Markov Models
and
Hidden Markov Models (HMMs)

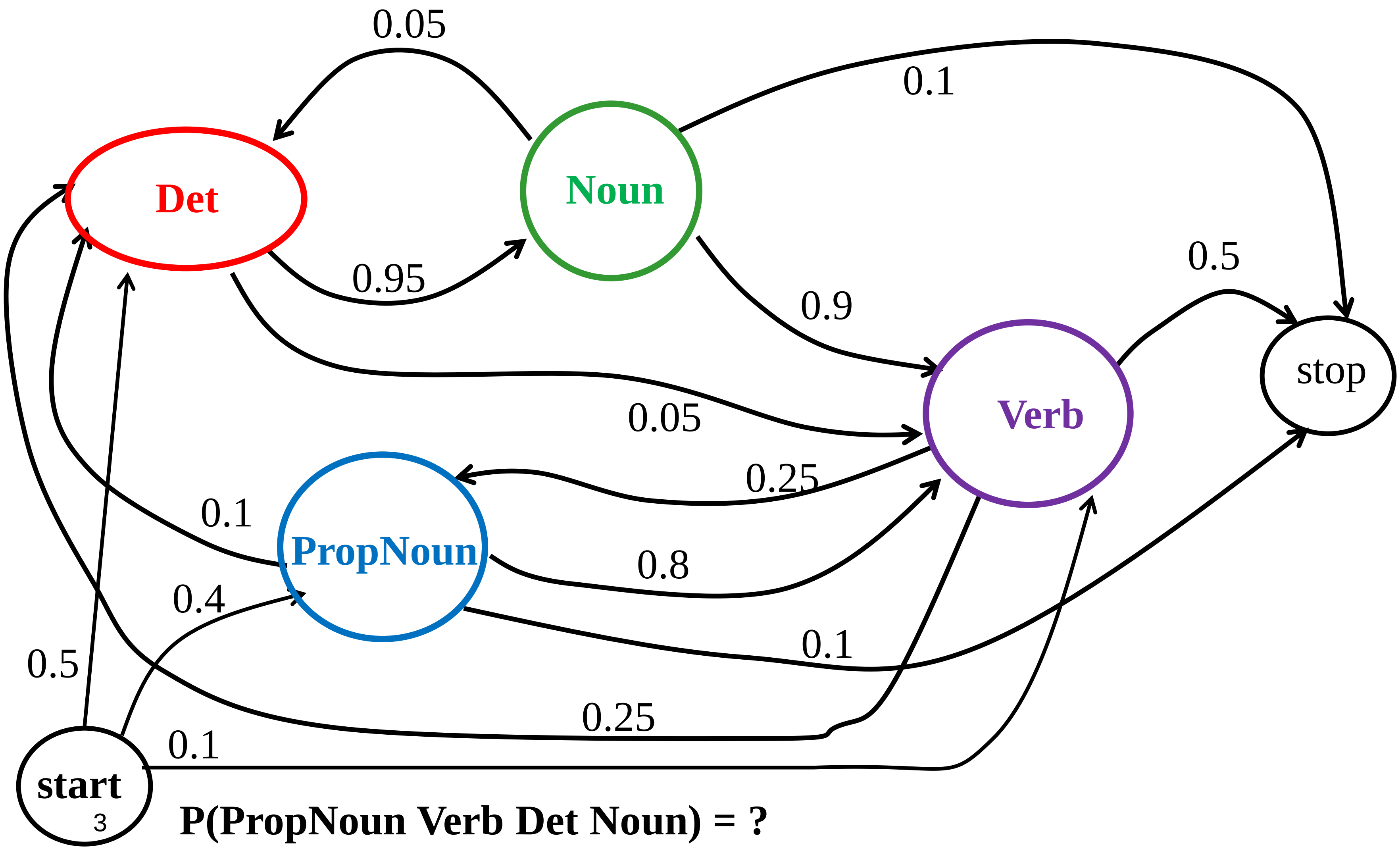
(Following slides are modified from Prof. Claire Cardie's slides and Prof. Raymond Mooney's slides. Some of the graphs are taken from the textbook.)

Markov Model (= Markov Chain)

- A sequence of random variables visiting a set of states
- Transition probability specifies the probability of transiting from one state to the other.
- Language Model!
- Markov Assumption: next state depends only on the current state and independent of previous history.

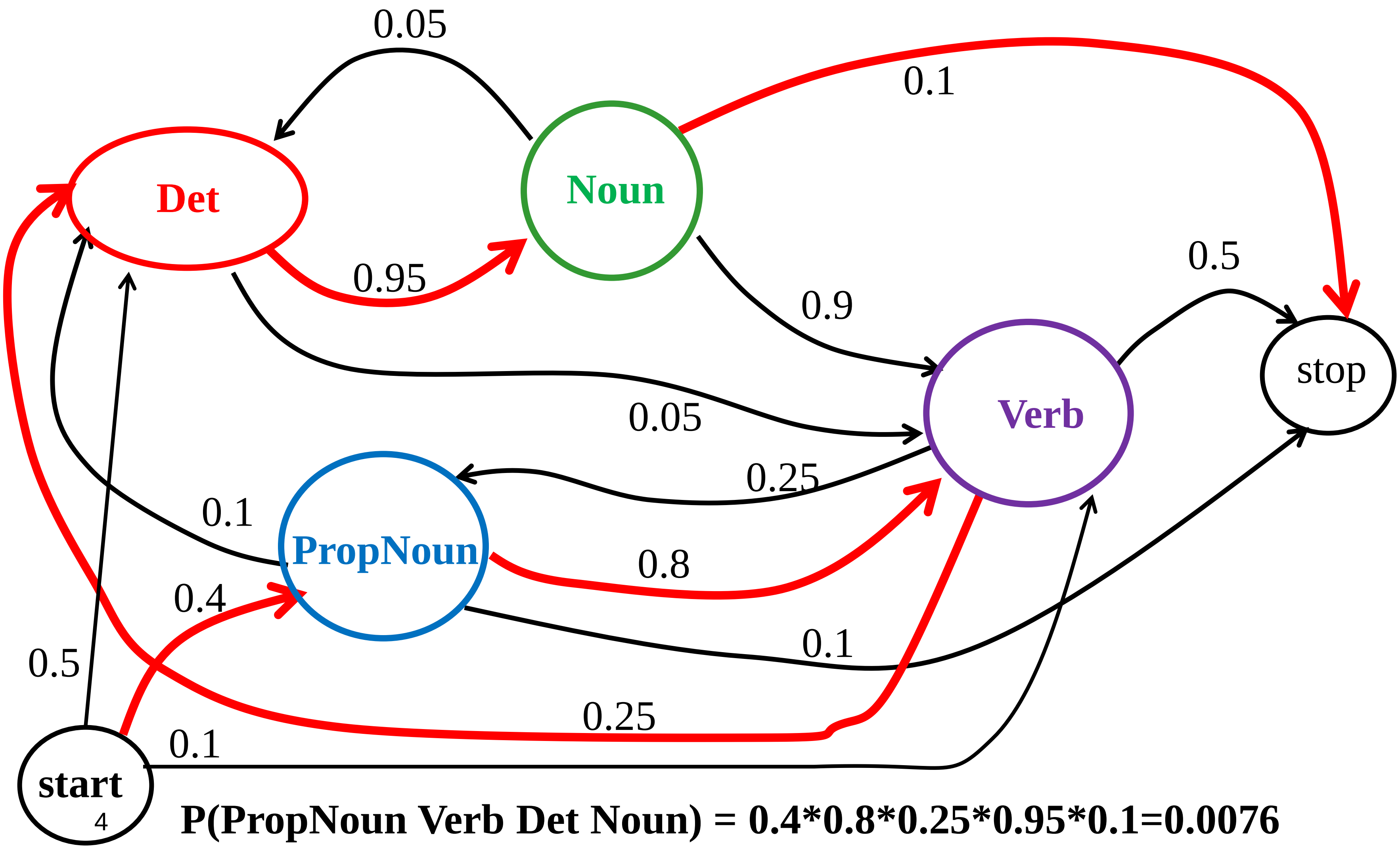
$$\Pr(X_{n+1} = x | X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = \Pr(X_{n+1} = x | X_n = x_n).$$

Sample Markov Model for POS



P(PropNoun Verb Det Noun) = ?

