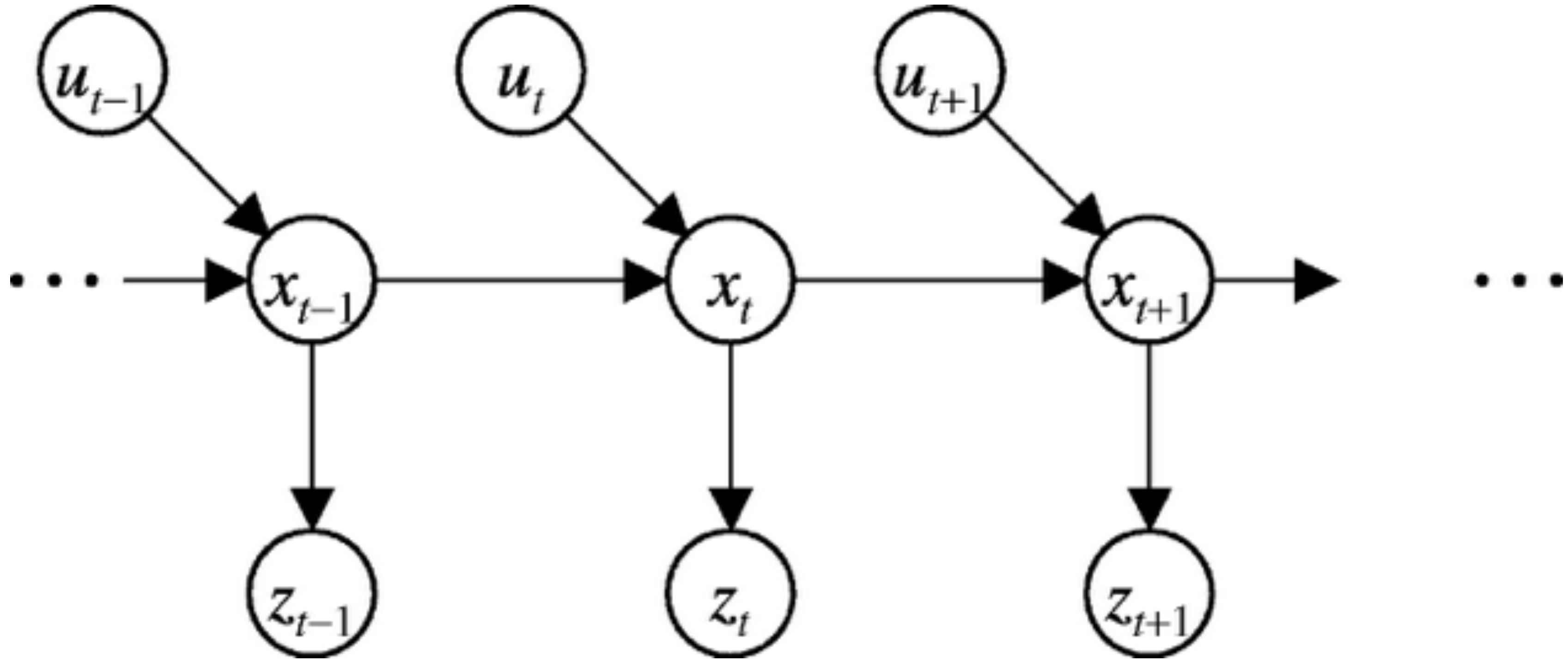
7. Sequential Data

Bayes Filter (Rep.)

We can describe the overall process using a *Dynamic Bayes Network*:



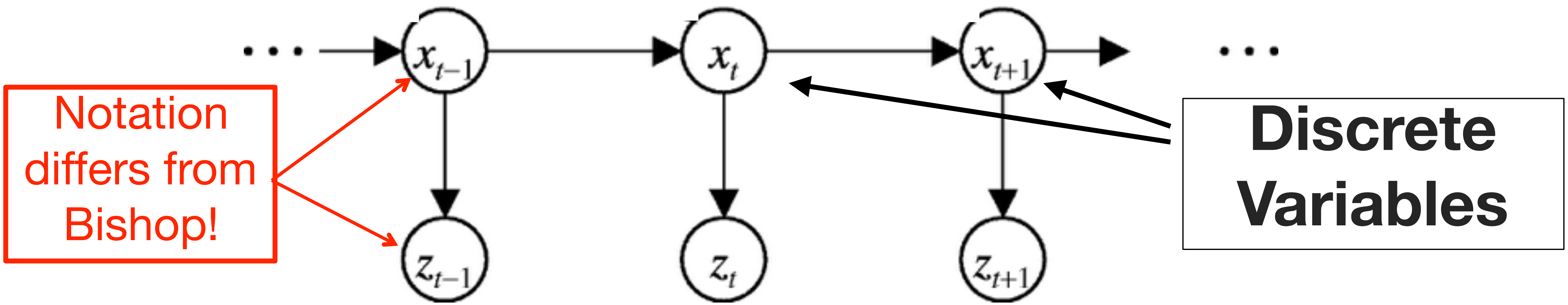
- This incorporates the following Markov assumptions:

$$p(z_t \mid x_{0:t}, u_{1:t}, z_{1:t}) = p(z_t \mid x_t) \quad \text{(measurement)}$$

$$p(x_t \mid x_{0:t-1}, u_{1:t}, z_{1:t}) = p(x_t \mid x_{t-1}, u_t) \quad \text{(state)}$$

Bayes Filter Without Actions

Removing the action variables we obtain:



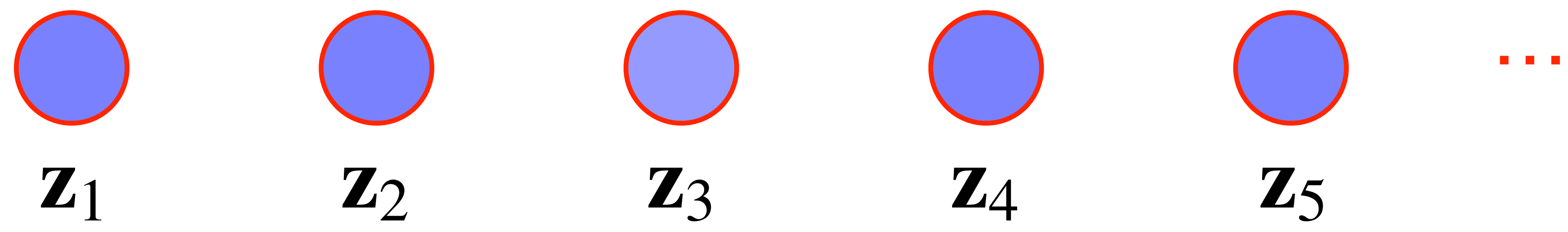
- This incorporates the following Markov assumptions:

$$p(z_t \mid x_{0:t}, \quad z_{1:t}) = p(z_t \mid x_t) \quad \text{(measurement)}$$

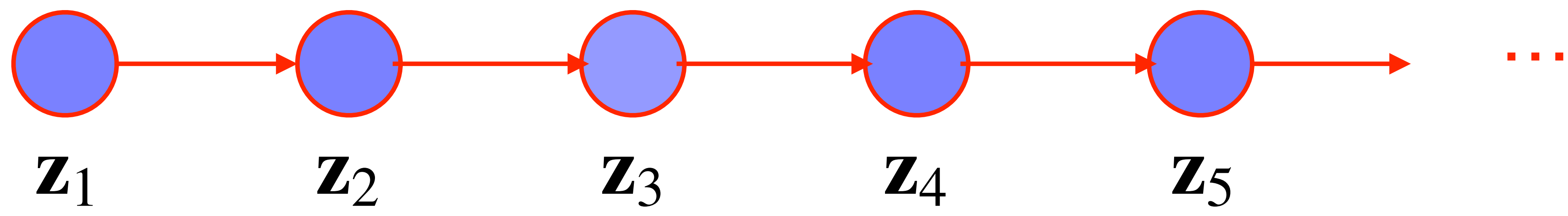
$$p(x_t \mid x_{0:t-1}, \quad z_{1:t}) = p(x_t \mid x_{t-1} \quad) \quad \text{(state)}$$

A Model for Sequential Data

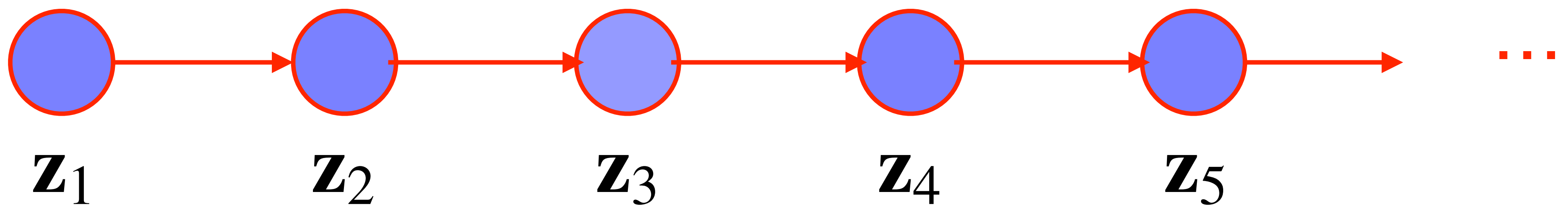
- Observations in sequential data should not be modeled as independent variables such as:



- Examples: weather forecast, speech, hand-written text, etc.
- The observation at time t depends on the observation(s) of (an) earlier time step(s):



