NUS National University of Singapore | School of Computing

CS5340
Uncertainty Modeling in AI

Lecture 8:
Hidden Markov Models (HMM)
(or "Learning with Sequential Data")

Asst. Prof. Harold Soh

AY 2022/23

Semester 2

Course Schedule

Week	Date	Lecture Topic	Tutorial Topic
1	12 Jan	Introduction to Uncertainty Modeling + Probability Basics	Introduction
2	19 Jan	Simple Probabilistic Models	Probability Basics
3	26 Jan	Bayesian networks (Directed graphical models)	More Basic Probability
4	2 Feb	Markov random Fields (Undirected graphical models)	DGM modelling and d-separation
5	9 Feb	Variable elimination and belief propagation	MRF + Sum/Max Product
6	16 Feb	Factor graph and the junction tree algorithm	**Quiz 1**
-	-	RECESS WEEK	
7	2 Mar	Mixture Models and Expectation Maximization (EM)	Linear Gaussian Models
8	9 Mar	Hidden Markov Models (HMM)	Probabilistic PCA
9	**16** Mar	Monte-Carlo Inference (Sampling)	Linear Gaussian Dynamical System
10	23 Mar	Variational Inference	MCMC + Sequential VAE
11	30 Mar	Inference and Decision-Making (Special Topic)	**Quiz 2**
12	6 Apr	Gaussian Processes (Special Topic)	Wellness Day
13	13 Apr	**Project Presentations**	Closing

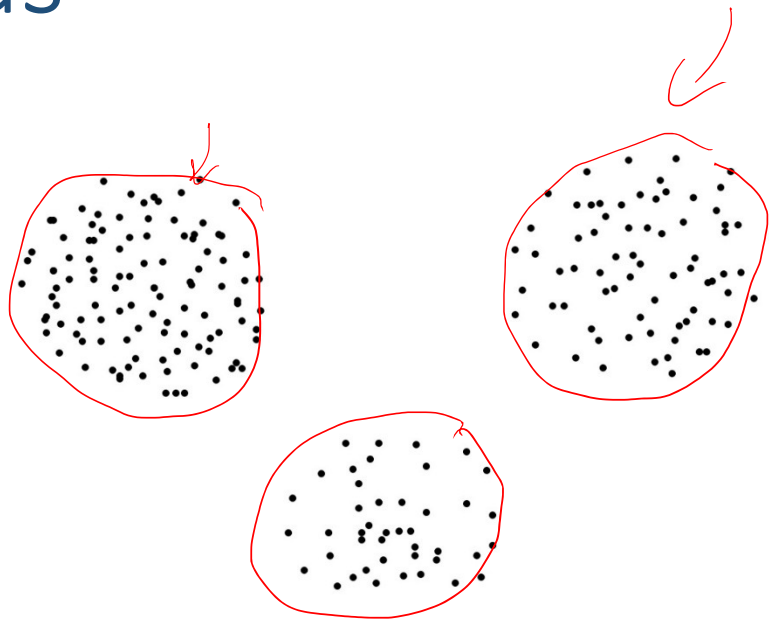
NUS National University of Singapore | School of Computing

Recap from Lecture 7

GMM, Expectation Maximization

Key Ideas

- The Clustering Problem
  - Has many applications
- Problem-Solving Approach
- Non-probabilistic Algorithm
  - K-means (iterative method)
  - Limitations of k-means



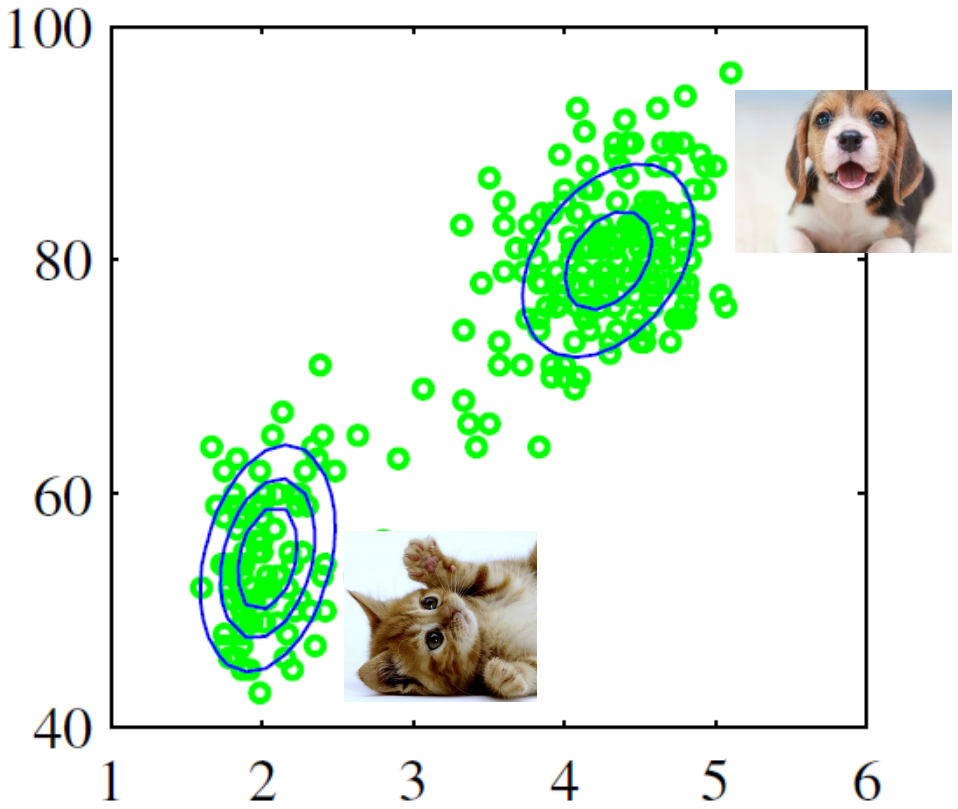
Non-Probabilistic Approach: K-Means

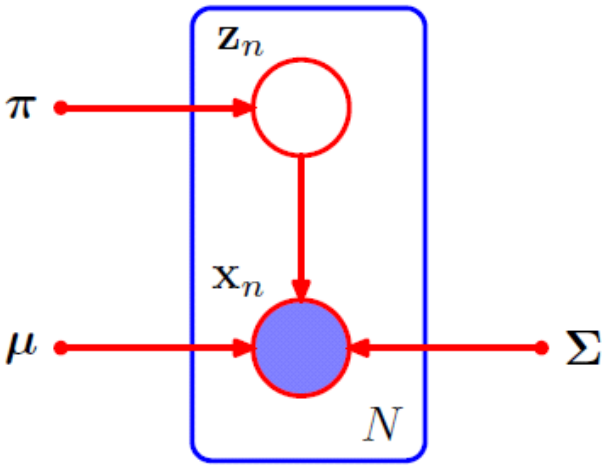
K-Means (a.k.a. Lloyd) Algorithm:

1. Initialization: Randomly choose some initial values for $\{\mu_k\}$.
2. Assignment step: Minimize J w.r.t. $\{r_{nk}\}$, while keeping $\{\mu_k\}$ fixed.
3. Update step: Minimize J w.r.t. $\{\mu_k\}$, while keeping $\{r_{nk}\}$ fixed.

Iterate until convergence

Intuition: Generative Story





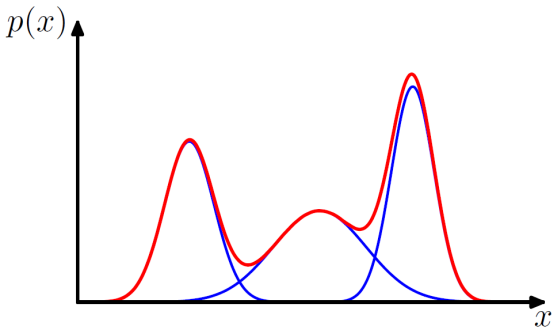
Gaussian Mixture Models

- Mixture distributions: linear combinations of more basic distributions e.g.,
  - Gaussian ⇒ Gaussian Mixture Model (GMM).
  - Poisson ⇒ Poisson Mixture Model
  - Bernoulli ⇒ Bernoulli Mixture Model

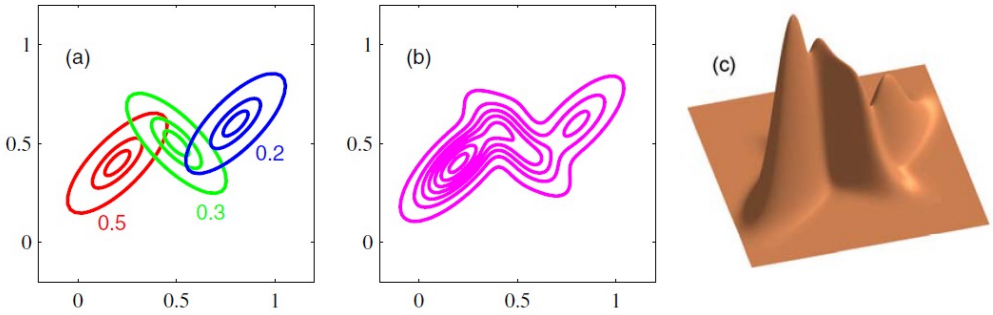
$p(x|z)$

- GMMs can approximate almost any continuous density with arbitrary accuracy.

Examples:



1D GMM (red) formed by 3 Gaussians (blue)



(b)-(c) 2D GMM formed by (a) 3 Gaussians

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

The General EM Algorithm

1. Choose an initial setting for the parameters θ^{old}.
2. Expectation step: Evaluate $p(\mathrm{Z}|\mathrm{X}, \theta^{old})$.
3. Maximization step: Evaluate θ^{new} given by:

$$\boldsymbol{\theta}^{\mathrm{new}} = \arg\max_{\boldsymbol{\theta}} \mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\mathrm{old}})$$

where

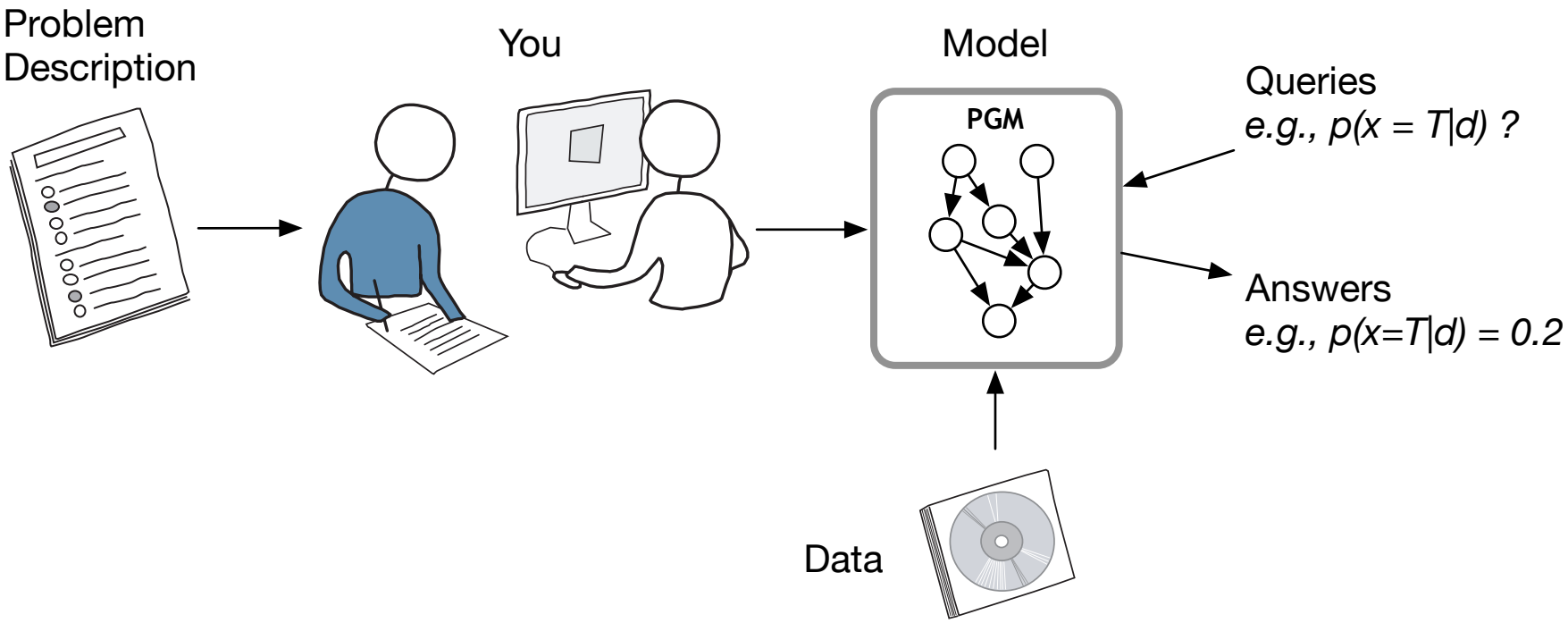
$$\mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\mathrm{old}}) = \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\mathrm{old}}) \ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$$

4. Check for convergence of either the log likelihood or the parameter values, if not converged:

$$\boldsymbol{\theta}^{\mathrm{old}} \leftarrow \boldsymbol{\theta}^{\mathrm{new}}$$

CS5340 in a nutshell

CS5340 is about how to "**represent**" and "**reason**" with uncertainty in a computer.



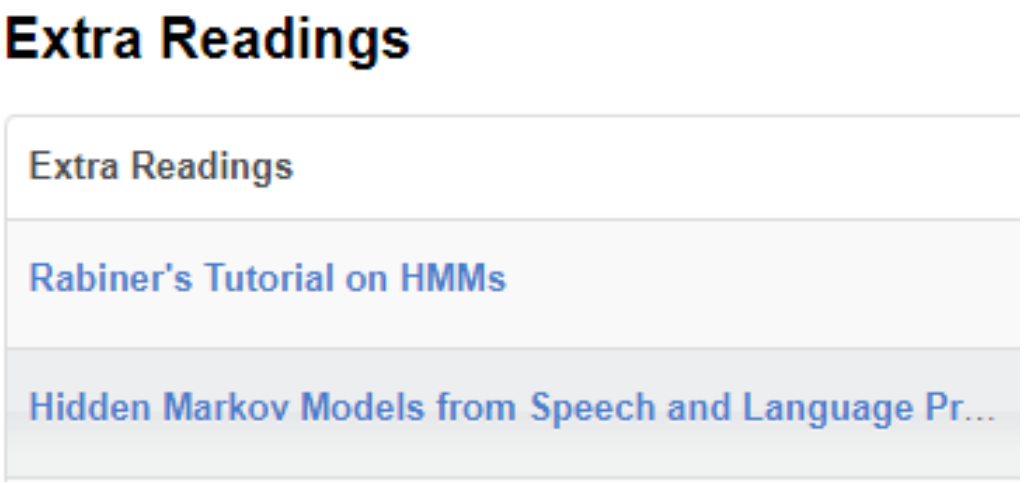
Acknowledgements

- 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.
5. Lee Gim Hee's slides.

Extra Readings

- Resources Section on Piazza and on Canvas



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 as part of the EM algorithm.
4. Apply the Viterbi algorithm to find the maximal probability and its configuration.

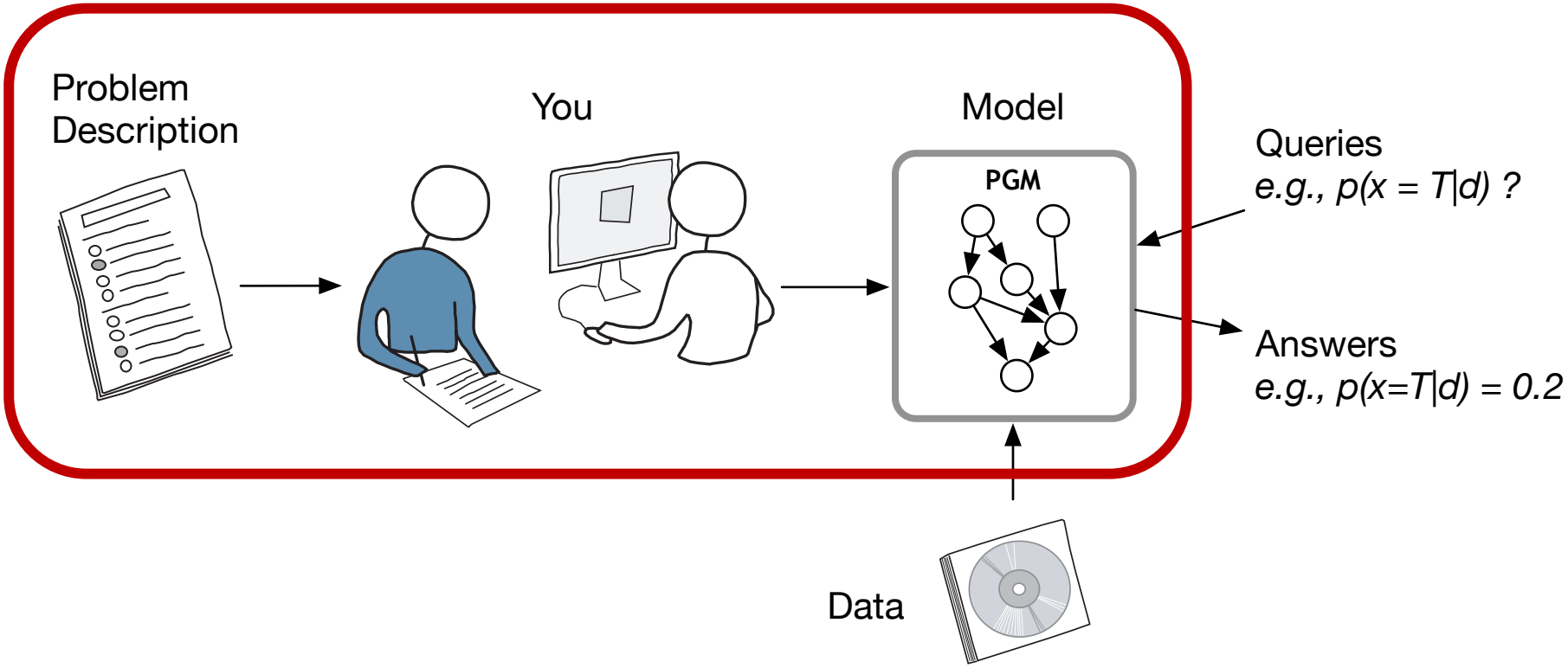
NUS National University of Singapore | School of Computing

Hidden Markov Model: Introduction

Motivation, Sequential Models, Model Structure

CS5340 in a nutshell

CS5340 is about how to "**represent**" and "**reason**" with uncertainty in a computer.

