# Hidden Markov Models

**Mitch Marcus**

**CIS 421/521**

Penn
University of Pennsylvania

# An NLP task– Determining Part of Speech Tags

- **Given a text, assign each token its correct *part of speech (POS) tag*, given its context and a list of *possible* POS tags for each word type**

| Word | POS listing in Brown Corpus | | |
|---|---|---|---|
| heat | noun | ***verb*** | |
| oil | ***noun*** | | |
| in | ***prep*** | noun | adv |
| a | ***det*** | noun | noun-proper |
| large | ***adj*** | noun | adv |
| pot | ***noun*** | | |

# What is POS tagging good for?

- **Speech synthesis:**
  - How to pronounce "lead"?
  - INsult inSULT
  - OBject obJECT
  - OVERflow overFLOW
  - DIScount disCOUNT
  - CONtent content
- **Machine Translation**
  - translations of nouns and verbs are different
- **Stemming for search**
  - Knowing a word is a V tells you it gets past tense, participles, etc.
  - Can search for "walk", can get "walked", "walking,…

# Equivalent Problem in Bioinformatics

- **From a sequence of amino acids (primary structure): ATCPLELLLD**
- **Infer secondary structure (features of the 3D structure, like helices, sheets, etc.): HHHBBBBBC..**

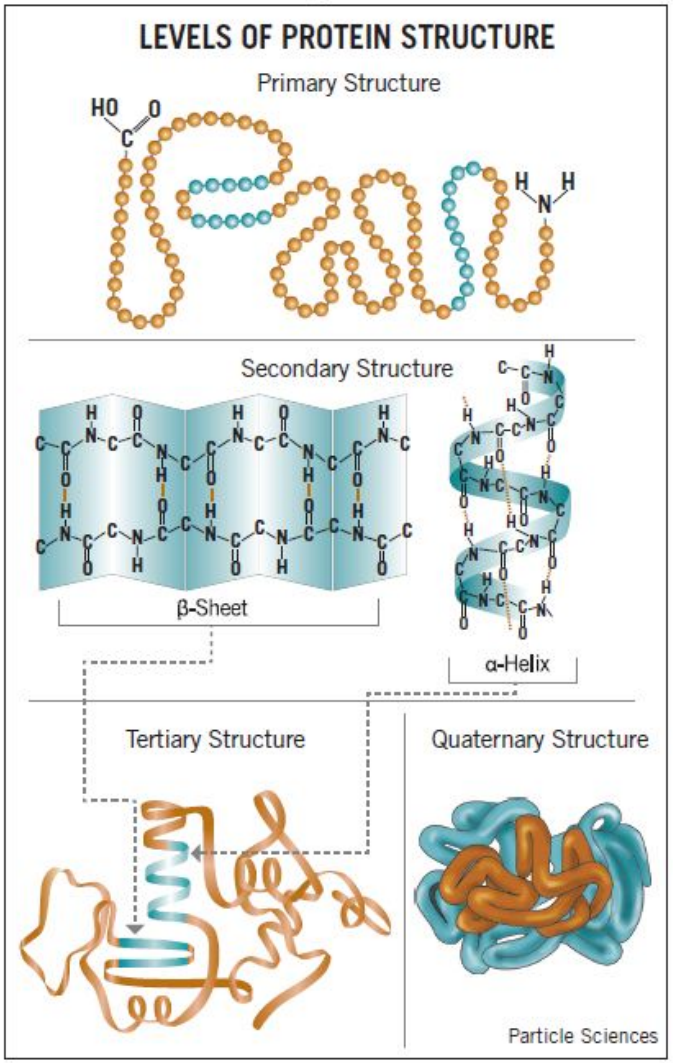



Figure from:
http://www.particlesciences.com/news/technical-briefs/2009/protein-structure.html

# Methods to Determine Part of Speech Tags

- **The Old Solution:** ***Depth First search.***
  - If each of *n* word tokens has *k* tags on average, try the $k^n$ combinations until one works.

- **Machine Learning Solutions:** ***Automatically learn*** **Part of Speech (POS) assignment.**
  - The best techniques achieve 97+% accuracy per word on new materials,
    - —given a POS-tagged training corpus of $10^6$ tokens
    - —with 3% inconsistency in the training corpus
    - —on a set of ~40 POS tags (Penn Treebank Tag Set)

# Statistical Approaches: Idea 1

Simply assign each word its most likely POS.

