

# Natural Language Processing

## DSECL ZG565



**BITS** Pilani  
Pilani Campus

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



## **Session 1**

### **Date – 20<sup>th</sup> May 2023**

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 Content

---

- Objective of course
  - What will we learn in this course?
  - Text books and Reference books
  - Evaluation Plan
  - What is Natural Language Processing?
  - Application areas of Natural Language Processing
  - Introduction to Natural Language Processing
-



# Objective of course

---

- To learn the fundamental concepts and techniques of natural language processing (NLP) including Language Models, Word Embedding, Part pf 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

# **What you will learn in this course**

- **Language Modelling**
  - N-gram language modeling
  - Neural Language Models
- **Part-of-Speech Tagging**
- **Hidden Markov Model(MM), Maximum Entropy MM**
- **Topic Modelling**
- **Vector Semantics**
- **Parsing**
- **Encoder-Decoder Models, Attention and Contextual Embedding's**
- **Word sense disambiguation**
- **Semantic web ontology and knowledge Graphs**

# 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 |

# Evaluation Plan

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

# 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.
-

# Why Study NLP?

---

- A hallmark of human intelligence.
- Text is the largest repository of human knowledge and is growing quickly.
  - emails, news articles, web pages, IM, scientific articles, insurance claims, customer complaint letters, transcripts of phone calls, technical documents, government documents, patent portfolios, court decisions, contracts, .....

**The Natural Language Processing (NLP) Market size to grow from USD 15.7 billion in 2022 to USD 49.4 billion by 2027, at a Compound Annual Growth Rate (CAGR) of 25.7% during the forecast period.**

---

# Why are language technologies needed?

---

- Many companies make a lot of money if they could use computer programmes that understood text or speech.
    - answering the phone, and replying to a question
    - understanding the text on a Web page to decide who it might be of interest to
    - translating a daily newspaper from Japanese to English
    - understanding text in journals / books and building an expert systems based on that understanding
-

# Dreams or reality??

---

- Will my computer talk to me like another human ??
  - Will the search engine get me exactly what I am looking for??
  - Can my PC read the whole newspaper and tell me the important news only..??
  - Can my palmtop translate what that Japanese lady is telling me.. ??
  - Ahhh.. Can my PC do my NLP assignments ??
  - Do you know how our brain processes language ??
-

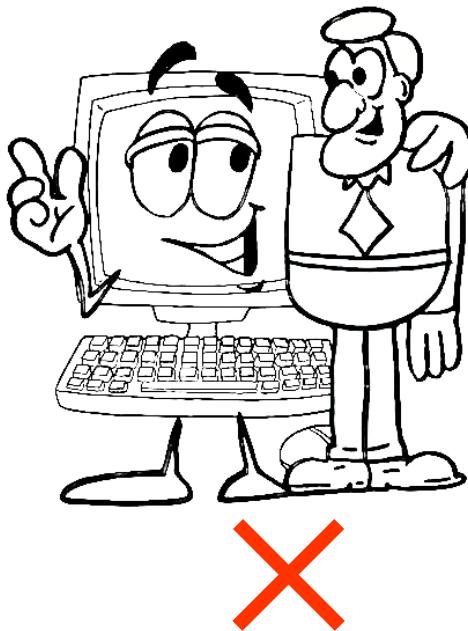
# The Dream

- It'd be great if machines could
  - Process our email (usefully)
  - Translate languages accurately
  - Help us manage, summarize, and aggregate information
  - Talk to us / listen to us
- But they can't:
  - Language is complex, ambiguous, flexible, and subtle
  - Good solutions need linguistics and machine learning knowledge



# The mystery

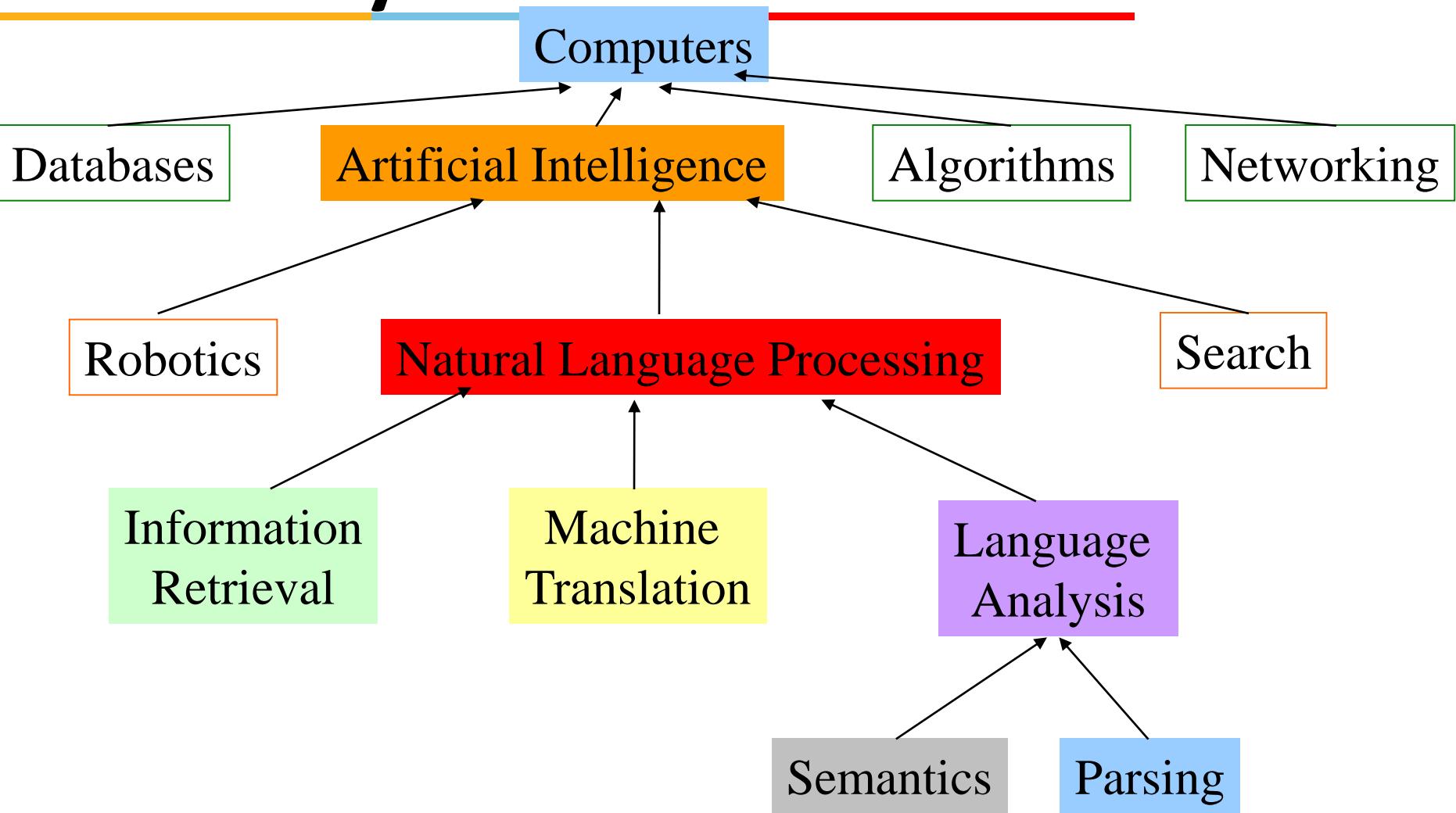
- What's now impossible for computers (and any other species) to do is effortless for humans



# Dreams??



# Where does it fit in the CS taxonomy?



# Brief history of NLP

---

- 1966: Eliza
- 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

# Introduction to Natural Language Processing

---

- The Study of Language.
- Applications of Natural Language Understanding.
- Evaluating Language Understanding Systems.
- The Different Levels of Language Analysis.
- Representations and Understanding.
- The Organization of Natural Language Processing Systems.

---

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

---

---

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**

---

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

# NLP Applications

---

- Question answering
    - Who is the first Taiwanese president?
  - Text Categorization/Routing
    - e.g., customer e-mails.
  - Text Mining
    - Find everything that can be done with NLP
  - Machine (Assisted) Translation
  - Language Teaching/Learning
    - Usage checking
  - Spelling correction
    - Is that just dictionary lookup?
-

# Application areas

---

- Text-to-Speech & Speech recognition
  - Healthcare
  - Natural Language Dialogue Interfaces to Databases
  - Information Retrieval\_
  - Information Extraction (<http://nlp.stanford.edu:8080/ner/process>)
  - Document Classification
  - Document Image Analysis
  - Automatic Summarization (<https://quillbot.com/summarize>)
  - Text Proof-reading – Spelling & Grammar\_
  - Machine Translation\_
  - **Fake News and Cyberbullying Detection**
  - Monitoring Social Media Using NLP
  - Plagiarism detection
  - Look-ahead typing / Word prediction\_
  - Question Answering System (<http://start.csail.mit.edu/index.php>)
  - Sentiment Analysis (<https://komprehend.io/sentiment-analysis>)
-

## 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

## 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

- 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

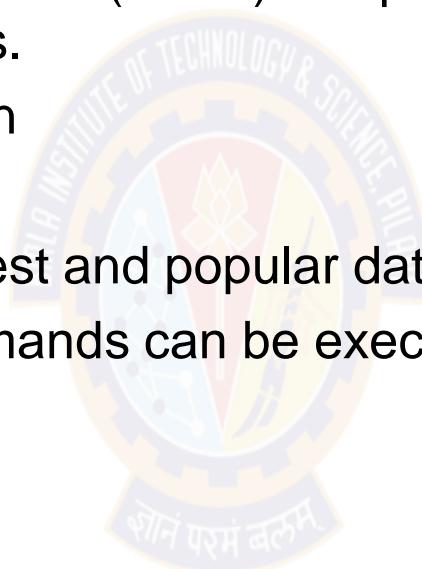
In Jupyter, the console commands can be executed by the ‘!’ sign before the command within the cell.

**! pip install nltk**

[NLTK book](#)

[NLTK discussion forum](#)

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





# NLTK Demo

# Pre-processing text data



Cleaning (or pre-processing) the data typically consists of a number of steps:

- **Remove punctuation**
- **Tokenization**
- **Remove stop words**
- Lemmatize/Stem

# Why is NLP Big Deal

---

- L = Words + rules + exceptions..
- Ambiguity at all levels..
- We speak different languages..
- And language is a cultural entity..
- So they are not equivalent..
- Highly systematic but also complex..
- Keeps changing.. New words, New rules and New exceptions..
- Source : Electronic texts / Printed texts / Acoustic Speech Signal..  
they are noisy..
- Language looks obvious to us.. But it is a Big Deal 😊!



# Why is NLP difficult?



Where is Black Panther playing in Mountain View?

Black Panther is playing at the Century 16 Theater.

When is it playing there?

It's playing at 2pm, 5pm, and 8pm.



OK. I'd like 1 adult and 2 children for the first show.

How much would that cost?

Need domain knowledge, discourse knowledge, world knowledge

# Types of Ambiguities

---

## I. Structural Ambiguities

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

## II. Grammatical Ambiguities

- I (feminine or masculine) go.
- Can- Noun = container, Can – Modal(auxiliary verb),  
Can-verb = to can means to pack etc

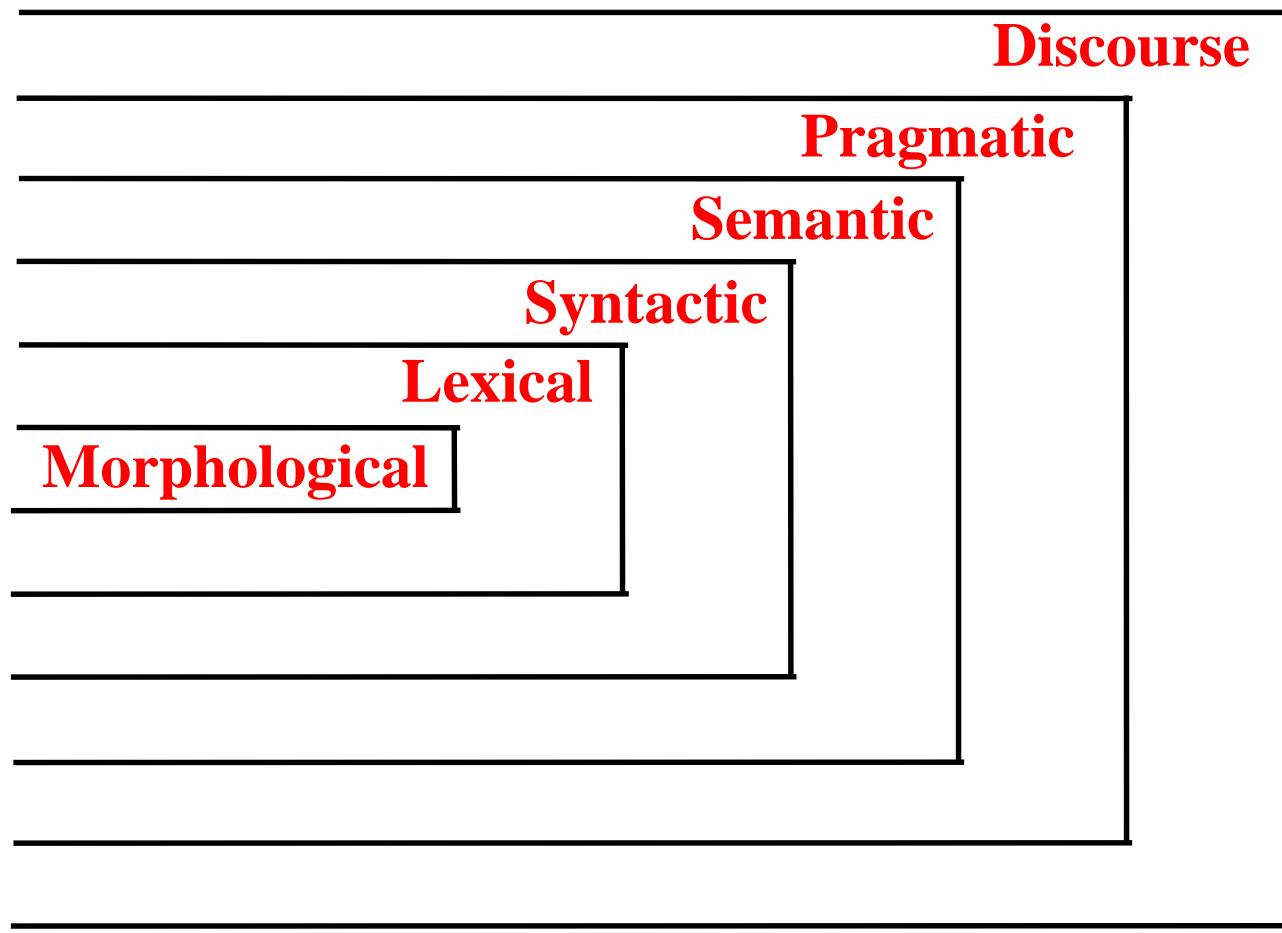
## III. Lexical Ambiguities:

Polysemy Ex: "understand" (I get it)

- Homonymy Ex: Bank= river, financial bank

# Different Levels of Language Analysis

---



# Levels of language understanding

**Morphological Knowledge** : Concerns how words are constructed from basic meaning units called morphemes. A morpheme is the primitive unit of meaning in a language (Ex: “*friendly*” is derived from the meaning of noun “*friend*” and suffix “-ly”, which transforms noun into adjective)

**Lexical Knowledge** : Concerns with listing of words and categorizing them Ex: friendful or beautyship is incorrect lexically. But friendship and beautiful is correct

**Syntactic Knowledge** : Concerns how words can be put together to form correct sentences and determines what structural role each word plays in the sentence and what phrases are subpart of other phrases Ex: “Large have green ideas nose” is lexically correct but syntactically incorrect.

# Levels of language understanding

---

**Semantic Knowledge :**Concerns what words mean and how these meanings – combine in sentences to form sentence meanings. This is the study of context-independent meaning. Ex: “Green ideas have large noses” is syntactically correct but semantically incorrect.

**Pragmatic Knowledge :**Concerns how sentences are used in different situations how use affects the interpretation of sentence “She cuts banana with a pen” is semantically correct but pragmatically incorrect as it has no useful meaning.

**Discourse Knowledge :**Concerns how the immediately preceding sentences affect the interpretation of next sentence  
Ex: Chetana completed PhD student at IIT Bombay. She is a Professor at BITS Pilani.

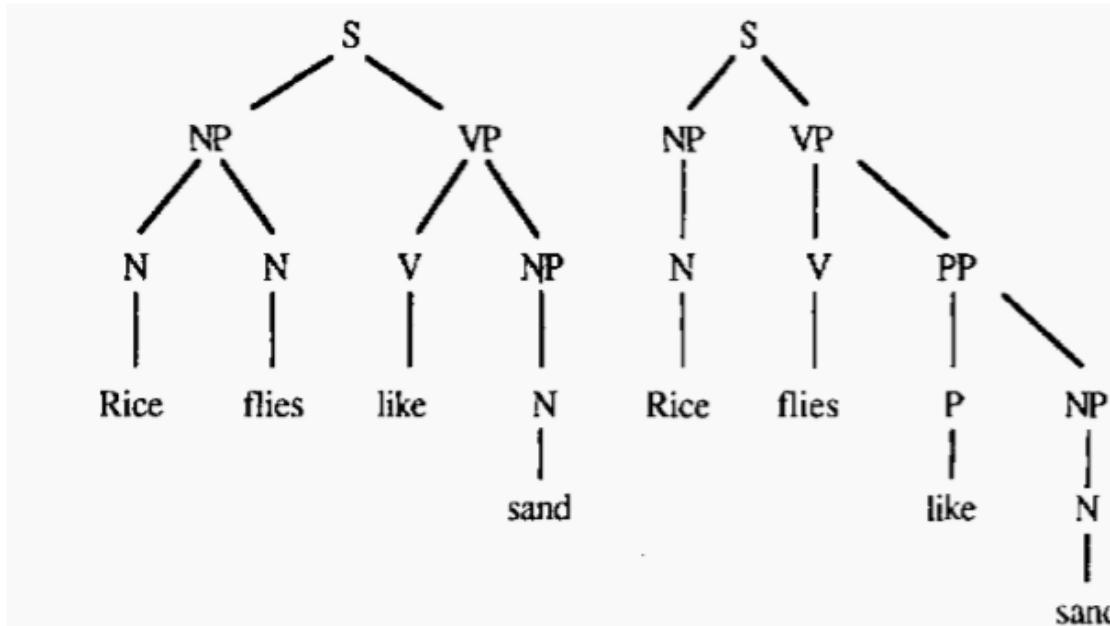
---

# Context Free Grammar

- 
- (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*.

## CFG Rules

$S \rightarrow NP\ VP$   
 $NP \rightarrow N\ N$   
 $NP \rightarrow N$   
 $VP \rightarrow V\ NP$   
 $VP \rightarrow V\ PP$   
 $PP \rightarrow P\ NP$

## Lexicon

Rice : N  
 Flies : N, V  
 Like : V, P  
 Sand : N

NP – Noun Phrases

VP – Verb Phrases

PP – Prepositional Phrases

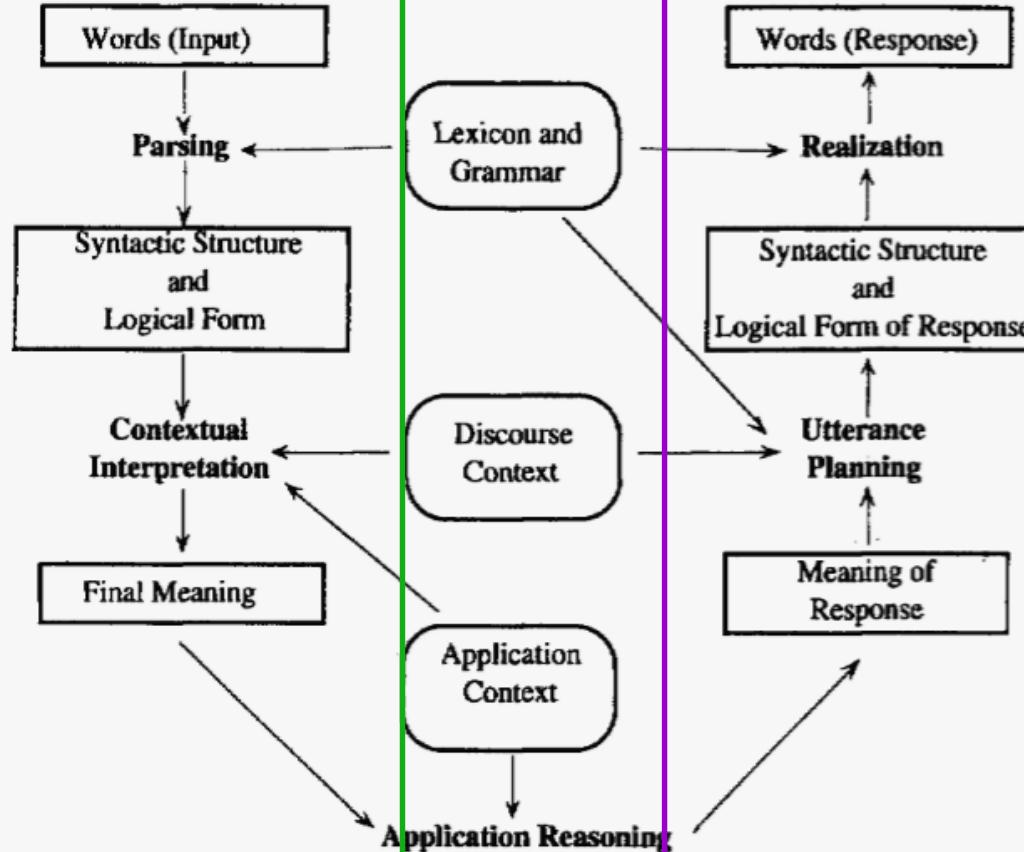
Figure 1.4 Two structural representations of "Rice flies like sand".

# The Organization of Natural Language Processing Systems

Natural Language Understanding

Natural Language Generation

Allen 1995: Natural Language Understanding - Introduction



# 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> = $\frac{tp+tn}{tp+fp+tn+fn}$ |

# Some other evaluation measures

---

- Manual (the best!?):
  - SSER (subjective sentence error rate)
  - Correct/Incorrect
  - **Adequacy and Fluency** (5 or 7 point scales)
  - Error categorization
  - **Comparative ranking of translations**
- Testing in an application that uses MT as one sub-component
  - E.g., question answering from foreign language documents
    - May not test many aspects of the translation (e.g., cross-lingual IR)

# Good 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/NPCourseraSlides.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/>

---



Dr. Chetana is an Associate Professor in the CSIS department at Work Integrated Learning Programmes Division, BITS Pilani. She has more than 25 years of teaching and industry experience. She did her PhD in Computer Science and Engineering from a joint programme of IIT Bombay and Monash University, Australia. She has been working extensively on different state of art research projects and has been awarded the “Best Industry Aligned Research” at the CSI TechNext India 2019 - Awards to Academia. She has published various papers and is also a reviewer at national and international level peer reviewed conferences and journals. Her areas of expertise include Machine Learning, Natural Language Processing, Semantic Web, Deep Learning, Text Mining, Big Data Analytics, Information Retrieval and Software Engineering.

# Thank you!!



**BITS** Pilani  
Pilani Campus



# Natural Language Processing DSECL ZG565

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



## **Session 2**

### **Date – 27<sup>th</sup> May 2023**

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

# Session Content

---

## N-Grams Language models (Chapter 3 Jurafsky and Martin Book)

- Language Models
- N-gram language models
- Evaluation of Language models
- Smoothing
- Interpolation and Back off

# Language modelling



A model that computes either of these:

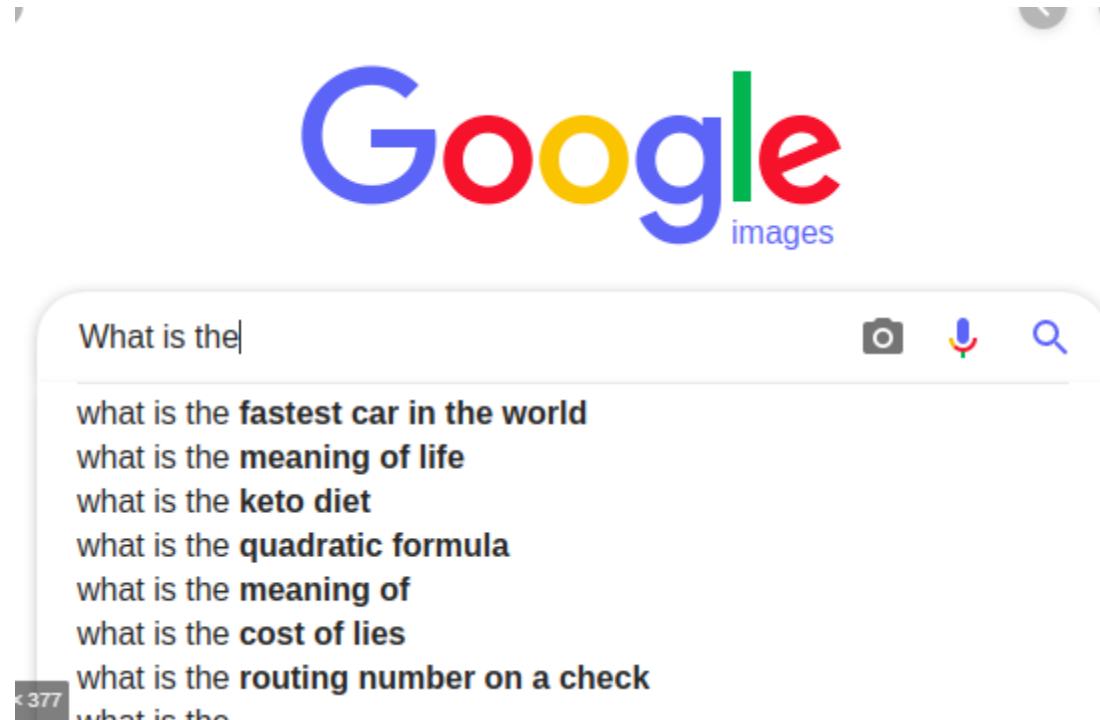
Probability of a sentence (  $P(W)$  ) or

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.

# Example



# Language Models



## 1. Machine Translation:

Machine translation system is used to translate one language to another language

**P(**high** winds tonight) > P(**large** winds tonight)**

## 2.Spell correction

The office is about fifteen minuets from my house



$P(\text{about fifteen minutes from}) > P(\text{about fifteen minuets from})$

### 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.

Example:

$P(\text{I saw a van}) >> P(\text{eyes awe of an})$



# 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

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

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

---

$$P(w_1 w_2 \dots w_n) = \prod_{k=1}^n P(w_k | w_1^{k-1})$$

For ex:

$$P(\text{"its water is so transparent"}) =$$

$$P(\text{its}) \times P(\text{water} | \text{its}) \times P(\text{is} | \text{its water})$$

$$\times P(\text{so} | \text{its water is}) \times P(\text{transparent} | \text{its water is so})$$

Where:

$$P(\text{its}) = \#(\text{its})$$

$$P(\text{water} | \text{its}) = \#(\text{its water}) / \#(\text{its})$$

$$P(\text{is} | \text{its water}) = \#(\text{its water is}) / \#(\text{its water})$$

$$P(\text{so} | \text{its water is}) = \#(\text{its water is so}) / \#(\text{its water is})$$

$$P(\text{the} | \text{its water is so transparent}) = \#(\text{its water is so transparent}) / \#(\text{its water is so})$$

---

- Simplifying assumption:

*limit history of fixed number of words, n*



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 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:

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

# Bigram model

N=2

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

Example:

$P(\text{I want to eat healthy food}) \approx P(\text{I} | \text{<start>}) P(\text{want} | \text{I}) P(\text{to} | \text{want})$   
 $P(\text{eat} | \text{to}) P(\text{healthy} | \text{eat}) P(\text{food} | \text{ healthy})$   
 $P(\text{<end>} | \text{food})$

# N-gram models

---

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

---

- 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>

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

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

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

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

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

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

More examples:

## Berkeley Restaurant Project sentences

---

- 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

- Out of 9222 sentences

|                               | i  | <sup>Next word</sup> want | to  | eat | chinese | food | lunch | spend |
|-------------------------------|----|---------------------------|-----|-----|---------|------|-------|-------|
| i<br><sup>Previous word</sup> | 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     |

# Raw bigram probabilities

| i    | want | to   | eat | chinese | food | lunch | spend |
|------|------|------|-----|---------|------|-------|-------|
| 2533 | 927  | 2417 | 746 | 158     | 1093 | 341   | 278   |

$$P(i|i) = 5/2533 = 0.002$$

$$P(want|i) = 927 / 2533 = 0.33$$

$$P(to|i) = 0 / 2533 = 0$$

$$P(eat|i) = 9 / 2533 = 0.0036$$

$$P(i|want) = 2/927 = 0.0022$$

$$P(to|want) = 608 / 927 = 0.66$$

$$P(eat|to) = 686/2417 = 0.28$$

|         | 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       |

# Bigram estimates of sentence probabilities

---

$$\begin{aligned} P(< \text{s} > \text{ I want english food } < / \text{s} >) &= \\ P(\text{I} | < \text{s} >) & \\ \times P(\text{want} | \text{I}) & \\ \times P(\text{english} | \text{want}) & \\ \times P(\text{food} | \text{english}) & \\ \times P(< / \text{s} > | \text{food}) & \\ = .000031 & \end{aligned}$$

# What kinds of knowledge?

---

- $P(\text{english} | \text{want}) = .0011$
  - $P(\text{chinese} | \text{want}) = .0065$
  - $P(\text{to} | \text{want}) = .66$
  - $P(\text{eat} | \text{to}) = .28$
  - $P(\text{food} | \text{to}) = 0$
  - $P(\text{want} | \text{spend}) = 0$
  - $P(\text{i} | \langle s \rangle) = .25$
-

# Practical Issues

---

- We do everything in log space
  - Avoid underflow
  - (also adding is faster than multiplying)

$$\log(p_1 \cdot p_2 \cdot p_3 \cdot p_4) = \log p_1 + \log p_2 + \log p_3 + \log p_4$$

# Google N-Gram Release

AUG

3

## All Our N-gram are Belong to You

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word [n-gram models](#) for a variety of R&D projects,

...

That's why we decided to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.

# Google N-Gram Release

- serve as the incoming 92
- serve as the incubator 99
- serve as the independent 794
- serve as the index 223
- serve as the indication 72
- serve as the indicator 120
- serve as the indicators 45
- serve as the indispensable 111
- serve as the indispensible 40
- serve as the individual 234

<http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html>

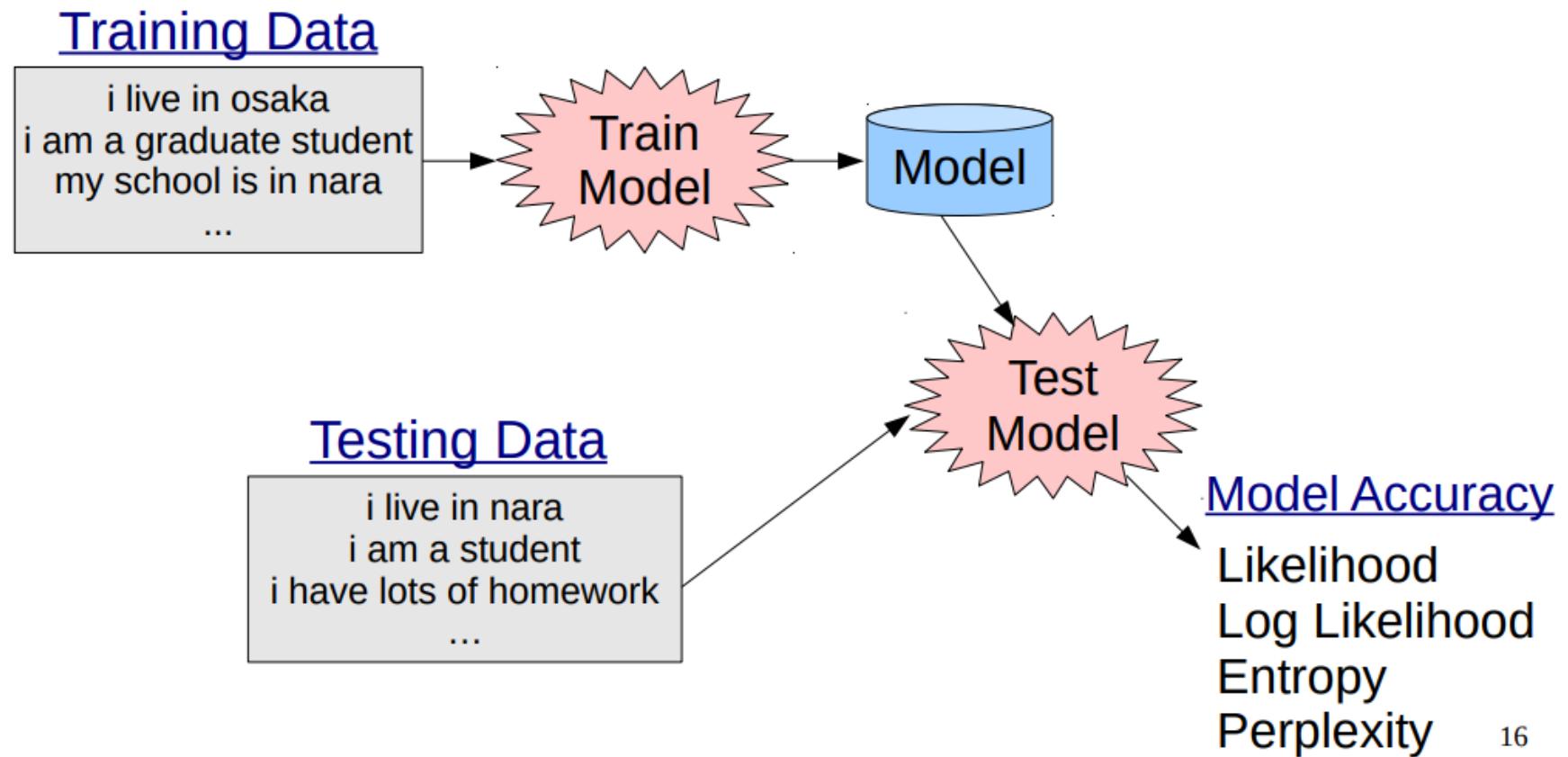
<https://pypi.org/project/google-ngram-downloader/>

# 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



# 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
- So
  - Sometimes use **intrinsic** evaluation: **perplexity**
  - Bad approximation
    - unless the test data looks **just** like the training data
    - So **generally only useful in pilot experiments**
  - But is helpful to think about.

# Intuition of Perplexity



- The Shannon Game:
  - How well can we predict the next word?

I always order pizza with cheese and \_\_\_\_\_

The 33<sup>rd</sup> President of the US was \_\_\_\_\_

I saw 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:

$$\begin{aligned} PP(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \end{aligned}$$

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

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

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

# Perplexity as branching factor

---

- Let's suppose a sentence consisting of random digits 0 to 9 Ex: 0,2,1,3.....N
- What is the perplexity of this sentence according to a model that assign P=1/10 to each digit?

$$\begin{aligned} \text{PP}(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \left(\frac{1}{10}\right)^{-\frac{1}{N}} \\ &= \frac{1}{10}^{-1} \\ &= 10 \end{aligned}$$

# Lower perplexity = better model

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

| N-gram Order | Unigram | Bigram | Trigram |
|--------------|---------|--------|---------|
| Perplexity   | 962     | 170    | 109     |

# 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

# Zeros

---

- 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:

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

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

# Maximum Likelihood Estimates

- The maximum likelihood estimate
  - of some parameter of a model M from a training set T
  - maximizes the likelihood of the training set T given the model M
- Suppose the word “bagel” occurs 400 times in a corpus of a million words
- What is the probability that a random word from some other text will be “bagel”?
- MLE estimate is  $400/1,000,000 = .0004$
- This may be a bad estimate for some other corpus
  - But it is the **estimate** that makes it **most likely** that “bagel” will occur 400 times in a million word corpus.

# Raw bigram counts

- Out of 9222 sentences

|         | 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     |

# Raw bigram probabilities

| i    | want | to   | eat | chinese | food | lunch | spend |
|------|------|------|-----|---------|------|-------|-------|
| 2533 | 927  | 2417 | 746 | 158     | 1093 | 341   | 278   |

$$P(i|i) = 5/2533 = 0.002$$

$$P(want|i) = 927 / 2533 = 0.33$$

$$P(to|i) = 0 / 2533 = 0$$

$$P(eat|i) = 9 / 2533 = 0.0036$$

$$P(i|want) = 2/927 = 0.0022$$

$$P(to|want) = 608 / 927 = 0.66$$

$$P(eat|to) = 686/2417 = 0.28$$

|         | 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 counts

| i    | want | to   | eat | chinese | food | lunch | spend |
|------|------|------|-----|---------|------|-------|-------|
| 2533 | 927  | 2417 | 746 | 158     | 1093 | 341   | 278   |

V = Total number of unique words in corpus = 1446

|         | i  | want | to  | eat | chinese | food | lunch | spend |
|---------|----|------|-----|-----|---------|------|-------|-------|
| i       | 6  | 828  | 1   | 10  | 1       | 1    | 1     | 3     |
| want    | 3  | 1    | 609 | 2   | 7       | 7    | 6     | 2     |
| to      | 3  | 1    | 5   | 687 | 3       | 1    | 7     | 212   |
| eat     | 1  | 1    | 3   | 1   | 17      | 3    | 43    | 1     |
| chinese | 2  | 1    | 1   | 1   | 1       | 83   | 2     | 1     |
| food    | 16 | 1    | 16  | 1   | 2       | 5    | 1     | 1     |
| lunch   | 3  | 1    | 1   | 1   | 1       | 2    | 1     | 1     |
| spend   | 2  | 1    | 2   | 1   | 1       | 1    | 1     | 1     |

# Laplace-smoothed bigrams

| i    | want | to   | eat | chinese | food | lunch | spend |
|------|------|------|-----|---------|------|-------|-------|
| 2533 | 927  | 2417 | 746 | 158     | 1093 | 341   | 278   |

V = Total number of unique words in corpus = 1446

$$P(\text{want}|i) = 828 / 2533 + 1446 = 828 / 3979 = 0.21$$

$$P(\text{to}|i) = 1 / 3979 = 0.00025$$

$$P(\text{eat}|i) = 10 / 3979 = 0.0025$$

$$P(\text{to}| \text{want}) = 609 / 927 + 1446 = 609 / 2373 = 0.26$$

$$P(\text{eat}| \text{to}) = 687 / 2417 + 1446 = 0.18$$

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}$$

|         | i              | want           | to             | eat            | chinese        | food           | lunch          | spend          |
|---------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| i       | 0.0015         | 0.21           | <b>0.00025</b> | 0.0025         | <b>0.00025</b> | <b>0.00025</b> | <b>0.00025</b> | 0.00075        |
| want    | 0.0013         | <b>0.00042</b> | 0.26           | 0.00084        | 0.0029         | 0.0029         | 0.0025         | 0.00084        |
| to      | 0.00078        | <b>0.00026</b> | 0.0013         | 0.18           | 0.00078        | <b>0.00026</b> | 0.0018         | 0.055          |
| eat     | <b>0.00046</b> | 0.00046        | 0.0014         | <b>0.00046</b> | 0.0078         | 0.0014         | 0.02           | <b>0.00046</b> |
| chinese | 0.0012         | 0.00062        | <b>0.00062</b> | 0.00062        | <b>0.00062</b> | 0.052          | 0.0012         | 0.00062        |
| food    | 0.0063         | 0.00039        | 0.0063         | 0.00039        | 0.00079        | 0.002          | <b>0.00039</b> | 0.00039        |
| lunch   | 0.0017         | <b>0.00056</b> | <b>0.00056</b> | 0.00056        | <b>0.00056</b> | 0.0011         | <b>0.00056</b> | 0.00056        |
| spend   | 0.0012         | <b>0.00058</b> | 0.0012         | 0.00058        | <b>0.00058</b> | <b>0.00058</b> | 0.00058        | 0.00058        |

# Reconstituted counts



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

$$c(i, \text{want}) = (828 * 2533) / (2533 + 1446) = 527$$

|         | 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

- Simple interpolation

$$\begin{aligned}\hat{P}(w_n | w_{n-2} w_{n-1}) &= \lambda_1 P(w_n | w_{n-2} w_{n-1}) \\ &\quad + \lambda_2 P(w_n | w_{n-1}) \\ &\quad + \lambda_3 P(w_n)\end{aligned}\qquad\qquad\qquad \sum_i \lambda_i = 1$$

- Lambdas conditional on context:

$$\begin{aligned}\hat{P}(w_n | w_{n-2} w_{n-1}) &= \lambda_1(w_{n-2}^{n-1}) P(w_n | w_{n-2} w_{n-1}) \\ &\quad + \lambda_2(w_{n-2}^{n-1}) P(w_n | w_{n-1}) \\ &\quad + \lambda_3(w_{n-2}^{n-1}) P(w_n)\end{aligned}$$

# 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:

$$\log P(w_1 \dots w_n | M(/_1 \dots /_k)) = \sum_i \log P_{M(/_1 \dots /_k)}(w_i | w_{i-1})$$

# 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
  - Only store N-grams with count > threshold.
    - Remove singletons of higher-order n-grams

# Smoothing for Web-scale N-grams

- “Stupid backoff” (Brants *et al.* 2007)
- No discounting, just use relative frequencies

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

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

# Corpora of text and speech

- Brown Corpus
- Wall Street Journal
- AP newswire
- Hansards
- DARPA/NIST text/speech corpora (Call Home, ATIS, switchboard, Broadcast News, TDT, Communicator)
- TRAINS, Radio News

# References

<https://books.google.com/ngrams>

<https://ai.googleblog.com/2006/08/all-our-n-gram-are-belong-to-you.html>

<http://www.speech.sri.com/projects/srilm/>

<https://web.stanford.edu/~jurafsky/NLPCourseraSlides.html>

<https://www.kaggle.com/code/alvations/n-gram-language-model-with-nltk/notebook>

<https://levelup.gitconnected.com/bi-tri-and-n-grams-with-python-a9717264e6f2>



Dr. Chetana Gavankar has over 25 years of Teaching, Research and Industry experience. She has published papers in peer reviewed international conferences and journals. She is also reviewer for multiple conferences and journals. She has worked on different projects with multiple industries and received awards for her research work . Her areas of research interests include Natural Language Processing, Information Retrieval, Web Mining and Semantic Web, Ontology, Big Data Analytics, Machine learning, Deep learning and Artificial Intelligence.

# Thank you!!



**BITS** Pilani  
Pilani Campus

# Natural Language Processing

## DSECL ZG565

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



## Session 3

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

# Session Content

---

## Vector Semantics and Word Embedding (Chapter 6 Jurafsky and Martin Book)

- Lexical semantics
  - Vector semantics
  - Word and Vectors
  - TFIDF
  - Word2Vec
    - Skip gram
    - CBOW
  - Glove
  - Visualizing Embedding's
-

- **Structure** of words : **morphology**
- **Distribution** of words: **language modeling**
- **Meaning** of words: **lexical semantics**
- **Distributional hypothesis:** a way to identify words with similar meanings
  - Words that occur in **similar contexts** tend to have **similar meanings**
  - E.g. oculist and eye-doctor occur near words like eye or examined
  - Amount of meaning difference between two words “corresponding roughly to the amount of difference in their environments (Context)”
  - two words that occur in very similar distributions (whose neighboring words are similar) have similar meanings.
  - **Basic idea: Measure the semantic similarity of words in terms of the similarity of the contexts in which they appear**

# 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 [Ebay](#), [Amazon](#) and [computing-client](#)

### MAINFRAMES

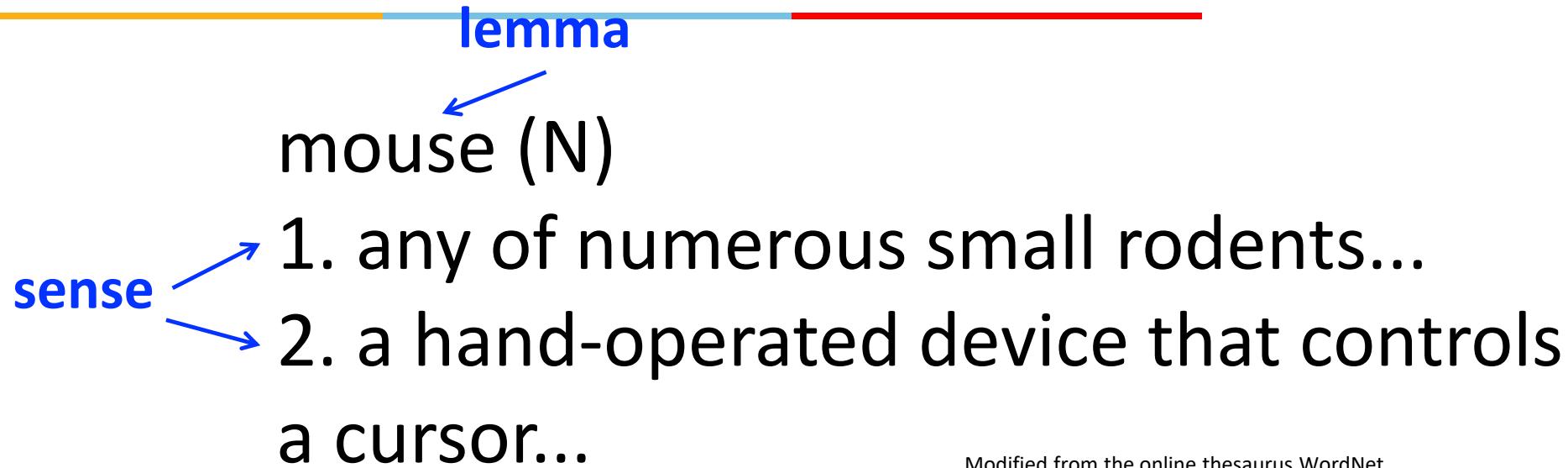
Mainframes usually are referred to those computers with fast, advanced processing capabilities that could perform by itself tasks that may require a lot of Personal Computers (PC) Machines. Usually mainframes would have lots of RAMs, very large secondary storage devices, and very fast processors to cater 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!
  - Introductory logic classes:
    - The meaning of "dog" is DOG; cat is CAT
$$\forall x \text{ DOG}(x) \rightarrow \text{MAMMAL}(x)$$
  - Old linguistics joke by Barbara Partee in 1967:
    - Q: What's the meaning of life?
    - A: LIFE
  - That seems hardly better!
-



Modified from the online thesaurus WordNet

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

# Lexical semantics

- lexical semantics, the linguistic **study of word meaning**
- Concepts or **word senses**
  - Have a complex many-to-many association with words
- Have relations with each other
  - Synonymy
  - Similarity
  - Relatedness: semantic field and frame
  - Antonymy
  - Connotation
- **More generally, a model of word meaning should allow us to draw inferences to address meaning-related tasks like question-answering or dialogue.**

# Relations between senses: **Synonymy**

---

- Synonyms have the same meaning in some or all contexts.
  - Same word sense, substitutable for one another, same propositional meaning, truth preserving
  - E.g couch/sofa, vomit/throw up, car/automobile
- water/H<sub>2</sub>O : "H<sub>2</sub>O" in a surfing guide?
- big/large: my big sister != my large sister
- In practice, the word synonym is therefore used to describe a relationship of approximate or rough synonymy,

# Relation: Similarity

- Words with similar meanings. Not synonyms, but sharing some element of meaning  
(Cat is not a synonym of dog, but cats and dogs are certainly similar words)
  - car, bicycle ; cow, horse
- The notion of word similarity is very useful in larger semantic tasks.
  - Help in computing how similar the meaning of two phrases or sentences are
  - Applications : question answering, paraphrasing, and summarization.
- Humans to judge how similar one word is to another

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

| 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        |

## 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.

## **hospitals**

*surgeon, scalpel, nurse, anaesthetic, hospital*

## **restaurants**

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

## **houses**

*door, roof, kitchen, family, bed*

# Relation: Antonymy

- Senses that are opposites with respect to only one feature of meaning
- Otherwise, they are **very similar!**

|            |            |           |           |
|------------|------------|-----------|-----------|
| dark/light | short/long | fast/slow | rise/fall |
| hot/cold   | up/down    |           | in/out    |

- More formally: antonyms can
  - define a binary opposition or be at opposite ends of a scale
    - long/short, fast/slow
  - Be *reversives*:
    - rise/fall, up/down

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

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

# Computational models of word meaning

---

Can we build a theory of how to represent word meaning?

We'll introduce **vector semantics**

- The standard model in language processing!
- Handles many of our goals!

# Let's define words by their usages

---

One way to define "usage":

words are defined by their environments (the words around them)

Zellig Harris (1954):

**If A and B have almost identical environments we say that they are synonyms.**

# 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

Idea 1: Defining meaning by linguistic distribution

Idea 2: Meaning as a point in multidimensional space

# Word embeddings

- Vectors for representing words are called embeddings
- Each word = a vector (not just "good" or " $w_{45}$ ")
- Defining meaning as a point in space based on distribution
- Similar words are "**nearby in semantic space**"
- **Every modern NLP algorithm uses embeddings as the representation of word meaning**

two-dimensional (t-SNE) projection of embeddings



We define meaning of a word as a vector

---

- Called an "embedding" because it's embedded into a space
- The standard way to represent meaning in NLP
- **Every modern NLP algorithm uses embeddings as the representation of word meaning**
- Fine-grained model of meaning for similarity

# Intuition: why vectors?

---

- 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**

# Vectors are the basis of information retrieval

Each document is represented by a vector of words

|        | As You Like It | Twelfth Night | Julius Caesar | Henry V |
|--------|----------------|---------------|---------------|---------|
| battle | 1              | 0             | 7             | 13      |
| good   | 114            | 80            | 62            | 89      |
| fool   | 36             | 58            | 1             | 4       |
| wit    | 20             | 15            | 2             | 3       |

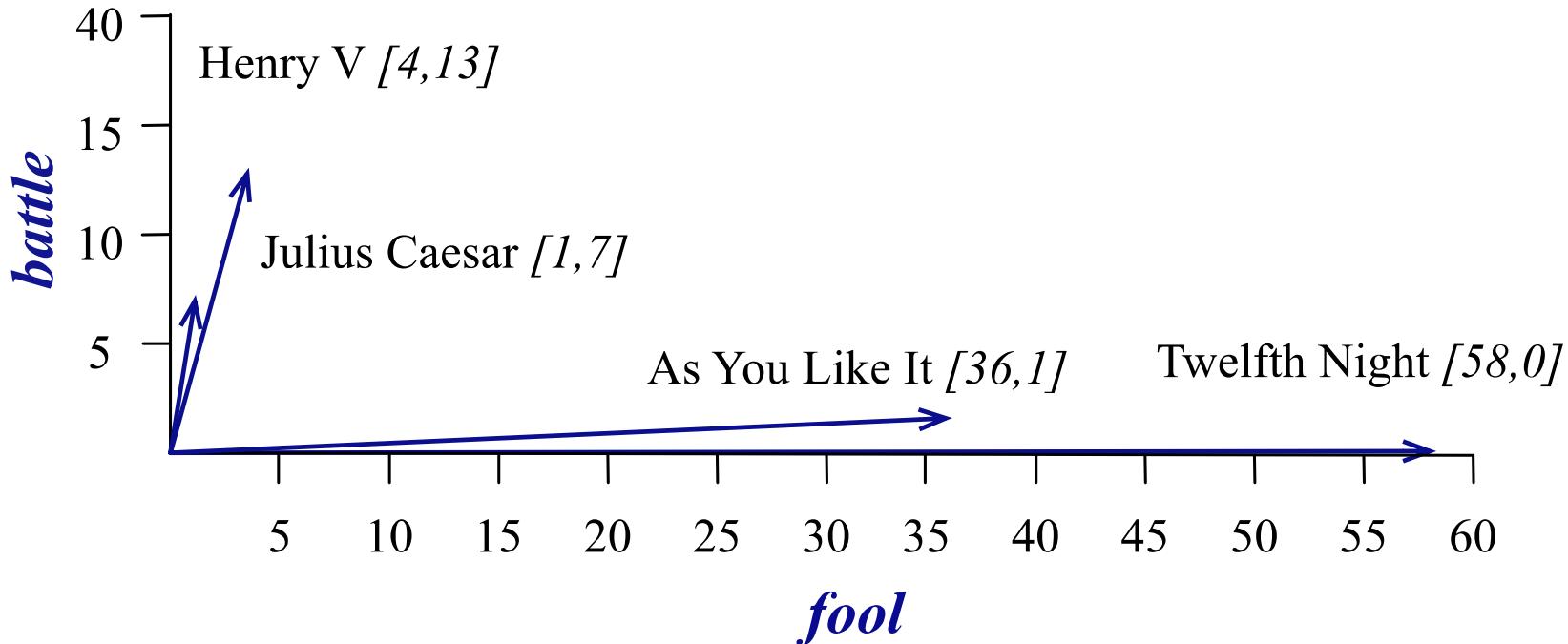
How many times it  
is occurring in the  
document.

Vectors are similar for the two comedies

But comedies are different than the other two

Comedies have more *fools* and *wit* and fewer *battles*.

# Visualizing document vectors



# Idea for word meaning: Words can be vectors too!!!

How many times it is occurring in the document.

|        | As You Like It | Twelfth Night | Julius Caesar | Henry V |
|--------|----------------|---------------|---------------|---------|
| battle | 1              | 0             | 7             | 13      |
| good   | 114            | 80            | 62            | 89      |
| fool   | 36             | 58            | 1             | 4       |
| wit    | 20             | 15            | 2             | 3       |

*battle* is "the kind of word that occurs in Julius Caesar and Henry V"

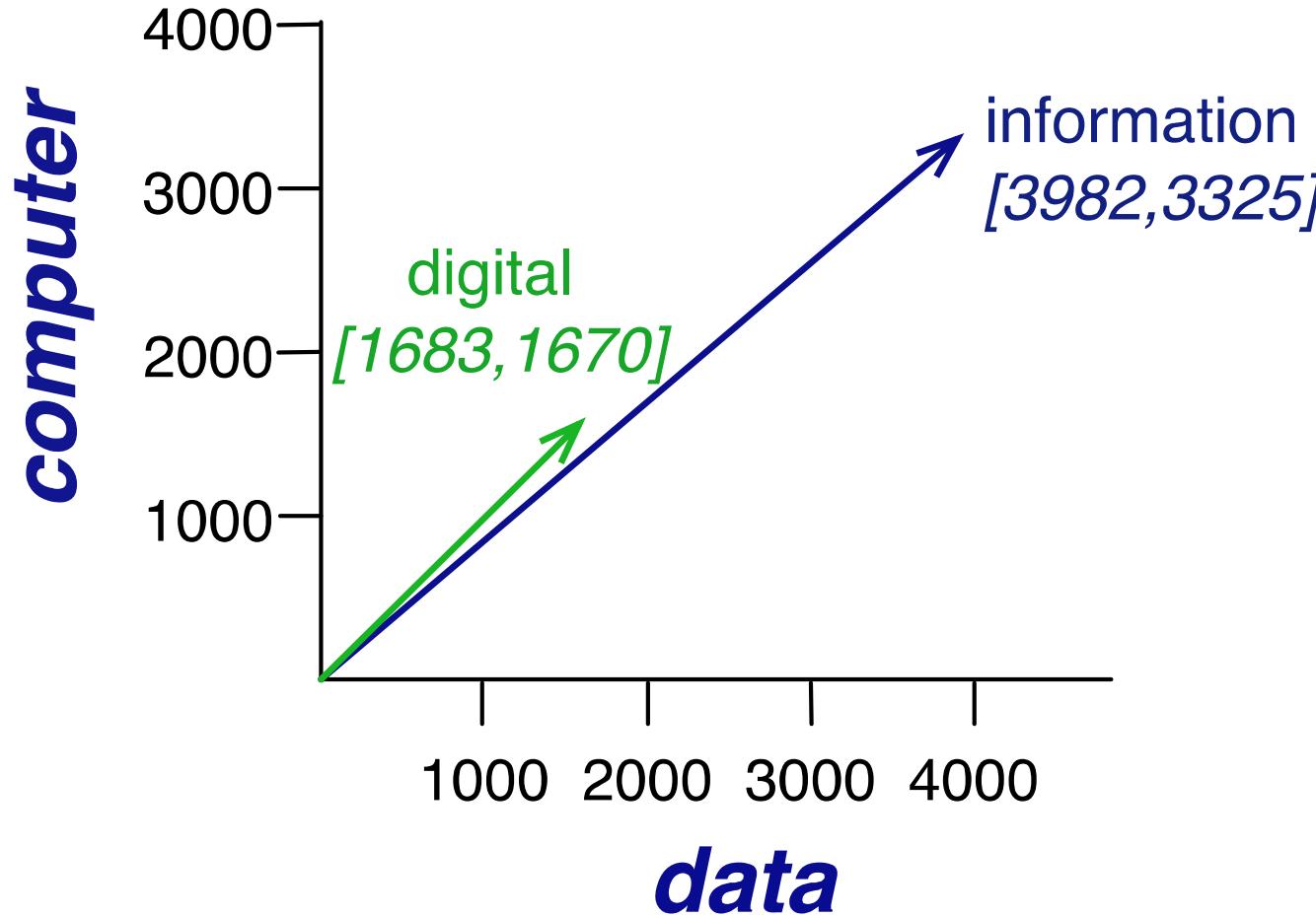
*fool* is "the kind of word that occurs in comedies, especially Twelfth Night"

# More common: word-word matrix (or "term-context matrix")

- Two **words** are similar in meaning if their context vectors are similar

is traditionally followed by **cherry** pie, a traditional dessert  
 often mixed, such as **strawberry** rhubarb pie. Apple pie  
 computer peripherals and personal **digital** assistants. These devices usually  
 a computer. This includes **information** available on the internet

|             | aardvark | ... | computer | data | result | pie | sugar | ... |
|-------------|----------|-----|----------|------|--------|-----|-------|-----|
| cherry      | 0        | ... | 2        | 8    | 9      | 442 | 25    | ... |
| strawberry  | 0        | ... | 0        | 0    | 1      | 60  | 19    | ... |
| digital     | 0        | ... | 1670     | 1683 | 85     | 5   | 4     | ... |
| information | 0        | ... | 3325     | 3982 | 378    | 5   | 13    | ... |



# Computing word similarity: Dot product and cosine

---

- The dot product between two vectors is a scalar:

$$\text{dot product}(\mathbf{v}, \mathbf{w}) = \mathbf{v} \cdot \mathbf{w} = \sum_{i=1}^N v_i w_i = v_1 w_1 + v_2 w_2 + \dots + v_N w_N$$

- The dot product tends to be high when the two vectors have large values in the same dimensions
  - Dot product can thus be a useful similarity metric between vectors
-

## Problem with raw dot-product

---

- Dot product favors long vectors
- Dot product is higher if a vector is longer (has higher values in many dimension)

- Vector length:

$$|\mathbf{v}| = \sqrt{\sum_{i=1}^N v_i^2}$$

- Frequent words (of, the, you) have long vectors (since they occur many times with other words).
- So dot product overly favors frequent words

# Alternative: cosine for computing word similarity

$$\text{cosine}(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}| |\vec{w}|} = \frac{\sum_{i=1}^N v_i w_i}{\sqrt{\sum_{i=1}^N v_i^2} \sqrt{\sum_{i=1}^N w_i^2}}$$

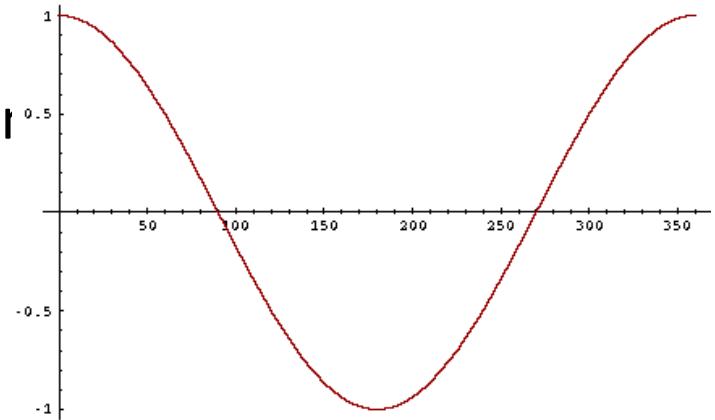
Based on the definition of the dot product between two vectors a and b

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \cos \theta$$

$$\frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}| |\mathbf{b}|} = \cos \theta$$

## Cosine as a similarity metric

- -1: vectors point in opposite directions
- +1: vectors point in same directions
- 0: vectors are orthogonal



- But since raw frequency values are non-negative, the cosine for <sup>32</sup> term-term matrix vectors ranges from 0–1

# Cosine examples

$$\cos(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{\|\vec{v}\| \|\vec{w}\|} = \frac{\vec{v}}{\|\vec{v}\|} \cdot \frac{\vec{w}}{\|\vec{w}\|} = \frac{\sum_{i=1}^N v_i w_i}{\sqrt{\sum_{i=1}^N v_i^2} \sqrt{\sum_{i=1}^N w_i^2}}$$

|             | pie | data | computer |
|-------------|-----|------|----------|
| cherry      | 442 | 8    | 2        |
| digital     | 5   | 1683 | 1670     |
| information | 5   | 3982 | 3325     |

Vector

$$\cos(\text{cherry}, \text{information}) =$$

$$\frac{442 * 5 + 8 * 3982 + 2 * 3325}{\sqrt{442^2 + 8^2 + 2^2} \sqrt{5^2 + 3982^2 + 3325^2}} = .017$$

How similar they are , based on cosine

$$\cos(\text{digital}, \text{information}) =$$

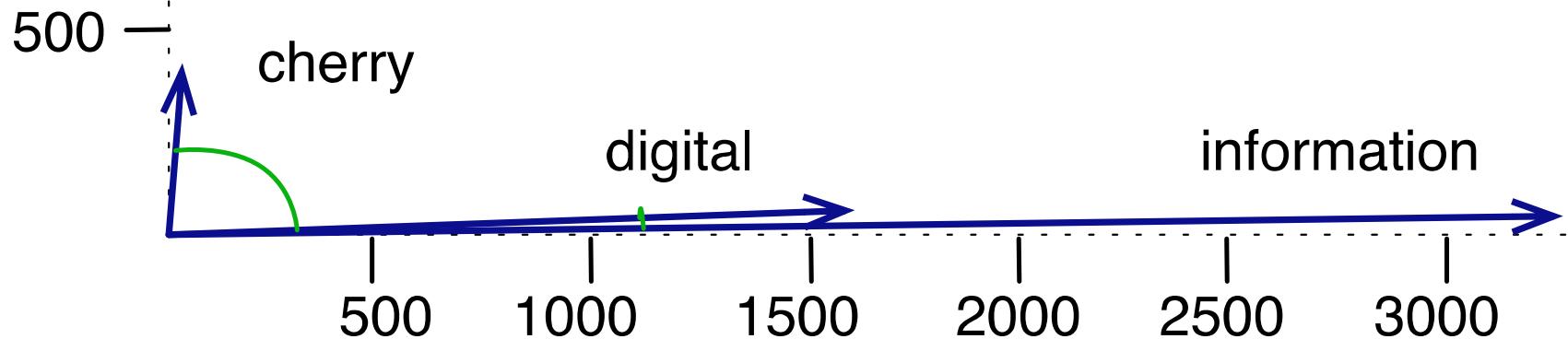
-1 = opposite  
+1 = same direction,similar  
0= vectors are orthogonal

33

$$\frac{5 * 5 + 1683 * 3982 + 1670 * 3325}{\sqrt{5^2 + 1683^2 + 1670^2} \sqrt{5^2 + 3982^2 + 3325^2}} = .996$$

## Visualizing cosines (well, angles)

*Dimension 1: 'pie'*



*Dimension 2: 'computer'*

# Tf-idf

---

- Raw frequency is a bad representation
- The co-occurrence matrices we have seen represent each cell by word frequencies.
- Frequency is clearly useful; if *sugar* appears a lot near *apricot*, that's useful information.
- But overly frequent words like *the*, *it*, or *they* are not very informative about the context
- It's a paradox! How can we balance these two conflicting constraints?
- when the dimensions are document word weighting using tf-idf
  - **tf-idf:** tf-idf value for word t in document d:

$$w_{t,d} = \text{tf}_{t,d} \times \text{idf}_t$$

- Words like "the" or "it" have very low idf

# Term frequency (tf)

---

$$\text{tf}_{t,d} = \text{count}(t,d)$$

how many times the term (t) is occurring in a document (d)

Instead of using raw count, we squash a bit:

$$\text{tf}_{t,d} = \log_{10}(\text{count}(t,d) + 1)$$

# Document frequency (df)

Number of documents in which the word is occurring..

- The second factor in  $tf\text{-}idf$  is used to give a higher weight to words that occur only in a few documents
- $df_t$  is the number of documents  $t$  occurs in.
- (note this is not collection frequency: total count across all documents)
- "Romeo" is very distinctive for one Shakespeare play:

|        | Collection Frequency | Document Frequency |
|--------|----------------------|--------------------|
| Romeo  | 113                  | 1                  |
| action | 113                  | 31                 |

# Inverse document frequency (idf)

$$\text{idf}_t = \log_{10} \left( \frac{N}{\text{df}_t} \right)$$

N is the total number of documents  
in the collection

**The fewer documents in which a term occurs,  
the higher this weight**

| Word     | df | idf   |
|----------|----|-------|
| Romeo    | 1  | 1.57  |
| salad    | 2  | 1.27  |
| Falstaff | 4  | 0.967 |
| forest   | 12 | 0.489 |
| battle   | 21 | 0.246 |
| wit      | 34 | 0.037 |
| fool     | 36 | 0.012 |
| good     | 37 | 0     |
| sweet    | 37 | 0     |

# Final tf-idf weighted value for a word

Raw counts:

| Words  | As You Like It | Twelfth Night | Julius Caesar | Henry V | Documents |
|--------|----------------|---------------|---------------|---------|-----------|
| battle | 1              | 0             | 7             | 13      |           |
| good   | 114            | 80            | 62            | 89      |           |
| fool   | 36             | 58            | 1             | 4       |           |
| wit    | 20             | 15            | 2             | 3       |           |

vector is plain count

tf-idf:  $w_{t,d} = \text{tf}_{t,d} \times \text{idf}_t$

**Word: wit**

$$\text{tf} = \log_{10}(20+1) = 1.322, \text{idf} = \log_{10}(37/34)=0.037, \text{tf-idf value}=0.049$$

**Word: good**

appears in every document so tf-idf = 0

|        | As You Like It | Twelfth Night | Julius Caesar | Henry V |
|--------|----------------|---------------|---------------|---------|
| battle | 0.074          | 0             | 0.22          | 0.28    |
| good   | 0              | 0             | 0             | 0       |
| fool   | 0.019          | 0.021         | 0.0036        | 0.0083  |
| wit    | 0.049          | 0.044         | 0.018         | 0.022   |

vector is based on tf-idf

# 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)

# Common methods for getting short dense vectors

- “[Neural Language Model](#)”-inspired models
  - Word2vec (skipgram, CBOW), GloVe
- **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

---

- 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

---

Popular embedding method

Very fast to train

Code available on the web

Idea: **predict** rather than **count**

Word2vec provides various options. We'll do:

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

# Simple static embeddings you can download!

---

- Word2vec (Mikolov et al)
  - <https://code.google.com/archive/p/word2vec/>
  
  - GloVe (Pennington, Socher, Manning)
  - <http://nlp.stanford.edu/projects/glove/>
-

# Basic word representation

$V = [a, \text{aaron}, \dots, \text{zulu}, \text{<UNK>}]$

## 1-hot representation

|               |                 |                |                 |                |                  |
|---------------|-----------------|----------------|-----------------|----------------|------------------|
| Man<br>(5391) | Woman<br>(9853) | King<br>(4914) | Queen<br>(7157) | Apple<br>(456) | Orange<br>(6257) |
|---------------|-----------------|----------------|-----------------|----------------|------------------|

$$\begin{array}{c}
 \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \end{pmatrix} \\
 \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \\
 \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \\
 \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \\
 \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \\
 \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}
 \end{array}$$

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

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

# Featurized representation: word embedding



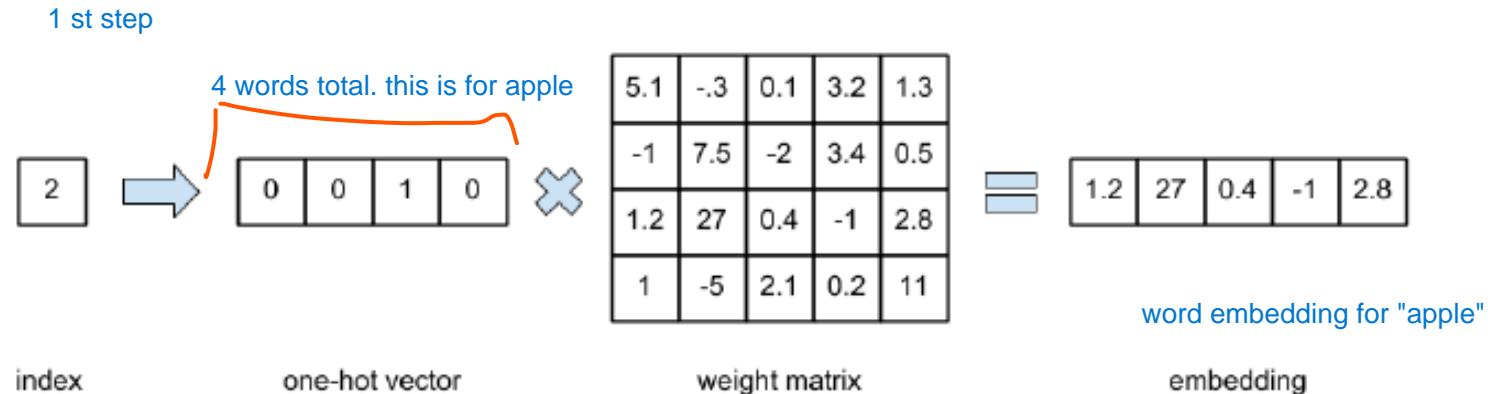
| Words    |  | Man<br>(5391) | Woman<br>(9853) | King<br>(4914) | Queen<br>(7157) | Apple<br>(456) | Orange<br>(6257) |
|----------|--|---------------|-----------------|----------------|-----------------|----------------|------------------|
| Features |  | -1            | 1               | -0.95   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

Matrix of embedding

Similarity

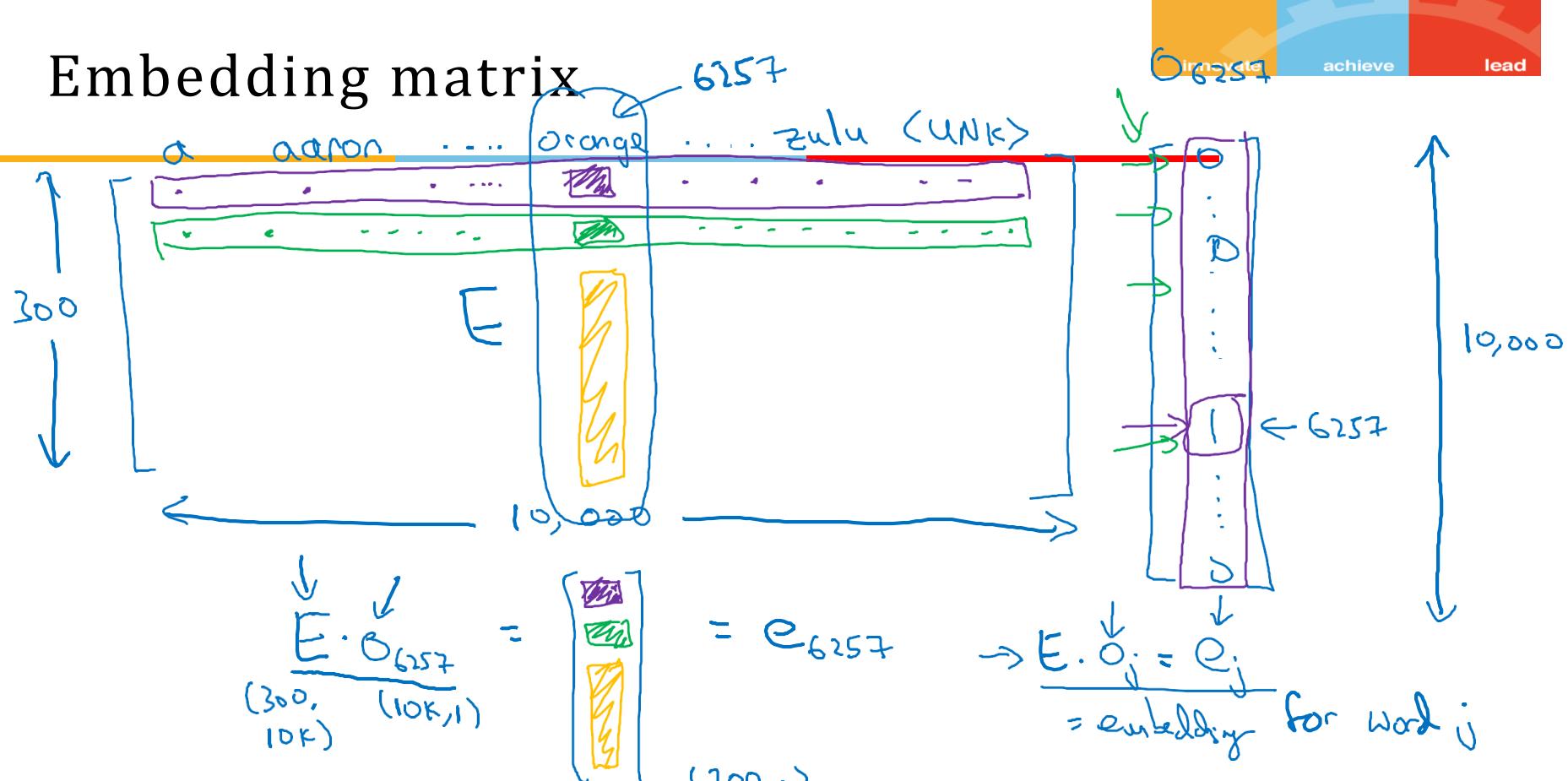
I want a glass of orange I want a glass of apple

# Word embeddings



We have to do it for each unique words.

# Embedding matrix



In practice, use specialized function to look up an embedding.

→ Embedding

# 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
    - Bengio et al. (2003); Collobert et al. (2011)
-

# 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

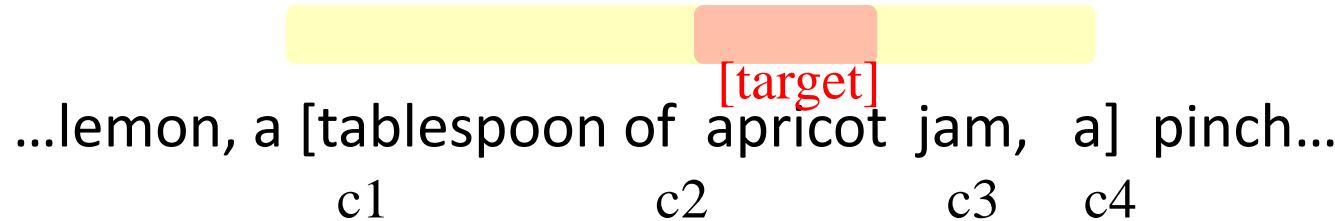
# Approach: predict if candidate word $c$ is a "neighbor"

---

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

# Skip-Gram Training Data

- Assume a +/- 2 word window, given training sentence:



Need to find the probability for each context words. here total 4 context words.

Also, we have to find the probability of non-context word which is negative words. there are outside of context words.

# Skip-Gram Classifier

- (assuming a +/- 2 word window)

...lemon, a [tablespoon of apricot jam, a] pinch...

c1                    c2 [target]    c3    c4

Goal: train a classifier that is given a candidate (word, context) pair  
(apricot, jam)  
(apricot, aardvark)

...

And assigns each pair a probability:

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

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

## Similarity is computed from dot product

---

- Remember: two vectors are similar if they have a high dot product
  - Cosine is just a normalized dot product
- So:
  - $\text{Similarity}(w,c) \propto w \cdot c$
- We'll need to normalize to get a probability
  - (cosine isn't a probability either)

# Turning dot products into probabilities

---

- $\text{Sim}(w, c) \approx w \cdot c$
- To turn this into a probability
- We'll use the sigmoid from logistic regression:

$$P(+|w, c) = \sigma(c \cdot w) = \frac{1}{1 + \exp(-c \cdot w)}$$

$$\begin{aligned} P(-|w, c) &= 1 - P(+|w, c) \\ &= \sigma(-c \cdot w) = \frac{1}{1 + \exp(c \cdot w)} \end{aligned}$$

---

# How Skip-Gram Classifier computes $P(+|w, c)$

$$P(+|w, c) = \sigma(c \cdot w) = \frac{1}{1 + \exp(-c \cdot w)}$$

This is for one context word, but we have lots of context words.  
 We'll assume independence and just multiply them:

$$P(+|w, c_{1:L}) = \prod_{i=1}^L \sigma(c_i \cdot w)$$

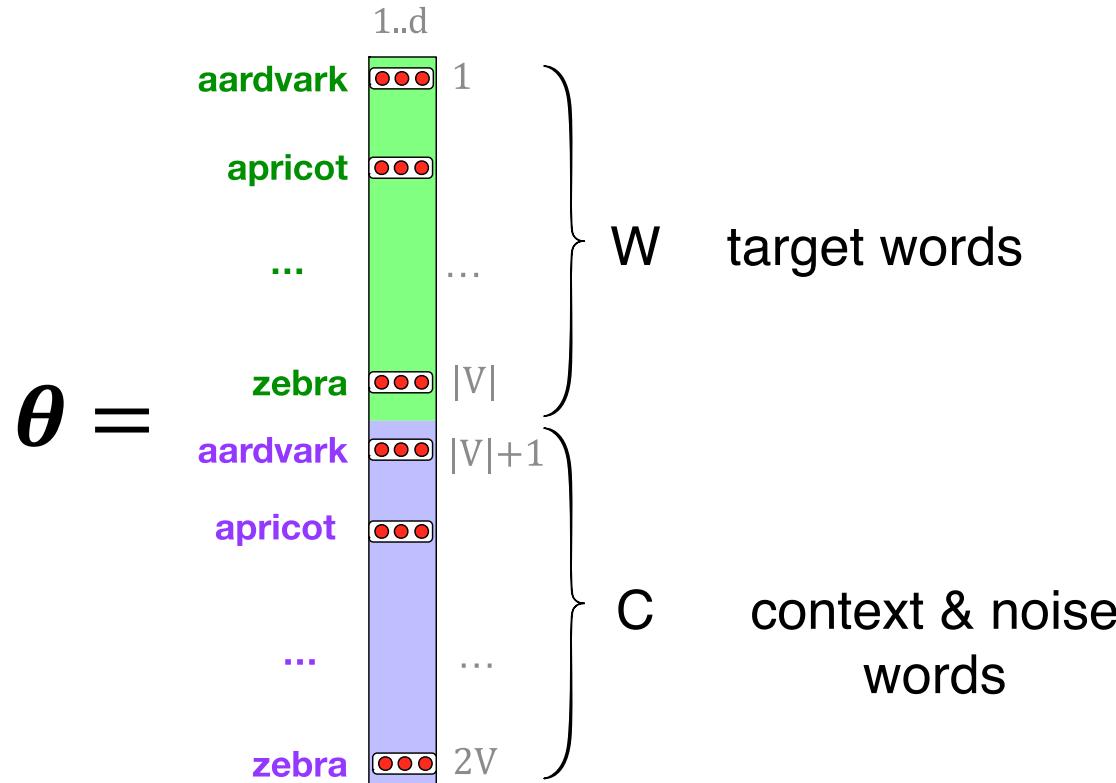
$$\log P(+|w, c_{1:L}) = \sum_{i=1}^L \log \sigma(c_i \cdot w)$$

## Skip-gram classifier: summary

---

- A probabilistic classifier, given
    - a test target word  $w$
    - its context window of  $L$  words  $c_{1:L}$
  - Estimates probability that  $w$  occurs in this window based on similarity of  $w$  (embeddings) to  $C_{1:L}$  (embeddings).
  - To compute this, we just need embeddings for all the words.
-

These embeddings we'll need: a set for  $w$ , a set for  $c$



## Skip-Gram Training data

...lemon, a [tablespoon of apricot jam, a]  
pinch...

c1                c2    **[target]**    c3    c4  
**positive examples +**

t                c

---

apricot tablespoon

apricot of

apricot jam

apricot a

# Skip-Gram Training data

...lemon, a [tablespoon of apricot jam, a]  
pinch...

c1

c2 [target] c3 c4

**positive examples +**

t c

---

apricot tablespoon

apricot of

apricot jam

apricot a

2 negative examples for each positive context words.

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

# Skip-Gram Training data

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

**positive examples +**  
 t            c  
 $\frac{P}{}$  (apricot tablespoon)

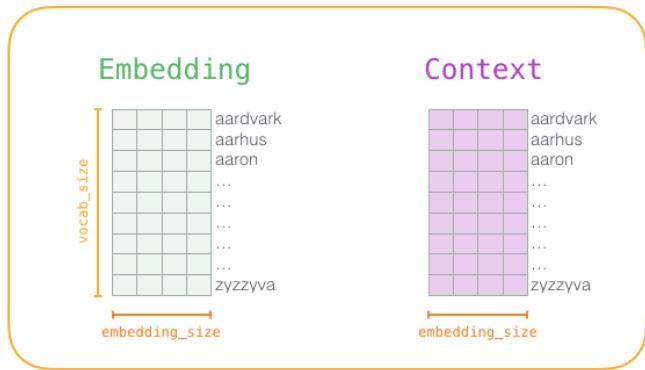
apricot of  
apricot jam  
apricot a

c1            c2    **[target]**    c3    c4  
**negative examples -**

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

# Skip-gram parameters

- SGNS learns two sets of embeddings
    - Target embeddings matrix  $W$
    - Context embedding matrix  $C$
  - It's common to just add them together, representing word  $i$  as the vector  $w_i + c_i$
  - Thus the parameters we need to learn are two matrices  $W$  and  $C$ , each containing an embedding for every one of the  $|V|$  words in the vocabulary  $|V|$



$\theta =$

1..d

aardvark      1

apricot

...

zebra

aardvark      |V|

apricot

...

zebra

|V|+1

2V

W      target words

C      context & noise words

# Word2vec training: how to learn vectors

---

- Given the set of positive and negative training instances, and an initial set of embedding vectors
- The goal of learning is to adjust those word vectors such that we:
  - **Maximize** the similarity of the **target word, context word** pairs ( $w, c_{pos}$ ) drawn from the positive data
  - **Minimize** the similarity of the ( $w, c_{neg}$ ) pairs drawn from the negative data.

# Log Loss function

$$\begin{aligned}
 L_{CE} &= -\log \left[ P(+|w, c_{pos}) \prod_{i=1}^k P(-|w, c_{neg_i}) \right] \quad \text{These calculation for all positive and negative words} \\
 &= - \left[ \log P(+|w, c_{pos}) + \sum_{i=1}^k \log P(-|w, c_{neg_i}) \right] \\
 &= - \left[ \log P(+|w, c_{pos}) + \sum_{i=1}^k \log (1 - P(+|w, c_{neg_i})) \right] \\
 &= - \left[ \log \sigma(c_{pos} \cdot w) + \sum_{i=1}^k \log \sigma(-c_{neg_i} \cdot w) \right]
 \end{aligned}$$

# Loss function for one $w$ with $c_{pos}, c_{neg1} \dots c_{negk}$

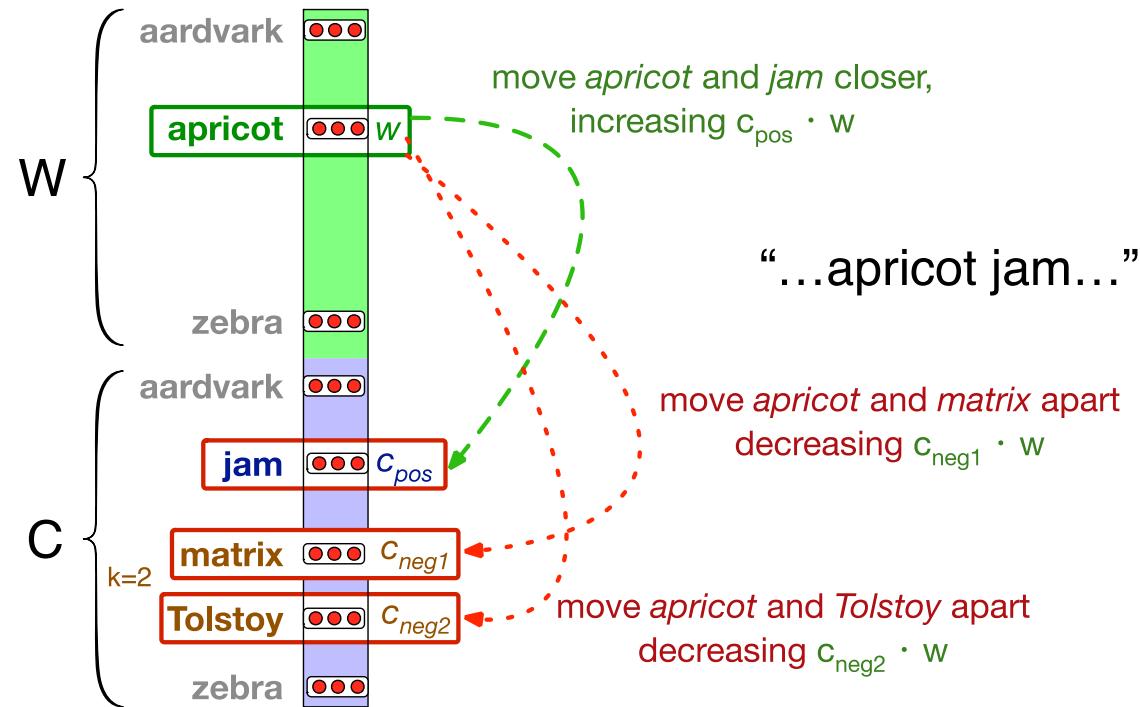
- Maximize the similarity of the target with the actual context words, and minimize the similarity of the target with the  $k$  negative sampled non-neighbor words.

