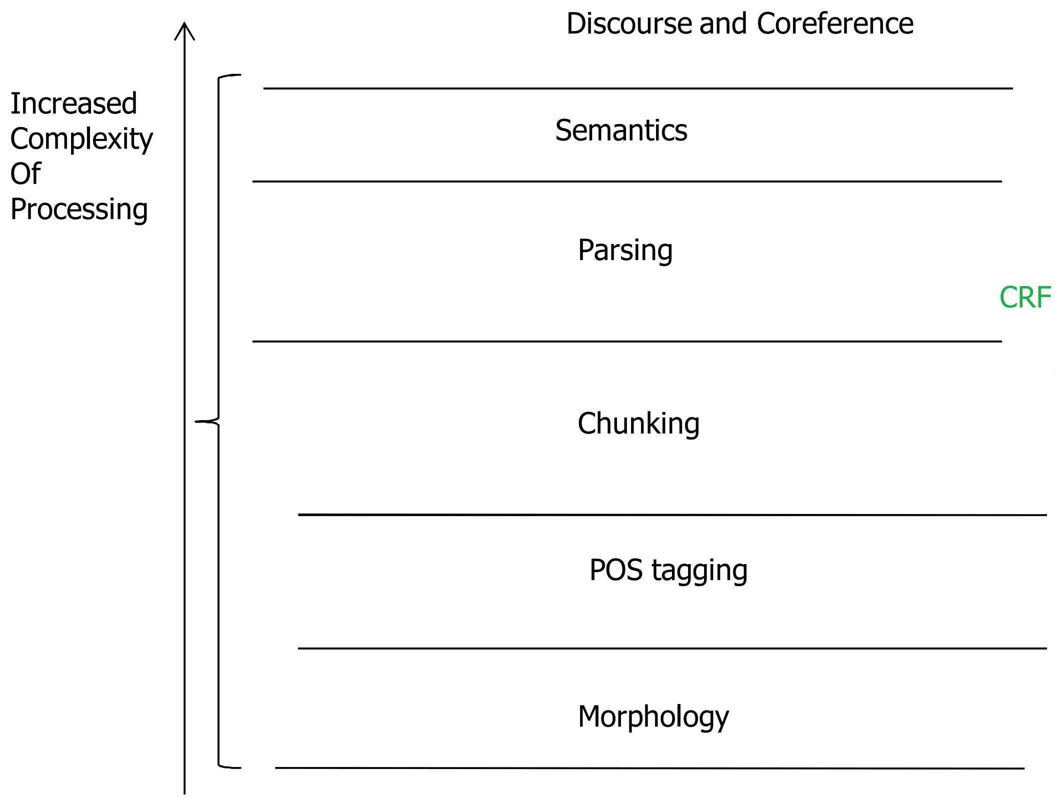
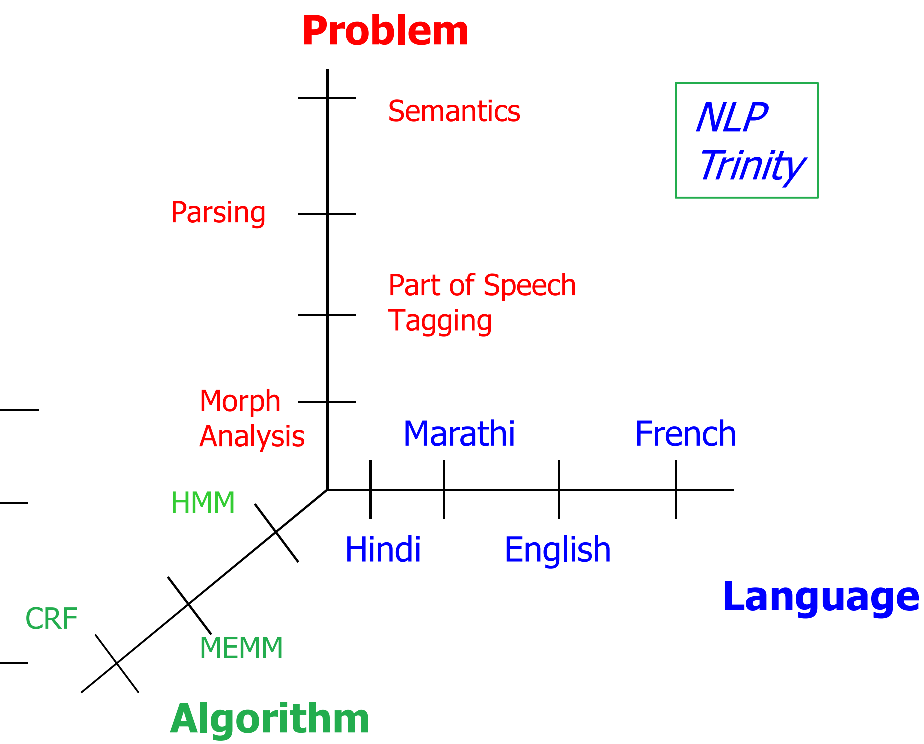
Natural Language Processing

Pushpak Bhattacharyya
CSE Dept,
IIT Patna

POS tagging

NLP: multilayered, multidimensional





Part of Speech Tagging

With Hidden Markov Model

NLP Layer

What a gripping movie was Dangal!

What/WP a/DT gripping/JJmovie/NN was/VBD Dangal/NNP !/.

Parse

```
(ROOT
  (FRAG
    (SBAR
      (WHNP
        (WP What))
        (S
          (NP
            (DT a)
            (JJ gripping)
            (NN movie)
          )
          (VP
            (VBD was)
            (NP
            (NNP Dangal)))))
          (. !)
  )
)
```

Universal dependencies

dobj(Dangal-6, What-1)
det(movie-4, a-2)
amod(movie-4, gripping-3)
nsubj(Dangal-6, movie-4)
cop(Dangal-6, was-5)
root(ROOT-0, Dangal-6

Part of Speech Tagging

- POS Tagging: attaches to each word in a sentence a part of speech tag from a given set of tags called the **Tag-Set**
- Standard Tag-set : Penn Treebank (for English).

POS ambiguity instances

best ADJ ADV NP V
better ADJ ADV V DET
close ADV ADJ V N
cut V N VN VD
even ADV DET ADJ V
grant NP N V –
hit V VD VN N
lay ADJ V NP VD
left VD ADJ N VN
like CNJ V ADJ P –
near P ADV ADJ DET
open ADJ V N ADV
past N ADJ DET P
present ADJ ADV V N
read V VN VD NP
right ADJ N DET ADV
second NUM ADV DET N
set VN V VD N –
that CNJ V WH DET

Part-of-speech tag

- A word can have more than one POS tags.

Adjective

- E.g.
  1. *What a* ***gripping*** *movie was Dangal!*
  2. *He is* ***gripping*** *it firm.*

Verb

Linguistic fundamentals

- A word can have two roles
  - Grammatical role (Dictionary POS tag)
  - Functional role (Contextual POS tag)

  - E.g. ***<u>Golf</u> stick***

- POS tag of "*Golf*"
  - Grammatical: Noun
  - Functional: Adjective (+ al)

The "al" rule!

- If a word has different functional POS tag than its grammatical pos then add **"al"** to the functional POS tag

- E.g. ***Golf*** *stick*

Adjective + al

Adjectival

Noun	**+ al**	**= Nominal**
Verb	**+ al**	**= Verbal**
Adjective	**+ al**	**= Adjectival**
Adverb	**+ al**	**= Adverbial**

The “al” rule cntd.

- Examples:
  - Nominal
    - *Many don’t understand the problem of* ***hungry****.*
  - Adverbial
    - *Come* ***quick.***
  - Verbal

POS tagging as an ML problem

- Question
  - Is one instance of example enough for ML?
  - E.g. Known example of "people"

    People → Noun

    POS Ambiguity

  - But it can be verb as well

    People → Verb (to populate)
- Answer
  - We need at least as many instances as number of different labels (POS tags)-1 to make decision.

Disambiguation of POS tag

- If no ambiguity, learn a table of words and its corresponding tags.
- If ambiguity, then look for the contextual information i.e. look-back or look-ahead.

Data for “present”

1. He gifted me the/a/this/that **present_NN**.
2. They **present_VB** innovative ideas.
3. He was **present_JJ** in the class.

Rules for disambiguating “present”

- For Present_NN (look-back)
  - If present is preceded by determiner (the/a) or demonstrative (this/that), then POS tag will be noun.

  - Does this rule guarantee 100% precision and 100% recall?
  - False positive:
    - *The **present_ADJ** case is not convincing.*

Adjective preceded by “the”

  - False negative:
    - ***Present** foretells the future.*

Noun but not preceded by “the”

Rules for disambiguating "present"

- For Present_NN (look-back and look ahead)
  - If present is preceded by determiner (the/a) or demonstrative (this/that) or followed by a verb, then POS tag will be noun.
  - E.g.
    - **Present_NN** will tell the future.
    - **Present_NN** fortells the future.

  - Does this rule guarantee 100% precision and 100% recall?

Need for ML in POS tagging

- New examples break rules, so we need a robust system.

