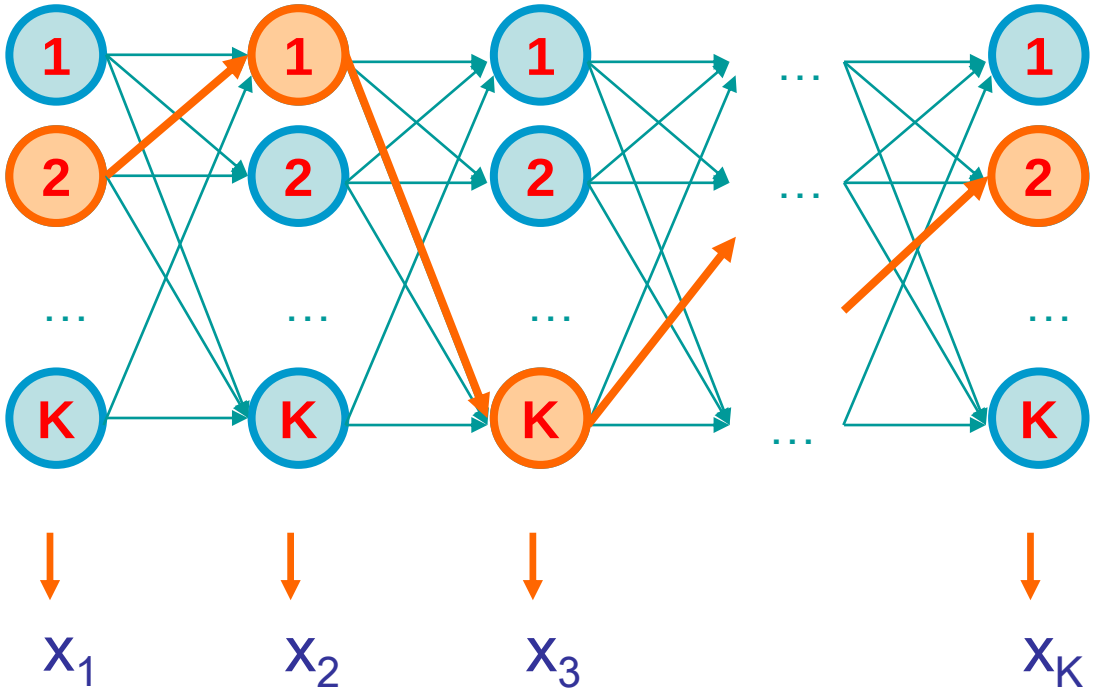
Hidden Markov Models



Time and uncertainty

The world changes; we need to track and predict it

Diabetes management vs vehicle diagnosis

Basic idea: copy state and evidence variables for each time step

$\mathbf{X}_t$ = set of unobservable state variables at time t
e.g., $BloodSugar_t$, $StomachContents_t$, etc.

$\mathbf{E}_t$ = set of observable evidence variables at time t
e.g., $MeasuredBloodSugar_t$, $PulseRate_t$, $FoodEaten_t$

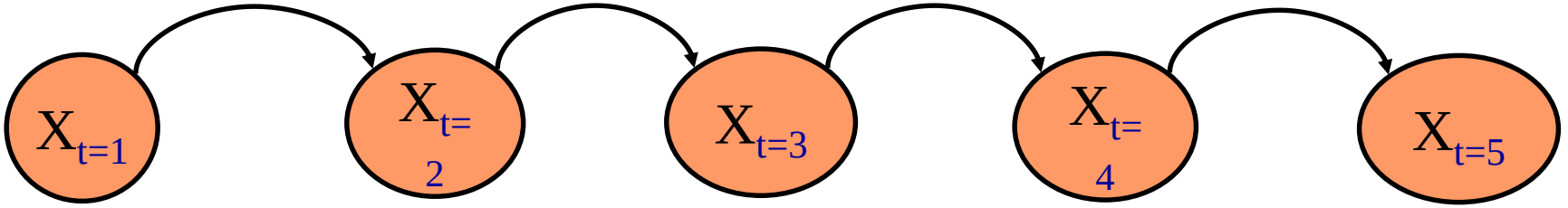
This assumes **discrete time**; step size depends on problem

Notation: $\mathbf{X}_{a:b} = \mathbf{X}_a, \mathbf{X}_{a+1}, \ldots, \mathbf{X}_{b-1}, \mathbf{X}_b$

Markov Property

- **Markov Property: The state of the system at time t+1 depends only on the state of the system at time t**

$$P[X_{t+1} = x_{t+1} \mid X_t = x_t, X_{t-1} = x_{t-1}, \ldots, X_1 = x_1, X_0 = x_0] = P[X_{t+1} = x_{t+1} \mid X_t = x_t]$$

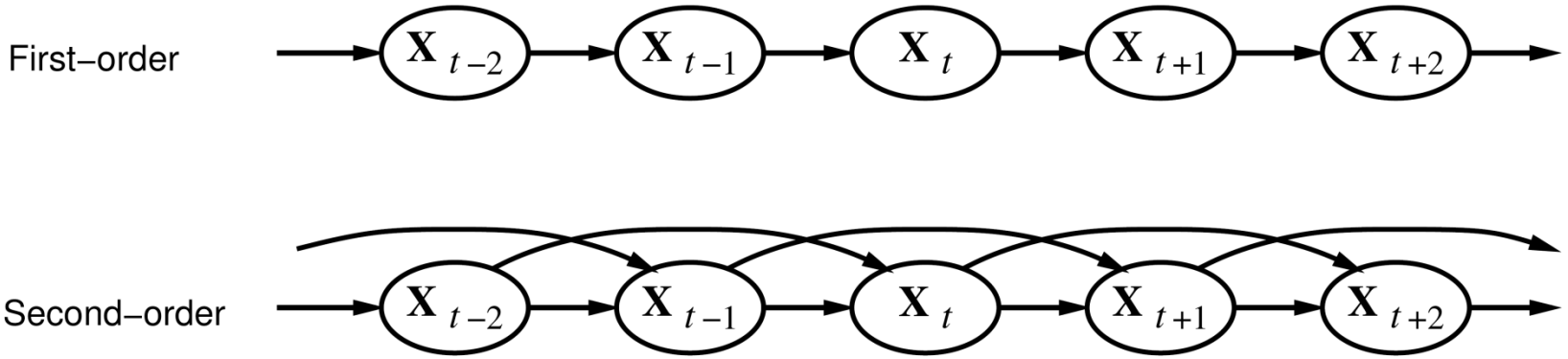


Markov Processes

Construct a Bayes net from these variables: parents?