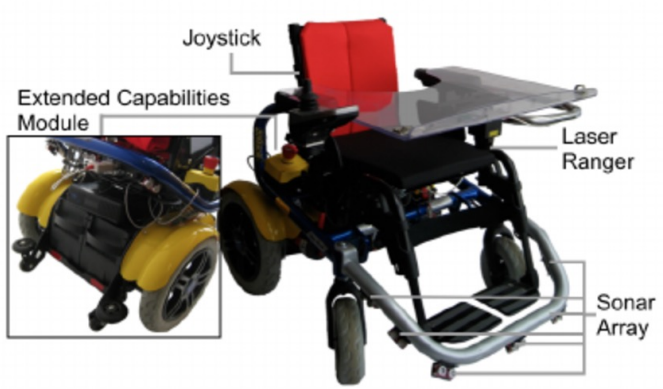


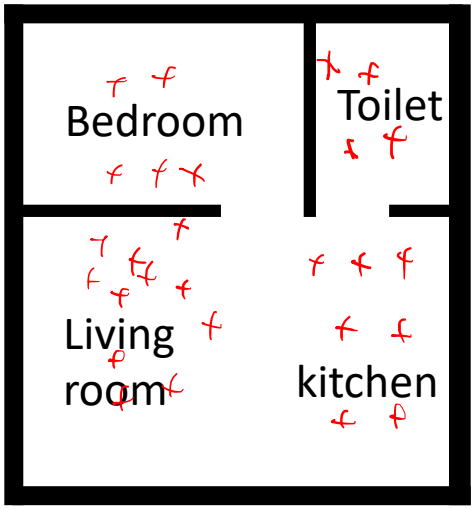
Key Ideas

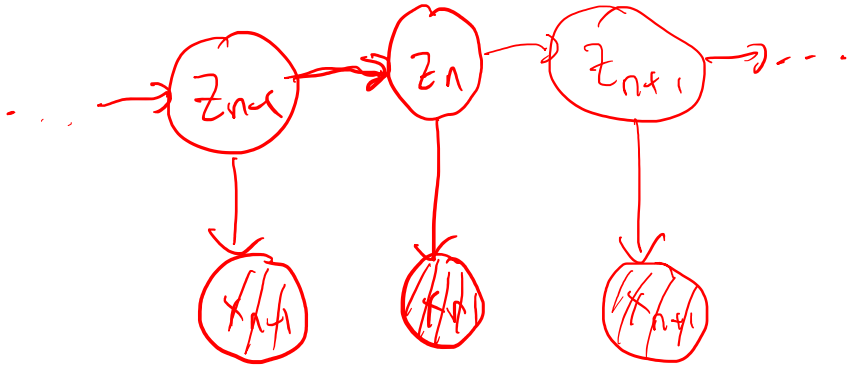
- Sequential Data
  - Noisy samples
  - Many real-world problems/applications
- Hidden Markov Model (HMM)
  - Markov Assumption

Problem & Intuition

- Modeling Human Trajectories
  - E.g., for intent Prediction
- Noisy sensors

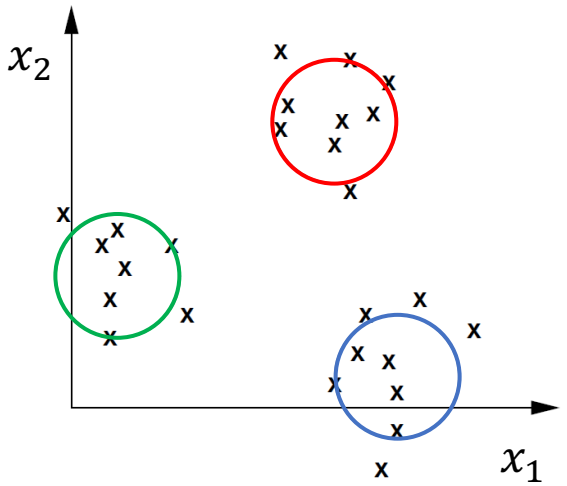






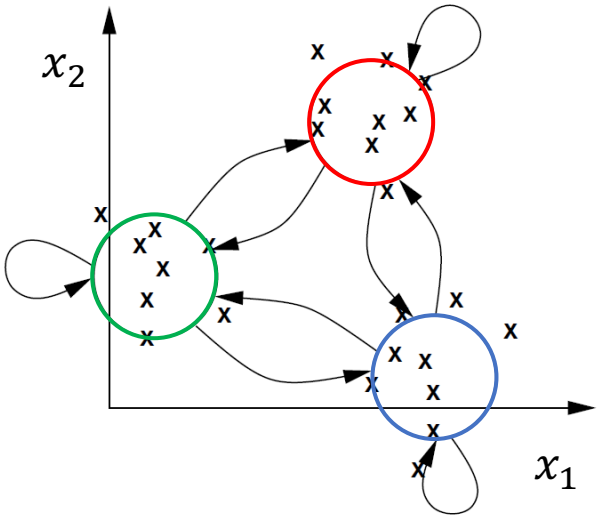
Sequential Data

Independent choice of mixture component



GMM

Choice of current mixture component is dependent on the previous observation



- In ~~EM~~, the choice of mixture component for each observation is independent.
- HMM is an extension of the mixture model, where the mixture component depends on the component for the previous observation.

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

Sequential Data

Example: Robot SLAM

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

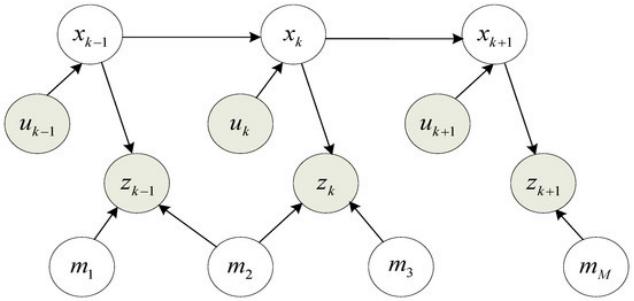
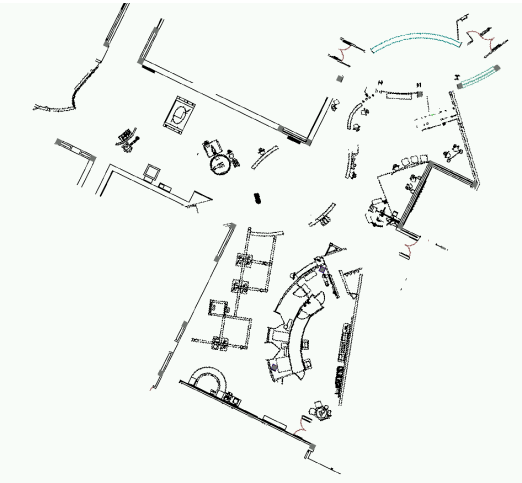
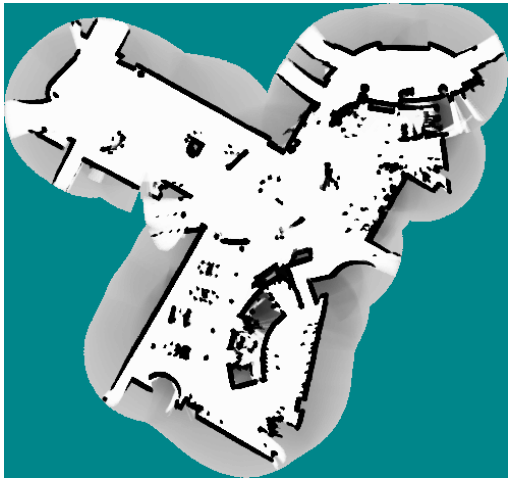


Image source: http://www.mdpi.com/1424-8220/17/5/1174

CAD Map



Estimated Map



(S. Thrun, San Jose Tech Museum)

Sequential Data

Example: Target Tracking

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

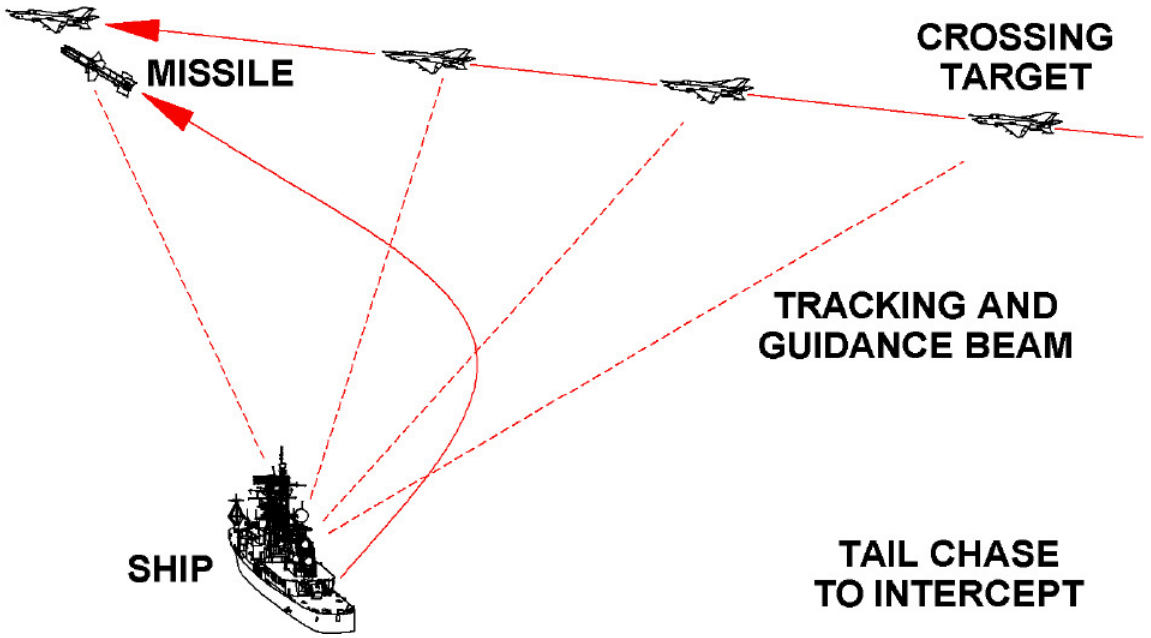


Image source: http://www.okieboat.com/History%20guidance%20and%20homing.html

Sequential Data

Example: Financial Forecasting

- Predict future market behavior from historical data.

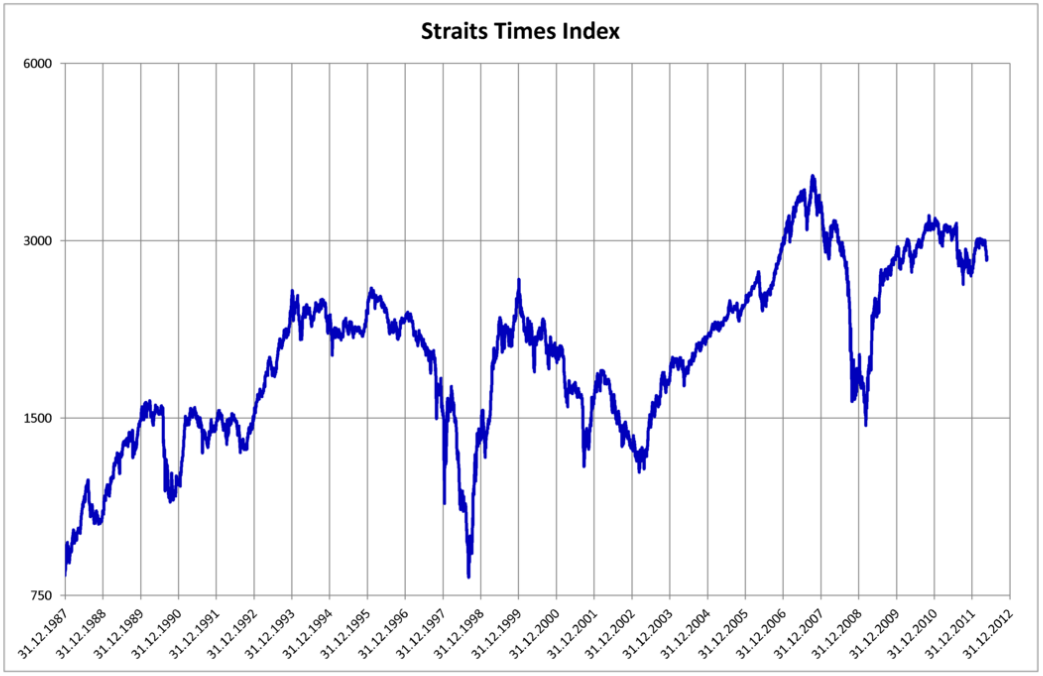


Image source: https://en.wikipedia.org/wiki/Straits_Times_Index

Sequential Data

Example: Speech Recognition

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

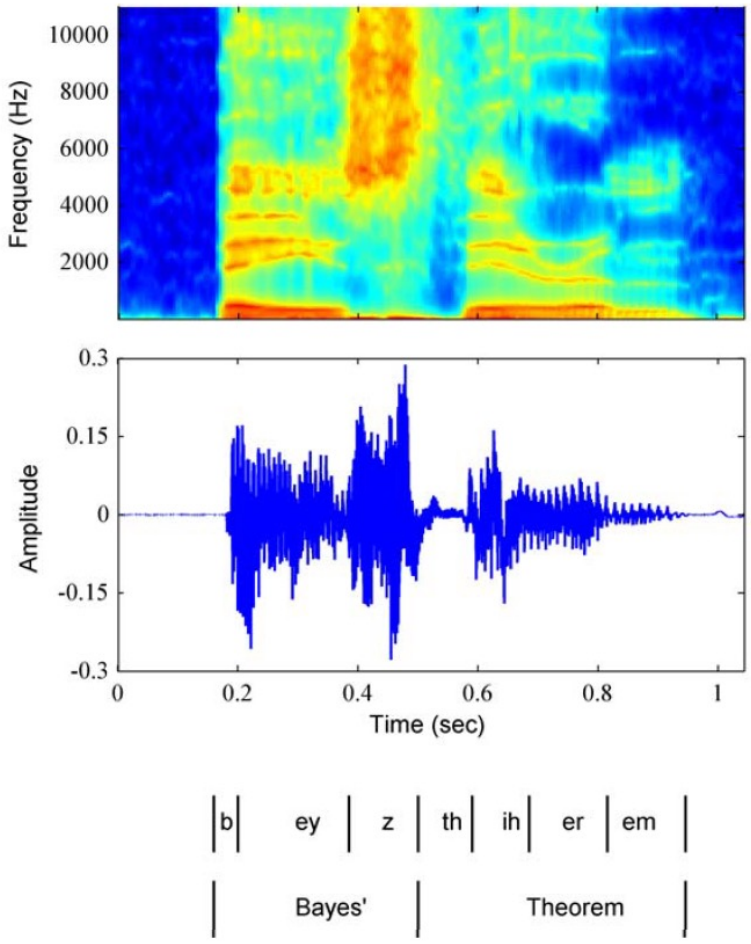


Image source: "Pattern recognition and machine learning", Christopher Bishop

HMM

A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition

LAWRENCE R. RABINER, FELLOW, IEEE

Although initially introduced and studied in the late 1960s and early 1970s, statistical methods of Markov source or hidden Markov modeling have become increasingly popular in the last several years. There are two strong reasons why this has occurred. First the models are very rich in mathematical structure and hence can form the theoretical basis for use in a wide range of applications. Second the models, when applied properly, work very well in practice for several important applications. In this paper we attempt to carefully and methodically review the theoretical aspects of this type of statistical modeling and show how they have been applied to selected problems in machine recognition of speech.

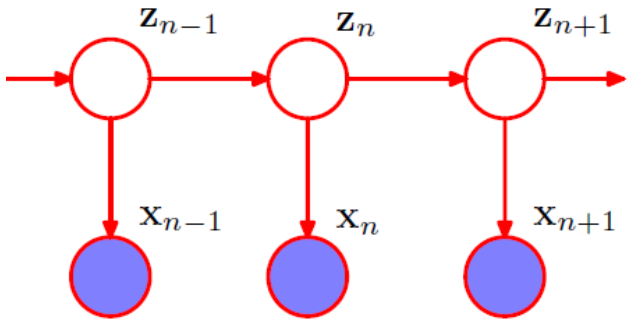
In this case, with a good signal model, we can simulate the source and learn as much as possible via simulations. Finally, the most important reason why signal models are important is that they often work extremely well in practice, and enable us to realize important practical systems—e.g., prediction systems, recognition systems, identification systems, etc., in a very efficient manner.

These are several possible choices for what type of signal model is used for characterizing the properties of a given signal. Broadly one can dichotomize the types of signal models into the class of deterministic models, and the class

[Rabiner, 1989]

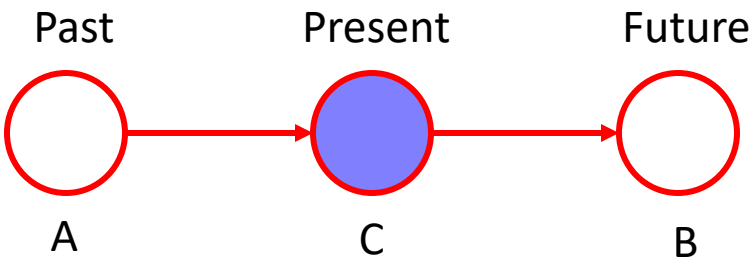
Hidden Markov Model (HMM)

- An HMM is a natural generalization of a mixture model
  - a "dynamic" mixture model.
- For each observed variable X_n, there is a latent variable Z_n
  - may be different type or dimensionality to X_n.
- The latent $\{Z_1, \ldots Z_{n-1}, Z_n \ldots\}$ form a Markov Chain
  - Z_n is dependent only on the previous state Z_{n-1}.



Recall from Lecture 3: Head-Tail

Intuitive interpretation:



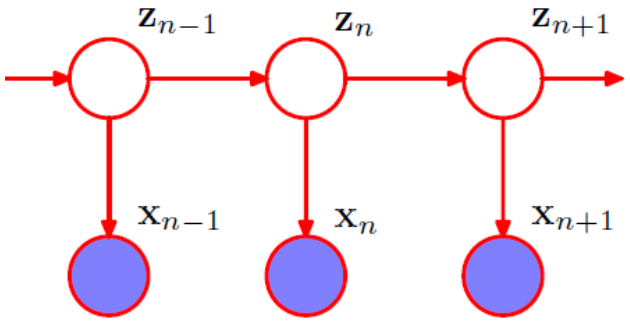
- The conditional independence $A \perp B \mid C$ translates into the statement: "the past is independent of the future given the present".
- This is an example of a simple classical Markov Chain.