- Machine learning based POS tagging:
  - HMM (Accuracy increased by 10-20% against rule based systems)
  - Jelinek's work

Mathematics of POS tagging

Argmax computation (1/2)

Best tag sequence
= T*
= argmax P(T|W)
= argmax P(T)P(W|T) (by Baye's Theorem)

$$P(T) = P(t_0{=}\hat{}\ t_1 t_2 \ldots t_{n+1}{=}.)$$
$$= P(t_0)P(t_1|t_0)P(t_2|t_1t_0)P(t_3|t_2t_1t_0) \ldots P(t_n|t_{n-1}t_{n-2}\ldots t_0)P(t_{n+1}|t_nt_{n-1}\ldots t_0)$$
$$= P(t_0)P(t_1|t_0)P(t_2|t_1) \ldots P(t_n|t_{n-1})P(t_{n+1}|t_n)$$

$$= \prod_{i=0}^{N+1} P(t_i|t_{i-1})$$ Bigram Assumption

Argmax computation (2/2)

$$P(W|T) = P(w_0|t_0\text{-}t_{n+1})P(w_1|w_0t_0\text{-}t_{n+1})P(w_2|w_1w_0t_0\text{-}t_{n+1}) \ldots P(w_n|w_0\text{-}w_{n-1}t_0\text{-}t_{n+1})P(w_{n+1}|w_0\text{-}w_nt_0\text{-}t_{n+1})$$

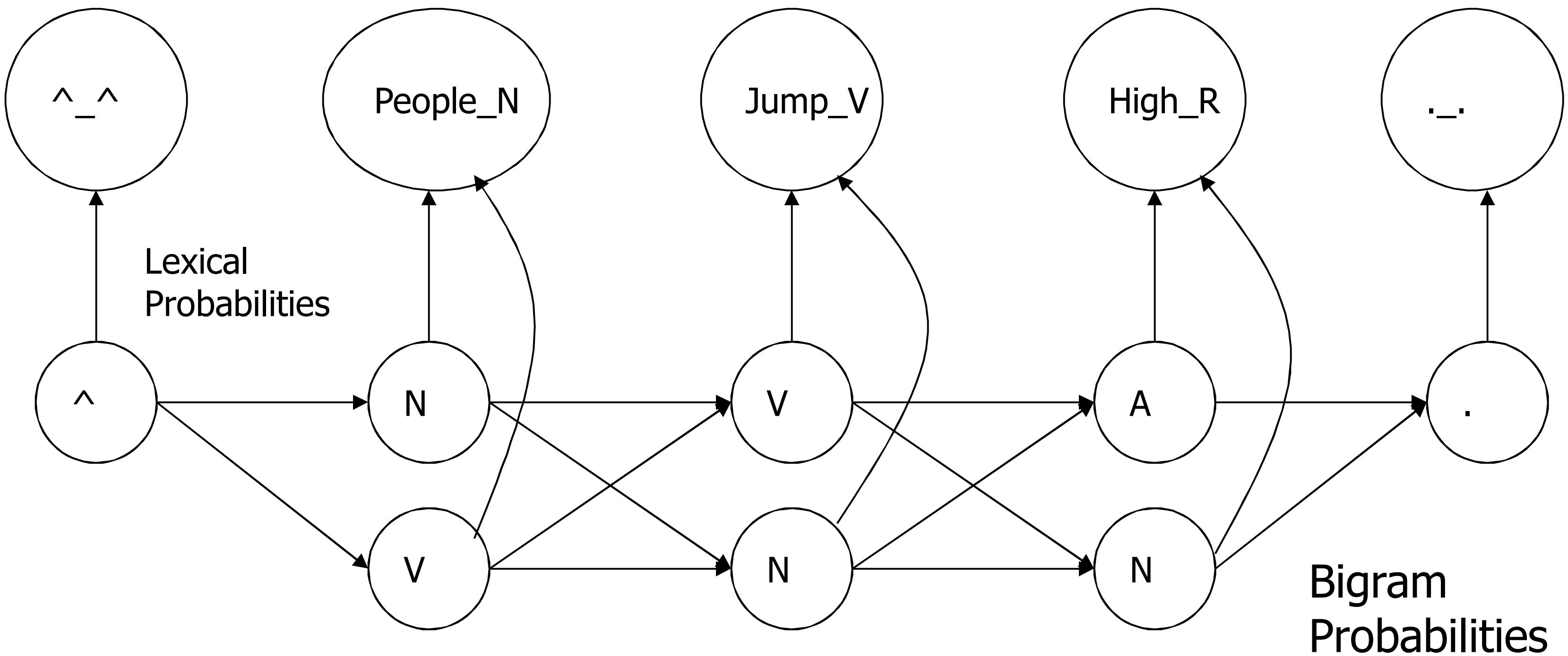
Assumption: A word is determined completely by its tag. This is inspired by speech recognition

$$= P(w_o|t_o)P(w_1|t_1) \ldots P(w_{n+1}|t_{n+1})$$

$$= \prod_{i=0}^{n+1} P(w_i|t_i)$$

$$= \prod_{i=1}^{n+1} P(w_i|t_i) \qquad \text{(Lexical Probability Assumption)}$$

Generative Model



This model is called Generative model.
Here words are observed from tags as states.
This is similar to HMM.

Typical POS tag steps

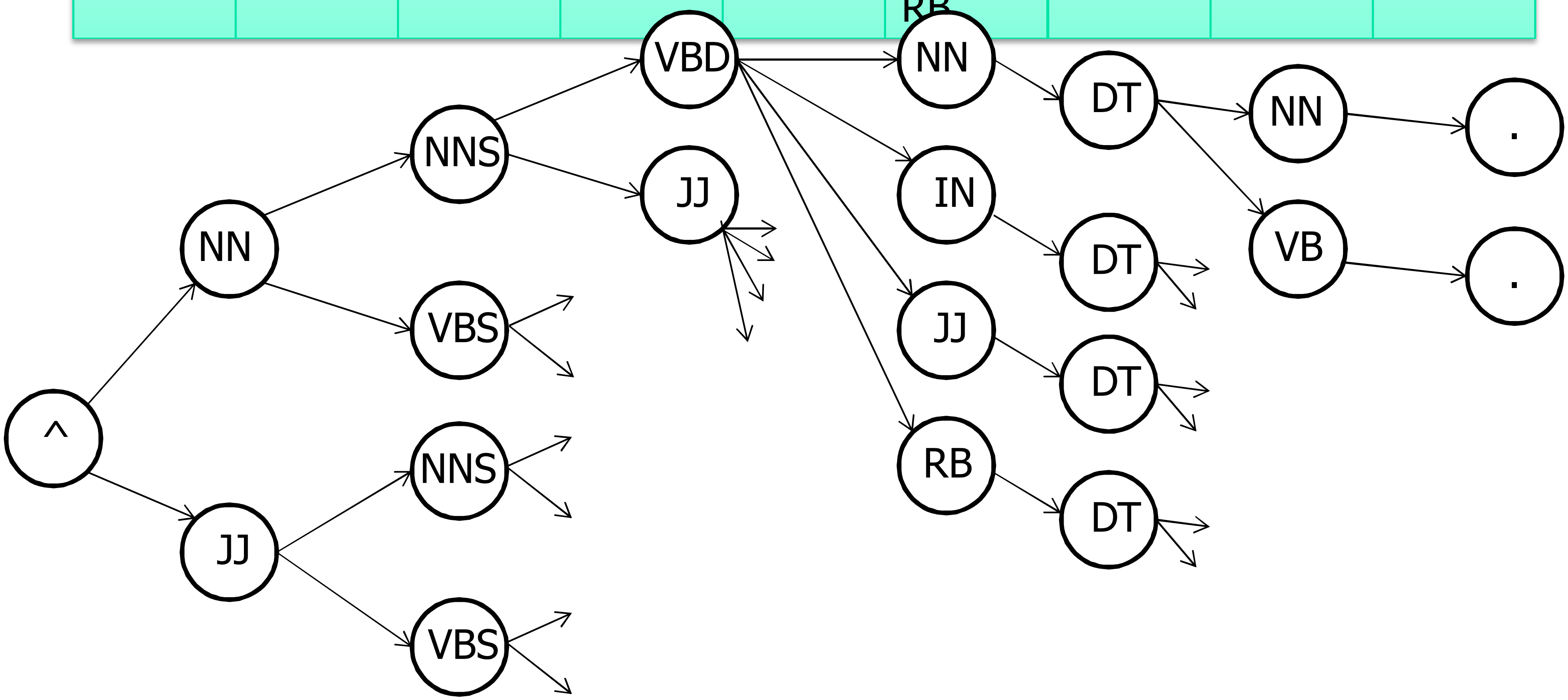
- Implementation of Viterbi – Unigram, Bigram.
- Five Fold Evaluation.
- Per POS Accuracy.
- Confusion Matrix.

