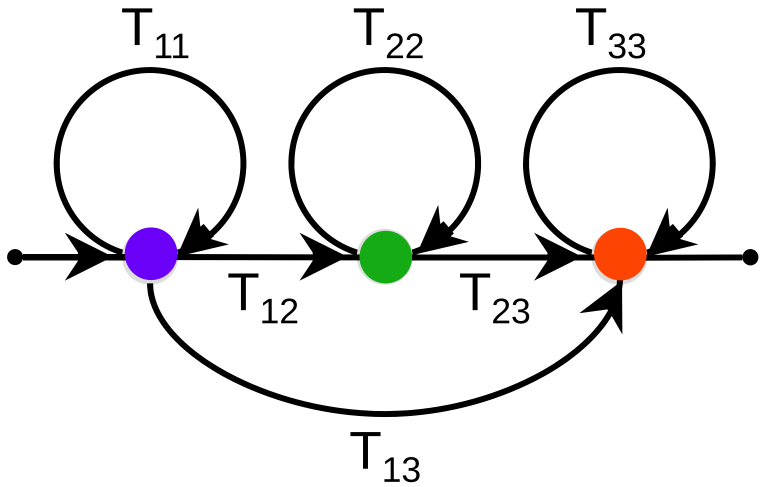
Hidden Markov Models for Speech Recognition

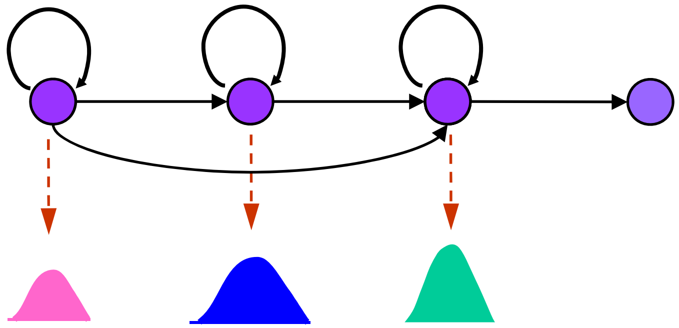
Bhiksha Raj

Recap: HMMs

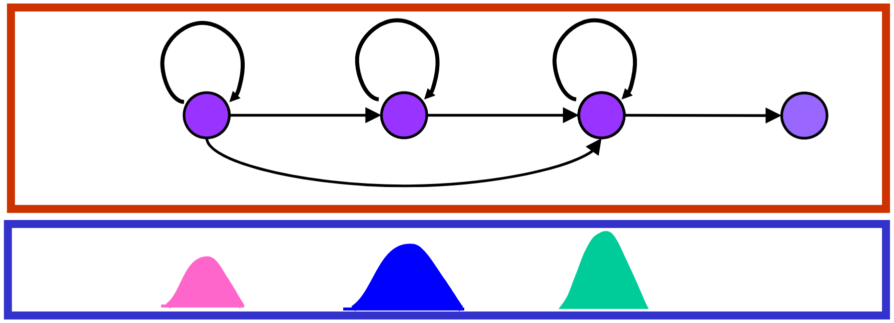


- This structure is a generic representation of a statistical model for processes that generate time series
- The "segments" in the time series are referred to as states
  - The process passes through these states to generate time series
- The entire structure may be viewed as *one* generalization of the DTW models we have discussed thus far

Hidden Markov Models



- A Hidden Markov Model consists of two components
    - A state/transition backbone that specifies how many states there are, and how they can follow one another
    - A set of probability distributions, one for each state, which specifies the distribution of all vectors in that state

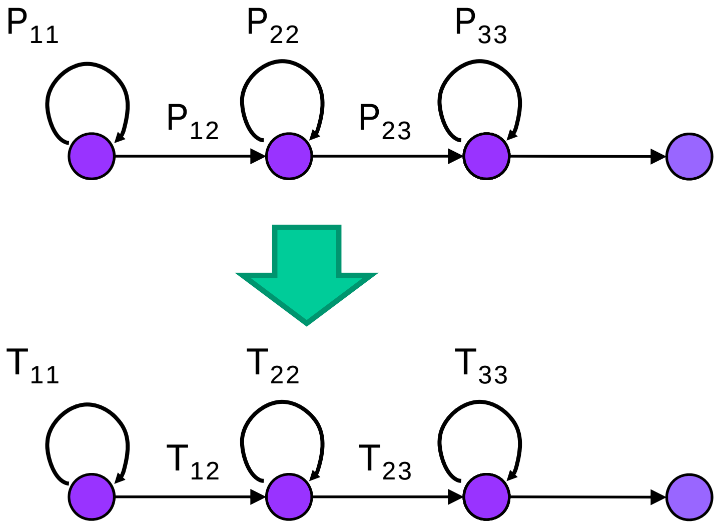


Markov chain

Data distributions

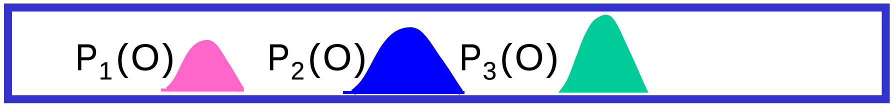
- This can be factored into two separate probabilistic entities
    - A probabilistic Markov chain with states and transitions
    - A set of data probability distributions, associated with the states

Relation to DTW: The transition structure



- The transitions in the HMM have associated probabilities
  - Derived by counting, as we saw earlier
- The transitions in the DTW template have associated penalties
  - $T_{ij} = -\log(P_{ij})$

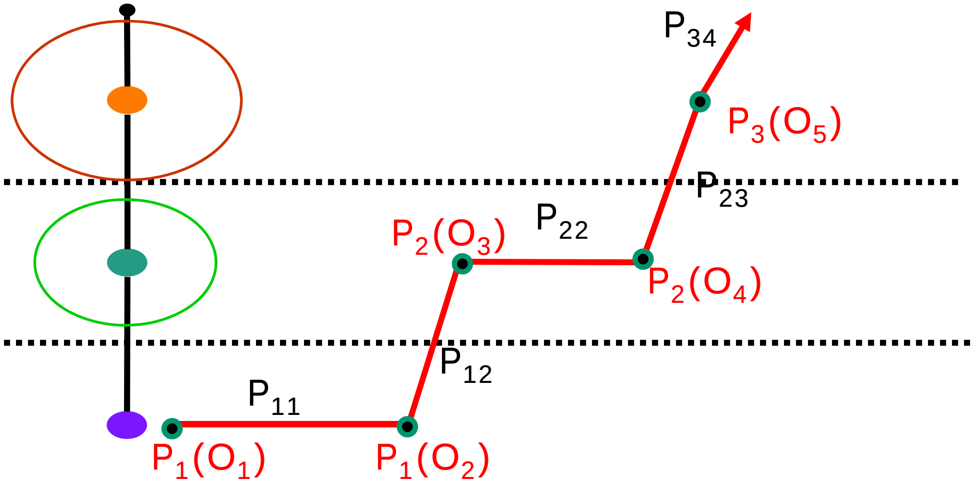
Relation to DTW: The Node Scores



Data distributions

- States in the HMM have associated state ouput distributions
  - Typically Gaussian
  - Means and variances are obtained from all the training vectors in the segment associated with the state
- Template Nodes in DTW has an associated node cost function
  - The cost associated with any node is dependent on the observation
  - $N_i (O) = -\log(P_i(O))$
- HMMs: We try to maximize probabilities
  - DTW: Minimize cost

Path Scores: Likelihoods, Log Likelihoods and Costs



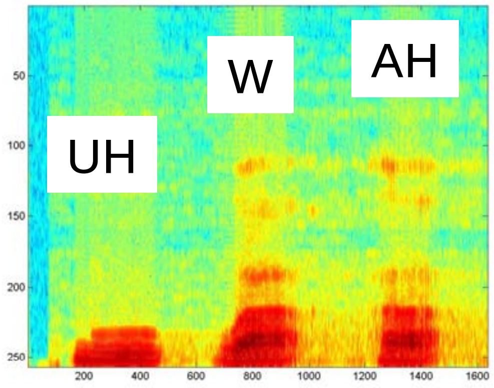
- Use probabilities or likelihoods instead of cost
  - Scores combines multiplicatively along a path
  - Path Score = $P_1(O_1) .P_{11} .P_1(O_2) .P_{12} .P_2(O_3) .P_{22} .P_2(O_4) .P_{23} .P_3(O_5) .P_{23}$

- Alternately use log probabilities as scores: $N_i(O) = \log(P_i(O))$, $T_{11} = \log(P_{11})$
  - Scores add as in DTW
  - Path Score = $N_1(O_1) + T_{11} + N_1(O_2) + T_{12} + N_2(O_3) + T_{22} + N_2(O_4) + T_{23} + N_3(O_5) + T_{23}$
    - Replace all “Min” operations in DTW by “Max”

- Alternately use *negative* log probabilities as cost: $N_i(O) = \log(P_i(O))$, $T_{11} = -\log(P_{11})$
  - *Cost* adds as in DTW
  - *Computation remains identical to DTW (with edge costs factored in)*

Modelling the process of speech production

- The HMM models the process underlying the observations as going through a number of states
  - For instance, in producing the sound “W”, it first goes through a state where it produces the sound “UH”, then goes into a state where it transitions from “UH” to “AH”, and finally to a state where it produced “AH”



- The true underlying process is the vocal tract here
  - Which roughly goes from the configuration for “UH” to the configuration for “AH”

HMMs are abstractions

- The states are not directly observed
  - Here states of the process are analogous to configurations of the vocal tract that produces the signal
  - We only hear the speech; we do not see the vocal tract
  - i.e. the states are *hidden*
- The interpretation of states is not always obvious
  - The vocal tract actually goes through a *continuum* of configurations
  - The model represents all of these using only a fixed number of states
- The model *abstracts* the process that generates the data
  - The system goes through a finite number of states
  - When in any state it can either remain at that state, or go to another with some probability
  - When at any states it generates observations according to a distribution associated with that state

HMM as a statistical model

- An HMM is a statistical model for a time-varying process
- The process is always in one of a countable number of states at any time

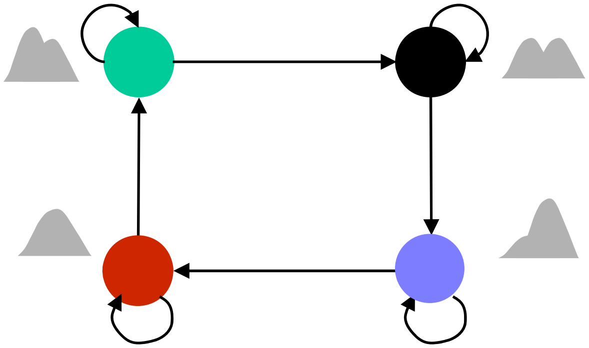
- When the process visits in any state, it generates an observation by a random draw from a distribution associated with that state

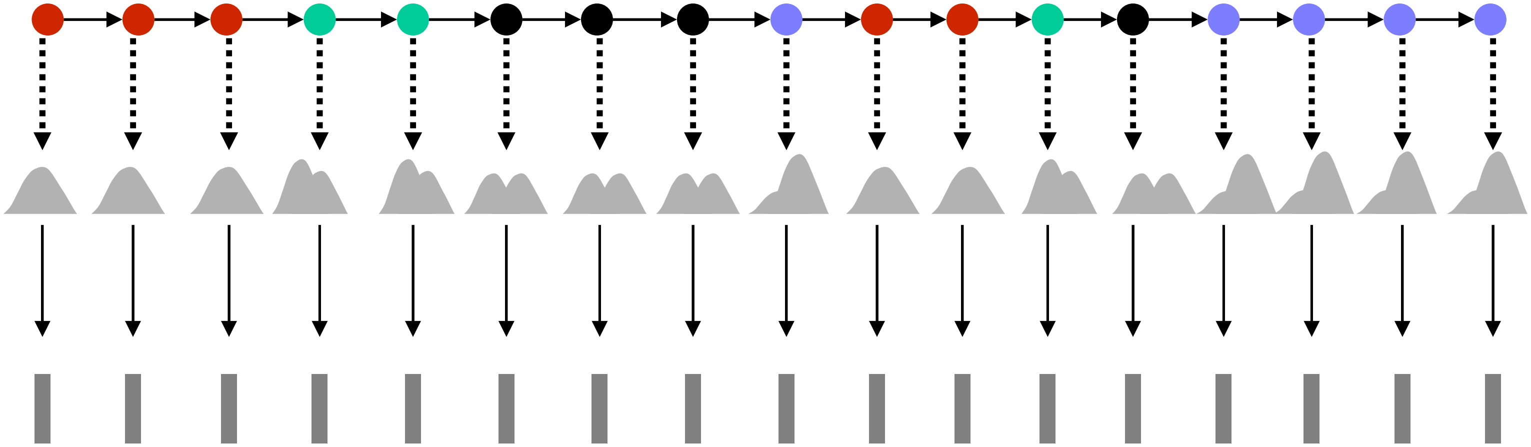
- The process constantly moves from state to state. The probability that the process will move to any state is determined solely by the current state
  - i.e. the dynamics of the process are Markovian

- The entire model represents a probability distribution over the sequence of observations
  - It has a specific probability of generating any particular sequence
  - The probabilities of all possible observation sequences sums to 1

How an HMM models a process

HMM assumed to be generating data





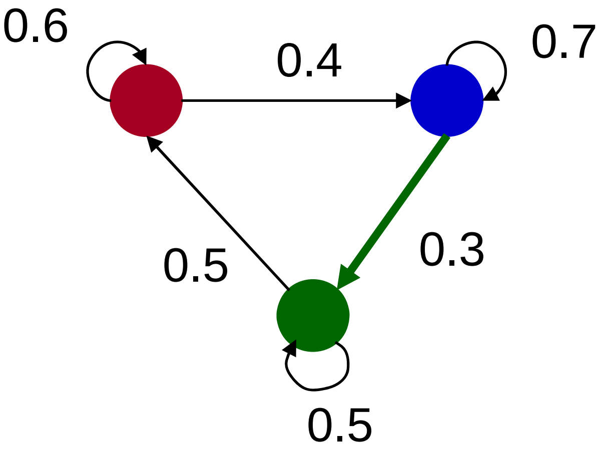
state sequence

state distributions

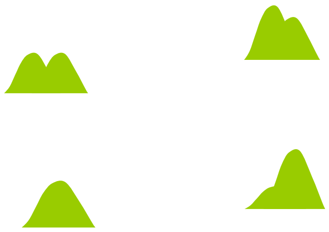
observation sequence

HMM Parameters

- The *topology* of the HMM
  - No. of states and allowed transitions
  - E.g. here we have 3 states and cannot go from the blue state to the red
- The transition probabilities
  - Often represented as a matrix as here
  - Tij is the probability that when in state i, the process will move to j
- The probability of beginning at a particular state
- The *state output distributions*



$$T = \begin{pmatrix} .6 & .4 & 0 \\ 0 & .7 & .3 \\ .5 & 0 & .5 \end{pmatrix}$$



HMM state output distributions

- The state output distribution represents the distribution of data produced from any state
- In the previous lecture we assumed the state output distribution to be Gaussian
  - Albeit largely in a DTW context

$$P(v) = Gaussian(v; m, C) = \frac{1}{\sqrt{2\pi|C|}} e^{-0.5(v-m)^T C^{-1}(v-m)}$$

- In reality, the distribution of vectors for any state need not be Gaussian
  - In the most general case it can be arbitrarily complex
  - The Gaussian is only a coarse representation of this distribution
- If we model the output distributions of states better, we can expect the model to be a better representation of the data

Gaussian Mixtures

- A Gaussian Mixture is literally a mixture of Gaussians. It is a weighted combination of several Gaussian distributions

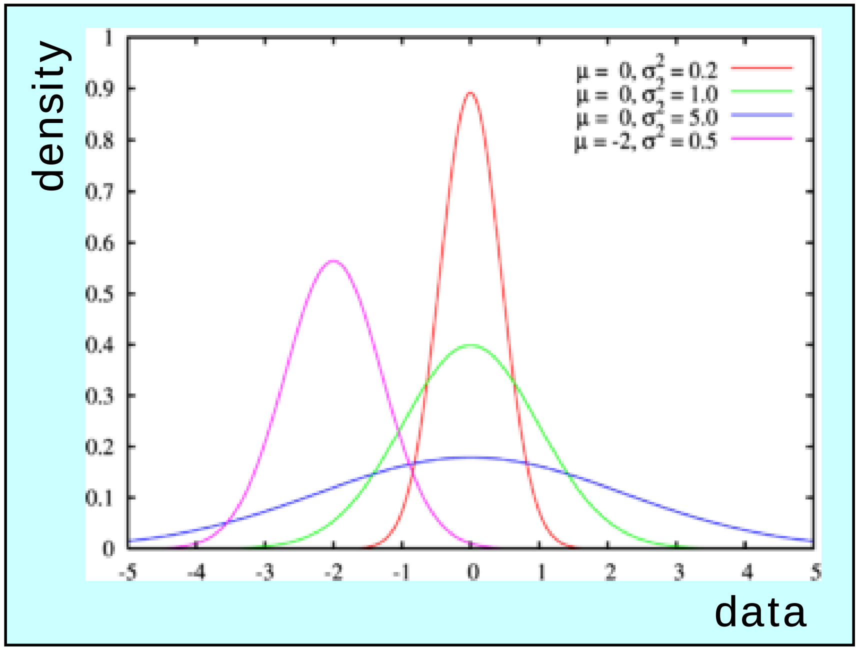
$$P(v) = \sum_{i=0}^{K-1} w_i Gaussian(v; m_i, C_i)$$

  - v is any data vector. P(v) is the probability given to that vector by the Gaussian mixture
  - K is the number of Gaussians being mixed
  - w_i is the mixture weight of the i^{th} Gaussian. m_i is its mean and C_i is its covariance

- The Gaussian mixture distribution is also a distribution
  - It is positive everywhere.
  - The total volume under a Gaussian mixture is 1.0.
  - Constraint: the mixture weights w_i must all be positive and sum to 1

Node Score: The Gaussian Distribution

- What does a Gaussian distribution look like?
- For a single (scalar) variable, it is a bell-shaped curve representing the density of data around the mean
- Example:



Four different scalar Gaussian distributions, with different means and variances

The mean is represented by μ, and variance by σ^2

μ and σ are the *parameters* of the Gaussian distribution

(Taken from Wikipedia)

The Scalar Gaussian Function

- The Gaussian density function (the bell curve) is:

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(x-\mu)^2}$$

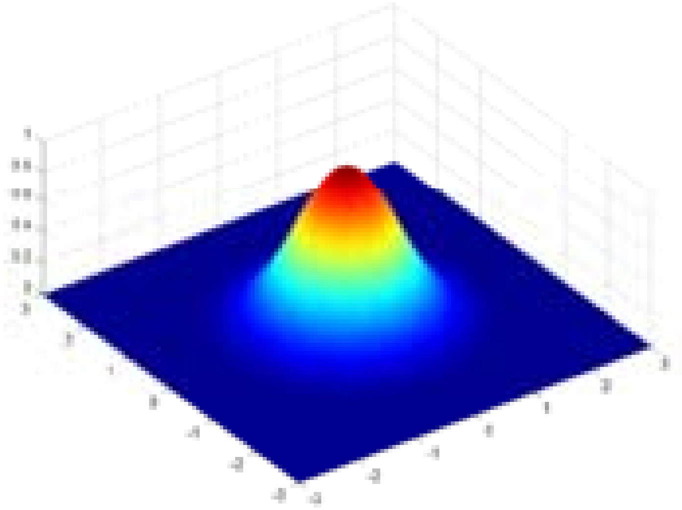
- $p(x)$ is the density function of the variable x, with mean μ and variance σ^2

- The attraction of the Gaussian function (regardless of how appropriate it is!) comes from how easily the mean and variance can be estimated from *sample data* x_1, x_2, x_3 … x_N
  - μ = (Sum x_i)/N
  - σ^2 = (Sum $(x_i^2 - \mu^2)$)/N

The 2-D Gaussian Distribution

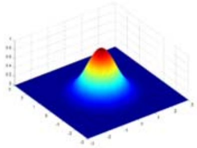
- Speech data are not scalar values, but vectors!

- Needs multi-variate (multi-dimensional) Gaussians
- Figure: A Gaussian for 2-D data
  - Shown as a 3-D plot



- Distributions for higher dimensions are tough to visualize!

