Probabilistic Reasoning over Time

- Goal: Represent and reason about changes in the world over time
- Examples:
  - WUMPUS evidence (stench, breeze, scream) arrives over time
  - Monitoring a diabetic patient
  - Inferring the current location of a robot from its sensor data

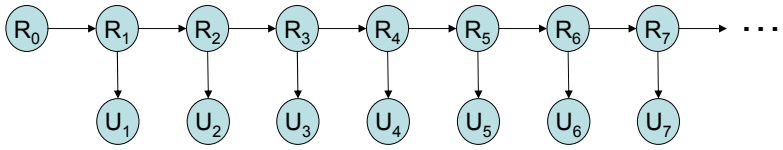
Umbrella World

- Suppose you are a security guard robot at an underground installation. You never go outside, but you would like to know what the weather is.
- Each morning, you see the Director come in. Some mornings he has a wet umbrella; other mornings he has no umbrella.

Notation

- State variables (is it raining on day i?): R_0, R_1, R_2, …
- Evidence variables (is he carrying an umbrella on day i?): U_1, U_2, U_3, …
- $\mathbf{X}_{a:b}$ denotes X_a, X_{a+1}, …, X_{b-1},X_b

Hidden Markov Model

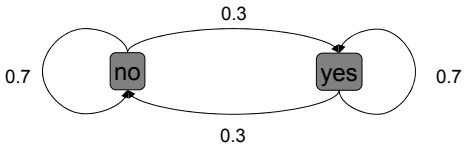


- Markov assumption:
  $P(R_t|R_{1:t-1}) = P(R_t|R_{t-1})$
  Captures the "dynamics" of the world. For example, rainy days and non-rainy days come in "groups"
- Sensor model: $P(U_t|R_t)$
- Stationarity: True for all times t

Probability Distributions

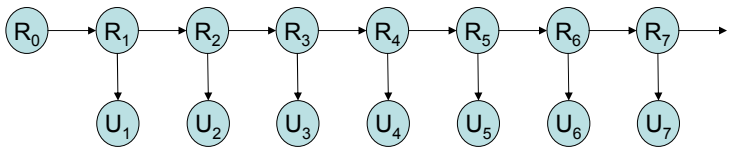
R_t	R_{t-1}=no	R_{t-1}=yes
no	0.7	0.3
yes	0.3	0.7

U_t	R_t=no	R_t=yes
no	0.8	0.1
yes	0.2	0.9



We can view the HMM as a probabilistic finite state machine

Joint Distribution



$\underline{P}(R_{0:n},U_{0:n}) = \underline{P}(R_0) \prod_{t=1} \underline{P}(R_t|R_{t-1}) \cdot \underline{P}(U_t|R_t)$

Can be generalized to multiple state variables (e.g., position, velocity, and acceleration) and multiple sensors (e.g., motor speed, battery level, wheel shaft encoders)

Temporal Reasoning Tasks

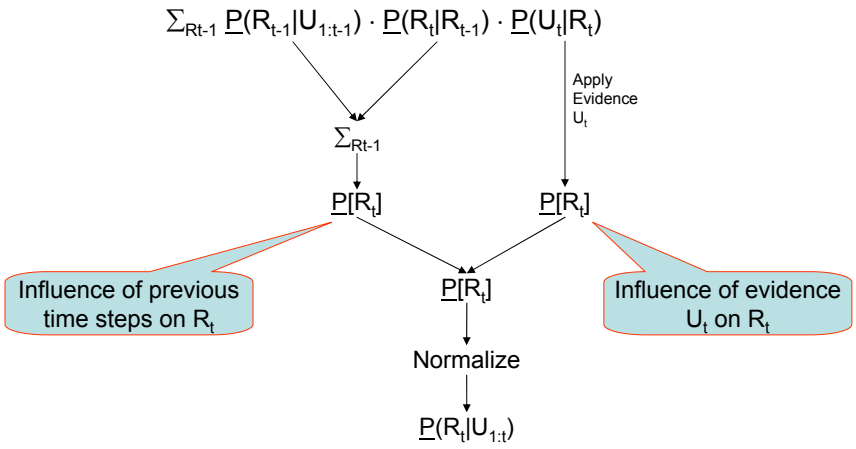
- *Filtering* or *Monitoring*: Compute the *belief state* given the history of sensor readings. $\underline{P}(R_t|U_{1:t})$
- *Prediction*: Predict future state for some $k > 0$. $\underline{P}(R_{t+k}|U_{1:t})$
- *Smoothing*: Reconstruct a previous state given subsequent evidence. $\underline{P}(R_k|U_{1:t})$
- *Most Likely Explanation*: Reconstruct entire sequence of states given entire sequence of sensor readings. $\text{argmax}_{R1:n}\ \underline{P}(R_{1:n}|U_{1:n})$

