


Image source: https://en.wikipedia.org/wiki/Straits_Times_Index

Transition Probabilities

- 1-of-K coding scheme for the discrete latent variables Z_n.

- This is to describe which component of the mixture is responsible for generating the corresponding observation X_n.

- Z_n depends on the state of the previous latent variable Z_{n-1} through a conditional distribution:

$$p(z_n | z_{n-1}).$$

Transition Probabilities

- Since $z_n \in \{0,1\}^K$, $p(z_n|z_{n-1})$ corresponds to a $K \times K$ matrix A, where the elements are known as transition probabilities.

- Properties of the state transition matrix $A \in \mathbb{R}^{K \times K}$:

1. $A_{jk} \equiv p(z_{nk} = 1 | z_{n-1,j} = 1)$

2. $0 \leqslant A_{jk} \leqslant 1$, with $\sum_k A_{jk} = 1$

3. $K(K-1)$ independent parameters.

Transition Probabilities

- Transition diagram showing a model whose latent variables have three possible states corresponding to the three boxes.

- The black lines denote the elements of the transition matrix A_{jk}.

This is NOT a graphical model!!!

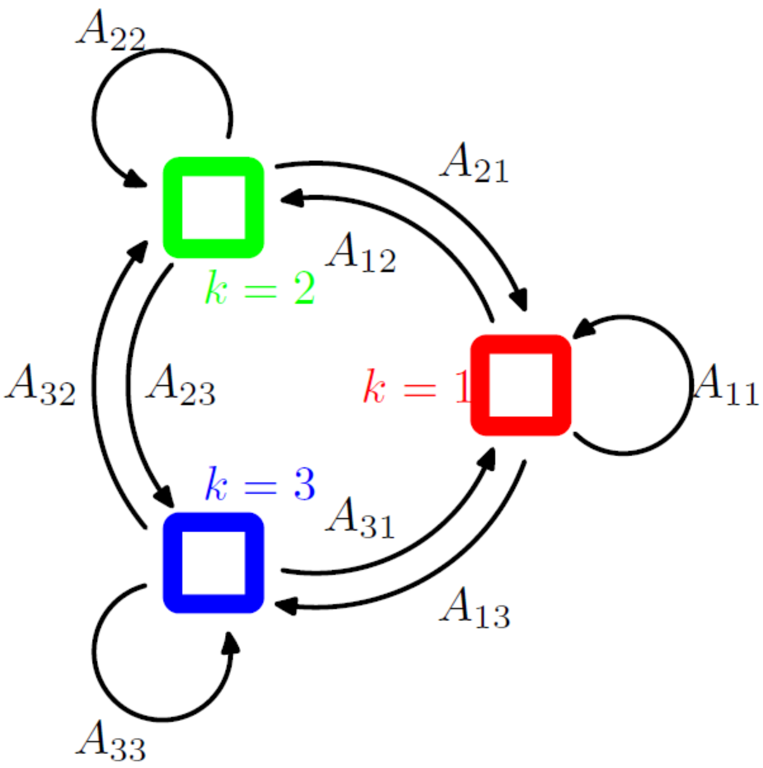


Image source: “Pattern recognition and machine learning”, Christopher Bishop

Transition Probabilities

- We obtain a lattice or trellis representation of the latent states by unfolding the state transition diagram.

- Each column in the diagram corresponds to one of the latent variables Z_n.

This is NOT a graphical model!!!

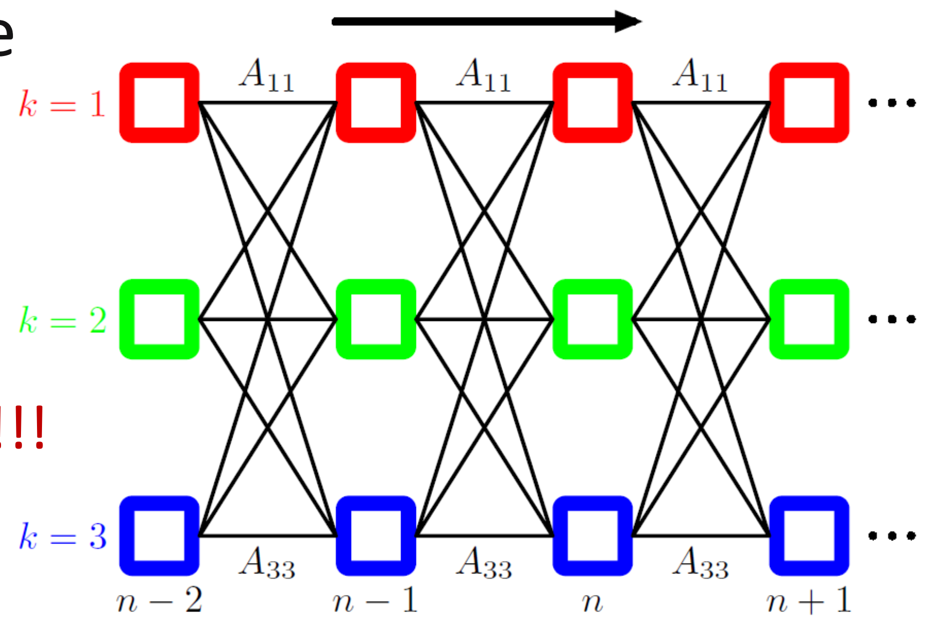


Image source: "Pattern recognition and machine learning", Christopher Bishop

Transition Probabilities

- We can then write the conditional distribution explicitly in the form:

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

- Initial latent variable Z_1 does not have a parent node, it is represented as a categorical distribution:

$$p(\mathbf{z}_1|\boldsymbol{\pi}) = \prod_{k=1}^{K} \pi_k^{z_{1k}}, \text{ where } \sum_k \pi_k = 1.$$

Emission Probabilities

- Emission probabilities: conditional distributions of the observed variable X_n, given the latent variable Z_n,

$$p(x_n|z_n, \phi).$$

- $\phi = \{\phi_1, \dots \phi_K\}$ is a set of parameters governing the distribution.

- Can be Gaussians if X_n is continuous, or conditional probability tables if X_n is discrete.

Emission Probabilities

- The emission probabilities is given by:

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

- For a given value of ϕ, $p(x_n|z_n, \phi)$ consists of a vector of K numbers corresponding to the K possible states of the binary vector Z_n.

Homogenous Model

- All of the conditional distributions governing the latent variables share the same parameters A.
- Similarly, all of the emission distributions share the same parameters ϕ.
- **Note:** Mixture model for an i.i.d. data set if all parameters A_{jk} are the same, i.e. $p(z_n|z_{n-1})$ is independent of Z_{n-1}.

HMM: Joint Probability Revisited

- The joint probability distribution over both latent and observed variables is given by:

$$p(\mathbf{X}, \mathbf{Z} | \boldsymbol{\theta}) = p(\mathbf{z}_1 | \boldsymbol{\pi}) \underbrace{\left[\prod_{n=2}^{N} p(\mathbf{z}_n | \mathbf{z}_{n-1}, \mathbf{A}) \right]}_{\text{Transition probabilities}} \underbrace{\prod_{m=1}^{N} p(\mathbf{x}_m | \mathbf{z}_m, \phi)}_{\text{Emission probabilities}}$$

- where $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}, \mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_N\},$ and $\boldsymbol{\theta} = \{\boldsymbol{\pi}, \mathbf{A}, \phi\}$ denotes the set of parameters governing the model.

Maximum Likelihood for HMM

- The likelihood function is obtained from the joint distribution by marginalizing over the latent variables:

$$p(\mathbf{X}|\boldsymbol{\theta}) = \sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$$

- Cannot treat each of the sum over Z_n independently because $p(X, Z|\theta)$ does not factorize over $Z_1, \dots, Z_n$.

- Performing the sums over all N variables results in a exponential complexity of $O(K^N)$.

- Moreover, direct maximization will lead to no closed-form solutions!

MLE for HMM: EM Algorithm

- We turn to the EM algorithm to find an efficient framework for maximizing the likelihood function in hidden Markov models.

- The EM algorithm starts with initialization of the model parameters, which we denote by θ^{old}.

- In the **E step**, we take these parameter values and find the posterior distribution of the latent variables $p(Z|X, \theta^{old})$.

