Hidden Markov Models

Douglas Rubin
7/6/18

- Markov Models motivation
- Learning in Markov Models
- Dynamic Programming

Markov Models Motivation

- In ML, regression, classification, clustering etc... good when the order of data generation is not important.
- But, when we believe that the data are generated in a very sequential manner, a Markov Model might be appropriate.

Toy example for a MM: 3 possible states of the world
{sunny, cloudy, rainy}

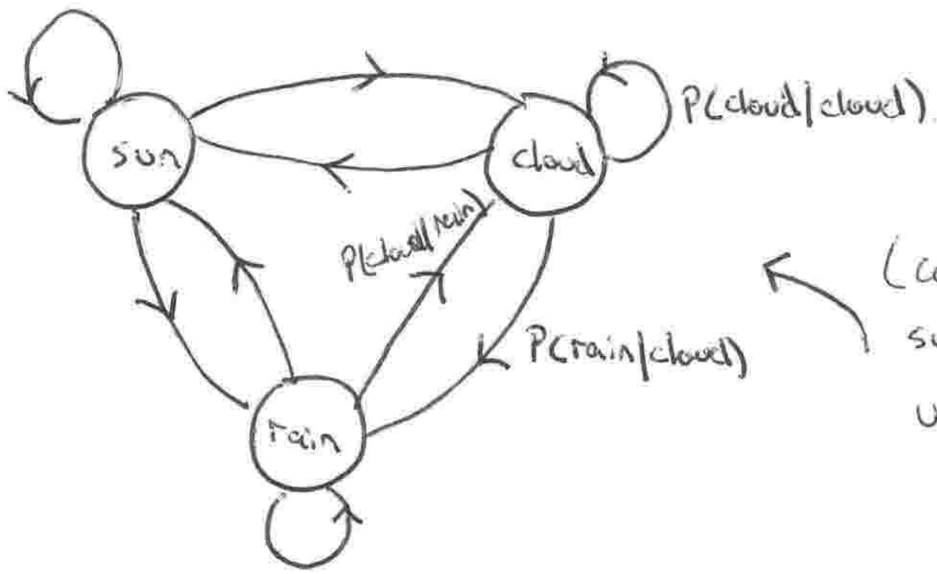
- we observe the sequence sun, sun, rain, sun, cloudy, cloudy, rain
- To model this probabilistically, we wish to learn

$$P(State_{t+1} \mid State_t, State_{t-1}, \ldots, State_1)$$

so that we can predict the future given the past. ②

<u>MM</u> <u>assumption</u>: the current state of the system encodes "enough" info about the past to predict the future. Mathematically, given the present, the future is conditionally independent of the past:

$$P(\text{State}_{t+1} \mid \text{State}_t, \text{State}_{t-1}, \dots \text{State}_1) = P(\text{State}_{t+1} \mid \text{State}_t)$$



(complete probabilistic summary of the system under MM assumption)

<u>Applications of MMs</u>

<u>e.g.</u> time series data

- weather, finance, language, music

e.g. spatially sequential data

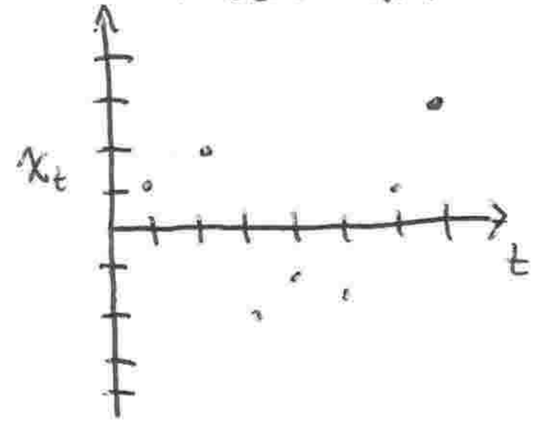
- DNA sequences, written language

Types of MMs:

- discrete time - discrete space

  eg: previous example, random walk

random walk



- disrete time - continuous space

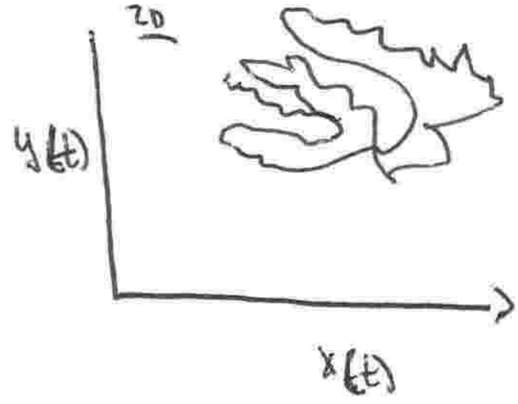
  eg: various time series models such as AR

- continuous time - discrete space

  eg: poisson process

- continuous time - continuous space

  eg: Brownian motion



Learning in MMs

★ note: we will consider discrete time - discrete space here

★ note: I will use the notation $P(x) := P(X = x)$, but will sometimes explicitly use the latter notation for clarity

Setup

- Let $S = \{s_1, s_2, \ldots, s_{|S|}\}$ be the possible states of the system (eg. {sunny, cloudy, rainy}).

- Let Z be a rv s.t. $Z(s_1) = 1$, $Z(s_2) = 2$, etc...

- Let $\vec{z} = (z_1, z_2, \ldots z_T) \in \{1, 2, \ldots, |S|\}^T$ be our observed sequence of data ($\vec{z}$ is a T dim. vector)

  eg: sun, sun, cloud, rain $\longrightarrow$ $\vec{z} = (1, 1, 2, 1)$.

. The observed sequence is just 1 realization of a random process over time

- Since we are trying to construct a probabilistic model, we model this process w/ PMFs, whose parameters we will learn from the data

PMF assumptions / simplifications

① Markov Property:

$$P(z_t \mid z_{t-1}, z_{t-2}, \dots, z_1) = P(z_t \mid z_{t-1})$$

$$\forall\ t = 2, 3, \dots, T$$

② Stationarity: PMF doesn't change over time

$$P(z_t \mid z_{t-1}) = P(z_2 \mid z_1) \quad \forall\ t = 3, 4, \dots, T$$

Our generative Model

We can fully parameterize this PMF with a matrix, $A \in \mathbb{R}^{|S| \times |S|}$ called the state transition matrix:

$\cancel{P(z_t \mid z_{t-1})}$

$$P(z_t = j \mid z_{t-1} = i; A) = A_{ij}$$ = prob. of transitioning from i to j.

eg: For the weather sequence system, perhaps

$A =$

	1(sun)	2(cloud)	3(rain)
(sun) 1	.8	.1	.1
(cloud) 2	.1	.8	.1
(rain) 3	.1	.2	.7

- note the strong diagonal (weather is self-correlated)
- note that: $A_{ij} \geq 0 \ \forall i,j$; $\sum_{j=1}^{|S|} A_{ij} = 1 \ \forall i$

Our goal is to learn A from the data

<u>initial PMF</u>

- A gives $P(Z_t | Z_{t-1}) \ \forall \ t = 2, 3, \ldots T$, but how do we model the start of this sequence, Z_1?
- We parameterize this with a vector, $\pi \in \mathbb{R}^{|S|}$, of probabilities:

$$P(Z_1 = i; \pi) = \pi_i$$

- note $\pi_i \geq 0 \ \forall i$; $\sum_i \pi_i = 1$

Our goal is to also learn $\cancel{A}$ π;

however, in the following, I focus on learning A, and assume that π is known/given (say, a uniform dist.).

