A Model for Sequential Data

Idea: Introduce hidden (unobserved) variables:

Now we have: \( \text{dsep}(x_n, \{x_1, \ldots, x_{n-2}\}, x_{n-1}) \)

\[ \iff p(x_n \mid x_1, \ldots, x_{n-2}, x_{n-1}) = p(x_n \mid x_{n-1}) \]

But: \( \neg\text{dsep}(z_n, \{z_1, \ldots, z_{n-2}\}, z_{n-1}) \)

\[ \iff p(z_n \mid z_1, \ldots, z_{n-2}, z_{n-1}) \neq p(z_n \mid z_{n-1}) \]

And: number of parameters is \( nK(K-1) + \text{const.} \).
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 \( K: x_n \in \{1..K\} \)

2. Transition model \( p(x_i | x_{i-1}) \)
   - 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 \( \pi_1, \pi_2, \pi_3 \)

3. Observation model \( p(z_i | x_i) \) with parameters \( \phi \)
   - Observation only depends on the current state
   - Example: output of a “local” place classifier
The Trellis Representation

![Trellis Diagram]

- $A_{11}$
- $A_{33}$

$k=1$  
$k=2$  
$k=3$

$n-2$  
$n-1$  
n
Application Example (1)

• Given an observation sequence \( z_1, z_2, z_3 \ldots \)

• 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 $p(x_n | z_{1:n})$, i.e. state probability only based on previous observations
- **Smoothing**: computes $p(x_n | z_{1:N})$, state probability based on all observations (including those from the future)
Application Example (3)

- Given an observation sequence $z_1, z_2, z_3, \ldots$
- Assume that the model parameters $\theta = (A, \pi, \varphi)$ are known
- What is the state sequence $x_1, x_2, x_3, \ldots$ 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 $z_1, z_2, z_3, \ldots$

• 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 $\theta$ in the following)

$$p(z_1, \ldots, z_n) = \sum_{x_n} p(z_1, \ldots, z_n, x_n) =: \sum_{x_n} \alpha(x_n)$$
The Forward Algorithm

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

$$p(z_1, \ldots, z_n) = \sum_{x_n} p(z_1, \ldots, z_n, x_n) =: \sum_{x_n} \alpha(x_n)$$

We can calculate $\alpha$ recursively:

$$\alpha(x_n) = p(z_n | x_n) \sum_{x_{n-1}} \alpha(x_{n-1}) p(x_n | x_{n-1})$$
The Forward Algorithm

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

$$p(z_1, \ldots, z_n) = \sum_{x_n} p(z_1, \ldots, z_n, x_n) =: \sum_{x_n} \alpha(x_n)$$

We can calculate $\alpha$ recursively:

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

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

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

$$p(z_1, \ldots, z_n) = \sum_{x_n} p(z_1, \ldots, z_n, x_n) =: \sum_{x_n} \alpha(x_n)$$

We can calculate $\alpha$ recursively:

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

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

Filtering: $p(x_n | z_1, \ldots, z_n) = \frac{p(z_1, \ldots, z_n, x_n)}{p(z_1, \ldots, z_n)} = \frac{\alpha(x_n)}{\sum_{x_n} \alpha(x_n)}$
The Forward-Backward Algorithm

- As before we set $\alpha(x_n) = p(z_1, \ldots, z_n, x_n)$
- We also define $\beta(x_n) = p(z_{n+1}, \ldots, z_N | x_n)$

\[ \begin{align*}
\alpha(x_n) &= p(z_1, \ldots, z_n, x_n) \\
\beta(x_n) &= p(z_{n+1}, \ldots, z_N | x_n)
\end{align*} \]

e.g. $n = 5$: 

$X_1 \rightarrow X_2 \rightarrow X_3 \rightarrow X_4 \rightarrow X_n \rightarrow \ldots \rightarrow X_{N-1} \rightarrow X_N$

$Z_1 \rightarrow Z_2 \rightarrow Z_3 \rightarrow Z_4 \rightarrow Z_n \rightarrow \ldots \rightarrow Z_{N-1} \rightarrow Z_N$
The Forward-Backward Algorithm

