CS4025: Syntax and Parts of Speech

- Why syntax?
- Grammars
- Parts of Speech
- Part of speech tagging and the Viterbi algorithm

See J&M chapter 8 in 1st ed, 5.1 to 5.5 in 2nd, Mellish and Ritchie notes

Why syntax?

- Natural language sentences have structure beyond simple word adjacency and this is relevant for meaning:
  - **James Thomason**, my wife's oldest friend, kindly **donated** the flowers.
    - *subject(donated, Thomason)*
    - *object(donated, flowers)*
- Different possible meanings can often be explained in terms of different structures
  - The explosives were found by (a security man in a plastic bag)
  - The explosives were (found by a security man) in a plastic bag

Why syntax?

- Knowledge of legal structures narrows down the alternatives for possible meanings
  - He saw the rope under the boxes, which was just what he needed (Relative Clause Attachment)
  - Ross looked at him in the mirror (Pronoun Binding)
- Any general account of how to extract meaning from a sentence (which can handle previously unseen sentences) must have some kind of structure to refer to

Grammar: Definition

The surface structure (syntax) of sentences is usually described by some kind of *grammar*.

- A grammar defines syntactically legal sentences.
  - John ate an apple (syn legal)
  - John ate apple (not syn legal)
  - John ate a building (syn legal)
- More importantly, a grammar provides a description of the structure of a legal sentence (whether or not it makes sense)

A very simple grammar

- S = NP VP
- VP = Verb NP
- NP = Det Noun
- NP = Name
- Det: *a* , *the*
- Noun: *dog* , *cat*
- Name: *Fido* , *Misty*
- Verb: *chases* , *sees*