Markov assumption: $\mathbf{X}_t$ depends on **bounded** subset of $\mathbf{X}_{0:t-1}$

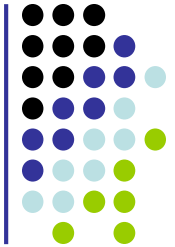
First-order Markov process: $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{0:t-1}) = \mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-1})$
Second-order Markov process: $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{0:t-1}) = \mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-2}, \mathbf{X}_{t-1})$



Sensor Markov assumption: $\mathbf{P}(\mathbf{E}_t|\mathbf{X}_{0:t}, \mathbf{E}_{0:t-1}) = \mathbf{P}(\mathbf{E}_t|\mathbf{X}_t)$

Stationary process: transition model $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-1})$ and
sensor model $\mathbf{P}(\mathbf{E}_t|\mathbf{X}_t)$ fixed for all t

Markov Chains



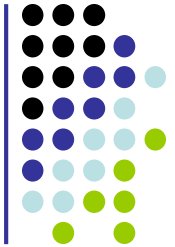
Stationarity Assumption

Probabilities independent of t when process is "stationary"

So, $\text{for all t, } P[X_{t+1} = x_j | X_t = x_i] = p_{ij}$

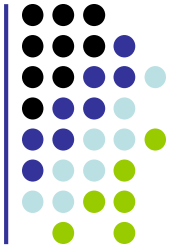
This means that if system is in state i, the probability that the system will next move to state j is p_{ij} , no matter what the value of t is

Simple Minded Weather Example



- raining today ⇒ rain tomorrow $p_{rr} = 0.4$
- raining today ⇒ no rain tomorrow $p_{rn} = 0.6$
- no raining today ⇒ rain tomorrow $p_{nr} = 0.2$
- no raining today ⇒ no rain tomorrow $p_{rr} = 0.8$

Simple Minded Weather Example

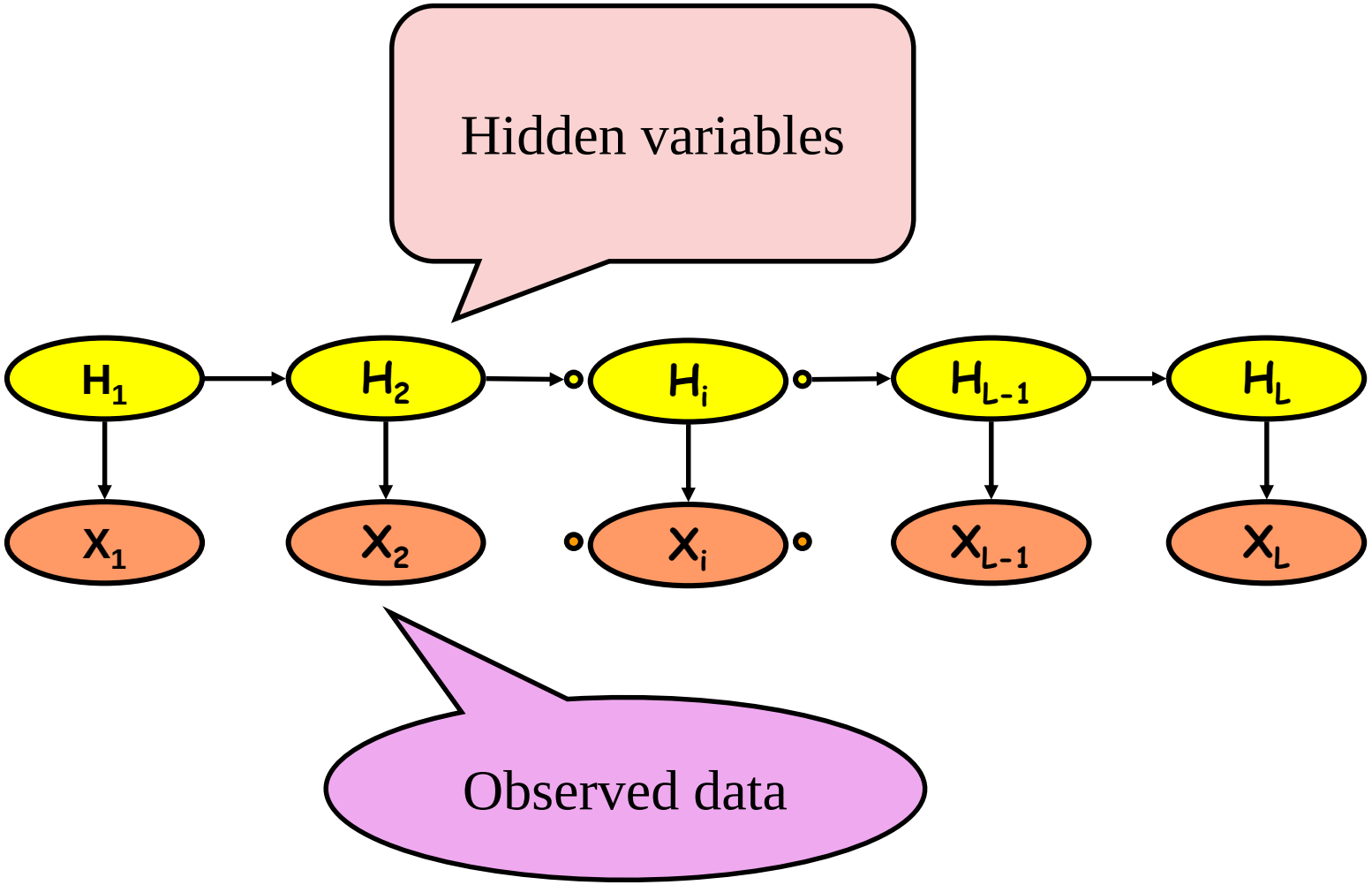


Transition matrix for our example

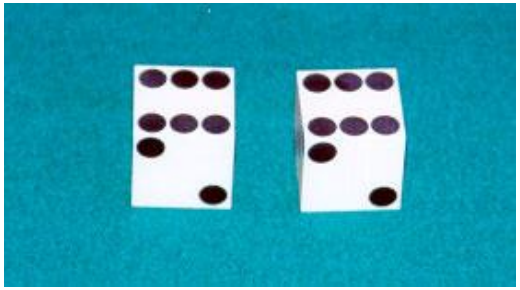
$$P = \begin{pmatrix} 0.4 & 0.6 \\ 0.2 & 0.8 \end{pmatrix}$$

- **Note that rows sum to 1**
- **Such a matrix is called a Stochastic Matrix**
- **If the rows of a matrix and the columns of a matrix all sum to 1, we have a Doubly Stochastic Matrix**

Hidden Markov Models - HMM



Example: The dishonest casino



A casino has two dice:

- Fair die

  P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = 1/6

- Loaded die

  P(1) = P(2) = P(3) = P(4) = P(5) = 1/10

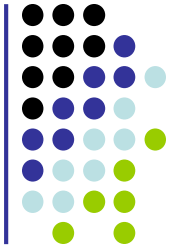
  P(6) = 1/2

Casino player switches between fair and loaded die with probability 1/20 at each turn

Game:

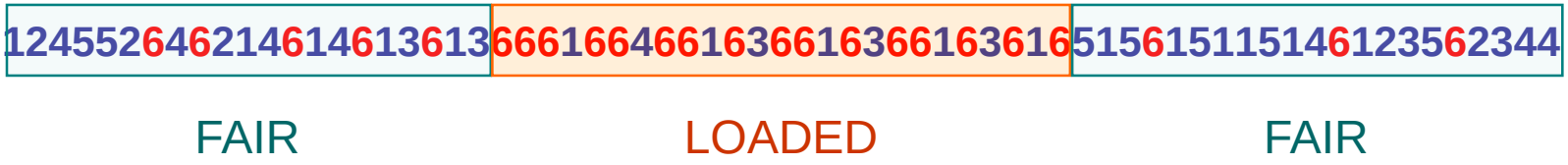
1. You bet $1
2. You roll (always with a fair die)
3. Casino player rolls (maybe with fair die, maybe with loaded die)
4. Highest number wins $2

Question # 1 – Decoding



GIVEN

A sequence of rolls by the casino player

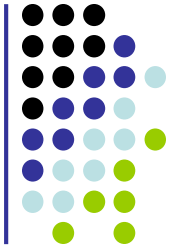


QUESTION

What portion of the sequence was generated with the fair die, and what portion with the loaded die?

This is the **DECODING** question in HMMs

Question # 2 – Evaluation



GIVEN

A sequence of rolls by the casino player

1245526462146146136136661664661636616366163616515615115146123562344

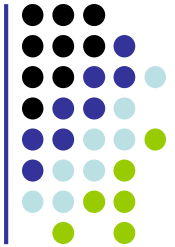
Prob = 1.3×10^{-35}

QUESTION

How likely is this sequence, given our model of how the casino works?

