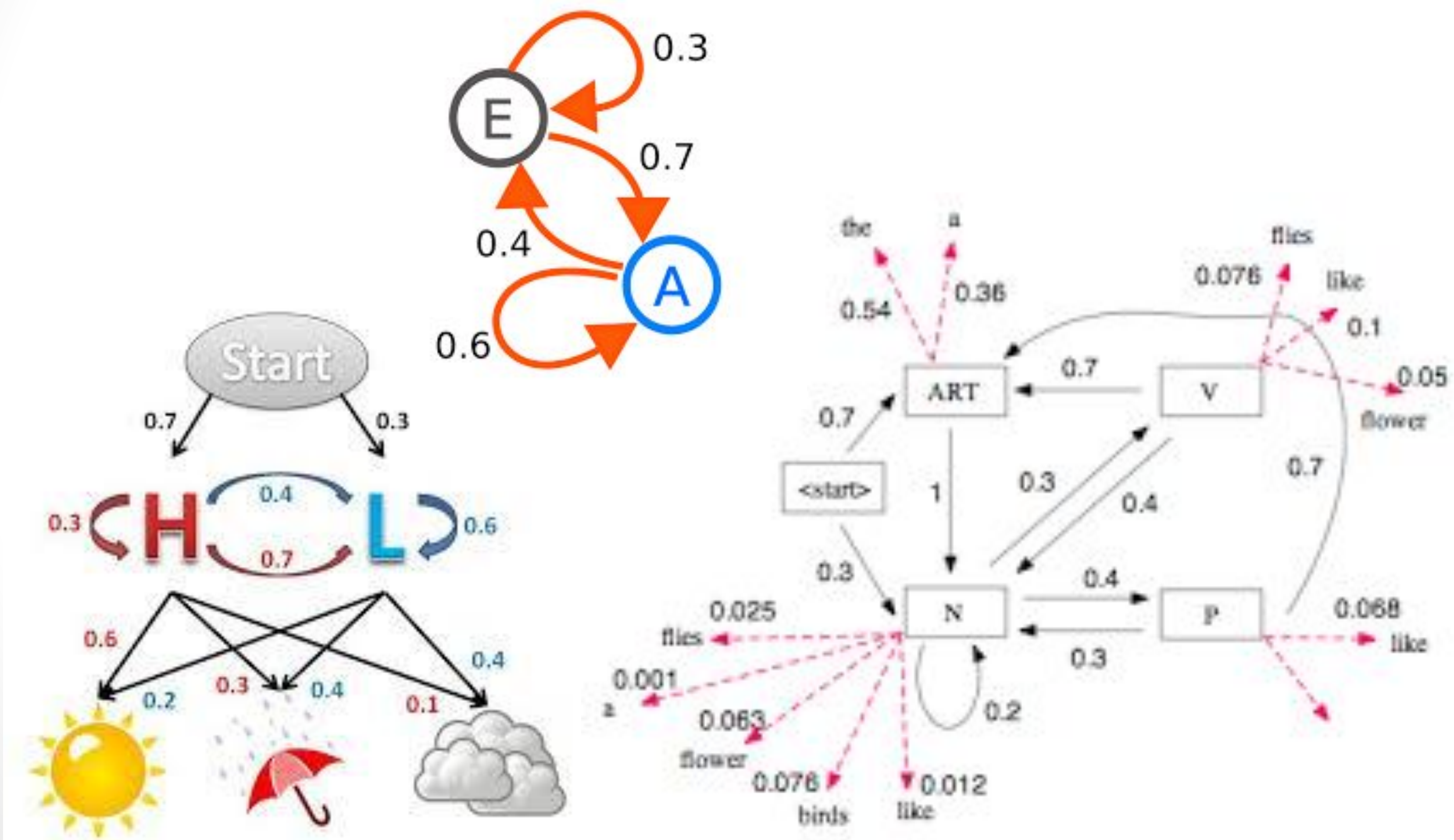
Topic 11 :
Hidden Markov Model

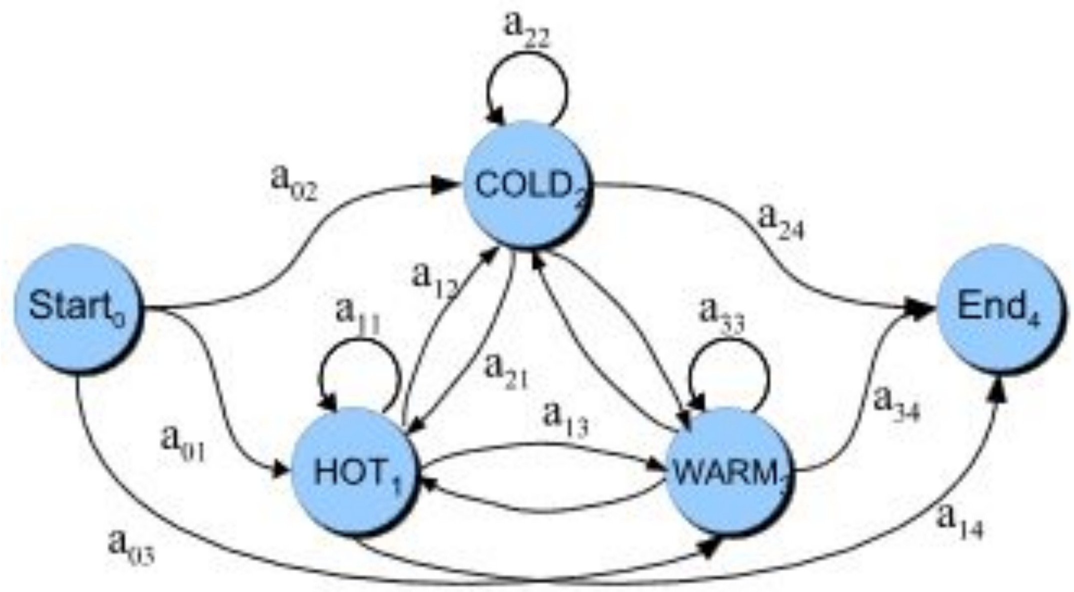


The Markov Chain (Revisited)

- A Markov Chain is a weighted finite state automaton
- Each arc is associated with a probability indicating how likely a path is to be taken
- The probability on all the arcs leaving a node must sum to 1
- Input sequence uniquely determines which states the automaton will go through
- Cannot represent ambiguity

Markov Chain Example 1

- Assign probabilities to a sequence of weather events **weather events**
- States, Q = {start, hot, cold, warm, end}
- Transition prob. = { a_{01}, …. , a_{34} }



Markov Assumption I

- **First order Markov Chain:**
  - The probability of a state is dependent only on the previous state

$$p(q_i|q_1 \dots q_{i-1}) = p(q_i|q_{i-1})$$

  - Each a_{ij} (i.e., transition prob. matrix) expresses the probability $p(q_j|q_i)$ conforming to the probability law → values of the outgoing arcs from a given state must sum to 1

$$\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$$

i = trans. prob
j = #states

Markov Assumption II

- **First order Markov Chain:**
  - Markov chain does not rely on start or end state, probability distributions done over initial and accepting states

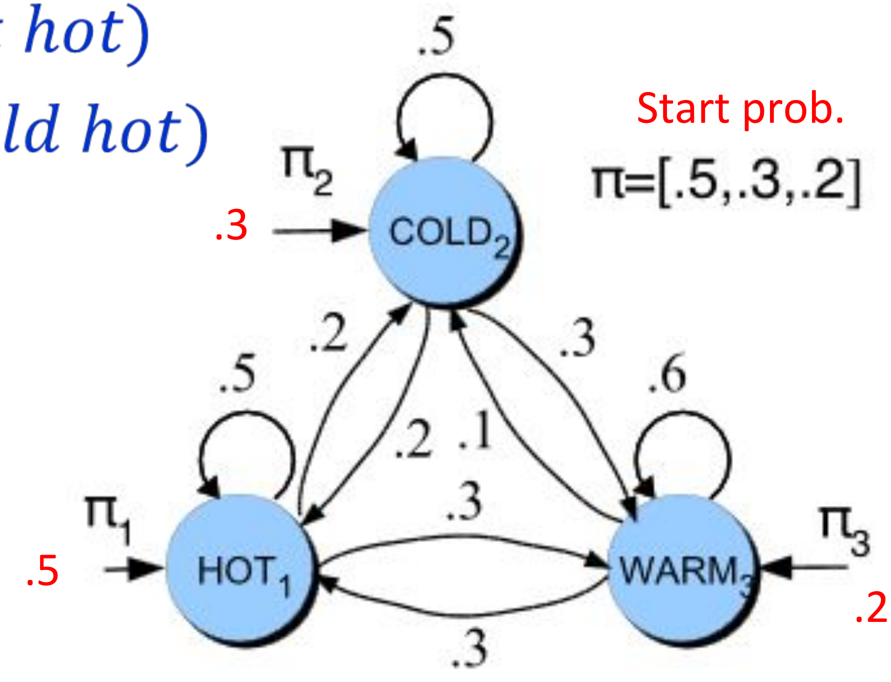
$$p(q_i|START) = p(q_i|q_{i-1})$$

  - Prob. of state 1 represented by a_{01} or π_1
  - All π_i probabilities must sum to 1

$$\sum_{i=1}^{n} \pi_i = 1$$

Markov Chain Example II

- Using the π vector, representing the distribution over start state probabilities :
  - What is the probability of the following?
    - $p(hot\ hot\ hot\ hot)$
    - $p(cold\ hot\ cold\ hot)$



Hidden Markov Model

- Hidden means “non-observable” sequence/events
  - Example: POS tagging – known/observable words but unknown**(hidden)** labels/tags.
  - Labels/tags are inferred from word sequence

Hidden Markov Model Example

- Conducting a study on weather forecast in 2018 in Kuantan, Pahang
  - **Not available:** records of the weather in Kuantan, Pahang for current month.
  - **Available**: a diary listing down how many ice creams X ate every day during hot weather in the current month.
  - **Goal**: use observations in diary to estimate daily temperature
  - **Assumption:** only 2 kinds of days - cold (C) and hot (H)

Hidden Markov Model Definition

- A set of **states**

$$Q = q_1 q_2 \dots q_n$$

- A **transition probability matrix** A, each a_{ij} represents the probability of moving from state i to state j, s.t $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$

$$A = a_{01} a_{02} \dots a_{n1} \dots a_{nn}$$

- A set of **observations**, each one drawn from a vocabulary $V = v_1, v_2, \dots, v_v$

$$O = o_1 o_2 \dots o_n$$

i.e, freq.