Grammar

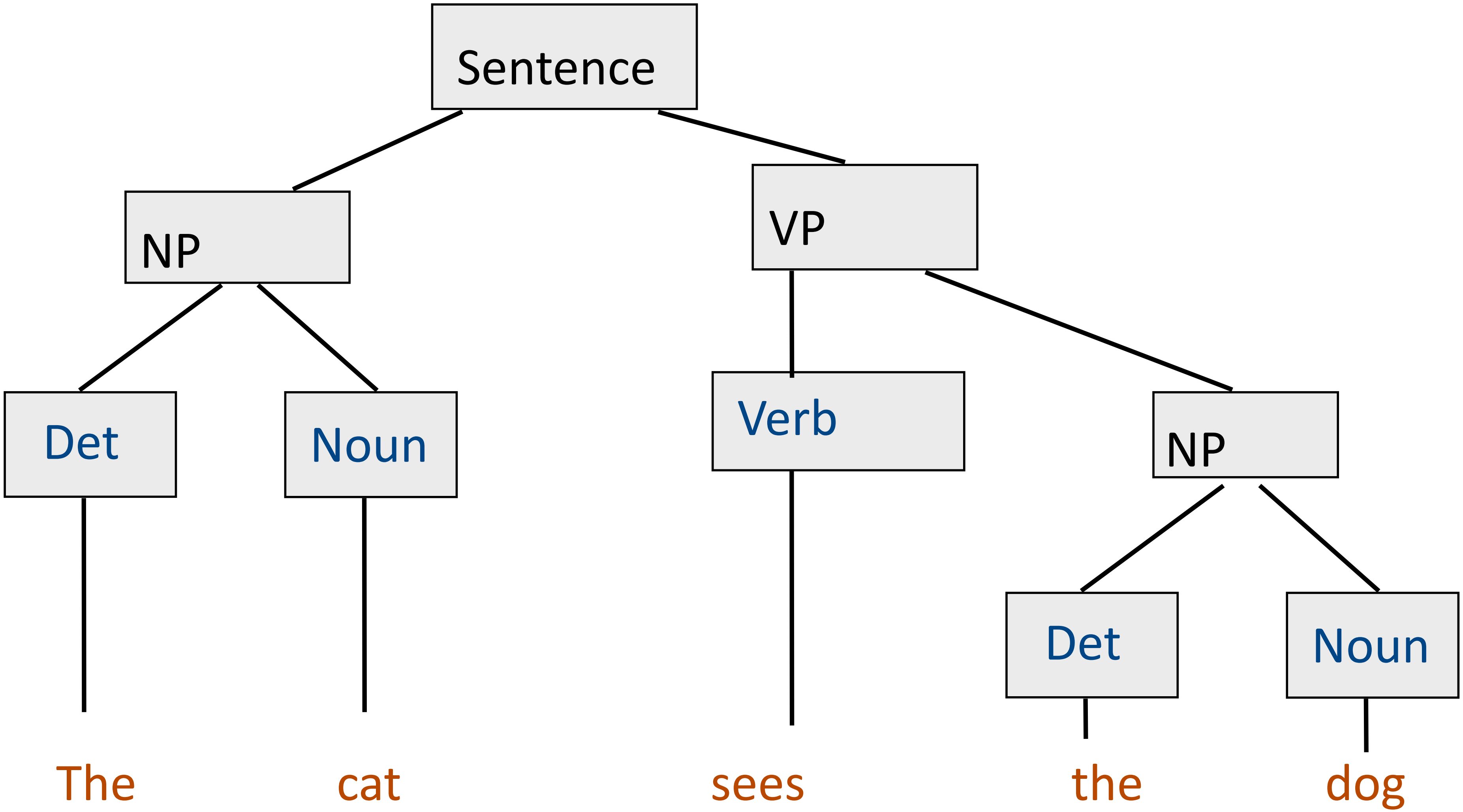
Terminals

Pre-Terminals

Fido chases the cat

A cat sees Misty

Ex: The cat sees the dog



Parts of Speech

- The preterminals (lexical categories) of a natural language grammar are called *parts of speech*
- Main parts of speech for English are:
  - Noun
  - Verb
  - Adjective
  - Adverbs
  - Prepositions
  - Determiners

Part of Speech Tagging

- POS tagging is the task of labelling every word with its part of speech, from a specified *tagset*
- It assumes a dictionary that specifies for each word which *tags* it could have
- POS tagging is a very simple type of syntactic analysis, and is useful, for instance, for:
  - Text to speech systems
  - Simple information extraction systems

How ambiguous are words?

- Words in the Brown corpus:

Unambiguous (1 tag)	**35,340**	
Ambiguous (2–7 tags)	**4,100**	
2 tags	3,760	
3 tags	264	
4 tags	61	
5 tags	12	
6 tags	2	
7 tags	1	("still")

- Unfortunately, often the most common words are ambiguous
- play (v) → perform a play (n)
- catch (v)→ Take a catch (n)
- hit (v)→ make a hit (n)

Examples of light verb constructions

Statistical POS Tagging

- We adapt the noisy channel model for spelling correction:

For POS Tagging, source is the sequence of tags and what is observed is the sequence of words

The model precisely

I	can	can	the	can	(noisy words …)
P	MD	VB	DT	NN	(guess at original tags)
P	VB	NN	DT	VB	(guess at original tags)

… (many other possible sequences)

- Assume we have received the words W
- We seek to choose the sequence of tags T which maximises P(T|W)
- P(T|W) is the probability that T was intended, given that W was received
- By Bayes' rule, this is equivalent to

$$P(T|W) = \frac{P(W|T) \cdot P(T)}{P(W)}$$

The model precisely

I	can	can	the	can
P	MD	VB	DT	NN
P	VB	NN	DT	VB

- We know W, and need to find the value of T that maximises:

$$P(T \mid W) = P(W|T) \cdot P(T)/P(W)$$

- Since P(W) is the same for all T, we just need to maximise:

$$P(W|T) \cdot P(T)$$

- For a sentence, we estimate P(W|T) as the product of the P(w_i|t_i) for each word/tag in the sentence.
- We can estimate P(T) using unigram, bigram or trigram models of tags.

