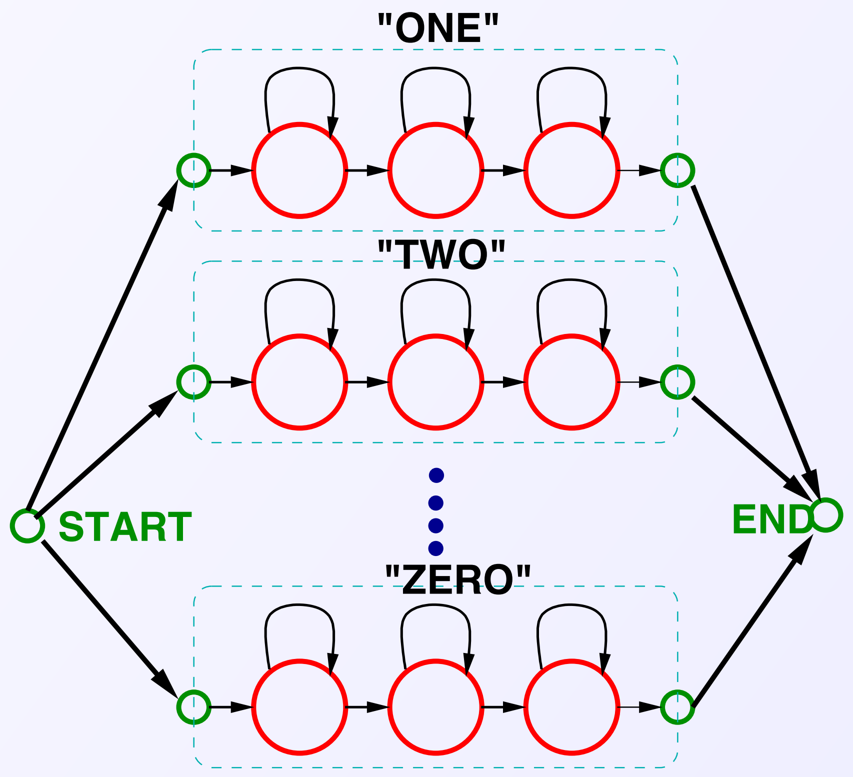
COM4511/COM6511 - Speech Technology

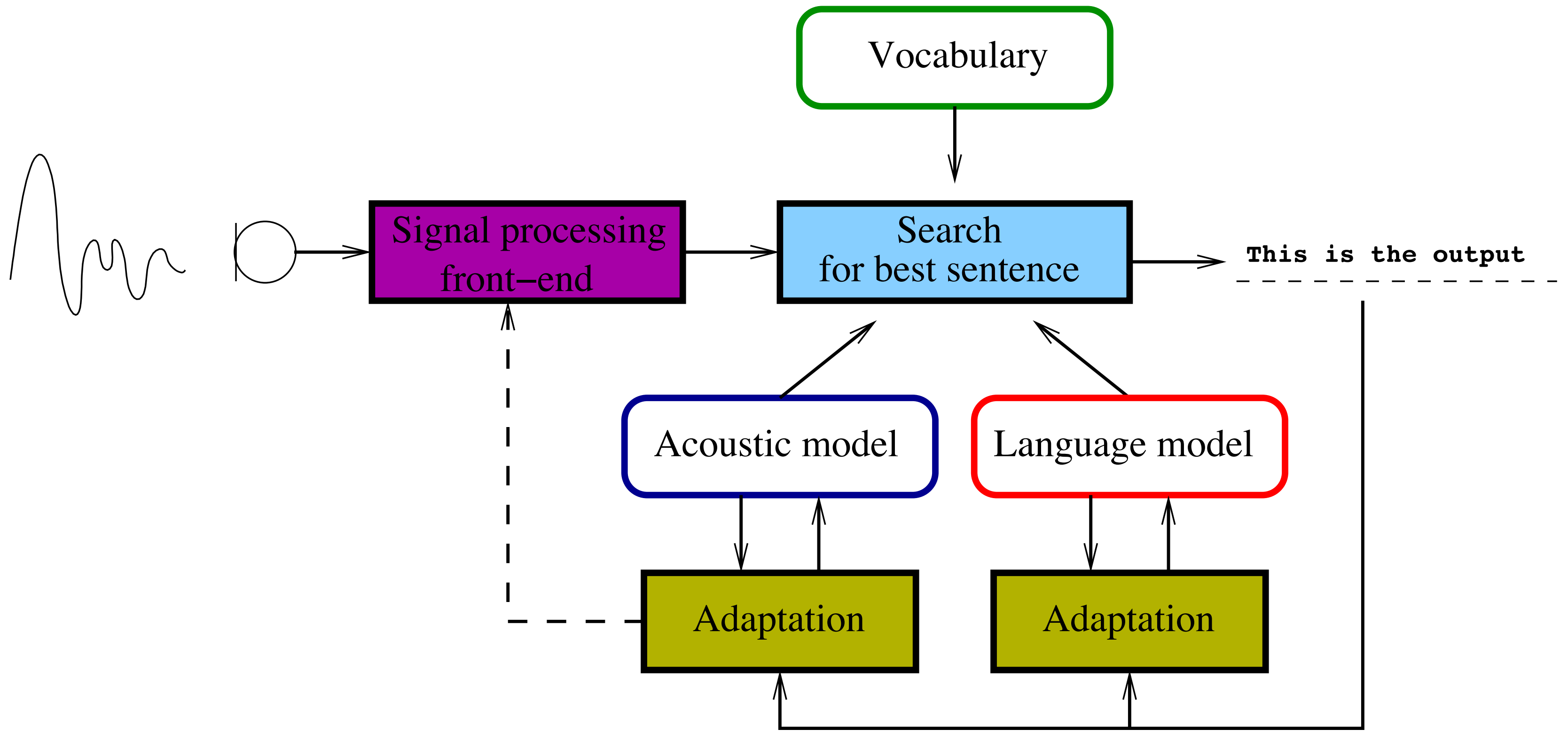
Lecture 12
Search



Thomas Hain
t.hain@sheffield.ac.uk
Spring Semester

Search

Recall: A search for the best word sequence is to be conducted using acoustic and language models.



We now have more precise information about the knowledge sources that are required for speech recognition. We have identified clustered **triphones** as standard for acoustic models and **N-grams** for language models. Further we have introduced how to use word lists and dictionaries. Now we have tot **search** for best word sequence given the observed signal, using all knowledge sources simultaneously !

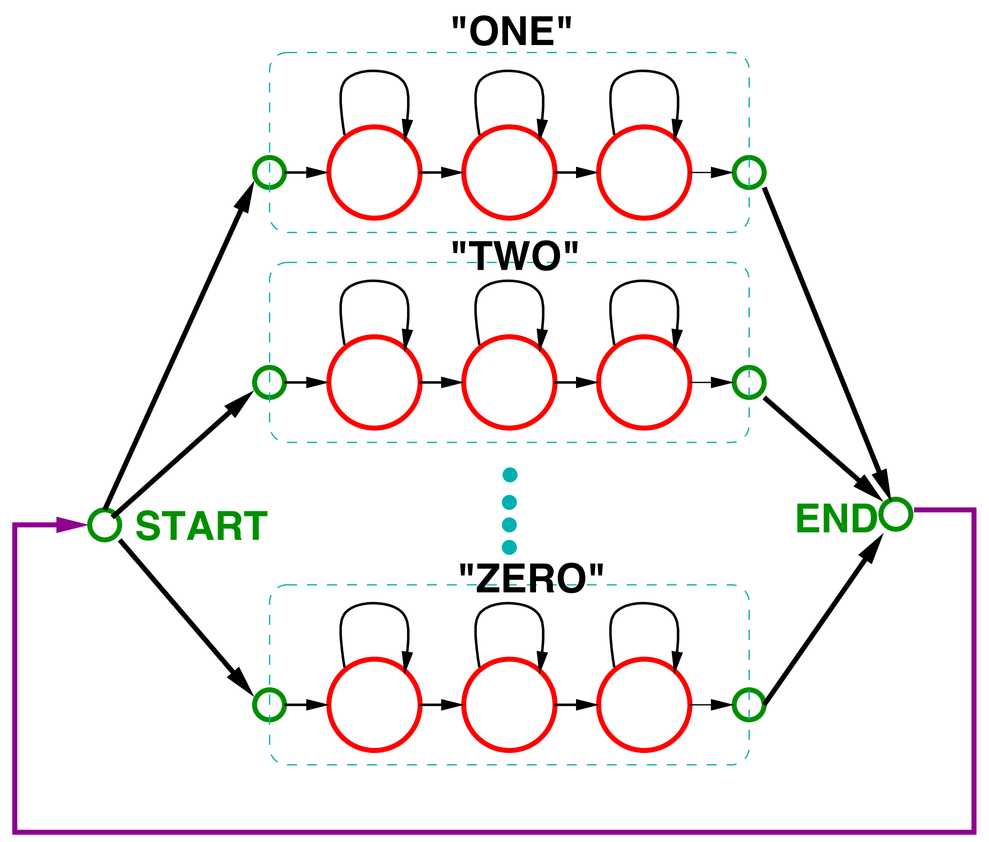
The optimal decoder

An implementation of a search algorithm is also called a decoder (”decoding” of signals into text). We can distinguish different search methodologies (e.g. depth first vs. breadth first) and algorithmic differences. Evidently a “good” implementation of a search algorithm will be usable on a variety of platforms. The following properties allow to categorise speech decoders:

- ➜ **Efficient:** The computational resources required by the decoder must be reasonable given the task. **“Faster than real-time”** performance is often required.
- ➜ **Accurate:** The decoder should always find the word sequence given the knowledge sources. This is not (!!!) identical to getting the correct sequence of words. All deviations from that are **search errors**.
- ➜ **Scalable**: The decoder should handle arbitrary vocabulary.
- ➜ **Versatile:** Ideally the decoder should be able to handle a large variety of knowledge sources.
- ➜ **Flexible Output:** The decoder should not only be capable of producing the “best” word sequence, it should also be possible to include alternative results.

Viterbi for continuous speech

Recall: We can use the Viterbi algorithm to determine the **most likely state sequence**. If a composite model for continuous speech is used the Viterbi algorithm implicitly yields the best sequence of words. If the state sequence is not required, to recover the identity of the component (word) models only the points where external transitions from the end state of each model is taken, need to be recorded.

