Time Series Analysis

ECE 6563, Electrical and Computer Engineering
University of New Brunswick

Recap

In the last section, we expanded our framework to include state space models

- There is a hidden or latent process, x_t, called the state process.
  Assumed to be a Markov process
- There are observations, y_t, that are generated by the state
  but are otherwise independent
- Linear state space models are a superset of our SARIMAX family of models
  We can rewrite them in state space form
- Dynamic Linear regression
  The weighting coefficients are allowed to change over time
- Kalman Filters
  Enables optimal estimation of state in the presence of noisy measurements

In this section, we'll explore Hidden Markov Models

- Markov Chains
- Viterbi, etc.

State Space Models

When we introduced state space models, we said that they followed two main principles:

1. There is a *hidden* or *latent* process, x_t, called the state process.
   - The state process is assumed to be a Markov process
2. The observations, y_t, are independent given the states x_t
   - Any relationship between observations is dictated by the originating states

We've been focused on linear Gaussian state space models, but this is just one (albeit popular) case.

Another well-known case is when the hidden state, x_t, follows a discrete-valued Markov chain.

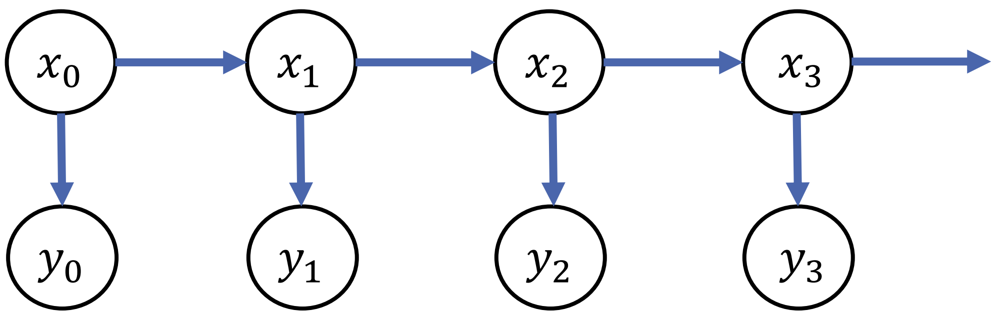
- The distribution of the possible observations at some time, t, is dictated by the current state at that time.
- The current state is one of m possible states, and the states evolve according to a Markov Chain over time.
- When the underlying state is unknown (we only have the observations), it is termed a hidden Markov model (HMM)

Markov Process & Chains

A Markov process is a stochastic process whose future value is only dependent on its current state - not its past.

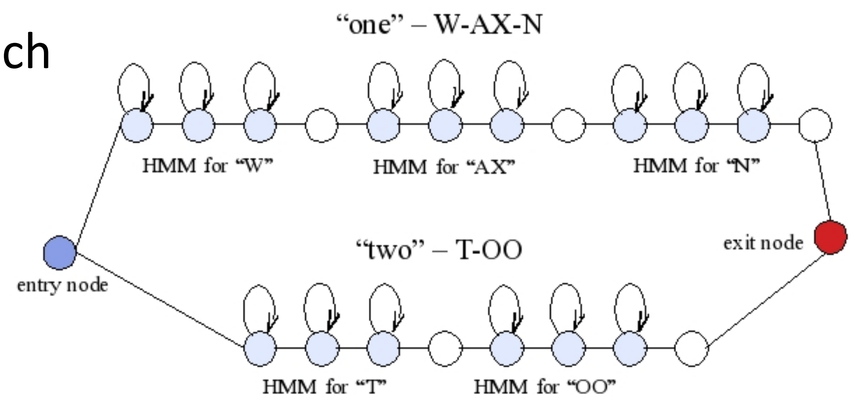
$$p(x_t|x_{t-1}, x_{t-2}, \dots x_{t-\infty}) = p(x_t|x_{t-1})$$

- A discrete Markov chain is a specific type of Markoff process, which has a countable number of states.
- Time can also be continuous or discrete (but we'll focus on discrete time)
- In a hidden Markov model, the state is hidden, and we only see a corresponding observation

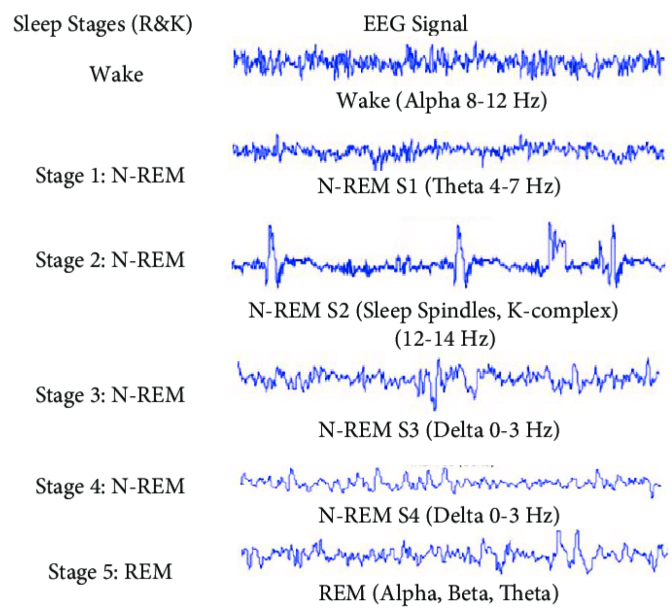


Markov Process & Chains

- Hidden States: Phonemes of speech
- Observations: Acoustic Spectra



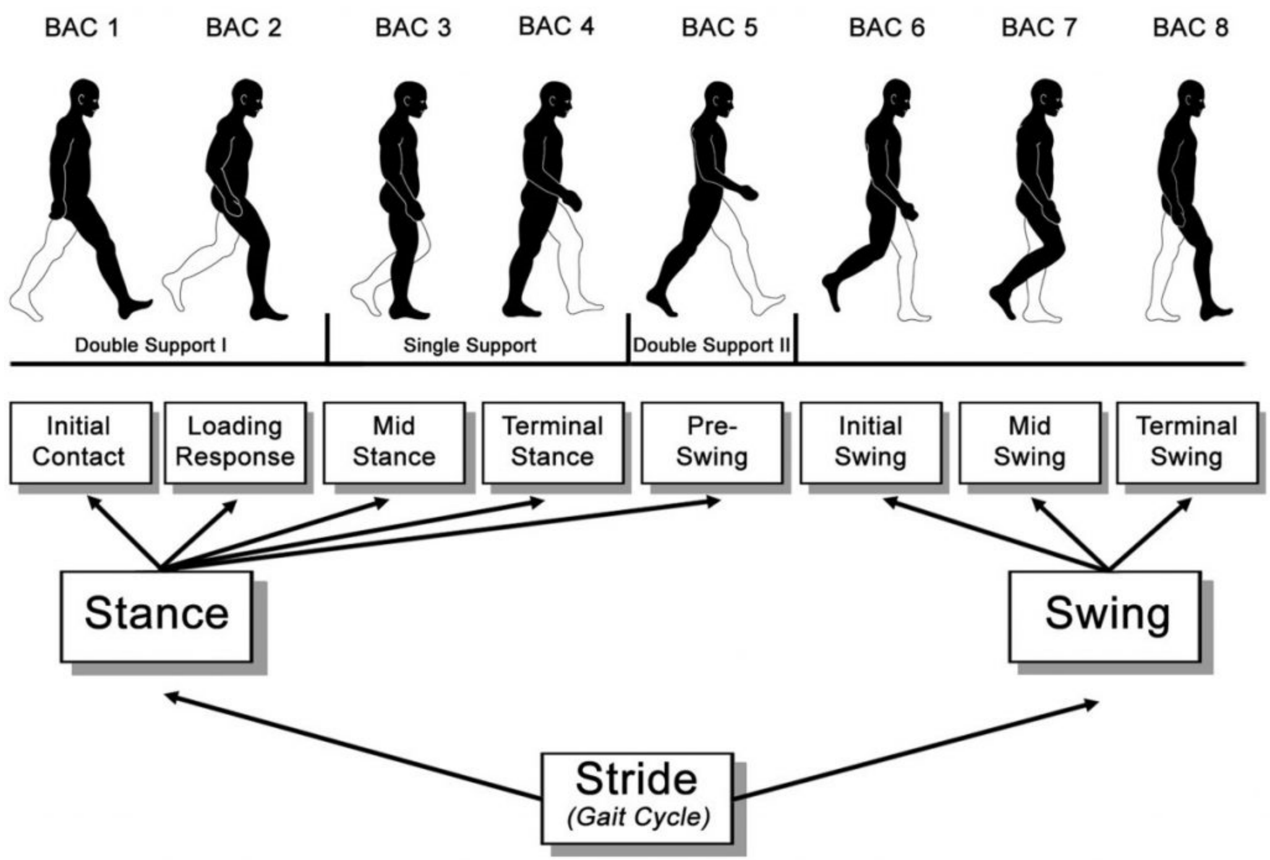
- Hidden States: Stages of sleep
- Observations: EEG signals



Markov Process & Chains

Examples:

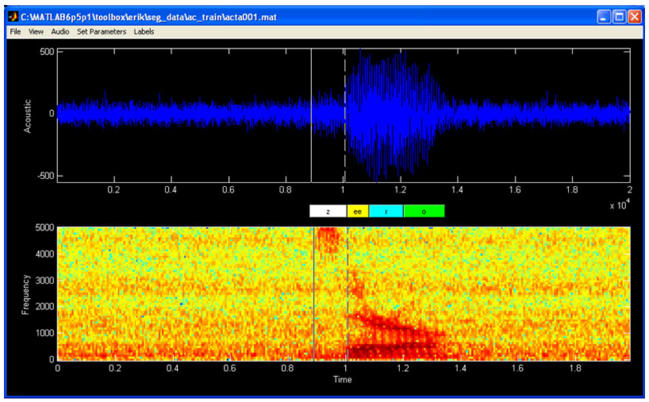
- Hidden States: Gait Phase
- Observations: IMU or Video data

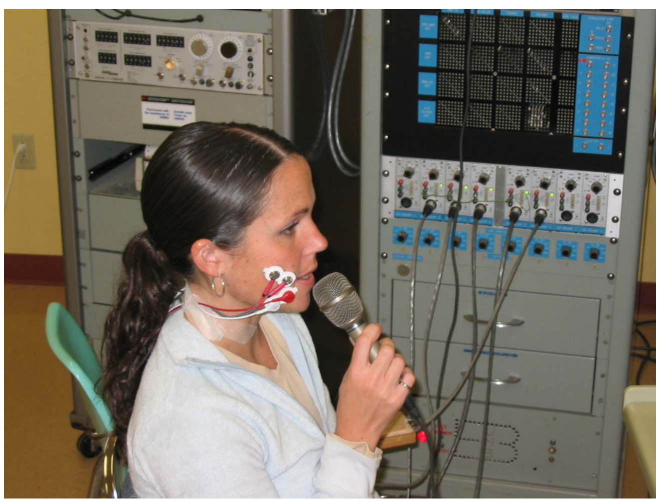


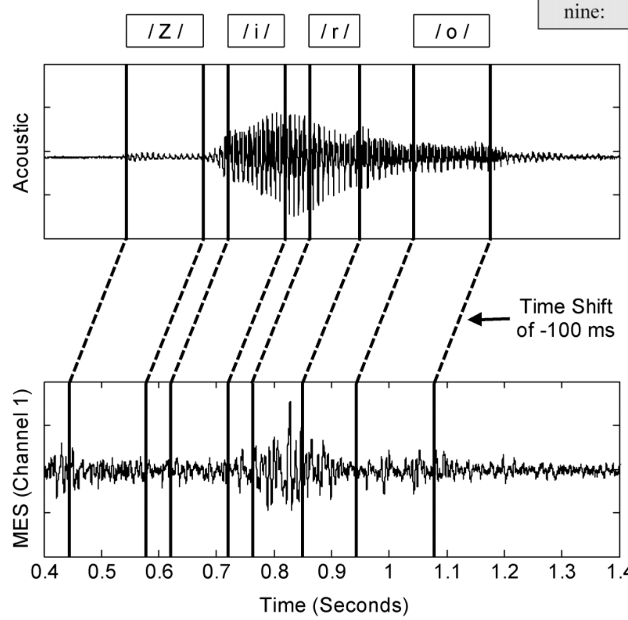
Markov Process & Chains

E. J. Scheme, B. Hudgins and P. A. Parker, "Myoelectric Signal Classification for Phoneme-Based Speech Recognition," in *IEEE Transactions on Biomedical Engineering*, vol. 54, no. 4, pp. 694-699, April 2007, doi: 10.1109/TBME.2006.889175.

