

- Manning, C. D. and Schütze, H. (1999): Foundations of Statistical Natural Language Processing. MIT Press: Cambridge, Massachusetts. Chapter 9.

From Markov Chains to HMMs

HIDDEN MARKOV MODELS

Hiding Markov

- Training of Markov Chains: Count n-grams, normalize to probabilities
- Sparse data: many n-grams not in training
- Back-off smoothing: use shorter n-grams for interpolated estimation

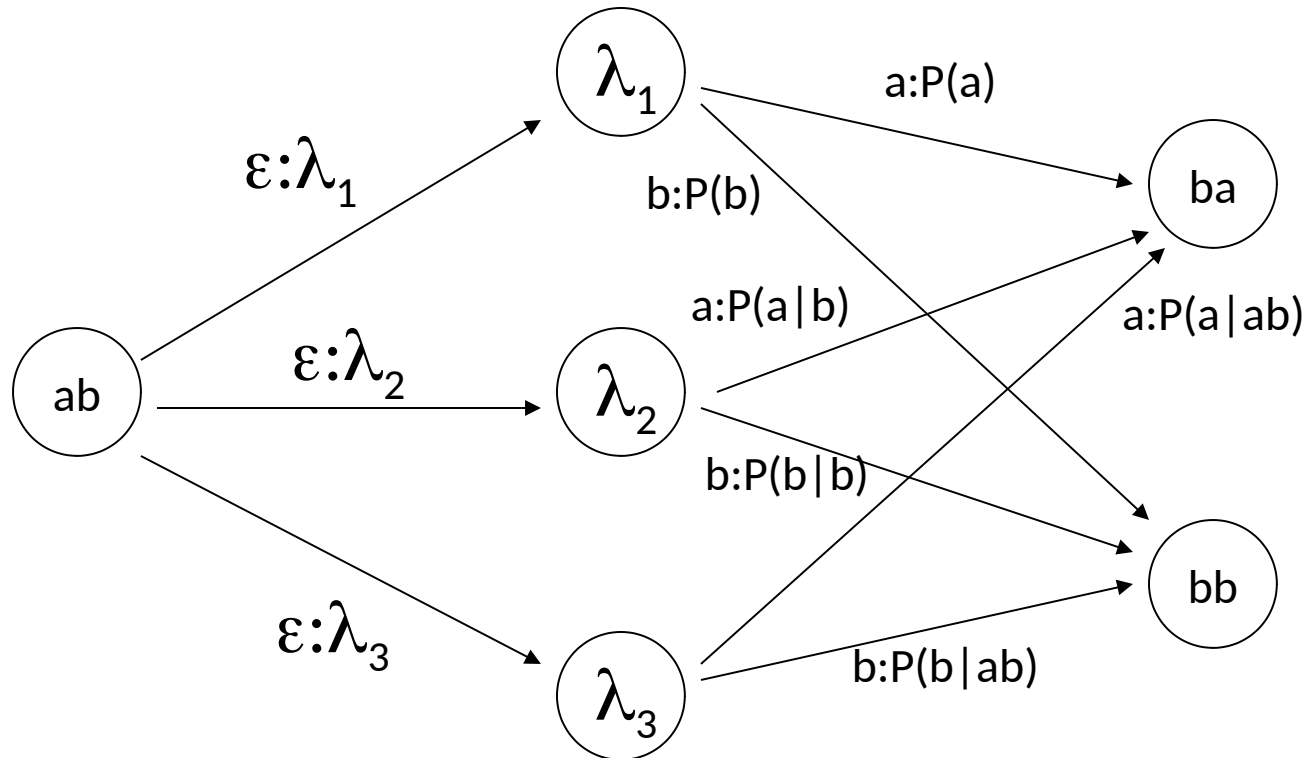
- Recap: Mixture Model:

$$P_{\text{H}}(w^t | w^{t-2}w^{t-1}) = \lambda_1 P_1(w^t) + \lambda_2 P_2(w^t | w^{t-1}) + \lambda_3 P_3(w^t | w^{t-2}w^{t-1}) \text{ where } 0 \leq \lambda_i \leq 1 \text{ and } \sum_i \lambda_i = 1$$

- Hidden Markov Models (HMMs) can be used to
 - model this interpolation
 - train the λ weights

Hidden States

$$P_{\lambda}(w^t | w^{t-2}w^{t-1}) = \lambda_1 P_1(w^t) + \lambda_2 P_2(w^t | w^{t-1}) + \lambda_3 P_3(w^t | w^{t-2}w^{t-1}) \text{ where } 0 \leq \lambda_i \leq 1 \text{ and } \sum_i \lambda_i = 1$$



The path through this non-deterministic WFSA is not determined by the sequence of symbols. There are “hidden” states.

Hidden Markov Model

A **Hidden Markov Model** $HMM=(\Phi, \Sigma, \delta, \Pi)$ consists of

- finite set of states $\Phi = \{z_0, \dots, z_n\}$
- initial state probability distribution $\Pi: \pi_i = P(z^1 = z_i)$
- finite alphabet $\Sigma = \{s_1, s_2, \dots, s_m\}$ of input symbols
- transition function $\delta: \Phi \rightarrow (\Sigma \cup \{\varepsilon\}) \times [0, 1] \times \Phi$

Non-deterministic: several transitions from the same state with the same input symbol are valid and common.

Normalized over all possible combinations of state sequence and symbols.

