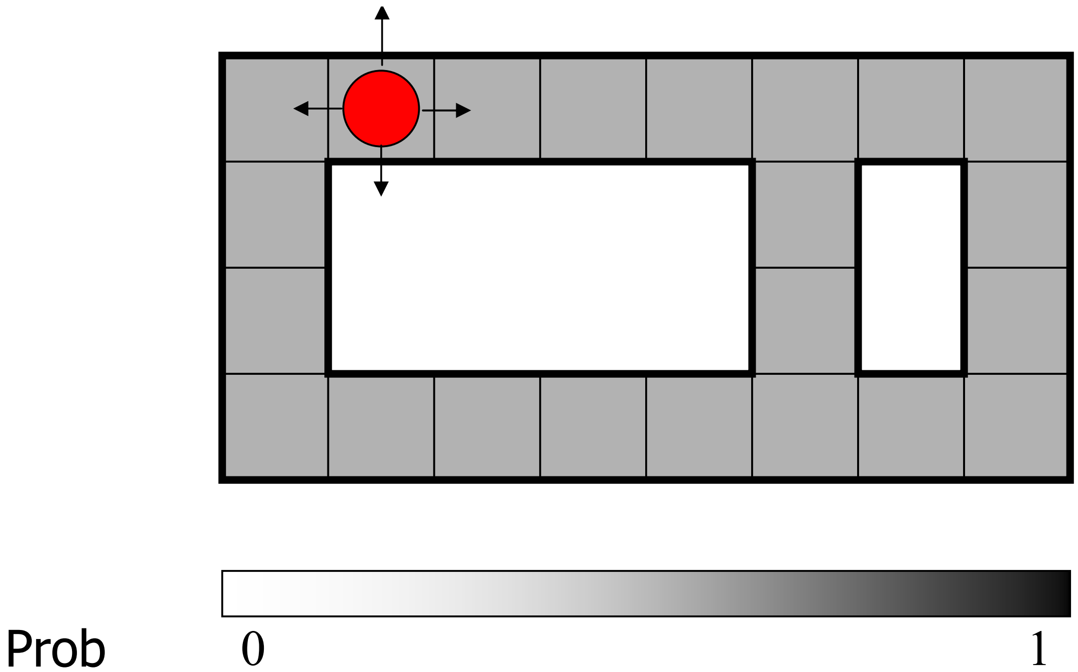
Lecture 9: Hidden Markov Models

- Working with time series data
- Hidden Markov Models
- Inference and learning problems
- Forward-backward algorithm
- Baum-Welch algorithm for parameter fitting

Time series/sequence data

- Very important in practice:
  - Speech recognition
  - Text processing (taking into account the sequence of words)
  - DNA analysis
  - Heart-rate monitoring
  - Financial market forecasting
  - Mobile robot sensor processing
  - ...
- Does this fit the machine learning paradigm as described so far?
  - The sequences are *not all the same length* (so we cannot just assume one attribute per time step)
  - The data at each time slice/index is *not independent*
  - The data distribution *may change over time*

Example: Robot position tracking[1]



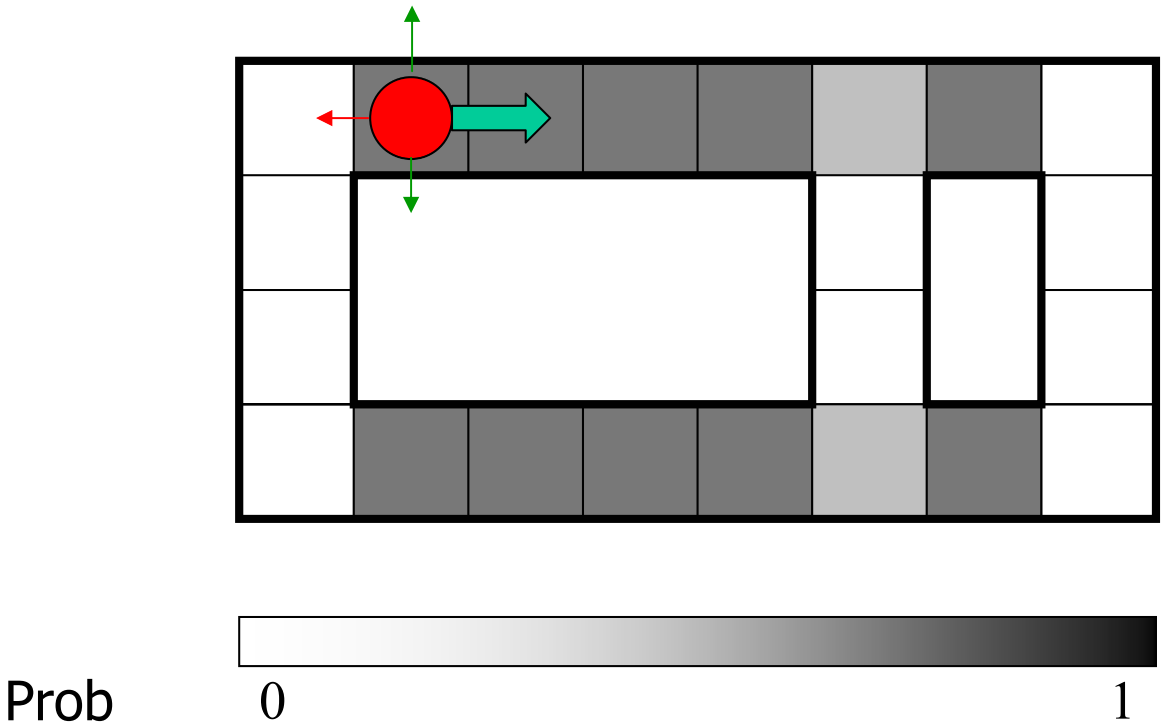
t=0

Sensory model: never more than 1 mistake

Motion model: may not execute action with small prob.

