

# CHAPTER 3: L2GO INFO

From

<https://lecture2go.uni-hamburg.de/l2go/-/get/v/24493> [1:20:00]

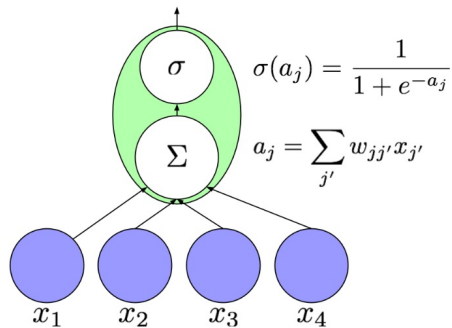
over

<https://lecture2go.uni-hamburg.de/l2go/-/get/v/24507> [full]

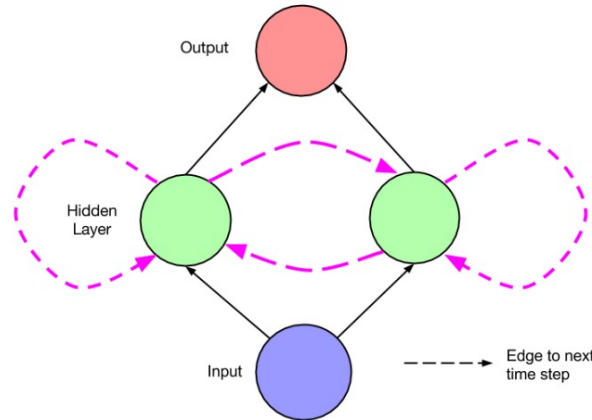
to

<https://lecture2go.uni-hamburg.de/l2go/-/get/v/24797> [23:15]

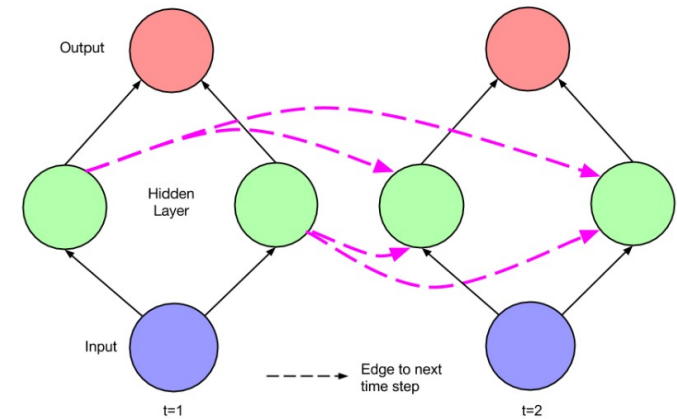
# RECURRENT NEURAL NETWORKS



Artificial  
neuron



Simple recurrent network with  
2 hidden units



Same recurrent network  
unfolded over time

- Recurrent connections: input at time step comes from activation at time step (t-1)
- Recurrent connections introduce notion of sequence into the network
- Unfolding: Can view recurrent network like a deep network, can apply gradient-based training

# NEURAL LANGUAGE MODELS (BENGIO ET AL., 2003)

## Summary of Approach:

1. associate with each word in the vocabulary  $V$  a distributed word feature vector: a real-valued vector in  $R^m$ , where  $m \ll |V|$  vocab size.
2. express the joint probability function of word sequences in terms of the feature vectors of these words in the sequence, and
3. learn simultaneously the word feature vectors (a.k.a. embeddings) and the parameters of that probability function.

Objective: learn a good model (low perplexity on held-out) for

$$f(w_t, \dots, w_{t-n+1}) = \hat{P}(w_t | w_1^{t-1})$$

subject to:

$$\sum_{i=1}^{|V|} f(i, w_{t-1}, \dots, w_{t-n+1}) = 1$$

# INTUITION: USE SIMILARITY BETWEEN REPRESENTATIONS

Assume these pairs are similar:

- dog – cat
- the – a
- room – bedroom
- is – was
- running – walking

Then, “The cat is walking in the bedroom” could transfer probability mass to:

- The cat is walking in the bedroom
- A dog was running in a room
- The cat is running in a room
- A dog is walking in a bedroom
- The dog was walking in the room
- ...

# TWO PARTS: EMBEDDING AND PREDICTION

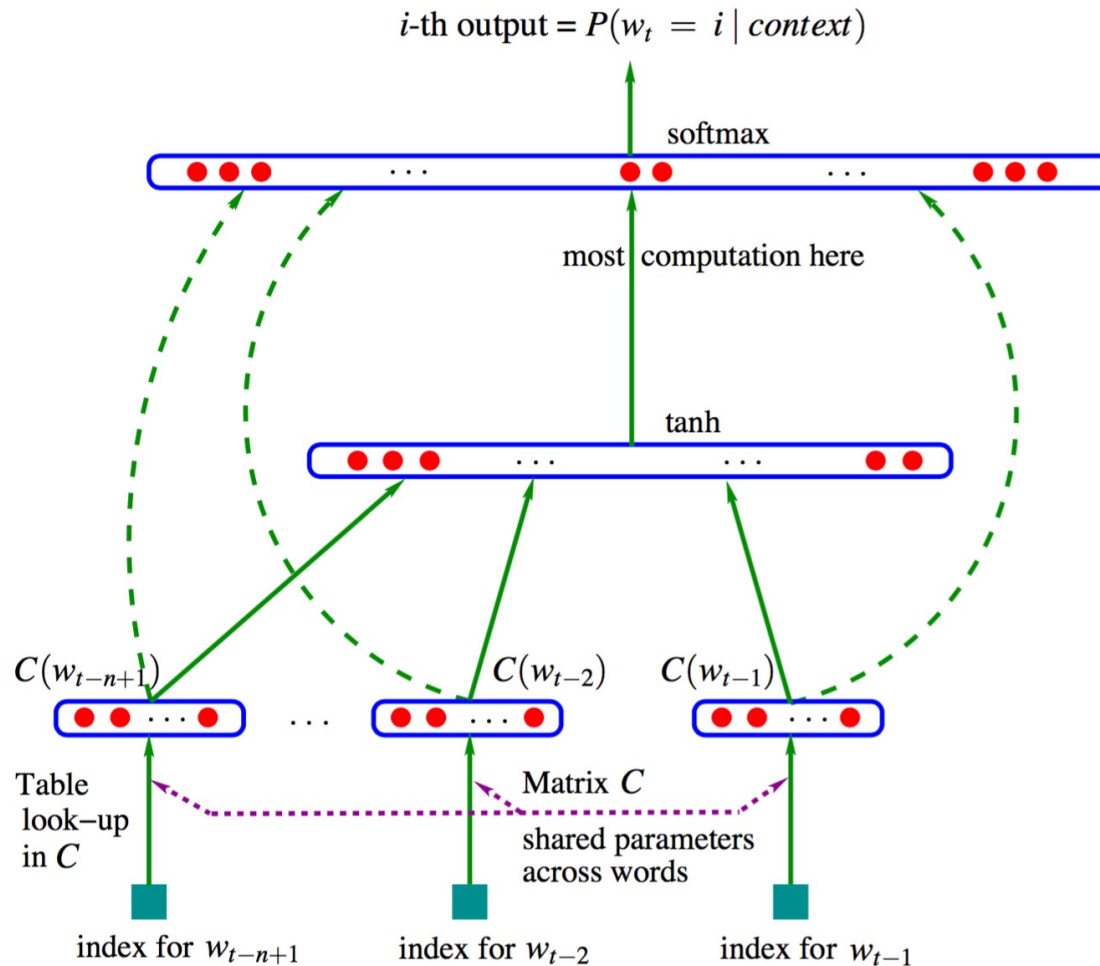
Function  $f(w_t, \dots, w_{t-n+1}) = \hat{P}(w_t | w_1^{t-1})$   
is decomposed in two parts:

1. A mapping  $C$  from any element  $i$  of  $V$  to a real vector  $C(i) \in R^m$ . It represents the distributed feature vectors associated with each word in the vocabulary. In practice,  $C$  is represented by a  $|V| \times m$  matrix of free parameters (dense vector embeddings).
2. The probability function over words, expressed with  $C$ : a function  $g$  maps an input sequence of feature vectors for words in context,  $(C(w_{t-n+1}), \dots, C(w_{t-1}))$ , to a conditional probability distribution over words in  $V$  for the next word  $w_t$ . The output of  $g$  is a vector whose  $i$ -th element estimates the probability

$$\hat{P}(w_t = i | w_1^{t-1})$$

$$f(w_t, \dots, w_{t-n+1}) = g(i, C(w_{t-1}), \dots, C(w_{t-n+1}))$$

# NEURAL ARCHITECTURE: NN-LM



$C(i)$  is  $i$ -th word feature vector;

“most computation here”: some neural network

Softmax normalizes  $P$ :

$$P(w_t \mid w_{t-1}, \dots, w_{t-n+1}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$

$y$ : un-normalized log-probs

$$y = b + Wx + U \tanh(d + Hx)$$

$b, d$ : biases

$W$ : words to output weights  
(direct connections)

$H$ : hidden layers weights

$U$ : hidden-to-output weights

$x$ : concatenation of  $C(w)$ 's

# TRAINING: FINDING THE RIGHT PARAMETER SET

- Overall parameter set:  $\theta=(C,\omega)$ , where  $\omega$  are the network's parameters and  $C$  is the embedding matrix
- Training: Maximize corpus likelihood  $L$ :

$$L = \frac{1}{T} \sum_t \log f(w_t, w_{t-1}, \dots, w_{t-n+1}; \theta) + R(\theta)$$

- $R(\theta)$ : Regularization term ( $\sim$ smoothing): prevent overfitting, here by weight decay penalty

Stochastic gradient ascent: iterative update with learning rate  $\varepsilon$ :

$$\theta \leftarrow \theta + \varepsilon \frac{\partial \log \hat{P}(w_t | w_{t-1}, \dots w_{t-n+1})}{\partial \theta}$$

# RESULTS ON THE BROWN CORPUS

	n	c	h	m	direct	mix	train.	valid.	test.
MLP1	5		50	60	yes	no	182	284	268
MLP2	5		50	60	yes	yes		275	257
MLP3	5		0	60	yes	no	201	327	310
MLP4	5		0	60	yes	yes		286	272
MLP5	5		50	30	yes	no	209	296	279
MLP6	5		50	30	yes	yes		273	259
MLP7	3		50	30	yes	no	210	309	293
MLP8	3		50	30	yes	yes		284	270
MLP9	5		100	30	no	no	175	280	276
MLP10	5		100	30	no	yes		265	<b>252</b>
Del. Int.	3						31	352	336
Kneser-Ney back-off	3							334	323
Kneser-Ney back-off	4							332	321
Kneser-Ney back-off	5							332	321
class-based back-off	3	150						348	334
class-based back-off	3	200						354	340
class-based back-off	3	500						326	<b>312</b>
class-based back-off	3	1000						335	319
class-based back-off	3	2000						343	326
class-based back-off	4	500						327	312
class-based back-off	5	500						327	312

Table 1: Comparative results on the Brown corpus. The deleted interpolation trigram has a test perplexity that is 33% above that of the neural network with the lowest validation perplexity. The difference is 24% in the case of the best n-gram (a class-based model with 500 word classes). *n* : order of the model. *c* : number of word classes in class-based n-grams. *h* : number of hidden units. *m* : number of word features for MLPs, number of classes for class-based n-grams. *direct*: whether there are direct connections from word features to outputs. *mix*: whether the output probabilities of the neural network are mixed with the output of the trigram (with a weight of 0.5 on each). The last three columns give perplexity on the training, validation and test sets.