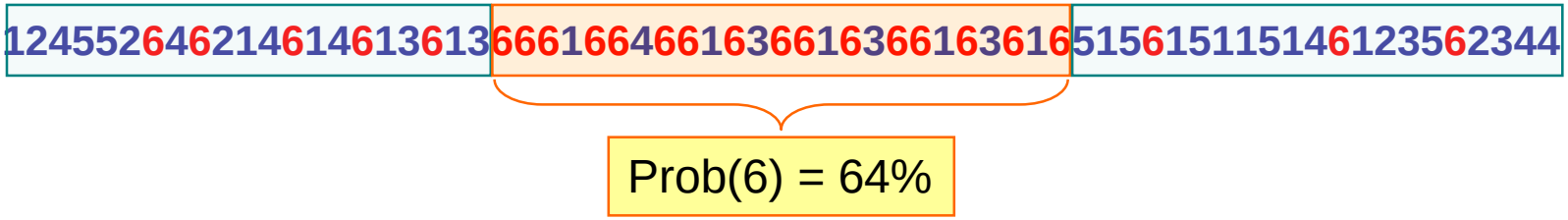
This is the **EVALUATION** problem in HMMs

Question # 3 – Learning



GIVEN

A sequence of rolls by the casino player

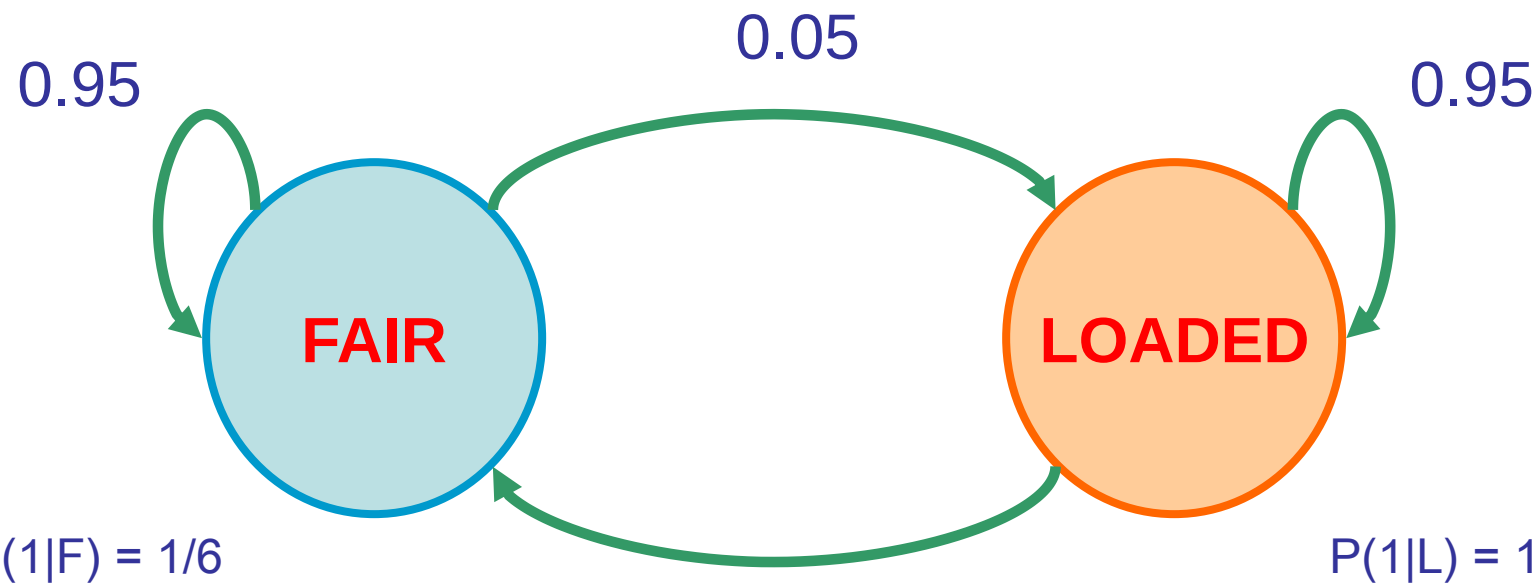


QUESTION

How “loaded” is the loaded die? How “fair” is the fair die? How often does the casino player change from fair to loaded, and back?

This is the **LEARNING** question in HMMs

The dishonest casino model



P(1|F) = 1/6
P(2|F) = 1/6
P(3|F) = 1/6
P(4|F) = 1/6
P(5|F) = 1/6
P(6|F) = 1/6

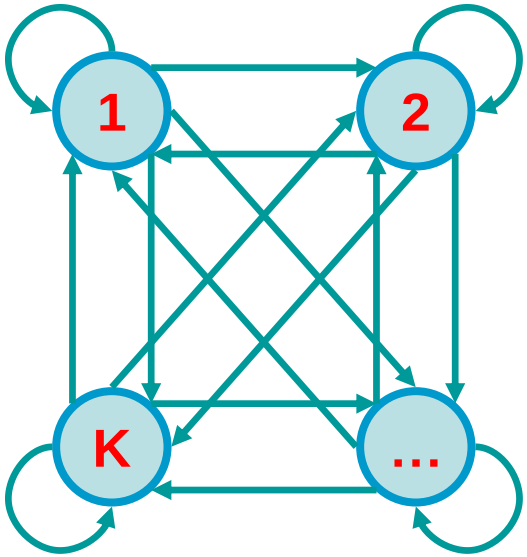
P(1|L) = 1/10
P(2|L) = 1/10
P(3|L) = 1/10
P(4|L) = 1/10
P(5|L) = 1/10
P(6|L) = 1/2

An HMM is memoryless

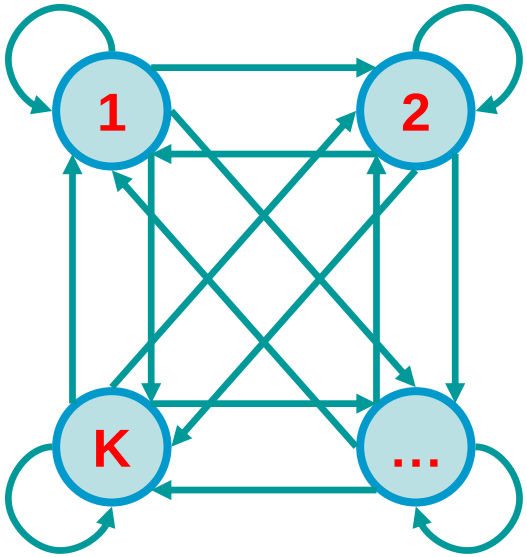
At each time step t,

the only thing that affects future states

is the current state z_t



An HMM is memoryless



At each time step t,

the only thing that affects future states

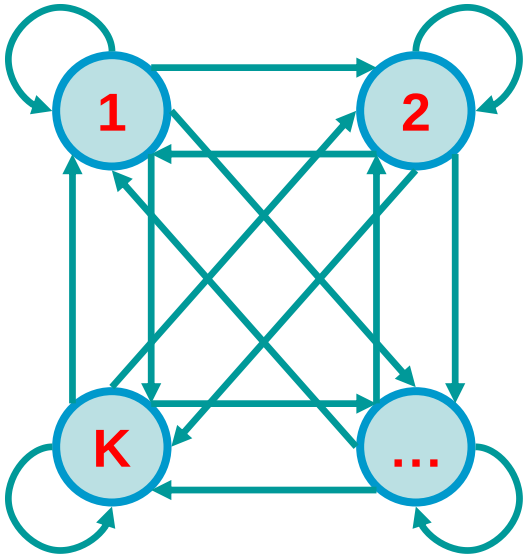
is the current state z_t

$P(z_{t+1} = k \mid \text{“whatever happened so far”}) =$

$P(z_{t+1} = k \mid z_1, z_2, \ldots, z_t, x_1, x_2, \ldots, x_t) =$

$P(z_{t+1} = k \mid z_t)$

An HMM is memoryless



At each time step t,

the only thing that affects x_t

is the current state z_t

$P(x_t = b \mid \text{“whatever happened so far”}) =$

$P(x_t = b \mid z_1, z_2, \ldots, z_t, x_1, x_2, \ldots, x_{t-1}) =$

$P(x_t = b \mid z_t)$

Definition of a hidden Markov model

Definition: A hidden Markov model (HMM)

