



# Natural Language Processing

## DSECL ZG565



**BITS** Pilani  
Pilani Campus

Dr. Vijayalakshmi Anand

BITS-Pilani



**Session 1  
Date – 2 Dec 2023  
Time – 1.40 to 3.40**

These slides are prepared by the instructor, with grateful acknowledgement of James Allen and many others who made their course materials freely available online.

# Agenda

- Course Objectives
- Course & Evaluation Plan
- Introduction to Natural Language Processing
- Application Areas
- Few Terminologies
- Frequently applied Data Preparation Process for NLP

# Objective of course

---

- To learn the fundamental concepts and techniques of natural language processing (NLP) including Language Models, Word Embedding, Part of speech Tagging, Parsing
- To learn computational properties of natural languages and the commonly used algorithms for processing linguistic information
- To introduce basic mathematical models and methods used in NLP applications to formulate computational solutions.
- To introduce research and development work in Natural language Processing

- M1 Natural Language Understanding and Generation
- M2 N-gram Language Modelling
- M3 Neural networks and Neural language Models
- M4 Part-of-Speech Tagging
- M5 Hidden Markov Models and MEMM
- M6 Topic Modelling
- M7 Vector semantics and Embedding
- M8 Grammars and Parsing
- M9 Statistical Constituency Parsing
- M10 Dependency Parsing
- M11 Encoder-Decoder Models, Attention and Contextual Embedding
- M12 Word sense disambiguation
- M13 Semantic web ontology and Knowledge Graph
- M14 Introduction to NLP Applications

# Text books and Reference books

---

T1	Jurafsky and Martin, SPEECH and LANGUAGE PROCESSING: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, McGraw Hill
T2	Manning and Schütze, Foundations of Statistical Natural Language Processing, MIT Press. Cambridge, MA

R1	Allen James, Natural Language Understanding
R2	Neural Machine Translation by Philipp Koehn
R3	Semantic Web Primer (Information Systems) By Antoniou, Grigoris; Van Harmelen, Frank

# Tentative Evaluation Plan

Name	Weight
Quiz (best 2 out of 3)	10%
Assignment 1 and 2	20%
Mid-term Exam	30%
End Semester Exam	40%

Students are requested to check the canvas announcements for the details

# What is Natural Language Processing?

---

- Natural Language Processing
  - Process information contained in natural language text.
  - Also known as Computational Linguistics (CL), Human Language Technology (HLT), Natural Language Engineering (NLE)

# What is it..

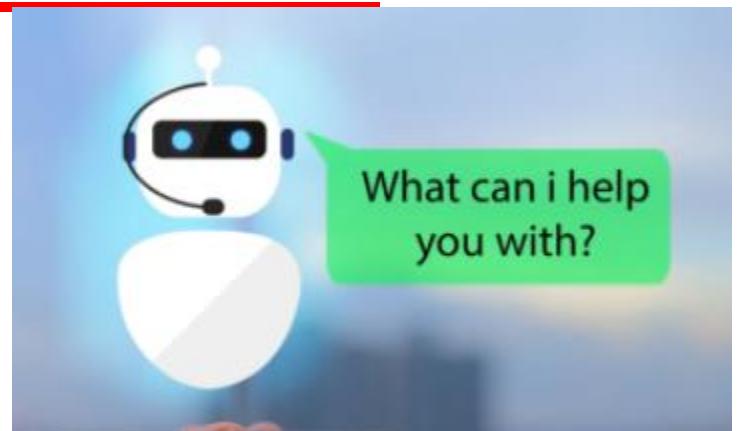
- Analyze, understand and generate human languages just like humans do.
- Applying computational techniques to language domain..
- To explain linguistic theories, to use the theories to build systems that can be of social use..
- Started off as a branch of Artificial Intelligence..
- Borrows from Linguistics, Psycholinguistics, Cognitive Science & Statistics.
- Make computers learn our language rather than we learn theirs.

# Natural Language processing



- Increase of online data
- Process unstructured data and extract meaningful information
- Challenges
- Solution-Natural language processing

# Application of NLP



English ▾

Hindi ▾

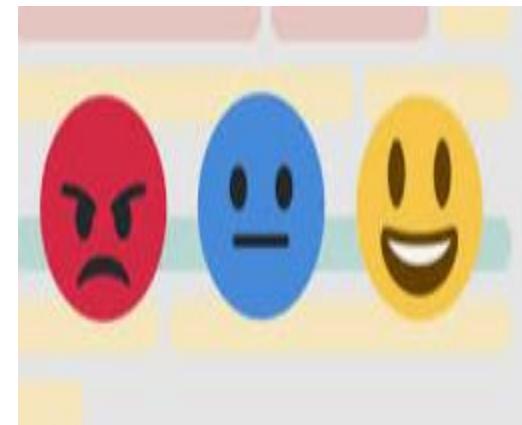
i am fine

×

मैं ठीक हूँ  
main theek hoon

1

Community verified



# History of NLP

---

1950-Alan turing published paper

1957-Chomsky published his book, Syntactic Structures

1958-LISP

1964-ELIZA

1966-AI,NLP ,Machine transalation

1980-IBM

- **1988:** Latent Semantic Analysis patent
- **January 2011:** IBM Watson beats Jeopardy! champions
- **October 2011:** Apple Siri launches in beta
- **April 2014:** Microsoft Cortana demoed
- **November 2014:** Amazon Alexa
- **May 2016:** Google Assistant

**2020 – Conversational Agents**

**2022 -- ChatGPT**

# Example

---

Dave: Open the pod bay doors, HAL.

HAL: I am sorry, Dave. I am afraid I can't do that.

Dave: What's the problem.

HAL: I think you know what the problem is just as well as I do.

Dave: I don't know what you're talking about.

HAL: I know that you and Frank were planning to disconnect me, and I'm afraid that's something I cannot allow to happen.

General speech and language understanding and generation capabilities

Politeness: emotional intelligence

Self-awareness: a model of self, including goals and plans

Belief ascription: modeling others; reasoning about their goals and plans

# Contd..



Hal: I can tell from the tone of your voice, Dave, that you're upset.  
Why don't you take a stress pill and get some rest.

[Dave has just drawn another sketch of Dr. Hunter].

HAL: Can you hold it a bit closer?

[Dave does so].

HAL: That's Dr. Hunter, isn't it?

Dave: Yes.

**Recognition of emotion from speech**

**Vision capability including visual recognition of emotions and faces**

**Also: situational ambiguity**

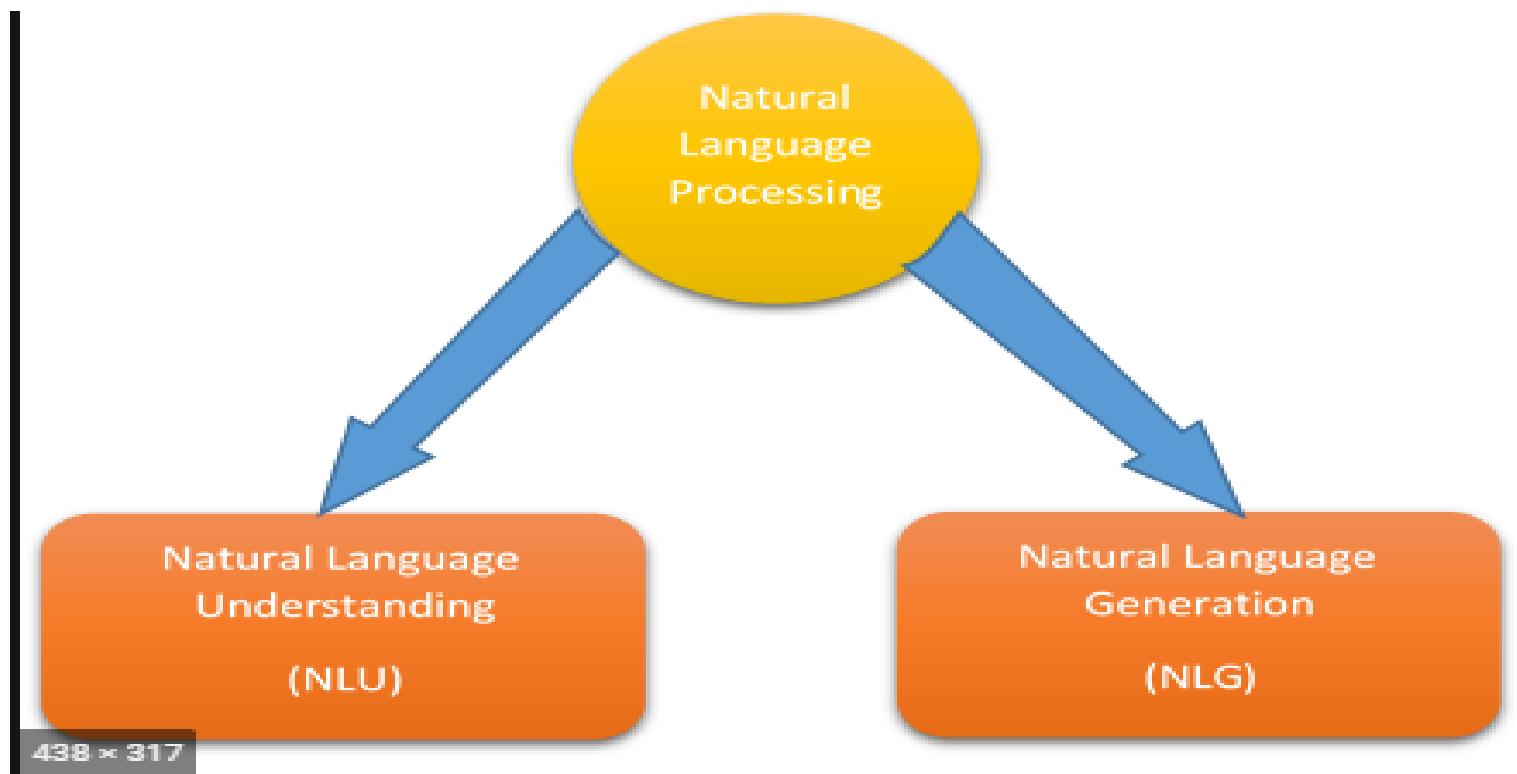
# Contd..

---

To attain the levels of performance we attribute to HAL, we need to be able to define, model, acquire and manipulate

- Knowledge of the world and of agents in it,
- Text meaning,
- Intention

# Main components of NLP



# Natural language understanding

---

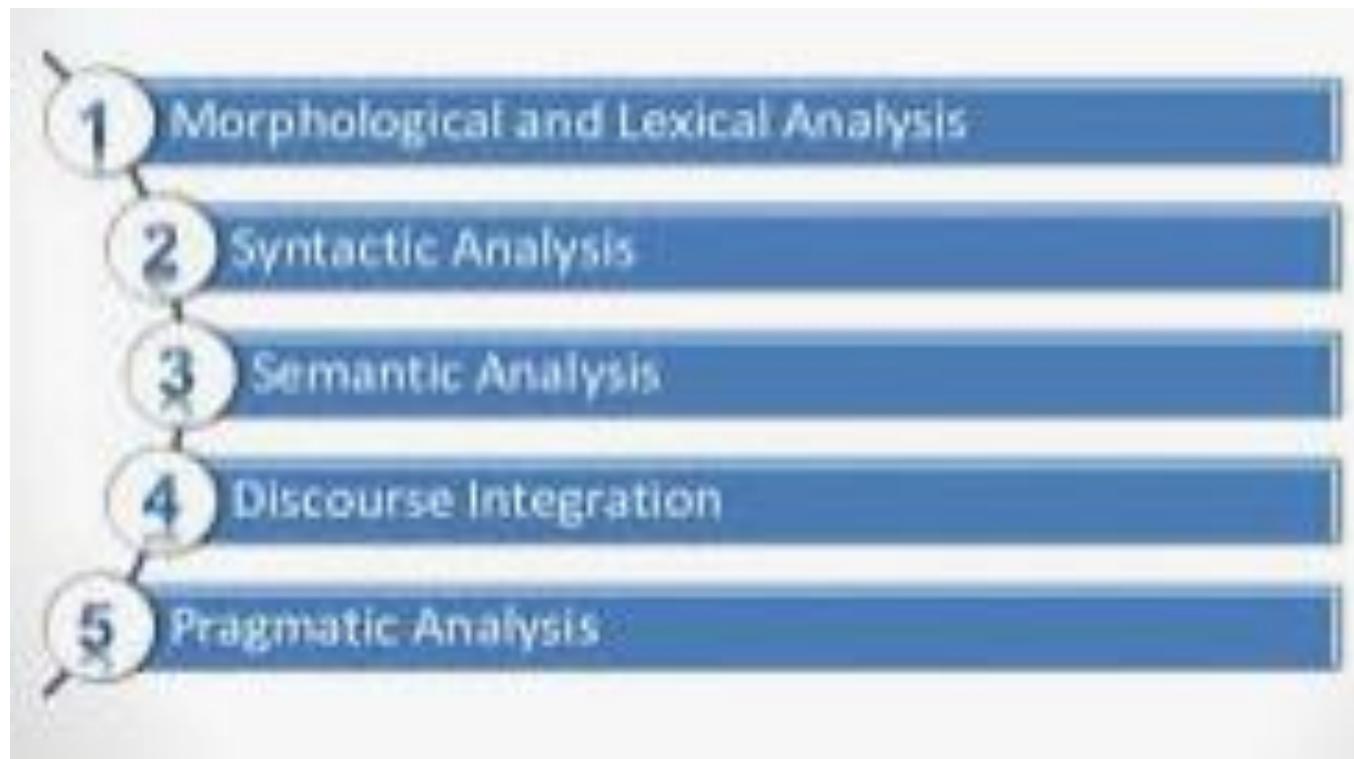
- allows users to interact more naturally with the computer.
- **Different levels of analysis**
  - Morphological Analysis
  - Syntactic analysis
  - Semantic analysis
  - Discourse analysis

# Natural language generation

---

- NLG will decide how to respond by
  - Deep Planning
  - Syntactic generation

# Steps used for NLU



# Morphology analysis

---

- Morphology is the arrangement and relationships of the smallest meaningful units in a language.
- Eg:
  - Firehouse
  - Doghouse
  - runs

# Lexical analysis

---

- Lexicon of a language means the collection of words and phrases in a language.
- Lexical analysis is dividing the whole chunk of text into paragraphs, sentences, and words.
- **friendful or beautyship** → **Lexically Incorrect**
- **friendship and beautiful** is **correct** → **Lexically Correct**

# Syntactic analysis

---

➤ Syntax analysis checks the text for meaningfulness comparing to the rules of formal

➤ Eg:

“the girl go to the school “

Agra goes to the Poonam

# Semantic analysis

---

- It draws the exact meaning or the dictionary meaning from the text. The text is checked for meaningfulness.
- Eg: Colourless blue idea

# Discourse integration

---

- The meaning of any sentence depends upon the meaning of the sentence just before it
- Eg:  
she wanted it

# Pragmatic analysis

---

- how sentences are used in different situations  
    how use affects the interpretation of sentence
- Eg:
- Close the window
- She cuts banana with a pen

# Introduction to Natural Language Processing

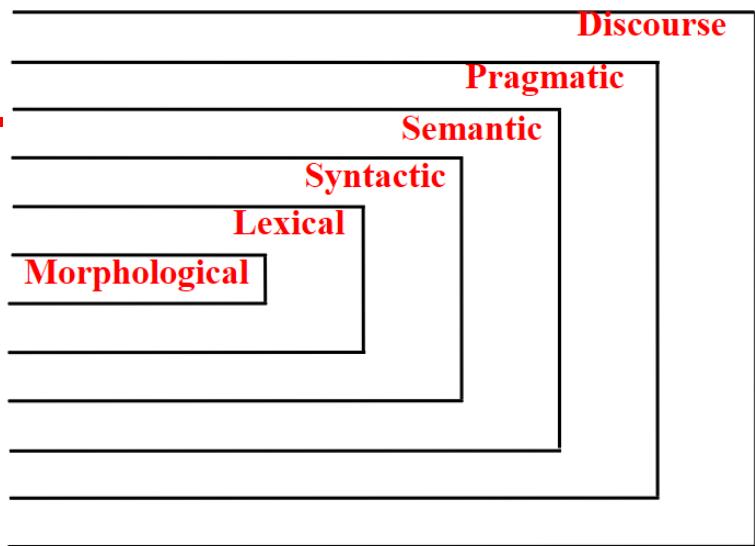


## Different Levels of Language Analysis

Examples:

Syntax, Semantics, and Pragmatics

1. Green frogs have large noses.  
**Pragmatically wrong**
  
2. Green ideas have large noses.  
**Semantics, and Pragmatics**
  
3. Large have green ideas nose.  
**All three wrong**
  
4. I are fond in animals (exercise)
  
5. My cat is studying Linguistics (exercise)
  
6. Spain is larger than China (exercise)



# Natural Language Generation

---

- producing text from computer data
- Steps
  - discourse planning
  - Surface realizer
  - Lexical selection

# Why NLP is hard

---

- Ambiguity
- Eg:
  - Teacher strikes idle kids.
  - Sarah gave a bath to her dog wearing a pink t-shirt.
  - RBI raises interest rates or RBI raises, interest rates.

- **Ambiguities at all level**

- I. **Structural Ambiguities**

- Namrata thinks she understands me.
    - She thinks Namrata understands me.
    - Visiting relatives can be nuisance. (two meanings)

- II. **Lexical Ambiguities:**

- I saw bats

Contd..

---

## Non Standard english

- don't follow any rules
- Eg: gm, gudnit etc.

## Segmentation issues

- Eg The old city-bus stop

## Idioms

- Eg:
  - dark horse
  - lose face

---

# World knowledge

Eg:

Mary and Sue are sisters.

Mary and Sue are mothers.

## Tricky entity names

- Let it be was recorded.
- Where is a bug's life playing?

# Some of the tasks in NLP

---

## Sentence segmentation

- It breaks the paragraph into separate sentences.
- Eg:**Independence Day is one of the important festivals for every Indian citizen. It is celebrated on the 15th of August each year ever since India got independence from the British rule. The day celebrates independence in the true sense.**
- "Independence Day is one of the important festivals for every Indian citizen."
- "It is celebrated on the 15th of August each year ever since India got independence from the British rule."
- "This day celebrates independence in the true sense."

# Contd..

---

## ➤ Tokenization

- splits longer strings of text into smaller pieces, or **tokens**.

Eg:JavaTpoint offers Corporate Training, Summer Training, Online Training, and Winter Training.

"JavaTpoint", "offers", "Corporate", "Training", "Summer",  
"Training", "Online", "Training", "and", "Winter", "Training",  
". "

## ➤ Lemmatization

- capture canonical forms based on a word's lemma.
- Eg:better → good

Contd..

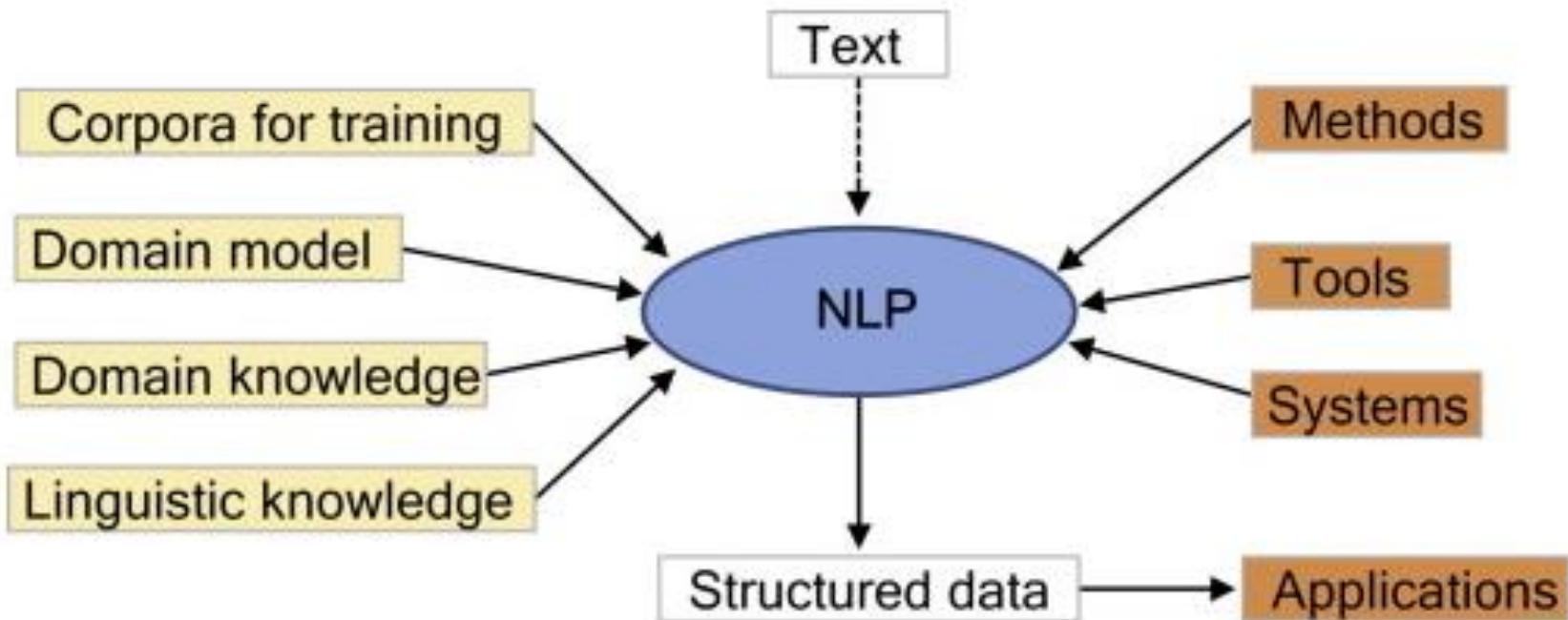
## ➤ **Stemming**

- process of eliminating affixes (suffixed, prefixes, infixes, circumfixes) from a word in order to obtain a word stem.
- eg. running → run

## ➤ **Stop Words**

- contribute little to overall meaning
- Eg:He is a good boy.

# Schematic representation NLP



---

# Representations and Understanding

# Syntactic representations of language

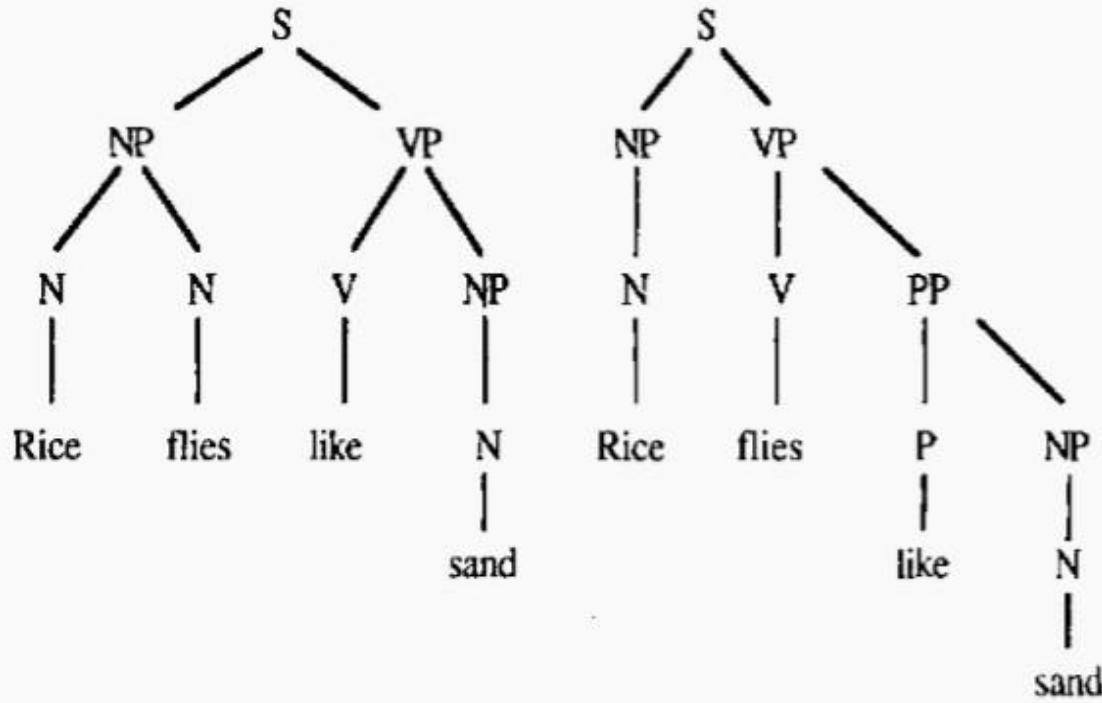
---

- Most syntactic representations of language are based on the notion of **context-free grammars**
- **CFGs** represent sentence structure in terms of what phrases are subparts of other phrases.
- Often presented in a tree form

- (1)  $S \rightarrow NP VP$
- (2)  $NP \rightarrow ART ADJ N$
- (3)  $NP \rightarrow ART N$
- (4)  $NP \rightarrow ADJ N$
- (5)  $VP \rightarrow AUX VP$
- (6)  $VP \rightarrow V NP$

# Representations and Understanding

Allen 1995: Natural Language Understanding - Introduction



**Figure 1.4** Two structural representations of *Rice flies like sand*.

NP – Noun Phrases

VP – Verb Phrases

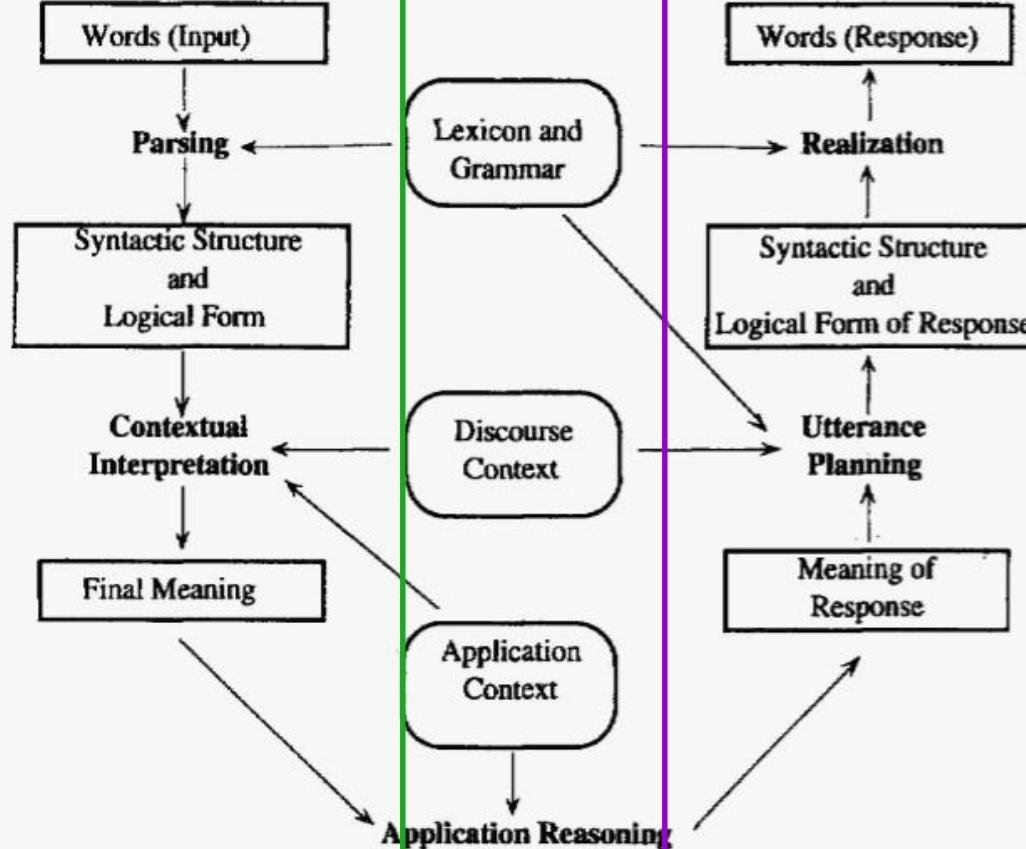
PP – Prepositional Phrases

Figure 1.4 Two structural representations of "Rice flies like sand".

# The Organization of Natural Language Processing Systems

**Natural  
Language  
Understanding**

Allen 1995: Natural Language Understanding - Introduction



**Natural  
Language  
Generation**

# Evaluating Language Understanding Systems

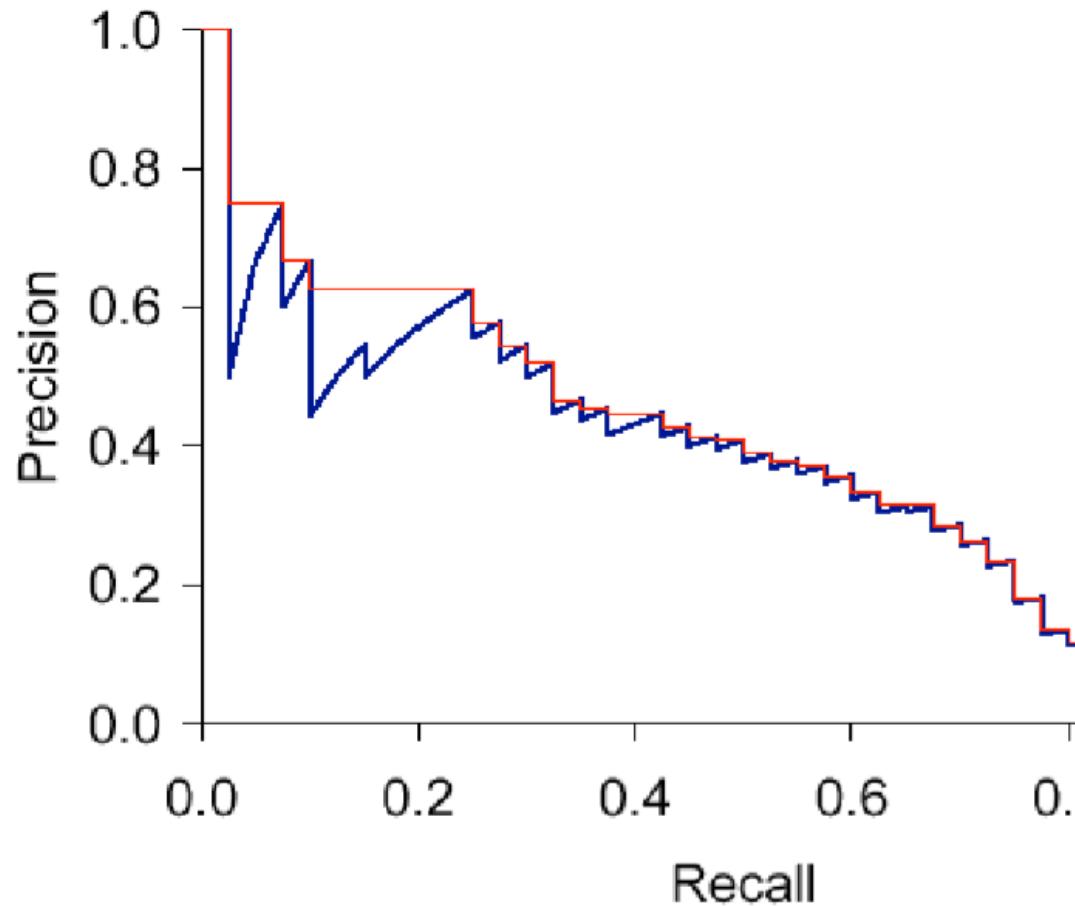
---

- What metrics to use?
- How to deal with complex outputs like translations?
- Are the human judgments measuring something real? reliable?
- Is the sample of texts sufficiently representative?
- How reliable or certain are the results?

# Contingency Table

		<i>gold standard labels</i>		
		gold positive	gold negative	
<i>system output labels</i>	system positive	<b>true positive</b>	<b>false positive</b>	<b>precision</b> = $\frac{tp}{tp+fp}$
	system negative	<b>false negative</b>	<b>true negative</b>	
		<b>recall</b> = $\frac{tp}{tp+fn}$		<b>accuracy</b> = $\frac{tp+tn}{tp+fp+tn+fn}$

# Tradeoff between Precision and Recall



# NLTK Installation

---

The Natural Language Toolkit (NLTK) is a platform used for building programs for text analysis.

Open Anaconda terminal, run

**pip install nltk.**

Anaconda and Jupiter are best and popular data science tools

In Jupiter, the console commands can be executed by the ‘!’ sign before the command within the cell.

**! pip install nltk**

[NLTK book](#)

[NLTK discussion forum](#)

<https://www.nltk.org/install.html>

# NLP Tools

---

Some commercial tools

[IBM Watson](#) | A pioneer AI platform for businesses

[Google Cloud NLP API](#) | Google technology applied to NLP

[Amazon Comprehend](#) | An AWS service to get insights from text

# NLP Tools

## Open Source Tools

[Stanford Core NLP](#) is a popular Java library built and maintained by Stanford University.

**SpaCy** - One of the newest open-source Natural Language Processing with Python libraries

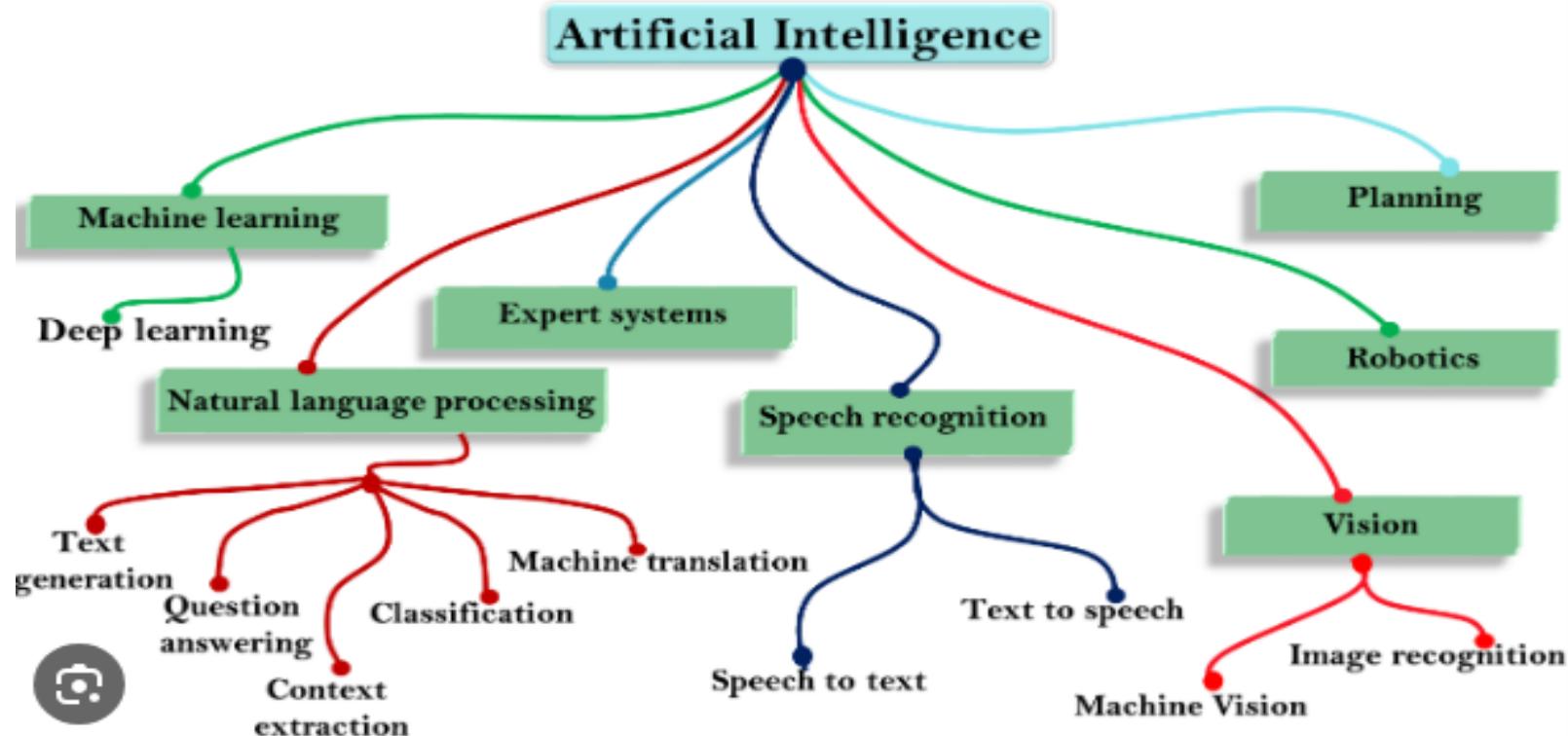
[Gensim](#) is a highly specialized Python library that largely deals with topic modeling tasks using algorithms like Latent Dirichlet Allocation (LDA)

[Natural Language Toolkit \(NLTK\)](#) is the most popular Python library

**Generative Pre-trained Transformer 3 (GPT-3)** is an [autoregressive language model](#) released recently by [Open AI](#), pre-trained on (175 billion parameters). It is autocompleting program and is used mainly for predicting text

**AllenNLP**: Powerful tool for prototyping with good text processing capabilities. Automates some of the tasks which are essential for almost every deep learning model. It provides a lot of modules like Seq2VecEncoder, Seq2SeqEncoder.

**Berkeley Neural Parser** (Python). It is a high-accuracy parser with models for 11 languages. It cracks the syntactic structure of sentences into nested sub phrases. This tool enables the easy extraction of information from syntactic constructs



# References

- <https://emerj.com/partner-content/nlp-current-applications-and-future-possibilities/>
- <https://venturebeat.com/2019/04/05/why-nlp-will-be-big-in-2019/>
- <https://www.nltk.org/book/>
- <https://www.coursera.org/learn/python-text-mining/home/week/1>
- <https://openai.com/api/>
- <https://analyticssteps.com/blogs/top-nlp-tools>
- <https://web.stanford.edu/~jurafsky/NLPCourseraSlides.html>
- [https://www.cstr.ed.ac.uk/emasters/course/natural\\_lang.html](https://www.cstr.ed.ac.uk/emasters/course/natural_lang.html)
- <https://web.stanford.edu/class/cs224u/2016/materials/cs224u-2016-intro.pdf>
- <https://www.mygreatlearning.com/blog/trending-natural-language-processing-applications/>

# Thank You... 😊

- Q&A
- Suggestions / Feedback



# Natural Language Processing

## DSECL ZG565



**BITS** Pilani  
Pilani Campus

Dr.Vijayalakshmi Anand

BITS-Pilani



**Session 2  
Date – 8DEC 2023  
Time – 6pm to 8pm**

These slides are prepared by the instructor, with grateful acknowledgement of James Allen and many others who made their course materials freely available online.

# Content Sequence Plan

- M1 Natural Language Understanding and Generation
- M2 N-gram Language Modelling
- M7 Vector semantics and Embedding
- M3 Neural networks and Neural language Models
- M4 Part-of-Speech Tagging
- M5 Hidden Markov Models and MEMM
- M6 Topic Modelling
- M8 Grammars and Parsing
- M9 Statistical Constituency Parsing
- M10 Dependency Parsing
- M11 Encoder-Decoder Models, Attention and Contextual Embedding
- M12 Word sense disambiguation
- M13 Semantic web ontology and Knowledge Graph
- M14 Introduction to NLP Applications

# Session#2 – N gram Language model

---

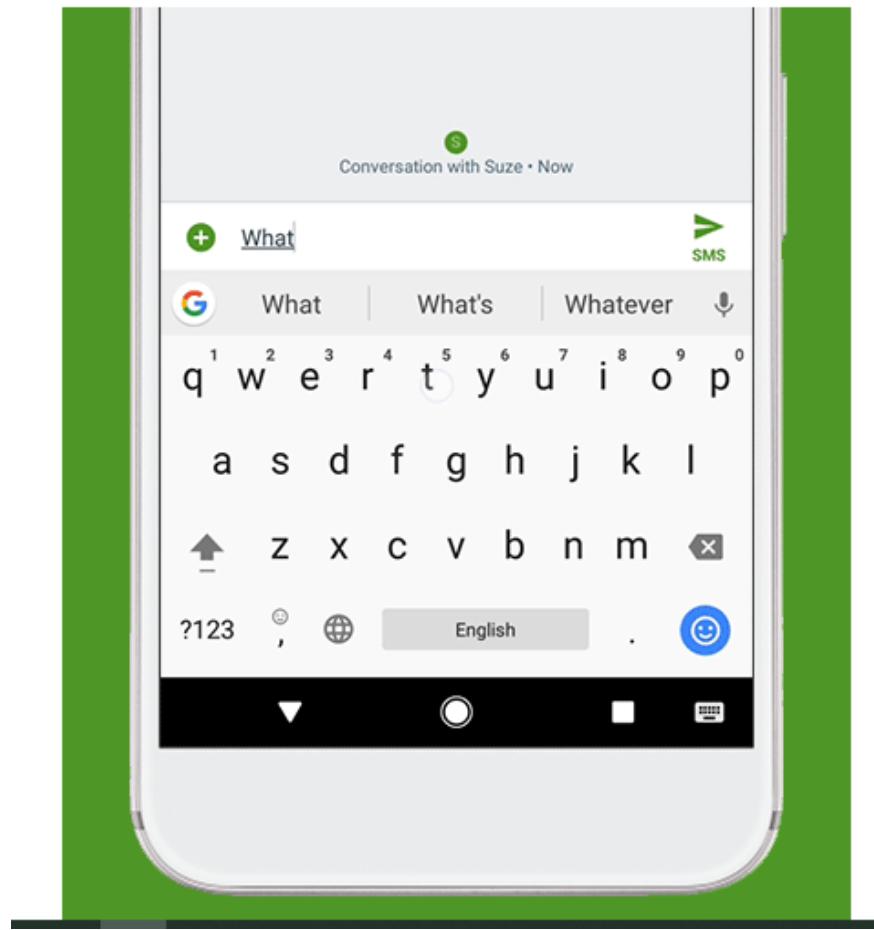
- Human word Prediction
- Language Models
  - What is language model
  - Why language models
- N-gram language models
  - Uni-gram , bi-gram and N-gram
- Evaluation of Language models
  - Perplexity
- Smoothing
  - Laplace smoothing
  - Interpolation and Back off

# Human word Prediction

---

- We may have ability to predict future words in an utterance .
  - How?
    - ❖ Based on domain knowledge  
*red blood*
    - ❖ Based on syntactic knowledge  
*the <adj/noun>*
    - ❖ Based on Lexical knowledge  
*baked <potato>*
-

# Example



# Language modelling



A model that computes either of these:

Probability of a sentence (  $P(W)$  ) or

Probability of an upcoming word (  $P(w_n | w_1, w_2 \dots w_{n-1})$  ) is called a **language model**.

Simply we can say that

A language model learns to predict the probability of a sequence of words.

# Language Models



## 1. Machine Translation:

Machine translation system is used to translate one language to another language .For example Chinese to English or German to English etc.



# Continued..

---

## 2.Spell correction

Example:

I picked up **the** phone to answer her fall

I picked up the phone to answer her call>>>>>I  
picked up the phone to answer her fall

---

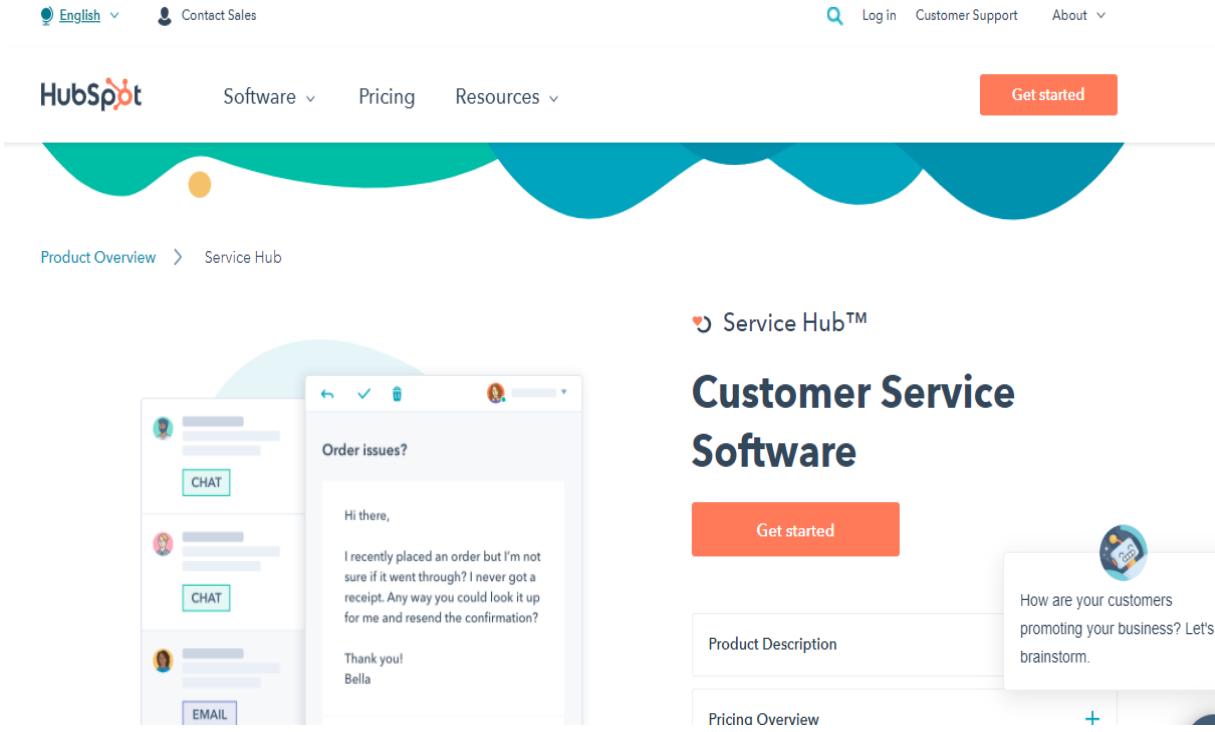
### 3.Speech Recognition

**Speech recognition** is the ability of a machine or program to identify words and phrases in spoken language and convert them to a machine-readable format.



# Sentiment analysis

## Example :Hubspot's Service



The screenshot illustrates the HubSpot Service Hub software. At the top, there is a navigation bar with links for English, Contact Sales, Log in, Customer Support, and About. The main menu includes Software, Pricing, and Resources. A prominent orange "Get started" button is located on the right side of the header.

The interface features a decorative wavy bar at the bottom in shades of green, yellow, and blue. Below this, a navigation breadcrumb shows Product Overview > Service Hub.

On the left, a sidebar displays three communication channels: CHAT (with two recent messages), EMAIL (with one recent message), and another CHAT section. The main content area shows a chat interaction between a customer and a representative named Bella:

**Order issues?**

Hi there,  
I recently placed an order but I'm not sure if it went through? I never got a receipt. Any way you could look it up for me and resend the confirmation?  
Thank you!  
Bella

To the right, the "Service Hub™" logo is displayed above the heading "Customer Service Software". A large orange "Get started" button is positioned below the heading. A callout box contains the text: "How are your customers promoting your business? Let's brainstorm." Below the heading, there are sections for "Product Description" and "Pricing Overview".

# How to build a language model



- Recall the definition of conditional probabilities

$$p(B|A) = P(A,B)/P(A) \quad \text{Rewriting: } P(A,B) = P(A)P(B|A)$$

- More variables:

$$P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$$

- The **Chain Rule** in General

$$P(x_1, x_2, x_3, \dots, x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)\dots P(x_n|x_{n-1}, x_{n-2}, \dots, x_1)$$

# Example

In a factory there are 100 units of a certain product, 5 of which are defective. We pick three units from the 100 units at random. What is the probability that none of them are defective?

Let  $A_i$  as the event and  $i=1,2,3$

$$P(A_1) = \frac{95}{100}.$$

Given that the first chosen item was good, the second item will be chosen from 94 good units and 5 defective units, thus

$$P(A_2|A_1) = \frac{94}{99}.$$

Given that the first and second chosen items were okay, the third item will be chosen from 93 good units and 5 defective units, thus

$$P(A_3|A_2, A_1) = \frac{93}{98}.$$

$$\begin{aligned} P(A_1 \cap A_2 \cap A_3) &= P(A_1)P(A_2|A_1)P(A_3|A_2, A_1) \\ &= \frac{95}{100} \frac{94}{99} \frac{93}{98} \\ &= 0.8560 \end{aligned}$$

# The Chain Rule applied to compute joint probability of words in sentence

---

- $P(w_1 w_2 \dots \dots w_n) = \prod_i (P(w_i | w_1 w_2 \dots \dots w_{i-1}))$
- For ex:  
 $P(\text{"its water is so transparent"}) =$   
 $P(\text{its}) \times P(\text{water} | \text{its}) \times P(\text{is} | \text{its water})$   
 $\times P(\text{so} | \text{its water is}) \times P(\text{transparent} | \text{its water is so})$
- **P(most biologists and specialist believe that in fact the mythical unicorn horns derived from the narwhal)**

# Markov Assumption

- Simplifying assumption:

*limit history of fixed number of words N1*



Andrei Markov

$P(\text{the} \mid \text{its water is so transparent that}) \gg P(\text{the} \mid \text{that})$

or

$P(\text{the} \mid \text{its water is so transparent that}) \gg P(\text{the} \mid \text{transparent that})$

# Markov Assumption

---

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i | w_{i-k} \dots w_{i-1})$$

In other words, we approximate each component in the product

$$P(w_i | w_1 w_2 \dots w_{i-1}) \approx P(w_i | w_{i-k} \dots w_{i-1})$$

# N -Gram Language models

# N-gram Language models

---

- N gram is a sequence of tokens(words)
- Unigram language model(N=1 )

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i)$$

Example:

$P(\text{I want to eat Chinese food}) \approx P(\text{I})P(\text{want})$   
 $P(\text{to})P(\text{eat})P(\text{Chinese})P(\text{food})$

# Bigram model

N=2

$$P(w_i | w_1 w_2 \dots w_{i-1}) \approx P(w_i | w_{i-1})$$

$\langle \text{S} \rangle$  —  $\langle / \text{S} \rangle$

Example:

$P(\text{I want to eat Chinese food}) \approx P(\text{I} | \langle \text{start} \rangle) P(\text{want} | \text{I}) P(\text{to} | \text{want})$   
 $P(\text{eat} | \text{to}) P(\text{Chinese} | \text{eat}) P(\text{food} | \text{Chinese})$   
 $P(\langle \text{end} \rangle | \text{food})$

# N-gram models

---

- We can extend to trigrams, 4-grams, 5-grams

## Advantages

- no human supervision, easy to extend to more data, allows querying about open-class relations,

## Disadvantage:

- In general this is an insufficient model of language
  - because language has **long-distance dependencies**:

“The computer(s) which I had just put into the machine room on the fifth floor is (are) crashing.”

# Estimating N-gram Probabilities

# Estimating bigram probabilities

---

- Estimating the probability as the relative frequency is the *maximum likelihood estimate* (or *MLE* ), because this value makes the observed data maximally likely
- The Maximum Likelihood Estimate

$$P(w_i | w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

$$P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

# An example

$$P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

$$P(\text{I} | \text{<s>}) = \frac{2}{3} = .67$$

$$P(\text{/s} | \text{Sam}) = \frac{1}{2} = 0.5$$

$$P(\text{Sam} | \text{<s>}) = \frac{1}{3} = .33$$

$$P(\text{Sam} | \text{am}) = \frac{1}{2} = .5$$

$$P(\text{am} | \text{I}) = \frac{2}{3} = .67$$

$$P(\text{do} | \text{I}) = \frac{1}{3} = .33$$

# Evaluation

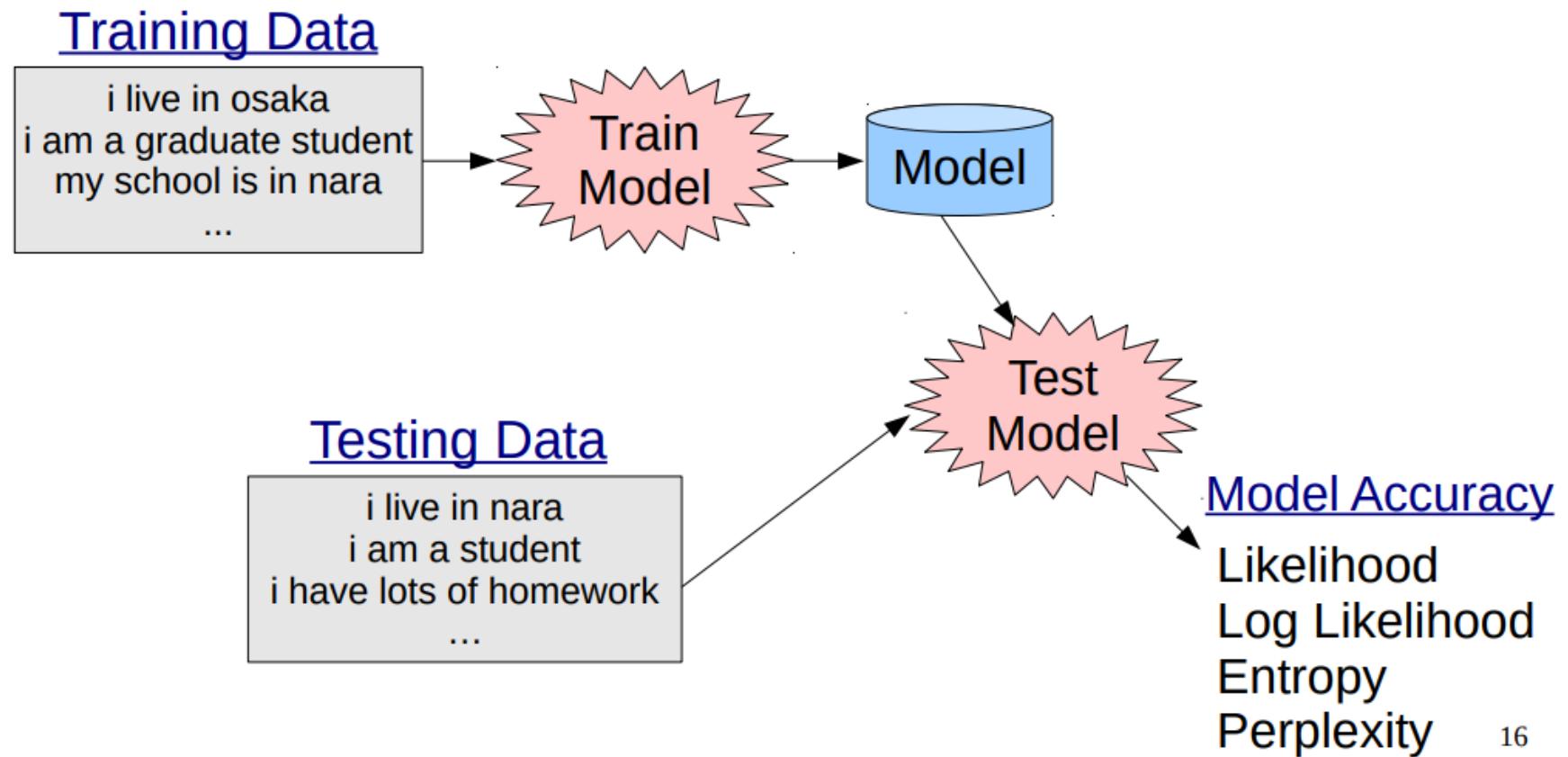


# How good is our model?

---

- Does our language model prefer good sentences to bad ones?
  - Assign higher probability to “real” or “frequently observed” sentences
    - Than “ungrammatical” or “rarely observed” sentences?
- We train parameters of our model on a **training set**.
- We test the model’s performance on data we haven’t seen.
  - A **test set** is an unseen dataset that is different from our training set, totally unused.
  - An **evaluation metric** tells us how well our model does on the test set.

# Experimental setup



# Example1:

Training data	Model	Test data
<p>There is a big house</p> <p>I buy a house</p> <p>They buy the new house</p>	<p><math>p(\text{big}   \text{a}) = 0.5</math></p> <p><math>p(\text{is}   \text{there}) = 1</math></p> <p><math>p(\text{buy}   \text{they}) = 1</math></p> <p><math>p(\text{house}   \text{a}) = 0.5</math></p> <p><math>p(\text{buy}   \text{i}) = 1</math></p> <p><math>p(\text{a}   \text{buy}) = 0.5</math></p> <p><math>p(\text{new}   \text{the}) = 1</math></p> <p><math>p(\text{house}   \text{big}) = 1</math></p> <p><math>p(\text{the}   \text{buy}) = 0.5</math></p> <p><math>p(\text{a}   \text{is}) = 1</math></p> <p><math>p(\text{house}   \text{new}) = 1</math></p> <p><math>p(\text{they}   \langle s \rangle) = .333</math></p>	<p>S1: they buy a big house  <math>P(S1) = 0.333 * 1 * 0.5 * 0.5 * 1</math>  <math>P(S1) = 0.0833</math></p> <p>S2: they buy a new house  <math>P(S2) = ?</math></p>

$$P(w_1, w_2, \dots, w_T) = \prod_{t=1}^T P(w_t | w_{t-1})$$

# Extrinsic evaluation of N-gram models

---



- Best evaluation for comparing models A and B
    - Put each model in a task
      - spelling corrector, speech recognizer, MT system
    - Run the task, get an accuracy for A and for B
      - How many misspelled words corrected properly
      - How many words translated correctly
    - Compare accuracy for A and B
-

# Difficulty of extrinsic (in-vivo) evaluation of N-gram models

---

- Extrinsic evaluation
  - Time-consuming; can take days or weeks
  - Bad approximation
    - unless the test data looks **just** like the training data
    - So **generally only useful in pilot experiments**
- So
  - Sometimes use **intrinsic** evaluation: **perplexity**

# Intuition of Perplexity



- The Shannon Game:

- How well can we predict the next word?

I always order pizza with cheese and....

The president of India is.....

I wrote a .....

{

- mushrooms 0.1
- pepperoni 0.1
- anchovies 0.01
- ....
- fried rice 0.0001
- ....
- and  $1e-100$

- Unigrams are terrible at this game. (Why?)
- A better model of a text
  - is one which assigns a higher probability to the word that actually occurs

# Perplexity



The best language model is one that best predicts an unseen test set

- Gives the highest  $P(\text{sentence})$

Perplexity is the inverse probability of the test set, normalized by the number of words:

Chain rule:

For bigrams:

$$PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}$$

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_1 \dots w_{i-1})}}$$

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_{i-1})}}$$

**Minimizing perplexity is the same as maximizing probability**

# Example1

Using bigram model

$$P(I \mid I \text{ am not}) =$$

$$P(I \mid <s>)P(I \mid I)P(\text{am} \mid I)P(\text{not} \mid \text{am})$$

$$= 3/3 * \frac{1}{4} * 2/4 * \frac{1}{2}$$

$$= 1 * .25 * .5 * .5 = 0.0625$$

$$\text{Perplexity} = \text{PP}(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$

$$\text{PP}(I \mid I \text{ am not}) = (0.0625)^{-1/4}$$

$$= 1 / (0.0625)^{1/4}$$

$$= 2$$

# Example2

<S> I am Henry </S>  
 <S> I like college </S>  
 <S> Do Henry like college </S>  
 <S> Henry I am </S>  
 <S> Do I like Henry </S>  
 <S> Do I like college </S>  
 <S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

Perplexity for Bigram <S> I like college </S>

$$\begin{aligned}
 &= P(I | <S>) \times P(\text{like} | I) \times P(\text{college} | \text{like}) \times P(</S> | \text{college}) \\
 &= 3/7 \times 3/6 \times 3/5 \times 3/3 = 9/70 = 0.13
 \end{aligned}$$

$$PP(w) = (1/0.13)^{1/4} = 1.67$$

Perplexity for Trigram <S> I like college </S>

$$\begin{aligned}
 P(w) &= P(\text{like} | <S> I) \times P(\text{college} | I \text{ like}) \times P(</S> | \text{like college}) \\
 P(w) &= 1/3 \times 2/3 \times 3/3 = 2/9 = 0.22 \\
 PP(w) &= (1/0.22)^{1/3} = 1.66
 \end{aligned}$$

# Corpora

## Examples of corpora (in chronological order)

Focusing on English; most released by the [Linguistic Data Consortium \(LDC\)](#):

**Brown:** 500 texts, 1M words in 15 genres. POS-tagged. **SemCor** subset (234K words) labelled with WordNet word senses.

**WSJ:** 6 years of *Wall Street Journal*; subsequently used to create Penn Treebank, PropBank, and more! Translated into Czech for the **Prague Czech-English Dependency Treebank**.

**ECI:** European Corpus Initiative, multilingual.

**BNC:** 100M words; balanced selection of written and spoken genres.

**Redwoods:** Treebank aligned to wide-coverage grammar; several genres.

**Gigaword:** 1B words of news text.

**AMI:** Multimedia (video, audio, synchronised transcripts).

**Google Books N-grams:** 5M books, 500B words (361B English).

**Flickr 8K:** images with NL captions

**English Visual Genome:** Images, bounding boxes ⇒ NL descriptions

# Lower perplexity = better model

- Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

# Generalization and Zeros

# The perils of overfitting

---

- N-grams only work well for word prediction if the test corpus looks like the training corpus
  - In real life, it often doesn't
  - We need to train robust models that generalize!
  - One kind of generalization: Zeros!
    - Things that don't ever occur in the training set
    - But occur in the test set

# Problems with simple MLE estimations

- Training set:
  - ... denied the allegations
  - ... denied the reports
  - ... denied the claims
  - ... denied the request
- Test set
  - ... denied the offer
  - ... denied the loan

$$P(\text{"offer"} \mid \text{denied the}) = 0$$

# Zero probability bigrams

---

- Bigrams with zero probability
    - mean that we will assign 0 probability to the test set!
  - And hence we cannot compute perplexity (can't divide by 0)!
-

# Laplace Smoothing (Add 1 smoothing)

---

- Pretend we saw each word one more time than we did
- Just add one to all the counts!
- MLE estimate:
- Add-1 estimate:

$$P_{MLE}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

$$P_{Add-1}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

# Example

JOHN READ MOBY DICK  
 MARY READ A DIFFERENT BOOK  
 SHE READ A BOOK BY CHER

$$p(\text{JOHN READ A BOOK})$$

$$\begin{aligned}
 &= p(\text{JOHN}|\bullet) \quad p(\text{READ}|\text{JOHN}) \quad p(\text{A}|\text{READ}) \quad p(\text{BOOK}|\text{A}) \quad p(\bullet|\text{BOOK}) \\
 &= \frac{c(\bullet \text{ JOHN})}{\sum_w c(\bullet \text{ } w)} \quad \frac{c(\text{JOHN READ})}{\sum_w c(\text{JOHN } w)} \quad \frac{c(\text{READ A})}{\sum_w c(\text{READ } w)} \quad \frac{c(\text{A BOOK})}{\sum_w c(\text{A } w)} \quad \frac{c(\text{BOOK } \bullet)}{\sum_w c(\text{BOOK } w)} \\
 &= \frac{1}{3} \quad \frac{1}{1} \quad \frac{2}{3} \quad \frac{1}{2} \quad \frac{1}{2} \\
 &\approx 0.06
 \end{aligned}$$

JOHN READ MOBY DICK  
 MARY READ A DIFFERENT BOOK  
 SHE READ A BOOK BY CHER

$$p(\text{CHER READ A BOOK})$$

$$\begin{aligned}
 &= p(\text{CHER}|\bullet) \quad p(\text{READ}|\text{CHER}) \quad p(\text{A}|\text{READ}) \quad p(\text{BOOK}|\text{A}) \quad p(\bullet|\text{BOOK}) \\
 &= \frac{c(\bullet \text{ CHER})}{\sum_w c(\bullet \text{ } w)} \quad \frac{c(\text{CHER READ})}{\sum_w c(\text{CHER } w)} \quad \frac{c(\text{READ A})}{\sum_w c(\text{READ } w)} \quad \frac{c(\text{A BOOK})}{\sum_w c(\text{A } w)} \quad \frac{c(\text{BOOK } \bullet)}{\sum_w c(\text{BOOK } w)} \\
 &= \frac{0}{3} \quad \frac{0}{1} \quad \frac{2}{3} \quad \frac{1}{2} \quad \frac{1}{2} \\
 &= 0
 \end{aligned}$$

# After Add-1 smoothing

JOHN READ MOBY DICK  
MARY READ A DIFFERENT BOOK  
SHE READ A BOOK BY CHER

$p(\text{JOHN READ A BOOK})$

$$= \frac{1+1}{11+3} \quad \frac{1+1}{11+1} \quad \frac{1+2}{11+3} \quad \frac{1+1}{11+2} \quad \frac{1+1}{11+2}$$

$$\approx 0.0001$$

$p(\text{CHER READ A BOOK})$

$$= \frac{1+0}{11+3} \quad \frac{1+0}{11+1} \quad \frac{1+2}{11+3} \quad \frac{1+1}{11+2} \quad \frac{1+1}{11+2}$$

$$\approx 0.00003$$

# Berkeley Restaurant Project corpus

- Let's compute simple  $N$ -gram models of speech queries about restaurants in Berkeley California.
  - E.g.,
    - can you tell me about any good cantonese restaurants close by*
    - mid priced thai food is what i'm looking for*
    - tell me about chez panisse*
    - can you give me a listing of the kinds of food that are available*
    - i'm looking for a good place to eat breakfast*
    - when is caffe venezia open during the day*

# Raw Bigram Counts

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

# Converting to probabilities

- Obtain likelihoods by dividing bigram counts by unigram counts.

	I	want	to	eat	Chinese	food	lunch	spend
Unigram counts:	2533	927	2417	746	158	1093	341	278
$P(w_t w_{t-1})$	I	want	to	eat	Chinese	food	lunch	spend
I	0.002	0.33	0	0.0036	0	0	0	0.00079

$$P(want|I) \approx \frac{Count(I\ want)}{Count(I)} = \frac{827}{2533} \approx 0.33$$

$$P(spend|I) \approx \frac{Count(I\ spend)}{Count(I)} = \frac{2}{2533} \approx 7.9 \times 10^{-4}$$

# Contd...

- Obtain likelihoods by dividing bigram counts by unigram counts.

	I	want	to	eat	Chinese	food	lunch	spend
Unigram counts:	2533	927	2417	746	158	1093	341	278
$P(w_t w_{t-1})$	I	want	to	eat	Chinese	food	lunch	spend

	I	want	to	eat	Chinese	food	lunch	spend
I	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
Chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

# Berkeley Restaurant Corpus: Laplace smoothed bigram



- Out of 9222 sentences in Berkeley restaurant corpus,
  - e.g., “I want” occurred 827 times so Laplace gives 828

Count( $w_{t-1}, w_t$ )		$w_t$							
		I	want	to	eat	Chinese	food	lunch	spend
$w_{t-1}$	I	5+1	827+1	1	9+1	1	1	1	2+1
	want	2+1	1	608+1	1+1	6+1	6+1	5+1	1+1
	to	2+1	1	4+1	686+1	2+1	1	6+1	211+1
	eat	1	1	2+1	1	16+1	2+1	42+1	1
	Chinese	1+1	1	1	1	1	82+1	1+1	1
	food	15+1	1	15+1	1	1+1	4+1	1	1
	lunch	2+1	1	1	1	1	1+1	1	1
	spend	1+1	1	1+1	1	1	1	1	1

# Laplace-smoothed bigrams

$$P_{Lap}(w_t|w_{t-1}) = \frac{C(w_{t-1}w_t) + 1}{C(w_{t-1}) + \|\mathcal{V}\|}$$

$P(w_t w_{t-1})$	I	want	to	eat	Chinese	food	lunch	spend
I	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00083	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
Chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

$V=1446$  in the Berkeley restaurant project corpus

# Reconstituted counts



$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n) + 1] \times C(w_{n-1})}{C(w_{n-1}) + V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

# Compare with raw bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

# Add-1 estimation is a blunt instrument

---

- So add-1 isn't used for N-grams:
  - We'll see better methods
- But add-1 is used to smooth other NLP models
  - For text classification
  - In domains where the number of zeros isn't so huge.

# Backoff and Interpolation

---

- Sometimes it helps to use **less** context
  - Condition on less context for contexts you haven't learned much about
- **Backoff:**
  - use trigram if you have good evidence,
  - otherwise bigram, otherwise unigram
- **Interpolation:**
  - mix unigram, bigram, trigram
- Interpolation works better

# Linear Interpolation

---

- Involves using different pieces of information to derive a probability

## Simple interpolation

$$\hat{P}(w_n | w_{n-2} w_{n-1}) = \lambda_1 P(w_n | w_{n-2} w_{n-1}) + \lambda_2 P(w_n | w_{n-1}) + \lambda_3 P(w_n)$$

$$\sum_i \lambda_i = 1$$

# How to set the lambdas?

- Use a **held-out** corpus

Training Data

Held-Out  
Data

Test  
Data

- Choose  $\lambda$ s to maximize the probability of held-out data:
  - Fix the N-gram probabilities (on the training data)
  - Then search for  $\lambda$ s that give largest probability to held-out set:

# Unknown words: Open versus closed vocabulary tasks

---

- If we know all the words in advanced
  - Vocabulary  $V$  is fixed
  - Closed vocabulary task
- Often we don't know this
  - **Out Of Vocabulary** = OOV words
  - Open vocabulary task
- Instead: create an unknown word token <UNK>
  - Training of <UNK> probabilities
    - Create a fixed lexicon  $L$  of size  $V$
    - At text normalization phase, any training word not in  $L$  changed to <UNK>
    - Now we train its probabilities like a normal word
  - At decoding time
    - If text input: Use UNK probabilities for any word not in training

# Huge web-scale n-grams

---

- How to deal with, e.g., Google N-gram corpus
- Pruning
  - Entropy-based pruning
  - Only store N-grams with count > threshold.
- Efficiency
  - Efficient data structures like tries
  - Bloom filters: approximate language models
  - Store words as indexes, not strings
    - Use Huffman coding to fit large numbers of words into two bytes

# Smoothing for Web-scale N-grams

- “Stupid backoff” (Brants *et al.* 2007)

$$S(w_i \mid w_{i-k+1}^{i-1}) = \begin{cases} \frac{\text{count}(w_{i-k+1}^i)}{\text{count}(w_{i-k+1}^{i-1})} & \text{if } \text{count}(w_{i-k+1}^i) > 0 \\ 0.4S(w_i \mid w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

$$S(w_i) = \frac{\text{count}(w_i)}{N}$$

# Exercise 1

What is the most probable next word predicted by the model for the following word sequence?

**Given Corpus**

>  
<S> I am Henry </S>  
<S> I like college </S>  
<S> Do Henry like college </S>  
<S> Henry I am </S>  
<S> Do I like Henry </S>  
<S> Do I like college </S>  
<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

- 1.<s> do? Two gram
- 2.<s>I like Hendry?  
Two gram
- 3.<s> Do I like? Use  
trigram model
- 4.<s>Do I like  
college?  
Four grams

## Exercise2

---

Which of the following sentence is better. Find out using bigram model

Given Corpus

<S>I am Henry </S>  
<S>I like college </S>  
<S>Do Henry like college </S>  
<S>Henry I am </S>  
<S>Do I like Henry </S>  
<S>Do I like college </S>  
<S>I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

- 1.<S>I like college</S>
- 2.<S>Do I like Hendry</S>

# Thank You... 😊

- Q&A
- Suggestions / Feedback



# Natural Language Processing

## DSECL ZG565



**BITS** Pilani  
Pilani Campus

Dr.Vijayalakshmi Anand

BITS-Pilani



## **Session 2**

**Date – 9DEC 2023**

**Time – 1.40pm to 3.40pm**

These slides are prepared by the instructor, with grateful acknowledgement of James Allen and many others who made their course materials freely available online.

# Content Sequence Plan

- 
- M1 Natural Language Understanding and Generation
  - M2 N-gram Language Modelling
  - M7 Vector semantics and Embedding
  - M3 Neural networks and Neural language Models
  - M4 Part-of-Speech Tagging
  - M5 Hidden Markov Models and MEMM
  - M6 Topic Modelling
  - M8 Grammars and Parsing
  - M9 Statistical Constituency Parsing
  - M10 Dependency Parsing
  - M11 Encoder-Decoder Models, Attention and Contextual Embedding
  - M12 Word sense disambiguation
  - M13 Semantic web ontology and Knowledge Graph
  - M14 Introduction to NLP Applications
-

# Session contents

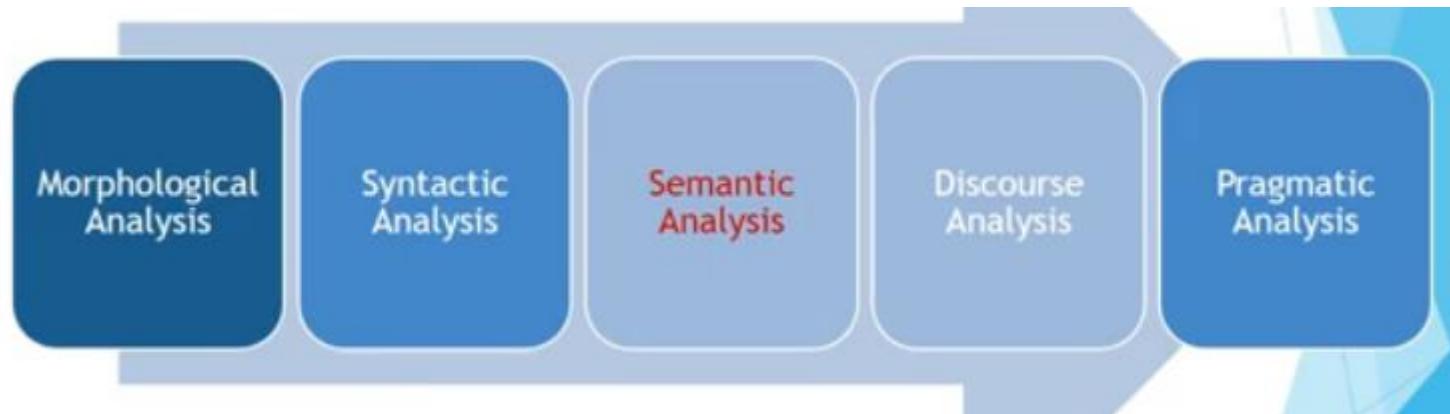
---

- **Lexical Semantics**
  - **Vector Semantics**
  - **Word embeddings**
    - Frequency based
    - Prediction based
-

---

# Lexical Semantics

# Stages in Natural language processing



# Semantic analysis



What is semantic analysis ?

Semantic analysis, in general, is a key method for assisting computers in comprehending the meaning of natural language text.

Why semantic analysis?

analyzes the grammatical format of sentences, including the arrangement of words, phrases, and clauses, to determine relationships between independent terms in a specific context

# Applications

## Question answering:

Q: “How **tall** is Mt. Everest?”

Ans: “The official **height** of Mount Everest is 29029 feet”

**“tall” is similar to “height”**

## Plagiarism detection

### MAINFRAMES

Mainframes are primarily referred to large computers with rapid, advanced processing capabilities that can execute and perform tasks equivalent to many Personal Computers (PCs) machines networked together. It is characterized with high quantity Random Access Memory (RAM), very large secondary storage devices, and high-speed processors to cater for the needs of the computers under its service.

Consisting of advanced components, mainframes have the capability of running multiple large applications required by many and most enterprises and organizations. This is one of its advantages. Mainframes are also suitable to cater for those applications (programs) or files that are of very high demand by its users (clients). Examples of such organizations and enterprises using mainframes are online shopping websites such as Flipkart, Amazon, and computing giant

### MAINFRAMES

Mainframes usually are referred to those computers with fast, advanced processing capabilities that could perform by itself tasks that may require a lot of Personal Computers (PC) Machines. Usually mainframes would have lots of RAMs, very large secondary storage devices, and very fast processors to cater for the needs of those computers under its service.

Due to the advanced components mainframes have, these computers have the capability of running multiple large applications required by most enterprises, which is one of its advantage. Mainframes are also suitable to cater for those applications or files that are of very large demand by its users (clients). Examples of these include the large online shopping websites -i.e.: Ebay, Amazon, Microsoft, etc.

# What do words mean?

---

- N-gram or text classification methods we've seen so far
  - Words are just strings (or indices  $w_i$  in a vocabulary list)
  - That's not very satisfactory!

# Lexical semantics

# Lemmas and senses



- A **sense** or “concept” is the meaning component of a word
- Lemmas can be **polysemous** (have multiple senses)

mouse (N) : **lemma**

1. any of numerous small rodents...  
2. a hand-operated device that controls a cursor...

**sense**

# Lexical semantics Relations

- Have relations with each other
  - Synonymy
  - Antonymy
  - Similarity
  - Relatedness: semantic field and frame
  - Connotation
- **More generally, a model of word meaning should allow us to draw inferences to address meaning-related tasks like question-answering or dialogue.**

# Synonyms

---

Word that have the same meaning in some or all contexts.

- filbert / hazelnut
- couch / sofa
- big / large
- automobile / car
- vomit / throw up
- Water / H<sub>2</sub>O

Two lexemes are synonyms if they can be successfully substituted for each other in all situations

# But

---

There are no examples of perfect synonymy

- Why should that be?
- Even if many aspects of meaning are identical
- Still may not preserve the acceptability based on notions of politeness, slang, register, genre, etc.

Example:

- Water and H<sub>2</sub>O

# Synonymy is a relation between senses rather than words

---

Consider the words *big* and *large*

Are they synonyms?

- How **big** is that plane?
- Would I be flying on a **large** or small plane?

How about here:

- Miss Nelson, for instance, became a kind of **big** sister to Benjamin.
- ?Miss Nelson, for instance, became a kind of **large** sister to Benjamin.

Why?

- *big* has a sense that means being older, or grown up
- *large* lacks this sense

# Antonyms

---

Senses that are opposites with respect to one feature of their meaning

Otherwise, they are very similar!

- dark / light
- short / long
- hot / cold
- up / down
- in / out

More formally: antonyms can

- define a binary opposition or at opposite ends of a scale (*long/short, fast/slow*)
- Be **reverses**: *rise/fall, up/down*

# Relation: Similarity

- Words with similar meanings.

word1	word2	similarity
vanish	disappear	9.8
behave	obey	7.3
belief	impression	5.95
muscle	bone	3.65
modest	flexible	0.98
hole	agreement	0.3

SimLex-999 dataset (Hill et al., 2015)

# Relation: Word relatedness

- Also called "word association"
- Words can be related in any way, perhaps via a **semantic frame or field**
  - coffee, tea: **similar**
  - coffee, cup: **related**, not similar

# Semantic field

---

- Words that
  - cover a particular semantic domain
  - bear structured relations with each other.
  - Useful for topic modelling

**Example**

**hospitals**

*surgeon, scalpel, nurse, anaesthetic, hospital*

**restaurants**

*waiter, menu, plate, food, menu, chef*

**houses**

*door, roof, kitchen, family, bed*

# Semantic frame

---

- A semantic frame is a set of words that denote perspectives or participants in a particular type of event.
  - E.g. verbs like buy (the event from the perspective of the buyer), sell (from the perspective of the seller), pay (focusing on the monetary aspect), or nouns like buyer.
  - Frames have semantic roles (like buyer, seller, goods, money), and words in a sentence can take on these roles.
  - A sentence like ***Sam bought the book from Ling*** could be paraphrased as ***Ling sold the book to Sam***

# Connotation (sentiment)

---

- Words have affective meanings
  - Positive connotations (happy)
  - Negative connotations (sad)
- Evaluation (sentiment!)
  - Positive evaluation (great, love)
  - Negative evaluation (terrible, hate)

# Connotation

Words seem to vary along 3 affective dimensions:

- **valence**: the pleasantness of the stimulus
- **arousal**: the intensity of emotion provoked by the stimulus (Ssang et al. 1957)
- **dominance**: the degree of control exerted by the stimulus

	Word	Score		Word	Score
Valence	love	1.000		toxic	0.008
	happy	1.000		nightmare	0.005
Arousal	elated	0.960		mellow	0.069
	frenzy	0.965		napping	0.046
Dominance	powerful	0.991		weak	0.045
	leadership	0.983		empty	0.081

Values from NRC VAD Lexicon (Mohammad 2018)

---

# Vector Semantics

# What does recent English borrowing *ongchoi* mean?

---

- Suppose you see these sentences:
  - Ong choi is delicious **sautéed with garlic**.
  - Ong choi is superb **over rice**
  - Ong choi **leaves** with salty sauces
- And you've also seen these:
  - ...spinach **sautéed with garlic over rice**
  - Chard stems and **leaves** are **delicious**
  - Collard greens and other **salty** leafy greens
- Conclusion:
  - Ongchoi is a leafy green like spinach, chard, or collard greens
  - We could conclude this based on words like "leaves" and "delicious" and "sautéed"

# Vector Semantics

Vector semantics is the standard way to represent word meaning in NLP, helping vector semantics us model many of the aspects of word meaning

# Word embeddings

- Vectors for representing words are called embeddings
- Each word = a vector (not just "good" or "worse")
- Defining meaning as a point in space based on distribution
- Similar words are "**nearby in semantic space**"



# Word embeddings: Why



Consider sentiment analysis:

- With **words**, a feature is a word identity
  - Feature 5: 'The previous word was "terrible"'
  - requires **exact same word** to be in training and test
- With **embeddings**:
  - Feature is a word vector
  - 'The previous word was vector [35,22,17...]'
  - Now in the test set we might see a similar vector [34,21,14]
  - We can generalize to **similar but unseen words!!!**

# Word embeddings: types

1. Frequency based Embedding
  - A common baseline model
  - **Sparse** long vectors
  - Words are represented by (a simple function of) the **counts** of nearby words
  - **dimensions** corresponding to **words** in the vocabulary or **documents** in a collection
  - **Count Vector**
  - **TF-IDF Vector**
  - **Co-Occurrence Vector**
2. Prediction based Embedding
  - **Dense** vectors
  - Representation is created by training a classifier to **predict** whether a word is likely to appear nearby
  - **Word2vec: Skip-gram and CBOW**
  - **GloVe**

---

# Frequency based Embedding

# Count vector

What is count vectorization?

Count vectorizer is a method to convert text to numerical data.

Example

```
text = ['Hello my name is james, this is my python notebook']
```

	hello	is	james	my	name	notebook	python	this
0	1	2	1	2	1	1	1	1

# Co-occurrence matrix

---

- We represent how often a word occurs in a document •
- Term-document matrix
  - Or how often a word occurs with another
- Term-term matrix
  - word-word co-occurrence matrix or word-context matrix.

# Term Document Matrix

Each cell: count of word w in a document d:

Each document is a count vector in  $\mathbb{N}^v$ : a column below

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

Two documents are similar if their vectors are similar

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

# The words in a term-document matrix



Each word is a count vector in  $\mathbb{N}^D$ : a row below

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

# The words in a term-document matrix



Two words are similar if their vectors are similar

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

# Term-context matrix for word similarity

- Two words are similar in meaning if their context vectors are similar
- The context could be the document, smaller contexts, generally a window around the word, for example of 4 words to the left and 4 words to the right

sugar, a sliced lemon, a tablespoonful of **apricot** preserve or jam, a pinch each of,

	aardvark	computer	data	pinch	result	sugar	...
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	

# Term frequency tf

---

- The term frequency  $tf_{t,d}$  of term  $t$  in document  $d$  is defined as the **number of times that  $t$  occurs in  $d$** .
- We want to use tf when computing query-document match scores.
- But how?
- Raw term frequency is not what we want because:
- A document with  $tf = 10$  occurrences of the term is more relevant than a document with  $tf = 1$  occurrence of the term.
- But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

# Frequency in document vs. frequency in collection



- In addition, to term frequency (the frequency of the term in the document) . . .
- . . . we also want to use the frequency of the term **in the collection** for weighting and ranking.

# Desired weight for rare terms

---

- Rare terms are more informative than frequent terms.
- Consider a term in the query that is **rare** in the collection (e.g., ARACHNOCENTRIC).
- A document containing this term is very likely to be relevant.
- → We want **high weights for rare terms** like ARACHNOCENTRIC.

# Desired weight for frequent terms

---

- Frequent terms are less informative than rare terms.
- Consider a term in the query that is **frequent** in the collection (e.g., GOOD, INCREASE, LINE).
- A document containing this term is more likely to be relevant than a document that doesn't . . .
- . . . but words like GOOD, INCREASE and LINE are not sure indicators of relevance.
- → **For frequent terms** like GOOD, INCREASE and LINE, we want positive weights . . .
- . . . but **lower weights** than for rare terms.

# Document frequency

---

- We want **high weights** for rare terms like ARACHNOCENTRIC.
- We want **low (positive) weights** for frequent words like GOOD, INCREASE and LINE.
- We will use **document frequency** to factor this into computing the matching score.
- The document frequency is **the number of documents in the collection that the term occurs in.**

# idf

## weight

- $df_t$  is the document frequency, the number of documents that  $t$  occurs in.
- $df_t$  is an inverse measure of the **informativeness** of term  $t$ .
- We define the **idf weight** of term  $t$  as follows:

$$idf_t = \log_{10} \frac{N}{df_t}$$

( $N$  is the number of documents in the collection.)

- $idf_t$  is a measure of the **informativeness** of the term.
- $[\log N/df_t]$  instead of  $[N/df_t]$  to “dampen” the effect of idf

# Examples for idf

- Compute  $\text{idf}_t$  using the formula:

$$\text{idf}_t = \log_{10} \frac{1,000,000}{\text{df}_t}$$

term	$\text{df}_t$	$\text{idf}_t$
calpurnia	1	6
animal	100	4
sunday	1000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

# Effect of idf on ranking

---

- idf affects the ranking of documents for **queries with at least two terms**.
- For example, in the query “arachnocentric line”, idf weighting **increases** the relative weight of ARACHNOCENTRIC and **decreases** the relative weight of LINE.
- idf has **little effect** on ranking for **one-term queries**.

# Collection frequency vs. Document frequency

word	collection frequency	document frequency
INSURANCE	10440	3997
TRY	10422	8760

- Collection frequency of  $t$ : number of tokens of  $t$  in the collection
- Document frequency of  $t$ : number of documents  $t$  occurs in
- Why these numbers?
- Which word is a better search term (and should get a higher weight)?
  - This example suggests that df (and idf) is better for weighting than cf (and “icf”).

# tf-idf

## weighting

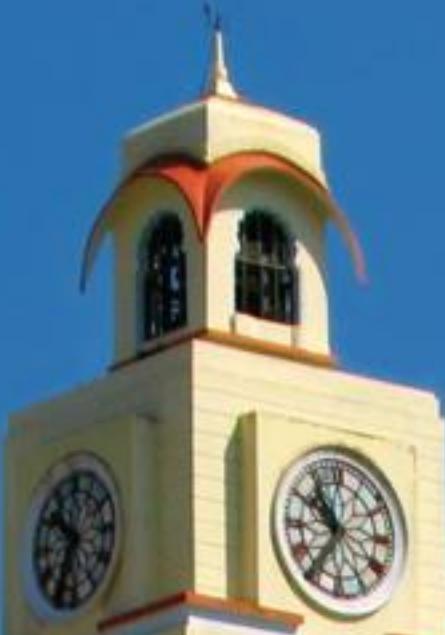
- The tf-idf weight of a term is the product of its **tf weight** and its **idf weight**.

$$w_{t,d} = (1 + \log \text{tf}_{t,d}) \cdot \log \frac{N}{\text{df}_t}$$

- tf-weight**
- idf-weight**
- Best known weighting scheme in information retrieval
- Note: the “-” in tf-idf is a hyphen, not a minus sign!
- Alternative names: tf.idf, tf x idf

# Thank You... 😊

- Q&A
- Suggestions / Feedback



# Natural Language Processing

## DSECL ZG565



**BITS** Pilani  
Pilani Campus

Dr.Vijayalakshmi Anand

BITS-Pilani



## **Session 2**

### **Date – 16DEC 2023**

### **Time – 1.40pm to 3.40pm**

These slides are prepared by the instructor, with grateful acknowledgement of James Allen and many others who made their course materials freely available online.

# Content Sequence Plan

- 
- M1 Natural Language Understanding and Generation
  - M2 N-gram Language Modelling
  - M7 Vector semantics and Embedding
  - M3 Neural networks and Neural language Models
  - M4 Part-of-Speech Tagging
  - M5 Hidden Markov Models and MEMM
  - M6 Topic Modelling
  - M8 Grammars and Parsing
  - M9 Statistical Constituency Parsing
  - M10 Dependency Parsing
  - M11 Encoder-Decoder Models, Attention and Contextual Embedding
  - M12 Word sense disambiguation
  - M13 Semantic web ontology and Knowledge Graph
  - M14 Introduction to NLP Applications
-



# Prediction based Embedding

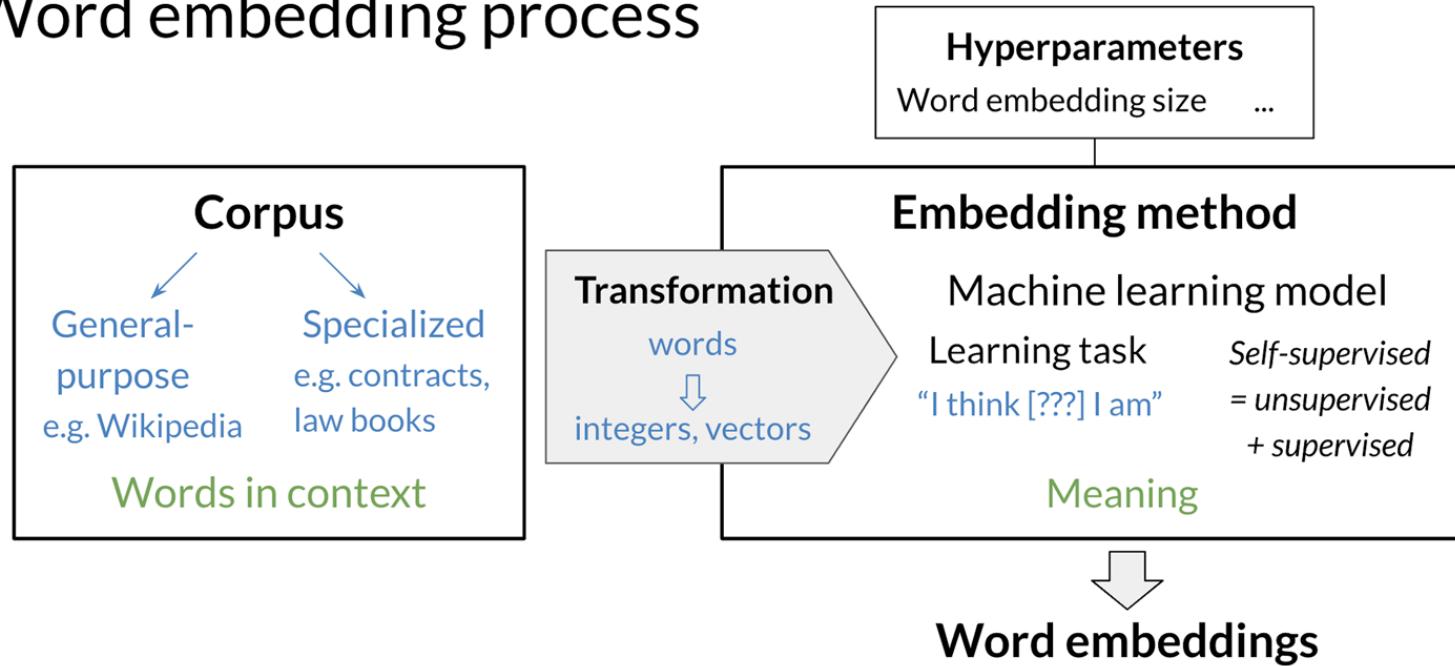
# Sparse versus dense vectors

---

- tf-idf (or PMI) vectors are
  - long** (length  $|V|= 20,000$  to  $50,000$ )
  - sparse** (most elements are zero)
- Alternative: learn vectors which are
  - short** (length  $50-1000$ )
  - dense** (most elements are non-zero)

# Word embeddings

## Word embedding process



# Common methods for getting short dense vectors

---

- “[Neural Language Model](#)”-inspired models
  - Word2vec (skipgram, CBOW), GloVe
- Singular Value Decomposition (SVD)
  - A special case of this is called LSA – Latent Semantic Analysis
- **Alternative to these "static embeddings":**
  - [Contextual Embeddings](#) (ELMo, BERT)
  - Compute distinct embeddings for a word in its context
  - Separate embeddings for each token of a word

# Word2vec -What is word2vec

---

- An unsupervised NLP method developed by Google in 2013 (Mikolov et al. 2013)
- Quantify the relationship between words
- Framework for learning word vectors Idea:
  - We have a large corpus (“body”) of text: a long list of words
  - Every word in a fixed vocabulary is represented by a vector
  - Go through each position  $t$  in the text, which has a center word  $c$  and context (“outside”) words  $o$
  - Use the similarity of the word vectors for  $c$  and  $o$  to calculate the probability of  $o$  given  $c$  (or vice versa)
  - Keep adjusting the word vectors to maximize this probability

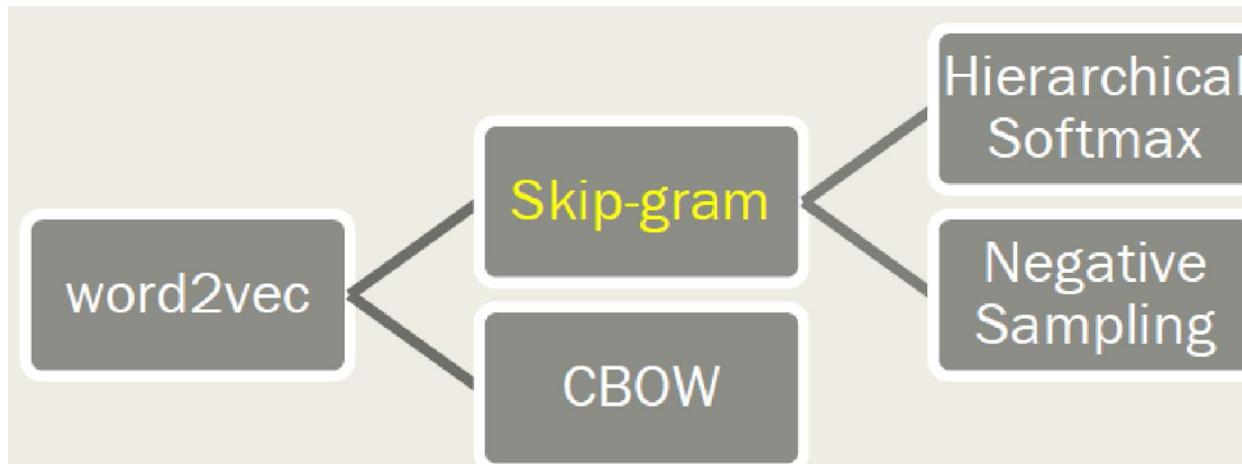
# Word2vec

---

- Instead of **counting** how often each word  $w$  occurs near "apricot"
  - Train a classifier on a binary **prediction** task:
    - Is  $w$  likely to show up near "apricot"?
- We don't actually care about this task
  - But we'll take the learned classifier weights as the word embeddings
- Big idea: **self-supervision**:
  - A word  $c$  that occurs near apricot in the corpus acts as the gold "correct answer" for supervised learning
  - No need for human labels

# Word2vec -Types of word2vec

---



# Basic word representation

\ V = [a, aaron, ..., zulu, <UNK>]

## 1-hot representation

Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
⋮	0	⋮	0	⋮	0
1	⋮	⋮	⋮	0	⋮
⋮	1	⋮	0	0	1
0	⋮	0	1	0	⋮
0	0	0	⋮	0	0

I want a glass of orange \_\_\_\_\_

I want a glass of apple \_\_\_\_\_.

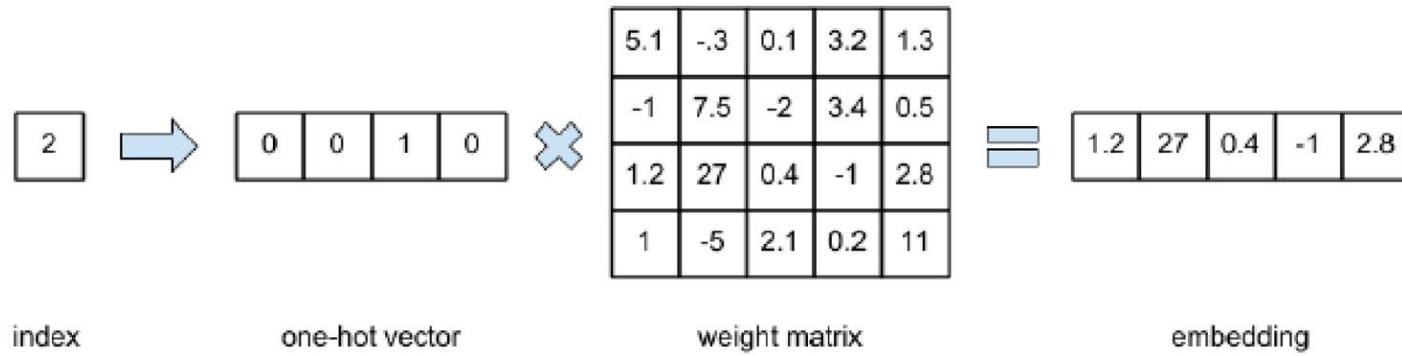
# Featurized representation: word embedding

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.70	0.69	0.03	-0.02
Food	0.09	0.01	0.02	0.01	0.95	0.97

I want a glass of orange

I want a glass of apple

# Word embeddings



# CBOW [Continuous bag of words]

---

## What is CBOW?

-predict a target word given the context words in a sentence

e.g The product **is** really good

# Working of CBOW with example

---

## Corpus

The product is really good

The product is wonderful

The product is awful

### Step1:

combine the sentences

The product is really good The product is wonderful The product is awful

### Step2:

Select window size

# Contd..

The product is really good The product is wonderful The product is awful



**Window Size**

**1**

The product is really good The product is wonderful The product is awful



**Window Size**

**2**

# Contd..

## Step3: Get one hot encoding for each word

1	The	1	0	0	0	0	0	0
2	product	0	1	0	0	0	0	0
3	is	0	0	1	0	0	0	0
4	really	0	0	0	1	0	0	0
5	wonderful	0	0	0	0	1	0	0
6	good	0	0	0	0	0	1	0
7	awful	0	0	0	0	0	0	1

## Step4:Training data

Context Words	Target Word
The,is	Product
product,really	is
Is,good	really
Good,product	the

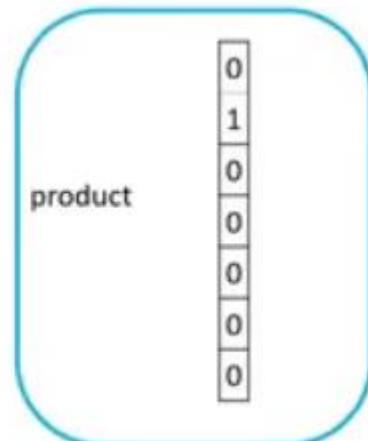
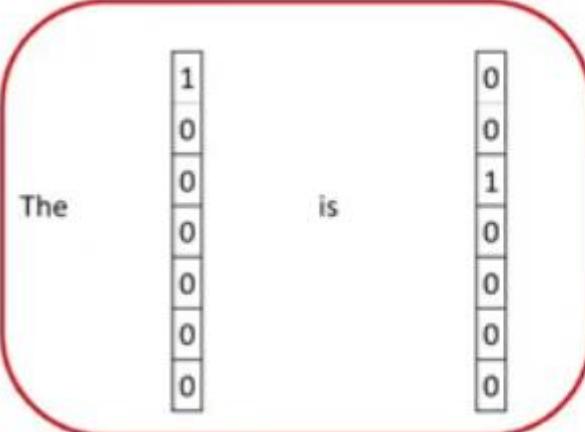
# Contd..

**One Hot Encoding**

1	The	1	0	0	0	0	0	0
2	product	0	1	0	0	0	0	0
3	is	0	0	1	0	0	0	0
4	really	0	0	0	1	0	0	0
5	wonderful	0	0	0	0	1	0	0
6	good	0	0	0	0	0	1	0
7	awful	0	0	0	0	0	0	1

**Training Data**

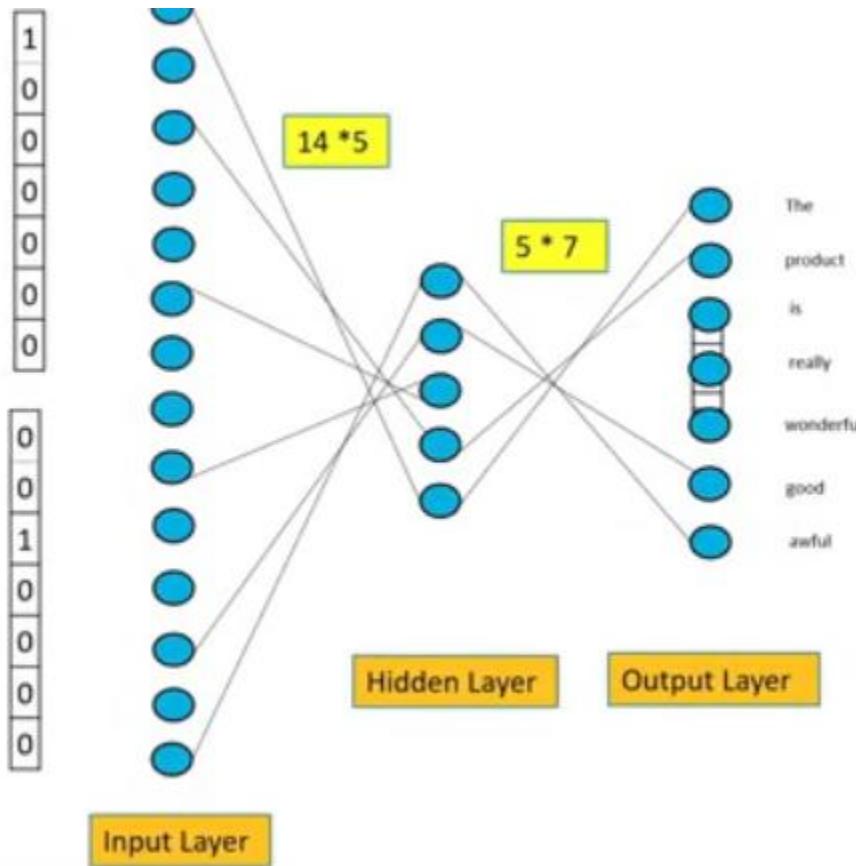
Context Words	Target Word
The,is	product
product,really	is
Is,good	really
Good,product	the



# Contd..

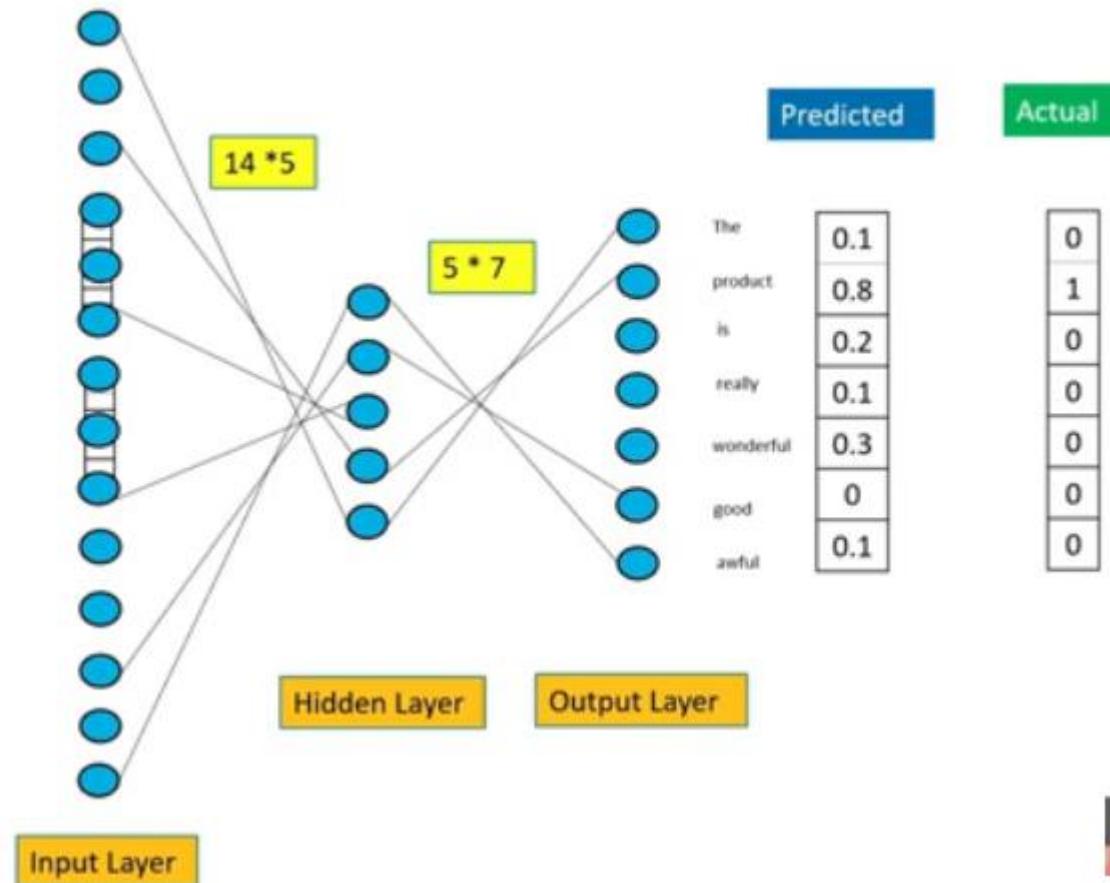
One hot encoding of - The

One hot encoding of - is



# Cond..

One hot encoding of - The



One hot encoding of - is

# Word Embedding matrix

5 \* 7

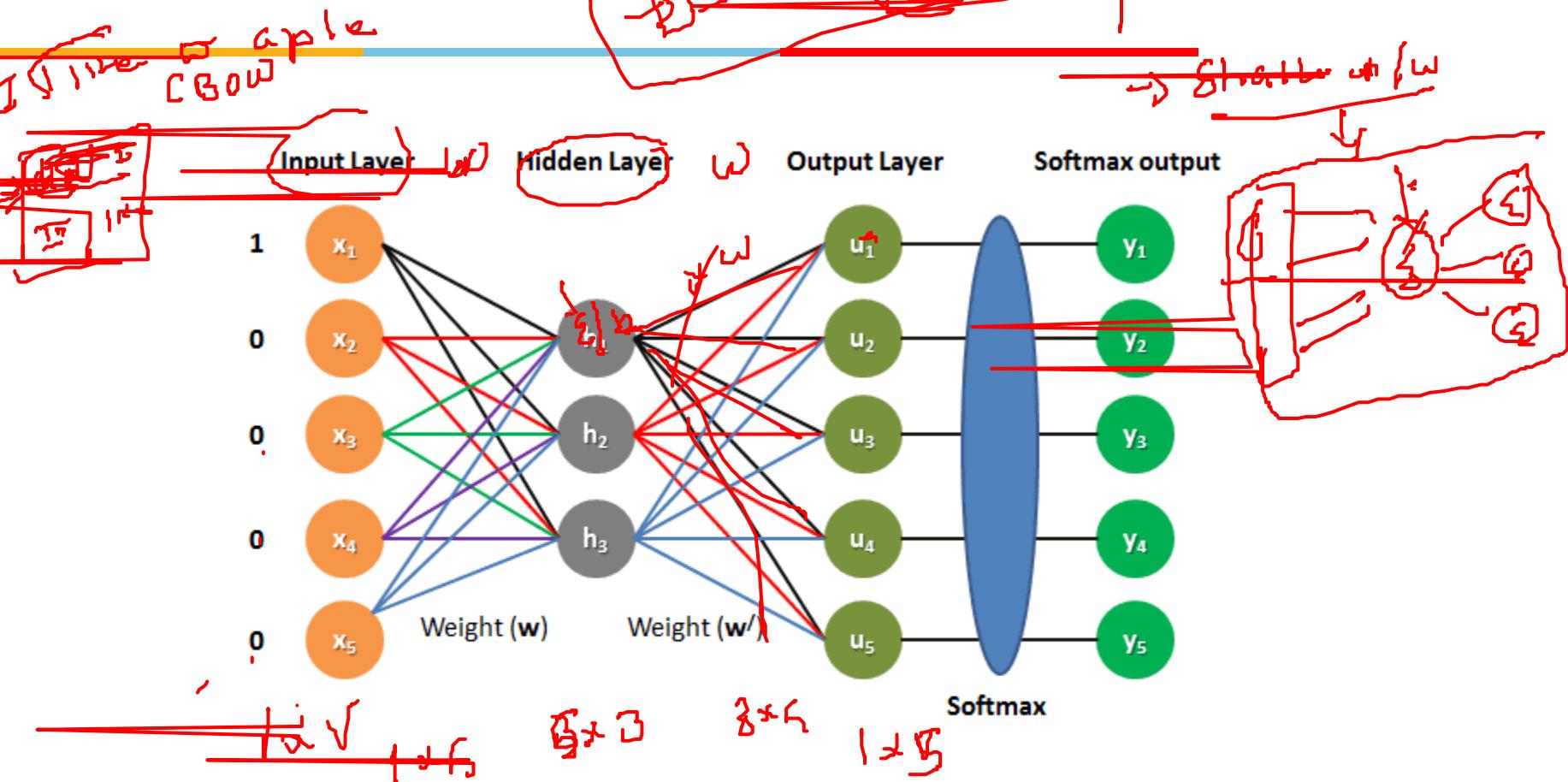
0.12	0.56	0.23	0.67	0.87	0.9	0.56
0.23	0.45	0.98	0.76	0.98	0.56	0.94
0.45	0.78	0.87	0.62	0.13	0.78	0.1
0.91	0.6	0.24	0.84	0.45	0.67	0.34
0.12	0.87	0.34	0.67	0.78	0.34	0.86

# Training model

---

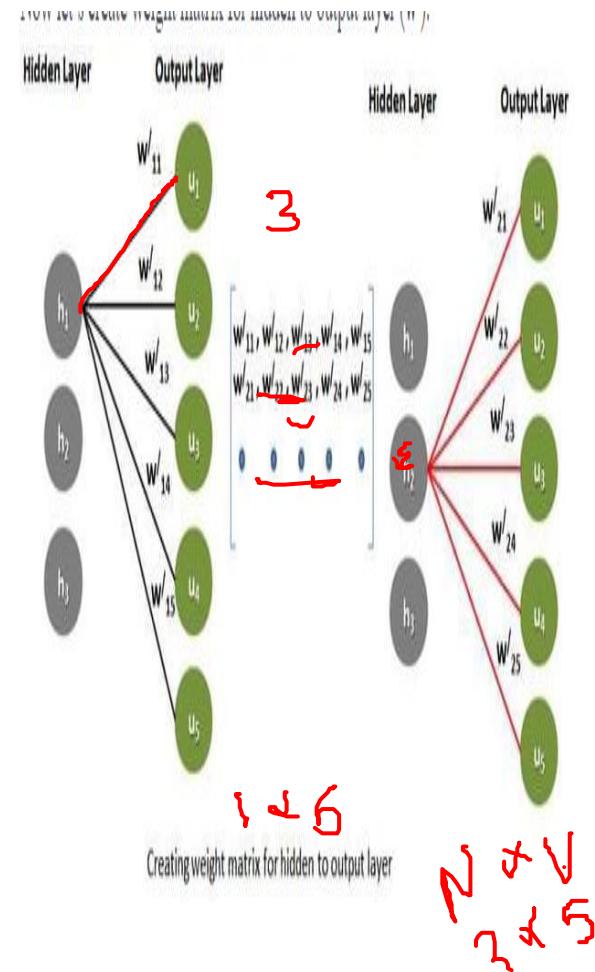
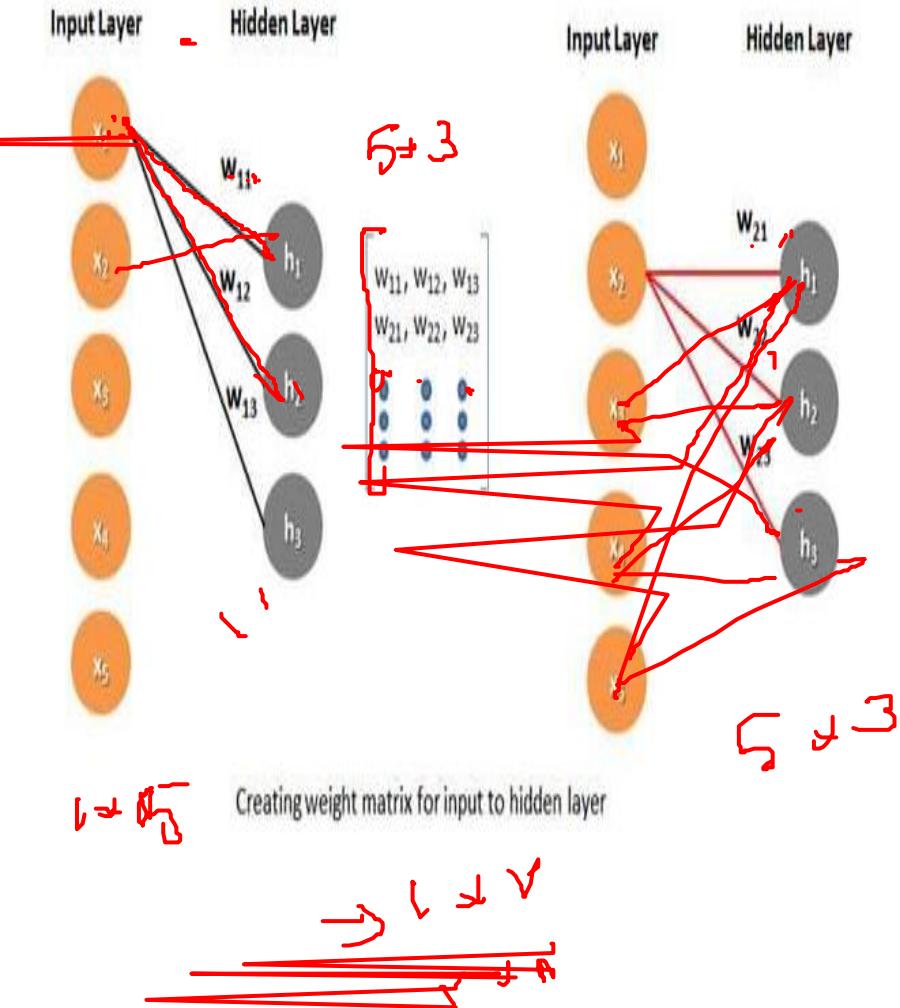
1. Create model Architecture
  2. Forward Propagation
  3. Error Calculation
  4. Weight tuning using backward pass
-

# Model architecture

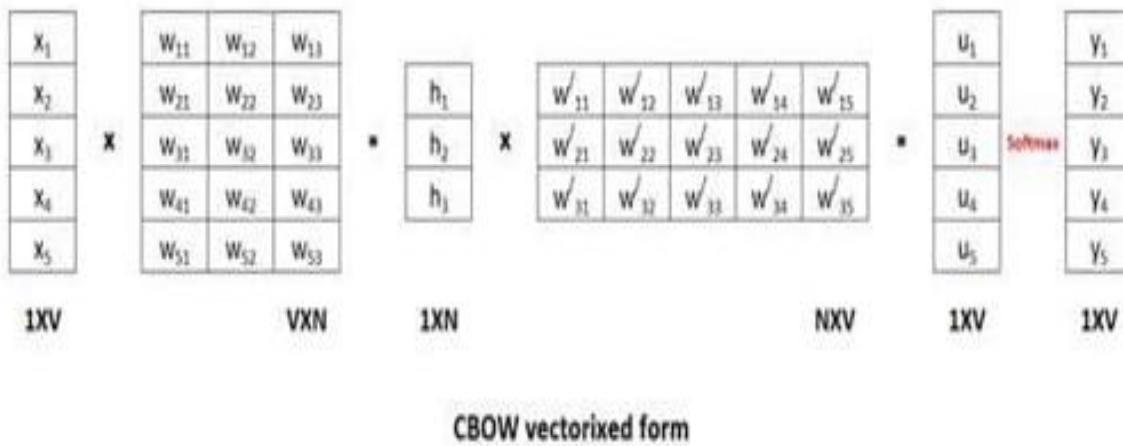


First training data point: The context word is "I" and the target word is "like".

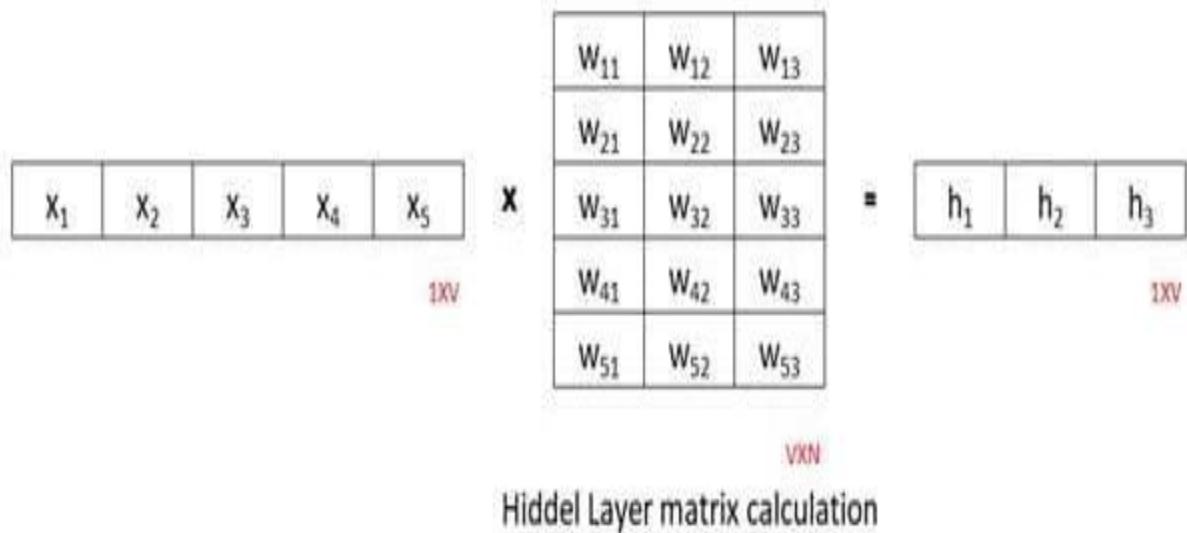
# Weighed matrix



### **CBOW Vectorized form:**



# Forward Propagation



$$h_1 = w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + w_{41}x_4 + w_{51}x_5$$

$$h_2 = w_{12}x_1 + w_{22}x_2 + w_{32}x_3 + w_{42}x_4 + w_{52}x_5$$

$$h_3 = w_{13}x_1 + w_{23}x_2 + w_{33}x_3 + w_{43}x_4 + w_{53}x_5$$

⋮ ⋮ ⋮

# Contd..

**Calculate output layer matrix (u):**

$$\begin{array}{|c|c|c|} \hline h_1 & h_2 & h_3 \\ \hline \end{array} \times \begin{array}{|c|c|c|c|c|} \hline w'_{11} & w'_{12} & w'_{13} & w'_{14} & w'_{15} \\ \hline w'_{21} & w'_{22} & w'_{23} & w'_{24} & w'_{25} \\ \hline w'_{31} & w'_{32} & w'_{33} & w'_{34} & w'_{35} \\ \hline \end{array} = \begin{array}{|c|c|c|c|c|} \hline u_1 & u_2 & u_3 & u_4 & u_5 \\ \hline \end{array}$$

Output Layer matrix calculation

So now:

$$u_1 = w'_{11}h_1 + w'_{21}h_2 + w'_{31}h_3$$

$$u_2 = w'_{12}h_1 + w'_{22}h_2 + w'_{32}h_3$$

$$u_3 = w'_{13}h_1 + w'_{23}h_2 + w'_{33}h_3$$

$$u_4 = w'_{14}h_1 + w'_{24}h_2 + w'_{34}h_3$$

$$u_5 = w'_{15}h_1 + w'_{25}h_2 + w'_{35}h_3$$

# contd

Calculate final Softmax output (y):

$u_1$	$y_1$
$u_2$	$y_2$
$u_3$	$y_3$
$u_4$	$y_4$
$u_5$	$y_5$

Softmax

**1XV**

**1XV**

$$y_1 = \text{Softmax}(u_1)$$

$$y_2 = \text{Softmax}(u_2)$$

$$y_3 = \text{Softmax}(u_3)$$

$$y_4 = \text{Softmax}(u_4)$$

$$y_5 = \text{Softmax}(u_5)$$

$$y_1 = \frac{e^{u_1}}{(e^{u_1} + e^{u_2} + e^{u_3} + e^{u_4} + e^{u_5})}$$

...    ...    ...    ...    ...    ...

$$y_j = \frac{e^j}{\sum_{j=1}^V e^j}$$

# Error calculation – log loss function

$$E = -\log(P(w_t|w_c))$$

$w_t$  = Target word

$w_c$  = Context word

out out

I

~~1, 2, 3~~

$$E(y_2) = -\log(w_{y_2}|w_{x_1})$$

$$= -\log \frac{e^{u_2}}{e^{u_1} + e^{u_2} + e^{u_3} + e^{u_4} + e^{u_5}}$$

$$= -\log(e^{u_2}) + \log(e^{u_1} + e^{u_2} + e^{u_3} + e^{u_4} + e^{u_5})$$

$$= -u_2 + \log(e^{u_1} + e^{u_2} + e^{u_3} + e^{u_4} + e^{u_5})$$

$$E = -u_{j^*} + \log \sum_{j=1}^V e^{u_j}$$

# Back propagation

Step1: Gradient of E with respect to  $w'_{11}$ :

$$\begin{bmatrix} w'_{11} & w'_{12} & w'_{13} & w'_{14} & w'_{15} \\ w'_{21} & w'_{22} & w'_{23} & w'_{24} & w'_{25} \\ w'_{31} & w'_{32} & w'_{33} & w'_{34} & w'_{35} \end{bmatrix}$$

$$\begin{array}{c} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{array} \xrightarrow{\text{softmax}} \begin{array}{c} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{array}$$

$$\frac{dE(y_1)}{dw'_{11}} = \frac{dE(y_1)}{du_1} \cdot \frac{du_1}{dw'_{11}}$$

Now as we know

$$E(y_1) = -u_1 + \log(e^{u_1} + e^{u_2} + e^{u_3} + e^{u_4} + e^{u_5})$$

$$So, \frac{dE(y_1)}{du_1} = -\frac{du_1}{du_1} + \frac{d[\log(e^{u_1} + e^{u_2} + e^{u_3} + e^{u_4} + e^{u_5})]}{du_1}$$

Derivative of log function with chain rule.

$$\begin{aligned}&= -1 + \frac{1}{e^{u_1} + e^{u_2} + e^{u_3} + e^{u_4} + e^{u_5}} * u_1 \\&= -1 + \frac{u_1}{e^{u_1} + e^{u_2} + e^{u_3} + e^{u_4} + e^{u_5}} \\&= -1 + y_1\end{aligned}$$

# Contd..

Taking derivative of E with respect to  $u_j$ :

$$\begin{aligned}\frac{dE}{du_j} &= -\frac{d(u_{j*})}{du_j} + \frac{d(\log \sum_{j=1}^V e^{u_j})}{du_j} \\ &= (-t_j + y_j)\end{aligned}$$

$$= (y_j - t_j)$$

$$= \textcircled{e_j}$$

Note:  $t_j = 1$  if  $t_j = t_{j*}$  else  $t_j = 0$

# Contd..

So for first iteration,

$$\frac{dE(y_1)}{du_1} = e_1$$

And,  $\frac{du_1}{dw'_{11}} = \frac{d(w'_{11}h_1 + w'_{21}h_2 + w'_{31}h_3)}{dw'_{11}} = h_1$

Now finally coming back to main derivative which we were trying to solve:

$$\frac{dE(y_1)}{dw'_{11}} = \frac{dE(y_1)}{du_1} \cdot \frac{du_1}{dw'_{11}} = e_1 h_1$$

So generalized form will looks like below:

$$\frac{dE}{dw'} = e * h$$

# Step2: Updating the weight of $w'_{11}$ :

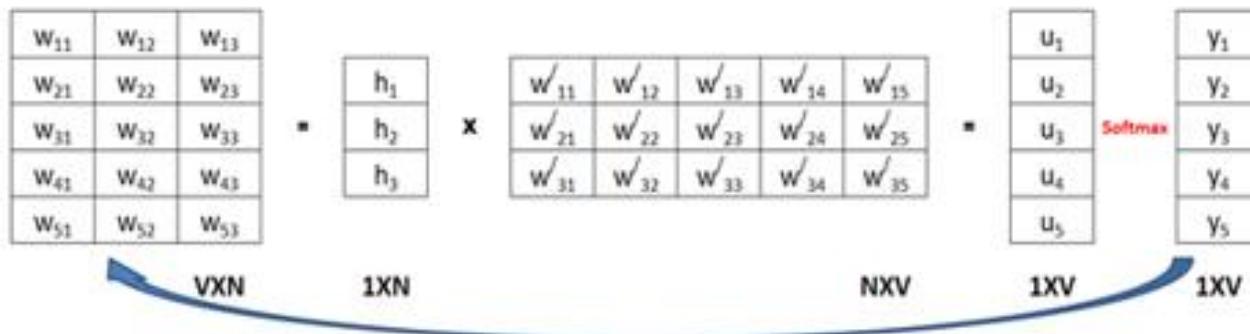
---

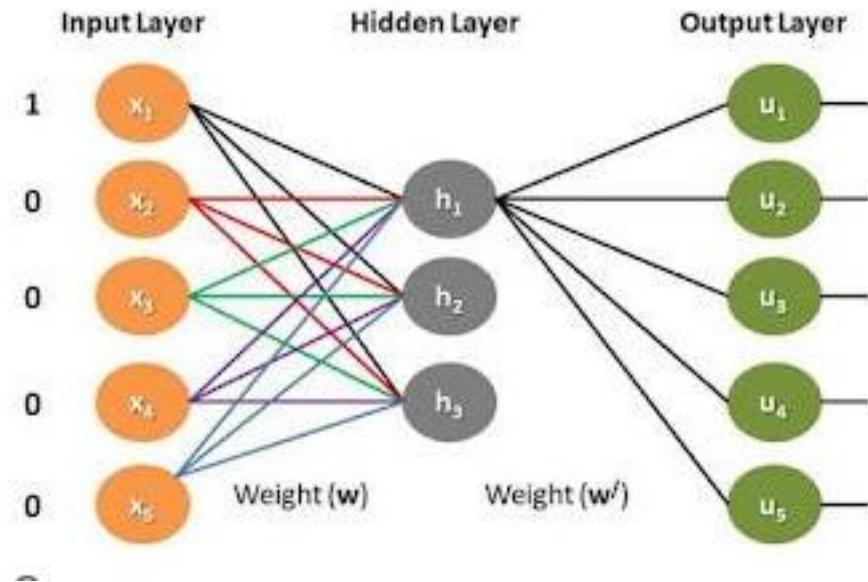
Step2: Updating the weight of  $w'_{11}$ :

$$new(w'_{11}) = w'_{11} - \frac{dE(y_1)}{w'_{11}} = (w'_{11} - e_1 h_1)$$

# Now Updating the first weight

Step1: Gradient of E with respect to  $w_{11}$ :





$$\begin{aligned}
 \frac{dE}{dh_1} &= \left( \frac{dE}{du_1}, \frac{du_1}{dh_1} \right) + \left( \frac{dE}{du_2}, \frac{du_2}{dh_1} \right) + \left( \frac{dE}{du_3}, \frac{du_3}{dh_1} \right) + \left( \frac{dE}{du_4}, \frac{du_4}{dh_1} \right) + \left( \frac{dE}{du_5}, \frac{du_5}{dh_1} \right) \\
 &= ew'_{11} + ew'_{12} + ew'_{13} + ew'_{14} + ew'_{15}
 \end{aligned}$$

As for  $u_1$  and  $h_1$ ,

$$\frac{du_1}{dh_1} = \frac{d(w'_{11}h_1 + w'_{21}h_2 + w'_{31}h_3)}{dh_1} = w'_{11}$$

*As,  $h_2$  and  $h_3$  are constant with respect to  $h_1$*

*Similarly we can calculate :  $\frac{du_2}{dh_1}, \frac{du_3}{dh_1}, \frac{du_4}{dh_1}, \frac{du_5}{dh_1}$*

$$And, \frac{dh_1}{dw_{11}} = \frac{d(w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + w_{41}x_4 + w_{51}x_5)}{dw_{11}}$$

# Contd..

$$\frac{dE}{dw_{11}} = \frac{dE}{dh_1}, \frac{dh_1}{dw_{11}}$$

$$= (ew'_{11} + ew'_{12} + ew'_{13} + ew'_{14} + ew'_{15}) * x$$

Step2: Updating the weight of  $w_{11}$ :

$$new(w_{11}) = w_{11} - \frac{dE}{dw_{11}}$$

$$= w_{11} - (ew'_{11} + ew'_{12} + ew'_{13} + ew'_{14} + ew'_{15}) * x$$

# Skip gram

**Step - 1**      The product is really good The product is wonderful The product is awful

**Step - 2**      Window Size      1

**Step - 3**      Get one hot encoding for each word

# Contd..

1	The	1	0	0	0	0	0	0
2	product	0	1	0	0	0	0	0
3	is	0	0	1	0	0	0	0
4	really	0	0	0	1	0	0	0
5	wonderful	0	0	0	0	1	0	0
6	good	0	0	0	0	0	1	0
7	awful	0	0	0	0	0	0	1

## Step - 1

The product is really good The product is wonderful The product is awful

## Step - 4

Input Words	Target Word
product	The
product	is
is	product
is	really

### One Hot Encoding

1	The	1	0	0	0	0	0	0
2	product	0	1	0	0	0	0	0
3	is	0	0	1	0	0	0	0
4	really	0	0	0	1	0	0	0
5	wonderful	0	0	0	0	1	0	0
6	good	0	0	0	0	0	1	0
7	awful	0	0	0	0	0	0	1

### Training Data

Context Words	Target Word
The,is	Product
product,really	is
Is,good	really
Good,product	the

product

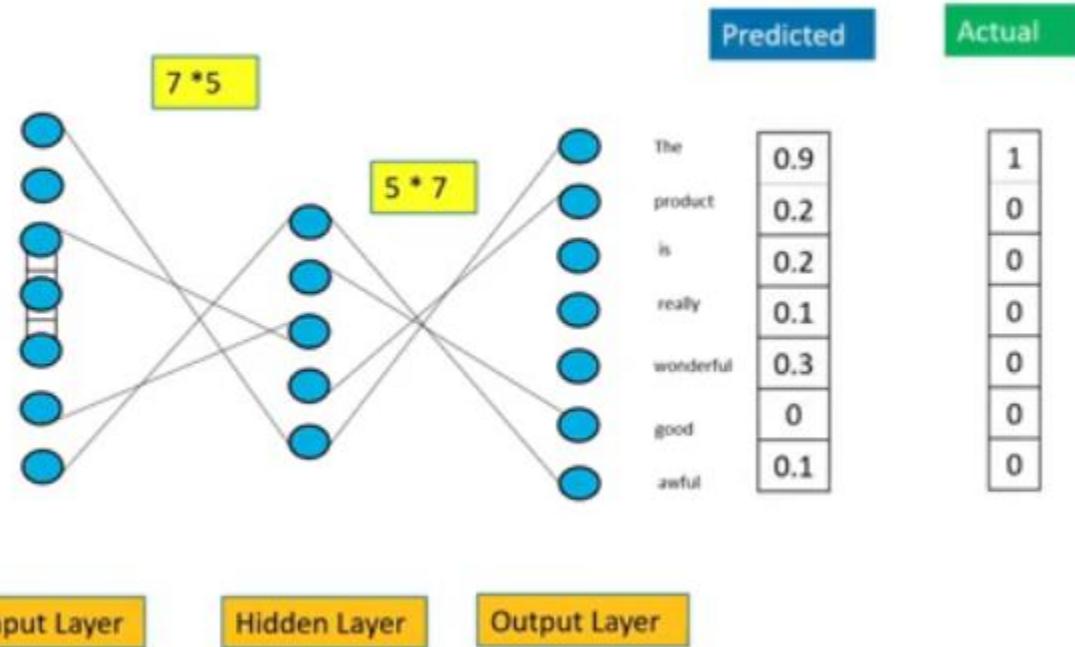
0
1
0
0
0
0
0

The

1
0
0
0
0
0
0

# Contd..

One hot encoding of - Product



# Thank You... 😊

- Q&A
- Suggestions / Feedback



# Natural Language Processing

## DSECL ZG565



**BITS** Pilani  
Pilani Campus

Dr.Vijayalakshmi Anand

BITS-Pilani



## **Session 2**

### **Date – 16DEC 2023**

### **Time – 1.40pm to 3.40pm**

These slides are prepared by the instructor, with grateful acknowledgement of James Allen and many others who made their course materials freely available online.

# Content Sequence Plan

- 
- M1 Natural Language Understanding and Generation
  - M2 N-gram Language Modelling
  - M7 Vector semantics and Embedding
  - M3 Neural networks and Neural language Models
  - M4 Part-of-Speech Tagging
  - M5 Hidden Markov Models and MEMM
  - M6 Topic Modelling
  - M8 Grammars and Parsing
  - M9 Statistical Constituency Parsing
  - M10 Dependency Parsing
  - M11 Encoder-Decoder Models, Attention and Contextual Embedding
  - M12 Word sense disambiguation
  - M13 Semantic web ontology and Knowledge Graph
  - M14 Introduction to NLP Applications
-



# Prediction based Embedding

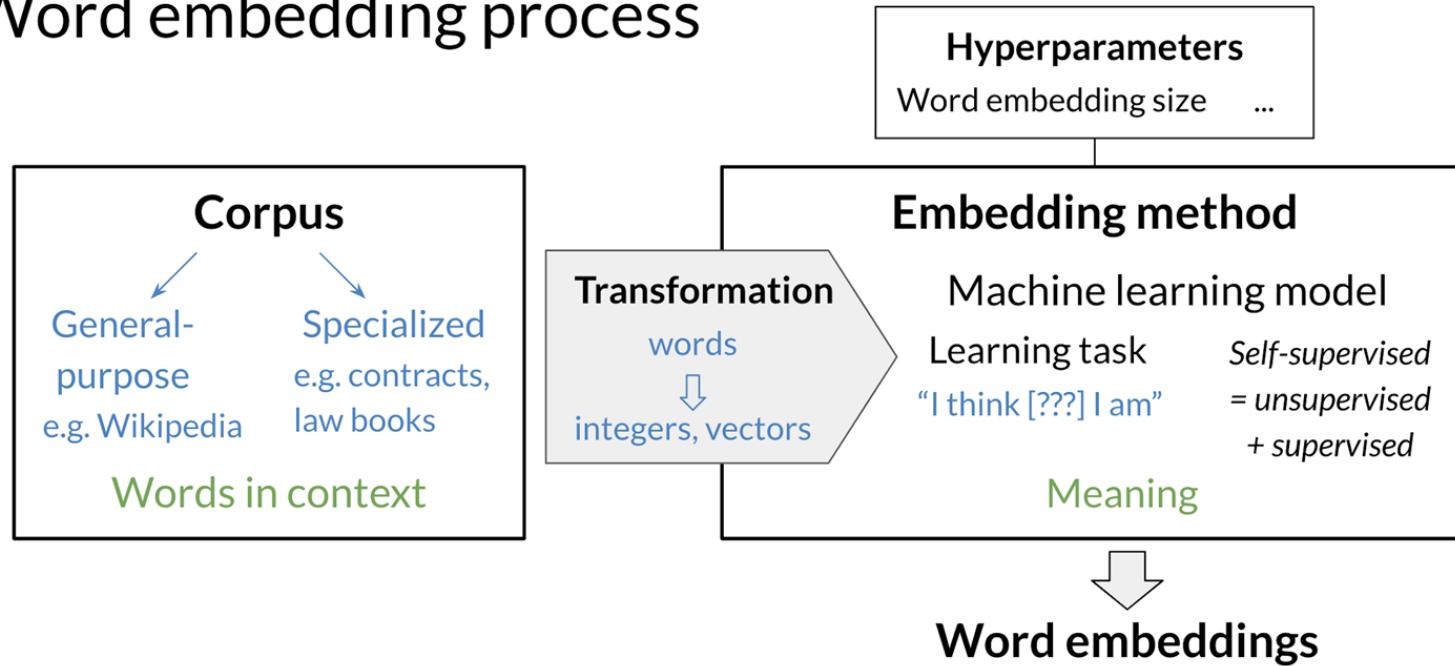
# Sparse versus dense vectors

---

- tf-idf (or PMI) vectors are
  - long** (length  $|V|= 20,000$  to  $50,000$ )
  - sparse** (most elements are zero)
- Alternative: learn vectors which are
  - short** (length  $50-1000$ )
  - dense** (most elements are non-zero)

# Word embeddings

## Word embedding process



# Common methods for getting short dense vectors

---

- “[Neural Language Model](#)”-inspired models
  - Word2vec (skipgram, CBOW), GloVe
- Singular Value Decomposition (SVD)
  - A special case of this is called LSA – Latent Semantic Analysis
- **Alternative to these "static embeddings":**
  - [Contextual Embeddings](#) (ELMo, BERT)
  - Compute distinct embeddings for a word in its context
  - Separate embeddings for each token of a word

# Word2vec -What is word2vec

---

- An unsupervised NLP method developed by Google in 2013 (Mikolov et al. 2013)
- Quantify the relationship between words
- Framework for learning word vectors Idea:
  - We have a large corpus (“body”) of text: a long list of words
  - Every word in a fixed vocabulary is represented by a vector
  - Go through each position  $t$  in the text, which has a center word  $c$  and context (“outside”) words  $o$
  - Use the similarity of the word vectors for  $c$  and  $o$  to calculate the probability of  $o$  given  $c$  (or vice versa)
  - Keep adjusting the word vectors to maximize this probability

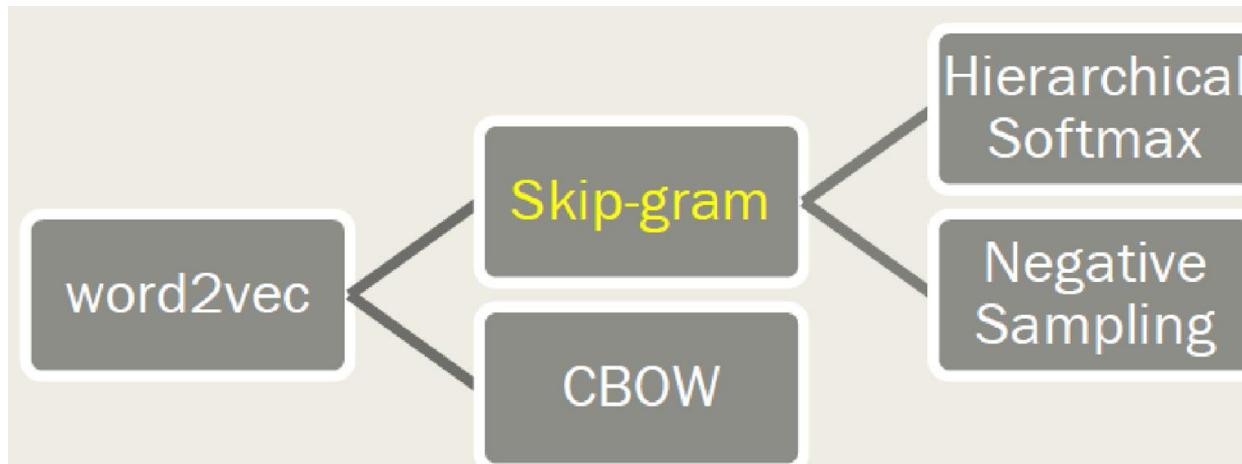
# Word2vec

---

- Instead of **counting** how often each word  $w$  occurs near "apricot"
  - Train a classifier on a binary **prediction** task:
    - Is  $w$  likely to show up near "apricot"?
- We don't actually care about this task
  - But we'll take the learned classifier weights as the word embeddings
- Big idea: **self-supervision**:
  - A word  $c$  that occurs near apricot in the corpus acts as the gold "correct answer" for supervised learning
  - No need for human labels

# Word2vec -Types of word2vec

---



# Basic word representation

\ V = [a, aaron, ..., zulu, <UNK>]

## 1-hot representation

Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
⋮	0	⋮	0	⋮	0
1	⋮	⋮	⋮	0	⋮
⋮	1	⋮	0	0	1
0	⋮	0	1	0	⋮
0	0	0	⋮	0	0

I want a glass of orange \_\_\_\_\_

I want a glass of apple \_\_\_\_\_.