Filtering by Variable Elimination

$$\underline{P}(R_1|U_1) = \text{Normalize}[\ \text{ApplyEvidence}[U_1, \Sigma_{R0}\ \underline{P}(R_0) \cdot \underline{P}(R_1|R_0) \cdot \underline{P}(U_1|R_1)\ ]\ ]$$
$$= \text{Normalize}[\ \Sigma_{R0}\ \underline{P}(R_0) \cdot \underline{P}(R_1|R_0) \cdot \underline{P}[R_1]\ ]$$
$$= \text{Normalize}[\ \underline{P}[R_1] \cdot \Sigma_{R0}\ \underline{P}(R_0) \cdot \underline{P}(R_1|R_0)\ ]$$
$$= \text{Normalize}[\ \underline{P}[R_1] \cdot \underline{P}[R_1]\ ]$$
$$= \text{Normalize}[\ \underline{P}[R_1]\ ]$$

$$\underline{P}(R_2|U_{1:2}) = \text{Normalize}[\ \text{ApplyEvidence}[\ U_{1:2},$$
$$\Sigma_{R0:1}\ \underline{P}(R_0) \cdot \underline{P}(R_1|R_0) \cdot \underline{P}(U_1|R_1) \cdot \underline{P}(R_2|R_1) \cdot \underline{P}(U_2|R_2)\ ]\ ]$$
$$= \text{Normalize}[\ \Sigma_{R0:1}\ \underline{P}(R_0) \cdot \underline{P}(R_1|R_0) \cdot \underline{P}[R_1] \cdot \underline{P}(R_2|R_1) \cdot \underline{P}[R_2]\ ]$$
$$= \text{Normalize}[\ \Sigma_{R1}\ [\Sigma_{R0}\ \underline{P}(R_0) \cdot \underline{P}(R_1|R_0)] \cdot \underline{P}[R_1] \cdot \underline{P}(R_2|R_1) \cdot \underline{P}[R_2]\ ]$$
$$= \text{Normalize}[\ [\Sigma_{R1}\ \underline{P}[R_1] \cdot \underline{P}[R_1] \cdot \underline{P}(R_2|R_1)] \cdot \underline{P}[R_2]\ ]$$
$$= \text{Normalize}[\ \underline{P}[R_2] \cdot \underline{P}[R_2]]$$
$$= \text{Normalize}[\ \underline{P}[R_2]\ ]$$

General Pattern



The Forward Algorithm

Define:
Forward($\underline{P}(R_{t-1}|U_{1:t-1})$, Ut) =
$\sum_{Rt-1} \underline{P}(R_{t-1}|U_{1:t-1}) \cdot \underline{P}(R_t|R_{t-1}) \cdot$ ApplyEvidence[U_t , $\underline{P}(U_t|R_t)$]

Then filtering can be written recursively as:
$\underline{P}(R_t|U_{1:t})$ = Normalize[Forward($\underline{P}(R_{t-1}|U_{1:t-1})$, U_t)]

In general, we can iterate over multiple time steps:
Forward($\underline{P}(R_i|U_{1:i-1})$, $U_{i:t}$) =
Forward(Forward($\underline{P}(R_i|U_{1:i-1})$, U_i), $U_{i+1:t}$) while $i \leq t$

Example: Day 1

- day 1: Umbrella. U_1 = yes
  $\underline{P}(R_1)$ = Normalize[Forward($\underline{P}(R_0)$,yes)]

Normalize[$\sum_{R0}$

R_0	$P(R_0)$
no	0.5
yes	0.5

·

R_1	R_0=no	R_0=yes
no	0.7	0.3
yes	0.3	0.7

·

U_1	R_1=no	R_1=yes
no	0.8	0.1
yes	0.2	0.9

]

Normalize[$\sum_{R0}$

R_1	R_0=no	R_0=yes
no	0.7 * 0.5	0.3 * 0.5
yes	0.3 * 0.5	0.7 * 0.5

·

U_1	R_1=no	R_1=yes
no	0.8	0.1
yes	0.2	0.9

]

Example: Day 1 (continued)