Success rate: 91%!

| Word | POS listings in Brown | | |
|---|---|---|---|
| heat | noun/89 | **verb/5** | |
| oil | **noun/87** | | |
| in | **prep/20731** | noun/1 | adv/462 |
| a | **det/22943** | noun/50 | noun-proper/30 |
| large | **adj/354** | noun/2 | adv/5 |
| pot | **noun/27** | | |

# Statistical Approaches: Idea 2

**For a string of words**

$$W = w_1 w_2 w_3 \ldots w_n$$

**find the string of POS tags**

$$T = t_1\, t_2\, t_3 \ldots t_n$$

**which maximizes $P(T|W)$**

- i.e., the most likely POS tag $t_i$ for each word $w_i$ given its surrounding context

# The Sparse Data Problem …

**A Simple, *Impossible* Approach to Compute $P(T \mid W)$:**

1. **Find all instances of the string "heat oil in a large pot" in a *tagged* training corpus**
2. **Pick the *most common tag assignment* for that string.**

# One more time: A BOTEC Estimate of What Works

## What parameters can we estimate with a million words of hand tagged training data?

- Assume the best case: a distribution of 5000 words and 40 part of speech tags with a distribution not far from uniform

| Event | Count | Estimate Quality? |
|---|---|---|
| tags | 40 | Excellent |
| tag bigrams | 1600 | Excellent |
| tag trigrams | 64,000 | OK |
| tag 4-grams | 2.5M | Poor |
| words | 5000 | Very Good |
| word bigrams | 25M | Poor |
| word $x$ tag pairs | 200,000 | OK |

**We can get reasonable estimates of**

- ***Tag* bigrams**
- ***Word x tag pairs***

# Review: Bayes' Rule

- Two ways to factor a joint distribution over two variables:

$$P(x, y) = P(x|y)P(y) = P(y|x)P(x)$$

- Dividing, we get:

$$P(x|y) = \frac{P(y|x)}{P(y)}P(x)$$

- Why is this at all helpful?
  - Lets us build one conditional from its reverse
  - Often one conditional is tricky but the other one is simple
  - Foundation of many systems we'll see later (e.g. ASR, MT)

- In the running for most important AI equation!

# Bayes Rule plus Markov Assumptions yields a practical POS tagger!

I. **By Bayes Rule** $$P(T \mid W) = \frac{P(W \mid T) * P(T)}{P(W)}$$

where $W = w_1 w_2 w_3 \ldots w_n$, $T = t_1 t_2 t_3 \ldots t_n$

II. **So we want to find**

$$\arg\max_T P(T \mid W) = \arg\max_T P(W \mid T) * P(T)$$

III. **To compute *P(W|T):***

- Use the chain rule + a Markov assumption (like last time!!)

IV. **To compute *P(T):***

- Use the chain rule + a slightly different Markov assumption

# II. Compute P(T) just like P(W) last lecture

I. **By the chain rule,**

$$P(T) = P(t_1) * P(t_2 \mid t_1) * P(t_3 \mid t_1 t_2) * \ldots * P(t_n \mid t_1 \ldots t_{n-1})$$

II. **Applying the 1st order Markov Assumption**

$$P(T) = P(t_1) * P(t_2 \mid t_1) * P(t_3 \mid t_2) * \ldots * P(t_n \mid t_{n-1})$$

# III. To compute *P(W|T):*

I. **Assume that the words $w_i$ are conditionally independent given the tag sequence $T=t_1 t_2 \ldots t_n$:** $P(W \mid T) = \prod_{i=1}^{n} P(w_i \mid T)$

II. **Applying a zeroth-order Markov Assumption:**

$$P(w_i \mid T) = P(w_i \mid t_i)$$

**by which** $$P(W \mid T) = \prod_{i=1}^{n} P(w_i \mid t_i)$$

**So, for a given string W = $w_1 w_2 w_3 \ldots w_n$, the tagger needs to *find the string of tags T which maximizes***

$$P(T) * P(W|T) = P(t_1) * P(t_2|t_1) * P(t_3|t_2) * \ldots * P(t_n|t_{n-1}) * P(w_1|t_1) * P(w_2|t_2) * \ldots * P(w_n|t_n)$$

