1. Markov models

1.1 Markov-chain

Let X be a random variable $X = (X_1, \ldots, X_t)$ taking values in some set $S = \{s_1, \ldots, s_N\}$. The sequence is *Markov chain* if it has the following properties:

1. **Limited horizon:**

$$P(X_{t+1} = s_k | X_1, \ldots, X_t) = P(X_{t+1} = s_k | X_t) \tag{1}$$

2. **Time invariant (stationary):**

$$P(X_2 = s_k | X_1 = s_j) = P(X_{t+1} = s_k | X_t = s_j), \forall t, k, j \tag{2}$$

The subsequent variables may be dependent. In the general (non-Markovian) case, a variable X_t may depend on the all previous variables $X_1, \ldots, X_{t-1}$.

Transition matrix

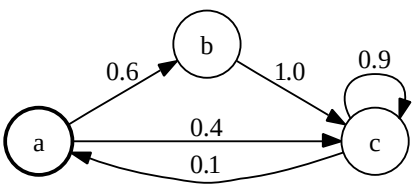
A Markov model can be represented as a set of states (observations) S, and a transition matrix A. For example, $S = \{a, e, i, h, t, p\}$ and $a_{ij} = P(X_{t+1} = s_j | X_t = s_i)$:

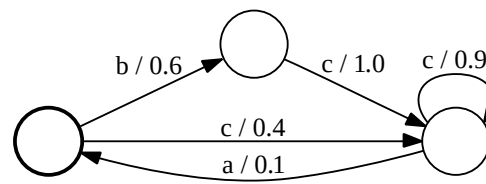
$a_{i,j}$	a	e	i	h	t	p
a	0.6	0	0	0	0	0.4
e	...					...
i					0.6	0.4
h	...					...
t	0.3	0	0.3	0.4	0	0
p	...					...
s_0	0.9	0.1				

Initial state probabilitites π_{s_i} can be represented as transitions from an auxilliary initial state s_0.

Markov model as finite-state automaton

- Finite-State Automaton (FSA): states and transitions.
- Weighted or probabilistic automaton: each transition has a probability, and transitions leaving a state sum to one.
- A Markov model can be represented as a FSA. Observations either in states or transitions.





Probability of a sequence: Given a Markov model (set of states S and transition matrix A) and a sequence of states (observations) X, it is straightforward to compute the probability of the sequence:

$$\begin{aligned} P(X_1 \dots X_T) &= P(X_1)P(X_2|X_1)P(X_3|X_1X_2)\dots P(X_T|X_1\dots X_{T-1}) && (3)\\ &= P(X_1)P(X_2|X_1)P(X_3|X_2)\dots P(X_T|X_{T-1}) && (4)\\ &= \pi_{X_1}\prod_{t=1}^{T-1} a_{X_tX_{t+1}} && (5) \end{aligned}$$

where a_{ij} is the transition probability from state i to state j, and π_{X_1} is the initial state probability.

N-gram model is a Markov model:

In written natural language, the probability distribution of the next word (or letter) depends heavily on the previous words (or letters).

- For example, in a 3-gram (trigram) model $P(w_3|w_1w_2)$, the set of states (observations) S is the set of all 2-grams $S = \{w_1w_1, w_1w_2, \dots w_1w_N$ Transition from w_aw_b to w_xw_y allowed only if $b = x$.

- Even long word histories can be represented as a finite set of states so that the 1st Markov property is valid.

1.2 Hidden Markov models

Sometimes the random process has hidden states that we want to model. For example, we observe speech (waveform or spectral representation) but we would like to model the phonemes that have "generated" the observations. We can think that each phoneme is a hidden state, and different states (phonemes) generate observations from different probability distributions.

Hidden Markov model (HMM)

HMM is a tuple (S, K, Π, A, B).