MLE for HMM: EM Algorithm

- Use $p(Z|X,\theta^{old})$ to evaluate the expectation of the log complete-data likelihood defined by:

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}}) \ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}).$$

- Let us now denote the marginal posterior distribution of a latent variable Z_n as:

$$\gamma(\mathbf{z}_n) = p(\mathbf{z}_n|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})$$

- And the joint posterior distribution of two successive latent variables (Z_{n-1}, Z_n) as:

$$\xi(\mathbf{z}_{n-1}, \mathbf{z}_n) = p(\mathbf{z}_{n-1}, \mathbf{z}_n|\mathbf{X}, \boldsymbol{\theta}^{\text{old}}).$$

MLE for HMM: EM Algorithm

- For each value of n, we can store $\gamma(z_n)$ using a set of K nonnegative numbers that sum to unity.
- Similarly, we can store $\xi(z_{n-1}, z_n)$ using a $K \times K$ matrix of nonnegative numbers that again sum to unity.
- We shall also use $\gamma(z_{nk})$ to denote the conditional probability of $z_{nk} = 1$, with a similar use of notation for $\xi(z_{n-1,j}, z_{nk})$.

MLE for HMM: EM Algorithm

- Because the expectation of a binary random variable is just the probability that it takes the value of 1, we have:

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

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

MLE for HMM: EM Algorithm

- Putting everything together:

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}}) \ln \boxed{p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})}.$$

where

$$\boxed{p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})} = \underbrace{p(\mathbf{z}_1|\boldsymbol{\pi})}_{\prod_{k=1}^{K} \pi_k^{z_{1k}}} \left[\prod_{n=2}^{N} \underbrace{p(\mathbf{z}_n|\mathbf{z}_{n-1}, \mathbf{A})}_{\prod_{k=1}^{K} \prod_{j=1}^{K} A_{jk}^{z_{n-1,j} z_{nk}}} \right] \prod_{m=1}^{N} \underbrace{p(\mathbf{x}_m|\mathbf{z}_m, \boldsymbol{\phi})}_{\prod_{k=1}^{K} p(\mathbf{x}_m|\boldsymbol{\phi}_k)^{z_{mk}}}$$

MLE for HMM: EM Algorithm

- We get:

$$Q(\theta, \theta^{old}) = \sum_{Z} p(Z|X, \theta^{old}) \left\{ \sum_{k=1}^{K} z_{1k} \ln \pi_k + \sum_{n=2}^{N} \sum_{k=1}^{K} \sum_{j=1}^{K} z_{n-1,j} z_{nk} \ln A_{jk} + \sum_{m=1}^{N} \sum_{k=1}^{K} z_{mk} \ln p(x_m|\phi_k) \right\}$$

- Which evaluates to:

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = \sum_{k=1}^{K} \gamma(z_{1k}) \ln \pi_k + \sum_{n=2}^{N} \sum_{j=1}^{K} \sum_{k=1}^{K} \xi(z_{n-1,j}, z_{nk}) \ln A_{jk} + \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_{nk}) \ln p(\mathbf{x}_n|\boldsymbol{\phi}_k).$$

MLE for HMM: EM Algorithm

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

- The goal of the E step will be to evaluate the quantities $\gamma(z_n)$ and $\xi(z_{n-1,j}, z_{nk})$ efficiently!
- We shall discuss this shortly in the Forward-backward algorithm.

MLE for HMM: EM Algorithm

- In the **M step**, we maximize $Q(\theta, \theta^{old})$ w.r.t. the parameters $\theta = \{\pi, A, \phi\}$ in which we treat $\gamma(z_n)$ and $\xi(z_{n-1}, z_n)$ as constant.

- Maximization with respect to π and A is easily achieved using appropriate Lagrange multipliers with:

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

MLE for HMM: EM Algorithm

Notes:

1. EM algorithm must be initialized by choosing starting values for π and A with the summation constraints.
2. Initial values for π and A CANNOT be set to zero, they will remain zero in subsequent EM updates.
3. Typically, we set random starting values for these parameters s.t. the summation and non-negativity constraints.

MLE for HMM: EM Algorithm

- Assuming that $p(x_n|\phi_k) = \mathcal{N}(x_n|\mu_k, \Sigma_k)$, maximizing of $Q(\theta, \theta^{old})$ w.r.t. $\phi_k = \{\mu_k, \Sigma_k\}$ gives:

$$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk})\mathbf{x}_n}{\sum_{n=1}^{N} \gamma(z_{nk})}, \quad \boldsymbol{\Sigma}_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk})(\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}}}{\sum_{n=1}^{N} \gamma(z_{nk})}.$$

MLE for HMM: EM Algorithm

- For discrete multinomial X_n, i.e. $x_n \in \mathbb{R}^D$ and $x_{ni} \in \{0,1\}$ and $\sum_{i=1}^{D} x_{ni} = 1$, the conditional distribution of the observations takes the form:

$$p(\mathbf{x}|\mathbf{z}) = \prod_{i=1}^{D} \prod_{k=1}^{K} \mu_{ik}^{x_i z_k}$$

- and the corresponding M-step equations are given by:

$$\mu_{ik} = \frac{\sum_{n=1}^{N} \gamma(z_{nk}) x_{ni}}{\sum_{n=1}^{N} \gamma(z_{nk})}.$$

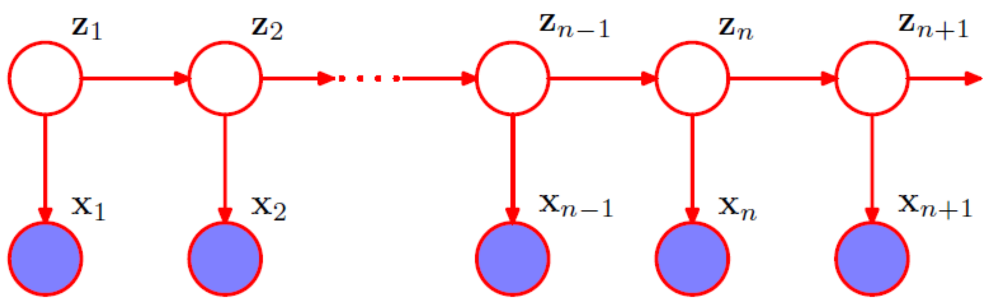
The Forward-Backward Algorithm

- We use the forward-backward algorithm to compute $\gamma(z_n)$ and $\xi(z_{n-1}, z_n)$ efficiently.

- Also known as the Baum-Welch algorithm.

- Many variants of the algorithm, we shall focus on the most widely used of these, known as the alpha-beta algorithm.

Conditional Independence Properties



$$\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$$

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

Using d-separation, we get these C.I. from the DGM of HMM.

Image source: "Pattern recognition and machine learning", Christopher Bishop

The Forward-Backward Algorithm

- Evaluating $\gamma(z_{nk})$: we are interested in finding the posterior distribution $p(z_n|x_1, \dots, x_N)$.

$$\gamma(\mathbf{z}_n) = p(\mathbf{z}_n|\mathbf{X}) = \frac{p(\mathbf{X}|\mathbf{z}_n)p(\mathbf{z}_n)}{p(\mathbf{X})}$$ (Bayes' Rule)

$$= \frac{p(x_1, \dots, x_n|z_n)p(x_{n+1}, \dots, x_N|z_n)p(z_n)}{p(\mathrm{X})}$$ (Conditional Independence)

$$= \frac{\frac{p(\mathrm{x}_1, \dots, \mathrm{x}_n, \mathrm{z}_n)}{\cancel{p(z_n)}}\cancel{p(z_n)}p(x_{n+1}, \dots, x_N|z_n)}{p(\mathrm{X})}$$ (Product Rule)

$$= \frac{p(\mathbf{x}_1, \dots, \mathbf{x}_n, \mathbf{z}_n)p(\mathbf{x}_{n+1}, \dots, \mathbf{x}_N|\mathbf{z}_n)}{p(\mathbf{X})} = \frac{\alpha(\mathbf{z}_n)\beta(\mathbf{z}_n)}{p(\mathbf{X})}$$

The Forward-Backward Algorithm

We have defined:

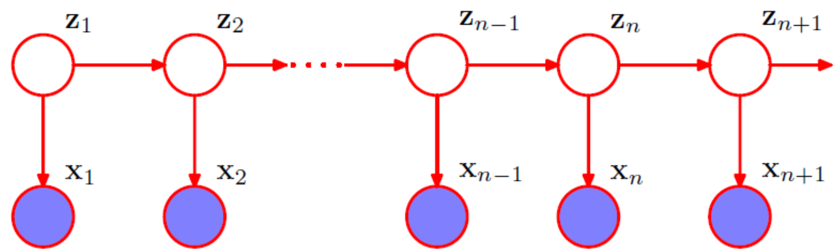
$$\begin{aligned}\alpha(\mathbf{z}_n) &\equiv p(\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{z}_n) \\ \beta(\mathbf{z}_n) &\equiv p(\mathbf{x}_{n+1}, \ldots, \mathbf{x}_N | \mathbf{z}_n)\end{aligned}$$

- $\alpha(z_n)$ represents the joint probability of observing all of the given data up to time n and value of Z_n.
- $\beta(z_n)$represents the conditional probability of all future data from time $n+1$ up to N given the value of Z_n.
- $\alpha(z_n)$ and $\beta(z_n)$ each represent set of K numbers, one for each of the possible settings of the 1-of-K coded binary vector Z_n.

