Lecture # 15
Sessio 2003

Part I: Designing HMM-based ASR systems
Part II: Training continuous density HMMs

Rita Singh

School of Computer Science
Carnegie Mellon University

Designing graph structures for the language HMM : Understanding and dealing with the complexity introduced by the use of sub-word units

Creating HMMs for word sequences: units

- Large vocabulary systems do not use words as units of sound
  - Vocabulary consists of tens of thousands of words
  - Difficult to find enough examples of every word even in large training corpora
  - Words not seen during training can never be learned, and never be recognized

- Instead, words are broken down into sub-word units
  - Many more instances of sub-word units in a corpus than of words, HMMs parameters can be better estimated
  - Sub-word units may be combined in different ways to form new words, which can be recognized
    - Not necessary to see all words in the vocabulary during training
  - usually phonetically motivated and therefore called phones

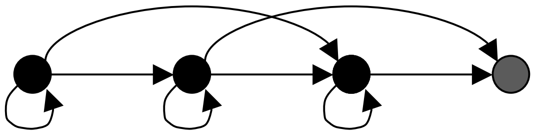
Creating HMMs for word sequences: Context independent units

Example:

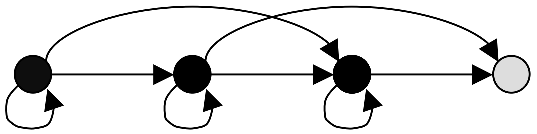
Word	Phones
Rock	R AO K

Units are *context indepdenent* because the identity of the phonemes does not depend on their neighbors

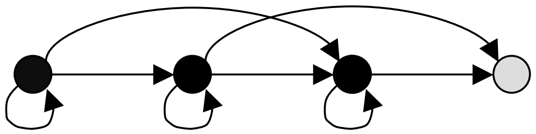
- Every word is expressed as a sequence of sub-word units
- Each sub-word unit is modeled by an HMM
- Word HMMs are constructed by concatenating HMMs of sub-word units
- Composing word HMMs with context independent units does not increase the complexity the language HMM



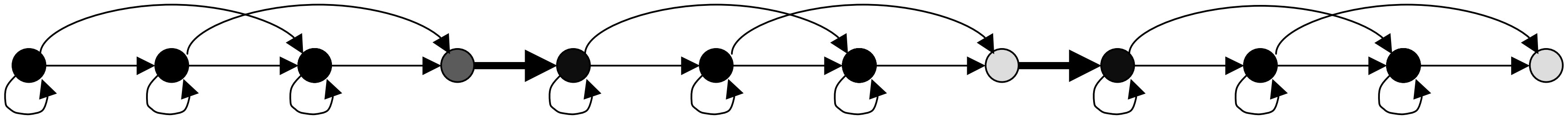
HMM for /R/



HMM for /AO/



HMM for /K/



Composed HMM for ROCK

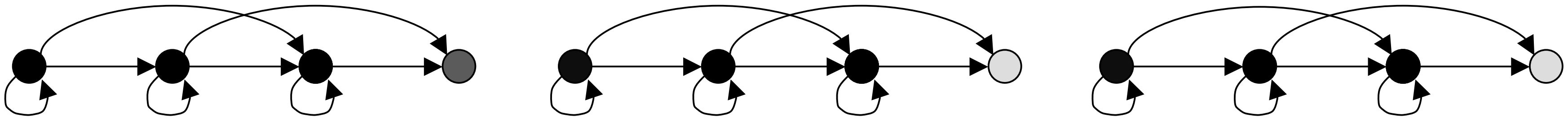
Creating HMMs for word sequences: Word-internal context dependent units

Example:

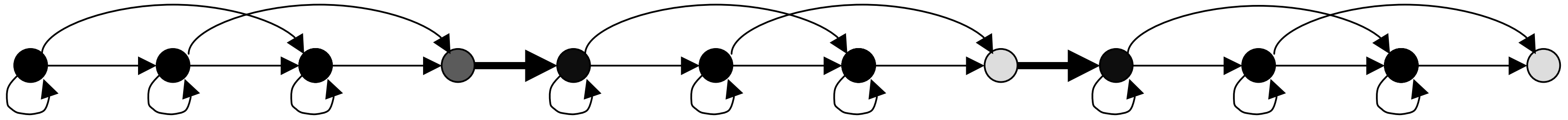
Word	Phones	Triphones
Rock	R AO K	R,AO(R,K),K

The sub-word unit AO(R,K) is related to the contexts R and K. It is *context depdenent.*

- Phonemes are coarse units
  - When /AO/ is preceded by R and followed by K, it is spectrographically different than when it is preceded by /B/ and followed by /L/
- Triphones are phonetic units *in context.*
- If triphones were used only inside words, and the units used at word endings were context independent, the language HMM complexity is the same as that when all units are context independent.



HMM for /R/ HMM for /AO(R,K)/ HMM for /K/



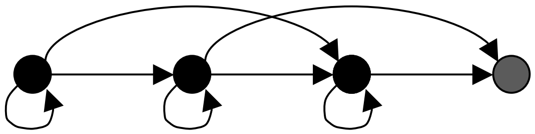
Composed HMM for ROCK

Creating HMMs for word sequences: Cross-word context dependent units

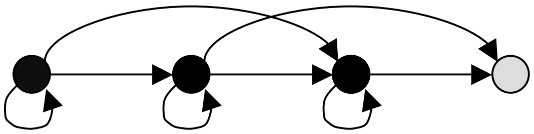
Example:

Word	Phones	Triphones
Rock	R AO K	R(*,AO), AO(R,K),K(AO, *)

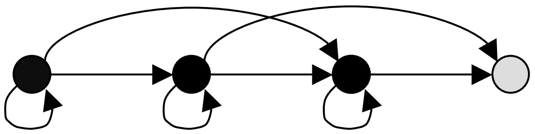
- When triphones are used at word boundaries, the HMMs used to compose the word become dependent on adjacent words!
  - If “Rock” were followed by “STAR   S T AA R”, the final triphone for ROCK would be K(AO,S)
  - If “Rock” were followed by “MUSIC M Y UW Z I K”, the final triphone in ROCK would be K(AO, M)



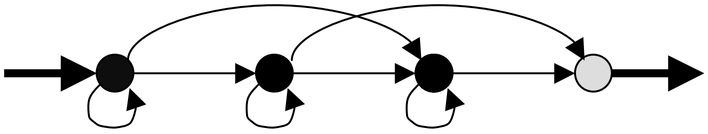
HMM for /R(?, AO)/



HMM for /AO(R,K)/



HMM for /K(AO, ?)/



Composed HMM for ROCK

Building sentence HMMs using sub-word units

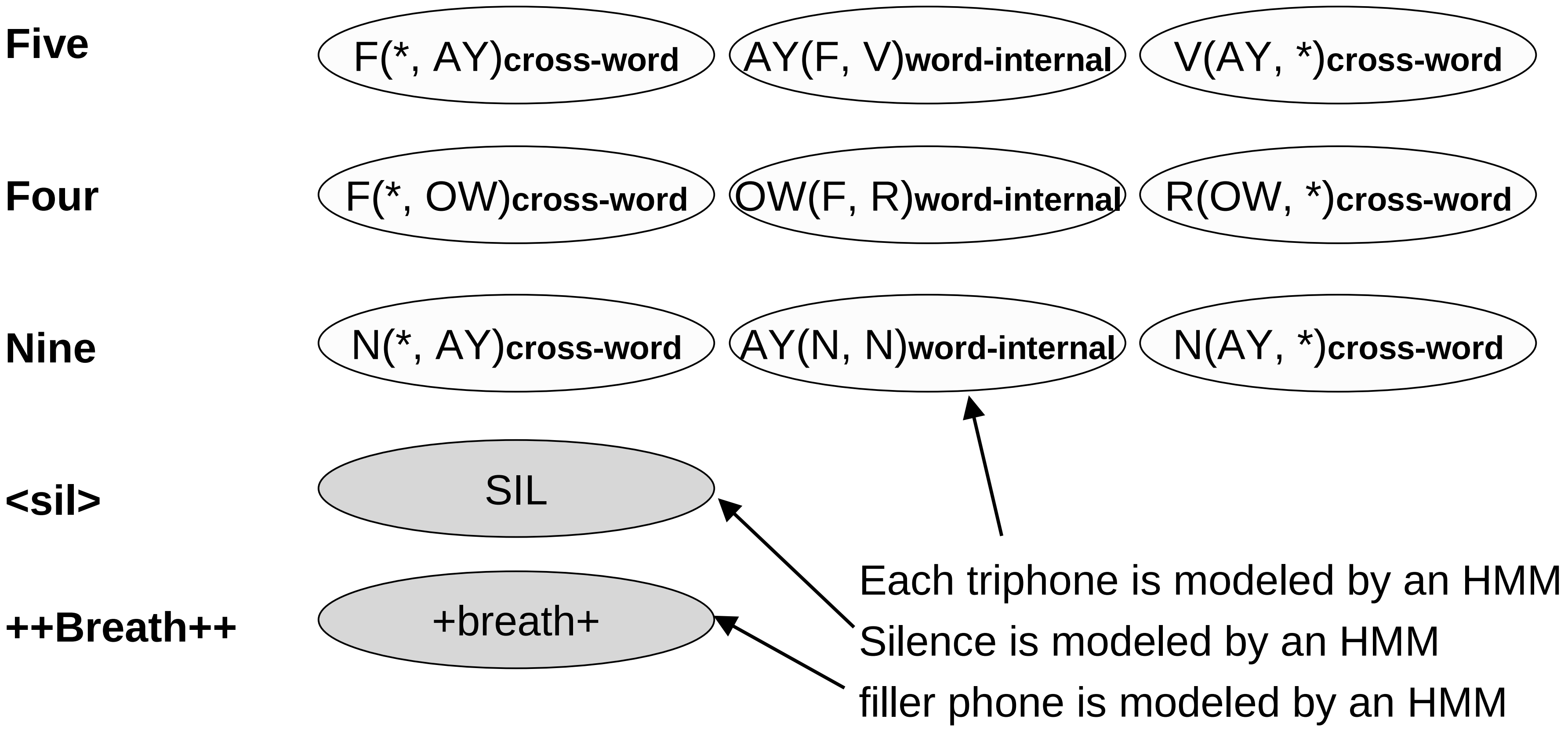
Dictionary

Five:	**F AY V**
Four:	**F OW R**
Nine:	**N AY N**
<sil>:	**SIL**
++breath++:	**+breath+**

Listed here are five "words" and their pronunciations in terms of "phones". Let us assume that these are the only words in the current speech to be recognized. The recognition vocabulary thus consists of five words. The system uses a dictionary as a reference for these mappings.

Building sentence HMMs using sub-word units

Using the dictionary as reference, the system first maps each word into triphone-based pronunciations. Each triphone further has a characteristic label or type, according to where it occurs in the word. Context is not initially known for cross-word triphones.



Five	F(*, AY)**cross-word**	AY(F, V)**word-internal**	V(AY, *)**cross-word**
Four	F(*, OW)**cross-word**	OW(F, R)**word-internal**	R(OW, *)**cross-word**
Nine	N(*, AY)**cross-word**	AY(N, N)**word-internal**	N(AY, *)**cross-word**
<sil>	SIL		
++Breath++	+breath+		

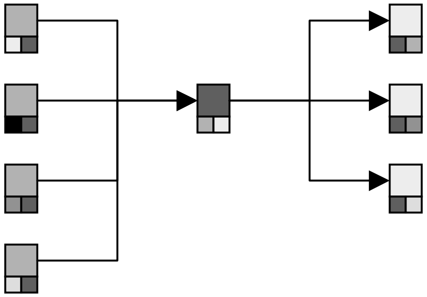
Each triphone is modeled by an HMM

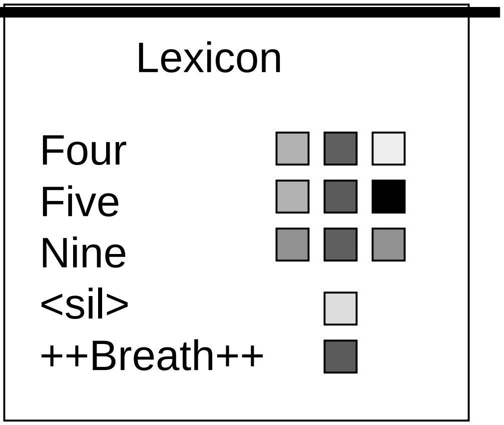
Silence is modeled by an HMM

filler phone is modeled by an HMM

Building the triphone-based UNIGRAM sentence HMM

HMM for “Four”.
This is composed of 8 HMMs.

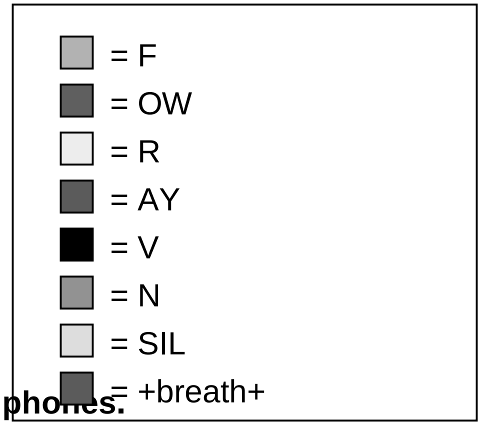




Lexicon

Four
Five
Nine
<sil>
++Breath++

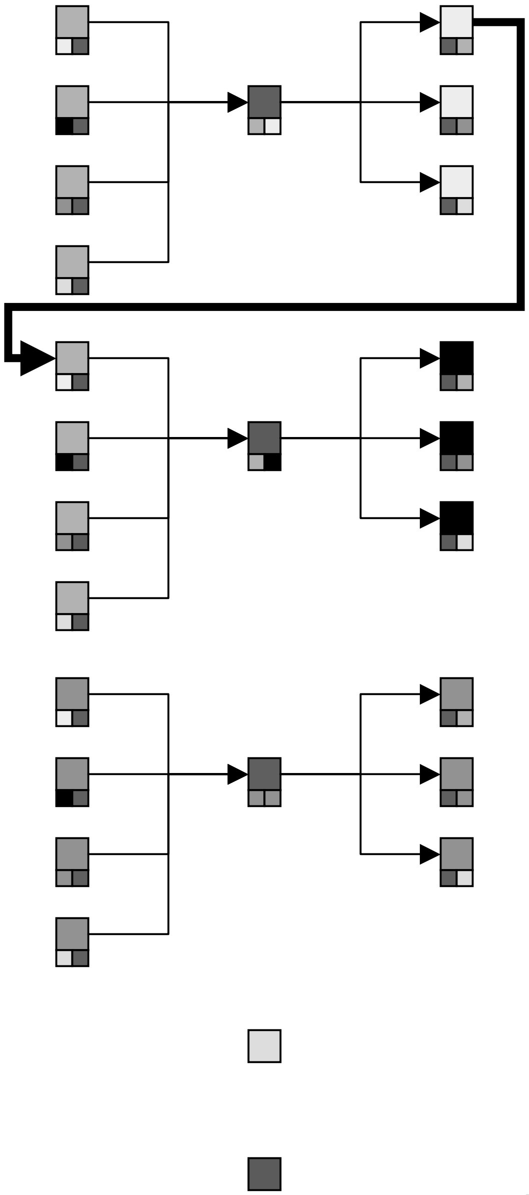
Each triple-box represents a triphone. Each triphone model is actually a left-to-right HMM (could have any number of states. Each state is a senone.)



