MPhil in Machine Learning and Machine Intelligence

Module MLMI2: Speech Recognition

L4: Viterbi Algorithm in Recognition & Training

Phil Woodland
pcw@eng.cam.ac.uk

Michaelmas 2021

Cambridge University Engineering Department

Introduction

This lecture will discuss more details of the Viterbi algorithm and how it is applied in **recognition**, including an outline of the computations required.

This includes:

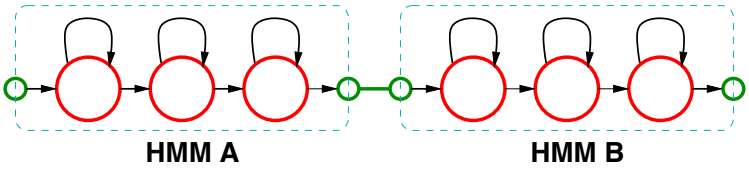
- Composite HMMs and generator models
- Recognition using a composite model
  - Isolated unit recognition
  - Continuous unit recognition
- Viterbi algorithm (revision) & Paths
- Token passing implementation approach
- Traceback
- Pruning
- Partial traceback (Continuous operation)

We will then discuss the use of Viterbi algorithm in **parameter estimation**

- Viterbi Training

Composite HMMs

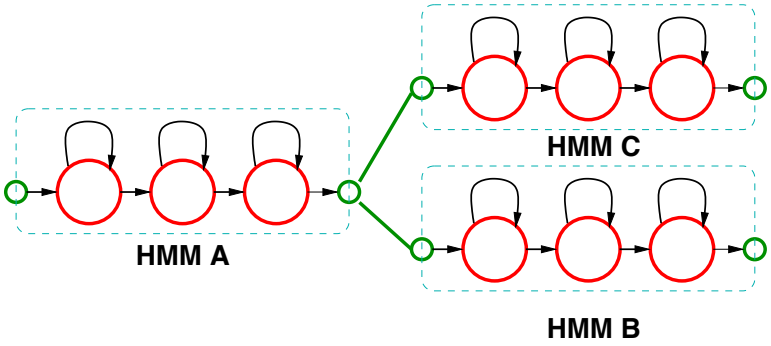
One important advantage of HMMs (& the use of non-emitting entry/exit states) is that larger HMMs can be constructed by the **composition** of smaller ones:



The use of unique entry and exit states allows the simple concatenation of HMMs.

This allows:

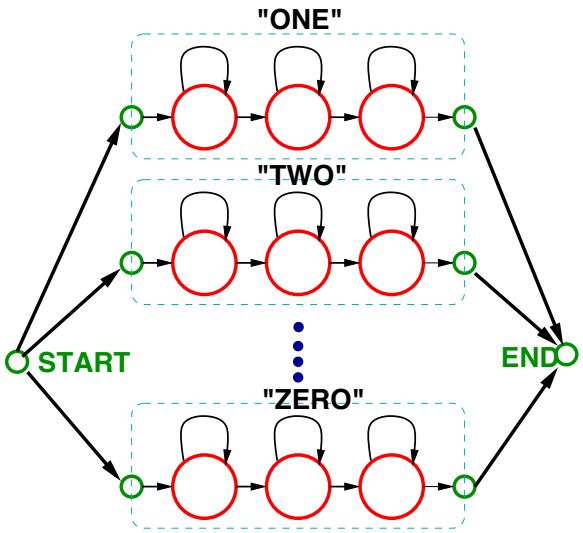
- word models from sub-word units
- sentence models from word models (used in training and recognition ...)



Note that probabilities may be assigned to the transition between HMMs.

Recognition with a Composite Model

The Viterbi algorithm finds the likelihood, along with the optimal state sequence, that the HMM generated an unknown utterance observation.