HMM: State-Space Model

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

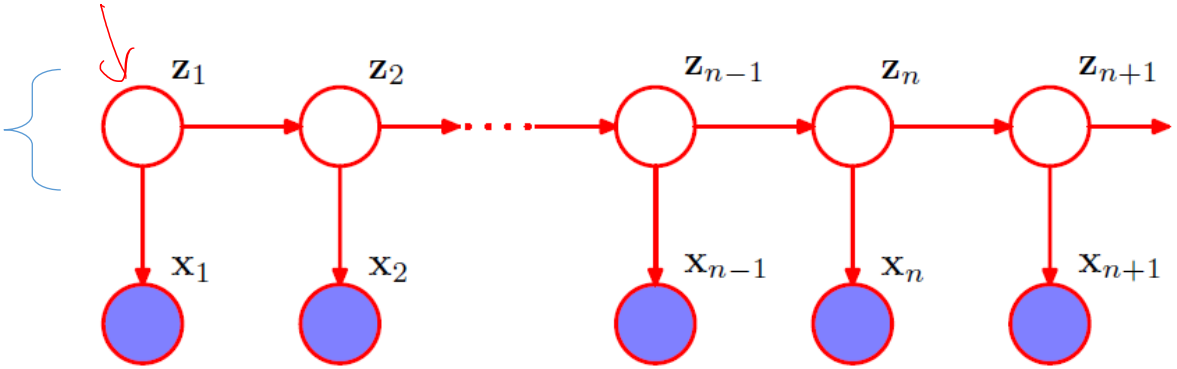
- Satisfies the Markov assumption: conditional independence that Z_{n-1} and Z_{n+1} are independent given Z_n:

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



HMM: Joint Distribution

1st order Markov Chain

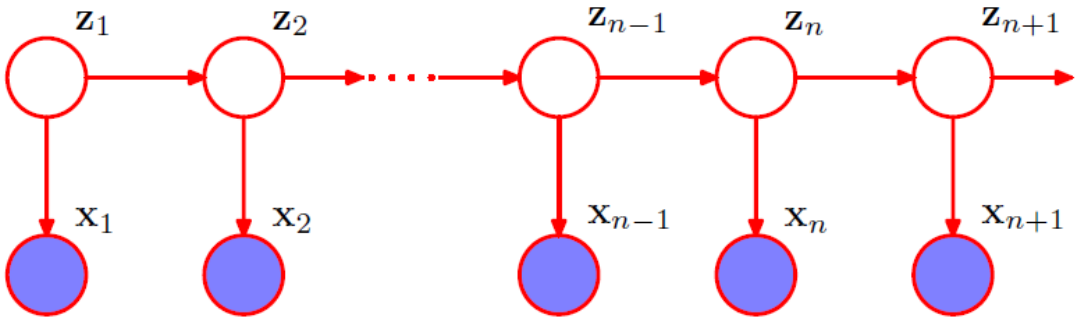


- The joint distribution is given by:

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

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

HMM: Joint Distribution

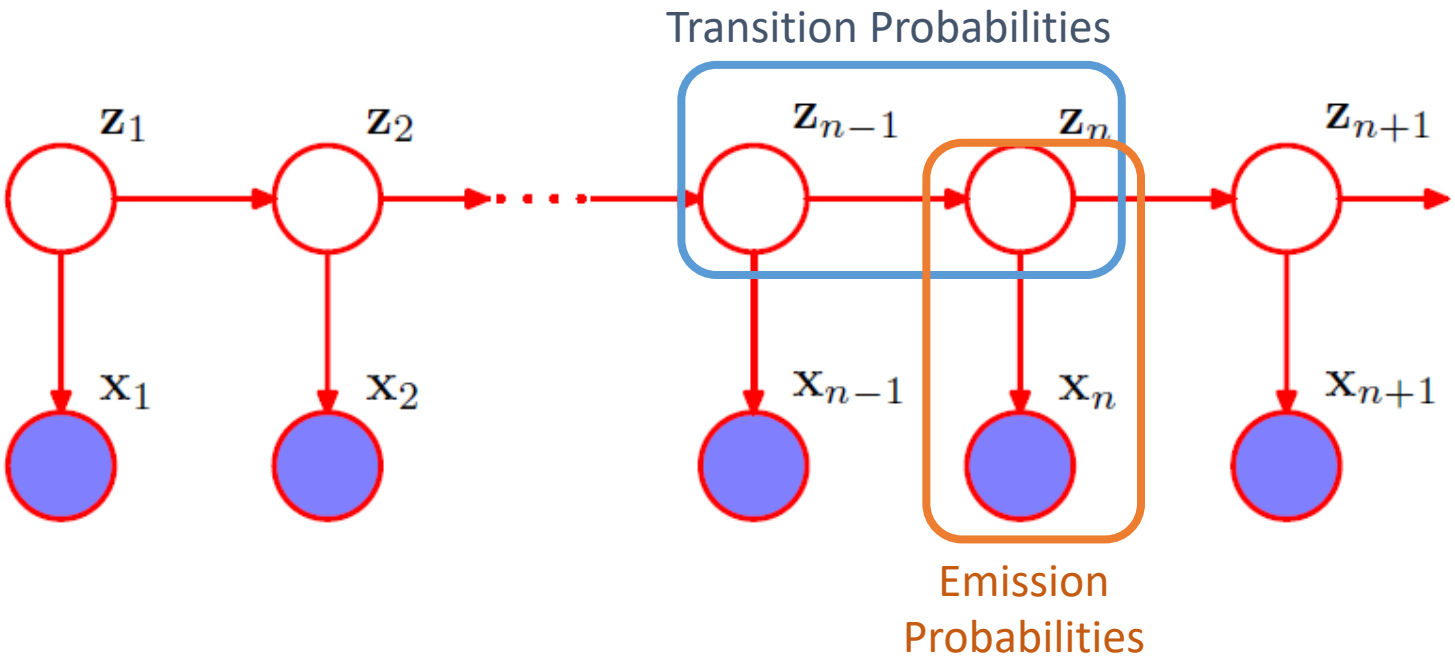


- **HMM** (covered in this lecture): Latent variables are discrete, observed variable can be either discrete or continuous.

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

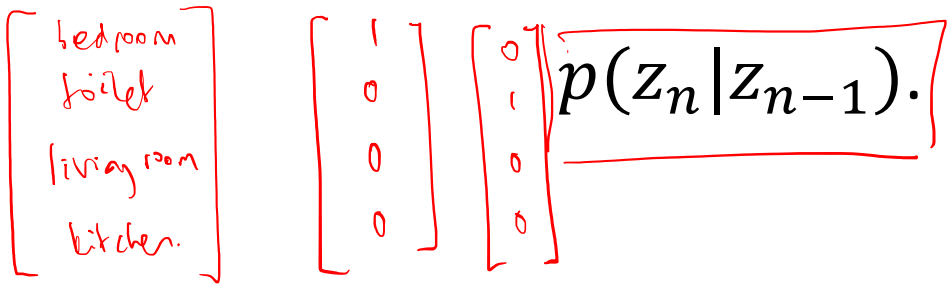
Image source: "Pattern recognition and machine learning", Christopher Bishop

Transition and Emission Probabilities



Transition Probabilities

- 1-of-K coding scheme for the discrete latent variables Z_n.
  - Describes which mixture component is responsible for generating the observation X_n.
- Z_n depends on the previous latent variable Z_{n-1} through a conditional distribution:



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

Transition Probabilities

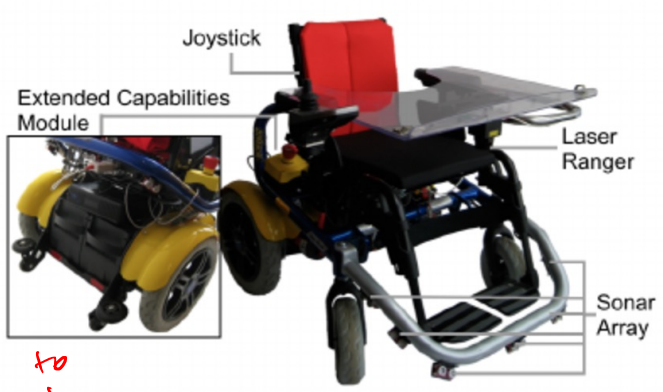
- Since $z_n \in \{0,1\}^K$, $p(z_n|z_{n-1})$ corresponds to a $K \times K$ matrix A, where the elements are known as transition probabilities.

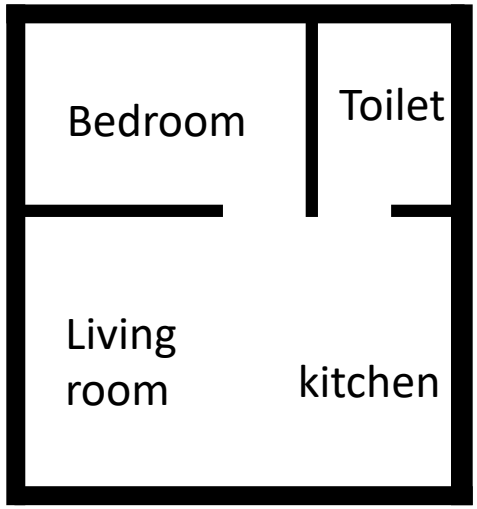
- Properties of the state transition matrix $A \in \mathbb{R}^{K \times K}$:

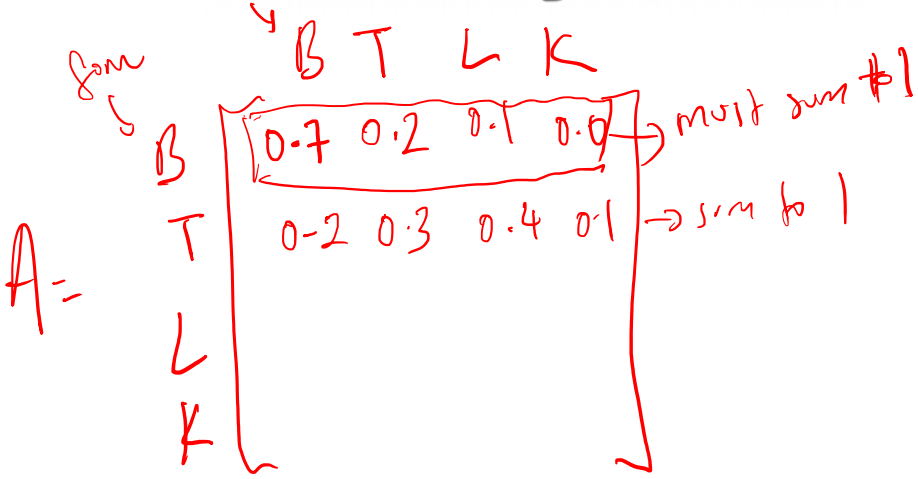
1. $A_{jk} \equiv p(z_{nk} = 1 | z_{n-1,j} = 1)$
2. $0 \leqslant A_{jk} \leqslant 1$, with $\sum_k A_{jk} = 1$
3. $K(K-1)$ independent parameters.

Example: Transition Matrix

- Modeling Human Trajectories
  - E.g., for intent prediction
- Noisy sensors







Transition Diagram

- 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.

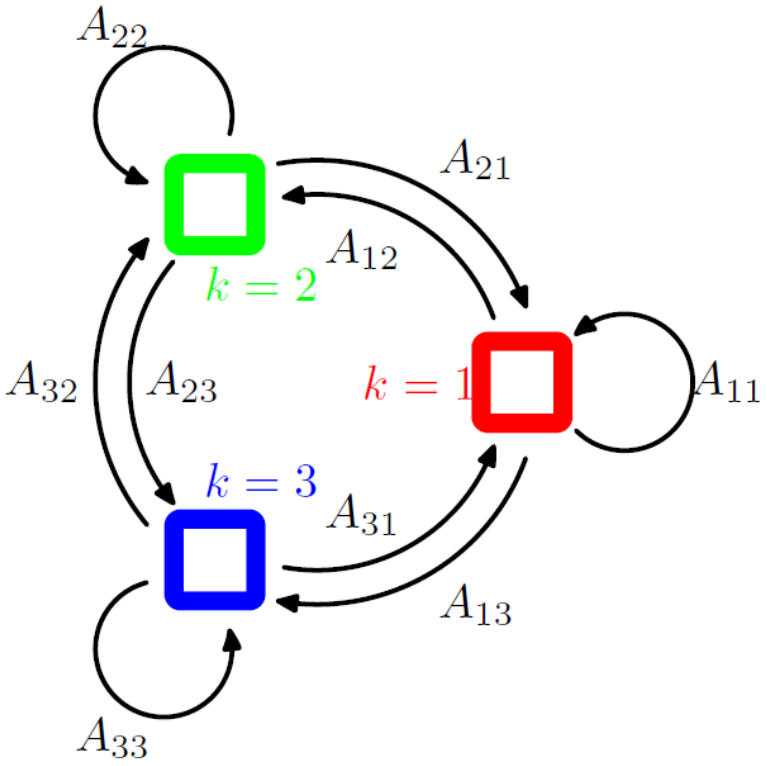
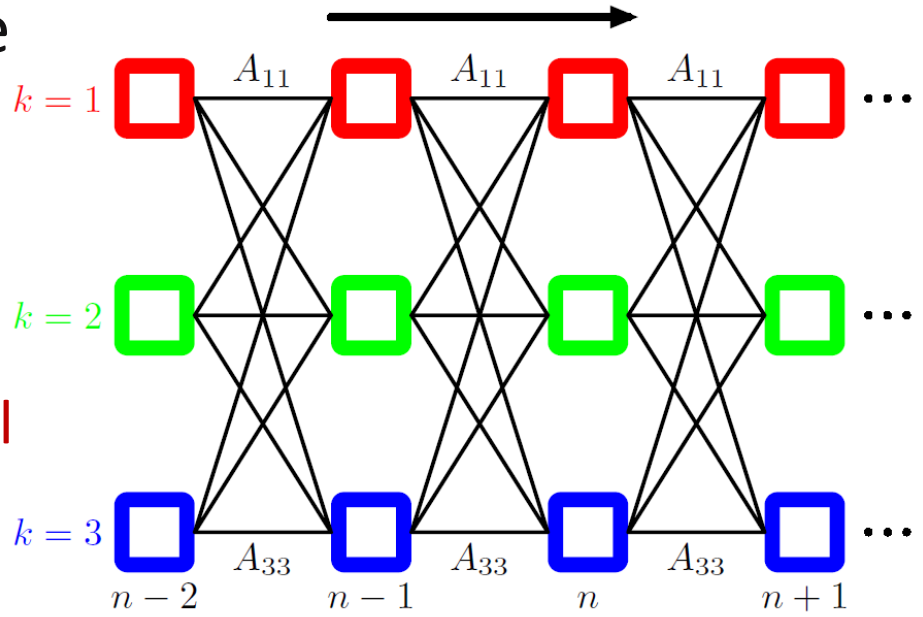


Image source: "Pattern recognition and machine learning", Christopher Bishop

Lattice / Trellis Representation

- 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

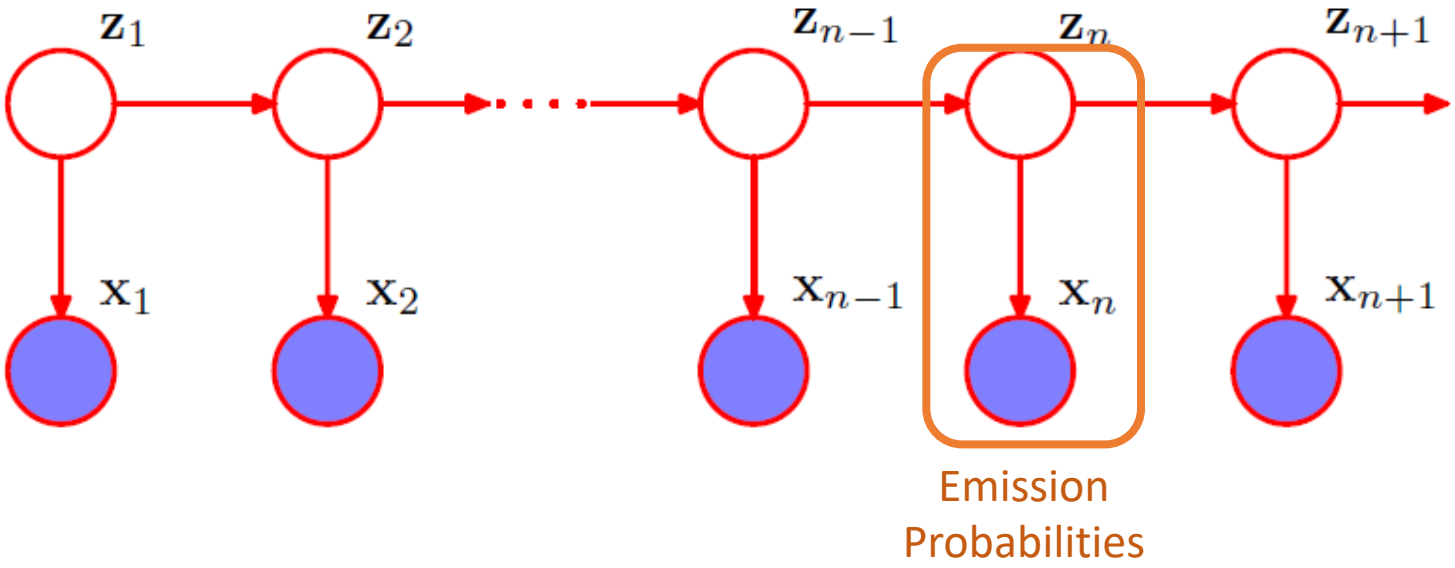
- Can write the conditional distribution :

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

- Initial latent variable Z_1 does not have a parent node. Represented as a categorical distribution:

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

Transition and Emission Probabilities



Emission Probabilities

- Conditional distributions of the observed variable X_n, given the latent variable Z_n,

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

- $\phi = \{\phi_1, \ldots \phi_K\}$ 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

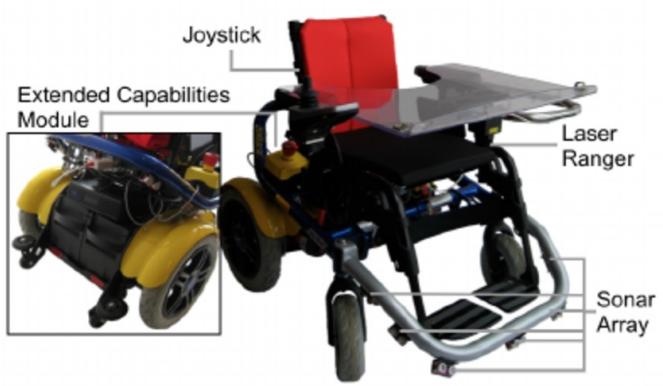
- The emission probabilities is given by:

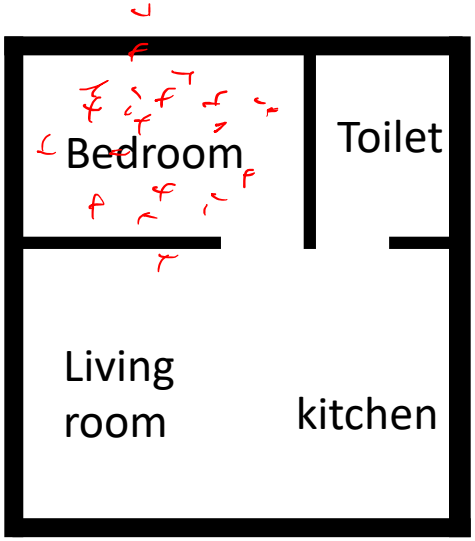
$$p(\mathbf{x}_n|\mathbf{z}_n, \boldsymbol{\phi}) = \prod_{k=1}^{K} p(\mathbf{x}_n|\boldsymbol{\phi}_k)^{z_{nk}}.$$

- 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.