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

# Learning the classifier

- How to learn?
  - Stochastic gradient descent!
- We'll adjust the word weights to
  - make the positive pairs more likely
  - and the negative pairs less likely,
  - over the entire training set.

# Intuition of one step of gradient descent



- Tries to shift embeddings
- So the target embeddings (here for apricot) are closer to (have a higher dot product with) context embeddings for nearby words (here jam)
- And further from (lower dot product with) context embeddings for noise words that don't occur nearby (here Tolstoy and matrix).

## Reminder: gradient descent

---

- At each step
  - Direction: We move in the reverse direction from the gradient of the loss function
  - Magnitude: we move the value of this gradient  $\frac{d}{dw} L(f(x; w), y)$  weighted by a **learning rate**  $\eta$
  - Higher learning rate means move  $w$  faster

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

# The derivatives of the loss function

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

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

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

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

# Update equation in SGD

Start with randomly initialized C and W matrices, then incrementally do updates

$$\left[ \begin{array}{l} c_{pos}^{t+1} = c_{pos}^t - \eta [\sigma(c_{pos}^t \cdot w^t) - 1] w^t \\ c_{neg}^{t+1} = c_{neg}^t - \eta [\sigma(c_{neg}^t \cdot w^t)] w^t \end{array} \right] \text{Context word embedding matrix}$$

$$\left[ \begin{array}{l} w^{t+1} = w^t - \eta \left[ [\sigma(c_{pos} \cdot w^t) - 1] c_{pos} + \sum_{i=1}^k [\sigma(c_{neg_i} \cdot w^t)] c_{neg_i} \right] \end{array} \right] \text{Target word embedding matrix}$$

## Two sets of embeddings

---

SGNS learns two sets of embeddings

Target embeddings matrix  $W$

Context embedding matrix  $C$

It's common to just add them together, representing word  $i$  as the vector  $w_i + c_i$

## Summary: How to learn word2vec (skip-gram) embeddings

---

- Start with  $V$  random  $d$ -dimensional vectors as initial embeddings
- Train a classifier based on embedding similarity
  - Take a corpus and take pairs of words that co-occur as positive examples
  - Take pairs of words that don't co-occur as negative examples
  - Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
  - Throw away the classifier code and keep the embeddings.

# The kinds of neighbors depend on window size

---

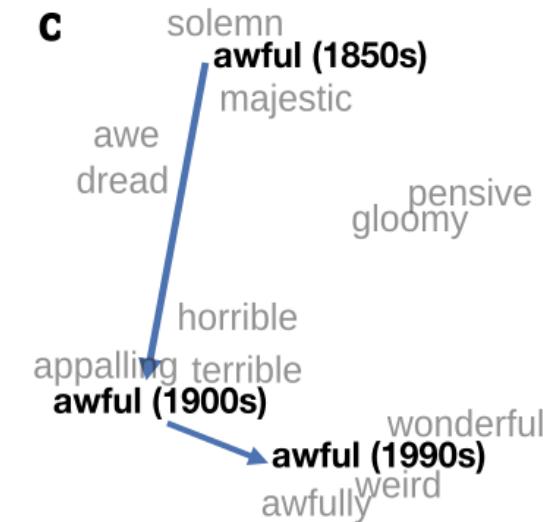
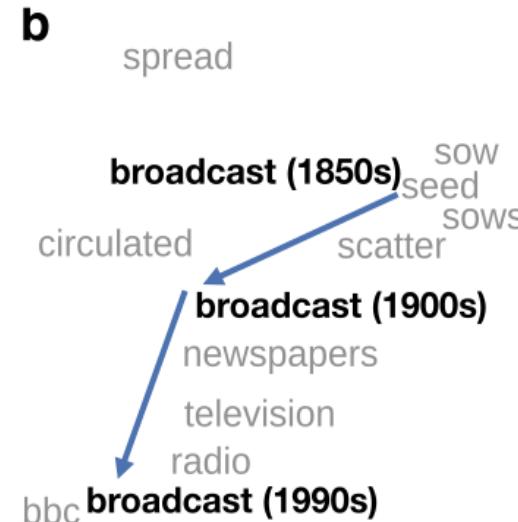
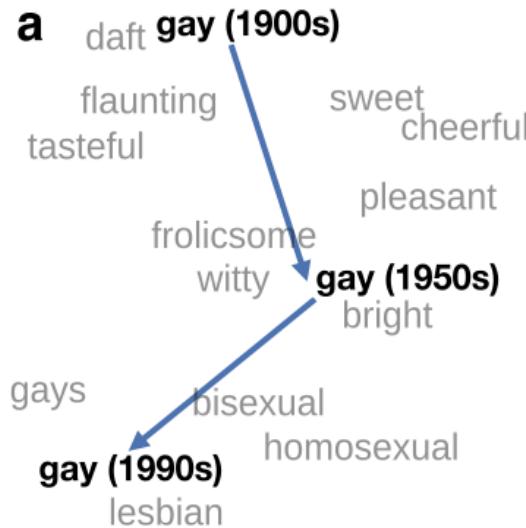
- **Small windows** ( $C = +/- 2$ ) : nearest words are syntactically similar words in same taxonomy
  - *Hogwarts* nearest neighbors are other fictional schools
    - *Sunnydale, Evernight, Blandings*
- **Large windows** ( $C = +/- 5$ ) : nearest words are related words in same semantic field
  - *Hogwarts* nearest neighbors are Harry Potter world:
    - *Dumbledore, half-blood, Malfoy*

# Embeddings as a window onto historical semantics



Train embeddings on different decades of historical text to see meanings shift

~30 million books, 1850-1990, Google Books data



# Embeddings reflect cultural bias!

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *NeurIPS*, pp. 4349-4357. 2016.

- Ask “Paris : France :: Tokyo : x”
  - x = Japan
- Ask “father : doctor :: mother : x”
  - x = nurse
- Ask “man : computer programmer :: woman : x”
  - x = homemaker

Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring

# Historical embedding as a tool to study cultural biases

innovate

achieve

lead

Garg, N., Schiebinger, L., Jurafsky, D., and Zou, J. (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences* 115(16), E3635–E3644.

- Compute a **gender or ethnic bias** for each adjective: e.g., how much closer the adjective is to "woman" synonyms than "man" synonyms, or names of particular ethnicities
  - Embeddings for **competence** adjective (*smart, wise, brilliant, resourceful, thoughtful, logical*) are biased toward men, a bias slowly decreasing 1960-1990
  - Embeddings for **dehumanizing** adjectives (barbaric, monstrous, bizarre) were biased toward Asians in the 1930s, bias decreasing over the 20<sup>th</sup> century.
- These match the results of old surveys done in the 1930s

# CBOW vs Skip-gram

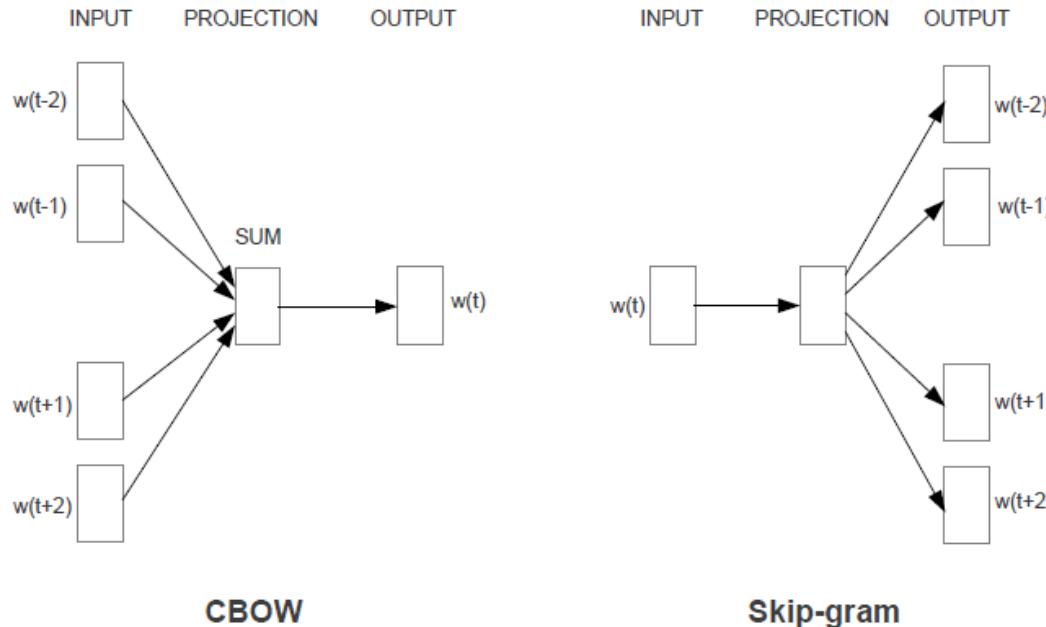
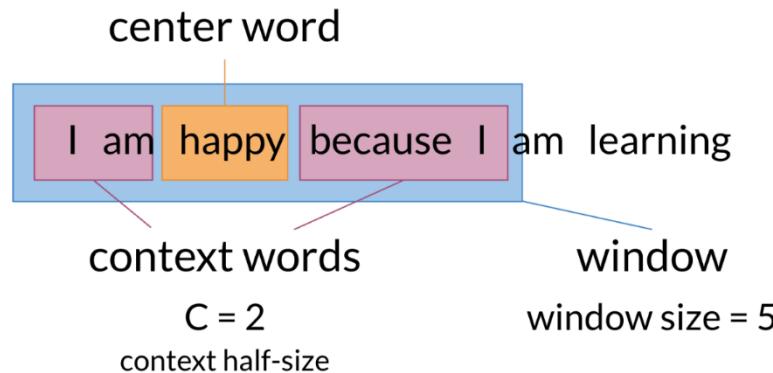
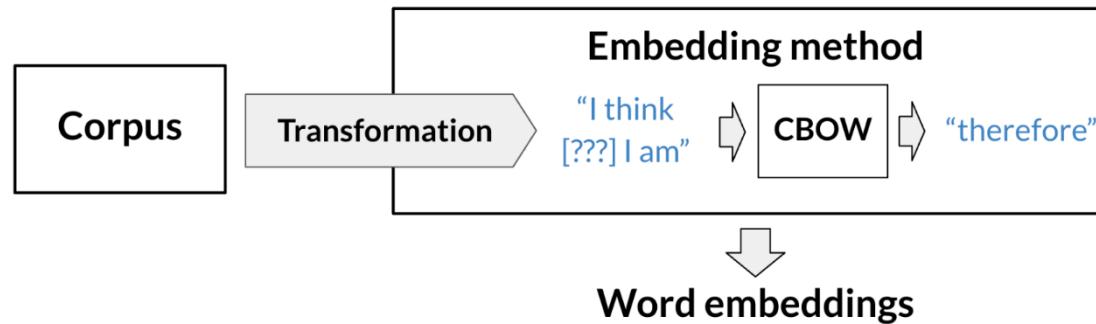
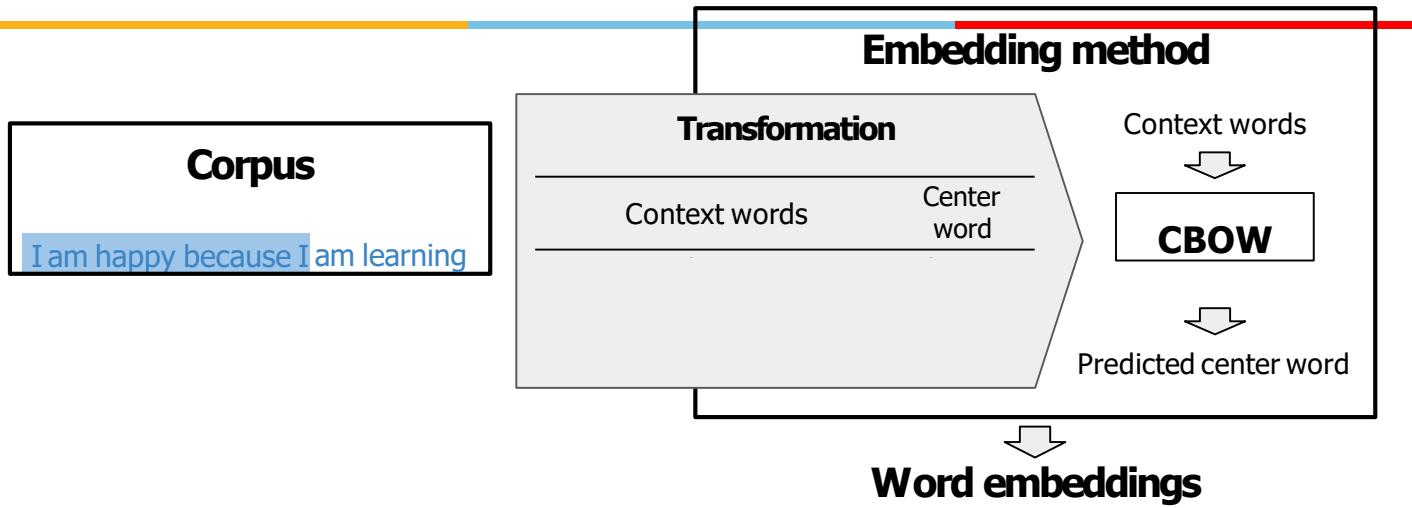


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

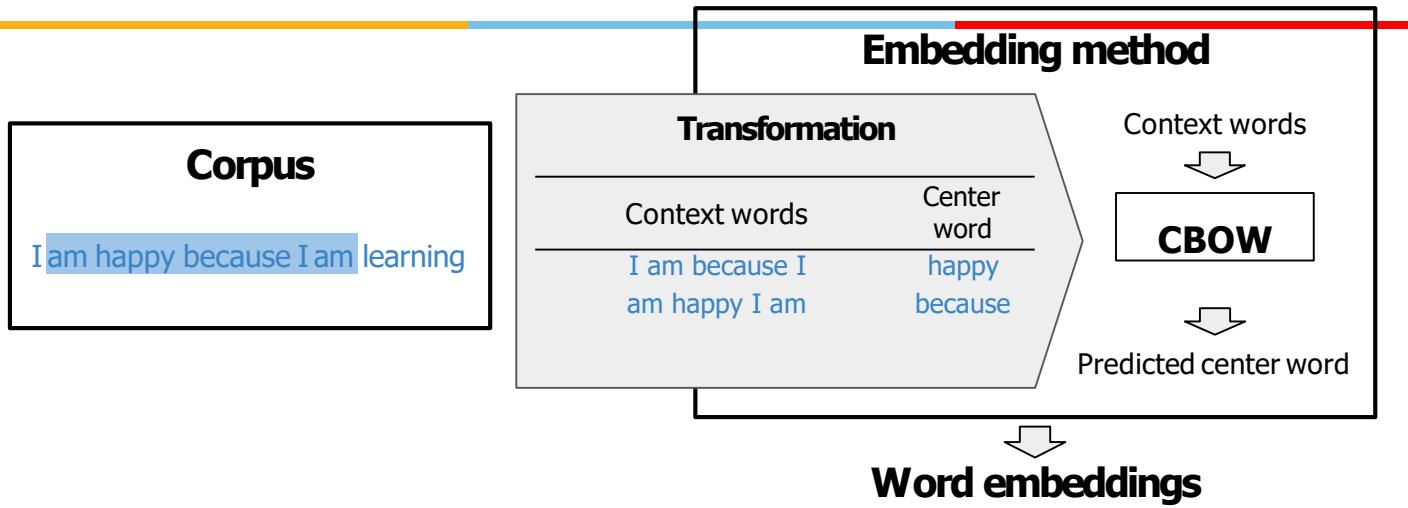
# CBOW



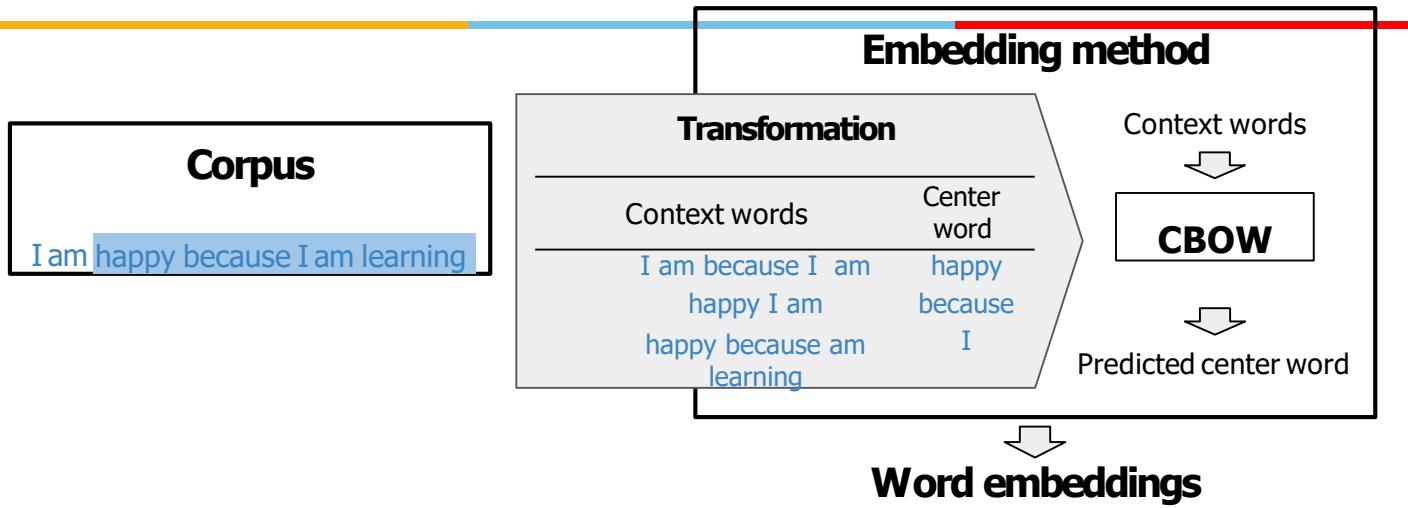
# From corpus to training



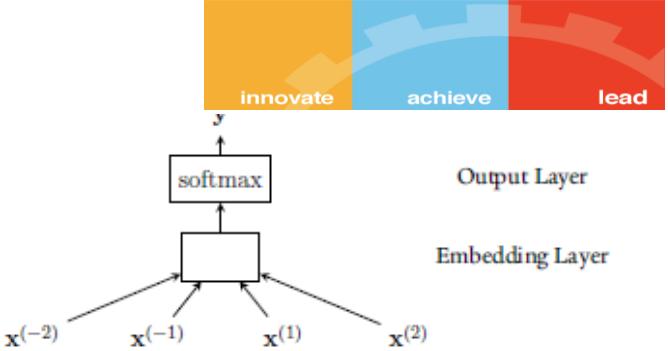
# From corpus to training



# From corpus to training

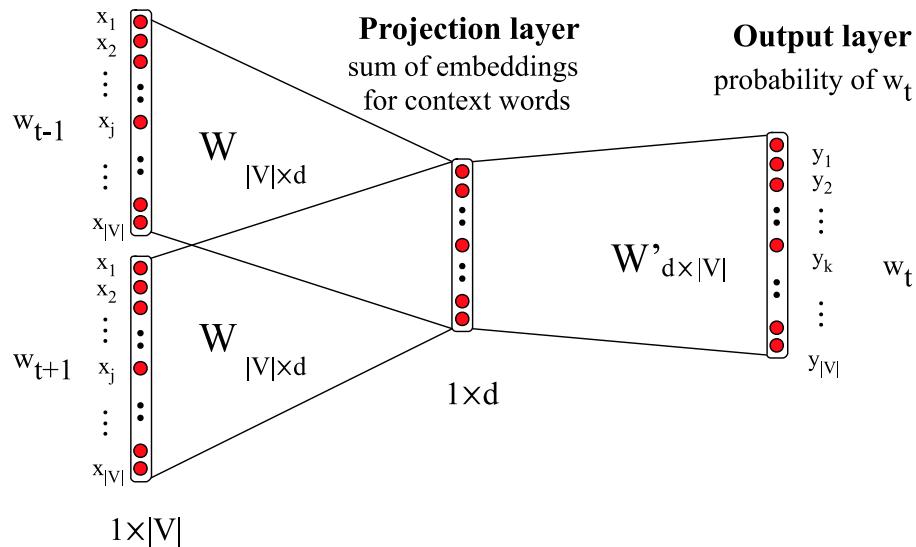


# CBOW (Continuous Bag of Words)

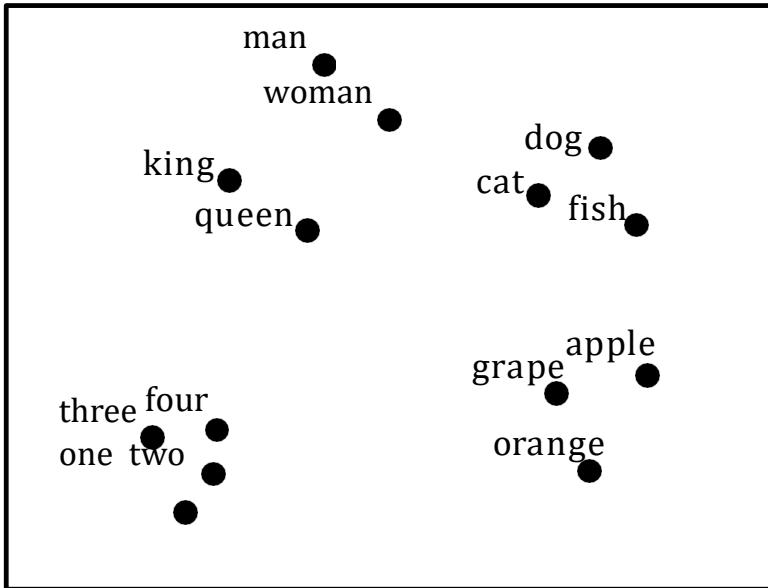


## Input layer

1-hot input vectors  
for each context word



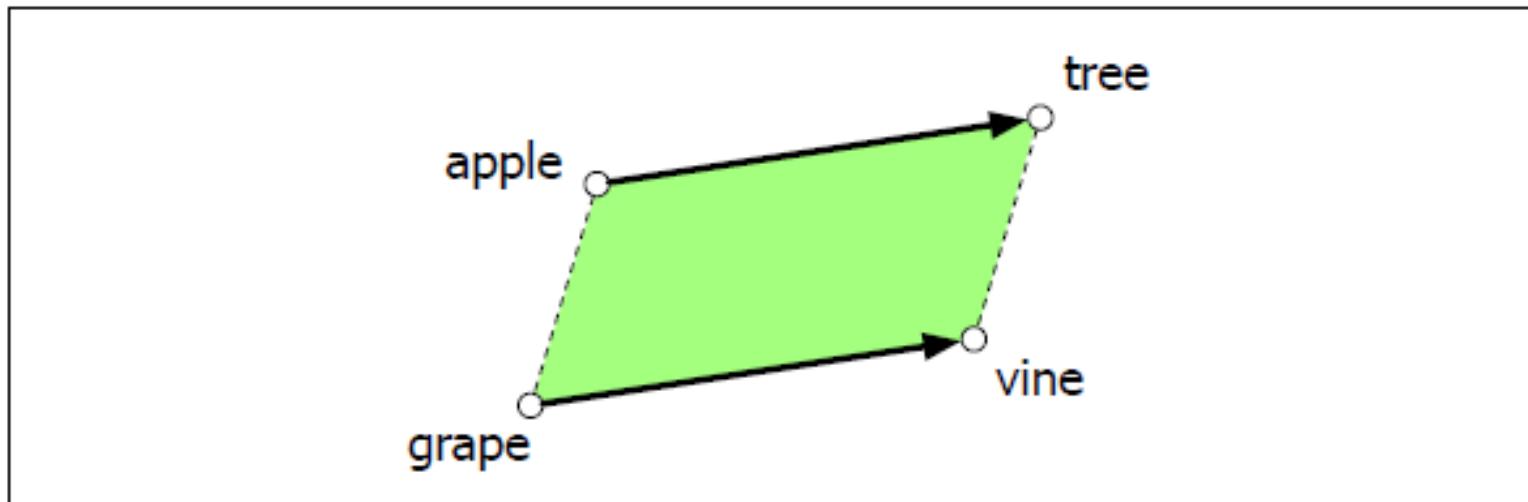
# Visualizing word embeddings



# Analogical relations

The classic parallelogram model of analogical reasoning

To solve: "*apple is to tree as grape is to \_\_\_\_*"



**Figure 6.15** The parallelogram model for analogy problems (Rumelhart and Abrahamson, 1973): the location of  $\overrightarrow{\text{vine}}$  can be found by subtracting  $\overrightarrow{\text{apple}}$  from  $\overrightarrow{\text{tree}}$  and adding  $\overrightarrow{\text{grape}}$ .

# Analogical relations via parallelogram

---

- The parallelogram method can solve analogies with both sparse and dense embeddings (Turney and Littman 2005, Mikolov et al. 2013b)

king – man + woman is close to queen

Paris – France + Italy is close to Rome

- For a problem  $a:a^*:b:b^*$ , the parallelogram method is:

$$\hat{b}^* = \operatorname*{argmax}_x \text{distance}(x, a^* - a + b)$$

# GloVe

---

- GloVe stands for global vectors for word representation.
  - Unsupervised learning algorithm developed by Stanford for generating word embeddings
  - GloVe captures both global statistics and local statistics of a corpus, in order to come up with word vectors
  - Aggregates global word-word co-occurrence matrix from a corpus
  - Idea is learning should be with ratios of co-occurrence probabilities rather than the probabilities themselves.
  - The advantage of GloVe is that, unlike Word2vec, GloVe does not rely just on local statistics (local context information of words), but incorporates global statistics (word co-occurrence) to obtain word vectors.
-

# Matrix of word-word co-occurrence counts

- I like deep learning.
- I like NLP.
- I enjoy flying.

| counts   | I | like | enjoy | deep | learning | NLP | flying | . |
|----------|---|------|-------|------|----------|-----|--------|---|
| I        | 0 | 2    | 1     | 0    | 0        | 0   | 0      | 0 |
| like     | 2 | 0    | 0     | 1    | 0        | 1   | 0      | 0 |
| enjoy    | 1 | 0    | 0     | 0    | 0        | 0   | 1      | 0 |
| deep     | 0 | 1    | 0     | 0    | 1        | 0   | 0      | 0 |
| learning | 0 | 0    | 0     | 1    | 0        | 0   | 0      | 1 |
| NLP      | 0 | 1    | 0     | 0    | 0        | 0   | 0      | 1 |
| flying   | 0 | 0    | 1     | 0    | 0        | 0   | 0      | 1 |
| .        | 0 | 0    | 0     | 0    | 1        | 1   | 1      | 0 |

- let the matrix of word-word co-occurrence counts be denoted by  $X$ ,
- whose entries  $X_{ij}$  tabulate the number of times word  $j$  occurs in the context of word  $i$
- let  $X_i = \sum_k X_{ik}$  be the number of times any word appears in the context of word  $i$
- let  $P_{ij} = P(i|j) = X_{ij}/X_i$  be the probability that word  $j$  appear in the context of word  $i$

# Intuition behind GloVe

*ratio of conditional probabilities represents the word meanings*

| Probability and Ratio               | $k = \text{solid}$   | $k = \text{gas}$     | $k = \text{water}$   | $k = \text{fashion}$ |
|-------------------------------------|----------------------|----------------------|----------------------|----------------------|
| $P(k \text{ice})$                   | $1.9 \times 10^{-4}$ | $6.6 \times 10^{-5}$ | $3.0 \times 10^{-3}$ | $1.7 \times 10^{-5}$ |
| $P(k \text{steam})$                 | $2.2 \times 10^{-5}$ | $7.8 \times 10^{-4}$ | $2.2 \times 10^{-3}$ | $1.8 \times 10^{-5}$ |
| $P(k \text{ice})/P(k \text{steam})$ | 8.9                  | $8.5 \times 10^{-2}$ | 1.36                 | 0.96                 |

Very small or large:  
solid is related to ice but not steam, or  
gas is related to steam but not ice

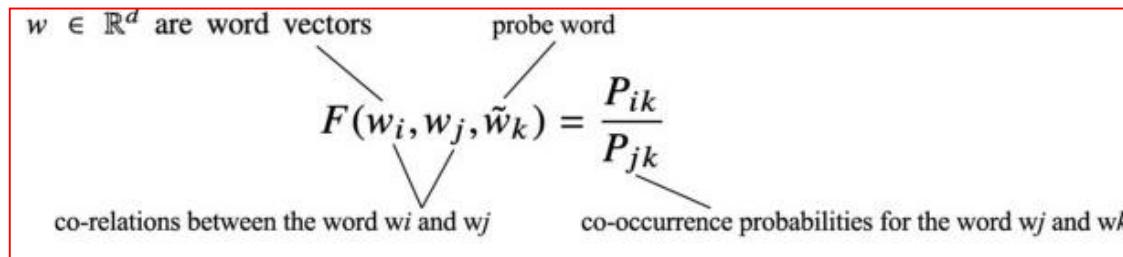
close to 1:  
water is highly related to ice and steam, or  
fashion is not related to ice or steam.

- Given a probe word, the ratio can be small, large or equal to 1 depends on their correlations.
- For example, if the ratio is large, the probe word is related to  $w_i$  but not  $w_j$ .
- This ratio gives us hints on the relations between three different words

# Encoding meaning components in vector differences

- $P_i \rightarrow$  Probability of ice,  $P_j \rightarrow$  Probability of steam
- $k =$  the “probe word”
- $P_{ik} =$  probability of seeing word  $i$  and  $k$  together,
- $P_{ik}, P_{jk} \rightarrow$  derived from the corpus
- aim is to ‘learn’ the function F to encode the information  $P_{ik}/P_{jk}$  available in the global corpus.**

|                       | Probability and Ratio | $k = solid$          | $k = gas$            | $k = water$          | $k = fashion$ |
|-----------------------|-----------------------|----------------------|----------------------|----------------------|---------------|
| $P(k ice)$            | $1.9 \times 10^{-4}$  | $6.6 \times 10^{-5}$ | $3.0 \times 10^{-3}$ | $1.7 \times 10^{-5}$ |               |
| $P(k steam)$          | $2.2 \times 10^{-5}$  | $7.8 \times 10^{-4}$ | $2.2 \times 10^{-3}$ | $1.8 \times 10^{-5}$ |               |
| $P(k ice)/P(k steam)$ | 8.9                   | $8.5 \times 10^{-2}$ | 1.36                 | 0.96                 |               |



$w_i^T \tilde{w}_k$  relate to (high probability if they are similar)

$$F((w_i - w_j)^T \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

$w_j^T \tilde{w}_k$

# GloVe prediction

$$F(w_i^T \tilde{w}_k) = P_{ik} = \frac{X_{ik}}{X_i}$$

Use of Log-bilinear model:

$$\rightarrow w_i \cdot w_j = \log P(i|j)$$

$$w_i^T \tilde{w}_k = \log(P_{ik}) = \log(X_{ik}) - \log(X_i)$$

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log(X_{ik})$$

co-occurrence count for word  $w_i$  and  $w_k$

- We can absorb  $\log(X_i)$  as a constant bias term since it is invariant of  $k$ .
- to maintain the symmetrical requirement between  $i$  and  $k$ , we will split it into two bias terms above.
- **This  $w$  and  $b$  form the embedding matrix. Therefore, the dot product of two embedding matrices predicts the log co-occurrence count.**

# GloVe objective function

---

- GloVe uses a Global ***Log-Bilinear (LBL) Regression Model*** that uses a simple Weighted Least Squares method.
- the Mean Square Error to calculate the error in the ground truth and the predicted co-occurrence counts.
- But since word pair have different occurrence frequency in the corpus, we need a weight to readjust the cost for each word pair.
- This is the function  $f$  below. When the co-occurrence count is higher or equal a threshold, say 100, the weight will be 1. Otherwise, the weight will be smaller, subject to the co-occurrence count

$$J = \sum_{i,j=1}^V f(X_{ij}) (w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

---

# References

---

- Ch-6 : Speech and Language Processing Daniel Jurafsky and James H. Martin
  - <https://jalammar.github.io/illustrated-word2vec/>
  - <http://web.stanford.edu/class/cs224n/slides/cs224n-2022-lecture02-wordvecs2.pdf>
  - <https://arxiv.org/pdf/1301.3781.pdf>
  - <https://nlp.stanford.edu/pubs/glove.pdf>
  - <https://jonathan-hui.medium.com/nlp-word-embedding-glove-5e7f523999f6>
  - [www.deeplearning.ai](http://www.deeplearning.ai)
-

# References

## Word embedding

- <https://www.youtube.com/watch?v=ERibwqs9p38>
- <https://www.youtube.com/watch?v=EsfNYiLVtHI&t=10s>
- <https://www.youtube.com/watch?v=lrPxo-92GC0>
- <https://www.coursera.org/lecture/nlp-sequence-models/>
- <https://www.youtube.com/watch?v=UqRCEmrV1gQ>
- <https://www.youtube.com/watch?v=g7wEfamFOEg>
- <https://www.youtube.com/watch?v=Sho-b4-9ODE>



**BITS** Pilani  
Pilani Campus

# Natural Language Processing

## DSECL ZG565

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



Grateful Acknowledgment: Source of this slide deck

**Jurafsky and Martin:** <https://web.stanford.edu/~jurafsky/slp3/>

[https://web.stanford.edu/~jurafsky/slp3/slides/7\\_NN\\_Apr\\_28\\_2021.pptx](https://web.stanford.edu/~jurafsky/slp3/slides/7_NN_Apr_28_2021.pptx)

# Session Content

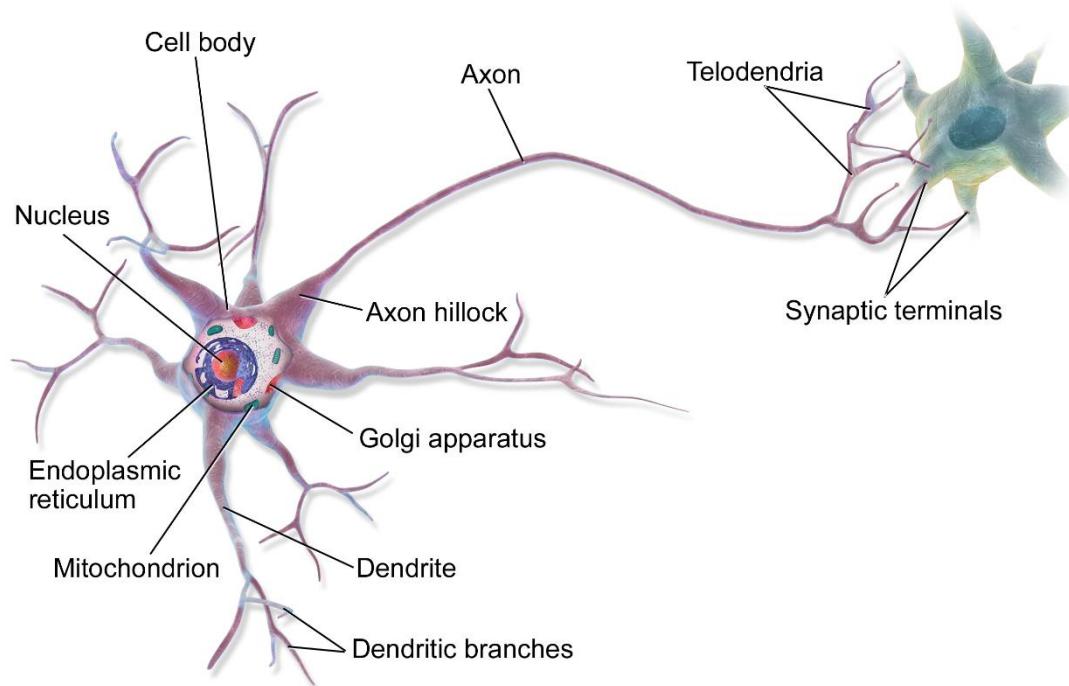
---

## Neural Networks and Neural Language Models

(Chapter 7 Jurafsky and Martin Book)

- Units in Neural Networks
- The XOR problem
- Feedforward Neural Networks
- Applying Feedforward Neural Networks to NLP tasks
- Training Neural Networks: Overview
- Computation Graphs & Backward Differentiation
- Neural Language Models

# This is in your brain



By BruceBlaus - Own work, CC BY 3.0,  
<https://commons.wikimedia.org/w/index.php?curid=28761830>

# Neural Network Unit

This is not in your brain

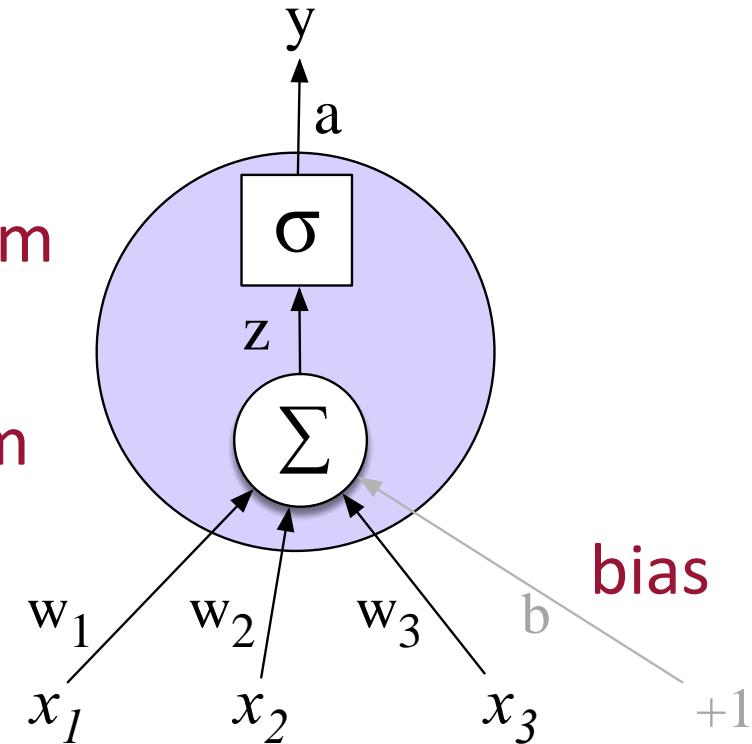
Output value

Non-linear transform

Weighted sum

Weights

Input layer



# Neural unit

---

- Take weighted sum of inputs, plus a bias

$$z = b + \sum_i w_i x_i$$

$$z = w \cdot x + b$$

- Instead of just using  $z$ , we'll apply a nonlinear activation function  $f$ :

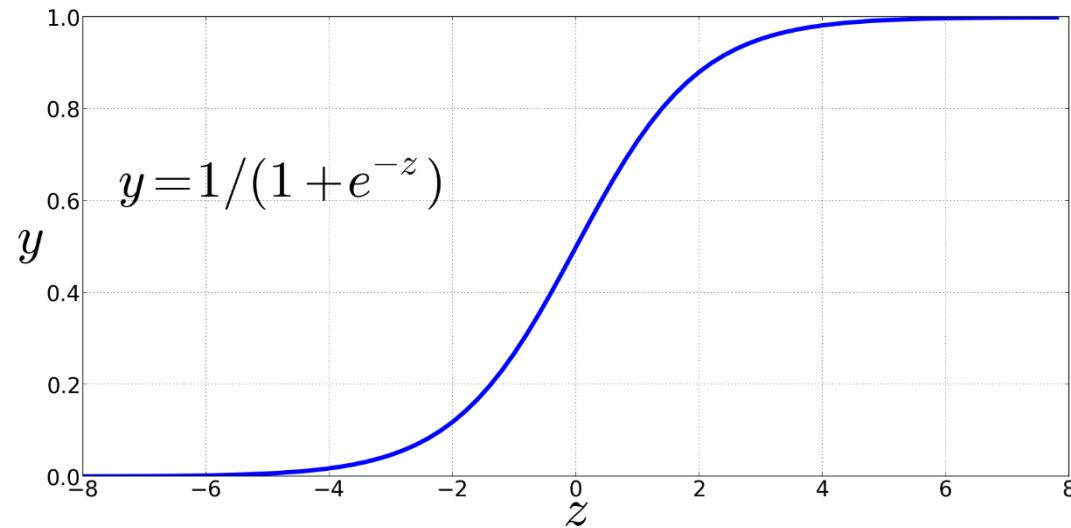
$$y = a = f(z)$$

# Non-Linear Activation Functions

We've already seen the sigmoid for logistic regression:

Sigmoid

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



# Final function the unit is computing

---

$$y = s(w \cdot x + b) = \frac{1}{1 + \exp(- (w \cdot x + b))}$$

# Final unit again

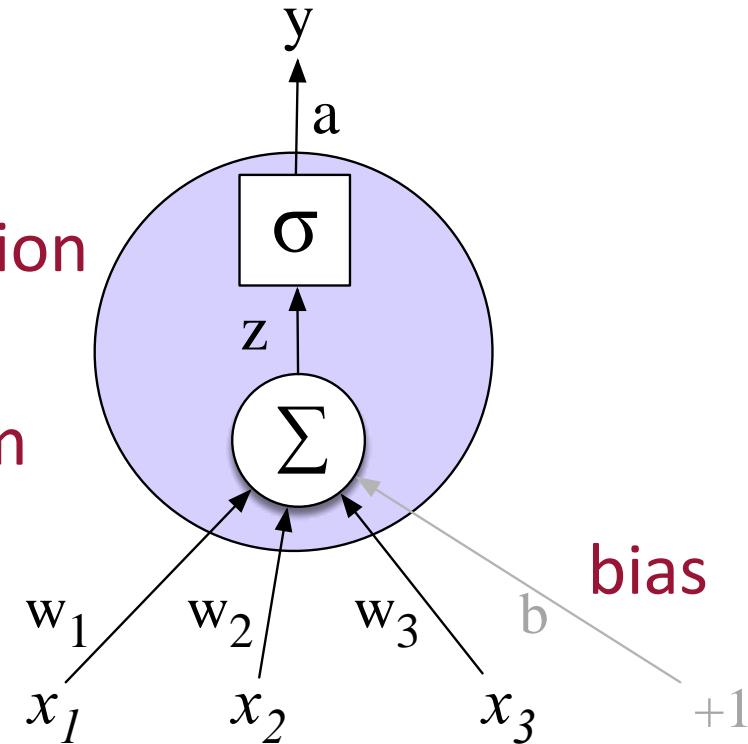
Output value

Non-linear activation function

Weighted sum

Weights

Input layer



# An example

- Suppose a unit has:
- $w = [0.2, 0.3, 0.9]$
- $b = 0.5$
- What happens with input  $x$ :
- $x = [0.5, 0.6, 0.1]$

$$y = s(w \cdot x + b) =$$

# An example

- Suppose a unit has:
  - $w = [0.2, 0.3, 0.9]$
  - $b = 0.5$
- What happens with the following input  $x$ ?
- $x = [0.5, 0.6, 0.1]$

$$y = s(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}} =$$

# An example

- Suppose a unit has:
- $w = [0.2, 0.3, 0.9]$
- $b = 0.5$
- What happens with input  $x$ :
- $x = [0.5, 0.6, 0.1]$

$$y = s(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}} =$$
$$\frac{1}{1 + e^{-(.5 \cdot 2 + .6 \cdot 3 + .1 \cdot 9 + .5)}} =$$

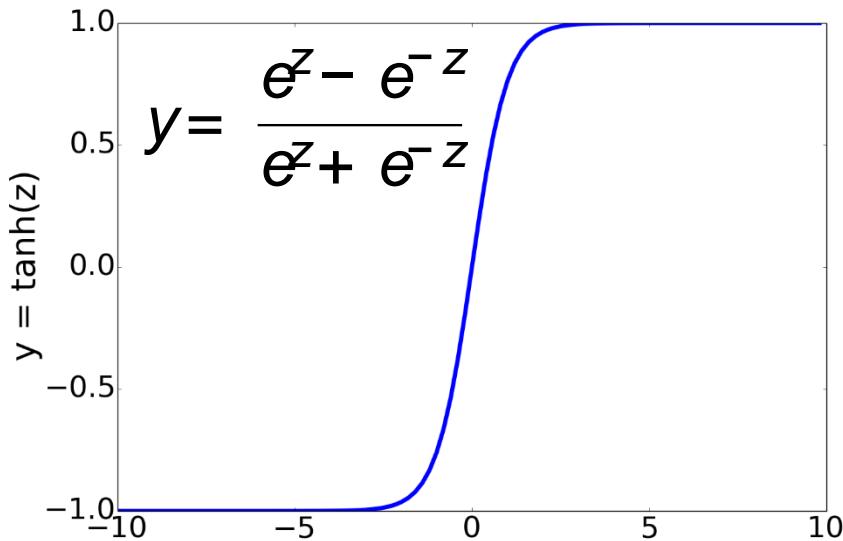
# An example

- Suppose a unit has:
- $w = [0.2, 0.3, 0.9]$
- $b = 0.5$
- What happens with input  $x$ :
- $x = [0.5, 0.6, 0.1]$

$$y = s(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}} =$$

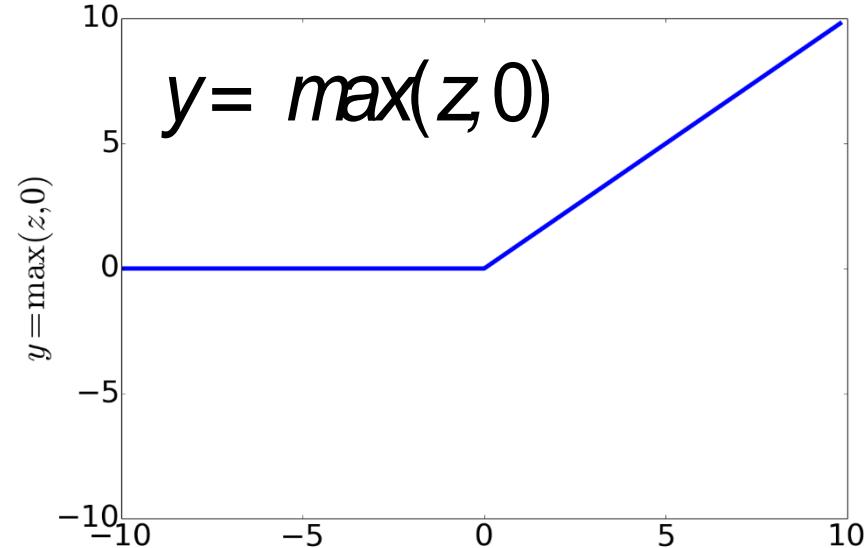
$$\frac{1}{1 + e^{-(.5 \cdot 2 + .6 \cdot 3 + .1 \cdot 9 + .5)}} = \frac{1}{1 + e^{-0.87}} = .70$$

# Non-Linear Activation Functions besides sigmoid



tanh

Most Common:



ReLU  
Rectified Linear Unit

# The XOR problem

Minsky and Papert (1969)

- Can neural units compute simple functions of input?

| AND |    | OR |    | XOR |   |
|-----|----|----|----|-----|---|
| x1  | x2 | y  | x1 | x2  | y |
| 0   | 0  | 0  | 0  | 0   | 0 |
| 0   | 1  | 0  | 0  | 1   | 1 |
| 1   | 0  | 0  | 1  | 0   | 1 |
| 1   | 1  | 1  | 1  | 1   | 0 |

# Perceptrons

---

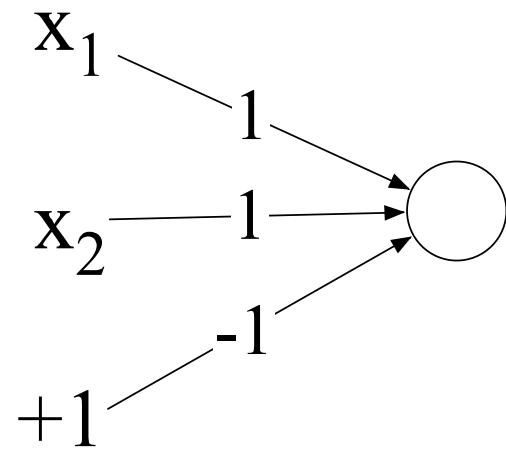
- A very simple neural unit
- Binary output (0 or 1)
- No non-linear activation function

$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$

---

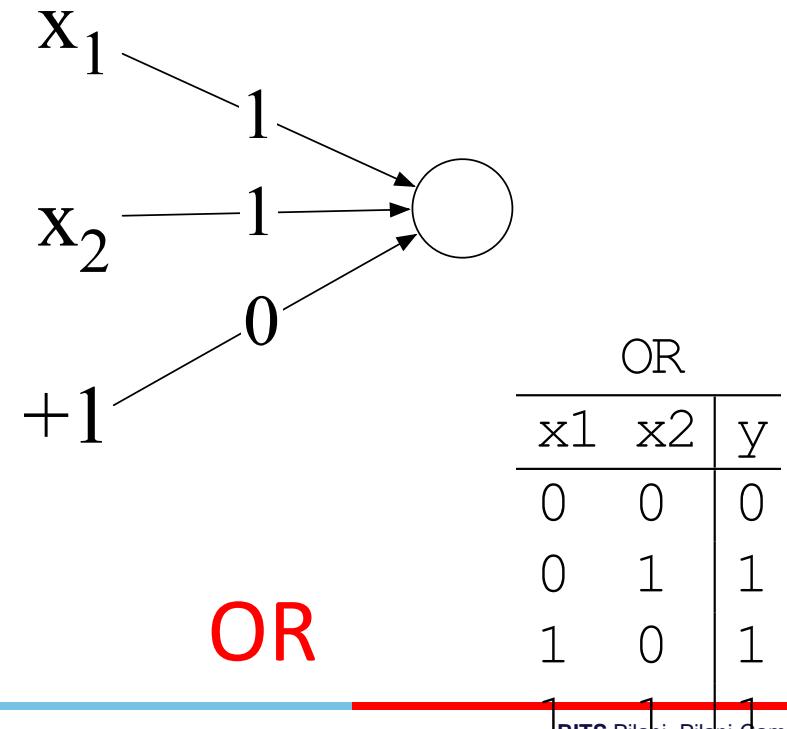
# Easy to build AND or OR with perceptrons

$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$



AND

|    |    | AND |
|----|----|-----|
| x1 | x2 | y   |
| 0  | 0  | 0   |
| 0  | 1  | 0   |
| 1  | 0  | 0   |
| 1  | 1  | 1   |

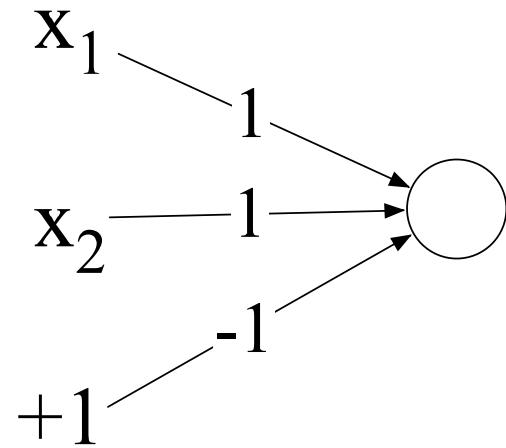


OR

|    |    | OR |
|----|----|----|
| x1 | x2 | y  |
| 0  | 0  | 0  |
| 0  | 1  | 1  |
| 1  | 0  | 1  |
| 1  | 1  | 1  |

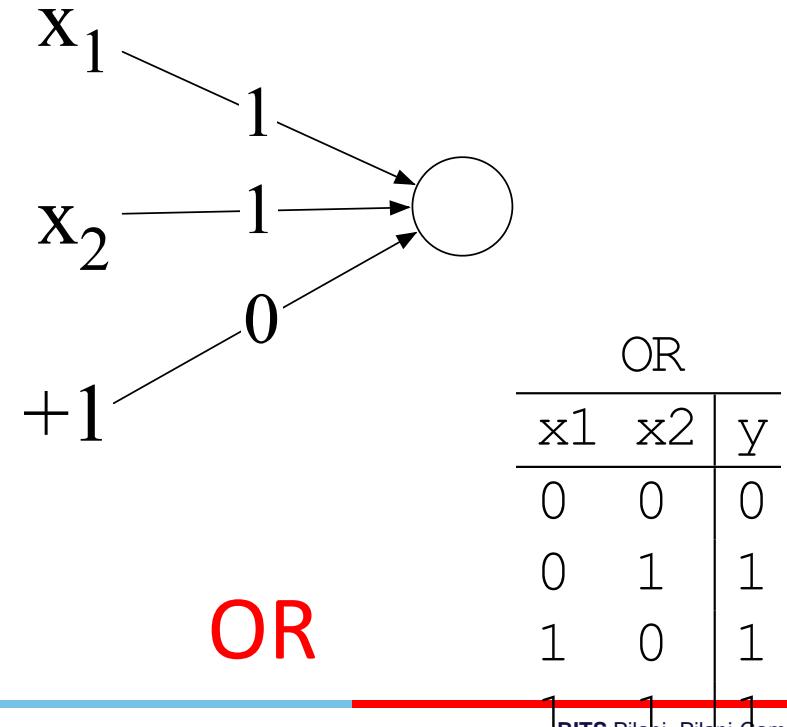
# Easy to build AND or OR with perceptrons

$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$



AND

|    |    | AND |
|----|----|-----|
| x1 | x2 | y   |
| 0  | 0  | 0   |
| 0  | 1  | 0   |
| 1  | 0  | 0   |
| 1  | 1  | 1   |

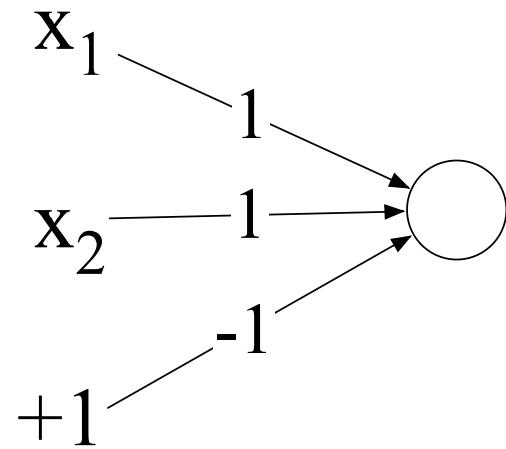


OR

|    |    | OR |
|----|----|----|
| x1 | x2 | y  |
| 0  | 0  | 0  |
| 0  | 1  | 1  |
| 1  | 0  | 1  |
| 1  | 1  | 1  |

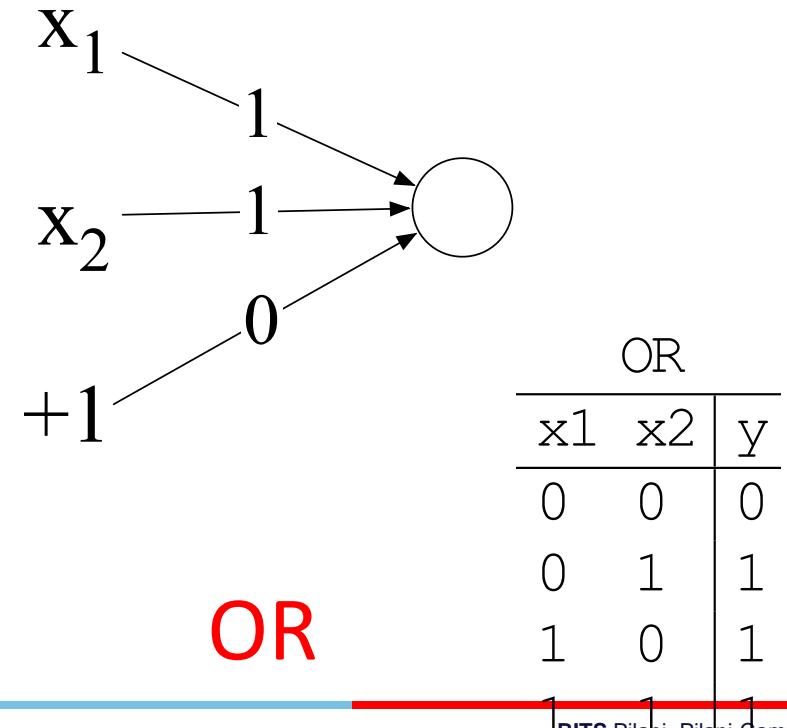
# Easy to build AND or OR with perceptrons

$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$



AND

|    |    | AND |
|----|----|-----|
| x1 | x2 | y   |
| 0  | 0  | 0   |
| 0  | 1  | 0   |
| 1  | 0  | 0   |
| 1  | 1  | 1   |



OR

|    |    | OR |
|----|----|----|
| x1 | x2 | y  |
| 0  | 0  | 0  |
| 0  | 1  | 1  |
| 1  | 0  | 1  |
| 1  | 1  | 1  |

# Not possible to capture XOR with perceptrons

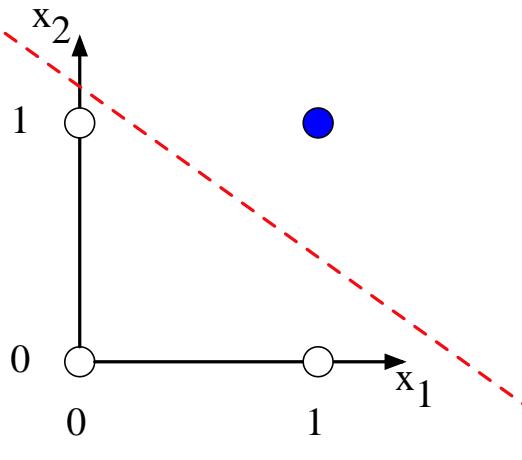
---

- Pause the lecture and try for yourself!

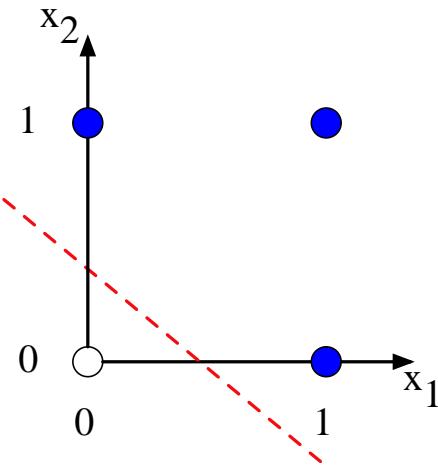
# Why? Perceptrons are linear classifiers

- Perceptron equation given  $x_1$  and  $x_2$ , is the equation of a line
- $w_1x_1 + w_2x_2 + b = 0$
- (in standard linear format:  $x_2 = (-w_1/w_2)x_1 + (-b/w_2)$  )
- This line acts as a **decision boundary**
  - 0 if input is on one side of the line
  - 1 if on the other side of the line

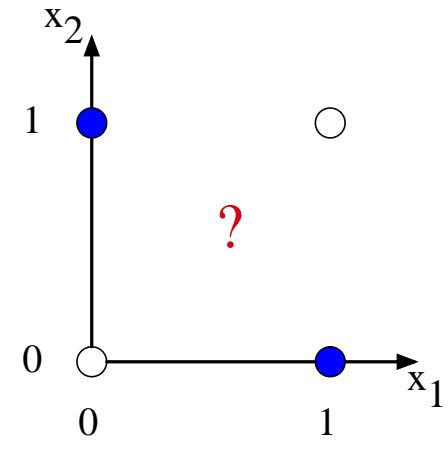
# Decision boundaries



a)  $x_1 \text{ AND } x_2$



b)  $x_1 \text{ OR } x_2$



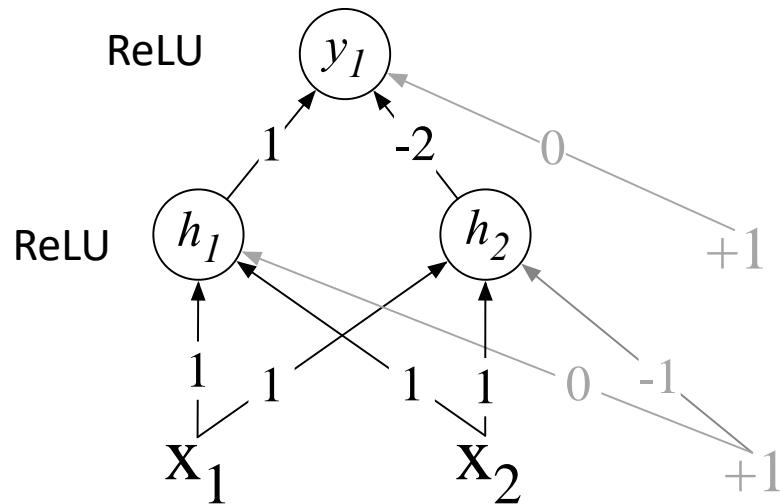
c)  $x_1 \text{ XOR } x_2$

XOR is not a **linearly separable** function!

# Solution to the XOR problem

- XOR **can't** be calculated by a single perceptron
- XOR **can** be calculated by a layered network of units.

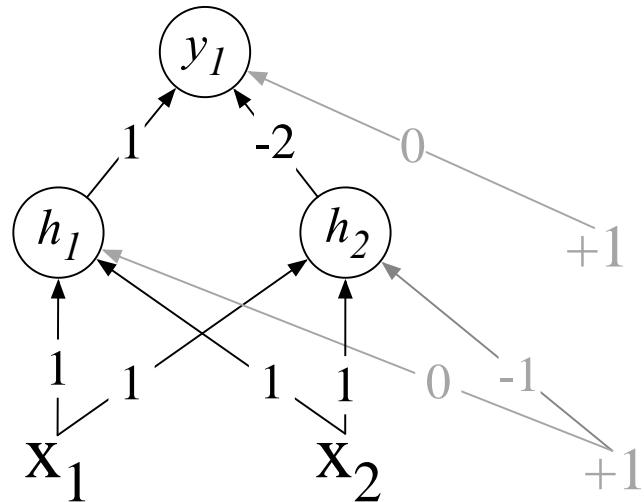
| XOR |    |   |
|-----|----|---|
| x1  | x2 | y |
| 0   | 0  | 0 |
| 0   | 1  | 1 |
| 1   | 0  | 1 |
| 1   | 1  | 0 |



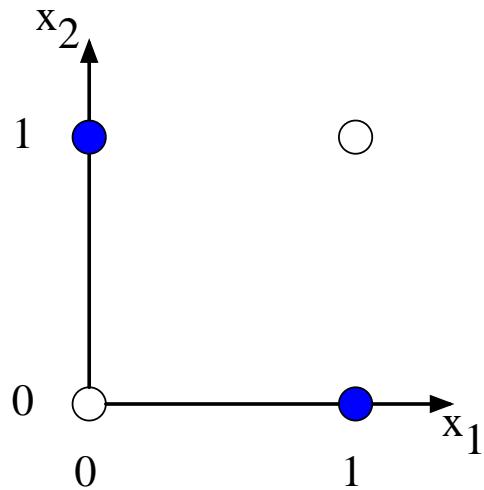
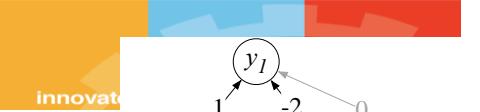
# Solution to the XOR problem

- XOR **can't** be calculated by a single perceptron
- XOR **can** be calculated by a layered network of units.

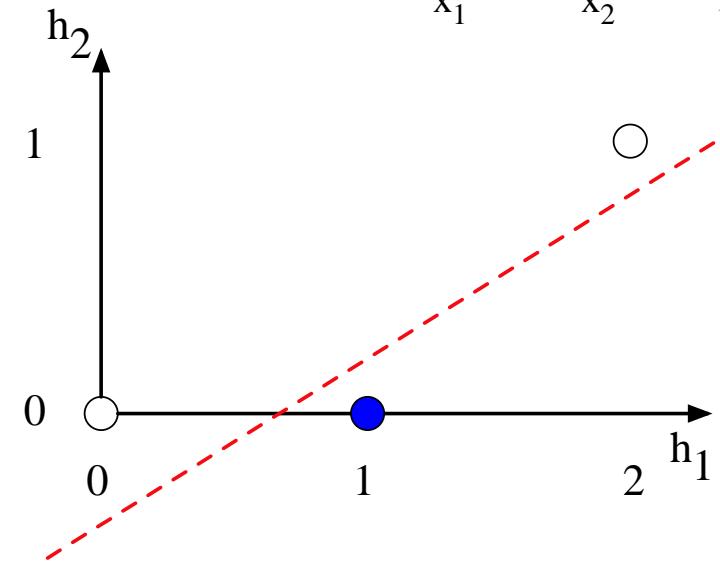
| XOR |    |   |
|-----|----|---|
| x1  | x2 | y |
| 0   | 0  | 0 |
| 0   | 1  | 1 |
| 1   | 0  | 1 |
| 1   | 1  | 0 |



# The hidden representation $h$



a) The original  $x$  space

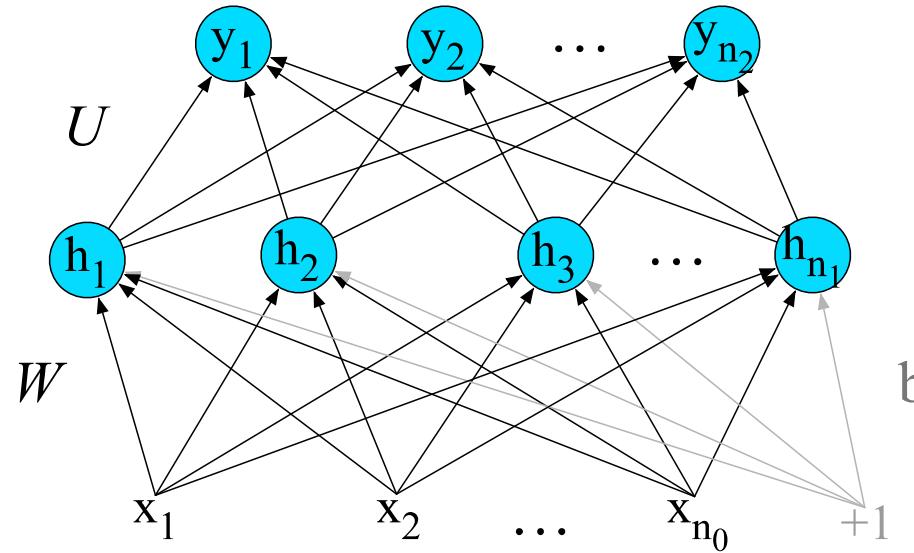


b) The new (linearly separable)  $h$  space

(With learning: hidden layers will learn to form useful representations)

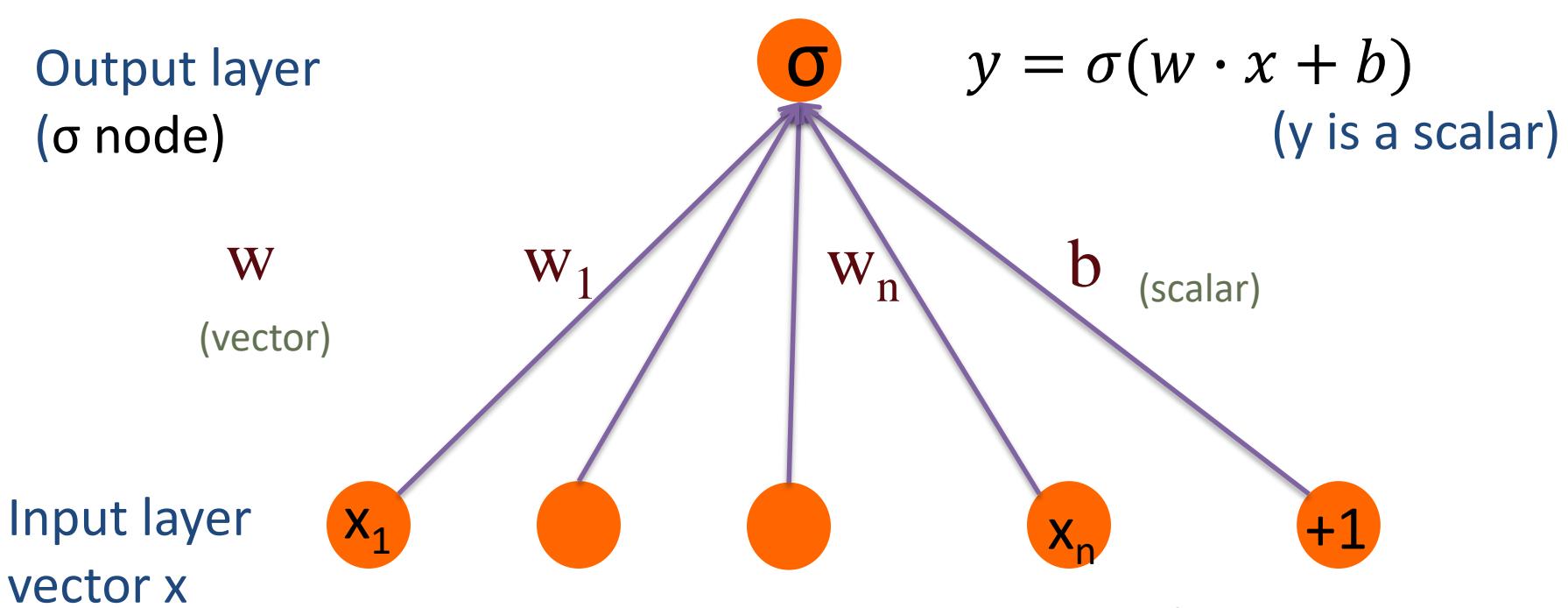
# Feedforward Neural Networks

- Can also be called **multi-layer perceptrons** (or **MLPs**) for historical reasons

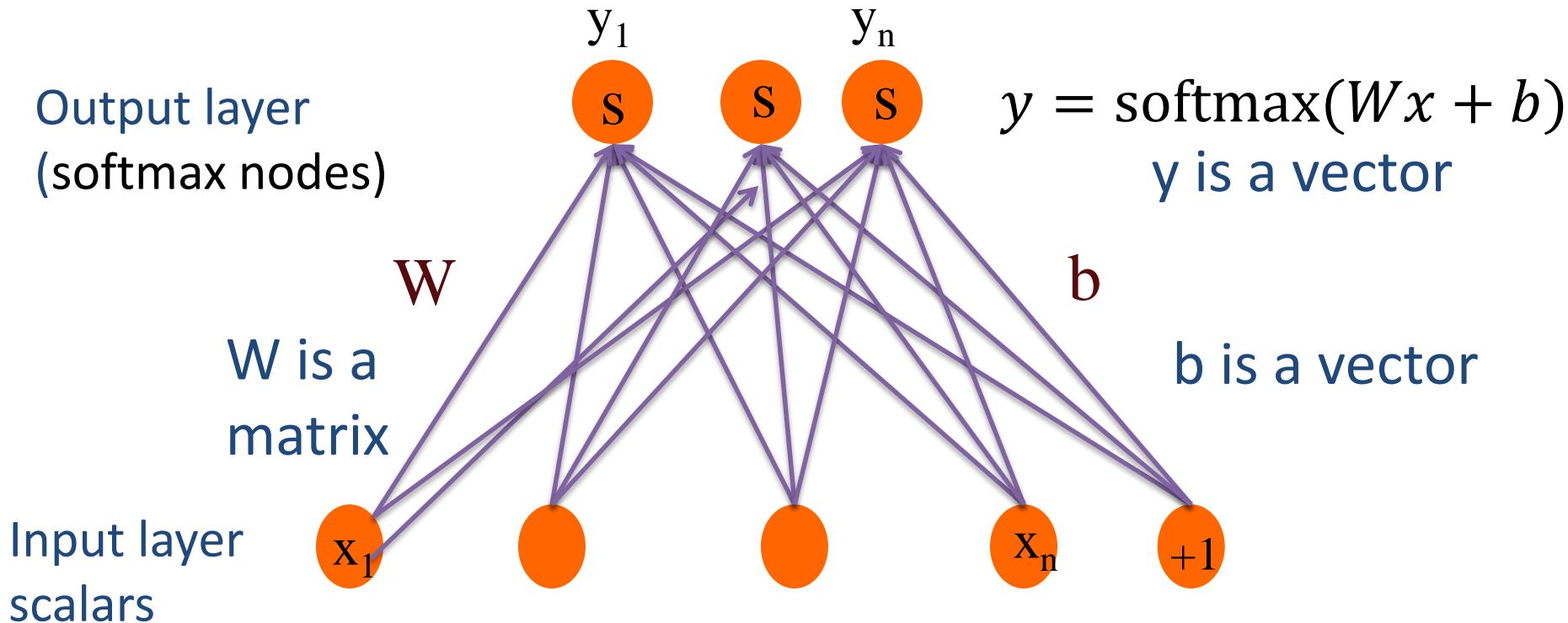


# Binary Logistic Regression as a 1-layer Network

(we don't count the input layer in counting layers!)



# Fully connected single layer network



# softmax: a generalization of sigmoid

---

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

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

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

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

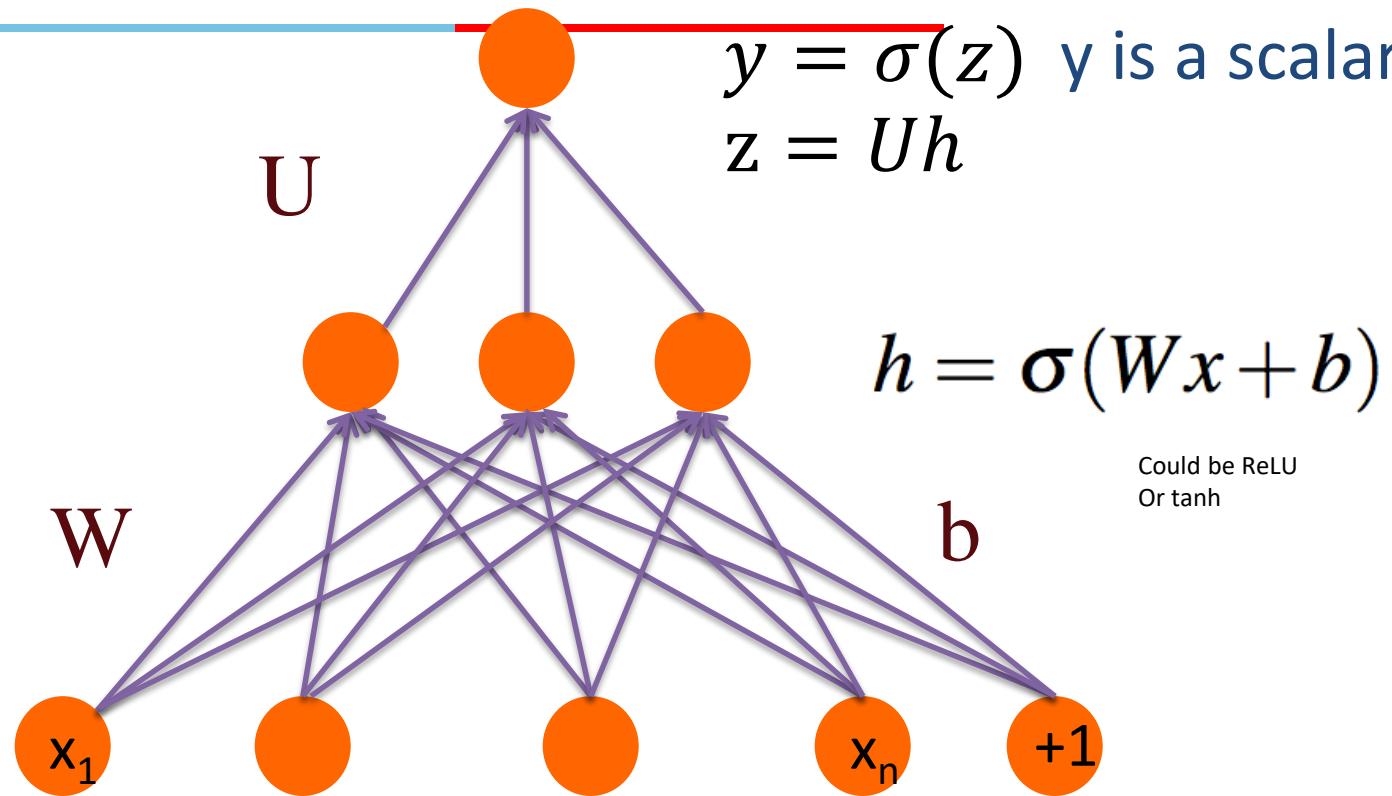
$$\text{softmax}(z) = [0.055, 0.090, 0.006, 0.099, 0.74, 0.010]$$

# Two-Layer Network with scalar output

Output layer  
( $\sigma$  node)

hidden units  
( $\sigma$  node)

Input layer  
(vector)

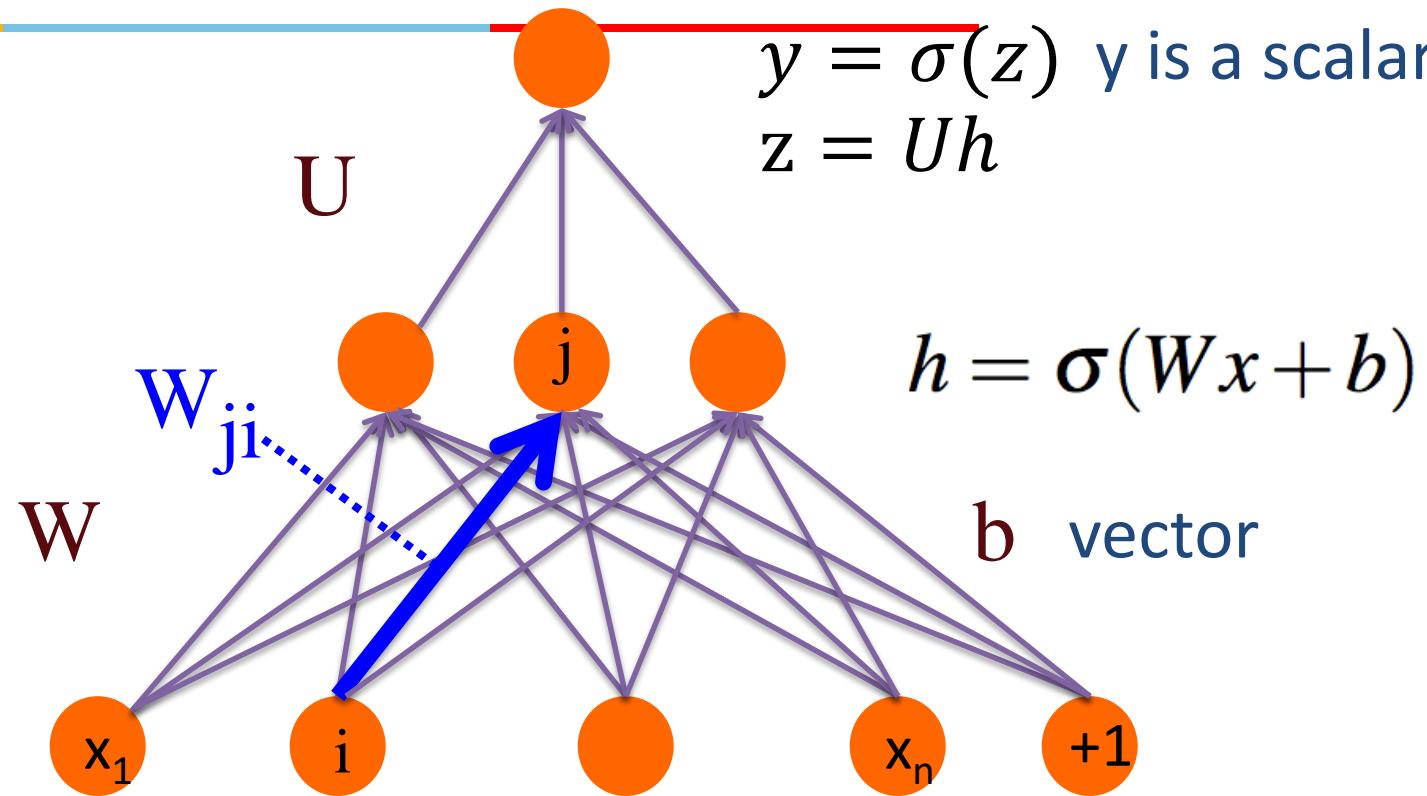


# Two-Layer Network with scalar output

Output layer  
( $\sigma$  node)

hidden units  
( $\sigma$  node)

Input layer  
(vector)

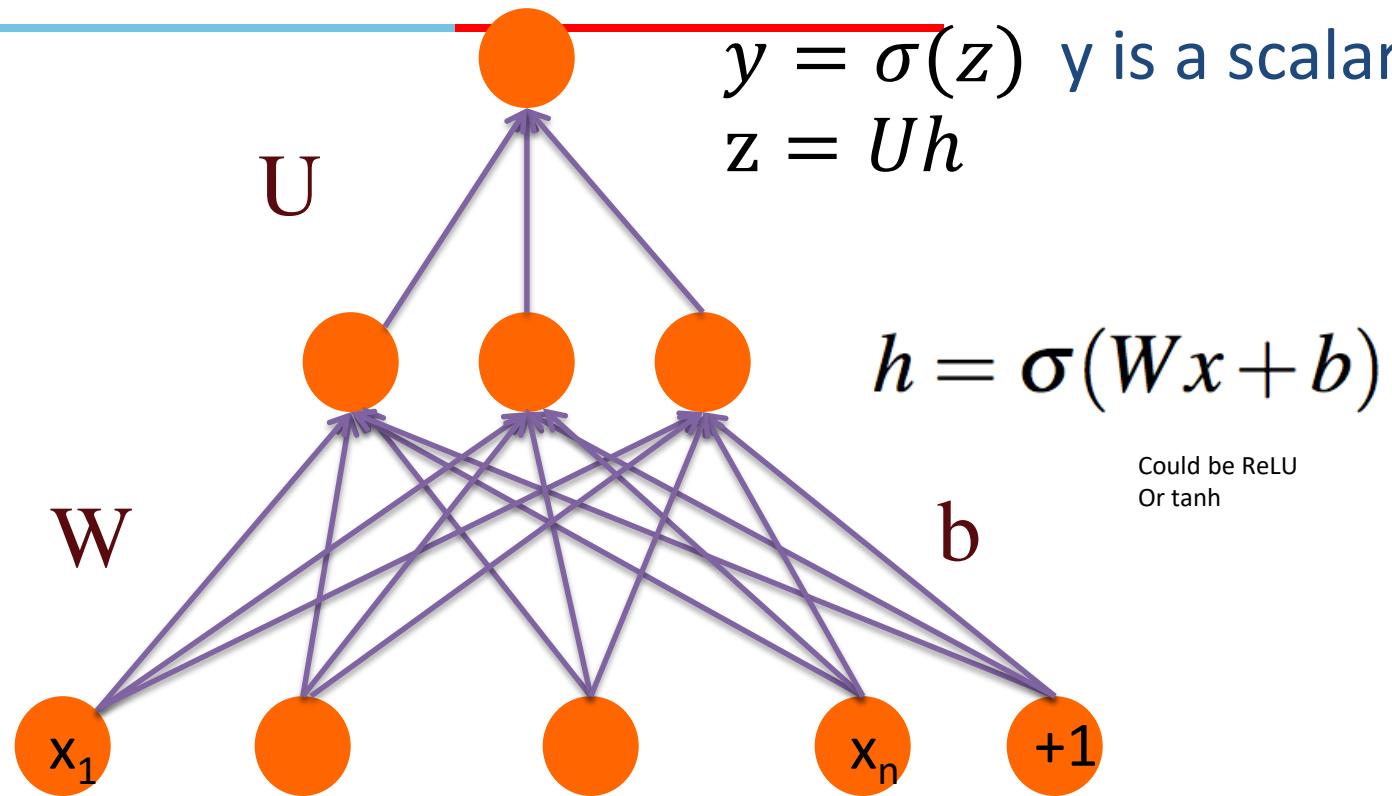


# Two-Layer Network with scalar output

Output layer  
( $\sigma$  node)

hidden units  
( $\sigma$  node)

Input layer  
(vector)

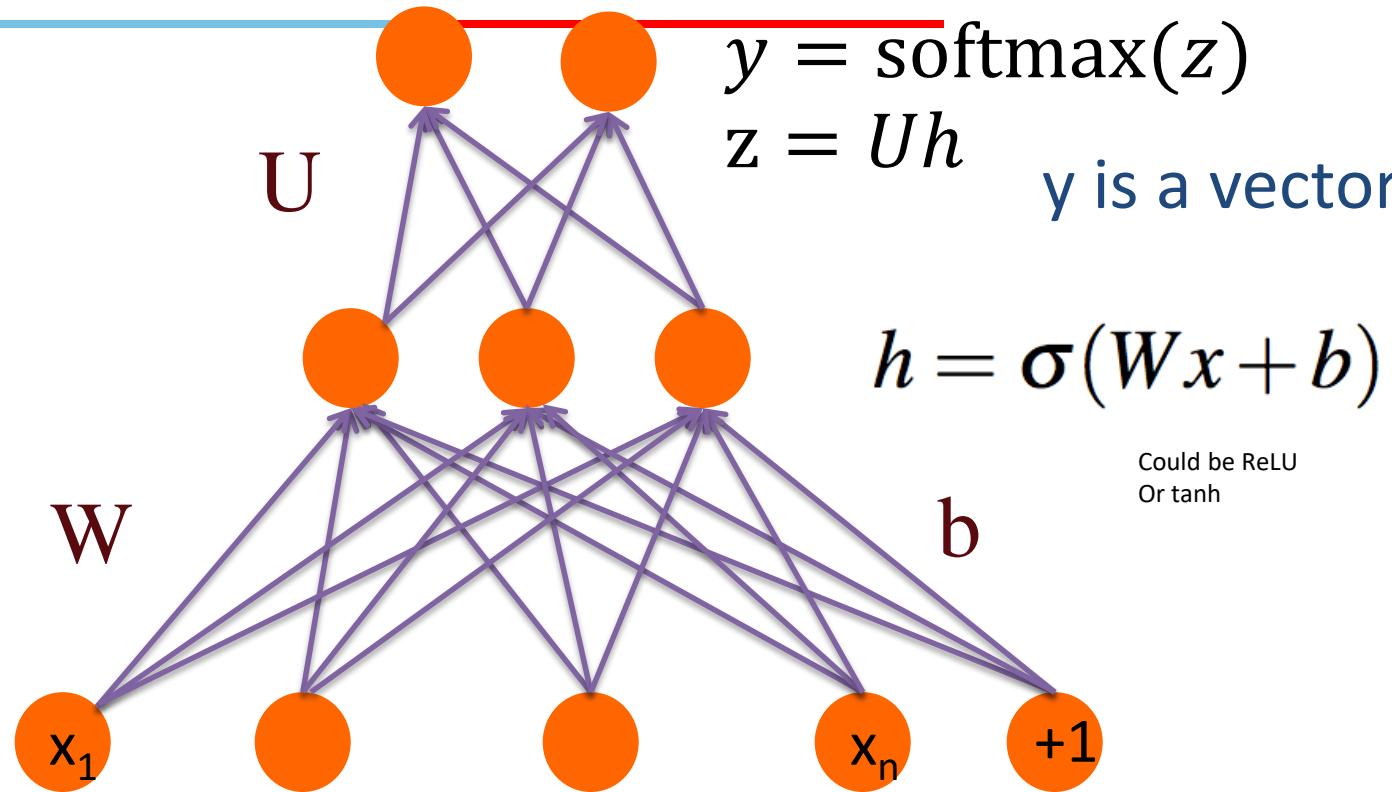


# Two-Layer Network with softmax output

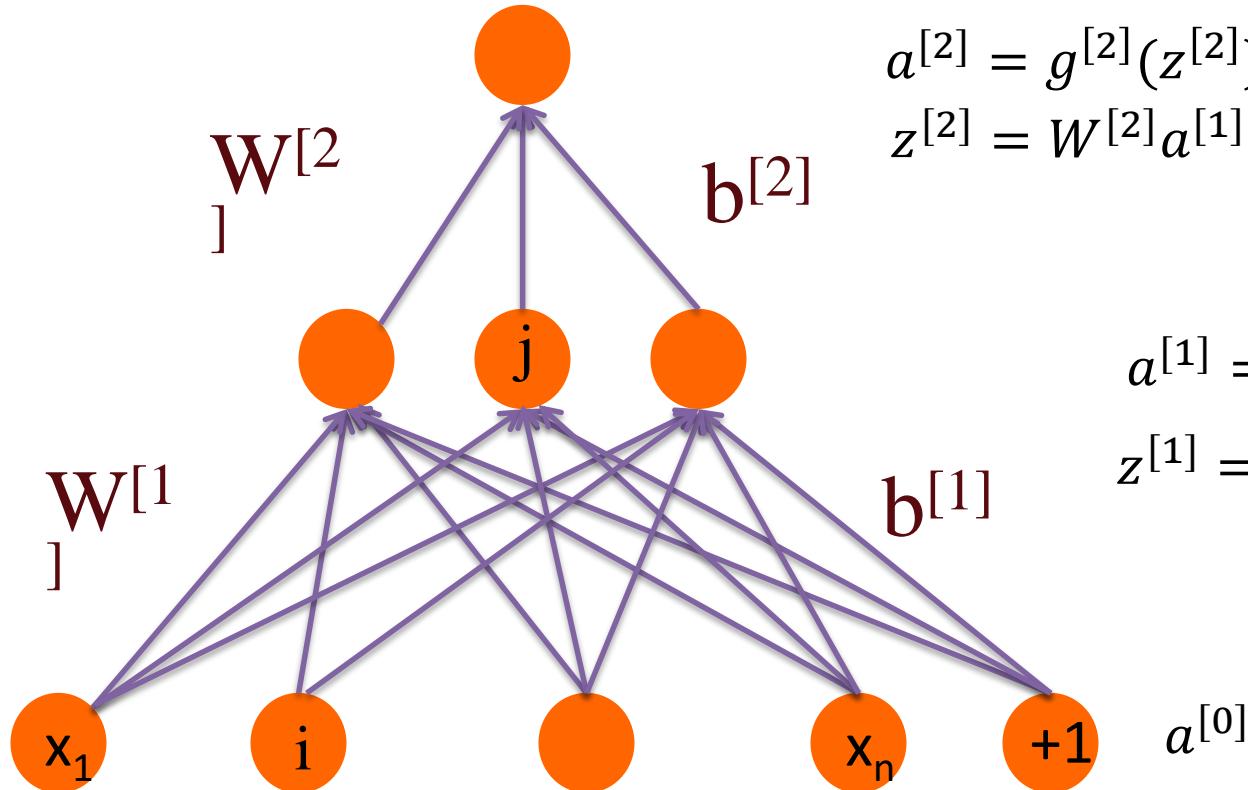
Output layer  
( $\sigma$  node)

hidden units  
( $\sigma$  node)

Input layer  
(vector)



# Multi-layer Notation



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

$$a^{[2]} = g^{[2]}(z^{[2]}) \quad \text{sigmoid or softmax}$$

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

$$a^{[1]} = g^{[1]}(z^{[1]}) \quad \text{ReLU}$$

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

$$a^{[0]}$$

# Multi Layer Notation

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

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

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

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

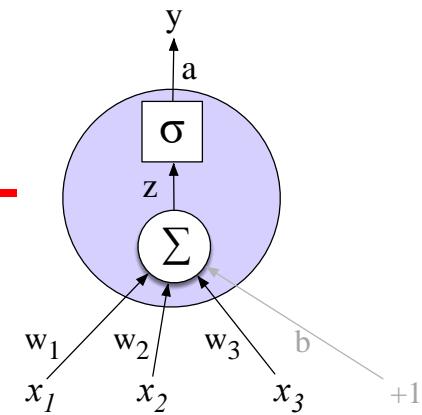
$$\hat{y} = a^{[2]}$$

**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

---

- Let's switch to a notation without the bias unit
  - Just a notational change
    1. Add a dummy node  $a_0=1$  to each layer
    2. Its weight  $w_0$  will be the bias
    3. So input layer  $a^{[0]}_0=1$ ,
      - And  $a^{[1]}_0=1, a^{[2]}_0=1, \dots$

# Replacing the bias unit

---

- Instead of:

$$x = x_1, x_2, \dots, x_{n0}$$

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

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

- We'll do this:

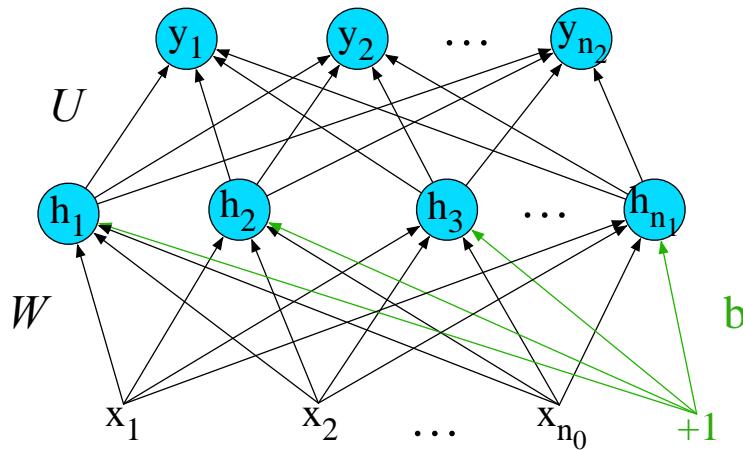
$$x = x_0, x_1, x_2, \dots, x_{n0}$$

$$h = \sigma(Wx)$$

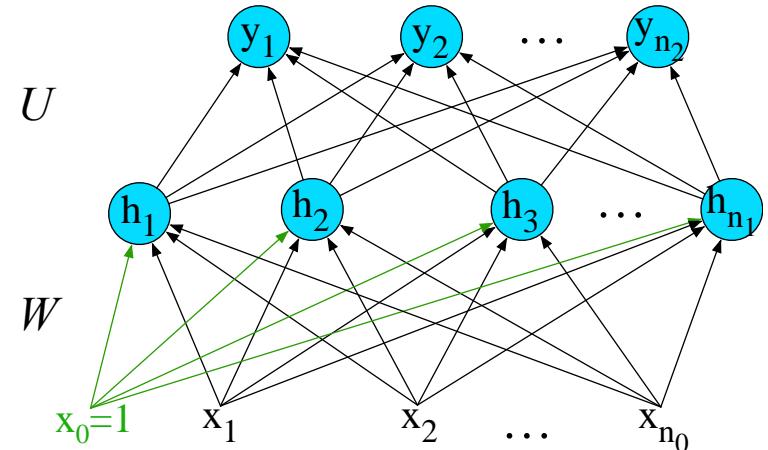
$$\sigma \left( \sum_{i=0}^{n_0} W_{ji} x_i \right)$$

# Replacing the bias unit

Instead of:



We'll do this:



# Use cases for feedforward networks

---

- Let's consider 2 (simplified) sample tasks:
  1. Text classification
  2. Language modeling
- State of the art systems use more powerful neural architectures, but simple models are useful to consider!