- Best results for a mixture model: mixing NN-LM with KN-trigram
- Hidden Layer helps
- Direct connections not needed
- Low number of dimensions (30) for embeddings



# NOTE ON TRAINING TIMES AND HYPERPARAMETERS

Training time:

- traditional n-gram model: just count, then smooth.
- NN-LM: amount of computation for output probabilities is large: for obtaining a particular  $P(w_t/w_{t-1}, \dots, w_{t-n+1})$ , need all probabilities for all the words in the vocabulary
  - ➔ requires parallel processing

Hyperparameters: the 'art' of optimization

- learning rate
- epochs
- regularization
- # hidden units
- dimension
  - ➔ optimizing requires both experience and time

# RECURRENT NN-LM: 'INFINITE' HISTORY (MIKOLOV ET AL. 2010)

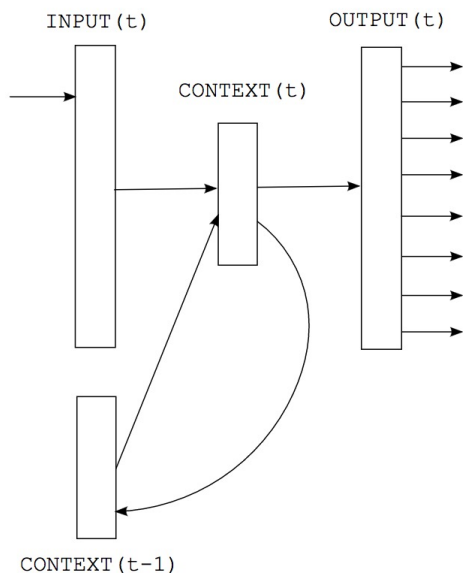


Table 1: *Performance of models on WSJ DEV set when increasing size of training data.*

Model	# words	PPL	WER
KN5 LM	200K	336	16.4
KN5 LM + RNN 90/2	200K	271	15.4
KN5 LM	1M	287	15.1
KN5 LM + RNN 90/2	1M	225	14.0
KN5 LM	6.4M	221	13.5
KN5 LM + RNN 250/5	6.4M	156	11.7

- Key idea: use a recurrent neural network
- A single 'context' vector (~300 dimensions) encodes 'all the history', computed from the previous context and the input
- input: again, word embedding
- output: again, softmax

# CONCLUSIONS ON NEURAL LANGUAGE MODELS

- Symbolic units (words) are transformed into continuous representations: dense vector embeddings
- good representations: similar words have similar vectors, allowing generalization and smoothing
- better performance than sparse n-gram models
- more compact representation in model application
- much more expensive training
- many more hyper-parameters

Neural Language models are becoming the standard in NLP; dense vector embeddings are also beneficial for word similarity tasks (stay tuned).

Current direction: contextualized embeddings.

- 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

- 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

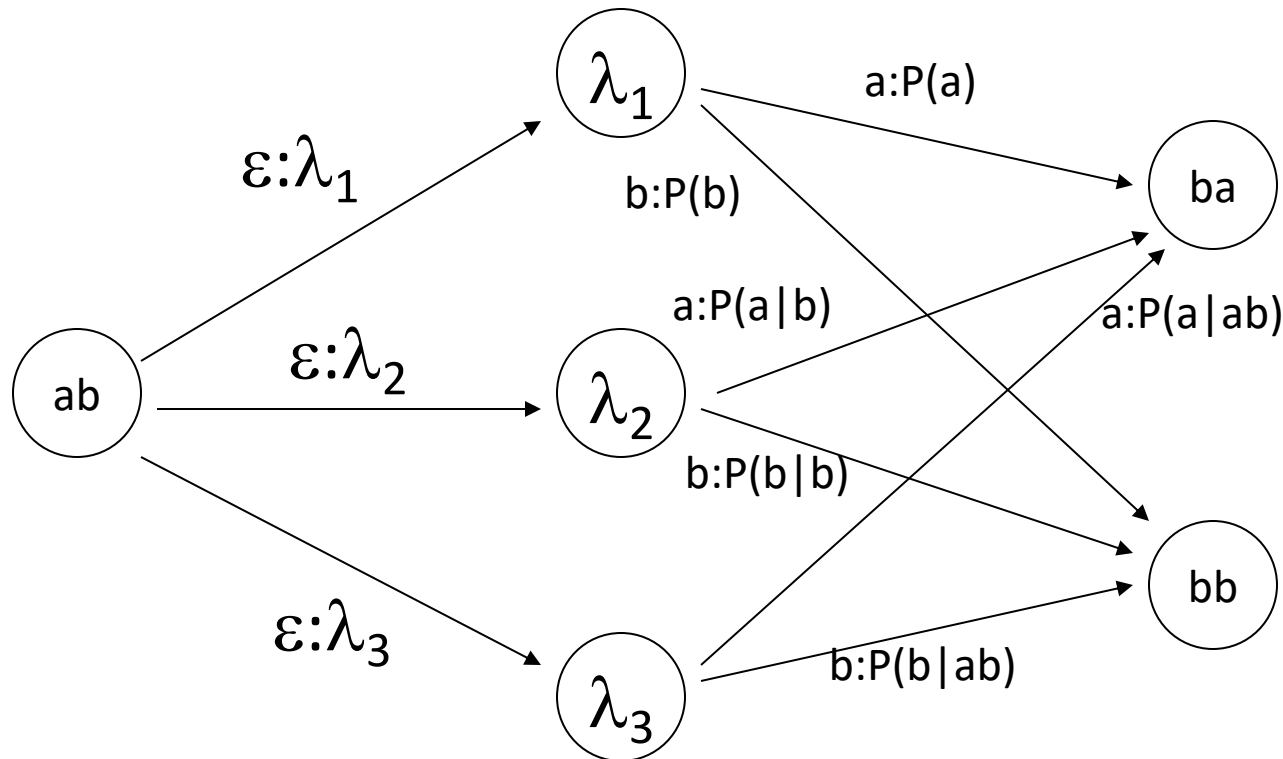
- Recap: Mixture Model:

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

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

# HIDDEN STATES

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



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

$$\begin{aligned} P(s^1 \dots s^n) &= \sum_{z^1 \dots z^{n+1}} P(s^1 \dots s^n, z^1 \dots z^{n+1}) \\ &= \sum_{z^1 \dots z^{n+1}} P(z^1) P(s^1, z^2 \mid z^1) P(s^2, z^3 \mid s^1, z^1 z^2) \dots P(s^n, z^{n+1} \mid s^1 \dots s^{n-1}, z^1 \dots z^n) \end{aligned} \quad (1)$$

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

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

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

- (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  $O(T \cdot N^T)$ !