The model precisely - 3

I	can	can	the	can
P	MD	VB	DT	NN
P	VB	NN	DT	VB

$$
\begin{aligned}
\hat{t}_1^n &= \operatorname*{argmax}_{t_1^n} P(t_1^n | w_1^n) \\
&= \operatorname*{argmax}_{t_1^n} \frac{P(w_1^n | t_1^n) P(t_1^n)}{P(w_1^n)} \quad \text{using Bayes' rule} \\
&= \operatorname*{argmax}_{t_1^n} P(w_1^n | t_1^n) P(t_1^n) \quad \text{denominator does not change}
\end{aligned}
$$

Conditional likelihood

Prior (does not depend on words in the sentence)

The model precisely - 3

I	can	can	the	can
P	MD	VB	DT	NN
P	VB	NN	DT	VB

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n)$$

$$= \underset{t_1^n}{\operatorname{argmax}} \frac{P(w_1^n | t_1^n) P(t_1^n)}{P(w_1^n)} \quad \text{using Bayes' rule}$$

$$= \underset{t_1^n}{\operatorname{argmax}} P(w_1^n | t_1^n) P(t_1^n) \quad \text{denominator does not change}$$

Emission Probability

Transition Probability

Estimating the Probabilities as bigrams

I	can	can	the	can
P	MD	VB	DT	NN
P	VB	NN	DT	VB

- For words $w_1, w_2, \dots w_n$ and tags $t_1, t_2, \dots t_n$, we calculate

$$P(W|T) = P(w_1|t_1) \cdot P(w_2|t_2) \cdot \cdots P(w_n|t_n)$$
$$P(T) = P(t_1|start) \cdot P(t_2|t_1) \cdot \cdots P(end|t_n)$$

- Emission Probabilities: $P(w_i|t_i)$ is estimated from a tagged corpus (remember n-gram lecture?):

$$\frac{Number\ of\ times\ w_i\ appears\ with\ t_i}{Number\ of\ times\ t_i\ appears}$$