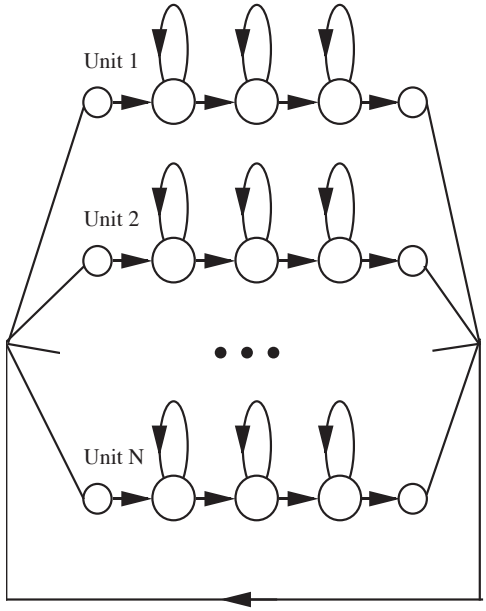


- For the isolated word case we can build a composite model by joining all the start states and all the end states of all possible word models together.
- For **isolated word recognition** find the optimal state sequence through the composite model and hence find the single word model that was most likely to have generated the input speech.

Composite HMM for recognition of isolated digits

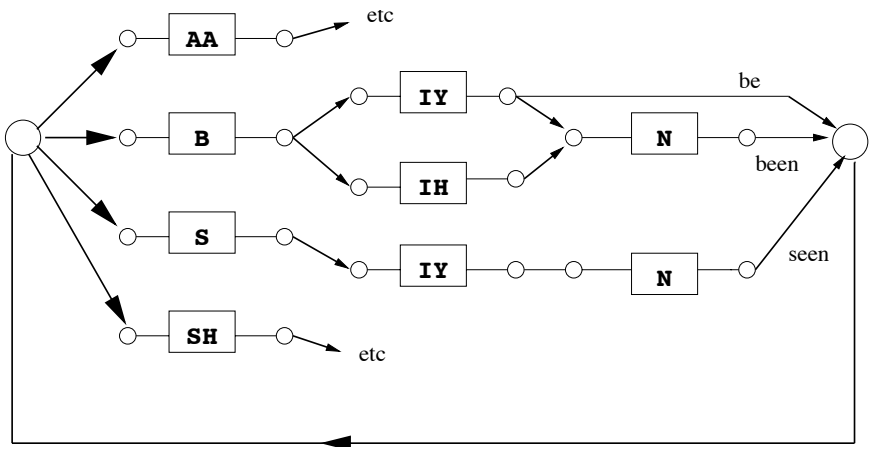
Application to Continuous Speech Recognition

- In continuous speech recognition the location of unit boundaries (words, phones etc.) is **unknown**. Need to allow for boundaries at each frame.
- Composite HMMs can be created as **sentence generators**.



- The best **path** through this composite HMM then defines the recognised sequence.
- Best path can be found using the Viterbi algorithm, can recognise continuous speech!
- Note that probabilities may be assigned to the transition between HMMs
  - Allows language model to be directly included in recognition network
  - Can use a simple "penalty" between each unit (control insertions)

Simple Word Loop & Sub-Word Units



- Here, basic phone HMMs are used with a dictionary to build word models
- Word models connected in a loop: HMMs **duplicated** when the **full network is expanded**
  - we talk of **model instances**
- Network transitions may have probabilities attached to them.
- In general, **all models** (basic acoustic model HMMs, dictionary, language model) **combined** into the complete network, and best path found

Viterbi Algorithm (revision)

The **most likely HMM state sequence** through trellis may be found by **Viterbi Algorithm**.

It computes

$$\hat{p}(\mathbf{O}|\lambda) = \max_X \{p(\mathbf{O}, X|\lambda\}$$

and the associated most likely state sequence $\mathbf{X}^*$ in the maximisation.

Introduce a variable

$$\phi_j(t) = \max_X \{p(\mathbf{o}_1, \ldots, \mathbf{o}_t, x(t) = j|\lambda)\}$$

We now need to define an efficient recursion. This may be achieved using

$$\phi_j(t) = \max_i \left\{\phi_i(t-1)a_{ij}\right\} b_j(\mathbf{o}_t)$$

The algorithm is initialised with

$$\phi_j(0) = \begin{cases} 1, & j = 1 \\ 0, & j > 1 \end{cases}$$

In words: $\phi_j(t)$ represents probability of a partial path through the trellis and the *max* is over all partial paths ending in state j at time t.

In practice it is convenient to use logs to avoid underflow.

$$\psi_j(t) = \max_i \left\{\psi_i(t-1) + \log(a_{ij})\right\} + \log(b_j(\mathbf{o}_t))$$

Paths

We would also like to be able to obtain the state sequence associated with $\hat{p}(\mathbf{O}|\lambda)$.

To do this we introduce the concept of a **path**. This is crucial to understanding continuous speech recognition.