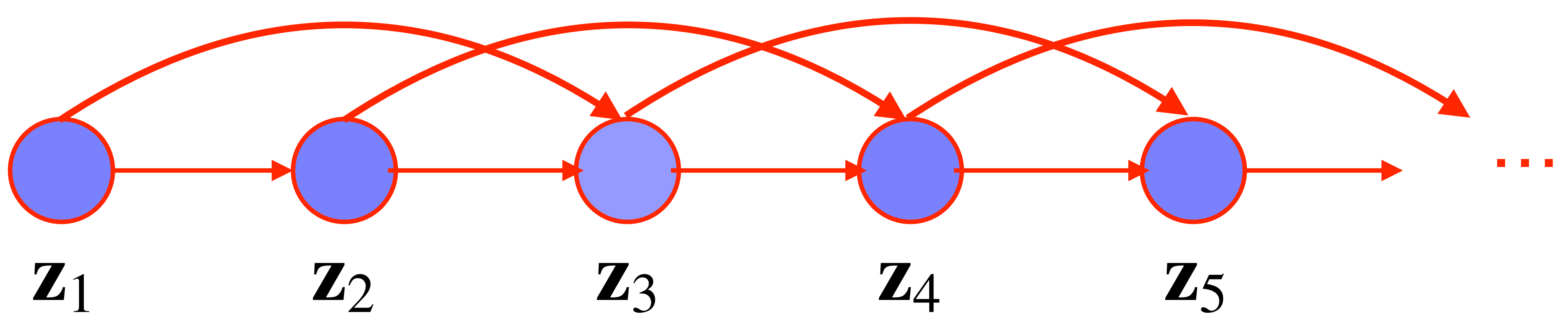
A Model for Sequential Data



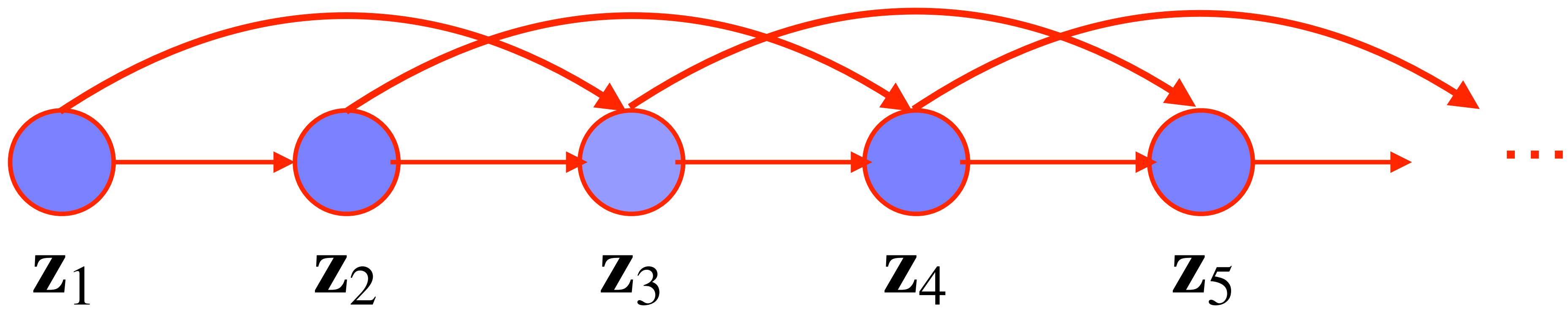
- The joint distribution is therefore (d-sep):

$$p(\mathbf{z}_1 \dots \mathbf{z}_n) = p(\mathbf{z}_1) \prod_{i=2}^{n} p(\mathbf{z}_i \mid \mathbf{z}_{i-1})$$

- **However:** often data depends on several earlier observations (not just one)



A Model for Sequential Data

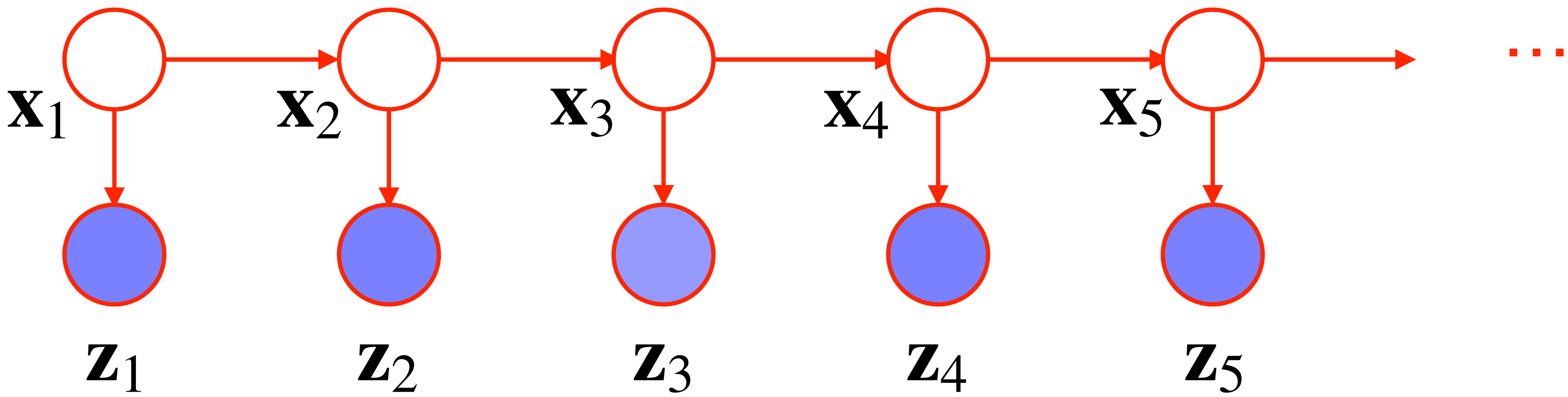


$$p(\mathbf{z}_1 \ldots \mathbf{z}_n) = p(\mathbf{z}_1)p(\mathbf{z}_2 \mid \mathbf{z}_1) \prod_{i=3}^{n} p(\mathbf{z}_i \mid \mathbf{z}_{i-1}, \mathbf{z}_{i-2})$$

- **Problem:** number of stored parameters grows exponentially with the **order** of the Markov chain
- **Question:** can we model dependency of all previous observations with a limited number of parameters?

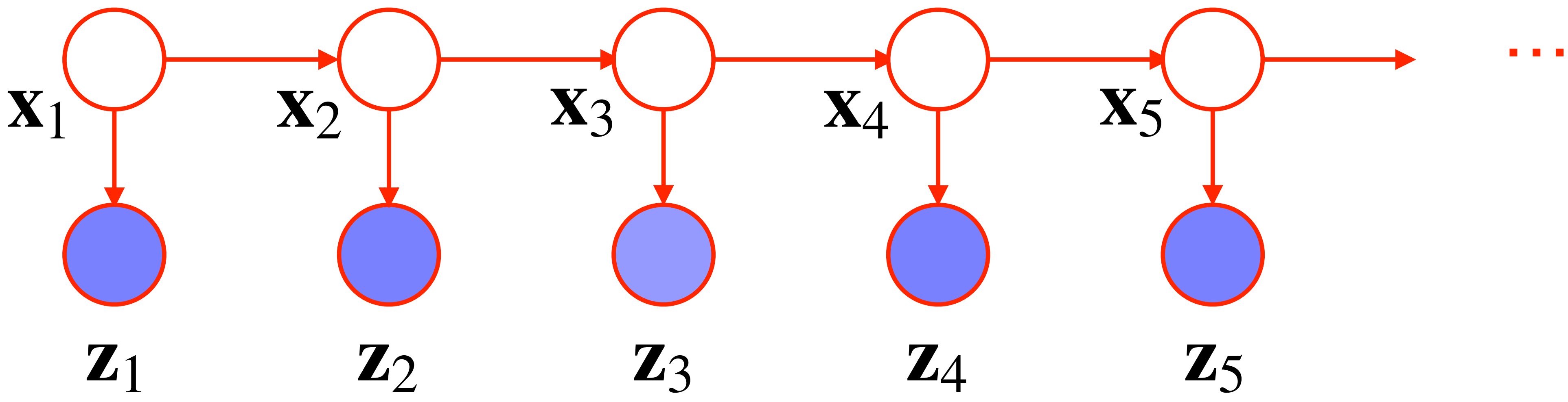
A Model for Sequential Data

Idea: Introduce **hidden** (unobserved) variables:



A Model for Sequential Data

Idea: Introduce **hidden** (unobserved) variables:



Now we have: $\text{dsep}(\mathbf{x}_n, \{\mathbf{x}_1, \dots, \mathbf{x}_{n-2}\}, \mathbf{x}_{n-1})$

$$\Leftrightarrow p(\mathbf{x}_n \mid \mathbf{x}_1, \dots, \mathbf{x}_{n-2}, \mathbf{x}_{n-1}) = p(\mathbf{x}_n \mid \mathbf{x}_{n-1})$$