Normalize[$\sum_{R0}$

R_1	R_0=no	R_0=yes
no	0.35	0.15
yes	0.15	0.35

·

U_1	R_1=no	R_1=yes
no	0.8	0.1
yes	0.2	0.9

]

Normalize[

R_1	
no	0.50
yes	0.50

·

U_1	R_1=no	R_1=yes
no	0.8	0.1
yes	0.2	0.9

]

Normalize[

R_1	$P(R_1)$
no	0.10
yes	0.45

] =

R_1	$P(R_1)$
no	0.18
yes	0.82

Example: Day 2

- Day 2: U_2 = yes

Normalize[$\sum_{R1}$

R_1	$P(R_1)$
no	0.18
yes	0.82

·

R_2	R_1=no	R_1=yes
no	0.7	0.3
yes	0.3	0.7

·

U_2	R_2=no	R_2=yes
no	0.8	0.1
yes	0.2	0.9

]

Normalize[$\sum_{R1}$

R_2	R_1=no	R_1=yes
no	0.7 * 0.18	0.3 * 0.82
yes	0.3 * 0.18	0.7 * 0.82

·

U_2	R_2=no	R_2=yes
no	0.8	0.1
yes	0.2	0.9

]

Normalize[$\sum_{R1}$

R_2	R_1=no	R_1=yes
no	0.127	0.245
yes	0.055	0.573

·

U_2	R_2=no	R_2=yes
no	0.8	0.1
yes	0.2	0.9

] =

Day 2 (continued)

Normalize[$\sum_{R1}$

R_2	R_1=no	R_1=yes
no	0.127	0.245
yes	0.055	0.573

·

U_2	R_2=no	R_2=yes
no	0.8	0.1
yes	0.2	0.9

] =

Normalize[

R_2	
no	0.373
yes	0.627

·

U_2	R_2=no	R_2=yes
no	0.8	0.1
yes	0.2	0.9

] =

Normalize[

R_2	
no	0.075
yes	0.565

] =

R_2	$P(R_2)$
no	0.116
yes	0.883

Prediction: Multiply by the Transition Probabilities and Sum Away

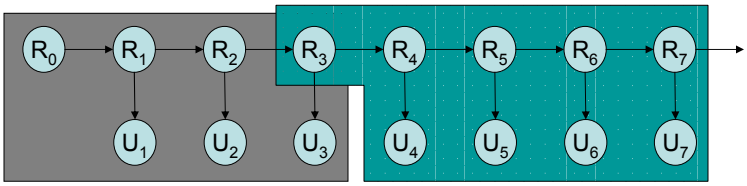
- $\underline{P}(R_{t+k} \mid U_{1:t}) = \sum_{Rt:t+k-1} \underline{P}(R_t \mid U_{1:t}) \cdot \underline{P}(R_{t+1}|R_t) \cdot \underline{P}(R_{t+2}|R_{t+1}) \cdot \ldots \cdot \underline{P}(R_{t+k}|R_{t+k-1})$

- $\underline{P}(R_{t+1} \mid U_{1:t}) = \sum_{Rt} \underline{P}(R_t \mid U_{1:t}) \cdot \underline{P}(R_{t+1}|R_t)$
- $\underline{P}(R_{t+2} \mid U_{1:t}) = \sum_{Rt+1} \underline{P}(R_{t+1} \mid U_{1:t}) \cdot \underline{P}(R_{t+2}|R_{t+1})$
- …

Question: What Happens if We Predict Far Into the Future?

- Each multiplication by $\underline{P}(R_{t+1}|R_t)$ makes our predictions "fuzzier". Eventually, (for this problem) they converge to $\langle 0.5,0.5 \rangle$. This is called the *stationary distribution* of the Markov process. Much is known about the stationary distribution and the rate of convergence. The stationary distribution depends on the transition probability distribution.

Smoothing: Reconstructing R_k given $U_{1:t}$



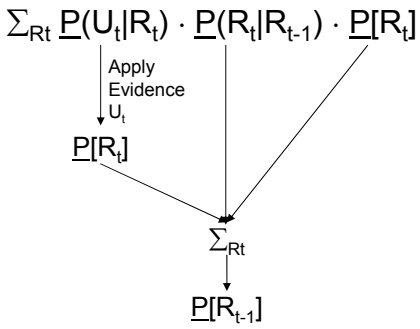
Forward→ ←Backward

Assume k < t.

