Part-Of-Speech (POS) Tagging: Hidden Markov Model (HMM) algorithm

Hidden Markov Models

- Recall that we estimated the best probable tag sequence for a given sequence of words as:

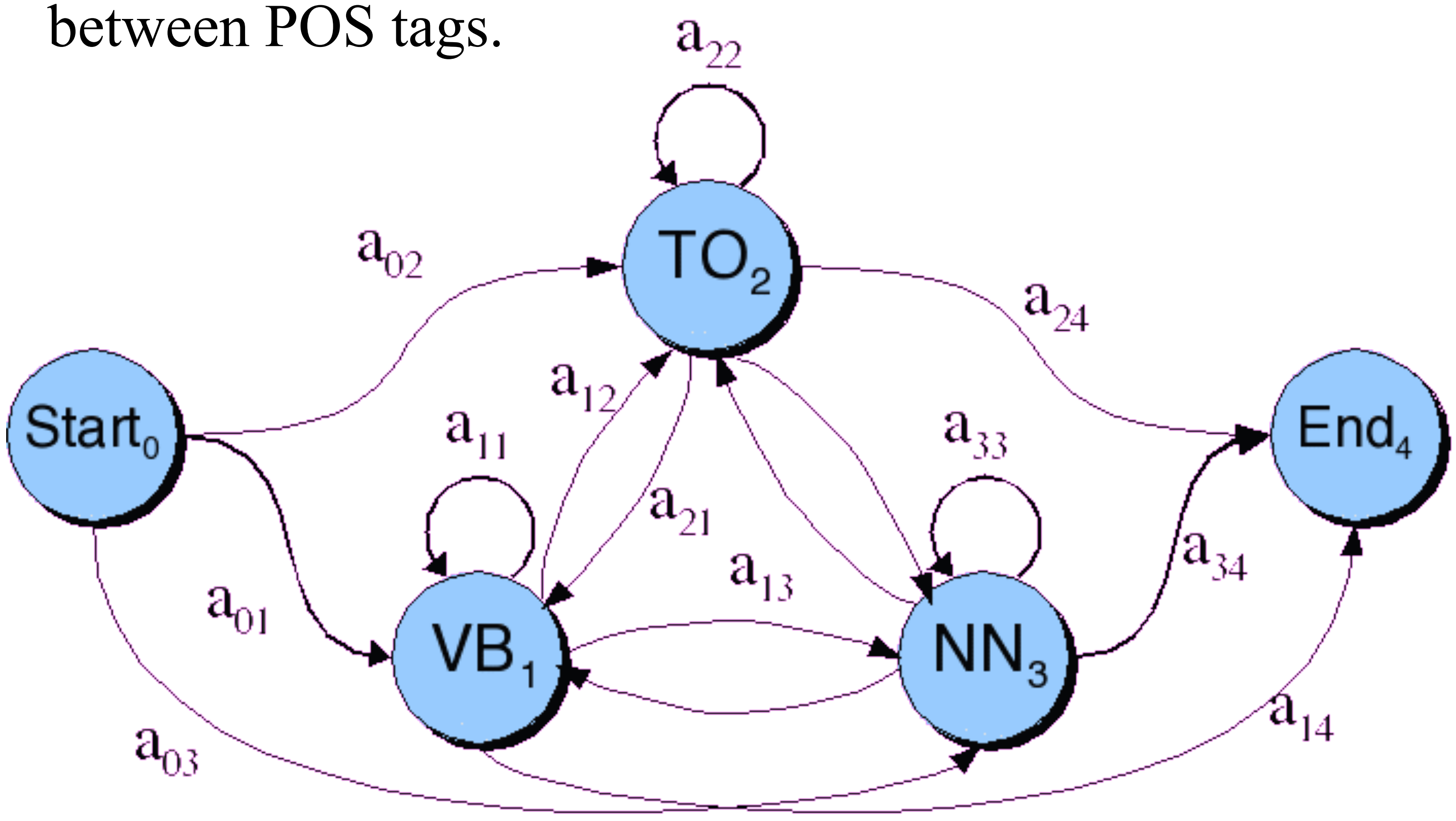
$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(w_i | t_i) P(t_i | t_{i-1})$$

  with the word likelihood x the tag transition probabilities

- When we evaluated the probabilities by hand for a sentence, we could pick the optimum tag sequence
- But in general, we need an optimization algorithm to most efficiently pick the best tag sequence without computing all possible combinations of tag sequence probabilities
- What we've described with these two kinds of probabilities is a Hidden Markov Model
  - the Markov Model is the sequence of words and the hidden states are the POS tags for each word.