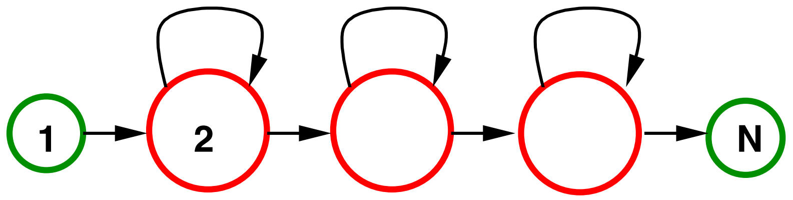


Composite HMM for recognition of connected digits

The composite model as seen on the left does not integrate language models or state-of-the-art acoustic modelling techniques. Modifications to the search network can be made to integrate advanced techniques from both knowledge sources. The Viterbi algorithm allows **time-synchronous** operation. An efficient implementation of the Viterbi algorithm is in the form of **token-passing**.

Token Passing

Recall the **Viterbi algorithm** for a single HMM with N states:



At each point (j, t) in the trellis we compute

$$\phi_j(t) = \max_{1 \leq i < N} \left[\phi_i(t-1) + \log\left(a_{ij}\right)\right] + \log\left(b_j\left(\mathbf{o}_t\right)\right)$$

In the final state:

$$\phi_{\max} = \max_{1 < i < N} \left[\phi_i(T) + \log\left(a_{iN}\right)\right]$$

In order to retain the best state sequence pointers to the best predecessor have to be stored. This is called **back-trace information**, the process of recovering the best state sequence is called **trace-back**. For decoding it is not necessary to store back-trace information for every state if a full state alignment is never required. Instead all that we will record is can the start time of the current model.

Each state of the composite HMM holds one **token**. The token q has two components. Tokens are propagated through the HMM.

q_j.prob	log likelihood, i.e. $\phi_j(t)$
q_j.wlink	a pointer to the complete word history, all preceding words and their start times.

Token Passing - Algorithm

The token passing algorithm can be implemented as follows:

Initialisation

Store token with q.prob $= 0$ in state 1 of every model
Store token with q.prob $= -\infty$ (i.e. $\log 0$) in all other states

Recursion

```
for (each frame: $t = 1;\ t \leq T\ ;\ t{+}{+}$ ){
    for (each model: $m = 1;\ m \leq M\ ;\ m{+}{+}$ ){
        for (each model state: $j = 2;\ j < N\ ;\ j{+}{+}$ ){
            wordinternal_token_propagation
        }
        best_exit_token
    }
    exit_token_propagation
}
```

Token Passing - Token propagation

`wordinternal_token_propagation`

➜ find the predecessor state i in model m for which $\mathrm{q}_i.\mathrm{prob} + \log(a_{ij})$ is maximum

➜ copy that token into state j

➜ increment $\mathrm{q}_j.\mathrm{prob}$ by $\log(a_{ij}) + \log(b_j(\mathbf{o}_t))$

`best_exit_token`

➜ find the predecessor state i in model m for which $\mathrm{q}_i.\mathrm{prob} + \log a_{iN}$ is maximum

➜ copy that token into state N

➜ increment $\mathrm{q}_i.\mathrm{prob}$ by $\log(a_{iN})$

Note that the propagation of tokens must be done simultaneously. On completion, the model holding the token in state N with the highest likelihood identifies the unknown speech. The states 1 and N are treated differently since they are non-emitting, but allow a set of models to be joined together.

Token Passing - Continuous Speech

The utility of the **token passing** approach is that it extends naturally to the continuous model case. The above algorithm must be extended to

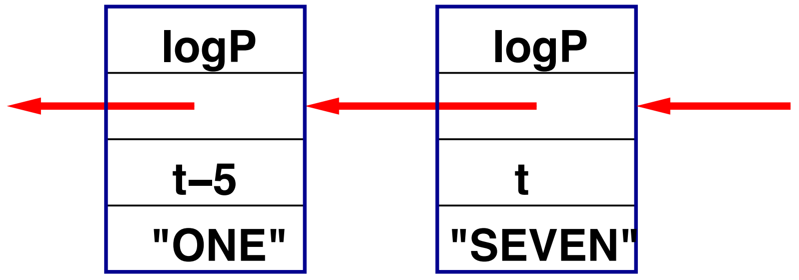
- ➜ **identify** the **best** token in the final states of all models
- ➜ allow the best token to **propagate** around the loop **into state 1** of all the connected models.
- ➜ **store** the **identity** of the model which generated that token **and** its **history**

The information necessary for traceback is stored in the form of word link records (WLRs) that are organised in linked lists.