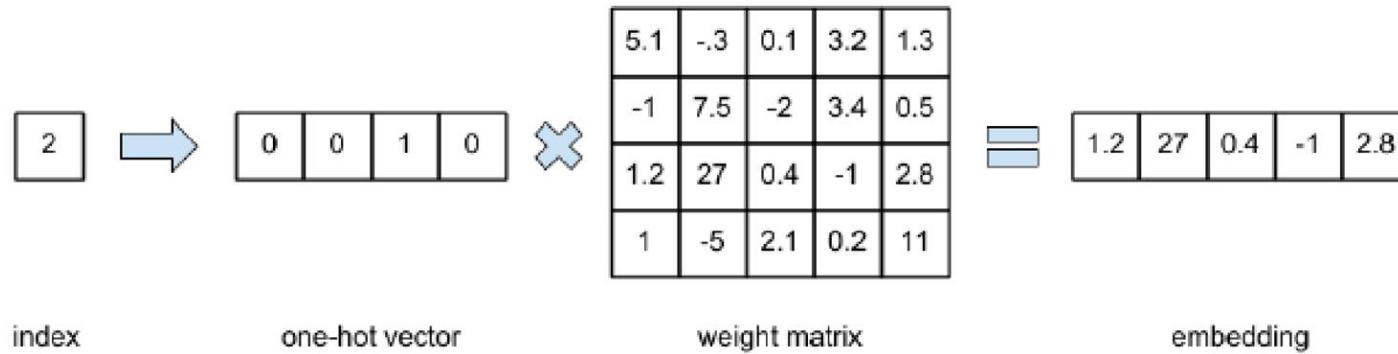
# Featurized representation: word embedding

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.70	0.69	0.03	-0.02
Food	0.09	0.01	0.02	0.01	0.95	0.97

I want a glass of orange

I want a glass of apple

# Word embeddings



# CBOW [Continuous bag of words]

---

## What is CBOW?

-predict a target word given the context words in a sentence

e.g The product **is** really good

# Working of CBOW with example

---

## Corpus

The product is really good

The product is wonderful

The product is awful

### Step1:

combine the sentences

The product is really good The product is wonderful The product is awful

### Step2:

Select window size

# Contd..

The product is really good The product is wonderful The product is awful



**Window Size**

**1**

The product is really good The product is wonderful The product is awful



**Window Size**

**2**

# Contd..

## Step3: Get one hot encoding for each word

1	The	1	0	0	0	0	0	0
2	product	0	1	0	0	0	0	0
3	is	0	0	1	0	0	0	0
4	really	0	0	0	1	0	0	0
5	wonderful	0	0	0	0	1	0	0
6	good	0	0	0	0	0	1	0
7	awful	0	0	0	0	0	0	1

## Step4:Training data

Context Words	Target Word
The,is	Product
product,really	is
Is,good	really
Good,product	the

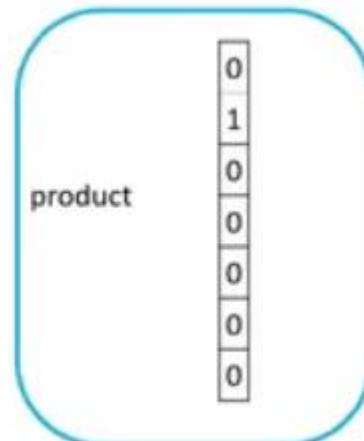
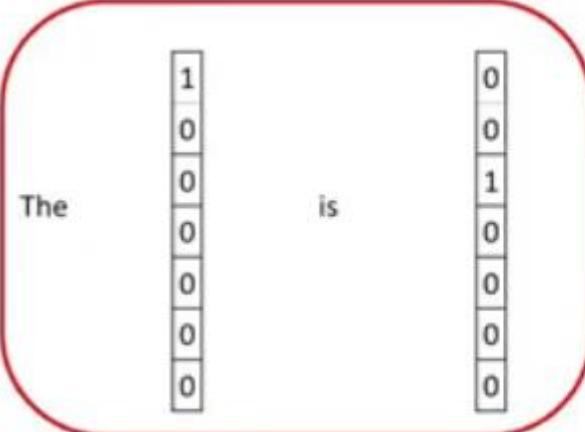
# Contd..

**One Hot Encoding**

1	The	1	0	0	0	0	0	0
2	product	0	1	0	0	0	0	0
3	is	0	0	1	0	0	0	0
4	really	0	0	0	1	0	0	0
5	wonderful	0	0	0	0	1	0	0
6	good	0	0	0	0	0	1	0
7	awful	0	0	0	0	0	0	1

**Training Data**

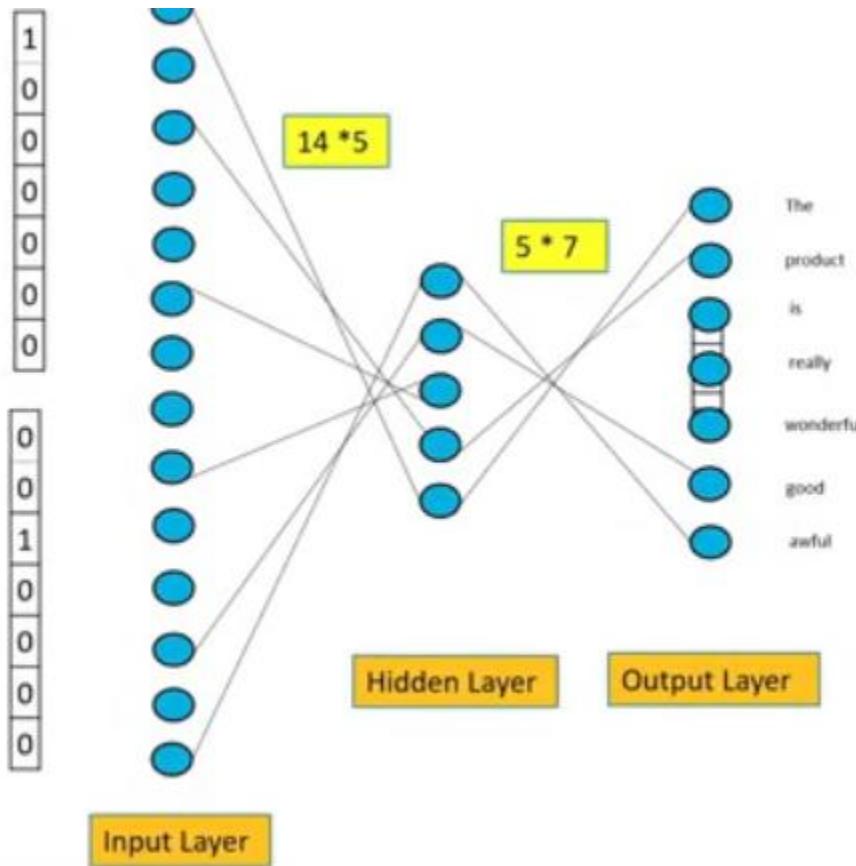
Context Words	Target Word
The,is	product
product,really	is
Is,good	really
Good,product	the



# Contd..

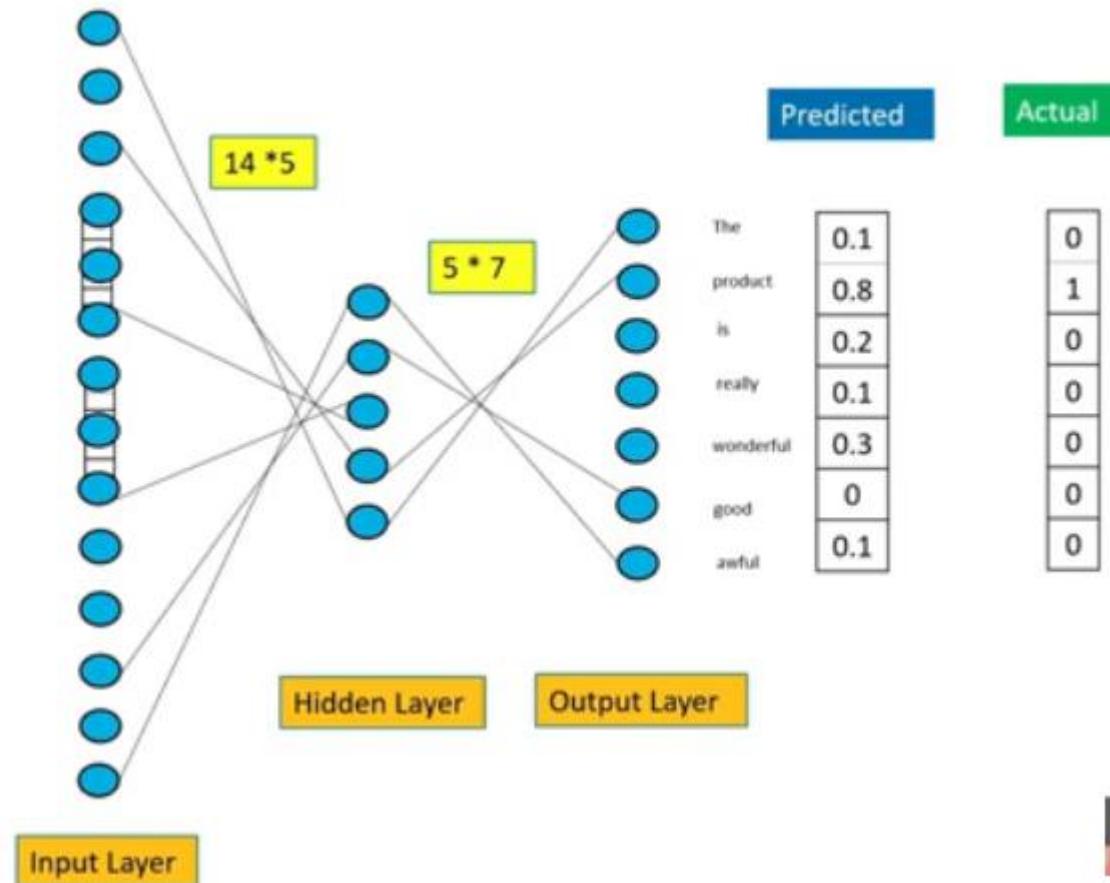
One hot encoding of - The

One hot encoding of - is



# Cond..

One hot encoding of - The



One hot encoding of - is

# Word Embedding matrix

5 \* 7

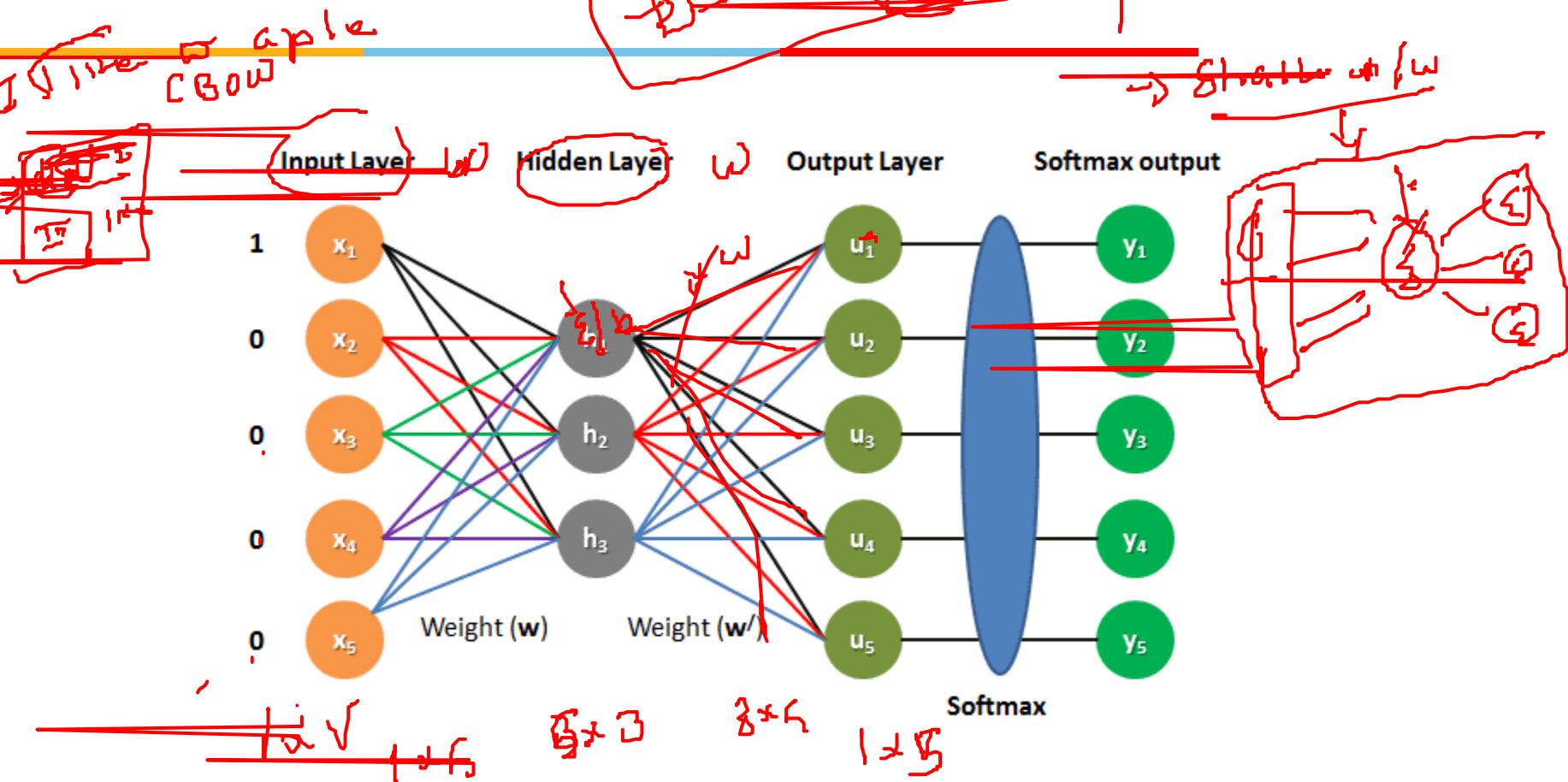
0.12	0.56	0.23	0.67	0.87	0.9	0.56
0.23	0.45	0.98	0.76	0.98	0.56	0.94
0.45	0.78	0.87	0.62	0.13	0.78	0.1
0.91	0.6	0.24	0.84	0.45	0.67	0.34
0.12	0.87	0.34	0.67	0.78	0.34	0.86

# Training model

---

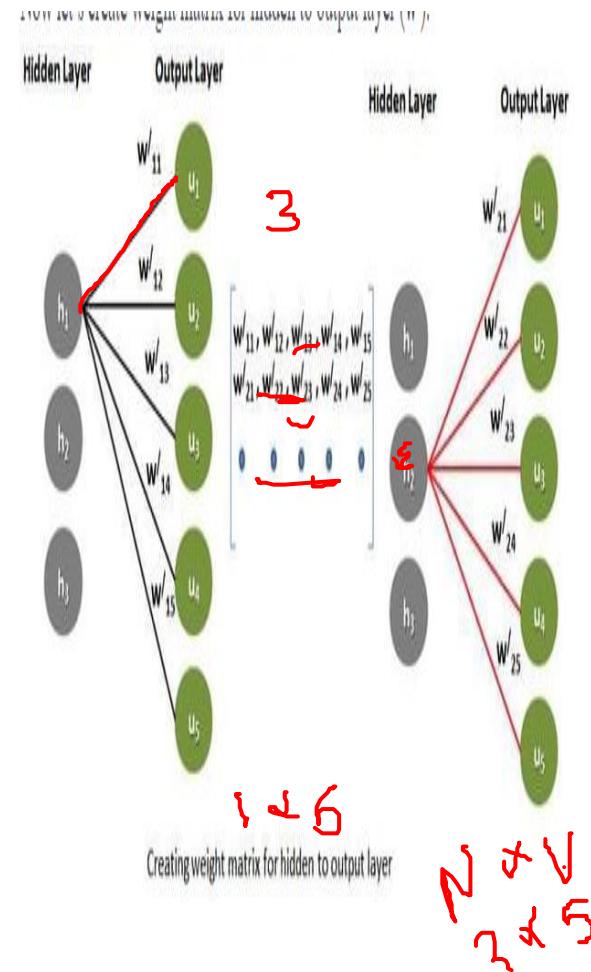
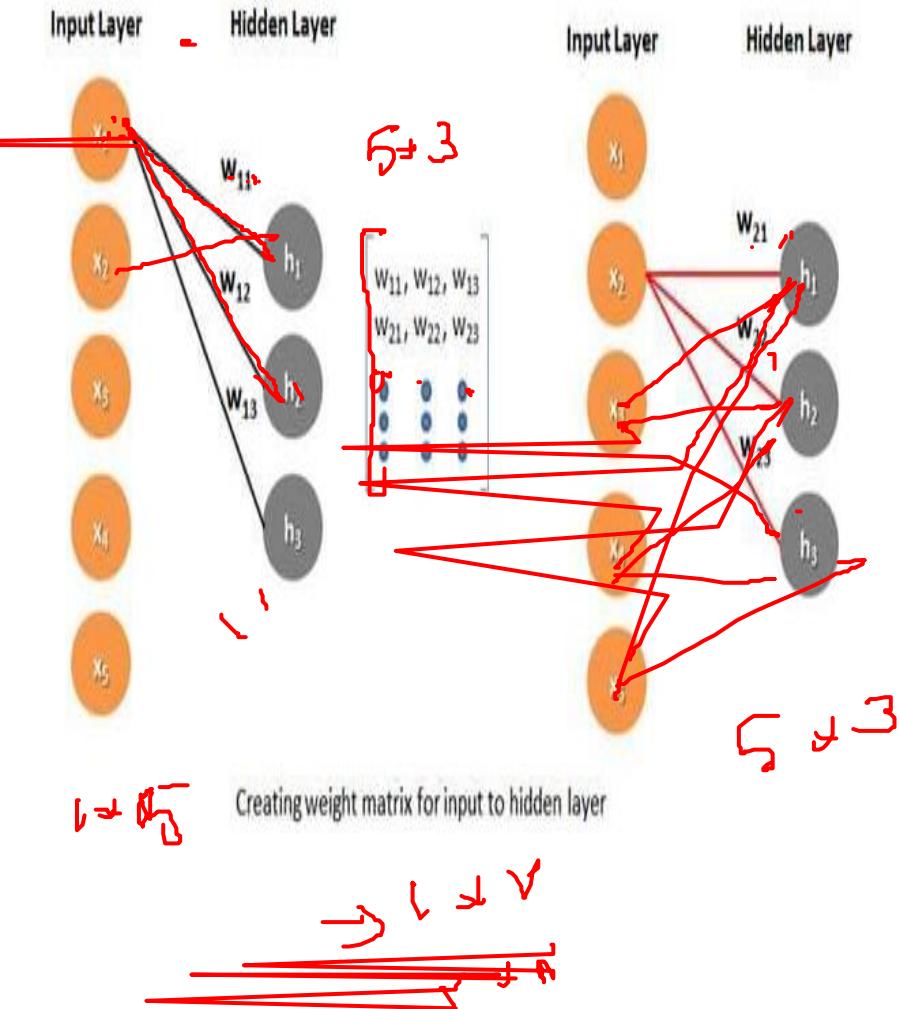
1. Create model Architecture
  2. Forward Propagation
  3. Error Calculation
  4. Weight tuning using backward pass
-

# Model architecture

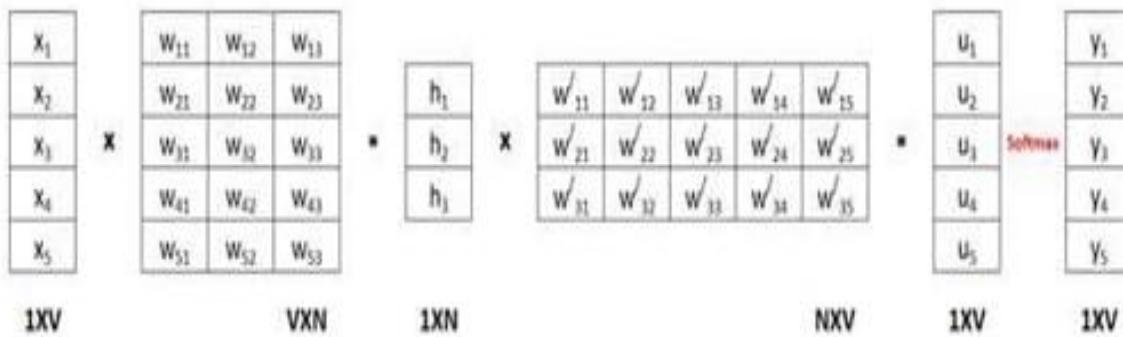


First training data point: The context word is "I" and the target word is "like".

# Weighed matrix

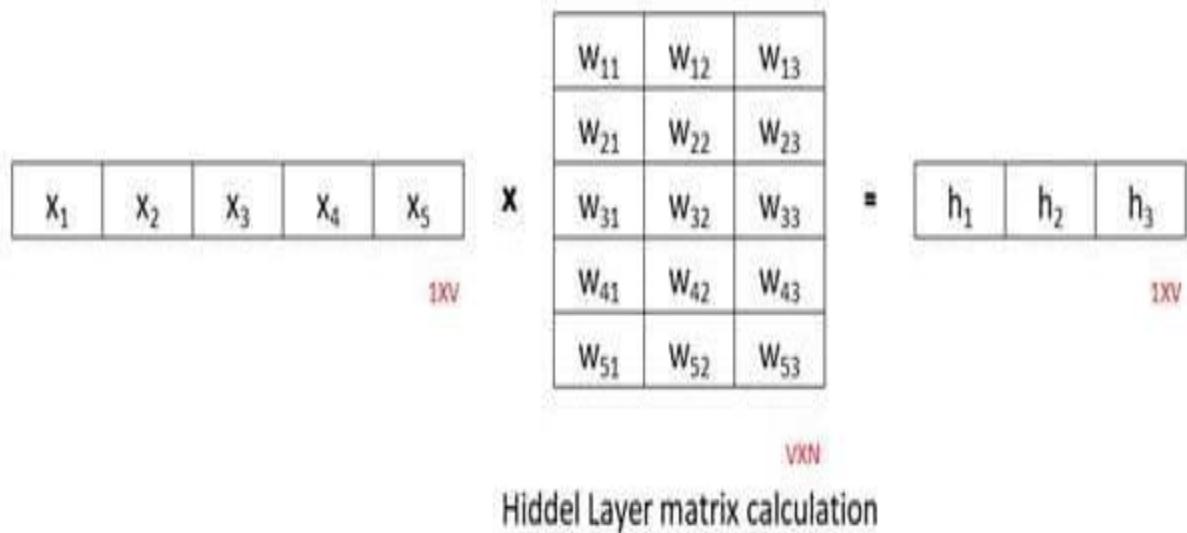


## CBOV Vectorized form:



CBOW vectorized form

# Forward Propagation



$$h_1 = w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + w_{41}x_4 + w_{51}x_5$$

$$h_2 = w_{12}x_1 + w_{22}x_2 + w_{32}x_3 + w_{42}x_4 + w_{52}x_5$$

$$h_3 = w_{13}x_1 + w_{23}x_2 + w_{33}x_3 + w_{43}x_4 + w_{53}x_5$$

⋮ ⋮ ⋮

# Contd..

**Calculate output layer matrix (u):**

$$\begin{array}{|c|c|c|} \hline h_1 & h_2 & h_3 \\ \hline \end{array} \times \begin{array}{|c|c|c|c|c|} \hline w'_{11} & w'_{12} & w'_{13} & w'_{14} & w'_{15} \\ \hline w'_{21} & w'_{22} & w'_{23} & w'_{24} & w'_{25} \\ \hline w'_{31} & w'_{32} & w'_{33} & w'_{34} & w'_{35} \\ \hline \end{array} = \begin{array}{|c|c|c|c|c|} \hline u_1 & u_2 & u_3 & u_4 & u_5 \\ \hline \end{array}$$

1x3      3x5      =      1x5

Output Layer matrix calculation

-

So now:

$$u_1 = w'_{11}h_1 + w'_{21}h_2 + w'_{31}h_3$$

$$u_2 = w'_{12}h_1 + w'_{22}h_2 + w'_{32}h_3$$

$$u_3 = w'_{13}h_1 + w'_{23}h_2 + w'_{33}h_3$$

$$u_4 = w'_{14}h_1 + w'_{24}h_2 + w'_{34}h_3$$

$$u_5 = w'_{15}h_1 + w'_{25}h_2 + w'_{35}h_3$$

# contd

Calculate final Softmax output (y):

$u_1$	$y_1$
$u_2$	$y_2$
$u_3$	$y_3$
$u_4$	$y_4$
$u_5$	$y_5$

Softmax

**1XV**

**1XV**

$$y_1 = \text{Softmax}(u_1)$$

$$y_2 = \text{Softmax}(u_2)$$

$$y_3 = \text{Softmax}(u_3)$$

$$y_4 = \text{Softmax}(u_4)$$

$$y_5 = \text{Softmax}(u_5)$$

$$y_1 = \frac{e^{u_1}}{(e^{u_1} + e^{u_2} + e^{u_3} + e^{u_4} + e^{u_5})}$$

...    ...    ...    ...    ...    ...

$$y_j = \frac{e^j}{\sum_{j=1}^V e^j}$$

# Error calculation – log loss function

$$E = -\log(P(w_t|w_c))$$

$w_t$  = Target word

$w_c$  = Context word

out out

I

~~1, 2, 3~~

$$E(y_2) = -\log(w_{y_2}|w_{x_1})$$

$$= -\log \frac{e^{u_2}}{e^{u_1} + e^{u_2} + e^{u_3} + e^{u_4} + e^{u_5}}$$

$$= -\log(e^{u_2}) + \log(e^{u_1} + e^{u_2} + e^{u_3} + e^{u_4} + e^{u_5})$$

$$= -u_2 + \log(e^{u_1} + e^{u_2} + e^{u_3} + e^{u_4} + e^{u_5})$$

$$E = -u_{j^*} + \log \sum_{j=1}^V e^{u_j}$$

# Back propagation

Step1: Gradient of E with respect to  $w'_{11}$ :

$$\begin{bmatrix} w'_{11} & w'_{12} & w'_{13} & w'_{14} & w'_{15} \\ w'_{21} & w'_{22} & w'_{23} & w'_{24} & w'_{25} \\ w'_{31} & w'_{32} & w'_{33} & w'_{34} & w'_{35} \end{bmatrix} * \begin{array}{c} \begin{matrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{matrix} & \text{softmax} & \begin{matrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{matrix} \\ \text{1XV} & & \text{1XV} \end{array}$$

$$\frac{dE(y_1)}{dw'_{11}} = \frac{dE(y_1)}{du_1} \cdot \frac{du_1}{dw'_{11}}$$

Now as we know

$$E(y_1) = -u_1 + \log(e^{u_1} + e^{u_2} + e^{u_3} + e^{u_4} + e^{u_5})$$

$$So, \frac{dE(y_1)}{du_1} = -\frac{du_1}{du_1} + \frac{d[\log(e^{u_1} + e^{u_2} + e^{u_3} + e^{u_4} + e^{u_5})]}{du_1}$$

Derivative of log function with chain rule.

$$\begin{aligned}&= -1 + \frac{1}{e^{u_1} + e^{u_2} + e^{u_3} + e^{u_4} + e^{u_5}} * u_1 \\&= -1 + \frac{u_1}{e^{u_1} + e^{u_2} + e^{u_3} + e^{u_4} + e^{u_5}} \\&= -1 + y_1\end{aligned}$$

# Contd..

Taking derivative of E with respect to  $u_j$ :

$$\begin{aligned}\frac{dE}{du_j} &= -\frac{d(u_{j*})}{du_j} + \frac{d(\log \sum_{j=1}^V e^{u_j})}{du_j} \\ &= (-t_j + y_j)\end{aligned}$$

$$= (y_j - t_j)$$

$$= \textcircled{e_j}$$

Note:  $t_j = 1$  if  $t_j = t_{j*}$  else  $t_j = 0$

# Contd..

So for first iteration,

$$\frac{dE(y_1)}{du_1} = e_1$$

And,  $\frac{du_1}{dw'_{11}} = \frac{d(w'_{11}h_1 + w'_{21}h_2 + w'_{31}h_3)}{dw'_{11}} = h_1$

Now finally coming back to main derivative which we were trying to solve:

$$\frac{dE(y_1)}{dw'_{11}} = \frac{dE(y_1)}{du_1} \cdot \frac{du_1}{dw'_{11}} = e_1 h_1$$

So generalized form will looks like below:

$$\frac{dE}{dw'} = e * h$$

# Step2: Updating the weight of $w'_{11}$ :

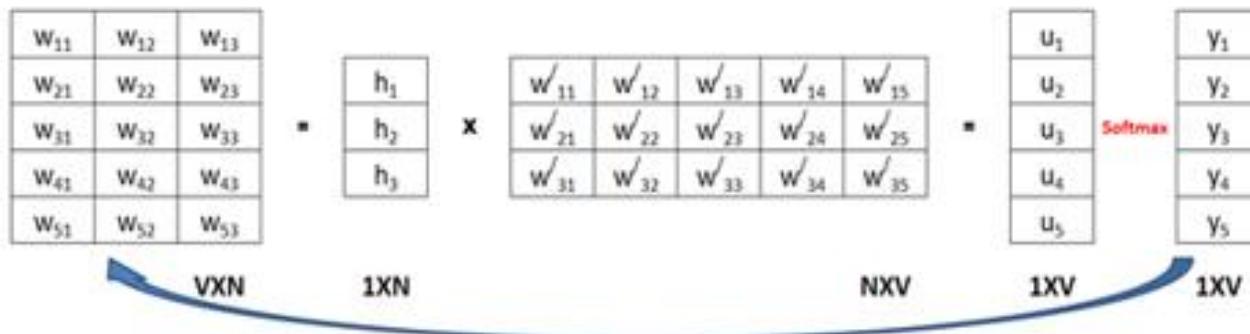
---

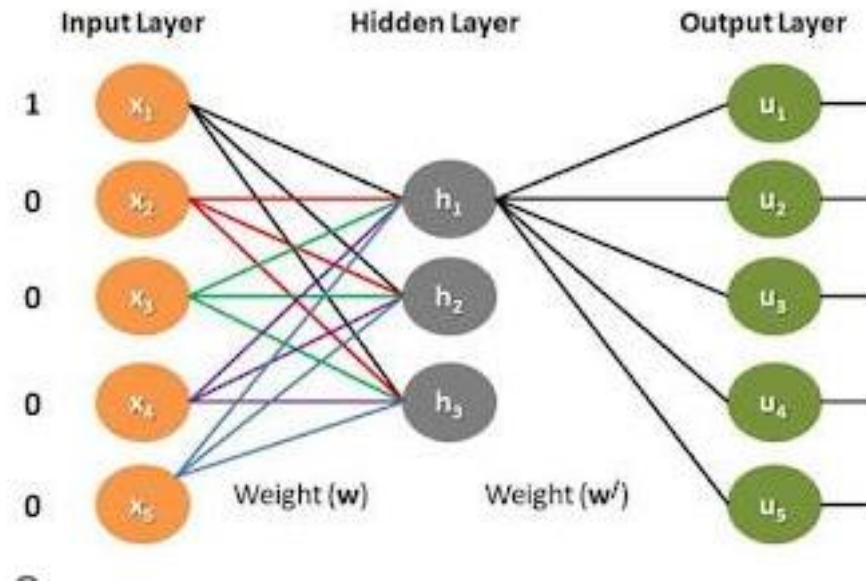
Step2: Updating the weight of  $w'_{11}$ :

$$new(w'_{11}) = w'_{11} - \frac{dE(y_1)}{w'_{11}} = (w'_{11} - e_1 h_1)$$

# Now Updating the first weight

Step1: Gradient of E with respect to  $w_{11}$ :





$$\begin{aligned}
 \frac{dE}{dh_1} &= \left( \frac{dE}{du_1}, \frac{du_1}{dh_1} \right) + \left( \frac{dE}{du_2}, \frac{du_2}{dh_1} \right) + \left( \frac{dE}{du_3}, \frac{du_3}{dh_1} \right) + \left( \frac{dE}{du_4}, \frac{du_4}{dh_1} \right) + \left( \frac{dE}{du_5}, \frac{du_5}{dh_1} \right) \\
 &= ew'_{11} + ew'_{12} + ew'_{13} + ew'_{14} + ew'_{15}
 \end{aligned}$$

As for  $u_1$  and  $h_1$ ,

$$\frac{du_1}{dh_1} = \frac{d(w'_{11}h_1 + w'_{21}h_2 + w'_{31}h_3)}{dh_1} = w'_{11}$$

*As,  $h_2$  and  $h_3$  are constant with respect to  $h_1$*

*Similarly we can calculate :  $\frac{du_2}{dh_1}, \frac{du_3}{dh_1}, \frac{du_4}{dh_1}, \frac{du_5}{dh_1}$*

$$And, \frac{dh_1}{dw_{11}} = \frac{d(w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + w_{41}x_4 + w_{51}x_5)}{dw_{11}}$$

# Contd..

$$\frac{dE}{dw_{11}} = \frac{dE}{dh_1} \cdot \frac{dh_1}{dw_{11}}$$

$$= (ew'_{11} + ew'_{12} + ew'_{13} + ew'_{14} + ew'_{15}) * x$$

Step2: Updating the weight of  $w_{11}$ :

$$new(w_{11}) = w_{11} - \frac{dE}{dw_{11}}$$

$$= w_{11} - (ew'_{11} + ew'_{12} + ew'_{13} + ew'_{14} + ew'_{15}) * x$$

# Skip gram

**Step - 1**      The product is really good The product is wonderful The product is awful

**Step - 2**      Window Size      1

**Step - 3**      Get one hot encoding for each word

# Contd..

1	The	1	0	0	0	0	0	0
2	product	0	1	0	0	0	0	0
3	is	0	0	1	0	0	0	0
4	really	0	0	0	1	0	0	0
5	wonderful	0	0	0	0	1	0	0
6	good	0	0	0	0	0	1	0
7	awful	0	0	0	0	0	0	1

## Step - 1

The product is really good The product is wonderful The product is awful

## Step - 4

Input Words	Target Word
product	The
product	is
is	product
is	really

### One Hot Encoding

1	The	1	0	0	0	0	0	0
2	product	0	1	0	0	0	0	0
3	is	0	0	1	0	0	0	0
4	really	0	0	0	1	0	0	0
5	wonderful	0	0	0	0	1	0	0
6	good	0	0	0	0	0	1	0
7	awful	0	0	0	0	0	0	1

### Training Data

Context Words	Target Word
The,is	Product
product,really	is
Is,good	really
Good,product	the

product

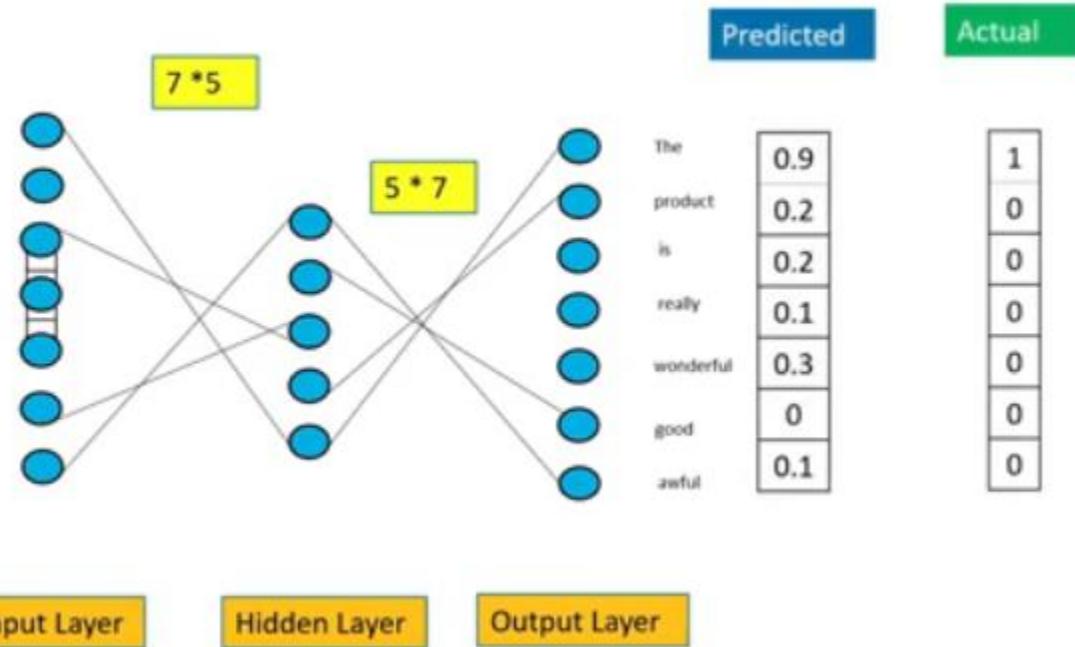
0
1
0
0
0
0
0

The

1
0
0
0
0
0
0

# Contd..

One hot encoding of - Product



# Skip gram negative example

---

## Why negative sampling?

- Negative sampling allows us to only modify a small percentage of the weights, rather than all of them for each training sample.

## How it works?

K negative samples are randomly drawn from a noise distribution

K is a hyper-parameter that can be empirically tuned, with a typical range of [5,20]

# Algorithm

---

The intuition of skip-gram is:

1. Treat the target word and a neighboring context word as positive examples.
  2. Randomly sample other words in the lexicon to get negative samples
  3. Use **logistic regression** to train a classifier to distinguish those two cases
  4. Use the **regression weights as the embeddings**
-

# Skip gram training data

- Asssume that **context words** are those in +/- 2 word window.
- A training sentence:

... lemon, a tablespoon of **apricot** jam a pinch ...

c1            c2 **target**    c3    c4

# Contd..

- skip-gram with negative sampling (**SGNS**)

Text : .../*lemon, a [tablespoon of apricot jam, a] pinch...*

c1            c2    **[target]**    c3    c4

- For each positive example we'll grab k negative examples, sampling by frequency

### **positive examples +**

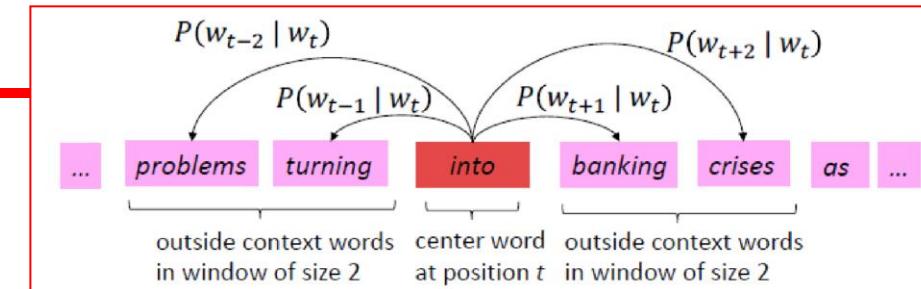
t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

### **negative examples -**

t	c	t	c
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

# How does negative sampling work

- skip-gram with negative sampling (SGNS)
- $N = 2, C=4$  (Here C is the context window)



Thou shalt not make a machine in the likeness of a human mind

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine
a	not
a	make
a	machine
a	in
machine	make
machine	a
machine	in
machine	the
in	a
in	machine
in	the
in	likeness

Source Credit : <https://jalammar.github.io/illustrated-word2vec/>

# Contd..

- skip-gram with negative sampling (**SGNS**)
- $N = 2$  ,  $C=4$  (Here C is the context window)

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine

input word	output word	target
not	thou	1
not	shalt	1
not	make	1
not	a	1
make	shalt	1
make	not	1
make	a	1
make	machine	1

input word	output word	target
not	thou	1
not		0
not		0
not	shalt	1
not	make	1

↗ Negative examples

# Skip gram goal

Our goal is to train a classifier such that,

- Given a **tuple  $(t,c)$**  of a **target word  $t$**  paired with a **context word  $c$** 
  - $(\text{apricot}, \text{jam})$
  - $(\text{apricot}, \text{aardvark})$
- it will return the **probability that  $c$  is a real context word** (true for **jam**, false for **aardvark**):

$$P(+ | t, c)$$

Probability that word  $c$  is nota real context word for  $t$

$$P(- | t, c) = 1 - P(+|t,c)$$

Probability that word  $c$  is **not** a real context word for  $t$

# How to Compute $P(+|t,c)$ ?

---

The intuition of the skipgram model is to base this probability on similarity:

- *A word is likely to occur near the target if its embedding is similar to the target embedding.*
- *Two vectors are similar if they have a high dot product.*
  - Similarity( $t,c$ )  $\propto t \cdot c$
  - The dot product  $t \cdot c$  is not a probability, it's just a number ranging from 0 to  $\infty$ .

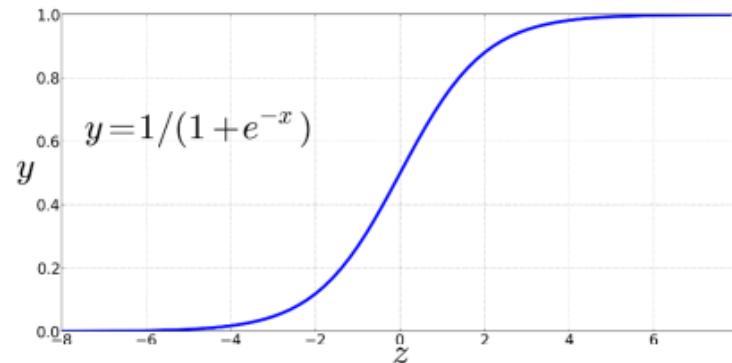
# How to Compute $P(+|t,c)$ ?

*Turning dot product into a probability*



- To turn **dot product** into a probability, we'll use **logistic** or **sigmoid function**  $\sigma(x)$ .

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



- The probability that word  $c$  is a real context word for target word  $t$  is:

$$P(+|t,c) = \frac{1}{1 + e^{-t \cdot c}}$$

$$\begin{aligned} P(-|t,c) &= 1 - P(+|t,c) \\ &= \frac{e^{-t \cdot c}}{1 + e^{-t \cdot c}} \end{aligned}$$

# Learning skip-gram embeddings

## Skip-Gram Training



- **Training sentence:**

... lemon, a tablespoon of **apricot** jam a pinch ...

c1            c2      **t**      c3    c4

**positive examples +**  
t            c  
apricot    tablespoon  
apricot    of  
apricot    jam  
apricot    a

- For each positive example, we'll create ***k* negative examples.**
- Using ***noise*** words
  - Noise word is any random word that isn't ***target word t (apricot)***

# Contd..

- **Training sentence:**

... lemon, a tablespoon of **apricot** jam a pinch ...

c1      c2      **t**      c3      c4

Create k (=2) negative examples.

**positive examples +**

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

**negative examples -**

t	c	t	c
apricot	aardvark	apricot	twelve
apricot	puddle	apricot	hello
apricot	where	apricot	dear
apricot	coaxial	apricot	forever

# How to select negative samples

---

- Could pick  $w$  as a noise word according to their unigram frequency  $P(w)$
- More common to chosen then according to  $P_a(w)$

$$P_\alpha(w) = \frac{\text{count}(w)^\alpha}{\sum_w \text{count}(w)^\alpha}$$

- $\alpha = 0.75$  works well because it gives rare noise words slightly higher probability
- To show this, imagine two events  $P(a) = .99$  and  $P(b) = .01$ :

$$P_\alpha(a) = \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = .97$$

$$P_\alpha(b) = \frac{.01^{.75}}{.99^{.75} + .01^{.75}} = .03$$

# Word2Vec Algorithm

## Model Training - Skip gram – How the weights are learnt ?

- $\text{Sim}(w, c) \approx w \cdot c$ . To turn this into a probability use the sigmoid from logistic regression
- Loss Function**
- **Maximize** the similarity of the target word, context word pairs ( $w, c_{\text{pos}}$ ) drawn from the positive data
  - **Minimize** the similarity of the ( $w, c_{\text{neg}}$ ) pairs drawn from the negative data.
  - **Stochastic gradient descent!**

$$\begin{aligned}
 L_{CE} &= -\log \left[ P(+|w, c_{\text{pos}}) \prod_{i=1}^k P(-|w, c_{\text{neg}_i}) \right] \\
 &= - \left[ \log P(+|w, c_{\text{pos}}) + \sum_{i=1}^k \log P(-|w, c_{\text{neg}_i}) \right] \\
 &= - \left[ \log P(+|w, c_{\text{pos}}) + \sum_{i=1}^k \log (1 - P(+|w, c_{\text{neg}_i})) \right] \\
 &= - \left[ \log \sigma(c_{\text{pos}} \cdot w) + \sum_{i=1}^k \log \sigma(-c_{\text{neg}_i} \cdot w) \right]
 \end{aligned}$$

$$\begin{aligned}
 P(+|w, c) &= \sigma(c \cdot w) = \frac{1}{1 + \exp(-c \cdot w)} \\
 P(-|w, c) &= 1 - P(+|w, c) \\
 &= \sigma(-c \cdot w) = \frac{1}{1 + \exp(c \cdot w)} \\
 P(+|w, c_{1:L}) &= \prod_{i=1}^L \sigma(c_i \cdot w) \\
 \log P(+|w, c_{1:L}) &= \sum_{i=1}^L \log \sigma(c_i \cdot w)
 \end{aligned}$$

# Word2Vec Algorithm

## Model Training - Skip gram – How the weights are learnt ?

- Stochastic gradient descent!

$$L_{CE} = - \left[ \log \sigma(c_{pos} \cdot w) + \sum_{i=1}^k \log \sigma(-c_{neg_i} \cdot w) \right]$$

$$w^{t+1} = w^t - h \frac{d}{dw} L(f(x; w), y)$$

$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(c_{pos} \cdot w) - 1]w$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(c_{neg} \cdot w)]w$$

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(c_{pos} \cdot w) - 1]c_{pos} + \sum_{i=1}^k [\sigma(c_{neg_i} \cdot w)]c_{neg_i}$$

$$c_{pos}^{t+1} = c_{pos}^t - \eta [\sigma(c_{pos}^t \cdot w^t) - 1]w^t$$

$$c_{neg}^{t+1} = c_{neg}^t - \eta [\sigma(c_{neg}^t \cdot w^t)]w^t$$

$$w^{t+1} = w^t - \eta \left[ [\sigma(c_{pos} \cdot w^t) - 1]c_{pos} + \sum_{i=1}^k [\sigma(c_{neg_i} \cdot w^t)]c_{neg_i} \right]$$

# Learning skip-gram embeddings

## *Train using gradient descent*

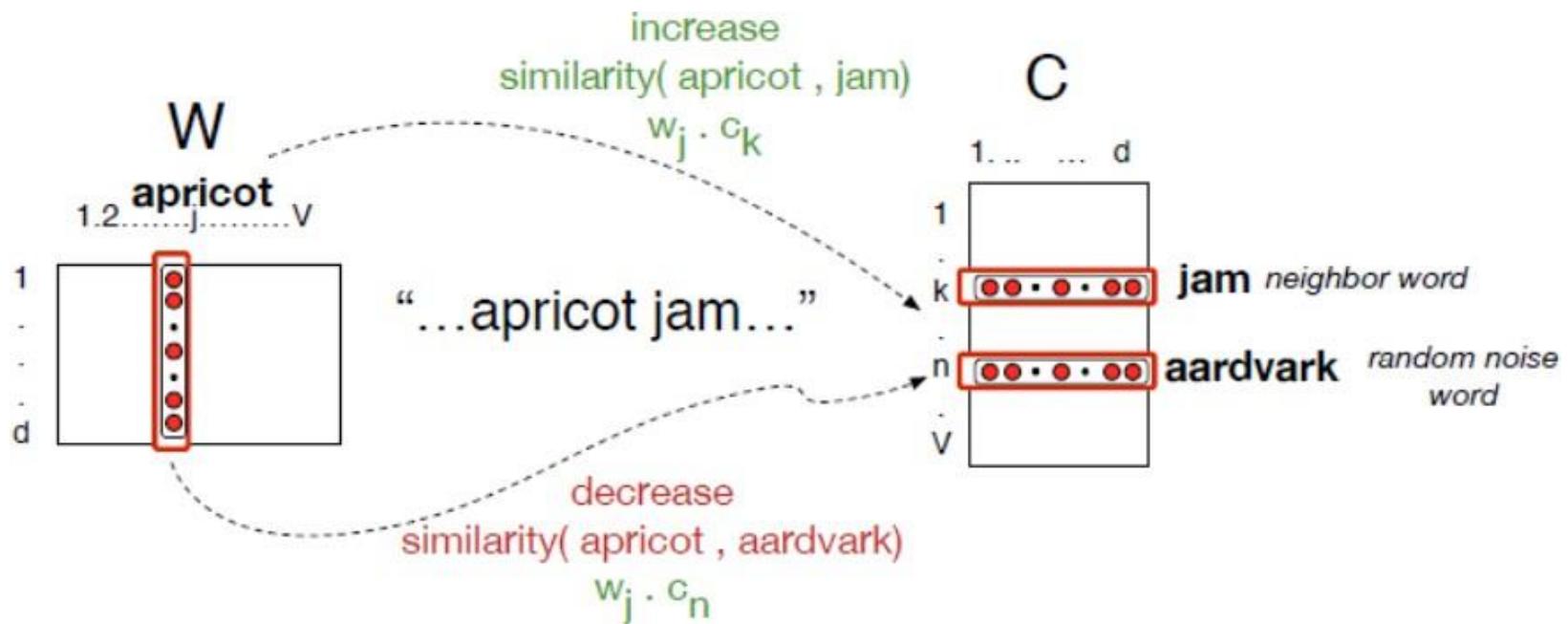


- We can then **use stochastic gradient descent to train to this objective**,
  - iteratively modifying the parameters (the embeddings for each target word  $t$  and each context word or noise word  $c$  in the vocabulary) to maximize the objective.
- **Skip-gram model** actually learns **two separate embeddings for each word  $w$ :**
  - **target embedding  $t$**  and
  - **context embedding  $c$ .**
- These embeddings are stored in **two matrices**,
  - **target matrix  $W$**  and
  - **context matrix  $C$ .**

# Learning skip-gram embeddings

## *Train using gradient descent*

- The skip-gram model tries to shift embeddings so the target embedding (**apricot**) are closer to (have a higher dot product with) context embeddings for nearby words (**jam**) and further from (have a lower dot product with) context embeddings for words that don't occur nearby (**aardvark**).



# Learning skip-gram embeddings

## *Train using gradient descent*

---

- Just as in logistic regression, the learning algorithm starts with randomly initialized  $W$  and  $C$  matrices, and then walks through the training corpus using gradient descent to update  $W$  and  $C$  so as to maximize the objective criteria.
- Thus matrices  $W$  and  $C$  function as the parameters  $\theta$  that logistic regression is tuning.
- Once embeddings are learned, we'll have two embeddings for each word  $w_i$ :  $t_i$  and  $c_i$ .
  - We can choose to throw away the  $C$  matrix and just keep  $W$ , in which case each word  $i$  will be represented by the vector  $t_i$ .
  - Alternatively we can add the two embeddings together, using the summed embedding  $t_i + c_i$  as the new  $d$ -dimensional embedding



---

Thank you



# Natural Language Processing



**BITS** Pilani

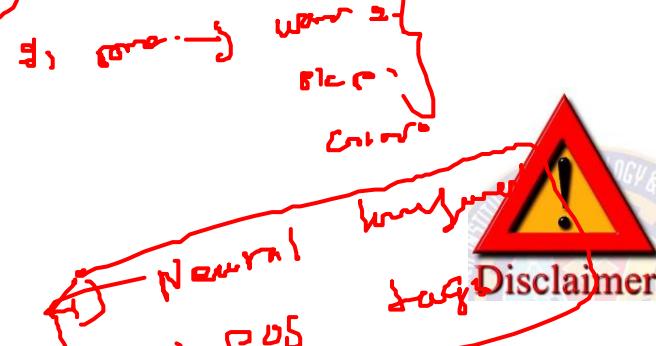
Pilani Campus

Dr. Vijayalakshmi Anand  
BITS Pilani

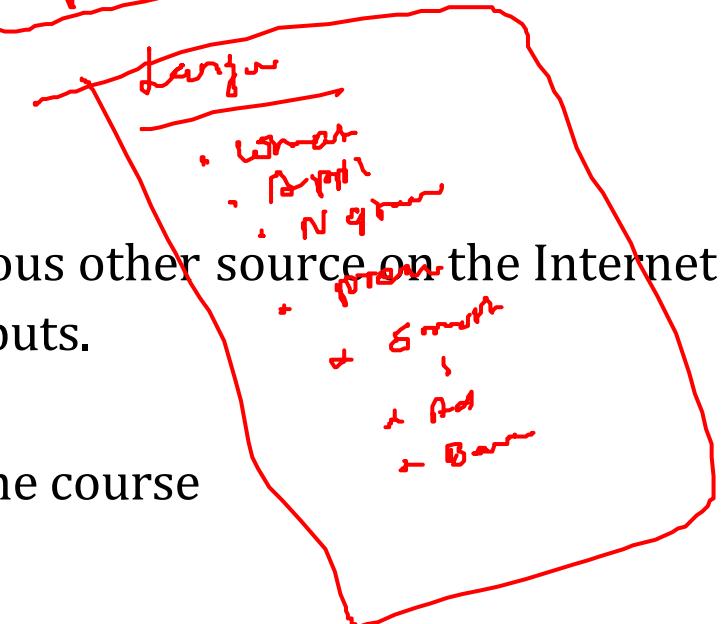
# Natural Language Processing

3) Vector generate | TAC PT | Content [ in ]

1) F → **Disclaimer and Acknowledgement**



That NLP  
application  
diff. NLG & NLP  
process



- The content for these slides has been obtained from books and various other source on the Internet
- I hereby acknowledge all the contributors for their material and inputs.
- I have provided source information wherever necessary
- I have added and modified the content to suit the requirements of the course

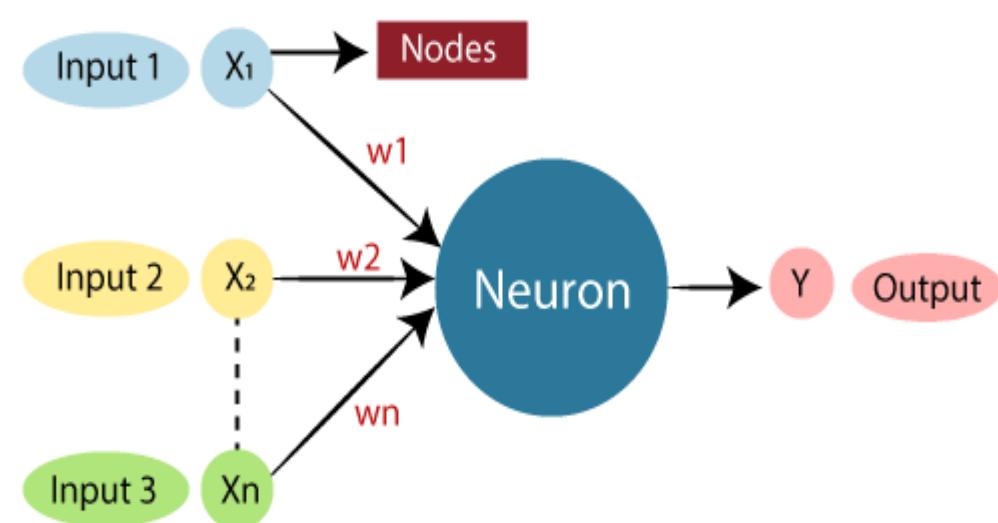
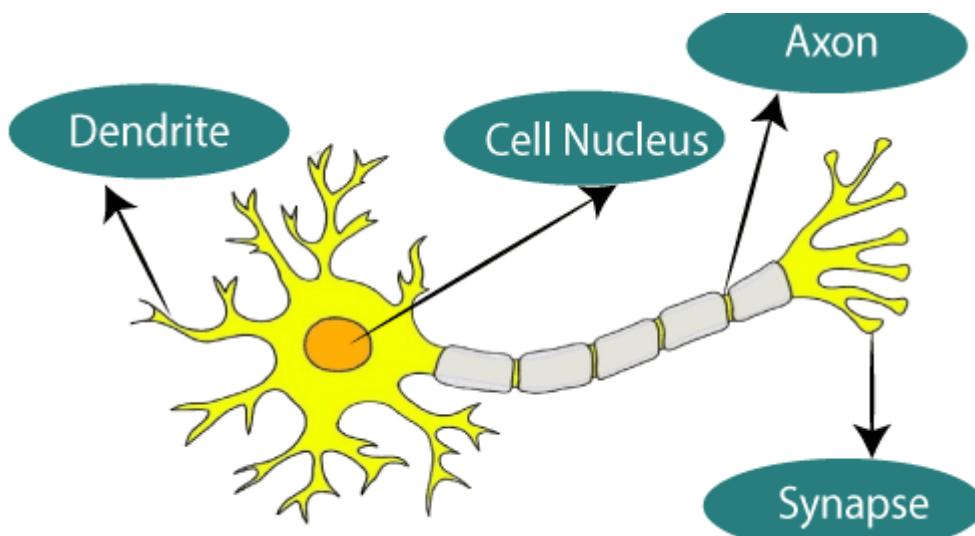
# Session Contents

- Revision -Neural Network (Forward and backward pass)
- Application – Sentiment Analysis
  - Using logistic regression
  - Using Neural Network
- Neural Language model

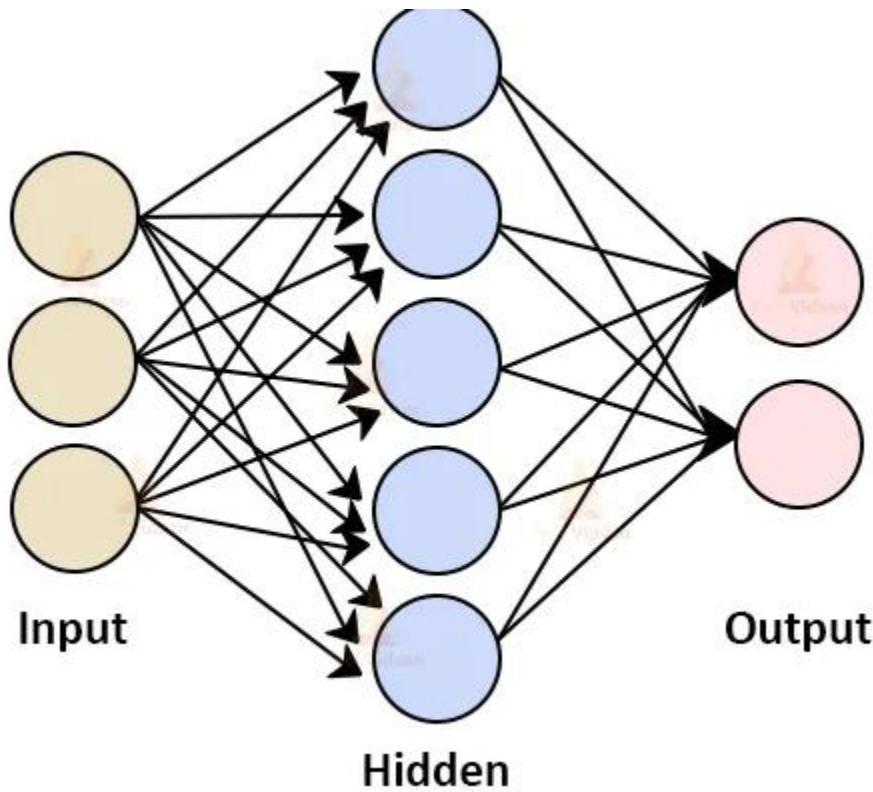
# Neural network -Introduction

- What is neural network

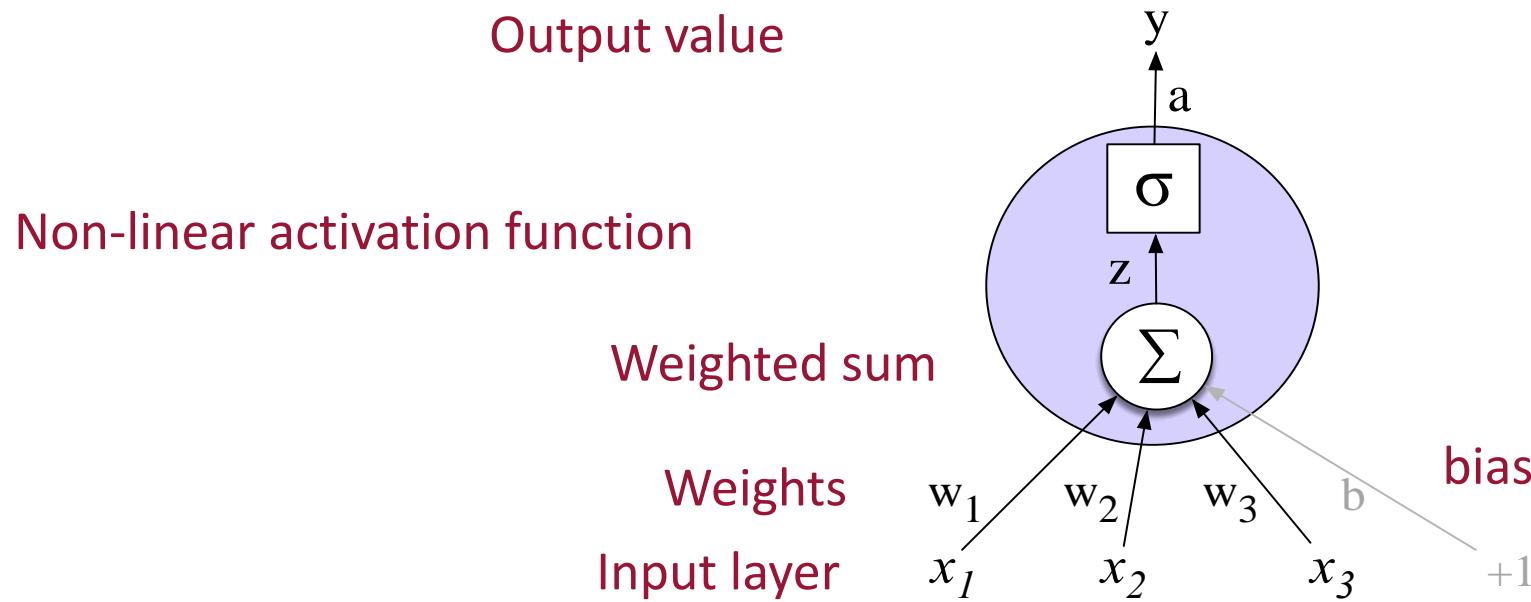
Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks. These neurons are known as nodes.



# Architecture of neural networks

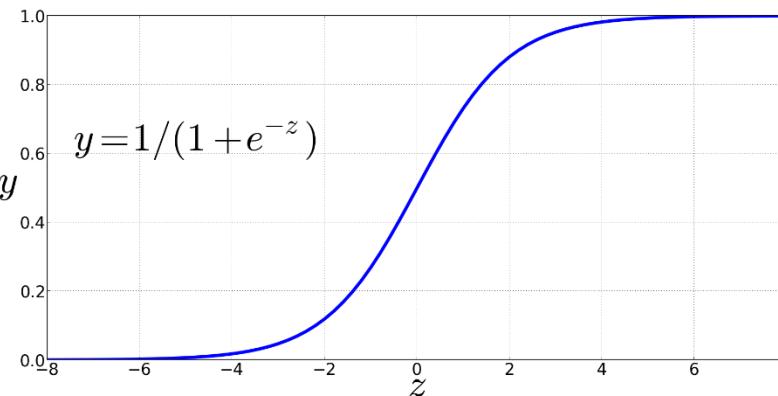


# Neural Network-How it works



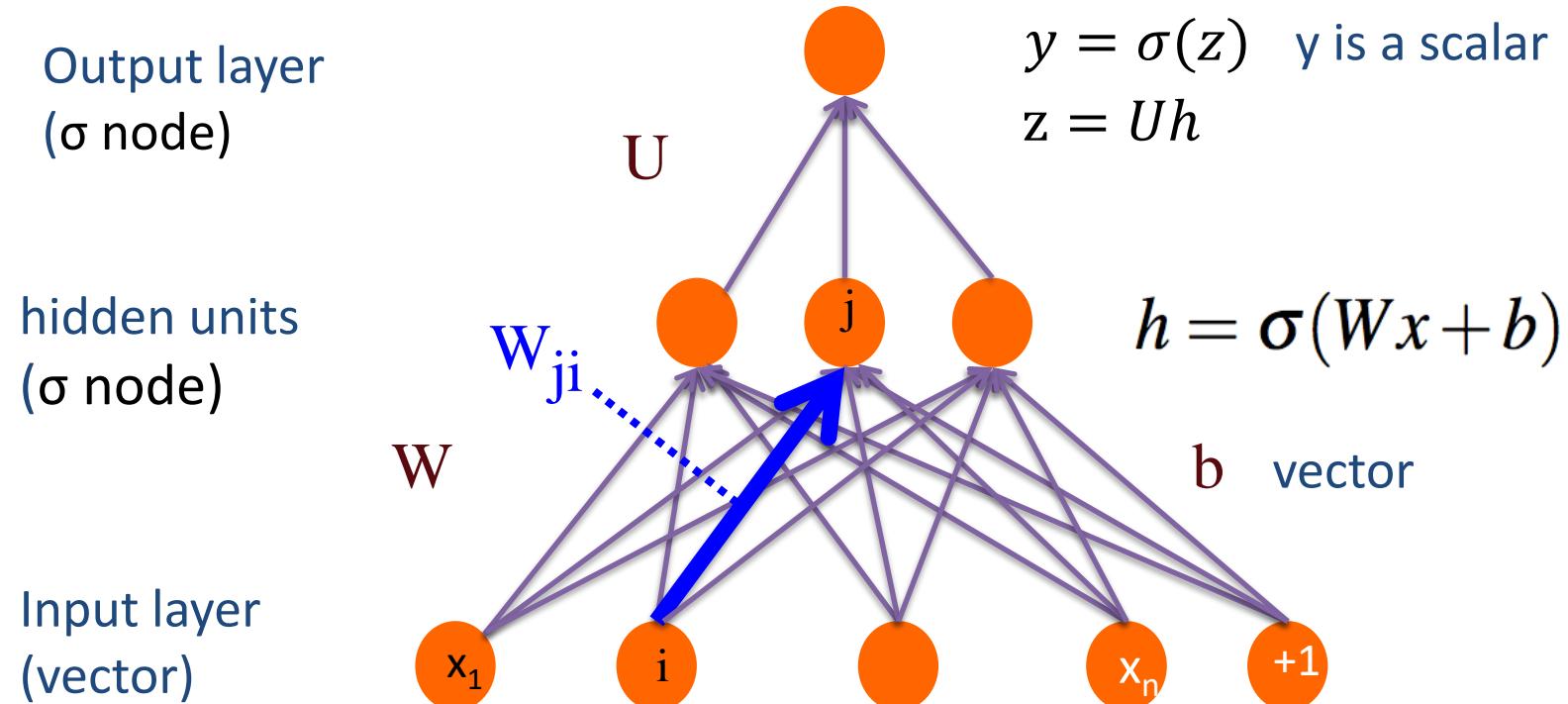
Sigmoid

$$y = s(z) = \frac{1}{1 + e^{-z}}$$

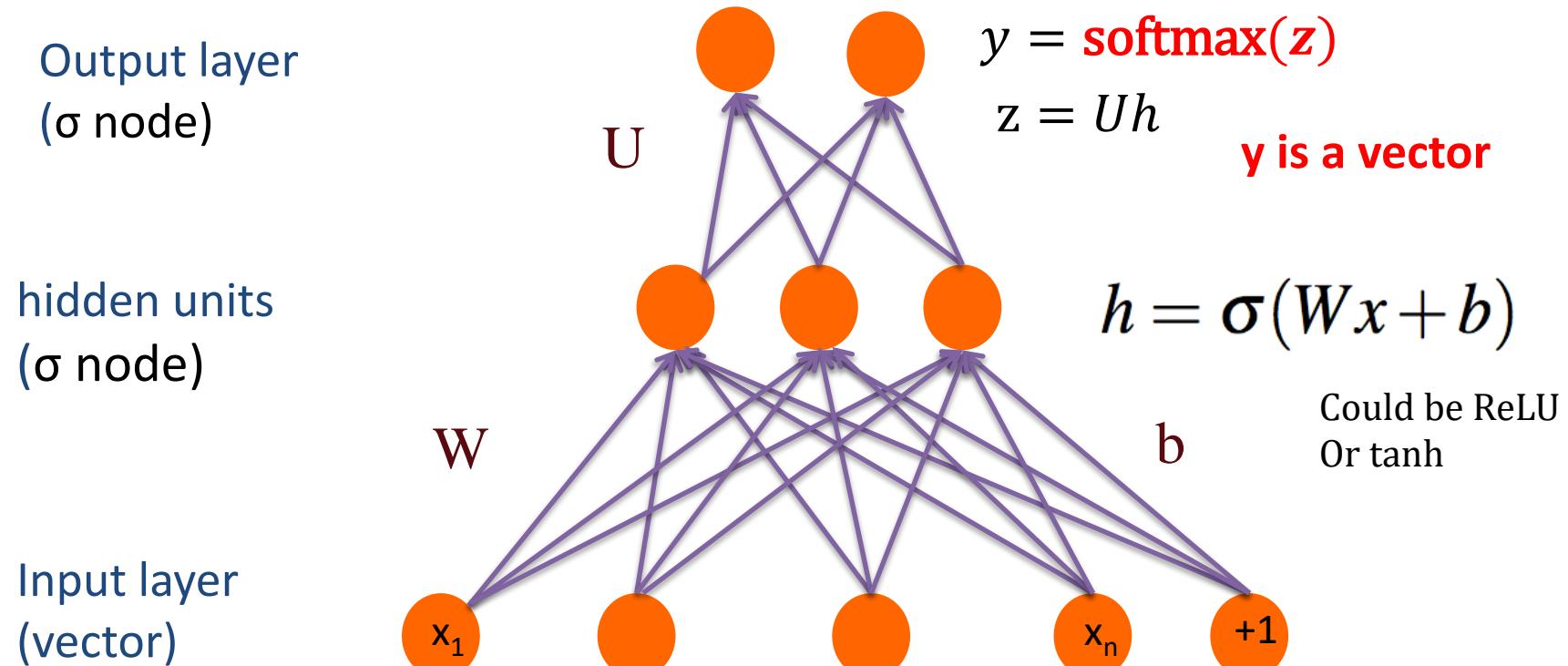


6

# Two-Layer Network with scalar output



# Two-Layer Network with softmax output



# Reminder: softmax: a generalization of sigmoid

- For a vector  $z$  of dimensionality  $k$ , the softmax is:

$$\text{softmax}(z) = \left[ \frac{\exp(z_1)}{\sum_{i=1}^k \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^k \exp(z_i)}, \dots, \frac{\exp(z_k)}{\sum_{i=1}^k \exp(z_i)} \right]$$

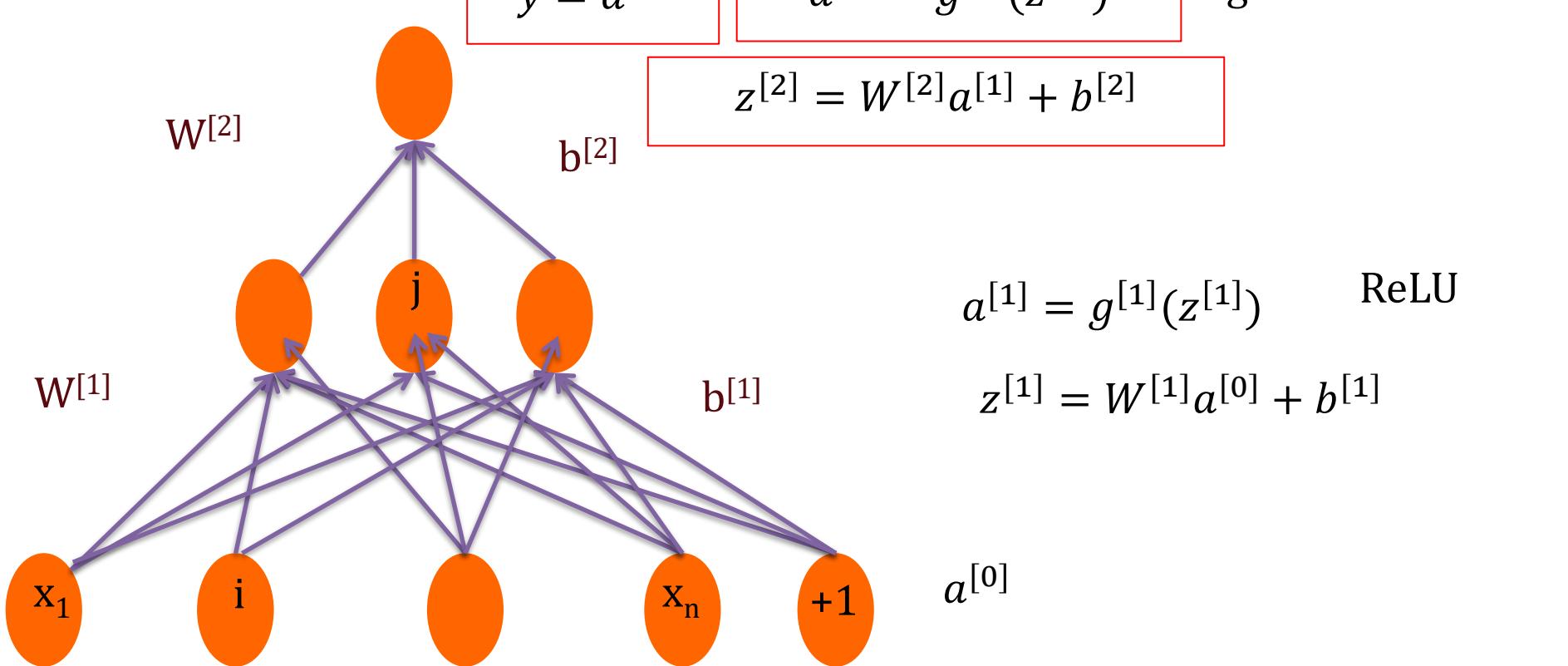
$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^k \exp(z_j)} \quad 1 \leq i \leq k$$

- Example:

$$z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

$$\text{softmax}(z) = [0.055, 0.090, 0.006, 0.099, 0.74, 0.010]$$

# Multi-layer Notation



# Multi-layer Notation

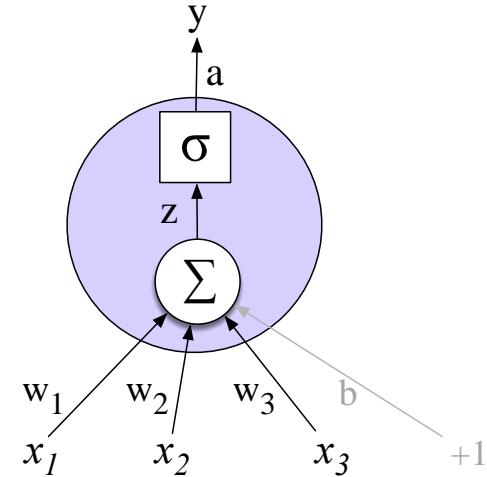
$$z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$$

$$a^{[1]} = g^{[1]}(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = g^{[2]}(z^{[2]})$$

$$\hat{y} = a^{[2]}$$



**for  $i$  in 1..n**

$$z^{[i]} = W^{[i]}a^{[i-1]} + b^{[i]}$$

$$a^{[i]} = g^{[i]}(z^{[i]})$$

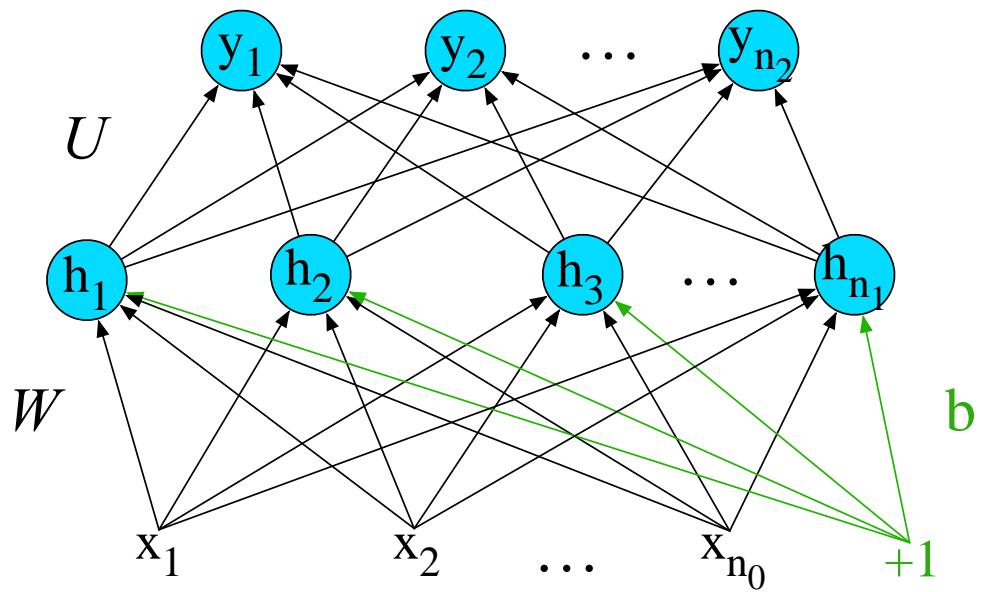
$$\hat{y} = a^{[n]}$$

# Replacing the bias unit

- Instead of:  $x = x_1, x_2, \dots, x_{n_0}$   

$$h = \sigma(Wx + b)$$

$$h_j = \sigma \left( \sum_{i=1}^{n_0} W_{ji} x_i + b_j \right)$$

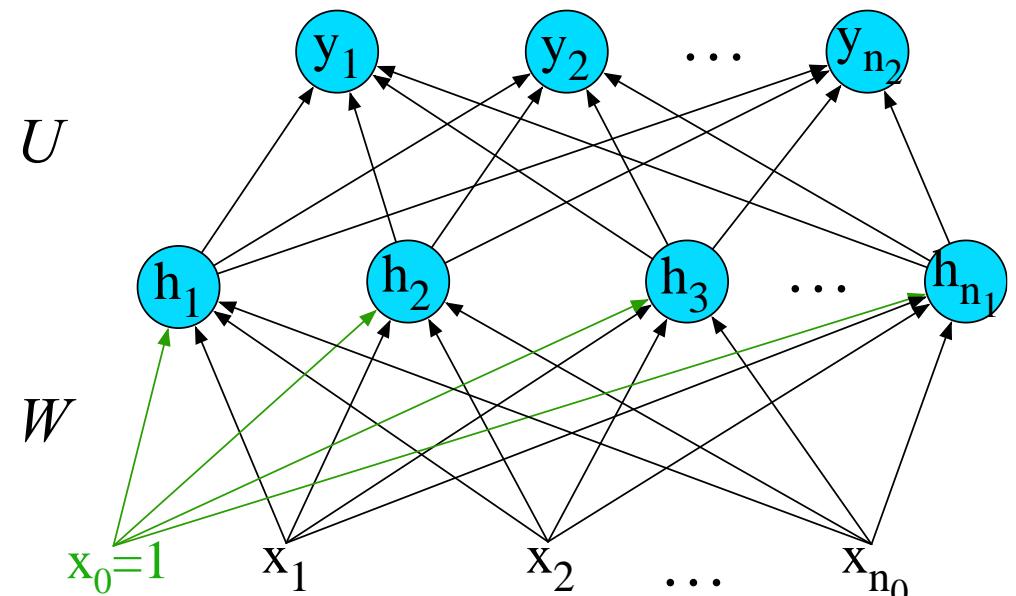


- We'll do this:

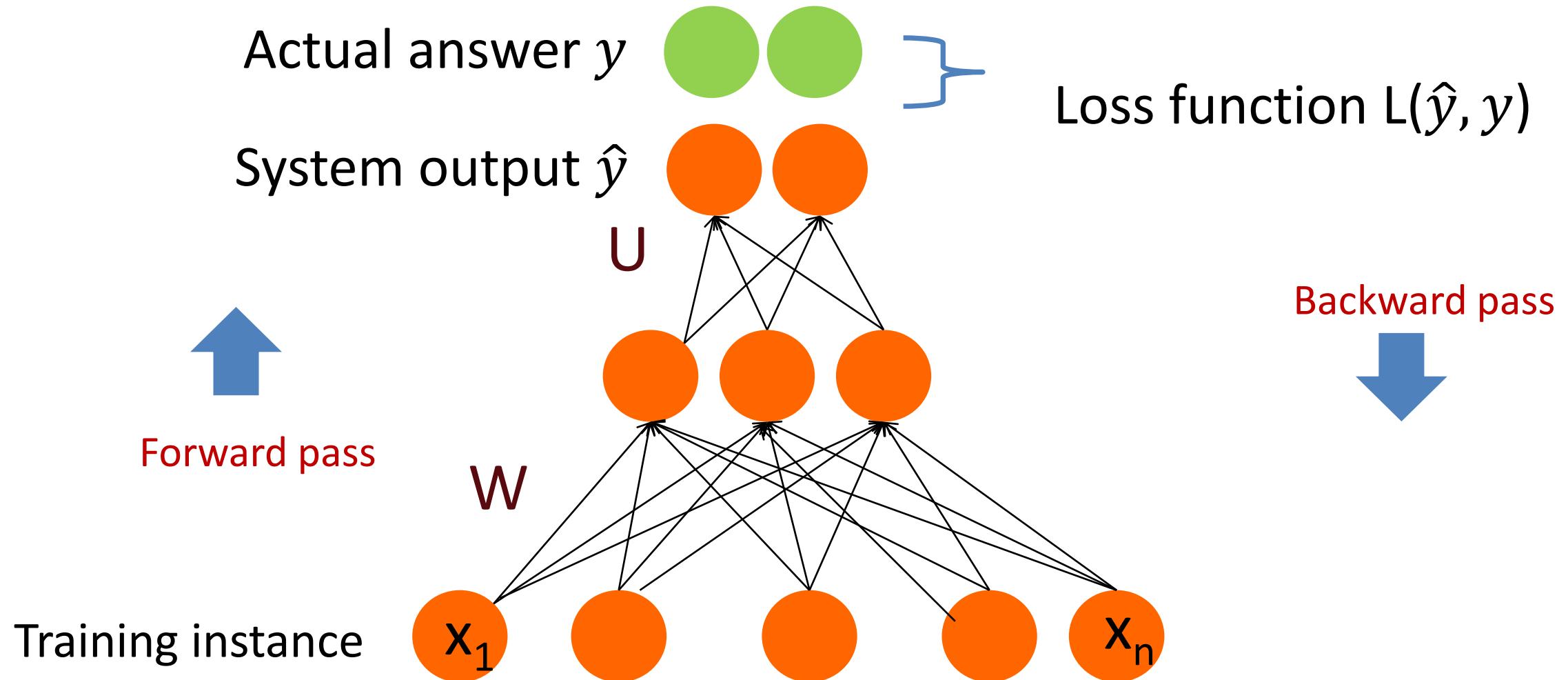
$$x = x_0, x_1, x_2, \dots, x_{n_0}$$

$$h = \sigma(Wx)$$

$$\sigma \left( \sum_{i=0}^{n_0} W_{ji} x_i \right)$$



# Intuition: training a 2-layer Network



# Intuition: Training a 2-layer network

---

- For every training tuple  $(x, y)$ 
  - Run *forward* computation to find our estimate  $\hat{y}$
  - Run *backward* computation to update weights:
    - For every output node
      - Compute loss  $L$  between true  $y$  and the estimated  $\hat{y}$
      - For every weight  $w$  from hidden layer to the output layer
        - » Update the weight
    - For every hidden node
      - Assess how much blame it deserves for the current answer
      - For every weight  $w$  from input layer to the hidden layer
        - » Update the weight

# Reminder: Loss Function for binary logistic regression

---

- A measure for how far off the current answer is to the right answer
- Cross entropy loss for logistic regression:

$$\begin{aligned} L_{\text{CE}}(\hat{y}, y) &= -\log p(y|x) = -[y \log \hat{y} + (1-y) \log(1-\hat{y})] \\ &= -[y \log \sigma(w \cdot x + b) + (1-y) \log(1 - \sigma(w \cdot x + b))] \end{aligned}$$

# Reminder: gradient descent for weight updates

---

- Use the derivative of the loss function with respect to weights  $\frac{d}{dw} L(f(x; w), y)$
- To tell us how to adjust weights for each training item
  - Move them in the opposite direction of the gradient

$$w^{t+1} = w^t - h \frac{d}{dw} L(f(x, w), y)$$

– For logistic regression

$$\frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$$

# Backward differentiation

- For training, we need the derivative of the loss with respect to weights in early layers of the network
- But loss is computed only at the very end of the network!
- Solution: **backward differentiation**
- Given a computation graph and the derivatives of all the functions in it we can automatically compute the derivative of the loss with respect to these early weights.

# Backprop

---

- For training, we need the derivative of the loss with respect to each weight in every layer of the network
- But the loss is computed only at the very end of the network!
- Solution: **error backpropagation** (Rumelhart, Hinton, Williams, 1986)
- **Backprop** is a special case of **backward differentiation**
- Which relies on **computation graphs**.



# Application : Sentiment Analysis

# Use cases for feedforward networks

Let's consider 2 (simplified) sample tasks:

1. Text classification → Sentiment analysis

2. Language modeling → News language model

# Sentiment example: does $y=1$ or $y=0$ ?

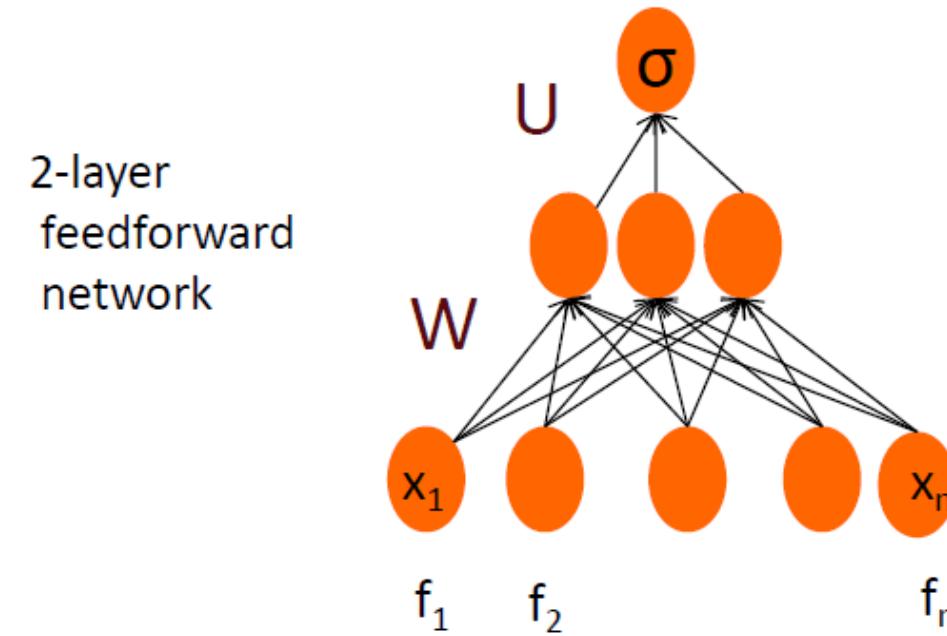
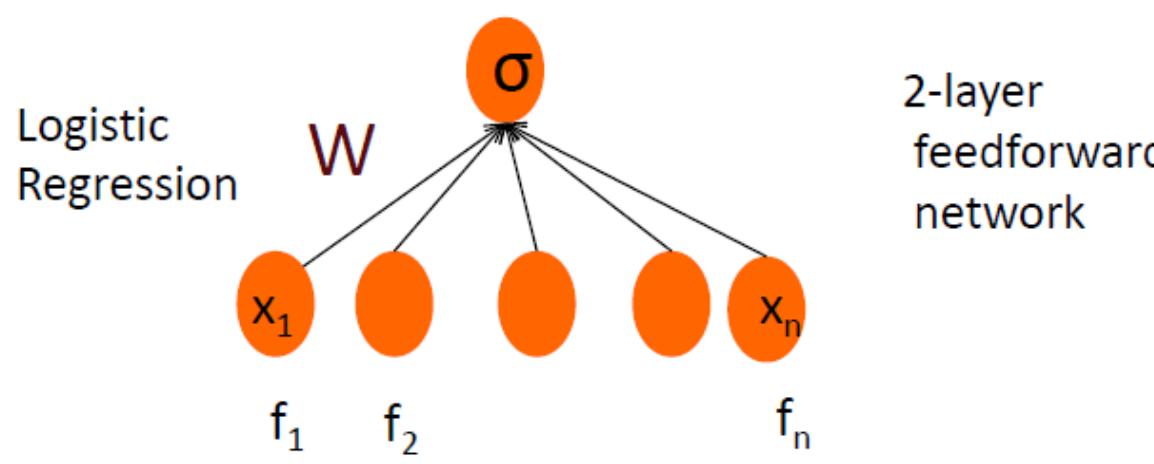
---

It's hokey . There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable ? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .

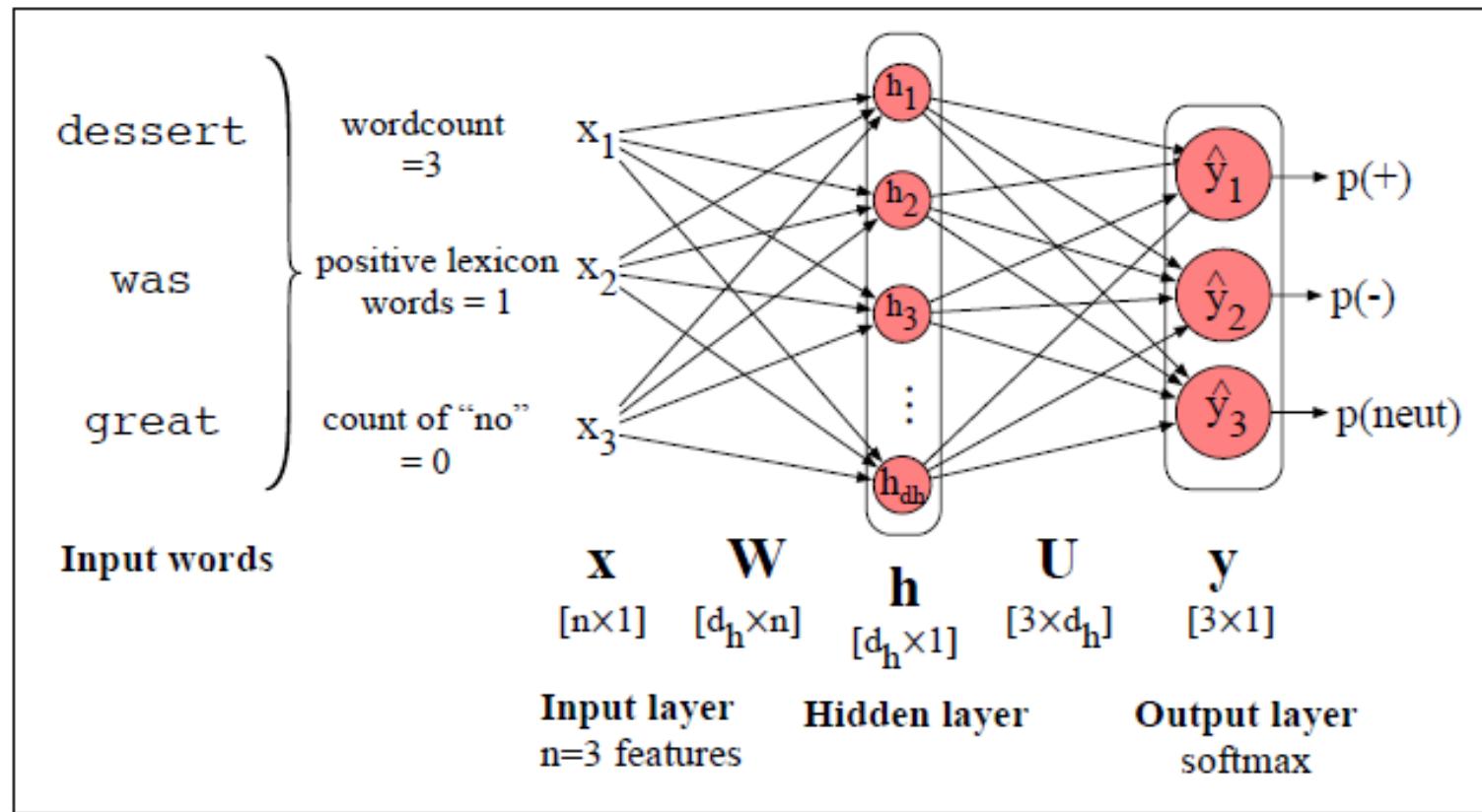
# Feedforward nets for simple classification



- Just adding a hidden layer to logistic regression allows the network to use non-linear interactions between features which may (or may not) improve performance



# Feedforward networks for NLP: Classification



(each  $x_i$  is a hand-designed feature)

$$\mathbf{x} = [x_1, x_2, \dots x_N]$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$z = Uh$$

$$\hat{y} = \text{softmax}(z)$$

$$\begin{array}{c} (3 \times 1) \quad (4 \times 1) \\ \text{---} \quad \text{---} \\ (3 \times 1) \quad (5 \times 3) \\ \text{---} \quad \text{---} \\ (5 \times 1) \\ (3 \times 5) \end{array}$$

**Figure 7.10** Feedforward network sentiment analysis using traditional hand-built features of the input text.

# Classification: Sentiment Analysis



It's **hokey**. There are virtually **no** surprises , and the writing is **second-rate**.  
So why was it so **enjoyable** ? For one thing , the cast is  
**great** . Another **nice** touch is the music **I** was overcome with the urge to get off  
the couch and start dancing . It sucked **me** in , and it'll do the same to **you** .

$x_1 = 3$        $x_5 = 0$        $x_6 = 4.19$

$x_2 = 2$        $x_3 = 1$        $x_4 = 3$

Var	Definition	Value in Fig. 5.2
$x_1$	count(positive lexicon) $\in$ doc)	3
$x_2$	count(negative lexicon) $\in$ doc)	2
$x_3$	$\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
$x_4$	count(1st and 2nd pronouns $\in$ doc)	3
$x_5$	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
$x_6$	log(word count of doc)	$\ln(66) = 4.19$

## Sentiment Features

# Classifying sentiment using logistic regression

Suppose  $w = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$

$b = 0.1$

$$\frac{e^z}{1 + e^z}$$

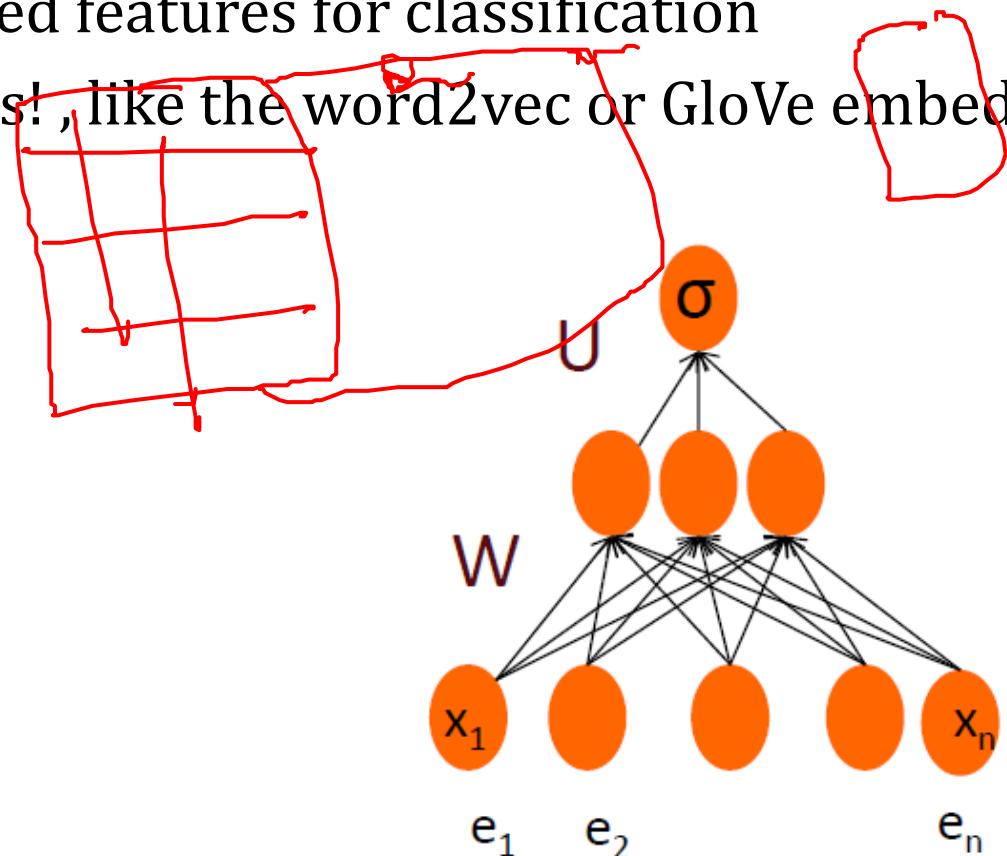
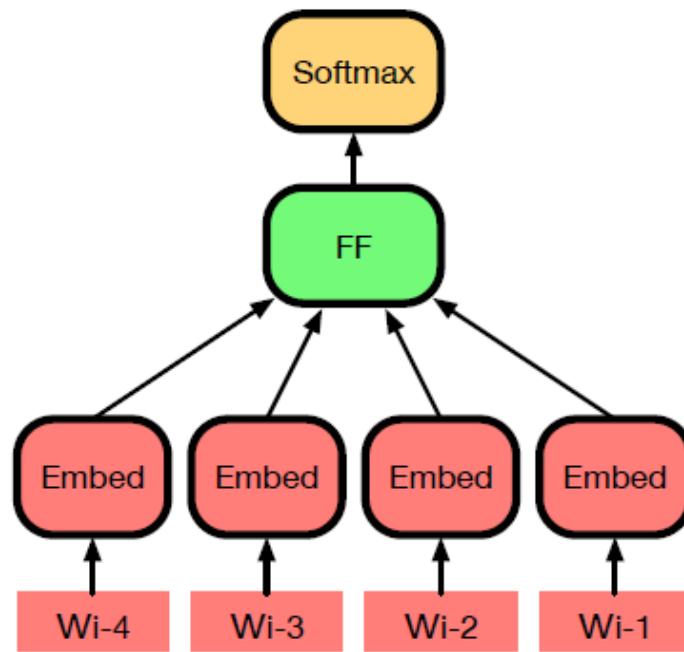
$$\in \mathcal{Z}_t$$

$$\begin{aligned}
 p(+|x) = P(Y = 1|x) &= \sigma(w \cdot x + b) \\
 &= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1) \\
 &= \sigma(.833) \\
 &= 0.70
 \end{aligned}$$

$$\begin{aligned}
 p(-|x) = P(Y = 0|x) &= 1 - \sigma(w \cdot x + b) \\
 &= 0.30
 \end{aligned}$$

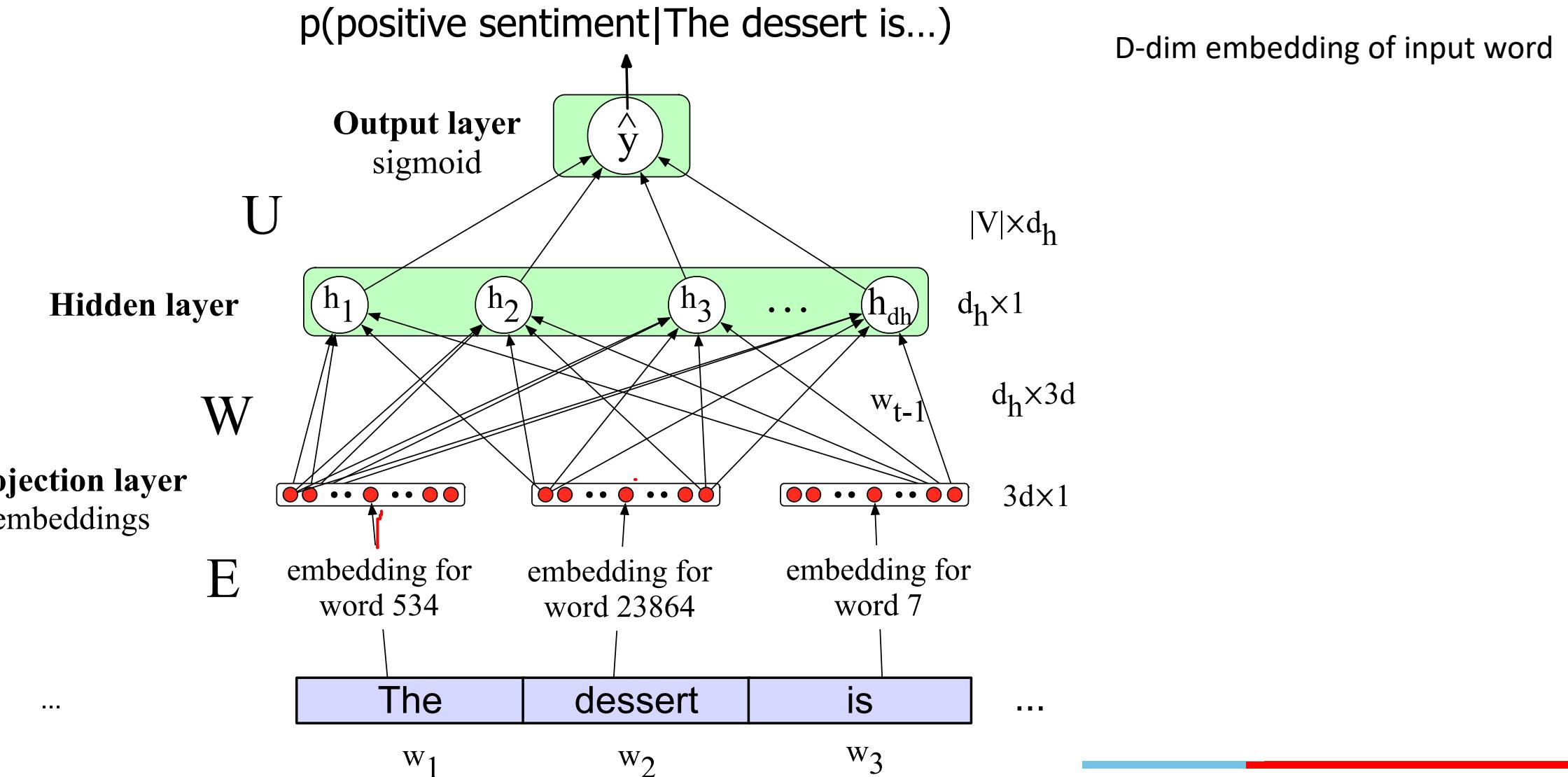
# Representation learning

- The real power of deep learning comes from the ability to **learn** features from the data
- Instead of using hand-built human-engineered features for classification
- Use learned representations like embeddings!, like the word2vec or GloVe embeddings



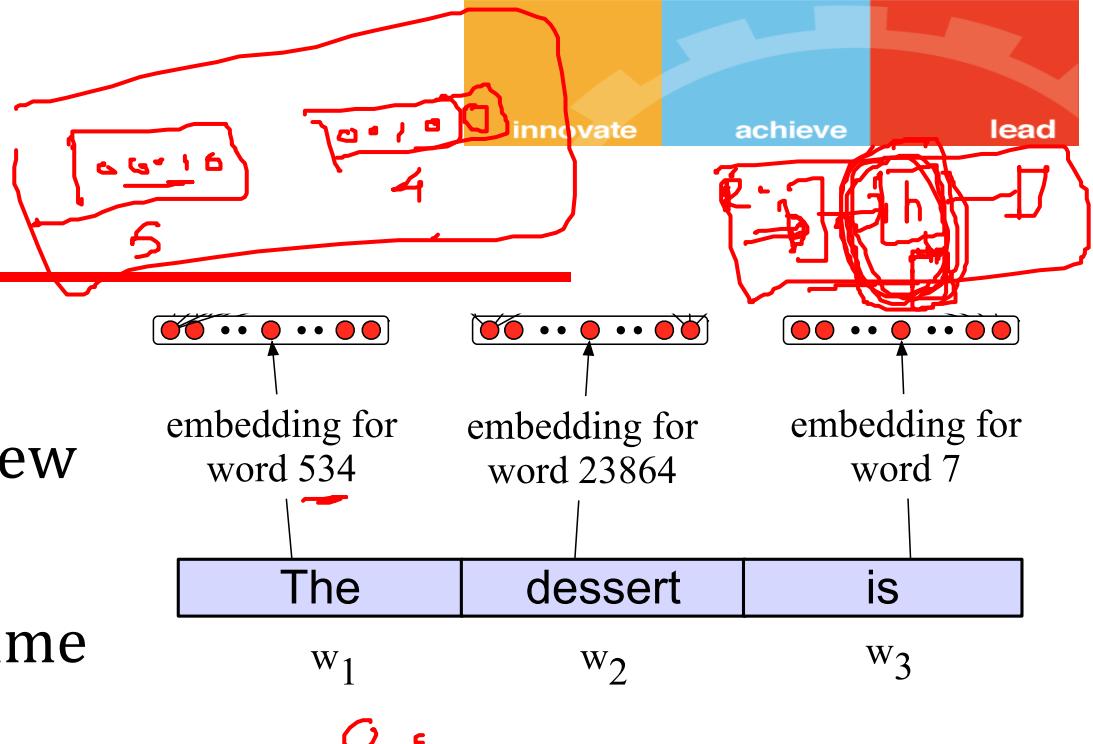
# Neural Net Classification with embeddings as input features

lead



# Issue: texts come in different sizes

- This assumes a fixed size length (3 words)!
  1. Make the input the length of the longest review
    - If shorter then pad with zero embeddings
    - Truncate if you get longer reviews at test time
  2. Create a single "sentence embedding" (the same dimensionality as a word -  $d$ ) to represent all the words
    - Take the mean of all the word embeddings
    - Take the element-wise max of all the word embeddings
      - For each dimension -  $d$ , pick the max value from all words



# Using Pool embeddings

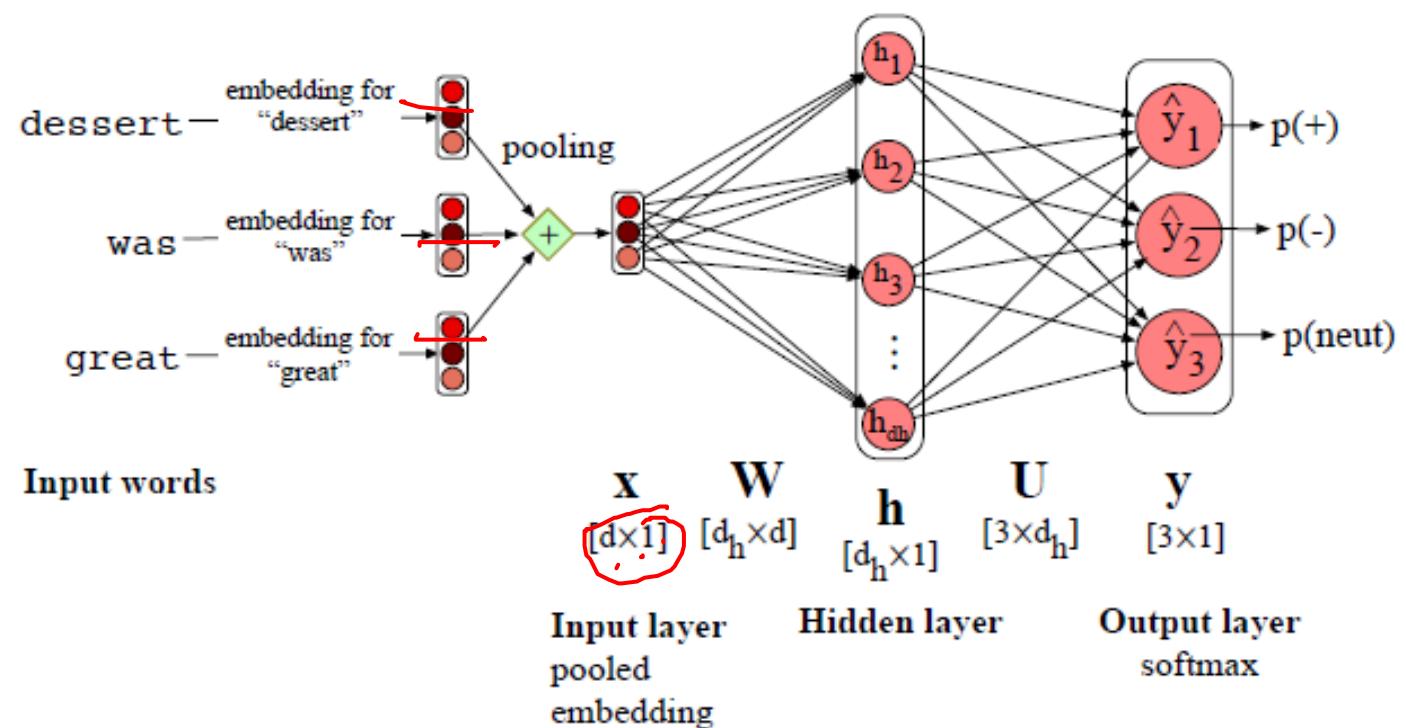
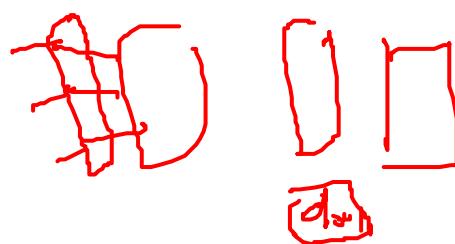


Figure 7.11 Feedforward sentiment analysis using a pooled embedding of the input words.

w<sub>1</sub>...w<sub>n</sub>: n input tokens  
 e(w<sub>1</sub>)....e(w<sub>n</sub>) : n embedding (each d-dim)  
 X: single embedding of d-dim, mean of all embedding

$$x_{mean} = \frac{1}{n} \sum_{i=1}^n e(w_i)$$

$$\begin{aligned} x &= \text{mean}(e(w_1), e(w_2), \dots, e(w_n)) \\ h &= \sigma(Wx + b) \\ z &= Uh \\ \hat{y} &= \text{softmax}(z) \end{aligned}$$

vectorising

$$\begin{aligned} H &= \sigma(XW^\top + b) \\ Z &= HU^\top \\ \hat{Y} &= \text{softmax}(Z) \end{aligned}$$



# Neural Language Model

# Types of neural language model

- Statistical model
- Neural language model



The day will be

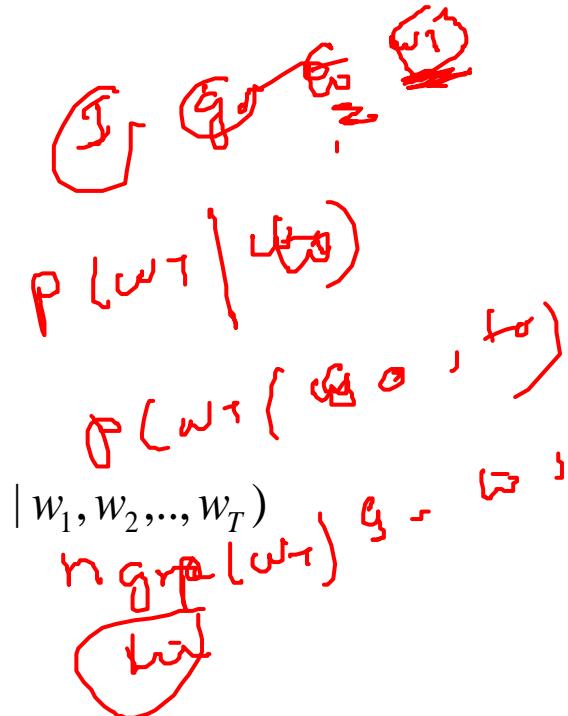
# Probabilistic Language Models

innovate

achieve

lead

- Probability of a sequence of words:  $P(W) = P(w_1, w_2, \dots, w_{t-1}, w_T)$
- **Conditional probability** of an upcoming word:  $P(w_T | w_1, w_2, \dots, w_{t-1})$
- Chain rule of probability:  $P(w_1, w_2, \dots, w_{t-1}, w_T) = P(w_1)P(w_2 | w_1)P(w_3 | w_1, w_2) \dots P(w_T | w_1, w_2, \dots, w_{t-1})$
- $(n-1)^{\text{th}}$  order Markov assumption  $P(w_1, w_2, \dots, w_{t-1}, w_T) = \prod_{t=1}^T P(w_t | w_1, w_2, \dots, w_{t-1})$
- Each  $p(w_i | w_{i-4}, w_{i-3}, w_{i-2}, w_{i-1})$  may not have enough statistics to estimate
  - we back off to  $p(w_i | w_{i-3}, w_{i-2}, w_{i-1})$ ,  $p(w_i | w_{i-2}, w_{i-1})$ , etc., all the way to  $p(w_i)$



# Example

day

**Have a good** ↗

weather

terrible

time

**4-gram language model:**

$p(\text{day} | \text{have a good}) =$

$$\frac{c(\text{have a good day})}{c(\text{have a good})}$$

$\tilde{P}(P | B)$

$\sim P(P, B)$

$C(I)$

A diagram illustrating a 4-gram language model calculation. The input sentence "Have a good day" is shown with "day" highlighted in green. The model is calculated as the ratio of the count of the 4-gram "have a good day" to the count of the 3-gram "have a good". Red annotations include circled terms like "have a good", "day", and "P(B|A)", and a large red X over the term "P(P, B)". A red bracket groups the first three words of the input.

# Curse of dimensionality

Consider this sentences

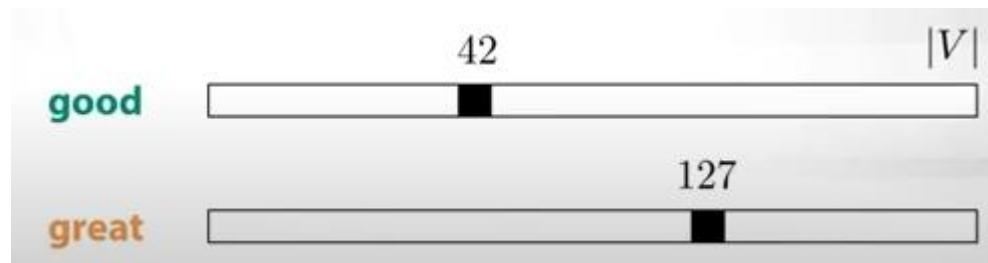
**Have a good day**

**Have a great day**

$$P[\theta \in U]$$

What happens in smoothing ?

Since  
= 0

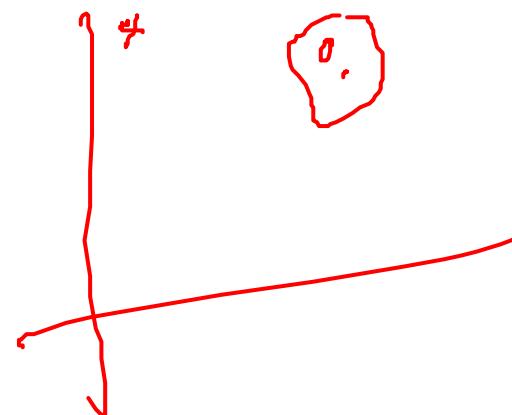


# How to generalize better

- Learn distributed representation of words
- What is distributed representation?
  - also known as embedding.
  - Eg:



*R - man + woman = Queen*



# Neural Probabilistic Language Models



- Instead of treating words as tokens, exploit semantic similarity
  - Learn a distributed representation of words that will allow sentences like these to be seen as similar

The cat is walking in the bedroom.  
A dog was walking in the room.  
The cat is running in a room.  
The dog was running in the bedroom.  
etc.
  - Use a neural net to represent the conditional probability function
$$P(w_t | w_{t-n}, w_{t-n+1}, \dots, w_{t-1})$$
  - Learn the word representation and the probability function simultaneously

# Neural Language Models (LMs)

---

- **Language Modeling:** Calculating the probability of the next word in a sequence given some history.
- We've seen N-gram based LMs
- But neural network LMs far outperform n-gram language models
- State-of-the-art neural LMs are based on more powerful neural network technology like **Transformers**
- But **simple feedforward LMs** can do almost as well!

# Simple feedforward Neural Language Models

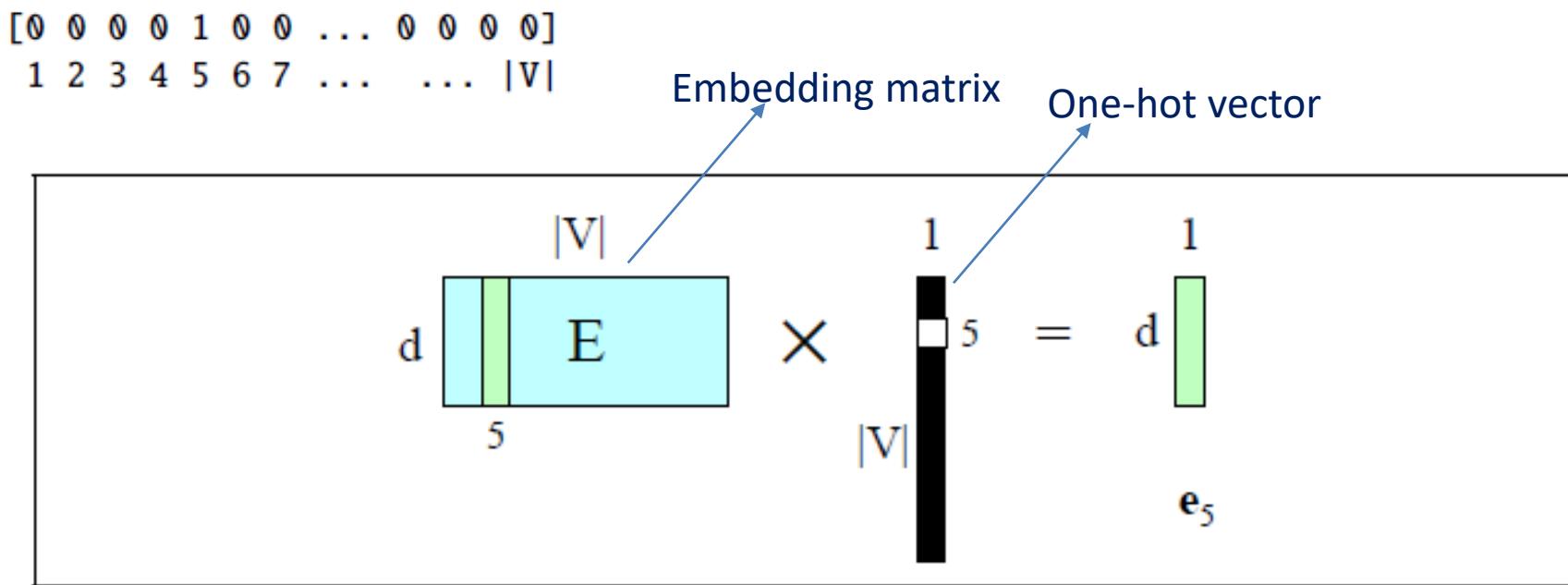
- **Task:** predict next word  $w_t$   
given prior words  $w_{t-1}, w_{t-2}, w_{t-3}, \dots$
- **Output :** probability distribution over possible next words.

$$P(w_t | w_1, \dots, w_{t-1}) \approx P(w_t | w_{t-N+1}, \dots, w_{t-1})$$

- **Problem:** Now we're dealing with sequences of arbitrary length.
- **Solution:** Sliding windows (of fixed length)

# Word Embeddings

- one-hot vector representation , e.g., dog =  $(0,0,0,0,1,0,0,0,0,\dots)$ , cat =  $(0,0,0,0,0,0,0,1,0,\dots)$
- Represent each of the N previous words as a one-hot vector of length  $|V|$  one-hot vector , i.e., with one dimension for each word in the vocabulary
- word “toothpaste”, supposing it is  $V_5$ , i.e., index 5 in the vocabulary,  $x_5 = 1$ , and  $x_i = 0 \quad i \forall \neq 5$ ,



**Figure 7.12** Selecting the embedding vector for word  $V_5$  by multiplying the embedding matrix  $E$  with a one-hot vector with a 1 in index 5.

# Why Neural LMs work better than N-gram LMs

## embeddings

---

- Neural language models represent words in this prior context by their embeddings, rather than just by their word identity as used in n-gram language models.
- Using embeddings allows neural language models to generalize better to unseen data.

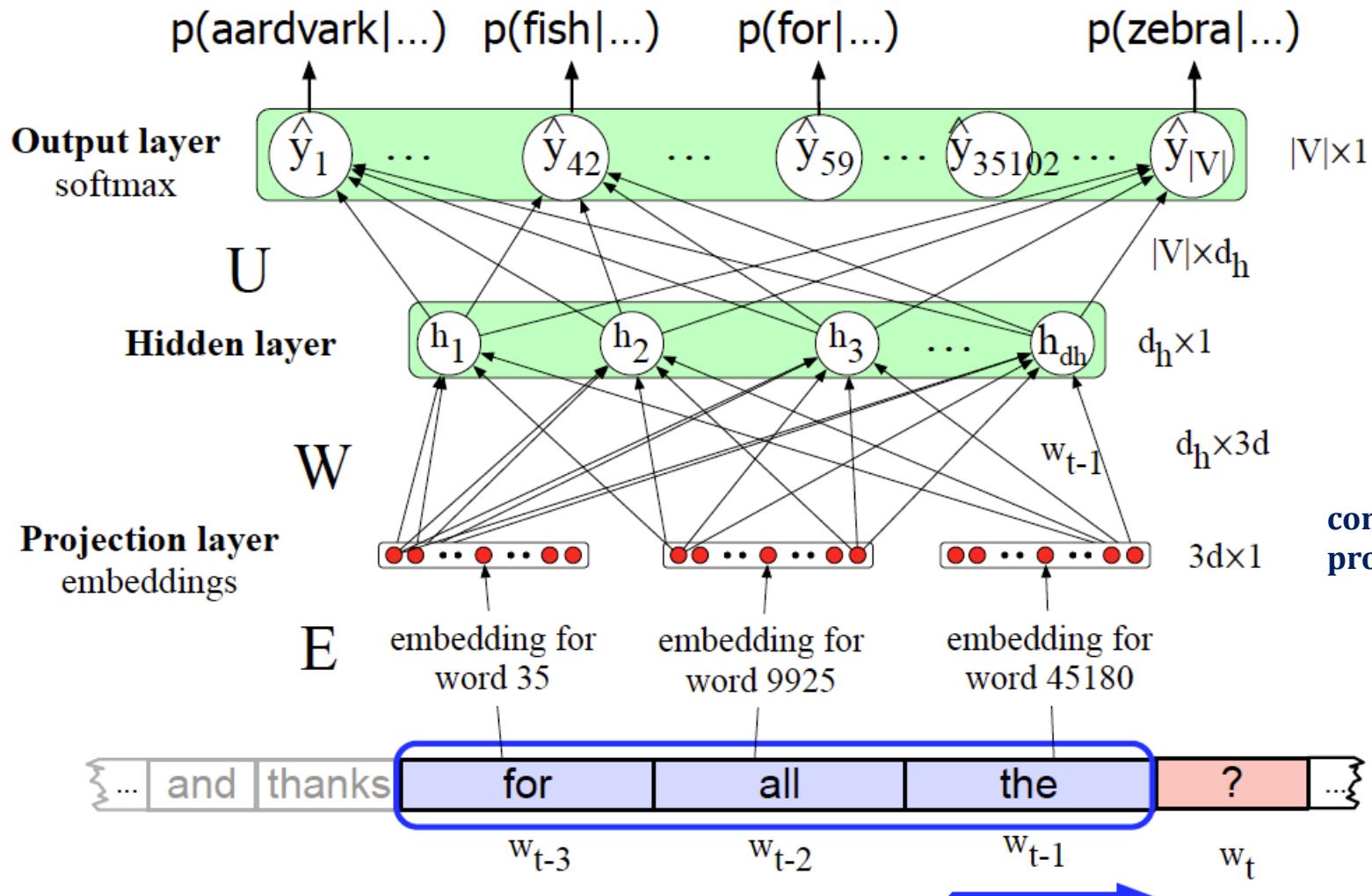
### Training data:

- We've seen: I have to make sure that the cat gets fed.
- Never seen: dog gets fed

### Test data:

- I forgot to make sure that the dog gets \_\_
- N-gram LM can't predict "fed"!
- Neural LM can use similarity of "cat" and "dog" embeddings to generalize and predict "fed" after dog

# Neural Language Model



Equations:

$$\begin{aligned} e &= [E_{x_{t-3}}; E_{x_{t-2}}; E_{x_{t-1}}] \\ h &= \sigma(We + b) \\ z &= Uh \\ \hat{y} &= \text{softmax}(z) \end{aligned}$$

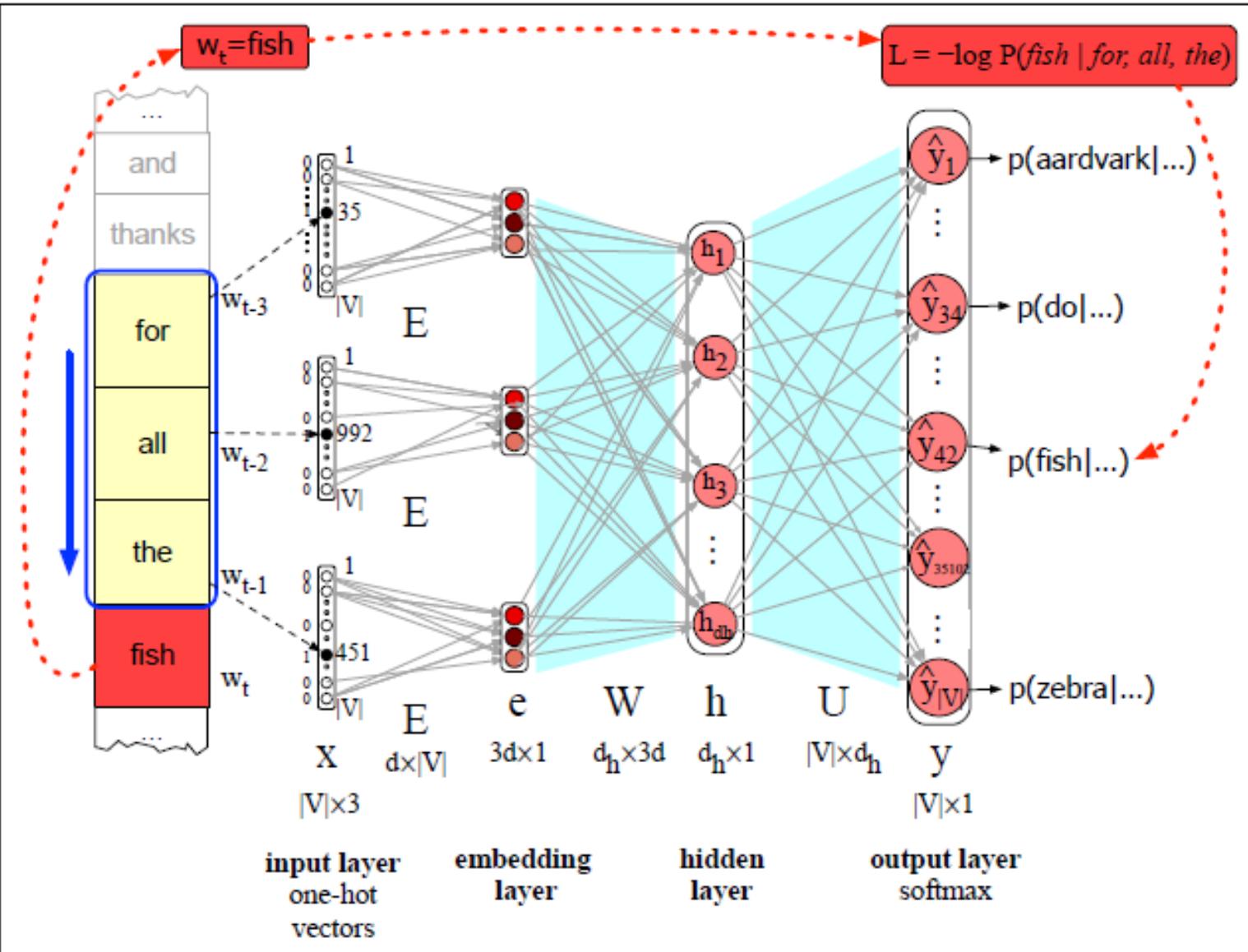
concatenate 3 embeddings for the 3 context words to produce the embedding layer  $e$

# Training the neural language model

---

- Two ways:
- Freeze the embedding layer E with initial word2vec values.
  - Freezing means we use **word2vec or some other pretraining algorithm** to compute the initial embedding matrix E, and then hold it constant while we only modify W, U, and b, i.e., we don't update E during language model training
- Learn the embeddings simultaneously with training the network.
  - Useful when the task the network is designed for (like sentiment classification, translation, or parsing) places strong constraints on what makes a good representation for words.

# Training the neural language model



$$\theta = E, W, U, b.$$

one single embedding dictionary  $E$  that's shared among one-hot vectors

Training the parameters to minimize loss will result both in an algorithm for **language modeling (a word predictor)** but also a **new set of embeddings  $E$**  that can be used as word representations for other tasks.

Figure 7.18 Learning all the way back to embeddings. Again, the embedding matrix  $E$  is

# Training the neural language model

- Input a very long text, concatenating all the sentences
- Starting with random weights
- Iteratively moving through the text predicting each word  $w_t$
- For language modeling, the classes are the words in the vocabulary,

$$y_i - \hat{y} = P(w_t | w_{t-n}, w_{t-n+1}, \dots, w_{t-1})$$

- At each word  $w_t$ , we use the cross-entropy (negative log likelihood) loss

$$L_{CE}(\hat{y}, y) = -\log \hat{y}_i, \quad (\text{where } i \text{ is the correct class}) \quad \Rightarrow \quad L_{CE} = -\log p(w_t | w_{t-1}, \dots, w_{t-n+1})$$

- Gradient Descent update:  $\theta^{s+1} = \theta^s - \eta \frac{\partial [-\log p(w_t | w_{t-1}, \dots, w_{t-n+1})]}{\partial \theta}$
- This gradient can be computed in any standard neural network framework which will then backpropagate through  $\theta = E, W, U, b$ .



# References

CH-7 - Speech and Language Processing by Daniel Jurafsky



**BITS** Pilani  
Pilani Campus

# Natural Language Processing

DSECL ZG565

Prof.Vijayalakshmi  
BITS-Pilani



**Session 3: POS tagging  
Date – 19<sup>th</sup> Mar2022  
Time – 9 am to 11 am**

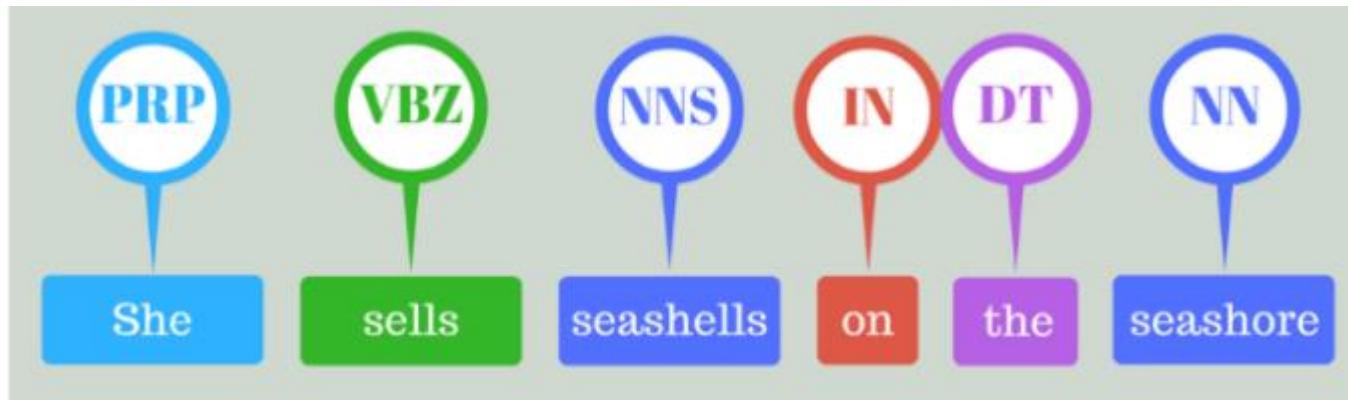
These slides are prepared by the instructor, with grateful acknowledgement of James Allen and many others who made their course materials freely available online.

# Session3:POS tagging

- ❖ What is Part of speech tagging
- ❖ Why POS tagging is required
- ❖ Application
- ❖ (Mostly) English Word Classes
- ❖ Tag set
  - Penn tree bank
- ❖ Part-of-Speech Tagging
- ❖ Different approaches
  - Rule based
  - Stochastic based
    - HMM Part-of-Speech Tagging
  - Hybrid system

# What is POS Tagging

- The process of assigning a part-of-speech or lexical class marker to each word in a sentence (and all sentences in a collection).



# Process

---

- ❖ List of all possible tag for each word in sentence

- ❖ Eg:

"People jump high".

People : Noun/Verb

jump : Noun/Verb

high : Noun/Verb/Adjective

- ❖ Choose best suitable tag sequence

start with probabilities.

---

# Why POS?

---

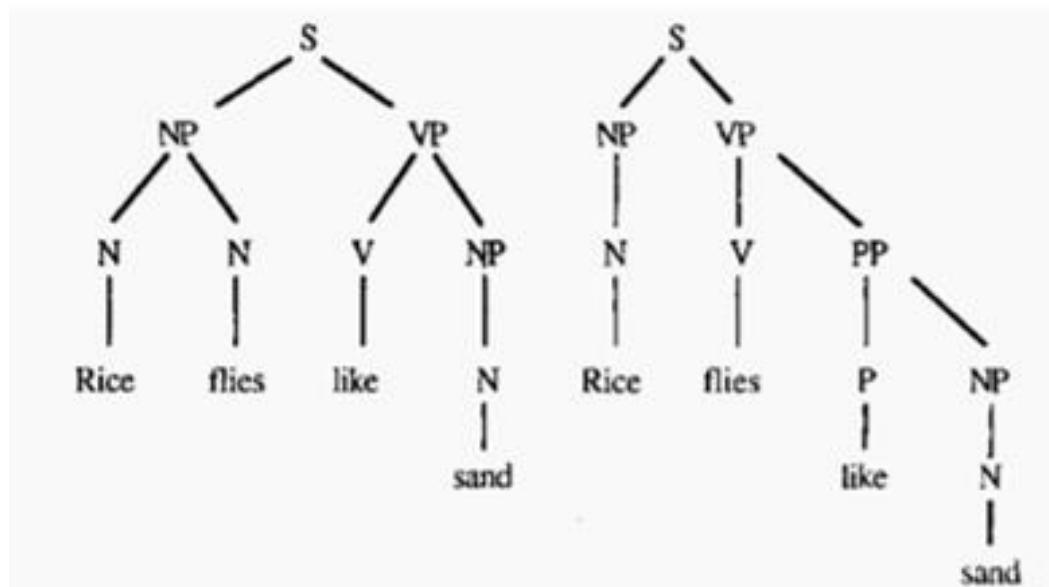
- ❖ First step in many applications
  - ❖ POS tell us a lot about word
  - ❖ Pronunciation depends on POS
  - ❖ To find named entities
  - ❖ Stemming
-

# Application of POS

## ❖ Parsing

Recovering syntactic structures requires correct  
POS tags

Eg:



# Information retrieval

---

## ❖ Word Disambiguation

-ability to determine which meaning of word is activated by the use of word in a particular context.

Eg:

- I can hear **bass** sound.
- He likes to eat grilled **bass**.

# Question answering system

---

- ❖ automatically answer questions posed by humans in a natural language.
- ❖ Eg: Does he eat bass?

# Why POS tagging is hard?

---

## ❖ Ambiguity

Eg:

- He will race/VB the car.
  - When will the race/NOUN end?
  - The boat floated/VBD ... down the river sank
- ❖ Average of ~2 parts of speech for each word

# Parts of Speech

---

- 8 (ish) traditional parts of speech
  - Noun, verb, adjective, preposition, adverb, article, interjection, pronoun, conjunction, etc
  - Called: parts-of-speech, lexical categories, word classes, morphological classes, lexical tags...
  - Lots of debate within linguistics about the number, nature, and universality of these
    - We'll completely ignore this debate.

# POS examples

---

- N noun *chair, bandwidth, pacing*
- V verb *study, debate, munch*
- ADJ adjective *purple, tall, ridiculous*
- ADV adverb *unfortunately, slowly*
- P preposition *of, by, to*
- PRO pronoun *I, me, mine*
- DET determiner *the, a, that, those*

# POS Tagging

- Assigning label to something or someone
- The process of assigning a part-of-speech to each word in a collection.

<u>WORD</u>	<u>tag</u>
the	DET
koala	N
put	V
the	DET
keys	N
on	P
the	DET
table	N

# Open and Closed Classes

---

- Open class: new ones can be created all the time
  - English has 4: Nouns, Verbs, Adjectives, Adverbs
  - Many languages have these 4, but not all!
- Closed class: a small fixed membership
  - Prepositions: of, in, by, ...
  - Auxiliaries: may, can, will had, been, ...
  - Pronouns: I, you, she, mine, his, them, ...
  - Usually **function words** (short common words which play a role in grammar)

# Closed Class Words

## Examples:

- prepositions: *on, under, over, ...*
- particles: *up, down, on, off, ...*
- determiners: *a, an, the, ...*
- pronouns: *she, who, I, ..*
- conjunctions: *and, but, or, ...*
- auxiliary verbs: *can, may should, ...*
- numerals: *one, two, three, third, ...*

# Tagsets

---

- A set of all POS tags used in a corpus is called a tag set
- There are various **standard tag sets** to choose from; some have a lot more tags than others
- The choice of tagset is based on the application
- Accurate tagging can be done with even large tag sets

# Penn tree bank

---

- ❖ Background
  - From the early 90s
  - Developed at the University of Pennsylvania
  - (Marcus, Santorini and Marcinkiewicz 1993)
- ❖ Size
  - 40000 training sentences
  - 2400 test sentences
- Genre
  - Mostly wall street journal news stories and some spoken conversations
- ❖ Importance
  - Helped launch modern automatic parsing methods.