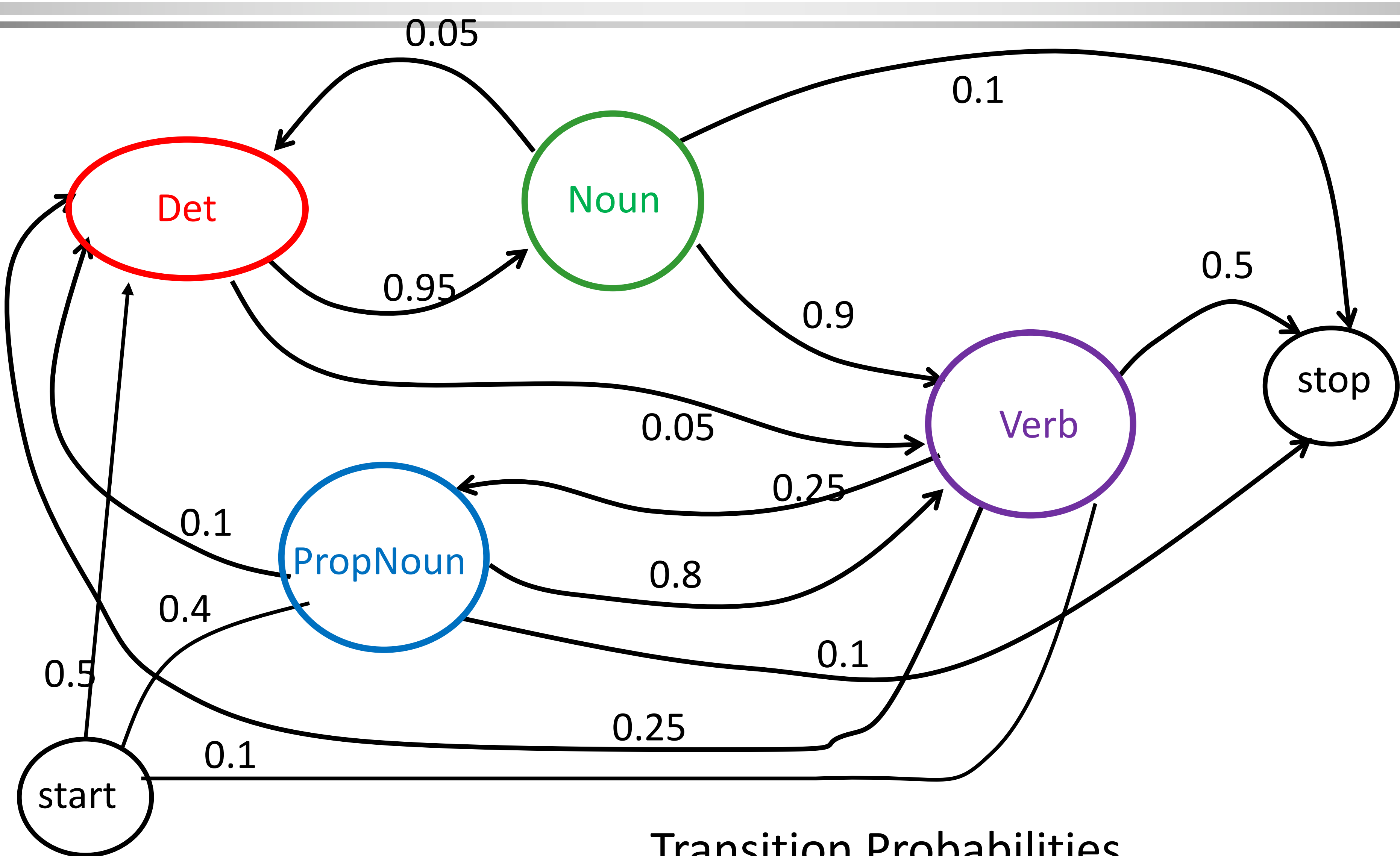
- Transition Probabilities: $P\left(t_i|t_j\right)$ is estimated from a tagged corpus:

$$\frac{Number\ of\ times\ w_i\ appears\ with\ t_j}{Number\ of\ times\ t_j\ appears}$$