Screen shot of typical Confusion Matrix

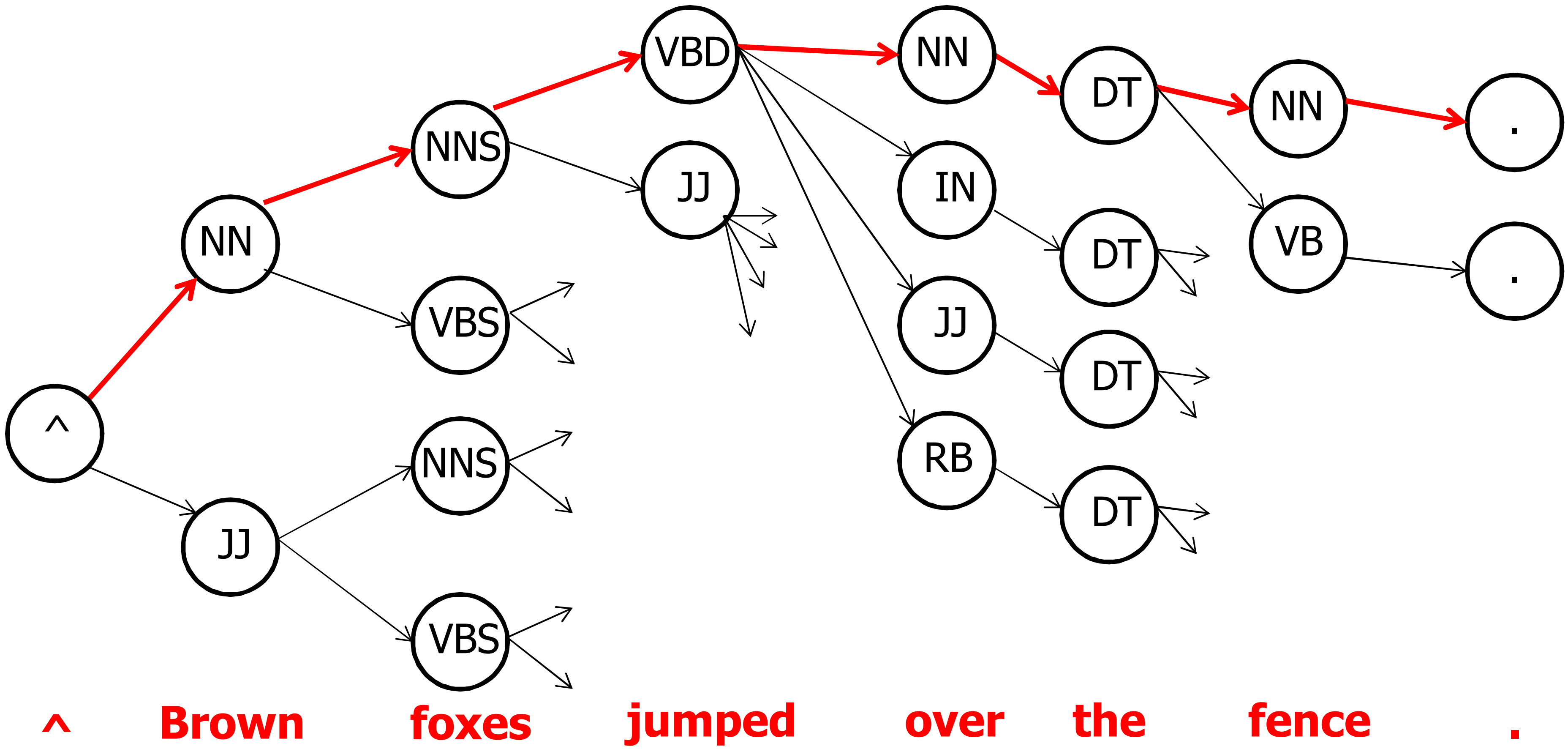
	AJ0	AJ0-AV0	AJ0-NN1	AJ0-VVD	AJ0-VVG	AJ0-VVN	AJC	AJS	AT0	AV0	AV0-AJ0	AVP
AJ0	2899	20	32	1	3	3	0	0	18	35	27	1
AJ0-AV0	31	18	2	0	0	0	0	0	0	1	15	0
AJ0-NN1	161	0	116	0	0	0	0	0	0	0	1	0
AJ0-VVD	7	0	0	0	0	0	0	0	0	0	0	0
AJ0-VVG	8	0	0	0	2	0	0	0	1	0	0	0
AJ0-VVN	8	0	0	3	0	2	0	0	1	0	0	0
AJC	2	0	0	0	0	0	69	0	0	11	0	0
AJS	6	0	0	0	0	0	0	38	0	2	0	0
AT0	192	0	0	0	0	0	0	0	7000	13	0	0
AV0	120	8	2	0	0	0	15	2	24	2444	29	11
AV0-AJ0	10	7	0	0	0	0	0	0	0	16	33	0
AVP	24	0	0	0	0	0	0	0	1	11	0	737

Computation of POS tags

W:	^	Brown	foxes	jumped	over	the	fence	.
T:	^	JJ	NNS	VBD	NN	DT	NN	.
		NN	VBS	JJ	IN		VB	
					JJ			
					RB			



^ **Brown** **foxes** **jumped** **over** **the** **fence** **.**



Probability of a path (e.g. Top most path) = *P*(T) * *P*(W|T)

P(^) . P(NN|^) . P(NNS|NN) . P(VBD|NNS) . P(NN|VBD) . P(DT|NN) . P(NN|DT) . P(.|NN) . P(.)

*

P(^|^) . P(brown|NN) . P(foxes|NNS) . P(jumped|VBD) . P(over|NN) . P(the|DT) . P(fence|NN) . P(.|.)

Questions?

- Where do tags come from?
  - Tag set
- How to get probability values i.e. P()?
  - Annotated corpora

After modeling of the problem, emphasis should be on the corpus.

Computing P() values

Lets suppose annotated corpus has following sentence

I	**have**	**a**	**brown**	**bag**	**.**
PRN	**VB**	**DT**	**JJ**	**NN**	**.**

$$P(NN \mid JJ) = \frac{Number\_of\_times\_JJ\_followed\_by\_NN}{Number\_of\_times\_JJ\_appeared}$$

$$P(Brown \mid JJ) = \frac{Number\_of\_times\_Brown\_tagged\_as\_JJ}{Number\_of\_times\_JJ\_appeared}$$

Next question?

- How to decode efficiently?
- E.g.
  - T: Tags
  - W: Words
  - Two special symbol: '^' and '.'

  Find out number of paths in the tree given word sequence.

Exponential *w.r.t.* number of words

Number of path = Number of leaves in the tree.

$$O(T^n)$$

How to avoid it?

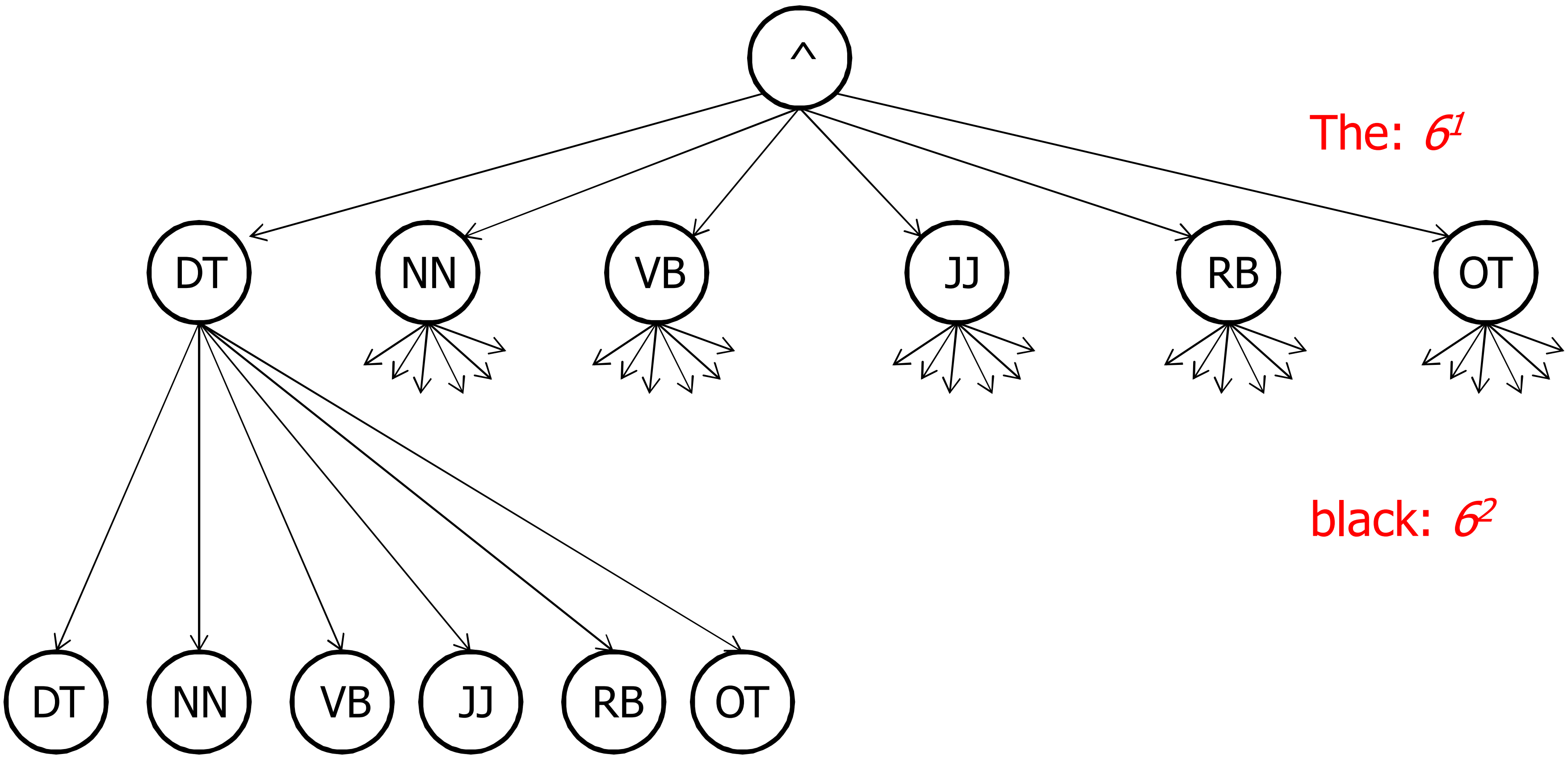
We do not need exponential work!

- Suppose our tags are
  - DT, NN, VB, JJ, RB and OT
- E.g.