# Penn TreeBank POS Tagset



Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	<i>and, but, or</i>	SYM	symbol	+,%,&
CD	cardinal number	<i>one, two, three</i>	TO	“to”	<i>to</i>
DT	determiner	<i>a, the</i>	UH	interjection	<i>ah, oops</i>
EX	existential ‘there’	<i>there</i>	VB	verb, base form	<i>eat</i>
FW	foreign word	<i>mea culpa</i>	VBD	verb, past tense	<i>ate</i>
IN	preposition/sub-conj	<i>of, in, by</i>	VBG	verb, gerund	<i>eating</i>
JJ	adjective	<i>yellow</i>	VBN	verb, past participle	<i>eaten</i>
JJR	adj., comparative	<i>bigger</i>	VBP	verb, non-3sg pres	<i>eat</i>
JJS	adj., superlative	<i>wildest</i>	VBZ	verb, 3sg pres	<i>eats</i>
LS	list item marker	<i>1, 2, One</i>	WDT	wh-determiner	<i>which, that</i>
MD	modal	<i>can, should</i>	WP	wh-pronoun	<i>what, who</i>
NN	noun, sing. or mass	<i>llama</i>	WP\$	possessive wh-	<i>whose</i>
NNS	noun, plural	<i>llamas</i>	WRB	wh-adverb	<i>how, where</i>
NNP	proper noun, singular	<i>IBM</i>	\$	dollar sign	\$
NNPS	proper noun, plural	<i>Carolinas</i>	#	pound sign	#
PDT	predeterminer	<i>all, both</i>	“	left quote	‘ or “
POS	possessive ending	<i>’s</i>	”	right quote	’ or ”
PRP	personal pronoun	<i>I, you, he</i>	(	left parenthesis	[, (, {, <
PRP\$	possessive pronoun	<i>your, one’s</i>	)	right parenthesis	], ), }, >
RB	adverb	<i>quickly, never</i>	,	comma	,
RBR	adverb, comparative	<i>faster</i>	.	sentence-final punc	. ! ?
RBS	adverb, superlative	<i>fastest</i>	:	mid-sentence punc	: ; ... --
RP	particle	<i>up, off</i>			

## Just for Fun...

---

- Using Penn Treebank tags, tag the following sentence from the Brown Corpus:
- The grand jury commented on a number of other topics.

## Just for Fun...

---

- Using Penn Treebank tags, tag the following sentence from the Brown Corpus:
- The/DT grand/JJ jury/NN commented/VBD on/IN a/DT number/NN of/IN other/JJ topics/NNS ./.

# POS Tagging-

## Choosing a Tag set

---

- ❖ There are so many parts of speech, potential distinctions we can draw
- ❖ To do POS tagging, we need to choose a standard set of tags to work with
- ❖ Could pick very coarse tagsets
  - ❖ N, V, Adj, Adv.
- ❖ More commonly used set is finer grained, the “Penn TreeBank tagset”, 45 tags
  - ❖ PRP\$, WRB, WP\$, VBG
- ❖ Even more fine-grained tagsets exist

# Approaches to POS Tagging

---

- ❖ Rule-based Approach
    - Uses handcrafted sets of rules to tag input sentences
  - ❖ Statistical approaches
    - Use training corpus to compute probability of a tag in a context-HMM tagger
  - ❖ Hybrid systems (e.g. Brill's transformation-based learning)
-

# Rule based POS tagging

---

- ❖ **First stage** – In the first stage, it uses a dictionary to assign each word a list of potential parts-of-speech.
  - ❖ **Second stage** – In the second stage, it uses large lists of hand-written disambiguation rules to sort down the list to a single part-of-speech for each word.
-

# Hidden Markov model

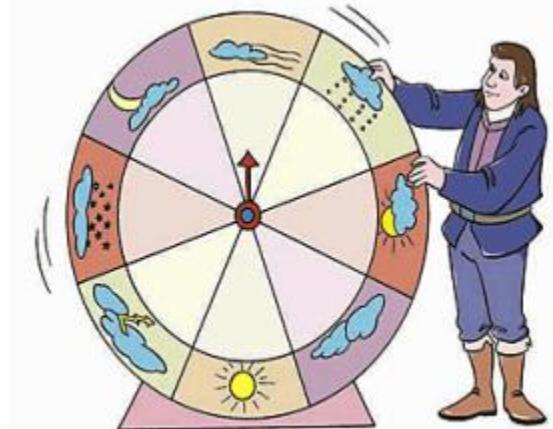
# Markov model /Markov chain

---

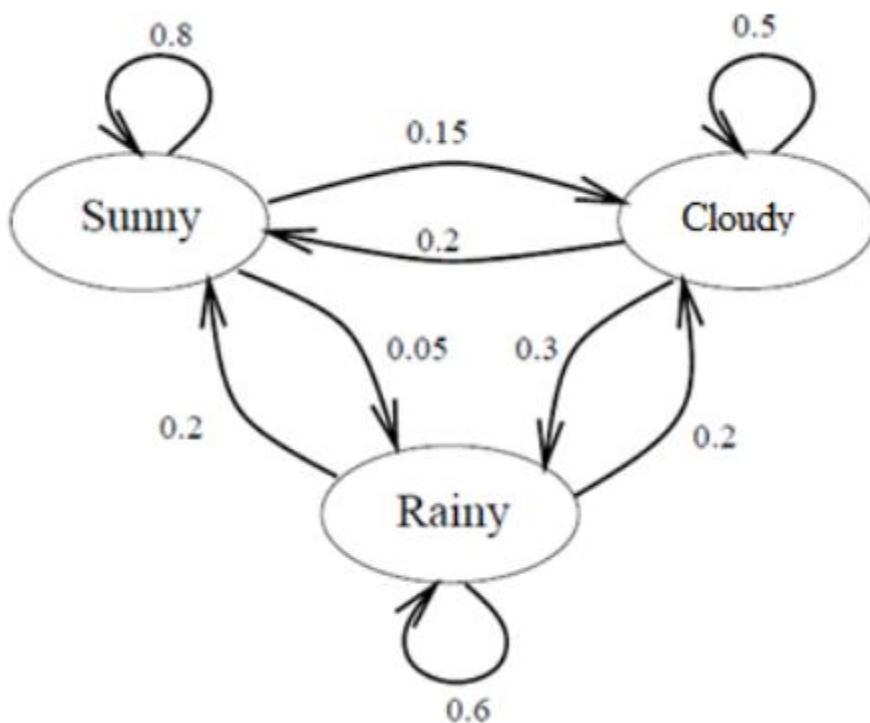
A Markov process is a process that generates a sequence of outcomes in such a way that the probability of the next outcome depends only on the current outcome and not on what happened earlier.

# MARKOV CHAIN: WEATHER EXAMPLE

- ❖ Design a Markov Chain to predict the weather of tomorrow using previous information of the past days.
- ❖ Our model has only 3 states:  
 $S = S_1, S_2, S_3$
- ❖  $S_1 = \text{Sunny}$  ,  $S_2 = \text{Rainy}$ ,  $S_3 = \text{Cloudy}$ .



# Contd..



$$\begin{aligned}
 P(\text{Sunny}|\text{Sunny}) &= 0.8 \\
 P(\text{Rainy}|\text{Sunny}) &= 0.05 \\
 P(\text{Cloudy}|\text{Sunny}) &= 0.15
 \end{aligned}
 \quad \left. \begin{array}{l} \\ \\ \end{array} \right\} 1$$
  

$$\begin{aligned}
 P(\text{Sunny}|\text{Rainy}) &= 0.2 \\
 P(\text{Rainy}|\text{Rainy}) &= 0.6 \\
 P(\text{Cloudy}|\text{Rainy}) &= 0.2
 \end{aligned}
 \quad \left. \begin{array}{l} \\ \\ \end{array} \right\} 1$$
  

$$\begin{aligned}
 P(\text{Sunny}|\text{Cloudy}) &= 0.2 \\
 P(\text{Rainy}|\text{Cloudy}) &= 0.3 \\
 P(\text{Cloudy}|\text{Cloudy}) &= 0.5
 \end{aligned}
 \quad \left. \begin{array}{l} \\ \\ \end{array} \right\} 1$$

Contd..

---

- ❖ state sequence notation:  $q_1, q_2, q_3, q_4, q_5, \dots, \dots$ , where  $q_i \in \{Sunny, Rainy, Cloudy\}$ .
- ❖ Markov Property

$$P(q_1, \dots, q_n) = \prod_{i=1}^n P(q_i | q_{i-1})$$

# Example

Given that today is Sunny, what's the probability that tomorrow is Sunny and the next day Rainy?

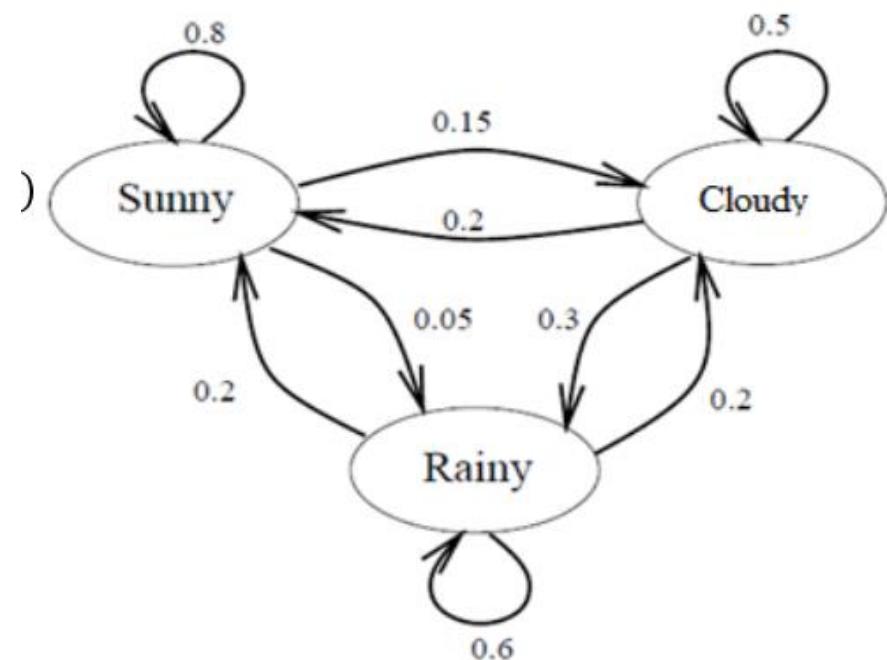
$$P(q_2, q_3 | q_1) = P(q_2 | q_1)P(q_3 | q_1, q_2)$$

$$= P(q_2 | q_1) P(q_3 | q_2)$$

$$= P(\text{Sunny}|\text{Sunny}) P(\text{Rainy}|\text{Sunny})$$

$$= (0.8)(0.05)$$

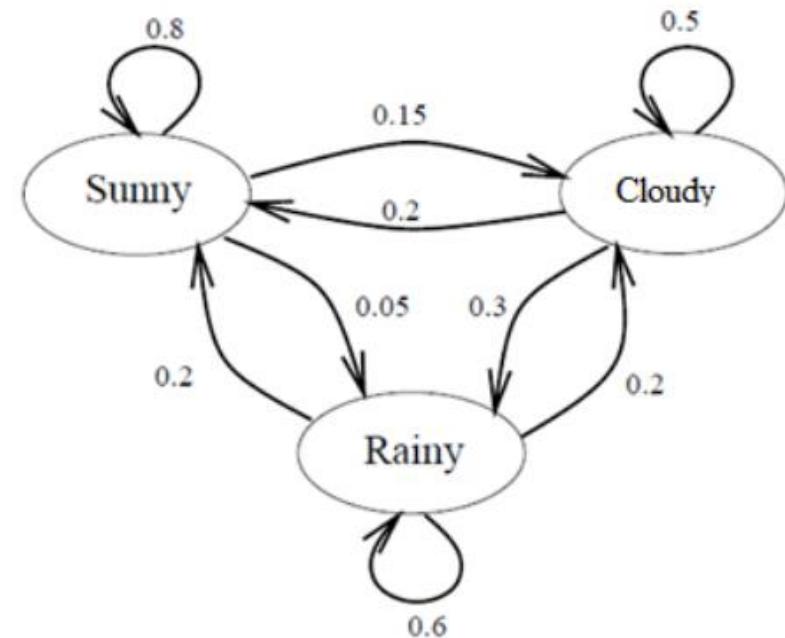
$$= 0.04$$



## Example2

Assume that yesterday's weather was Rainy, and today is Cloudy, what is the probability that tomorrow will be Sunny?

$$\begin{aligned}
 P(q_3|q_1, q_2) &= P(q_3|q_2) \\
 &= P(\text{Sunny}|\text{Cloudy}) \\
 &= 0.2
 \end{aligned}$$



# WHAT IS A HIDDEN MARKOV MODEL (HMM)?

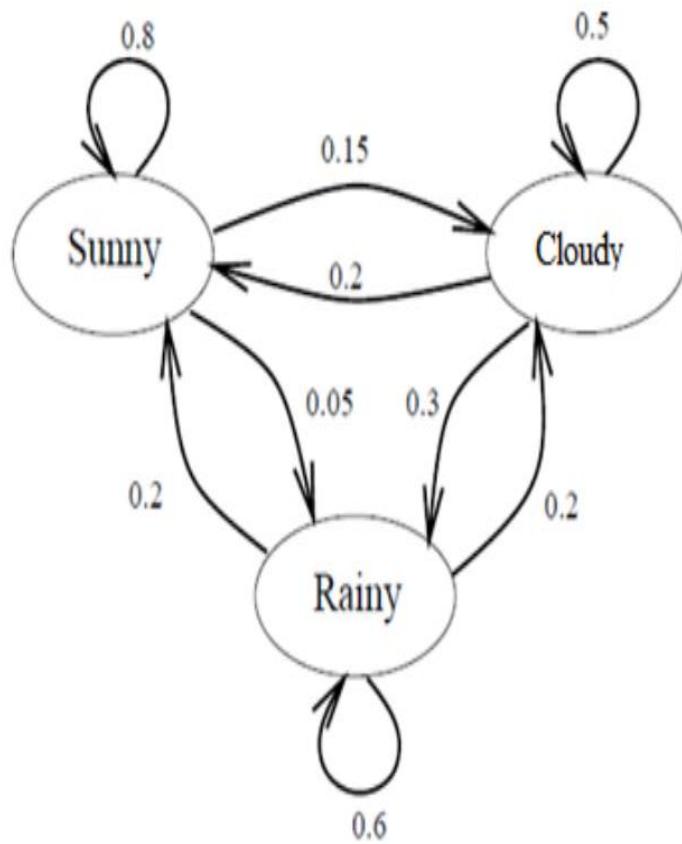
---



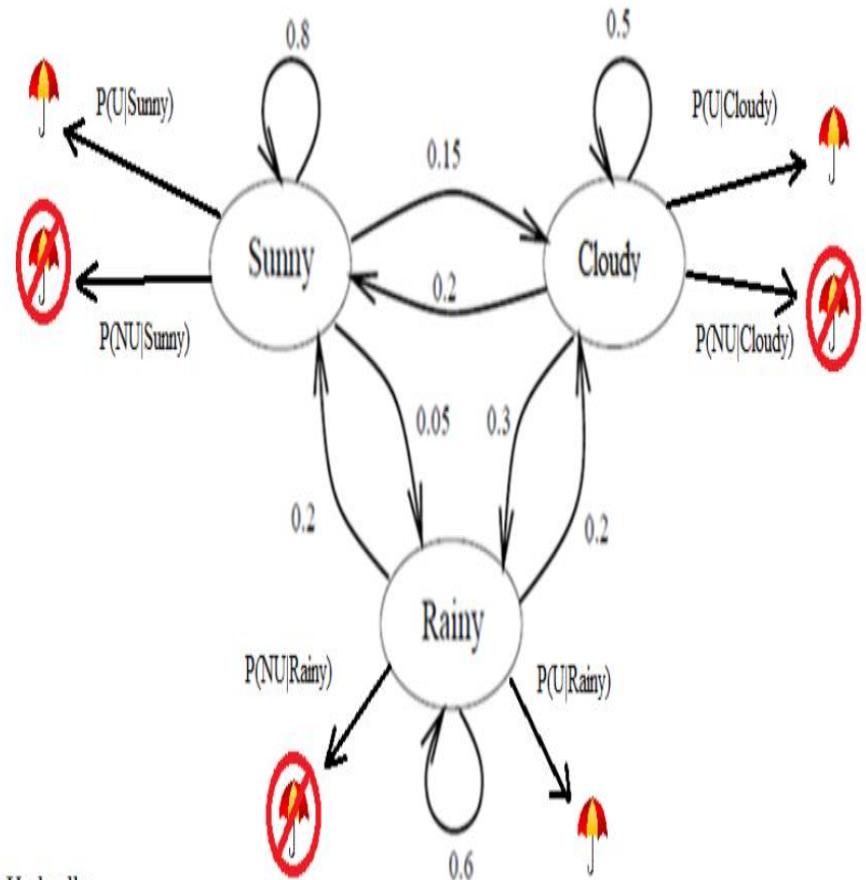
- ❖ A Hidden Markov Model, is a stochastic model where the states of the model are hidden. Each state can emit an output which is observed.
  - ❖ Imagine: You were locked in a room for several days and you were asked about the weather outside. The only piece of evidence you have is whether the person who comes into the room bringing your daily meal is carrying an umbrella or not.
  - What is hidden? Sunny, Rainy, Cloudy
  - What can you observe? Umbrella or Not
-

# Markov chain Vs HMM

## Markov chain



## HMM



# Hidden Markov Models (Formal)

---

- States  $Q = q_1, q_2 \dots q_N;$
- Observations  $O = o_1, o_2 \dots o_N;$
- Transition probabilities
  - Transition probability matrix  $A = \{a_{ij}\}$   
 $a_{ij} = P(q_t = j | q_{t-1} = i) \quad 1 \leq i, j \leq N$
- Emission Probability /Output probability
  - Output probability matrix  $B = \{b_i(k)\}$   
 $b_i(k) = P(X_t = o_k | q_t = i)$
- Special initial probability vector  $\pi$   
 $\rho_i = P(q_1 = i) \quad 1 \leq i \leq N$

# First-Order HMM Assumptions

- **Markov assumption: probability of a state depends only on the state that precedes it**

$$P(q_i | q_1 \dots q_{i-1}) = P(q_i | q_{i-1})$$

# How to build a second-order HMM?

---

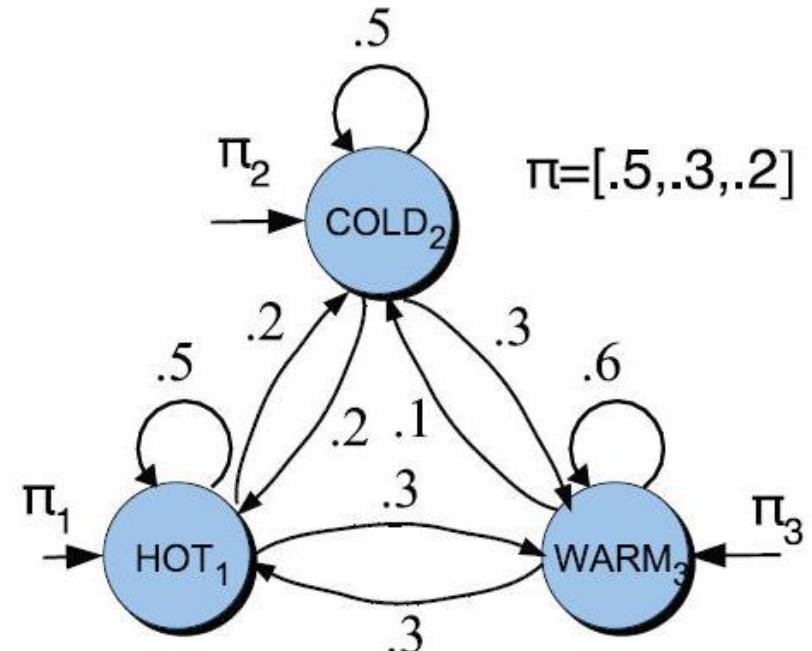
- Second-order HMM
  - Current state only depends on previous 2 states
- Example
  - Trigram model over POS tags
  - $P(\mathbf{t}) = \prod_{i=1}^n P(t_i | t_{i-1}, t_{i-2})$
  - $P(\mathbf{w}, \mathbf{t}) = \prod_{i=1}^n P(t_i | t_{i-1}, t_{i-2})P(w_i | t_i)$

# Markov Chain for Weather

- What is the probability of 4 consecutive warm days?

- Sequence is  
warm-warm-warm-warm
- And state sequence is  
3-3-3-3
- $P(3,3,3,3) =$

$$-\pi_3 a_{33} a_{33} a_{33} a_{33} = 0.2 \times (0.6)^3 = 0.0432$$



# POS Tagging as Sequence Classification

---



- We are given a sentence (an “observation” or “sequence of observations”)
  - *Secretariat is expected to race tomorrow*
- What is the best sequence of tags that corresponds to this sequence of observations?
- Probabilistic view
  - Consider all possible sequences of tags
  - Out of this universe of sequences, choose the tag sequence which is most probable given the observation sequence of n words  $w_1 \dots w_n$ .

# Probabilistic Sequence Models

---

- Probabilistic sequence models allow integrating uncertainty over multiple, interdependent classifications and collectively determine the most likely global assignment.
- standard model
  - Hidden Markov Model (HMM)

# How you predict the tags?

---

- Two types of information are useful
  - Relations between words and tags
  - Relations between tags and tags
    - DT NN, DT JJ NN...

# Statistical POS Tagging

- We want, out of all sequences of n tags  $t_1 \dots t_n$  the single tag sequence such that

$P(t_1 \dots t_n | w_1 \dots w_n)$  is highest.

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

- Hat ^ means “our estimate of the best one”
- Argmax<sub>x</sub> f(x) means “the x such that f(x) is maximized”

- ❖ This equation should give us the best tag sequence

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

- ❖ But how to make it operational? How to compute this value?
- ❖ Intuition of Bayesian inference:
  - Use Bayes rule to transform this equation into a set of probabilities that are easier to compute (and give the right answer)

# Using Bayes Rule

---

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} P(t_1^n | w_1^n) = \text{PROB}(T_1, \dots, T_n | w_1, \dots, w_n)$$

Estimating the above takes far too much data. Need to do some reasonable approximations.

## Bayes Rule:

$$\text{PROB}(A | B) = \text{PROB}(B | A) * \text{PROB}(A) / \text{PROB}(B)$$

## Rewriting:

$$\text{PROB}(w_1, \dots, w_n | T_1, \dots, T_n) * \text{PROB}(T_1, \dots, T_n) / \text{PROB}(w_1, \dots, w_n)$$

---

Contd..

---

$$= \text{PROB}(w_1, \dots, w_n \mid T_1, \dots, T_n) * \text{PROB}(T_1, \dots, T_n) / \\ \text{PROB}(w_1, \dots, w_n)$$

$$= \text{PROB}(w_1, \dots, w_n \mid T_1, \dots, T_n) * \text{PROB}(T_1, \dots, T_n)$$

# Independent assumptions

---

So, we want to find the sequence of tags that maximizes

$$\text{PROB}(T_1, \dots, T_n) * \text{PROB}(w_1, \dots, w_n | T_1, \dots, T_n)$$

- ❖ For Tags – use bigram probabilities

$$\text{PROB}(T_1, \dots, T_n) \approx \pi_{i=1,n} \text{PROB}(T_i | T_{i-1})$$

$$\text{PROB}(ART \ N \ V \ N) \approx \text{PROB}(ART | \Phi) * \text{PROB}(N | ART) * \text{PROB}(V | N) * \text{PROB}(N | V)$$

- ❖ For second probability: assume word tag is independent of words around it:

$$\text{PROB}(w_1, \dots, w_n | T_1, \dots, T_n) \approx \pi_{i=1,n} \text{PROB}(w_i | T_i)$$

# POS formula

---

- Find the sequence of tags that maximizes:

$$\pi_{i=1,n} \text{PROB}(T_i | T_{i-1}) * \text{PROB}(w_i | T_i)$$

# POS Tagging using HMM

- States  $T = t_1, t_2 \dots t_N;$
- Observations  $W = w_1, w_2 \dots w_N;$ 
  - Each observation is a symbol from a vocabulary  $V = \{v_1, v_2, \dots v_V\}$
- Transition probabilities
  - Transition probability matrix  $A = \{a_{ij}\}$ 
$$a_{ij} = P(t_i = j \mid t_{i-1} = i) \quad 1 \leq i, j \leq N$$
- Observation likelihoods
  - Output probability matrix  $B = \{b_i(k)\}$ 
$$b_i(k) = P(w_i = v_k \mid t_i = i)$$
- Special initial probability vector  $\pi$ 
$$\pi_i = P(t_1 = i) \quad 1 \leq i \leq N$$

---

## 1. State transition probabilities -- $p(t_i | t_{i-1})$

- State-to-state transition probabilities

$$P(t_i | t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

## 2. Observation/Emission probabilities -- $p(w_i | t_i)$

- Probabilities of observing various values at a given state

$$P(w_i | t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

# Two Kinds of Probabilities

## 1. Tag transition probabilities -- $p(t_i|t_{i-1})$

- Determiners likely to precede adjs and nouns
  - That/DT flight/NN
  - The/DT yellow/JJ hat/NN
  - So we expect  $P(NN|DT)$  and  $P(JJ|DT)$  to be high
- Compute  $P(NN|DT)$  by counting in a labeled corpus:

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

$$P(NN|DT) = \frac{C(DT, NN)}{C(DT)} = \frac{56,509}{116,454} = .49$$

# Two Kinds of Probabilities

## 2. Word likelihood/emission probabilities

$p(w_i|t_i)$

- VBZ (3sg Pres Verb) likely to be “is”
- Compute  $P(is|VBZ)$  by counting in a labeled corpus:

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

$$P(is|VBZ) = \frac{C(VBZ, is)}{C(VBZ)} = \frac{10,073}{21,627} = .47$$

# Sample Probabilities



## ***Bigram Probabilities***

Bigram( $T_i, T_j$ )	Count( $i, i + 1$ )	Prob( $T_j T_i$ )
$\emptyset, \text{ART}$	213	.71 (213/300)
$\emptyset, \text{N}$	87	.29 (87/300)
$\emptyset, \text{V}$	10	.03 (10/300)
$\text{ART}, \text{N}$	633	1
$\text{N}, \text{V}$	358	.32
$\text{N}, \text{N}$	108	.10
$\text{N}, \text{P}$	366	.33
$\text{V}, \text{N}$	134	.37
$\text{V}, \text{P}$	150	.42
$\text{V}, \text{ART}$	194	.54
$\text{P}, \text{ART}$	226	.62
$\text{P}, \text{N}$	140	.38
$\text{V}, \text{V}$	30	.08

## ***Tag Frequencies***

$\Phi$	ART	N	V	P
300	633	1102	358	366

# Sample Lexical Generation Probabilities

- $P(\text{an} \mid \text{ART})$  .36
- $P(\text{an} \mid \text{N})$  .001
- $P(\text{flies} \mid \text{N})$  .025
- $P(\text{flies} \mid \text{V})$  .076
- $P(\text{time} \mid \text{N})$  .063
- $P(\text{time} \mid \text{V})$  .012
- $P(\text{arrow} \mid \text{N})$  .076
- $P(\text{like} \mid \text{N})$  .012
- $P(\text{like} \mid \text{V})$  .10

# Two types of stochastic POS tagging

---

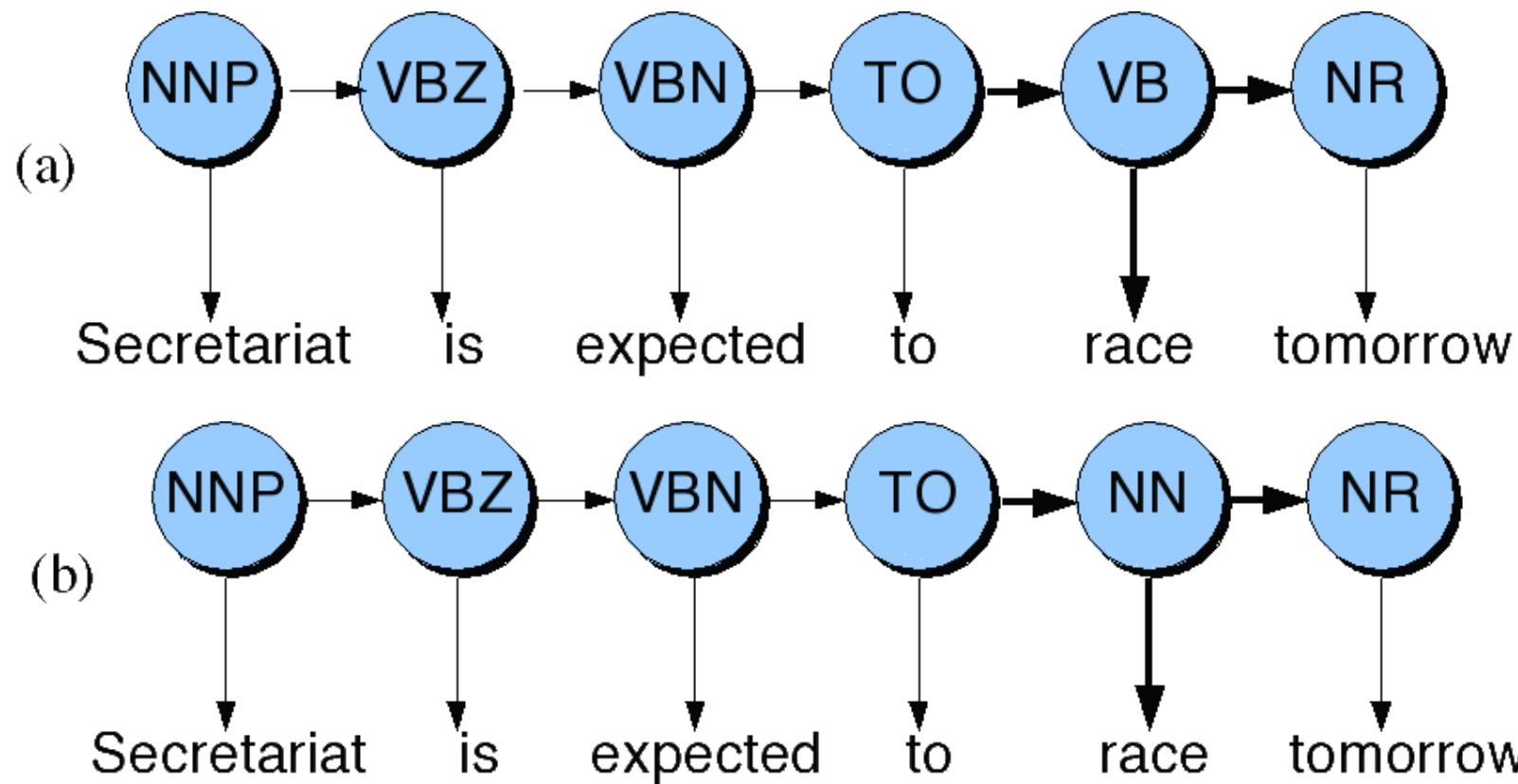
- ❖ Word frequency tagging
  - based on the probability that a word occurs with a particular tag.
- ❖ Tag sequence probabilities
  - calculates the probability of a given sequence of tags occurring.

## Example1-Some Data on race

---

- Secretariat/NNP is/VBZ expected/VBN to/TO race/VB tomorrow/NR
- People/NNS continue/VB to/TO inquire/VB the/DT reason/NN for/IN the/DT race/NN for/IN outer/JJ space/NN
- How do we pick the right tag for race in new data?

# Disambiguating **to race tomorrow**



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

- $P(\text{NN} | \text{TO}) = .00047$
- $P(\text{VB} | \text{TO}) = .83$
- $P(\text{race} | \text{NN}) = .00057$
- $P(\text{race} | \text{VB}) = .00012$
- $P(\text{NR} | \text{VB}) = .0027$
- $P(\text{NR} | \text{NN}) = .0012$
- $P(\text{VB} | \text{TO})P(\text{NR} | \text{VB})P(\text{race} | \text{VB}) = .00000027$
- $P(\text{NN} | \text{TO})P(\text{NR} | \text{NN})P(\text{race} | \text{NN}) = .0000000032$
- So we (correctly) choose the verb reading

## Example2:Statistical POS tagging- Whole tag sequence

---

- What is the most likely sequence of tags for the given sequence of words  $w$

$$\begin{aligned}
 \operatorname{argmax}_{\mathbf{t}} P(\mathbf{t}|\mathbf{w}) &= \operatorname{argmax}_{\mathbf{t}} \frac{P(\mathbf{t}, \mathbf{w})}{P(\mathbf{w})} \\
 &= \operatorname{argmax}_{\mathbf{t}} P(\mathbf{t}, \mathbf{w}) \\
 &= \operatorname{argmax}_{\mathbf{t}} P(\mathbf{t})P(\mathbf{w}|\mathbf{t})
 \end{aligned}$$

$$\begin{aligned}
 &P(\text{ DT JJ NN } | \text{ a smart dog}) \\
 &= P(\text{DD JJ NN a smart dog}) / P(\text{a smart dog}) \\
 &= P(\text{DD JJ NN}) P(\text{a smart dog} | \text{ DD JJ NN })
 \end{aligned}$$

# Tag Transition Probability

- Joint probability  $P(\mathbf{t}, \mathbf{w}) = P(\mathbf{t})P(\mathbf{w}|\mathbf{t})$
- $P(\mathbf{t}) = P(t_1, t_2, \dots, t_n)$ 
 $= P(t_1)P(t_2 | t_1)P(t_3 | t_2, t_1) \dots P(t_n | t_1 \dots t_{n-1})$ 
 $\sim P(t_1)P(t_2 | t_1)P(t_3 | t_2) \dots P(t_n | t_{n-1})$ 
 $= \prod_{i=1}^n P(t_i | t_{i-1})$ 
 $= P(\text{DD} | \text{start}) P(\text{JJ} | \text{DD}) P(\text{NN} | \text{JJ})$

Markov assumption

- Bigram model over POS tags!  
 (similarly, we can define a n-gram model over POS tags, usually we called high-order HMM)

# Word likelihood /Emission Probability

---

- Joint probability  $P(\mathbf{t}, \mathbf{w}) = P(\mathbf{t})P(\mathbf{w}|\mathbf{t})$
  - Assume words only depend on their POS-tag
  - $P(\mathbf{w}|\mathbf{t}) \sim P(w_1 | t_1)P(w_2 | t_2) \dots P(w_n | t_n)$
- Independent assumption
- $$= \prod_{i=1}^n P(w_i | t_i)$$

i.e.,  $P(\text{a smart dog} | \text{ DD JJ NN })$

$$= P(\text{a} | \text{ DD}) P(\text{smart} | \text{ JJ }) P(\text{ dog} | \text{ NN })$$

# Put them together

---

- Joint probability  $P(\mathbf{t}, \mathbf{w}) = P(\mathbf{t})P(\mathbf{w}|\mathbf{t})$
- $P(\mathbf{t}, \mathbf{w})$

$$= P(t_1)P(t_2 | t_1)P(t_3 | t_2) \dots P(t_n | t_{n-1})$$

$$P(w_1 | t_1)P(w_2 | t_2) \dots P(w_n | t_n)$$

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

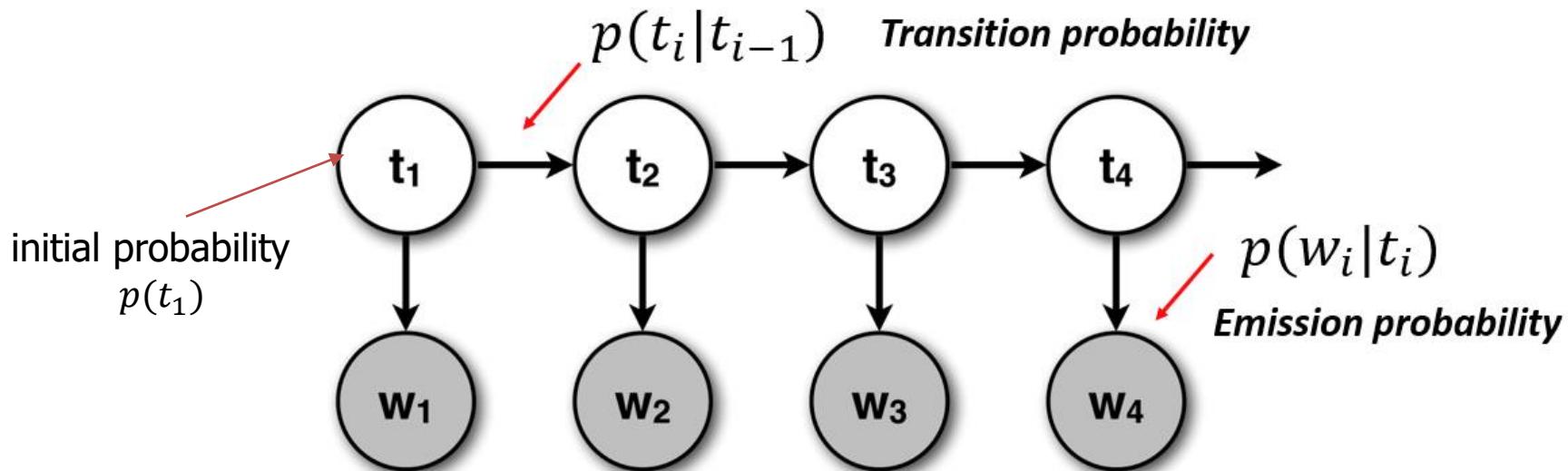
e.g.,  $P(\text{a smart dog , DD JJ NN })$

$$= P(\text{a} | \text{DD}) P(\text{smart} | \text{JJ}) P(\text{dog} | \text{NN})$$

$$P(\text{DD} | \text{start}) P(\text{JJ} | \text{DD}) P(\text{NN} | \text{JJ})$$

# Put them together

- Two independent assumptions
  - Approximate  $P(t)$  by a bi(or N)-gram model
  - Assume each word depends only on its POS tag



# Example1-HMM

Will can spot Mary

M V N N

Will Can spot Mary

N M V N

# Emission probabilities

	N	M	V
Mary	4	0	0
Jane	2	0	0
Will	1	3	0
Spot	2	0	1
Can	0	1	0
See	0	0	2
Pat	0	0	1

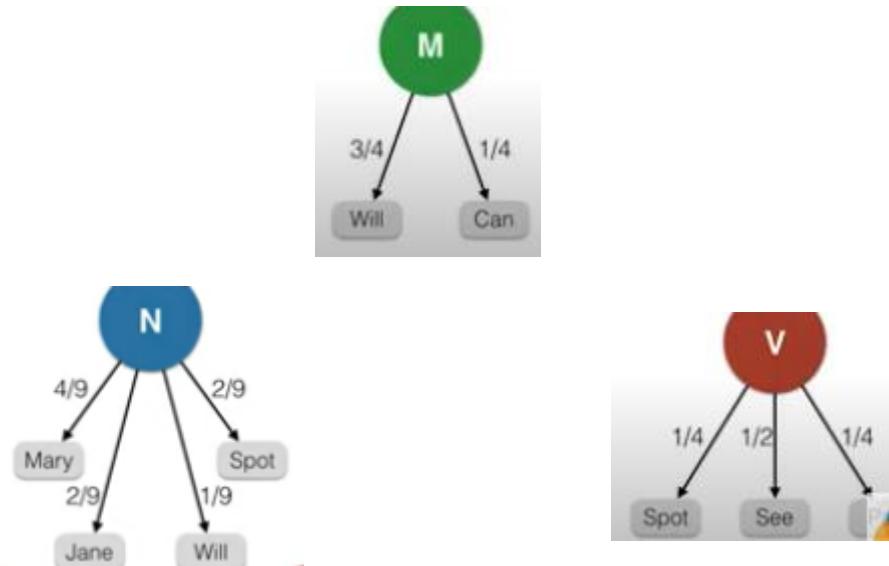
Mary Jane can see Will.

Spot will see Mary.

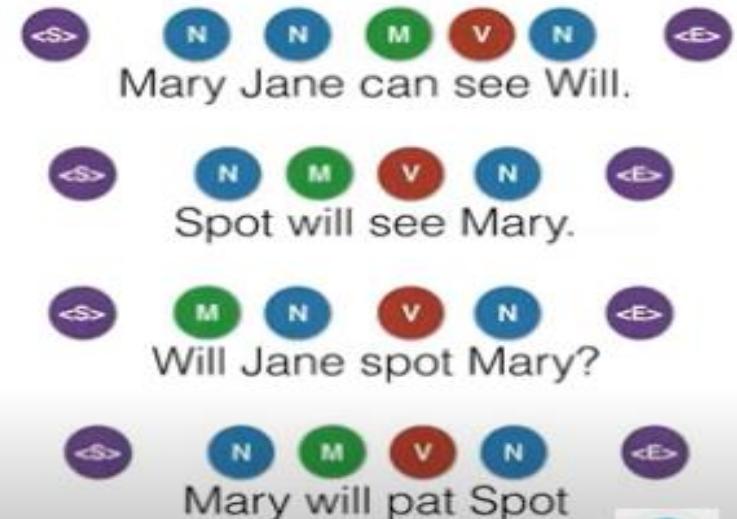
Will Jane spot Mary?

Mary will pat Spot

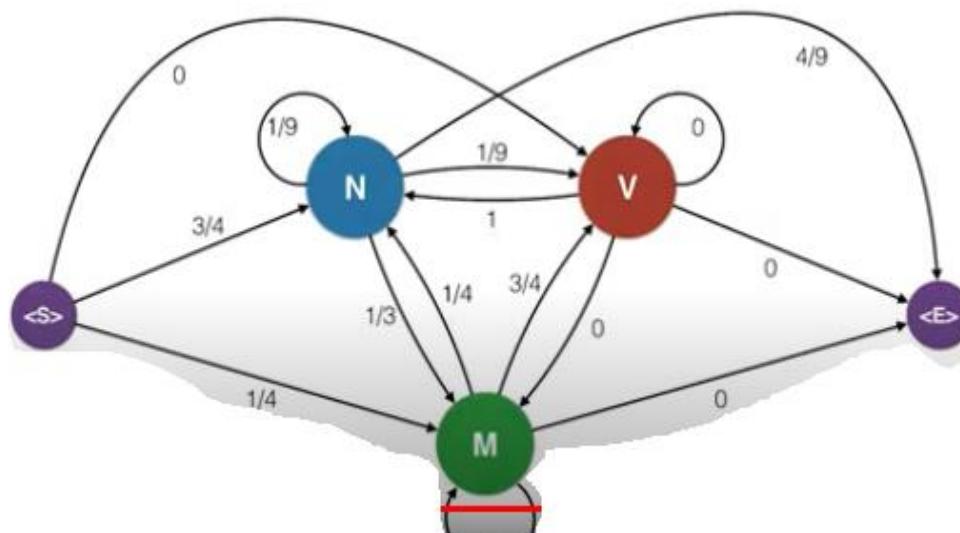
	N	M	V
Mary	4/9	0	0
Jane	2/9	0	0
Will	1/9	3/4	0
Spot	2/9	0	1/4
Can	0	1/4	0
See	0	0	1/2
Pat	0	0	1/4



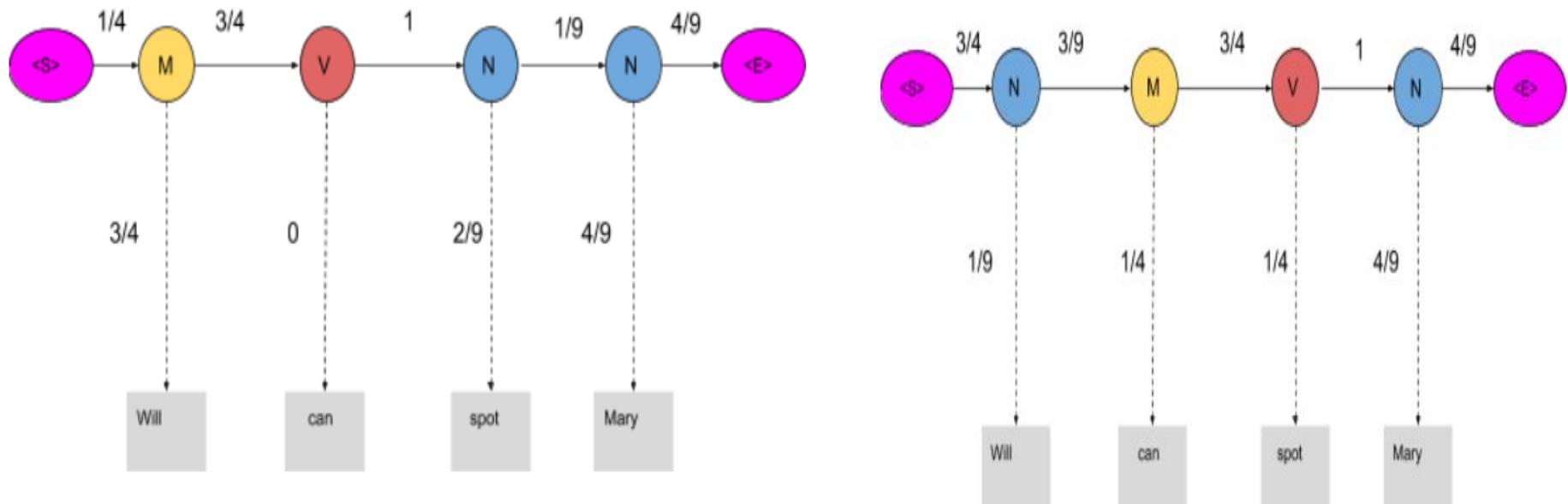
# Transition probabilities



	N	M	V	<E>
<S>	3	1	0	0
N	1	3	1	4
M	1	0	3	0
V	4	0	0	0



	N	M	V	<E>
<S>	3/4	1/4	0	0
N	1/9	1/3	1/9	4/9
M	1/4	0	3/4	0
V	1	0	0	0



$$1/4 * 3/4 * 3/4 * 0 * 1 * 2/9 * 1/9 * 4/9 * 4/9 = 0$$

$$3/4 * 1/9 * 3/9 * 1/4 * 3/4 * 1/4 * 1/4 * 4/9 * 4/9 = 0.00025720164$$

Correct sentence is Will/N can/M Spot/V Mary /N

# Thank You... 😊

- Q&A
- Suggestions / Feedback



# Natural Language Processing

## DSECL ZG565



**BITS** Pilani  
Pilani Campus

Prof.Vijayalakshmi Anand

BITS-Pilani



## **Session 4-Part-of-Speech Tagging (Viterbi ,Maximum entropy)**

**Date – 5<sup>th</sup> Dec 2021**

**Time – 9 am to 11am**

These slides are prepared by the instructor, with grateful acknowledgement of James Allen and many others who made their course materials freely available online.



- ❖ What is POS tagging
- ❖ Application of POS tagging
- ❖ Tag sets –standard tag set
- ❖ Approaches of POS tagging
- ❖ Introduction to HMM
- ❖ How HMM is used in POS tagging

# Session4-POS tagging(Viterbi ,Maximum entropy model)

---

- The Hidden Markov Model
- Likelihood Computation:
  - The Forward Algorithm
- Decoding: The Viterbi Algorithm
- Bidirectionality
- Maximum entropy model

# Hidden Markov Models

---

It is a **sequence model**.

Assigns a label or class to each unit in a sequence, thus mapping a **sequence of observations** to a **sequence of labels**.

Probabilistic sequence model: given a sequence of units (e.g. words, letters, morphemes, sentences), compute a probability distribution over possible sequences of labels and choose the best label sequence.

This is a kind of *generative* model.

# Hidden Markov Model (HMM)

---

Oftentimes we want to know what produced the sequence

– the **hidden sequence** for the **observed sequence**.

For example,

- Inferring the **words** (hidden) from **acoustic signal** (observed) in speech recognition
- Assigning **part-of-speech tags** (hidden) to a **sentence** (sequence of words) – **POS tagging**.
- Assigning named **entity categories** (hidden) to a **sentence** (sequence of words) – Named Entity Recognition.

# Definition of HMM

---

States  $Q = q_1, q_2 \dots q_N$ ;

Observations  $O = o_1, o_2 \dots o_N$ ;

- Each observation is a symbol from a vocabulary  $V = \{v_1, v_2, \dots v_V\}$

Transition probabilities

- Transition probability matrix  $A = \{a_{ij}\}$   
$$a_{ij} = P(q_t = j | q_{t-1} = i) \quad 1 \leq i, j \leq N$$

Observation likelihoods

- Output probability matrix  $B = \{b_i(k)\}$   
$$b_i(k) = P(X_t = o_k | q_t = i)$$

Special initial probability vector  $\pi$

$$\rho_i = P(q_1 = i) \quad 1 \leq i \leq N$$

---

# Three Problems

---

Given this framework there are 3 problems that we can pose to an HMM

1. Given an observation sequence, what is the probability of that sequence given a model?
2. Given an observation sequence and a model, what is the most likely state sequence?
3. Given an observation sequence, find the best model parameters for a partially specified model

# Problem 1:

## Observation Likelihood

---

- The probability of a observation sequence given a model and state sequence
- Evaluation problem

# Problem 2:



- Most probable state sequence given a model and an observation sequence
- Decoding problem

# Problem 3:

---

- Infer the best model parameters, given a partial model and an observation sequence...
  - That is, fill in the A and B tables with the right numbers --
    - the numbers that make the observation sequence most likely
- This is to learn the probabilities!

# Solutions

---

Problem 1: Forward (learn observation sequence)

Problem 2: Viterbi (learn state sequence)

Problem 3: Forward-Backward (learn probabilities)

- An instance of EM (Expectation Maximization)

# Example :HMMs for Ice Cream

---

You are a climatologist in the year 2799 studying global warming

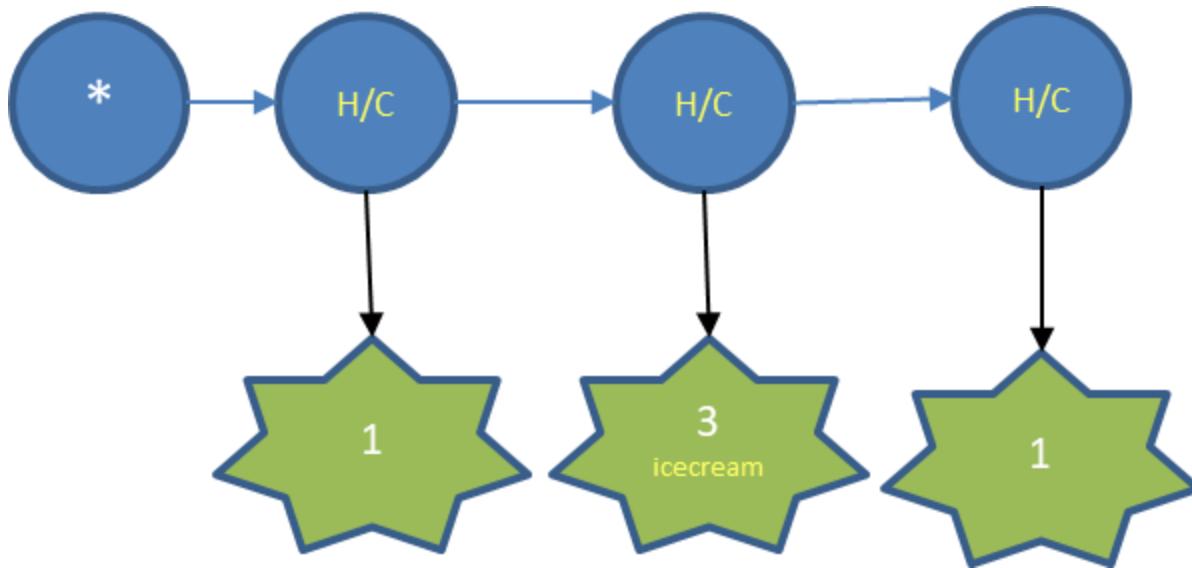
You can't find any records of the weather in Baltimore for summer of 2007

But you find Jason Eisner's diary which lists how many ice creams Jason ate every day that summer



Your job: figure out how hot it was each day

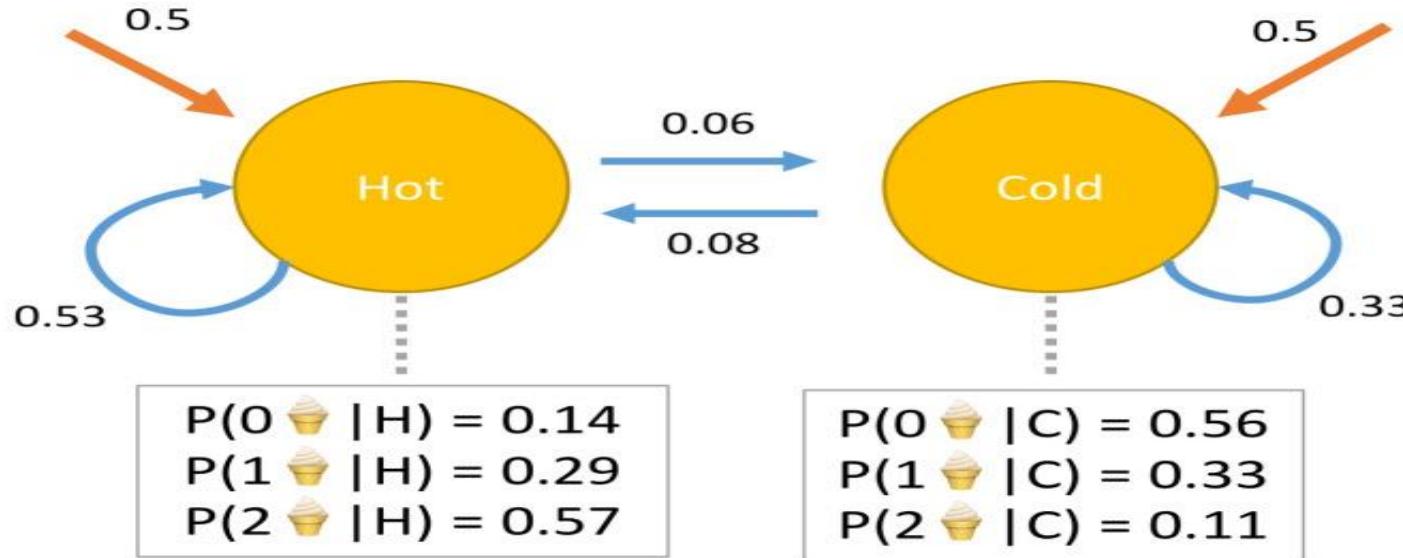
# Problem 1



- ❖ 1. Consider all possible 3-day weather sequences [H, H, H], [H, H, C], [H, C, H], .....
  2. For each 3-day weather sequence, consider the probability of the ice cream Consumption sequences [131,]
  3. Add all the probability
- ❖ Not efficient
- ❖ Forward algorithm

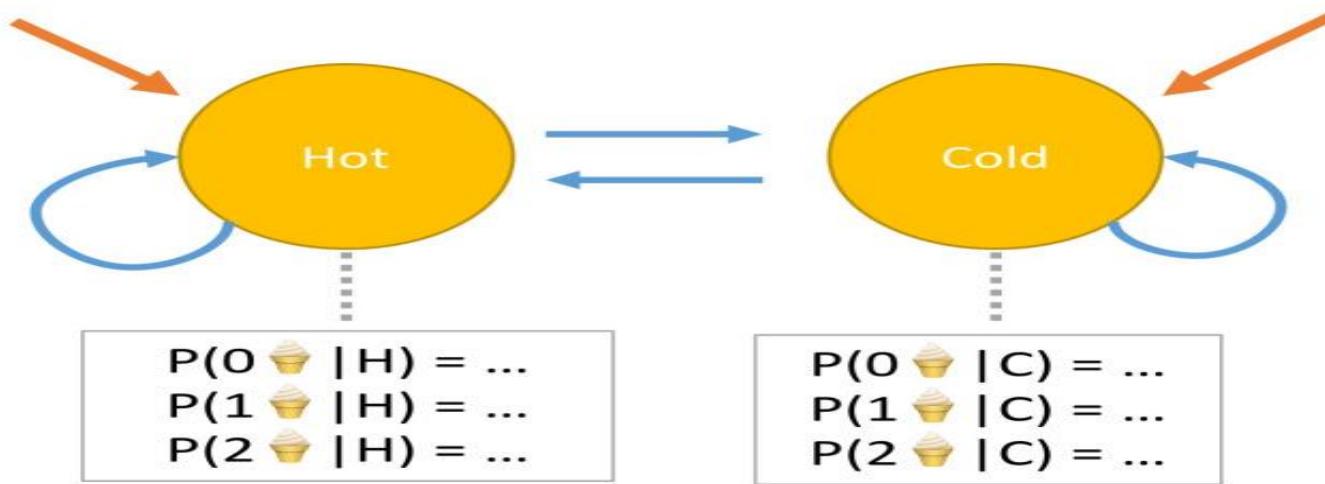
# Problem2

1      2      3



- 1. Consider all possible 3-day weather sequences [H, H, H], [H, H, C], [H, C, H],  
2. For each 3-day weather sequence, consider the probability of the ice cream Consumption sequences [0,1,2]  
3. Pick out the sequence that has the highest probability from step #2.
- Not efficient
- Viterbi algorithm

# Problem3



- **Find :**
  - The start probabilities
  - The transition probabilities
  - Emission probabilities
- Forward –backward algorithm

# Getting to HMMs

---

This equation should give us the best tag sequence

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

But how to make it operational? How to compute this value?

Intuition of Bayesian inference:

- Use Bayes rule to transform this equation into a set of probabilities that are easier to compute (and give the right answer)

# Using Bayes Rule

$$P(\underline{x}|y) = \frac{P(y|x)P(x)}{P(y)}$$

Know this.

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

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

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

# Likelihood and Prior

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} \overbrace{P(w_1^n | t_1^n)}^{\text{likelihood}} \overbrace{P(t_1^n)}^{\text{prior}}$$

$$P(w_1^n | t_1^n) \approx \prod_{i=1}^n P(w_i | t_i)$$

$$P(t_1^n) \approx \prod_{i=1}^n P(t_i | t_{i-1})$$

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

# HMM for Ice Cream

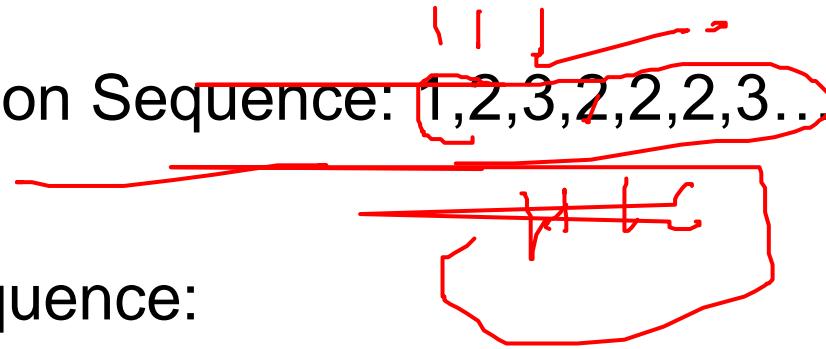
Given

- Ice Cream Observation Sequence:  $1, 2, 3, 2, 2, 2, 3, \dots$

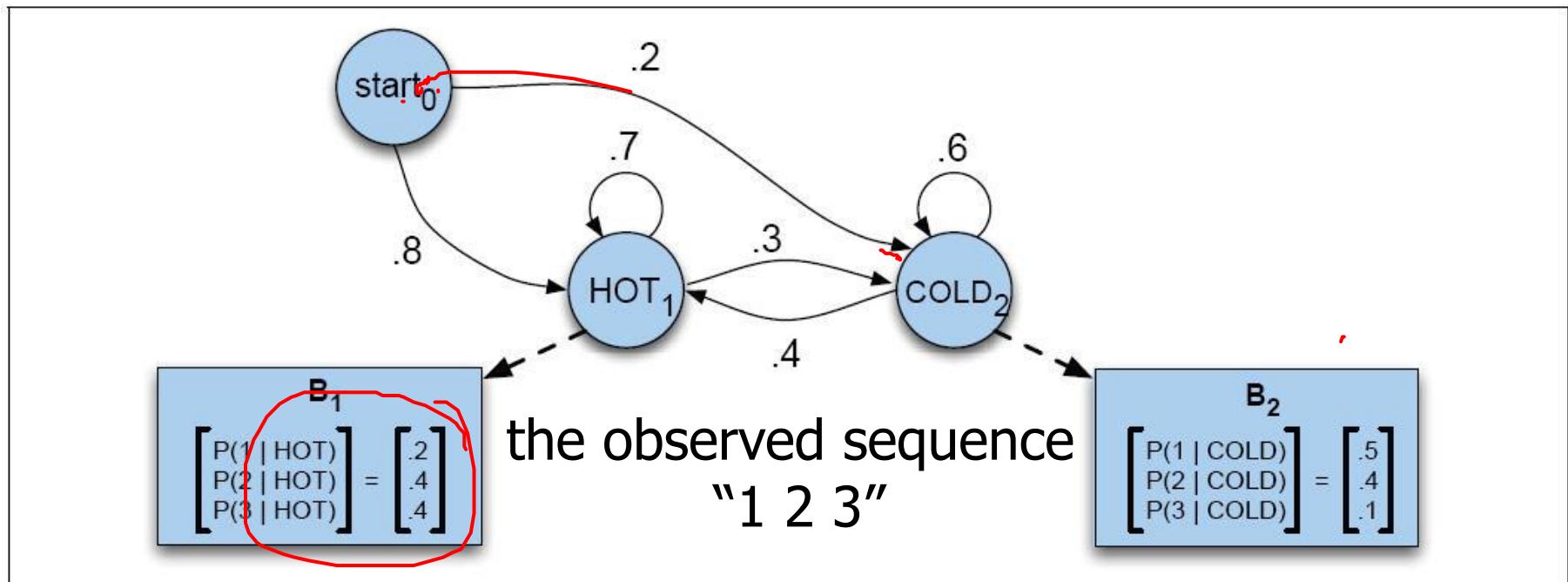
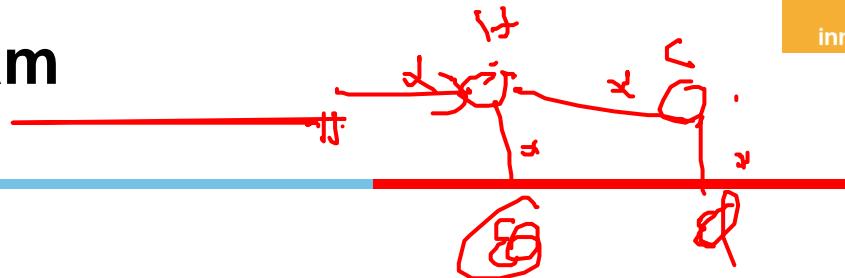
Produce:

- Hidden Weather Sequence:

$H, C, H, H, H, H, C, C, \dots$

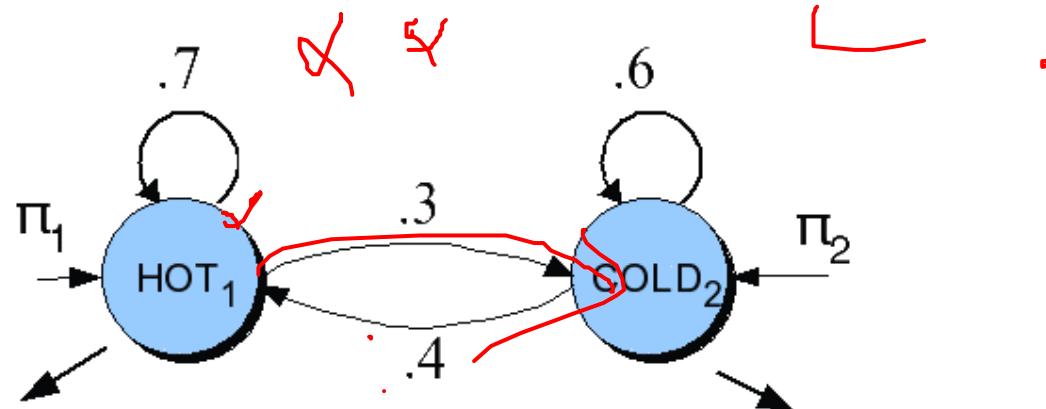


# HMM for Ice Cream



# HMM for Ice Cream

$$\pi = [.8, .2]$$



$$B_1 \left[ \begin{array}{c} P(1 \mid HOT) \\ P(2 \mid HOT) \\ P(3 \mid HOT) \end{array} \right] = \left[ \begin{array}{c} .2 \\ .4 \\ .4 \end{array} \right]$$

Transition probability

$$B_2 \left[ \begin{array}{c} P(1 \mid COLD) \\ P(2 \mid COLD) \\ P(3 \mid COLD) \end{array} \right] = \left[ \begin{array}{c} .5 \\ .4 \\ .1 \end{array} \right]$$

Emission probabilities

	H	C
H	0.7	0.3
C	0.4	0.6

	1	2	3
H	0.2	0.4	0.4
C	0.5	0.4	0.1

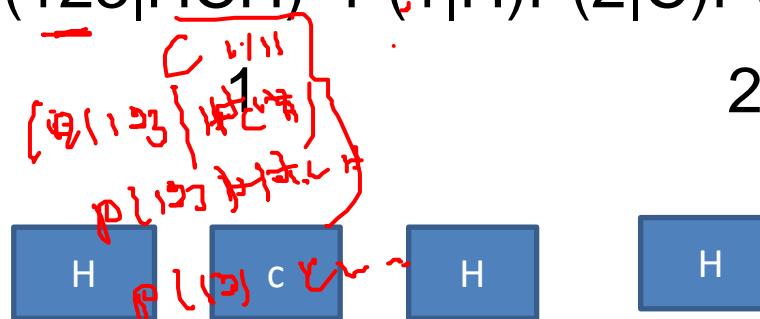
# Find 123 sequence

(1 2 3)

$$P(a,b) = P(a|b) P(b)$$

$$P(123|HCH) = P(1|H)P(2|C)P(3|H) P(S|H)P(C|H)P(H|C)$$

-----8(N^T=2^3)



2

-----8(N^T=2^3)



# Decoding

---

- Given an observation sequence
  - 123
- And an HMM
- The task of the decoder
  - To find the best hidden state sequence most likely to have produced the observed sequence
- Given the observation sequence  $O = (o_1 o_2 \dots o_T)$ , and an HMM model  $\Phi = (A, B)$ ,  
**how do we choose a corresponding state sequence  $Q = (q_1 q_2 \dots q_T)$  that is optimal in some sense (i.e., best explains the observations)**

# Contd..

---

- One possibility to find the best sequence is :
  - For each hidden state sequence Q
  - HHH, HHC, HCH,
  - Compute  $P(O|Q)$
  - Pick the highest one
- Why not?
  - $N^T$
- Instead:
  - The Viterbi algorithm
  - Is a dynamic programming algorithm

# Viterbi intuition

---

- We want to compute the joint probability of the observation sequence together with the best state sequence

$$v_t(j) = \max_{q_0, q_1, \dots, q_{t-1}} P(q_0, q_1 \dots q_{t-1}, o_1, o_2 \dots o_t, q_t = j | \lambda)$$

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t)$$

probabilities

	2	3
2	0.4	0.4
3	0.4	0.1

$$\begin{aligned} P(1,H) &= P(1/H) * P(H/H) \\ &= 0.2 * 0.7 \\ &= 0.14 \end{aligned}$$

$$\begin{aligned} P(1,H) &= P(1/H) * P(C/H) \\ &= 0.2 * 0.4 \\ &= 0.08 \end{aligned}$$

$$\begin{aligned} P(3,H) &= P(3/H) * P(H) \\ &= 0.4 * 0.8 \\ &= 0.32 \end{aligned}$$

$$\begin{aligned} P(3,H) &= P(3/H) * P(H/H) \\ &= 0.4 * 0.7 \\ &= 0.28 \end{aligned}$$

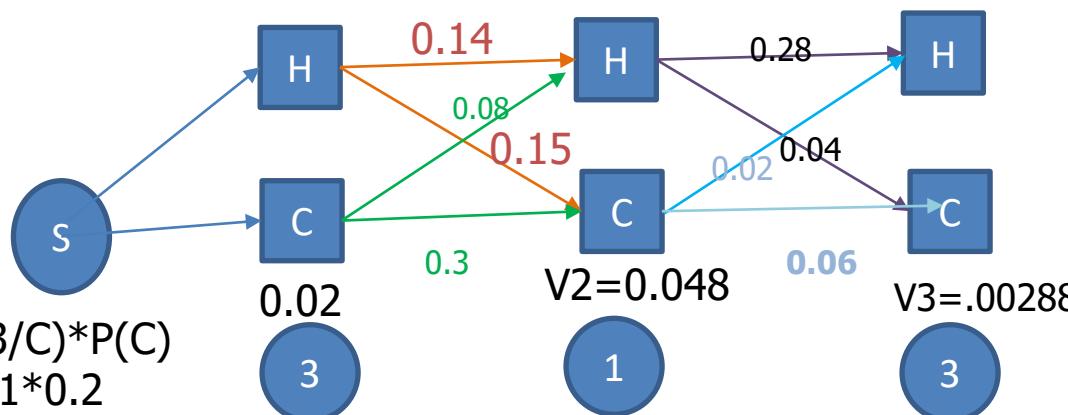
$$\begin{aligned} P(3,H) &= P(3/H) * P(H/C) \\ &= 0.1 * 0.2 \\ &= 0.02 \end{aligned}$$

$$V2 = \text{Max} [(0.14 * 0.32 = 0.0448), (0.08 * 0.02 = 0.0016)]$$

$$V3 = \text{Max} [(0.28 * 0.0448 = 0.013), (0.02 * 0.048 = 0.00096)]$$

$$V1 = 0.32 \quad V2 = 0.0448 \quad V3 = 0.013$$

Transition probability



$$\begin{aligned} P(3,C) &= P(3/C) * P(C) \\ &= 0.1 * 0.2 \\ &= 0.02 \end{aligned}$$

$$\begin{aligned} P(1,C) &= P(1/C) * P(C/H) \\ &= 0.5 * 0.3 \\ &= 0.15 \end{aligned}$$

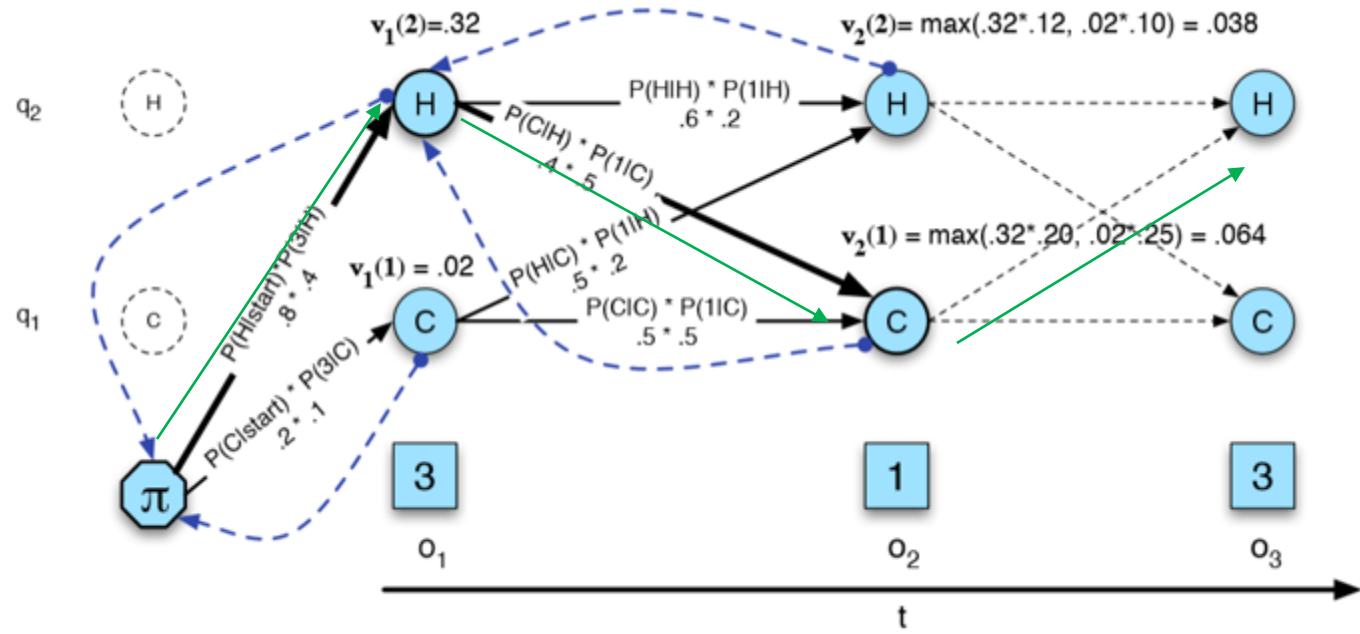
$$\begin{aligned} P(1,C) &= P(1/C) * P(C/C) \\ &= 0.5 * 0.6 \\ &= 0.30 \end{aligned}$$

$$V2 = \text{Max} [(0.15 * 0.32 = 0.048), (0.02 * 0.3 = 0.006)]$$

$$\begin{aligned} P(3,C) &= P(3/C) * P(C/C) \\ &= 0.1 * 0.6 \\ &= 0.06 \end{aligned}$$

$$\begin{aligned} P(3,C) &= P(3/C) * P(C/H) \\ &= 0.1 * 0.4 \\ &= 0.04 \end{aligned}$$

	H	C
H	0.7	0.3
C	0.4	0.6



# Example: Rainy and Sunny Days

Your colleague in another city either walks to work or drives every day and his decision is usually based on the weather

Given daily emails that include whether he has walked or driven to work, you want to guess the most likely sequence of whether the days were rainy or sunny

Two hidden states: rainy and sunny

Two observables: walking and driving

Assume equal likelihood of the first day being rainy or sunny

Transitional probabilities

rainy given yesterday was (rainy = .7, sunny = .3)

sunny given yesterday was (rainy = .4, sunny = .6)

Output (emission) probabilities

sunny given walking = .1, driving = .9

rainy given walking = .8, driving = .2

Given that your colleague walked, drove, walked, what is the most likely sequence of days?

## State1:Walk

$$\begin{aligned} P(\text{rainy, walk}) &= P(\text{walk} | \text{rainy}) * P(\text{rainy}) \\ &= 0.8 * 0.5 = 0.40 \end{aligned} \quad V1 = 0.40$$

$$\begin{aligned} P(\text{sunny, walk}) &= P(\text{walk} | \text{sunny}) * P(\text{sunny}) \\ &= 0.1 * 0.5 = 0.05 \end{aligned} \quad V1 = 0.05$$

## State2: Drive

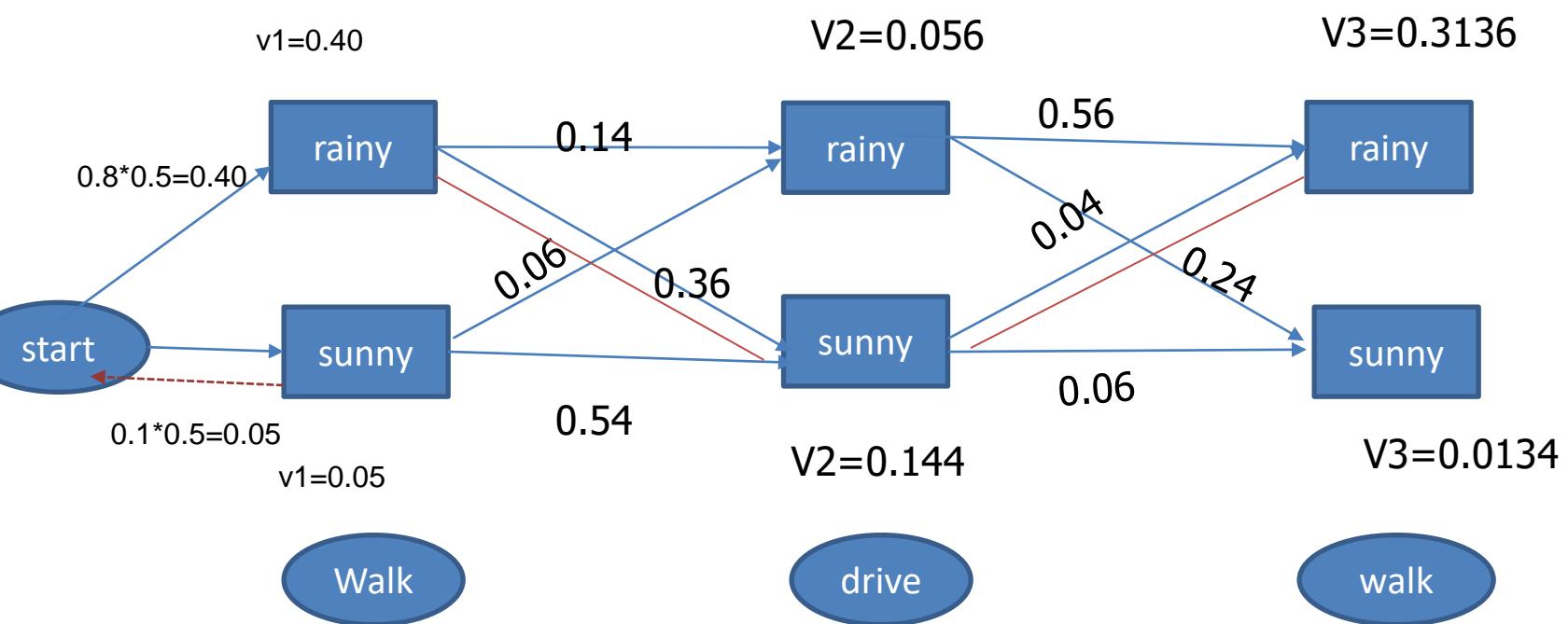
$$\begin{aligned} P(\text{rainy, drive}) &= P(\text{drive} | \text{rainy}) * P(\text{rainy} | \text{rainy}) \\ &= 0.2 * 0.7 = 0.14 \\ &= P(\text{drive} | \text{rainy}) * P(\text{rainy} | \text{sunny}) \\ &= 0.2 * 0.3 = 0.06 \end{aligned} \quad \begin{aligned} V2 &= \max(0.40 * 0.14 = 0.056, \\ &\quad 0.05 * 0.06 = 0.003) \\ &= 0.56 \end{aligned}$$

$$\begin{aligned} P(\text{sunny, drive}) &= P(\text{drive} | \text{sunny}) * P(\text{sunny} | \text{sunny}) \\ &= 0.9 * 0.6 = 0.54 \\ &= P(\text{drive} | \text{sunny}) * P(\text{sunny} | \text{rainy}) \\ &= 0.9 * 0.4 = 0.36 \end{aligned} \quad \begin{aligned} V2 &= \max(0.36 * 0.40 = 0.144, \\ &\quad 0.05 * 0.5 = 0.0027) \\ &= 0.144 \end{aligned}$$

### State 3:Walk

$$\begin{aligned} P(\text{rainy walk}) &= P(\text{walk|rainy}) * P(\text{rainy|rainy}) \\ &= 0.8 * 0.7 = 0.56 \quad (0.0144 * 0.04 = 0.00051) \\ &= P(\text{rainy|walk}) * P(\text{rainy|sunny}) = 0.03136 \\ &= 0.8 * 0.3 = 0.24 \end{aligned}$$

$$\begin{aligned} P(\text{sunny,walk}) &= P(\text{walk|sunny}) * P(\text{sunny|sunny}) \\ &= 0.1 * 0.6 = 0.06 \quad 0.144 * 0.06 = 0.008 \\ &= P(\text{sunny|walk}) * P(\text{sunny|rainy}) = 0.0134 \\ &= 0.1 * 0.4 = 0.04 \end{aligned}$$



The best sequence =rainy sunny rainy

# The Viterbi Algorithm

**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**

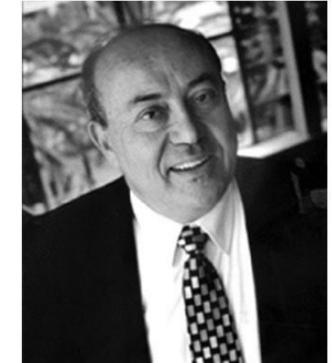
$viterbi[s,t] \leftarrow \max_{s'=1}^N viterbi[s',t-1] * a_{s',s} * b_s(o_t)$

$backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^N viterbi[s',t-1] * a_{s',s}$

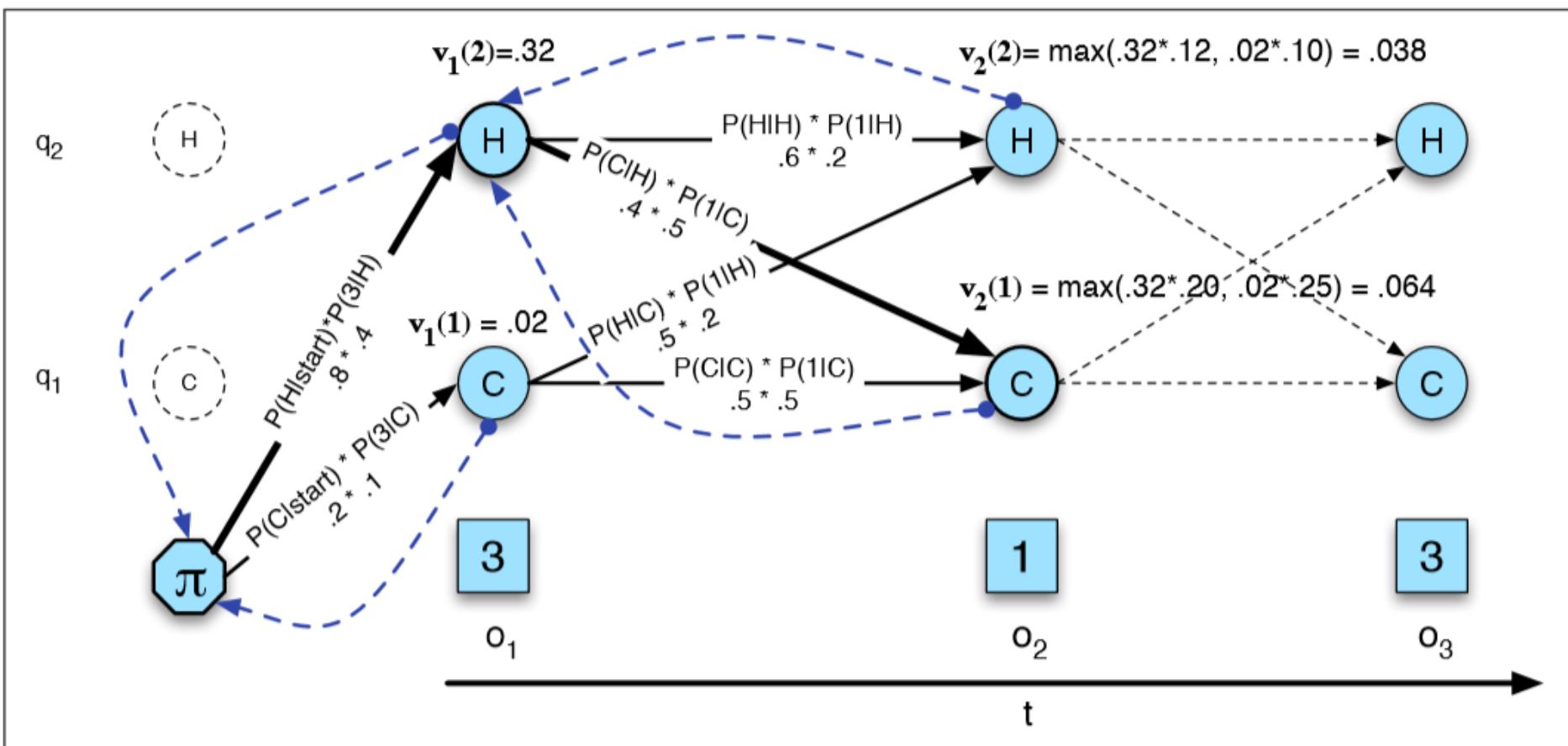
$viterbi[q_F,T] \leftarrow \max_{s=1}^N viterbi[s,T] * a_{s,q_F}$  ; termination step

$backpointer[q_F,T] \leftarrow \operatorname{argmax}_{s=1}^N viterbi[s,T] * a_{s,q_F}$  ; termination step

**return** the backtrace path by following backpointers to states back in time from  $backpointer[q_F,T]$



# Viterbi Example: Ice Cream



**Figure A.10** The Viterbi backtrace. As we extend each path to a new state account for the next observation, we keep a backpointer (shown with broken lines) to the best path that led us to this state.

# Example3

---

Janet will back the bill

The correct series of tags is:

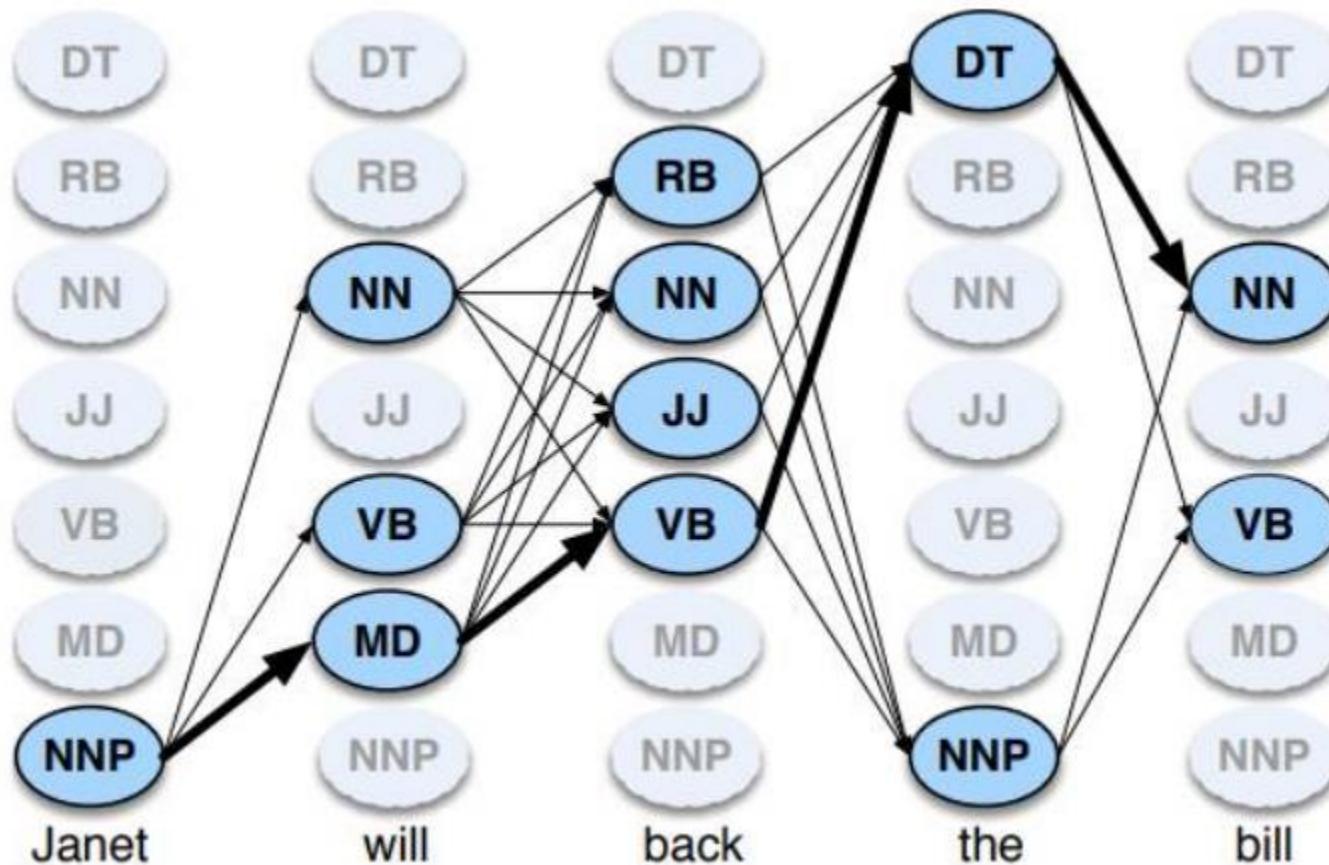
Janet/NNP will/MD back/VB the/DT bill/NN

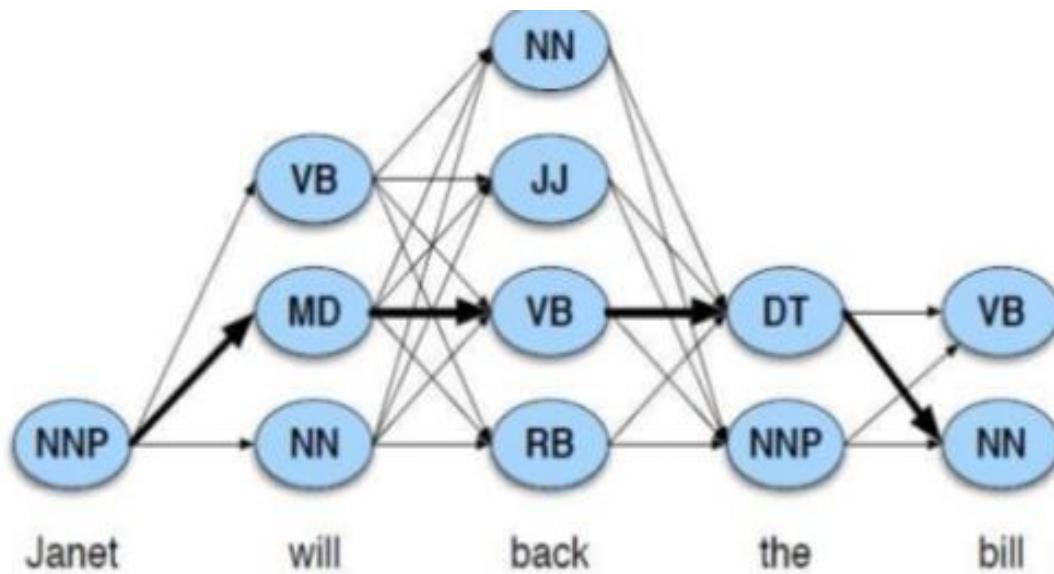
	<b>NNP</b>	<b>MD</b>	<b>VB</b>	<b>JJ</b>	<b>NN</b>	<b>RB</b>	<b>DT</b>
<i>&lt; s &gt;</i>	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
<b>NNP</b>	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
<b>MD</b>	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
<b>VB</b>	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
<b>JJ</b>	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
<b>NN</b>	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
<b>RB</b>	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
<b>DT</b>	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

	<b>Janet</b>	<b>will</b>	<b>back</b>	<b>the</b>	<b>bill</b>
<b>NNP</b>	0.000032	0	0	0.000048	0
<b>MD</b>	0	0.308431	0	0	0
<b>VB</b>	0	0.000028	0.000672	0	0.000028
<b>JJ</b>	0	0	0.000340	0.000097	0
<b>NN</b>	0	0.000200	0.000223	0.000006	0.002337
<b>RB</b>	0	0	0.010446	0	0
<b>DT</b>	0	0	0	0.506099	0

- $v_1(NNP) = 0.2767 \times 0.000032$  INITIALIZATION STEP
- For  $t=2$ : we compute  $v_2(MD)$ ,  $v_2(VB)$ , and  $v_2(NN)$  and chose the maximum

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t)$$





- Janet/NNP will/MD back/VB the/DT bill/NN

# Practice question

---

Suppose you want to use a HMM tagger to tag the phrase , "the light book ", Where we have the following probabilities

$P(\text{the}|\text{Det})=0.3, P(\text{the}|\text{Noun})=0.1, P(\text{light}|\text{noun})=0.003, P(\text{light}|\text{Adj})=0.002, P(\text{light}|\text{verb})=0.06, P(\text{book}|\text{noun})=0.003, P(\text{book}|\text{verb})=0.01$

$P(\text{Verb}|\text{Det})=0.00001, P(\text{Noun}|\text{Det})=0.5, P(\text{Adj}|\text{Det})=0.3, P(\text{Noun}|\text{Noun})=0.2, P(\text{Adj}|\text{Noun})=0.002, P(\text{Noun}|\text{Adj})=0.2, P(\text{Noun}|\text{Verb})=0.3, P(\text{Verb}|\text{Noun})=0.3, P(\text{Verb}|\text{Adj})=0.001, P(\text{verb}|\text{verb})=0.1$

Assume that all the tags have the same probabilities at the beginning of the sentence . Using Viterbi algorithm find out the best tag sequence.

---

# Forward

---

Efficiently computes the probability of an observed sequence given a model

- $P(\text{sequence}|\text{model})$

Nearly identical to Viterbi; **replace the MAX with a SUM**

For our particular case, we would sum over the eight 3-event sequences *cold cold cold, cold cold hot*, that is,

$$P(3 \ 1 \ 3) = P(3 \ 1 \ 3, \text{cold cold cold}) + P(3 \ 1 \ 3, \text{cold cold hot}) + P(3 \ 1 \ 3, \text{hot hot cold}) + \dots$$

# The Forward Algorithm

**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

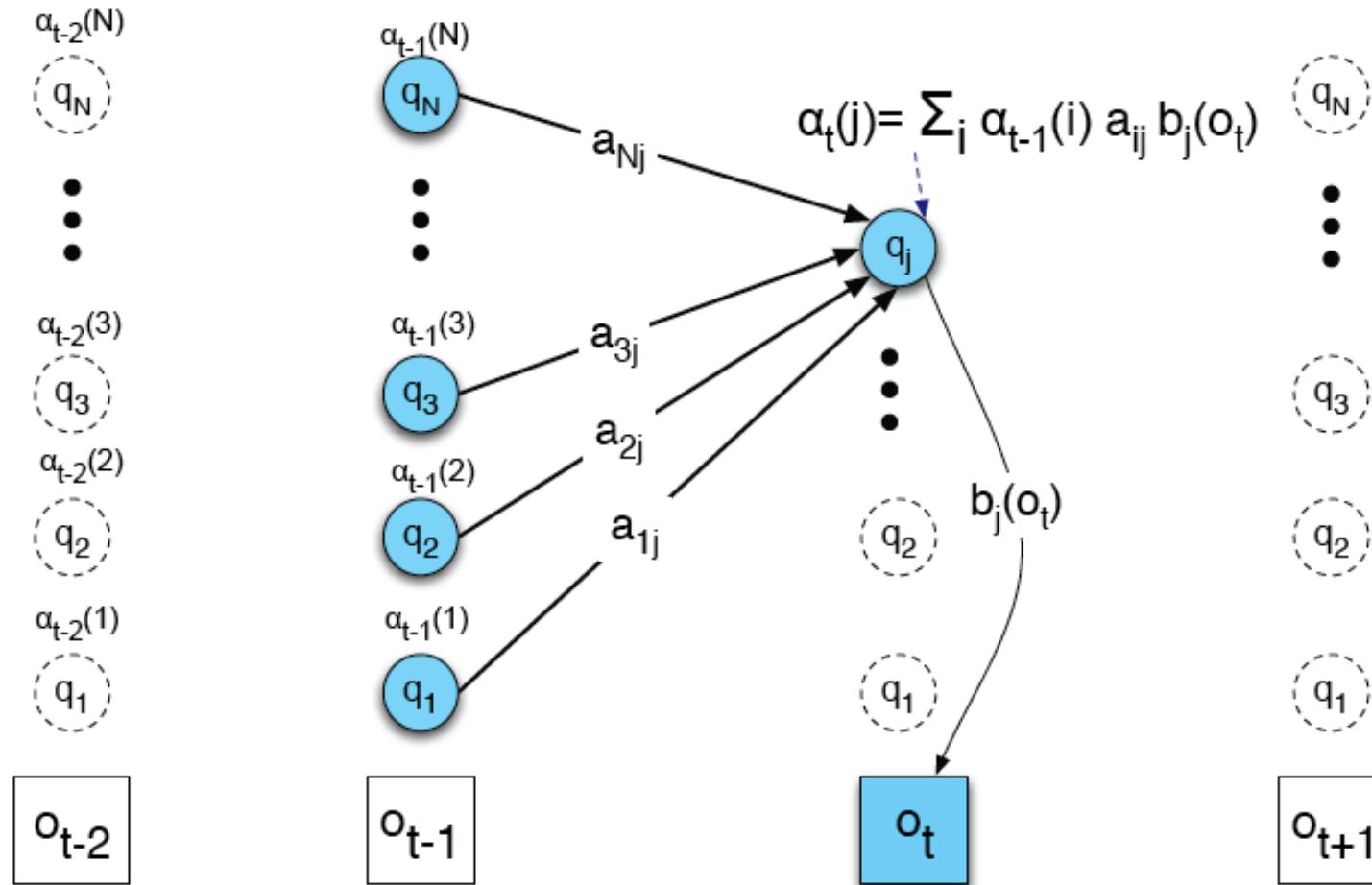
**for** each state  $s$  **from** 1 **to**  $N$  **do**

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

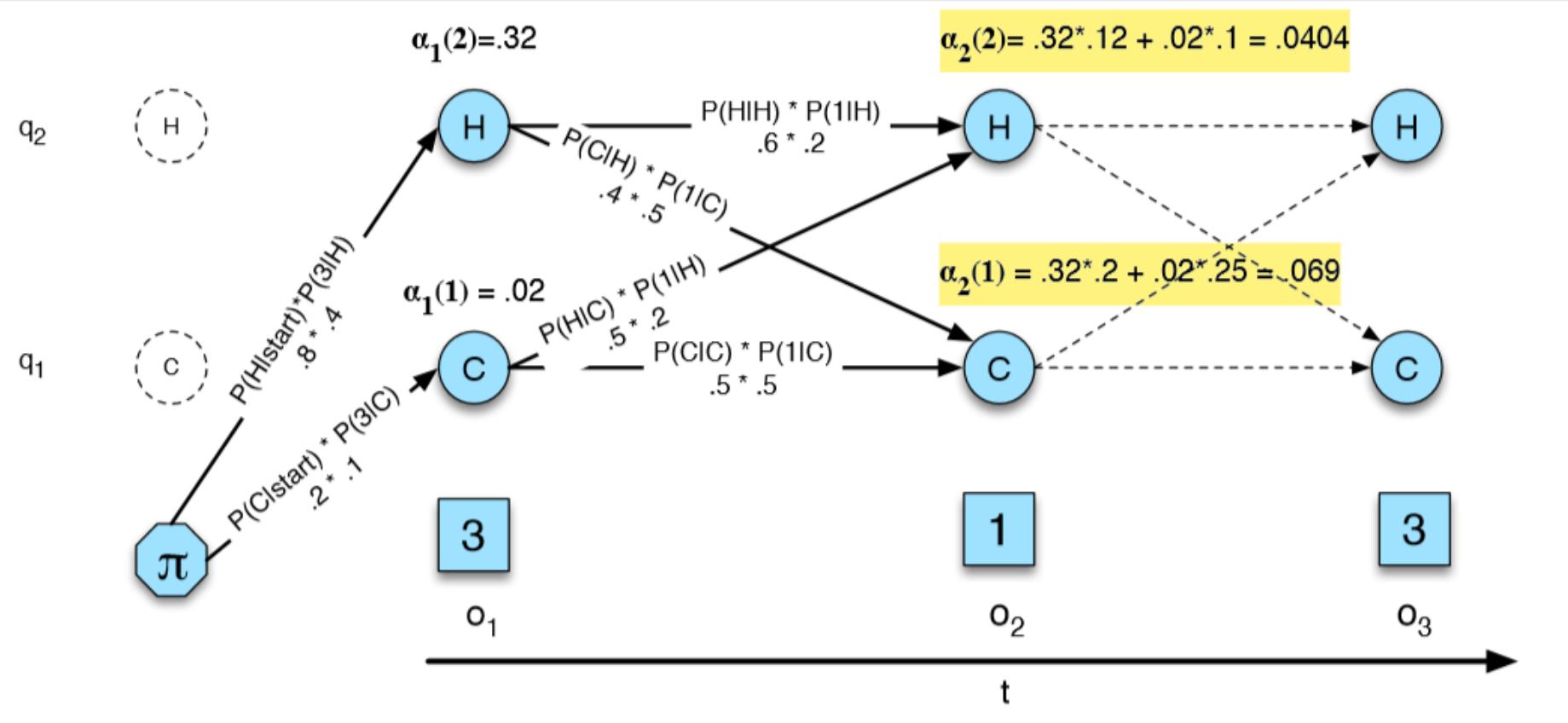
$forward[q_F, T] \leftarrow \sum_{s=1}^N forward[s, T] * a_{s,q_F}$  ; termination step

**return**  $forward[q_F, T]$

# Visualizing Forward



# Forward Algorithm: Ice Cream



**Figure A.5** The forward trellis for computing the total observation likelihood for the ice-cream events 3 1 3. Hidden states are in circles, observations in squares. The figure shows the computation of  $\alpha_t(j)$  for two states at two time steps. The computation in each cell follows Eq. A.12:  $\alpha_t(j) = \sum_{i=1}^N \alpha_{t-1}(i) a_{ij} b_j(o_t)$ . The resulting probability expressed in each cell is Eq. A.11:  $\alpha_t(j) = P(o_1, o_2 \dots o_t, q_t = j | \lambda)$ .

# Problem 3

---

Infer the best model parameters, given a skeletal model and an observation sequence...

- That is, fill in the A and B tables with the right numbers...
  - The numbers that make the observation sequence most likely
- Useful for getting an HMM without having to hire annotators...

# Forward-Backward

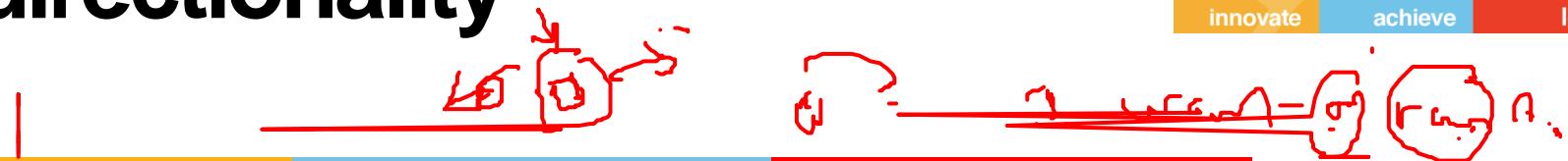
---

**Learning:** Given an observation sequence  $O$  and the set of possible states in the HMM, learn the HMM parameters  $A$  and  $B$ .

Is a special case of the EM or Expectation-Maximization algorithm

The algorithm will let us learn the transition probabilities  $A = \{a_{ij}\}$  and the emission probabilities  $B = \{b_i(o_t)\}$  of the HMM

# Bidirectionality



- One problem with the HMM models as presented is that they are exclusively run left-to-right.  $b_{1-1}$   $b_{2-2}$
- Viterbi algorithm still allows present decisions to be influenced indirectly by future decisions,
- It would help even more if a decision about word  $w_i$  could directly use information about future tags  $t_{i+1}$  and  $t_{i+2}$ .

# Bidirectionality



- Any sequence model can be turned into a bidirectional model by using multiple passes.
- For example, the first pass would use only part-of-speech features from already-disambiguated words on the left. In the second pass, tags for all words, including those on the right, can be used.
- Alternately, the tagger can be run twice, once left-to-right and once right-to-left.
- In Viterbi decoding, the classifier chooses the higher scoring of the two sequences (left-to-right or right-to-left).
- Modern taggers are generally run bidirectionally.

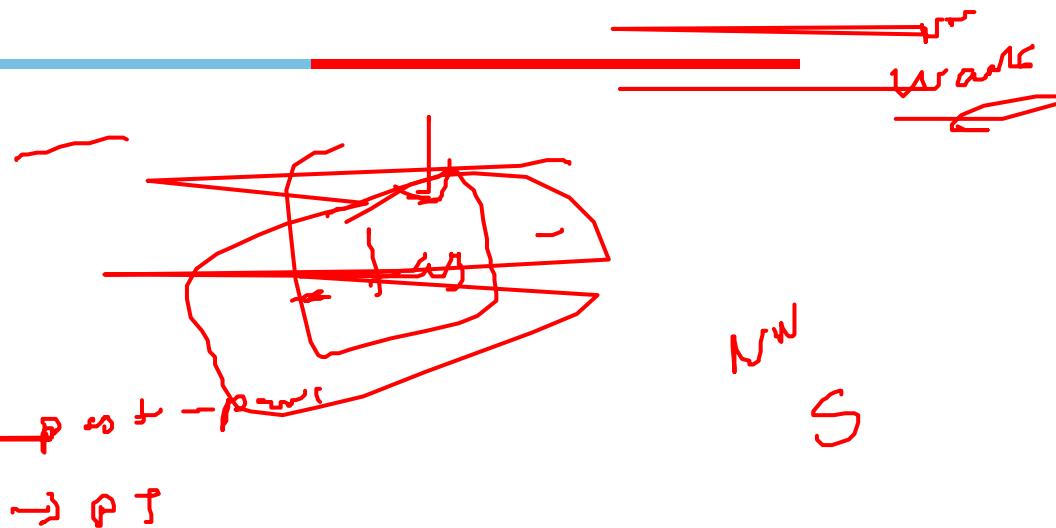
# Some limitation of HMMS

- Unknown words
- First order HMM

Eg:

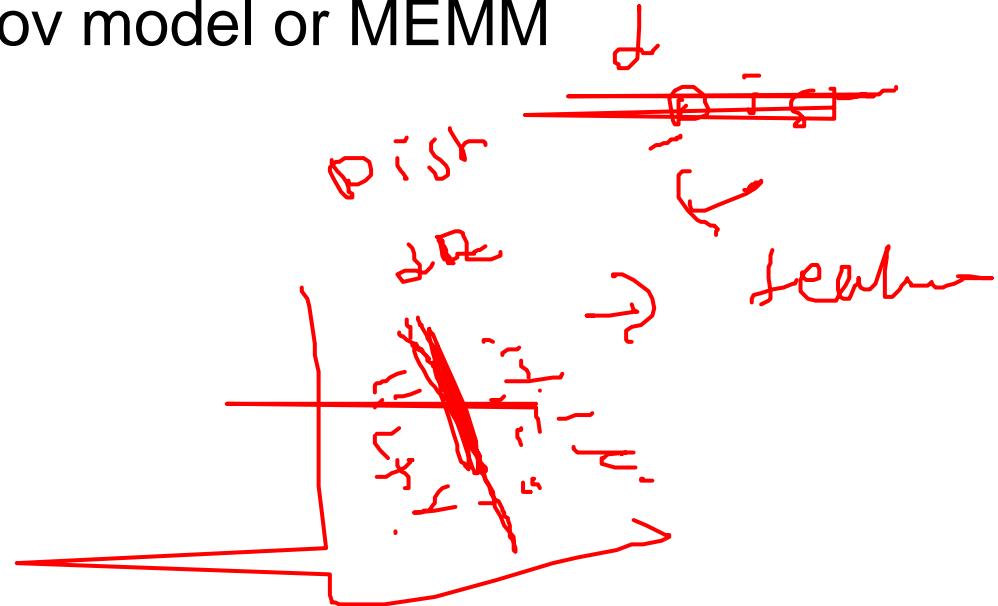
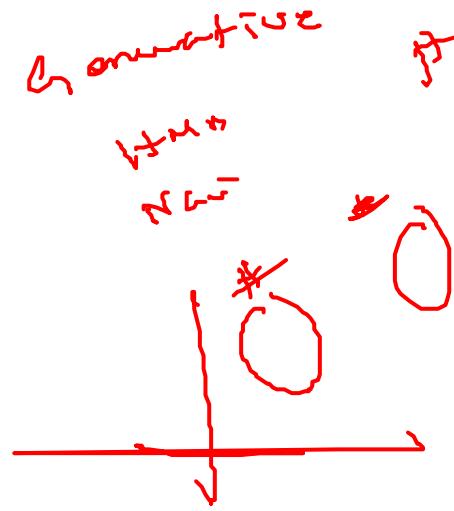
Is clearly marked

He clearly marked



# Maximum Entropy Markov Model

- Turn logistic regression into a discriminative sequence model simply by running it on successive words, using the class assigned to the prior word as a feature in the classification of the next word.
- When we apply logistic regression in this way, it's called maximum entropy Markov model or MEMM



# Maximum Entropy Markov Model



- Let the sequence of words be  $W = w^n_1$  and the sequence of tags  $T = t^n_1$ .
- In an HMM to compute the best tag sequence that maximizes  $P(T|W)$  we rely on Bayes' rule and the likelihood  $P(W|T)$ :

$$\hat{T} = \operatorname{argmax}_T P(T|W)$$

$$= \operatorname{argmax}_T P(W|T)P(T)$$

$$= \operatorname{argmax}_T \prod_i P(\text{word}_i | \underline{\text{tag}}_i) \prod_i P(\underline{\text{tag}}_i | \underline{\text{tag}}_{i-1})$$

$P((\underline{B})) =$   
 ~~$P(B_1, B_2, B_3, B_4)$~~

$\rightarrow P(B_1)$

# Maximum Entropy Markov Model

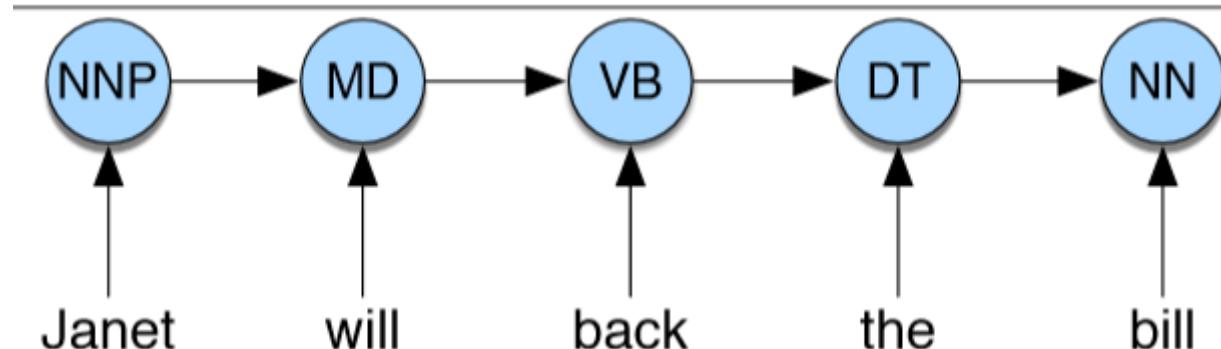
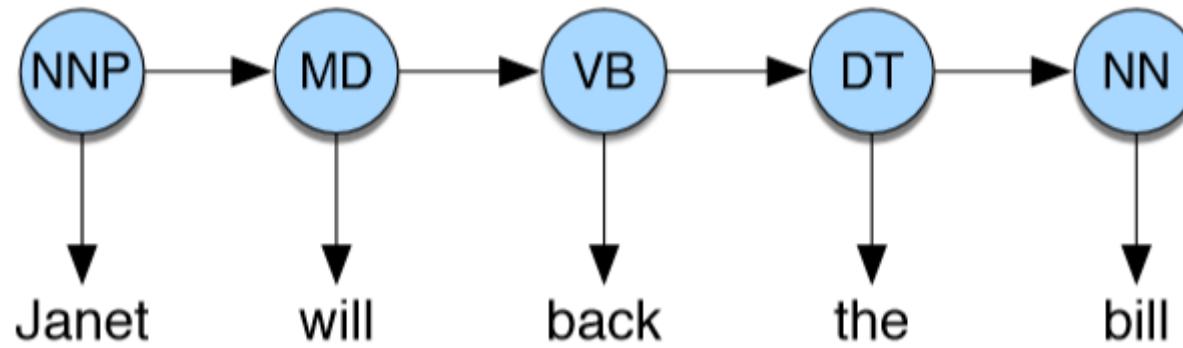


- In an MEMM, by contrast, we compute the posterior  $P(T|W)$  directly, training it to discriminate among the possible tag sequences:

$$\begin{aligned}\hat{T} &= \operatorname{argmax}_T P(T|W) \\ &= \operatorname{argmax}_T \prod_i P(t_i|w_i, t_{i-1})\end{aligned}$$
A red circle highlights the term  $P(t_i|w_i, t_{i-1})$ .

# Maximum Entropy Markov Model

- Consider tagging just one word. A multinomial logistic regression classifier could compute the single probability  $P(t_i|w_i, t_{i-1})$  in a different way than an HMM
- HMMs compute likelihood (observation word conditioned on tags) but MEMMs compute posterior (tags conditioned on observation words).



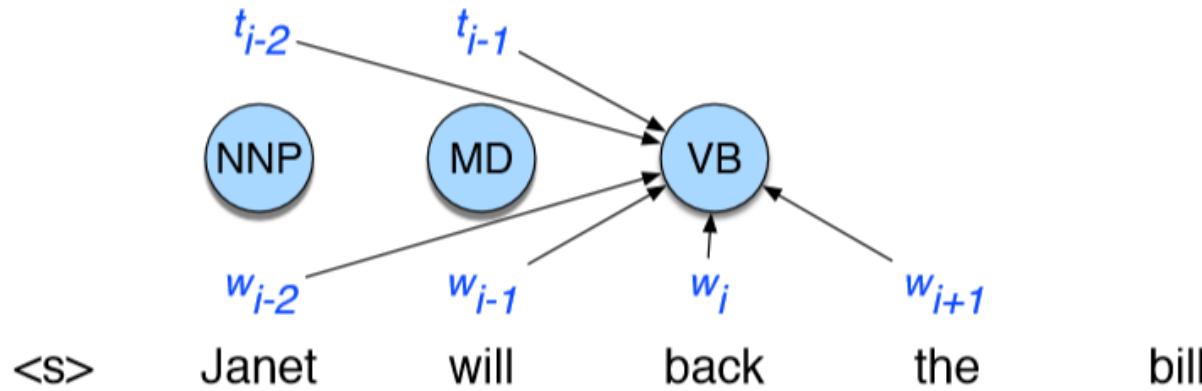
# Learning MEMM

---

- Learning in MEMMs relies on the same supervised learning algorithms we presented for logistic regression.
- Given a sequence of observations, feature functions, and corresponding hidden states, we use gradient descent to train the weights to maximize the log-likelihood of the training corpus.

# Maximum Entropy Markov Model

- Reason to use a discriminative sequence model is that it's easier to incorporate a lot of features



**Figure 8.13** An MEMM for part-of-speech tagging showing the ability to condition on more features.

# MEMM

---

- Janet/NNP will/MD back/VB the/DT bill/NN, when  $w_i$  is the word back, would generate the following features

$t_i = \text{VB}$  and  $w_{i-2} = \text{Janet}$

$t_i = \text{VB}$  and  $w_{i-1} = \text{will}$

$t_i = \text{VB}$  and  $w_i = \text{back}$

$t_i = \text{VB}$  and  $w_{i+1} = \text{the}$

$t_i = \text{VB}$  and  $w_{i+2} = \text{bill}$

$t_i = \text{VB}$  and  $t_{i-1} = \text{MD}$

$t_i = \text{VB}$  and  $t_{i-1} = \text{MD}$  and  $t_{i-2} = \text{NNP}$

$t_i = \text{VB}$  and  $w_i = \text{back}$  and  $w_{i+1} = \text{the}$

# Training MEMMs

- The most likely sequence of tags is then computed by combining these features of the input word  $w_i$ , its neighbors within  $l$  words  $w_{i-l}^{i+l}$ , and the previous  $k$  tags  $t_{i-k}^{i-1}$  as follows (using  $\theta$  to refer to feature weights instead of  $w$  to avoid the confusion with  $w$  meaning words):

$$\begin{aligned}
 \hat{T} &= \operatorname{argmax}_T P(T|W) \\
 &= \operatorname{argmax}_T \prod_i P(t_i | w_{i-l}^{i+l}, t_{i-k}^{i-1}) \\
 &= \operatorname{argmax}_T \prod_i \frac{\exp \left( \sum_j \theta_j f_j(t_i, w_{i-l}^{i+l}, t_{i-k}^{i-1}) \right)}{\sum_{t' \in \text{tagset}} \exp \left( \sum_j \theta_j f_j(t', w_{i-l}^{i+l}, t_{i-k}^{i-1}) \right)}
 \end{aligned}$$

# How to decode to find this optimal tag sequence $\hat{T}$ ?

- Simplest way to turn logistic regression into a sequence model is to build a local classifier that classifies each word left to right, making a hard classification on the first word in the sentence, then a hard decision on the second word, and so on.
- This is called a greedy decoding algorithm

```

function GREEDY SEQUENCE DECODING(words W, model P) returns tag sequence T
    for  $i = 1$  to  $\text{length}(W)$ 
         $\hat{t}_i = \underset{t' \in T}{\text{argmax}} P(t' | w_{i-l}^{i+l}, t_{i-k}^{i-1})$ 

```

**Figure 8.14** In greedy decoding we simply run the classifier on each token, left to right, each time making a hard decision about which is the best tag.

# Issue with greedy algorithm

---

- The problem with the greedy algorithm is that by making a hard decision on each word before moving on to the next word, the classifier can't use evidence from future decisions.
- Although the greedy algorithm is very fast, and occasionally has sufficient accuracy to be useful, in general the hard decision causes too great a drop in performance, and we don't use it.
- MEMM with the Viterbi algorithm just as with the HMM, Viterbi finding the sequence of part-of-speech tags that is optimal for the whole sentence

# MEMM with Viterbi algorithm

---

- Finding the sequence of part-of-speech tags that is optimal for the whole sentence. Viterbi value of time t for

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \leq j \leq N, 1 < t \leq T$$

- $v_t(j) = \max_{i=1}^N v_{t-1}(i) P(s_j|s_i) P(o_t|s_j) \quad 1 \leq j \leq N, 1 < t \leq T$
  - $v_t(j) = \max_{i=1}^N v_{t-1}(i) P(s_j|s_i, o_t) \quad 1 \leq j \leq N, 1 < t \leq T$
-

# Open Source POS Tagger

---

- Stanford PoS Tagger python bind- Java based, but can be used in python but difficult to install
  - Flair - POS tagger available for python.
  - NLTK implementation to be very precise, around 97% and its quite fast.
  - SPACY
-

Q1.

Introduction- 3 marks

N-gram LM- 4 marks

Q2 Neural LM- 3 marks

Word embeddings- - 4/5 marks

Q3 Vector semantics- 6 marks

Q4. POS tagging- 5 marks

Q5 HMM/Viterbi - 4/5 marks

---

## References

- <https://www.nltk.org/>
- <https://likegeeks.com/nlp-tutorial-using-python-nltk/>
- <https://www.guru99.com/pos-tagging-chunking-nltk.html>
- <https://medium.com/greyatom/learning-pos-tagging-chunking-in-nlp-85f7f811a8cb>
- <https://nlp.stanford.edu/software/tagger.shtml>
- <https://www.forbes.com/sites/mariyayao/2020/01/22/what-are--important-ai--machine-learning-trends-for-2020/#601ce9623239>
- <https://medium.com/fintechexplained/nlp-text-processing-in-data-science-projects-f083009d78fc>



# Natural Language Processing

DSECLZG530  
**Hidden Markov Models(MEMM)**

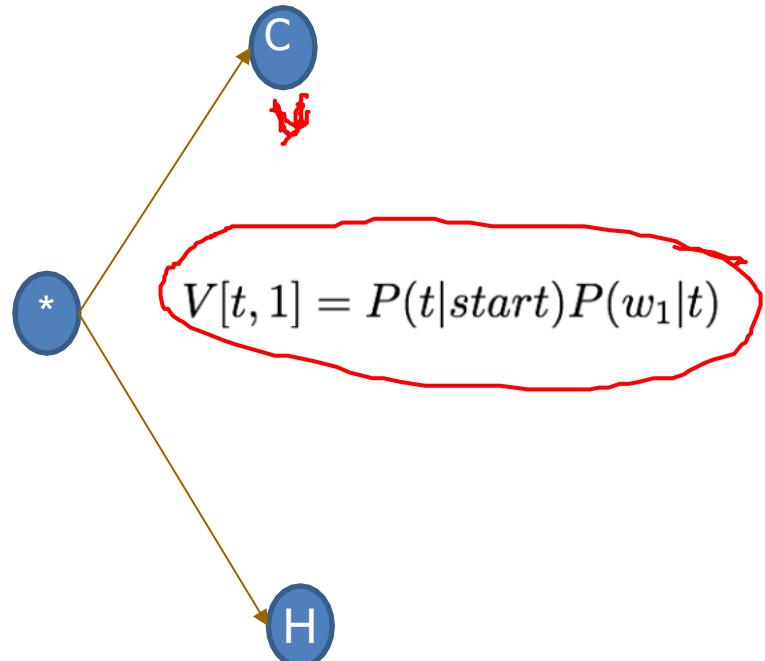


**BITS** Pilani  
Pilani Campus

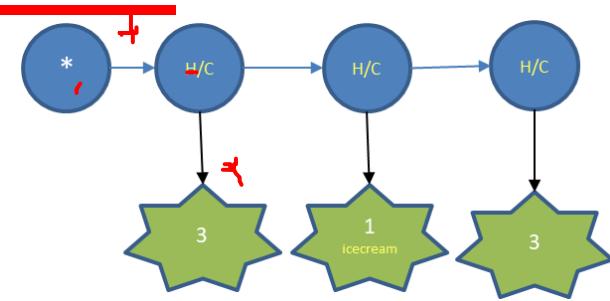
# Hidden Markov Model

## Example -1 – Viterbi Algorithm - Initialization

$$P(C) * P(3|C) = 0.2 * 0.1 = 0.02$$



$$P(H) * P(3|H) = 0.8 * 0.4 = 0.32$$



	Hot	Cold
<S>	0.8	0.2
A		
Hot	0.6	0.4
Cold	0.5	0.5

A	Hot	Cold
Hot	0.6	0.4
Cold	0.5	0.5

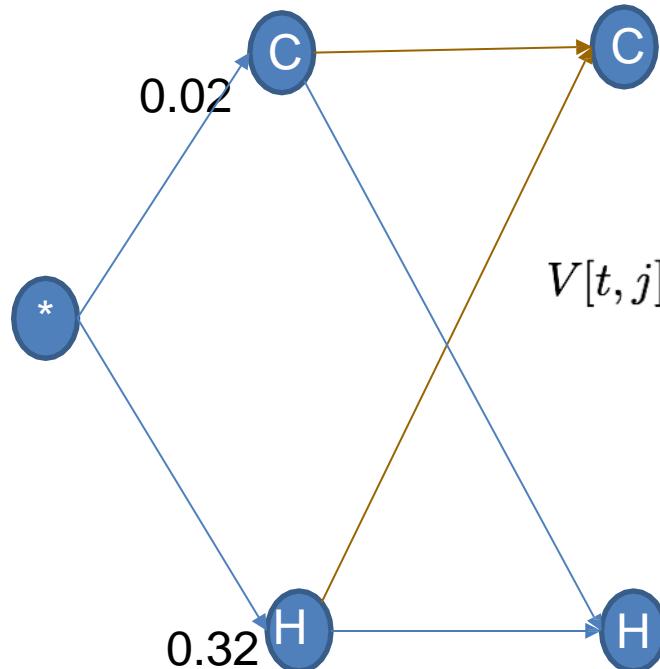
B	1	2	3
Hot	.2	.4	.4
Cold	.5	.4	.1

# Hidden Markov Model

## Example -1 – Viterbi Algorithm - Recursion

$$P(C)*P(C|C)*P(1|C) = 0.02*0.5*0.5 = 0.005$$

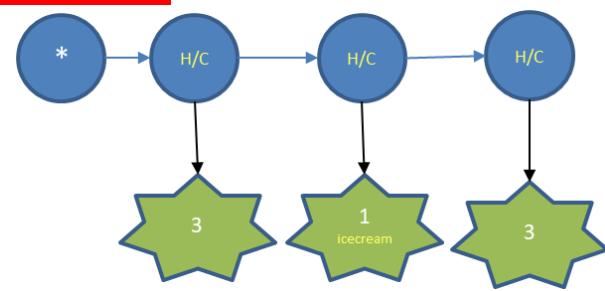
$$P(H)*P(C|H)*P(1|C) = 0.32*0.4*0.5 = 0.064$$



$$V[t, j] = \max_{t'} V[t', j - 1] P(t|t') P(w_j|t)$$

$$P(C)*P(H|C)*P(1|H) = 0.02*0.5*0.2 = 0.002$$

$$P(H)*P(H|H)*P(1|H) = 0.32*0.6*0.2 = 0.0384$$



	Hot	Cold
<S>	0.8	0.2
A		

A	Hot	Cold
Hot	0.6	0.4
Cold	0.5	0.5

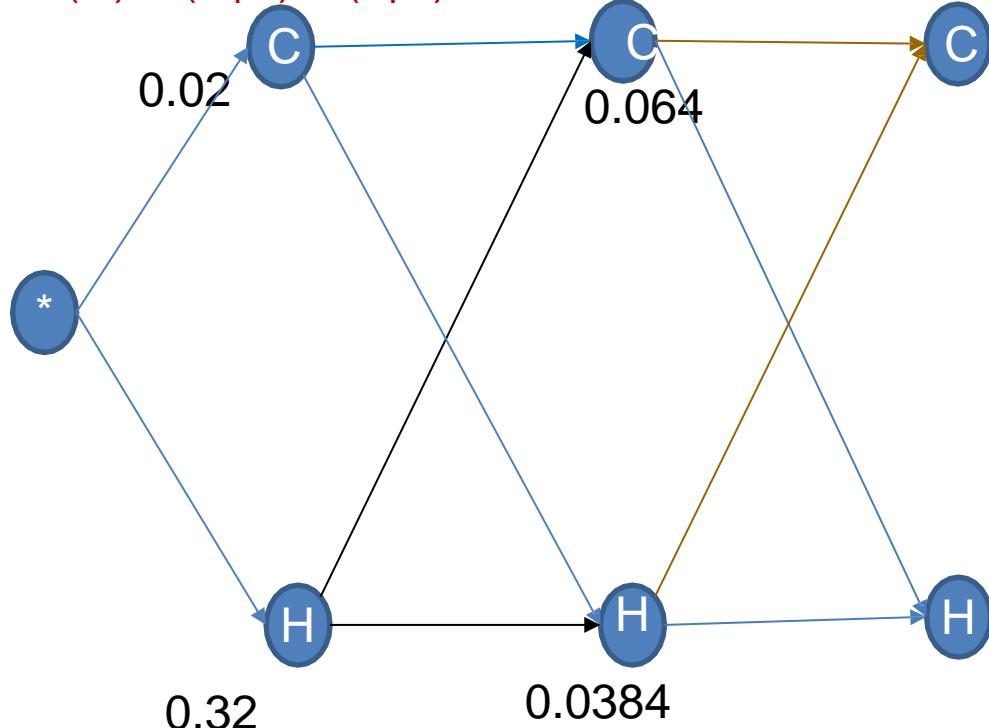
B	1	2	3
Hot	.2	.4	.4
Cold	.5	.4	.1

# Hidden Markov Model

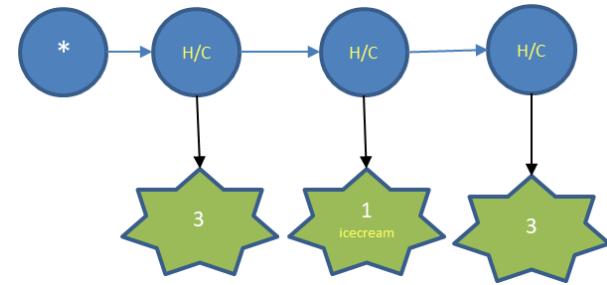
## Example -1 – Viterbi Algorithm – Termination through Back Trace

$$P(C)*P(C|C)*P(3|C) = 0.064 * 0.5 * 0.1 = \textcolor{red}{0.032}$$

$$P(H)*P(C|H)*P(3|C) = 0.0384 * 0.4 * 0.1 = \textcolor{green}{0.0015}$$



Best Sequence:  
 Hot  Cold



$$P(C)*P(H|C)*P(3|H) = 0.064 * 0.5 * 0.2 = 0.0064$$

$$P(H)*P(H|H)*P(3|H) = 0.0384 * 0.6 * 0.2 = 0.0046$$

	Hot	Cold
<S>	0.8	0.2
A	Hot	Cold

	Hot	Cold
Hot	0.6	0.4
Cold	0.5	0.5

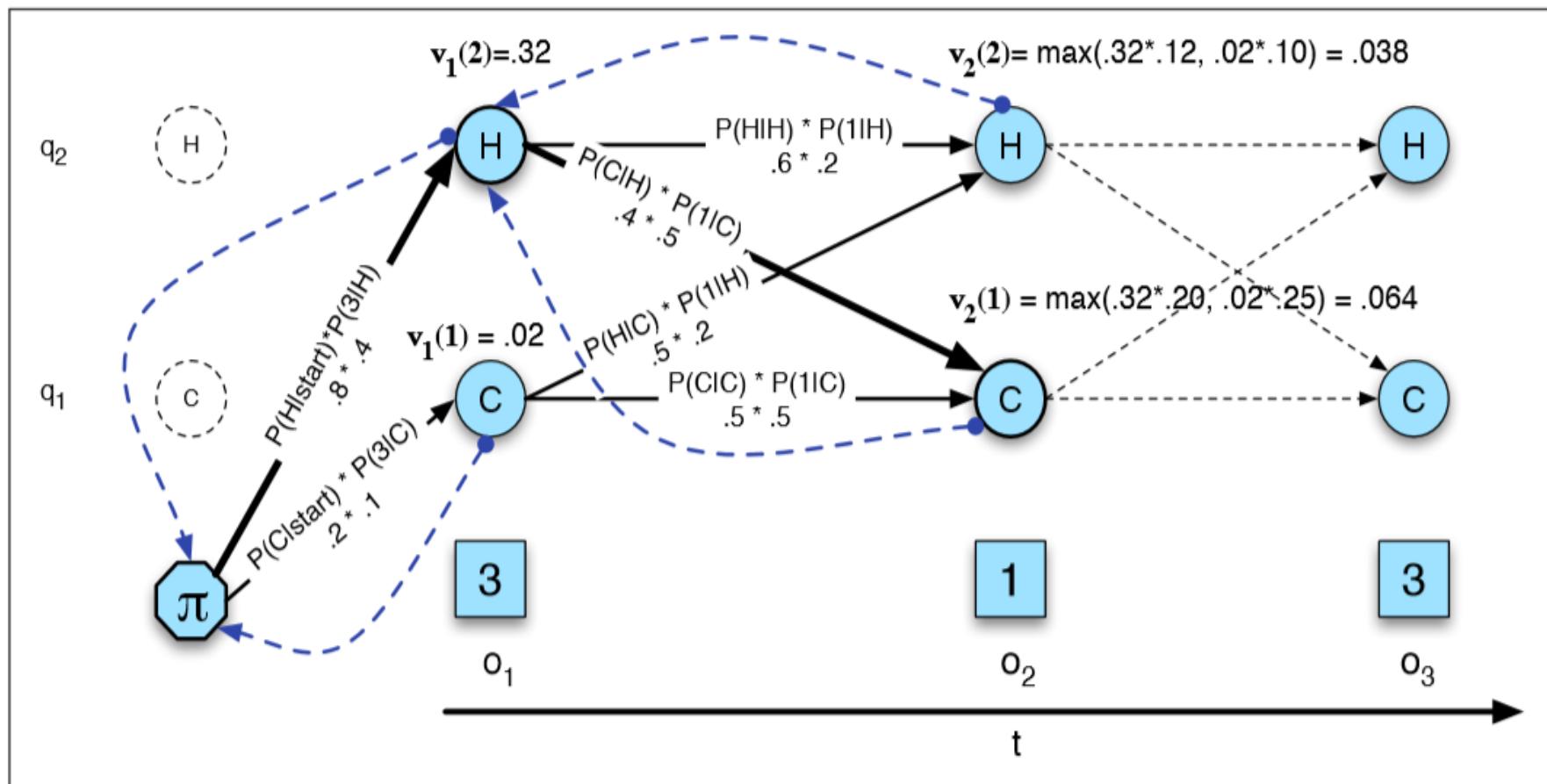
B	1	2	3
Hot	.2	.4	.4
Cold	.5	.4	.1

# Hidden Markov Model

## Example -1 – Viterbi Algorithm

Handwritten notes:

States: H, C  
Observations: 1, 2, 3  
Sequence: 1, 2, 3



**Figure A.10** The Viterbi backtrace. As we extend each path to a new state account for the next observation, we keep a backpointer (shown with broken lines) to the best path that led us to this state.