Hidden Markov Model Definition

- A set of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation $\boldsymbol{o_t}$ ($\boldsymbol{t}$ = time) being generated from a state $\boldsymbol{i}$

  $$B = b_i(o_t)$$

- A special **start state** and **end state** which are not associated with observation:

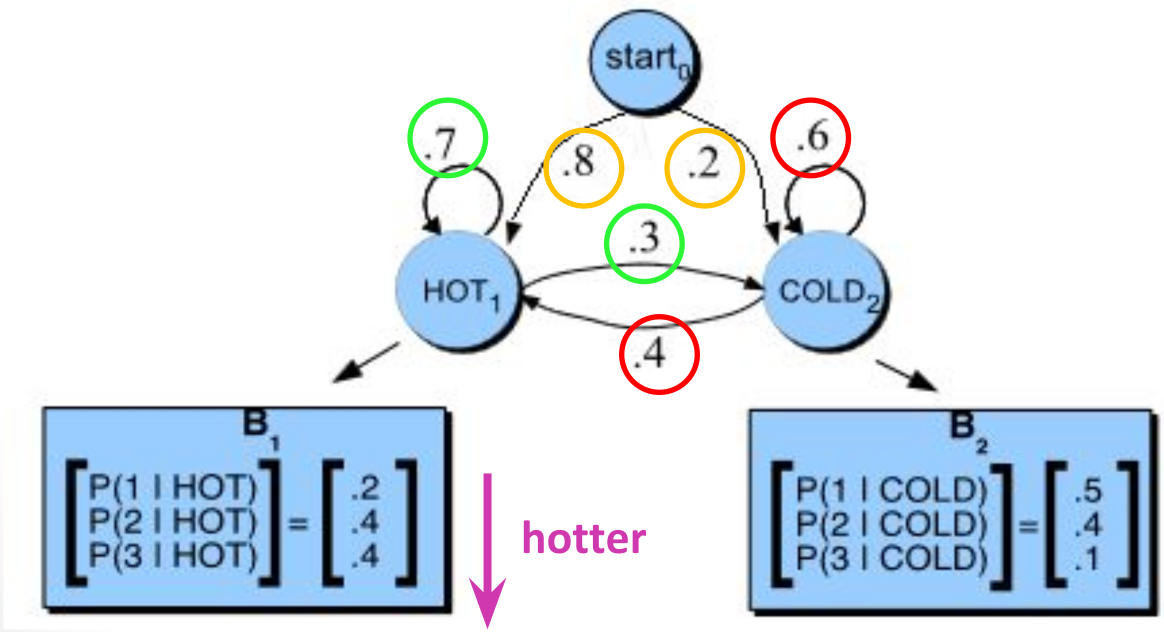
  $$q_o, q_{end}$$

- An initial probability distribution over states

  $$\pi = \pi_1, \pi_2, \ldots, \pi_2 \qquad \text{s.t } \sum_{i=1}^{n} \pi_i = 1$$

HMM Solution to Ice-cream Task

- Hidden states: H(Hot), C (Cold) weather
- Observations (drawn from O = {1,2,3})= no. of ice-creams eaten by X on a specific day



hotter

colder

$$\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$$

$$\sum_{i=1}^{n} \pi_i = 1$$

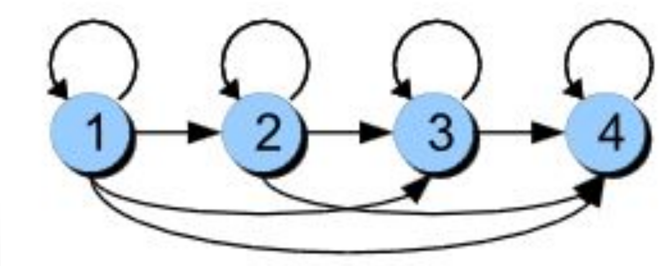
more ice-cream = $\uparrow$ prob. = $\uparrow$ temp.

less ice-cream = $\uparrow$ prob. = $\downarrow$ temp.

HMM with Zero Probability Transition

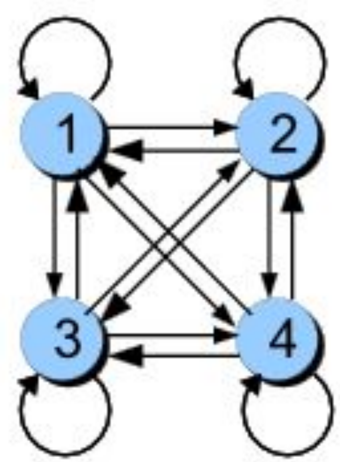
- HMM with transition that proceeds from left to right (or from lower to higher states only) known as Bakis HMM
- Commonly used to model temporal processes such as speech (i.e., time sensitive events)

Each state is connected to each other



Left-to-right (Bakis) HMM

Each state leads back to itself



Fully connected(ergodic) HMM

Fundamental Problems in HMM

- **Computing likelihood**
  - Given a HMM $\boldsymbol{\lambda} = (\boldsymbol{A}, \boldsymbol{B})$ and an observation $\boldsymbol{O}$, determine the likelihood of $\boldsymbol{P(O \mid \lambda)}$
- **Decoding/deciphering**
  - Determine which sequence of variables/states is the source of some sequence of observations
- **Learning**
  - Given an observation sequence $\boldsymbol{O}$ and the set of possible states in HMM, learn the HMM parameters $\boldsymbol{A}$ and $\boldsymbol{B}$

Computing Likelihood

- What is the **probability of the sequence 3 1 3** (i.e., ice-cream problem)?
  - hidden state sequence,

$$Q = q_0, q_1, q_2, \dots, q_n$$

  - observation sequence,

$$O = o_1, o_2, \dots, o_n$$

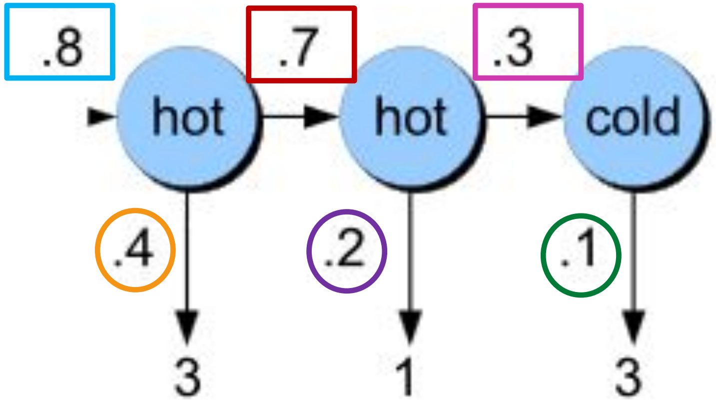
$$P(O|Q) = \prod_{i=1}^{n} P(o_i|q_i) \times \prod_{i=1}^{n} P(q_i|q_{i-1})$$

Computing Likelihood

- One possible hidden state sequence :

$$P(3\ 1\ 3|hot\ hot\ cold) = [P(3|hot) \times P(1|hot) \times P(3|cold)] \times [P(hot|start) \times P(hot|hot) \times P(cold|hot)]$$

$$= [0.4 \times 0.2 \times 0.1] \times [0.8 \times 0.7 \times 0.3]$$



Computing Likelihood

- All possible hidden state sequences = 2^3
  - (hot hot cold), (hot cold hot), (hot cold cold), (hot hot hot), (cold cold hot),(cold hot cold), (cold hot hot), (cold cold cold)
- Total likelihood for sequence **3 1 3**:
  - P(3 1 3|hot hot cold) + P(3 1 3|hot cold hot) + P(3 1 3|hot cold cold) + P(3 1 3|hot hot hot) + ...
- ***N*** hidden states and ***T*** observations = N^T

Forward algorithm

Computing Likelihood : (Forward Algorithm)

- Use forward algorithm for efficiency
  - A dynamic programming algorithm that **uses a table to store intermediate values** as it builds up the probability of the observation sequence
  - Computes observation probability by **summing over the probabilities of all possible hidden-state paths** that could generate an observation sequence
- Implicitly fold each of these paths into a single **forward trellis**

Computing Likelihood : (Forward Algorithm)

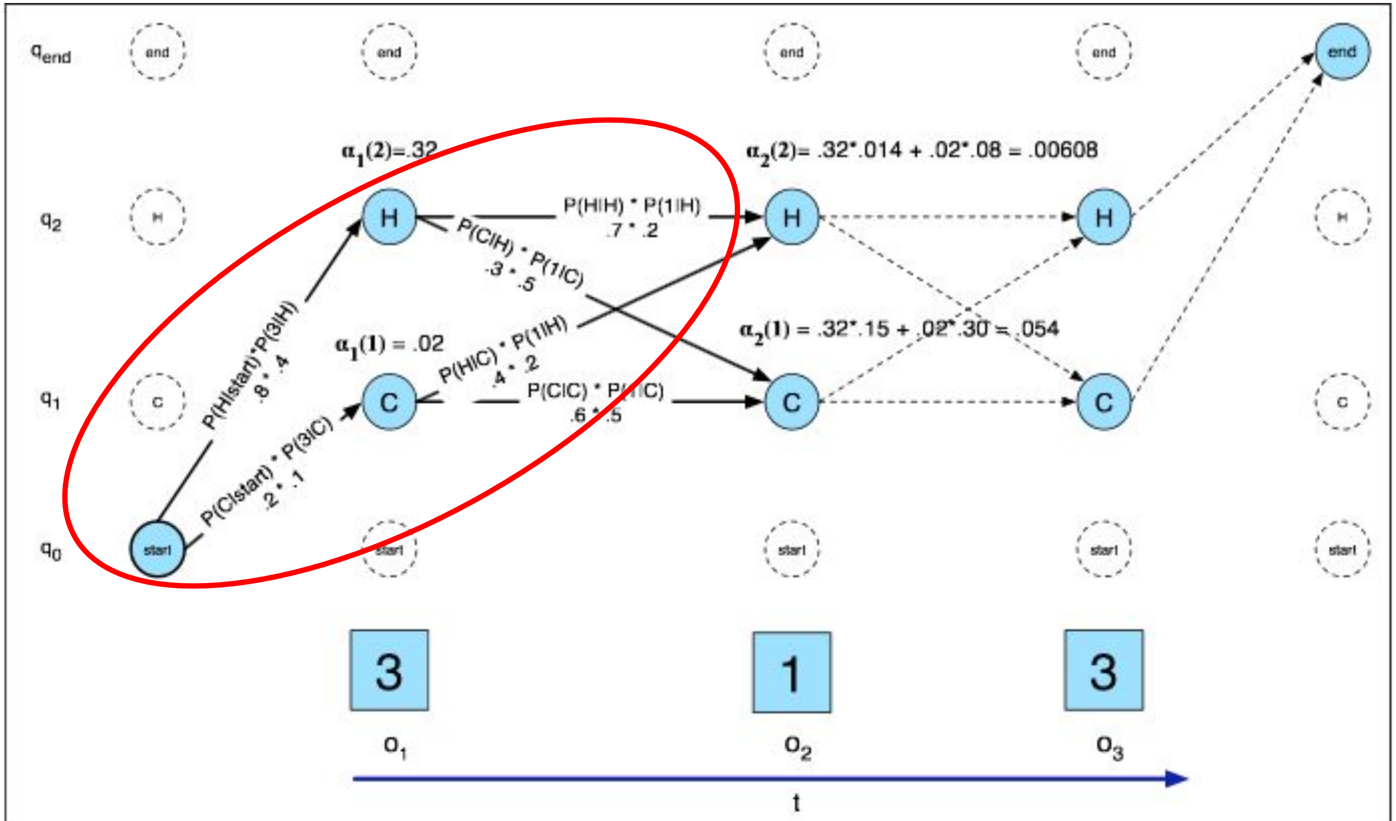


Figure 6.6 The forward trellis for computing the total observation likelihood for the ice-cream events *3 1 3*. Hidden states are in circles, observations in squares. White (unfilled) circles indicate illegal transitions. The figure shows the computation of $\alpha_t(j)$ for two states at two time steps. The computation in each cell follows Eq˙ 6.10: $\alpha_t(j) = \sum_{i=1}^{N-1} \alpha_{t-1}(i) a_{ij} b_j(o_t)$. The resulting probability expressed in each cell is Eq˙ 6.11: $\alpha_t(j) = P(o_1, o_2 \ldots o_t, q_t = j|\lambda)$.