# 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

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

## 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

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

## 2. Induction

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

## 3. Total

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

# 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:

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

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

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

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

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

## 3. Termination:

last state in *MAXPATH*:

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

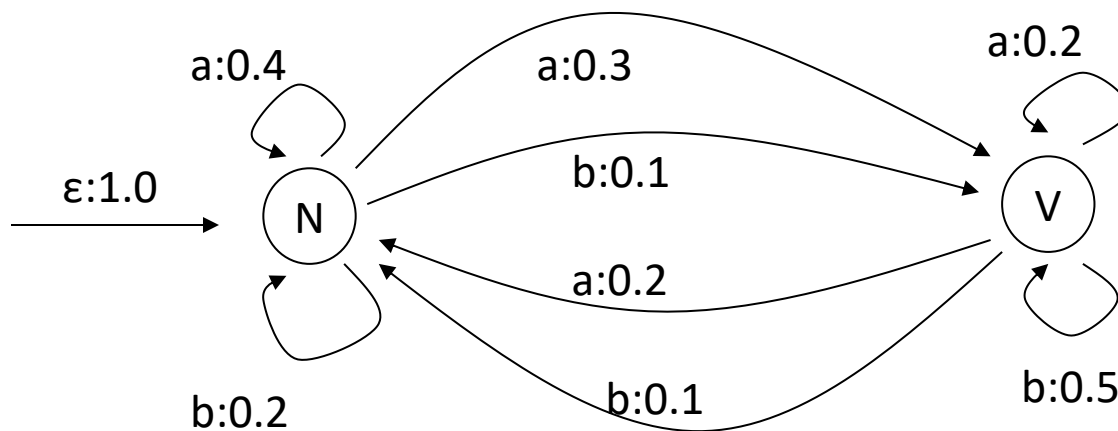
read sequence according to:

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

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



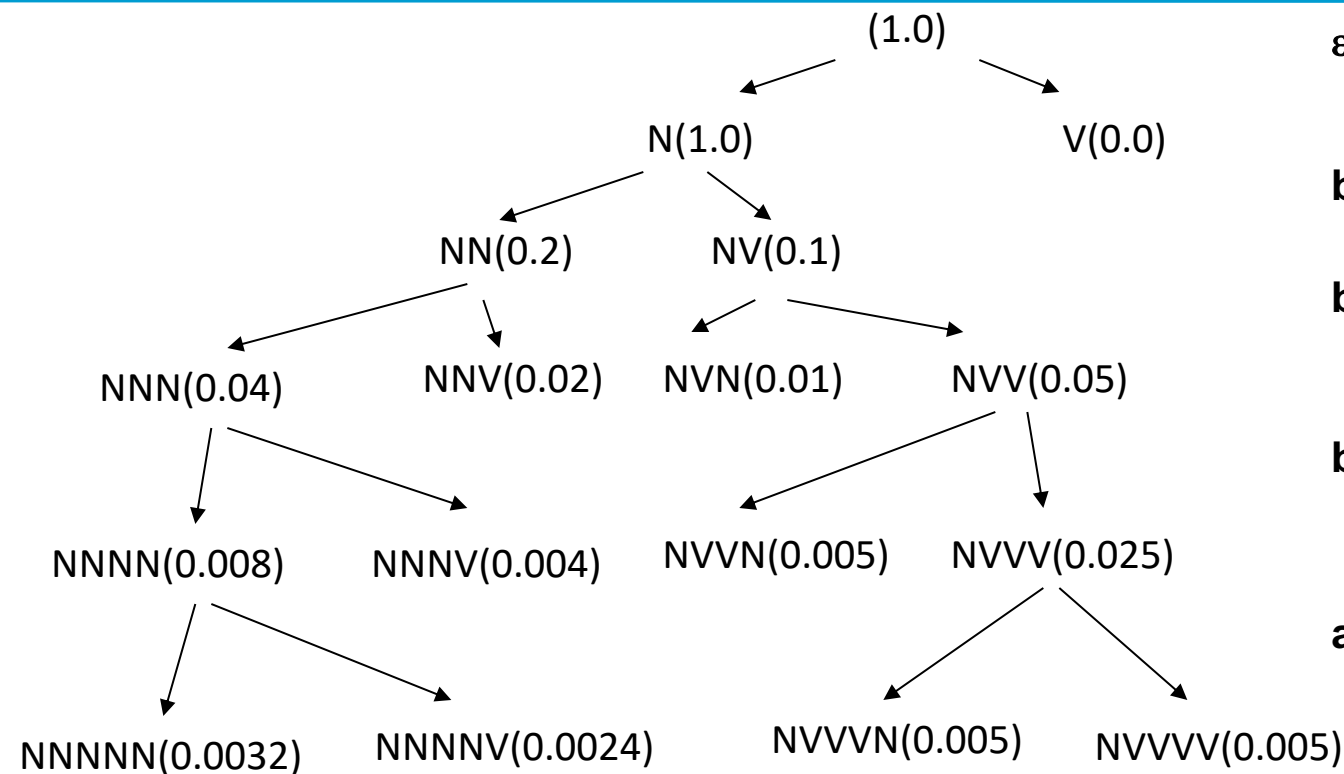
# VITERBI EXAMPLE



Observation sequence: bbba

input read	$\epsilon$	b	bb	bbb	bbba
N - sequence	N	NN	NNN	NNNN	NVVVN
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