The Multidimensional Gaussian Distribution



- Instead of variance, the multidimensional Gaussian has a *covariance matrix*
- The multi-dimensional Gaussian distribution of a vector variable x with mean μ and covariance Σ is given by:

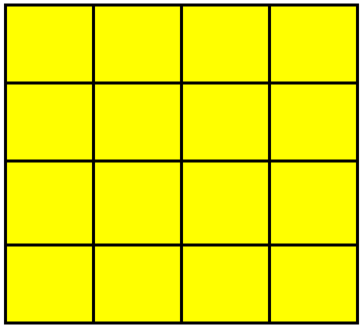
$$f(x) = (2\pi)^{-N/2} \det(\Sigma)^{-1/2} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

  - where *N* is the vector dimensionality, and *det* is the determinant function

- The complexity in a full multi-dimensional Gaussian distribution comes from the covariance matrix, which accounts for *dependencies* between the dimensions

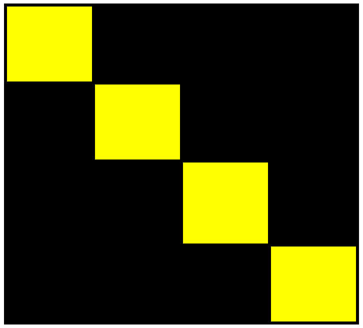
The Diagonal Covariance Matrix

Full covariance:
all elements are
non-zero



$-0.5(x-\mu)^T C^{-1}(x-\mu)$

Diagonal covariance:
off-diagonal elements
are zero



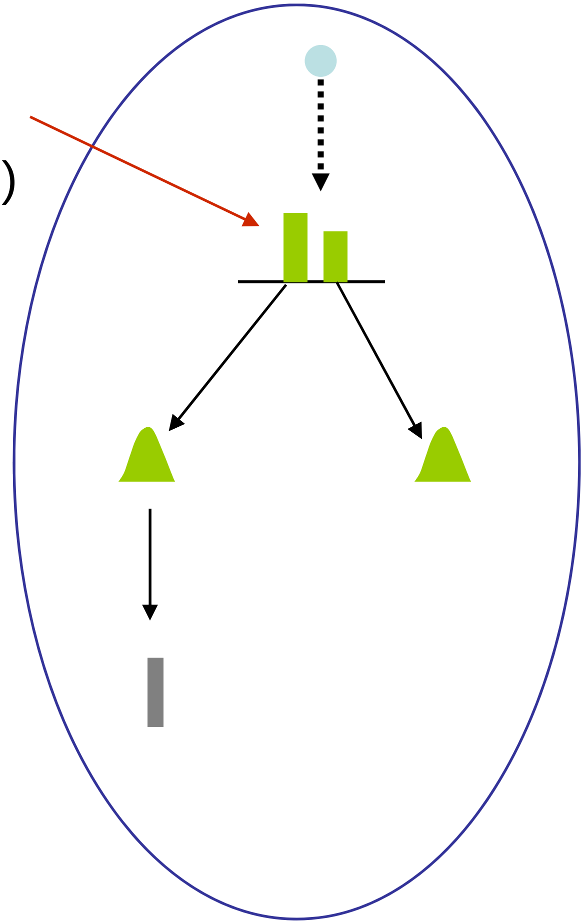
$\Sigma_i (x_i-\mu_i)^2 / 2\sigma_i^2$

- In speech recognition, we frequently assume that the feature vector dimensions are all *independent* of each other
- *Result*: The covariance matrix is reduced to a diagonal form
  - The exponential term becomes, simply:
    $(\Sigma_i (x_i - \mu_i)^2/\sigma_i^2)/2$, i going over all vector dimensions
  - The determinant of the diagonal Σ matrix is easy to compute
- Further, each σ_i^2 (the i-th digonal element in the covariance matrix) is easily estimated from x_i and μ_i like a scalar

Generating an observation from a Gaussian mixture state distribution

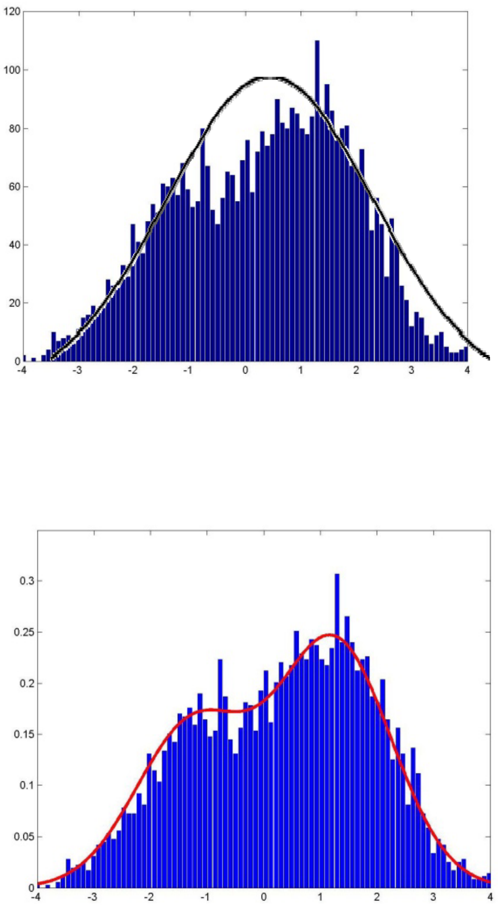
First draw the identity of the Gaussian from the a priori probability distribution of Gaussians (mixture weights)

Then draw a vector from the selected Gaussian



Gaussian Mixtures

- A Gaussian mixture can represent data distributions far better than a simple Gaussian
- The two panels show the histogram of an unknown random variable
- The first panel shows how it is modeled by a simple Gaussian
- The second panel models the histogram by a mixture of two Gaussians
- Caveat: It is hard to know the optimal number of Gaussians in a mixture distribution for any random variable



HMMs with Gaussian mixture state distributions

- The parameters of an HMM with Gaussian mixture state distributions are:
  - π the set of initial state probabilities for all states
  - ***T*** the matrix of transition probabilities
  - A Gaussian mixture distribution for every state in the HMM. The Gaussian mixture for the *i*th state is characterized by
    - K_i, the number of Gaussians in the mixture for the ith state
    - The set of mixture weights $w_{i,j}$ $0<j<K_i$
    - The set of Gaussian means $m_{i,j}$ $0 <j<K_i$
    - The set of Covariance matrices $C_{i,j}$ $0 < j <K_i$

Three Basic HMM Problems

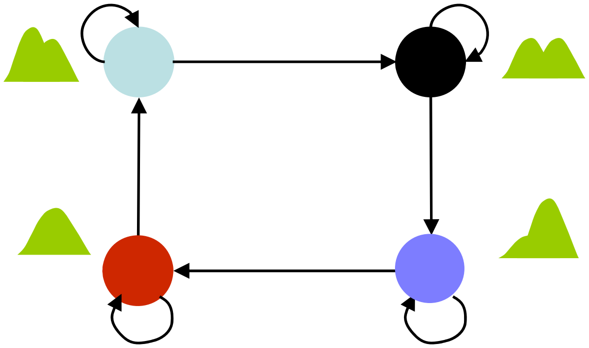
- Given an HMM:
  - What is the probability that it will generate a specific observation sequence
  - Given a observation sequence, how do we determine which observation was generated from which state
    - The state segmentation problem
  - How do we *learn* the parameters of the HMM from observation sequences

Computing the Probability of an Observation Sequence

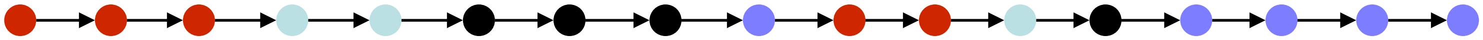
- Two aspects to producing the observation:
  - – Precessing through a sequence of states
  - – Producing observations from these states

Precessing through states

HMM assumed to be generating data



state sequence



- The process begins at some state (red) here
- From that state, it makes an allowed transition
  - To arrive at the same or any other state
- From that state it makes another allowed transition
  - And so on

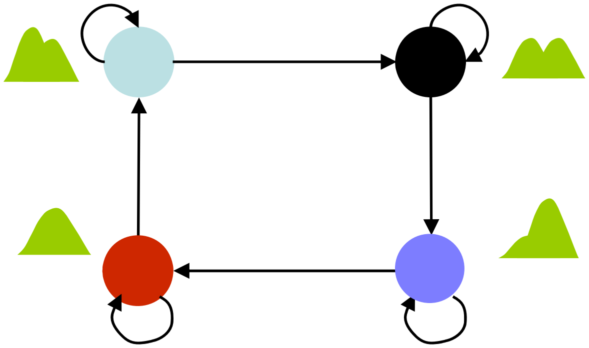
Probability that the HMM will follow a particular state sequence

$$P(s_1, s_2, s_3, \ldots) = P(s_1)P(s_2|s_1)P(s_3|s_2)\ldots$$

- $P(s_1)$ is the probability that the process will initially be in state s_1
- $P(s_i \mid s_i)$ is the transition probability of moving to state s_i at the next time instant when the system is currently in s_i
  - Also denoted by P_{ij} earlier
  - Related to edge scores in DTW as $T_{ij} = -\log(P(s_i \mid s_i))$

Generating Observations from States

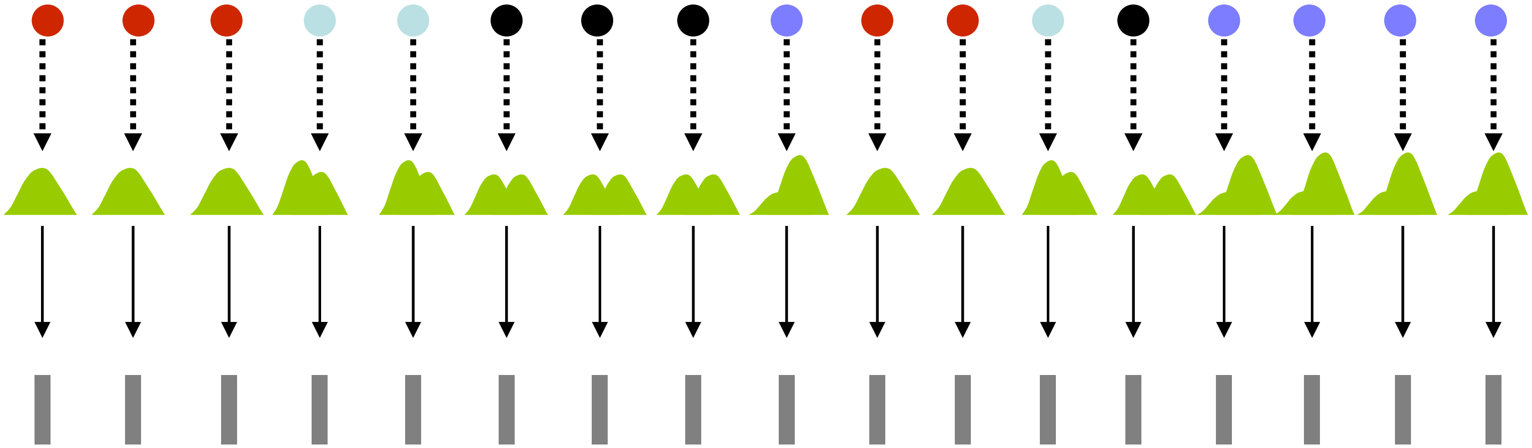
HMM assumed to be generating data



state sequence

state distributions

observation sequence



- At each time it generates an observation from the state it is in at that time

Probability that the HMM will generate a particular observation sequence given a state sequence (state sequence known)

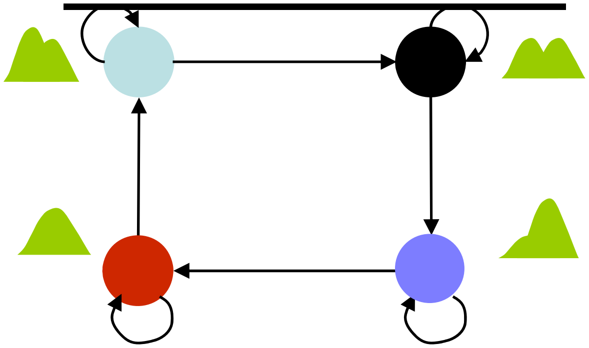
$$P(o_1, o_2, o_3, \ldots | s_1, s_2, s_3, \ldots) = P(o_1|s_1)P(o_2|s_2)P(o_3|s_3)\ldots$$

Computed from the Gaussian or Gaussian mixture for state s_1

- $P(o_i \mid s_i)$ is the probability of generating observation o_i when the system is in state s_i
  - Related to node scores in DTW trellis as: $N_i(O) = -\log(P(o_i \mid s_i))$

Precessing through States and Producing Observations

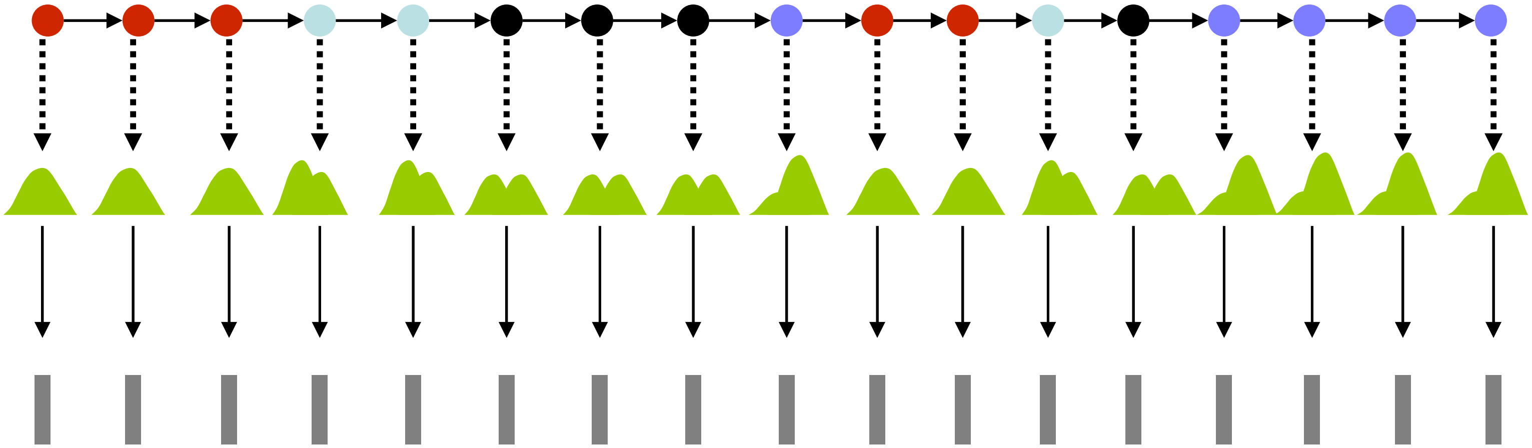
HMM assumed to be generating data



state sequence

state distributions

observation sequence



- At each time it produces an observation and makes a transition

Probability that the HMM will generate a particular state sequence and from it, a particular observation sequence

$$P(o_1, o_2, o_3, \ldots, s_1, s_2, s_3, \ldots) =$$

$$P(o_1, o_2, o_3, \ldots | s_1, s_2, s_3, \ldots) P(s_1, s_2, s_3, \ldots) =$$

$$P(o_1|s_1) P(o_2|s_2) P(o_3|s_3) \ldots P(s_1) P(s_2|s_1) P(s_3|s_2) \ldots$$

Probability of Generating an Observation Sequence

- If only the observation is known, the precise state sequence followed to produce it is not known
- All possible state sequences must be considered

$$P(o_1, o_2, o_3, \ldots) = \sum_{\substack{all.possible \\ state.sequences}} P(o_1, o_2, o_3, \ldots, s_1, s_2, s_3, \ldots) =$$

$$\sum_{\substack{all.possible \\ state.sequences}} P(o_1|s_1) P(o_2|s_2) P(o_3|s_3) \ldots P(s_1) P(s_2|s_1) P(s_3|s_2) \ldots$$

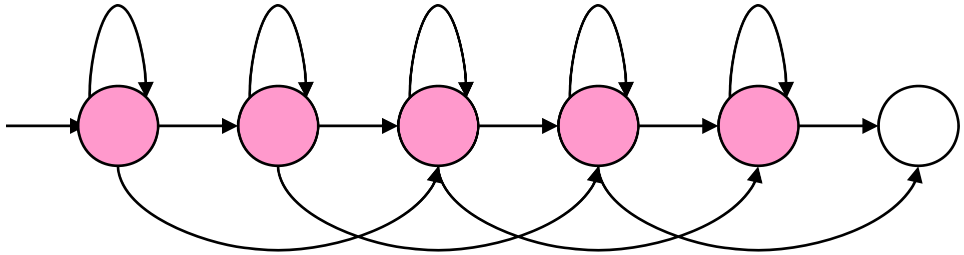
Computing it Efficiently

- Explicit summing over all state sequences is not efficient
    - A very large number of possible state sequences
    - For long observation sequences it may be intractable

- Fortunately, we have an efficient algorithm for this: The forward algorithm

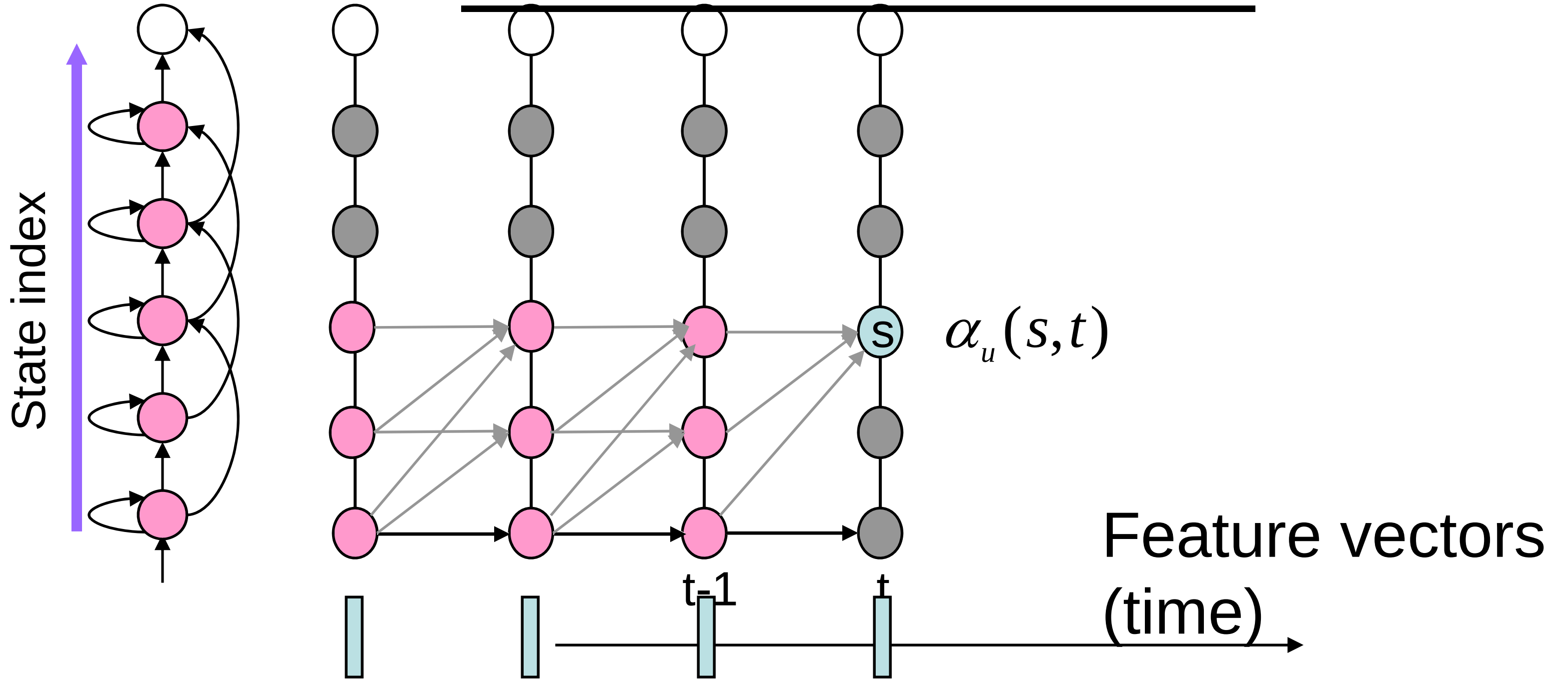
- At each time, for each state compute the total probability of all state sequences that generate observations until that time and end at that state

Illustrative Example



- Consider a generic HMM with 5 states and a "terminating state". We wish to find the probability of the best state sequence for an observation sequence assuming it was generated by this HMM
  - $P(s_i)$ = 1 for state 1 and 0 for others
  - The arrows represent transition for which the probability is not 0. $P(s_i \mid s_i) = a_{ij}$
  - We sometimes also represent the state output probability of s_i as $P(o_t \mid s_i) = b_i(t)$ for brevity

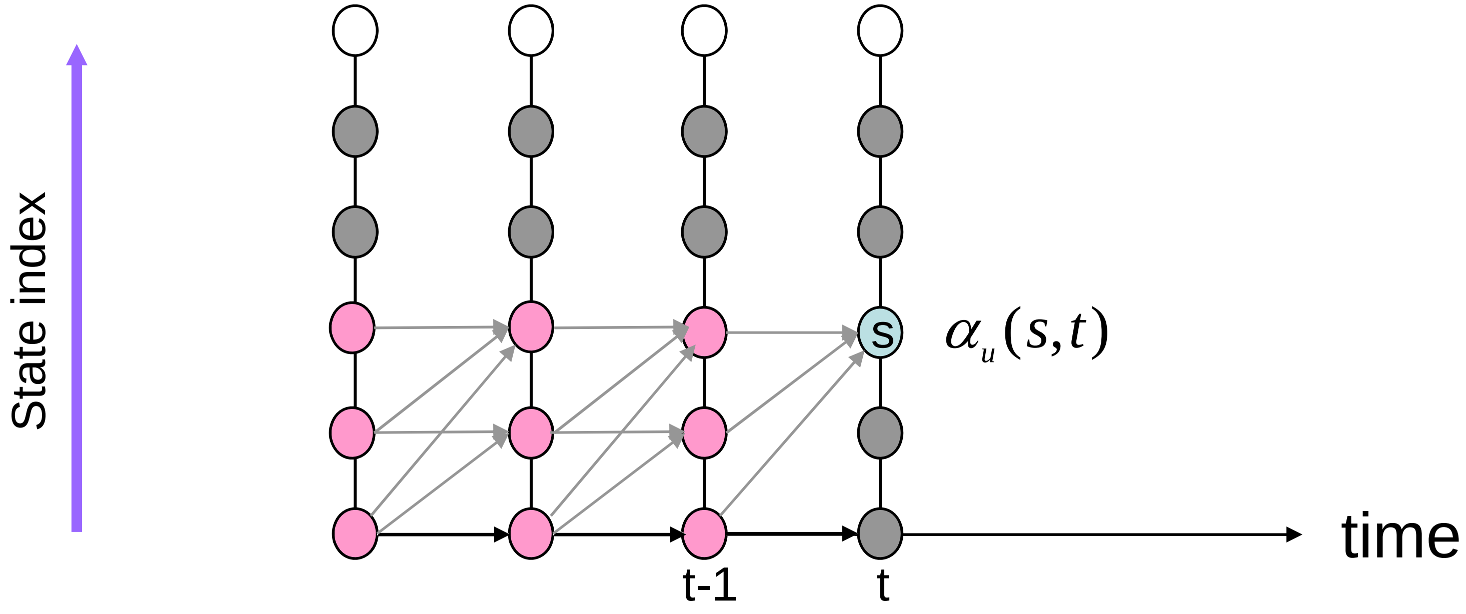
Diversion: The Trellis



- The trellis is a graphical representation of all possible paths through the HMM to produce a given observation
  - Analogous to the DTW search graph / trellis
- The Y-axis represents HMM states, X axis represents observations
- Every edge in the graph represents a valid transition in the HMM over a single time step
- Every node represents the event of a particular observation being generated from a particular state