Source Credit : Speech and Language Processing - Jurafsky and Martin

## POS Tagging – Viterbi Algorithm

**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**

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

$backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^N viterbi[s',t-1] * a_{s',s}$

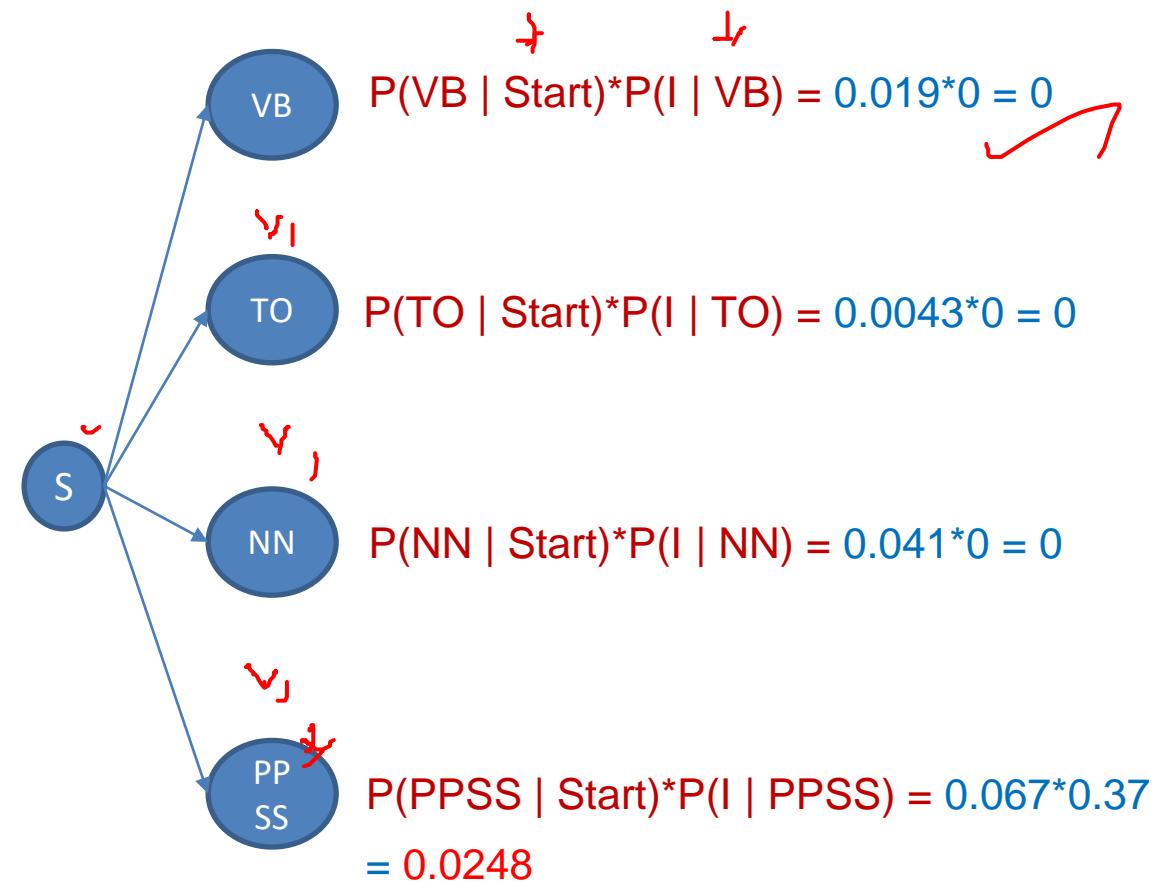
$viterbi[q_F, T] \leftarrow \max_{s=1}^N viterbi[s, T] * a_{s,q_F}$  ; termination step

$backpointer[q_F, T] \leftarrow \operatorname{argmax}_{s=1}^N viterbi[s, T] * a_{s,q_F}$  ; termination step

**return** the backtrace path by following backpointers to states back in time from  $backpointer[q_F, T]$

# Hidden Markov Model

## POS Tagging – Example -2 – Viterbi Algorithm - Initialization



$$V[t, 1] = P(t|start)P(w_1|t)$$

A	VB	TO	NN	PPSS
<S>	0.019	0.0043	0.041	0.067

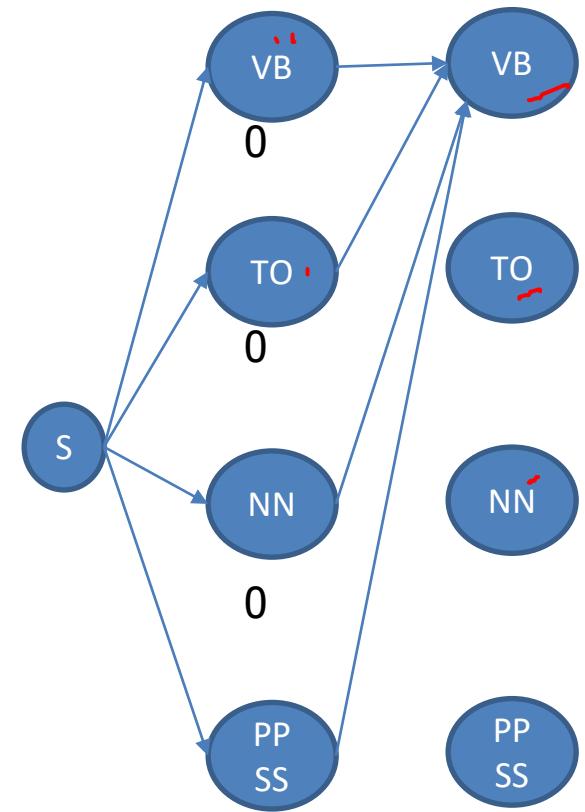
A	VB	TO	NN	PPSS
VB	0.0038	0.035	0.047	0.0070
TO	0.83	0	0.00047	0
NN	0.0040	0.016	0.087	0.0045
PPSS	0.23	0.00079	0.0012	0.00014

B	I	want	to	race
VB	0	0.0093	0	0.00012
TO	0	0	0.99	0
NN	0	0.000054	0	0.00057
PPSS	0.37	0	0	0

# Hidden Markov Model

## POS Tagging – Example -2 – Viterbi Algorithm - Recursion

~~0.0248~~



$$V[t, j] = \max_{t'} V[t', j - 1] P(t|t') P(w_j|t)$$

	VB	TO	NN	PPSS
<S>	0.019	0.0043	0.041	0.067
A	VB	TO	NN	PPSS
VB	0.0038	0.035	0.047	0.0070
TO	0.83	0	0.00047	0
NN	0.0040	0.016	0.087	0.0045
PPSS	0.23	0.00079	0.0012	0.00014

B	I	want	to	race
VB	0	0.0093	0	0.00012
TO	0	0	0.99	0
NN	0	0.000054	0	0.00057
PPSS	0.37	0	0	0

$$P(VB) * P(VB | VB)*P(want | VB) = 0*0.0038*0.0093= 0$$

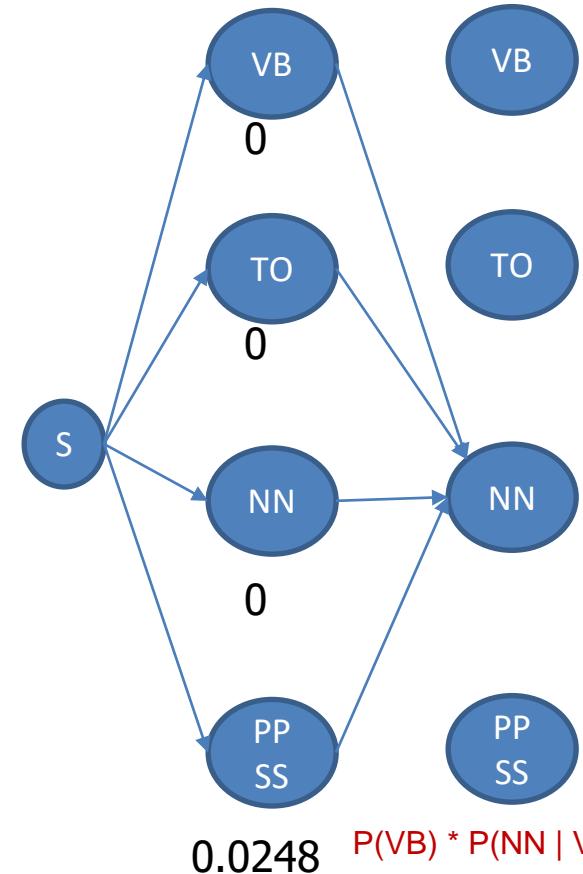
$$P(TO) * P(VB | TO)*P(want | VB) = 0*0.83*0.0093= 0$$

$$P(NN) * P(VB | NN)*P(want | VB) = 0*0.004*0.0093= 0$$

$$P(PPSS) * P(VB | PPSS)*P(want | VB) = 0.0248*0.23*0.0093= 0.0005$$

# Hidden Markov Model

## POS Tagging – Example -2 – Viterbi Algorithm - Recursion



$$V[t, j] = \max_{t'} V[t', j - 1] P(t|t') P(w_j|t)$$

	VB	TO	NN	PPSS
<S>	0.019	0.0043	0.041	0.067
A	VB	TO	NN	PPSS
VB	0.0038	0.035	0.047	0.0070
TO	0.83	0	0.00047	0
NN	0.0040	0.016	0.087	0.0045
PPSS	0.23	0.00079	0.0012	0.00014

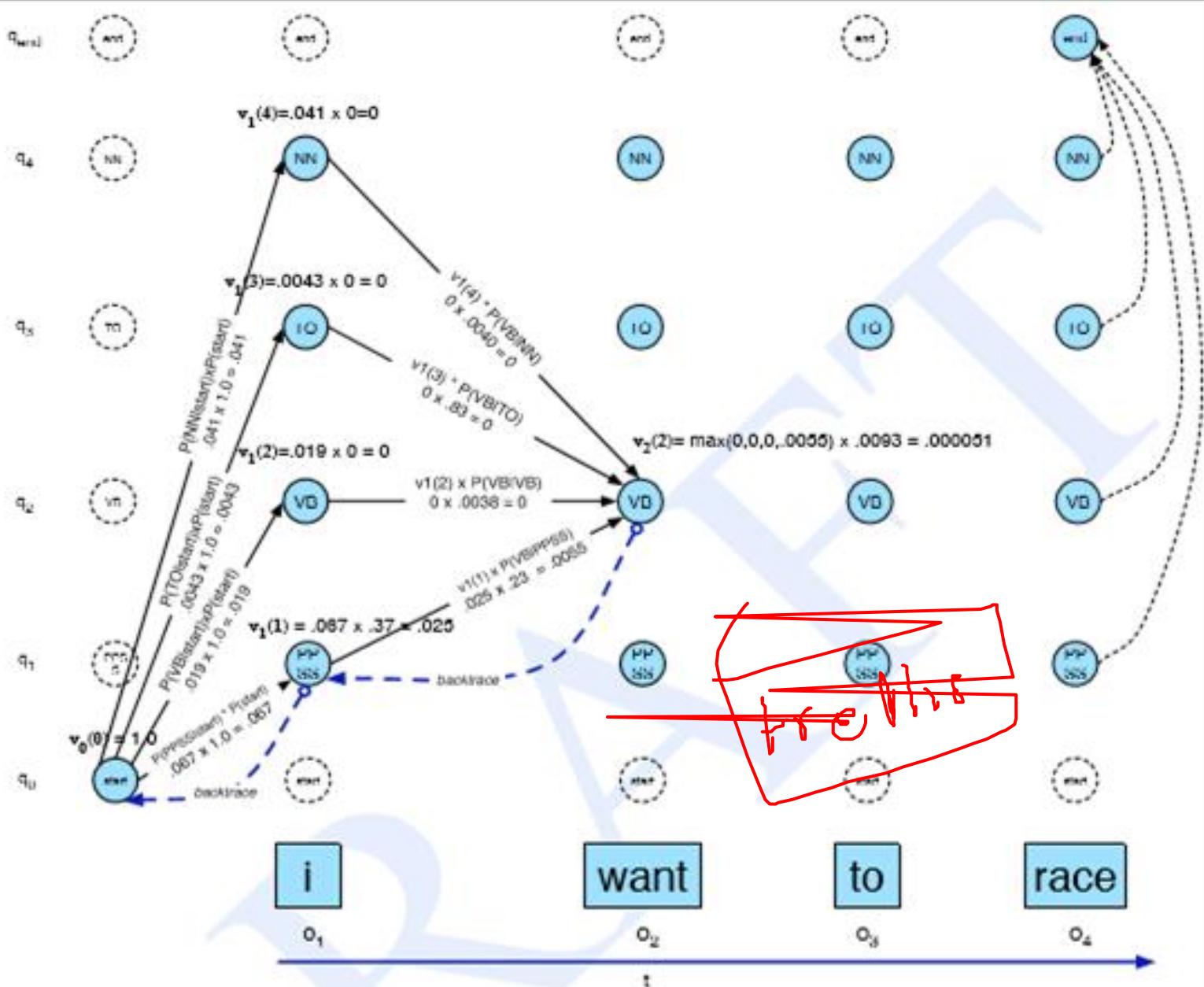
B	I	want	to	race
VB	0	0.0093	0	0.00012
TO	0	0	0.99	0
NN	0	0.000054	0	0.00057
PPSS	0.37	0	0	0

$$P(VB) * P(NN | VB)*P(want | NN) = 0*0.047*0.000054= 0$$

$$P(TO) * P(NN | TO)*P(want | NN) = 0*0.0047*0.000054= 0$$

$$P(NN) * P(NN | NN)*P(want | NN) = 0*0.087*0.000054= 0$$

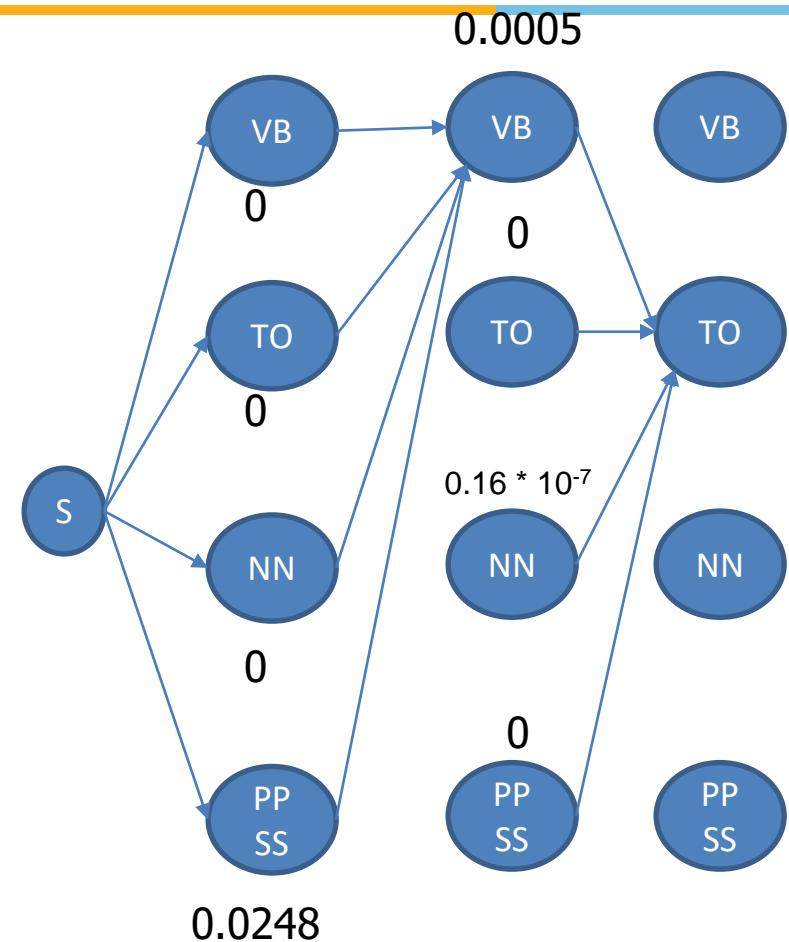
$$P(PPSS) * P(NN | PPSS)*P(want | NN) = 0.0012*0.23*0.000054= 0.16 * 10^{-7}$$



Source Credit : Speech and Language Processing - Jurafsky and Martin

# Hidden Markov Model

## POS Tagging – Example -2 – Viterbi Algorithm - Recursion



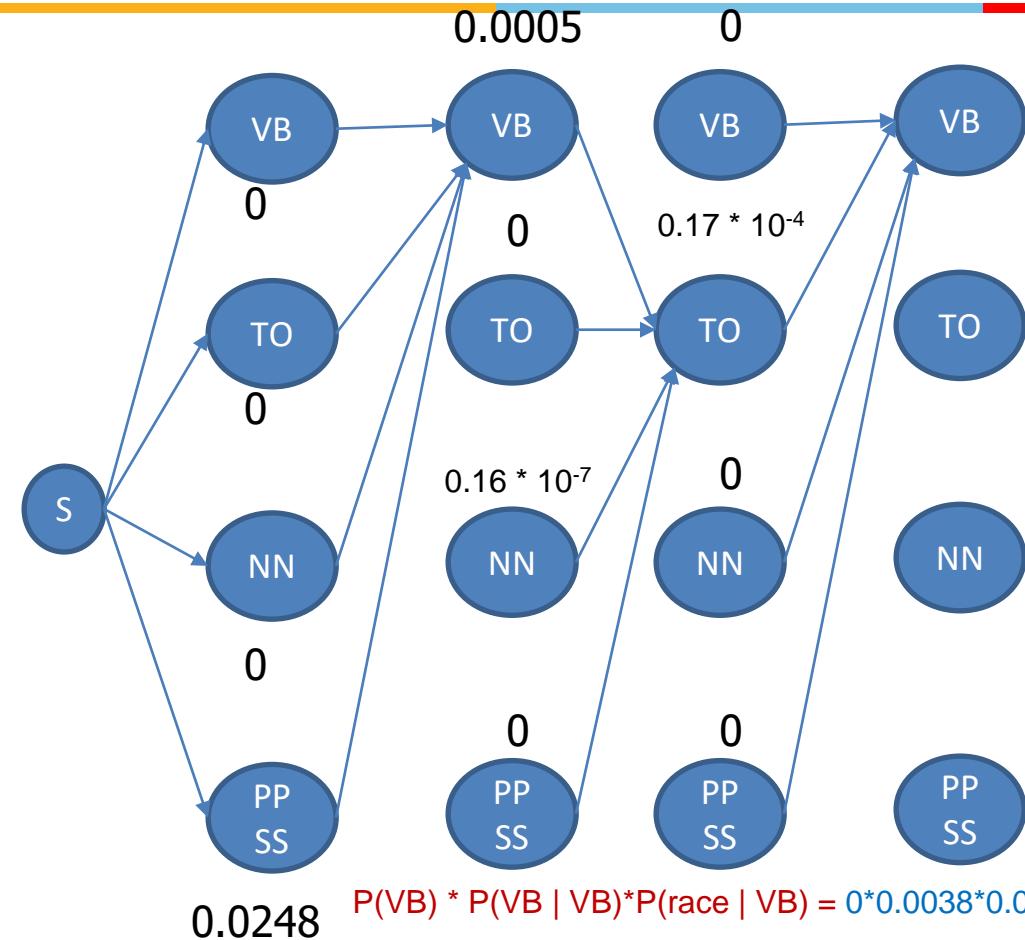
$$V[t, j] = \max_{t'} V[t', j - 1] P(t|t') P(w_j|t)$$

	VB	TO	NN	PPSS
<S>	0.019	0.0043	0.041	0.067
A	VB	TO	NN	PPSS
VB	0.0038	0.035	0.047	0.0070
TO	0.83	0	0.00047	0
NN	0.0040	0.016	0.087	0.0045
PPSS	0.23	0.00079	0.0012	0.00014

B	I	want	to	race
VB	0	0.0093	0	0.00012
TO	0	0	0.99	0
NN	0	0.000054	0	0.00057
PPSS	0.37	0	0	0

# Hidden Markov Model

## POS Tagging – Example -2 – Viterbi Algorithm - Recursion



$$P(VB) * P(VB | VB) * P(race | VB) = 0 * 0.0038 * 0.00012 = 0$$

$$\begin{aligned} P(TO) * P(VB | TO) * P(race | VB) &= 0.17 * 10^{-4} * 0.83 * 0.00012 \\ &= 0.17 * 10^{-8} \end{aligned}$$

$$P(NN) * P(VB | NN) * P(race | VB) = 0 * 0.0040 * 0.00012 = 0$$

$$P(PPSS) * P(VB | PPSS) * P(race | VB) = 0 * 0.23 * 0.00012 = 0$$

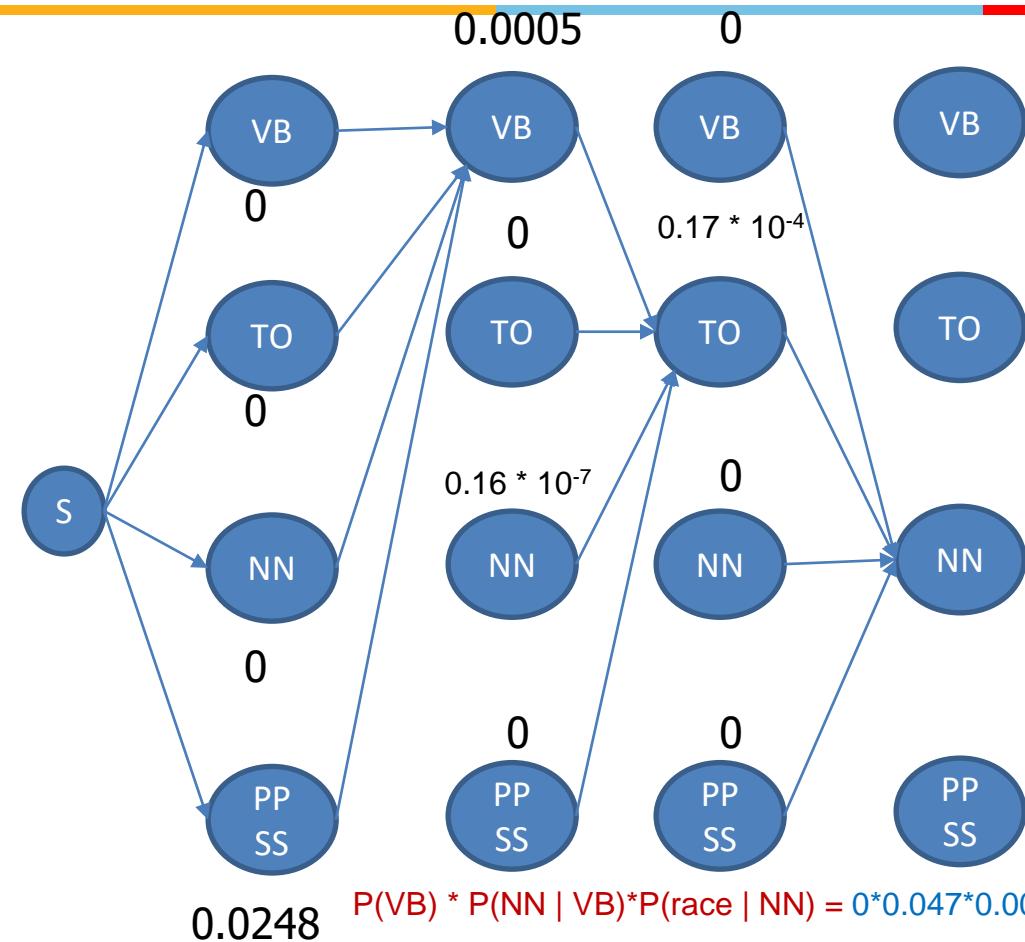
$$V[t, j] = \max_{t'} V[t', j - 1] P(t|t') P(w_j|t)$$

	VB	TO	NN	PPSS
<S>	0.019	0.0043	0.041	0.067
A	VB	TO	NN	PPSS
VB	0.0038	0.035	0.047	0.0070
TO	0.83	0	0.00047	0
NN	0.0040	0.016	0.087	0.0045
PPSS	0.23	0.00079	0.0012	0.00014

B	I	want	to	race
VB	0	0.0093	0	0.00012
TO	0	0	0.99	0
NN	0	0.000054	0	0.00057
PPSS	0.37	0	0	0

# Hidden Markov Model

## POS Tagging – Example -2 – Viterbi Algorithm - Recursion



$$V[t, j] = \max_{t'} V[t', j - 1] P(t|t') P(w_j|t)$$

	VB	TO	NN	PPSS
<S>	0.019	0.0043	0.041	0.067
A	VB	TO	NN	PPSS
VB	0.0038	0.035	0.047	0.0070
TO	0.83	0	0.00047	0
NN	0.0040	0.016	0.087	0.0045
PPSS	0.23	0.00079	0.0012	0.00014

B	I	want	to	race
VB	0	0.0093	0	0.00012
TO	0	0	0.99	0
NN	0	0.000054	0	0.00057
PPSS	0.37	0	0	0

$$P(VB) * P(NN | VB) * P(race | NN) = 0 * 0.047 * 0.00057$$

$$P(TO) * P(NN | TO) * P(race | NN) = 0.17 * 10^{-4} * 0.00047 * 0.00057$$

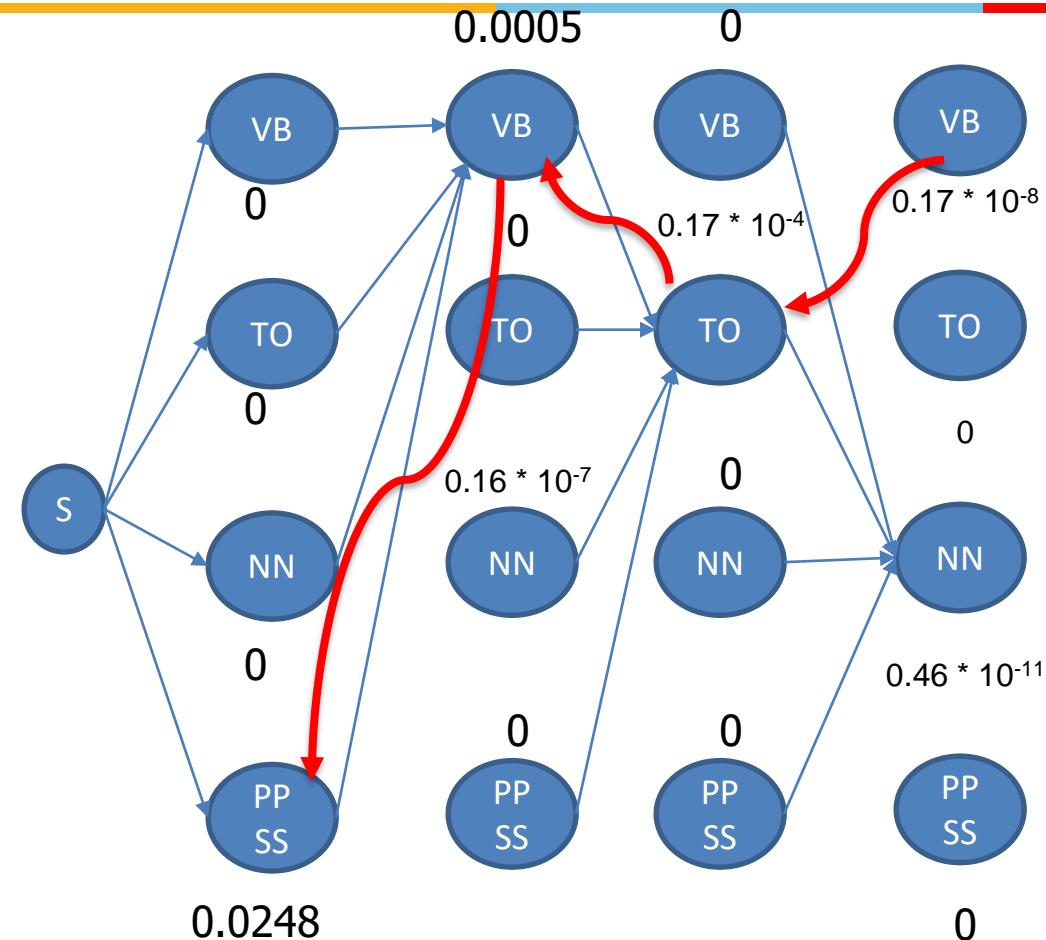
$$= 0.46 * 10^{-11}$$

$$P(NN) * P(NN | NN) * P(race | NN) = 0 * 0.087 * 0.00057 = 0$$

$$P(PPSS) * P(NN | PPSS) * P(race | NN) = 0 * 0.0012 * 0.00057 = 0$$

# Hidden Markov Model

## POS Tagging – Example -2 – Viterbi Algorithm - Termination



$$V[t, j] = \max_{t'} V[t', j - 1] P(t|t') P(w_j|t)$$

	VB	TO	NN	PPSS
<S>	0.019	0.0043	0.041	0.067
A	VB	TO	NN	PPSS
VB	0.0038	0.035	0.047	0.0070
TO	0.83	0	0.00047	0
NN	0.0040	0.016	0.087	0.0045
PPSS	0.23	0.00079	0.0012	0.00014

B	I	want	to	race
VB	0	0.0093	0	0.00012
TO	0	0	0.99	0
NN	0	0.000054	0	0.00057
PPSS	0.37	0	0	0

# Hidden Markov Model



## POS Tagging – Example -3 – Viterbi Algorithm – Practice

Transition matrix:  $P(t_i|t_{i-1})$

	NOUN	Verb	Det	Prep	ADV	STOP
<S>	.3	.1	.3	.2	.1	0
Noun	.2	.4	.01	.3	.04	.05
Verb	.3	.05	.3	.2	.1	.05
Det	.9	.01	.01	.01	.07	0
Prep	.4	.05	.4	.1	.05	0
Adv	.1	.5	.1	.1	.1	.1

Emission matrix:  $P(w_i|t_i)$

	a	cat	doctor	in	is	the	very
Noun	0	.5	.4	0	0.1	0	0
Verb	0	0	.1	0	.9	0	0
Det	.3	0	0	0	0	.7	0
Prep	0	0	0	1.0	0	0	0
Adv	0	0	0	.1	0	0	.9

$$V[t, 1] = P(t|start)P(w_1|t)$$

	w1=the	w2=doctor	w3=is	w4=in	STOP
Noun	0				
Verb	0				
Det	.21				
Prep	0				
Adv	0				

$$V(\text{Noun}, \text{the}) = P(\text{Noun}|<\text{S}>)P(\text{the}|\text{Noun}) = .3 \times 0 = 0$$

$$V(\text{Verb}, \text{the}) = P(\text{Verb}|<\text{S}>)P(\text{the}|\text{Verb}) = .1 \times 0 = 0$$

$$V(\text{Det}, \text{the}) = P(\text{Det}|<\text{S}>)P(\text{the}|\text{Det}) = .3 \times .7 = .21$$

$$V(\text{Prep}, \text{the}) = P(\text{Prep}|<\text{S}>)P(\text{the}|\text{Prep}) = .2 \times 0 = 0$$

$$V(\text{Adv}, \text{the}) = P(\text{Adv}|<\text{S}>)P(\text{the}|\text{Adv}) = .2 \times 0 = 0$$

## POS Tagging – Example -3 – Viterbi Algorithm – Practice

$$V(\text{Noun}, \text{doctor}) = \max_{t'} V(t', \text{the})XP(\text{Noun}|t')X P(\text{doctor}|\text{Noun}) \\ = \max \{0, 0, .21 (.9 \times .4), 0, 0\} = .0756$$

$$V(\text{Verb}, \text{doctor}) = \max_{t'} V(t', \text{the})XP(\text{Verb}|t')X P(\text{doctor}|\text{Verb}) \\ = \max \{0, 0, .21(.01 \times .1), 0, 0\} = .00021$$

	w1=the	w2=doctor	w3=is	w4=in	STOP
Noun	0	.0756			
Verb	0	.00021			
Det	.21	0			
Prep	0	0			
Adv	0	0			

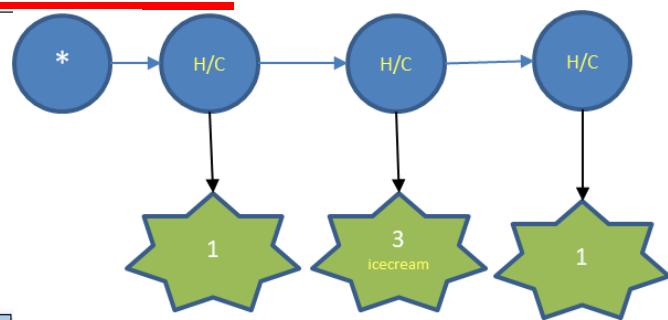
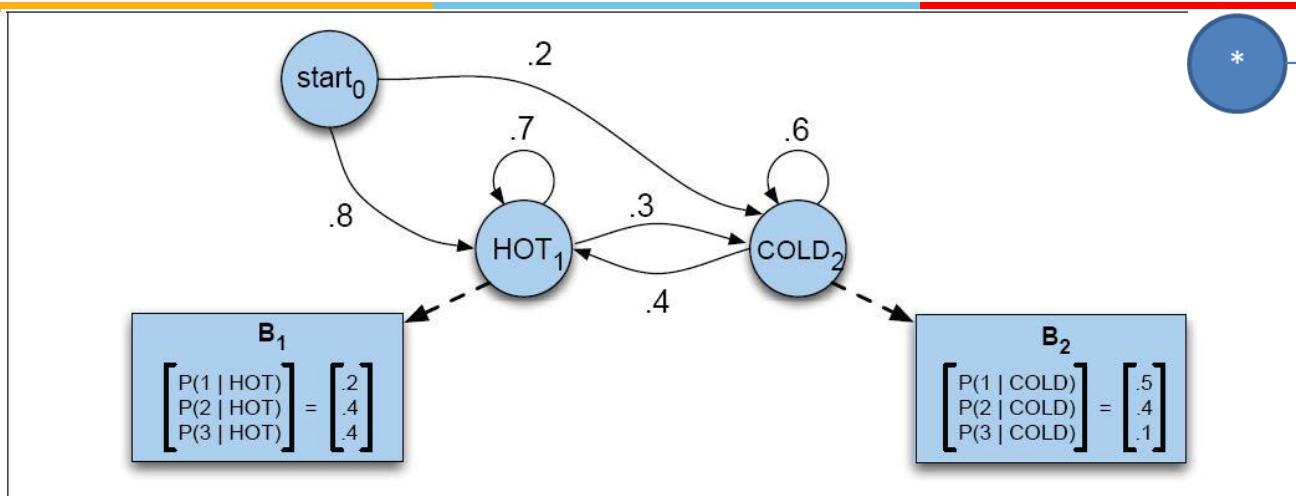
	w1=the	w2=doctor	w3=is	w4=in	STOP
Noun	0	.0756	.001512	0	
Verb	0	.00021	.027216	0	
Det	.21	0	0	0	.0000272
Prep	0	0	0	.005443	
Adv	0	0	0	.000272	

Det      Noun      Verb      Prep

# Hidden Markov Model



## Example -4 – Naïve Search

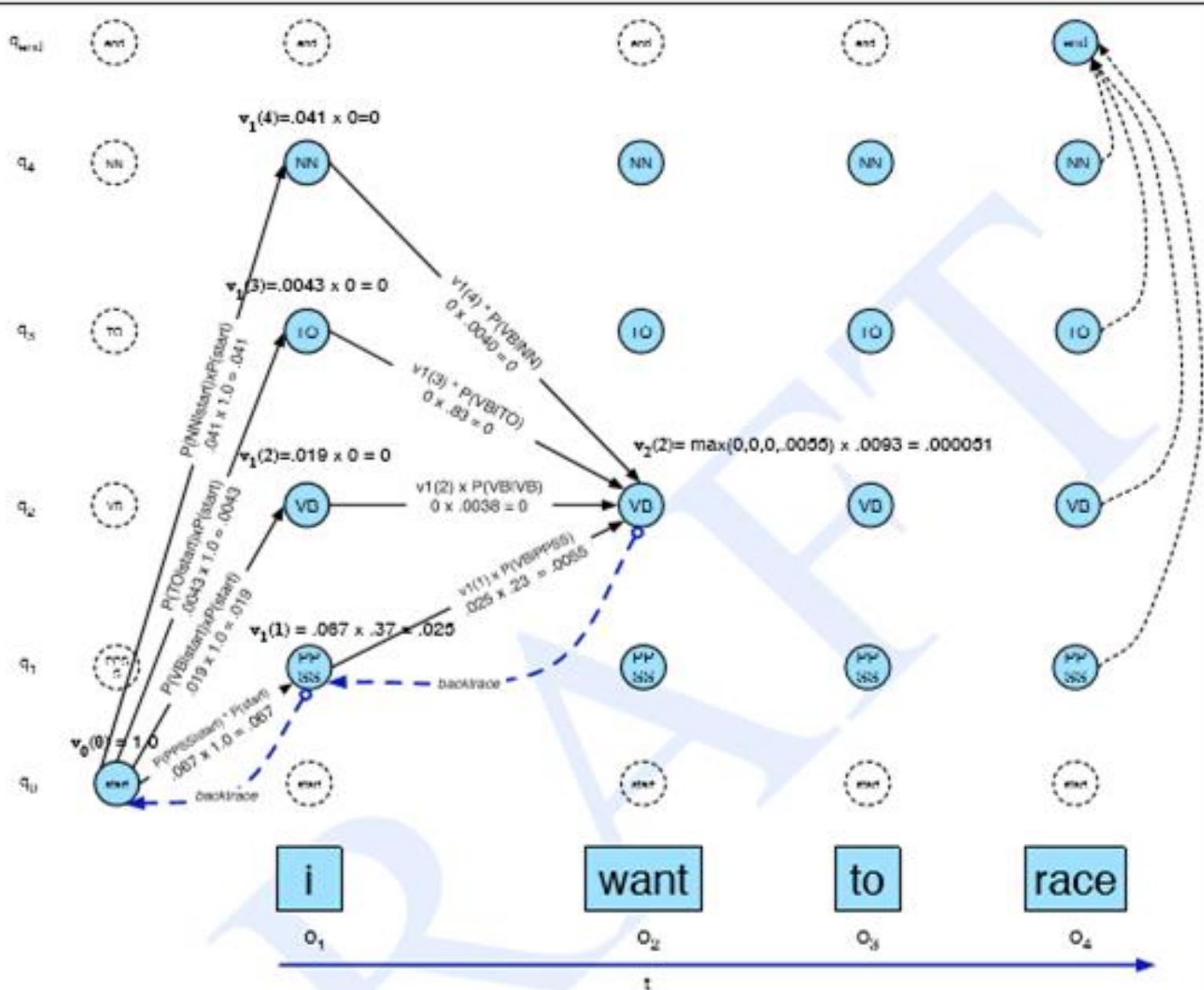


HHH	$P(H)^*P(1 H)^*P(H H)^*P(3 H)^*P(H H)^*P(1 H)$	
HHC		
HCC		
CCC		
CHC	$P(C)^*P(1 C)^*P(H C)^*P(3 H)^*P(C H)^*P(1 C)$ $=0.2^*0.5^*0.4^*0.4^*0.3^*0.5$	0.0024
CCH		
CHH		
HCH		

	Hot	Cold
<S>	0.8	0.2

A	Hot	Cold
Hot	0.7	0.3
Cold	0.4	0.6

B	1	2	3
Hot	.2	.4	.4
Cold	.5	.4	.1

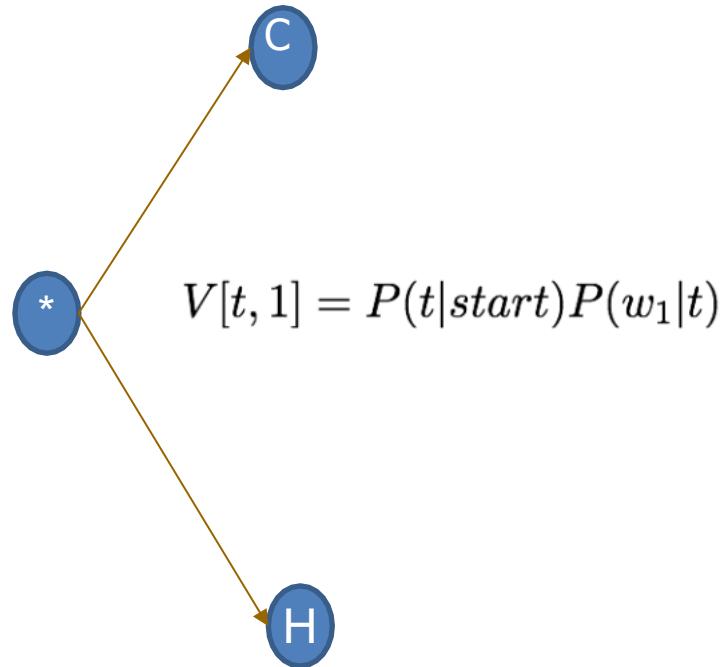


Source Credit : Speech and Language Processing - Jurafsky and Martin

# Hidden Markov Model

## Example – 5 – Observation Likelihood by Forward Propagation

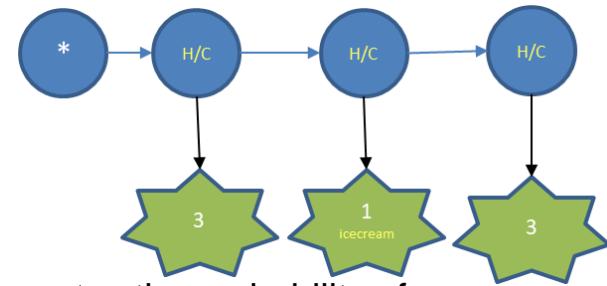
$$P(C)*P(3|C) = 0.2*0.1 = 0.02$$



$$P(H)*P(3|H) = 0.8*0.4 = 0.32$$

For our particular case, we would sum over the eight 3-event sequences *cold cold cold*, *cold cold hot*, that is,

$$P(3\ 1\ 3) = P(3\ 1\ 3, \text{cold cold cold}) + P(3\ 1\ 3, \text{cold cold hot}) + P(3\ 1\ 3, \text{hot hot cold}) + \dots$$



Efficiently computes the probability of an observed sequence given a model

$$P(\text{sequence} \mid \text{model})$$

Identical to Viterbi; **replace the MAX with a SUM**

	Hot	Cold
<S>	0.8	0.2
A		
Hot	0.6	0.4
Cold	0.5	0.5

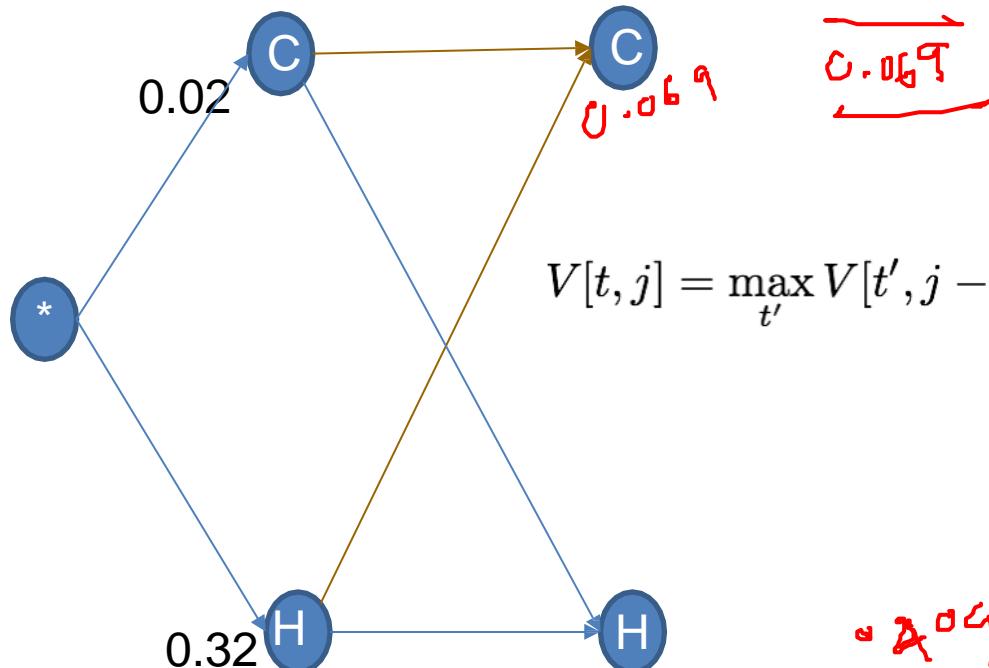
B	1	2	3
Hot	.2	.4	.4
Cold	.5	.4	.1

# Hidden Markov Model

## Example -5 – Observation Likelihood by Forward Propagation - Recursion

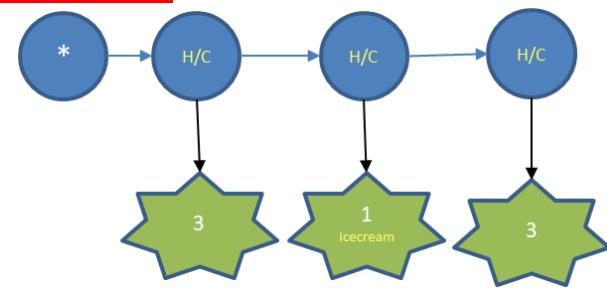
$$P(C) * P(C|C) * P(1|C) = 0.02 * 0.5 * 0.5 = 0.005$$

$$P(H) * P(C|H) * P(1|C) = 0.32 * 0.4 * 0.5 = 0.064$$



$$P(C) * P(H|C) * P(1|H) = 0.02 * 0.5 * 0.2 = 0.002$$

$$P(H) * P(H|H) * P(1|H) = 0.32 * 0.6 * 0.2 = 0.0384$$



	Hot	Cold
<S>	0.8	0.2
A		
Hot	0.6	0.4
Cold	0.5	0.5

A	Hot	Cold
Hot	0.6	0.4
Cold	0.5	0.5

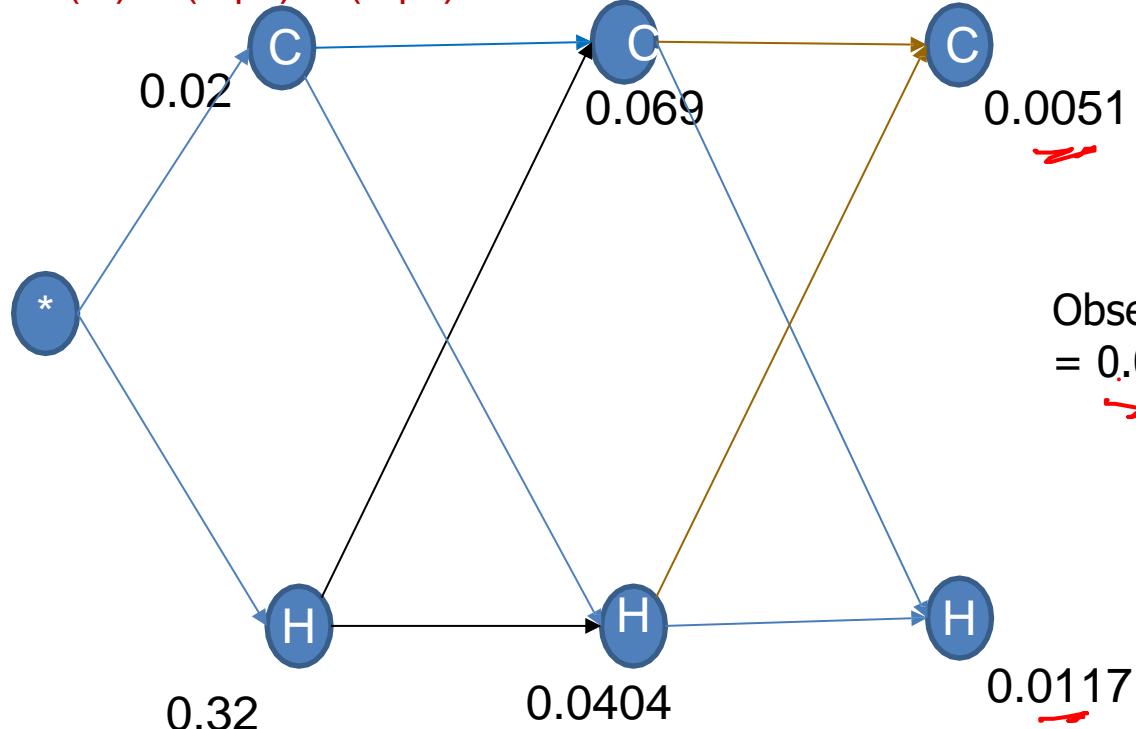
B	1	2	3
Hot	.2	.4	.4
Cold	.5	.4	.1

# Hidden Markov Model

## Example -5 – Observation Likelihood by Forward Propagation

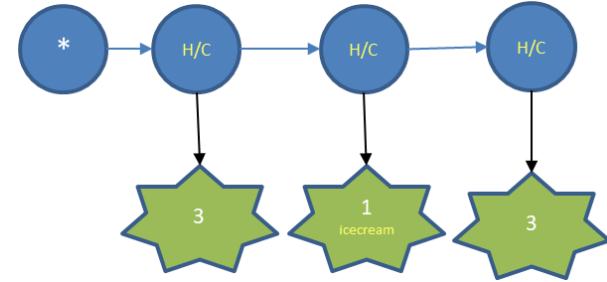
$$P(C) * P(C|C) * P(R|C) = 0.069 * 0.5 * 0.1 = \textcolor{red}{0.0035}$$

$$P(H) * P(C|H) * P(R|C) = 0.0404 * 0.4 * 0.1 = \textcolor{green}{0.0016}$$



$$P(C) * P(H|C) * P(R|H) = 0.069 * 0.5 * 0.2 = 0.0069$$

$$P(H) * P(H|H) * P(R|H) = 0.0404 * 0.6 * 0.2 = 0.0048$$

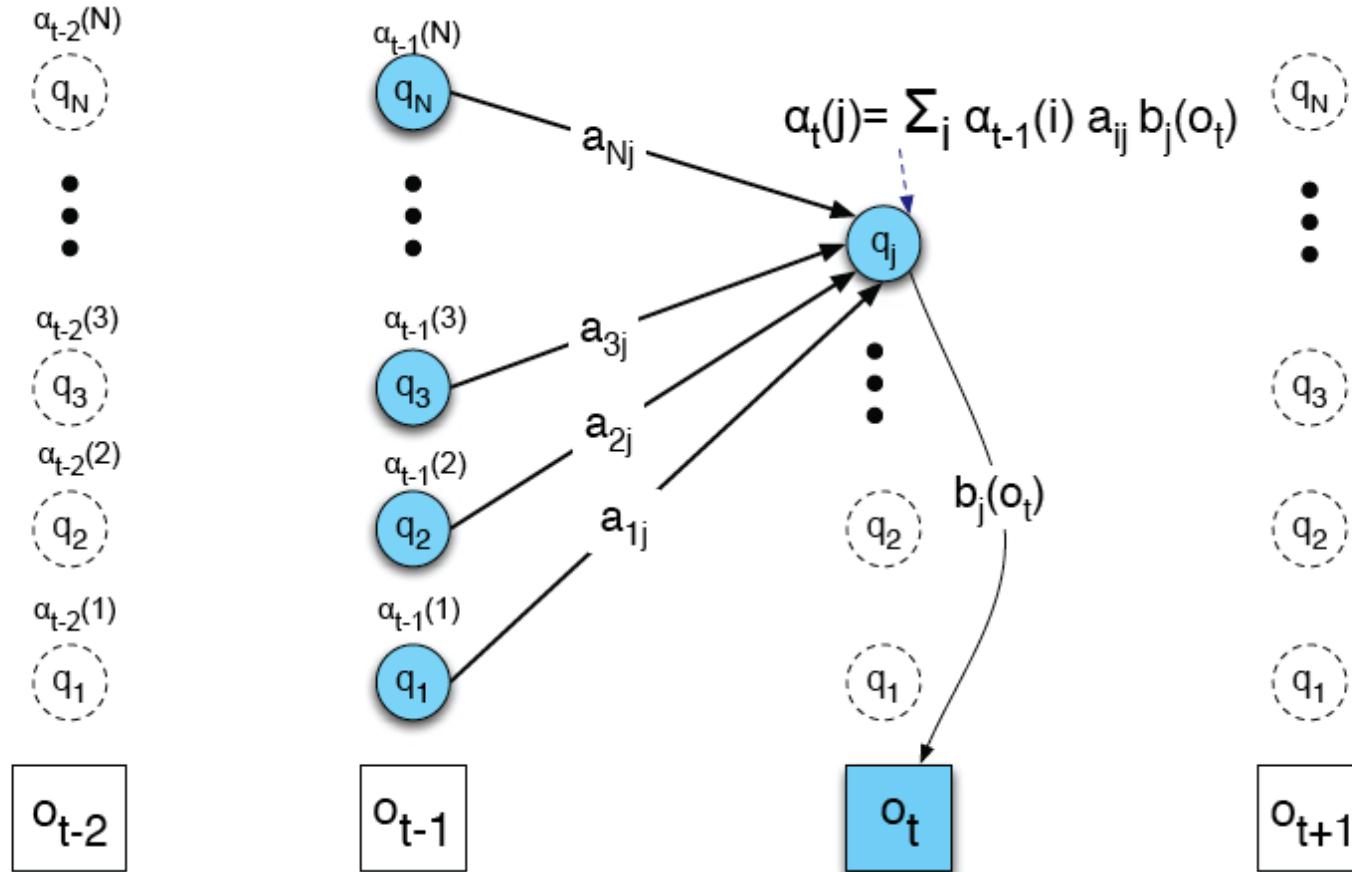


	Hot	Cold
<S>	0.8	0.2
A	Hot	Cold

	Hot	Cold
Hot	0.6	0.4
Cold	0.5	0.5

B	1	2	3
Hot	.2	.4	.4
Cold	.5	.4	.1

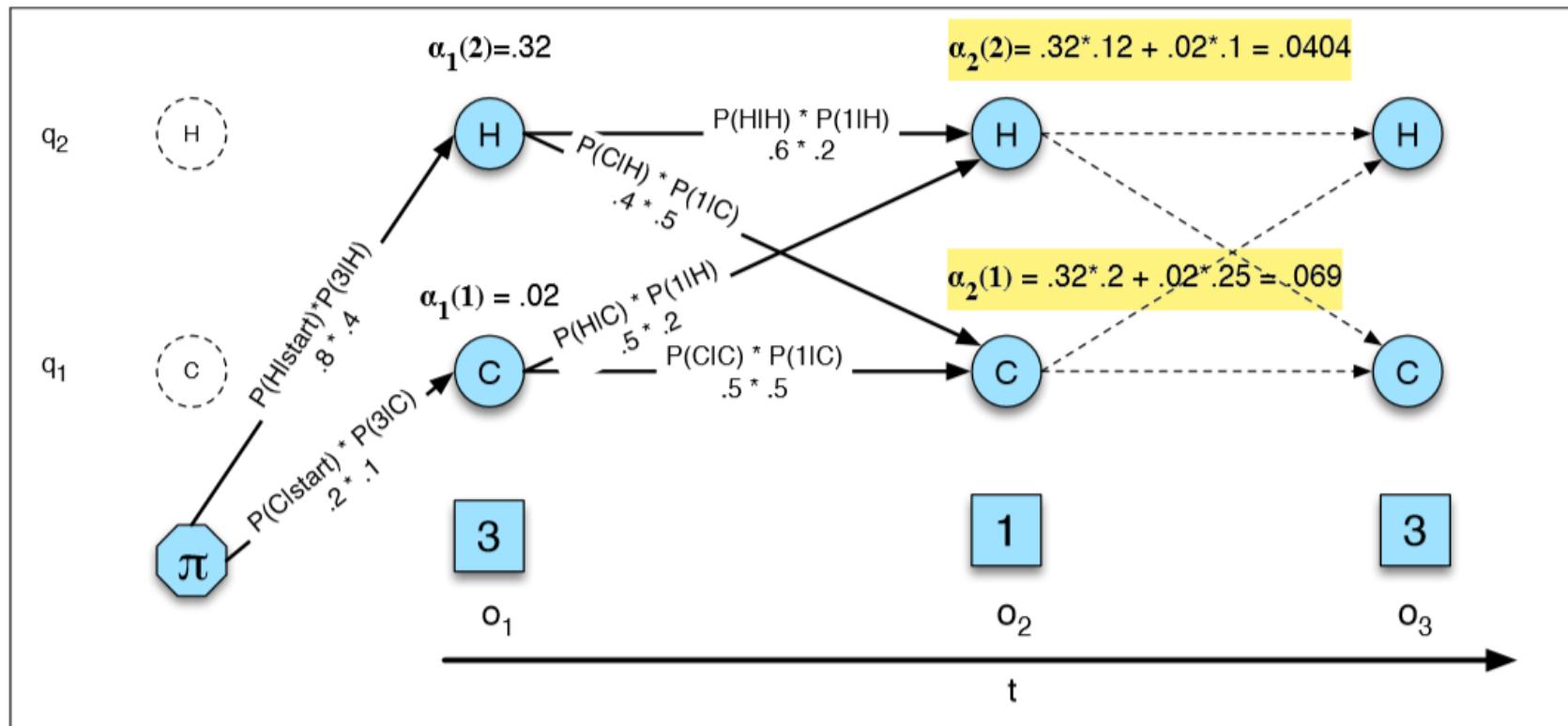
## Observation Likelihood – Forward Propagation



Source Credit : Speech and Language Processing - Jurafsky and Martin

# Hidden Markov Model

## Observation Likelihood – Forward Propagation



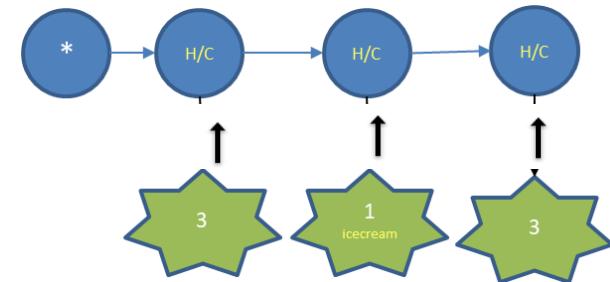
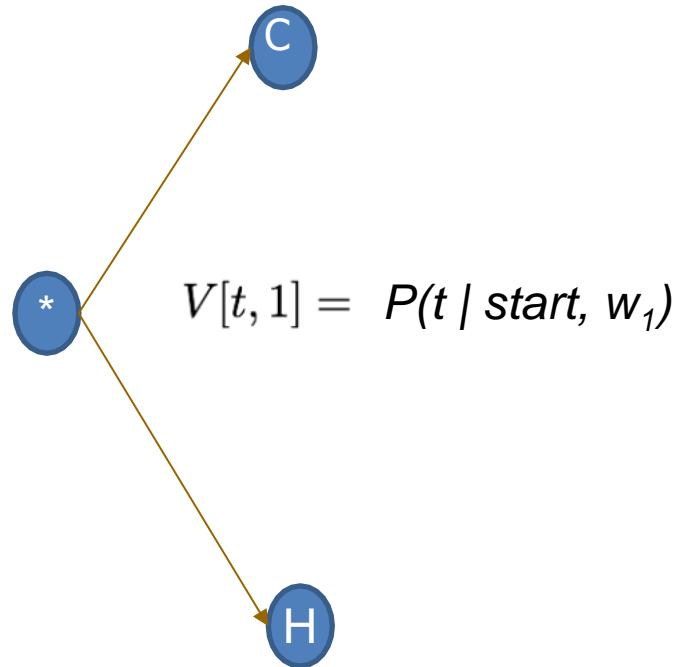
**Figure A.5** The forward trellis for computing the total observation likelihood for the ice-cream events 3 1 3. Hidden states are in circles, observations in squares. The figure shows the computation of  $\alpha_t(j)$  for two states at two time steps. The computation in each cell follows Eq. A.12:  $\alpha_t(j) = \sum_{i=1}^N \alpha_{t-1}(i) a_{ij} b_j(o_t)$ . The resulting probability expressed in each cell is Eq. A.11:  $\alpha_t(j) = P(o_1, o_2 \dots o_t, q_t = j | \lambda)$ .

Source Credit : Speech and Language Processing - Jurafsky and Martin

# Maximum Entropy Markov Model

## Example -6 – Viterbi Algorithm - Initialization

$$P(C \mid \text{Start}, 3) = 0.55$$



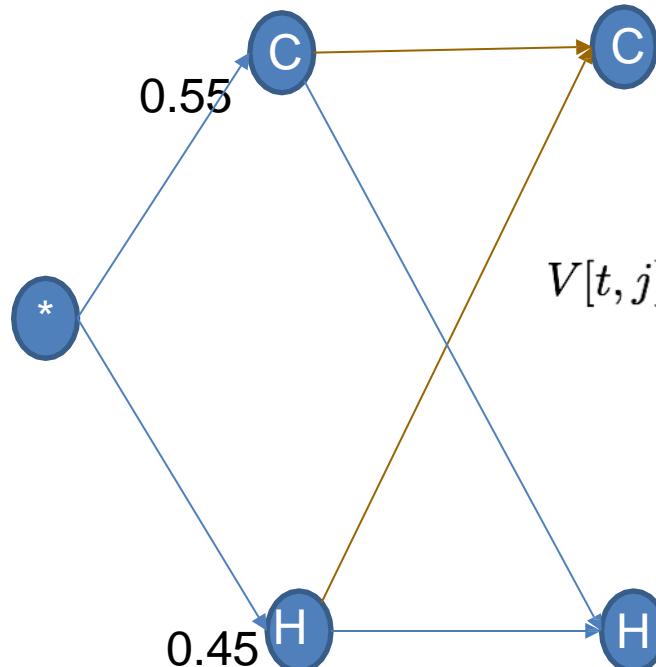
$X_1 = T_{i-1}$	$X_2 = w_i$	$\text{Label } Y = T_i$	$P(T_i   T_{i-1} W_i)$
Start	1	Hot	0.3
Start	1	Cold	
Start	3	Hot	0.45
Start	3	Cold	
.....	....	...	.....
Hot	1	Hot	0.15
Hot	1	Cold	
Hot	3	Hot	0.6
Hot	3	Cold	
.....	....	....	.....
Cold	1	Hot	0.5
Cold	1	Cold	
Cold	3	Hot	0.8
Cold	3	Cold	
....	....	....	....

# Maximum Entropy Markov Model

## Example -6 – Viterbi Algorithm - Recursion

$$P(C) * P(C|C, 1) = 0.55 * 0.5 = 0.275$$

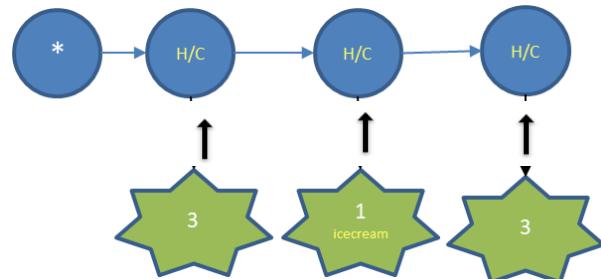
$$P(H) * P(C|H, 1) = 0.45 * 0.85 = 0.3825$$



$$V[t, j] = \max_{t'} V[t', j - 1] P(t | t', w_j)$$

$$P(C) * P(H|C, 1) = 0.55 * 0.5 = 0.275$$

$$P(H) * P(H|H, 1) = 0.45 * 0.15 = 0.0675$$



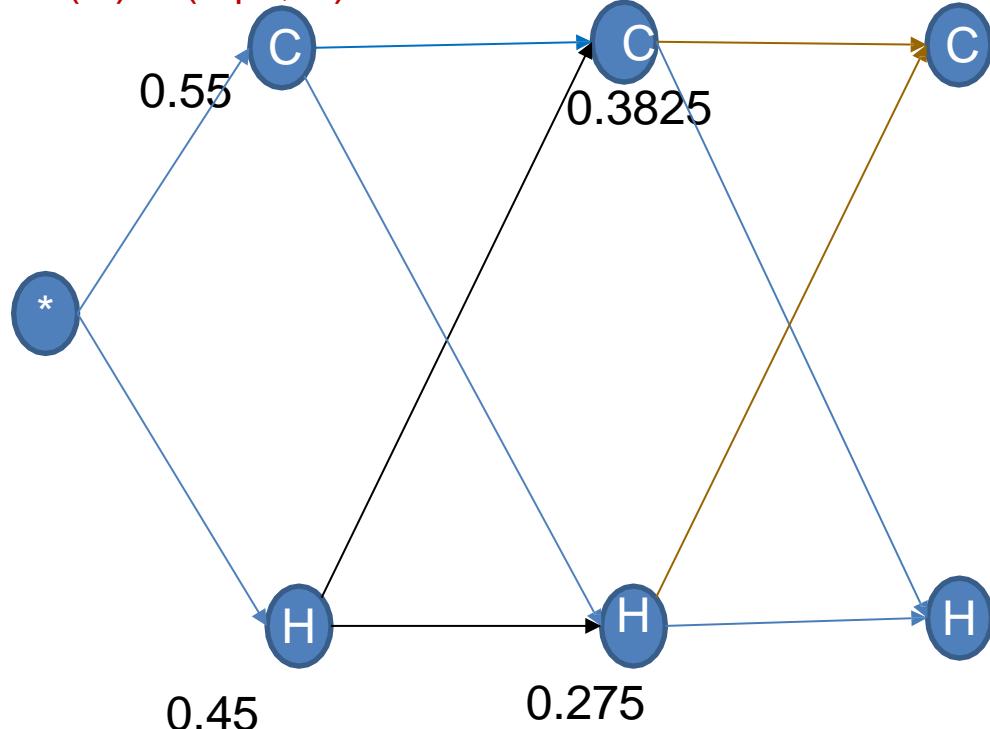
$X_1 = T_{i-1}$	$X_2 = w_i$	Label $Y = T_i$	$P(T_i   T_{i-1}, w_i)$
Start	1	Hot	0.3
Start	1	Cold	
Start	3	Hot	0.45
Start	3	Cold	
.....	....	...	.....
Hot	1	Hot	0.15
Hot	1	Cold	
Hot	3	Hot	0.6
Hot	3	Cold	
.....	....	....	.....
Cold	1	Hot	0.5
Cold	1	Cold	
Cold	3	Hot	0.8
Cold	3	Cold	
....	....	....	.....

# Maximum Entropy Markov Model

## Example -6 – Viterbi Algorithm – Termination through Back Trace

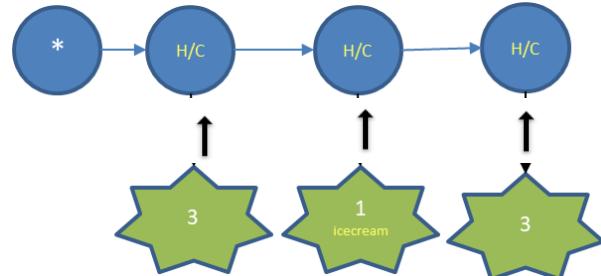
$$P(C) * P(C|C, 3) = 0.3825 * 0.2 = 0.0765$$

$$P(H) * P(C|H, 3) = 0.275 * 0.4 = 0.11$$



$$P(C) * P(H|C, 3) = 0.3825 * 0.8 = 0.306$$

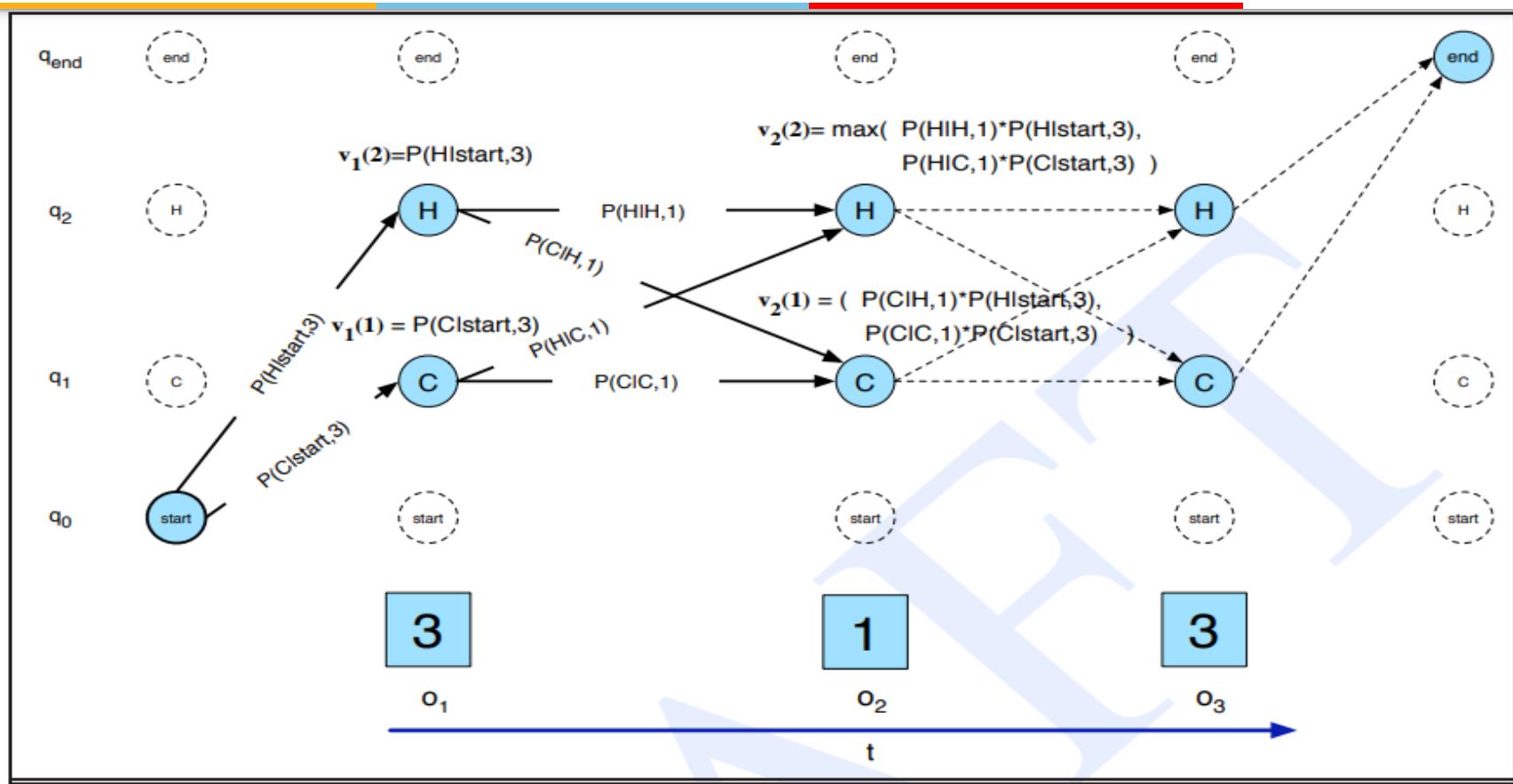
$$P(H) * P(H|H, 3) = 0.275 * 0.6 = 0.165$$



$X_1 = T_{i-1}$	$X_2 = w_i$	Label $Y = T_i$	$P(T_i / T_{i-1} W_i)$
Start	1	Hot	0.3
Start	1	Cold	
Start	3	Hot	0.45
Start	3	Cold	
.....	....	...	.....
Hot	1	Hot	0.15
Hot	1	Cold	
Hot	3	Hot	0.6
Hot	3	Cold	
.....	....	....	.....
Cold	1	Hot	0.5
Cold	1	Cold	
Cold	3	Hot	0.8
Cold	3	Cold	
....	....	....	....

# Maximum Entropy Markov Model

## Example -6 – Viterbi Algorithm



**Figure 6.22** Inference from ice-cream eating computed by an MEMM instead of an HMM. The Viterbi trellis for computing the best path through the hidden state space for the ice-cream eating events 3 1 3, modified from the HMM figure in Fig. 6.10.

Source Credit : Speech and Language Processing - Jurafsky and Martin

# POS Tagging – Viterbi Algorithm

**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  $backpointer[q_F, T]$

# Modified for Maximum Entropy Markov Model



## POS Tagging – Viterbi Algorithm

**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

$viterbi[s,1] \leftarrow P(s | O_0, O_1)$

$backpointer[s,1] \leftarrow 0$

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

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

$viterbi[s,t] \leftarrow \max_{s'=1}^N viterbi[s',t-1] * P(s | S', O_t)$

$backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^N viterbi[s',t-1] * P(s | S', O_t)$

$viterbi[q_F, T] \leftarrow \max_{s=1}^N viterbi[s, T] * P(q_f | S, \_)$  ; termination step

$backpointer[q_F, T] \leftarrow \operatorname{argmax}_{s=1}^N viterbi[s, T] * P(q_f | S, \_)$  ; termination step

**return** the backtrace path by following backpointers to states back in time from  $backpointer[q_F, T]$

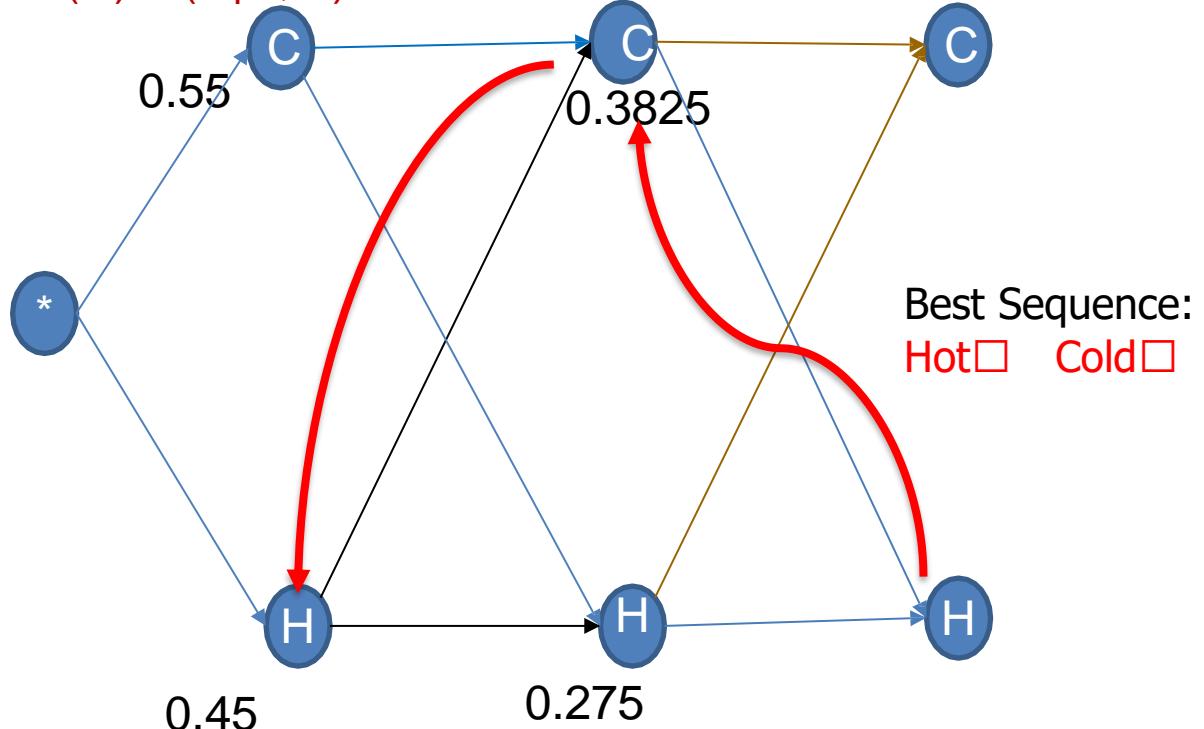
Source Credit : Speech and Language Processing - Jurafsky and Martin

# Maximum Entropy Markov Model

## Example -6 – Viterbi Algorithm – Termination through Back Trace

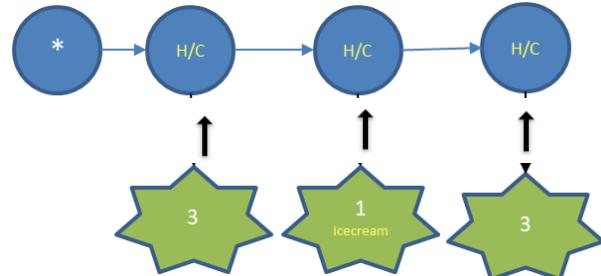
$$P(C) * P(C|C, 3) = 0.3825 * 0.2 = 0.0765$$

$$P(H) * P(C|H, 3) = 0.275 * 0.4 = 0.11$$



$$P(C) * P(H|C, 3) = 0.3825 * 0.8 = 0.306$$

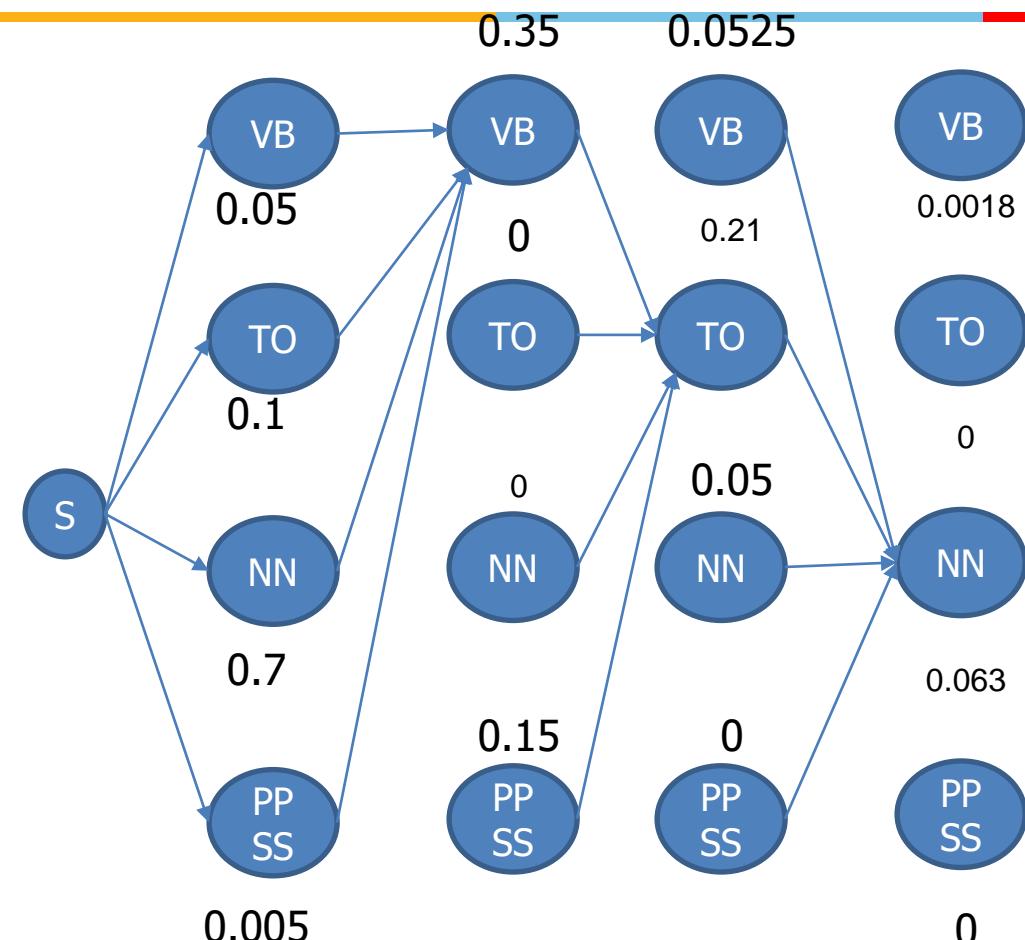
$$P(H) * P(H|H, 3) = 0.275 * 0.6 = 0.165$$



$X_1 = T_{i-1}$	$X_2 = w_i$	Label $Y = T_i$	$P(T_i / T_{i-1} W_i)$
Start	1	Hot	0.3
Start	1	Cold	
Start	3	Hot	0.45
Start	3	Cold	
.....	....	...	.....
Hot	1	Hot	0.15
Hot	1	Cold	
Hot	3	Hot	0.6
Hot	3	Cold	
.....	....	....	.....
Cold	1	Hot	0.5
Cold	1	Cold	
Cold	3	Hot	0.8
Cold	3	Cold	
....	....	....	....

# Maximum Entropy Markov Model

## Example -7 – Viterbi Algorithm

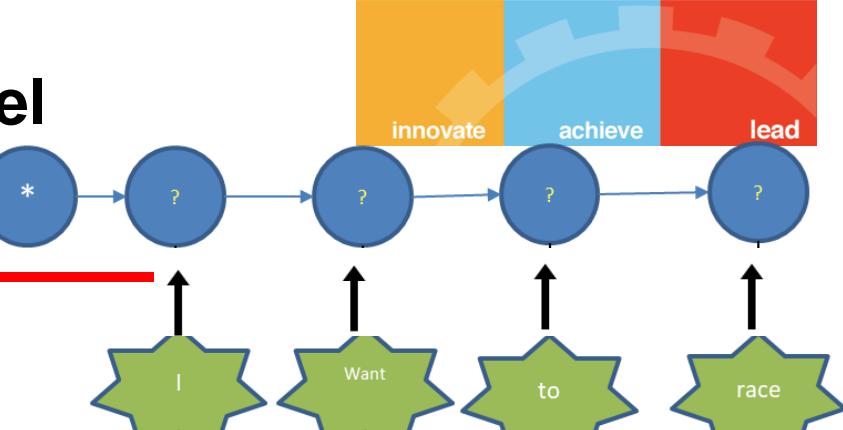


$$P(VB) * P(VB | VB, want) = 0.05 * 0.3 = 0.015$$

$$P(TO) * P(VB | TO, want) = 0.1 * 0.5 = 0.05$$

$$P(NN) * P(VB | NN, want) = 0.7 * 0.5 = 0.35$$

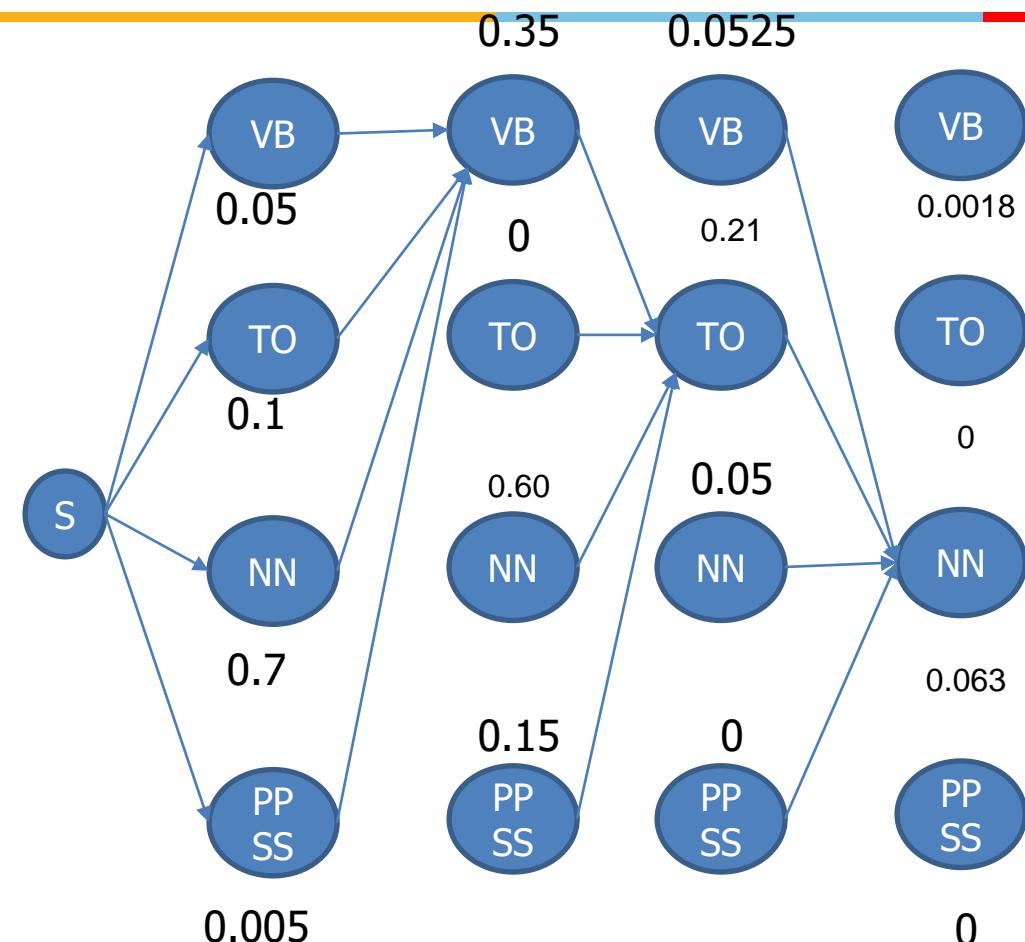
$$P(PPSS) * P(VB | PPSS, want) = 0.005 * 0.5 = 0.0025$$



$X_1 = T_{i-1}$	$X_2 = w_i$	Label $Y = T_i$	$P(T_i / T_{i-1} W_i)$
Start	I	NN	0.7
Start	I	TO	0.1
Start	I	VB	0.05
Start	I	PPSS	0.005
Start	The	NN	0.01
....	....	...	....
NN	want	NN	0.15
NN	want	VB	0.5
VB	want	VB	0.3
VB	to	TO	0.6
VB	to	NN	0.2
....	....	....	....
TO	race	NN	0.3
TO	race	VB	0.7
TO	race	TO	0
....	....	....	....
....	....	....	....

# Maximum Entropy Markov Model

## Example -7 – Viterbi Algorithm

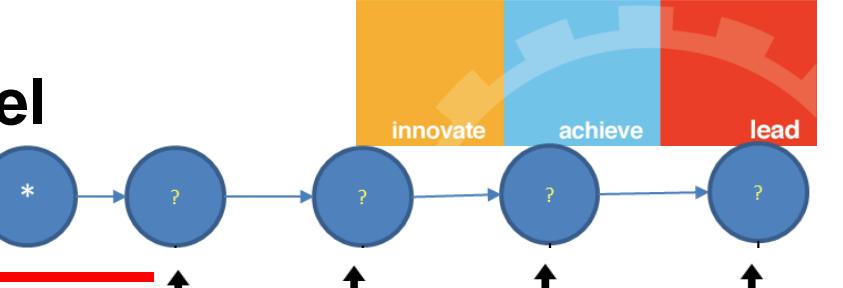


$$P(VB) * P(TO | VB, to) = 0.35 * 0.15 = 0.0525$$

$$P(TO) * P(TO | TO, to) = 0 * 0 = 0$$

$$P(NN) * P(TO | NN, to) = 0.6 * 0 = 0$$

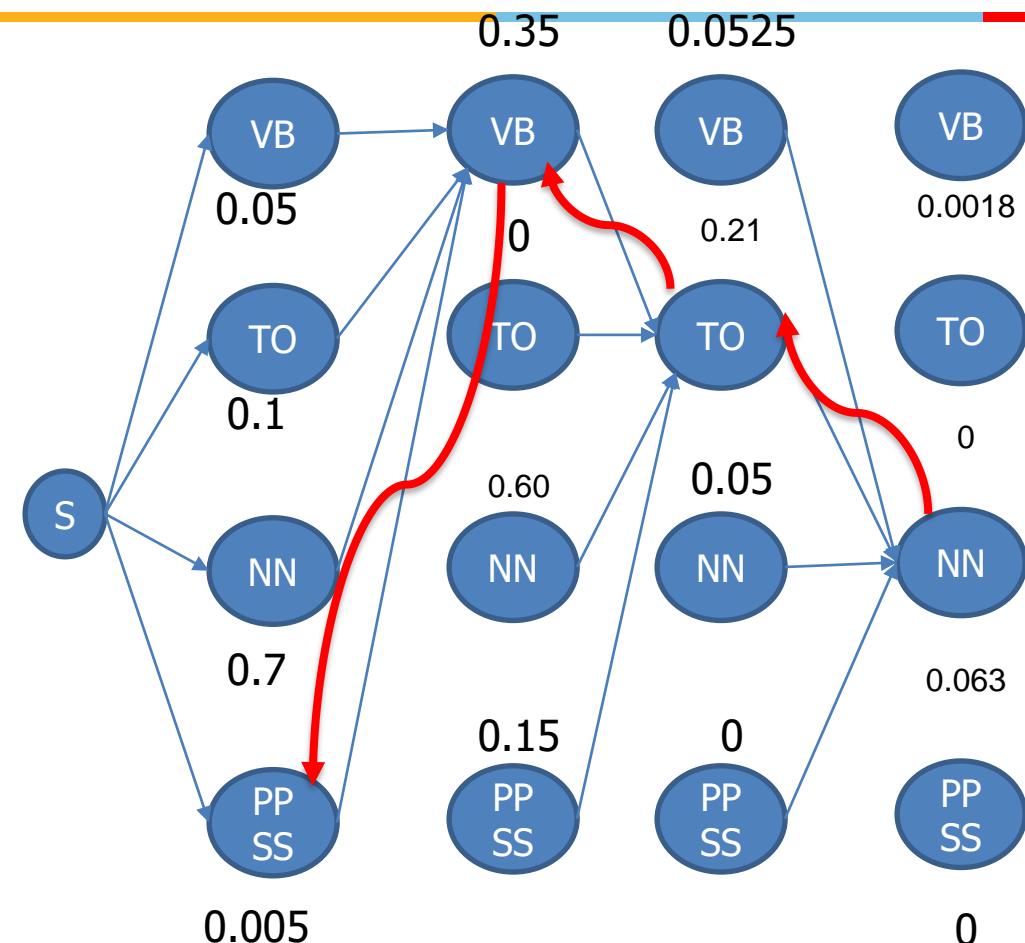
$$P(PPSS) * P(TO | PPSS, to) = 0.15 * 0 = 0$$



$X_1 = T_{i-1}$	$X_2 = w_i$	Label $Y = T_i$	$P(T_i / T_{i-1} W_i)$
Start	I	NN	0.7
Start	I	TO	0.1
Start	I	VB	0.05
Start	I	PPSS	0.005
Start	The	NN	0.01
....	....	...	....
NN	want	NN	0.15
NN	want	VB	0.5
VB	want	VB	0.3
VB	to	TO	0.6
VB	to	NN	0.2
....	....	....	....
TO	race	NN	0.3
TO	race	VB	0.7
VB	race	TO	0
....	....	....	....
....	....	....	....

# Maximum Entropy Markov Model

## Example -7 – Viterbi Algorithm

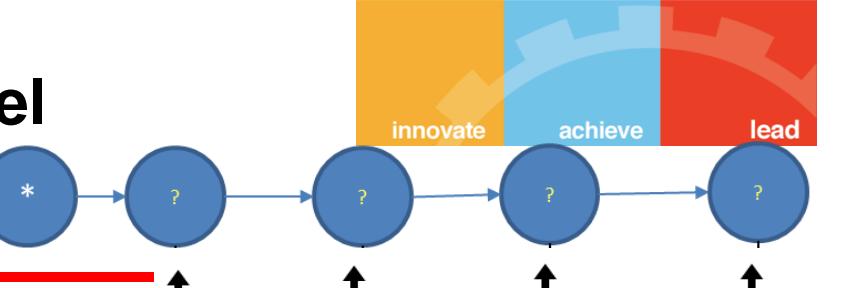


$$P(VB) * P(NN | VB, race) = 0.0525 * 0.1 = 0.00525$$

$$P(TO) * P(NN | TO, race) = 0.21 * 0.3 = 0.063$$

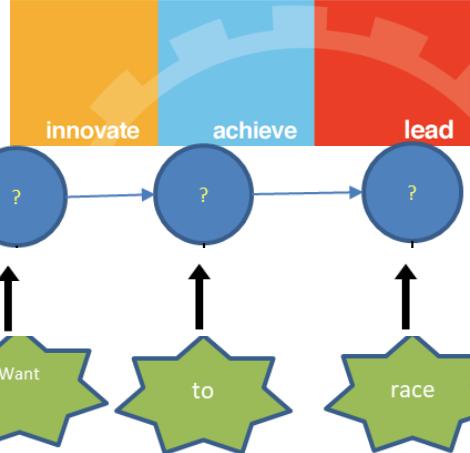
$$P(NN) * P(NN | NN, race) = 0.05 * 0.005 = 0.00025$$

$$P(PPSS) * P(NN | PPSS, race) = 0 * 0 = 0$$



$X_1 = T_{i-1}$	$X_2 = w_i$	Label $Y = T_i$	$P(T_i / T_{i-1} W_i)$
Start	I	NN	0.7
Start	I	TO	0.1
Start	I	VB	0.05
Start	I	PPSS	0.005
Start	The	NN	0.01
....	....	...	....
NN	want	NN	0.15
NN	want	VB	0.5
VB	want	VB	0.3
VB	to	TO	0.6
VB	to	NN	0.2
....	....	....	....
TO	race	NN	0.3
TO	race	VB	0.7
VB	race	TO	0
....	....	....	....
....	....	....	....

# Maximum Entropy Markov Model



## Learning the $P(T_i | T_{i-1} W_i)$ : Posterior Tags

- Training  
Multinomial Logistic Regression – MaxEnt

- Features
  - Engineer Features' design
  - “Secretariat is expected to **race** tomorrow”

NNP      VBZ      VBN      TO      VB      RB

		f1	f2	f3	f4	f5	f6
VB	f	0	1	0	1	1	0
VB	w		.8		.01	.1	
NN	f	1	0	0	0	0	1
NN	w	.8					-1.3

Figure 6.19 Some sample feature values and weights for tagging the word *race* in (6.81).

$$f_1(c, x) = \begin{cases} 1 & \text{if } word_i = "race" \& c = \text{NN} \\ 0 & \text{otherwise} \end{cases}$$

$$f_2(c, x) = \begin{cases} 1 & \text{if } t_{i-1} = \text{TO} \& c = \text{VB} \\ 0 & \text{otherwise} \end{cases}$$

$$f_3(c, x) = \begin{cases} 1 & \text{if } \text{suffix}(word_i) = "ing" \& c = \text{VBG} \\ 0 & \text{otherwise} \end{cases}$$

$$f_4(c, x) = \begin{cases} 1 & \text{if } \text{is\_lower\_case}(word_i) \& c = \text{VB} \\ 0 & \text{otherwise} \end{cases}$$

$$f_5(c, x) = \begin{cases} 1 & \text{if } word_i = "race" \& c = \text{VB} \\ 0 & \text{otherwise} \end{cases}$$

$$f_6(c, x) = \begin{cases} 1 & \text{if } t_{i-1} = \text{TO} \& c = \text{NN} \\ 0 & \text{otherwise} \end{cases}$$

$$f_{\text{example}}(x^{(*)}, t, y_i) := \begin{cases} 1 & x^{(t-1)} = "the" \wedge y_i = \text{NN} \\ 0 & \text{otherwise} \end{cases}$$

Source Credit : Speech and Language Processing - Jurafsky and Martin

Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	<i>and, but, or</i>	SYM	symbol	+,%,&
CD	cardinal number	<i>one, two, three</i>	TO	“to”	<i>to</i>
DT	determiner	<i>a, the</i>	UH	interjection	<i>ah, oops</i>
EX	existential ‘there’	<i>there</i>	VB	verb, base form	<i>eat</i>
FW	foreign word	<i>mea culpa</i>	VBD	verb, past tense	<i>ate</i>
IN	preposition/sub-conj	<i>of, in, by</i>	VBG	verb, gerund	<i>eating</i>
JJ	adjective	<i>yellow</i>	VBN	verb, past participle	<i>eaten</i>
JJR	adj., comparative	<i>bigger</i>	VBP	verb, non-3sg pres	<i>eat</i>
JJS	adj., superlative	<i>wildest</i>	VBZ	verb, 3sg pres	<i>eats</i>
LS	list item marker	<i>1, 2, One</i>	WDT	wh-determiner	<i>which, that</i>
MD	modal	<i>can, should</i>	WP	wh-pronoun	<i>what, who</i>
NN	noun, sing. or mass	<i>llama</i>	WP\$	possessive wh-	<i>whose</i>
NNS	noun, plural	<i>llamas</i>	WRB	wh-adverb	<i>how, where</i>
NNP	proper noun, singular	<i>IBM</i>	\$	dollar sign	\$
NNPS	proper noun, plural	<i>Carolinas</i>	#	pound sign	#
PDT	predeterminer	<i>all, both</i>	“	left quote	‘ or “
POS	possessive ending	<i>'s</i>	”	right quote	’ or ”
PRP	personal pronoun	<i>I, you, he</i>	(	left parenthesis	[, (, {, <
PRP\$	possessive pronoun	<i>your, one's</i>	)	right parenthesis	], ), }, >
RB	adverb	<i>quickly, never</i>	,	comma	,
RBR	adverb, comparative	<i>faster</i>	.	sentence-final punc	. ! ?
RBS	adverb, superlative	<i>fastest</i>	:	mid-sentence punc	: ; ... – -
RP	particle	<i>up, off</i>			

# Maximum Entropy Markov Model

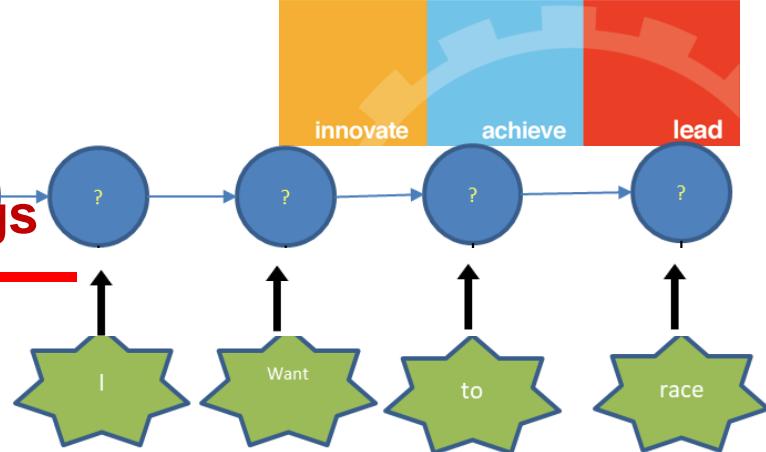
Learning the  $P(T_i | T_{i-1} W_i)$  : Posterior Tags

- Training  
Multinomial Logistic Regression – MaxEnt

- Features
  - Engineer Features' design
  - “Secretariat is expected to **race** tomorrow”  
NNP VBZ VBN TO VB RB

- Model : Log Linear / Logistic Regression Equation
- In Practice : Complex Features involving  $x$  is designed

$$f_{125}(c, x) = \begin{cases} 1 & \text{if } word_{i-1} = <\text{s}> \text{ & } \text{isupperfirst}(word_i) \text{ & } c = \text{NNP} \\ 0 & \text{otherwise} \end{cases}$$



$$P(NN|x) = \frac{e^{.8}e^{-1.3}}{e^{.8}e^{-1.3} + e^{.8}e^{.01}e^{.1}} = .20$$

$$P(VB|x) = \frac{e^{.8}e^{.01}e^{.1}}{e^{.8}e^{-1.3} + e^{.8}e^{.01}e^{.1}} = .80$$

Source Credit : Speech and Language Processing - Jurafsky and Martin

# Maximum Entropy Markov Model

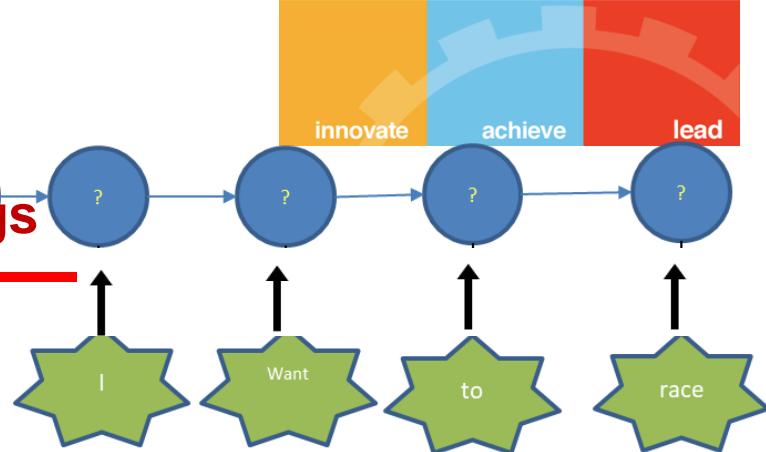
Learning the  $P(T_i | T_{i-1} W_i)$  : Posterior Tags

- Training  
Multinomial Logistic Regression – MaxEnt

- Features
    - Engineer Features' design
    - “Secretariat is expected to **race** tomorrow”
- |     |     |     |    |    |    |
|-----|-----|-----|----|----|----|
| NNP | VBZ | VBN | TO | VB | RB |
|-----|-----|-----|----|----|----|

- Model : Log Linear / Logistic Regression Equation

$$p(c|x) = \frac{\exp\left(\sum_{i=0}^N w_{ci} f_i(c, x)\right)}{\sum_{c' \in C} \exp\left(\sum_{i=0}^N w_{c'i} f_i(c', x)\right)}$$



$$P(NN|x) = \frac{e^8 e^{-1.3}}{e^{-8} e^{-1.3} + e^8 e^{.01} e^{-.1}} = .20$$

$$P(VB|x) = \frac{e^{-8} e^{.01} e^{-.1}}{e^{-8} e^{-1.3} + e^8 e^{.01} e^{-.1}} = .80$$

		f1	f2	f3	f4	f5	f6
VB	f	0	1	0	1	1	0
VB	w		.8		.01	.1	
NN	f	1	0	0	0	0	1
NN	w	.8					-1.3

Figure 6.19 Some sample feature values and weights for tagging the word *race* in (6.81).

In boolean logistic regression, classification involves building one linear expression which separates the observations in the class from the observations not in the class. Classification in MaxEnt, by contrast, involves building a separate linear expression for each of C classes

Source Credit : Speech and Language Processing - Jurafsky and Martin

# Maximum Entropy Markov Model

Learning the  $W_c$  : Weights not words

- Training  
Multinomial Logistic Regression – MaxEnt

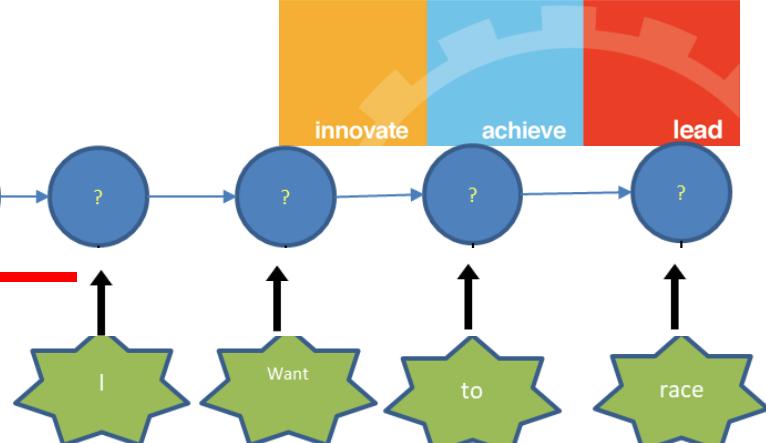
- Features
  - Engineer Features' design
  - “Secretariat is expected to **race** tomorrow”

NNP      VBZ      VBN      TO      VB      RB

- Model : Log Linear / Logistic Regression Equation
- Learn the weights (with or without Regularization)
  - **Gradient Descent**

$$\hat{w} = \operatorname{argmax}_w \sum_i \log P(y^{(i)} | x^{(i)}) - \sum_{j=1}^N \frac{w_j^2}{2\sigma_j^2}$$

$$\hat{w} = \operatorname{argmax}_w \sum_i \log P(y^{(i)} | x^{(i)}) - \alpha \sum_{j=1}^N w_j^2$$



$X_1 = T_{i-1}$	$X_2 = w_i$	Label $Y = T_i$	$P(T_i / T_{i-1} W_i)$
Start	I	NN	0.7
Start	I	TO	0.1
Start	I	VB	0.05
Start	I	PPSS	0.005
Start	The	NN	0.01
.....	....	...	.....
NN	want	NN	0.15
NN	want	VB	0.5
VB	want	VB	0.3
VB	to	TO	0.6
VB	to	NN	0.2
.....	.....	.....	.....
TO	race	NN	0.3
TO	race	VB	0.7
VB	race	TO	0
....	....	....	....
....	....	....	....

# Maximum entropy classification

---

From: LeChuck@yahoo.com

Har har!

Allow mes to introduce myself. My name is Lechuck ,and I got your email from the interwebs mail directory. I threw a dart at the directory and hit your name. You seam like a good lad based on your name, so I here I am writing this email.

I live out here in this faraway island, and I have some moneys (\$122K USD, to be exact) that I need to send overseas. Could you do mes a favor and help me with my moneys transfer?

- 1) Provide mes a bank account where this money would be transferred to.
- 2) Send me a picture of yourslefs so I know who to look for and thank when I sail to the US. Click heres to my Facebook  
[[www.lechuck.myfacebook.com/fake.link/give\\_me\\_money](http://www.lechuck.myfacebook.com/fake.link/give_me_money)] and post me your pic.

Monkeys bananas money rich

As reward, I are willing to offer you 15% of the moneys as compensation for effort input after the successful transfer of this fund to your designate account overseas.  
please feel free to contact ,me via this email address  
lechuck@yahoo.com

# Features and weights

---

F1=Email contains spelling/grammatical errors

F2=Email asks for money to be transferred

F3=Email mentions account holder's name

## Weights for spam

W1 =Email contains spelling/grammatical errors): 0.5

W2=Email asks for money to be transferred): 0.2

W3=Email mentions account holder's name): -0.5

## Weights for not spam

W1=(Email contains spelling/grammatical errors): -0.2

W2=(Email asks for money to be transferred): 0

W3 =(Email mentions account holder's name): 1

---

# Function

$$f(x) = \begin{cases} 1, & \text{if the feature is present in } x \\ 0, & \text{otherwise} \end{cases}$$

Formula

$$Score_d(x) = \frac{\exp(\sum_{i=1}^N w_i(d)*f_i(x))}{\sum_d \exp(\sum_{i=1}^N w_i(d)*f_i(x))}$$

---

F1=Email contains spelling/grammatical errors -Yes

F2=Email asks for money to be transferred-Yes

F3=Email mentions account holder's name-No

## Spam score

$$Score_{spam}(email_{LeChuck}) = \frac{\exp(\sum_{i=1}^N w_i(spam)*f_i(email_{LeChuck}))}{\sum_d \exp(\sum_{i=1}^N w_i(d)*f_i(email_{LeChuck}))} =$$

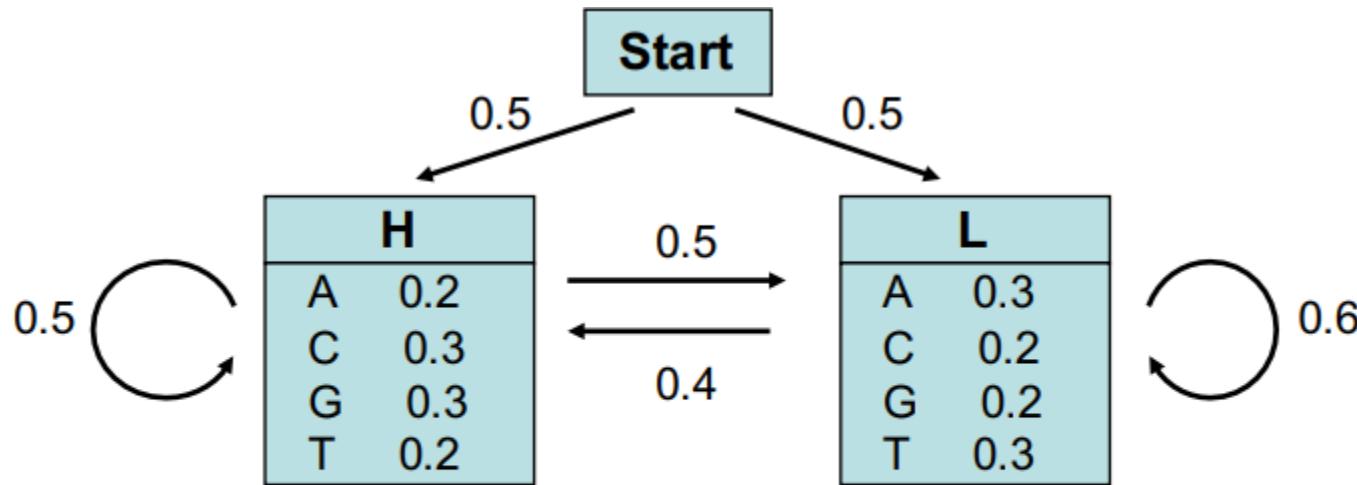
$$\frac{\exp(0.5*1+0.2*1-0.5*0)}{\exp(0.5*1+0.2*1-0.5*0)+\exp(-0.2*1+0*1+1*0)} = 0.71$$

## Not spam score

$$Score_{notspam}(email_{LeChuck}) = \frac{\exp(\sum_{i=1}^N w_i(notspam)*f_i(email_{LeChuck}))}{\sum_d \exp(\sum_{i=1}^N w_i(d)*f_i(email_{LeChuck}))} =$$

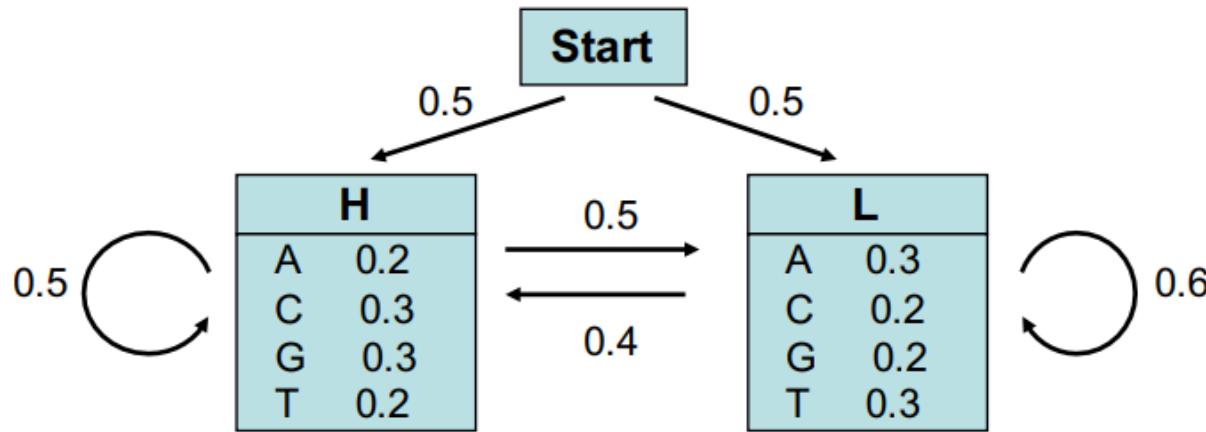
$$\frac{\exp(-0.2*1+0*1+1*0)}{\exp(0.5*1+0.2*1-0.5*0)+\exp(-0.2*1+0*1+1*0)} = 0.29$$

# Forward algorithm-example



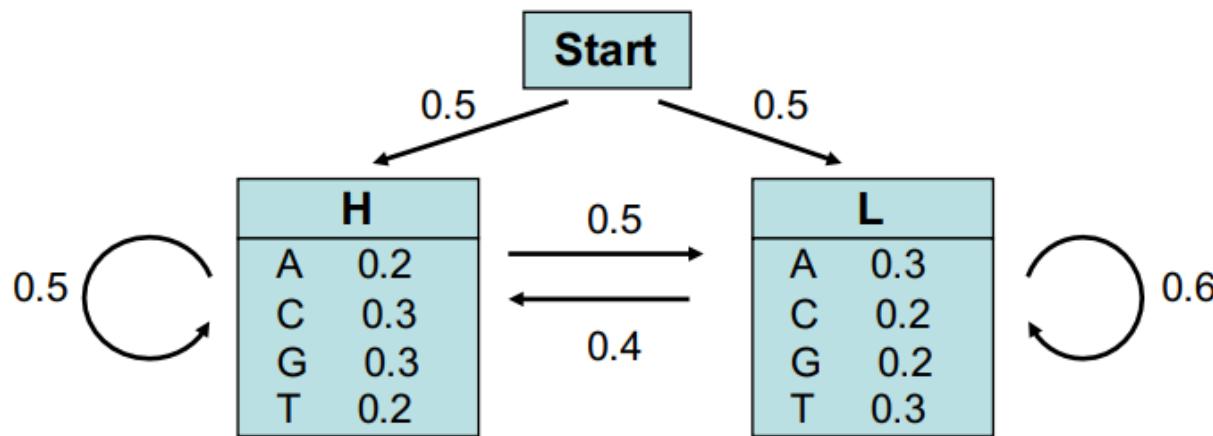
Consider now the sequence S= **GGCA**

	Start	G	G	C	A
H	0	$0.5 * 0.3 = 0.15$			
L	0	$0.5 * 0.2 = 0.1$			



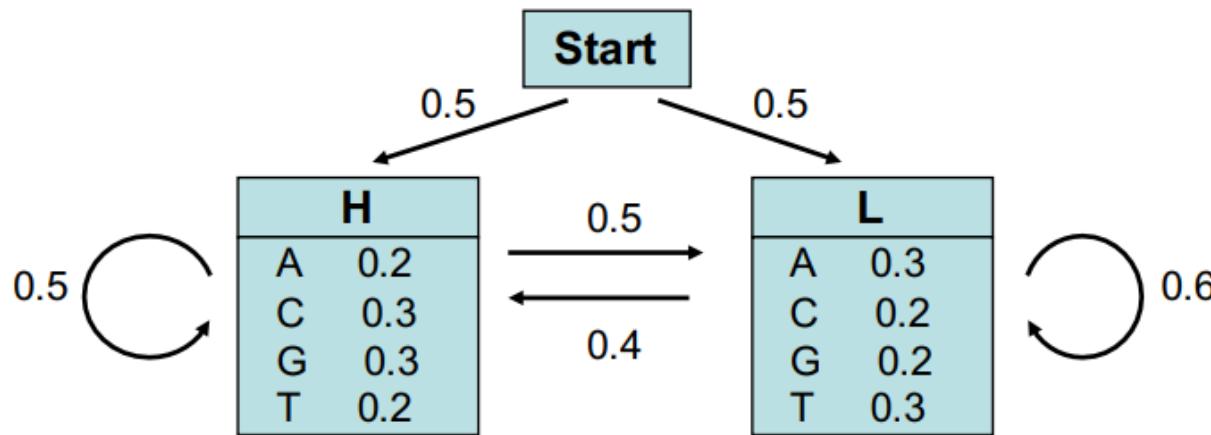
Consider now the sequence S= **GGCA**

	Start	G	G	C	A
H	0	$0.5 * 0.3 = 0.15$	$0.15 * 0.5 * 0.3 + 0.1 * 0.4 * 0.3 = 0.0345$		
L	0	$0.5 * 0.2 = 0.1$			



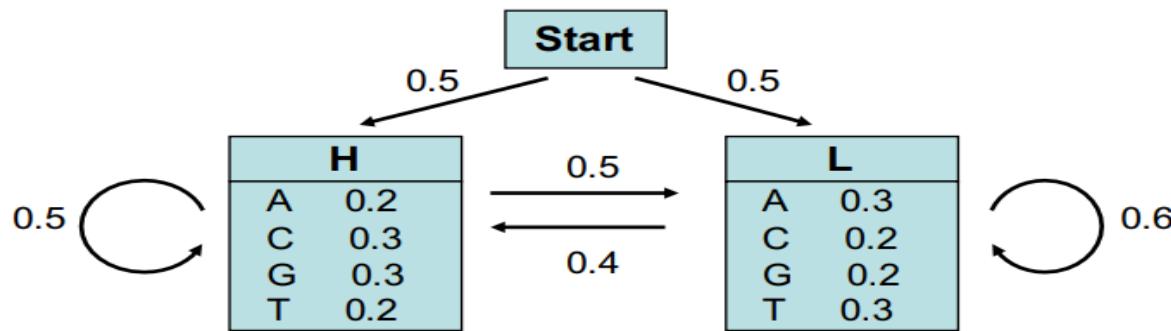
Consider now the sequence S= **GGCA**

	Start	G	G	C	A
H	0	$0.5 \cdot 0.3 = 0.15$	$0.15 \cdot 0.5 \cdot 0.3 + 0.1 \cdot 0.4 \cdot 0.3 = 0.0345$		
L	0	$0.5 \cdot 0.2 = 0.1$	$0.1 \cdot 0.6 \cdot 0.2 + 0.15 \cdot 0.5 \cdot 0.2 = 0.027$		



Consider now the sequence S= **GGCA**

	Start	G	G	C	A
H	0	$0.5 \cdot 0.3 = 0.15$	$0.15 \cdot 0.5 \cdot 0.3 + 0.1 \cdot 0.4 \cdot 0.3 = 0.0345$	$\dots + \dots$	
L	0	$0.5 \cdot 0.2 = 0.1$	$0.1 \cdot 0.6 \cdot 0.2 + 0.15 \cdot 0.5 \cdot 0.2 = 0.027$	$\dots + \dots$	



Consider now the sequence S= **GGCA**

	Start	G	G	C	A
H	0	$0.5 * 0.3 = 0.15$	$0.15 * 0.5 * 0.3 + 0.1 * 0.4 * 0.3 = 0.0345$	$\dots + \dots$	0.0013767
L	0	$0.5 * 0.2 = 0.1$	$0.1 * 0.6 * 0.2 + 0.15 * 0.5 * 0.2 = 0.027$	$\dots + \dots$	0.0024665

=> The probability that the sequence S was generated by the HMM model is thus  $P(S)=0.0038432$ .

$$\Sigma = 0.0038432$$



# Natural Language Processing

## DSECL ZG565



**BITS** Pilani  
Pilani Campus

Prof.Vijayalakshmi Anand

BITS-Pilani



**Session 5-Parsing  
Date – 9 May 2021  
Time – 9to 11am**

These slides are prepared by the instructor, with grateful acknowledgement of James Allen and many others who made their course materials freely available online.

# Session 5: Parsing

- 
- Grammars and Sentence Structure
  - What Makes a Good Grammar
  - Parsing
  - A Top-Down Parser
  - A Bottom-Up Chart Parser
  - Top-Down Chart Parsing
  - Finite State Models and Morphological Parsing.

# What is the structure of a sentence in natural language ?

---

- Sentence structure is hierarchical:
  - A sentence consists of words (I, eat, sushi, with, tuna)
    - ..which form phrases: “sushi with tuna”
- Sentence structure defines dependencies between words or phrases:



# How to compute the Sentence structure

---



To compute the syntactic structure ,must consider 2 things

- Grammar

A precise way to define and describe the structure of the sentence

- Parsing

The method of analyzing a sentence to determine its structure according to the grammar.

# Why NLP needs grammars ?

---

- Regular languages and part of speech refers to the way words are arranged together but cannot support easily: Constituency, Grammatical relations and Subcategorization and dependency relations
- They can be modelled by grammars.

# Applications -Machine translation



The output of current systems is often ungrammatical:

Daniel Tse, a spokesman for the Executive Yuan said the referendum demonstrated for democracy and human rights, the President on behalf of the people of two. 3 million people for the national space right, it cannot say on the referendum, the legitimacy of Taiwan's position full. (BBC Chinese news, translated by Google Chinese to E

**Correct translation requires grammatical knowledge:  
English)**

# Question Answering

---

**This requires grammatical knowledge.**

...: John persuaded/promised Mary to leave.

- Who left? ...

**and inference:** John managed/failed to leave. - Did John leave?

John and his parents visited Prague. They went to the castle.

- Was John in Prague?
- Has John been to the Czech Republic?
- Has John's dad ever seen a castle?

# Contd..

---

## Sentiment Analysis:

"I like Frozen"

"I do not like Frozen"

"I like frozen yogurt"

## Relation Extraction:

"Rome is the capital of Italy and the region of Lazio".

# Context free grammar

---

- Simple yet powerful formalism to describe the syntactic structure of natural languages
- Developed in the mid-1950s by Noam Chomsky
- Allows one to specify rules that state how a constituent can be segmented into smaller and smaller constituents, up to the level of individual words

# Context free grammar - Definition

---

**A CFG is a 4-tuple  $\langle N, \Sigma, R, S \rangle$**

**A set of nonterminals N**

(e.g.  $N = \{S, NP, VP, PP, \text{Noun}, \text{Verb}, \dots\}$ )

**A set of terminals  $\Sigma$**

(e.g.  $\Sigma = \{I, you, he, eat, drink, sushi, ball, \dots\}$ )

**A set of rules R**

$R \subseteq \{A \rightarrow \beta \text{ with left-hand-side (LHS)} \ A \in N$   
 $\text{and right-hand-side (RHS)} \ \beta \in (N \cup \Sigma)^*\}$

**A start symbol S (sentence)**

# Contd..

- Terminals  $\Sigma$ 
  - words - {sleeps, saw, man, woman, telescope, the, with, in}
- Non-Terminals -N
  - The constituents in a language . Such as noun phrases, verb phrases and sentences
  - {S, NP, VP, PP, DT, Vi, Vt, NN, IN}
- Rules
  - Rules are equations that consist of a single non-terminal on the left and any number of terminals and nonterminals on the right.

# Some sample rules

$R =$

grammar

S	$\rightarrow$	NP	VP
VP	$\rightarrow$	Vi	
VP	$\rightarrow$	Vt	NP
VP	$\rightarrow$	VP	PP
NP	$\rightarrow$	DT	NN
NP	$\rightarrow$	NP	PP
PP	$\rightarrow$	IN	NP

lexicons

Vi	$\rightarrow$	sleeps
Vt	$\rightarrow$	saw
NN	$\rightarrow$	man
NN	$\rightarrow$	woman
NN	$\rightarrow$	telescope
DT	$\rightarrow$	the
IN	$\rightarrow$	with
IN	$\rightarrow$	in

# Categories of Phrases

**Noun phrase (NP):** Noun acts as the head word. They start with an article or noun.

**Verb phrase (VP):** Verb acts as the head word. They start with an verb

**Adjective phrase (ADJP):** Adjective as the head word. They start with an adjective

**Adverb phrase (ADVP):** Adverb acts as the head word. They usually start with an adjective

**Prepositional phrase (PP):** Preposition as the head word. They start with an preposition.

# Verb phrase

---

English VPs consist of a head verb along with 0 or more following constituents which we'll call arguments.

*VP* → *Verb* disappear

*VP* → *Verb NP* prefer a morning flight

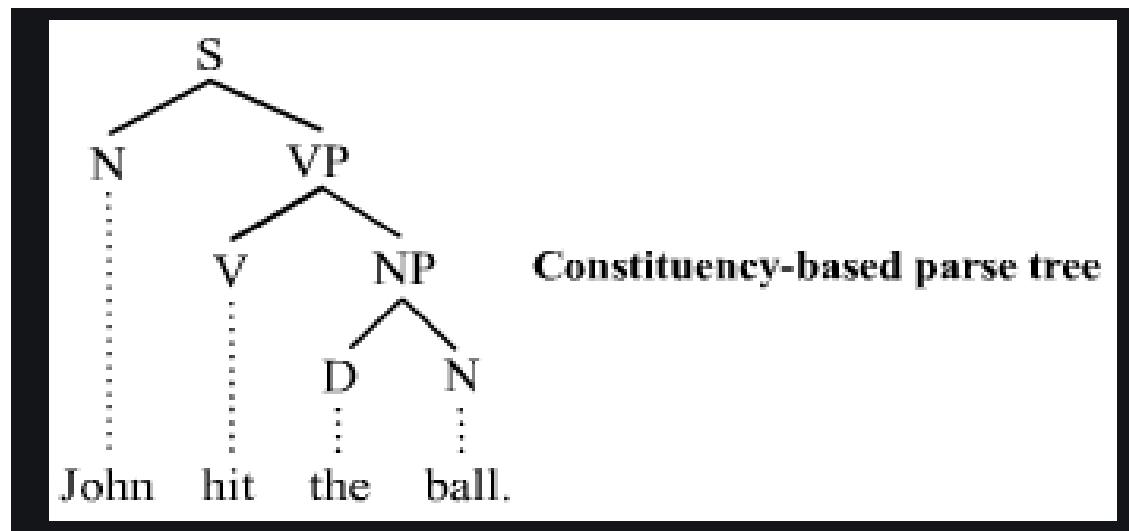
*VP* → *Verb NP PP* leave Boston in the morning

*VP* → *Verb PP* leaving on Thursday

# Parsing

Given a string of non-terminals and a CFG, determine if the string can be generated by the CFG.

- Also return a parse tree for the string
- Also return all possible parse trees for the string



# Why parsing ?

---

- Parsing adds information about sentence structure and constituents
- Allows us to see what constructions words enter into
  - eg, transitivity, passivization, argument structure for verbs
- Allows us to see how words function relative to each other
  - eg, what words can modify / be modified by other words

# Parsing as a search procedure

---

- Parsing as a special case of a search problem as defined in AI.
  1. Select the first state from the possibilities list (and remove it from the list).
  2. Generate the new states by trying every possible option from the selected state (there may be none if we are on a bad path).
  3. Add the states generated in step 2 to the possibilities list.

# Two types of Parsing

---

Two basic search strategies:

- **Top-down:** start at the root of the tree
- **Bottom-up:** start at the leaves

# A Simple Top-Down Parsing Algorithm

---

The algorithm starts with the initial state ((S) 1) and no backup states.

1. Take the first state off the possibilities list and call it  $C$ .  
IF the list is empty, THEN fails
2. IF  $C$  consists of an empty symbol list and the word position is at the end of the sentence, THEN succeeds
3. OTHERWISE, generate the next possible states.
  1. IF the first symbol of  $C$  is a lexical symbol, AND the next word in the sentence can be in that class,  
THEN
    - create a new state by removing the first symbol
    - updating the word position
    - add it to the possibilities list.
  2. OTHERWISE, IF the first symbol of  $C$  is a non-terminal  
THEN
    - generate a new state for each rule that can rewrite that non-terminal symbol
    - add them all to the possibilities list

# Toy example

People      laugh

1                2                3

These are positions

## Lexicon:

People - N, V  
Laugh - N, V

This indicate that both  
Noun and Verb is  
possible for the word  
“People”

- Grammar –  
 $S \rightarrow NP\ VP$   
 $NP \rightarrow DT\ N \mid N$   
 $VP \rightarrow V\ ADV \mid V$

# Contd..

State	Backup State	Action
1. ((S) 1)	-	-
	Position of input pointer	
2. ((NP VP)1)	-	-
3a. ((DT N VP)1)	((N VP) 1)	-
3b. ((N VP)1)	-	-
4. ((VP)2)	-	Consume "People"
5a. ((V ADV)2)	((V)2)	-
6. ((ADV)3)	((V)2)	Consume "laugh"
5b. ((V)2)	-	-
6. ((.)3)	-	Consume "laugh"

Termination Condition : All inputs over. No symbols remaining.

Note: Input symbols can be pushed back.

# Example2

*"**1 The 2 dogs 3 cried 4**"*

- |                                 |                           |
|---------------------------------|---------------------------|
| 1. $S \rightarrow NP\ VP$       | 4. $VP \rightarrow V$     |
| 2. $NP \rightarrow ART\ N$      | 5. $VP \rightarrow V\ NP$ |
| 3. $NP \rightarrow ART\ ADJ\ N$ |                           |

cried: V  
dogs: N, V  
the: ART

Lexicon

*"**1 The 2 old 3 man 4 cried 5**"*

- |                                 |                           |
|---------------------------------|---------------------------|
| 1. $S \rightarrow NP\ VP$       | 4. $VP \rightarrow V$     |
| 2. $NP \rightarrow ART\ N$      | 5. $VP \rightarrow V\ NP$ |
| 3. $NP \rightarrow ART\ ADJ\ N$ |                           |

cried: V  
old: ADJ, N  
man: N, V  
the: ART

# Example- “The dogs cried”



Step	Current State	Backup States	Comment
1.	((S) 1)		initial position
2.	((NP VP) 1)		rewrite S by rule 1
3.	((ART N VP) 1)	((ART ADJ N VP) 1)	rewrite NP by rules 2&3
4.	((N VP) 2)		match ART with <i>the</i>
5.	((VP) 3)	((ART ADJ N VP) 1)	match N with <i>dogs</i>
6.	((V) 3)	((ART ADJ N VP) 1)	rewrite VP by rules 4&5
7.	( )	((V NP) 3) ((ART ADJ N VP) 1)	the parse succeeds as V is matched to <i>cried</i>

## Example—“The old man cried”

<b>Step</b>	<b>Current State</b>	<b>Backup States</b>	<b>Comment</b>
1.	((S) 1)		initial position
2.	((NP VP) 1)		S rewritten to NP VP
3.	((ART N VP) 1)		NP rewritten by rules 2&3
4.	((N VP) 2)	((ART ADJ N VP) 1)	
5.	((VP) 3)	((ART ADJ N VP) 1)	
6.	((V) 3)	((ART ADJ N VP) 1)	VP rewritten by rules 4&5
7.	(( ) 4)	((V NP) 3) ((ART ADJ N VP) 1)	
8.	((V NP) 3)	((ART ADJ N VP) 1)	the first backup is chosen

Step	Current State	Backup States	Comment
9.	((NP) 4)		
10.	((ART N) 4)	((ART ADJ N VP) 1) ((ART ADJ N) 4) ((ART ADJ N VP) 1)	looking for ART fails
11.	((ART ADJ N) 4)	((ART ADJ N VP) 1)	fails again
12.	((ART ADJ N VP) 1)		exploring backup state saved in step 3
13.	((ADJ N VP) 2)		
14.	((N VP) 3)		
15.	((VP) 4)		
16.	((V) 4)	((V NP) 4)	
17.	(( ) 5)		success!

# Bottom up parsing

---

- Data-driven
- Looks at words in input string first, checks / assigns their category(ies), and tries to combine them into acceptable structures in the grammar
- Involves scanning the derivation so far for sub-strings which match the right-hand-side of grammar / production rules and using the rule that would show their derivation from the non-terminal symbol of that rule

# Sample grammar

$S \rightarrow NP\ VP$   
 $S \rightarrow Aux\ NP\ VP$   
 $S \rightarrow VP$   
 $NP \rightarrow Pronoun$   
 $NP \rightarrow Proper-Noun$   
 $NP \rightarrow Det\ Nominal$   
 $Nominal \rightarrow Noun$   
 $Nominal \rightarrow Nominal\ Noun$   
 $Nominal \rightarrow Nominal\ PP$   
 $VP \rightarrow Verb$   
 $VP \rightarrow Verb\ NP$   
 $VP \rightarrow Verb\ NP\ PP$   
 $VP \rightarrow Verb\ PP$   
 $VP \rightarrow VP\ PP$   
 $PP \rightarrow Preposition\ NP$

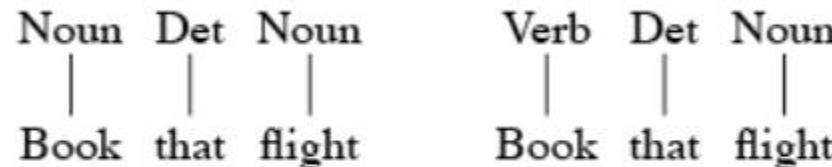
$Det \rightarrow that | this | a$   
 $Noun \rightarrow book | flight | meal | money$   
 $Verb \rightarrow book | include | prefer$   
 $Pronoun \rightarrow I | she | me$   
 $Proper-Noun \rightarrow Houston | TWA$   
 $Aux \rightarrow does$   
 $Preposition \rightarrow from | to | on | near | through$

# Example

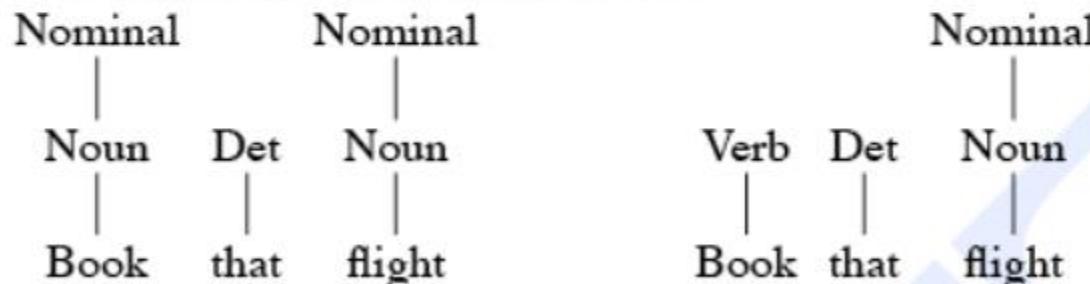
- Starts with input text

Book that flight

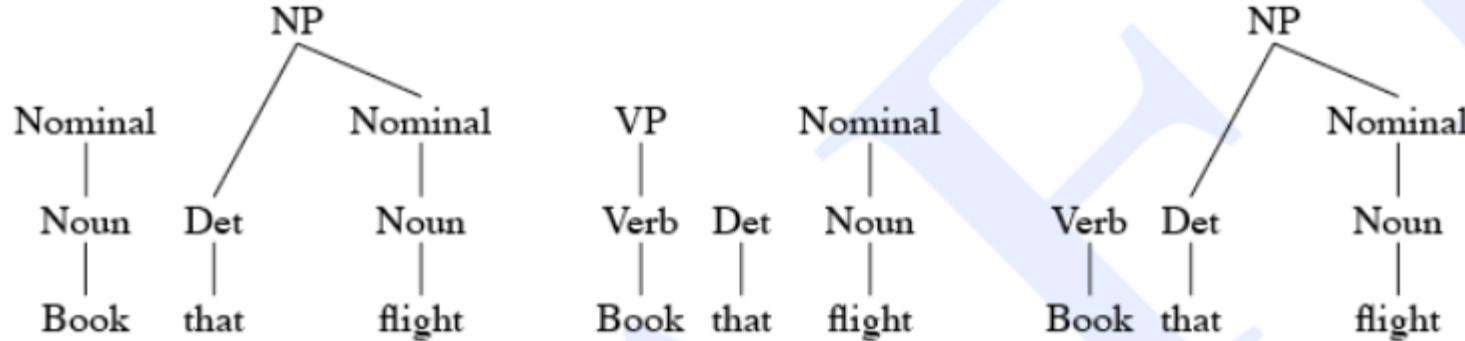
- derive the text from rules, in this case, two possible lexical rules



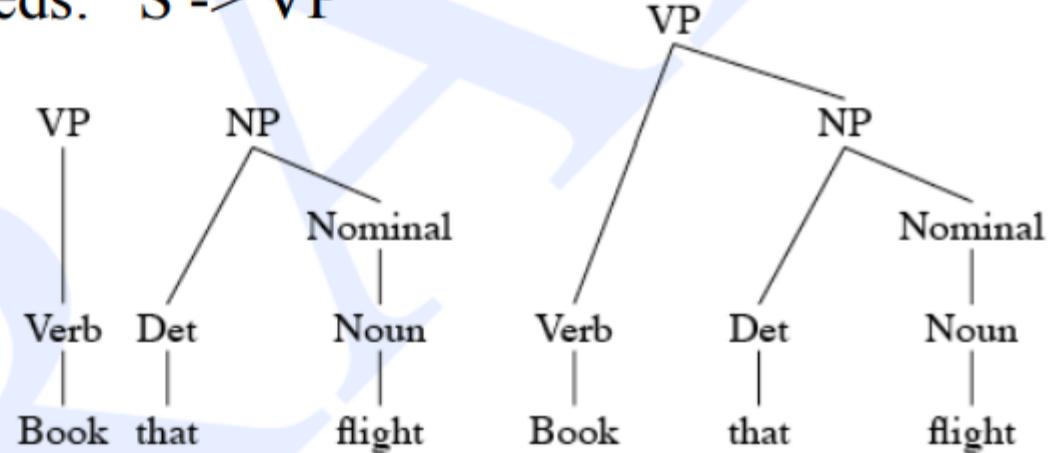
- Each of those can be derived from nonterminals



- Only the rightmost tree can continue the derivation here:



- And only one succeeds:  $S \rightarrow VP$



# Chart Parsing

- The *Chart* allows storing partial analyses, so that they can be shared.
- Data structures used by the algorithm:
  - **The Key:** the current constituent we are attempting to “match”
  - **An Active Arc:** a grammar rule that has a partially matched RHS
  - **The Agenda:** Keeps track of newly found unprocessed constituents
  - **The Chart:** Records processed constituents (non-terminals) that span substrings of the input

# Chart Parsing

1. S → NP VP	4. VP → V
2. NP → ART N	5. VP → V NP
3. NP → ART ADJ N	



- Assume you are parsing a sentence that starts with an ART.
- With this ART as the key, rules 2 and 3 are matched because they start with ART.
- To record this for analyzing the next key, you need to record that rules 2 and 3 could be continued at the point after the ART.
- You denote this fact by writing the rule with a dot (o), indicating what has been seen so far. Thus you record
  - 2'. NP → ART o ADJ N
  - 3'. NP → ART o N
- If the next input key is an ADJ, then rule 4 may be started, and the modified rule 2 may be extended to give
  - 2''. NP → ART ADJ o N

# The Chart Parsing Algorithm

---

## Bottom-UP Chart Parsing Algorithm

Do until there is no input left:

1. If the agenda is empty, get next word from the input, look up word categories, add to agenda (as constituent spanning two positions).
2. Select a constituent from the agenda: constituent  $C$  from  $p_1$  to  $p_2$ .
3. Insert  $C$  into the chart from position  $p_1$  to  $p_2$ .
4. For each rule in the grammar of form  $X \rightarrow C X_1 \dots X_n$ , add an active edge of form  $X \rightarrow C \circ X_1 \dots X_n$  from  $p_1$  to  $p_2$ .
5. Extend existing edges that are looking for a  $C$ .
  - (a) For any active edge of form  $X \rightarrow X_1 \dots \circ C X_n$  from  $p_0$  to  $p_1$ , add a new active edge  $X \rightarrow X_1 \dots C \circ X_n$  from  $p_0$  to  $p_2$ .
  - (b) For any active edge of form  $X \rightarrow X_1 \dots X_n \circ C$  from  $p_0$  to  $p_1$ , add a new (completed) constituent of type  $X$  from  $p_0$  to  $p_2$  to the agenda.