^	The	black	dog	barks	.
^	DT NN VB JJ RB OT	DT NN VB JJ RB OT	DT NN VB JJ RB OT	DT NN VB JJ RB OT	.

Possible tags

So, 6^4 possible path



The: 6^1

black: 6^2

dog: 6^3

barks: 6^4

.: 6^4

Total 6^4 paths

- Now consider the paths that end in NN after seeing input "The black"

^	The	black	
^	DT	NN	***P*(T).*P*(W|T)** = *P*(DT|^) . *P*(NN|DT) . *P*(The|DT) . *P*(Black|NN)
^	NN	NN	***P*(T).*P*(W|T)** = *P*(NN|^) . *P*(NN|NN) . *P*(The|NN) . *P*(Black|NN)
^	VB	NN	***P*(T).*P*(W|T)** = *P*(VB|^) . *P*(NN|VB) . *P*(The|VB) . *P*(Black|NN)
^	JJ	NN	***P*(T).*P*(W|T)** = *P*(JJ|^) . *P*(NN|JJ) . *P*(The|JJ) . *P*(Black|NN)
^	RB	NN	***P*(T).*P*(W|T)** = *P*(RB|^) . *P*(NN|RB) . *P*(The|RB) . *P*(Black|NN)
^	OT	NN	***P*(T).*P*(W|T)** = *P*(OT|^) . *P*(NN|OT) . *P*(The|OT) . *P*(Black|NN)

$$Complexity = W_n * T$$

For each tag, only path with ***highest probability*** value are ***retained***, others are simply ***discarded***.

MT v/s POS tagging!

- Similarity
  - POS
    - Every word in a sentence has one corresponding tag.
  - MT
    - Every word in a sentence has one (or more) corresponding translated word.
- Difference
  - Order: Order of translated word may change.
  - Fertility: One word corresponds to many. Many to one also possible.

Complexity

- POS and HMM
  - Linear time complexity
- MT and Bean search
  - Exponential time complexity
  - Permutation of words produces exponential searc space

  - However, for related languages, MT is like POS tagging

Properties of related languages

1. Order preserving

2. Fertility ~ 1

3. Morphology preserving

Hindi	*Jaaunga*
Bengali	*Jaabo*
English	*Will go*

Hindi & Bengali ↑

Hindi & English ↓

Properties of related languages

4. Syncretism: Suffix features should be similarly loaded

Hindi	Main *jaaunga*	Hum *jaayenge*
Bengali	Ami *jaabo*	Aamra *jaabo*

Hindi & Bengali ↓

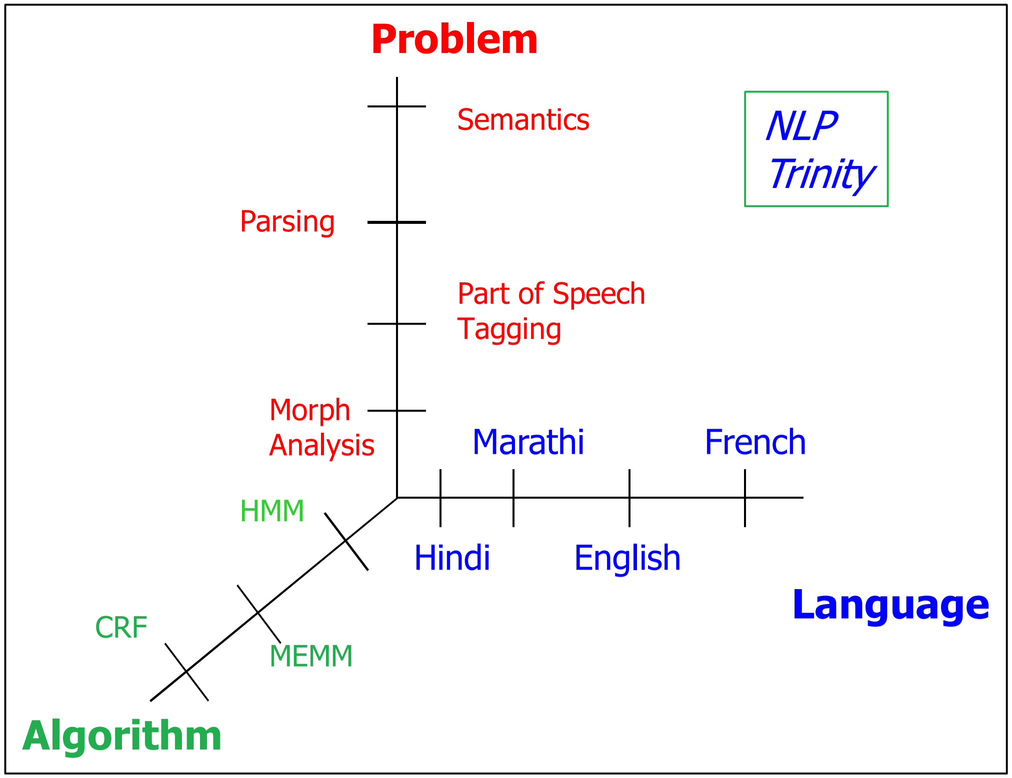
5. Idiomaticity: Literal translation should be high

Hindi	*Aap Kaise Ho?*
Bengali	*Aapni Kemon Achen?*
English	*How do you do?*

Hindi & Bengali ↑

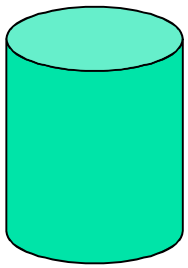
Hindi & English ↓

HMM

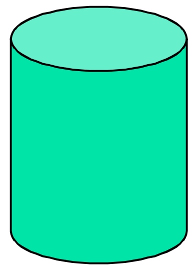

Problem
Semantics
NLP
Trinity
Parsing
Part of Speech
Tagging
Morph
Analysis
Marathi
French
HMM
Hindi
English
CRF
MEMM
Language
Algorithm

A Motivating Example

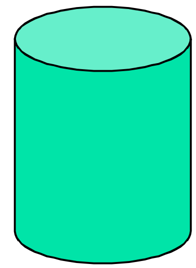
Colored Ball choosing



Urn 1
# of Red = 30
# of Green = 50
# of Blue = 20



Urn 2
# of Red = 10
# of Green = 40
# of Blue = 50



Urn 3
# of Red =60
# of Green =10
# of Blue = 30

Example (contd.)

Given :