The Forward Algorithm

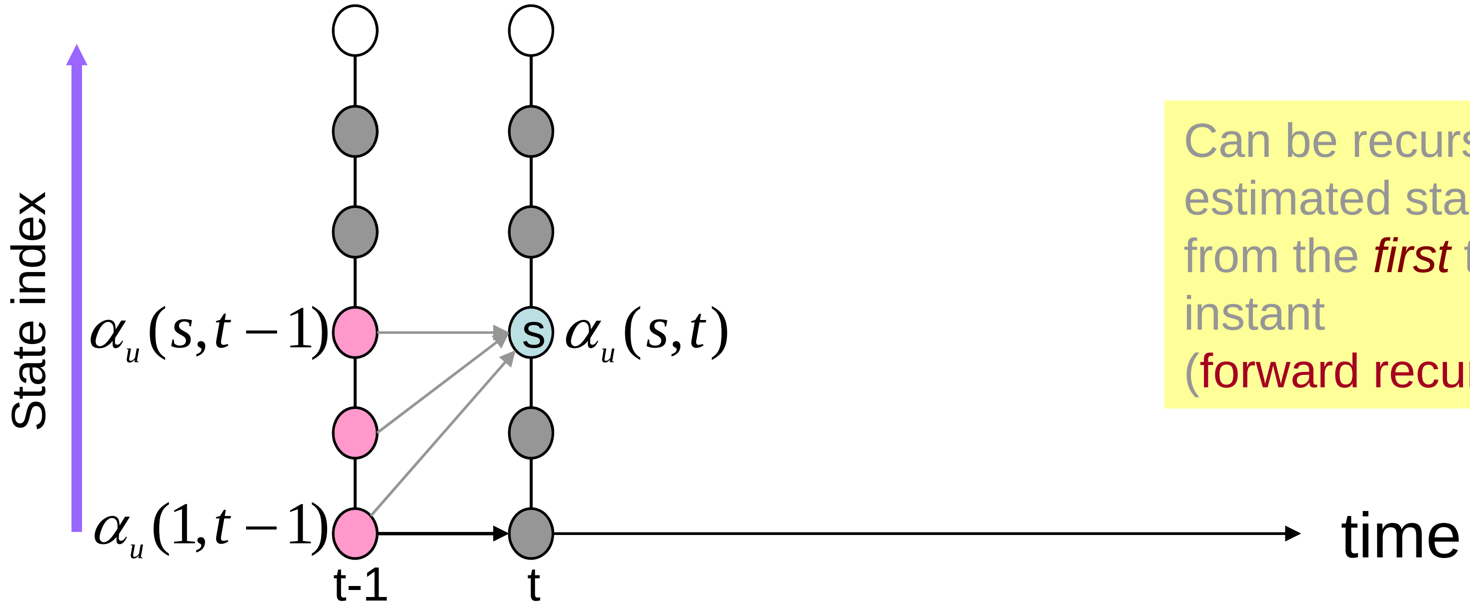
$$\alpha_u(s,t) = P(x_{u,1}, x_{u,2}, \ldots, x_{u,t}, state(t) = s | \lambda)$$



- $\alpha_u(s,t)$ is the total probability of ALL state sequences that end at state *s* at time *t*, and all observations until x_t

The Forward Algorithm

$$\alpha_u(s,t) = P(x_{u,1}, x_{u,2}, \ldots, x_{u,t}, state(t) = s | \lambda)$$



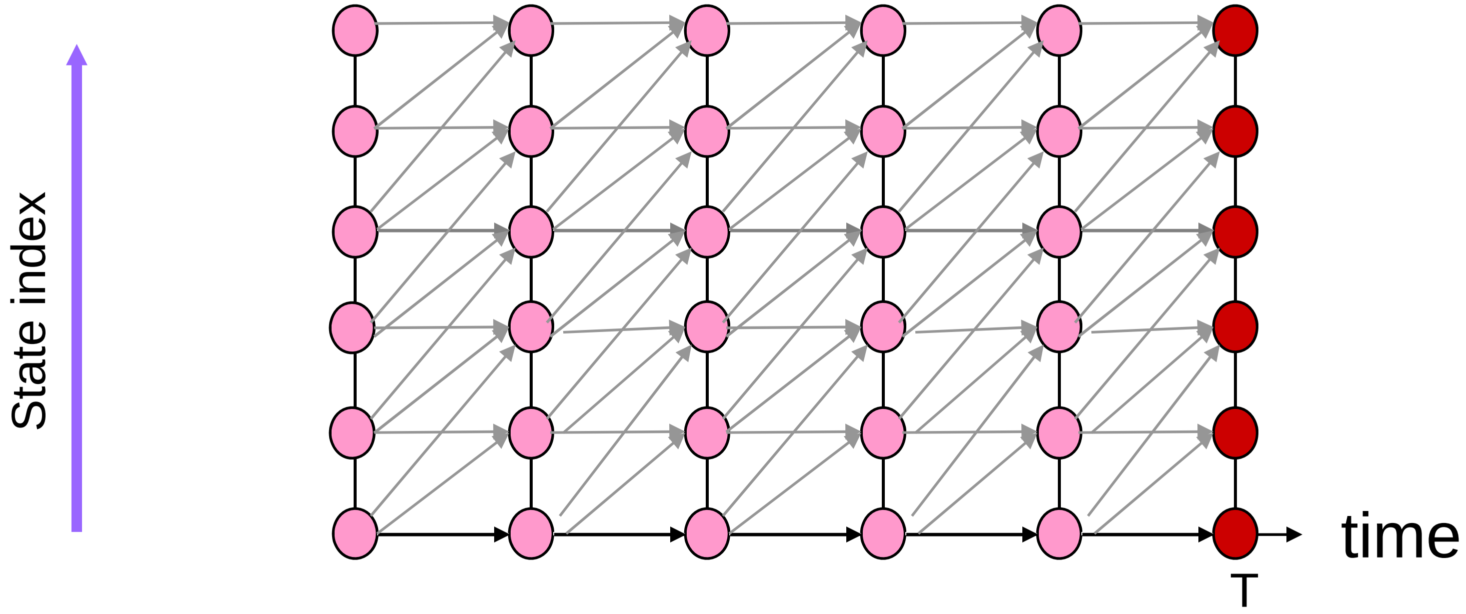
Can be recursively estimated starting from the *first* time instant (forward recursion)

$$\alpha_u(s,t) = \sum_{s'} \alpha_u(s',t-1) P(s|s') P(x_{u,t}|s)$$

- $\alpha_u(s,t)$ can be recursively computed in terms of $\alpha_u(s',t')$, the forward probabilities at time t-1

The Forward Algorithm

$$Totalprob = \sum_{s} \alpha_u(s,T)$$

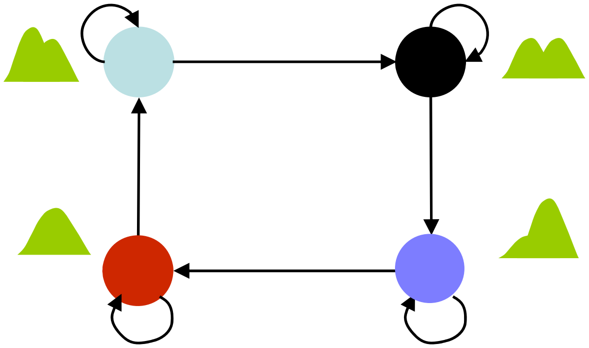


- In the final observation the alpha at each state gives the probability of all state sequences ending at that state
- The total probability of the observation is the sum of the alpha values at all states

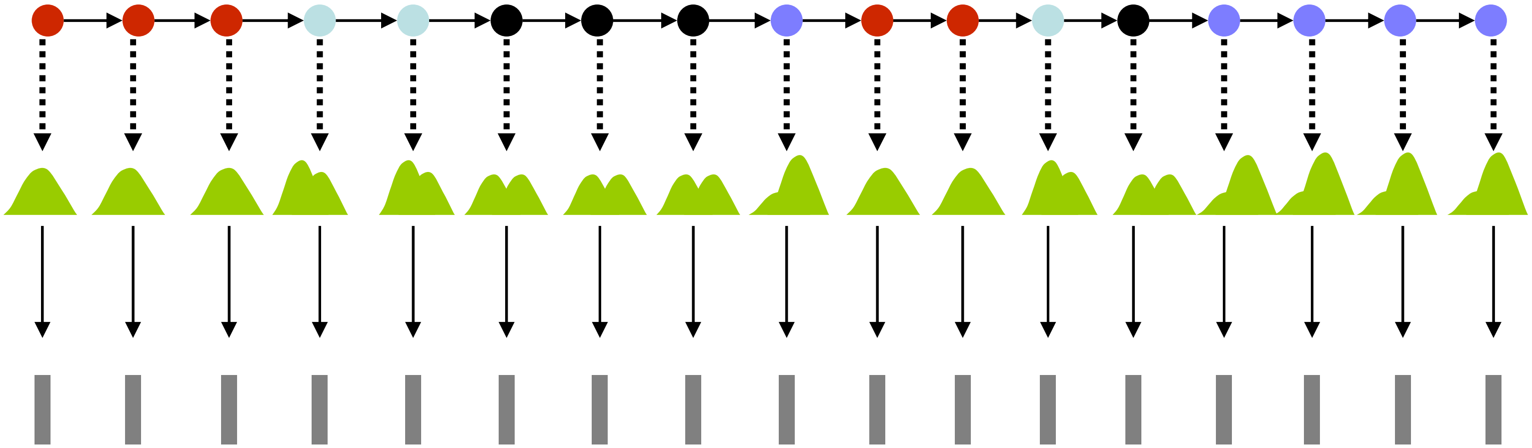
Problem 2: The state segmentation problem

- Given only a sequence of observations, how do we determine which sequence of states was followed in producing it?

The HMM as a generator



HMM assumed to be generating data



state sequence

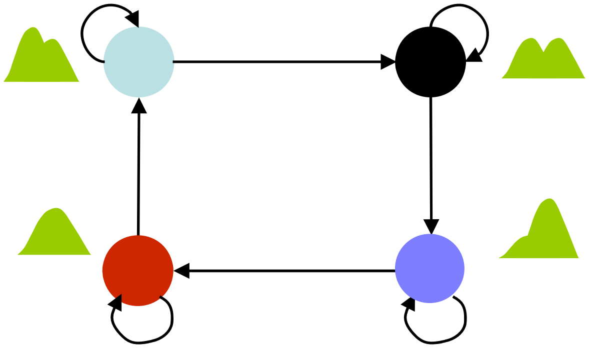
state distributions

observation sequence

- The process goes through a series of states and produces observations from them

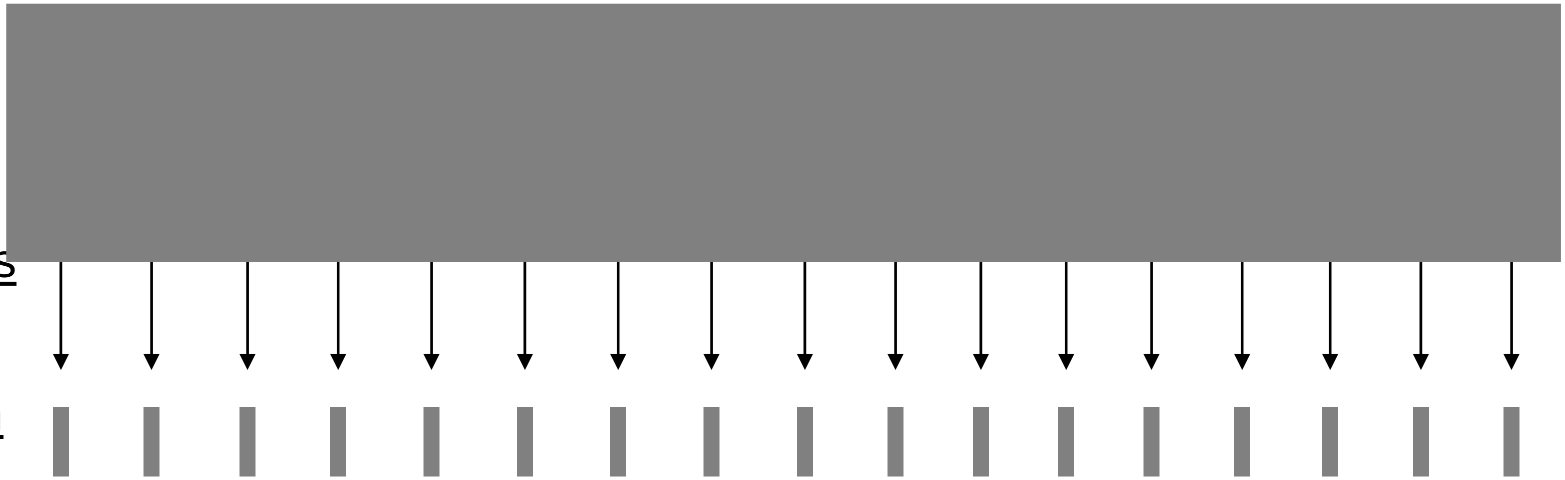
States are Hidden

HMM assumed to be generating data



state sequence

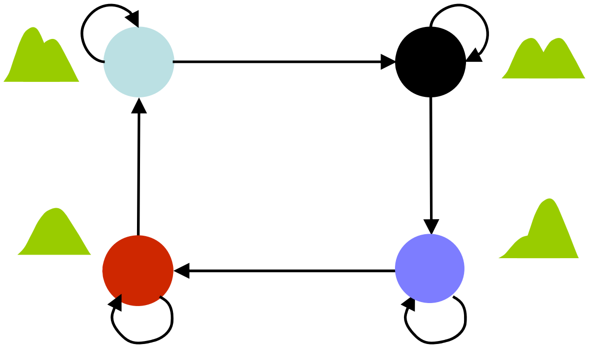
state distributions



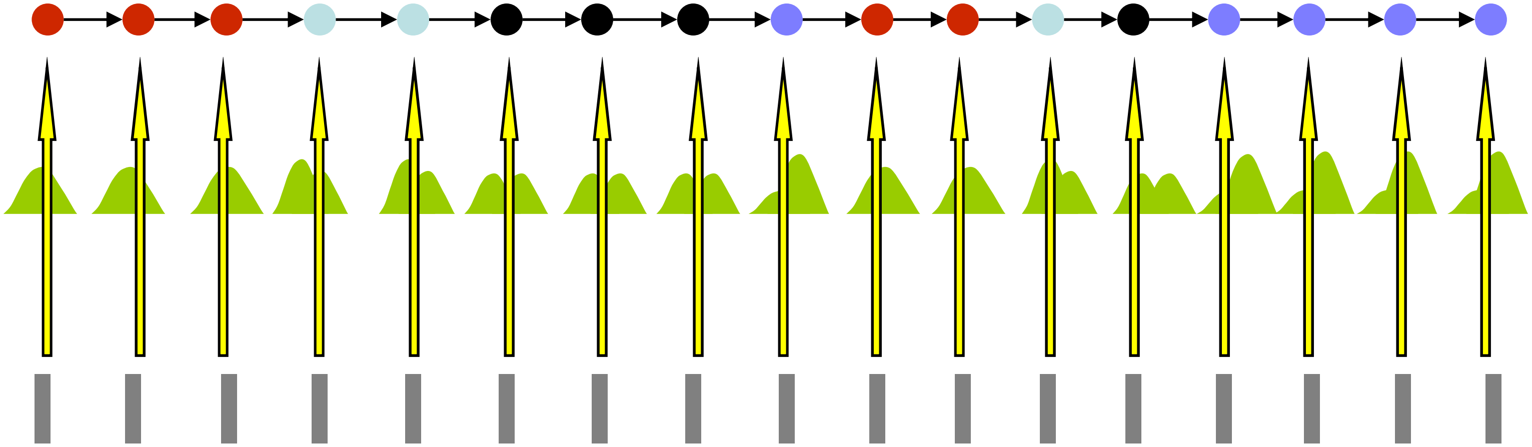
observation sequence

- The observations do not reveal the underlying state

The state segmentation problem



HMM assumed to be generating data



state sequence

state distributions

observation sequence

- State segmentation: Estimate state sequence given observations

Estimating the State Sequence

- Any number of state sequences could have been traversed in producing the observation
  - In the worst case *every* state sequence may have produced it

- Solution: Identify the most *probable* state sequence
  - The state sequence for which the probability of progressing through that sequence and gen erating the observation sequence is maximum
  - i.e $P(o_1, o_2, o_3, \ldots, s_1, s_2, s_3, \ldots) =$ is maximum

Estimating the state sequence

- Once again, exhaustive evaluation is impossibly expensive
- But once again a simple dynamic-programming solution is available

$$P(o_1, o_2, o_3, \ldots, s_1, s_2, s_3, \ldots) =$$

$$P(o_1|s_1)P(o_2|s_2)P(o_3|s_3)\ldots P(s_1)P(s_2|s_1)P(s_3|s_2)\ldots$$

- Needed:

$$\arg\max_{s_1, s_2, s_3, \ldots} P(o_1 | s_1)P(s_1)P(o_2 | s_2)P(s_2 | s_1)P(o_3 | s_3)P(s_3 | s_2)$$

Estimating the state sequence

- Once again, exhaustive evaluation is impossibly expensive
- But once again a simple dynamic-programming solution is available

$$P(o_1, o_2, o_3, \ldots, s_1, s_2, s_3, \ldots) =$$

$$P(o_1|s_1)P(o_2|s_2)P(o_3|s_3)\ldots P(s_1)P(s_2|s_1)P(s_3|s_2)\ldots$$

- Needed:

$$\arg\max_{s_1, s_2, s_3, \ldots} P(o_1 \mid s_1)P(s_1)P(o_2 \mid s_2)P(s_2 \mid s_1)P(o_3 \mid s_3)P(s_3 \mid s_2)$$

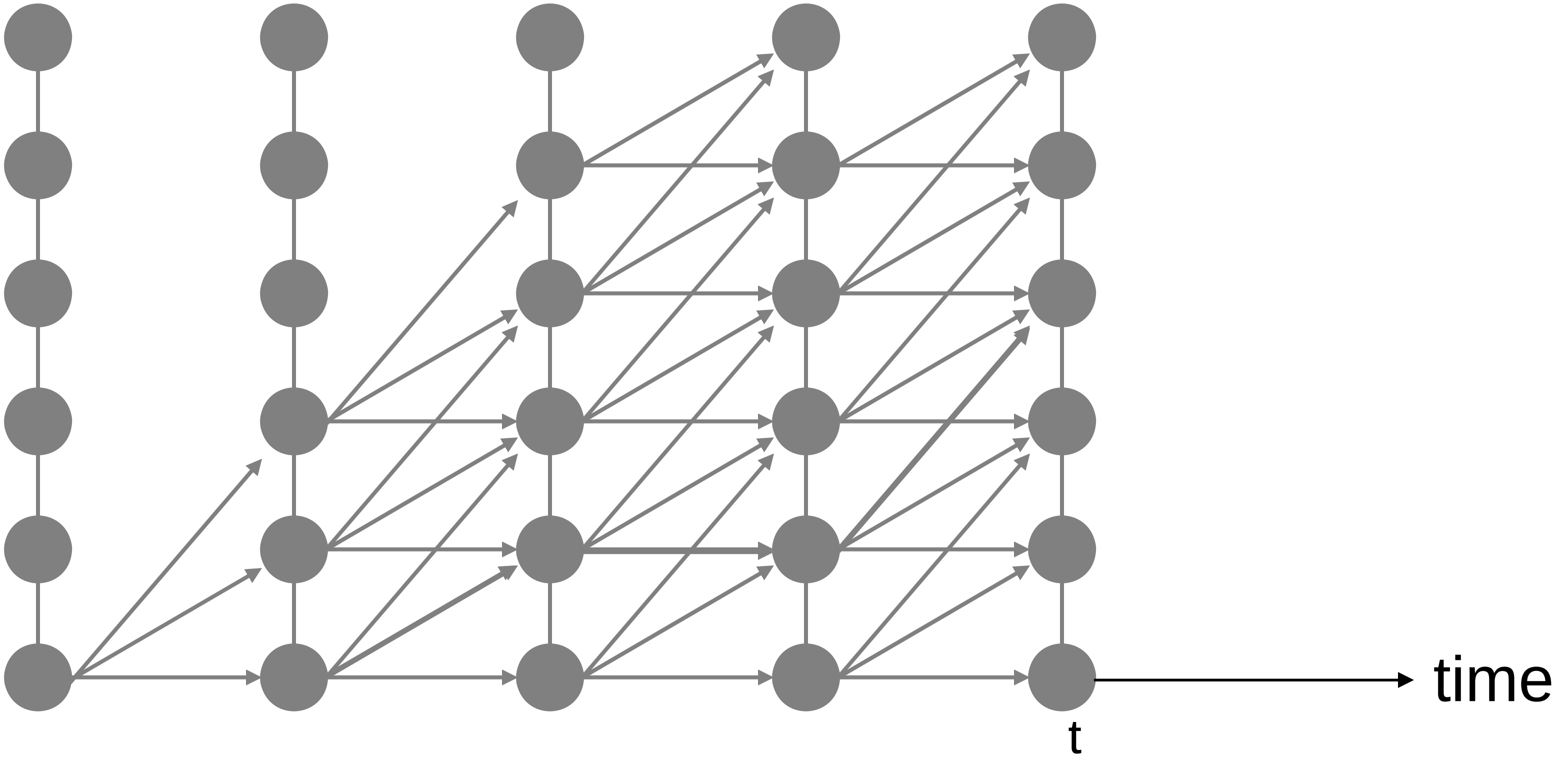
The state sequence

- The probability of a state sequence ?,?,?,?,s_{x},s_{y} ending at time *t* is simply
  - $\mathrm{P}(?,?,?,?,\ \mathrm{s}_{\mathrm{x}},\mathrm{s}_{\mathrm{y}}) = \mathrm{P}(?,?,?,?,\ \mathrm{s}_{\mathrm{x}})\ \mathrm{P}(o_t|s_y)\mathrm{P}(s_y|s_x)$

- The *best* state sequence that ends with s_x,s_y at *t* will have a probability equal to the probability of the best state sequence ending at *t-1* at s_x times $\mathrm{P}(o_t|s_y)\mathrm{P}(s_y|s_x)$
  - Since the last term is independent of the state sequence leading to s_x at *t-1*