Example: Emissions

- Modeling Human Trajectories
  - E.g., for intent prediction
- Noisy sensors

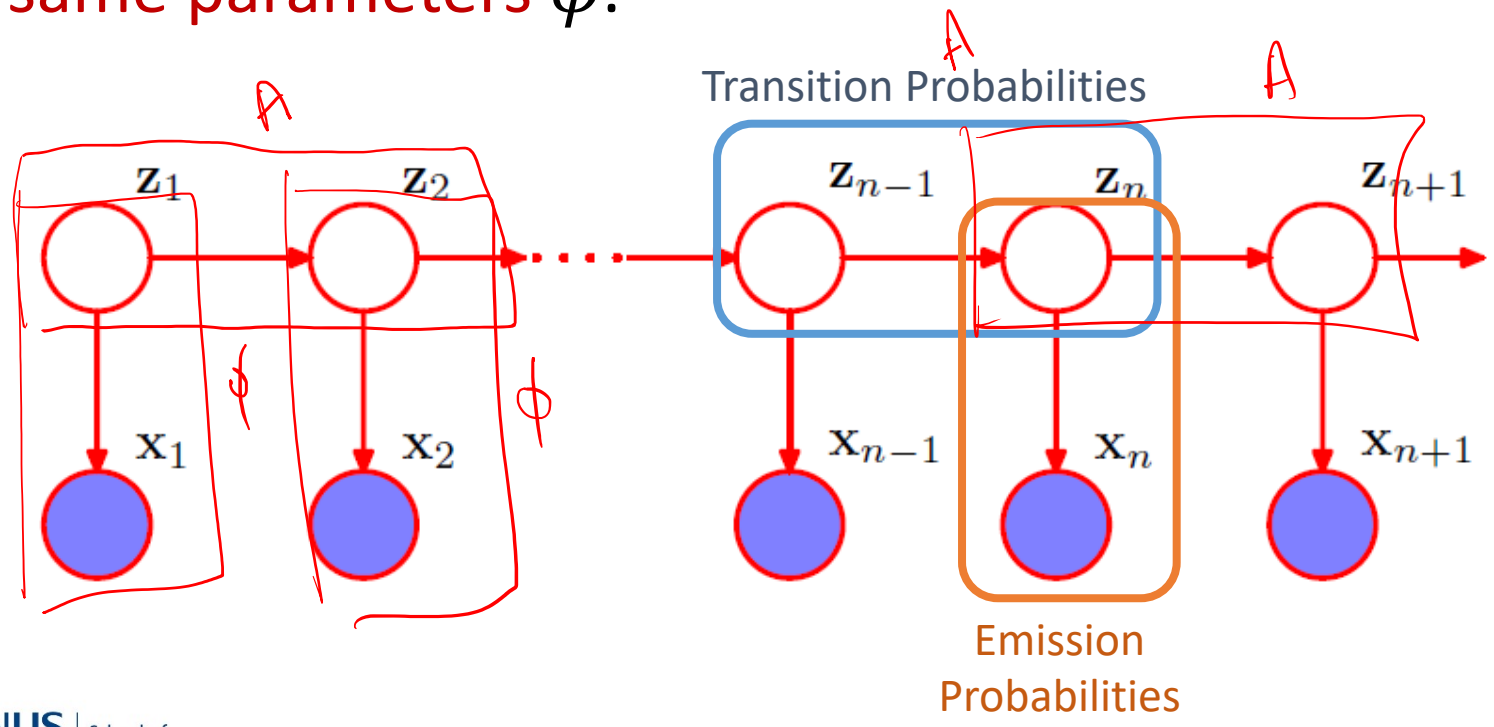




$$p(x_n \mid z_n, \phi_k) = \mathcal{N}(x_n \mid \mu_{k=1}, \Sigma_1)$$

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.



HMM: Joint Probability Revisited

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

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

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

Let's try it out.

https://github.com/crslab/CS5340-notebooks

NUS National University of Singapore | School of Computing

Learning the HMM Parameters

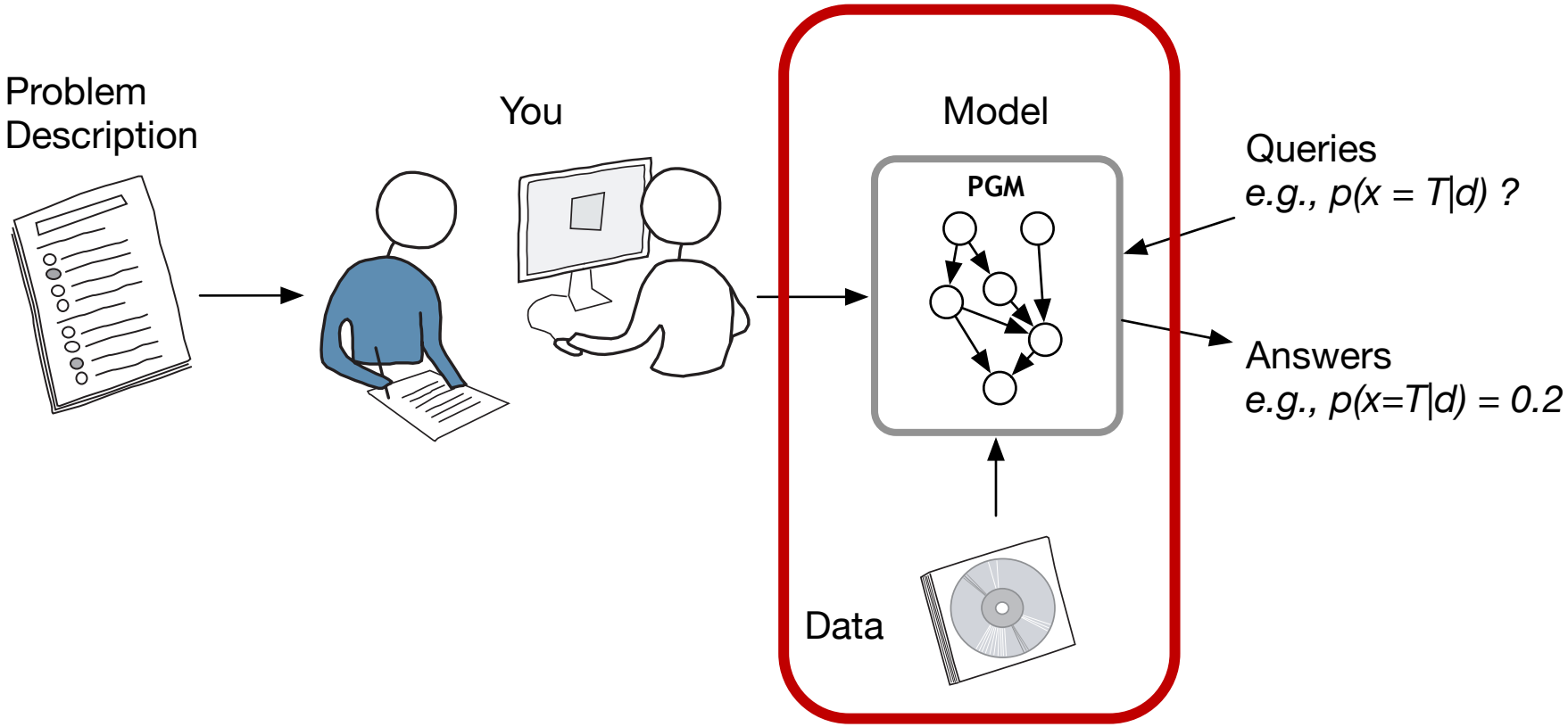
Expectation Maximization

Key Ideas

- How to construct a HMM from data?
- MLE estimate of the parameters $\theta = \{\pi, \boldsymbol{A}, \phi\}$ via the EM algorithm

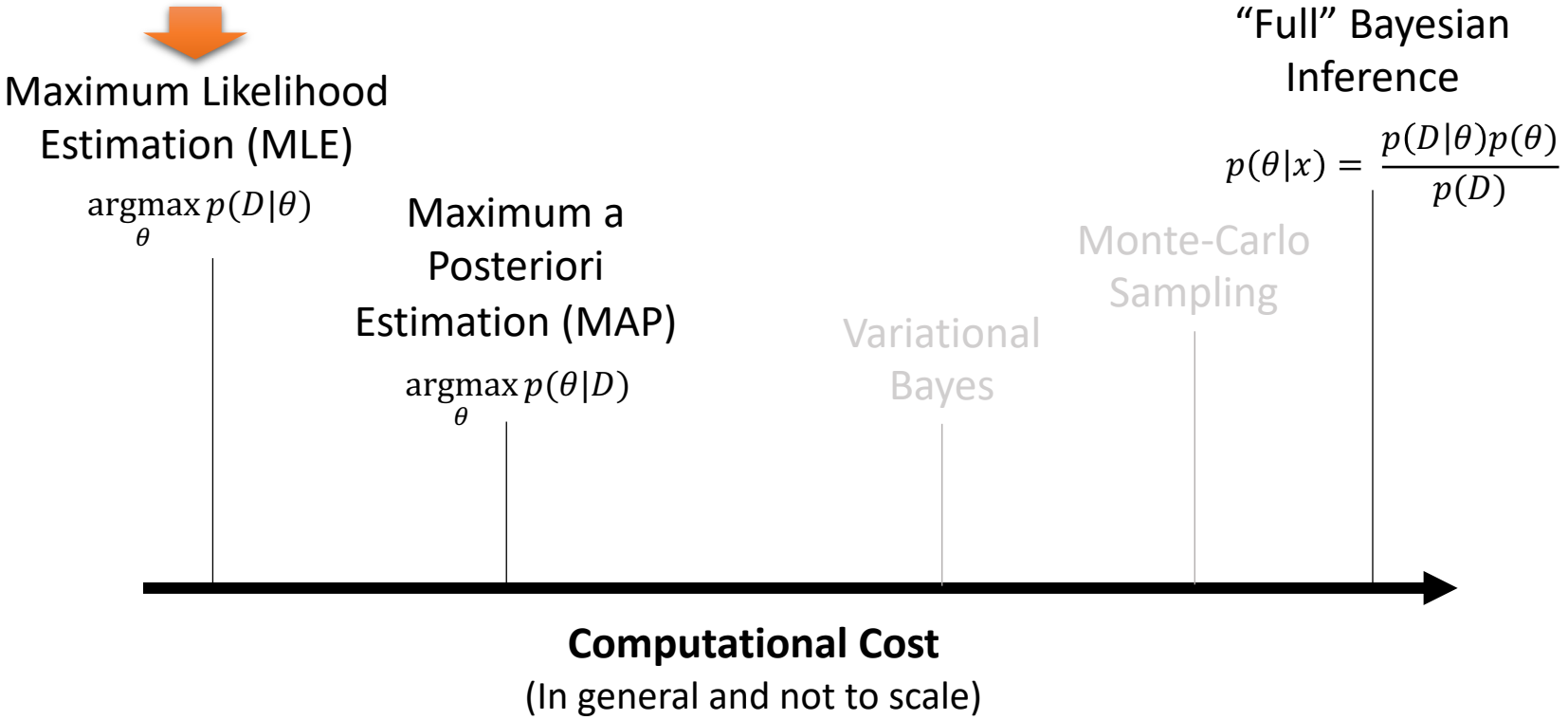
CS5340 in a nutshell

CS5340 is about how to "**represent**" and "**reason**" with uncertainty in a computer.



Learning Parameters

- Common approaches to learn the unknown parameters θ from a set of given data $\mathcal{D} = \{x[1], \ldots, x[N]\}$:



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
  - $p(X, Z|\theta)$ does not factorize over $Z_1, \ldots, Z_n$.
- Performing the sums over all N variables results in a exponential complexity of $O(K^N)$.

MLE for HMM: EM Algorithm

- **Solution?** EM algorithm to find an efficient framework for MLE

From Lecture 7: The General EM Algorithm

1. Choose an initial setting for the parameters θ^{old}.
2. Expectation step: Evaluate $p(\mathrm{Z}|\mathrm{X}, \theta^{old})$.
3. Maximization step: Evaluate θ^{new} given by:

$$\boldsymbol{\theta}^{\mathrm{new}} = \arg\max_{\boldsymbol{\theta}} \mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\mathrm{old}})$$

where

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

4. Check for convergence of either the log likelihood or the parameter values, if not converged:

$$\boldsymbol{\theta}^{\mathrm{old}} \leftarrow \boldsymbol{\theta}^{\mathrm{new}}$$

MLE for HMM: EM Algorithm

- **Solution?** EM algorithm to find an efficient framework for MLE
- The EM algorithm starts with initialization of the model parameters, which we denote by θ^{old}.
- **E step**: find the posterior distribution of the latent variables $p(Z|X, \theta^{old})$.
- **M step:** maximize the Q function to get $\boldsymbol{\theta}^{\text{new}}$

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

MLE for HMM: EM Algorithm

- Denote the marginal posterior distribution of a latent variable Z_n as:

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

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

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

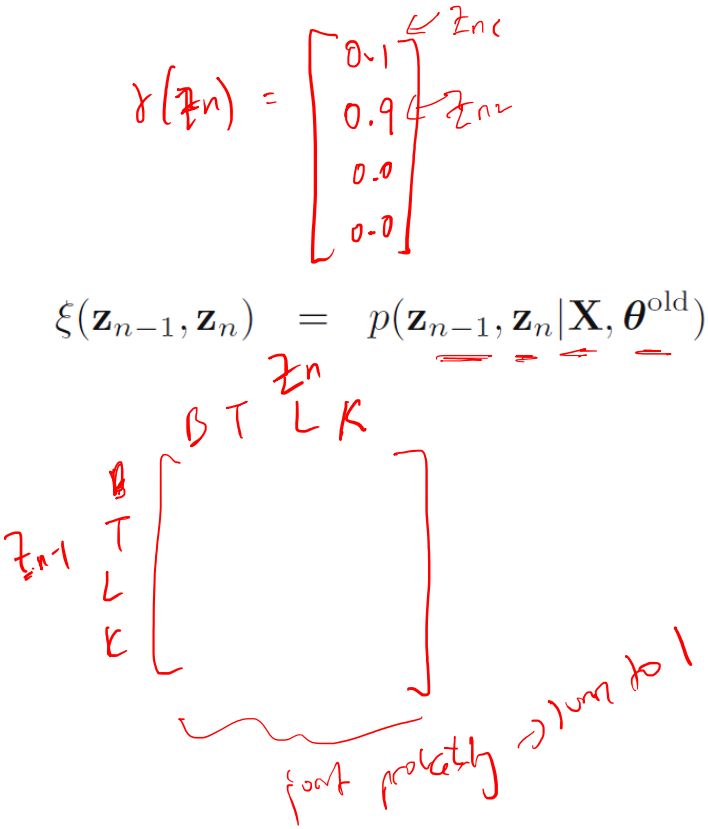
MLE for HMM: EM Algorithm

- For each value of n, store $\gamma(z_n)$ using a set of K nonnegative numbers that sum to 1.
- Store $\xi(z_{n-1}, z_n)$ using a $K \times K$ matrix of nonnegative numbers that again sum to 1.
- $\gamma(z_{nk})$ denotes the conditional probability of $z_{nk} = 1$
- Similarly for $\xi(z_{n-1,j}, z_{nk})$.

Example

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

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



MLE for HMM: EM Algorithm

- Putting everything together:

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

where

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

$$\prod_{k=1}^{K} \pi_k^{z_{1k}} \qquad \prod_{k=1}^{K} \prod_{j=1}^{K} A_{jk}^{z_{n-1,j} z_{nk}} \qquad \prod_{k=1}^{K} p(\mathbf{x} \ |\boldsymbol{\phi}_k)^{z_{mk}}$$

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

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

- 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:

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

,

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

MLE for HMM: EM Algorithm

- 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:

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

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 M-step equations are given by:

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

We need $\gamma(z_n)$ and $\xi(z_{n-1,j}, z_{nk})$

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

- 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.