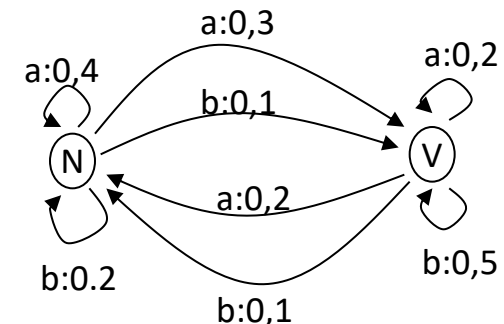
$\varepsilon$

**b**

**b**

**b**

**a**



Hopeless path prefixes are dropped from consideration

# HMM TRAINING

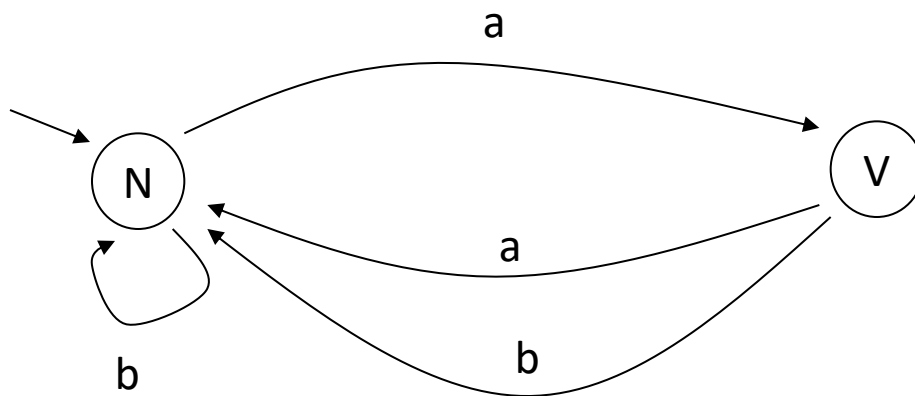
- Training: We have a fixed structure and optimize the parameters according to observations
- Name of training algorithm: **Baum-Welch** or **Forward-Backward algorithm**
- Idea:
  - start with random parameters
  - tune parameters such that the probability of the training sequence increases

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: abbaababbaaa

# 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

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

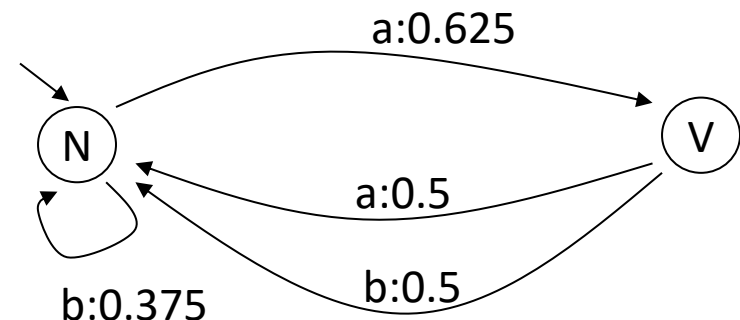
$$P_e(N \xrightarrow{b} N) = 3 / 8$$

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

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

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

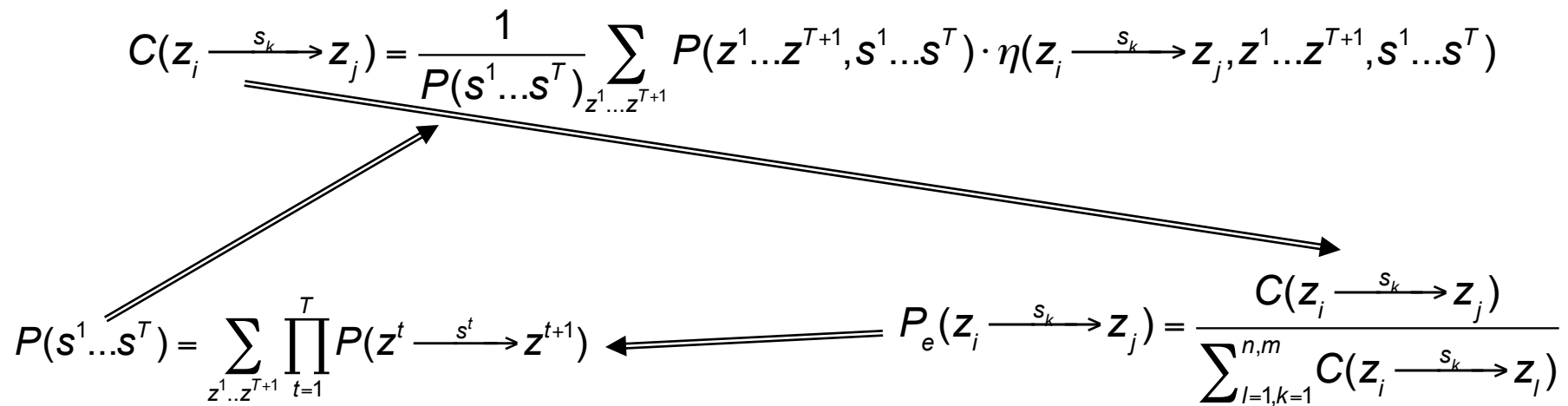
General case:

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

$\eta(z_i \xrightarrow{s_k} z_j, z^1 \dots z^{n+1}, s^1 \dots s^n)$  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

$$\begin{aligned}
 C(z_i \xrightarrow{s_k} z_j) &= \frac{1}{P(s^1 \dots s^T)} \sum_{z^1 \dots z^{T+1}} P(z^1 \dots z^{T+1}, s^1 \dots s^T) \cdot \eta(z_i \xrightarrow{s_k} z_j, z^1 \dots z^{T+1}, s^1 \dots s^T) \\
 P(s^1 \dots s^T) &= \sum_{z^1 \dots z^{T+1}} \prod_{t=1}^T P(z^t \xrightarrow{s^t} z^{t+1}) \\
 P_e(z_i \xrightarrow{s_k} z_j) &= \frac{C(z_i \xrightarrow{s_k} z_j)}{\sum_{l=1, k=1}^{n, m} C(z_i \xrightarrow{s_k} z_l)}
 \end{aligned}$$


Deadlock?