	U_1	U_2	U_3
U_1	0.1	0.4	0.5
U_2	0.6	0.2	0.2
U_3	0.3	0.4	0.3

Transition probability table

and

	R	G	B
U_1	0.3	0.5	0.2
U_2	0.1	0.4	0.5
U_3	0.6	0.1	0.3

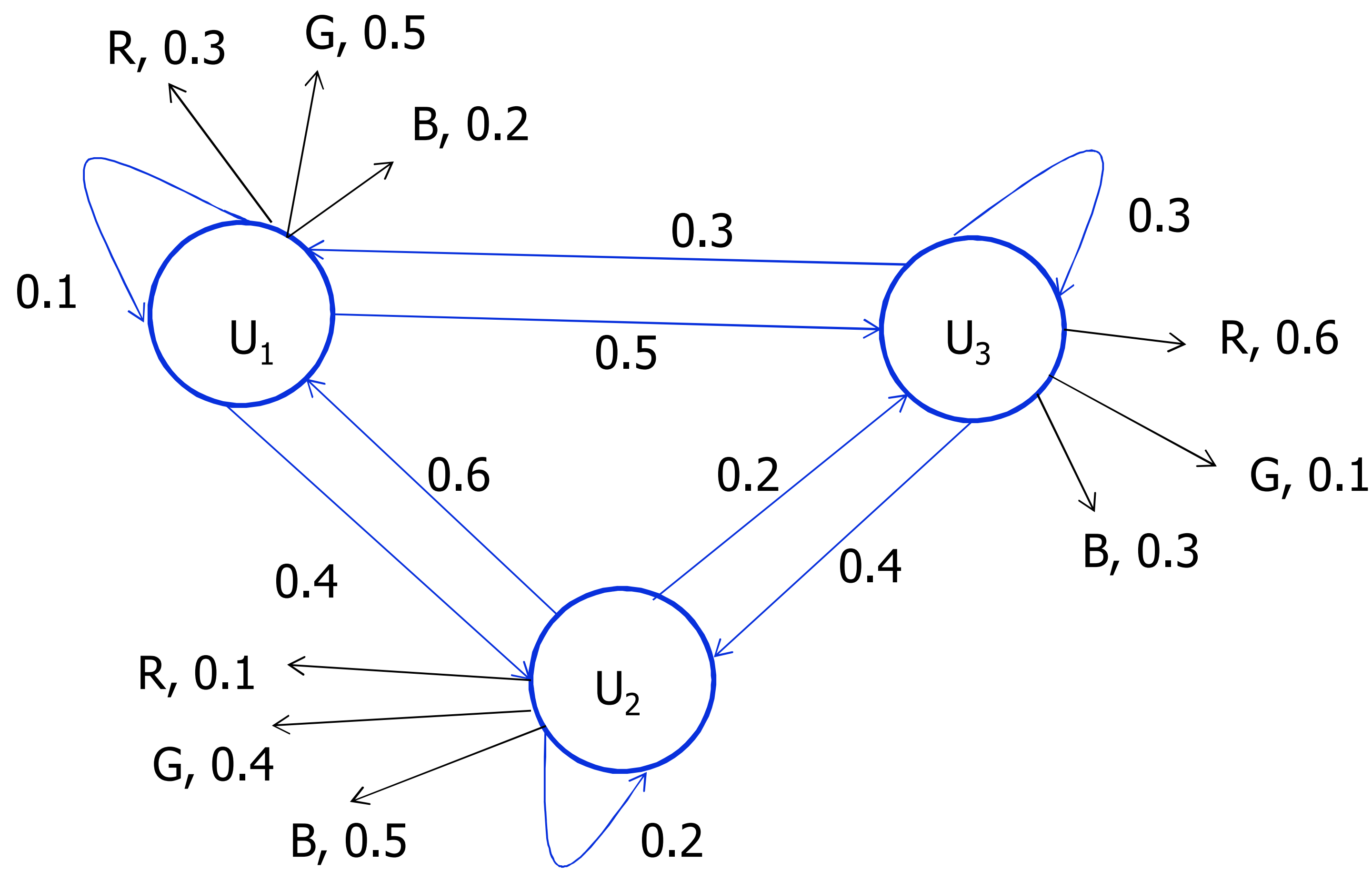
Emission probability table

Observation : RRGGBRGR

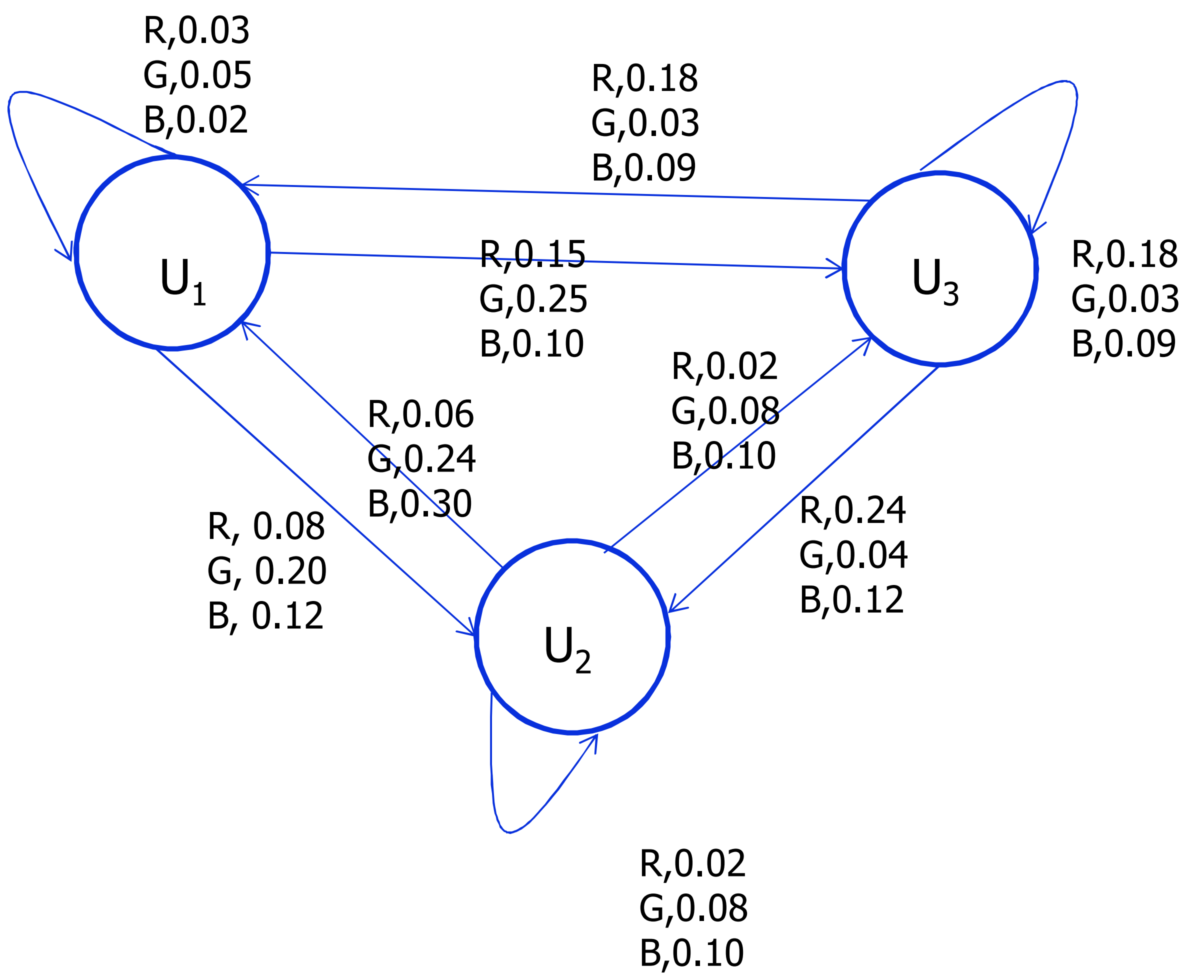
State Sequence : ??

Not so Easily Computable.

Diagrammatic representation (1/2)



Diagrammatic representation (2/2)



Classic problems with respect to HMM

1. Given the observation sequence, find the possible state sequences- Viterbi
2. Given the observation sequence, find its probability- forward/backward algorithm
3. Given the observation sequence find the HMM prameters.- Baum-Welch algorithm

Illustration of Viterbi

- The “start” and “end” are important in a sequence.
- Subtrees get eliminated due to the Markov Assumption.

<u>POS Tagset</u>

- N(noun), V(verb), O(other)   [simplified]
- ^ (start), . (end)   [start & end states]

Illustration of Viterbi

Lexicon

people: N, V

laugh: N, V

.

.

.

Corpora for Training

^ $w_{11}\_t_{11}$ $w_{12}\_t_{12}$ $w_{13}\_t_{13}$ ....$w_{1k\_1}\_t_{1k\_1}$.

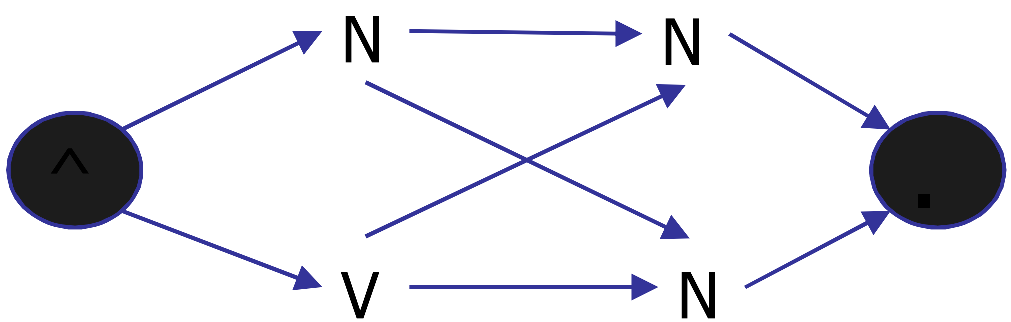
^ $w_{21}\_t_{21}$ $w_{22}\_t_{22}$ $w_{23}\_t_{23}$ ....$w_{2k\_2}\_t_{2k\_2}$.

.

.

^ $w_{n1}\_t_{n1}$ $w_{n2}\_t_{n2}$ $w_{n3}\_t_{n3}$ ....$w_{nk\_n}\_t_{nk\_n}$.

Inference



Partial sequence graph

	^	N	V	O	.
^	0	0.6	0.2	0.2	0
N	0	0.1	0.4	0.3	0.2
V	0	0.3	0.1	0.3	0.3
O	0	0.3	0.2	0.3	0.2
.	1	0	0	0	0

This transition table will change from language to language due to language divergences.

Lexical Probability Table

	∈	people	laugh	…	…
^	1	0	0	…	0
N	0	$1x10^{-3}$	$1x10^{-5}$	…	…
V	0	$1x10^{-6}$	$1x10^{-3}$	…	…
O	0	0	0	…	…
.	1	0	0	0	0

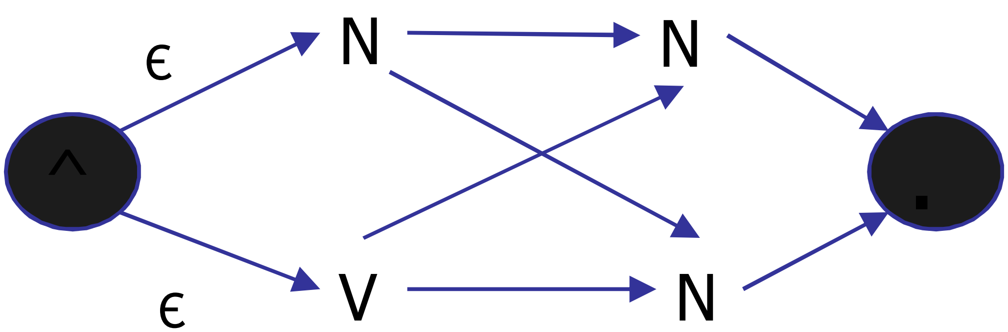
Size of this table = # pos tags in tagset X vocabulary size

vocabulary size = # unique words in corpus

Inference

New Sentence:

^ people laugh .



p(^ N N . | ^ people laugh .)