The Forward-Backward Algorithm

Forward recursion:

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



Proof:

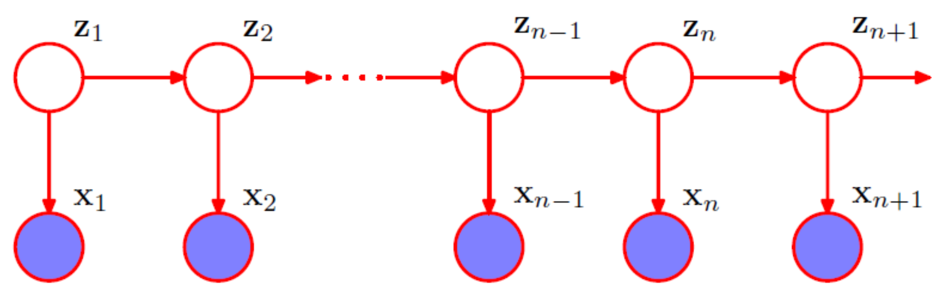
$$\begin{aligned}
\alpha(\mathbf{z}_n) &= p(\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{z}_n) \\
&= p(\mathbf{x}_1, \ldots, \mathbf{x}_n|\mathbf{z}_n) p(\mathbf{z}_n) && \text{(product rule)} \\
&= p(\mathbf{x}_n|\mathbf{z}_n) p(\mathbf{x}_1, \ldots, \mathbf{x}_{n-1}|\mathbf{z}_n) p(\mathbf{z}_n) && \text{(conditional independence)} \\
&= p(\mathbf{x}_n|\mathbf{z}_n) p(\mathbf{x}_1, \ldots, \mathbf{x}_{n-1}, \mathbf{z}_n) && \text{(product rule)} \\
&= p(\mathbf{x}_n|\mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} p(\mathbf{x}_1, \ldots, \mathbf{x}_{n-1}, \mathbf{z}_{n-1}, \mathbf{z}_n) && \text{(marginalization)} \\
&= p(\mathbf{x}_n|\mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} p(\mathbf{x}_1, \ldots, \mathbf{x}_{n-1}, \mathbf{z}_n|\mathbf{z}_{n-1}) p(\mathbf{z}_{n-1}) && \text{(product rule)} \\
&= p(\mathbf{x}_n|\mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} p(\mathbf{x}_1, \ldots, \mathbf{x}_{n-1}|\mathbf{z}_{n-1}) p(\mathbf{z}_n|\mathbf{z}_{n-1}) p(\mathbf{z}_{n-1}) && \text{(conditional independence)} \\
&= p(\mathbf{x}_n|\mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} \overbrace{\boxed{p(\mathbf{x}_1, \ldots, \mathbf{x}_{n-1}, \mathbf{z}_{n-1})}}^{\alpha(z_{n-1})} p(\mathbf{z}_n|\mathbf{z}_{n-1}) && \text{(product rule)}
\end{aligned}$$

Image source: “Pattern recognition and machine learning”, Christopher Bishop

The Forward-Backward Algorithm

Forward recursion:

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



- There are K terms in the summation over Z_{n-1}.
- RHS has to be evaluated for K values of Z_n.
- So each step of the forward recursion has computational cost of $O(K^2)$.

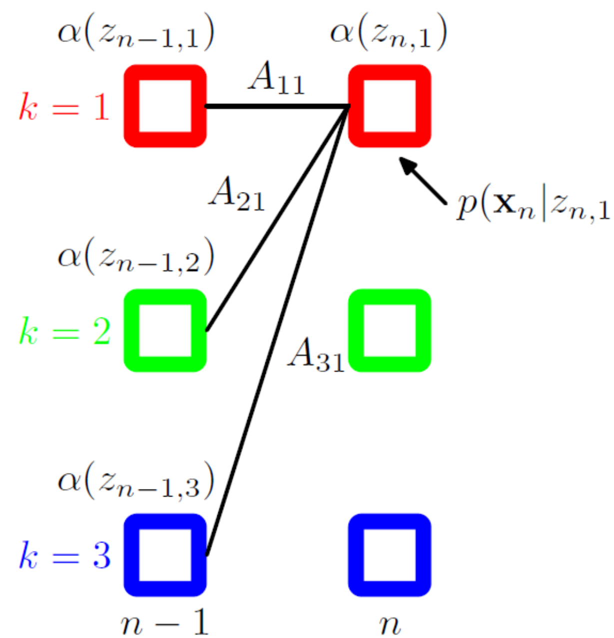
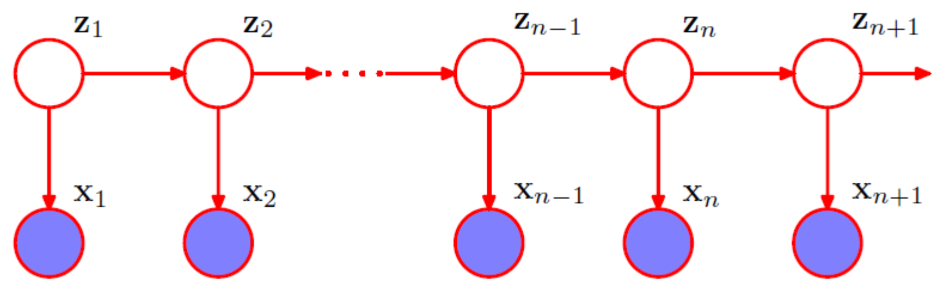


Image source: "Pattern recognition and machine learning", Christopher Bishop

The Forward-Backward Algorithm

Forward recursion:

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



$\alpha(z_{n,1})$ is obtained by taking:

1. elements $\alpha(z_{n-1,j})$ of $\alpha(z_{n-1})$ at step $n-1$
2. summing them up with weights given by A_{j1}, i.e. values of $p(z_n|z_{n-1})$
3. and then multiplying by the data contribution $p(x_n|z_{n1})$.

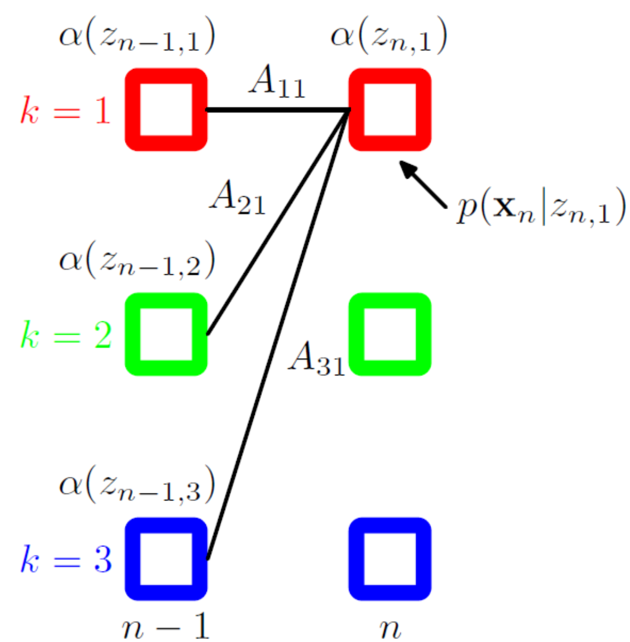
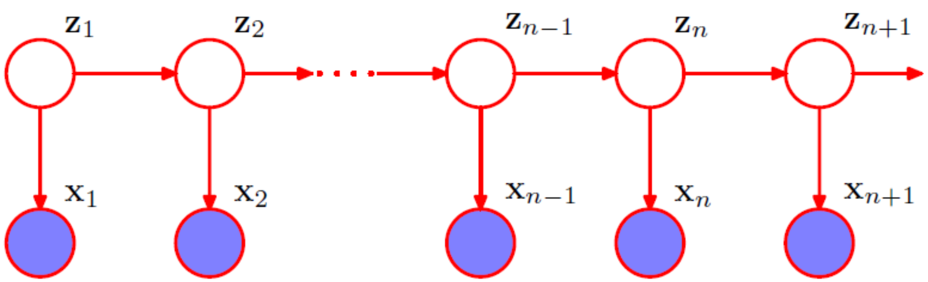


Image source: "Pattern recognition and machine learning", Christopher Bishop

The Forward-Backward Algorithm

Forward recursion:

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



- Initialization:

  $\alpha(z_{1k})$ for $k = 1, \ldots, K$ takes the value $\pi_k p(x_1|\phi_k)$

$$\alpha(\mathbf{z}_1) = p(\mathbf{x}_1, \mathbf{z}_1) = p(\mathbf{z}_1) p(\mathbf{x}_1|\mathbf{z}_1) = \prod_{k=1}^{K} \{\pi_k p(\mathbf{x}_1|\boldsymbol{\phi}_k)\}^{z_{1k}}$$

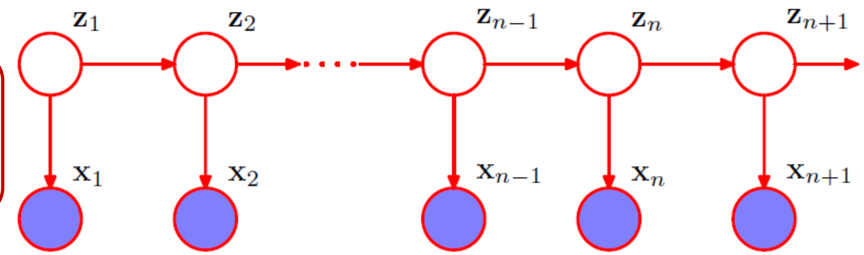
- Total complexity for the whole chain: $O(K^2 N)$.

Image source: "Pattern recognition and machine learning", Christopher Bishop

The Forward-Backward Algorithm

Backward recursion:

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



Proof:

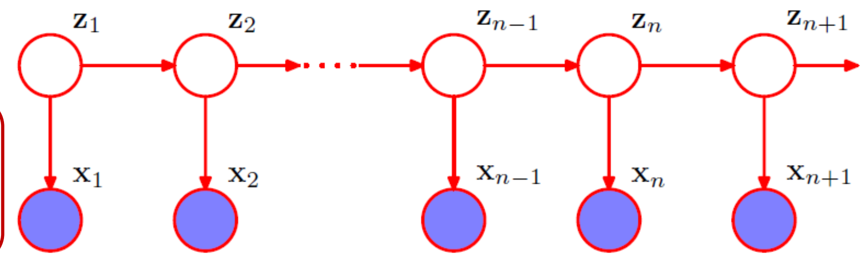
$$\begin{aligned}
\beta(\mathbf{z}_n) &= p(\mathbf{x}_{n+1}, \ldots, \mathbf{x}_N|\mathbf{z}_n) \\
&= \sum_{\mathbf{z}_{n+1}} p(\mathbf{x}_{n+1}, \ldots, \mathbf{x}_N, \mathbf{z}_{n+1}|\mathbf{z}_n) && \text{(marginalization)} \\
&= \sum_{\mathbf{z}_{n+1}} p(\mathbf{x}_{n+1}, \ldots, \mathbf{x}_N|\mathbf{z}_n, \mathbf{z}_{n+1}) p(\mathbf{z}_{n+1}|\mathbf{z}_n) && \text{(product rule)} \\
&= \sum_{\mathbf{z}_{n+1}} p(\mathbf{x}_{n+1}, \ldots, \mathbf{x}_N|\mathbf{z}_{n+1}) p(\mathbf{z}_{n+1}|\mathbf{z}_n) && \text{(conditional independence)} \\
&= \sum_{\mathbf{z}_{n+1}} \underbrace{\boxed{p(\mathbf{x}_{n+2}, \ldots, \mathbf{x}_N|\mathbf{z}_{n+1})}}_{\beta(z_{n+1})} p(\mathbf{x}_{n+1}|\mathbf{z}_{n+1}) p(\mathbf{z}_{n+1}|\mathbf{z}_n) && \text{(conditional independence)}
\end{aligned}$$

Image source: “Pattern recognition and machine learning”, Christopher Bishop

The Forward-Backward Algorithm

Backward recursion:

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



$\beta(z_{n1})$ is obtained by taking:

1. components $\beta(z_{n+1,k})$ of $\beta(z_{n+1})$ at step $n+1$
2. summing them up with weights given by the products of A_{1k}, i.e. values of $p(z_{n+1}|z_n)$
3. and the corresponding values of the emission density $p(x_{n+1}|z_{n+1,k})$.

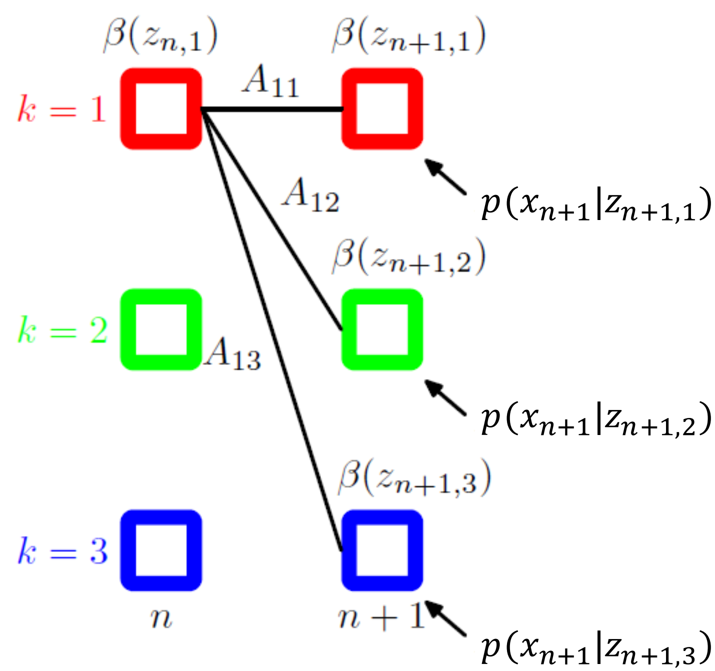


Image source: "Pattern recognition and machine learning", Christopher Bishop

The Forward-Backward Algorithm

- Initialization: $\beta(z_N) = 1$ for all settings of Z_N.

Proof:

Recall that we have:

$$\gamma(\mathbf{z}_n) = p(\mathbf{z}_n|\mathbf{X}) = \frac{p(\mathbf{X}|\mathbf{z}_n)p(\mathbf{z}_n)}{p(\mathbf{X})} = \frac{\alpha(\mathbf{z}_n)\beta(\mathbf{z}_n)}{p(\mathbf{X})}$$

Marginalizing both sides over z_n gives us:

$$p(\mathbf{X}) = \sum_{\mathbf{z}_n} \alpha(\mathbf{z}_n)\beta(\mathbf{z}_n)$$

In the case of $n = N$, we have:

$$p(\mathbf{X}) = \sum_{\mathbf{z}_N} \alpha(\mathbf{z}_N) = \sum_{z_N} p(x_1, \dots, x_N, z_N) \quad \Longrightarrow \beta(z_N) = 1$$

The Forward-Backward Algorithm

- Evaluating $\xi(z_{n-1}, z_n)$: which corresponds to the values of the conditional probabilities $p(z_{n-1}, z_n | X)$ for each of the $K \times K$ settings for (z_{n-1}, z_n).

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

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

(Bayes' rule)

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

(Conditional Independence)

$$= \frac{\overbrace{\alpha(\mathbf{z}_{n-1})}^{\text{Forward recursion}}p(\mathbf{x}_n|\mathbf{z}_n)p(\mathbf{z}_n|\mathbf{z}_{n-1})\overbrace{\beta(\mathbf{z}_n)}^{\text{Backward recursion}}}{p(\mathbf{X})}$$

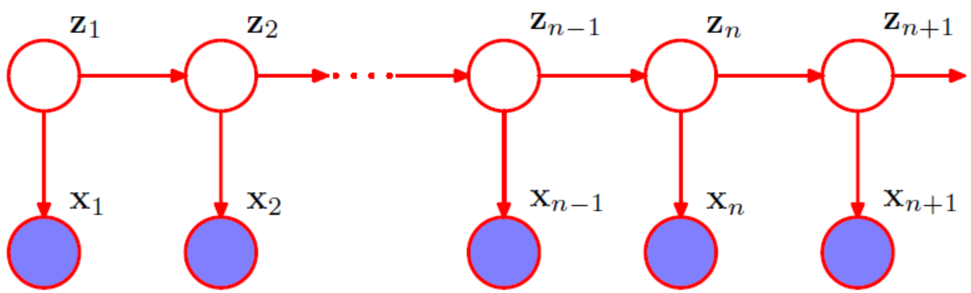
$$\sum_{\mathbf{z}_N} \alpha(\mathbf{z}_N)$$

Predictive Distribution

- Given the observed data is $X = \{x_1, \dots, x_N\}$, predict $x_N + 1$, e.g. financial forecasting.