# SOLUTION: EXPECTATION MAXIMIZATION (EM)

Top-level EM algorithm:

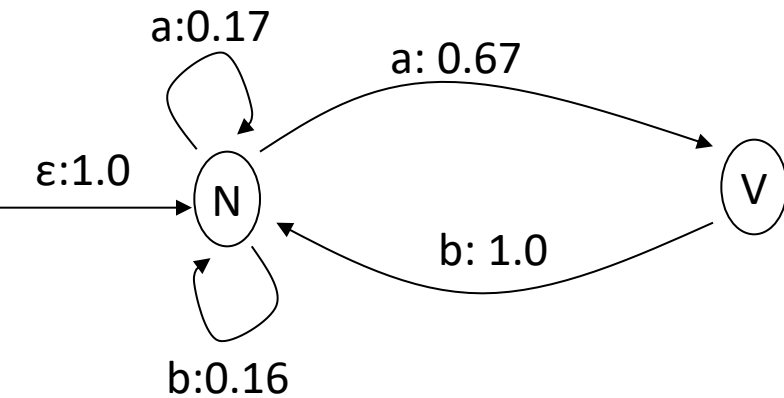
```
Old_Cross_Entropy=infinite;  
Guess HMM Parameters;  
New_Cross_Entropy=Re-estimate_Parameters();  
While not (Old_Cross_Entropy  $\approx$  New_Cross_Entropy) {  
    Old_Cross_Entropy=New_Cross_Entropy;  
    New_Cross_Entropy=Re-estimate_Parameters();  
}
```

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

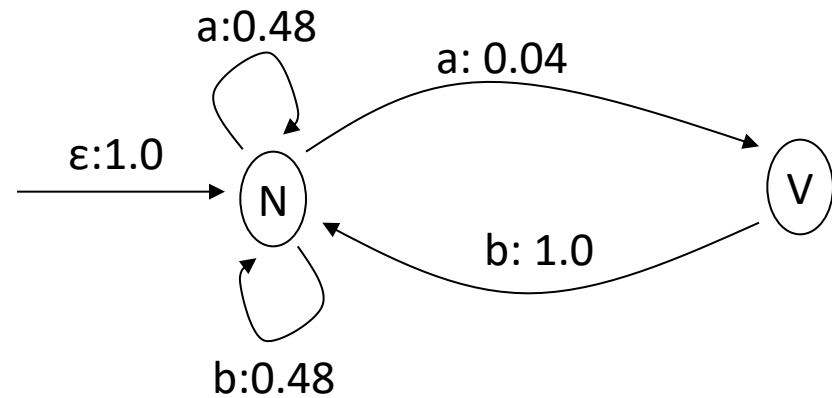


'true' HMM:  $P(ababb)=0.0778$

training sequence: `ababb`

possible paths:

- NVNVNN
- NVNNNN
- NNNVNN
- NNNNNN



initial random estimate

# 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.

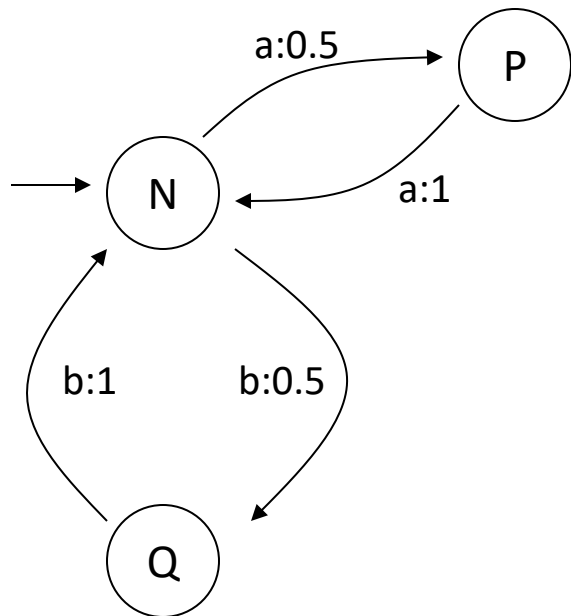
<i>path</i>	$P(\text{path})$	$P(V a,N)$	$P(N b,V)$	$P(N a,N)$	$P(N b,N)$	$\text{sum } P(.,N)$
NVNVNN	0.00209	0.00418	0.00418	0.0	0.00209	
NVNNNN	0.00727	0.00727	0.00727	0.00727	0.01454	
NNNVNN	0.00727	0.00727	0.00727	0.00727	0.01454	
NNNNNN	0.02529	0.0	0.0	0.05058	0.07587	
<i>sum over paths</i>	0.04192	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