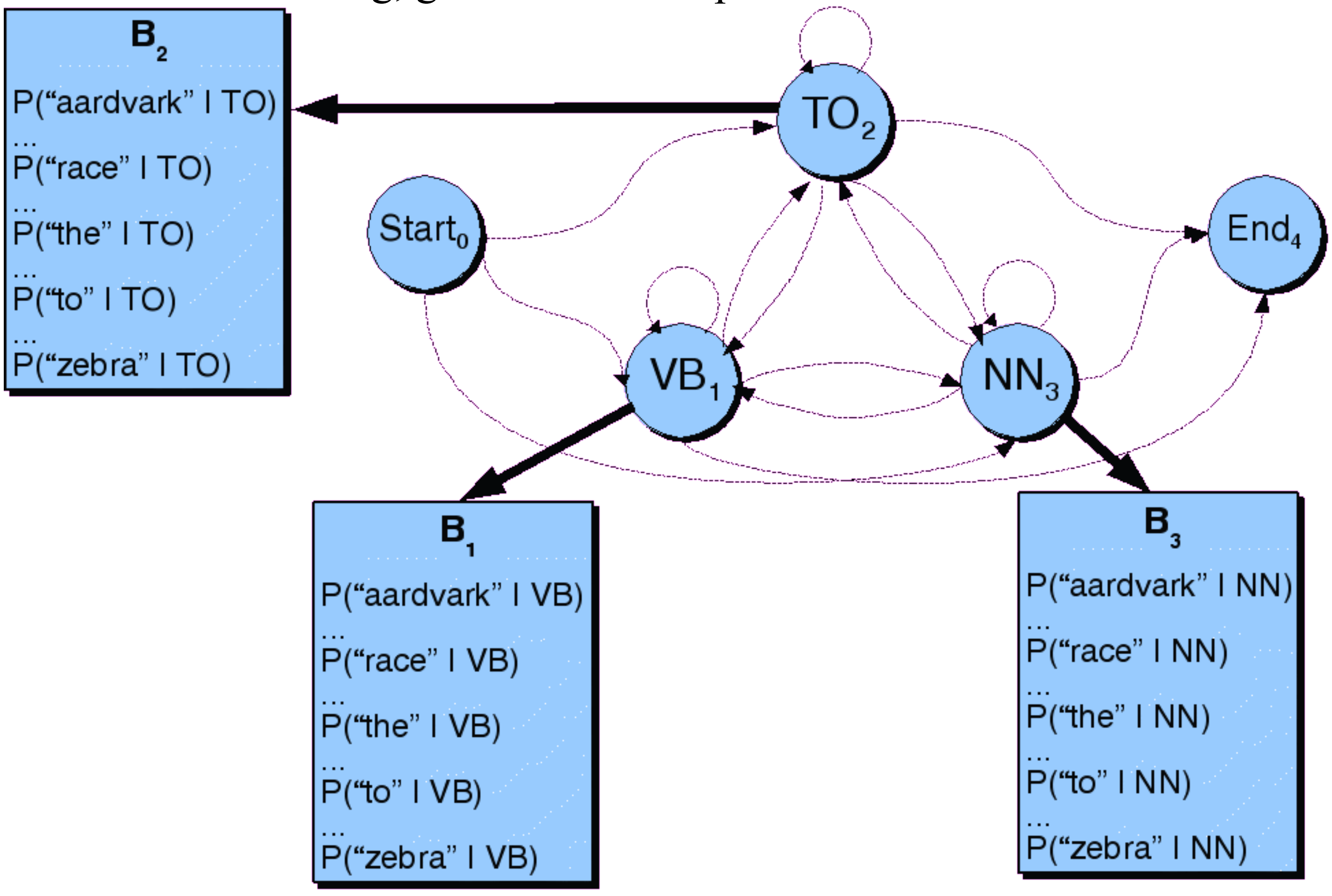
Tag Transition Probabilities for an HMM

- The HMM hidden states, the POS tags, can be represented in a graph where the edges are the transition probabilities between POS tags.