WLR

$w.q$	a copy of q that is present in the final node
$w.\mathrm{time}$	the current time
$w.\mathrm{word}$	the word associated

For example



Token Passing - Between-word token propagation

For best token search, exit token propagation and storage of trace-back information the following code is to be used:

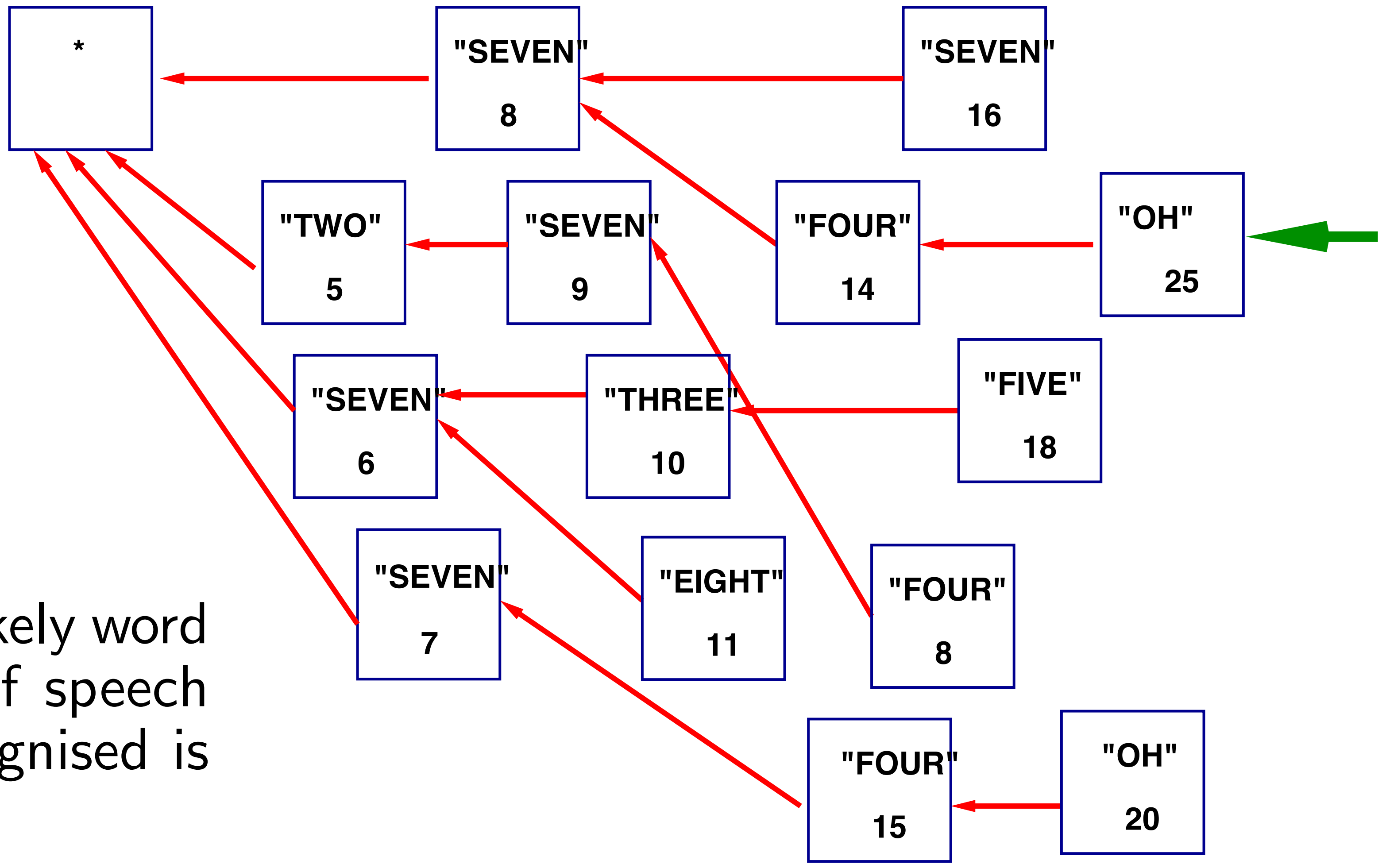
```
exit_token_propagation

➜ find the model $m$ that holds the token $q$ with the highest
  probability in the final state $N$.
➜ Create a WLR holding
  1. a copy of that token $q$
  2. the current time
  3. the word ID $m$
➜ Update the token
          $q$.start = t+1
          $q$.prob = $q$.prob + P
➜ copy $q$ into state 1 of all models
```

Here P is a constant inter-model **log** transition probability (insertion penalty) which can be independently adjusted to control the rate of insertion errors.

Token passing - Back-trace information

Note that the above procedure will generate a new WLR at each time t. The complete set of WLRs represents the back-trace information in recognition. The diagram below shows WLR records with associated words and word-end times.



The word "*" denotes the start of a sentence, the red arrows are holding the back-trace information. Each WLR was created whenever a decision between words had to be taken (in the case of the connected digit recogniser this is the end node).
The green arrow shows the most likely word after processing the final frame of speech data. Evidently the sentence recognised is "SEVEN FOUR OH".

Computational cost

For W words, Nstates/word (on average), for each time frame (10ms typically) we need to perform

1. $W \times N$ internal token propagations
2. W external token propagations.