Computing Likelihood : (Forward Algorithm)

- Each cell of the forward algorithm trellis $\alpha_t(j)$ represents the **probability of being in state** j after seeing the first t **observations**, given the **automaton** λ
- Value of each cell $\alpha_t(j)$ is computed by **summing over the probabilities of every path** leading to this cell

$$\alpha_t(j) = P(o_1, o_2 \ldots o_t, q_t = j | \lambda)$$

- where $q_t = j$ is "the probability that the t^{th} state in the sequence of states is state j"

Computing Likelihood : (Forward Algorithm)

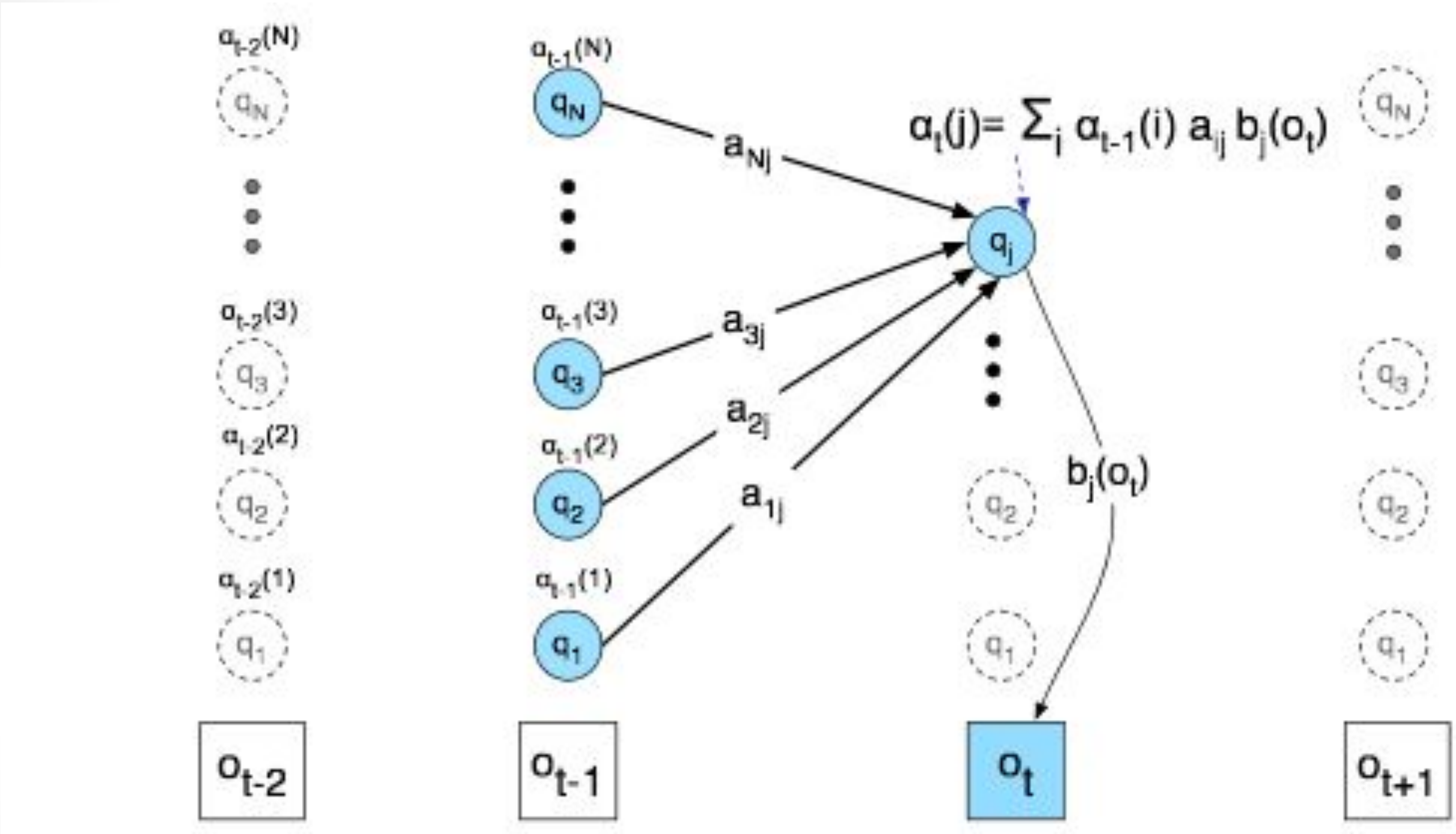
- Compute probability by summing over the extensions of all the paths that lead to current cell.
- For a given state q_j at time t, the value $\alpha_t(j)$ is computed as:

$$\alpha_t(j) = \sum_{i=1}^{N-1} \alpha_{t-1}(i) a_{ij} b_j(o_t)$$

Computing Likelihood : (Forward Algorithm)

- Three factors multiplied in equation to extend previous paths to compute the forward probability at time ***t***
  - Previous forward path probability from the previous time step $\boldsymbol{\alpha_{t-1}(i)}$
  - Transition probability from previous state $\boldsymbol{q_i}$ to current state $\boldsymbol{q_j}$ $\mathbf{a_{ij}}$
  - State observation likelihood of the observation symbol $\boldsymbol{o_t}$ given the current state ***j*** $\mathbf{b_j(o_t)}$

Forward Algorithm



Computation of a single element $\boldsymbol{\alpha_t(i)}$ in the trellis by summing all previous values $\boldsymbol{\alpha_{t-1}}$ weighted by their transition probabilities $\boldsymbol{a}$ and multiplying by the observation probability $\boldsymbol{b_i(o_{t+1})}$

Forward Algorithm Pseudocode

function FORWARD(*observations* of len *T*, *state-graph*) **returns** *forward-probability*

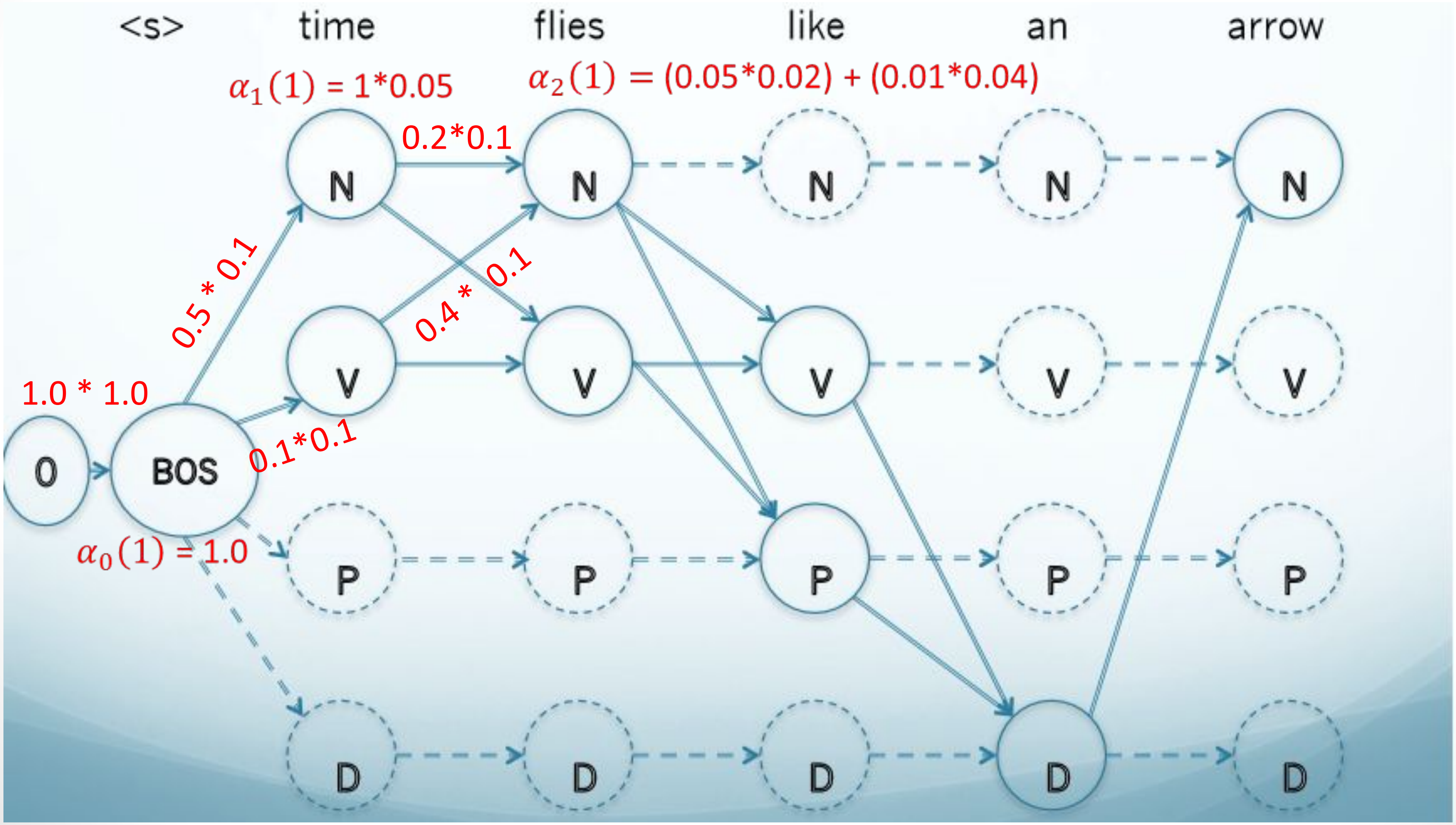
num-states ← NUM-OF-STATES(*state-graph*)
Create a probability matrix *forward[num-states+2,T+2]*
forward[0,0] ← 1.0
for each time step *t* **from** 1 **to** *T* **do**
  for each state *s* **from** 1 **to** *num-states* **do**
    $$forward[s,t] \leftarrow \sum_{1 \leq s' \leq num\text{-}states} forward[s',t-1] * a_{s',s} * b_s(o_t)$$
return the sum of the probabilities in the final column of *forward*

NLP HMM Example (Forward algo)

Example : Time flies like an arrow

Transition	Emission
0 BOS 1.0	BOS <s> 1.0
BOS N 0.5	N time 0.1
BOS DT 0.4	V time 0.1
BOS V 0.1	N flies 0.1
DT N 1.0	V flies 0.2
N N 0.2	V like 0.2
N V 0.7	P like 0.1
N P 0.1	DT an 0.3
V DT 0.4	N arrow 0.1
V N 0.4	
V P 0.1	
V V 0.1	
P DT 0.6	
P N 0.4	

HMM Trellis (Forward Algo)



HMM Example

Given the values in the table (Slide 25) and the trellis (Slide 26), use the Forward algorithm to calculate the probabilities of each of the possible path for the sentence “Time flies like an arrow”

Solution (Independent Paths – Likelihood)

Example for path(1) - Product of all transitions

P(time flies like an arrow | O BOS N N V D N)

O -> BOS -> N -> N -> V -> D -> N

= (1.0 * 1.0) * (0.5 * 0.1) * (0.2 * 0.1) *
(0.7 * 0.2) * (0.4 * 0.3) * (1 * 0.1)

= 1.0 * 0.05 * 0.02 * 0.14 * 0.12 * 0.1

= **0.00000168**

Solution (Trellis: Computing each α)

- $\alpha_t(j) = \sum_{i=1}^{n} \alpha_{t-1}(i) * a_{ij} * b_j(o_t)$

$\alpha_0(1)$ = P(<s>|O -> BOS) = 1.0 * 1.0 = 1.0

$\alpha_1(1)$ = P(time|BOS -> N) = 0.5 * 0.1 = 0.05

$\alpha_1(2)$ = P(time|BOS -> V) = 0.1 * 0.1 = 0.01

$\alpha_2(1)$ = (P(time|BOS -> N) * P (flies|N->N))+

(P(time|BOS -> V) * P (flies|V->N))

= (0.05*0.02) + (0.01*0.04)

= 0.001 + 0.0004 = **0.0014**

Trellis: Computing each α

$\boldsymbol{\alpha_2(2) =}$ (P(time|BOS -> N) * P (flies|N->V))+
(P(time|BOS -> V) * P (flies|V->V))
= (0.05*0.14) + (0.01*0.02)
= 0.007 + 0.0002 = 0.0072

$\boldsymbol{\alpha_3(1) =}$ (P(time|BOS -> N) * P(flies|N->N) * P(like|N->V)) +
(P(time|BOS -> N) * P(flies|N->V) * P(like|V->V)) +
(P(time|BOS -> V) * P(flies|V->N) * P(like|N->V)) +
(P(time|BOS -> V) * P(flies|V->V) * P(like|V->V)) +
= (0.05 * 0.02 * 0.14) + (0.05 * 0.14 *0.02) +
(0.01 * 0.04 * 0.14) + (0.01 * 0.02 * 0.02)
= 0.00014 + 0.00014 + 0.000056 + 0.000004 = **0.00034**
= $(\alpha_2(1) * 0.14) + (\alpha_2(2) * 0.02)$ **[0.000196 + 0.000144]**

Trellis: Computing each α

$\boldsymbol{\alpha_3(2)}$ **=** (P(time|BOS -> N) * P(flies|N->N) * P(like|N->P)) +

(P(time|BOS -> N) * P(flies|N->V) * P(like|V->P)) +

(P(time|BOS -> V) * P(flies|V->N) * P(like|N->P)) +

(P(time|BOS -> V) * P(flies|V->V) * P(like|V->P)) +

= (0.05 * 0.02 * 0.01) + (0.05 * 0.14 *0.01) +

(0.01 * 0.04 * 0.01) + (0.01 * 0.02 * 0.01)

= 0.00001 + 0.00007 + 0.000004 + 0.000002 = **0.000086**

= $(\alpha_2(1) * 0.01) + (\alpha_2(2) * 0.01)$ **[0.000014 + 0.000072]**

Trellis(Prob) for all possible paths

$\boldsymbol{\alpha_4(1)}$ **=** (P(time|BOS -> N) * P(flies|N->N) * P(like|N->V)) * P(an|V -> D) + …….. + ….. + ….

= (0.05 * 0.02 * 0.14 * 0.12) + …. + …..