- Note that $P(Z_1; \pi)$ is $Cat(\pi)$ and $P(Z_t = j | Z_{t-1} = i; A)$ is $cat(A_{i,*})$

Thus, our <u>generative model</u> is:

$$Z_1 \sim cat(\pi)$$

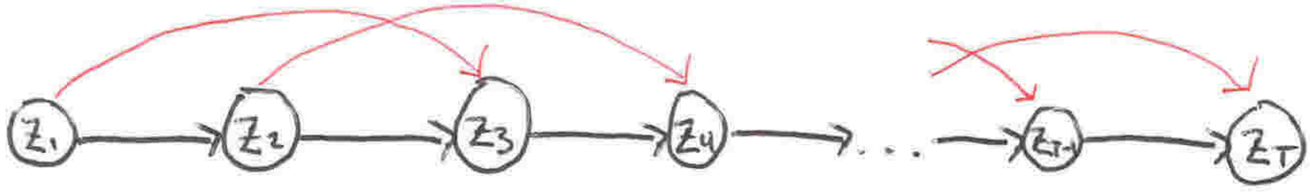
$$Z_2 | Z_1 = i \sim cat(A_{i,*})$$

$$\vdots$$

- this PMF can be encoded in a <u>graphical model</u>, which shows the joint PMF's conditional dependencies (with a directed arrow). This looks like a chain and is where Markov Chain gets its name from

— : 1^{st} order MM

— + — : 2^{nd} order MM



MLE learning

Once we know A, π, we can compute many interesting ~~___~~ things

eg:
- Prob., $P(Z)$, of a particular sequence
- mean hitting times
- mean return times
- Stationary distribution (Google's pagerank)

likelihood function

(joint PMF)

$$P(Z_1; A) = P(Z_1, \ldots, Z_t; A)$$

$$\overset{\text{chain rule}}{=} P(Z_1) P(Z_2 | Z_1) P(Z_3 | Z_2, Z_1) \ldots P(Z_t | Z_{t-1}, \ldots, Z_1)$$

$$\overset{\text{markov prop.}}{=} P(Z_1; \pi) P(Z_2 | Z_1; A) \ldots P(Z_T | Z_{T-1}; A)$$

$$= \pi(Z_1) \prod_{t=2}^{T} A_{Z_{t-1}, Z_t}$$

$\Rightarrow$ log-likelihood:

$$\ell(A) = \log\left(P(z; A)\right)$$
$$= \log \prod_{t=2}^{T} A_{z_{t-1}, z_t} + \log \pi(z_1)$$
$$= \sum_{t=2}^{T} \log A_{z_{t-1}, z_t} + \log \pi(z_1)$$
$$= \sum_{i=1}^{|S|} \sum_{j=1}^{|S|} \sum_{t=2}^{T} \mathbb{1}\{z_{t-1} = i \wedge z_t = j\} \log A_{ij} + \log \pi(z_1)$$

(indicator function introduced b/c I'll need to take derivatives wrt A_{ij}).

- The argmax is our MLE of A:

$$\hat{A} = \underset{A}{\operatorname{argmax}} \{\ell(A)\}$$

$$\text{s.t.} \sum_{j=1}^{|S|} A_{ij} = 1 \qquad \forall\, i = 1, \ldots, |S|$$

$$A_{ij} \geq 0 \qquad \forall\, i, j = 1, \ldots, |S|$$

$\rightarrow$

- equality **and** inequality constraints $\Rightarrow$ must use Lagrange Duality. However, if we ignore inequalities and use Method of Lagrange Multipliers, all $A_{ij} \geq 0$ anyway.

$\Rightarrow$ goal is to find $\hat{A}, \hat{\alpha}$ which minimize Lagrangian:

$$\mathcal{L}(A, \alpha) = \sum_{i=1}^{|S|} \sum_{j=1}^{|S|} \sum_{t=2}^{T} \mathbb{1}\{z_{t-1} = i \wedge z_t = j\} \log A_{ij} + \log \pi(z_1)$$
$$+ \sum_{i=1}^{|S|} \alpha_i \left(1 - \sum_{j=1}^{|S|} A_{ij}\right) \quad ; \quad (\alpha \in \mathbb{R}^{|S|} \text{ are the lagrange multipliers})$$

$\Rightarrow$ solve for $\hat{A}, \hat{\alpha}$ w/ the following eqns. (i.e., solve by setting partial derivatives equal to 0).

$$\begin{cases} \nabla_A \mathcal{L}(\hat{A}, \hat{\alpha}) = 0 & \leftarrow |S| \times |S| \text{ matrix of 0s} \\ \nabla_\alpha \mathcal{L}(\hat{A}, \hat{\alpha}) = 0 & \leftarrow |S| \times 1 \text{ vector of 0s} \end{cases}$$

$$\Rightarrow \boxed{\hat{A}_{ij} = \frac{\sum_{t=2}^{T} \mathbb{1}\{z_{t-1} = i \wedge z_t = j\}}{\sum_{t=2}^{T} \mathbb{1}\{z_{t-1} = i\}}}$$

As is typical w/ ML estimates, this formula is very intuitive: $\hat{A}_{ij}$ = MLE of $P(Z_t = j \mid Z_{t-1} = i)$ = the fraction of observed data that started in j and transiting to i 1 time step later.

* Note: in practice we may want to employ Laplace smoothing.

EM Algo (Expectation-Maximization)

- Brief review
- see previous notes for math

- In general ML estimates for generative models w/ hidden (latent variables), like Hidden MM is not analytically tractable.

- In many cases, if the dists. used are from exponential family, we can employ a numerical algo the estimate the MLEs.

~>

EM Algo

Input: Data $\{x^{(1)}, \ldots, x^{(m)}\}$, parameterized conditional, and joint PMFs: $P(Z^{(i)} | x^{(i)}; \Theta)$, $P(x^{(i)}, Z^{(i)}; \Theta)$, w/ Θ: set of all parameters (matrices, vectors, ...)

Output: Θ^* the MLE approximation from the algo.

① initialize Θ^*

② repeat till convergence {

(e-step) 2a) compute $Q_i(Z^{(i)}) := P(Z^{(i)} | x^{(i)}; \Theta^*)$ $\forall\, i = 1, \ldots, m$
$\forall$ values of $Z^{(i)}$

(m-step) 2b) $$\Theta^* := \underset{\Theta}{\operatorname{argmax}} \left\{ \sum_i \sum_{Z^{(i)}} Q_i(Z^{(i)}) \log \frac{P(x^{(i)}, Z^{(i)}; \Theta)}{Q_i(Z^{(i)})} \right\}$$