A triphone is a single phone with context INFORMATION. It is not a literal sequence of 3 phones.

Expand the word Four
- All last phones (except filler) become left contexts for first phone of Four.
- All first phones (except filler) become right contexts for last phone of Four
- Silence can form contexts, but itself does not have any context dependency.
- Filler phones (e.g. +breath+) are treated as silence when building contexts. Like silence, they themselves do not have any context dependency.

Building the triphone-based UNIGRAM sentence HMM

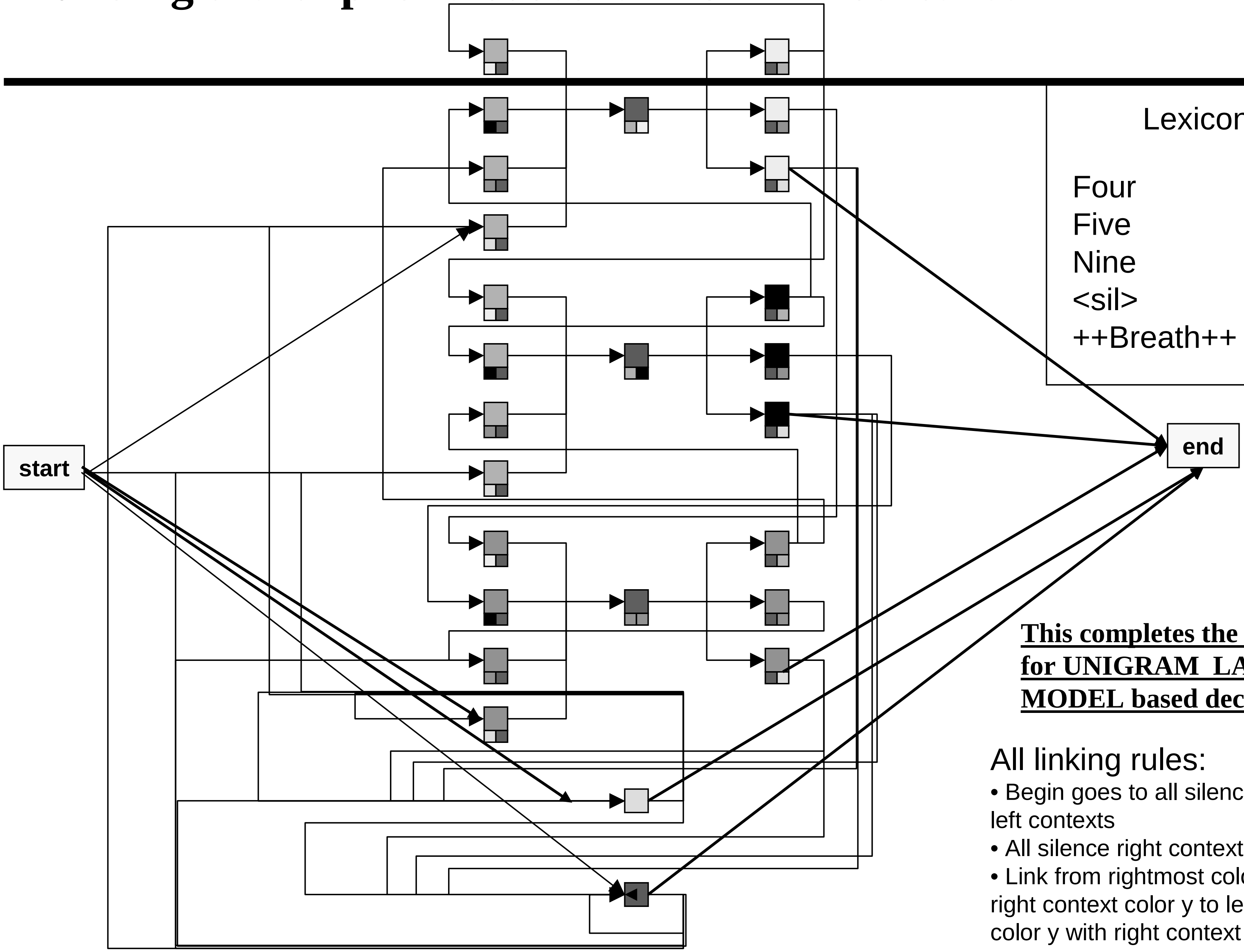


Lexicon

Four	▒	▓	□
Five	▒	▓	■
Nine	▒	▓	▒
<sil>		□	
++Breath++		▓	

Linking rule:
Link from rightmost color x
with right context color y
to leftmost color y with
right context color x

Building the triphone-based UNIGRAM sentence HMM



Lexicon

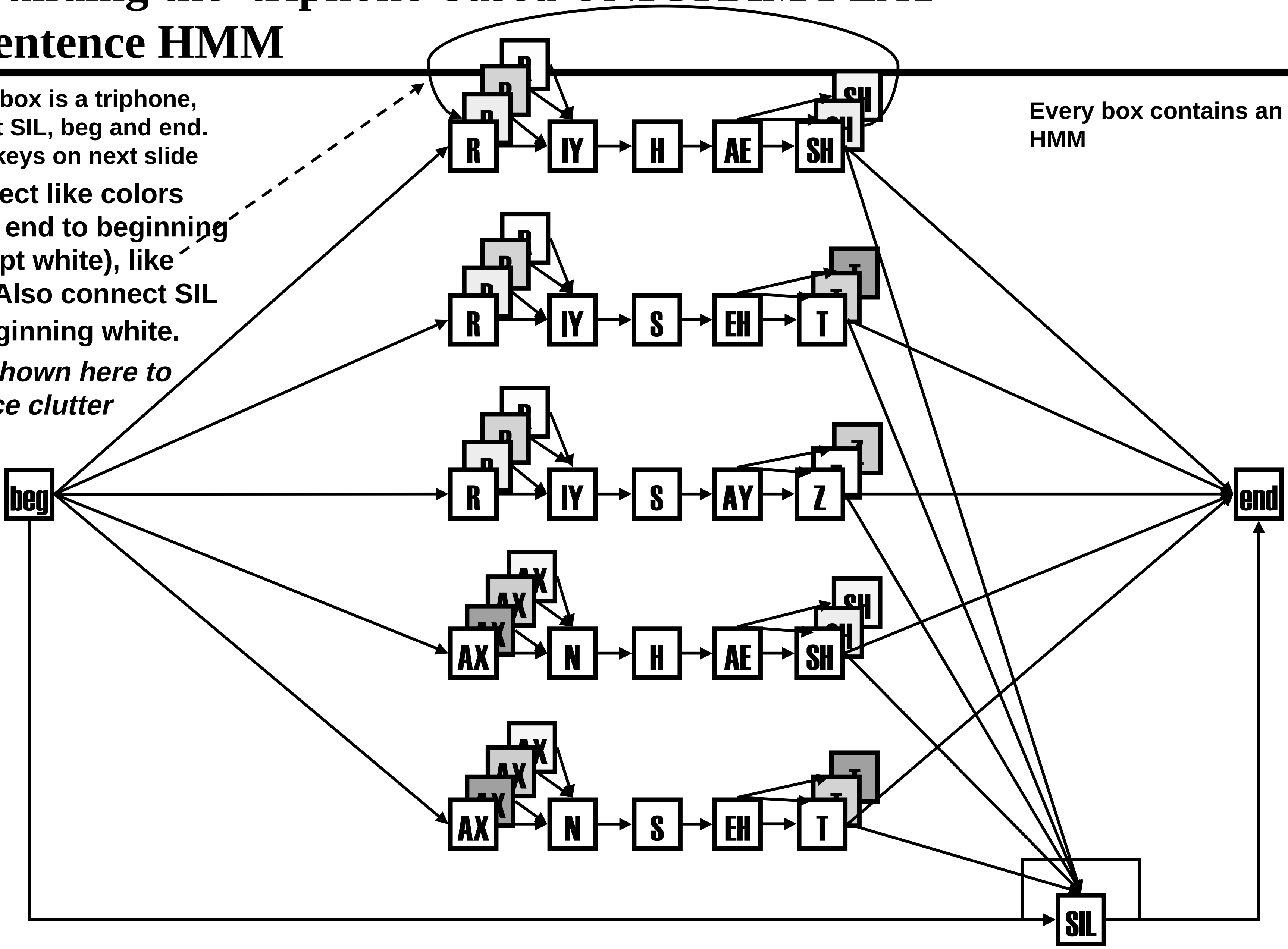
Four	▒ ▓ ░
Five	▒ ▓ █
Nine	▒ ▓ ▒
<sil>	░
++Breath++	▓

This completes the HMM for UNIGRAM LANGUAGE MODEL based decoding.

All linking rules:

- Begin goes to all silence left contexts
- All silence right contexts go to end
- Link from rightmost color x with right context color y to leftmost color y with right context color x

Building the triphone-based UNIGRAM *FLAT* sentence HMM



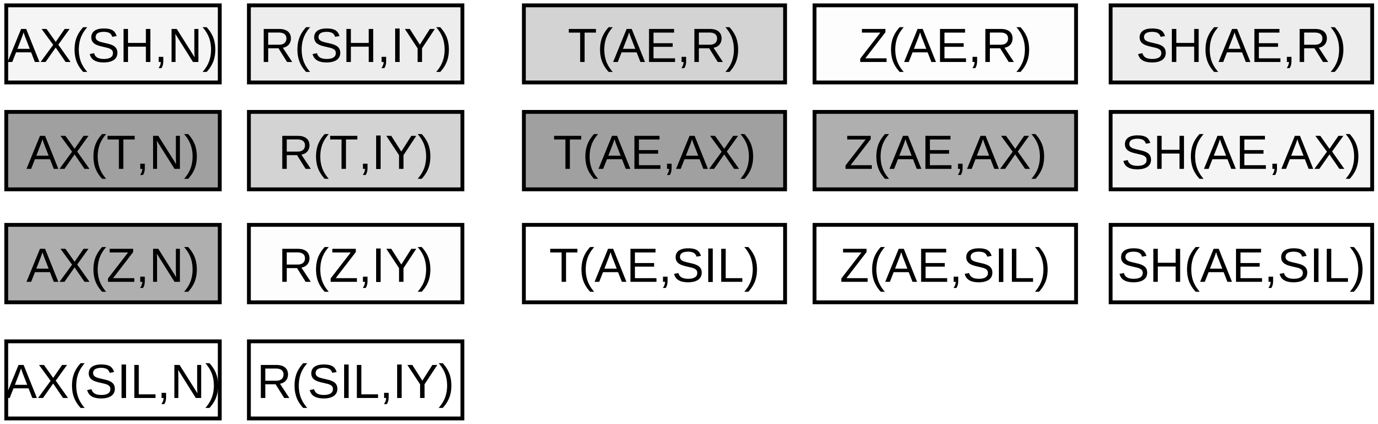
Every box is a triphone, except SIL, beg and end. color keys on next slide

Connect like colors from, end to beginning (except white), like this. Also connect SIL to beginning white.

Not shown here to reduce clutter

Every box contains an HMM

Building the triphone-based UNIGRAM sentence HMM: color key to the previous slide

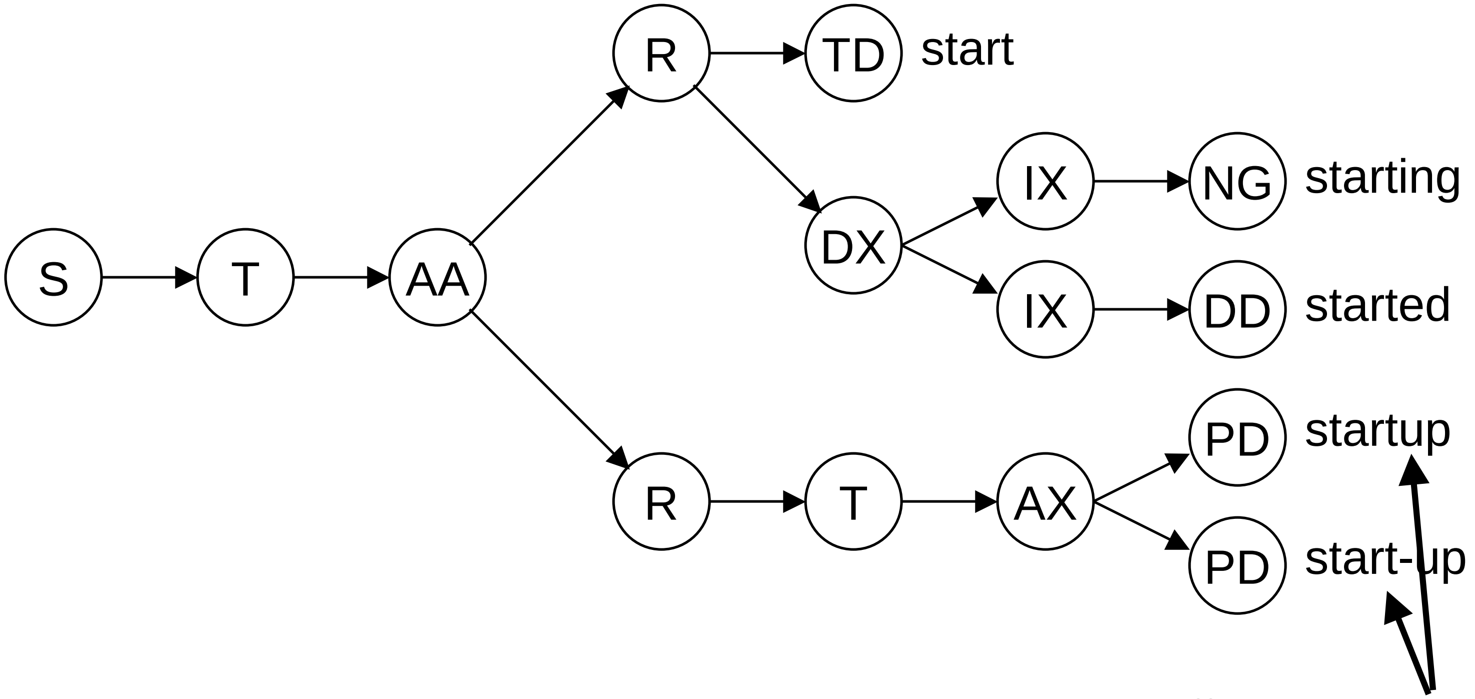


Rehash	R IY H AE SH
Reset	R IY S EH T
Resize	R IY S AY Z
Unhash	AX N H AE SH
Unset	AX N S EH T
<sil>	SIL

Simplifying decoding

- The example we have seen is that of FLAT search decoding with unigram language structure
  - The structure of the vocabulary is flat : each word has its own representation
- Sentence HMM for bigram and trigram language model based flat search type graphs can get very large and complicated
- Reducing the size of the Sentence HMM is an important engineering issue in designing a decoder
- FLAT search is accurate, but memory-intensive and slow