= **0.0000562**

=

$= (\alpha_3(1) * 0.12) + (\alpha_3(2) * 0.18)$

$= (0.00034 * 0.12) + (0.000086 * 0.18)$

= [0.0000408 + 0.0000154]

$\boldsymbol{\alpha_5(1)}$ **=** $\alpha_4(1) * 0.1$

= **0.00000562**

In class exercise: HMM

Example : The cook prepares a lovely drink

Transition		Emission
0 BOS 1.0	P DT 0.6	BOS <s> 1.0
BOS N 0.5	P N 0.4	N drink 0.2
BOS DT 0.4	ADV N 0.5	V drink 0.5
BOS V 0.1	ADV ADJ 0.1	N cook 0.1
DT N 0.8	ADV V 0.3	V cook 0.4
DT V 0.1		V prepares 0.1
DT ADV 0.1		DT a 0.4
N N 0.5		DT The 0.3
N V 0.4		ADV lovely 0.2
N P 0.1		
V DT 0.3		
V N 0.4		
V P 0.1		
V V 0.1		

In class Exercise : HMM

- Based on the values in Slide 33 :

i. Draw a finite state machine where states are POS and edges are labeled with the transition probabilities for the sentence “The cook prepares a lovely drink”

ii. Calculate the probabilities of α_{ij} at each time *t* using the *Forward* algorithm.

iii. Subsequently, propose the final best path for the given sentence. Show the steps and details of your calculations accordingly

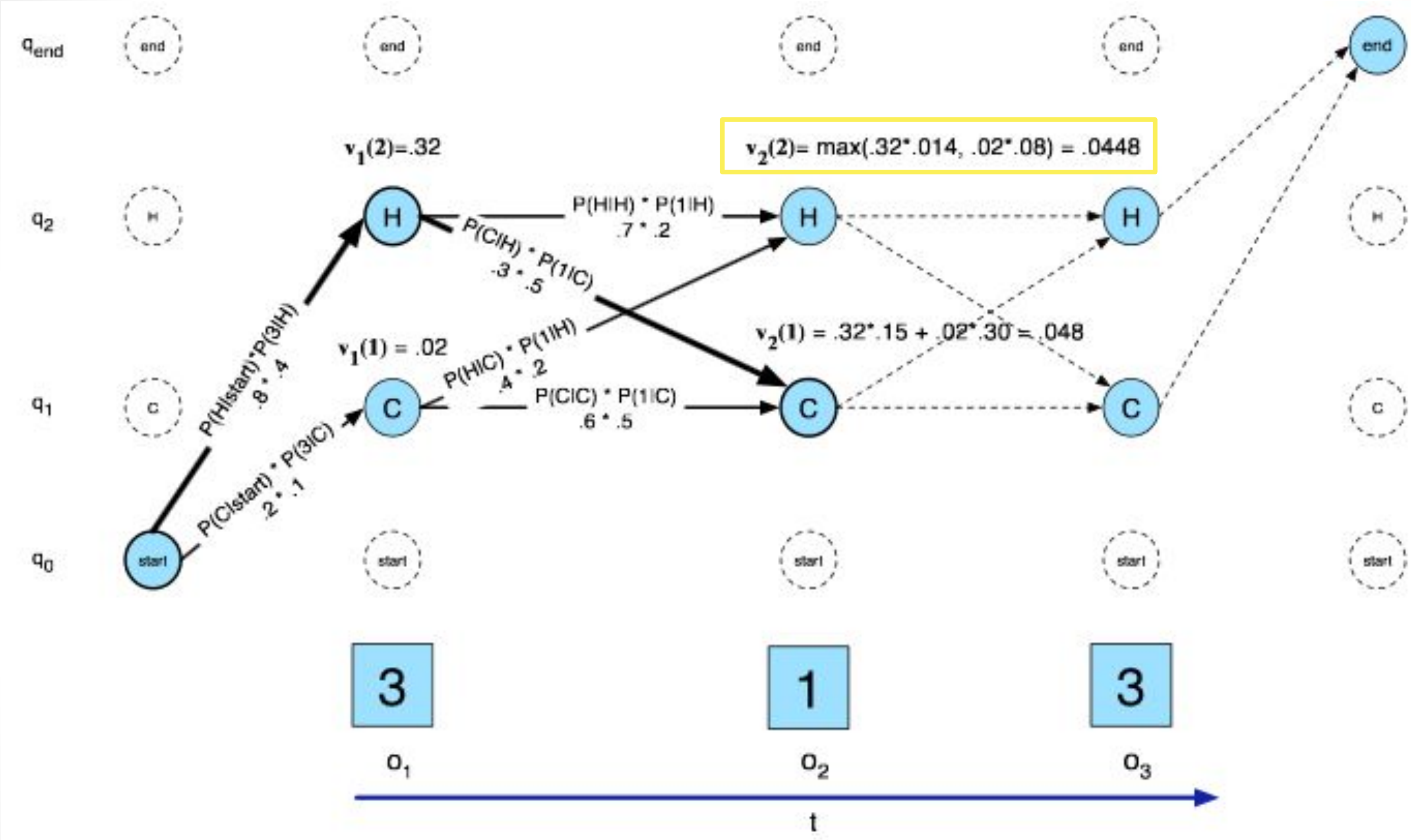
Viterbi algorithm

HMM Decoding (Viterbi algorithm)

- Viterbi is a kind of dynamic programming and makes use of a dynamic programming trellis
- The task of the decoder is to find the best hidden sequence
- Resembles minimum edit distance in aligning 2 sequences

Viterbi Algorithm

- Given as input a **HMM λ = (A,B)** and a sequence of observations, $\mathbf{O=o_1,o_2,\ldots,o_T}$, find the most probable sequence of states $\mathbf{Q=q_1q_2q_3\ldots q_T}$
- The idea is to process the observation sequence left to right, filling out the trellis.



The Viterbi trellis for computing the best path through the hidden state space for the ice-cream eating events 3 1 3

Viterbi Algorithm

- Each cell of the Viterbi trellis, $v_t(j)$ represents the probability that the HMM is in state j after seeing the first t observations and passing through the most likely state sequence $q_1 \ldots q_{t-1}$, given the automaton λ.
- The value of each cell $v_t(j)$ is computed by recursively taking the most probable path that could lead us to this cell. Formally, each cell expresses the following probability

$$v_t(j) = P(q_0, q_1 \ldots q_{t-1}, o_1, o_2 \ldots o_t, q_t = j | \lambda)$$

Viterbi Algorithm

- Given that we had already computed the probability of being in every state at time $t - 1$,
- We compute the Viterbi probability by taking the most probable of the extensions of the paths that lead to the current cell. For a given state Qj at time t, the value $v_t(j)$ is computed as:

$$v_t(j) = \max_{1 \leq i \leq N-1} v_{t-1}(i)\, a_{ij}\, b_j(o_t)$$

- $v_{t-1}(j)$: the previous Viterbi path probability from the previous time step
- a_{ij} : the transition probability from previous state qi to current state q_j
- $b_j(o_t)$: the state observation likelihood of the observation symbol ot given the current state j

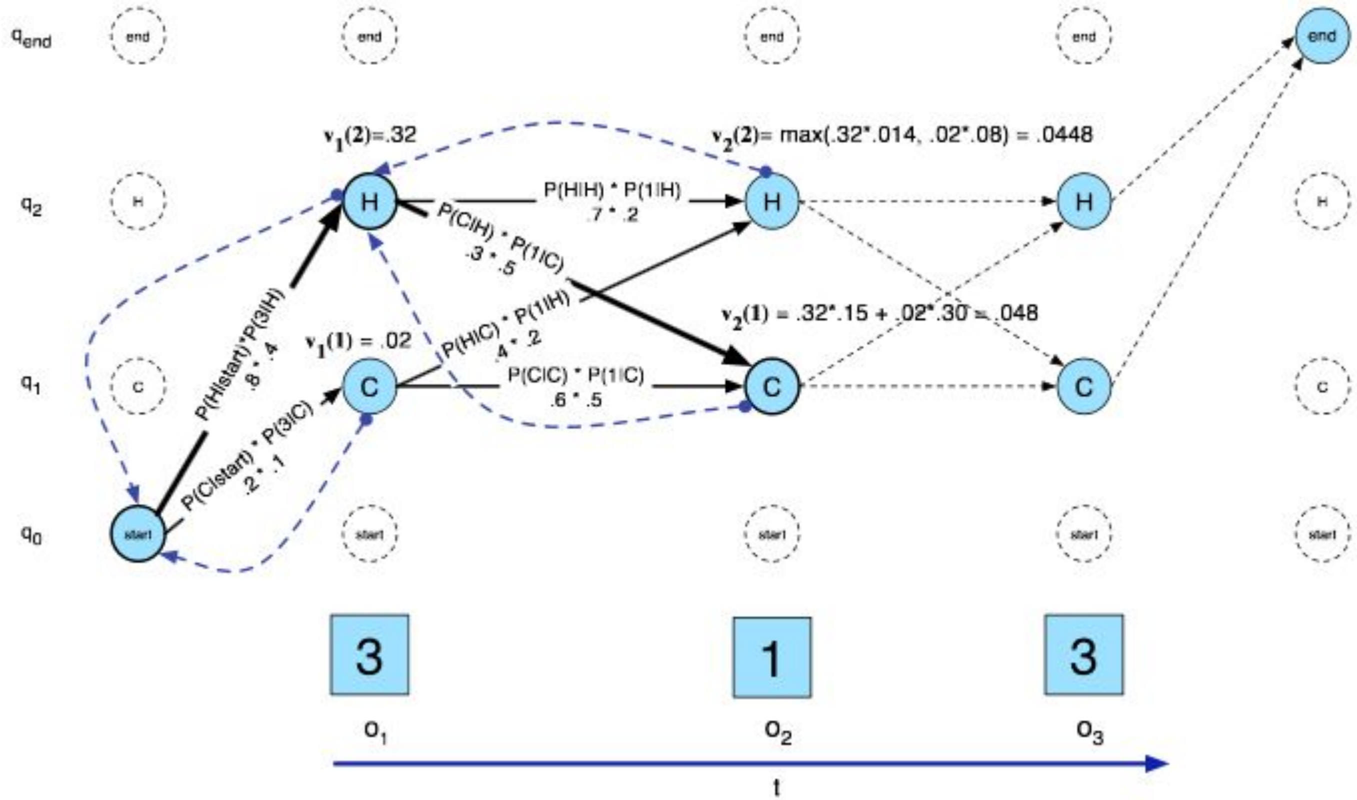
Viterbi Pseudocode

- Viterbi returns the state-path through the HMM which assigns maximum likelihood to the observation sequence
- **Viterbi** is identical to the forward algorithm EXCEPT that it takes the max over the previous path probabilities while **Forward** takes the sum

function VITERBI(*observations* of len *T*,*state-graph*) **returns** *best-path*