$= (0.6 \times 0.1) \times (0.1 \times 1 \times 10^{-3}) \times (0.2 \times 1 \times 10^{-5})$

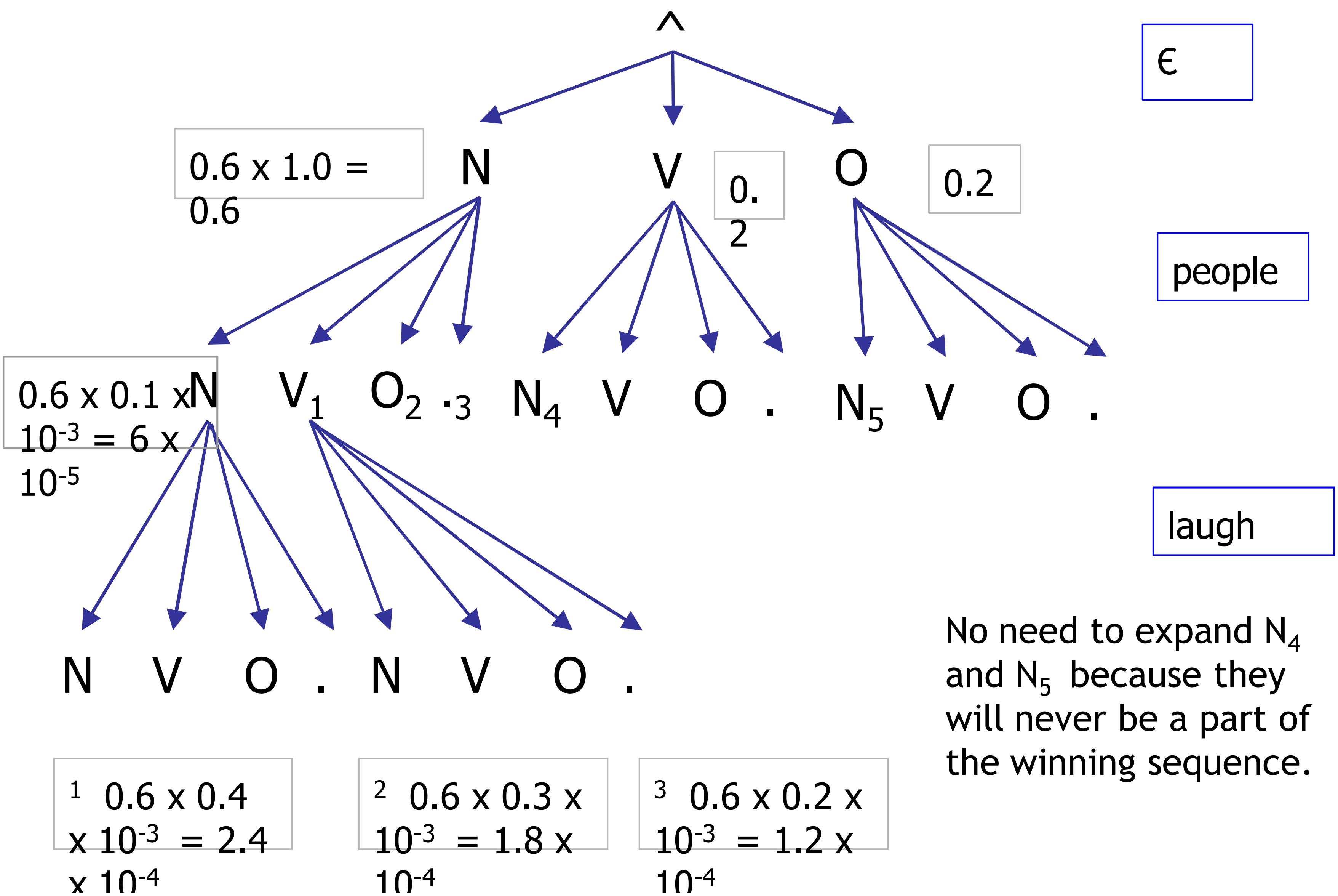
Computational Complexity

- If we have to get the probability of each sequence and then find maximum among them, we would run into exponential number of computations.
- If |s| = #states (tags + ^ + .)

  and |o| = length of sentence (words + ^ + .)

  Then, #sequences = $s^{|o|-2}$
- But, a large number of partial computations can be reused using Dynamic Programming.

Dynamic Programming



No need to expand N_4 and N_5 because they will never be a part of the winning sequence.

Computational Complexity

- Retain only those N / V / O nodes which ends in the highest sequence probability.
- Now, complexity reduces from $|s|^{|o|}$ to $|s|.|o|$
- Here, we followed the Markov assumption of order 1.

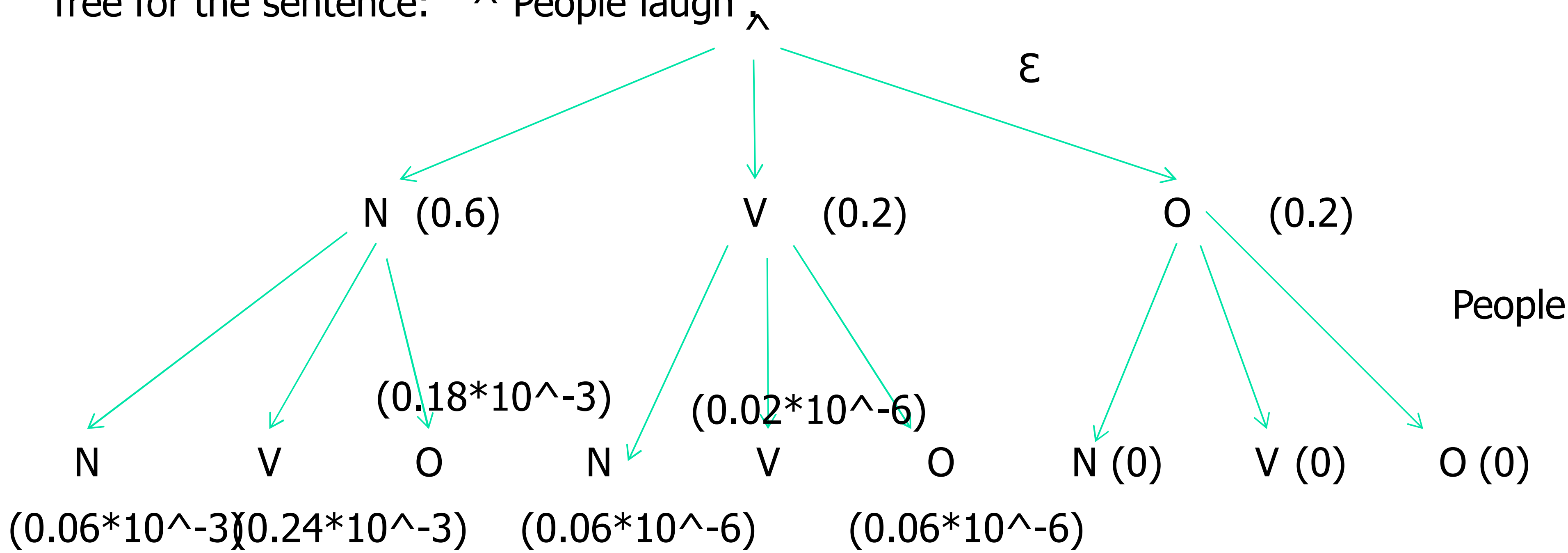
Points to ponder wrt HMM and Viterbi

Viterbi Algorithm

- Start with the start state.
- Keep advancing sequences that are "maximum" amongst all those ending in the same state

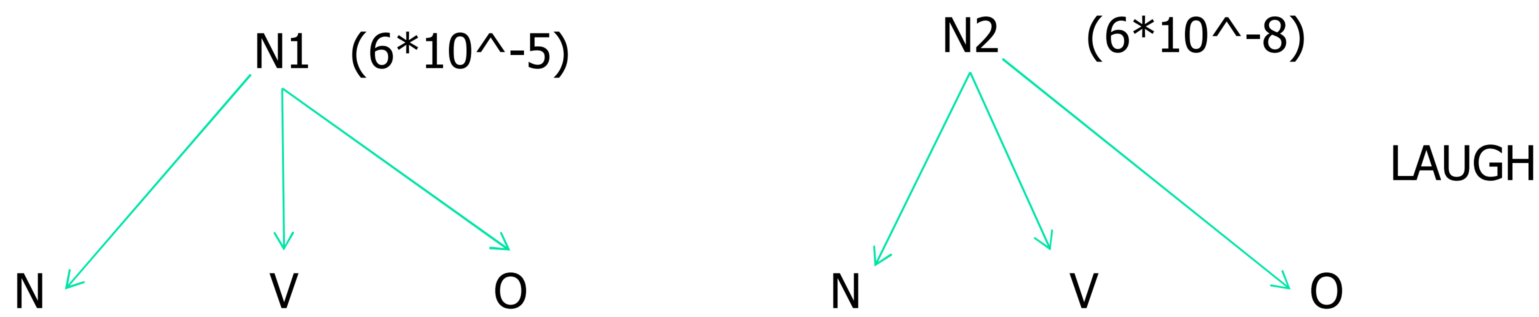
Viterbi Algorithm

Tree for the sentence: "^ People laugh ."



Claim: We do not need to draw all the subtrees in the algorithm

Viterbi phenomenon (Markov process)



Next step all the probabilities will be multiplied by identical probability (lexical and transition). So children of N2 will have probability less than the children of N1.

What does P(A|B) mean?

- $P(A|B)= P(B|A)$

  If $P(A)=P(B)$

- P(A|B) means??
  - Causality?? B causes A??
  - Sequentiality?? A follows B?