Example: k=3, t=7:

$\underline{P}(R_3|U_{1:7}) = \text{Normalize}[$
$\quad \text{ApplyEvidence}[U_{1:7}, \underline{P}(R_3|U_{1:3}) \cdot \underline{P}(U_{4:7}|R_3)]]$

The Backward Algorithm

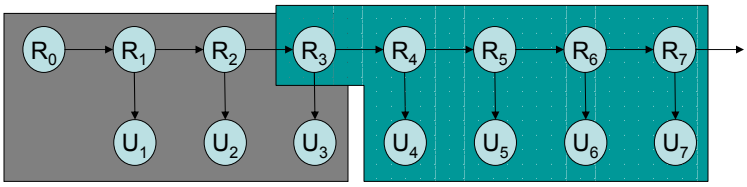


The Backward Algorithm (2)

Backward($\underline{P}[R_t]$, U_t)=
$\sum_{Rt}$ ApplyEvidence[U_t, $\underline{P}(U_t|R_t)$] · $\underline{P}(R_t|R_{t-1})$ · $\underline{P}[R_t]$

This can then be applied recursively
$\underline{P}[R_{t-1}]$ = Backward($\underline{P}[R_t]$, U_t)

Forward-Backward Algorithm for Smoothing



$\underline{P}(R_k|U_{1:t})$ = Normalize[Forward($\underline{P}(R_0)$, $U_{1:k}$) ·
Backward($\underline{1}$, $U_{k+1:t}$)]

Umbrella Example: $\underline{P}(R_1|U_{1:2})$

Normalize[Forward($\underline{P}(R_0)$, U_1) · Backward($\underline{1}$, U_2)]

Forward($\underline{P}$(R0), U1) =

R_1	$P(R_1)$
no	0.18
yes	0.82

Backward($\underline{1}$, U1) = $\sum_{R2}\ \underline{1} \cdot \underline{P}(R_2|R_1) \cdot \underline{P}(U_2|R_2)$

Backward from Day 2
U_2 = yes

$\sum_{R2}$

R_2	$P[R_2]$
no	1
yes	1

·

R_2	R_1=no	R_1=yes
no	0.7	0.3
yes	0.3	0.7

·

U_2	R_2=no	R_2=yes
no	0.8	0.1
yes	0.2	0.9

$\sum_{R2}$

R_2	R_1=no	R_1=yes
no	0.7 * 1 * 0.2	0.3 * 1 * 0.2
yes	0.3 * 1 * 0.9	0.7 * 1* 0.9

$\sum_{R2}$

R_2	R_1=no	R_1=yes
no	0.14	0.06
yes	0.27	0.63

=

R_1	$P[R_1]$
no	0.41
yes	0.69

Forward-Backward:

Normalize[

R_1	$P(R_1)$
no	0.18
yes	0.82

.

R_1	$P[R_1]$
no	0.41
yes	0.69

] =

Normalize[

R_1	$P(R_1)$
no	0.074
yes	0.566

] =

R_1	$P(R_1)$
no	0.115
yes	0.885

Notice that $P(R_1=yes|U_1=yes) < P(R_1=yes|U_1=yes,U_2=yes)$

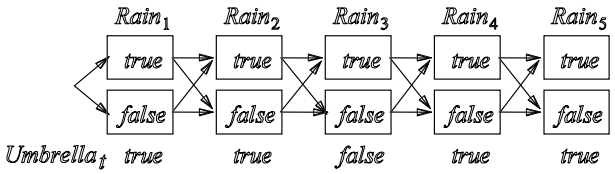
Evidence from the future allows us to revise our beliefs about the past.

Most Likely Explanation

- Find $\text{argmax}_{R1:n} \underline{P}(R_{1:n}|U_{1:n})$
  - Note that this is the maximum over all *sequences* of rain states: $R_{1:n}$
  - There are 2^n such sequences!
  - Fortunately, there is a dynamic programming algorithm: the Viterbi Algorithm

Viterbi Algorithm

- Suppose we observe ⟨yes,yes,no,yes,yes⟩ for $U_{1:5}$
- Our goal is to find the best path through a "trellis" of possible rain states:



Max distributes over conformal product

$\max_{A,B,C}$

B	A=no	A=yes
no	0.10	0.20
yes	0.30	0.40

.