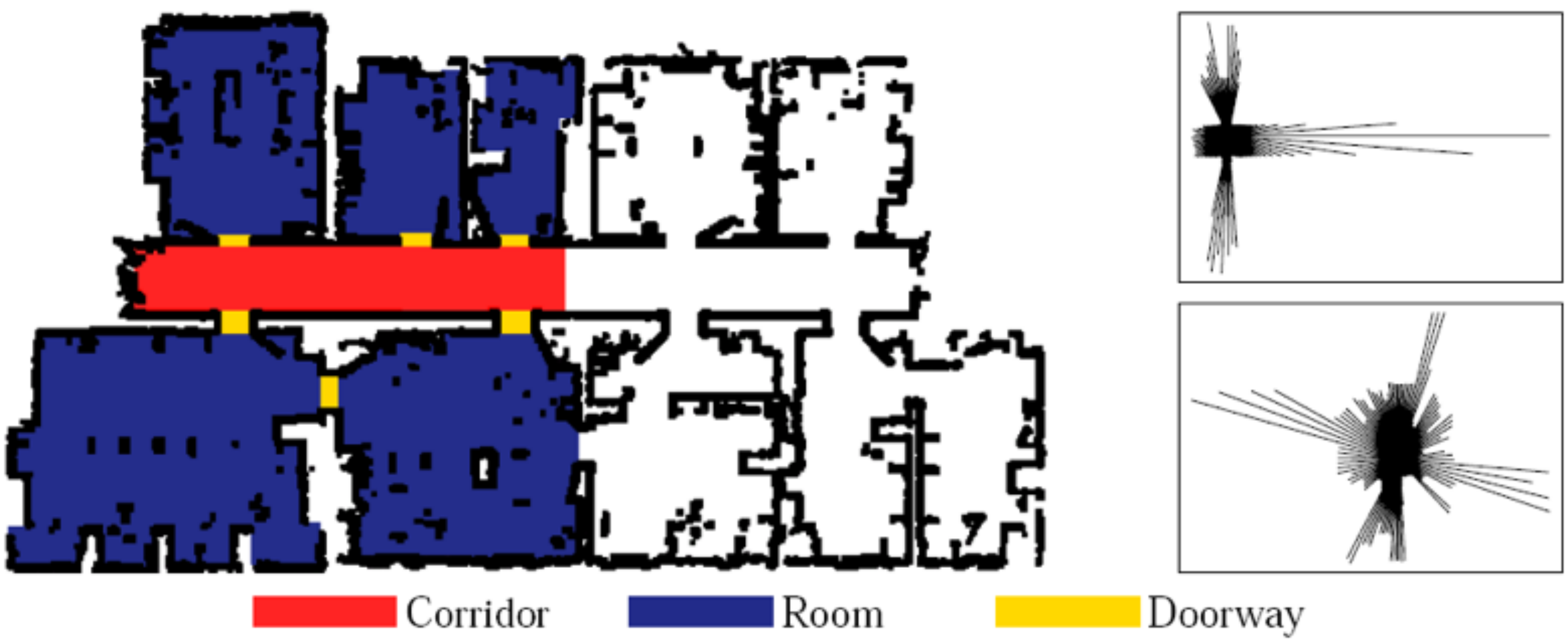
But: $\neg\text{dsep}(\mathbf{z}_n, \{\mathbf{z}_1, \dots, \mathbf{z}_{n-2}\}, \mathbf{z}_{n-1})$

$$\Leftrightarrow p(\mathbf{z}_n \mid \mathbf{z}_1, \dots, \mathbf{z}_{n-2}, \mathbf{z}_{n-1}) \neq p(\mathbf{z}_n \mid \mathbf{z}_{n-1})$$

And: number of parameters is $nK(K-1) + const.$

Example

- Place recognition for mobile robots
- 3 different states: corridor, room, doorway
- Problem: misclassifications
- Idea: use information from previous time step

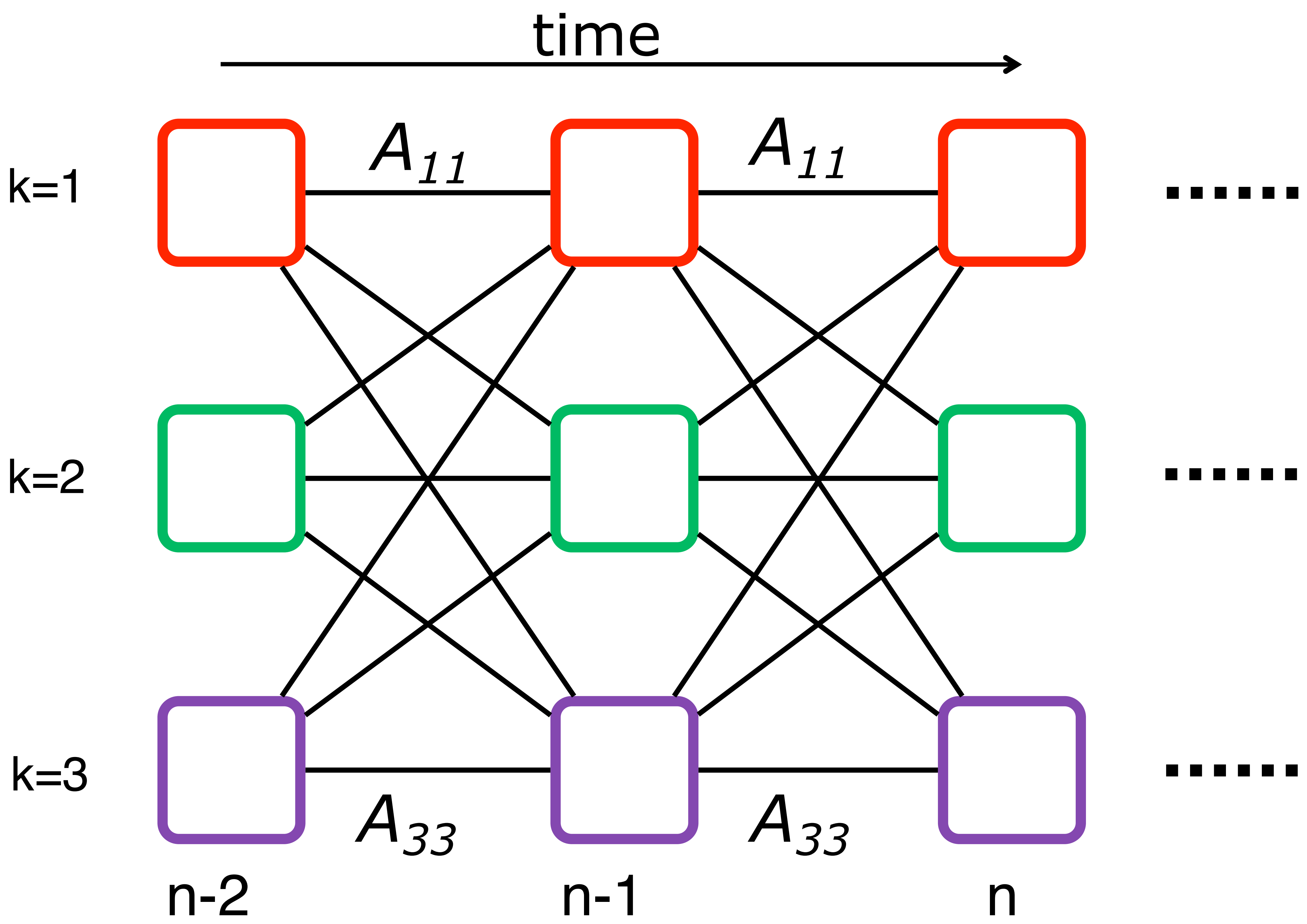


General Formulation of an HMM

1. Discrete random variables
   - **Observation** variables: $\{z_n\}, n = 1..N$
   - Discrete **state** variables (unobservable): $\{x_n\}, n = 1..N$
   - **Number** of states K: $x_n \epsilon \{1 \ldots K\}$
2. Transition model $p(x_i \mid x_{i-1})$
   - Markov assumption (x_i only depends on x_{i-1})
   - Represented as a $K \times K$ **transition matrix** A
   - Initial probability: $p(x_0)$ repr. as π_1, π_2, π_3
3. Observation model $p(z_i \mid x_i)$ with parameters φ
   - Observation only depends on the current state
   - Example: output of a "local" place classifier

Model Parameters
θ

The Trellis Representation



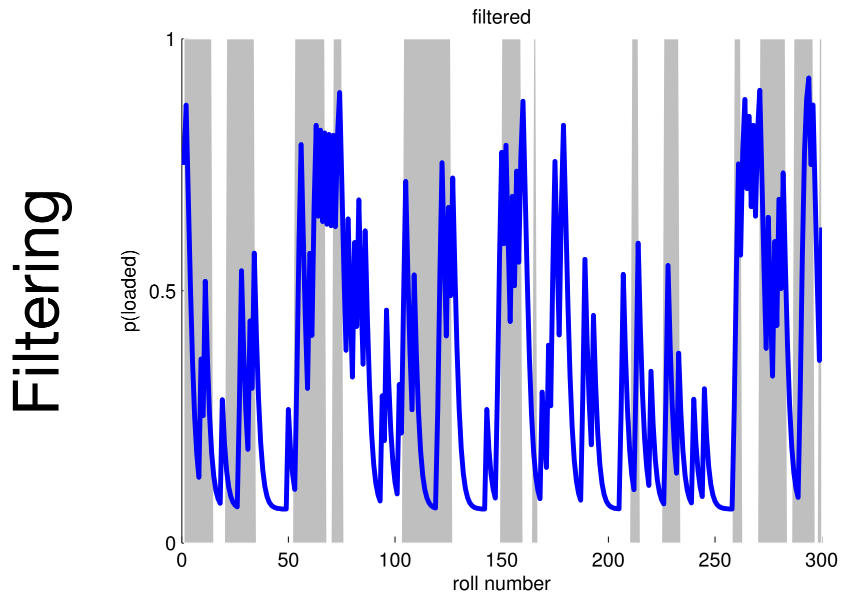
Application Example (1)

