# HMM (2)

LING 570

Fei Xia

Week 7: 11/09/09

# 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

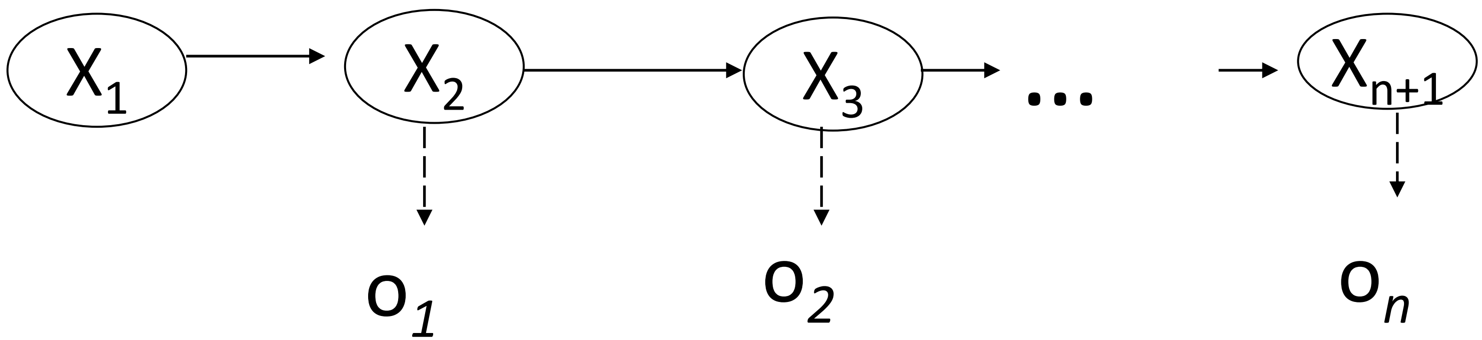
- Supervised learning:
  - The state sequences in the training data are known
  - ML estimation

