12-hmm

Saturday, April 11, 2026 9:06 AM

Last time we discussed Markov chains as models of cell differentiation, where the states were either visible or were an aggregation of visible cells. What if we believe that what we see is actually an imperfect proxy for underlying hidden states?

The Dishonest Casino

(example taken from Manolis Kellis' OCW book)

A casino has two dice

Fair die: $P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = \frac{1}{6}$

Loaded die: $P(1) = P(2) = P(3) = P(4) = P(5) = \frac{1}{10}$

$P(6) = \frac{1}{2}$

Casino player switches back & forth between fair & loaded die once every 20 turns.

Game:

- You bet $1
- You roll fair die
- Casino rolls (maybe fair)
- Highest number wins $2.

Questions:

- Is the casino being fair?
  - easy to tell if casino always cheats
- Can we figure out when the casino is cheating?
- Given a frequency of cheating, what's the probability of winning?

Markov Chain

Consider a Markov Chain (Q, p, A), where

Q is a finite set of states

p is the initial state probabilities

A is the state transition probabilities

$\forall q, r \in Q, \quad a_{qr} \equiv P(x_t = q \mid x_{t-1} = r)$

Key: outputs $x_1, x_2, \ldots$ are the states themselves (observable)

x_i are states chosen at time t

Markov Property: $P(x_t \mid x_{t-1}, \ldots, x_1) = P(x_t \mid x_{t-1})$

$\Rightarrow P(\vec{x}) = P(x_L, x_{L-1}, \ldots, x_1) = P(x_L \mid x_{L-1}) \cdots P(x_2 \mid x_1) P(x_1)$

Hidden Markov Model

5-tuple (Q, V, p, A, E) where

In bioinformatics, states might be things like chromatin accessibility

5-tuple (Q, V, p, A, E) where

Q is a finite set of states

V is a finite set of observable symbols

p is initial state probabilities

A is state transition probabilities

$\forall q, r \in Q, \quad a_{qr} \equiv P(x_t = q \mid x_{t-1} = r)$

E is probability emission matrix

$\forall q \in Q, \; k \in V, \quad e_q(k) \equiv P(o_t = k \mid x_t = q)$

In bioinformatics, states might be things like chromatin accessibility, genomic ancestral haplotypes, membrane accessibility for proteins, etc.

o_i are observed states at time step t

Output: Only emitted symbols are observed by the system but not the underlying hidden random walk between states.

Markov Property: Emissions & transitions are dependent only on current state and nothing else in the past.

Three primary HMM tasks

(1) Scoring/Evaluation. Given a fully specified HMM $M = (Q, V, p, A, E)$ and a sequence of emissions $\vec{o}$, find $\text{Prob}[\vec{o} \mid M]$ or $\text{Prob}[\vec{o}, \vec{x} \mid M]$ for known path.

How good is my model as an explanation of the dice rolls I saw?

Note: the first thing evaluates the model over all possible state paths, not just the optimal.

(2) Decoding. Given a fully specified HMM $M = (Q, V, p, A, E)$, a sequence of emissions $\vec{o}$, find the sequence $\vec{x}$ of states that maximizes $P[\vec{o}, \vec{x} \mid M]$.

When was the casino die fair? When was it loaded?
What portion of $\vec{o}$ was generated by fair die?

(3) Learning Given a partially specified HMM $M = (Q, V, p, ?, ?)$ with unknown transition & emission prob., and a sequence $\vec{o}$, find parameters $\Theta = (A, E)$ that maximize $P[\vec{o} \mid \theta]$.

Expectation Maximization

How loaded is the loaded die? How fair is the fair die?
How often does the casino swap between them?