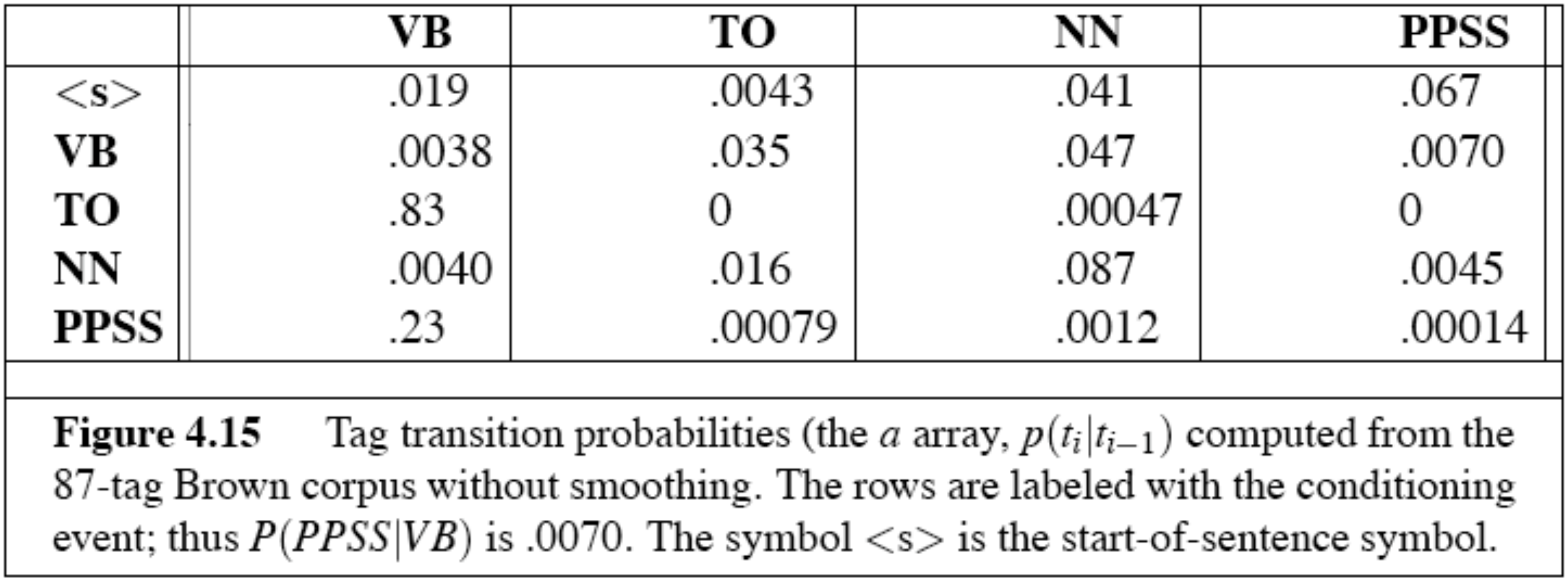
Word likelihoods for POS HMM

- For each POS tag, give words with probabilities



The A matrix for the POS HMM

- Example of tag transition probabilities represented in a matrix, usually called the A matrix in an HMM:
  - The probability that VB follows <s> is .019, …