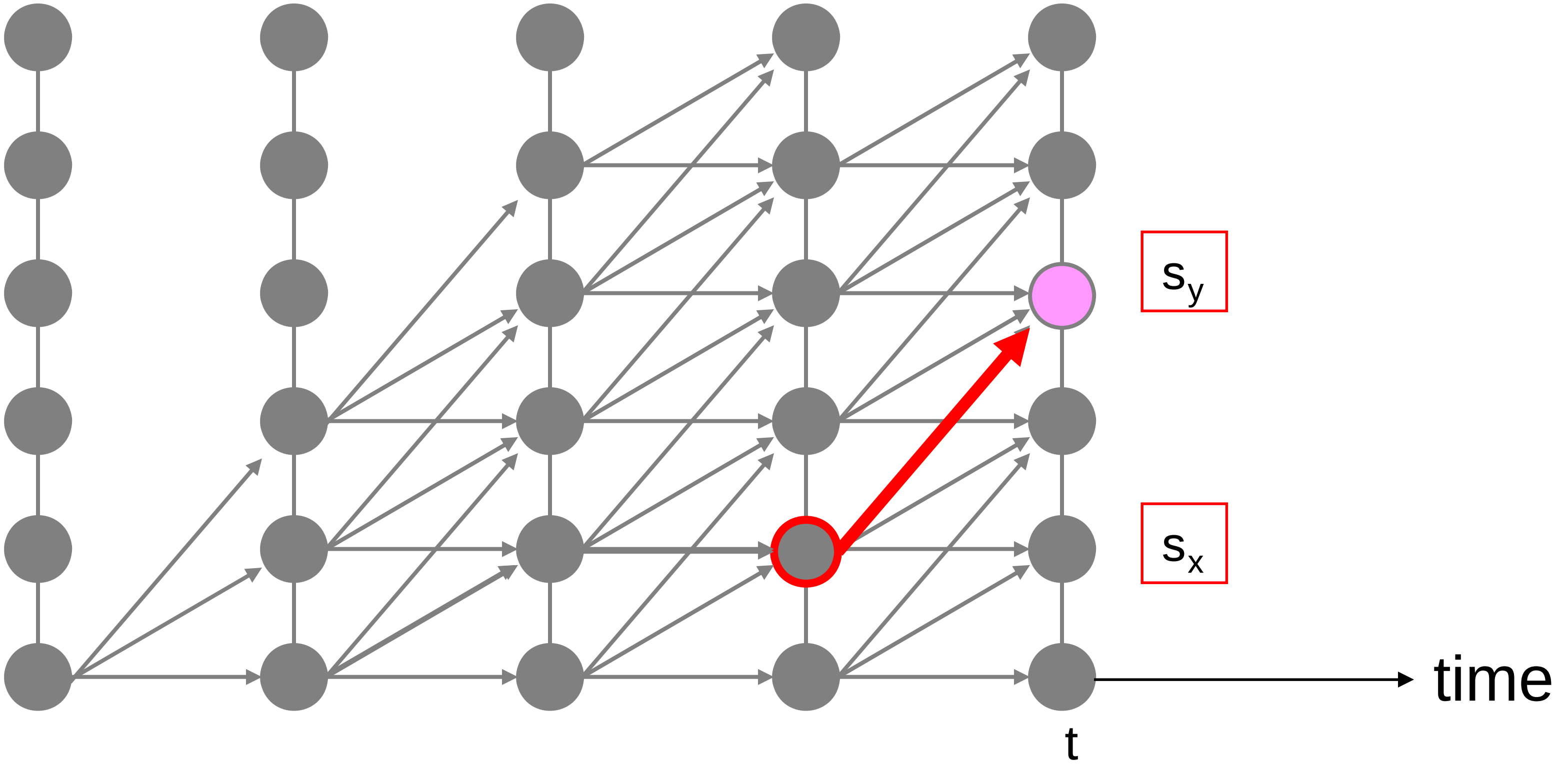
Trellis

- The graph below shows the set of all possible state sequences through this HMM in five time intants



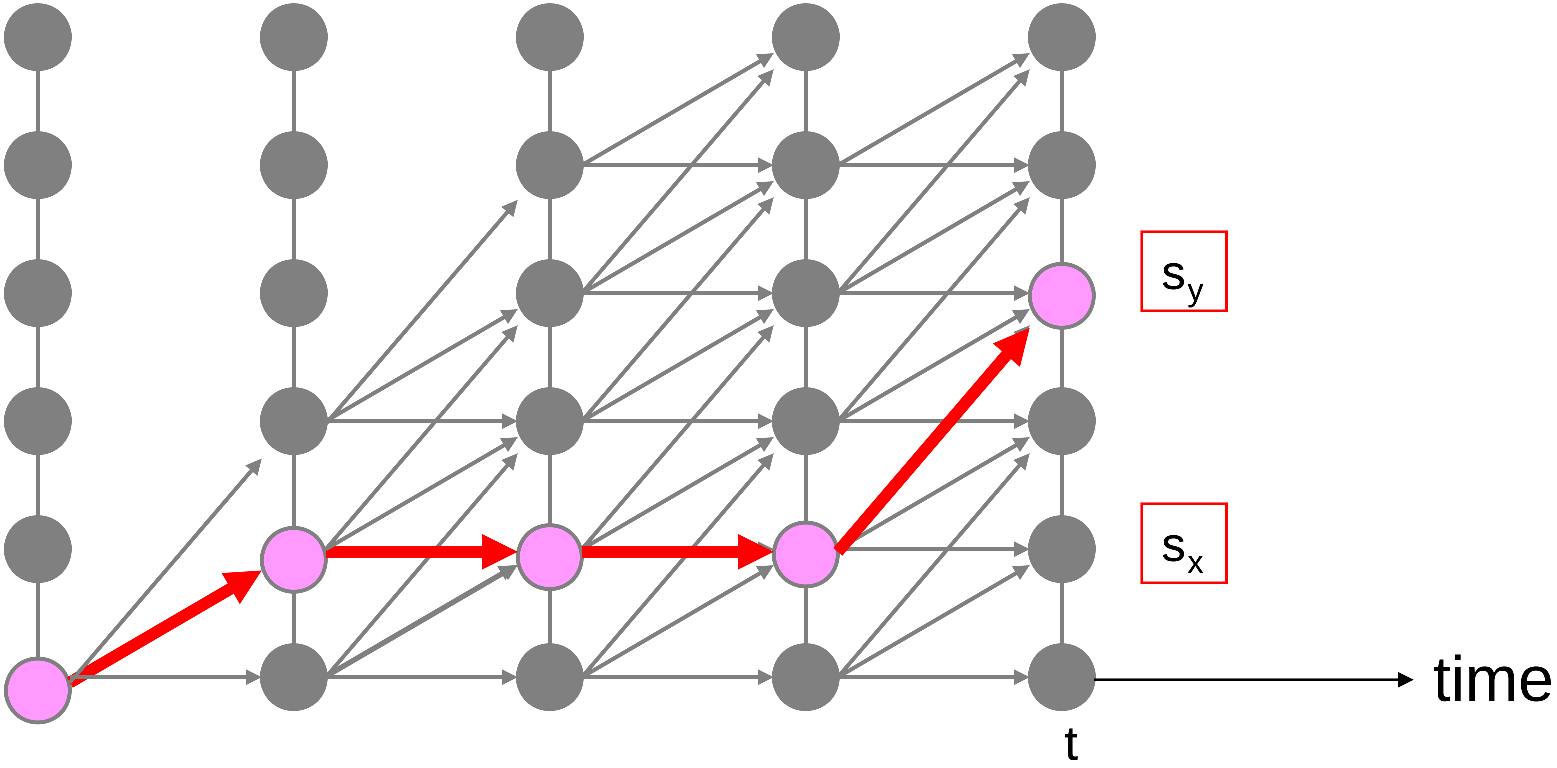
The cost of extending a state sequence

- The cost of extending a state sequence ending at s_x is only dependent on the transition from s_x to s_y, and the observation probability at s_y



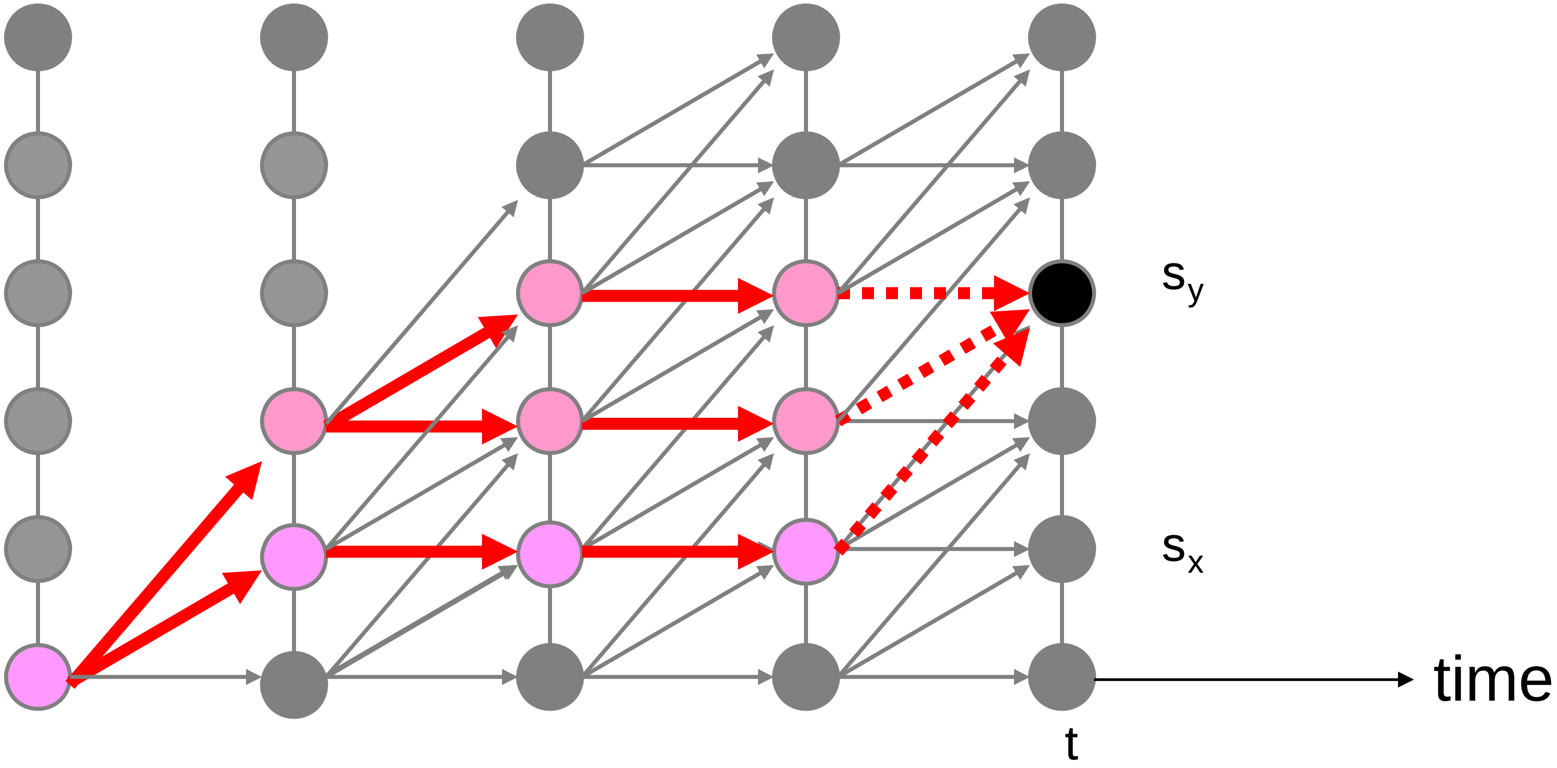
The cost of extending a state sequence

- The best path to s_y through s_x is simply an extension of the best path to s_x



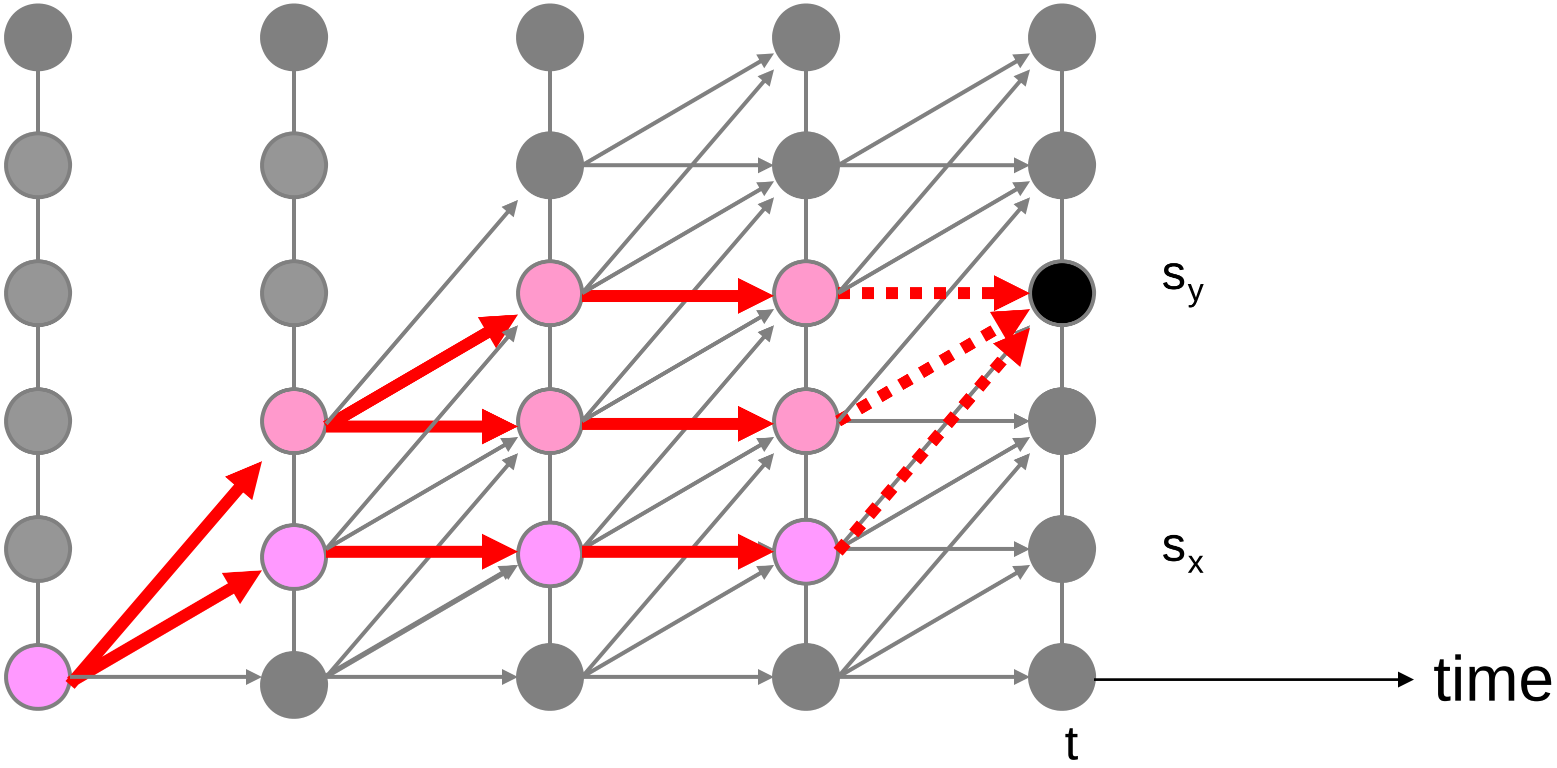
The Recursion

- The overall best path to s_x is an extension of the best path to one of the states at the previous time



The Recursion

- Bestpath prob(s_y,t) =
  Best (Bestpath prob($s_?$,t) * P(s_y | $s_?$) * P(o_t|s_y))



Finding the best state sequence

- This gives us a simple recursive formulation to find the overall best state sequence:

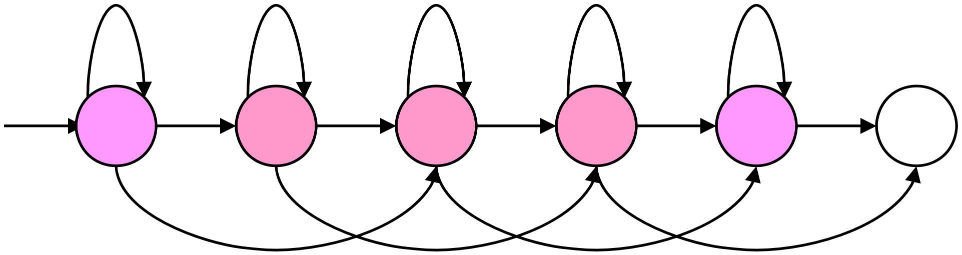
1. The best state sequence $X_{1,i}$ of length 1 ending at state s_i is simply s_i.
   - The probability $C(X_{1,i})$ of $X_{1,i}$ is $P(o_1 \mid s_i)\ P(s_i)$
2. The best state sequence of length $t+1$ is simply given by
   - $(\text{argmax}_{X_{t,i}}\ C(X_{t,i})P(o_{t+1} \mid s_j)\ P(s_j \mid s_i))\ s_i$
3. The best overall state sequence for an utterance of length T is given by argmax $_{X_{t,i}\ s_j}\ C(X_{T,i})$
   - The state sequence of length T with the highest overall probability

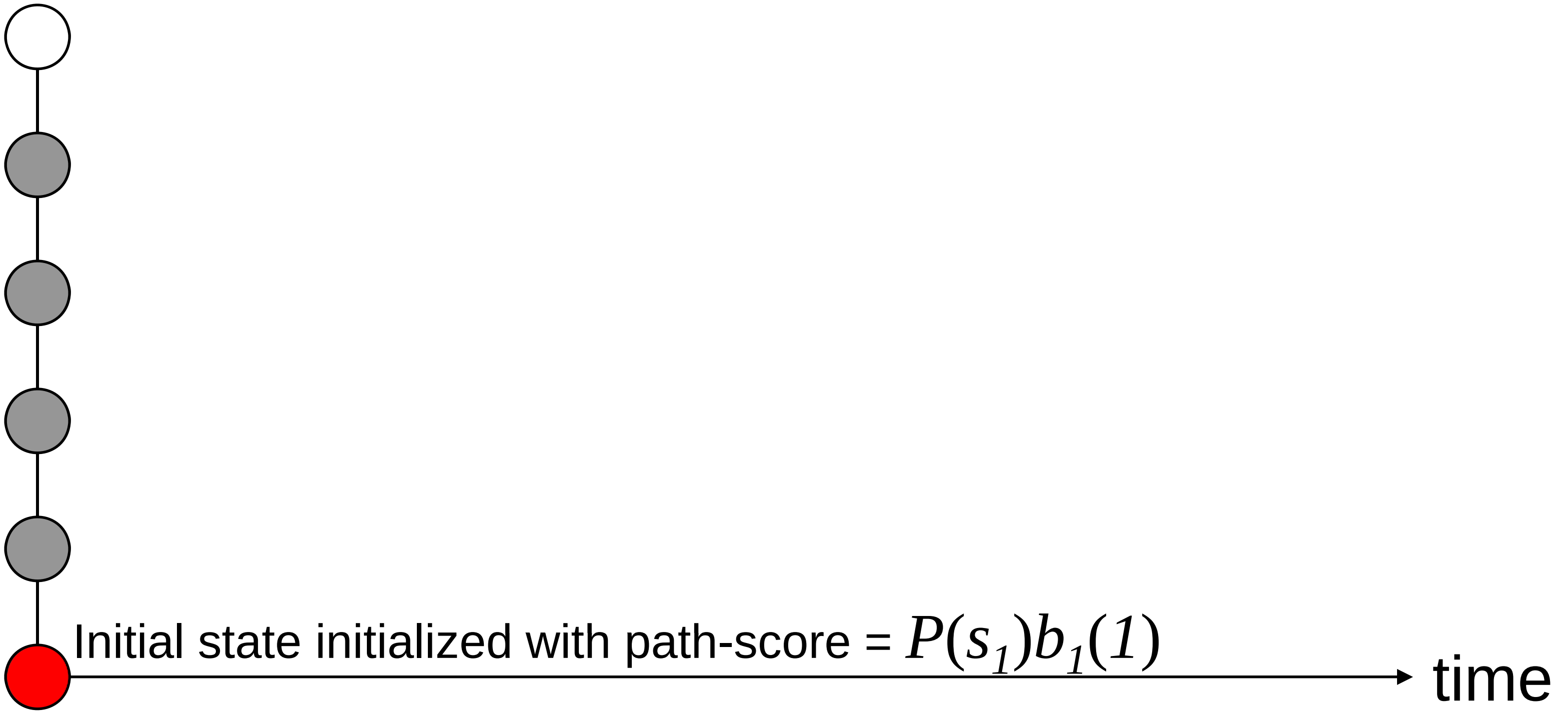
Finding the best state sequence

- The simple algorithm just presented is called the VITERBI algorithm in the literature
  - After A.J.Viterbi, who invented this dynamic programming algorithm for a completely different purpose: decoding error correction codes!

- The Viterbi algorithm can also be viewed as a breadth-first graph search algorithm
  - The HMM forms the Y axis of a 2-D plane
    - Edge costs of this graph are transition probabilities P(s|s). Node costs are P(o|s)
  - A linear graph with every node at a time step forms the X axis
  - A trellis is a graph formed as the crossproduct of these two graphs
  - The Viterbi algorithm finds the best path through this graph

Viterbi Search (contd.)