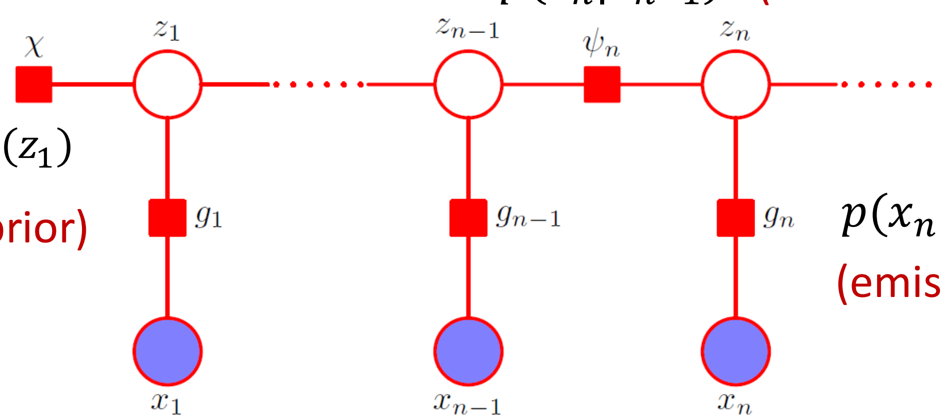
$$
\begin{aligned}
p(\mathbf{x}_{N+1}|\mathbf{X}) &= \sum_{\mathbf{z}_{N+1}} p(\mathbf{x}_{N+1}, \mathbf{z}_{N+1}|\mathbf{X}) && \text{(marginalization)} \\
&= \sum_{\mathbf{z}_{N+1}} p(\mathbf{x}_{N+1}|\mathbf{z}_{N+1}) p(\mathbf{z}_{N+1}|\mathbf{X}) && \text{(product rule)} \\
&= \sum_{\mathbf{z}_{N+1}} p(\mathbf{x}_{N+1}|\mathbf{z}_{N+1}) \sum_{\mathbf{z}_N} p(\mathbf{z}_{N+1}, \mathbf{z}_N|\mathbf{X}) && \text{(marginalization)} \\
&= \sum_{\mathbf{z}_{N+1}} p(\mathbf{x}_{N+1}|\mathbf{z}_{N+1}) \sum_{\mathbf{z}_N} p(\mathbf{z}_{N+1}|\mathbf{z}_N) p(\mathbf{z}_N|\mathbf{X}) && \text{(product rule)} \\
&= \sum_{\mathbf{z}_{N+1}} p(\mathbf{x}_{N+1}|\mathbf{z}_{N+1}) \sum_{\mathbf{z}_N} p(\mathbf{z}_{N+1}|\mathbf{z}_N) \frac{p(\mathbf{z}_N, \mathbf{X})}{p(\mathbf{X})} && \text{(product rule)} \\
&= \frac{1}{\underbrace{p(\mathbf{X})}_{\sum_{\mathbf{z}_N} \alpha(\mathbf{z}_N)}} \sum_{\mathbf{z}_{N+1}} p(\mathbf{x}_{N+1}|\mathbf{z}_{N+1}) \sum_{\mathbf{z}_N} p(\mathbf{z}_{N+1}|\mathbf{z}_N) \underbrace{\alpha(\mathbf{z}_N)}_{\text{Forward recursion}}
\end{aligned}
$$

HMM: Sum-of-Product Algorithm



Convert the Bayesian network into a factor graph

$p(z_n|z_{n-1})$ (transition probability)



$p(z_1)$

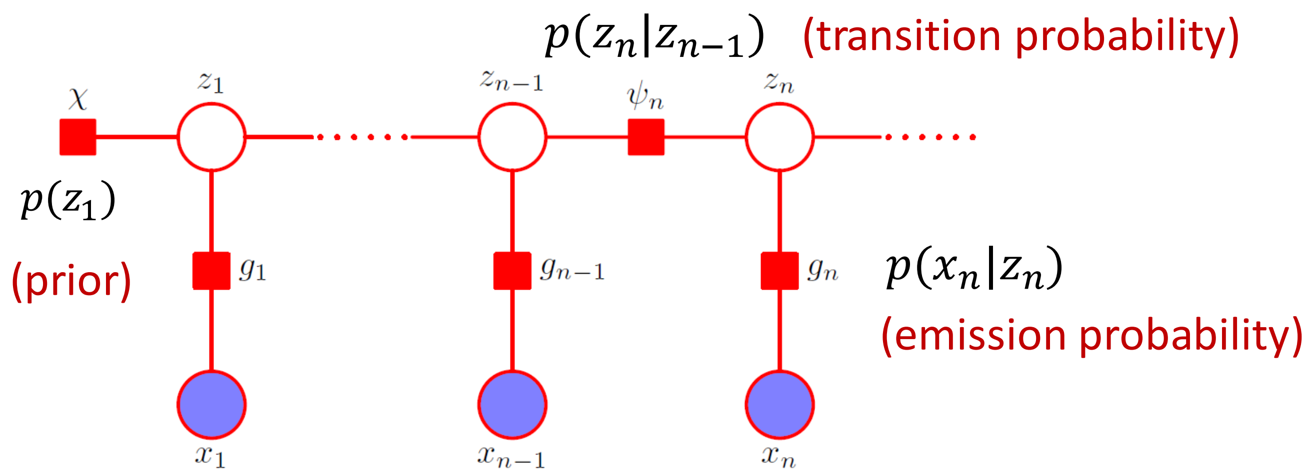
(prior)

$p(x_n|z_n)$

(emission probability)

Image source: “Pattern recognition and machine learning”, Christopher Bishop

HMM: Sum-of-Product Algorithm



Since we are always conditioning on $x_1, \dots, x_N$, we can simplify the factor graph by absorbing the emission probabilities into the transition probability factors.

$$
\begin{aligned}
h(\mathbf{z}_1) &= p(\mathbf{z}_1)p(\mathbf{x}_1|\mathbf{z}_1) \\
f_n(\mathbf{z}_{n-1}, \mathbf{z}_n) &= p(\mathbf{z}_n|\mathbf{z}_{n-1})p(\mathbf{x}_n|\mathbf{z}_n)
\end{aligned}
$$

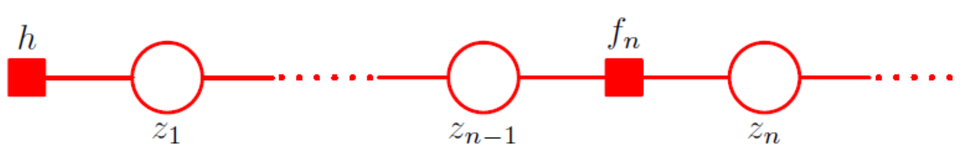
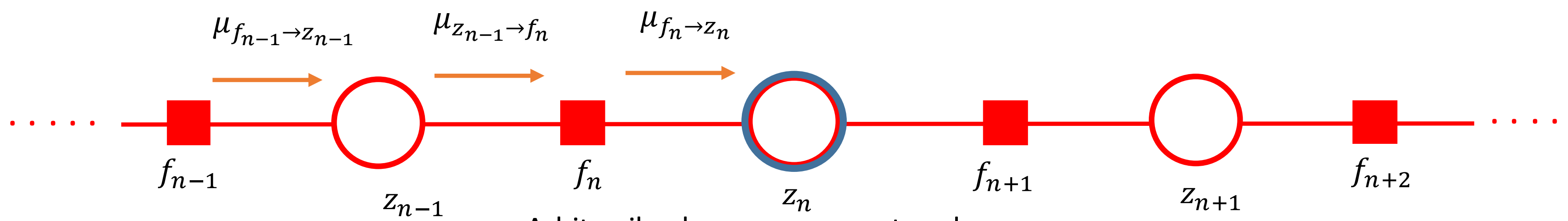


Image source: "Pattern recognition and machine learning", Christopher Bishop

HMM: Sum-of-Product Algorithm

$$
\begin{aligned}
h(\mathbf{z}_1) &= p(\mathbf{z}_1)p(\mathbf{x}_1|\mathbf{z}_1) \\
f_n(\mathbf{z}_{n-1}, \mathbf{z}_n) &= p(\mathbf{z}_n|\mathbf{z}_{n-1})p(\mathbf{x}_n|\mathbf{z}_n)
\end{aligned}
$$



Arbitrarily choose z_n as root node

- Messages from the left towards the root node:

Node-to-factor: $\mu_{\mathbf{z}_{n-1}\to f_n}(\mathbf{z}_{n-1}) = \mu_{f_{n-1}\to \mathbf{z}_{n-1}}(\mathbf{z}_{n-1})$

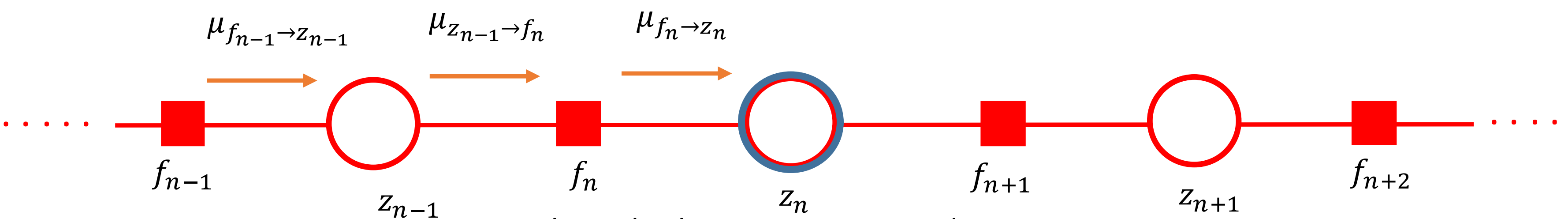
NO computation since there is only two neighbor nodes!