Modelling Transitions: Markov Models

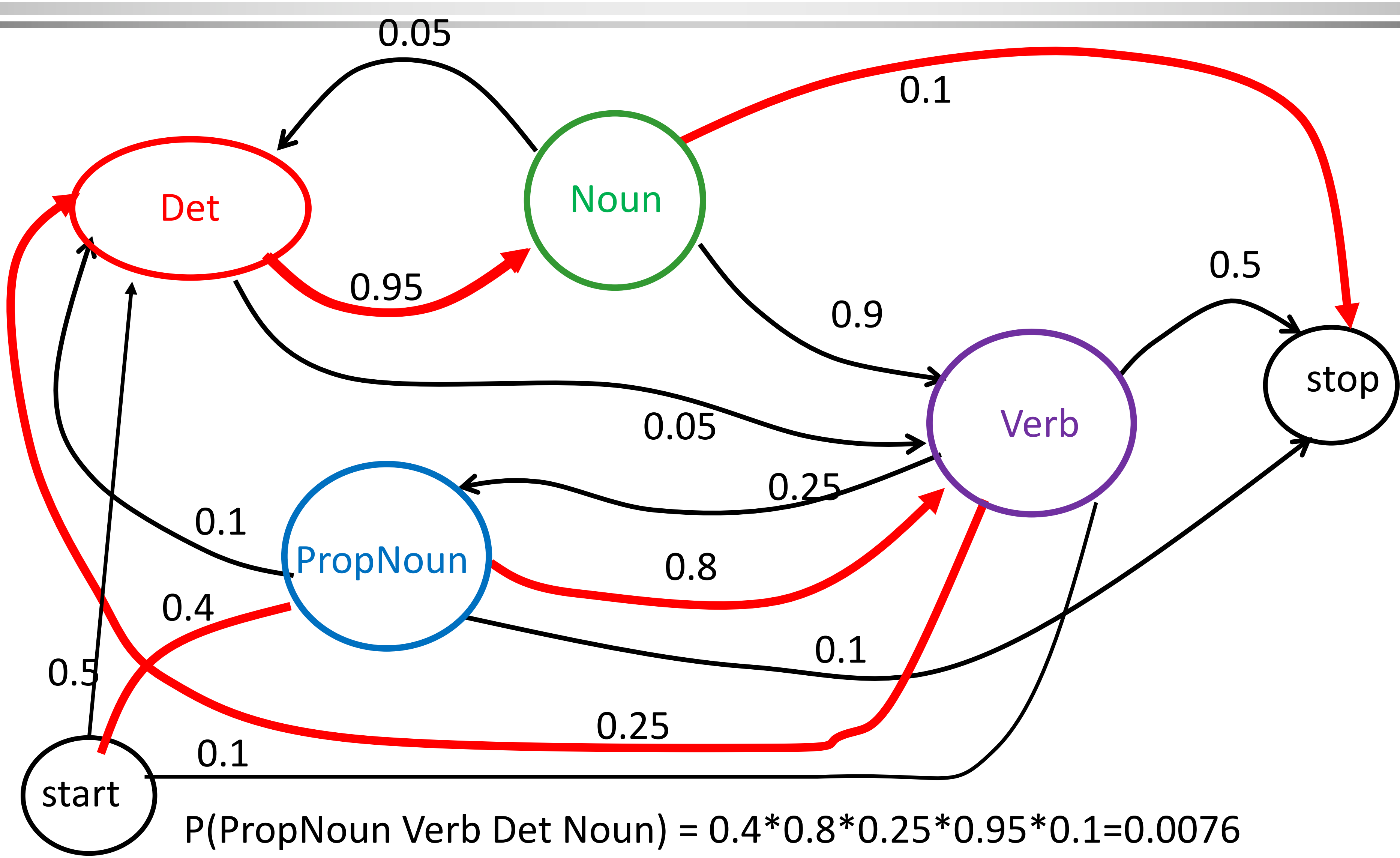
- A finite state machine with probabilistic state transitions.
- Makes Markov assumption that next state only depends on the current state and independent of previous history.

Sample Markov Model for POS (Credit Raymond J. Mooney)



Transition Probabilities

Sample Markov Model for POS (Credit Raymond J. Mooney)