Note that there are many variations of these three main problems, reflecting both

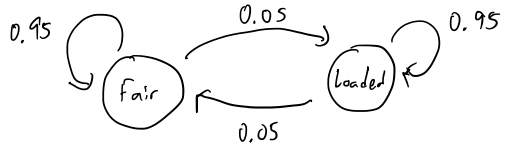
Note that there are many variations of these three main problems, reflecting both our understanding of the system or what we are looking for (e.g. one path vs. all paths)

Scoring

Easiest version: scoring $\vec{o}$ and a single path $\vec{x}$

$P(\vec{o}, \vec{x})$ = prob of a path & emissions

Ex. Dishonest casino



underlying Markov Chain

$P(1|F) = \frac{1}{6}$
$P(2|F) = \frac{1}{6}$
$P(3|F) = \frac{1}{6}$
$P(4|F) = \frac{1}{6}$
$P(5|F) = \frac{1}{6}$
$P(6|F) = \frac{1}{6}$

$P(1|L) = \frac{1}{10}$
$P(2|L) = \frac{1}{10}$
$P(3|L) = \frac{1}{10}$
$P(4|L) = \frac{1}{10}$
$P(5|L) = \frac{1}{10}$
$P(6|L) = \frac{1}{2}$

emissions prob.

$\vec{x} =$ 0.5 → F →(0.95) F →(0.95) F →(0.95) F →(0.95) F →(0.95) F →(0.95) F →(0.95) F →(0.95) F →(0.95) F

$\vec{o} =$ 1 2 1 5 6 2 1 6 2 4

$P(\vec{o}, \vec{x}) = P(\vec{o}|\vec{x}) \cdot P(\vec{x})$, $P(\vec{x}) = \frac{1}{2}$

$= P(1|F) \cdot P(F|F) \cdot P(2|F) \cdot P(F|F) \cdot P(1|F) \cdot P(F|F) \cdots P(4|F)$

$= \frac{1}{2} \cdot \left(\frac{1}{6}\right)^{10} \cdot (0.95)^9 = 5.2 \cdot 10^{-9}$

Notice $P(\vec{o}|\vec{x})$ is tiny for all possible seqs because they are all equally likely.

Alternately, consider $\vec{x} =$ 0.5 → L →(0.95) L →(0.95) L →(0.95) L →(0.95) L →(0.95) L →(0.95) L →(0.95) L →(0.95) L →(0.95) L

$\vec{o} =$ 1 2 1 5 6 2 1 6 2 4

$P(\vec{o}, \vec{x}) = \frac{1}{2} \cdot \left(\frac{1}{10}\right)^8 \cdot \left(\frac{1}{2}\right)^2 \cdot (0.95)^9 = 7.9 \cdot 10^{-10}$ ← lower prob than all fair

Path 3: $\vec{x} =$ 0.5 → F →(0.95) F →(0.95) F →(0.95) F →(0.05) L →(0.95) L →(0.95) L →(0.05) F →(0.95) F

1 2 1 5 6 2 1 6 2 4

$P(\vec{o}, \vec{x}) = 4.6 \cdot 10^{-11}$ ← switching twice extremely unlikely

$P(\vec{o}, \vec{x}) = 4.6 \cdot 10^{-11}$ ← switching twice extremely unlikely

Of these three possibilities, fair dice was most likely.

Alternately, if we saw $\vec{x} = 1\ 6\ 6\ 5\ 6\ 2\ 6\ 6\ 3\ 6$, unfair dice would be a lot more likely.

Exercise: Scoring $\sum_{\vec{x}} P(\vec{o}, \vec{x})$, the sum over all possible paths would be naively exponential. Design a DP algorithm to solve it in polynomial time.

Hint: we are about to see a DP algorithm for decoding.

Decoding: Given $\vec{o}$, what was $\vec{x}$?

Viterbi decoding: $\vec{x}^* = \arg\max_{\vec{x}} P(\vec{o}, \vec{x})$ ← most likely path

We will use the Markov property.

Notice: the best path through a given state can be decomposed into
(1) best path to previous state
(2) best transition from previous state to given state
(3) best path to the end state

Since we use optimal subproblems in computation, can use DP to recursively compute optimal paths.