Sample Markov Model for POS

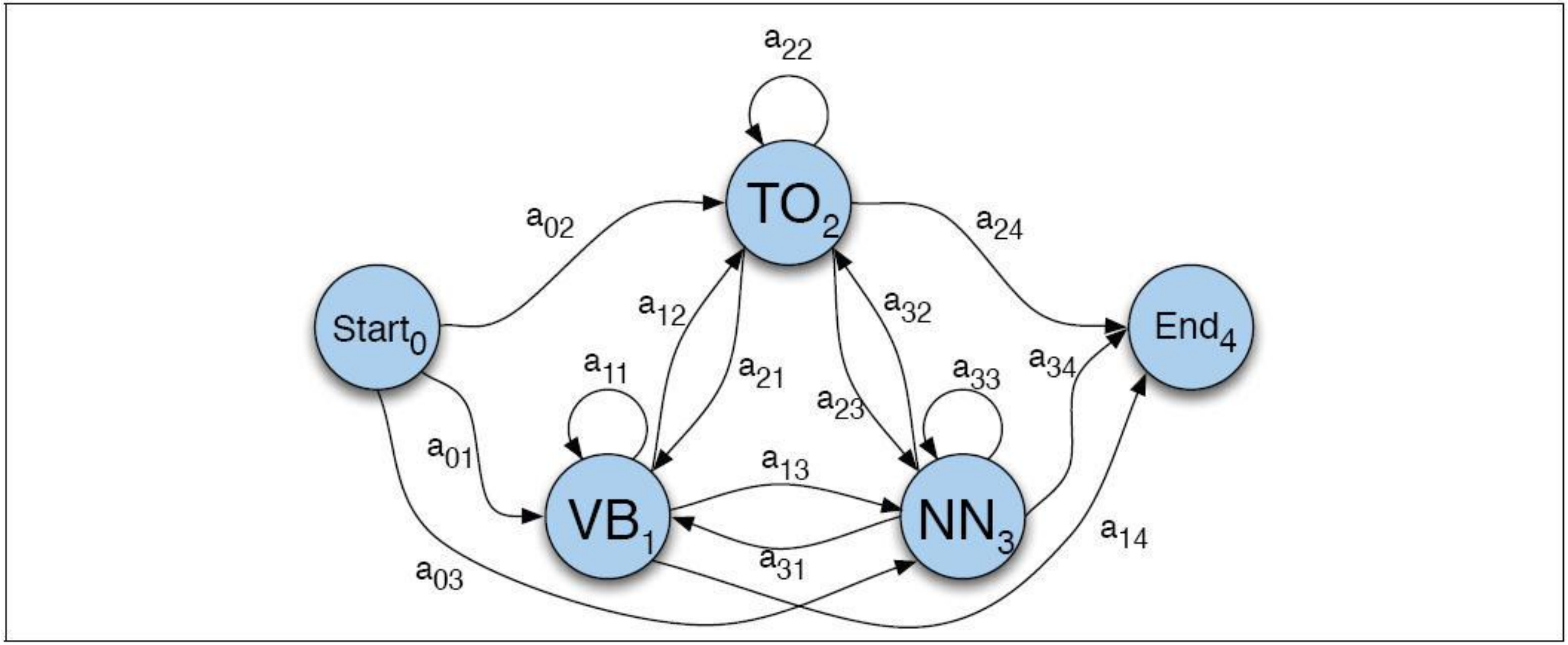


P(PropNoun Verb Det Noun) = 0.4*0.8*0.25*0.95*0.1=0.0076

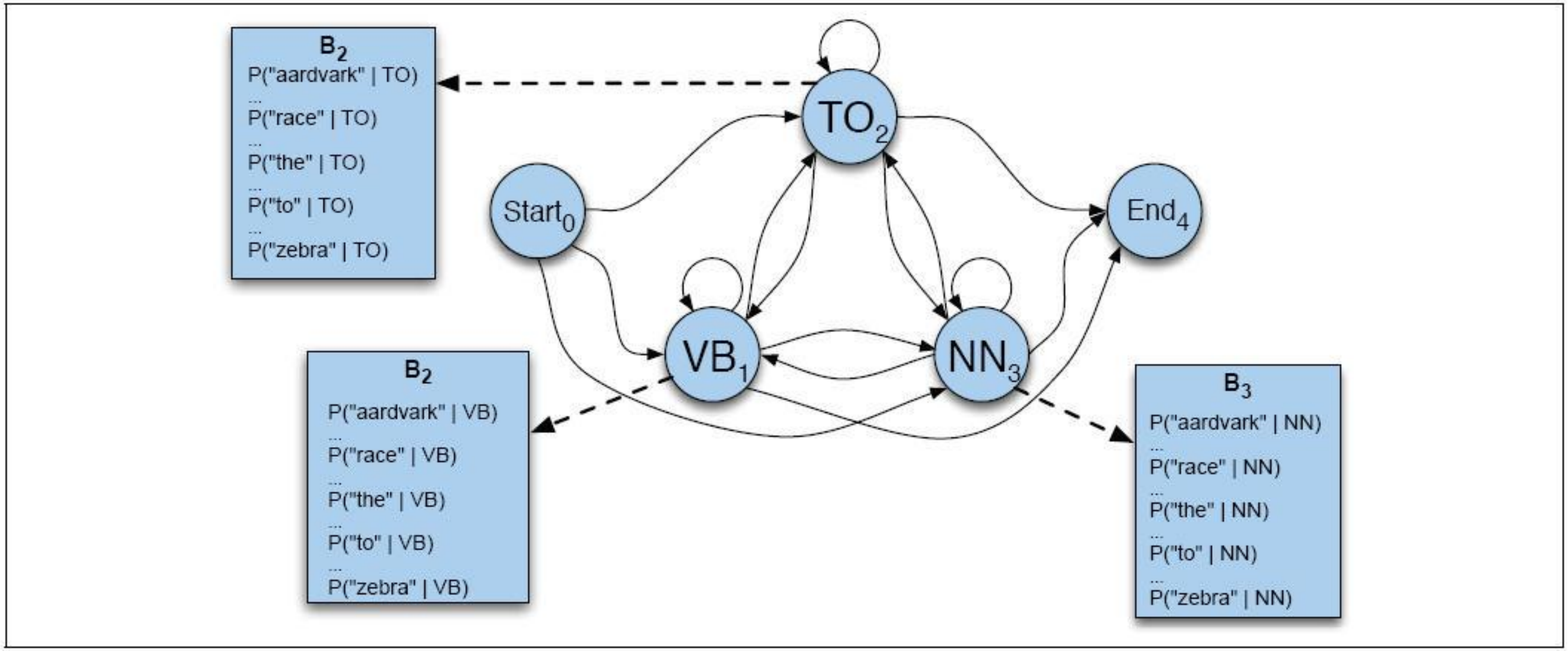
Hidden Markov Model (HMM)

- Probabilistic ***generative*** model for sequences.
- HMM Definition with respect to POS tagging:
  - States = POS tags
  - Observation = a sequence of words
  - Transition probability = bigram model for POS tags
  - Observation probability = probability of generating each token (word) from a given POS tag
- "Hidden" means the exact sequence of states (a sequence of POS tags) that generated the observation (a sequence of words) are hidden.
- .

Hidden Markov Model (HMM) represented as finite state machine

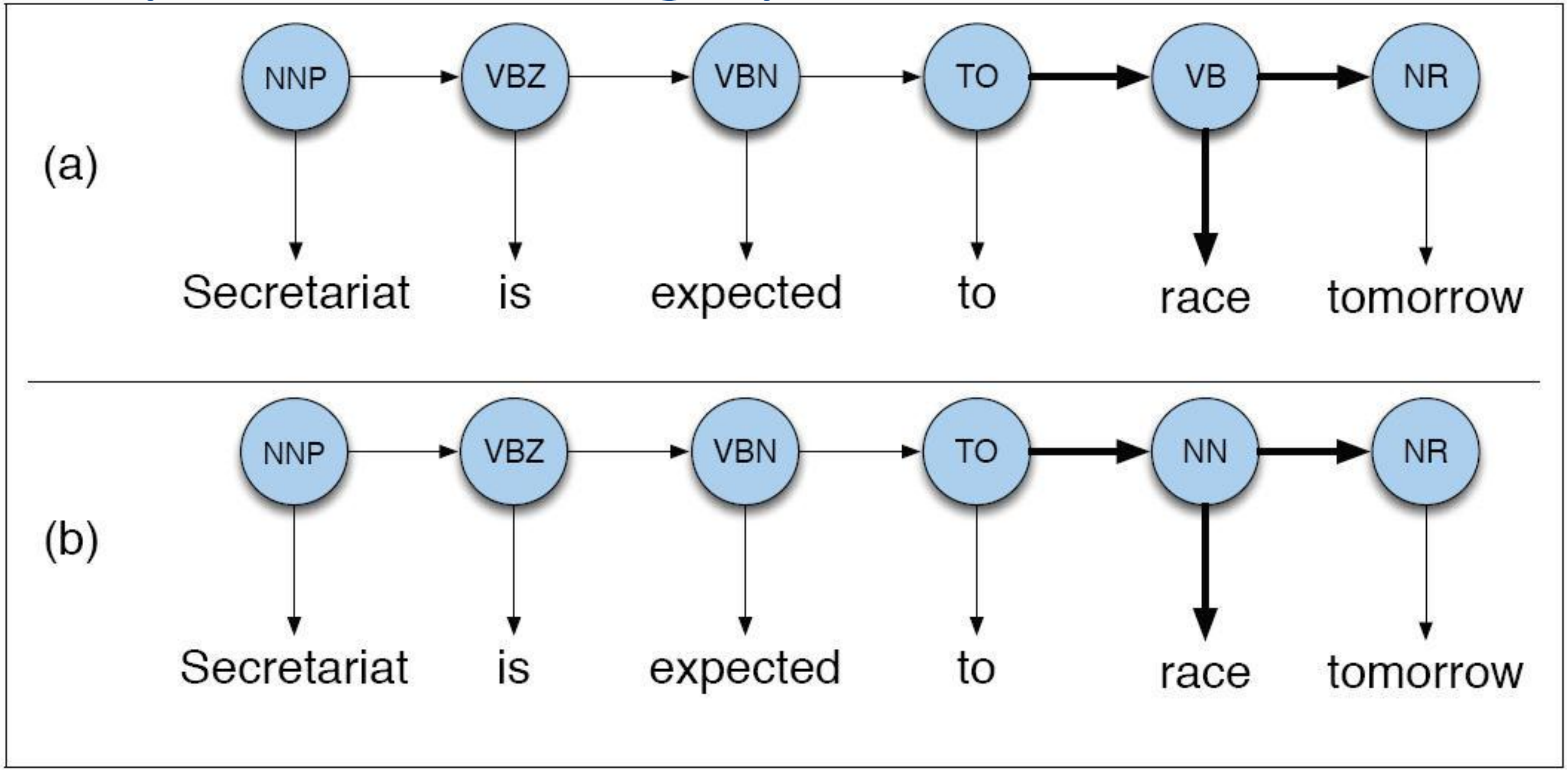


Hidden Markov Model (HMM) represented as finite state machine



- Note that in this representation, the number of nodes (states) = the size of the set of POS tags

Hidden Markov Model (HMM) represented as a graphical model



- Note that in this representation, the number of nodes (states) = the length of the word sequence.

Formal Definition of an HMM

$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} a_{12} \ldots a_{n1} \ldots a_{nn}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \ldots o_T$	a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, \ldots, v_V$
$B = b_i(o_t)$	a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state i
q_0, q_F	a special **start state** and **end (final) state** that are not associated with observations, together with transition probabilities $a_{01} a_{02} \ldots a_{0n}$ out of the start state and $a_{1F} a_{2F} \ldots a_{nF}$ into the end state

- **What are the parameters of HMM?**

Three important problems in HMM

Problem 1 (Likelihood):	Given an HMM $\lambda = (A, B)$ and an observation sequence O, determine the likelihood $P(O\|\lambda)$.
Problem 2 (Decoding):	Given an observation sequence O and an HMM $\lambda = (A, B)$, discover the best hidden state sequence Q.
Problem 3 (Learning):	Given an observation sequence O and the set of states in the HMM, learn the HMM parameters A and B.

- "**Likelihood**" function $L(\theta\,; X)$
  - Strictly speaking, likelihood is ***not*** a probability.
  - Likelihood is proportionate to $P(X \mid \theta)$

Three important problems in HMM

Problem 1 (Likelihood):	Given an HMM $\lambda = (A, B)$ and an observation sequence O, determine the likelihood $P(O\|\lambda)$.
Problem 2 (Decoding):	Given an observation sequence O and an HMM $\lambda = (A, B)$, discover the best hidden state sequence Q.
Problem 3 (Learning):	Given an observation sequence O and the set of states in the HMM, learn the HMM parameters A and B.