- As before we set  \( \alpha(x_n) = p(z_1, \ldots, z_n, x_n) \)
- We also define  \( \beta(x_n) = p(z_{n+1}, \ldots, z_N | x_n) \)
- This can be recursively computed (backwards):

\[
\beta(x_{n-1}) = p(z_n, \ldots, z_N | x_{n-1}) \\
= \sum_{x_n} p(x_n, z_n, \ldots, z_N | x_{n-1}) \\
= \sum_{x_n} p(z_{n+1}, \ldots, z_N | x_n) p(z_n | x_{n-1}) p(x_n | x_{n-1}) \\
= \sum_{x_n} \beta(x_n) p(z_n | x_n) p(x_n | x_{n-1})
\]
The Forward-Backward Algorithm

- As before we set \( \alpha(x_n) = p(z_1, \ldots, z_n, x_n) \)
- We also define \( \beta(x_n) = p(z_{n+1}, \ldots, z_N | x_n) \)
- This can be recursively computed (backwards):

\[
\beta(x_n) = \sum_{x_{n+1}} \beta(x_{n+1}) p(z_{n+1} | x_{n+1}) p(x_{n+1} | x_n)
\]

- This is also known as the **message-passing algorithm** ("sum-product")!
  - forward messages \( \alpha_n \) (vector of length \( K \))
  - backward messages \( \beta_n \) (vector of length \( K \))
Smoothing with Forward-Backward

First we compute $p(x_n, z_1, \ldots, z_N)$:

$$p(x_n, z_1, \ldots, z_N) = p(z_1, \ldots, z_N \mid x_n)p(x_n)$$

$$= p(z_1, \ldots, z_n \mid x_n)p(z_{n+1}, \ldots, z_N \mid x_n)p(x_n)$$

$$= p(z_1, \ldots, z_n, x_n)p(z_{n+1}, \ldots, z_N \mid x_n)$$

$$= \alpha(x_n)\beta(x_n)$$
Smoothing with Forward-Backward

First we compute \( p(x_n, z_1, \ldots, z_N) \):

\[
p(x_n, z_1, \ldots, z_N) = \alpha(x_n)\beta(x_n)
\]

with that we can compute \( p(z_1, \ldots, z_N) \):

\[
p(z_1, \ldots, z_N) = \sum_{x_n} p(x_n, z_1, \ldots, z_N) = \sum_{x_n} \alpha(x_n)\beta(x_n)
\]
Smoothing with Forward-Backward

First we compute \( p(x_n, z_1, \ldots, z_N) \):

\[
p(x_n, z_1, \ldots, z_N) = \alpha(x_n)\beta(x_n)
\]

with that we can compute \( p(z_1, \ldots, z_N) \):

\[
p(z_1, \ldots, z_N) = \sum_{x_n} p(x_n, z_1, \ldots, z_N) = \sum_{x_n} \alpha(x_n)\beta(x_n)
\]

and finally:

\[
p(x_n \mid z_1, \ldots, z_N) = \frac{p(x_n, z_1, \ldots, z_N)}{p(z_1, \ldots, z_N)} = \frac{\alpha(x_n)\beta(x_n)}{\sum_{x_n} \alpha(x_n)\beta(x_n)}
\]
2. Computing the Most Likely States

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

- Define $\delta(x_n) = \max_{x_1, \ldots, x_{n-1}} p(x_1, \ldots, x_n | z_1, \ldots, z_n)$

  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 $p(X, Z | \theta)$

- **Define**
  \[
  \delta(x_n) = \max_{x_1, \ldots, x_{n-1}} p(x_1, \ldots, x_n | z_1, \ldots, z_n)
  \]

  This can be computed recursively:
  \[
  \delta(x_n) = \max_{x_{n-1}} \delta(x_{n-1}) p(x_n | x_{n-1}) p(z_n | x_n)
  \]

  we also have to compute the argmax:
  \[
  \psi(x_n) = \arg \max_{x_{n-1}} \delta(x_{n-1}) p(x_n | x_{n-1}) p(z_n | x_n)
  \]
The Viterbi algorithm

- Initialize:
  - \( \delta(x_0) = p(x_0) p(z_0 | x_0) \)
  - \( \psi(x_0) = 0 \)

- Compute recursively for \( n=1 \ldots N \):
  - \( \delta(x_n) = p(z_n | x_n) \max_{x_{n-1}} \left[ \delta(x_{n-1}) p(x_n | x_{n-1}) \right] \)
  - \( \psi(x_n) = \arg\max_{x_{n-1}} \left[ \delta(x_{n-1}) p(x_n | x_{n-1}) \right] \)