A (partial) path represents an alignment of states with the frames of speech starting from the start of the utterance and continuing to time t.

Thus a path is represented by

- a score – usually a log likelihood
- a history – to record the preceding sequence of states

For isolated word recognition where we might be interested in finding the best state sequence the history should be at the state level.

The extension to handle **continuous speech recognition** requires that the history be able to give the word (or phone) sequence.

The problem is to have a representation of the path that allows:

- a compact form of the history to be stored;
- at the end of the utterance the representation of the history can be mapped to the state/phone/word sequence.

To achieve this we will use **token passing** and **traceback**. The traceback information is stored at the unit level, if the state sequence within a model isn't needed.

Token passing

The information describing the head of a (partial) path can be stored in a **token** containing

`LogP` log-likelihood of (partial) path
`Link` pointer to history information

All states of the expanded HMM network hold one token.

We can then define

- **start** token: `LogP`$= 0$, `Link=NULL (*)`
- **null** token: `LogP` $= -\infty$, `Link=NULL (*)`

At time t, the token in each model state represents a path through trellis covering input speech from time 1 to

(a) $t - \delta t$ for state 1
(b) t for states $2 \rightarrow N-1$
(c) $t + \delta t$ for state N

On completion, `LogP` of token in state N should be $\hat{p}(\mathbf{O}|\lambda)$

Token Passing (cont)

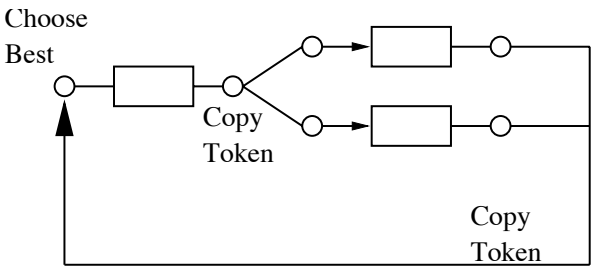
Initially consider *isolated word* recognition. We will only be interested in the word log likelihood (isolated word recognition, so we do not need to worry about the history).

Viterbi algorithm can be restated as:

```
Put a start token in entry node;
Put null tokens in all other nodes;
for each time t = 1 to T do
    /* Start of Step Model */
    for each state i < N do
        Pass a copy of the token Q in state i to
           all connecting states j;
        Q.LogP = Q.LogP + log(a_ij) + log(b_j(o_t))
    end;
    Discard all original tokens;
    for each state i < N do
        Find token in state i with max LogP
           and discard the rest
    end;
    for each state i connected to state N do
        Pass a copy of the token Q in state i to state N
        Q.LogP = Q.LogP+ log(a_iN)
    end;
    Find token in state N with max LogP
       and discard the rest;
    Put null token in entry state
    /* End of Step Model */
end;
```

Connected Unit Case

Extension of the token passing algorithm to deal with connected units is now simple. Algorithm becomes:




```
Put a start token in all network entry states;
Put null tokens in all other states;
for each time t = 1 to T do
    Step All Models;
    Propagate Exit Tokens to all connecting entry states;
    Record Decisions;
    Delete all but the best token in each entry state
end
```

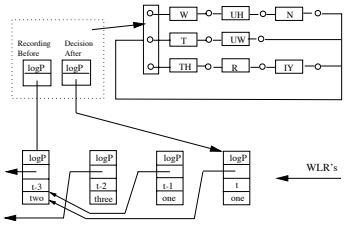
On completion, best token in exit states of all valid network final models represents most likely model sequence. Note that common entry states should be regarded as being a single entry state for this and subsequent algorithms.
We need to decide how `Record Decisions` should be implemented.

Record Decisions

This function is used to give a compact history representation (for this algorithm at the word level). The points at which tokens are propagated from the exit states of words to entry states of other words must be recorded. A **Word Link Record** (WLR) is used.

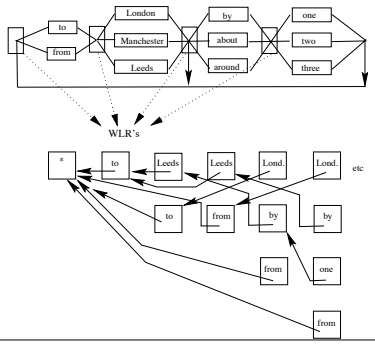
```
for each best token Q in each entry state at each t do
    Create a new WLR w containing:
           (1) a copy of Q;   (2) time t;   (3) identity of emitting word
    Q.link = w
end;
```

For simple loop network:



1 WLR generated per speech frame.

When syntax constraints are included:



1 WLR is generated for each **syntactically distinct node in the network**.

Note that when (bigram) transition probabilities are added to a loop word network then each word-end becomes syntactically distinct and a WLR may need to be generated for each (or each active - see later).