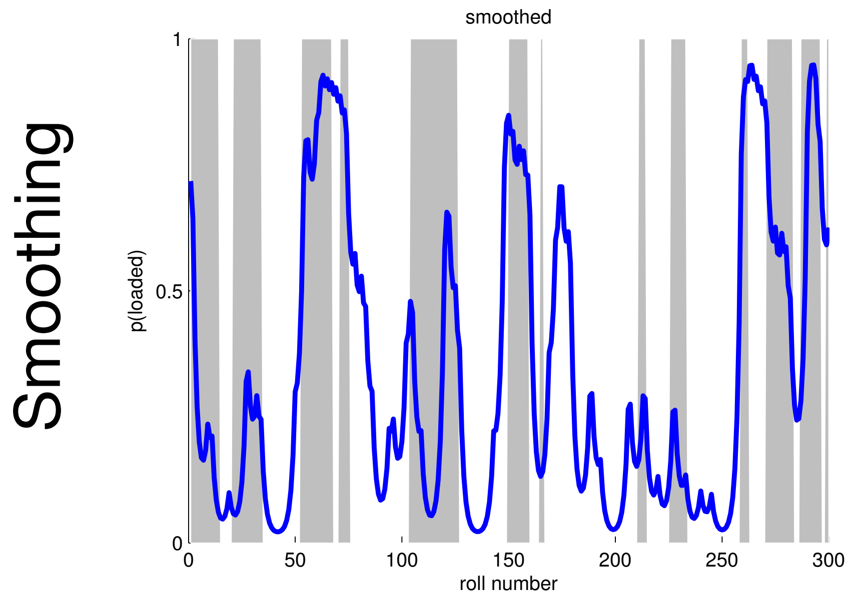
- Given an observation sequence $\mathbf{z}_1,\mathbf{z}_2,\mathbf{z}_3\ldots$
- Assume that the model parameters $\theta = (A, \pi, \varphi)$ are known
- What is the probability that the given observation sequence is actually observed under this model, i.e. the **data likelihood** $p(Z|\ \theta)$?
- If we are given several different models, we can choose the one with highest probability
- Expressed as a **supervised learning problem**, this can be interpreted as the inference step (classification step)

Application Example (2)

Based on the data likelihood we can solve two different kinds of problems:

- **Filtering:** computes $p(\mathbf{x}_n \mid \mathbf{z}_{1:n})$, i.e. state probability only based on previous observations
- **Smoothing:** computes $p(\mathbf{x}_n \mid \mathbf{z}_{1:N})$, state probability based on **all** observations (including those from the future)





Application Example (3)

- Given an observation sequence $\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3 \ldots$
- Assume that the model parameters $\theta = (A, \pi, \varphi)$ are known
- What is the state sequence $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \ldots$ that **explains best** the given observation sequence?
- In the case of place recognition: which is the sequence of **truly visited places** that explains best the sequence of obtained place labels (classifications)?

Application Example (4)

- Given an observation sequence $\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3 \ldots$
- What are the optimal model parameters $\theta = (A, \pi, \varphi)$?
- This can be interpreted as the **training step**
- It is in general the most difficult problem

Summary: 4 Operations on HMMs

1. Compute data likelihood $p(Z|\theta)$ from a known model
   - Can be computed with the **forward** algorithm
2. Filtering or Smoothing of the state probability
   - Filtering: **forward** algorithm
   - Smoothing: **forward-backward** algorithm
3. Compute optimal state sequence with a known model
   - Can be computed with the **Viterbi**-Algorithm
4. Learn model parameters for an observation sequence
   - Can be computed using **Expectation-Maximization** (or Baum-Welch)

The Forward Algorithm

Goal: compute $p(Z|\theta)$ (we drop θ in the following)

$$p(\mathbf{z}_1, \ldots, \mathbf{z}_n) = \sum_{\mathbf{x}_n} p(\mathbf{z}_1, \ldots, \mathbf{z}_n, \mathbf{x}_n) =: \sum_{\mathbf{x}_n} \alpha(\mathbf{x}_n)$$

The Forward Algorithm

Goal: compute $p(Z|\theta)$ (we drop θ in the following)

$$p(\mathbf{z}_1, \dots, \mathbf{z}_n) = \sum_{\mathbf{x}_n} p(\mathbf{z}_1, \dots, \mathbf{z}_n, \mathbf{x}_n) =: \sum_{\mathbf{x}_n} \alpha(\mathbf{x}_n)$$

We can calculate α recursively:

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

The Forward Algorithm

Goal: compute $p(Z|\theta)$ (we drop θ in the following)

$$p(\mathbf{z}_1, \ldots, \mathbf{z}_n) = \sum_{\mathbf{x}_n} p(\mathbf{z}_1, \ldots, \mathbf{z}_n, \mathbf{x}_n) =: \sum_{\mathbf{x}_n} \alpha(\mathbf{x}_n)$$

We can calculate α recursively:

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

This is (almost) the same recursive formula as we had in the first lecture!

The Forward Algorithm

Goal: compute $p(Z|\theta)$ (we drop θ in the following)

$$p(\mathbf{z}_1, \ldots, \mathbf{z}_n) = \sum_{\mathbf{x}_n} p(\mathbf{z}_1, \ldots, \mathbf{z}_n, \mathbf{x}_n) =: \sum_{\mathbf{x}_n} \alpha(\mathbf{x}_n)$$

We can calculate α recursively:

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

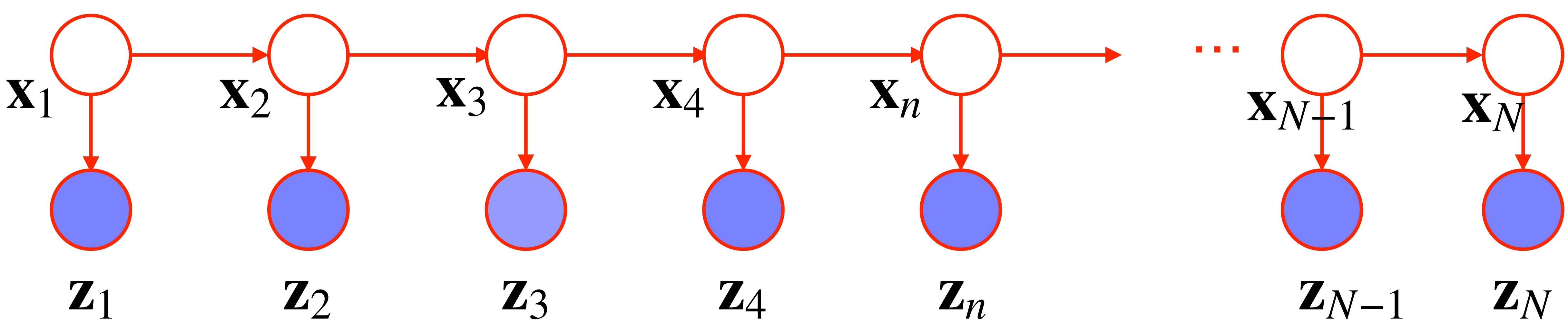
This is (almost) the same recursive formula as we had in the first lecture!

Filtering: $$p(\mathbf{x}_n \mid \mathbf{z}_1, \ldots, \mathbf{z}_n) = \frac{p(\mathbf{z}_1, \ldots, \mathbf{z}_n, \mathbf{x}_n)}{p(\mathbf{z}_1, \ldots, \mathbf{z}_n)} = \frac{\alpha(\mathbf{x}_n)}{\sum_{\mathbf{x}_n} \alpha(\mathbf{x}_n)}$$

The Forward-Backward Algorithm

- As before we set $\alpha(\mathbf{x}_n) = p(\mathbf{z}_1, \ldots, \mathbf{z}_n, \mathbf{x}_n)$
- We also define $\beta(\mathbf{x}_n) = p(\mathbf{z}_{n+1}, \ldots, \mathbf{z}_N \mid \mathbf{x}_n)$

e.g. $n = 5$:



The Forward-Backward Algorithm

- As before we set $\alpha(\mathbf{x}_n) = p(\mathbf{z}_1, \dots, \mathbf{z}_n, \mathbf{x}_n)$
- We also define $\beta(\mathbf{x}_n) = p(\mathbf{z}_{n+1}, \dots, \mathbf{z}_N \mid \mathbf{x}_n)$
- This can be recursively computed (backwards):

$$\beta(\mathbf{x}_{n-1}) = p(\mathbf{z}_n, \dots, \mathbf{z}_N \mid \mathbf{x}_{n-1})$$

$$= \sum_{\mathbf{x}_n} p(\mathbf{x}_n, \mathbf{z}_n, \dots, \mathbf{z}_N \mid \mathbf{x}_{n-1})$$

$$= \sum_{\mathbf{x}_n} p(\mathbf{z}_{n+1}, \dots, \mathbf{z}_N \mid \mathbf{x}_n, \cancel{\mathbf{z}_n}, \cancel{\mathbf{x}_{n-1}}) p(\mathbf{x}_n, \mathbf{z}_n \mid \mathbf{x}_{n-1})$$

$$= \sum_{\mathbf{x}_n} p(\mathbf{z}_{n+1}, \dots, \mathbf{z}_N \mid \mathbf{x}_n) p(\mathbf{z}_n \mid \cancel{\mathbf{x}_{n-1}}, \mathbf{x}_n) p(\mathbf{x}_n \mid \mathbf{x}_{n-1})$$

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

The Forward-Backward Algorithm

- As before we set $\alpha(\mathbf{x}_n) = p(\mathbf{z}_1, \ldots, \mathbf{z}_n, \mathbf{x}_n)$
- We also define $\beta(\mathbf{x}_n) = p(\mathbf{z}_{n+1}, \ldots, \mathbf{z}_N \mid \mathbf{x}_n)$
- This can be recursively computed (backwards):

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

- This is also known as the **message-passing** algorithm ("sum-product")!
    - forward messages α_n (vector of length K)
    - backward messages β_n (vector of length K)