Each internal propagation involves

1. Output probability calculation
2. 2 or 3 *add and compare* steps per state

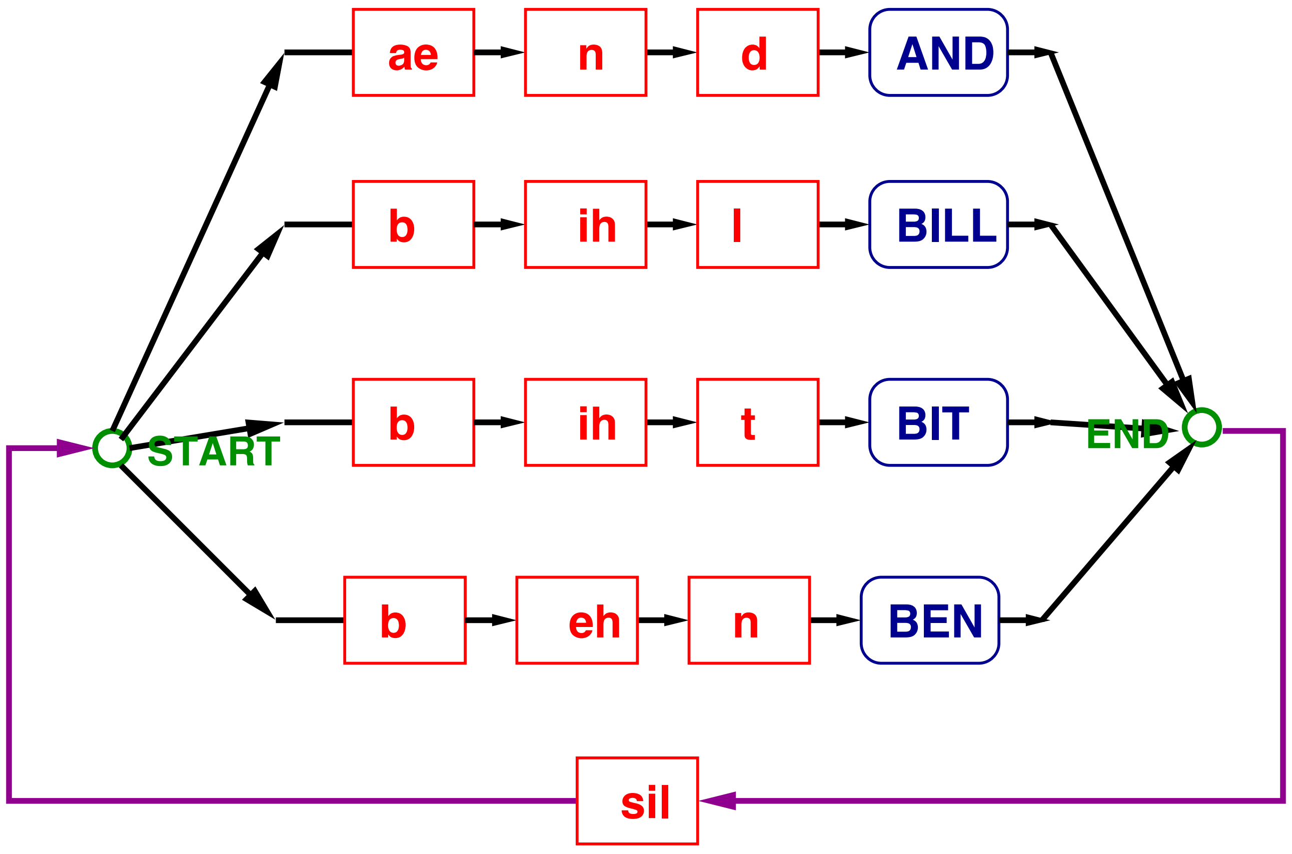
Normally the execution time for internal propagation is dominated by the state output probability calculations. For example if we have in total K output distributions with M Gaussians each and a vector size V this requires $K \times M \times V \times 2$*multiply/add* per frame. For a relatively low complexity system ($K = 2000$,$M = 10$,$V = 39$) this means approximately 1.5 million *multiply/add* operations per 10ms frame! In addition the modelling techniques discussed in earlier lectures have a considerable impact .....

Monophone models

Acoustic modelling has an impact on the construction of recognition networks and thus on the search space. Monophones are a simple acoustic models that allow the construction of word models from phone models.

The following examples use a limited vocabulary to illustrate the effects of techniques on search networks. The following dictionary is used.