C	B=no	B=yes
no	0.10	0.35
yes	0.40	0.15

=

B	C	A=no	A=yes
no	no	0.10*0.10	0.20*0.10
no	yes	0.10*0.40	0.20*0.40
yes	no	0.30*0.35	0.40*0.35
yes	yes	0.30*0.15	0.40*0.15

=

B	C	A=no	A=yes
no	no	0.010	0.020
no	yes	0.040	0.080
yes	no	0.105	0.140
yes	yes	0.045	0.060

Max propagation

$\max_{A,B}$

B	A=no	A=yes
no	0.10	0.20
yes	0.30	0.40

$\cdot$ $\max_{C}$

C	B=no	B=yes
no	0.10	0.35
yes	0.40	0.15

=

$\max_{A,B}$

B	A=no	A=yes
no	0.10	0.20
yes	0.30	0.40

$\cdot$

B=no	B=yes
0.40	0.35

=

B	A=no	A=yes
no	0.10*0.40	0.20*0.40
yes	0.30*0.35	0.40*0.35

=

B	A=no	A=yes
no	0.040	0.080
yes	0.105	0.140

Follow the Maxes

$\max_{A,B,C}$

B	A=no	A=yes
no	0.10	0.20
yes	0.30	0.40

$\cdot$

C	B=no	B=yes
no	0.10	0.35
yes	0.40	0.15

=

B	C	A=no	A=yes
no	no	0.10*0.10	0.20*0.10
no	yes	0.10*0.40	0.20*0.40
yes	no	0.30*0.35	0.40*0.35
yes	yes	0.30*0.15	0.40*0.15

=

B	C	A=no	A=yes
no	no	0.010	0.020
no	yes	0.040	0.080
yes	no	0.105	0.140
yes	yes	0.045	0.060

Because the “losers” (0.10 and 0.15) will be multiplied against the same values as the “winners” (0.40 and 0.35), they can never be the overall winners.

Extracting the Maximum Configuration

- Remember the winning combinations

$\max_{A,B}$

B	A=no	A=yes
no	0.10	0.20
yes	0.30	0.40

$\cdot \max_{C}$

C	B=no	B=yes
no	0.10	0.35
yes	0.40	0.15

=

$\max_{A,B}$

B	A=no	A=yes
no	0.10	0.20
yes	0.30	0.40

$\cdot$

B=no	B=yes
0.40	0.35
C=yes	C=no

=

B	A=no	A=yes
no	0.10*0.40	0.20*0.40
yes	0.30*0.35	0.40*0.35

=

B	A=no	A=yes
no	0.040	0.080
yes	0.105	0.140

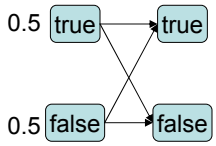
(B=yes,A=yes) is winner of final table. Corresponding value is C=no

Viterbi Algorithm

$$\max_{R0:2} \underline{P}(R_{0:2}|U_{1:2}) =$$
$$\max_{R0:2} P(R_0) \cdot P(R_1|R_0) \cdot P(U_1|R_1) \cdot P(R_2|R_1) \cdot P(U_2|R_2) =$$
$$\max_{R2} P(U_2|R_2) \cdot [\max_{R1} P(U_1|R_1) \cdot P(R_2|R_1) \cdot [\max_{R0} P(R_0) \cdot P(R_1|R_0)]] =$$

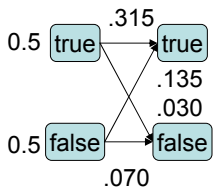
Viterbi

- $[\max_{R0} \underline{P}(R_0) \cdot \underline{P}(R_1|R_0)] \cdot \underline{P}(U_1|R_1)$



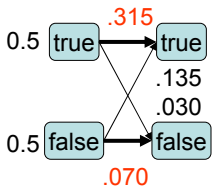
Viterbi

- $[\max_{R0} \underline{P}(R_0) \cdot \underline{P}(R_1|R_0)] \cdot \underline{P}(U_1|R_1)$



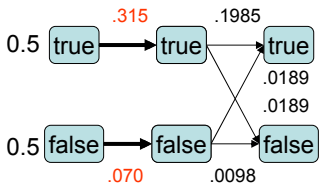
Viterbi

- $[\max_{R0} \underline{P}(R_0) \cdot \underline{P}(R_1|R_0)] \cdot \underline{P}(U_1|R_1)$



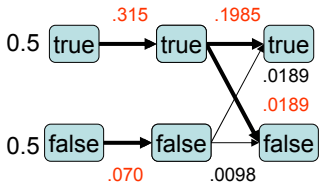
Viterbi

- $[\max_{R1} \underline{P}[R_1] \cdot \underline{P}(R_2|R_1)] \cdot \underline{P}(U_2|R_2)$