# Chart parsing example

The input: " $x = \text{The large can can hold the water}$ "

POS of Input Words:

- the: *ART*
- large: *ADJ*
- can: *N, AUX, V*
- hold: *N, V*
- water: *N, V*

- (1)  $S \rightarrow NP VP$
- (2)  $NP \rightarrow ART ADJ N$
- (3)  $NP \rightarrow ART N$
- (4)  $NP \rightarrow ADJ N$
- (5)  $VP \rightarrow AUX VP$
- (6)  $VP \rightarrow V NP$

# Chart parsing example

The input: “ $x = \text{The large can can hold the water}$ ”

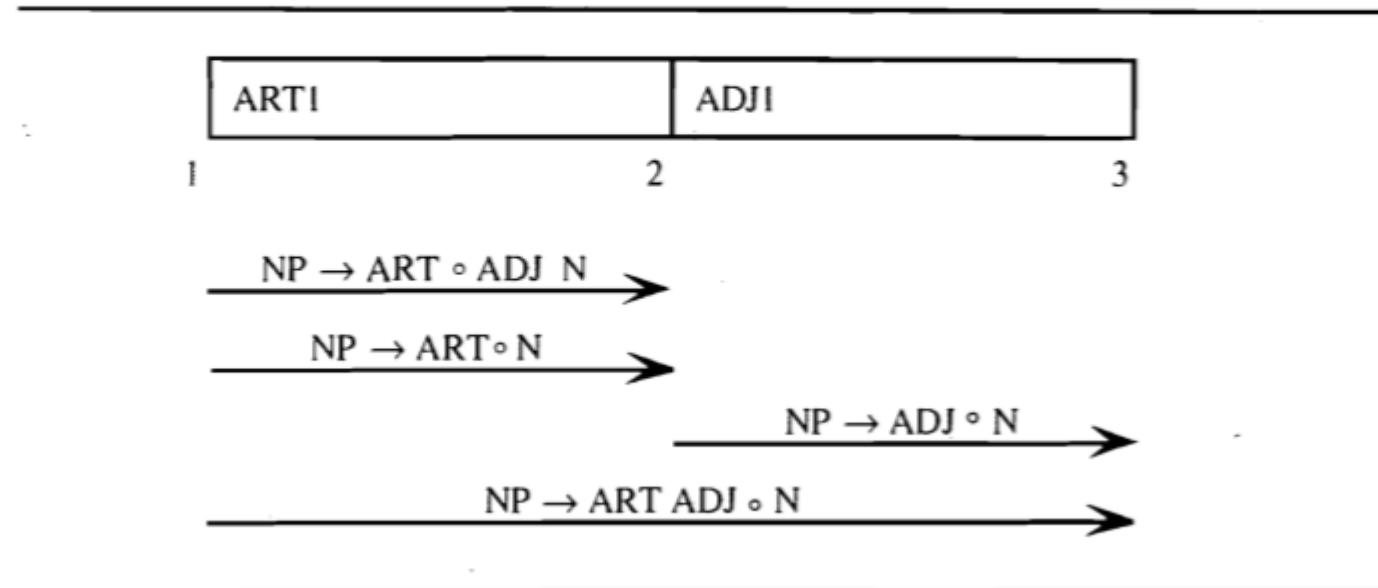


Figure 3.9 The chart after seeing an ADJ in position 2

# Chart parsing example

The input: “ $x = \text{The large can can hold the water}$ ”

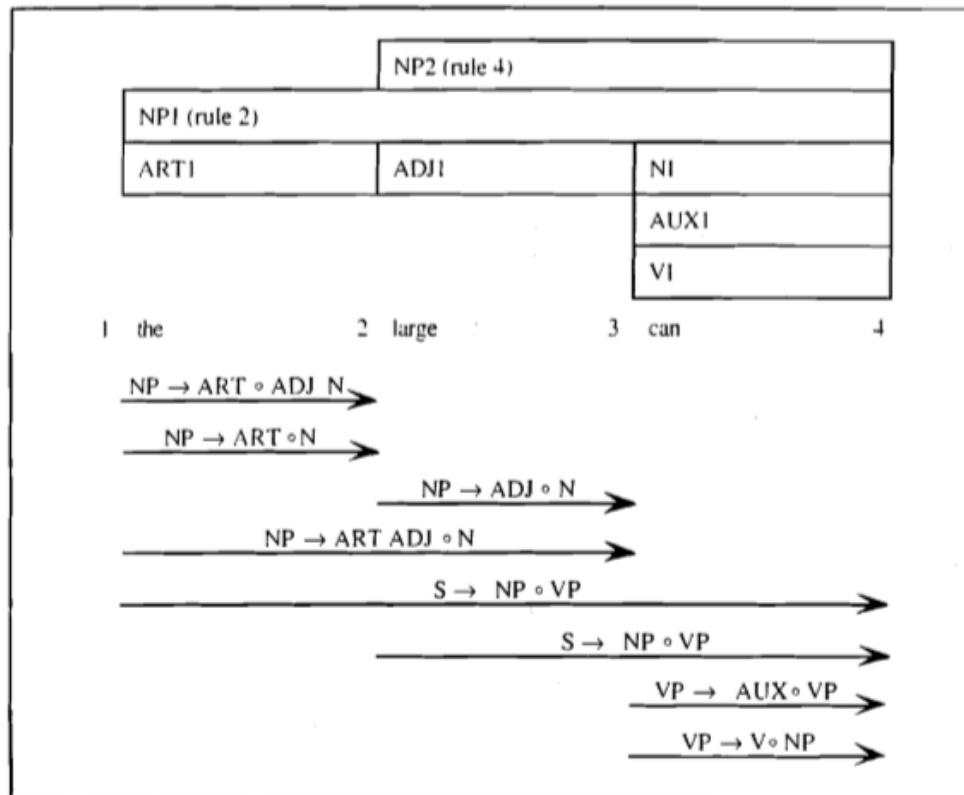


Figure 3.12 After parsing *the large can*

# Chart parsing example

The input: “***x = The large can can hold the water***”

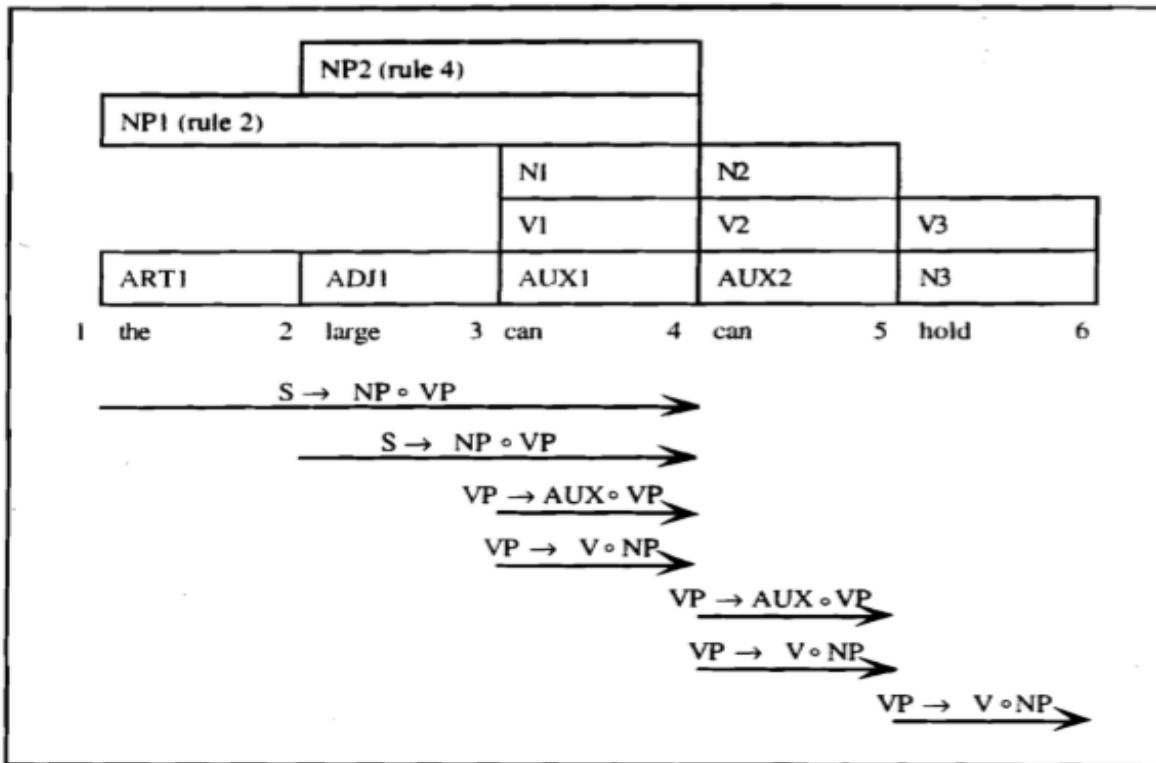


Figure 3.13 The chart after adding *hold*, omitting arcs generated for the first NP

# Chart parsing example

The input: “**x = The large can can hold the water**”

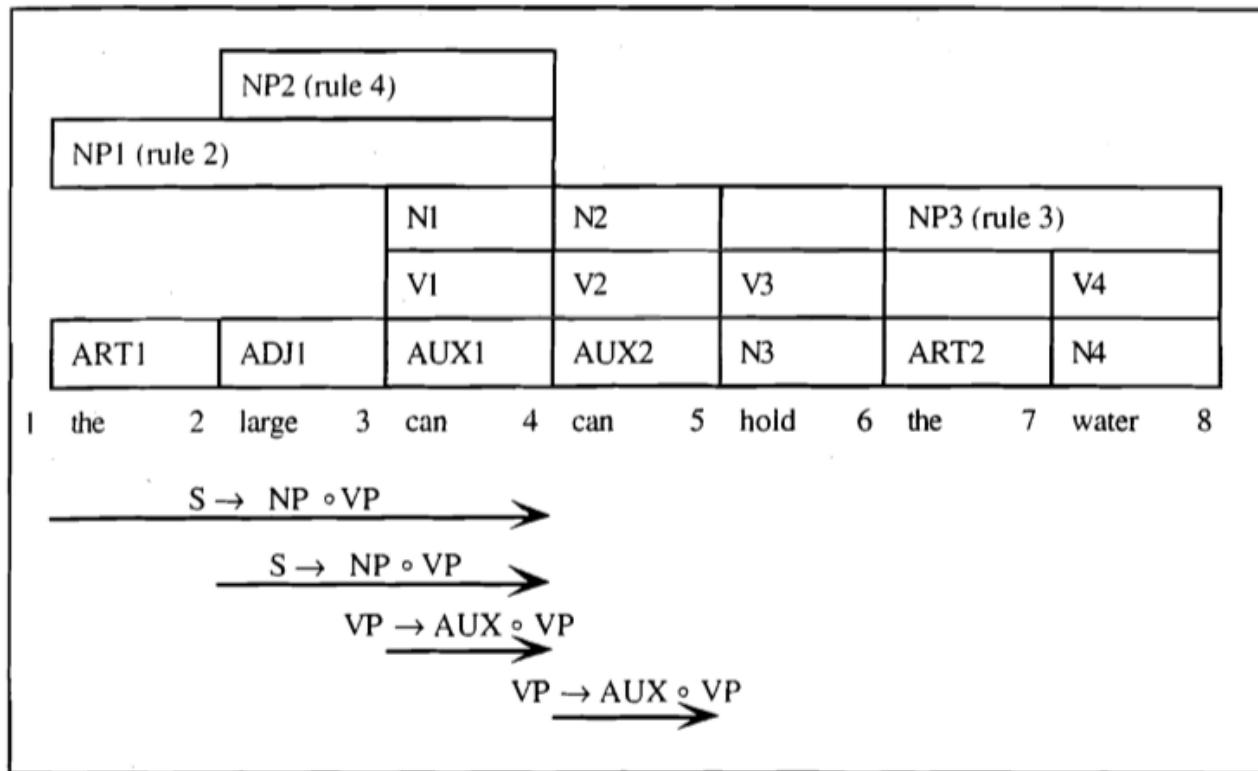


Figure 3.14 The chart after all the NPs are found, omitting all but the crucial active arcs

# Final Chart

The input: “ $x = \text{The large can can hold the water}$ ”

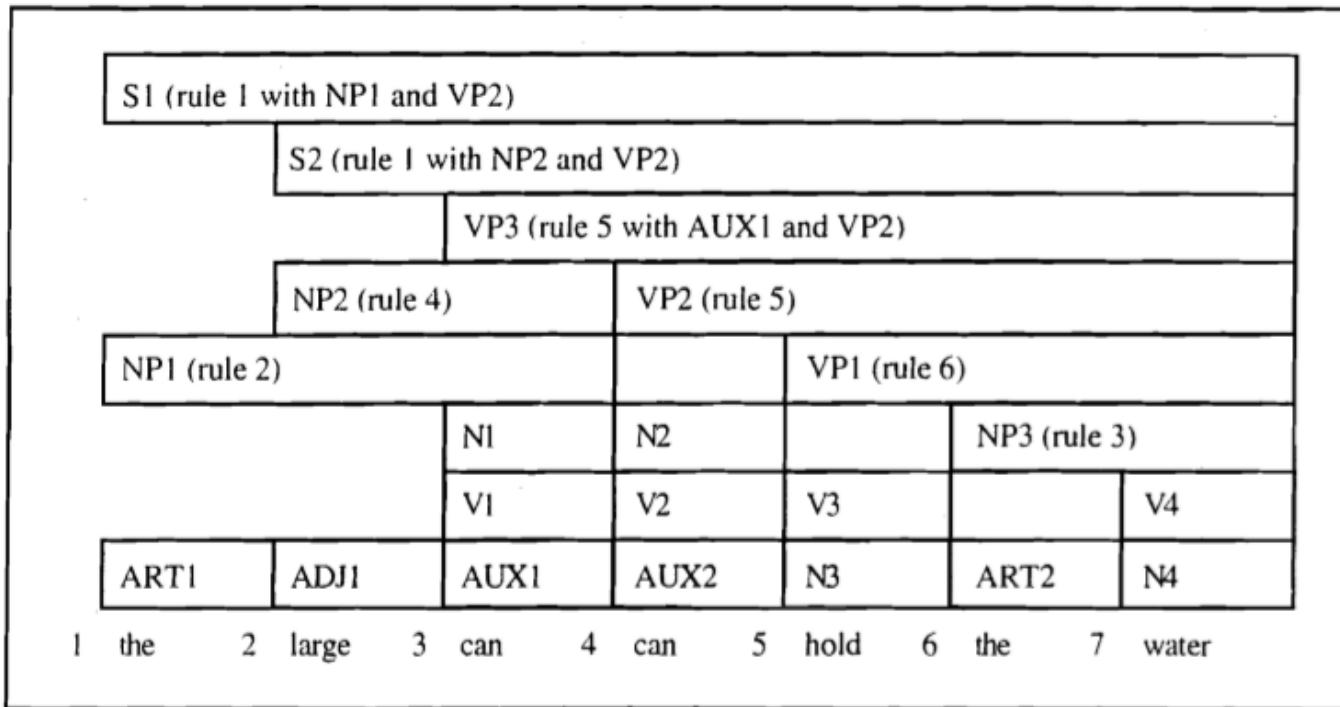


Figure 3.15 The final chart

# Example2

## Grammar and Lexicon

### Grammar:

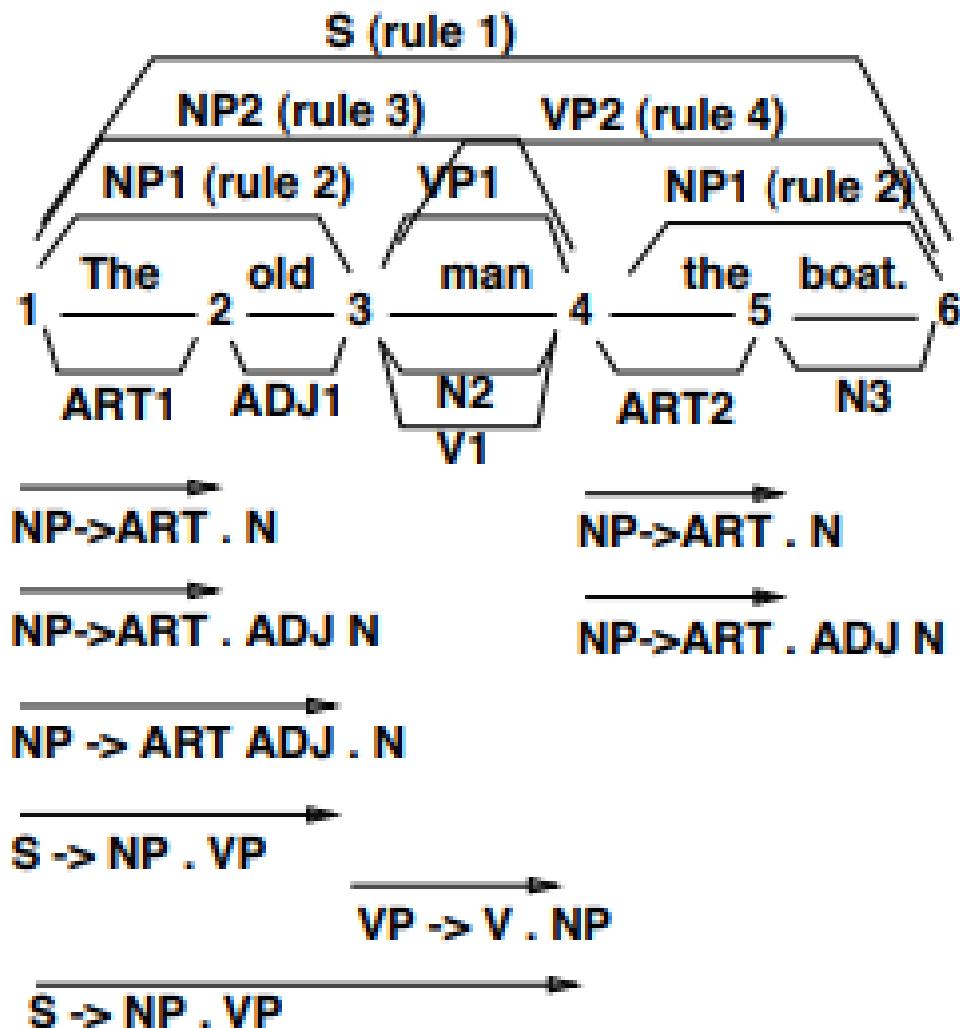
1.  $S \rightarrow NP\ VP$
2.  $NP \rightarrow ART\ N$
3.  $NP \rightarrow ART\ ADJ\ N$
4.  $VP \rightarrow V\ NP$

### Lexicon:

- |             |           |
|-------------|-----------|
| the: ART    | man: N, V |
| old: ADJ, N | boat: N   |

**Sentence:**  $_1\ The\ _2\ old\ _3\ man\ _4\ the\ _5\ boat\ _6$

# Contd..



# Top down chart parsing

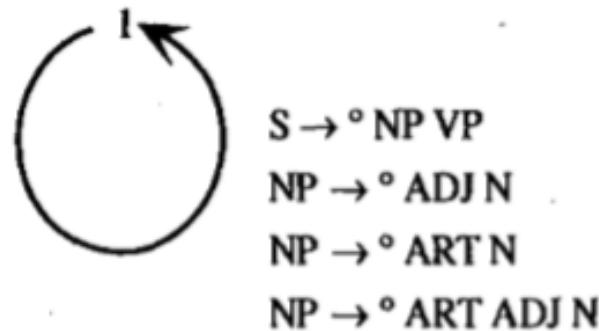
---

Initialization: For every rule in the grammar of form  $S \rightarrow X_1 \dots X_k$ , add an arc labeled  $S \rightarrow o | X_1 \dots X_k$  using the arc introduction algorithm.

Parsing: Do until there is no input left:

1. If the agenda is empty, look up the interpretations of the next word and add them to the agenda.
2. Select a constituent from the agenda (call it constituent C).
3. Using the arc extension algorithm, combine C with every active arc on the chart. Any new constituents are added to the agenda.
4. For any active arcs created in step 3, add them to the chart using the top-down arc introduction algorithm.

# Top down chart parsing example



**Figure 3.23** The initial chart

# Top down chart parsing example

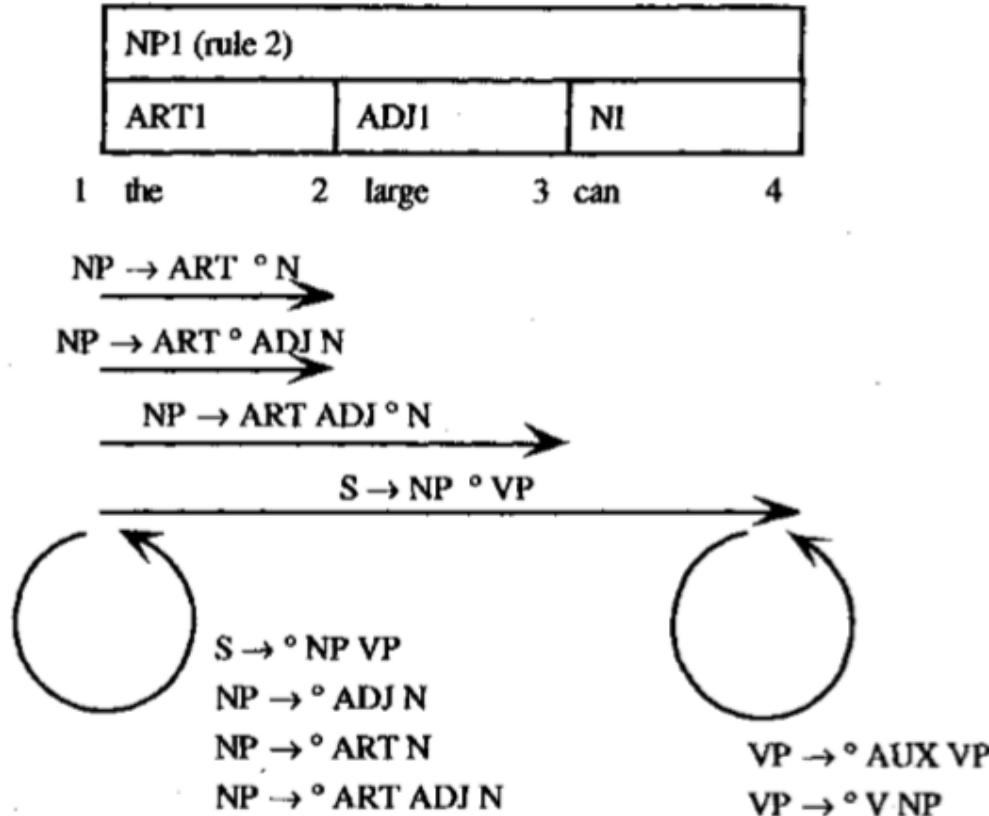


Figure 3.24 The chart after building the first NP

# Top down chart parsing example

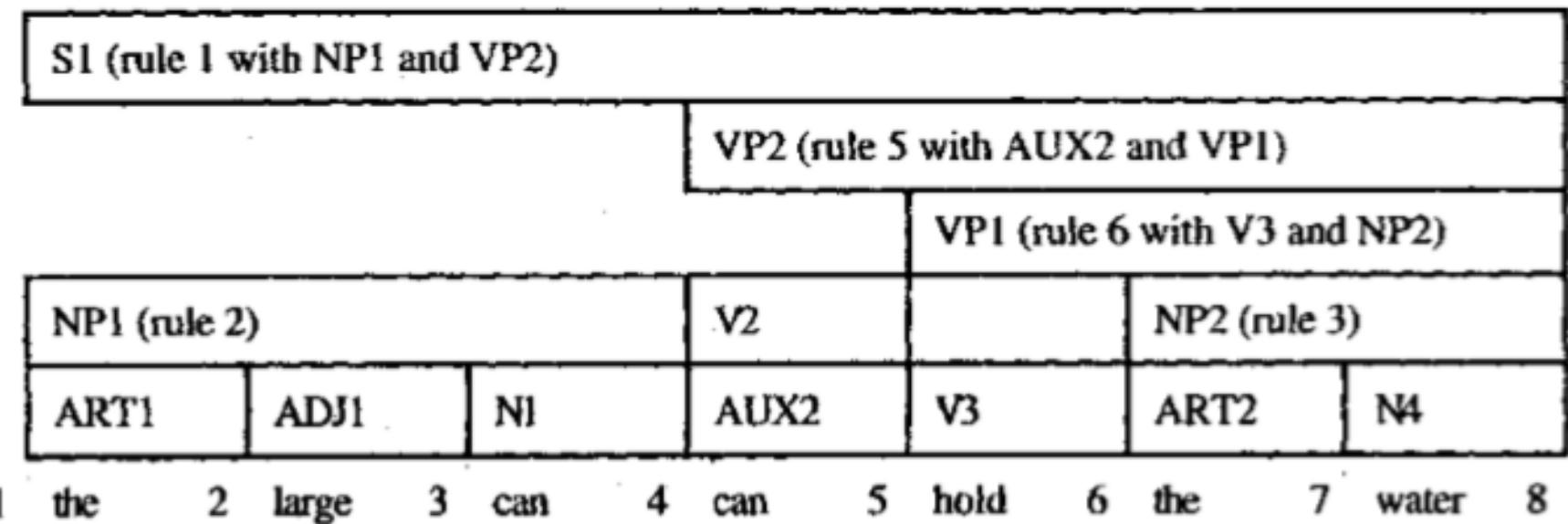


Figure 3.26 The final chart for the top-down filtering algorithm

# Morphological Parsing

---

- Not only are there a large number of words, but each word may combine with affixes to produce additional related words.
- One way to address this problem is to preprocess the input sentence into a sequence of morphemes.
- A word may consist of single morpheme, but often a word consists of a root form plus an affix. .

## Examples of morphological parsing:

– Separating words into stems/roots and affixes

e.g., input: cats      parse output: cat +s

– Labeling morphemes with category labels

e.g., input: cats      parse output: cat +N +PL

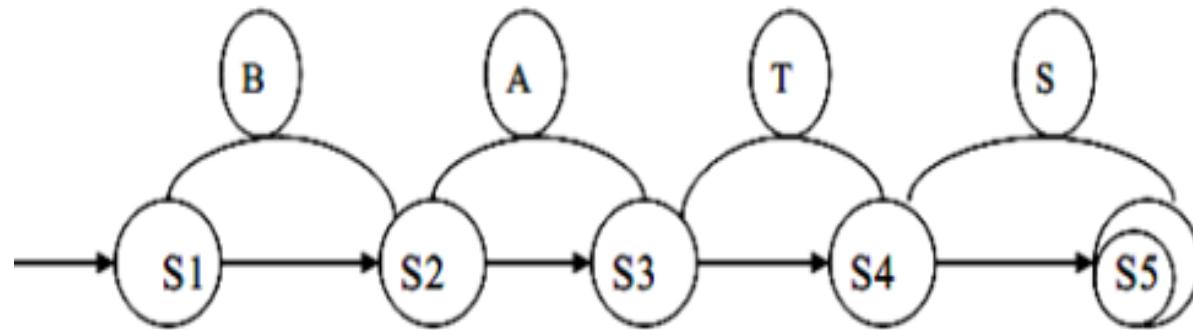
                        ate                            eat +V +PAST

# Components of Morphological parsing

---

- Lexicon
- Morphotactic
- Orthographic rules

# Finite state automation



# Finite State Transducers

- The simple story
  - Add another tape
  - Add extra symbols to the transitions
  - On one tape we read “*cats*”, on the other we write “*cat +N +PL*”, or the other way around.

# FSTs

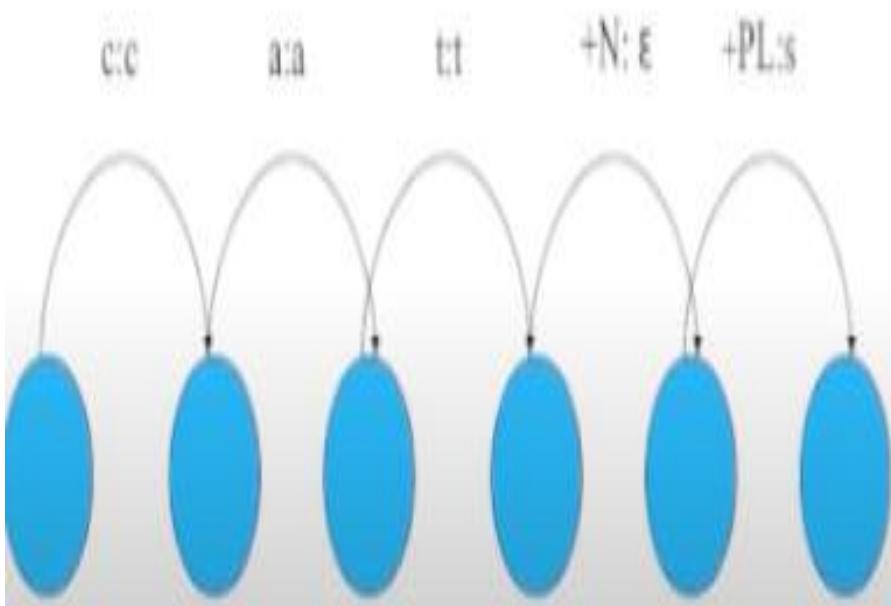
*Lexical*



*Surface*

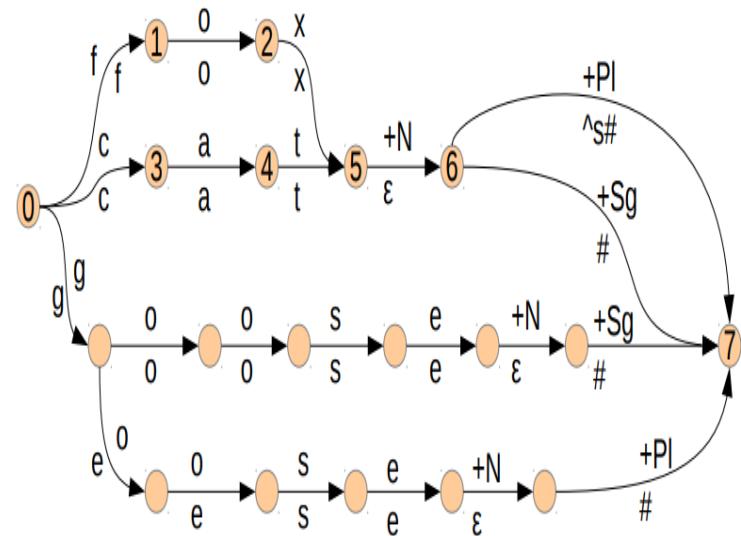


# Example



goose/geese: g:g o:e o:e s:s e:e

- Feasible pairs (e.g., o:e) vs. default pairs (g:g)



# Parsing/Generation vs. Recognition

---

- Recognition is usually not quite what we need.
  - Usually if we find some string in the language we need to find the structure in it (**parsing**)
  - Or we have some structure and we want to produce a surface form (**production/generation**)
- Example
  - From “cats” to “cat +N +PL” and back

→**Morphological analysis**

# Stemming in IR

- Run a stemmer on the documents to be indexed
- Run a stemmer on users queries
- Match
  - This is basically a form of hashing
- Example: Computerization
  - ization -> -ize computerize
  - ize -> ε computer

# Errors in Stemming:

There are mainly two errors in stemming –

- over-stemming
- under-stemming

# Porter

- No lexicon needed
- Basically a set of staged sets of rewrite rules that strip suffixes
- Handles both inflectional and derivational suffixes
  - ➔ Doesn't guarantee that the resulting stem is really a stem (see first bullet)
  - ➔ Lack of guarantee doesn't matter for IR

# Porter Stemmer

---

## ■ Errors of Omission

- European      Europe
- analysis      analyzes
- matrices      matrix
- noise      noisy
- explain      explanation

## ■ Errors of Commission

- organization      organ
- doing      doe
- generalization      generic
- numerical      numerous
- university      universe

# References

---

- Speech and Language processing: An introduction to Natural Language Processing, Computational Linguistics and speech Recognition by Daniel Jurafsky and James H. Martin
- Natural language understanding by James Allen

# Thank You... 😊

- Q&A
- Suggestions / Feedback



# Natural Language Processing

## DSECL ZG565



**BITS** Pilani  
Pilani Campus

Prof.Vijayalakshmi  
BITS-Pilani



## **Session 6 -Probabilistic Context Free Grammar**

**Date – 26 Dec 2021**

**Time – 8.45 to 10.45**

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Jurafsky and Prof. Martin and many others who made their course materials freely available online.

# Session Content



(Ref: Chapter 14 Jurafsky and Martin)

---

- CKY Parsing
- Probabilistic Context-Free Grammars
- PCFG for disambiguation
- Probabilistic CKY Parsing of PCFGs
- Ways to Learn PCFG Rule Probabilities
- Probabilistic Lexicalized CFGs
- Evaluating Parsers
- Problems with PCFGs

# Parsing algorithm

---

- **Top-down vs. bottom-up:**

Top-down: (goal-driven): from the start symbol down.

Bottom-up: (data-driven): from the symbols up.

- **Naive vs. dynamic programming:**

Naive: enumerate everything.

Backtracking: try something, discard partial solutions.

Dynamic programming: save partial solutions in a table.

- **Examples:**

CKY: bottom-up dynamic programming.

Earley parsing: top-down dynamic programming.

Chart parsing –bottom up chart parsing

# Issues with top down and Bottom up parsing

- Top-down and bottom-up parsing both lead to repeated substructures
  - Globally bad parses can construct good subtrees
  - But overall parse will fail
  - Require reconstruction on other branch
- No static backtracking strategy can avoid
- Efficient parsing techniques require storage of shared substructure
  - Typically with dynamic programming

Example: a flight from Indianapolis to Houston on TWA

# CKY parsing

classic, bottom-up dynamic programming algorithm (Cocke-Kasami-Younger).

Requires input grammar based on Chomsky Normal Form

- A CNF grammar is a Context-Free Grammar in which:
  - Every rule LHS is a non-terminal
  - Every rule RHS consists of either a single terminal or two non-terminals.
  - Examples:
    - $A \rightarrow BC$
    - $NP \rightarrow N PP$
    - $A \rightarrow a$
    - Noun  $\rightarrow$  man
  - But not:
    - $NP \rightarrow \text{the } N$
    - $S \rightarrow VP$

# Chomsky Normal Form

---

Any CFG can be re-written in CNF, without any loss of expressiveness.

- That is, for any CFG, there is a corresponding CNF grammar which accepts exactly the same set of strings as the original CFG.

# Converting a CFG to CNF



To convert a CFG to CNF, we need to deal with three issues:

1. Rules that mix terminals and non-terminals on the RHS
  - E.g.  $NP \rightarrow \text{the Nominal}$
2. Rules with a single non-terminal on the RHS (called unit productions)
  - E.g.  $NP \rightarrow \text{Nominal}$
3. Rules which have more than two items on the RHS
  - E.g.  ~~$NP \rightarrow \text{Det Noun PP}$~~

# Converting a CFG to CNF

---

1. Rules that mix terminals and non-terminals on the RHS

- E.g.  $NP \rightarrow \text{the Nominal}$
- Solution:
  - Introduce a dummy non-terminal to cover the original terminal
    - E.g.  $\text{Det} \rightarrow \text{the}$
  - Re-write the original rule:
    - $NP \rightarrow \text{Det Nominal}$

# Converting a CFG to CNF

## 2. Rules with a single non-terminal on the RHS (called unit productions)

- E.g.  $NP \rightarrow Nominal$
- Solution:
  - Find all rules that have the form  $Nominal \rightarrow \dots$ 
    - $Nominal \rightarrow Noun\ PP$
    - $Nominal \rightarrow Det\ Noun$
  - Re-write the above rule several times to eliminate the intermediate non-terminal:
    - $NP \rightarrow Noun\ PP$
    - $NP \rightarrow Det\ Noun$
- Note that this makes our grammar “flatter”

# Converting a CFG to CNF



3. Rules which have more than two items on the RHS
  - E.g.  $NP \rightarrow Det\ Noun\ PP$

Solution:

- Introduce new non-terminals to spread the sequence on the RHS over more than 1 rule.
  - $Nominal \rightarrow Noun\ PP$
  - $NP \rightarrow Det\ Nominal$

# CNF Grammar

---

If we parse a sentence with a CNF grammar, we know that:

- Every phrase-level non-terminal (above the part of speech level) will have exactly 2 daughters.
  - $\text{NP} \rightarrow \text{Det N}$
- Every part-of-speech level non-terminal will have exactly 1 daughter, and that daughter is a terminal:
  - $\text{N} \rightarrow \text{lady}$

# Recognising strings with CKY

---

Example input: The flight includes a meal.

The CKY algorithm proceeds by:

1. Splitting the input into words and indexing each position.  
(0) the (1) flight (2) includes (3) a (4) meal (5)
  
2. Setting up a table. For a sentence of length  $n$ , we need  $(n+1)$  rows and  $(n+1)$  columns.
  
3. Traversing the input sentence left-to-right
  
4. Use the table to store constituents and their span.

# CKY example

---

$S \rightarrow NP\ VP$

$NP \rightarrow Det\ N$

$VP \rightarrow V\ NP$

$V \rightarrow \text{includes}$

$Det \rightarrow \text{the}$

$Det \rightarrow a$

$N \rightarrow \text{meal}$

$N \rightarrow \text{flight}$

# CKY: lexical step ( $j = 1$ )

	1	2	3	4	5
0	Det				
1					
2					
3					
4					
5					

Lexical lookup

- Matches Det → the

*The flight includes a meal.*

# CKY: lexical step ( $j = 2$ )

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>0</b>	Det				
<b>1</b>		N			
<b>2</b>					
<b>3</b>					
<b>4</b>					
<b>5</b>					

Lexical lookup

- Matches N → flight

*The **flight** includes a meal.*

# CKY: syntactic step ( $j = 2$ )

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>0</b>	Det	NP			
<b>1</b>		N			
<b>2</b>					
<b>3</b>					
<b>4</b>					
<b>5</b>					

Syntactic lookup:

- look backwards and see if there is any rule that will cover what we've done so far.

*The flight includes a meal.*

# CKY: lexical step ( $j = 3$ )

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>0</b>	Det	NP			
<b>1</b>		N			
<b>2</b>			V		
<b>3</b>					
<b>4</b>					
<b>5</b>					

Lexical lookup

- Matches V → includes

*The flight **includes** a meal.*

# CKY: lexical step ( $j = 3$ )

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>0</b>	Det	NP			
<b>1</b>		N			
<b>2</b>			V		
<b>3</b>					
<b>4</b>					
<b>5</b>					

## Syntactic lookup

- There are no rules in our grammar that will cover Det, NP, V

*The flight **includes** a meal.*

# CKY: lexical step ( $j = 4$ )

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>0</b>	Det	NP			
<b>1</b>		N			
<b>2</b>			V		
<b>3</b>				Det	
<b>4</b>					
<b>5</b>					

Lexical lookup

- Matches Det → a

*The flight includes a meal.*

# CKY: lexical step ( $j = 5$ )

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>0</b>	Det	NP			
<b>1</b>		N			
<b>2</b>			V		
<b>3</b>				Det	
<b>4</b>					N

Lexical lookup

- Matches N → meal

*The flight includes a meal.*

# CKY: syntactic step ( $j = 5$ )

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>0</b>	Det	NP			
<b>1</b>		N			
<b>2</b>			V		
<b>3</b>				Det	NP
<b>4</b>					N

## Syntactic lookup

- We find that we have  
 $NP \rightarrow Det\ N$

*The flight includes a meal.*

# CKY: syntactic step ( $j = 5$ )

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>0</b>	Det	NP			
<b>1</b>		N			
<b>2</b>			V		VP
<b>3</b>				Det	NP
<b>4</b>					N

Syntactic lookup

- We find that we have  
 $\text{VP} \rightarrow \text{V NP}$

*The flight includes a meal.*

# CKY: syntactic step ( $j = 5$ )

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>0</b>	Det	NP			S
<b>1</b>		N			
<b>2</b>			V		VP
<b>3</b>				Det	NP
<b>4</b>					N

Syntactic lookup

- We find that we have S  
 $\rightarrow$  NP VP

*The flight includes a meal.*

# From recognition to parsing

---

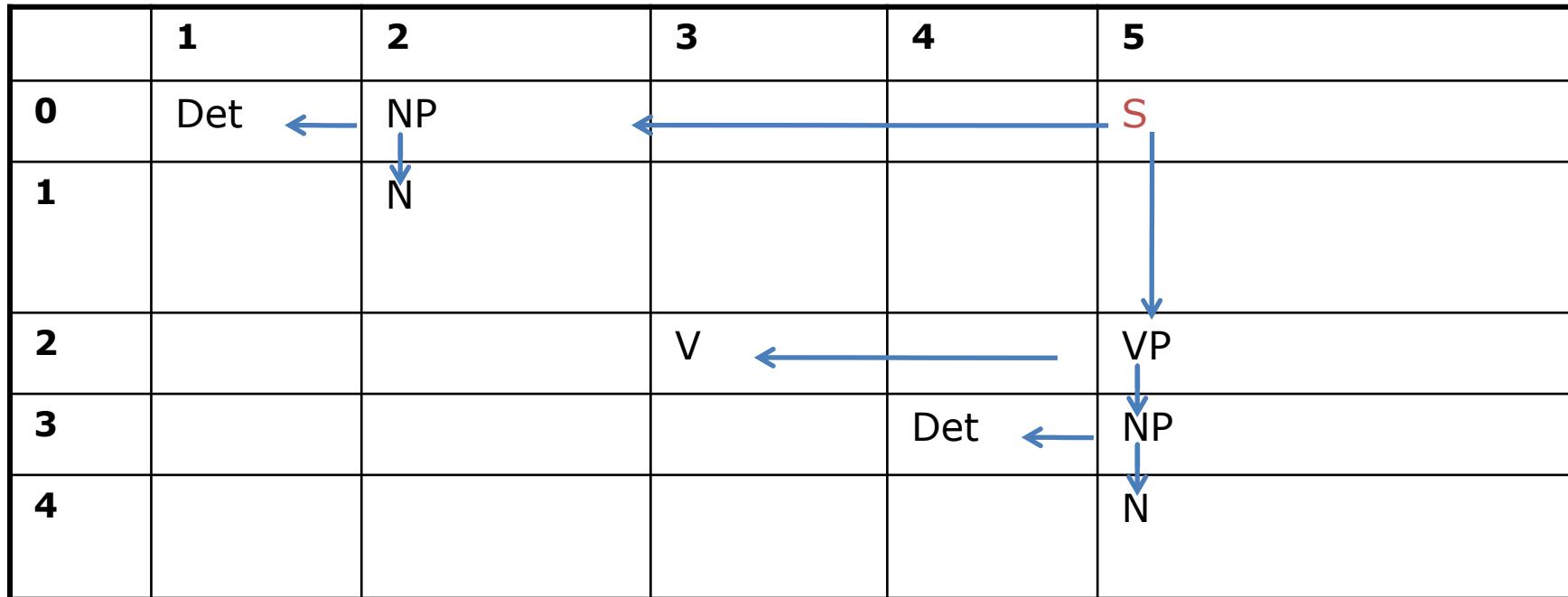
The procedure so far will recognise a string as a legal sentence in English.

But we'd like to get a parse tree back!

Solution:

- We can work our way back through the table and collect all the partial solutions into one parse tree.
- Cells will need to be augmented with “backpointers”, i.e. With a pointer to the cells that the current cell covers.

# From recognition to parsing



# From recognition to parsing

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>0</b>	Det ←	NP ↓			S ↓
<b>1</b>		N			
<b>2</b>			V ←		VP ↓
<b>3</b>				Det ←	NP ↓
<b>4</b>					N ↓

NB: This algorithm always fills the top “triangle” of the table!

# What about ambiguity?

---

The algorithm does not assume that there is only one parse tree for a sentence.

- (Our simple grammar did not admit of any ambiguity, but this isn't realistic of course).

There is nothing to stop it returning several parse trees.

If there are multiple local solutions, then more than one non-terminal will be stored in a cell of the table.

---

## Some funny examples

- Policeman to littleboy: “We are looking for a thief with a bicycle.” Little boy: “Wouldn’t you be better using your eyes.”
- Why is the teacher wearing sun-glasses.  
Because the class is so bright.

# Ambiguity is Explosive

Ambiguities compound to generate enormous numbers of possible interpretations.

In English, a sentence ending in  $n$  prepositional phrases has over  $2^n$  syntactic interpretations (cf. Catalan numbers).

- “I saw the man with the telescope”: 2 parses
- “I saw the man on the hill with the telescope.”: 5 parses
- “I saw the man on the hill in Texas with the telescope”: 14 parses
- “I saw the man on the hill in Texas with the telescope at noon.”: 42 parses
- “I saw the man on the hill in Texas with the telescope at noon on Monday” 132 parses

# Motivation

- Context-free grammars can be generalized to include probabilistic information by adding it to CFG rule
- Probabilistic Context Free Grammars (PCFGs) are the simplest and most natural probabilistic model for tree structures and the algorithms for them are closely related to those for HMMs.
- PCFG are also known as Stochastic Context-Free Grammar (SCFG)

# CFG definition (reminder)

---

A CFG is a 4-tuple:  $(N, \Sigma, R, S)$ :

- $N$  = a set of non-terminal symbols (e.g. NP, VP)
- $\Sigma$  = a set of terminals (e.g. words)
  - $N$  and  $\Sigma$  are disjoint (no element of  $N$  is also an element of  $\Sigma$ )
- $R$  = a set of rules of the form  $A \rightarrow \beta$  where:
  - $A$  is a non-terminal (a member of  $N$ )
  - $\beta$  is any string of terminals and non-terminals
- $S$  = a designated start symbol (usually, “sentence”)

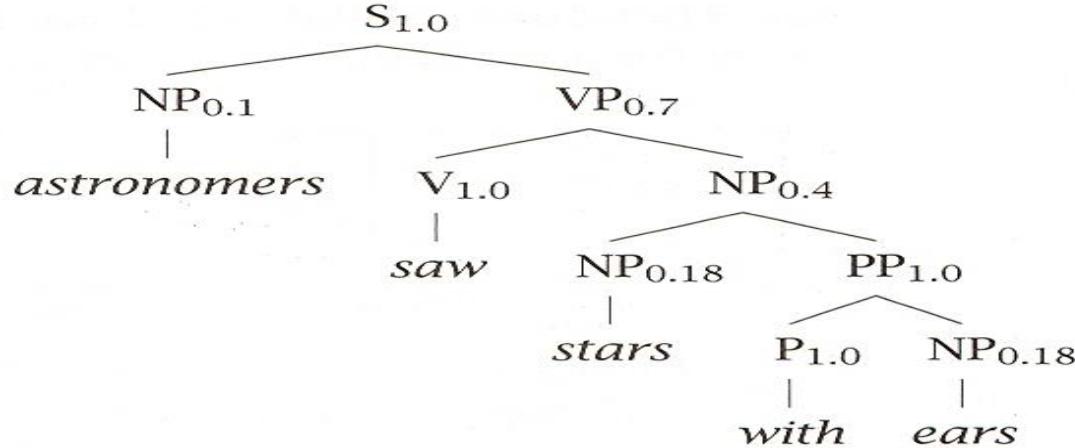
# Formal Definition of a PCFG

---

- A PCFG consists of:
  - A set of terminals,  $\{w^k\}$ ,  $k= 1, \dots, V$
  - A set of nonterminals,  $N^i$ ,  $i= 1, \dots, n$
  - A designated start symbol  $N^1$
  - A set of **rules**,  $\{N^i \rightarrow \xi^j\}$ , (where  $\xi^j$  is a sequence of terminals and nonterminals)
  - A corresponding set of **probabilities on rules** such that:  $\forall i \quad \sum_j P(N^i \rightarrow \xi^j) = 1$

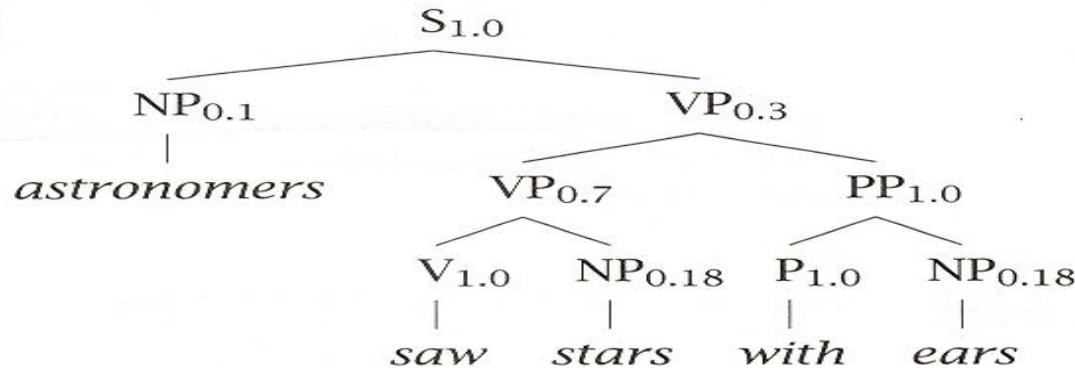
# Example: Probability of a Derivation Tree

$t_1:$



$S \rightarrow NP\ VP$	1.0	$NP \rightarrow NP\ PP$	0.4
$PP \rightarrow P\ NP$	1.0	$NP \rightarrow astronomers$	0.1
$VP \rightarrow V\ NP$	0.7	$NP \rightarrow ears$	0.18
$VP \rightarrow VP\ PP$	0.3	$NP \rightarrow saw$	0.04
$P \rightarrow with$	1.0	$NP \rightarrow stars$	0.18
$V \rightarrow saw$	1.0	$NP \rightarrow telescopes$	0.1

$t_2:$



# Probability of a Derivation Tree and a String



- The probability of a derivation (i.e. parse) tree:

$$P(T) = \prod_{i=1..k} p(r(i))$$

where  $r(1), \dots, r(k)$  are the rules of the CFG used to generate the sentence  $w_{1m}$  of which  $T$  is a parse.

- The probability of a sentence (according to grammar  $G$ ) is given by:

$$P(w_{1m}) = \sum_t P(w_{1m}, t) = \sum_{\{t: \text{yield}(t)=w_{1m}\}} P(t)$$

where  $t$  is a parse tree of the sentence. Need dynamic programming to make this efficient!

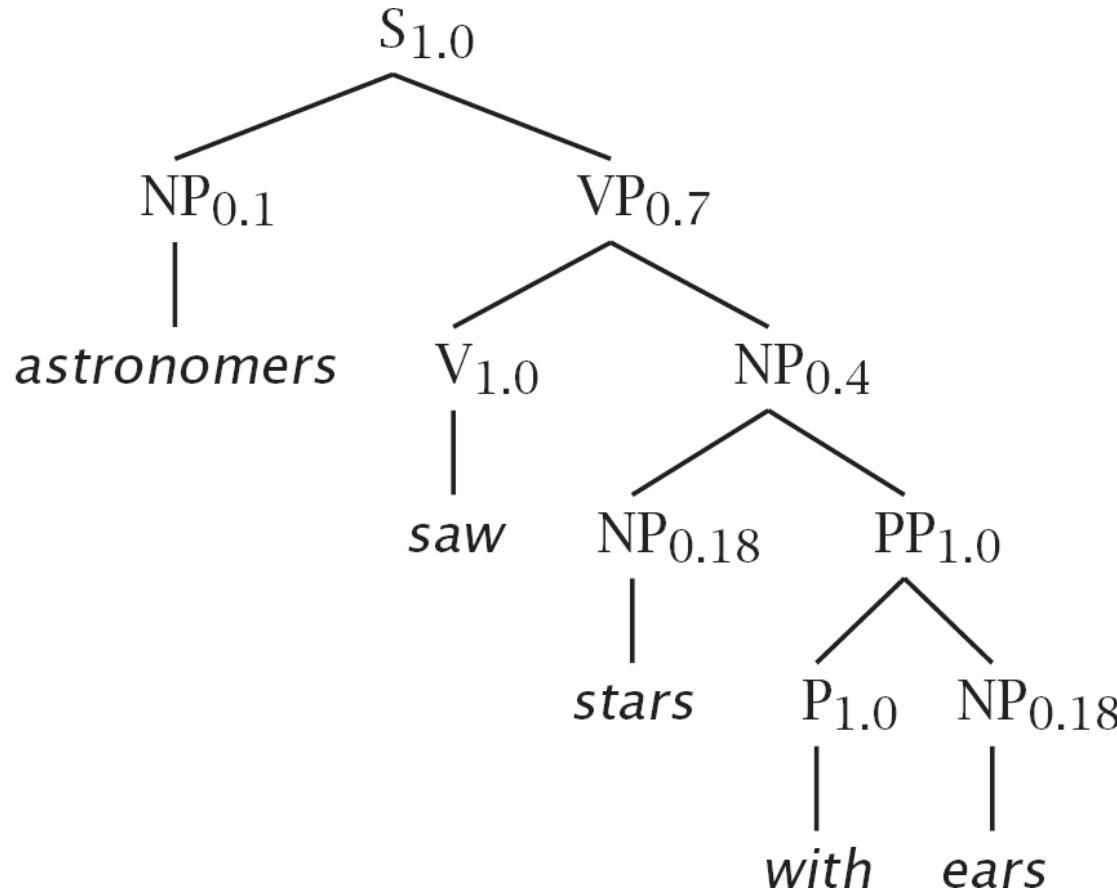
# Example

$S \rightarrow NP\ VP$	1.0	$NP \rightarrow NP\ PP$	0.4
$PP \rightarrow P\ NP$	1.0	$NP \rightarrow \text{astronomers}$	0.1
$VP \rightarrow V\ NP$	0.7	$NP \rightarrow \text{ears}$	0.18
$VP \rightarrow VP\ PP$	0.3	$NP \rightarrow \text{saw}$	0.04
$P \rightarrow \text{with}$	1.0	$NP \rightarrow \text{stars}$	0.18
$V \rightarrow \text{saw}$	1.0	$NP \rightarrow \text{telescopes}$	0.1

- Terminals      with, saw, astronomers, ears, stars, telescopes
- Nonterminals    S, PP, P, NP, VP, V
- Start symbol    S

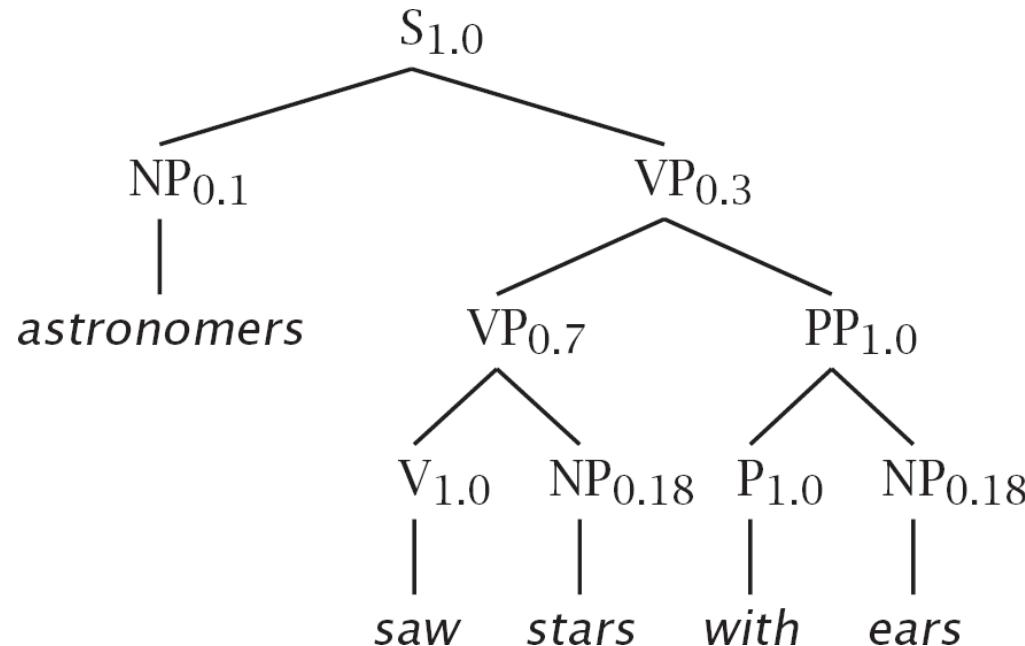
# astronomers saw stars with ears

$t_1:$



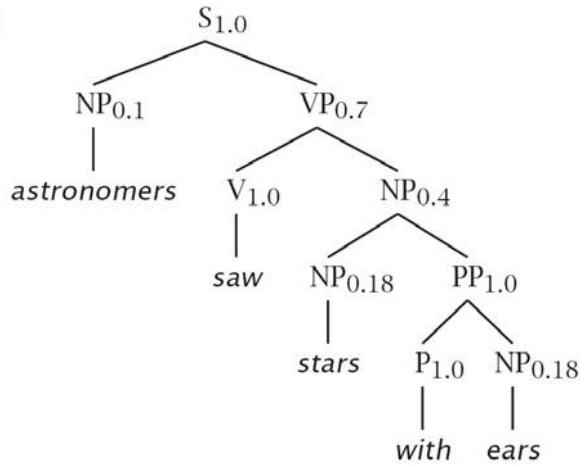
# astronomers saw stars with ears

$t_2:$

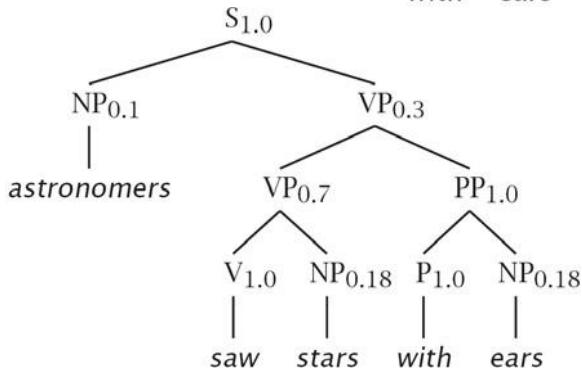


# Probabilities

$t_1:$



$t_2:$

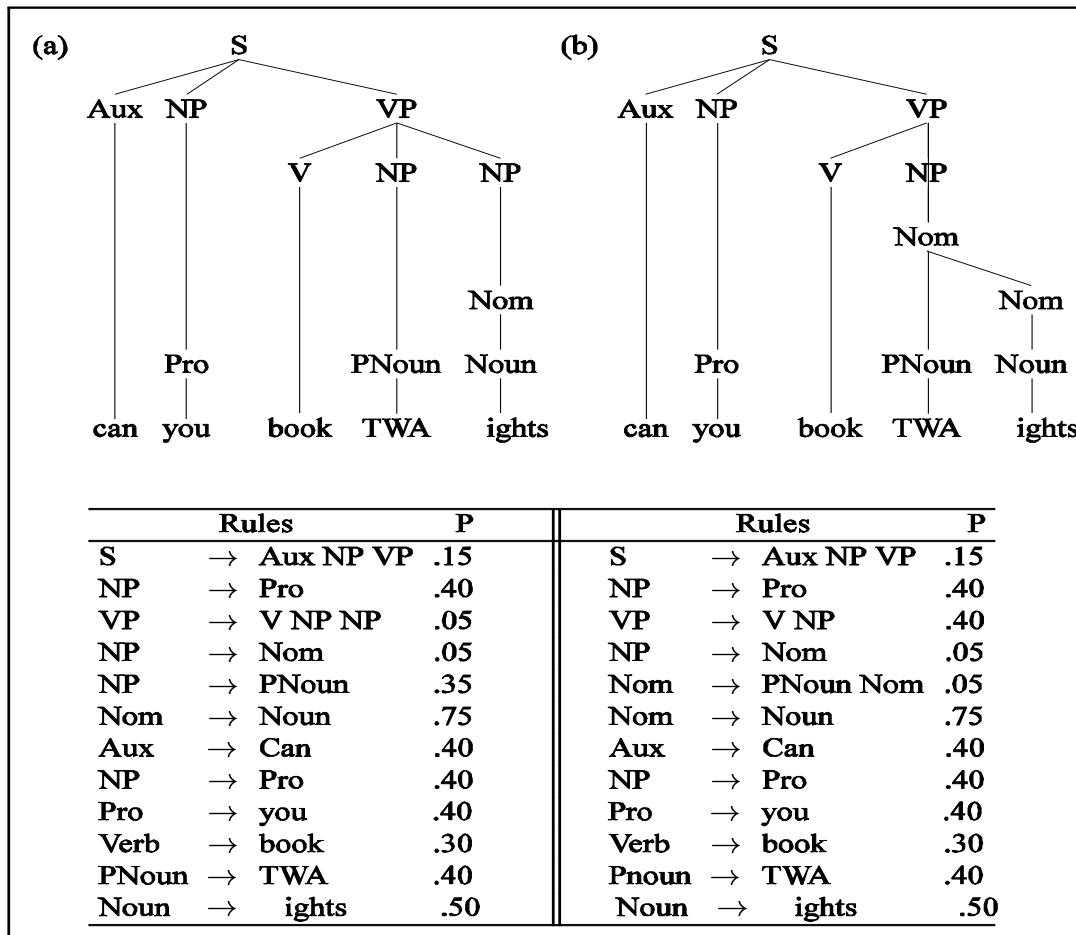


$$\begin{aligned}
 P(t_1) &= 1.0 \times 0.1 \times 0.7 \times 1.0 \times 0.4 \\
 &\quad \times 0.18 \times 1.0 \times 1.0 \times 0.18 \\
 &= 0.0009072
 \end{aligned}$$

$$\begin{aligned}
 P(t_2) &= 1.0 \times 0.1 \times 0.3 \times 0.7 \times 1.0 \\
 &\quad \times 0.18 \times 1.0 \times 1.0 \times 0.18 \\
 &= 0.0006804
 \end{aligned}$$

$$P(w_{15}) = P(t_1) + P(t_2) = 0.0015876$$

# Example2



$$P(TL) = 1.5 \times 10^{-6}$$

$$P(TR) = 1.7 \times 10^{-6}$$

$$P(S) = 3.2 \times 10^{-6}$$

# Some Features of PCFGs

- A PCFG gives some idea of the plausibility of different parses; however, the probabilities are based on **structural factors** and **not lexical ones**.
- PCFGs are good for grammar induction.
- PCFGs are robust.
- PCFGs give a **probabilistic language model** for English.
- The predictive power of a PCFG tends to be greater than for an HMM.
- PCFGs are not good models alone but they can be combined with a trigram model.

# Properties of PCFGs

---

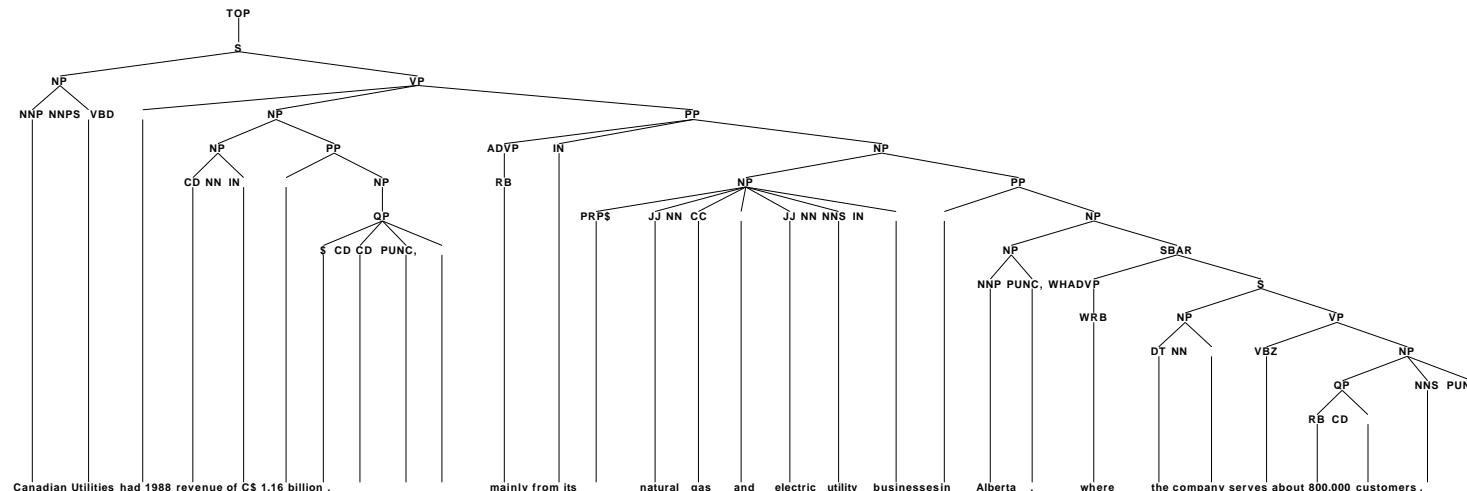
- ▶ Assigns a probability to each *left-most derivation*, or parse-tree, allowed by the underlying CFG
- ▶ Say we have a sentence  $s$ , set of derivations for that sentence is  $T(s)$ . Then a PCFG assigns a probability  $p(t)$  to each member of  $T(s)$ . i.e., *we now have a ranking in order of probability.*
- ▶ The most likely parse tree for a sentence  $s$  is

$$\arg \max_{t \in T(s)} p(t)$$

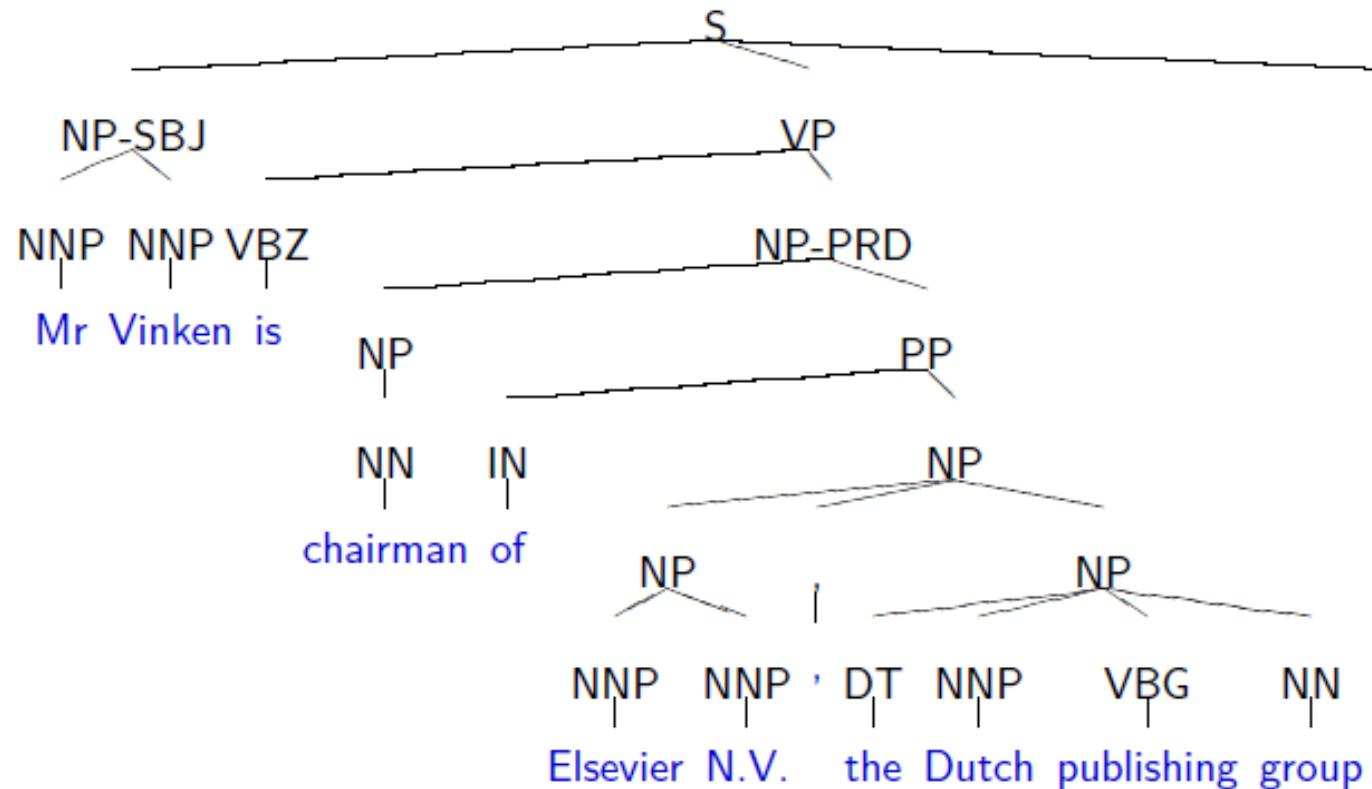
# Data for Parsing Experiments: Treebanks

- ▶ Penn WSJ Treebank = 50,000 sentences with associated trees
- ▶ Usual set-up: 40,000 training sentences, 2400 test sentences

An example tree:



# Example tree



# Word/Tag Counts

	<b>N</b>	<b>V</b>	<b>ARI</b>	<b>P</b>	<b>TOTAL</b>
<i>flies</i>	21	23	0	0	44
<i>fruit</i>	49	5	1	0	55
<i>like</i>	10	30	0	21	61
<i>a</i>	1	0	201	0	202
<i>the</i>	1	0	300	2	303
<i>flower</i>	53	15	0	0	68
<i>flowers</i>	42	16	0	0	58
<i>birds</i>	64	1	0	0	65
<i>others</i>	592	210	56	284	1142
<b>TOTAL</b>	<b>833</b>	<b>300</b>	<b>558</b>	<b>307</b>	<b>1998</b>

# Lexical Probability Estimates

Reckless	.54	Reckless	.36
Reckless	.05	Reckless	.01
Reckless	.05	Reckless	.03
Reckless	.1	Reckless	.05
Reckless	.08	Reckless	.05
Reckless	.02		

# The PCFG

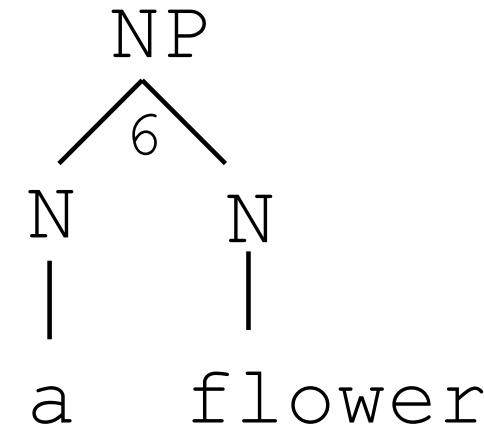
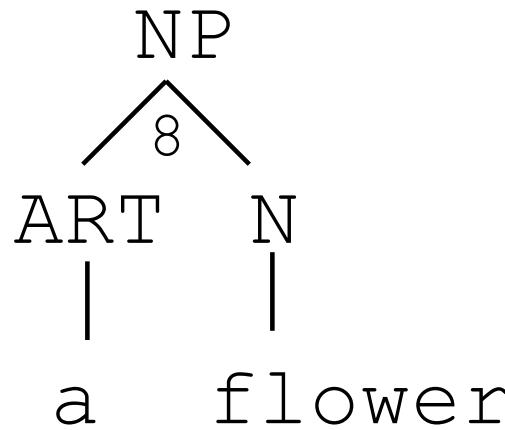
- Below is a probabilistic CFG (PCFG) with probabilities derived from analyzing a parsed version of Allen's corpus.

<b>Rule</b>	<b>Count for LHS</b>	<b>Count for Rule</b>	<b>Prob</b>
1. $S \rightarrow NP VP$	300	300	1
2. $VP \rightarrow V$	300	116	.386
3. $VP \rightarrow VNP$	300	118	.393
4. $VP \rightarrow VNPPP$	300	66	.22
5. $NP \rightarrow NPPP$	1032	241	.23
6. $NP \rightarrow NN$	1032	92	.09
7. $NP \rightarrow N$	1032	141	.14
8. $NP \rightarrow ARTN$	1032	558	.54
9. $PP \rightarrow PNP$	307	307	1

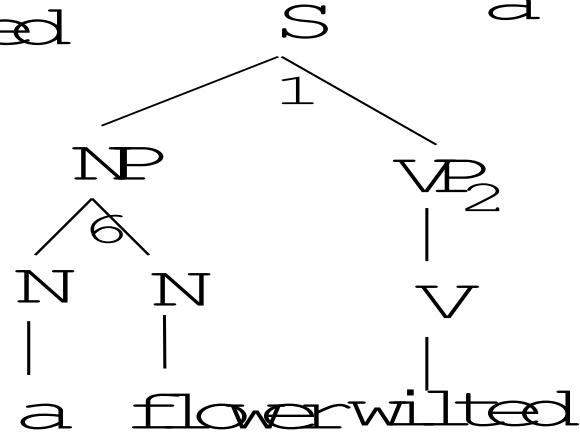
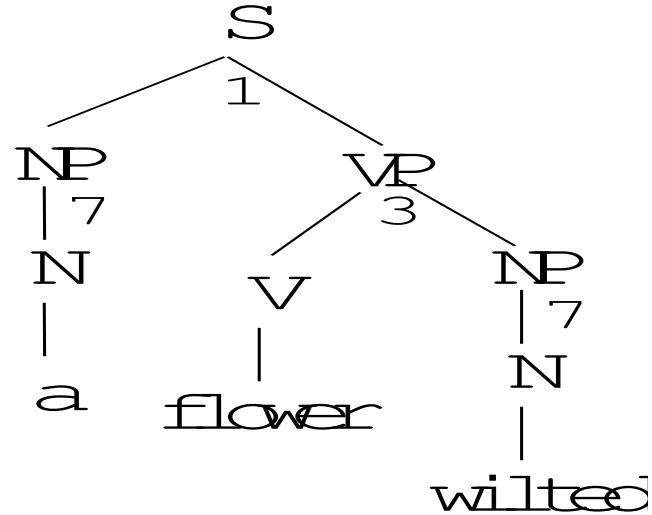
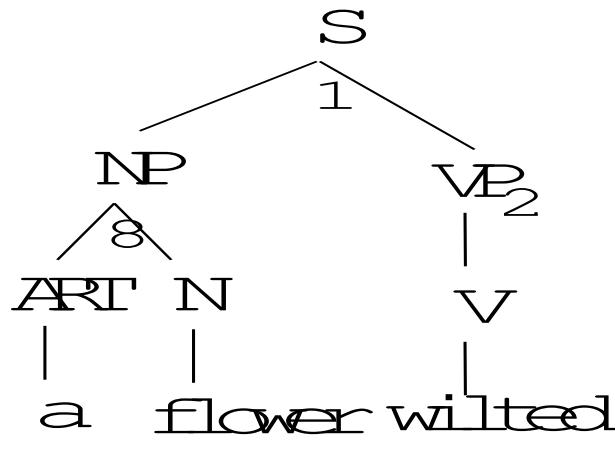
# Parsing with a PCFG



- Using the lexical probabilities, we can derive probabilities that the constituent NP generates a sequence like *a flower*. Two rules could generate the string of words:



# Three Possible Trees for an S



# Parsing with a PCFG

- The probability of a sentence generating *A flower wilted*:

$$P(a \text{ flower } \text{wilted} | S) = P(R_1 | S) \times P(a \text{ flower} | NP) \times \\ P(\text{wilted} | VP) + P(R_1 | S) \times P(a | NP) \times P(\text{flower} \\ \text{wilted} | VP) + ..$$

Using this approach, the probability that a given sentence will be generated by the grammar can be efficiently computed.

- It only requires some way of recording the value of each constituent between each two possible positions. The requirement can be filled by a packed chart structure.

# Uses of probabilities in parsing

**Disambiguation:** given  $n$  legal parses of a string, which is the most likely?

- e.g. PP-attachment ambiguity can be resolved this way

**Speed:** we've defined parsing as a search problem

- search through space of possible applicable derivations
- search space can be pruned by focusing on the most likely sub-parses of a parse

Parser can be used as a model to determine the probability of a sentence, given a parse

- typical use in speech recognition, where input utterance can be “heard” as several possible sentences

# Using PCFG probabilities

---

PCFG assigns a probability to every parse-tree  $t$  of a string  $W$

- e.g. every possible parse (derivation) of a sentence recognised by the grammar

Notation:

- $G$  = a PCFG
- $s$  = a sentence
- $t$  = a particular tree under our grammar
  - $t$  consists of several nodes  $n$
  - each node is generated by applying some rule  $r$

# Probability of a tree vs. a sentence

---

We work out the probability of a parse tree  $t$  by multiplying the probability of every rule (node) that gives rise to  $t$  (i.e. the derivation of  $t$ ).

Note that:

- A tree can have multiple derivations
  - (different sequences of rule applications could give rise to the same tree)
- But the probability of the tree remains the same
  - (it's the same probabilities being multiplied)
- We usually speak as if a tree has only one derivation, called the **canonical derivation**

# Picking the best parse in a PCFG

---

A sentence will usually have several parses

- we usually want them ranked, or only want the  $n$  best parses
- we need to focus on  $P(t|s, G)$ 
  - probability of a parse, given our sentence and our grammar
- definition of the best parse for  $s$ :
  - The tree for which  $P(t|s, G)$  is highest

# Probability of a sentence

---

Given a probabilistic context-free grammar  $G$ , we can the probability of a sentence (as opposed to a tree).

Observe that:

- As far as our grammar is concerned, a sentence is only a sentence if it can be recognised by the grammar (it is “legal”)
- There can be multiple parse trees for a sentence.
  - Many trees whose **yield** is the sentence
- The probability of the sentence is the sum of all the probabilities of the various trees that yield the sentence.

# Using CKY to parse with a PCFG

---

The basic CKY algorithm remains unchanged.

However, rather than only keeping partial solutions in our table cells (i.e. The rules that match some input), we also keep their probabilities.

# Probabilistic CKY: example PCYK

---

S → NP VP [.80]

NP → Det N [.30]

VP → V NP [.20]

V → includes [.05]

Det → the [.4]

Det → a [.4]

N → meal [.01]

N → flight [.02]

# Probabilistic CYK: initialisation

	1	2	3	4	5
0					
1					
2					
3					
4					
5					

- $S \rightarrow NP\ VP$  [.80]
- $NP \rightarrow Det\ N$  [.30]
- $VP \rightarrow V\ NP$  [.20]
- $V \rightarrow \text{includes}$  [.05]
- $Det \rightarrow \text{the}$  [.4]
- $Det \rightarrow a$  [.4]
- $N \rightarrow \text{meal}$  [.01]
- $N \rightarrow \text{flight}$  [.02]

*The flight includes a meal.*

# Probabilistic CYK: lexical step

	1	2	3	4	5
0	Det (.4)				
1					
2					
3					
4					
5					

- $S \rightarrow NP\ VP$  [.80]
- $NP \rightarrow Det\ N$  [.30]
- $VP \rightarrow V\ NP$  [.20]
- $V \rightarrow$  includes [.05]
- $Det \rightarrow$  the [.4]
- $Det \rightarrow$  a [.4]
- $N \rightarrow$  meal [.01]
- $N \rightarrow$  flight [.02]

*The flight includes a meal.*

# Probabilistic CYK: lexical step

	1	2	3	4	5
0	Det (.4)				
1		N .02			
2					
3					
4					
5					

- $S \rightarrow NP\ VP$  [.80]
- $NP \rightarrow Det\ N$  [.30]
- $VP \rightarrow V\ NP$  [.20]
- $V \rightarrow includes$  [.05]
- $Det \rightarrow the$  [.4]
- $Det \rightarrow a$  [.4]
- $N \rightarrow meal$  [.01]
- $N \rightarrow flight$  [.02]

*The **flight** **includes** a meal.*

# Probabilistic CYK: syntactic step

	1	2	3	4	5
0	Det (.4)	NP .0024			
1		N .02			
2					
3					
4					
5					

- $S \rightarrow NP VP [ .80 ]$
- $NP \rightarrow Det N [ .30 ]$
- $VP \rightarrow V NP [ .20 ]$
- $V \rightarrow includes [ .05 ]$
- $Det \rightarrow the [ .4 ]$
- $Det \rightarrow a [ .4 ]$
- $N \rightarrow meal [ .01 ]$
- $N \rightarrow flight [ .02 ]$

*The flight includes a meal.*

Note: probability of NP in [0,2]  
 $P(Det \rightarrow the) * P(N \rightarrow flight) * P(NP \rightarrow Det N)$

# Probabilistic CYK: lexical step

- $S \rightarrow NP VP [.80]$
- $NP \rightarrow Det N [.30]$
- $VP \rightarrow V NP [.20]$
- $V \rightarrow \text{includes} [.05]$
- $Det \rightarrow \text{the} [.4]$
- $Det \rightarrow a [.4]$
- $N \rightarrow \text{meal} [.01]$
- $N \rightarrow \text{flight} [.02]$

	1	2	3	4	5
0	De t (.4 )	NP .0024			
1		N .02			
2			V .05		
3					
4					
5					

*The flight includes a meal.*

# Probabilistic CYK: lexical step

- $S \rightarrow NP\ VP [ .80 ]$
- $NP \rightarrow Det\ N [ .30 ]$
- $VP \rightarrow V\ NP [ .20 ]$
- $V \rightarrow includes [ .05 ]$
- $Det \rightarrow the [ .4 ]$
- $Det \rightarrow a [ .4 ]$
- $N \rightarrow meal [ .01 ]$
- $N \rightarrow flight [ .02 ]$

	1	2	3	4	5
0	Det (.4 )	NP .0024			
1		N .02			
2			V .05		
3				Det .4	
4					
5					

*The flight includes a meal.*

# Probabilistic CYK: syntactic step

- $S \rightarrow NP VP [.80]$
- $NP \rightarrow Det N [.30]$
- $VP \rightarrow V NP [.20]$
- $V \rightarrow \text{includes} [.05]$
- $Det \rightarrow \text{the} [.4]$
- $Det \rightarrow a [.4]$
- $N \rightarrow \text{meal} [.01]$
- $N \rightarrow \text{flight} [.02]$

	1	2	3	4	5
0	Det (.4)	NP .0024			
1		N .02			
2			V .05		
3				Det .4	
4					N .01

*The flight includes a meal.*

# Probabilistic CYK: syntactic step

- $S \rightarrow NP\ VP$  [.80]
- $NP \rightarrow Det\ N$   
[.30]
- $VP \rightarrow V\ NP$   
[.20]
- $V \rightarrow \text{includes}$   
[.05]
- $Det \rightarrow \text{the}$  [.4]
- $Det \rightarrow a$  [.4]
- $N \rightarrow \text{meal}$  [.01]
- $N \rightarrow \text{flight}$  [.02]

	1	2	3	4	5
0	Det .4)	NP .0024			
1		N .02			
2			V .05		
3				Det .4	NP .001
4					N .01

*The flight includes a meal.*

# Probabilistic CYK: syntactic step

- $S \rightarrow NP\ VP$  [.80]
- $NP \rightarrow Det\ N$  [.30]
- $VP \rightarrow V\ NP$  [.20]
- $V \rightarrow includes$  [.05]
- $Det \rightarrow the$  [.4]
- $Det \rightarrow a$  [.4]
- $N \rightarrow meal$  [.01]
- $N \rightarrow flight$  [.02]

	1	2	3	4	5
0	Det .4)	NP .0024			
1		N .02			
2			V .05		VP .00001
3				Det .4	NP .001
4					N .01

*The flight includes a meal.*

- $S \rightarrow NP\ VP$  [.80]
- $NP \rightarrow Det\ N$  [.30]
- $VP \rightarrow V\ NP$  [.20]
- $V \rightarrow$  includes [.05]
- $Det \rightarrow$  the [.4]
- $Det \rightarrow$  a [.4]
- $N \rightarrow$  meal [.01]
- $N \rightarrow$  flight [.02]

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>0</b>	Det (.4)	NP .0024			<b>S</b> .0000000192
<b>1</b>		N .02			
<b>2</b>			V .05		VP .00001
<b>3</b>				Det .4	NP .001
<b>4</b>					N .01

*The flight includes a meal.*

# Problems with PCFGs

---

- No Context
  - (immediate prior context, speaker, ...)
- No Lexicalization
  - “VP NP NP” more likely if verb is “hand” or “tell”
  - fail to capture lexical dependencies (n-grams do)

# Flaws I: Structural independence

---

Probability of a rule  $r$  expanding node  $n$  depends only on  $n$ .  
Independent of other non-terminals

Example:

- $P(NP \rightarrow Pro)$  is independent of where the NP is in the sentence
- but we know that  $NP \rightarrow Pro$  is much more likely in subject position
- Francis et al (1999) using the Switchboard corpus:
  - 91% of subjects are pronouns;
  - only 34% of objects are pronouns

# Flaws II: lexical independence

---

vanilla PCFGs ignore lexical material

- e.g.  $P(VP \rightarrow V NP PP)$  independent of the head of NP or PP or lexical head V

Examples:

- prepositional phrase attachment preferences depend on lexical items; cf:
  - dump [sacks into a bin]
  - dump [sacks] [into a bin] (preferred parse)
- coordination ambiguity:
  - [dogs in houses] and [cats]
  - [dogs] [in houses and cats]

# Lexicalised PCFGs

---

Attempt to weaken the lexical independence assumption.

Most common technique:

- mark each phrasal head (N,V, etc) with the lexical material
- this is based on the idea that the most crucial lexical dependencies are between head and dependent
- E.g.: Charniak 1997, Collins 1999

# References

---

- <https://www.youtube.com/watch?v=Z6GsoBA-09k&list=PLQiyVNMPDLKnZYBTUOISI9mi9wAErFtFm&index=62>
- <https://lost-contact.mit.edu/afs/cs.pitt.edu/projects/nltk/docs/tutorial/pcfg/nochunks.html>
- <http://www.nltk.org/howto/grammar.html>



---

Thank You



# Natural Language Processing

## DSECL ZG565

Prof.Vijayalakshmi  
BITS-Pilani

**BITS** Pilani  
Pilani Campus





## **Session 7 - Dependency Parsing**

**Date – 20<sup>th</sup> June 2021**

**Time – 9 am to 11 am**

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Jurafsky and Prof. Martin and many others who made their course materials freely available online.

# Outline

- ❖ Motivation
- ❖ Two types of parsing
  - Dependency parsing
  - Phrase structure parsing
- ❖ Dependency structure and Dependency grammar
- ❖ Dependency Relation
- ❖ Universal Dependencies
- ❖ Method of Dependency Parsing
  - Dynamic programming
  - Graph algorithms
  - Constraint satisfaction
  - Deterministic Parsing
- ❖ Transition based dependency parsing
- ❖ Graph based dependency Parsing
- ❖ Evaluation

# Interpreting Language is Hard!

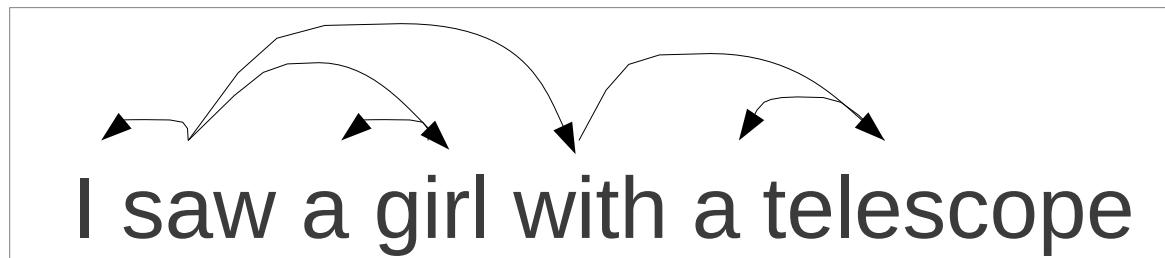
I saw a girl with a telescope



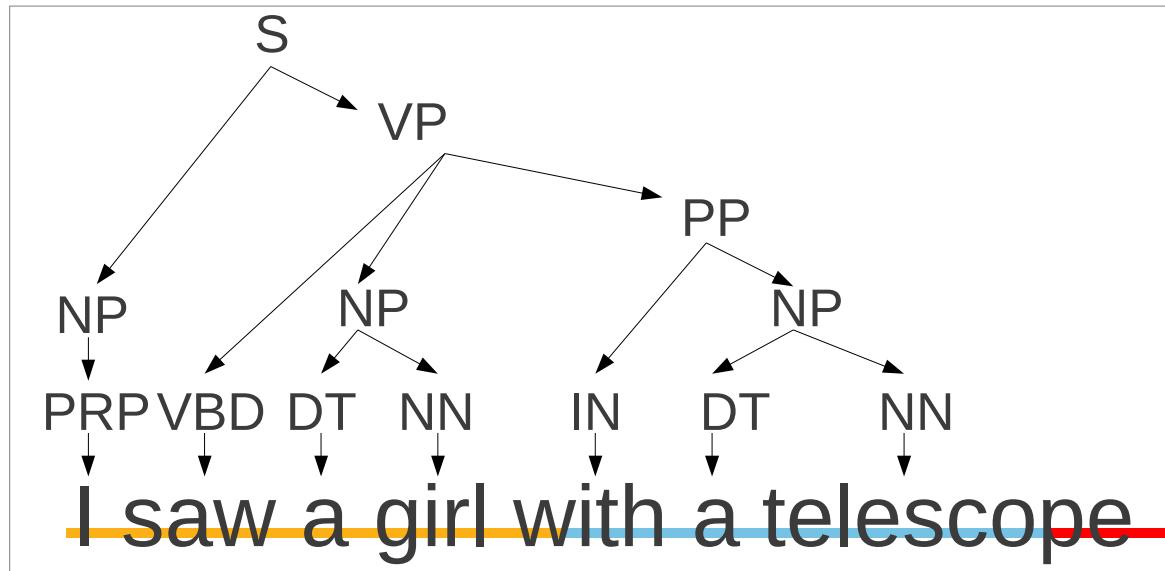
- “Parsing” resolves structural ambiguity in a formal way

## Two Types of Parsing

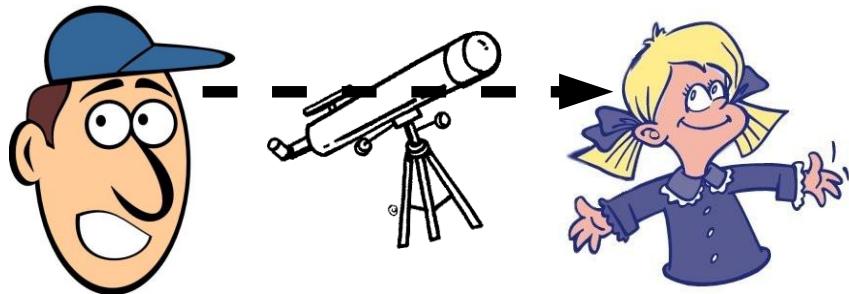
- **Dependency:** focuses on relations between words



- **Phrase structure:** focuses on identifying phrases and their recursive structure



# Dependencies Also Resolve Ambiguity



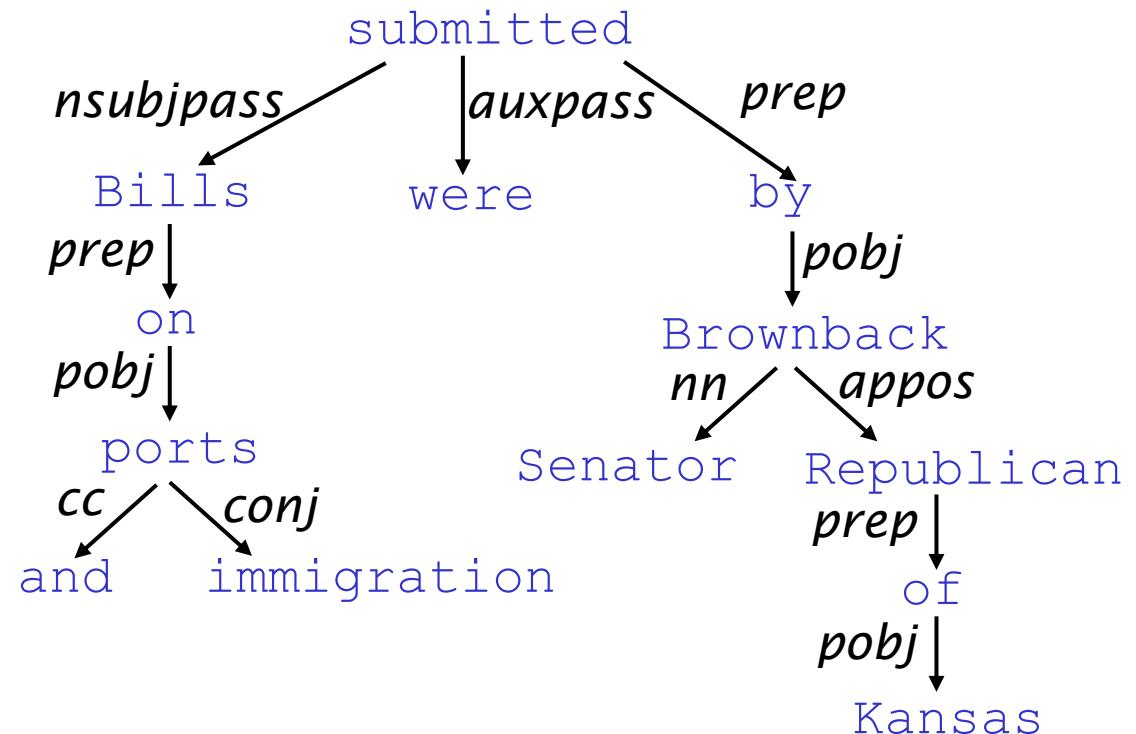
I saw a girl with a telescope

I saw a girl with a telescope

# Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of lexical items linked by binary asymmetric relations (“arrows”) called dependencies

The arrows are commonly **typed** with the name of grammatical relations (subject, prepositional object, apposition, etc.)

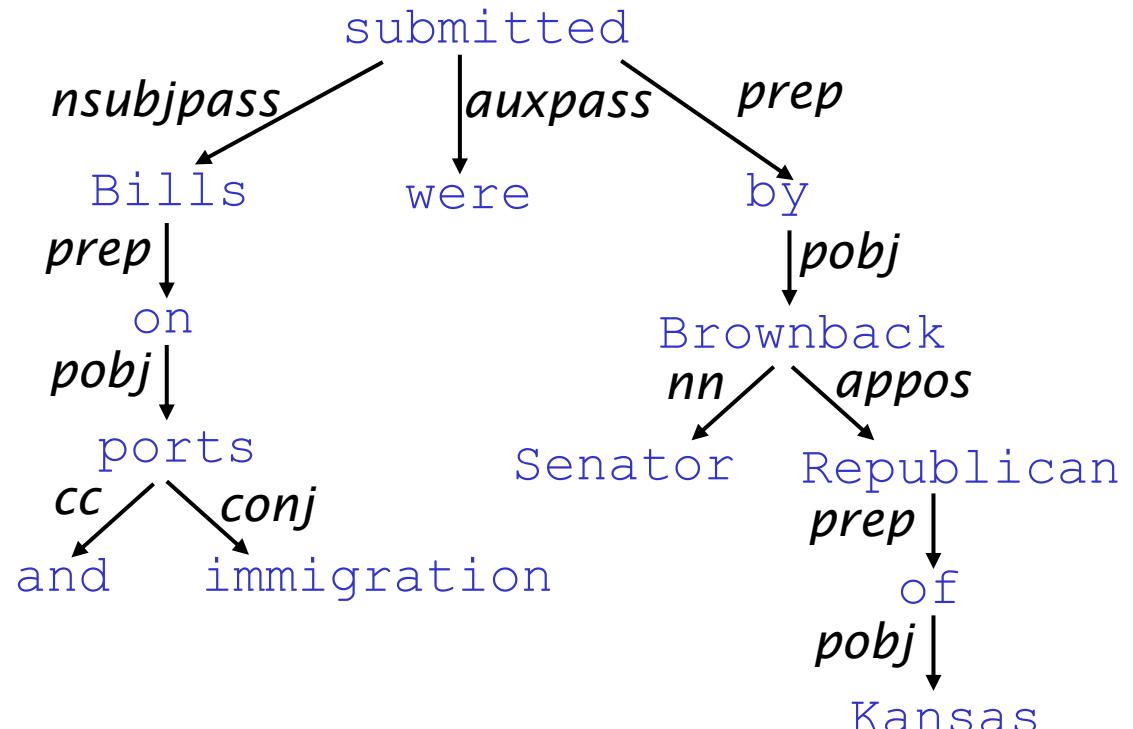


# Dependency Grammar and Dependency Structure



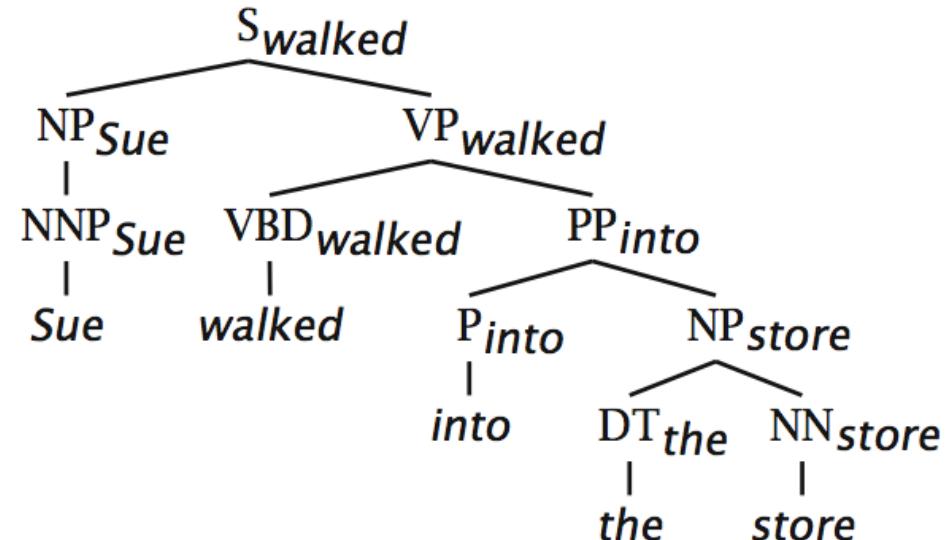
The arrow connects a **head** (governor, superior, regent) with a **dependent** (modifier, inferior, subordinate)

Usually, dependencies form a tree (connected, acyclic, single-head)



# Relation between phrase structure and dependency structure

- A dependency grammar has a notion of a head. Officially, CFGs don't.
- But modern linguistic theory and all modern statistical parsers (Charniak, Collins, Stanford, ...) do, via hand-written phrasal “head rules”:
  - The head of a Noun Phrase is a noun/number/adj/...
  - The head of a Verb Phrase is a verb/modal/....
- The head rules can be used to extract a dependency parse from a CFG parse
- The closure of dependencies give constituency from a dependency tree
- But the dependents of a word must be at the same level (i.e., “flat”)



# Dependency graph

- A dependency structure can be defined as a directed graph  $G$ , consisting of
  - ▶ a set  $V$  of nodes,
  - ▶ a set  $A$  of arcs (edges),
- Labeled graphs:
  - ▶ Nodes in  $V$  are labeled with word forms (and annotation).
  - ▶ Arcs in  $A$  are labeled with dependency types.
- Notational convention:
  - ▶ Arc  $(w_i, d, w_j)$  links head  $w_i$  to dependent  $w_j$  with label  $d$
  - ▶  $w_i \xrightarrow{d} w_j \Leftrightarrow (w_i, d, w_j) \in A$
  - ▶  $i \rightarrow j \equiv (i, j) \in A$

# Formal conditions on dependency graph



- $G$  is connected:
  - For every node  $i$  there is a node  $j$  such that  $i \rightarrow j$  or  $j \rightarrow i$ .
- $G$  is acyclic:
  - if  $i \rightarrow j$  then not  $j \rightarrow^* i$ .
- $G$  obeys the single head constraint:
  - if  $i \rightarrow j$  then not  $k \rightarrow j$ , for any  $k \neq i$ .
- $G$  is projective:
  - if  $i \rightarrow j$  then  $j \rightarrow^* k$ , for any  $k$  such that both  $j$  and  $k$  lie on the same side of  $i$ .

# Universal dependencies

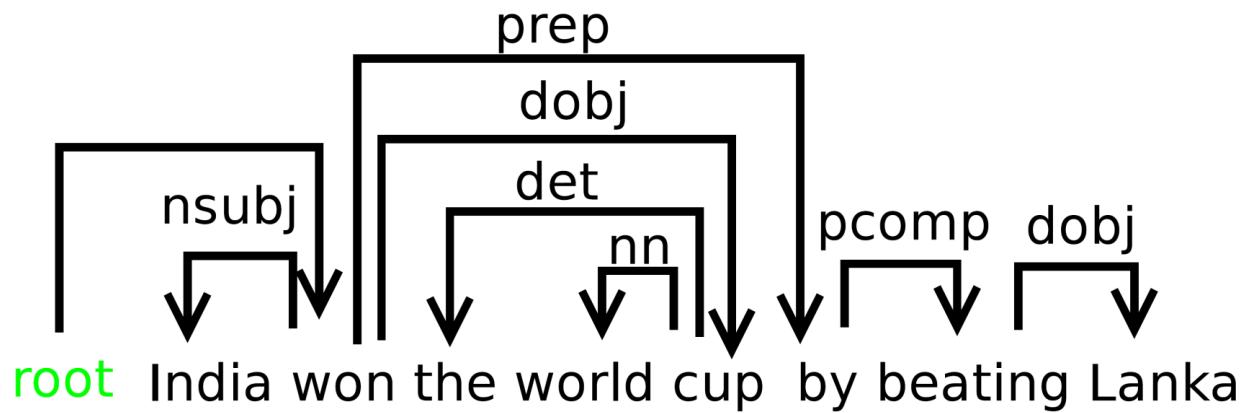
<http://universaldependencies.org/>

- Annotated treebanks in many languages
- Uniform annotation scheme across all languages:
  - Universal POS tags
  - Universal dependency relations

# Dependency Relations

Argument Dependencies	Description
<b>nsubj</b>	nominal subject
<b>csubj</b>	clausal subject
<b>dobj</b>	direct object
<b>iobj</b>	indirect object
<b>pobj</b>	object of preposition
Modifier Dependencies	Description
<b>tmod</b>	temporal modifier
<b>appos</b>	appositional modifier
<b>det</b>	determiner
<b>prep</b>	prepositional modifier

# Example Dependency Parse



# Method of Dependency Parsing

# Methods of Dependency Parsing

---

## 1. Dynamic programming (like in the CKY algorithm)

You can do it similarly to lexicalized PCFG parsing: an  $O(n^5)$  algorithm

Eisner (1996) gives a clever algorithm that reduces the complexity to  $O(n^3)$ , by producing parse items with heads at the ends rather than in the middle

## 2. Graph algorithms

You create a Maximum Spanning Tree for a sentence

McDonald et al.'s (2005) MSTParser scores dependencies independently using a ML classifier (he uses MIRA, for online learning, but it could be MaxEnt)

## 3. Constraint Satisfaction

Edges are eliminated that don't satisfy hard constraints. Karlsson (1990), etc.

## 4. Deterministic parsing

Greedy choice of attachments guided by machine learning classifiers

MaltParser (Nivre et al. 2008) – discussed in the next segment

---

# Deterministic parsing

# Deterministic parsing

## Basic idea

Derive a single syntactic representation (dependency graph) through a deterministic sequence of elementary parsing actions

## Configurations

A parser configuration is a triple  $c = (S, B, A)$ , where

- $S$  : a stack  $[ \dots, w_i ]_S$  of partially processed words,
- $B$  : a buffer  $[ w_j, \dots ]_B$  of remaining input words,
- $A$  : a set of labeled arcs  $(w_i, d, w_j)$ .

### Stack

$[\text{sent, her, a}]_S$

### Buffer

$[\text{letter, .}]_B$

### Arcs

$\text{He} \xleftarrow{\text{SBJ}} \text{sent}$

# Transition based systems for Dependency parsing

---

- Stack of partially processed words
- Input buffer
- Set of dependency arcs
- ✓ Attach the word on the top of the stack to the word at the current position in the buffer

- Intial configuration

- ✓ Stack (including the root token w0)
- ✓ Buffer(sentence)
- ✓ arcs(empty)

- Goal configuration

- ✓ Stack(empty)
- ✓ Buffer(empty)
- ✓ arcs(complete tree)

- Arc standard parser  
shift, left-arc, right-arc
- Arc –eager parser  
Shift , reduce, left arc ,right arc

# Arc eager parsing(Malt parser)

**Left-Arc( $d$ )**  $\frac{([\dots, w_i]_S, [w_j, \dots]_B, A)}{([\dots]_S, [w_j, \dots]_B, A \cup \{(w_j, d, w_i)\})} \neg \text{HEAD}(w_i)$

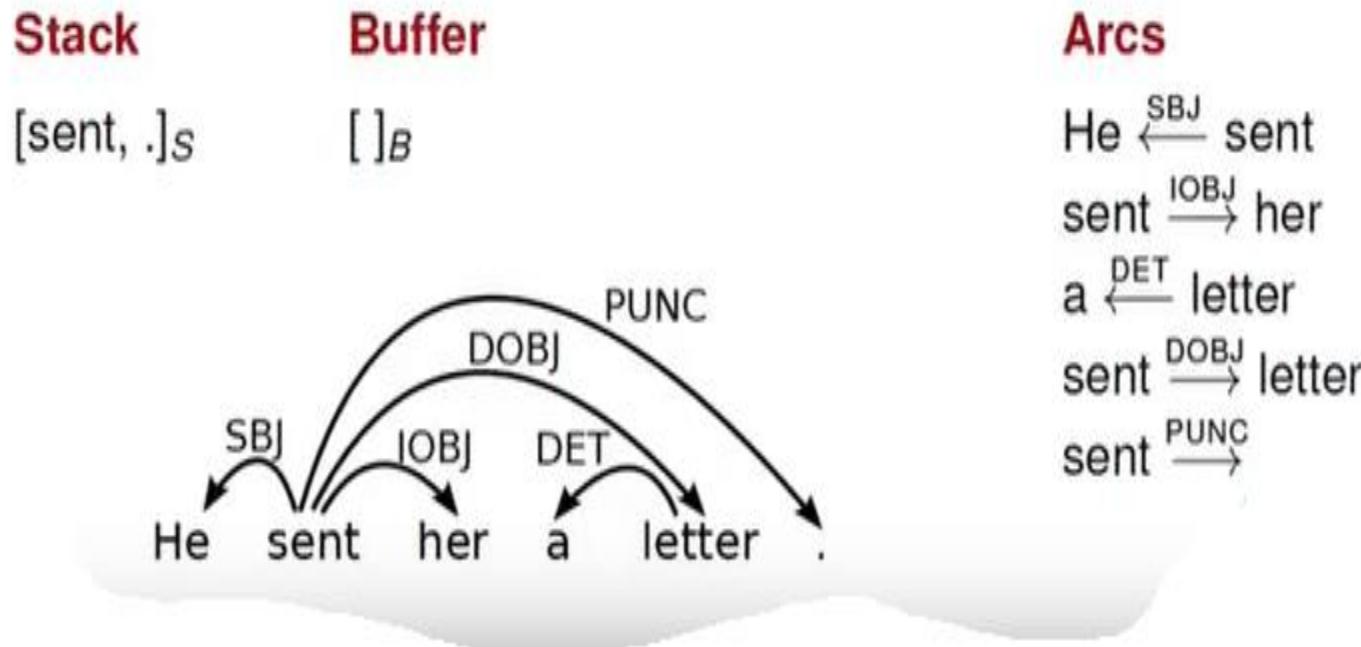
**Right-Arc( $d$ )**  $\frac{([\dots, w_i]_S, [w_j, \dots]_B, A)}{([\dots, w_i, w_j]_S, [\dots]_B, A \cup \{(w_i, d, w_j)\})}$

**Reduce**  $\frac{([\dots, w_i]_S, B, A)}{([\dots]_S, B, A)} \text{ HEAD}(w_i)$

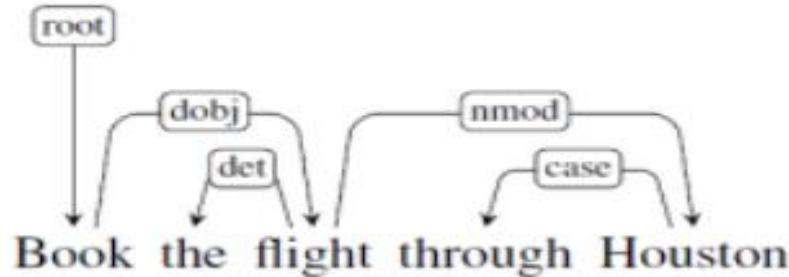
**Shift**  $\frac{([\dots]_S, [w_i, \dots]_B, A)}{([\dots, w_i]_S, [\dots]_B, A)}$

# Example1

**Transitions:** SH-LA-SH-RA-SH-LA-RE-RA-RE-RA



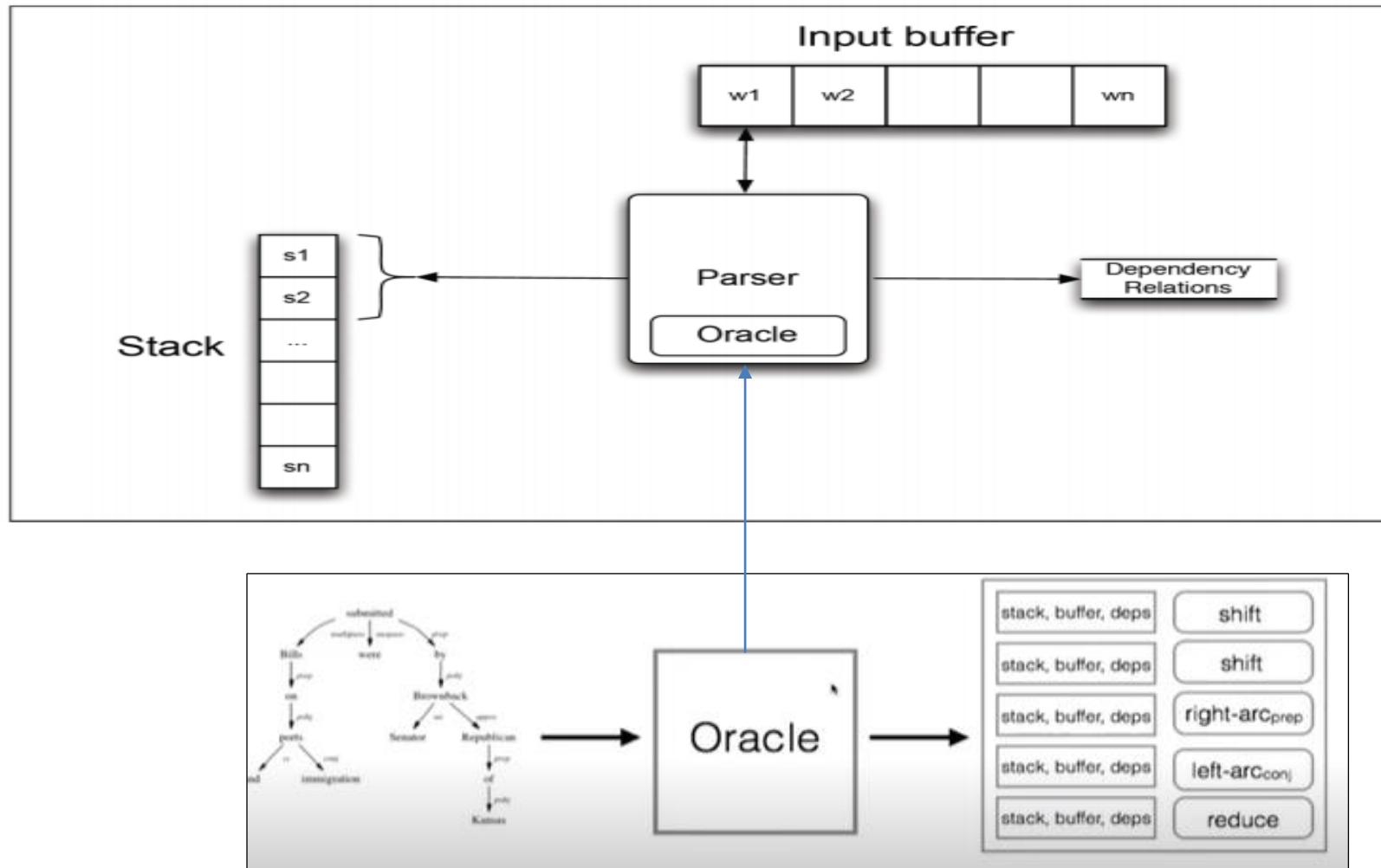
# Example2



Step	Stack	Word List	Action	Relation Added
0	[root]	[book, the, flight, through, houston]	RIGHTARC	(root → book)
1	[root, book]	[the, flight, through, houston]	SHIFT	
2	[root, book, the]	[flight, through, houston]	LEFTARC	(the ← flight)
3	[root, book]	[flight, through, houston]	RIGHTARC	(book → flight)
4	[root, book, flight]	[through, houston]	SHIFT	
5	[root, book, flight, through]	[houston]	LEFTARC	(through ← houston)
6	[root, book, flight]	[houston]	RIGHTARC	(flight → houston)
7	[root, book, flight, houston]	[]	REDUCE	
8	[root, book, flight]	[]	REDUCE	
9	[root, book]	[]	REDUCE	
10	[root]	[]	Done	



# Creating an Oracle



# How the classifier the learns ?

## *Learning Problem*

Approximate a function from **configurations**, represented by feature vectors to **transitions**, given a training set of gold standard **transition sequences**.

## *Three issues*

- How to represent configurations by feature vectors?
- How to derive training data from treebanks?
- How to learn classifiers?

# Feature Models

A feature representation  $f(c)$  of a configuration  $c$  is a vector of simple features  $f_i(c)$ .

## *Typical Features*

- Nodes:
  - Target nodes (top of  $S$ , head of  $B$ )
  - Linear context (neighbors in  $S$  and  $B$ )
  - Structural context (parents, children, siblings in  $G$ )
- Attributes:
  - Word form (and lemma)
  - Part-of-speech (and morpho-syntactic features)
  - Dependency type (if labeled)
  - Distance (between target tokens)

Feature template:

$$\langle s_1.w, op \rangle, \langle s_2.w, op \rangle \langle s_1.x, op \rangle, \langle s_2.x, op \rangle \\ \langle b_1.w, op \rangle, \langle b_1.x, op \rangle \langle s_1.wt, op \rangle$$

# Feature examples

Stack	Word buffer	Relations
[root, canceled, flights]	[to Houston]	(canceled → United) (flights → morning) (flights → the)

$\langle s_1.w = \text{flights}, op = \text{shift} \rangle$   
 $\langle s_2.w = \text{canceled}, op = \text{shift} \rangle$   
 $\langle s_1.x = \text{NNS}, op = \text{shift} \rangle$   
 $\langle s_2.x = \text{VBD}, op = \text{shift} \rangle$   
 $\langle b_1.w = \text{to}, op = \text{shift} \rangle$   
 $\langle b_1.x = \text{TO}, op = \text{shift} \rangle$   
 $\langle s_1.wt = \text{flightsNNS}, op = \text{shift} \rangle$

# Classifier at runtime

To guide the parser, a linear classifier can be used:

$$t^* = \arg \max_t w.f(c, t)$$

Weight vector  $w$  learned from treebank data.

## *Using classifier at run-time*

**PARSE**( $w_1, \dots, w_n$ )

- 1      $c \leftarrow ([], [w_1, \dots, w_n]_B, \{ \})$
- 2     **while**  $B_c \neq []$
- 3          $t^* \leftarrow \arg \max_t w.f(c, t)$
- 4          $c \leftarrow t^*(c)$
- 5     **return**  $T = (\{w_1, \dots, w_n\}, A_c)$

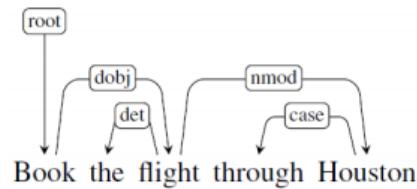
# Training Data

- Training instances have the form  $(f(c), t)$ , where
  - ▶  $f(c)$  is a feature representation of a configuration  $c$ ,
  - ▶  $t$  is the correct transition out of  $c$  (i.e.,  $o(c) = t$ ).
- Given a dependency treebank, we can sample the oracle function  $o$  as follows:
  - ▶ For each sentence  $x$  with gold standard dependency graph  $G_x$ , construct a transition sequence  $C_{0,m} = (c_0, c_1, \dots, c_m)$  such that
$$c_0 = c_s(x),$$
$$G_{c_m} = G_x$$
  - ▶ For each configuration  $c_i (i < m)$ , we construct a training instance  $(f(c_i), t_i)$ , where  $t_i(c_i) = c_{i+1}$ .



# Generating training data example

- What we have in a treebank



- What we need to train an oracle
  - Pairs of configurations and predicted parsing action

Step	Stack	Word List	Predicted Action
0	[root]	[book, the, flight, through, houston]	SHIFT
1	[root, book]	[the, flight, through, houston]	SHIFT
2	[root, book, the]	[flight, through, houston]	SHIFT
3	[root, book, the, flight]	[through, houston]	LEFTARC
4	[root, book, flight]	[through, houston]	SHIFT
5	[root, book, flight, through]	[houston]	SHIFT
6	[root, book, flight, through, houston]	[]	LEFTARC
7	[root, book, flight, houston ]	[]	RIGHTARC
8	[root, book, flight]	[]	RIGHTARC
9	[root, book]	[]	RIGHTARC
10	[root]	[]	Done

# Standard Oracle for Arc Eager parsing



$o(c, T) =$

- **Left-Arc** if  $\text{top}(S_c) \leftarrow \text{first}(B_c)$  in  $T$
- **Right-Arc** if  $\text{top}(S_c) \rightarrow \text{first}(B_c)$  in  $T$
- **Reduce** if  $\exists w < \text{top}(S_c) : w \leftrightarrow \text{first}(B_c)$  in  $T$
- **Shift** otherwise

# Online learning with an oracle

```
LEARN({ $T_1, \dots, T_N$ })
1    $w \leftarrow 0.0$ 
2   for  $i$  in  $1..K$ 
3     for  $j$  in  $1..N$ 
4        $c \leftarrow ([], [w_1, \dots, w_{n_j}]_B, \{\})$ 
5       while  $B_c \neq []$ 
6          $t^* \leftarrow \arg \max_t w.f(c, t)$ 
7          $t_o \leftarrow o(c, T_i)$ 
8         if  $t^* \neq t_o$ 
9            $w \leftarrow w + f(c, t_o) - f(c, t^*)$ 
10           $c \leftarrow t_o(c)$ 
11    return  $w$ 
```

Oracle  $o(c, T_i)$  returns the optimal transition of  $c$  and  $T_i$

# Example

Consider the sentence, ‘John saw Mary’.

Draw a dependency graph for this sentence.

Assume that you are learning a classifier for the data-driven deterministic parsing and the above sentence is a gold-standard parse in your training data. You are also given that *John* and *Mary* are ‘Nouns’, while the POS tag of *saw* is ‘Verb’. Assume that your features correspond to the following conditions:

- ▶ The stack is empty
- ▶ Top of stack is Noun and Top of buffer is Verb
- ▶ Top of stack is Verb and Top of buffer is Noun

Initialize the weights of all your features to 5.0, except that in all of the above cases, you give a weight of 5.5 to *Left-Arc*. Define your feature vector and the initial weight vector.

Use this gold standard parse during online learning and report the weights after completing one full iteration of Arc-Eager parsing over this sentence.

$F(c,t) = [(c_0, LA), (c_1, LA), (c_2, LA) | (c_0, RA), (c_1, RA), (c_2, RA) \dots]$

$W = [5.5, 5.5, 5.5 | 5.0, 5.0, 5.0 | 5.0, 5.0, 5.0 | \dots]$

So for the given conditions

$F(c, LA) = [1, 0, 0 | 0, 0, 0 | \dots]$

$F(c, RA) = [0, 0, 0 | 1, 0, 0 | \dots]$

$t^* = \text{argmax}(w^* f(c, t))$

$t^* = LA$

as per oracle /optimal transition  $t^0 = SH$

So we need to update the weights

To update the weights

$W = W + f(c, t^0) - f(c, t^*)$

$W = [5.5, 5.0, 5.0 | 5.0, 5.0 \dots] + [0, 0, 0 | 0, 0, 0 | \dots 1, 0, 0] - [1, 0, 0 | 0, 0, 0 | \dots]$

New vector =  $[4.5, 5.5, 5.5 | 5.0 \dots 6.0, 5.0, 5.0]$

Now

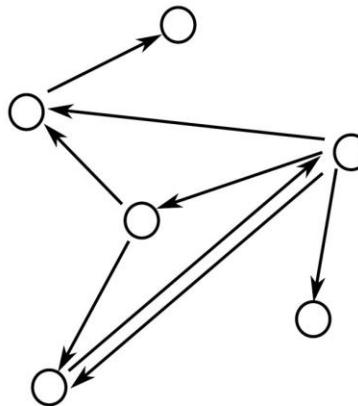
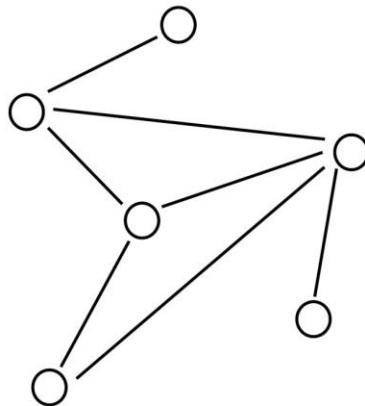
John saw Mary

Next configuration

# Graph-based parsing

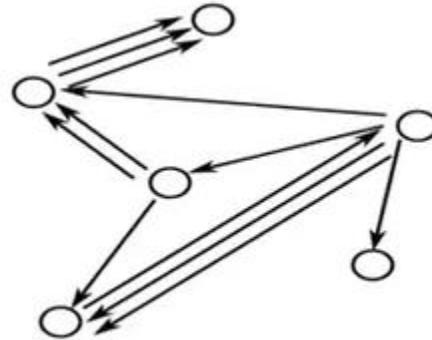
# Graph concepts refresher

- ▶ A graph  $G = (V, A)$  is a set of vertices  $V$  and arcs  $(i, j) \in A$ , where  $i, j \in V$
- ▶ Undirected graphs:  $(i, j) \in A \Leftrightarrow (j, i) \in A$
- ▶ **Directed graphs (digraphs):**  $(i, j) \in A \not\Leftrightarrow (j, i) \in A$



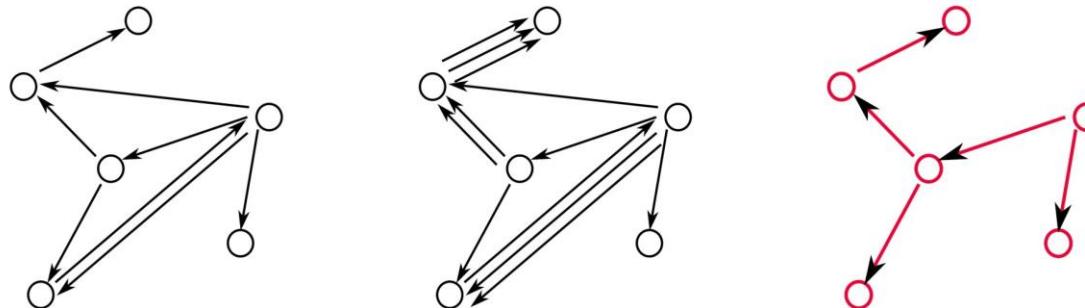
# Multi Digraph

- A multi-digraph is a digraph where multiple arcs between vertices are possible
- $(i,j,k) \in A$  represents the  $k^{th}$  arc from vertex  $i$  to vertex  $j$ .



# Spanning Trees

- ▶ A directed spanning tree of a (multi-)digraph  $G = (V, A)$ , is a subgraph  $G' = (V', A')$  such that:
  - ▶  $V' = V$
  - ▶  $A' \subseteq A$ , and  $|A'| = |V'| - 1$
  - ▶  $G'$  is a tree (acyclic)
- ▶ A spanning tree of the following (multi-)digraphs



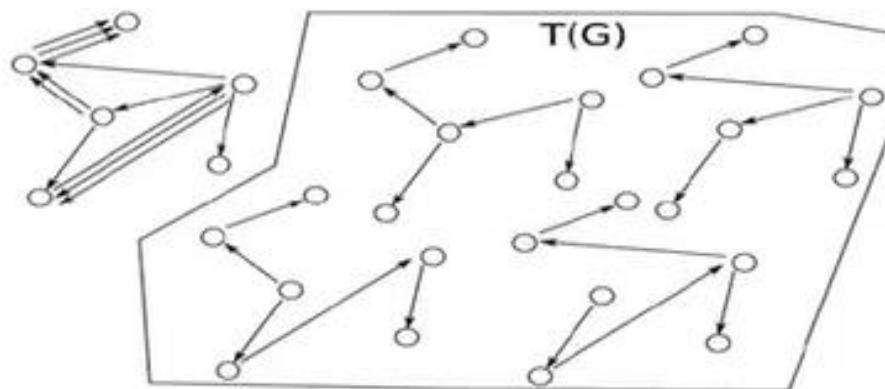
# Weighted Spanning tree

- Assume we have a weight function for each arc in a multi-digraph  $G = (V, A)$ .
- Define  $w_{ij}^k \geq 0$  to be the weight of  $(i, j, k) \in A$  for a multi-digraph
- Define the weight of directed spanning tree  $G'$  of graph  $G$  as

$$w(G') = \sum_{(i,j,k) \in G'} w_{ij}^k$$

# MST

Let  $T(G)$  be the set of all spanning trees for graph  $G$



The MST problem

Find the spanning tree  $G'$  of the graph  $G$  that has the highest weight

$$G' = \arg \max_{G' \in T(G)} w(G') = \arg \max_{G' \in T(G)} \sum_{(i,j,k) \in G'} w_{ij}^k$$

# Finding MST

## Directed Graph

For each sentence  $x$ , define the directed graph  $G_x = (V_x, E_x)$  given by

$$V_x = \{x_0 = \text{root}, x_1, \dots, x_n\}$$

$$E_x = \{(i, j) : i \neq j, (i, j) \in [0 : n] \times [1 : n]\}$$

$G_x$  is a graph with

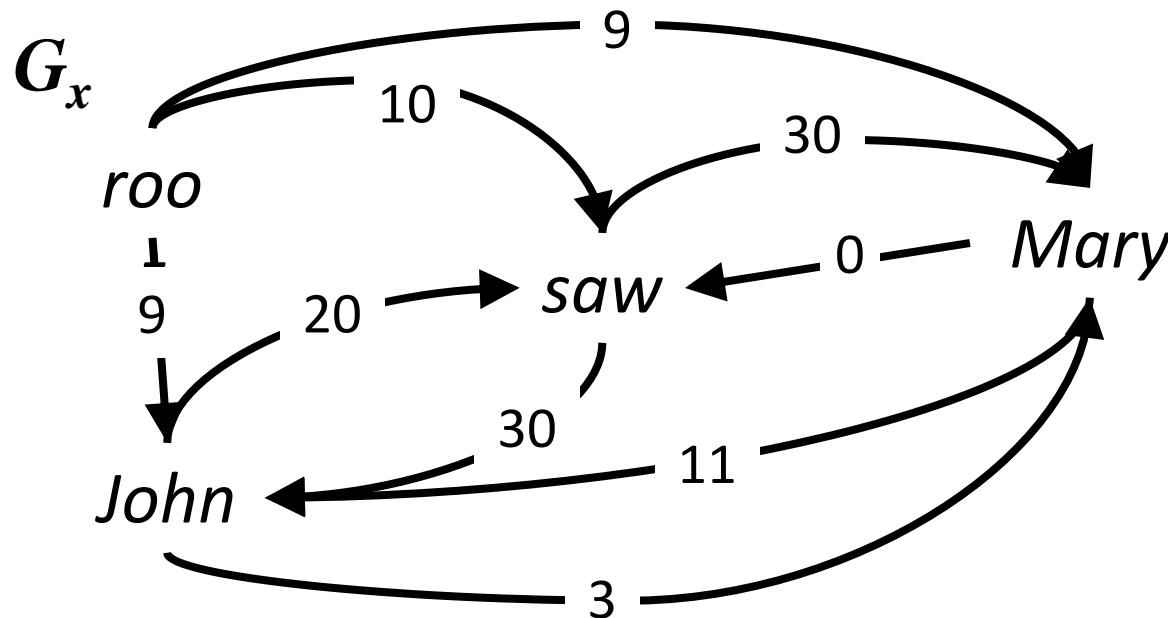
- the sentence words and the dummy root symbol as vertices and
- a directed edge between every pair of distinct words and
- a directed edge from the root symbol to every word

# Chu-Liu-Edmonds algorithm

- Each vertex in the graph greedily selects the incoming edge with the highest weight.
- If a tree results, it must be a maximum spanning tree.
- If not, there must be a cycle.
  - Identify the cycle and contract it into a single vertex.
  - Recalculate edge weights going into and out of the cycle.

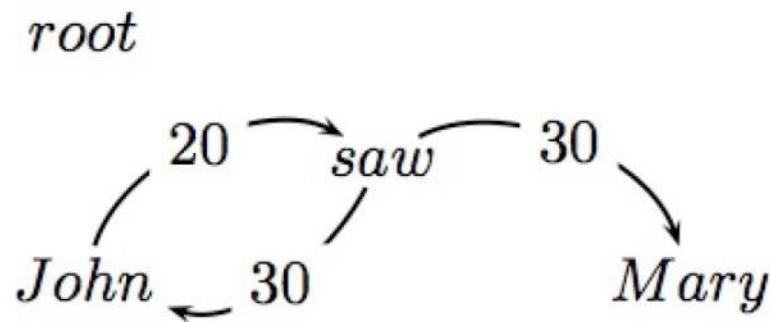
# Chu-Liu-Edmonds Algorithm (2/12)

- $x = John \ saw \ Mary$



# Chu-Liu-Edmonds Example

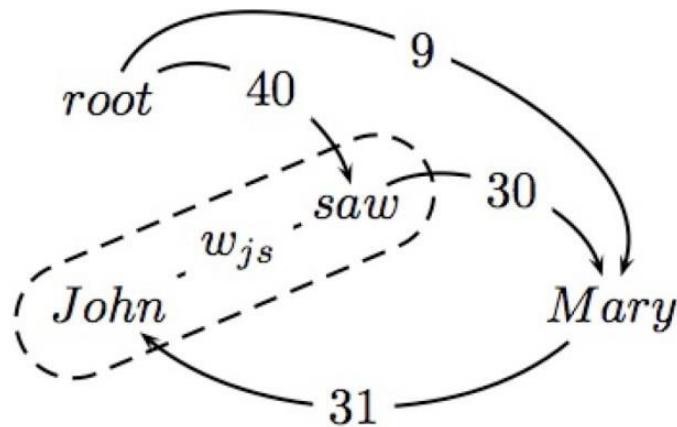
- ▶ Find highest scoring incoming arc for each vertex



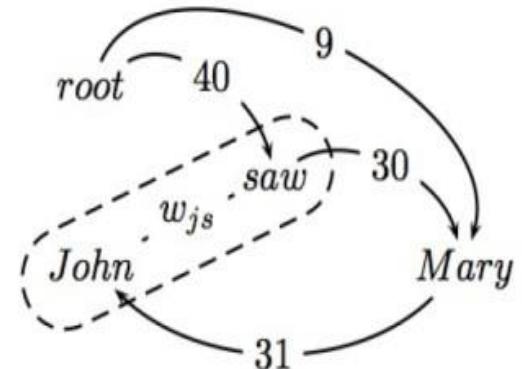
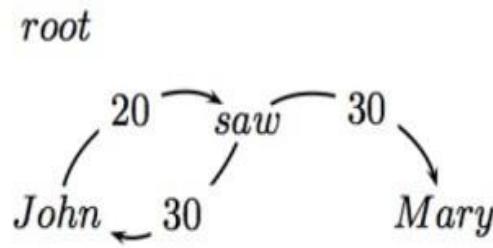
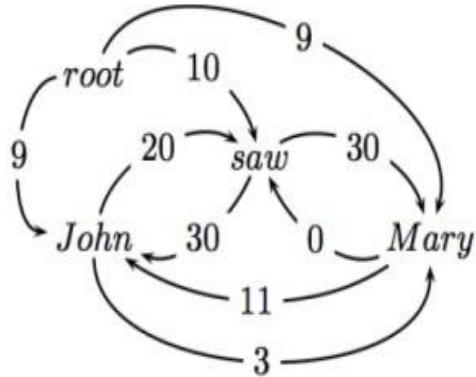
- ▶ If this is a tree, then we have found MST!!

# Chu-Liu-Edmonds Example

- ▶ If not a tree, identify cycle and contract
- ▶ Recalculate arc weights into and out-of cycle

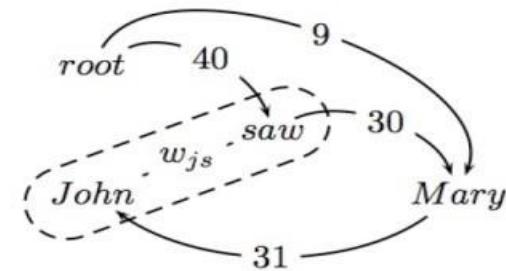
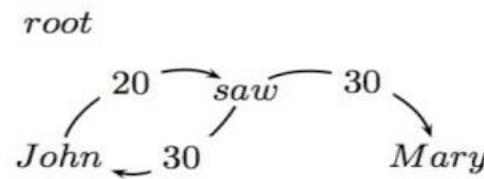
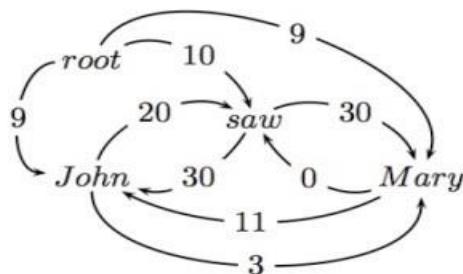


# Chu-Liu-Edmonds Example



- ▶ Outgoing arc weights
  - ▶ Equal to the max of outgoing arc over all vertexes in cycle
  - ▶ e.g., John → Mary is 3 and saw → Mary is 30

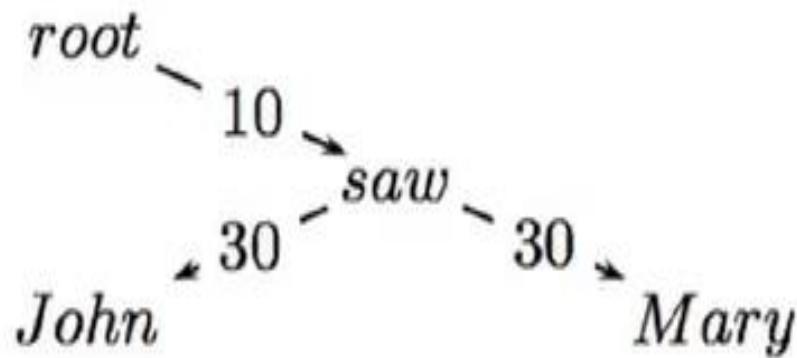
# Chu-Liu-Edmonds Example



## ► Incoming arc weights

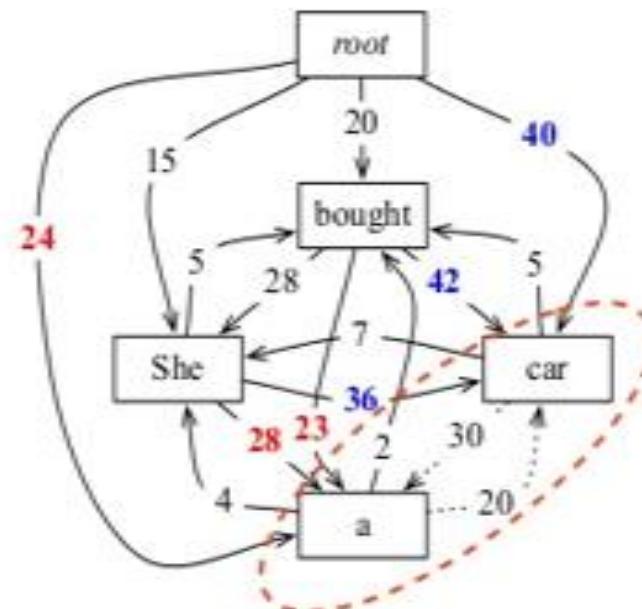
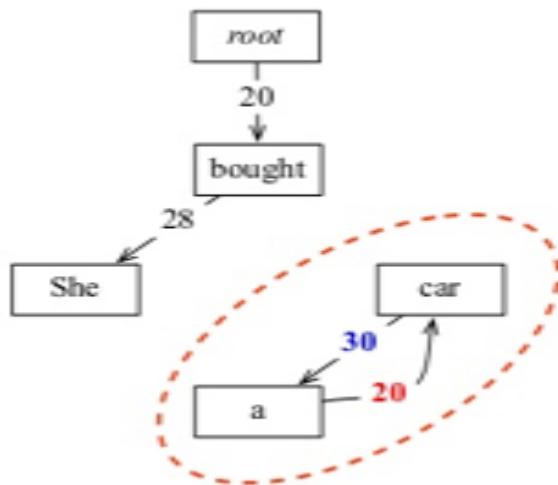
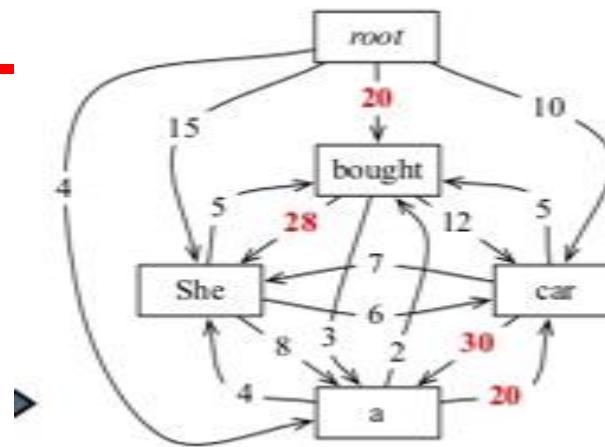
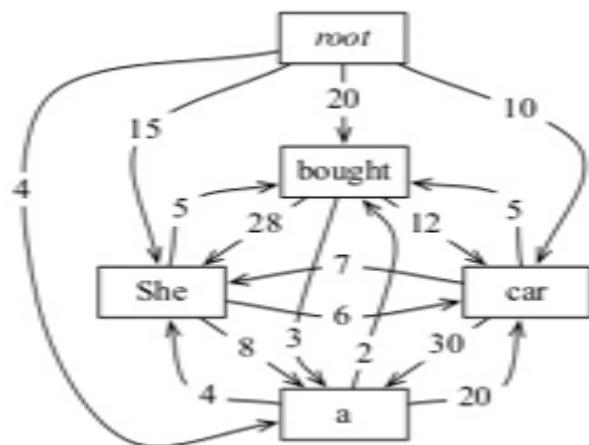
- Equal to the weight of best spanning tree that includes head of incoming arc, and all nodes in cycle
- $\text{root} \rightarrow \text{saw} \rightarrow \text{John}$  is 40 (\*\*)
- $\text{root} \rightarrow \text{John} \rightarrow \text{saw}$  is 29

# Chu-Liu-Edmonds Example



- The edge from  $w_{js}$  to *Mary* was from *saw*
- The edge from *root* to  $w_{js}$  represented a tree from *root* to *saw* to *John*.

# Example2







# MST Learning

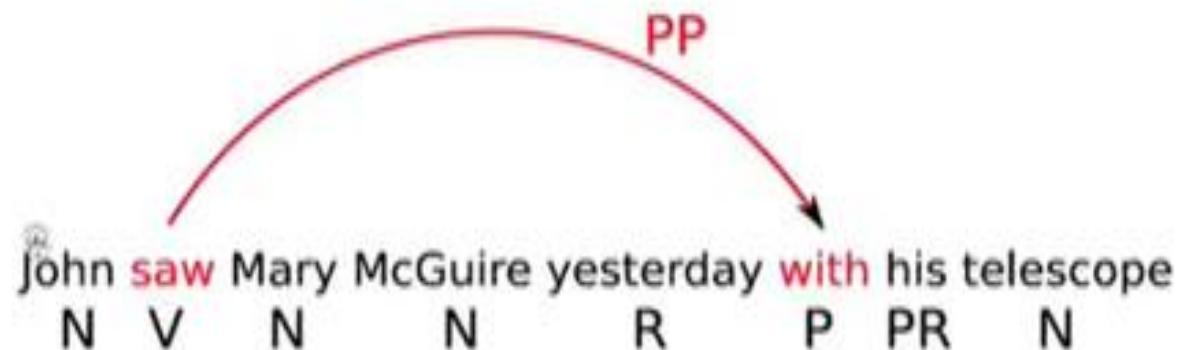
# Linear classifiers

10

$$w_{ij}^k = w.f(i,j,k)$$

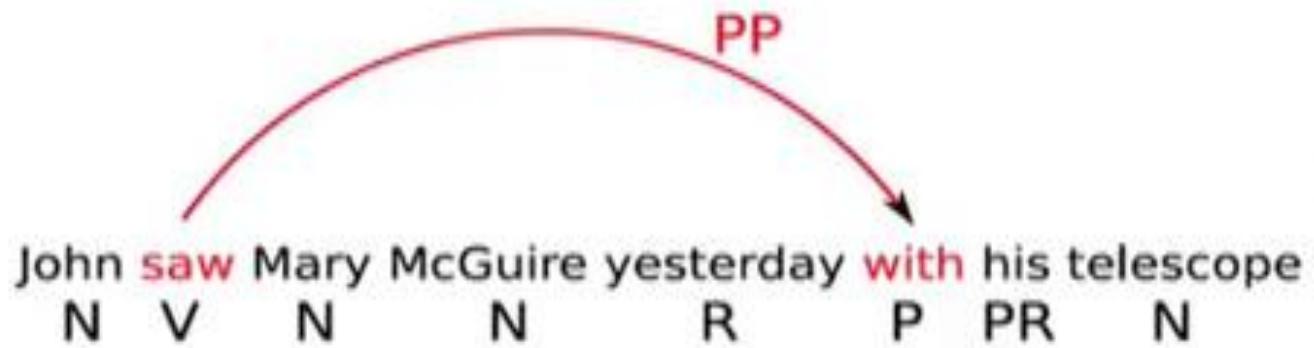
- Arc weights are a linear combination of features of the arc  $f(i,j,k)$  and a corresponding weight vector  $w$
- What arc features?

# Arch features



## Features

Identities of the words  $w_i$  and  $w_j$  for a label  $l_k$   
head = saw & dependent=with



## Features

Part-of-speech tags of the words  $w_i$  and  $w_j$  for a label  $l_k$   
head-pos = Verb & dependent-pos=Preposition



### Features

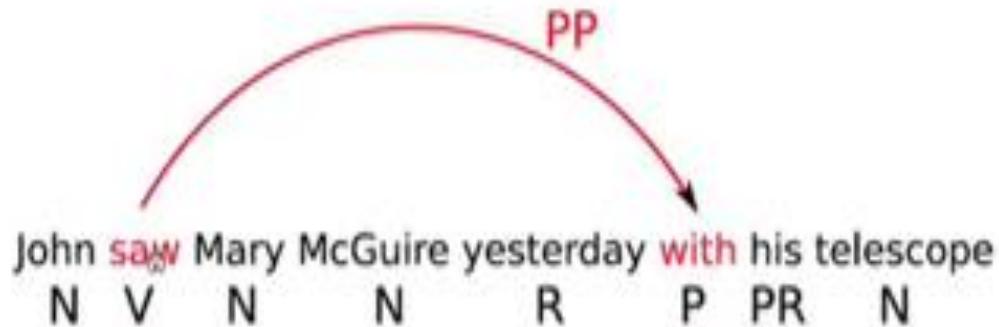
Part-of-speech of words surrounding and between  $w_i$  and  $w_j$

inbetween-pos = Noun

inbetween-pos = Adverb

dependent-pos-right = Pronoun

head-pos-left=Noun



### Features

Number of words between  $w_i$  and  $w_j$ , and their orientation

arc-distance = 3

arc-direction = right

# Learning the parameters

- Re-write the inference problem

$$G = \arg \max_{G \in T(G_x)} \sum_{(i,j,k) \in G} {w_{ij}}^k$$

$$= \arg \max_{G \in T(G_x)} w \cdot \sum_{(i,j,k) \in G} f(i,j,k)$$

$$= \arg \max_{G \in T(G_x)} w \cdot f(G)$$

# Inference based learning

Training data:  $T = \{(x_t, G_t)\}_{t=1}^{|T|}$

1.  $w^{(0)} = 0; i = 0$
2. **for**  $n : 1..N$
3.     **for**  $t : 1..|T|$
4.         Let  $G' = argmax_{G'} w^{(i)}.f(G')$
5.         if  $G' \neq G_t$
6.              $w^{(i+1)} = w^{(i)} + f(G_t) - f(G')$
7.          $i = i + 1$
8.     **return**  $w^i$

# Example

Suppose you are training MST Parser for dependency and the sentence, "John saw Mary" occurs in the training set. Also, for simplicity, assume that there is only one dependency relation, "rel". Thus, for every arc from word  $w_i$  to  $w_j$ , your features may be simplified to depend only on words  $w_i$  and  $w_j$  and not on the relation label.