AND	ae n d
BILL	b ih l
BIT	b ih t
BEN	b eh n

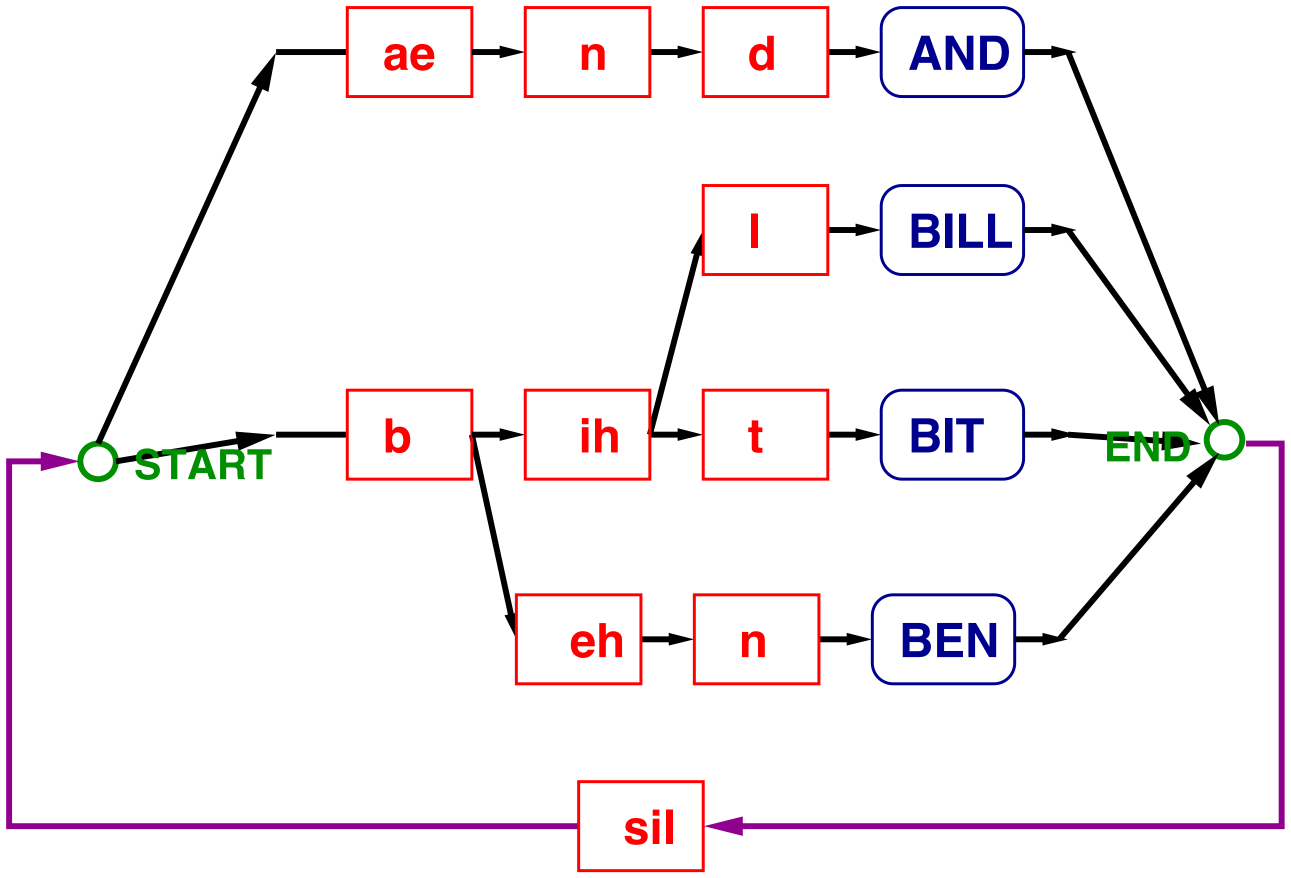


The diagram on the right shows the search network for this vocabulary. Each phone model consists of 3 states. Additional **word-end nodes** were added.

Tree organisation of pronunciations

The parallel organisation of the dictionary as outlined in the previous slide is suboptimal. For example many words start with the phoneme b and consequently use the same model at the start. Thus the parallel branches can be merged.

The reduction in the number of models in the network has a direct impact on the computational complexity. For the token passing algorithm each state in the network has to hold exactly one token. Each token requires propagation and thus incurs computational cost.
By reducing the number of models in the network by 3 we have removed 9 states from the network (there are $10 \times 3 = 30$ emoting states left in this network)