Let $M_q(t)$ = Probability of most likely path to state $\vec{x}_t = q$

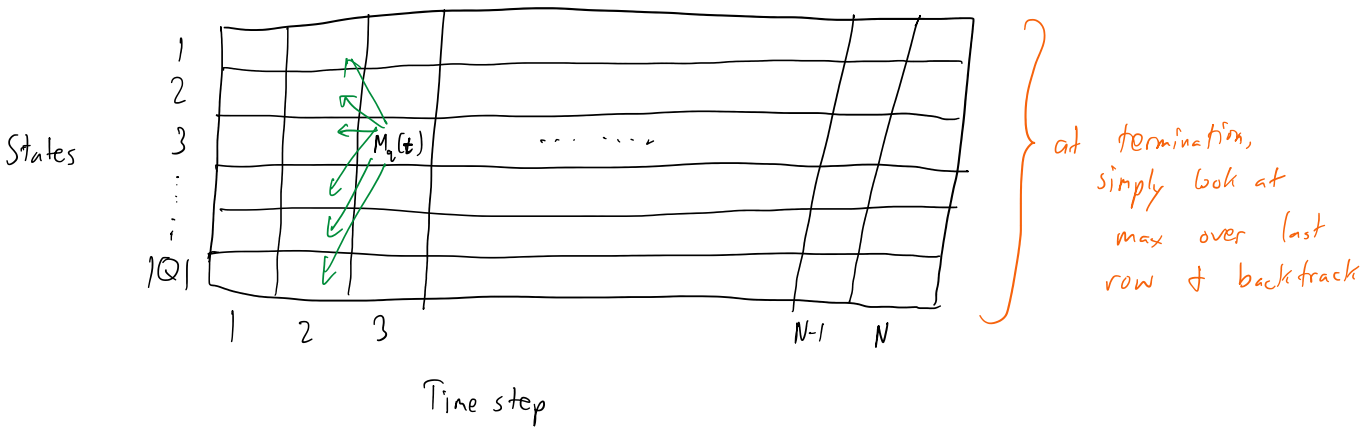
Then $M_q(t+1) = e_q(o_{t+1}) \cdot \max_j a_{jq} M_j(t)$

$e_q(o_{t+1})$: prob of seeing next signal

$\max_j a_{jq} M_j(t)$: optimal transition from all possible previous states

Base case: $M_q(0)$ is starting probabilities of each state. ← for Casino example above, we made it uniform.

We can structure this all in a DP matrix.



And of course we keep track of optimal path so we can backtrack at the end.

Runtime: $O(N|Q|^2)$

Space: $O(N|Q|)$

In practice, use log-scores to prevent floating point errors.

Evaluation: As mentioned in exercise above, we can use a similar DP algorithm to compute $P(\vec{o}\,|M) = \sum_{\vec{x}} P(\vec{o}, \vec{x})$. ← Also called forward algorithm

Learning: This is somewhat beyond scope of this class because it is ML, rather than algorithms.

One way: Baum-Welch training, which is an Expectation-Maximization algorithm.

Input: Fixed HMM structure Q, V.
Some observations $\vec{o}$.

Output: Probability parameters $\theta = (A, E, p)$ that are a local maximum for producing $\vec{o}$.

Intuition: The forward DP algorithm computes as subproblems

$$\alpha_q(t) = P(o_1, o_2, \ldots, o_t, x_t = q \mid \theta)$$ ← probability of seeing first t observations & being in state q at time t given model parameters.

We can also compute backward subproblems

$$\beta_q(t) = P(o_{t+1}, o_{t+2}, \ldots, o_N \mid x_t = q, \theta)$$ ← probability of seeing rest of observations given we are in state q at time t

Together, we can use Bayes' rule to compute the probabilities

$$P(x_t = q, \vec{o}\,|\theta) \qquad \alpha_q(t)\beta_q(t)$$

rest of here

Together, we can use Bayes' rule to compute the probabilities

prob of being in state k at time t → $\gamma_q(t) = P(x_t = q \mid \vec{o}, \theta) = \frac{P(x_t = q, \vec{o} \mid \theta)}{P(\vec{o} \mid \theta)} = \frac{\alpha_q(t)\beta_q(t)}{\sum_{j=1}^{|Q|} \alpha_j(t)\beta_j(t)}$

and $\xi_{qr}(t) = P(x_t = q, x_{t+1} = r \mid \vec{o}, \theta)$, ← prob of transition from q to r at time t

Now set

$$p_q^* = \gamma_q(1)$$

$$a_{qr}^* = \frac{\sum_t \xi_{qr}(t)}{\sum_t \gamma_q(t)}$$

$$e_q^*(k) = \frac{\sum_t \mathbb{1}_{o_t = k}\, \gamma_q(t)}{\sum_t \gamma_q(t)}$$

} maximum likelihood parameters are just ones where expected fraction is observed.