- Set of states: $S = \{s_1, \dots s_N\}$
- Set of possible observations: $K = \{k_1 \dots k_t\}$
- Initial state probabilitites: $\Pi = \{\pi_i\}$
- Transition probabilities: $A = \{a_{ij}\}, 1 \leq i, j \leq N$
- Observation probability distribution for each transition: $B = \{b_{ijk}\}$

(probability that observation k is generated in transition from s_i to s_j).

Difference to (visible) Markov model: Even if the observations are known, the hidden state sequence is not known. However, we can compute probabilities for different state sequences.

Some random processes:

Partner's behaviour:

- Observations: Slams door, "That's nice!", cooking, "Great!", not speaking
- Hidden states: Happy, neutral, angry

Lecturer's speech:

- Observations: speech signal
- Hidden states: phonemes

Variants

- null-transitions: some transitions do not generate observations (marked as epsilon ϵ)
- arc emission: observations generated in transition: b_{ijk}
- state emission: observations generated in states: b_{ik}

1.3 HMM algorithms

There are three important HMM algorithms:

1. Given model $\mu = (\Pi, A, B)$ and observation sequence $O = (o_1, \ldots, o_T)$, compute probability $P(O|\mu)$.

2. Given model μ and observation sequence O, compute the most likely hidden state sequence X. (Decoding)

3. Given observation sequence O, compute model μ that best explains the observations. (Training)

Algorithm 1: Probability of observation sequence

Straightforward (and inefficient) solution: consider each possible state sequence and compute the probability of the observation sequence given the state sequence:

$$P(O|X,\mu) = \prod_{t=1}^{T} P(o_t|X_t, X_{t+1}, \mu) \quad (6)$$

$$P(O|\mu) = \sum_{X} P(O,X|\mu) = \sum_{X} P(O|X,\mu)P(X|\mu) \quad (7)$$

$$= \sum_{X_1 \ldots X_{T+1}} \pi_{X_1} \prod_{t=1}^{T} a_{X_t X_t+1} b_{X_t, X_{t+1}, o_t} \quad (8)$$

Note that many state sequences have common subsequences. *Dynamic programming* exploit this redundancy and is much more efficient.

Forward procedure:

Compute forward variable $\alpha_i(t)$: **probability to be in state** i **after** t **observations**: $\alpha_i(t) = P(o_1 o_2 \dots o_t - 1, X_t = i|\mu)$

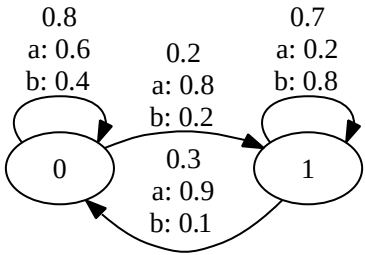
Initial probabilities for all i: $\alpha_i(0) = \pi_i$
for t in $\{1..T\}$
    for all i: $\alpha_i(t) = \sum_{i=1}^{N} \alpha_i(t-1) a_{ij} b_{ijo_t}$
Finally: $P(O|\mu) = \sum_{i=1}^{N} \alpha_i(T)$

Backward procedure: similarly, but computing from the end to start. Forward variable $\alpha_i(t)$ and backward variable $\beta_i(t)$ can be combined at any time:

$$P(O|\mu) = \sum_{i=1}^{N} \alpha_i(t)\beta_i(t), \text{ jossa } 1 \leq t \leq T+1 \tag{9}$$

This property is used in the third algorithm (training).

Example: forward procedure



States: 0, 1.
Initial state: 0.
Observation sequence: a, b, a.

	#	a	b
State 0	1.0	$1.0 \cdot .8 \cdot .6 = .48$	$(.48 \cdot .8 \cdot .4) + (.16 \cdot .3 \cdot .1) = .1584$
State 1	0	$1 \cdot .2 \cdot .8 = .16$	$(.48 \cdot .2 \cdot .2) + (.16 \cdot .7 \cdot .8) = .1088$

a
$(.1584 \cdot .8 \cdot .6) + (.1088 \cdot .3 \cdot .9) = .1054$
$(.1584 \cdot .2 \cdot .8) + (.1088 \cdot .7 \cdot .2) = .0406$

$$P(O|\mu) = .1054 + .0406 = .146$$

Algorithm 2: Decoding

Task: find state sequence X that best explains observations O. The usual interpretation: find the *state sequence* that is most probable given the observations. Can be computed using the Viterbi algorithm (also known as DP alignment, Dynamic Time Warping).