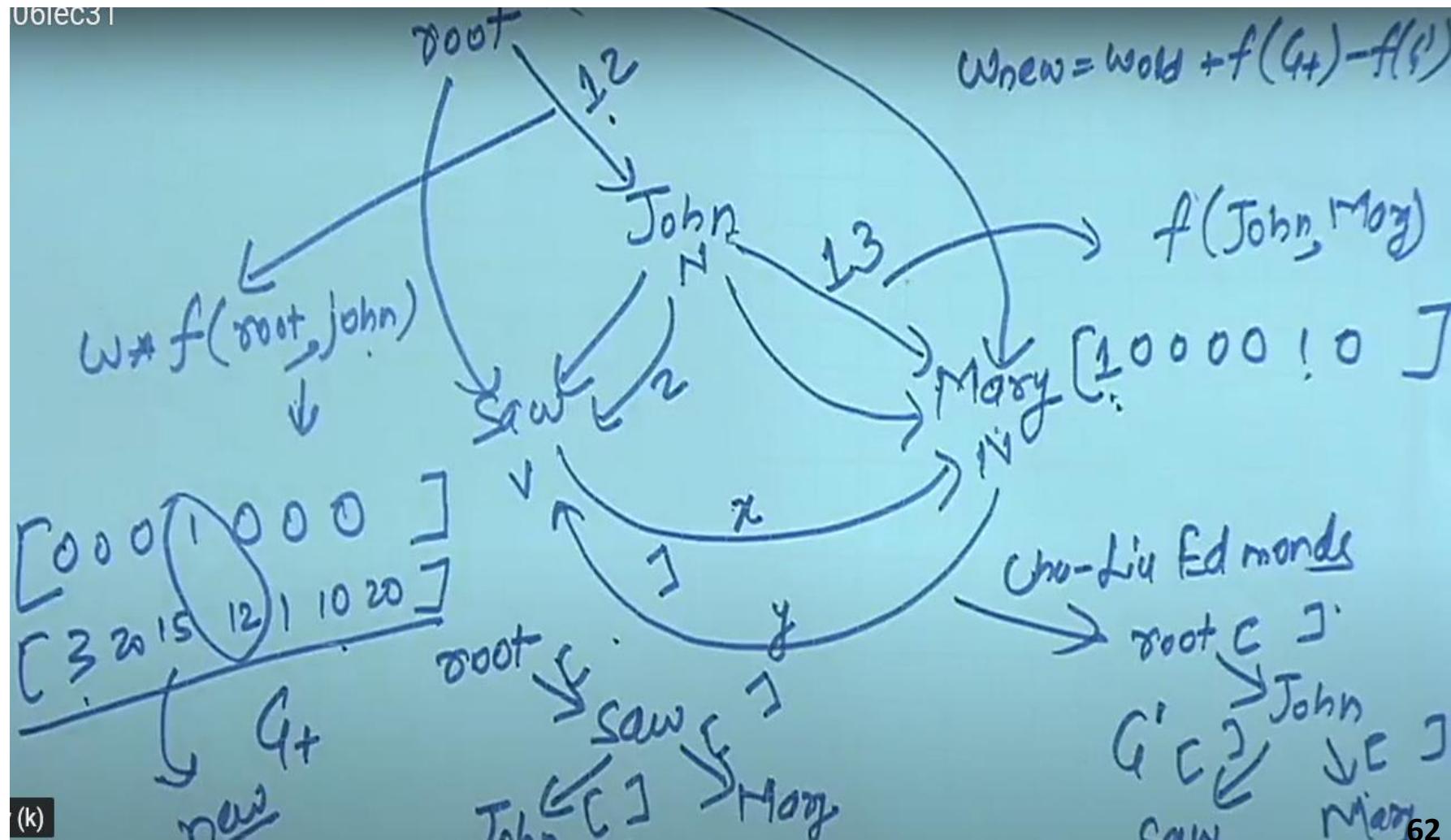
Below is the set of features

- $f_1: \text{pos}(w_i) = \text{Noun}$  and  $\text{pos}(w_j) = \text{Noun}$
- $f_2: \text{pos}(w_i) = \text{Verb}$  and  $\text{pos}(w_j) = \text{Noun}$
- $f_3: w_i = \text{Root}$  and  $\text{pos}(w_j) = \text{Verb}$
- $f_4: w_i = \text{Root}$  and  $\text{pos}(w_j) = \text{Noun}$
- $f_5: w_i = \text{Root}$  and  $w_j$  occurs at the end of sentence
- $f_6: w_i$  occurs before  $w_j$  in the sentence
- $f_7: \text{pos}(w_i) = \text{Noun}$  and  $\text{pos}(w_j) = \text{Verb}$

The feature weights before the start of the iteration are: {3, 20, 15, 12, 1, 10, 20}.  
Determine the weights after an iteration over this example.

# EXAMPLE

Topic 31



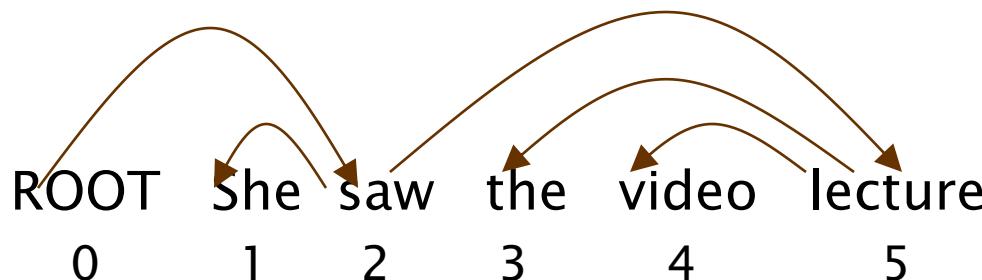


# Evaluation

---

- Unlabeled Attachment Score (UAS), which corresponds to the number of correctly predicted dependencies over the number of possibilities;
  - Labeled Attachment Score (LAS), which corresponds to the number of correctly predicted dependencies and relations over the number of possibilities.

# Evaluation



$$\text{Acc} = \frac{\# \text{ correct deps}}{\# \text{ of deps}}$$

UAS(unlabeled attachment score) =  
LAS ( Labeled Attachment Score) = 2

## Gold

1	2	She	nsubj
2	0	saw	root
3	5	the	det
4	5	video	nn
5	2	lecture	dobj

## Parsed

1	2	She	nsubj
2	0	saw	root
3	4	the	det
4	5	video	nsubj
5	2	lecture	ccomp

# References



- 
1. Speech and Language processing: An introduction to Natural Language Processing, Computational Linguistics and speech Recognition by Daniel Jurafsky and James H. Martin[3rd edition].
  2. <https://www.youtube.com/watch?v=PVShkZgXznc>
  3. <https://www.youtube.com/watch?v=02QWRAhGc7g&list=PLJJzI13YAXCHxbVgiFaSI88hj-mRSoMtl>
  4. [https://www.researchgate.net/publication/328731166\\_Weighted\\_Machine\\_Learning](https://www.researchgate.net/publication/328731166_Weighted_Machine_Learning)

---

Any Questions?

---

# Thank you



# Natural Language Processing

## DSECL ZG565

Prof.Vijayalakshmi  
BITS-Pilani

**BITS** Pilani  
Pilani Campus





## **Dependency Parsing**

**Date – 9/3/2024**

**Time – 1.40 to 3.40pm**

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Jurafsky and Prof. Martin and many others who made their course materials freely available online.

# Outline

- ❖ Motivation
- ❖ Two types of parsing
  - Dependency parsing
  - Phrase structure parsing
- ❖ Dependency structure and Dependency grammar
- ❖ Dependency Relation
- ❖ Universal Dependencies
- ❖ Method of Dependency Parsing
  - Dynamic programming
  - Graph algorithms
  - Constraint satisfaction
  - Deterministic Parsing
- ❖ Transition based dependency parsing
- ❖ Graph based dependency Parsing
- ❖ Evaluation

# Interpreting Language is Hard!

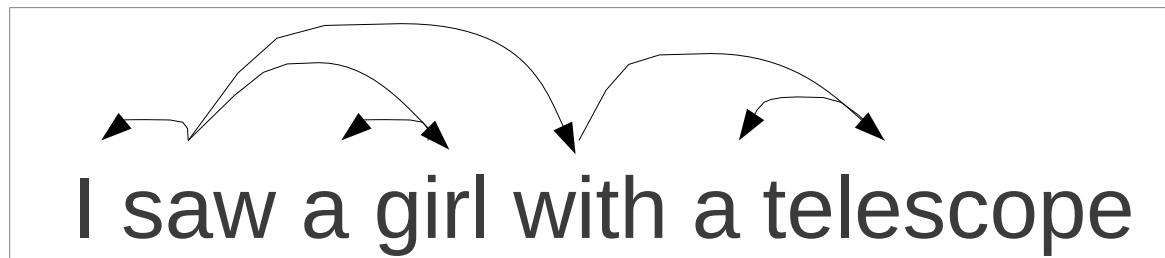
I saw a girl with a telescope



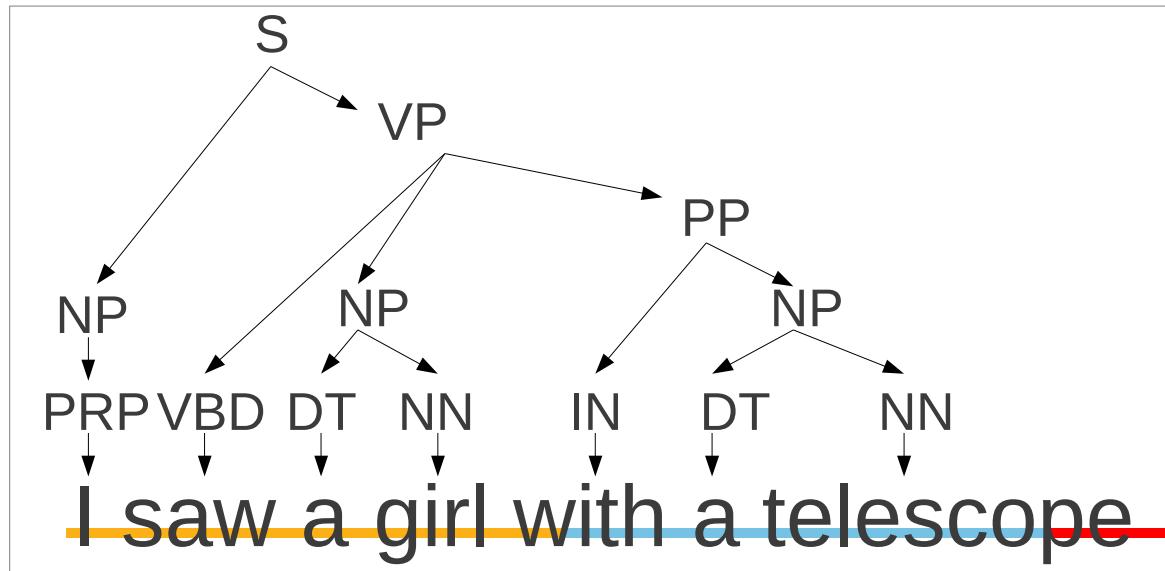
- “Parsing” resolves structural ambiguity in a formal way

## Two Types of Parsing

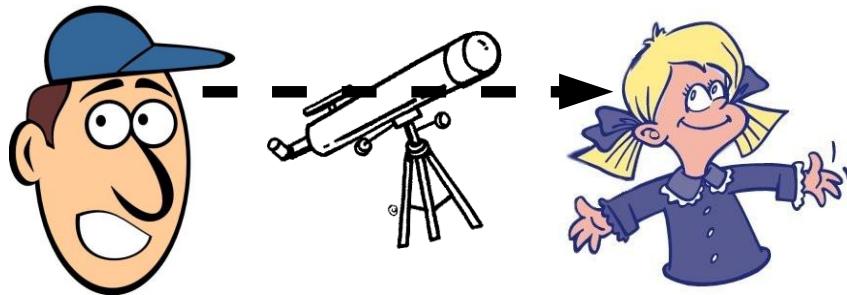
- **Dependency:** focuses on relations between words



- **Phrase structure:** focuses on identifying phrases and their recursive structure



# Dependencies Also Resolve Ambiguity



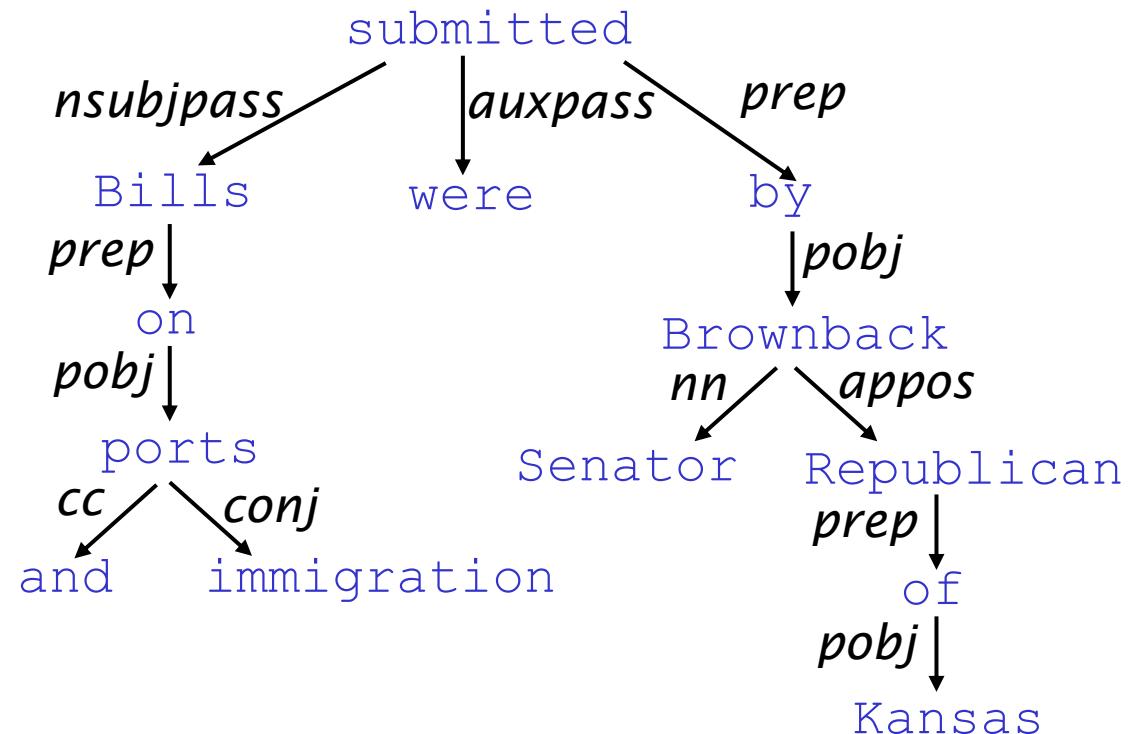
I saw a girl with a telescope

I saw a girl with a telescope

# Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of lexical items linked by binary asymmetric relations (“arrows”) called dependencies

The arrows are commonly **typed** with the name of grammatical relations (subject, prepositional object, apposition, etc.)

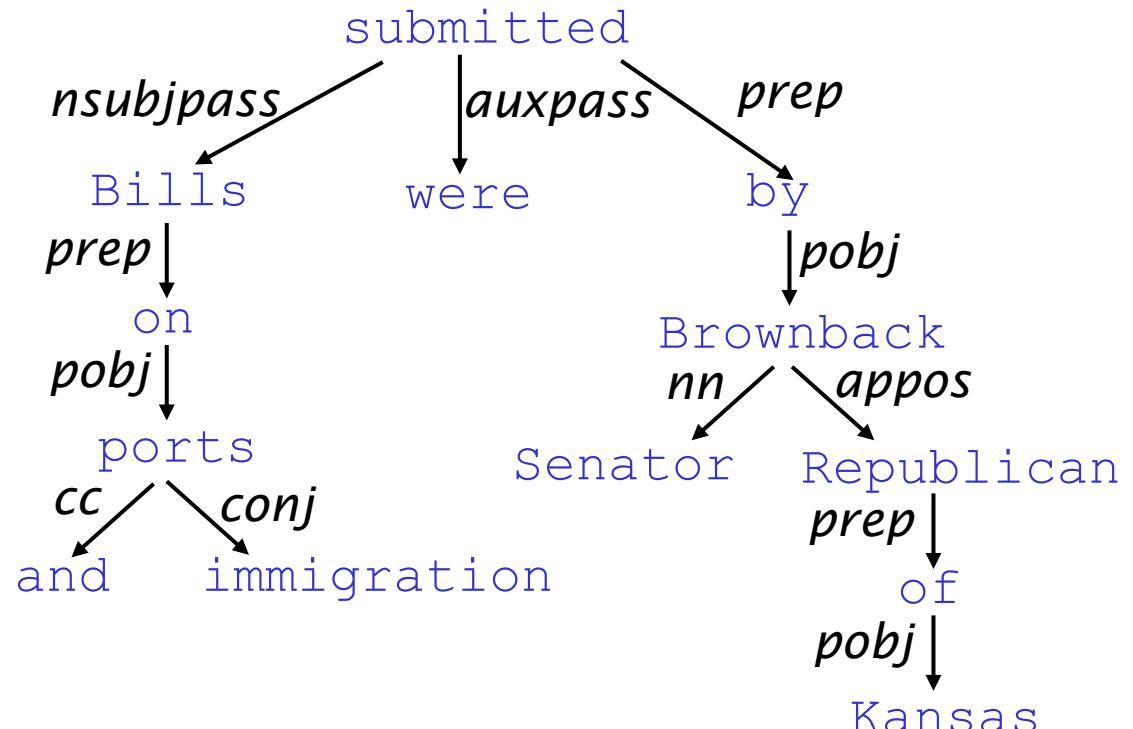


# Dependency Grammar and Dependency Structure



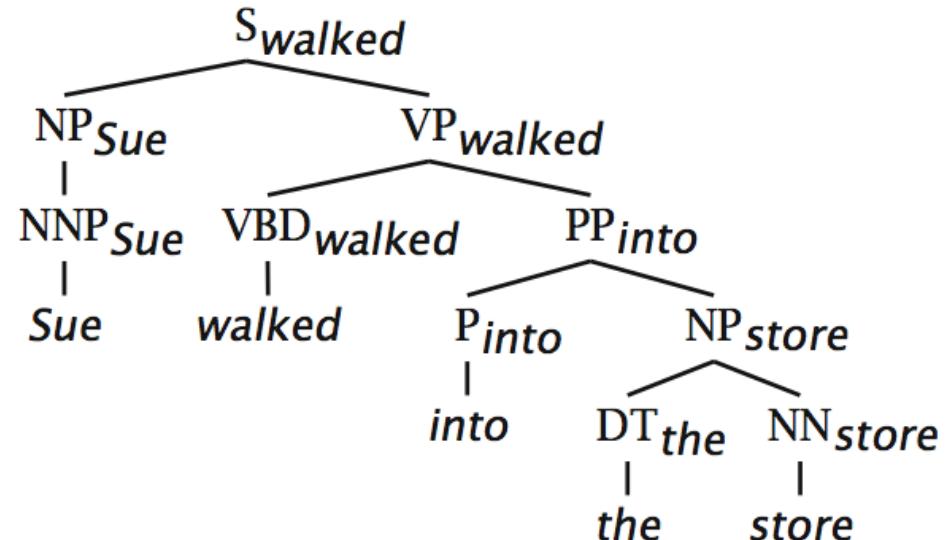
The arrow connects a **head** (governor, superior, regent) with a **dependent** (modifier, inferior, subordinate)

Usually, dependencies form a tree (connected, acyclic, single-head)



# Relation between phrase structure and dependency structure

- A dependency grammar has a notion of a head. Officially, CFGs don't.
- But modern linguistic theory and all modern statistical parsers (Charniak, Collins, Stanford, ...) do, via hand-written phrasal “head rules”:
  - The head of a Noun Phrase is a noun/number/adj/...
  - The head of a Verb Phrase is a verb/modal/....
- The head rules can be used to extract a dependency parse from a CFG parse
- The closure of dependencies give constituency from a dependency tree
- But the dependents of a word must be at the same level (i.e., “flat”)



# Dependency graph

- A dependency structure can be defined as a directed graph  $G$ , consisting of
  - ▶ a set  $V$  of nodes,
  - ▶ a set  $A$  of arcs (edges),
- Labeled graphs:
  - ▶ Nodes in  $V$  are labeled with word forms (and annotation).
  - ▶ Arcs in  $A$  are labeled with dependency types.
- Notational convention:
  - ▶ Arc  $(w_i, d, w_j)$  links head  $w_i$  to dependent  $w_j$  with label  $d$
  - ▶  $w_i \xrightarrow{d} w_j \Leftrightarrow (w_i, d, w_j) \in A$
  - ▶  $i \rightarrow j \equiv (i, j) \in A$

# Formal conditions on dependency graph



- $G$  is connected:
  - For every node  $i$  there is a node  $j$  such that  $i \rightarrow j$  or  $j \rightarrow i$ .
- $G$  is acyclic:
  - if  $i \rightarrow j$  then not  $j \rightarrow^* i$ .
- $G$  obeys the single head constraint:
  - if  $i \rightarrow j$  then not  $k \rightarrow j$ , for any  $k \neq i$ .
- $G$  is projective:
  - if  $i \rightarrow j$  then  $j \rightarrow^* k$ , for any  $k$  such that both  $j$  and  $k$  lie on the same side of  $i$ .

# Universal dependencies

---

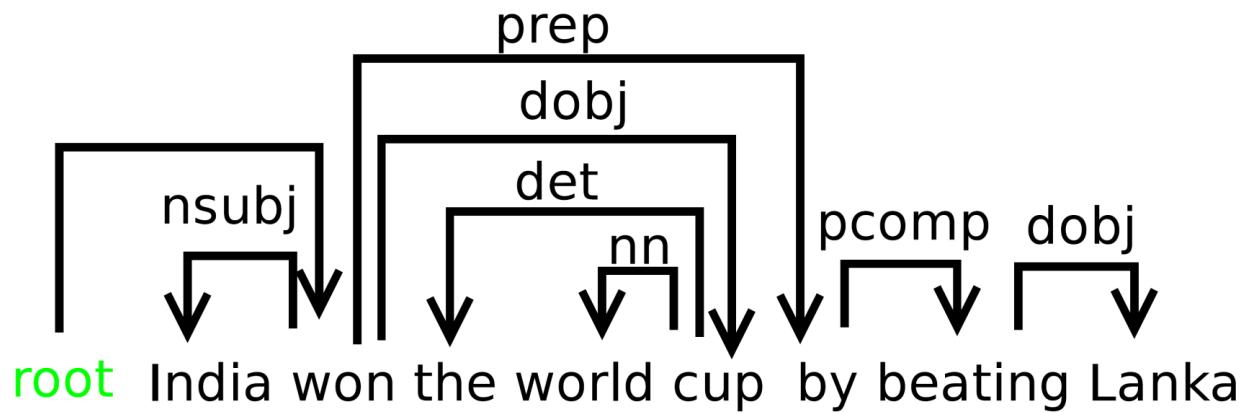
<http://universaldependencies.org/>

- Annotated treebanks in many languages
- Uniform annotation scheme across all languages:
  - Universal POS tags
  - Universal dependency relations

# Dependency Relations

Argument Dependencies	Description
<b>nsubj</b>	nominal subject
<b>csubj</b>	clausal subject
<b>dobj</b>	direct object
<b>iobj</b>	indirect object
<b>pobj</b>	object of preposition
Modifier Dependencies	Description
<b>tmod</b>	temporal modifier
<b>appos</b>	appositional modifier
<b>det</b>	determiner
<b>prep</b>	prepositional modifier

# Example Dependency Parse



# Method of Dependency Parsing

# Methods of Dependency Parsing

---

## 1. Dynamic programming (like in the CKY algorithm)

You can do it similarly to lexicalized PCFG parsing: an  $O(n^5)$  algorithm

Eisner (1996) gives a clever algorithm that reduces the complexity to  $O(n^3)$ , by producing parse items with heads at the ends rather than in the middle

## 2. Graph algorithms

You create a Maximum Spanning Tree for a sentence

McDonald et al.'s (2005) MSTParser scores dependencies independently using a ML classifier (he uses MIRA, for online learning, but it could be MaxEnt)

## 3. Constraint Satisfaction

Edges are eliminated that don't satisfy hard constraints. Karlsson (1990), etc.

## 4. Deterministic parsing

Greedy choice of attachments guided by machine learning classifiers

MaltParser (Nivre et al. 2008) – discussed in the next segment

---

# Deterministic parsing

# Deterministic parsing

## Basic idea

Derive a single syntactic representation (dependency graph) through a deterministic sequence of elementary parsing actions

## Configurations

A parser configuration is a triple  $c = (S, B, A)$ , where

- $S$  : a stack  $[ \dots, w_i ]_S$  of partially processed words,
- $B$  : a buffer  $[ w_j, \dots ]_B$  of remaining input words,
- $A$  : a set of labeled arcs  $(w_i, d, w_j)$ .

### Stack

$[\text{sent}, \text{her}, \text{a}]_S$

### Buffer

$[\text{letter}, .]_B$

### Arcs

$\text{He} \xleftarrow{\text{SBJ}} \text{sent}$

# Transition based systems for Dependency parsing

---

- Stack of partially processed words
- Input buffer
- Set of dependency arcs
- ✓ Attach the word on the top of the stack to the word at the current position in the buffer

- Intial configuration

- ✓ Stack (including the root token w0)
- ✓ Buffer(sentence)
- ✓ arcs(empty)

- Goal configuration

- ✓ Stack(empty)
- ✓ Buffer(empty)
- ✓ arcs(complete tree)

- Arc standard parser  
shift, left-arc, right-arc
- Arc –eager parser  
Shift , reduce, left arc ,right arc

# Arc eager parsing(Malt parser)

Left-Arc( $d$ )  $\frac{([\dots, w_i]_S, [w_j, \dots]_B, A)}{([\dots]_S, [w_j, \dots]_B, A \cup \{(w_j, d, w_i)\})} \neg \text{HEAD}(w_i)$

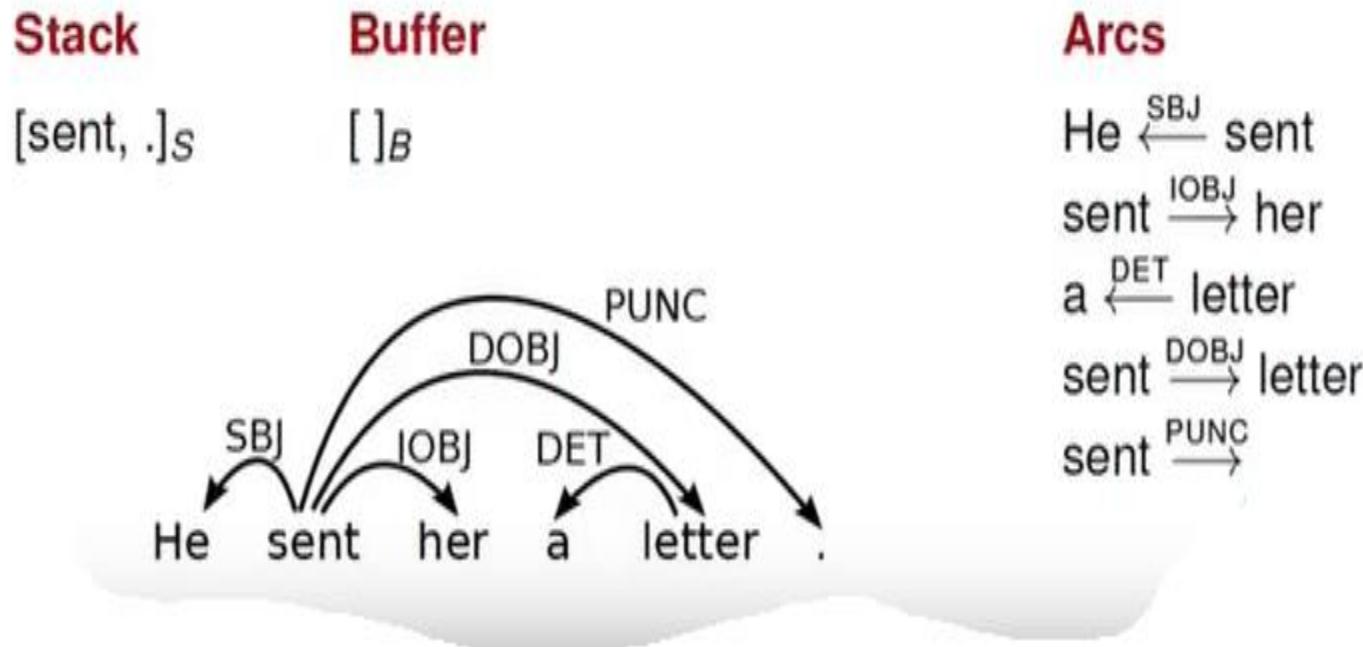
Right-Arc( $d$ )  $\frac{([\dots, w_i]_S, [w_j, \dots]_B, A)}{([\dots, w_i, w_j]_S, [\dots]_B, A \cup \{(w_i, d, w_j)\})}$

Reduce  $\frac{([\dots, w_i]_S, B, A)}{([\dots]_S, B, A)} \text{HEAD}(w_i)$

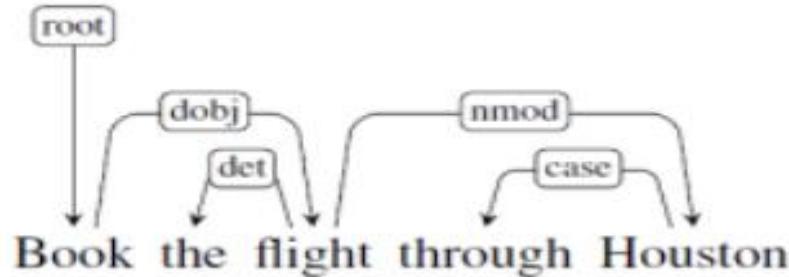
Shift  $\frac{([\dots]_S, [w_i, \dots]_B, A)}{([\dots, w_i]_S, [\dots]_B, A)}$

# Example1

**Transitions:** SH-LA-SH-RA-SH-LA-RE-RA-RE-RA



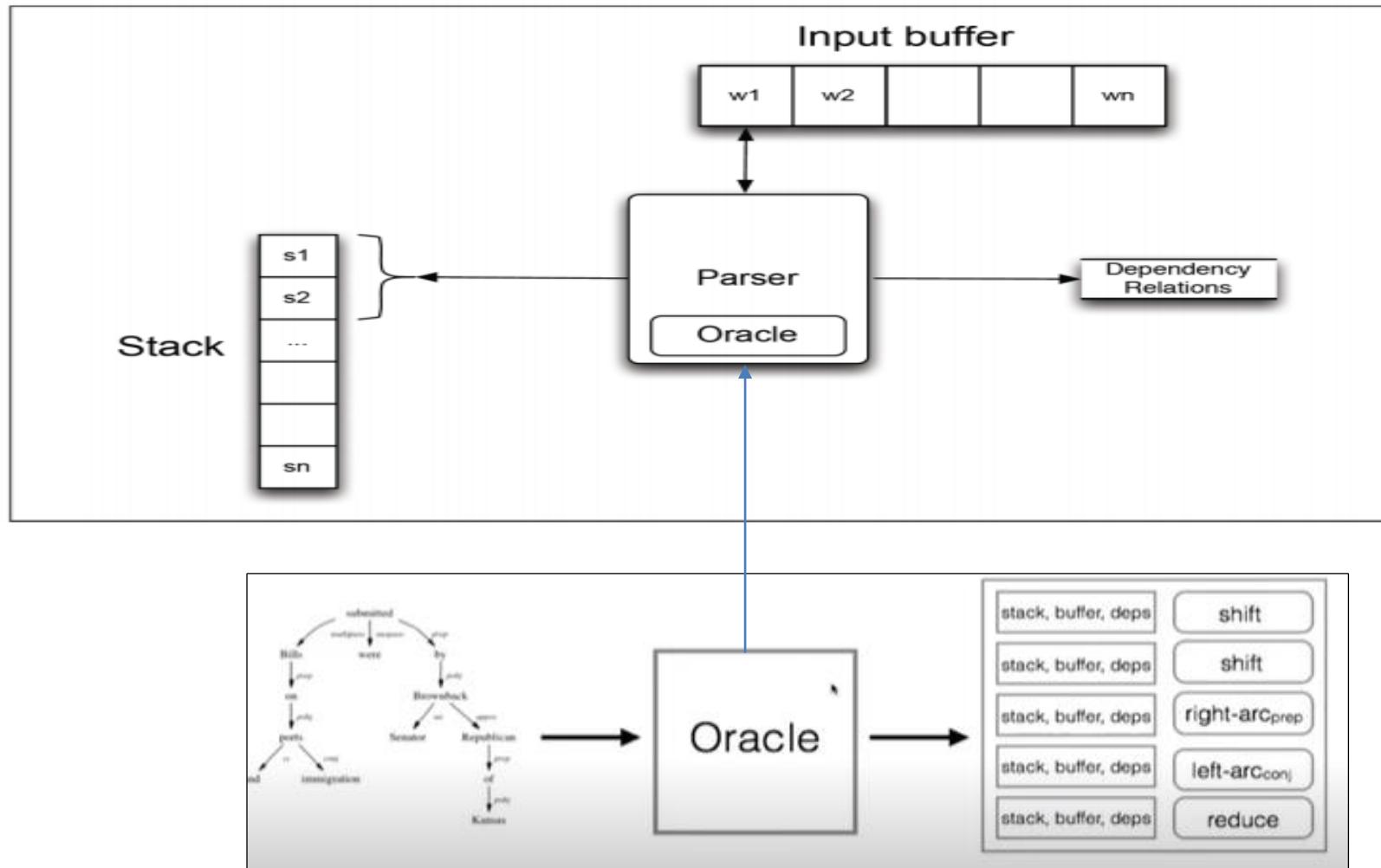
# Example2



Step	Stack	Word List	Action	Relation Added
0	[root]	[book, the, flight, through, houston]	RIGHTARC	(root → book)
1	[root, book]	[the, flight, through, houston]	SHIFT	
2	[root, book, the]	[flight, through, houston]	LEFTARC	(the ← flight)
3	[root, book]	[flight, through, houston]	RIGHTARC	(book → flight)
4	[root, book, flight]	[through, houston]	SHIFT	
5	[root, book, flight, through]	[houston]	LEFTARC	(through ← houston)
6	[root, book, flight]	[houston]	RIGHTARC	(flight → houston)
7	[root, book, flight, houston]	[]	REDUCE	
8	[root, book, flight]	[]	REDUCE	
9	[root, book]	[]	REDUCE	
10	[root]	[]	Done	



# Creating an Oracle



# How the classifier the learns ?

## *Learning Problem*

Approximate a function from **configurations**, represented by feature vectors to **transitions**, given a training set of gold standard **transition sequences**.

## *Three issues*

- How to represent configurations by feature vectors?
- How to derive training data from treebanks?
- How to learn classifiers?

# Feature Models

A feature representation  $f(c)$  of a configuration  $c$  is a vector of simple features  $f_i(c)$ .

## Typical Features

- Nodes:
  - Target nodes (top of  $S$ , head of  $B$ )
  - Linear context (neighbors in  $S$  and  $B$ )
  - Structural context (parents, children, siblings in  $G$ )
- Attributes:
  - Word form (and lemma)
  - Part-of-speech (and morpho-syntactic features)
  - Dependency type (if labeled)
  - Distance (between target tokens)

Feature template:

$$\langle s_1.w, op \rangle, \langle s_2.w, op \rangle \langle s_1.x, op \rangle, \langle s_2.x, op \rangle \\ \langle b_1.w, op \rangle, \langle b_1.x, op \rangle \langle s_1.wt, op \rangle$$

# Feature examples

Stack	Word buffer	Relations
[root, canceled, flights]	[to Houston]	(canceled → United) (flights → morning) (flights → the)

$\langle s_1.w = \text{flights}, op = \text{shift} \rangle$   
 $\langle s_2.w = \text{canceled}, op = \text{shift} \rangle$   
 $\langle s_1.x = \text{NNS}, op = \text{shift} \rangle$   
 $\langle s_2.x = \text{VBD}, op = \text{shift} \rangle$   
 $\langle b_1.w = \text{to}, op = \text{shift} \rangle$   
 $\langle b_1.x = \text{TO}, op = \text{shift} \rangle$   
 $\langle s_1.wt = \text{flightsNNS}, op = \text{shift} \rangle$

# Classifier at runtime

To guide the parser, a linear classifier can be used:

$$t^* = \arg \max_t w.f(c, t)$$

Weight vector  $w$  learned from treebank data.

## *Using classifier at run-time*

```
PARSE( $w_1, \dots, w_n$ )
1    $c \leftarrow ([], [w_1, \dots, w_n]_B, \{\})$ 
2   while  $B_c \neq []$ 
3        $t^* \leftarrow \arg \max_t w.f(c, t)$ 
4        $c \leftarrow t^*(c)$ 
5   return  $T = (\{w_1, \dots, w_n\}, A_c)$ 
```

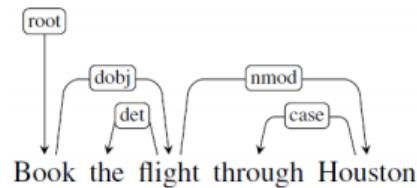
# Training Data

- Training instances have the form  $(f(c), t)$ , where
  - ▶  $f(c)$  is a feature representation of a configuration  $c$ ,
  - ▶  $t$  is the correct transition out of  $c$  (i.e.,  $o(c) = t$ ).
- Given a dependency treebank, we can sample the oracle function  $o$  as follows:
  - ▶ For each sentence  $x$  with gold standard dependency graph  $G_x$ , construct a transition sequence  $C_{0,m} = (c_0, c_1, \dots, c_m)$  such that
$$c_0 = c_s(x),$$
$$G_{c_m} = G_x$$
  - ▶ For each configuration  $c_i (i < m)$ , we construct a training instance  $(f(c_i), t_i)$ , where  $t_i(c_i) = c_{i+1}$ .



# Generating training data example

- What we have in a treebank



- What we need to train an oracle
  - Pairs of configurations and predicted parsing action

Step	Stack	Word List	Predicted Action
0	[root]	[book, the, flight, through, houston]	SHIFT
1	[root, book]	[the, flight, through, houston]	SHIFT
2	[root, book, the]	[flight, through, houston]	SHIFT
3	[root, book, the, flight]	[through, houston]	LEFTARC
4	[root, book, flight]	[through, houston]	SHIFT
5	[root, book, flight, through]	[houston]	SHIFT
6	[root, book, flight, through, houston]	[]	LEFTARC
7	[root, book, flight, houston ]	[]	RIGHTARC
8	[root, book, flight]	[]	RIGHTARC
9	[root, book]	[]	RIGHTARC
10	[root]	[]	Done

# Standard Oracle for Arc Eager parsing



$o(c, T) =$

- **Left-Arc** if  $\text{top}(S_c) \leftarrow \text{first}(B_c)$  in  $T$
- **Right-Arc** if  $\text{top}(S_c) \rightarrow \text{first}(B_c)$  in  $T$
- **Reduce** if  $\exists w < \text{top}(S_c) : w \leftrightarrow \text{first}(B_c)$  in  $T$
- **Shift** otherwise

# Online learning with an oracle

```
LEARN({ $T_1, \dots, T_N$ })
1    $w \leftarrow 0.0$ 
2   for  $i$  in  $1..K$ 
3     for  $j$  in  $1..N$ 
4        $c \leftarrow ([], [w_1, \dots, w_{n_j}]_B, \{\})$ 
5       while  $B_c \neq []$ 
6          $t^* \leftarrow \arg \max_t w.f(c, t)$ 
7          $t_o \leftarrow o(c, T_i)$ 
8         if  $t^* \neq t_o$ 
9            $w \leftarrow w + f(c, t_o) - f(c, t^*)$ 
10           $c \leftarrow t_o(c)$ 
11    return  $w$ 
```

Oracle  $o(c, T_i)$  returns the optimal transition of  $c$  and  $T_i$

# Example

Consider the sentence, ‘John saw Mary’.

Draw a dependency graph for this sentence.

Assume that you are learning a classifier for the data-driven deterministic parsing and the above sentence is a gold-standard parse in your training data. You are also given that *John* and *Mary* are ‘Nouns’, while the POS tag of *saw* is ‘Verb’. Assume that your features correspond to the following conditions:

- ▶ The stack is empty
- ▶ Top of stack is Noun and Top of buffer is Verb
- ▶ Top of stack is Verb and Top of buffer is Noun

Initialize the weights of all your features to 5.0, except that in all of the above cases, you give a weight of 5.5 to *Left-Arc*. Define your feature vector and the initial weight vector.

Use this gold standard parse during online learning and report the weights after completing one full iteration of Arc-Eager parsing over this sentence.

$F(c,t) = [(c_0, LA), (c_1, LA), (c_2, LA) | (c_0, RA), (c_1, RA), (c_2, RA) \dots]$

$W = [5.5, 5.5, 5.5 | 5.0, 5.0, 5.0 | 5.0, 5.0, 5.0 | \dots]$

So for the given conditions

$F(c, LA) = [1, 0, 0 | 0, 0, 0 | \dots]$

$F(c, RA) = [0, 0, 0 | 1, 0, 0 | \dots]$

$t^* = \text{argmax}(w^* f(c, t))$

$t^* = LA$

as per oracle /optimal transition  $t^0 = SH$

So we need to update the weights

To update the weights

$W = W + f(c, t^0) - f(c, t^*)$

$W = [5.5, 5.0, 5.0 | 5.0, 5.0 \dots] + [0, 0, 0 | 0, 0, 0 | \dots 1, 0, 0] - [1, 0, 0 | 0, 0, 0 | \dots]$

New vector =  $[4.5, 5.5, 5.5 | 5.0 \dots 6.0, 5.0, 5.0]$

Now

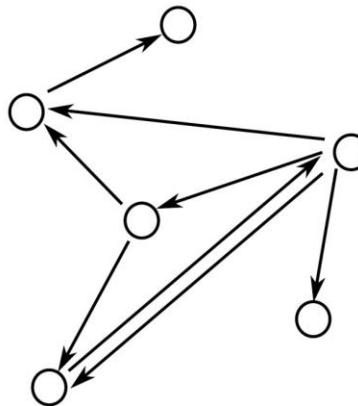
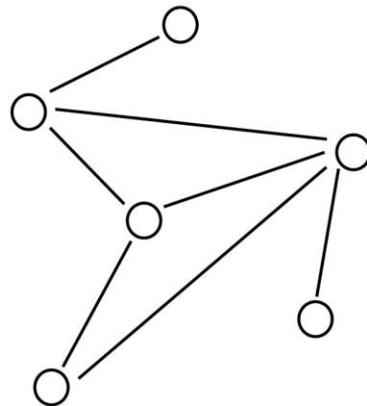
John saw Mary

Next configuration

# Graph-based parsing

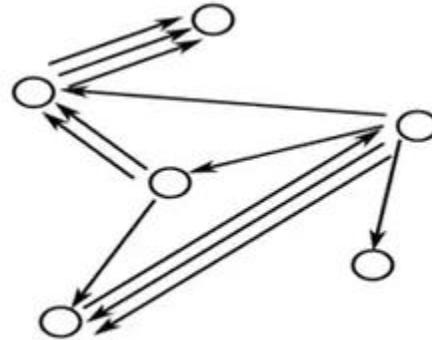
# Graph concepts refresher

- ▶ A graph  $G = (V, A)$  is a set of vertices  $V$  and arcs  $(i, j) \in A$ , where  $i, j \in V$
- ▶ Undirected graphs:  $(i, j) \in A \Leftrightarrow (j, i) \in A$
- ▶ **Directed graphs (digraphs):**  $(i, j) \in A \not\Leftrightarrow (j, i) \in A$



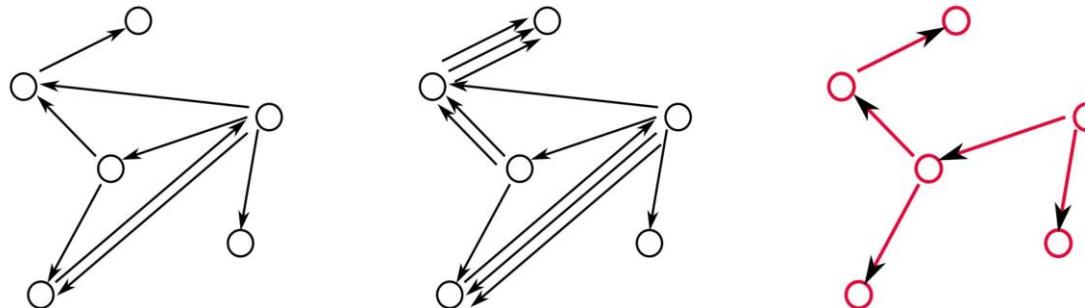
# Multi Digraph

- A multi-digraph is a digraph where multiple arcs between vertices are possible
- $(i,j,k) \in A$  represents the  $k^{th}$  arc from vertex  $i$  to vertex  $j$ .



# Spanning Trees

- ▶ A directed spanning tree of a (multi-)digraph  $G = (V, A)$ , is a subgraph  $G' = (V', A')$  such that:
  - ▶  $V' = V$
  - ▶  $A' \subseteq A$ , and  $|A'| = |V'| - 1$
  - ▶  $G'$  is a tree (acyclic)
- ▶ A spanning tree of the following (multi-)digraphs



# Weighted Spanning tree

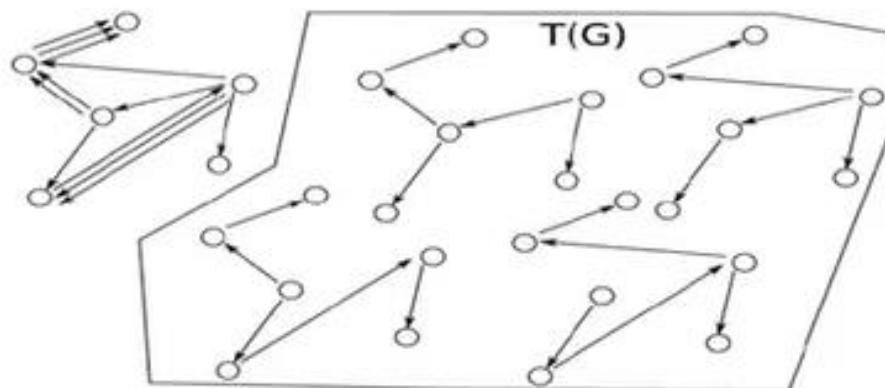
---

- Assume we have a weight function for each arc in a multi-digraph  $G = (V, A)$ .
- Define  $w_{ij}^k \geq 0$  to be the weight of  $(i, j, k) \in A$  for a multi-digraph
- Define the weight of directed spanning tree  $G'$  of graph  $G$  as

$$w(G') = \sum_{(i,j,k) \in G'} w_{ij}^k$$

# MST

Let  $T(G)$  be the set of all spanning trees for graph  $G$



The MST problem

Find the spanning tree  $G'$  of the graph  $G$  that has the highest weight

$$G' = \arg \max_{G' \in T(G)} w(G') = \arg \max_{G' \in T(G)} \sum_{(i,j,k) \in G'} w_{ij}^k$$

# Finding MST

## Directed Graph

For each sentence  $x$ , define the directed graph  $G_x = (V_x, E_x)$  given by

$$V_x = \{x_0 = \text{root}, x_1, \dots, x_n\}$$

$$E_x = \{(i, j) : i \neq j, (i, j) \in [0 : n] \times [1 : n]\}$$

$G_x$  is a graph with

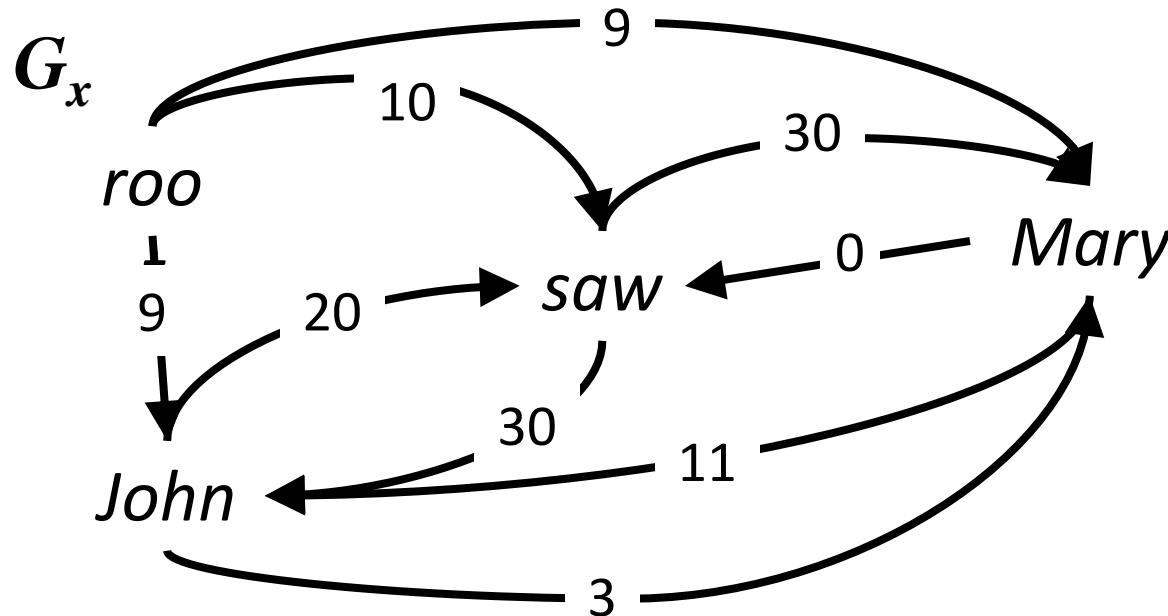
- the sentence words and the dummy root symbol as vertices and
- a directed edge between every pair of distinct words and
- a directed edge from the root symbol to every word

# Chu-Liu-Edmonds algorithm

- Each vertex in the graph greedily selects the incoming edge with the highest weight.
- If a tree results, it must be a maximum spanning tree.
- If not, there must be a cycle.
  - Identify the cycle and contract it into a single vertex.
  - Recalculate edge weights going into and out of the cycle.

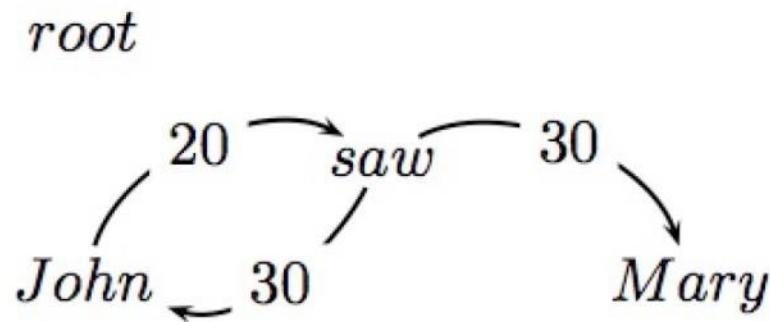
# Chu-Liu-Edmonds Algorithm (2/12)

- $x = John \ saw \ Mary$



# Chu-Liu-Edmonds Example

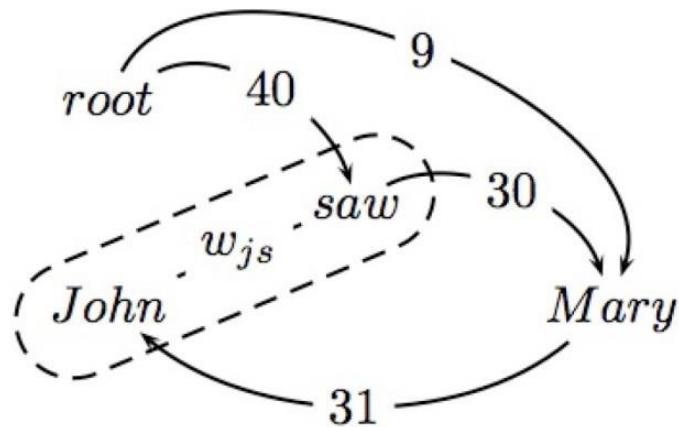
- ▶ Find highest scoring incoming arc for each vertex



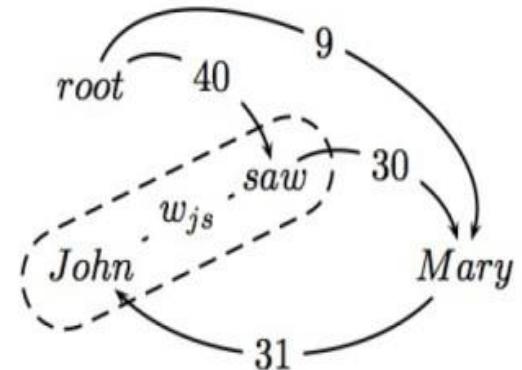
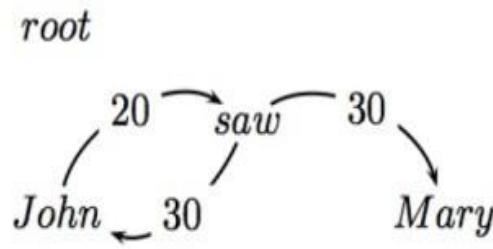
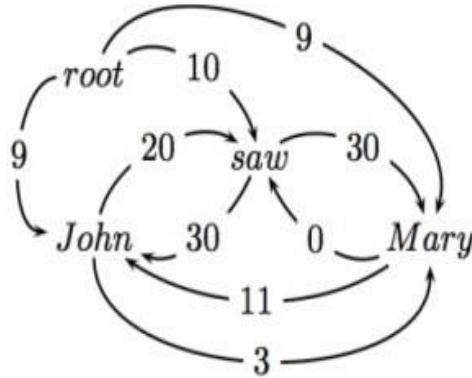
- ▶ If this is a tree, then we have found MST!!

# Chu-Liu-Edmonds Example

- ▶ If not a tree, identify cycle and contract
- ▶ Recalculate arc weights into and out-of cycle

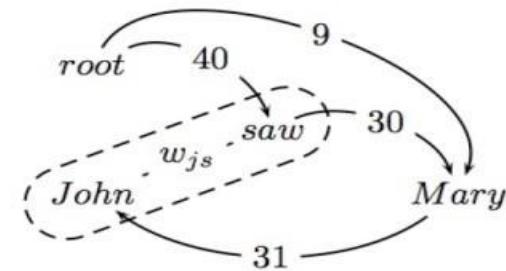
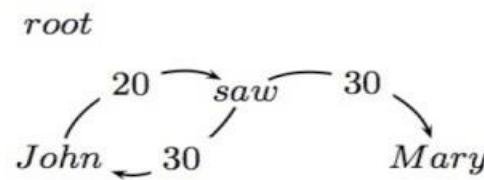
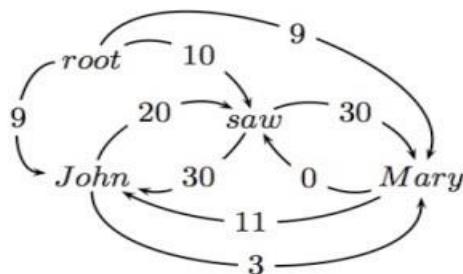


# Chu-Liu-Edmonds Example



- ▶ Outgoing arc weights
  - ▶ Equal to the max of outgoing arc over all vertexes in cycle
  - ▶ e.g., John → Mary is 3 and saw → Mary is 30

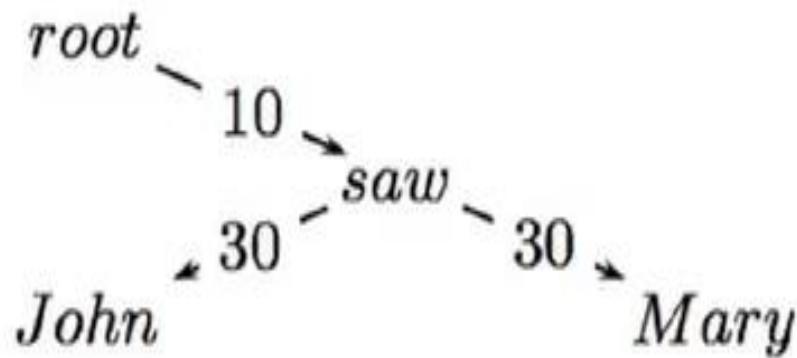
# Chu-Liu-Edmonds Example



## ► Incoming arc weights

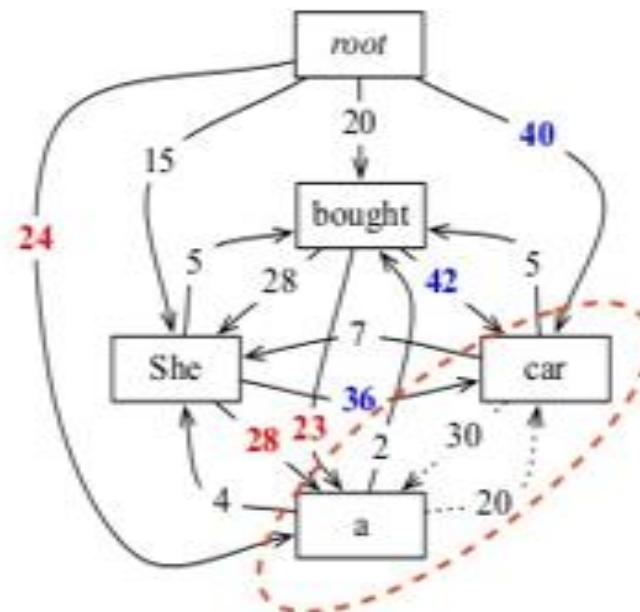
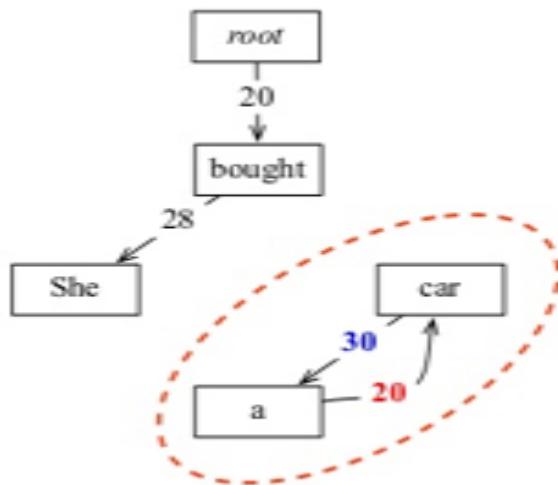
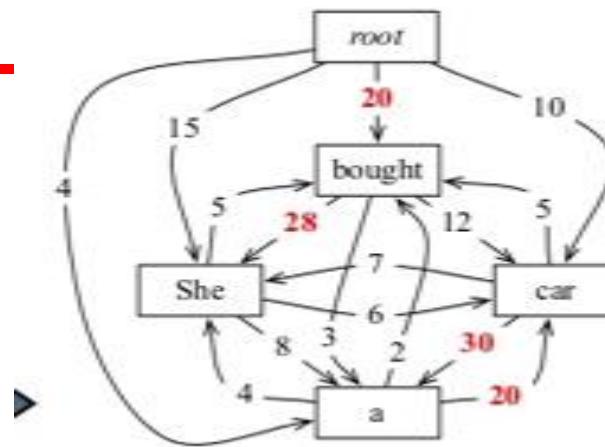
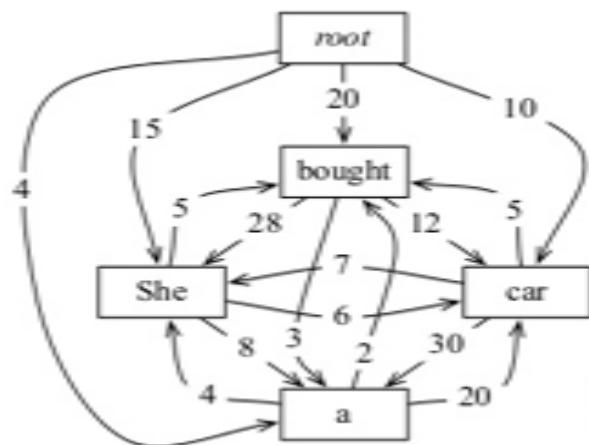
- Equal to the weight of best spanning tree that includes head of incoming arc, and all nodes in cycle
- $\text{root} \rightarrow \text{saw} \rightarrow \text{John}$  is 40 (\*\*)
- $\text{root} \rightarrow \text{John} \rightarrow \text{saw}$  is 29

# Chu-Liu-Edmonds Example



- The edge from  $w_{js}$  to *Mary* was from *saw*
- The edge from *root* to  $w_{js}$  represented a tree from *root* to *saw* to *John*.

# Example2







# MST Learning

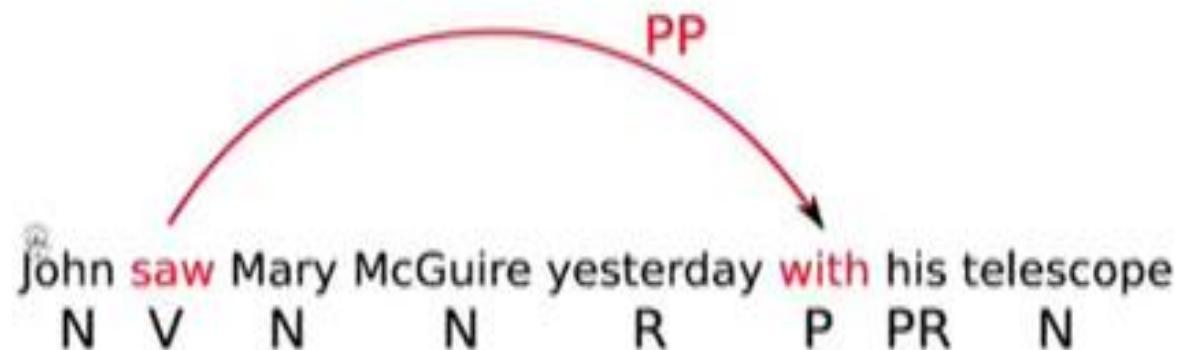
# Linear classifiers

10

$$w_{ij}^k = w.f(i,j,k)$$

- Arc weights are a linear combination of features of the arc  $f(i,j,k)$  and a corresponding weight vector  $w$
- What arc features?

# Arch features



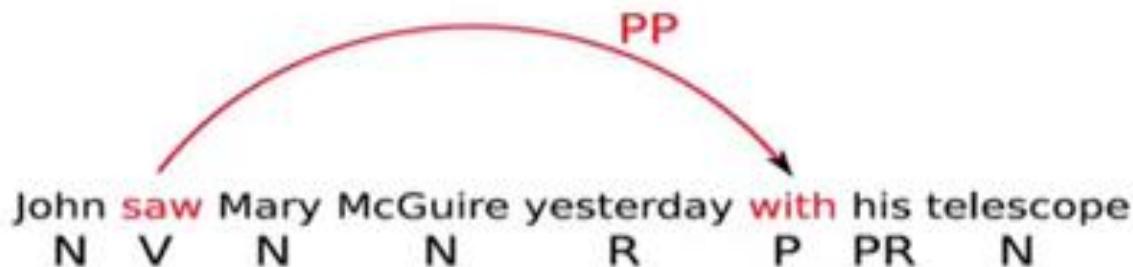
## Features

Identities of the words  $w_i$  and  $w_j$  for a label  $l_k$   
head = saw & dependent=with



## Features

Part-of-speech tags of the words  $w_i$  and  $w_j$  for a label  $l_k$   
head-pos = Verb & dependent-pos=Preposition



## Features

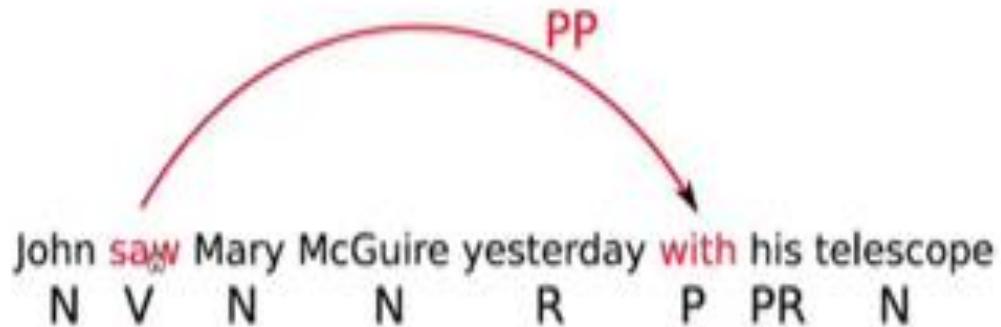
Part-of-speech of words surrounding and between  $w_i$  and  $w_j$

inbetween-pos = Noun

inbetween-pos = Adverb

dependent-pos-right = Pronoun

head-pos-left=Noun



### Features

Number of words between  $w_i$  and  $w_j$ , and their orientation

arc-distance = 3

arc-direction = right

# Learning the parameters

- Re-write the inference problem

$$G = \arg \max_{G \in T(G_x)} \sum_{(i,j,k) \in G} {w_{ij}}^k$$

$$= \arg \max_{G \in T(G_x)} w \cdot \sum_{(i,j,k) \in G} f(i,j,k)$$

$$= \arg \max_{G \in T(G_x)} w \cdot f(G)$$

# Inference based learning

Training data:  $T = \{(x_t, G_t)\}_{t=1}^{|T|}$

1.  $w^{(0)} = 0; i = 0$
2. **for**  $n : 1..N$
3.     **for**  $t : 1..|T|$
4.         Let  $G' = argmax_{G'} w^{(i)}.f(G')$
5.         if  $G' \neq G_t$
6.              $w^{(i+1)} = w^{(i)} + f(G_t) - f(G')$
7.          $i = i + 1$
8.     **return**  $w^i$

# Example

Suppose you are training MST Parser for dependency and the sentence, "John saw Mary" occurs in the training set. Also, for simplicity, assume that there is only one dependency relation, "rel". Thus, for every arc from word  $w_i$  to  $w_j$ , your features may be simplified to depend only on words  $w_i$  and  $w_j$  and not on the relation label.

Below is the set of features

- $f_1: \text{pos}(w_i) = \text{Noun}$  and  $\text{pos}(w_j) = \text{Noun}$
- $f_2: \text{pos}(w_i) = \text{Verb}$  and  $\text{pos}(w_j) = \text{Noun}$
- $f_3: w_i = \text{Root}$  and  $\text{pos}(w_j) = \text{Verb}$
- $f_4: w_i = \text{Root}$  and  $\text{pos}(w_j) = \text{Noun}$
- $f_5: w_i = \text{Root}$  and  $w_j$  occurs at the end of sentence
- $f_6: w_i$  occurs before  $w_j$  in the sentence
- $f_7: \text{pos}(w_i) = \text{Noun}$  and  $\text{pos}(w_j) = \text{Verb}$

The feature weights before the start of the iteration are: {3, 20, 15, 12, 1, 10, 20}.  
Determine the weights after an iteration over this example.

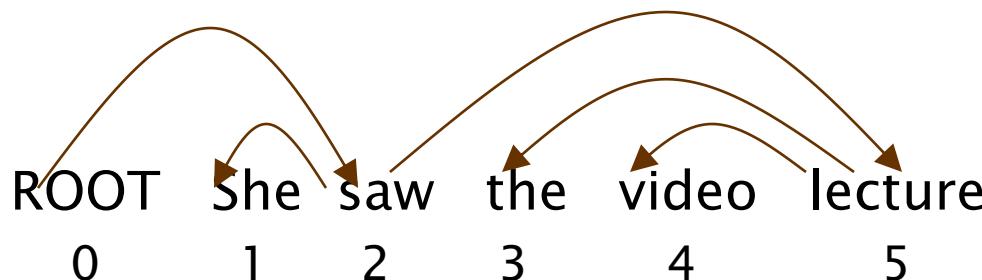


# Evaluation

---

- Unlabeled Attachment Score (UAS), which corresponds to the number of correctly predicted dependencies over the number of possibilities;
  - Labeled Attachment Score (LAS), which corresponds to the number of correctly predicted dependencies and relations over the number of possibilities.

# Evaluation



$$\text{Acc} = \frac{\# \text{ correct deps}}{\# \text{ of deps}}$$

UAS(unlabeled attachment score) = 2  
LAS ( Labeled Attachment Score) = 2

## Gold

1	2	She	nsubj
2	0	saw	root
3	5	the	det
4	5	video	nn
5	2	lecture	dobj

## Parsed

1	2	She	nsubj
2	0	saw	root
3	4	the	det
4	5	video	nsubj
5	2	lecture	ccomp

# References



- 
1. Speech and Language processing: An introduction to Natural Language Processing, Computational Linguistics and speech Recognition by Daniel Jurafsky and James H. Martin[3rd edition].
  2. <https://www.youtube.com/watch?v=PVShkZgXznc>
  3. <https://www.youtube.com/watch?v=02QWRAhGc7g&list=PLJJzI13YAXCHxbVgiFaSI88hj-mRSoMtl>
  4. [https://www.researchgate.net/publication/328731166\\_Weighted\\_Machine\\_Learning](https://www.researchgate.net/publication/328731166_Weighted_Machine_Learning)

---

Any Questions?

---

# Thank you



# Natural Language Processing

## DSECL ZG565

Prof. Vijayalakshmi  
BITS-Pilani

**BITS** Pilani  
Pilani Campus



# Session Content

(Ref: Chapter 19 Jurafsky and Martin)

---

- Word Senses
- Relations between Senses
- WordNet: A Database of Lexical Relations
- Word Sense Disambiguation
- Alternate WSD algorithms and Tasks
- Using Thesauruses to Improve Embeddings
- Word Sense Induction

# Motivation

---

- Computationally determining which sense of a word is activated by its use in a particular context.
  - E.g. I am going to withdraw money from the ***bank***.

# Application



- Machine Translation
- Question Answering
- Information Retrieval
- Information Extraction
- Speech processing

# What's a word sense?

---

- A sense (or word sense) is a discrete representation of one aspect of the meaning of a word
- Example
  - We went to see the play Romeo and Juliet at the theater.
  - The children went out to play in the park.

# Relationships between word senses

---

- Homonymy
  - Polysemy
  - Synonymy
  - Antonymy
  - Hypernymy
  - Hyponomy
-

# Homonymy

---

Lexemes that share a form

- Phonological, orthographic or both

But have unrelated, distinct meanings

- Examples
  - bat (wooden stick-like thing) vs bat (flying scary mammal thing)
  - bank (financial institution) versus bank (riverside)
- Can be homophones, homographs, or both:
  - Homophones:
    - Write and right
    - Piece and peace

# Homonymy causes problems for NLP applications

---

- Text-to-Speech
  - Bass vs bass
- Information retrieval
  - Bat care
- Machine Translation
  - I went to bank
- Speech recognition
  - Piece and peace



# Polysemy

---

The **bank** is constructed from red brick

I withdrew the money from the **bank**

- Which sense of bank is this?
  - Is it distinct from the river bank sense?
  - How about the savings bank sense?

Another example:

His cottage is near a small **wood**.

The statue was made out of a block of **wood**.

Are those the same sense?

---

# Polysemy

---

- A single lexeme with multiple **related** meanings (bank the building, bank the financial institution)
- Most non-rare words have multiple meanings
  - The number of meanings is related to its frequency
  - Verbs tend more to polysemy
  - Distinguishing polysemy from homonymy isn't always easy (or necessary)

# Synonyms

---

Word that have the same meaning in some or all contexts.

- filbert / hazelnut
- couch / sofa
- big / large
- automobile / car
- vomit / throw up
- Water / H<sub>2</sub>O

Two lexemes are synonyms if they can be successfully substituted for each other in all situations

# But

---

There are no examples of perfect synonymy

- Why should that be?
- Even if many aspects of meaning are identical
- Still may not preserve the acceptability based on notions of politeness, slang, register, genre, etc.

Example:

- Water and H<sub>2</sub>O

# Synonymy is a relation between senses rather than words

---

Consider the words *big* and *large*

Are they synonyms?

- How **big** is that plane?
- Would I be flying on a **large** or small plane?

How about here:

- Miss Nelson, for instance, became a kind of **big** sister to Benjamin.
- ?Miss Nelson, for instance, became a kind of **large** sister to Benjamin.

Why?

- *big* has a sense that means being older, or grown up
- *large* lacks this sense

# Antonyms

---

Senses that are opposites with respect to one feature of their meaning

Otherwise, they are very similar!

- dark / light
- short / long
- hot / cold
- up / down
- in / out

More formally: antonyms can

- define a binary opposition or at opposite ends of a scale (*long/short, fast/slow*)
- Be **reverses**: *rise/fall, up/down*

# Hyponymy

- One sense is a **hyponym** of another if the first sense is more specific, denoting a subclass of the other
  - *car* is a hyponym of *vehicle*
  - *dog* is a hyponym of *animal*
  - *mango* is a hyponym of *fruit*
- Conversely
  - *vehicle* is a hypernym/superordinate of *car*
  - *animal* is a hypernym of *dog*
  - *fruit* is a hypernym of *mango*

<b>superordinate</b>	vehicle	fruit	furniture	mammal
<b>hyponym</b>	car	mango	chair	dog

# WordNet

- A hierarchically organized lexical database
- On-line thesaurus + aspects of a dictionary
  - Versions for other languages are under development

Category	Entries
Noun	117,097
Verb	11,488
Adjective	22,141
Adverb	4,601

# Format of Wordnet Entries

## WordNet Search - 3.1

- [WordNet home page](#) - [Glossary](#) - [Help](#)

Word to search for:

Display Options:

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations

Display options for sense: (gloss) "an example sentence"

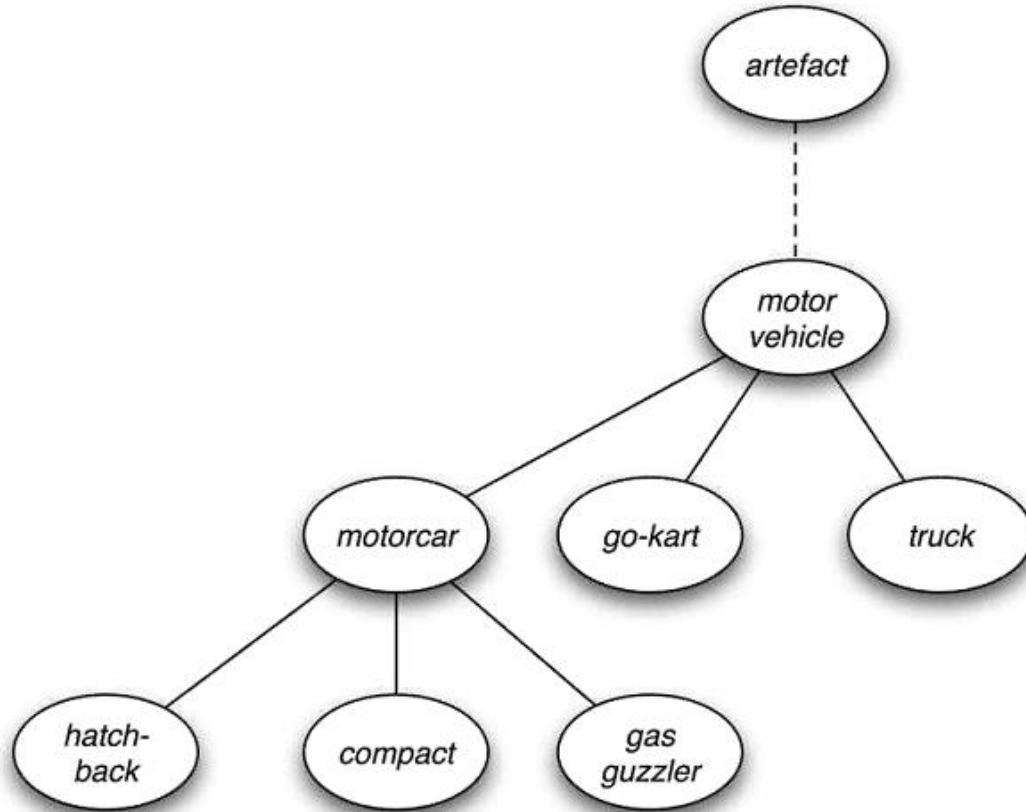
### Noun

- S: (n) **bass** (the lowest part of the musical range)
- S: (n) **bass**, [bass part](#) (the lowest part in polyphonic music)
- S: (n) **bass**, [basso](#) (an adult male singer with the lowest voice)
- S: (n) [sea bass](#), **bass** (the lean flesh of a saltwater fish of the family Serranidae)
- S: (n) [freshwater bass](#), **bass** (any of various North American freshwater fish with lean flesh (especially of the genus Micropterus))
- S: (n) **bass**, [bass voice](#), [basso](#) (the lowest adult male singing voice)
- S: (n) **bass** (the member with the lowest range of a family of musical instruments)
- S: (n) **bass** (nontechnical name for any of numerous edible marine and freshwater spiny-finned fishes)

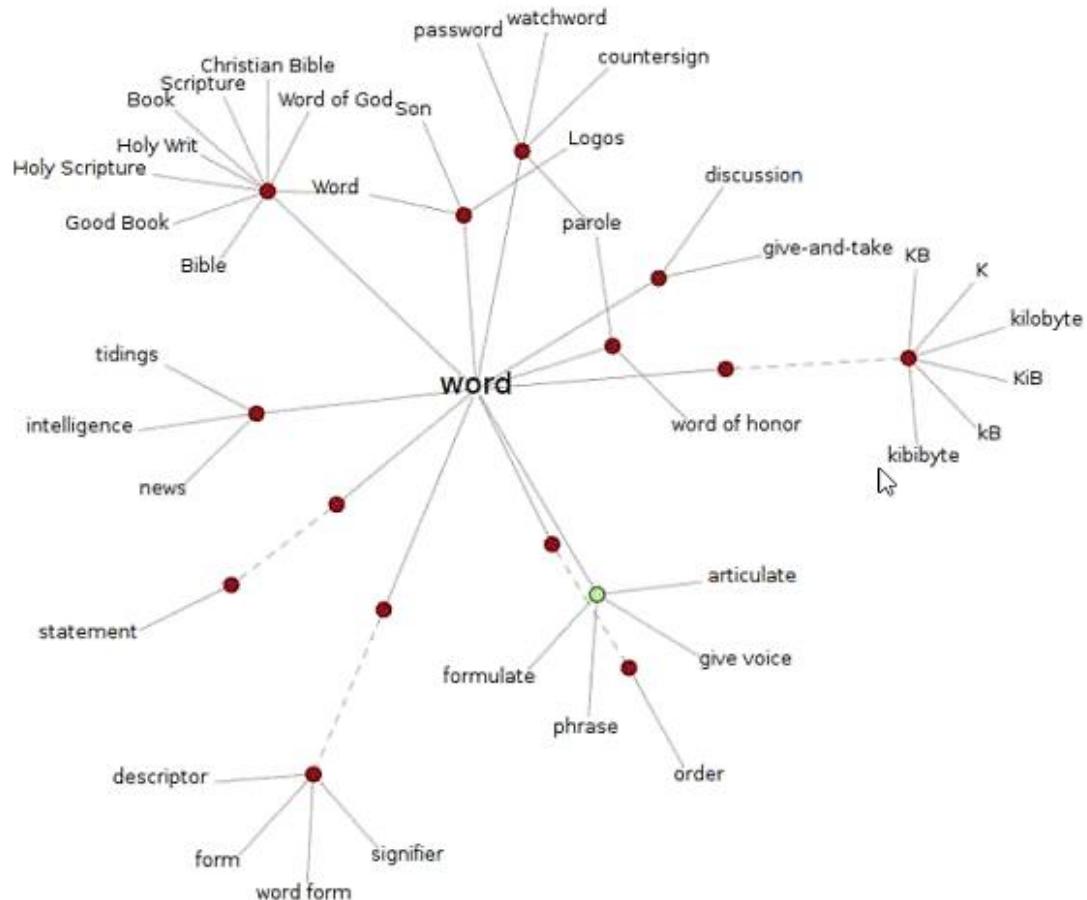
### Adjective

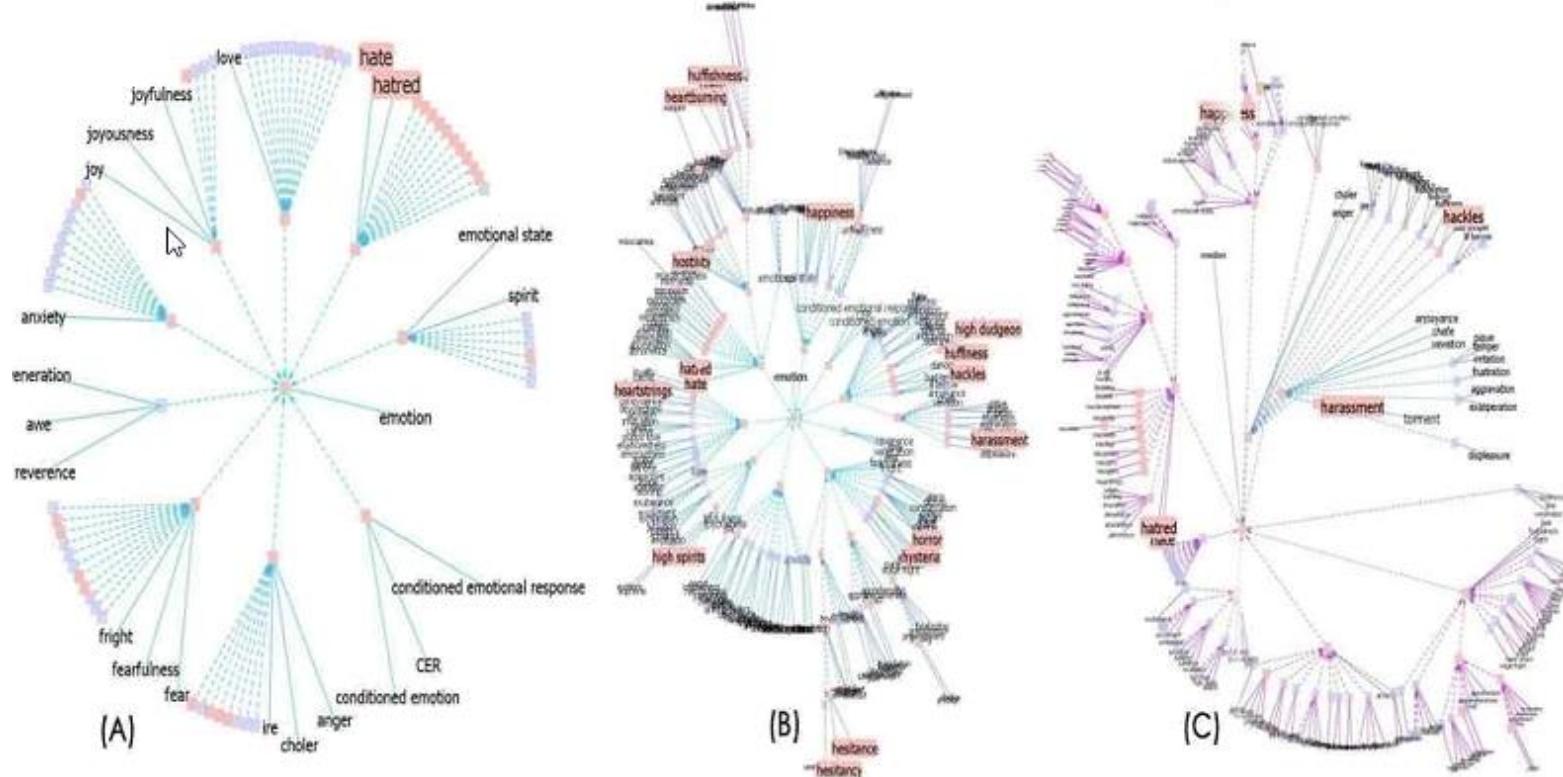
- S: (adj) **bass**, [deep](#) (having or denoting a low vocal or instrumental range) "a deep voice"; "a bass voice is lower than a baritone voice"; "a bass clarinet"

# Structure



► Source: [www. docs.huihoo.com/nltk/0.9.5/en/ch02.html](http://www. docs.huihoo.com/nltk/0.9.5/en/ch02.html)





<sup>12</sup> Department of Commerce, Board of Trade, "Trade statistics with U.S. and other countries concerning sugar products over last half century in U.S.A." Board of Trade publications on World Sugar Survey of the World Sugar Conference, London, 1933, pp. 10-11.

# Wordnet

- The set of near-synonyms for a WordNet sense is called a **synset (synonym set)**; it's their version of a sense or a concept

Example: chump as a noun to mean

'a person who is gullible and easy to take advantage of'

{chump<sup>1</sup>, fool<sup>2</sup>, gull<sup>1</sup>, mark<sup>9</sup>, patsy<sup>1</sup>, fall guy<sup>1</sup>, sucker<sup>1</sup>, soft touch<sup>1</sup>, mug<sup>2</sup>}

- Each of these senses share this same gloss
- Thus for WordNet, the meaning of this sense of chump is this list.

# WordNet Noun Relations

---

Relation	Also called	Definition	Example
Hypernym	Superordinate	From concepts to superordinates	<i>breakfast</i> <sup>1</sup> → <i>meal</i> <sup>1</sup>
Hyponym	Subordinate	From concepts to subtypes	<i>meal</i> <sup>1</sup> → <i>lunch</i> <sup>1</sup>
Member Meronym	Has-Member	From groups to their members	<i>faculty</i> <sup>2</sup> → <i>professor</i> <sup>1</sup>
Has-Instance		From concepts to instances of the concept	<i>composer</i> <sup>1</sup> → <i>Bach</i> <sup>1</sup>
Instance		From instances to their concepts	<i>Austen</i> <sup>1</sup> → <i>author</i> <sup>1</sup>
Member Holonym	Member-Of	From members to their groups	<i>copilot</i> <sup>1</sup> → <i>crew</i> <sup>1</sup>
Part Meronym	Has-Part	From wholes to parts	<i>table</i> <sup>2</sup> → <i>leg</i> <sup>3</sup>
Part Holonym	Part-Of	From parts to wholes	<i>course</i> <sup>7</sup> → <i>meal</i> <sup>1</sup>
Antonym		Opposites	<i>leader</i> <sup>1</sup> → <i>follower</i> <sup>1</sup>

# WordNet Verb Relations

---

Relation	Definition	Example
Hypernym	From events to superordinate events	<i>fly</i> <sup>9</sup> → <i>travel</i> <sup>P</sup>
Troponym	From a verb (event) to a specific manner elaboration of that verb	<i>walk</i> <sup>1</sup> → <i>stroll</i> <sup>1</sup>
Entails	From verbs (events) to the verbs (events) they entail	<i>snore</i> <sup>1</sup> → <i>sleep</i> <sup>1</sup>
Antonym	Opposites	<i>increase</i> <sup>1</sup> ⇔ <i>decrease</i> <sup>1</sup>

# WordNet Hierarchies

```
Sense 3
bass, basso --
(an adult male singer with the lowest voice)
=> singer, vocalist, vocalizer, vocaliser
    => musician, instrumentalist, player
        => performer, performing artist
            => entertainer
                => person, individual, someone...
                    => organism, being
                        => living thing, animate thing,
                            => whole, unit
                                => object, physical object
                                    => physical entity
                                        => entity
                => causal agent, cause, causal agency
                    => physical entity
                        => entity

Sense 7
bass --
(the member with the lowest range of a family of
musical instruments)
=> musical instrument, instrument
    => device
        => instrumentality, instrumentation
            => artifact, artefact
                => whole, unit
                    => object, physical object
                        => physical entity
                            => entity
```

# Word Sense Disambiguation (WSD)

---

The task of selecting the correct sense for a word is called word sense disambiguation, or WSD

Given

- a word in context,
- a fixed inventory of potential word sense

Decide which sense of the word this is

Examples

- English-to-Spanish MT
  - Inventory is set of Spanish translations
- Speech Synthesis
  - Inventory is homographs with different pronunciations like *bass* and *bow*

# WSD: The Task and Datasets

---

- The inventory of sense tags depends on the task.
- Example
  - *For sense tagging in the context of translation from English to Spanish, the sense tag inventory for an English word might be the set of different Spanish translations.*
  - *For automatic indexing of medical articles, the sense-tag inventory might be the set of MeSH (Medical Subject Headings) thesaurus entries.*
- We can use the set of senses from a resource like WordNet, or supersenses if we want a coarser-grain set.

# Example :Inventory of sense tags for bass

WordNet Sense	Spanish Translation	WordNet Supersense	Target Word in Context
bass <sup>4</sup>	lubina	FOOD	... fish as Pacific salmon and striped bass and...
bass <sup>7</sup>	bajo	ARTIFACT	... play bass because he doesn't have to solo...

# Two variants of WSD task

---

## Lexical sample task

- Small pre-selected set of target words
- And inventory of senses for each word
- Since these words and the set of senses are small, simple supervised classification approaches work very well

## All-words task

- In this all-words task, the system is given an all-words entire texts and
- lexicon with an inventory of senses for each entry
- we have to disambiguate every word in the text (or sometimes just every content word).

# WSD Methods

- 
- Supervised
  - Dictionary based
  - Semi supervised
-

# **Word Sense Disambiguation**

**Supervised  
Machine Learning**

# Supervised Machine Learning Approaches

---

Supervised machine learning approach:

- a **training corpus** of words tagged in context with their sense
- used to train a classifier that can tag words in new text

Summary of what we need:

- the **tag set** (“sense inventory”)
- the **training corpus**
- A set of **features** extracted from the training corpus
- A **classifier**

# Supervised WSD 1: WSD Tags

---

What's a tag?

- A dictionary sense?

For example, for WordNet an instance of “bass” in a text has 8 possible tags or labels (bass1 through bass8).

# WordNet Bass

---

The noun ``bass" has 8 senses in WordNet

1. bass - (the lowest part of the musical range)
2. bass, bass part - (the lowest part in polyphonic music)
3. bass, basso - (an adult male singer with the lowest voice)
4. sea bass, bass - (flesh of lean-fleshed saltwater fish of the family Serranidae)
5. freshwater bass, bass - (any of various North American lean-fleshed freshwater fishes especially of the genus Micropterus)
6. bass, bass voice, basso - (the lowest adult male singing voice)
7. bass - (the member with the lowest range of a family of musical instruments)
8. bass -(nontechnical name for any of numerous edible marine and freshwater spiny-finned fishes)

# Supervised WSD 2: Get a corpus

---

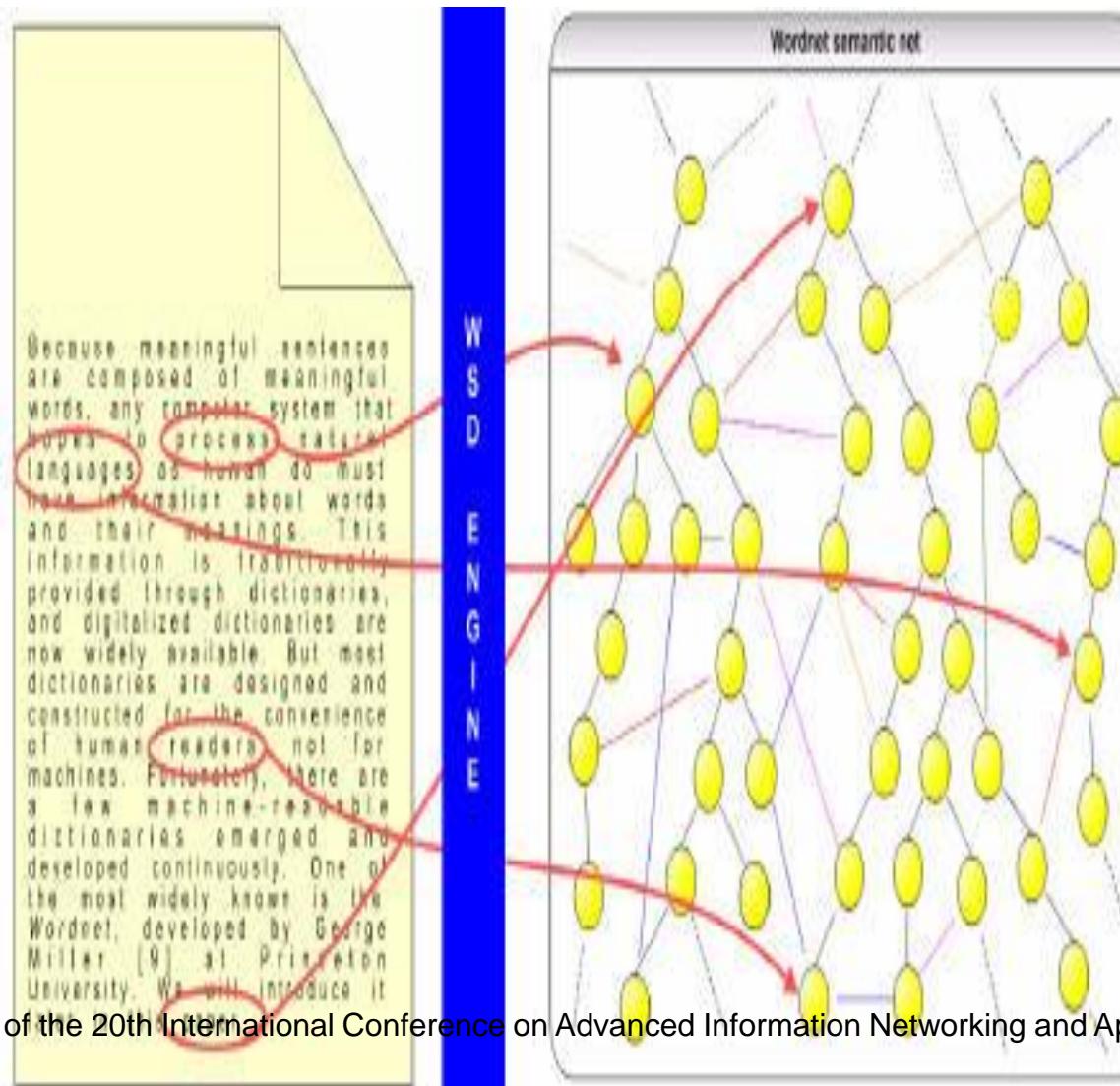
Lexical sample task:

- *Line-hard-serve* corpus - 4000 examples of each
- *Interest* corpus - 2369 sense-tagged examples

All words:

- **Semantic concordance**: a corpus in which each open-class word is labeled with a sense from a specific dictionary/thesaurus.
  - SemCor: 234,000 words from Brown Corpus, manually tagged with WordNet senses
  - SENSEVAL-3 competition corpora - 2081 tagged word tokens

# WSD: Semantic relatedness and word sense disambiguation



Source: Proceedings of the 20th International Conference on Advanced Information Networking and Applications

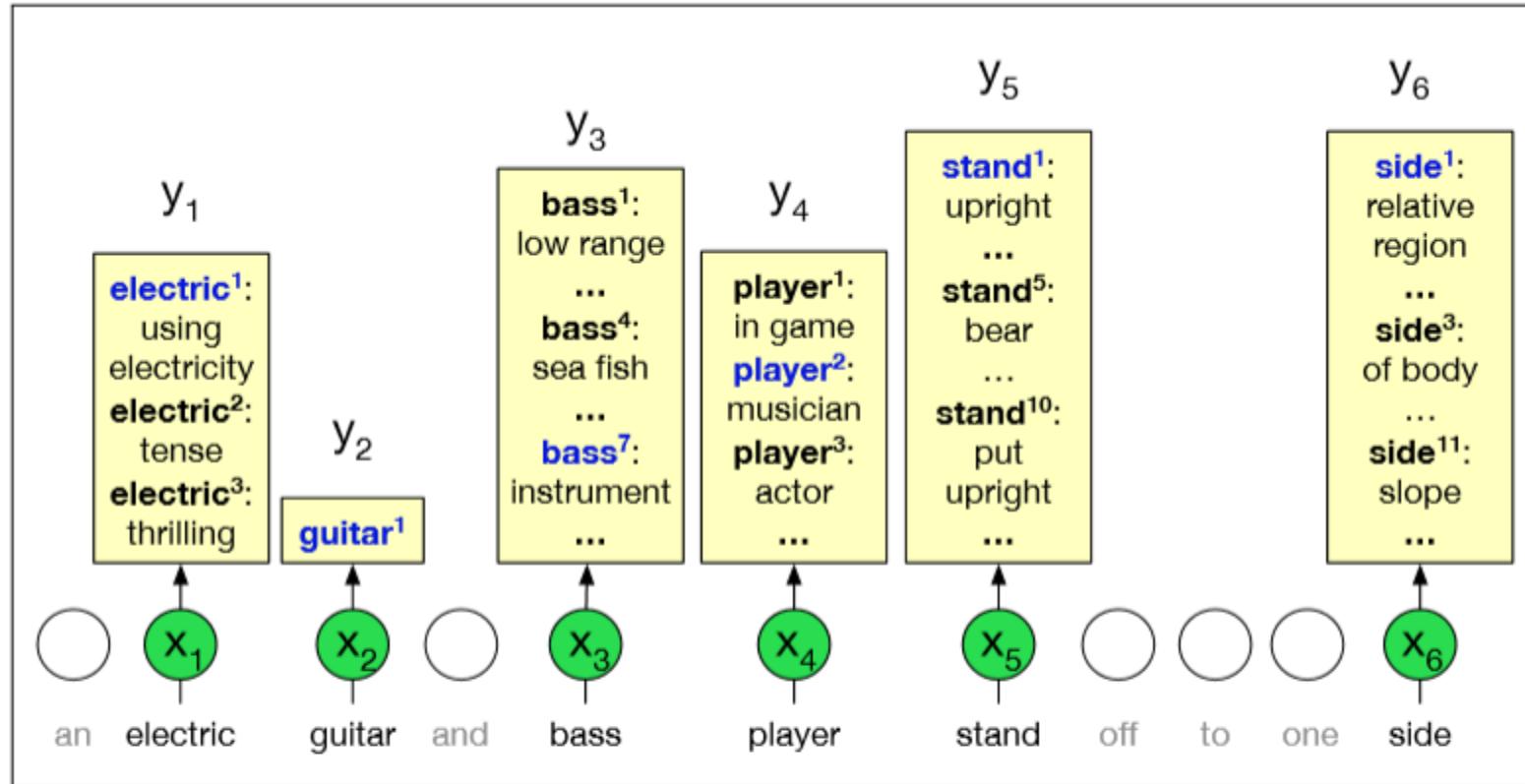
# Example of SemCor with wordnet sense numbers

---

**You will find<sup>9</sup>, that avocado<sup>1</sup>, is<sup>1</sup>, unlike<sup>1</sup>,**  
**other<sup>1</sup>, fruit<sup>1</sup>, you have ever<sup>1</sup>, tasted<sup>2</sup>,**

- Given each noun, verb, adjective, or adverb word in the hand-labeled test set. Ex: fruit<sup>1</sup>, (the ripened reproductive body of a seed plant), and the other two senses fruit<sup>2</sup> (yield;an amount of a product) and fruit<sup>3</sup> (the consequence of some effort or action).
-

# All word WSD task



**Figure 19.8** The all-words WSD task, mapping from input words ( $x$ ) to WordNet senses ( $y$ ). Only nouns, verbs, adjectives, and adverbs are mapped, and note that some words (like *guitar* in the example) only have one sense in WordNet. Figure inspired by Chaplot and Salakhutdinov (2018).

# Supervised WSD 3: Extract feature vectors

---

A simple representation for each observation  
(each instance of a target word)

- Vectors of sets of feature/value pairs
  - I.e. files of comma-separated values
- These vectors should represent the window of words around the target

# Two kinds of features in the vectors

---

Collocational features and **bag-of-words** features

## – Collocational

- Features about words at **specific** positions near target word
  - Often limited to just word identity and POS

## – Bag-of-words

- Features about words that occur anywhere in the window (regardless of position)
  - Typically limited to frequency counts

# Examples

---

Example text (WSJ)

An electric guitar and **bass** player stand off to one side not really part of the scene, just as a sort of nod to gringo expectations perhaps

- Assume a window of +/- 2 from the target

# Examples

---

Example text (WSJ)

An electric **guitar** and **bass** **player** **stand** off to one side not really part of the scene,

Assume a window of +/- 2 from the target

# Collocational

---

Position-specific information about the words in the window  
**guitar and bass player stand**

- [guitar, NN, and, CC, player, NN, stand, VB]
- Word<sub>n-2</sub>, POS<sub>n-2</sub>, word<sub>n-1</sub>, POS<sub>n-1</sub>, Word<sub>n+1</sub> POS<sub>n+1</sub>...
- In other words, a vector consisting of
- [position n word, position n part-of-speech...]

# Collocational

---

- An electric guitar and bass player stand off to one side, If we used as small 2-word window, a standard feature vector might include parts-of speech, unigram and bigram collocation features, and a weighted sum g of embeddings, that is:

$[w_{i-2}, POS_{i-2}, w_{i-1}, POS_{i-1}, w_{i+1}, POS_{i+1}, w_{i+2}, POS_{i+2},$   
 $w_{i-2}^{i-1}, w_{i+1}^{i+2}$ , would yield the following vector:

**[guitar, NN, and, CC, player, NN, stand, VB, and  
guitar, player stand]**

# Bag-of-words

---

Words that occur within the window, regardless of specific position

First derive a set of terms to place in the vector

Then note how often each of those terms occurs in a given window

# Bag of words Example

---

Assume we've settled on a possible vocabulary of 12 words that includes **guitar** and **player** but not **and** and **stand**

[*fish*, *big*, *sound*, *player*, *fly*, *rod*, *pound*, *double*, *runs*, *playing*, **guitar**, **band**]

- The vector for:  
**guitar** and **bass** **player** **stand**  
[0,0,0,1,0,0,0,0,0,1,0]

# Supervised WSD4:Classifier

---

## *Input:*

- a word  $w$  in a text window  $d$  (which we'll call a "document")
- a fixed set of classes  $C = \{c_1, c_2, \dots, c_J\}$
- A training set of  $m$  hand-labeled text windows again called "documents"  $(d_1, c_1), \dots, (d_m, c_m)$

## *Output:*

- a learned classifier  $y: d \rightarrow c$

---

## Any kind of classifier

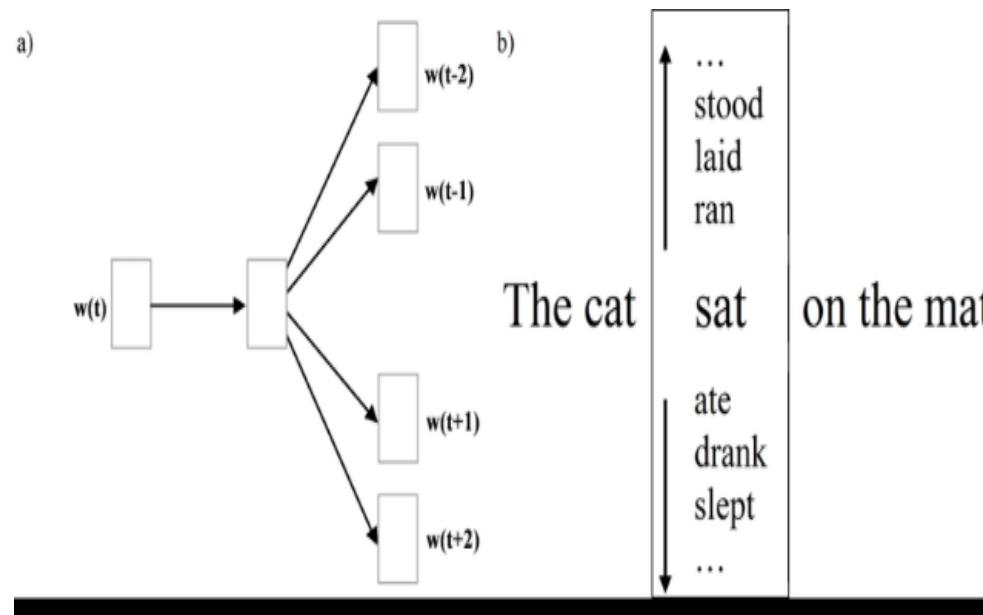
- Naive Bayes
- Logistic regression
- Neural Networks
- Support-vector machines
- k-Nearest Neighbors

# Contextual Embedding

- **Word embedding** is the collective name for a set of language modeling and feature learning techniques in NLP where words or phrases from the vocabulary are mapped to vectors of real numbers
- Intuition of embedding models like word2vec or GloVe is that the meaning of a word can be defined by its co-occurrences, the counts of words that often occur nearby.
- But doesn't tell us how to define the meaning of a word
- Contextual embedding's like ELMo or BERT go further by offering an embedding that represents the meaning of a word in its textual context, and we'll see that contextual embedding's lie at the heart of modern algorithms for word sense disambiguation.

# Word2Vec Skip gram

Skip-gram is one of the unsupervised learning techniques used to find the most related words for a given word.



# Contextual Embedding

---

Back when I was living in Oakland, my house was burglarized. I was living with my brother, his wife, her sister, their two dogs, and my five-month old son and in the middle of the night, someone broke in and cleaned us out. The worst part for me was that he or she stole the camcorder that I'd been recording my son with. I'd actually caught my baby's first laugh on tape, and the thief stole it. Other stuff, too, but it's the laugh that hurts.

This story uses multiple elements of contextual embedding

- where the house is—Oakland, California.
- people who were there—my brother, etc.
- sense of a specific moment in time—when my son was five months old.
- Contextual embedding sets a scene. It provides context for the story, by including information about the place and time where an event happened.

# BERT (Bidirectional Encoder Representations from Transformers)

---

- BERT is a recent [paper](#) published by researchers at Google AI Language.
- Used in Question Answering (SQuAD v1.1), Natural Language Inference (MNLI), and others.
- Technical innovation is applying the bidirectional training of Transformer, a popular attention model, to language modelling.
- This is in contrast to previous efforts which looked at a text sequence either from left to right or combined left-to-right and right-to-left training.
- BERT makes use of Transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text.

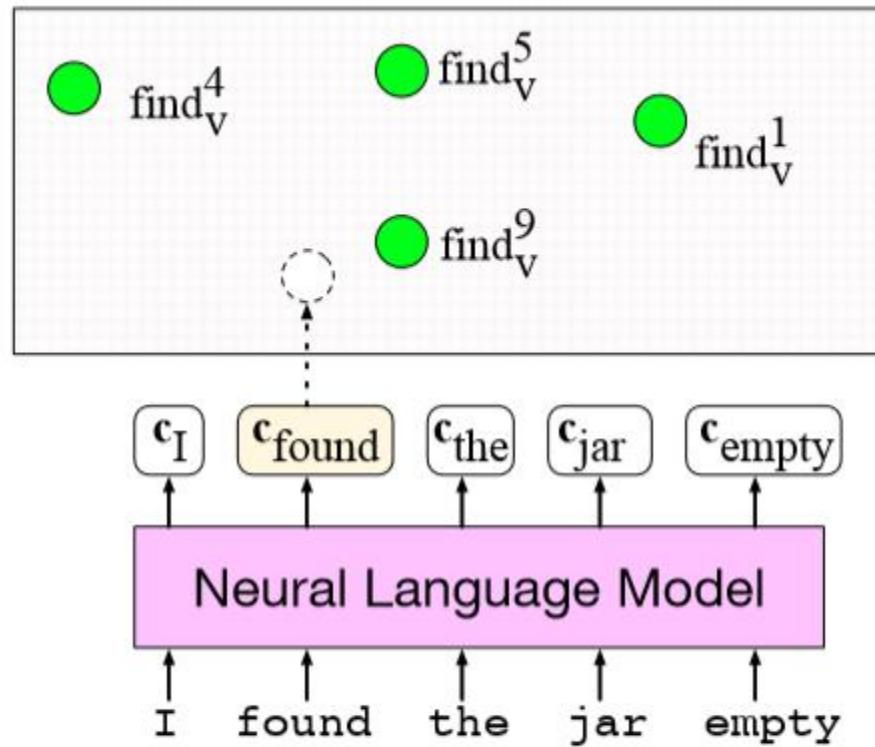
# The WSD Algorithm: Contextual Embeddings

---

- Best-performing WSD algorithm is a simple 1-nearest-neighbor algorithm using contextual word embeddings
- For each token  $c_i$  of each sense  $c$  of each word, we average the contextual representations to produce a contextual **sense embedding  $v_s$**  for  $c$

$$v_s = \frac{1}{n} \sum_i c_i$$

- At test time we similarly compute a contextual embedding  $t$  for the target word, and choose its nearest neighbor sense (the sense with the highest cosine with  $t$ ) from the training set.



**Figure 19.9** The nearest-neighbor algorithm for WSD. In green are the contextual embeddings precomputed for each sense of each word; here we just show a few of the senses for *find*. A contextual embedding is computed for the target word *found*, and then the nearest neighbor sense (in this case  $\text{find}_n^9$ ) would be chosen. Figure inspired by Loureiro and Jorge (2019).

# Alternate WSD algorithms and Tasks



- Feature-Based WSD
- LESK Algorithm
- Word-in-Context Evaluation
- Wikipedia as a source of training data

# Feature-Based WSD

---

Uses an SVM classifier to choose the sense for each input word with the following simple features of the surrounding words:

- part-of-speech tags (for a window of 3 words on each side, stopping at sentence boundaries)
- collocation features of words or n-grams of lengths 1, 2, 3) at a particular collocation location in a window of 3 word on each side (i.e., exactly one word to the right, or the two words starting 3 words to the left, and so on).
- weighted average of embeddings (of all words in a window of 10 words on each side, weighted exponentially by distance)

# **Word Sense Disambiguation**

**Dictionary and  
Thesaurus Methods**

# The Lesk Algorithm as WSD Baseline

---

- Lesk algorithm is the most powerful knowledge-based WSD algorithm
- Lesk is really a family of algorithms that choose the sense whose dictionary gloss or definition shares the most words with the target word's neighborhood

# Lesk Algorithm Example

- Consider disambiguating the word bank in the following context:

***The bank can guarantee deposits will eventually cover future tuition costs because it invests in adjustable-rate mortgage securities.***

bank <sup>1</sup>		“ <i>the bank</i> ”
	Gloss:	a financial institution that accepts deposits and channels the money into lending activities
	Examples:	“he cashed a check at the bank”, “that bank holds the mortgage on my home”
bank <sup>2</sup>		“ <i>the bank</i> ”
	Gloss:	sloping land (especially the slope beside a body of water)
	Examples:	“they pulled the canoe up on the bank”, “he sat on the bank of the river and watched the currents”

- Sense bank1 has two non-stopwords overlapping with the context in deposits and mortgage, while sense bank2 has zero words, so bank1 is chosen.

# Lesk Algorithm

```
function SIMPLIFIED LESK(word, sentence) returns best sense of word
    best-sense  $\leftarrow$  most frequent sense for word
    max-overlap  $\leftarrow$  0
    context  $\leftarrow$  set of words in sentence
    for each sense in senses of word do
        signature  $\leftarrow$  set of words in the gloss and examples of sense
        overlap  $\leftarrow$  COMPUTEOVERLAP(signature, context)
        if overlap > max-overlap then
            max-overlap  $\leftarrow$  overlap
            best-sense  $\leftarrow$  sense
    end
    return(best-sense)
```

**Figure 19.10** The Simplified Lesk algorithm. The COMPUTEOVERLAP function returns the number of words in common between two sets, ignoring function words or other words on a stop list. The original Lesk algorithm defines the *context* in a more complex way.

# Word-in-Context Evaluation

---

- WSD as a kind of contextualized similarity task, since our goal is to be able to distinguish the meaning of a word like bass in one context (playing music) from another context (fishing).
- word-in-context task: System is given word-in-context two sentences, each with the same target word but in a different sentential context. The system must decide whether the target words are used in the same sense in the two sentences or in a different sense

- 
- F There's a lot of trash on the **bed** of the river —  
I keep a glass of water next to my **bed** when I sleep
  - F **Justify** the margins — The end **justifies** the means
  - T Air pollution — Open a window and let in some **air**
  - T The expanded **window** will give us time to catch the thieves —  
You have a two-hour **window** of clear weather to finish working on the lawn
- 

**Figure 19.11** Positive (T) and negative (F) pairs from the WiC dataset ([Pilehvar and Camacho-Collados, 2019](#)).

- The WiC sentences are mainly taken from the example usages for senses in WordNet.
- WordNet senses are very fine-grained. For this reason tasks like word-in-context first cluster the word senses into coarser clusters, so that the two sentential contexts for the target word are marked as T if the two senses are in the same cluster.
- WiC clusters all pairs of senses if they are first degree connections in the WordNet semantic graph

# Wikipedia as a source of training data

---



- Concept is mentioned in a Wikipedia: article text may contain an explicit link to the concept's Wikipedia page, which is named by a unique identifier (can be used as a sense annotation)
- For example, BAR (LAW), the page BAR (MUSIC), and so on, as in the following Wikipedia
  - *In 1834, Sumner was admitted to the [[bar (law)|bar]] at the age of twenty-three, and entered private practice in Boston.*
  - *It is danced in 3/4 time (like most waltzes), with the couple turning approx. 180 degrees every[[bar(music)|bar]].*
- These sentences can then be added to the training data for a supervised system.

# Wikipedia as a source of training data

---



- It is necessary to map from Wikipedia concepts to whatever inventory of senses is relevant for the WSD application.
  - Automatic algorithms that map from Wikipedia to WordNet
  - Ex: involve finding the WordNet sense that has the greatest lexical overlap with the Wikipedia sense, by comparing the vector of words in the WordNet synset, gloss, and related senses with the vector of words in the Wikipedia page title, outgoing links, and page category
  - The resulting mapping has been used to create BabelNet, a large sense-annotated resource.
-

# Using Thesauruses to Improve Embeddings

---

- Thesauruses have also been used to improve both static and contextual word embeddings.
- For example, static word embeddings have a problem with antonyms.
- A word like expensive is often very similar in embedding cosine to its antonym like cheap.

# Word Sense Induction

---

- Expensive and difficult to build large corpora in which each word is labeled for its word sense
- Word sense induction or WSI, is an important direction. In word sense induction unsupervised approaches, we don't use human-defined word senses.
- Instead, the set of “senses” of each word is created automatically from the instances of each word in the training set.

# Word Sense Induction

---

In training, we use three steps:

- For each token  $w_i$  of word  $w$  in a corpus, compute a context vector  $\mathbf{c}$
- Use a **clustering algorithm** to **cluster** these word-token context vectors  $\mathbf{c}$  in to a predefined number of groups or clusters. Each cluster defines a sense of  $w$ .
- Compute the **vector centroid** of each cluster. Each vector centroid  $\mathbf{s}_j$  is a **sense vector** representing that sense of  $w$ .
- We don't have names for each of these "senses" of  $w$ ; we just refer to the  $j$ th sense of  $w$ .

# Word Sense Induction

---

To disambiguate a particular token  $t$  of  $w$  we again have three steps:

1. Compute a context vector  $c$  for  $t$ .
2. Retrieve all sense vectors  $s_j$  for  $w$ .
3. Assign  $t$  to the sense represented by the sense vector  $s_j$  that is closest to  $t$ .

All we need is a clustering algorithm and a distance metric between vectors. Ex: Agglomerative clustering

# Agglomerative clustering

---

- Each of the  $N$  training instances is initially assigned to its own cluster.
- New clusters are then formed in a bottom-up fashion by the successive merging of the two clusters that are most similar.
- This process continues until either a specified number of clusters is reached, or some global goodness measure among the clusters is achieved.
- In cases no. of training instances makes method too expensive, random sampling can be used on the original training set to achieve similar results.

# References

---

- <https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270>
- <https://wordnet.princeton.edu/>
- Chapter 19 Jurafsky and Martin



---

# Thank You



# Natural Language Processing

## DSECL ZG565



**BITS** Pilani  
Pilani Campus

Prof.Vijayalakshmi  
BITS -Pilani

# Session Content

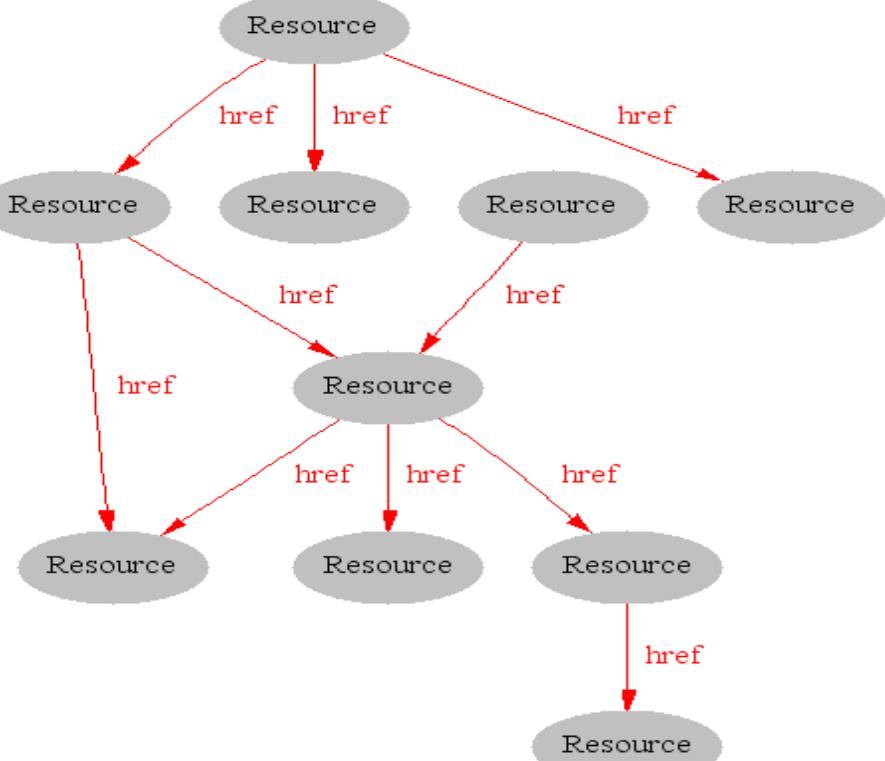
- Introduction
- Semantic Web and Ontology
- Ontology Languages
- Ontology Engineering
- State of the Art
  - Semantic Web and web services
  - Ontology Learning
  - Linked open data
  - Knowledge Graphs

# Where we are Today: The Syntactic Web

innovate

achieve

**lead**



[Hendler & Miller 02]

# Is it possible to get answers for these using syntactic web?



- Complex queries involving **background knowledge**
  - Find information about “animals that use sonar but are not either bats or dolphins”
  - Another example

Which articles on telemarketing were written in 2007 by authors working in companies with less than 100 employees?

1. Find all articles in bibliographic databases or on the web dealing with telemarketing written in 2007.
2. Make a list of the affiliations of the authors of those articles, if available, and check the names of the authors of whom no affiliation is mentioned, in bibliographic databases, in databases about companies, or on the web.
3. Check which of the firms you found this way have less than 100 employees, using economical databases, the websites of those firms, etc.
4. Make a list of all results.

# What is the Problem?

Consider a typical web page:



The screenshot shows the homepage of the WWW 2002 conference. At the top, it features the URL <http://www2002.org>, the title "WWW 2002", and the subtitle "THE ELEVENTH INTERNATIONAL WORLD WIDE WEB CONFERENCE". Below this, there's a logo for "Hawaii 2002" and information about the location: "Sheraton Waikiki Hotel, Honolulu, Hawaii, USA" and the dates "7-11 May 2002". A tagline "1 LOCATION. 5 DAYS. LEARN. INTERACT." is displayed. On the left, a sidebar contains links to "Conference Proceedings", "Call for Participation", "Program", "Registration Information", "Hotel Accommodation", "Conference Committee", "Sponsorship/Exhibition Opportunities", "Volunteer Information", "Information about Hawaii", and "Previous & Future WWW Conferences". The main content area includes sections for "Registered participants coming from:" (listing countries like Australia, Canada, Chile, etc.) and "FEATURED SPEAKERS (CONFIRMED)" (listing Tim Berners-Lee, Richard A. DeMillo, Ian Foster, and others). A "REGISTER NOW" button is prominently displayed.

- Markup consists of:
  - rendering information (e.g., font size and colour)
  - Hyper-links to related content
- Semantic content is accessible to humans but not (easily) to computers...

[Davies, 03]

# What information can we see...

WWW2002

The eleventh international world wide web conference

Sheraton waikiki hotel

Honolulu, hawaii, USA

7-11 may 2002

1 location 5 days learn interact

Registered participants coming from

australia, canada, chile denmark, france, germany, ghana, hong kong, india, ireland, italy, japan, malta, new zealand, the netherlands, norway, singapore, switzerland, the united kingdom, the united states, vietnam, zaire

Register now

On the 7<sup>th</sup> May Honolulu will provide the backdrop of the eleventh international world wide web conference. This prestigious event ...

Speakers confirmed

Tim berners-lee

Tim is the well known inventor of the Web, ...

Ian Foster

Ian is the pioneer of the Grid, the next generation internet ...

# What information can a machine see...

# Data on the Web

---

**The data on the Web increases every second**

- government data, health related data, general knowledge, company information, flight information, restaurants,...

**More and more applications rely on the availability of that data**

# But... data are often in isolation, “silos”



# Imagine...

---

## A “Web” where

- documents are available for download on the Internet
- but there would be no hyperlinks among them

# And the problem is real...

**CoCoDat - Collation of Cortical Data - Mozilla Firefox**

File Edit View History Bookmarks Tools Help

CoCoMac DATABASES ORT EXAMPLES

**CoCoDat: Collation of Cortical [sic] microcircuitry] Data**

CoCoDat is a microcircuitry database that published experimental reports. The data and cellular compartment), as well as the

- Morphology
- Firing properties
- Ionic currents
- Ionic conductances
- Synaptic currents
- Connectivity

The database is available for download u data tables but also a Search Board with manual or automatic relaxation of the sea

- Brain region
- Layer
- Neuron type

<http://www.cocomac.org/cocodat/catalyzer/index.html>

**Cell Centered Database - Mozilla Firefox**

File Edit View History Bookmarks Tools Help

http://ccdb.ucsd.edu/sand/main?event=gallery&action=show&cdp=y

**Cell Centered Database™**  
National Center for Microscopy and Imaging Research

**Gallery**

Data | Search | Gallery | Dictionary | Publications | MyCCDB | Data Download | Contact us | Help

2D image Reconstruction Segmentation Animation

**NeuronDB - Thalamic relay neuron - Overview (A) () - Mozilla Firefox**

File Edit View History Bookmarks Tools Help

http://senselab.med.yale.edu

**NeuronDB**

Back Thalamic relay neuron

Mode: **Overview** Data/Search plus Connectivity plus Classical References/Notes Models

Region: Distal equivalent dendrite Middle equivalent dendrite Proximal equivalent dendrite Soma Axon hillock Axon fiber Axon terminal All Compartments

Properties: Receptors Channels Transmitters **All Properties**

Interoperation: Gene and Chromosome Experimental Data (neurodatabase.org) Microscopy Data (CCDB)

Neuron type: principal Organism: Vertebrates

1. Equivalent dendrite Show other  
2. Distal equivalent dendrite Show other  
3. Middle equivalent dendrite Show other  
4. Proximal equivalent dendrite Show other  
5. Soma Show other

# Data on the Web is not enough...

---

We need a proper infrastructure for a real Web of Data

- data is available on the Web
  - accessible via standard Web technologies
- data are interlinked over the Web
  - ie, data can be integrated over the Web

This is where Semantic Web technologies come in

# The Web of Data

A vision of the Web as a huge database  
Can we query the Web as a database?  
What were the most popular songs when Obama  
was elected?

```
SELECT ?song, COUNT(?vote) AS ?likes FROM
WorldWideWeb WHERE (?vote in FavourOf ?song) AND
(?vote madeAt ?date) AND (Obama elected At ?date)
ORDER BY ?likes DESC
```

# A Web of Data unleashes new applications

Get the latest public health information from CDC: <https://www.coronavirus.gov>  
Get the latest research information from NIH: <https://www.nih.gov/coronavirus>

**Medical Subject Headings**

[MeSH Home](#) | [Learn About MeSH](#) | [MeSH Browser](#) | [Download MeSH Data](#) | [MeSH on Demand](#) | [Suggestions](#)

[Home](#)

## Welcome to Medical Subject Headings

The Medical Subject Headings (MeSH) thesaurus is a controlled and hierarchically-organized vocabulary produced by the National Library of Medicine. It is used for indexing, cataloging, and searching of biomedical and health-related information. MeSH includes the subject headings appearing in MEDLINE/PubMed, the NLM Catalog, and other NLM databases.

### What's New

Visit our [What's New](#) page to see all recent MeSH developments including the most recent ones listed below

- [2020 MeSH files are now in production](#)
  - The MeSH Browser now displays [2020 MeSH](#) and [2021 MeSH Preview Version](#) vocabularies
  - All 2020 MeSH files are now available via FTP download
- [MeSH in Resource Description Format\(RDF\)](#) is now in production
  - The [downloadable files](#) contain a full representation of XML MeSH in RDF format
  - An [open MeSH API](#) is available for retrieving MeSH data
  - You can use our [SPARQL query editor](#) for querying MeSH data

### Learn About MeSH

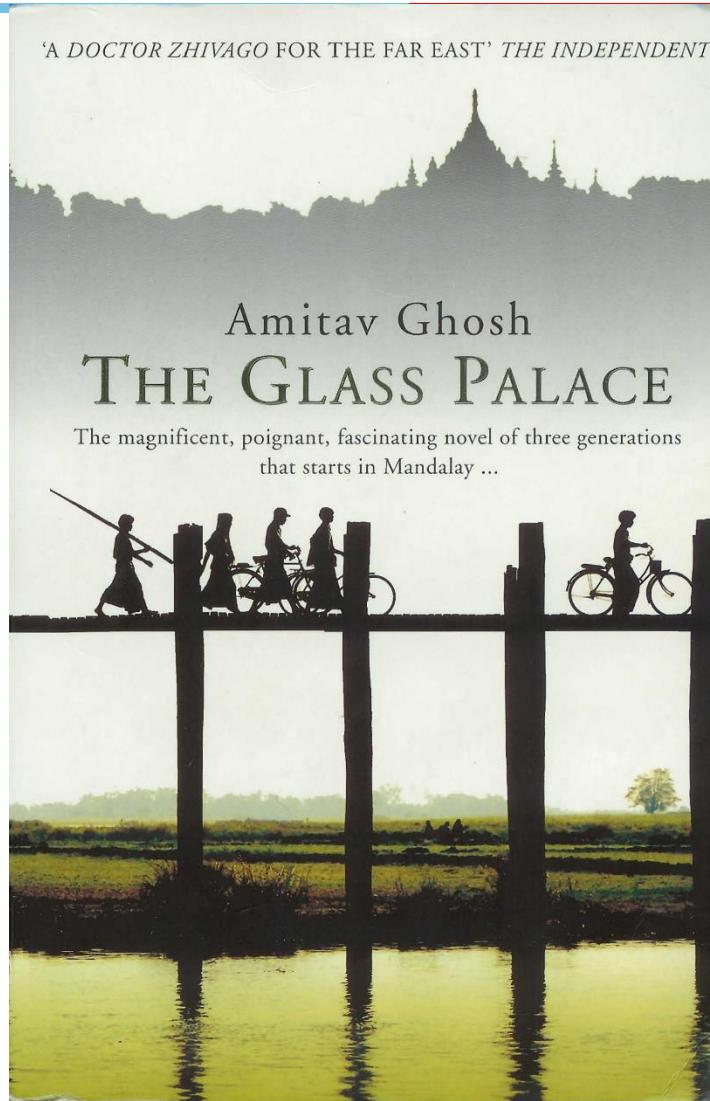
- [Tutorials and Webinars](#)
- [MeSH Vocabulary](#)
  - [Introduction to MeSH](#)
  - [Browser Instructions](#)
  - [Finding Keywords for Publications](#)
  - [MeSH Qualifiers List](#)
  - [MeSH Qualifiers with Scope Notes](#)
  - [Publication Characteristics \(Publication Types\) with Scope Notes](#)
- [Search and Retrieval using MeSH](#)
  - [Cataloging with MeSH Terminology](#)

# The rough structure of data integration

---

1. Map the various data onto an abstract data representation
  - .make the data independent of its internal representation...
2. Merge the resulting representations
3. Start making queries on the whole!
  - .queries that could not have been done on the individual data sets

# We start with a book...



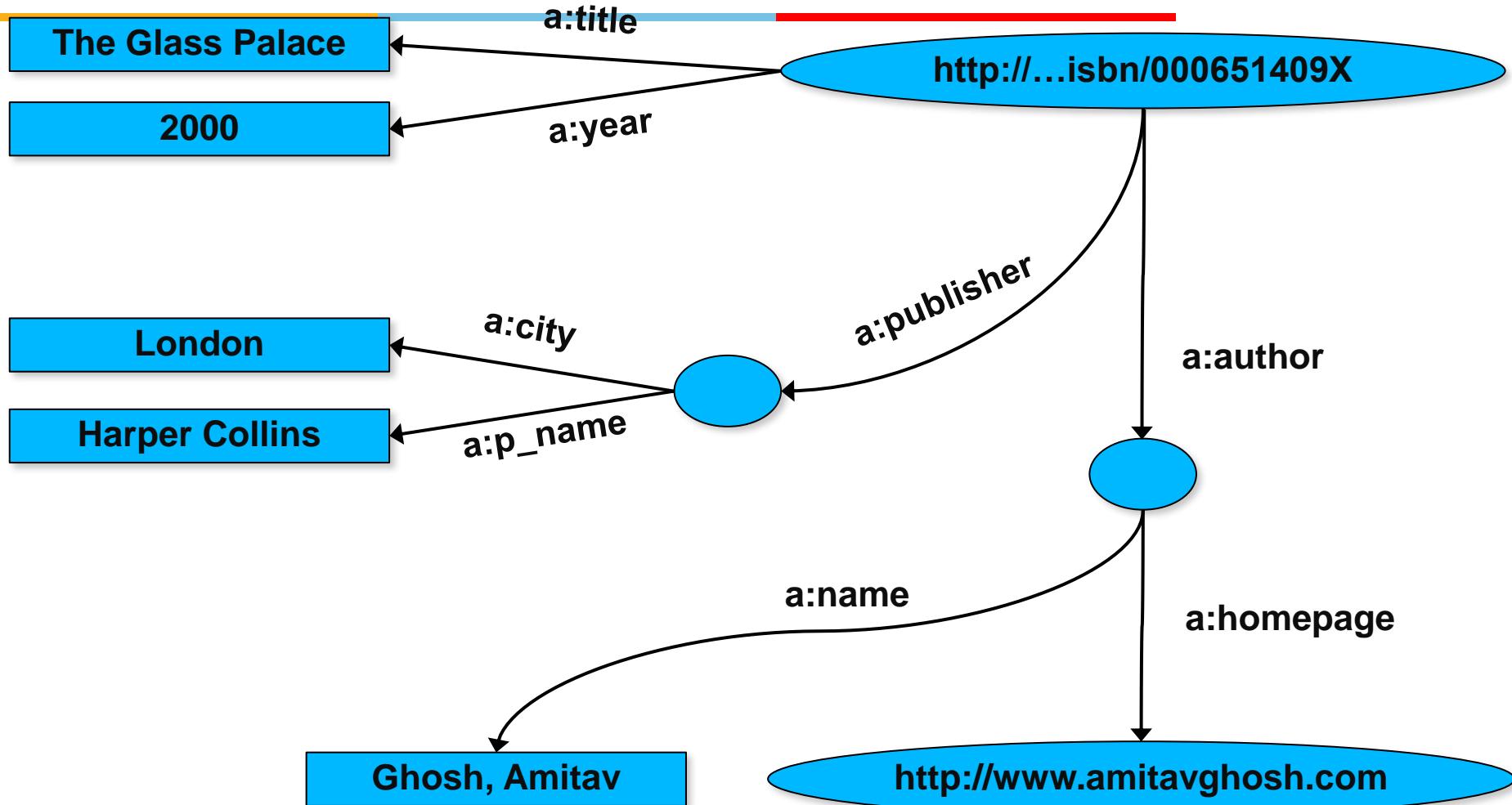
# A simplified bookstore data (dataset “A”)

<b>ID</b>	<b>Author</b>	<b>Title</b>	<b>Publisher</b>	<b>Year</b>
ISBN 0-00-6511409-X	id_xyz	The Glass Palace	id_qpr	2000

<b>ID</b>	<b>Name</b>	<b>Homepage</b>
id_xyz	Ghosh, Amitav	<a href="http://www.amitavghosh.com">http://www.amitavghosh.com</a>

<b>ID</b>	<b>Publisher's name</b>	<b>City</b>
id_qpr	Harper Collins	London

# 1<sup>st</sup>: export your data as a set of relations



# Some notes on the exporting the data

---

## Relations form a graph

- the nodes refer to the “real” data or contain some literal
- graph is represented in machine using semantic web technologies

# Some notes on the exporting the data

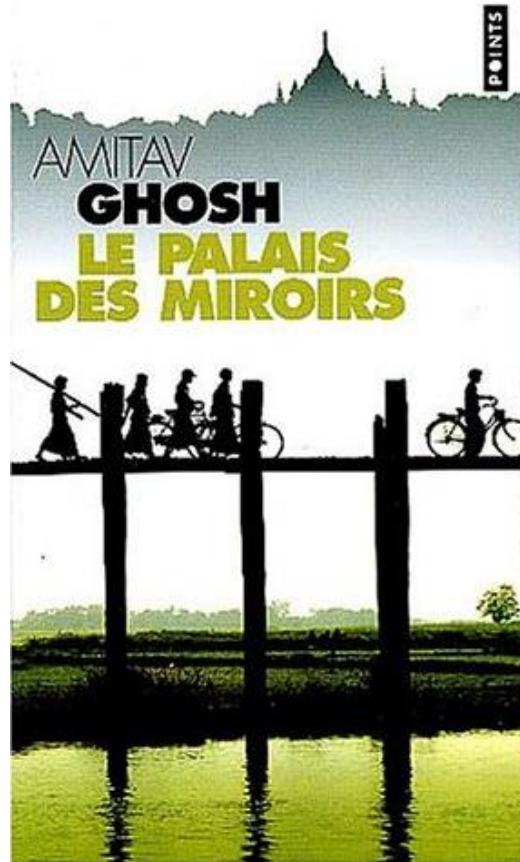
---

**Data export does not necessarily mean physical conversion of the data**

- relations can be generated on-the-fly at query time
  - via SQL “bridges”
  - scraping HTML pages
  - extracting data from Excel sheets
  - etc.

**One can export part of the data**

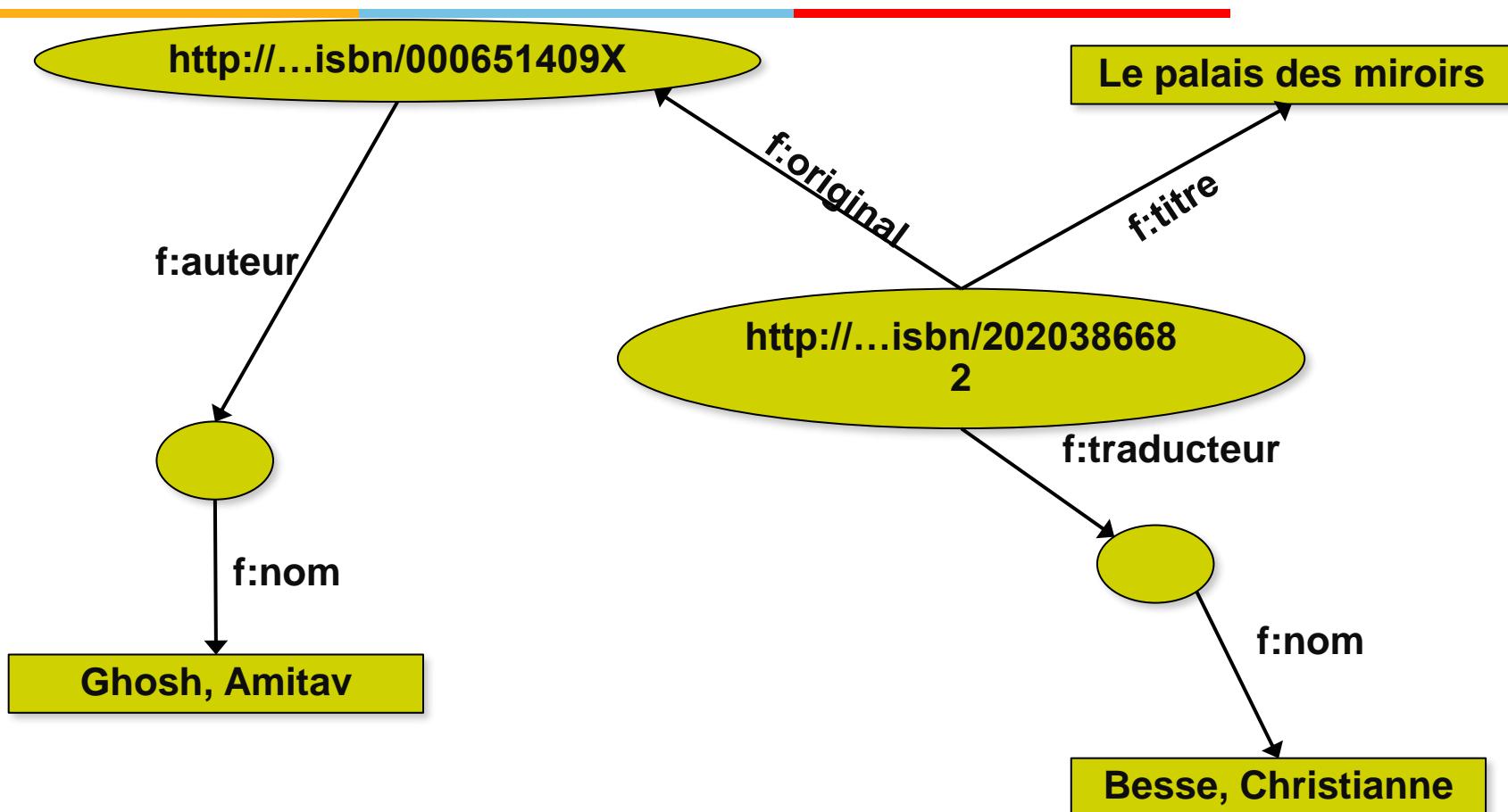
# Same book in French...