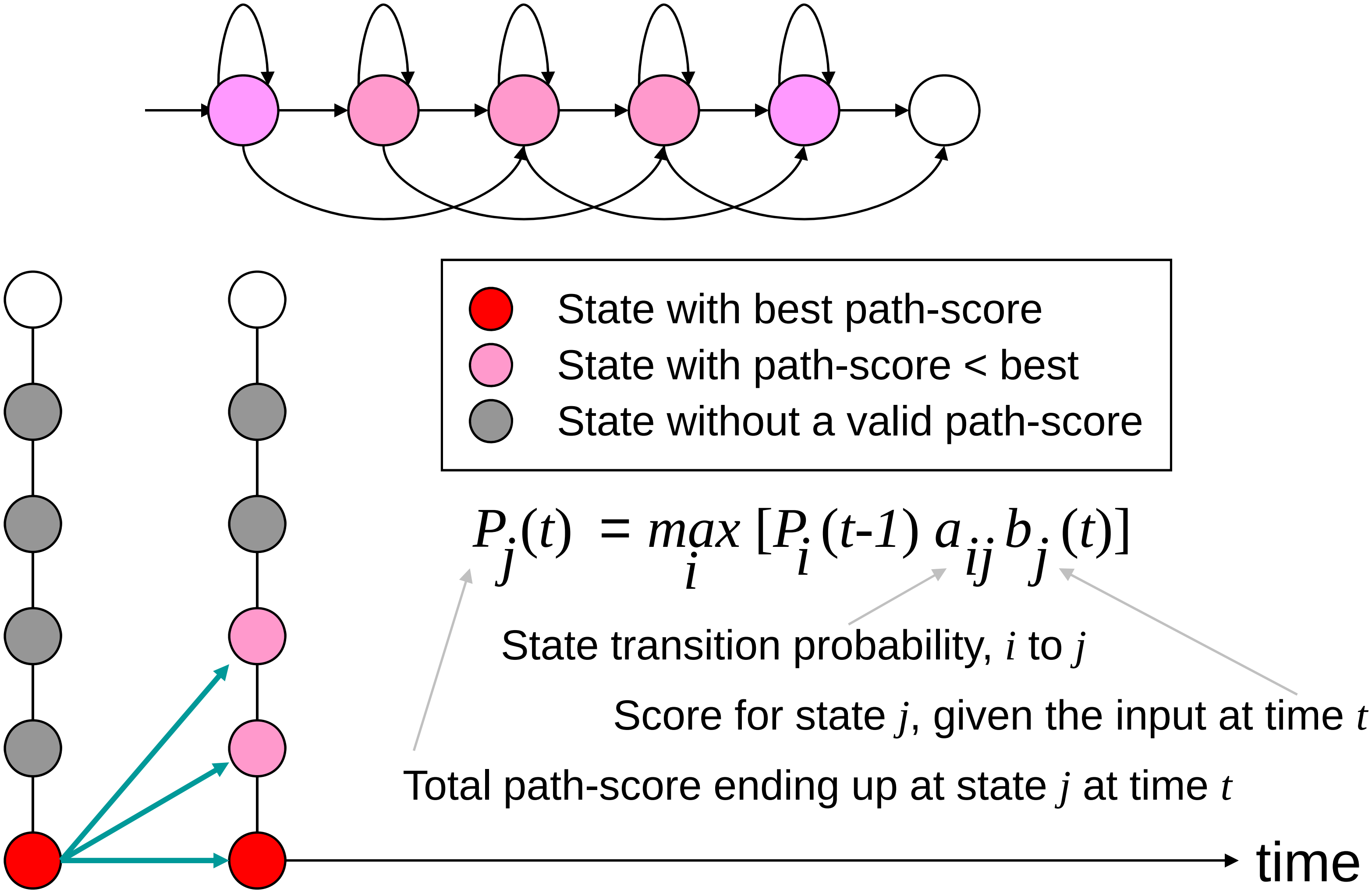




Initial state initialized with path-score = $P(s_1)b_1(1)$

All other states have score 0 since $P(s_i) = 0$ for them

Viterbi Search (contd.)



- 🟥 State with best path-score
- ● State with path-score < best
- ● State without a valid path-score

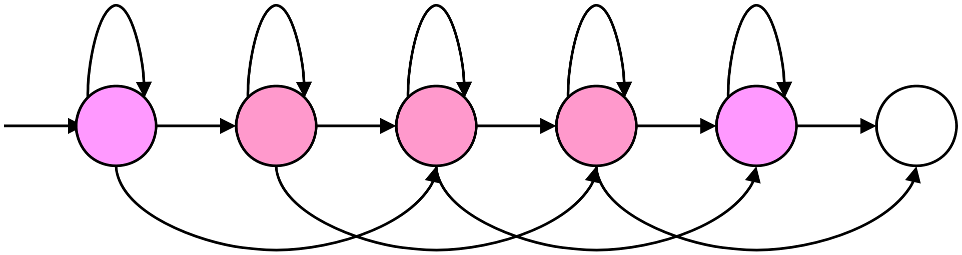
$$P_j(t) = \max_i \left[P_i(t\text{-}1)\, a_{ij}\, b_j(t)\right]$$

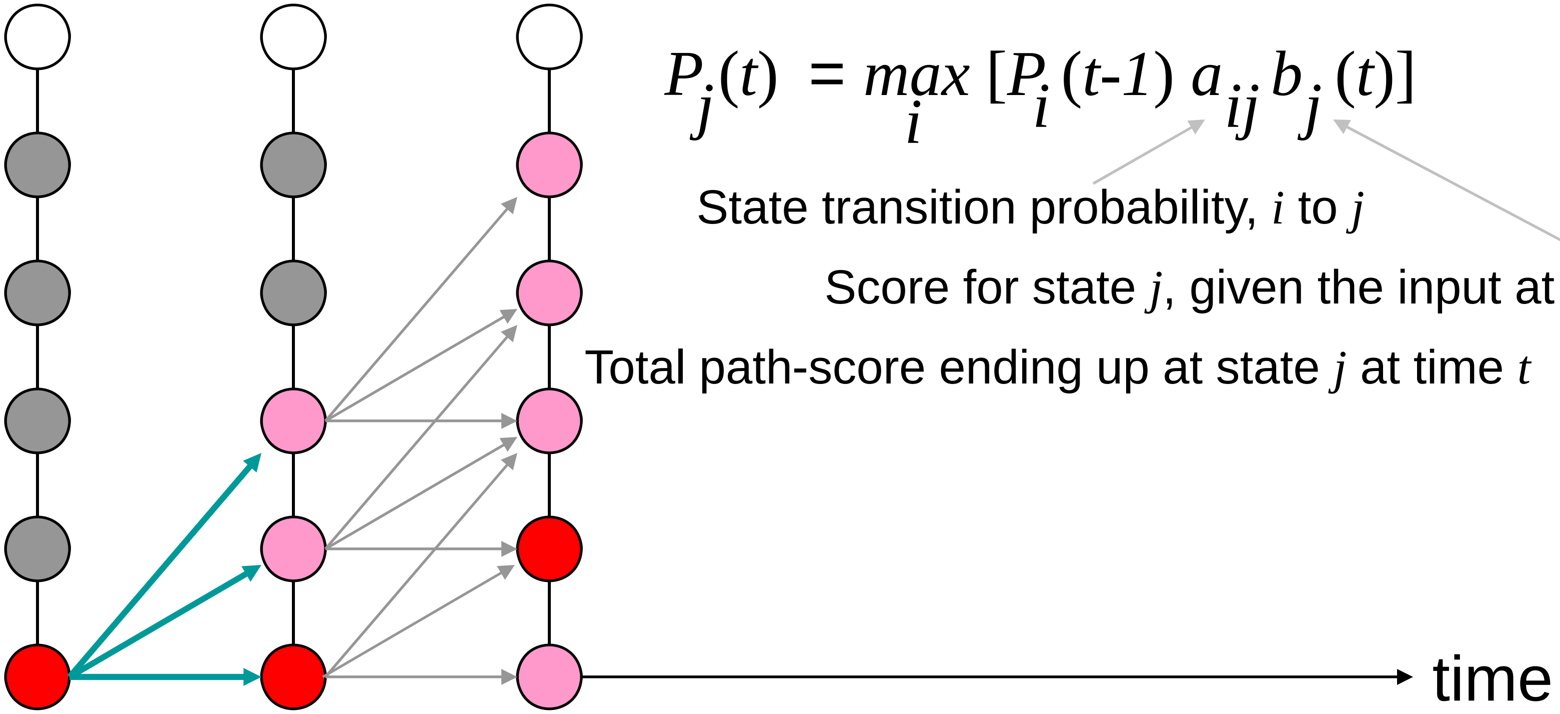
State transition probability, *i* to *j*

Score for state *j*, given the input at time *t*

Total path-score ending up at state *j* at time *t*

Viterbi Search (contd.)





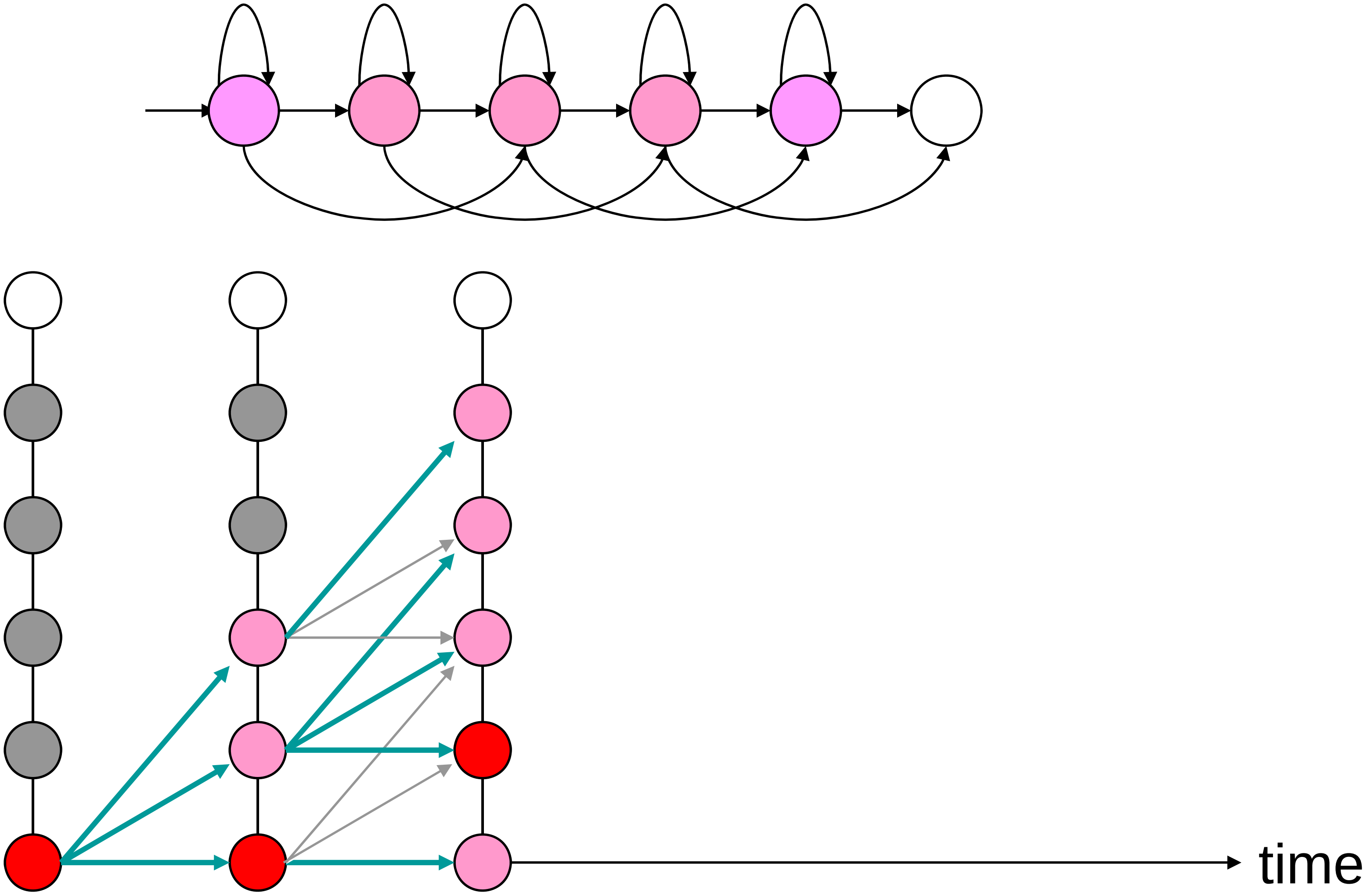
$$P_j(t) = \max_i \,[P_i(t\text{-}1)\; a_{ij}\; b_j(t)]$$

State transition probability, *i* to *j*

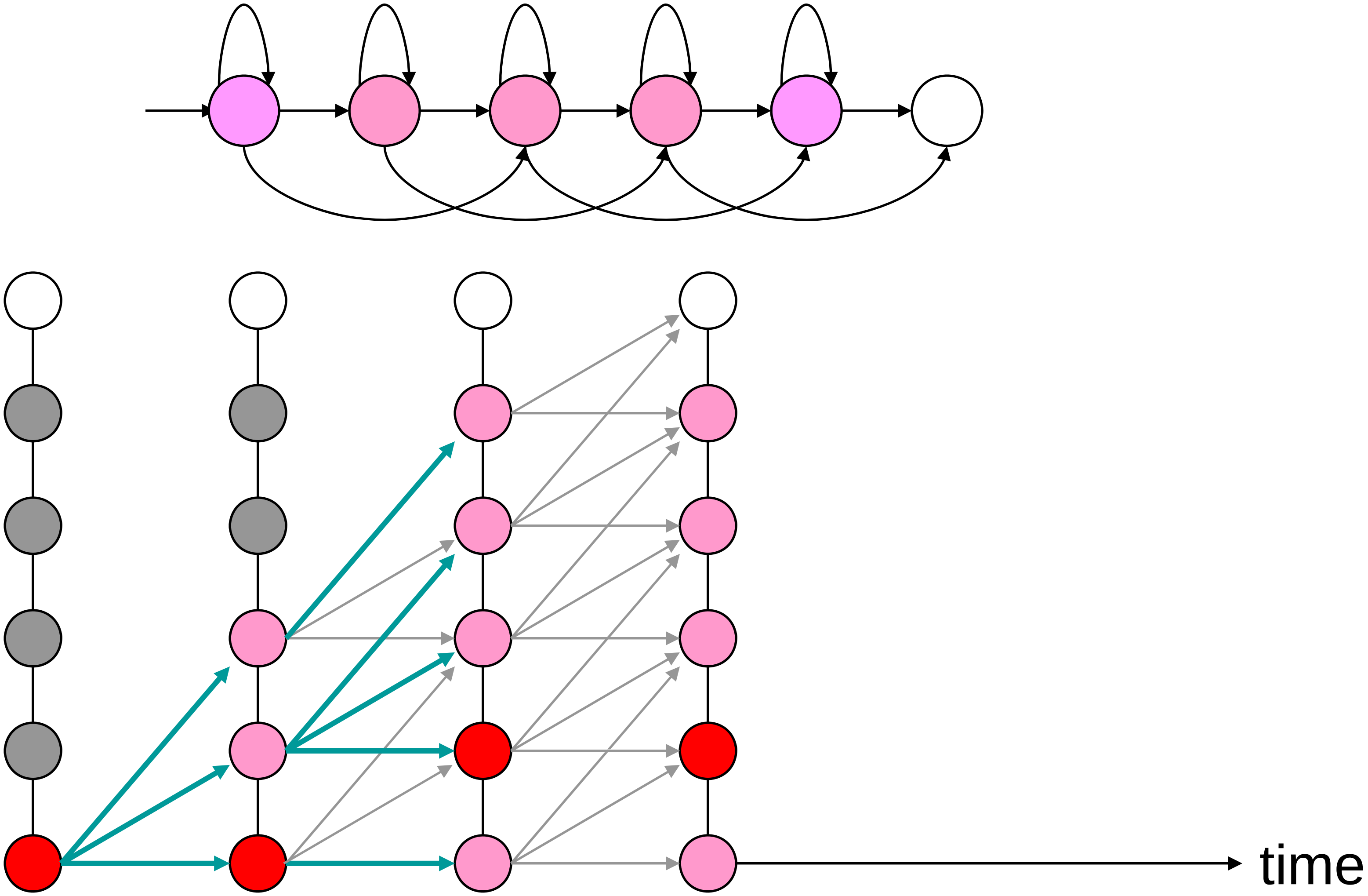
Score for state *j*, given the input at time *t*

Total path-score ending up at state *j* at time *t*

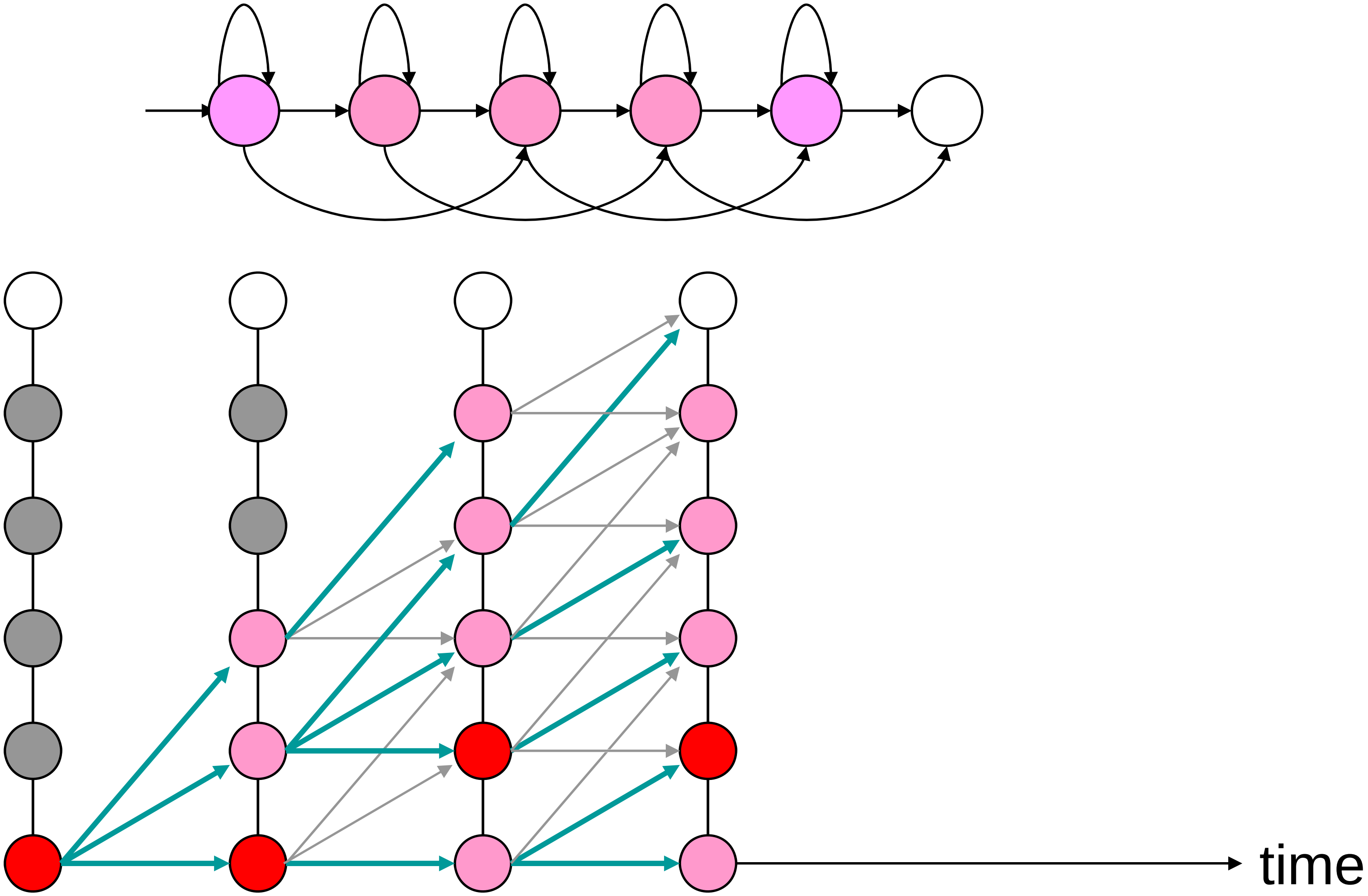
Viterbi Search (contd.)



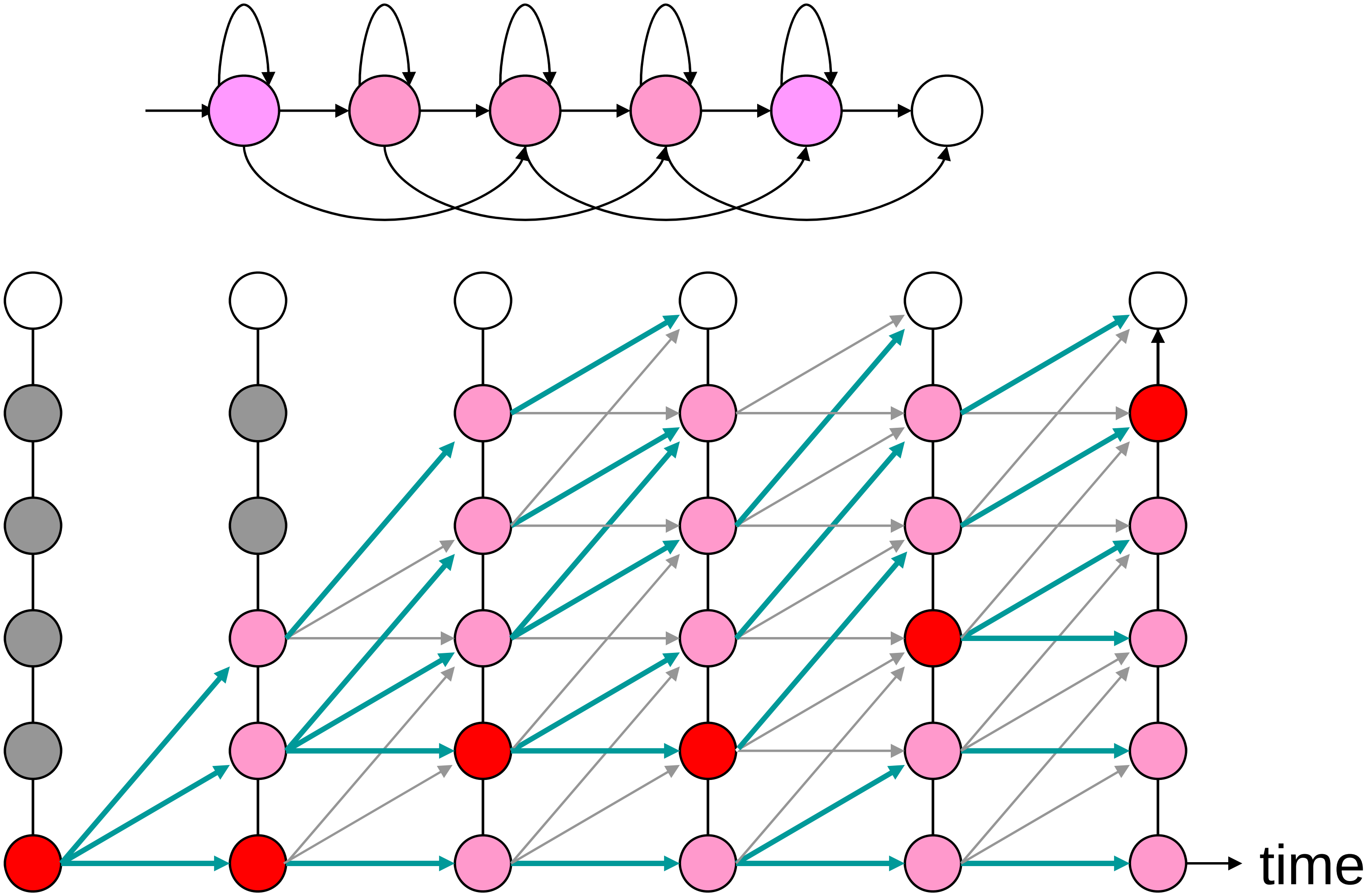
Viterbi Search (contd.)