# Training and Performance

- **To estimate the parameters of this model, given an annotated training corpus use the MLE:**

**To estimate $P(t_i|t_{i-1})$:**

$$\frac{Count(t_{i-1}t_i)}{Count(t_{i-1})}$$

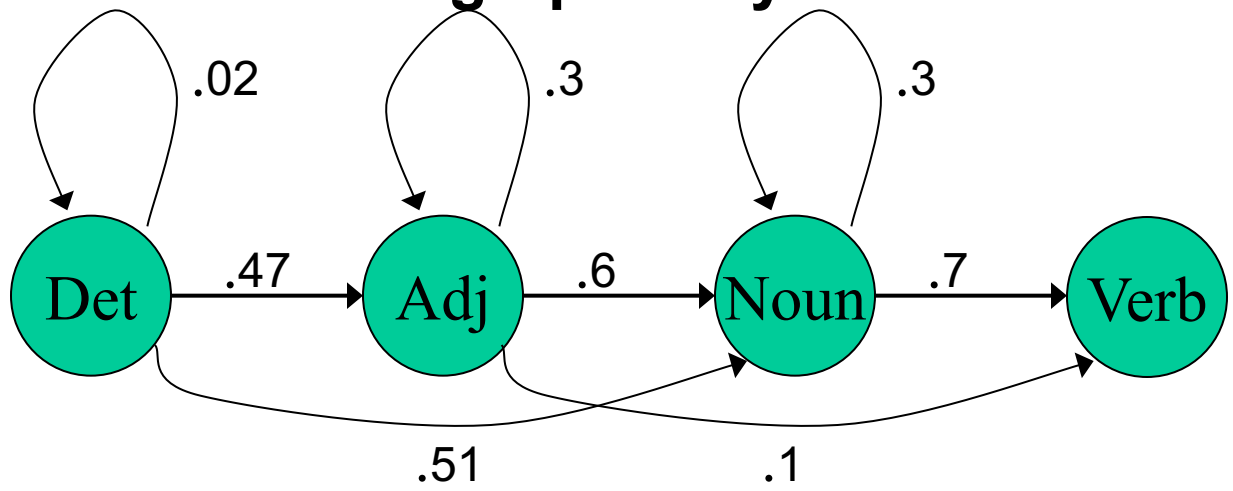
**To estimate $P(w_i|t_i)$:**

$$\frac{Count(\ w_i \text{ tagged } t_i)}{Count(\text{ all words tagged } t_i)}$$

- **Because many of these counts are small, *smoothing* is necessary for best results…**
- **Such taggers typically achieve about 95-96% correct tagging,** for the standard 40-tag POS set.
- A few tricks for unknown words increase accuracy to 97%.

# Hidden Markov Models

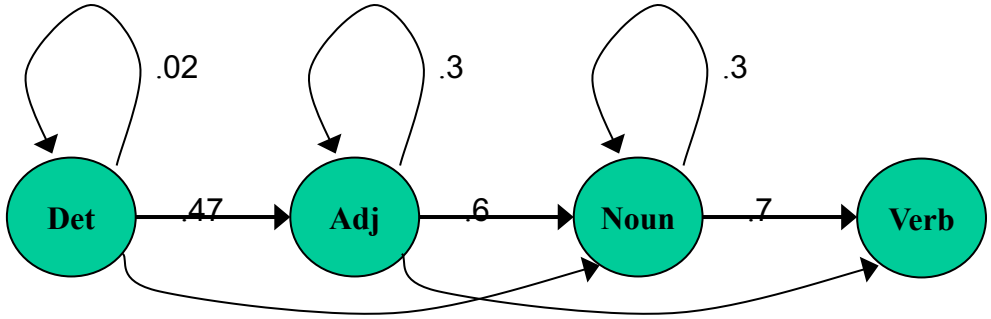
**This model is an instance of a Hidden Markov Model. Viewed graphically:**




| P(w\|Det) | | P(w\|Adj) | | P(w\|Noun) | |
|---|---|---|---|---|---|
| a | .4 | good | .02 | price | .001 |
| the | .4 | low | .04 | deal | .0001 |

# Viewed as a generator, an HMM:

- Starts in some initial state $t_1$ with probability $\pi(t_1)$,
- On each move goes from state $t_i$ to state $t_j$ according to transition probability $a(t_i, t_j)$.
- At each state $t_i$, it emits a symbol $w_k$ according to the emit probabilities $b(t_i, w_k)$.