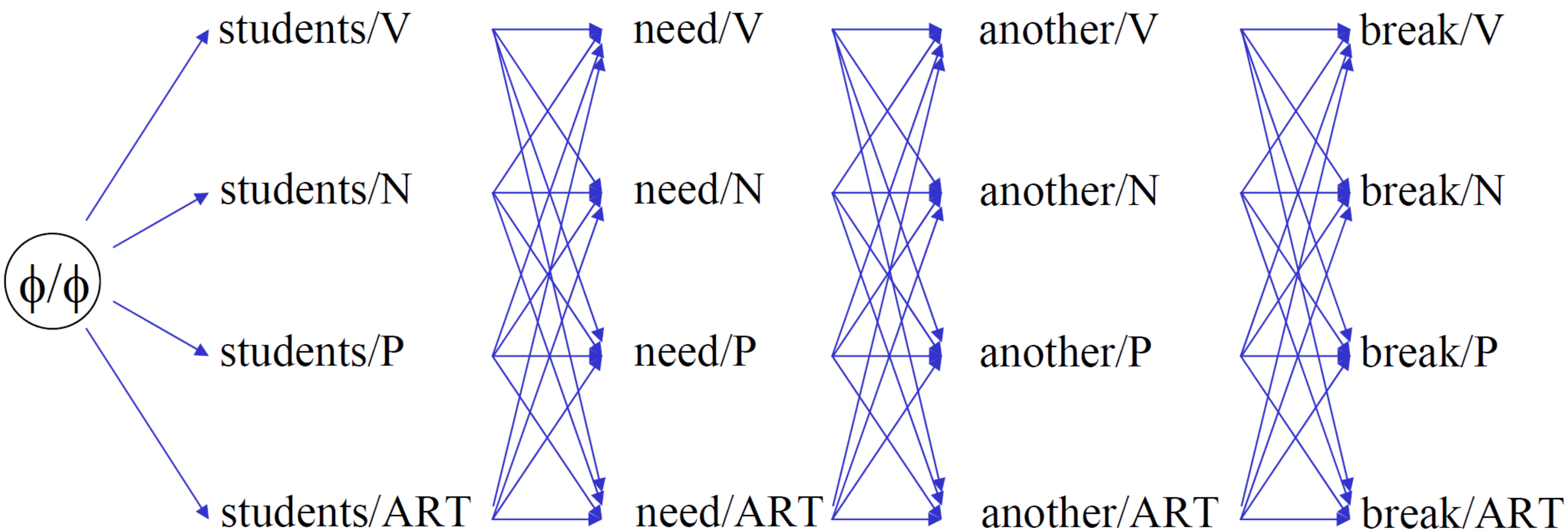
- Problem 1 (Likelihood) → **Forward** Algorithm
- Problem 2 (Decoding) → **Viterbi** Algorithm
- Problem 3 (Learning) → **Forward-backward** Algorithm

HMM Decoding: Viterbi Algorithm

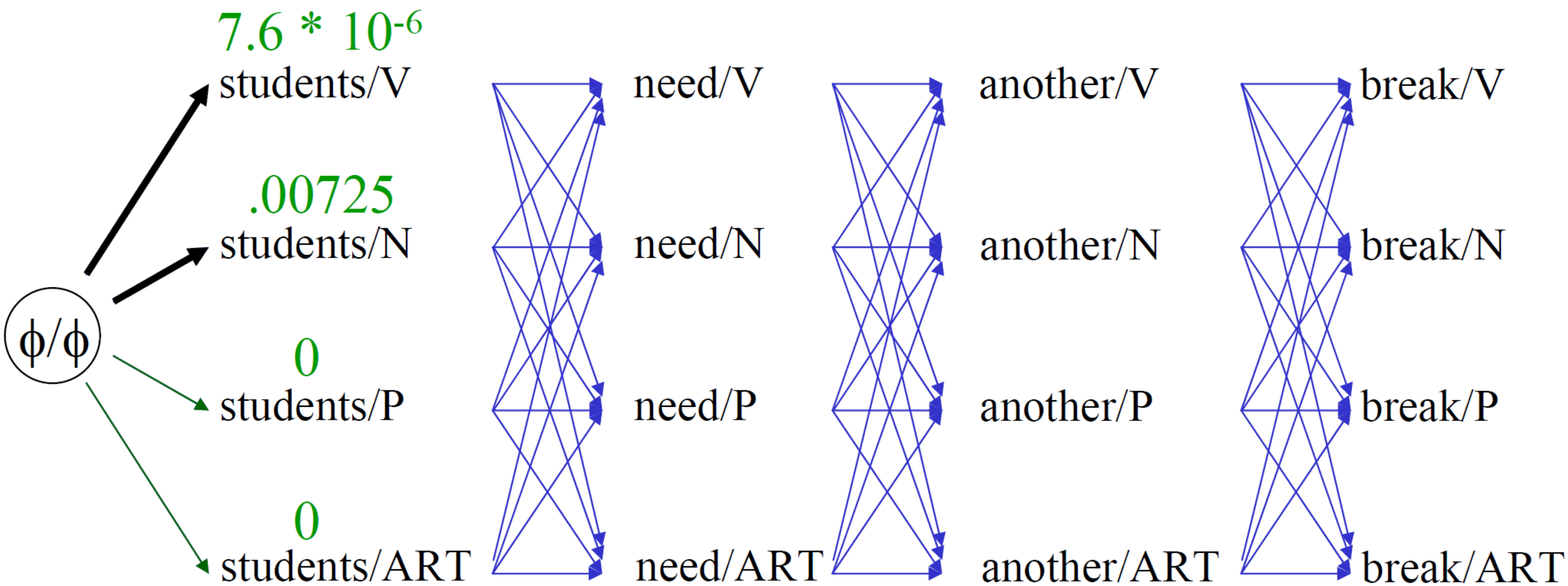
- Decoding finds the most likely sequence of states that produced the observed sequence.
  - A sequence of states = pos-tags
  - A sequence of observation = words
- Naïve solution: brute force search by enumerating all possible sequences of states.

  ➔ problem?
- Dynamic Programming!
- Standard procedure is called the Viterbi algorithm (Viterbi, 1967) and has $O(N^2T)$ time complexity.

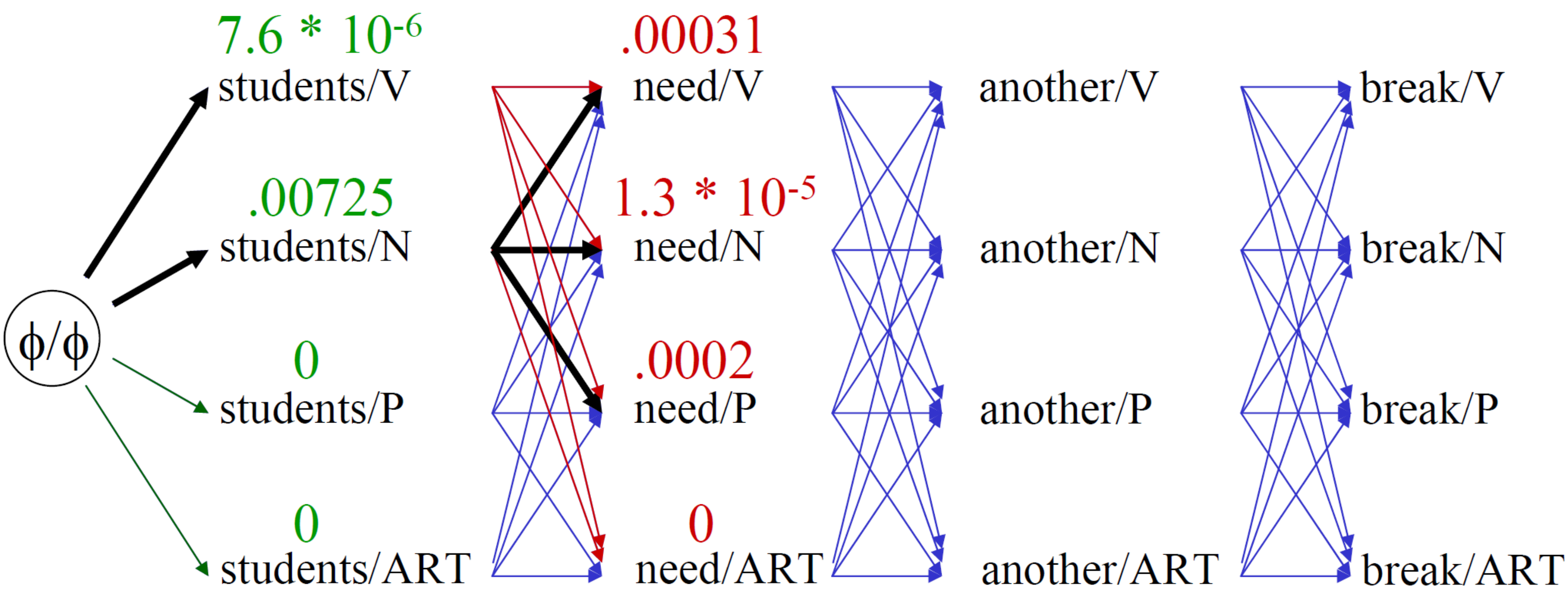
HMM Decoding: Viterbi Algorithm Intuition:



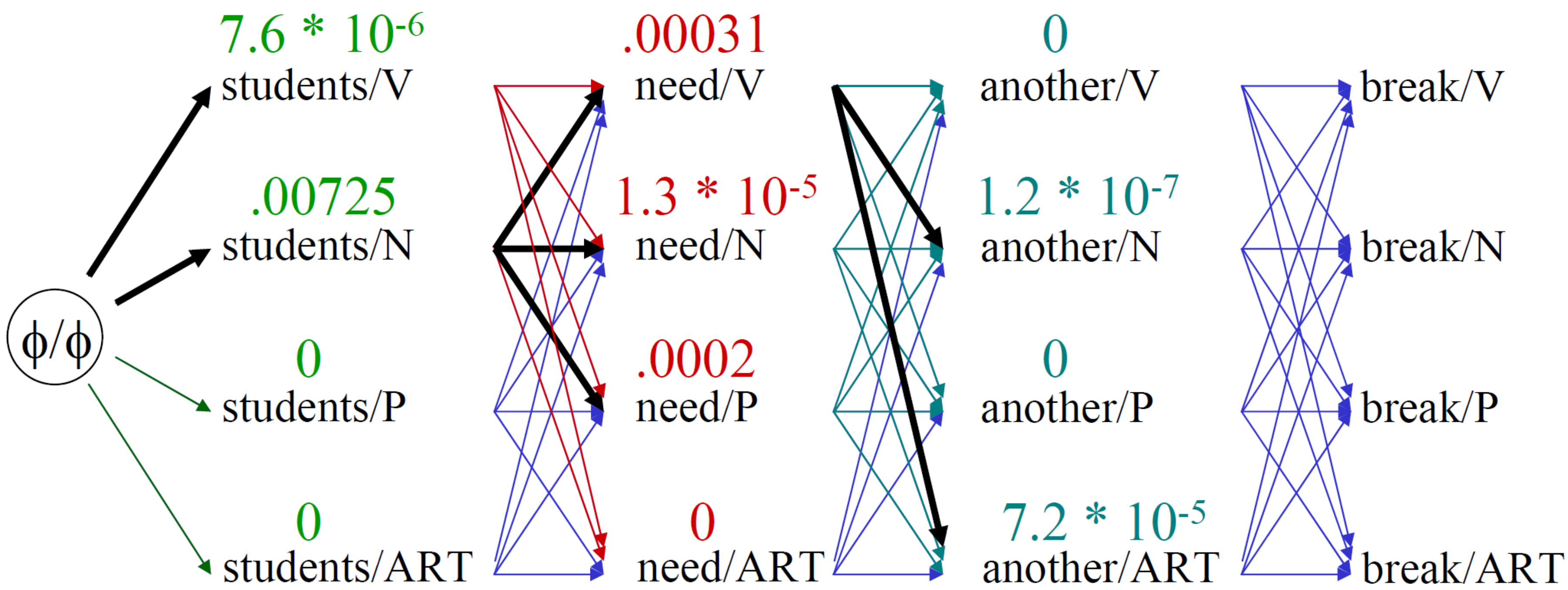
HMM Decoding: Viterbi Algorithm Intuition:



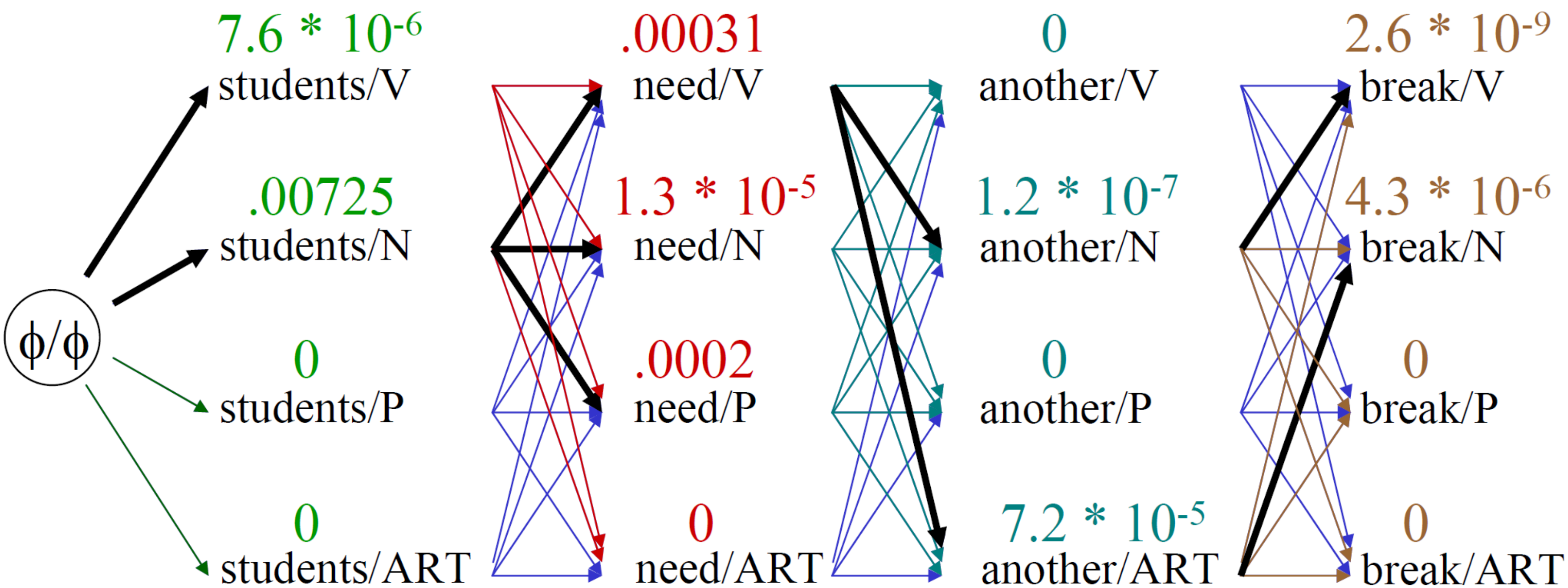
HMM Decoding: Viterbi Algorithm Intuition:



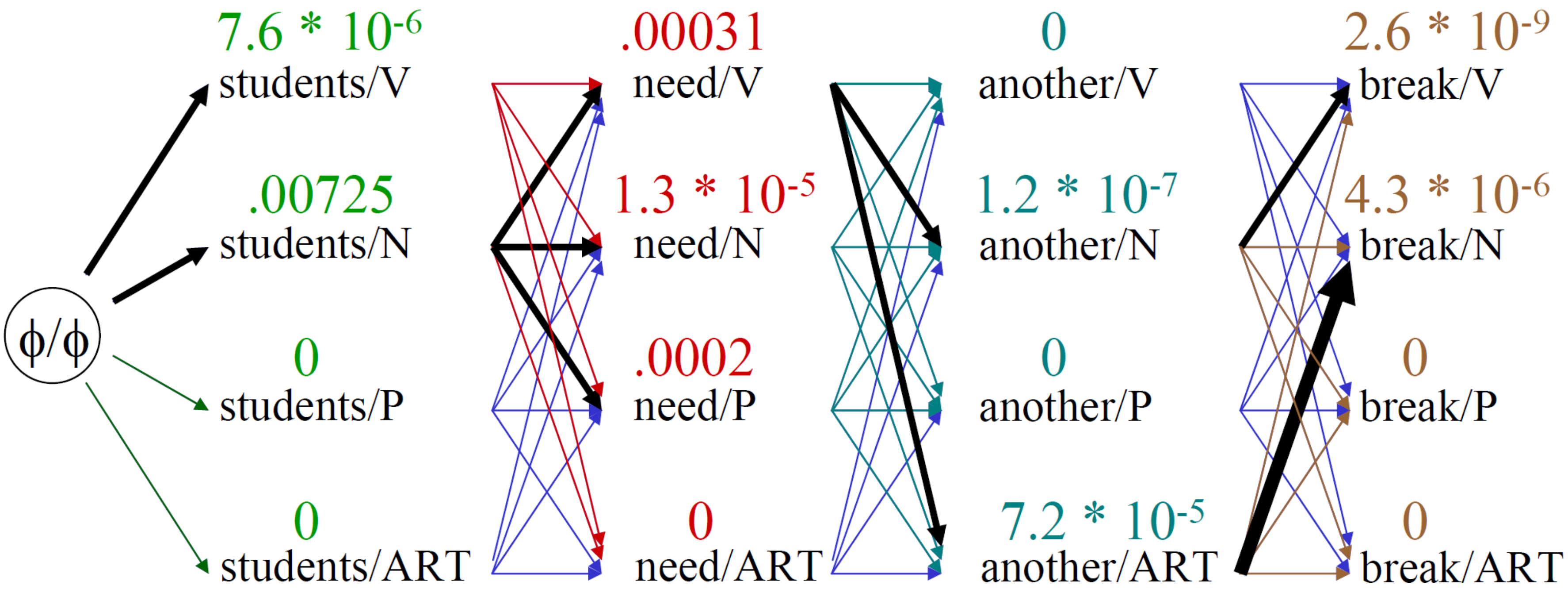
HMM Decoding: Viterbi Algorithm Intuition:



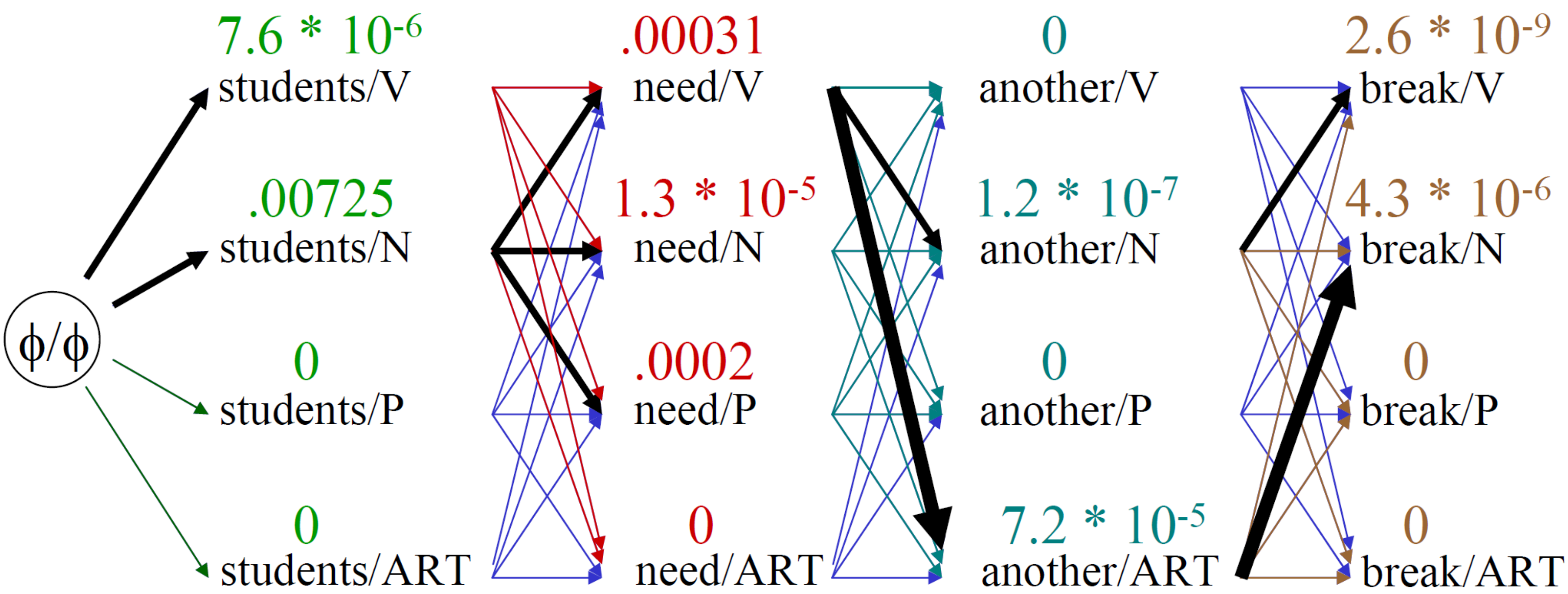
HMM Decoding: Viterbi Algorithm Intuition:



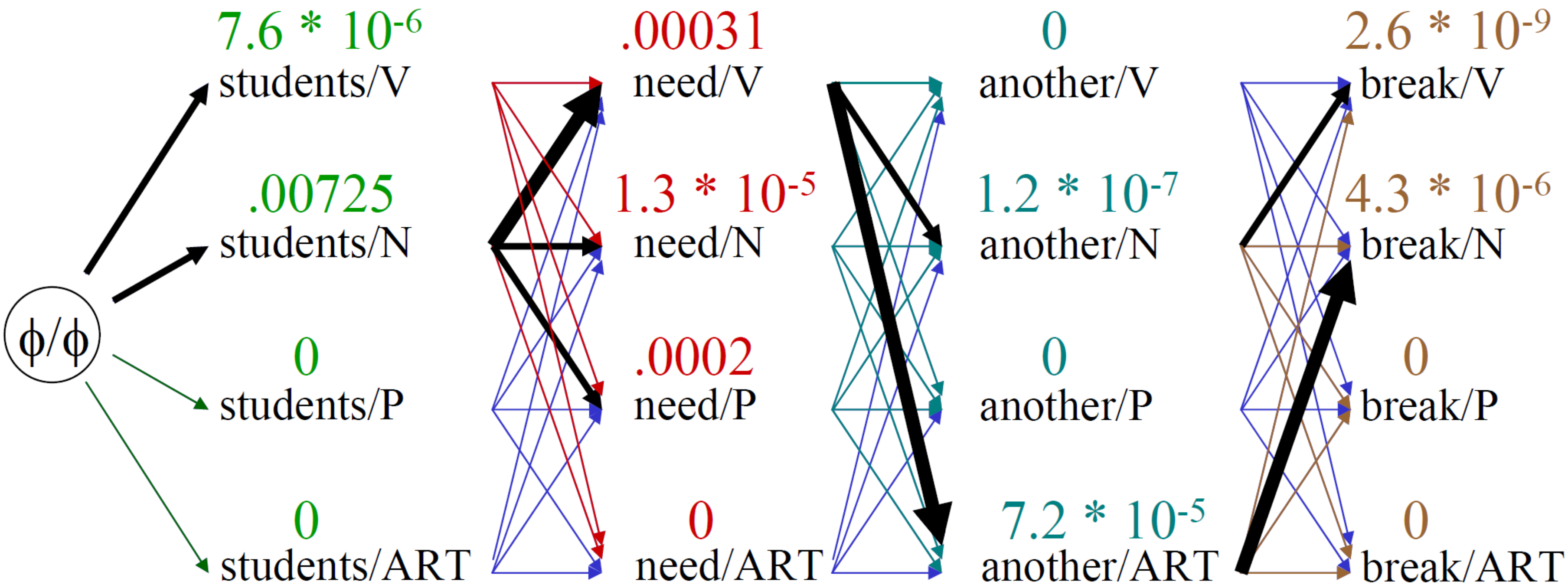
HMM Decoding: Viterbi Algorithm Intuition:



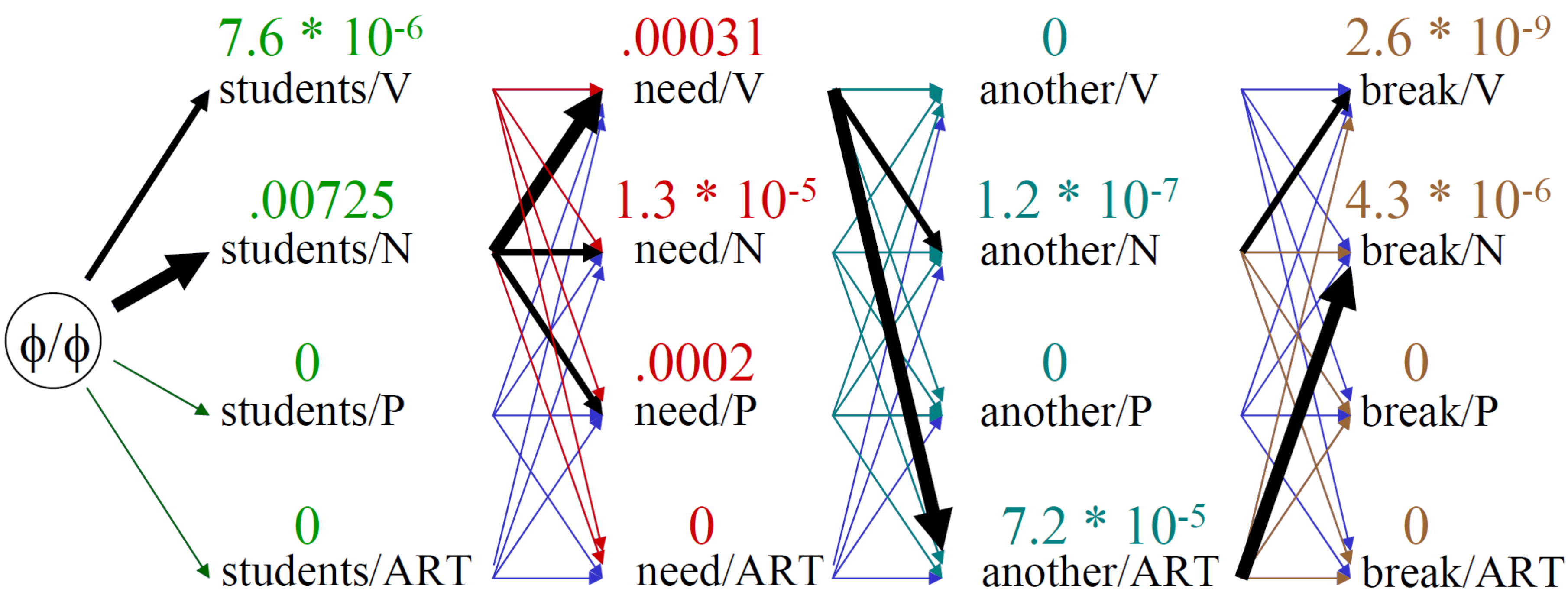
HMM Decoding: Viterbi Algorithm Intuition:



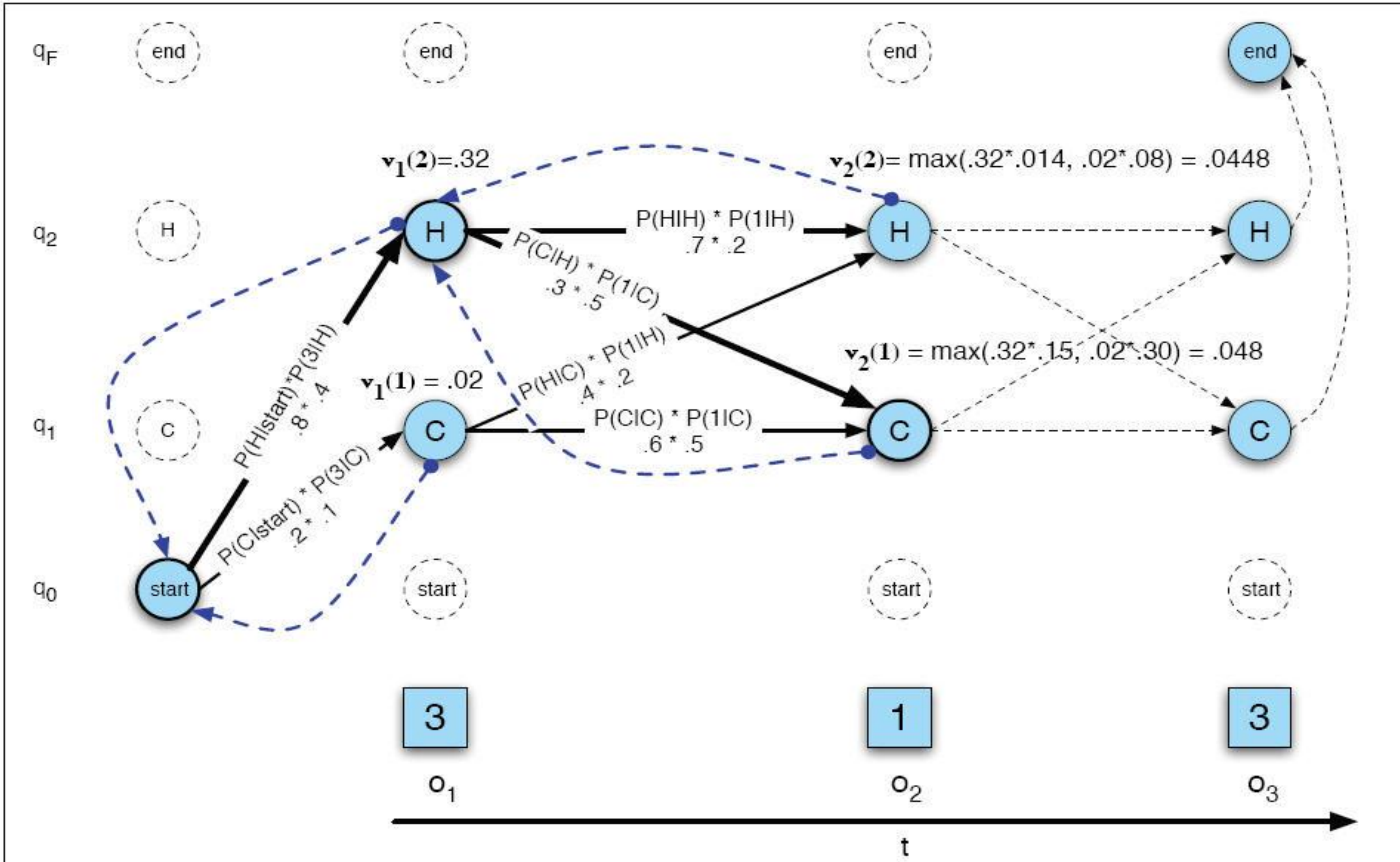
HMM Decoding: Viterbi Algorithm Intuition:



HMM Decoding: Viterbi Algorithm Intuition:



$v_{t-1}(i)$	the **previous Viterbi path probability** from the previous time step
a_{ij}	the **transition probability** from previous state q_i to current state q_j
$b_j(o_t)$	the **state observation likelihood** of the observation symbol o_t given the current state j



$v_{t-1}(i)$	the **previous Viterbi path probability** from the previous time step
a_{ij}	the **transition probability** from previous state q_i to current state q_j
$b_j(o_t)$	the **state observation likelihood** of the observation symbol o_t given the current state j

function VITERBI(*observations* of len T, *state-graph* of len N) **returns** *best-path*

create a path probability matrix *viterbi[N+2,T]*
for each state s **from** 1 **to** N **do** ; initialization step
  $viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$
  $backpointer[s,1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
  for each state s **from** 1 **to** N **do**
    $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
    $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s}$
$viterbi[q_F,T] \leftarrow \max_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$; termination step
$backpointer[q_F,T] \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$; termination step
return the backtrace path by following backpointers to states back in
  time from $backpointer[q_F,T]$

HMM Likelihood of Observation

- Given a sequence of observations, *O,* and a model with a set of parameters, λ, what is the probability that this observation was generated by this model: $P(O|\lambda)$?