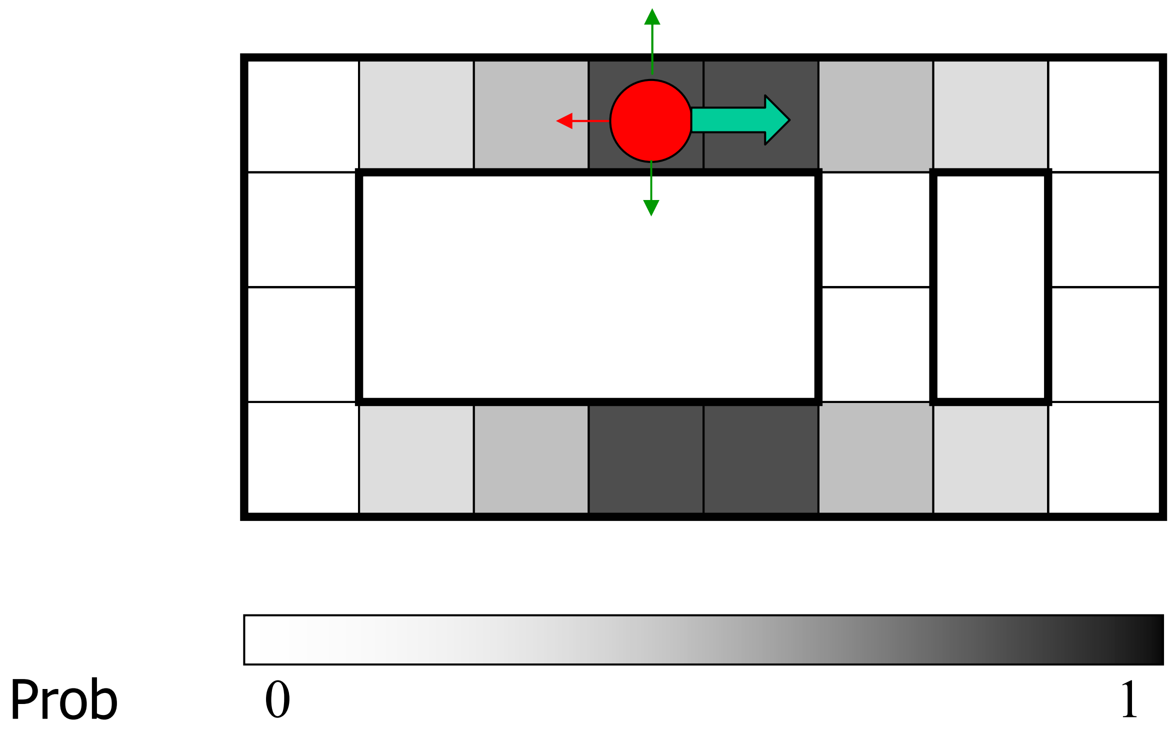
[1] From Pfeiffer, 2004

Example (II)



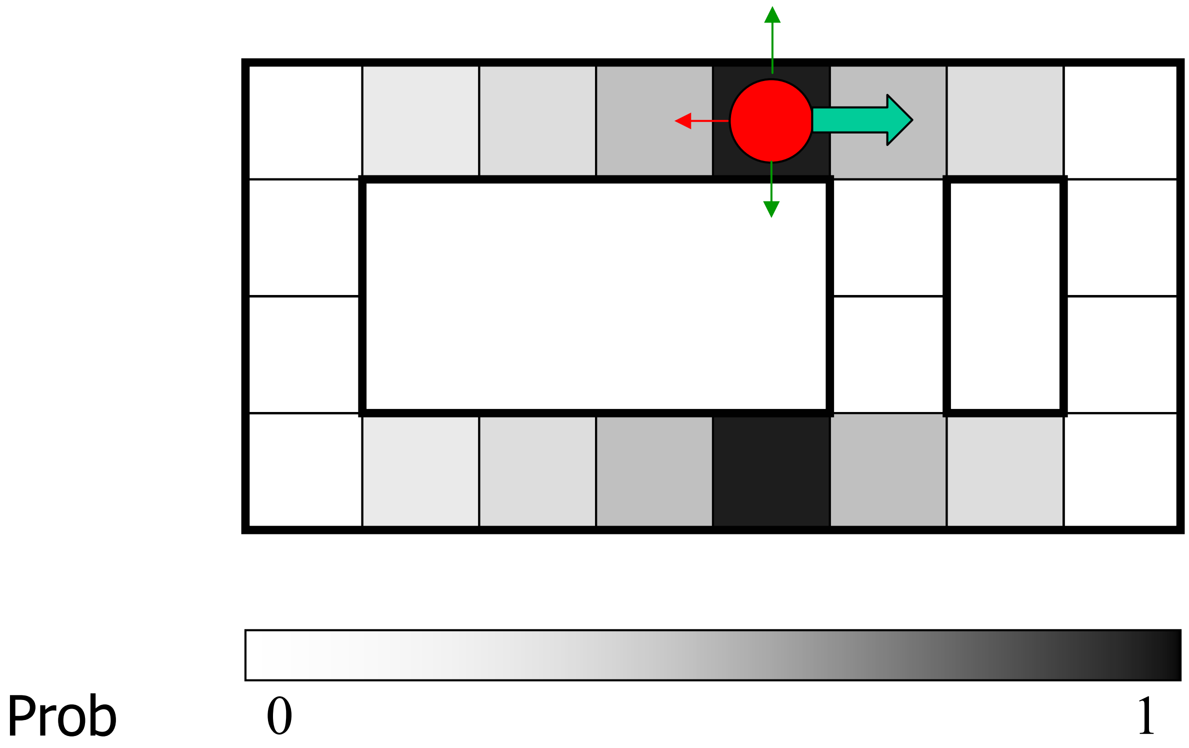
t=1

Example (III)