Word	Phonemes			
zero:	/ Z /	/ i /	/ r /	/ o /
one:	/ w /	/ A /	/ n /	
two:	/ t /	/ u /		
three:	/ T /	/ r /	/ i /	
four:	/ f /	/ o /	/ r /	
five:	/ f /	/ aI /	/ v /	
six:	/ s /	/ I /	/ x /	
seven:	/ s /	/ E /	/ v /	/ n /
eight:	/ e /	/ t /		
nine:	/ n /	/ aI /	/ n /	







HMMs vs Kalman

In Kalman state space models, we assumed that:

- The unobserved state and the observations were Gaussian
- The model evolves continuously according to linear dynamics

In HMMs, we generally assume that:

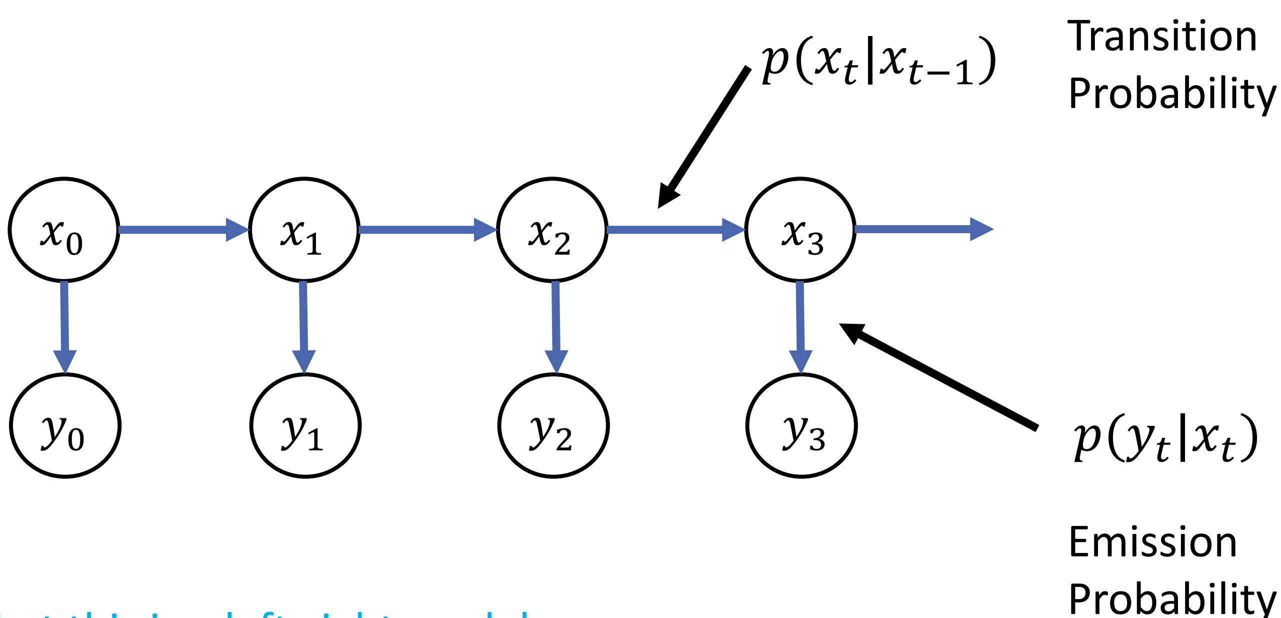
- The hidden state is one of a set number of classes (discrete)
- The state/class changes according to a discrete Markov chain
- The observations may be discrete or continuous

Markov Process & Chains

In a discrete Markov chain, the observations are deterministic for each given state

- The state IS the observable event, so it is not hidden

A discrete hidden Markov model is an extension, where the observation is a probabilistic function of the state

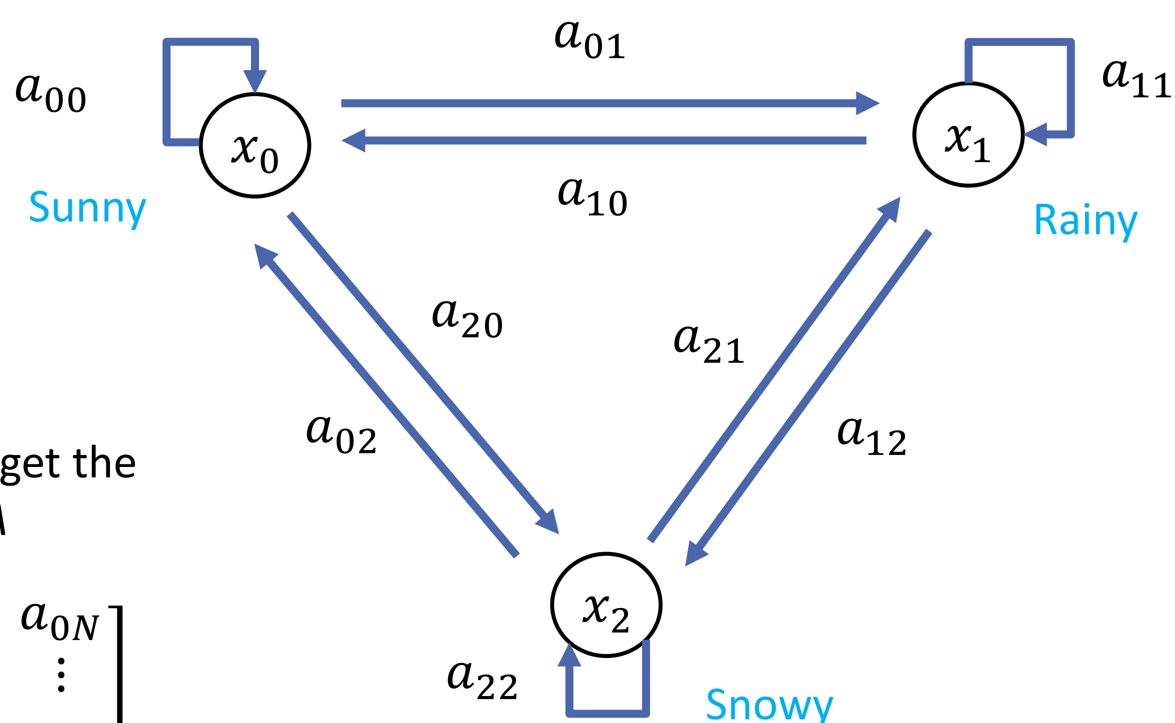


Note that this is a left-right model

Markov Process & Chains

Below, is a 3-state Markov Chain, with transition probabilities:

$$a_{ij} = p(x_j | x_i) \qquad a_{ij} \geq 0 \qquad \sum_{j=0}^{N-1} a_{ij} = 1$$



For an N-state chain, we get the State Transition Matrix, *A*

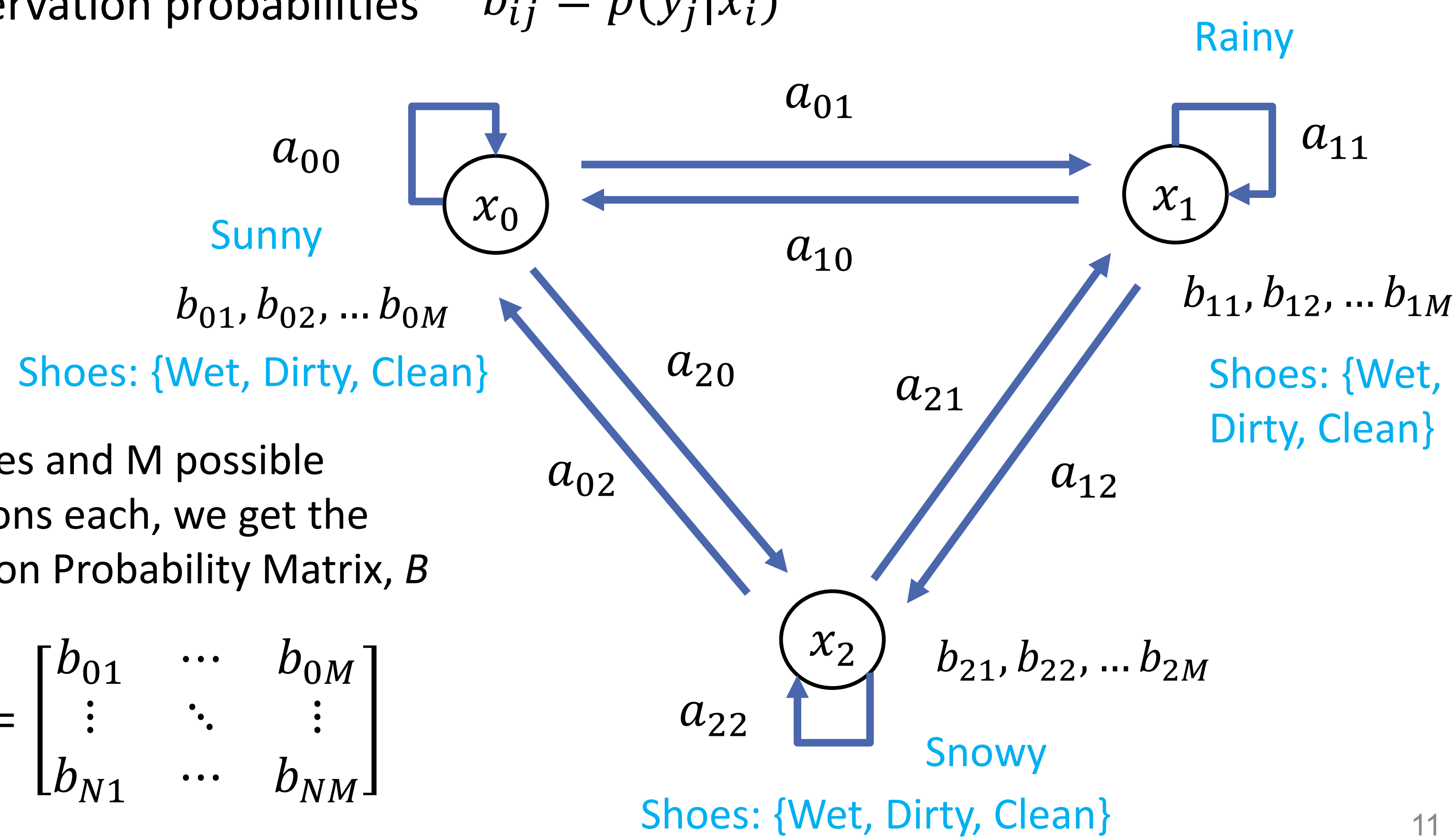
$$A = \begin{bmatrix} a_{00} & \cdots & a_{0N} \\ \vdots & \ddots & \vdots \\ a_{N0} & \cdots & a_{NN} \end{bmatrix}$$

Hidden Markov Model

For a similar 3-state Hidden Markov Model, we have transition probabilities:

$$a_{ij} = p(x_j|x_i) \qquad a_{ij} \geq 0 \qquad \sum_{j=0}^{N-1} a_{ij} = 1$$

And observation probabilities $\quad b_{ij} = p(y_j|x_i)$



For N states and M possible observations each, we get the Observation Probability Matrix, *B*

$$B = \begin{bmatrix} b_{01} & \cdots & b_{0M} \\ \vdots & \ddots & \vdots \\ b_{N1} & \cdots & b_{NM} \end{bmatrix}$$

HMM Example

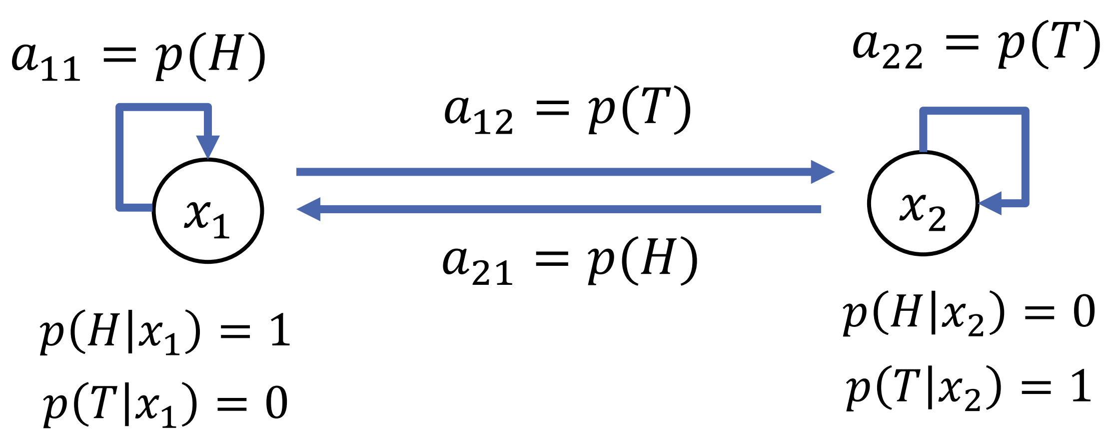
Let's look at a series of coin tosses as a simple example:

- Someone is flipping one or more coins (you can't see them)
- The observation sequence is the corresponding series of heads and tails

Example observations: $\mathrm{O} = \{o_1, o_2, o_3 \dots o_T\} = \{H\ H\ T\ \dots T\}$