The probability of a sequence

$$P(s^1...s^n) = \sum_{z^1..z^{n+1}} P(s^1...s^n, z^1..z^{n+1})$$

$$= \sum_{z^1..z^{n+1}} P(z^1)P(s^1, z^2 | z^1)P(s^2, z^3 | s^1, z^1z^2).....P(s^n, z^{n+1} | s^1..s^{n-1}, z^1..z^n) \quad (1)$$

$$= \sum_{z^1..z^{n+1}} P(s^1, z^2 | z^1)P(s^2, z^3 | z^2).....P(s^n, z^{n+1} | z^n) \quad (2)$$

$$= \sum_{z^1..z^{n+1}} \prod_{i=1}^n P(s^i, z^{i+1} | z^i) \quad (3)$$

$$= \sum_{z^1..z^{n+1}} \prod_{i=1}^n P(z^i \xrightarrow{s^i} z^{i+1}) \quad (4)$$

- (1) Expansion with conditional probabilities
- (2) Markov assumption, horizon = 1
- (3) write as product
- (4) probability of a sequence is the sum of the probability of all possible paths through the HMM for this sequence of symbols

Note that superscripts indicate time points, subscripts enumerate symbols of an alphabet

Three Tasks for HMMs

- Given a model defined by a HMM, how do we **efficiently compute** the probability of an observation sequence?
- Given an observation sequence and a model defined by a HMM, how do we choose the **state sequence that best explains** the observations?
- Given an observation sequence and a space of possible models found by varying model parameters, how do we find the **model that best explains** the observation? This is called **training** of the model.

Computing the probability of an observation

- Observation: sequence of symbols $s^1 s^2 \dots s^T$, given
- Naïve strategy:
 - exhaustive search of all possible paths for that sequence
 - probabilities of single paths: product of transition probabilities
 - probabilities of the observation: sum of single path probabilities

This is obviously not efficient for long paths and many bifurcations. If N is the number of states and T the length of the sequence, then the computational complexity is $O(T * N^T)$!

Forward/backward procedures

- Idea: use common short sub-paths and combine into larger sub-paths
- dynamic programming: store intermediate results and thus re-use previous computations for further computations

Forward Probability:

$\alpha_i(t) = P(s^1 \dots s^{t-1}, z^t = z_i)$ is the probability of ending up in state z_i after $t-1$ symbols.

This is computed iteratively for all t in $1..T$ for all states z_i with i in $1..N$.

Sum over $\alpha_i(T+1)$ is the final result.

Whether we start from the beginning or from the end is irrelevant: This can alternatively formulated as a **Backward probability**:

$$\beta_i(t) = P(s^t \dots s^T \mid z^t = z_i)$$

Forward procedure

1. Initialization

$$\alpha_i(1) = \pi_i \quad \text{for } 1 \leq i \leq N$$

2. Induction

$$\alpha_j(t+1) = \sum_{i=1}^N \alpha_i(t) \cdot P(z_i - \overset{s^t}{\longrightarrow} z_j) \quad \text{for } 1 \leq t \leq T, 1 \leq j \leq N$$

3. Total

$$P(s^1, \dots, s^T) = \sum_{i=1}^n \alpha_i(T+1)$$

Complexity: $O(T \cdot N^2)$ multiplications. Much better!

Backward procedure

1. Initialization

$$\beta_i(T+1) = 1 \text{ for } 1 \leq i \leq N$$

2. Induction

$$\beta_j(t) = \sum_{i=1}^N \beta_i(t+1) \cdot P(z_j - \overset{s^t}{\longrightarrow} z_i) \text{ for } 1 \leq t \leq T, 1 \leq j \leq N$$

3. Total

$$P(s^1, \dots, s^T) = \sum_{i=1}^n \pi_i \beta_i(1)$$

Combination to forward/backward procedure

$$\begin{aligned} P(s^1 \dots s^T, z^t = z_i) \\ &= P(s^1 \dots s^{t-1}, z^t = z_i, s^t \dots s^T) = \\ &= P(s^1 \dots s^{t-1}, z^t = z_i) \cdot P(s^t \dots s^T \mid s^1 \dots s^{t-1}, z^t = z_i) = \\ &= P(s^1 \dots s^{t-1}, z^t = z_i) \cdot P(s^t \dots s^T \mid z^t = z_i) = \\ &= \alpha_i(t) \cdot \beta_i(t). \end{aligned}$$

Therefore:

$$P(s^1 \dots s^T) = \sum_{i=1}^N \alpha_i(t) \cdot \beta_i(t) \quad \text{for all } 1 \leq t \leq T + 1$$

Equations for forward and backward probabilities are special cases of this one.

Finding the best state sequence: Decoding

We want to find the most likely complete path given the observation:

$$MAXPATH = \operatorname{argmax}_{z^1 \dots z^T} P(z^1 \dots z^T | s^1 \dots s^T) = \operatorname{argmax}_{z^1 \dots z^T} P(z^1 \dots z^T, s^1 \dots s^T)$$

Like in the previous problem, we use dynamic programming and define the probability of the best path to state z_i after t symbols:

$$\delta_i(t) = \max_{z^1 \dots z^t} P(z^1 \dots z^t, s^1 \dots s^{t-1}, z^t = z_i)$$

The iterative algorithm to solve this problem is called the **Viterbi algorithm**.

Viterbi Algorithm

1. Initialization

$$\delta_i(1) = \pi_i \quad \text{for } 1 \leq i \leq N$$

2. Induction

$$\delta_j(t+1) = \max_{i=1..N} \delta_i(t) * P(z_i - s_t \rightarrow z_j) \quad \text{for } 1 \leq j \leq N$$

store backtrace: per state j , memorize the previous state for $\delta_j(t+1)$

$$\psi_j(t+1) = \operatorname{argmax}_{i=1..N} \delta_i(t) * P(z_i - s_t \rightarrow z_j) \quad \text{for } 1 \leq j \leq N$$

3. Termination:

last state in *MAXPATH*:

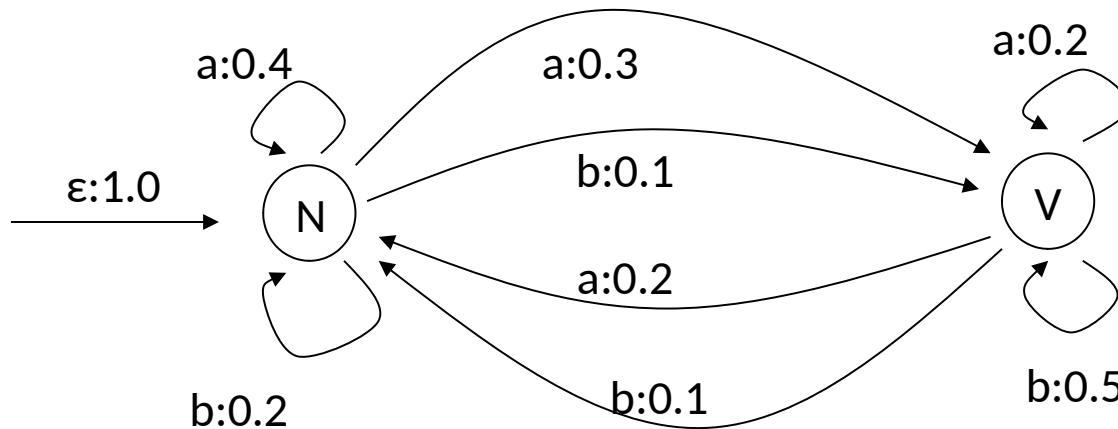
$$z_{\max}^{T+1} = \operatorname{argmax}_{i=1..N} \delta_i(T+1)$$

read sequence according to:

$$z_{\max}^t = \psi_{z_{\max}^{t+1}}(t+1)$$

Ties: resolve randomly or store n-best-list.

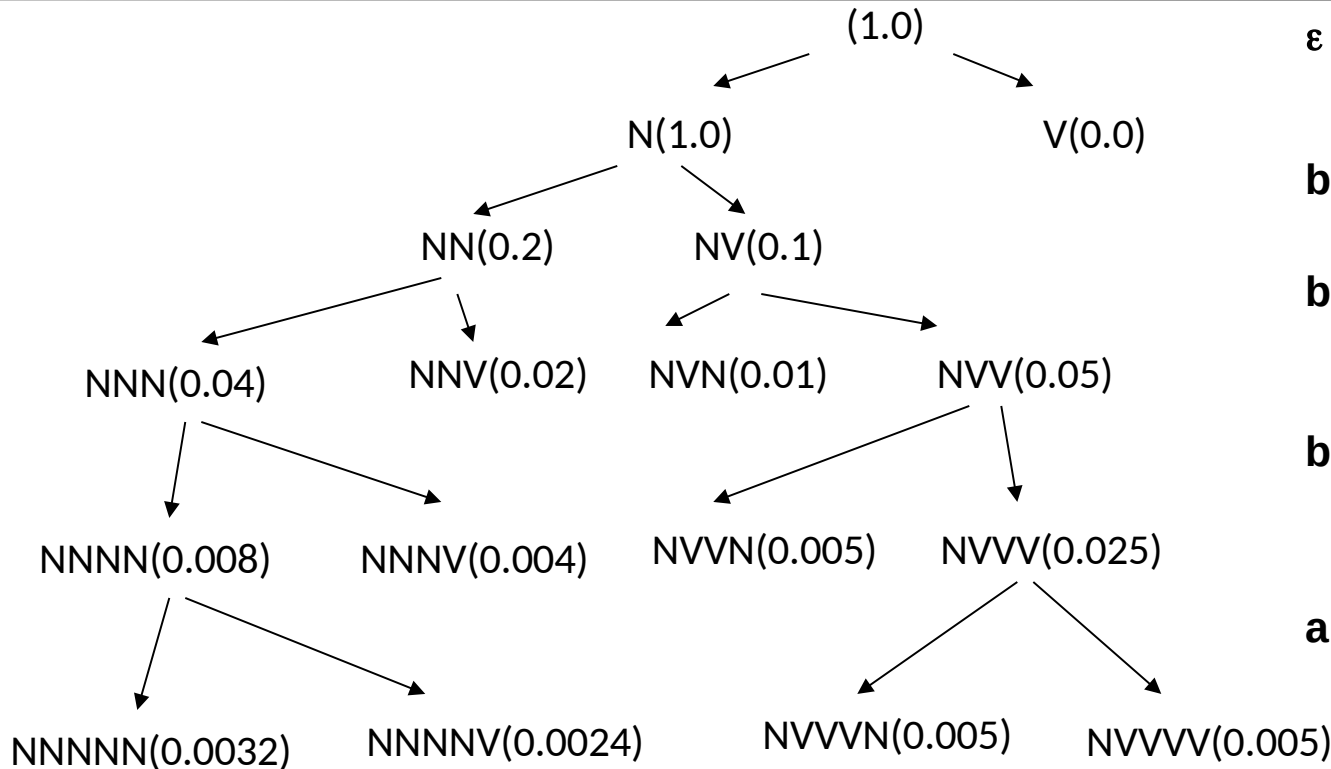
Viterbi Example



Observation sequence: bbba

input read	ε	b	bb	bbb	bbba
N - sequence	N	NN	NNN	NNNN	NVNVN
N - max. P	1.0	0.2	0.04	0.008	0.005
V - sequence	V	NV	NVV	NVVV	NVVVV
V - max. P	0.0	0.1	0.05	0.025	0.005