Lextree



Different words
with identical pronunications
must have different terminal
nodes

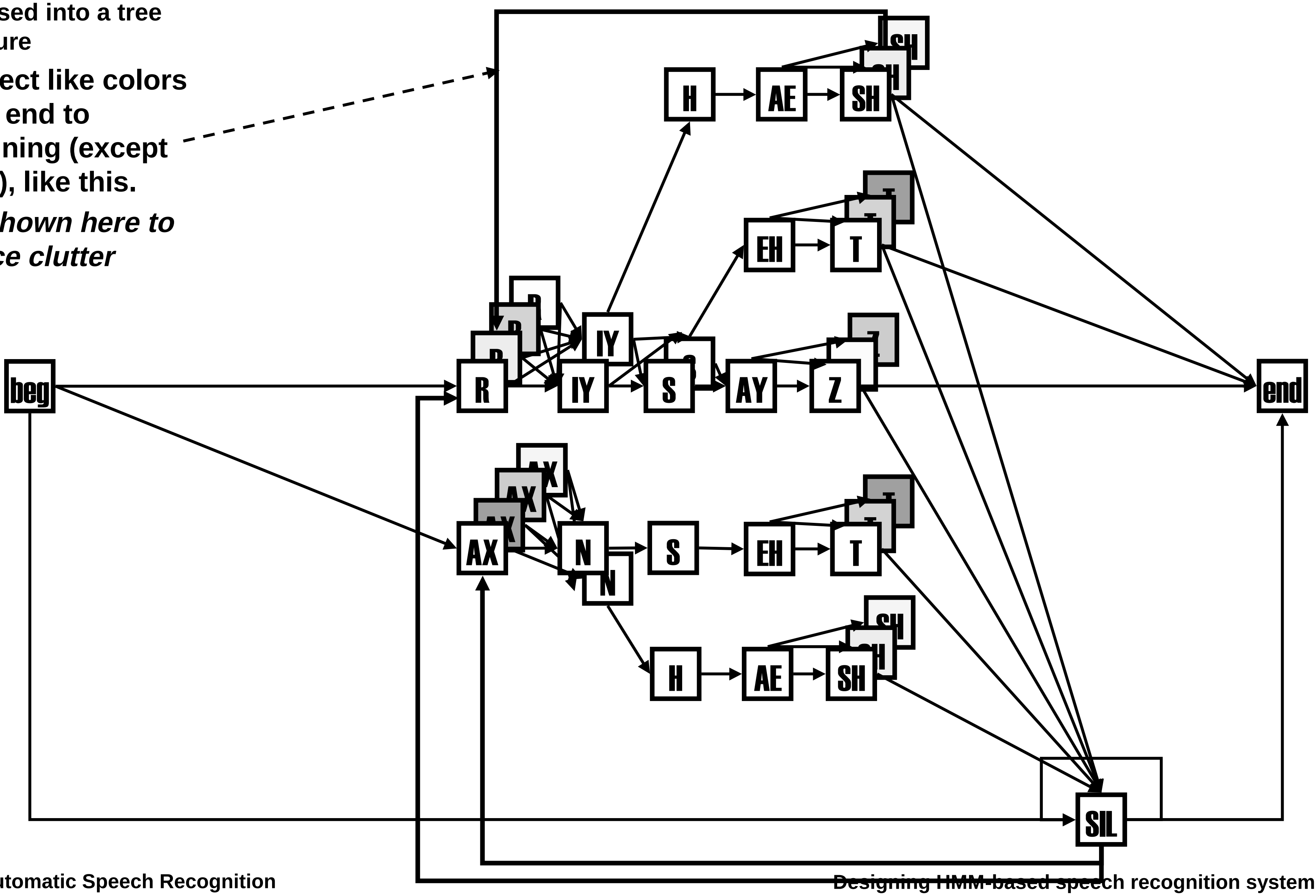
- Words share phone (or triphone) HMMs. Use phonetic similarity to reduce size and memory requirements, and decrease computation to increase speed of decoding

Building the triphone-based UNIGRAM *LEXTREE* sentence HMM

In a lextree, phones are collapsed into a tree structure

Connect like colors from, end to beginning (except white), like this.

Not shown here to reduce clutter

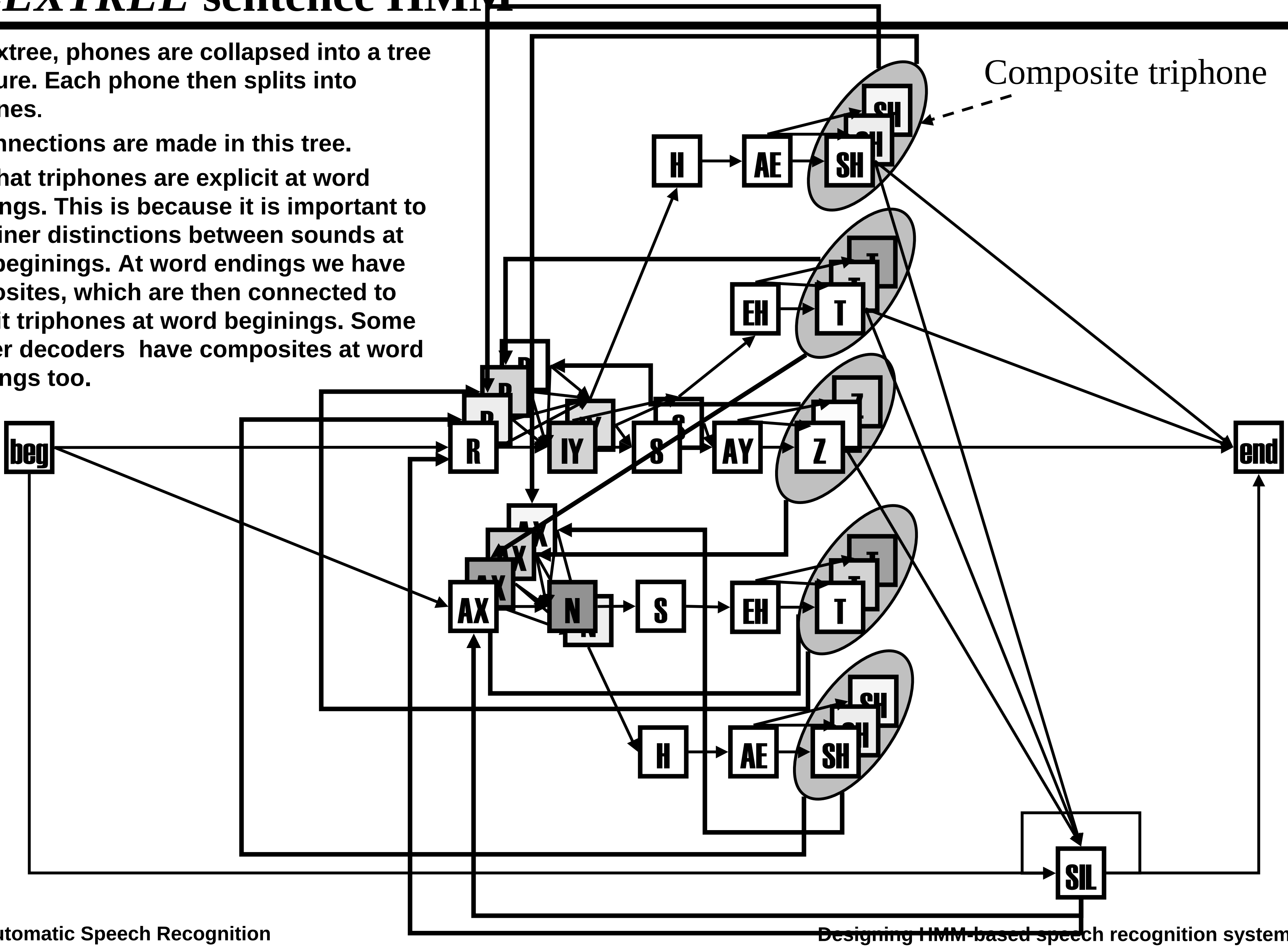


Building the triphone-based UNIGRAM *LEXTREE* sentence HMM

In a lextree, phones are collapsed into a tree structure. Each phone then splits into triphones.

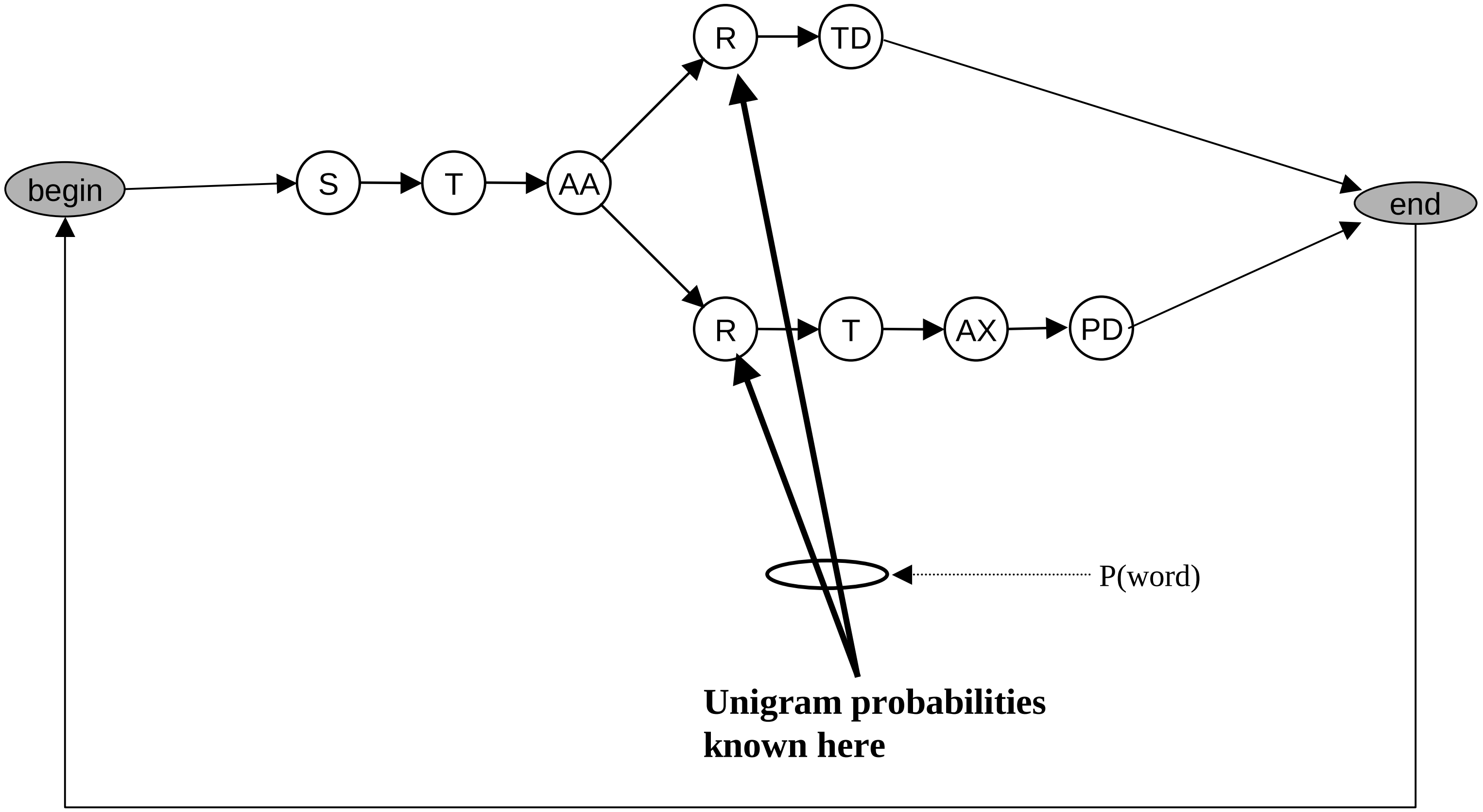
All connections are made in this tree.

Note that triphones are explicit at word beginings. This is because it is important to have finer distinctions between sounds at word beginings. At word endings we have composites, which are then connected to explicit triphones at word beginings. Some simpler decoders have composites at word beginings too.



Unigram Lextree Decoding

Figure is conceptual. A more precise figure for triphone-based lextrees would consider triphonetic contexts

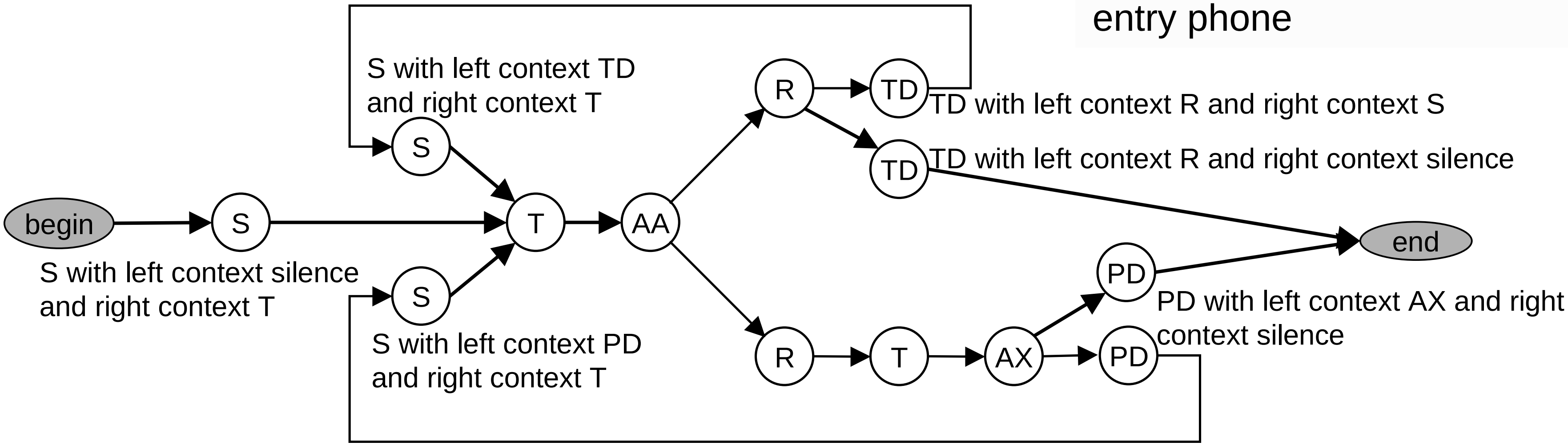


Unigram probabilities known here

Unigram Lextree Decoding (More precisely)

Detailed triphone based lextree for this example

Had there been multiple entry phones, there would have been multiple copies of TD and PD at the exit, one for each possible entry phone