}

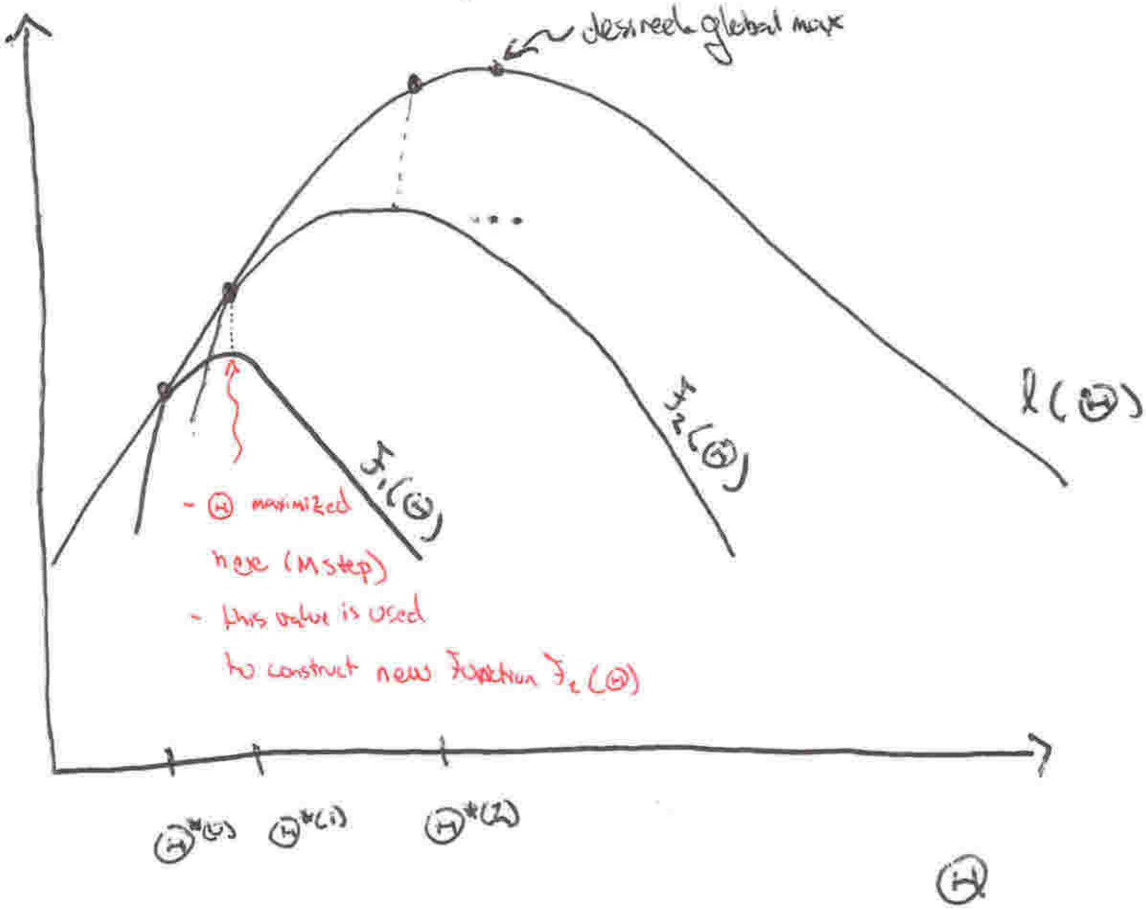
- works b/c the optimization in 2b is typically much easier than the original optimization.

- for HMMs we will use a clever dynamic programming subroutine for 2b)

Why this works

- define function in argmax as $F(\Theta)$.
- By construction $F(\Theta)$ is upperbounded by $l(\Theta)$, w/ tight equality at current value of Θ^*.

1D example



Dynamic Programming (DP)

- Algo technique widely used by optimization / constrained optimization over a finite set.

- typically turns an exponential time complexity problem into a polynomial time algo.

- Recursion + Memoization

Recursion on a Computer

e.g. Fibonacci numbers

$$0, 1, 1, 2, 3, 5, 8, \ldots$$

$$F_n = \begin{cases} 0 & \text{for } n = 0 \\ 1 & \text{for } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- general structure of a recursive program:

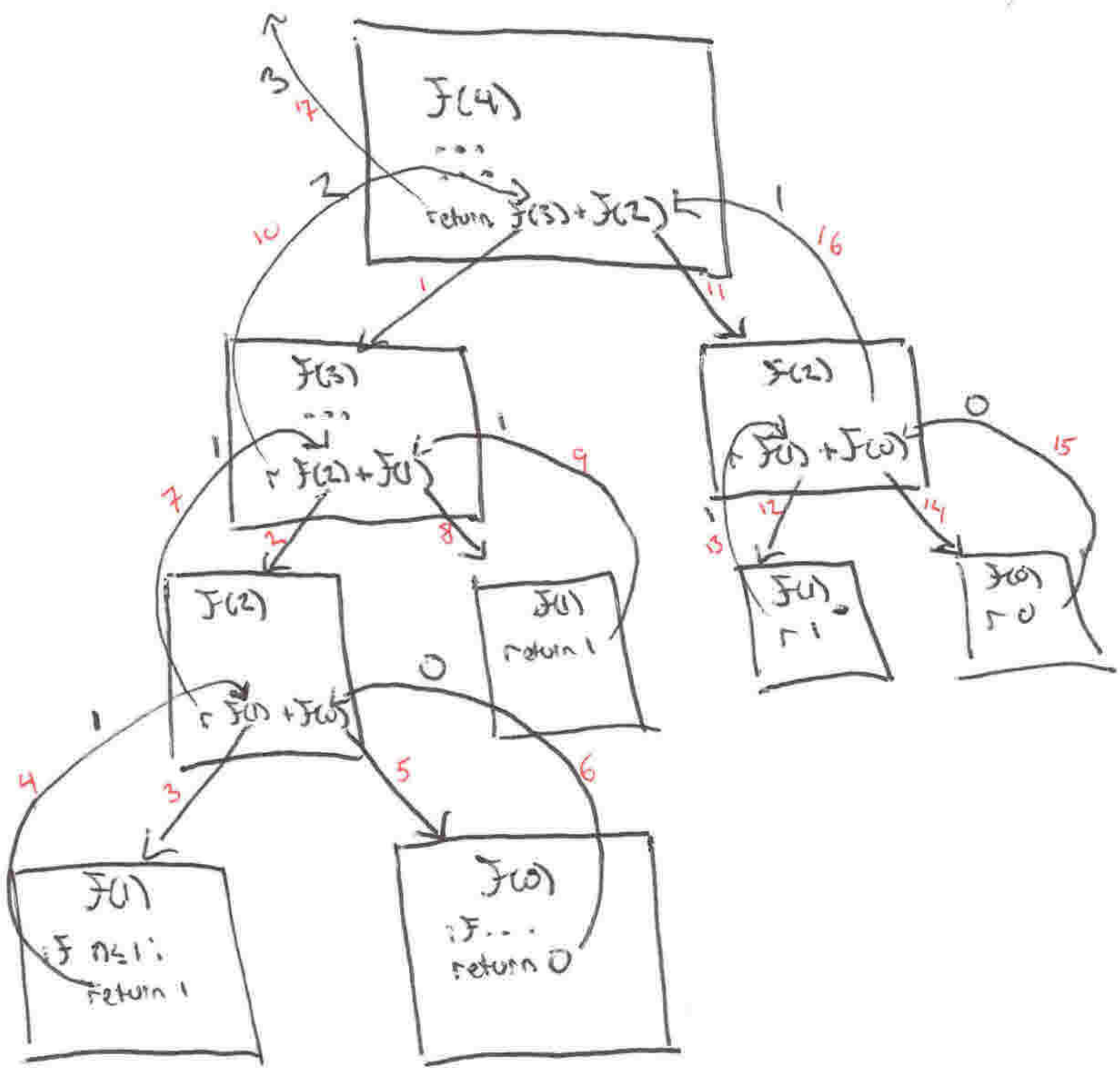
  ① check if in base case
    - if so, return base

  ② if not
    - return recursive calls

Recursive Fibonacci Algo

```
F(n)
    if n ≤ 1:          // check base
        return n

    return F(n-1) + F(n-2)     // return recursive calls
```

how is a recursive algo executed on a computer?