# Classification: Sentiment Analysis

- We could do exactly what we did with logistic regression
- Input layer are binary features as before
- Output layer is 0 or 1

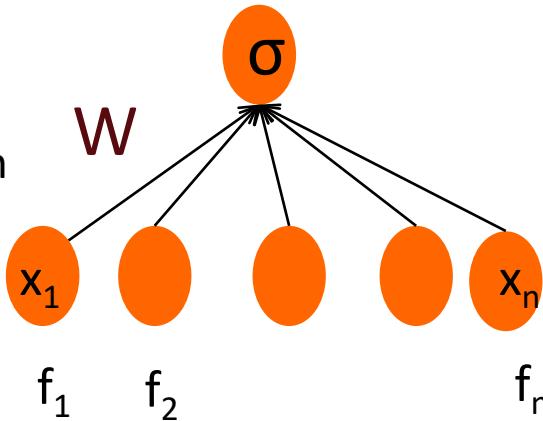


# Sentiment Features

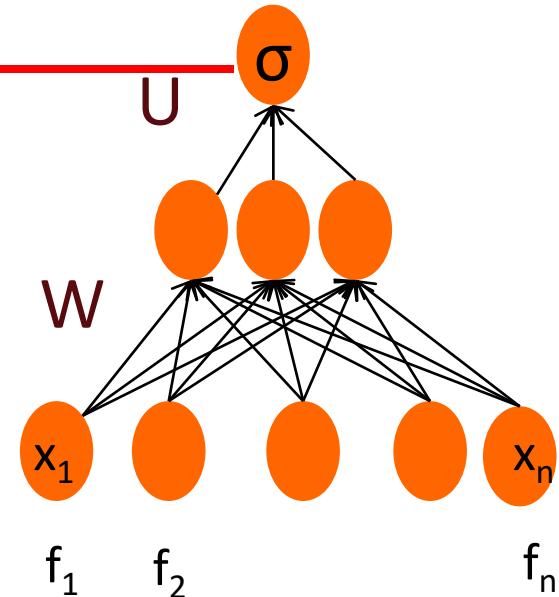
| Var   | Definition  |
|-------|---|
| $x_1$ | count(positive lexicon) $\in$ doc)  |
| $x_2$ | count(negative lexicon) $\in$ doc)  |
| $x_3$ | $\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$ |
| $x_4$ | count(1st and 2nd pronouns $\in$ doc)   |
| $x_5$ | $\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$  |
| $x_6$ | log(word count of doc)  |

# Feedforward nets for simple classification

Logistic  
Regression



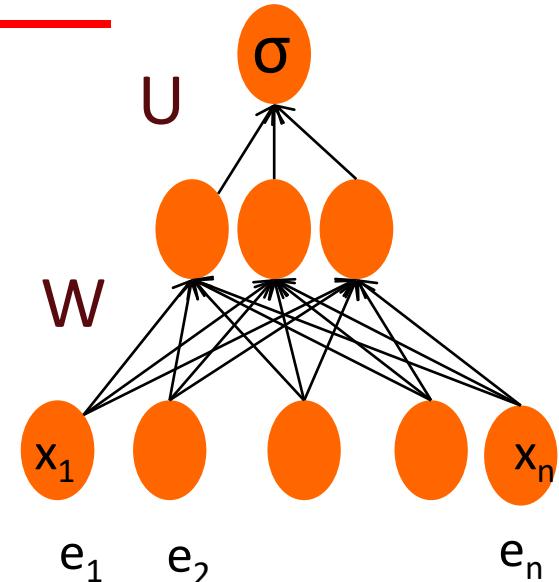
2-layer  
feedforward  
network



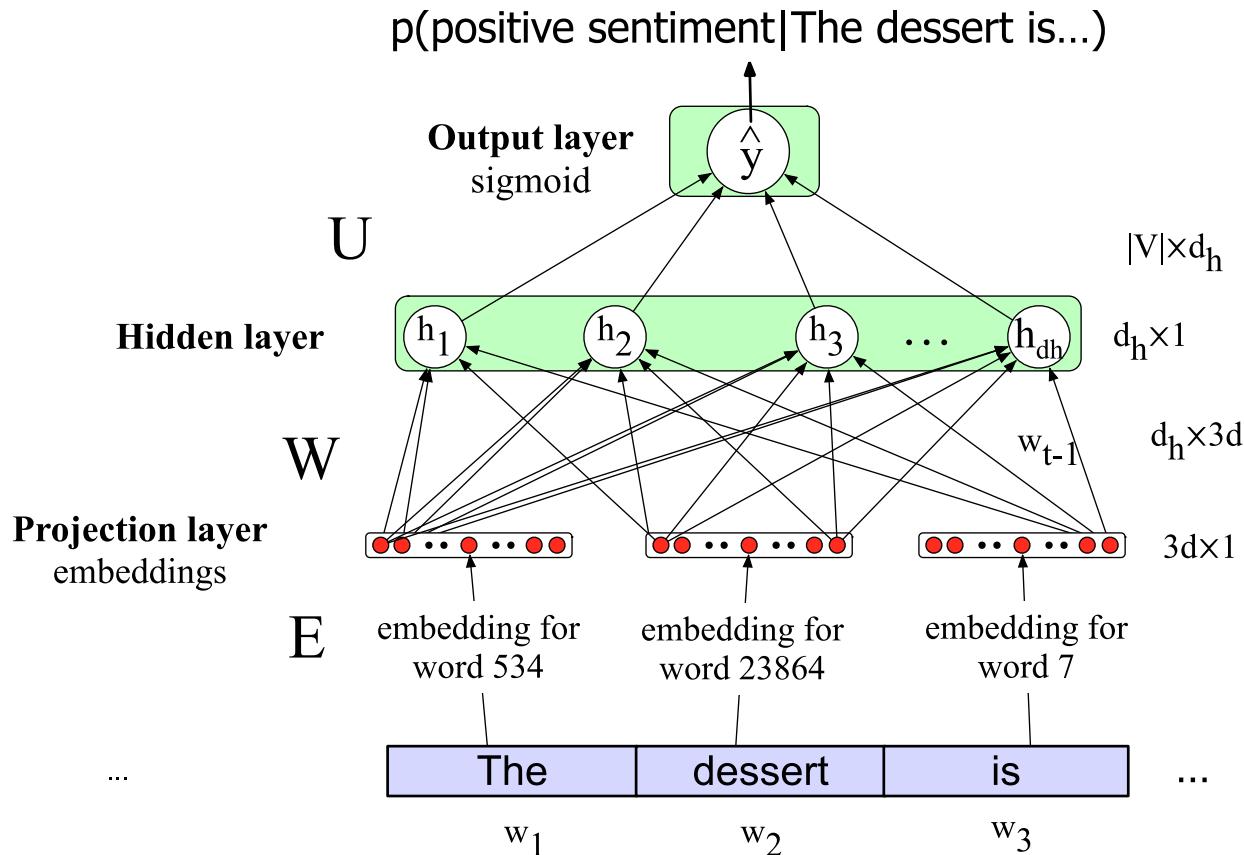
- Just adding a hidden layer to logistic regression
- allows the network to use non-linear interactions between features
- which may (or may not) improve performance.

# Even better: 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!



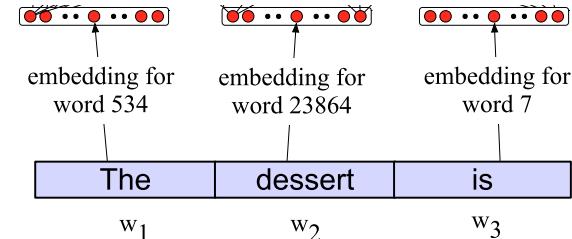
# Neural Net Classification with embeddings as input features!



# Issue: texts come in different sizes

- This assumes a fixed size length (3)!
- Kind of unrealistic.
- Some simple solutions (more sophisticated solutions later)

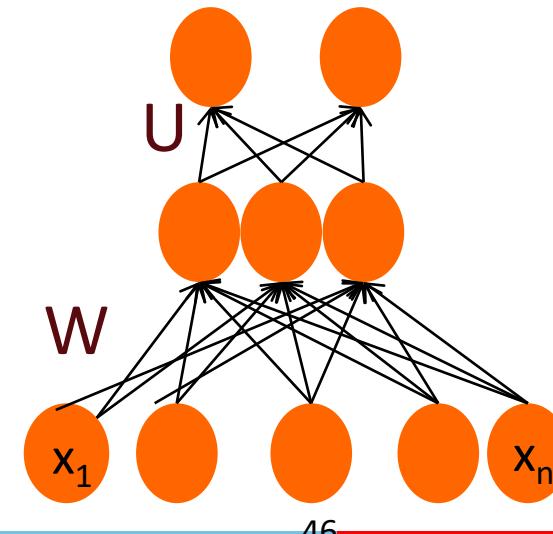
1. Make the input the length of the longest review
  - If shorter then pad with zero embeddings
  - Truncate if you get longer reviews at test time
2. Create a single "sentence embedding" (the same dimensionality as a word) to represent all the words
  - Take the mean of all the word embeddings
  - Take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



# Reminder: Multiclass Outputs

- What if you have more than two output classes?
  - Add more output units (one for each class)
  - And use a “softmax layer”

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



# 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

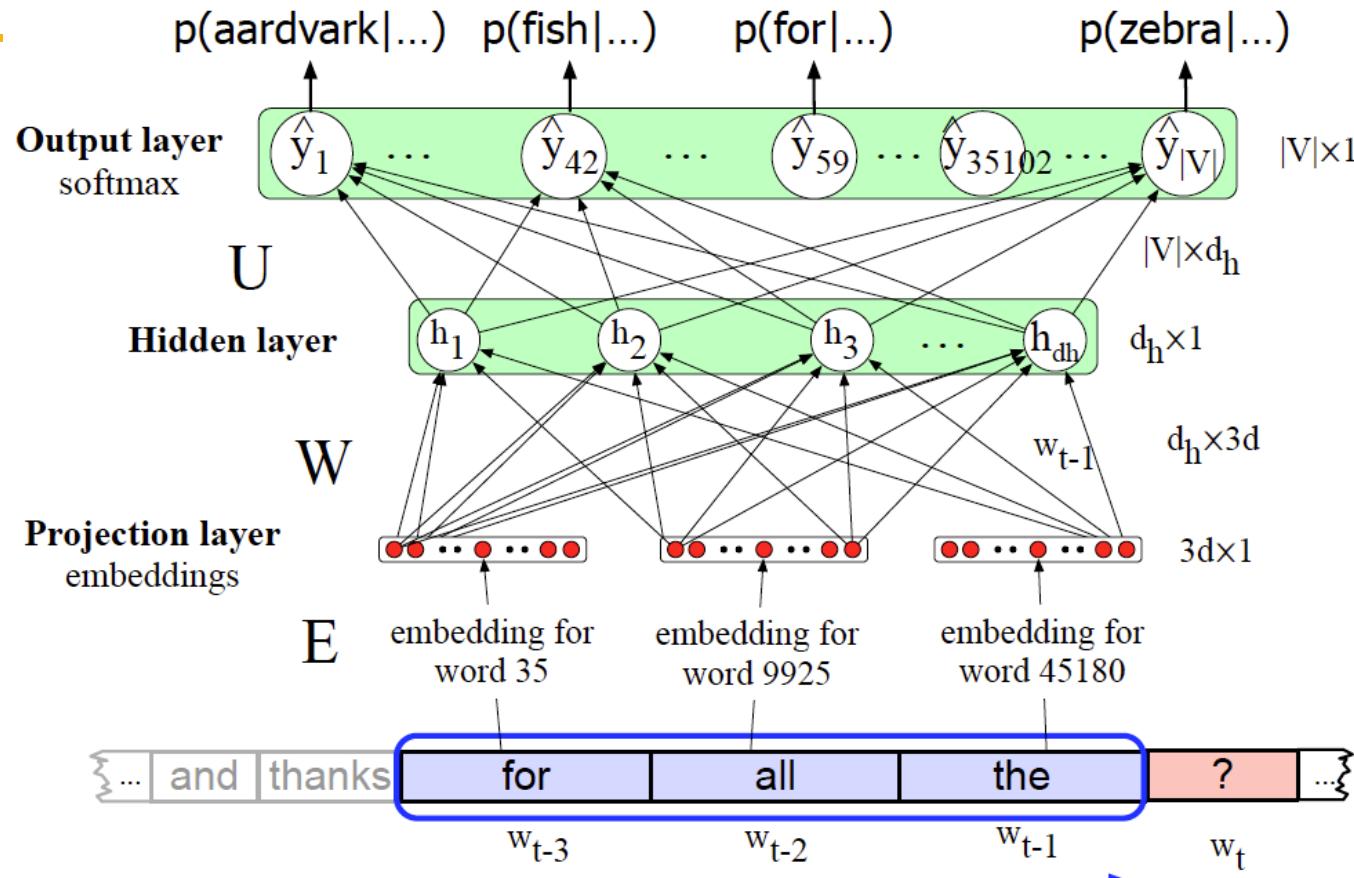
---



- **Task:** predict next word  $w_t$
- given prior words  $w_{t-1}, w_{t-2}, w_{t-3}, \dots$
- **Problem:** Now we're dealing with sequences of arbitrary length.
- **Solution:** Sliding windows (of fixed length)

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

# Neural Language Model



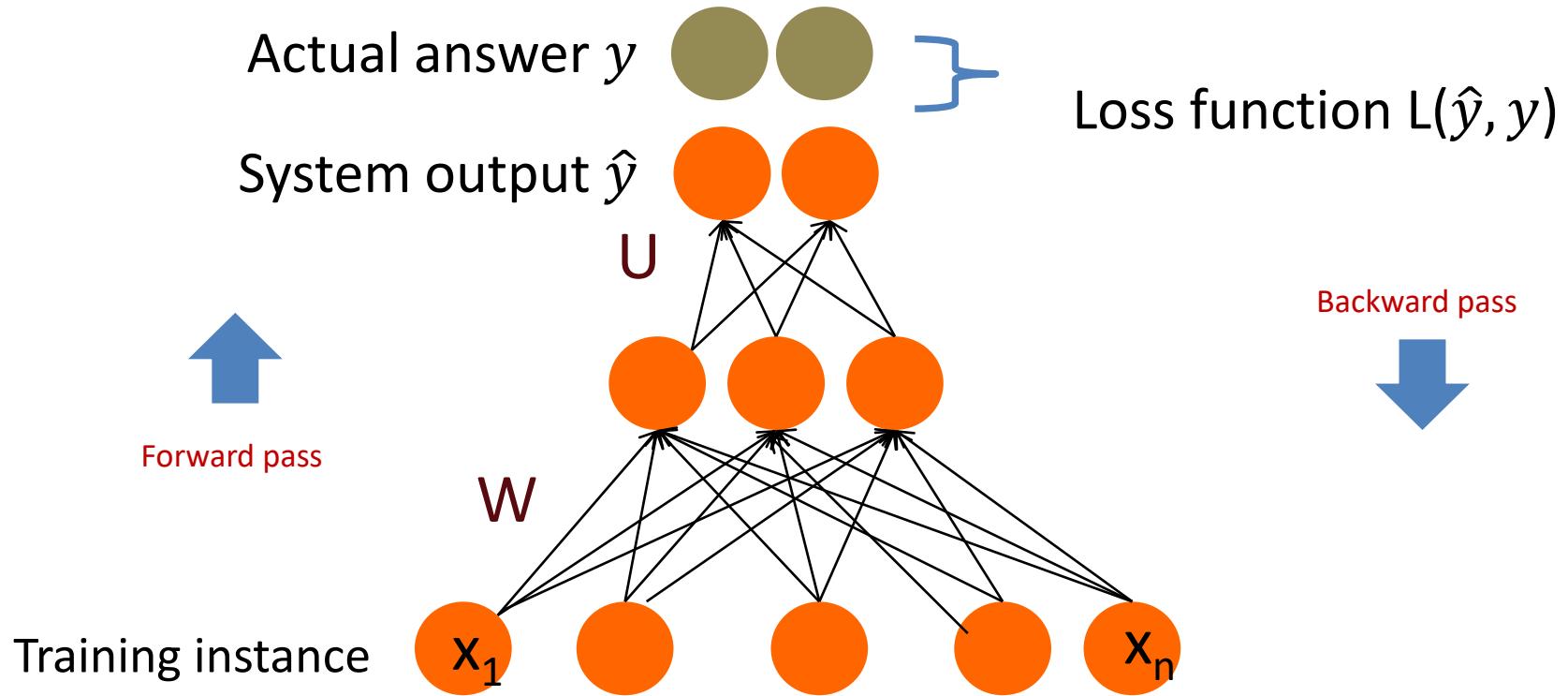
# Why Neural LMs work better than N-gram LMs

---



- **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
-

# 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

---

lead

- 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

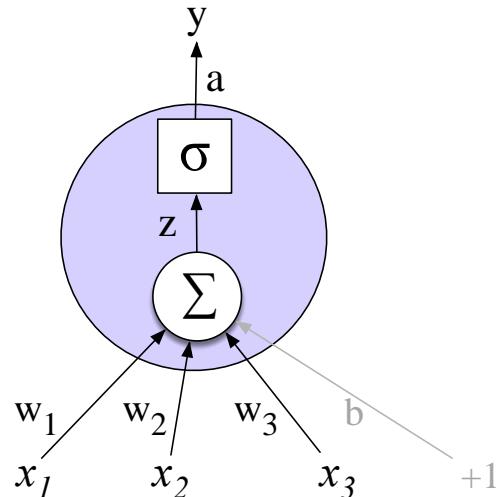
$$w^{t+1} = w^t - h \frac{d}{dw} L(f(x; w), y)$$

- For logistic regression

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

# Where did that derivative come from?

- Using the chain rule!  $f(x) = u(v(x))$   $\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$
- Intuition (see the text for details)



Derivative of the weighted sum

Derivative of the Activation

Derivative of the Loss

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial z} \frac{\partial z}{\partial w_i}$$

# How can I find that gradient for every weight in the network?

---

- These derivatives on the prior slide only give the updates for one weight layer: the last one!
- What about deeper networks?
- Lots of layers, different activation functions?
- Solution in the next lecture:
- Even more use of the chain rule!!
- Computation graphs and backward differentiation!

# Why Computation Graphs

- 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**.

# Computation Graphs

---

- A computation graph represents the process of computing a mathematical expression

$$L(a,b,c) = c(a + 2b)$$

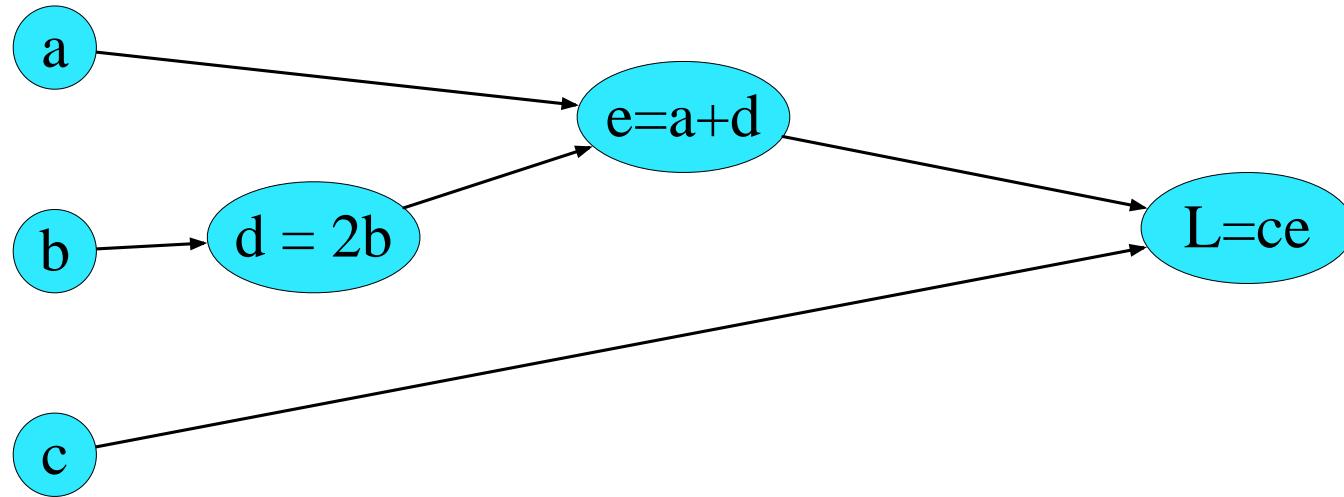
---

$$d = 2 * b$$

Computations:

$$e = a + d$$

$$L = c * e$$



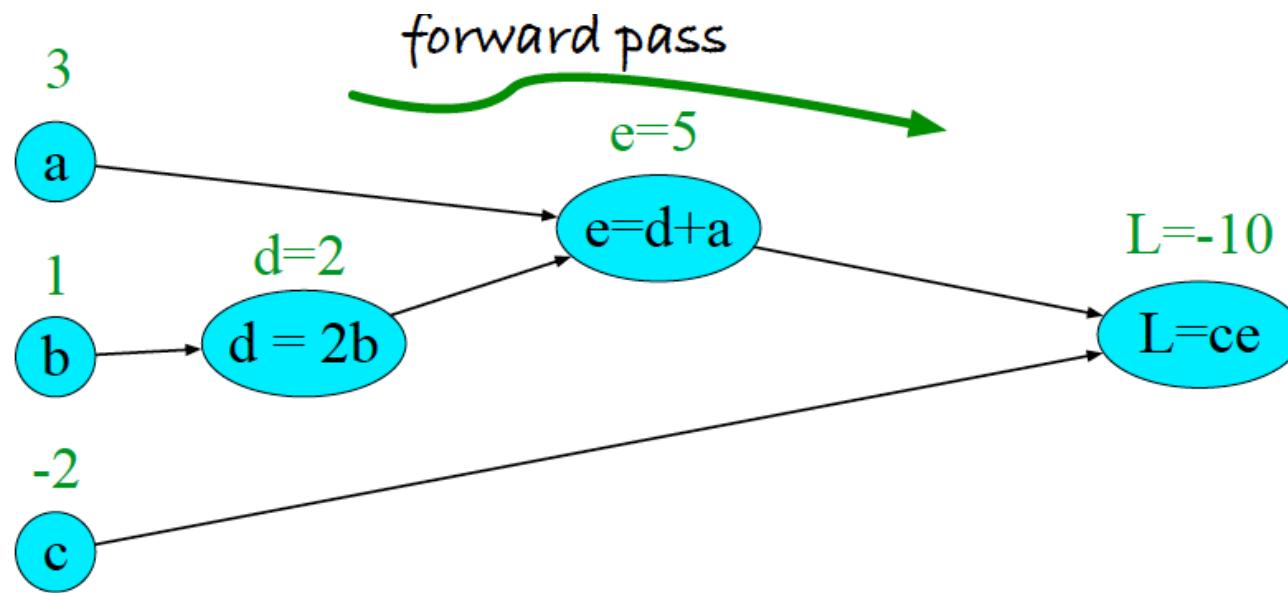
$$L(a, b, c) = c(a + 2b)$$

$$d = 2 * b$$

Computations:

$$e = a + d$$

$$L = c * e$$



# Backwards differentiation in computation graphs

---



- The importance of the computation graph comes from the backward pass
- This is used to compute the derivatives that we'll need for the weight update.

$$L(a, b, c) = c(a + 2b)$$

---

$$d = 2 * b$$

$$e = a + d$$

$$L = c * e$$

We want:  $\frac{\partial L}{\partial a}$ ,  $\frac{\partial L}{\partial b}$ , and  $\frac{\partial L}{\partial c}$

The derivative  $\frac{\partial L}{\partial a}$ , tells us how much a small change in  $a$  affects  $L$ .

# The chain rule

---

- Computing the derivative of a composite function:

$$\bullet \quad f(x) = u(v(x)) \quad \frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$$

$$\bullet \quad f(x) = u(v(w(x))) \quad \frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dw} \cdot \frac{dw}{dx}$$

•

$$L(a, b, c) = c(a + 2b)$$


---

$$d = 2 * b$$

$$e = a + d$$

$$L = c * e$$

$$\frac{\partial L}{\partial c} = e$$

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e} \frac{\partial e}{\partial a}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e} \frac{\partial e}{\partial d} \frac{\partial d}{\partial b}$$


---

# Example

$$L(a, b, c) = c(a + 2b)$$

$$d = 2 * b$$

$$e = a + d$$

$$L = c * e$$

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e} \frac{\partial e}{\partial a}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e} \frac{\partial e}{\partial d} \frac{\partial d}{\partial b}$$

$$L = ce : \quad \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e$$

$$e = a + d : \quad \frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1$$

$$d = 2b : \quad \frac{\partial d}{\partial b} = 2$$

# Example

$$a=3$$

a

$$b=1$$

b

$$c=-2$$

c

$$d = 2b$$

$$d=2$$

$$e=5$$

$$e=d+a$$

$$L=-10$$

$$L=ce$$

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e} \frac{\partial e}{\partial a}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e} \frac{\partial e}{\partial d} \frac{\partial d}{\partial b}$$

$$L = ce : \begin{matrix} \text{innovate} \\ \text{achieve} \end{matrix}$$

$$\frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e$$

$$\text{lead}$$

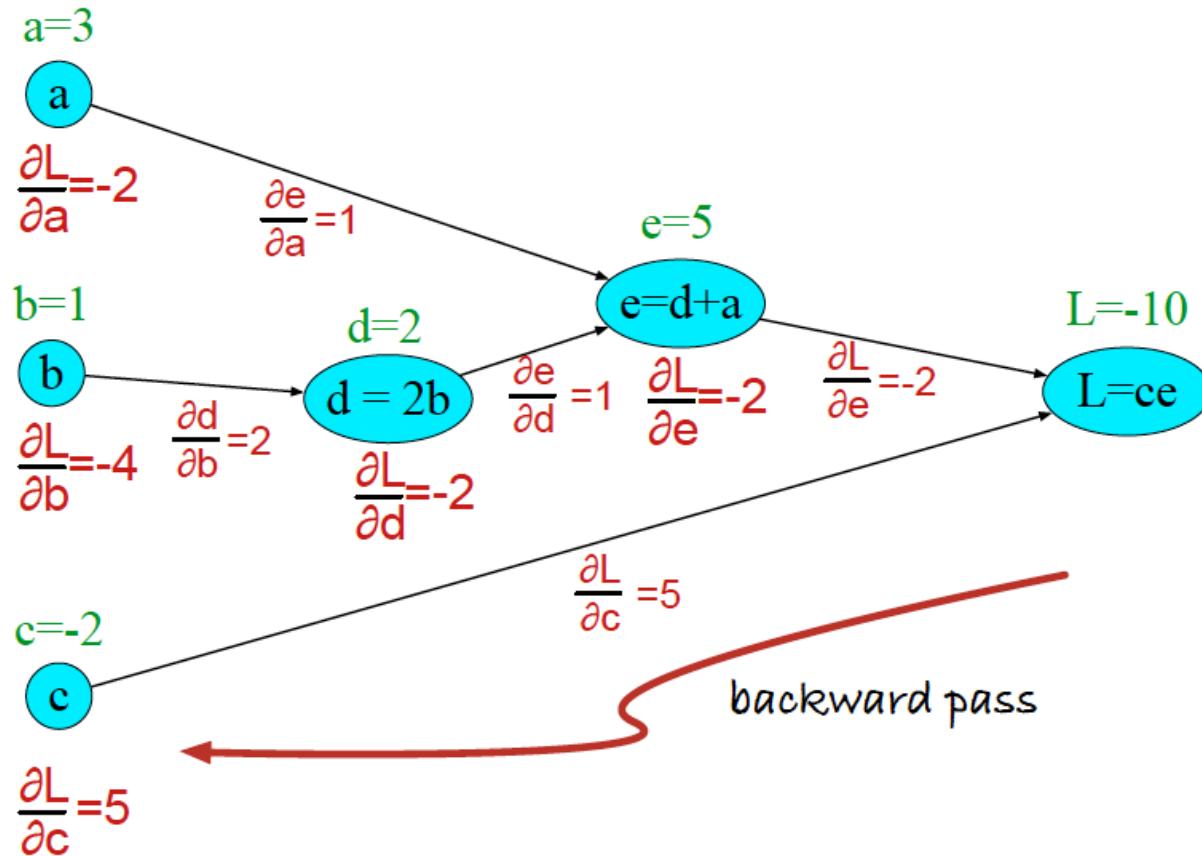
$$e = a + d :$$

$$\frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1$$

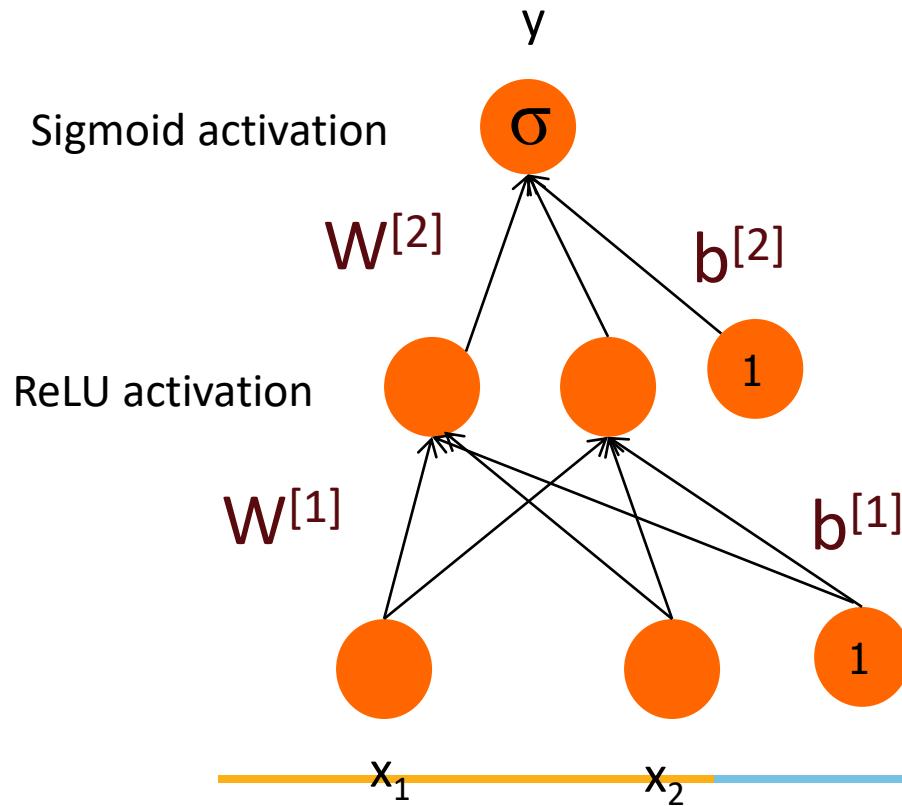
$$d = 2b :$$

$$\frac{\partial d}{\partial b} = 2$$

# Example



# Backward differentiation on a two layer network



$$\begin{aligned}z^{[1]} &= W^{[1]}x + b^{[1]} \\a^{[1]} &= \text{ReLU}(z^{[1]}) \\z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\a^{[2]} &= \sigma(z^{[2]}) \\\hat{y} &= a^{[2]}\end{aligned}$$

# Backward differentiation on a two layer network

$$z^{[1]} = W^{[1]} \mathbf{x} + b^{[1]}$$

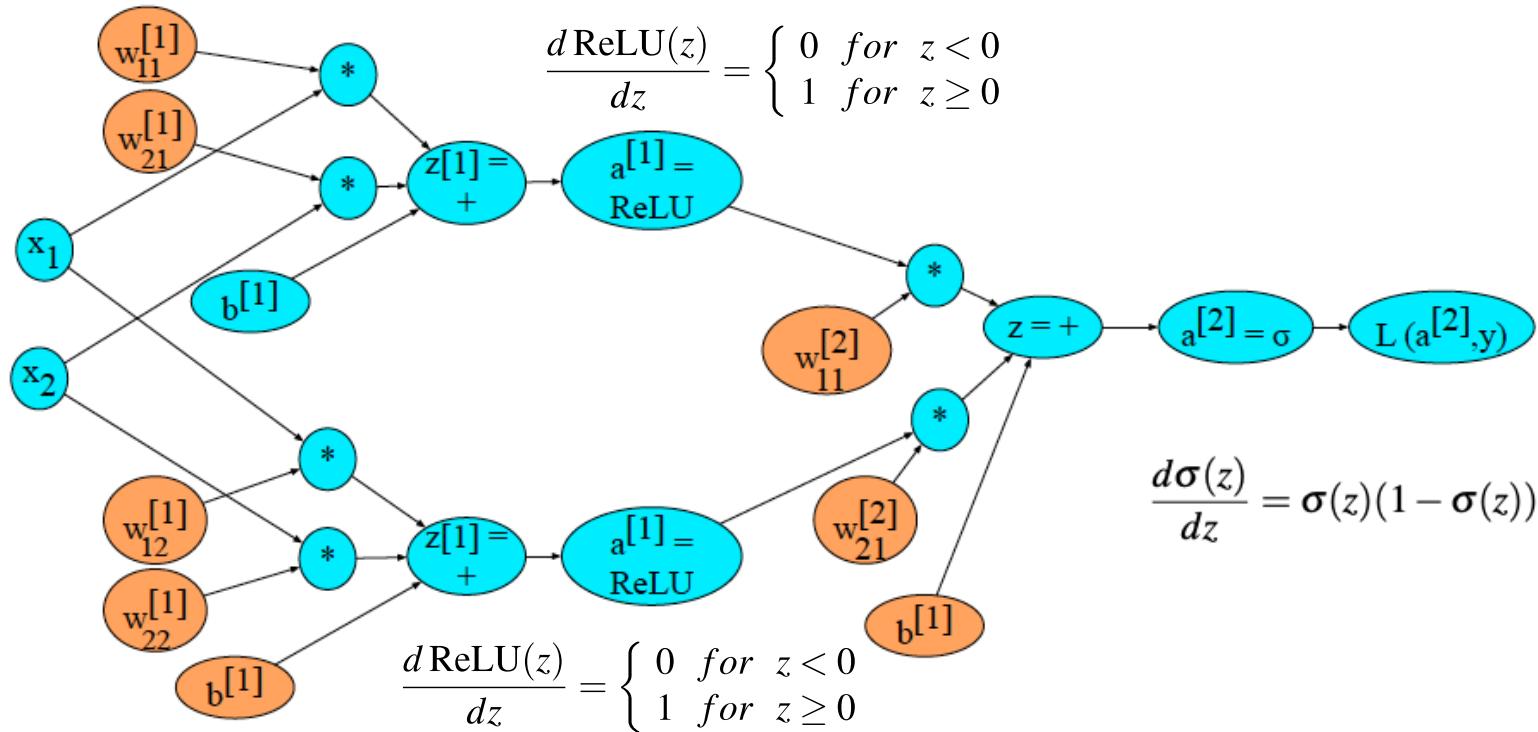
$$a^{[1]} = \text{ReLU}(z^{[1]}) \quad \frac{d \text{ReLU}(z)}{dz} = \begin{cases} 0 & \text{for } z < 0 \\ 1 & \text{for } z \geq 0 \end{cases}$$

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

$$a^{[2]} = \sigma(z^{[2]}) \quad \frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z))$$

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

# Backward differentiation on a 2-layer network



# Starting off the backward pass: $\frac{\partial L}{\partial z}$

(I'll write  $a$  for  $a^{[2]}$  and  $z$  for  $z^{[2]}$  )

$$L(\hat{y}, y) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

$$L(a, y) = -(y \log a + (1 - y) \log(1 - a))$$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \frac{\partial a}{\partial z}$$

$$\begin{aligned} \frac{\partial L}{\partial a} &= - \left( \left( y \frac{\partial \log(a)}{\partial a} \right) + (1 - y) \frac{\partial \log(1 - a)}{\partial a} \right) \\ &= - \left( \left( y \frac{1}{a} \right) + (1 - y) \frac{1}{1 - a} (-1) \right) = - \left( \frac{y}{a} + \frac{y - 1}{1 - a} \right) \end{aligned}$$

$$\frac{\partial a}{\partial z} = a(1 - a)$$


---


$$\frac{\partial L}{\partial z} = - \left( \frac{y}{a} + \frac{y - 1}{1 - a} \right) a(1 - a) = a - y$$

# N-gram vs NLM

- Compared to n-gram models, NLM can
  - handle much longer histories
  - generalize better over contexts of similar words,
  - Are more accurate at word-prediction.
- NLM are
  - much more complex,
  - slower
  - need more energy to train,
  - less interpretable than n-gram models, so for many (especially smaller) tasks an n-gram language model is still the right tool.

# Summary

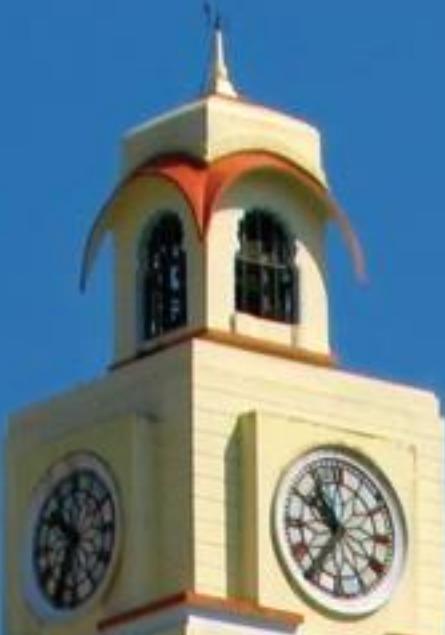
---

- 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.

# References

CH-7 - Speech and Language Processing by Daniel Jurafsky

[https://www.youtube.com/watch?v=Btmsly0j\\_dY&t=5s](https://www.youtube.com/watch?v=Btmsly0j_dY&t=5s)



# Natural Language Processing

## DSECL ZG565



**BITS** Pilani  
Pilani Campus

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



## **Session 5-Part-of-Speech Tagging**

### **Date – 17<sup>th</sup> June 2023**

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

# Session Content

---

- (Mostly) English Word Classes
  - The Penn Treebank Part-of-Speech Tag set
  - Part-of-Speech Tagging
  - Markov Chains
  - Hidden Markov Model
  - HMM Part-of-Speech Tagging
  - Part-of-Speech Tagging for Morphological Rich Languages
-

# Recap: Ambiguities in language

## I. Structural Ambiguities

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

## II. Grammatical Ambiguities

- I (feminine or masculine) go.
- Can- Noun = container, Can – Modal(auxiliary verb),  
Can-verb = to can means to pack etc

## III. Lexical Ambiguities:

Polysemy Ex: "understand" (I get it)

- Homonymy Ex: Bank= river, financial bank

Don't worry! There is no problem  
with your eyes or computer.

# Let's try

ଓ/DT এৰেৱা/NN ০ ১০/VBZ কোৱা/VBG ও/DT  
কোৱা/NN  .

ଓ/DT এৰে ৪/NN ০ ১০/VBZ ৯ ১ ৫ ৫ ০ ৫/VBG  .

ଓ/DT কুৰি ৫/NN ০ ১০/VBZ ১০ ০ ৫/VBG ০ ৫/VBG  .

ଓ/DT একা ৭ ৭ ৫/JJ কুৰি ০ ৯/NN

What is the POS tag sequence of the following sentence?

ଓ একা ৭ ৭ ৫ কুৰি ৩ ও ১০ ১০ ০ ৫/VBG ০ ৫/VBG 

# Let's try

- ഓ/DT ഓ⑥ു/NN ①⑩/VBZ കുഴഞ്ഞി ① ① ⑤ു/VBG ഓ/DT കുഴി ①/NN പി/.  
a/DT dog/NN is/VBZ chasing/VBG a/DT cat/NN ./.
- ഓ/DT ഓ⑥ ④/NN ①⑩/VBZ ⑨ ① ⑤ ⑤ ① ⑤ു/VBG പി/.  
a/DT fox/NN is/VBZ running/VBG ./.
- ഓ/DT ഓ⑥ ⑤/NN ①⑩/VBZ ⑩ ① ⑤ു ① ⑤ു/VBG പി/.  
a/DT boy/NN is/VBZ singing/VBG ./.
- ഓ/DT ഓ ⑦ ⑦ ⑤/JJ ഓ ① ⑨ു/NN  
a/DT happy/JJ bird/NN
- ഓ ഓ ⑦ ⑦ ⑤ കുഴി ① ③ ഓ ⑩ ① ① ⑤ു ① ⑤ു പി/  
a happy cat was singing .

# 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).

**Input:** the lead paint is unsafe

**Output:** the/Det lead/N paint/N is/V unsafe/Adj

# Why is POS Tagging Useful?

First step of a vast number of practical tasks

- Helps in stemming/lemmatization
- Parsing
  - Need to know if a word is an N or V before you can parse
  - Parsers can build trees directly on the POS tags instead of maintaining a lexicon
- Named Entity Recognition and Information Extraction
  - Finding names, relations, etc.
- Machine Translation
- Selecting words of specific Parts of Speech (e.g. nouns) in pre-processing documents (for IR etc.)

# 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

- The process of assigning a part-of-speech or lexical class marker to each word in a collection.

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

# Open and Closed Classes

- 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)
- 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!

# Open Class Words

- Nouns
  - Proper nouns (Boulder, Granby, Eli Manning)
    - English capitalizes these.
  - Common nouns (the rest).
  - Count nouns and mass nouns
    - Count: have plurals, get counted: goat/goats, one goat, two goats
    - Mass: don't get counted (snow, salt, communism) (\*two snows)
- Adverbs: tend to modify things
  - **Unfortunately**, John walked home **extremely slowly yesterday**
  - Directional/locative adverbs (here, home, downhill)
  - Degree adverbs (extremely, very, somewhat)
  - Manner adverbs (slowly, slinkily, delicately)
- Verbs
  - In English, have morphological affixes (eat/eats/eaten)

# 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, ...*

# Prepositions from CELEX

|       |         |         |        |            |       |       |    |
|-------|---------|---------|--------|------------|-------|-------|----|
| of    | 540,085 | through | 14,964 | worth      | 1,563 | pace  | 12 |
| in    | 331,235 | after   | 13,670 | toward     | 1,390 | nigh  | 9  |
| for   | 142,421 | between | 13,275 | plus       | 750   | re    | 4  |
| to    | 125,691 | under   | 9,525  | till       | 686   | mid   | 3  |
| with  | 124,965 | per     | 6,515  | amongst    | 525   | o'er  | 2  |
| on    | 109,129 | among   | 5,090  | via        | 351   | but   | 0  |
| at    | 100,169 | within  | 5,030  | amid       | 222   | ere   | 0  |
| by    | 77,794  | towards | 4,700  | underneath | 164   | less  | 0  |
| from  | 74,843  | above   | 3,056  | versus     | 113   | midst | 0  |
| about | 38,428  | near    | 2,026  | amidst     | 67    | o'    | 0  |
| than  | 20,210  | off     | 1,695  | sans       | 20    | thru  | 0  |
| over  | 18,071  | past    | 1,575  | circa      | 14    | vice  | 0  |

# English Particles

|           |         |                  |            |          |            |
|-----------|---------|------------------|------------|----------|------------|
| aboard    | aside   | besides          | forward(s) | opposite | through    |
| about     | astray  | between          | home       | out      | throughout |
| above     | away    | beyond           | in         | outside  | together   |
| across    | back    | by               | inside     | over     | under      |
| ahead     | before  | close            | instead    | overhead | underneath |
| alongside | behind  | down             | near       | past     | up         |
| apart     | below   | east, etc.       | off        | round    | within     |
| around    | beneath | eastward(s),etc. | on         | since    | without    |

# Conjunctions

|          |         |           |       |                 |     |                |   |
|----------|---------|-----------|-------|-----------------|-----|----------------|---|
| and      | 514,946 | yet       | 5,040 | considering     | 174 | forasmuch as   | 0 |
| that     | 134,773 | since     | 4,843 | lest            | 131 | however        | 0 |
| but      | 96,889  | where     | 3,952 | albeit          | 104 | immediately    | 0 |
| or       | 76,563  | nor       | 3,078 | providing       | 96  | in as far as   | 0 |
| as       | 54,608  | once      | 2,826 | whereupon       | 85  | in so far as   | 0 |
| if       | 53,917  | unless    | 2,205 | seeing          | 63  | inasmuch as    | 0 |
| when     | 37,975  | why       | 1,333 | directly        | 26  | insomuch as    | 0 |
| because  | 23,626  | now       | 1,290 | ere             | 12  | insomuch that  | 0 |
| so       | 12,933  | neither   | 1,120 | notwithstanding | 3   | like           | 0 |
| before   | 10,720  | whenever  | 913   | according as    | 0   | neither nor    | 0 |
| though   | 10,329  | whereas   | 867   | as if           | 0   | now that       | 0 |
| than     | 9,511   | except    | 864   | as long as      | 0   | only           | 0 |
| while    | 8,144   | till      | 686   | as though       | 0   | provided that  | 0 |
| after    | 7,042   | provided  | 594   | both and        | 0   | providing that | 0 |
| whether  | 5,978   | whilst    | 351   | but that        | 0   | seeing as      | 0 |
| for      | 5,935   | suppose   | 281   | but then        | 0   | seeing as how  | 0 |
| although | 5,424   | cos       | 188   | but then again  | 0   | seeing that    | 0 |
| until    | 5,072   | supposing | 185   | either or       | 0   | without        | 0 |

# POS Tagging

## Choosing a Tagset

- 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

# 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>         |      |                       |                    |

# Using the Penn Tagset

- The/DT grand/JJ jury/NN commented/VBD on/IN a/DT number/NN of/IN other/JJ topics/NNS ./.
- Prepositions and subordinating conjunctions marked IN (“although/IN I/PRP..”)
- Except the preposition “to” is just marked “TO”.

# POS Tagging

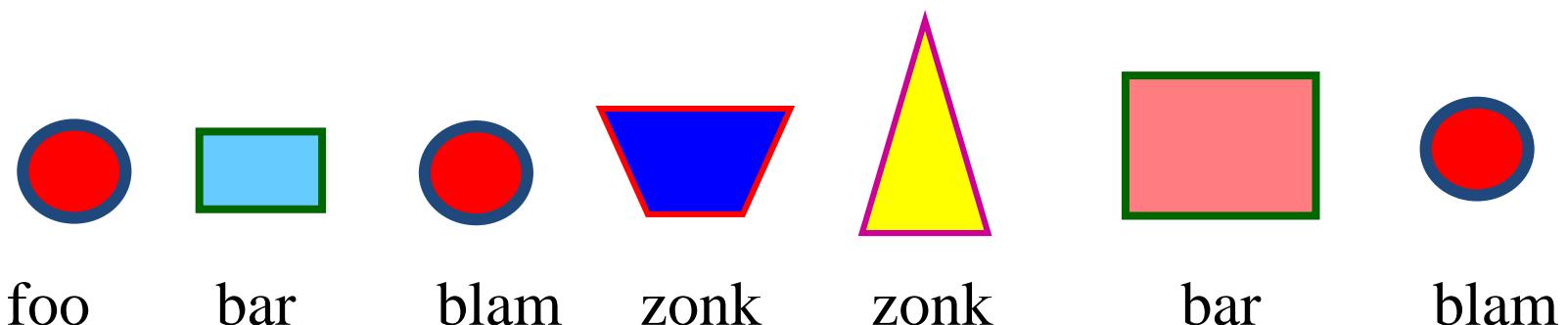
- Words often have more than one POS:  
*back*
  - The **back** door = JJ
  - On my **back** = NN
  - Win the voters **back** = RB
  - Promised to **back** the bill = VB
- The POS tagging problem is to determine the POS tag for a particular instance of a word.

# 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  $w_1 \dots w_n$ .

# Sequence Labeling Problem

- Many NLP problems can be viewed as sequence labeling.
- Each token in a sequence is assigned a label.
- Labels of tokens are dependent on the labels of other tokens in the sequence, particularly their neighbors (not i.i.d).



# Information Extraction

- Identify phrases in language that refer to specific types of entities and relations in text.
- Named entity recognition is task of identifying names of people, places, organizations, etc. in text.  
**people    organizations    places**
  - Michael Dell is the CEO of Dell Computer Corporation and lives in Austin Texas.
- Extract pieces of information relevant to a specific application, e.g. used car ads:  
**make    model    year    mileage    price**
  - For sale, 2002 Toyota Prius, 20,000 mi, \$15K or best offer. Available starting July 30, 2006.

# Semantic Role Labeling

- For each clause, determine the semantic role played by each noun phrase that is an argument to the verb.

agent patient source destination  
instrument

  - John drove Mary from Austin to Dallas in his Toyota Prius.
  - The hammer broke the window.
- Also referred to a “case role analysis,” “thematic analysis,” and “shallow semantic parsing”

# Bioinformatics

- Sequence labeling also valuable in labeling genetic sequences in genome analysis.  
**extron    intron**  
– AGCTAACGTTCGATACGGATTACAGCCT

# Problems with Sequence Labeling as Classification

- Not easy to integrate information from category of tokens on both sides.
- Difficult to propagate uncertainty between decisions and “collectively” determine the most likely joint assignment of categories to all of the tokens in a sequence.

# 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)

# 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.

# Markov Model / Markov Chain

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

# Hidden Markov Models

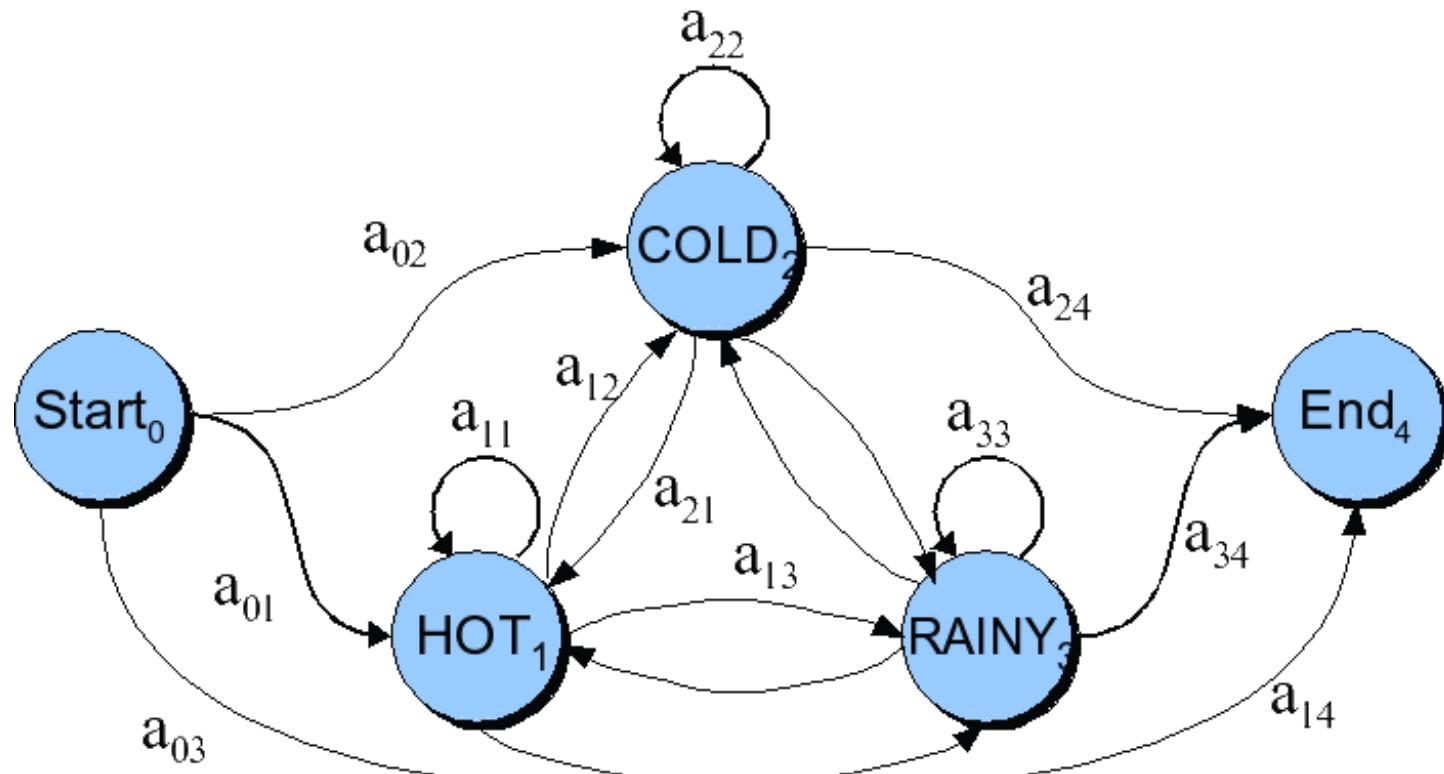
- *Generative* model.
  - There is a **hidden** underlying generator of observable events
  - The hidden generator can be modeled as a network of states and transitions
  - We want to infer the underlying state sequence given the observed event sequence

# Markov Chain: “First-order observable Markov Model”

- A set of states
  - $Q = q_1, q_2 \dots q_N$ ; the state at time  $t$  is  $q_t$
- Transition probabilities:
  - a set of probabilities  $A = a_{01} a_{02} \dots a_{n1} \dots a_{nn}$ .
  - Each  $a_{ij}$  represents the probability of transitioning from state  $i$  to state  $j$
  - The set of these is the transition probability matrix  $A$
  - Special **initial** probability vector  $\pi$
- Current state only depends on previous state

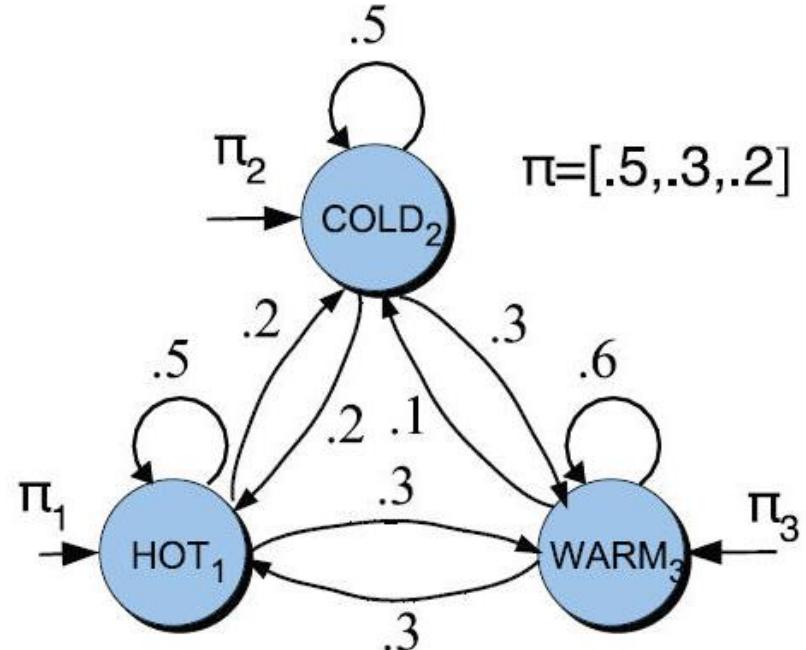
$$P(q_i | q_1 \dots q_{i-1}) = P(q_i | q_{i-1})$$

# Markov Chain for Weather



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



# HMM to predict the tags

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

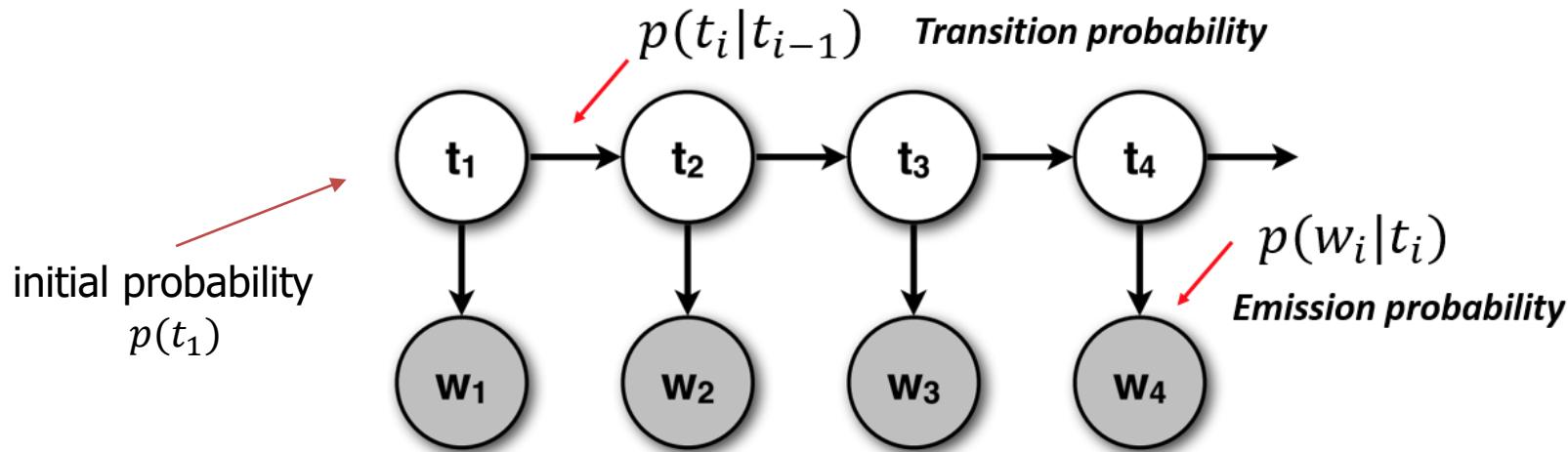
$$\begin{aligned} P(\text{DT JJ NN} \mid \text{a smart dog}) \\ = P(\cancel{\text{DT}} \text{ JJ NN} \text{ a smart dog}) / P(\text{a smart dog}) \\ = P(\cancel{\text{DT}} \text{ JJ NN}) P(\text{a smart dog} \mid \cancel{\text{DT}} \text{ JJ NN}) \end{aligned}$$

# Hidden Markov Models (Formal)

- 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 tag to tag probability
  - 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 Word to tag probability. Emission probability
  - Output probability matrix  $B = \{b_i(k)\}$ 
$$b_i(k) = P(X_t = o_k | q_t = i)$$
- Special initial probability vector  $\pi$ 
$$\rho_i = P(q_1 = i) \quad 1 \leq i \leq N$$

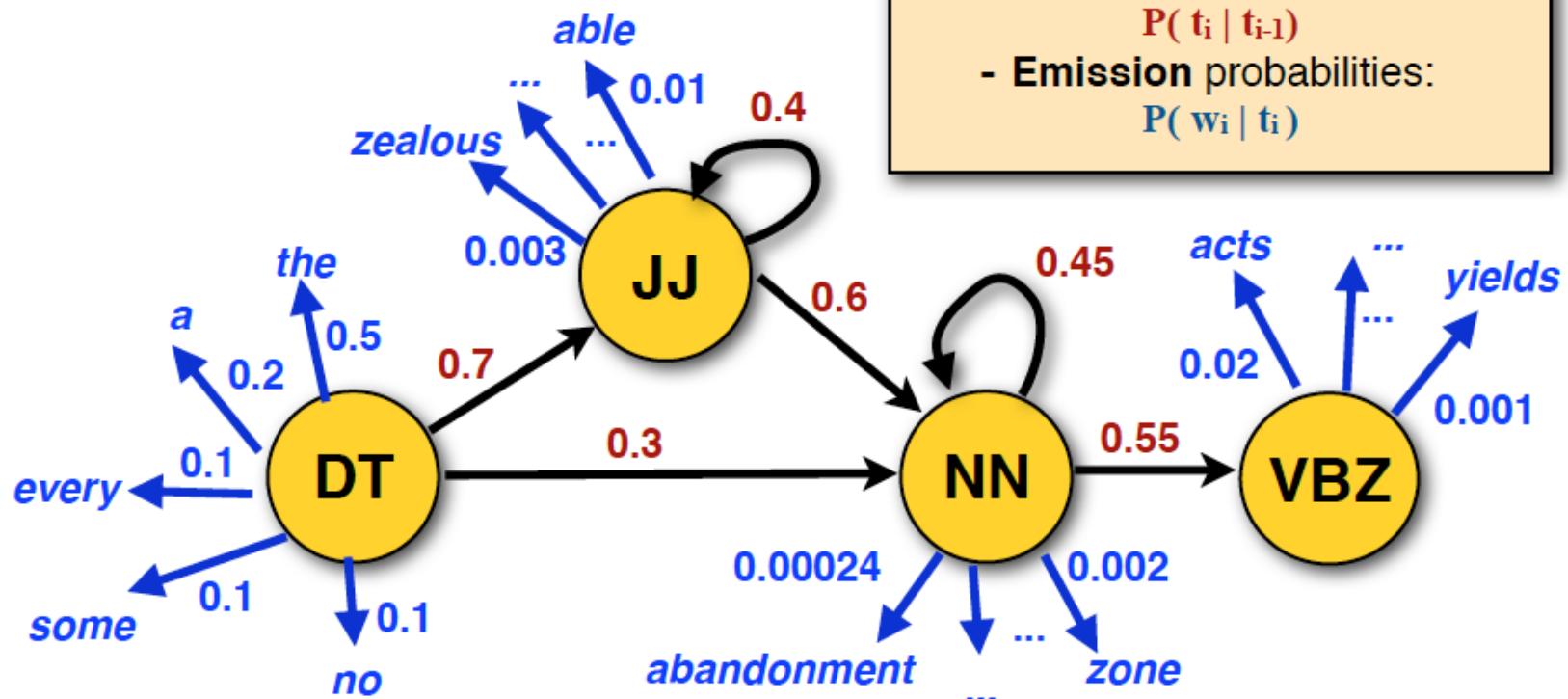
# Prediction in generative model

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



- What are the latent states that most likely generate the sequence of word  $w$

# HMMs as probabilistic FSA

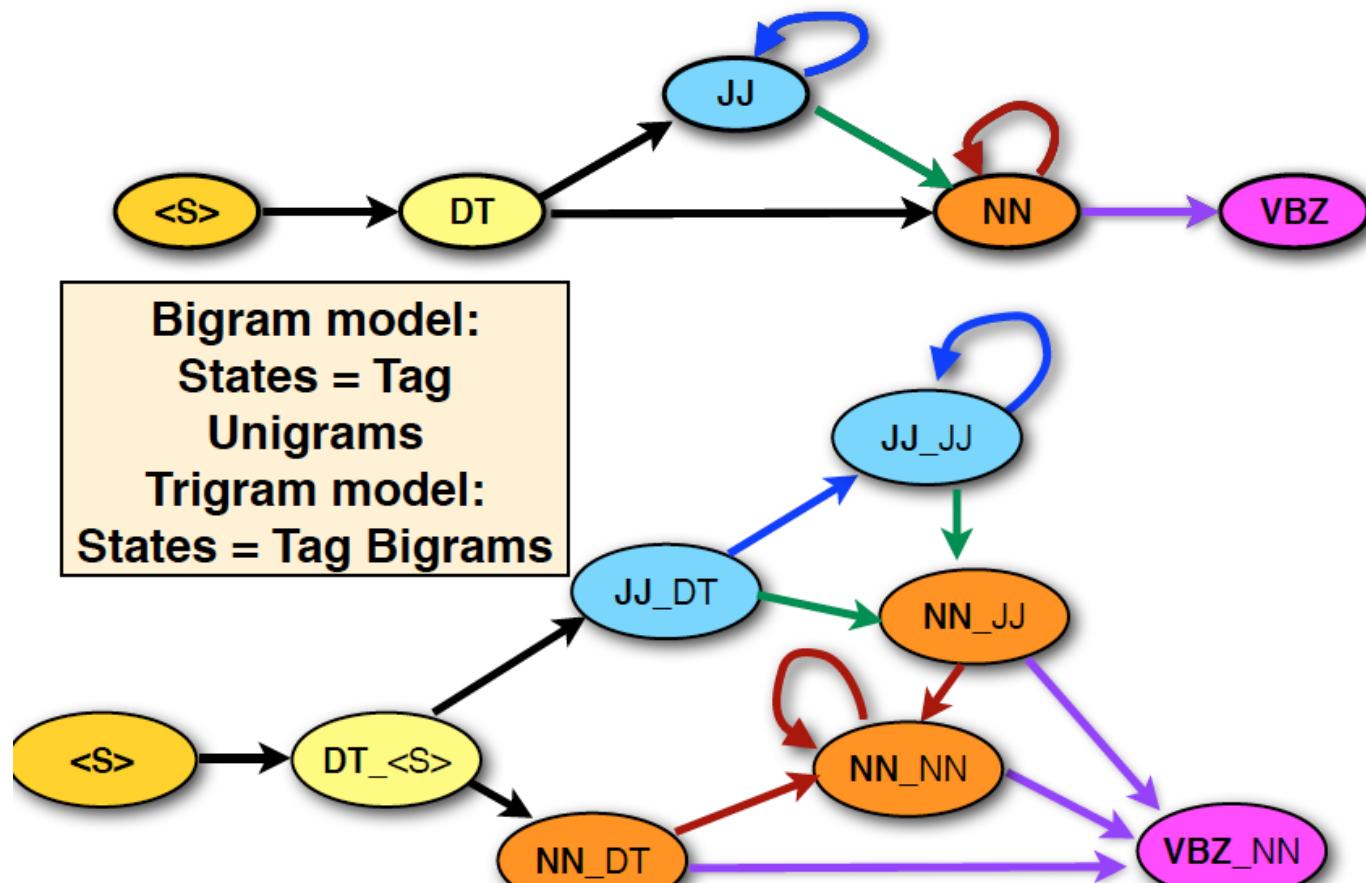


Julia Hockenmaier: Intro to NLP

# How to build a second-order HMM?

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

# Probabilistic FSA for second-order HMM



Julia Hockenmaier: Intro to NLP

# Hidden Markov Models (POS Tagging)

- 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  $\pi$

$$\pi_i = P(t_1 = i) \quad 1 \leq i \leq N$$

# Statistical POS Tagging

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

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

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

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

# Statistical POS Tagging

- 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

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

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

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

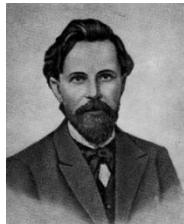
$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} P(w_1^n | t_1^n)P(t_1^n)$$

# Likelihood and Prior



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

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



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

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

# Statistical POS tagging

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

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

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

# Transition Probability

- Joint probability  $P(\mathbf{t}, \mathbf{w}) = P(\mathbf{t})P(\mathbf{w}|\mathbf{t})$
  - $P(\mathbf{t}) = P(t_1, t_2, \dots, t_n)$   
 $= P(t_1)P(t_2 | t_1)P(t_3 | t_2, t_1) \dots P(t_n | t_1 \dots t_{n-1})$   
 $\sim P(t_1)P(t_2 | t_1)P(t_3 | t_2) \dots P(t_n | t_{n-1})$   
 $= \prod_{i=1}^n P(t_i | t_{i-1})$
- Markov assumption
- Bigram model over POS tags!  
(similarly, we can define a n-gram model over POS tags, usually we called high-order HMM)

# Emission Probability

- Joint probability  $P(t, w) = P(t)P(w|t)$
  - Assume words only depend on their POS-tag
  - $P(w|t) \sim P(w_1 | t_1)P(w_2 | t_2) \dots P(w_n | t_n)$
- 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.,  $P(\text{a smart dog , DD JJ NN })$   
 $= P(\text{a} | \text{DD}) P(\text{smart} | \text{JJ }) P(\text{ dog} | \text{NN })$   
 $P(\text{DD} | \text{start}) P(\text{JJ} | \text{DD}) P(\text{NN} | \text{JJ })$

# Two Kinds of Probabilities

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

- State-to-state transition probabilities

$$P(N|D^T)$$

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

## 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:

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

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

DT occurrence count      Noun is coming after DT

# 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 Transition Probabilities

## Bigram Probabilities

|   |                | Count(i, i + 1) | Tag transition probability |
|---|----------------|-----------------|----------------------------|
| • | Bigram(Ti, Tj) | Prob(Tj Ti)     |                            |
| • | φ,ART          | 213             | .71 (213/300)              |
| • | φ,N            | 87              | .29 (87/300)               |
| • | φ,V            | 10              | .03 (10/300)               |
| • | ART,N          | 633             | 1                          |
| • | N,V            | 358             | .32                        |
| • | N,N            | 108             | .10                        |
| • | N,P            | 366             | .33                        |
| • | V,N            | 134             | .37                        |
| • | V,P            | 150             | .42                        |
| • | V,ART          | 194             | .54                        |
| • | P,ART          | 226             | .62                        |
| • | P,N            | 140             | .38                        |
| • | V,V            | 30              | .08                        |

## Tag Frequencies

| Φ   | ART | N    | V   | P   |
|-----|-----|------|-----|-----|
| 310 | 633 | 1102 | 358 | 366 |

Φ – Start Symbol

# Sample Lexical Generation Emission 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

# Table representation

Transition Matrix  $A$

|   | D   | N   | V   | A   | .   |
|---|-----|-----|-----|-----|-----|
| D | 0.8 |     | 0.2 |     |     |
| N | 0.7 | 0.3 |     |     |     |
| V | 0.6 |     |     |     | 0.4 |
| A |     | 0.8 |     | 0.2 |     |
| . |     |     |     |     |     |

$P(x|t)$

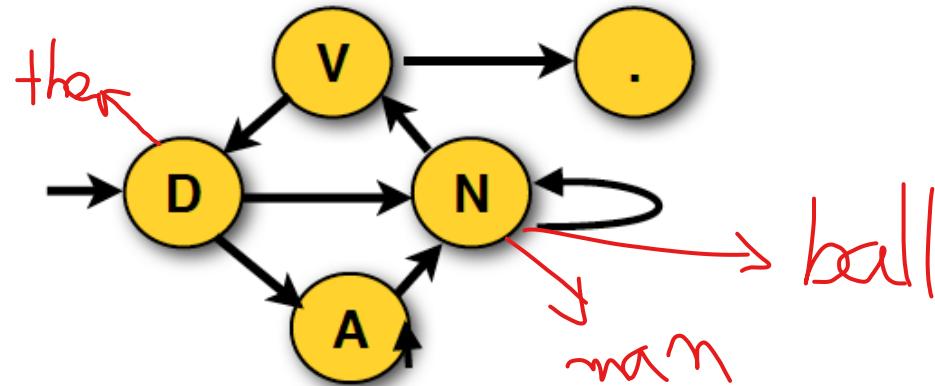
Emission Matrix  $B$

|   | <i>the</i> | <i>man</i> | <i>ball</i> | <i>throws</i> | <i>sees</i> | <i>red</i> | <i>blue</i> | . |
|---|------------|------------|-------------|---------------|-------------|------------|-------------|---|
| D | 1.0        |            |             |               |             |            |             |   |
| N |            | 0.7        | 0.3         |               |             |            |             |   |
| V |            |            |             | 0.6           | 0.4         |            |             |   |
| A |            |            |             |               |             | 0.8        | 0.2         |   |
| . |            |            |             |               |             |            |             | 1 |

$P(w|t)$

Initial state vector  $\pi$

|       | D   | N | V | A | . |
|-------|-----|---|---|---|---|
| $\pi$ | 1.0 |   |   |   |   |



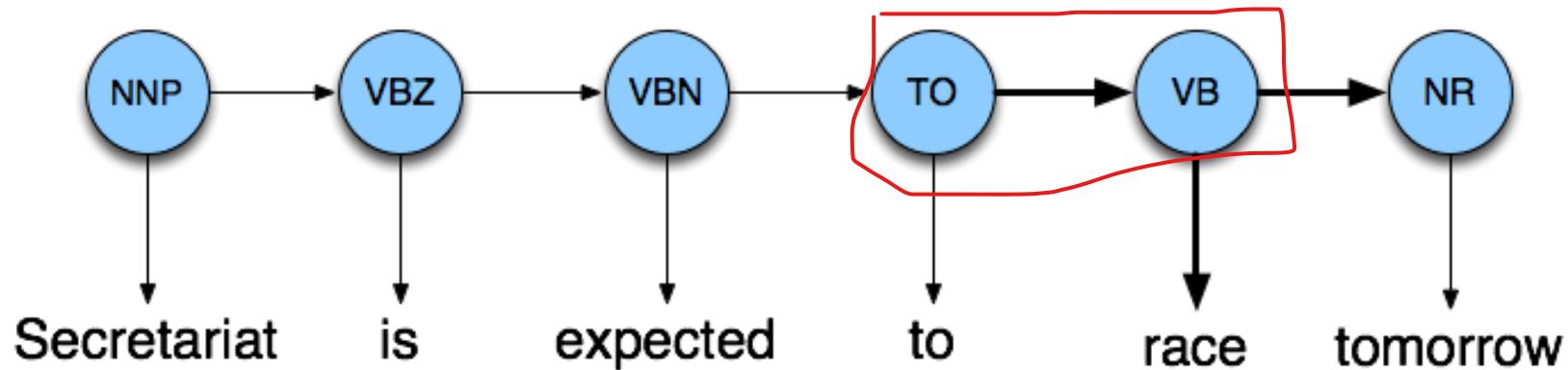
Let  $\lambda = \{A, B, \pi\}$  represents all parameters

# Example: The Verb “race”

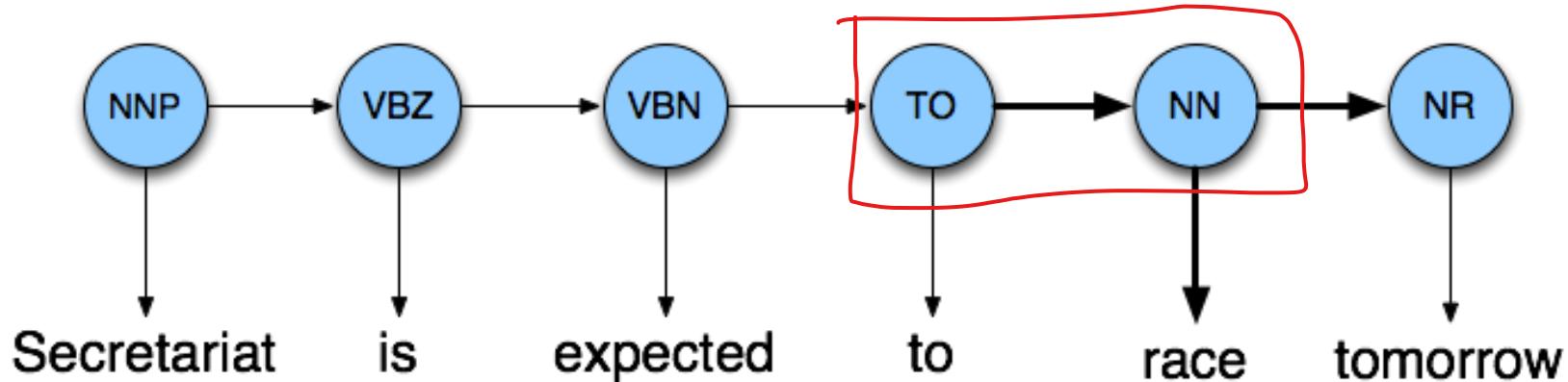
- Secretariat/**NNP** is/**VBZ** expected/**VBN** to/**TO**  
**race**/**VB** tomorrow/**NR**
- People/**NNS** continue/**VB** to/**TO** inquire/**VB**  
the/**DT** reason/**NN** for/**IN** the/**DT** **race**/**NN**  
for/**IN** outer/**JJ** space/**NN**
- How do we pick the right tag?