- Alphabet $\Sigma = \{ b_1, b_2, \ldots, b_M \}$
- Set of states $Q = \{ 1, ..., K \}$
- Transition probabilities between any two states

  a_{ij} = transition prob from state i to state j

  $a_{i1} + \ldots + a_{iK} = 1$, for all states i = 1…K

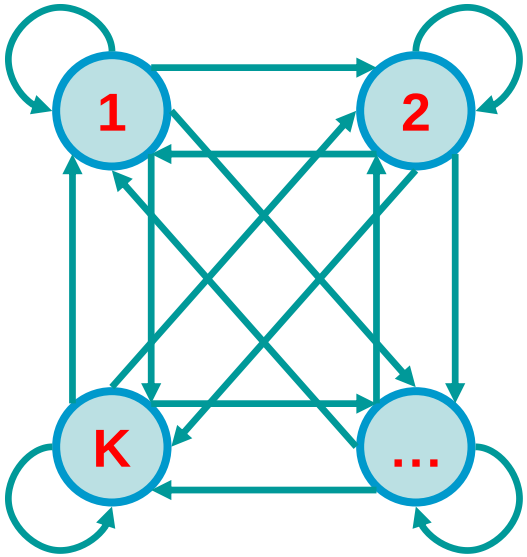
- Start probabilities a_{0i}

  $a_{01} + \ldots + a_{0K} = 1$

- Emission probabilities within each state

  $e_i(b) = P(x_i = b \mid z_i = k)$

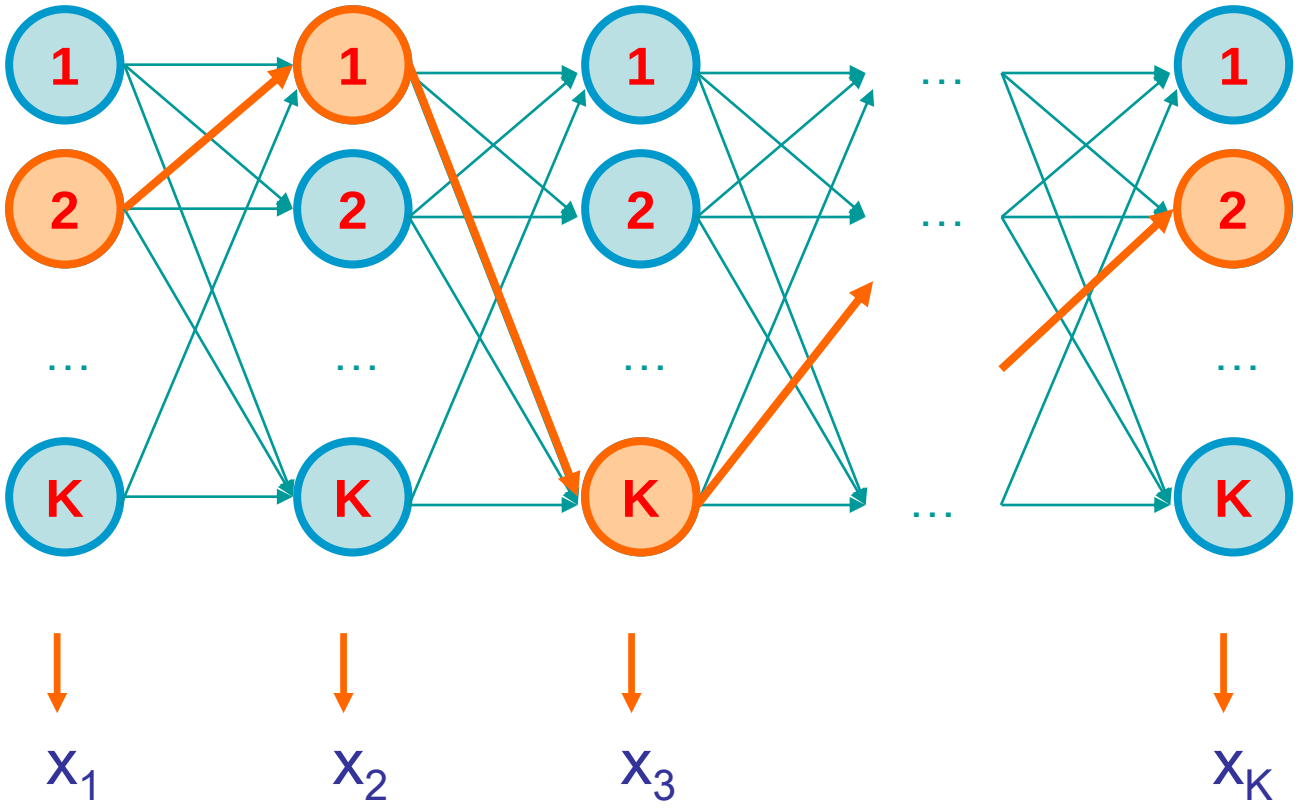
  $e_i(b_1) + \ldots + e_i(b_M) = 1$, for all states i = 1…K



A parse of a sequence

Given a sequence $x = x_1 \ldots\ldots x_N$,

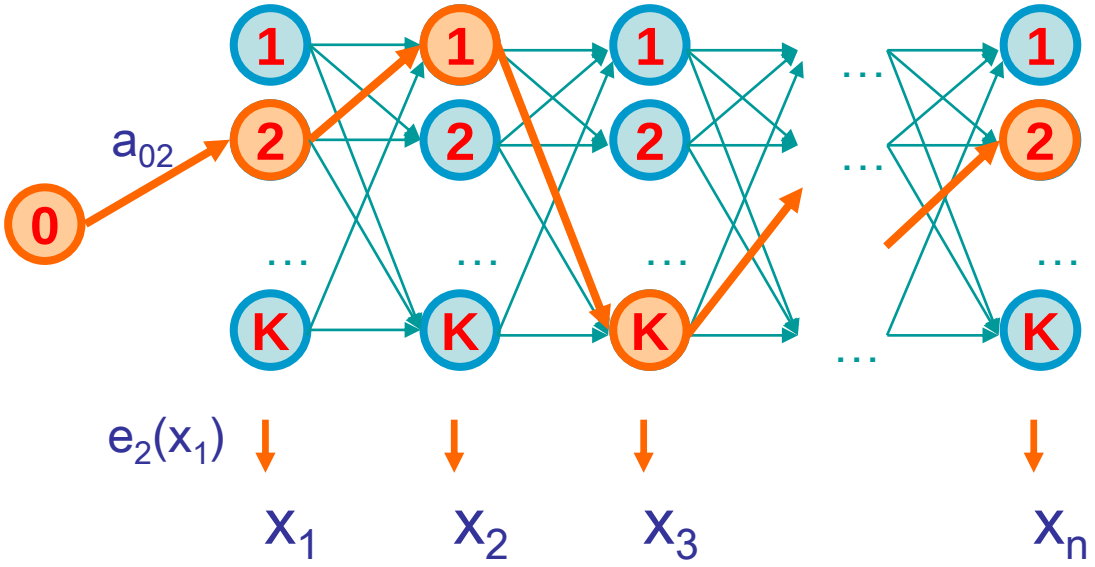
A parse of x is a sequence of states $z = z_1, \ldots\ldots, z_N$