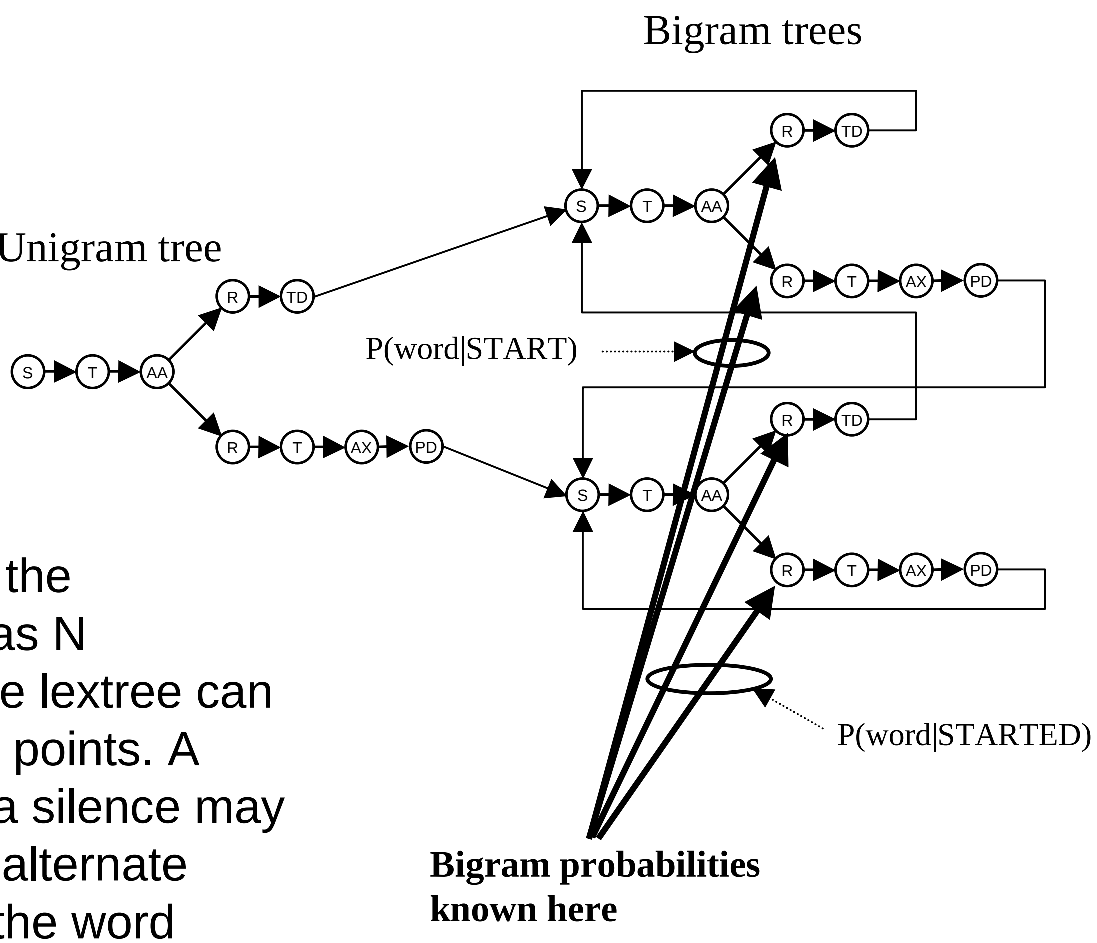

S with left context TD
and right context T
R
TD
TD with left context R and right context S
S
TD
TD with left context R and right context silence
begin
S
T
AA
end
S with left context silence
and right context T
S
PD
PD with left context AX and right
context silence
S with left context PD
and right context T
R
T
AX
PD

A unigram lextree has several entry points, one for each possible preceding phone

- Here the possible preceding phones are TD, PD and silence (at “begin”)
- There are two triphone models for S, one with left context TD, and the other with left context PD
- Note that the model is a tree only from the second phone

Bigram Lextree Decoding

If all words have only a single pronunciation, all lextrees have only a single entry point since they can only be entered from a specific word

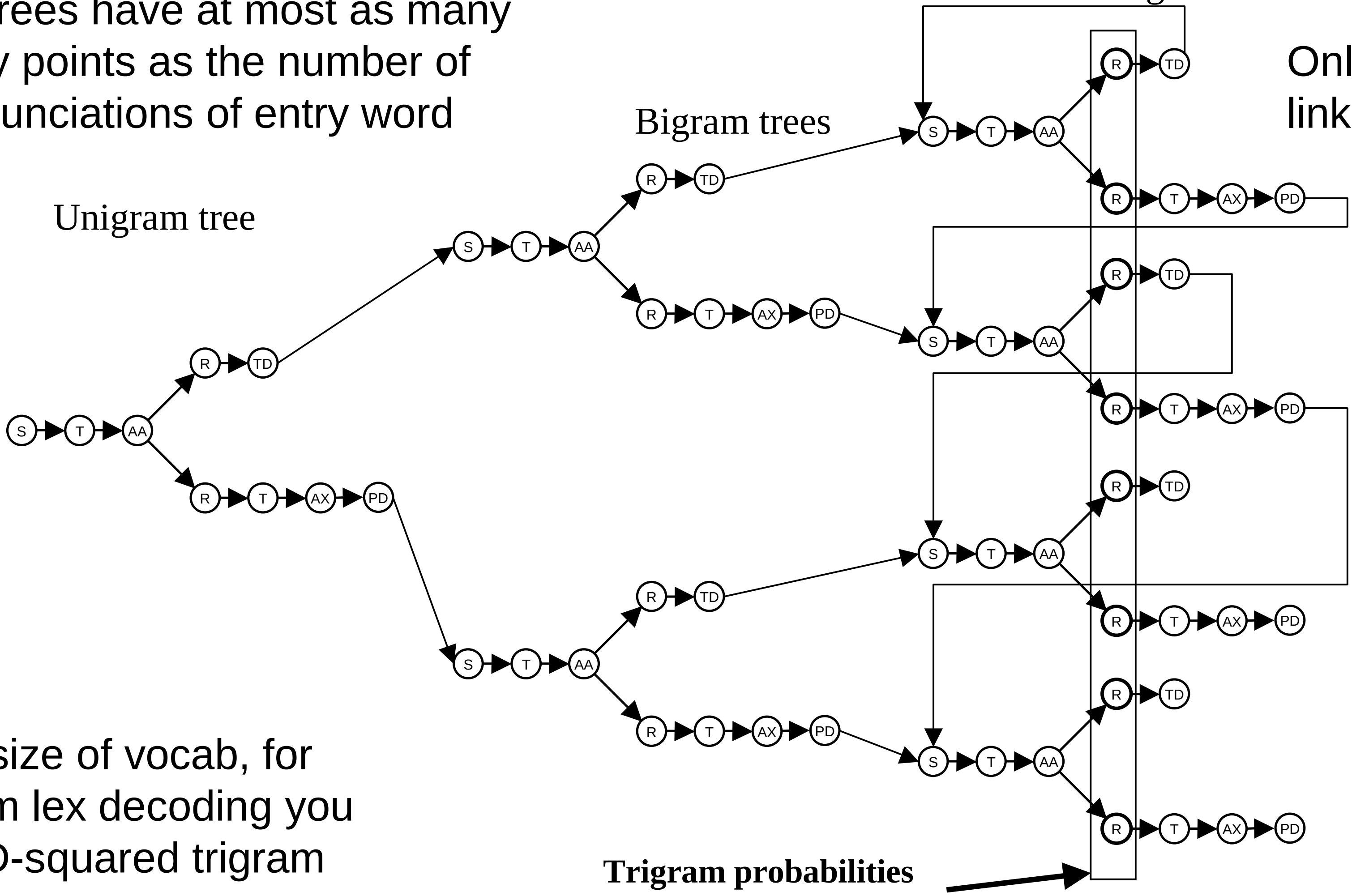


More generally, If the preceding word has N pronunciations, the lextree can have upto N entry points. A word followed by a silence may be considered an alternate pronunciation for the word

Trigram Lextree Decoding

Lextrees have at most as many entry points as the number of pronunciations of entry word

Only some links shown



Trigram trees

Bigram trees

Unigram tree

Trigram probabilities known here

If D = size of vocab, for Trigram lex decoding you need D-squared trigram trees

Issues with Lextrees

- Word identities not known on entry. This complicates language HMM structure even more than in flat search
  - Lextree based language HMMs actually get much *bigger* than the corresponding flat HMMs in all but the unigram case
    - A flat HMM that incoroporates Ngram probabilities and has a vocabulary of D words needs $D^{N-1} + D^{N-2} + .. D$ word HMMs. A lextree HMM for the same vocabulary needs $D^{N-1} + D^{N-2} + .. D$ lextrees.
  - The number or transitions between sentence HMM state is proportionately larger
  - Several heuristics proposed to amend this problem

Reducing the size of lextrees incorporating Ngrams: Approximate decoding structures

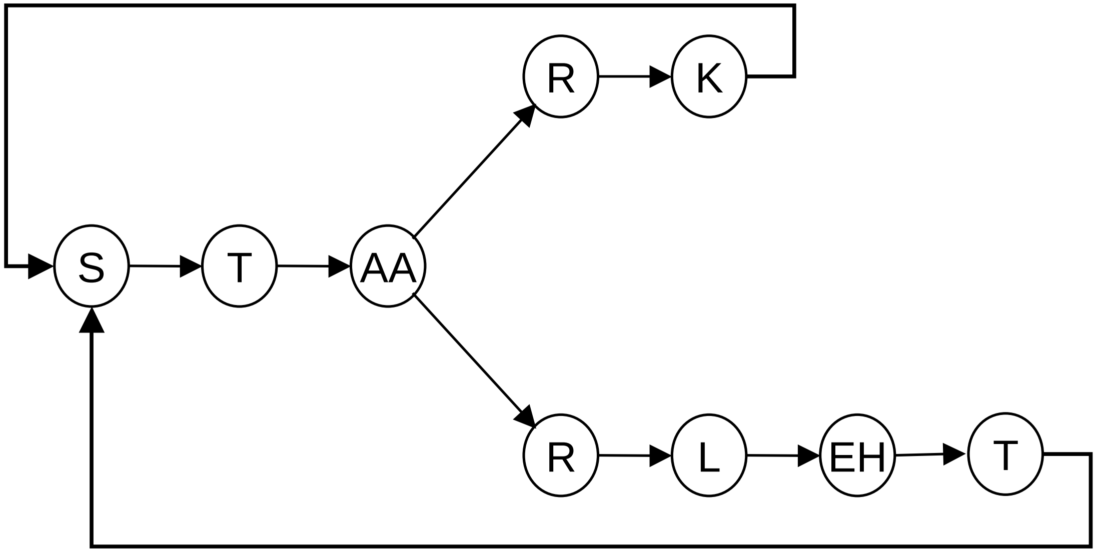
- Reduced size lextrees
  - Ngram decoding with single lextree
  - Ngram decoding with switching lextrees

- Effect on recognition
  - The HMM structure supports weaker linguistic constraints
  - Recognition is suboptimal, but less memory intensive

Approximate decoding structures

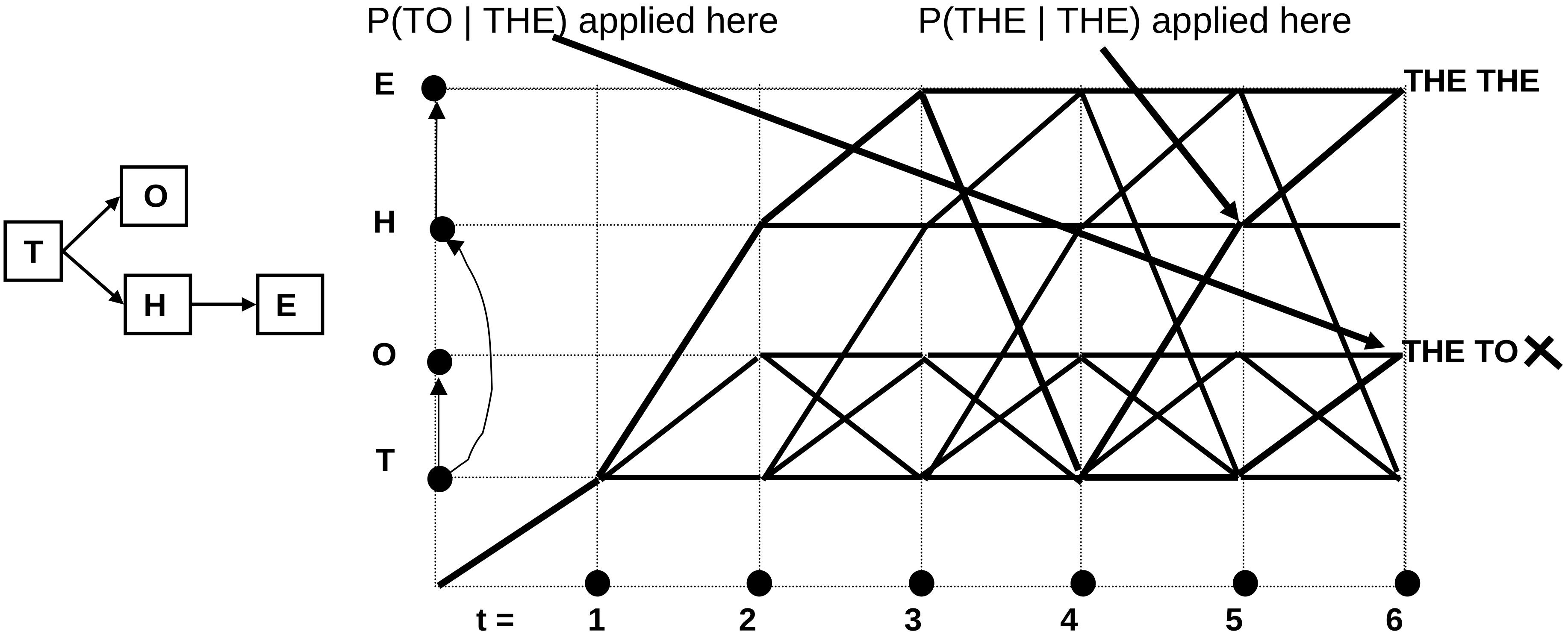
Single lextree based decoding

Word histories for active words at every time instant kept in a backpointer table



Decoding with a single lextree introduces errors

- Example with simplified two-state HMMs
  - P(x1..x3, THE) > P(x1..x3, TO)
  - P(x1..x3,THE)*P(x4..x6,THE | THE) < P(x1..x3,TO)*P(x4..x6,THE | TO).
  - However, P(x4..x6,THE | TO) = P(x4..x6 | THE)*P(THE|TO) can never be computed since TO is never considered as a context.
  - Although, mathematically TO THE must win, here only THE THE can be hypothesized



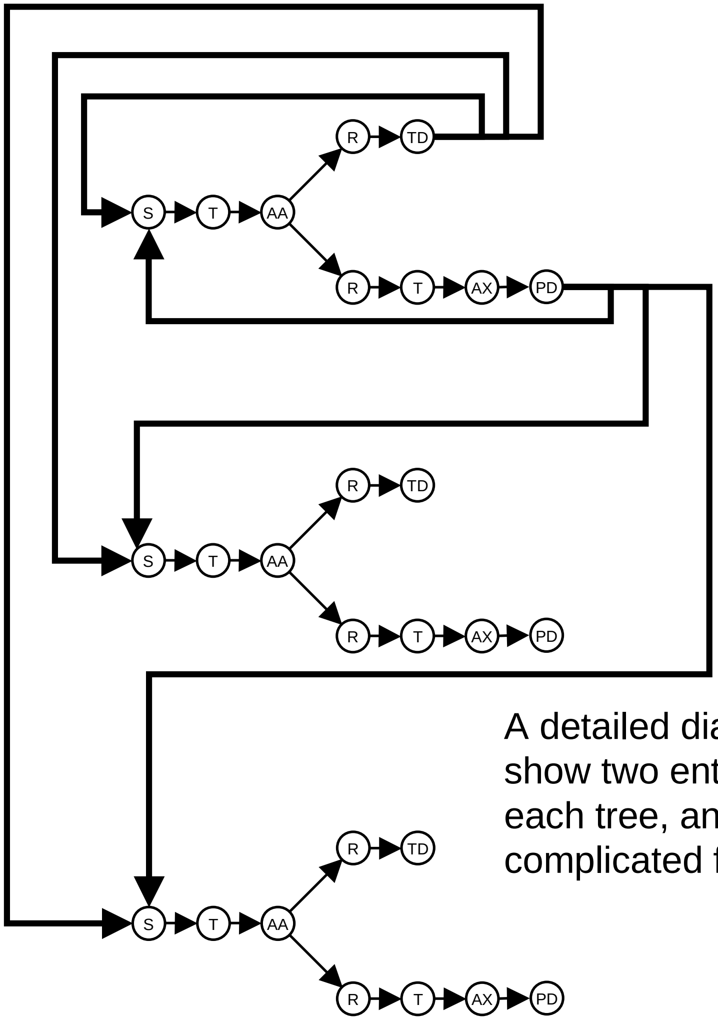
Approximate decoding structures

Switching Lextree with 3 lextrees: Multiplexing in time

- All three lextrees similarly connected

- Entry points to tree staggered in time
  - E.g. one may transition into lextree1 only at t=1,4,7,.., into lextree2 only at t=2,5,8…, and into lextree3 only at t=3,6,9,..

