HMM (2): Decoding

LING 570

Fei Xia

Three fundamental questions for HMMs

- Training an HMM: given a set of observation sequences, learn its distribution, i.e., learn the transition and emission probabilities
- HMM as a parser: Finding the best state sequence for a given observation
- HMM as an LM: compute the probability of a given observation

Training an HMM: estimating the probabilities

- Supervised learning:
  - The state sequences in the training data are known
  - ML estimation

- Unsupervised learning:
  - The state sequences in the training data are unknown
  - forward-backward algorithm

Dynamic programming

The meaning of “**programming**”?

“programming” ≠ “computer programming”

This programming is **optimization/solving**, often max / arg max

- Linear programming: optimize linear function
- Quadratic programming: optimize quadratic function
- Dynamic programming: optimize by divide and conquer

Where did the name, dynamic programming, come from? (Bellman's 1984 book)

"I spent the Fall quarter (of 1950) at RAND … The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word, research. I'm not using the term lightly; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term, research, in his presence. You can imagine how he felt, then, about the term, mathematical. The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose? …. It was something not even a Congressman could object to. So I used it as an umbrella for my activities."

http://en.wikipedia.org/wiki/Dynamic_programming

Where DP is applicable?

A problem must have two key attributes in order to apply DP:

- Optimal substructure: the solution to the problem can be obtained by the combination of optimal solutions to its subproblems.

  => Divide and conquer

  => Can use recursive functions

- Overlapping subproblems: any recursive problem would solve the same subproblem many times.

  => Memorize the solution to the subproblem

DP: Fibonacci numbers

- $f(0) \coloneqq 1$
- $f(1) \coloneqq 1$
- $\forall n \geq 2 . f(n) \coloneqq$ f(n-1) + f(n-2)

Fibonacci in Python

```
#!/usr/bin/python

def fib(n):
    if n == 0: return 1
    if n == 1: return 1
    return fib(n-1) + fib(n-2)

print fib(40)
```

Runtime

```
$ time ./fib.py
165580141

real	1m20.136s
user	1m20.076s
sys	0m0.038s
```

$$\begin{tikzcd}
 & & & & \mathrm{Fib}(5) \arrow[dll] \arrow[drr] & & & \\
 & & \mathrm{Fib}(4) \arrow[dl] \arrow[dr] & & & & \mathrm{Fib}(3) \arrow[dl] \arrow[dr] & \\
 & \mathrm{Fib}(3) \arrow[dl] \arrow[dr] & & \mathrm{Fib}(2) \arrow[d] \arrow[dr] & & \mathrm{Fib}(2) \arrow[d] \arrow[dr] & & \mathrm{Fib}(1) \\
\mathrm{Fib}(2) \arrow[d] \arrow[dr] & & \mathrm{Fib}(1) & \mathrm{Fib}(1) & \mathrm{Fib}(0) & \mathrm{Fib}(1) & \mathrm{Fib}(0) & \\
\mathrm{Fib}(1) & \mathrm{Fib}(0) & & & & & &
\end{tikzcd}$$

Dynamic programming solution

```
#!/usr/bin/python

def fib(n):
  x = []
  x.append(1)
  x.append(1)
  for i in range(2, n + 1):
    x.append(x[i-2] + x[i-1])
  return x[n]

print fib(40)
```

```
$ time ./fib_dp.py
165580141


real    0m0.020s
user    0m0.013s
sys     0m0.007s
```

Algorithms that use DP

- Find longest common subsequence of two strings
- Calculate edit distance between two strings
- CYK algorithm for CFG parsing
- Viterbi algorithm
- …

HMM as a parser

PFA: Finding the best path for input x

- Read Section 3.2 of the 2005 PFA paper.

$$\tilde{\theta} = \operatorname*{argmax}_{\theta \in \Theta_{\mathcal{A}}(x)} \Pr_{\mathcal{A}}(\theta).$$

The computation of $\Pr_{\mathcal{A}}(x)$ can be efficiently performed by defining a function $\gamma_x(i, q)$ $\forall q \in Q$, $0 \leq i \leq |x|$, as the probability of generating the prefix $x_1 \ldots x_i$ through the best path and reaching state q:

$$\gamma_x(i,q) = \max_{(s_0,s_1,\ldots,s_i)\in\Theta_A(x_1\ldots x_i)} I(s_0)\cdot\prod_{j=1}^{i} P(s_{j-1},x_j,s_j)\cdot 1(q,s_i)$$

where $1(q,q') = 1$ if $q = q'$ and 0 if $q \neq q'$.