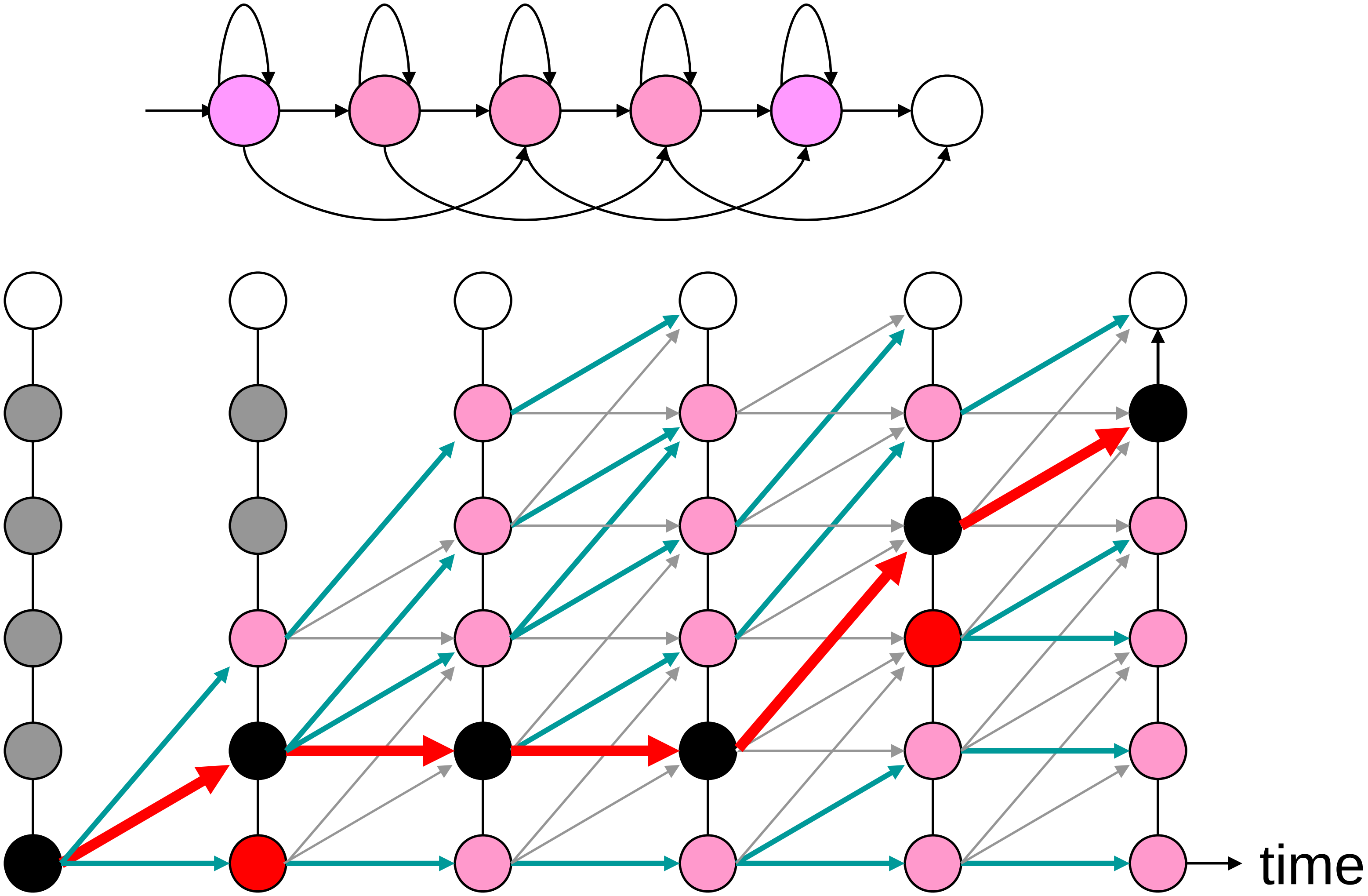
Viterbi Search (contd.)



Viterbi Search (contd.)

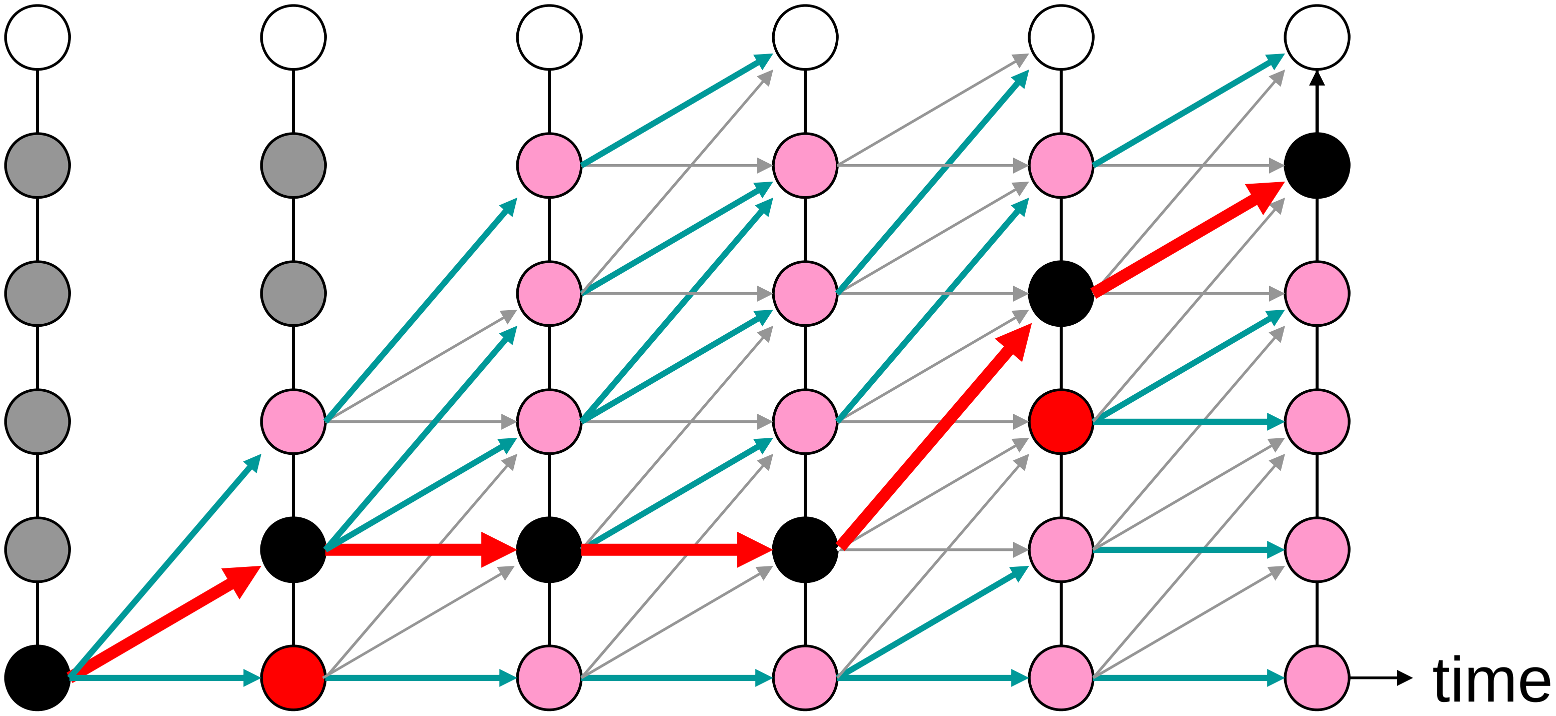


Viterbi Search (contd.)



Viterbi Search (contd.)

THE BEST STATE SEQUENCE IS THE ESTIMATE OF THE STATE SEQUENCE FOLLOWED IN GENERATING THE OBSERVATION



Viterbi and DTW

- The Viterbi algorithm is identical to the string-matching procedure used for DTW that we saw earlier

- It computes an estimate of the state sequence followed in producing the observation

- *It also gives us the probability of the best state sequence*

Problem3: Training HMM parameters

- We can compute the probability of an observation, and the best state sequence given an observation, using the HMM’s parameters

- But where do the HMM parameters come from?

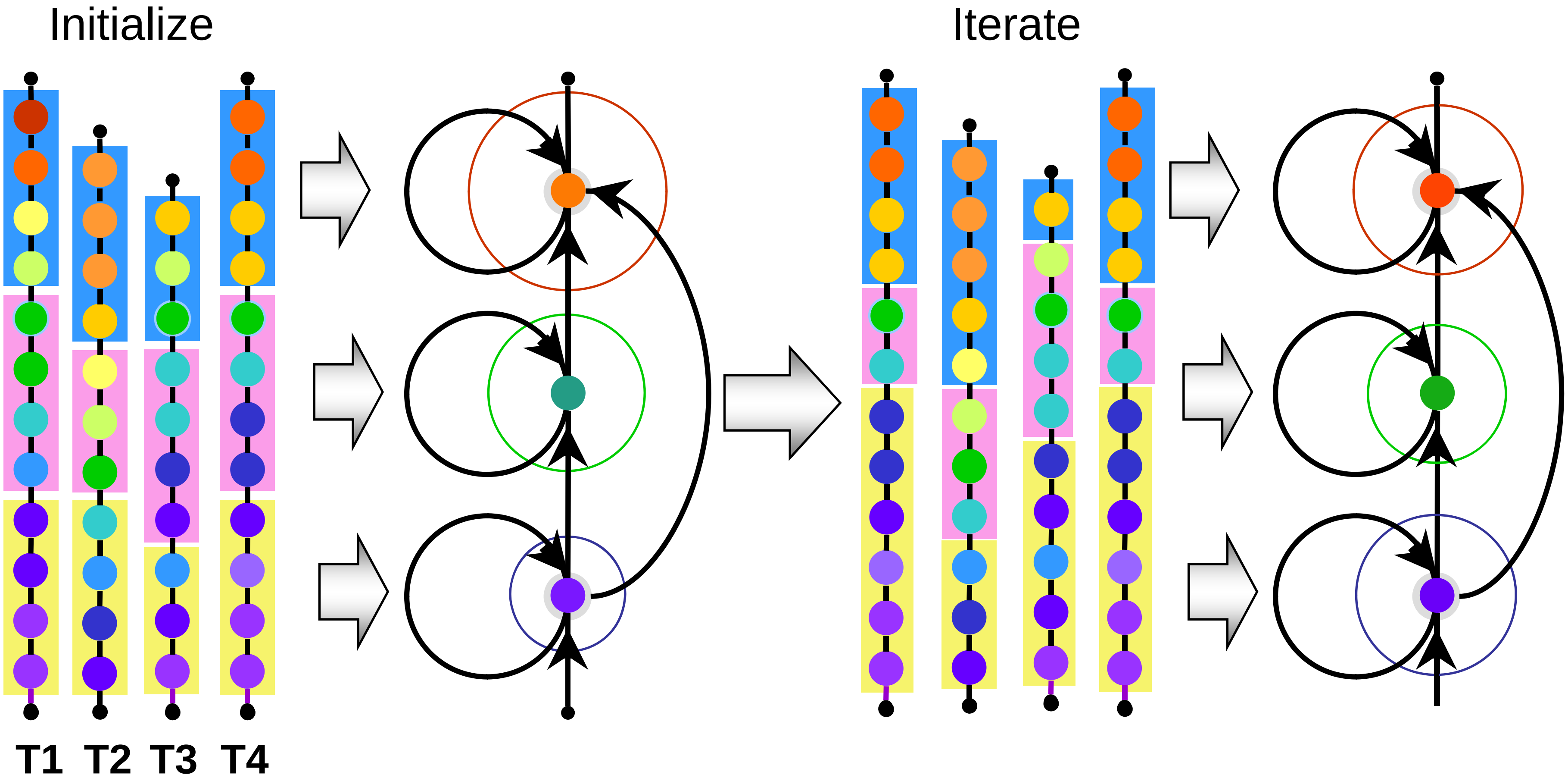
- They must be learned from a collection of observation sequences

- We have already seen one technique for training HMMs: The segmental K-means procedure

Modified segmental K-means AKA Viterbi training

- The entire segmental K-means algorithm:
  1. Initialize all parameters
     - State means and covariances
     - Transition probabilities
     - Initial state probabilities
  2. Segment all training sequences
  3. Reestimate parameters from segmented training sequences
  4. If not converged, return to 2

Segmental K-means



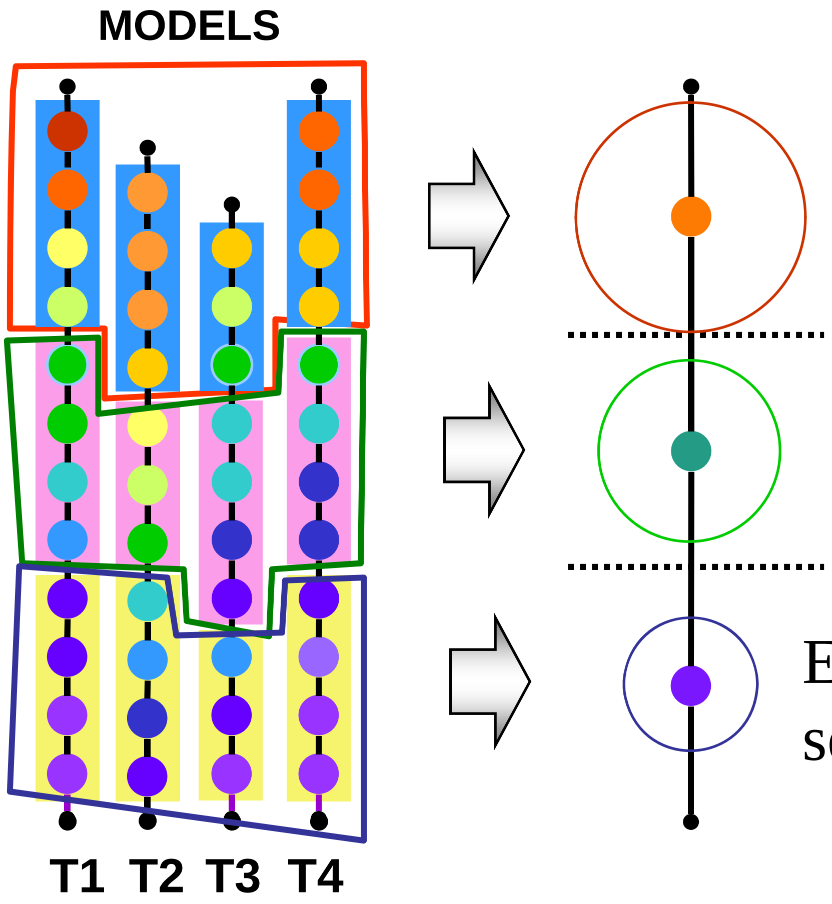
The procedure can be continued until convergence

Convergence is achieved when the total best-alignment error for all training sequences does not change significantly with further refinement of the model

A Better Technique

- The Segmental K-means technique uniquely assigns each observation to one state
- However, this is only an estimate and may be wrong
- A better approach is to take a “soft” decision
  - Assign each observation to *every* state with a probability

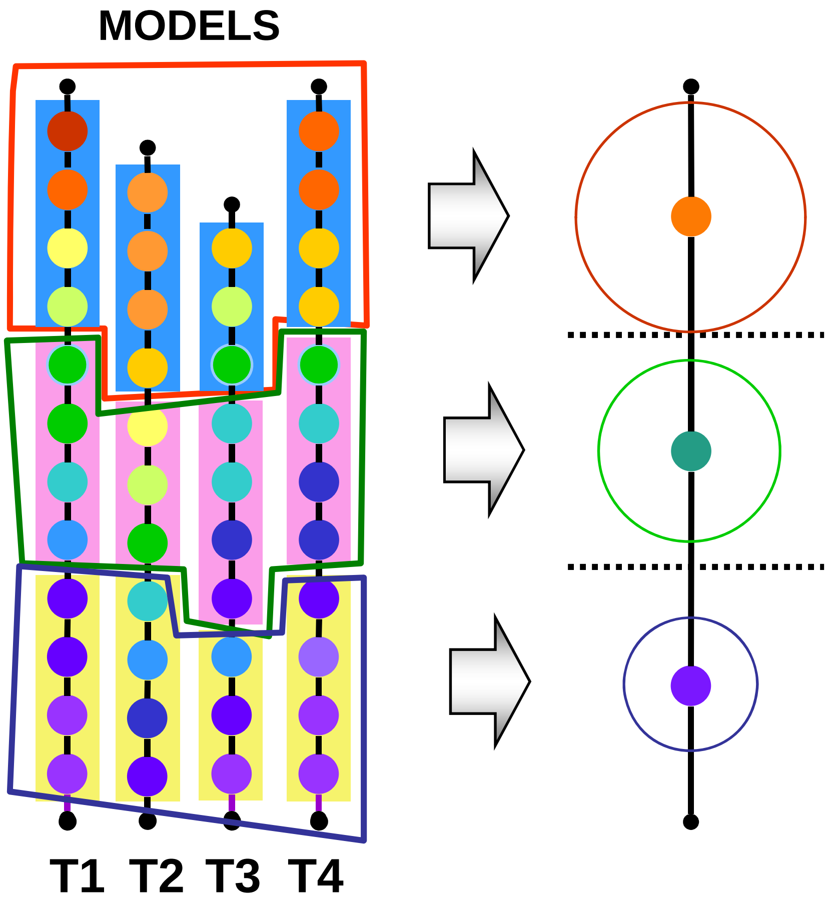
Training by segmentation: Hard Assignment



$$\mu_j = \frac{1}{\sum_{i \in segment(j)} 1} \sum_{i \in segment(j)} x_i$$

Each vector belongs uniquely to a segment

Training by segmentation: Soft Assignment



$$\mu_j = \frac{1}{\sum_{i \in All\ vectors} f_{i,j}} \sum_{i \in All\ vectors} f_{i,j} x_i$$

$$\sum_{j \in all\ segments} f_{i,j} = 1$$

Assignment is *fractioned:*
Every segment gets a piece of every vector

Means and variances are computed from fractioned vectors

Where do the fractions come from?