Looking for the best path



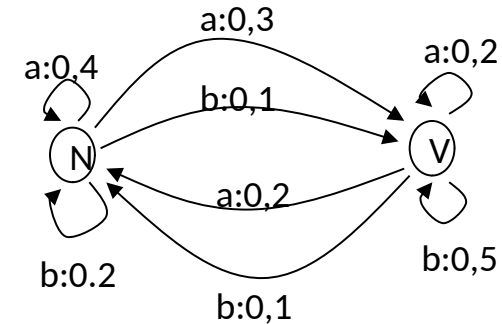
ε

b

b

b

a



Hopeless path prefixes are dropped from consideration

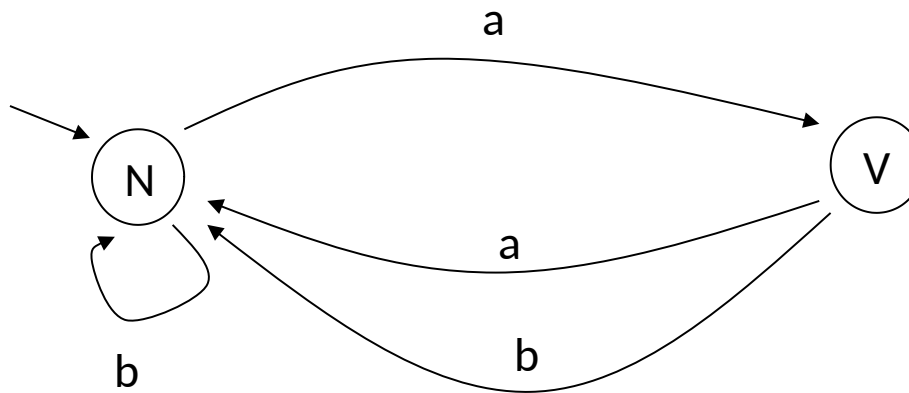
HMM Training

- Training: We have a fixed structure and optimize the parameters according to observations
- Name of training algorithm: **Baum-Welch** or **Forward-Backward algorithm**
- Idea:
 - start with random parameters
 - tune parameters such that the probability of the training sequence increases

Why is this necessary? Why can't we just train it like a Markov Chain?

Recap: Training a Markov Chain

- Given:
 - Markov Chain without transition probabilities
 - training sequence
- Wanted:
 - transition probabilities



training sequence: abbaababbbaaa

Markov Chain: Count, (smooth), and done.

from state	to state	symbol	count
N	V	a	5
N	N	b	3
V	N	a	2
V	N	b	2

sequence: abbaababbaaa

$$P_e(N \xrightarrow{a} V) = 5 / 8$$

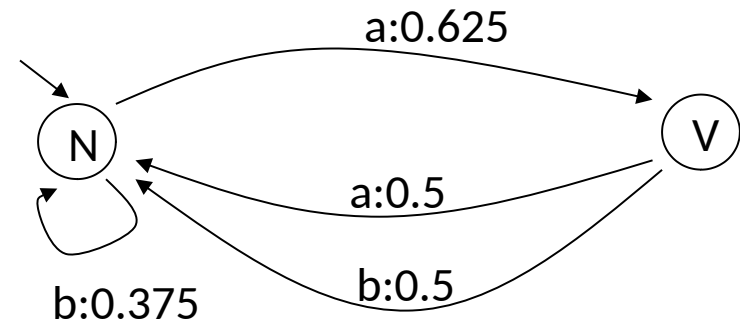
$$P_e(N \xrightarrow{b} N) = 3 / 8$$

$$P_e(V \xrightarrow{a} N) = 2 / 4$$

$$P_e(V \xrightarrow{b} N) = 2 / 4$$

Problem with HMMs:

- non-deterministic: assume that all possible transitions are used at the same time?
- how to kick it off?



Counting transitions for HMMs

Example:

- two possible paths through HMM for a given sequence
 - $P(1^{\text{st}} \text{ path}) = 1/3$ using transition T1
 - $P(2^{\text{nd}} \text{ path}) = 2/3$ using transition T2
- ➔ increase count of T1 by 1/3 and count of T2 by 2/3

General case:

$$\begin{aligned} C(z_i - \overset{s_k}{\rightarrow} z_j) &= \sum_{z^1 \dots z^{T+1}} P(z^1 \dots z^{T+1} | s^1 \dots s^T) \cdot \eta(z_i - \overset{s_k}{\rightarrow} z_j, z^1 \dots z^{T+1}, s^1 \dots s^T) = \\ &= \frac{1}{P(s^1 \dots s^T)} \sum_{z^1 \dots z^{T+1}} P(z^1 \dots z^{T+1}, s^1 \dots s^T) \cdot \eta(z_i - \overset{s_k}{\rightarrow} z_j, z^1 \dots z^{T+1}, s^1 \dots s^T) \end{aligned}$$

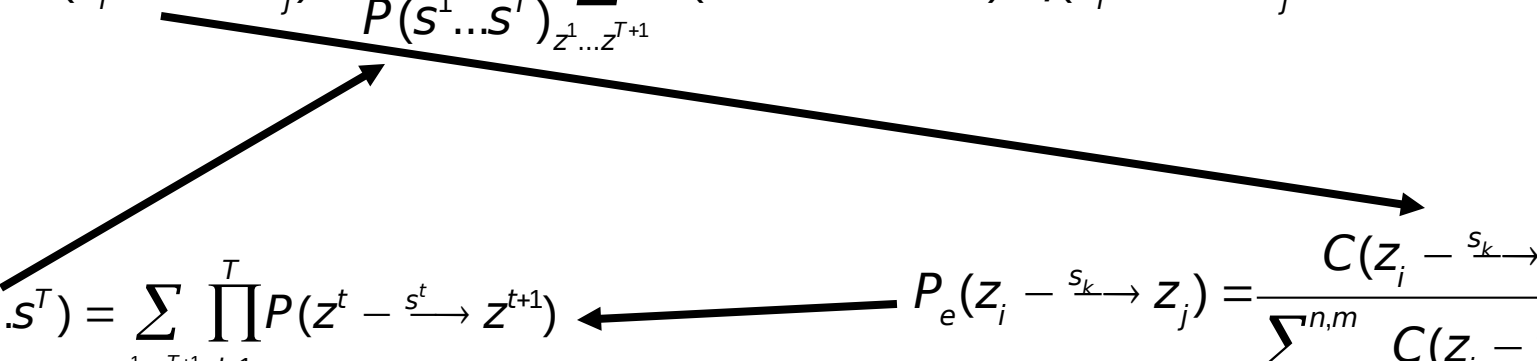
$\eta(z_i - \overset{s_k}{\rightarrow} z_j, z^1 \dots z^{n+1}, s^1 \dots s^n)$ is the number of times that the transition from z_i to z_j with symbol s_k is contained in $(z^1 \dots z^{n+1}, s^1 \dots s^{n+1})$