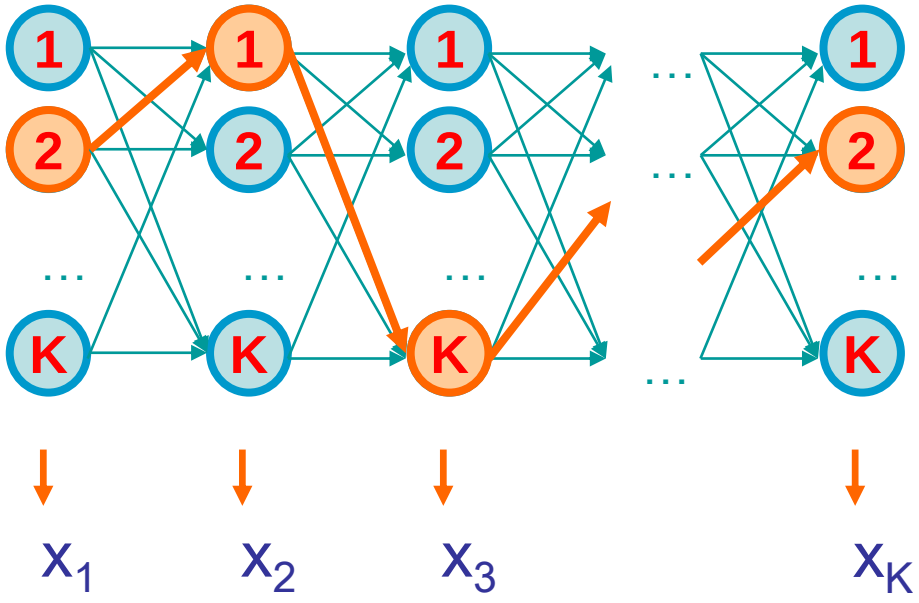
Generating a sequence by the model

Given a HMM, we can generate a sequence of length n as follows:

1. Start at state z_1 according to prob a_{0z1}
2. Emit letter x_1 according to prob $e_{z1}(x_1)$
3. Go to state z_2 according to prob a_{z1z2}
4. … until emitting x_n



Likelihood of a parse

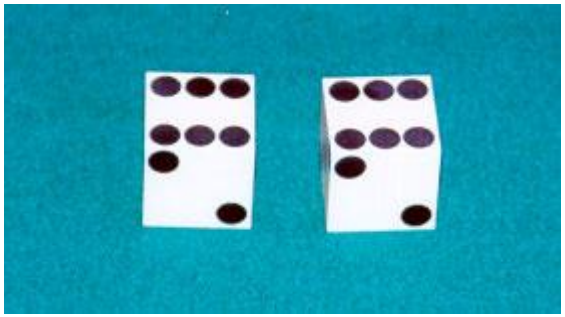


Given a sequence $x = x_1 \ldots\ldots x_N$
and a parse $z = z_1, \ldots\ldots, z_N$,

To find how likely this scenario is:
(given our HMM)

$$P(x, z) = P(x_1, \ldots, x_N, z_1, \ldots\ldots, z_N) =$$
$$P(x_N \mid z_N)\, P(z_N \mid z_{N-1}) \ldots\ldots P(x_2 \mid z_2)\, P(z_2 \mid z_1)\, P(x_1 \mid z_1)\, P(z_1) =$$
$$a_{0z1}\, a_{z1z2} \ldots\ldots a_{zN-1zN}\, e_{z1}(x_1) \ldots\ldots e_{zN}(x_N)$$

Example: the dishonest casino



Let the sequence of rolls be:

x = 1, 2, 1, 5, 6, 2, 1, 5, 2, 4

Then, what is the likelihood of

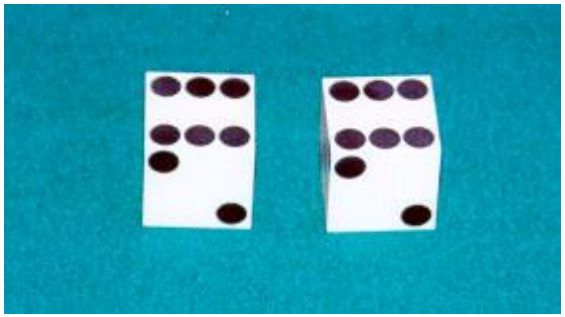
z = Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair?

(say initial probs $a_{0Fair} = ½$, $a_{oLoaded} = ½$)

$½ \times$ P(1 | Fair) P(Fair | Fair) P(2 | Fair) P(Fair | Fair) … P(4 | Fair) =

$½ \times (1/6)^{10} \times (0.95)^{9} = .00000005211586472 11 \sim= 0.5 \times 10^{-9}$

Example: the dishonest casino



So, the likelihood the die is fair in this run is just 0.521×10^{-9}

What is the likelihood of

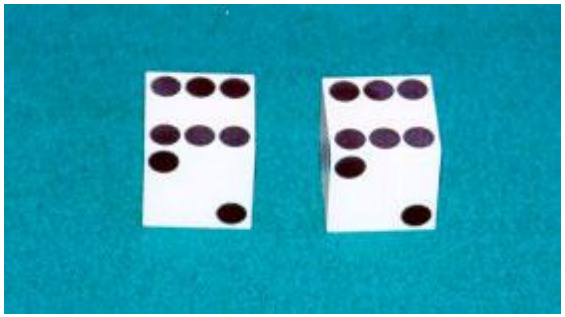
z = Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded?

$\frac{1}{2} \times P(1 \mid \text{Loaded})\, P(\text{Loaded}, \text{Loaded}) \ldots P(4 \mid \text{Loaded}) =$

$\frac{1}{2} \times (1/10)^9 \times (1/2)^1\, (0.95)^9 = .000000000015756235243 \sim= 0.16 \times 10^{-9}$

Therefore, it's somewhat more likely that all the rolls are done with the fair die, than that they are all done with the loaded die

Example: the dishonest casino



Let the sequence of rolls be:

x = 1, 6, 6, 5, 6, 2, 6, 6, 3, 6

Now, what is the likelihood z = F, F, …, F?

$\frac{1}{2} \times (1/6)^{10} \times (0.95)^{9} \sim= 0.5 \times 10^{-9}$, same as before

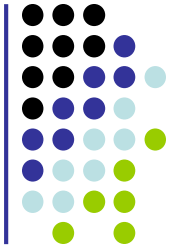
What is the likelihood

z = L, L, …, L?