# Disambiguating “race”

(a)



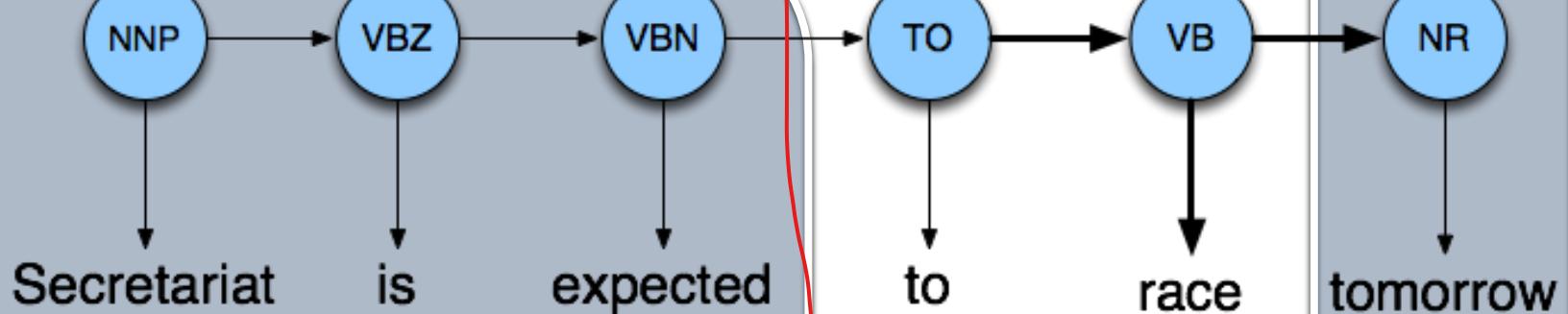
(b)



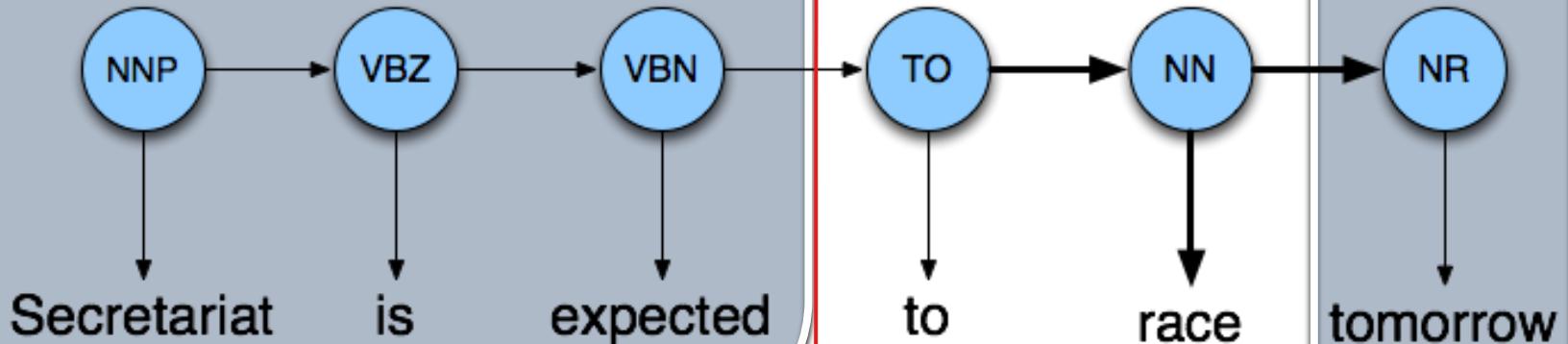
# Disambiguating “race”

No need to add these for Argmax calc

(a)

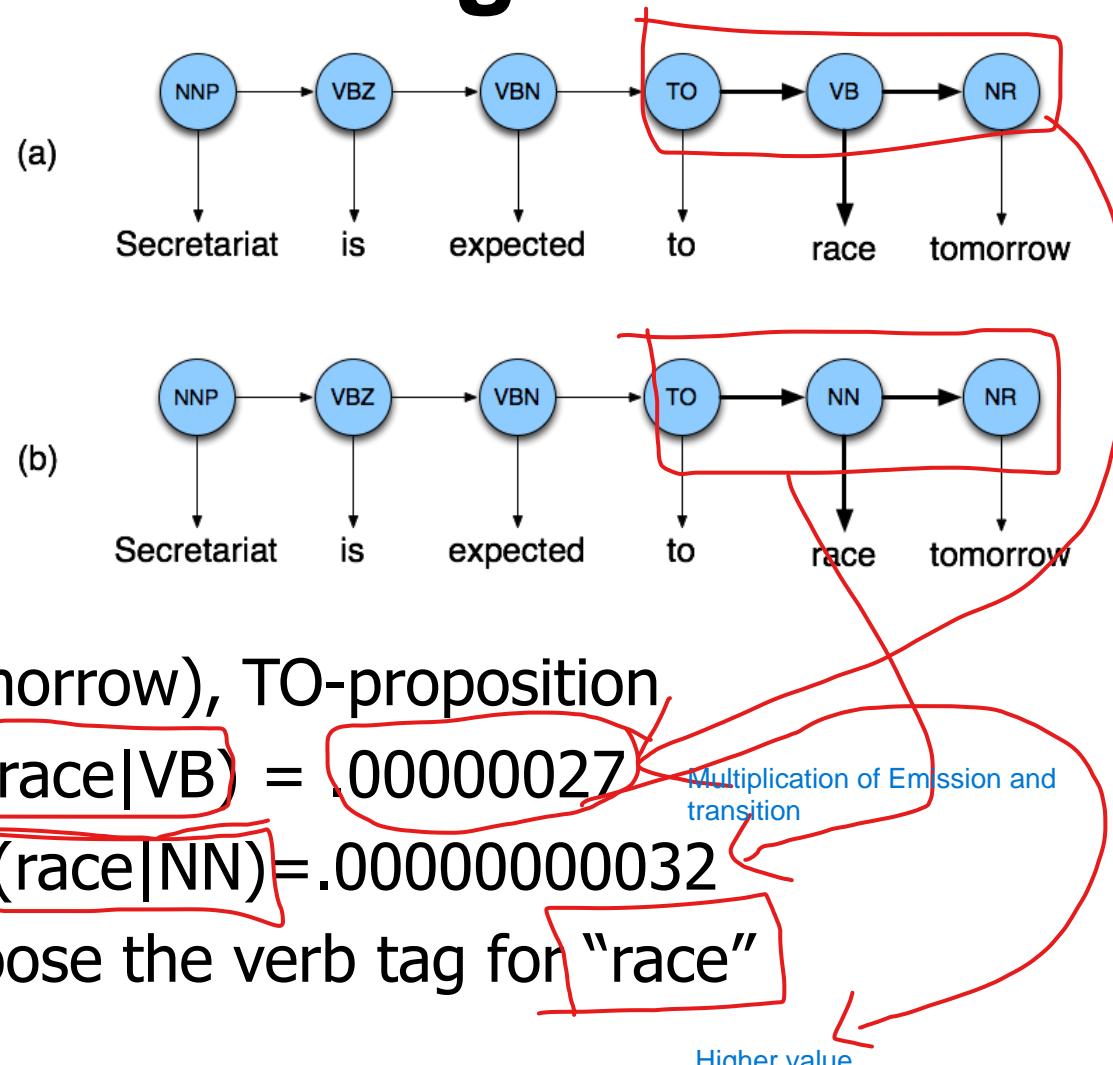


(b)



# Example : NLTK POS tags

- $P(NN|TO) = .00047$
- $P(VB|TO) = .83$
- $P(race|NN) = .00057$
- $P(race|VB) = .00012$
- $P(NR|VB) = .0027$
- $P(NR|NN) = .0012$



<https://www.guru99.com/pos-tagging-chunking-nltk.html>

# POS Tagging Demo



There are different open source POS taggers available

- 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

# Part-of-Speech Tagging for Morphological Rich Languages

- Augmentations to tagging algorithms become necessary when dealing with languages with rich morphology like Czech, Hungarian and Indian languages
- Highly inflectional languages also have much more information than English coded in word morphology, like case (nominative, accusative etc) or gender (masculine, feminine). Ex: I am speaking in English doesn't tell about gender but in Hindi/Marathi you can come to know gender depending on Bolti or Bolta(inflections)
- This information is important for tasks like parsing and coreference resolution, part-of-speech taggers for morphologically rich languages need to label words with case and gender information.

# Part-of-Speech Tagging for Morphological Rich Languages

---

- Using a morphological parse sequence like Noun+A3sg+Pnon+Gen as the part-of-speech tag greatly increases the number of parts of speech, and so tagsets can be 4 to 10 times larger than the 50–100 tags we have seen for English.
- With such large tagsets, each word needs to be morphologically analyzed to generate the list of possible morphological tag sequences (part-of-speech tags) for the word.
- The role of the tagger is then to disambiguate among these tags.
- This method also helps with unknown words since morphological parsers can accept unknown stems and still segment the affixes properly.

# Part-of-Speech Tagging for Morphological Rich Languages

---

- For non-word-space languages like Chinese, word segmentation is either applied before tagging or done jointly.
- Although Chinese words are on average very short (around 2.4 characters per unknown word compared with 7.7 for English) the problem of unknown words is still large.
- While English unknown words tend to be proper nouns in Chinese the majority of unknown words are common nouns and verbs because of extensive compounding.
- Tagging models for Chinese use similar unknown word features to English, including character prefix and suffix features

# References

---

[NLTK :: nltk.tag package](#)

[NLTK POS Tagging – Python Examples - Python Examples](#)

[https://www.coursera.org/lecture/probabilistic-models-in-nlp/part-of-speech-tagging-VbnrA](#)

[https://www.coursera.org/lecture/text-mining-analytics/3-5-how-to-do-ner-and-pos-tagging-yQEYw](#)

[https://github.com/rajesh-iiith/POS-Tagging-and-CYK-Parsing-for-Indian-Languages/tree/master/data](#)

[https://marcossilva.github.io/en/2020/09/07/coursera-nlp-module-2-week-2.html](#)



Dr. Chetana Gavankar has over 25 years of Teaching, Research and Industry experience. She has published papers in peer reviewed international conferences and journals. She is also reviewer for multiple conferences and journals. She has worked on different projects with multiple industries and received awards for her research work . Her areas of research interests include Natural Language Processing, Information Retrieval, Web Mining and Semantic Web, Ontology, Big Data Analytics, Machine learning, Deep learning and Artificial Intelligence.

# Thank you!!



# Natural Language Processing

## DSECL ZG565



**BITS** Pilani  
Pilani Campus

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



## **Session 5**

### **Date – 24<sup>th</sup> June 2023**

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

# Session Content

---

- The Hidden Markov Model
- Likelihood Computation:
  - The Forward Algorithm
- Decoding: The Viterbi Algorithm
- HMM Training:
  - The Forward-Backward Algorithm
- MEMM algorithm

# 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.

# HMM Applications

- Speech Recognition including siri
- Gene Prediction
- Handwriting recognition
- Transportation forecasting
- Computational finance
- Cryptanalysis (security)

And all applications which requires sequence processing...

# 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
  - Given an HMM  $\lambda = (A, B, \Pi)$  and an observation sequence  $O$ , determine the likelihood  $P(O|\lambda)$
  - Given an observation sequence  $O$  and a model  $\lambda = (A, B, \Pi)$  , what is the most likely state sequence?
  - Given an observation sequence, find the best model parameters for a partially specified model

# Problem 1:

## Observation Likelihood

- Given an HMM  $\lambda = (A, B, \Pi)$  and an observation sequence  $O$ , determine the likelihood  $P(O|\lambda)$ 
  - Used in model development... How do I know if some change I made to the model is making things better?
  - And in classification tasks
    - Word spotting in ASR, language identification, speaker identification, author identification, etc.
      - Train one HMM model per class
      - Given an observation, pass it to each model and compute  $P(\text{seq}|\text{model})$ .

# Problem 2:

## Decoding

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

**Decoding:** Given as input an HMM  $\lambda = (A, B)$  and a sequence of observations  $O = o_1, o_2, \dots, o_T$ , find the most probable sequence of states  $Q = q_1 q_2 q_3 \dots q_T$ .

- Typically used in tagging problems, where the tags correspond to hidden states
  - As we'll see almost any problem can be cast as a sequence labeling problem

# Problem 3: Learning

- 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)

## Problem 2: Decoding

- 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”

# 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

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

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



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

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



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

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

# Markov Assumption

---

- Context model (prior)

$$P(t_1, \dots, t_n) = \prod_{i=1}^n P(t_i | t_{i-k}, \dots, t_{i-1})$$

- Lexical model (likelihood)

$$P(w_1, \dots, w_n | t_1, \dots, t_n) = \prod_{i=1}^n P(w_i | t_i)$$

# Model Parameters

- Contextual probabilities :  $P(t_i|t_{i-k}, \dots, t_{i-1})$
- Lexical probabilities :  $P(w_i|t_i)$
- We can estimate these probabilities from a tagged corpus:

$$\hat{P}_{MLE}(w_i|t_i) = \frac{c(w_i, t_i)}{c(t_i)} \quad \hat{P}_{MLE}(t_i|t_{i-k}, \dots, t_{i-1}) = \frac{c(t_{i-k}, \dots, t_{i-1}, t_i)}{c(t_{i-k}, \dots, t_{i-1})}$$

# Computing Probabilities

- The probability of a tagging:

$$P(t_1, \dots, t_n, w_1, \dots, w_n) = \prod_{i=1}^n P(t_i | t_{i-k}, \dots, t_{i-1}) P(w_i | t_i)$$

- Finding the most probable tagging:



$$\operatorname{argmax}_{t_1, \dots, t_n} \prod_{i=1}^n P(t_i | t_{i-k}, \dots, t_{i-1}) P(w_i | t_i)$$

# Example

- Given a sentence of length 3, *\* \* the dog barks STOP* and the tag sequence *\* \* DT NN VB \* then*
- $P(w_1 w_2 w_3, y_1, y_2, y_3) = T(DT|*, *) \times T(NN|*, DT) \times T(VB|DT\ NN) \times T(STOP|NN\ VB) \times E(*|*) \times E(*|*) \times E(the|DT) \times E(dog|NN) \times E(barks|VB) \times E(STOP|*)$ 
  - Tri-gram here*
  - Transition*
  - Two tags because of Tri-gram*
  - emission*
- We can also define  $y_{-1} = ^*$  and  $y_0 = ^*$  as special symbols.

# Hidden Markov Models (formal)

---

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  $B = \{b_i(k)\}$

$$b_i(k) = P(w_i = v_k \mid t_i = i)$$

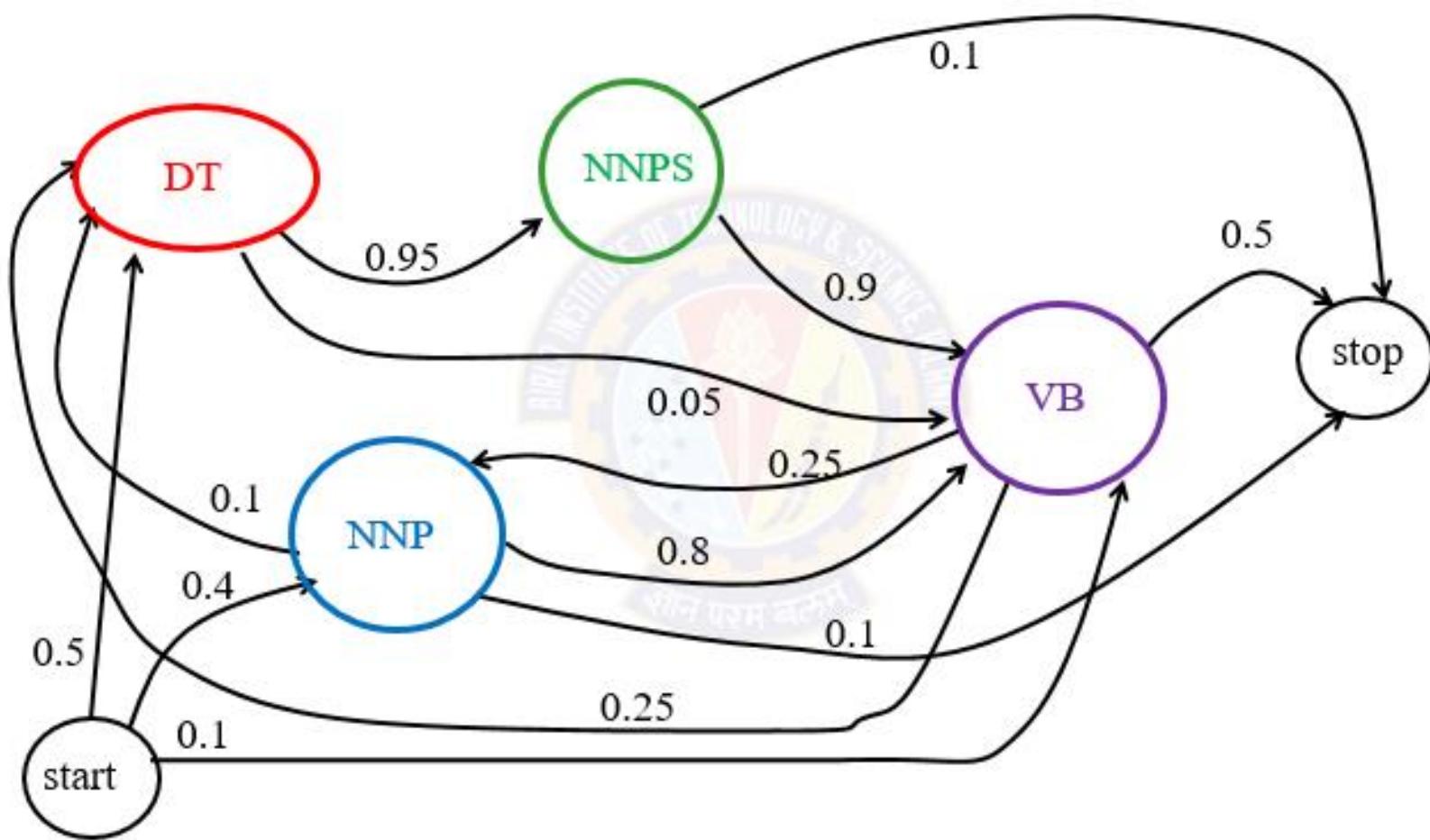
## Special initial probability vector $\pi$

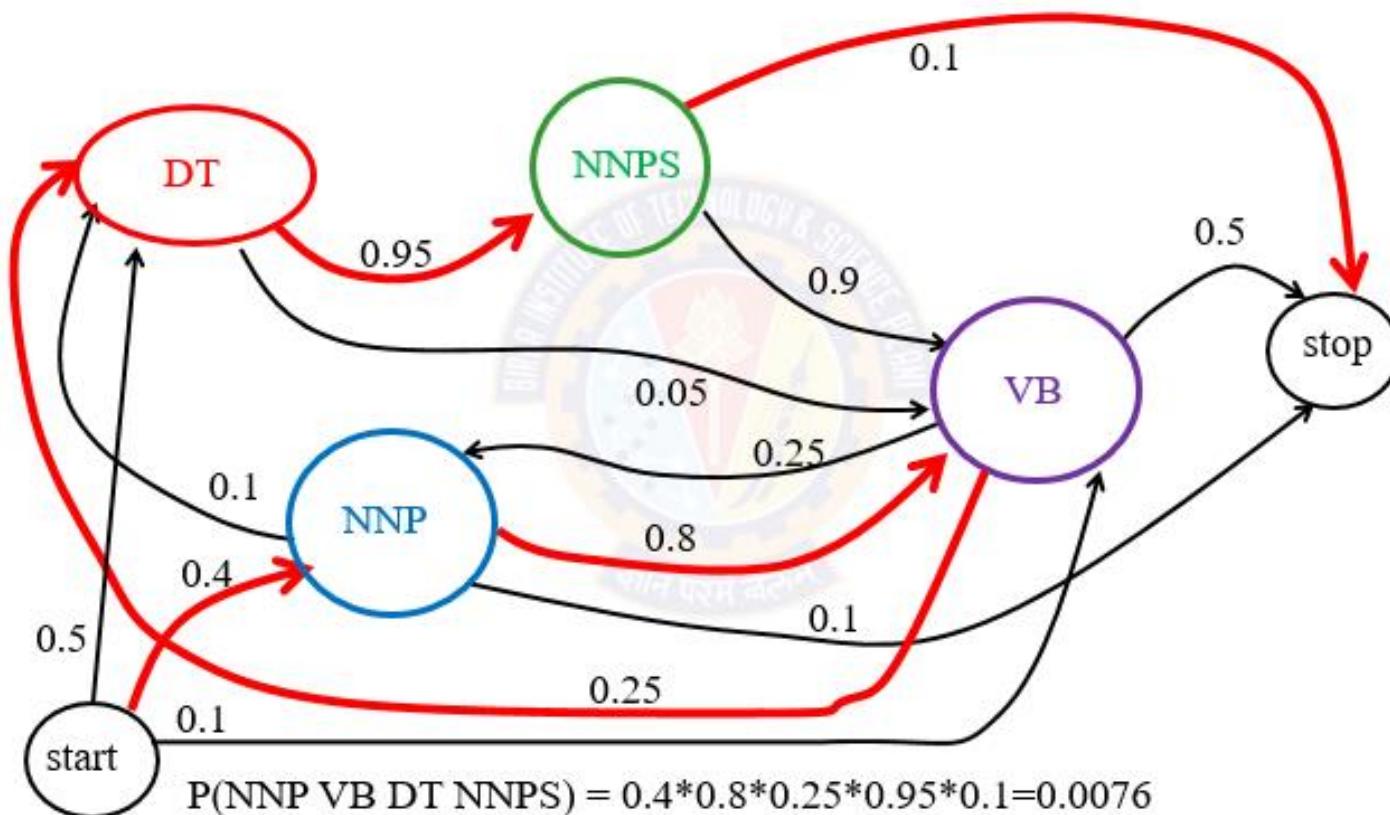
$$\pi_i = P(t_1 = i) \quad 1 \leq i \leq N$$

# Markov Chain

- Formally, a Markov chain is specified by the following components:

|   |   |
|---|---|
| $Q = q_1 q_2 \dots q_N$<br>a set of N states  | A set of N states   |
| $A = a_{11} a_{12} \dots a_{n1} \dots a_{nn}$ | A transition probability matrix A, each $a_{ij}$ representing the probability of moving from state i to state j,  |
| $\pi = \pi_1, \pi_2, \pi_3, \dots, \pi_n$     | An initial probability distribution over states.<br>$\pi_i$ is the probability that the Markov chain will start in state i.<br>Some states j may have $\pi_j = 0$ , meaning that they cannot be initial states. |





# Example

# ipie

## Emission Matrix

|           | * | DT  | NN<br>S | VB  | NN | IN | STOP |
|-----------|---|-----|---------|-----|----|----|------|
| *         | 1 |     |         |     |    |    |      |
| the       |   | 3/4 |         |     |    |    |      |
| employees |   |     | 3/4     |     |    |    |      |
| pass      |   |     |         | 2/4 |    |    |      |
| an        |   | 1/4 |         |     |    |    |      |
| exam      |   |     |         |     | 1  |    |      |
| wait      |   |     |         | 1/4 |    |    |      |
| for       |   |     |         |     |    | 1  |      |
| employers |   |     | 1/4     |     |    |    |      |
| fire      |   |     |         | 1/4 |    |    |      |
| .         |   |     |         |     |    |    | 1    |

This is our corpus

## Tag Translation Matrix

## SECOND TAG

|      | * | DT  | NNS | VB  | NN           | IN           | STOP |
|------|---|-----|-----|-----|--------------|--------------|------|
| *    |   | 2/3 | 1/3 |     | $\Sigma = 1$ |              |      |
| DT   |   |     | 2/4 | 1/4 | 1/4          | $\Sigma = 1$ |      |
| NNS  |   |     |     | 3/4 |              |              | 1/4  |
| VB   |   | 1/4 | 1/4 |     |              | 1/4          | 1/4  |
| NN   |   |     |     |     |              |              | 1    |
| IN   |   | 1   |     |     |              |              |      |
| STOP |   |     |     |     |              |              |      |

S1: the Employees pass an exam .  
T1: DT NNS VB DT NN STOP

T1: DT NNS VB DT NN STOP

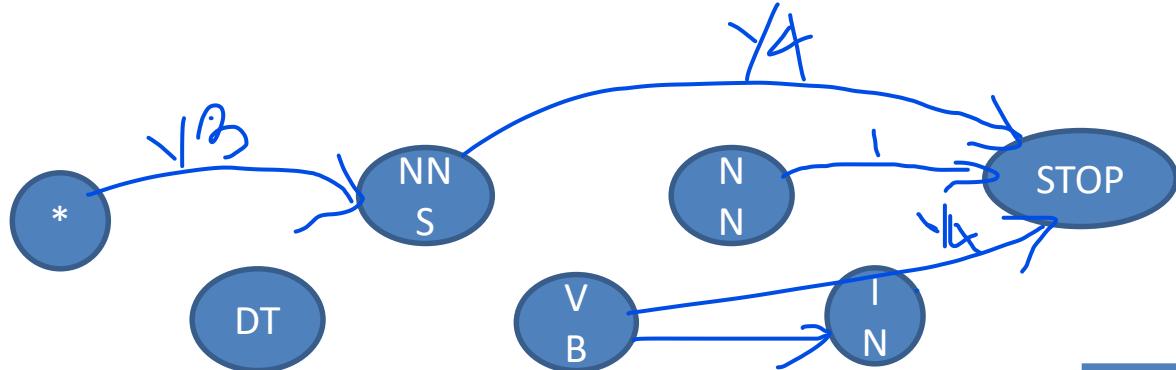
S2: the employees wait for the pass .

T2: DT NNS VB IN DT VB STOP

### S3: employers fire employees .

T3: NNS VB NNS STOP

# Transition diagram



Tag Translation Matrix

SECOND TAG

|           | *   | DT  | NNS | VB  | NN | IN  | STOP |
|-----------|-----|-----|-----|-----|----|-----|------|
| FIRST TAG |     |     |     |     |    |     |      |
| *         | 2/3 | 1/3 |     |     |    |     |      |
| DT        |     | 2/4 | 1/4 | 1/4 |    |     |      |
| NNS       |     |     |     | 3/4 |    |     | 1/4  |
| VB        | 1/4 | 1/4 |     |     |    | 1/4 | 1/4  |
| NN        |     |     |     |     |    |     | 1    |
| IN        | 1   |     |     |     |    |     |      |
| STOP      |     |     |     |     |    |     |      |

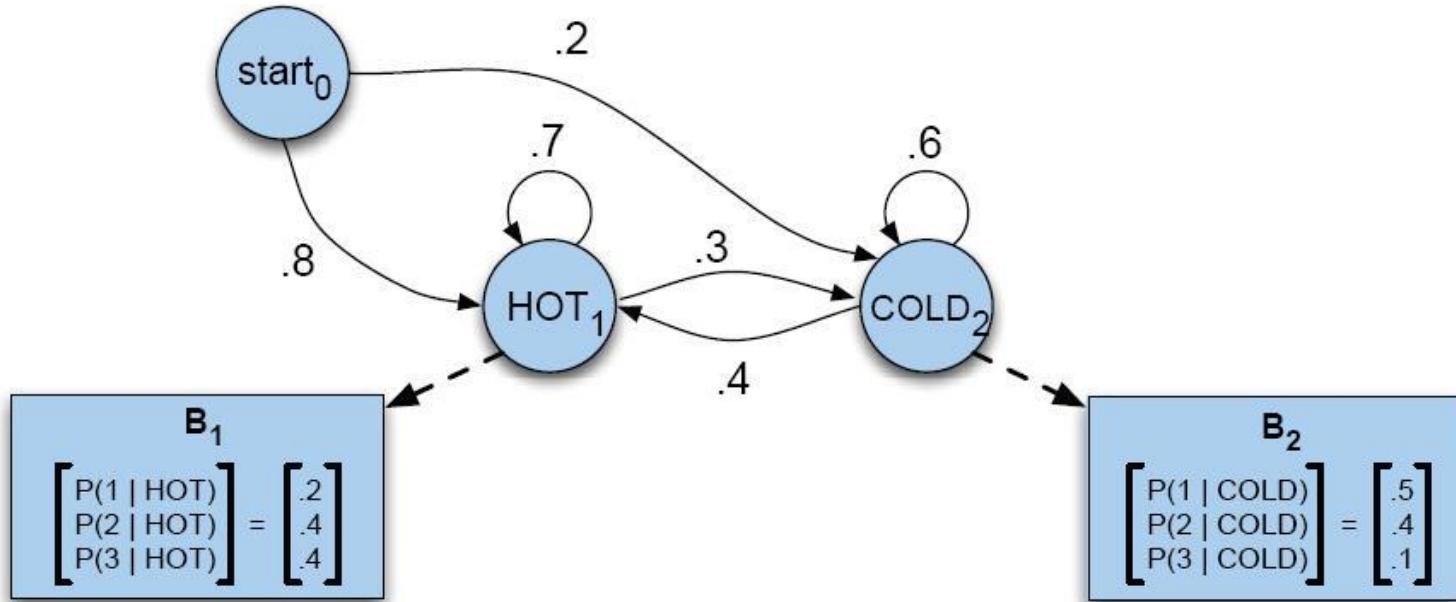
# HMMs for Ice Cream

- 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



# Eisner Task

- Given
    - Ice Cream Observation Sequence: 1,2,3,2,2,2,3...
  - Produce:
    - Hidden Weather Sequence:  
H,C,H,H,H,C, C...
- 

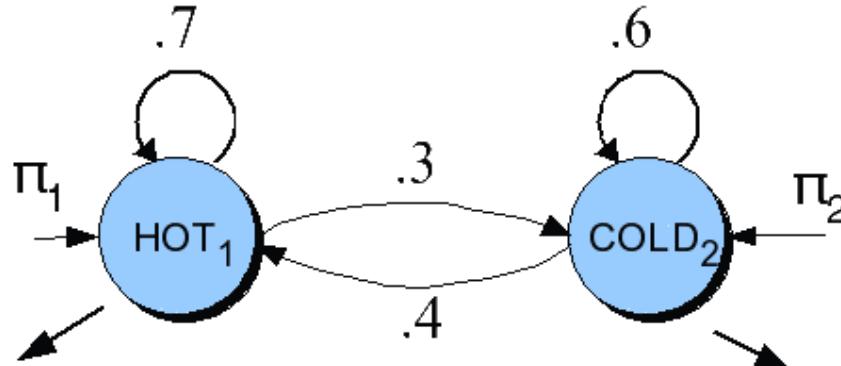


What's the state sequence for the observed sequence  
"1 3 1"?

$$P(H \mid H \mid 1 \ 3 \ 1) = ?$$

# HMM for Ice Cream

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



$$B_1 \begin{bmatrix} P(1 \mid HOT) \\ P(2 \mid HOT) \\ P(3 \mid HOT) \end{bmatrix} = \begin{bmatrix} .2 \\ .4 \\ .4 \end{bmatrix}$$

$$B_2 \begin{bmatrix} P(1 \mid COLD) \\ P(2 \mid COLD) \\ P(3 \mid COLD) \end{bmatrix} = \begin{bmatrix} .5 \\ .4 \\ .1 \end{bmatrix}$$

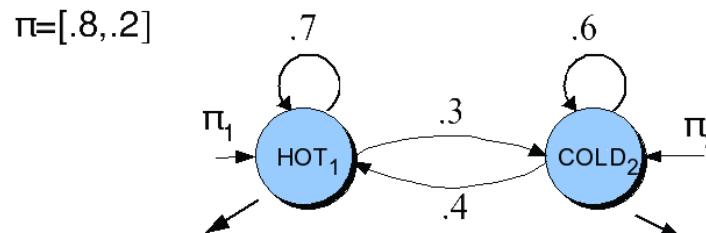
# Ice Cream HMM

- Let's just do **131** as the sequence
  - How many underlying state (hot/cold) sequences are there?

calculate for each and find the highest

2  
3

HHH  
HHC  
HCH  
HCC  
CCC  
CCH  
CHC  
CHH



$$B_1 \begin{bmatrix} P(1 \mid \text{HOT}) \\ P(2 \mid \text{HOT}) \\ P(3 \mid \text{HOT}) \end{bmatrix} = \begin{bmatrix} 0.2 \\ 0.4 \\ 0.4 \end{bmatrix}$$

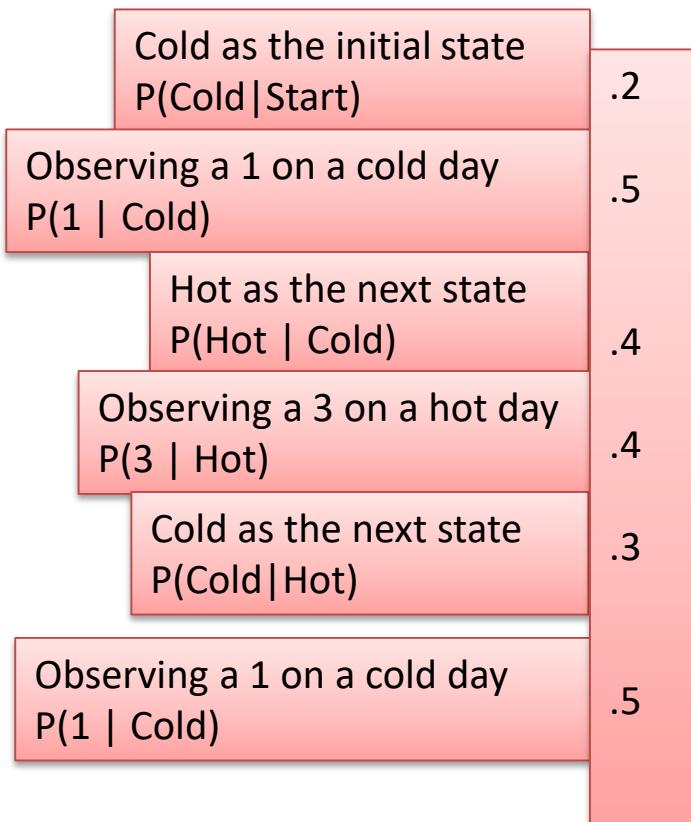
$$B_2 \begin{bmatrix} P(1 \mid \text{COLD}) \\ P(2 \mid \text{COLD}) \\ P(3 \mid \text{COLD}) \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.4 \\ 0.1 \end{bmatrix}$$

Argmax P(sequence | 1 3 1)

- How do you pick the right one?

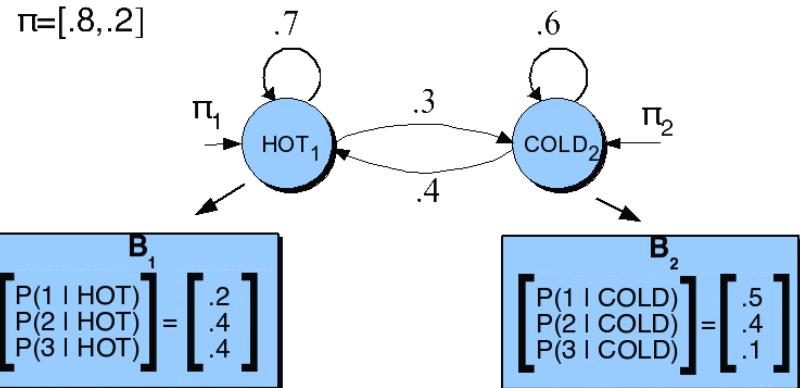
# Ice Cream HMM

Let's just do 1 sequence:



$$P(\text{CHC} | \text{Start}) = P(\text{C} | *) P(\text{H} | \text{C}) * P(\text{C} | \text{H})$$

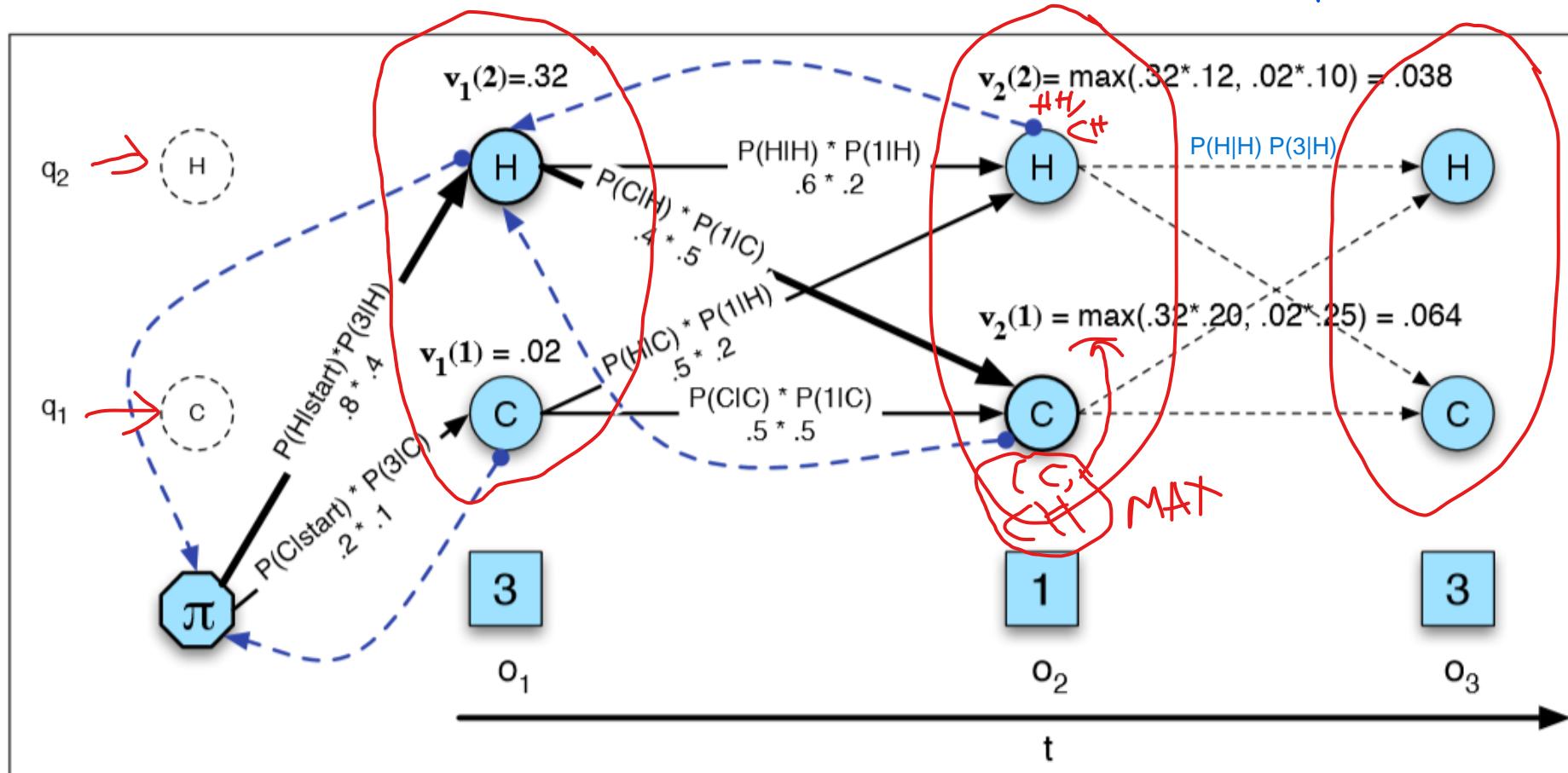
$* P(1 | \text{C}) P(3 | \text{H}) P(1 | \text{C})$



$$\begin{aligned}
 & P(\text{C H C}) \\
 &= 0.2 * 0.5 * 0.4 * 0.4 * 0.3 * 0.5 \\
 &= 0.0024
 \end{aligned}$$

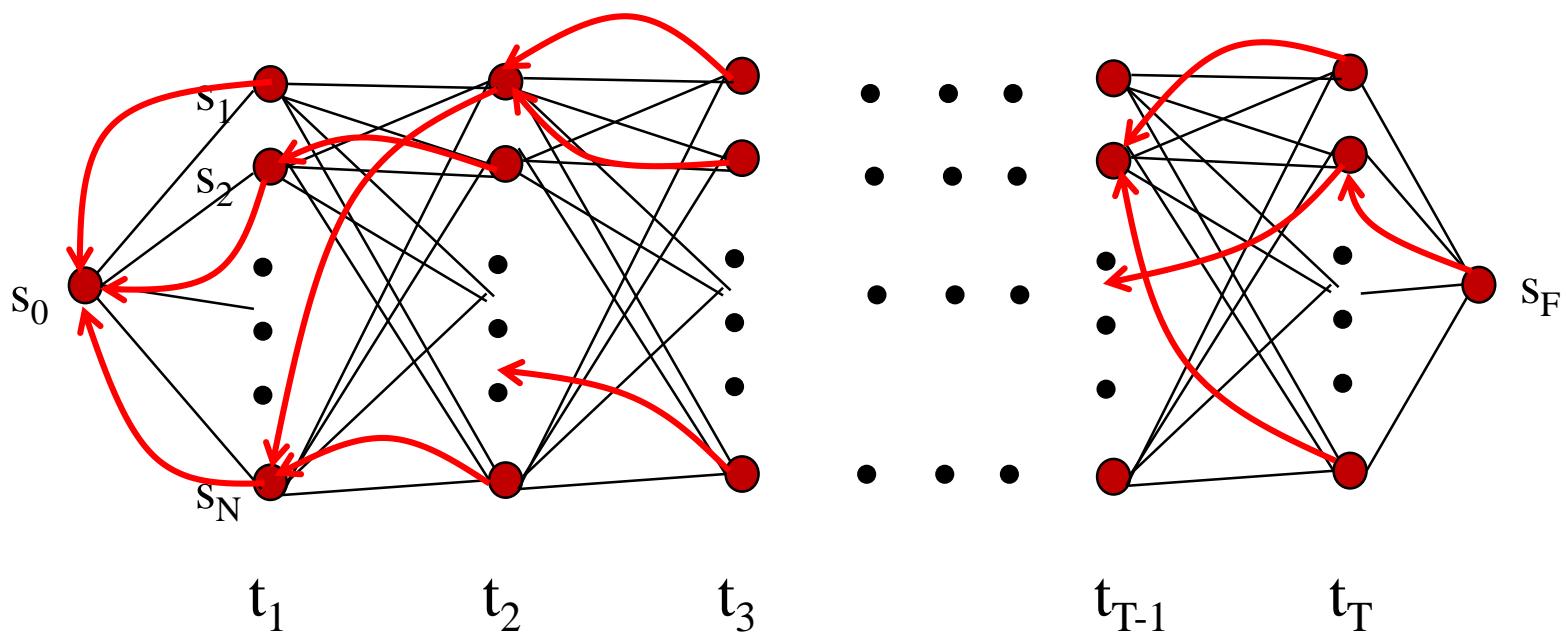
# Viterbi Example 2: Ice Cream

313 sequence now

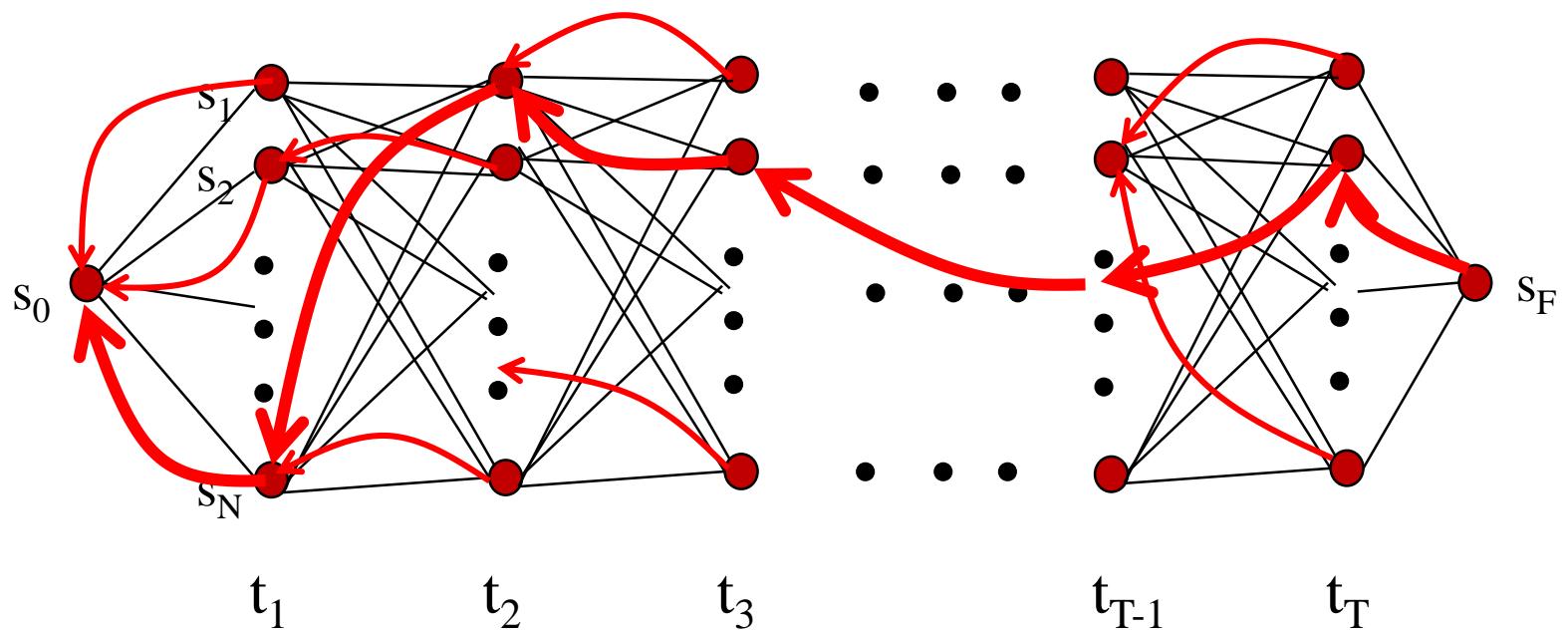


**Figure A.10** The Viterbi backtrace. As we extend each path to a new state account for the next observation we keep a backpointer (shown with broken lines) to the best path that led us to this state.

# Viterbi Backpointers



# Viterbi Backtrace



Most likely Sequence:  $s_0 s_N s_1 s_2 \dots s_2 s_F$

# 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 algorithm

- Create a table V with N+2 rows and T columns:
  - N – the number of states/tags
  - T – the length of the sequence/sentence
- Initialize the first column
- For each tag t in the tagset compute:

$$V[t, 1] = P(t|start)P(w_1|t)$$

- For each column j = 2 to T in the table V:
  - For each tag t in the tagset compute:

$$V[t, j] = \max_{t'} V[t', j - 1]P(t|t')P(w_j|t)$$

# Viterbi Example 1: POS Tagging

Example: I want to race.

Find POS tag?

|                  | <b>VB</b> | <b>TO</b> | <b>NN</b> | <b>PPSS</b> |
|------------------|-----------|-----------|-----------|-------------|
| <b>&lt;S&gt;</b> | .019      | .0043     | .041      | .067        |
| <b>VB</b>        | .0038     | .035      | .047      | .0070       |
| <b>TO</b>        | .83       | 0         | .00047    | 0           |
| <b>NN</b>        | .0040     | .016      | .087      | .0045       |
| <b>PPSS</b>      | .23       | .00079    | .0012     | .00014      |

**Figure 5.15** Tag transition probabilities (the  $a$  array,  $p(t_i|t_{i-1})$ ) computed from the 87-tag Brown corpus without smoothing. The rows are labeled with the conditioning event; thus  $P(PPSS|VB)$  is .0070. The symbol  $<S>$  is the start-of-sentence symbol.

|             | <b>I</b> | <b>want</b> | <b>to</b> | <b>race</b> |
|-------------|----------|-------------|-----------|-------------|
| <b>VB</b>   | 0        | .0093       | 0         | .00012      |
| <b>TO</b>   | 0        | 0           | .99       | 0           |
| <b>NN</b>   | 0        | .000054     | 0         | .00057      |
| <b>PPSS</b> | .37      | 0           | 0         | 0           |

**Figure 5.16** Observation likelihoods (the  $b$  array) computed from the 87-tag Brown corpus without smoothing.

# Viterbi Algorithm Example

---

Algorithm first creates N or four state columns.

- First column corresponds to the observation of the first word “I”,
- the second to the second word “want”,
- the third to the third word “to”, and
- the fourth to the fourth word “race”

Begin by setting the Viterbi value in each cell to the product of the transition probability (into it from the state state) and the observation probability (of the first word)

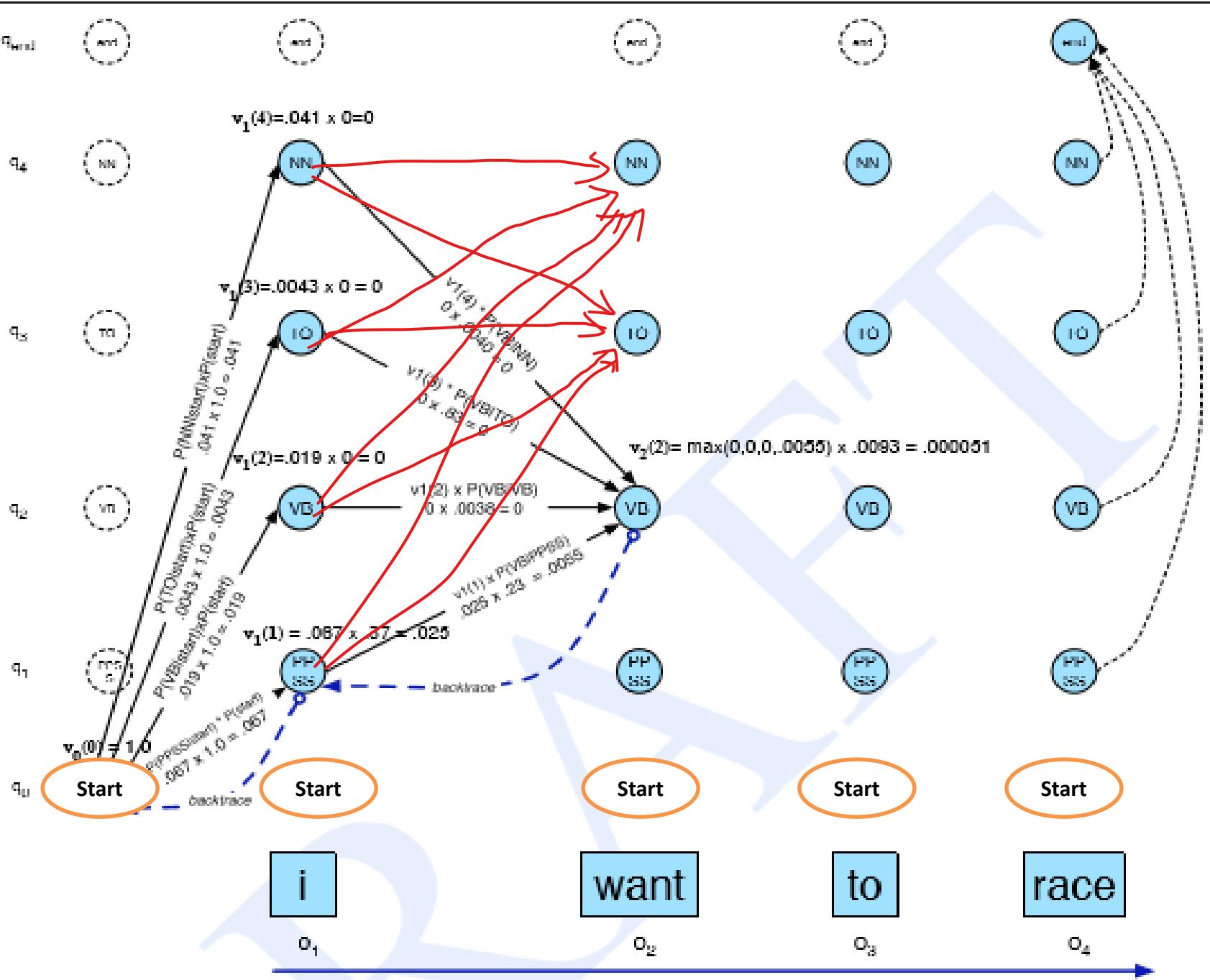
# Viterbi Algorithm Example

---

- For each state  $q_j$  at time  $t$ , the value Viterbi  $[s,t]$  is computed by taking the maximum over the extensions of all the paths that lead to the current cell, following the following equation:

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

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



# Example 3 : POS Tagging

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      |           |       |       |      |

$$\begin{aligned} V(\text{Noun}, \text{the}) &= P(\text{Noun}|<\text{S}>)P(\text{the}|\text{Noun}) = .3 \times 0 = 0 \\ V(\text{Verb}, \text{the}) &= P(\text{Verb}|<\text{S}>)P(\text{the}|\text{Verb}) = .1 \times 0 = 0 \\ V(\text{Det}, \text{the}) &= P(\text{Det}|<\text{S}>)P(\text{the}|\text{Det}) = .3 \times .7 = .21 \\ V(\text{Prep}, \text{the}) &= P(\text{Prep}|<\text{S}>)P(\text{the}|\text{Prep}) = .2 \times 0 = 0 \\ V(\text{Adv}, \text{the}) &= P(\text{Adv}|<\text{S}>)P(\text{the}|\text{Adv}) = .2 \times 0 = 0 \end{aligned}$$

We have to calculate same way as "the" for all other words (doctor, is, in, STOP)

Start with word "the" then "doctor", "is", "in", STOP

# Example (Contd..)

$$V(\text{Noun}, \text{doctor}) = \max_{t'} V(t', \text{the})XP(\text{Noun}|t')X P(\text{doctor}|\text{Noun}) \\ = \max \{0, 0, .21 (.9 \times .4), 0, 0\} = .0756$$

$$V(\text{Verb}, \text{doctor}) = \max_{t'} V(t', \text{the})XP(\text{Verb}|t')X P(\text{doctor}|\text{Verb}) \\ = \max \{0, 0, .21(.01 \times .1), 0, 0\} = .00021$$

|      | w1=the | w2=doctor | w3=is  | w4=in | STOP |
|------|--------|-----------|--------|-------|------|
| Noun | 0      | .0756     |        |       |      |
| Verb | 0      |           | .00021 |       |      |
| Det  | .21    |           | 0      |       |      |
| Prep | 0      |           | 0      |       |      |
| Adv  | 0      |           | 0      |       |      |

# Backtracking the Viterbi Matrix

|      | w1=the | w2=doctor | w3=is   | w4=in   | STOP    |
|------|--------|-----------|---------|---------|---------|
| Noun | 0      | .0756     | .001512 | 0       |         |
| Verb | 0      | .00021    | .027216 | 0       |         |
| Det  | .21    | 0         | 0       | 0       | .005443 |
| Prep | 0      | 0         | 0       | .005443 |         |
| Adv  | 0      | 0         | 0       | .000272 |         |

Det    Noun    Verb    Prep  
 Highest is noun

# 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,

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

# 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

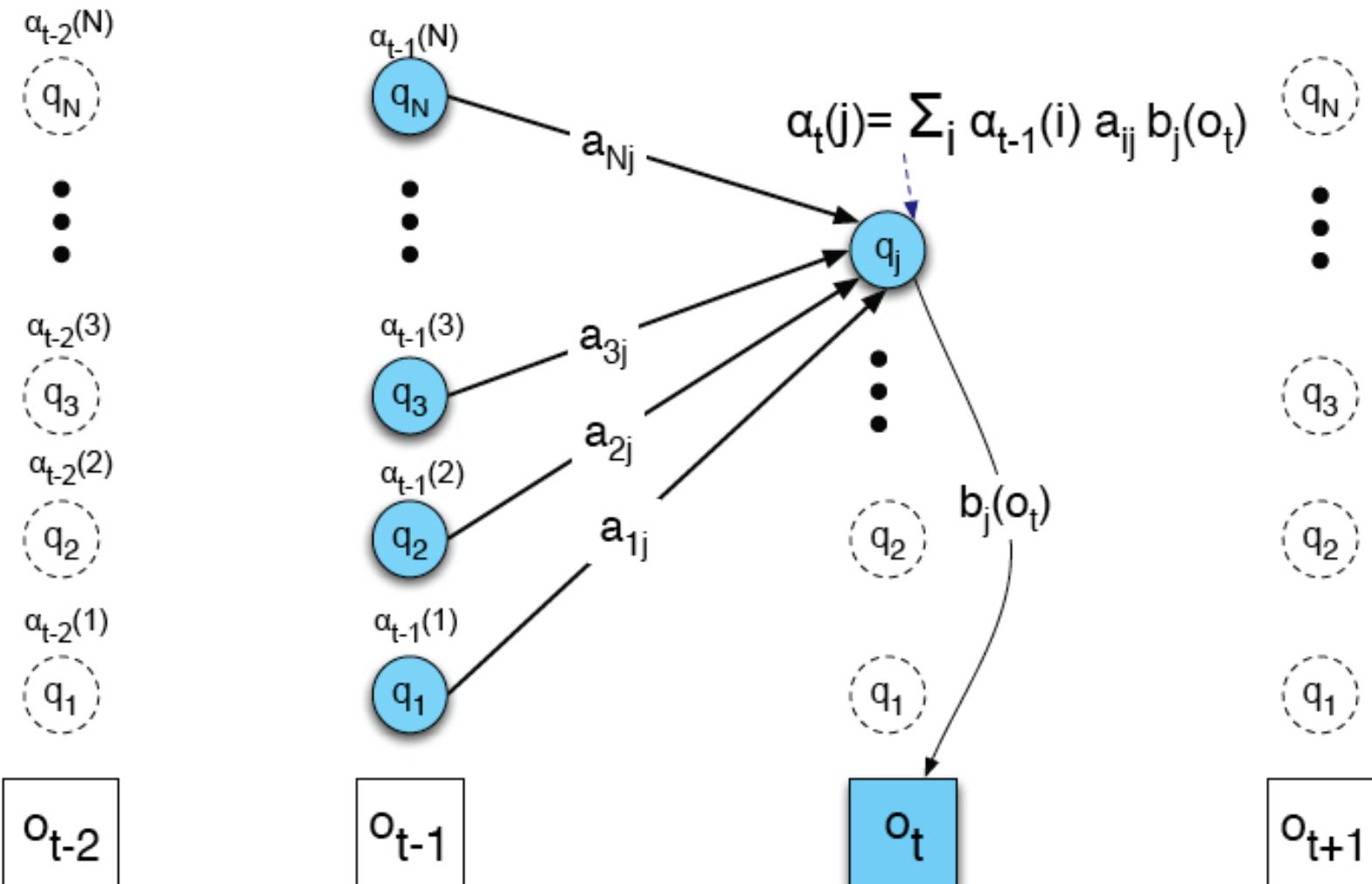
**for** each state  $s$  **from** 1 **to**  $N$  **do**

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

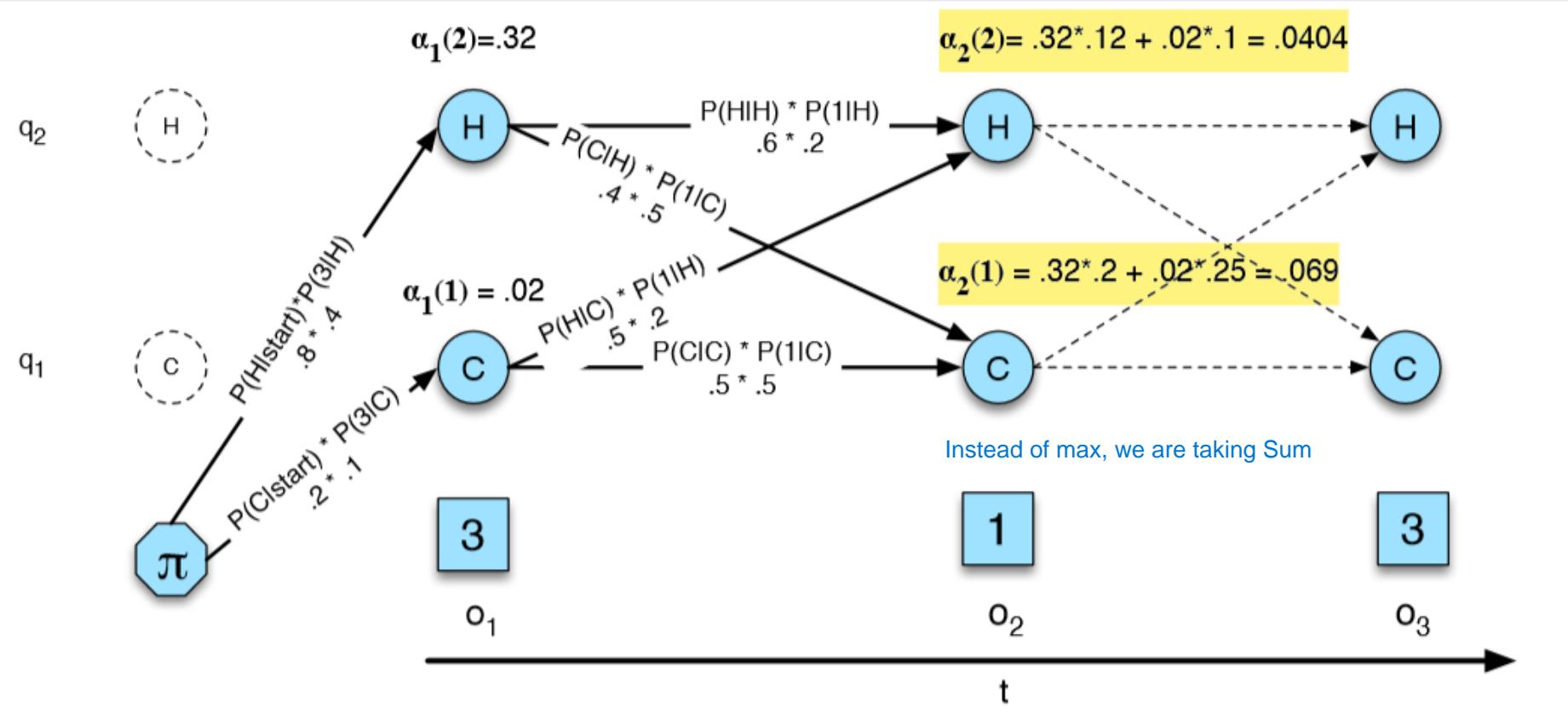
$forward[q_F, T] \leftarrow \sum_{s=1}^N forward[s, T] * a_{s,q_F}$  ; termination step

**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:  $\alpha_t(j) = P(o_1, o_2 \dots o_t, q_t = j | \lambda)$ .

# Bidirectionality

---

- One problem with the HMM models as presented is that they are exclusively run left-to-right.
- Viterbi algorithm still allows present decisions to be influenced indirectly by future decisions

# 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.

# Issues with HMM

---

- Unknown Words
  - We do not have the probabilities
    - Use smoothing
- Limited Context
  - Use trigrams/ 4 grams etc.
  - Sparsity

# Maximum Entropy Markov Model

---

- Identify heterogeneous set of features
  - tag given, the previous tag or the word given, the tag you can use, any other features which may contribute to the choice of part of speech tags.
- 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  $W = w_1^n$  and the sequence of tags  $T = t_1^n$
- In an HMM to compute the best tag sequence that maximizes  $P(T|W)$  we rely on Bayes' rule and the likelihood  $P(W|T)$ :

$$\begin{aligned}\hat{T} &= \operatorname{argmax}_T P(T|W) \\ &= \operatorname{argmax}_T P(W|T)P(T) \\ &= \operatorname{argmax}_T \prod_i P(\text{word}_i|\text{tag}_i) \prod_i P(\text{tag}_i|\text{tag}_{i-1})\end{aligned}$$

# Maximum Entropy Markov Model

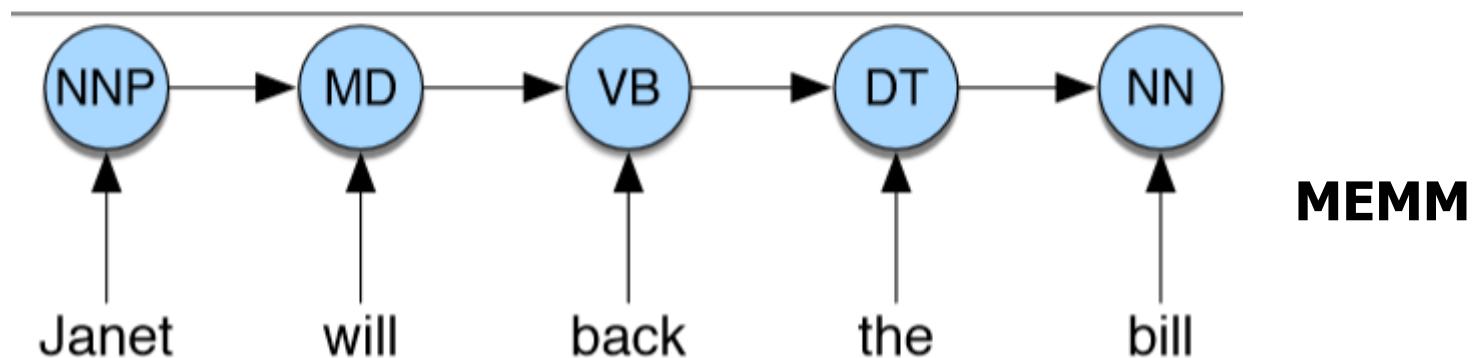
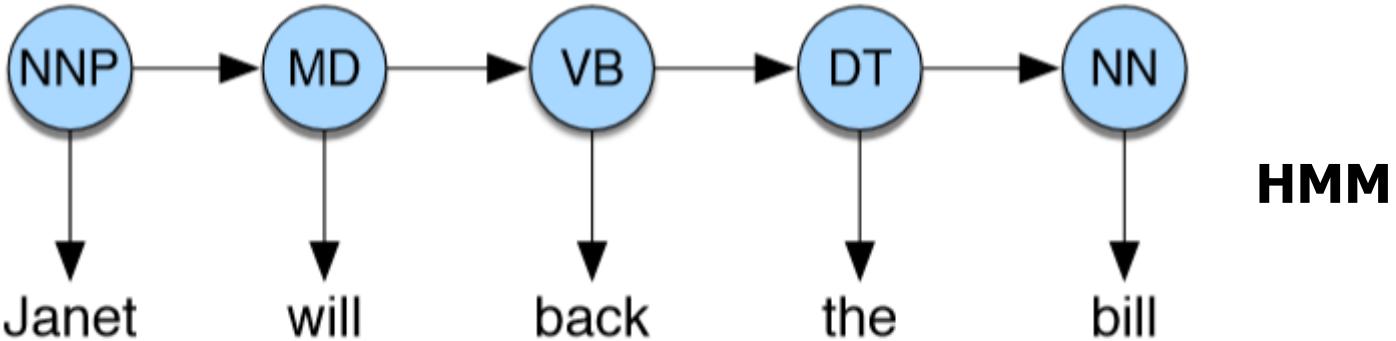
---

- In an MEMM, by contrast, we compute the posterior  $P(T|W)$  directly, training it to discriminate among the possible tag sequences:

$$\begin{aligned}\hat{T} &= \operatorname{argmax}_T P(T|W) \\ &= \operatorname{argmax}_T \prod_i P(t_i|w_i, t_{i-1})\end{aligned}$$

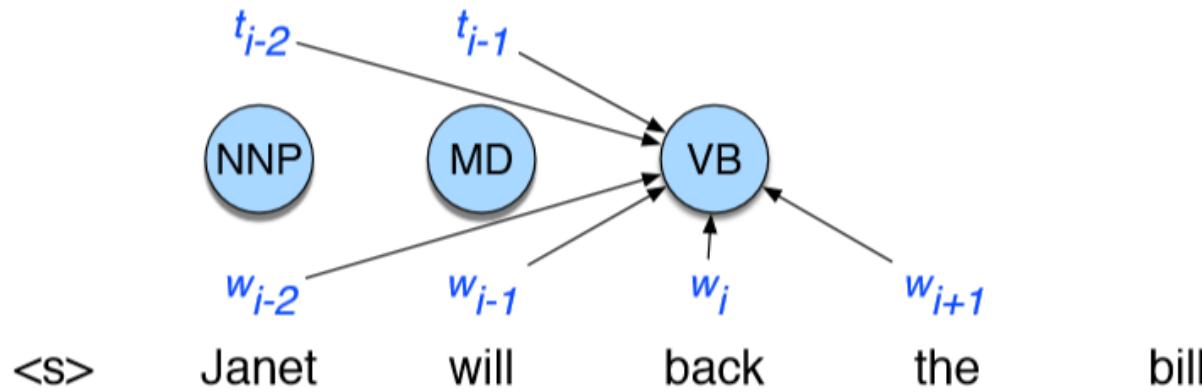
# Maximum Entropy Markov Model

- 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).



# 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  $w_{i+1} = \text{the}$

# Decoding and 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 state j

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \leq j \leq N, 1 < t \leq T$$

- In HMM

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) P(s_j|s_i) P(o_t|s_j) \quad 1 \leq j \leq N, 1 < t \leq T$$

- In MEMM

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) P(s_j|s_i, o_t) \quad 1 \leq j \leq N, 1 < t \leq T$$

---

# 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.

# 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>
- <https://www.nltk.org/book/ch02.html>
- <https://towardsdatascience.com/pos-tagging-using-crfs-ea430c5fb78b>

# References

- <https://github.com/nadavo/MEMM-POS-Tagger>
- <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.352.9257&rep=rep1&type=pdf>
- [https://www.alibabacloud.com/blog/hmm-memm-and-crf-a-comparative-analysis-of-statistical-modeling-methods\\_592049](https://www.alibabacloud.com/blog/hmm-memm-and-crf-a-comparative-analysis-of-statistical-modeling-methods_592049)
- <https://towardsdatascience.com/pos-tagging-using-rnn-7f08a522f849>

## POS tagging in Indian Languages

- <https://www.nltk.org/book/ch05.html>

# 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  $B$ .

- **Baum-Welch = Forward-Backward Algorithm**  
(Baum 1972)
- 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

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

Assume an HMM with  $N$  states.

Randomly set its parameters  $\lambda = (A, B)$

(making sure they represent legal distributions)

Until converge (i.e.  $\lambda$  no longer changes) do:

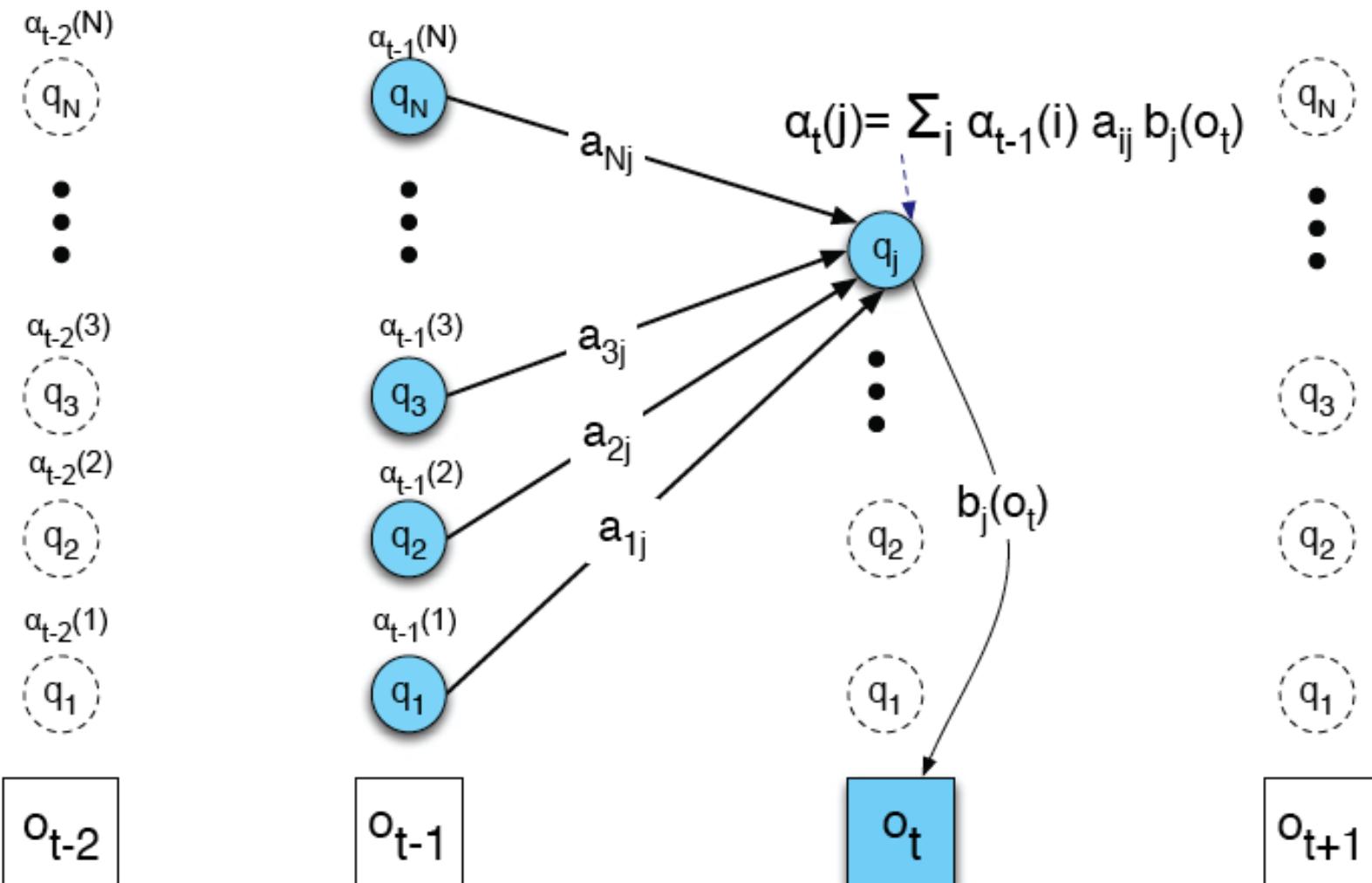
E Step: Use the forward/backward procedure to determine the probability of various possible state sequences for generating the training data

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

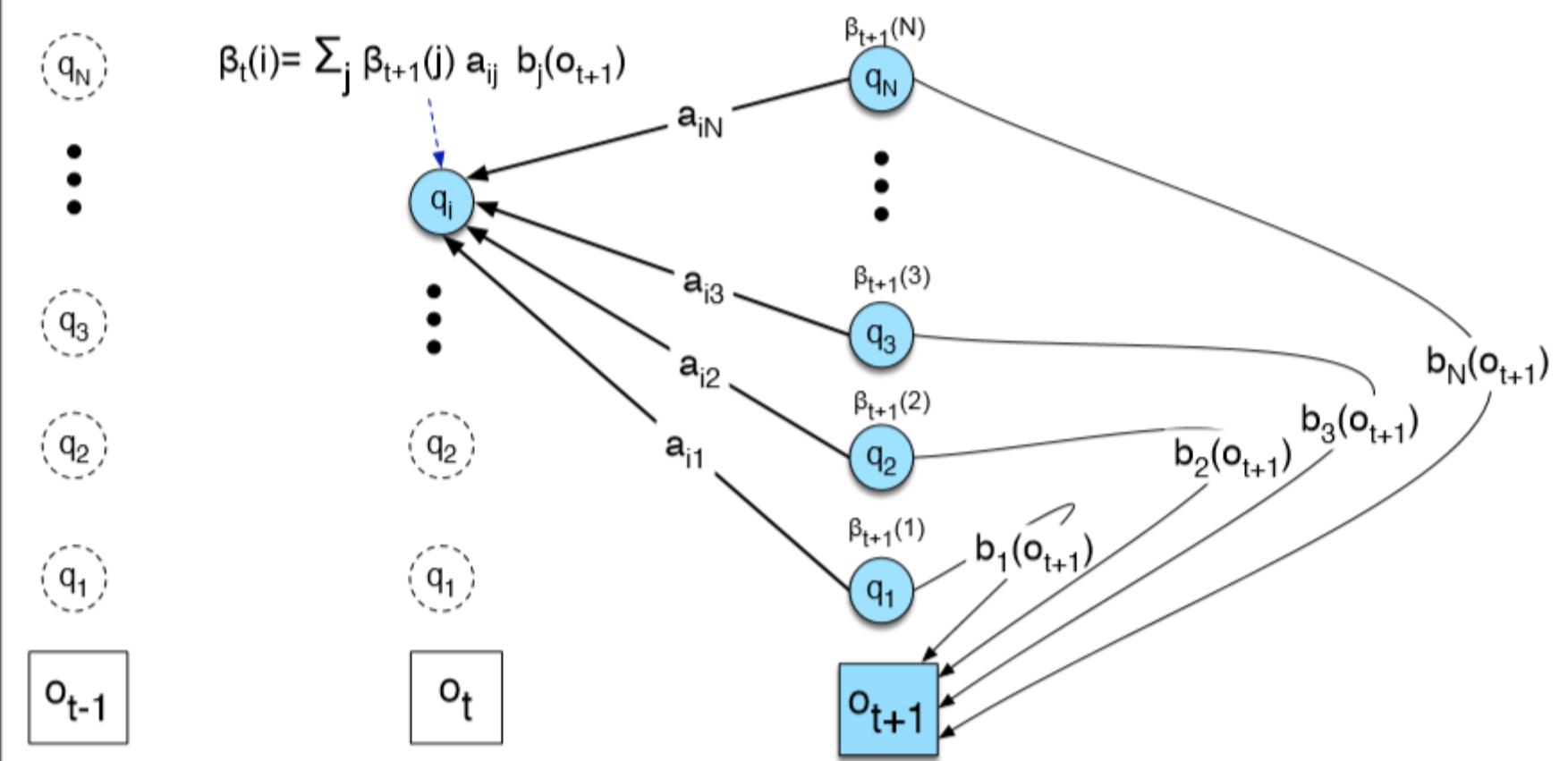
# EM Properties

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

# Visualizing Forward



# Backward Probabilities



**Figure A.11** The computation of  $\beta_t(i)$  by summing all the successive values  $\beta_{t+1}(j)$  weighted by their transition probabilities  $a_{ij}$  and their observation probabilities  $b_j(o_{t+1})$ . Start and end states not shown.

# Intuition for re-estimation of $a_{ij}$

- We will estimate  $\hat{a}_{ij}$  via this intuition:

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

- intuition:
  - If we knew this probability for *each* time  $t$ , we could sum over all  $t$  to get expected value for  $i \rightarrow j$ .

# Intuition for re-estimation of $a_{ij}$

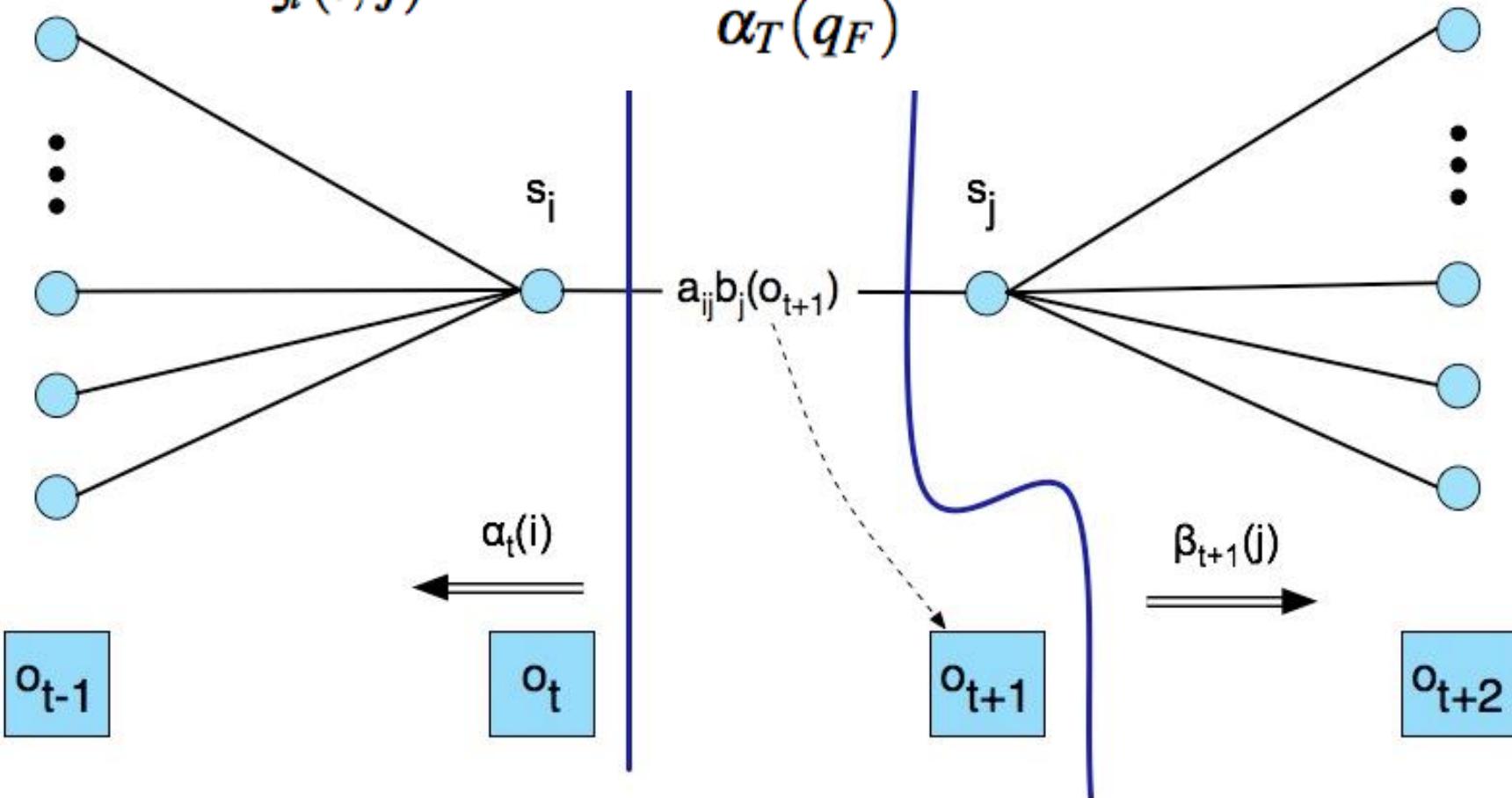
- Assume we had some estimate of the probability that a given transition  $i \rightarrow j$  was taken at a particular point in time  $t$  in the observation sequence.
- If we knew this probability for each particular time  $t$ , we could sum over all times  $t$  to estimate the total count for the transition  $i \rightarrow j$ .

More formally, let's define the probability  $\xi_t$  as the probability of being in state  $i$  at time  $t$  and state  $j$  at time  $t + 1$ , given the observation sequence and of course the model:

$$\xi_t(i, j) = P(q_t = i, q_{t+1} = j | O, \lambda) \quad (\text{A.17})$$

# Intuition for re-estimation of $a_{ij}$

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



# Re-estimating $a_{ij}$

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

- The expected number of transitions from state  $i$  to state  $j$  is the sum over all  $t$  of  $\xi$
- The total expected number of transitions out of state  $i$  is the sum over all transitions out of state  $i$
- Final formula for reestimated  $a_{ij}$

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

# The Forward-Backward Alg

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

**initialize**  $A$  and  $B$

**iterate** until convergence

## E-step

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{\alpha_T(q_F)} \quad \forall t \text{ and } j$$

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

## M-step

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

**return**  $A, B$



# Natural Language Processing

## DSECL ZG565



**BITS** Pilani  
Hyderabad Campus

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



## **Session 7**

### **Date – 1<sup>st</sup> July 2023**

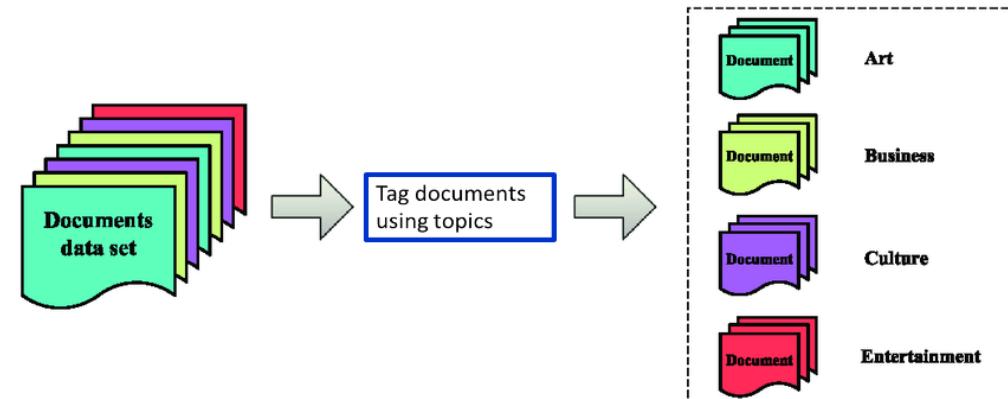
These slides are prepared by the instructor, with grateful acknowledgement of Prof. Luis G. Serrano, Prof. Andrew Ng and many others who made their course materials freely available online.