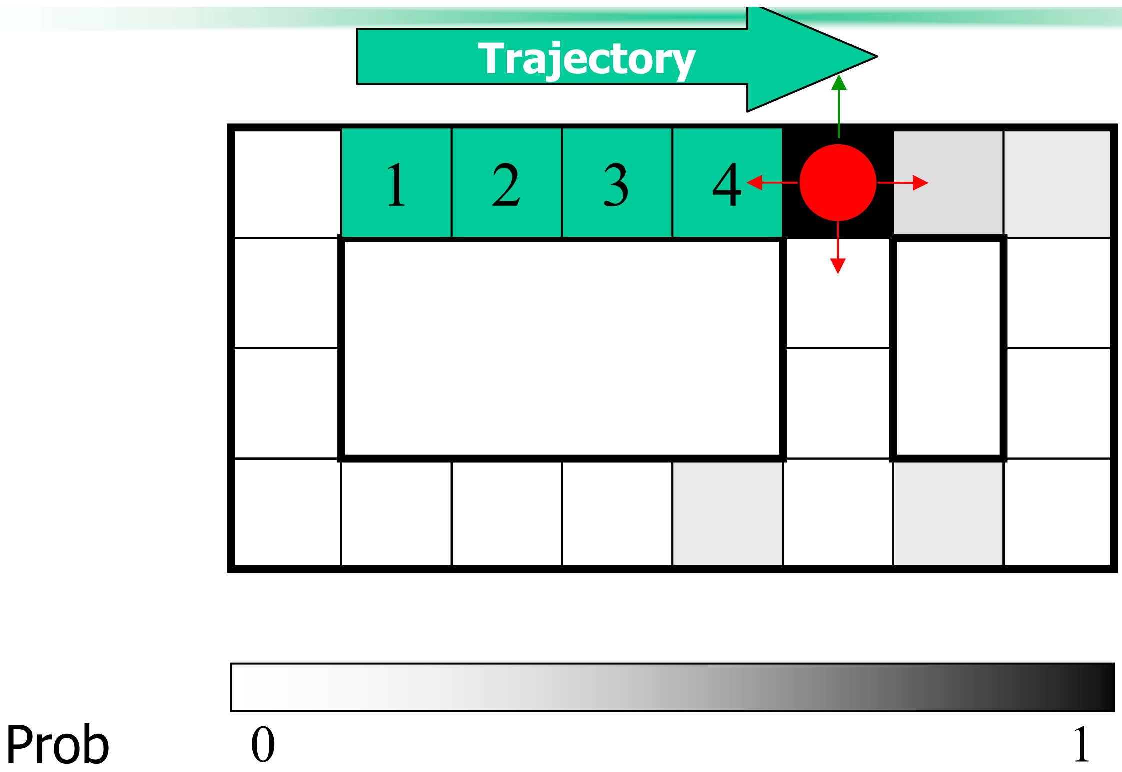
t=3

Example (IV)



t=4

Example (V)



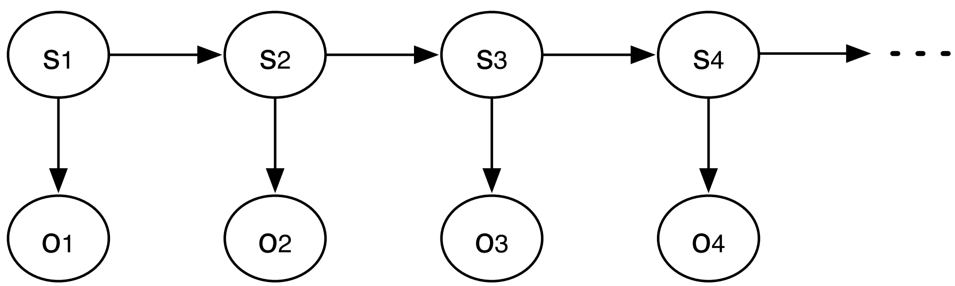
t=5

Hidden Markov Models (HMMs)

- Hidden Markov Models (HMMs) are used for situations in which:
  - The data consists of a *sequence of observations*
  - The observations depend (probabilistically) on the internal state of a *dynamical system*
  - *The true state of the system is unknown* (i.e., it is a hidden or latent variable)
- There are numerous applications, including:
  - Speech recognition
  - Robot localization
  - Gene finding
  - User modelling
  - Fetal heart rate monitoring
  - ...

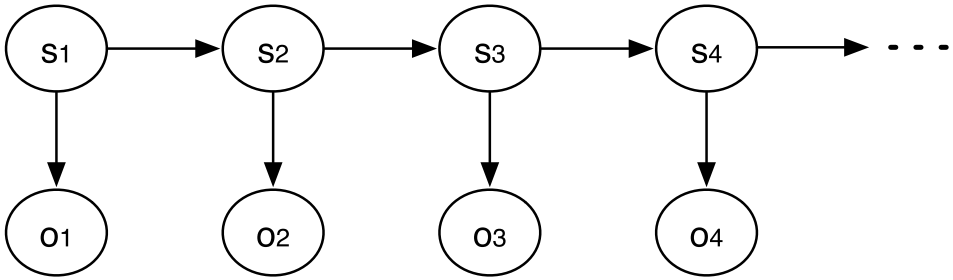
How an HMM works

- Assume a discrete clock $t = 0, 1, 2, \ldots$
- At each t, the system is in some internal (hidden) state $S_t = s$ and an observation $O_t = o$ is emitted (stochastically) *based only on s*
(Random variables are denoted with capital letters)
- The system transitions (stochastically) to a new state S_{t+1}, according to a probability distribution $P(S_{t+1}|S_t)$, and the process repeats.
- This interaction can be represented as a graphical model (recall that each circle is a random variable, S_t or O_t in this case):



- *Markov assumption*: $S_{t+1} \perp\!\!\!\perp S_{t-1} | S_t$ (future is independent of the past given the present)

HMM definition



- An HMM consists of:
  - A *set of states* $\mathcal{S}$ (usually assumed to be finite)
  - A *start state distribution* $P(S_1 = s), \forall s \in S$
    This annotates the top left node in the graphical model
  - *State transition probabilities*: $P(S_{t+1} = s'|S_t = s), \forall s, s' \in S$
    These annotate the right-going arcs in the graphical model
  - A *set of observations* $\mathcal{O}$ (often assumed to be finite)
  - *Observation emission probabilities* $P(O_t = o|S_t = s), \forall s \in S, o \in \mathcal{O}$.
    These annotate the down-going arcs above
- The model is *homogeneous*: the transition and emission probabilities *do not depend on time*, only on the states/observations

Finite HMMs

- If $\mathcal{S}$ and $\mathcal{O}$ are finite, the initial state distribution can be represented as a vector $\mathbf{b}_0$ of size $|\mathcal{S}|$
- Transition probabilities form a matrix $\mathbf{T}$ of size $|\mathcal{S}| \times |\mathcal{S}|$; each row i is the multinomial of the next state given that the current state is i
- Similarly, the emission probabilities form a matrix $\mathbf{Q}$ of size $|\mathcal{S}| \times |\mathcal{O}|$; each row is a multinomial distribution over the observations, given the state.
- Together, $\mathbf{b}_0$, $\mathbf{T}$ and $\mathbf{Q}$ form the *model* of the HMM.
- If $\mathcal{O}$ is not not finite, the multinomial can be replaced with an appropriate parametric distribution (e.g. Normal)
- If $\mathcal{S}$ is not finite, the model is usually not called an HMM, and different ways of expressing the distributions may be used, e.g
  - Kalman filter
  - Extended Kalman filter
  - ...