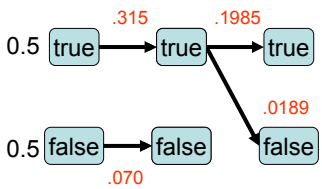
Viterbi

- $[\max_{R2} \underline{P}[R_1] \cdot \underline{P}(R_2|R_1)] \cdot \underline{P}(U_2|R_2)$



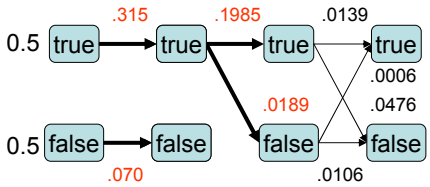
Viterbi

- $[\max_{R2} \underline{P}[R_1] \cdot \underline{P}(R_2|R_1)] \cdot \underline{P}(U_2|R_2)$



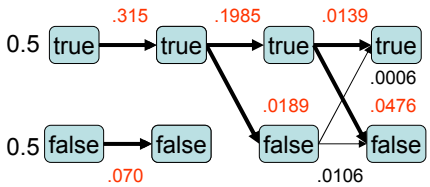
Viterbi

- $[\max_{R3} \underline{P}[R_2] \cdot \underline{P}(R_3|R_2)] \cdot \underline{P}(U_3|R_3)$



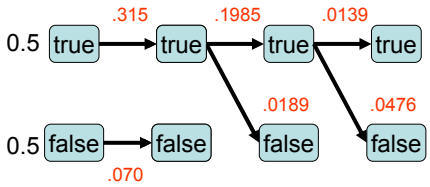
Viterbi

- $[\max_{R3} \underline{P}[R_2] \cdot \underline{P}(R_3|R_2)] \cdot \underline{P}(U_3|R_3)$



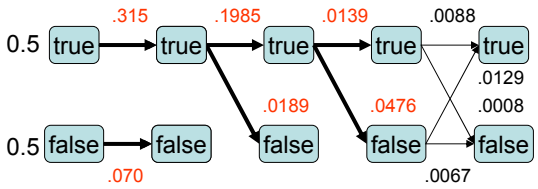
Viterbi

- $[\max_{R3} \underline{P}[R_2] \cdot \underline{P}(R_3|R_2)] \cdot \underline{P}(U_3|R_3)$



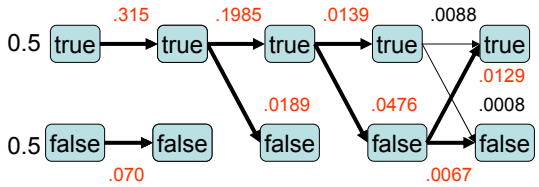
Viterbi

- $[\max_{R4} \underline{P}[R_3] \cdot \underline{P}(R_4|R_3)] \cdot \underline{P}(U_4|R_4)$



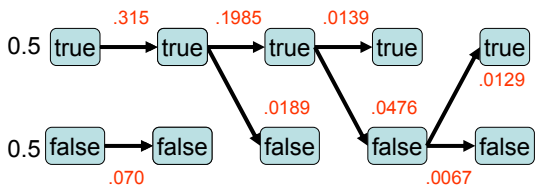
Viterbi

- $[\max_{R4} \underline{P}[R_3] \cdot \underline{P}(R_4|R_3)] \cdot \underline{P}(U_4|R_4)$



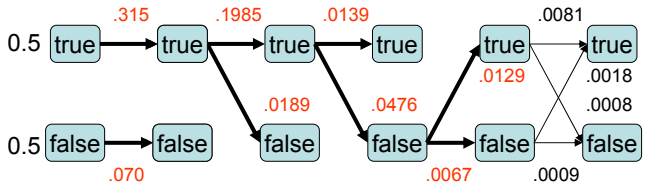
Viterbi

- $[\max_{R4} \underline{P}[R_3] \cdot \underline{P}(R_4|R_3)] \cdot \underline{P}(U_4|R_4)$



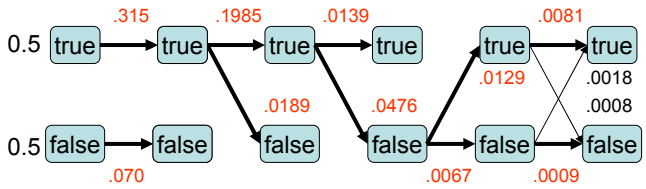
Viterbi

- $[\max_{R4} \underline{P}[R_4] \cdot \underline{P}(R_5|R_4)] \cdot \underline{P}(U_5|R_5)$