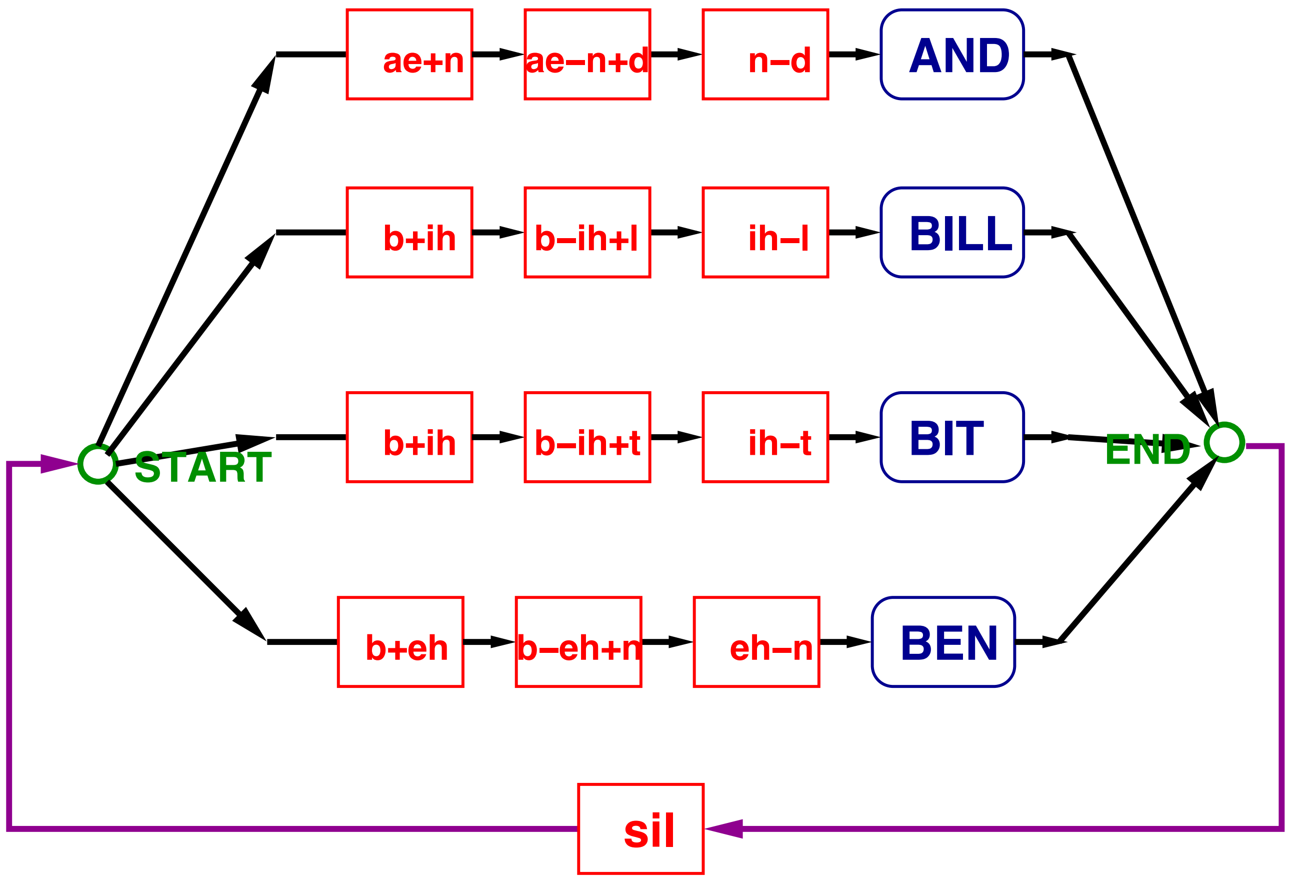


Triphones (Word-internal)

Monophone models normally have inferior performance to triphone models. In the case of word-internal triphones the expansion of the above network (non-tree organised) is trivial.

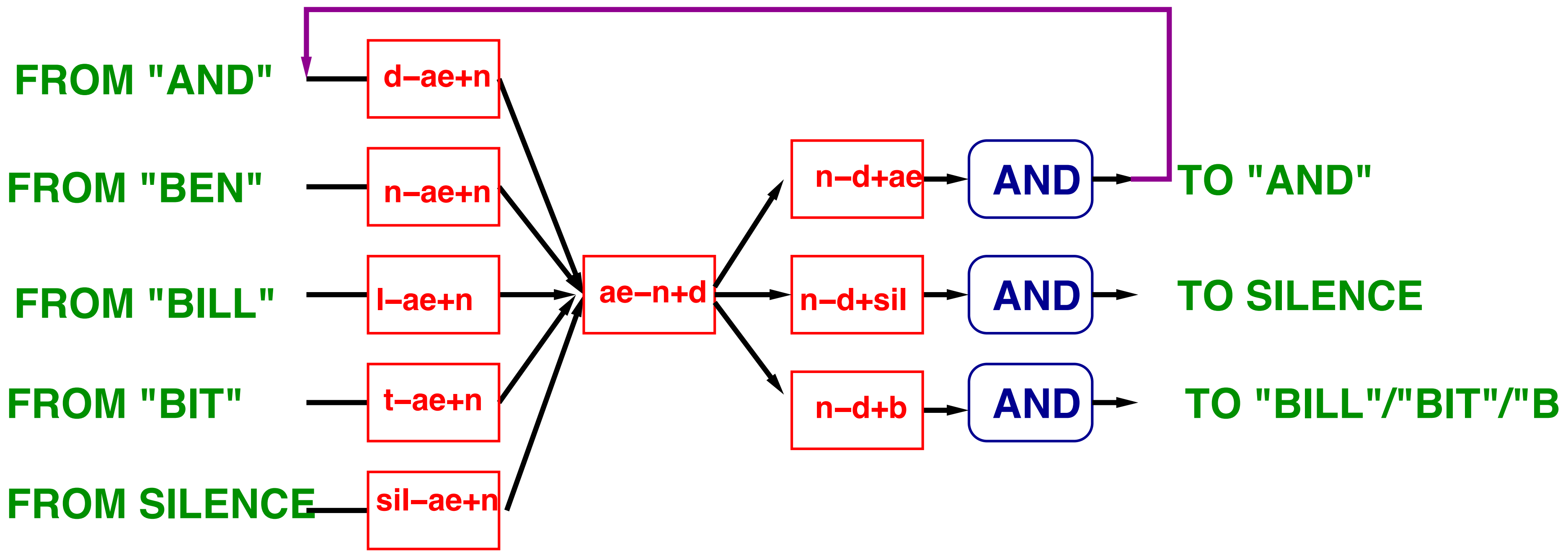
As only the model names changed the impact on the cost of search is virtually nothing (there is an indirect impact as the triphones are unlikely to share the same output distributions and thus more output probabilities have to be computed).

Can you outline the impact of using word internal triphone models with pronunciation trees ?