Path probabilities and transition probabilities

- We count transitions to compute transition probabilities
- in the computation, we use path probabilities
- ... but path probabilities are computed by transition probabilities

$$C(z_i - s_k \rightarrow z_j) = \frac{1}{P(s^1 \dots s^T)} \sum_{z^1 \dots z^{T+1}} P(z^1 \dots z^{T+1}, s^1 \dots s^T) \cdot \eta(z_i - s_k \rightarrow z_j, z^1 \dots z^{T+1}, s^1 \dots s^T)$$

$$P(s^1 \dots s^T) = \sum_{z^1 \dots z^{T+1}} \prod_{t=1}^T P(z^t - s^t \rightarrow z^{t+1})$$

$$P_e(z_i - s_k \rightarrow z_j) = \frac{C(z_i - s_k \rightarrow z_j)}{\sum_{l=1, k=1}^{n, m} C(z_i - s_k \rightarrow z_l)}$$


Deadlock?

Solution: Expectation Maximization (EM)

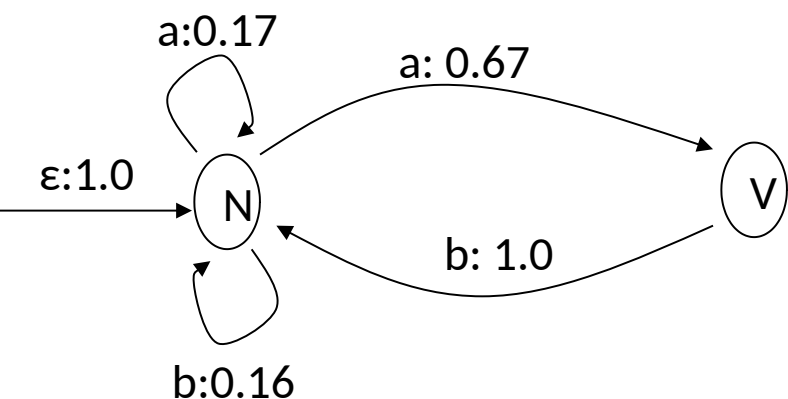
Top-level EM algorithm:

```
Old_Cross_Entropy=infinite;  
Guess HMM Parameters;  
New_Cross_Entropy=Re-estimate_Parameters();  
While not (Old_Cross_Entropy  $\approx$  New_Cross_Entropy) {  
    Old_Cross_Entropy=New_Cross_Entropy;  
    New_Cross_Entropy=Re-estimate_Parameters();  
}
```

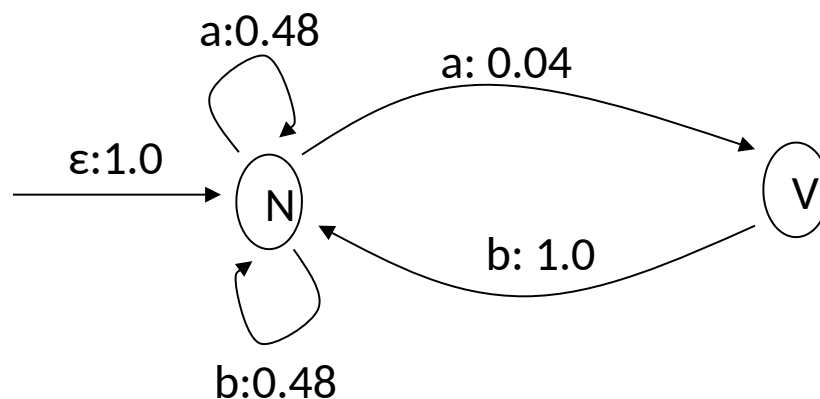
If it is guaranteed that **Re-estimate_Parameters()** lowers the cross entropy between sequence and HMM, then the EM algorithm converges to a (local) maximum.

For HMMs: re-estimation procedure of iteratively using path probabilities to estimate transition probabilities has been proved by Baum to lower cross entropy (and thus increases the probability of the sequence).

Example: Parameter Estimation



'true' HMM: $P(ababb) = 0.0778$



initial random estimate

training sequence: ababb

possible paths:

- NVNVNN
- NVNNNN
- NNNVNN
- NNNNNN

Two iterations

training sequence: ababb

1.

<i>path</i>	$P(\text{path})$	$P(V a,N)$	$P(N b,V)$	$P(N a,N)$	$P(N b,N)$	$\text{sum } P(. .,N)$
NVNVNN	0.00077	0.00154	0.00154	0.0	0.00077	
NVNNNN	0.00442	0.00442	0.00442	0.00442	0.00884	
NNNVNN	0.00442	0.00442	0.00442	0.00442	0.00884	
NNNNNN	0.02548	0.0	0.0	0.05096	0.07644	
<i>sum over paths</i>	0.03509	0.01038	0.01038	0.05970	0.09489	0.165
<i>new P</i>		0.06	1.0	0.36	0.58	

2.