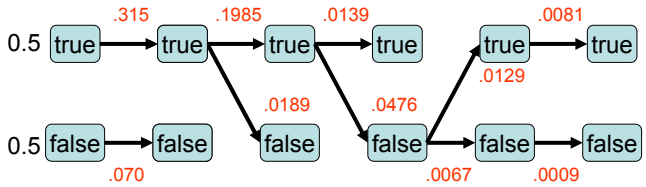
Viterbi

- $[\max_{R4} \underline{P}[R_4] \cdot \underline{P}(R_5|R_4)] \cdot \underline{P}(U_5|R_5)$



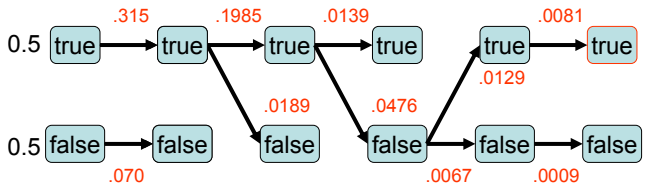
Viterbi

- $[\max_{R4} \underline{P}[R_4] \cdot \underline{P}(R_5|R_4)] \cdot \underline{P}(U_5|R_5)$



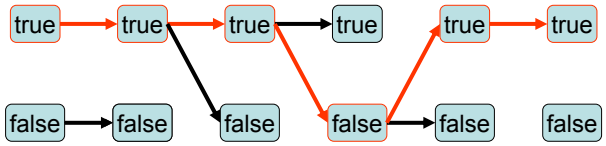
Viterbi

- $\max_{R5} \underline{P}[R_5]$



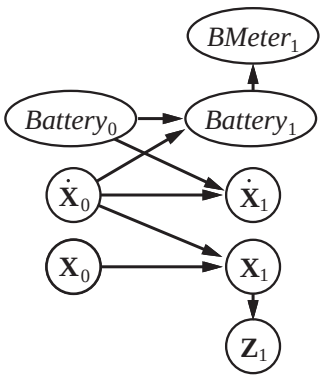
Viterbi

- Traceback



Dynamic Bayesian Networks

- Multiple State Variables and Multiple Sensors
- Robot state variables:
  - Position X_t
  - Velocity $Xdot_t$
  - Battery power
- Sensors
  - Battery meter
  - GPS sensor
- DBN captures sparseness in the interactions among the variables



Inference for DBNs

- Problem: The cost of inference for DBNs is generally exponential in the number of state variables.
- Solution: Approximate Inference using Particle Filters

Particle Filters

- Key idea: Represent

  $\underline{P}(X_t, Xdot_t, Batt_t | Z_{1:t}, BM_{1:t})$

  as a set of points ("particles")
- Implement the Forward algorithm by simulating the behavior of these points

Particle Filtering (we will use HMMs for simplicity)

- HMM: $\underline{P}(X_t|X_{t-1})$; $\underline{P}(Z_t|X_t)$; $\underline{P}(X_1)$
- At each time t, we will have a set of points $S_t = \{x_1, \ldots, x_N\}$ that represent $\underline{P}(X_t|Z_{1:t})$.
- Step 1: Apply $\underline{P}(X_{t+1}|X_t)$: Push each point "forward" in time $x_i \sim \underline{P}(X_{t+1} \mid x_i)$
- Step 2: Apply evidence. Assign a weight to each point: $w_i = \underline{P}(Z_{t+1}|x_i)$
- Step 3: Normalize by drawing a new sample according to weight w_i.
  - Let $W = \sum_i w_i$ be the total amount of weight.
  - Draw N points *with replacement* from $S = \{x_i\}$, where point x_i has probability w_i/W of being chosen.

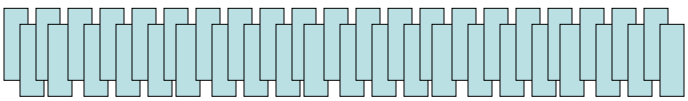
More on Particle Filters

- Sebastian Thrun (cs.stanford.edu)
- Dieter Fox (cs.washington.edu)

Speech Recognition

- Given speech signal
- Determine most likely sequence of words spoken

Levels of Analysis

Words	You say “tomato”
Phones	[ih] [y] [uw] [s] [eh] [t] [ah] [m] [ey] [t] [ow]
Overlapping frames	
Raw signal	

Signal Processing

- Divide speech signal into short chunks (e.g., 10ms) called “frames”
  - Frames overlap by 5ms
- Extract from each frame a vector of real-valued “features”
  - Frequency x Energy features (“Cepstral coefficients”)
  - Changes in these, etc.