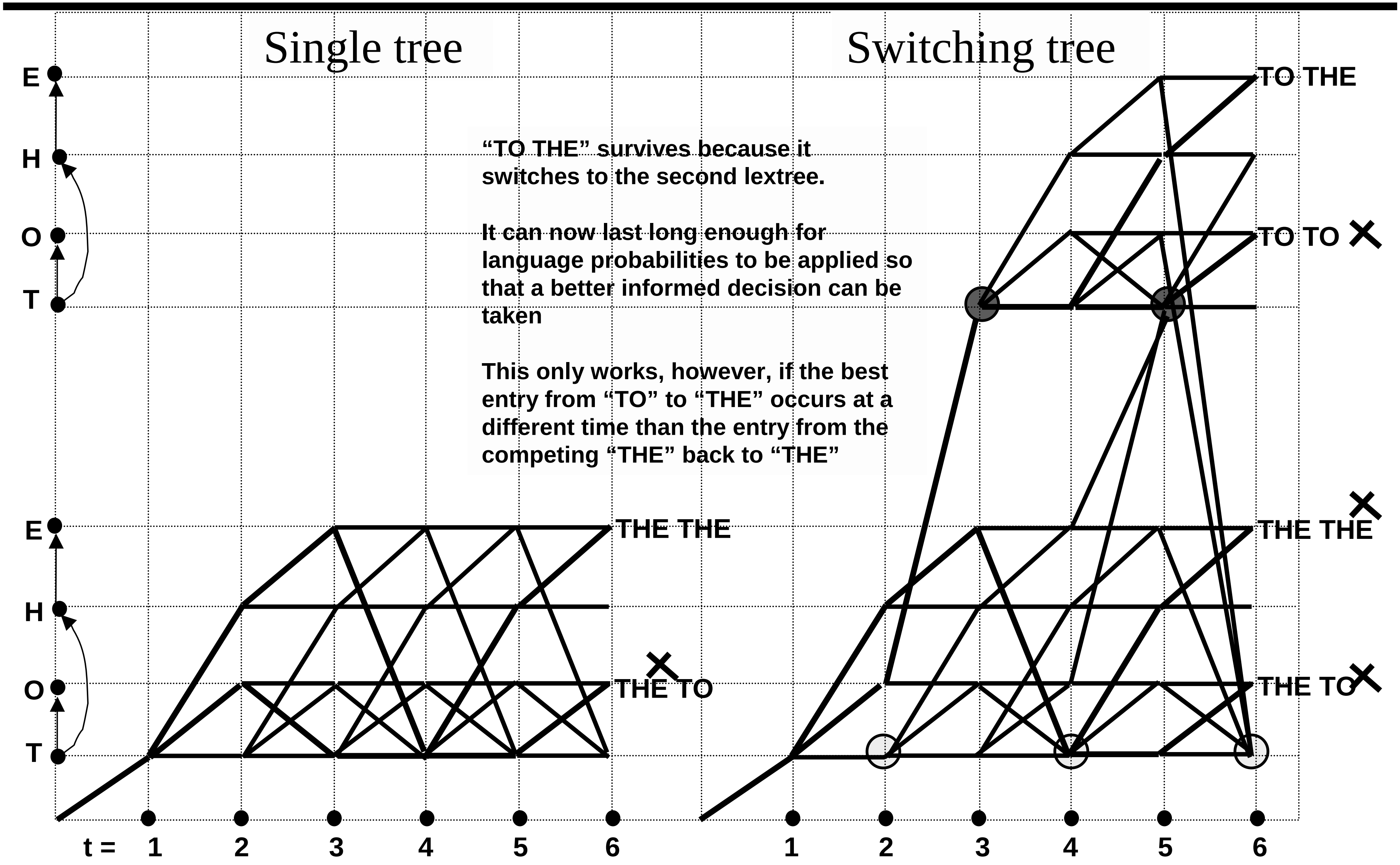
- Ngram contexts needed for word probabilities are obtained from a backpointer table that maintains Viterbi history of any path



A detailed diagram would show two entry points for each tree, and a more complicated figure

Each lextree can have many entry points.

Switching lextrees



“TO THE” survives because it switches to the second lextree.

It can now last long enough for language probabilities to be applied so that a better informed decision can be taken

This only works, however, if the best entry from “TO” to “THE” occurs at a different time than the entry from the competing “THE” back to “THE”

Reducing the size of flat HMMs: Approximate decoding structures

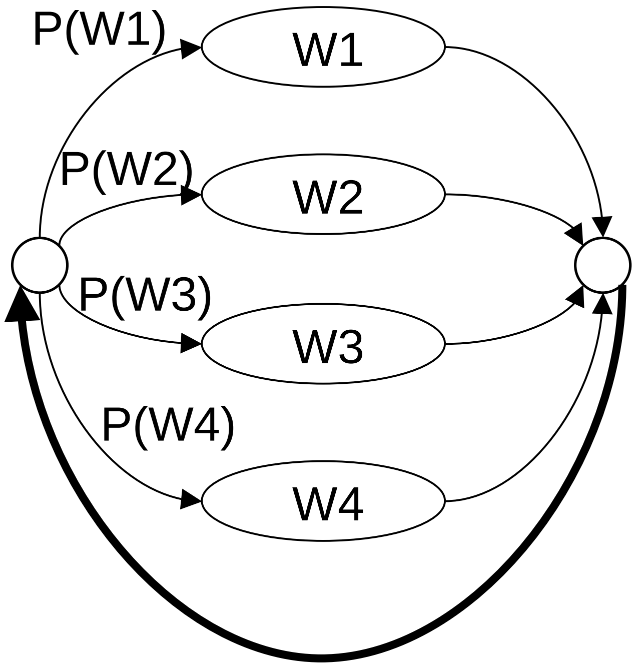
- Use lower order Ngram HMM structures to perform recognition using higher order Ngram probabilities
  - Ngram decoding from unigram structures
  - Ngram decoding from bigram structures (Pseudo-trigram search)
  - Use backpointer table to obtain word history
- Effect on recognition
  - Imprecise application of high-order Ngram probabilities
  - Reduced memory requirements

Approximate decoding structures

pseudo-bigram decoding from unigram structure in Flat search

- Use a simple unigram structure
- Apply bigram probabilities
  - Context for bigram obtained from word history
  - Simpler structure needs less memory and computation
  - Imprecise

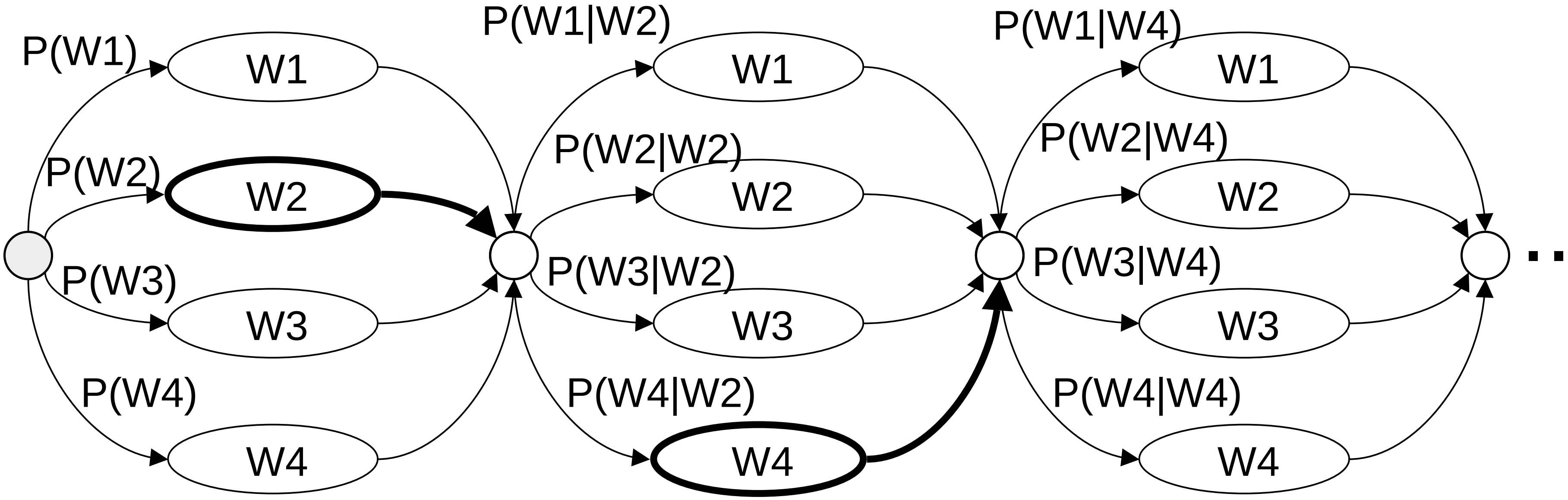
Conventional Unigram



Approximate decoding structures
pseudo-bigram decoding from unigram structure in Flat search

- Use a simple unigram structure
- Apply bigram probabilities
  - Context for bigram obtained from word history
  - Simpler structure needs less memory and computation
  - Imprecise

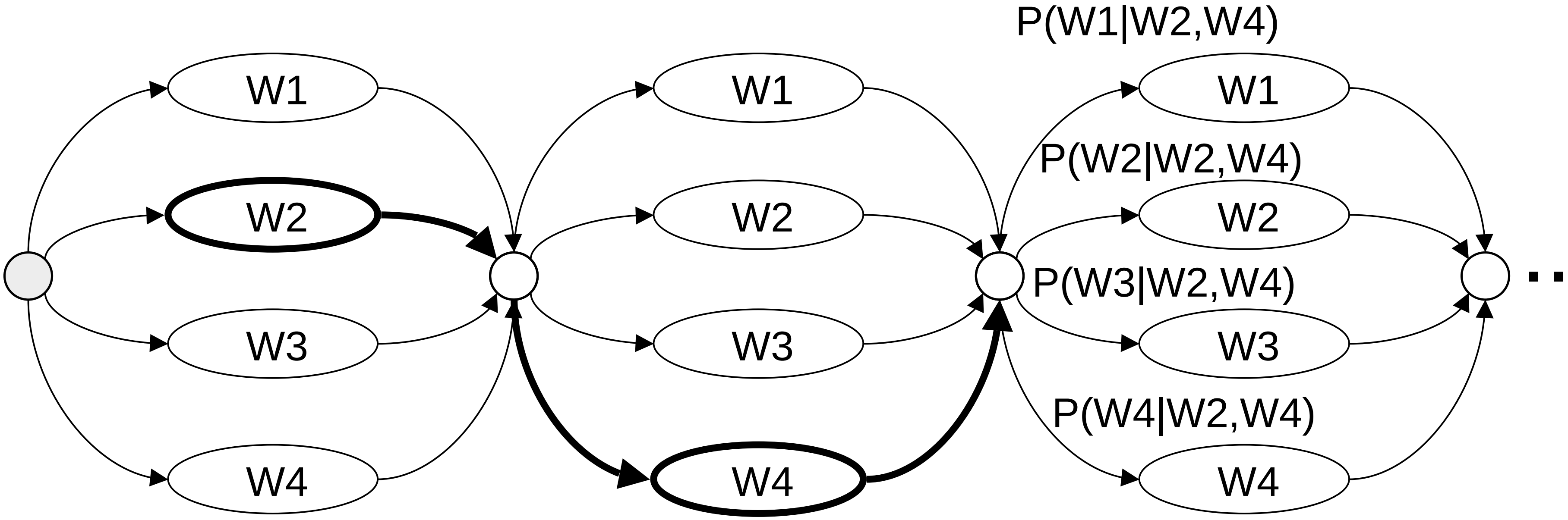
Pseudo-bigram from Unigram structure



Approximate decoding structures
pseudo-trigram decoding from unigram structure in Flat search

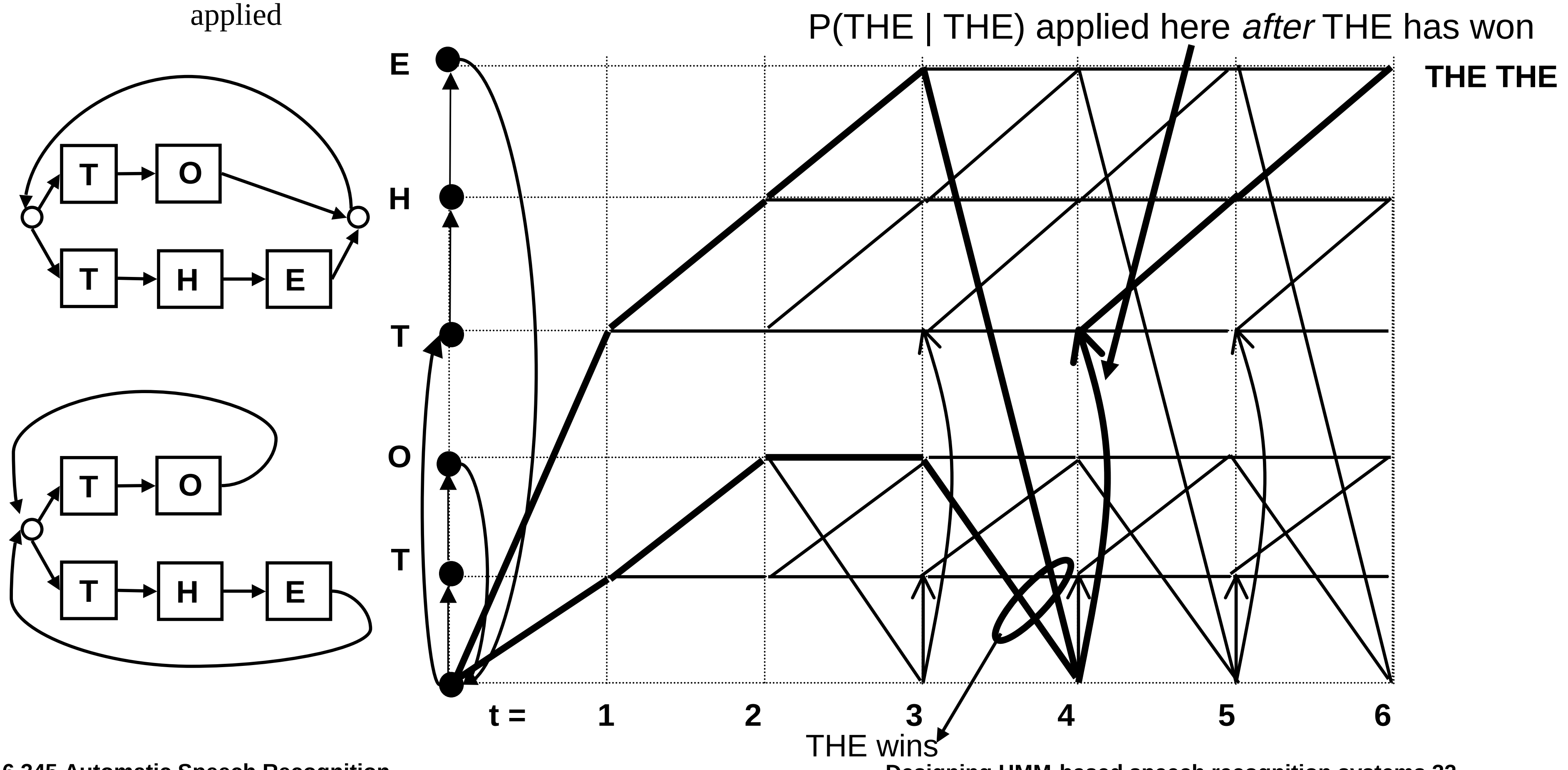
- Use a simple unigram structure
- Apply bigram probabilities
  - Context for bigram obtained from word history
  - Simpler structure needs less memory and computation
  - Imprecise