Triphones (Cross-word)

Word internal triphone models have several disadvantages (see handout 2). The diagram below outlines the effect of cross-word context on the part of the network associated with the word "AND" alone (connections are only indicated):



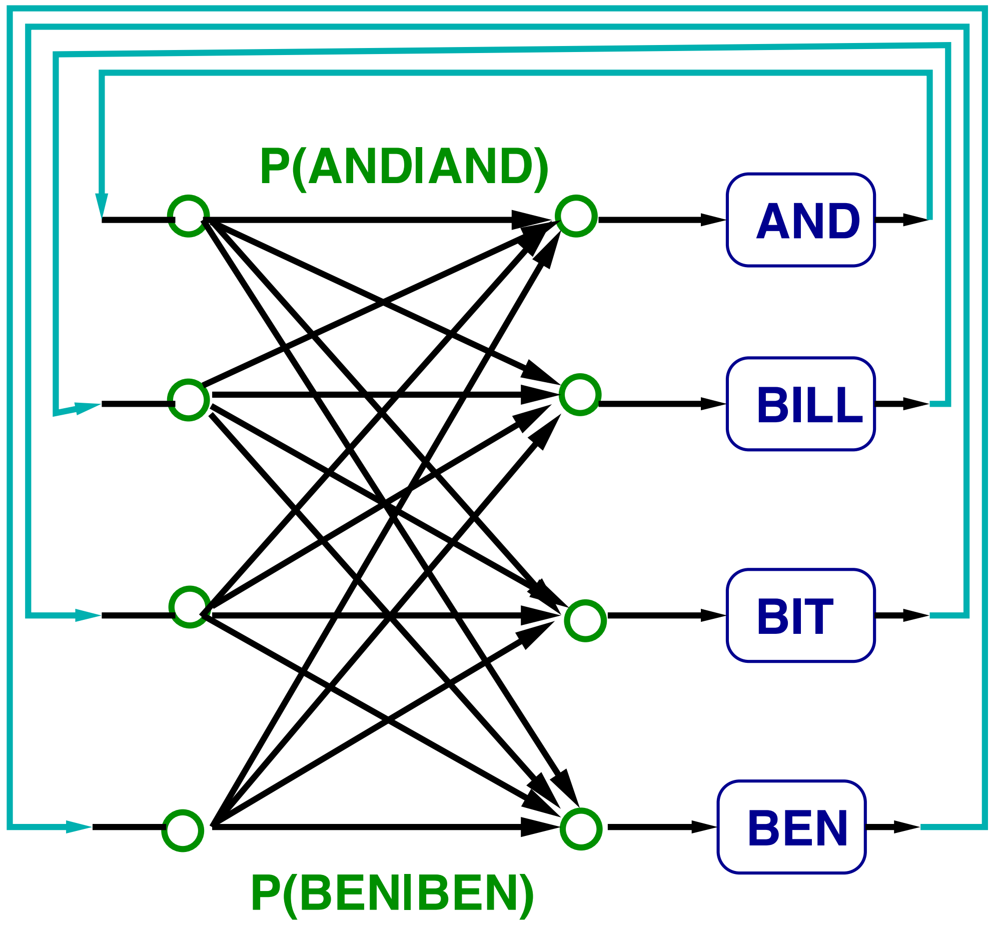
Note that word-end nodes have been tripled due to the 3 possible start phones. This means that word-end nodes are not connected to all words at the front of the loop ! Hence the token passing algorithm has to be modified. The "best word" decision is now to be made for each set of links entering one particular word !

The search space is dramatically increased !

Bigram networks

Similar to the use of triphone models, the use of language models adds contextual information to the network and hence increases the search space. The diagram below shows the modifications to the network for bigram recognition. For reasons of simplification word models and no inter-word silence model is used in the illustration.

- ➜ Again the word-end nodes are duplicated
- ➜ The language model probabilities are part of the search network ! They can be interpreted as transition probabilities in a huge composite model.
- ➜ A similar structure is theoretically constructible for **trigram models**. However in practice the networks become vast (typically several hundred Megabytes), thus other implementations have to be used.



Beam search

As discussed before straight-forward use of Viterbi is difficult even without the added effects of crossword modelling or complex language models.

Thus appropriate methods for constraining the search need to be found. This implies that the best word sequence may not be obtained.

Beam search is a standard technique to limit the computational cost while retaining reasonable performance. The basic idea is to only compute likelihoods of those word sequences close (in likelihood) to the best path. The removal of other paths is called **pruning**. Pruning inevitably introduces **search errors** due to suboptimal decisions.

For beam-search two steps have to be added to the search algorithm:

- **Highest likelihood:** The highest likelihood $L_{\max}(t)$ of all tokens q_t in the complete (!) network at each time instance t has to be determined.
- **Pruning:** All tokens with a likelihood below $L_{\max}(t) - L_{\text{thresh}}$ are removed from the network (i.e. the states are marked as inactive). If complete models to not have any active states they are marked as inactive. The threshold L_{thresh} can be selected to yield the desired speed/performance relationship

Alternatives/Improvements to Viterbi search

- **Fast match procedures**
  The search procedure can be greatly enhanced by looking forward into the “future”, i.e. a few frames ahead to make local decisions on pruning.
- **Stack-decoders**
  Stack decoders run **time-asynchronously** and allow the seamless integration of high context depths for language and acoustic models. Stack decoders have been popular for that reason. However, pruning is difficult in this framework and leads to substantial search errors. A particular variant is A^*search.
- **Multi-pass search**
  The basic idea is to decode the data multiple times with models of varying complexity. For example monophone models can be used to produce a word network (**lattice**) of the words that are most probably associated with that particular utterance. This word network then is used as a constrained search space for use with higher complexity models such as for example crossword triphones and 4-gram language models.