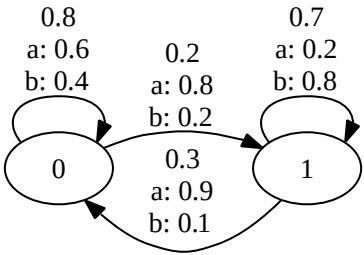
Viterbi algorithm is similar to the forward procedure. Uses variable $\delta_i(t)$ to store the **probability of the single most probable path to state j after t observations**.

- Initialize: $\delta_j(1) = \pi_j$
- Induction: choose the previous state that gives the best probability for the path: $\delta_j(t+1) = \max_{1 \leq i \leq N} \delta_i(t) a_{ij} b_{ijo_t}$

  Remember also where we came from:
  $\phi_j(t+1) = \arg\max_{1 \leq i \leq N} \delta_i(t) a_{ij} b_{ijo_t}$
- Finally: Follow ϕ_j to get the state sequence.

Difference to forward procedure: instead of summing incoming probabilities, we choose the maximum.

Example: Viterbi



States: 0, 1.
Initial state: 0.
Observation sequence: a, b, a.

	#	a	b
State 0	1.0	$\leftarrow 1 \cdot .8 \cdot .6 = .48$	$\leftarrow \max(.48 \cdot .8 \cdot .4, .16 \cdot .3 \cdot .1) = .1536$
State 1	0.0	$\nwarrow 1 \cdot .2 \cdot .8 = .16$	$\leftarrow \max(.48 \cdot .2 \cdot .2, .16 \cdot .7 \cdot .8) = .0896$

a
$\leftarrow \max(.1536 \cdot .8 \cdot .6, .0896 \cdot .3 \cdot .9) =$ **.0737**
$\nwarrow \max(.1536 \cdot .2 \cdot .8, .0896 \cdot .7 \cdot .2) = .0125$

Best state sequence: $0 \rightarrow 0 \rightarrow 0 \rightarrow 0$.

Algorithm 3: Parameter estimation (Training)

Given observations O (training data), find the HMM parameters $\mu = (A, B, \pi)$ that best explain the observations. Maximum likelihood estimation (MLE):

$$\hat{\mu} = \arg\max_{\mu} P(O|\mu) \tag{10}$$

There is no analytical solution.

Baum-Welch (forward-backward algorithm)

- Special case of Expectation Maximization (EM) method. Alternate the following steps:
  1. Keep μ fixed and compute forward and backward variables $\alpha_i(t)$ and $\beta_j(t+1)$. Then α and β can be used to compute the probability that the random process traversed transition ij at time instant t. (Details in Manning 9.3.3)
  2. Using the transition probabilities computed above, re-estimate model parameters $\mu \rightarrow \hat{\mu}$ so that O is as likely as possible.

- Guaranteed: $P(O|\hat{\mu}) \geq P(O|\mu)$
- Finds a local maximum, not necessarily global maximum.

Implementation details and problems

Underflow: lots of small probabilities are multiplied

- Use logarithmic probabilities: multiplications become additions.

Large number of parameters in the model

- Lots of training data required
- Apply "parameter tying": some states share the same emission distribution or transitions
- Do not allow transitions between every state pair: less parameters

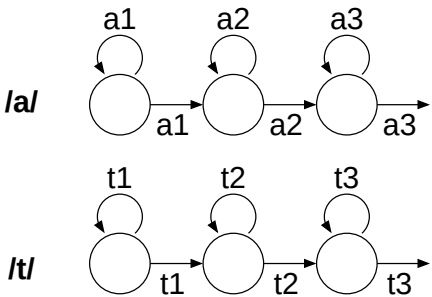
In speech recognition, the HMM models are often trained using simpler Viterbi training instead of Baum-Welch. Alternate the following steps:

- Keep μ fixed and find the best state sequence through the observations.
- Update μ keeping the state sequence fixed.