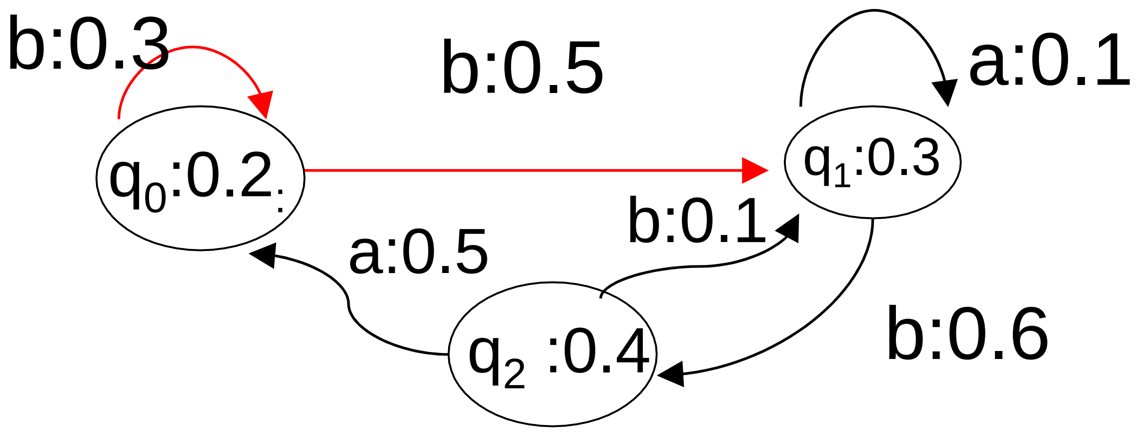
Viterbi algorithm:

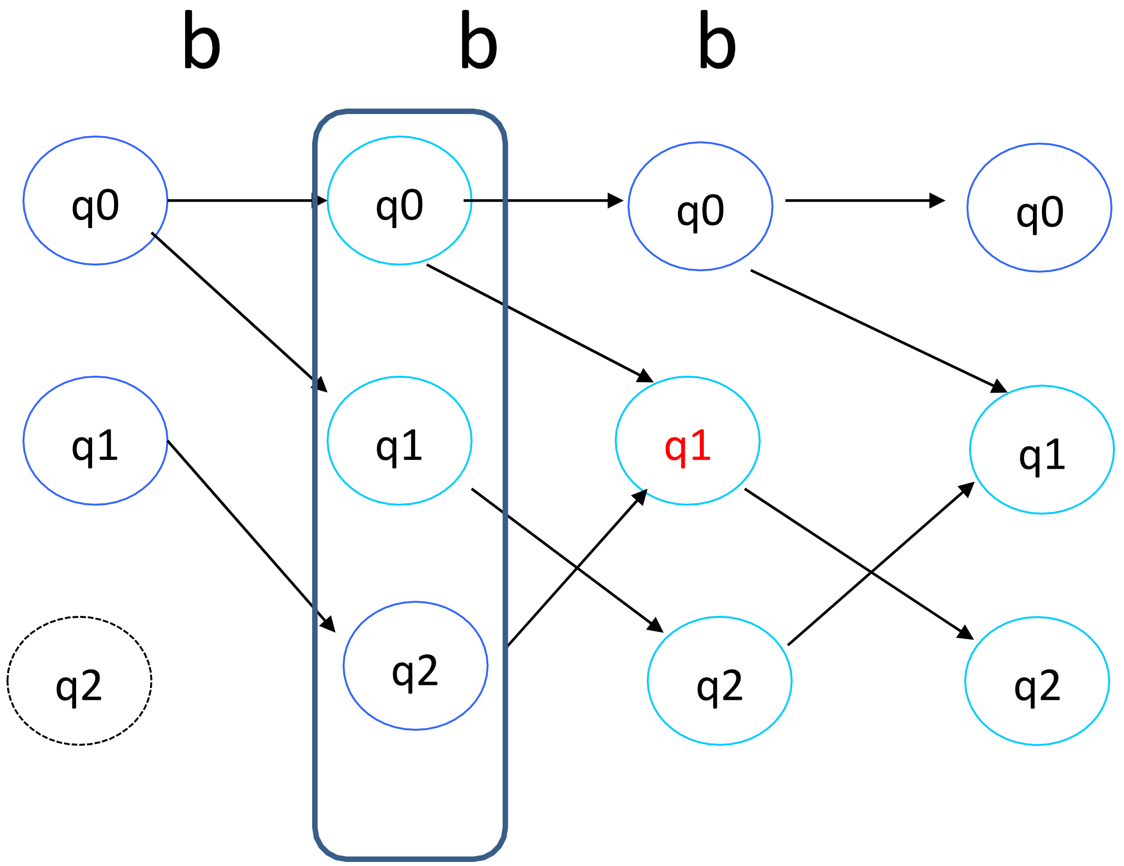
$$\gamma_x(0,q) = I(q),$$
$$\gamma_x(i,q) = \max_{q'\in Q}\gamma_x(i-1,q')\cdot P(q',x_i,q), \quad 1 \le i \le |x|;$$

Remember to include the final-prob:

$$\widetilde{\mathbf{Pr}}_A(x) = \max_{q\in Q}\gamma_x(|x|,q)\cdot F(q)$$

I(q0)=0.4
I(q1)=0.6





gamma(2, q1)

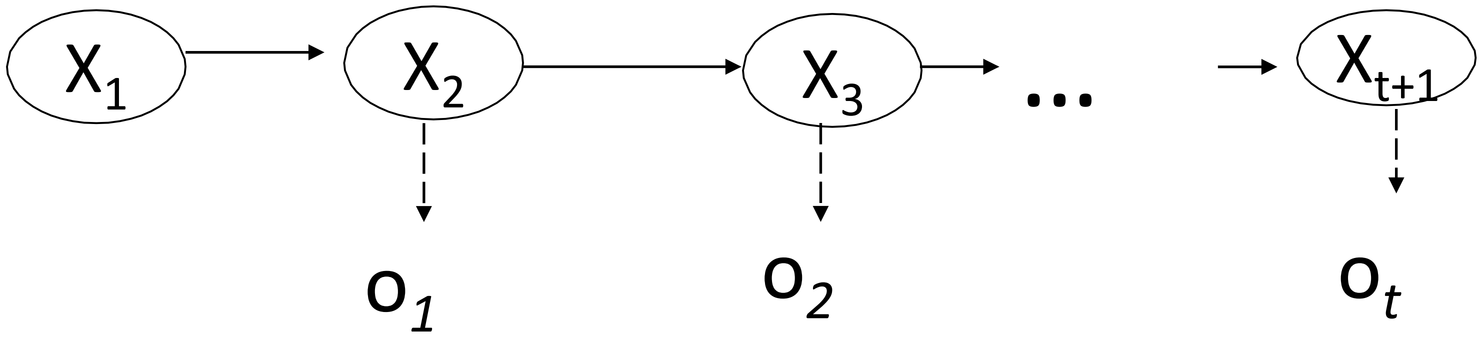
- Calculate the gamma function efficiently:
  - Use a two-dimensional array g[i, q], not a recursive function

- Need to remember the best path, not just the highest probability:
  - For each [i, q], remember q’
  - In other words, you need one array for gamma, another for backpointer: b[i, q] = q’

- For hw3 Q3, you need to find the best path and the corresponding output sequence given input x and an FST:
  - Since you know the (q’, x_i, q) arc on the best path, you can find the corresponding y_i for that arc.
  - For Hw3 Q3, assume that there is no epsilon-transition in the FST.