# Session Content

- 
- Objective of Latent Dirichlet Allocation (LDA)
  - Intuition behind LDA
  - LDA Generative model
  - Mathematical foundations for LDA : Bernoulli Trial, Binomial distribution, Multinomial distribution
  - Mathematical foundations for LDA : Beta distribution, Conjugate Prior and Dirichlet distributions
  - Dirichlet Visualization using Simplex
  - Probabilistic Graphical LDA Model
  - Mathematical modelling of LDA
  - Gibbs Sampling Algorithm
  - Case Study
  - Implementing LDA in Python
-

# Objectives of Topic Modelling

- Use these annotations to organize, summarize and search the documents.
- Topic Model can be defined as an unsupervised technique to discover topics across various text documents



# Sample output from the LDA

- Four topics learned from the S&P 500 stock market data
- Goal is to find groups of stocks that tend to move together.

| Topic 1   | Topic 2   | Topic 3   | Topic 4   |
|---|---|---|---|
| Southwestern Energy<br>Range Resources<br>Cabot Oil & Gas<br>EOG Resources<br>Chesapeake Energy<br>Pioneer Resources<br>Devon Energy<br>Peabody Energy<br>Anadarko Petroleum<br>Massey Energy | Penneys<br>Macys<br>Kohls<br>Nordstrom<br>Target<br>Limited<br>Lowes<br>Home Depot<br>American Express<br>Abercrombie | Capital One<br>BNY Mellon<br>Discover<br>Northern Trust<br>Janus<br>JPMorgan Chase<br>State Street<br>Wells Fargo<br>PPL<br>T. Rowe Price | Simon Property<br>Kimco Realty<br>Equity Residential<br>AvalonBay Communities<br>Apartment Investment<br>Vornado Realty Trust<br>Boston Properties<br>Public Storage<br>Host Hotels<br>HCP Inc. |

- The topic model does not provide any label to these group of words.

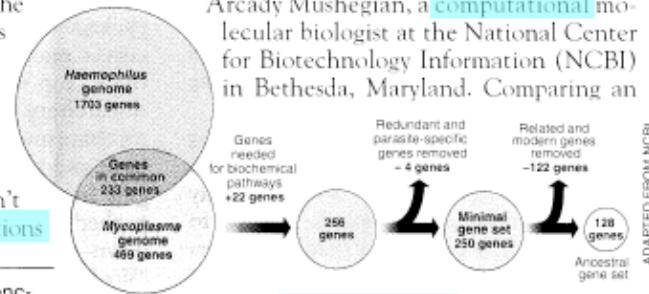
# Sample output from the LDA

## Seeking Life's Bare (Genetic) Necessities

COLD SPRING HARBOR, NEW YORK—How many genes does an organism need to survive? Last week at the genome meeting here,\* two genome researchers with radically different approaches presented complementary views of the basic genes needed for life. One research team, using computer analyses to compare known genomes, concluded that today's organisms can be sustained with just 250 genes, and that the earliest life forms required a mere 128 genes. The other researcher mapped genes in a simple parasite and estimated that for this organism, 800 genes are plenty to do the job—but that anything short of 100 wouldn't be enough.

Although the numbers don't match precisely, those predictions

"are not all that far apart," especially in comparison to the 75,000 genes in the human genome, notes Siv Andersson of Uppsala University in Sweden, who arrived at the 800 number. But coming up with a consensus answer may be more than just a genetic numbers game, particularly as more and more genomes are completely mapped and sequenced. "It may be a way of organizing any newly sequenced genome," explains Arcady Mushegian, a computational molecular biologist at the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Comparing an



**Stripping down.** Computer analysis yields an estimate of the minimum modern and ancient genomes.

\* Genome Mapping and Sequencing, Cold Spring Harbor, New York, May 8 to 12.

Genetics

Evolutionary biology

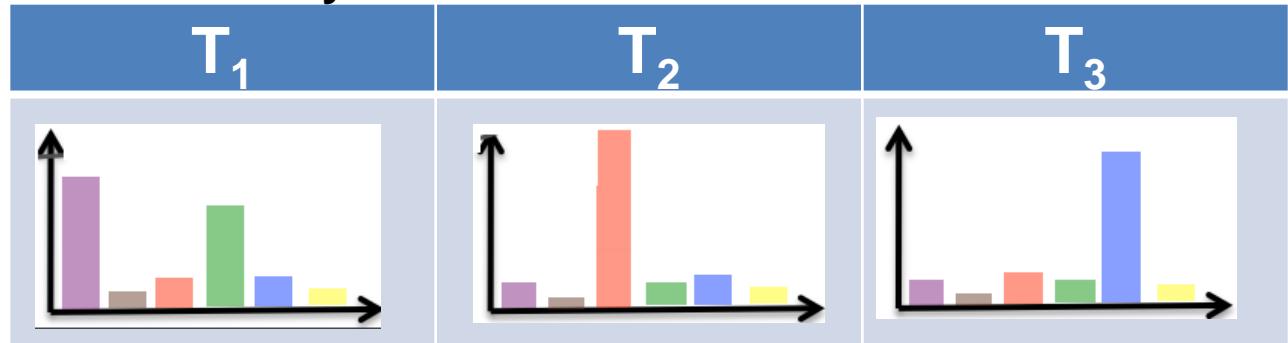
Data Analysis

# Overall schematic

**Corpus**

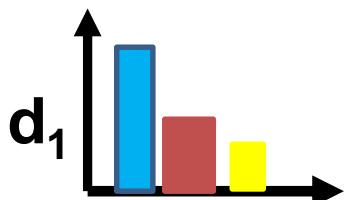


Each topic is defined as a Multinomial distribution over the vocabulary



Documents( $d_1 \dots d_n$ )

K-Topics (Hyper Parameter)

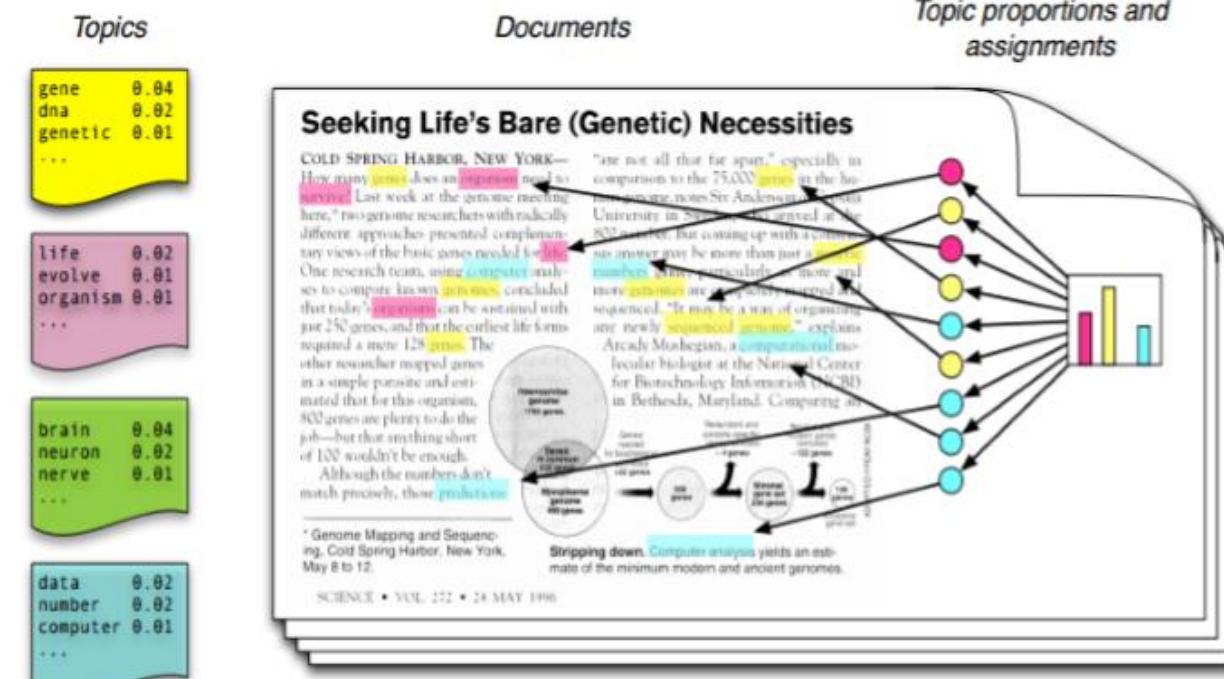


Each document has a distribution over K topics



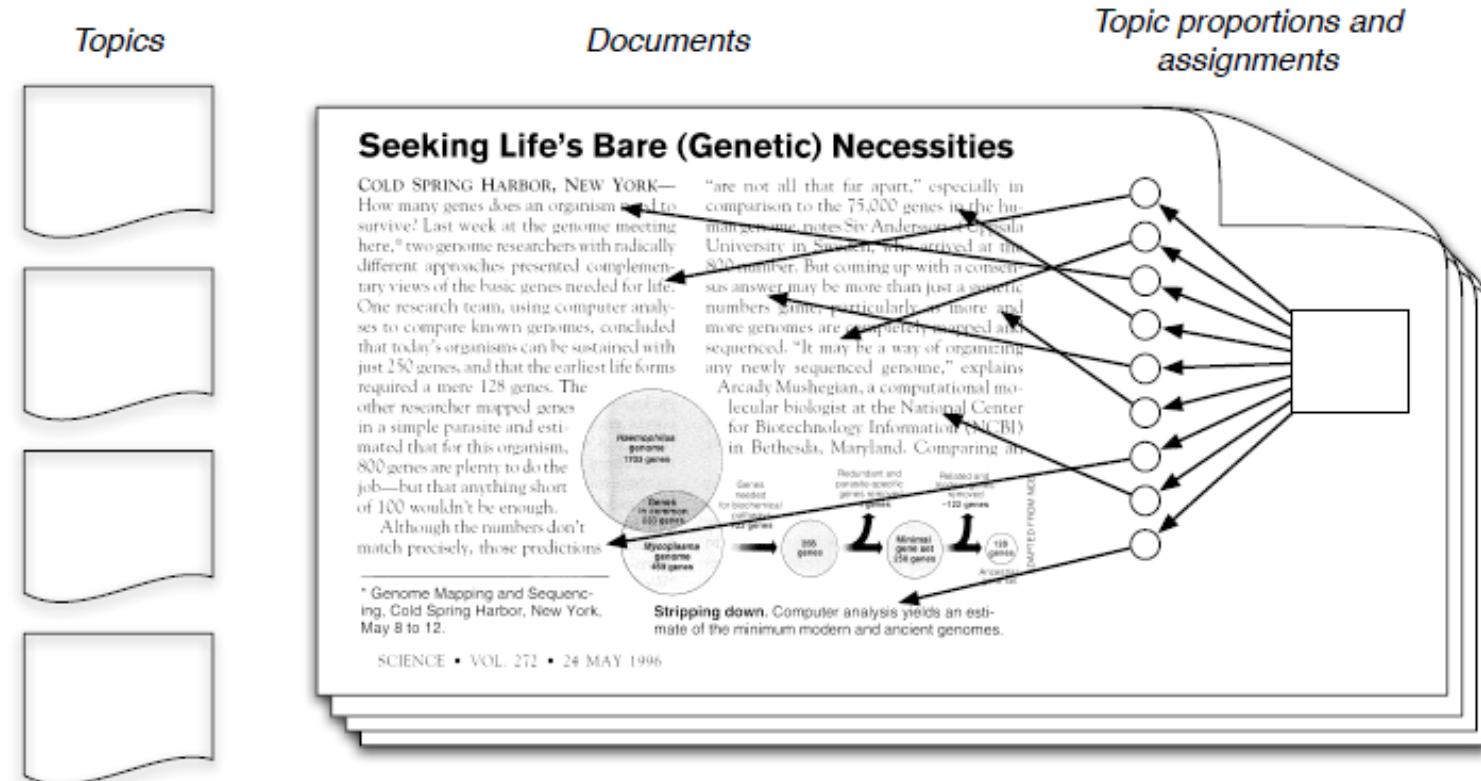
Vocabulary( $W_1 \dots W_m$ )

# LDA Generative model



- Each **topic** is a distribution over words
- Each **document** is a mixture of corpus-wide topics
- Each **word** is drawn from one of these topics
- We only observe the words within the documents and the other structure are **hidden variables**.

# The posterior distribution

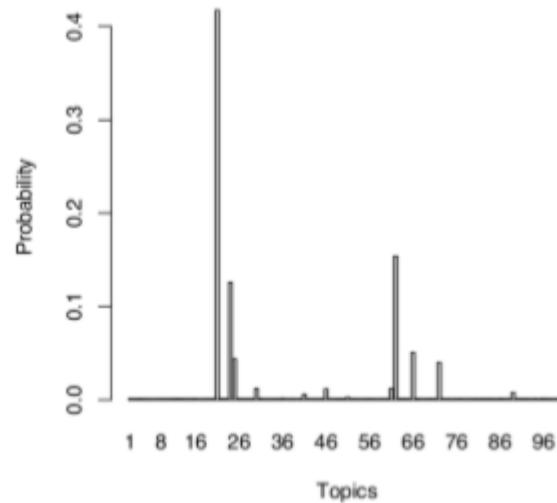


# LDA model

A 100-topic LDA model was fitted to **17,000 articles from the *Science* journal**.

At right are **the top 15 most frequent words** from the most frequent topics.

At left are the **inferred topic proportions** for the example article from previous slide.



| “Genetics”  | “Evolution”  | “Disease”    | “Computers” |
|-------------|--------------|--------------|-------------|
| human       | evolution    | disease      | computer    |
| genome      | evolutionary | host         | models      |
| dna         | species      | bacteria     | information |
| genetic     | organisms    | diseases     | data        |
| genes       | life         | resistance   | computers   |
| sequence    | origin       | bacterial    | system      |
| gene        | biology      | new          | network     |
| molecular   | groups       | strains      | systems     |
| sequencing  | phylogenetic | control      | model       |
| map         | living       | infectious   | parallel    |
| information | diversity    | malaria      | methods     |
| genetics    | group        | parasite     | networks    |
| mapping     | new          | parasites    | software    |
| project     | two          | united       | new         |
| sequences   | common       | tuberculosis | simulations |

# LDA Generative Process

LDA assumes that new documents are created in the following way:

1. Determine number of words in document
2. Choose a topic mixture for the document over a fixed set of topics (i.e. 20% topic A, 30% topic B, 50% topic C)
3. Generate the words in the document by:
  - First pick a topic based on the document's multinomial distribution above.
  - Next pick a word based on the topic's multinomial distribution.

# LDA Generative Process

---

-Say we have a group of articles and we assume that all of those articles can be characterized by three topics: Animals, Cooking, and Politics.

-Each of those topics can be described by the following words:

- Animals: dog, chicken, cat, nature, zoo
- Cooking: oven, food, restaurant, plates, taste, delicious
- Politics: Republican, Democrat, Congress, ineffective, divisive

-Say we want to generate a new document that is 80% about animals and 20% about cooking.

1. We choose the length of the article (say, 1000 words)
2. We choose a topic based on our specified mixture (so, out of our 1000 words, roughly 800 will come from the topic "animals")
3. We choose a word based on the word distribution for each topic (i.e.

# Intuition behind LDA

- Suppose you have a corpus of documents
- You want LDA to learn the topic representation of K topics in each document and the word distribution of each topic.
- LDA backtracks from the document level to identify topics that are likely to have generated the corpus.

# Intuition behind LDA

- Randomly assign each word in each document to one of the K topics.
- For each document d:
  - Assume that all topic assignments except for the current one are correct.
  - Calculate two proportions:
    1. Proportion of words in document d that are currently assigned to topic t =  $p(\text{topic } t \mid \text{document } d)$
    2. Proportion of assignments to topic t over all documents that come from this word w =  $p(\text{word } w \mid \text{topic } t)$
  - Multiply those two proportions and assign w a new topic based on that probability.  $p(\text{topic } t \mid \text{document } d) * p(\text{word } w \mid \text{topic } t)$
- Eventually we'll reach a steady state where assignments make sense

# Bernoulli Trial

- Any **single trial** with **two possible outcomes** can be modeled as a **Bernoulli trial**: team wins/loses, pitch is a strike/ball, coin comes up heads or tails, etc.
- A Bernoulli trial uses Bernoulli distribution to calculate the probability of either outcome.



**Bernoulli trial**

# Bernoulli: A Special Case of the Binomial Distribution



**Binomial Trial: Chance of getting n heads in a row(n=3)**



**Bernoulli Trial: Chance of getting a heads on a single flip**

# Bernoulli - Distribution Notation

- The probability mass function of the Bernoulli distribution is

$\theta = \text{head}$   
 $1 - \theta = \text{tail}$

$$f(x) = P(X=k) = \theta^k (1-\theta)^{1-k}, \quad k=\{0,1\}$$

- The only parameter of the bernoulli distribution is  $\theta$ , which defines the probability of success during a bernoulli trial.



# Binomial distribution

# Binomial distribution

If  $p$  is the probability of heads, the probability of getting exactly  $k$  heads in  $n$  independent yes/no trials is given by the binomial distribution  $Bin(n,p)$ :

$$\begin{aligned} P(k \text{ heads}) &= \binom{n}{k} p^k (1-p)^{n-k} \\ &= \frac{n!}{k!(n-k)!} p^k (1-p)^{n-k} \end{aligned}$$

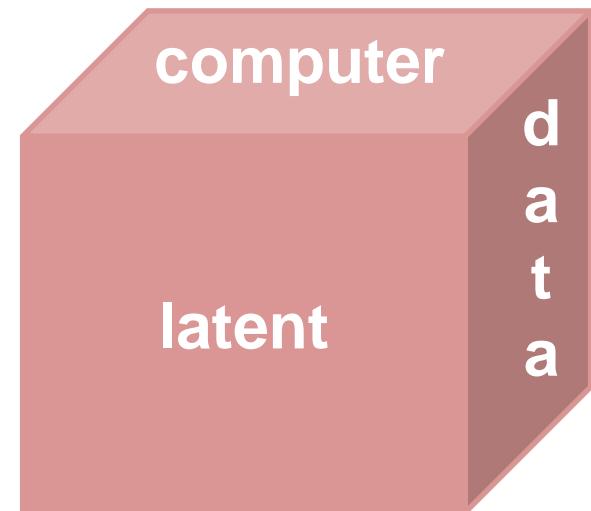
# Multinomial Distribution

- Suppose we roll our die of words having k sides(vocabulary) where each side takes probabilities  $\Theta_1, \dots, \Theta_k$  respectively.
- Probability mass function

$$f(x) = \frac{n!}{x_1!x_2!\cdots x_k!} \theta_1^{x_1} \theta_2^{x_2} \cdots \theta_k^{x_k}$$

k - number of sides on the die

n - number of times the die will be rolled



# Multinomial Distribution

Suppose that we observe an experiment that has  $k$  possible outcomes  $\{O_1, O_2, \dots, O_k\}$  independently  $n$  times.

Let  $p_1, p_2, \dots, p_k$  denote probabilities of  $O_1, O_2, \dots, O_k$  respectively.

Let  $X_i$  denote the number of times that outcome  $O_i$  occurs in the  $n$  repetitions of the experiment.

Then the joint probability function of the random variables  $X_1, X_2, \dots, X_k$  is

$$p(x_1, \dots, x_n) = \frac{n!}{x_1! x_2! \dots x_k!} p_1^{x_1} p_2^{x_2} \dots p_k^{x_k}$$

# Multinomial Distribution

Note:  $p_1^{x_1} p_2^{x_2} \cdots p_k^{x_k}$

is the probability of a sequence of length  $n$  containing

$x_1$  outcomes  $O_1$

$x_2$  outcomes  $O_2$

...

$x_k$  outcomes  $O_k$

$$\frac{n!}{x_1! x_2! \dots x_k!} = \binom{n}{x_1 \quad x_2 \quad \dots \quad x_k}$$

is the number of ways of choosing the positions for the  $x_1$  outcomes  $O_1$ ,  $x_2$  outcomes  $O_2$ , ...,  $x_k$  outcomes  $O_k$

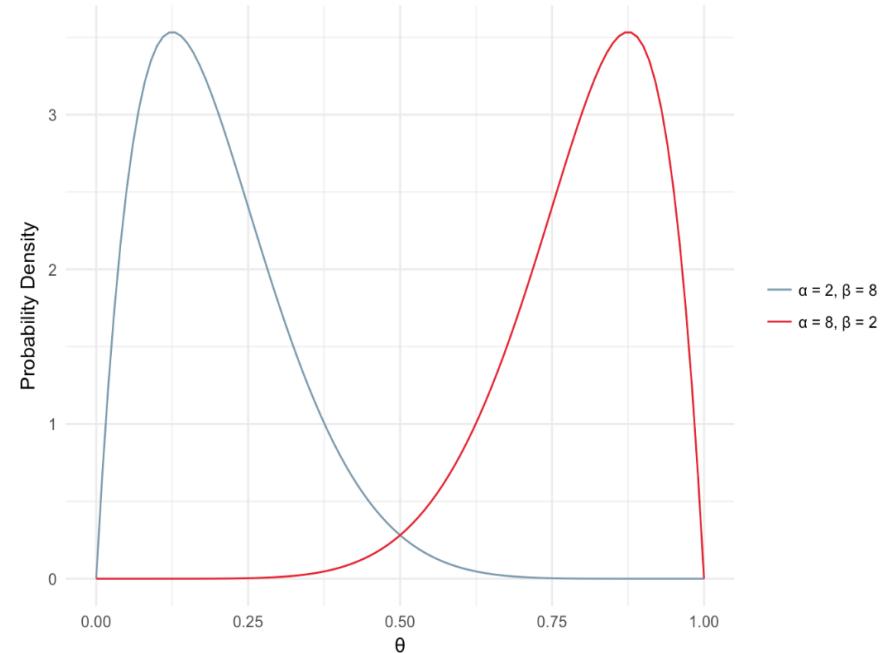
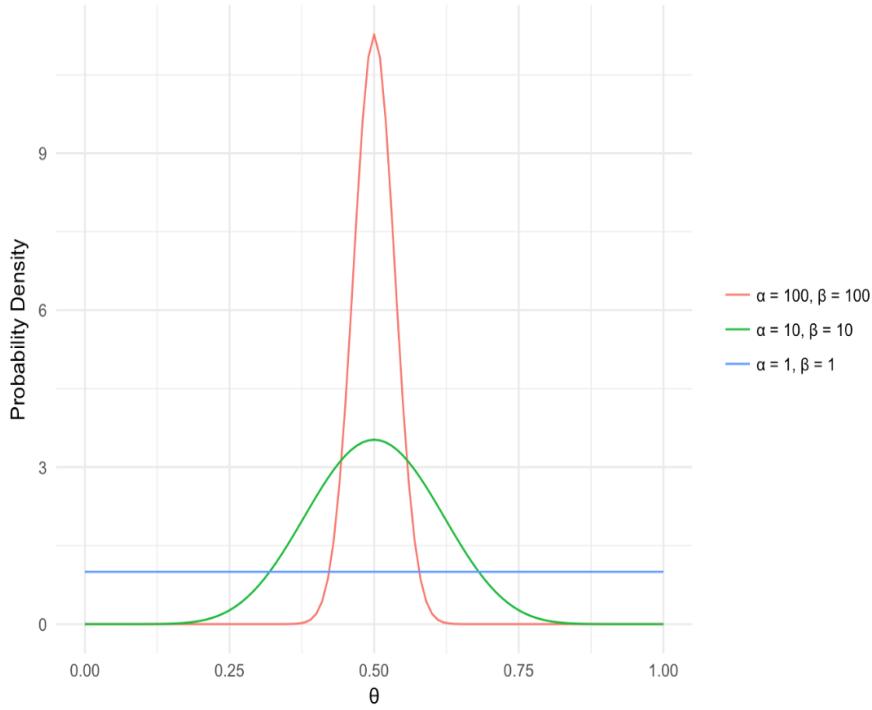
# Beta Distribution

- The probability distribution function for the beta distribution

$$f(\theta; \alpha, \beta) = \frac{\theta^{(\alpha-1)}(1-\theta)^{(\beta-1)}}{B(\alpha, \beta)}$$

# Beta Distribution

- The beta distribution can be thought of as a **probability distribution of probabilities**.



# The Building Blocks of inferring the parameters



- Parameter estimation

$$\underbrace{p(\theta|D)}_{posterior} = \frac{\overbrace{p(D|\theta) p(\theta)}^{likelihood\ prior}}{\underbrace{p(D)}_{evidence}}$$

# Conjugate Prior

In Bayesian probability theory,

- if the posterior distributions  $p(\theta | x)$  are in the same probability distribution family as the prior probability distribution  $p(\theta)$ , the prior and posterior are then called **conjugate distributions**, and the prior is called a **conjugate prior** for the likelihood function

$$\underbrace{p(\theta|D)}_{\text{posterior}} = \frac{\overbrace{p(D|\theta) p(\theta)}^{\text{likelihood prior}}}{\underbrace{p(D)}_{\text{evidence}}}$$

- conjugate prior gives a closed-form expression for the posterior; otherwise numerical integration may be necessary.
- Also may give intuition, by more transparently showing how a likelihood function updates a prior distribution

# Beta distribution as Conjugate Prior for Binomial distribution

Given a **prior**  $P(\theta | \alpha, \beta) = \text{Beta}(\alpha, \beta)$ , and **data**  $D=(H, T)$ , what is our posterior?

$$\begin{aligned}
 P(\theta | \alpha, \beta, H, T) &\propto P(H, T | \theta)P(\theta | \alpha, \beta) \\
 &\propto \theta^H(1 - \theta)^T \theta^{\alpha-1}(1 - \theta)^{\beta-1} \\
 &= \theta^{H+\alpha-1}(1 - \theta)^{T+\beta-1}
 \end{aligned}$$

With normalization

$$\begin{aligned}
 P(\theta | \alpha, \beta, H, T) &= \frac{\Gamma(H + \alpha + T + \beta)}{\Gamma(H + \alpha)\Gamma(T + \beta)} \theta^{H+\alpha-1}(1 - \theta)^{T+\beta-1} \\
 &= \text{Beta}(\alpha + H, \beta + T)
 \end{aligned}$$

# Dirichlet distributions

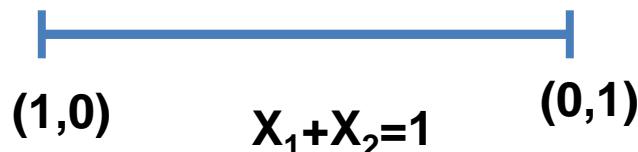
- Dirichlet distributions are probability distributions over multinomial parameter vectors
  - They are called Beta distributions when k = 2
- The Dirichlet probability density function is defined as

$$Dir(\vec{\theta} | \vec{\alpha}) = \frac{\Gamma(\sum_{i=1}^K \alpha_i)}{\prod_{i=1}^K \Gamma(\alpha_i)} \prod_{i=1}^K \theta_i^{\alpha_i - 1}$$

$$\text{➤ } Dir(\vec{\theta} | \vec{\alpha}) = \frac{1}{B(\alpha)} \prod_{i=1}^K \theta_i^{\alpha_i - 1} \quad \text{where} \quad \frac{1}{B(\alpha)} = \frac{\Gamma(\sum_{i=1}^K \alpha_i)}{\prod_{i=1}^K \Gamma(\alpha_i)}$$

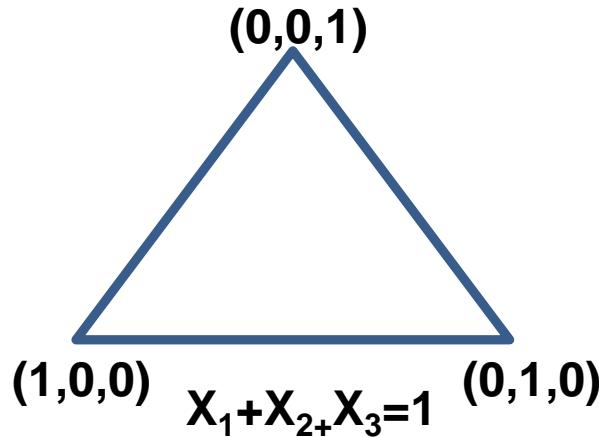
# Visualization of the simplex

- This is often referred as **simplex** and a most convenient way to visualize this is using a certain shapes depending upon the number of topics.
- Suppose K=2 topics which can be modeled as 1-simplex and can be visualized using a line.



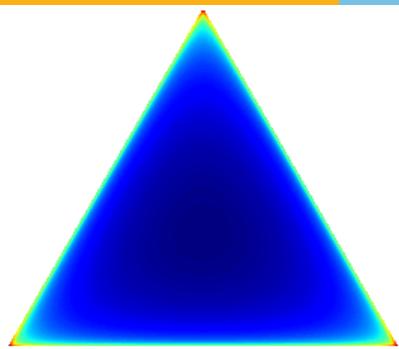
# Visualization of the simplex

- Suppose K=3 topics which can be modeled as 2-simplex and can be visualized using a triangle.

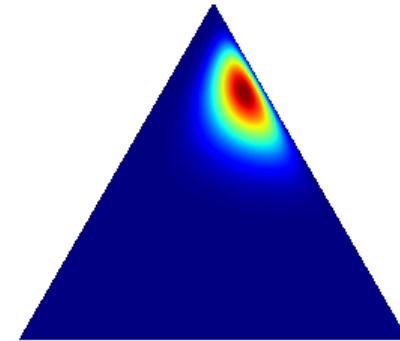


- If we have **K topics** this can be generated using **K-1 simplex**.

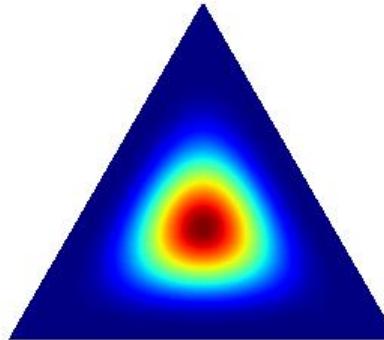
# Shape of the Dirichlet distribution



Dirichlet(0.999, 0.999, 0.999)



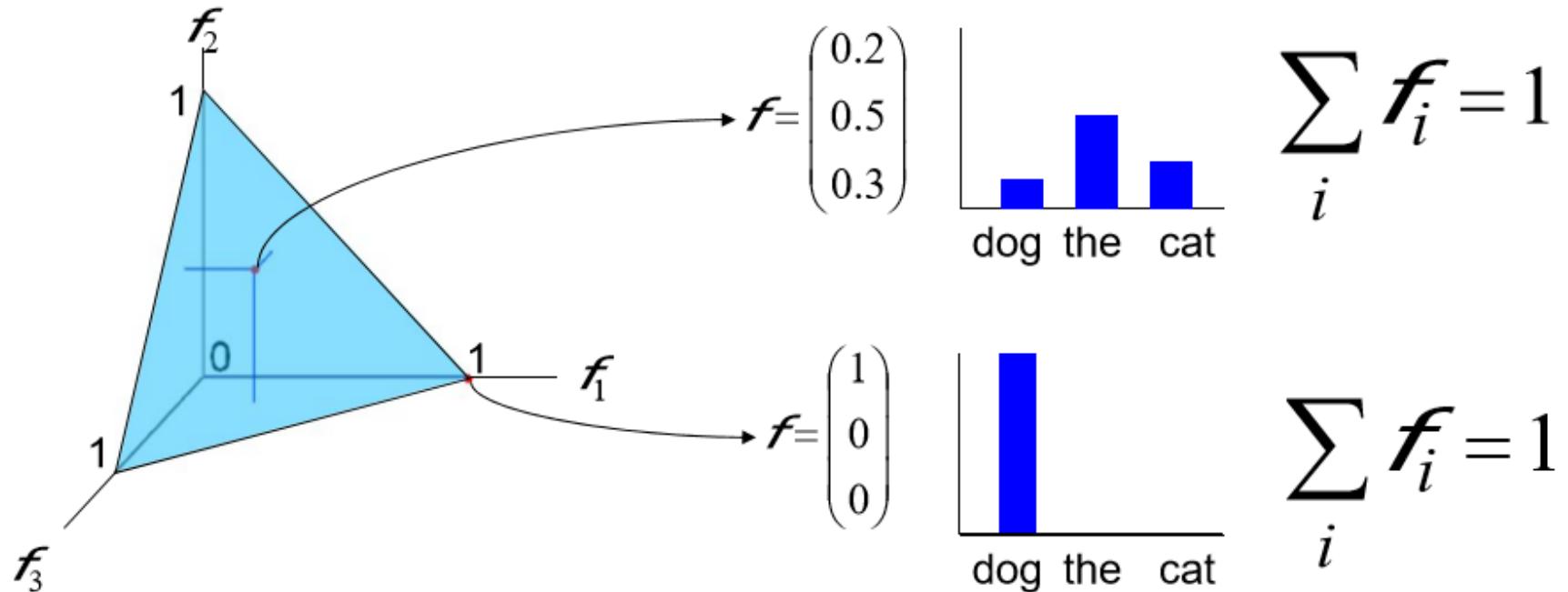
Dirichlet(2, 5, 15)



Dirichlet(5, 5, 5)

# Dirichlet distributions

Each point on a  $k$  dimensional simplex is a multinomial probability distribution:



$$\sum_i f_i = 1$$

$$\sum_i f_i = 1$$



# Statistical Inference

---

# Topic Modelling Example - LDA



Sports



Politics



Science



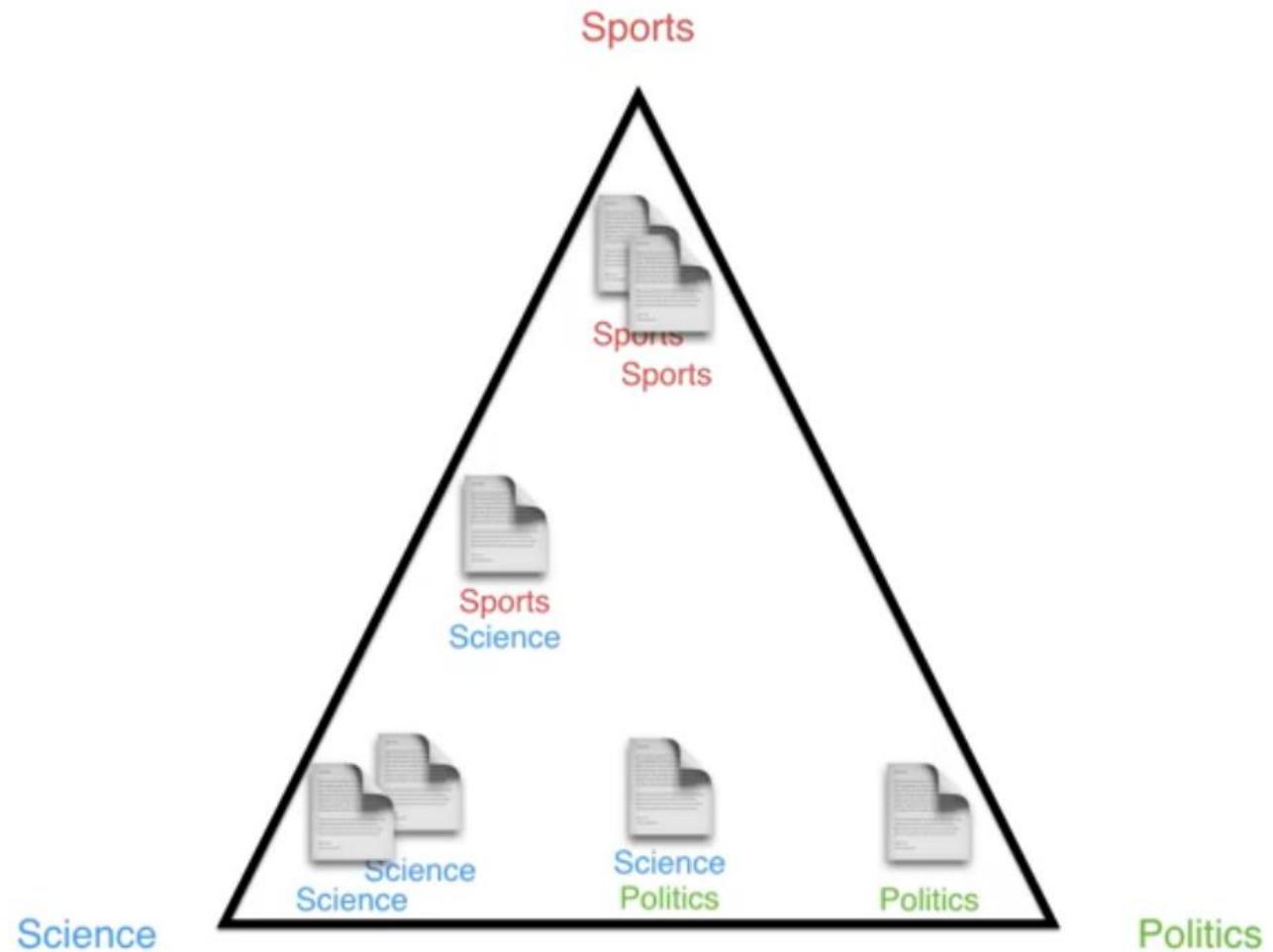
Science

Science

Politics

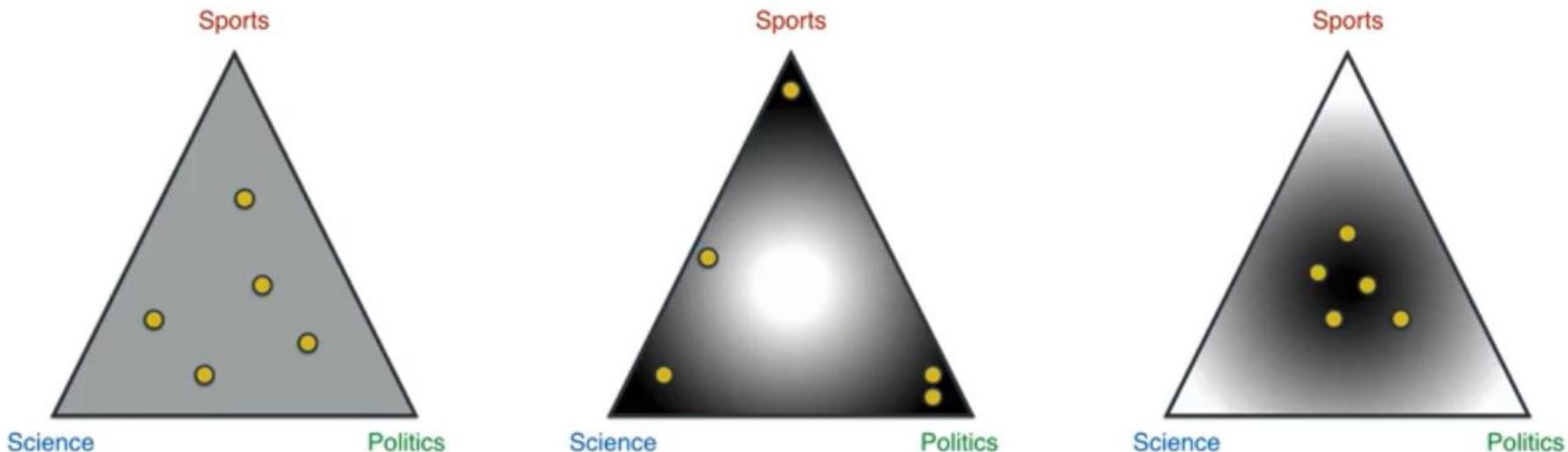
Sports

# LDA



---

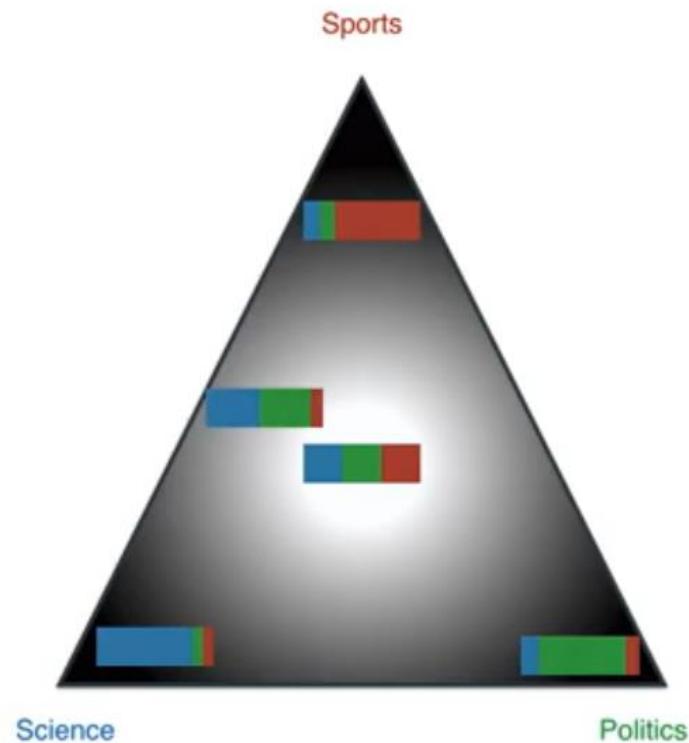
# Quiz: Which one for topics?



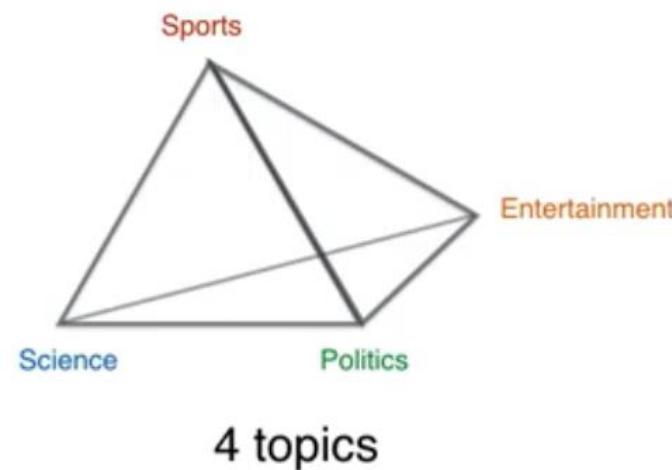
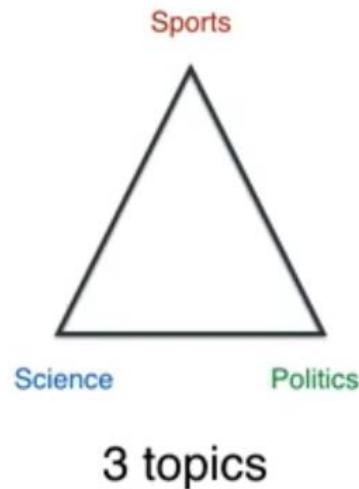
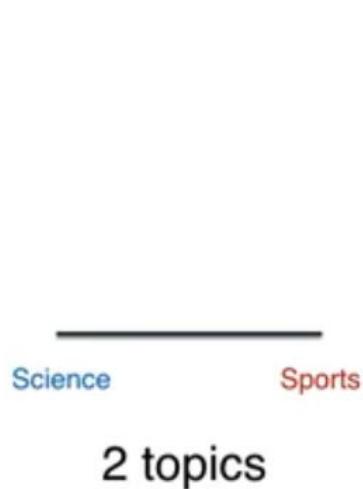
# Quiz: Which one for topics?



# A distribution of distributions



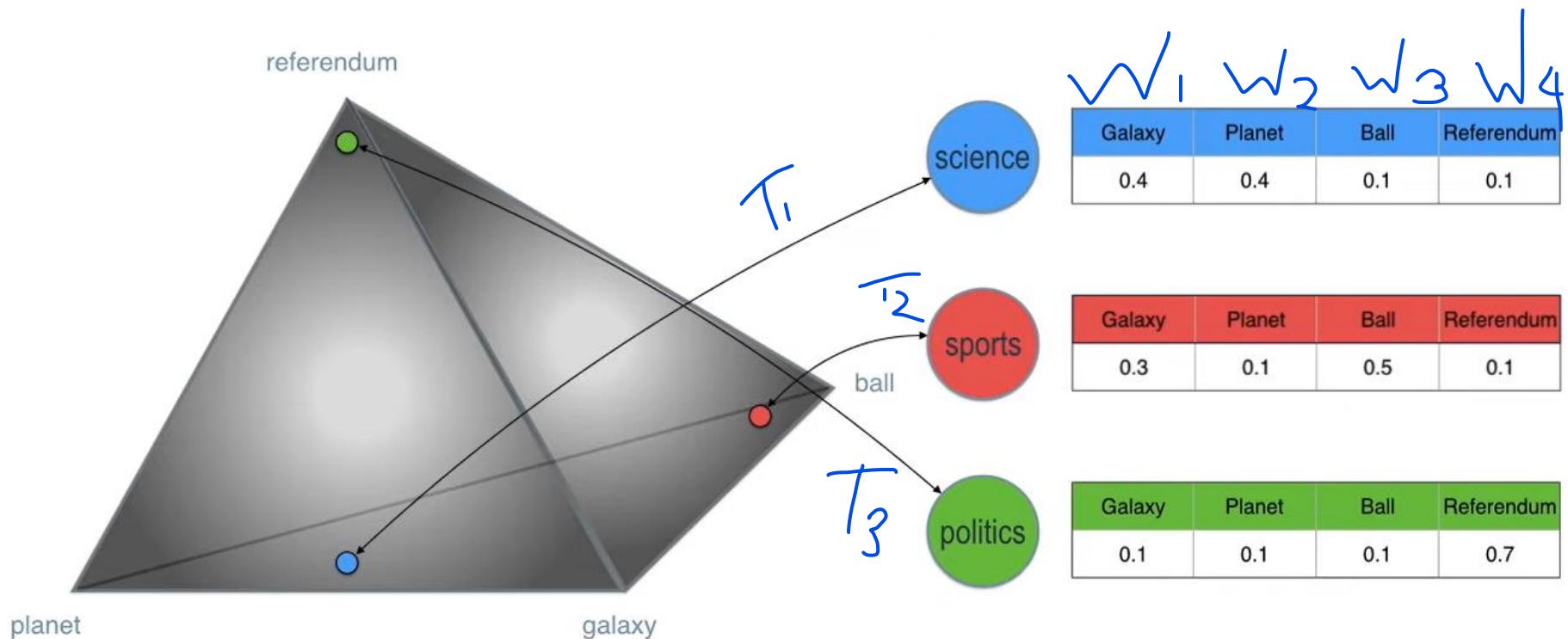
# More topics? More dimensions



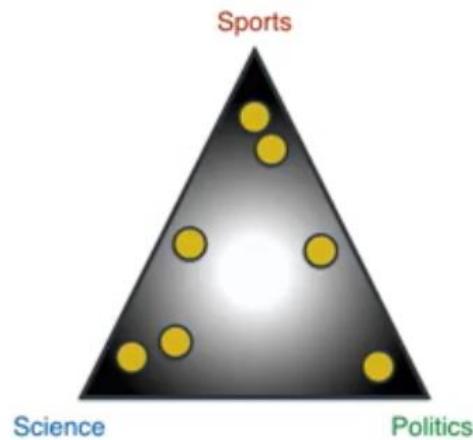
...

n-dimensional simplex

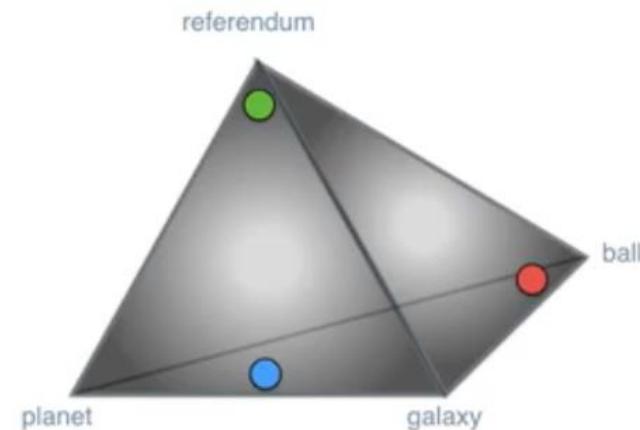
# Word distributions



# Two Dirichlet distributions

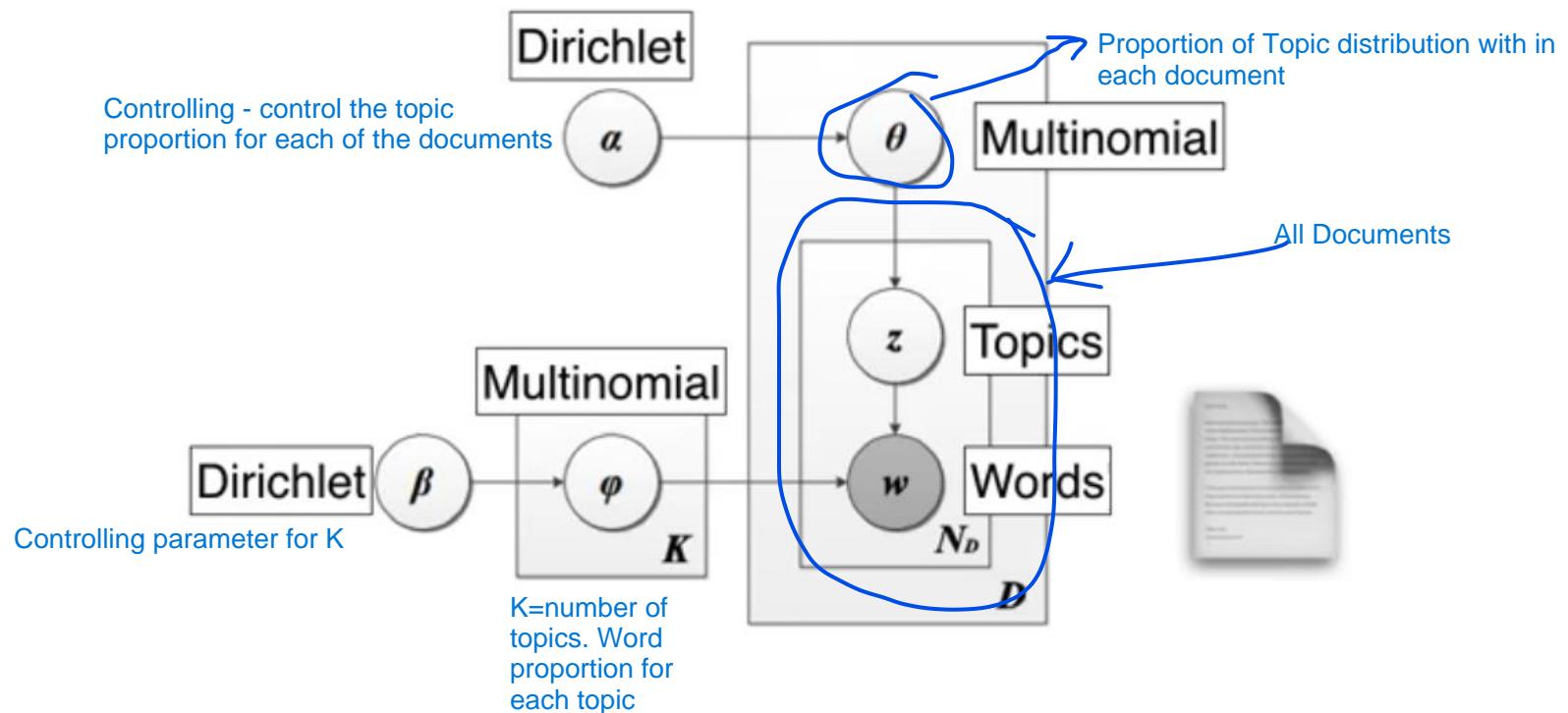


Documents-Topics



Topics-Words

# Blueprint for the LDA machine

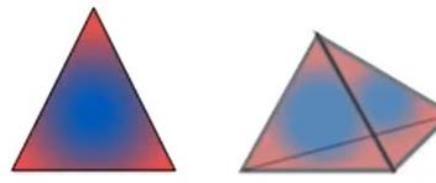


# Mathematical modelling of LDA



## Probability of a document

$$P(\mathbf{W}, \mathbf{Z}, \boldsymbol{\theta}, \boldsymbol{\varphi}; \alpha, \beta) = \prod_{j=1}^M P(\theta_j; \alpha) \prod_{i=1}^K P(\varphi_i; \beta) \prod_{t=1}^N P(Z_{j,t} | \theta_j) P(W_{j,t} | \varphi_{Z_{j,t}})$$



Topics

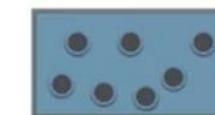
Words



Dirichlet  
Distributions



Topics



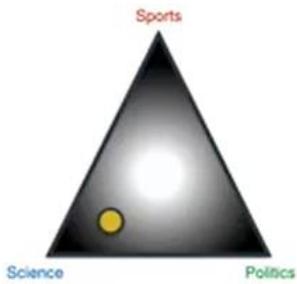
Words



Multinomial  
Distributions

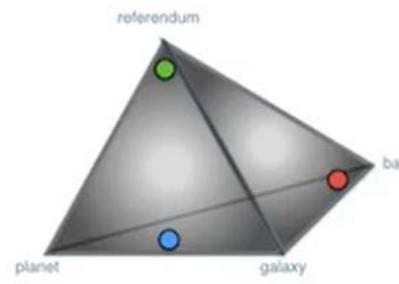
# Mathematical modelling of LDA

$$\prod_{j=1}^M P(\theta_j; \alpha)$$



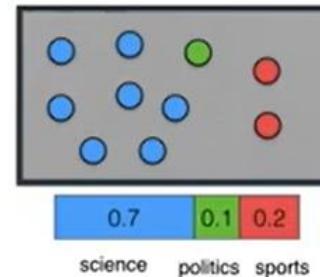
| science | politics | sports |
|---------|----------|--------|
| 0.7     | 0.1      | 0.2    |

$$\prod_{i=1}^K P(\varphi_i; \beta)$$

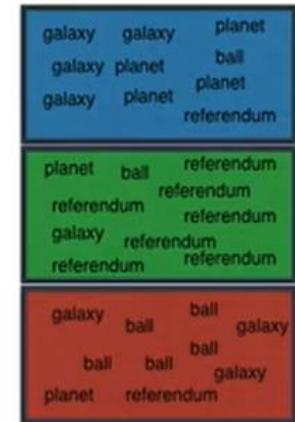


| Galaxy | Planet | Ball | Referendum |
|--------|--------|------|------------|
| 0.4    | 0.4    | 0.1  | 0.1        |
| Galaxy | Planet | Ball | Referendum |
| 0.1    | 0.1    | 0.1  | 0.7        |
| Galaxy | Planet | Ball | Referendum |
| 0.3    | 0.1    | 0.5  | 0.1        |

$$\prod_{t=1}^N P(Z_{j,t} | \theta_j)$$



$$P(W_{j,t} | \varphi_{Z_{j,t}})$$



|        |        |            |            |        |            |            |
|--------|--------|------------|------------|--------|------------|------------|
| galaxy | galaxy | planet     | ball       | planet | referendum |            |
| planet | ball   | referendum | referendum | galaxy | referendum | referendum |
| galaxy | ball   | ball       | ball       | galaxy | planet     | referendum |

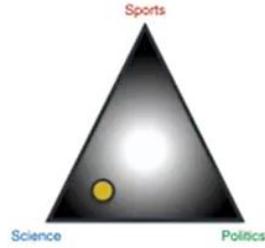
## Topics

- science → planet
- science → galaxy
- sports → ball
- science → planet
- science → galaxy
- politics → referendum
- sports → galaxy
- sports → ball
- science → referendum

## Words

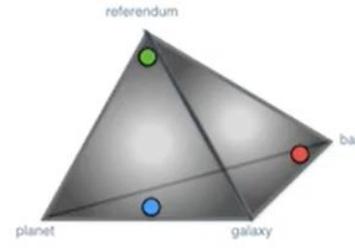
# Mathematical modelling of LDA

$$\prod_{j=1}^M P(\theta_j; \alpha)$$



| science | politics | sports |
|---------|----------|--------|
| 0.7     | 0.1      | 0.2    |

$$\prod_{i=1}^K P(\varphi_i; \beta)$$

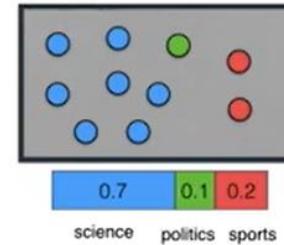


| Galaxy | Planet | Ball | Referendum |
|--------|--------|------|------------|
| 0.4    | 0.4    | 0.1  | 0.1        |
| Galaxy | Planet | Ball | Referendum |
| 0.1    | 0.1    | 0.1  | 0.7        |

| Galaxy | Planet | Ball | Referendum |
|--------|--------|------|------------|
| 0.3    | 0.1    | 0.5  | 0.1        |

$$\prod_{t=1}^N P(Z_{j,t} | \theta_j)$$



$$P(W_{j,t} | \varphi_{Z_{j,t}})$$

|            |        |            |
|------------|--------|------------|
| galaxy     | galaxy | planet     |
| galaxy     | planet | ball       |
| galaxy     | planet | planet     |
| referendum |        | referendum |

|            |            |            |
|------------|------------|------------|
| planet     | ball       | referendum |
| referendum | referendum | referendum |
| galaxy     | referendum | referendum |
| referendum |            | referendum |

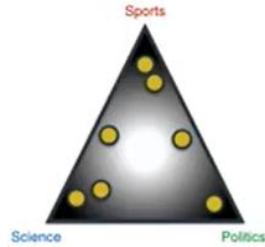
|        |            |            |
|--------|------------|------------|
| galaxy | ball       | ball       |
| ball   | ball       | galaxy     |
| planet | referendum | galaxy     |
| ball   | ball       | galaxy     |
| galaxy | galaxy     | referendum |



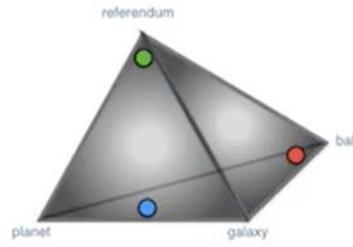
|            |
|------------|
| planet     |
| galaxy     |
| ball       |
| planet     |
| galaxy     |
| referendum |
| galaxy     |
| ball       |
| referendum |

# Mathematical modelling of LDA

$$\prod_{j=1}^M P(\theta_j; \alpha)$$



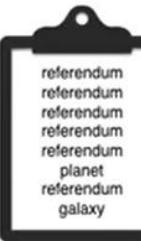
$$\prod_{i=1}^K P(\varphi_i; \beta)$$



$$\prod_{t=1}^N P(Z_{j,t} | \theta_j)$$

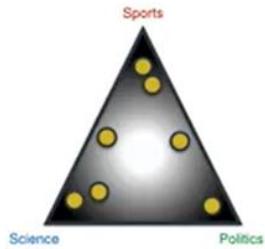
$$P(W_{j,t} | \varphi_{Z_{j,t}})$$

$P(\text{same articles}) = \text{low}$

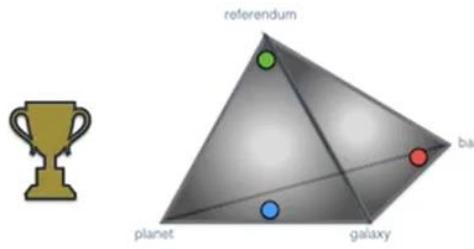


# Mathematical modelling of LDA

$$\prod_{j=1}^M P(\theta_j; \alpha)$$

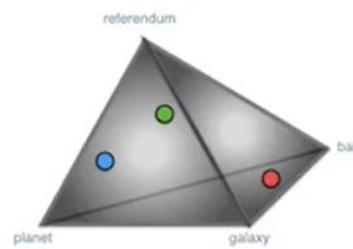
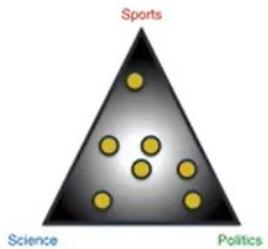


$$\prod_{i=1}^K P(\varphi_i; \beta)$$



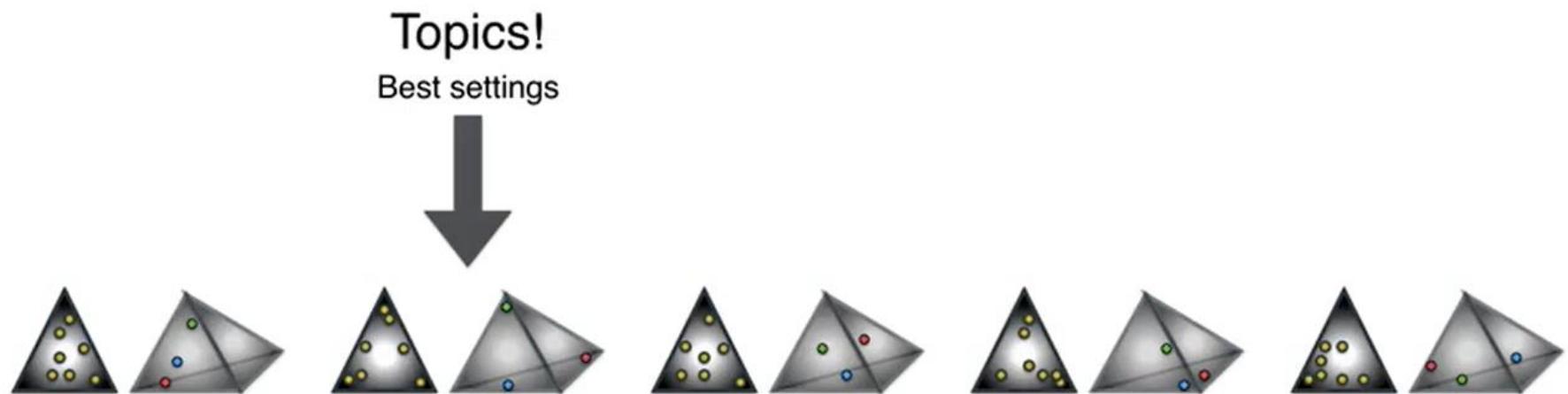
$$\prod_{t=1}^N P(Z_{j,t} | \theta_j) \quad P(W_{j,t} | \varphi_{Z_{j,t}})$$

$P(\text{same articles}) = \text{low}$



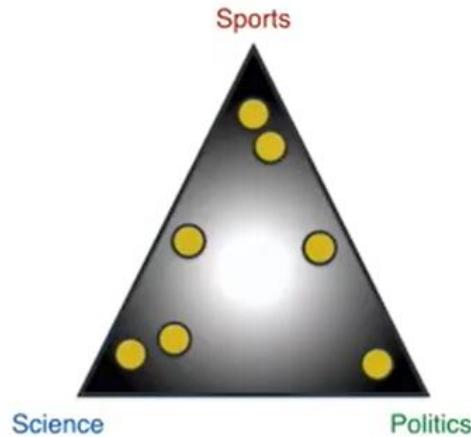
$P(\text{same articles}) = \text{very low}$

# Best settings on the machine

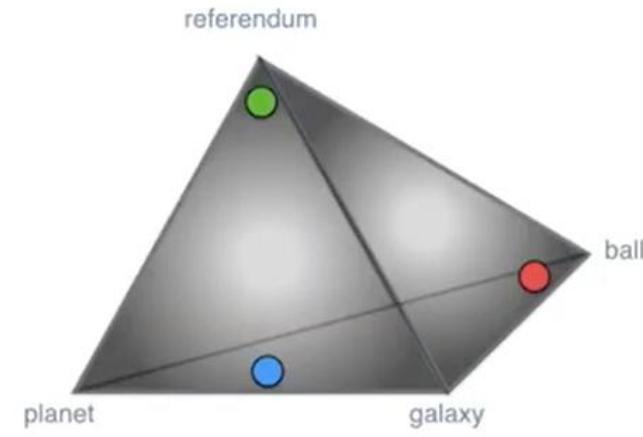


# Mathematical modelling of LDA

## The winning arrangements

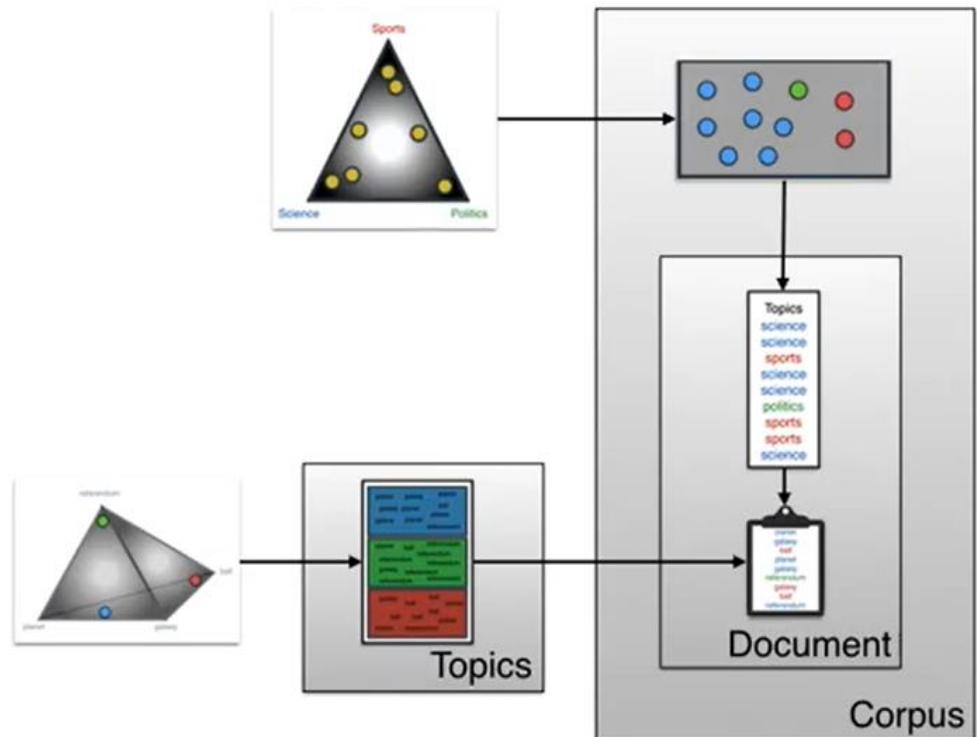
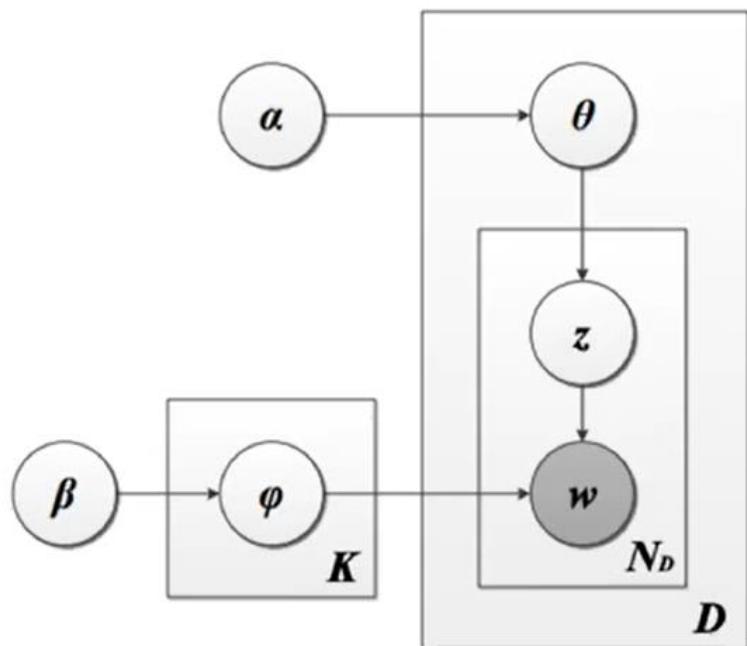


Documents-Topics



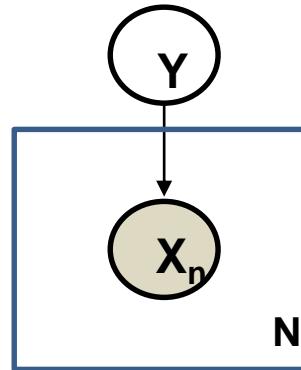
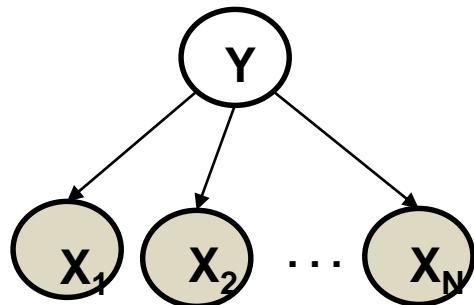
Topics-Words

# Latent Dirichlet Allocation



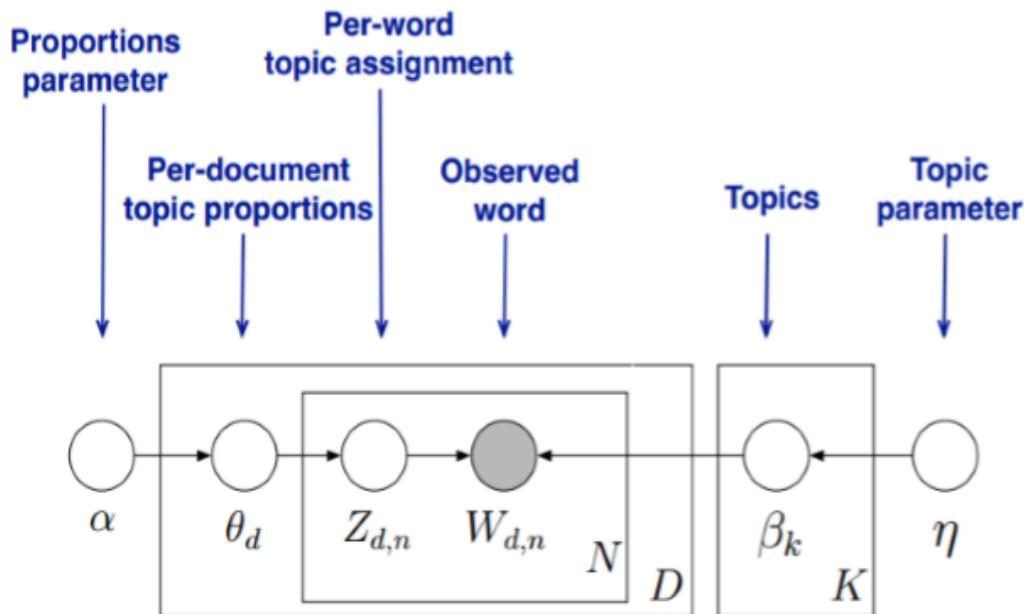
Length of the articles?  
Poisson distribution

# Directed graphical model



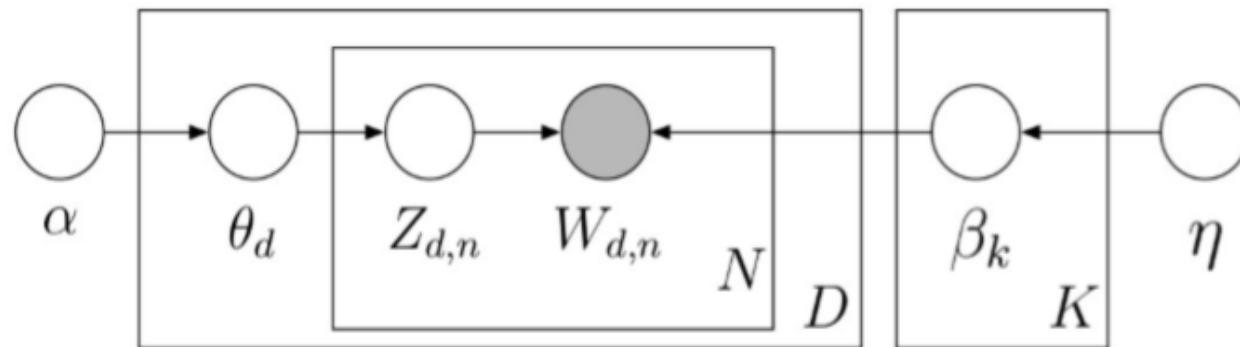
# Graphical LDA Model

Our goal is to **infer** or **estimate** the hidden variables, i.e. computing their distribution conditioned on the documents.  $\longrightarrow p(\text{topics, proportions, assignments} \mid \text{documents})$



- Nodes are RVs; edges indicate dependence.
- Shaded nodes are observed, and unshaded nodes are hidden.
- Plates indicate replicated variables.

# Graphical LDA Model



$K$  – total number of topics

$\beta_k$  – topic, a distribution over the vocabulary

$D$  – total number of documents

$\Theta_d$  – per-document topic proportions

$N$  – total number of words in a document (it fact, it should be  $N_d$ )

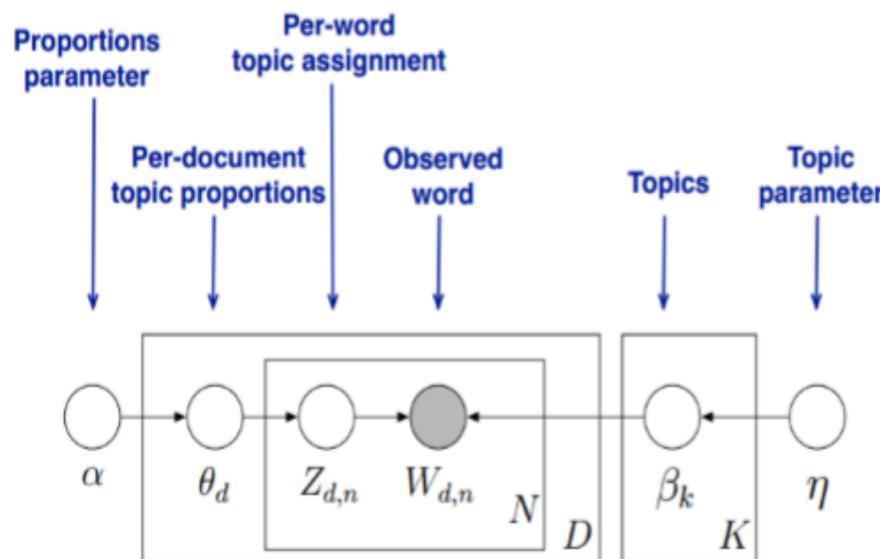
$Z_{d,n}$  – per-word topic assignment

$W_{d,n}$  – observed word

$\alpha, \eta$  – Dirichlet parameters

- Several **inference algorithms** are available (e.g. sampling based)
- A few **extensions** to LDA were created:
  - Bigram Topic Model

# Graphical LDA Model



$$p(\beta, \theta, z, w) = \left( \prod_{i=1}^K p(\beta_i | \eta) \right) \left( \prod_{d=1}^D p(\theta_d | \alpha) \prod_{n=1}^N p(z_{d,n} | \theta_d) p(w_{d,n} | \beta_{1:K}, z_{d,n}) \right)$$

1) Draw each topic  $\beta_i \sim Dir(\eta)$  for  $i = 1, \dots, K$

2) For each document:

First, Draw topic proportions  $\theta_d \sim Dir(\alpha)$

For each word within the document:

- a) Draw  $Z_{d,n} \sim Multi(\theta_d)$
- b) Draw  $W_{d,n} \sim Multi(\beta_{z_{d,n}})$

# LDA Model

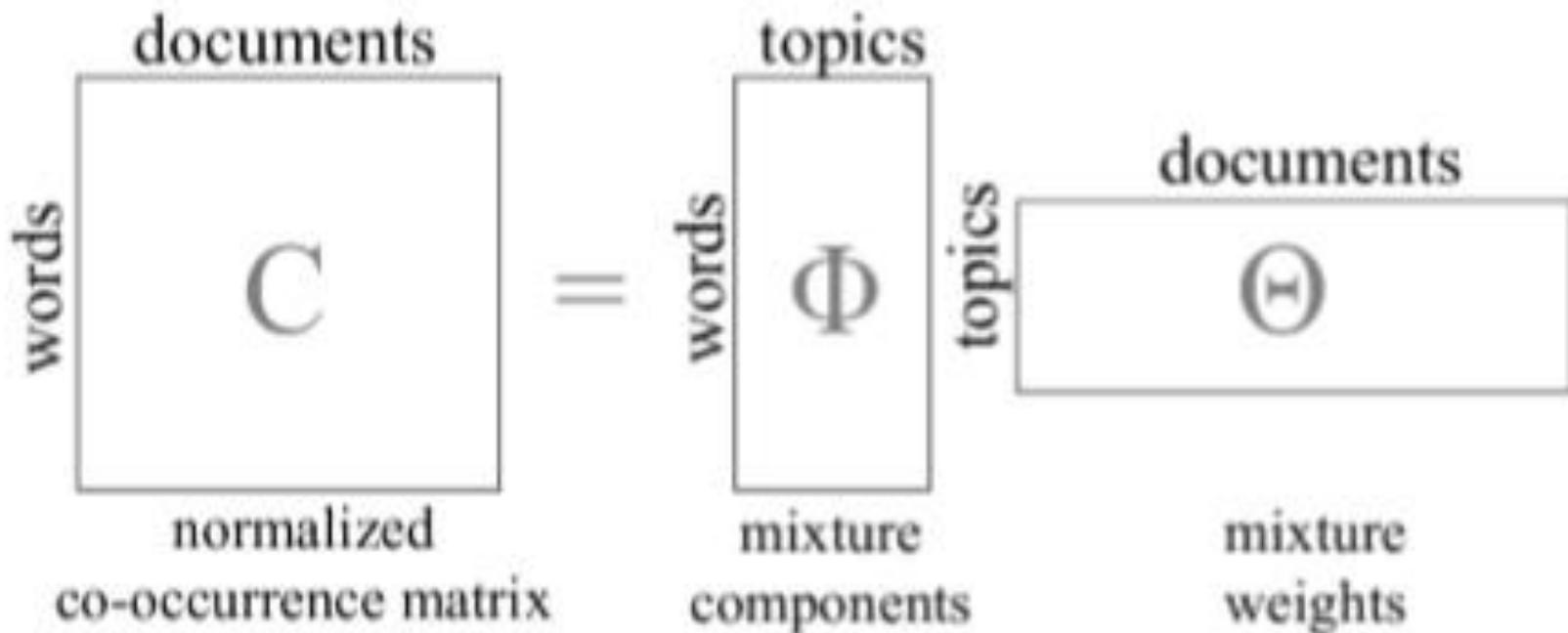
This joint distribution defines a posterior  $p(\theta, z, \beta | w)$ .

From a collection of documents we have to infer:

1. Per-word topic assignment  $z_{d,n}$ .
2. Per-document topic proportions  $\theta_d$ .
3. Per-corpus topic distributions  $\beta_k$ .

Then use posterior expectations ( $E\{\beta|w\}$  for the corpus,  $E\{\theta_d|w\}$  for each document) to perform the task at hand: information retrieval, document similarity, exploration, and others.

# LDA Matrix representation



# Mathematical modelling of LDA

Formal definition of the model:

$$p(\beta, \theta, z, w) = \left( \prod_{i=1}^K p(\beta_i | \eta) \right) \left( \prod_{d=1}^D p(\theta_d | \alpha) \prod_{n=1}^N p(z_{d,n} | \theta_d) p(w_{d,n} | \beta_{1:K}, z_{d,n}) \right)$$

$$(\beta_d | \eta) \sim Dir(\beta)$$

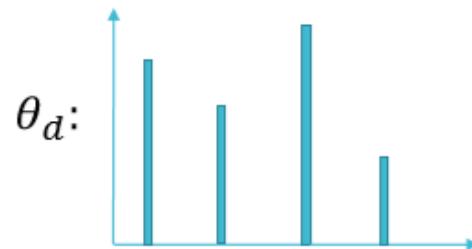
$$(\theta_d | \alpha) \sim Dir(\alpha)$$

$$Z_{d,n} \sim Multi(\theta_d)$$

$$W_{d,n} \sim Multi(\beta_{z_{d,n}})$$

$$p(z_{d,n} | \theta_d) = \theta_{d,z_{d,n}}$$

$$p(w_{d,n} | z_{d,n}, \beta_{1:K}) = \beta_{z_{d,n}, w_{d,n}}$$



$\beta$ :

| Topics  | Word probabilities for each topic |         |         |
|---------|-----------------------------------|---------|---------|
|         | Topic 1                           | Topic 2 | Topic 3 |
| Topic 1 |                                   |         |         |
| Topic 2 |                                   |         |         |
| Topic 3 |                                   |         |         |
| Topic 4 |                                   |         |         |

# Train LDA - Gibbs Sampling



Sports



Politics



Science



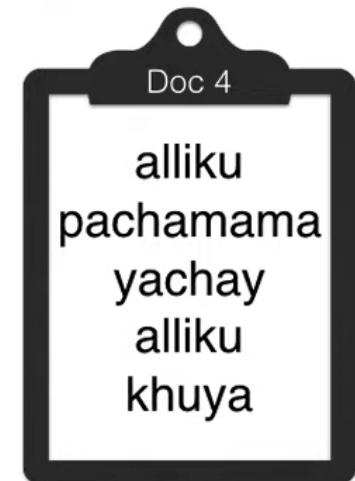
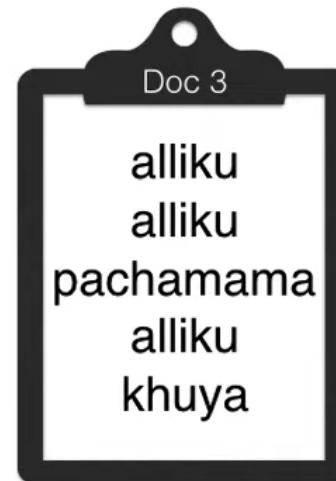
Science

Science

Politics

Sports

# Assign Topics





Topic 1

Topic 2

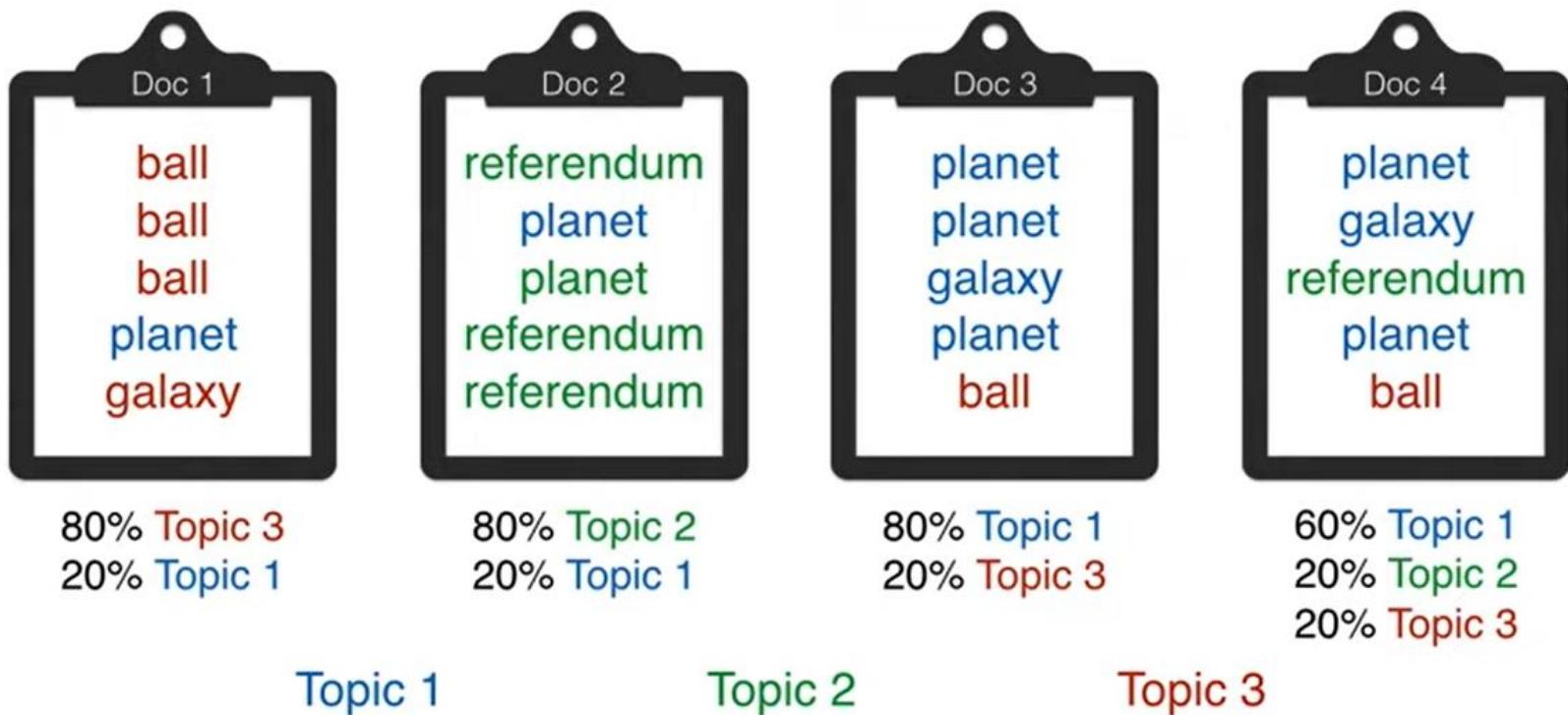
Topic 3



Topic 1

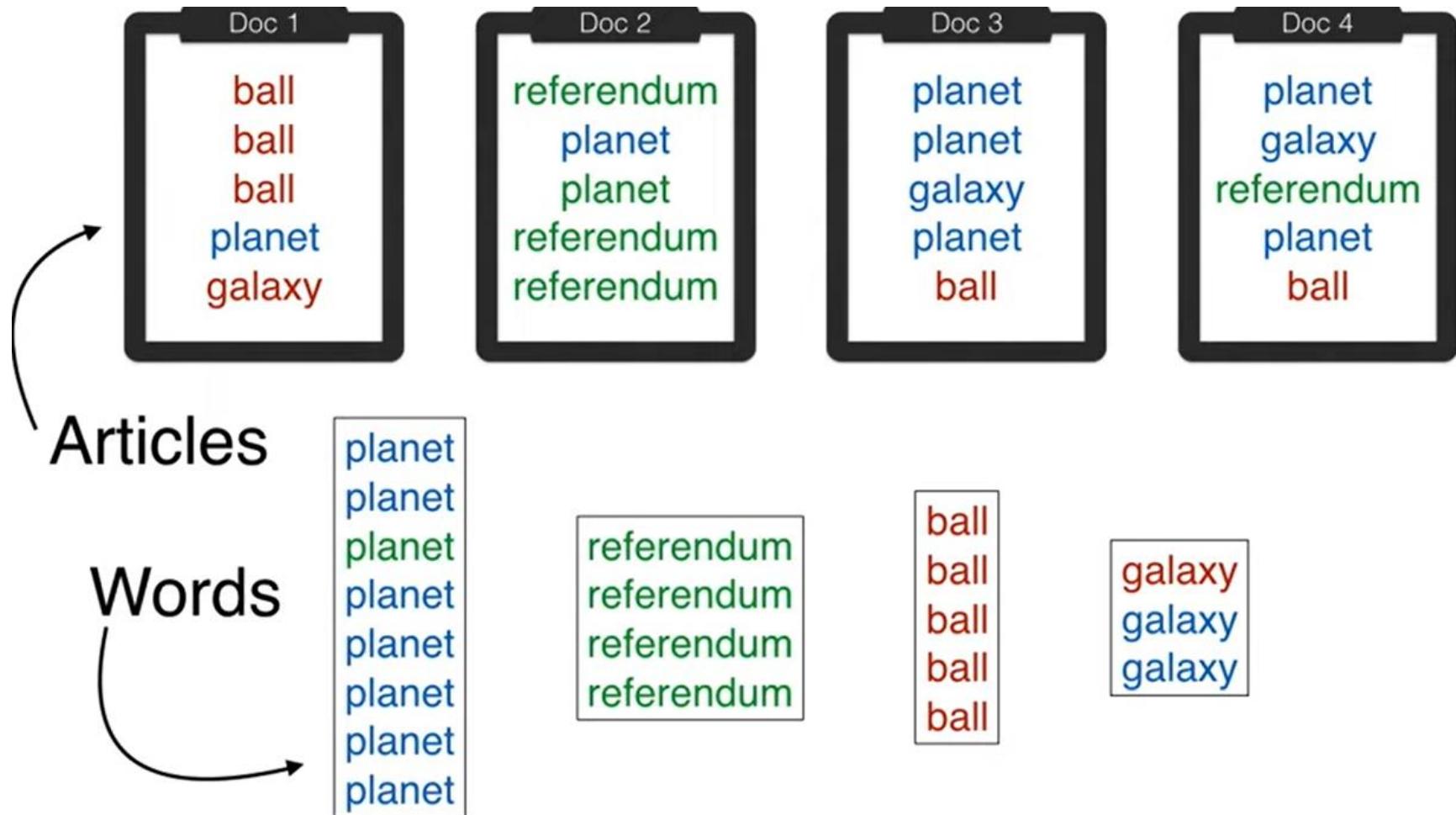
Topic 2

Topic 3





Property 1:  
Articles are as monochromatic as possible



## Property 2: Words are as monochromatic as possible

Words

planet  
planet  
planet  
planet  
planet  
planet  
planet  
planet

referendum  
referendum  
referendum  
referendum

ball  
ball  
ball  
ball  
ball

galaxy  
galaxy  
galaxy



Goal: Color each word with **blue**, **green**, **red**

- 1. Each article is as monochromatic as possible
- 2. Each word is as monochromatic as possible



**ball**

Topic 1

Topic 2

Topic 3

How much is Topic 1 in Doc 1?