HMM Likelihood of Observation

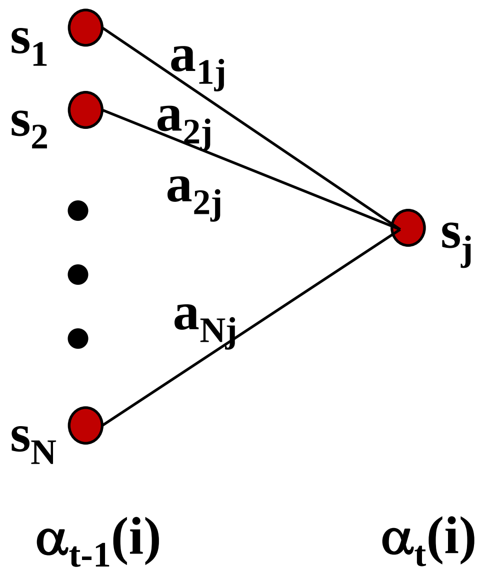
- Due to the Markov assumption, the probability of being in any state at any given time t only relies on the probability of being in each of the possible states at time $t-1$.
- Forward Algorithm: Uses dynamic programming to exploit this fact to efficiently compute observation likelihood in $O(TN^2)$ time.
  - Compute a ***forward trellis*** that compactly and implicitly encodes information about all possible state paths.

Forward Probabilities

- Let $\alpha_t(j)$ be the probability of being in state *j* after seeing the first *t* observations (by summing over all initial paths leading to *j*).

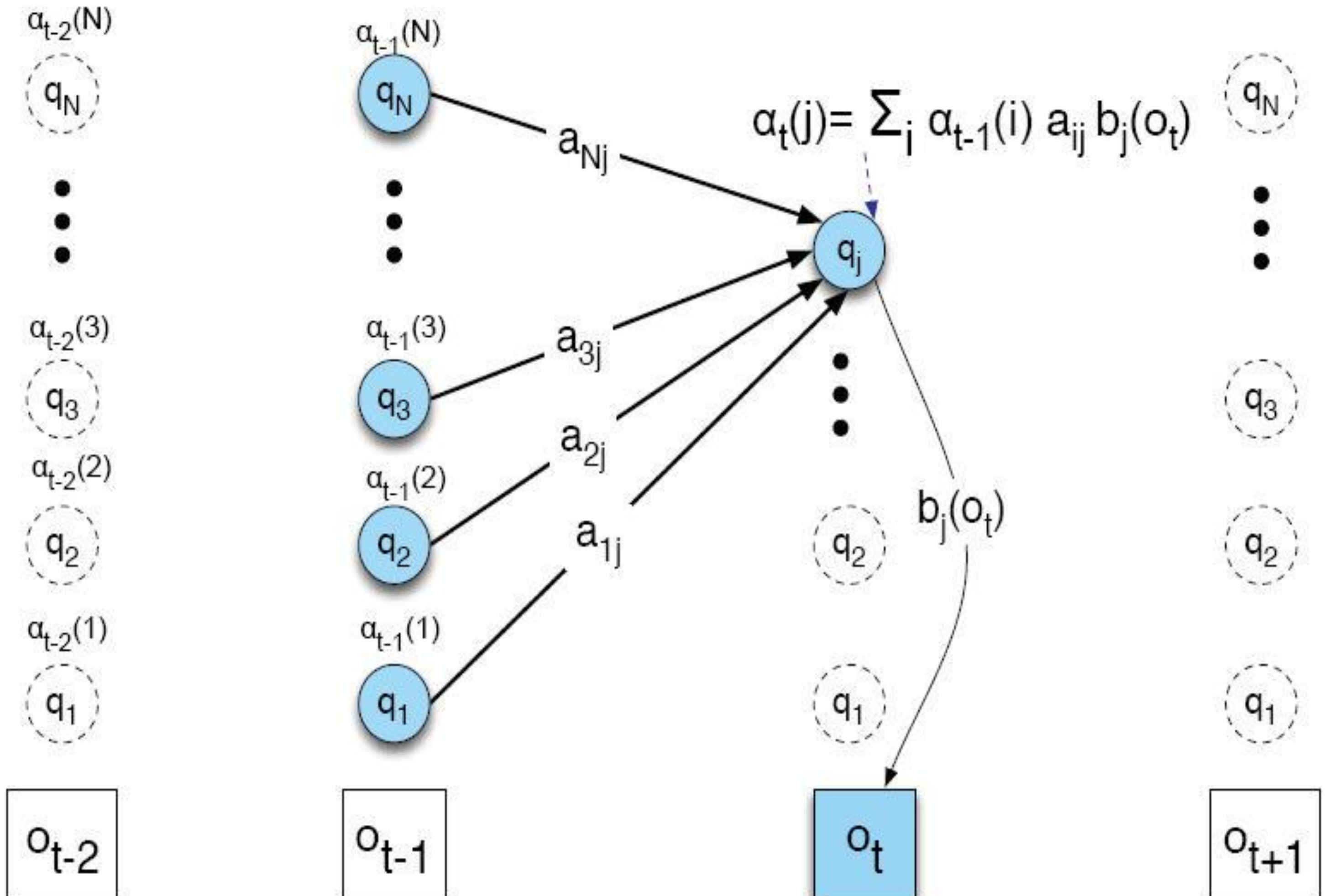
$$\alpha_t(j) = P(o_1, o_2, \ldots o_t,\ q_t = s_j \mid \lambda)$$

Forward Step

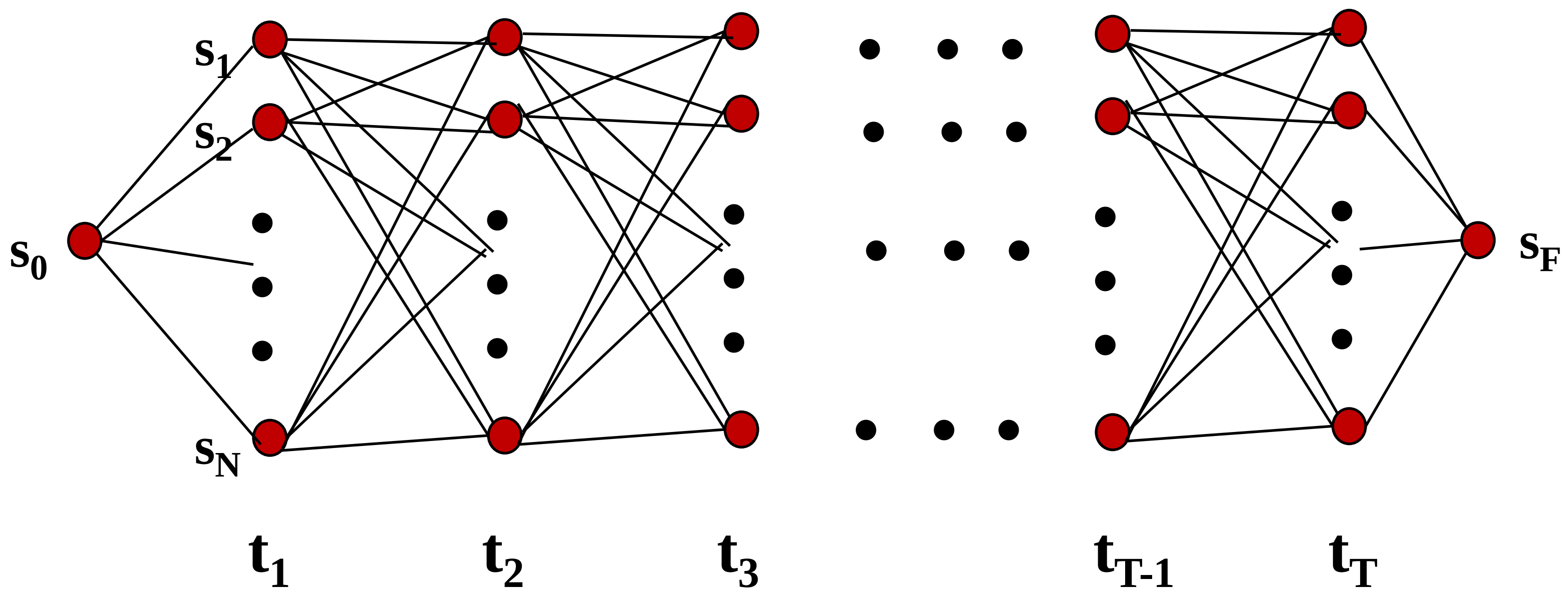


- Consider all possible ways of getting to s_j at time t by coming from all possible states s_i and determine probability of each.
- Sum these to get the total probability of being in state s_j at time t while accounting for the first $t-1$ observations.
- Then multiply by the probability of actually observing o_t in $s_{j.}$

$\alpha_{t-1}(i)$	the **previous forward path probability** from the previous time step
a_{ij}	the **transition probability** from previous state q_i to current state q_j
$b_j(o_t)$	the **state observation likelihood** of the observation symbol o_t given the current state j



Forward Trellis



- Continue forward in time until reaching final time point and sum probability of ending in final state.

$\alpha_{t-1}(i)$	the **previous forward path probability** from the previous time step
a_{ij}	the **transition probability** from previous state q_i to current state q_j
$b_j(o_t)$	the **state observation likelihood** of the observation symbol o_t given the current state j

function FORWARD(*observations* of len T, *state-graph* of len N) **returns** *forward-prob*

create a probability matrix *forward[N+2,T]*

for each state s **from** 1 **to** N **do** ; initialization step

$forward[s,1] \leftarrow a_{0,s} * b_s(o_1)$

for each time step t **from** 2 **to** T **do** ; recursion step

for each state s **from** 1 **to** N **do**

$$forward[s,t] \leftarrow \sum_{s'=1}^{N} forward[s',t-1] * a_{s',s} * b_s(o_t)$$

$$forward[q_F,\mathrm{T}] \leftarrow \sum_{s=1}^{N} forward[s,T] * a_{s,q_F}$$; termination step

return $forward[q_F,T]$

Forward Computational Complexity

- Requires only O(TN^2) time to compute the probability of an observed sequence given a model.
- Exploits the fact that all state sequences must merge into one of the *N* possible states at any point in time and the Markov assumption that only the last state effects the next one.