Back to the Urn Example

- Here :
  - S = {U1, U2, U3}
  - V = { R,G,B}
- For observation:
  - O ={o_1… o_n}
- And State sequence
  - Q ={q_1… q_n}
- Π is $\pi_i = P(q_1 = U_i)$

A =

	U_1	U_2	U_3
U_1	0.1	0.4	0.5
U_2	0.6	0.2	0.2
U_3	0.3	0.4	0.3

B=

	R	G	B
U_1	0.3	0.5	0.2
U_2	0.1	0.4	0.5
U_3	0.6	0.1	0.3

Observations and states

	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8
OBS:	R	R	G	G	B	R	G	R
State:	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8

$S_i = U_1/U_2/U_3$; A particular state

S: State sequence

O: Observation sequence

S* = "best" possible state (urn) sequence

Goal: Maximize P(S*|O) by choosing "best" S

Goal

- Maximize P(S|O) where S is the State Sequence and O is the Observation Sequence

$$S^* = \arg\max_S (P(S \mid O))$$

False Start

	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8
OBS:	R	R	G	G	B	R	G	R
State:	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8

$$P(S|O) = P(S_{1-8}|O_{1-8})$$

$$P(S|O) = P(S_1|O).P(S_2|S_1,O).P(S_3|S_{1-2},O)...P(S_8|S_{1-7},O)$$

By Markov Assumption (a state depends only on the previous state)

$$P(S|O) = P(S_1|O).P(S_2|S_1,O).P(S_3|S_2,O)...P(S_8|S_7,O)$$

Baye's Theorem

$$P(A \mid B) = P(A).P(B \mid A) / P(B)$$

P(A) -: Prior
P(B|A) -: Likelihood

$$\mathrm{argmax}_S\, P(S \mid O) = \mathrm{argmax}_S\, P(S).P(O \mid S)$$

State Transitions Probability

$$P(S) = P(S_{1-8})$$

$$P(S) = P(S_1).P(S_2 | S_1).P(S_3 | S_{1-2}).P(S_4 | S_{1-3})..P(S_8 | S_{1-7})$$

By Markov Assumption (k=1)

$$P(S) = P(S_1).P(S_2 | S_1).P(S_3 | S_2).P(S_4 | S_3)..P(S_8 | S_7)$$

Observation Sequence probability

$$P(O|S) = P(O_1|S_{1-8}).P(O_2|O_1, S_{1-8}).P(O_3|O_{1-2}, S_{1-8})..P(O_8|O_{1-7}, S_{1-8})$$

Assumption that ball drawn depends only on the Urn chosen

$$P(O|S) = P(O_1|S_1).P(O_2|S_2).P(O_3|S_3)...P(O_8|S_8)$$

$$P(S|O) = P(S).P(O|S)$$

$$P(S|O) = P(S_1).P(S_2|S_1).P(S_3|S_2).P(S_4|S_3)...P(S_8|S_7).$$
$$P(O_1|S_1).P(O_2|S_2).P(O_3|S_3)...P(O_8|S_8)$$

Grouping terms

	O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8	
Obs:	ε	R	R	G	G	B	R	G	R	
State:	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9

$P(S).P(O|S)$

$= [P(O_0|S_0).P(S_1|S_0)].$
$[P(O_1|S_1). \quad P(S_2|S_1)].$
$[P(O_2|S_2). \quad P(S_3|S_2)].$
$[P(O_3|S_3).P(S_4|S_3)].$
$[P(O_4|S_4).P(S_5|S_4)].$
$[P(O_5|S_5).P(S_6|S_5)].$
$[P(O_6|S_6).P(S_7|S_6)].$
$[P(O_7|S_7).P(S_8|S_7)].$
$[P(O_8|S_8).P(S_9|S_8)].$

We introduce the states S_0 and S_9 as initial and final states respectively.

After S_8 the next state is S_9 with probability 1, i.e., $P(S_9|S_8)=1$

O_0 is ε-transition

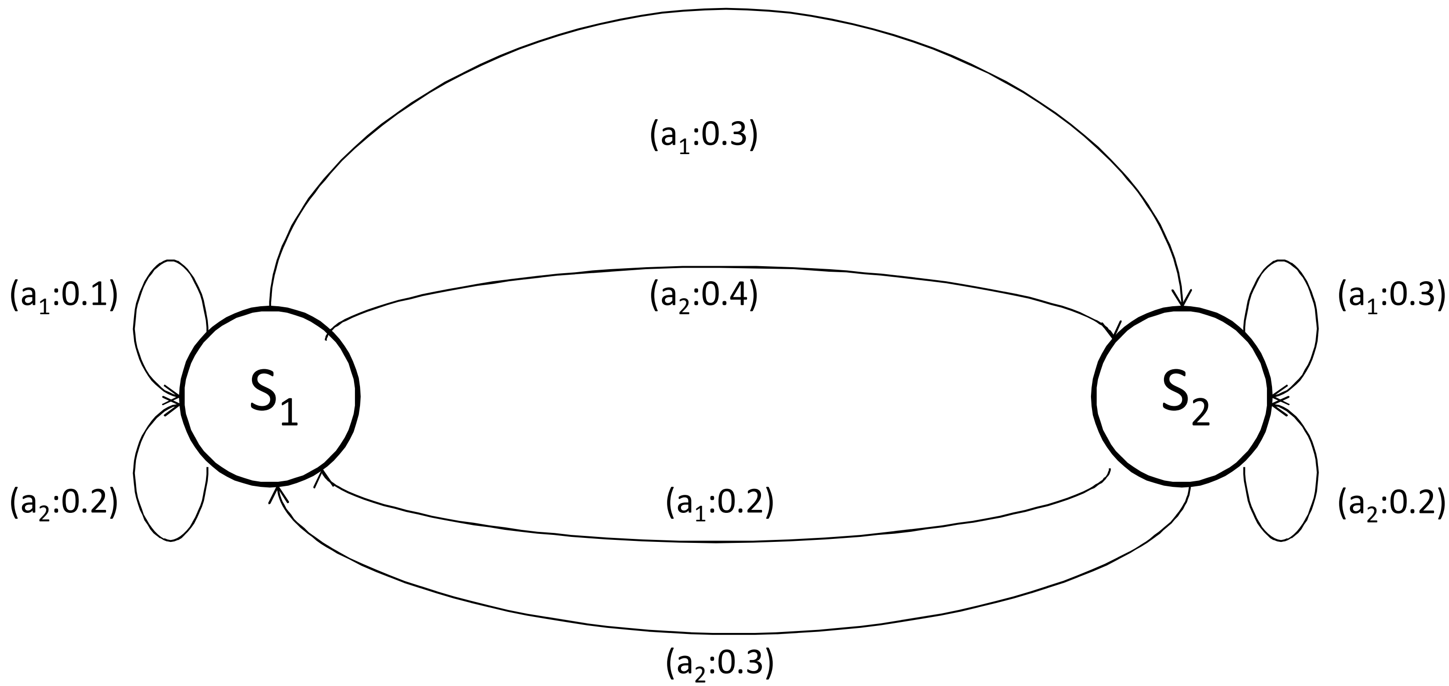
Introducing useful notation

	O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8	
Obs:	ε	R	R	G	G	B	R	G	R	
State:	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9

$$S_0 \xrightarrow{\varepsilon} S_1 \xrightarrow{R} S_2 \xrightarrow{R} S_3 \xrightarrow{G} S_4 \xrightarrow{G} S_5 \xrightarrow{B} S_6 \xrightarrow{R} S_7 \xrightarrow{G} S_8 \xrightarrow{R} S_9$$

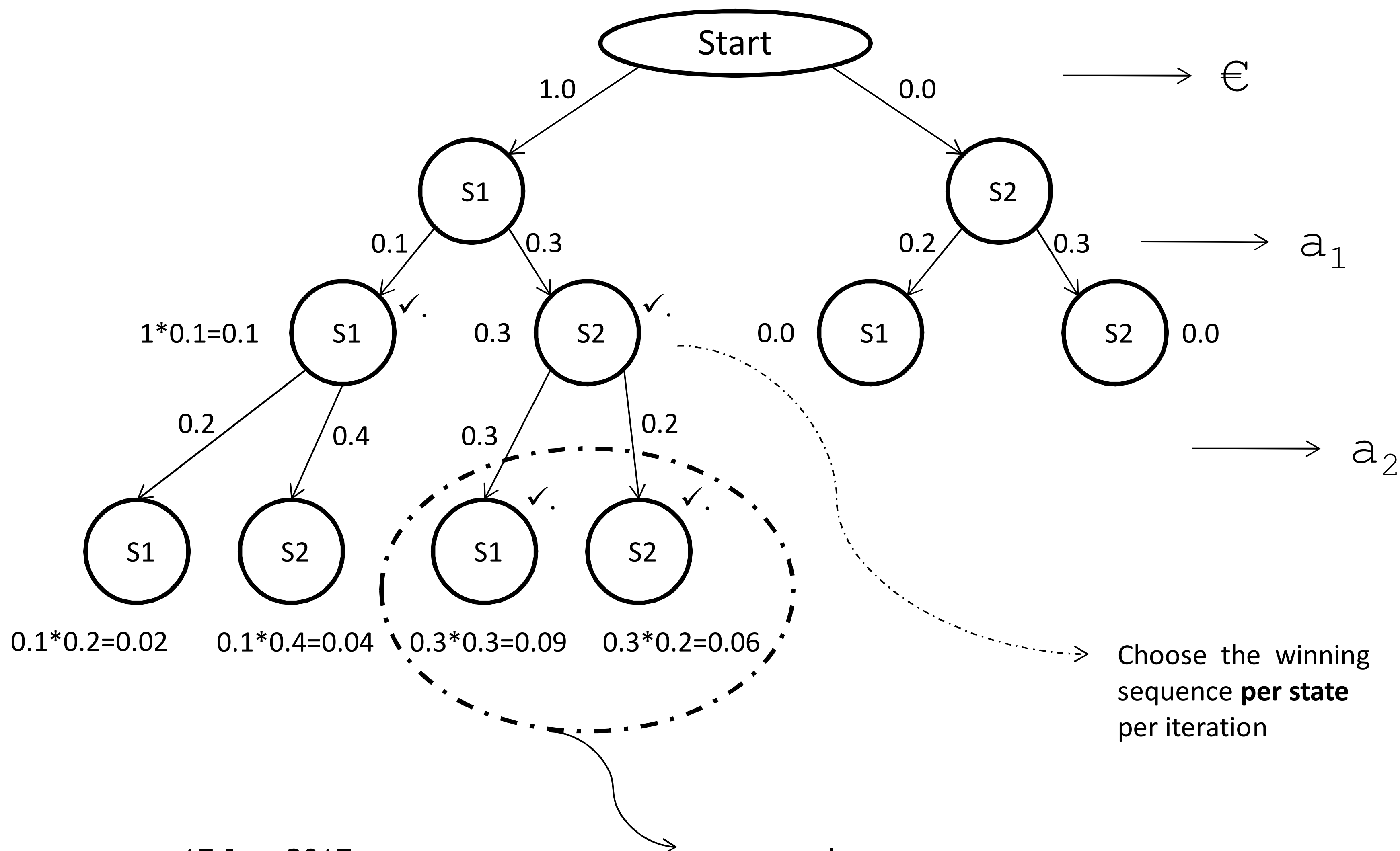
$$P(O_k|S_k).P(S_{k+1}|S_k)=P(S_k \xrightarrow{O_k} S_{k+1})$$