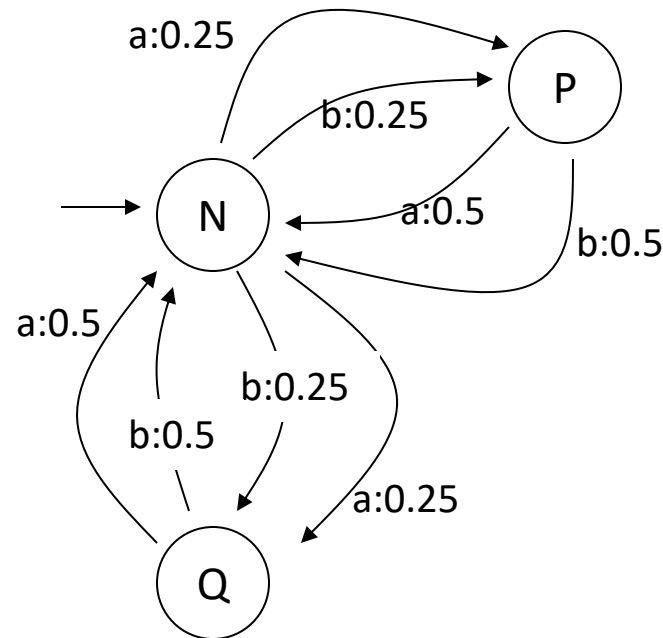
# PROBLEMS OF THIS TRAINING ALGORITHM

- 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  
 $P(aabb)=0.25$

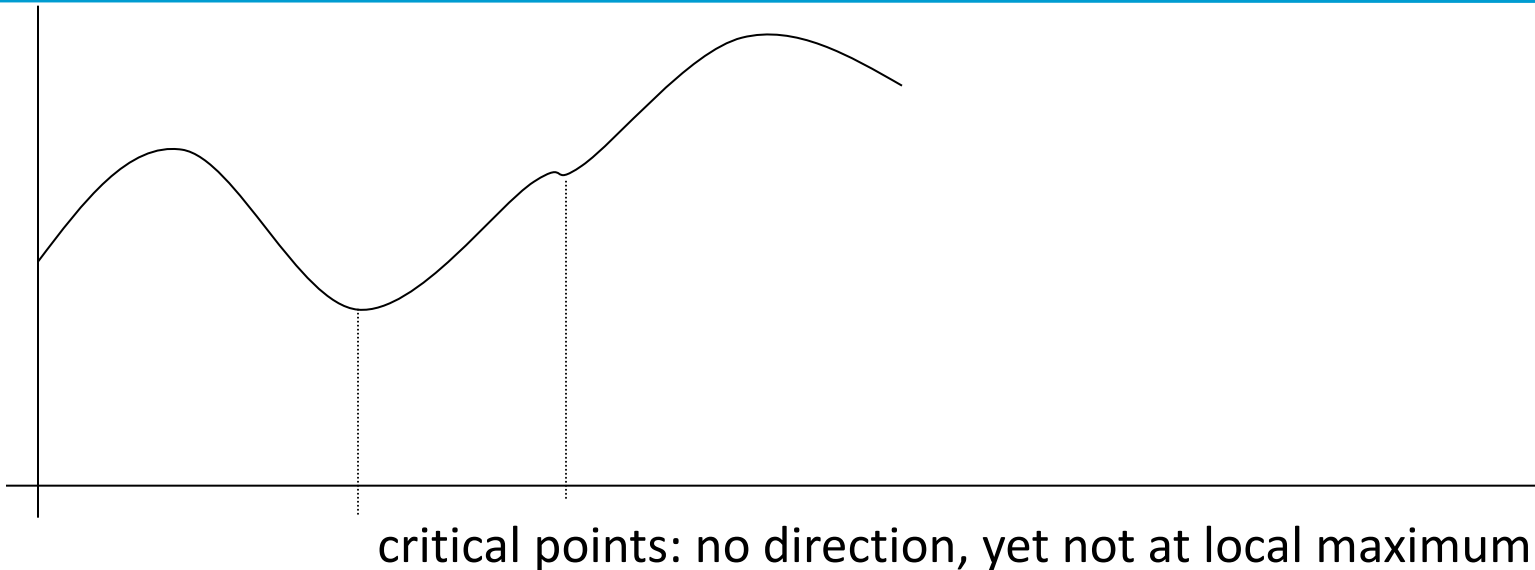


HMM with critical point  
 $P(aabb)=0.0625$

training sequence: aabb

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

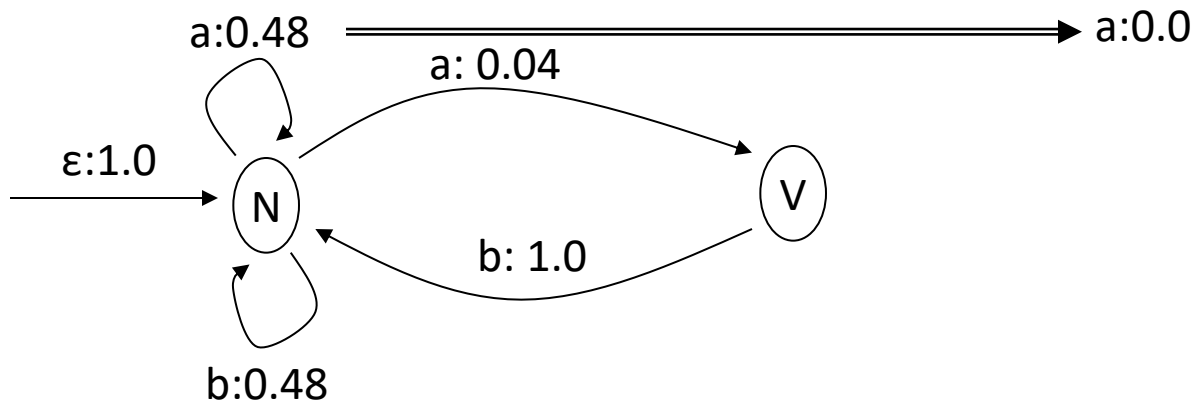
# DEALING WITH CRITICAL POINTS: ADD NOISE



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

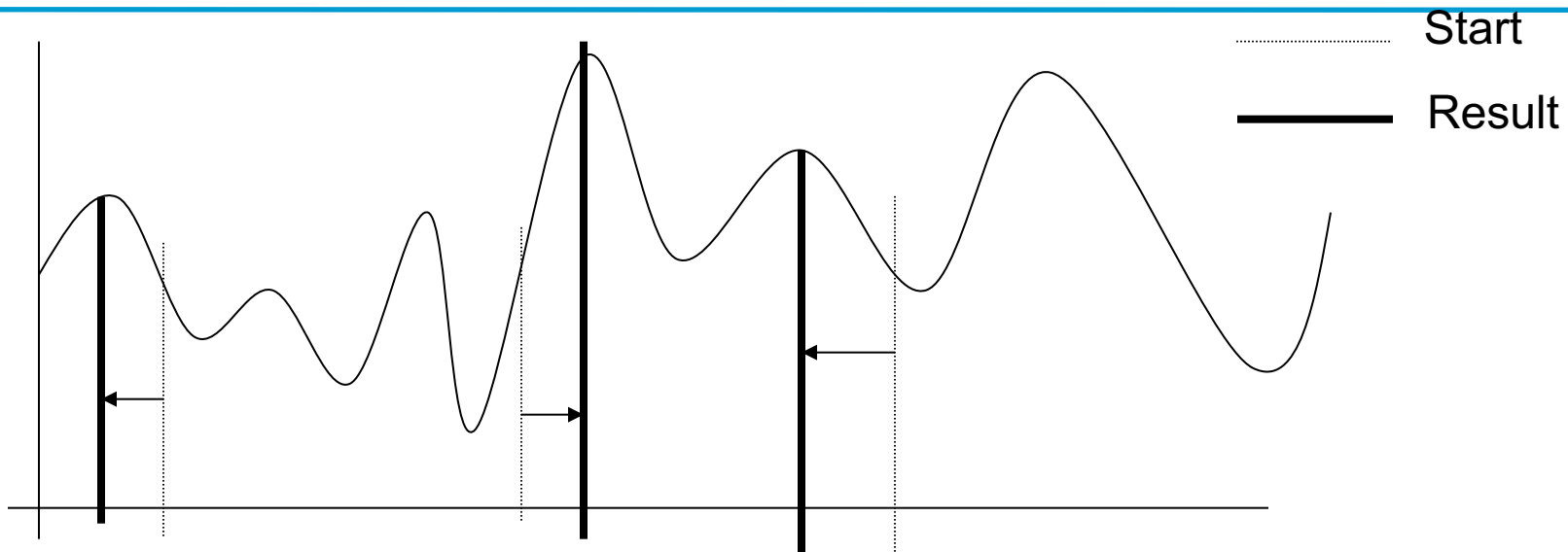
# WHAT ABOUT THE ZEROS?

