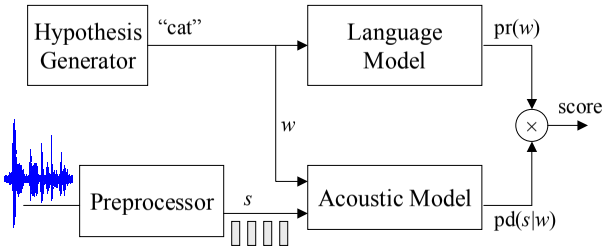
Lecture 15

Isolated Word Recognition

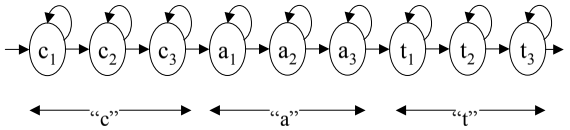
- Hidden Markov Model of speech
  - State transitions and alignment probabilities
- Searching all possible alignments
  - Dynamic Programming
  - Viterbi Alignment
- Isolated Word Recognition

Isolated Word Recogniser



- Preprocessor
  - Identifies start/end of each word
  - Converts speech into a sequence of feature vectors at intervals of around 10 ms.
- Hypothesis Generator
  - Try each possible word in turn
- Language Model
  - Estimate the probability of the word
    - Can depend on the preceding words
- Acoustic Model
  - Calculate the probability density that an observed sequence of feature vectors corresponds to the chosen word

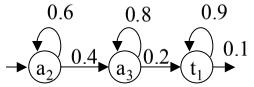
Speech Production Model

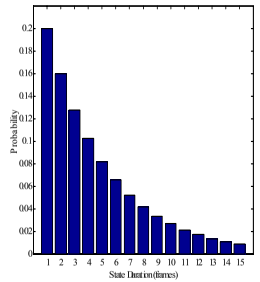


- Each phoneme in a word corresponds to a number of *model states* .
- Each model state represents a distinct sound with its own acoustic spectrum.
  - For each state, we store the mean value and variance for each of F features.
- When saying a word, the speaker stays in each state for one or more frames and then goes on to the next state.
  - The time in each state will vary according to how fast he/she is speaking.
  - Some speech sounds last longer than others

State Transitions

- Assume that the probability of proceeding onto the next state is p and the probability of staying in the same state is $1-p$.
  - The probability of being in any state for the next frame depends only on the current state and not on any previous history ⇒ this is a *Markov process*.
  - More realistic assumptions increase computation without improving performance.
  - The length of time, D, spent in any state follows a negative exponential distribution with an average duration of $1/p$ frames.



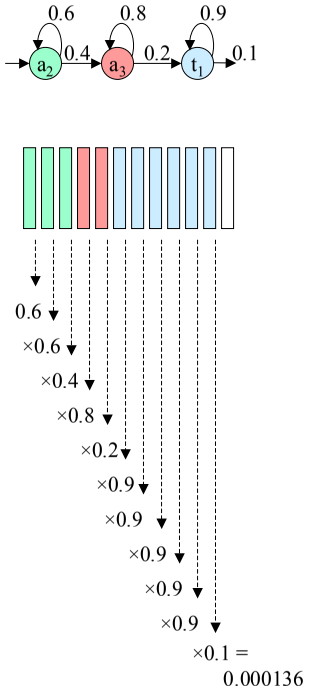


$$\text{pr}(D=n) = p(1-p)^{n-1} \quad \Rightarrow \quad \text{E}(D) = \sum_{n=1}^{\infty} np(1-p)^{n-1} = \frac{1}{p}$$

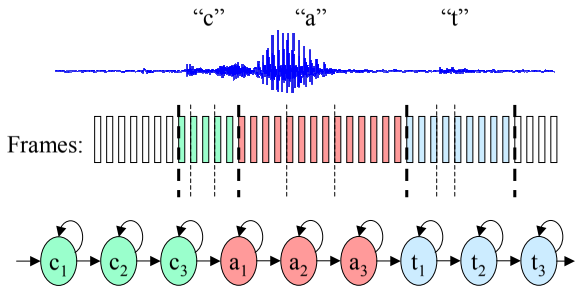
Since by differentiating $\sum_{n=0}^{\infty} x^n = \frac{1}{1-x} \quad \Rightarrow \quad \sum_{n=0}^{\infty} nx^{n-1} = \frac{1}{(1-x)^2}$