We could establish several different HMM models for the problem

- M1: Assume one coin

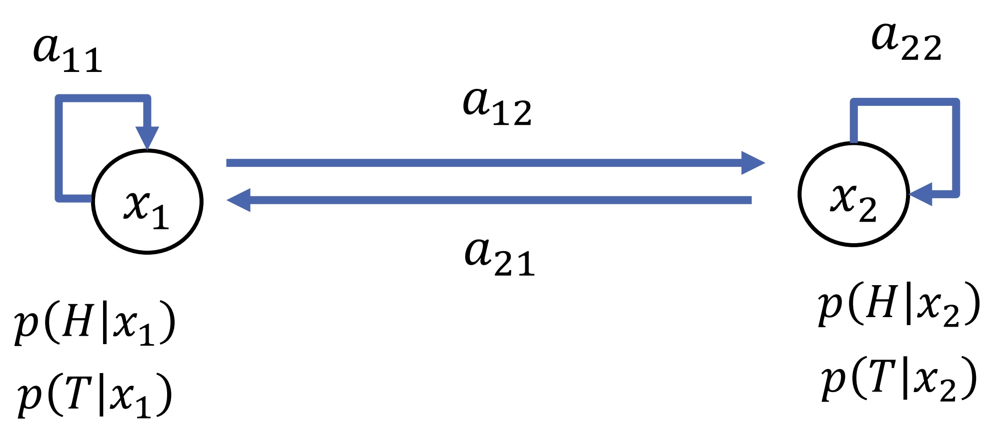


Observation, O={H H T T H T H H T T H}
State Sequence, Q={1 1 2 2 1 2 1 1 2 2 1}

HMM Example

We could establish several different HMM models for the problem

- M2: Assume two coins



Observation, O={H H T T H T H H T T H}
State Sequence, Q={2 1 1 2 2 2 1 2 2 1 2}

- In this case, we don't know if the coins are biased (e.g. $p(H) \neq 0.5$) AND we don't know if they are biased equally
- There is a non-zero probability that the same flip result (H or T) can come from either coin

HMM Example

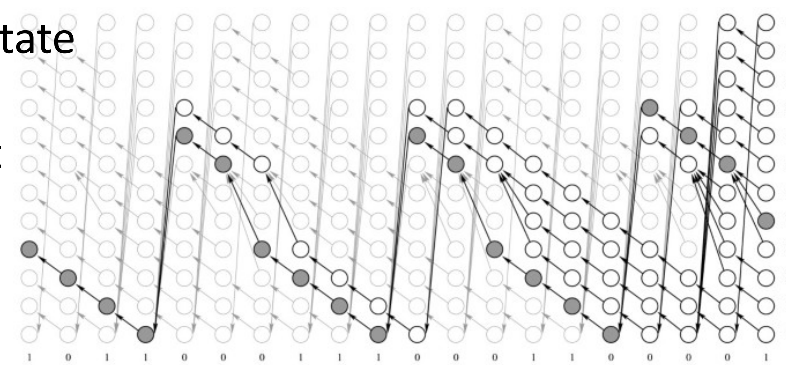
We could establish several different HMM models for the problem

- M3: We could similarly assume that we have three coins, but this would look just like our previous 3-state example
- We might get an example sequence of:

Observation, O={H H T T H T H H T T H}
State Sequence, Q={3 1 2 3 3 1 1 2 3 1 3}

We can explain a sequence of observations with a variety of different models

- In the 2-state example, there were 2^T possible state sequences, whereas
- In the 3-state case, there were 3^T possible state sequences…
- There are an infinite number of HMMs that could explain that particular sequence
- As in all ML, there is a capacity tradeoff, so choice is important



Uses of HMMs

There are two main uses of HMM, in practice

- Modeling: match an HMM to an observation sequence to gain a better understanding of the process

- Pattern Classification: Leverage the temporal information to classify a sequence (e.g. Sign language gesture recognition)

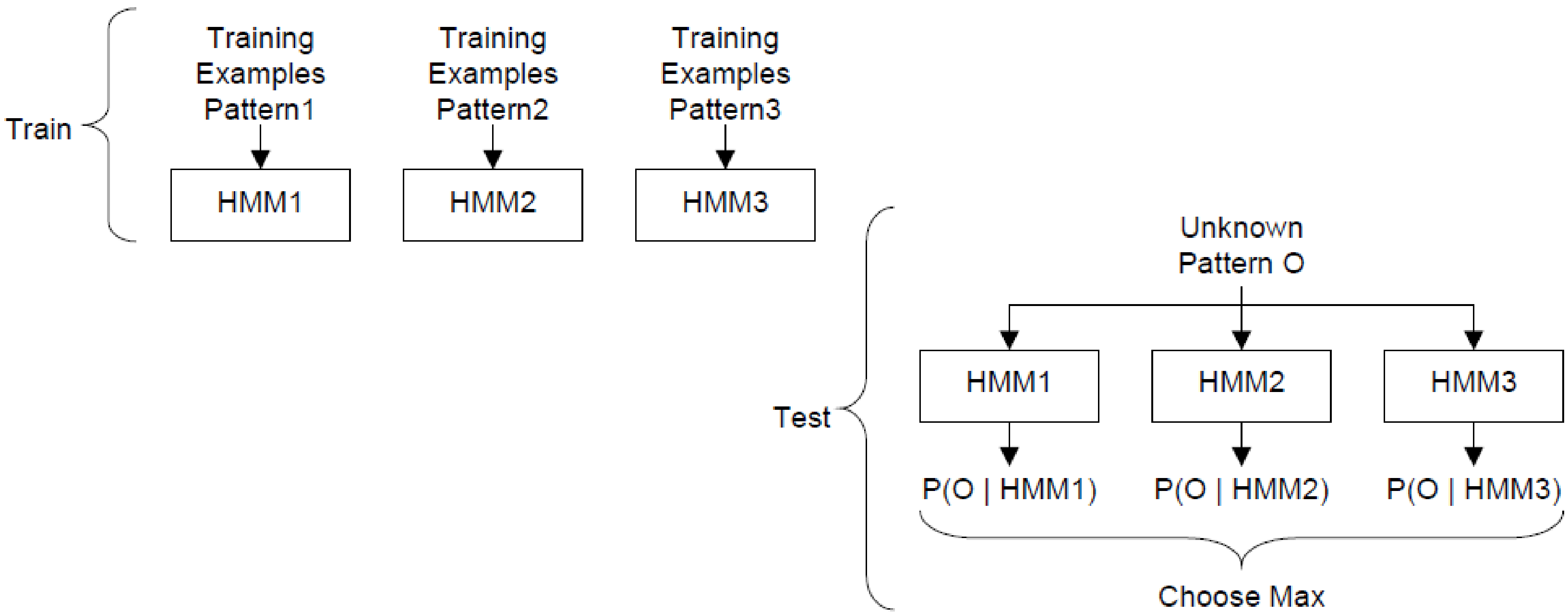


Image by Adrian Chan

Hidden Markov Models

HMMs are defined by a set of 3 main components

For a set of *N* hidden states, *S* ={S_1, S_2 … S_N}

and a set of *M possible* observations, *O* ={O_1, O_2,… O_M}

a. State transition probability matrix, *A* of size $[N, N]$, composed of

$$a_{ij} = P(q_{t+1} = S_j | q_t = S_i)$$ (q_t is the hidden state at time t)

b. State emission probabilities, B of size $[N, M]$, composed of

$$b_j(k) = P(o_t = O_k | q_t = S_j)$$ (o_t is the observation at time t)

c. Initial distribution vector, π of size $[N]$, where

$$\pi_i = P(q_1 = S_i)$$

Hidden Markov Models

From a list of 3000 words from the Oxford Advanced Learner's Dictionary, find the

- Probability of having any word start with a given letter
- Probability of a transition from any one letter to the next

	a	b	c	d	e	f	g	h
<start>	7.4	4.8	10.1	5.4	5.5	4.7	2.6	3.1
a	0.0	3.1	6.1	4.9	0.1	1.1	3.3	0.2
b	13.7	0.0	0.0	0.0	17.4	0.0	0.0	0.0
c	11.4	0.0	2.3	0.0	15.8	0.0	0.0	10.7
d	6.7	0.0	0.0	2.4	39.2	0.5	1.7	0.2
e	8.9	0.4	5.5	3.7	3.6	2.1	1.9	0.3
f	12.9	0.0	0.0	0.0	15.9	9.1	0.0	0.0
g	11.2	0.0	0.0	0.0	26.8	0.0	1.6	13.1
h	15.5	0.5	0.0	0.5	26.7	0.0	0.0	0.0

1st row: Initial distribution vector

- 10.1% chance that a word starts with the letter *C*.

All other rows: transition probability matrix, A

- 39.2% chance that the letter *D* is followed by an *E*.

Question: What use case might lead to non-deterministic state emission probabilities?

https://medium.com/analytics-vidhya/hidden-markov-model-part-1-of-the-hmm-series-3f7fea28a08

Hidden Markov Models

From a list of 3000 words from the Oxford Advanced Learner's Dictionary, find the

- Probability of having any word start with a given letter
- Probability of a transition from any one letter to the next

Here are some state emission probabilities, B, for a case where you type the right key 70% of the time.

They assume that you hit one of the surrounding keys the rest of the time.

	a	b	c	d	e	f	g	h
a	70.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
b	0.0	70.0	0.0	0.0	0.0	0.0	7.5	0.0
c	0.0	0.0	70.0	7.5	0.0	0.0	0.0	0.0
d	0.0	0.0	5.0	70.0	5.0	5.0	0.0	0.0
e	0.0	0.0	0.0	7.5	70.0	0.0	0.0	0.0
f	0.0	0.0	5.0	5.0	0.0	70.0	5.0	0.0
g	0.0	5.0	0.0	0.0	0.0	5.0	70.0	0.0
h	0.0	5.0	0.0	0.0	0.0	0.0	5.0	70.0

https://medium.com/analytics-vidhya/hidden-markov-model-part-1-of-the-hmm-series-3f7fea28a08

Hidden Markov Models

The math associated with HMMs can then be broken into 3 main problems

- Evaluation: Given an HMM, say λ, and a sequence of observations, say O, find the probability of that set of observations being explained (generated) by that model (Forward, Backward algorithms)

$$P(O|\lambda)$$

- Decoding: Given an HMM, λ, and a sequence of observations, O, find the sequence of hidden states, Q that maximizes (Viterbi algorithm)

$$P(O, Q|\lambda)$$

- Learning: Given an unknown HMM, λ, and a sequence of observations, O, find the parameters of $\lambda(A, B, \pi)$ that maximize:

  (Baum-Welch, Forward-Backward algorithm)

$$P(O|\lambda(A, B, \pi))$$

Evaluation

Evaluation: Given an HMM, say λ, and a sequence of observations, say O, find the probability of that set of observations being explained (generated) by that model:

$$P(O|\lambda)$$

Computing this term allows us to determine how well an HMM matches a given observation sequence.

Decoding

Decoding: Given an HMM, λ, and a sequence of observations, O, find the sequence of hidden states, Q that maximizes:

$$P(O, Q|\lambda)$$

Determine the state sequence with the highest probability for a given observation sequence and HMM.

$$argmax\{P(Q\ |O, \lambda)\}$$

This optimal state sequence, Q_{opt}, is not necessarily the correct state sequence (the true sequence that generated the observation), but it gives some indication about the average statistics of the model and individual states

Learning

Learning: Given an unknown HMM, λ, and a sequence of observations, O, find the parameters of $\lambda(A, B, \pi)$ that maximize:

$$P(O|\lambda(A, B, \pi))$$

The solution of this problem lets us to *train* an HMM based on a set of observed training data.

Evaluation: Direct Computation

There are different ways to execute these three different problems when working with HMMs. Let's first look at the evaluation problem, $P(O|\lambda)$

For a given HMM, we can calculate the probability of a particular observation and state sequence as

$$P(O,Q|\lambda) = \pi_1\, b_{q_1}(o_1)\, a_{q_1 q_2}\, b_{q_2}(o_2)\, a_{q_2 q_3}\, b_{q_3}(o_3) \ldots a_{q_{T-1} q_T}\, b_{q_T}(o_T)$$

π_1 is the probability of starting in state q_1
$b_{q_1}(o_1)$ is the probability of observing o_1, given that we are in state q_1
$a_{q_1 q_2}$ is the probability of a state transition from q_1 to q_2
$b_{q_2}(o_2)$ is then the probability of observing o_2, given that we are in state q_2

Evaluation: Direct Computation

$$P(O, Q|\lambda) = \pi_1\, b_{q_1}(o_1)\, a_{q_1 q_2}\, b_{q_2}(o_2)\, a_{q_2 q_3}\, b_{q_3}(o_3) \ldots a_{q_{T-1} q_T}\, b_{q_T}(o_T)$$

This is the probability for one specific set of observations AND state sequence, though.