Examples

- Gene regulation
  - $\mathcal{O} = \{A, C, G, T\}$
  - $\mathcal{S} = \{\text{Gene, Transcription factor binding site, Junk DNA}, \dots\}$
- Speech processing
  - $\mathcal{O} =$ speech signal
  - $\mathcal{S} =$ word or phoneme being uttered
- Text understanding
  - $\mathcal{O} =$ words
  - $\mathcal{S} =$ topic (e.g. sports, weather, etc)
- Robot localization
  - $\mathcal{O} =$ sensor readings
  - $\mathcal{S} =$ discretized position of the robot

HMM problems

- How likely is a given observation sequence, $o_0, o_1, \ldots o_T$?
  I.e., compute $P(O_1 = o_1, O_2 = o_2, \ldots O_T = o_T)$
- Given an observation sequence, what is the probability distribution for the current state?
  I.e., compute $P(S_T = s | O_1 = o_1, O_2 = o_2, \ldots O_T = o_T)$
- What is the most likely state sequence for explaining a given observation sequence? ("Decoding problem")

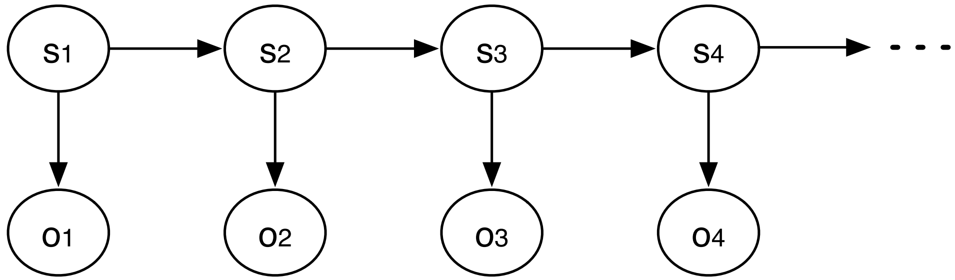
$$\arg \max_{s_1, \ldots s_T} P(S_1 = s_1, \ldots S_T = s_T | O_1 = o_1, \ldots O_T = o_T)$$

- Given one (or more) observation sequence(s), compute the model parameters

Computing the probability of an observation sequence

- Very useful in learning for:
  - Seeing if an observation sequence is likely to be generated by a certain HMM from a set of candidates (often used in classification of sequences)
  - Evaluating if learning the model parameters is working
- How to do it: *belief propagation*

Decomposing the probability of an observation sequence



$$P(o_1, \ldots o_T) = \sum_{s_1, \ldots s_T} P(o_1, \ldots o_T, s_1, \ldots s_T)$$

$$= \sum_{s_1, \ldots s_T} P(s_1) \left(\prod_{t=2}^{T} P(s_t | s_{t-1}) \right) \left(\prod_{t=1}^{T} P(o_t | s_t) \right) \quad \text{(using the model)}$$

$$= \sum_{s_T} P(o_T | s_T) \sum_{s_1, \ldots s_{T-1}} P(s_T | s_{T-1}) P(s_1) \left(\prod_{t=2}^{T-1} P(s_t | s_{t-1}) \right) \left(\prod_{t=1}^{T-1} P(o_t | s_t) \right)$$

This form suggests a dynamic programming solution!

Dynamic programming idea

- By inspection of the previous formula, note that we actually wrote:

$$
\begin{aligned}
P(o_1, o_2, \ldots o_T) &= \sum_{s_T} P(o_1, o_2, \ldots o_T, s_T) \\
&= \sum_{s_T} P(o_T|s_T) \sum_{s_{T-1}} P(s_T|s_{T-1}) P(o_1, \ldots o_{T-1}, s_{T-1})
\end{aligned}
$$

- The variables for the dynamic programming will be $P(o_1, o_2, \ldots o_t, s_t)$.

The forward algorithm

- Given an HMM model and an observation sequence $o_1, \dots o_T$, define:

$$\alpha_t(s) = P(o_1, \dots o_t, S_t = s)$$

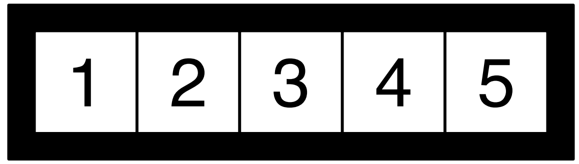
- We can put these variables together in a vector α_t of size $\mathcal{S}$.
- In particular,

$$\alpha_1(s) = P(o_1, S_1 = s) = P(o_1 | S_1 = s) P(S_1 = s) = q_{so_1} b_0(s)$$

- For $t = 2, \dots T, \alpha_t(s) = p_{so_t} \sum_{s'} p_{s's} \alpha_{t-1}(s')$
- The solution is then

$$P(o_1, \dots o_T) = \sum_s \alpha_T(s)$$

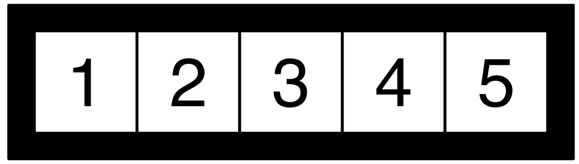
Example



- Consider the 5-state hallway shown above
- The start state is always state 3
- The observation is the number of walls surrounding the state (2 or 3)
- There is a 0.5 probability of staying in the same state, and 0.25 probability of moving left or right; if the movement would lead to a wall, the state is unchanged.

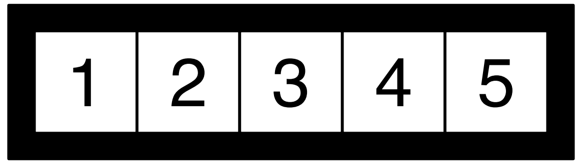
	start	to state					see walls				
state		1	2	3	4	5	0	1	2	3	4
1	0.00	0.75	0.25	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
2	0.00	0.25	0.50	0.25	0.00	0.00	0.00	0.00	1.00	0.00	0.00
3	1.00	0.00	0.25	0.50	0.25	0.00	0.00	0.00	1.00	0.00	0.00
4	0.00	0.00	0.00	0.25	0.50	0.25	0.00	0.00	1.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.25	0.75	0.00	0.00	0.00	1.00	0.00

Example: Forward algorithm



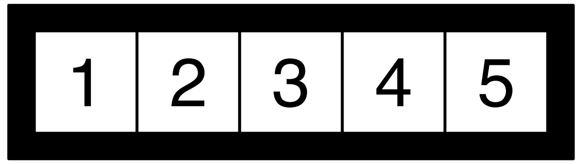
Time t	1
Obs	2
$\alpha_t(1)$	0.00000
$\alpha_t(2)$	0.00000
$\alpha_t(3)$	1.00000
$\alpha_t(4)$	0.00000
$\alpha_t(5)$	0.00000

Example: Forward algorithm



Time t	1	2
Obs	2	2
$\alpha_t(1)$	0.00000	0.00000
$\alpha_t(2)$	0.00000	0.25000
$\alpha_t(3)$	1.00000	0.50000
$\alpha_t(4)$	0.00000	0.25000
$\alpha_t(5)$	0.00000	0.00000

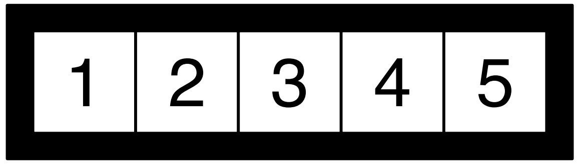
Example: Forward algorithm: two different observation sequences