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

# HMM as a parser

# 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 $P(X_{1,T+1} \mid O_{1,T})$.



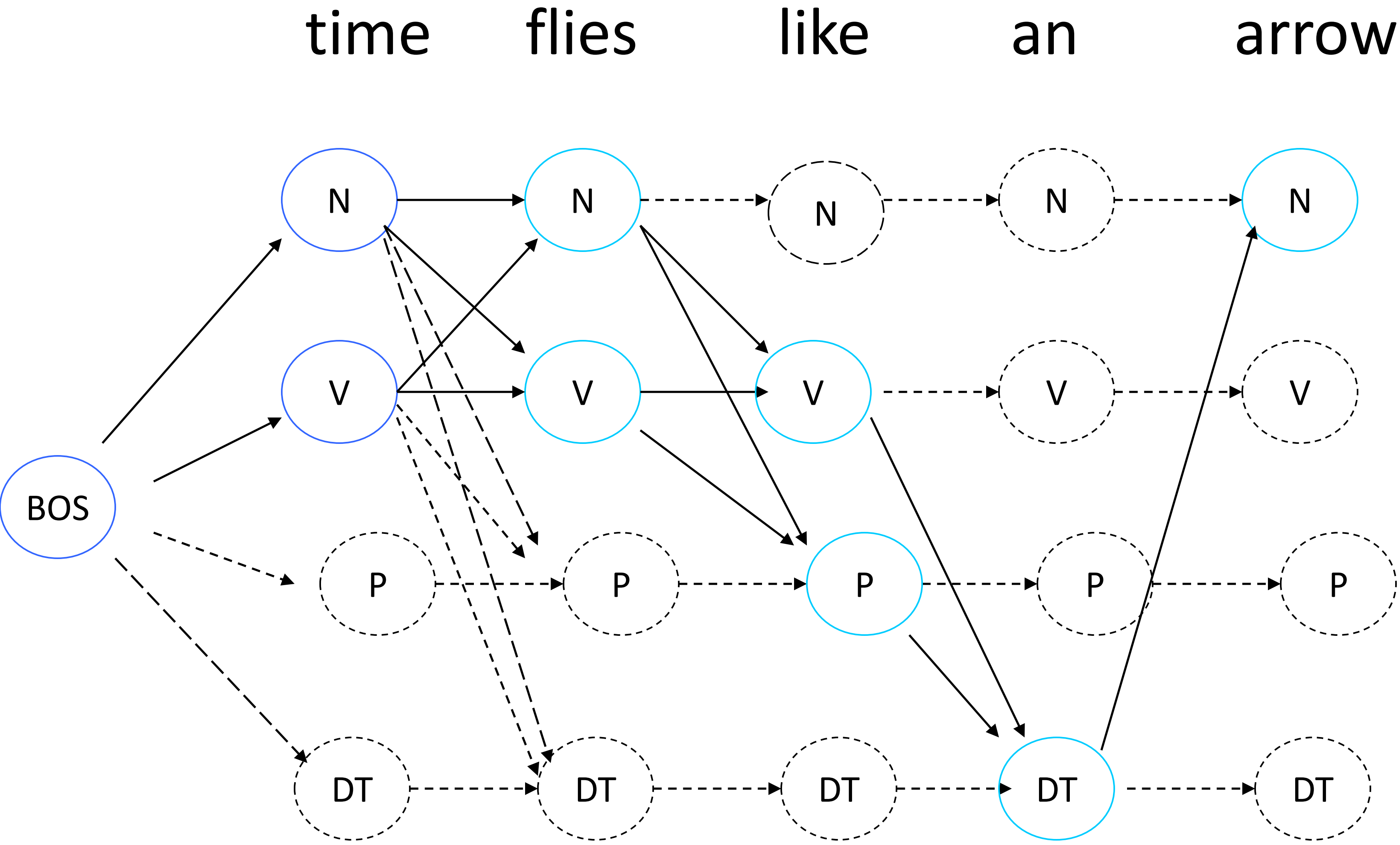

➔ Viterbi algorithm

# “time flies like an arrow”

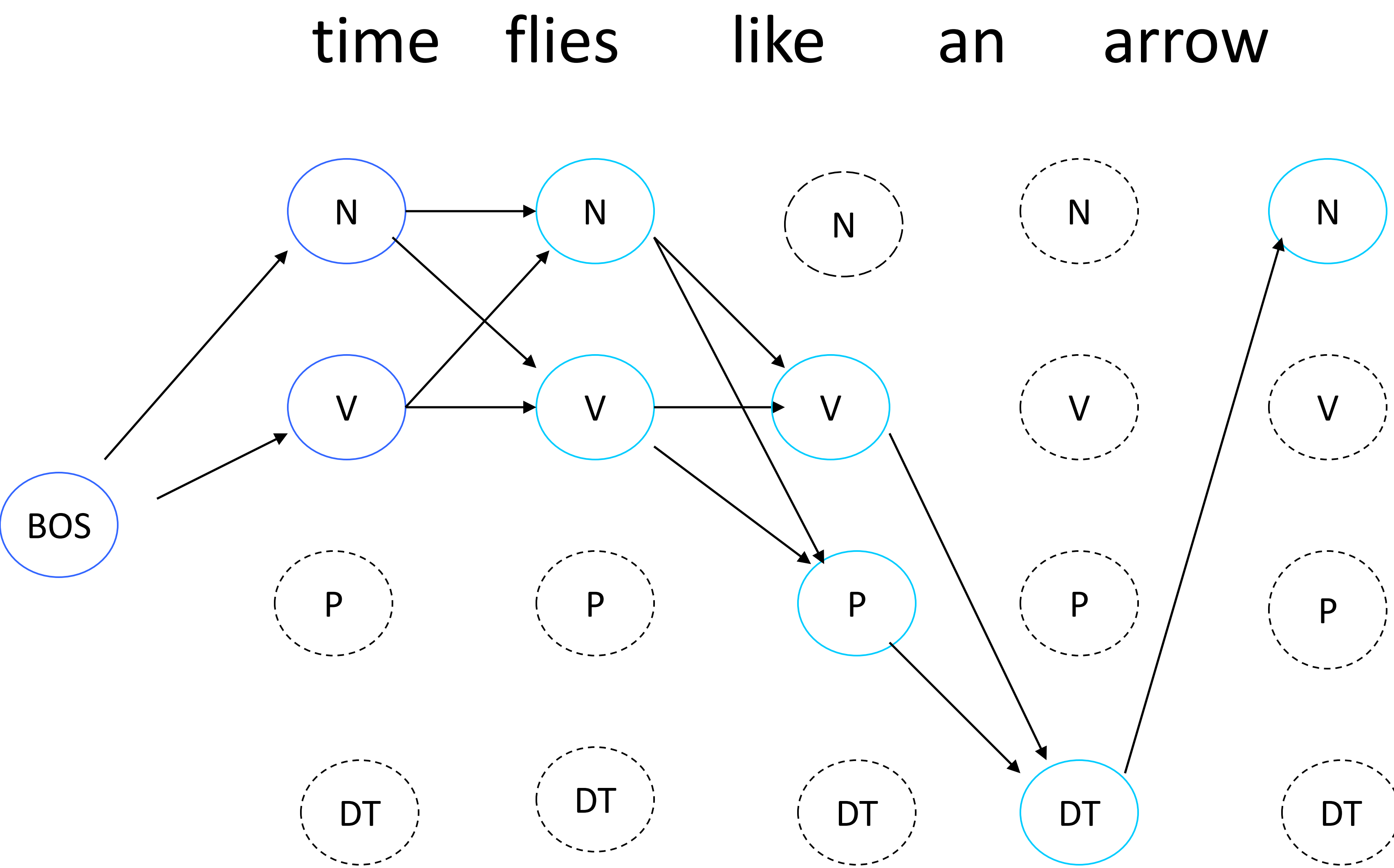
```
\init
  BOS 1.0

\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

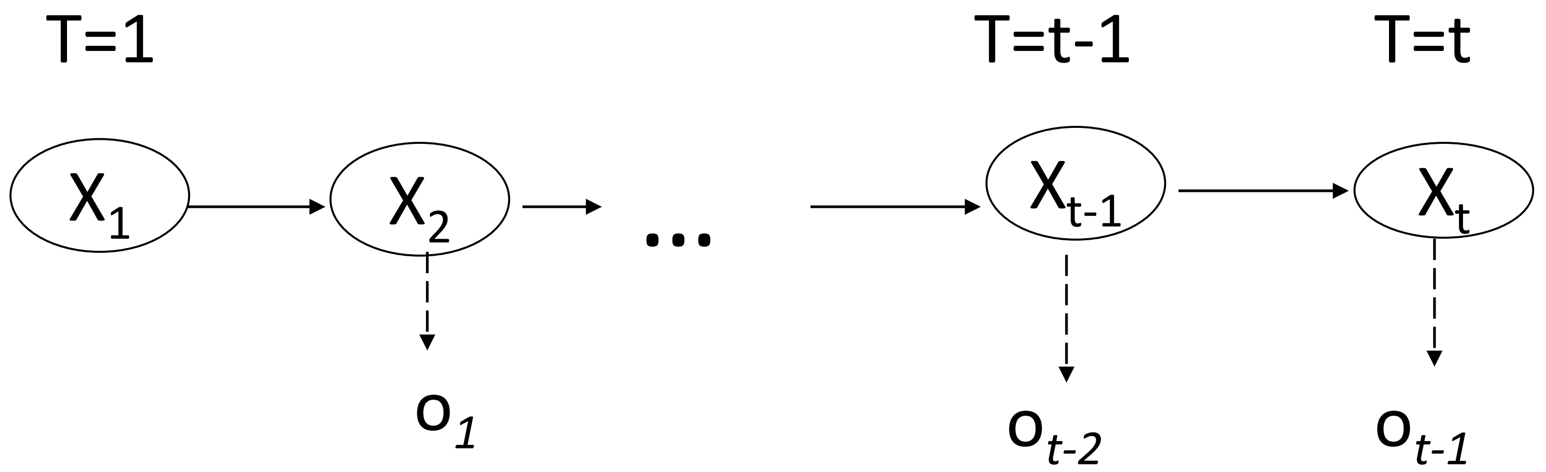
The probability of the best path that produces $O_{1,t-1}$ while ending up in state $s_j$:

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

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




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

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

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

# Proof of the recursive function**

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

$$= \max_{X_{1,t}} P(X_{1,t-1}, O_{1,t-1}, O_t, X_t, X_{t+1} = j)$$

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

$$= \max_{X_t = i} \max_{X_{1,t-1}} P(X_{1,t-1}, O_{1,t-1}, X_t = i) a_{ij} b_{jo_t}$$

$$= \max_{X_t = i} a_{ij} b_{jo_t} (\max_{X_{1,t-1}} P(X_{1,t-1}, O_{1,t-1}, X_t = i))$$

$$= \max_{X_t = i} \delta_i(t) a_{ij} b_{jo_t}$$

# Viterbi algorithm: calculating $\delta_j$(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
    delta[0][j] = $\pi$[j]
    back-pointer[0][j] = -1   # dummy

for (t=0; t<leng; t++)
   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