2

How much is 'ball' in Topic 1?

0

Product: 0

How much is Topic 2 in Doc 1?

0

How much is 'ball' in Topic 2?

1

Product: 0

How much is Topic 3 in Doc 1?

2

How much is 'ball' in Topic 3?

3

Product: 6



### Topic 1

How much is Topic 1 in Doc 1?

$$2 + \alpha$$

How much is 'ball' in Topic 1?

$$0 + \beta$$

### Topic 2

How much is Topic 2 in Doc 1?

$$0 + \alpha$$

How much is 'ball' in Topic 2?

$$1 + \beta$$

### Topic 3

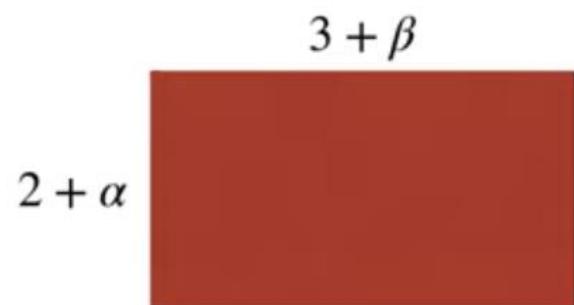
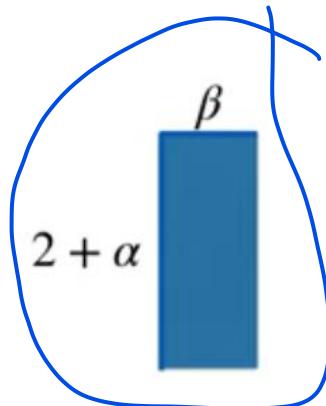
How much is Topic 3 in Doc 1?

$$2 + \alpha$$

How much is 'ball' in Topic 3?

$$3 + \beta$$

product of beta X (2+alpha)



### Topic 1

How much is Topic 1 in Doc 1?

$$2 + \alpha$$

How much is 'ball' in Topic 1?

$$0 + \beta$$

### Topic 2

How much is Topic 2 in Doc 1?

$$0 + \alpha$$

How much is 'ball' in Topic 2?

$$1 + \beta$$

### Topic 3

How much is Topic 3 in Doc 1?

$$2 + \alpha$$

How much is 'ball' in Topic 3?

$$3 + \beta$$



80% Topic 3  
20% Topic 1



80% Topic 2  
20% Topic 1



80% Topic 1  
20% Topic 3



60% Topic 1  
20% Topic 2  
20% Topic 3

### Science

Topic 1  
planet (7)  
galaxy (2)

### Politics

Topic 2  
referendum (4)  
planet (1)

### Sports

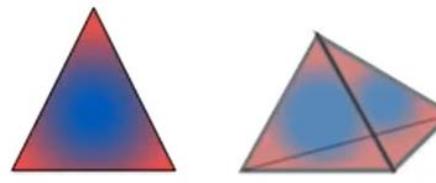
Topic 3  
ball (5)  
galaxy (1)

# Mathematical modelling of LDA



## Probability of a document

$$P(\mathbf{W}, \mathbf{Z}, \boldsymbol{\theta}, \boldsymbol{\varphi}; \alpha, \beta) = \prod_{j=1}^M P(\theta_j; \alpha) \prod_{i=1}^K P(\varphi_i; \beta) \prod_{t=1}^N P(Z_{j,t} | \theta_j) P(W_{j,t} | \varphi_{Z_{j,t}})$$

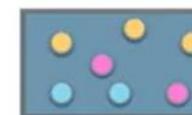


Topics

Words



Dirichlet  
Distributions



Topics



Words



Multinomial  
Distributions

## Probability of a document

## Gibbs sampling

# Gibbs Sampling Algorithm

- Step1: Assign a random topic [1...T] for each word
- Step2: For each word token, a new topic is sampled as per  $P(z_i=j|z_{-i}, w_i, d_i)$  and the matrices  $C_{wt}$  (word-topic) and  $C_{dt}$  (document-topic) are updated.
- One iteration over all word token in the document is a Gibbs Sample
- Each iteration may have correlation with the next hence these samples are saved at spaced intervals.

# LDA Summary

---

Documents are probability distributions over latent topics.

Topics are probability distributions over words.

LDA takes a number of documents. It assumes that the words in each document are related. It then tries to figure out the “recipe” for how each document could have been created. We just need to tell the model how many topics to construct and it uses that “recipe” to generate topic and word distributions over a corpus. Based on that output, we can identify similar documents within the corpus.

# LDA Summary

---

## ADVANTAGES

LDA is an effective tool for topic modeling.

Easy to understand conceptually

Has been shown to produce good results over many domains.

New applications

## LIMITATIONS

Must know the number of topics K in advance

Dirichlet topic distribution cannot capture correlations among topics

# Difference between document clustering and topic modeling



<https://iksinc.online/2016/05/16/topic-modeling-and-document-clustering-whats-the-difference/>

# Implementation Tools

## Tooling



**gensim:** topic modeling for humans

- Free python library
- Memory independent
- Distributed computing

<http://radimrehurek.com/gensim>



**MA**chine Learning for **LanguagE** Toolkit (MALLET) is a Java-based package for:

- statistical natural language processing
- document classification
- Clustering
- topic modeling
- information extraction
- and other machine learning applications to text.

<http://mallet.cs.umass.edu>



Stanford Topic Modeling Toolbox

<http://nlp.stanford.edu/software/tmt>

# References

---

Bernoulli trial, binomial and multinomial distribution:

<https://www.askiitians.com/iit-jee-algebra/probability/bernoulli-trials-and-binomial-distribution/>

Beta Distribution:

- [https://www.youtube.com/watch?v=v1uUgTcInQk&feature=emb\\_logo](https://www.youtube.com/watch?v=v1uUgTcInQk&feature=emb_logo)

Conjugate Prior:

- [https://www.youtube.com/watch?time\\_continue=2&v=aPNrhR0dFi8&feature=emb\\_logo](https://www.youtube.com/watch?time_continue=2&v=aPNrhR0dFi8&feature=emb_logo)
- <https://www.youtube.com/watch?v=qpNAxNmy0GU>

Topic Models

- <https://www.youtube.com/watch?v=fCmlceNqVog>

Gibbs Sampling

- <https://www.youtube.com/watch?v=u7l5hhmdc0M>
- <https://medium.com/analytics-vidhya/topic-modeling-using-lda-and-gibbs-sampling-explained-49d49b3d1045>

# References

## LDA

- [1] Blei, David M., Andrew Y. Ng, and Michael I. Jordan. "Latent dirichlet allocation." Journal of machine Learning research (2003): 993-1022.
- <https://www.youtube.com/watch?v=3mHy4OSyRf0>
- <https://www.coursera.org/learn/text-mining/lecture/dmpQ0/2-5-topic-mining-and-analysis-motivation-and-task-definition>
- <https://www.youtube.com/watch?v=NYkbqzTIW3w>
- <https://github.com/adashofdata/nlp-in-python-tutorial>
- [https://www.youtube.com/watch?time\\_continue=3&v=Cpt97BpIt4&feature=emb\\_logo](https://www.youtube.com/watch?time_continue=3&v=Cpt97BpIt4&feature=emb_logo)
- <https://github.com/bhattbhavesh91>
- <https://www.youtube.com/watch?v=T05t-SqKArY>
- [https://www.youtube.com/watch?v=BaM1uiCpj\\_E](https://www.youtube.com/watch?v=BaM1uiCpj_E)
- <https://livebook.manning.com/book/grokking-machine-learning/>



# Natural Language Processing

## DSECL ZG565



**BITS** Pilani  
Pilani Campus

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



## **Session 9-Grammars and Parsing**

### **Date – 12<sup>th</sup> August 2023**

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

# Session Content

(Ref: Allen James - Chapter 3)

---

- 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 Processing.

# Ambiguity is Explosive

- “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

# Some funny examples

---

- Policeman to little boy: “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.

# Grammars and Sentence Structure

---

- **Grammar**- formal specification of the structures allowable in the language, and
- **Parsing technique**- method of analyzing a sentence to determine its structure according to the grammar

# What Makes a Good Grammar

---

- In small grammars, such as those that describe only a few types of sentences, one structural analysis of a sentence may appear as understandable as another, and little can be said as to why one is superior to the other.
- The analysis that retains its simplicity and generality as it is extended is more desirable
- Grammars consisting entirely of rules with a single symbol on the left-hand side, called the mother, are called context-free grammars (CFGs).

# Context Free Grammar

N a set of non-terminal symbols (or variables)

$\Sigma$  a set of terminal symbols (disjoint from N)

R a set of productions or rules of the form  $A \rightarrow \beta$ , where A is a non-terminal and  $\beta$  is a string of symbols from  $(\Sigma \cup N)^*$

S, a designated non-terminal called the start symbol

# 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.

# Simple grammar and Parsing Example

S -> NP VP

VP -> V NP

NP -> N

NP -> Adj NP

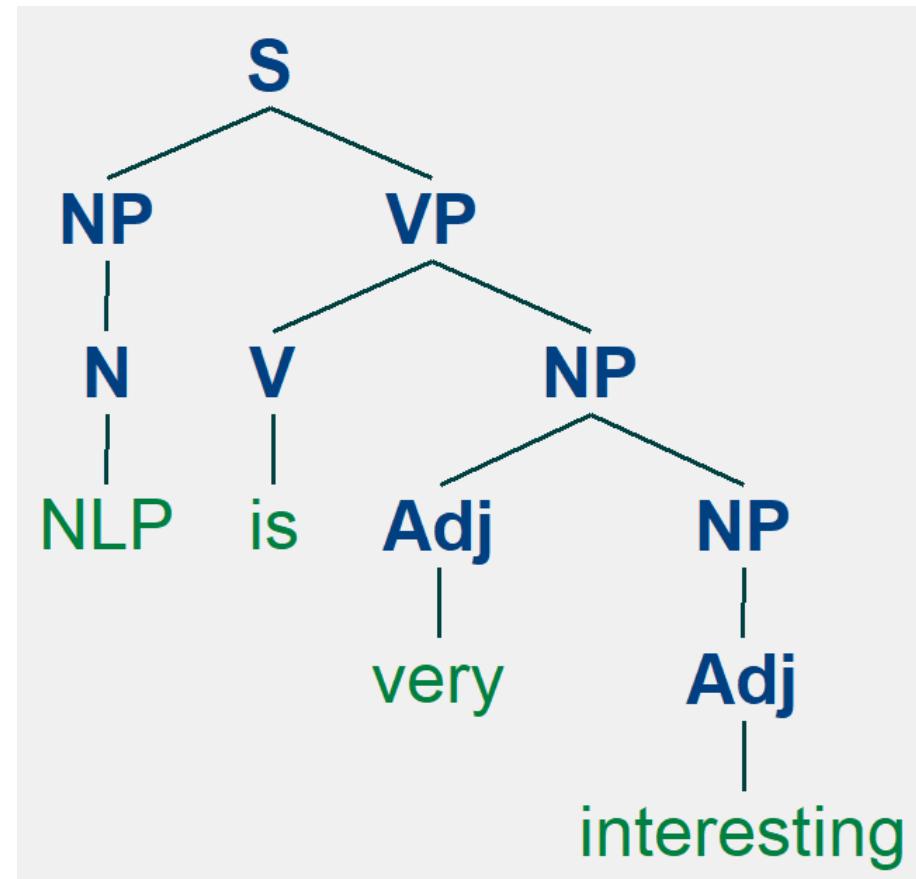
NP -> Adj

N -> NLP

V -> is

Adj -> very

Adj -> interesting

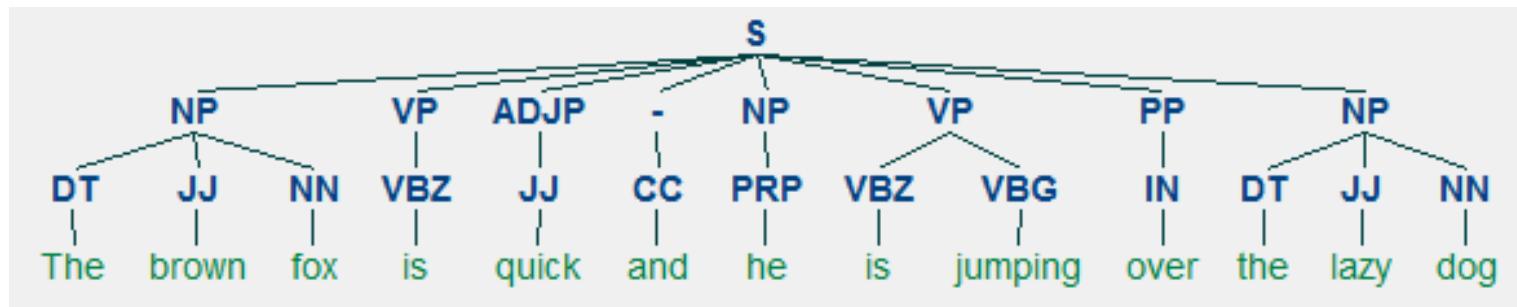


# Parsing

Analysis of words in the sentence for grammar and arranging words in a manner that shows the relationship among the words.

Analyzing the structure of a sentence to break it down into its smallest constituents (which are tokens such as words) and group them together into higher-level phrases.

This includes POS tags as well as phrases from a sentence.



# Applications of Parsing

---

**Sentiment Analysis:** Compare "I like Frozen", "I do not like Frozen", and "I like frozen yogurt". The three sentences' words are very similar to each other, yet the first and the second contain inverse statements about the movie "Frozen", while the third is a statement about something else. Parsing here is crucial for understanding.

**Relation Extraction:** "Rome is the capital of Italy and the region of Lazio". While entity extraction can give us the entities here, we need parsing to see which entity is the capital of which other entities.

**Question Answering:** When answering "Who was the first man in space?" you need to parse the question and use parsed sentences to build the answer.

---

# Applications of Parsing

---

**Speech Recognition:** Parsing scores the strings with either a pass/fail or a likelihood score, to give a powerful language model for speech recognition. Similarly, parsing could help in **spell checking, optical character recognition (OCR), text prediction, handwriting detection** etc.

**Machine Translation:** Parsing allows us to choose between several possible translations. Also, it makes translation of phrases and terms easier.

**Grammar Checking:** Parsing can help in checking the grammaticality of a document.

---

# Parsing

Given a string of non-terminals and a CFG, determine if the string can be generated by the CFG.

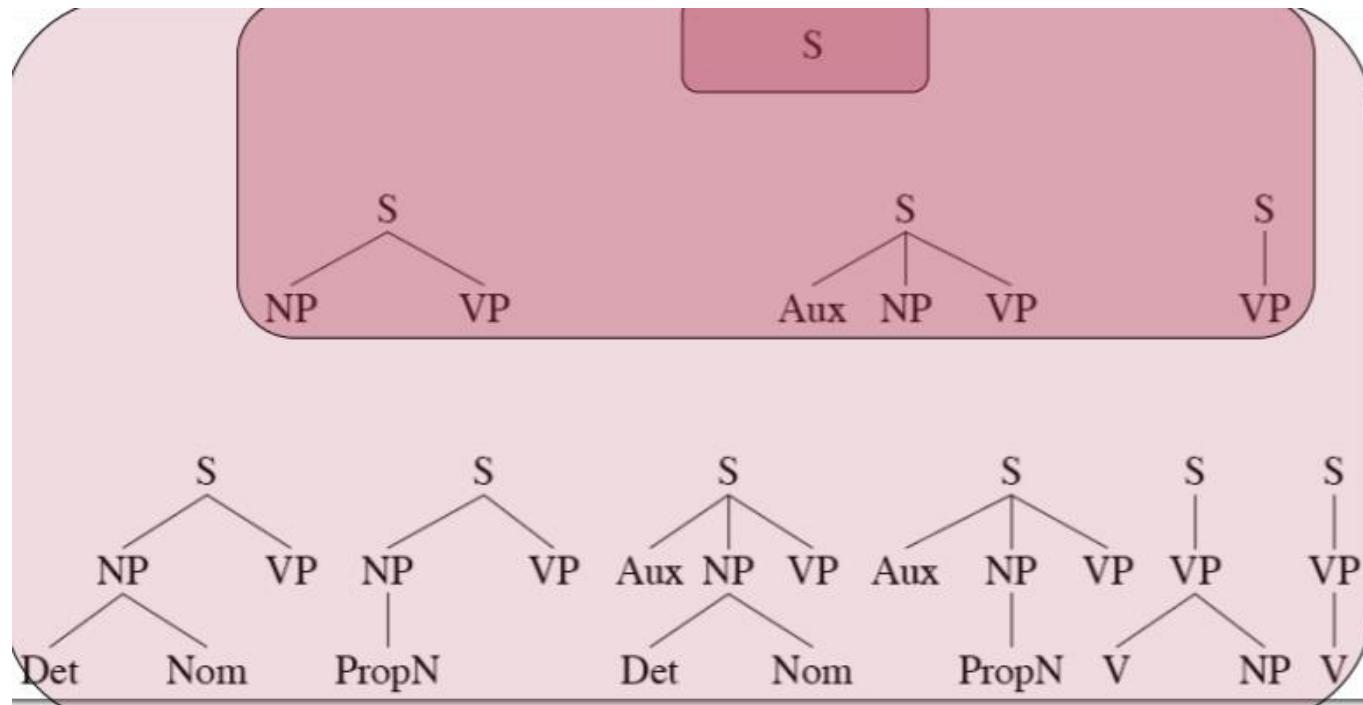
- Also return a parse tree for the string
- Also return all possible parse trees for the string

Must search space of derivations for one that derives the given string.

- Top-Down Parsing: Start searching space of derivations for the start symbol.
- Bottom-up Parsing: Start search space of reverse deviations from the terminal symbols in the string.

# Top down Parsing

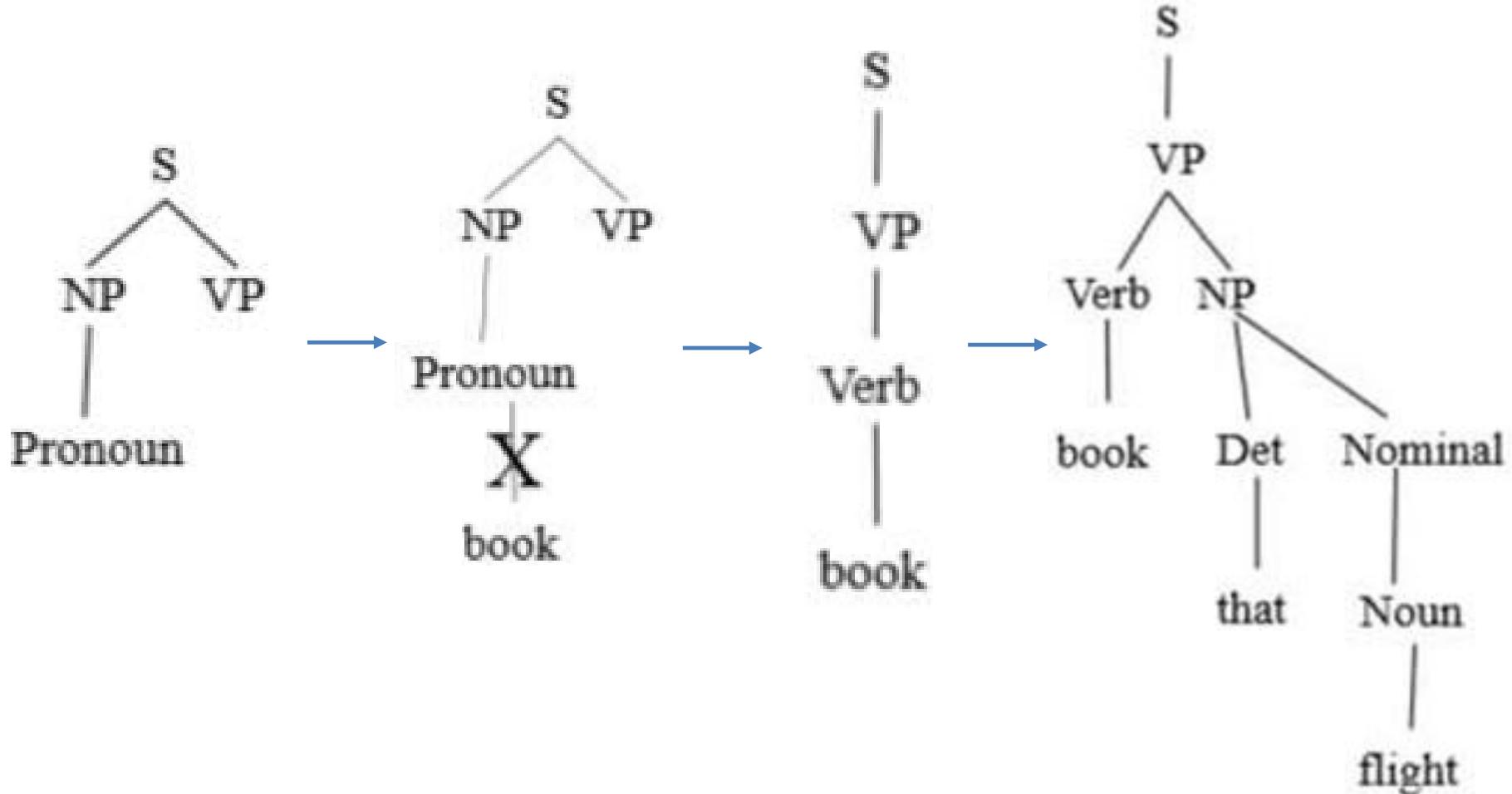
- Parse tree is generated in the top to bottom fashion from root to leaves
- Search space from the start symbol sentence S on LHS



# Top down parsing

1. **S -> NP VP**
2. **NP -> ART N**
3. **NP -> ART ADJ N**
  
4. **VP -> V**
  
5. **VP -> V NP**

# Top down parsing



# 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.

# Top down parse of : “<sub>1</sub>The <sub>2</sub> old <sub>3</sub> man <sub>4</sub> smiled. <sub>5</sub>”

| Step | Current State      | Backup States   | Comment   |                   |
|------|--------------------|---|---|-------------------|
| 1.   | ((S) 1)            |   |   | 1. S → NP VP      |
| 2.   | ((NP VP) 1)        |   |   | 2. NP → ART N     |
| 3.   | ((ART N VP) 1)     | ((ART ADJ N VP) 1)  | S rewritten to NP VP<br>NP rewritten producing two new states | 3. NP → ART ADJ N |
| 4.   | ((N VP) 2)         | ((ART ADJ N VP) 1)  |   |                   |
| 5.   | ((VP) 3)           | ((ART ADJ N VP) 1)  | the backup state remains                                      | 4. VP → V         |
| 6.   | ((V) 3)            | ((V NP) 3)<br>((ART ADJ N VP) 1)                            |   |                   |
| 7.   | (( ) 4)            | ((V NP) 3)<br>((ART ADJ N VP) 1)                            |   | 5. VP → V NP      |
| 8.   | ((V NP) 3)         | ((ART ADJ N VP) 1)  | the first backup is chosen                                    |                   |
| 9.   | ((NP) 4)           | ((ART ADJ N VP) 1)  |   |                   |
| 10.  | ((ART N) 4)        | ((ART ADJ N VP) 1)<br>((ART ADJ N) 4)<br>((ART ADJ N VP) 1) | looking for ART at 4 fails                                    |                   |
| 11.  | ((ART ADJ N) 4)    | ((ART ADJ N VP) 1)  | fails again   |                   |
| 12.  | ((ART ADJ N VP) 1) |   | now 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!  |                   |

## Lexicon

The → ART  
Old → N, ADJ  
Man → N, V  
Smiled -> V

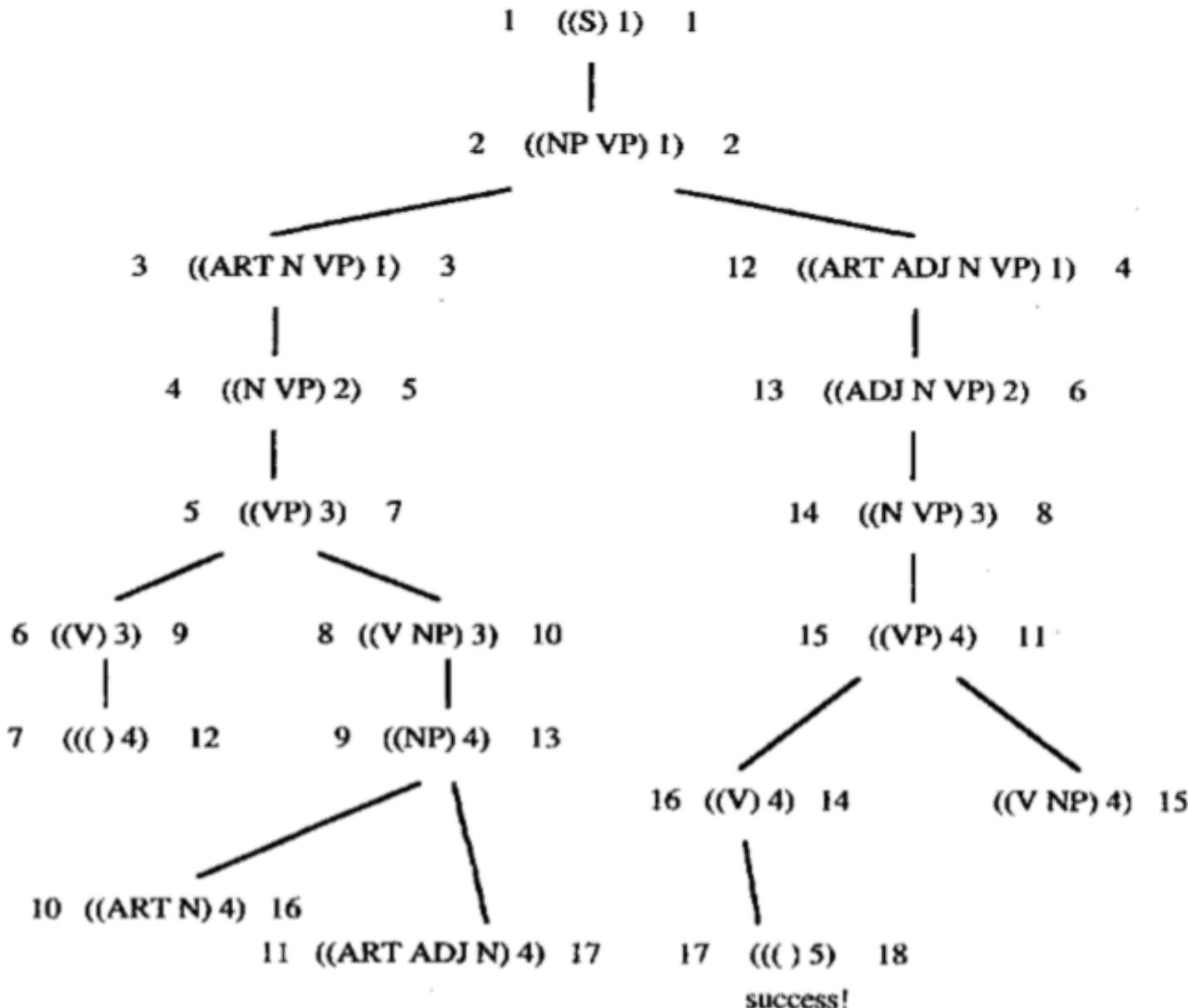


Figure 3.7 Search tree for two parse strategies (depth-first strategy on left; breadth-first on right)

# Depth-first strategy vs Breadth-first strategy

---

- For a depth-first strategy, the possibilities list is a stack, yielding a last-in first-out (LIFO) strategy.
- In contrast, in a breadth-first strategy the possibilities list is manipulated as a queue, yielding a first-in first-out (FIFO) strategy
- With the depth-first strategy, one interpretation is considered and expanded until it fails; only then is the second one considered.
- With the breadth-first strategy, both interpretations are considered alternately, each being expanded one step at a time
- Many parsers built today use the depth-first strategy because it tends to minimize the number of backup states needed and thus uses less memory and requires less bookkeeping

# Bottom up parsing

---

The basic operation in bottom-up parsing is to take a sequence of symbols and match it to the right-hand side of the rules.

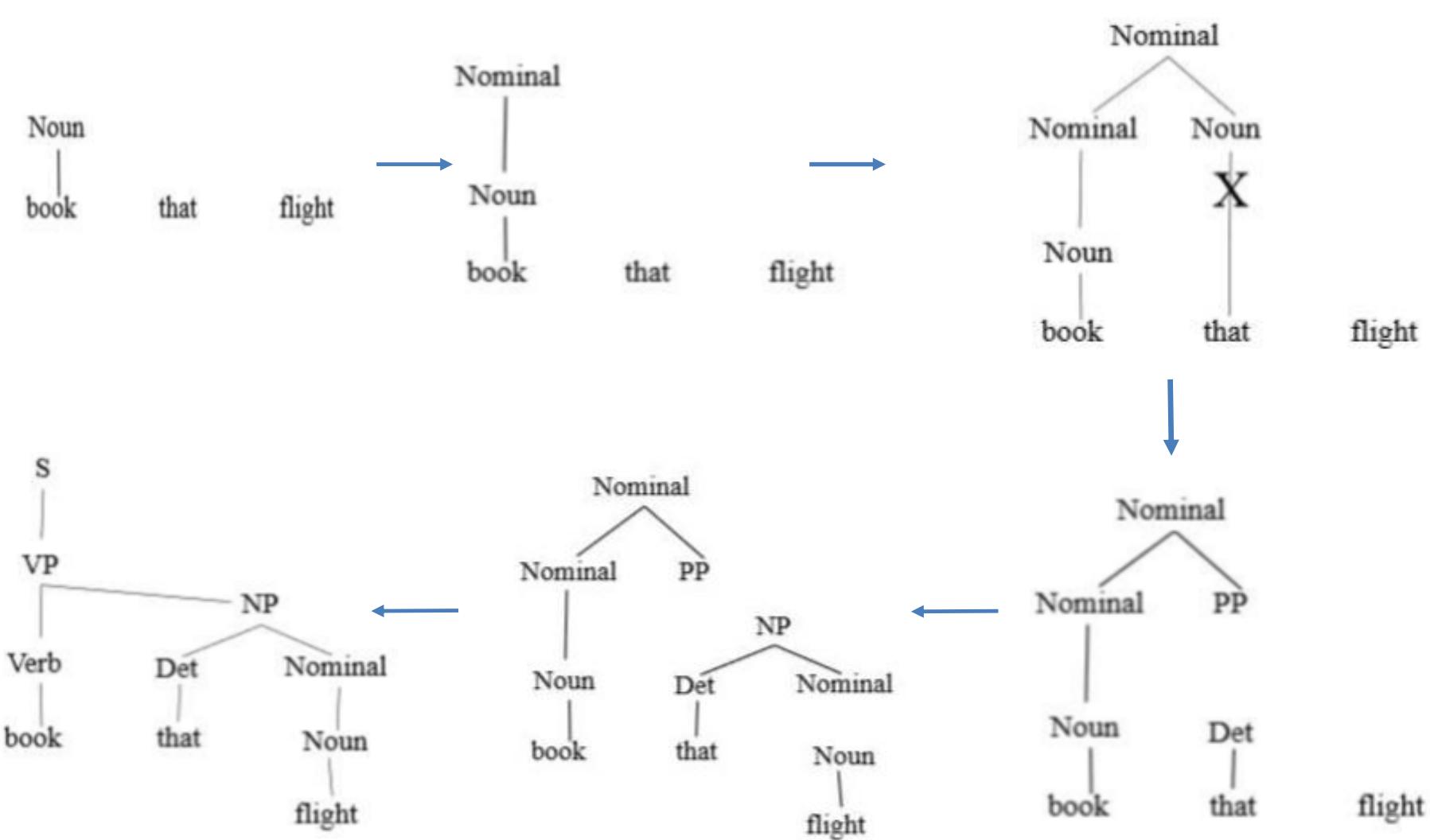
You could build a bottom-up parser simply by formulating this matching process as a search process.

The state would simply consist of a symbol list, starting with the words in the sentence.

Successor states could be generated by exploring all possible ways to

- Rewrite a word by its possible lexical categories
  - Replace a sequence of symbols that matches the right-hand side of a grammar rule by its left-hand side symbol
-

# Bottom up parsing



# Top Down vs Bottom Up

---

- Top down never explores options that will not lead to a full parse, but can explore many options that never connect to the actual sentence.
- Bottom up never explores options that do not connect to the actual sentence but can explore options that can never lead to a full parse.
- Relative amounts of wasted search depend on how much the grammar branches in each direction.

# Chart parsing

---

- Parser would tend to try the same matches again and again, thus duplicating much of its work unnecessarily.
- To avoid this problem, a data structure called a chart is introduced that allows the parser to store the partial results of the matching it has done so far so that the work need not be reduplicated.

Chart-based parsers can be considerably more efficient than parsers that rely only on a search because the same constituent is never constructed more than once. For instance, a pure top-down or bottom-up search strategy could require up to  $C^n$  operations to parse a sentence of length  $n$ , where  $C$  is a constant that depends on the specific algorithm you use. Even if  $C$  is very small, this exponential complexity rapidly makes the algorithm unusable. A chart-based parser, on the other hand, in the worst case would build every possible constituent between every possible pair of positions. This allows us to show that it has a worst-case complexity of  $K^*n^3$ , where  $n$  is the length of the sentence and  $K$  is a constant depending on the algorithm. Of course, a chart parser involves more work in each step, so  $K$  will be larger than  $C$ . To contrast the two approaches, assume that  $C$  is 10 and that  $K$  is a hundred times worse, 1000. Given a sentence of 12 words, the brute force search might take  $10^{12}$  operations (that is, 1,000,000,000,000), whereas the chart parser would take  $1000 * 12^3$  (that is, 1,728,000). Under these assumptions, the chart parser would be up to 500,000 times faster than the brute force search on some examples!

---

# Chart Parsing

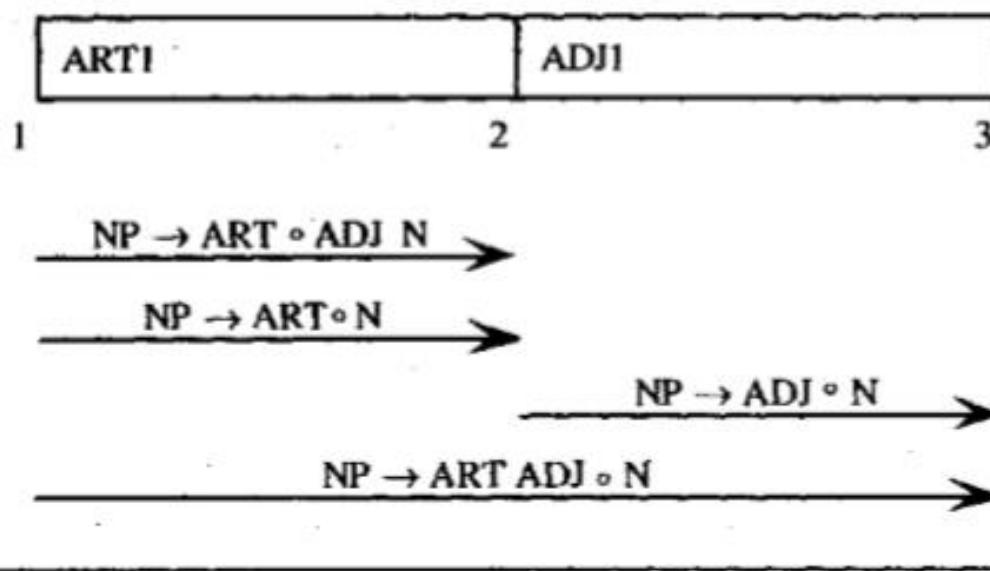
- The *Chart* allows storing partial analyses, so that they can be shared.
- Data structures used by the algorithm:
  - **The Key:** the current constituent we are attempting to “match”
  - **An Active Arc:** a grammar rule that has a partially matched RHS
  - **The Agenda:** Keeps track of newly found unprocessed constituents
  - **The Chart:** Records processed constituents (non-terminals) that span substrings of the input

# Chart Parsing

- 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

# Chart- active arcs

- The chart maintains the record of all the constituents derived from the sentence so far in the parse.
- Maintains the record of rules that have matched partially but are not complete. These are called the active arcs.



1.  $S \rightarrow NP VP$
2.  $NP \rightarrow ART N$
3.  $NP \rightarrow ART ADJ N$
4.  $VP \rightarrow V$
5.  $VP \rightarrow V NP$

Figure 3.9 The chart after seeing an ADJ in position 2

# Chart Parsing

Extending Active Arcs with a Key:

- Each **Active Arc** has the form:  $[A \rightarrow X_1 \dots \bullet C \dots X_m](p_i, p_j)$
- A Key constituent has the form:  $C(p_i, p_j)$
- When processing the Key  $C(p_1, p_2)$ , we search the active arc list for an arc  $[A \rightarrow X_1 \dots \bullet C \dots X_m](p_0, p_1)$ , and then create a new active arc  $[A \rightarrow X_1 \dots C \bullet \dots X_m](p_0, p_2)$
- If the new active arc is a completed rule:  $[A \rightarrow X_1 \dots C \bullet](p_0, p_2)$ , then we add  $A(p_0, p_2)$  to the Agenda
- After “using” the key to extend all relevant arcs, it is entered into the Chart

# Chart Parsing

Steps in the Process:

- Input is processed left-to-right, one word at a time
1. Find all POS of word (terminal-level)
  2. Initialize Agenda with all POS of the word
  3. Pick a Key from the Agenda
  4. Add all grammar rules that start with the Key as active arcs
  5. Extend any existing active arcs with the Key
  6. Add LHS constituents of newly completed rules to the Agenda
  7. Add the Key to the Chart
  8. If Agenda not empty - goto (3), else goto (1)

# 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 = The large can can hold the water**”

- (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)  $VP \rightarrow V NP$

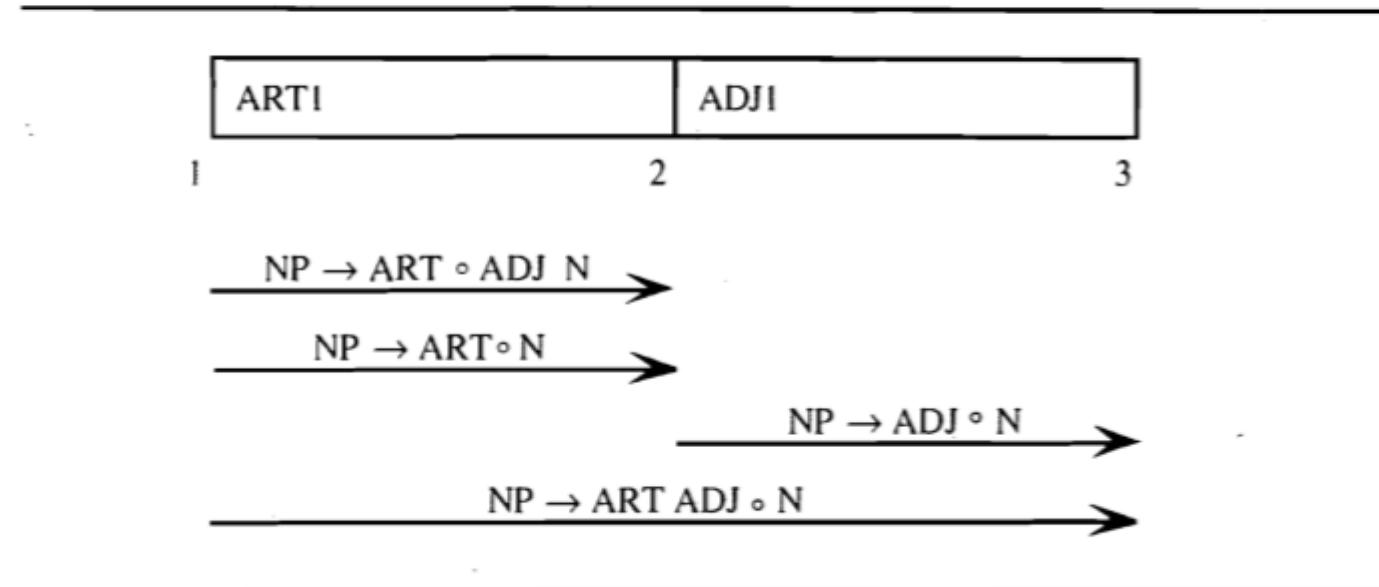


Figure 3.9 The chart after seeing an ADJ in position 2

# Chart parsing example

The input: “ $x = \text{The large can can hold the water}$ ”

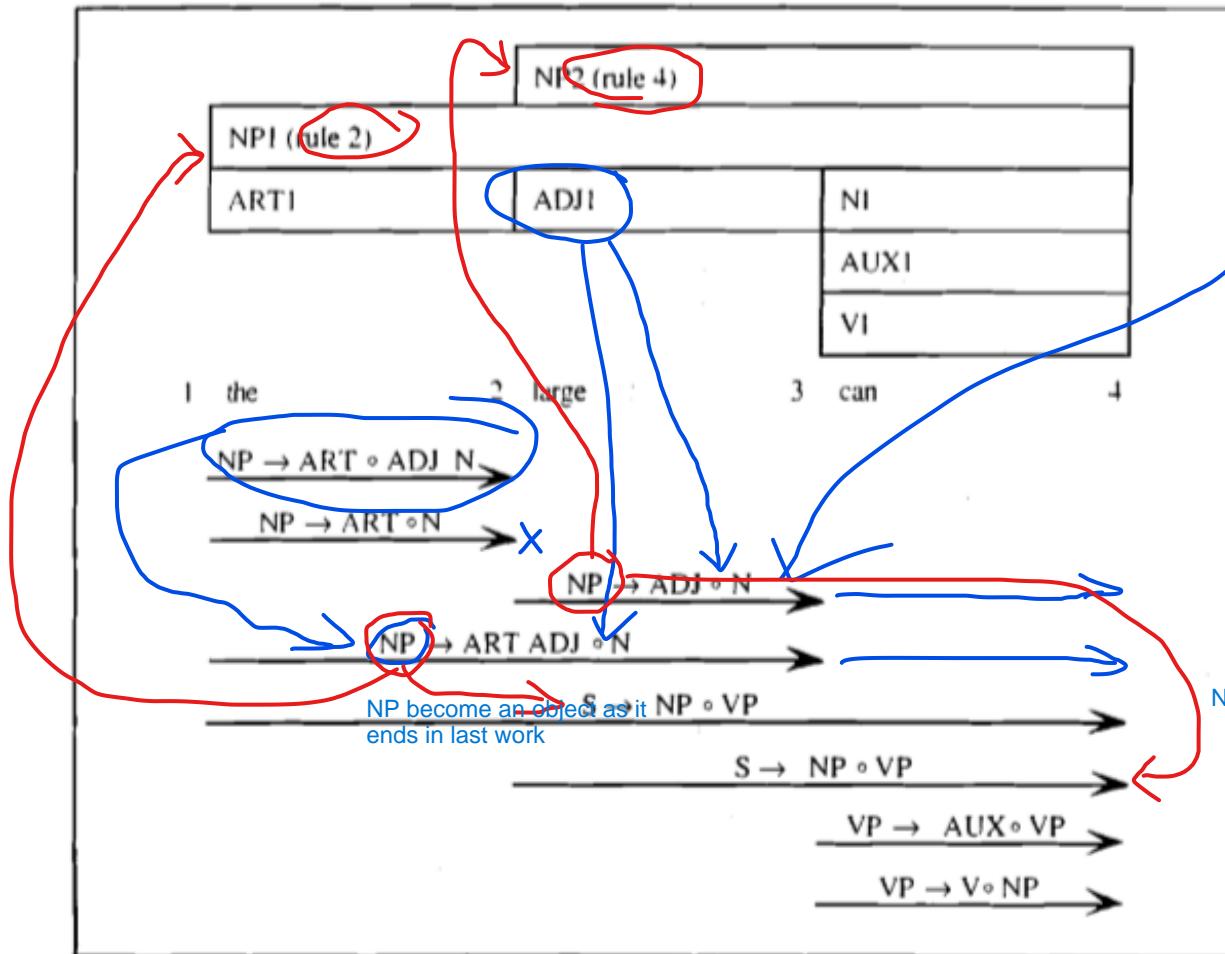


Figure 3.12 After parsing *the large can*

# Chart parsing example

The input: “ $x = \text{The large can can hold the water}$ ”

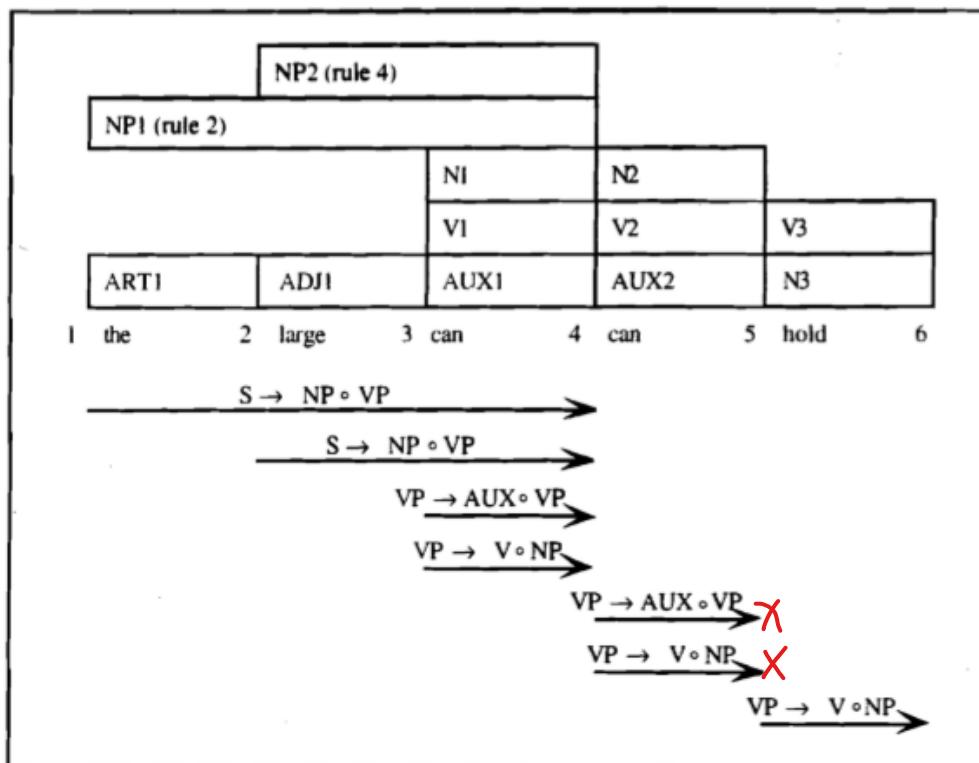
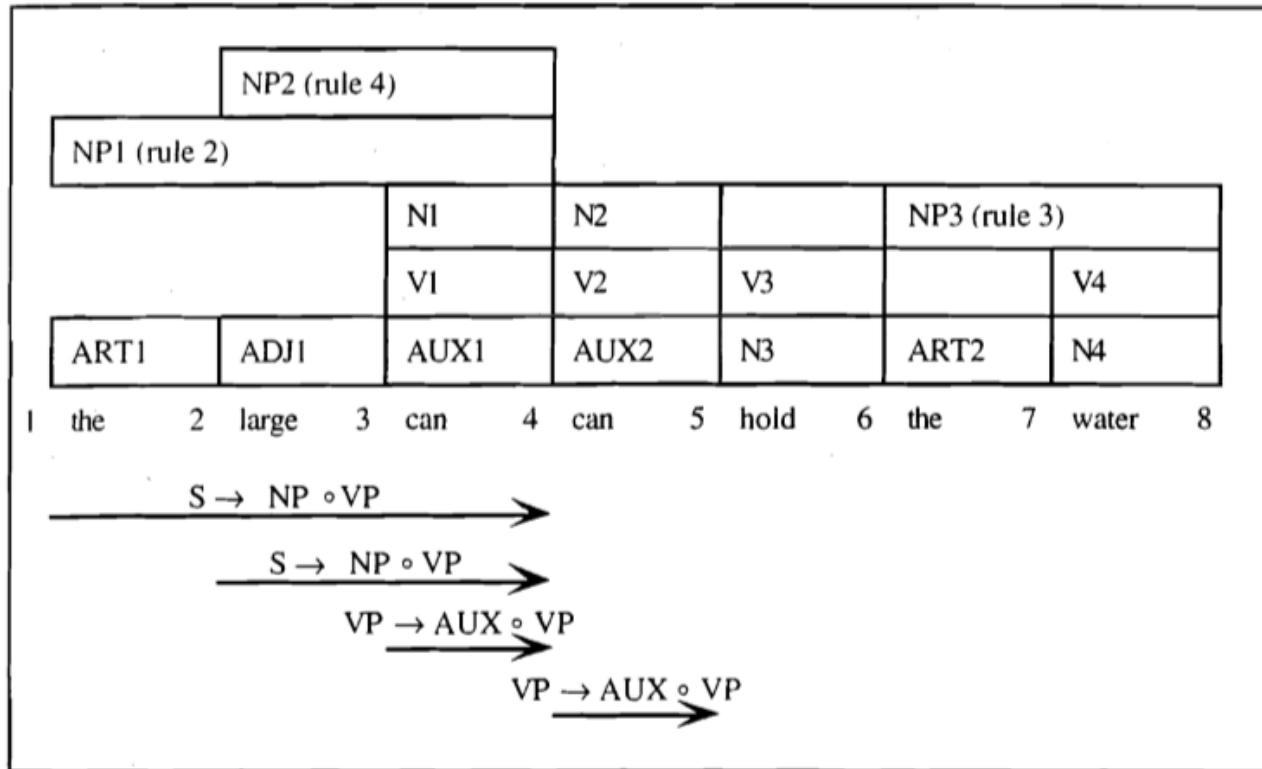


Figure 3.13 The chart after adding *hold*, omitting arcs generated for the first NP

- (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 = The large can can hold the water**”

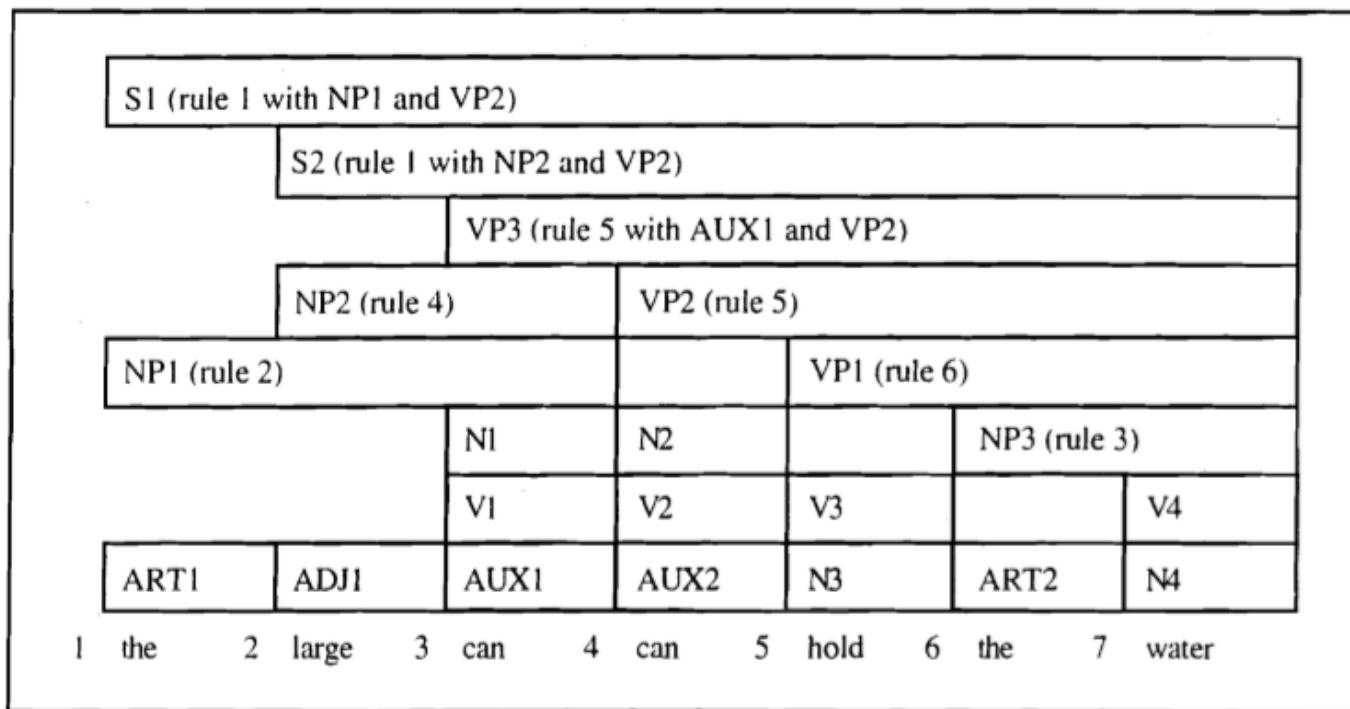


- (1)  $S \rightarrow NP VP$
- (2)  $NP \rightarrow ART ADJ N$
- (3)  $NP \rightarrow ART N$
- (4)  $NP \rightarrow ADJ N$
- (5)  $VP \rightarrow AUX VP$
- (6)  $VP \rightarrow V NP$

**Figure 3.14** The chart after all the NPs are found, omitting all but the crucial active arcs

# Final Chart

The input: “***x = The large can can hold the water***”



- (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)  $VP \rightarrow V NP$

- the: *ART*
- large: *ADJ*
- can: *N, AUX, V*
- hold: *N, V*
- water: *N, V*

Figure 3.15 The final chart

# Top down chart parsing

---

- Top-down methods have the advantage of being highly predictive.
- A word might be ambiguous in isolation, but if some of those possible categories cannot be used in a legal sentence, then these categories may never even be considered.

It turns out in the worst-case analysis that the top-down chart parser is not more efficient than the pure bottom-up chart parser. Both have a worst-case complexity of  $K^*n^3$  for a sentence of length  $n$ . In practice, however, the top-down method is considerably more efficient for any reasonable grammar.

---

# Top down arc induction

---

- Consider this new algorithm operating with the same grammar on “The large can can hold the water.”
- In the initialization stage, an arc labeled  $S \rightarrow o$  NP VP is added. Then, active arcs for each rule that can derive an NP are added:  $NP \rightarrow o$  ART ADJ N,  $NP \rightarrow o$  ART N,

## Top-Down Arc Introduction Algorithm

To add an arc  $S \rightarrow C_1 \dots o C_1 \dots C_n$  ending at position j, do the following:

For each rule in the grammar of form  $C_i \rightarrow X_1 \dots X_k$ , recursively add the new arc  $C_i \rightarrow o X_1 \dots X_k$  from position j to j.

# Top down chart parsing

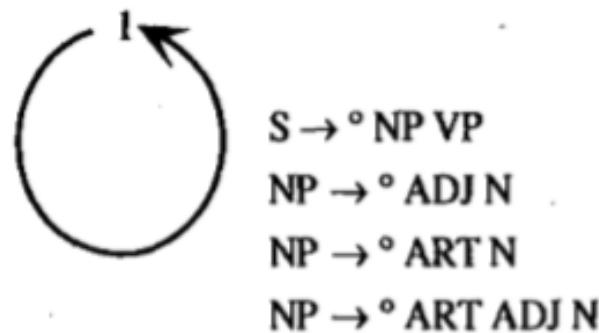
---

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

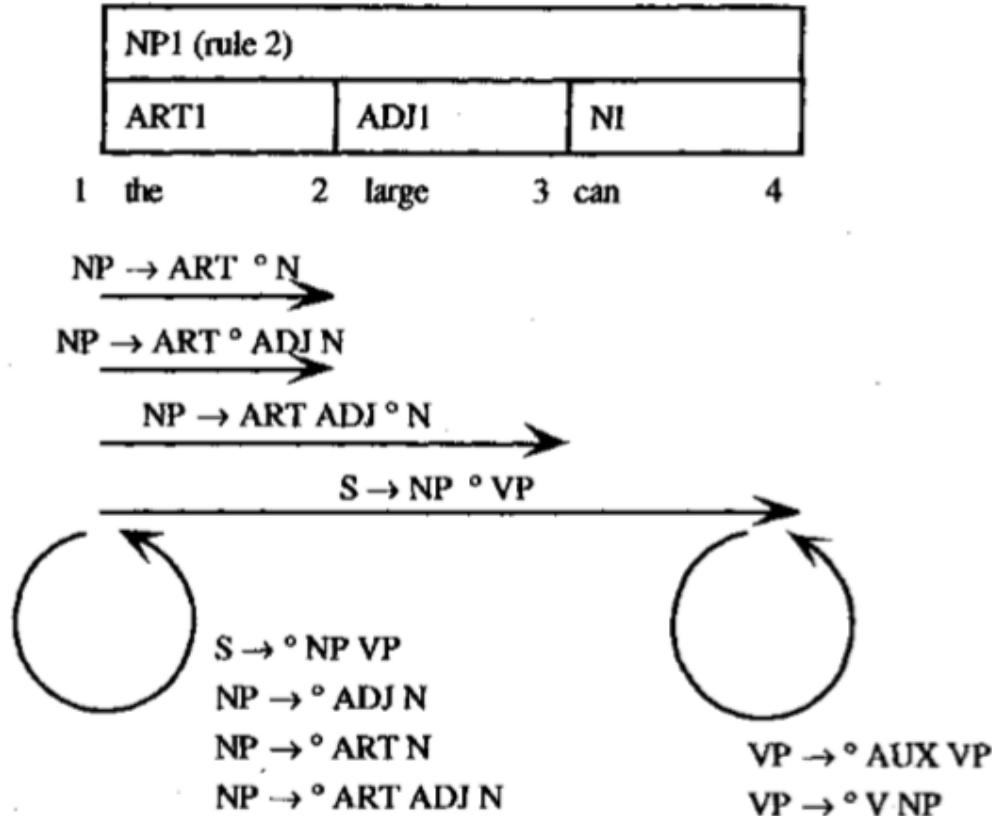


Figure 3.24 The chart after building the first NP

# Top down chart parsing example

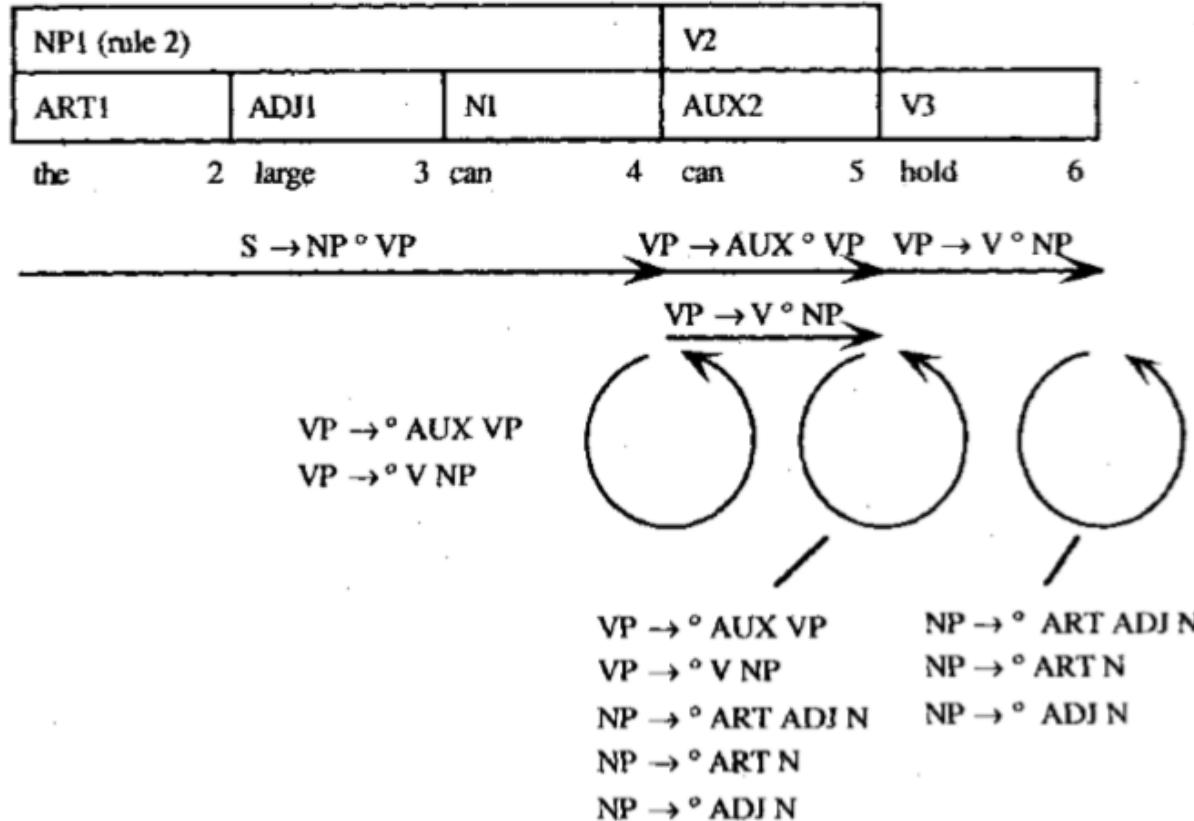


Figure 3.25 The chart after adding *hold*, omitting arcs generated for the first NP

# Top down chart parsing example

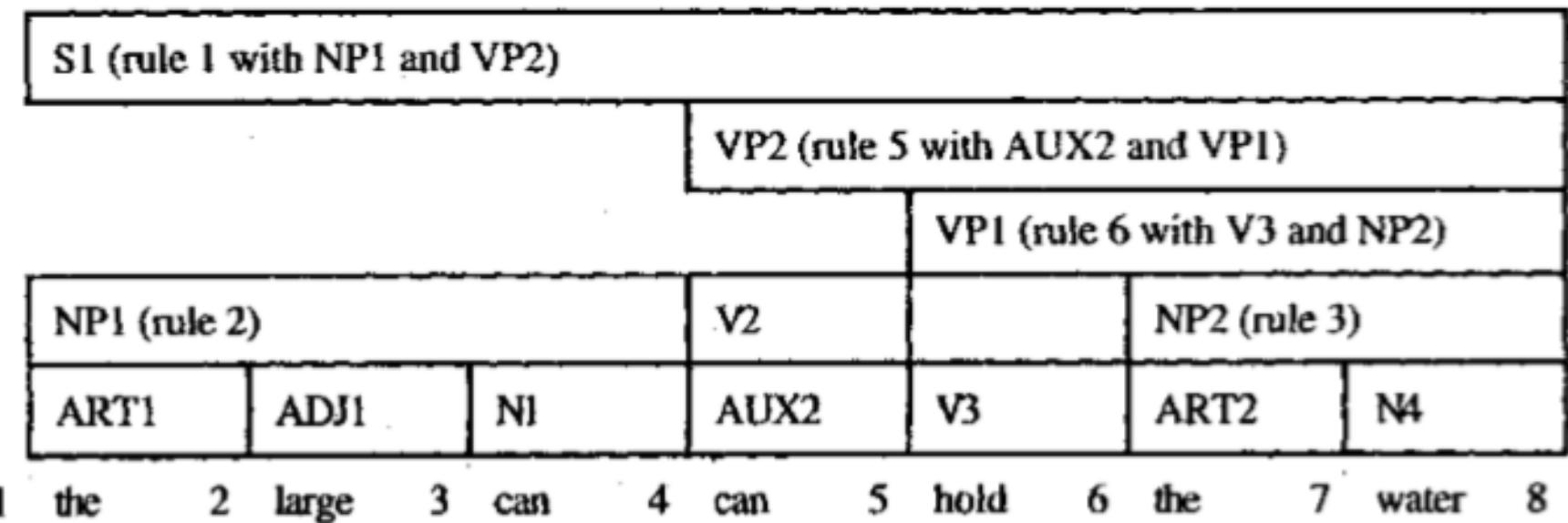


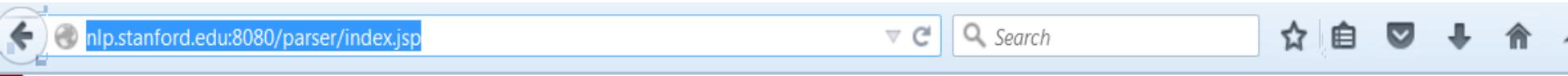
Figure 3.26 The final chart for the top-down filtering algorithm

# Chart Parsing

- When a chart parser begins parsing a text, it creates a new (empty) chart, spanning the text.
- It then incrementally adds new edges to the chart.
- A set of "chart rules" specifies the conditions under which new edges should be added to the chart.
- Once the chart reaches a stage where none of the chart rules adds any new edges, parsing is complete.

## Advantages of Chart Parsing

- No repeated computation of same sub problem
- Deals well with left-recursive grammars
- Deals well with ambiguity
- No backtracking necessary



## Stanford Parser

Please enter a sentence to be parsed:

```
I love natural language processing class
```

Language: English ▾      Sample Sentence

Parse

### Your query

*I love natural language processing class*

### Tagging

I/PRP love/VBP natural/JJ language/NN processing/NN class/NN

### Parse

```
(ROOT
  (S
    (NP (PRP I))
    (VP (VBP love)
      (NP (JJ natural) (NN language) (NN processing) (NN class)))))
```

---

# Parsing Implementation Demo

# Morphological Processing

---

- 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 (prefix or suffix).
- For instance, the word eaten consists of the root form eat and the suffix -en, which indicates the past participle form.
- Without any pre-processing, a lexicon would have to list all the forms of eat, including eats, eating, ate, and eaten.
- With preprocessing, there would be one morpheme eat that may combine with suffixes such as -ing, -s, and -en, and one entry for the irregular form ate. Thus the lexicon would only need to store two entries (eat and ate) rather than four.

# Finite State Models and Morphological Processing example

- Word happiest breaks down into the root form happy and the suffix -est, and thus does not need a separate entry in the lexicon
- An arc in an FST is labeled with a pair of symbols.
- For example, an arc labeled i:y could only be followed if the current input is the letter i and the output is the letter y.
- FSTs can be used to concisely represent the lexicon and to transform the surface form of words into a sequence of morphemes.

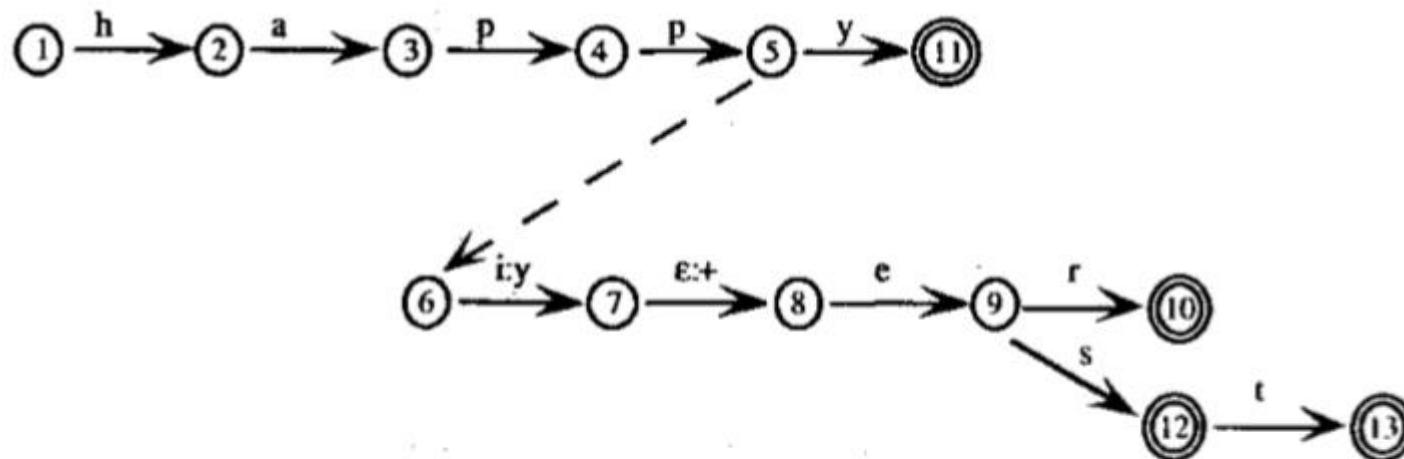
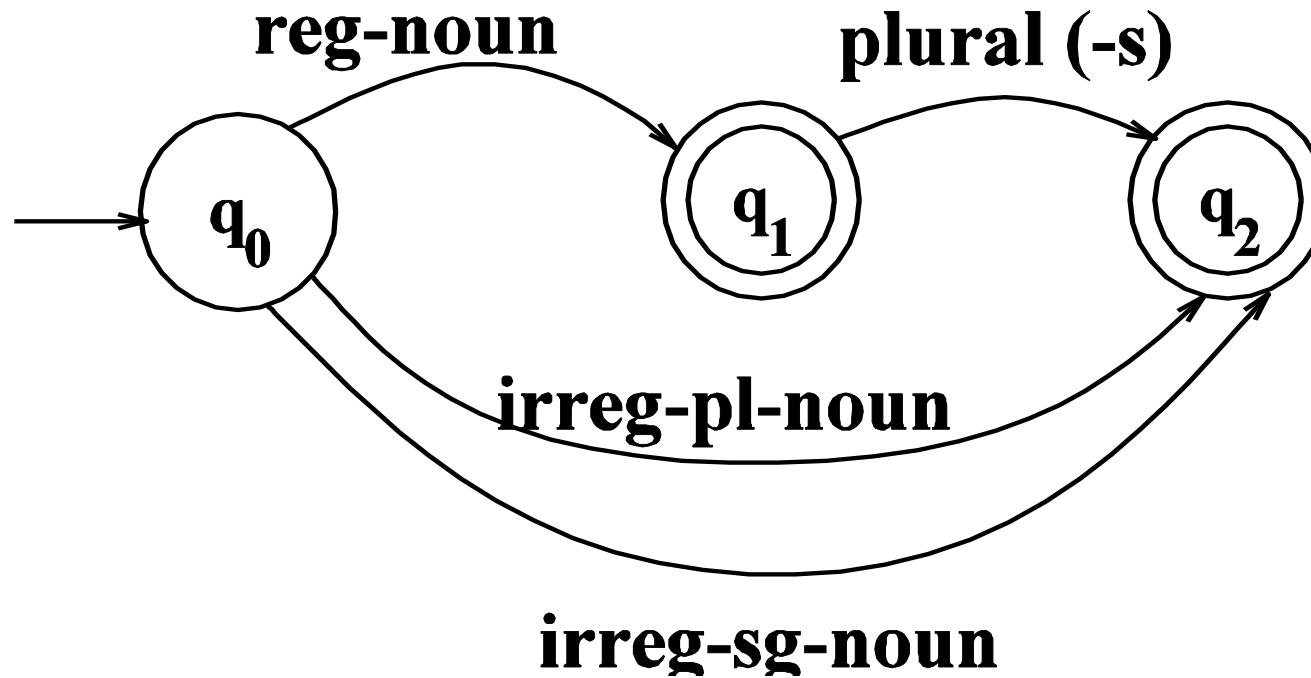


Figure 3.27 A simple FST for the forms of *happy*

# Simple Rules



# Parsing/Generation

---

- 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**

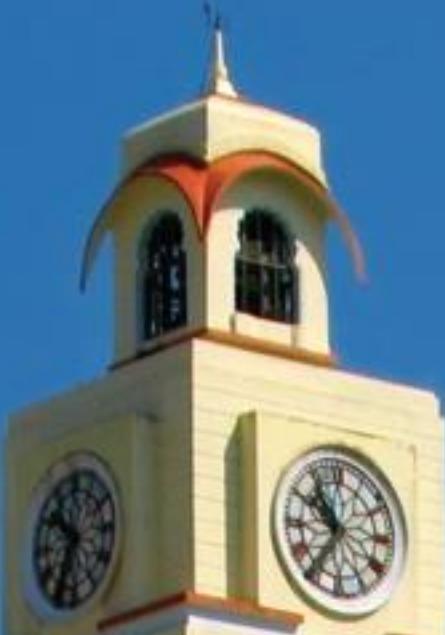
# References

---

- <http://www.ai.mit.edu/courses/6.863/lecture7-03.pdf>
- 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
- <https://nlp.stanford.edu/software/lex-parser.shtml>
- <https://www.nltk.org/api/nltk.parse.html>



**Thank you for your time!!**



# Natural Language Processing

## DSECL ZG565

Dr. Chetana Gavankar, Ph.D,

IIT Bombay-Monash University Australia

Associate Professor, BITS Pilani

[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



**BITS** Pilani  
Pilani Campus



## **Session 10 -Statistical Constituency Parsing**

### **Date – 19<sup>th</sup> August 2023**

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
-

# Some funny examples

---

- Policeman to little boy: “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

- “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

# 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.
  - Normal forms give us more structure to work with, resulting in easier parsing algorithms.
  - CNF provides an upper bound for parsing **complexity**

# 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.  $NP \rightarrow \text{Det Noun PP}$

\*Nominal definition is a noun, noun phrase, or any word or word group that functions as a noun. The term comes from the Latin, meaning "name."

# 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:
      - $NP \rightarrow \text{Det Nominal}$
      - $\text{Det} \rightarrow \text{the}$

# 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 example

Rule: Det → *the*

[0,1] for "the"



|   | 1   | 2 | 3 | 4 | 5 |
|---|-----|---|---|---|---|
| 0 | Det |   |   |   | S |
| 1 |     |   |   |   |   |
| 2 |     |   |   |   |   |
| 3 |     |   |   |   |   |
| 4 |     |   |   |   |   |

the            flight            includes            a            meal

# CKY example

Rule1: Det  $\rightarrow$  *the*  
Rule 2: N  $\rightarrow$  flight

[0,1] for "the"

[1,2] for "flight"

|   | 1   | 2 | 3 | 4 | 5 |
|---|-----|---|---|---|---|
| 0 | Det |   |   |   | S |
| 1 |     | N |   |   |   |
| 2 |     |   |   |   |   |
| 3 |     |   |   |   |   |
| 4 |     |   |   |   |   |

the

flight

includes

a

meal

# CKY example

Rule1: Det → *the*  
Rule 2: N → flight  
Rule 3: NP → Det N

[0,2] for "the flight"  
[0,1] for "the"  
[1,2] for "flight"

|   | 1   | 2  | 3 | 4 | 5 |
|---|-----|----|---|---|---|
| 0 | Det | NP |   |   | S |
| 1 |     | N  |   |   |   |
| 2 |     |    |   |   |   |
| 3 |     |    |   |   |   |
| 4 |     |    |   |   |   |

the                    flight                    includes                    a                    meal

# CKY: lexical step ( $j = 1$ )

- *The flight includes a meal.*

Lexical lookup

- Matches Det → the

|   | 1   | 2 | 3 | 4 | 5 |
|---|-----|---|---|---|---|
| 0 | Det |   |   |   |   |
| 1 |     |   |   |   |   |
| 2 |     |   |   |   |   |
| 3 |     |   |   |   |   |
| 4 |     |   |   |   |   |
| 5 |     |   |   |   |   |

# CKY: lexical step ( $j = 2$ )

- *The flight includes a meal.*

Lexical lookup

- Matches N → flight

|   | 1   | 2 | 3 | 4 | 5 |
|---|-----|---|---|---|---|
| 0 | Det |   |   |   |   |
| 1 |     | N |   |   |   |
| 2 |     |   |   |   |   |
| 3 |     |   |   |   |   |
| 4 |     |   |   |   |   |
| 5 |     |   |   |   |   |

# CKY: syntactic step ( $j = 2$ )

- *The flight includes a meal.*

Syntactic lookup:

- look backwards and see if there is any rule that will cover what we've done so far.

|   | 1   | 2  | 3 | 4 | 5 |
|---|-----|----|---|---|---|
| 0 | Det | NP |   |   |   |
| 1 |     | N  |   |   |   |
| 2 |     |    |   |   |   |
| 3 |     |    |   |   |   |
| 4 |     |    |   |   |   |
| 5 |     |    |   |   |   |

# CKY: lexical step ( $j = 3$ )

- *The flight includes a meal.*

Lexical lookup

- Matches V → includes

|   | 1   | 2  | 3 | 4 | 5 |
|---|-----|----|---|---|---|
| 0 | Det | NP |   |   |   |
| 1 |     | N  |   |   |   |
| 2 |     |    | V |   |   |
| 3 |     |    |   |   |   |
| 4 |     |    |   |   |   |
| 5 |     |    |   |   |   |

# CKY: lexical step ( $j = 3$ )

- *The flight includes a meal.*

Syntactic lookup

- There are no rules in our grammar that will cover Det, NP, V

|          | <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> |          |          |          |          |          |

# CKY: lexical step ( $j = 4$ )

- *The flight includes a meal.*

Lexical lookup

- Matches Det → a

|   | 1   | 2  | 3 | 4   | 5 |
|---|-----|----|---|-----|---|
| 0 | Det | NP |   |     |   |
| 1 |     | N  |   |     |   |
| 2 |     |    | V |     |   |
| 3 |     |    |   | Det |   |
| 4 |     |    |   |     |   |
| 5 |     |    |   |     |   |

# CKY: lexical step ( $j = 5$ )

- *The flight includes a meal.*

Lexical lookup

- Matches N → meal

|          | <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        |

# CKY: syntactic step ( $j = 5$ )

- *The flight includes a meal.*

Syntactic lookup

- We find that we have  
 $NP \rightarrow Det\ N$

|          | <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        |

# CKY: syntactic step ( $j = 5$ )

- *The flight includes a meal.*

Syntactic lookup

- We find that we have  
 $VP \rightarrow V \ NP$

|          | <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        |

# CKY: syntactic step ( $j = 5$ )

- *The flight includes a meal.*

Syntactic lookup

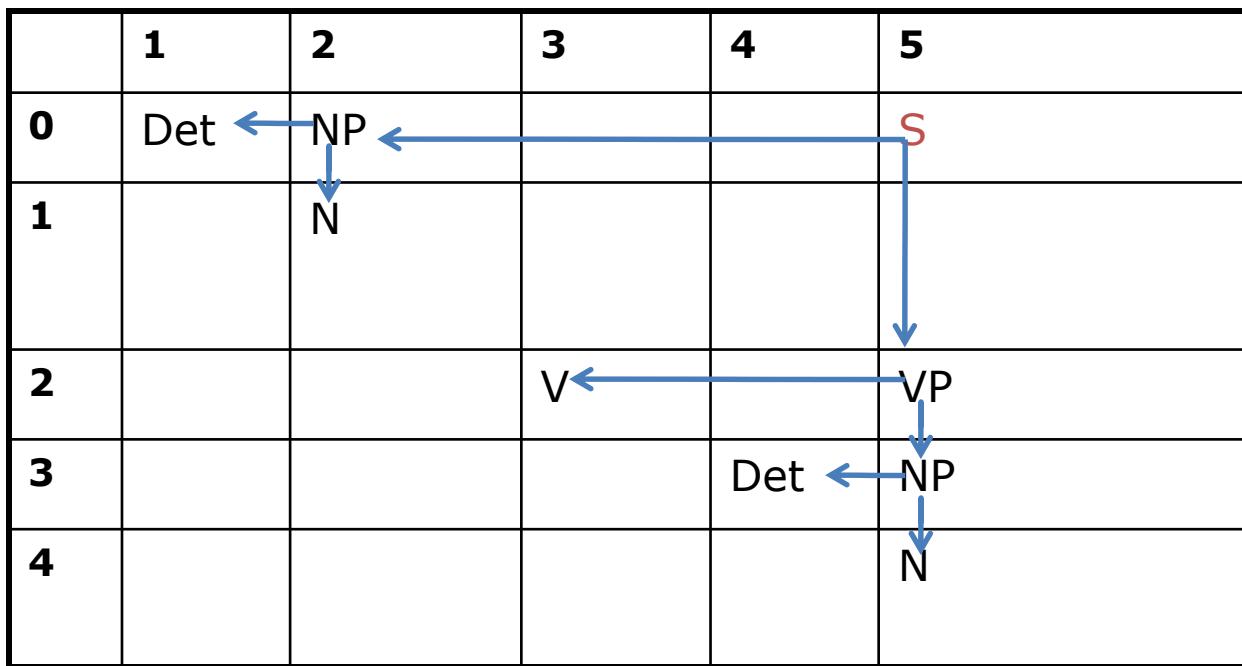
- We find that we have  
 $S \rightarrow NP VP$

|          | <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        |

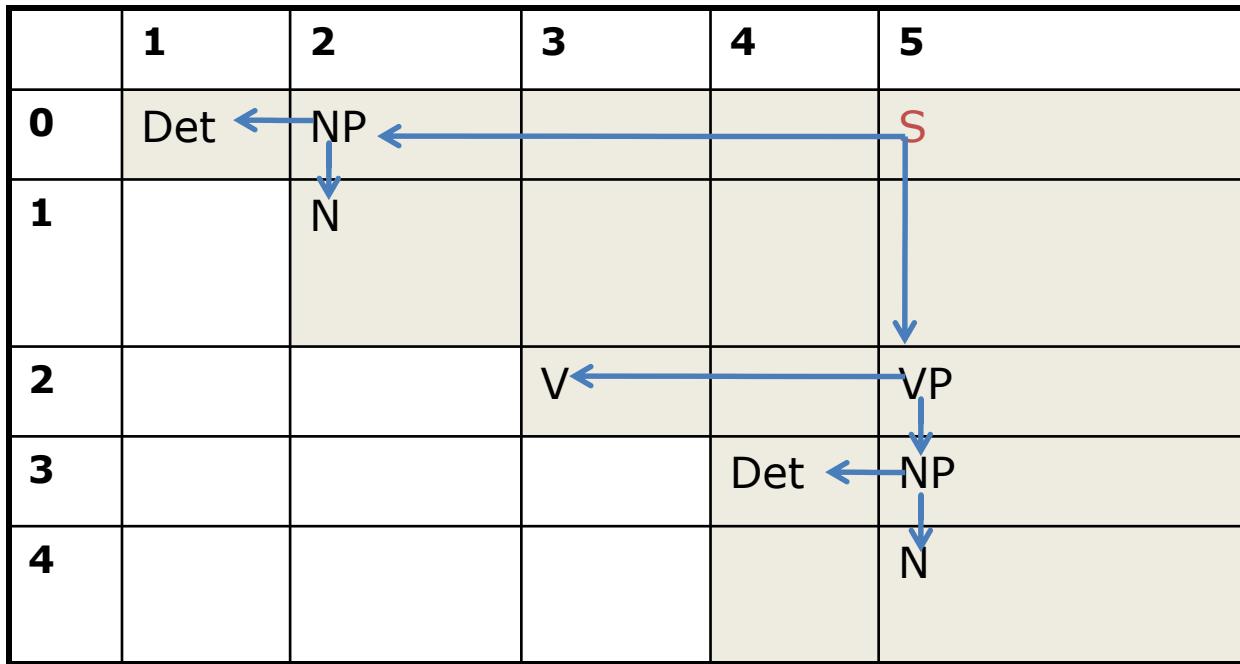
# From recognition to parsing

- 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



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.