$\frac{1}{2} \times (1/10)^{4} \times (1/2)^{6} (0.95)^{9} = .00000049238235134735 \sim= 0.5 \times 10^{-7}$

So, it is 100 times more likely the die is loaded

The three main questions on HMMs



1. Decoding

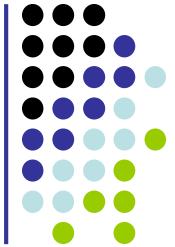
GIVEN a HMM M, and a sequence x,
FIND the sequence z of states that maximizes P[x, z | M]

2. Evaluation

GIVEN a HMM M, and a sequence x,
FIND Prob[x | M]

3. Learning

GIVEN a HMM M, with unspecified transition/emission probs.,
and a sequence x,
FIND parameters $\theta = (e_i(.), a_{ij})$ that maximize P[x | θ]



Problem 1: Decoding

Find the most likely parse of a sequence

Decoding

GIVEN $x = x_1 x_2 \ldots\ldots x_N$

Find $z = z_1, \ldots\ldots, z_N$,
to maximize P[x, z]

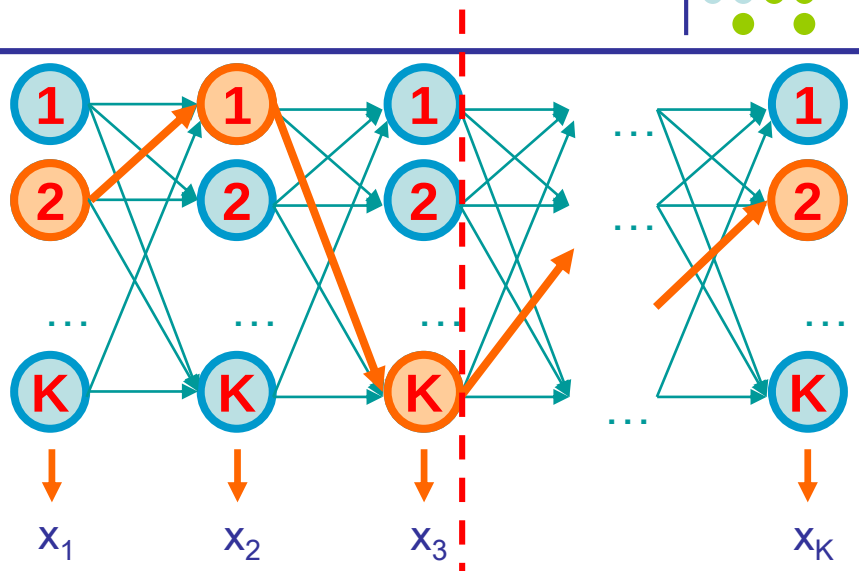
$z^* = \text{argmax}_z P[x, z]$

Maximizes $a_{0z1}\, e_{z1}(x_1)\, a_{z1z2} \ldots\ldots a_{zN-1zN}\, e_{zN}(x_N)$

Dynamic Programming!

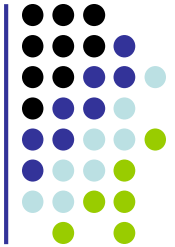
$V_k(i) = \max_{\{z1 \ldots zi-1\}} P[x_1 \ldots x_{i-1}, z_1, \ldots, z_{i-1}, x_i, z_i = k]$

= Prob. of most likely sequence of states ending at state $z_i = k$



Given that we end up in state k at step i, maximize product to the left and right

Decoding – main idea



Induction: Given that for all states k, and for a fixed position i,

$$\mathbf{V_k(i) = \max_{\{z1\ldots\ zi\text{-}1\}} P[x_1 \ldots x_{i-1}, z_1, \ldots, z_{i-1}, x_i, z_i = k]}$$

What is $V_l(i+1)$?

From definition,

$$V_l(i+1) = \max_{\{z1\ldots\ zi\}} P[\ x_1 \ldots x_i, z_1, \ldots, z_i, x_{i+1}, z_{i+1} = l\ ]$$

$$= \max_{\{z1\ldots\ zi\}} P(x_{i+1}, z_{i+1} = l \mid x_1 \ldots x_i, z_1, \ldots, z_i)\ P[x_1 \ldots x_i, z_1, \ldots, z_i]$$

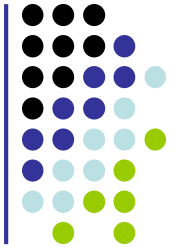
$$= \max_{\{z1\ldots\ zi\}} P(x_{i+1}, z_{i+1} = l \mid z_i)\ P[x_1 \ldots x_{i-1}, z_1, \ldots, z_{i-1}, x_i, z_i]$$

$$= \max_k \left[P(x_{i+1}, z_{i+1} = l \mid z_i = k)\ \mathbf{\max_{\{z1\ldots\ zi\text{-}1\}} P[x_1 \ldots x_{i-1}, z_1, \ldots, z_{i-1}, x_i, z_i = k]} \right]$$

$$= \max_k \left[\ P(x_{i+1} \mid z_{i+1} = l\ )\ P(z_{i+1} = l \mid z_i = k)\ \mathbf{V_k(i)}\ \right]$$

$$= e_l(x_{i+1})\ \max_k a_{kl}\ \mathbf{V_k(i)}$$

The Viterbi Algorithm



Input: $x = x_1 \ldots\ldots x_N$

Initialization:

$V_0(0) = 1$ (0 is the imaginary first position)

$V_k(0) = 0$, for all $k > 0$

Iteration:

$V_j(i) = e_j(x_i) \times \max_k a_{kj} V_k(i-1)$

$Ptr_j(i) = \text{argmax}_k a_{kj} V_k(i-1)$

Termination:

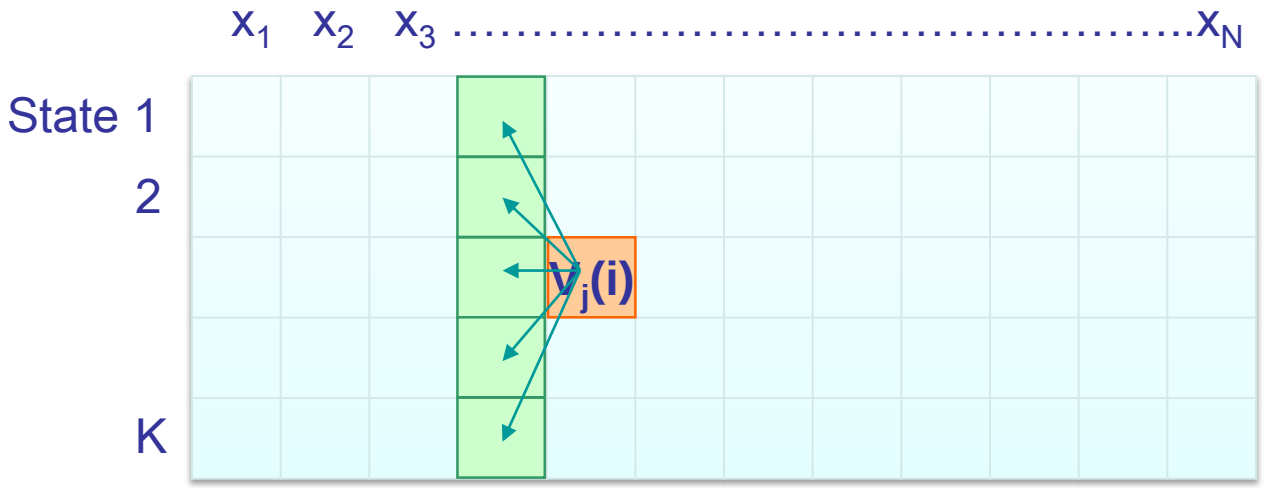
$P(x, z^*) = \max_k V_k(N)$

Traceback:

$z_N^* = \text{argmax}_k V_k(N)$

$z_{i-1}^* = Ptr_{zi}(i)$

The Viterbi Algorithm



Time:

$O(K^2N)$

Space:

$O(KN)$

Viterbi Algorithm – a practical detail

Underflows are a significant problem

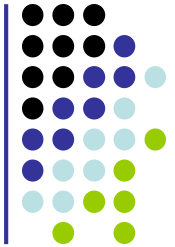
$$P[x_1,\ldots., x_i, z_1, \ldots, z_i] = a_{0z1}\, a_{z1z2}\ldots\ldots a_{zi}\, e_{z1}(x_1)\ldots\ldots e_{zi}(x_i)$$