Smoothing with Forward-Backward

First we compute $p(\mathbf{x}_n, \mathbf{z}_1, \dots, \mathbf{z}_N)$:

$$p(\mathbf{x}_n, \mathbf{z}_1, \dots, \mathbf{z}_N) = p(\mathbf{z}_1, \dots, \mathbf{z}_N \mid \mathbf{x}_n) p(\mathbf{x}_n)$$

$$= p(\mathbf{z}_1, \dots, \mathbf{z}_n \mid \mathbf{x}_n) p(\mathbf{z}_{n+1}, \dots, \mathbf{z}_N \mid \mathbf{x}_n) p(\mathbf{x}_n)$$

$$= p(\mathbf{z}_1, \dots, \mathbf{z}_n, \mathbf{x}_n) p(\mathbf{z}_{n+1}, \dots, \mathbf{z}_N \mid \mathbf{x}_n)$$

$$= \alpha(\mathbf{x}_n) \beta(\mathbf{x}_n)$$

Smoothing with Forward-Backward

First we compute $p(\mathbf{x}_n, \mathbf{z}_1, \dots, \mathbf{z}_N)$:

$$p(\mathbf{x}_n, \mathbf{z}_1, \dots, \mathbf{z}_N) = \alpha(\mathbf{x}_n)\beta(\mathbf{x}_n)$$

with that we can compute $p(\mathbf{z}_1, \dots, \mathbf{z}_N)$:

$$p(\mathbf{z}_1, \dots, \mathbf{z}_N) = \sum_{\mathbf{x}_n} p(\mathbf{x}_n, \mathbf{z}_1, \dots, \mathbf{z}_N) = \sum_{\mathbf{x}_n} \alpha(\mathbf{x}_n)\beta(\mathbf{x}_n)$$

Smoothing with Forward-Backward

First we compute $p(\mathbf{x}_n, \mathbf{z}_1, \dots, \mathbf{z}_N)$:

$$p(\mathbf{x}_n, \mathbf{z}_1, \dots, \mathbf{z}_N) = \alpha(\mathbf{x}_n)\beta(\mathbf{x}_n)$$

with that we can compute $p(\mathbf{z}_1, \dots, \mathbf{z}_N)$:

$$p(\mathbf{z}_1, \dots, \mathbf{z}_N) = \sum_{\mathbf{x}_n} p(\mathbf{x}_n, \mathbf{z}_1, \dots, \mathbf{z}_N) = \sum_{\mathbf{x}_n} \alpha(\mathbf{x}_n)\beta(\mathbf{x}_n)$$

and finally:

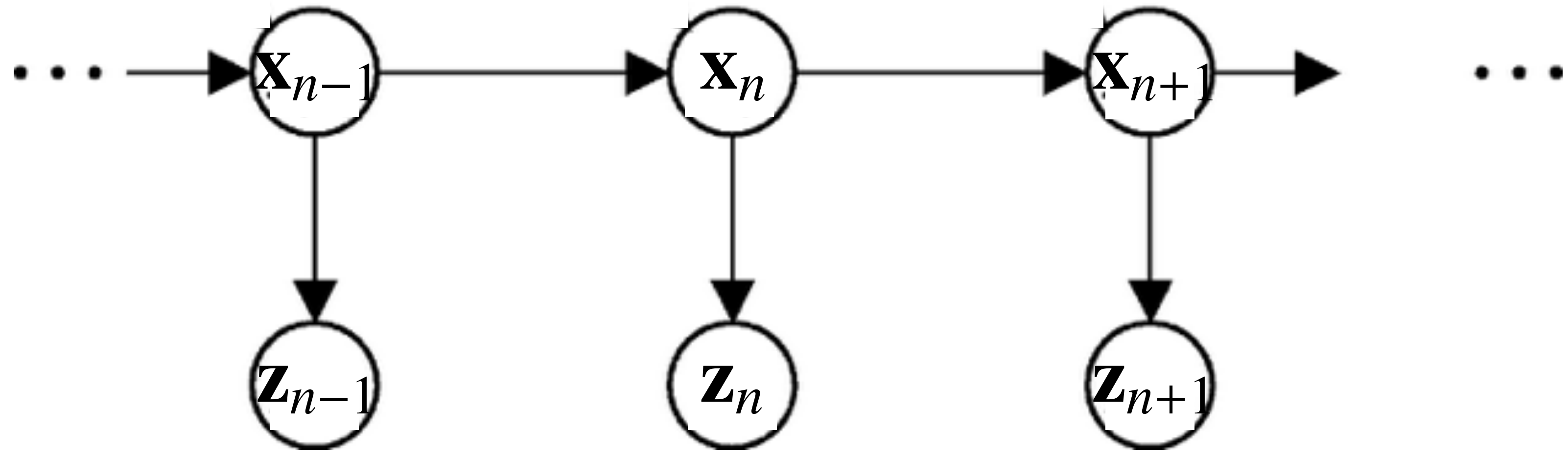
$$p(\mathbf{x}_n \mid \mathbf{z}_1, \dots, \mathbf{z}_N) = \frac{p(\mathbf{x}_n, \mathbf{z}_1, \dots, \mathbf{z}_N)}{p(\mathbf{z}_1, \dots, \mathbf{z}_N)} = \frac{\alpha(\mathbf{x}_n)\beta(\mathbf{x}_n)}{\sum_{\mathbf{x}_n} \alpha(\mathbf{x}_n)\beta(\mathbf{x}_n)}$$

2. Computing the Most Likely States

- Goal: find a state sequence $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \ldots$ that maximizes the probability $p(X,Z|\theta)$

- Define $$\delta(\mathbf{x}_n) = \max_{\mathbf{x}_1,\ldots,\mathbf{x}_{n-1}} p(\mathbf{x}_1, \ldots \mathbf{x}_n \mid \mathbf{z}_1, \ldots \mathbf{z}_n)$$

  This is the probability of state j by taking the most probable path.



2. Computing the Most Likely States

- Goal: find a state sequence $x_1, x_2, x_3 \ldots$ that maximizes the probability $\mathrm{p}(\mathrm{X},\mathrm{Z}|\theta)$

- Define $\delta(\mathbf{x}_n) = \max_{\mathbf{x}_1,\ldots,\mathbf{x}_{n-1}} p(\mathbf{x}_1,\ldots\mathbf{x}_n \mid \mathbf{z}_1,\ldots\mathbf{z}_n)$

  This can be computed recursively:

$$\delta(\mathbf{x}_n) = \max_{\mathbf{x}_{n-1}} \delta(\mathbf{x}_{n-1}) p(\mathbf{x}_n \mid \mathbf{x}_{n-1}) p(\mathbf{z}_n, \mid \mathbf{x}_n)$$

  we also have to compute the argmax:

$$\psi(\mathbf{x}_n) = \arg\max_{\mathbf{x}_{n-1}} \delta(\mathbf{x}_{n-1}) p(\mathbf{x}_n \mid \mathbf{x}_{n-1}) p(\mathbf{z}_n, \mid \mathbf{x}_n)$$

The Viterbi algorithm

- Initialize:
  - $\delta(\mathbf{x}_0) = p(\mathbf{x}_0)\, p(\mathbf{z}_0 \mid \mathbf{x}_0)$
  - $\psi(\mathbf{x}_0) = 0$
- Compute recursively for $n=1\ldots N$:
  - $\delta(\mathbf{x}_n) = p(\mathbf{z}_n|\mathbf{x}_n) \max_{x_{n-1}} [\delta(\mathbf{x}_{n-1})\, p(\mathbf{x}_n|\mathbf{x}_{n-1})]$
  - $\psi(\mathbf{x}_n) = \operatorname{argmax}_{x_{n-1}} [\delta(\mathbf{x}_{n-1})\, p(\mathbf{x}_n|\mathbf{x}_{n-1})]$
- On termination:
  - $p(Z,X|\theta) = \max_{x_N} \delta(x_N)$
  - $x_N^* = \operatorname{argmax}_{x_N} \delta(x_N)$
- Backtracking:
  - $x_n^* = \psi(x_{n+1})$

3. Learning the Model Parameters

- Given an observation sequence $\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3 \ldots$
- Find optimal model parameters $\theta = \pi, A, \varphi$
- We need to maximize the likelihood $p(Z|\theta)$
- Can not be solved in closed form
- Iterative algorithm “Baum-Welch”: a special case of the Expectation Maximization (EM) algorithm

3. Learning the Model Parameters

- Idea: instead of maximizing

$$p(\mathbf{z}_1, \dots, \mathbf{z}_N \mid \theta) = \sum_X p(\mathbf{z}_1, \dots, \mathbf{z}_N, \mathbf{x}_1, \dots, \mathbf{x}_N \mid \theta)$$

- we maximize the **expected** log likelihood:

$$\sum_X p(\mathbf{x}_1, \dots, \mathbf{x}_N \mid \mathbf{z}_1, \dots, \mathbf{z}_N, \theta) \log p(\mathbf{z}_1, \dots, \mathbf{z}_N, \mathbf{x}_1, \dots, \mathbf{x}_N \mid \theta)$$

- it can be shown that this is a lower bound of the actual log-likelihood $p(Z|\theta)$
- this is the general idea of the Expectation-Maximization (EM) algorithm

The Baum-Welsh algorithm

- E-Step (assuming we know π, A, φ, i.e. θ^{old})
- Define the posterior probability of being in state i at step k:
- Define $\gamma(\mathbf{x}_n) = p(\mathbf{x}_n | Z)$

The Baum-Welsh algorithm

- E-Step (assuming we know π, A, φ, i.e. θ^{old})
- Define the posterior probability of being in state i at step k:
- Define $\gamma(\mathbf{x}_n) = p(\mathbf{x}_n | \mathbf{z}_1, \ldots, \mathbf{z}_N)$
- It follows that $\gamma(\mathbf{x}_n) = \alpha(\mathbf{x}_n)\, \beta(\mathbf{x}_n) \,/\, p(Z)$