| P(w\|Det) | |
|---|---|
| a | .4 |
| the | .4 |

| P(w\|Adj) | |
|---|---|
| good | .02 |
| low | .04 |

| P(w\|Noun) | |
|---|---|
| price | .001 |
| deal | .0001 |

# Summary: Recognition using an HMM

I. **By Bayes Rule**

$$P(T \mid W) = \frac{P(T) * P(W \mid T)}{P(W)}$$

II. **We select the Tag sequence T that maximizes** $P(T|W)$:

$$\arg\max_{T} P(T \mid W)$$

$$= \arg \max_{T=t_1 t_2 \ldots t_n} P(T) * P(W \mid T)$$

$$= \arg \max_{T=t_1 t_2 \ldots t_n} \pi(t_1) * \prod_{i=1}^{n-1} a(t_i, t_{i+1}) * \prod_{i=1}^{n} b(t_i, w_i)$$

# Practical Tagging using HMMs

- **Finding this maximum can be done using an exponential search through all strings for T.**

- **However, there is a *linear time* solution using *dynamic programming* called *Viterbi decoding*.**

# The three basic HMM problems

*(These slides follow the classic formulation by Ferguson, as published by Rabiner and Juang, as adapted by Manning and Schutze.*

*Note the change in notation from the last lecture!!)*

# Parameters of an HMM

- ***States***: A set of states $S=s_1, \ldots s_n$
- ***Transition probabilities***: $A= a_{1,1}, a_{1,2}, \ldots, a_{n,n}$ Each $a_{i,j}$ represents the probability of transitioning from state $s_i$ to $s_j$.
- ***Emission probabilities:*** a set $B$ of functions of the form $b_i(o_t)$ which is the probability of observation $o_t$ being emitted by $s_i$
- ***Initial state distribution***: $\pi_i$ is the probability that $s_i$ is a start state