The “probability” of a state

- The probability assigned to any state *s*, for any observation x_t is the probability that the process was at *s* when it generated x_t

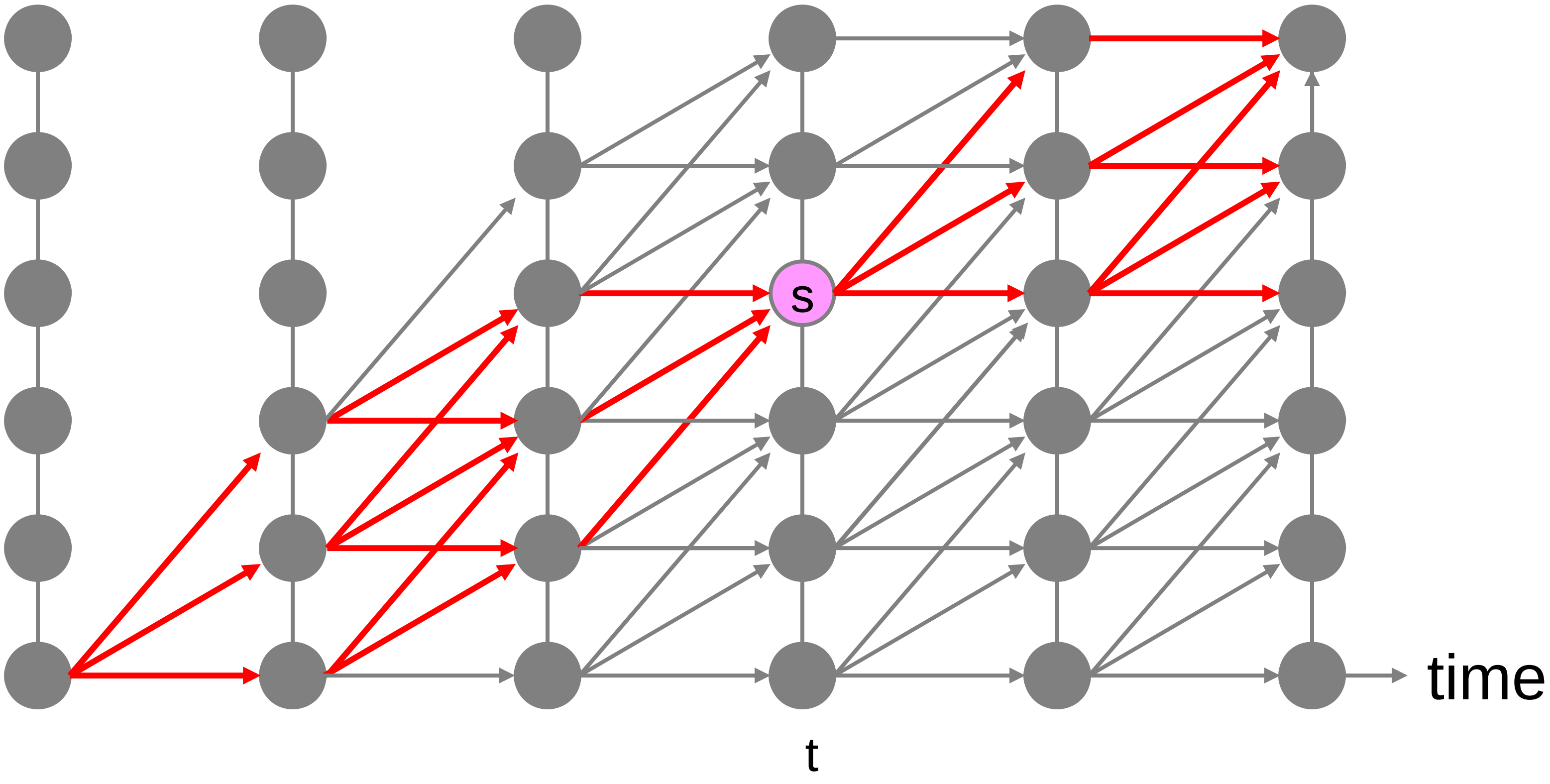
- We want to compute

$$P(state(t) = s \mid x_1, x_2, \ldots, x_T) \propto P(state(t) = s, x_1, x_2, \ldots, x_T)$$

- We will compute $P(state(t) = s, x_1, x_2, \ldots, x_T)$ first
  - This is the probability that the process visited *s* at time *t* while producing the entire observation

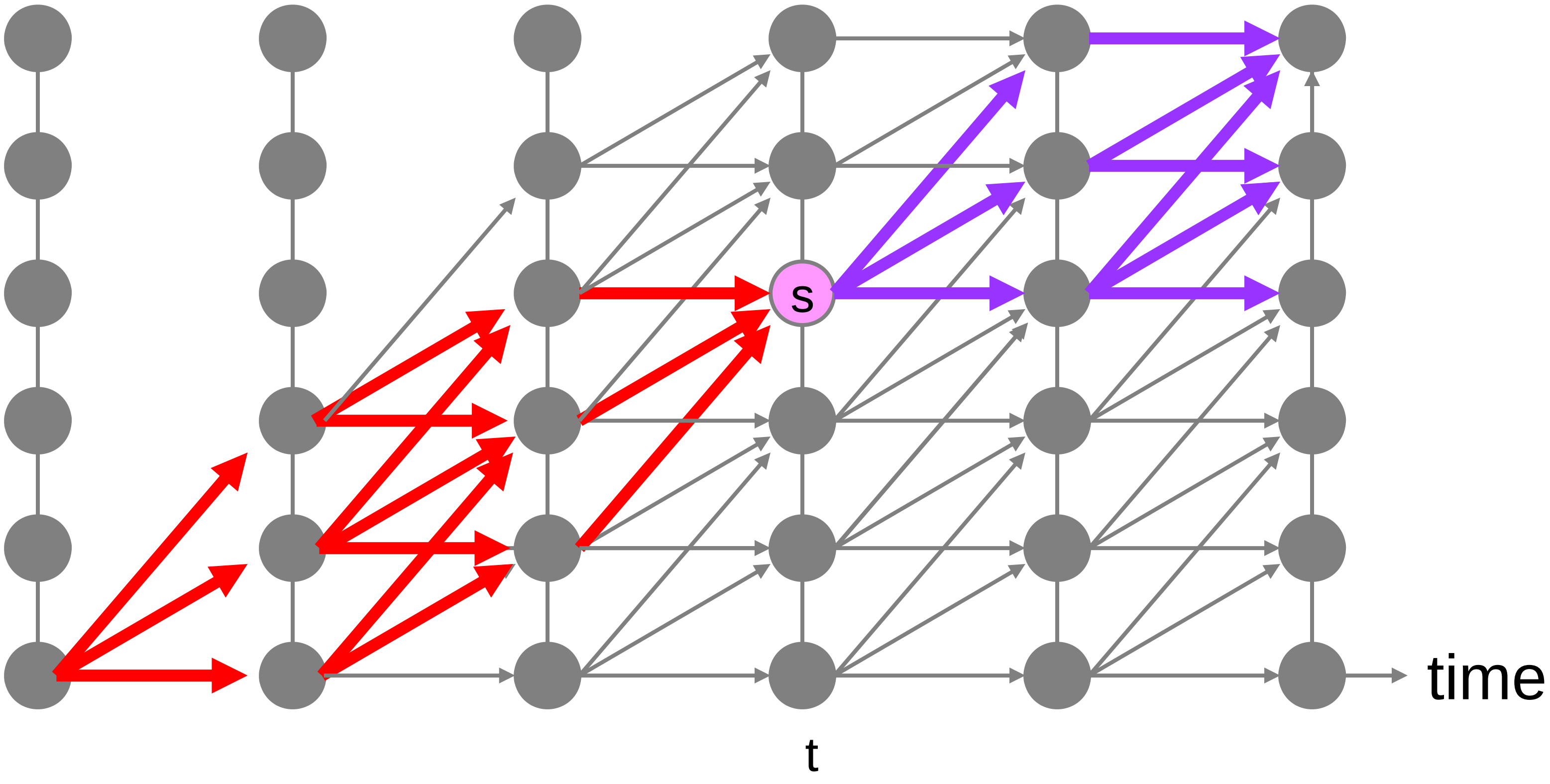
Probability of Assigning an Observation to a State

- The probability that the HMM was in a particular state *s* when generating the observation sequence is the probability that it followed a state sequence that passed through *s* at time *t*



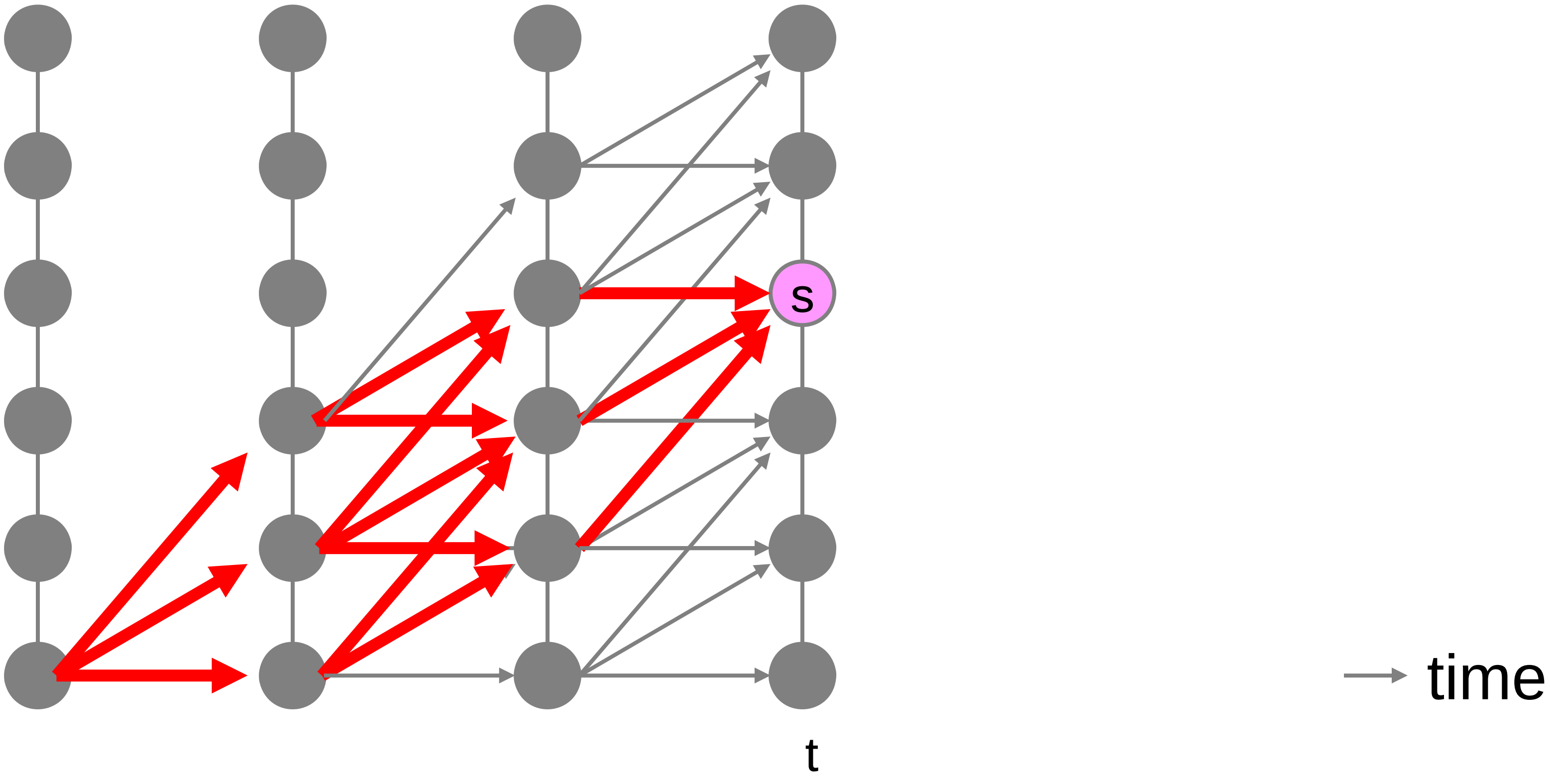
Probability of Assigning an Observation to a State

- This can be decomposed into two multiplicative sections
  - The section of the lattice leading into state *s* at time t and the section leading out of it



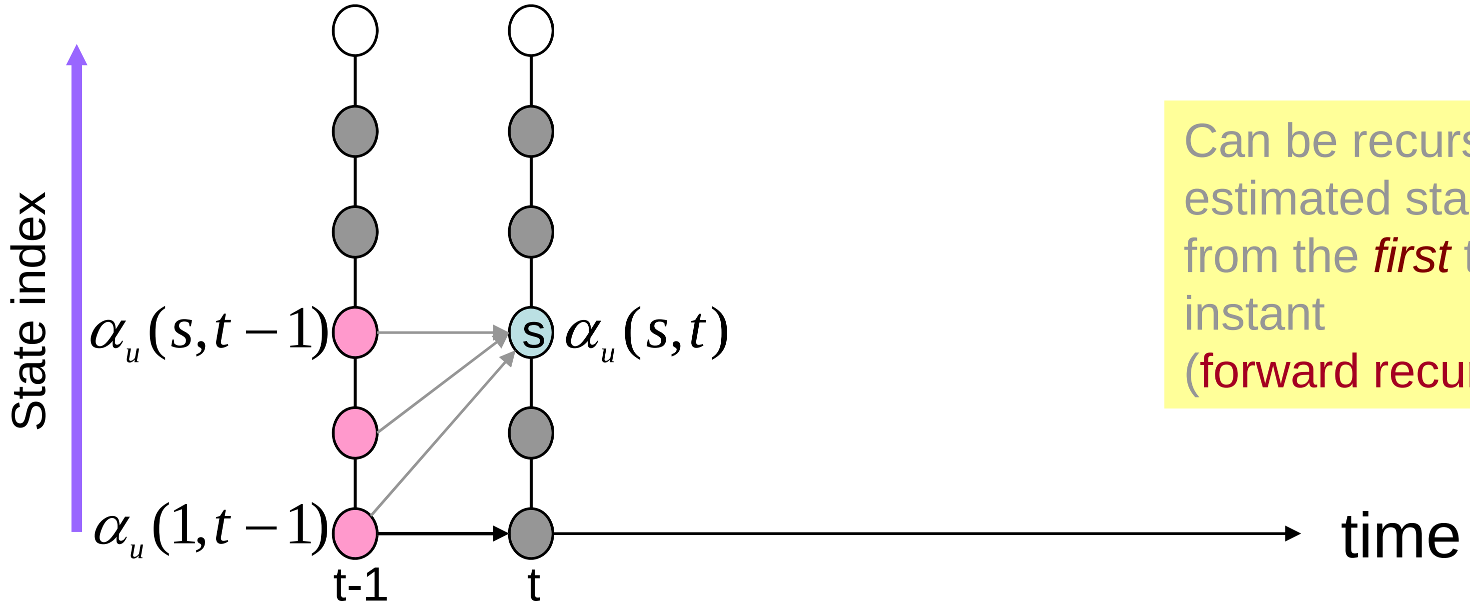
Probability of Assigning an Observation to a State

- The probability of the red section is the total probability of all state sequences ending at state *s* at time *t*
  - This is simply $\alpha(s,t)$
  - Can be computed using the forward algorithm



The forward algorithm

$$\alpha_u(s,t) = P(x_{u,1}, x_{u,2}, \ldots, x_{u,t}, state(t) = s|\lambda)$$



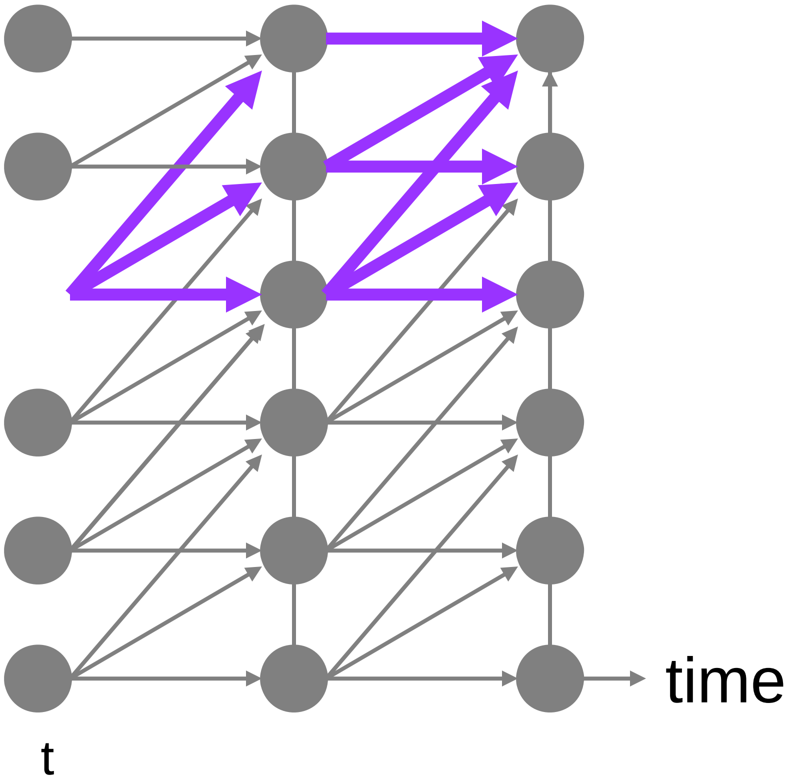
Can be recursively estimated starting from the *first* time instant (forward recursion)

$$\alpha_u(s,t) = \sum_{s'} \alpha_u(s',t-1) P(s|s') P(x_{u,t}|s)$$

λ represents the complete current set of HMM parameters

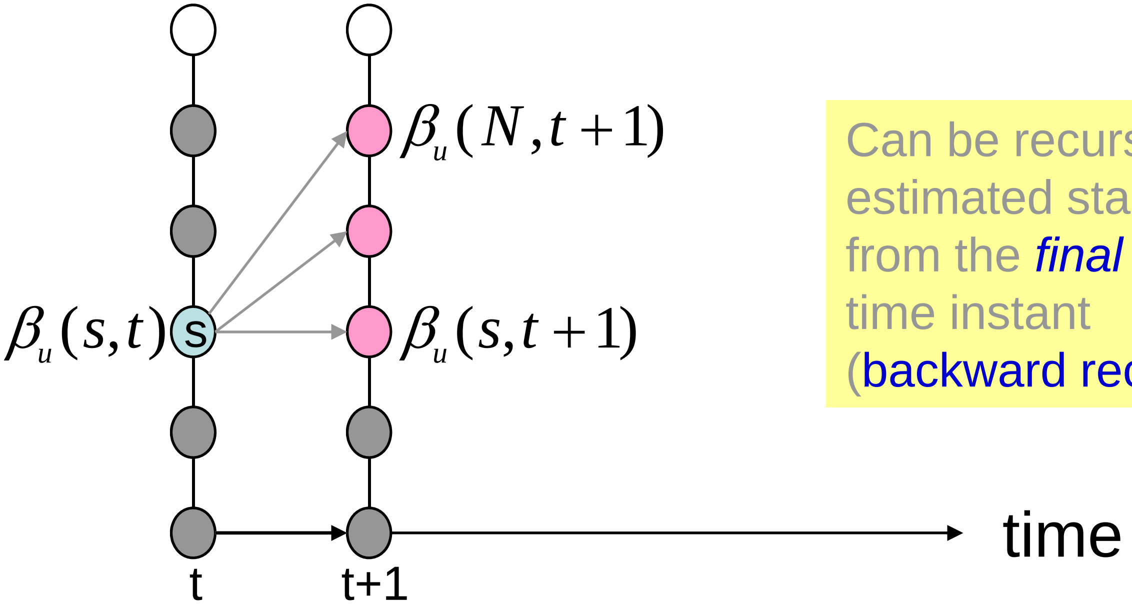
The Future Paths

- The blue portion represents the probability of all state sequences that began at state *s* at time *t*
  - Like the red portion it can be computed using a *backward recursion*



The Backward Recursion

$$\beta_u(s,t) = P(x_{u,t+1}, x_{u,t+2}, \ldots, x_{u,T} | state(t) = s, \lambda)$$



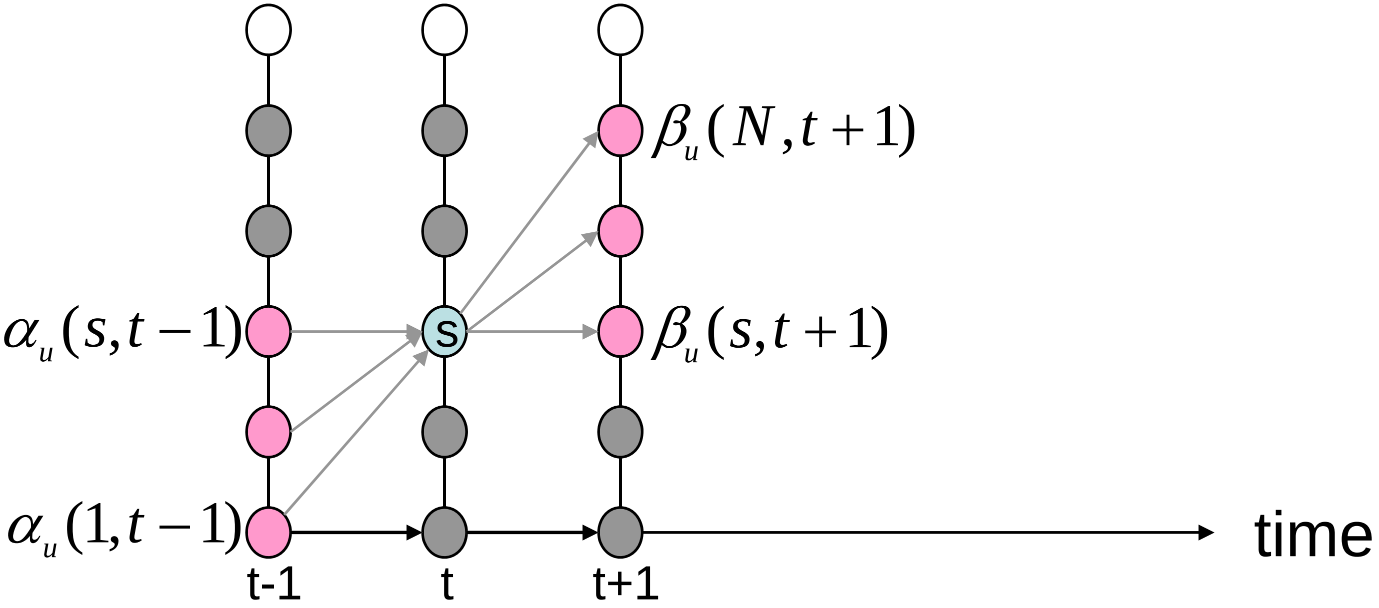
Can be recursively estimated starting from the *final* time time instant (backward recursion)

$$\beta_u(s,t) = \sum_{s'} \beta_u(s',t+1) P(s'|s) P(x_{u,t+1}|s')$$

- $\beta_u(s,t)$ is the total probability of ALL state sequences that depart from *s* at time *t*, and all observations after x_t
  - $\beta(s,T) = 1$ at the final time instant for all valid final states

The complete probability

$$\alpha_u(s,t)\beta_u(s,t) = P(x_{u,1}, x_{u,2}, \ldots, x_{u,T}, state(t) = s|\lambda)$$



$$= P(\mathbf{X}_u, state(t) = s|\lambda)$$

Posterior probability of a state

- The probability that the process was in state *s* at time *t*, given that we have observed the data is obtained by simple normalization

$$P(state(t)=s|\mathbf{X}_u,\lambda)=\frac{P(\mathbf{X}_u,state(t)=s|\lambda)}{\sum_{s'}P(\mathbf{X}_u,state(t)=s'|\lambda)}=\frac{\alpha_u(s,t)\beta_u(s,t)}{\sum_{s'}\alpha_u(s',t)\beta_u(s',t)}$$

- This term is often referred to as the gamma term and denoted by $\gamma_{s,t}$

Update Rules

- Once we have the state probabilities (the gammas) the update rules are obtained through a simple modification of the formulae used for segmental K-means
  - This new learning algorithm is known as the Baum-Welch learning procedure
- Case1: State output densities are Gaussians

Update Rules

$$\mu_s = \frac{1}{N_s}\sum_{x \in s} x$$

$$C_s = \frac{1}{N_s}\sum_{x \in s}(x-\mu_s)^T(x-\mu_s)$$

$$\mu_s = \frac{\sum_u \sum_t \gamma_{s,u,t} x_{u,t}}{\sum_u \sum_t \gamma_{s,u,t}}$$

$$C_s = \frac{\sum_u \sum_t \gamma_{s,u,t}(x-\mu_s)^T(x-\mu_s)}{\sum_u \sum_t \gamma_{s,u,t}}$$

Segmental K-means

Baum Welch

- A similar update formula reestimates transition probabilities
- The *initial* state probabilities P(*s*) also have a similar update rule