Pseudo-trigram from Unigram structure



Decoding with a unigram structure introduces errors

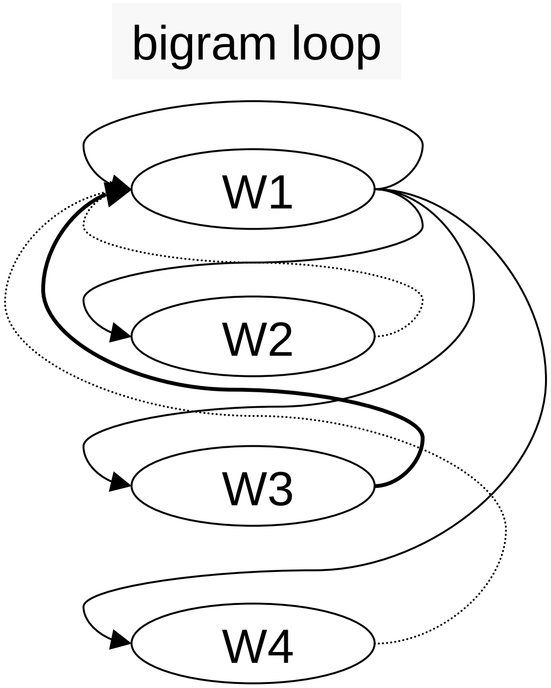
- Example with simplified HMMs
  - At t=4 THE competes with TO and wins. TO is no longer considered as a candidate first word on this path
  - The competition between THE and TO occurs before bigram probability P(THE|context) is applied.
  - P(THE|TO) may have been higher than P(THE|THE) and reversed the decision at t=4
  - However, the future word is unknown at the non-emitting node at t=4. Bigram probabilities could not be applied



Approximate decoding structures

Decoding using bigram structure

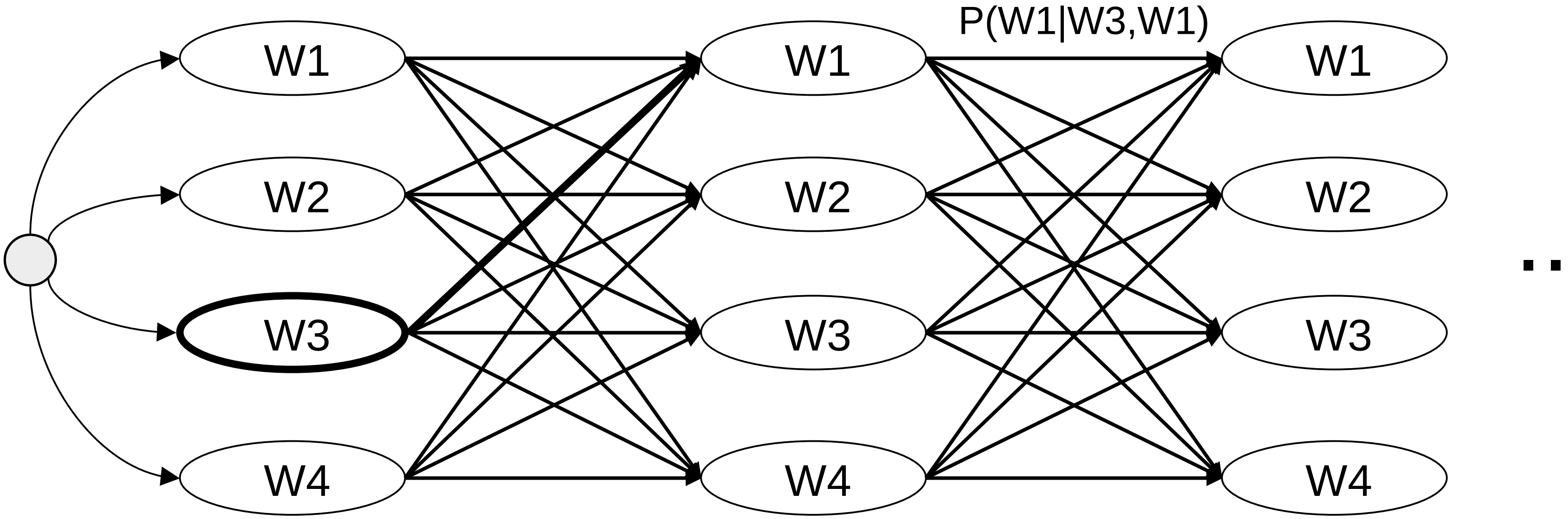
o Instead of a unigram structure, use a bigram structure. This is precise for bigrams but approximate for decoding with trigrams



Approximate decoding structures
pseudo-trigram decoding from bigram structure in Flat search

- Use a bigram structure
- Apply trigram probabilities
  - Context for trigram obtained from word history
  - Again, this is imprecise

Conventional Bigram Unrolled



Reducing computation further

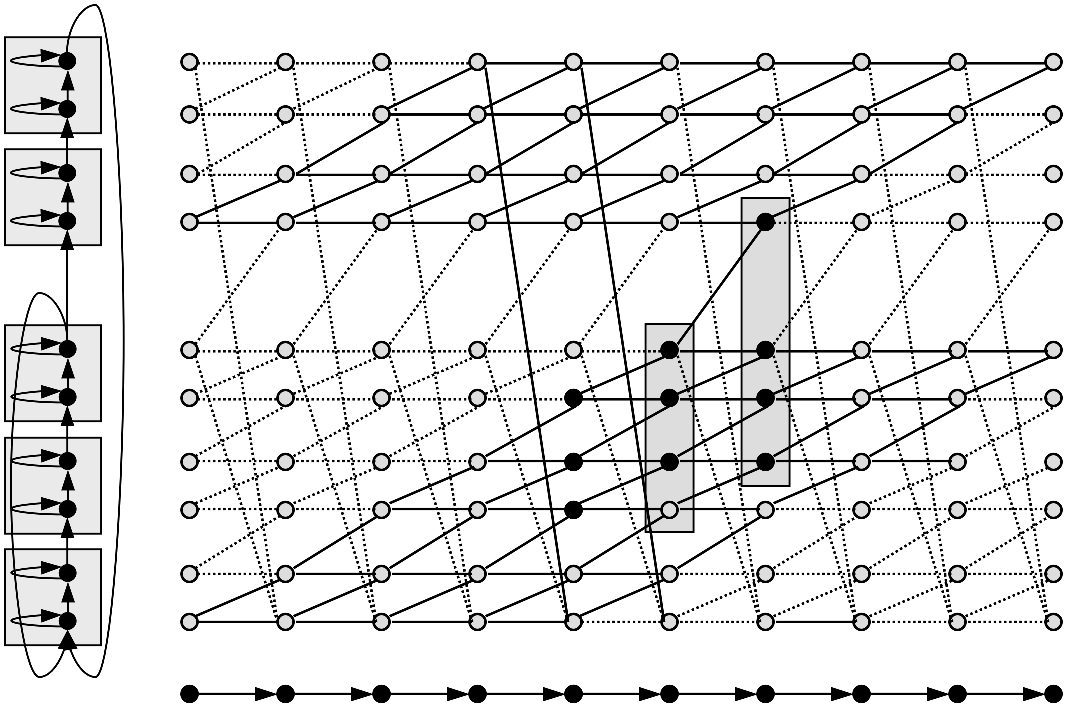
- Approximate structures are still large
  - Exhaustive search of all paths through them is prohibitive

- Search must be further constrained
  - Beam search

- Computation must be constrained
  - Gaussian selection

Constraining search for memory and speed: Beam Search

- At any time instant, paths that score above a threshold score are allowed to survive. The threshold score may be fixed (fixed beam search), or relative to the highest scoring path at that time instant (relative beam search). Thus beam search involves pruning out of low scoring paths.
- The nodes which are allowed to survive at any time comprise the active-list. Note that each node is an HMM state. Active lists are not always generated through direct score comparisons (that is slow). Many other methods are used.

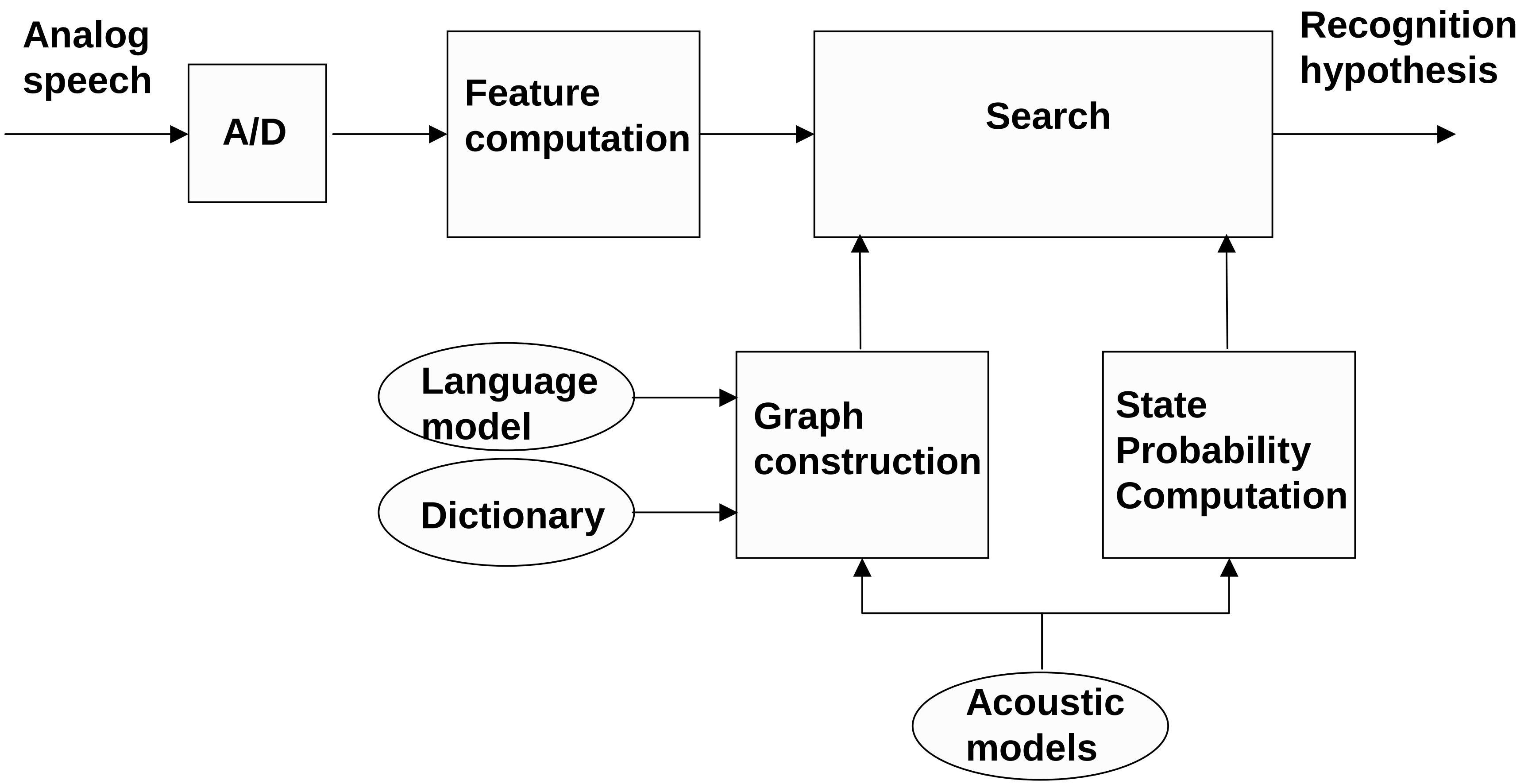


Relative Beam search

Constraining computation: Gaussian selection

- State probability densities are typically Gaussian mixtures
- Explicit computation of probabilities from all Gaussians present in the active list is expensive. A subset of these Gaussians is first selected based on some Gaussian selection algorithm, and then only those Gaussians are explicitly computed.
- Gaussian selection algorithms are based on
  - Prediction
  - Sharing of state densities
    - clustering
  - Pre-calculation of approximate scores from codebook for fast identification of best Gaussians
    - Suvector-quantization

Overall Decoder Architecture



Summary and conclusions

- We have discussed basic decoding issues
- We have discussed the construction of language HMMs for decoding
  - The dependence of the graph on the expected langauge
  - Statistical Ngram models and finite state grammars
- We have discussed some issues relating to graph size, memory requirement, computational requirements etc.
- This should place you in a position to comprehend the workings of most HMM-based speech recognition systems
  - And perhaps write a simple one yourself!

PART II

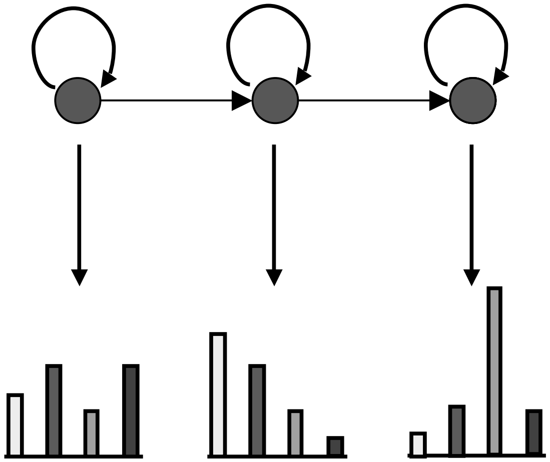
Training continuous density HMMs

Table of contents

- Review of continuous density HMMs
- Training context independent sub-word units
  - Outline
  - Viterbi training
  - Baum-Welch training
- Training context dependent sub-word units
  - State tying
  - Baum-Welch for shared parameters

Discrete HMM

- Data can take only a finite set of values
  - Balls from an urn
  - The faces of a dice
  - Values from a codebook
- The state output distribution of any state is a normalized histogram
- Every state has its own distribution



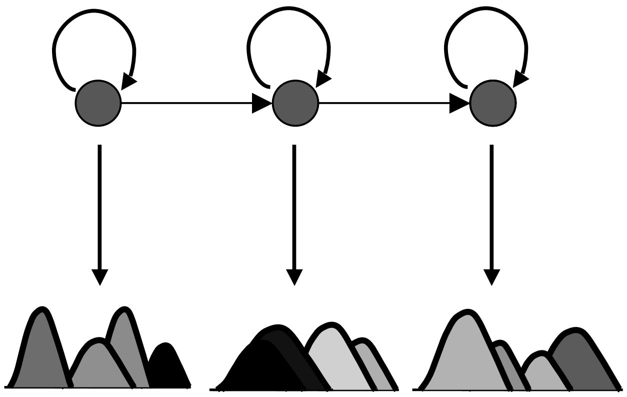
HMM that outputs one of four colors at each instant.

Each of the three states has a different probability distribution for the colors.