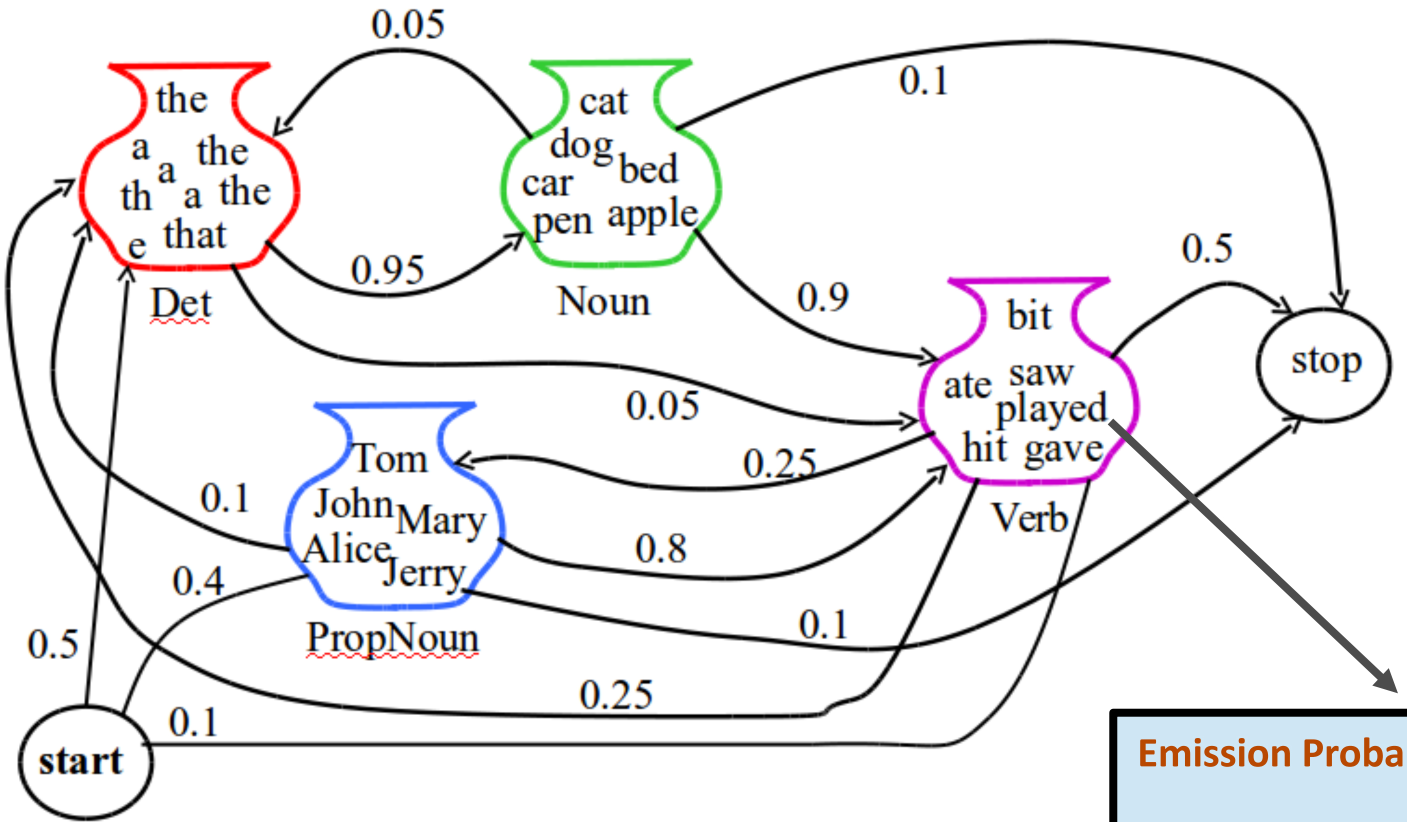


P(PropNoun Verb Det Noun) = 0.4*0.8*0.25*0.95*0.1=0.0076

Hidden Markov Model

- Probabilistic generative model for sequences.
- Assume an underlying set of ***hidden*** (unobserved) states in which the model can be (e.g. parts of speech).
- Assume probabilistic transitions between states over time (e.g. transition from POS to another POS as sequence is generated).
- Assume a ***probabilistic*** generation of tokens from states (e.g. words generated for each POS).

Sample HMM for POS (Credit Raymond J. Mooney)