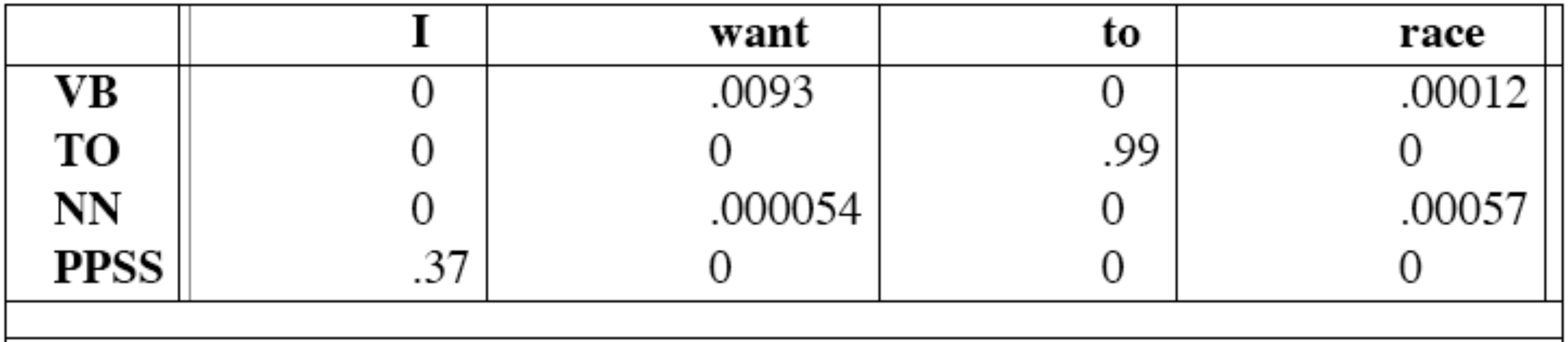


	VB	TO	NN	PPSS
<s>	.019	.0043	.041	.067
VB	.0038	.035	.047	.0070
TO	.83	0	.00047	0
NN	.0040	.016	.087	.0045
PPSS	.23	.00079	.0012	.00014

Figure 4.15 Tag transition probabilities (the *a* array, $p(t_i|t_{i-1})$ computed from the 87-tag Brown corpus without smoothing. The rows are labeled with the conditioning event; thus $P(PPSS|VB)$ is .0070. The symbol <s> is the start-of-sentence symbol.