Since the number of colors is discrete, the state output distributions are multinomial

Continuous density HMM

- There data can take a continuum of values
  - *e.g.* cepstral vectors
- Each state has a state output *density*
- When the process visits a state, it draws a vector from the state output density for that state

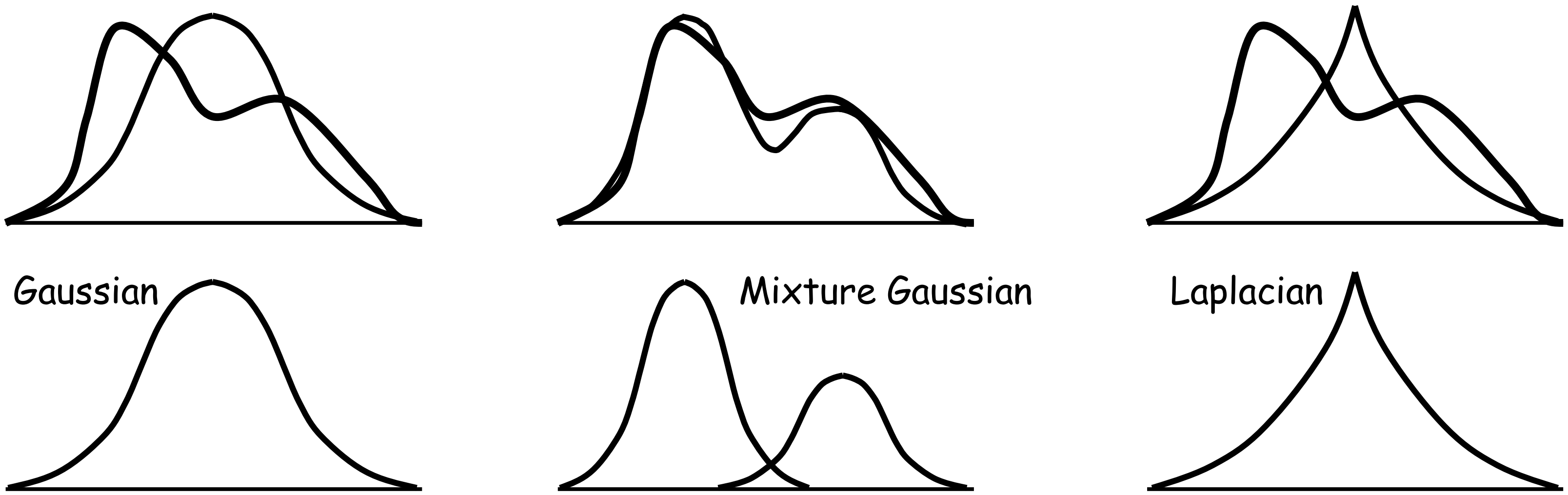


HMM outputs a continuous valued random variable at each state.

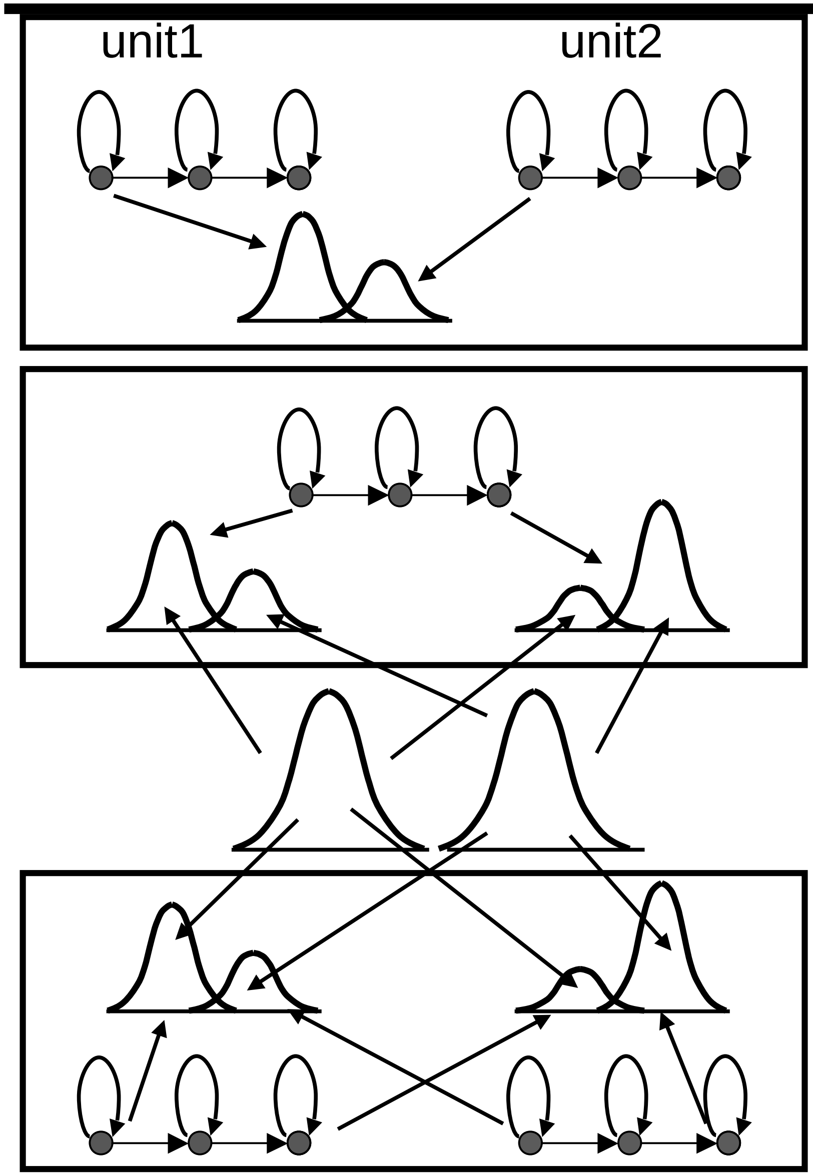
State output densities are mixtures of Gaussians. The output at each state is drawn from this mixture.

Modeling state output densities

- The state output distributions might be anything in reality
- We model these state output distributions using various simple densities
  - The models are chosen such that their parameters can be easily estimated
  - Gaussian
  - Mixture Gaussian
  - Other exponential densities
- If the density model is inappropriate for the data, the HMM will be a poor statistical model
  - Gaussians are poor models for the distribution of power spectra

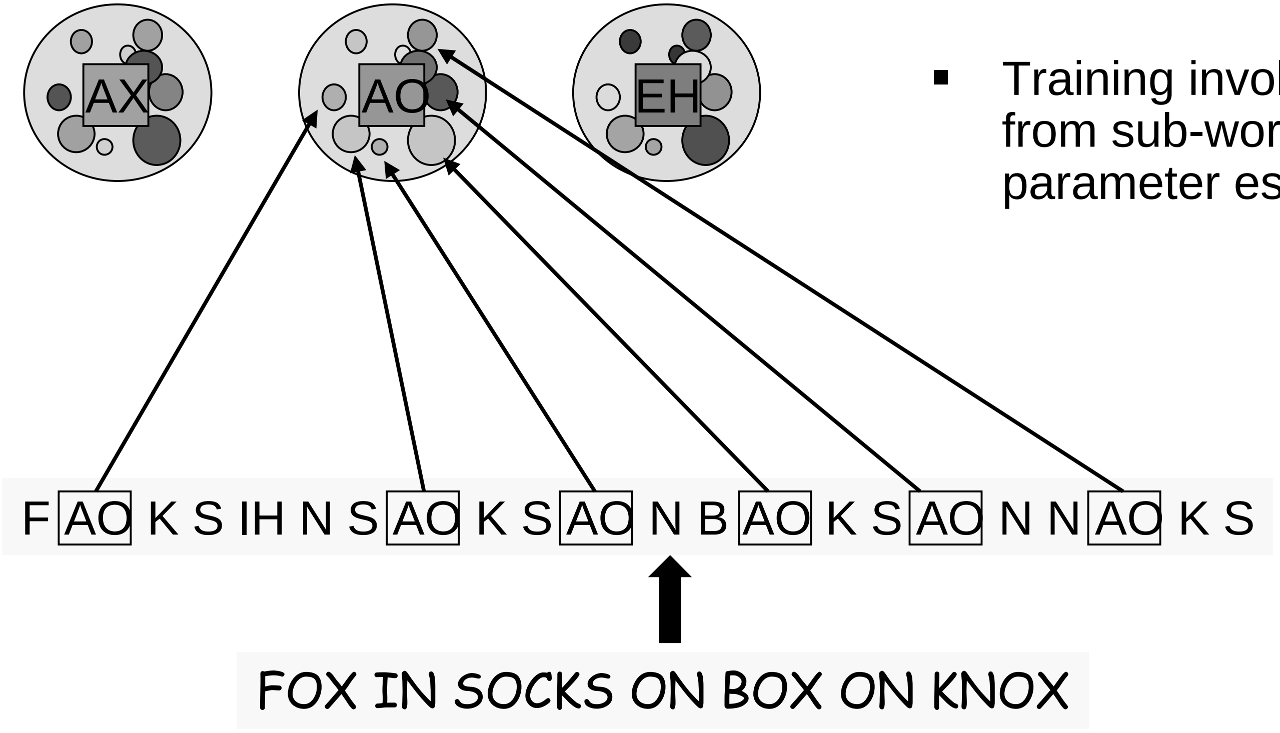


Sharing Parameters



- Insufficient data to estimate all parameters of all Gaussians
- Assume states from different HMMs have the same state output distribution
  - Tied-state HMMs
- Assume all states have different mixtures of the same Gaussians
  - Semi-continuous HMMs
- Assume all states have different mixtures of the same Gaussians and some states have the same mixtures
  - Semi-continuous HMMs with tied states
- Other combinations are possible

Training models for a sound unit

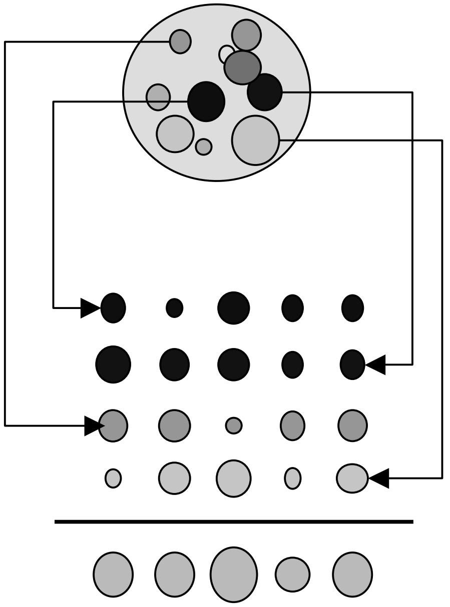


F AO K S IH N S AO K S AO N B AO K S AO N N AO K S

FOX IN SOCKS ON BOX ON KNOX

- Training involves grouping data from sub-word units followed by parameter estimation

Training models for a sound unit



For a 5-state HMM, segment data from each instance of sub-word unit to 5 parts, aggregate all data from corresponding parts, and find the statistical parameters of each of the aggregates

- Training involves grouping data from sub-word units followed by parameter estimation
- Indiscriminate grouping of vectors of a unit from different locations in the corpus results in Context-Independent (CI) models
- Explicit boundaries (segmentation) of sub-word units not available
  - We do not know where each sub-word unit begins or ends
- Boundaries must be estimated

Learning HMM Parameters

- Viterbi training
  - Segmental K-Means algorithm
  - Every data point associated with only one state

- Baum-Welch
  - Expectation Maximization algorithm
  - Every data point associated with every state, with a probability
    - A (data point, probability) pair is associated with each state

Viterbi Training

- 1. Initialize all HMM parameters
- 2. For each training utterance, find best state sequence using Viterbi algorithm
- 3. Bin each data vector of utterance into the bin corresponding to its state according to the best state sequence
- 4. Update counts of data vectors in each state and number of transitions out of each state
- 5. Re-estimate HMM parameters
  - State output density parameters
  - Transition matrices
  - Initial state probabilities
- 6. If the likelihoods have not converged, return to step 2.

Viterbi Training: Estimating Model Parameters

- Initial State Probability
  - Initial state probability $\pi(s)$ for any state *s* is the ratio of the number of utterances for which the state sequence began with *s* to the total number of utterances

$$\pi(s) = \frac{\sum_{utterance} \delta(state(1) = s)}{\text{No. of utterances}}$$

- Transition probabilities
  - The transition probability *a*(*s*,*s'*) of transiting from state *s* to *s'* is the ratio of the number of observation from state *s*, for which the subsequent observation was from state *s'*, to the number of observations that were in *s*

$$a(s, s') = \frac{\sum_{utterance} \sum_{t} \delta(state(t) = s, state(t+1) = s')}{\sum_{utterance} \sum_{t} \delta(state(t) = s)}$$

Viterbi Training: Estimating Model Parameters

- State output density parameters
  - Use all the vectors in the bin for a state to compute its state output density
  - For Gaussian state output densities only the means and variances of the bins need be computed
  - For Gaussian mixtures, iterative EM estimation of parameters is required within each Viterbi iteration

$$P(k\,|\,x) = \frac{P(k)P(x\,|\,k)}{\sum_j P(j)P(x\,|\,j)}$$

Posterior prob of kth Gaussian given observation x

P(x|k) is a Gaussian

Mean of kth Gaussian $$\mu_k = \frac{\sum_x P(k\,|\,x)x}{\sum_x P(k\,|\,x)}$$

Covariance of kth Gaussian $$C_k = \frac{\sum_x P(k\,|\,x)(x-\mu_k)(x-\mu_k)^T}{\sum_x P(k\,|\,x)}$$

Mixture weight of kth Gaussian $$P(k) = \frac{\sum_x P(k\,|\,x)}{\text{No. of vectors in bin}}$$

Baum-Welch Training

- 1. Initialize HMM parameters
- 2. On each utterance run forward backward to compute following terms:
  - $\gamma_{utt}(s,t)$ = *a posteriori* probability given the utterance, that the process was in state *s* at time *t*
  - $\gamma_{utt}(s,t,s',t+1)$ = *a posteriori* probability given the utterance, that the process was in state *s* at time *t*, and subsequently in state *s'* at time *t*+1
- 3. Re-estimate HMM parameters using gamma terms
- 4. If the likelihood of the training set has not converged, return to step 2.