- red #s indicate order of execution

- when a function calls itself recursively, it essentially gets "paused" and put on a call stack. when execution returns to that function, it gets "unpaused" and taken off the call stack.

time complexity for Fib. algo

$$T(n) = \begin{cases} T(n-1) + T(n-2) + \Theta(1) & n > 1 \\ \Theta(1) & n=0 \\ \Theta(1) & n=1 \end{cases}$$

work from comparison, addition, etc...

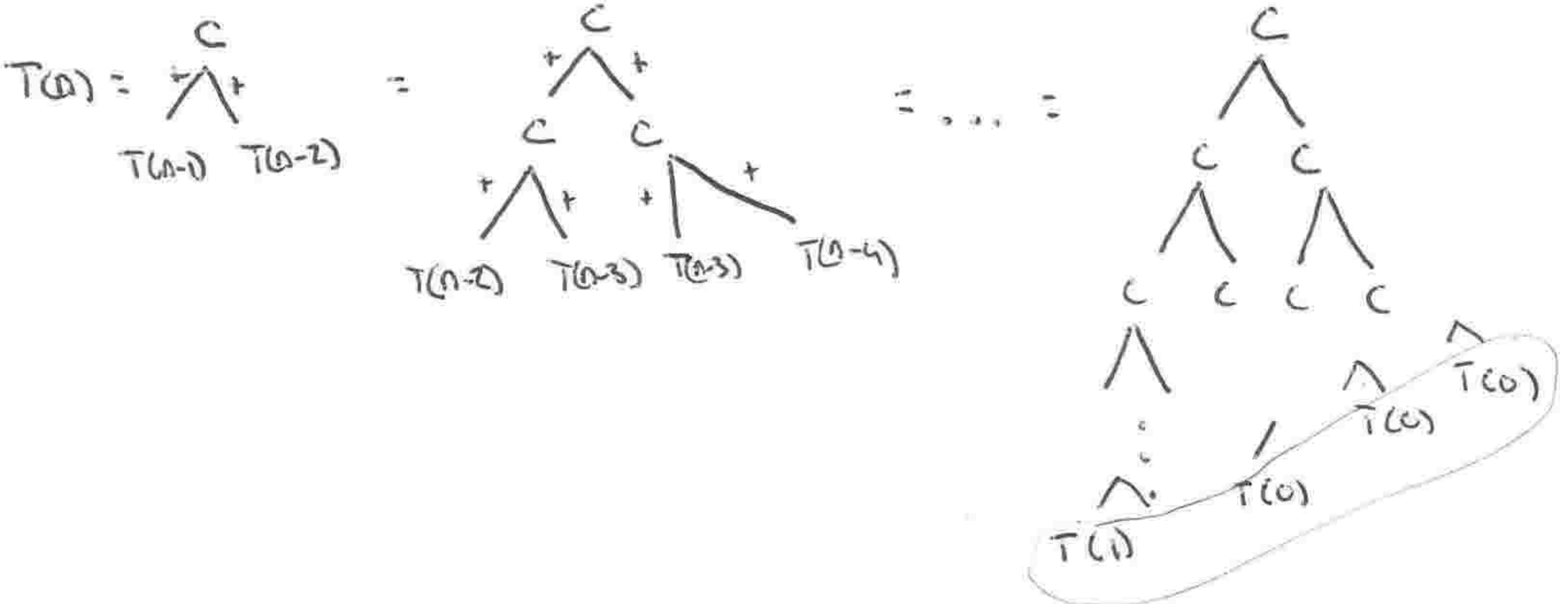
- we solve for $T(n)$ by "unrolling" it:

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + c \\ &= \{T(n-2) + T(n-3) + c\} + \{T(n-3) + T(n-4) + c\} + c \\ &= T(n-2) + T(n-3) + T(n-3) + T(n-4) + c + c + c \end{aligned}$$

$$\ldots$$

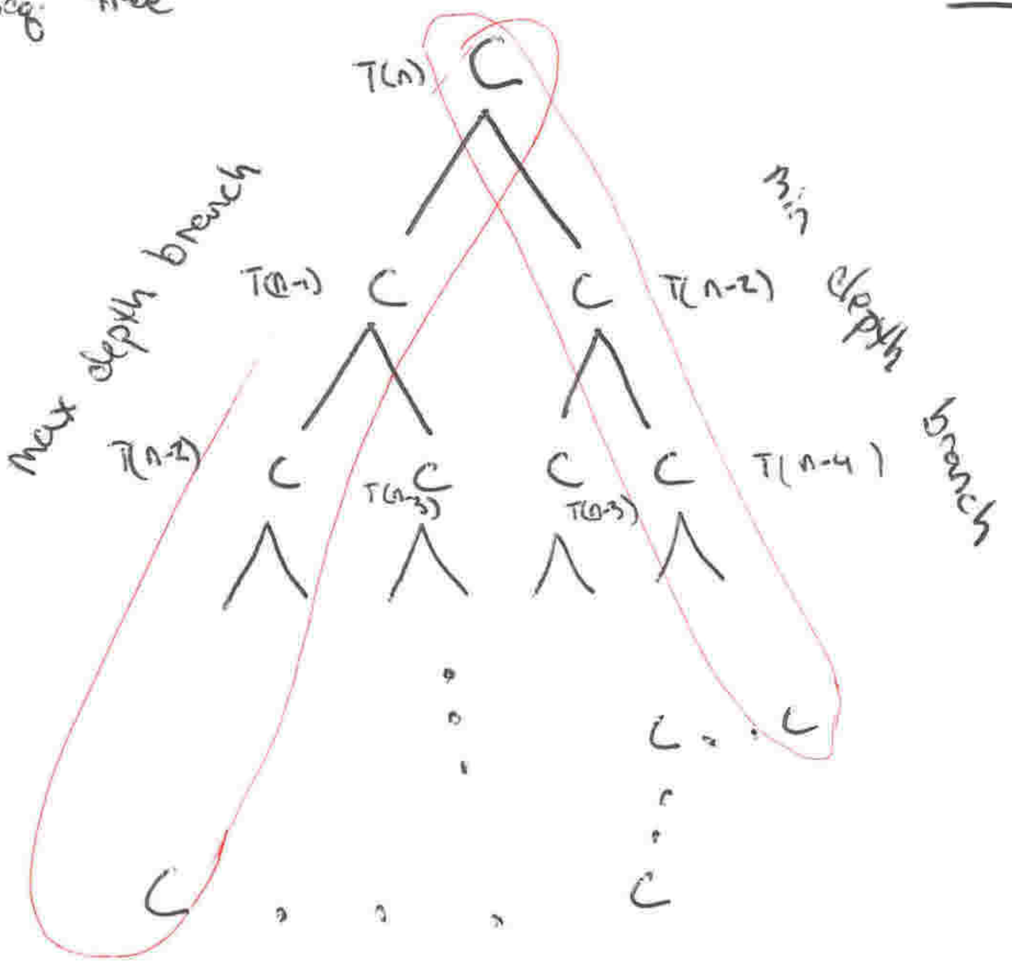
$$c + c + \ldots + c$$

- unrolling quickly gets unweildy, so we visually organize this using a "recursion tree" technique, then add up level-by-level in the tree.



tree bottoms out at base cases $(T(1) = T(0) = c)$

Fib. seq. tree



work/level

$c \cdot 2^0$

$c \cdot 2^1$

$c \cdot 2^2$

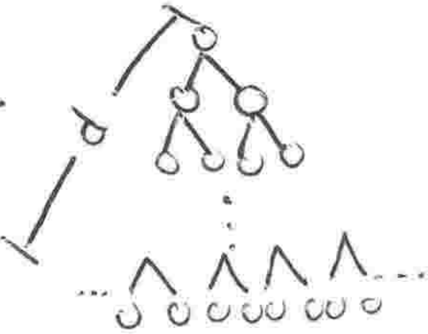
$\vdots$

$c \cdot 2^k$

$\vdots$

- Can bound $T(n)$ from above and below.