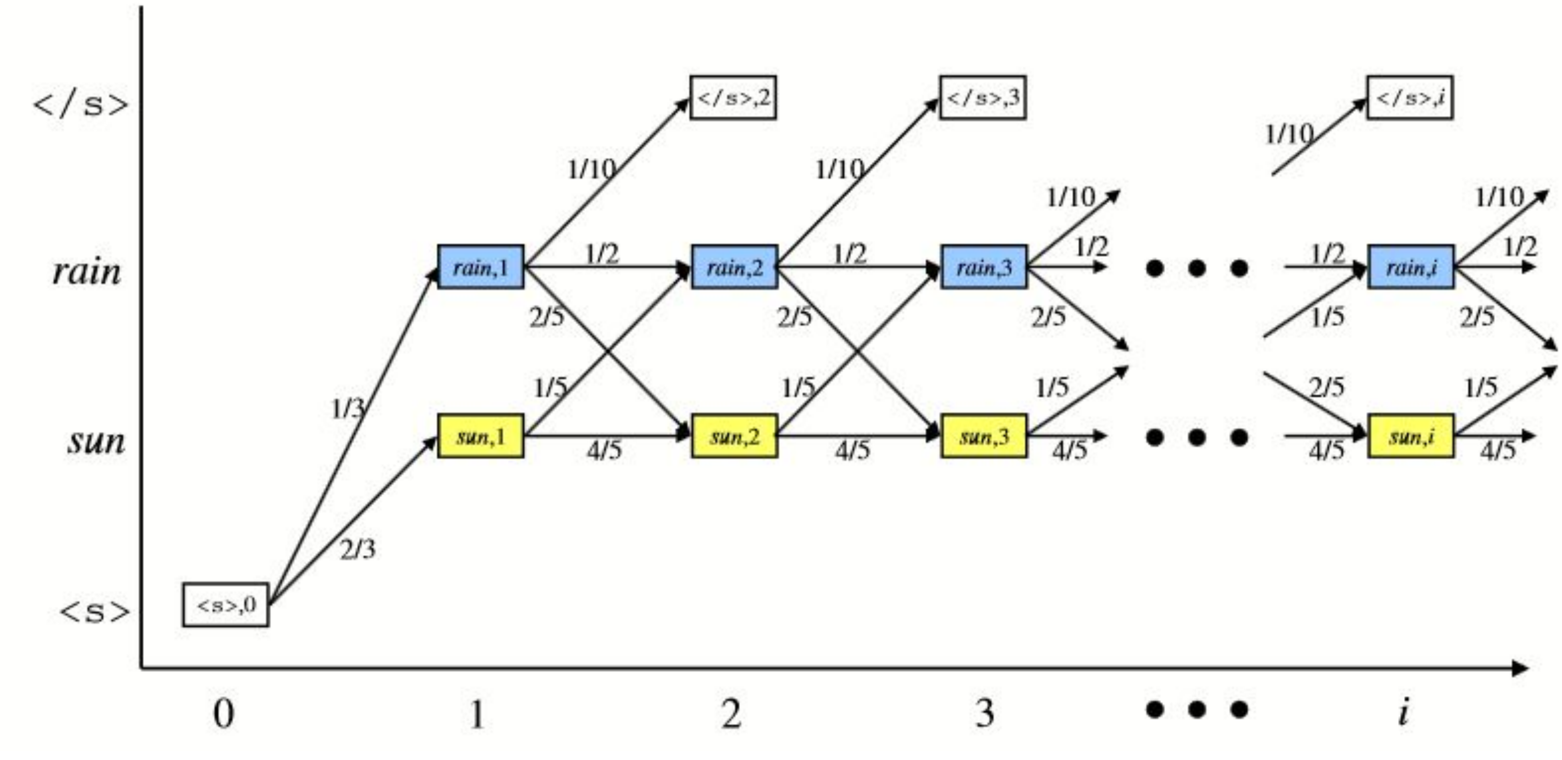
num-states $\leftarrow$ NUM-OF-STATES(*state-graph*)
Create a path probability matrix *viterbi[num-states+2,T+2]*
viterbi[0,0] $\leftarrow$ 1.0
for each time step *t* **from** 1 **to** *T* **do**
  for each state *s* **from** 1 **to** *num-states* **do**
    $viterbi[s,t] \leftarrow \max_{1 \leq s' \leq num\text{-}states} viterbi[s', t-1] * a_{s',s} * b_s(o_t)$
    $backpointer[s,t] \leftarrow \operatorname{argmax}_{1 \leq s' \leq num\text{-}states} viterbi[s', t-1] * a_{s',s}$
Backtrace from highest probability state in final column of *viterbi[]* and return path

Viterbi Backtrace

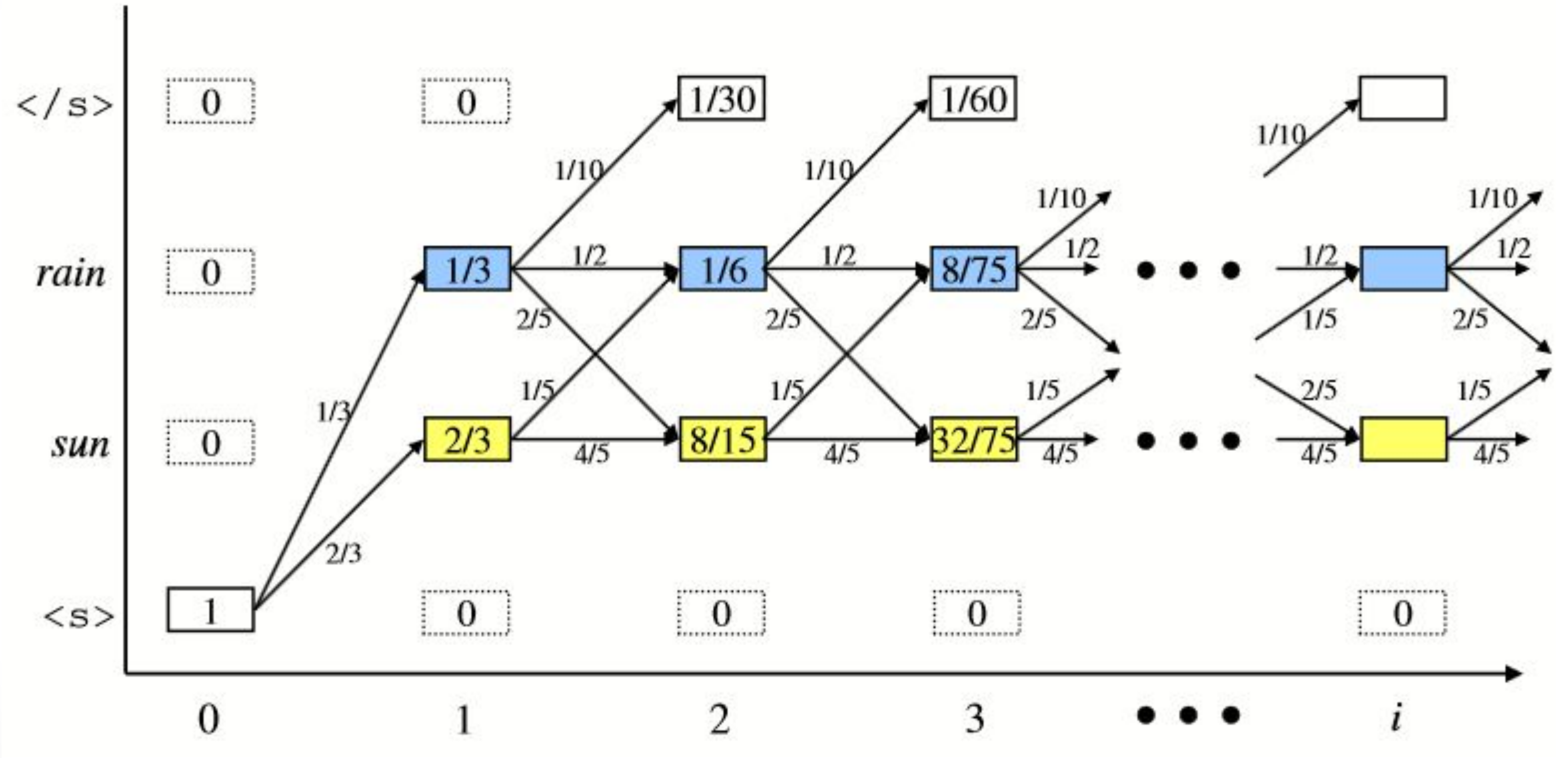


- As we extend each path to a new state account for the next observation, we keep a backpointer (shown with broken blue lines) to the best path that led us to this state

More Trellis Examples (Weather)



More Trellis Examples (Weather)



Example with POS tags

- Transition and observation probabilities

Transition probabilities: $P(t_i|t_{i-1})$

	VB	TO	NN	PPSS
start	0.019	0.0043	0.041	0.067
VB	0.0038	0.0345	0.047	0.070
TO	0.83	0	0.00047	0
NN	0.0040	0.016	0.087	0.0045
PPSS	0.23	0.00079	0.0012	0.00014

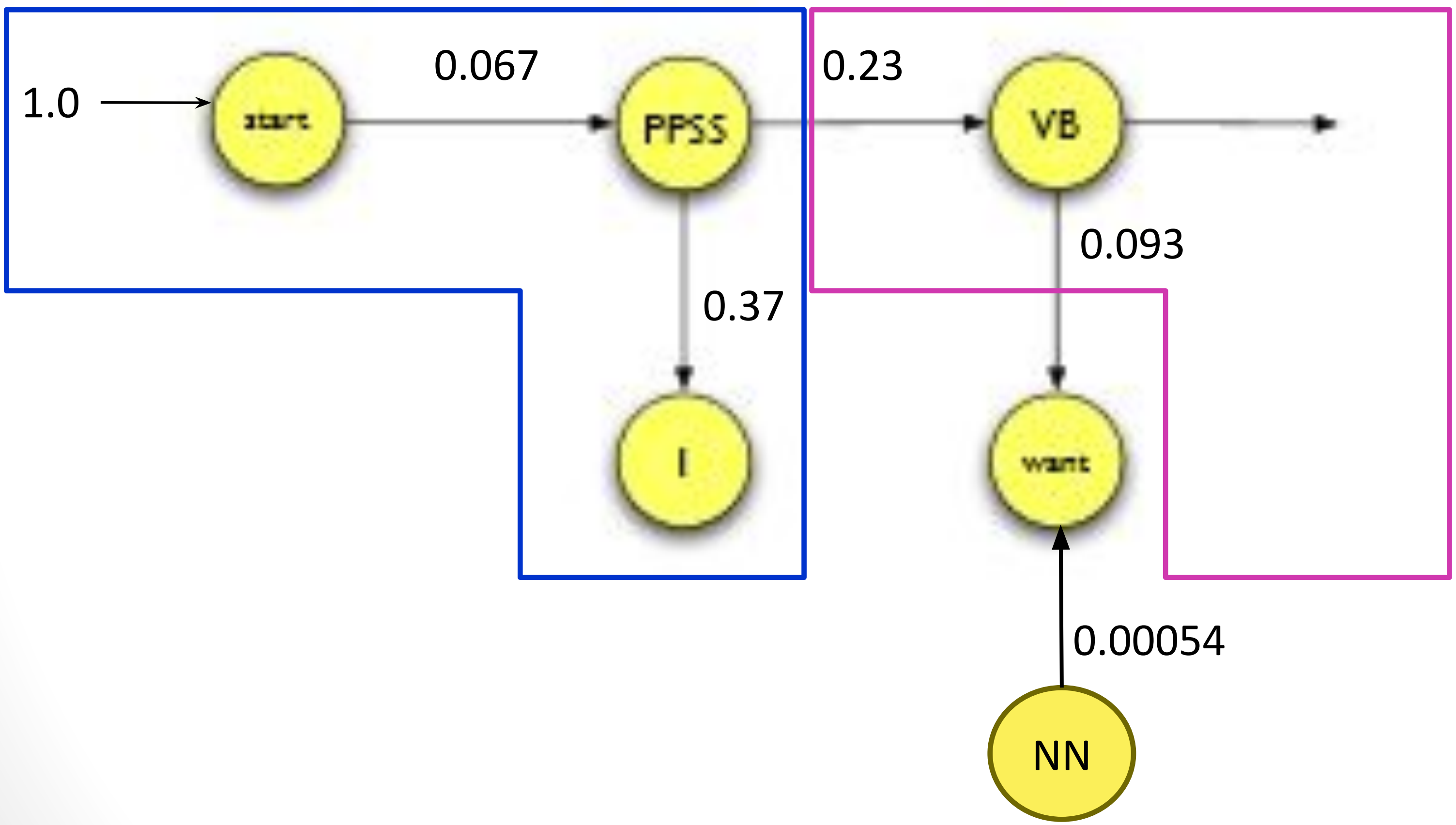
Prior

Observation likelihoods: $P(w_i|t_i)$

	I	want	to	race
VB	0	0.0093	0	0.00012
TO	0	0	0.99	0
NN	0	0.000054	0	0.00057
PPSS	0.37	0	0	0