The B matrix for the POS HMM

- Word likelihood probabilities are represented in a matrix, where for each tag, we show the probability that the tag on that word



	I	**want**	**to**	**race**
VB	0	.0093	0	.00012
TO	0	0	.99	0
NN	0	.000054	0	.00057
PPSS	.37	0	0	0

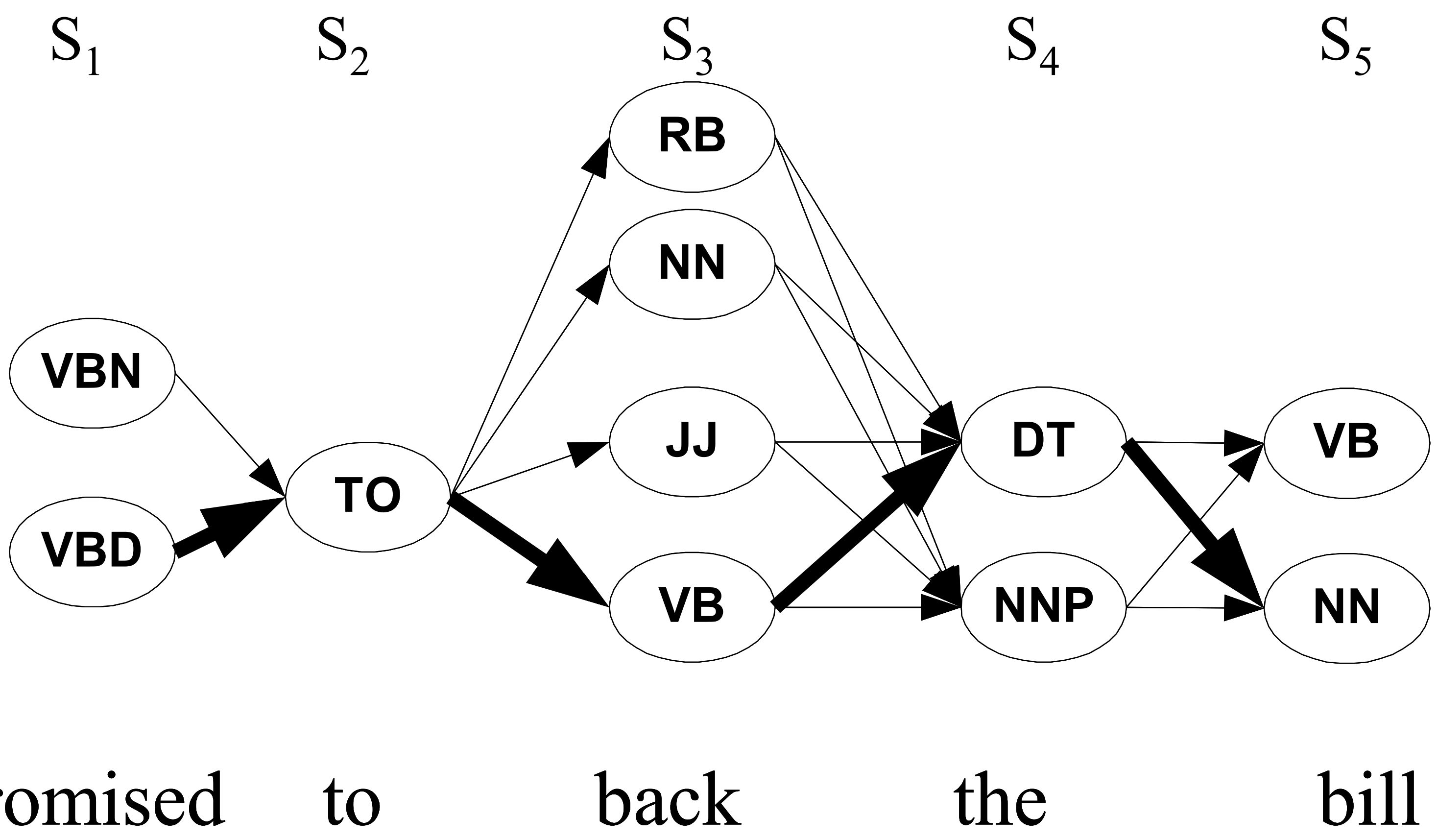
Figure 4.16 Observation likelihoods (the *b* array) computed from the 87-tag Brown corpus without smoothing.