- For a <u>full</u> binary tree



$$T(n) = c(2^0 + 2^1 + \dots + 2^d) = c\sum_{k=0}^{d} 2^k = c\left(\frac{1-2^{d+1}}{1-2}\right)$$ (geometric series)

- For Fib, max depth $= n$; min depth $= n/2$

$$\Rightarrow T(n) \leq c\sum_{k=0}^{n} 2^k = c2^{n+1} - c \Rightarrow T(n) = O(2^n)$$

$$\Rightarrow T(n) \geq c\sum_{k=0}^{n/2} 2^k = c2^{n/2+1} - c \Rightarrow T(n) = \Omega(2^{n/2})$$

$$= \Omega(\sqrt{2}^n)$$

$$= \Omega(1.41^n)$$

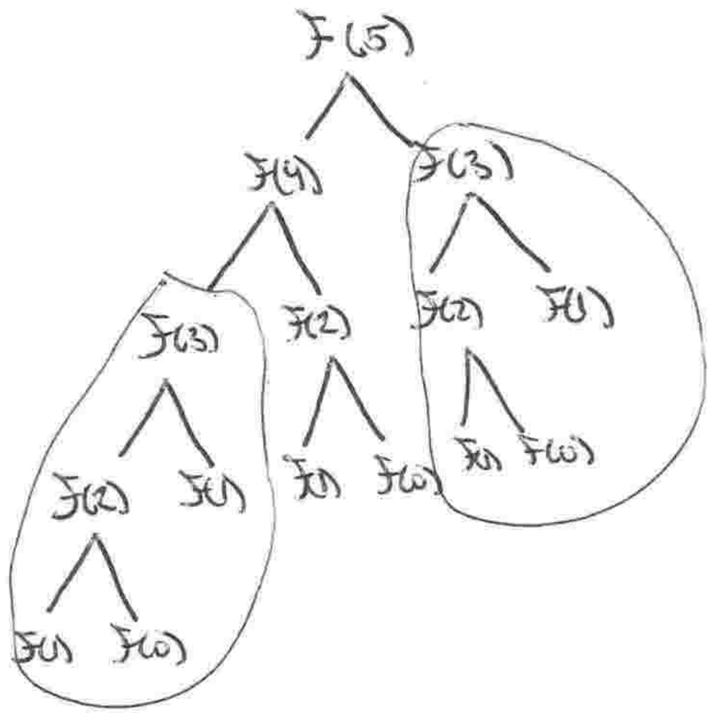
$\Rightarrow$ running time is bounded between 2^n and 1.41^n

<u>Exponential</u> running time (bad!)

* Note a careful analysis for Fib. shows that $T(n) = \Theta(\phi^n)$

where $\phi = \frac{1+\sqrt{5}}{2}$ = golden ratio

Why is time complexity so bad?



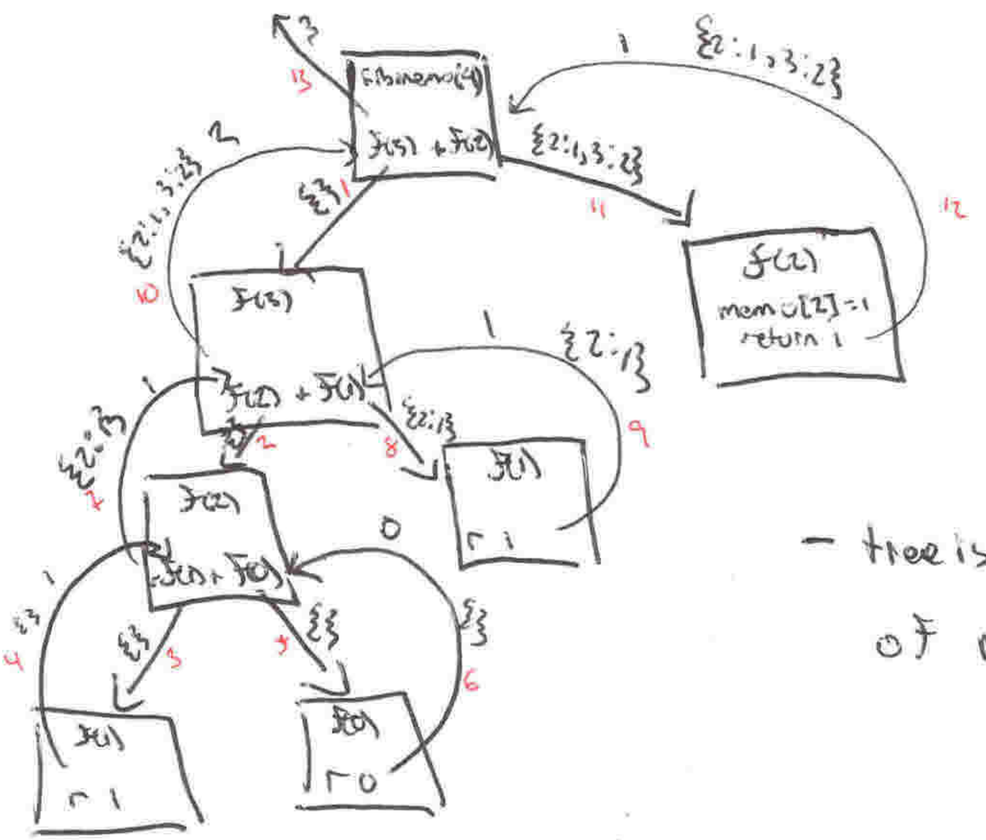
- Many repeated calculations

<u>Solution</u>: memoization

- once a new value of F(·) is computed, cache it
- if we ever need it again, instead of computing a big subtree over again, we look it up in O(1) time.

- only a few lines need be added to code to get a <u>huge</u> savings in cost.
- memory / time trade-off.

```
memo = {}
FibMemo(n):

    if n in memo:                         // check cache
        return memo[n]

    if n ≤ 1:                             // base
        return n

    F = FibMemo(n-1) + FibMemo(n-2)

    memo[n] = F                           // memoize

    return F
```



- tree is smaller b/c of memoization

Time complexity of memoized / DP Algo

- each ~~items~~ unique sub-problem only solved explicitly once before being memoized

$$\Rightarrow T(n) = \frac{\text{work}}{\text{unique subproblem}} \times (\#\text{ unique subproblems})$$

- in FibMemo(n), F(2), F(3), ..., F(n) are all unique subproblems ∴

$$T(n) = n \cdot \Theta(1) = \Theta(n)$$

- linear time vs. exponential ⇒ (vast improvement w/ ~ 4 lines new code)

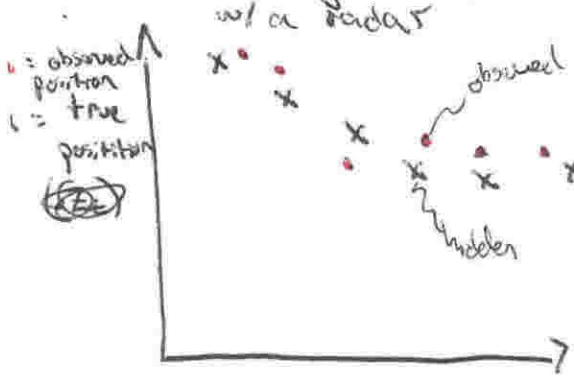
- HMMs
- Forward/backward Algo
- Viterbi Algo
- EM Algo for HMMs.