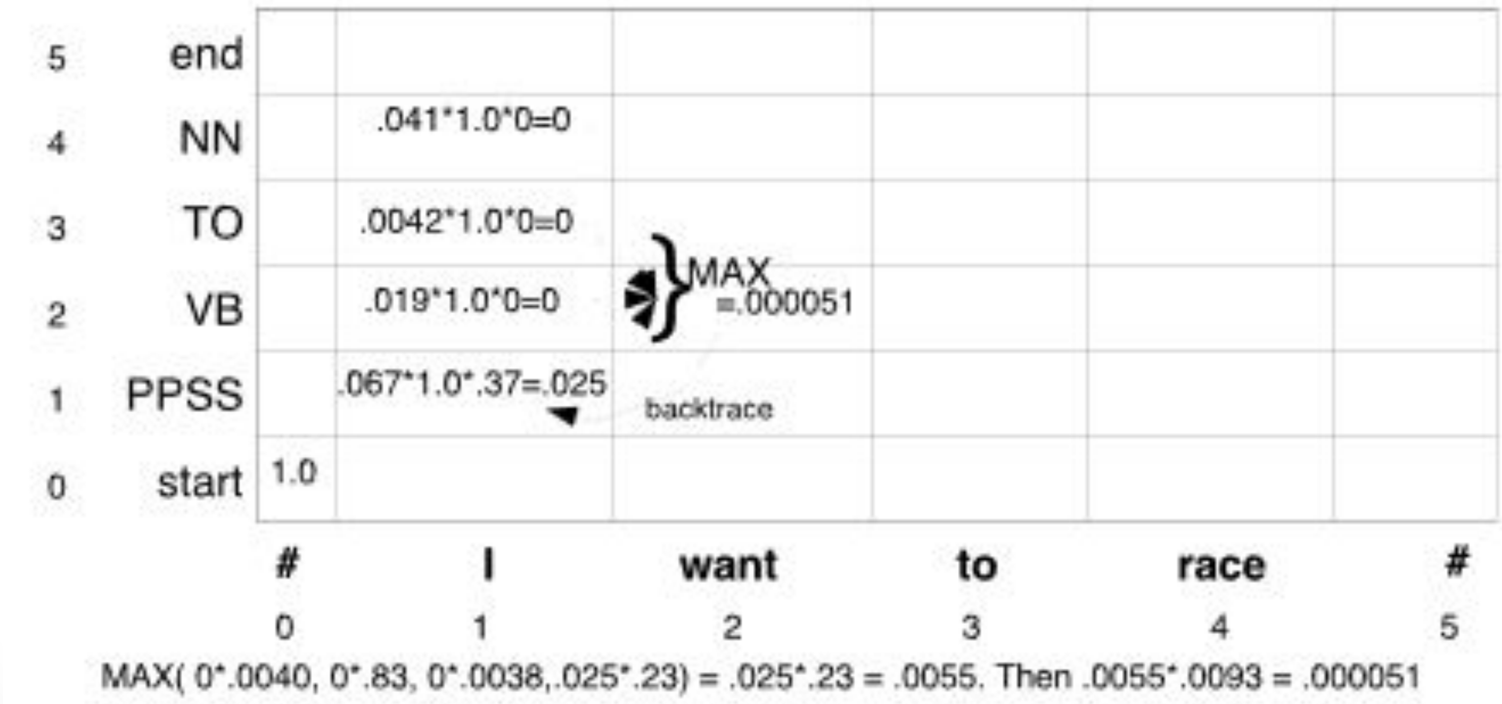
Likelihood

Decoded HMM

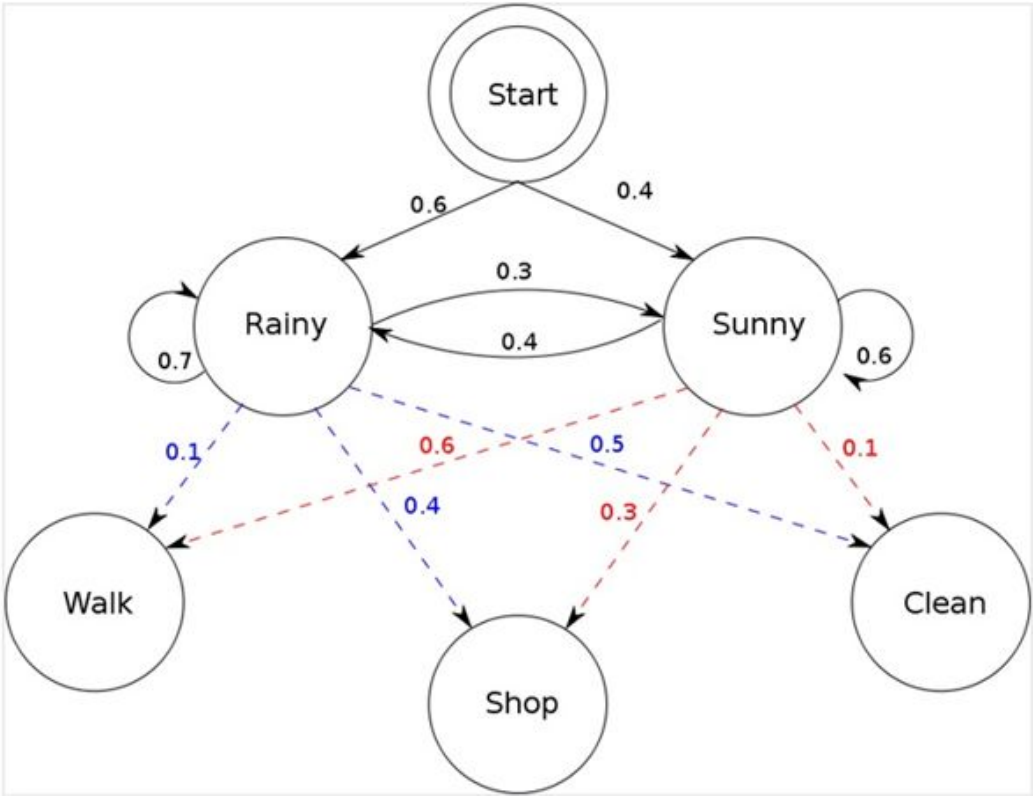


Decoding

- Transition and observation probabilities



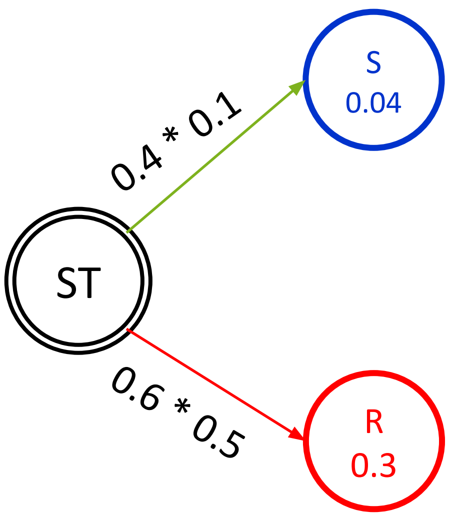
HMM Example: Weather & Activity



Program: viterbi.py

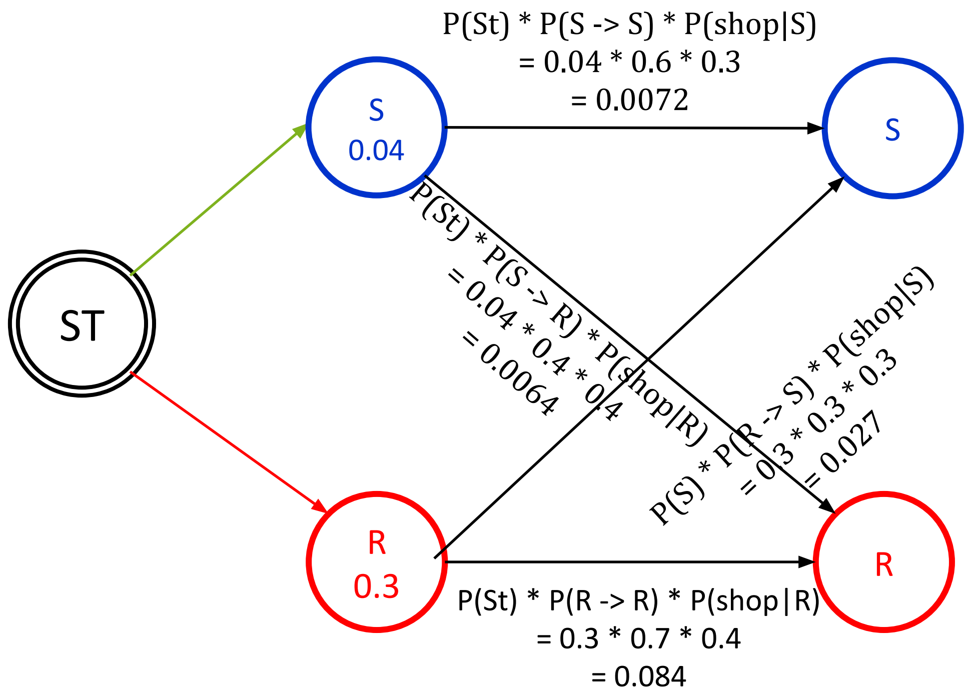
HMM Example: Weather & Activity

Day 1
Observation ‘Clean’



Calculate
P_start(state) * P_obs(‘clean’)

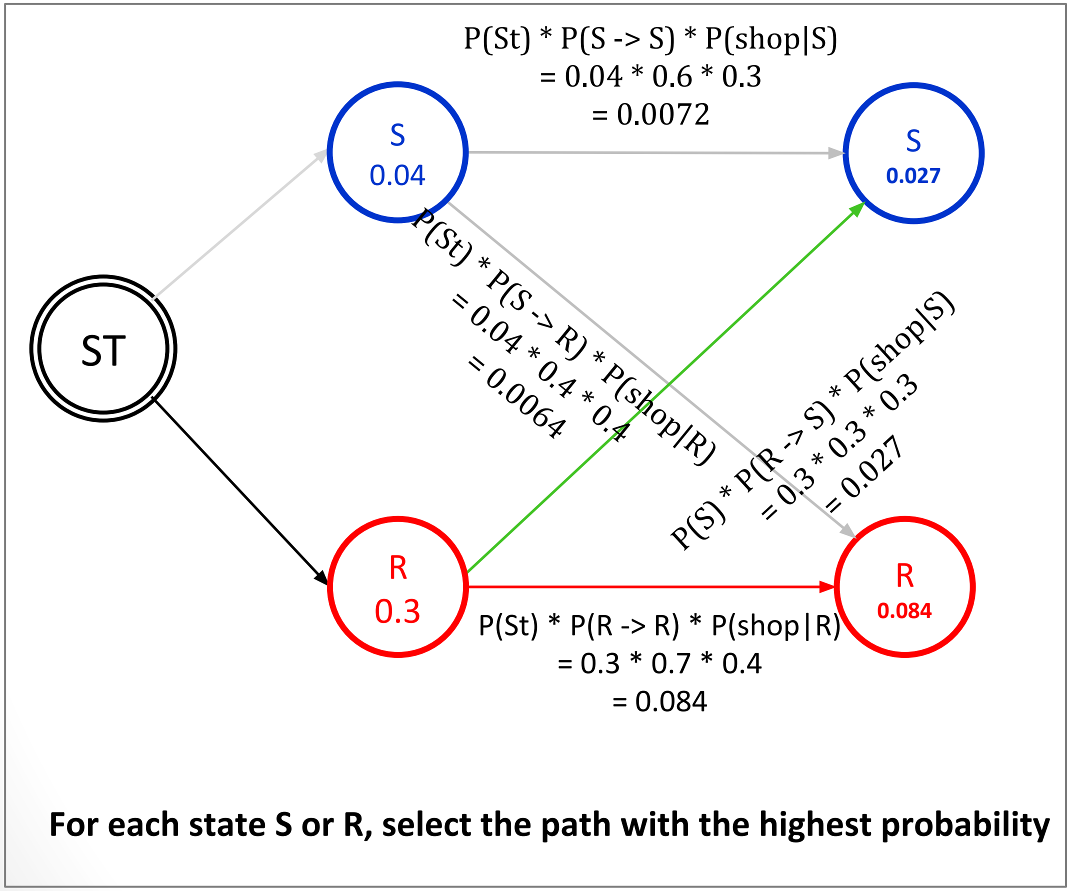
Day 2
Observation ‘Shop’



Calculate
P(oldSt) * P_trans(oldSt-newSt) * P_obs(‘shop’|newSt)