HMM as a parser: Finding the best state sequence

- Given the observation $O_{1,t}=o_1 \ldots o_t$, find the state sequence $X_{1,t+1}=X_1 \ldots X_{t+1}$ that maximizes $P(X_{1,t+1} \mid O_{1,t})$.



➔ Viterbi algorithm

“time flies like an arrow”

\init

BOS	1.0

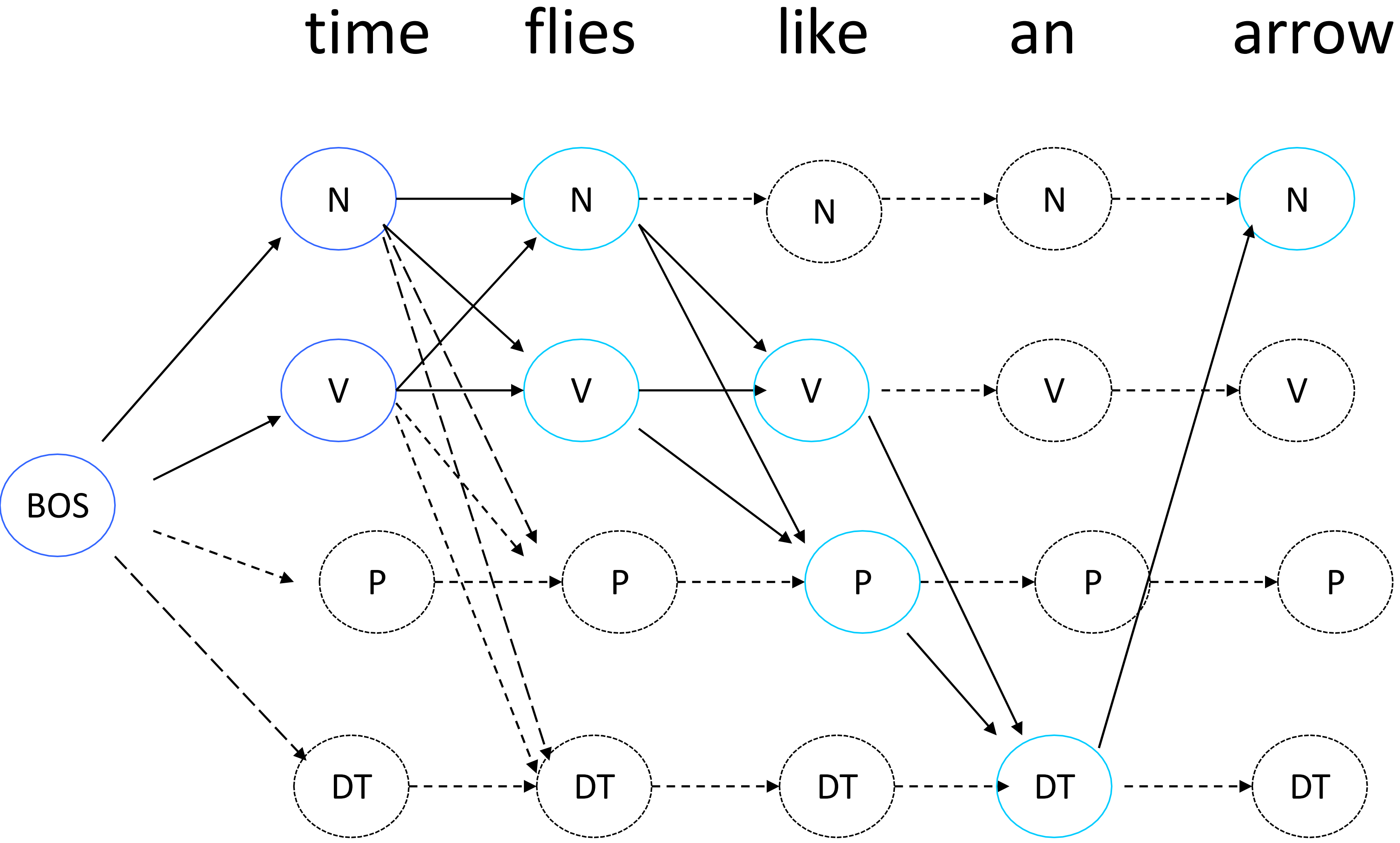
\transition

BOS	N	0.5
BOS	DT	0.4
BOS	V	0.1
DT	N	1.0
N	N	0.2
N	V	0.7
N	P	0.1
V	DT	0.4
V	N	0.4
V	P	0.1
V	V	0.1
P	DT	0.6
P	N	0.4

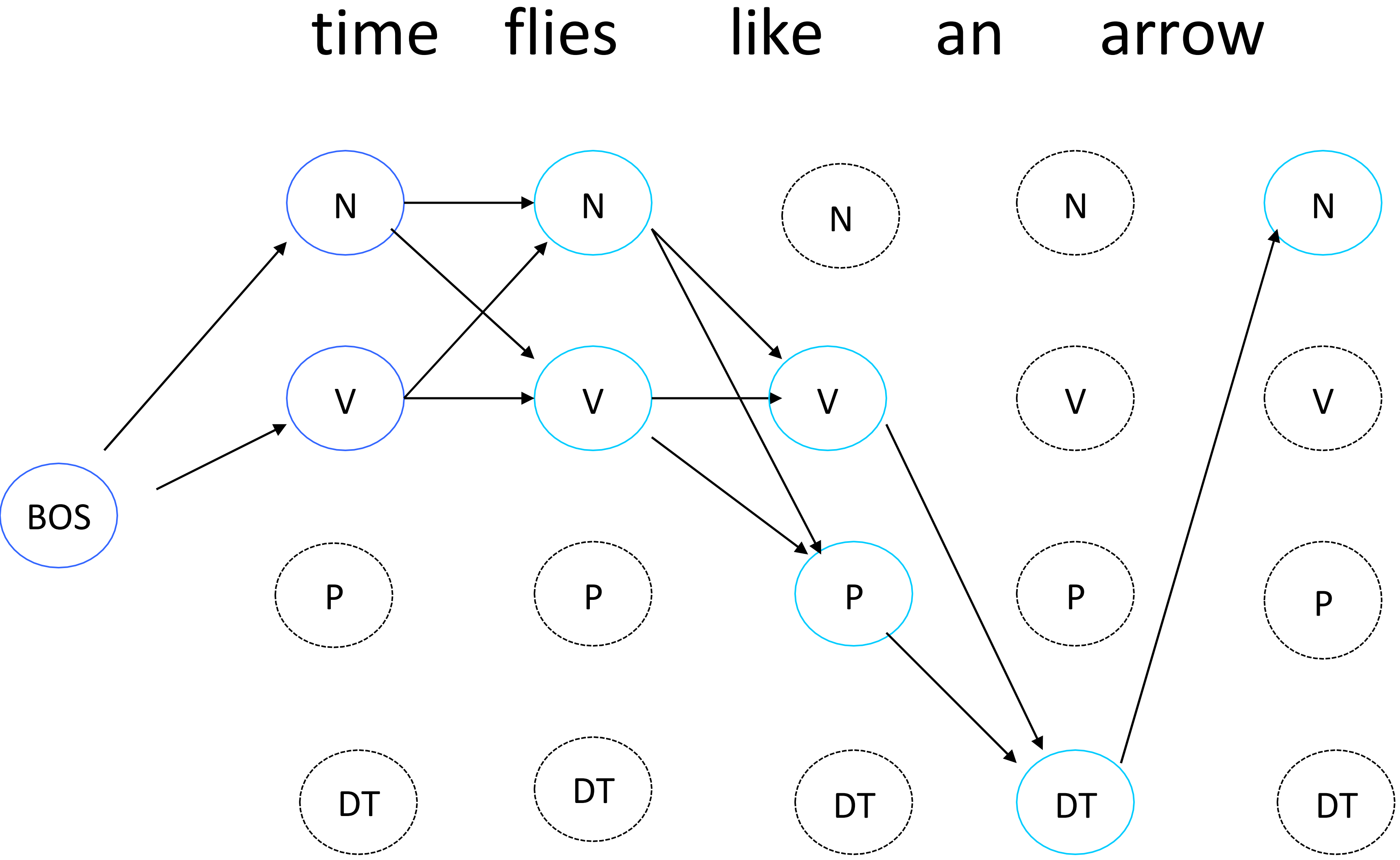
\emission

N	time	0.1
V	time	0.1
N	flies	0.1
V	flies	0.2
V	like	0.2
P	like	0.1
DT	an	0.3
N	arrow	0.1

Finding all the paths: to build the trellis



Finding all the paths (cont)



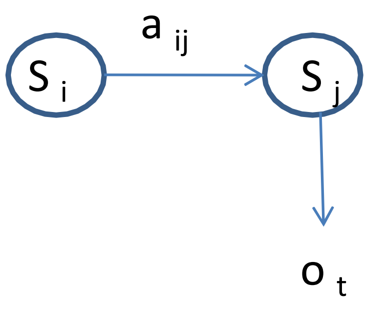
Viterbi algorithm