HMMs

- Markov models were good for sequential data when we got to observe all data in the data generating process

- However, in many instances (sequential) RVs are generated that we do not get to observe $(Z_1, \ldots, Z_T)$ (called "hidden" variables), but we do get to observe $(X_1, \ldots, X_T)$, which they themselves are generated from the hidden variables. E.g., we only get to see a noisy/corrupted version of the true sequence

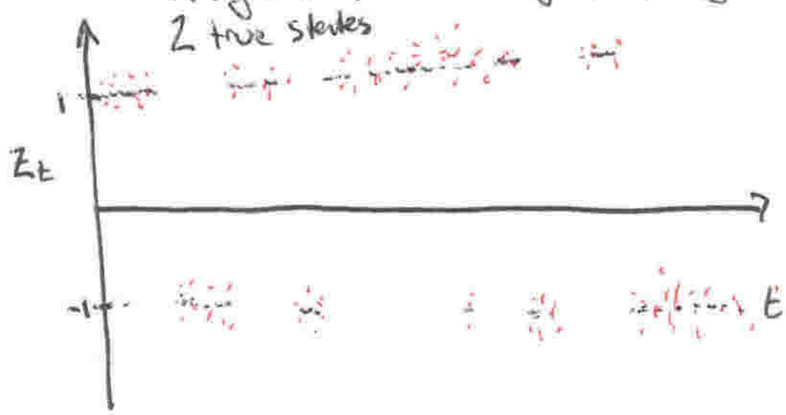
observing an airplane trajectory w/ a radar



- we'd like to try to reconstruct $Z_1, \ldots, Z_T$ from $X_1, \ldots X_T$

(Kalman Filtering)

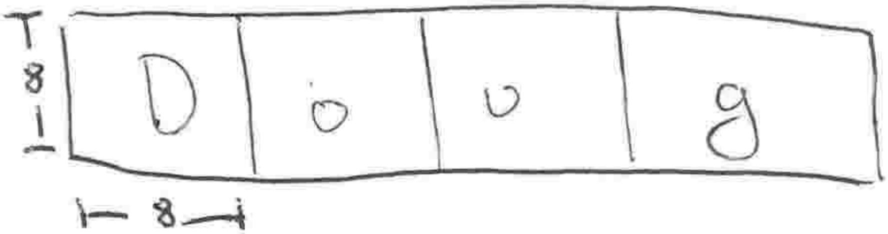
A system w/ a "sticky" property w/ only 2 true states



• : observed sequence, $X_t \in \mathbb{R}$ (say drawn from a Gaussian about Z_t)

• : hidden sequence, $Z_t \in \{-1, 1\}$

- But, the true state doesn't have to be hidden just b/c of noise. Sometimes we just don't know it, eg. teaching a computer handwriting recognition from a sequence of written letters:



- say we observe a sequence of 8×8 images, and want to infer the actual letter that was written: $X_t \in \mathbb{R}^{64}$; $Z_t \in \{A, B, C, \ldots, Z, a, b, \ldots z\}$

Set up

- let $Z_t \in \{1, 2, \ldots, |S|\}$ (we only consider discrete hidden states here) be our hidden variables. with sequence $Z = (Z_1, Z_2, \ldots, Z_T) \in S^T$
- let $x_t \in \{1, 2, \ldots, |O|\}$ (we only consider discrete observed states)

- Like the Markov Property $\left(P(Z_t \mid Z_{t-1}, Z_{t-2} \ldots Z_1) = P(Z_t \mid Z_{t-1}) \right)$, in HMMs we assume that the probability that $X_t = x_t$ only conditionally depends on the current hidden state: $P(x_t \mid x_{t-1}, x_{t-2} \ldots x_1, Z_t, Z_{t-1} \ldots Z_1) = P(x_t \mid Z_t)$ ~>

- This is called the emission probability. Again, let $A \in \mathbb{R}^{|S| \times |S|}$, with $A_{ij} \equiv P(Z_t = j \mid Z_{t-1} = i)$, be the transitition matrix ($A_{ij} \geq 0 \ \forall i,j$, $\sum_j A_{ij} = 1 \ \forall i$)

- The Emission matrix is defined analogously. $B \in \mathbb{R}^{|S| \times |O|}$, with $B_{ij} \equiv P(X_t = j \mid Z_t = i)$, the probability that the emission is j given that we are in state i. ($B_{ij} \geq 0 \ \forall i,j$ $\sum_j B_{ij} = 1 \ \forall i$)

Thus, our generative model is:

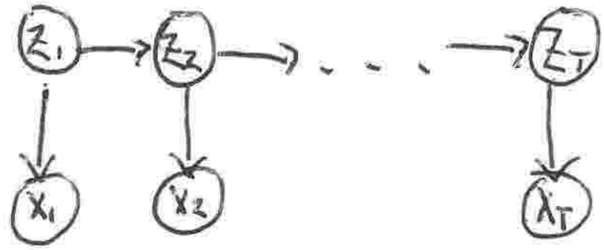
$$Z_1 \sim \text{cat}(\pi)$$

(observed) $X_1 \mid Z_1 = i_1 \sim \text{Cat}(B_{i_1,*})$

$$Z_2 \mid Z_1 = i_1 \sim \text{cat}(A_{i_1,*})$$

$$\vdots$$

(observed) $X_T \mid Z_T = i_T \sim \text{Cat}(B_{i_T,*})$



* Note that when Xs are continuous, we could model the conditional probability w/, say, a multivariate normal:

$$X_t \mid Z_t = i \sim \mathcal{N}(\mu_i, \Sigma_i)$$

Inference in HMMs w/ Forward-Backward Algos.

- once $\hat{A}, \hat{B}, \hat{\pi}$ are learned (in next section), it is natural to want to do inference on the hidden states. Concretely, the following would be very interesting to know:

$$P(Z_{zt} \mid X) \quad \forall t$$ (probability that $Z_t = z_{zt}$ given the data sequence)

$$Z^*$$ (most probable hidden sequence)

- eg. in the Gaussian mixtures model, $P(Z^{(i)} \mid X)$ was interesting b/c it gave us the probability of class labels for each datapoint.

- We could compute these quantities explicitly, but it would take exponential time:

$$P(Z_{zt} \mid X; \hat{A}, \hat{B}, \hat{\pi}) = \frac{P(Z_t, X)}{P(X)} \underset{Z_t}{\propto} \sum_{\substack{z_1, z_2, \\ \dots z_{t-1}, z_{t+1}, \dots \\ z_T}} P(X, Z) = \sum P(X \mid Z) P(Z)$$

$P(Z)$ already computed in Markov Model w/ chain rule.
$P(X|Z)$ can also be computed w/ chain rule:

$$\Rightarrow P(Z_t|X;\hat{A},\hat{B},\hat{\pi}) \propto \sum_{\substack{Z_1, Z_2, \dots \\ Z_{t-1}, Z_{t+1}, \dots Z_T}} \left(\prod_{t'=1}^{T} \hat{B}_{Z_{t'},X_{t'}}\right)\left(\hat{\pi}(Z_1)\prod_{t'=2}^{T} \hat{A}_{Z_{t'-1},Z_{t'}}\right)$$

The denominator, $P(X)$ could also be computed explicitly w/ a sum,
And thus we see that computing $P(Z_t|X)$ boils down to many multiplications of matrix entries. However, there are $|S|^{T-1}$ labellings we would have to sum over, so this algo is at least $\Omega(|S|^{T-1}) = \Omega(|S|^T)$ time complexity.