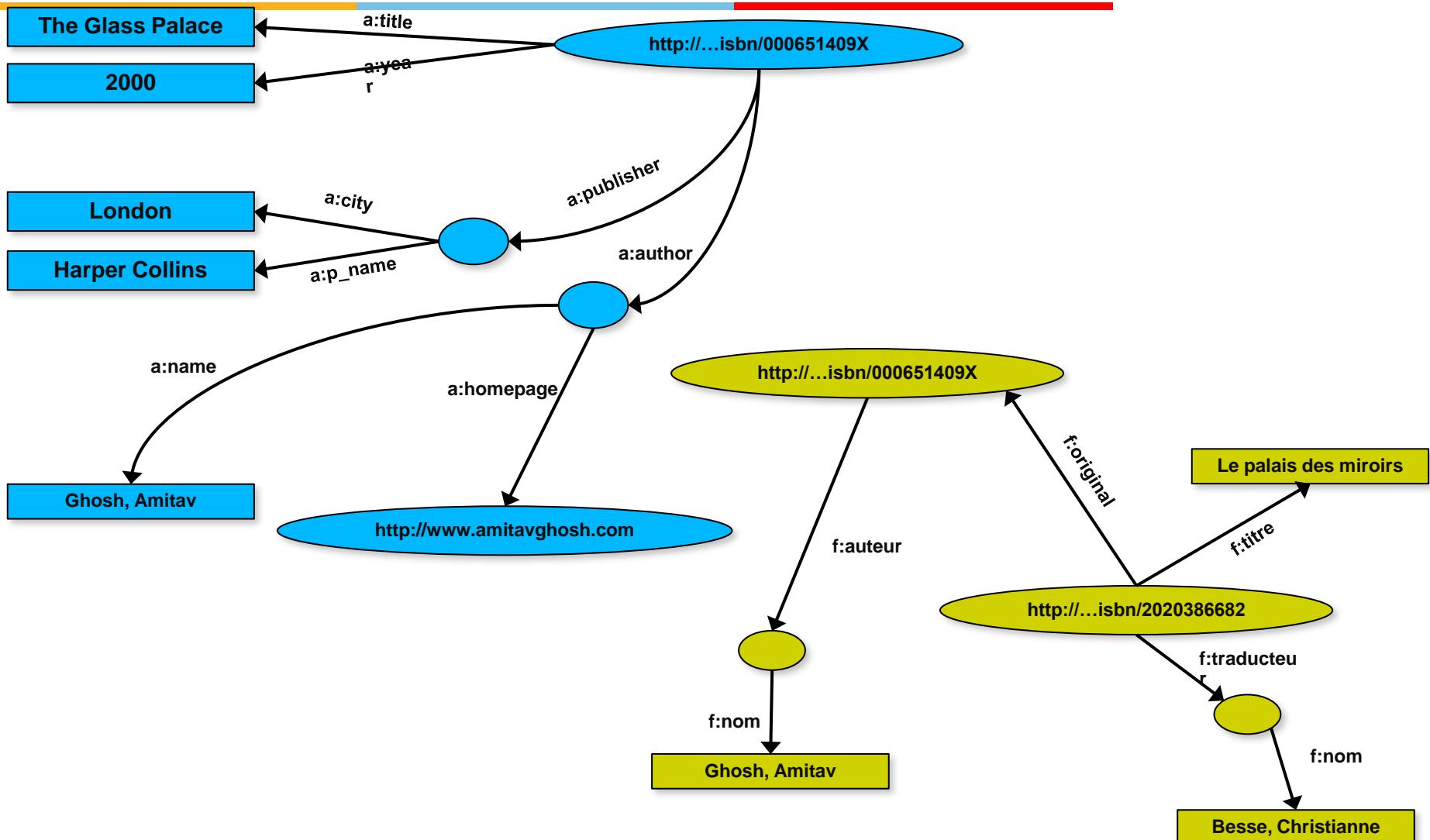
# Another bookstore data (dataset “F”)

A	B	C	D	
1	ID	Titre	Traducteur	Original
2	ISBN 2020286682	Le Palais des Miroirs	\$A12\$	ISBN 0-00-6511409-X
3				
4				
5				
6	ID	Auteur		
7	ISBN 0-00-6511409-X	\$A11\$		
8				
9				
10	Nom			
11	Ghosh, Amitav			
12	Besse, Christianne			

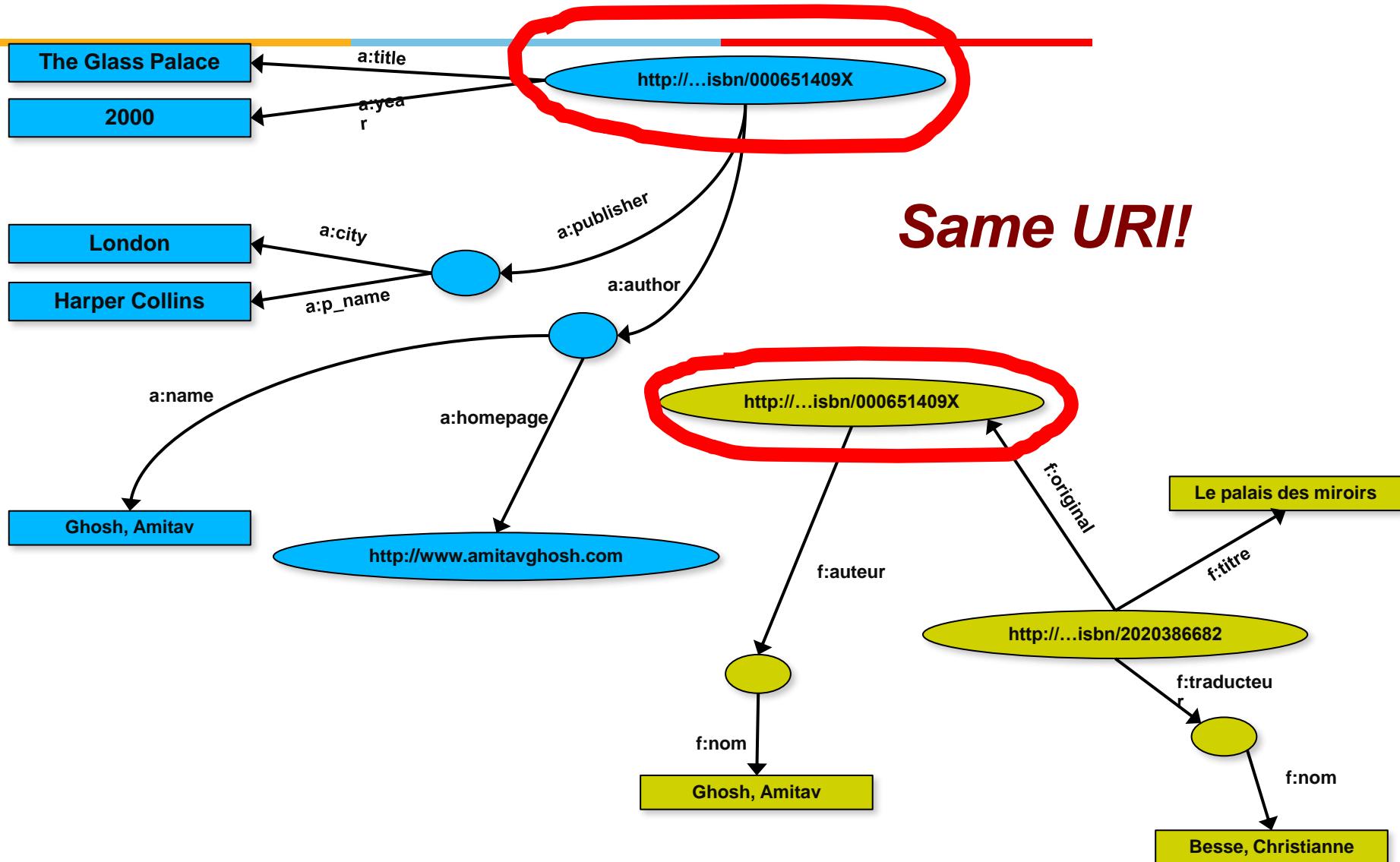
# 2<sup>nd</sup>: export your second set of data



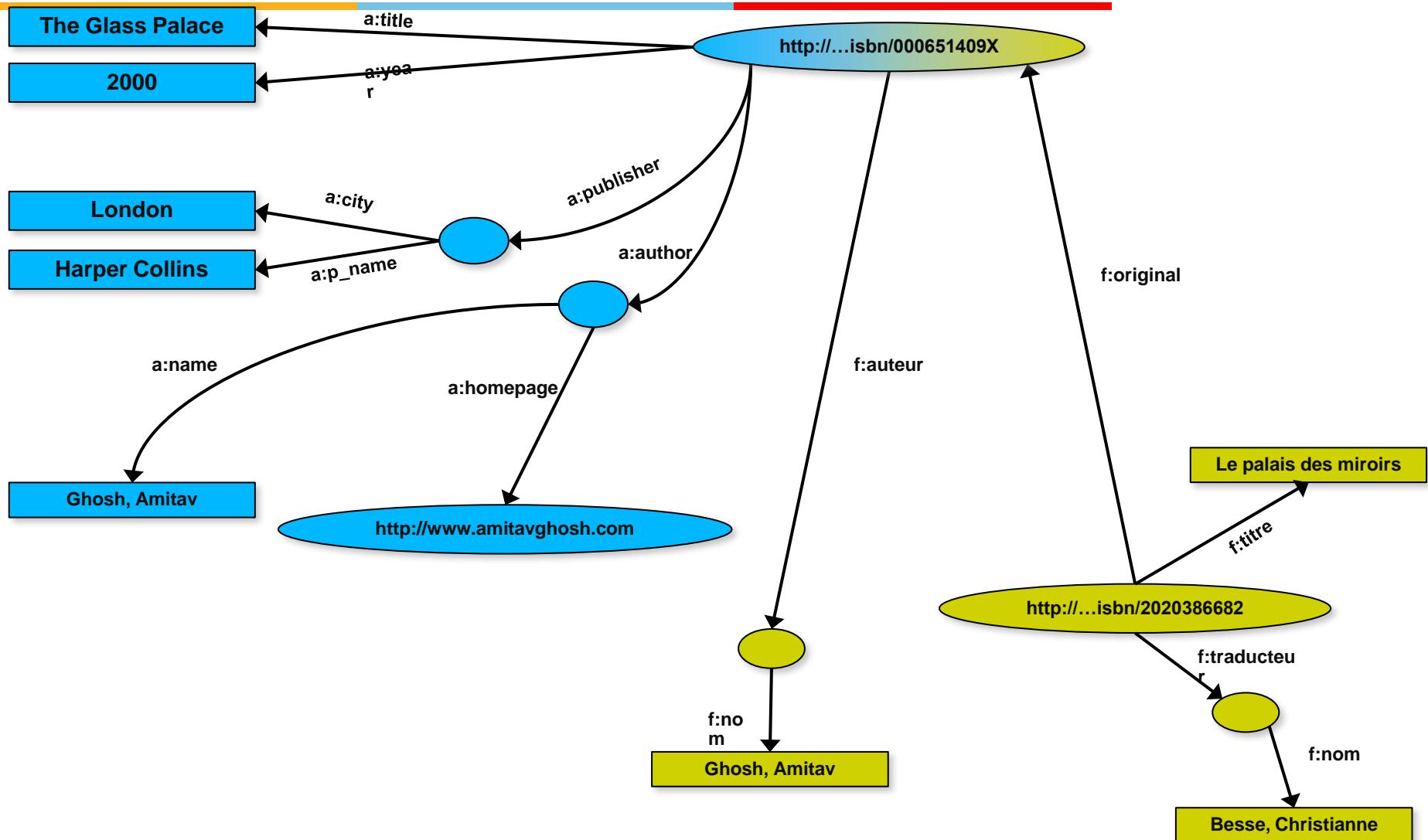
# 3<sup>rd</sup>: start merging your data



# 3<sup>rd</sup>: start merging your data (cont)



# 3<sup>rd</sup>: start merging your data



# Start making queries...

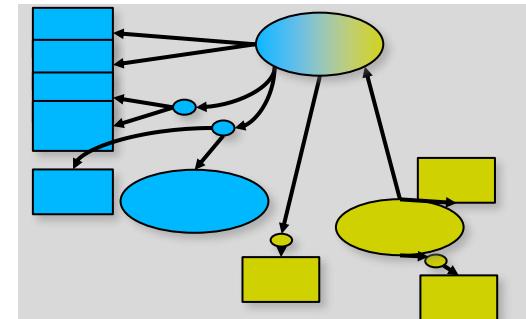
---

**User of data “F” can now ask queries like:**

- “give me the title of the original”
  - well, ... « donnez-moi le titre de l’original »

**This information is not in the dataset “F”...**

**...but can be retrieved by merging with  
dataset “A”!**



# However, more can be achieved...

---

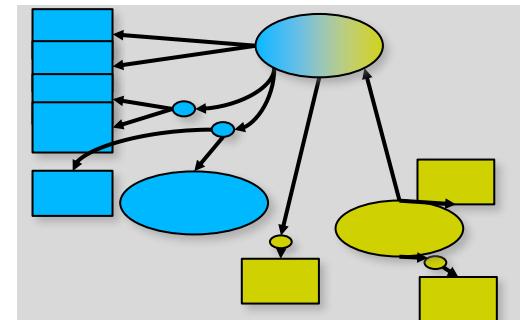
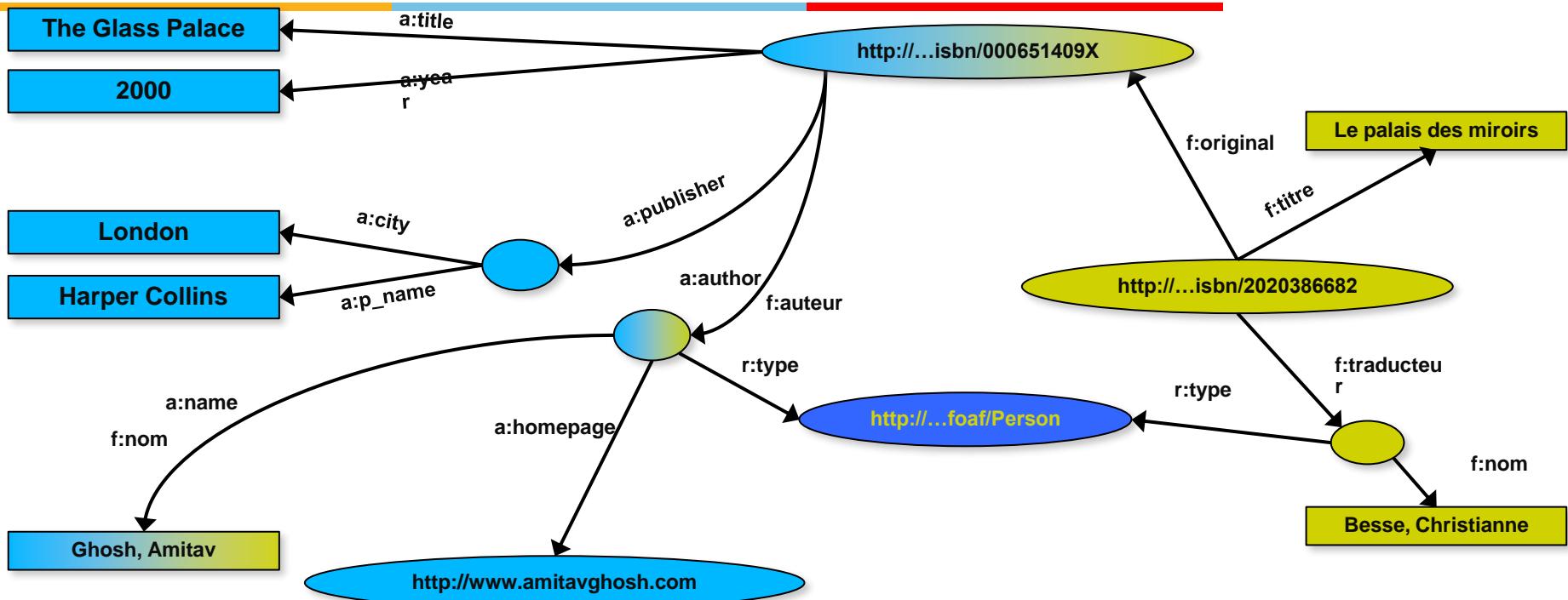
We “feel” that a:author and f:auteur should be the same

But an automatic merge does not know that!

Let us add some extra information to the merged data:

- a:author same as f:auteur
- both identify a “Person”
- a term that a community may have already defined:
  - a “Person” is uniquely identified by his/her name and, say, homepage
  - it can be used as a “category” for certain type of resources

# 3<sup>rd</sup> revisited: use the extra knowledge



# Start making richer queries!

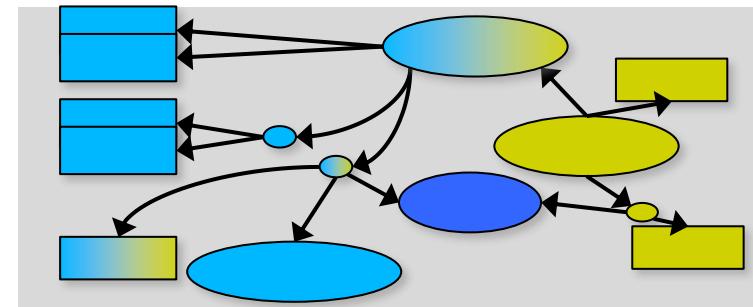
**User of dataset “F” can now query:**

- “donnes-moi la page d'accueil de l'auteur de l'original”
  - well... “give me the home page of the original's ‘auteur’”

**The information is not in datasets “F” or “A”...**

**...but was made available by:**

- merging datasets “A” and datasets “F”
- adding three simple extra statements as an extra “glue”



# Combine with different datasets

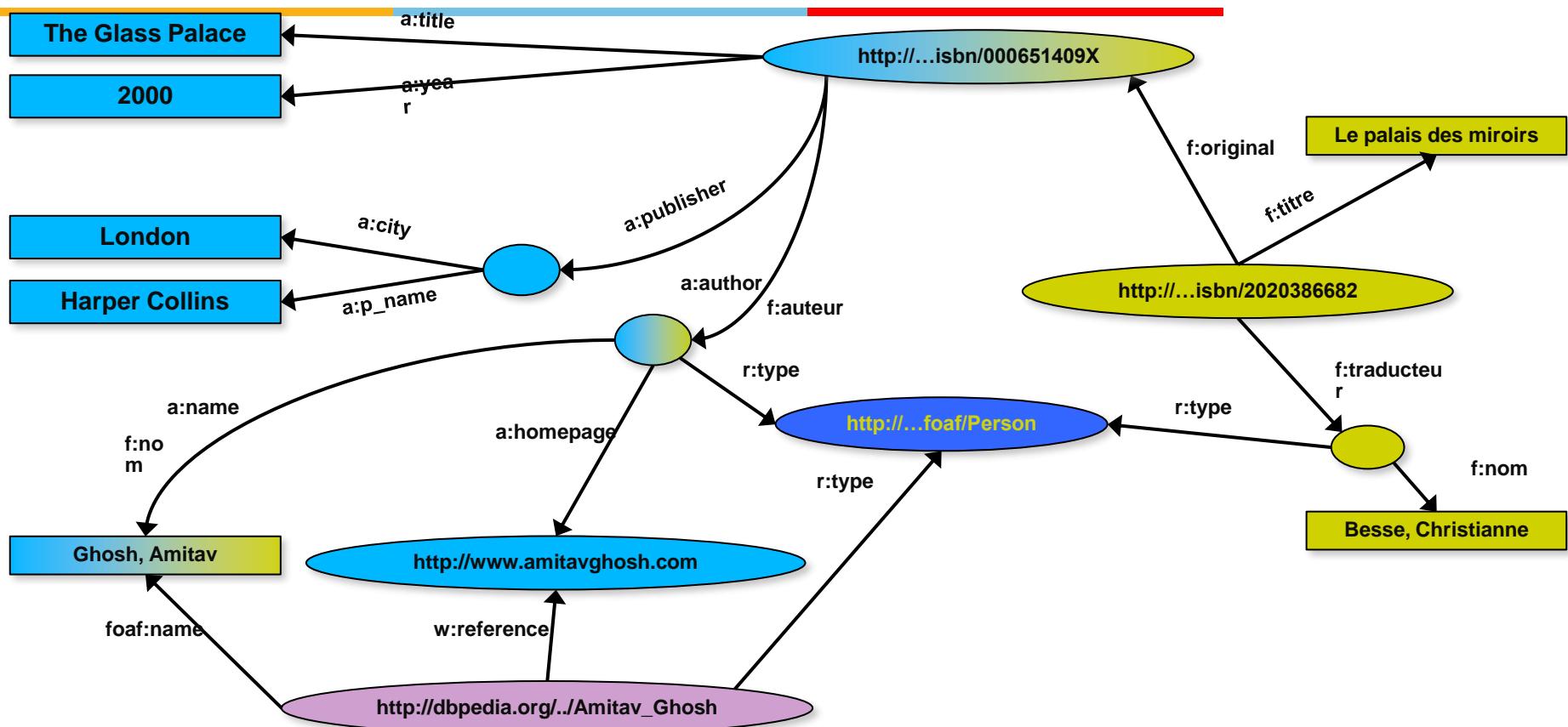
---

**Using, e.g., the “Person”, the dataset can be combined with other sources**

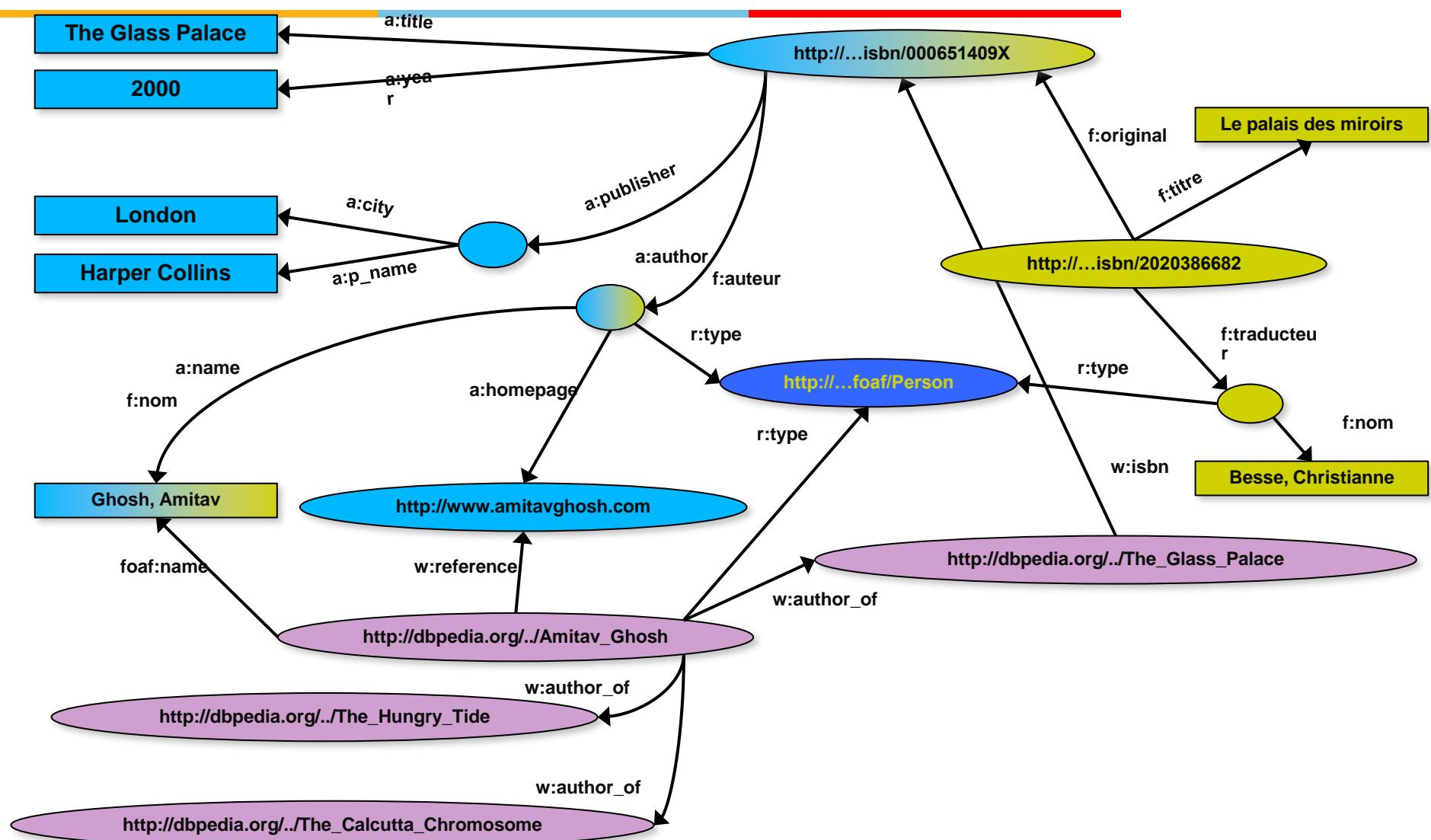
**For example, data in Wikipedia can be extracted using dedicated tools**

- e.g., the “[dbpedia](#)” project can extract the “infobox” information from Wikipedia already...

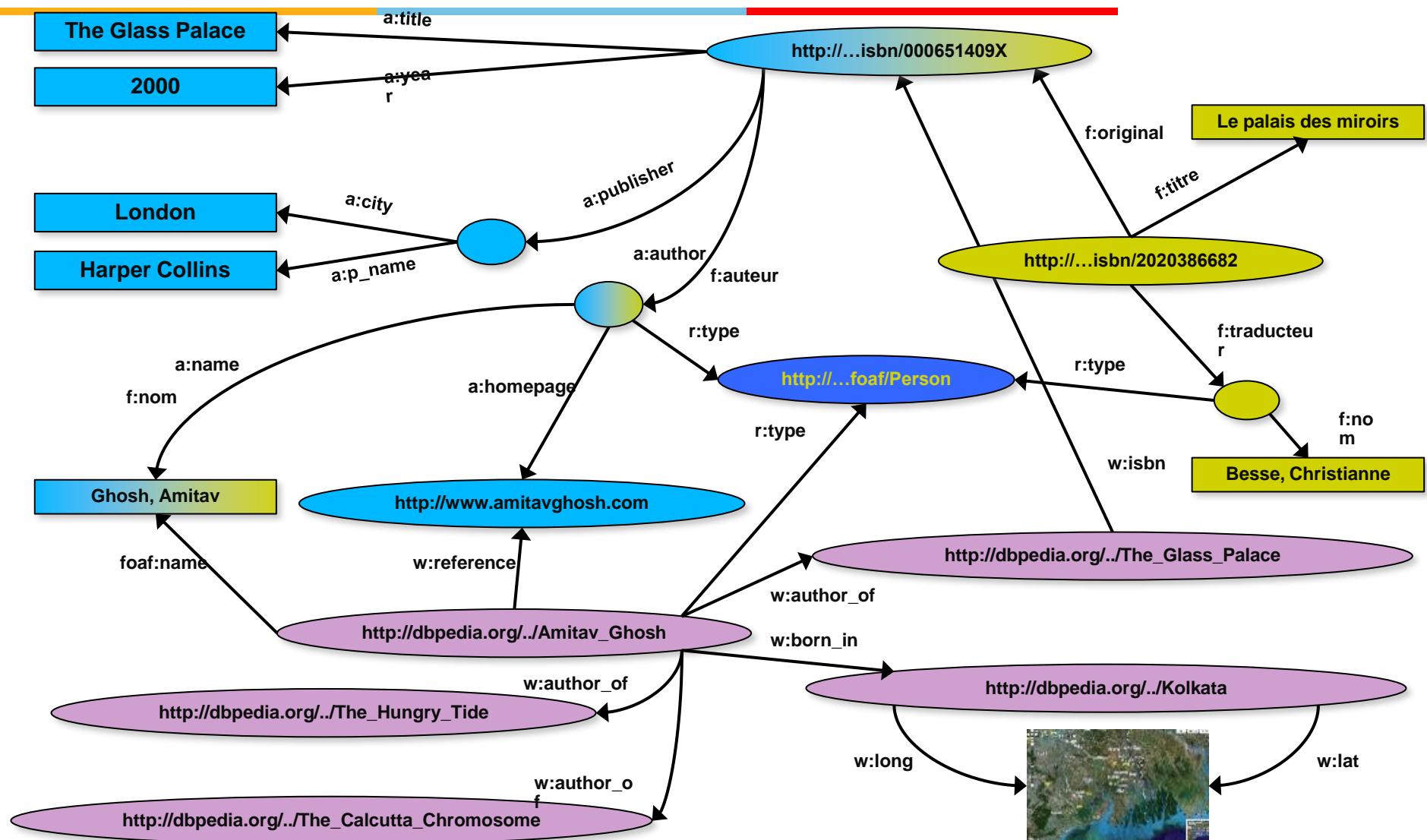
# Merge with Wikipedia data



# Merge with Wikipedia data



# Merge with Wikipedia data



# Is that surprising?

---

**It may look like easy but, in fact, it should not be...**

**What happened via automatic means is done every day by Web users!**

**The difference: a bit of extra rigour so that machines could do this, too**

# It could become even more powerful

---

We could add extra knowledge to the merged datasets

- e.g., a full classification of various types of library data
- geographical information etc

This is where ontologies come in

- ontologies sets can be relatively simple and small, or huge, or anything in between...

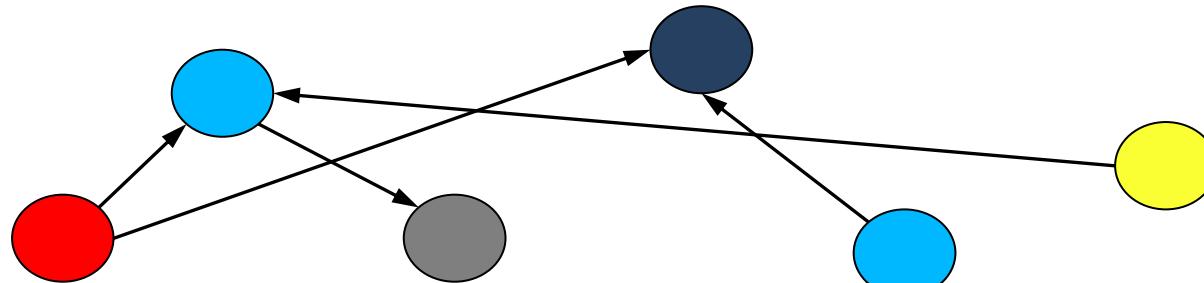
Even more powerful queries can be asked as a result

# What did we do?

**Applications**

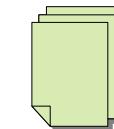
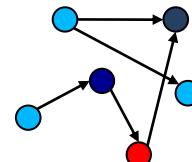
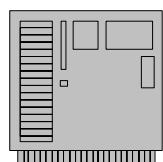


Manipulate  
Query  
...



**Data represented in abstract format**

Map,  
Expose,  
...



**Data in various formats**



# So where is the Semantic Web?

**The Semantic Web provides technologies to make such integration possible!**

# The Semantic Web Vision

“... the idea of having data on the Web defined and linked in a way that it can be used by machines not just for display purposes ,but for automation, integration and reuse of data across various applications”



<http://www.w3.org/sw/>



# Semantic Web

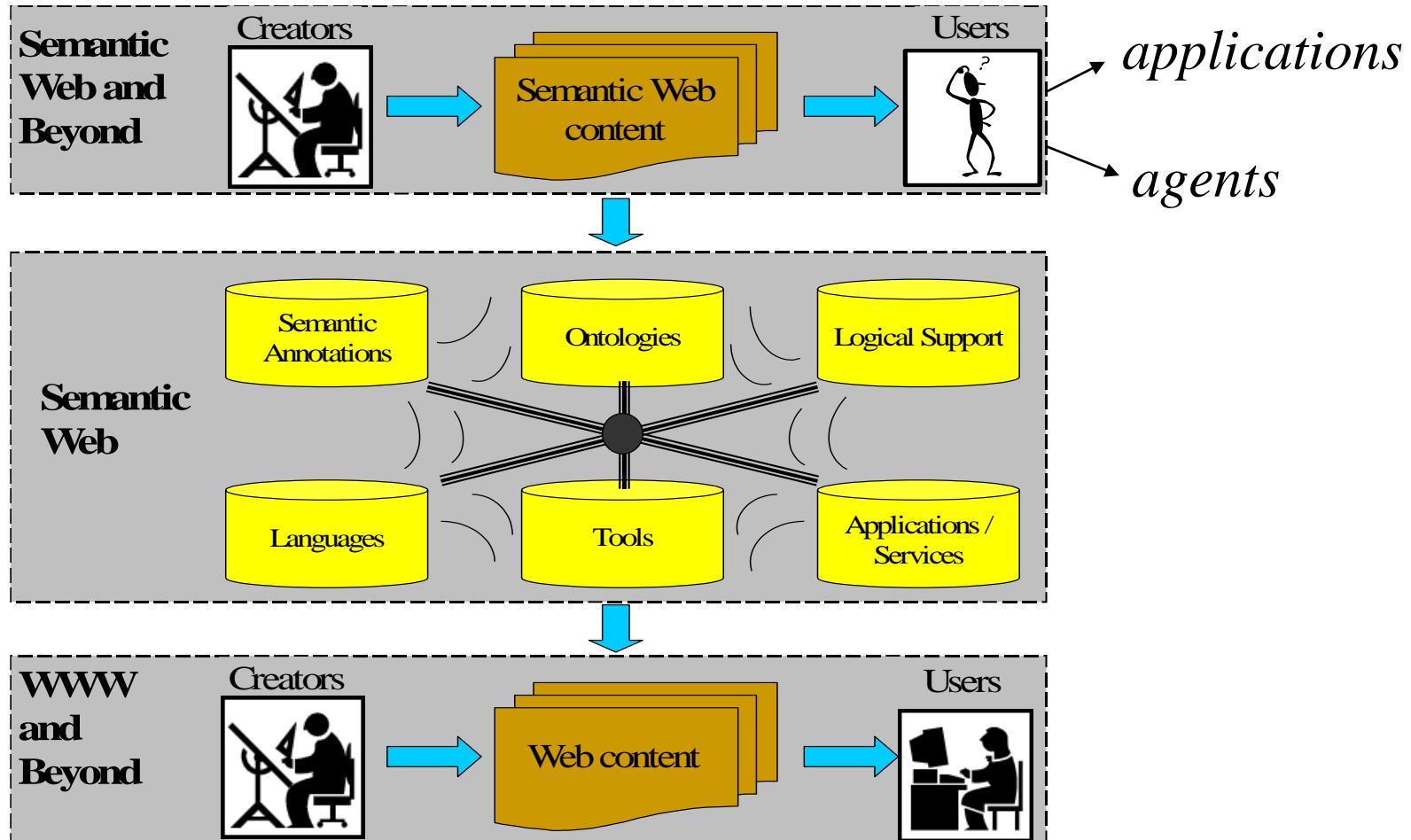
---

Tim Berner Lee's Vision:

- ◆ Web as a means of collaboration for people
- ◆ Web as a means of collaboration for machines

Semantic Web is a web of data that machines can “understand” too.

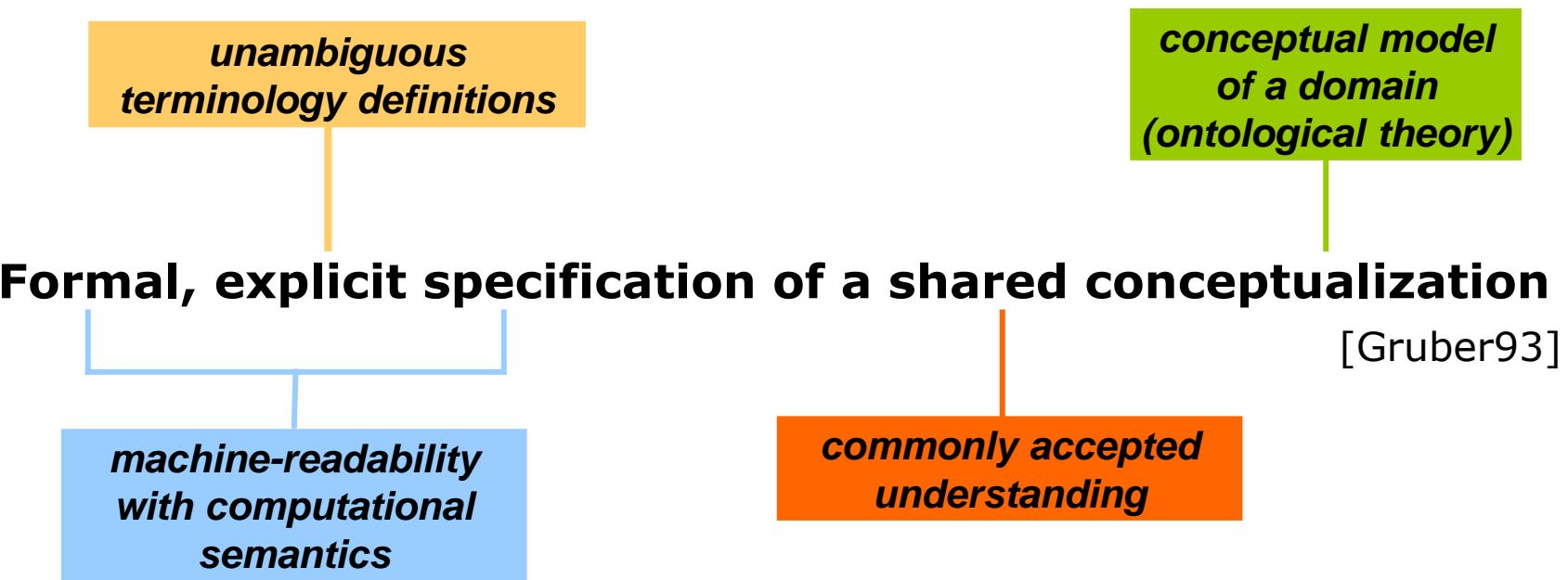
# Semantic Web – New Users



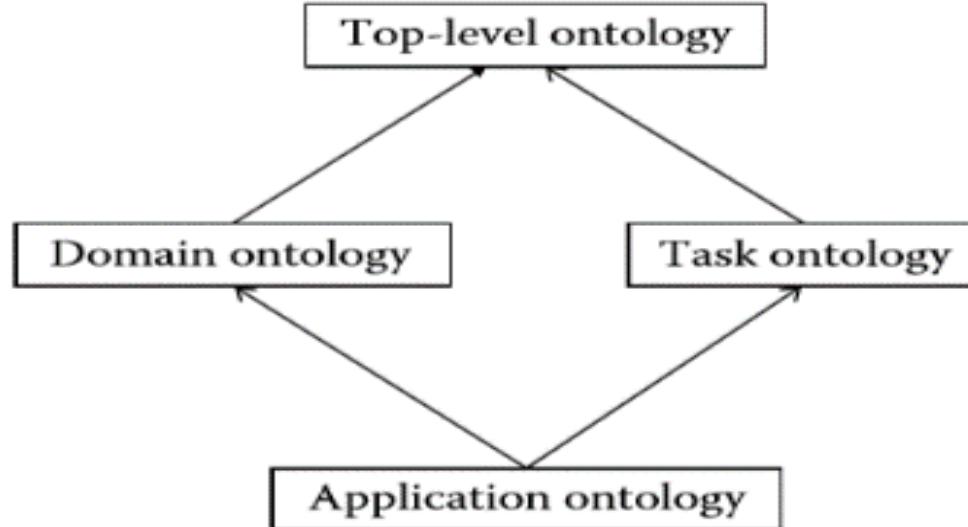
# Need to Add “Semantics”

- Use **Ontologies** to specify meaning of annotations
  - Ontologies provide a vocabulary of terms
  - New terms can be formed by combining existing ones
  - Meaning (**semantics**) of such terms is formally specified
  - Can also specify relationships between terms in multiple ontologies

# Ontology Definition



# Types of Ontologies



# Application Areas of Ontologies

---

## Information Retrieval

As a tool for intelligent search instead of keyword matching

Cross Language Information Retrieval

Improve recall by query expansion through the synonymy

Improve precision through Word Sense Disambiguation

(identification of the relevant meaning of a word in a given context among all its possible meanings)

## Information Integration

Seamless integration of information from different websites and databases

## Knowledge Engineering and Management

As a knowledge management tools for selective semantic access (meaning oriented access)

## Natural Language Processing

Better machine translation

Queries using natural language

# Structure of an Ontology

---

**There are three main components to an ontology, which are usually described as follows:**

**Classes: the distinct types of things that exist in our data.**

**Relationships: properties that connect two classes.**

**Attributes: properties that describe an individual class.**

# Ontology Example

Books

Title	Author	Publisher	Year Published	Followed By
To Kill a Mockingbird	Harper Lee	J. B. Lippincott Company	1960	Go Set a Watchman
Go Set a Watchman	Harper Lee	HarperCollins, LLC; Heinemann	2015	
The Picture of Dorian Gray	Oscar Wilde	J. B. Lippincott & Co.	1890	
2001: A Space Odyssey	Arthur C. Clarke	New American Library, Hutchinson	1968	

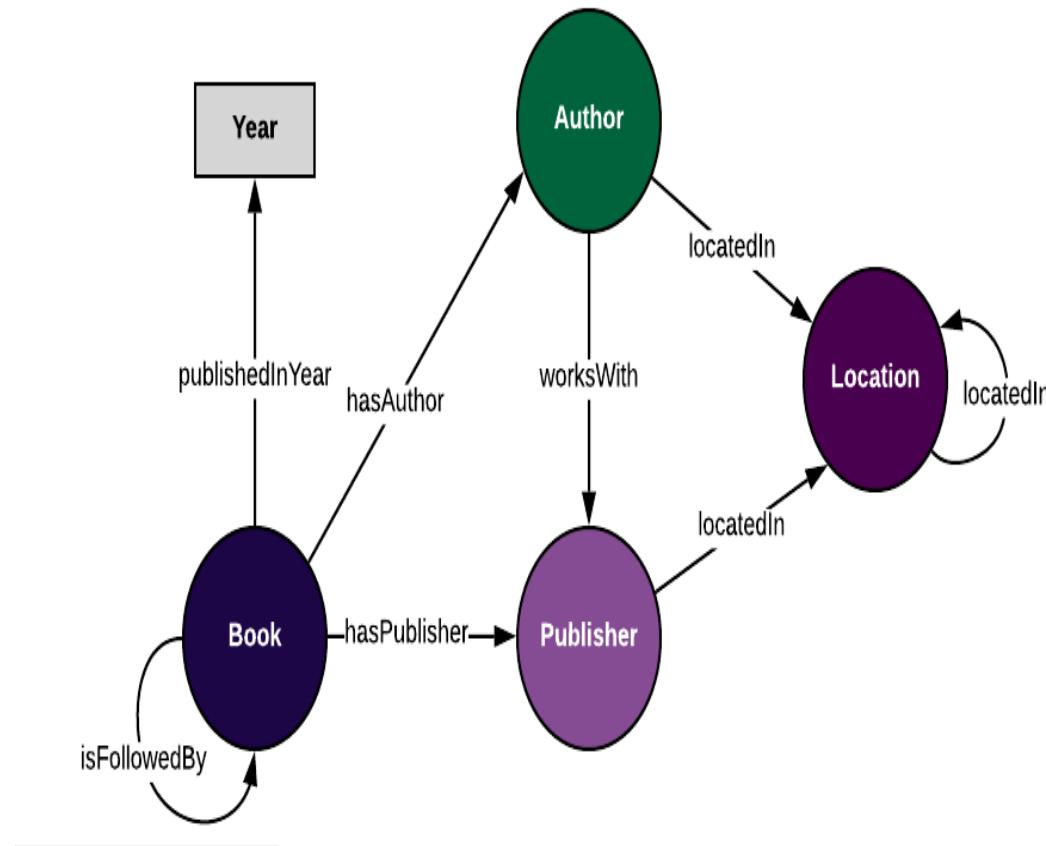
## Publishers

Name	City	Country
J. B. Lippincott & Company	Philadelphia	United States
HarperCollins, LLC	New York City	United States
Heinemann	Portsmouth	United States
New American Library	New York City	United States
Hutchinson	London	United Kingdom

## Authors

Name	Country of Birth
Harper Lee	United States
Oscar Wilde	Ireland
Arthur C. Clarke	United Kingdom

# Contd..

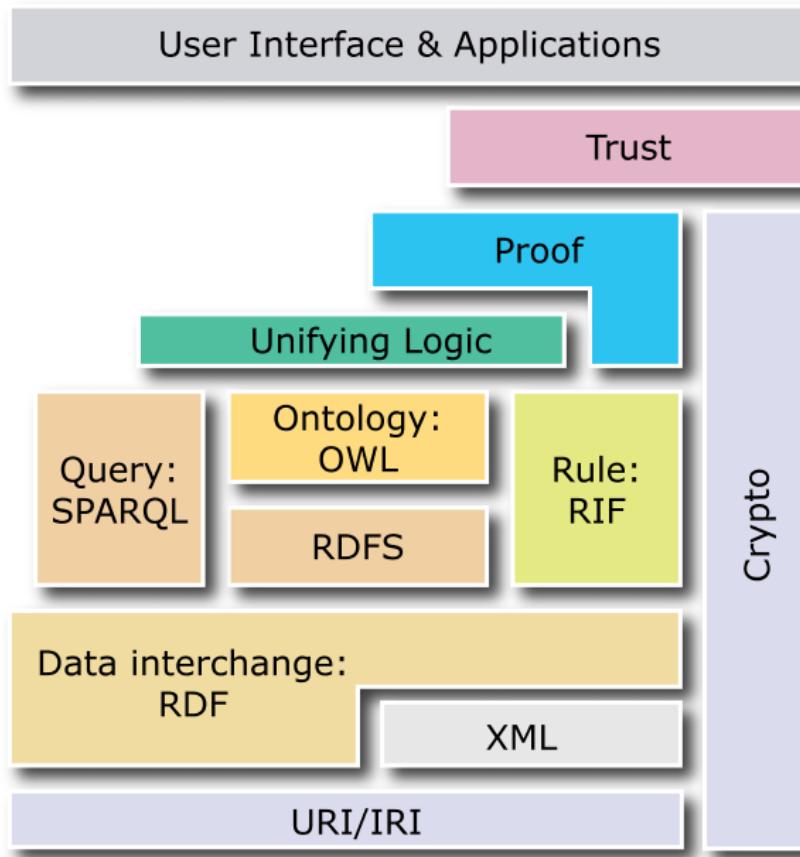


- Ontology languages

**For the purpose of formalizing ontologies, variants of first-order logic with standard model-theoretic semantics are used**

- **RDF (Resource Description Framework)**
  - Specifies relationship between data
- **RDFS(Resource Description Framework Schema)**
  - Specifies relationship between schema
- **OWL (Web Ontology Language)**
  - Specifies more complex relationship between schema based on description logics

# Semantic Web Layers



**URI/IRI**

Universal Resource Identifier  
Internationalized Resource Identifier

**XML**

eXtended Markup Language

**RDF**

Resource Description Framework

**RDFS**

RDF Schema

**OWL**

Web Ontology Language

**SPARQL**

Simple Protocol and RDF Query Language

# RDF and RDFS

---

- **RDF stands for Resource Description Framework**
- **is a W3C standard, which provides tool to describe Web resources**
- **provides interoperability between applications that exchange machine-understandable information**

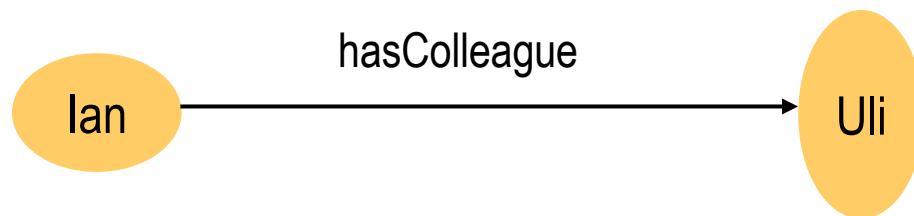
# RDF and RDFS

---

- **RDFS extends RDF with “schema vocabulary”, e.g.:**
  - Class, Property
  - type, subClassOf, subPropertyOf
  - range, domain

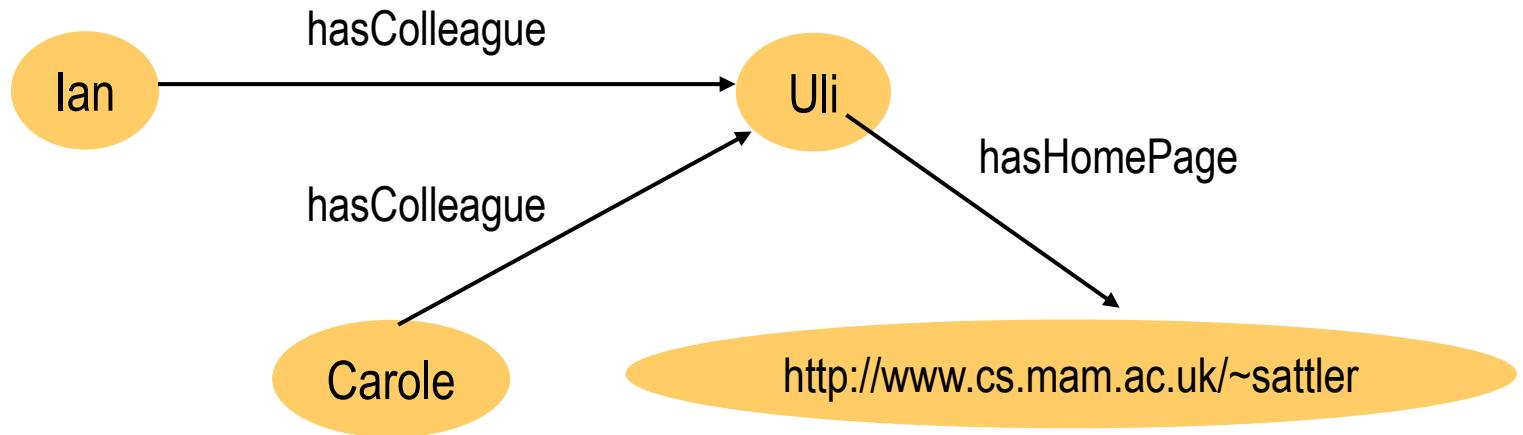
# The RDF Data Model

- Statements are  
**<subject, predicate, object> triples:**  
    <Ian, hasColleague, Uli>
- Can be represented as a graph:



# Linking Statements

- Note that the object of a triple can also be a “literal” (a string)



# RDF Syntax

---

- **Subject** of an RDF statement is a resource
- **Predicate** of an RDF statement is a property of a resource
- **Object** of an RDF statement is the value of a property of a resource

# RDF Example

---

- *Ora Lassila is the creator of the resource*

<http://www.w3.org/Home/Lassila>

```
<rdf:RDF>
  <rdf:Description about=
    "http://www.w3.org/Home/Lassila">
    <s:Creator>Ora Lassila</s:Creator>
  </rdf:Description>
</rdf:RDF>
```

# RDFS Examples

---

- **RDF Schema terms (just a few examples):**
  - Class
  - Property
  - type
  - subClassOf
  - range
  - domain
- **These terms are the RDF Schema building blocks**

**(constructors) used to create vocabularies:**

```
<Person, type, Class>
<hasColleague, type, Property>
<Professor, subClassOf, Person>
<Carole, type, Professor>
<hasColleague, range, Person>
<hasColleague, domain, Person>
```

# From RDF to OWL

---

- OWL is a language for defining Web Ontologies and their associated Knowledge Bases

- Description Logics (DLs)



- A key feature of OWL is its basis in Description Logics
- DL is family of logic-based knowledge representation formalisms that have a formal semantics based on first-order logic (FOL).

- Description Logics (Ontology)

- **TBox T: defining terminology of application domain**
  - Inclusion assertion on concept : $C \sqsubseteq D$ 
    - Pericardium  $\sqsubseteq$  Tissue  $\sqcap$  part-of.Heart
  - Inclusion assertion on roles:  $R \sqsubseteq S$ 
    - Part-of  $\sqsubseteq$  has-location
- **ABox A: stating facts about a specific “world”**
  - membership assertion:  $C(a)$  or  $R(a,b)$ 
    - HappyMan(Bob), HasChild(Bob, Mary)

# OWL Example



- There are two types of animals, **Male** and **Female**.

```
<rdfs:Class rdf:ID="Male">
  <rdfs:subClassOf rdf:resource="#Animal"/>
</rdfs:Class>
```

- The **subClassOf** element asserts that its subject - **Male** - is a subclass of its object -- the resource identified by **#Animal**.

```
<rdfs:Class rdf:ID="Female">
  <rdfs:subClassOf rdf:resource="#Animal"/>
  <owl:disjointWith rdf:resource="#Male"/>
</rdfs:Class>
```

- Some animals are **Female**, too, but nothing can be both **Male** and **Female** (in this ontology) because these two classes are disjoint (using the **disjointWith** tag).

# OWL Example in Protégé



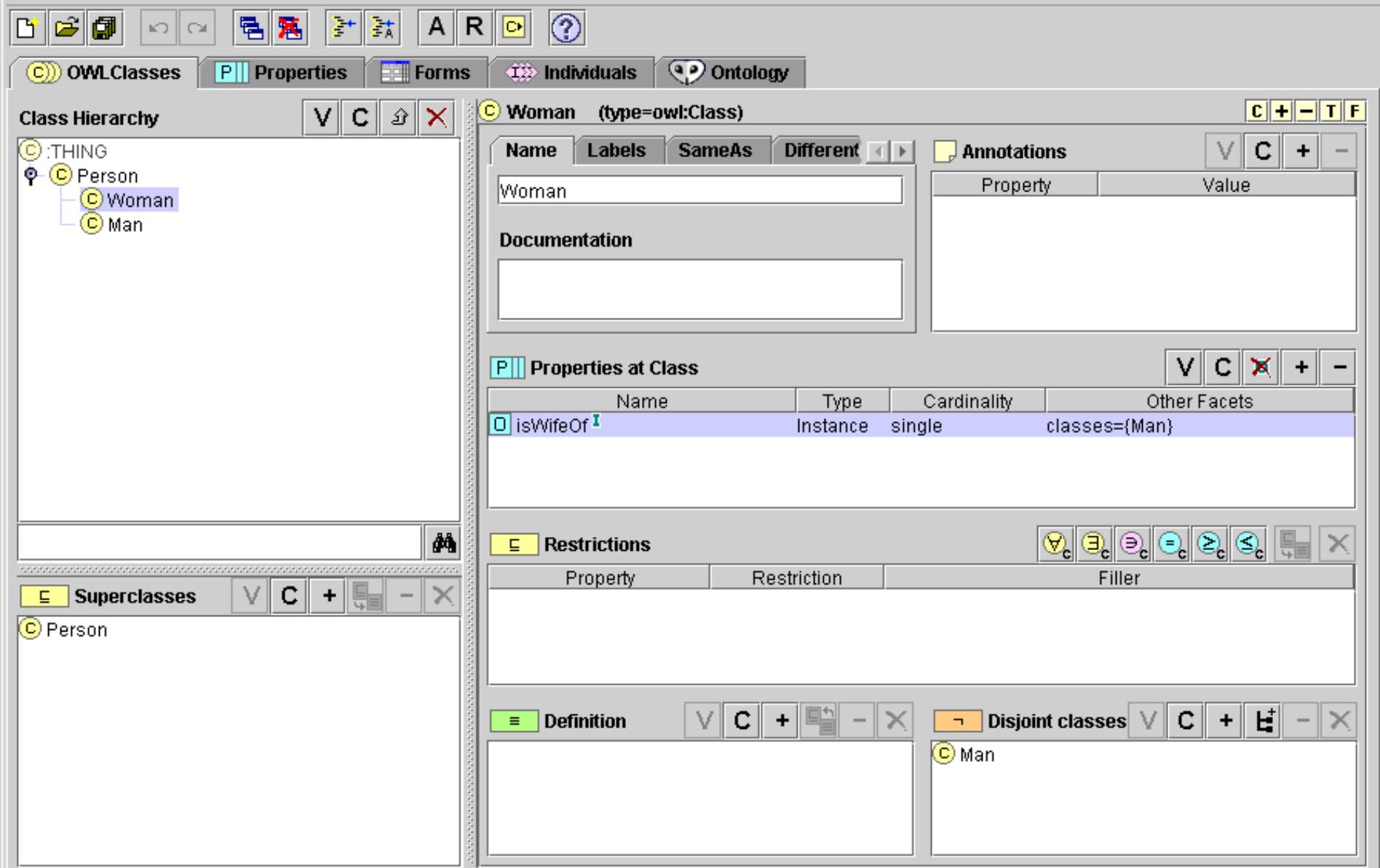
- Class
  - Person superclass
  - Man, Woman subclasses
- Properties
  - isWifeOf, isHusbandOf
- Property characteristics, restrictions
  - inverseOf
  - domain
  - range
  - Cardinality
- Class expressions
  - disjointWith

# OWL Example in Protégé



MyOntology Protégé 2.0 beta (file:/C:/ellisr/ontology/MyOntology.pprj, OWL files)

Project Edit Window owl Help



# OWL Example in Protégé



MyOntology Protégé 2.0 beta (file:/C:/ellisr/ontology/MyOntology.pprj, OWL files)

Project Edit Window OWL Help

OWLClasses Properties Forms Individuals Ontology

Properties

V C X

isHusbandOf (type=owl:ObjectProperty)

Name Labels SameAs DifferentFrom

Annotations

Property Value

Documentation

Cardinality

HasValue V C + -

Equivalent V C + -

required at least

multiple at most 1

Range Some Values From

Instance

Domain defined

Domain V C + -

Woman

Man

Inverse Property V C + -

Symmetric

AnnotationProperty

isWifeOf

Transitive

InverseFunctional

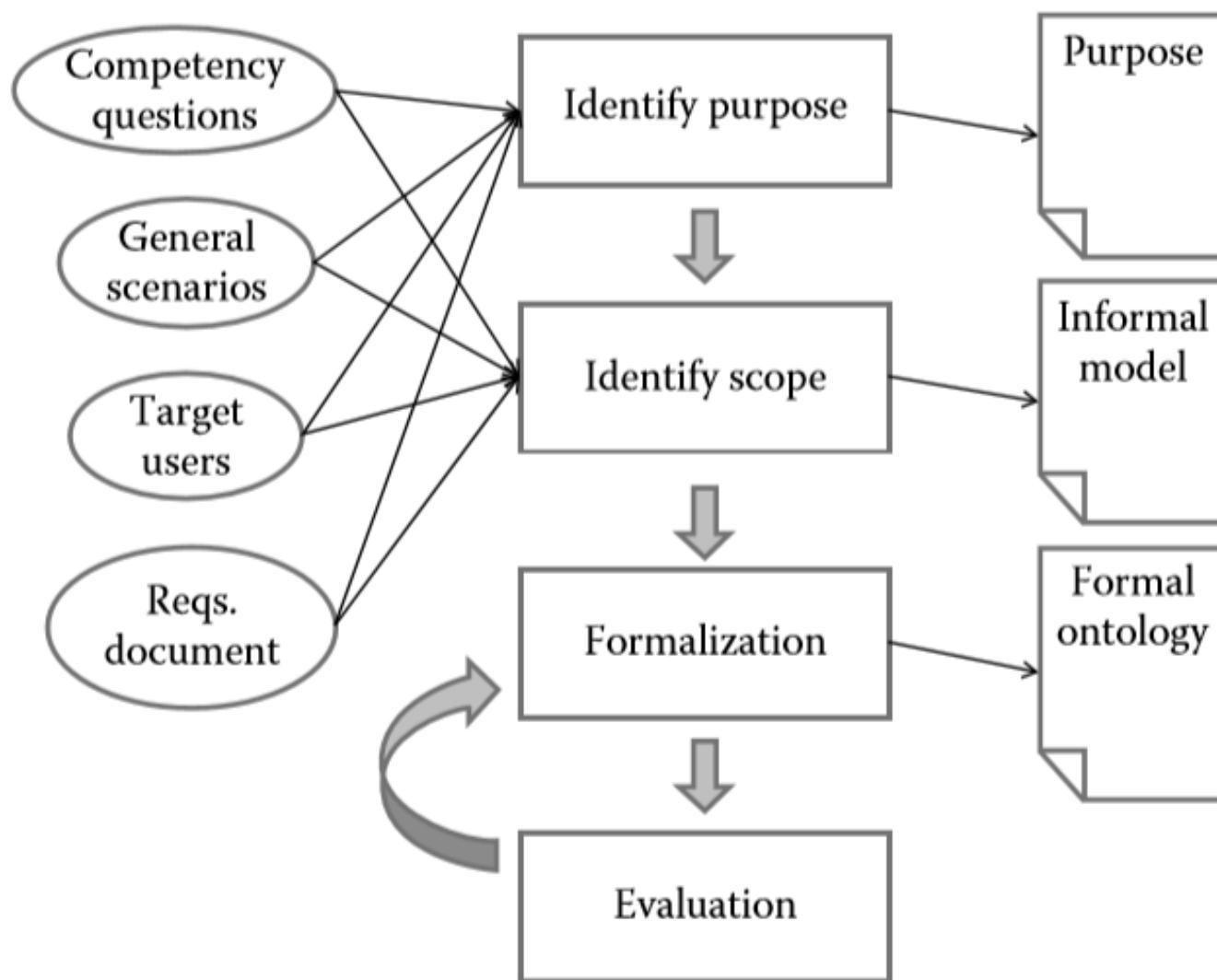
956

# • Ontology Languages and Semantic Technologies

---

- **RDF is a flexible data model for Semantic Web**
- **RDF Schema provides simple inference capability**
- **OWL allows more expressive representation of knowledge but is hard to scale to Web data**
- **Semantic technologies have been adopted by major companies such as Google, Yahoo and Facebook**

# Ontology Engineering



# • Ontology Engineering



- **Collaborative design of ontologies**
- **Ontology reuse**
  - Fusion or Merging: Process of unifying various ontologies
  - Composition or Integration: Process of selecting parts of certain ontologies
- **Engineering Paradigms**
  - Mathematical, Cognitive, Linguistic, Computer Science, Domain specialist

- State of art

- **Ontology Learning**
- **Linked open data**
- **Knowledge graph**
- **Semantic Web and web services**

# • Ontology Learning



## Automated generation of ontologies from natural language text including

- **Term extraction:**
  - Mostly run a part-of-speech tagger over the domain corpus used for the ontology learning task and identifying terms over manually constructed ad hoc patterns
- **Taxonomy induction:**
  - Concept learning should provide an intentional definition of the concept, an extensional set of concept instances, and a set of linguistic realizations for the concept.
  - Population of existing concepts with instances is typically referred to as **Ontology Population**.
- **Relation learning:**
  - Concept Hierarchy – set of taxonomic relationships among classes
  - Relation extraction is related to the problem of acquiring selection restrictions for verb arguments

# Ontology Learning

$\forall x (\text{country}(x) \rightarrow \exists y \text{ capital\_of}(y,x) \wedge \forall z (\text{capital\_of}(z,x) \rightarrow y = z))$

General axioms

disjoint (river, mountain)

Axiom schemata

capital\_of  $\leq_R$  located\_in

Relation hierarchy

flow\_through (dom:river, range:GE)

Relations

capital  $\leq_c$  city, city  $\leq_c$  inhabited\_GE

Concept hierarchy

c := country := < i(c), ||c||, Ref<sub>c</sub>(c) >

Concepts

{ country, nation }

Synonyms

river, country, nation, city, capital ...

Terms

# Linked Open Data Project

---

**Goal: “expose” open datasets in RDF  
Set RDF links among the data items  
from different datasets  
Set up, if possible, query endpoints**

# Example data source: DBpedia

---

## DBpedia is a community effort to

- extract structured (“infobox”) information from Wikipedia
- provide a query endpoint to the dataset
- interlink the DBpedia dataset with other datasets on the Web



UNIVERSITÄT LEIPZIG



# Extracting structured data from Wikipedia

```
@prefix dbpedia <http://dbpedia.org/resource/>.
@prefix dbterm <http://dbpedia.org/property/>.
```

dbpedia:Amsterdam

```
dbterm:officialName "Amsterdam" ;
dbterm:longd "4" ;
dbterm:longm "53" ;
dbterm:longs "32" ;
dbterm:leaderName dbpedia:Lodewijk_Asscher ;
```

...

```
dbterm:areaTotalKm "219" ;
```

...

dbpedia:ABN\_AMRO

```
dbterm:location dbpedia:Amsterdam ;
```

...

Amsterdam	
The Keizersgracht at dusk	
Location of Amsterdam	
Coordinates:	52°22'23"N 4°53'32"E
Country	Netherlands
Province	North Holland
Government	Municipality
- Type	Job Cohen <sup>[1]</sup> (PvdA)
- Mayor	Lodewijk Asscher
- Aldermen	Carolien Gehrels
	Tjeerd Herrema
	Maarten van Poelgeest
	Marijke Vos
	Erik Gerritsen
- Secretary	
Area <sup>[2][3]</sup>	
- City	219 km <sup>2</sup> (84.6 sq mi)
- Land	166 km <sup>2</sup> (64.1 sq mi)
- Water	53 km <sup>2</sup> (20.5 sq mi)
- Urban	1,003 km <sup>2</sup> (387.3 sq mi)
- Metro	1,815 km <sup>2</sup> (700.8 sq mi)
Elevation <sup>[4]</sup>	2 m (7 ft)
Population <sup>(1 October 2008)[5][6]</sup>	
- City	755,269
- Density	4,459/km <sup>2</sup> (11,548.8/sq mi)
- Urban	1,364,422
- Metro	2,158,372
- Demonym	Amsterdamer
Time zone	CET (UTC+1)
- Summer (DST)	CEST (UTC+2)
Postcodes	1011 – 1109
Area code(s)	020
Website:	<a href="http://www.amsterdam.nl">www.amsterdam.nl</a>

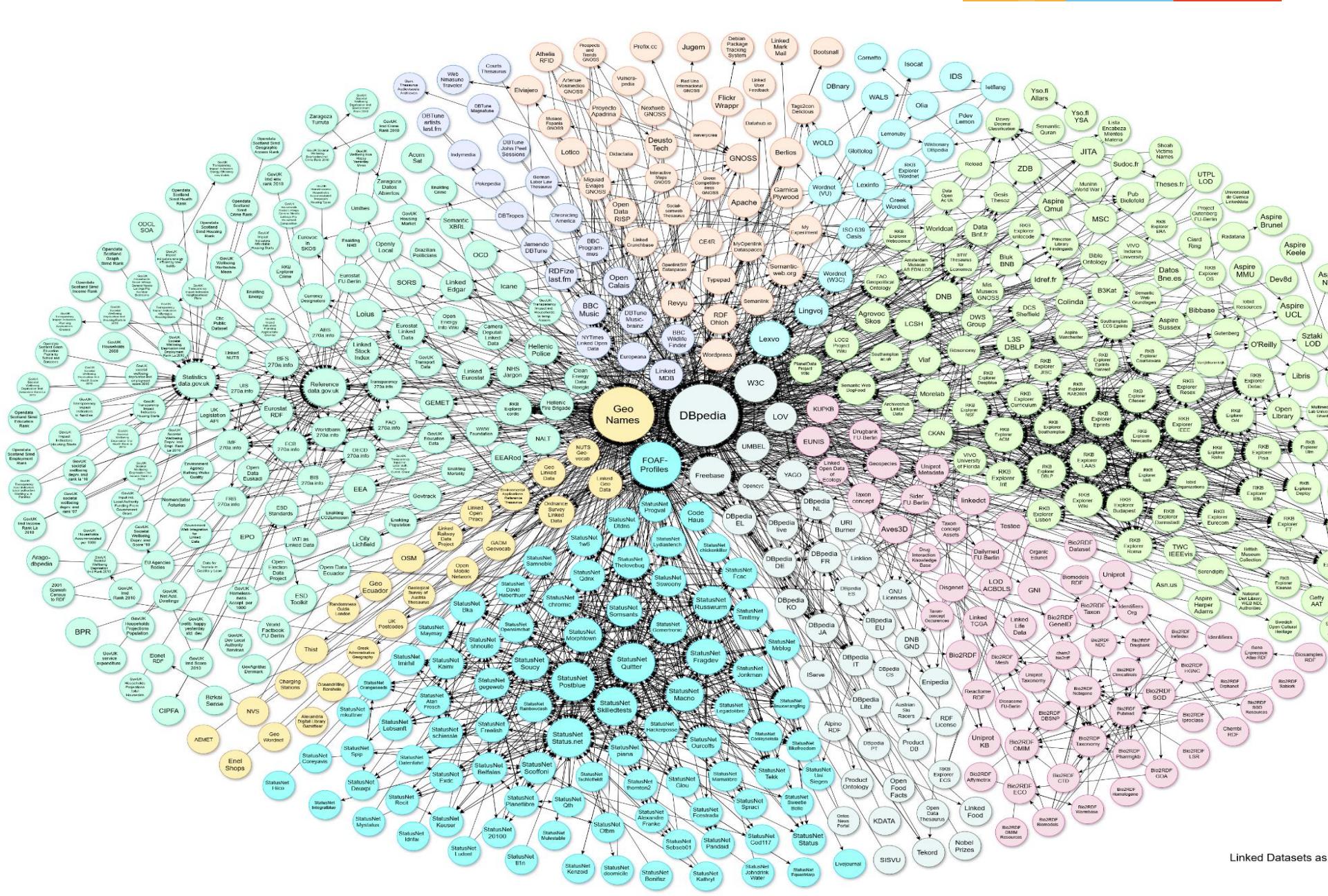
# Automatic links among open datasets

```
<http://dbpedia.org/resource/Amsterdam>
owl:sameAs <http://rdf.freebase.com/ns/...> ;
owl:sameAs <http://sws.geonames.org/2759793> ;
...
```

```
<http://sws.geonames.org/2759793>
owl:sameAs <http://dbpedia.org/resource/Amsterdam>
wgs84_pos:lat "52.3666667" ;
wgs84_pos:long "4.8833333";
geo:inCountry <http://www.geonames.org/countries/#NL> ;
...
```

Processors can switch automatically from one to the other...

# The LOD “cloud”



# References



- Berners-lee. A roadmap to the semantic web. Tim berners-lee's design issues @ <http://bit.Ly/2z29fgp>
- Berners-lee. Linked data. Tim berners-lee's design issues @ <http://bit.Ly/21mr3zt>
- Heath, bizer. Linked data: evolving the web into a global data space. Synthesis lectures on the semantic web: theory and technology @ <http://bit.Ly/2xlzkou>
- <Https://wiki.Dbpedia.Org/services-resources/ontology>
- <Https://www.Emse.Fr/~zimmermann/teaching/semweb/semwebintro.Pdf>
- <Https://www.Youtube.Com/watch?V=c9m7n979piu>
- <Https://enterprise-knowledge.Com/whats-the-difference-between-an-ontology-and-a-knowledge-graph/>
- <Https://www.W3.Org/consortium/facts>
- <Https://pages.Semanticscholar.Org/coronavirus-research>



**BITS** Pilani  
Pilani Campus



# Natural Language Processing

**Dr. Vijayalakshmi anand**  
**BITS Pilani**

## Disclaimer and Acknowledgement

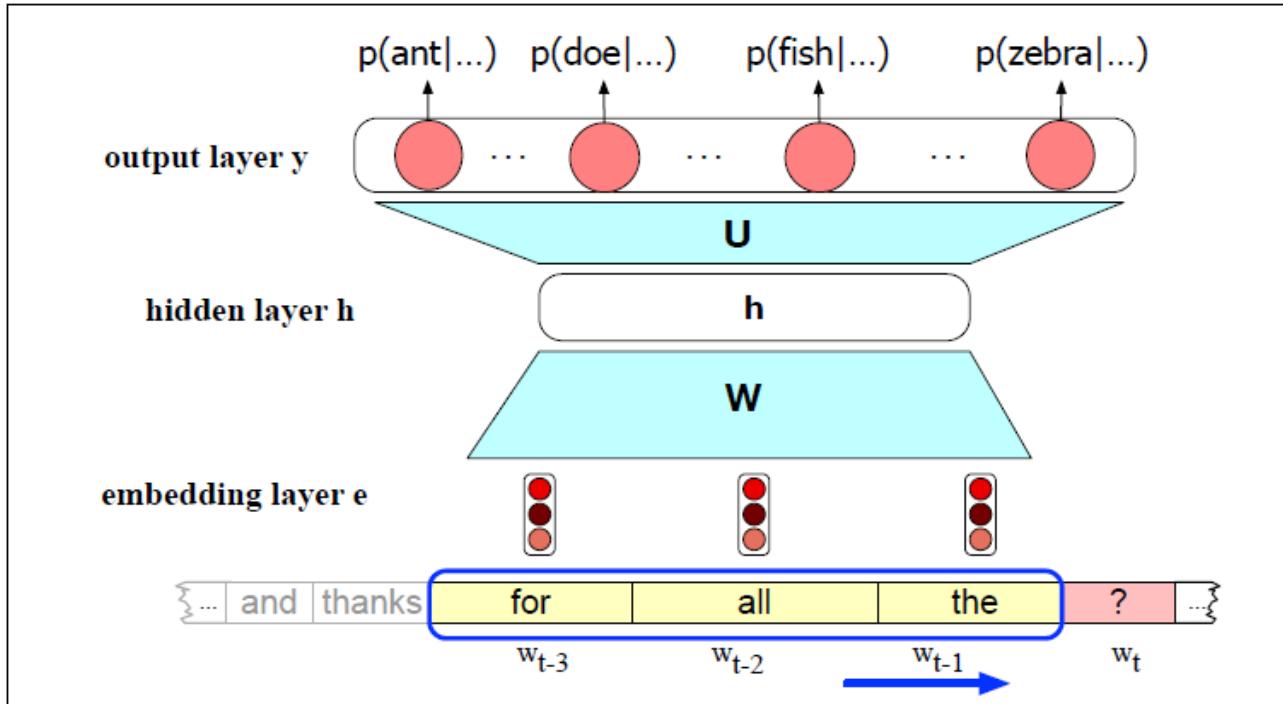


- The content for these slides has been obtained from books and various other source on the Internet
- I here by acknowledge all the contributors for their material and inputs.
- I have provided source information wherever necessary
- I have added and modified the content to suit the requirements of the course

# Session Content

- Issues with recurrent models
- Encoder-Decoder architecture
- Attention
- Transformer architecture
  - Self attention and Multihead-attention
  - Position encoding
  - Masked Self attention
  - Cross attention
  - Transformer encoder-decoder
- Word structure and subword models
- Pre training
- Types of language modelling
- BERT

# Simple feedforward sliding-window as a language model



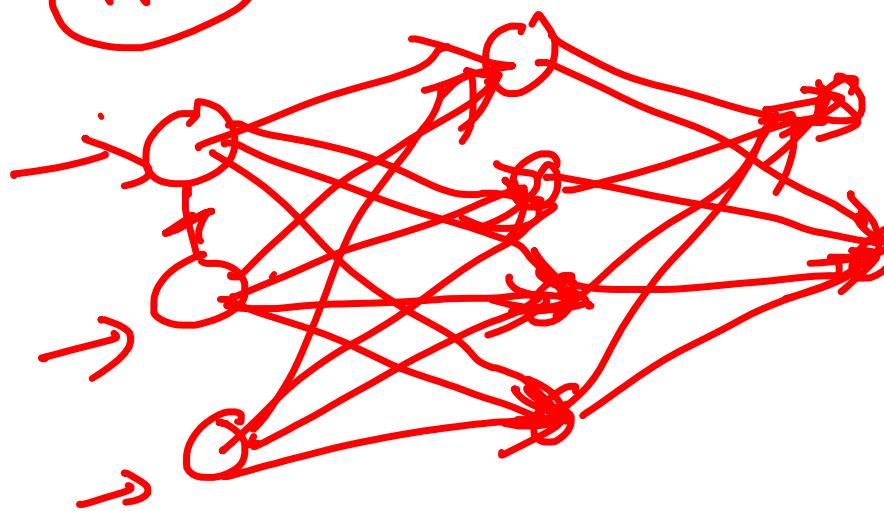
**Figure 9.1** Simplified sketch of a feedforward neural language model moving through a text. At each time step  $t$  the network converts  $N$  context words, each to a  $d$ -dimensional embedding, and concatenates the  $N$  embeddings together to get the  $Nd \times 1$  unit input vector  $x$  for the network. The output of the network is a probability distribution over the vocabulary representing the model's belief with respect to each word being the next possible word.

- **Language Model:** A system that predicts the next word
- Embeddings as inputs
- Solve the main problem of the simple n-gram models
  - n-grams are based on words rather than embeddings
- Limitation:
  - Limited context.
  - Anything outside the context window has no impact on the decision being made
  - E.g phrase 'all the' appears in one window in the 2 and 3 positions, and in the next window in 1 and 2 positions, forcing the network to learn two separate patterns for the same item.

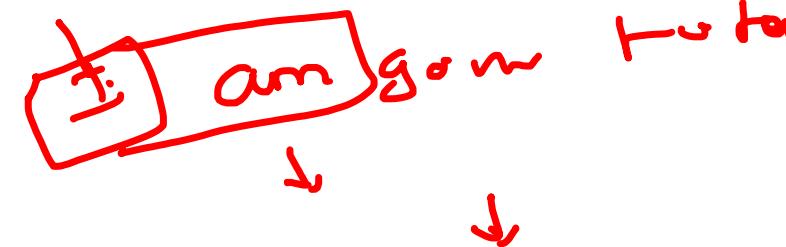
Feed  $\rightarrow$  ANN

IIP

Hidden  $\rightarrow$  OLP



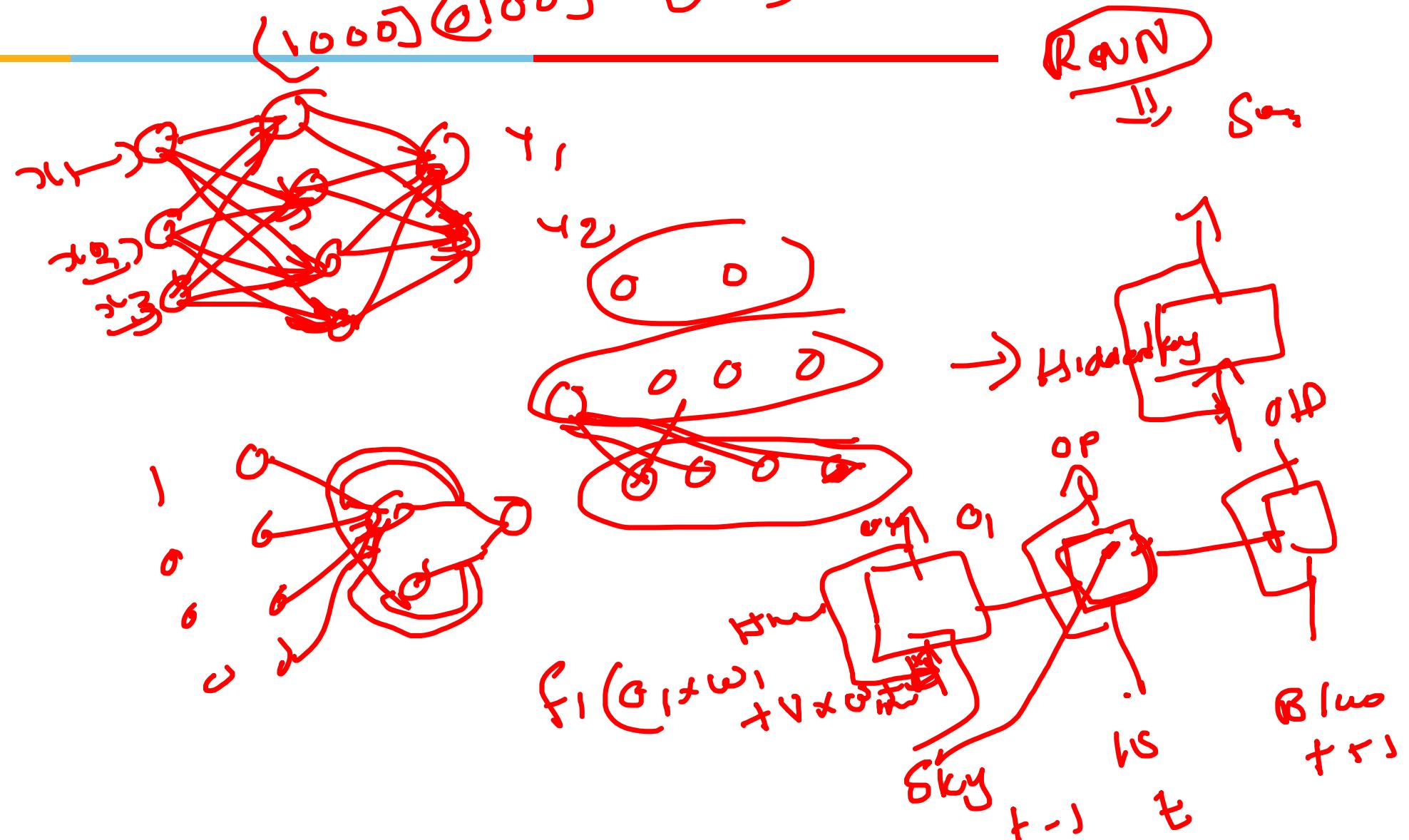
sliding window



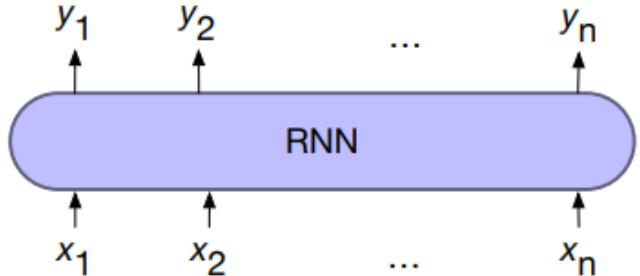
# Deep Learning Architectures for Sequence Processing

- **Recurrent neural networks and transformer networks**
- Both capture and exploit the **temporal** nature of language
- RNN uses the **prior context**, allowing the model's decision to depend on information from hundreds of words in the past.
- Recurrent Neural Network  $\neq$  Language Model
  - RNNs can be used for many other things
- Now everything in NLP is being rebuilt upon Language Modeling
- **The transformer uses mechanisms (self-attention and positional encodings) that help focus on how words relate to each other over long distances**

sky is blue  
 $(1000) (0100) \{0010\}$

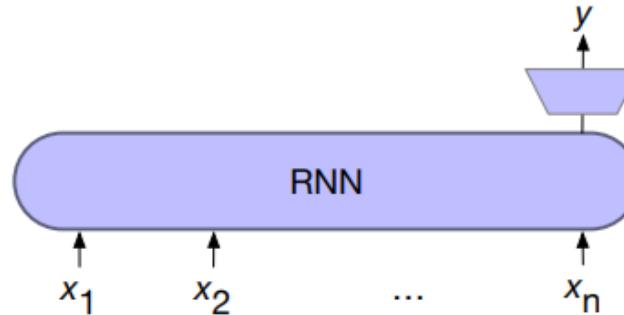


# RNN Architectures for NLP Tasks



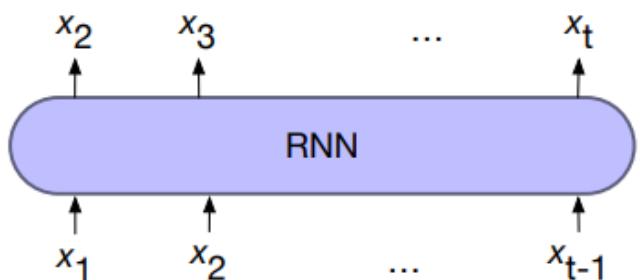
a) sequence labeling

Ex: POS Tagging, Named Entity Tagging



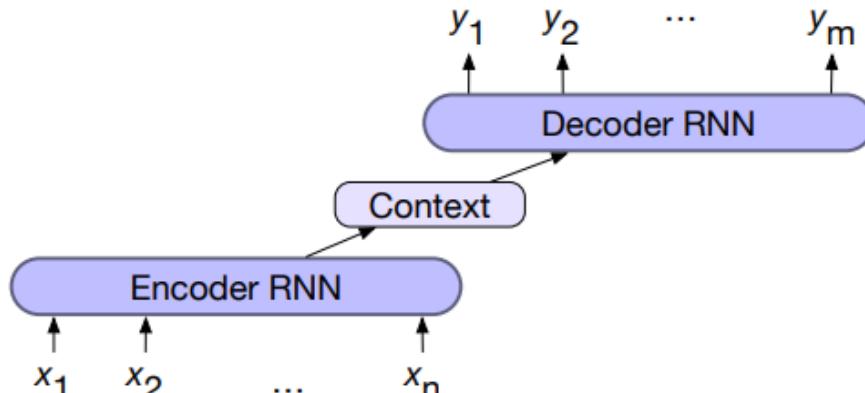
b) sequence classification

Ex: Sentiment Analysis



c) language modeling

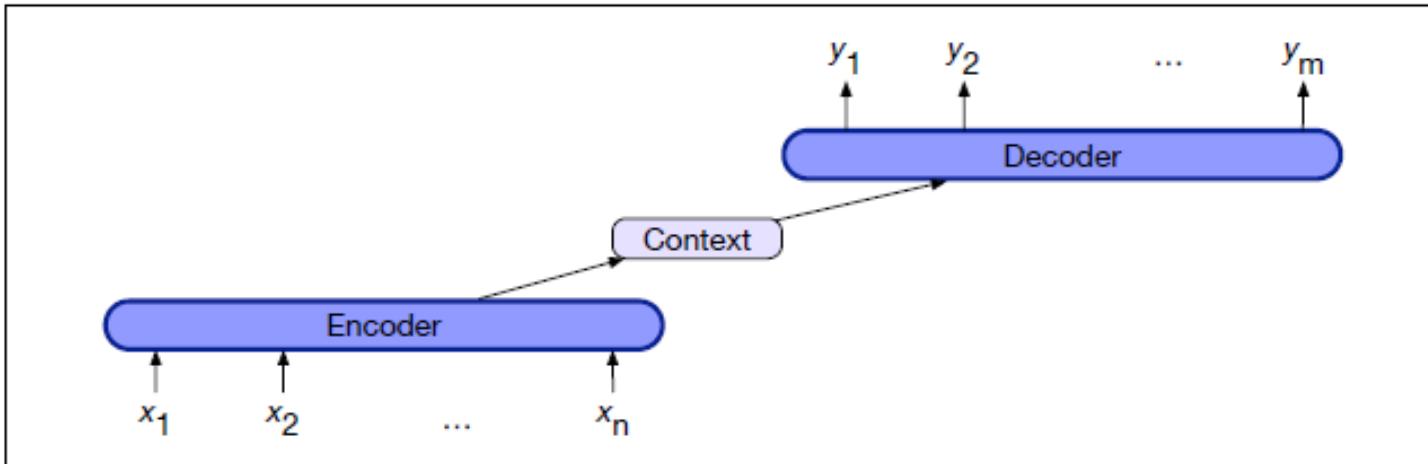
Ex: Predict Next Word



d) encoder-decoder

Ex: Language Translation

# Encoder-Decoder architecture

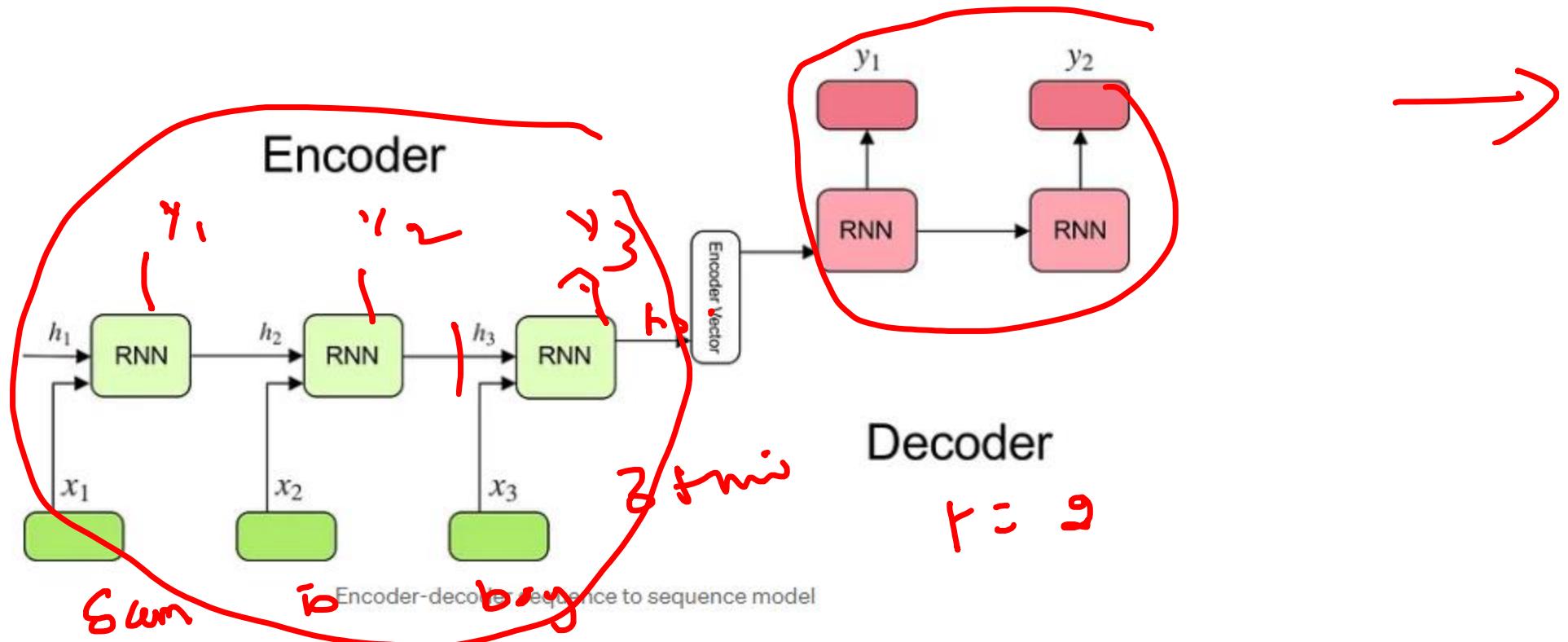


**Figure 10.3** The encoder-decoder architecture. The context is a function of the hidden representations of the input, and may be used by the decoder in a variety of ways.

- An encoder that accepts an input sequence,  $x_1 \dots x_n$  and generates a corresponding sequence of contextualized representations,  $h_1 \dots h_n$
- A context vector,  $c$ , which is a function of  $h_1 \dots h_n$ , and conveys the essence of the input to the decoder.
- A decoder, which accepts  $c$  as input and generates an arbitrary length sequence of hidden states  $h_1 \dots h_m$ , from which a corresponding sequence of output states  $y_1 \dots y_m$ , can be obtained.
- **LSTMs, convolutional networks, and Transformers can all be employed as encoders/decoders**

# Encoder - Decoder using RNN

## Encoder - Decoder for Language Translation

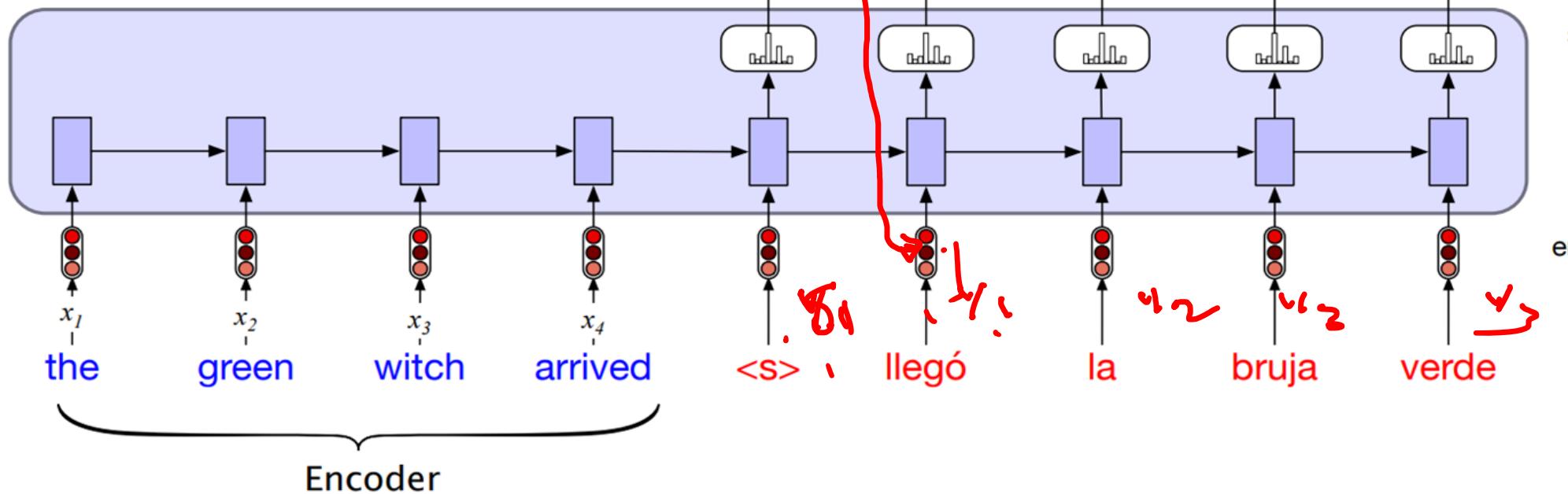


# Encoder - Decoder

Training

Total loss is the average cross-entropy loss per target word:

$$L = \frac{1}{T} \sum_{i=1}^T L_i$$



# Encoder - Decoder

## Teaching Forcing



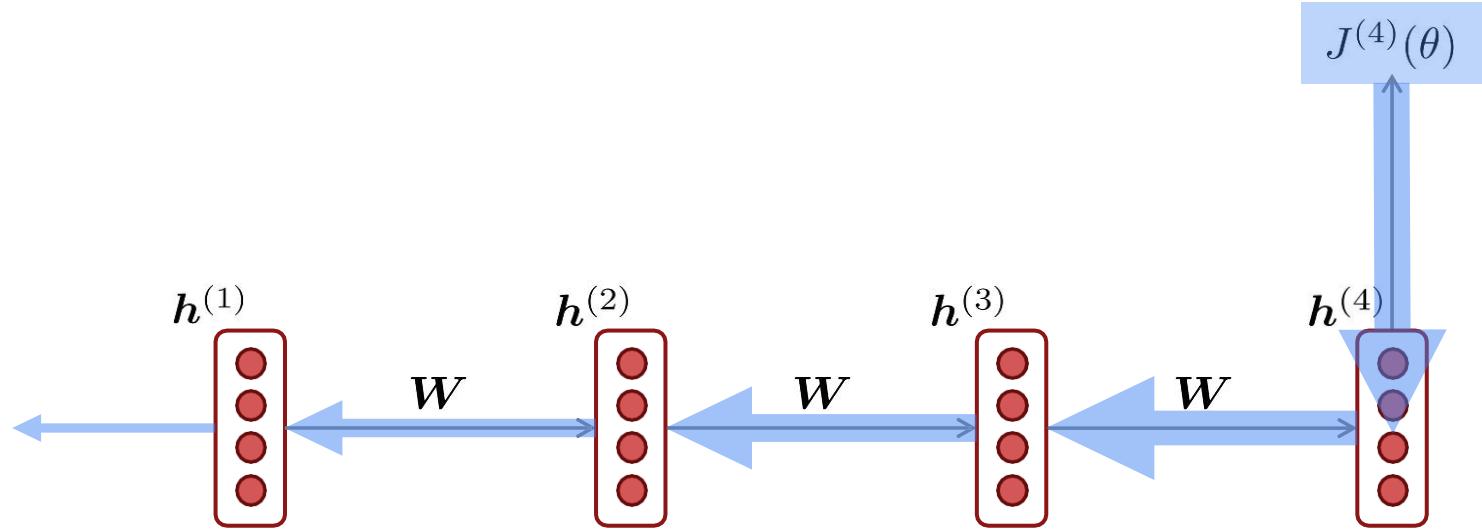
- Force the system to use the gold target token from training as the next input  $x_{t+1}$ , rather than allowing it to rely on the (possibly erroneous) decoder output  $\hat{y}_t$ .
- Speeds up training

# Sequence-to-sequence is versatile!

---

- The general notion here is an **encoder-decoder** model
  - One neural network takes input and produces a neural representation
  - Another network produces output based on that neural representation
  - **If the input and output are sequences, we call it a seq2seq model**
- Sequence-to-sequence is useful for *more than just MT*
- Many NLP tasks can be phrased as sequence-to-sequence:
  - **Summarization** (long text → short text)
  - **Dialogue** (previous utterances → next utterance)
  - **Parsing** (input text → output parse as sequence)
  - **Code generation** (natural language → Python code)

# Vanishing gradient intuition



$$\frac{\partial J^{(4)}}{\partial \mathbf{h}^{(1)}} = \boxed{\frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}}} \times \boxed{\frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{h}^{(2)}}} \times \boxed{\frac{\partial \mathbf{h}^{(4)}}{\partial \mathbf{h}^{(3)}}} \times \boxed{\frac{\partial J^{(4)}}{\partial \mathbf{h}^{(4)}}}$$

↓ ↓ ↓ ↓

What happens if these are small?

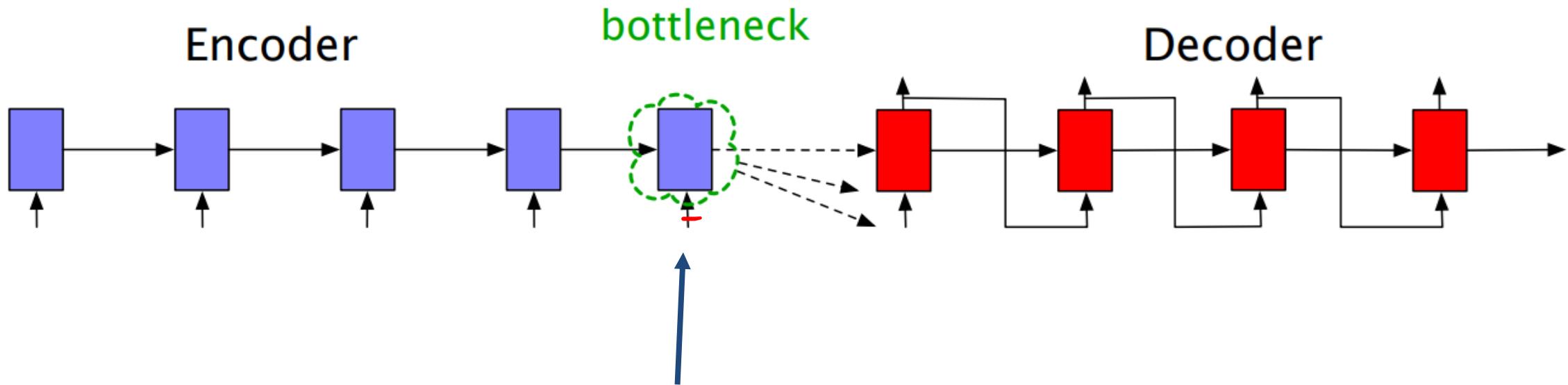
chain rule!

**Vanishing gradient problem:**  
When these are small, the gradient signal gets smaller and smaller as it backpropagates further

# Effect of vanishing gradient on RNN-LM

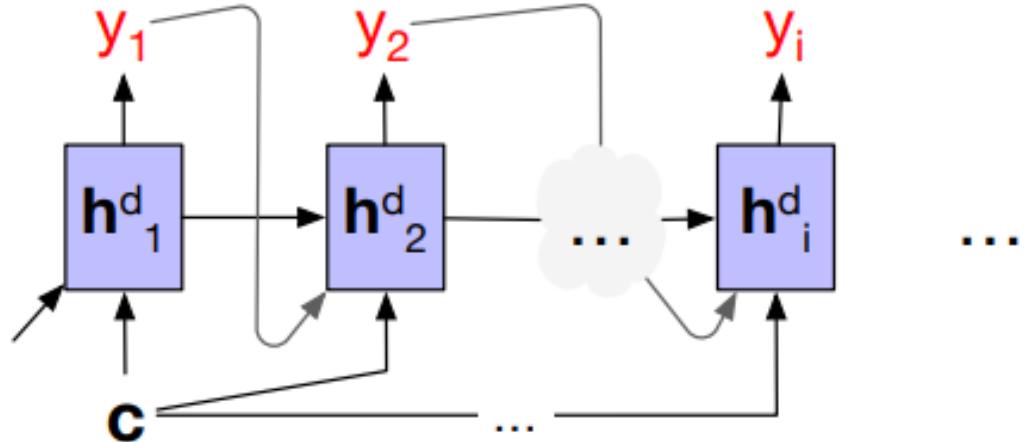
- **LM task:** *When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her \_\_\_\_\_*
- To learn from this training example, the RNN-LM needs to **model the dependency** between “*tickets*” on the 7<sup>th</sup> step and the target word “*tickets*” at the end.
- But if the gradient is small, the model **can't learn this dependency**
  - So, the model is **unable to predict similar long-distance dependencies** at test time
- In practice a simple RNN will only condition ~7 tokens back [**vague rule-of-thumb**]

# Encoder-decoder bottleneck



- In an encoder-decoder arch, the final hidden state acts as a bottleneck:
- It must represent absolutely everything about the meaning of the source text
- The only thing the decoder knows about the source text is what's in this context vector

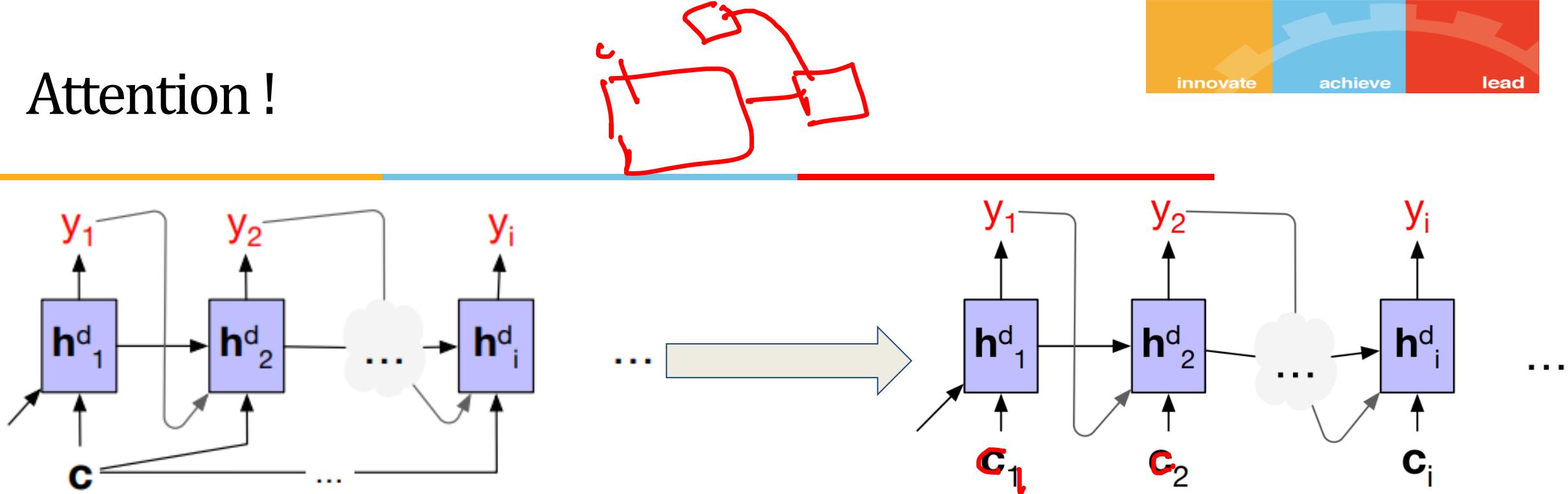
# Attention !



Without attention, a decoder sees the same context vector , which is a static function of all the encoder hidden states

$$\mathbf{h}_t^d = g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c})$$

# Attention !



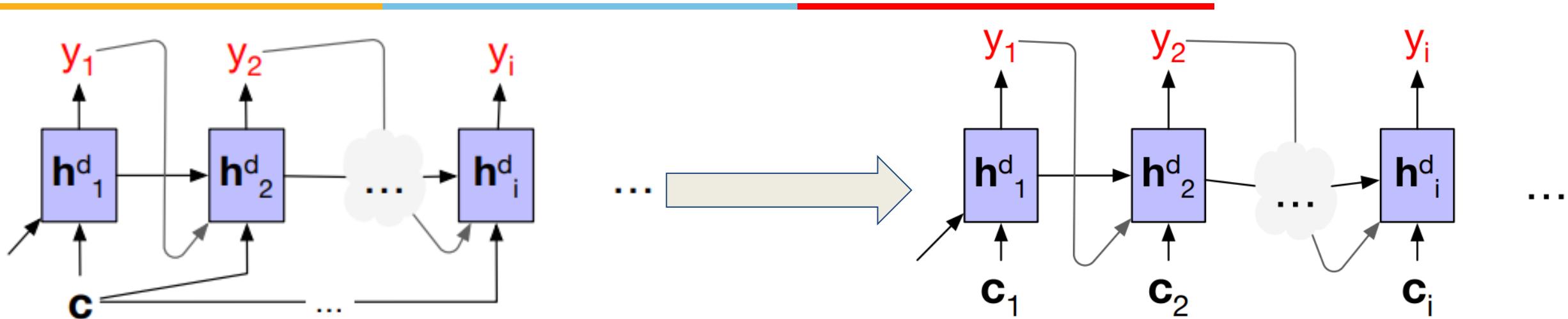
Without attention, a decoder sees the same context vector ,  
which is a static function of all the encoder hidden states

$$\mathbf{h}_t^d = g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c})$$

With attention, decoder to sees a different, dynamic, context,  
which is a function of all the encoder hidden states

$$\mathbf{h}_i^d = g(\hat{y}_{i-1}, \mathbf{h}_{i-1}^d, \mathbf{c}_i)$$

# Attention !



Without attention, a decoder sees the same context vector ,  
which is a static function of all the encoder hidden states

$$\mathbf{h}_t^d = g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c})$$

With attention, decoder gets information from all the hidden states of the encoder, not just the last hidden state of the encoder

Each context vector is obtained by taking a weighted sum of all the encoder hidden states.

The weights focus on ('attend to') a particular part of the source text that is relevant for the token the decoder is currently producing

# Attention !

**Step -1 :** Find out how relevant each encoder state is to the present decoder state  $\mathbf{h}_{i-1}^d$

Compute a score of similarity between  $\mathbf{h}_{i-1}^d$  and all the encoder states :  $score(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e)$

**Dot Product Attention :**  $score(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e) = \mathbf{h}_{i-1}^d \cdot \mathbf{h}_j^e$

**Step -2 :** Normalize all the scores with softmax to create a vector of weights,  $\alpha_{i,j}$

$\alpha_{i,j}$  indicates the proportional relevance of each encoder hidden state  $j$  to the prior hidden decoder state,  $\mathbf{h}_{i-1}^d$

$$\begin{aligned}\alpha_{ij} &= \text{softmax}(\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e) \quad \forall j \in e) \\ &= \frac{\exp(\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e))}{\sum_k \exp(\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_k^e))}\end{aligned}$$

# Attention !

**Step -3 :** Given the distribution in  $\alpha$ , compute a fixed-length context vector for the current decoder state by taking a weighted average over all the encoder hidden states

$$\mathbf{c}_i = \sum_j \alpha_{ij} \mathbf{h}_j^e$$

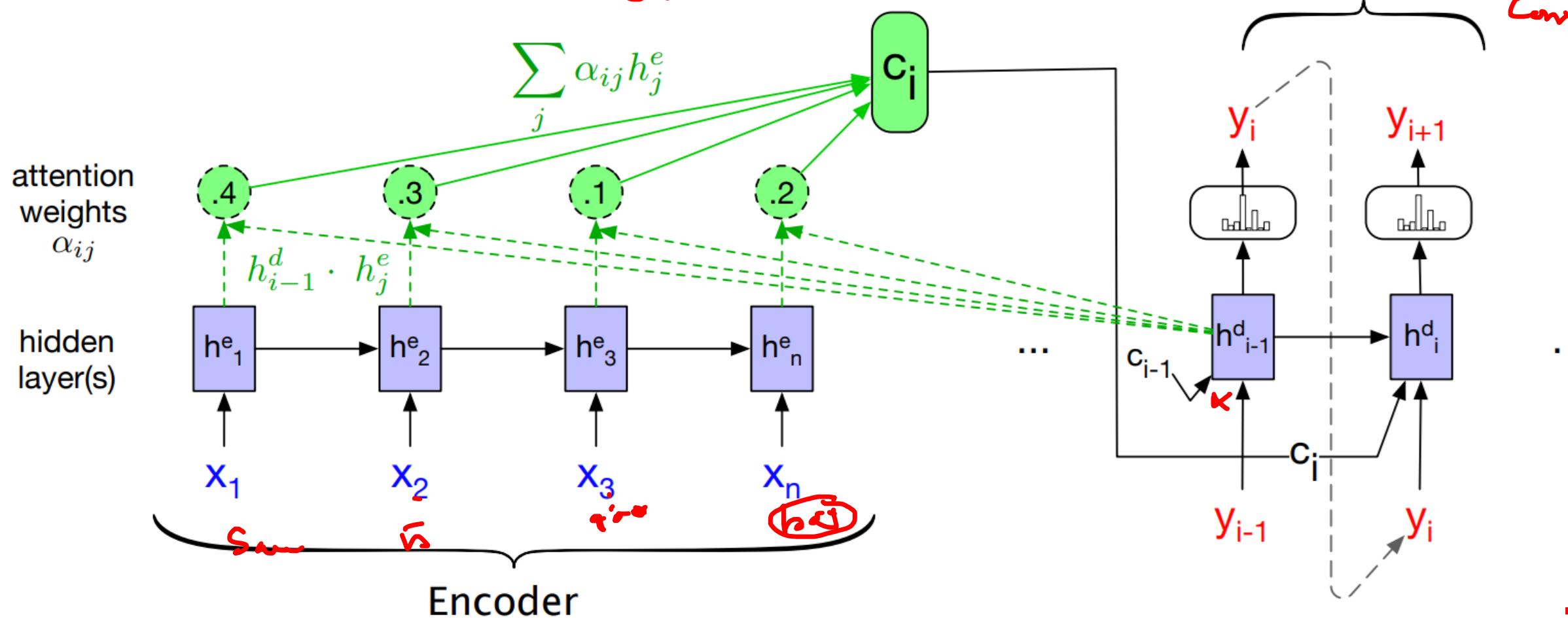
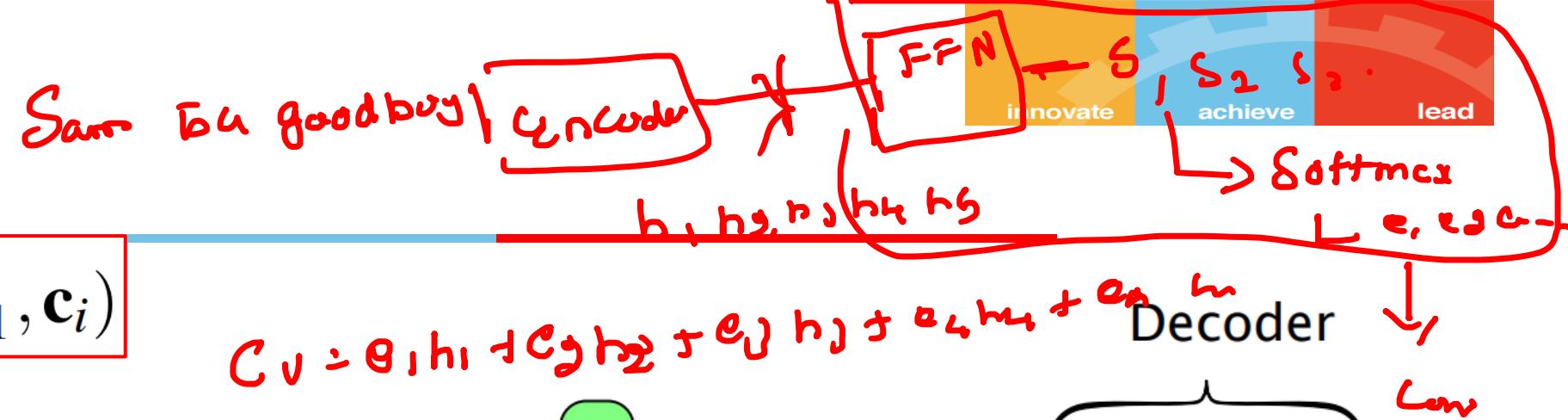
**Plus :** In step-1, we can get a more powerful scoring function by parameterizing the score with its own set of weights,  $\mathbf{W}_s$ :

$$\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e) = \mathbf{h}_{i-1}^d \mathbf{W}_s \mathbf{h}_j^e$$

$\mathbf{W}_s$  , is trained during normal end-to-end training,

$\mathbf{W}_s$  , gives the network the ability to learn which aspects of similarity between the decoder and encoder states are important to the current application.

# Attention !



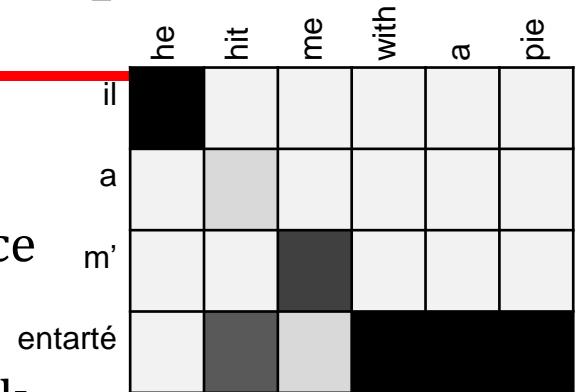
# Attention is a *general* Deep Learning technique

- Attention significantly **improves NMT performance**
  - It's very useful to allow decoder to focus on certain parts of the source
- Attention **solves the bottleneck problem**
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention **helps with the vanishing gradient problem**
  - Provides shortcut to faraway states
- By inspecting attention distribution, we see what the decoder was focusing on

## More general definition of attention:

**Given a set of vector *values*, and a vector *query*, attention is a technique to compute a weighted sum of the values, dependent on the query.**

- We sometimes say that the *query attends to the values*.
- For example, in the seq2seq + attention model, each decoder hidden state (query) *attends to* all the encoder hidden states (values).



# Transformers

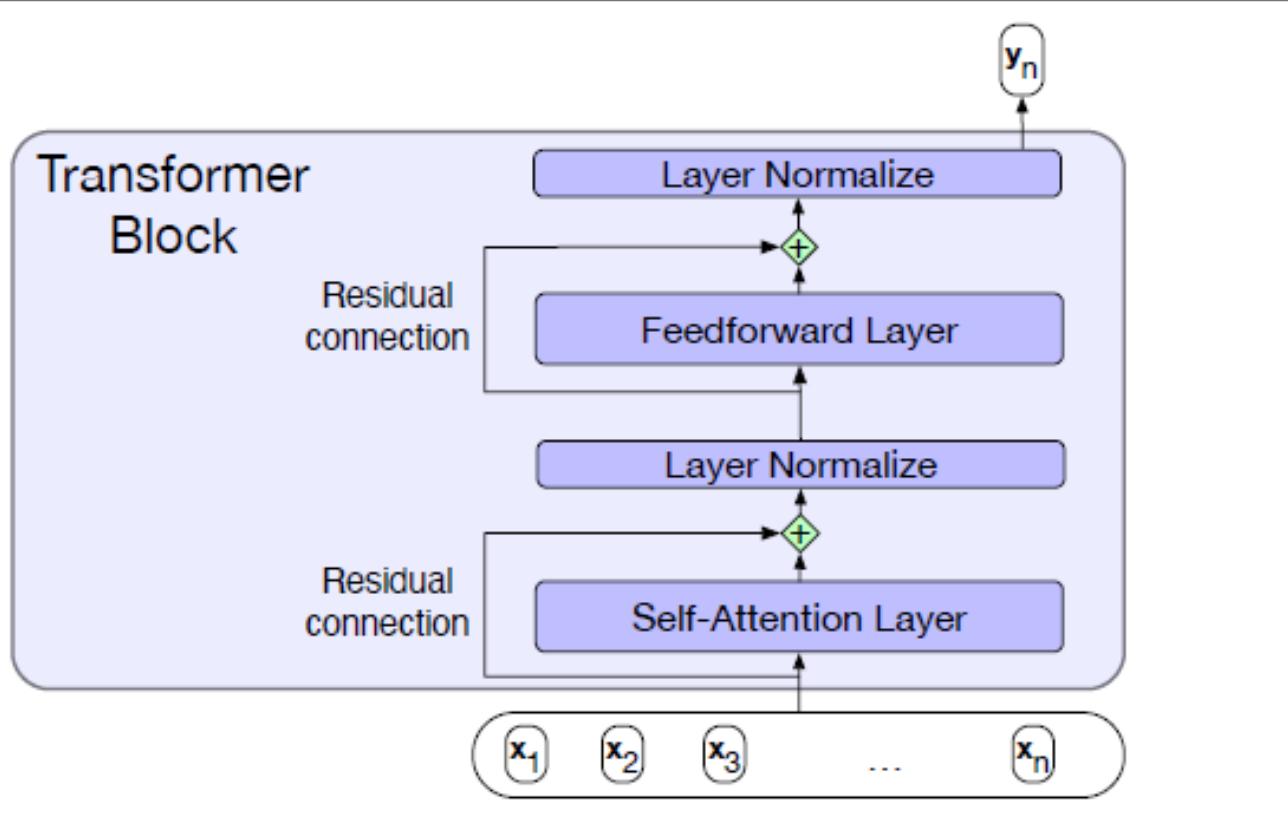
---

- 2017, NIPS, Vaswani et. al., **Attention Is All You Need !!!**
- Transformers **map sequences** of input vectors ( $x_1, \dots, x_n$ ) to sequences of output vectors ( $y_1, \dots, y_n$ ) **of the same length**
- Made up of **transformer blocks** in which the key component is **self-attention** layers

[ Self-attention allows a network to directly extract and use information from arbitrarily large contexts directly !!! ]
- Transformers are **not based on recurrent connections**  $\Rightarrow$  Parallel implementations possible  $\Rightarrow$  Efficient to scale ( comparing LSTM)

# Transformer

- Each Block consists of:
  - Self-attention
  - Add & Norm
  - Feed-Forward
  - Add & Norm



**Figure 9.18** A transformer block showing all the layers.



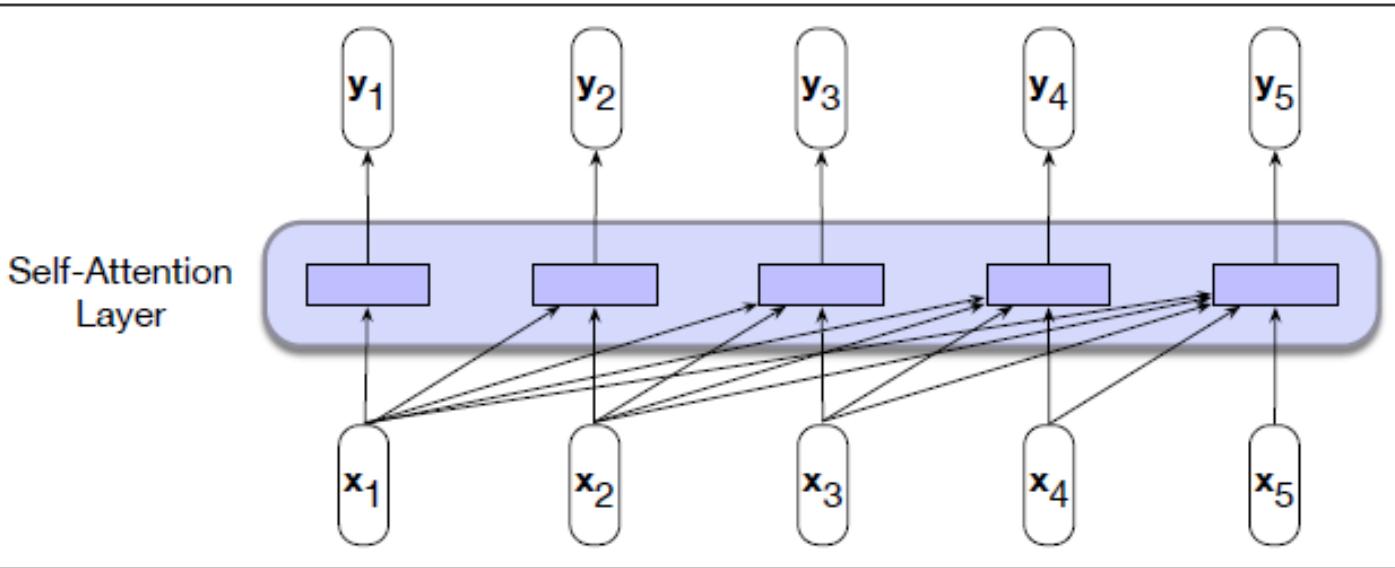
1)

# Self-Attention | Transformers



- Attention  $\Rightarrow$  Ability to compare an item of interest to a collection of other items in a way that reveals their relevance in the current context.
- Self-attention  $\Rightarrow$ 
  - > Set of **comparisons** are to other elements **within a given sequence**
  - > Use these comparisons to compute an output for the current input

# Self attention layer



**Figure 9.15** Information flow in a causal (or masked) self-attention model. In processing each element of the sequence, the model attends to all the inputs up to, and including, the current one. Unlike RNNs, the computations at each time step are independent of all the other steps and therefore can be performed in parallel.

- Computation of  $y_3$  is based on a set of comparisons between the input  $x_3$  and its preceding elements  $x_1$  and  $x_2$ , and to  $x_3$  itself
- When processing each item in the input, the model has no access to information about inputs beyond the current one.
- This ensures that we can use this approach to create language models and use them for autoregressive generation

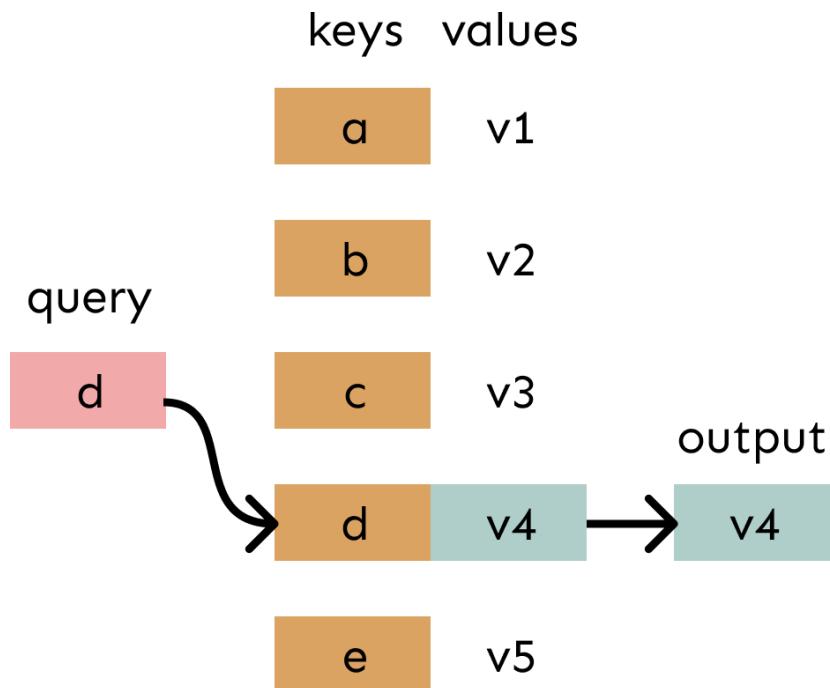
The cut sat on a key value.

It met Query key value.

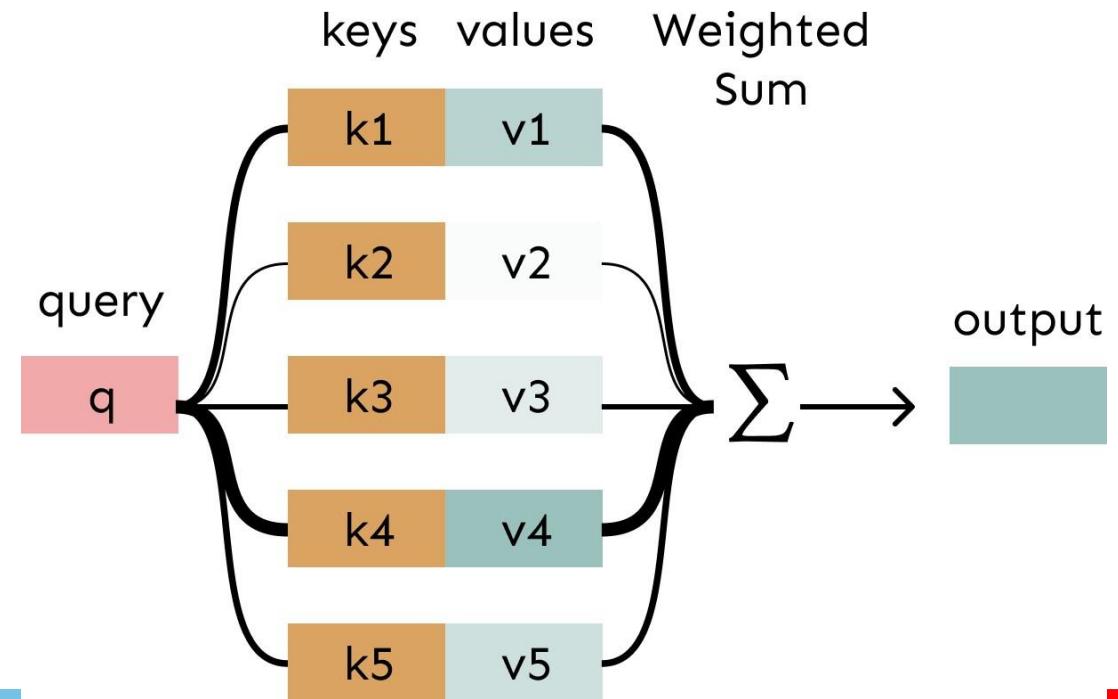
# Attention as a soft, averaging lookup table

We can think of **attention** as performing fuzzy lookup in a key-value store.

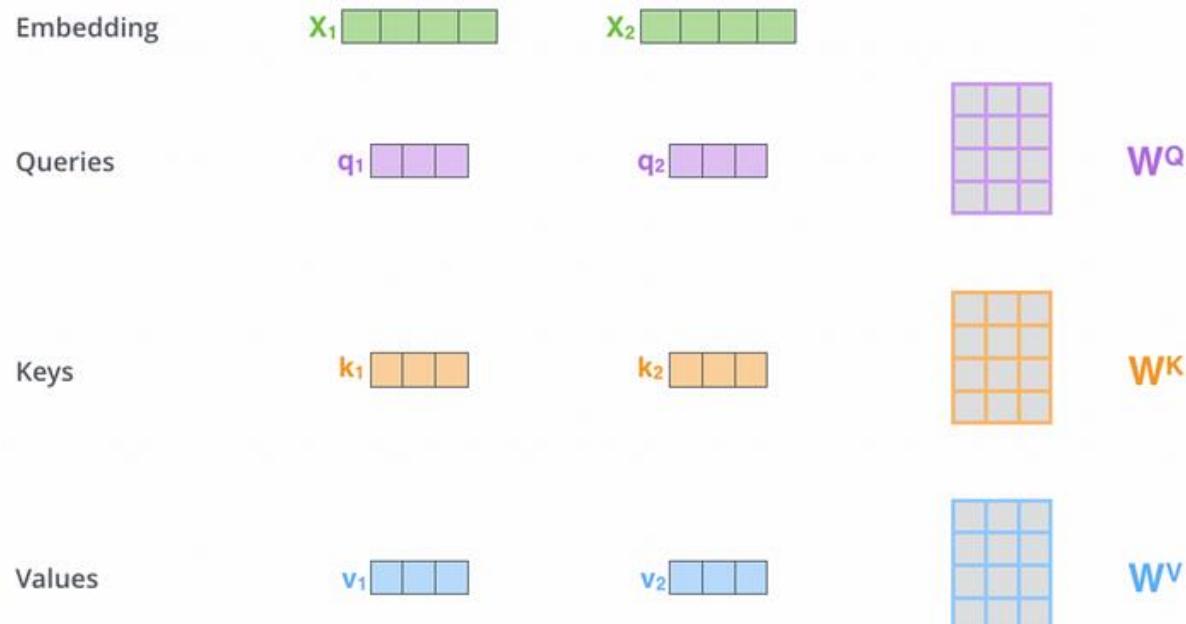
In a **lookup table**, we have a table of **keys** that map to **values**. The **query** matches one of the keys, returning its value.



In **attention**, the **query** matches all **keys** *softly*, to a weight between 0 and 1. The keys' **values** are multiplied by the weights and summed.



# Parallelized using efficient matrix multiplication



**Key:** In its role as a preceding input being compared to the current focus of attention.

**Query:** As the *current focus of attention* when being compared to all of the other preceding inputs. Check all available keys and selects the one, that matches best.

**Value:** Key and values always come in pairs. When a query matches a key, not the key itself but the value of the word gets propagated further. Value is used to compute the output for the current focus of attention.

# Self-Attention: keys, queries, values from the same sequence

1. Transform each word embedding with weight matrices  $Q, K, V$ , each in  $\mathbb{R}^{d \times d}$

$$q_i = Qx_i \text{ (queries)}$$

$$k_i = Kx_i \text{ (keys)}$$

$$v_i = Vx_i \text{ (values)}$$

2. Compute pairwise similarities between keys and queries; normalize with softmax

$$e_{ij} = q_i^T k_j \quad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

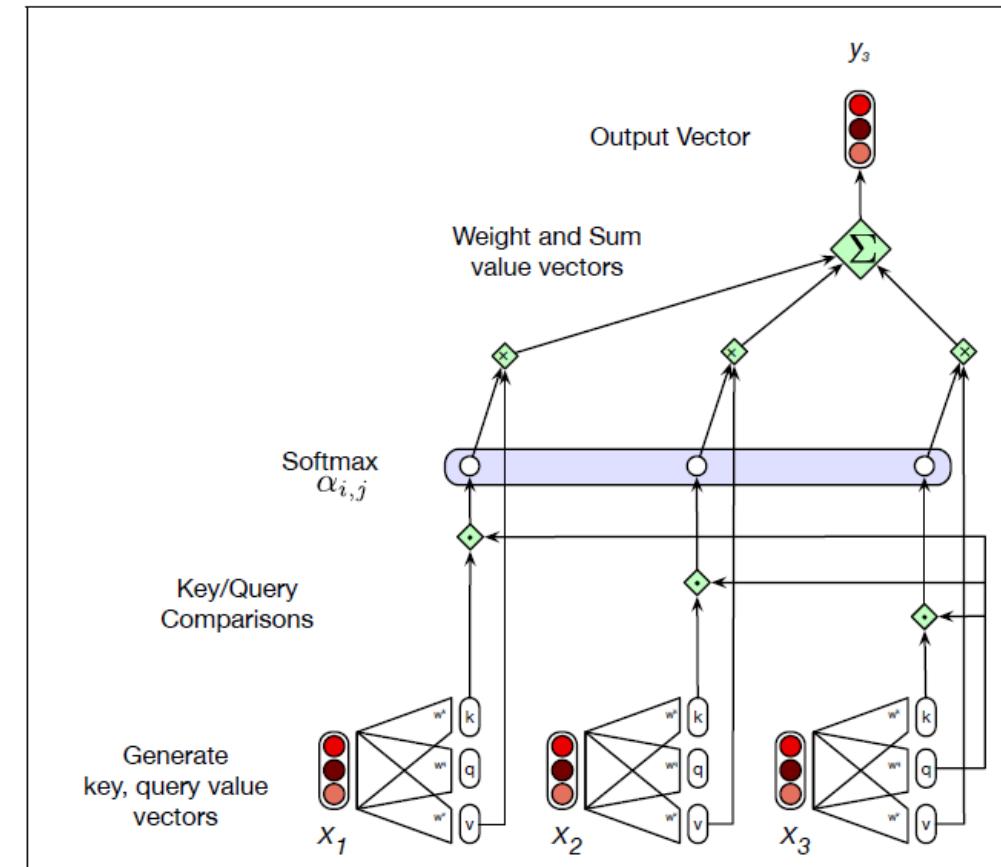
3. Compute output for each word as weighted sum of values

$$o_i = \sum_j \alpha_{ij} v_i$$

4. The similarity between words is called **alignment**

5. The query and key vectors are used to calculate **alignment scores** that are measures of how well the query and keys match.

computing a single output at a single time step i



**Figure 9.16** Calculating the value of  $y_3$ , the third element of a sequence using causal (left-to-right) self-attention.

Ans → Sliding window concept

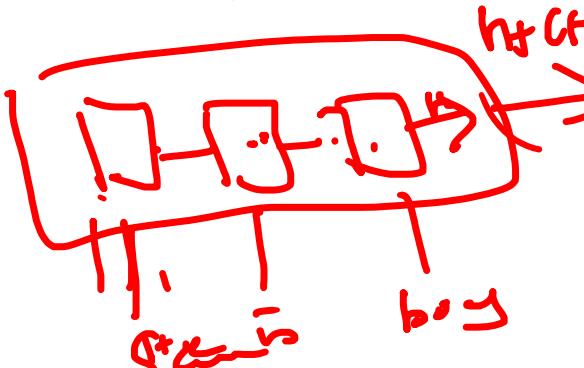
innovate

achieve

lead

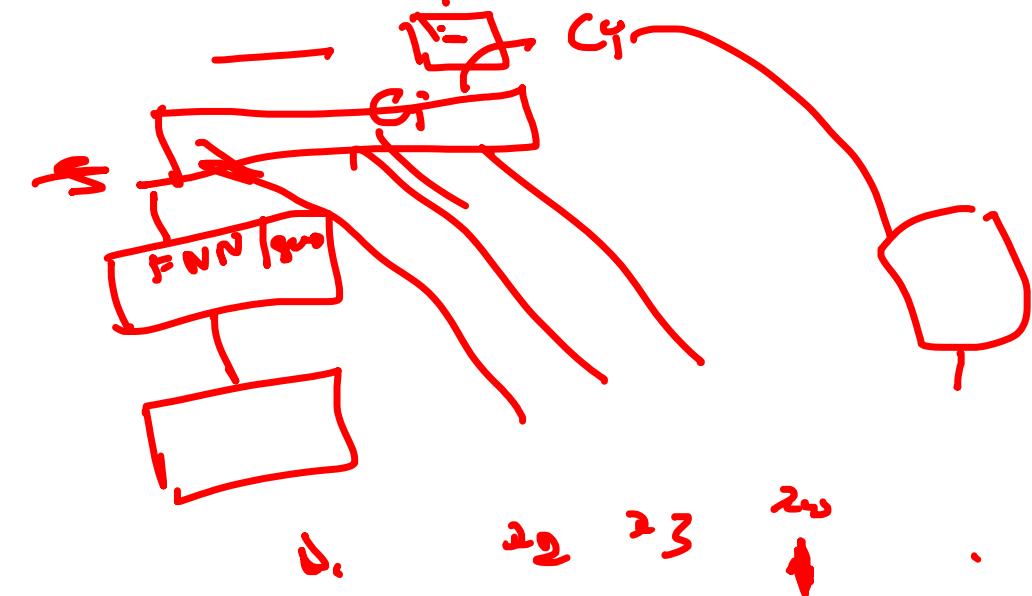
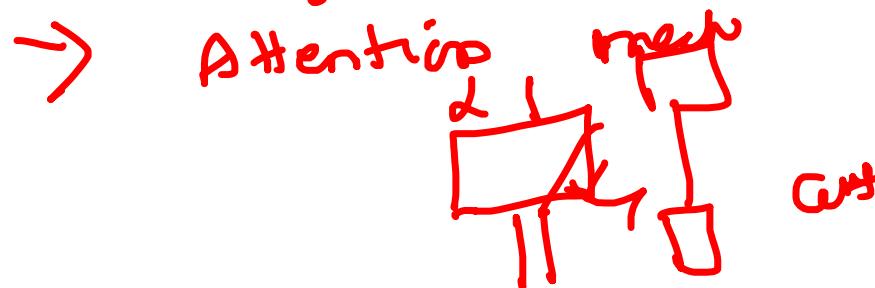
⇒ RNN →

⇒ Encoder →



3 . 3

↑ ↑ ↑  
pre

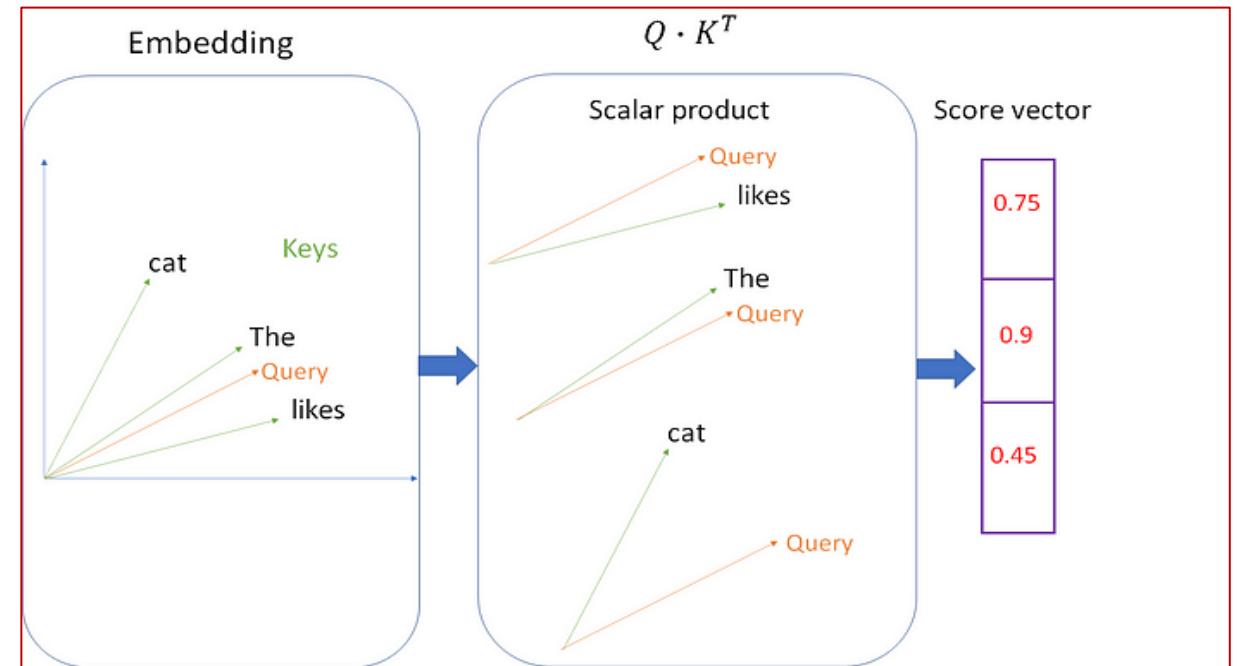
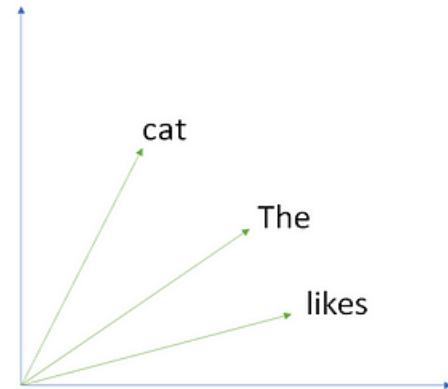


# Self-Attention Hypothetical Example

The

cat

likes



Self-attention step for an entire sequence of N tokens:

$$\text{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

$D_k$  = dimensionality of the query and key vectors  
acts as regularization and improve performance of larger models

# The cat sat on the mat

## Self attention Example

Query of the cat

3 input vectors

$x_1$   
 $x_2$   
 $x_3$

[1, 0, 1, 0]  
[0, 2, 0, 2]  
[1, 1, 1, 1]

innovate

achieve

lead

$q_i = Qx_i$  (queries)

$k_i = Kx_i$  (keys)

$v_i = Vx_i$  (values)

Weights for Key

[[0, 0, 1],  
[1, 1, 0],  
[0, 1, 0],  
[1, 1, 0]]

Key representation for Input

$$\begin{bmatrix} 1, 0, 1, 0 \end{bmatrix} \cdot \begin{bmatrix} 0, 0, 1 \\ 1, 1, 0 \\ 0, 1, 0 \\ 1, 1, 0 \end{bmatrix} = \begin{bmatrix} 0, 1, 1 \end{bmatrix}$$

(0 1 0 1)

$$1 \times 0 + 0 \times 1 + 1 \times 0 + 0 \times 1 = 0$$

Key for all inputs  
[0 1 1]  
[4 4 0]  
[2 3 1]

Weights for query

[[1, 0, 1],  
[1, 0, 0],  
[0, 0, 1],  
[0, 1, 1]]

Query representation for Input

$$\begin{bmatrix} 1, 0, 1, 0 \end{bmatrix} \cdot \begin{bmatrix} 1, 0, 1 \\ 1, 0, 0 \\ 0, 0, 1 \\ 0, 1, 1 \end{bmatrix} = [1, 0, 2]$$

Query for all inputs  
[1, 0, 2]  
[2, 2, 2]  
[2, 1, 3]

Weights for Value

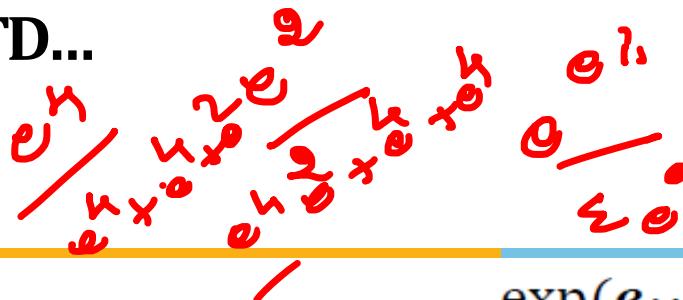
[[0, 2, 0],  
[0, 3, 0],  
[1, 0, 3],  
[1, 1, 0]]

Value representation for Input

$$\begin{bmatrix} 1, 0, 1, 0 \end{bmatrix} \cdot \begin{bmatrix} 0, 2, 0 \\ 0, 3, 0 \\ 1, 0, 3 \\ 1, 1, 0 \end{bmatrix} = [1, 2, 3]$$

Value for all inputs  
[1, 2, 3]  
[2, 8, 0]  
[2, 6, 3]

CONTD...



$$e_{ij} = \mathbf{q}_i^T \mathbf{k}_j \quad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_j \exp(e_{ij'})}$$

3 attention scores for first input  $i$  :  $(e_{ij})$   
 $[0, 1, 1]$

$$[1, 0, 2] \cdot [4, 4, 0] = [2, 4, 4] \quad \text{[?]} \\ [2, 3, 1]$$

3 softmax attention scores ( $\alpha_{ij}$ )

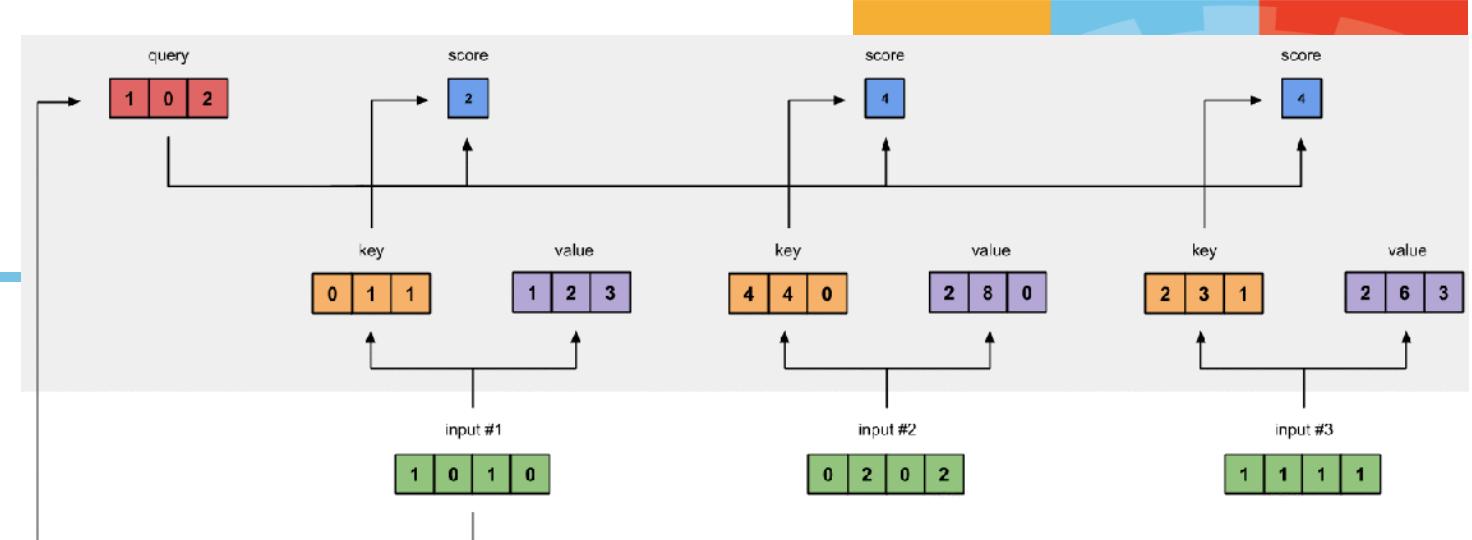
$$\text{softmax}([2, 4, 4]) = [0.0, 0.5, 0.5]$$

weighted values

$$1: 0.0 * [1, 2, 3] = [0.0, 0.0, 0.0]$$

$$2: 0.5 * [2, 8, 0] = [1.0, 4.0, 0.0]$$

$$3: 0.5 * [2, 6, 3] = [1.0, 3.0, 1.5]$$



Output 1, which is based on the query representation from Input 1 interacting with all other keys, including itself

$$\begin{aligned} & [0.0, 0.0, 0.0] \\ & + [1.0, 4.0, 0.0] \\ & + [1.0, 3.0, 1.5] \\ \hline & = [2.0, 7.0, 1.5] \end{aligned}$$

$$o_i = \sum_j \alpha_{ij} v_i$$

All the outputs after first iteration

$$[2.0, 7.0, 1.5], \# \text{ Output 1}$$

$$[2.0000, 8.0, 0.0], \# \text{ Output 2}$$

$$[2.0, 7.8, 0.3] \# \text{ Output 3}$$

# Multihead-Attention | Transformers

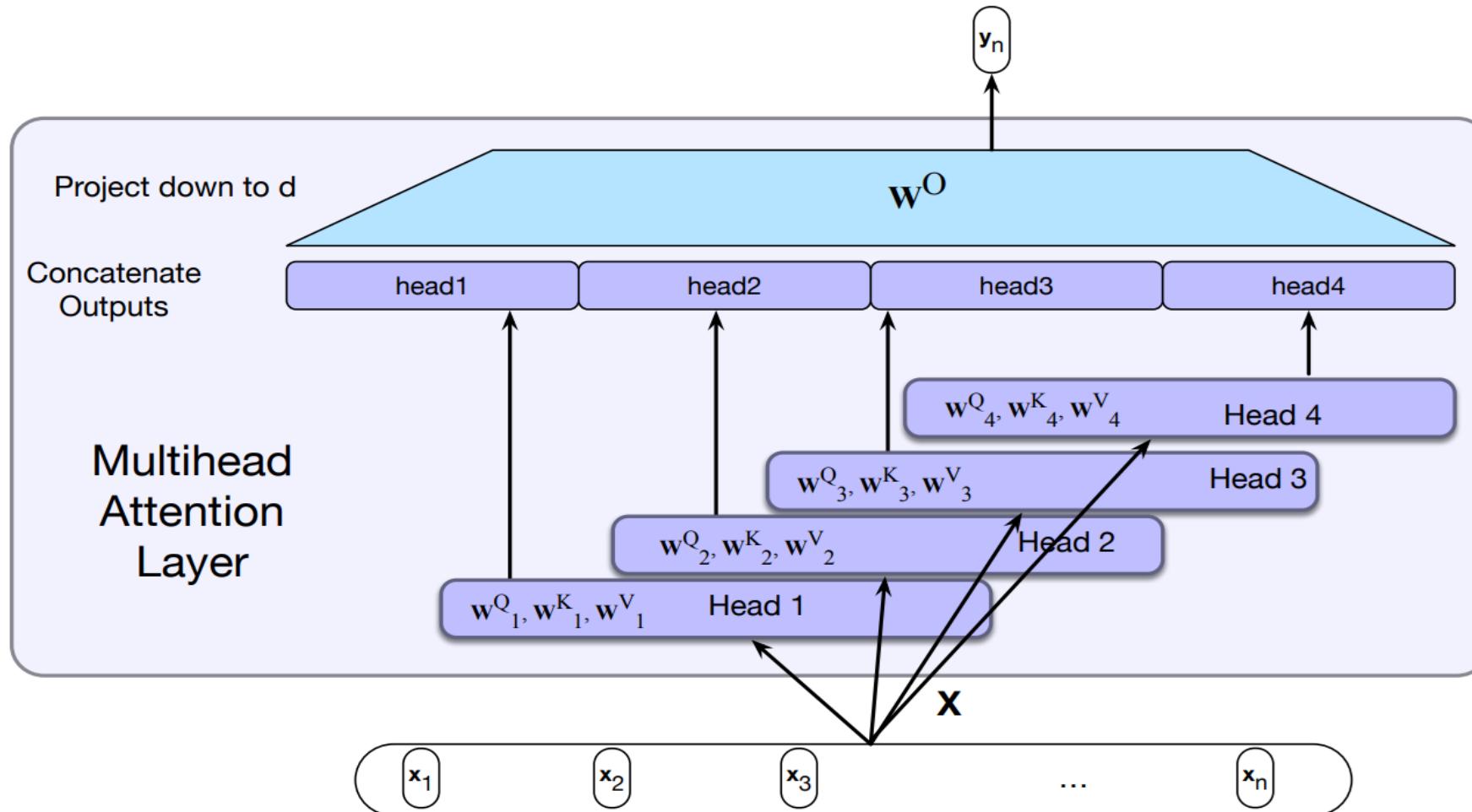
- Different words in a sentence can relate to each other in many different ways simultaneously
  - >> A single transformer block to learn to capture all of the different kinds of parallel relations among its inputs is inadequate.
- **Multihead** self-attention layers
  - >> **Heads** ⇒ sets of self-attention layers, that reside in parallel layers at the same depth in a model, each with its own set of parameters.
  - >> Each head learn different aspects of the relationships that exist among inputs at the same level of abstraction

$$\text{MultiHeadAttention}(\mathbf{X}) = (\mathbf{head}_1 \oplus \mathbf{head}_2 \dots \oplus \mathbf{head}_h) \mathbf{W}^O$$

$$\mathbf{Q} = \mathbf{X} \mathbf{W}_i^Q ; \mathbf{K} = \mathbf{X} \mathbf{W}_i^K ; \mathbf{V} = \mathbf{X} \mathbf{W}_i^V$$

$$\mathbf{head}_i = \text{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$$

# Multihead-Attention | Transformers



Each of the multihead self-attention layers is provided with its own set of key, query and value weight matrices.