- On termination:
  - \( p(Z,X|\theta) = \max_{x_N} \delta(x_N) \)
  - \( x_N^* = \arg\max_{x_N} \delta(x_N) \)
- Backtracking:
  - \( x_n^* = \psi(x_{n+1}) \)
3. Learning the Model Parameters

- Given an observation sequence $z_1, z_2, z_3, \ldots$
- Find optimal model parameters $\theta = \pi, A, \phi$
- 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

\[ p(z_1, \ldots, z_N \mid \theta) = \sum_X p(z_1, \ldots, z_N, x_1, \ldots, x_N \mid \theta) \]

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

\[ \sum_X p(x_1, \ldots, x_N \mid z_1, \ldots, z_N, \theta) \log p(z_1, \ldots, z_N, x_1, \ldots, x_N \mid \theta) \]

• it can be shown that this is a lower bound of the actual log-likelihood \( p(Z \mid \theta) \)

• this is the general idea of the Expectation-Maximization (EM) algorithm
The Baum-Welsh algorithm

- E-Step (assuming we know $\pi, A, \varphi$, i.e. $\theta^{\text{old}}$)
- Define the posterior probability of being in state $i$ at step $k$:
  
  - Define $\gamma(x_n) = p(x_n | Z)$
The Baum-Welsh algorithm

- E-Step (assuming we know $\pi, A, \varphi$, i.e. $\theta^{\text{old}}$)
- Define the posterior probability of being in state $i$ at step $k$:
  - Define $\gamma(x_n) = p(x_n|z_1, \ldots, z_N)$
  - It follows that $\gamma(x_n) = \alpha(x_n) \beta(x_n) / p(Z)$
The Baum-Welsh algorithm

• E-Step (assuming we know \( \pi, A, \varphi \), i.e. \( \theta^{\text{old}} \))
• Define the posterior probability of being in state \( i \) at step \( k \):
  • Define \( \gamma(x_n) = p(x_n|z_1, \ldots, z_n) \)
  • It follows that \( \gamma(x_n) = \alpha(x_n) \beta(x_n) / p(Z) \)
• Define \( \xi(x_{n-1}, x_n) = p(x_{n-1}, x_n|Z) \)
• It follows that
  \[
  \xi(x_{n-1}, x_n) = \alpha(x_{n-1})p(z_n|x_n)p(x_n|x_{n-1})\beta(x_n) / p(Z)
  \]
The Baum-Welsh algorithm

• Note: $\gamma(x_n)$ is a vector of length $K$; each entry $\gamma_k(x_n)$ 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

$$\sum_{i=1}^{N} \gamma_k(x_i)$$
The Baum-Welsh algorithm

• Note: $\gamma(x_n)$ is a vector of length $K$; each entry $\gamma_k(x_n)$ 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 $$\sum_{i=1}^{N} \gamma_k(x_i)$$

• Similarly: The expected number of transitions from state $j$ to state $k$ in the sequence $X$ is $$\sum_{i=1}^{N-1} \xi_{j,k}(x_i, x_{i+1})$$
The Baum-Welsh algorithm

• With that we can compute new values for $\pi, A, \varphi$:

$$\pi_k = \gamma_k(x_1)$$

$$A_{j,k} = \frac{\sum_{i=1}^{N-1} \xi_{j,k}(x_i, x_{i+1})}{\sum_{i=1}^{N} \gamma_j(x_i)}$$

$$\varphi_{j,k} = \frac{\sum_{i=1}^{N} \gamma_j(x_i) \delta_{k,x_i}}{\sum_{i=1}^{N} \gamma_j(x_i)}$$

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 $\theta=(\pi,A,\phi)$
  e.g. uniformly and k-means for $\phi$
• Compute messages (E-Step)
• Compute new $\theta=(\pi,A,\phi)$ (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 represented:

- **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 $q$”

\[
w(x) = \frac{p(x)}{q(x)}
\]

• $p$ 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 $q$:

$$E_q[w(z)I(z \in A)] \approx \frac{1}{L} \sum_{l=1}^{L} w(z_l)I(z_l \in A)$$