Alignment Probabilities

- We can calculate the probability of having:
  - 3 frames in the first state,
  - 2 in the second state, and
  - 6 in the third state
- A complete specification of which frames are in each state is an *alignment*.
- As before we can use log probabilities and add them instead of multiplying
  - Avoids dynamic range problems

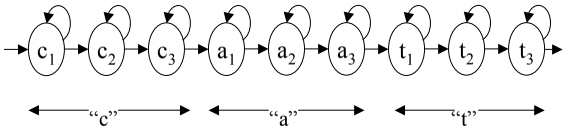


Hidden Markov Model



- To calculate the probability density (pd) that an observation matches a particular word with a given alignment, we multiply together:
  - the probability of the alignment
  - the output pds of each frame
- Try this for *every possible* alignment of *every possible* word sequence and choose the one with highest probability.
  - *Hidden* Markov Model ⇒ the correct alignment is hidden: we can't observe it directly.
- We talk of the probability density of the model *"generating"* the observed frames sequence.

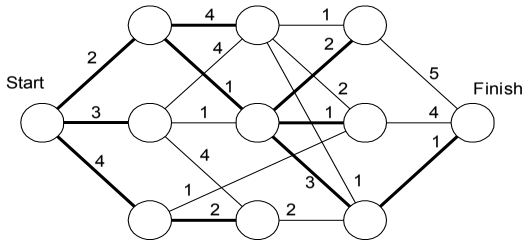
Hidden Markov Model Parameters



- A Hidden Markov Model for a word must specify the following parameters for state s:
  - The mean and variance for each of the F elements of the parameter vector: $\boldsymbol{\mu}_s$ and $\boldsymbol{\sigma}_s^2$.
    - These allow us to calculate $d_s(\mathbf{x})$: the output probability density of input frame $\mathbf{x}$ in state s.
  - The transition probabilities $a_{s,j}$ to every possible successor state.
    - $a_{s,j}$ is often zero for all j except $j=s$ and $j=s+1$ it is then called a *left-to-right, no skips* model.
- For a Hidden Markov Model with S states we therefore have around $(2F+1)S$ parameters.
  - A typical word might have S=15 and F=39 giving 1200 parameters in all.

Minimum Cost Paths

- Suppose we want to find the cheapest path through a toll road system: Start → Finish
  - In each circle we will enter the lowest cost of a journey from Start

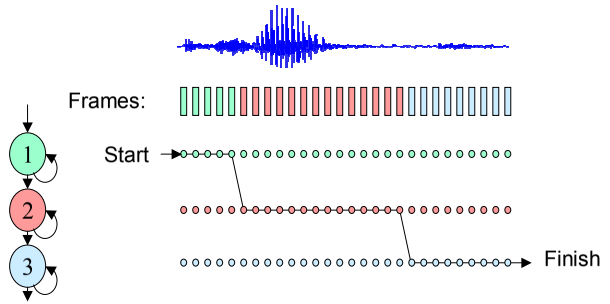


  - Begin by putting 0 in the Start circle
  - Now put 2, 3, 4 in the 2nd column circles and mark all three segments in bold.
    - This shows the lowest cost to each of these circles and the route by which you go.
  - Put 6, 3, 6 in the 3rd column circles and, in each case, mark the best segment from the previous column in bold.
  - Put 5, 4 and 6 in the 4th column.
  - Put 7 in the Finish circle.
- We can trace bold segments backwards from Finish to find the best overall path.

Dynamic Programming

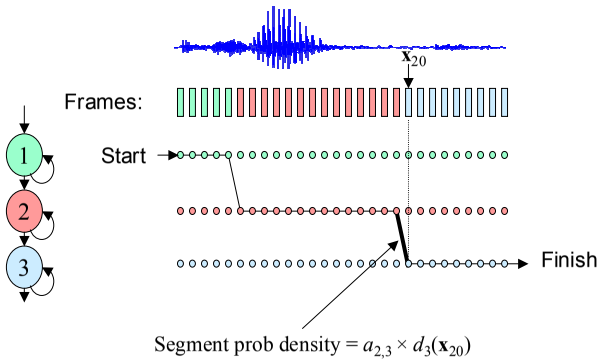
- This technique for finding the minimum cost path through a graph is known as *dynamic programming*.
- Three conditions must be true:
  - All paths through the graph must go from left to right.
  - The cost of each segment of a path must be fixed in advance: it must not depend on which of the other segments are included in the route.
  - The total cost of a path must just be the sum of each of its segment costs.
- Dynamic programming is guaranteed to find the path with minimum cost.
- We can also find the *maximum* cost path in the same way: in this case the "*costs*" are usually called "*utilities*" instead.
- We can use Dynamic Programming to find the "best" alignment of a sequence of feature vectors with a word's model.
  - "best" means the alignment with the highest production probability density.