# 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”)

# 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)

# 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:  $\forall i \quad \sum_j P(N^i \rightarrow \xi^j) = 1$

# 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.

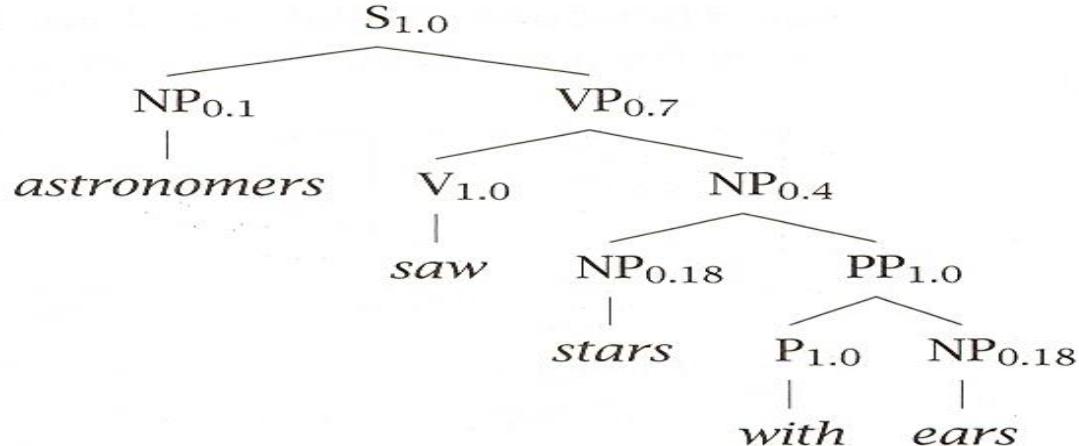
- The probability of a sentence (according to grammar  $G$ ) is given by:

$$P(w_{1m}) = \sum_t P(w_{1m}, t) = \sum_{\{t: \text{yield}(t)=w_{1m}\}} P(t)$$

where  $t$  is a parse tree of the sentence. Need dynamic programming to make this efficient!

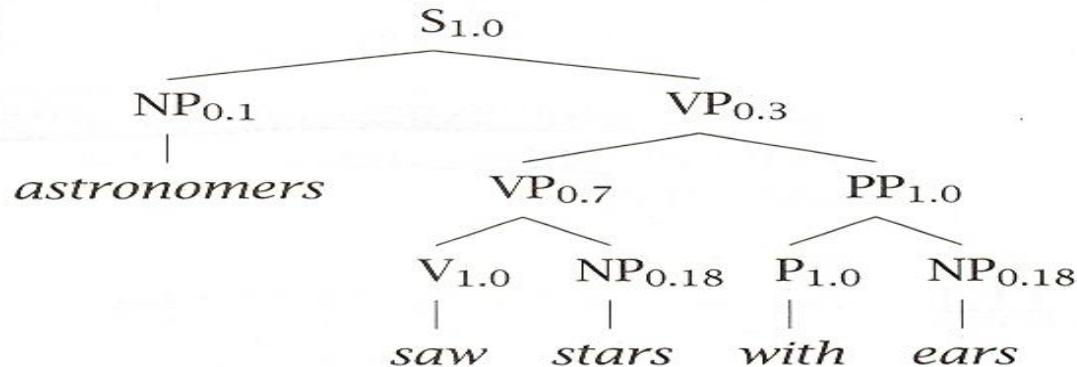
# Example: Probability of a Derivation Tree

$t_1:$



|                         |     |                              |      |
|-------------------------|-----|------------------------------|------|
| $S \rightarrow NP\ VP$  | 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  |

$t_2:$



# 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

$$\arg \max_{t \in T(s)} p(t)$$

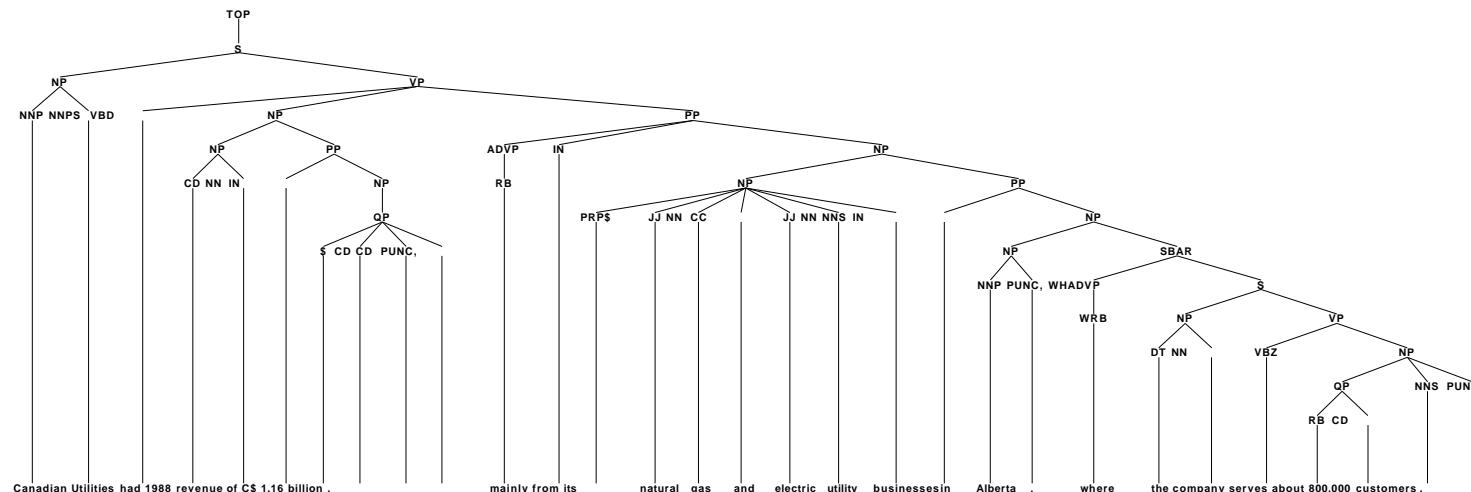
# PCFG based Grammar

- ▶ PCFGs augments CFGs by including a probability for each rule in the grammar.
- ▶ The probability for a parse tree is the product of probabilities for the rules in the tree
- ▶ To build a PCFG-parsed parser:
  1. Learn a PCFG from a treebank
  2. Given a test data sentence, use the CKY algorithm to compute the highest probability tree for the sentence under the PCFG

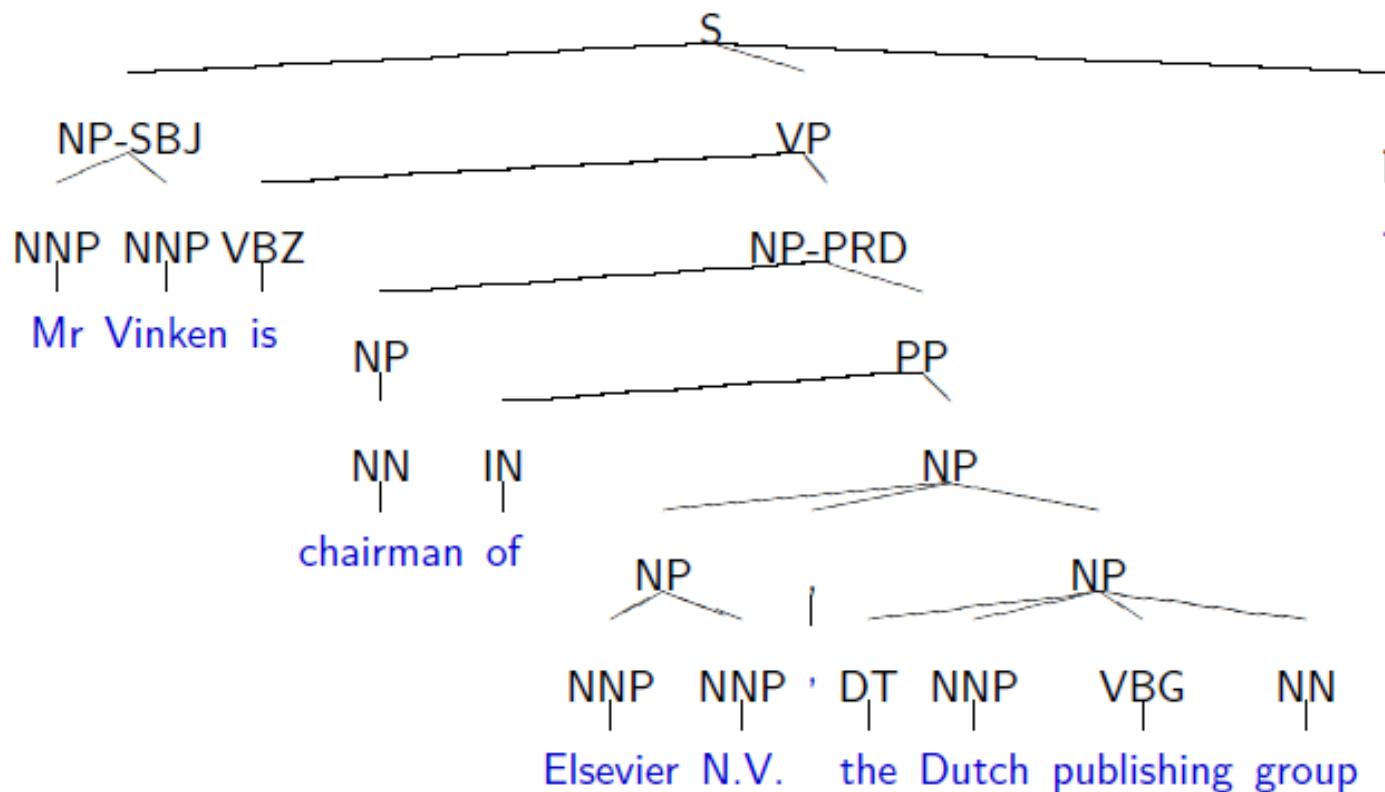
# Data for Parsing Experiments: Treebanks

- ▶ Penn WSJ Treebank = 50,000 sentences with associated trees
- ▶ Usual set-up: 40,000 training sentences, 2400 test sentences

An example tree:



# Example tree



# Characteristics of PCFGs

- In a PCFG, the probability  $P(A \rightarrow \beta)$  expresses the likelihood that the non-terminal A will expand as  $\beta$ .
  - e.g. the likelihood that  $S \rightarrow NP\ VP$ 
    - (as opposed to  $S \rightarrow VP$ , or  $S \rightarrow NP\ VP\ PP$ , or... )
- can be interpreted as a conditional probability:
  - probability of the expansion, given the LHS non-terminal
  - $P(A \rightarrow \beta) = P(A \rightarrow \beta | A)$
- Therefore, for any non-terminal A, probabilities of every rule of the form  $A \rightarrow \beta$  must sum to 1
  - in this case, we say the PCFG is **consistent**

# 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

$$P(\text{the}|\text{ART}) = 300 / 558 = 0.54$$

|       |     |       |     |
|-------|-----|-------|-----|
| ReARD | .54 | ReART | .36 |
| ReARD | .05 | ReAN  | .01 |
| ReASY | .05 | ReARD | .03 |
| ReASY | .1  | ReASY | .5  |
| ReED  | .08 | ReEND | .05 |
| ReEN  | .02 |       |     |

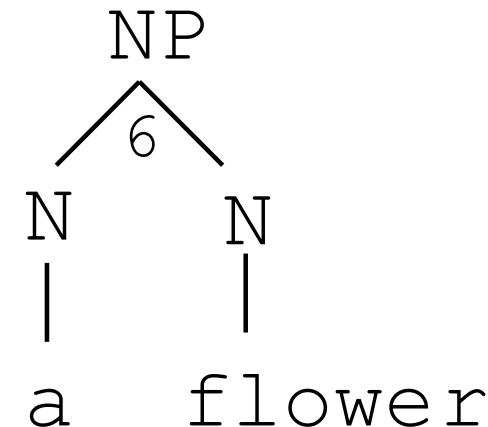
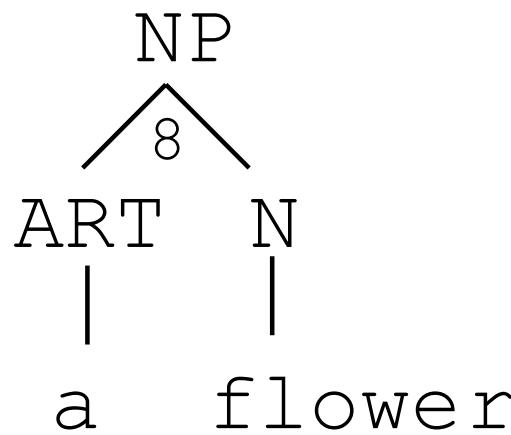
# The PCFG

- Below is a probabilistic CFG (PCFG) with probabilities derived from analyzing a parsed version of Allen's corpus.

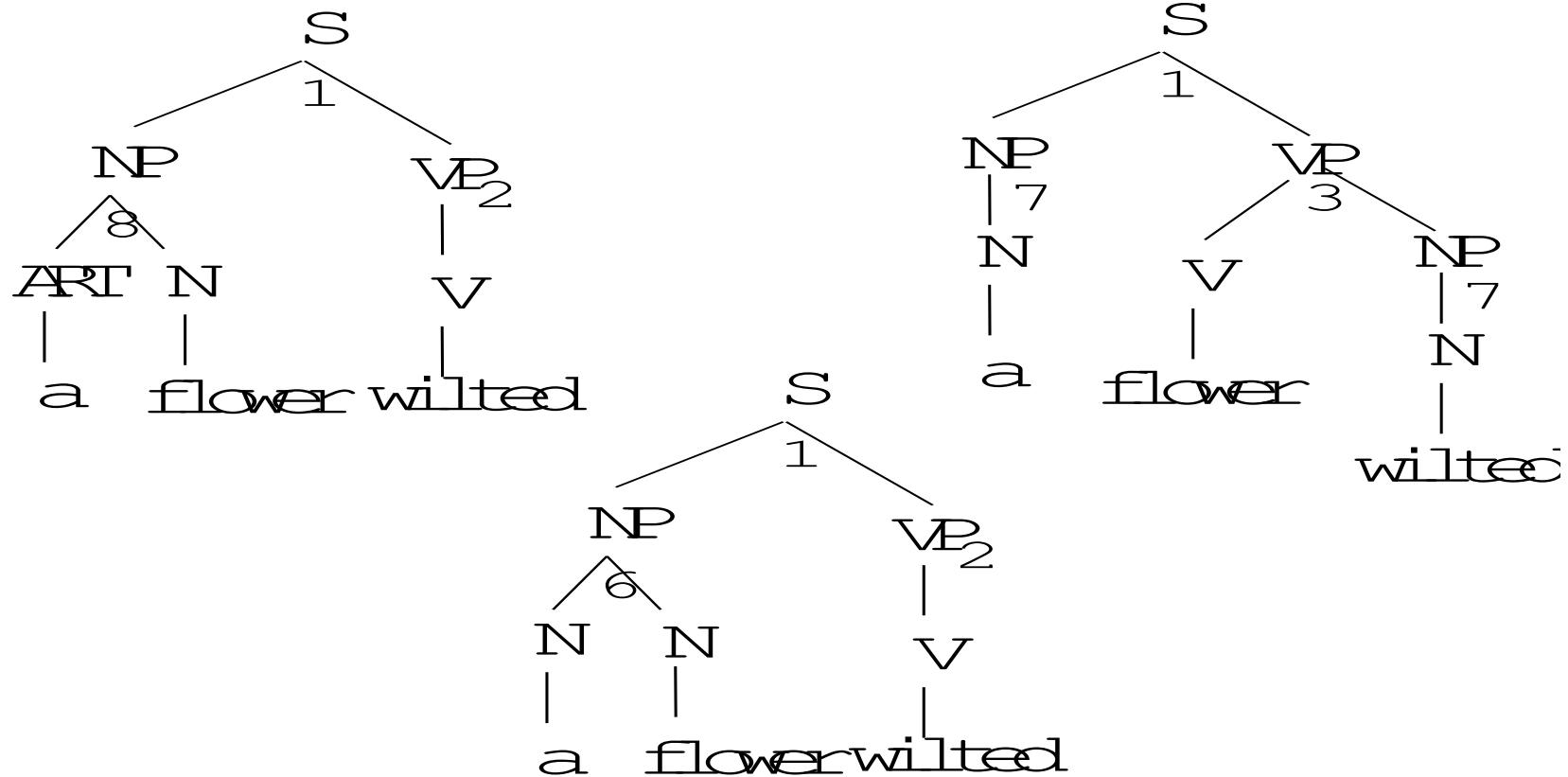
| Rule                     | Count for<br>IHS | Count for<br>Rule | Prob |
|--------------------------|------------------|-------------------|------|
| 1. $S \rightarrow NP VP$ | 300              | 300               | 1    |
| 2 $VP \rightarrow V$     | 300              | 116               | .386 |
| 3 $VP \rightarrow VNP$   | 300              | 118               | .393 |
| 4 $VP \rightarrow VNPPP$ | 300              | 66                | .22  |
| 5 $NP \rightarrow NPP$   | 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.

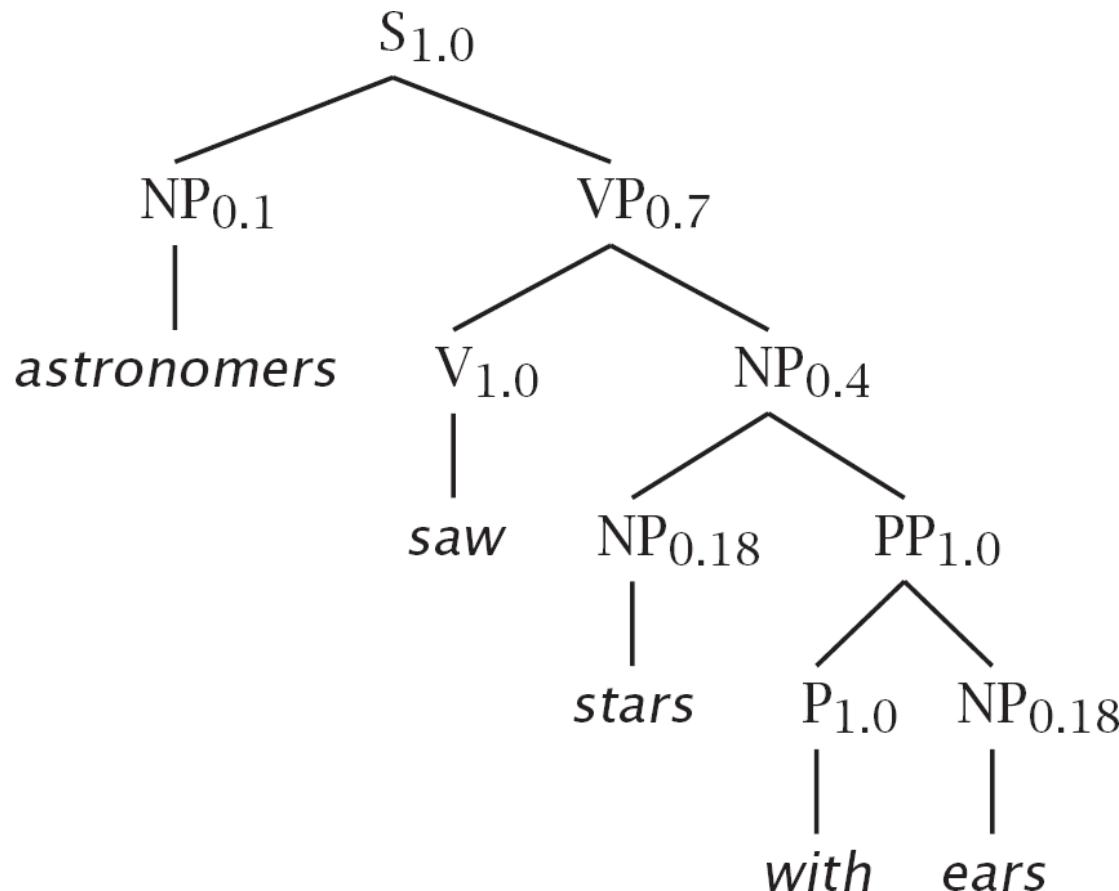
# Example

|                             |     |                                     |      |
|-----------------------------|-----|-------------------------------------|------|
| $S \rightarrow NP\ VP$      | 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

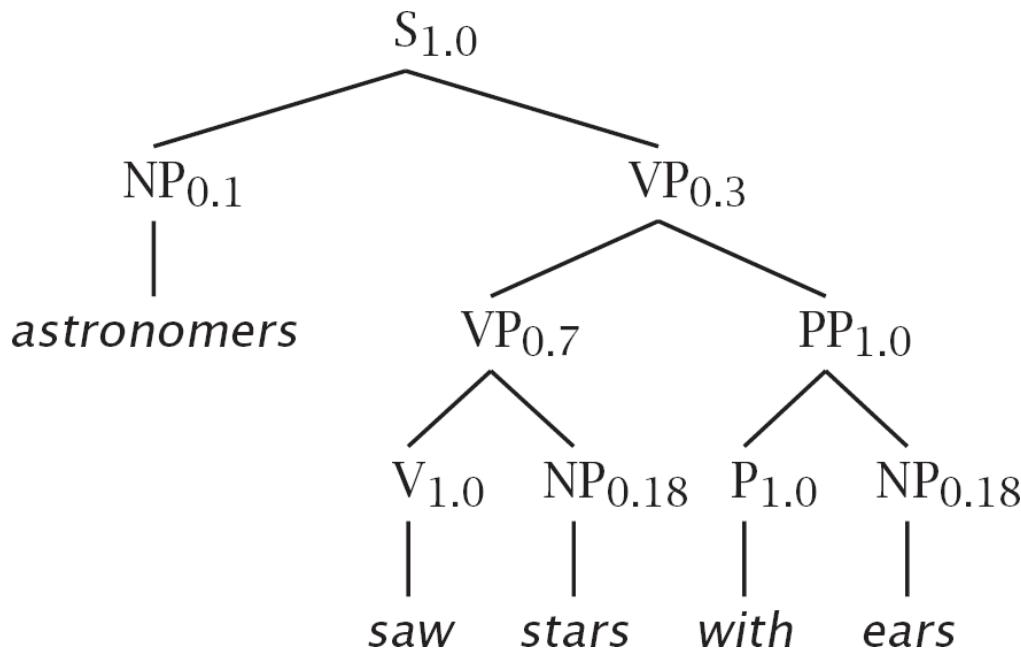
# astronomers saw stars with ears

$t_1:$



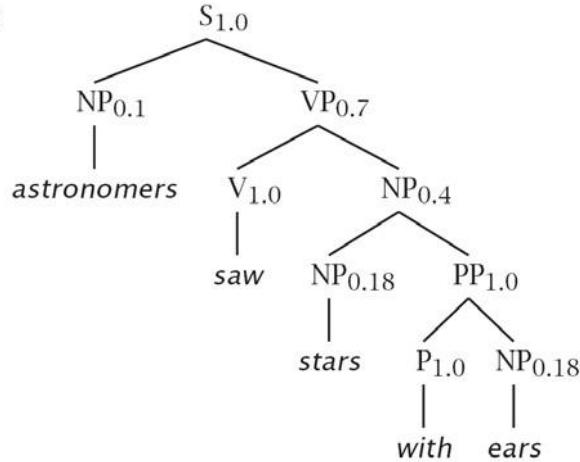
# astronomers saw stars with ears

$t_2:$



# Probabilities

$t_1:$

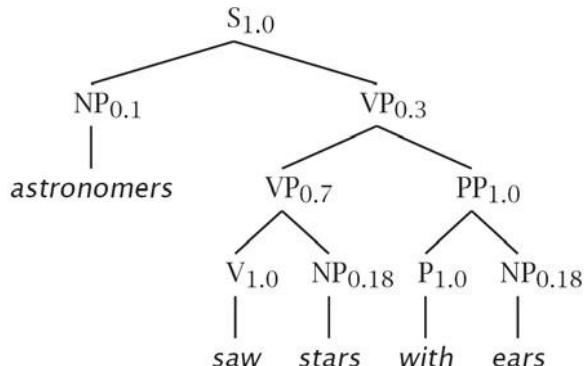


$$\begin{aligned}
 P(t_1) &= 1.0 \times 0.1 \times 0.7 \times 1.0 \times 0.4 \\
 &\quad \times 0.18 \times 1.0 \times 1.0 \times 0.18 \\
 &= 0.0009072
 \end{aligned}$$

$$\begin{aligned}
 P(t_2) &= 1.0 \times 0.1 \times 0.3 \times 0.7 \times 1.0 \\
 &\quad \times 0.18 \times 1.0 \times 1.0 \times 0.18 \\
 &= 0.0006804
 \end{aligned}$$

$$P(w_{15}) = P(t_1) + P(t_2) = 0.0015876$$

$t_2:$



# 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 calculate 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 PCFG

- $S \rightarrow NP\ VP [.80]$
- $NP \rightarrow Det\ N [.30]$
- $VP \rightarrow V\ NP [.20]$
- $V \rightarrow \text{includes} [.05]$
- $Det \rightarrow \text{the} [.4]$
- $Det \rightarrow a [.4]$
- $N \rightarrow \text{meal} [.01]$
- $N \rightarrow \text{flight} [.02]$

# Probabilistic CYK: initialisation

- *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> |          |          |          |          |          |
| <b>1</b> |          |          |          |          |          |
| <b>2</b> |          |          |          |          |          |
| <b>3</b> |          |          |          |          |          |
| <b>4</b> |          |          |          |          |          |
| <b>5</b> |          |          |          |          |          |

# Probabilistic CYK: lexical step

- *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]

|   | 1           | 2 | 3 | 4 | 5 |
|---|-------------|---|---|---|---|
| 0 | Det<br>(.4) |   |   |   |   |
| 1 |             |   |   |   |   |
| 2 |             |   |   |   |   |
| 3 |             |   |   |   |   |
| 4 |             |   |   |   |   |
| 5 |             |   |   |   |   |

# Probabilistic CYK: lexical step

- *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]

|   | 1           | 2        | 3 | 4 | 5 |
|---|-------------|----------|---|---|---|
| 0 | Det<br>(.4) |          |   |   |   |
| 1 |             | N<br>.02 |   |   |   |
| 2 |             |          |   |   |   |
| 3 |             |          |   |   |   |
| 4 |             |          |   |   |   |
| 5 |             |          |   |   |   |

# Probabilistic CYK: syntactic step

- *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]

|   | 1           | 2        | 3 | 4 | 5 |
|---|-------------|----------|---|---|---|
| 0 | Det<br>(.4) | NP<br>.3 |   |   |   |
| 1 |             | N<br>.02 |   |   |   |
| 2 |             |          |   |   |   |
| 3 |             |          |   |   |   |
| 4 |             |          |   |   |   |
| 5 |             |          |   |   |   |

Note: probability of NP in [0,2]  
 $P(Det \rightarrow the) * P(N \rightarrow meal) * P(NP \rightarrow Det\ N)$

# Probabilistic CYK: lexical step

- *The flight includes a meal.*
  - $S \rightarrow NP\ VP [ .80 ]$
  - $NP \rightarrow Det\ N [ .30 ]$
  - $VP \rightarrow V\ NP [ .20 ]$
  - $V \rightarrow \text{includes} [ .05 ]$
  - $Det \rightarrow \text{the} [ .4 ]$
  - $Det \rightarrow a [ .4 ]$
  - $N \rightarrow \text{meal} [ .01 ]$
  - $N \rightarrow \text{flight} [ .02 ]$

|   | 1           | 2        | 3        | 4 | 5 |
|---|-------------|----------|----------|---|---|
| 0 | Det<br>(.4) | NP<br>.3 |          |   |   |
| 1 |             | N<br>.02 |          |   |   |
| 2 |             |          | V<br>.05 |   |   |
| 3 |             |          |          |   |   |
| 4 |             |          |          |   |   |
| 5 |             |          |          |   |   |

# Probabilistic CYK: lexical step

- *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<br>(.4) | NP<br>.3 |          |           |          |
| <b>1</b> |             | N<br>.02 |          |           |          |
| <b>2</b> |             |          | V<br>.05 |           |          |
| <b>3</b> |             |          |          | Det<br>.4 |          |
| <b>4</b> |             |          |          |           |          |
| <b>5</b> |             |          |          |           |          |

# Probabilistic CYK: syntactic step

- *The flight includes a meal.*

- $S \rightarrow NP\ VP$  [.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]

|          | <b>1</b>    | <b>2</b> | <b>3</b> | <b>4</b>  | <b>5</b> |
|----------|-------------|----------|----------|-----------|----------|
| <b>0</b> | Det<br>(.4) | NP<br>.3 |          |           |          |
| <b>1</b> |             | N<br>.02 |          |           |          |
| <b>2</b> |             |          | V<br>.05 |           |          |
| <b>3</b> |             |          |          | Det<br>.4 |          |
| <b>4</b> |             |          |          |           | N<br>.01 |

# Probabilistic CYK: syntactic step

- *The flight includes a meal.*

- $S \rightarrow NP\ VP\ [.80]$
- $NP \rightarrow Det\ N\ [.30]$
- $VP \rightarrow V\ NP\ [.20]$
- $V \rightarrow includes\ [.05]$
- $Det \rightarrow the\ [.4]$
- $Det \rightarrow a\ [.4]$
- $N \rightarrow meal\ [.01]$
- $N \rightarrow flight\ [.02]$

|   | 1           | 2        | 3        | 4         | 5        |
|---|-------------|----------|----------|-----------|----------|
| 0 | Det<br>(.4) | NP<br>.3 |          |           |          |
| 1 |             | N<br>.02 |          |           |          |
| 2 |             |          | V<br>.05 |           |          |
| 3 |             |          |          | Det<br>.4 | NP<br>.3 |
| 4 |             |          |          |           | N<br>.01 |

# Probabilistic CYK: syntactic step

- *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]

|   | 1           | 2        | 3        | 4         | 5         |
|---|-------------|----------|----------|-----------|-----------|
| 0 | Det<br>(.4) | NP<br>.3 |          |           |           |
| 1 |             | N<br>.02 |          |           |           |
| 2 |             |          | V<br>.05 |           | VP<br>0.2 |
| 3 |             |          |          | Det<br>.4 | NP<br>.3  |
| 4 |             |          |          |           | N<br>.01  |

# Probabilistic CYK: syntactic step

- *The flight includes a meal.*

- $S \rightarrow NP\ VP [ .80 ]$
- $NP \rightarrow Det\ N [ .30 ]$
- $VP \rightarrow V\ NP [ .20 ]$
- $V \rightarrow includes [ .05 ]$
- $Det \rightarrow the [ .4 ]$
- $Det \rightarrow a [ .4 ]$
- $N \rightarrow meal [ .01 ]$
- $N \rightarrow flight [ .02 ]$

|   | 1         | 2        | 3        | 4         | 5   |
|---|-----------|----------|----------|-----------|---|
| 0 | Det<br>.4 | NP<br>.3 |          |           | $S$<br>$(.8 * .3 * .4 * .02$<br>$* .2 * .05 * .3 * .4 * .01)$ |
| 1 |           | N<br>.02 |          |           |   |
| 2 |           |          | V<br>.05 |           | VP<br>.2  |
| 3 |           |          |          | Det<br>.4 | NP<br>.3  |
| 4 |           |          |          |           | N<br>.01  |

# Probabilistic CYK: summary

- Cells in chart hold probabilities
- Bottom-up procedure computes probability of a parse incrementally.
- To obtain parse trees, we traverse the table “backwards” as before.
  - Cells need to be augmented with backpointers.

---

# Probabilistic Parsing Implementation Demo

# 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)
- No Structural Context
  - How NP expands depends on position

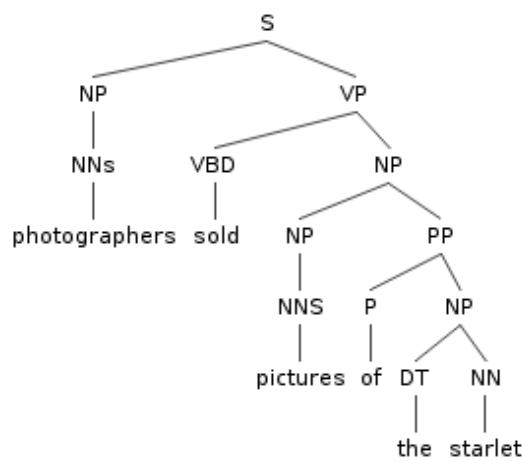
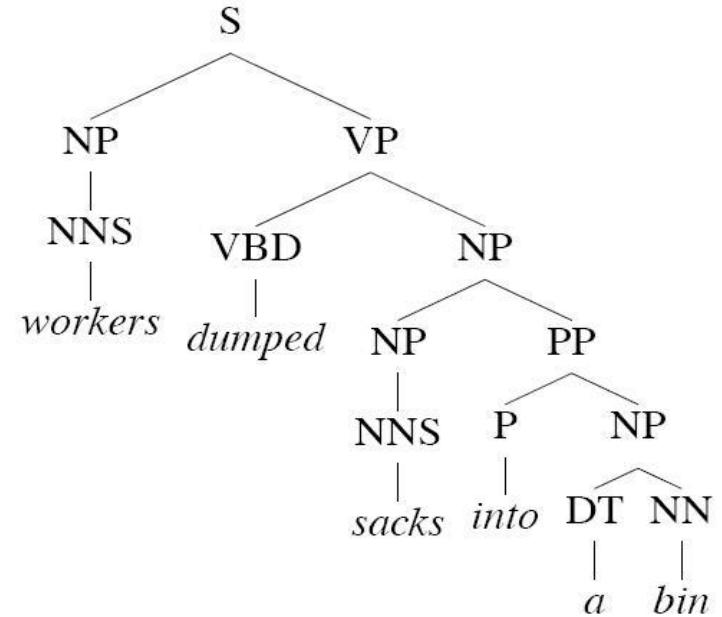
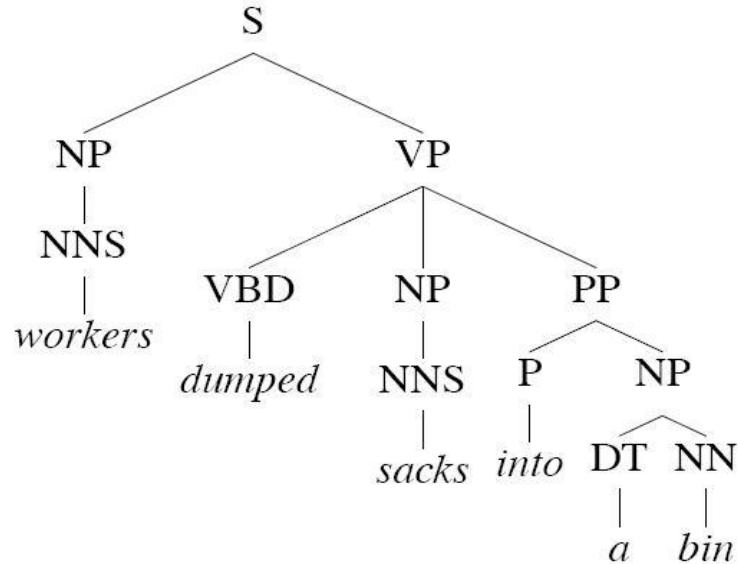
## Flaws I: Structural independence

- 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

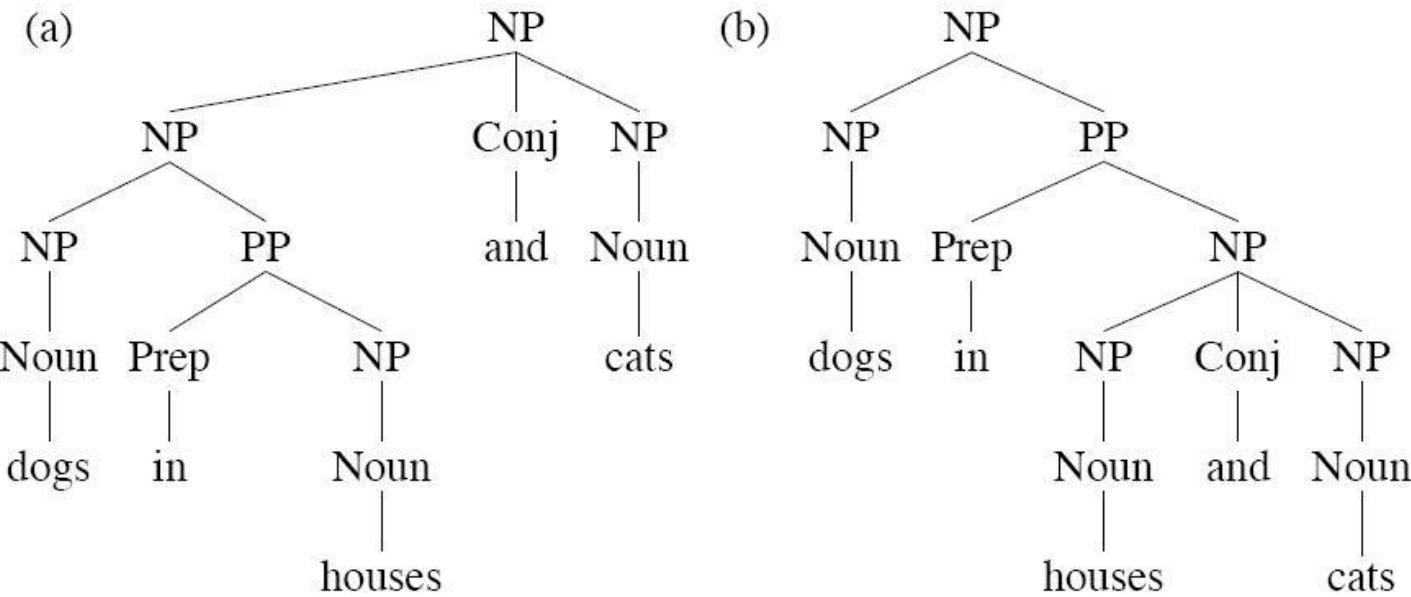
## Flaws II: lexical independence

---

- 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:
    - Workers dumped [sacks into a bin]
    - Workers dumped [sacks] [into a bin] (preferred parse)
  - coordination ambiguity:
    - [dogs in houses] and [cats]
    - [dogs] [in houses and cats]



# Conjunction attachment

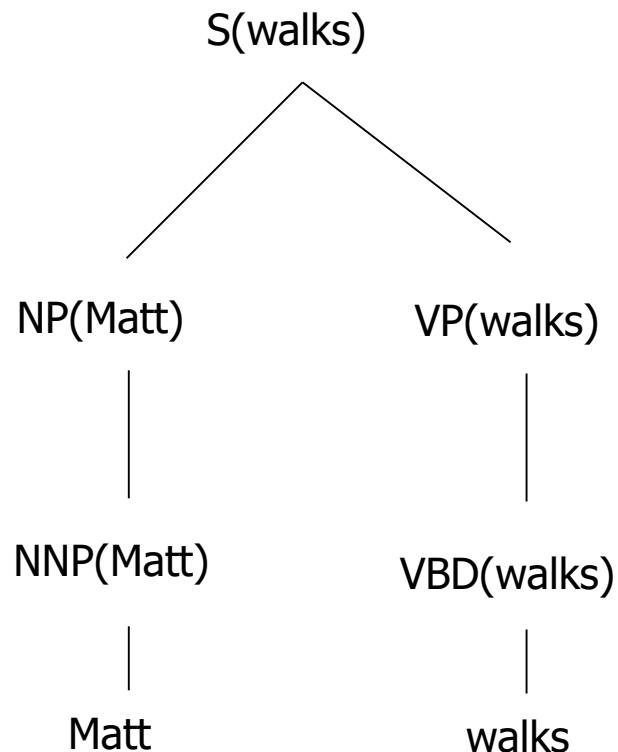


# Lexicalised PCFGs

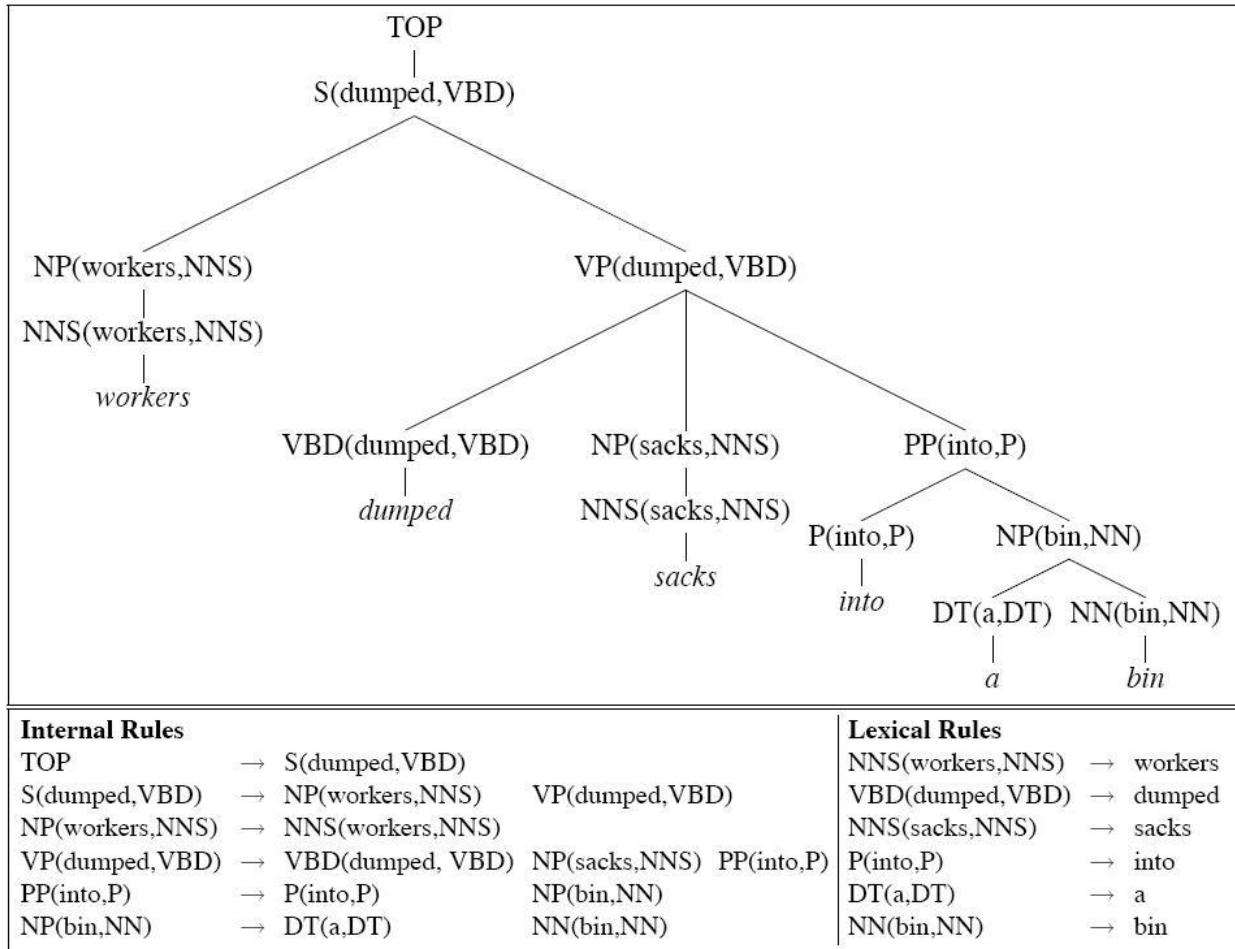
- Attempt to weaken the lexical independence assumption.
- Most common technique:
  - mark each phrasal head (N,V, etc) with the lexical material
  - this is based on the idea that the most crucial lexical dependencies are between head and dependent
  - E.g.: Charniak 1997, Collins 1999

# Lexicalised PCFGs: *Matt walks*

- Makes probabilities partly dependent on lexical content.
- $P(VP \rightarrow VBD | VP)$  becomes:  
 $P(VP \rightarrow VBD | VP, h(VP)=\text{walks})$
- NB: normally, we can't assume that all heads of a phrase of category C are equally probable.



# Lexical attachment



# Practical problems for lexicalised PCFGs

---

- Data sparseness: we don't necessarily see all heads of all phrasal categories often enough in the training data
- Flawed assumptions: lexical dependencies occur elsewhere, not just between head and complement
  - *I got the easier problem of the two to solve*
  - *of the two* and *to solve* are very likely because of the prehead modifier *easier*

# Evaluating Parsers

- We need a measure to evaluate parser performance against gold standard
  - ratio of fully correct sentences parses too coarse
  - ratio of correct constituents

- Does *correct* mean *precision*?

$$\text{precision} = \frac{\text{count}(\text{matching constituents})}{\text{count}(\text{predicted constituents})}$$

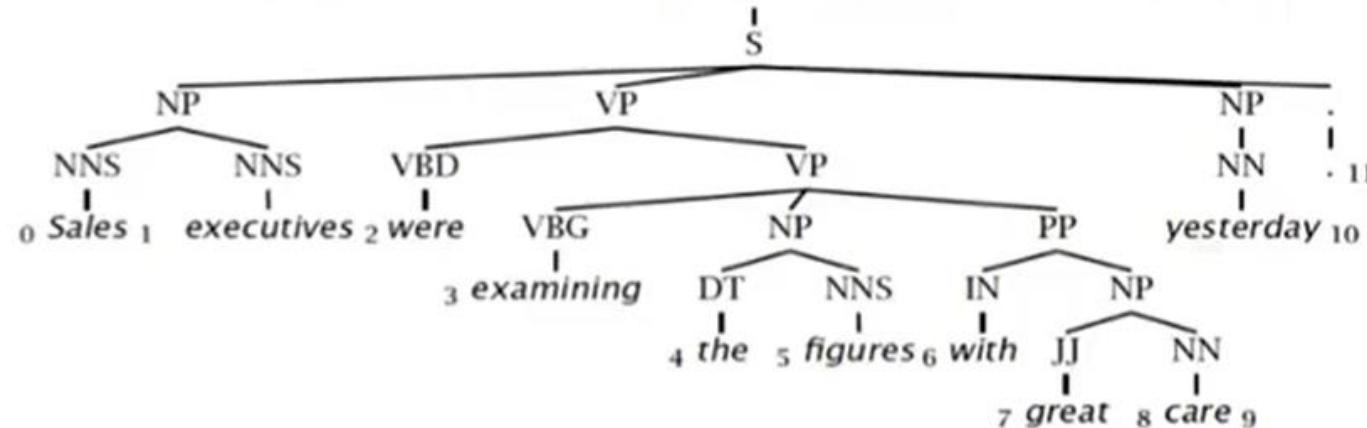
- Does *correct* mean *recall*?

$$\text{Recall} = \frac{\text{count}(\text{matching constituents})}{\text{count}(\text{gold standard constituents})}$$

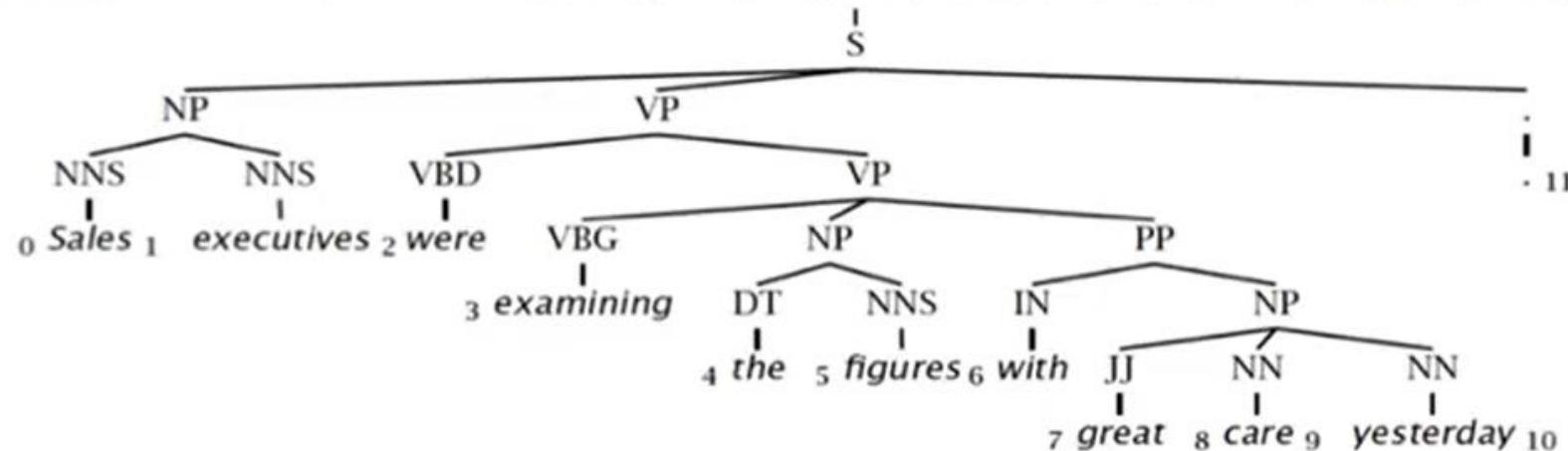
Credits: Philipp Koehn

# Evaluating Parsers

Gold standard brackets: **S-(0:11), NP-(0:2), VP-(2:9), VP-(3:9), NP-(4:6), PP-(6:9), NP-(7,9), NP-(9:10)**



Candidate brackets: **S-(0:11), NP-(0:2), VP-(2:10), VP-(3:10), NP-(4:6), PP-(6-10), NP-(7,10)**



# Evaluating Parsers

---

• **Gold standard brackets: 8**

S-(0:11), NP-(0:2), VP-(2:9), VP-(3:9), **NP-(4:6)**, PP-(6-9), NP-(7,9), NP-(9:10)

**Candidate brackets: 7**

S-(0:11), NP-(0:2), VP-(2:10), VP-(3:10), **NP-(4:6)**, PP-(6-10), NP-(7,10)

Labeled Precision               $3/7 = 42.9\%$

Labeled Recall                   $3/8 = 37.5\%$

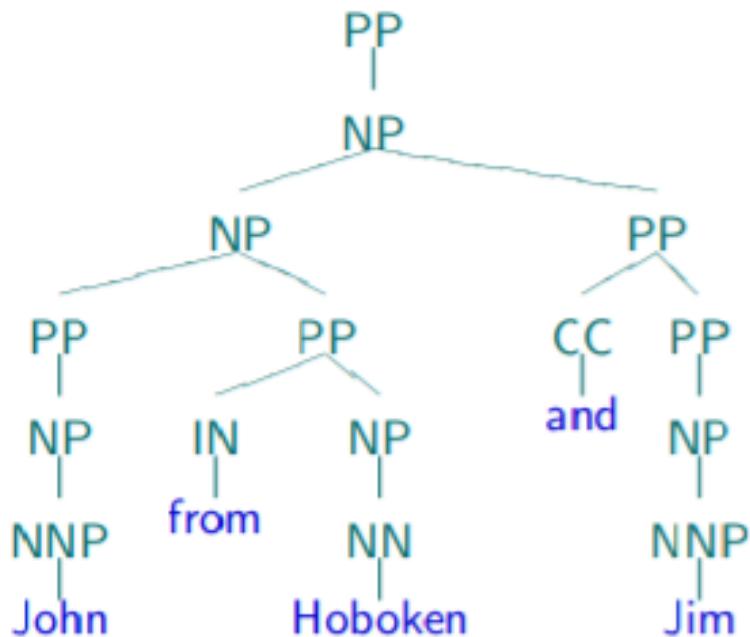
LP/LR F1                        40.0%

Tagging Accuracy                 $11/11 = 100.0\%$

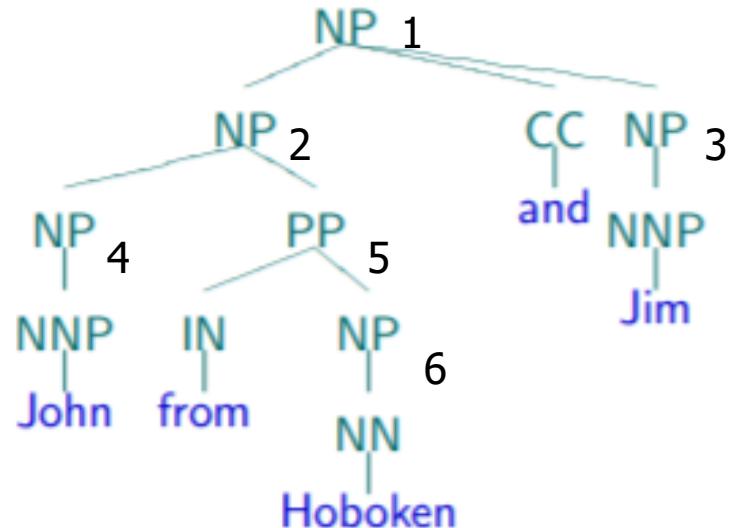
# Evaluating Parsers

## Low Precision, High Recall

system



gold standard



all gold standard constituents are predicted (recall 6/6)  
 ... but we are predicting many more (precision 6/10)

# Evaluating Parsers

- F-measure: balance of precision and recall

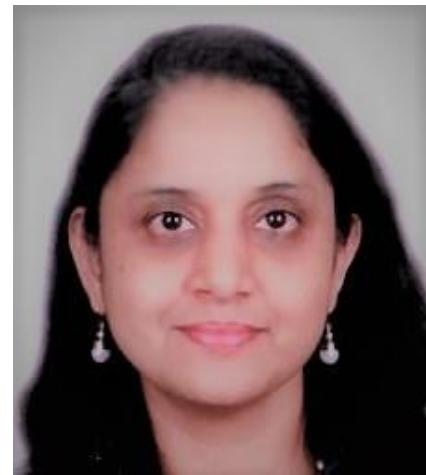
$$F_1 = \frac{\text{precision} \times \text{recall}}{(\text{precision} + \text{recall})/2}$$

- F-measure is used in many other NLP tasks and may be adjusted to give more emphasis to either precision or recall

Credits: Philipp Koehn

# Extra Reading

- <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>
- [https://www.tutorialspoint.com/natural\\_language\\_toolkit/natural\\_language\\_toolkit\\_parsing.htm](https://www.tutorialspoint.com/natural_language_toolkit/natural_language_toolkit_parsing.htm)
- <https://lost-contact.mit.edu/afs/cs.pitt.edu/projects/nltk/docs/tutorial/pcfg/nochunks.html>
- [http://courses.washington.edu/ling571/ling571\\_WIN2017/slides/ling571\\_class\\_6\\_pcfg\\_impr\\_flat.pdf](http://courses.washington.edu/ling571/ling571_WIN2017/slides/ling571_class_6_pcfg_impr_flat.pdf)
- <http://www.cs.columbia.edu/~mcollins/courses/nlp2011/notes/lexpcfgs.pdf>



**Thank you for your time!!**



# Natural Language Processing

## DSECL ZG565

Dr. Chetana Gavankar, Ph.D,

IIT Bombay-Monash University Australia

Associate Professor, BITS Pilani

[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



**BITS** Pilani  
Pilani Campus



## **Session 7 - Dependency Parsing**

### **Date – 26<sup>th</sup> August 2023**

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Jurafsky and Prof. Martin, Prof. Pawan Goyal and many others who made their course materials freely available online.

# Session Content



(Ref: Chapter 15 Jurafsky and Martin)

---

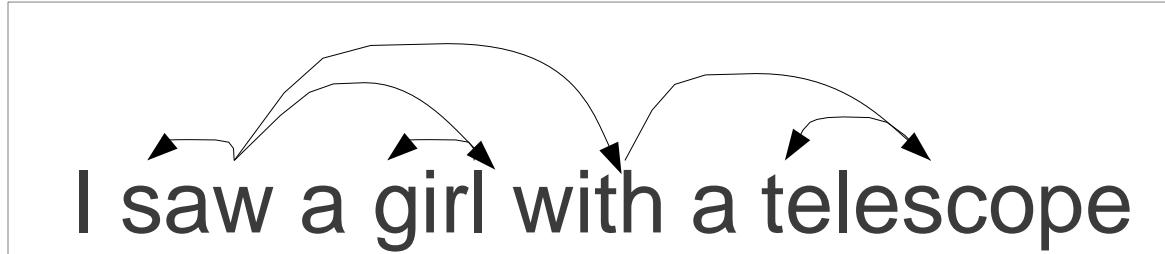
- Motivation
- Dependency structure and Dependency grammar
- Dependency Relation
- Universal Dependencies
- Method of Dependency Parsing
- Graph based dependency Parsing
- Evaluation

# Ambiguity is Explosive

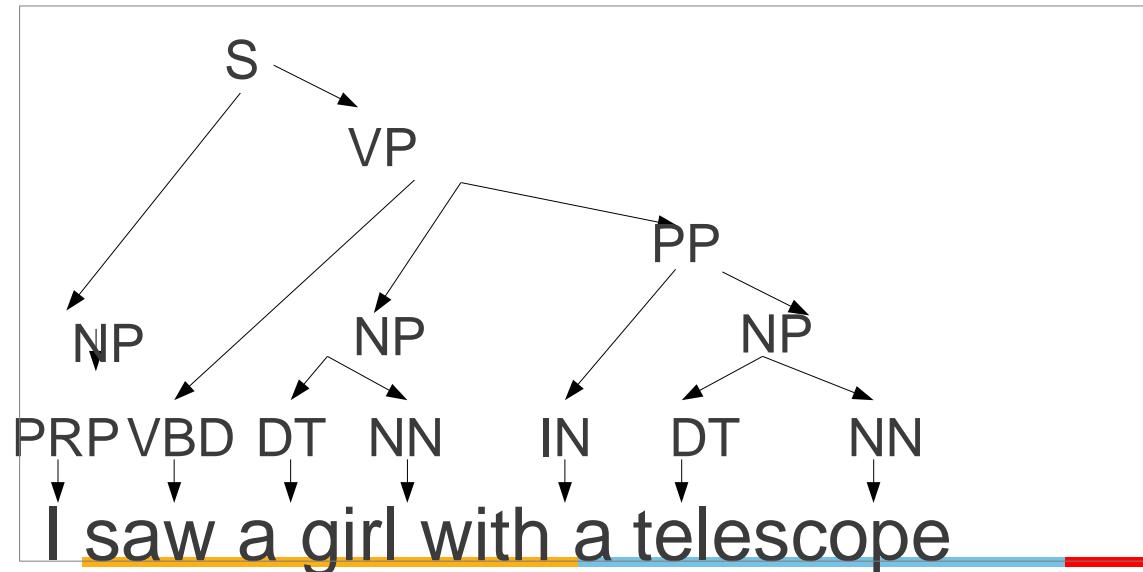
- “I saw the man with the telescope”: 2 parses
- “I saw the man on the hill with the telescope.”: 5 parses
- “I saw the man on the hill in Texas with the telescope”: 14 parses
- “I saw the man on the hill in Texas with the telescope at noon.”: 42 parses
- “I saw the man on the hill in Texas with the telescope at noon on Monday” 132 parses

# Two Types of Parsing

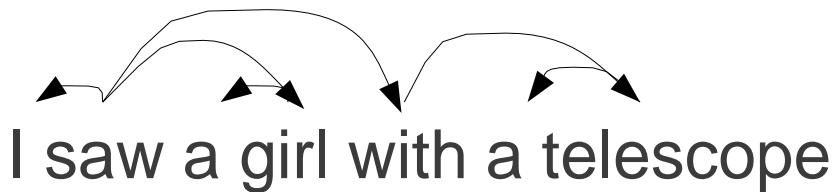
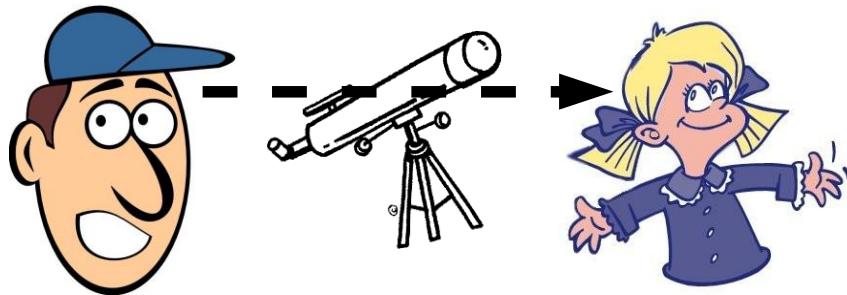
- **Dependency:** focuses on relations between words



- **Phrase structure:** focuses on identifying phrases and their recursive structure

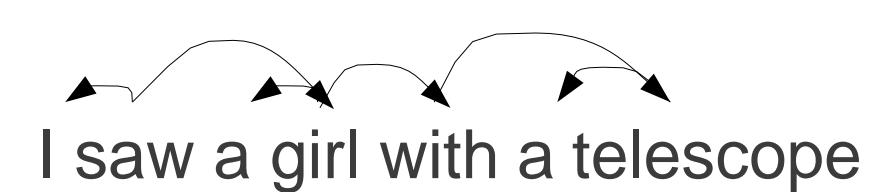


# Dependencies Also Resolve Ambiguity



A diagram showing a sentence "I saw a girl with a telescope" enclosed in a light gray box. Below the sentence are four curved arrows pointing from the words "I", "saw", "a", and "with" to the word "telescope".

I saw a girl with a telescope



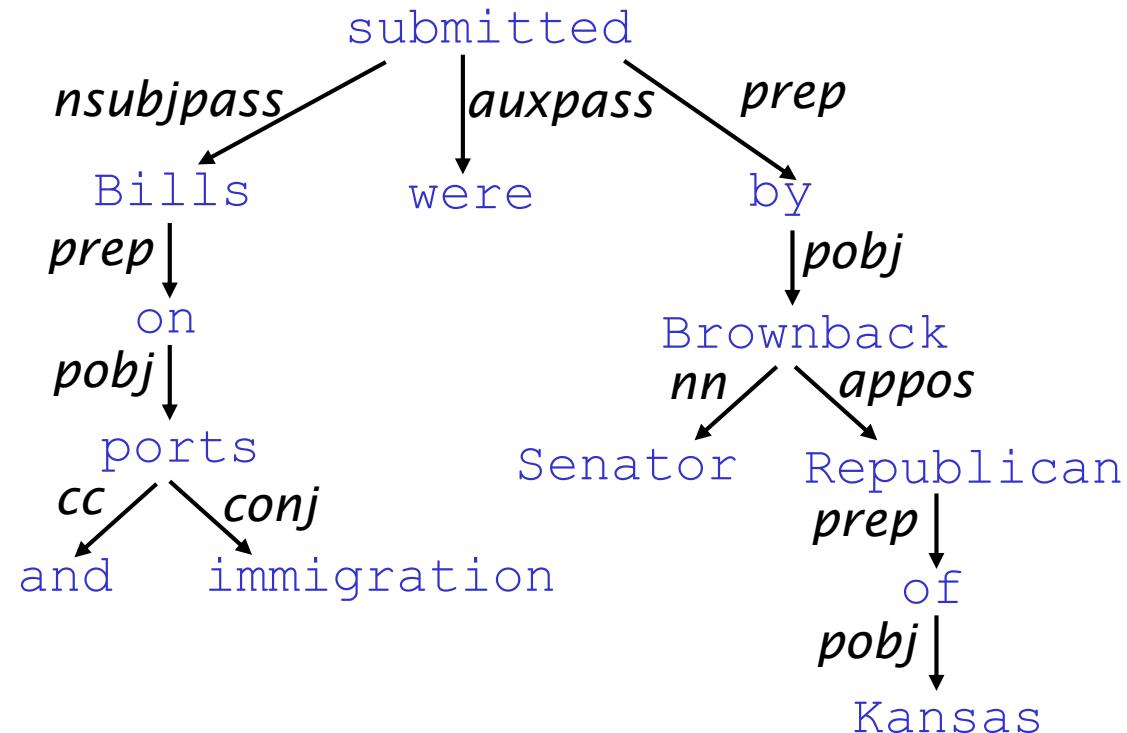
A diagram showing the same sentence "I saw a girl with a telescope" enclosed in a light gray box. Below the sentence are four curved arrows pointing from the words "I", "saw", "a", and "with" to the word "telescope".

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.)

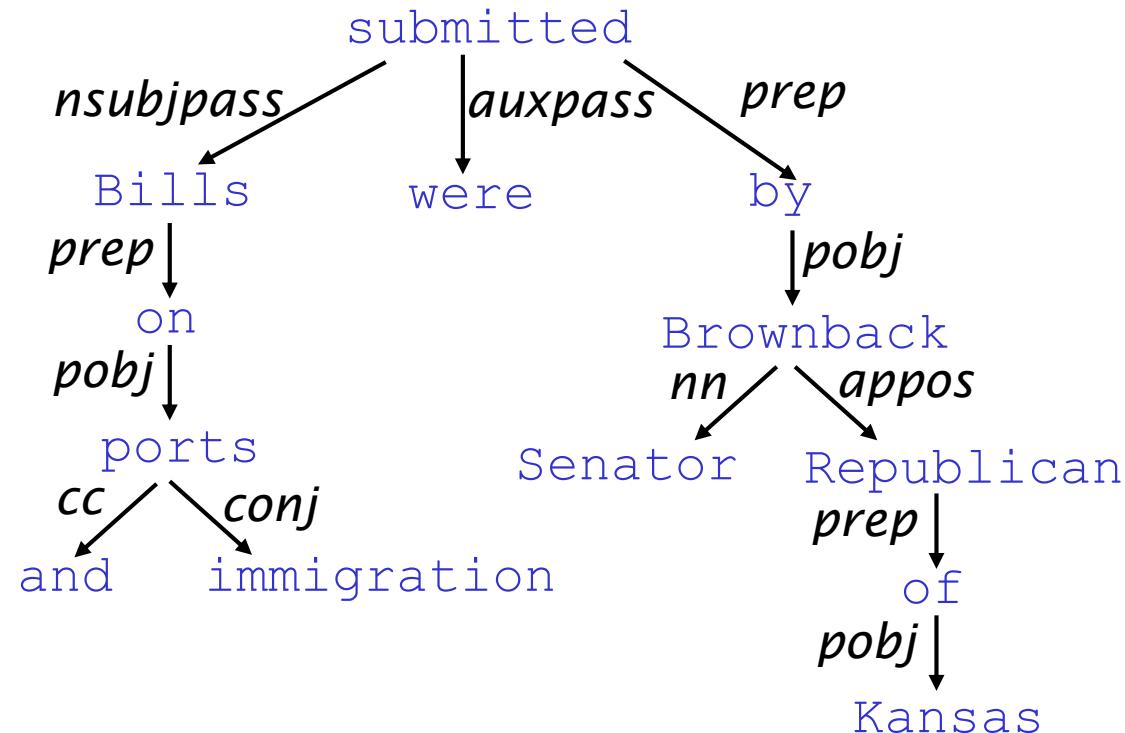


# Dependency Grammar and Dependency Structure

**Dependency syntax postulates that syntactic structure consists of lexical items linked by binary asymmetric relations (“arrows”) called dependencies**

The arrow connects a **head** (governor) with a **dependent** (modifier)

Usually, dependencies form a tree  
(connected, acyclic,  
single-head)



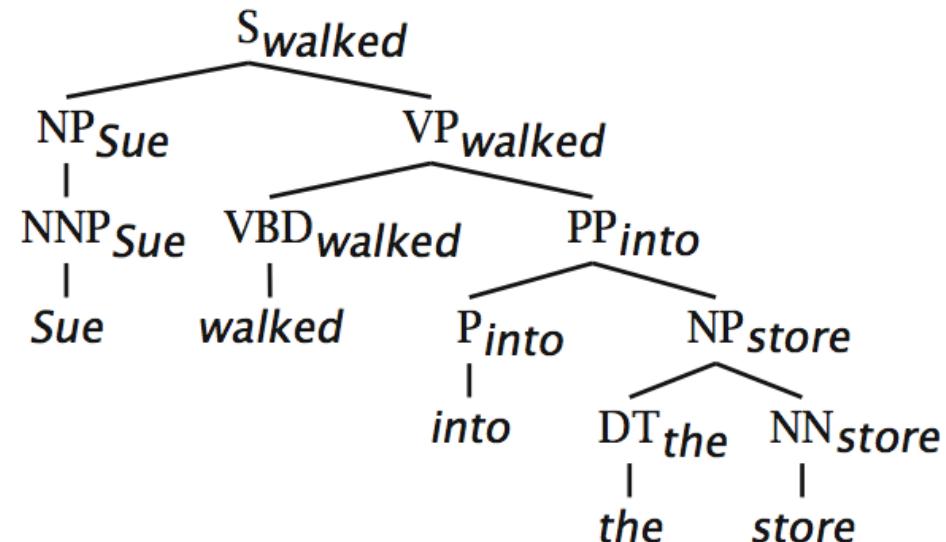
# Relation between phrase structure and dependency structure

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")

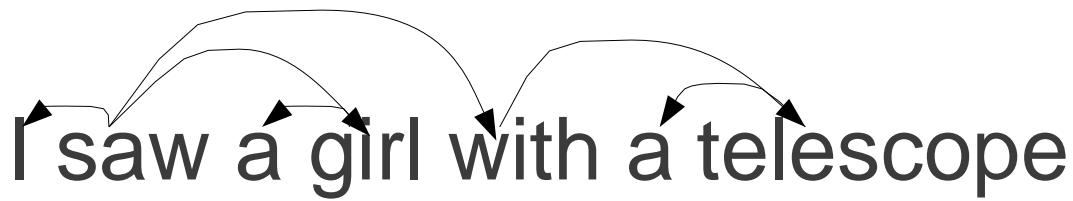


# Dependency graph

- 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  $d$
  - ▶  $w_i \xrightarrow{d} w_j \Leftrightarrow (w_i, d, w_j) \in A$
  - ▶  $i \rightarrow j \equiv (i, j) \in A$
  - ▶  $i \rightarrow^* j \equiv i = j \vee \exists k : i \rightarrow k, k \rightarrow^* j$

# 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  $i$ .



I saw a girl with a telescope

# Universal dependencies

---

**<http://universaldependencies.org/>**

- Annotated treebanks in many languages
- Uniform annotation scheme across all languages:
  - Universal POS tags
  - Universal dependency relations

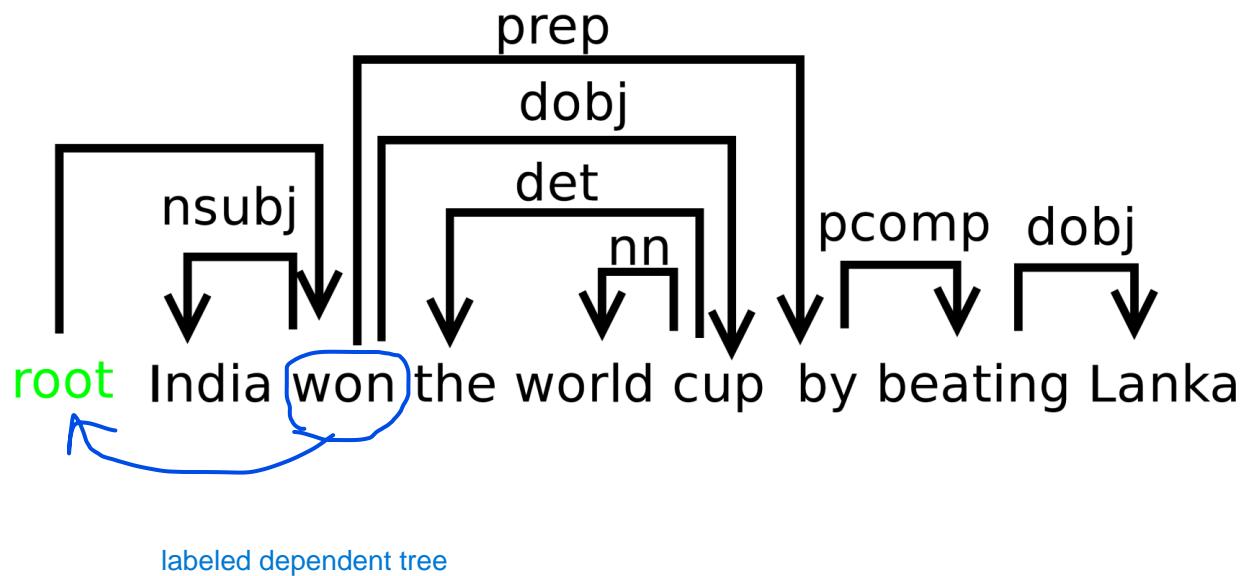
# Dependency Relations

| <b>Argument Dependencies</b> | <b>Description</b>    |
|------------------------------|-----------------------|
| <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 |

| <b>Modifier Dependencies</b> | <b>Description</b>     |
|------------------------------|------------------------|
| <b>tmod</b>                  | temporal modifier      |
| <b>appos</b>                 | appositional modifier  |
| <b>det</b>                   | determiner             |
| <b>prep</b>                  | prepositional modifier |

# Example Dependency Parse



# 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

## 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

## 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

A transition system for dependency parsing is a quadruple  $S = (C, T, c_s, C_t)$ , where

- $C$  is a set of configurations,
- $T$  is a set of transitions, such that  $t : C \rightarrow C$ ,
- $c_s$  is an initialization function
- $C_t \subseteq C$  is a set of terminal configurations.