- To determine the probability of a set of observations for a given HMM, we must use the equation determine above for *every possible state sequence* and sum them all.

$$P(O|\lambda) = \sum_{allQ} P(O, Q|\lambda)$$

- Each $P(O, Q|\lambda)$ requires the multiplication of 2T terms (where T is the number of observations)
- For an *N* state HMM, there are N^T of the terms because at every *t* there are *N* possible states that can be reached.
- Even for small values of *N* and *T*, computing $P(O|\lambda)$ becomes impractical (e.g. $N = 5$, $T = 100$, $2T \cdot N^T = 1.57 \times 10^{72}$)
- Clearly, direct computation of $P(O|\lambda)$ is unfeasible and a more efficient procedure is required.

Evaluation: Forward Procedure

Instead, we can define the forward variable $\alpha_t(i)$

$$\alpha_t(i) = P(o_1 o_2 \dots o_t, q_t = i|\lambda)$$

as the probability of a partial observation (an observation sequence, *O*, up to time t), AND being in state i at time t.

This is known as the forward procedure because we start with a partial observation of $o_1$and move forward, solving for $\alpha_t(i)$ iteratively until time T so that the entire observation sequence is taken into account.

1. Initialization

$$\alpha_1(i) = \pi_i b_i(o_1) \qquad 1 \leq i \leq N$$

The forward variable $\alpha_1(i)$ is the probability of observing o_1 in state i.

- the probability of starting in state i multiplied by the probability of observing o_1 while in state i.

Evaluation: Forward Procedure

2. Induction

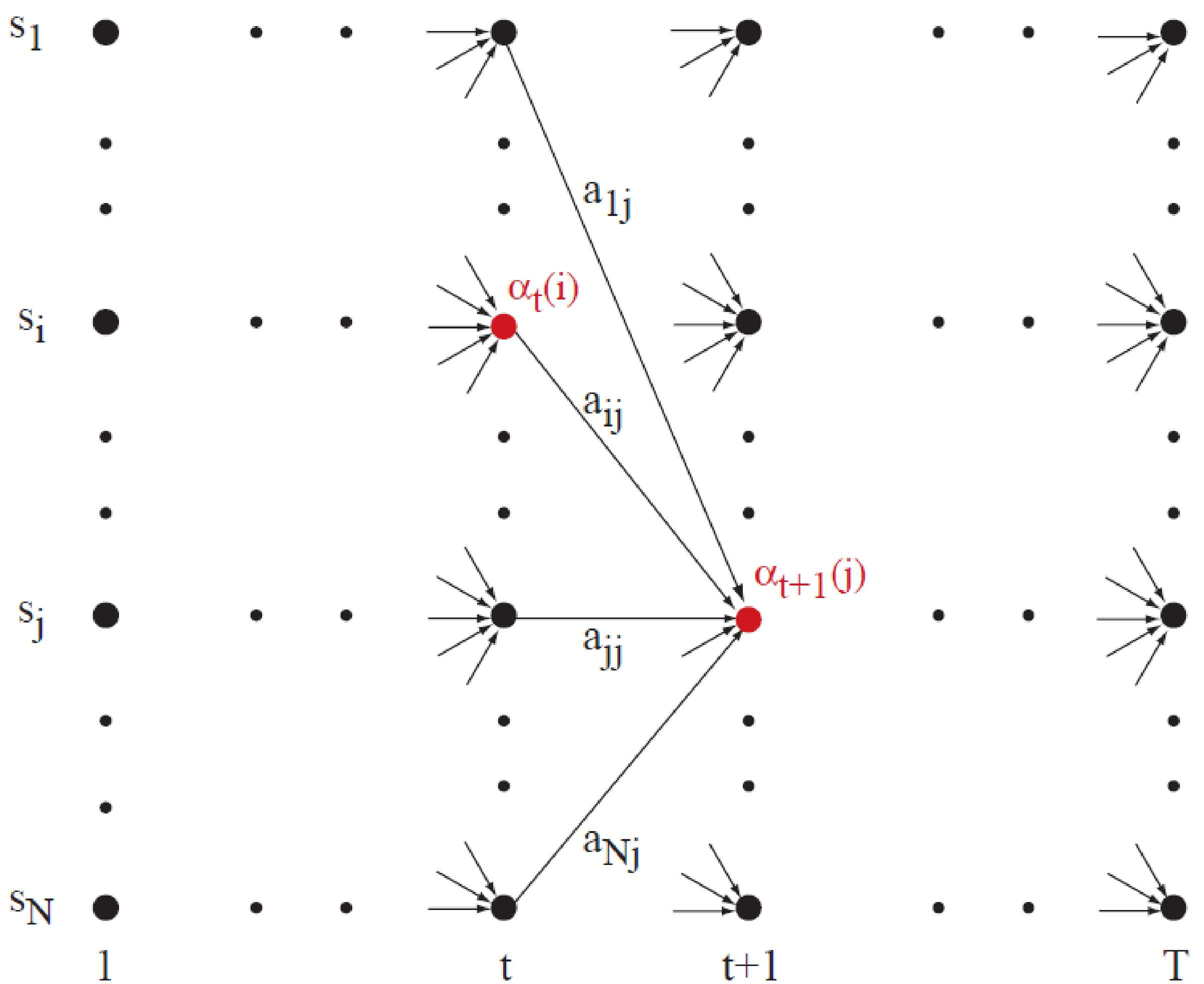
$$\alpha_{t+1}(j) = \left(\sum_{i=1}^{N} \alpha_t(i)\, a_{ij}\right) b_j(o_{t+1}) \qquad 1 \le t \le T-1 \qquad 1 \le j \le N$$

In the induction step, the probability of the partial observation sequence up to time t is assumed to be known.

We can then determine the probability of being in state j at the next time step $(t+1)$ by summing the $\alpha_t(i)a_{ij}$ terms, the probabilities of being in state i and having a state transition into state j

The $b_j(o_{t+1})$ term accounts for the probability of actually observing o_{t+1} given that we are to be in state j

Evaluation: Forward Procedure



Evaluation: Forward Procedure

3. Termination

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

At this point, the forward variable at time *T* is no longer a partial observation because it takes into account the entire observation sequence *O*.

- The term $\alpha_T(i) = P(O, q_T = i|\lambda)$ is the probability of observing *O* and ending in state i.
- $P(O|\lambda)$ is them just the sum of $\alpha_T(i)$ for all possible ending states.

The calculation of the forward algorithm is on the order of N^2T.

(e.g. N = 5, T = 100, $N^2 \cdot T = 100 \cdot 5^2$ instead of $2T \cdot N^T = 1.57 \times 10^{72}$)

Evaluation: Forward Procedure

1. Initialization

- Determine the forward variable the probability of observing o_1 in state i.

$$\alpha_1(i) = \pi_i b_i(o_1)$$

2. Induction

- We determine the probability of going to state j at the next time step

$$\alpha_{t+1}(j) = \left(\sum_{i=1}^{N} \alpha_t(i)\, a_{ij}\right) b_j(o_{t+1})$$

3. Termination

- We sum the probabilities of observing the sequence O and ending in state i

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

Note: In the time-series sense, this operation can be viewed as *filtering*

Evaluation: Backward Procedure

The backward procedure is like the forward procedure except that we instead start at time T and work our way backwards in time, solving iteratively for the backward variable $\beta_t(i)$.

$$\beta_t(i) = P(o_{t+1}o_{t+2} \dots o_T | q_t = i, \lambda)$$

This *backward variable* is the probability of the partial observation (observation sequence from time $t + 1$ to time *T*) given that at time t we are in state i.

- In the forward variable, the probability was a joint probability between the partial observation and the state q_t

$$\alpha_t(i) = P(o_1 o_2 \dots o_t, q_t = i | \lambda)$$

- In the backward variable, it is conditional probability between the partial observation and the state q_t

Evaluation: Backward Procedure

1. Initialization

$$\beta_T(i) = 1 \qquad 1 \leq i \leq N$$

We're starting at the end, so we can't define the partial observation sequence from o_{T+1} (beyond our sequence) to o_T. So, for the procedure to work we set $\beta_T(i)$ equal to 1.

Evaluation: Backward Procedure

2. Induction

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\, b_j(o_{t+1})\beta_{t+1}(j) \qquad T-1 \geq t \geq 1 \qquad 1 \leq i \leq N$$

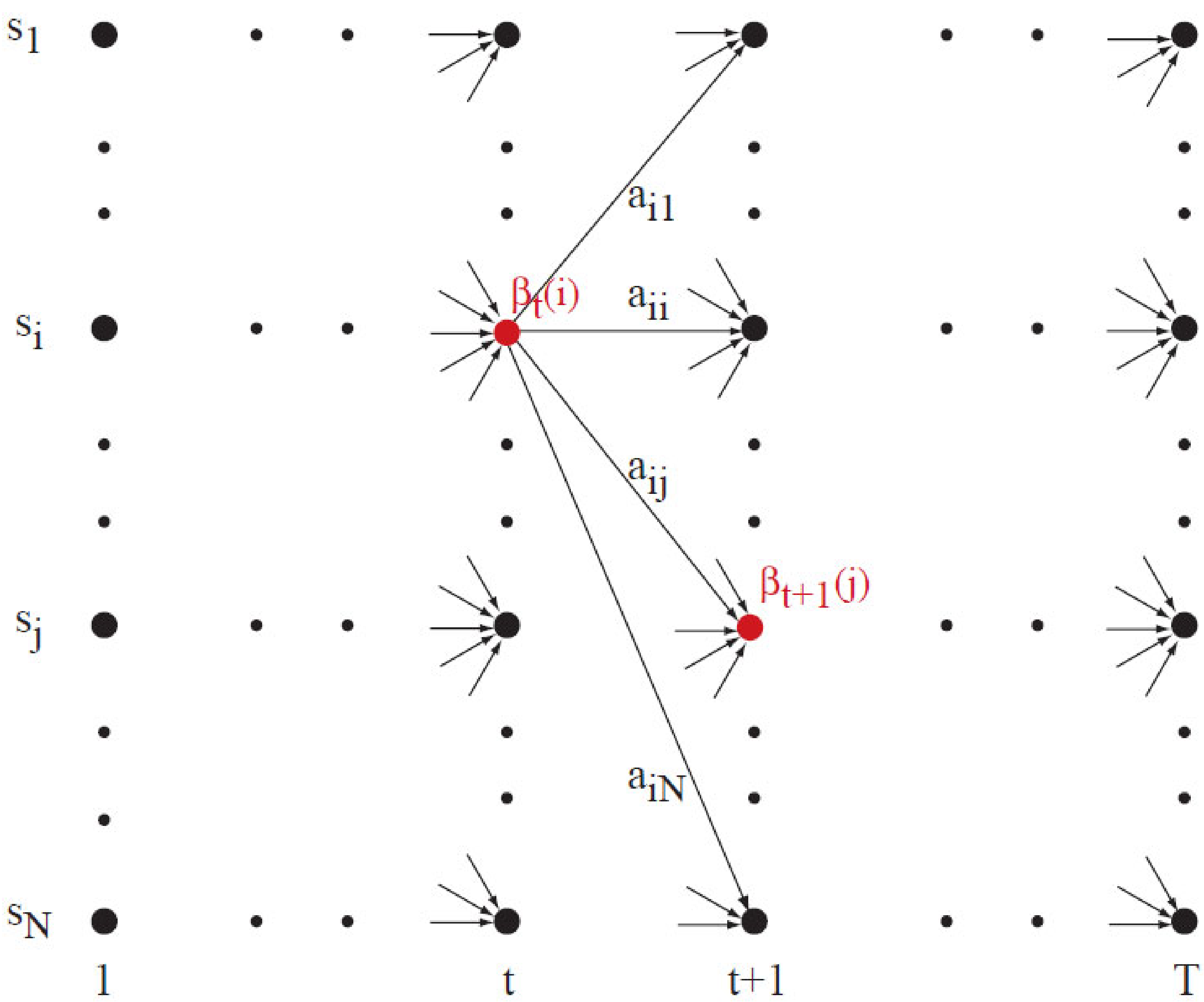
In the backward induction step, we know the backward variable from time $t+1$ to time T

$\beta_{t+1}(j)$ accounts for everything that happens after $t+1$, given that we are in state j at time $t+1$

For a given state i at time t, the induction step must consider transitioning into every state j = 1, 2..., N

Because we know we'll be in $\beta_{t+1}(j)$, we need the single observation probability $b_j(o_{t+1})$, and the transition probabilities a_{ij}

Evaluation: Backward Procedure



Evaluation: Backward Procedure

3. Termination

$$P(O|\lambda) = \sum_{i=1}^{N} \pi_i b_i(o_1) \beta_1(i)$$

Finally, $\beta_1(i)$ is the probability of the partial observation $\{o_2 o_3 \dots o_T\}$ given that the initial state is state i.

The π_i removes the condition on the probability that the initial state is given to be state i.

The $b_i(o_1)$ is the probability of seeing the observation o_1 at the initial state i.

To compute $P(O|\lambda)$ we need to sum to account for all possible initial states.