Repeat process with new parameters until convergence.

Note that this only gives a local maximum.

Exercise? Can we do something similar with just a single optimum path for a given set of parameters, using the Viterbi alg above?

Aside: Can also simultaneously fit multiple observations.

Protein modelling & MSA

How to fold a protein?

One way: use physics! The sequence gives all covalent bonds in the protein molecule, and we can model the atomic forces that twist the protein. This technique is known as molecular dynamics, & is nice in theory. Unfortunately, proteins fold on long timescales, sometimes on the order of milliseconds, & we need custom-built supercomputers to run MD for complicated proteins.

(MD can also be used to understand dynamics of already folded proteins)

Alternative: use what we know of already solved protein structures to help predict new structures.

to help predict new structures.

Perhaps: throw all known structures into a giant neural network. In theory would work with enough data, but solved structures can be hard to come by.

AlphaFold: first identify similar proteins / families to our sequence & then use their structures to guide us.

How to identify similar proteins?

Sequence alignment is one way to find similar proteins.
But structurally similar proteins might have completely different sequences, so this misses out on many structures.

Profile HMMs (used to create e.g. protein families/clans in Pfam)

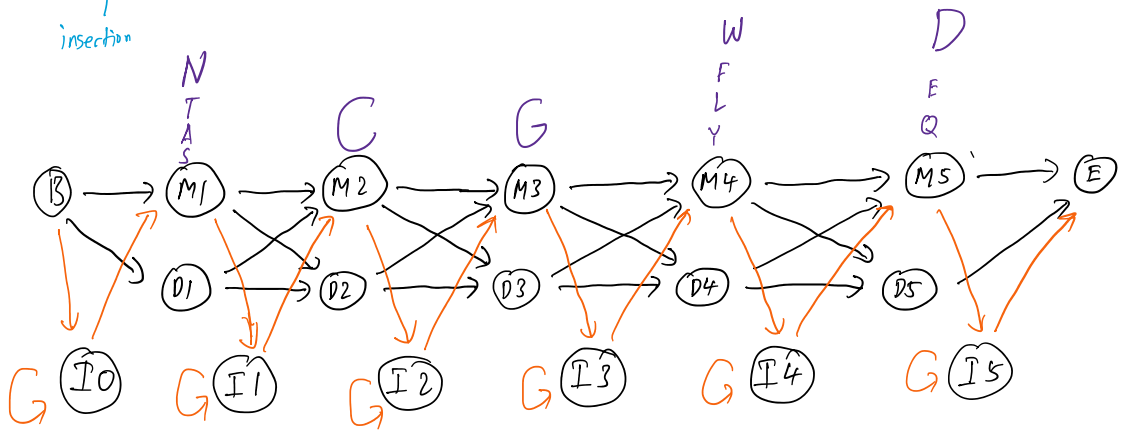
Idea: create a position-dependent HMM to capture entire protein family. Then, we can test if a protein comes from this family using HMM.

Start from MSA (example from EBI Pfam training course)

seq 1	A	C	G	-	L	D
seq 2	S	C	G	-	-	E
seq 3	N	C	G	G	F	D
seq 4	T	C	G	-	W	Q
	1	2	3		4	5

↑ insertion

Each position can have a match, insert, or delete state, & the emissions depend on values in alignment.



Constructing these profile HMMs is expensive, so we might iteratively add or remove members of the family as we converge on something.

Once built, these profile HMMs are fast to score new proteins for remote homology

Once built, these profile HMMs are fast to score new proteins for remote homology.