Factor-to-node: $\mu_{f_n\to \mathbf{z}_n}(\mathbf{z}_n) = \sum_{\mathbf{z}_{n-1}} f_n(\mathbf{z}_{n-1}, \mathbf{z}_n)\mu_{\mathbf{z}_{n-1}\to f_n}(\mathbf{z}_{n-1})$

$$\mu_{f_n\to \mathbf{z}_n}(\mathbf{z}_n) = \sum_{\mathbf{z}_{n-1}} f_n(\mathbf{z}_{n-1}, \mathbf{z}_n)\mu_{f_{n-1}\to \mathbf{z}_{n-1}}(\mathbf{z}_{n-1})$$

HMM: Sum-of-Product Algorithm

$$\begin{aligned} h(\mathbf{z}_1) &= p(\mathbf{z}_1)p(\mathbf{x}_1|\mathbf{z}_1) \\ f_n(\mathbf{z}_{n-1}, \mathbf{z}_n) &= p(\mathbf{z}_n|\mathbf{z}_{n-1})p(\mathbf{x}_n|\mathbf{z}_n) \end{aligned}$$



Arbitrarily choose z_n as root node

- Messages from the left towards the root node:

$$\underbrace{\mu_{f_n\rightarrow \mathbf{z}_n}(\mathbf{z}_n)}_{\alpha(z_n)} = \sum_{\mathbf{z}_{n-1}} \overbrace{f_n(\mathbf{z}_{n-1}, \mathbf{z}_n)}^{p(z_n|z_{n-1})p(x_n|z_n)} \underbrace{\mu_{f_{n-1}\rightarrow \mathbf{z}_{n-1}}(\mathbf{z}_{n-1})}_{\alpha(z_{n-1})}$$

$$\Downarrow$$

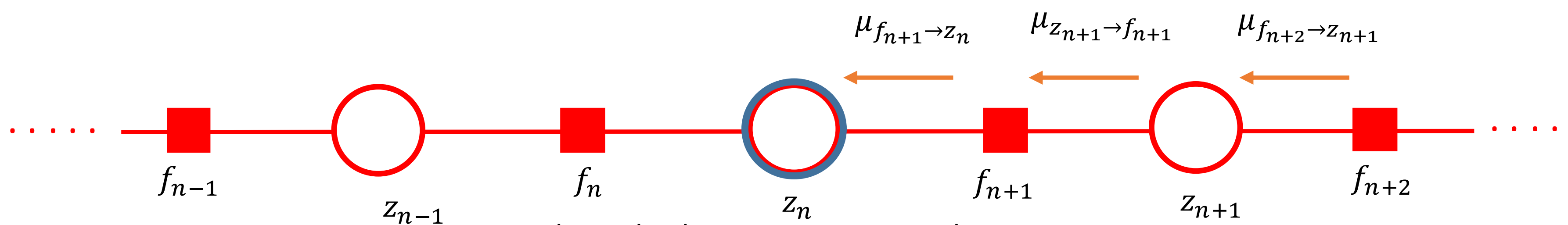
$$\alpha(\mathbf{z}_n) = p(\mathbf{x}_n|\mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} \alpha(\mathbf{z}_{n-1}) p(\mathbf{z}_n|\mathbf{z}_{n-1})$$

Same as forward recursion!

Image modified from: “Pattern recognition and machine learning”, Christopher Bishop

HMM: Sum-of-Product Algorithm

$$
\begin{aligned}
h(\mathbf{z}_1) &= p(\mathbf{z}_1)p(\mathbf{x}_1|\mathbf{z}_1) \\
f_n(\mathbf{z}_{n-1}, \mathbf{z}_n) &= p(\mathbf{z}_n|\mathbf{z}_{n-1})p(\mathbf{x}_n|\mathbf{z}_n)
\end{aligned}
$$



Arbitrarily choose z_n as root node

- Messages from the right towards the root node:

Node-to-factor: $\mu_{z_{n+1}\to f_{n+1}}(z_{n+1}) = \mu_{f_{n+2}\to z_{n+1}}(z_{n+1})$

NO computation since there is only two neighbor nodes!

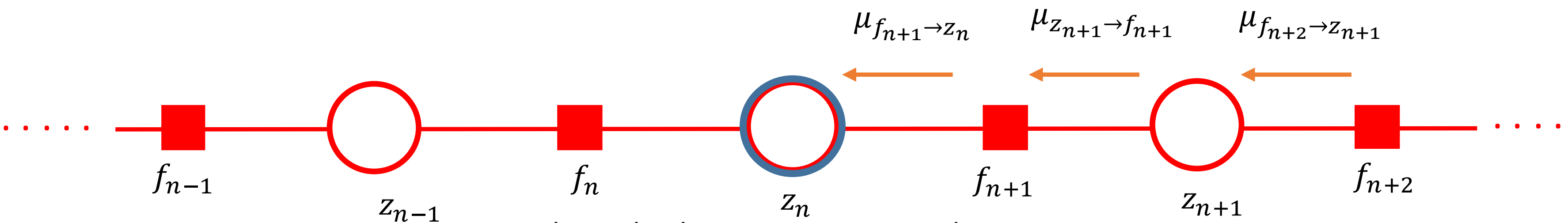
Factor-to-node: $\mu_{f_{n+1}\to z_n}(z_n) = \sum_{z_{n+1}} f_{n+1}(z_n, z_{n+1})\mu_{z_{n+1}\to f_{n+1}}(z_{n+1})$

$$
\mu_{f_{n+1}\to z_n}(z_n) = \sum_{\mathbf{z}_{n+1}} f_{n+1}(\mathbf{z}_n, \mathbf{z}_{n+1})\mu_{f_{n+2}\to z_{n+1}}(z_{n+1})
$$

Image modified from: “Pattern recognition and machine learning”, Christopher Bishop

HMM: Sum-of-Product Algorithm

$$
\begin{aligned}
h(\mathbf{z}_1) &= p(\mathbf{z}_1)p(\mathbf{x}_1|\mathbf{z}_1) \\
f_n(\mathbf{z}_{n-1}, \mathbf{z}_n) &= p(\mathbf{z}_n|\mathbf{z}_{n-1})p(\mathbf{x}_n|\mathbf{z}_n)
\end{aligned}
$$



Arbitrarily choose z_n as root node

- Messages from the right towards the root node:

$$
\underbrace{\mu_{f_{n+1} \to z_n}(z_n)}_{\beta(z_n)} = \sum_{\mathbf{z}_{n+1}} \overbrace{f_{n+1}(\mathbf{z}_n, \mathbf{z}_{n+1})}^{p(z_{n+1}|z_n)p(x_{n+1}|z_{n+1})} \underbrace{\mu_{f_{n+2} \to z_{n+1}}(z_{n+1})}_{\beta(z_{n+1})}
$$

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

Same as backward recursion!

Image modified from: "Pattern recognition and machine learning", Christopher Bishop

Scaling Factors

Forward recursion: $$\alpha(\mathbf{z}_n) = p(\mathbf{x}_n|\mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} \alpha(\mathbf{z}_{n-1}) p(\mathbf{z}_n|\mathbf{z}_{n-1})$$

- Each step the new value $\alpha(z_n)$ is obtained from the previous value $\alpha(z_{n-1})$ by multiplying by quantities $p(z_n|z_{n-1})$ and $p(x_n|z_n)$.
- These probabilities are often significantly less than unity as we work our way forward along the chain, the values of $\alpha(z_n)$ can go to zero exponentially quickly.