```
# 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:

- $\pi_i$: 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

- $\pi_i$: 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 logprob 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 $P(o_1, \ldots, o_T)$

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

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
- Bigram POS tagger: decoding with Viterbi algorithm

# Hw7

- Q1: Viterbi algorithm
  - viterbi.sh hmm test_file output_file
  - HMM: the same format as in Hw6
  - test_file: "$o_1\ o_2 \dots o_k$"
  - output_file: "$o_1\ o_2 \dots o_k$ => $x_1\ x_2 \dots x_{k+1}$ logprob"
  - logprob is $\lg P(o_1\ o_2 \dots o_k, x_1\ x_2 \dots x_{k+1})$

  - Do not try to smooth the input HMM. It might have been smoothed already.
  - You can reuse some code from check_hmm.sh in Hw6

# Q2: trigram POS tagger with HMM

- training:
  - cat wsj sec0.word pos | create_3gram_hmm.sh hmm1 1.0 0 0 unk_prob_sec22
- decoding: “w1 w2 … => x1 x2 … logprob”
  - viterbi.sh hmm1 test.word sys1
- convert format: “w1/t1 w2/t2 …”
  - cat sys1 | conv format.sh > sys1 res
- evaluation
  - calc tagging accuracy.pl test.word pos sys1 res > sys1 res.acc 2>&1