HMM Learning

- **Supervised Learning:**
  - All training sequences are completely labeled (tagged).
  - That is, nothing is really "hidden" strictly speaking.
  - Learning is very simple ➔ by **MLE estimate**
- **Unsupervised Learning**:
  - All training sequences are unlabeled (tags are unknown)
  - We do assume the number of tags, i.e. states
  - True HMM case. ➔ **Forward-Backward Algorithm**, (also known as "**Baum-Welch algorithm**") which is a special case of ***Expectation Maximization (EM)*** training

HMM Learning: Supervised

- Estimate state transition probabilities based on tag bigram and unigram statistics in the labeled data.

$$a_{ij} = \frac{C(q_t = s_i, \mathrm{q}_{\mathrm{t+1}} = s_j)}{C(q_t = s_i)}$$

- Estimate the observation probabilities based on tag/word co-occurrence statistics in the labeled data.

$$b_j(k) = \frac{C(q_i = s_j, o_i = v_k)}{C(q_i = s_j)}$$

- Use appropriate smoothing if training data is sparse.

HMM Learning: Unsupervised

Sketch of Baum-Welch (EM) Algorithm for Training HMMs

Assume an HMM with N states.
Randomly set its parameters $\lambda=(A,B)$
(making sure they represent legal distributions)
Until converge (i.e. λ no longer changes) do:

E Step: Use the forward/backward procedure to determine the probability of various possible state sequences for generating the training data

M Step: Use these probability estimates to re-estimate values for all of the parameters λ

Backward Probabilities

- Let $\beta_t(i)$ be the probability of observing the final set of observations from time t+1 to T given that one is in state i at time t.

$$\beta_t(i) = P(o_{t+1}, o_{t+2}, \ldots o_T \mid q_t = s_{i,}\ \lambda)$$

Computing the Backward Probabilities

- Initialization

$$\beta_T(i) = a_{iF} \quad 1 \le i \le N$$

- Recursion

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) \quad 1 \le i \le N, \ 1 \le t < T$$

- Termination

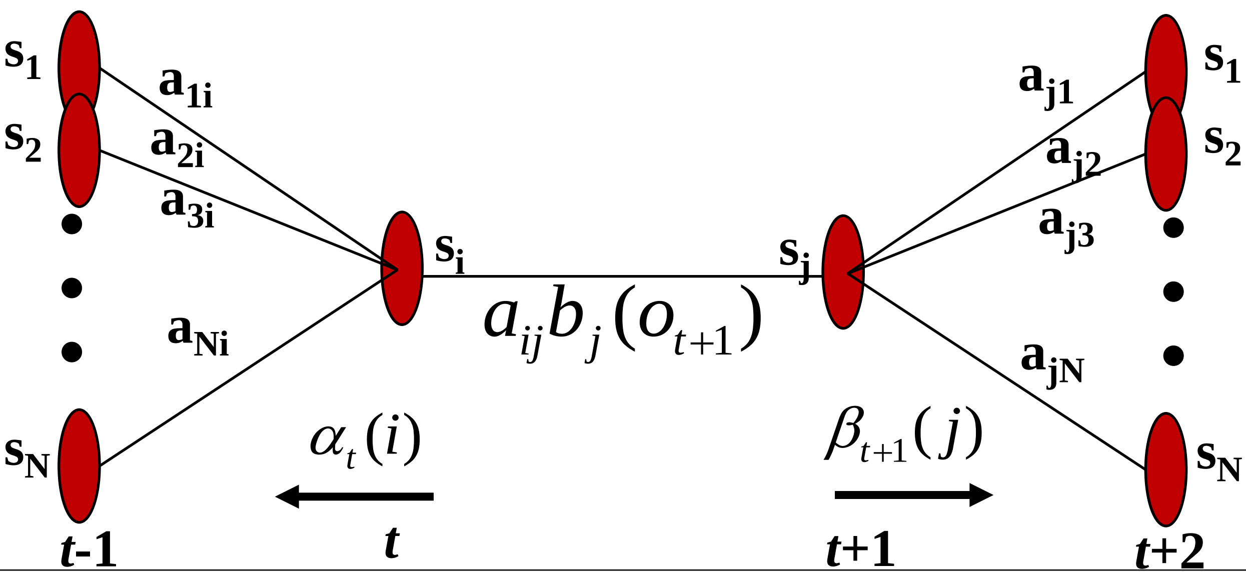
$$P(O \mid \lambda) = \alpha_T(s_F) = \beta_1(s_0) = \sum_{j=1}^{N} a_{0j} b_j(o_1) \beta_1(j)$$

Estimating Probability of State Transitions

- Let $\xi_t(i,j)$ be the probability of being in state *i* at time *t* and state *j* at time *t* + 1

$$\xi_t(i,j) = P(q_t = s_i, q_{t+1} = s_j \mid O, \lambda)$$

$$\xi_t(i,j) = \frac{P(q_t = s_i, q_{t+1} = s_j, O \mid \lambda)}{P(O \mid \lambda)} = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{P(O \mid \lambda)}$$



Re-estimating *A*

$$\hat{a}_{ij} = \frac{\text{expected number of transitions from state } i \text{ to } j}{\text{expected number of transitions from state } i}$$

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \xi_t(i, j)}$$

Estimating Observation Probabilities

- Let $\gamma_t(i)$ be the probability of being in state *i* at time *t* given the observations and the model.

$$\gamma_t(j) = P(q_t = s_j \mid O, \lambda) = \frac{P(q_t = s_j, O \mid \lambda)}{P(O \mid \lambda)} = \frac{\alpha_t(j)\beta_t(j)}{P(O \mid \lambda)}$$

Re-estimating B

$$\hat{b}_j(v_k) = \frac{\text{expected number of times in state } j \text{ observing } v_k}{\text{expected number of times in state } j}$$

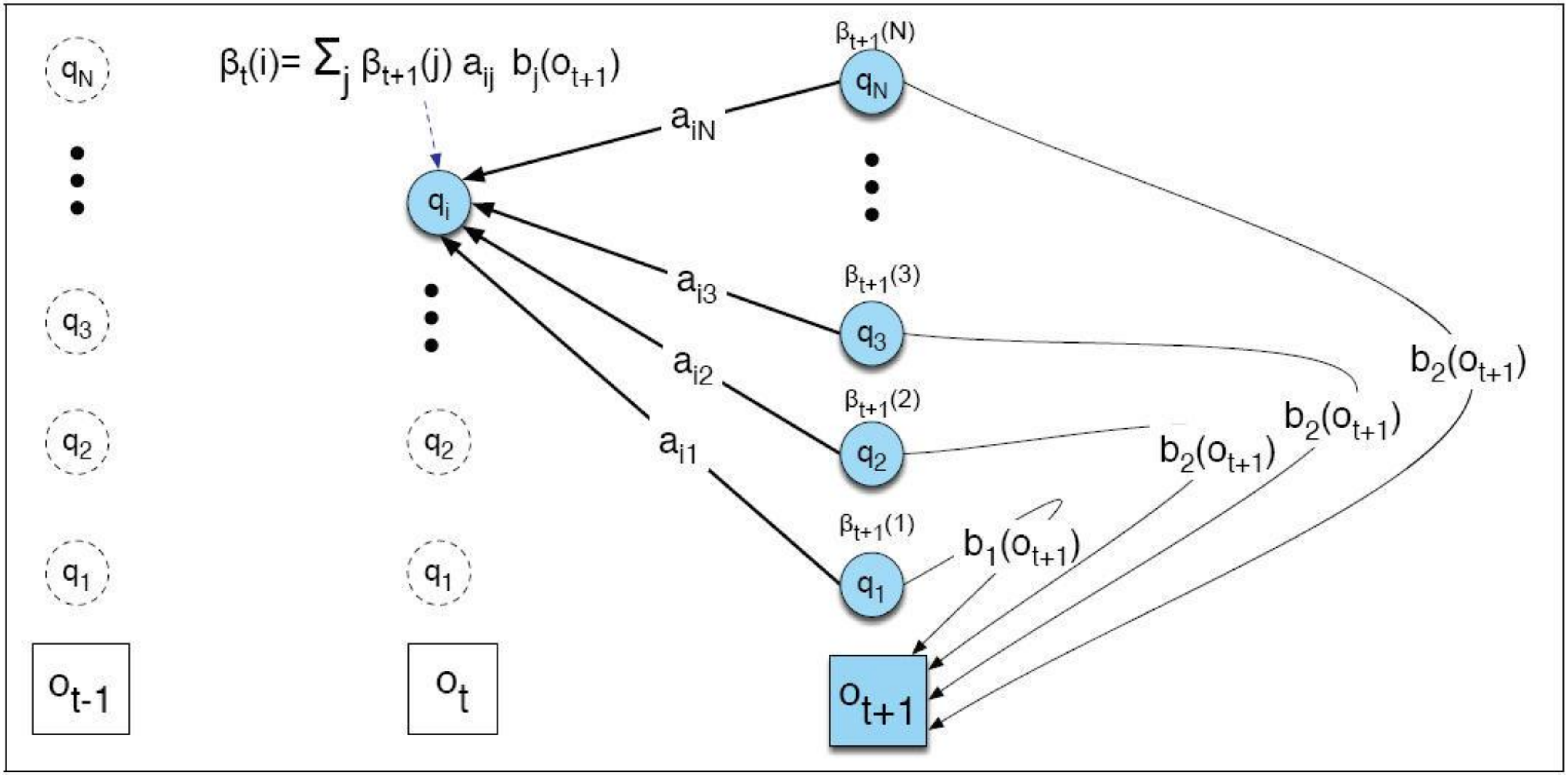
$$\hat{b}_j(v_k) = \frac{\sum_{\mathrm{t=1, s.t. o_t}=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