HMM Example: Weather & Activity

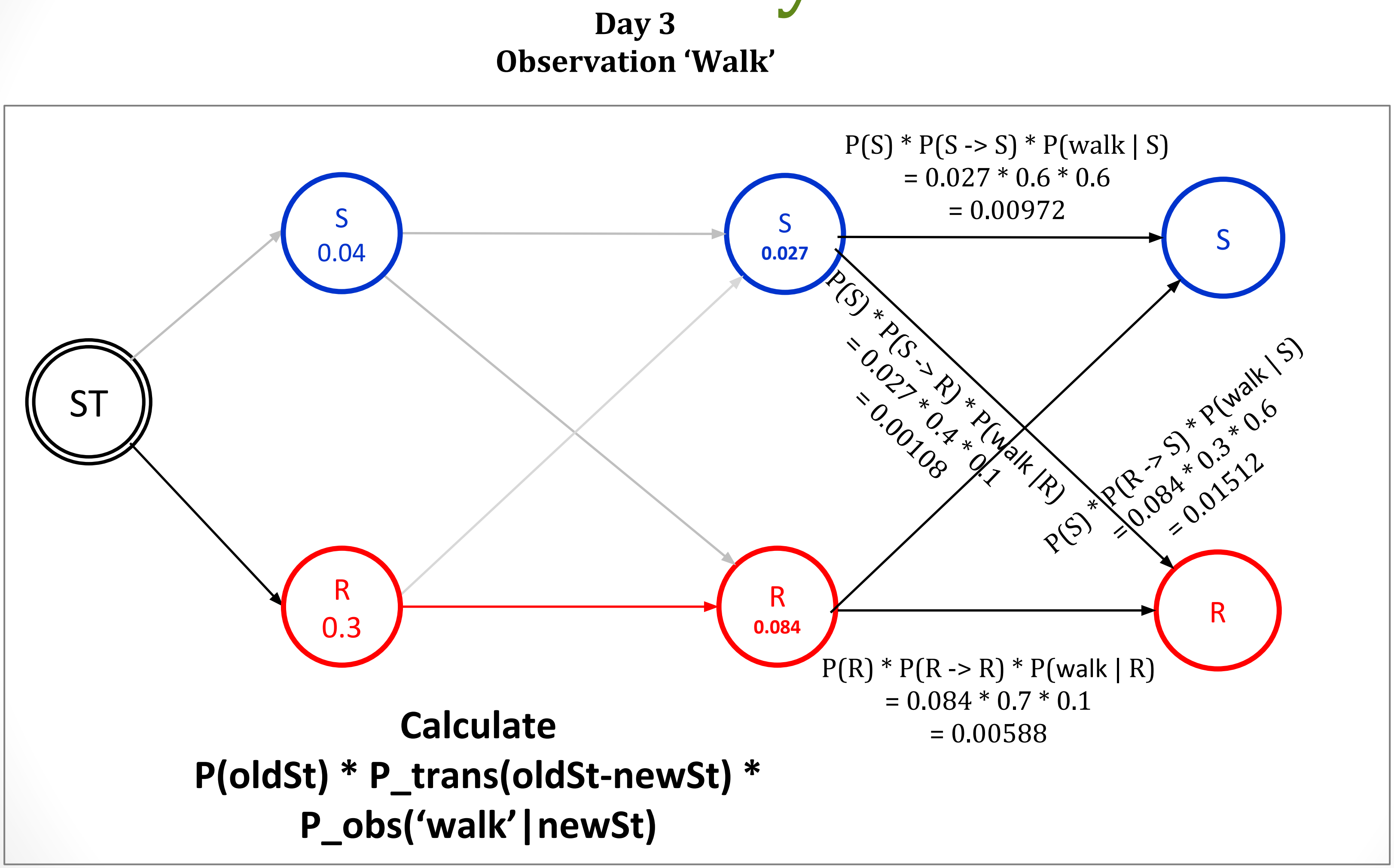
Day 2
Observation 'Shop'



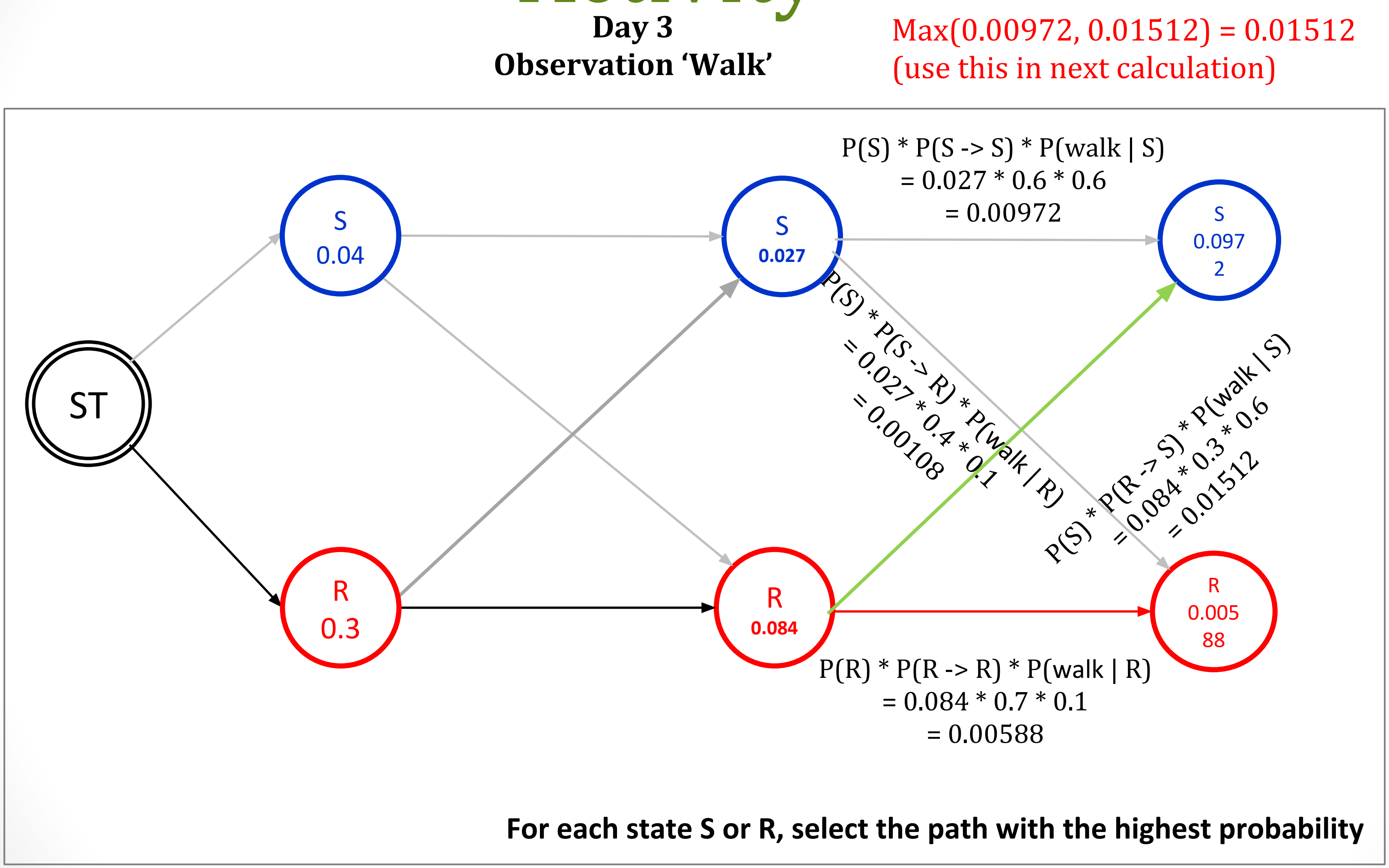
Max(0.0072, 0.027) = **0.027**
(use this in next calculation)

Max(0.0064, 0.084) = **0.084**
(use this in next calculation)

HMM Example: Weather & Activity



HMM Example: Weather & Activity

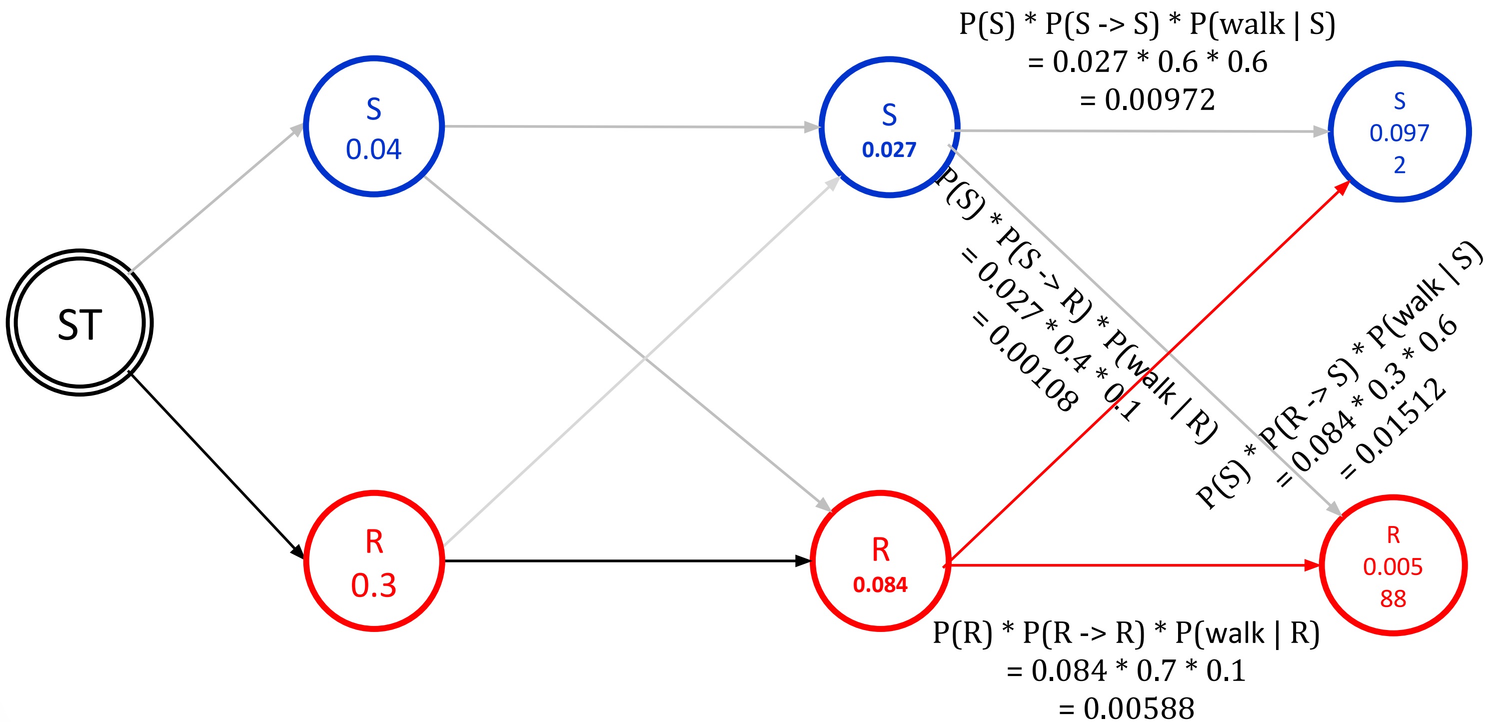


After Day 3, the most likely path is ['Rainy', 'Rainy', 'Sunny']

Max(0.00108, 0.00588) = 0.00588
(use this in next calculation)

HMM Example: Weather & Activity

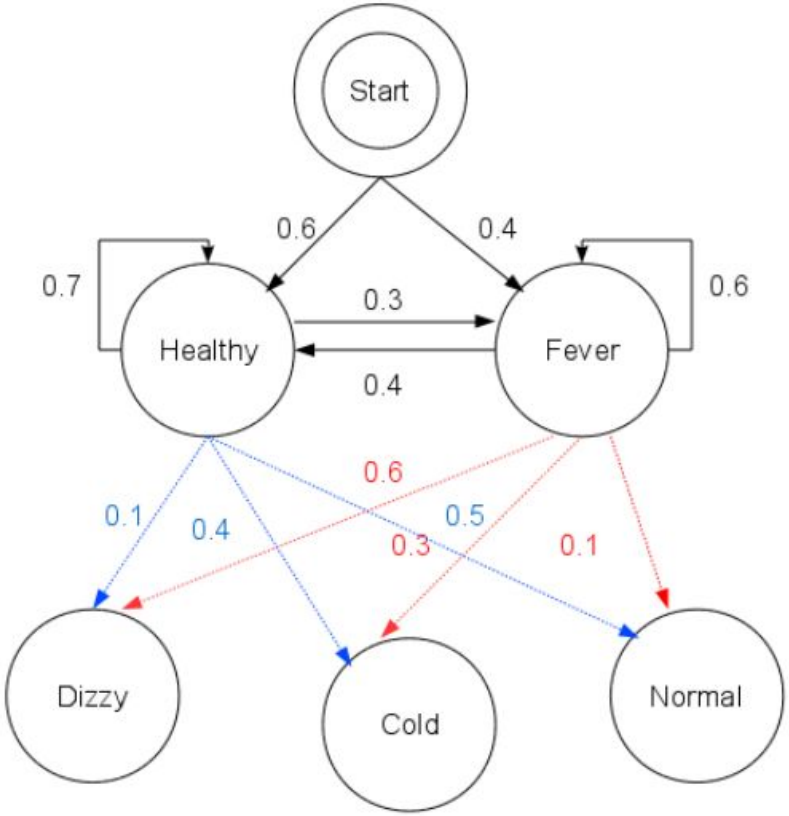
Day 3
Observation 'Walk'