Note: In the time-series sense, this operation can be viewed as *smoothing*

Evaluation: Backward Procedure

1. Initialization

- We require that $\beta_T(i) = 1 \qquad 1 \le i \le N$

2. Induction

- We determine the probability of going to $\beta_t(i)$ given $\beta_{t+1}(j)$

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\, b_j(o_{t+1})\beta_{t+1}(j)$$

3. Termination

- We sum the probabilities of the partial observations $\{o_2 o_3 \dots o_T\}$ given the possible initial states i

$$P(O|\lambda) = \sum\nolimits_{i=1}^{N} \pi_i b_i(o_1)\beta_1(i)$$

Decoding: Viterbi Algorithm

Remember, in the decoding problem, we need to determine the optimal state sequence for a given set of observations.

- The criteria we used to define 'optimal' was the state sequence that maximized $P(Q|O,\lambda)$.

The formal solution for this is called the Viterbi algorithm.

We define $\delta_t(i)$ to be the highest probability for a partial state sequence $\{q_1, q_2, \dots q_t\}$ ending at state $q_t = i$ for a partial observation sequence $\{o_1, o_2, \dots o_t\}$.

$$\delta_t(i) = \max_{q_1, q_2, \dots q_{t-1}} [P(q_1, q_2, \dots q_{t-1}, q_t = i, o_1, o_2, \dots o_t | \lambda)]$$

The calculation of the Viterbi algorithm is accomplished using dynamic programming.

Decoding: Viterbi Algorithm

We can solve for $\delta_t(i)$ inductively using

$$\delta_{t+1}(j) = \max_i \left[\delta_t(i) a_{ij}\right] \cdot b_j(o_{t+1})$$

The probability $\delta_t(i)$ is the probability of the optimal path for each state up to time t.

To find the optimal path to state j at time $t+1$, we calculate, for all i, the probability of moving from state i (given that it was reached optimally at time t) to state j, and take the path with maximum probability.

The term $b_j(o_{t+1})$ is required by the definition of $\delta_t(i)$ because it takes into account the probability of observing o_{t+1} at time t.

If we track the optimal state transition at each induction step, then we get the optimal state sequence $Q_{opt} = \{q_{opt(1)}, q_{opt(2)}, \ldots q_{opt(N)}\}$.

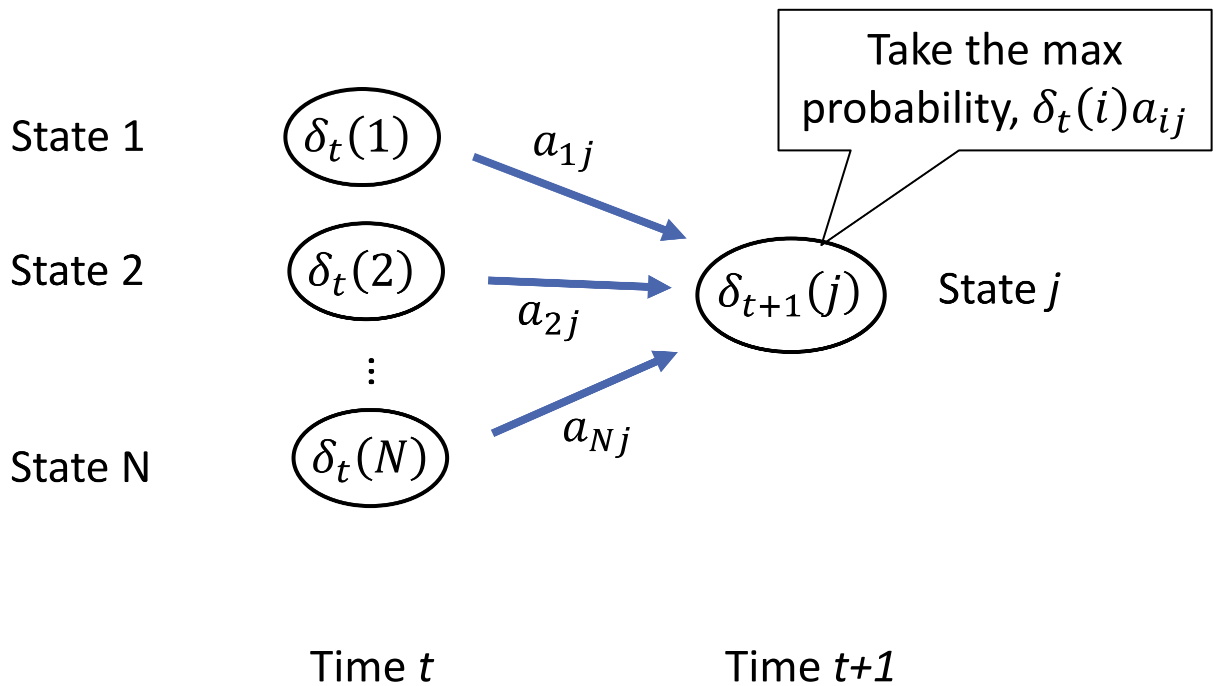
- The array $\psi_1(j)$ is used to keep track of the optimal state transition argument

Decoding: Viterbi Algorithm

We can solve for $\delta_t(i)$ inductively using

$$\delta_{t+1}(j) = \max_i \left[\delta_t(i) a_{ij}\right] \cdot b_j(o_{t+1})$$

The probability $\delta_t(i)$ is the probability of the optimal path for each state up to time t.



Decoding: Viterbi Algorithm

1. Initialization

$$\delta_1(i) = \pi_i b_i(o_1) \qquad 1 \le i \le N$$

$$\psi_1(i) = 0 \qquad 1 \le i \le N$$

For time $t = 1$, $\delta_1(i)$ is just the probability of the initial state i multiplied by the probability of observing o_1 when in state i.

The array $\psi_1(i)$ is set arbitrarily to zero at time $t = 1$, but is never be actually used in the algorithm

- there was no state to transition from prior to this

Decoding: Viterbi Algorithm

2. Recursion

$$\delta_{t+1}(j) = \max_i \left[\delta_t(i) a_{ij}\right] \cdot b_j(o_{t+1})$$

$$\psi_t(i) = arg\left[\max_i \left[\delta_t(i) a_{ij}\right]\right]$$

$$1 \leq j \leq N \qquad 2 \leq t \leq T$$

The probability $\delta_t(i)$ is determined recursively as we just walked through.

The argument that indicates the optimal state transition is stored for each state at each time t.

Decoding: Viterbi Algorithm

3. Termination

$$P_{opt} = \max_i [\delta_T(i)]$$

$$q_{opt}(T) = arg\left[\max_i [\delta_T(i)]\right]$$

P_{opt} is the probability of the optimal state sequence (the sequence with the maximum probability) for the observation sequence

- has an end state of $q_{opt}(T)$

Decoding: Viterbi Algorithm

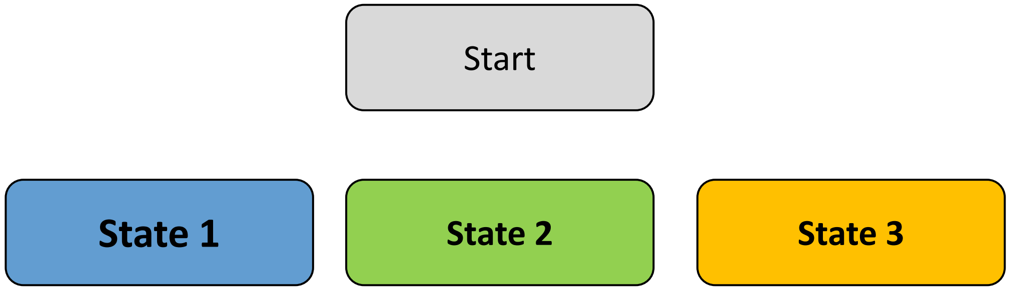
4. Path Backtracking

$$q_{opt}(t) = \psi_{t+1}\left(q_{opt}(t+1)\right) \qquad T-1 \geq t \geq 1$$

Now that we know that the optimal path ends at $q_{opt}(T)$, and we know the optimal state transitions, we can trace backwards to find the optimal path.

We start with $q_{opt}(T)$ and work backwards in time to find the optimal sequence of states, Q_{opt}.

Decoding: Viterbi Algorithm



$$\pi = \begin{bmatrix} 1/6 \\ 3/6 \\ 2/6 \end{bmatrix} \quad A = \begin{bmatrix} 4/6 & 1/6 & 1/6 \\ 2/6 & 2/6 & 2/6 \\ 1/6 & 1/6 & 4/6 \end{bmatrix}$$

$$B = \begin{bmatrix} 3/6 & 3/6 \\ 5/6 & 1/6 \\ 2/6 & 4/6 \end{bmatrix}$$

$$O = \begin{bmatrix} 1 & 2 & 1 & 1 & 2 \end{bmatrix}$$

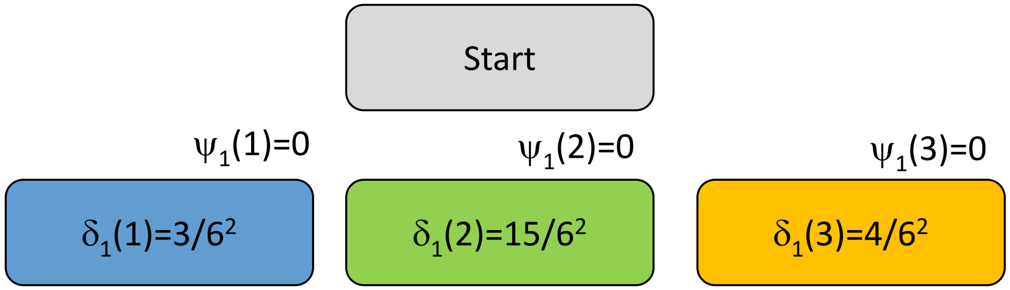
$$Q_{opt} = \ ?$$

1. Initialization

$$\delta_1(i) = \pi_i b_i(o_1)$$

$$\psi_1(i) = 0$$

Decoding: Viterbi Algorithm



$$\pi = \begin{bmatrix} 1/6 \\ 3/6 \\ 2/6 \end{bmatrix} \quad A = \begin{bmatrix} 4/6 & 1/6 & 1/6 \\ 2/6 & 2/6 & 2/6 \\ 1/6 & 1/6 & 4/6 \end{bmatrix}$$

$$B = \begin{bmatrix} 3/6 & 3/6 \\ 5/6 & 1/6 \\ 2/6 & 4/6 \end{bmatrix}$$

$$O = \begin{bmatrix} 1 & 2 & 1 & 1 & 2 \end{bmatrix}$$

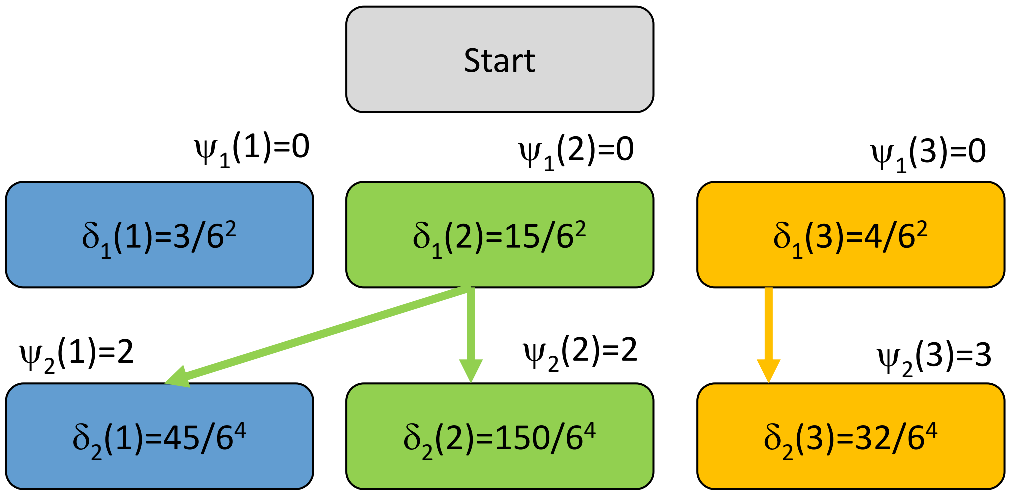
$$Q_{opt} = \ ?$$

1. Initialization

$$\delta_1(i) = \pi_i b_i(o_1)$$

$$\psi_1(i) = 0$$

Decoding: Viterbi Algorithm



$$\pi = \begin{bmatrix} 1/6 \\ 3/6 \\ 2/6 \end{bmatrix} \quad A = \begin{bmatrix} 4/6 & 1/6 & 1/6 \\ 2/6 & 2/6 & 2/6 \\ 1/6 & 1/6 & 4/6 \end{bmatrix}$$