These numbers become extremely small – underflow

Solution: Take the logs of all values

$$V_l(i) = \log e_k(x_i) + \max_k [V_k(i-1) + \log a_{kl}]$$

Example



Let x be a long sequence with a portion of ~ 1/6 6's,
followed by a portion of ~ ½ 6's…

x = 123456123456…12345 6626364656…1626364656

Then, it is not hard to show that optimal parse is (exercise):

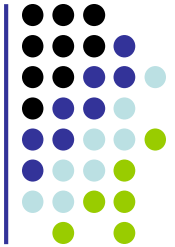
FFF……...F LLL……...L

6 characters "123456" parsed as F, contribute $.95^6 \times (1/6)^6 = 1.6 \times 10^{-5}$

parsed as L, contribute $.95^6 \times (1/2)^1 \times (1/10)^5 = 0.4 \times 10^{-5}$

"162636" parsed as F, contribute $.95^6 \times (1/6)^6 = 1.6 \times 10^{-5}$

parsed as L, contribute $.95^6 \times (1/2)^3 \times (1/10)^3 = 9.0 \times 10^{-5}$



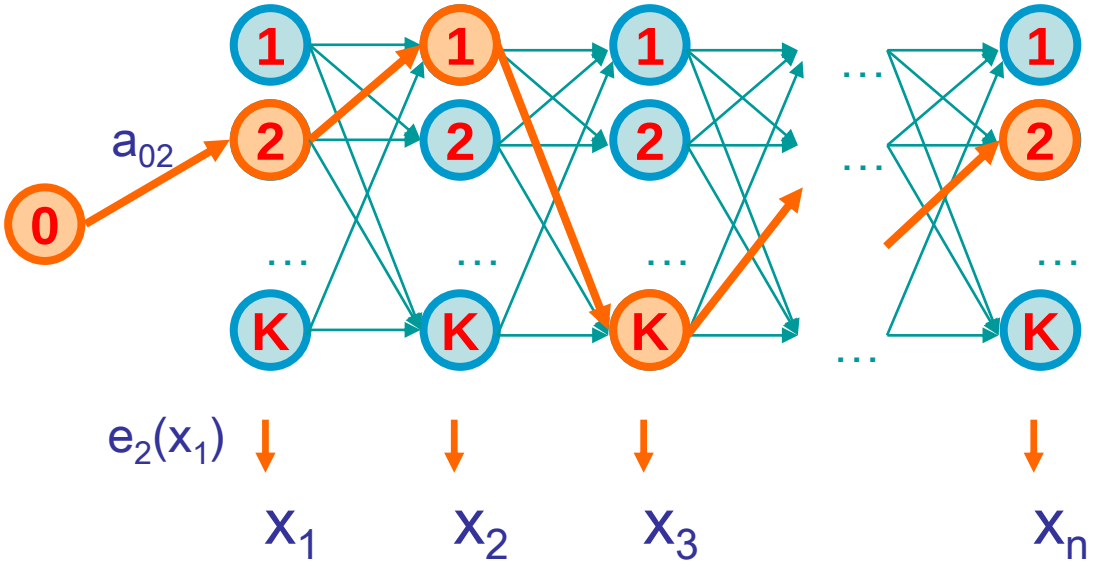
Problem 2: Evaluation

Compute the likelihood that a sequence is generated by the model

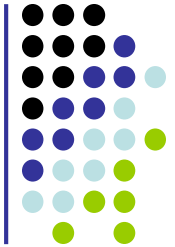
Generating a sequence by the model

Given a HMM, we can generate a sequence of length n as follows:

1. Start at state z_1 according to prob a_{0z1}
2. Emit letter x_1 according to prob $e_{z1}(x_1)$
3. Go to state z_2 according to prob a_{z1z2}
4. … until emitting x_n



A couple of questions



Given a sequence x,

- What is the probability that x was generated by the model?
- Given a position i, what is the most likely state that emitted x_i?

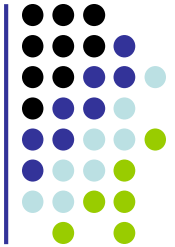
Example: the dishonest casino

Say x = 12341…2316261636461623411 2…21341

F F

Most likely path: z = FF……F
(too "unlikely" to transition F $\rightarrow$ L $\rightarrow$ F)
However: marked letters more likely to be L than unmarked letters

Evaluation

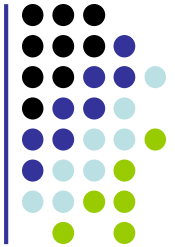


We will develop algorithms that allow us to compute:

$P(x)$	Probability of x given the model
$P(x_i \ldots x_j)$	Probability of a substring of x given the model
$P(z_i = k \mid x)$	"Posterior" probability that the i^{th} state is k, given x

A more refined measure of which states x may be in

The Forward Algorithm



We want to calculate

P(x) = probability of x, given the HMM

Sum over all possible ways of generating x:

$$P(x) = \Sigma_z P(x, z) = \Sigma_z P(x \mid z)\, P(z)$$

To avoid summing over an exponential number of paths z, define

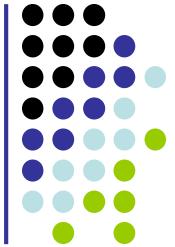
$$f_k(i) = P(x_1 \ldots x_i, z_i = k) \quad \text{(the forward probability)}$$

"generate i first observations and end up in state k"

Relation between Forward and Viterbi

VITERBI	FORWARD
Initialization:	**Initialization:**
$V_0(0) = 1$	$f_0(0) = 1$
Iteration:	**Iteration:**
$V_j(i) = e_j(x_i)\ \max_k V_k(i-1)\ a_{kj}$	$f_l(i) = e_l(x_i) \sum_k f_k(i-1)\ a_{kl}$
Termination:	**Termination:**
$P(x, z^*) = \max_k V_k(N)$	$P(x) = \sum_k f_k(N)$

Motivation for the Backward Algorithm



We want to compute

$$P(z_i = k \mid x),$$

the probability distribution on the i^{th} position, given x

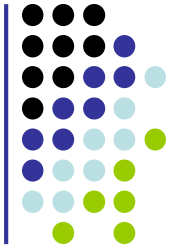
We start by computing

$$P(z_i = k, x) = P(x_1 \ldots x_i, z_i = k, x_{i+1} \ldots x_N)$$
$$= P(x_1 \ldots x_i, z_i = k)\, P(x_{i+1} \ldots x_N \mid x_1 \ldots x_i, z_i = k)$$
$$= P(x_1 \ldots x_i, z_i = k)\, P(x_{i+1} \ldots x_N \mid z_i = k)$$

Forward, $f_k(i)$ Backward, $b_k(i)$

Then, $P(z_i = k \mid x) = P(z_i = k, x) / P(x)$

The Backward Algorithm – derivation



Define the backward probability:

$b_k(i) = P(x_{i+1} \ldots x_N \mid z_i = k)$ *"starting from i^{th} state = k, generate rest of x"*

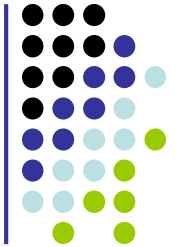
$= \sum_{zi+1 \ldots zN} P(x_{i+1}, x_{i+2}, \ldots, x_N, z_{i+1}, \ldots, z_N \mid z_i = k)$

$= \sum_l \sum_{zi+1 \ldots zN} P(x_{i+1}, x_{i+2}, \ldots, x_N, z_{i+1} = l, z_{i+2}, \ldots, z_N \mid z_i = k)$

$= \sum_l e_l(x_{i+1})\, a_{kl} \sum_{zi+1 \ldots zN} P(x_{i+2}, \ldots, x_N, z_{i+2}, \ldots, z_N \mid z_{i+1} = l)$

$= \sum_l e_l(x_{i+1})\, a_{kl}\, b_l(i+1)$

The Backward Algorithm



We can compute $b_k(i)$ for all k, i, using dynamic programming

Initialization:

$b_k(N) = 1$, for all k

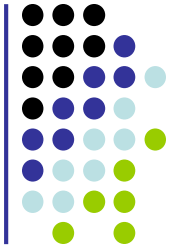
Iteration:

$$b_k(i) = \sum_l e_l(x_{i+1})\, a_{kl}\, b_l(i+1)$$

Termination:

$$P(x) = \sum_l a_{0l}\, e_l(x_1)\, b_l(1)$$

Computational Complexity



What is the running time, and space required, for Forward and Backward?