Baum-Welch: Computing *A Posteriori* State Probabilities and Other Counts

- Compute α and β terms using the forward backward algorithm

$$\alpha(s,t\,|\,word) = \sum_{s'} \alpha(s',t-1\,|\,word)P(s\,|\,s')P(X_t\,|\,s)$$

$$\beta(s,t\,|\,word) = \sum_{s'} \beta(s',t+1\,|\,word)P(s'|\,s)P(X_{t+1}\,|\,s')$$

- Compute *a posteriori* probabilities of states and state transitions using α and β values

$$\gamma(s,t\,|\,word) = \frac{\alpha(s,t)\beta(s,t)}{\sum_{s'} \alpha(s',t)\beta(s',t)}$$

$$\gamma(s,t,\breve{s},t+1\,|\,word) = \frac{\alpha(s,t)P(\breve{s}\,|\,s)P(X_{t+1}\,|\,\breve{s})\beta(\breve{s},t+1)}{\sum_{s'} \alpha(s',t)\beta(s',t)}$$

Baum-Welch: Estimating Model Parameters

- Initial State Probability
  - Initial state probability $\pi(s)$ for any state *s* is the ratio of the *expected* number of utterances for which the state sequence began with *s* to the total number of utterances

$$\pi(s) = \frac{\sum_{utterance} \gamma_{utt}(s,1)}{\text{No. of utterances}}$$

- Transition probabilities
  - The transition probability *a*(*s*,*s'*) of transiting from state *s* to *s'* is the ratio of the *expected* number of observations from state *s* for which the subsequent observation was from state *s'*, to the *expected* number of observations that were in *s*

$$a(s,s') = \frac{\sum_{utterance} \sum_{t} \gamma_{utt}(s,t,s',t+1)}{\sum_{utterance} \sum_{t} \gamma_{utt}(s,t)}$$

Baum-Welch: Estimating Model Parameters

- State output density parameters
  - The *a posteriori* state probabilities are used along with *a posteriori* probabilities of the Gaussians as weights for the vectors
  - Means, covariances and mixture weights are computed from the weighted vectors

$$P(k \mid x_t, s) = \frac{P_s(k)P_s(x_t \mid k)}{\sum_j P_s(j)P_s(x_t \mid j)}$$

Posterior prob of kth Gaussian given observation x

P(x|k) is a Gaussian

Mean of kth Gaussian of state s

$$\mu_k^s = \frac{\sum_{utterance}\sum_t \gamma_{utt}(s,t)P(k \mid x_t, s)x_t}{\sum_{utterance}\sum_t \gamma_{utt}(s,t)P(k \mid x_t, s)}$$

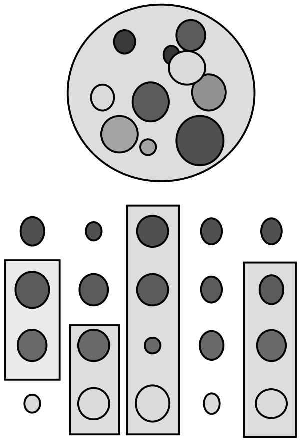
Covariance of kth Gaussian of state s

$$C_k^s = \frac{\sum_{utterance}\sum_t \gamma_{utt}(s,t)P(k \mid x_t, s)(x_t - \mu_k)(x_t - \mu_k)^T}{\sum_{utterance}\sum_t \gamma_{utt}(s,t)P(k \mid x_t, s)}$$

Mixture weight of kth Gaussian of state s

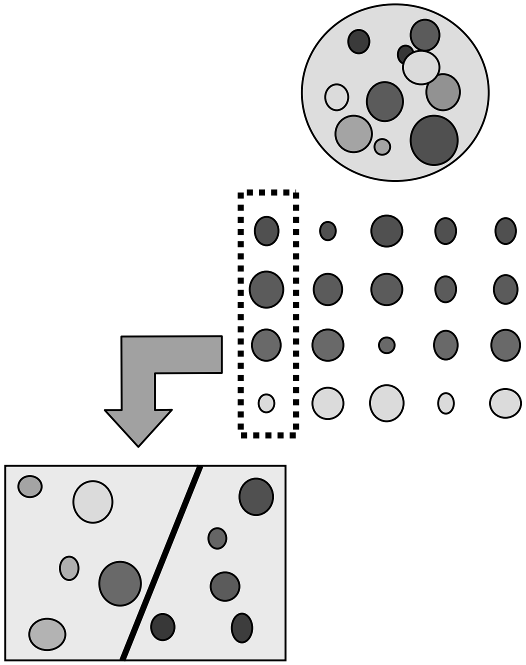
$$P_s(k) = \frac{\sum_{utterance}\sum_t \gamma_{utt}(s,t)P(k \mid x_t, s)}{\sum_{utterance}\sum_t\sum_j \gamma_{utt}(s,t)P(j \mid x_t, s)}$$

Training context dependent (triphone) models



- Context based grouping of observations results in finer, Context-Dependent (CD) models
- CD models can be trained just like CI models, if no parameter sharing is performed
- Usually insufficient training data to learn all triphone models properly
  - Parameter estimation problems
- Parameter estimation problems for CD models can be reduced by parameter sharing. For HMMs this is done by cross-triphone, within-state grouping

Grouping of context-dependent units for parameter estimation



- Partitioning any set of observation vectors into two groups increases the average (expected) likelihood of the vectors

The expected log-likelihood of a vector drawn from a Gaussian distribution with mean μ and variance ***C*** is

$$E\left[\log\left(\frac{1}{\sqrt{2\pi^{d}|C|}}e^{-0.5(x-\mu)^{T}C^{-1}(x-\mu)}\right)\right]$$

The assignment of vectors to states can be done using previously trained CI models or with CD models that have been trained without parameter sharing

Expected log-likelihood of a vector drawn from a Gaussian distribution

$$E\left[\log\left(\frac{1}{\sqrt{2\pi^d|C|}}e^{-0.5(x-\mu)^T C^{-1}(x-\mu)}\right)\right]=$$

$$E\left[-0.5(x-\mu)^T C^{-1}(x-\mu)-0.5\log\left(2\pi^d|C|\right)\right]=$$

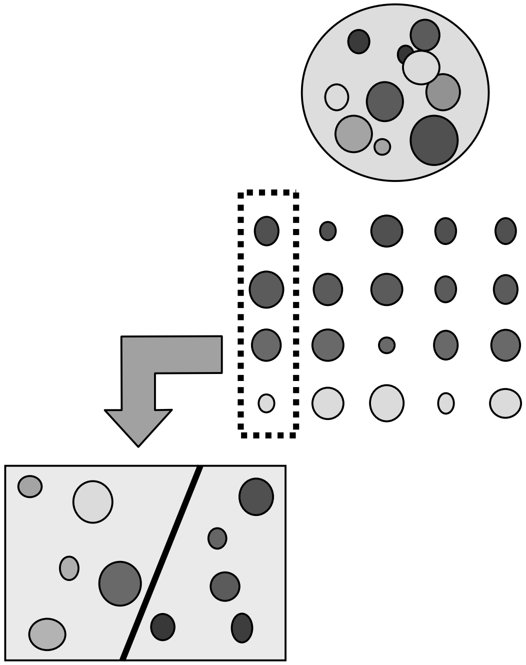
$$-0.5E\left[(x-\mu)^T C^{-1}(x-\mu)\right]-0.5E\left[\log\left(2\pi^d|C|\right)\right]=$$

$$-0.5d-0.5\log\left(2\pi^d|C|\right)$$

- This is a function only of the variance of the Gaussian
- The expected log-likelihood of a set of ***N*** vectors is

$$-0.5Nd-0.5N\log\left(2\pi^d|C|\right)$$

Grouping of context-dependent units for parameter estimation



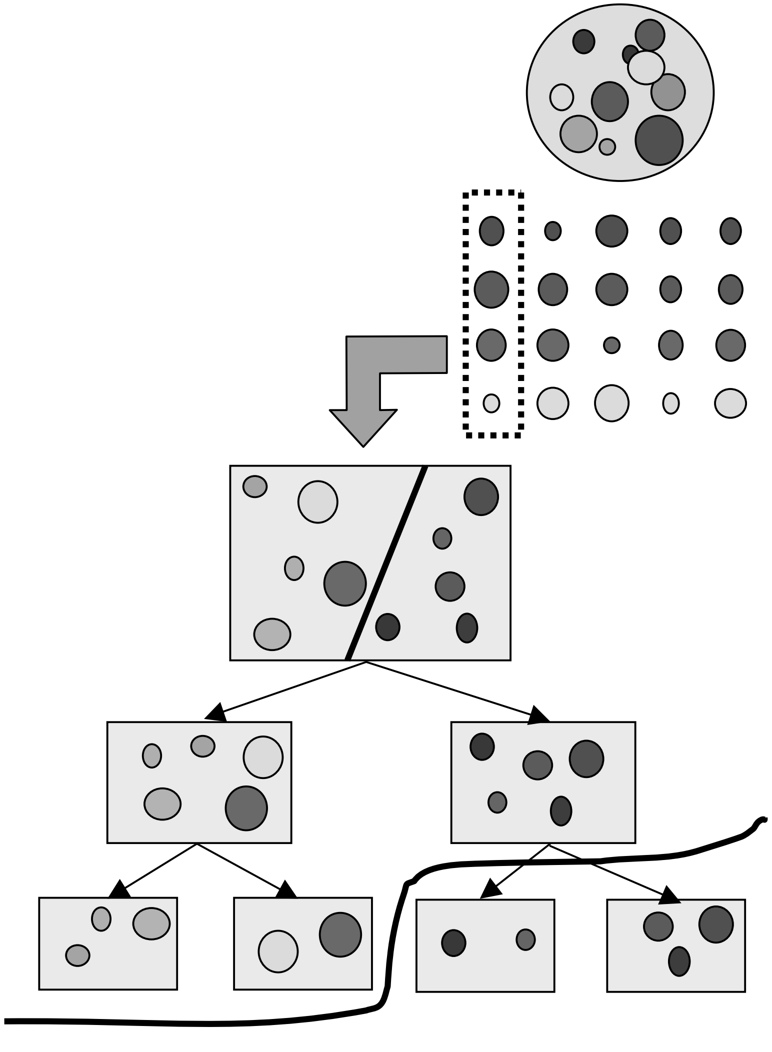
- If we partition a set of $\boldsymbol{N}$ vectors with mean μ and variance $\boldsymbol{C}$ into two sets of vectors of size $\boldsymbol{N_1}$ and $\boldsymbol{N_2}$, with means μ_1 and μ_2 and variances $\boldsymbol{C_1}$ and $\boldsymbol{C_2}$ respectively, the total expected log-likelihood of the vectors after splitting becomes

$$-0.5N_1d - 0.5N_1\log\left(2\pi^d\left|C_1\right|\right) - 0.5N_2d - 0.5N_2\log\left(2\pi^d\left|C_2\right|\right)$$

- The total log-likelihood has increased by

$$N\log\left(2\pi^d\left|C\right|\right) - 0.5N_1\log\left(2\pi^d\left|C_1\right|\right) - 0.5N_2\log\left(2\pi^d\left|C_2\right|\right)$$

Grouping of context-dependent units for parameter estimation



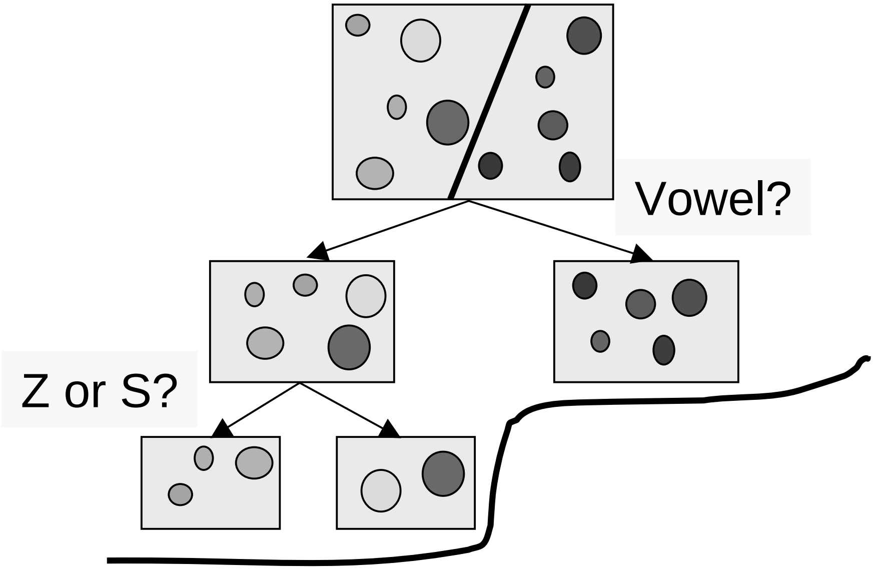
- Observation vectors partitioned into groups to maximize within class likelihoods
- Recursively partition vectors into a complete tree
- Prune out leaves until desired number of leaves obtained
- The leaves represent tied states (sometimes called *senones*)
    - All the states within a leaf share the same state distribution
- 2^{n-1} possible partitions for n vector groups. Exhaustive evaluation too expensive
- Linguistic questions used to reduce search space

Linguistic Questions

- Linguistic questions are pre-defined phone classes. Candidate partitions are based on whether a context belongs to the phone class or not
- Linguistic question based clustering also permits us to compose HMMs for triphones that were never seen during training (unseen triphones)

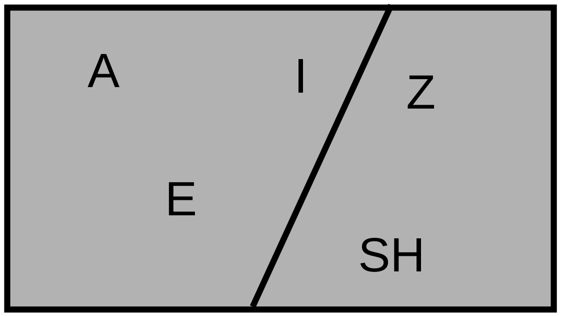
Composing HMMs for unseen triphones

- For every state of the N-state HMM for the unseen triphone, locate appropriate leaf of the tree for that state
- Locate leaf by answering the partitioning questions at every branching of the tree



Linguistic Questions

- Linguistic questions are pre-defined phone classes. Candidate partitions are based on whether a context belongs to the phone class or not
- Linguistic question based clustering also permits us to compose HMMs for triphones that were never seen during training (unseen triphones)
- Linguistic questions must be meaningful in order to deal effectively with unseen triphones



Meaningful Linguistic Questions?

Left context: (A,E,I,Z,SH)

ML Partition: (A,E,I) (Z,SH)

(A,E,I) vs. Not(A,E,I)

(A,E,I,O,U) vs. Not(A,E,I,O,U)

- Linguistic questions can be automatically designed by clustering of context-independent models

Other forms of parameter sharing

- Ad-hoc sharing: sharing based on human decision
  - Semi-continuous HMMs – all state densities share the same Gaussians
  - This sort of parameter sharing can coexist with the more refined sharing described earlier.

Baum-Welch: Sharing Model Parameters

- Model parameters are shared between sets of states
  - Update formulae are the same as before, except that the numerator and denominator for any parameter are also aggregated over all the states that share the parameter

Mean of kth Gaussian
of any state in the set of states Θ
that share the kth Gaussian

$$\mu_k^{\Theta} = \frac{\sum_{s\in\Theta}\sum_{utterance}\sum_{t}\gamma_{utt}(s,t)P(k\,|\,x_t,s)x_t}{\sum_{s\in\Theta}\sum_{utterance}\sum_{t}\gamma_{utt}(s,t)P(k\,|\,x_t,s)}$$

Covariance of kth Gaussian
of any state in the set of
states Θ that share the kth
Gaussian

$$C_k^{\Theta} = \frac{\sum_{s\in\Theta}\sum_{utterance}\sum_{t}\gamma_{utt}(t,s)P(k\,|\,x_t,s)(x_t-\mu_k)(x_t-\mu_k)^T}{\sum_{s\in\Theta}\sum_{utterance}\sum_{t}\gamma_{utt}(t,s)P(k\,|\,x_t,s)}$$

Mixture weight of kth Gaussian
of any state in the set of states Θ that
share a Gaussian mixture

$$P_{\Theta}(k) = \frac{\sum_{s\in\Theta}\sum_{utterance}\sum_{t}\gamma_{utt}(t,s)P(k\,|\,x_t,s)}{\sum_{s\in\Theta}\sum_{utterance}\sum_{t}\sum_{j}\gamma_{utt}(t,s)P(j\,|\,x_t,s)}$$

Conclusions

- Continuous density HMMs can be trained with data that have a continuum of values
- To reduce parameter estimation problems, state distributions or densities are shared
- Parameter sharing has to be done in such a way that discrimination between sounds is not lost, and new sounds are accounted for
  - Done through regression trees
- HMMs parameters can be estimated using either Viterbi or Baum-Welch training