Calculate
P_start(state) * P_obs('clean')

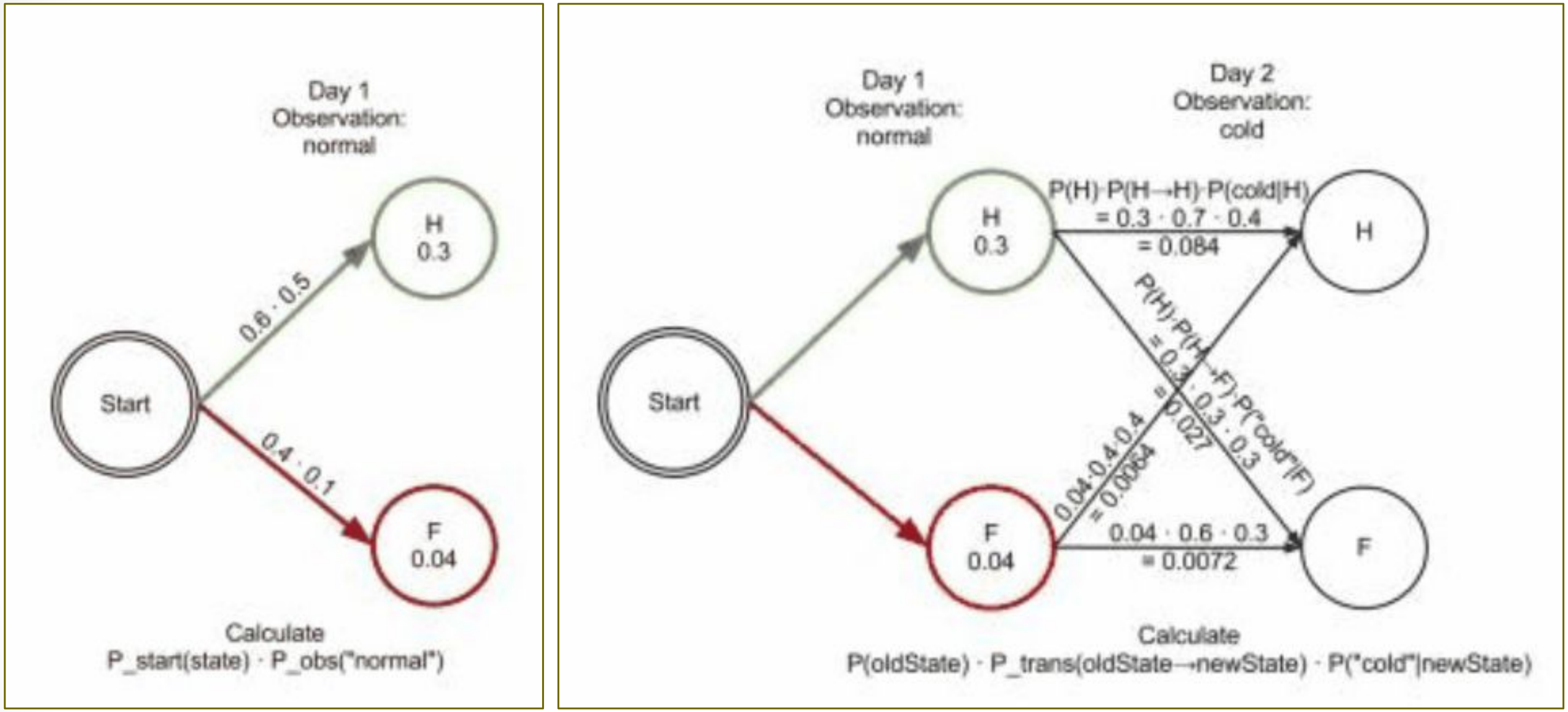
After Day 3, the most likely path is ['Rainy', 'Rainy', 'Sunny']

HMM Example: Health & Condition

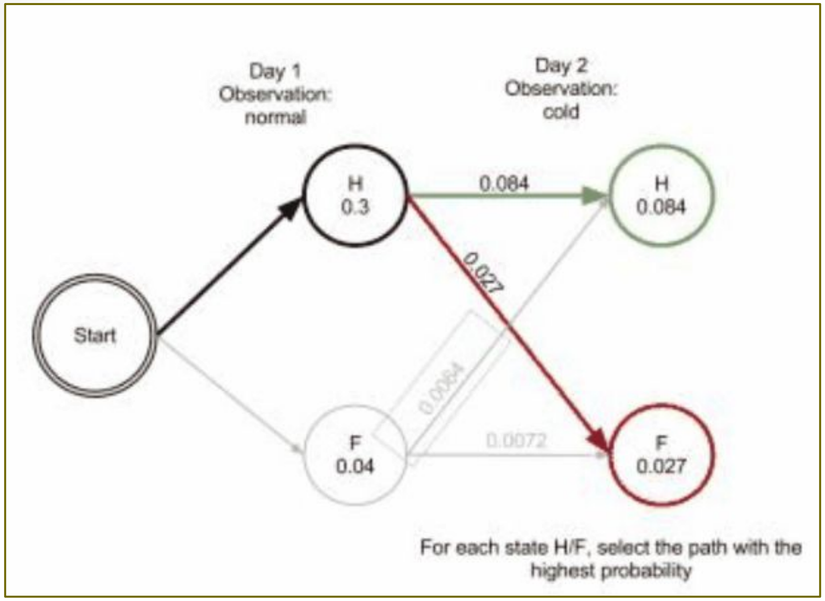


Program: viterbi2.py

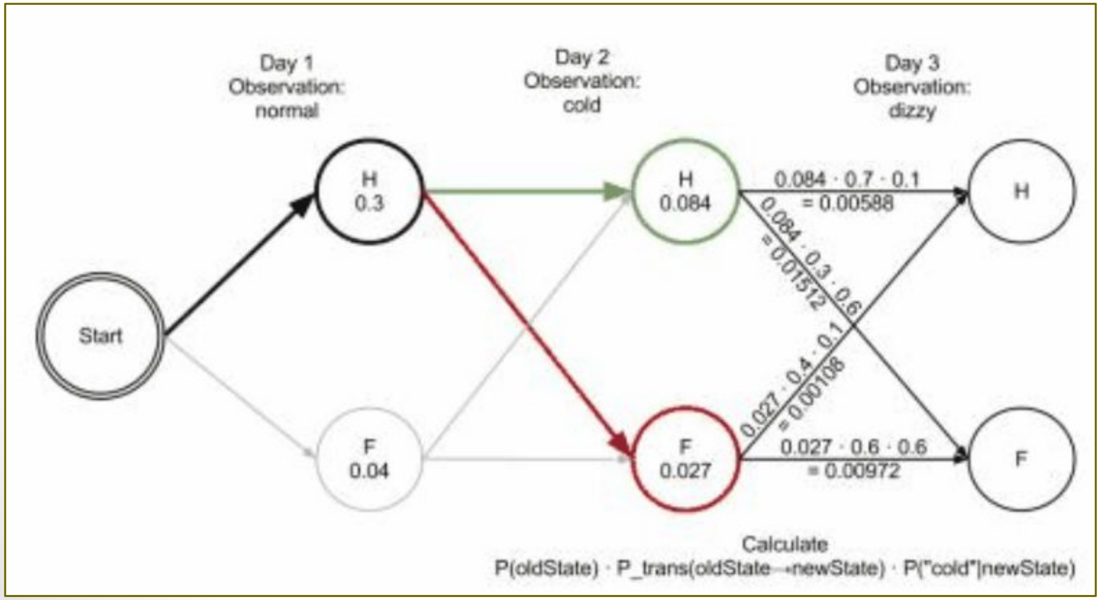
HMM Viterbi Trellis



HMM Viterbi Trellis

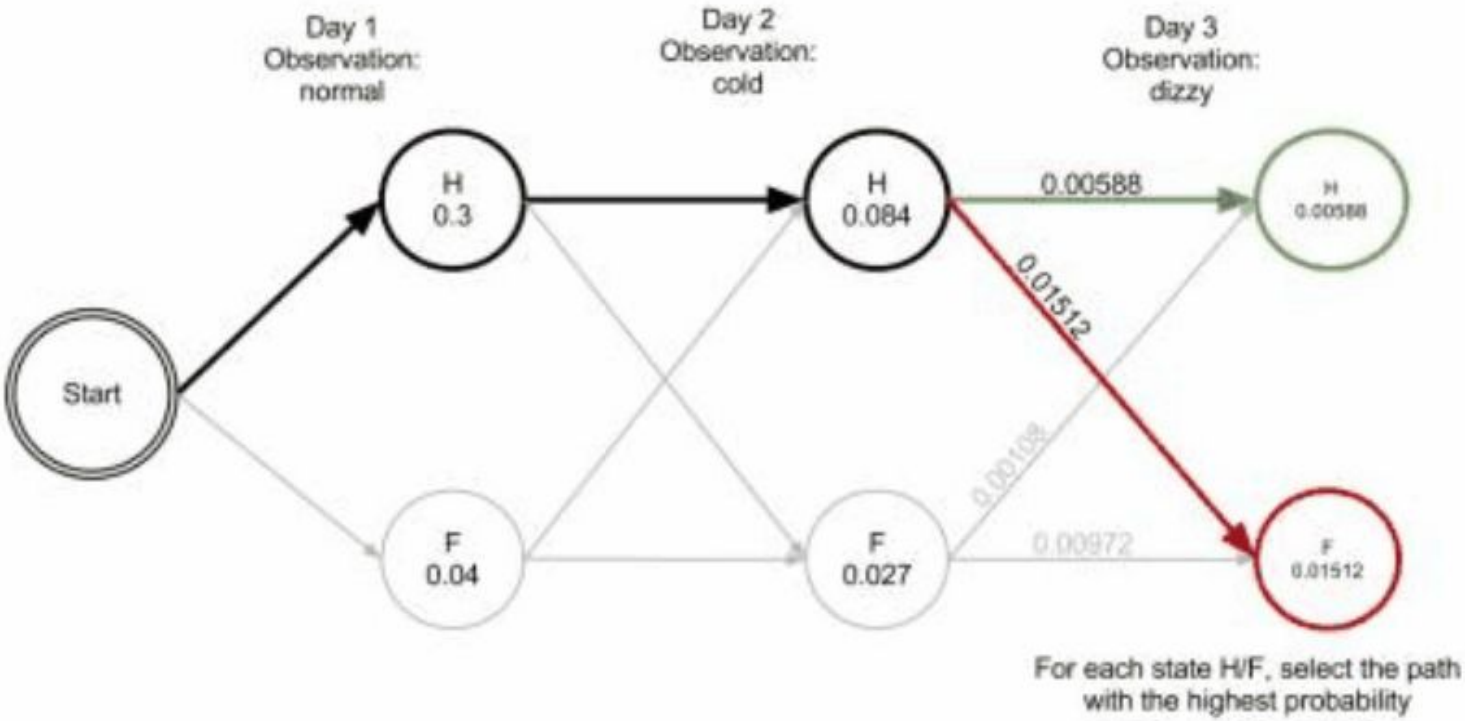


Max(0.084, 0.0064) = **0.084**
(use this in next calculation)



HMM Viterbi Trellis

Max(0.00588, 0.00106)
= **0.00588**



- After Day 3, the most likely path is [‘Healthy’, ‘Healthy’, ‘Fever’]

Group Exercise
Due in next class

Based on the values in Slide 33 :

i. Calculate the probabilities of $v_t(j)$ at each time *t* using the ***viterbi*** algorithm.

ii. Subsequently, modify the program in viterbi2.py to calculate the probabilities of **“The cook prepares a lovely drink”** using the ***viterbi*** algorithm in Python to verify your answer in (i)