Time t	1	2	3
Obs	2	2	2
$\alpha_t(1)$	0.00000	0.00000	0.00000
$\alpha_t(2)$	0.00000	0.25000	0.25000
$\alpha_t(3)$	1.00000	0.50000	0.37500
$\alpha_t(4)$	0.00000	0.25000	0.25000
$\alpha_t(5)$	0.00000	0.00000	0.00000

Time t	1	2	3
Obs	2	2	3
$\alpha_t(1)$	0.00000	0.00000	0.06250
$\alpha_t(2)$	0.00000	0.25000	0.00000
$\alpha_t(3)$	1.00000	0.50000	0.00000
$\alpha_t(4)$	0.00000	0.25000	0.00000
$\alpha_t(5)$	0.00000	0.00000	0.06250

Example: Forward algorithm



Time t	1	2	3	4	5	6	7	8	9	10
Obs	2	2	3	2	3	2	2	2	3	3
$\alpha_t(1)$	0.0	0.00	0.0625	0.00000	0.00391	0.00000	0.00000	0.00000	0.00009	0.00007
$\alpha_t(2)$	0.0	0.25	0.0000	0.01562	0.00000	0.00098	0.00049	0.00037	0.00000	0.00000
$\alpha_t(3)$	1.0	0.50	0.0000	0.00000	0.00000	0.00000	0.00049	0.00049	0.00000	0.00000
$\alpha_t(4)$	0.0	0.25	0.0000	0.01562	0.00000	0.00098	0.00049	0.00037	0.00000	0.00000
$\alpha_t(5)$	0.0	0.00	0.0625	0.00000	0.00391	0.00000	0.00000	0.00000	0.00009	0.00007

- Note that probabilities decrease with the length of the sequence
- This is due to the fact that we are looking at a joint probability; this phenomenon would not happen for conditional probabilities
- This can be a source of numerical problems for very long sequences.

Conditional probability queries in an HMM

- Because the state is never observed, we are often interested to *infer its conditional distribution* from the observations.
- There are several interesting types of queries:
  - Monitoring (filtering, belief state maintenance): what is the current state, given the past observations?
  - Prediction: what will the state be in several time steps, given the past observations?
  - Smoothing (hindsight): update the state distribution of past time steps, given new data
  - Most likely explanation: compute the most likely sequence of states that could have caused the observation sequence

Belief state monitoring

- Given an observation sequence $o_1, \ldots o_t$, the *belief state* of an HMM at time step t is defined as:

$$b_t(s) = P(S_t = s | o_1, \ldots o_t)$$

  Note that if $\mathcal{S}$ is finite b_t is a probability vector of size $\mathcal{S}$ (so its elements sum to 1)

- In particular,

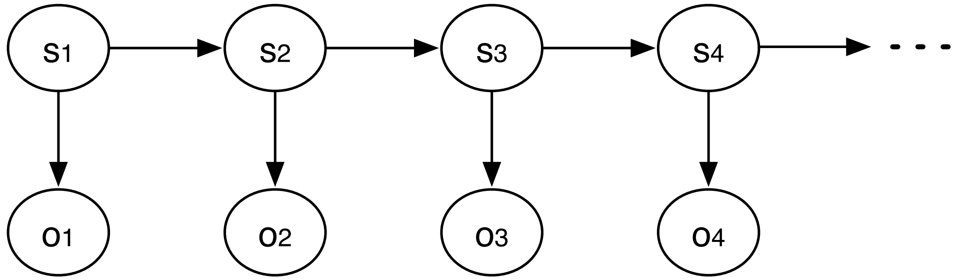
$$b_1(s) = P(S_1 = s | o_1) = \frac{P(S_1 = s, o_1)}{P(o_1)} = \frac{P(S_1 = s, o_1)}{\sum_{s'} P(S_1 = s', o_1)} = \frac{b_0(s) q_{so_1}}{\sum_{s'} b_0(s') q_{s'o_1}}$$

- To compute this, we would assign:

$$b_1(s) \leftarrow b_0(s) q_{so_1}$$

  and then normalize it (dividing by $\sum_s b_1(s)$)

Updating the belief state after a new observation

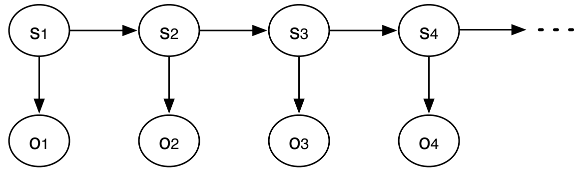


- Suppose we have $b_t(s)$ and we receive a new observation o_{t+1}. What is b_{t+1}?

$$b_{t+1}(s) = P(S_{t+1} = s | o_1, \dots o_t o_{t+1}) = \frac{P(S_{t+1} = s, o_1, \dots o_t, o_{t+1})}{P(o_1, \dots o_t, o_{t+1})}$$

- The denominator is just a normalization constant, so we will work on the numerator

Updating the belief state after a new observation (II)



$$b_{t+1}(s) \propto P(S_{t+1} = s, o_1, \dots o_t, o_{t+1})$$

$$= P(o_{t+1}|S_{t+1} = s, o_1, \dots o_t) \sum_{s'} P(S_{t+1} = s|S_t = s', o_1, \dots o_t) P(S_t = s', o_1, \dots o_t)$$

$$= P(o_{t+1}|S_{t+1} = s) \sum_{s'} P(S_{t+1} = s|S_t = s') P(S_t = s', o_1, \dots o_t) \text{ (cond. independence)}$$

$$\propto P(o_{t+1}|S_{t+1} = s) \sum_{s'} P(S_{t+1} = s|S_t = s') P(S_t = s'|o_1, \dots o_t)$$

$$= q_{so_{t+1}} \sum_{s'} b_t(s') p_{s's} \text{ (using notation)}$$

Algorithmically, at every time step t, update:

$$b_{t+1}(s) \leftarrow q_{so_{t+1}} \sum_{s'} b_t(s') p_{s's}, \qquad \text{then normalize}$$

Computing state probabilities in general

- If we know the model parameters and an observation sequence, how do we compute $P(S_t = s|o_1, o_2, \ldots o_T)$?

$$P(S_t = s|o_1, \ldots o_T) = \frac{P(o_1, \ldots o_T, S_t = s)}{P(o_1, \ldots o_T)}$$

$$= \frac{P(o_{t+1}, \ldots o_T|o_1, \ldots o_t, S_t = s)P(o_1, \ldots o_t, S_t = s)}{P(o_1, \ldots o_T)}$$

$$= \frac{P(o_{t+1}, \ldots o_T|S_t = s)P(o_1, \ldots o_t, S_t = s)}{P(o_1, \ldots o_T)}$$