NUS National University of Singapore | School of Computing

The Forward-Backward Algorithm

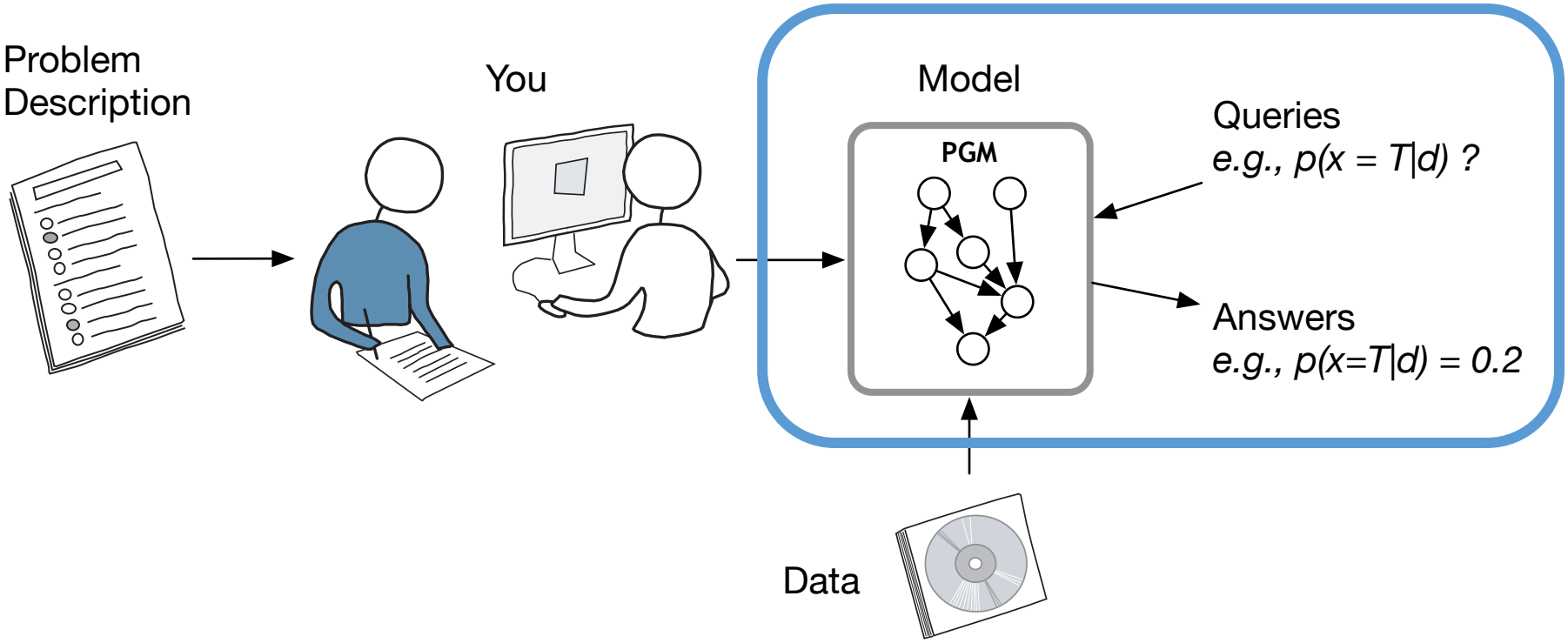
Computing the posteriors efficiently

Key Ideas

- How to compute $\gamma(z_n)$ and $\xi(z_{n-1}, z_n)$?
  - Note that the above are posteriors (conditional distributions)

CS5340 in a nutshell

CS5340 is about how to "**represent**" and "**reason**" with uncertainty in a computer.



Key Ideas

- How to compute $\gamma(z_n)$ and $\xi(z_{n-1}, z_n)$?
  - Note that the above are posteriors (conditional distributions)
  - We are performing inference!
    - Please go back to L5 and L6 to revise if needed

Key Ideas

- How to compute $\gamma(z_n)$ and $\xi(z_{n-1}, z_n)$?
  - Note that the above are posteriors (conditional distributions)
  - We are performing inference!
    - Please go back to L5 and L6 to revise if needed
- Introduce the forward-backward algorithm
  - Relationship to the Factor-Tree Sum-Product Algorithm (Lecture 6)
  - Scaling Factors

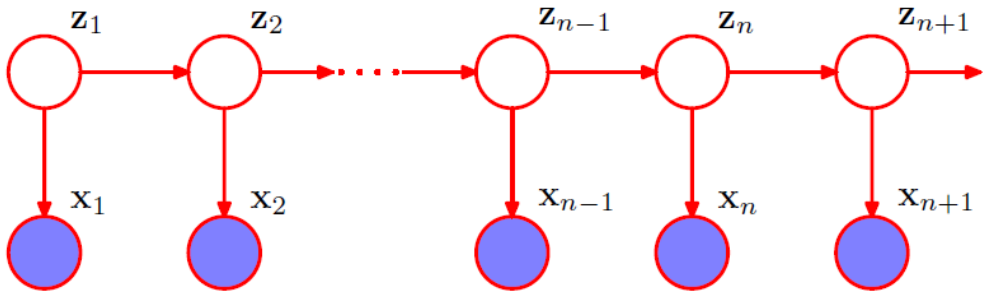
The Forward-Backward Algorithm

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

- Many variants of the algorithm, focus on the most widely-used alpha-beta algorithm.
  - Recursive forms.
  - First derive the alpha-beta algorithm
  - Then show connection to sum-product

$\alpha(z_{n+1}) = f(\alpha(z_n))$

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 $p(z_n|x_1, \ldots, x_N)$.

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

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

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

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

The Forward-Backward Algorithm

We have defined:

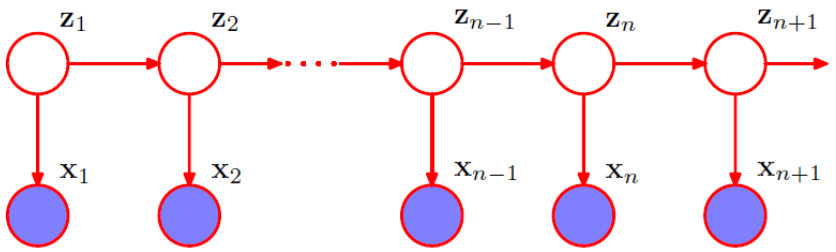
$$\alpha(\mathbf{z}_n) \equiv p(\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{z}_n)$$
$$\beta(\mathbf{z}_n) \equiv p(\mathbf{x}_{n+1}, \ldots, \mathbf{x}_N | \mathbf{z}_n)$$

- $\alpha(z_n)$ 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:

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



Proof Sketch:

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

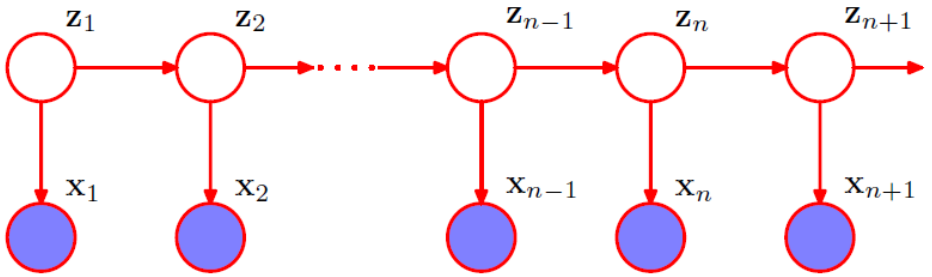
$\alpha(z_{n-1})$

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

The Forward-Backward Algorithm

Forward recursion:

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



- 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 $O(K^2)$.

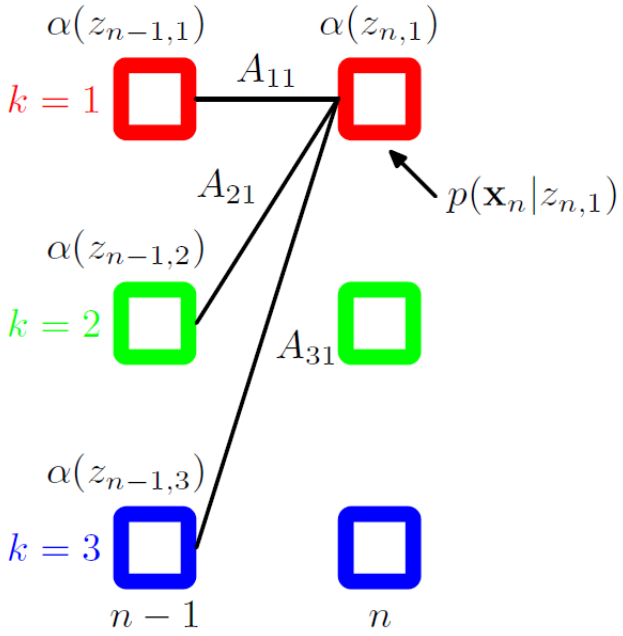
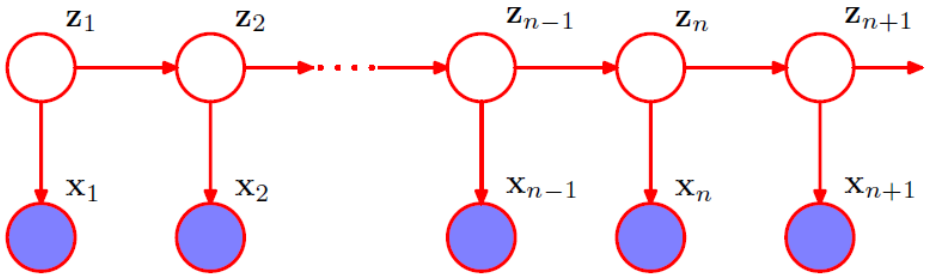


Image source: "Pattern recognition and machine learning", Christopher Bishop

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. $p(z_n|z_{n-1})$
3. and then multiplying by the data contribution $p(x_n|z_{n1})$.

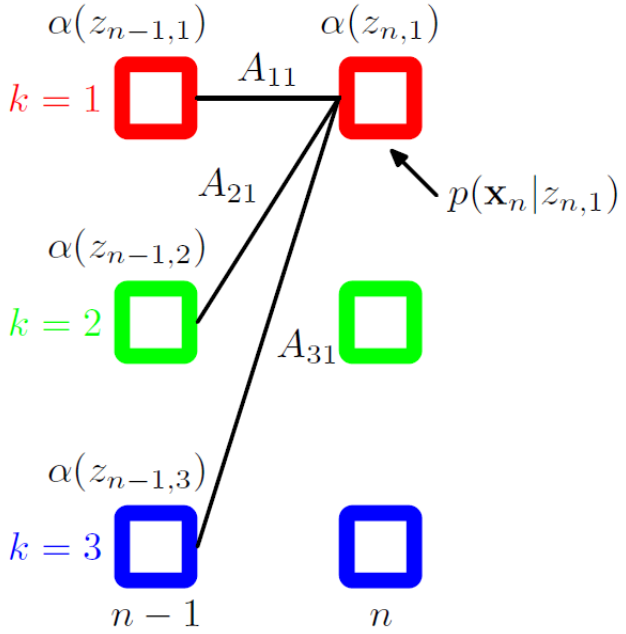
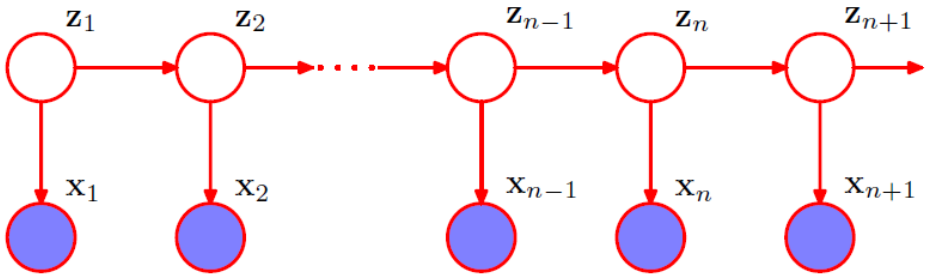


Image source: "Pattern recognition and machine learning", Christopher Bishop

The Forward-Backward Algorithm

Forward recursion:

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



- Initialization:

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

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

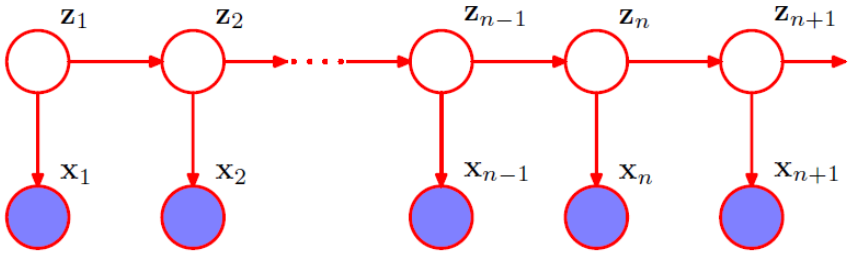
- Total complexity for the whole chain: $O(K^2 N)$.

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

The Forward-Backward Algorithm

Backward recursion:

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



Proof Sketch:

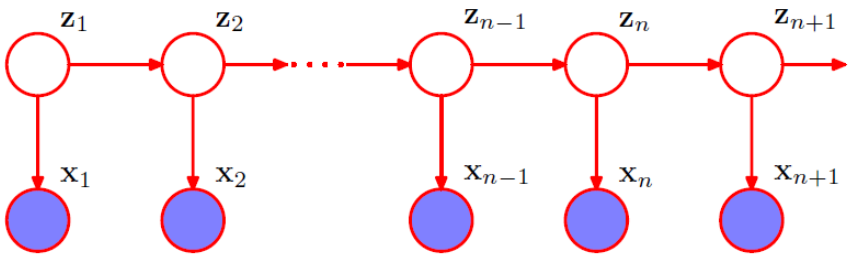
$$\begin{aligned}
\beta(\mathbf{z}_n) &= p(\mathbf{x}_{n+1}, \ldots, \mathbf{x}_N | \mathbf{z}_n) \\
&= \sum_{\mathbf{z}_{n+1}} p(\mathbf{x}_{n+1}, \ldots, \mathbf{x}_N, \mathbf{z}_{n+1} | \mathbf{z}_n) && \text{(marginalization)} \\
&= \sum_{\mathbf{z}_{n+1}} p(\mathbf{x}_{n+1}, \ldots, \mathbf{x}_N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

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

The Forward-Backward Algorithm

Backward recursion:

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



$\beta(z_{n1})$ 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 $p(x_{n+1}|z_{n+1,k})$.

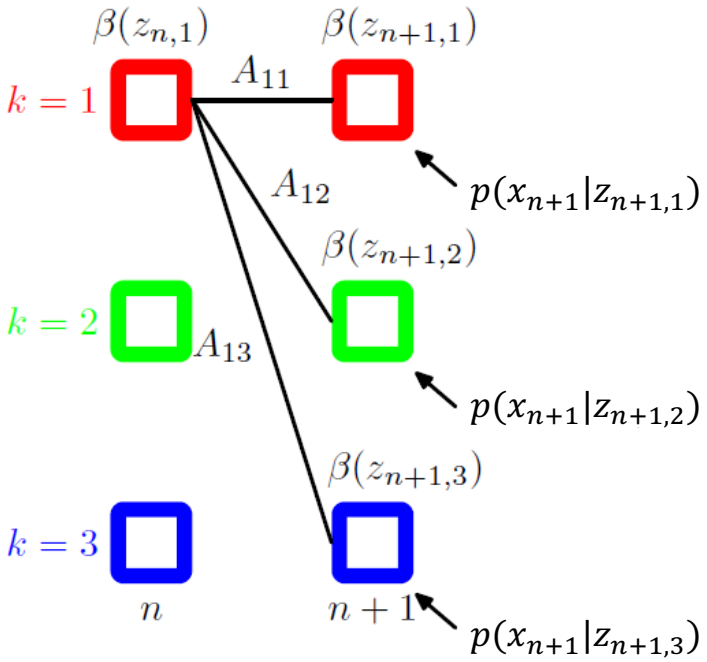


Image source: "Pattern recognition and machine learning", Christopher Bishop

The Forward-Backward Algorithm

Questions to test your understanding:

1. What is the value of $\beta(\boldsymbol{z}_N)$?
   - i.e. the "first" $\beta(\boldsymbol{z}_N)$
2. What is the computational complexity of the backward pass?

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).

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

$$= \frac{p(\mathbf{X}|\mathbf{z}_{n-1}, \mathbf{z}_n)p(\mathbf{z}_{n-1}, \mathbf{z}_n)}{p(\mathbf{X})} \quad \text{(Bayes' rule)}$$

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

(Conditional Independence)

Forward recursion — Backward recursion

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

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

Summary:

- We can compute $\gamma(z_n)$ and $\xi(z_{n-1}, z_n)$ *efficiently*

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

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

- Where we can compute $\alpha(z_n)$ and $\beta(z_n)$ recursively:

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

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

The Forward-Backward Algorithm

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

- Many variants of the algorithm, focus on the most widely-used alpha-beta algorithm.
  - Recursive forms.
  - First derive the alpha-beta algorithm
  - Then show connection to sum-product

Recall from Lecture 6: Factor Tree Sum-Product Algorithm

SUM-PRODUCT($\mathcal{T}$, E) // main steps of **Sum-Product algorithm**

1. EVIDENCE(E)
   f = CHOOSEROOT($\mathcal{V}$)
2. **for** $s \in \mathcal{N}(f)$
   μ-COLLECT(f, s)
3. **for** $s \in \mathcal{N}(f)$
   ν-DISTRIBUTE(f, s)
4. **for** $i \in \mathcal{V}$
   COMPUTEMARGINAL(i)

1. EVIDENCE(E) // add **evidence potentials** (convert conditioning into marginalization)
   for $i \in E$
   $\psi^E(x_i) = \psi(x_i)\delta(x_i, \bar{x}_i)$
   for $i \notin E$
   $\psi^E(x_i) = \psi(x_i)$

2. μ-COLLECT(i, s) // recursively collect messages from leaves to root
   for $j \in \mathcal{N}(s) \backslash i$
   ν-COLLECT(s, j)
   μ-SENDMESSAGE(s, i)

   ν-COLLECT(s, i)
   for $t \in \mathcal{N}(i) \backslash s$
   μ-COLLECT(i, t)
   ν-SENDMESSAGE(i, s)

Message from variable node X_i to the factor node f_s:
ν-SENDMESSAGE(i, s)

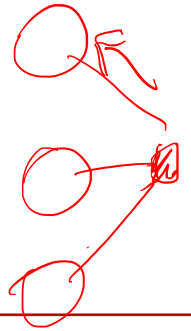
$$\nu_{is}(x_i) = \prod_{t \in \mathcal{N}(i) \backslash s} \mu_{ti}(x_i)$$

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

Recall from Lecture 6: Factor Tree Sum-Product Algorithm

SUM-PRODUCT($\mathcal{T}$, E) // main steps of **Sum-Product algorithm**

1. EVIDENCE(E)
   f = CHOOSEROOT($\mathcal{V}$)
2. **for** $s \in \mathcal{N}(f)$
   μ-COLLECT(f, s)
3. **for** $s \in \mathcal{N}(f)$
   ν-DISTRIBUTE(f, s)
4. **for** $i \in \mathcal{V}$
   COMPUTEMARGINAL(i)