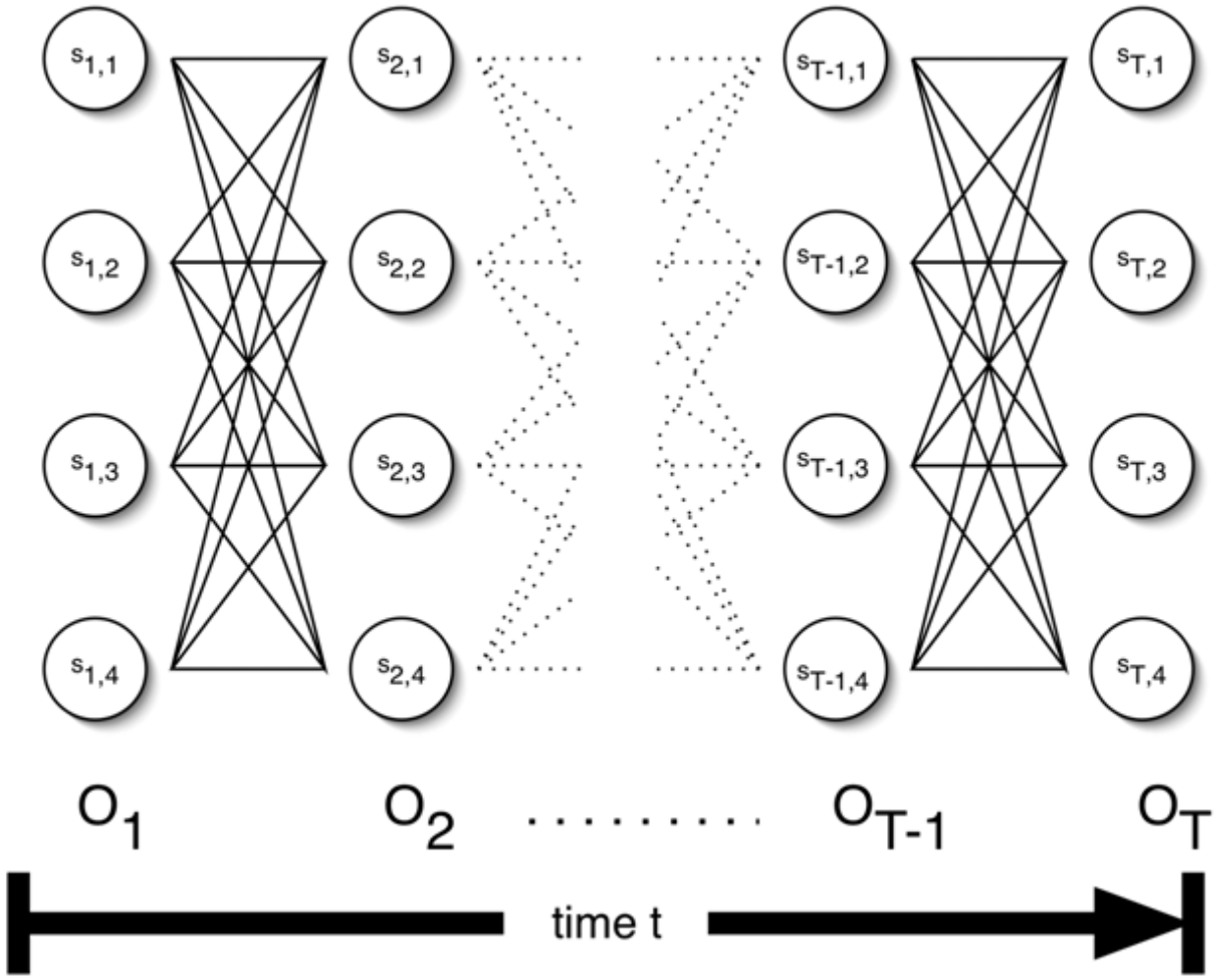
# The Three Basic HMM Problems

- ***Problem 1 (Evaluation):*** **Given the observation sequence $O=o_1,\ldots,o_T$ and an HMM model $\lambda = (A,B,\pi)$, how do we compute the probability of $O$ given the model?**
- ***Problem 2 (Decoding):*** **Given the observation sequence $O$ and an HMM model $\lambda$, how do we find the state sequence that best explains the observations?**
- ***Problem 3 (Learning):*** **How do we adjust the model parameters $\lambda = (A,B,\pi)$, to maximize $P(O \mid \lambda)$?**

# Problem 1: Probability of an Observation Sequence

- **Q: What is $P(O \mid \lambda)$ ?**
- **A: the sum of the probabilities of all possible state sequences in the HMM.**
- **Naïve computation is very expensive. Given $T$ observations and $N$ states, there are $N^T$ possible state sequences.**
  - (for $T$=10 and $N$=10, 10 billion different paths!!)
- **Solution: linear time dynamic programming!**