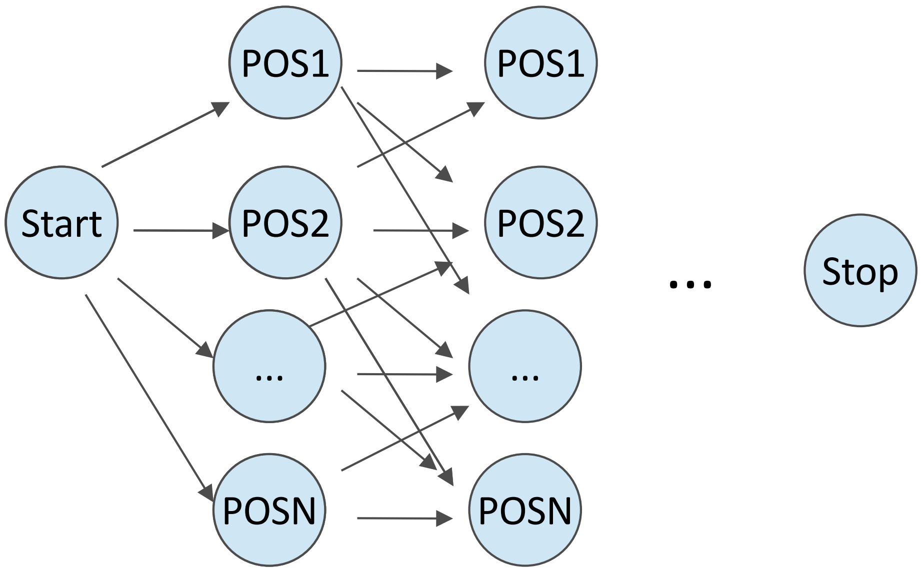


Emission Probabilities:

p(bit|verb) = 0.000001
p(played|verb) = 0.0003
p(ate|verb) = 0.00002
….

Optimisation of search

- With a HMM, we can calculate probability of any sequence
- But how to efficiently find the sequence with maximum probability?
  - Important question, because there are exponentially many paths through an HMM.



Visualising the process

<start>	fire	that	man	<end>
<start>	Noun	Adverb	Noun	<end>
	Verb	Pronoun	Verb	
		Determiner		
		Complementiser		

Optimisation

- We are looking for the best path through a sequence of tags that are possible for the words of the sentence
- For each path, take the product of the tag transition probabilities and the p(word|tag) emission probabilities
- In principle, we could compute the probability of each path, then choose the path with the highest probability
- This would require a lot of computation, particularly for long sentences

Optimisation

- Standard solution is a kind of dynamic programming: the Viterbi algorithm
  - A recursive approach that doesn't compute the same thing many times
  - Because we use a bigram model, the best path through t_i for w_j only needs to consider the best paths to tags for w_j-1 (and t_i and w_j themselves)

Example (N = Noun, etc.)

<start>	fire	that	man	<end>
<start>	Noun 0.1	Adverb 0.1	Noun 0.1	<end>
	Verb 0.1	Pronoun 0.1	Verb 0.2	
		Determiner 0.2		
		Complementiser 0.6		

Assume these bigram prob's:

P(N|start)=0.4, P(V|start)=0.2,

P(A|N)=0.2, P(P|N)=0, P(D|N)=0, P(C|N)=0.3,

P(A|V)=0.1, P(P|V)=0.2, P(D|V)=0.5, P(C|V)=0.1,

P(N|A)=0.1, P(V|A)=0.6,

P(N|P)=0, P(V|P)=0.4,

P(N|D)=0.8, P(V|D)=0,

P(N|C)=0.2, P(V|C)=0.3,

P(end|N)=0.7, P(end|V)=0.3

P(word|tag) given in table. Real values would generally be much smaller.

Example (cont) (f = “fire”)

<start>	fire	that	man	<end>
<start>	Noun P(N|s)*P(f|N) = 0.04	Adverb	Noun	<end>
	Verb P(V|s)*P(f|V) = 0.02	Pronoun	Verb	
		Determiner		
		Complementiser		

Example (cont) (t = “that”)

<start>	fire	that	man	<end>
<start>	Noun 0.04	Adverb Max(0.04*P(A|N)*P(t|A), 0.02*P(A|V)*P(t|A)) = 0.0008 (from N)	Noun	<end>
	Verb 0.02	Pronoun Max(0.04*P(P|N)*P(t|P), 0.02*P(P|V)*P(t|P)) = 0.0004 (from V)	Verb	
		Determiner = 0.002 (from V)		
		Complementiser = 0.0072 (from N)		

Example (cont) (m = “man”)

<start>	fire	that	man	<end>
<start>	Noun 0.04	Adverb 0.0008 (from N)	Noun Max(0.0008*P(N|A)*P(m|N), 0.0004*P(N|P)*P(m|N), 0.0002*P(N|D)*P(m|N), 0.0072*P(N|C)*P(m|N)) = 0.00016 (from D)	<end>
	Verb 0.02	Pronoun 0.0004 (from V)	Verb = 0.00022 (from C)	
		Determiner 0.002 (from V)		
		Complementiser 0.0072 (from N)		