1. EVIDENCE(E) // add **evidence potentials** (convert conditioning into marginalization)
   for $i \in E$
   $\psi^E(x_i) = \psi(x_i)\delta(x_i, \bar{x}_i)$
   for $i \notin E$
   $\psi^E(x_i) = \psi(x_i)$

2. μ-COLLECT(i, s) // recursively collect messages from leaves to root
   for $j \in \mathcal{N}(s) \backslash i$
   ν-COLLECT(s, j)
   μ-SENDMESSAGE(s, i)

   Message from factor node f_s to the variable node X_i:

   $$\mu_{si}(x_i) = \sum_{x_{\mathcal{N}(s)\backslash i}} \left(f_s(x_{\mathcal{N}(s)}) \prod_{j \in \mathcal{N}(s)\backslash i} \nu_{js}(x_j) \right)$$

   ν-COLLECT(s, i)
   for $t \in \mathcal{N}(i) \backslash s$
   μ-COLLECT(i, t)
   ν-SENDMESSAGE(i, s)

   Message from variable node X_i to the factor node f_s:

   $$\nu_{is}(x_i) = \prod_{t \in \mathcal{N}(i)\backslash s} \mu_{ti}(x_i)$$

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

Recall from Lecture 6: Factor Tree Sum-Product Algorithm

SUM-PRODUCT($\mathcal{T}$, E) // main steps of **Sum-Product algorithm**

1. EVIDENCE(E)
   f = CHOOSEROOT($\mathcal{V}$)
2. **for** $s \in \mathcal{N}(f)$
   μ-COLLECT(f, s)
3. **for** $s \in \mathcal{N}(f)$
   ν-DISTRIBUTE(f, s)
4. **for** $i \in \mathcal{V}$
   COMPUTEMARGINAL(i)

3. ν-DISTRIBUTE(i, s) // distribute messages from root to leaves
   ν-SENDMESSAGE(i, s)
   for $j \in \mathcal{N}(s) \backslash i$
   μ-DISTRIBUTE(s, j)

μ-DISTRIBUTE(s, i)
   μ-SENDMESSAGE(s, i)
   for $t \in \mathcal{N}(i) \backslash s$
   ν-DISTRIBUTE(i, t)

Message from variable node X_i to the factor node f_s:

ν-SENDMESSAGE(i, s)

$$\nu_{is}(x_i) = \prod_{t \in \mathcal{N}(i) \backslash s} \mu_{ti}(x_i)$$

Message from factor node f_s to the variable node X_i:

μ-SENDMESSAGE(s, i)

$$\mu_{si}(x_i) = \sum_{x_{\mathcal{N}(s) \backslash i}} \left(f_s(x_{\mathcal{N}(s)}) \prod_{j \in \mathcal{N}(s) \backslash i} \nu_{js}(x_j) \right)$$

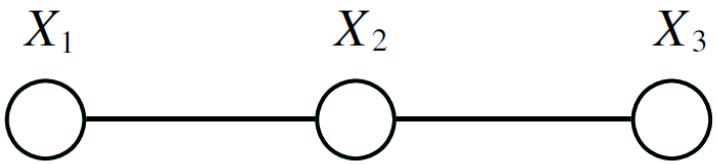
4. COMPUTEMARGINAL(i) // compute **marginal probability**
   $p(x_i) \propto \nu_{is}(x_i) \mu_{si}(x_i)$

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

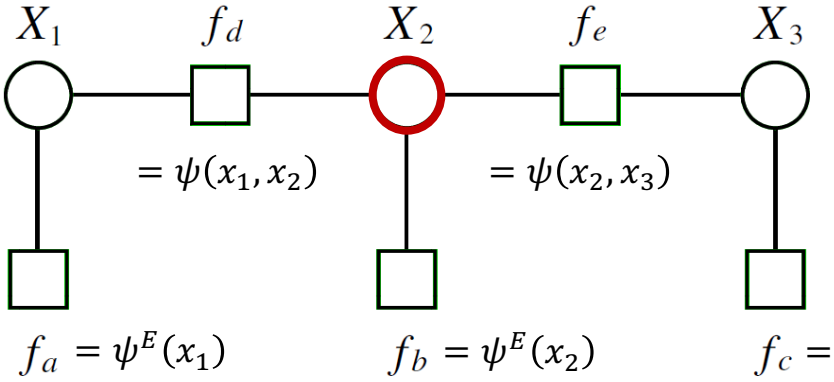
Recall from Lecture 6: Factor Tree Sum-Product Algorithm

Example:

$$p(x|\bar{x}_E) = \frac{1}{Z^E}\left(\psi^E(x_1)\psi^E(x_2)\psi^E(x_3)\psi(x_1,x_2)\psi(x_2,x_3)\right)$$



Convert UGM into a factor graph

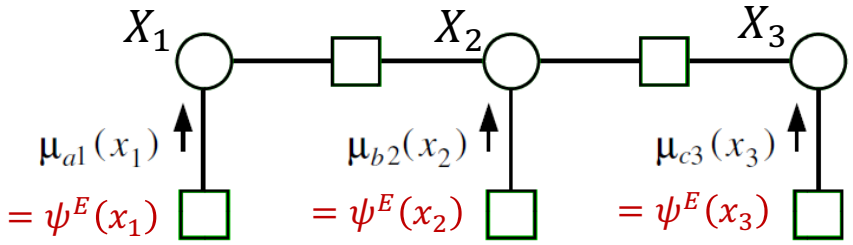


Choose X_2 as root node

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

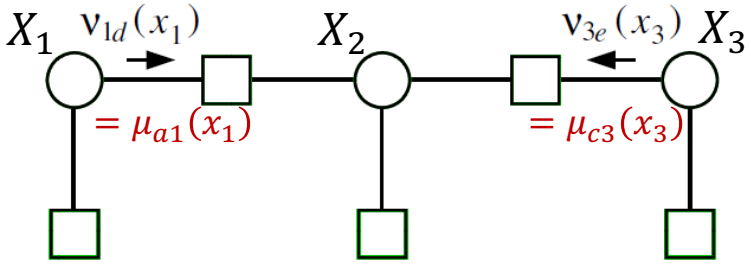
Recall from Lecture 6: Factor Tree Sum-Product Algorithm

Example:



Collect messages from leaf nodes:

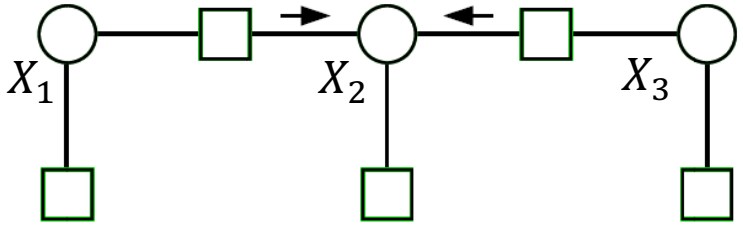
$$\mu_{si}(x_i) = f_s(x_i) = \psi^E(x_i)$$



Collect variable to factor messages:

$$\nu_{is}(x_i) = \prod_{t \in \mathcal{N}(i) \setminus s} \mu_{ti}(x_i)$$

$\mu_{d2}(x_2) = \sum_{x_1} \psi(x_1, x_2)\mu_{a1}(x_1)$ $\quad$ $\mu_{e2}(x_2) = \sum_{x_3} \psi(x_2, x_3)\mu_{c3}(x_3)$



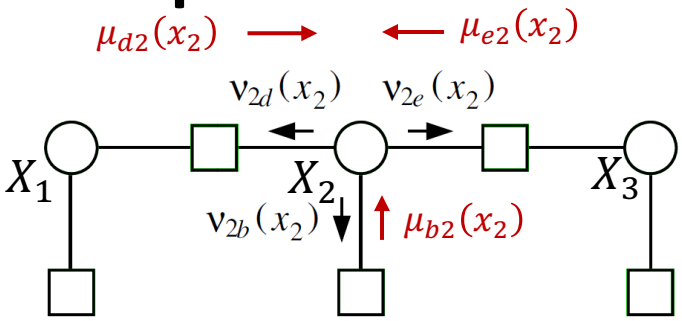
Collect factor to variable messages:

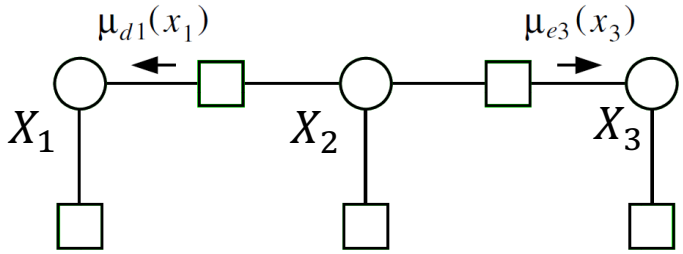
$$\mu_{si}(x_i) = \sum_{x_{\mathcal{N}(s) \setminus i}} \left(f_s(x_{\mathcal{N}(s)}) \prod_{j \in \mathcal{N}(s) \setminus i} \nu_{js}(x_j) \right)$$

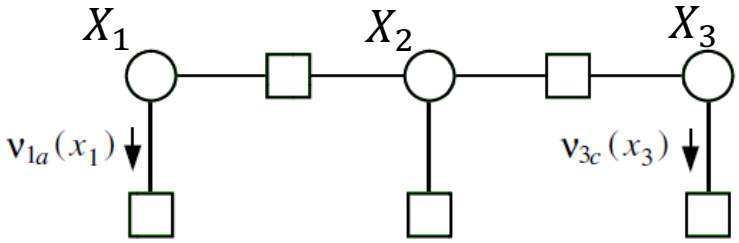
Image Source: "An introduction to probabilistic graphical models", Michael I. Jordan, 2002.

Recall from Lecture 6: Factor Tree Sum-Product Algorithm

Example:







Distribute variable to factor messages:

$$\nu_{is}(x_i) = \prod_{t \in \mathcal{N}(i) \setminus s} \mu_{ti}(x_i)$$

$$\upsilon_{2b}(x_2) = \mu_{d2}(x_2)\mu_{e2}(x_2)$$

$$\upsilon_{2d}(x_2) = \mu_{b2}(x_2)\mu_{e2}(x_2)$$

$$\upsilon_{2e}(x_2) = \mu_{b2}(x_2)\mu_{d2}(x_2)$$

Distribute factor to variable messages:

$$\mu_{si}(x_i) = \sum_{x_{\mathcal{N}(s) \setminus i}} \left(f_s(x_{\mathcal{N}(s)}) \prod_{j \in \mathcal{N}(s) \setminus i} \nu_{js}(x_j) \right)$$

$$\mu_{d1}(x_1) = \sum_{x_2} \psi(x_1, x_2)\upsilon_{2d}(x_2)$$

$$\mu_{e3}(x_3) = \sum_{x_2} \psi(x_2, x_3)\upsilon_{2e}(x_2)$$

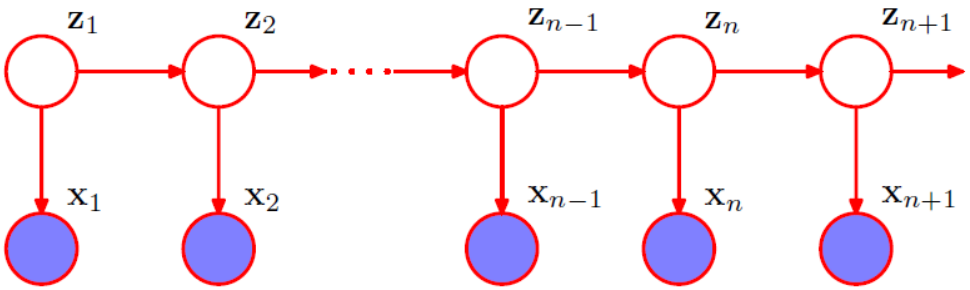
Distribute variable to factor messages:

$$\nu_{is}(x_i) = \prod_{t \in \mathcal{N}(i) \setminus s} \mu_{ti}(x_i)$$

$$\upsilon_{1a}(x_1) = \mu_{d1}(x_1), \quad \upsilon_{3c}(x_3) = \mu_{e3}(x_3)$$

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

HMM: Sum-of-Product Algorithm



Convert the Bayesian network into a factor graph

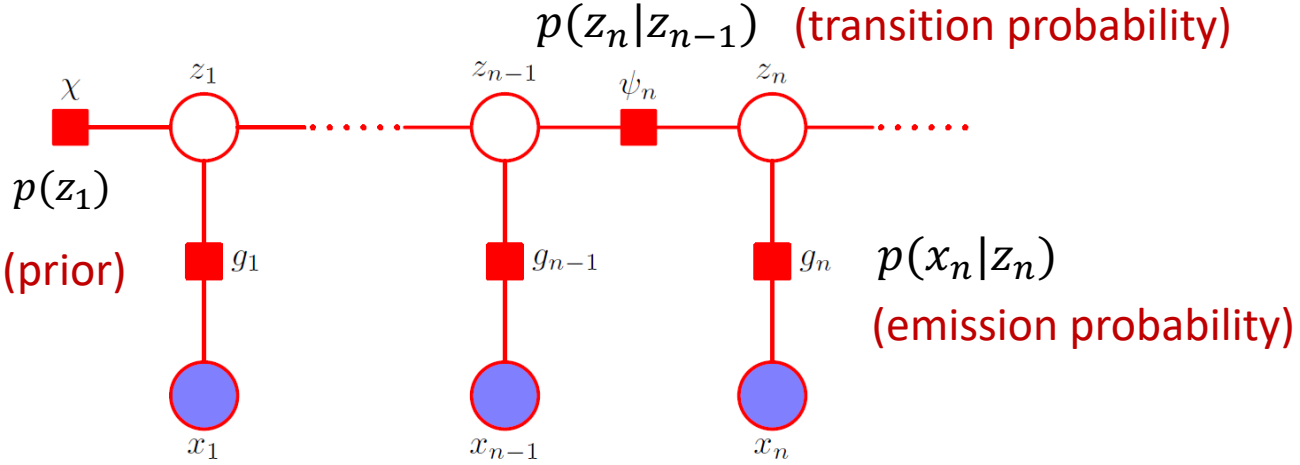
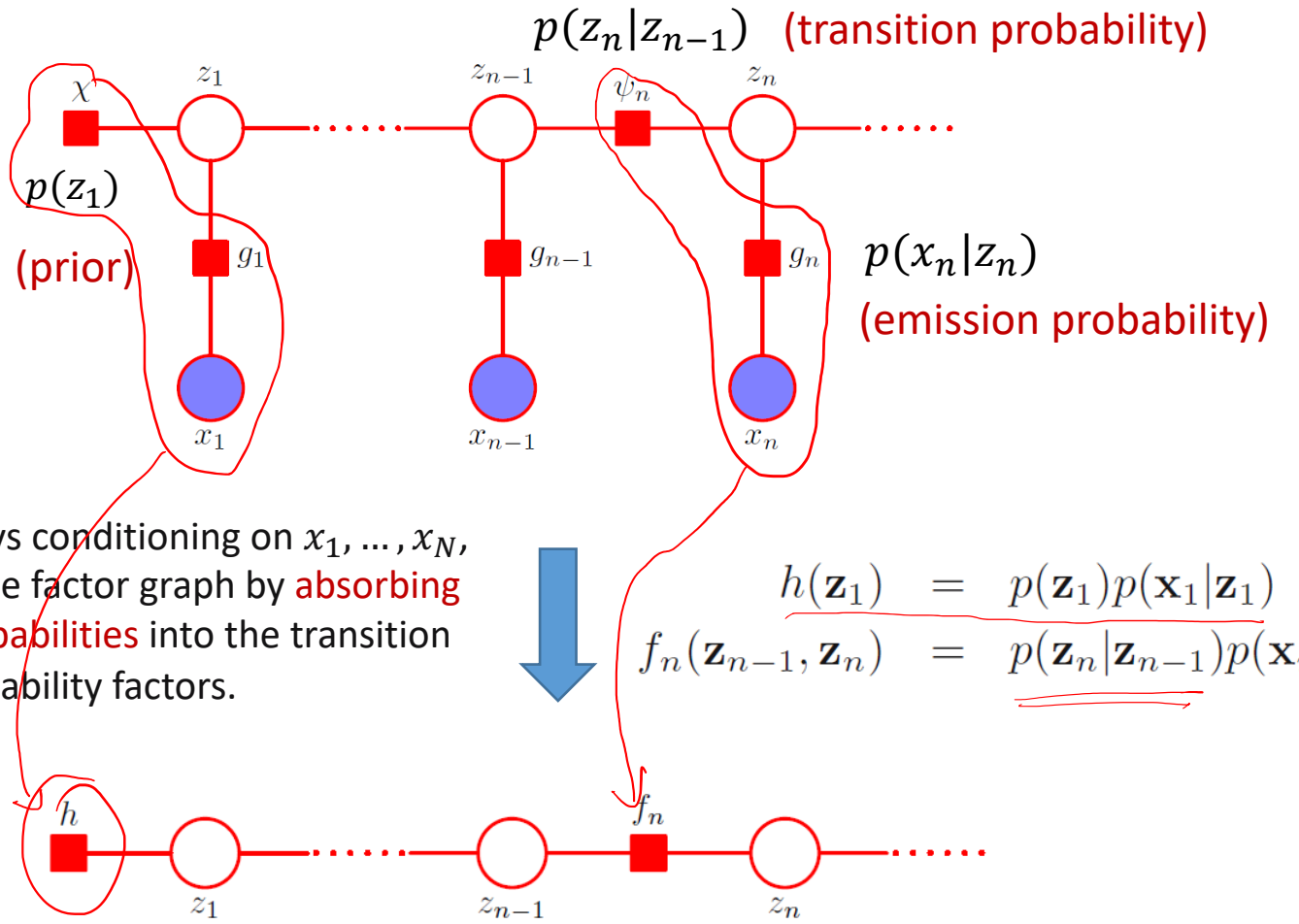


Image source: "Pattern recognition and machine learning", Christopher Bishop

HMM: Sum-of-Product Algorithm



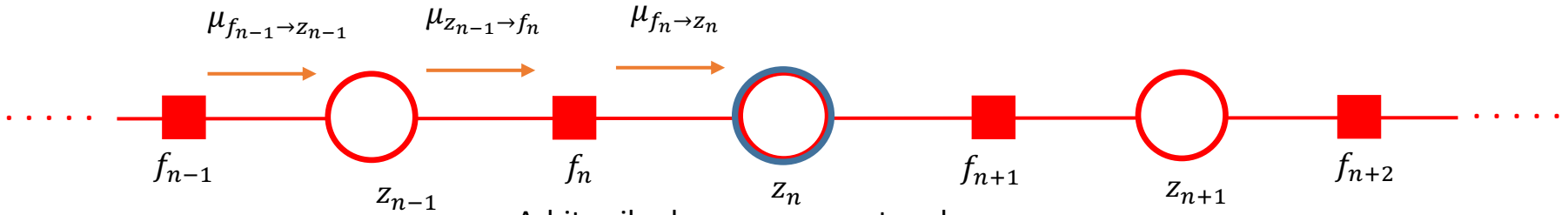
Since we are always conditioning on $x_1, \ldots, x_N$, we can simplify the factor graph by absorbing the emission probabilities into the transition probability factors.