# *The Crucial Data Structure:* The Trellis



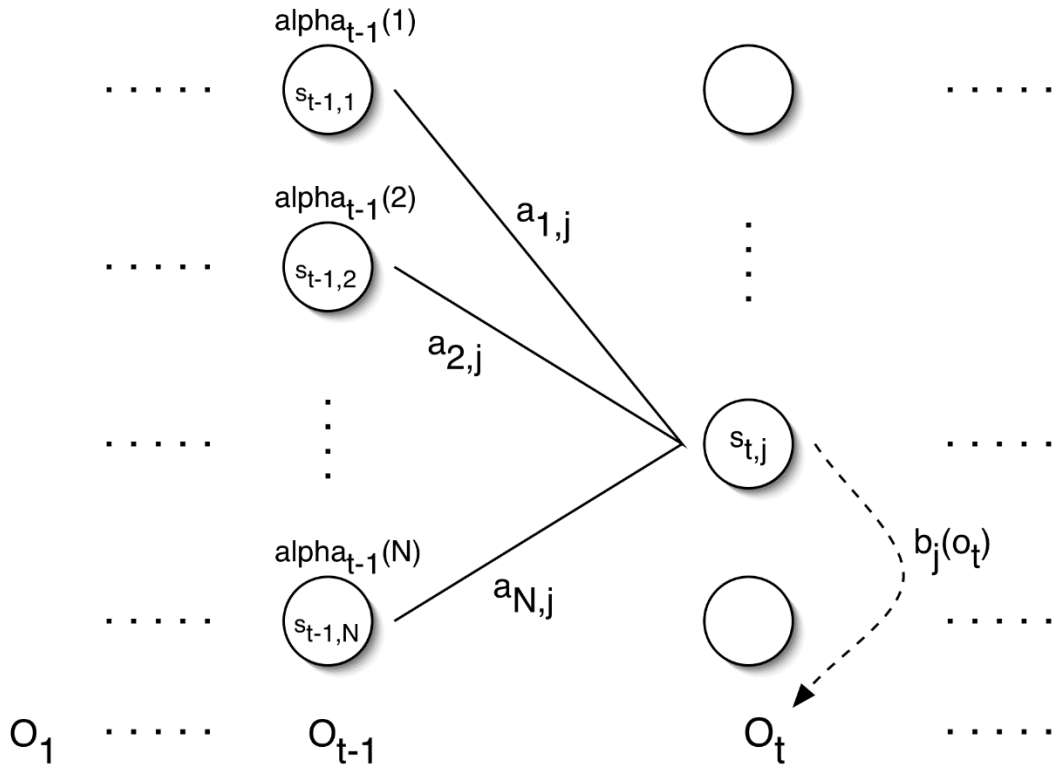
# Forward Probabilities: α

- For a given HMM $\lambda$, for some time $t$, what is the probability that the partial observation $o_1 \ldots o_t$ ***has been*** generated and that the state at time $t$ is $i$ ?

$$\alpha_t(i) = P(o_1 \ldots o_t, q_t = s_i \mid \lambda)$$

- ***Forward algorithm computes $\alpha_t(i)$ $0<i<N$, $0<t<T$ in time $0(N^2T)$ using the trellis***

# Forward Algorithm: Induction step




$$\alpha_t(j) = \left[ \sum_{i=1}^{N} \alpha_{t-1}(i)\, a_{ij} \right] b_j(o_t)$$

# Forward Algorithm

- **Initialization (probability that $o_1$ has been generated and that the state is $i$ at time $t=1$:**

$$\alpha_1(i) = \pi_i b_i(o_1) \quad 1 \le i \le N$$

- **Induction:**

$$\alpha_t(j) = \left[ \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} \right] b_j(o_t) \quad 2 \le t \le T, \quad 1 \le j \le N$$

- **Termination:**

$$P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

# Forward Algorithm Complexity

- Naïve approach requires exponential time to evaluate all $N^T$ state sequences
- Forward algorithm using dynamic programming takes $O(N^2T)$ computations

# Problem 2: Decoding

- The Forward algorithm gives the *sum of all paths* through an HMM efficiently.
- Here, we want to find the *highest probability path*.
- We want to find the state sequence $Q=q_1 \ldots q_T$, such that

$$Q = \arg\max_{Q'} P(Q' \mid O, \lambda)$$

# Viterbi Algorithm: Recursion step

- **Just like the Forward algorithm, but instead of *summing* over transitions from incoming states, compute the *maximum***