Case 2: State ouput densities are Gaussian Mixtures

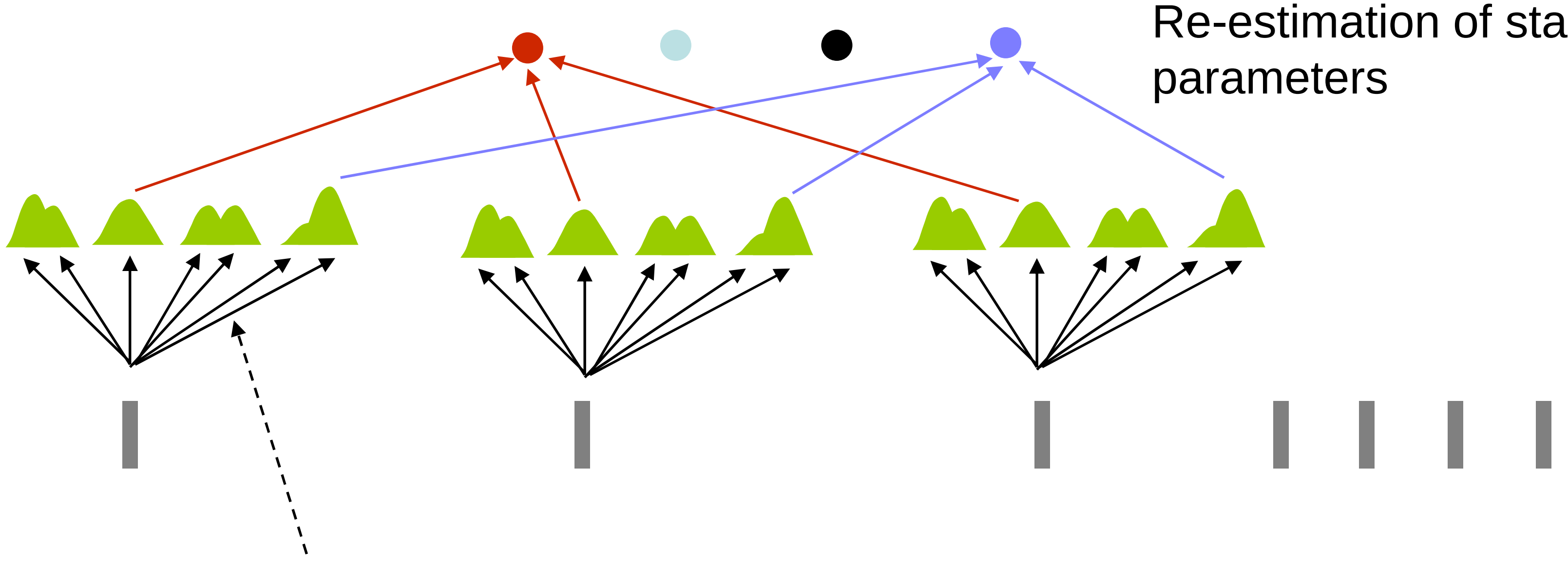
- When state output densities are Gaussian *mixtures*, more parameters must be estimated

$$P(x \mid s) = \sum_{i=0}^{K-1} w_{s,i} Gaussian(x; \mu_{s,i}, C_{s,i})$$

- The mixture weights $w_{s,i}$, mean $\mu_{s,i}$ and covariance $C_{s,i}$ of every Gaussian in the distribution of each state must be estimated

Splitting the Gamma

We split the gamma for any state among all the Gaussians at that state



Re-estimation of state parameters

A posteriori probability that the t^{th} vector was generated by the k^{th} Gaussian of state s

$$\gamma_{k,s,u,t} = P(state(t) = s | \mathbf{X}_u, \lambda) P(k^{th}.Gaussian | state(t) = s, x_{u,t}, \lambda)$$

Splitting the Gamma among Gaussians

A posteriori probability that the tth vector was generated by the kth Gaussian of state s

$$\gamma_{k,s,u,t} = P(state(t) = s | \mathbf{X}_u, \lambda) P(k^{th}.Gaussian | state(t) = s, x_{u,t}, \lambda)$$

$$\gamma_{k,s,u,t} = P(state(t) = s | \mathbf{X}_u, \lambda) \frac{w_{k,s} \frac{1}{\sqrt{(2\pi)^d |\mathbf{C}_k|}} e^{-\frac{1}{2}(x_{u,t} - \mu_{k,s})^T \mathbf{C}_k^{-1} (x_{u,t} - \mu_{k,s})}}{\sum_{k'} w_{k',s} \frac{1}{\sqrt{(2\pi)^d |\mathbf{C}_{k',s}|}} e^{-\frac{1}{2}(x_{u,t} - \mu_{k',s})^T \mathbf{C}_k^{-1} (x_{u,t} - \mu_{k',s})}}$$

Updating HMM Parameters

$$\tilde{\mu}_{k,s} = \frac{\sum_u \sum_t \gamma_{k,s,u,t} x_{u,t}}{\sum_u \sum_t \gamma_{k,s,u,t}}$$

$$\tilde{\mathbf{C}}_{k,s} = \frac{\sum_u \sum_t \gamma_{k,s,u,t} \left(x_{u,t} - \tilde{\mu}_{k,s}\right)\left(x_{u,t} - \tilde{\mu}_{k,s}\right)^T}{\sum_u \sum_t \gamma_{k,s,u,t}}$$

$$\tilde{w}_{k,s} = \frac{\sum_u \sum_t \gamma_{k,s,u,t}}{\sum_u \sum_t \sum_j \gamma_{j,s,u,t}}$$

- Note: Every observation contributes to the update of parameter values of every Gaussian of every state

Overall Training Procedure: Single Gaussian PDF

- Determine a topology for the HMM
- Initialize all HMM parameters
  - Initialize all allowed transitions to have the same probability
  - Initialize all state output densities to be Gaussians
    - We'll revisit initialization

1. Over all utterances, compute the "sufficient" statistics $\sum_u \sum_t \gamma_{s,u,t}$ $\sum_u \sum_t \gamma_{s,u,t} x_{u,t}$ $\sum_u \sum_t \gamma_{s,u,t} (x - \mu_s)^T (x - \mu_s)$
2. Use update formulae to compute new HMM parameters
3. If the overall probability of the training data has not converged, return to step 1

An Implementational Detail

- Step1 computes "buffers" over all utterance

$$\sum_{u}\sum_{t}\gamma_{s,u,t} = \sum_{u\in U_1}\sum_{t}\gamma_{s,u,t} + \sum_{u\in U_2}\sum_{t}\gamma_{s,u,t} + ..$$

$$\sum_{u}\sum_{t}\gamma_{s,u,t}x_{u,t} = \sum_{u\in U_1}\sum_{t}\gamma_{s,u,t}x_{u,t} + \sum_{u\in U_2}\sum_{t}\gamma_{s,u,t}x_{u,t} + ...$$

$$\sum_{u}\sum_{t}\gamma_{s,u,t}(x-\mu_s)^T(x-\mu_s) = \sum_{u\in U_1}\sum_{t}\gamma_{s,u,t}(x-\mu_s)^T(x-\mu_s) + \sum_{u\in U_2}\sum_{t}\gamma_{s,u,t}(x-\mu_s)^T(x-\mu_s) + ..$$

- This can be split and parallelized
  - U_1, U_2 etc. can be processed on separate machines

Machine 1

$$\sum_{u\in U_1}\sum_{t}\gamma_{s,u,t} \qquad \sum_{u\in U_1}\sum_{t}\gamma_{s,u,t}x_{u,t}$$

$$\sum_{u\in U_1}\sum_{t}\gamma_{s,u,t}(x-\mu_s)^T(x-\mu_s)$$

Machine 2

$$\sum_{u\in U_2}\sum_{t}\gamma_{s,u,t} \qquad \sum_{u\in U_2}\sum_{t}\gamma_{s,u,t}x_{u,t}$$

$$\sum_{u\in U_2}\sum_{t}\gamma_{s,u,t}(x-\mu_s)^T(x-\mu_s)$$

An Implementational Detail

- Step2 *aggregates and adds* buffers before updating the models

$$\sum_u \sum_t \gamma_{s,u,t} = \sum_{u \in U_1} \sum_t \gamma_{s,u,t} + \sum_{u \in U_2} \sum_t \gamma_{s,u,t} + ..$$

$$\sum_u \sum_t \gamma_{s,u,t} x_{u,t} = \sum_{u \in U_1} \sum_t \gamma_{s,u,t} x_{u,t} + \sum_{u \in U_2} \sum_t \gamma_{s,u,t} x_{u,t} + \ldots$$

$$\sum_u \sum_t \gamma_{s,u,t} (x-\mu_s)^T (x-\mu_s) = \sum_{u \in U_1} \sum_t \gamma_{s,u,t} (x-\mu_s)^T (x-\mu_s) + \sum_{u \in U_2} \sum_t \gamma_{s,u,t} (x-\mu_s)^T (x-\mu_s) + ..$$

$$\tilde{\mu}_{k,s} = \frac{\sum_u \sum_t \gamma_{k,s,u,t} x_{u,t}}{\sum_u \sum_t \gamma_{k,s,u,t}}$$

$$\tilde{\mathbf{C}}_{k,s} = \frac{\sum_u \sum_t \gamma_{k,s,u,t} \left(x_{u,t} - \tilde{\mu}_{k,s}\right)\left(x_{u,t} - \tilde{\mu}_{k,s}\right)^T}{\sum_u \sum_t \gamma_{k,s,u,t}}$$

$$\tilde{w}_{k,s} = \frac{\sum_u \sum_t \gamma_{k,s,u,t}}{\sum_u \sum_t \sum_j \gamma_{j,s,u,t}}$$

An Implementational Detail

- Step2 *aggregates and adds* buffers before updating the models

$$\sum_u\sum_t \gamma_{s,u,t} = \sum_{u\in U_1}\sum_t \gamma_{s,u,t} + \sum_{u\in U_2}\sum_t \gamma_{s,u,t} + \ldots$$

$$\sum_u\sum_t \gamma_{s,u,t}x_{u,t} = \sum_{u\in U_1}\sum_t \gamma_{s,u,t}x_{u,t} + \sum_{u\in U_2}\sum_t \gamma_{s,u,t}x_{u,t} + \ldots$$

$$\sum_u\sum_t \gamma_{s,u,t}(x-\mu_s)^T(x-\mu_s) = \sum_{u\in U_1}\sum_t \gamma_{s,u,t}(x-\mu_s)^T(x-\mu_s) + \sum_{u\in U_2}\sum_t \gamma_{s,u,t}(x-\mu_s)^T(x-\mu_s) + \ldots$$

$$\tilde{\mu}_{k,s} = \frac{\sum_u\sum_t \gamma_{k,s,u,t}x_{u,t}}{\sum_u\sum_t \gamma_{k,s,u,t}}$$

$$\tilde{\mathbf{C}}_{k,s} = \frac{\sum_u\sum_t \gamma_{k,s,u,t}\left(x_{u,t}-\tilde{\mu}_{k,s}\right)\left(x_{u,t}-\tilde{\mu}_{k,s}\right)^T}{\sum_u\sum_t \gamma_{k,s,u,t}}$$

$$\tilde{w}_{k,s} = \frac{\sum_u\sum_t \gamma_{k,s,u,t}}{\sum_u\sum_t\sum_j \gamma_{j,s,u,t}}$$

Computed by machine 1

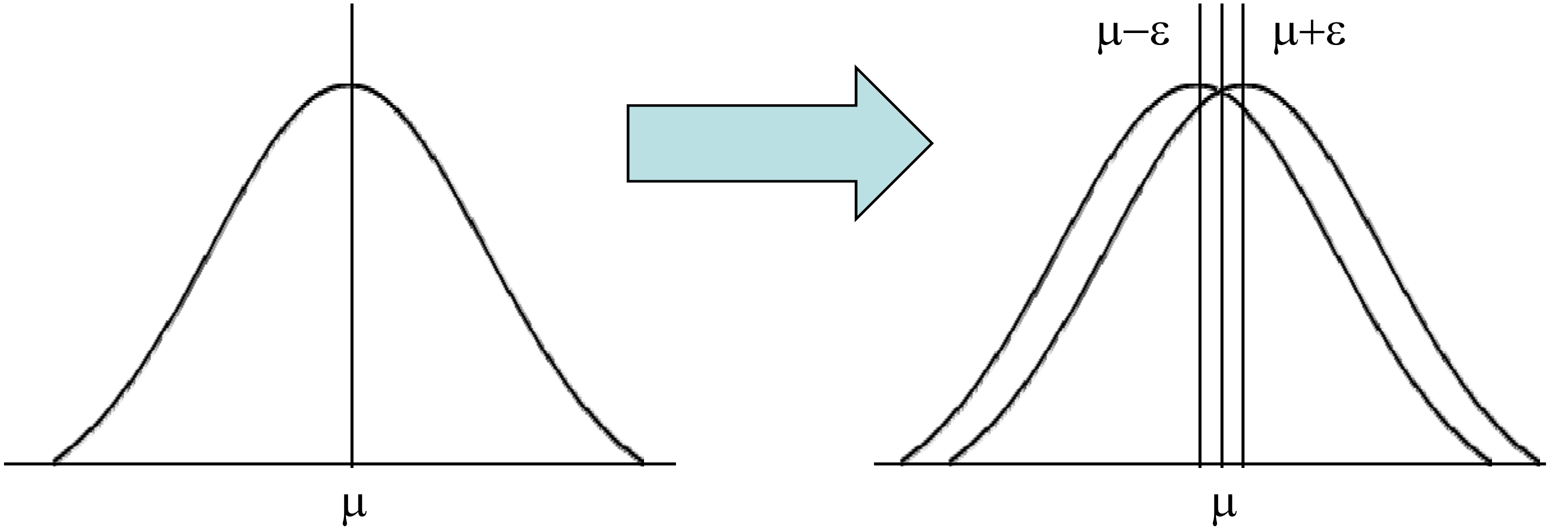
Computed by machine 2

Training for HMMs with *Gaussian Mixture* State Output Distributions

- Gaussian *Mixtures* are obtained by splitting

1. Train an HMM with (single) Gaussian state output distributions
2. Split the Gaussian with the largest variance
    - Perturb the mean by adding and subtracting a small number
    - This gives us 2 Gaussians. Partition the mixture weight of the Gaussian into two halves, one for each Gaussian
    - A mixture with N Gaussians now becomes a mixture of N+1 Gaussians
3. Iterate BW to convergence
4. If the desired number of Gaussians not obtained, return to 2

Splitting a Gaussian



- The mixture weight *w* for the Gaussian gets shared as 0.5*w* by each of the two split Gaussians

Implementation of BW: underflow

- Arithmetic underflow is a problem

$$\underset{\substack{\downarrow \\ \text{probability term}}}{\alpha_u(s,t)} = \underbrace{\sum_{s'} \alpha_u(s',t-1)P(s|s')P(x_{u,t}|s)}_{\text{probability terms}}$$

- The alpha terms are a recursive product of probability terms
  - As t increases, an increasingly greater number probability terms are factored into the alpha
- All probability terms are less than 1
  - State output probabilities are actually probability densities
  - Probability density values *can* be greater than 1
  - On the other hand, for large dimensional data, probability density values are usually *much* less than 1
- With increasing time, alpha values decrease
- Within a few time instants, they underflow to 0
  - Every alpha goes to 0 at some time t. All future alphas remain 0
  - As the dimensionality of the data increases, alphas goes to 0 faster

Underflow: Solution

- One method of avoiding underflow is to scale all alphas at each time instant
  - Scale with respect to the largest alpha to make sure the largest scaled alpha is 1.0
  - Scale with respect to the sum of the alphas to ensure that all alphas sum to 1.0
  - Scaling constants must be appropriately considered when computing the final probabilities of an observation sequence

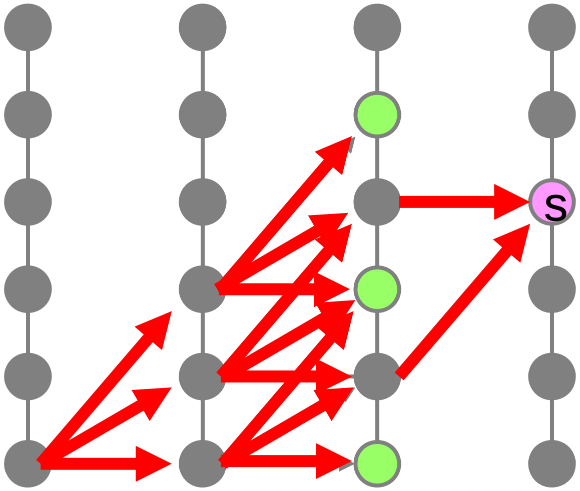
Implementation of BW: underflow

- Similarly, arithmetic underflow can occur during beta computation

$$\beta_u(s,t) = \sum_{s'} \beta_u(s',t+1) \log P(s'|s) P(x_{u,t+1}|s')$$

- The beta terms are also a recursive product of probability terms and can underflow
- Underflow can be prevented by
  - Scaling: Divide all beta terms by a constant that prevents underflow
  - By performing beta computation in the log domain

Implementation of BW: pruning



🟢 = pruned out

- The forward backward computation can get very expensive
- Solution: Prune
- Pruning in the forward backward algorithm raises some additional issues
    - Pruning from forward pass can be employed by backward pass
    - Convergence criteria and tests may be affected
    - More later

Building a recognizer for isolated words

- Now have all necessary components to build an HMM-based recognizer for isolated words
  - Where each word is spoken by itself in isolation
  - E.g. a simple application, where one may either say “Yes” or “No” to a recognizer and it must recognize what was said

Isolated Word Recognition with HMMs

- Assuming all words are equally likely
- Training
  - Collect a set of “training” recordings for each word
  - Compute feature vector sequences for the words
  - Train HMMs for each word

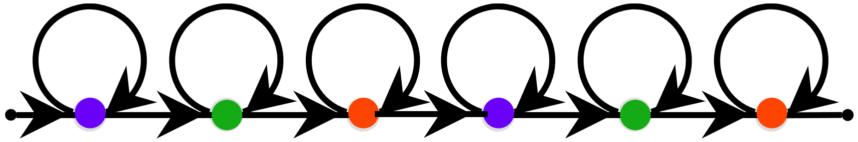
- Recognition:
  - Compute feature vector sequence for test utterance
  - Compute the forward probability of the feature vector sequence from the HMM for each word
    - Alternately compute the best state sequence probability using Viterbi
  - Select the word for which this value is highest

Issues

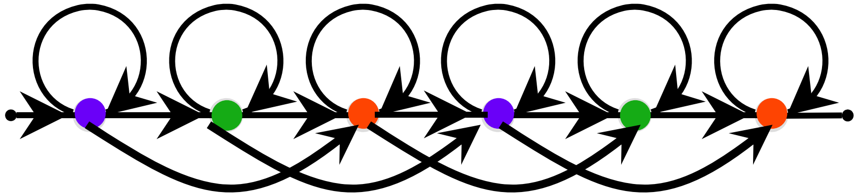
- What is the topology to use for the HMMs
  - How many states
  - What kind of transition structure
  - If state output densities have Gaussian Mixtures: how many Gaussians?

HMM Topology

- For speech a left-to-right topology works best
  - The “Bakis” topology
  - Note that the initial state probability P(s) is 1 for the 1st state and 0 for others. This need not be *learned*



- States may be skipped



24 Feb 2010

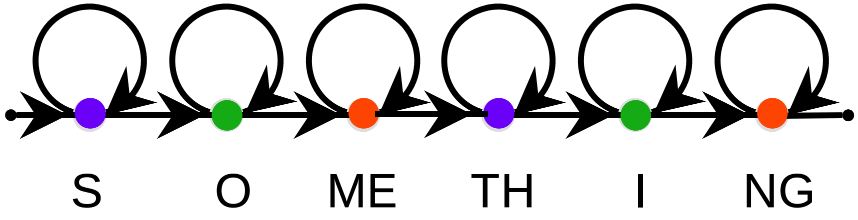
Determining the Number of States

- How do we know the number of states to use for any word?
  - We do not, really
  - Ideally there should be at least one state for each "basic sound" within the word
    - Otherwise widely differing sounds may be collapsed into one state
    - The average feature vector for that state would be a poor representation

- For computational efficiency, the number of states should be small
  - These two are conflicting requirements, usually solved by making some educated guesses

Determining the Number of States

- For small vocabularies, it is possible to examine each word in detail and arrive at reasonable numbers:



- For larger vocabularies, we may be forced to rely on some *ad hoc* principles
  - *E.g.* proportional to the number of letters in the word
    - Works better for some languages than others
    - Spanish and Indian languages are good examples where this works as almost every letter in a word produces a sound

How many Gaussians

- No clear answer for this either
- The number of Gaussians is usually a function of the amount of training data available
  - Often set by trial and error
  - A minimum of 4 Gaussians is usually required for reasonable recognition

Implementation of BW: initialization of alphas and betas

- Initialization for alpha: $\alpha_u(s,1)$ set to 0 for all states except the first state of the model. $\alpha_u(s,1)$ set to 1 for the first state
  - All observations *must* begin at the first state

- Initialization for beta: $\beta_u(s, T)$ set to 0 for all states except the terminating state. $\beta_u(s, t)$ set to 1 for this state
  - All observations *must* terminate at the final state

Initializing State Output Density Parameters

1. Initially only a single Gaussian per state assumed
   - Mixtures obtained by splitting Gaussians

2. For Bakis-topology HMMs, a good initialization is the "flat" initialization
   - Compute the *global* mean and variance of all feature vectors in all training instances of the word
   - Initialize *all Gaussians* (i.e all state output distributions) with this mean and variance
   - Their means and variances will converge to appropriate values automatically with iteration
   - Gaussian splitting to compute Gaussian mixtures takes care of the rest

Isolated word recognition: Final thoughts

- All relevant topics covered
  - How to compute features from recordings of the words
    - We will not explicitly refer to feature computation in future lectures
  - How to set HMM topologies for the words
  - How to train HMMs for the words
    - Baum-Welch algorithm
  - How to select the most probable HMM for a test instance
    - Computing probabilities using the forward algorithm
    - Computing probabilities using the Viterbi algorithm
      - Which also gives the state segmentation

Questions

- ?