Traceback

Word link records give a compact representation for the history.

We now need to obtain the best word sequence.

After the final frame of speech has been processed

```
Examine WLRs generated at time T, find WLR with max LogP;
Print WLR.time, WLR.LogP, WLR.word;
while WLR.Link != NULL do
      WLR = WLR.Link;
      Print WLR.time, WLR.LogP, WLR.word;
end;
```

The traceback procedure yields:

1. best sequence (in reverse order)
2. word boundary locations
3. LogP for each individual word (can be calculated)

Performance Issues

For W words, N states/word (average), we need to perform

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

for each time frame (typically 10ms)

Each internal propagation involves

(a) output prob calculation (may be shared)

(b) 2 or 3 add and compare ops for state instance

The internal propagation is (at least for medium vocabs) by state-output probability calculations.

For $W \times N$ distinct states, M Gaussians per state (in a GMM) and vector size V, need to have $W \times N \times M \times V \times 2$ multiply-adds per frame.

If $W = 1000, N = 10, M = 16, V = 39$ which is 12.5 million multiply-adds per 10ms frame.

For a large vocab system (esp one with more complex acoustic models and a language model) the internal/external token propagation is roughly as expensive (due to the very large number of state instances).

In practice, a substantial reduction in computation can be achieved by only considering paths which are close in score to the best path. This is called **Beam Search**.

Basic beam search algorithm

We would like to ensure that our search is **admissible** (i.e. the best path is **guaranteed** to be found).

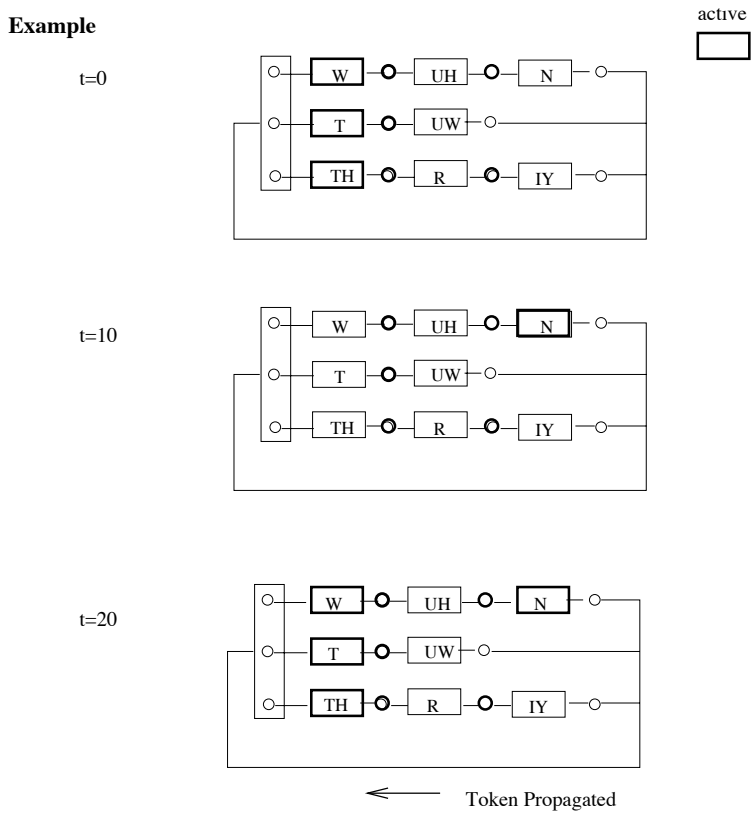
In practice this is too expensive, and hence we will accept a certain level of **search errors**.

Each model instance is either **active** or **inactive**.