The outputs from each of the layers are concatenated and then projected down to  $d$ , thus producing an output of the same size as the input so layers can be stacked.

# Barriers and solutions for Self-Attention as a building block

## Barriers

- Doesn't have an inherent notion of order!
- No nonlinearities for deep learning magic! It's all just weighted averages
- Need to ensure we don't "look at the future" when predicting a sequence
  - Like in machine translation
  - Or language modeling

## Solutions

- Add position representations to the inputs
- Easy fix: apply the same feedforward network to each self- attention output.
- Mask out the future by artificially setting attention weights to 0!

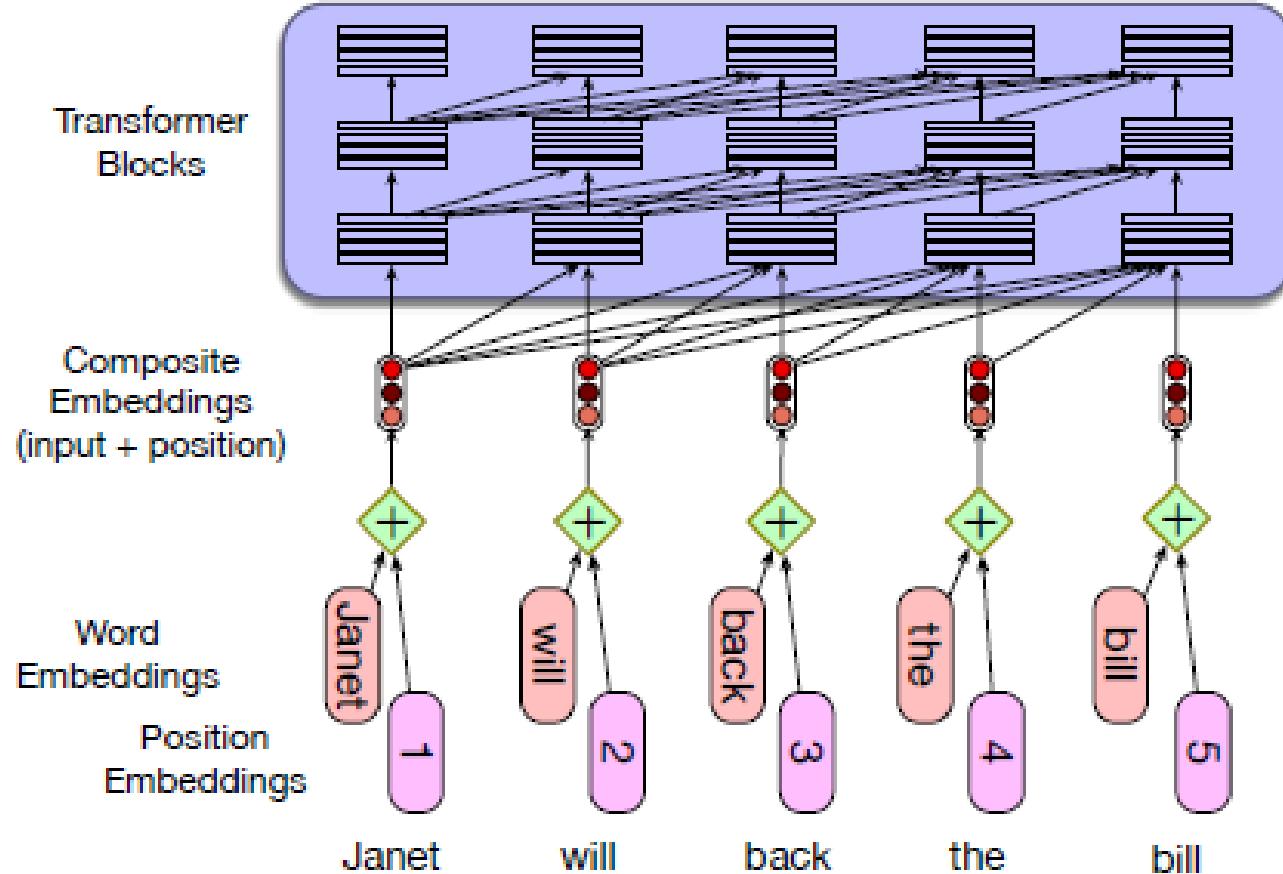
# Fixing the first self-attention problem: sequence order



- With RNNs, information about the order of the inputs was built into the structure of the model.
- self-attention ditches sequential operations in favor of parallel computation
- Since self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries, and values.
- Consider representing each **sequence index** as a **vector**  
 $p_i \in \mathbb{R}^d$ , for  $i \in \{1, 2, \dots, n\}$  are position vectors
- Easy to incorporate this info into our self-attention block: just add the  $p_i$  to our inputs!
- $x_i$  is the embedding of the word at index  $i$ . The positioned embedding is:

$$\tilde{x}_i = x_i + p_i$$

# Position encoding : simple way

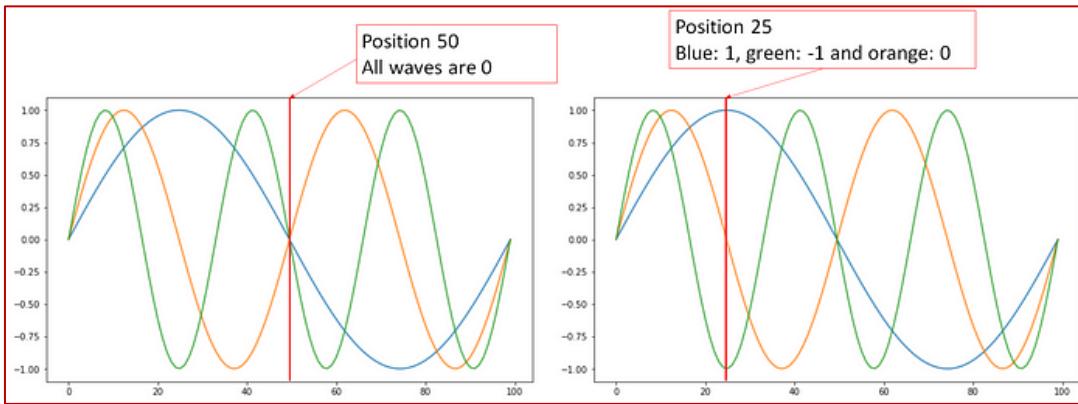


**Figure 9.20** A simple way to model position: simply adding an embedding representation of the absolute position to the input word embedding.

# Position representation vectors through sinusoids

**Sinusoidal position representations:** concatenate sinusoidal functions of varying periods

$$p_i = \begin{Bmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{Bmatrix}$$



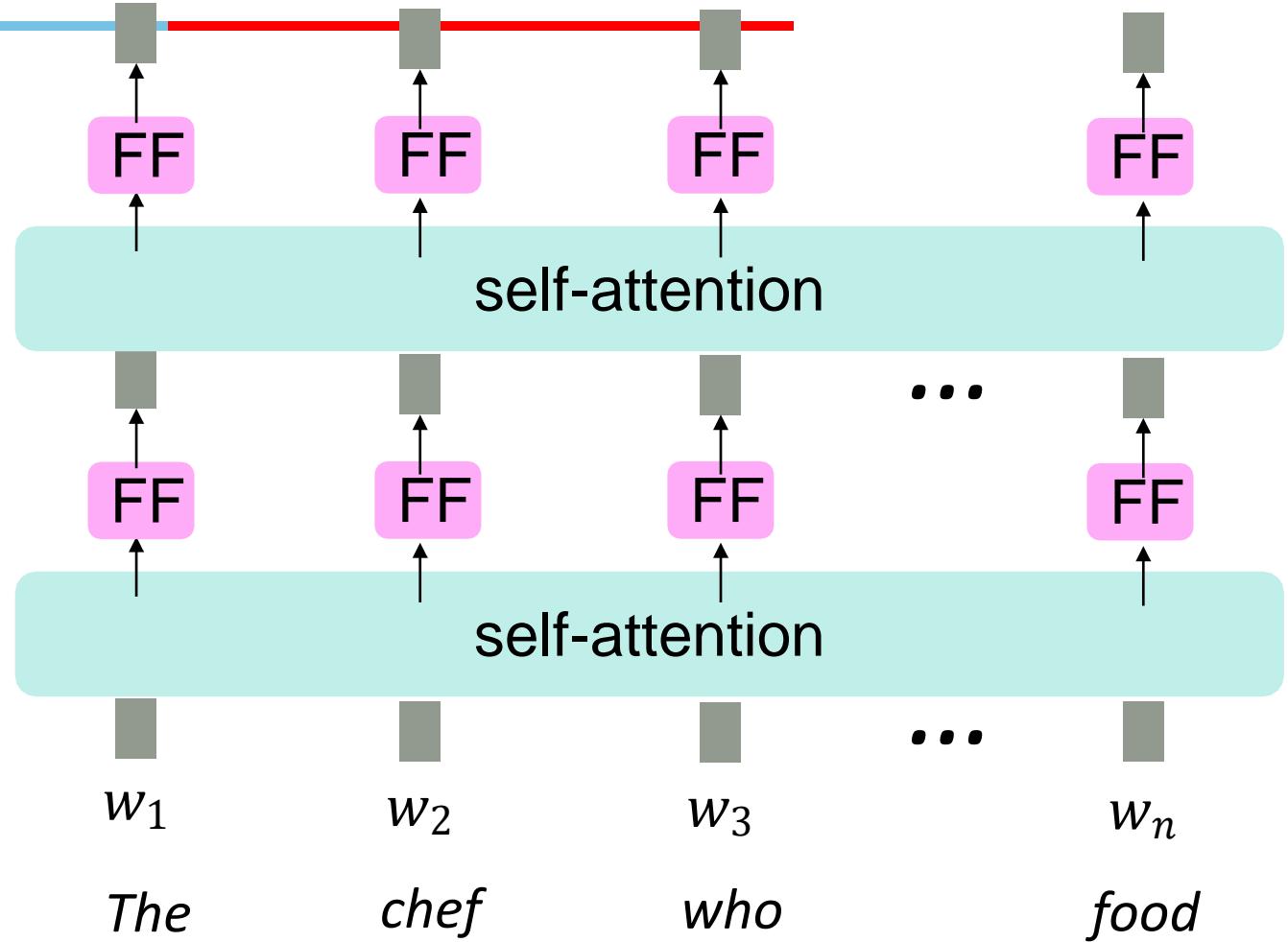
$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

Sequence	Index of token, k	Positional Encoding Matrix with d=4, n=100			
		i=0	i=0	i=1	i=1
I	0	P <sub>00</sub> =sin(0) = 0	P <sub>01</sub> =cos(0) = 1	P <sub>02</sub> =sin(0) = 0	P <sub>03</sub> =cos(0) = 1
am	1	P <sub>10</sub> =sin(1/1) = 0.84	P <sub>11</sub> =cos(1/1) = 0.54	P <sub>12</sub> =sin(1/10) = 0.10	P <sub>13</sub> =cos(1/10) = 1.0
a	2	P <sub>20</sub> =sin(2/1) = 0.91	P <sub>21</sub> =cos(2/1) = -0.42	P <sub>22</sub> =sin(2/10) = 0.20	P <sub>23</sub> =cos(2/10) = 0.98
Robot	3	P <sub>30</sub> =sin(3/1) = 0.14	P <sub>31</sub> =cos(3/1) = -0.99	P <sub>32</sub> =sin(3/10) = 0.30	P <sub>33</sub> =cos(3/10) = 0.96

Positional Encoding Matrix for the sequence 'I am a robot'

# Adding nonlinearities in self-attention

- Note that there are **no elementwise nonlinearities in self-attention**; stacking more self-attention layers just re-averages **value** vectors
- Easy fix: add a **feed-forward network** to post-process each output vector.



Intuition: the FF network processes the result of attention

# Masking the future in self-attention

$$\text{SelfAttention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

- Calculation of the comparisons in QK results in a score for each query value to every key value, **including those that follow the query**
- Inappropriate in the setting of language modeling
- To use self-attention in **decoders**, we need to ensure we can't peek at the future.
- To enable parallelization, we **mask out attention** to future words by setting attention scores to  $-\infty$

$$e_{ij} = \begin{cases} q_i^T k_j, & j \leq i \\ -\infty, & j > i \end{cases}$$

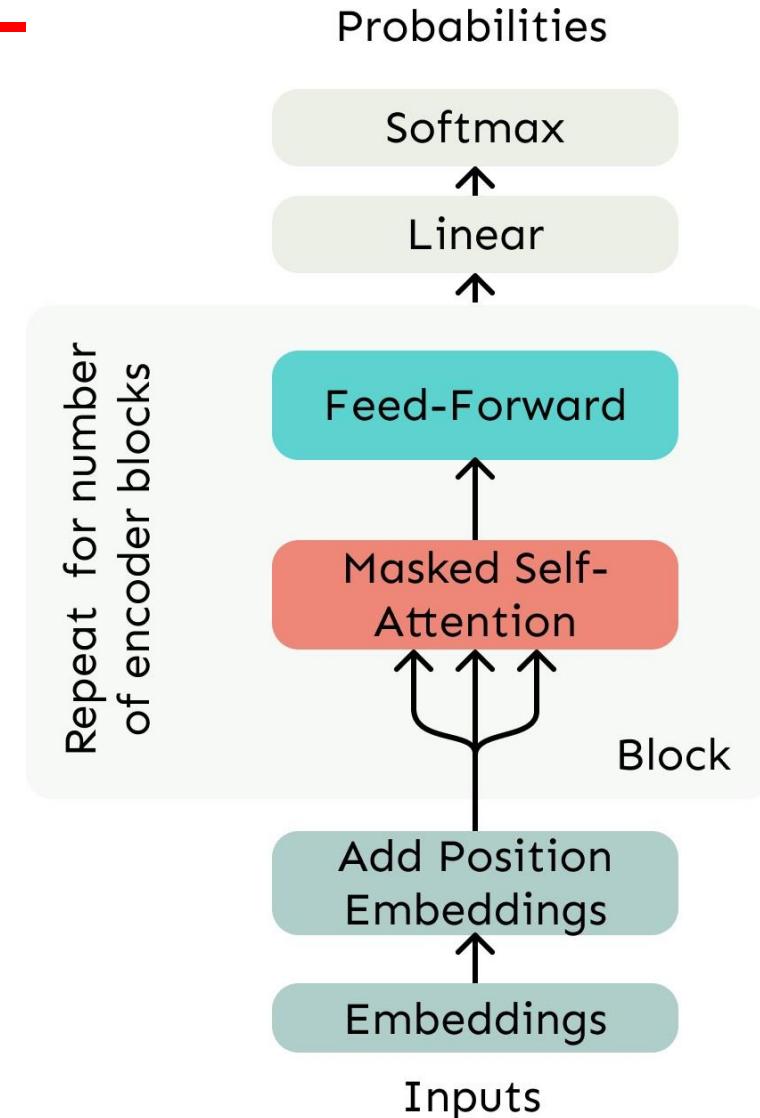
For encoding  
these words

We can look at these (not greyed out) words

[START]	The	chef	who
[START]	$-\infty$	$-\infty$	$-\infty$
The		$-\infty$	$-\infty$
chef			$-\infty$
who			

# Necessities for a self-attention building block

- **Self-attention:**
  - the basis of the method.
- **Position representations:**
  - Specify the sequence order, since self-attention is an unordered function of its inputs.
- **Nonlinearities:**
  - At the output of the self-attention block
  - Frequently implemented as a simple feed-forward network.
- **Masking:**
  - In order to parallelize operations while not looking at the future.
  - Keeps information about the future from “leaking” to the past.



# Layer normalization

- **Layer normalization** is a trick to help models train faster.

Layer norm is a variation of the standard score, or z-score, from statistics applied to a single hidden layer

- Let  $x \in \mathbb{R}^d$  be an individual (word) vector in the model.

$$\hat{x} = \frac{(x - \mu)}{\sigma}$$

$$z = \text{LayerNorm}(x + \text{SelfAttn}(x))$$

$$y = \text{LayerNorm}(z + \text{FFNN}(z))$$

$$\mu = \frac{1}{d_h} \sum_{i=1}^{d_h} x_i$$

$$\sigma = \sqrt{\frac{1}{d_h} \sum_{i=1}^{d_h} (x_i - \mu)^2}$$

$\gamma$  and  $\beta$  are learnable parameters

$$\text{LayerNorm} = \gamma \hat{x} + \beta$$

# Residual connections

- **Residual connections** are a trick to help models train better.
- Pass information from a lower layer to a higher layer without going through the intermediate layer.
- Allowing information from the activation going forward and the gradient going backwards to skip a layer

Instead of  $X^{(i)} = \text{Layer}(X^{(i-1)})$  (where  $i$  represents the layer)

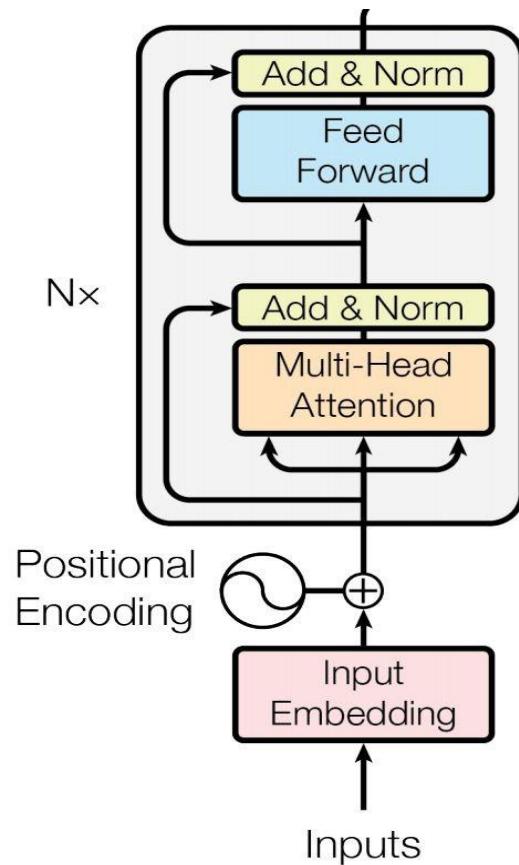


We let  $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$   
 (so we only have to learn “the residual” from the previous layer)



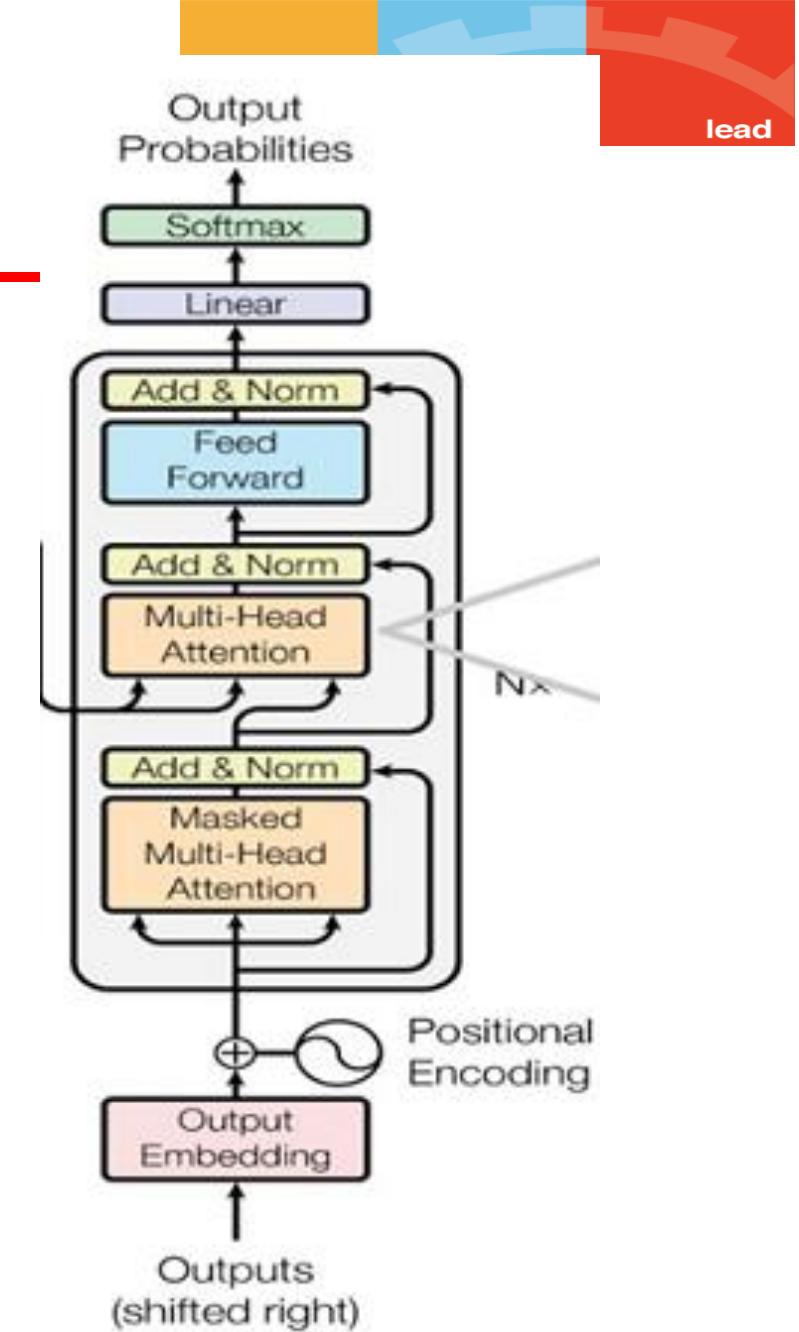
# The Transformer Encoder

- It receives an input and **builds a representation of it (its features): contextual embeddings.**
- It is optimized to acquire understanding from the input
- **Bidirectional** model
- A stack of  $N \times = 6$  identical layers
- Multi-headed self attention
  - Models context
- Feed-forward layers
  - Computes non-linear hierarchical features
- Layer norm and residuals
  - Makes training deep networks healthy
- Positional embeddings
  - Allows model to learn relative positioning

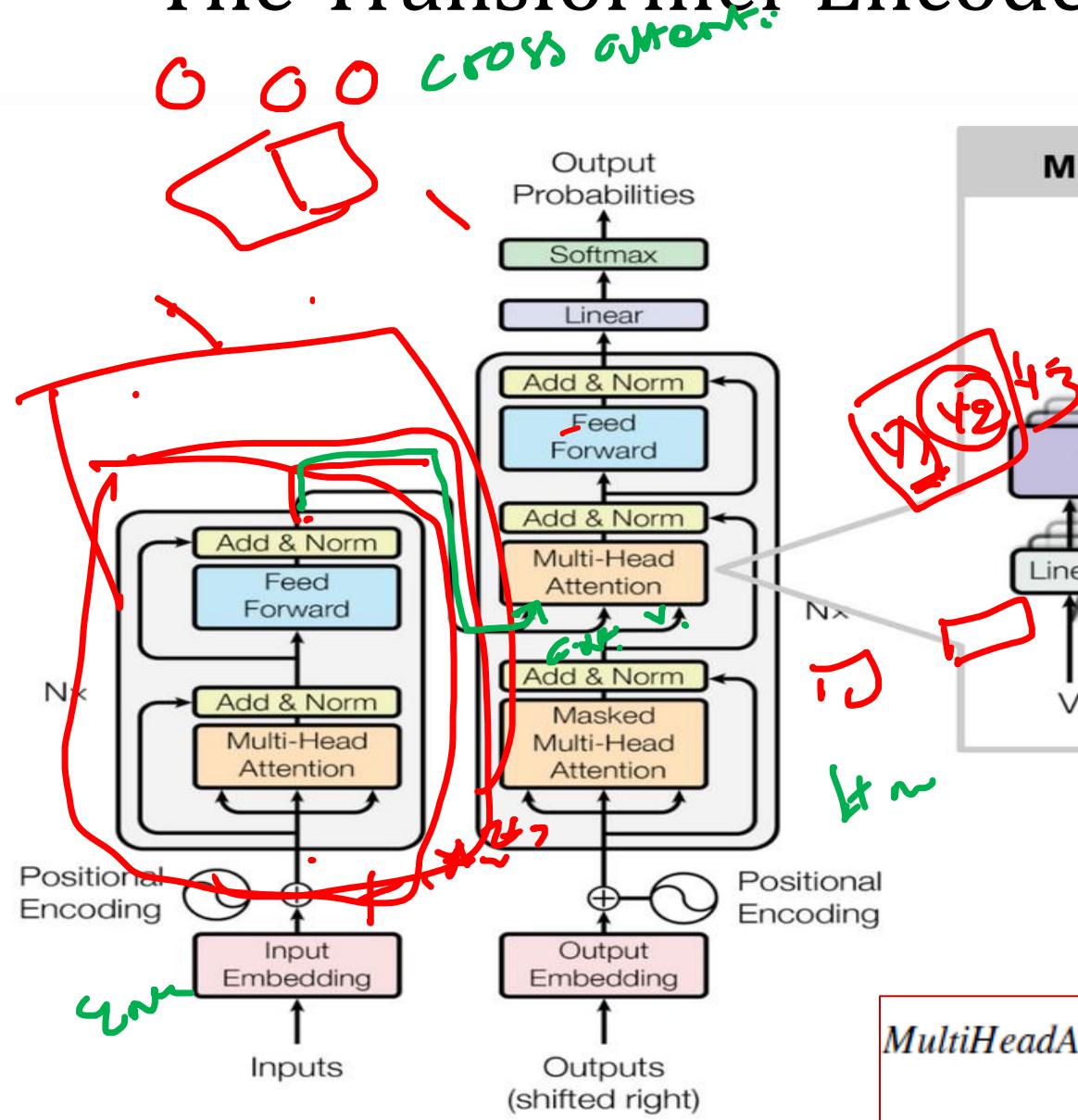


# The Transformer Decoder

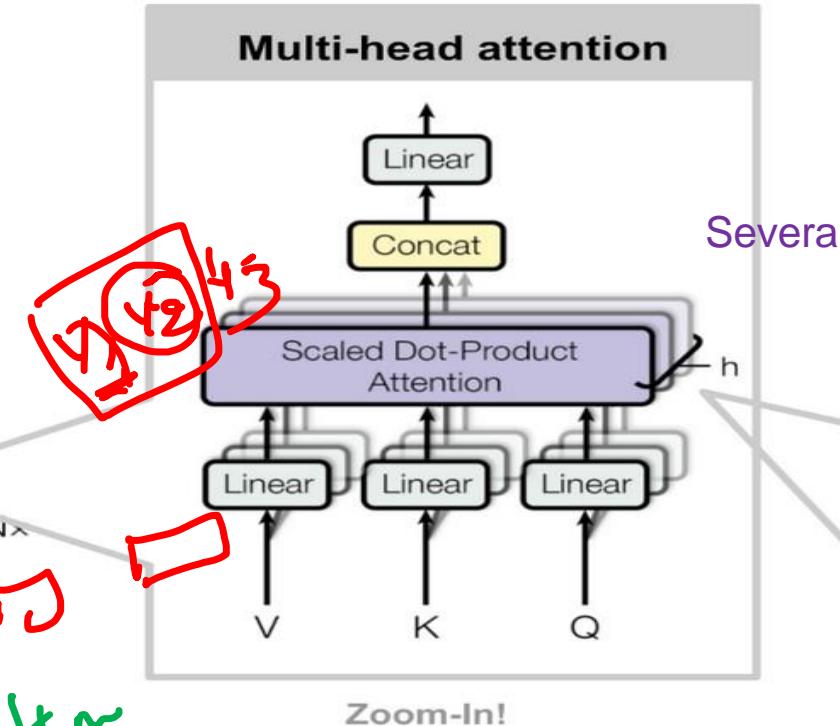
- A **conditional language model** that attends to the encoder representation and generates the target words one by one, at each timestep
- conditioning on the source sentence and the previously generated target language words.
- Transformer Decoder is modified to perform **cross-attention** (also sometimes called encoder-decoder attention or source attention) to the output of the Encoder
- The decoder, works sequentially and can only pay attention to the words in the sentence that it has already translated (Masked attention layer)
- **Unidirectional model**



# The Transformer Encoder-Decoder



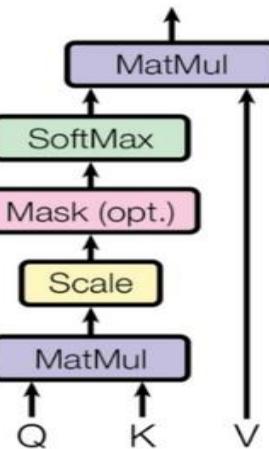
## Multi-head attention



$$\text{SelfAttention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Several attention layers run in parallel

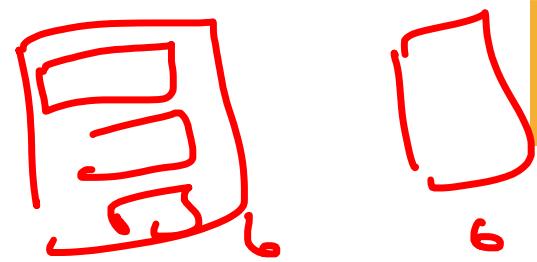
## Scaled dot-product attention



Zoom-In!

$$\begin{aligned} \text{MultiHeadAttn}(X) &= (\text{head}_1 \oplus \text{head}_2 \dots \oplus \text{head}_h)W^O \\ Q &= XW_i^Q ; K = XW_i^K ; V = XW_i^V \\ \text{head}_i &= \text{SelfAttention}(Q, K, V) \end{aligned}$$

FFN  
SelfAttention



→ The cat sat on the mat 4) Positional encoding.

Tokens(2) "The" "cat" "sat" "on" "the" "mat" → [0.2 0.6 ...]

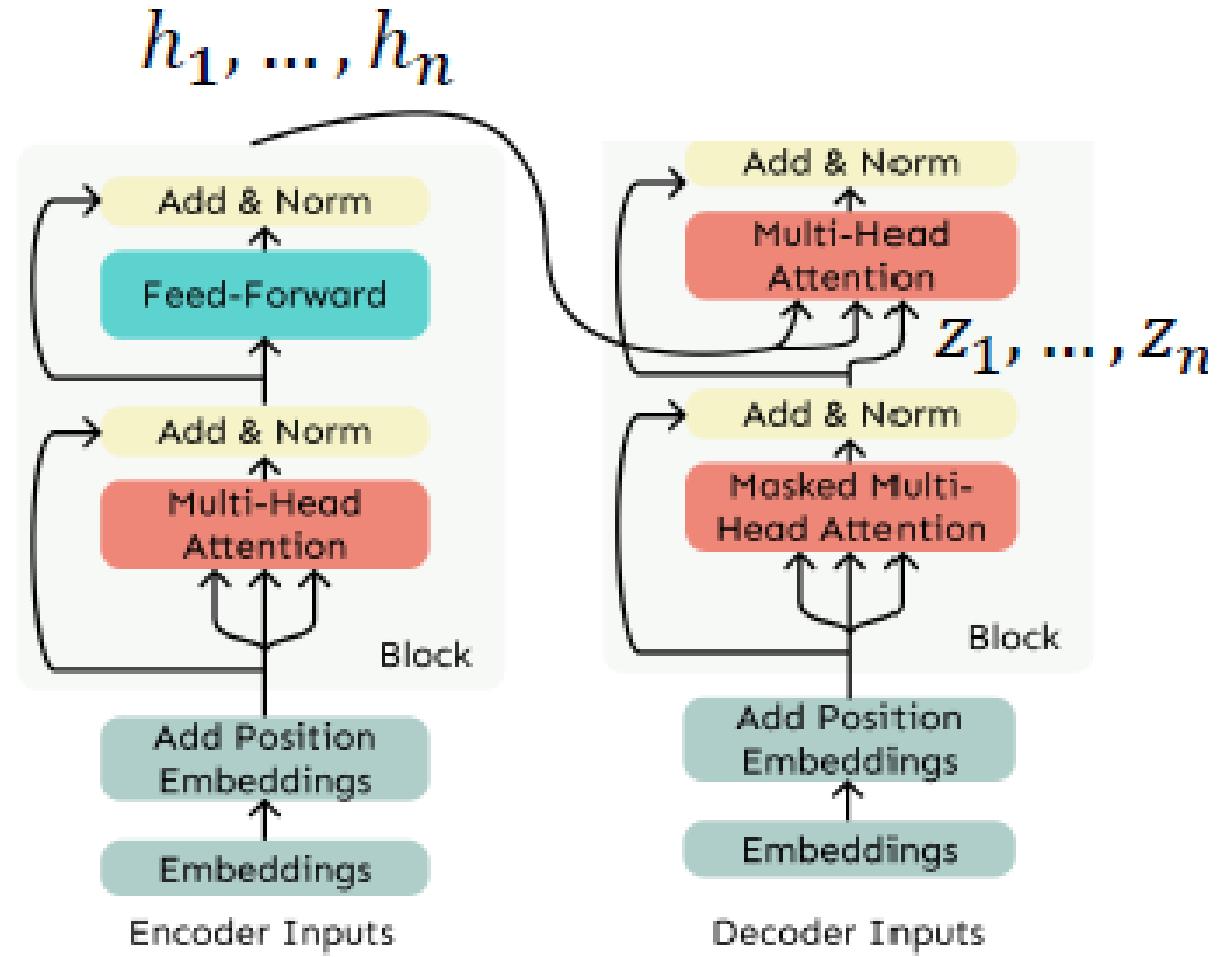
embedding(3) The → [0.2 0.3 0.4] → The → [0.2 0.6 ...]  
cat → [0.5 0.1 0.7]  
sat → [0.9 0.6 0.2]  
on → [0.3 0.8 0.5]  
the → [0.4 0.2 0.9]  
mat → [0.7 0.4 0.6]

# Cross-attention (details)

- In self-attention keys, queries, and values come from the same source.
- In cross attention,
- Let  $h_1, \dots, h_n$  be **output vectors from the Transformer encoder**;  $x_i \in \mathbb{R}^d$
- Let  $z_1, \dots, z_n$  be input vectors from the Transformer **decoder**,  $z_i \in \mathbb{R}^d$
- queries as usual come from the previous layer of the decoder  $q_i = Qz_i$
- keys and values come from the output of the encoder  $k_i = Kh_i, v_i = Vh_i$

$$\mathbf{Q} = \mathbf{W}^Q \mathbf{H}^{dec[i-1]}, \quad \mathbf{K} = \mathbf{W}^K \mathbf{H}^{enc}, \quad \mathbf{V} = \mathbf{W}^V \mathbf{H}^{enc}$$

$$CrossAttention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}$$

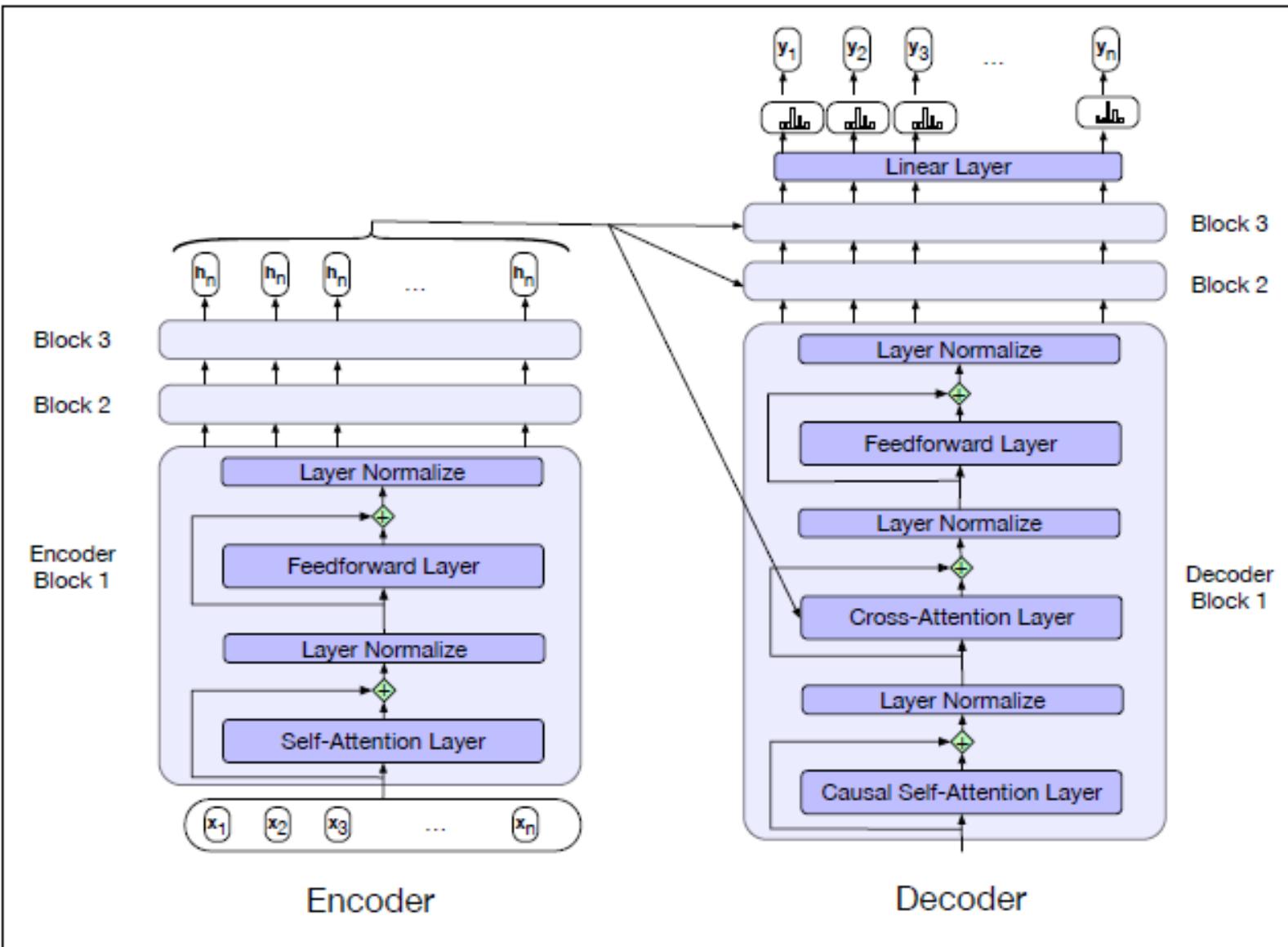


# The transformer block for the encoder and the decoder

te

achieve

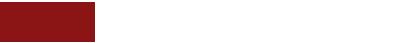
lead



**Figure 10.16** The transformer block for the encoder and the decoder. Each decoder block has an extra cross-attention layer, which uses the output of the final encoder layer  $H^{enc} = h_1, \dots, h_n$  to produce its key and value vectors.

# Word structure and subword models

- Common words end up being a part of the subword vocabulary, while rarer words are split into (sometimes intuitive, sometimes not) components.
- In the worst case, words are split into as many subwords as they have characters.

	word	vocab mapping	embedding
Common words	hat	→ hat	
	learn	→ learn	
Variations	taaaaasty	→ taa## aaa## sty	
	laern	→ la## ern##	
misspellings			
novel items	Transformerify	→ Transformer## ify	

# Text tokenization

- Instead of
  - white-space segmentation
  - single-character segmentation
- **Use the data** to tell us how to tokenize.
- **Subword tokenization** (because tokens can be parts of words as well as whole words)

*One word.*

# Subword tokenization

---

- Three common algorithms:
  - **Byte-Pair Encoding (BPE)** (Sennrich et al., 2016)
  - **Unigram language modeling tokenization** (Kudo, 2018)
  - **WordPiece** (Schuster and Nakajima, 2012)
- All have 2 parts:
  - A token **learner** that takes a raw training corpus and induces a vocabulary (a set of tokens).
  - A token **segmenter** that takes a raw test sentence and tokenizes it according to that vocabulary

# WordPiece tokenization algorithm

---

- WordPiece is the tokenization algorithm Google developed to pretrain BERT.
- Starts from a small vocabulary : special tokens used by the model and the initial alphabet.
- Identifies subwords by adding a prefix (like ## for BERT),

**"word" gets split like this: w ##o ##r ##d**

- Initial vocabulary contains all the characters present at the beginning of a word and the characters present inside a word preceded by the WordPiece prefix.

# WordPiece tokenization algorithm

- Instead of selecting the most frequent pair (BPE), WordPiece computes a score for each pair

score=(freq\_of\_pair)/(freq\_of\_first\_element × freq\_of\_second\_element)

- Algorithm prioritizes the merging of pairs where the individual parts are less frequent in the vocabulary.**
  - It won't necessarily merge ("un", "#able") even if that pair occurs very frequently in the vocabulary, because the two pairs "un" and "#able" will likely each appear in a lot of other words and have a high frequency.
  - ("hu", "#gging") will probably be merged faster (assuming the word "hugging" appears often in the vocabulary) since "hu" and "#gging" are likely to be less frequent individually.

# Tokenization example using wordpiece



- Corpus uses these five words: "hug", "pug", "pun", "bun", "hugs"
- Assume the words had the following frequencies: ("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)
- Training: token **learner** : splitting words and adding a prefix ## for subwords
  - ("h" "##u" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12),
  - ("b" "##u" "##n", 4), ("h" "##u" "##g" "##s", 5)
- First iteration:  
 initial vocabulary : ["b", "h", "p", "##g", "##n", "##s", "##u"] (ignoring special tokens)
  - Score ("##u", "##g") : pair freq/individual freq =  $20/36 \times 20 = 1/36$
  - Score ("##g", "##s") : pair freq/individual freq =  $5/20 \times 5 = 1/20$
  - Likewise compute scores for all the pairs
  - first merge learned is ("##g", "##s") -> **("##gs")**
- Vocabulary after first iteration: ["b", "h", "p", "##g", "##n", "##s", "##u", **"##gs"** ]
- Repeat iterations till you get a desired vocab size.

# Tokenization example using wordpiece

- Token **segmenter**: New inputs are tokenized by applying the following steps:
  - Pre-tokenize it
  - Split it
  - Apply the tokenization algorithm on each word.
    - Look for the biggest subword starting at the beginning of the first word and split it
    - Repeat the process on the second part
    - And so on for the rest of that word and the following words in the text
- Continuing the example: suppose the final vocab learnt:
  - Vocabulary: **["b", "h", "p", "#g", "#n", "#s", "#u", "#gs", "hu", "hug"]**
  - Corpus: ("hug", 10), ("p" "#u" "#g", 5), ("p" "#u" "#n", 12), ("b" "#u" "#n", 4), ("hu" "#gs", 5)
  - **Test word : "hugs"**
  - **Tokenization of "hugs" is ["hug", "#s"]**

# Pre-training in NLP

- Word embeddings are the basis of deep learning for NLP

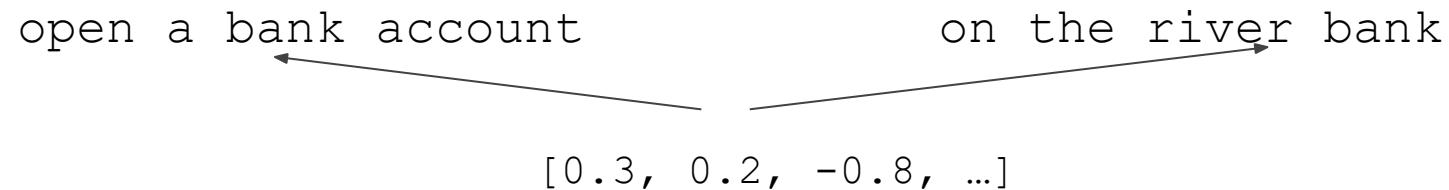


- Word embeddings (word2vec, GloVe) are often *pre-trained* on text corpus from co-occurrence statistics



# Contextual Representations

- Problem: Word embeddings are applied in a context free manner



- Solution: Train contextual representations on text corpus

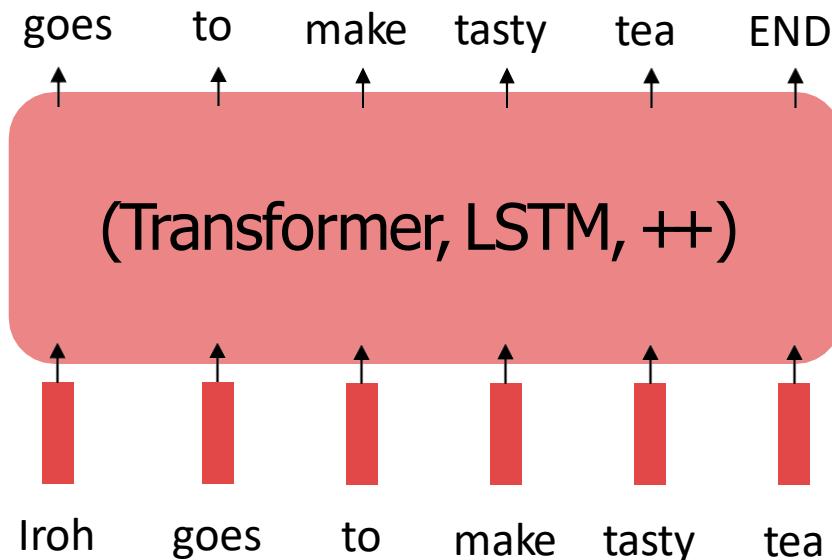


# The Pre training / Fine tuning Paradigm

Pretraining can improve NLP applications by serving as parameter initialization.

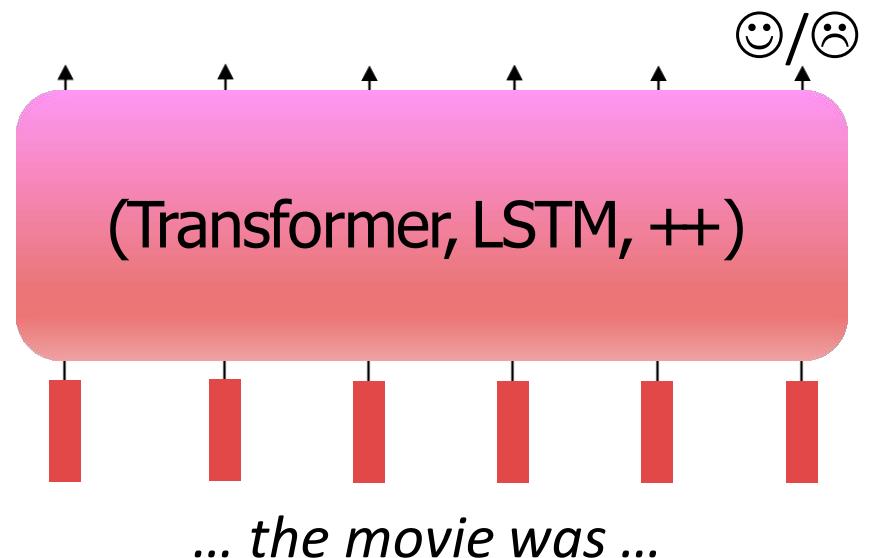
## Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



## Step 2: Finetune (on your task)

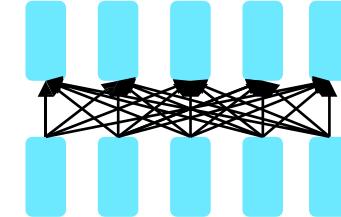
Not many labels; adapt to the task!



# Pre-training for three types of architectures

## Encoder-only models

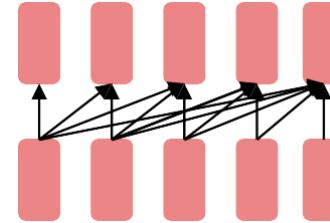
- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?
- Also known as *auto-encoding models*
- For tasks that require understanding of the input, e.g sentence classification and named entity recognition
- Representatives of this family of models include:
  - ALBERT, BERT, DistilBERT, ELECTRA, RoBERTa



# Pre-training for three types of architectures

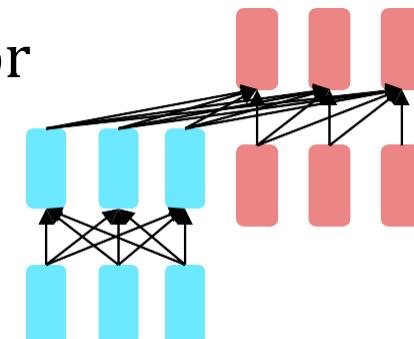
## Decoder-only models

- Language models
- Nice to generate from; can't condition on future words
- These models are also known as *auto-regressive models*.
- Representatives of this family of models include:
  - CTRL, GPT, GPT-2, Transformer XL



## Encoder-decoder models or sequence-to-sequence models

- For generative tasks that require an input, such as translation or summarization.
- BART/T5-like



# Types of language modelling

---

- *Causal language modelling*
  - predicting the next word in a sentence in which output depends on the past and present inputs, but not the future ones
  - left-to-right
- *Masked language modeling*
  - the model predicts a masked word in the sentence
  - introduced with the BERT model
  - allows the model to see entire texts at a time, including both the right and left context.

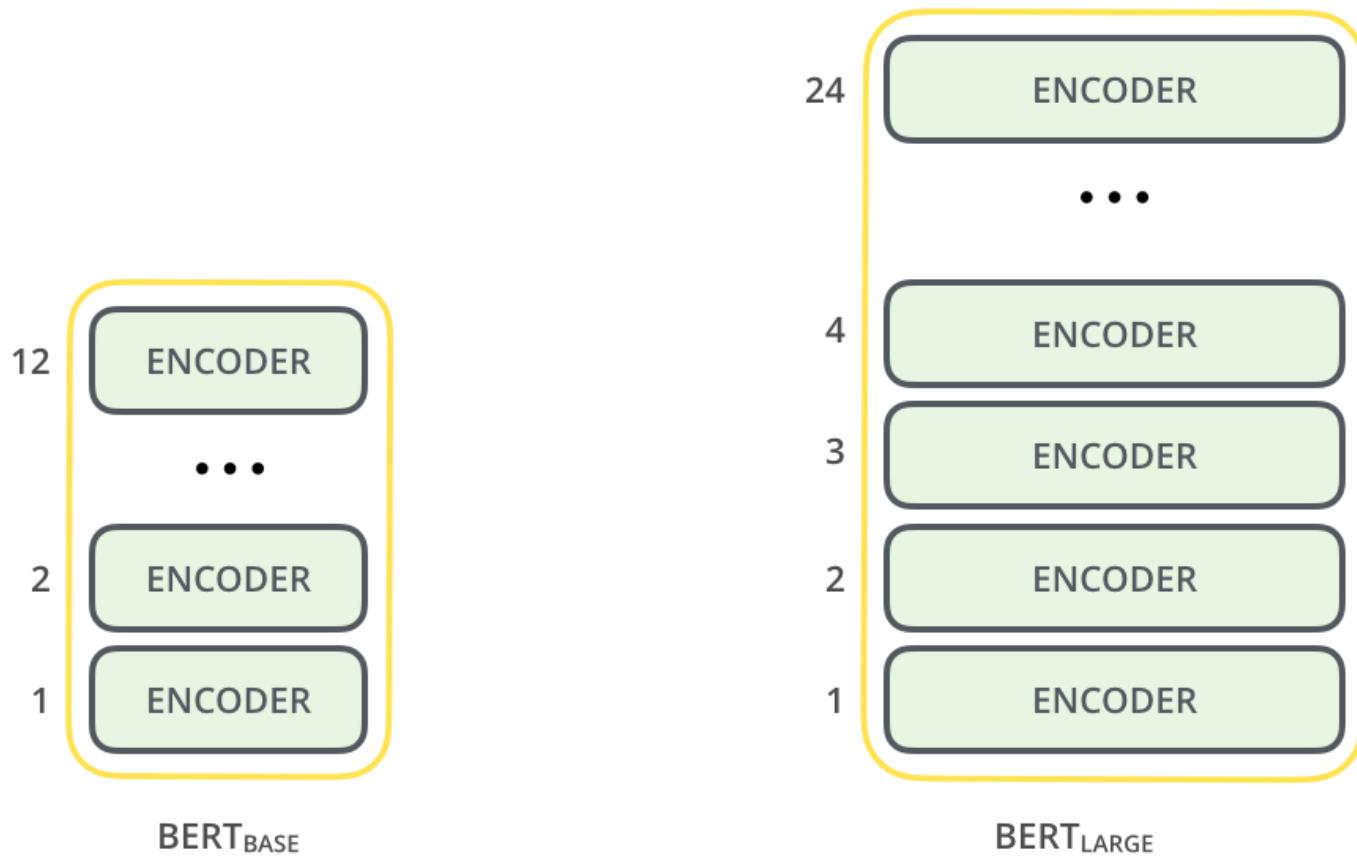
# BERT: Bidirectional Encoder Representations from Transformers



## Details about BERT

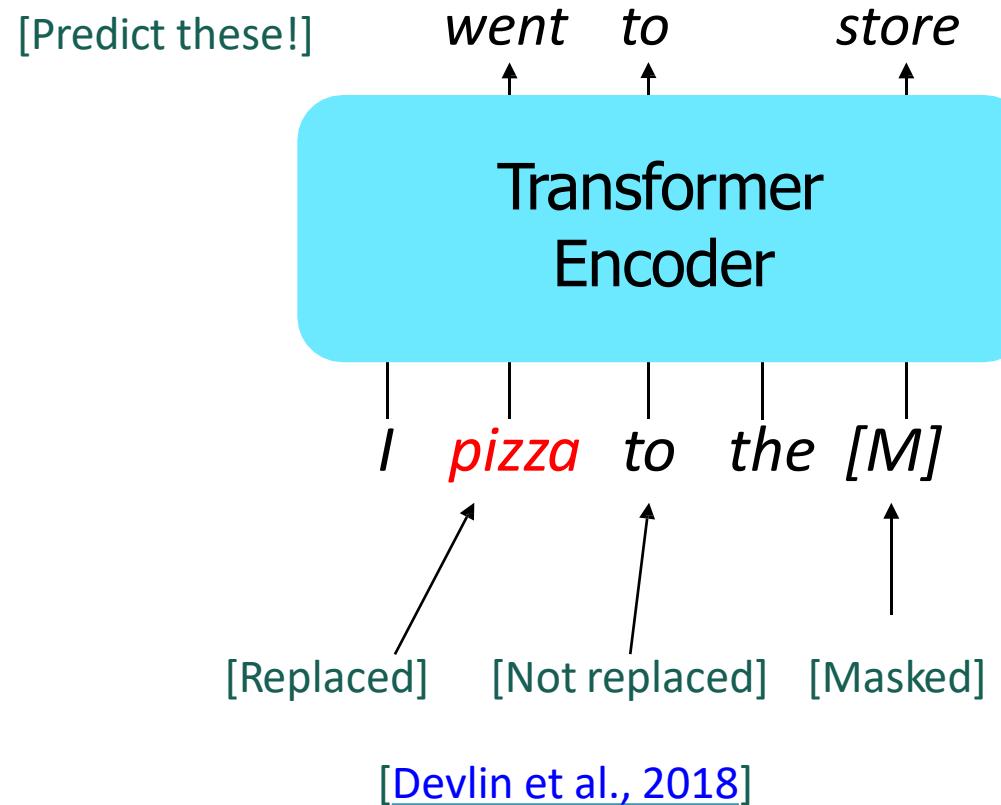
- Two models were released:
  - BERT-base: 12 layers, 768-dim hidden states, 12 attention heads, 110 million params.
  - BERT-large: 24 layers, 1024-dim hidden states, 16 attention heads, 340 million params.
- Trained on:
  - BooksCorpus (800 million words)
  - English Wikipedia (2,500 million words)
- Pre training is expensive and impractical on a single GPU.
  - BERT was pre trained with 64 TPU chips for a total of 4 days.
  - (TPUs are special tensor operation acceleration hardware)
- Fine tuning is practical and common on a single GPU
  - “Pre train once, fine tune many times.”

# BERT



# BERT Pre training Task 1: masked words

- Predict a random 15% of (sub)word tokens.
  - Replace input word with [MASK] 80% of the time
  - Replace input word with a random token 10% of the time
  - Leave input word unchanged 10% of the time (but still predict it!)



# BERT Pre training Task 1: masked words



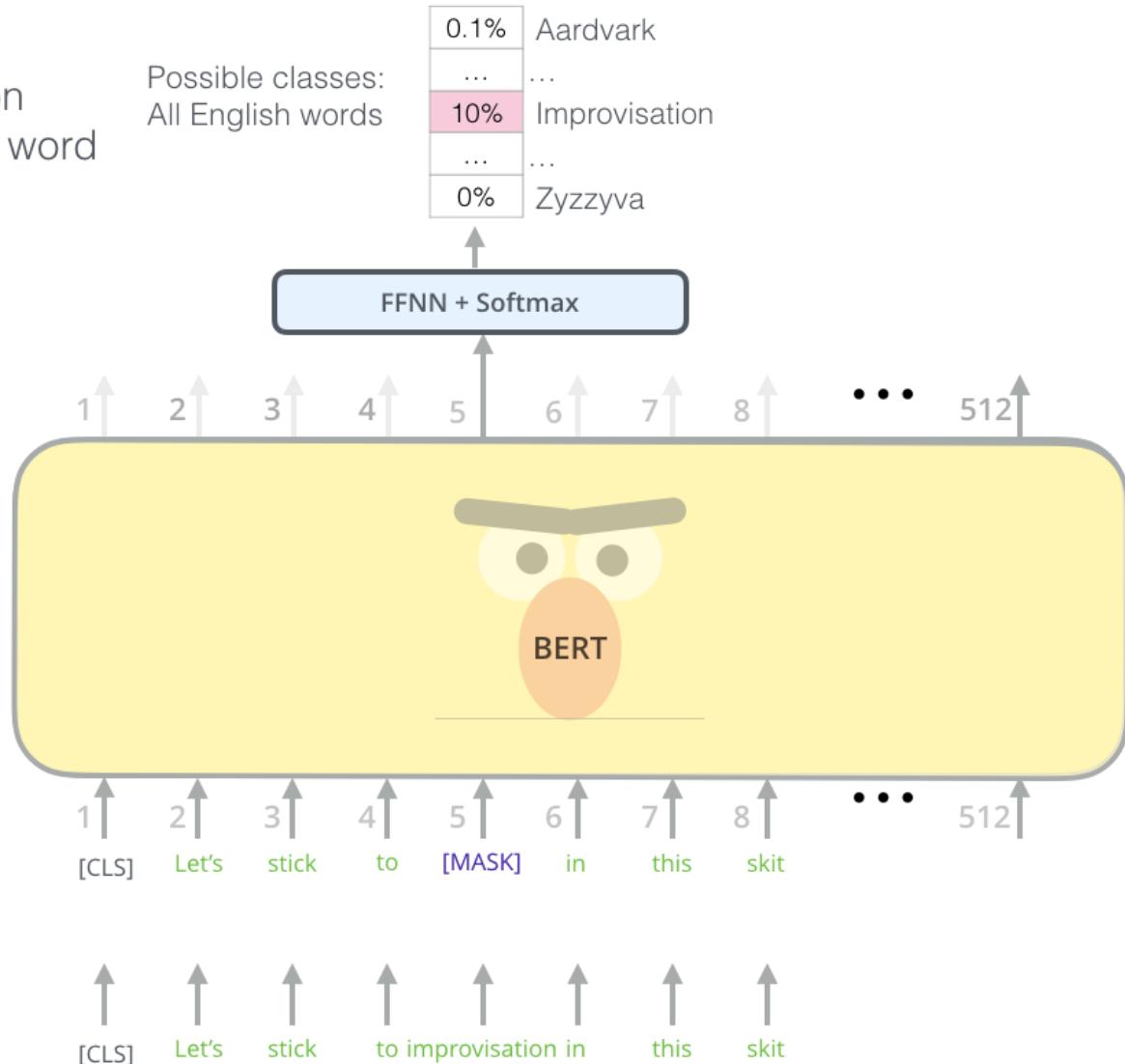
Use the output of the masked word's position to predict the masked word

Possible classes:  
All English words

0.1%	Aardvark
...	...
10%	Improvisation
...	...
0%	Zyzyva

Randomly mask  
15% of tokens

Input



# BERT Pretraining Task 2: Predict next sentence

- Intelligent about two sentences (e.g. are they simply paraphrased versions of each other?)
- To make BERT better at handling relationships between multiple sentences, the pre-training process includes an additional task: Given two sentences (A and B), is B likely to be the sentence that follows A, or not?

Predict likelihood that sentence B belongs after sentence A

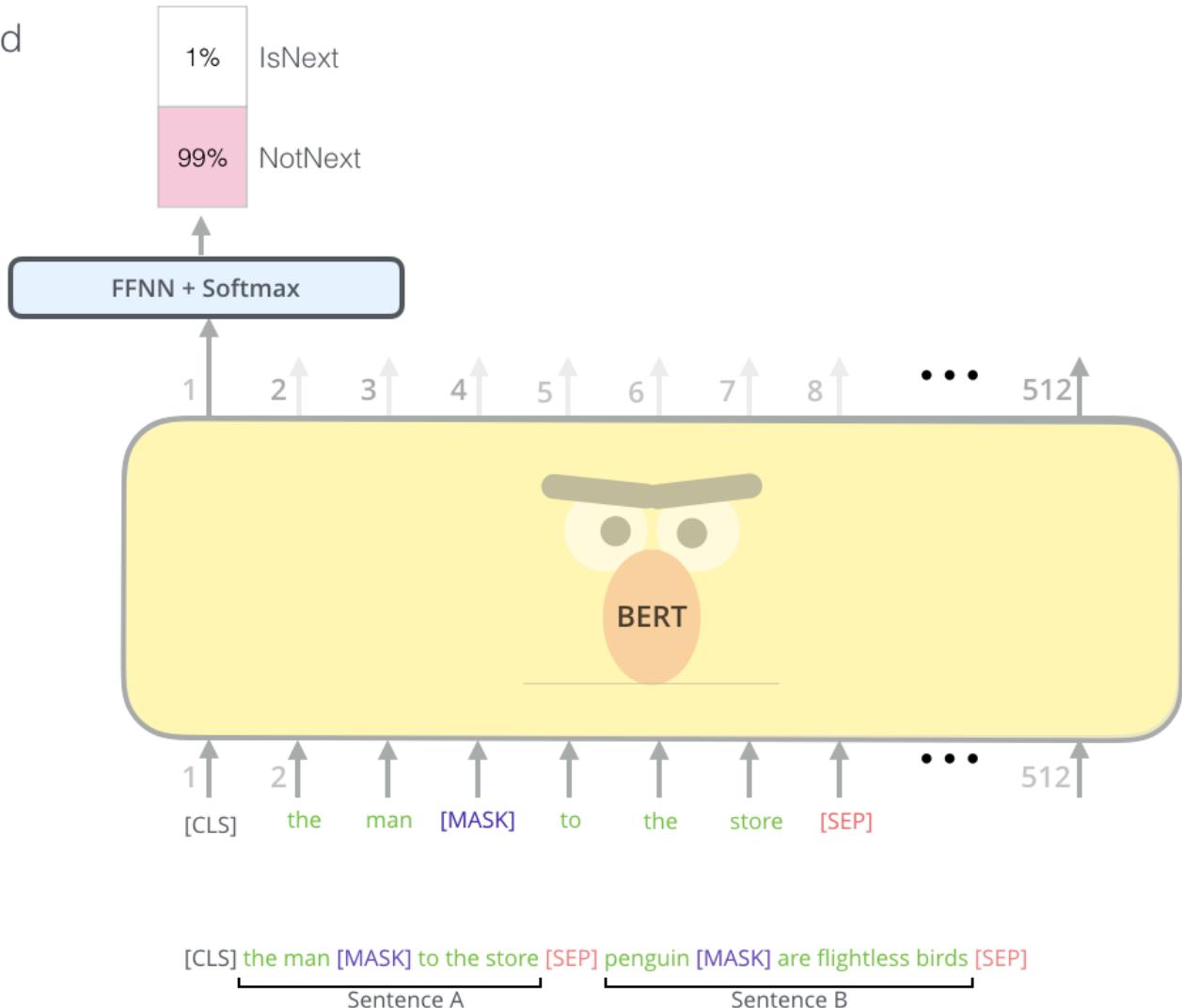
Tokenized Input

Input

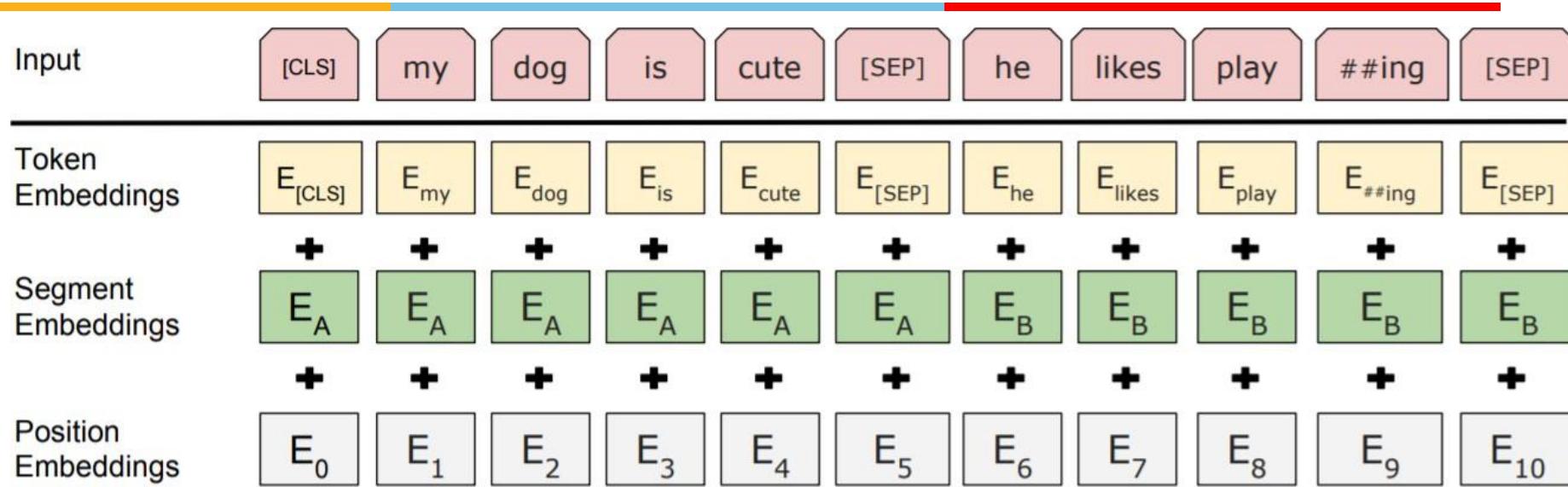
[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flightless birds [SEP]

Sentence A

Sentence B



# BERT: Embedding layer



- The Token Embeddings layer will convert each **wordpiece** token into a 768-dimensional vector representation
- The Segment Embeddings layer only has 2 vector representations. The first vector (index 0) is assigned to all tokens that belong to input 1 while the last vector (index 1) is assigned to all tokens that belong to input 2
- Classifying whether two pieces of text are semantically similar. The pair of input text are simply concatenated and fed into the model

# References



- Speech and Language Processing by Daniel Jurafsky
- <https://jalammar.github.io/illustrated-bert/>
- <https://huggingface.co/course/chapter1/4?fw=pt>
- <https://arxiv.org/abs/1706.03762>
- <https://arxiv.org/abs/1810.04805>
- <https://arxiv.org/abs/1406.1078>
- <https://arxiv.org/abs/1609.08144>



# Natural Language Processing

## DSECL ZG565

Prof.Vijayalakshmi anand

BITS-Pilani

**BITS** Pilani  
Pilani Campus



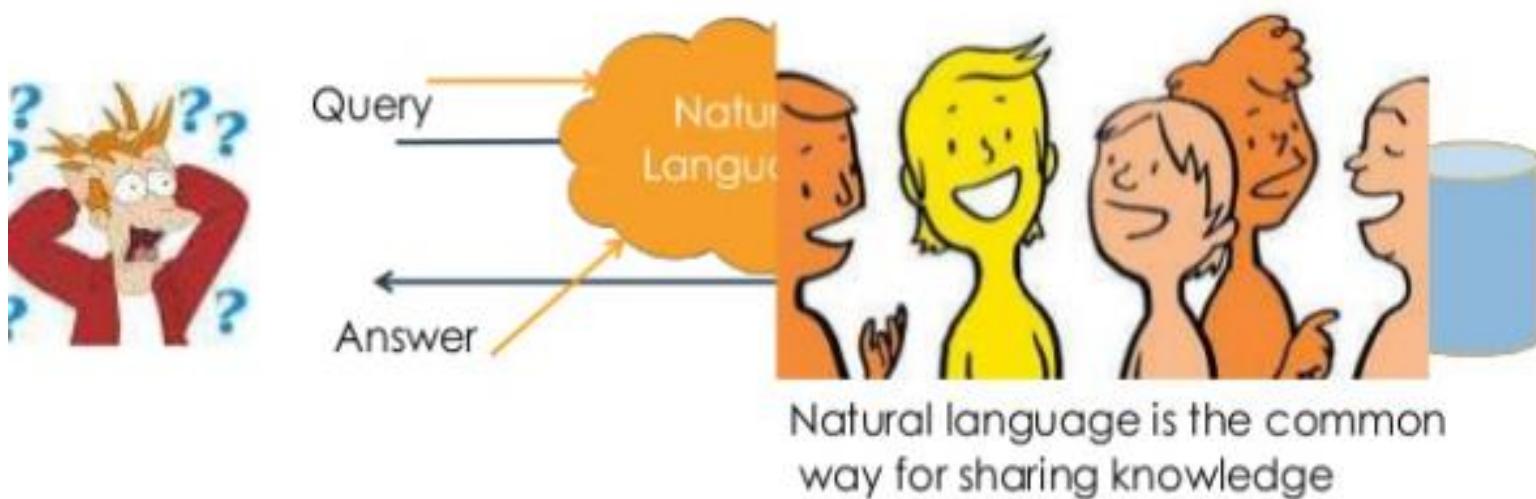
# Session Content

---

- Types of Questions
- IR-based Factoid Question Answering
- Text summarization
- Classification
  - Single document text summarization
  - Multi documents summarization
  - Neural model

# What is question answering system

Systems that automatically answer questions posed by humans in natural language query.



# Examples

---

- **AskJeeves** is probably most well known example
  - **AnswerBus** is an open-domain question answering system
  - **Ionaut, EasyAsk, AnswerLogic, AnswerFriend, Start, LCC, Quasm, Mulder, Webclopedia, etc.**
-

# Why QA system?



- Point to point answer
- Search engines do not speak your language.
- Difference between QA system and Search engine?

# Types of Questions in Modern Systems



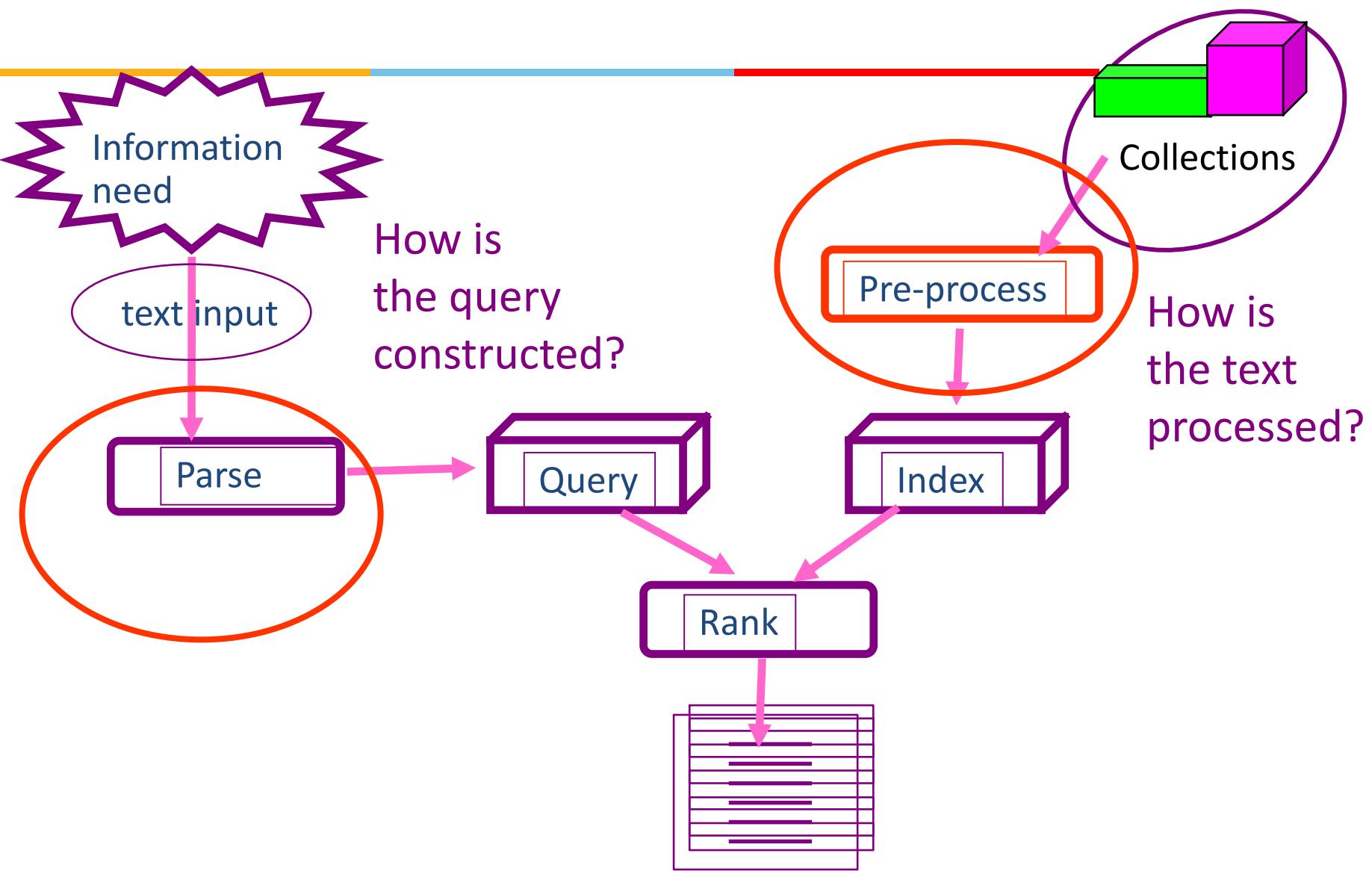
- Factoid questions
  - *Who wrote “The Universal Declaration of Human Rights”?*
  - *How many calories are there in two slices of apple pie?*
  - *What is the average age of the onset of autism?*
  - *Where is Apple Computer based?*
- Complex (narrative) questions:
  - *In children with an acute febrile illness, what is the efficacy of acetaminophen in reducing fever?*
  - *What do scholars think about Jefferson’s position on dealing with pirates?*

# Paradigms for QA

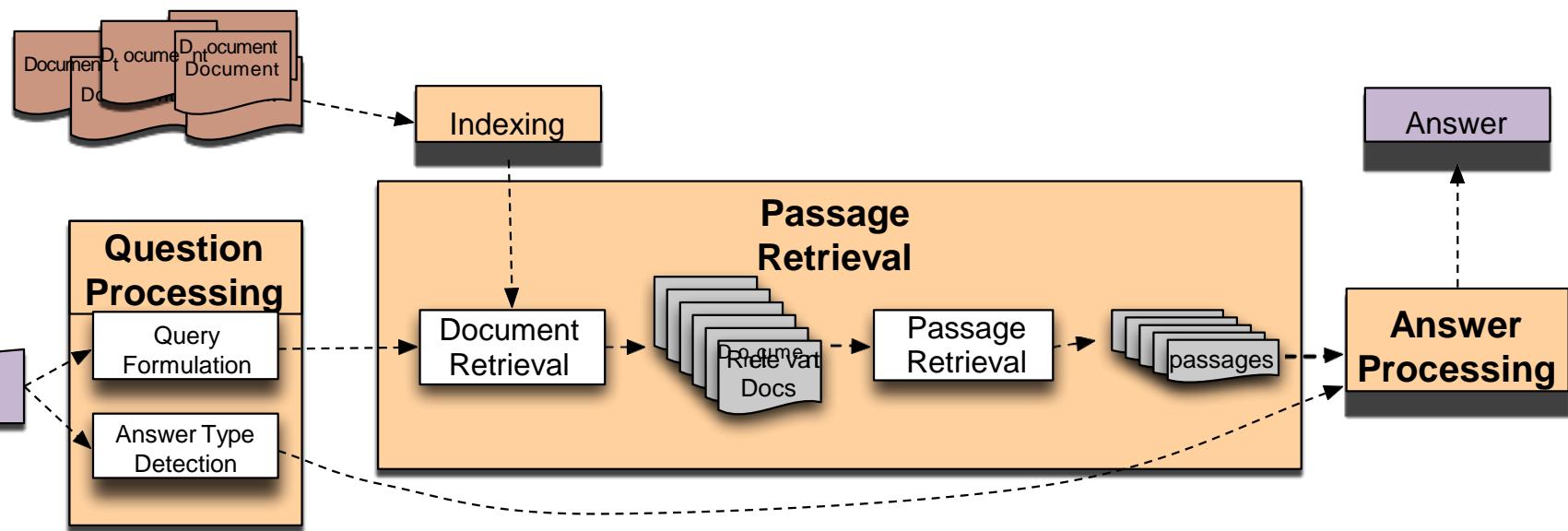


- IR-based approaches
  - TREC; IBM Watson; Google
- Knowledge-based and Hybrid approaches
  - IBM Watson; Apple Siri; Wolfram Alpha; True Knowledge Evi

# Information Retrieval Process



# Factoid Q/A



# Question Processing

## Things to extract from the question

---

- Answer Type Detection
  - Decide the **named entity type** (person, place) of the answer
- Query Formulation
  - Choose **query keywords** for the IR system
- Question Type classification
  - Is this a definition question, a math question, a list question?

# Passage Retrieval

---

- Step 1: IR engine retrieves documents using query terms
- Step 2: Segment the documents into shorter units
  - something like paragraphs
- Step 3: Passage ranking
  - Use answer type to help rerank passages

# Features for Passage Ranking

---

Either in rule-based classifiers or with supervised machine learning

- Number of Named Entities of the right type in passage
- Number of query words in passage
- Number of question N-grams also in passage
- Proximity of query keywords to each other in passage
- Longest sequence of question words
- Rank of the document containing passage

# Answer Extraction

- Run an answer-type named-entity tagger on the passages
- Return the string with the right type:
  - Who was the first prime minister of India (**PERSON**)

# Ranking Candidate Answers

---

- But what if there are multiple candidate answers!

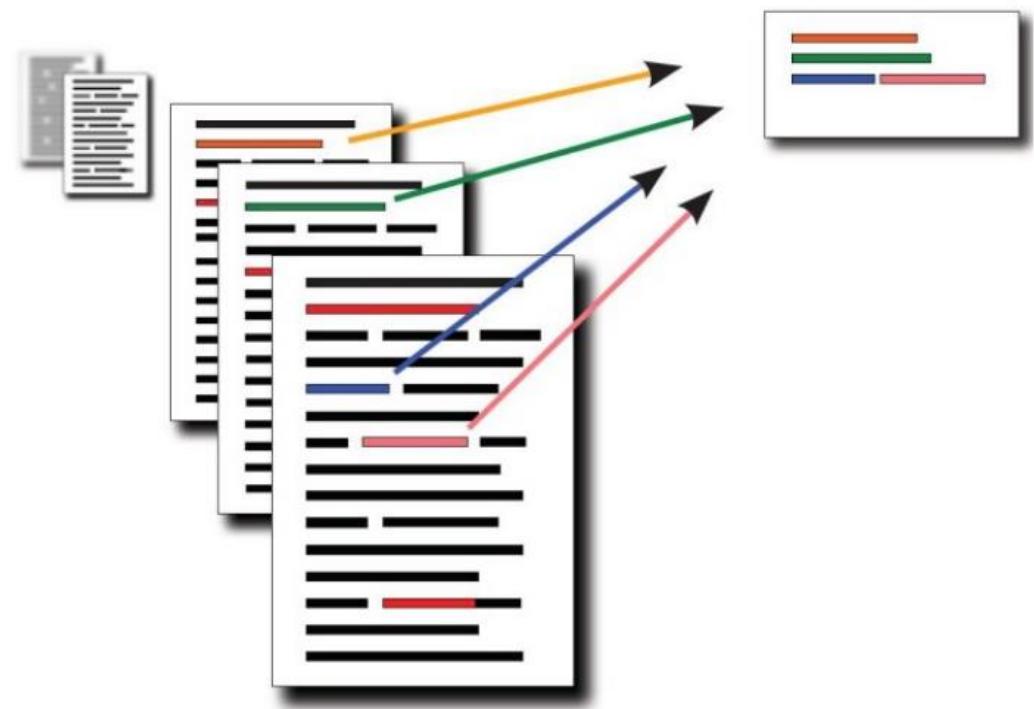
Q: Who was Queen Victoria's second son?

- Answer Type: **Person**
- Passage:

The Marie biscuit is named after Marie Alexandrovna, the daughter of Czar Alexander II of Russia and wife of Alfred, the second son of Queen Victoria and Prince Albert

# What is Text Summarization?

**Task:** produce an abridged version of a text while retaining the key, relevant information



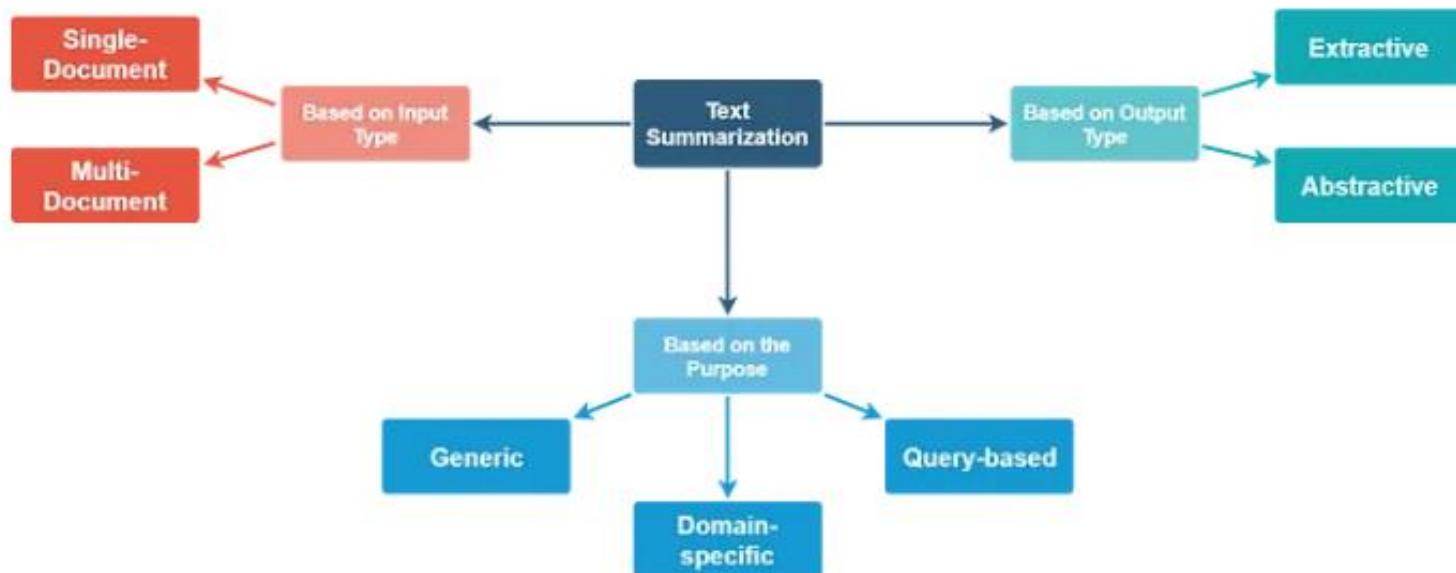
# Applications

---

Useful for creating

- outlines or abstracts of any document, article, etc
- summaries of chat and email
- action items from a meeting
- simplifying text by compressing sentences

## Type of summarization:



# Text Summarization

## Input:

- single document summarization (SDS)
- multiple-document summarization (MDS)

## Output:

- extractive
- abstractive

## Focus:

- generic (unconditioned)
- query-focused (conditioned)

## Approach:

- supervised
- unsupervised

# What to summarize the Input?

---

- **Single-document summarization**
  - Given a single document, produce
    - abstract
    - outline
    - headline
- **Multiple-document summarization**
  - Given a group of documents, produce a gist of the content:
    - a series of news stories on the same event
    - a set of web pages about some topic or question

# Type of Summarization

---

- Generic summarization:
  - Summarize the content of a document
- Query-focused summarization:
  - summarize a document with respect to an information need expressed in a user query.
  - a kind of complex question answering:
    - Answer a question by summarizing a document that has the information to construct the answer

# Summarization for Question Answering Snippets



- Create **snippets** summarizing a web page for a query
  - Google: 156 characters (about 26 words) plus title and link

The screenshot shows a Google search results page. At the top left is the Google logo. To its right is a search bar containing the query "what is die brücke?". Below the search bar, the word "Search" is displayed in red, followed by the text "About 5,910,000 results (0.28 seconds)". On the left side, there is a sidebar with links: "Everything", "Images", "Maps", "Videos", "News", "Shopping", "Applications", and "More". The main content area displays two snippets. The first snippet is for "Die Brücke - Wikipedia, the free encyclopedia" (en.wikipedia.org/wiki/Die\_Brücke). It includes a brief description: "Die Brücke (The Bridge) was a group of German expressionist artists formed in Dresden in 1905, after which the Brücke Museum in Berlin was named. Founding ... You've visited this page 5 times. Last visit: 4/16/12". The second snippet is for "Die Brücke (film) - Wikipedia, the free encyclopedia" (en.wikipedia.org/wiki/Die\_Brücke\_(film)). It includes a brief description: "Die Brücke (English: The Bridge) is a 1959 West German film directed by Austrian filmmaker Bernhard Wicki. It is based on the eponymous 1958 novel by ...". At the bottom of the page, there is a footer with the text "BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956".

# Summarization for Question Answering Multiple Documents

---

Create answers to complex questions summarizing multiple documents.

- Instead of giving a snippet for each document
- Create a cohesive answer that combines information from each document

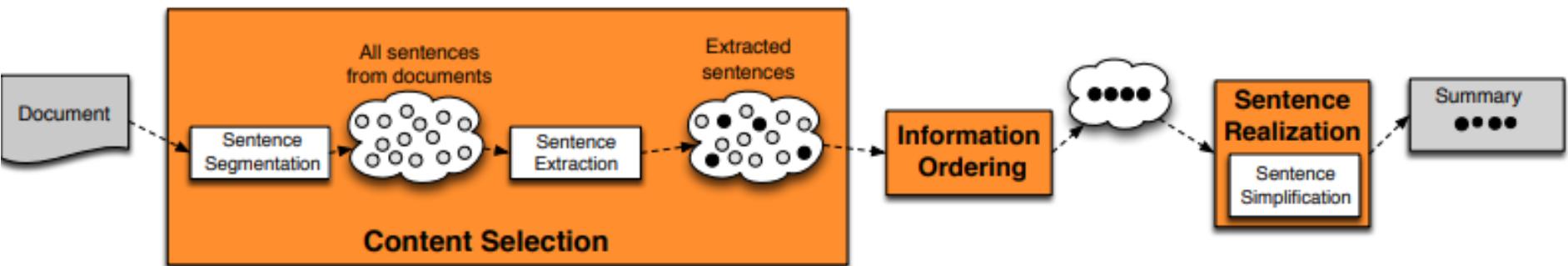
# Extractive summarization & Abstractive summarization

---

- Extractive summarization
  - create the summary from phrases or sentences in the source document(s)
- Abstractive summarization
  - express the ideas in the source documents using (at least in part) different words

# Summarization Three Stages

1. content selection: choose sentences to extract from the document
2. information ordering: choose an order to place them in the summary
3. sentence realization: clean up the sentence



# Unsupervised content selection

- Intuition dating back to Luhn (1958):
  - Choose sentences that have **salient** or **informative** words
- Two approaches to defining salient words
  1. **tf-idf**: weigh each word  $w_i$  in document  $j$  by tf-idf
$$weight(w_i) = tf_{ij} \times idf_i$$
  2. **topic signature**: choose a smaller set of salient words
    - mutual information
    - log-likelihood ratio (**LLR**) Dunning (1993), Lin and Hovy (2000)

$$weight(w_i) = \begin{cases} 1 & \text{if } -2\log \lambda(w_i) > 10 \\ 0 & \text{otherwise} \end{cases}$$

# Simple tf\*idf

---

$$w_{ik} = tf_{ik} * \log(N / n_k)$$

$T_k$  = term  $k$  in document  $D_i$

$tf_{ik}$  = frequency of term  $T_k$  in document  $D_i$

$idf_k$  = inverse document frequency of term  $T_k$  in  $C$

$N$  = total number of documents in the collection  $C$

$n_k$  = the number of documents in  $C$  that contain  $T_k$

$$idf_k = \log\left(\frac{N}{n_k}\right)$$

# Topic signature based content selection with queries



- choose words that are informative either
  - by log-likelihood ratio (LLR)
  - or by appearing in the query

$$weight(w_i) = \begin{cases} 1 & \text{if } -2\log \lambda(w_i) > 10 \\ 1 & \text{if } w_i \in question \\ 0 & \text{otherwise} \end{cases}$$

(could learn more complex weights)

- Weigh a sentence (or window) by weight of its words:

$$weight(s) = \frac{1}{|S|} \sum_{w \in S} weight(w)$$

# Log-Likelihood Ratio (LLR)

## Centroid Based Summarization

1. Tokenize the input sentence  $S_j$  into set of non-stop words :  $W_i$ 's
2. Compute LLR for every  $W_i$ 's
3. Find the weights of every  $W_i$ 's
4. Using weight( $W_i$ ) compute weight( $S$ ) for every  $S_j$ 's
5. Rank the  $S_j$ 's in descending order of weight or compare with threshold  $\theta$
6. Pick top "k" sentences for summary

$\lambda(w) = \log$  likelihood ratio for a word is given by

$$\lambda = \frac{b(k, N, p)}{b(k_I, N_I, p_I).b(k_B, N_B, p_B)}$$

$$weight(w_i) = \begin{cases} 1 & \text{if } -2\log \lambda(w_i) > 10 \\ 1 & \text{if } w_i \in \text{question} \\ 0 & \text{otherwise} \end{cases}$$

$$weight(s) = \frac{1}{|S|} \sum_{w \in S} weight(w) \quad weight(s_i) = \sum_{w \in s_i} \frac{weight(w)}{|\{w | w \in s_i\}|}$$

Water spinach (*ipomoea aquatica*) is a semiaquatic leafy green plant with long hollow stems and spear or heart shaped leaves. It is widely grown throughout Asia as a leaf vegetable. The leaves and stems are often eaten fried flavored with salt or in soups. Other common names include morning glory vegetable, kangkong (Malay). It is not related to spinach, but is closely related to sweet potato and convolvulus.

S1 : Water spinach ipomoea aquatica semiaquatic leafy green plant long hollow stems spear heart shaped leaves = 8/15 = 0.53

S2 : grown Asia leaf vegetable = 2/4 = 0.5

S3 : leaves stems eaten fried flavored salt soups = 2/7 = 0.28

S4 : common names morning glory vegetable kangkong Malay = 3/7 = 0.42

S5 : related spinach closely related sweet potato convolvulus = 4/7 = 0.57

Assume  $P_i(W | I) = 1/24$ ,  $P_b(W | B) = 50/10000000$

$P = (1+50)/(24+10000000)$

$L(W) = 0.000877 \rightarrow -2\log(L(w)) = 14.8$

Weight(vegetable) = 1

Weight(S2) =  $\frac{1}{4} * \text{sum\_of\_weights}\{\text{grown, asia, leaf, vegetable}\}$

Water spinach (*ipomoea aquatica*) is a semiaquatic leafy green plant with long hollow stems and spear or heart shaped leaves. It is not related to spinach, but is closely related to sweet potato and convolvulus.

# Supervised Content Selection

- Given:
  - a labeled training set of good summaries for each document
- Align:
  - the sentences in the document with sentences in the summary
- Extract features
  - position (first sentence?)
  - length of sentence
  - word informativeness, cue phrases
  - cohesion
- Train
  - a binary classifier (put sentence in summary? yes or no)
- Problems:
  - hard to get labeled training data
  - alignment difficult
  - performance not better than unsupervised algorithms
- So in practice:
  - **Unsupervised content selection is more common**

# Evaluating Summaries: ROUGE

---



ROUGE (Recall Oriented Understudy for Gisting Evaluation)

- Intrinsic metric for atomically evaluating summaries
- Based on BLEU (a metric used for machine translation)
- Not as good as human evaluation (“Did this answer the user’s question?”)
- But much more convenient

# ROUGE-2

Given a document D, and an automatic summary X:

1. Have N humans produce a set of reference summaries of D
2. Run system, giving automatic summary X
3. What percentage of the bigrams from the reference summaries appear in X?

$$ROUGE - 2 = \frac{\sum_{s \in \{\text{RefSummaries}\}} \sum_{\text{bigrams } i \in s} \min(\text{count}(i, X), \text{count}(i, S))}{\sum_{s \in \{\text{RefSummaries}\}} \sum_{\text{bigrams } i \in s} \text{count}(i, S)}$$

# ROUGE-2 Example

---

Q: “What is water spinach?”

Human 1: Water spinach is a green leafy vegetable grown in the tropics.

Human 2: Water spinach is a semi-aquatic tropical plant grown as a vegetable.

Human 3: Water spinach is a commonly eaten leaf vegetable of Asia.

- System answer: Water spinach is a leaf vegetable commonly eaten in tropical areas of Asia.

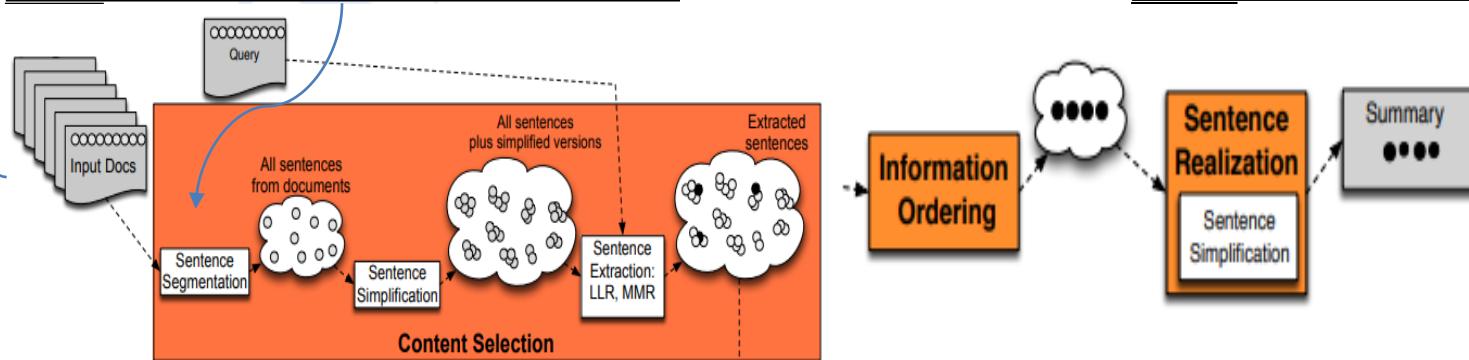
$$\text{Rouge-2 score} = \frac{3 + 3 + 6}{10 + 9 + 9} = 12/28 = .43$$

---

# Multi-Document Summarization

## Three Stages

Pattern	Question	Answer
<AP> such as <QP>	What is autism?	”, developmental disorders such as autism”
<QP> (an <AP>)	What is a caldera?	”the Long Valley caldera, a <u>volcanic crater</u> 19 miles long”



- 1 [The Justice Department]<sub>S</sub> is conducting an [anti-trust trial]<sub>O</sub> against [Microsoft Corp.]<sub>X</sub>
- 2 [Microsoft]<sub>O</sub> is accused of trying to forcefully buy into [markets]<sub>X</sub> where [its own products]<sub>S</sub> are not competitive enough to unseat [established brands]<sub>O</sub>
- 3 [The case]<sub>S</sub> revolves around [evidence]<sub>O</sub> of [Microsoft]<sub>S</sub> aggressively pressuring [Netscape]<sub>O</sub> into merging [browser software]<sub>O</sub>
- 4 [Microsoft]<sub>S</sub> claims [its tactics]<sub>S</sub> are commonplace and good economically.

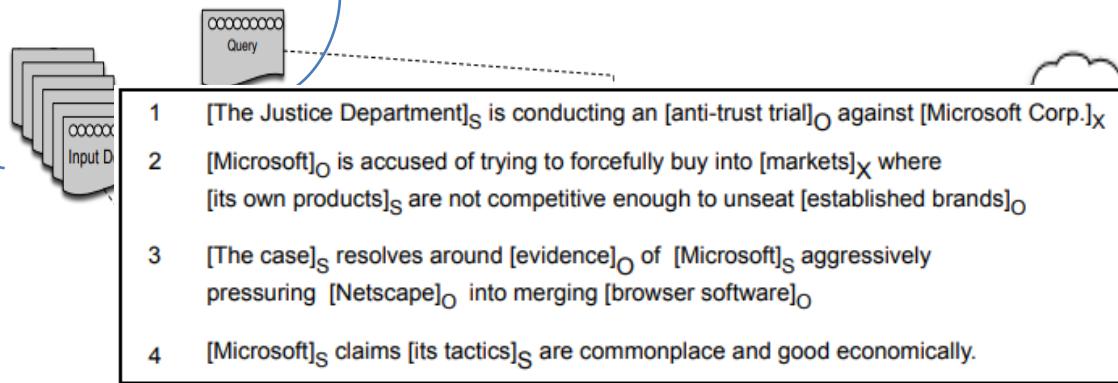
Department	Trial	Microsoft	Markets	Products	Brands	Case	Netscape	Software	Tactics
1	S	O	X	-	-	-	-	-	-
2	-	-	O	X	S	O	-	-	-
3	-	-	S	O	-	-	S	O	O
4	-	-	S	-	-	-	-	-	O

# Multi-Document Summarization



## Three Stages

Pattern	Question	Answer
<AP> such as <QP>	What is autism?	”, developmental disorders such as autism”
<QP> (an <AP>)	What is a caldera?	”the Long Valley caldera, a <u>volcanic crater</u> 19 miles long”



Sentences : S  
 : <Extract Worthy , Not Extract Worthy>

## ML : K – Means or Hierarchical Cluster

OR

Relevance Score + Saliency Score/Topic Signature

### Hyper-parameters:

- Length
- Ndocs
- $\lambda$  - Relevancy vs Redundancy

### Order

Chronology : **Sd1-6**, **Sd2-1** , **Sd4-2**

Lexical Coherence : **Sd2-1** **Sd1-6** **Sd4-2**

Entity based Coherence : **Sd1-6** **Sd4-2** **Sd2-1**

### Original summary:

Presidential advisers do not blame **O'Neill**, but they've long recognized that a shakeup of the economic team would help indicate **Bush** was doing everything he could to improve matters. U.S. President **George W. Bush** pushed out Treasury Secretary **Paul O'Neill** and top economic adviser Lawrence Lindsey on Friday, launching the first shake-up of his administration to tackle the ailing economy before the 2004 election campaign.

### Rewritten summary:

Presidential advisers do not blame **Treasury Secretary Paul O'Neill**, but they've long recognized that a shakeup of the economic team would help indicate U.S. President **George W. Bush** was doing everything he could to improve matters. **Bush** pushed out **O'Neill** and White House economic adviser Lawrence Lindsey on Friday, launching the first shake-up of his administration to tackle the ailing economy before the 2004 election campaign.

### Co-reference Resolution Algorithm- Rules

<S1, S3, S5, S8>

S1 : Clean Up

S3 : Clean Up

.....Rules based

### Sentence Compression/Simplification

Standard tf-idf cosine distance between each pair of sentences and choosing the overall ordering that minimizes the average distance between neighboring sentences

# Content Selection

## Relevance Score

$$MMR = \arg \max_{D_i \in R \setminus S} [\lambda \text{Sim}_1(D_i, Q) - (1 - \lambda) \max_{D_j \in S} \text{Sim}_2(D_i, D_j)]$$

C = document collection

$D_i$  = Documents in the collection C

Q = query or user profile

R = the ranked list of documents retrieved by an IR system,

S = subset of documents in R already selected

$R \setminus S$  = set difference, i.e., the set of unselected documents in R

$\text{Sim}_1$  can be the same as  $\text{Sim}_2$  or a different metric

### Hyper-parameters:

- Length = 3
- Ndocs = 5
- $\lambda$  = 0.5

### 3 RD ITERATION

$$S = \{d_1, d_2\}$$

$$\text{Max} \{\text{sim}(d_i, d_1), \text{sim}(d_i, d_2)\}$$

For  $d_3$ :

$$\text{max}\{\text{sim}(d_1, d_3), \text{sim}(d_2, d_3)\}$$

$$= \text{max}\{0.23, 0.29\} = 0.29$$

$$\text{sim}(d_3, q) = 0.50$$

$$MMR(d_3) = \lambda * 0.5 - (1 - \lambda) * 0.29 = 0.105$$

A document has high marginal relevance if it is both relevant to the query and contains minimal similarity to previously selected documents.

MMR → query-relevance + information-novelty



	innovate	achieve	lead			
$d_1$	1	0.11	0.23	0.76	0.25	0.91
$d_2$	1	0.29	0.57	0.51	0.90	
$d_3$		1	0.02	0.20	0.50	
$d_4$			1	0.33	0.06	
$d_5$				1	0.63	
q					1	

1<sup>ST</sup> ITERATION

$$S = \{\}$$

$$\begin{aligned} MMR &= \arg \max (\text{Sim}(d_i, q)) \\ &= \text{sim}(d_1, q) \\ &= 0.91 \end{aligned}$$

$$S = \{d_1\}$$

2 ND ITERATION

$$\begin{aligned} \text{For } d_2: \text{sim}(d_1, d_2) &= 0.11 \\ \text{sim}(d_2, q) &= 0.90 \\ MMR(d_2) &= \lambda(0.90) - (1 - \lambda)0.11 \\ &= 0.395 \end{aligned}$$

$$\begin{aligned} \text{MMR values for } d_3 &= 0.135 \\ d_4 &= -0.35 \\ d_5 &= 0.19 \end{aligned}$$

$$S = \{d_1, d_2\}$$

$$= \{d_1, d_2, d_3\}$$

# LLR+MMR: Choosing informative yet non redundant sentences

---

One of many ways to combine the intuitions of LLR and MMR:

1. Score each sentence based on LLR (including query words)
2. Include the sentence with highest score in the summary.
3. Iteratively add into the summary high scoring sentences that are not redundant with summary so far

# Information Ordering

---

- Chronological ordering:
  - Order sentences by the date of the document (for summarizing news) (Barzilay, Elhadad, and McKeown 2002)
- Coherence:
  - Choose orderings that make neighboring sentences similar (by cosine).
  - Choose orderings in which neighboring sentences discuss the same entity (Barzilay and Lapata 2007)
- Topical ordering
  - Learn the ordering of topics in the source documents

# Domain specific answering: Information Extraction method

---

- a good biography of a person contains:
  - a person's birth/death, fame factor, education, nationality and so on
- a good definition contains:
  - genus or hypernym
  - Hajj is a type of ritual
- a medical answer about a drug's use contains:
  - • the problem (the medical condition),
  - • the intervention (the drug or procedure), and
  - • the outcome (the result of the study).

# Information that should be in the answer for 3 kinds of questions

Definition	
<b>genus</b>	The Hajj is a type of ritual
<b>species</b>	the annual hajj begins in the twelfth month of the Islamic year
<b>synonym</b>	The Hajj, or Pilgrimage to Mecca, is the central duty of Islam
<b>subtype</b>	Qiran, Tamattu', and Ifrad are three different types of Hajj
Biography	
<b>dates</b>	was assassinated on April 4, 1968
<b>nationality</b>	was born in Atlanta, Georgia
<b>education</b>	entered Boston University as a doctoral student
Drug efficacy	
<b>population</b>	37 otherwise healthy children aged 2 to 12 years
<b>problem</b>	acute, intercurrent, febrile illness
<b>intervention</b>	acetaminophen (10 mg/kg)
<b>outcome</b>	ibuprofen provided greater temperature decrement and longer duration of antipyresis than acetaminophen when the two drugs were administered in approximately equal doses

# Architecture for complex question answering:

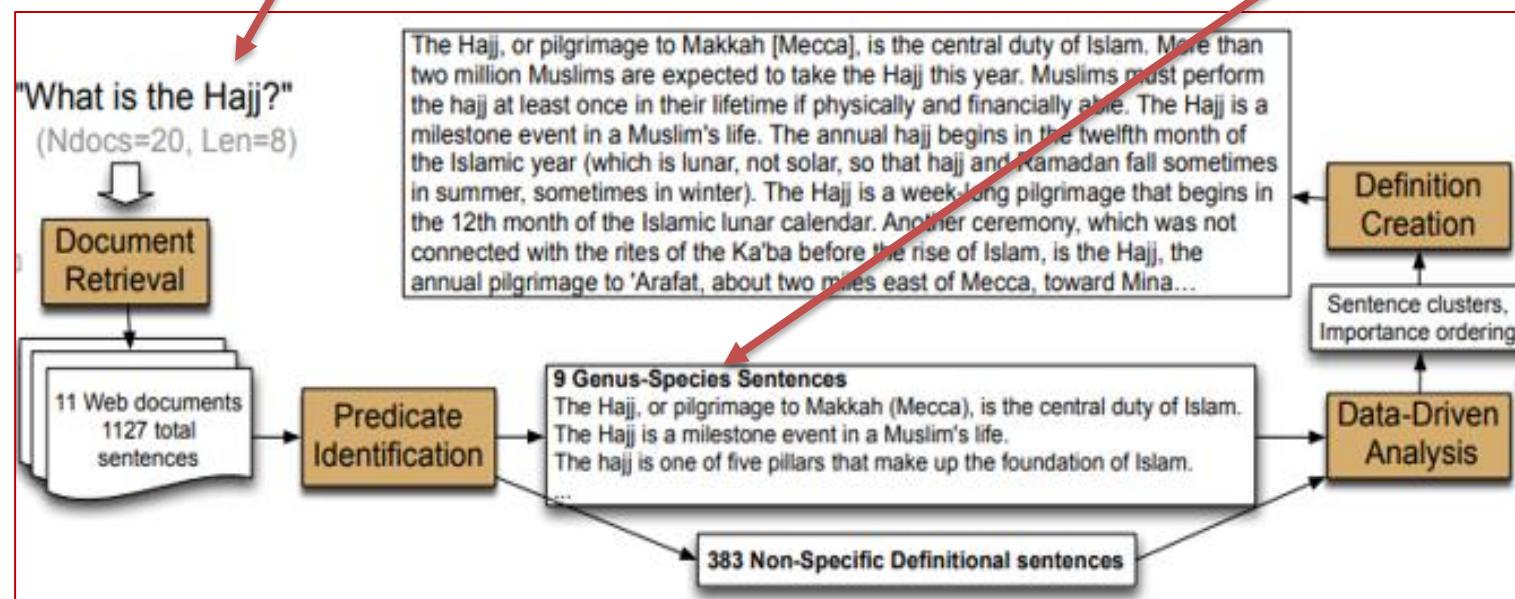
## Definition questions

T = definition question , N = no of documents to retrieve, L = length of the answer (in sentences)

handwritten set of patterns to extract the term (*Hajj*) to be defined from the query *T*

apply classifiers to label each sentence with a classe from the domain. For definition questions, there are four classes: **genus, species, other definitional, other**.

a set of relevant sentences is selected



## Overview : Neural based approach

---

### **span labeling :**

Answer is a span of text in the passage

### **Given:**

Q: "how tall is mt. Everest?"

Passage that contains the clause : ".....reaching 29,029 feet at its summit, a reader will output 29,029 feet...."

### **Output:**

Span labeling: identifying span in the passage a span (a continuous string of text) that constitutes an answer

## Span labeling

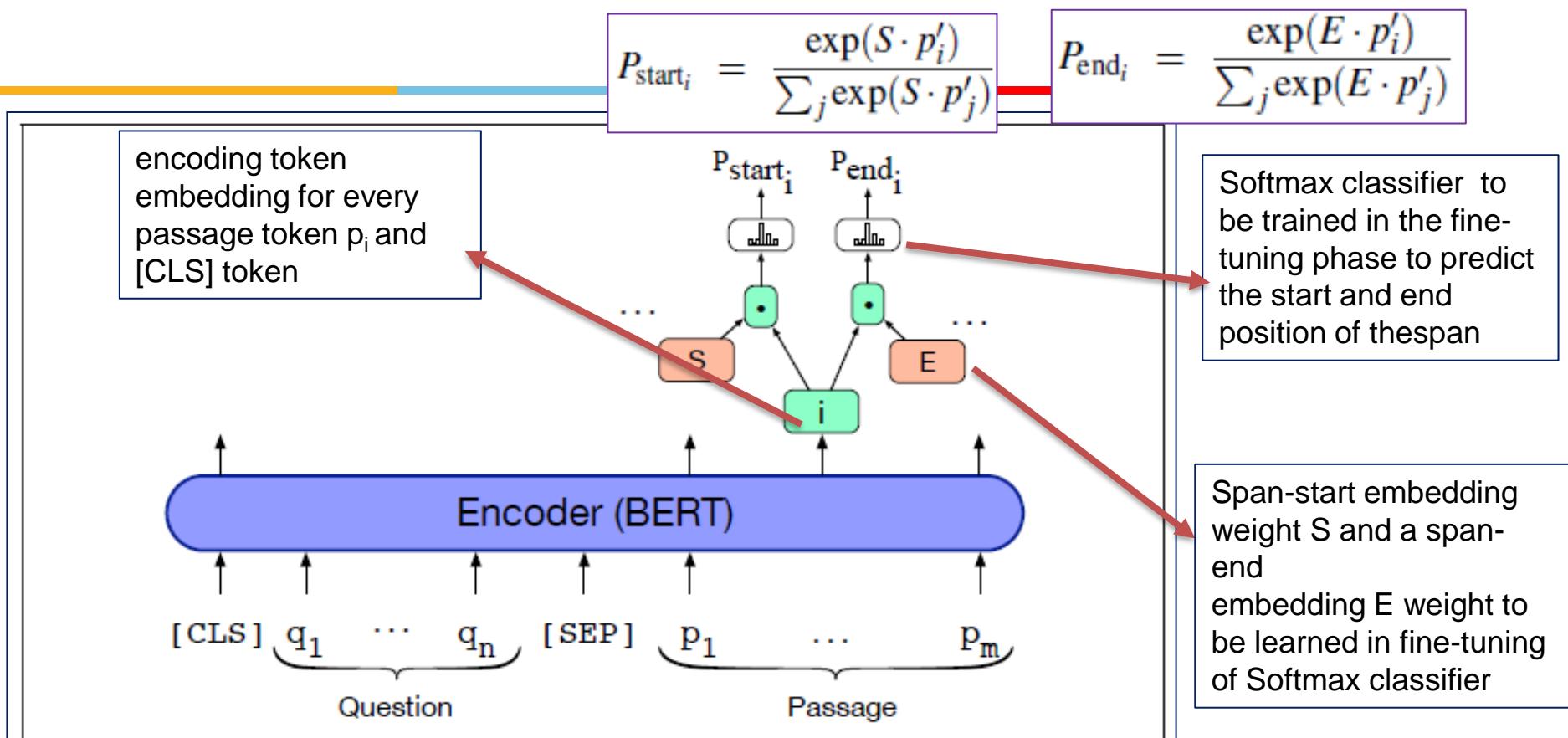
- **Input :** n question tokens  $q_1, \dots, q_n$   
m passage tokens  $p_1, \dots, p_m$
- **Output:**  $P(a|q, p) = \text{Probability of span } a \text{ given question } q \text{ and passage } p$

$a_s$ = span start and  $a_e$ = span end then

$$P(a|q, p) = P_{\text{start}}(a_s|q, p) P_{\text{end}}(a_e|q, p)$$

- Thus for each token  $p_i$  in the passage two probabilities are computed :  
 $p_{\text{start}}(i)$  and  $p_{\text{end}}(i)$

# Extractive QA : BERT



**Figure 14.12** An encoder model (using BERT) for span-based question answering from reading-comprehension-based question answering tasks.

Score of a candidate span from position  $i$  to  $j$  =  $S \cdot P_i + E \cdot P_j$

Highest scoring span in which  $j \geq i$  is chosen is the model prediction

# Approaches Summary

## Generation Way

- gen-ext : Extractive Summarization
- gen-abs : Abstractive Summarization
- gen-2stage : Two-stage Summarization (compressive, hybrid)

## Regressive Way

- regr-auto : Autoregressive Decoder (Pointer network)
- regr-nonauto : Non-autoregressive Decoder (Sequence labeling)

## Supervision

- sup-sup : Supervised Learning
- sup-weak (implies sup-sup) : Weakly Supervised Learning
- sup-unsup : Unsupervised Learning

## Task Settings

rich of task settings!

- task-single : Single-document Summarization
- task-multi : Multi-document Summarization
- task-senCompre : Sentence Compression
- task-sci : Scientific Paper
- task-multimodal : Multi-modal Summarization
- task-aspect : Aspect-based Summarization
- task-opinion : Opinion Summarization
- task-questoin : Question-based Summarization

## Architecture (Mechanism)

- arch-rnn : Recurrent Neural Networks (LSTM, GRU)
- arch-cnn : Convolutional Neural Networks (CNN)
- arch-transformer : Transformer
- arch-graph : Graph Neural Networks or Statistic Graph Models
- arch-gnn : Graph Neural Networks
- arch-att : Attention Mechanism
- arch-pointer : Pointer Layer
- arch-coverage : Coverage Mechanism

## Training

- train-multitask : Multi-task Learning
- train-multilingual : Multi-lingual Learning
- train-multimodal : Multi-modal Learning
- train-auxiliary : Joint Training
- train-transfer : Cross-domain Learning, Transfer Learning, Domain Adaptation
- train-active : Active Learning, Bootstrapping
- train-adver : Adversarial Learning
- train-template : Template-based Summarization
- train-augment : Data Augmentation
- train-curriculum : Curriculum Learning
- train-lowresource : Low-resource Summarization
- train-retrieval : Retrieval-based Summarization
- train-meta : Meta-learning

## Pre-trained Models

- pre-word2vec : word2vec
- pre-glove : GLoVe
- pre-bert : BERT



# References

---

- Speech and Language processing An introduction to Natural Language Processing, Computational Linguistics and speech Recognition by Daniel Jurafsky and James H. Martin[3rd edition] Chapter 21
- <https://www.youtube.com/watch?v=9PoKellNrBc>
- [https://www.youtube.com/watch?v=x9h5vJpkV\\_8](https://www.youtube.com/watch?v=x9h5vJpkV_8)
- <http://www.infocobuild.com/education/audio-video-courses/computer-science/NaturalLanguageProcessing-IIT-Kharagpur/lecture-52.html>
- [https://harvard-iacs.github.io/CS287/lectures/14\\_Summarization.pdf](https://harvard-iacs.github.io/CS287/lectures/14_Summarization.pdf)
- <http://demo.clab.cs.cmu.edu/algo4nlp19/slides/summarization.pdf>
- [https://people.engr.tamu.edu/huangrh/Fall16/I22\\_text\\_summarization.pdf](https://people.engr.tamu.edu/huangrh/Fall16/I22_text_summarization.pdf)



**BITS** Pilani  
Pilani | Dubai | Goa | Hyderabad

# Neural Language Model

**Dr. S. Prabakeran**

---

Teaching Assistant  
Birla Institute of Technology & Science, Pilani

# Agenda

- Neural Language Models
- How Large Language Models Work
- Feedforward Neural Network Basics
- Transformers for beginners - What are they and how do they work?
- Inside the Transformer Architecture
- How a Transformer works in machine translation
- Summary
- Demo Session

# Neural Language Models

- Introduction to Neural Language Models
- Applications of Neural Language Models
- Input Representations in Neural Language Models
- Structure of a Simple Neural Language Model
- Pretraining and Learning Word Embeddings
- Embedding Matrix and Model Optimization
- Visualization of Neural Language Model with Embedding Layer

# How Large Language Models Work

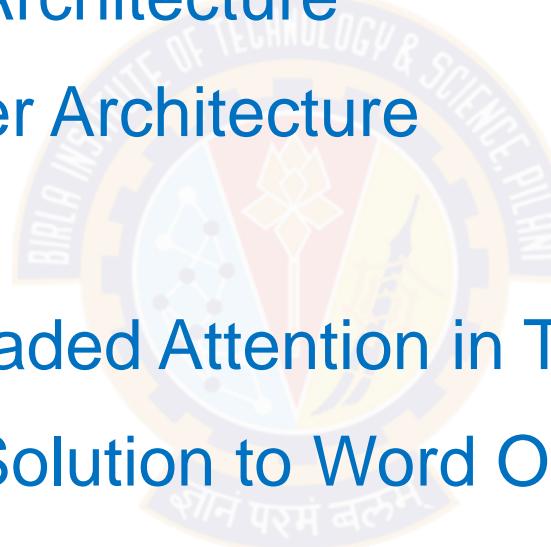
- Introduction to GPT
- Large Language Models and Foundation Models
- Scale of Text Data and Model Parameters
- Components of Large Language Models
- Training Process of Large Language Models
- Fine-Tuning for Specific Tasks
- Business Applications of Large Language Models

# Feedforward Neural Network Basics

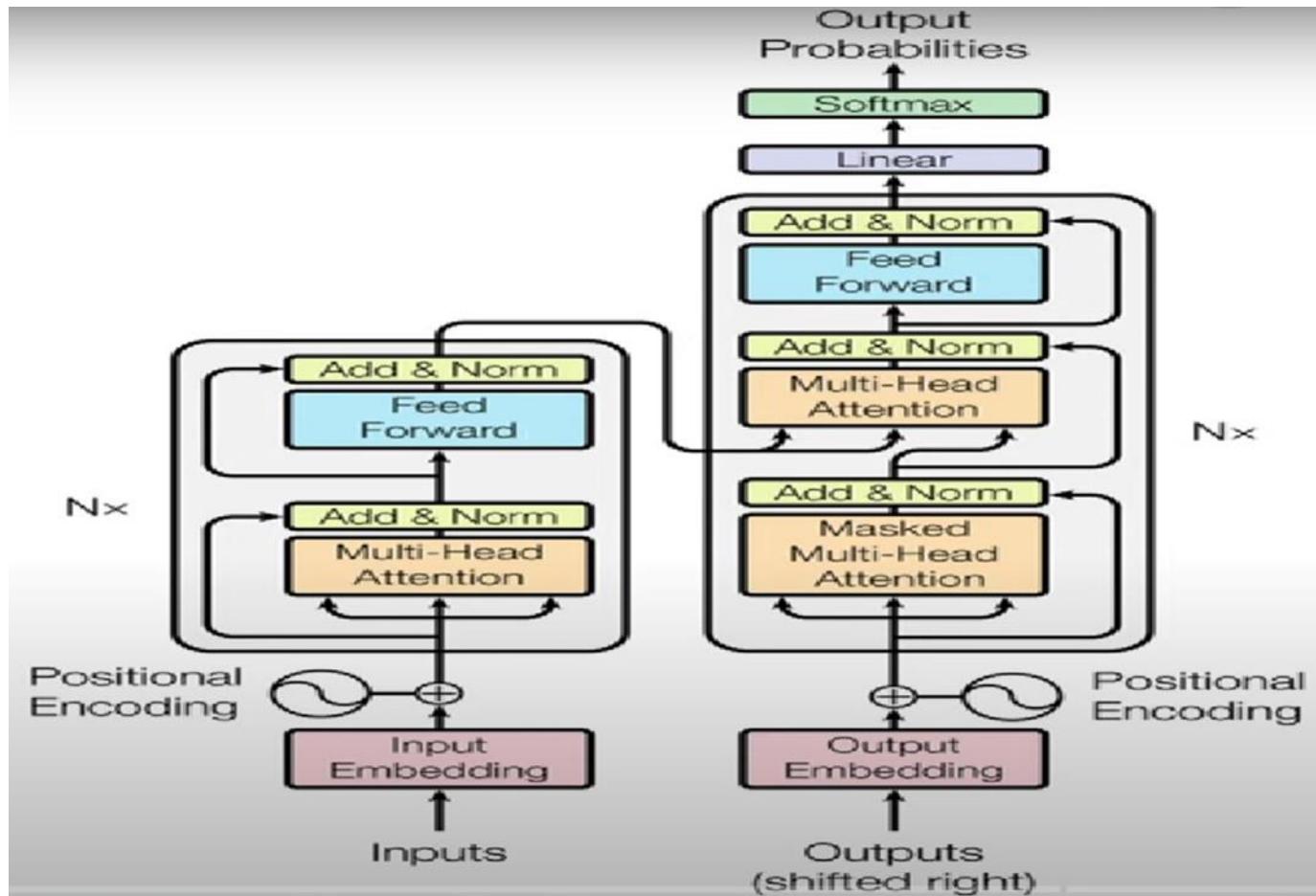
- Intuition Behind Feedforward Neural Network Design
- Structure of Feedforward Neural Networks
- General Flow of a Feedforward Neural Network
- Prediction Process in Feedforward Neural Networks
- Relationship Between Neural Networks and Deep Learning
- Differentiating Neural Networks in Depth
- Power of Neural Networks Over Traditional Models
- Feature Usage in Neural Networks

# Transformers for beginners - What are they and how do they work ?

- The Rise of Transformers in NLP
- Inside the Transformer Architecture
- Key Ideas in Transformer Architecture
- Multi-Headed Attention
- Understanding Multi-Headed Attention in Transformers
- Positional Encoding: A Solution to Word Order Ambiguity
- Bringing It All Together: How Transformers Operate
- Final Thoughts on Transformers: Simplicity in Complexity



# Inside the Transformer Architecture



# Inside the Transformer Architecture...

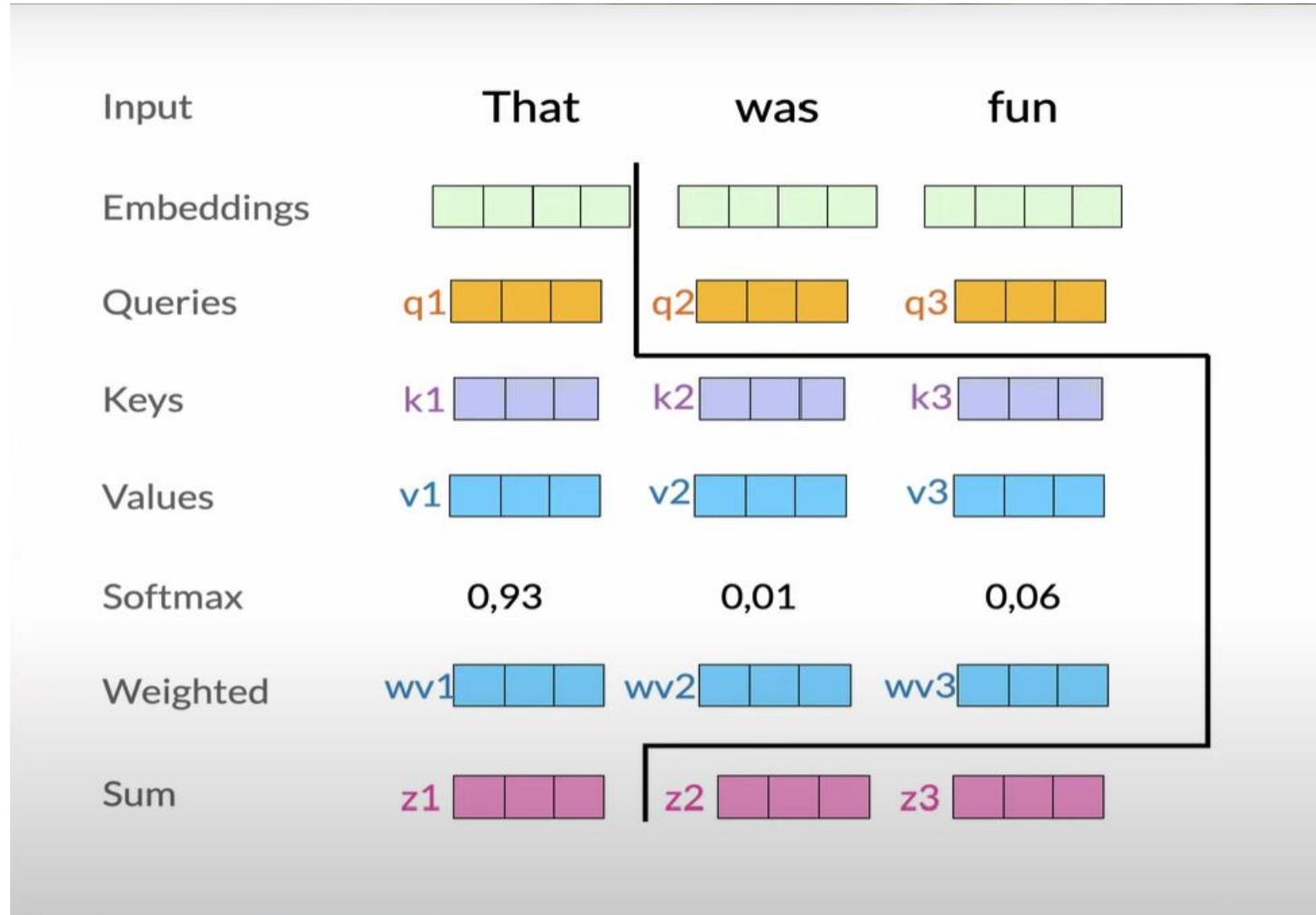
- Input Embedding
- Positional Encoding
- Multi-Head Attention: Query, Key-Value, Attention, Weighted Sum
- Add & Norm
- Feed Forward
- Output Embedding
- SoftMax



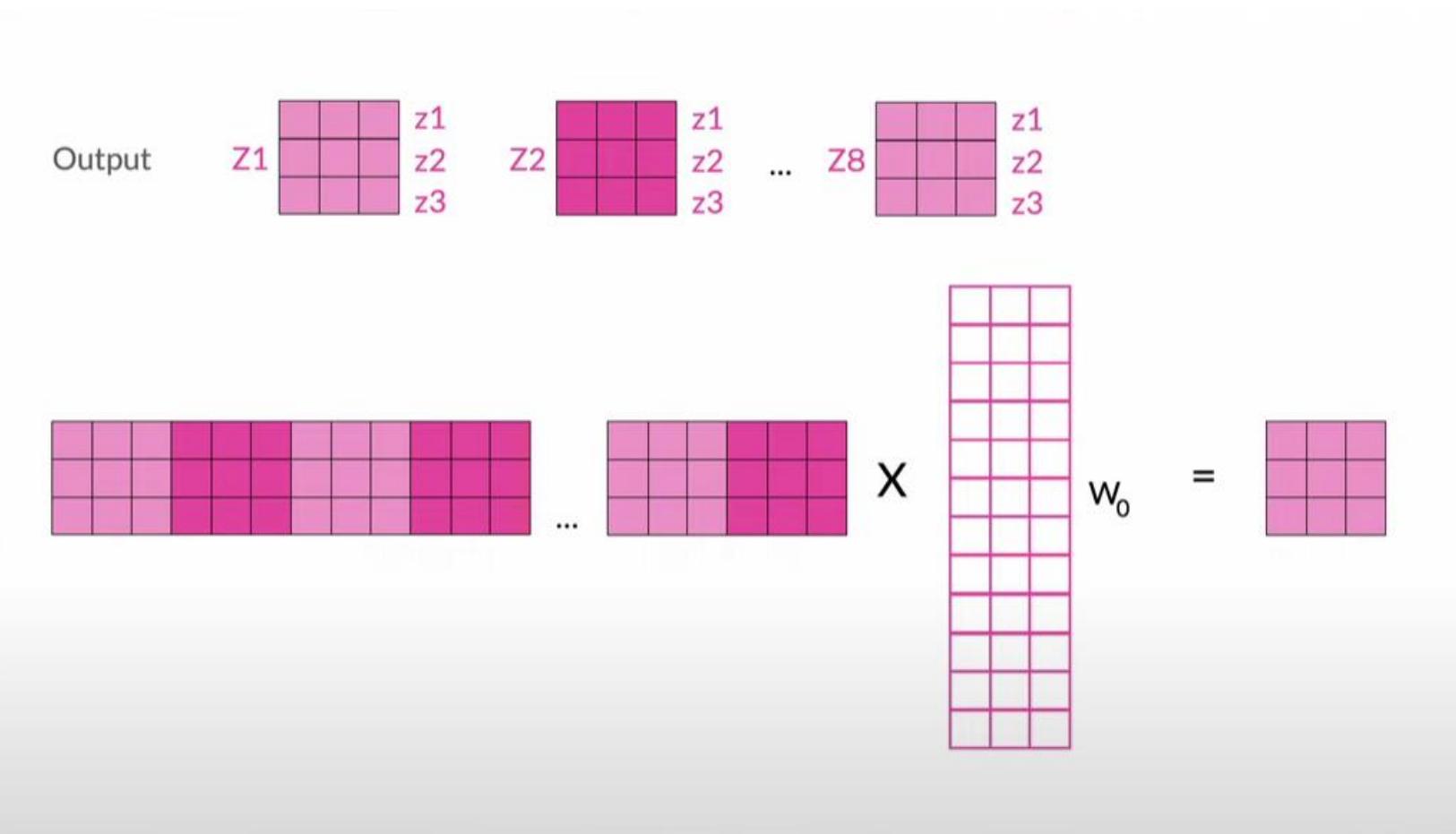
# Multi-Head Attention: Query, Key-Value, Attention, Weighted Sum...

Input	That	was	fun
Embeddings			
Queries			
Keys			
Values			
Score	$q1 \bullet k1 = 98$	$q1 \bullet k2 = 46$	$q1 \bullet k3 = 76$
Divide by 8	12,25	5,75	9,5
Softmax	0,93	0,01	0,06

# Multi-Head Attention: Query, Key-Value, Attention, Weighted Sum...



# Multi-Head Attention: Query, Key-Value, Attention, Weighted Sum...



# How a Transformer works in machine translation ?

Translate the English sentence "The cat sat on the mat" into French.

## Input Embedding:

- Each word in the English sentence is converted into a numerical representation (embedding).
- "The" becomes a vector of numbers, "cat" becomes a different vector, and so on.

## Positional Encoding:

- Information about the word order is added to the embeddings.
- This helps the model understand that "cat" comes before "mat" in the sentence.

## Multi-Head Attention:

- The model focuses on the relationships between different words in the sentence.
- It learns to pay more attention to words that are closely related, like "cat" and "mat."

# How a Transformer works in machine translation ?...

Translate the English sentence "The cat sat on the mat" into French.

## Feed Forward:

- The model applies additional processing to the information it has gathered.
- This helps it make more complex decisions about how to translate the sentence.

## Output:

- The model generates the French translation, "Le chat s'est assis sur le tapis."

# **DEMO Session on Machine Translation**



# **DEMO Session on Text Classification using Neural Networks**



# Thank you