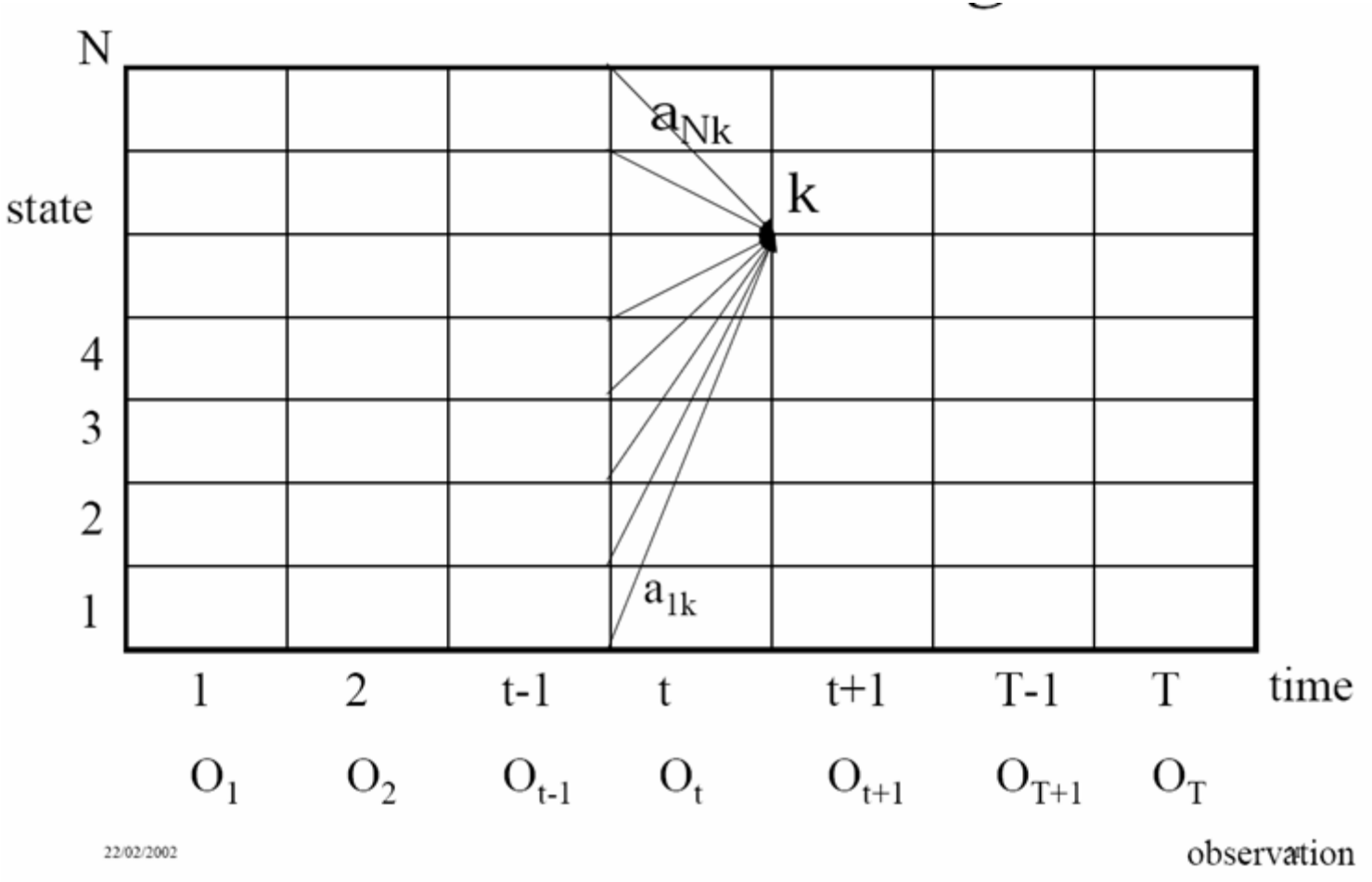
- **Forward:**

$$\alpha_t(j) = \left[ \sum_{i=1}^{N} \alpha_{t-1}(i)\, a_{ij} \right] b_j(o_t)$$

- **Viterbi Recursion:**

$$\delta_t(j) = \left[ \max_{1 \le i \le N} \delta_{t-1}(i)\, a_{ij} \right] b_j(o_t)$$

# Core Idea of Viterbi Algorithm



# Not quite what we want….

- **Viterbi recursion computes the *maximum probability* to state *j* at time *t* given that the partial observation $o_1 \ldots o_t$ *has been* generated**

$$\delta_t(j) = \left[ \max_{1 \le i \le N} \delta_{t-1}(i)\, a_{ij} \right] b_j(o_t)$$

- **But we want the *path itself* that gives the maximum probability**
- ***Solution:***
  1. ***Keep backpointers***
  2. ***Find*** $\arg\max_j \delta_T(j)$***, i.e. the most probable final state***
  3. ***Chase backpointers from state j at time T to find state sequence (backwards)***

# Viterbi Algorithm (For Notes)

- **Initialization:** $\delta_1(i) = \pi_i b_j(o_1) \quad 1 \le i \le N$
- **Induction:**

$$\delta_t(j) = \left[ \max_{1 \le i \le N} \delta_{t-1}(i)\, a_{ij} \right] b_j(o_t) \qquad 2 \le t \le T, 1 \le j \le N$$

$$\psi_t(j) = \left[ \arg\max_{1 \le i \le N} \delta_{t-1}(i) a_{ij} \right] \quad \textit{(Backpointers)}$$

- `
- **Termination:** $q_T^* = \arg\max_{1 \le i \le N} \delta_T(i)$ *(Final state!)*
- **Backpointer path:** $q_t^* = \psi_{t+1}(q_{t+1}^*) \quad t = T-1,\ldots,1$