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

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

HMM: Sum-of-Product Algorithm

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



Arbitrarily choose z_n as root node

- Messages from the left towards the root node:

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

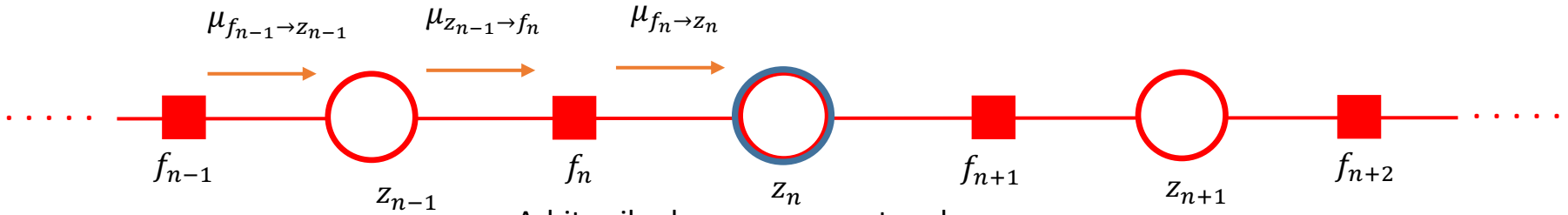
NO computation since there is only two neighbor nodes!

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

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



Arbitrarily choose z_n as root node

- Messages from the left towards the root node:

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

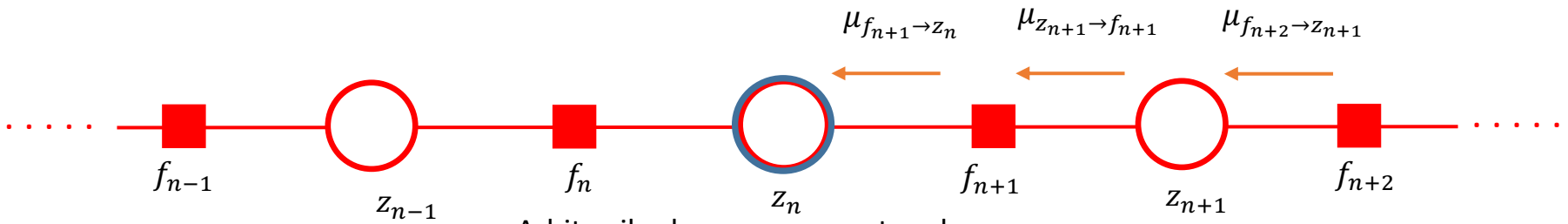
$$\alpha(\mathbf{z}_n) = p(\mathbf{x}_n|\mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} \alpha(\mathbf{z}_{n-1})p(\mathbf{z}_n|\mathbf{z}_{n-1})$$

Same as forward recursion!

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

HMM: Sum-of-Product Algorithm

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



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!

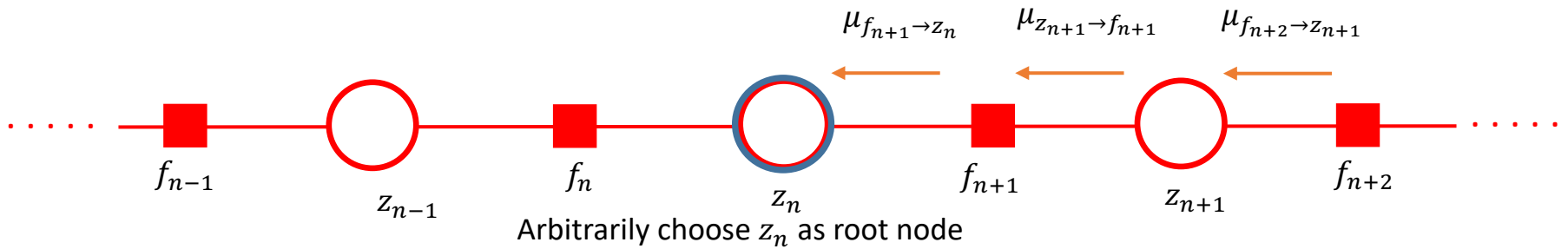
Factor-to-node: $\mu_{f_{n+1}\to z_n}(z_n) = \sum_{z_{n+1}} f_{n+1}(z_n, z_{n+1})\mu_{z_{n+1}\to f_{n+1}}(z_{n+1})$

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

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

HMM: Sum-of-Product Algorithm

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



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

The Forward-Backward Algorithm

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

- Many variants of the algorithm, focus on the most widely-used alpha-beta algorithm.
  - Recursive forms.
  - First derive the alpha-beta algorithm
  - Then show connection to sum-product

Summary:

- We can compute $\gamma(z_n)$ and $\xi(z_{n-1}, z_n)$ *efficiently*

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

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

- Where we can compute $\alpha(z_n)$ and $\beta(z_n)$ recursively:

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

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

Code Example and Numerical Issues

EM and Forward Backward in Practice

From Lecture 7: The General EM Algorithm

1. Choose an initial setting for the parameters θ^{old}.
2. Expectation step: Evaluate $p(\mathrm{Z}|\mathrm{X}, \theta^{old})$.
3. Maximization step: Evaluate θ^{new} given by:

$$\boldsymbol{\theta}^{\mathrm{new}} = \arg\max_{\boldsymbol{\theta}} \mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\mathrm{old}})$$

where

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

4. Check for convergence of either the log likelihood or the parameter values, if not converged:

$$\boldsymbol{\theta}^{\mathrm{old}} \leftarrow \boldsymbol{\theta}^{\mathrm{new}}$$

MLE for HMM: EM Algorithm

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

- 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.

Forward Backward Algorithm

- We can compute $\gamma(z_n)$ and $\xi(z_{n-1}, z_n)$ *efficiently*

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

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

- Where we can compute $\alpha(z_n)$ and $\beta(z_n)$ recursively:

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

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

Let's try it out.

https://github.com/crslab/CS5340-notebooks

Forward Backward Algorithm

- We can compute $\gamma(z_n)$ and $\xi(z_{n-1}, z_n)$ *efficiently*

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

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

- Where we can compute $\alpha(z_n)$ and $\beta(z_n)$ recursively:

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

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

Numerical Issues

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

- 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 Factor for α

- 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:
  - it is a probability distribution over K variables for any value of n.
  - Compare to: $\alpha(\mathbf{z}_n) \quad \equiv \quad p(\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{z}_n)$

Scaling Factor for α

- Rewriting the forward-recursion into the normalized form:

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

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

$$c_n = p(x_n|x_1,\dots,x_{n-1}) \quad \hat{\alpha}(z_n) \qquad \hat{\alpha}(z_{n-1})$$

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

Scaling Factor for α

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

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

  $c_n = p(x_n|x_1, \ldots, x_{n-1})$

  Sum over all k entries in Z_n

  Proof Sketch:

$$c_n = \sum_{z_n} \tilde{\alpha}(\mathbf{z}_n)$$

$$= \sum_{z_n} p(x_n|z_n) \sum_{z_{n-1}} \hat{\alpha}(z_{n-1}) p(z_n|z_{n-1})$$

$$= \sum_{z_n} p(x_n|z_n) \sum_{z_{n-1}} \frac{\alpha(z_{n-1})}{p(x_1,\ldots,x_{n-1})} p(z_n|z_{n-1})$$

$$= \sum_{z_n} \frac{p(x_1, \ldots, x_n, z_n)}{p(x_1, \ldots, x_{n-1})} = \frac{p(x_1, \ldots, x_n)}{p(x_1, \ldots, x_{n-1})} = p(x_n|x_1, \ldots, x_{n-1})$$

Scaling Factor for β

- 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:

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

$\gamma(z_n)$ and $\xi(z_{n-1}, z_n)$ with Scaling Factors

- We get:

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

Proof Sketch:

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

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

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

Scaling Factor

- As a result, we get:

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

Proof Sketch:

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

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

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

Summary (In Practice)

- We can compute $\gamma(z_n)$ and $\xi(z_{n-1}, z_n)$ *efficiently*

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

- Where we can compute $\hat{\alpha}\ (z_n)$ and $\hat{\beta}(z_n)$ recursively:

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

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

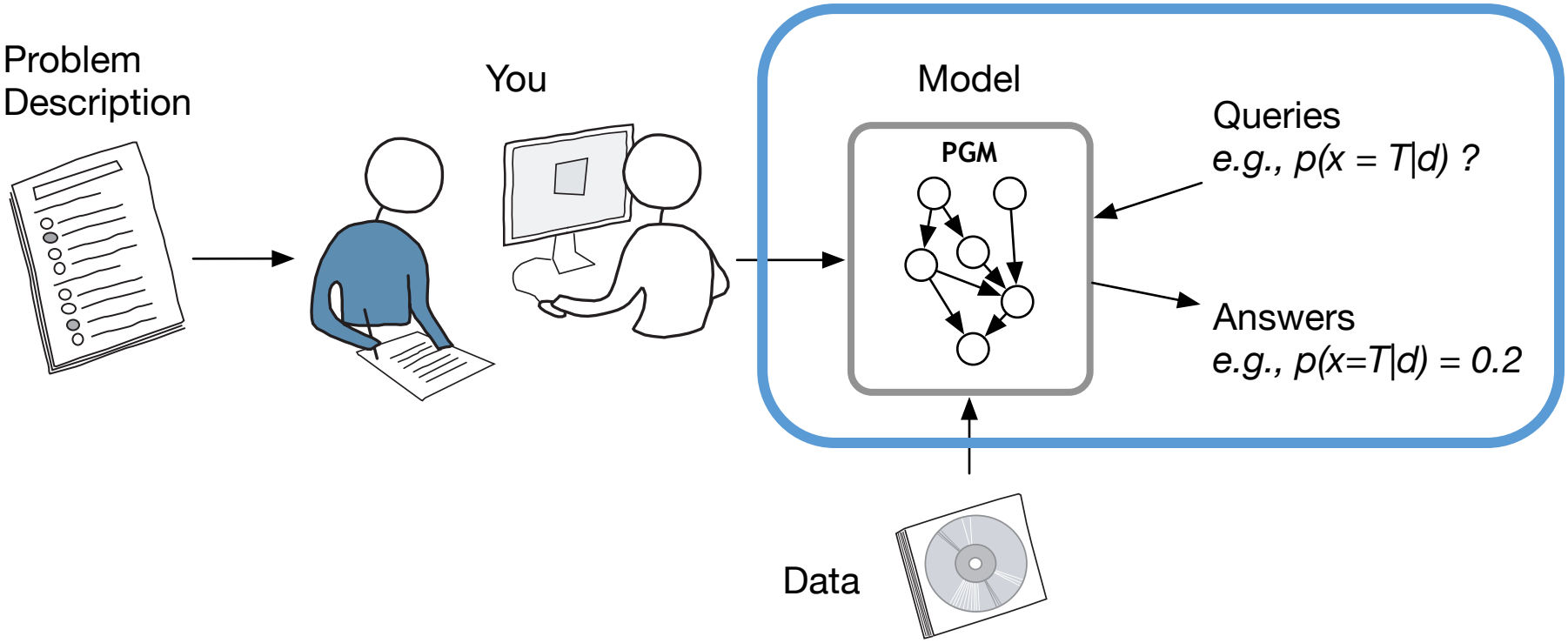
NUS National University of Singapore | School of Computing

More HMM Inference

Predictive Distribution, Viterbi Algorithm

CS5340 in a nutshell

CS5340 is about how to "**represent**" and "**reason**" with uncertainty in a computer.

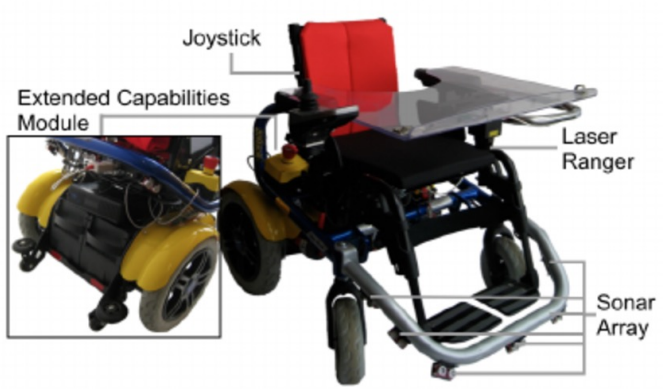


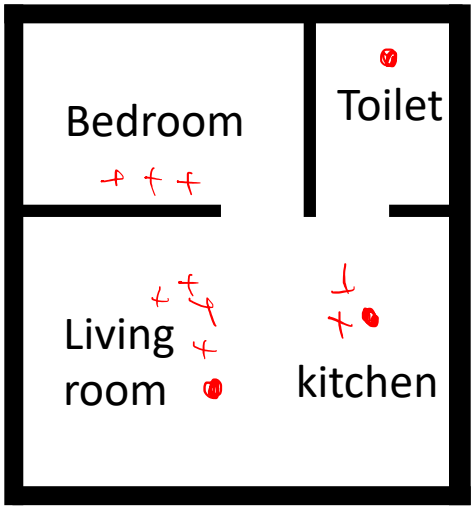
Key ideas

- How to compute a predictive distribution given some observed data?
- What is the most likely sequence of hidden states given the observed data?
  - Viterbi algorithm
  - Max-product / Max-sum algorithm

Predictive Distribution

- Modeling Human Trajectories
  - E.g., for intent prediction
- Noisy sensors





$p(x_{n+1} \mid x_n)$

Predictive Distribution

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

$$p(\mathbf{x}_{N+1}|\mathbf{X}) = \sum_{\mathbf{z}_{N+1}} p(\mathbf{x}_{N+1}, \mathbf{z}_{N+1}|\mathbf{X}) \quad \text{(marginalization)}$$

$$= \sum_{\mathbf{z}_{N+1}} p(\mathbf{x}_{N+1}|\mathbf{z}_{N+1}) p(\mathbf{z}_{N+1}|\mathbf{X}) \quad \text{(product rule)}$$

$$= \sum_{\mathbf{z}_{N+1}} p(\mathbf{x}_{N+1}|\mathbf{z}_{N+1}) \sum_{\mathbf{z}_N} p(\mathbf{z}_{N+1}, \mathbf{z}_N|\mathbf{X}) \quad \text{(marginalization)}$$

$$= \sum_{\mathbf{z}_{N+1}} p(\mathbf{x}_{N+1}|\mathbf{z}_{N+1}) \sum_{\mathbf{z}_N} p(\mathbf{z}_{N+1}|\mathbf{z}_N) p(\mathbf{z}_N|\mathbf{X}) \quad \text{(product rule)}$$

$$= \sum_{\mathbf{z}_{N+1}} p(\mathbf{x}_{N+1}|\mathbf{z}_{N+1}) \sum_{\mathbf{z}_N} p(\mathbf{z}_{N+1}|\mathbf{z}_N) \frac{p(\mathbf{z}_N, \mathbf{X})}{p(\mathbf{X})} \quad \text{(product rule)}$$

$$= \frac{1}{p(\mathbf{X})} \sum_{\mathbf{z}_{N+1}} p(\mathbf{x}_{N+1}|\mathbf{z}_{N+1}) \sum_{\mathbf{z}_N} p(\mathbf{z}_{N+1}|\mathbf{z}_N) \underbrace{\alpha(\mathbf{z}_N)}$$

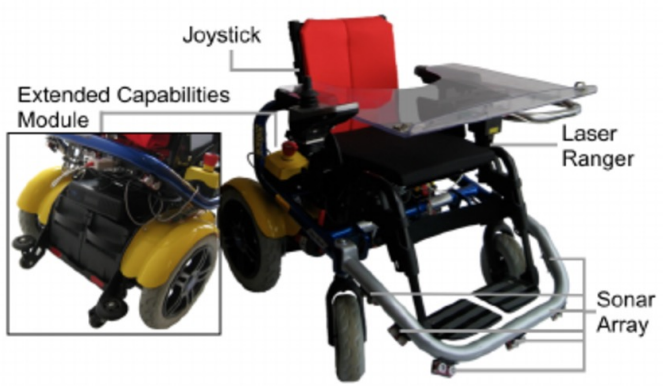
Forward recursion

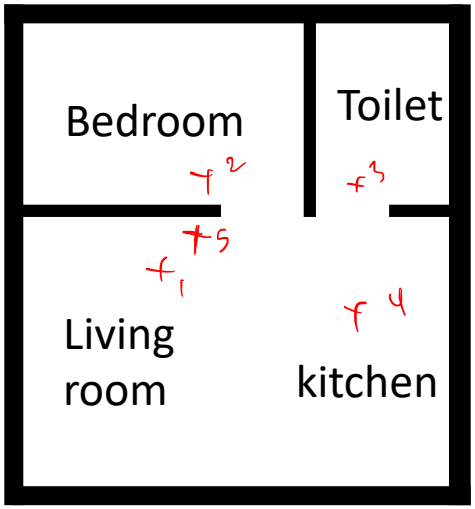
Key ideas

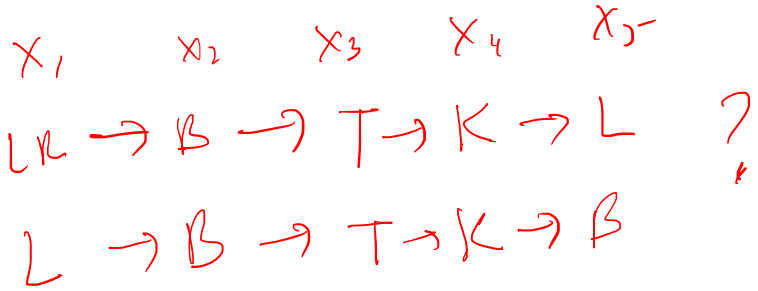
- How to compute a predictive distribution given some observed data?
- What is the most likely sequence of hidden states given the observed data?
  - Viterbi algorithm
  - Max-product / Max-sum algorithm

Most Probable Sequence

- Modeling Human Trajectories
  - E.g., for intent prediction
- Noisy sensors







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 ≠ 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.