<u>How can we do better?</u> $\Rightarrow$ Dynamic Programming

Let's somewhat arbitrarily define:

$$\alpha_i(t) \doteq P(X_1, X_2, \dots, X_t, Z_t = i; \hat{A}, \hat{B}, \hat{\pi})$$
$$\beta_i(t) \doteq P(X_{t+1}, \dots, X_T | Z_t = i; \hat{A}, \hat{B}, \hat{\pi})$$

Now:
$$P(Z_t|X;\hat{A},\hat{B},\hat{\pi}) = \frac{P(Z_t, X; \hat{A}\hat{B}\hat{\pi})}{P(X;\hat{A}\hat{B}\hat{\pi})}$$

working on the numerator:

$$P(Z_t = i, X) = \cancel{P(X_{t+1}, \ldots, X_T} \quad \text{(using Markov Properties)}$$

$$P(X_{t+1}, \ldots, X_T \mid \cancel{X_1, \ldots,} X_t, Z_t = i)\, P(Z_t = i, X_1, \ldots, X_t)$$

$$= \beta_i(t)\, \alpha_i(t)$$

likewise, it can be shown that:

$$P(X) = \sum_{i=1}^{|S|} \alpha_i(T)$$

$$\Rightarrow P(Z_t = i \mid X) = \frac{\alpha_i(t)\beta_i(t)}{\sum_{i=1}^{|S|} \alpha_i(T)}$$

It turns out that we can compute α_i, β_i dynamically in time much faster than exponential. Thus, if we can compute $\alpha_i(t), \beta_i(t)$ $\forall\, t = 1, \ldots, T$, $\forall\, i = 1, \ldots, |S|$, we can do all the inference we want.

<u>Forward-Algo</u>

- to compute $\alpha_i(t)$ dynamically, we will need a recurrence relation, which we can derive by taking advantage of Markov Properties

- Since we want a recurrence relation let's try to marginalize 1-step back, then we will condition using normal probability rules, then simplify with Markov properties. This will get us the recurrence:

$$\alpha_i(t) = P(x_1,\dots,x_t, Z_t = i ; \hat{A}, \hat{B}, \hat{\pi})$$

(marginalize) $$= \sum_{j=1}^{|S|} P(Z_{t-1}=j, Z_t=i, x_1,\dots,x_t)$$

conditioning
$$= \sum_{j=1}^{|S|} P(x_t \mid Z_{t-1}=j, Z_t=i, x_1,\dots,x_{t-1})\, P(Z_{t-1}=j, Z_t=i, x_1,\dots x_{t-1})$$

$$= \sum_{j=1}^{|S|} \underbrace{P(x_t \mid Z_t=i, \overset{\text{Markov property}}{\cancel{Z_{t-1}=j, x_1,\dots,x_{t-1}}})}_{\text{known (emission probability)}}\, \underbrace{P(Z_t=i \mid Z_{t-1}=j, \overset{\text{Markov property}}{\cancel{x_1,\dots,x_{t-1}}})}_{\text{known (transition probability)}}\, \underbrace{P(x_1,\dots,x_{t-1}, Z_{t-1}=j)}_{\alpha_j(t-1)}$$

$$= \sum_{j=1}^{|S|} B_{i,x_t} A_{ji}\, \alpha_j(t-1)$$

This is our recursion; we also need a base case:

$$\alpha_i(1) = P(x_1, Z_1=i) = P(x_1 \mid Z_1=i)\,P(Z_1=i) = B_{ix_1}\pi(i)$$

time complexity : $\frac{\text{work}}{\text{subproblem}}$ × # unique subproblems

unique subproblems: must solve $\alpha_i(t)$ $\forall i = 1, \ldots, |S|$, $t = 1, \ldots, T$

$\Rightarrow O(ST)$ subproblems

work : constant time + $O(|S|)$ for the summation.

$$\Rightarrow T(|S|, |T|) = O(|S|^2|T|)$$

A similar algo for β, the backward algo w/ the same time complexity exists as well (very similar)

$\therefore$ we can compute all $P(Z_t = j \mid X)$ in $O(|S|^2|T|)$ time.

Viterbi Algo

- the most probable sequence of states, Z^*, that explains our data would also be something that we are interested in.

$$Z^* = \underset{Z \in S^T}{\operatorname{argmax}} \left\{ P(Z \mid X, \hat{A}, \hat{B}, \hat{\pi}) \right\} = \underset{Z \in S^T}{\operatorname{argmax}} \left\{ \frac{P(X, Z; \hat{A}, \hat{B}, \hat{\pi})}{\sum_Z P(X, Z; \hat{A}, \hat{B}, \hat{\pi})} \right\}$$

$$= \underset{Z \in S^T}{\operatorname{argmax}} \left\{ P(X, Z; \hat{A}, \hat{B}, \hat{\pi}) \right\}$$

- again, this requires checking all $|S|^T$ possibilities to get the argmax
- The Viterbi Algo is another DP algo

Pseudo-code For Forward Algo

input: $x = [x_1, x_2, \ldots, x_T]$ length $|T|$ array of observed emissions

$\hat{A} = |S| \times |S|$ matrix of trans. probs.

$\hat{B} = |S| \times |O|$ matrix of emission probs.

$\hat{\pi}$ = length $|S|$ array of probs. of initial state.

output: $\alpha = |S| \times T$ matrix for all $\alpha_i(t)$ values

---

```
α = |S| x |T| matrix of Nils        // define Memo table

for i=1 to |S|:
    α[i,1] = B̂[i, X[1]] × π̂[i]      // fill in base case since this
                                     doesn't get filled in during
                                     recursion


fb(i,t):

    if α[i,t] != Nil: return α[i,t]       // check if in memo
    if t = 1: return B̂[i, X[1]] × π̂[i]    // base
```

$$r = \sum_{j=1}^{|S|} \hat{B}[i, x[t]] \cdot \hat{A}[j,i] \cdot fb(j, t-1) \quad // \text{ recursion}$$

```
    α[i,t] = r                           // memoize
    return r
fb(|S|,T)                                // fill in α table
```

Learning the Model Parameters

- in this section, I only focus on computing $\hat{A}$, $\hat{B}$. Let's assume π is known (say uniform).

- To obtain $\hat{A}$, $\hat{B}$, the MLE of A, B, the log-likelihood is;

$$
\begin{aligned}
l(A,B) &= \log P(X; A,B) \\
&= \log \sum_{Z} P(X, Z; A,B) \\
&= \log \sum_{Z} P(X|Z; A,B)\, P(Z; A,B) \\
&= \log \sum_{Z} \left(\prod_{t=1}^{T} B_{z_t x_t} \right) \left(\pi(z_1) \prod_{t=2}^{T} A_{z_{t-1} z_t} \right)
\end{aligned}
$$

- As w/ previous MLE of params in other models w/ hidden variables performing this maximization is intractable, and we resort to the EM algo, which has been very successful for computing MLEs for models with hidden variables.