The Baum-Welsh algorithm

- E-Step (assuming we know π, A, φ, i.e. θ^{old})
- Define the posterior probability of being in state i at step k:
- Define $\gamma(\mathbf{x}_n) = p(\mathbf{x}_n | \mathbf{z}_1, \ldots, \mathbf{z}_n)$
- It follows that $\gamma(\mathbf{x}_n) = \alpha(\mathbf{x}_n)\, \beta(\mathbf{x}_n) \,/\, p(Z)$
- Define $\xi(\mathbf{x}_{n\text{-}1}, \mathbf{x}_n) = p(\mathbf{x}_{n\text{-}1}, \mathbf{x}_n | Z)$
- It follows that

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

The Baum-Welsh algorithm

- Note: $\gamma(\mathbf{x}_n)$ is a vector of length K; each entry $\gamma_k(\mathbf{x}_n)$ represents the probability that the state at time n is equal to $k \in \{1, \ldots K\}$
- Thus: The **expected** number of transitions from state k in the sequence X is

$$\sum_{i=1}^{N} \gamma_k(\mathbf{x}_i)$$

The Baum-Welsh algorithm

- Note: $\gamma(\mathbf{x}_n)$ is a vector of length K; each entry $\gamma_k(\mathbf{x}_n)$ represents the probability that the state at time n is equal to $k \in \{1, \ldots K\}$
- Thus: The **expected** number of transitions from state k in the sequence X is $\sum_{i=1}^{N} \gamma_k(\mathbf{x}_i)$
- Similarly: The expected number of transitions from state j to state k in the sequence X is

$$\sum_{i=1}^{N-1} \xi_{j,k}(\mathbf{x}_i, \mathbf{x}_{i+1})$$

The Baum-Welsh algorithm

- With that we can compute new values for π, A, φ:

$$\pi_k = \gamma_k(\mathbf{x}_1)$$

$$A_{j,k} = \frac{\sum_{i=1}^{N-1} \xi_{j,k}(\mathbf{x}_i, \mathbf{x}_{i+1})}{\sum_{i=1}^{N} \gamma_j(\mathbf{x}_i)} \qquad \varphi_{j,k} = \frac{\sum_{i=1}^{N} \gamma_j(\mathbf{x}_i)\delta_{k,\mathbf{x}_t}}{\sum_{i=1}^{N} \gamma_j(\mathbf{x}_i)}$$

here, we need forward and backward step!

- This is done until the likelihood does not increase anymore (convergence)

The Baum-Welsh Algorithm - Summary

- Start with an initial estimate of $\theta=(\pi,A,\varphi)$ e.g. uniformly and k-means for φ
- Compute messages (E-Step)
- Compute new $\theta=(\pi,A,\varphi)$ (M-step)
- Iterate E and M until convergence
- In each iteration one full application of the forward-backward algorithm is performed
- Result gives a **local** optimum
- For other local optima, the algorithm needs to be started again with new initialization

Summary

- HMMs are a way to model sequential data
- They assume discrete states
- Three possible operations can be performed with HMMs:
  - Data likelihood, given a model and an observation
  - Most likely state sequence, given a model and an observation
  - Optimal Model parameters, given an observation
- Appropriate scaling solves numerical problems
- HMMs are widely used, e.g. in speech recognition

9. Sampling Methods

Sampling Methods

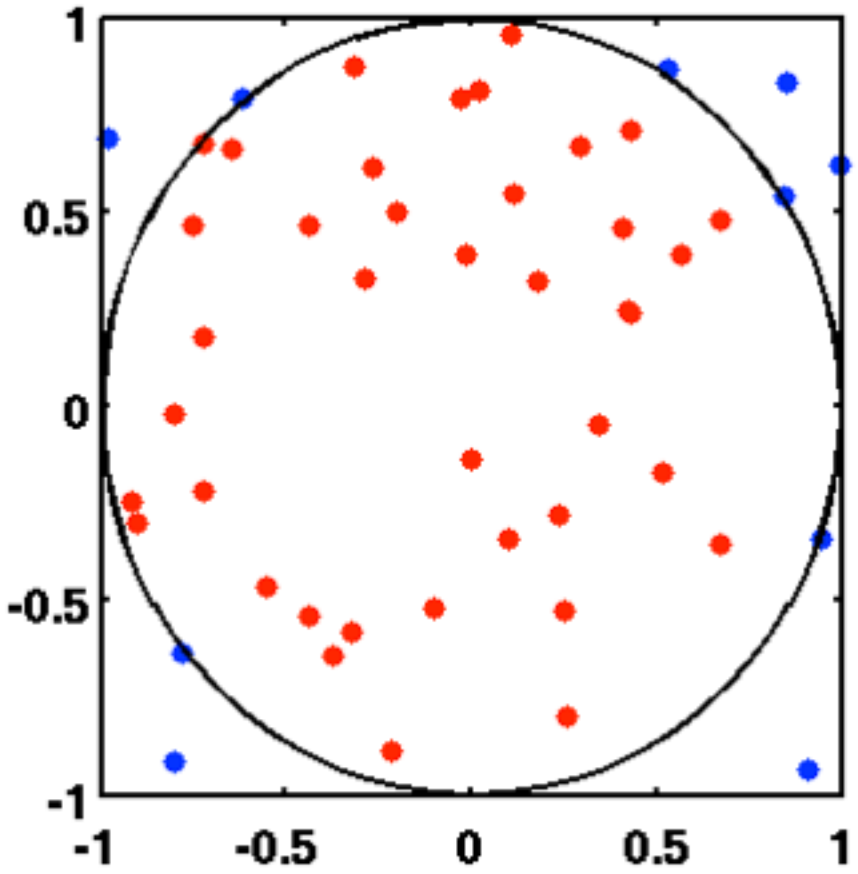
Sampling Methods are widely used in Computer Science

- as an **approximation** of a deterministic algorithm
- to represent **uncertainty** without a parametric model
- to obtain higher computational **efficiency** with a small approximation error

Sampling Methods are also often called **Monte Carlo Methods**

Example: Monte-Carlo Integration

- Sample in the bounding box
- Compute fraction of inliers
- Multiply fraction with box size



Non-Parametric Representation

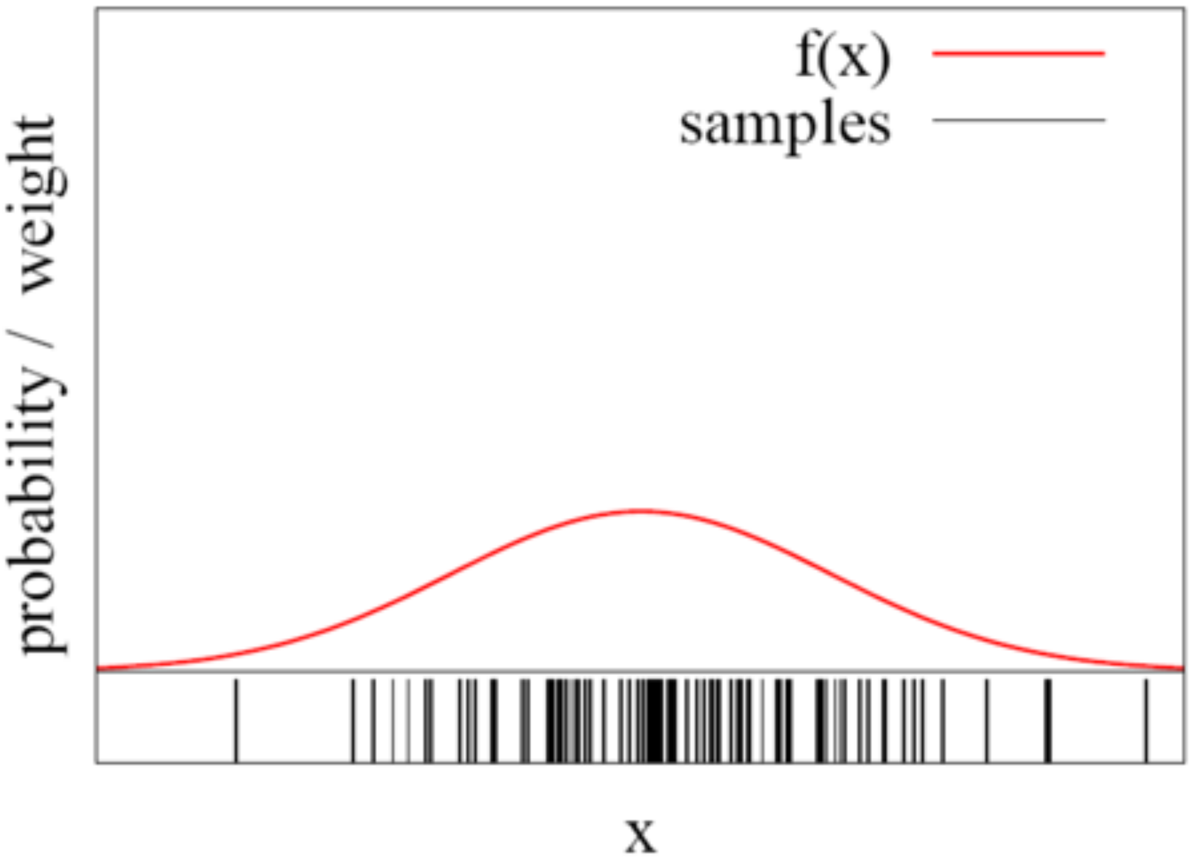
Probability distributions (e.g. a robot‘s belief) can be represeted:

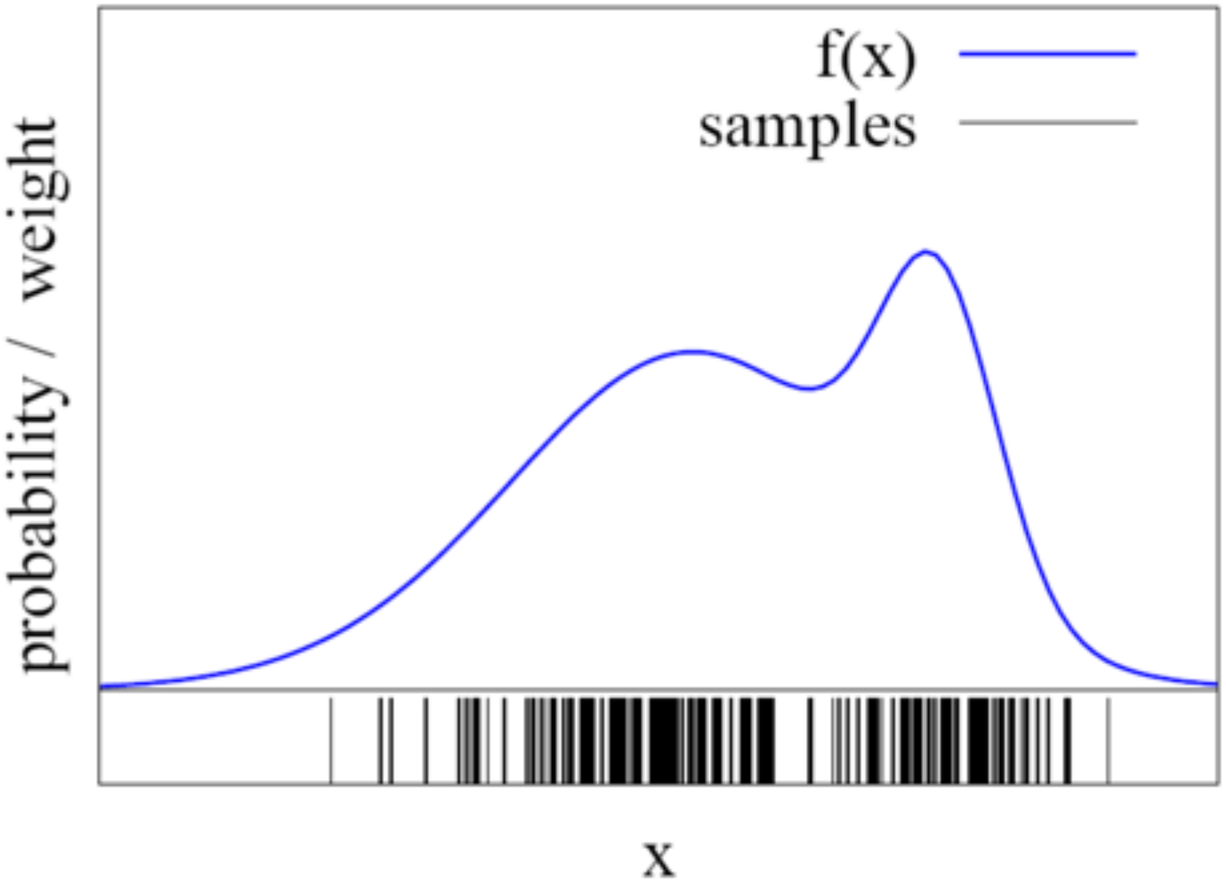
- **Parametrically**: e.g. using mean and covariance of a Gaussian
- **Non-parametrically:** using a set of *hypotheses* (samples) drawn from the distribution

Advantage of non-parametric representation:

- No restriction on the *type* of distribution (e.g. can be multi-modal, non- Gaussian, etc.)

Non-Parametric Representation





The more samples are in an interval, the higher the probability of that interval

But:

How to draw samples from a function/distribution?

Sampling from a Distribution

There are several approaches:

- Probability transformation
  - Uses inverse of the c.d.f (not considered here)
- Rejection Sampling
- Importance Sampling
- Markov Chain Monte Carlo

Rejection Sampling

1. Simplification:

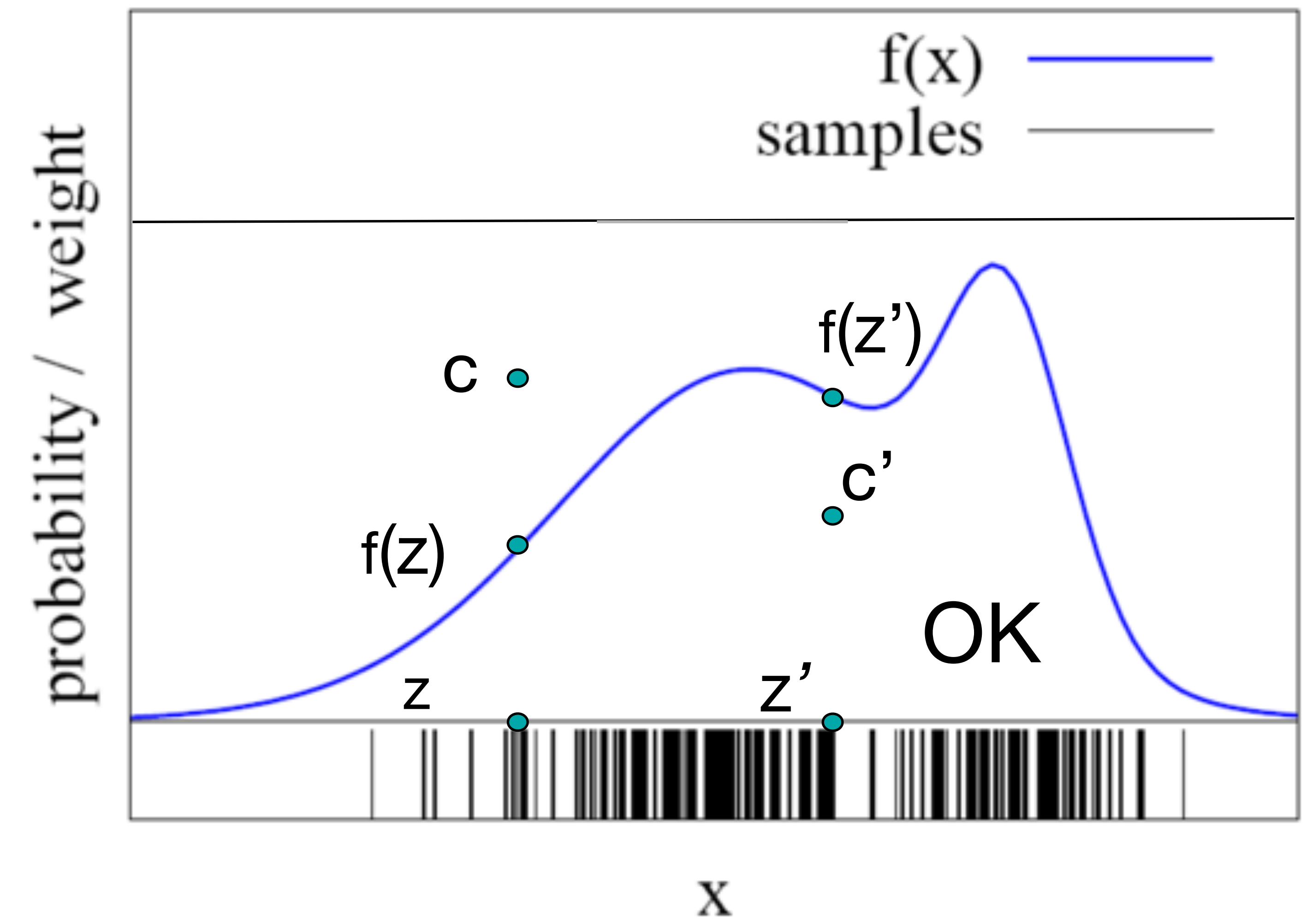
- Assume $p(z) < 1$ for all z
- Sample z uniformly
- Sample c from $[0, 1]$

- **If** $f(z) > c$:

  keep the sample

  otherwise:

  reject the sample

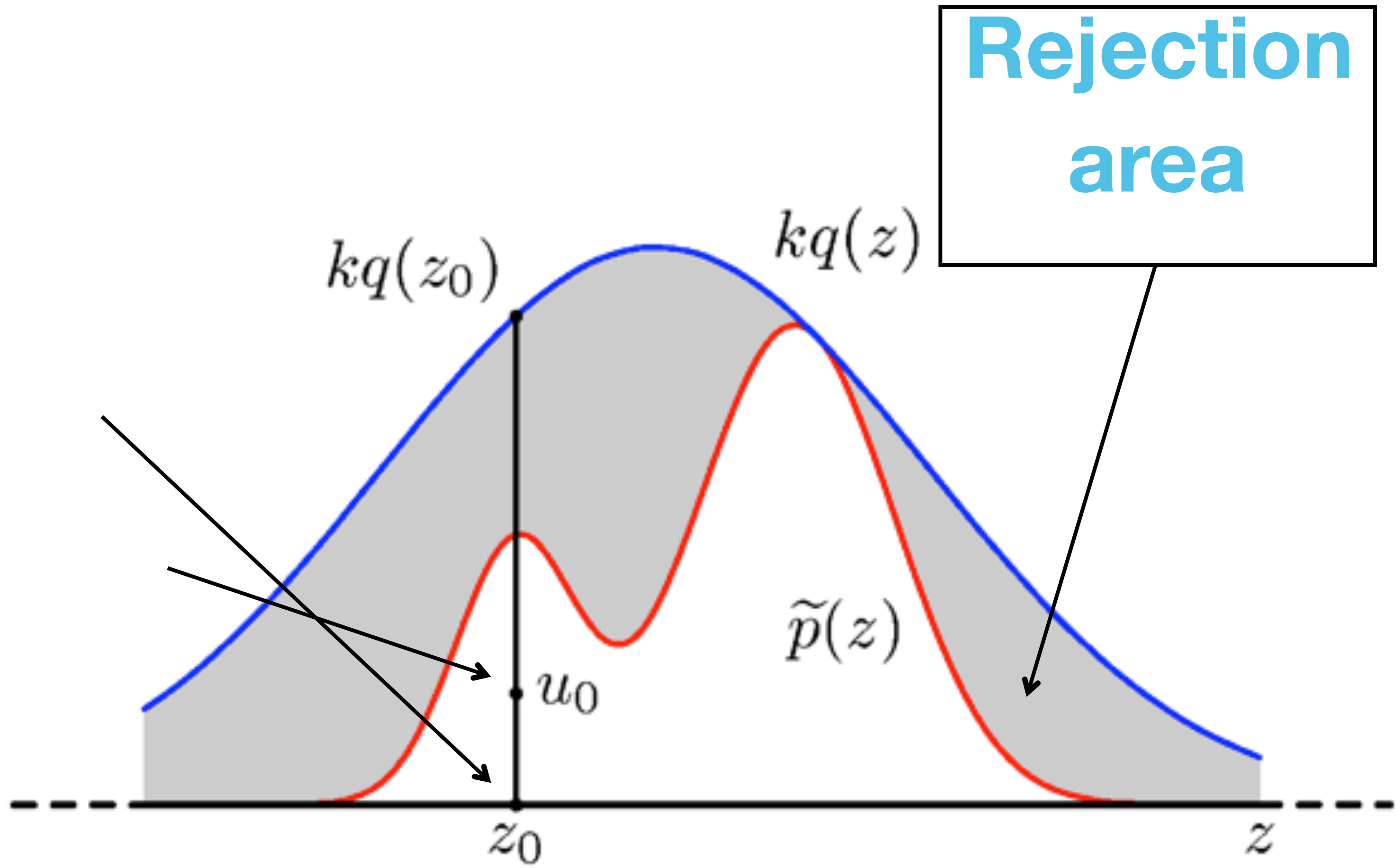


Rejection Sampling

2. General case:

Assume we can evaluate $p(z) = \frac{1}{Z_p}\tilde{p}(z)$ (unnormalized)

- Find **proposal distribution** q
  - Easy to sample from q
- Find k with $kq(z) \geq \tilde{p}(z)$
- Sample from q
- Sample uniformly from [0,kq(z_0)]
- Reject if $u_0 > \tilde{p}(z_0)$



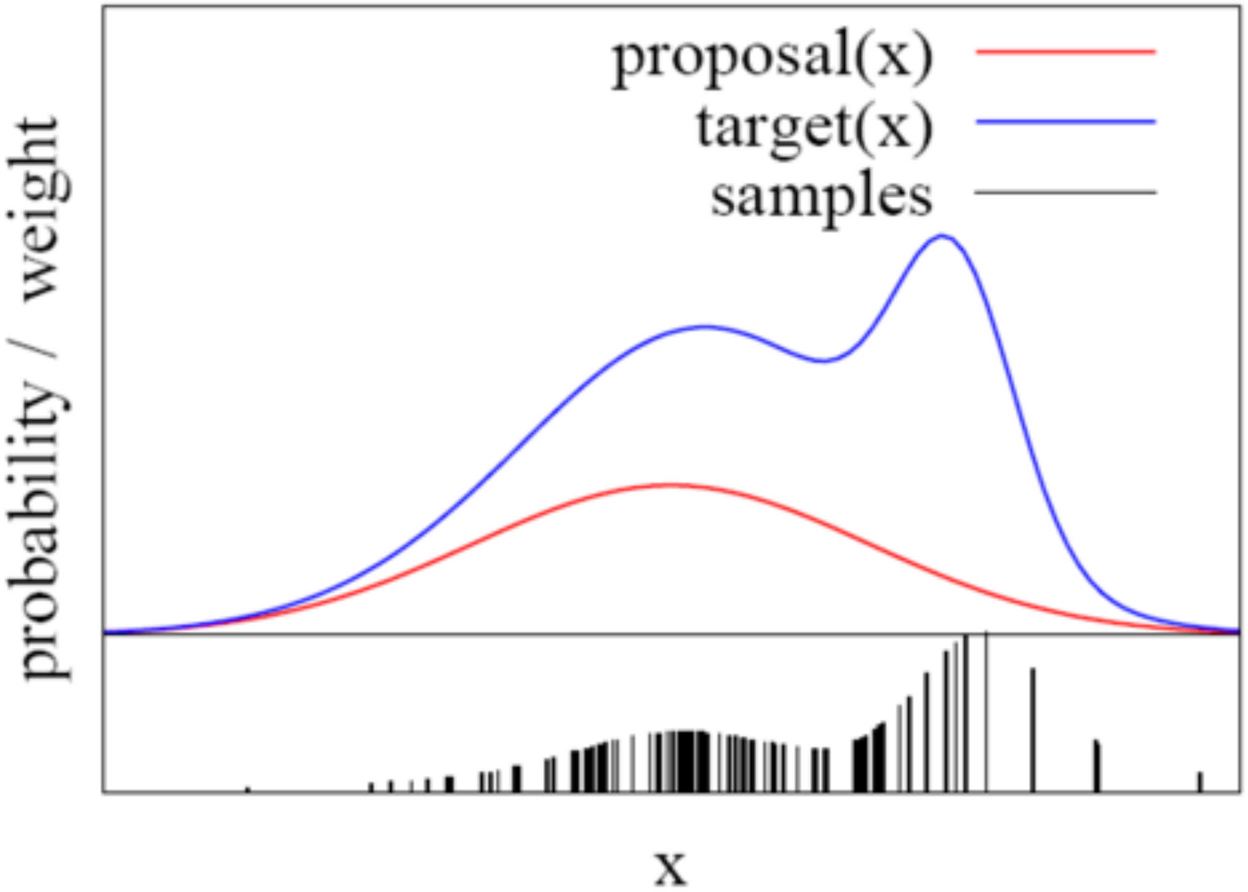
But: Rejection sampling is inefficient.

Importance Sampling

- **Idea:** assign an **importance weight** w to each sample
- With the importance weights, we can account for the "differences between p and q "

$$w(x) = p(x)/q(x)$$

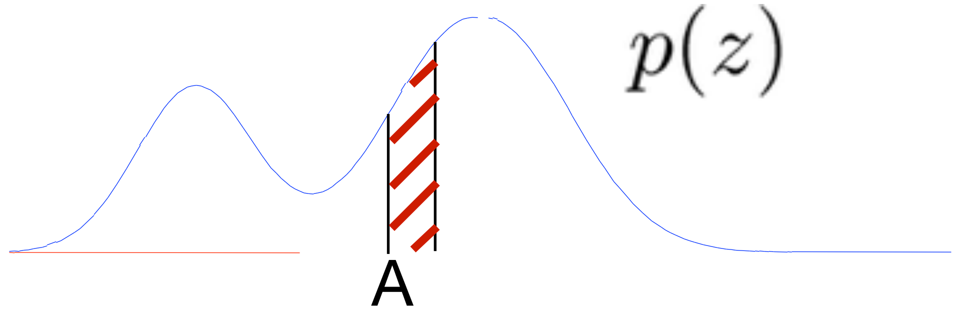
- p is called **target**
- q is called **proposal** (as before)



Importance Sampling

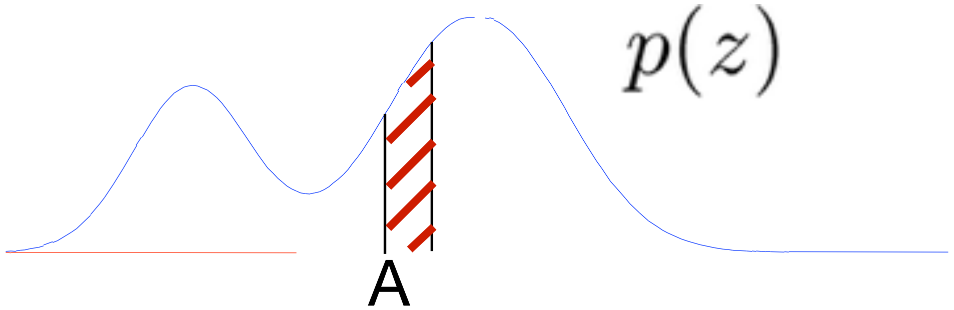
- **Explanation:** The prob. of falling in an interval A is the **area** under p
- This is equal to the expectation of the **indicator function** $I(x \in A)$

$$E_p[I(z \in A)] = \int p(z) I(z \in A) dz$$



Importance Sampling

- **Explanation:** The prob. of falling in an interval A is the **area** under p
- This is equal to the expectation of the **indicator function** $I(x \in A)$



$$E_p[I(z \in A)] = \int p(z) I(z \in A) dz$$

$$= \int \frac{p(z)}{q(z)} q(z) I(z \in A) dz = E_q[w(z) I(z \in A)]$$

Requirement: $p(x) > 0 \Rightarrow q(x) > 0$

Approximation with samples drawn from q: $$E_q[w(z) I(z \in A)] \approx \frac{1}{L} \sum_{l=1}^{L} w(z_l) I(z_l \in A)$$