L5: Max-Product Algorithm for Trees

MAX-PRODUCT($\mathcal{T}$, E) // main steps of the "MAP-Product Algorithm" for a tree $\mathcal{T}(\mathcal{V}, \mathcal{E})$

EVIDENCE(E)

f = CHOOSEROOT($\mathcal{V}$)

1. **for** $e \in \mathcal{N}(f)$
   COLLECT(f, e)

$MAP = \max_{x_f}(\psi^E(x_f) \prod_{e \in \mathcal{N}(f)} m_{ef}^{\max}(x_f))$ // compute MAP probability at root

$x_f^* = \arg\max_{x_f}(\psi^E(x_f) \prod_{e \in \mathcal{N}(f)} m_{ef}^{\max}(x_f))$ // get MAP configuration at root

2. **for** $e \in \mathcal{N}(f)$
   DISTRIBUTE(f, e)

1. COLLECT(i, j) // inward message passing
   for $k \in \mathcal{N}(j) \backslash i$
   COLLECT(j, k)
   SENDMESSAGE(j, i)

2. DISTRIBUTE(i, j) // outward message passing
   SETVALUE(i, j)
   for $k \in \mathcal{N}(j) \backslash i$
   DISTRIBUTE(j, k)

SENDMESSAGE(j, i)

$m_{ji}^{\max}(x_i) = \max_{x_j}(\psi^E(x_j)\psi(x_i, x_j) \prod_{k \in \mathcal{N}(j) \backslash i} m_{kj}^{\max}(x_j))$ // compute MAP probability message

$\delta_{ji}(x_i) \in \arg\max_{x_j}(\psi^E(x_j)\psi(x_i, x_j) \prod_{k \in \mathcal{N}(j) \backslash i} m_{kj}^{\max}(x_j))$ // get MAP configurations

SETVALUE(i, j) // get MAP configuration in outward pass

$x_j^* = \delta_{ji}(x_i^*)$

L5: Max-Product Algorithm for Trees

Example: $x_i \in \{0,1\}$

Inward message passing

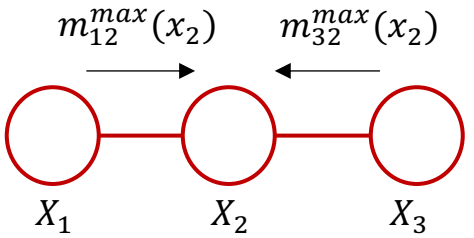
X_2	$m_{12}^{max}(x_2)$	$\delta_{12}(x_2)$
0	$\max_{x_1} \psi(x_1)\psi(x_1, x_2 = 0)$ $= \max(\psi(x_1 = 0)\psi(x_1 = 0, x_2 = 0), \psi(x_1 = 1)\psi(x_1 = 1, x_2 = 0))$ $= \max(a_1, a_2) = a_1$	$x_1^{max} = 0$
1	$\max_{x_1} \psi(x_1)\psi(x_1, x_2 = 1)$ $= \max(\psi(x_1 = 0)\psi(x_1 = 0, x_2 = 1), \psi(x_1 = 1)\psi(x_1 = 1, x_2 = 1))$ $= \max(a_3, a_4) = a_4$	$x_1^{max} = 1$

*In this example, we assume $a_1 > a_2$ and $a_4 > a_3$.

X_2	$m_{32}^{max}(x_2)$	$\delta_{32}(x_2)$
0	$\max_{x_3} \psi(x_3)\psi(x_3, x_2 = 0)$ $= \max(\psi(x_3 = 0)\psi(x_3 = 0, x_2 = 0), \psi(x_3 = 1)\psi(x_3 = 1, x_2 = 0))$ $= \max(b_1, b_3) = b_3$	$x_3^{max} = 1$
1	$\max_{x_3} \psi(x_3)\psi(x_3, x_2 = 1)$ $= \max(\psi(x_3 = 0)\psi(x_3 = 0, x_2 = 1), \psi(x_3 = 1)\psi(x_3 = 1, x_2 = 1))$ $= \max(b_2, b_4) = b_2$	$x_3^{max} = 0$

*In this example, we assume $b_3 > b_1$ and $b_2 > b_4$.

Find: $\underset{x_1,x_2,x_3}{\operatorname{argmax}}\, p(x_1, x_2, x_3)$



Root Node

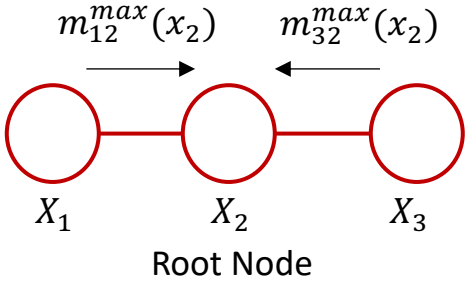
L5: Max-Product Algorithm for Trees

Example: $x_i \in \{0,1\}$
Root node

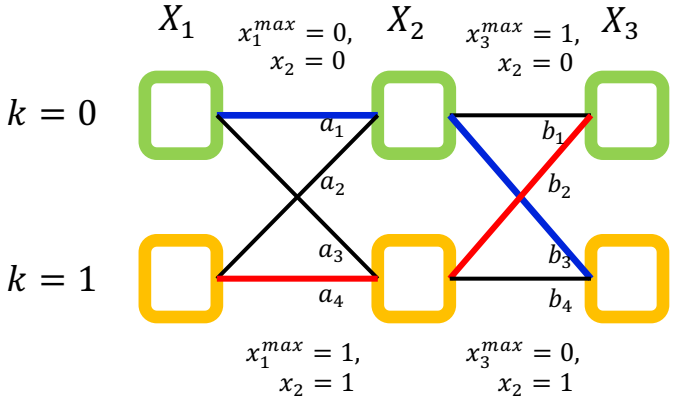
$m_2^{max}(x_2)$	$\delta_2(\cdot)$
$\max_{x_2} \psi(x_2) m_{12}^{max}(x_2) m_{32}^{max}(x_2)$ $= \max(\psi(x_2=0)a_1 b_3, \psi(x_2=1)a_4 b_2)$ $= \max(d_1, d_2) = d_1$ and d_2	$x_2^{max} = 0$ or 1

*In this example, we assume $d_1 = d_2$.

Find: $\underset{x_1,x_2,x_3}{\operatorname{argmax}}\, p(x_1, x_2, x_3)$



Trellis Diagram:



Downward pass

$\delta_{12}(x_2): x_1^{max} = 0 \leftarrow \delta_2(x_2): x_2^{max} = 0 \rightarrow \delta_{32}(x_2): x_3^{max} = 1$

$\delta_{12}(x_2): x_1^{max} = 1 \leftarrow \delta_2(x_2): x_2^{max} = 1 \rightarrow \delta_{32}(x_2): x_3^{max} = 0$

Recall from Lecture 5: Underflow problem

- Products of probabilities (numbers between 0 and 1) tend to underflow.
- Can be overcome by transforming to the monotone log scale:

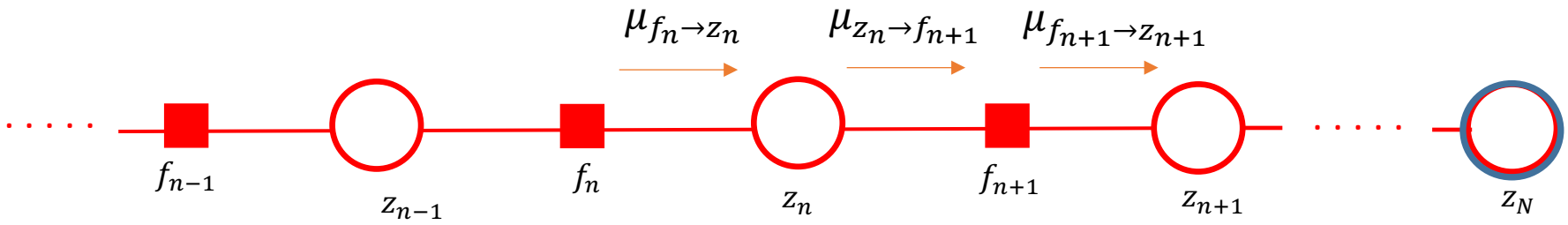
$$\max_x p^E(x) = \max_x \log p^E(x)$$

- Fortunately, the distributive law still holds:

$$\max(a + b_1, a + b_2, \ldots, a + b_n) = a + \max(b_1, b_2, \ldots, b_n)$$

- Turns the "max-product" algorithm into the "max-sum" algorithm.

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:

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

$K \times 1$ entries

which can be computed recursively

Proof:

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

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

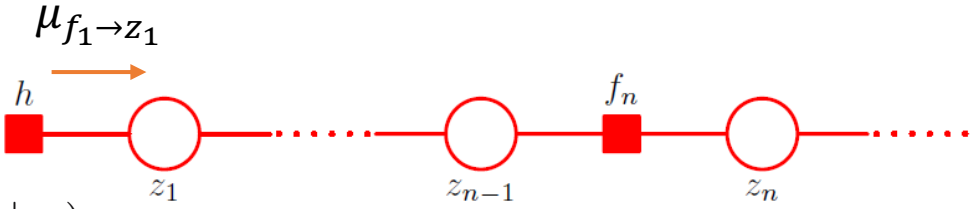
$$\Longrightarrow \quad \omega(z_{n+1}) = \ln p(x_{n+1}|z_{n+1}) + \max_{z_n}\{\ln p(z_{n+1}|z_n) + \omega(z_n)\}$$

The Viterbi Algorithm

$K \times 1$ entries

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

- Note that no need for scaling since the Viterbi algorithm works with log probabilities.
- **Initialization**: factor-to-node message



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

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

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

The Viterbi Algorithm

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

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

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 5 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

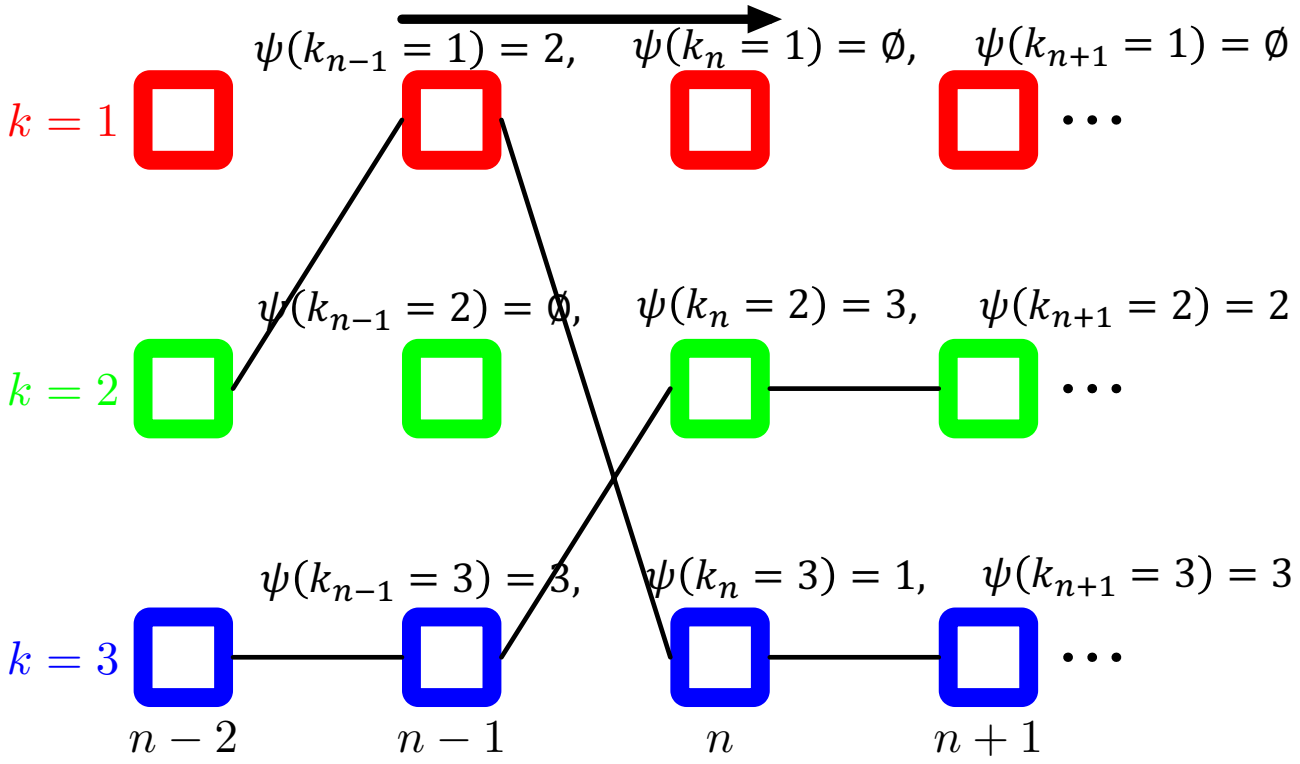


Image source: "Pattern recognition and machine learning", Christopher Bishop

Key ideas

- How to compute a predictive distribution given some observed data?
- What is the most likely sequence of hidden states given the observed data?
  - Viterbi algorithm
  - Max-product / Max-sum algorithm

Recap and Extensions

HMMs and EM

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 $\{Z_1, \ldots Z_{n-1}, Z_n \ldots\}$ form a Markov Chain, where the current state Z_n is dependent on the previous state Z_{n-1}.

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:

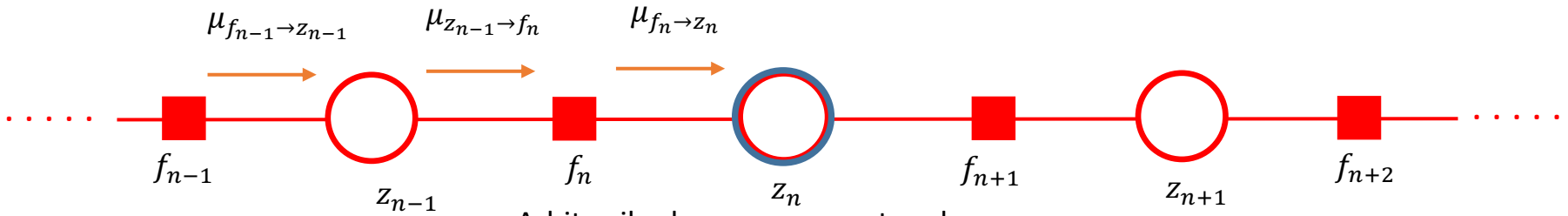
$$\gamma(\mathbf{z}_n) \quad = \quad p(\mathbf{z}_n|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})$$

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

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

HMM: Sum-of-Product Algorithm

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



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

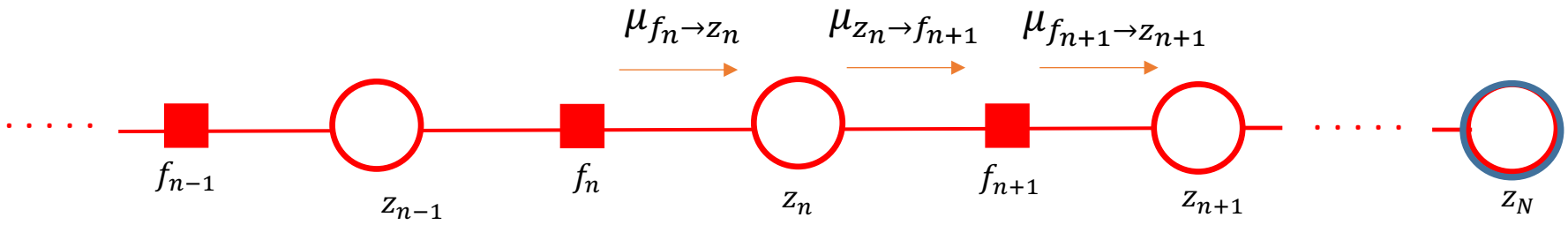
$$\mu_{f_n\to \mathbf{z}_n}(\mathbf{z}_n) = \sum_{\mathbf{z}_{n-1}} f_n(\mathbf{z}_{n-1}, \mathbf{z}_n)\mu_{f_{n-1}\to \mathbf{z}_{n-1}}(\mathbf{z}_{n-1})$$

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.

The Viterbi Algorithm



Choose Z_N as root node

- Messages in the max-sum algorithm:

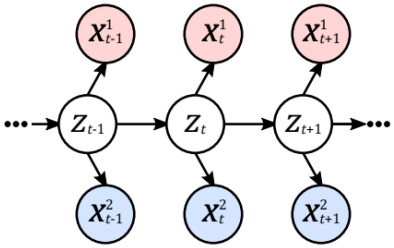
Node-to-Factor: $$\mu_{\mathbf{z}_n \to f_{n+1}}(\mathbf{z}_n) = \mu_{f_n \to \mathbf{z}_n}(\mathbf{z}_n)$$

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

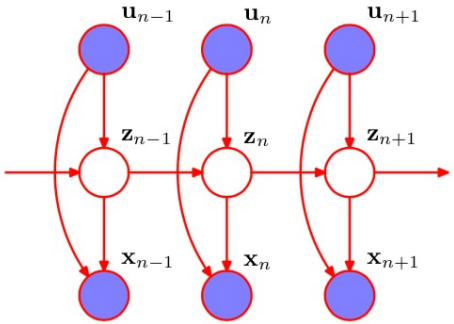
- Similar to forward recursion, except summation of Z_n is replaced with max of Z_n.
- No backward passing since the max probability is the same regardless of the choice of root node.

Extensions & Further Exploration

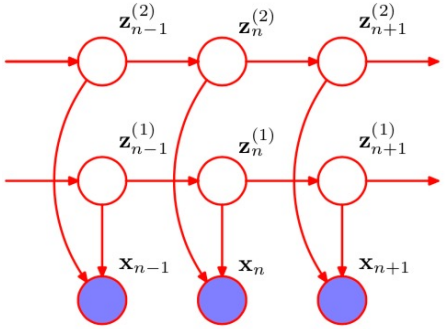
- Linear Dynamical Systems
  - ***Factorized Inference in Deep Markov Models for Incomplete Multimodal Time Series***
    Zhi-Xuan Tan, Harold Soh, Desmond Ong, AAAI 2020



- Modifications of the HMM model.



Input-Output HMM



Factorial HMM