The probability of the best path that produces $O_{1,t-1}$ while ending up in state s_j:

$$\delta_j(t) \overset{def}{=} \max_{X_{1,t-1}} P(X_{1,t-1}, O_{1,t-1}, X_t = j)$$

Initialization: $\delta_j(1) = \pi_j$

Induction: $\delta_j(t+1) = \max_i \delta_i(t) a_{ij} b_{jo_t}$



➔ Modify it to allow epsilon-emission

$$\begin{tikzcd}
T=1 & T=2 & & T=t & T=t+1 \\
X_1 \arrow[r] & X_2 \arrow[r] \arrow[d, dashed] & \cdots \arrow[r] & X_t \arrow[r] \arrow[d, dashed] & X_{t+1} \arrow[d, dashed] \\
& o_1 & & o_{t-1} & o_t
\end{tikzcd}$$

$$\delta_j(t) \overset{def}{=} \max_{X_{1,t-1}} P(X_{1,t-1}, O_{1,t-1}, X_t = j)$$

$$\delta_j(1) = \pi_j$$

$$\begin{tikzcd}
S_i \arrow[r, "a_{ij}"] & S_j \arrow[d] \\
& o_t
\end{tikzcd}$$

$$\delta_j(t+1) = \max_i \delta_i(t) a_{ij} b_{jo_t}$$

Proof of the recursive function**

$$
\begin{aligned}
\delta_j(t+1) &= \max_{X_{1,t}} P(X_{1,t}, O_{1,t}, X_{t+1} = j) \\
&= \max_{X_{1,t}} P(X_{1,t-1}, O_{1,t-1}, O_t, X_t, X_{t+1} = j) \\
&= \max_{X_t = i} \max_{X_{1,t-1}} P(X_{1,t-1}, O_{1,t-1}, X_t = i) P(o_t, X_{t+1} = j \mid X_{1,t-1}, O_{1,t-1}, X_t = i) \\
&= \max_{X_t = i} \max_{X_{1,t-1}} P(X_{1,t-1}, O_{1,t-1}, X_t = i) a_{ij} b_{jo_t} \\
&= \max_{X_t = i} a_{ij} b_{jo_t} (\max_{X_{1,t-1}} P(X_{1,t-1}, O_{1,t-1}, X_t = i)) \\
&= \max_{X_t = i} \delta_i(t) a_{ij} b_{jo_t}
\end{aligned}
$$

Viterbi for HMM (when the observation is produced by the from-state)