$$B = \begin{bmatrix} 3/6 & 3/6 \\ 5/6 & 1/6 \\ 2/6 & 4/6 \end{bmatrix}$$

$$O = \begin{bmatrix} 1 & 2 & 1 & 1 & 2 \end{bmatrix}$$

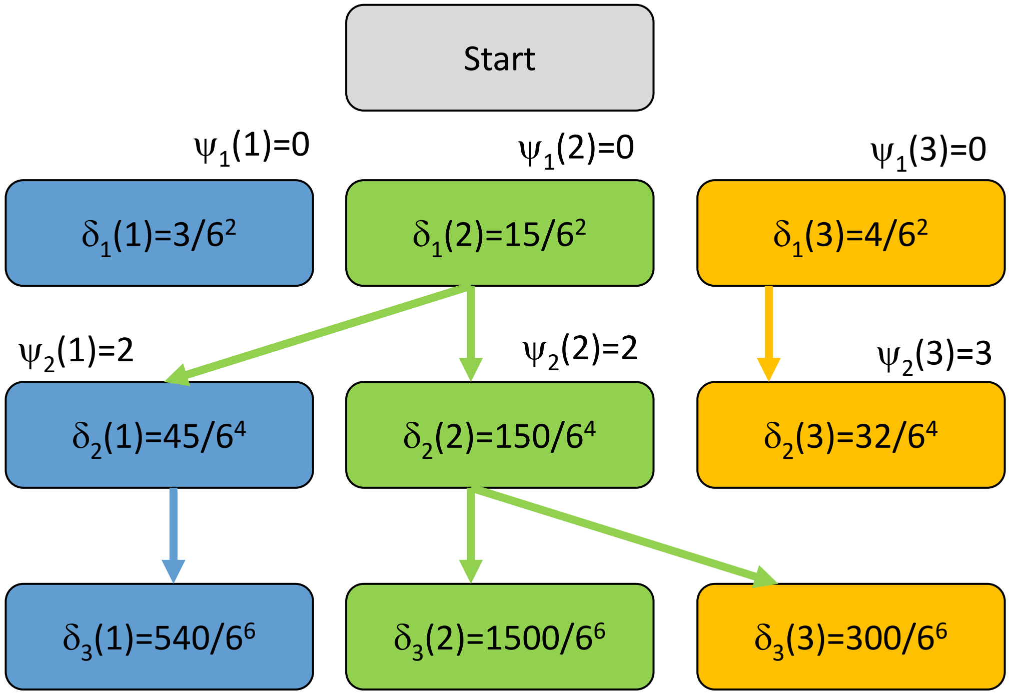
$$Q_{opt} = ?$$

2. Recursion

$$\delta_{t+1}(j) = \max_i \left[\delta_t(i) a_{ij}\right] \cdot b_j(o_{t+1})$$

$$\psi_t(i) = arg\left[\max_i \left[\delta_t(i) a_{ij}\right]\right]$$

Decoding: Viterbi Algorithm



$$\pi = \begin{bmatrix} 1/6 \\ 3/6 \\ 2/6 \end{bmatrix} \quad A = \begin{bmatrix} 4/6 & 1/6 & 1/6 \\ 2/6 & 2/6 & 2/6 \\ 1/6 & 1/6 & 4/6 \end{bmatrix}$$

$$B = \begin{bmatrix} 3/6 & 3/6 \\ 5/6 & 1/6 \\ 2/6 & 4/6 \end{bmatrix}$$

$$O = [1 \quad 2 \quad 1 \quad 1 \quad 2]$$

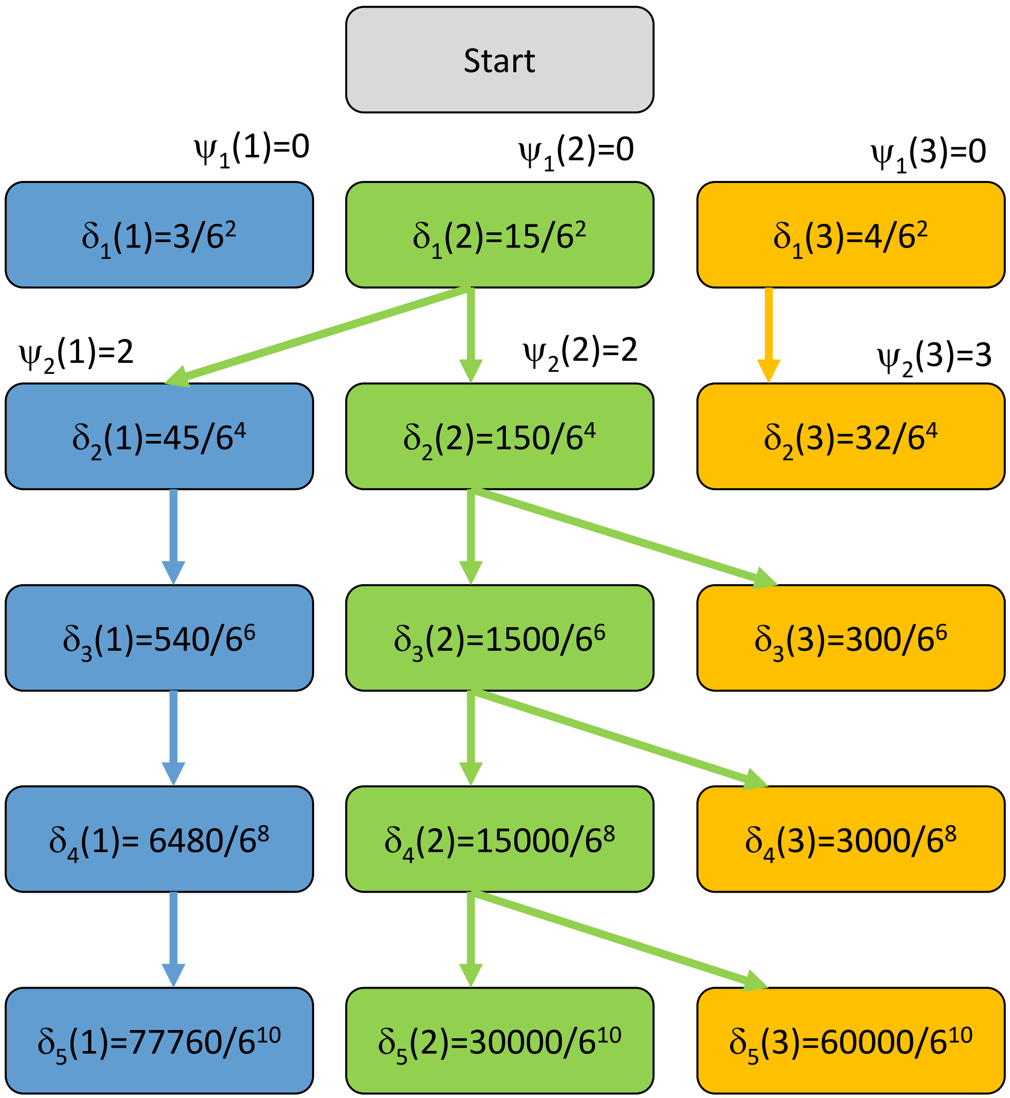
$$Q_{opt} = \ ?$$

2. Recursion

$$\delta_{t+1}(j) = \max_i [\delta_t(i) a_{ij}] \cdot b_j(o_{t+1})$$

$$\psi_t(i) = arg \left[\max_i [\delta_t(i) a_{ij}] \right]$$

Decoding: Viterbi Algorithm



$$\pi = \begin{bmatrix} 1/6 \\ 3/6 \\ 2/6 \end{bmatrix} \quad A = \begin{bmatrix} 4/6 & 1/6 & 1/6 \\ 2/6 & 2/6 & 2/6 \\ 1/6 & 1/6 & 4/6 \end{bmatrix}$$

$$B = \begin{bmatrix} 3/6 & 3/6 \\ 5/6 & 1/6 \\ 2/6 & 4/6 \end{bmatrix}$$

$$O = [1 \quad 2 \quad 1 \quad 1 \quad 2]$$

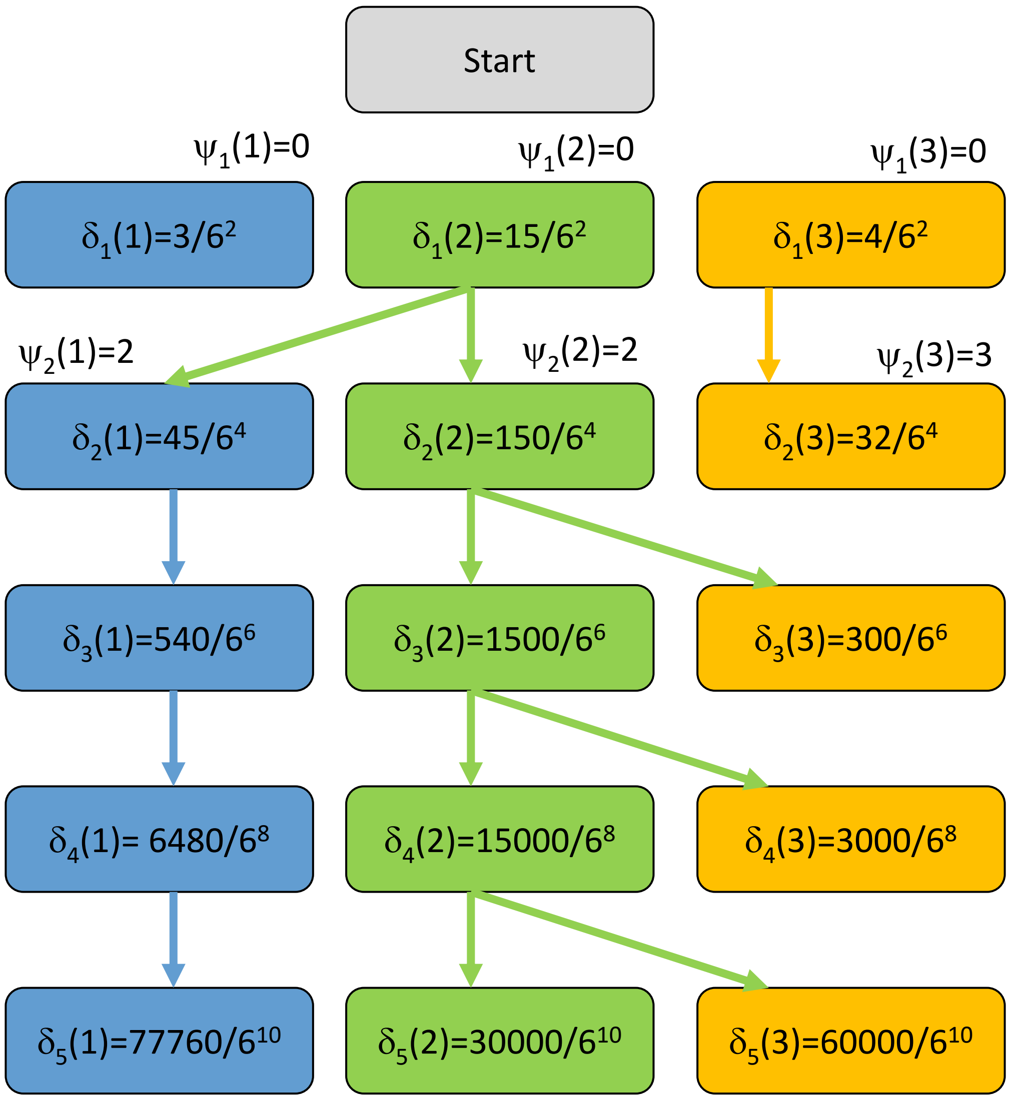
$$Q_{opt} = \ ?$$

2. Recursion

$$\delta_{t+1}(j) = \max_i [\delta_t(i) a_{ij}] \cdot b_j(o_{t+1})$$

$$\psi_t(i) = arg \left[\max_i [\delta_t(i) a_{ij}] \right]$$

Decoding: Viterbi Algorithm



$q_{opt}(T)$=1, $P_{opt} = 77760/6^{10}$

$$\pi = \begin{bmatrix} 1/6 \\ 3/6 \\ 2/6 \end{bmatrix} \quad A = \begin{bmatrix} 4/6 & 1/6 & 1/6 \\ 2/6 & 2/6 & 2/6 \\ 1/6 & 1/6 & 4/6 \end{bmatrix}$$