Pseudocode for Baum-Welch (EM) Algorithm for Training HMMs

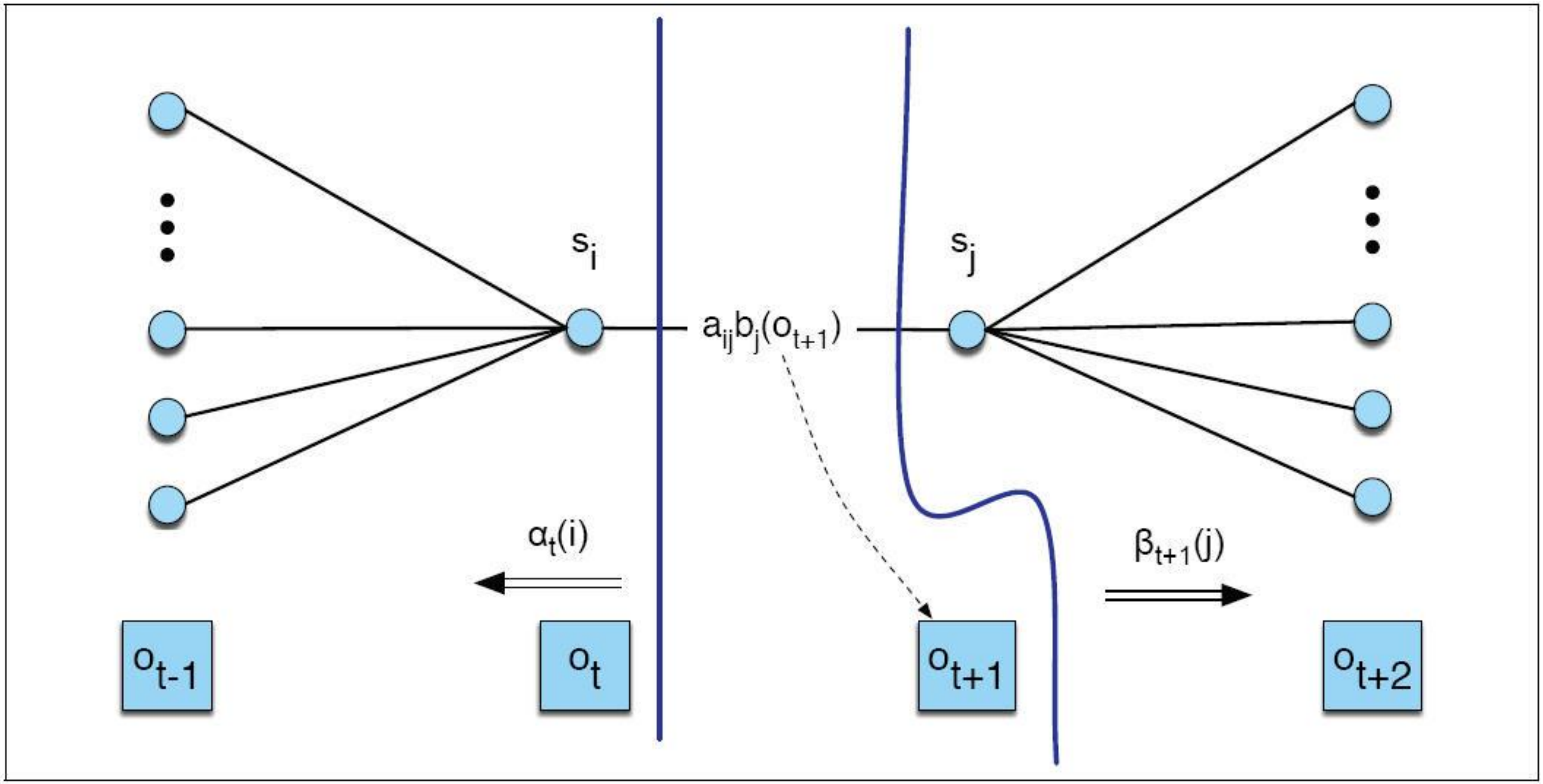
Assume an HMM with N states.
Randomly set its parameters $\lambda=(A,B)$
(making sure they represent legal distributions)
Until converge (i.e. λ no longer changes) do:
E Step:
Compute values for $\gamma_t(j)$ and $\xi_t(i,j)$ using current values for parameters A and B.
M Step:
Re-estimate parameters:

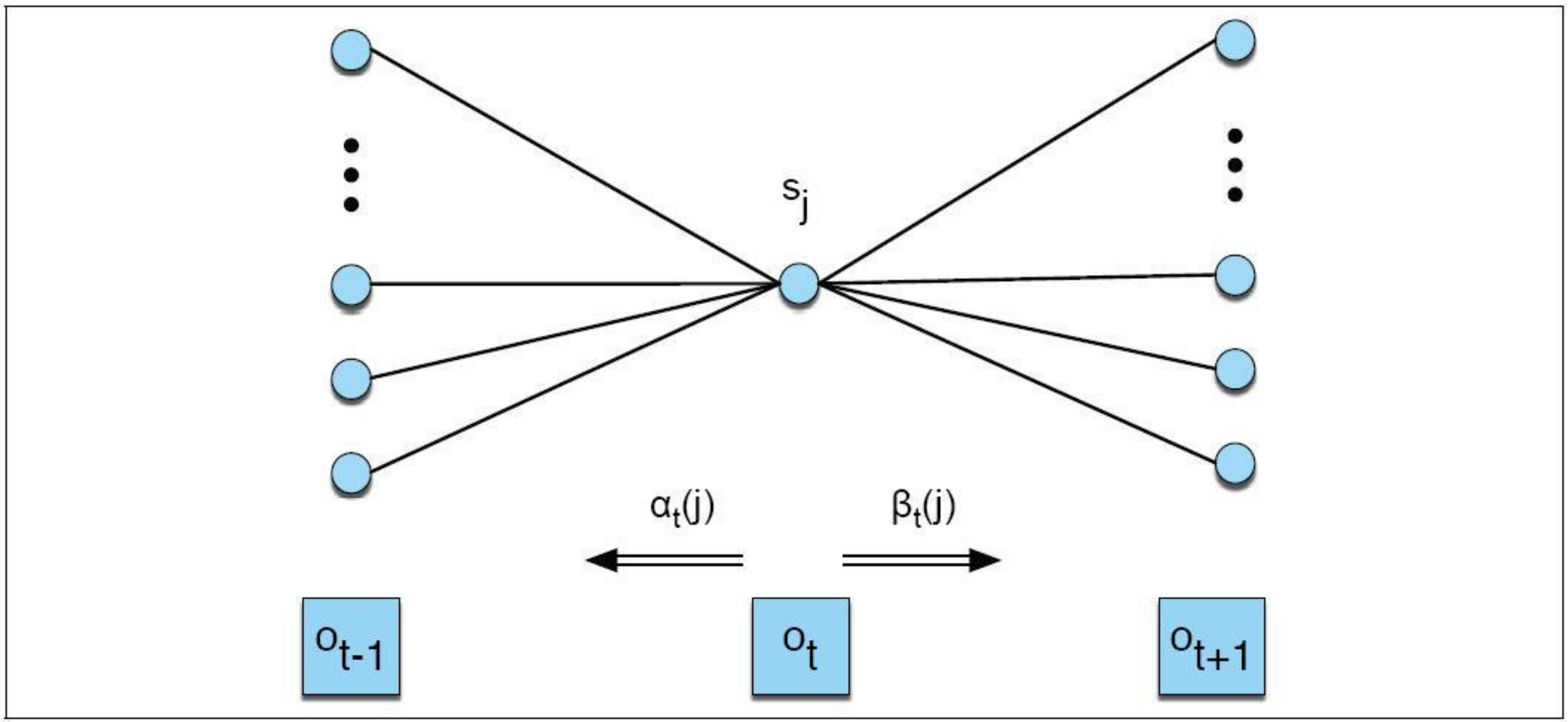
$$a_{ij} = \hat{a}_{ij}$$

$$b_j(v_k) = \hat{b}_j(v_k)$$



$\beta_t(i) = \sum_j \beta_{t+1}(j)\, a_{ij}\ b_j(o_{t+1})$
q_N
q_3
q_2
q_1
q_i
q_2
q_1
$\beta_{t+1}(N)$
q_N
a_{iN}
a_{i3}
a_{i2}
a_{i1}
$\beta_{t+1}(3)$
q_3
$\beta_{t+1}(2)$
q_2
$\beta_{t+1}(1)$
q_1
$b_1(o_{t+1})$
$b_2(o_{t+1})$
$b_2(o_{t+1})$
$b_2(o_{t+1})$
o_{t-1}
o_t
o_{t+1}


s_i
s_j
$a_{ij}b_j(o_{t+1})$
$\alpha_t(i)$
$\beta_{t+1}(j)$
o_{t-1}
o_t
o_{t+1}
o_{t+2}



s_j
$\alpha_t(j)$
$\beta_t(j)$
o_{t-1}
o_t
o_{t+1}

function FORWARD-BACKWARD(*observations* of len T, *output vocabulary* V, *hidden state set* Q) **returns** *HMM=(A,B)*

initialize A and B

iterate until convergence

E-step

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)} \ \forall\, t \text{ and } j$$

$$\xi_t(i,j) = \frac{\alpha_t(i)\, a_{ij} b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(N)} \ \forall\, t,\, i, \text{ and } j$$

M-step

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \xi_t(i,j)}$$

$$\hat{b}_j(v_k) = \frac{\sum_{t=1 s.t.\ O_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

return A, B

EM Properties

- Each iteration changes the parameters in a way that is guaranteed to increase the likelihood of the data: $P(O|\lambda)$.
- Anytime algorithm: Can stop at any time prior to convergence to get approximate solution.
- Converges to a local maximum.

Semi-Supervised Learning

- EM algorithms can be trained with a mix of labeled and unlabeled data.
- EM basically predicts a probabilistic (soft) labeling of the instances and then iteratively retrains using supervised learning on these predicted labels ("self training").
- EM can also exploit supervised data:
  - 1) Use supervised learning on labeled data to initialize the parameters (instead of initializing them randomly).
  - 2) Use known labels for supervised data instead of predicting soft labels for these examples during retraining iterations.

Semi-Supervised Results

- Use of additional unlabeled data improves on supervised learning when amount of labeled data is very small and amount of unlabeled data is large.
- Can degrade performance when there is sufficient labeled data to learn a decent model and when unsupervised learning tends to create labels that are incompatible with the desired ones.
  - There are negative results for semi-supervised POS tagging since unsupervised learning tends to learn semantic labels (e.g. eating verbs, animate nouns) that are better at predicting the data than purely syntactic labels (e.g. verb, noun).

Conclusions

- POS Tagging is the lowest level of syntactic analysis.
- It is an instance of sequence labeling, a collective classification task that also has applications in information extraction, phrase chunking, semantic role labeling, and bioinformatics.
- HMMs are a standard generative probabilistic model for sequence labeling that allows for efficiently computing the globally most probable sequence of labels and supports supervised, unsupervised and semi-supervised learning.