The basic beam search algorithm is:

```
Set all entry models of all network initial words are active;
for each time t = 1 to T do
    for each active model w do
        Step Model;
        Find maximum LogP in w, lMax(w);
    end;
    Find global maximum LogP, gMax
    for each active model w do
        if lmax(w) < gMax - Thresh
           De-Activate w;
    end;
    Propagate Exit Tokens to all connecting entry states if LogP > gMax - Thresh;
    Record Decisions;
    Delete all but the best token in each entry state;
    Re-Activate all entry models which have just received a new entry token;
end;
```


Beam Search Example



Partial Traceback

During recognition, every active token represents a possible path.

In the basic algorithm, we wait until end of speech and then **traceback** to find the best sequence.

However, a large grammar could generate 100's of WLR's per frame. This results in a large memory management overhead and a waste of memory.

This is not necessary since:

1. not all WLR's lie on active paths;
2. when a word is recognised "well" - all active paths pass through that WLR (esp. when pruning is used)

The aim of **partial traceback** is to:

- remove all WLR's that do not lie on active paths (count tracing back from active tokens = 0)
- if all active paths pass through the **same** node (model) in the network, output path up to that word (i.e. count tracing back from active tokens = num_active_tokens)

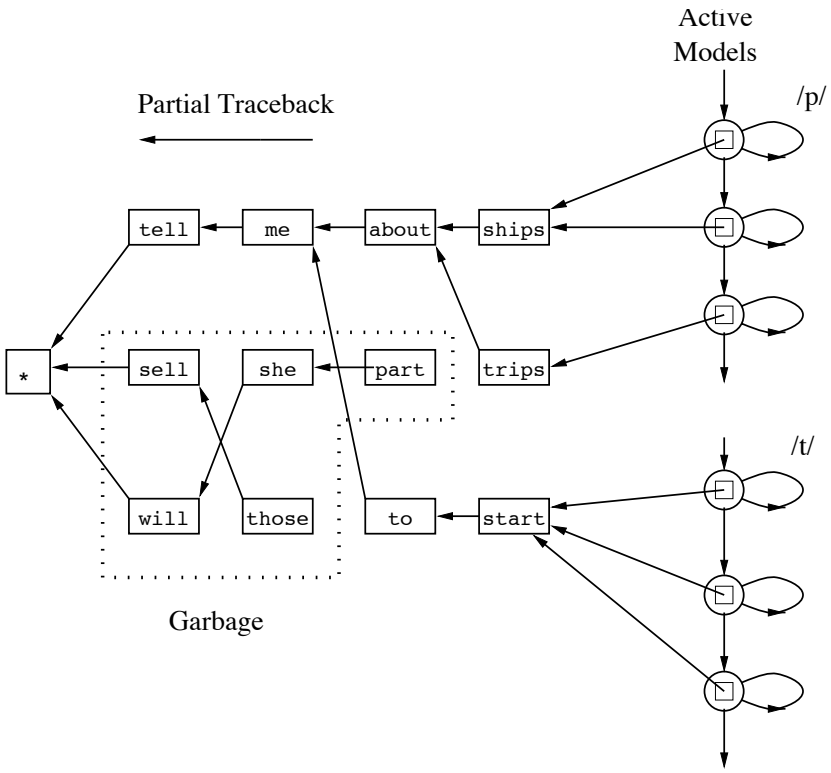
Note: partial traceback allows a recogniser to run continuously without having to wait until the speaker has finished speaking before outputting something

- low latency output is a great advantage in interactive use

Partial Traceback Example

In the example:

- 5 WLRs can be deleted as they are not on active paths;
- `tell me` can be printed.



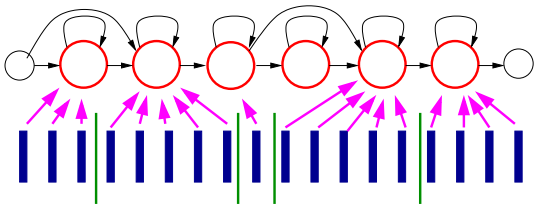
Viterbi Training

The **Viterbi algorithm** gives the **most likely state sequence** $\mathbf{X}^*$ given the current model parameters and can approximate the total likelihood (providing a lower bound), i.e.

$$p(\mathbf{O}|\lambda) \approx p(\mathbf{O}, \mathbf{X}^*|\lambda)$$

"Best" state sequence $\mathbf{X}^*$ assigns each observation vector to exactly one (emitting) state.