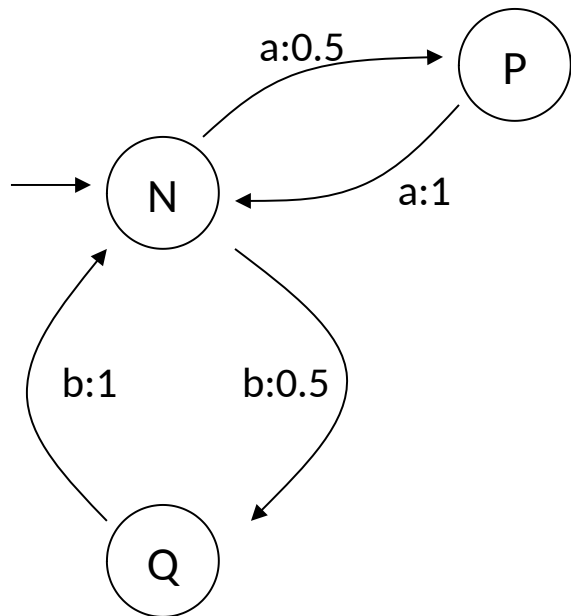
<i>path</i>	$P(\text{path})$	$P(V a,N)$	$P(N b,V)$	$P(N a,N)$	$P(N b,N)$	$\text{sum } P(. .,N)$
NVNVNN	0.00209	0.00418	0.00418	0.0	0.00209	
NVNNNN	0.00727	0.00727	0.00727	0.00727	0.01454	
NNNVNN	0.00727	0.00727	0.00727	0.00727	0.01454	
NNNNNN	0.02529	0.0	0.0	0.05058	0.07587	
<i>sum over paths</i>	0.04192	0.01872	0.01872	0.06512	0.10704	0.191
<i>new P</i>		0.10	1.0	0.34	0.56	

probability of sequence for iteration 3: 0.0472

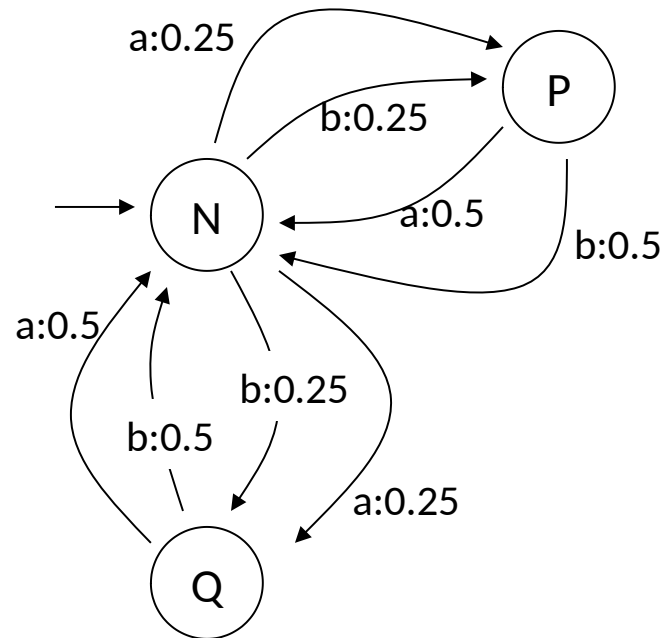
Problems of this training algorithm

- Critical points: Algorithm cannot decide for a direction and gets stuck
- if training sequence is not representative, estimated parameters will be suboptimal. This holds especially for short sequences
- even without critical points: algorithm finds local maximum, not the global maximum

Critical point example



correct HMM
 $P(aabb)=0.25$

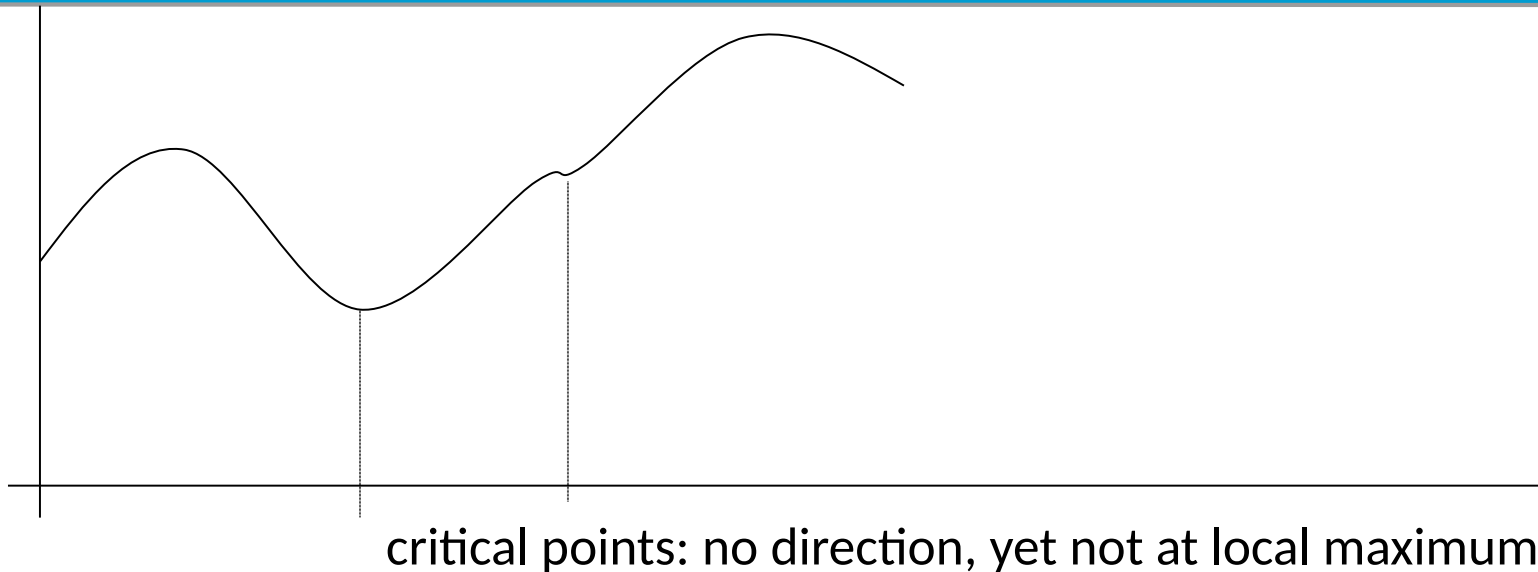


HMM with critical point
 $P(aabb)=0.0625$

training sequence: aabb

Parameters are stable under EM-iteration: Critical points are fix points.