- The denominator is a normalization constant and second factor in the numerator can be computed using the forward algorithm (it is $\alpha_t(s)$)
- We now compute the first factor

Computing state probabilities (II)

$$P(o_{t+1}, \ldots o_T | S_t = s) = \sum_{s'} P(o_{t+1}, \ldots o_T, S_{t+1} = s' | S_t = s)$$

$$= \sum_{s'} P(o_{t+1}, \ldots o_T | S_{t+1} = s', S_t = s) P(S_{t+1} = s' | S_t = s)$$

$$= \sum_{s'} P(o_{t+1} | S_{t+1} = s') P(o_{t+2}, \ldots o_T | S_{t+1} = s') P(S_{t+1} = s' | S_t = s)$$

$$= \sum_{s'} p_{ss'} q_{s' o_{t+1}} P(o_{t+2}, \ldots o_T | S_{t+1} = s') \text{ (using notation)}$$

- Define $\beta_t(s) = P(o_{t+1}, \ldots o_T | S_t = s)$
- Then we can compute the β_t by the following (backwards-in-time) dynamic program:

$$\beta_T(s) = 1$$

$$\beta_t(s) = \sum_{s'} p_{ss'} q_{s' o_{t+1}} \beta_{t+1}(s') \text{ for } t = T-1, T-2, T-3, \ldots$$

The forward-backward algorithm

- Given the observation sequence, $o_1, \dots o_T$ we can compute the probability of any state at any time as follows:
  1. Compute all the $\alpha_t(s)$, using the forward algorithm
  2. Compute all the $\beta_t(s)$, using the backward algorithm
  3. For any $s \in S$ and $t \in \{1, \dots T\}$:

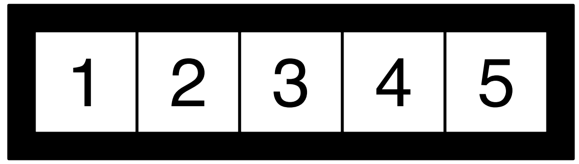
$$P(S_t = s | o_1, \dots o_T) = \frac{P(o_1, \dots o_t, S_t = s) P(o_{t+1}, \dots o_T | S_t = s)}{P(o_1, \dots o_T)} = \frac{\alpha_t(s)\beta_t(s)}{\sum_{s'} \alpha_T(s')}$$

- The complexity of the algorithm is $O(|\mathcal{S}|T)$.
- A similar dynamic programming approach can be used to compute the most likely state sequence, given a sequence of observations:

$$\arg\max_{s_1, \dots s_T} P(s_1, \dots s_T | o_1, \dots o_T)$$

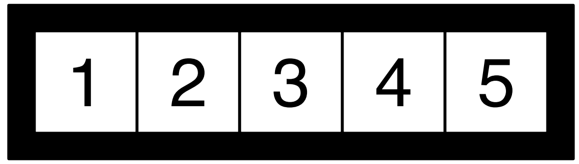
This is called the Viterbi algorithm (see Rabiner tutorial)

Example: Forward-backward algorithm



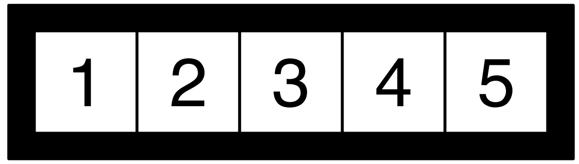
Time t	1	2	3	4	5	6
Obs	2	2	3	2	3	3
$\beta_t(1)$	0.00293	0.03516	0.04688	0.56250	0.75000	1.00000
$\beta_t(2)$	0.00586	0.01172	0.09375	0.18750	0.25000	1.00000
$\beta_t(3)$	0.00586	0.00000	0.09375	0.00000	0.00000	1.00000
$\beta_t(4)$	0.00586	0.01172	0.09375	0.18750	0.25000	1.00000
$\beta_t(5)$	0.00293	0.03516	0.04688	0.56250	0.75000	1.00000

Example: Forward-backward algorithm



Time t	1	2	3	4	5	6
Obs	2	2	3	2	3	3
$\alpha_t(1)$	0.00000	0.00000	0.06250	0.00000	0.00391	0.00293
$\alpha_t(2)$	0.00000	0.25000	0.00000	0.01562	0.00000	0.00000
$\alpha_t(3)$	1.00000	0.50000	0.00000	0.00000	0.00000	0.00000
$\alpha_t(4)$	0.00000	0.25000	0.00000	0.01562	0.00000	0.00000
$\alpha_t(5)$	0.00000	0.00000	0.06250	0.00000	0.00391	0.00293

Example: Forward-backward algorithm



Time t	1	2	3	4	5	6
Obs	2	2	3	2	3	3
$P(S_t = 1 \mid o_1, \ldots o_6)$	0.0	0.0	0.5	0.0	0.5	0.5
$P(S_t = 2 \mid o_1, \ldots o_6)$	0.0	0.5	0.0	0.5	0.0	0.0
$P(S_t = 3 \mid o_1, \ldots o_6)$	1.0	0.0	0.0	0.0	0.0	0.0
$P(S_t = 4 \mid o_1, \ldots o_6)$	0.0	0.5	0.0	0.5	0.0	0.0
$P(S_t = 5 \mid o_1, \ldots o_6)$	0.0	0.0	0.5	0.0	0.5	0.5

Learning HMM parameters

- Suppose we have access to observation sequences $o_1, \dots o_T$, and we know the state set $\mathcal{S}$. How can we find the parameters $\lambda = (p_{ss'}, q_{so}, b_0(s))$ of the HMM that generated the observations?
- Usual optimization criterion: *maximize the likelihood of the observed data* (we focus on this)
- Alternatively, in the Bayesian view, maximize the posterior probability of the observed data, given the prior over parameters
- Two main approaches:
  - Baum-Welch algorithm (an instance of Expectation-Maximization for the special case of HMM)
  - Cheat! Get complete trajectories, $s_1, o_1, s_2, o_2, \dots s_T, o_T$ and maximize $P(s_1, o_1, \dots s_T, o_T | \lambda)$
- Some other, direct optimization approaches are also possible with complete data, but less popular

Learning with complete state information

- In many applications, we can make special arrangements to obtain state information, at least for a few trajectories. For example:
  - In speech recognition, human listeners can determine exactly what word or phoneme is being spoken at each moment
  - In gene identification, biological experiments can verify what parts of the DNA are actually genes
  - In robot localization, we can collect data in a controlled environment where the robot's location is verified by other means (e.g., tape measure)
- Thus, at some extra (possibly high) cost, we can often obtain trajectories that include the true system state: $s_1, o_1, \ldots s_T, o_T$.
- It is *much, much, much easier* to train HMMs with such data than with observation data alone!
- If there is little complete data, this approach can be used to initialize the parameters before Baum-Welch