Example (concl)

<s>	fire	that	man	<end>
<s>	Noun 0.04	Adverb 0.0008 (from N)	Noun 0.00016 (from D)	<end> Max(0.00016* $P(e \mid N)$, 0.00022* $P(e \mid V)$) = 0.00012 (from N)
	Verb 0.02	Pronoun 0.0004 (from V)	Verb = 0.00022 (from C)	
		Determiner 0.002 (from V)		
		Complementiser 0.0072 (from N)		

$P(end \mid N)=0.7$,
$P(end \mid V)=0.3$

Reading off the solution

<s>	fire	that	man	<end>
<s>	Noun 0.04	Adverb 0.0008 (from N)	Noun 0.00016 (from D)	<end> Max(0.00016*P(e|N), 0.00022*P(e|V)) = 0.00012 (from N)
	Verb 0.02	Pronoun 0.0004 (from V)	Verb = 0.00022 (from C)	
		Determiner 0.002 (from V)		
		Complementiser 0.0072 (from N)		

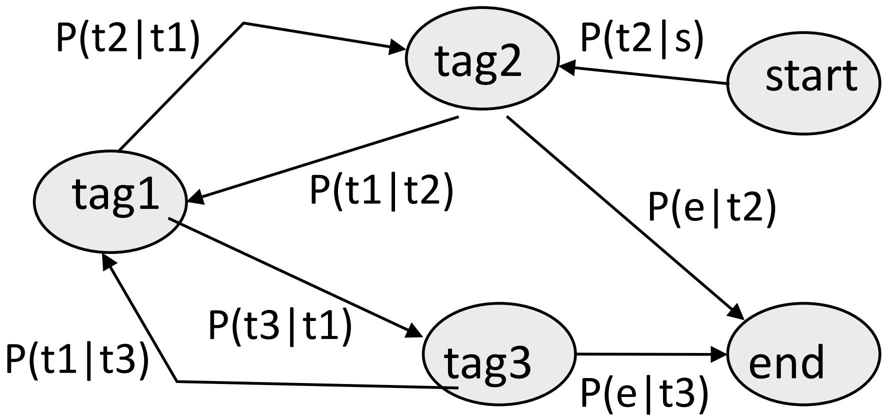
Algorithm

// Best(word,tag) records the probability of the
// best left-right path to a given word and tag

Best(<start>,<start>) = 1.0
For each word w_i in turn,
  For each possible tag t_j for w_i,
    Find the tag t_k for w_{i-1} which maximises:
      $Best(w_{i-1},t_k) * P(t_j|t_k) * P(w_i|t_j)$
    Assign this value to $Best(w_i, t_j)$

Relevance to HMMs

- Finding the best POS tagging of a sentence is the same as finding the best way through an HMM that produces the words of the sentence, such as:



Output probabilities:

tag1:	tag2:
w1: P(w1|t1)	w1: P(w1|t2)
w2: P(w2|t1)	w2: P(w2|t2)

- The above algorithm (for HMMs) is known as the *Viterbi* algorithm

Evaluation of POS tagging

- Most current tagging algorithms get 96% to 97% of tags correct
- Human annotators typically agree on about 96% to 97% of tags
- If one just selects the most likely tag for each word, one gets an accuracy of around 90% to 91%

Machine Learning: Sequence Modelling

- POS tagging is an example of classifying sequences.
- Many other problems use similar solutions
  - Speech recognition (Speech2Text)
  - Speech Generation (Text2Speech)
  - Named Entity Recognition
  - Gene Analysis
  - Activity Recognition from sensors
  - ….

Optional: more explanation on the Viterbi algorithm:

http://www.comp.leeds.ac.uk/roger/HiddenMarkovModels/html_dev/main.html