1.4 HMMs in speech recognition

Phoneme models

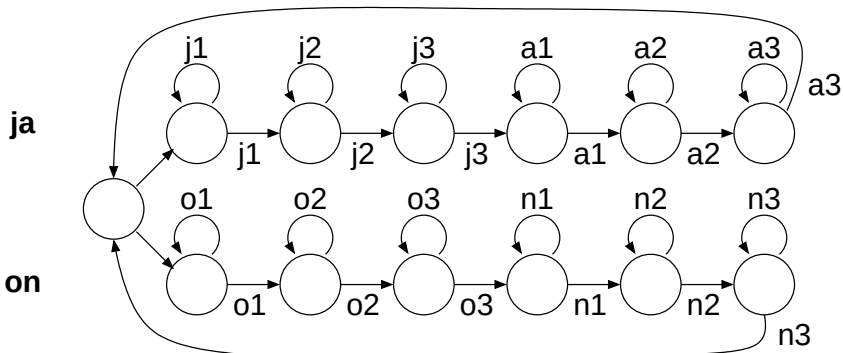
3-state HMM for each phoneme. Observations are features computed from a speech signal.



For training the phoneme models, we need hours or tens of hours speech and textual transcription.

We can also train several different models for one phoneme. For example, one model for "/a/ that follows /k/" or "/a/ that is between /t/ and /u/".

The phoneme models can be used to build a big HMM that contains all phoneme sequences (words or even sentences) that the recognizer should recognize. For English, manually created pronunciation dictionaries may be used. The HMM may also contain several pronunciations variants for each word.



Recognition using the Viterbi algorithm

Given the big HMM and a speech signal, Viterbi algorithm can be used to find the best state sequence through the observations.

- The best state sequence defines also the word sequence.
- Full search not possible for even small vocabularies. At each time instant, we have to prune the worst paths and hope that the correct hypothesis is not lost.

Recognition with n-gram language model

Viterbi algorithm assumes that transition probabilities do not depend on the previous states: we did store only the best path for each state and time instant t.

With n-gram models, the probability of a word depends on the previous words. There are two solutions:

- Build the whole n-gram language model in the HMM, so that there are unique states for each n-gram context. Now the transition probabilities depend only on the previous state again.

  The basic viterbi algorithm can be used, but the HMM easily becomes very large.

- Modify Viterbi algorithm so that, each state can store several paths that correspond to different word histories.

  The size of the HMM remains smaller, but the decoding algorithm becomes more complex.

Units of the n-gram model

In English, the n-gram models are trained using whole words as units. In Finnish (also in Turkish, Estonian and similar languages), the word-based approach does not work so well:

- The vocabulary easily becomes too large, because of inflections and compound words.
- N-gram modelling does not work well with too large vocabularies, because there are too few (or zero) samples for some usual words.

The words can be split in smaller units. The smaller units are used, the less units we need. However, short units require longer contexts in the n-gram model.

Several ways to split the words have been tried in Finnish:

- **Letters**: Very small number of units. However, the n-gram contexts have to be very long. Too short in recognition.
- **Syllables**: Small number of units. Work ok in speech recognition. Foreign words need special treatment.
- **Grammar-based morphemes**: Automatic morphological analyser needed for the target language. Work ok in recognition. Foreign words need special treatment.
- **Statistical morphs (Morfessor)**: Number of units can be controlled. Work well in recognition and handles foreign words too. muutkin. Data-driven method that has also been successful in Turkish and Estonian recognition.

  Demo at http://www.cis.hut.fi/projects/morpho/

Other HMM applications

- part-of-speech tagging
- analysis of DNA sequences