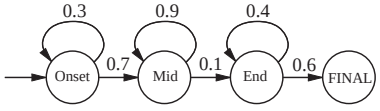
Generative Model of Frames

- P(frame | phone)
  - Vector Quantization: Discretize frames by clustering them into 256 clusters.
    - Frame becomes single 256-valued variable
  - Model frame as a mixture of multi-variate Gaussian random variables whose mean and variance depends on the phone.

HMM Models of Phones

- A phone lasts 50-100 ms (= 10-20 frames)
  - Different pronunciations, speaking rates

Phone HMM for [m]:



Here, C_1, C_2, etc. are frame cluster numbers

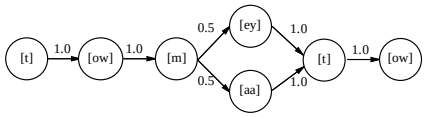
Output probabilities for the phone HMM:

Onset:	Mid:	End:
C_1: 0.5	C_3: 0.2	C_4: 0.1
C_2: 0.2	C_4: 0.7	C_6: 0.5
C_3: 0.3	C_5: 0.1	C_7: 0.4

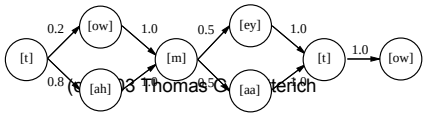
HMM Models of Words

- A word may produce more than one possible phone sequence
  - Different pronunciations: "[t][ah][m][ey][t][ow]" versus "[t][ah][m][aa][t][ow]"
  - Coarticulation effects: "[t][ah][m][ey][t][ow]" versus "[t][ow][m][ey][t][ow]"

(a) Word model with dialect variation:



(b) Word model with coarticulation and dialect variations



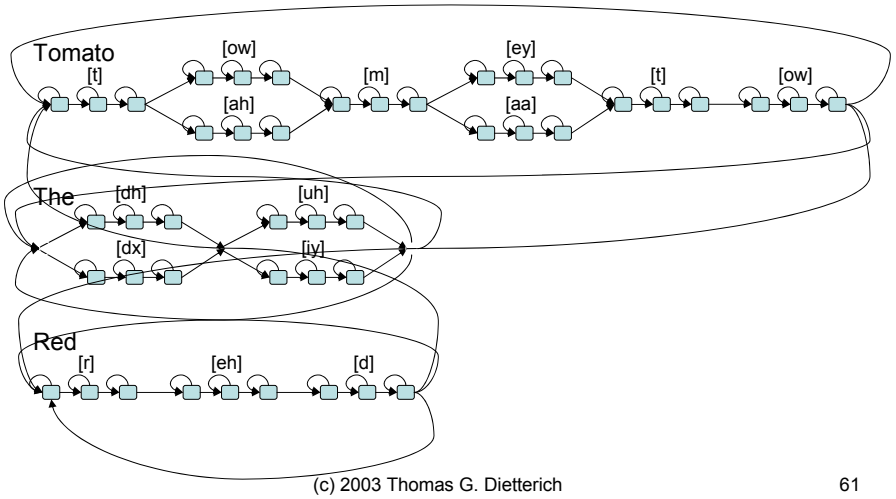
Language Model

- Bigram or Trigram Models

“Macro Expanding”

- We can combine the language model, word models, and phone models to obtain a very large HMM that contains only phones and frames

Fragment of the Flattened Phone Model – Each state generates frames



Learning the Model Parameters

- Fully-supervised: Manually label frames with phone states (onset, middle, end)
  - Very time-consuming
- Abstract supervision: Label each sentence with the sequence of words spoken
  - Treat phones as hidden variables
  - Apply EM algorithm for learning Bayesian networks with missing variables

Speech Recognition

- Viterbi algorithm finds most likely path through the flattened HMM
  - Does not necessarily find the most likely sequence of words. Why not?
- Beam Search
  - Too expensive to compute: Branching factor of 20,000
  - Keep track of the B most likely states in the HMM at each time t
- "It's hard to wreck a nice beach"