The EM Algo applied here is:

① initialize A^*, B^* to random probability matrices

② repeat till convergence

a) compute $Q(\vec{z}) := P(\vec{z} \mid x, A^*, B^*) \quad \forall \ \vec{z} \in S^T$

b) re-compute A^*, B^*:

$$A^*, B^* := \arg\max_{A,B} \left\{ \sum_{\vec{z}} Q(\vec{z}) \log \frac{P(\vec{z}, x; A, B)}{Q(\vec{z})} \right\}$$

$$\text{s.t.} \quad \sum_{j=1}^{|S|} A_{ij} = 1 \; ; \; A_{ij} \geq 0 \quad \forall \ j, i = 1, \ldots |S|$$

$$\sum_{k=1}^{|O|} B_{ik} = 1 \; ; \; B_{ik} \geq 0 \quad \forall \ i = 1, \ldots |S|, \ k = 1, \ldots |O|$$

- Note that we do not need to construct multiple Q_is (as we usually do for EM) since there is only 1 data point here (namely, the sequence x). A version of the EM algo does exist for multiple observed sequences.

- Also note that we need to compute $Q(\vec{z}) \ \forall \ \vec{z}$, and that the sum in b) is over all S^T. Obviously we'll need DP to handle this.

→

- Let's remaximize A^*, B^*. We can again ignore inequality constraints since we'll get all $A_{ij}, B_{ij} \geq 0$ anyway.

- Using some probability rules, and the defns. of A, B, as well as the Markov assumptions, it can easily be shown that the Lagrangian is:

$$\mathcal{L}(A,B,\delta,\varepsilon) = \sum_{\vec{z}} Q(\vec{z}) \sum_{i=1}^{|S|} \sum_{j=1}^{|S|} \sum_{k=1}^{|O|} \left\{ \sum_{t=1}^{T} \mathbb{1}\{z_t = j \wedge x_t = k\} \log B_{jk} + \sum_{t=2}^{T} \mathbb{1}\{z_{t-1} = i \wedge z_t = j\} \log A_{ij} + \log \pi(z_1) \right\}$$
$$+ \sum_{j=1}^{|S|} \varepsilon_j \left(1 - \sum_{k=1}^{|O|} B_{jk}\right) + \sum_{i=1}^{|S|} \delta_i \left(1 - \sum_{j=1}^{|S|} A_{ij}\right)$$

which can be maximized by solving the following set of eqns:

$$\begin{cases} \nabla_A \mathcal{L}(\hat{A},\hat{B},\hat{\delta},\hat{\varepsilon}) = 0 \\ \nabla_B \mathcal{L}(\hat{A},\hat{B},\hat{\delta},\hat{\varepsilon}) = 0 \\ \nabla_\delta \mathcal{L}(\hat{A},\hat{B},\hat{\delta},\hat{\varepsilon}) = 0 \\ \nabla_\varepsilon \mathcal{L}(\hat{A},\hat{B},\hat{\delta},\hat{\varepsilon}) = 0 \end{cases}$$

where the 0s are the appropriate sized matrices/vectors of 0s.

the maximized soln. is

$$\hat{A}_{ij} = \frac{\sum_{\vec{z}} Q(\vec{z}) \sum_{t=2}^{T} \mathbb{1}\{z_{t-1} = i \wedge z_t = j\}}{\sum_{\vec{z}} Q(\vec{z}) \sum_{t=2}^{T} \mathbb{1}\{z_{t-1} = i\}}$$

$$\hat{B}_{ij} = \frac{\sum_{\vec{z}} Q(\vec{z}) \sum_{t=1}^{T} \mathbb{1}\{z_t = i \wedge x_t = k\}}{\sum_{\vec{z}} Q(\vec{z}) \sum_{t=1}^{T} \mathbb{1}\{z_t = j\}}$$

Again, these formulas have a very intuitive form. $Q(\vec{z}) = P(\vec{z} | \vec{x}, A, B)$ is the conditional probability of $\vec{z}$ parameterized by the <u>old</u> estimates of A, B, and therefore:

$$\hat{A}_{ij} = \frac{E\left[\sum_{t=2}^{T} \mathbb{1}\{Z_{t-1} = i \wedge Z_t = j\} \,\middle|\, X = x\right]}{E\left[\sum_{t=2}^{T} \mathbb{1}\{Z_t = i\} \,\middle|\, X = x\right]}$$

- Thus, $\hat{A}_{ij}$ is just the total expected # of transitions from i to j across all times $t = 2, \ldots, T$ conditioned on the fact that we've observed the sequence $\vec{x}$, (and using the old parameters of A, B).

- $\hat{B}_{ij}$ has an analogous interpretation.

Obviously it is not computationally feasible to sum over all $|S|^T$ labellings of z. However, it is not difficult to re-write these expressions using some probability and the defns. of $\alpha_i(t)$, $\beta_i(t)$, A, B as:

$$\hat{A}_{ij} = \frac{\sum_{t=2}^{T} \alpha_i(t) A_{ij} B_{j x_t} \beta_j(t+1)}{\sum_{j=1}^{|S|} \sum_{t=1}^{T} \alpha_i(t) A_{ij} B_{j x_t} \beta_j(t+1)} = \frac{\sum_{t=2}^{T} \gamma_t(i,j)}{\sum_{j=1}^{|S|} \sum_{t=1}^{T} \gamma_t(i,j)}$$

and

$$\hat{B}_{ij} = \frac{\sum_{j=1}^{|S|} \sum_{t=1}^{T} \mathbb{1}\{x_t = k\} \gamma_t(i,j)}{\sum_{i=1}^{|S|} \sum_{t=1}^{T} \gamma_t(i,j)},$$

where $\alpha_i(t)$ and $\beta_j(t+1)$ are computed w/ the forward and backward algos using the <u>old</u> estimates of A, B

Thus, the full EM algo for computing the MLE of A, B for an HMM (called Baum-Welch algo) is:

Baum Welch Algo

Input: $A \in \mathbb{R}^{|S|\times|S|}$ and $B \in \mathbb{R}^{|S|\times|V|}$, which are randomized, valid probability matrices.

Output: $\hat{A}, \hat{B}$, the MLE of A, B.

① Repeat until convergence {

E-step 1a) Run Forward and Backward algos to compute

$$\alpha_i(t), \beta_i(t) \quad \forall\ i = 1, \dots, S, \quad t = 1, \dots, T$$

$$\gamma_t(i,j) := \alpha_i(t) A_{ij} B_{j x_t} \beta_j(t+1)$$

M-step 1b) Re-compute A, B with:

$$A_{ij} := \frac{\sum_{t=1}^{T} \gamma_t(i,j)}{\sum_{j=1}^{|S|} \sum_{t=1}^{T} \gamma_t(i,j)}$$

$$B_{jk} := \frac{\sum_{i=1}^{|S|} \sum_{t=1}^{T} \mathbb{1}\{x_t = k\}\, \gamma_t(i,j)}{\sum_{i=1}^{|S|} \sum_{t=1}^{T} \gamma_t(i,j)}$$

}

- thus, in the Algo, the forward / backward algos are used ~~to~~ ~~as~~ as subroutines.

- the time complexity is dominated by the DP subroutine in the Estep ($O(|S|^2 T)$), so that the total time complexity is :

$$\#\text{iterations} \times O(|S|^2 T).$$