Maximum likelihood learning with complete data in finite HMM

- Suppose that we have a finite state set $\mathcal{S}$ and observation set $\mathcal{O}$
- Suppose we have a set of m trajectories, with the i^{th} trajectory of the form:

$$\tau^i = (s^i_1, o^i_1, s^i_2, o^i_2, \dots s^i_{T^i}, o^i_{T^i})$$

- Maximum likelihood estimates of the HMM parameters are:

$$b_0(s) = \frac{\text{\# trajectories starting at } s}{m} = \frac{|\{i : s^i_1 = s\}|}{m}$$

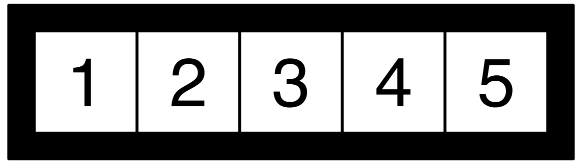
$$p_{ss'} = \frac{\text{number of } s\text{-to-}s' \text{ transitions}}{\text{number of occurrences of } s} = \frac{|\{(i,t) : s^i_t = s \text{ and } s^i_{t+1} = s'\}|}{|\{(i,t) : s^i_t = s \text{ and } t < T^i\}|}$$

$$q_{so} = \frac{\text{number of times } o \text{ was emitted in } s}{\text{number of occurrences of } s} = \frac{|\{(i,t) : s^i_t = s \text{ and } o^i_t = o\}|}{|\{(i,t) : s^i_t = s\}|}$$

What if the observation space is infinite?

- An adequate parametric representation is chosen for the observation distribution q_s at each discrete state s
  E.g. Gaussian, exponential etc.
- The parameters of q_s are then learned to maximize the likelihood of the observation data associated with s
- E.g. for a Gaussian, we can compute the mean and covariance of the observation vectors seen from each state s.

Example



- Data: one state-observation trajectory of 100 time steps
- Maximum likelihood model:

	start	to state					see walls				
state		1	2	3	4	5	0	1	2	3	4
1	0.00	0.64	0.36	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
2	0.00	0.18	0.59	0.23	0.00	0.00	0.00	0.00	1.00	0.00	0.00
3	1.00	0.00	0.25	0.35	0.40	0.00	0.00	0.00	1.00	0.00	0.00
4	0.00	0.00	0.00	0.20	0.63	0.17	0.00	0.00	1.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.45	0.55	0.00	0.00	0.00	1.00	0.00

- Note that the emission model is correct but the transition model still has errors compared to the true one, due to the limited amount of data
- In the limit, as $t \to \infty$, the learned model would converge to the true parameters

Maximum likelihood learning without state information

- Suppose we know $\mathcal{O}$ and $\mathcal{S}$ and they are finite
- Suppose we have a single observation trajectory $o_1, o_2, \ldots o_T$
- We want to solve the following optimization problem:

$$\begin{aligned}
\max \quad & P(o_1, \ldots o_T) \\
\text{w.r.t.} \quad & b_0(s), p_{ss'}, q_{so} \\
\text{s.t.} \quad & b_0(s), p_{ss'}, q_{so} \in [0, 1] \\
& \sum_s b_0(s) = 1 \\
& \sum_{s'} p_{ss'} = 1, \forall s \\
& \sum_o q_{so} = 1, \forall s
\end{aligned}$$

Learning without state information: Baum-Welch

- The Baum-Welch algorithm is an Expectation-Maximization (EM) algorithm for fitting HMM parameters.
- Recall that EM is a general approach for dealing with missing data, by alternating two steps:
  - "Fill in" the missing values based on the current model parameters
  - Re-compute the model parameters to maximize the likelihood of the completed data
- For HMMs, the missing data is the state sequence, so we start with an initial guess about the model parameters and alternate the following steps:
  - *Estimate the probability of the state sequence* given the observation sequence (using forward-backard algorithm)
  - *Fit new model parameters* based on the completed data (using the maximum likelihood algorithm)

Baum-Welch algorithm

- Given observation sequence $o_1, \dots o_T$ and initial parameters $\lambda = (b_0(s), p_{ss'}, q_{so})$
- Repeat the following steps until convergence:
  - E-Step:
    1. For every s, t compute: $P(S_t = s | o_1, \dots o_T)$
    2. For every s, s', t compute: $P(S_t = s, S_{t+1} = s' | o_1, \dots o_T)$
  - M-Step:

$$b_0(s) = P(S_1 = s | o_1, \dots o_T)$$

$$p_{ss'} = \frac{\text{Expected \# of } s \to s'}{\text{Expected } s \text{ occurences}} = \frac{\sum_{t<T} P(S_t = s, S_{t+1} = s' | o_1, \dots o_T)}{\sum_{t<T} P(S_t = s | o_1, \dots o_T)}$$

$$q_{so} = \frac{\text{Expected \# } o \text{ was emitted from } s}{\text{Expected } s \text{ occurrences}} = \frac{\sum_{t: o_t = o} P(S_t = s | o_1, \dots o_T)}{\sum_t P(S_t = s | o_1, \dots o_T)}$$

Details of E-Step

- $P(S_t = s|o_1, \dots o_T)$ is computed by the forward-backward algorithm.
- Recall: $P(S_t = s, S_{t+1} = s'|o_1, \dots o_T) = \frac{P(S_t=s, S_{t+1}=s', o_1, \dots o_T)}{P(o_1, \dots o_T)}$ where the denominator is $\sum_s \alpha_T(s)$.
- Working on the numerator:

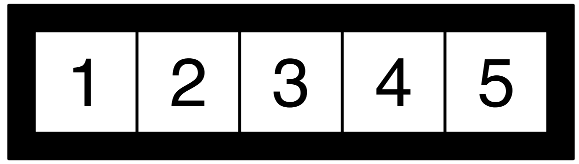
$$
\begin{aligned}
& P(S_t = s, S_{t+1} = s', o_1, \dots o_T) \\
& \quad = P(S_t = s, o_1, \dots o_t) P(S_{t+1} = s', o_{t+1}, \dots o_T | S_t = s, o_1, \dots o_t) \\
& \quad = \alpha_t(s) P(S_{t+1} = s' | S_t = s) P(o_{t+1}, \dots o_T | S_{t+1} = s') \\
& \quad = \alpha_t(s) p_{ss'} P(o_{t+1} | S_{t+1} = s') P(o_{t+1}, \dots o_T | S_{t+1} = s') \\
& \quad = \alpha_t(s) p_{ss'} q_{s' o_{t+1}} \beta_{t+1}(s')
\end{aligned}
$$

where the α's and β's are from the forward-backward algorithm.