- construct composite model for training utterance(s)
- find all observation vectors for each emitting state
- find the number of times that particular state transitions are taken



Can now find maximum likelihood estimates for an emitting state Gaussian output distribution

- sample mean and sample covariance matrix of aligned features for each state

For the transition parameters, the ML estimate (given the constraints that $a_{ij} \geq 0, \sum_j a_{ij} = 1$) is simply computed as:

$$a_{ij} = \frac{\text{Number of transitions state } i \rightarrow \text{ state } j}{\text{Number of transitions from state } i}$$

Viterbi Training - Gaussian Parameter estimation

From the best state sequences from the training utterances, know which data vectors have been aligned with each state.

Due to the HMM state conditional independence assumption, the parameters of each output distribution can be estimated independently.

Formally we can use an indicator variable $\delta(\cdot)$ to show if the observation vector at time t, $\mathbf{o}_t$, is aligned with state j:

$$\delta(x^*(t) = j) = \begin{cases} 1 & \text{if } x^*(t) = j \\ 0 & \text{else} \end{cases}$$

This yields simple expressions for mean vectors and covariance matrices of the Gaussian output distribution for all states (i.e. the sample mean and covariance of the data assigned to each emitting state).

Note that we need to sum over all training data sequences.

For each emitting state ($1 < j < N$):

$$\hat{\boldsymbol{\mu}}_j = \frac{\sum_{r=1}^{R} \sum_{t=1}^{T^{(r)}} \delta(x^{(r)*}(t) = j) \mathbf{o}_t^{(r)}}{\sum_{r=1}^{R} \sum_{t=1}^{T^{(r)}} \delta(x^{(r)*}(t) = j)}$$

$$\hat{\boldsymbol{\Sigma}}_j = \frac{\sum_{r=1}^{R} \sum_{t=1}^{T^{(r)}} \delta(x^{(r)*}(t) = j)(\mathbf{o}_t^{(r)} - \hat{\boldsymbol{\mu}}_j)(\mathbf{o}_t^{(r)} - \hat{\boldsymbol{\mu}}_j)'}{\sum_{r=1}^{R} \sum_{t=1}^{T^r} \delta(x^{(r)*}(t) = j)}$$

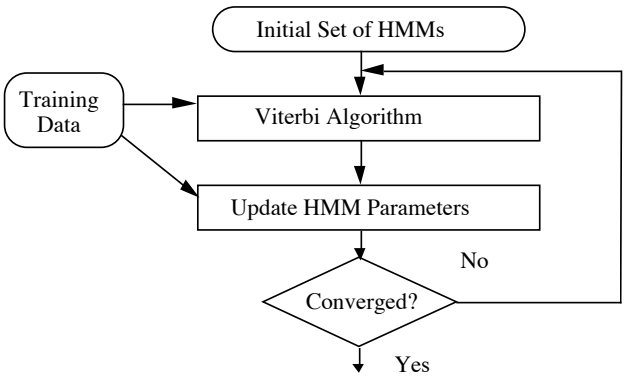
Iterative Training With the Best State-Sequence

Since the Viterbi Algorithm finds the most likely state sequence given the model parameters, can use the above method to find the best state sequence and update the parameters.

However the **most likely state sequence depends on the parameters**.

This suggests an iterative approach, which **interleaves**

- find best state sequence given the current model parameters
- model parameter update given the best state sequence



Each of the above steps will increase the likelihood (either by finding a better state sequence, or improving the model parameters given the state sequence).

Hence the likelihood is guaranteed to increase (or stay the same) with each iteration, and will converge to a **local maximum** of the likelihood taking into account only the best sequence. Note that since only a local maximum is obtained, different initialisations will in general converge to different solutions.

Summary

- Continuous speech recognition uses the Viterbi algorithm to find best path through a composite HMM generator model.
  - Best path defines recognised sequence.
- Assumes all model information is compiled into sentence generator recognition network (can lead to v. large networks)
- Viterbi algorithm can be implemented using token passing
- Beam search is approximate but makes searching large networks practical
- If beam is wide enough, beam search will find most likely path / sequence
  - Can trade off between computation and search errors
  - Similar beam-search can be also used in forward-backward training and is important for long-utterances (used in HTK by `HERest`).
- Partial traceback and garbage collection make continuous operation possible

- Viterbi training can find maximum likelihood parameter estimates. At each iteration find for training data:
  - most likely state sequence given current model parameters
  - parameter update given the state-sequence (alignment)