$$B = \begin{bmatrix} 3/6 & 3/6 \\ 5/6 & 1/6 \\ 2/6 & 4/6 \end{bmatrix}$$

$$O = [1 \quad 2 \quad 1 \quad 1 \quad 2]$$

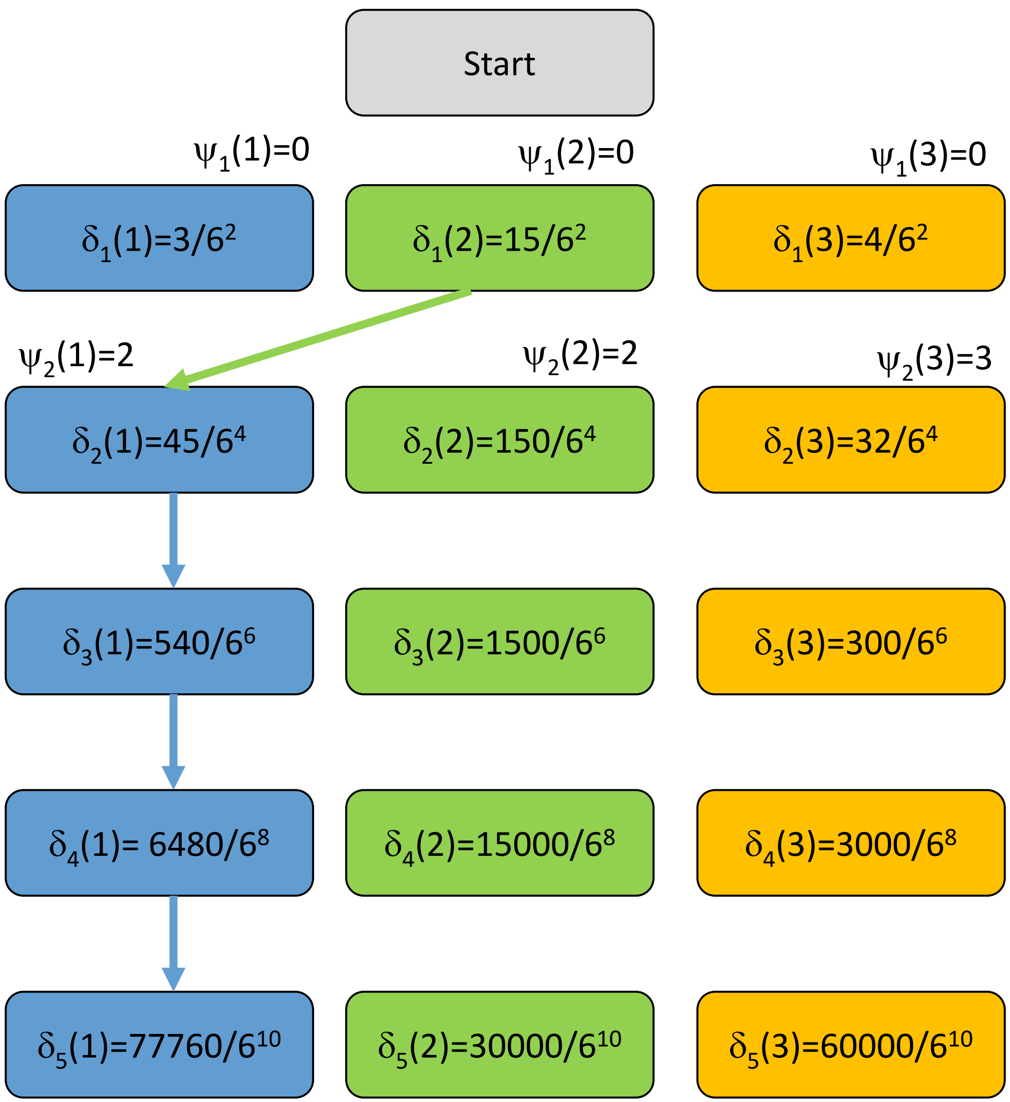
$$Q_{opt} = \ ?$$

3. Termination

$$P_{opt} = \max_i [\delta_T(i)]$$

$$q_{opt}(T) = arg\left[\max_i [\delta_T(i)]\right]$$

Decoding: Viterbi Algorithm



$q_{opt}(T)$=1, $P_{opt} = 77760/6^{10}$

$$\pi = \begin{bmatrix} 1/6 \\ 3/6 \\ 2/6 \end{bmatrix} \quad A = \begin{bmatrix} 4/6 & 1/6 & 1/6 \\ 2/6 & 2/6 & 2/6 \\ 1/6 & 1/6 & 4/6 \end{bmatrix}$$

$$B = \begin{bmatrix} 3/6 & 3/6 \\ 5/6 & 1/6 \\ 2/6 & 4/6 \end{bmatrix}$$

$$O = [1 \quad 2 \quad 1 \quad 1 \quad 2]$$

$$Q_{opt} = \ ?$$

4. Path Backtracking

$$Q_{opt} = [2 \quad 1 \quad 1 \quad 1 \quad 1]$$

Learning (HMM Training)

If you remember in the 3rd problem, learning, we needed to find the parameters of an HMM, $\lambda(A, B, \pi)$, that maximize the probability of having observed a given sequence, O:

$$P(O|\lambda(A, B, \pi))$$

The goal here, is therefore to learn, or train, the HMM hyperparameters based on a set of training data.

- That is, we need to learn the A (all the a_{ij}'s),B (all the $b_j(k)$), and π
- Here, we'll discuss the popular Baum-Welch method (also known as the forward-backward method)

Learning: Baum-Welch Algorithm

The Baum-Welch method is a dynamic programming approach and a special case of *expectation maximization* (EM)

- The goal is to iteratively tune the parameters such that we maximize the likelihood that the model "generated" the observations.

1. Initialization

- Initialize the A, B, π using prior knowledge if possible (otherwise random)

2. Forward Phase

- Compute the forward algorithm $$\alpha_{t+1}(j) = \left(\sum_{i=1}^{N} \alpha_t(i)\, a_{ij}\right) b_j(o_{t+1})$$

3. Backward Phase

- Compute the backward algorithm $$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\, b_j(o_{t+1}) \beta_{t+1}(j)$$

4. Re-Estimation

- Compute the new parameters

Learning: Baum-Welch Algorithm

First, let's define $\gamma_t(i)$ as the probability of being in state i at time t, given an observation sequence.

$$\gamma_t(i) = P(q_t = i|O,\lambda) = \frac{P(q_t = i|\lambda)}{P(O|\lambda)}$$

using Bayes' theorem

This numerator is equal to the product of the forward and backward variables for state i at time t, which we previously described.

$$\gamma_t(i) = P(q_t = i|O,\lambda) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)}$$

- The forward variable $\alpha_t(i)$ takes into account the partial observation sequence $\{o_1\ o_2\ \dots\ o_t\}$ ending at state i at time t
- The backward variable $\beta_t(i)$ takes into account the remainder of the observation $\{o_{t+1}\ o_{t+2}\ \dots\ o_T\}$ given state i at time t.

Learning: Baum-Welch Algorithm

Now, let's define $\xi_t(i,j)$ as the probability of having a state transition from i to j at time t, given an observation sequence.

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j | O, \lambda) = \frac{P(O, q_t = i, q_{t+1} = j|\lambda)}{P(O|\lambda)}$$

Again, using Bayes' theorem

The numerator is similar to what we found for $\gamma_t(i)$ and can again be simplified:

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{P(O|\lambda)}$$

- The forward variable $\alpha_t(i)$ takes into account the partial observation sequence $\{o_1\ o_2\ \ldots\ o_t\}$ ending at state i at time t
- The backward variable $\beta_{t+1}(j)$ takes into account the remainder of the observation $\{o_{t+1}\ o_{t+2}\ \ldots\ o_T\}$ given state j at time $t+1$

Learning: Baum-Welch Algorithm

Now we can explore the re-estimation of the HMM model.

- A re-estimated model, $\lambda^*(A, B, \pi)$ can be derived to maximize $P(O|\lambda)$ such that $P(O|\lambda^*) > P(O|\lambda)$
- This process is repeated iteratively, each time updating the model based on the previous estimates
- Unfortunately, the likelihood function for an HMM is quite complex and has many local maxima to which the re-estimation may converge.
  We'll explain concepts without getting too deep into these, but be aware

The re-estimated initial state probabilities, π_i^*, are just:

$$\pi_i^* = \text{Expected number of times in state } i \text{ at time } (t = 1)$$

$$\pi_i^* = \gamma_1(i)$$

Learning: Baum-Welch Algorithm

The re-estimated transition probabilities, a^*_{ij}, are given by:

$$a^*_{ij} = \frac{\text{expected number of transitions from state } i \text{ to } j}{\text{expected number of transitions from state } i} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

- Recall that $\xi_t(i,j)$ is the probability of going from state i to state j at time t.
- So, the numerator (the sum of $\xi_t(i,j)$ over all time) is the expected number of transitions from state i to state j.

- Recall that $\gamma_t(i)$ is the probability of being in state i at time t.
- Then, the denominator (the sum of $\gamma_t(i)$ over all time) is the expected number of transitions from state i.

Note: The summations don't include T because there are no transitions from time T

Learning: Baum-Welch Algorithm

The re-estimated transition probabilities, a_{ij}^*, are given by:

$$a_{ij}^* = \frac{\text{expected number of transitions from state } i \text{ to } j}{\text{expected number of transitions from state } i} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

Substituting in the formulas for $\xi_t(i,j)$ and $\xi_t(i)$, we get:

$$a_{ij}^* = \frac{\sum_{t=1}^{T-1} \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)}}{\sum_{t=1}^{T-1} \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)}} = \frac{\sum_{t=1}^{T-1} \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{t=1}^{T-1} \alpha_t(i)\beta_t(i)}$$

So, the result is simply composed of:

- The forward result
- The backward result
- The previous transition probabilities
- The previous observation probabilities

Learning: Baum-Welch Algorithm

The re-estimated observation probabilities, $b_j^*(k)$, are given by:

$$b_j^*(k) = \frac{\text{expected number of times observing } o_k \text{ when in state } j}{\text{expected number of times in state } j} = \frac{\sum_{t=1,o_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

- The denominator (the sum of $\gamma_t(j)$ over all time) is the same as before (the expected number of transitions from state j), except that it now includes the full 1:T
- The numerator is similar to the denominator, except that we only include terms for which we are in state j AND observing symbol o_k

- If we define $\delta(o_t, o_k) = \begin{cases} 1 & \text{if } o_t = o_k \\ 0 & \text{otherwise} \end{cases}$, and substitute the formula derived for $\gamma_t(i)$,

$$b_j^*(k) = \frac{\sum_{t=1}^{T} \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)} \delta(o_t, o_k)}{\sum_{t=1}^{T} \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)}} = \frac{\sum_{t=1}^{T} \alpha_t(i)\beta_t(i)\delta(o_t, o_k)}{\sum_{t=1}^{T} \alpha_t(i)\beta_t(i)}$$

Learning: Baum-Welch Algorithm

Using these formulas, we can iterative over the re-estimation until the results converge, such that $P\left(O\middle|\lambda^*\right) \approx P(O|\lambda)$

$$\pi_i^* = \gamma_1(i)$$

$$a_{ij}^* = \frac{\sum_{t=1}^{T-1} \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{t=1}^{T-1} \alpha_t(i) \beta_t(i)} \qquad b_j^*(k) = \frac{\sum_{t=1}^{T} \alpha_t(i) \beta_t(i) \delta(o_t, o_k)}{\sum_{t=1}^{T} \alpha_t(i) \beta_t(i)}$$

Note that these derivations have assumed that the observations were discrete

- We could count the number of times each observation occurred for each state

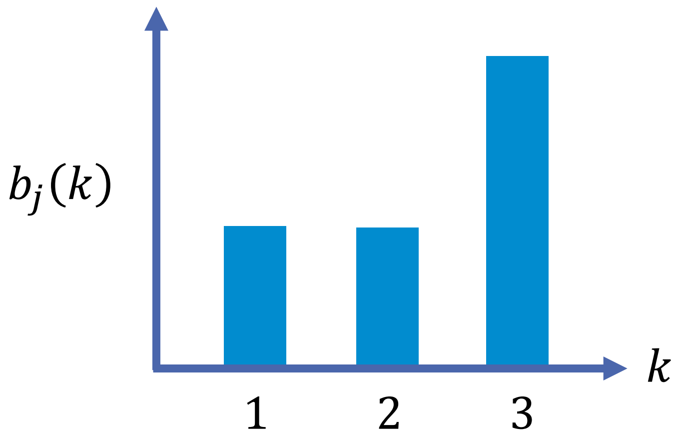
Continuous Observation Densities

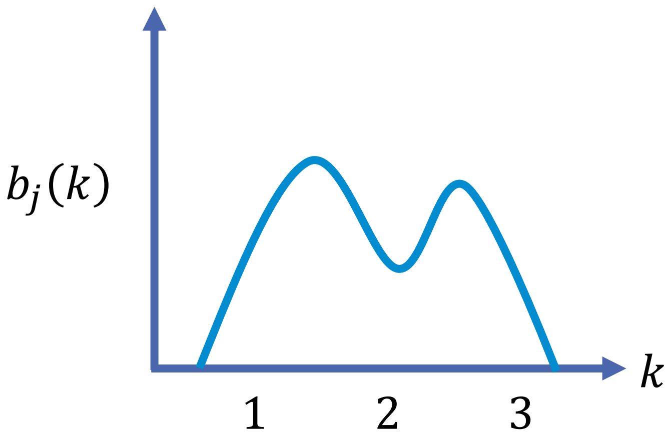
Note that these derivations have assumed that the observations were discrete

- We could count the number of times each observation occurred for each state

In practice, this is often not true, requiring the consideration of continuous observation densities

- π and A stay the same, but now we estimate the probability density function (pdf) for each state, instead of the (discrete) probability mass function, $b_j(k)$.