Remarks on Baum-Welch

- Each iteration increases $P(o_1, \ldots o_T)$ (since this is EM)
- Each iteration is computationally efficient:
  - E-step: $O(|\mathcal{S}|T)$ for forward-backward, plus $O(|\mathcal{S}|^2T)$ for the second estimation
  - M-step: $O(|\mathcal{S}|^2T)$ plus $O(|\mathcal{S}||\mathcal{O}|T)$ for parameter estimation (given that we already have the αs and βs)
- Iterations are stopped when the parameters do not change much (or after a fixed amount of time)
- The algorithm converges to a *local maximum of the likelihood*
- There can be *many, many local maxima that are not globally optimal*
- Reasonable initial guesses for parameters (obtained from prior knowledge, or from learning with a small amount of complete data) are a big help, but not a guarantee for good performance

Example: Baum-Welch from correct parameters



Learned model:

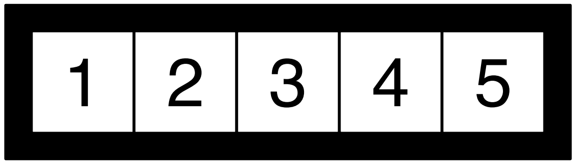
	start	to state					see walls				
state		1	2	3	4	5	0	1	2	3	4
1	0.00	0.59	0.41	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
2	0.00	0.35	0.01	0.65	0.00	0.00	0.00	0.00	1.00	0.00	0.00
3	1.00	0.00	0.20	0.60	0.20	0.00	0.00	0.00	1.00	0.00	0.00
4	0.00	0.00	0.00	0.65	0.01	0.35	0.00	0.00	1.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.41	0.59	0.00	0.00	0.00	1.00	0.00

Correct model:

	start	to state					see walls				
state		1	2	3	4	5	0	1	2	3	4
1	0.00	0.75	0.25	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
2	0.00	0.25	0.50	0.25	0.00	0.00	0.00	0.00	1.00	0.00	0.00
3	1.00	0.00	0.25	0.50	0.25	0.00	0.00	0.00	1.00	0.00	0.00
4	0.00	0.00	0.00	0.25	0.50	0.25	0.00	0.00	1.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.25	0.75	0.00	0.00	0.00	1.00	0.00

Likelihood of data: 3.8645e-19

Example: Baum-Welch from equal initial parameters (uniform initial distributions)

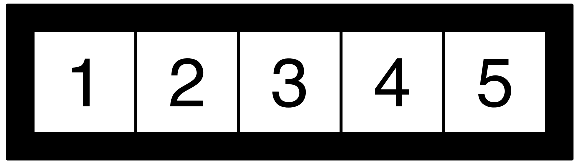


- Learned model:

	start	to state					see walls				
state		1	2	3	4	5	0	1	2	3	4
1	0.20	0.20	0.20	0.20	0.20	0.20	0.00	0.00	0.77	0.23	0.00
2	0.20	0.20	0.20	0.20	0.20	0.20	0.00	0.00	0.77	0.23	0.00
3	0.20	0.20	0.20	0.20	0.20	0.20	0.00	0.00	0.77	0.23	0.00
4	0.20	0.20	0.20	0.20	0.20	0.20	0.00	0.00	0.77	0.23	0.00
5	0.20	0.20	0.20	0.20	0.20	0.20	0.00	0.00	0.77	0.23	0.00

- Note that the learned model is *really different* from the true model
- Likelihood of data: 3.7977e-24

Example: Baum-Welch from randomly chosen initial parameters



- Learned model:

	start	to state					see walls				
state		1	2	3	4	5	0	1	2	3	4
1	0.00	0.07	0.04	0.16	0.00	0.73	0.00	0.00	0.00	1.00	0.00
2	1.00	0.00	0.22	0.31	0.47	0.00	0.00	0.00	1.00	0.00	0.00
3	0.00	0.00	0.79	0.21	0.00	0.00	0.00	0.00	1.00	0.00	0.00
4	0.00	0.48	0.05	0.47	0.00	0.00	0.00	0.00	1.00	0.00	0.00
5	0.00	0.00	0.00	0.01	0.59	0.40	0.00	0.00	0.00	1.00	0.00

- Note that the emission model is learned correctly, but the transition model is still quite different from the true model
- Likelihood of data: 1.7665e-17

The moral of the experiments

- The solution provided by EM can be *arbitrarily different* from the true model. Hence, interpreting the parameters learned by EM as having a meaning for the true problem is wrong
- Even when starting with the true model, EM may converge to something different
- Some of the solutions provided by EM are useless (e.g. when starting with uniform parameters)
- Choosing parameters at random is better than making them all equal, because it helps break symmetry
- A model with better likelihood *is not necessarily closer to the true model* (see training from the true model vs. training from a randomly chosen model)
- In general, in order to get EM to work, you either need a good initial model, or you need to do lots of random restarts

Learning the HMM structure

- All algorithms so far assume that we know the number of states
- If the number of states is not known, we can guess it and then learn parameters
- Note that the likelihood of the data usually *increases with more states*
- As a result, models with lots of states need to be penalized (using regularization, minimum description length or a Bayesian prior over the number of states)
- If $\mathcal{S}$ is unknown, the algorithms work a lot worse

Application: Detection of DNA regions

- Observation: DNA sequence
- Hidden state: gene, transcription factor, protein-coding region...
- Learning: EM
- Validation often against known regions, and then through biological experiment

Application: Music composition

- Observations: notes played
- States: chords
- Learning: music by one composer, labelled with correct chords, used for maximum likelihood learning
- Model "composes" by sampling chords and notes from the model
- If successful, new music is generated "in the style" of the composer

Application: Speech recognition

- Observations: sound wave readings
- States: phonemes
- Learning: use labelled data to initialize the model, then EM with a much larger set of speakers to further adapt the parameters
- Transcription system: use inference to determine the most likely state sequence, which provides the transcription of the word
- HMMs are the state-of-art speech recognition technology
- Can be coupled with classification, if desired, to improve recognition performance

Application: Classification of time series

- Use one HMM for each class, and learn its parameters from data
- When given a new observation sequence, compute its likelihood under each HMM
- The example is assigned the label of the class that yields the highest likelihood

Summary

- Hidden Markov Models formalize sequential observation of a system without perfect access to state (i.e., state is "hidden")
- A variety of inference problems can be solved using straightforward dynamic programming algorithms
- The learning (parameter fitting) problem is best done with "supervised" data – i.e., state & observation trajectories
- Parameter fitting can also be solved purely from observation data using EM (called the Baum-Welch algorithm), but results are only locally optimal
- EM can behave in strange ways, so getting it to work may take effort
- Lots of applications!