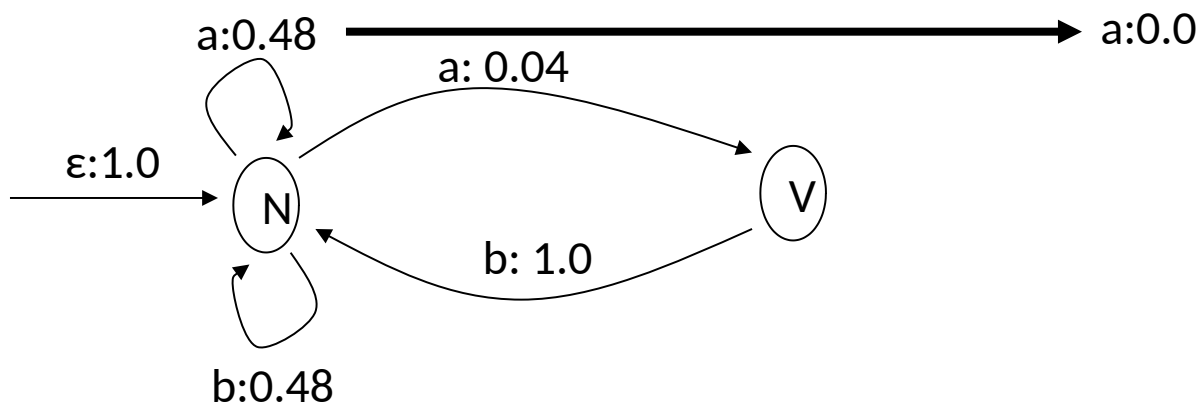
Dealing with critical points: Add noise



Add noise: small random changes in transition probabilities lead to a decision but do not change overall convergence in most cases.

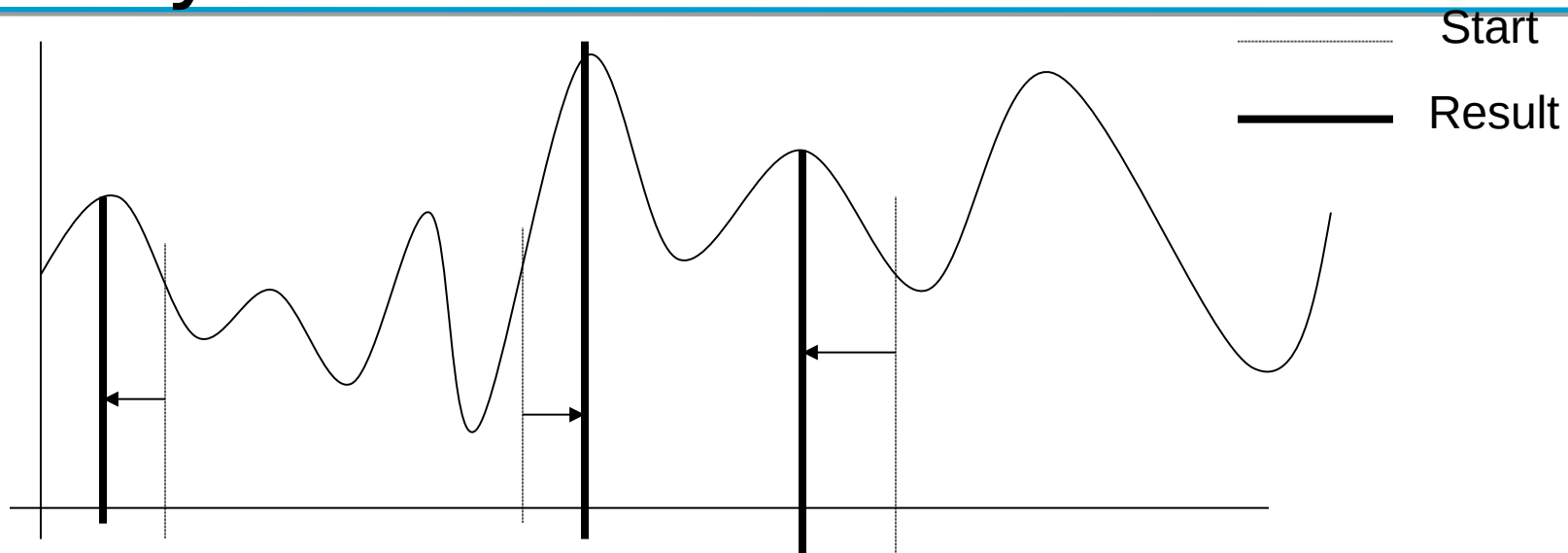
What about the Zeros?

- Minimizing the cross entropy means: transitions that never occur in the training will converge to 0.
- In the example ababb: sequence “aa” never occurs: $P(N|a,N)$ goes to 0



- Solution: Smoothing and back-off of counts
- Longer training sequences always help, but increase training time

Hill climbing: only local maxima



- EM algorithm implements hill climbing
- in case many local maxima exist, the result is heavily dependent on initialization. Chance of hitting the global maximum is very small

Strategy: Restart a few times with different initializations and observe.

Optimizing the training time

Recall the calculation of counts of transitions:

$$C(z_i - \overset{s_k}{\rightarrow} z_j) = \frac{1}{P(s^1 \dots s^n)} \sum_{z^1 \dots z^{n+1}} P(z^1 \dots z^{n+1}, s^1 \dots s^{n+1}) \cdot \eta(z_i - \overset{s_k}{\rightarrow} z_j, z^1 \dots z^{n+1}, s^1 \dots s^n)$$

This computes the sum over all possible paths per transition: inefficient!