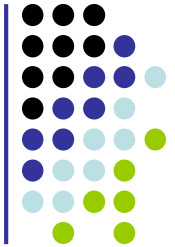
Time: $O(K^2N)$
Space: $O(KN)$

Useful implementation technique to avoid underflows

Viterbi: sum of logs

Forward/Backward: rescaling at each few positions by multiplying by a constant

Posterior Decoding



We can now calculate

$$P(z_i = k \mid x) = \frac{f_k(i)\, b_k(i)}{P(x)}$$

$P(z_i = k \mid x) =$

$P(z_i = k\ , x)/P(x) =$

$P(x_1, \ldots, x_i, z_i = k, x_{i+1}, \ldots x_n) / P(x) =$

$P(x_1, \ldots, x_i, z_i = k)\ P(x_{i+1}, \ldots x_n \mid z_i = k) / P(x) =$

$f_k(i)\ b_k(i) / P(x)$

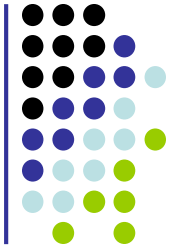
Then, we can ask

What is the most likely state at position i of sequence x:

Define $\hat{z}$ by Posterior Decoding:

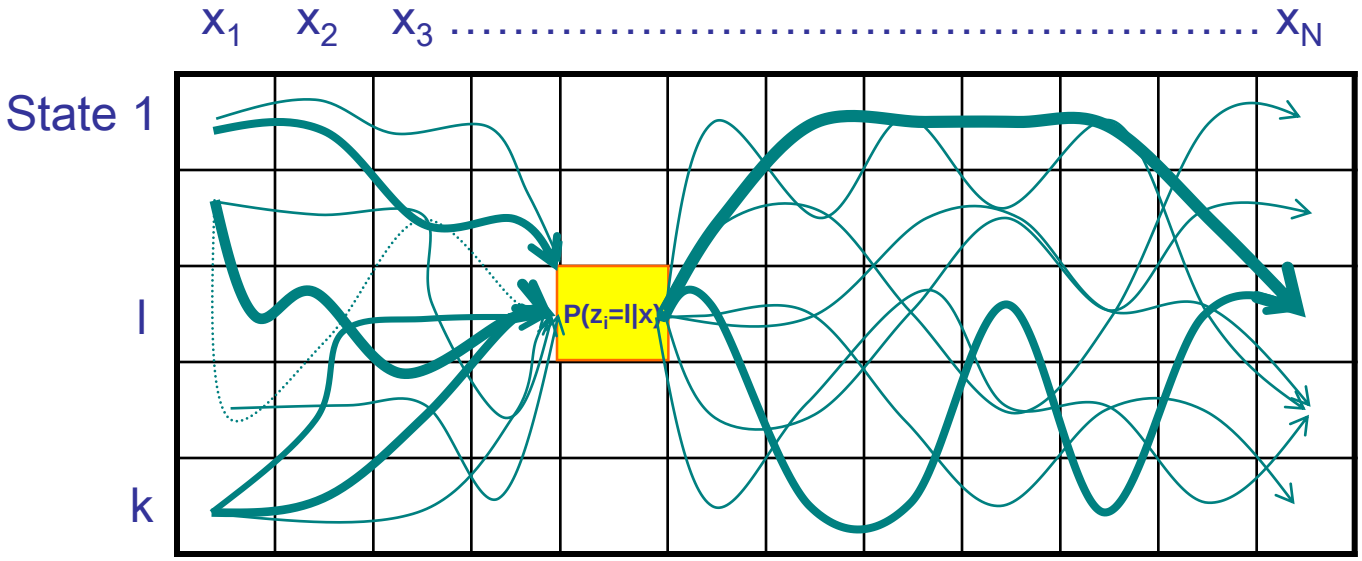
$$\hat{z}_i = \text{argmax}_k\ P(z_i = k \mid x)$$

Posterior Decoding



- For each state,
  - Posterior Decoding gives us a curve of likelihood of state for each position
  - That is sometimes more informative than Viterbi path z^*
- Posterior Decoding may give an invalid sequence of states (of probability 0)
  - Why?

Posterior Decoding

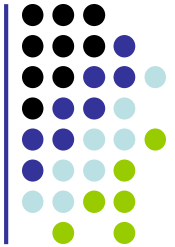


- $P(z_i = k \mid x) = \sum_z P(z \mid x)\, \mathbf{1}(z_i = k)$

$= \sum_{\{z:z[i] = k\}} P(z \mid x)$

$\mathbf{1}(\psi) = 1$, if ψ is true
0, otherwise

Viterbi, Forward, Backward



VITERBI

Initialization:

$V_0(0) = 1$

$V_k(0) = 0$, for all $k > 0$

Iteration:

$V_l(i) = e_l(x_i)\ \max_k V_k(i\text{-}1)\ a_{kl}$

Termination:

$P(x, z^*) = \max_k V_k(N)$

FORWARD

Initialization:

$f_0(0) = 1$

$f_k(0) = 0$, for all $k > 0$

Iteration:

$f_l(i) = e_l(x_i) \sum_k f_k(i\text{-}1)\ a_{kl}$

Termination:

$P(x) = \sum_k f_k(N)$

BACKWARD

Initialization:

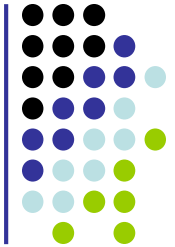
$b_k(N) = 1$, for all k

Iteration:

$b_l(i) = \sum_k e_l(x_i{+}1)\ a_{kl}\ b_k(i{+}1)$

Termination:

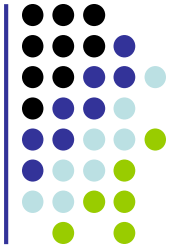
$P(x) = \sum_k a_{0k}\ e_k(x_1)\ b_k(1)$



Problem 3: Learning

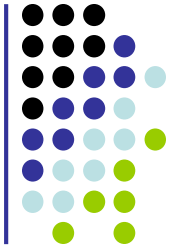
Find the parameters that maximize the likelihood of the observed sequence

Solution: Use the EM algorithm



- Guess initial HMM parameters
- **E step:** Compute distribution over paths
- **M step:** Compute max likelihood parameters
- But how do we do this efficiently?

The forward-backward algorithm



- Also known as the Baum-Welch algorithm
- Compute probability of each state at each position using forward and backward probabilities
  → (Expected) observation counts
- Compute probability of each pair of states at each pair of consecutive positions *i* and *i+1* using *forward(i)* and *backward(i+1)*
  → (Expected) transition counts

$$\text{Count}(k \rightarrow l) = \Sigma_{\mathbf{i}}\, f_k(i)\, a_{kl}\, b_l(i+1) / P(x)$$