Continuous Observation Densities

Whereas re-estimating the pmf, $b_j^*(k)$ was achieved using

$$b_j^*(k) = \frac{\sum_{t=1}^{T} \alpha_t(i)\beta_t(i)\delta(o_t, o_k)}{\sum_{t=1}^{T} \alpha_t(i)\beta_t(i)}$$

Estimation of the pdf is most often achieved using Gaussian Mixture Models

- We describe the pdf as a weighted sum of M Gaussian distributions, η

$$b_j(\bar{o}) = \sum_{k=1}^{M} c_{jk}\eta\left(\bar{o}, \bar{\mu}_{jk}, \bar{U}_{jk}\right) \;\; 1 \leq j \leq N$$

- Here, $\bar{\mu}_{jk}$ is the mean vector, and $\bar{U}_{jk}$ is the covariance matrix for the k^{th} mixture component in state j.
- The coefficients c_{jk} are the mixture coefficients, which weight the contribution of the Gaussians, subject to:

$$\sum_{k=1}^{M} c_{jk} = 1 \qquad c_{jk} \geq 0 \qquad 1 \leq j \leq N, 1 \leq k \leq M$$

Continuous Observation Densities

The training of a continuous observation HMM is similar to before, but requires the additional re-estimation of these c_{jk}, $\bar{\mu}_{jk}$, and $\bar{U}_{jk}$ parameters.

As before, we need to defined a helpful variable.

- $\gamma_t(j,k)$ is the probability of being in state j at time t with the k^{th} mixture component accounting for the observation vector $\bar{o}_t$

$$\gamma_t(j,k) = \left[\frac{\alpha_t(j)\beta_t(j)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)}\right]\left[\frac{c_{jk}\eta\left(\bar{o}_t, \bar{\mu}_{jk}, \bar{U}_{jk}\right)}{\sum_{m=1}^{M} c_{jm}\eta\left(\bar{o}_t, \bar{\mu}_{jm}, \bar{U}_{jm}\right)}\right]$$

- The first term is exactly the same as the discrete HMM equivalent, $\gamma_t(j)$.
- The second term is the ratio between the probability that the k^{th} mixture of state j can account for the observation vector $\bar{o}_t$ and the probability that the observation vector $\bar{o}_t$ can be accounted for in state j, in general.

Continuous Observation Densities

The mixture gains, c_{jk}, can then be re-estimated as

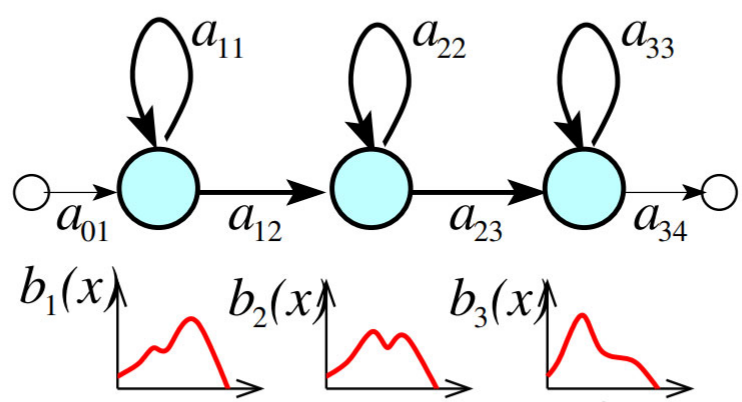
$$c_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)}{\sum_{t=1}^{T} \sum_{k=1}^{M} \gamma_t(j,k)}$$

- which is the ratio between the expected number of times the system is in state j, using the k^{th} mixture component, and the total number of times the system is in state j.

The means and covariances of the Gaussian mixtures can be re-estimated using

$$\bar{\mu}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k) \cdot \bar{o}_t}{\sum_{t=1}^{T} \gamma_t(j,k)} \qquad \bar{U}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k) \cdot (\bar{o}_t - \bar{\mu}_{ij})(\bar{o}_t - \bar{\mu}_{ij})'}{\sum_{t=1}^{T} \gamma_t(j,k)}$$

- where the numerators are the sum of the probabilities of being in state j, using the k^{th} mixture, weighted by the observation vector $\bar{o}_t$ (for the mean) or the covariance.



https://www.inf.ed.ac.uk/teaching/courses/asr/2016-17/asr03-hmmgmm-handout.pdf

Hidden Markov Models

Using priori knowledge about the problem, we can constrain HMMs for better performance or efficiency.   We can:

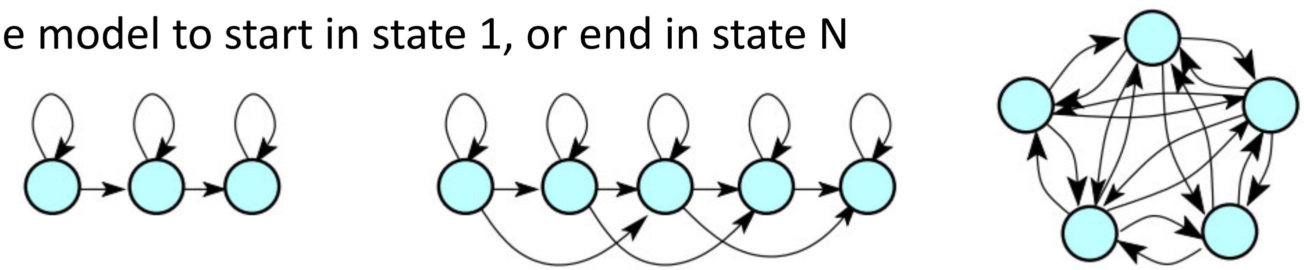
- constrain models to be left-right model (state index is non-decreasing)

$$a_{ij} = 0 \quad j < i$$

- limit how far 'ahead' a model can jump

$$a_{ij} = 0 \quad j > i + \Delta i$$

- Force the model to start in state 1, or end in state N



left–to–right model   parallel path left–to–right model   ergodic model

$$\begin{pmatrix} a_{11} & a_{12} & 0 \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{pmatrix} \quad \begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 \\ 0 & a_{22} & a_{23} & a_{24} & 0 \\ 0 & 0 & a_{33} & a_{34} & a_{35} \\ 0 & 0 & 0 & a_{44} & a_{45} \\ 0 & 0 & 0 & 0 & a_{55} \end{pmatrix} \quad \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{pmatrix}$$

https://www.inf.ed.ac.uk/teaching/courses/asr/2016-17/asr03-hmmgmm-handout.pdf

Hierarchical Hidden Markov Models

Complex HMMs can be designed to be more efficient by carefully constraining the transitions matrix, A

- However, they can sometimes be hard to interpret directly

Several extensions have therefore been proposed, particularly to embed or emphasis known structure.

A particularly interesting (and more common) variant is the hierarchical HMM

- As the name implies, hHMMs combines multiple HMMs in a hierarchical (or tree-like) arrangement
- Each state of the original (top) model is comprised of its own HMM
- In this way, the observations associated with a given state in the upper level(s) are informed by the state sequence through the associated 'leaf' HMM.

As an example, consider the English language

- Sentences are formed from sequences of words
- Words are formed from sequences of syllables
- Syllables are formed from sequences of phonemes

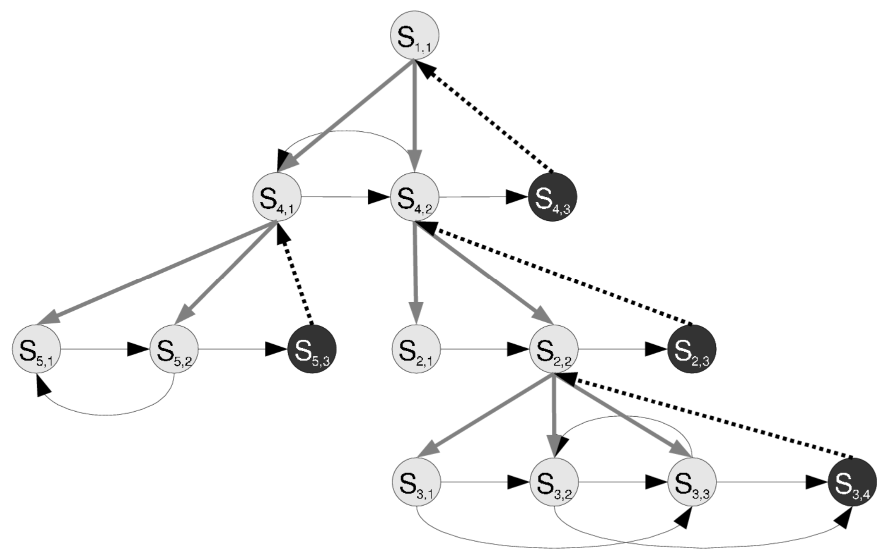
Hierarchical Hidden Markov Models

A hierarchical HMM could then be designed where the top level models the flow of words through a sentence

- For each word, and new hierarchical HMM could model the flow of syllables within that word
- And a final HMM could model the flow of phonemes through those syllables

Within each level of a hierarchical HMM, there is a sequence of states

- An output is only returned to the next level up once the sequence has been completed
- That is, the first and middle states of each HMM are 'non-producing', and only inform the observation that is generated by the last state.

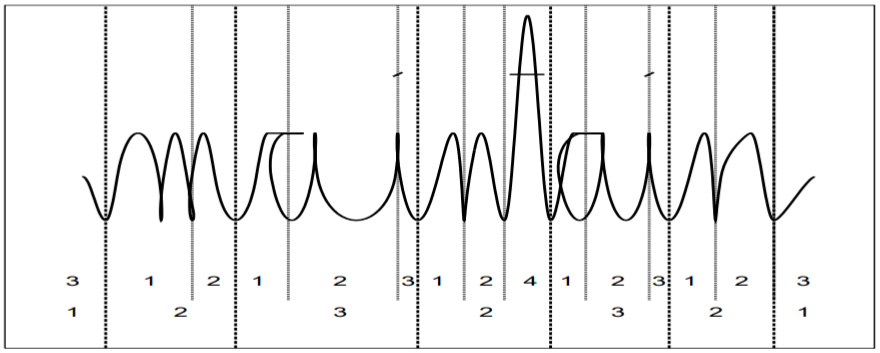


https://en.wikipedia.org/wiki/Hierarchical_hidden_Markov_model#/media/File:Hierarchical_hidden_Markov_model_(diagram).png

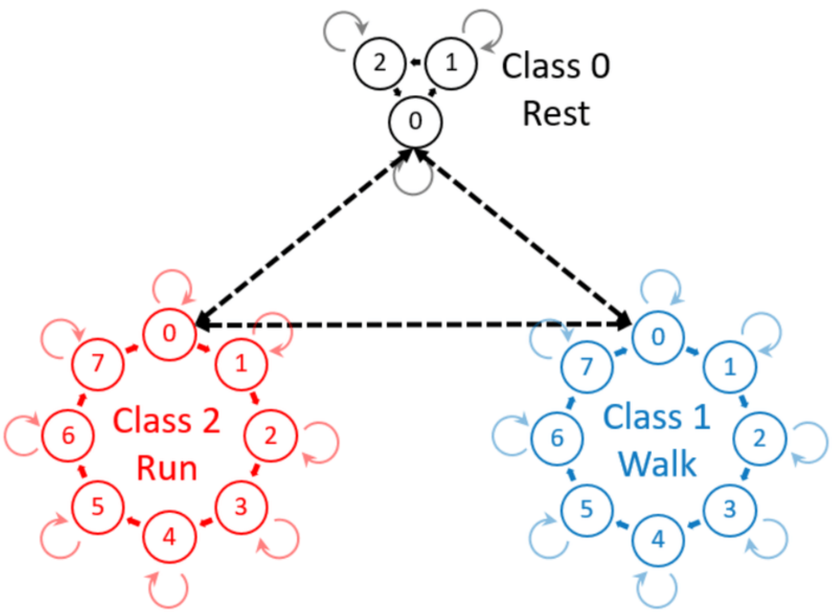
Hierarchical Hidden Markov Models

Recent applications of hHMMs include

- Speech
- Gesture and human activity recognition
- Gait
- Genetic sequencing



Fine, Shai, Singer, Yoram, Tishby, Naftali, The Hierarchical Hidden Markov Model: Analysis and Applications, Machine Learning, 32, 41–62 (1998)



Martindale, Christine & Hoenig, Florian & Strohrmann, Christina & Eskofier, Bjoern. (2017). Smart Annotation of Cyclic Data Using Hierarchical Hidden Markov Models. Sensors. 17. 2328. 10.3390/s17102328.

Summary

In this section, we continued with another common state-space model; the Hidden Markov Model

- A sequence of latent states, which each generate some observable output
  The observations can be deterministic, discrete, or continuous (based on a pdf)
- State progression is governed by the probabilities in a state transition matrix
  Observations depend only on the current state
- HMMs can be used for time series modeling, or for classification of temporal sequences
  We focused on the classification task
- Three main “problems”
  Evaluation (forward, backward), Decoding (Viterbi), Learning (Baum-Welch)
- The models can be constrained for more efficient learning or to better represent known structure/semantics
  Left-right vs ergodic models, hierarchical HMMs

In the next section, we’ll jump again and look at deep learning models

- Regression & Gradient Descent
- RNN, LSTM, maybe CNN (as time allows)

Q&A