Assuming that the observation is $o_1 \ldots o_n$

- Initialization:

$$v_1(j) = \pi_j \cdot b_j(o_1)$$
$$bt_1(j) = 0$$

- Recursion:

$$v_t(j) = \max_i v_{t-1}(i) \cdot a_{ij} \cdot b_j(o_t)$$
$$bt_t(j) = \arg\max_i v_{t-1}(i) \cdot a_{ij} \cdot b_j(o_t)$$

- Termination:

$$p^\star = \max_i v_n(i) \cdot a_{iF}$$
$$q_n^\star = \arg\max_i v_n(i) \cdot a_{iF}$$

Viterbi
(when the observation is produced by the to-state)

Assuming that the observation is $o_1 \dots o_n$

- Initialization:

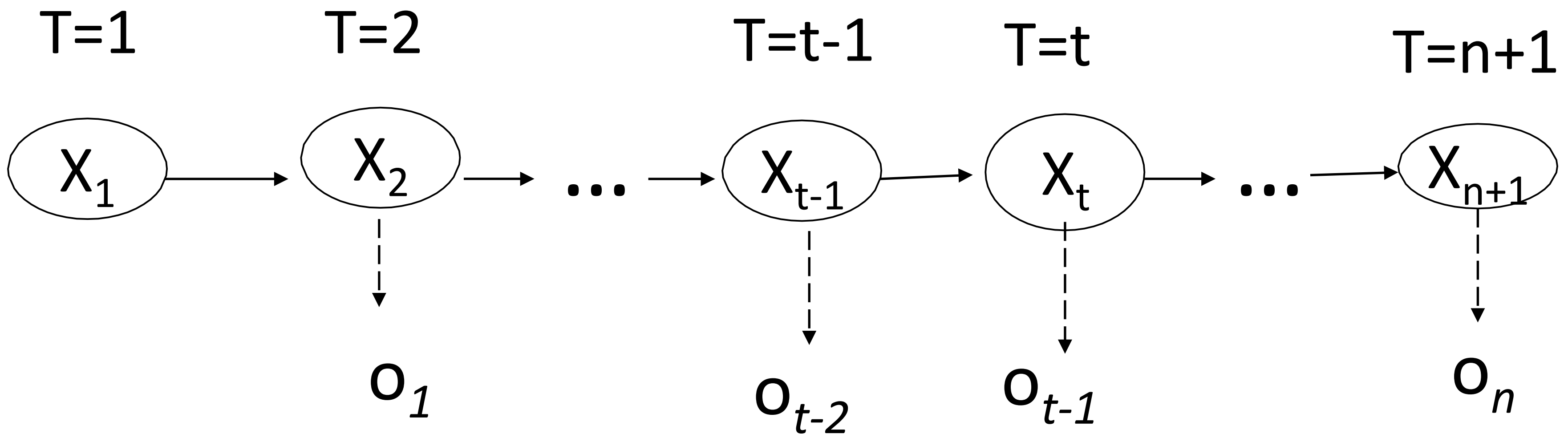
$$v_1(j) = \pi_j$$
$$bt_1(j) = 0$$

- Recursion:

$$v_t(j) = \max_i v_{t-1}(i) \cdot a_{ij} \cdot b_j(o_{t-1})$$
$$bt_t(j) = \arg\max_i v_{t-1}(i) \cdot a_{ij} \cdot b_j(o_{t-1})$$