Scaling Factors

- Taking logarithm does not help because we are forming sums of products of small numbers.

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

$$\log \alpha(z_n) = \log p(x_n|z_n) + \log \underbrace{\sum_{z_{n-1}} \alpha(z_{n-1}) p(z_n|z_{n-1})}_{\text{Small number}}$$

- We therefore work with re-scaled versions of $\alpha(z_n)$ whose values remain of order unity.

Scaling Factors

- We define a normalized version of α given by:

$$\widehat{\alpha}(\mathbf{z}_n) = p(\mathbf{z}_n|\mathbf{x}_1, \ldots, \mathbf{x}_n) = \frac{\alpha(\mathbf{z}_n)}{p(\mathbf{x}_1, \ldots, \mathbf{x}_n)}$$

- Which is well behaved numerically because it is a probability distribution over K variables for any value of n.

Scaling Factors

- Rewriting the forward-recursion into the normalized form:

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

$$\underbrace{\frac{p(x_1, \dots x_n)}{p(x_1, \dots, x_{n-1})}}_{c_n = p(x_n|x_1, \dots, x_{n-1})} \underbrace{\frac{\alpha(z_n)}{p(x_1, \dots x_n)}}_{\hat{\alpha}(z_n)} = p(x_n|z_n) \sum_{z_{n-1}} \underbrace{\frac{\alpha(z_{n-1})}{p(x_1, \dots, x_{n-1})}}_{\hat{\alpha}(z_{n-1})} p(z_n|z_{n-1})$$

$$\boxed{c_n \widehat{\alpha}(\mathbf{z}_n) = p(\mathbf{x}_n|\mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} \widehat{\alpha}(\mathbf{z}_{n-1}) p(\mathbf{z}_n|\mathbf{z}_{n-1})}$$

Scaling Factors

$$c_n\widehat{\alpha}(\mathbf{z}_n) = p(\mathbf{x}_n|\mathbf{z}_n)\sum_{\mathbf{z}_{n-1}}\widehat{\alpha}(\mathbf{z}_{n-1})p(\mathbf{z}_n|\mathbf{z}_{n-1}) = \tilde{\alpha}(\mathbf{z}_n)$$

- c_n can be recursively computed as: $c_n = \sum_{z_n}\tilde{\alpha}(\mathbf{z}_n)$

Sum over all k entries in Z_n

Proof:

$$\begin{aligned}
c_n &= \textstyle\sum_{z_n}\tilde{\alpha}(\mathbf{z}_n) \\
&= \textstyle\sum_{z_n} p(x_n|z_n)\sum_{z_{n-1}}\hat{\alpha}(z_{n-1})p(z_n|z_{n-1}) \\
&= \textstyle\sum_{z_n} p(x_n|z_n)\sum_{z_{n-1}}\frac{\alpha(z_{n-1})}{p(x_1,\dots,x_{n-1})}p(z_n|z_{n-1}) \\
&= \textstyle\sum_{z_n}\frac{p(x_1,\dots,x_n,z_n)}{p(x_1,\dots,x_{n-1})} = \frac{p(x_1,\dots,x_n)}{p(x_1,\dots,x_{n-1})}
\end{aligned}$$

Scaling Factor

- We can do similar normalization for β:

$\beta(z_n)$

$$\widehat{\beta}(\mathbf{z}_n) = \frac{p(\mathbf{x}_{n+1}, \ldots, \mathbf{x}_N | \mathbf{z}_n)}{p(\mathbf{x}_{n+1}, \ldots, \mathbf{x}_N | \mathbf{x}_1, \ldots, \mathbf{x}_n)}$$

- And re-writing the backward-recursion into the normalized form:

$$\boxed{c_{n+1}\widehat{\beta}(\mathbf{z}_n) = \sum_{\mathbf{z}_{n+1}} \widehat{\beta}(\mathbf{z}_{n+1}) p(\mathbf{x}_{n+1}|\mathbf{z}_{n+1}) p(\mathbf{z}_{n+1}|\mathbf{z}_n) = \tilde{\beta}(\mathbf{z}_n)}$$

- where

$$c_{n+1} = \sum_{z_n} \tilde{\beta}(\mathbf{z}_n)$$

Sum over all k entries in Z_n

Scaling Factor

- As a result, we get:

$$\boxed{\begin{aligned} \gamma(\mathbf{z}_n) &= \widehat{\alpha}(\mathbf{z}_n)\widehat{\beta}(\mathbf{z}_n) \\ \xi(\mathbf{z}_{n-1}, \mathbf{z}_n) &= c_n^{-1}\,\widehat{\alpha}(\mathbf{z}_{n-1})p(\mathbf{x}_n|\mathbf{z}_n)p(z_n|z_{n-1})\widehat{\beta}(\mathbf{z}_n) \end{aligned}}$$

Proof:

$$\gamma(z_n) = \hat{\alpha}(z_n)\hat{\beta}(z_n) = \frac{\alpha(z_n)}{p(x_1, \dots, x_n)} \frac{\beta(z_n)}{p(x_{n+1}, \dots, x_N | x_1, \dots, x_n)}$$

$$= \frac{\alpha(z_n)}{\cancel{p(x_1, \dots, x_n)}} \frac{\beta(z_n)}{\frac{p(x_1, \dots, x_N)}{\cancel{p(x_1, \dots, x_n)}}}$$

$$= \frac{\alpha(z_n)\beta(z_n)}{p(X)}$$

Scaling Factor

- As a result, we get:

$$\boxed{\begin{aligned}\gamma(\mathbf{z}_n) &= \widehat{\alpha}(\mathbf{z}_n)\widehat{\beta}(\mathbf{z}_n)\\ \xi(\mathbf{z}_{n-1},\mathbf{z}_n) &= c_n^{-1}\widehat{\alpha}(\mathbf{z}_{n-1})p(\mathbf{x}_n|\mathbf{z}_n)p(z_n|z_{n-1})\widehat{\beta}(\mathbf{z}_n)\end{aligned}}$$

Proof:

$$\xi(z_{n-1},z_n) = \frac{\cancel{p(x_1,\dots,x_{n-1})}}{p(x_1,\dots,x_n)}\frac{\alpha(z_{n-1})}{\cancel{p(x_1,\dots,x_{n-1})}}p(x_n|z_n)p(z_n|z_{n-1})\frac{\beta(z_n)}{p(x_{n+1},\dots,x_N|x_1,\dots,x_n)}$$

$$= \frac{\alpha(z_{n-1})}{\cancel{p(x_1,\dots,x_n)}}p(x_n|z_n)p(z_n|z_{n-1})\frac{\beta(z_n)}{\frac{p(x_1,\dots,x_N)}{\cancel{p(x_1,\dots,x_n)}}}$$

$$= \frac{\alpha(z_{n-1})p(x_n|z_n)p(z_n|z_{n-1})\beta(z_n)}{p(X)}$$

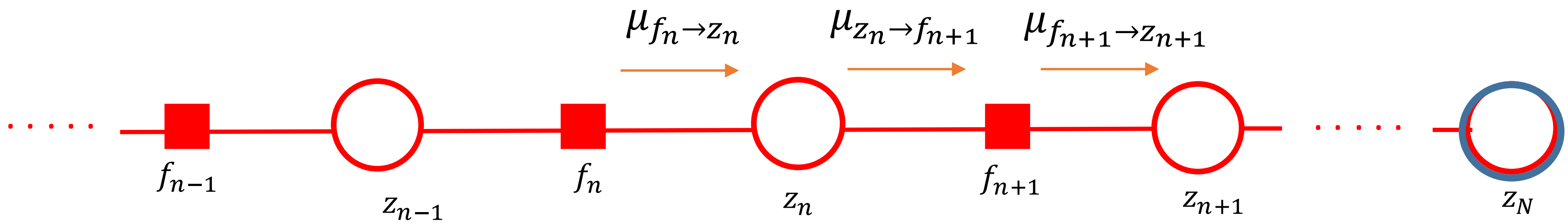
The Viterbi Algorithm

- We are now interested in finding the max probability and most probable sequence of hidden states for a given observation sequence.

- **Recall**: finding the most probable sequence of latent states $\neq$ finding the set of states that are individually the most probable.

- Finding the most probable *sequence* of states can be solved efficiently using the max-sum algorithm, i.e. *Viterbi* algorithm for HMM.