Alignment Graph



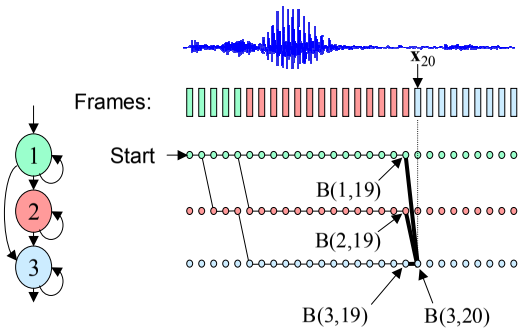
- We can draw an alignment graph (or lattice):
  - The columns correspond to speech frames
  - The rows correspond to model states
- Each possible path from Start to Finish corresponds to an alignment of the speech frames with the model.
  - All valid paths pass through each column in turn.
  - In going to the next column, a path is restricted to the state transitions allowed by the state diagram
    - In the above example a path must either remain in the same state or else go on to the next state.

Segment Utilities



- The probability density of a path segment going from state i to state j is the product of:
  - The probability of the transition: $a_{i,j}$
  - The output probability density of the corresponding input frame in state j: $d_j(\mathbf{x})$
- The probability density of the entire alignment is the product of its constituent segment pds.
  - This equals the pd that the model generates the observed sequence of feature vectors with this particular alignment.

Dynamic Programming Step



- Define $B(s,t)$ to be the probability density of the best partial alignment beginning at "Start" and ending with frame t in state s.
- Any alignment going through $(3,20)$ must go through either $(1,19)$, $(2,19)$ or $(3,19)$. Hence:

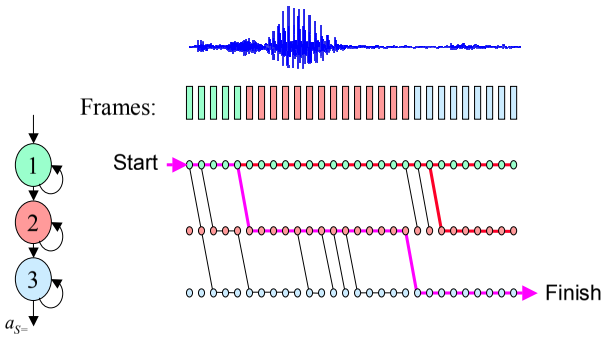
$$B(3,20) = \max_{s=1,2,3}\left(B(s,19) \times a_{s3}\right) \times d_3(\mathbf{x}_{20})$$

- In general, we can calculate $B(*,t)$ from $B(*,t-1)$:

$$B(k,t) = \max_{s}\left(B(s,t-1) \times a_{sk}\right) \times d_k(\mathbf{x}_t)$$

- Initialise this recursion by setting $B(1,1)=d_1(1)$

Viterbi Alignment



- This procedure is called Viterbi Alignment:

$B(1,1) = d_1(\mathbf{x}_1)$; $B(s,1) = 0$ for $s > 1$
for t=2:T
  for k=1:S
    $z(k,t)$=j=argmax$(B(s,t\text{-}1)\times a_{sk})$ for all s with $a_{sk} \neq 0$
    $B(k,t)$=$B(j,t\text{-}1)\times a_{jk}\times d_k(\mathbf{x}_t)$
  end
end

- The best path has prob density = $B(S,T)\times a_{S=}$
  - $a_{S=}$ is the exit probability from the final state
- The $z(k,t)$ array stores the information needed to trace back the best path.

Isolated Word Recognition

- Requires the speaker to insert a gap between each words
- Used for budget systems with little CPU power
- Recognition:
  - Extract a word-long segment of speech, s, from the input signal. Convert it into a sequence of frames.
  - Calculate $\text{pr}(w|s) = \text{pr}(w) \times \text{pr}(s|w)$ for each possible word, w, in the recognition vocabulary.
    - $\text{pr}(w)$ is the prior probability of the word: get this from word frequencies or word-pair frequencies (e.g. “minister” often follows “prime”).
    - $\text{pr}(s|w)$ is obtained by using the Viterbi alignment algorithm to find the log probability density of the best alignment of s with the model for w.
  - Choose the word with the highest probability
  - Need to create a separate Hidden Markov model for each word in the vocabulary. Long