- Termination:

$$p^\star = \max_i v_{n+1}(i)$$
$$q_n^\star = \arg\max_i v_{n+1}(i)$$



$$\delta_j(t) \overset{def}{=} \max_{X_{1,t-1}} P(X_{1,t-1}, O_{1,t-1}, X_t = j)$$

$$\delta_j(1) = \pi_j$$

$$\delta_j(t) = \max_i \delta_i(t-1) a_{ij} b_{jo_{t-1}}$$

Viterbi algorithm: calculating $\text{delta}_j(t)$

```
## N is the number of states in the HMM structure
## observ is the observation O, and leng is the length of observ.

Initialize delta[0..leng][0..N-1] to 0

for each state j                          ## this is the initialization step
    delta[0][j] = pi[j]
    back-pointer[0][j] = -1   # dummy

for (t=0; t<leng; t++)                    ## this is the recursion step
   for (j=0; j<N; j++)
      k=observ[t]     # the symbol at time t
      delta[t+1][j] = $\max_i$ delta[t][i] $a_{ij}$ $b_{jk}$
      back-pointer[t+1][j] = arg $\max_i$ delta[t][i] $a_{ij}$ $b_{jk}$
```

Viterbi algorithm: retrieving the best path (correspond to the termination step)

```
# find the best path
best_final_state = arg max$_j$ delta[leng] [j]

# start with the last state in the sequence
j = best_final_state

push(arr, j);

for (t=leng; t>0; t--)
   i = back-pointer[t] [j]
   push(arr, i)
   j = i

return reverse(arr)
```

Implementation issue storing HMM

Approach #1:

- π_i: pi {state_str}
- a_{ij}: a {from_state_str} {to_state_str}
- b_{jk}: b {state_str} {symbol}

Approach #2:

- state2idx{state_str} = state_idx
- symbol2idx{symbol_str} = symbol_idx

- π_i: pi [state_idx] = prob
- a_{ij}: a [from_state_idx] [to_state_idx] = prob
- b_{jk}: b [state_idx] [symbol_idx] = prob

- idx2state[state_idx] = state_str
- Idx2symbol[symbol_idx] = symbol_str

Storing HMM: sparse matrix

- a_{ij}: a [i] [j] = prob
- b_{jk}: b [j] [k] = prob

- a_{ij}: a[i] = “j1 p1 j2 p2 …”
- a_{ij}: a[j] = “i1 p1 i2 p2 …”

- b_{jk}: b[j] = “k1 p1 k2 p2 ….”
- b_{jk}: b[k] = “j1 p1 j2 p2 …”

Other implementation issues

- Index starts from 0 in programming, but often starts from 1 in algorithms
- The sum of lgprob is used in practice to replace the product of prob.
- Check constraints and print out warning if the constraints are not met.

HMM as LM

HMM as an LM: computing P(o_1, …, o_T)

$$P(o_1, ..., o_T) = \sum_{X_1, ..., X_{T+1}} P(o_1, ..., o_T, X_1, ..., X_{T+1})$$

1st try:

- enumerate all possible paths
- add the probabilities of all paths

Forward probabilities

- Forward probability: the probability of producing $O_{1,t-1}$ while ending up in state s_i:

$$\alpha_i(t) \overset{def}{=} P(O_{1,t-1}, X_t = i)$$

$$P(O) = \sum_{i=1}^{N} \alpha_i(T+1)$$

Calculating forward probability

Initialization: $\alpha_j(1) = \pi_j$

Induction: $\alpha_j(t+1) = P(O_{1,t}, X_{t+1} = j)$

$$= \sum_i \alpha_i(t) a_{ij} b_{jo_t}$$

$$\alpha_j(t+1) = P(O_{1,t}, X_{t+1} = j)$$

$$= \sum_i P(O_{1,t}, X_t = i, X_{t+1} = j)$$

$$= \sum_i P(O_{1,t-1}, X_t = i) * P(o_t, X_{t+1} = j \mid O_{1,t-1}, X_t = i)$$

$$= \sum_i P(O_{1,t-1}, X_t = i) * P(o_t, X_{t+1} = j \mid X_t = i)$$

$$= \sum_i \alpha_i(t) a_{ij} b_{jo_t}$$

Summary

- Definition: hidden states, output symbols
- Properties: Markov assumption
- Applications: POS tagging, etc.
- Three basic questions in HMM
  - Find the probability of an observation: forward probability
  - Find the best sequence: Viterbi algorithm
  - Estimate probability: MLE
- N-gram POS tagger: decoding with Viterbi algorithm