- 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.

Recall the calculation of counts of transitions:

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

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

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

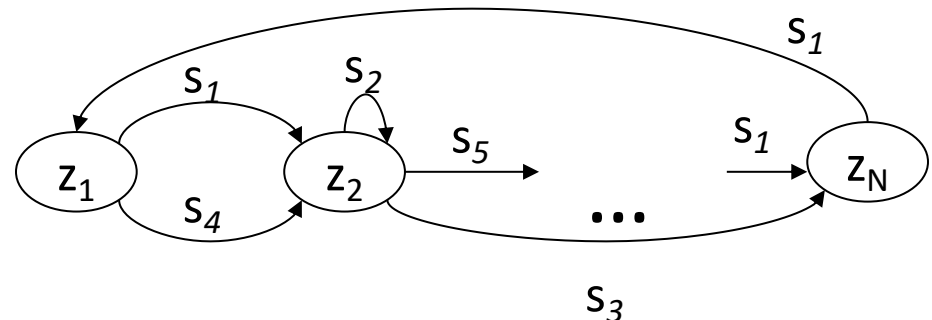
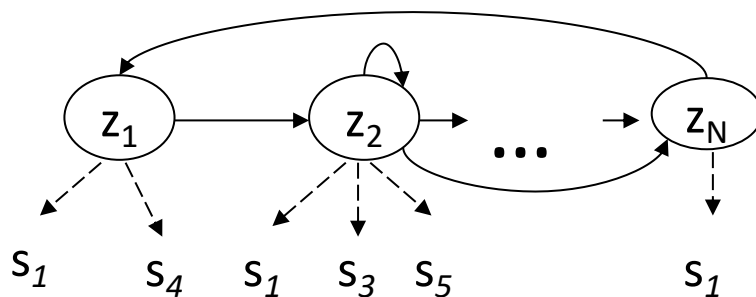
Using the combination  $P(s^1 \dots s^T) = \sum_{i=1}^N \alpha_i(t) \cdot \beta_i(t)$ , we reformulate as:

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



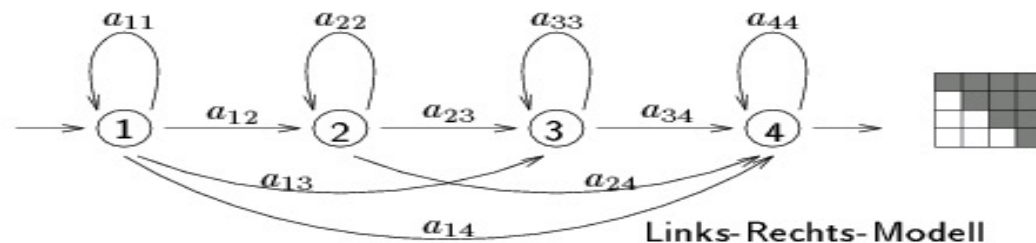
# VARIANTS OF HMMS

- 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

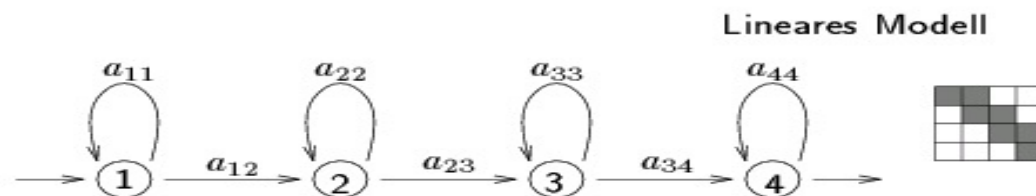
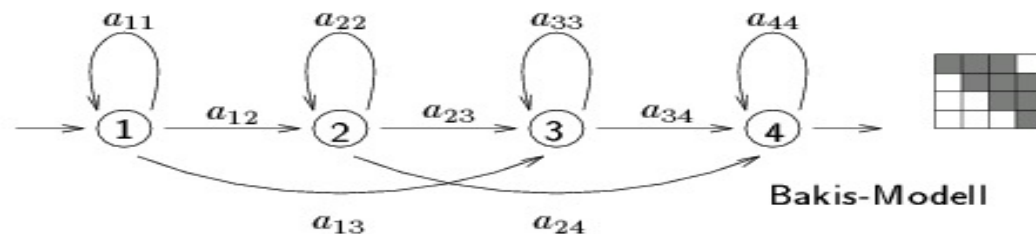


# APPLICATIONS OF HMMS I: LANGUAGE TECHNOLOGY

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



some HMM topologies  
for speech recognition



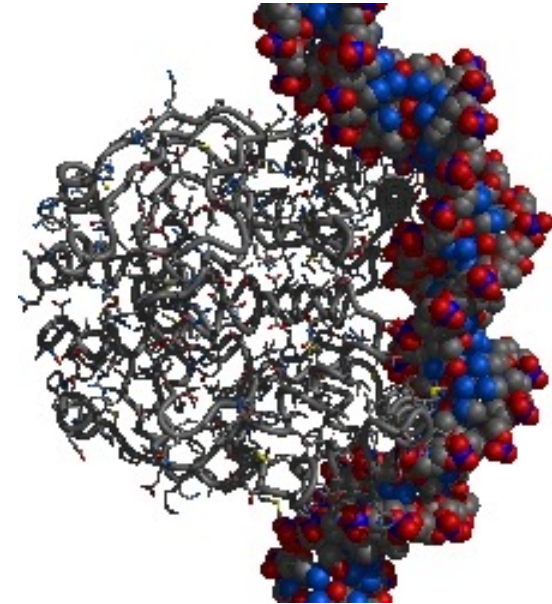
# APPLICATIONS OF HMMS II:

## 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