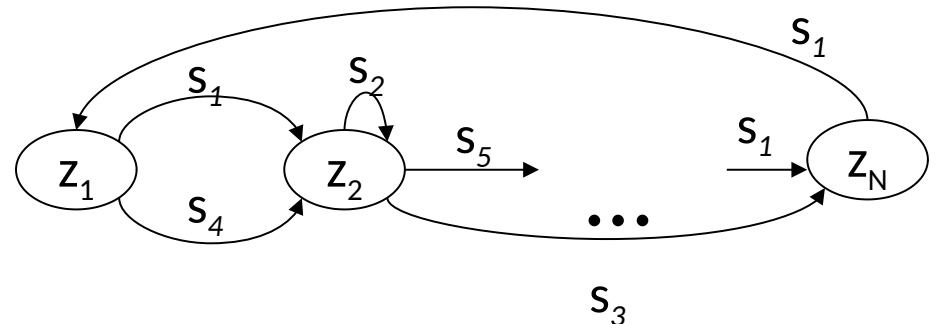
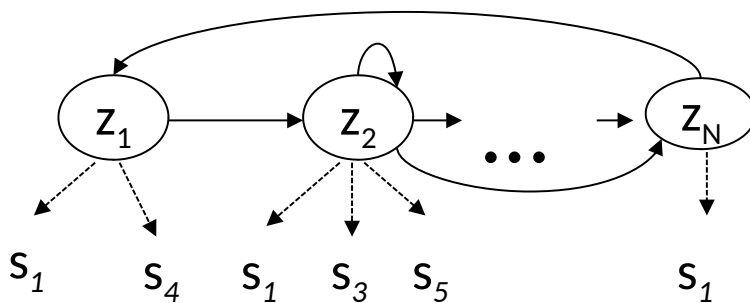
In analogy to the Forward/Backward procedure, we rather compute sub-paths.

Using the combination $P(s^1 \dots s^T) = \sum_{i=1}^N \alpha_i(t) \cdot \beta_i(t)$, we reformulate as:

$$C(z_i - \overset{s_k}{\rightarrow} z_j) = \frac{1}{P(s^1 \dots s^n)} \sum_{t=1}^n \alpha_t(t) P(z_i - \overset{s_k}{\rightarrow} z_j) \beta_j(t+1)$$

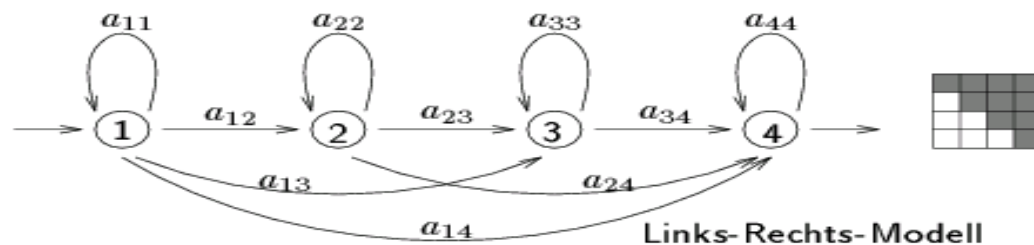
Variants of HMMs

- sparsity handling: parameter tying and tied states
- allowing null-emissions
- modeling of (continuous) time spent in a state
- different model topologies: number of states, structural zeros, feed-forward, ...
- automatic learning of topology
- continuous HMMs: in contrast to discrete symbols
- state emission vs. arc-emission

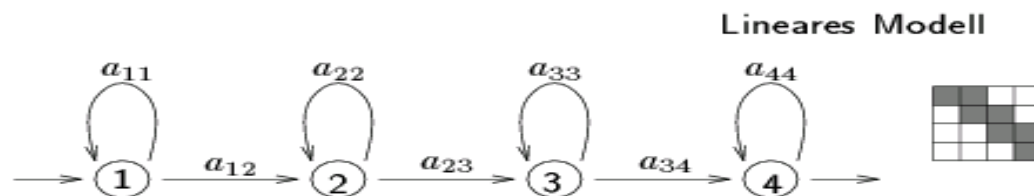
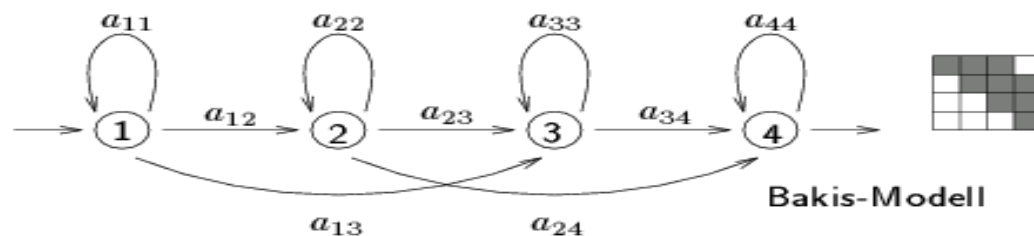


Applications of HMMs I: Language Technology

- Part of Speech tagging: coming up next!
- Speech-to-text: observe sequence of phonemes



some HMM topologies
for speech recognition

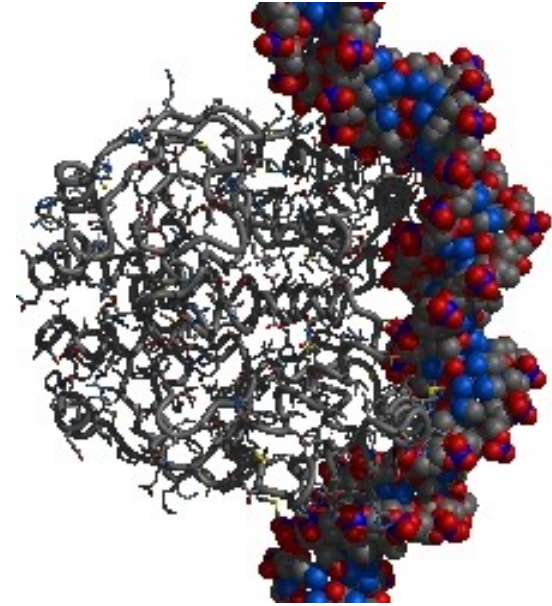


Applications of HMMs II: Other

- Molecule Folding: Prediction of structure based on sequence
- Pattern recognition, e.g. face emotion recognition



- Prediction tasks, e.g. stock market





coming up next

HMM, MEMM, CRF, LSTM

SEQUENCE TAGGING