Probabilistic FSM

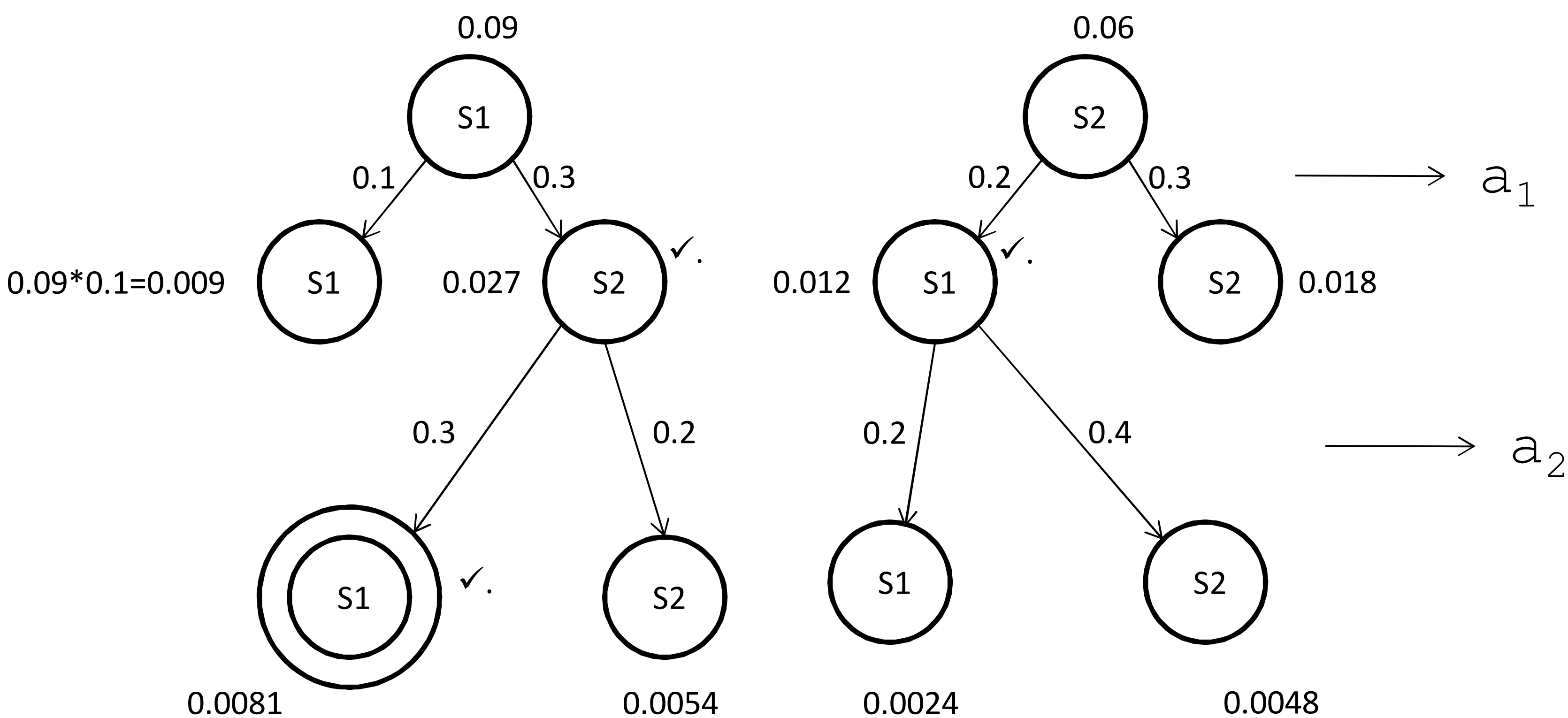


The question here is:
“what is the most likely state sequence given the output sequence seen”

Developing the tree



Tree structure contd…



The problem being addressed by this tree is $S^* = \arg\max_{s} P(S \mid a_1 - a_2 - a_1 - a_2, \mu)$

a1-a2-a1-a2 is the output sequence and μ the model or the machine

Path found:
(working backward)

$$S_1 \xrightarrow{a_1} S_2 \xrightarrow{a_2} S_1 \xrightarrow{a_1} S_2 \xrightarrow{a_2} S_1$$

Problem statement: Find the best possible sequence

$$S^* = \arg\max_{S} P(S \mid O, \mu)$$

$$where,\ S \rightarrow \text{State Seq},\ O \rightarrow \text{Output Seq},\ \mu \rightarrow \text{Model or Machine}$$

$$\text{Model or Machine} = \{S_0, S, A, T\}$$

Start symbol

State collection

Alphabet set

Transitions

T is defined as $P(S_i \xrightarrow{a_k} S_j) \quad \forall_{i,\ j,\ k}$

Tabular representation of the tree

Latest symbol observed / Ending state	€	a_1	a_2	a_1	a_2
S_1	1.0	(1.0*0.1,0.0*0.2)=(**0.1**,0.0)	(0.02, **0.09**)	(0.009, **0.012**)	(0.0024, **0.0081**)
S_2	0.0	(1.0*0.3,0.0*0.3)=(**0.3**,0.0)	(0.04,**0.06**)	(**0.027**,0.018)	(0.0048,0.0054)

Note: Every cell records the winning probability ending in that state

Final winner

The bold faced values in each cell shows the sequence probability ending in that state. Going backward from final winner sequence which ends in state S_2 (indicated By the 2nd tuple), we recover the sequence.

Algorithm

(following James Alan, Natural Language Understanding (2nd edition), Benjamin Cummins (pub.), 1995

Given:

1. The HMM, which means:
   a. Start State: S_1
   b. Alphabet: $A = \{a_1, a_2, \ldots a_p\}$
   c. Set of States: $S = \{S_1, S_2, \ldots S_n\}$
   d. Transition probability $P(S_i \xrightarrow{a_k} S_j) \quad \forall i, j, k$
      which is equal to $P(S_j, a_k \mid S_i)$
2. The output string $a_1 a_2 \ldots a_T$

To find:

The most likely sequence of states $C_1 C_2 \ldots C_T$ which produces the given output sequence, *i.e.*, $C_1 C_2 \ldots C_T = \arg\max_C [P(C \mid a_1, a_2, \ldots a_T, \mu]$

Algorithm contd…

Data Structure:

1. A N*T array called SEQSCORE to maintain the winner sequence always (N=#states, T=length of o/p sequence)
2. Another N*T array called BACKPTR to recover the path.

Three distinct steps in the Viterbi implementation

1. Initialization
2. Iteration
3. Sequence Identification

1. Initialization

SEQSCORE(1,1)=1.0
BACKPTR(1,1)=0
For(i=2 to N) do
    SEQSCORE(i,1)=0.0
[expressing the fact that first state is S_1]

2. Iteration

For(t=2 to T) do
    For(i=1 to N) do
        SEQSCORE(i,t) = $Max_{(j=1,N)}$

$$[SEQSCORE(j,(t-1)) * P(Sj \xrightarrow{a_k} Si)]$$

BACKPTR(I,t) = index *j* that gives the MAX above

3. Seq. Identification

C(T) = i that maximizes SEQSCORE(i,T)

For i from (T-1) to 1 do

C(i) = BACKPTR[C(i+1),(i+1)]

Optimizations possible:

1. BACKPTR can be 1*T
2. SEQSCORE can be T*2

Homework:- Compare this with A*, Beam Search [Homework]

Reason for this comparison:

Both of them work for finding and recovering sequence

Reading List

- TnT (http://www.aclweb.org/anthology-new/A/A00/A00-1031.pdf)
- Brill Tagger (http://delivery.acm.org/10.1145/1080000/1075553/p112-brill.pdf?ip=182.19.16.71&acc=OPEN&CFID=129797466&CFTOKEN=72601926&__acm__=1342975719_082233e0ca9b5d1d67a9997c03a649d1)
- Hindi POS Tagger built by IIT Bombay (http://www.cse.iitb.ac.in/pb/papers/ACL-2006-Hindi-POS-Tagging.pdf)
- Projection (http://www.dipanjandas.com/files/posInduction.pdf)