Using HMMs for POS tagging

- From the tagged corpus, create a tagger by computing the two matrices of probabilities, A and B
  - Straightforward for bigram HMM, done by counting
  - For higher-order HMMs, efficiently compute matrix by the forward-backward algorithm
- To apply the HMM tagger to unseen text, we must find the best sequence of transitions
  - Given a sequence of words, find the sequence of states (POS tags) with the highest probabilities along the path
  - This task is sometimes called "decoding"
  - Use the Viterbi algorithm

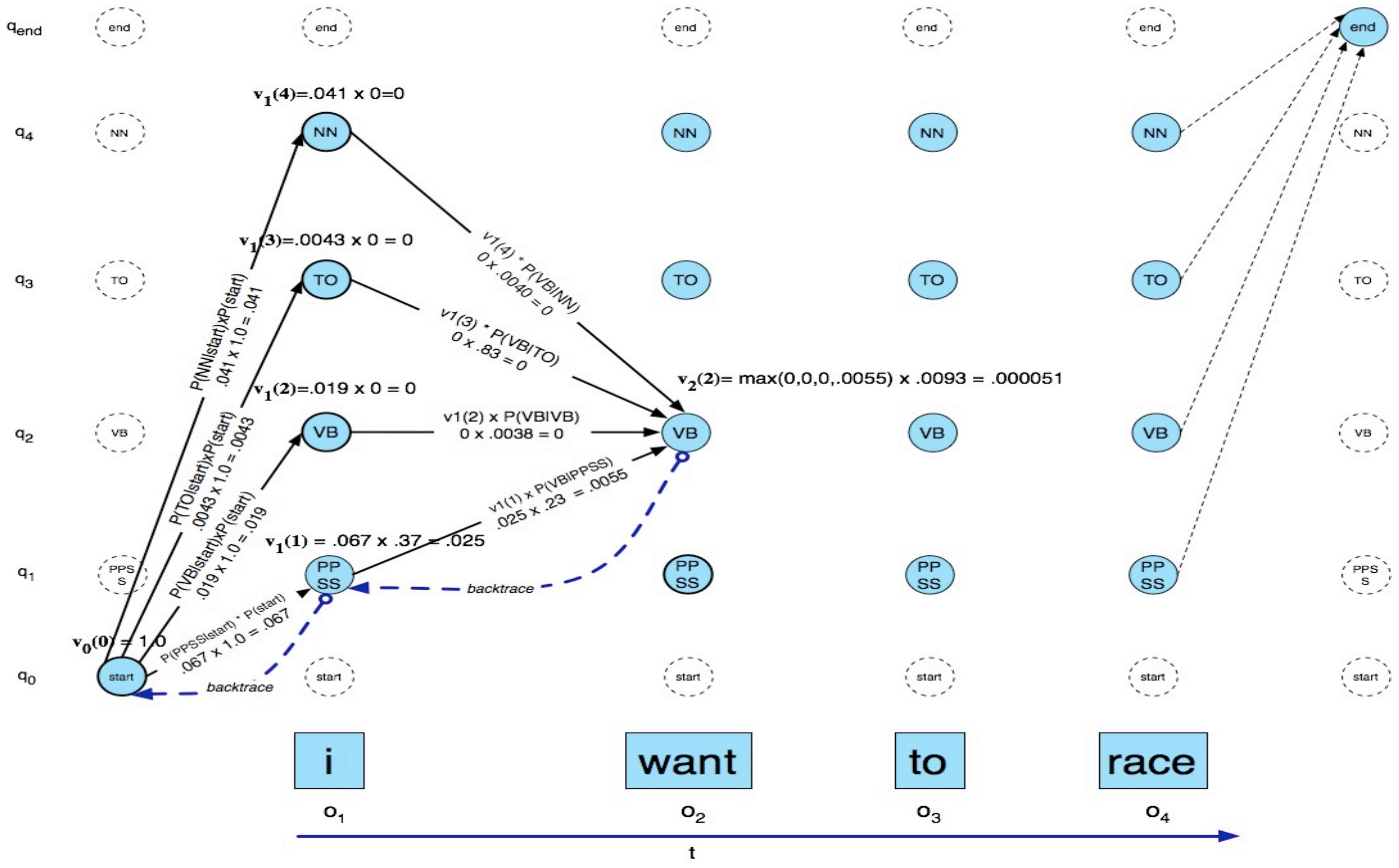
Viterbi intuition: we are looking for the best 'path'

Each word has states representing the possible POS tags:



Viterbi example

Each pair of tags labeled with an edge giving transition probability
Each tag in a state labeled with a Viterbi value giving max over states in previous word of:
(its viterbi value * transition prob * word likelihood), representing "best path to this node"



q_{end}
q_4
q_3
q_2
q_1
q_0
end
NN
TO
VB
PPSS
start
$v_1(4)$=.041 x 0=0
$v_1(3)$=.0043 x 0 = 0
$v_1(2)$=.019 x 0 = 0
$v_1(1)$ = .067 x .37 = .025
$v_0(0)$ = 1.0
P(NN|start)xP(start) .041 x 1.0 = .041
P(TO|start)xP(start) .0043 x 1.0 = .0043
P(VB|start)xP(start) .019 x 1.0 = .019
P(PPSS|start) * P(start) .067 x 1.0 = .067
v1(4) * P(VB|NN) 0 x .0040 = 0
v1(3) * P(VB|TO) 0 x .83 = 0
v1(2) x P(VB|VB) 0 x .0038 = 0
v1(1) x P(VB|PPSS) .025 x .23 = .0055
$v_2(2)$= max(0,0,0,.0055) x .0093 = .000051
backtrace
backtrace
i
want
to
race
o_1
o_2
o_3
o_4
t

Viterbi Algorithm sketch

- This algorithm fills in the elements of the array viterbi in the previous slide (cols are words, rows are states (POS tags))

```
function Viterbi
    for each state s, compute the initial column
        viterbi[s, 1] = A[0, s] * B[s, word1]
    for each word w from 2 to N (length of sequence)
        for each state s, compute the column for w
            viterbi[s, w] =
                    max over s' ( viterbi[s',w-1] * A[s',s] * B[s,w])
            <save back pointer to trace final path>
    return the trace of back pointers
```

where A is the matrix of state transitions

and B is the matrix of state/word likelihoods