The Viterbi Algorithm



Choose Z_N as root node

- Messages in the max-sum algorithm:

Node-to-Factor:

$$\mu_{\mathbf{z}_n \to f_{n+1}}(\mathbf{z}_n) = \mu_{f_n \to \mathbf{z}_n}(\mathbf{z}_n)$$

Factor-to-Node:

$$\mu_{f_{n+1} \to \mathbf{z}_{n+1}}(\mathbf{z}_{n+1}) = \max_{\mathbf{z}_n} \left\{ \ln \underbrace{f_{n+1}(\mathbf{z}_n, \mathbf{z}_{n+1})}_{p(z_{n+1}|z_n)p(x_{n+1}|z_{n+1})} + \mu_{\mathbf{z}_n \to f_{n+1}}(\mathbf{z}_n) \right\}$$

- Similar to forward recursion, except summation of Z_n is replaced with max of Z_n.

- No backward passing since the max probability is the same regardless of the choice of root node.

The Viterbi Algorithm

- Denote $\omega(z_n) \equiv \mu_{f_n \to z_n}(z_n)$, we can rewrite the message as:

$$\boxed{\omega(z_{n+1}) = \ln p(x_{n+1}|z_{n+1}) \; + \; \max_{z_n}\{\ln p(z_{n+1}|z_n) + \omega(z_n)\},}$$

$K \times 1$ entries

which can be computed recursively!

Proof:

$$\underbrace{\mu_{f_{n+1} \to \mathbf{z}_{n+1}}(\mathbf{z}_{n+1})}_{\omega(z_{n+1})} \quad = \quad \max_{\mathbf{z}_n} \left\{ \underbrace{\ln f_{n+1}(\mathbf{z}_n, \mathbf{z}_{n+1})}_{p(z_{n+1}|z_n)p(x_{n+1}|z_{n+1})} + \underbrace{\mu_{\mathbf{z}_n \to f_{n+1}}(\mathbf{z}_n)}_{\mu_{f_n \to z_n}(z_n)} \right\}$$

$$\Longrightarrow \quad \omega(z_{n+1}) = \max_{z_n}\{\ln p(z_{n+1}|z_n) + \ln p(x_{n+1}|z_{n+1}) + \underbrace{\mu_{z_n \to f_{n+1}}(z_n)}_{\omega(z_n)}\}$$

$$\Longrightarrow \quad \omega(z_{n+1}) = \ln p(x_{n+1}|z_{n+1}) \; + \; \max_{z_n}\{\ln p(z_{n+1}|z_n) + \omega(z_n)\}$$

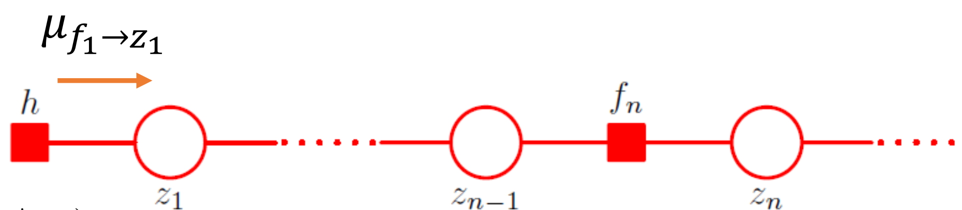
The Viterbi Algorithm

$K \times 1$
entries

$$\boxed{\omega(z_{n+1}) = \ln p(x_{n+1}|z_{n+1}) + \max_{z_n}\{\ln p(z_{n+1}|z_n) + \omega(z_n)\},}$$

- Note that no need for scaling since the Viterbi algorithm works with log probabilities.

- **Initialization**: factor-to-node message



$$h(\mathbf{z}_1) = p(\mathbf{z}_1)p(\mathbf{x}_1|\mathbf{z}_1)$$

$$\boxed{\omega(\mathbf{z}_1) = \ln p(\mathbf{z}_1) + \ln p(\mathbf{x}_1|\mathbf{z}_1).}$$

$K \times 1$
entries

Image source: "Pattern recognition and machine learning", Christopher Bishop

The Viterbi Algorithm

- **Root Node**: The maximal probability of the joint distribution $p(x_1, \dots x_N, z_1, \dots, z_N)$ is given by the max of $\omega(z_N)$ at the root node.

$$\max_{z_1, \dots, z_N} p(x_1, \dots x_N, z_1, \dots, z_N) = \max_{z_N} \omega(z_N)$$

The Viterbi Algorithm

- The forward recursion to Z_n will give us the maximal probability of the joint distribution $p(X, Z)$.
- We also wish to find the sequence of latent variable values that corresponds to the maximal probability.
- We will use the back-tracking procedure described in Lecture 6 to do this.

The Viterbi Algorithm

- Keep a record of the values of Z_n that correspond to the maxima for each value of the K values of Z_{n+1}, denoted by $\psi(k_n)$ where $k = 1, \ldots, K$.

$K \times N$
table

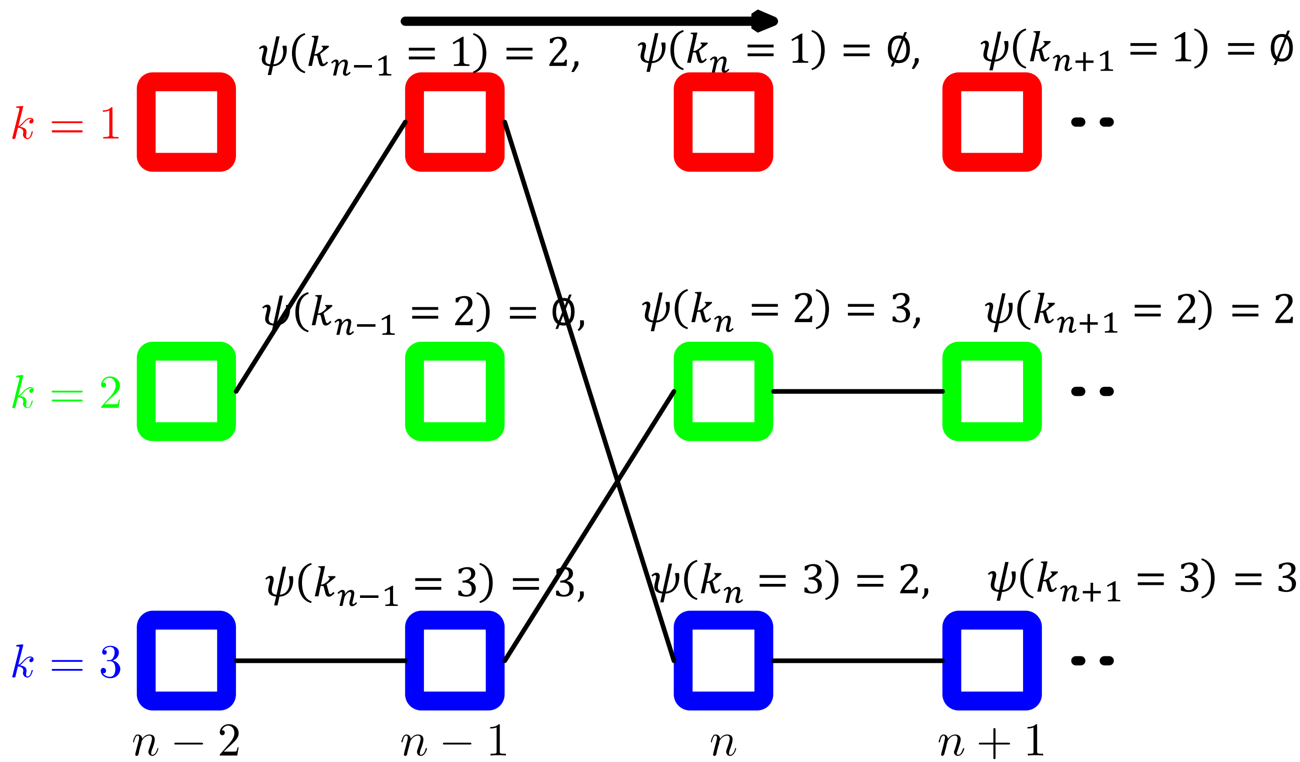


Image source: “Pattern recognition and machine learning”, Christopher Bishop

The Viterbi Algorithm

- We get $\psi(k_N)$ when we reach the end of the chain, i.e. root node Z_N.

- The sequence of latent variable values that corresponds to the maximal probability can then be obtained by backtracking the chain recursively:

$$k_n^{\max} = \psi(k_{n+1}^{\max}).$$

Summary

- We have looked at how to:

1. Describe the joint distribution of a HMM with the transition and emission probabilities.
2. Use the EM algorithm for maximum likelihood estimation of the latent variables and unknown parameters in the HMM.
3. Use the forward-backward algorithm to evaluate the EM algorithm.
4. Apply the Viterbi algorithm to find the maximal probability and its configuration.