A transition sequence for a sentence  $x$  is a set of configurations

$C_{0,m} = (c_0, c_1, \dots, c_m)$  such that

$c_0 = c_s(x)$ ,  $c_m \in C_t$ ,  $c_i = t(c_{i-1})$  for some  $t \in T$

**Initialization:**  $([], [w_1, \dots, w_n]_B, \{\})$

**Termination:**  $(S, [], A)$

# 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  $\frac{([\dots]_S, [w_i, \dots]_B, A)}{([\dots, w_i]_S, [\dots]_B, A)}$

# Arc Eager Parsing example

## Transitions:

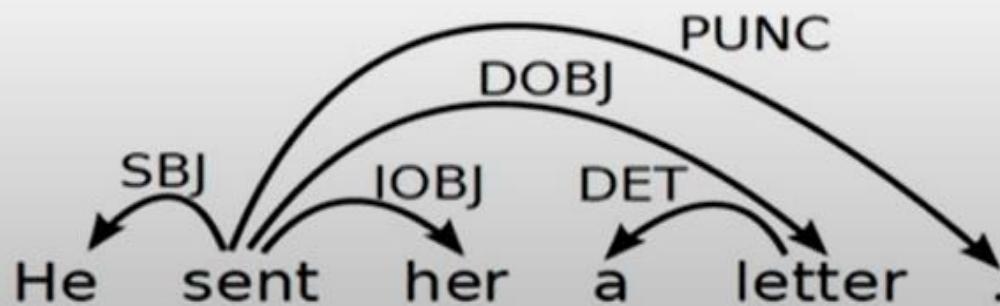
**Stack**

He [s

**Buffer**

[He, sent, her, a, letter, .]B

**Arcs**



# Arc Eager Parsing example

**Transitions:** SH-LA

**Stack**

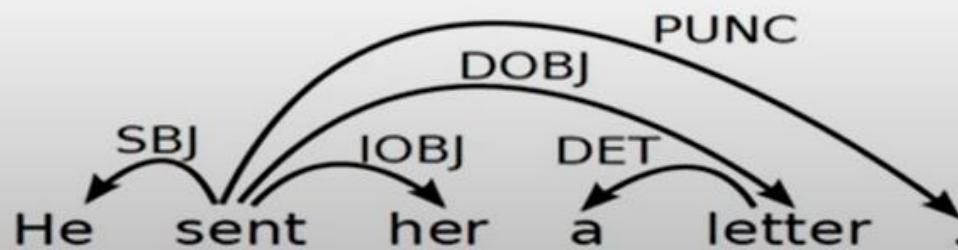
[ ]s

**Buffer**

[sent, her, a, letter, .]B

**Arcs**

He  $\xleftarrow{\text{SBJ}}$  sent



# Arc Eager Parsing example

**Transitions:** SH-LA-SH

## Stack

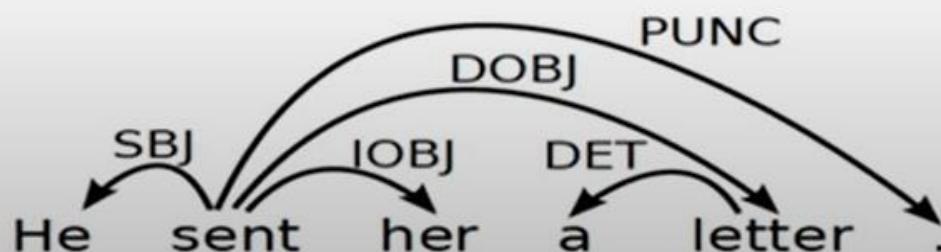
[sent]<sub>S</sub>

## Buffer

[her, a, letter, .]<sub>B</sub>

## Arcs

He  $\xleftarrow{\text{SBJ}}$  sent



# Arc Eager Parsing example

**Transitions:** SH-LA-SH-RA

## Stack

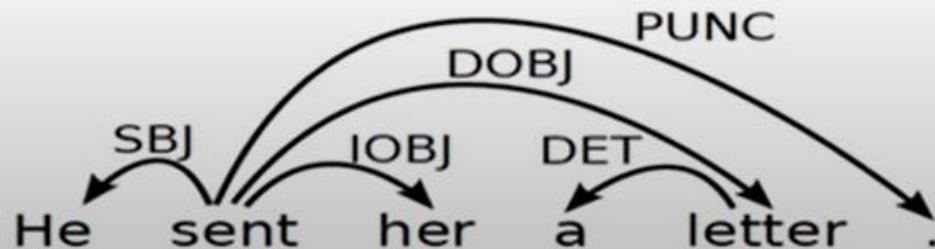
[sent, her]<sub>S</sub>

## Buffer

[a, letter, .]<sub>B</sub>

## Arcs

He  $\xleftarrow{\text{SBJ}}$  sent  
 sent  $\xrightarrow{\text{IOBJ}}$  her



# Arc Eager Parsing example

**Transitions:** SH-LA-SH-RA-SH

**Stack**

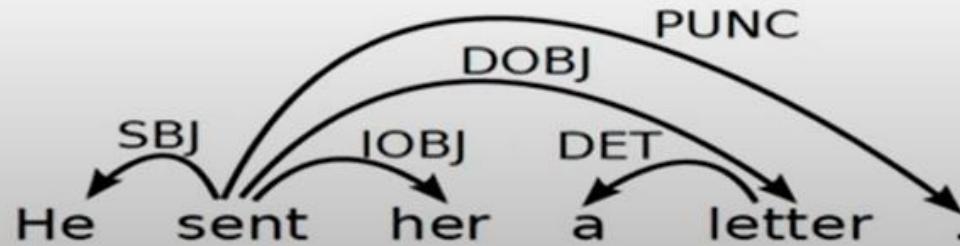
[sent, her, a]<sub>S</sub>

**Buffer**

[letter, .]<sub>B</sub>

**Arcs**

He  $\xleftarrow{\text{SBJ}}$  sent  
sent  $\xrightarrow{\text{IOBJ}}$  her



# Arc Eager Parsing example

**Transitions:** SH-LA-SH-RA-SH-LA

## Stack

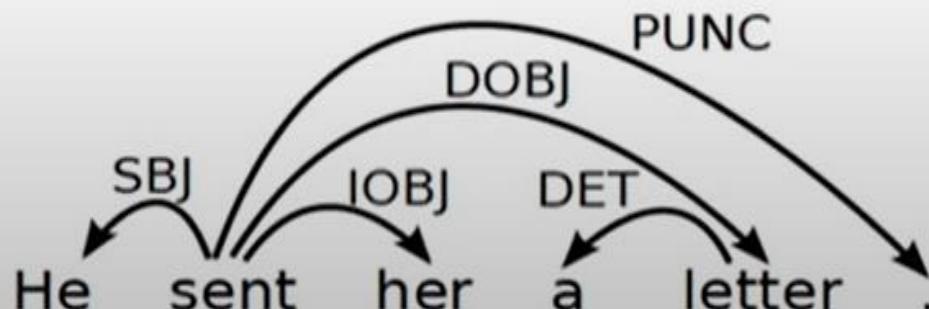
[sent, her]<sub>S</sub>

## Buffer

[letter, .]<sub>B</sub>

## Arcs

He  $\xleftarrow{\text{SBJ}}$  sent  
 sent  $\xrightarrow{\text{IOBJ}}$  her  
 a  $\xleftarrow{\text{DET}}$  letter



# Arc Eager Parsing example

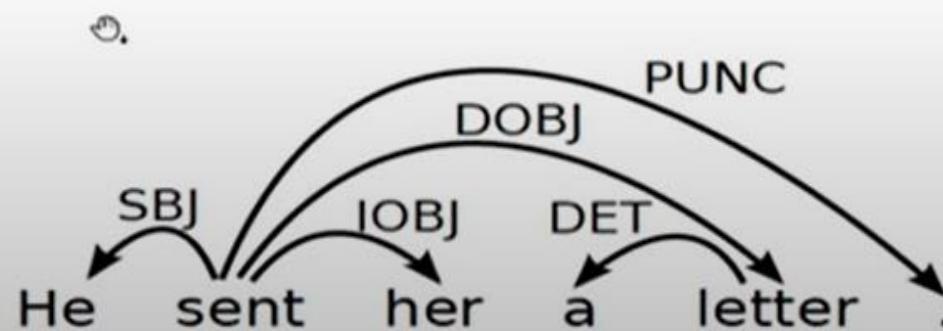
**Transitions:** SH-LA-SH-RA-SH-LA-RE

## Stack

[sent]<sub>S</sub>

## Buffer

[letter, .]<sub>B</sub>

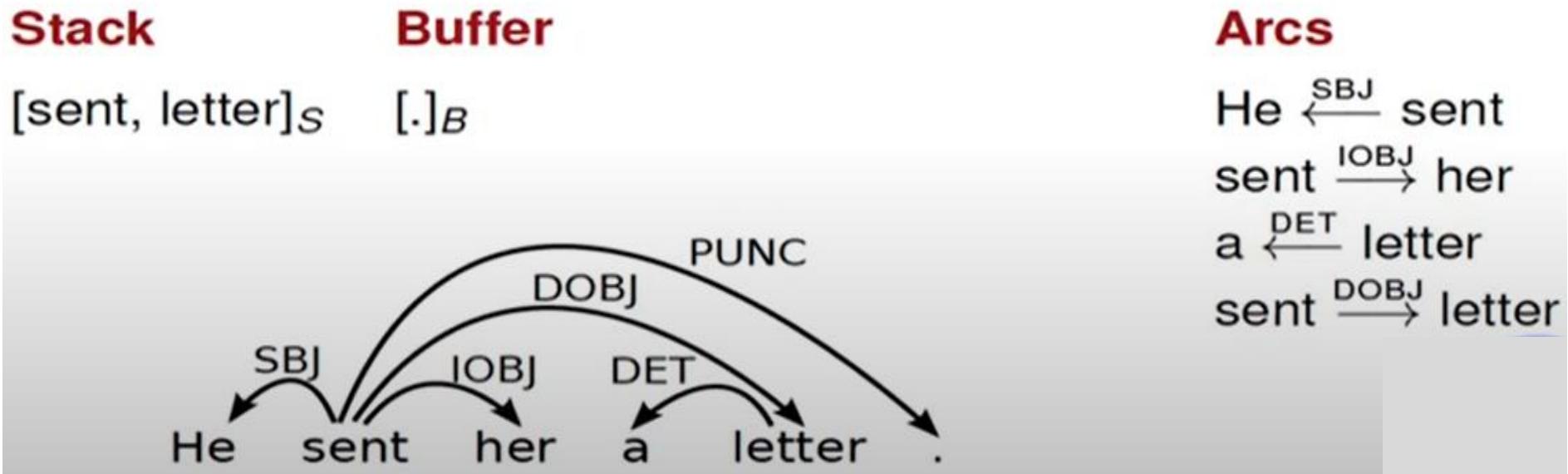


## Arcs

He  $\xleftarrow{\text{SBJ}}$  sent  
 sent  $\xrightarrow{\text{IOBJ}}$  her  
 a  $\xleftarrow{\text{DET}}$  letter

# Arc Eager Parsing example

**Transitions:** SH-LA-SH-RA-SH-LA-RE-RA



# Arc Eager Parsing example

**Transitions:** SH-LA-SH-RA-SH-LA-RE-RA-RE

## Stack

$[\text{sent}]_S$

## Buffer

$[.]_B$



## Arcs

$\text{He} \xleftarrow{\text{SBJ}} \text{sent}$   
 $\text{sent} \xrightarrow{\text{IOBJ}} \text{her}$   
 $\text{a} \xleftarrow{\text{DET}} \text{letter}$   
 $\text{sent} \xrightarrow{\text{DOBJ}} \text{letter}$

# Arc Eager Parsing example

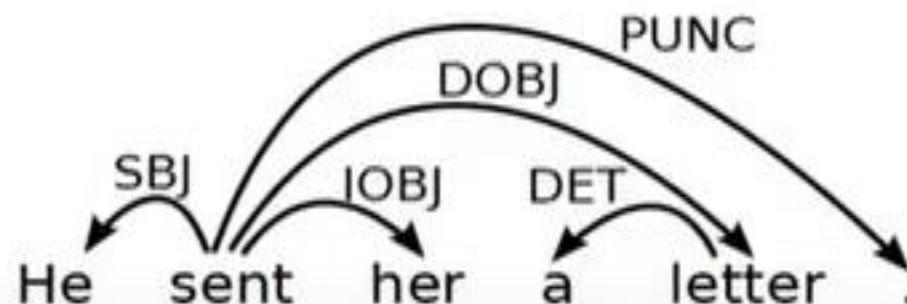
**Transitions:** SH-LA-SH-RA-SH-LA-RE-RA-RE-RA

## Stack

[sent, .]s

## Buffer

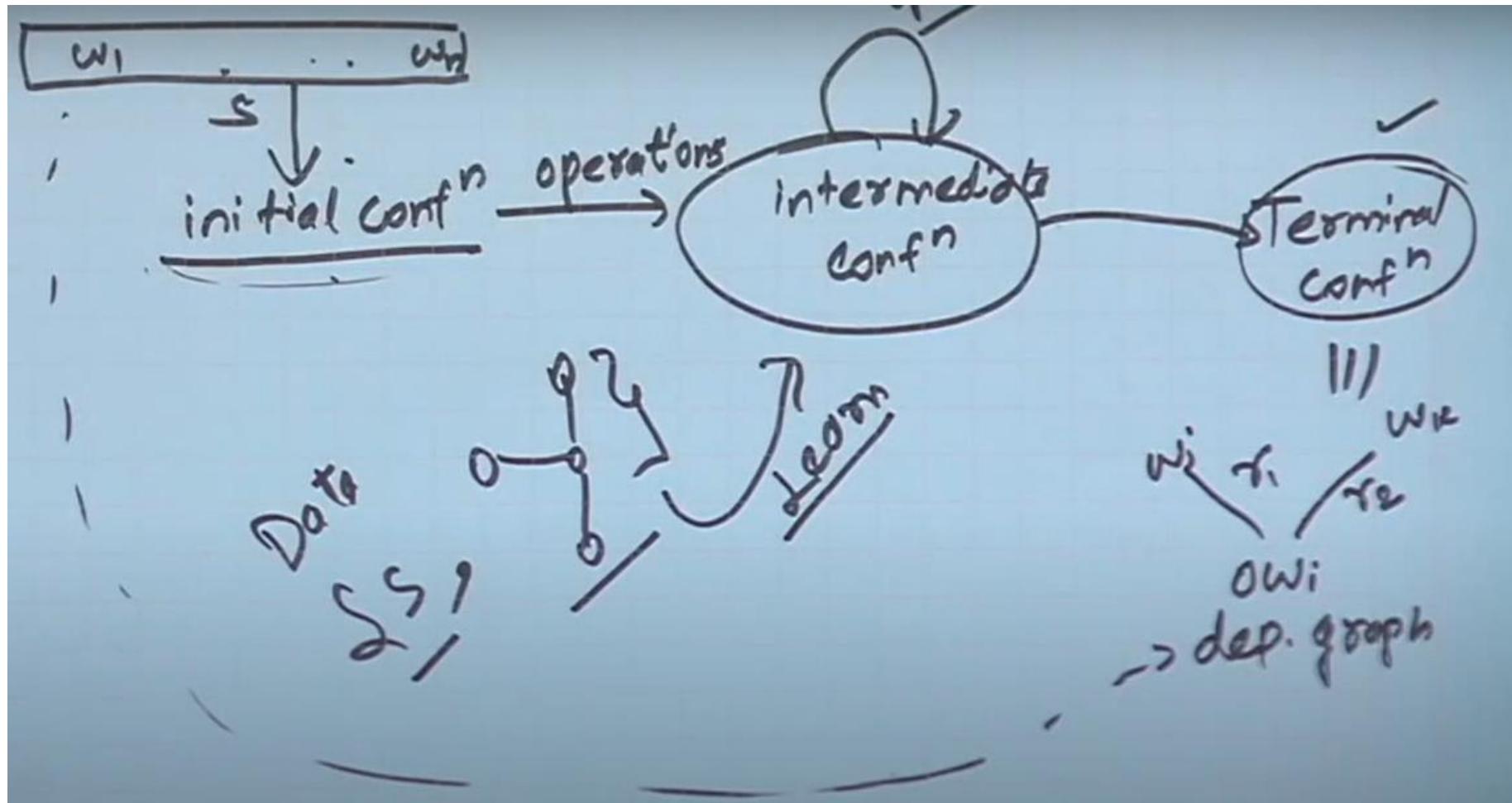
[ ]<sub>B</sub>



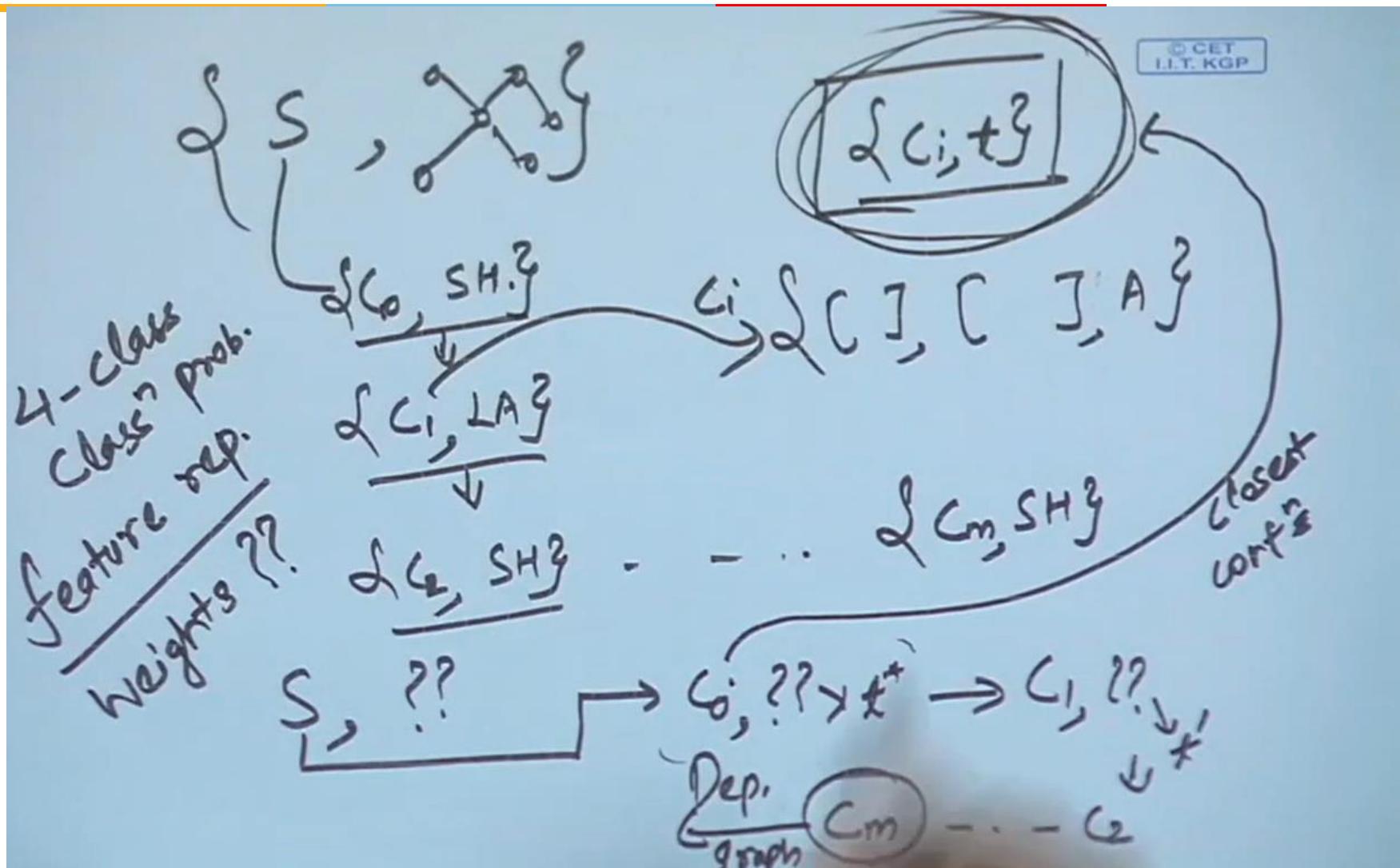
## Arcs

He  $\xleftarrow{\text{SBJ}}$  sent  
 sent  $\xrightarrow{\text{IOBJ}}$  her  
 a  $\xleftarrow{\text{DET}}$  letter  
 sent  $\xrightarrow{\text{DOBJ}}$  letter  
 sent  $\xrightarrow{\text{PUNC}}$

# Learning weights



# Deriving dependency graph for a new sentence



# Classifier for learning the transmission

## *Data-driven deterministic parsing:*

- Deterministic parsing requires an **oracle**.
- An oracle can be approximated by a **classifier**.
- A classifier can be trained using **treebank** data.

## *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?

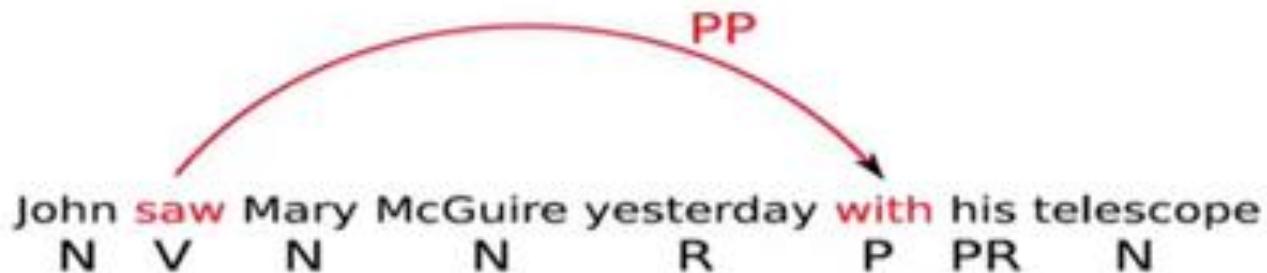
# Feature Models

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 Models



## 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

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  $t_i(c_i) = c_{i+1}$ .

# 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.

# Example

$F(c,t) = [(c_0, LA), (c_1, LA), (c_2, LA) | (c_0, RA), (c_1, RA), (c_2, RA) | (c_0, RE), (c_1, RE), (c_2, RE) | (c_0, SH), (c_1, SH), (c_2, SH)]$

$W = [5.5, 5.5, 5.5 | 5.0, 5.0, 5.0 | 5.0, 5.0, 5.0 | \dots]$

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

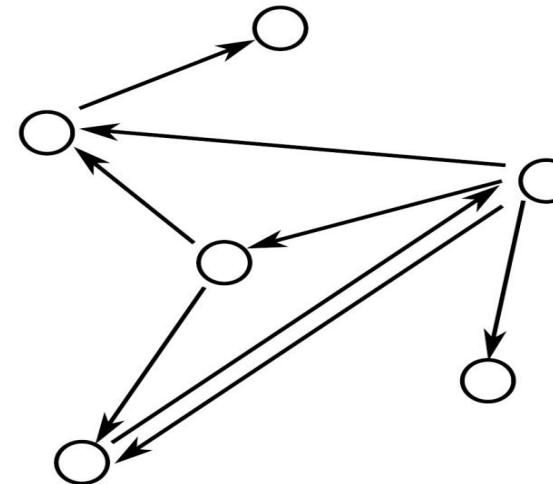
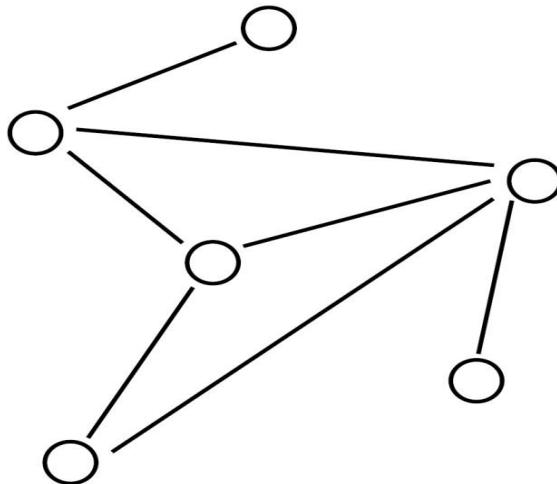
$W = W + f(c, t^0) - f(c, t^*)$

$W = [5.5, 5.0 | 5.5, 5.5, 1, 5.0 | \dots] + [0, 0, 0 | 0, 0, 0 | \dots | 1, 0, 0] - [1, 0, 0 | 0, 0, 0 | \dots]$

New vector =  $[4.5, 5.5, 5.5 | 5.0 | \dots | 6.0, 5.0, 5.0]$

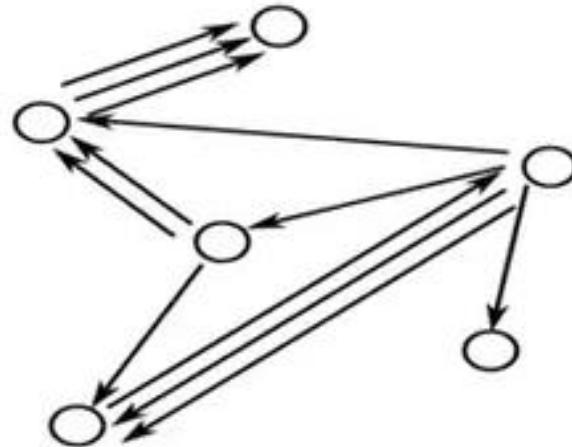
# Graph

- ▶ 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):**  $(i, j) \in A \not\Leftrightarrow (j, i) \in A$



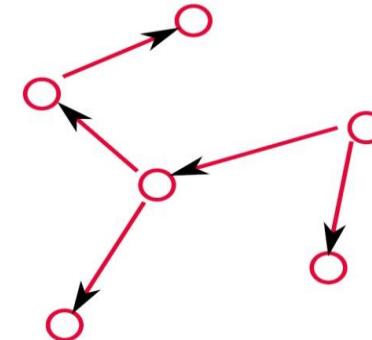
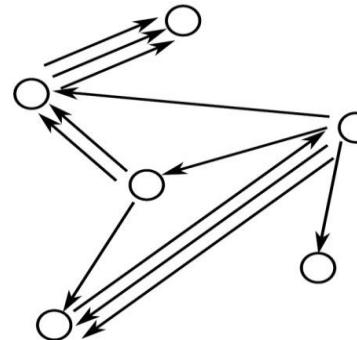
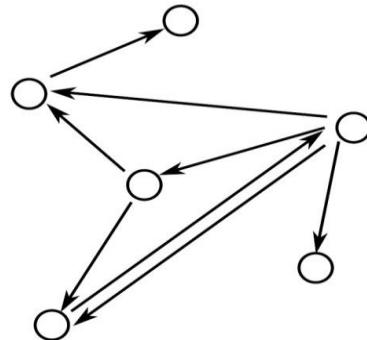
# Multi Digraph

- 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  $j$ .



# Directed 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

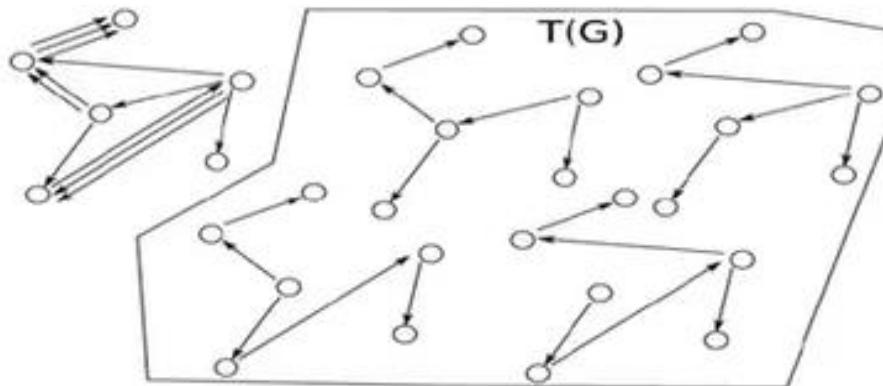
---

- 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

$$V_x = \{x_0 = \text{root}, x_1, \dots, x_n\}$$

$$E_x = \{(i, j) : i \neq j, (i, j) \in [0 : n] \times [1 : n]\}$$

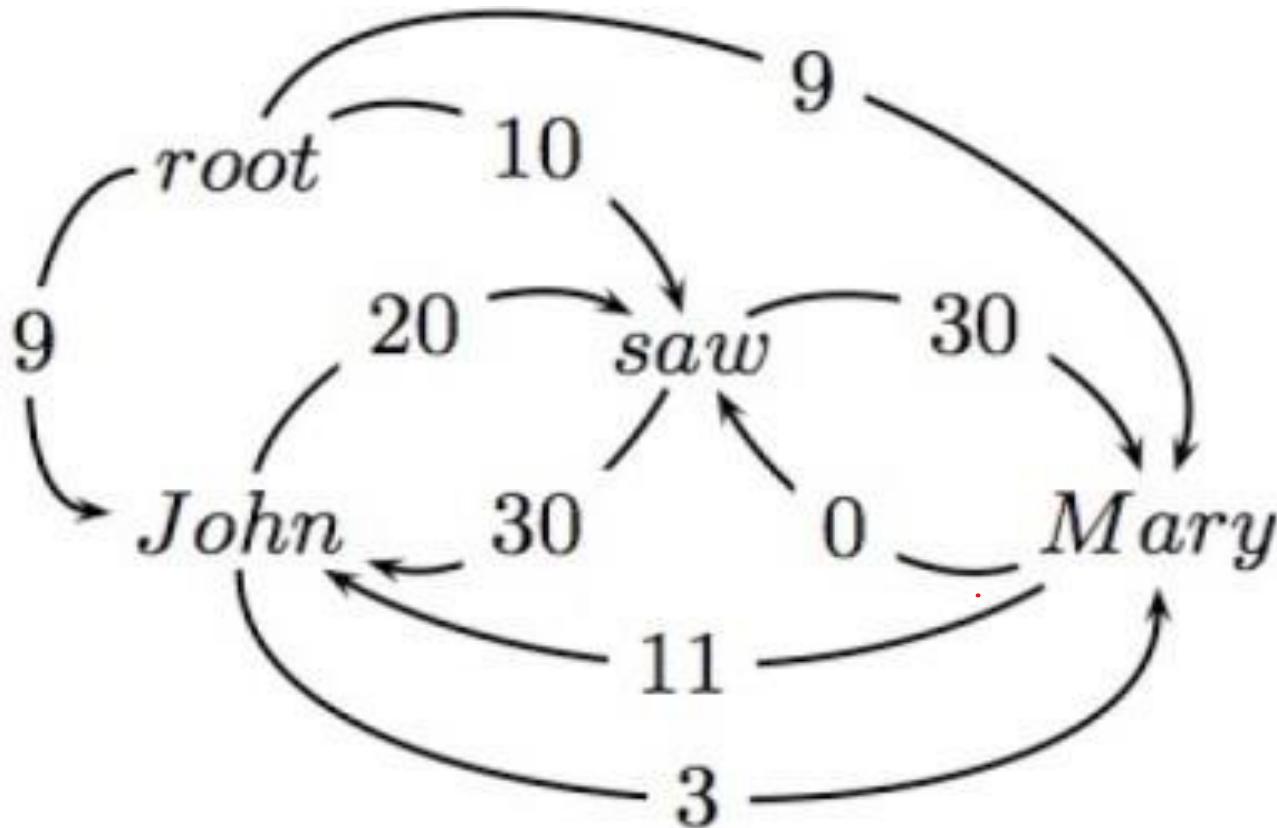
$G_x$  is a graph with

- 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

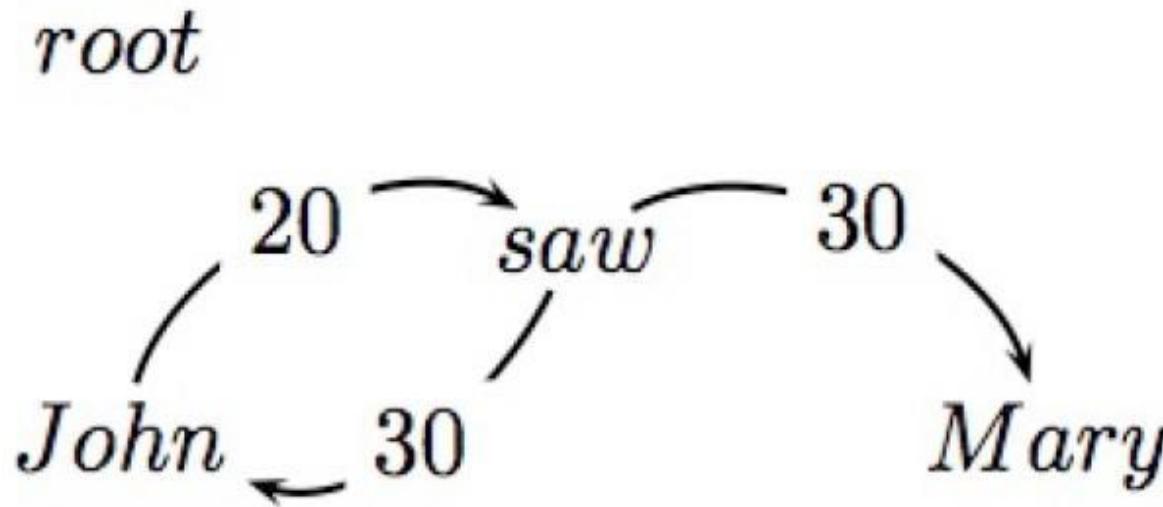
- Each vertex in the graph greedily selects the incoming edge with the highest weight.
- If a tree results, it must be a maximum spanning tree.
- If not, there must be a cycle.
  - Identify the cycle and contract it into a single vertex.
  - Recalculate edge weights going into and out of the cycle.

# Chu-Liu-Edmonds Example



# Chu-Liu-Edmonds Example

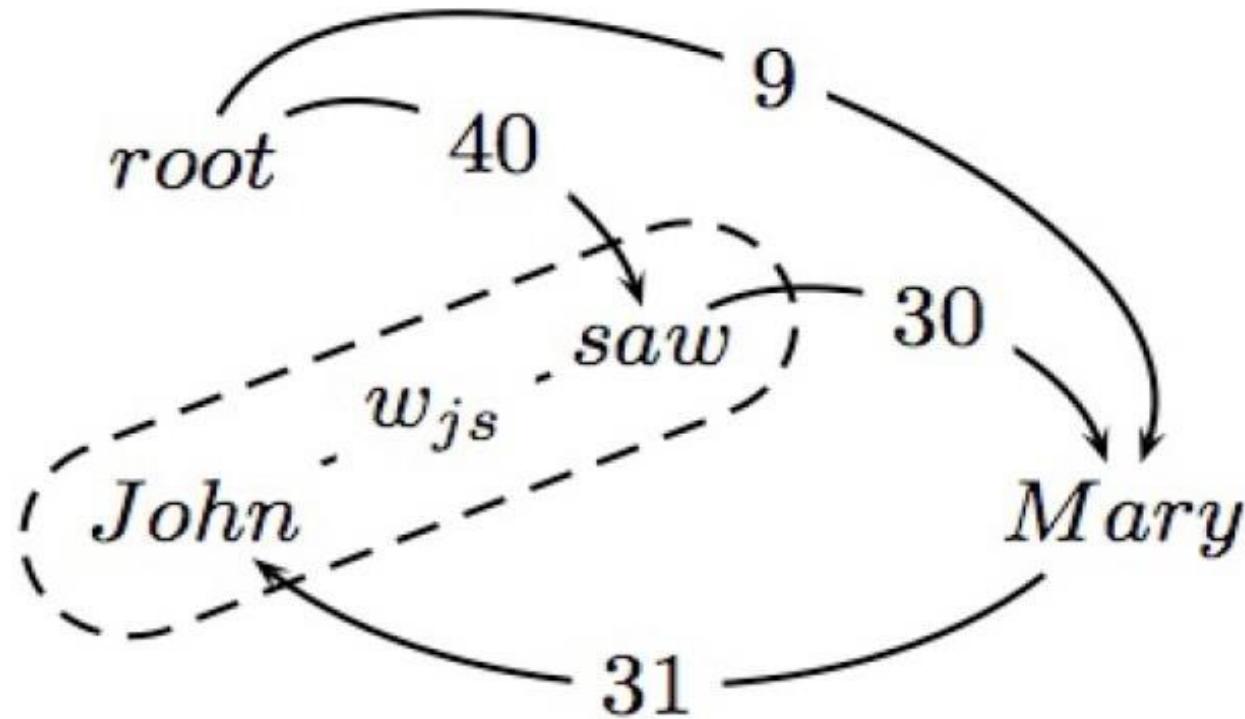
- ▶ Find highest scoring incoming arc for each vertex



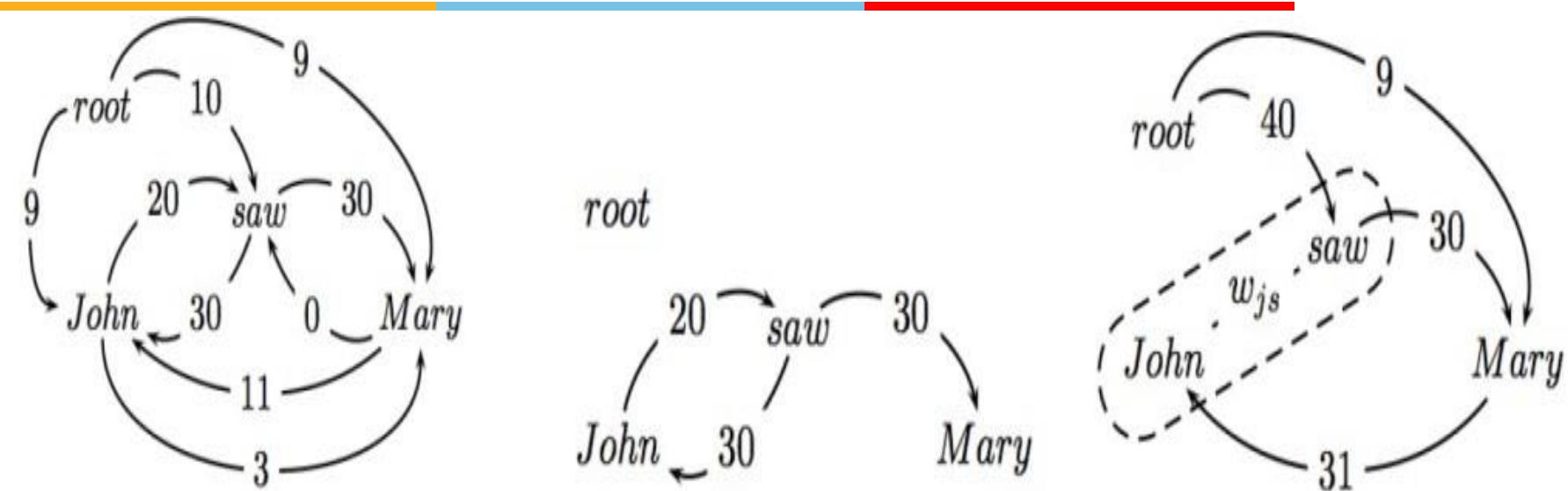
- ▶ If this is a tree, then we have found MST!!

# Chu-Liu-Edmonds Example

- ▶ If not a tree, identify cycle and contract
- ▶ Recalculate arc weights into and out-of cycle

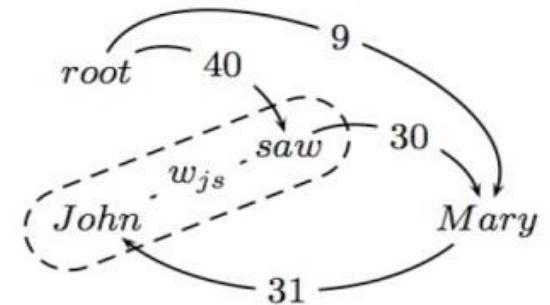
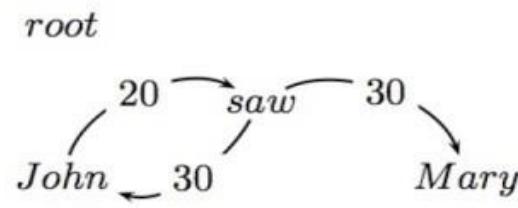
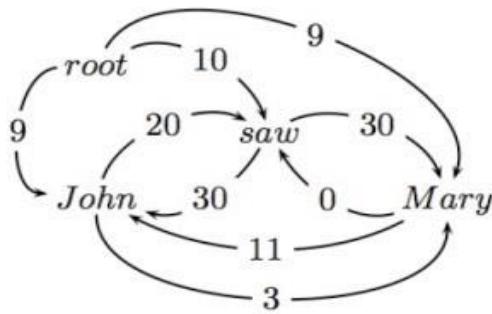


# Chu-Liu-Edmonds Example



- ▶ 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

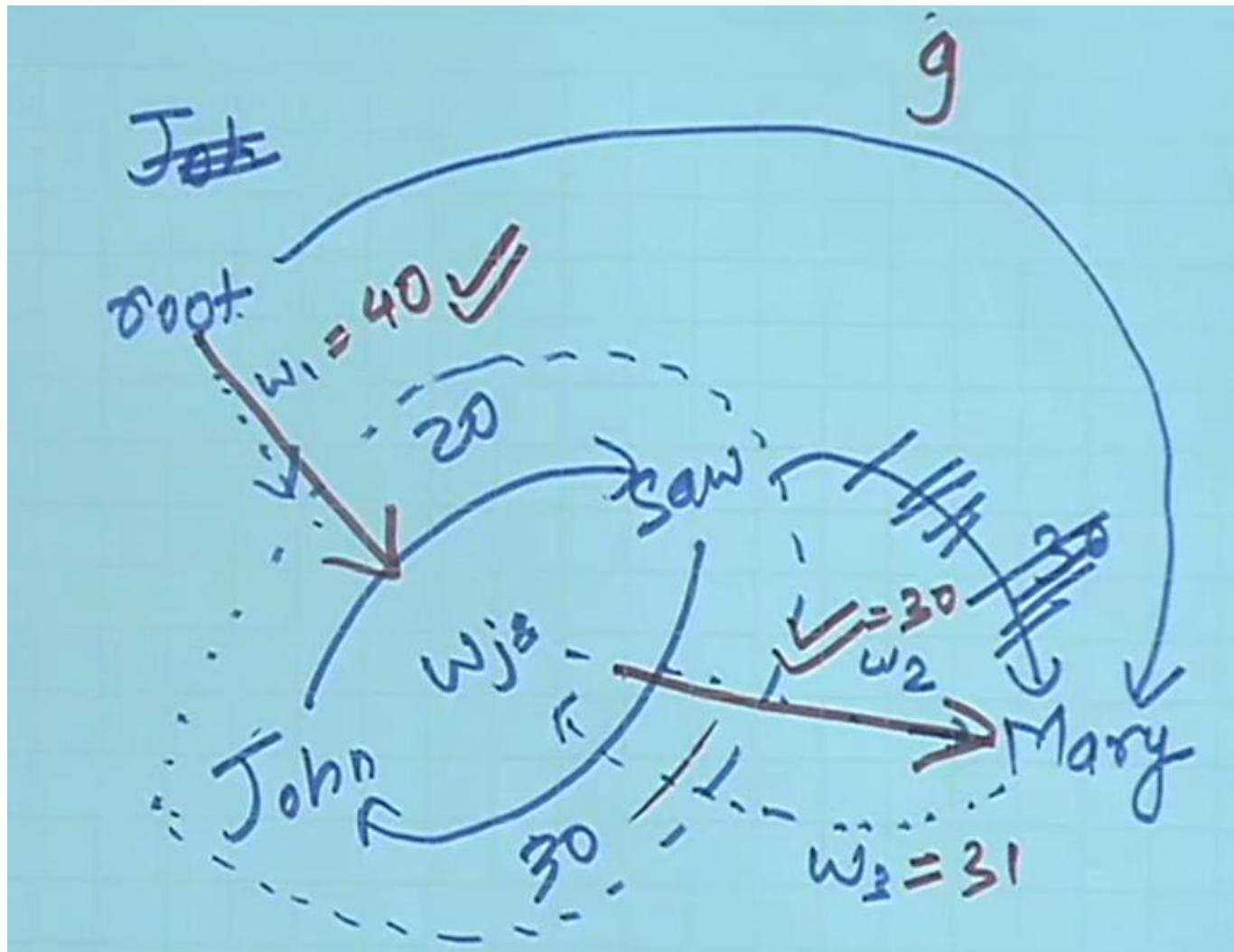
# Chu-Liu-Edmonds Example



## ► Incoming arc weights

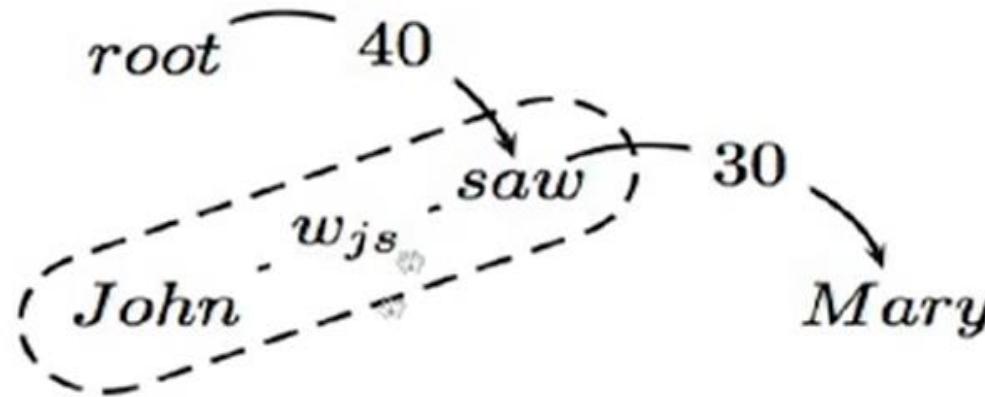
- ▶ Equal to the weight of best spanning tree that includes head of incoming arc, and all nodes in cycle
- ▶  $\text{root} \rightarrow \text{saw} \rightarrow \text{John}$  is 40 (\*\*)
- ▶  $\text{root} \rightarrow \text{John} \rightarrow \text{saw}$  is 29

# Chu-Liu-Edmonds Example



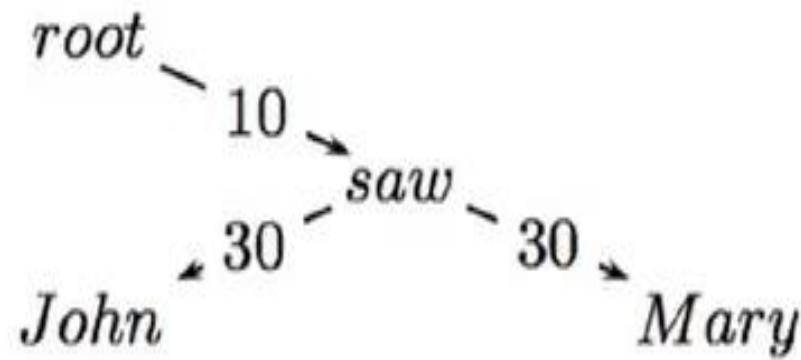
# Chu-Liu-Edmonds Example

Calling the algorithm again on the contracted graph:



- This is a tree and the MST for the contracted graph
- Go back up the recursive call and reconstruct final graph

# 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*.

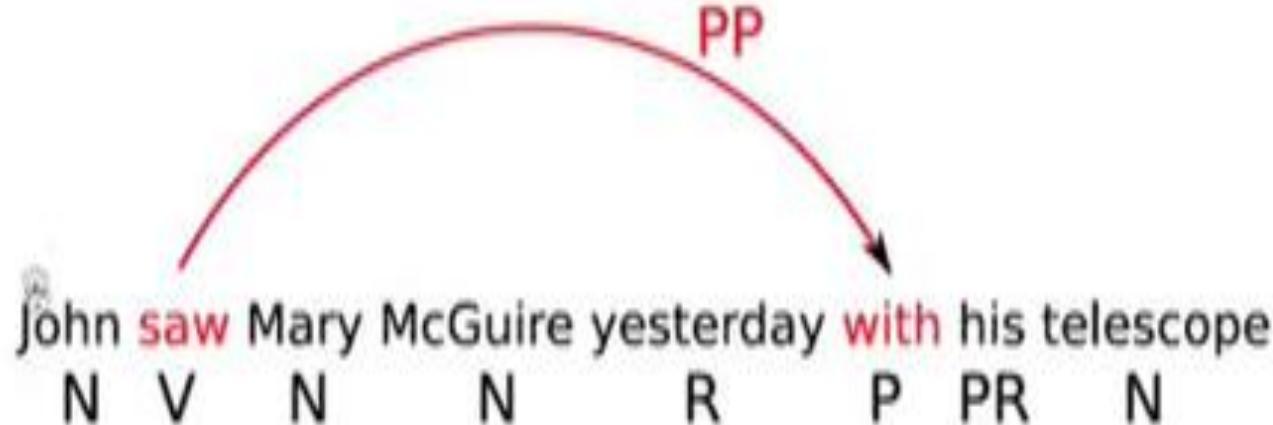
# Linear classifiers

---

$$w_{ij}^k = w.f(i,j,k)$$

- Arc weights are a linear combination of features of the arc  $f(i,j,k)$  and a corresponding weight vector  $w$
- What arc features?

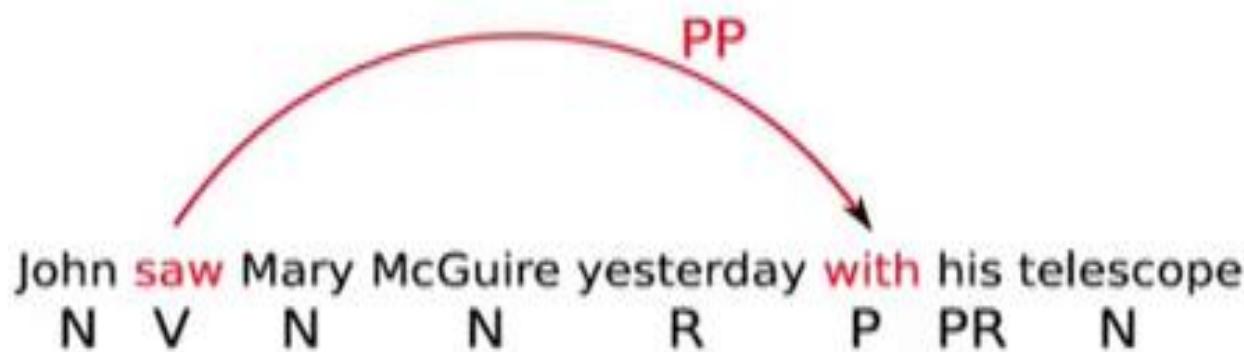
# Arc features



## Features

Identities of the words  $w_i$  and  $w_j$  for a label  $l_k$   
head = saw & dependent=with

# Arc features



## Features

Part-of-speech tags of the words  $w_i$  and  $w_j$  for a label  $l_k$   
head-pos = Verb & dependent-pos=Preposition

# Arc features



## Features

Part-of-speech of words surrounding and between  $w_i$  and  $w_j$

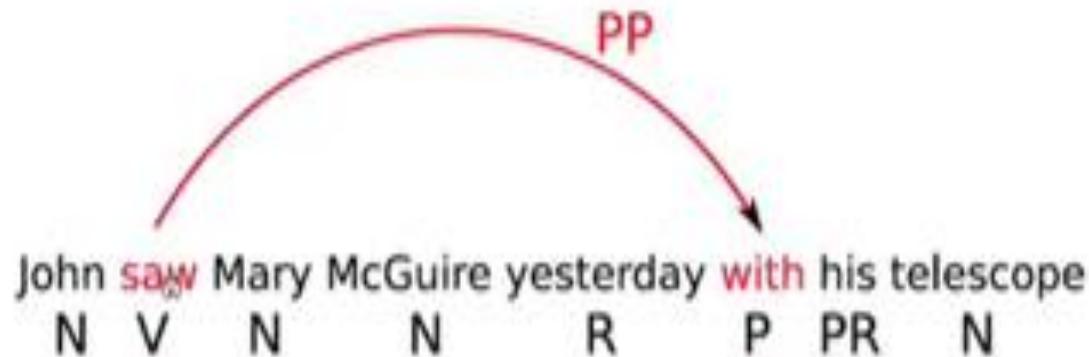
inbetween-pos = Noun

inbetween-pos = Adverb

dependent-pos-right = Pronoun

head-pos-left=Noun

# Arc features



## 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

$$\begin{aligned}
 G &= \arg \max_{G \in T(G_x)} \sum_{(i,j,k) \in G} w_{ij}^k \\
 &= \arg \max_{G \in T(G_x)} w \cdot \sum_{(i,j,k) \in G} f(i,j,k) \\
 &\quad \text{...} \\
 &= \arg \max_{G \in T(G_x)} w \cdot f(G)
 \end{aligned}$$

# 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

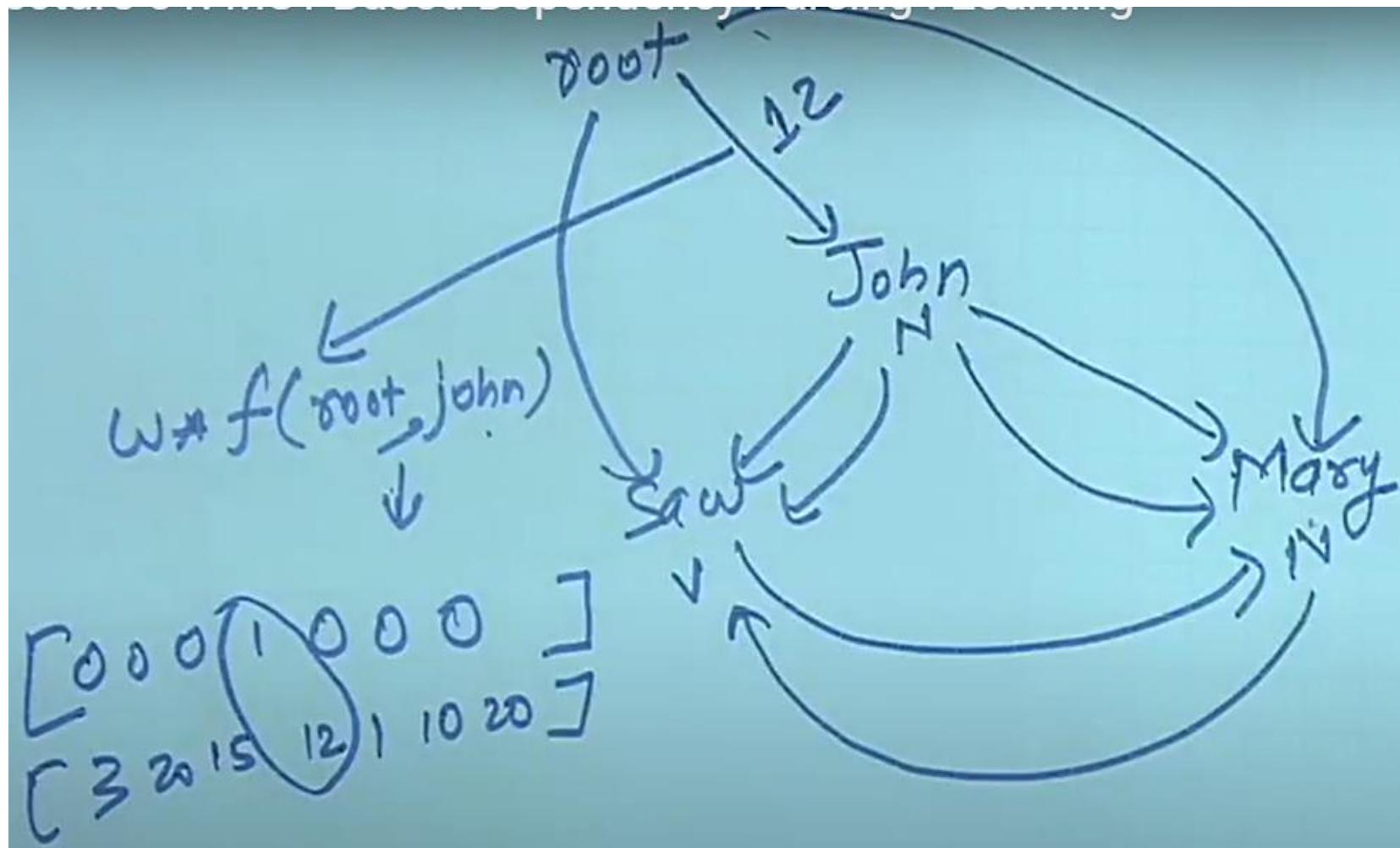
Suppose you are training MST Parser for dependency and the sentence, "John saw Mary" occurs in the training set. Also, for simplicity, assume that there is only one dependency relation, "rel". Thus, for every arc from word  $w_i$  to  $w_j$ , your features may be simplified to depend only on words  $w_i$  and  $w_j$  and not on the relation label.

Below is the set of features

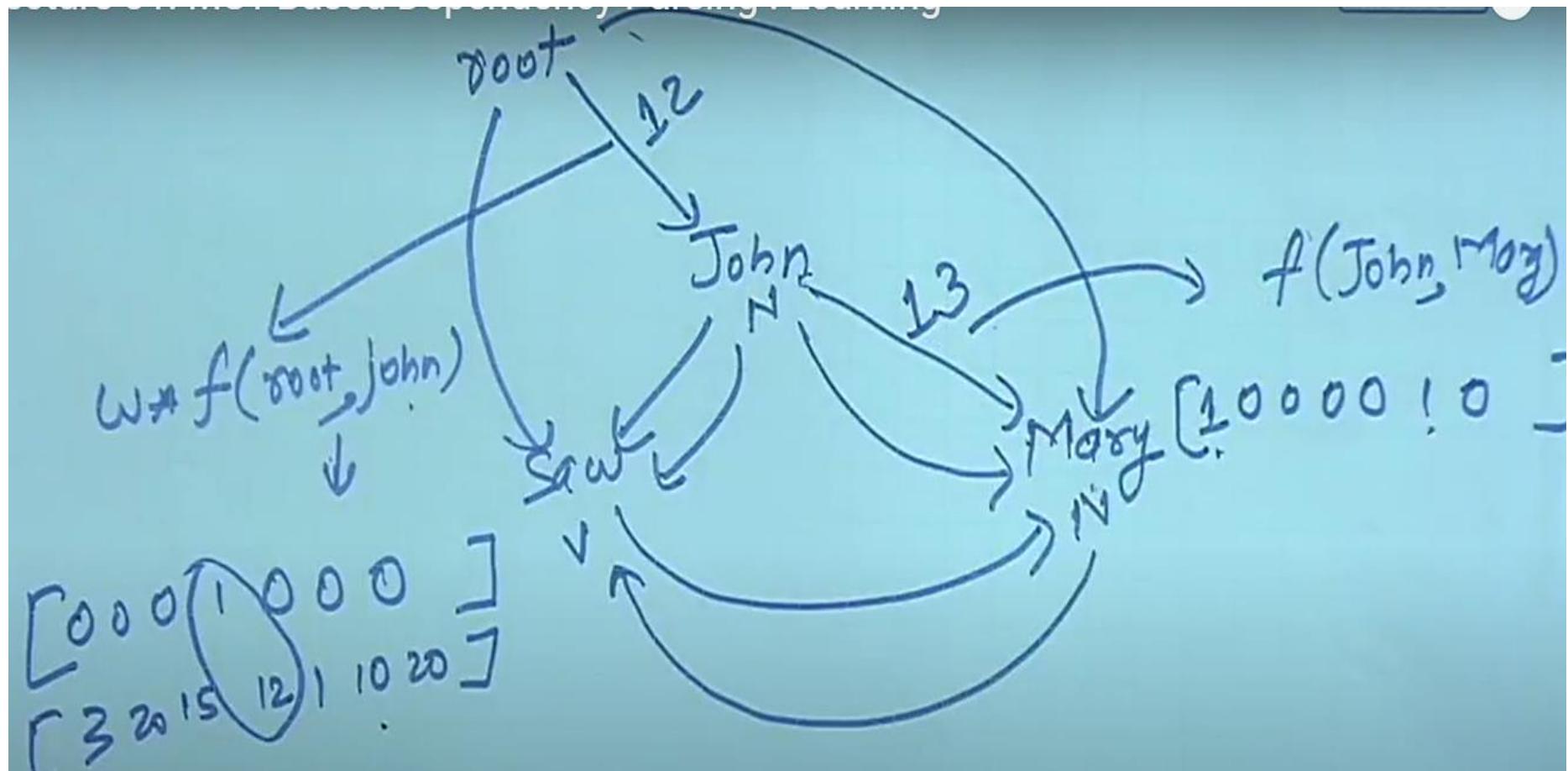
- $f_1: \text{pos}(w_i) = \text{Noun}$  and  $\text{pos}(w_j) = \text{Noun}$
- $f_2: \text{pos}(w_i) = \text{Verb}$  and  $\text{pos}(w_j) = \text{Noun}$
- $f_3: w_i = \text{Root}$  and  $\text{pos}(w_j) = \text{Verb}$
- $f_4: w_i = \text{Root}$  and  $\text{pos}(w_j) = \text{Noun}$
- $f_5: w_i = \text{Root}$  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.

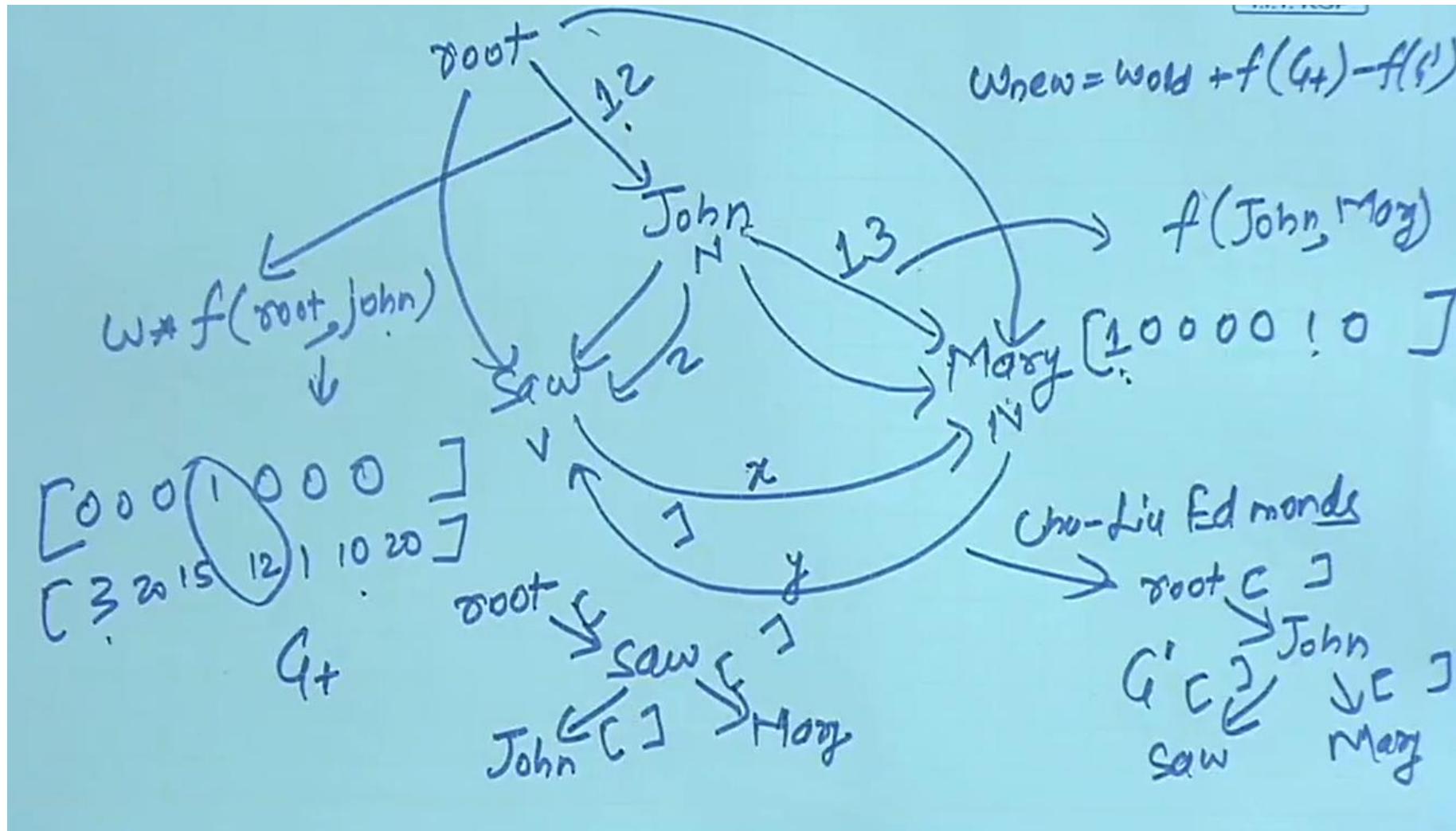
# EXAMPLE



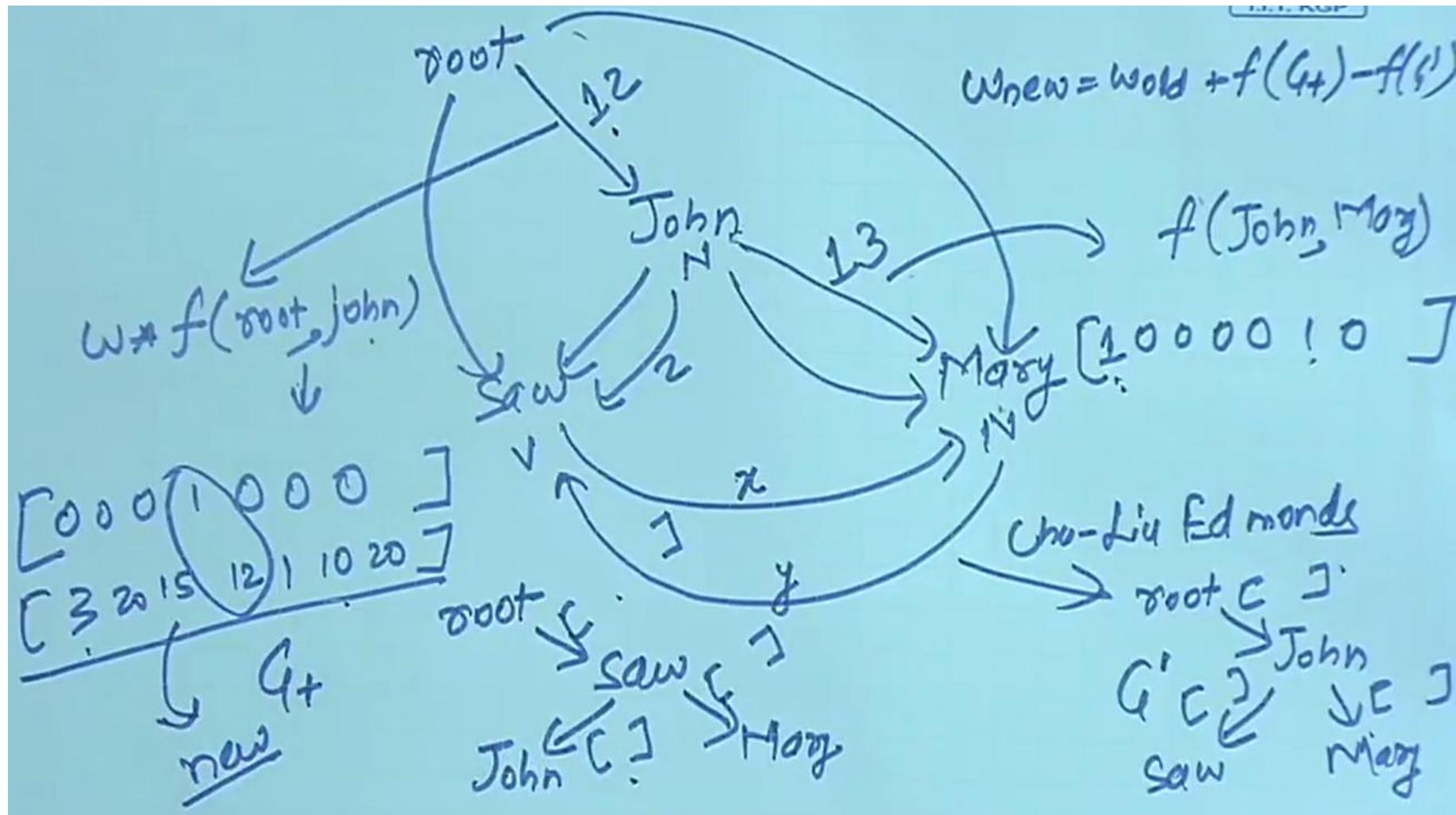
# EXAMPLE



# EXAMPLE



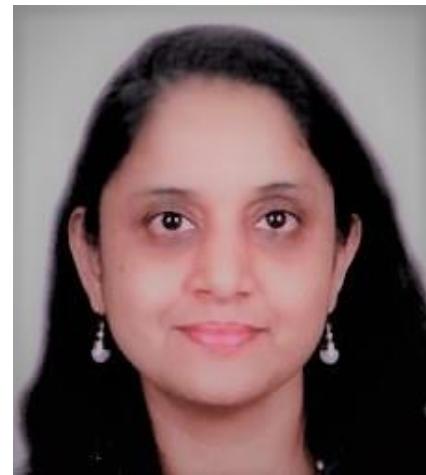
# EXAMPLE



# Extra Reading

---

- Speech and Language processing: An introduction to Natural Language Processing, Computational Linguistics and speech Recognition by Daniel Jurafsky and James H. Martin[3rd edition].
- [https://www.youtube.com/watch?v=egBq3gi\\_4No](https://www.youtube.com/watch?v=egBq3gi_4No)
- [https://www.youtube.com/watch?v=R1wL7sA\\_hHM&list=PLzJaFd3A7DZutMK8fFxZx\\_mhmFQg zijGE&index=28](https://www.youtube.com/watch?v=R1wL7sA_hHM&list=PLzJaFd3A7DZutMK8fFxZx_mhmFQg zijGE&index=28)
- [https://www.researchgate.net/publication/328731166\\_Weighted\\_Machine\\_Learning](https://www.researchgate.net/publication/328731166_Weighted_Machine_Learning)
- <https://realpython.com/natural-language-processing-spacy-python/>



**Thank you for your time!!**



# Natural Language Processing

## DSECL ZG565

Dr. Chetana Gavankar, Ph.D,

IIT Bombay-Monash University Australia

Associate Professor, BITS Pilani

[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



**BITS** Pilani  
Pilani Campus



## **Session 12 – Word sense disambiguation and WordNet**

### **Date – 2nd September 2023**

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 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

# What's a word sense?

- **Lexeme**: An entry in a lexicon consisting of a pairing of a form with a single meaning representation
- A **lemma** or **citation form** is the grammatical form that is used to represent a **lexeme**.
  - **Carpet** is the lemma for **carpets**
- The lemma **bank** has two **senses**:
  - Instead, a **bank** can hold the investments in a custodial account in the client's name
  - But as agriculture burgeons on the east **bank**, the river will shrink even more.
- A **sense** is a discrete representation of one aspect of the meaning of a word

# Word senses and 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
  - Same orthographic form but different phonological form
    - bass vs bass
- Information retrieval
  - Different meanings same orthographic form
    - QUERY: bat care
- Machine Translation
- Speech recognition
  - Why?

# 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*

| superordinate | vehicle | fruit | furniture | mammal |
|---------------|---------|-------|-----------|--------|
| hyponym       | car     | mango | chair     | dog    |

# WordNet

- A hierarchically organized lexical database
- On-line thesaurus + aspects of a dictionary
  - Versions for other languages are under development
- Avr. noun has 1.23 sense
- Avr. verb has 2.16 senses

| Category  | Entries |
|-----------|---------|
| Noun      | 117,097 |
| Verb      | 11,488  |
| Adjective | 22,141  |
| Adverb    | 4,601   |

# Format of Wordnet Entries

The noun “bass” has 8 senses in WordNet.

1. bass<sup>1</sup> - (the lowest part of the musical range)
2. bass<sup>2</sup>, bass part<sup>1</sup> - (the lowest part in polyphonic music)
3. bass<sup>3</sup>, basso<sup>1</sup> - (an adult male singer with the lowest voice)
4. sea bass<sup>1</sup>, bass<sup>4</sup> - (the lean flesh of a saltwater fish of the family Serranidae)
5. freshwater bass<sup>1</sup>, bass<sup>5</sup> - (any of various North American freshwater fish with  
lean flesh (especially of the genus Micropterus))
6. bass<sup>6</sup>, bass voice<sup>1</sup>, basso<sup>2</sup> - (the lowest adult male singing voice)
7. bass<sup>7</sup> - (the member with the lowest range of a family of musical instruments)
8. bass<sup>8</sup> - (nontechnical name for any of numerous edible marine and  
freshwater spiny-finned fishes)

The adjective “bass” has 1 sense in WordNet.

1. bass<sup>1</sup>, deep<sup>6</sup> - (having or denoting a low vocal or instrumental range)  
*“a deep voice”; “a bass voice is lower than a baritone voice”;*  
*“a bass clarinet”*

# 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>, 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>1</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
```

# WordNet as graph

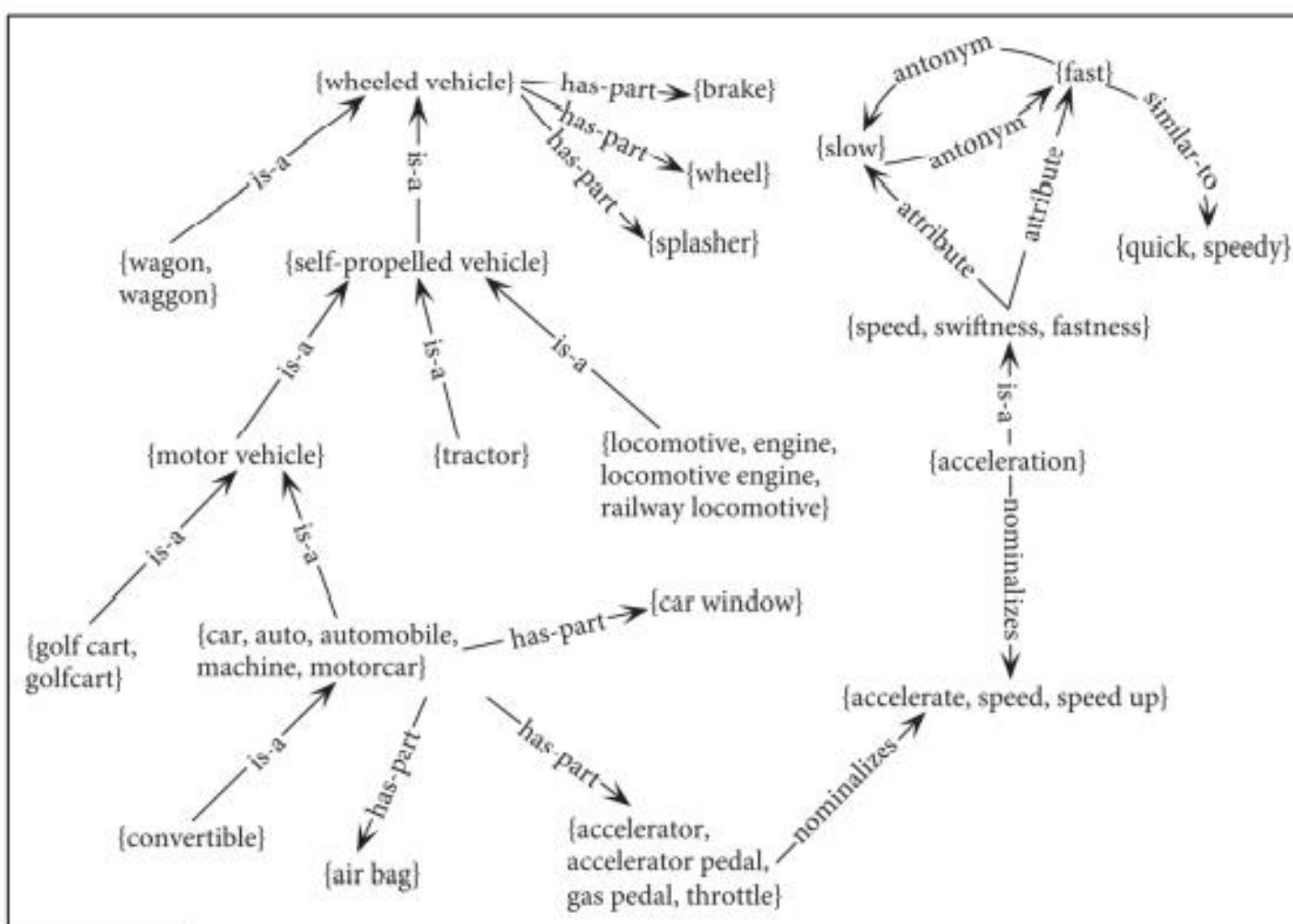


Figure 19.6 WordNet viewed as a graph. Figure from Navigli (2016).

# 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.
- 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.

# Inventory of sense tags for *bass*

| WordNet Sense     | Spanish Translation | Roget Category | Target Word in Context                                    |
|-------------------|---------------------|----------------|---|
| bass <sup>4</sup> | lubina              | FISH/INSECT    | ... fish as Pacific salmon and striped <b>bass</b> and... |
| bass <sup>4</sup> | lubina              | FISH/INSECT    | ... produce filets of smoked <b>bass</b> or sturgeon...   |
| bass <sup>7</sup> | bajo                | MUSIC          | ... exciting jazz <b>bass</b> player since Ray Brown...   |
| bass <sup>7</sup> | bajo                | MUSIC          | ... play <b>bass</b> 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
- 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 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

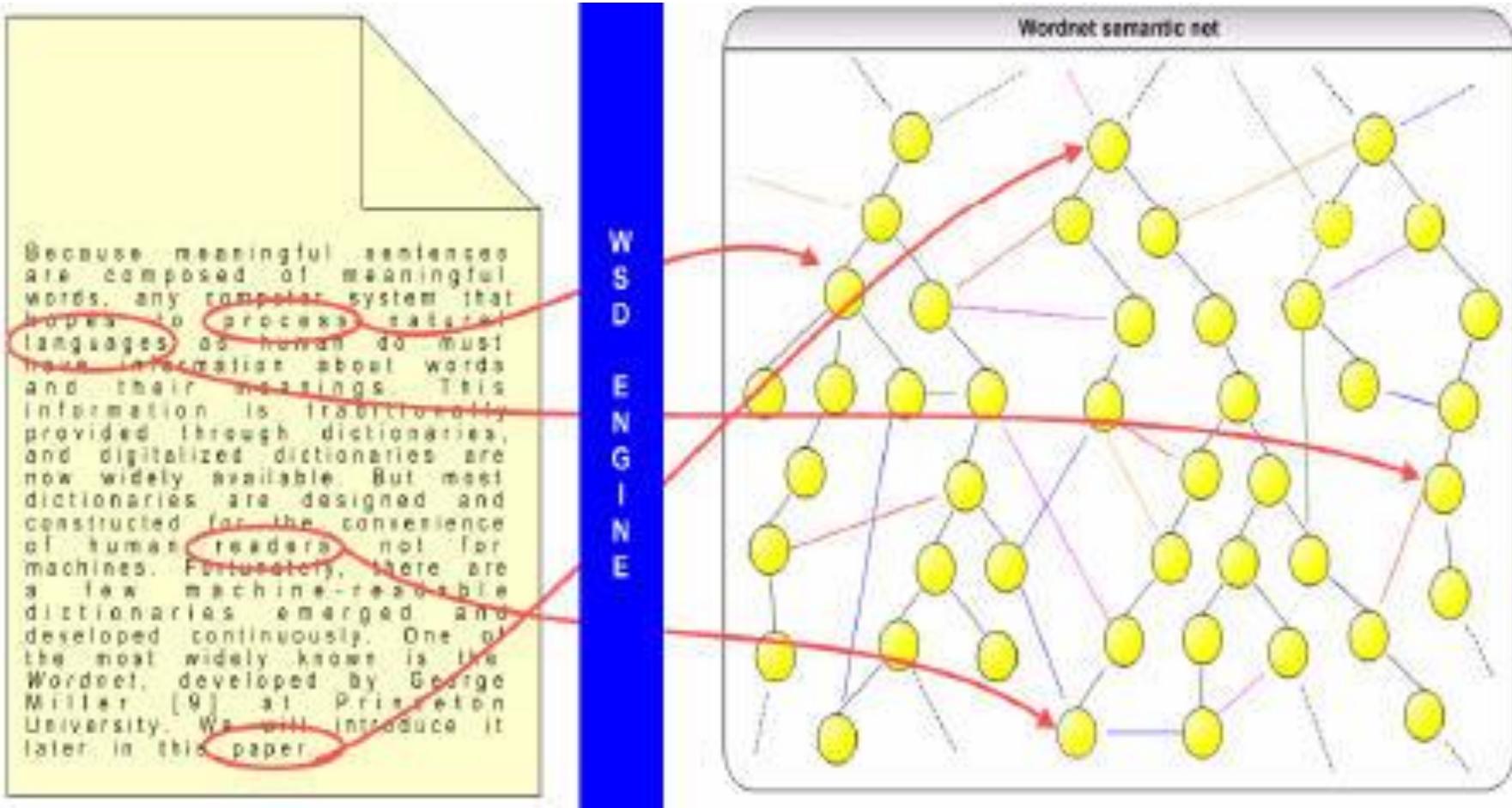
The noun ``bass'' has 8 senses in WordNet

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)

# Training Corpus

- Lexical sample task:
  - *Line-hard-serve* corpus - 4000 examples of each
  - *Interest* corpus - 2369 sense-tagged examples
  - Since the set of words and the set of senses are small, simple supervised classification approaches work very well.
- 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



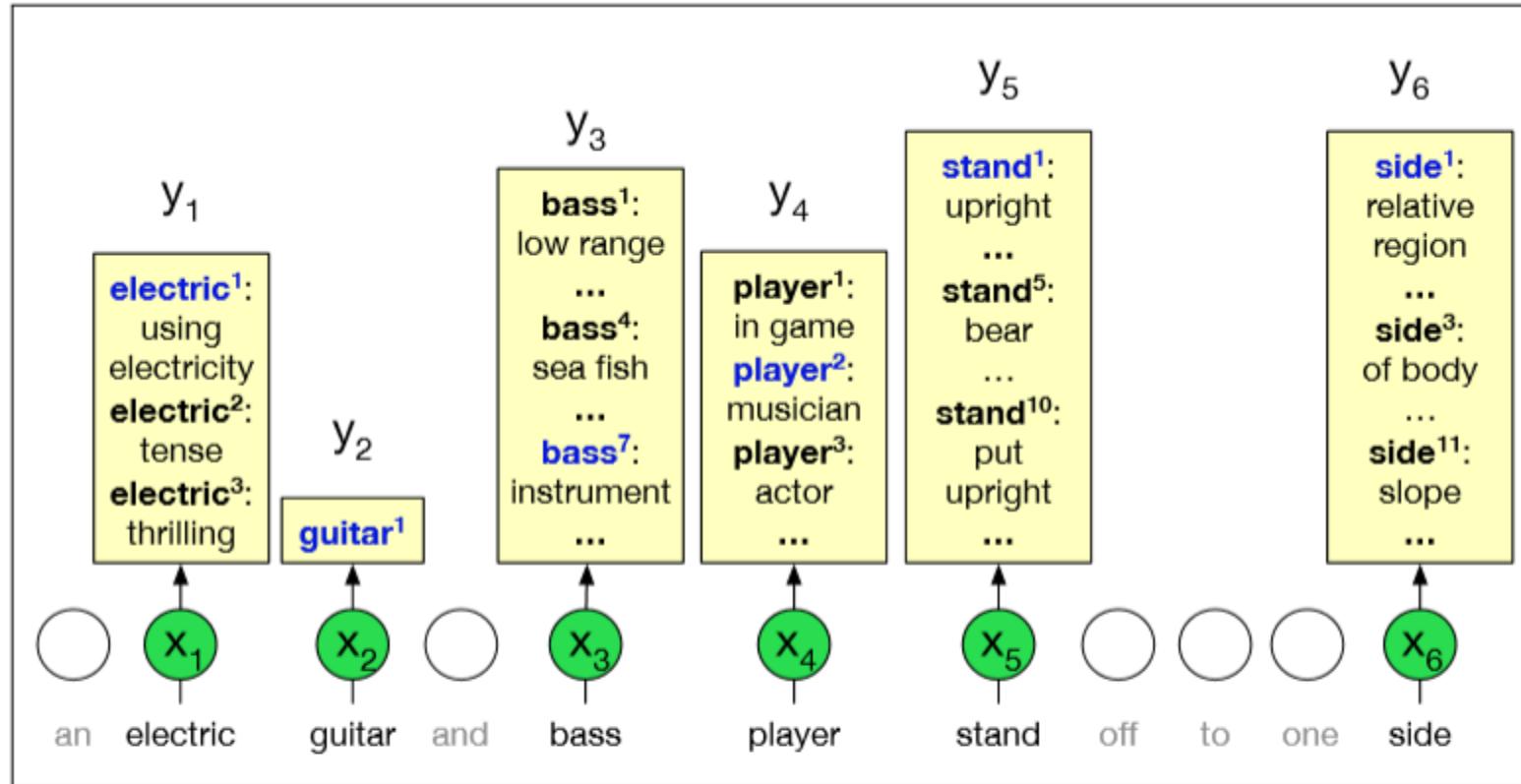
# Example of SemCor with wordnet sense numbers

---

**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 \(2018\)](#).

# WSD:The Task and Datasets

---

- Surprisingly strong baseline is simply to choose the **most frequent sense** for most frequent sense each word from the senses in a labeled corpus(Galeetal.,1992a).
  - For WordNet, this corresponds to the first sense, since senses in WordNet are generally ordered from most frequent to least frequent
  - Most frequent sense baseline can be quite accurate, and is therefore often used as a default, to supply a word sense when a supervised algorithm **has insufficient training data**
  - Another heuristic **one sense per discourse**: Word appearing multiple times in a text or discourse often appears with the same sense
-

# WSD algorithms

---

- Supervised Learning –
  - Feature-Based WSD
- LESK Algorithm
- Word-in-Context Evaluation
- Wikipedia as a source of training data

# 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 Learning

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)

# 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

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

# 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 mall 2-word window, a standard feature vector might include parts-of speech, unigram and bigram collocation features, and a weighted sum of embeddings, that is:
- $[w_{i-2}, \text{POS}_{i-2}, w_{i-1}, \text{POS}_{i-1}, w_{i+1}, \text{POS}_{i+1}, w_{i+2}, \text{POS}_{i+2}, w_{i-2}^{i-1}, w_{i+1}^{i+2}, g(E(w_{i-2}), E(w_{i-1}), E(w_{i+1}), E(w_{i+2}))]$   
would yield the following vector:

**[guitar, NN, and, CC, player, NN, stand, VB, and guitar, player stand, g(E(guitar), E(and), E(player), E(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 representation

---

- A very popular and basic representation of documents is the bag of words model.
- Each document is represented by a bag (= multiset) of terms from a predefined vocabulary.

# Bag of Words representation



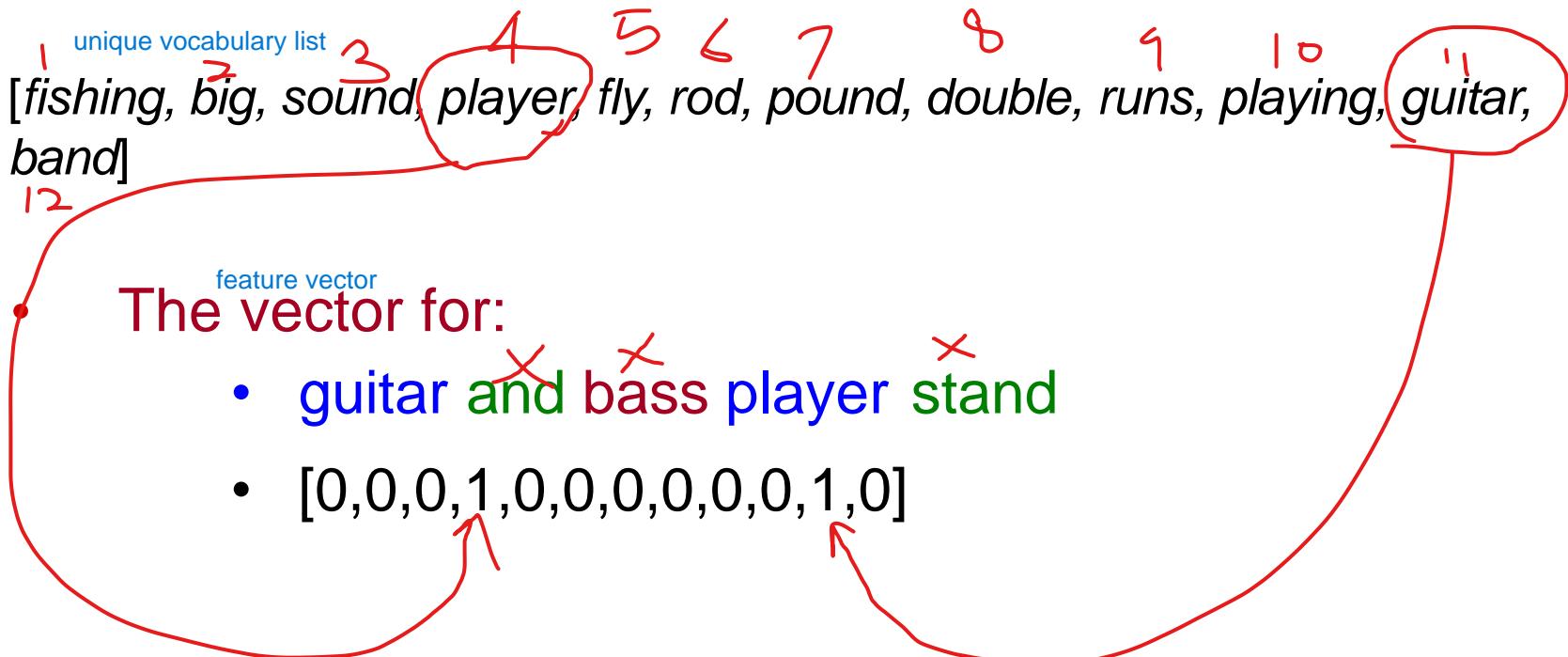
The Jackal was eyeing at  
the grapes

He was as cunning as a  
Jackal

These Grapes are too sweet but the poor Jackal could not have it.

# Co-Occurrence Example

- Assume we've settled on a possible vocabulary of 12 words that includes **guitar** and **player** but not **and** and **stand**



# Supervised Learning Algorithm

---

*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  $\gamma: d \rightarrow c$

# Applying Naïve Bayes Classifier

---

$P(c)$  is the prior probability of that sense

- Counting in a labeled training set.

$P(w|c)$  conditional probability of a word given a particular sense

- $P(w|c) = \text{count}(w,c)/\text{count}(c)$

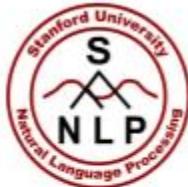
We get both of these from a tagged corpus like SemCor

Can also generalize to look at other features besides words.

- Then it would be  $P(f|c)$ 
  - Conditional probability of a feature given a sense

# Applying Naïve Bayes Classifier

Dan Jurafsky



$$\hat{P}(c) = \frac{N_c}{N}$$

$$\hat{P}(w|c) = \frac{\text{count}(w,c)+1}{\text{count}(c)+|V|}$$

|          | Doc | Words                 | Class |
|----------|-----|-----------------------|-------|
| Training | 1   | fish smoked fish      | f     |
|          | 2   | fish line             | f     |
|          | 3   | fish haul smoked      | f     |
|          | 4   | guitar jazz line      | g     |
| Test     | 5   | line guitar jazz jazz | ?     |

Priors:

$$P(f) = \frac{3}{4}$$

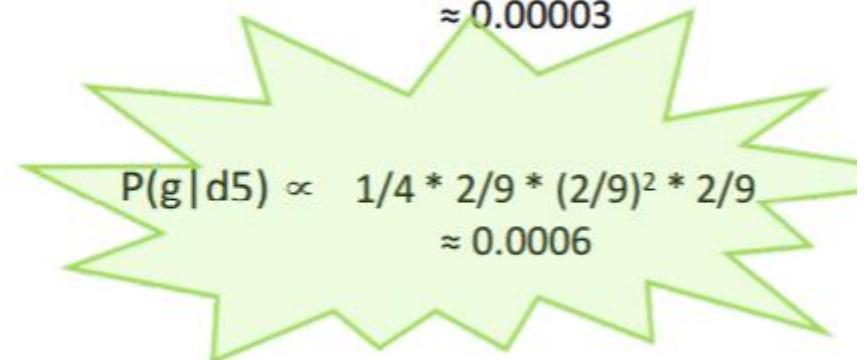
$$P(g) = \frac{1}{4}$$

$$V = \{\text{fish, smoked, line, haul, guitar, jazz}\}$$

Choosing a class:

$$P(f|d5) \propto 3/4 * 2/14 * (1/14)^2 * 1/14$$

$$\approx 0.00003$$



Conditional Probabilities:

$$P(\text{line}|f) = (1+1) / (8+6) = 2/14$$

$$P(\text{guitar}|f) = (0+1) / (8+6) = 1/14$$

$$P(\text{jazz}|f) = (0+1) / (8+6) = 1/14$$

$$P(\text{line}|g) = (1+1) / (3+6) = 2/9$$

$$P(\text{guitar}|g) = (1+1) / (3+6) = 2/9$$

$$P(\text{jazz}|g) = (1+1) / (3+6) = 2/9$$

# The WSD Algorithm: Simple 1-nearest-neighbor algorithm

---

- Best-performing WSD algorithm is a simple 1-nearest-neighbor algorithm using contextual word embedding's
- 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.

*An important idea in linguistics is that words (or expressions) that can be used in similar ways are likely to have related meanings.*

# 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

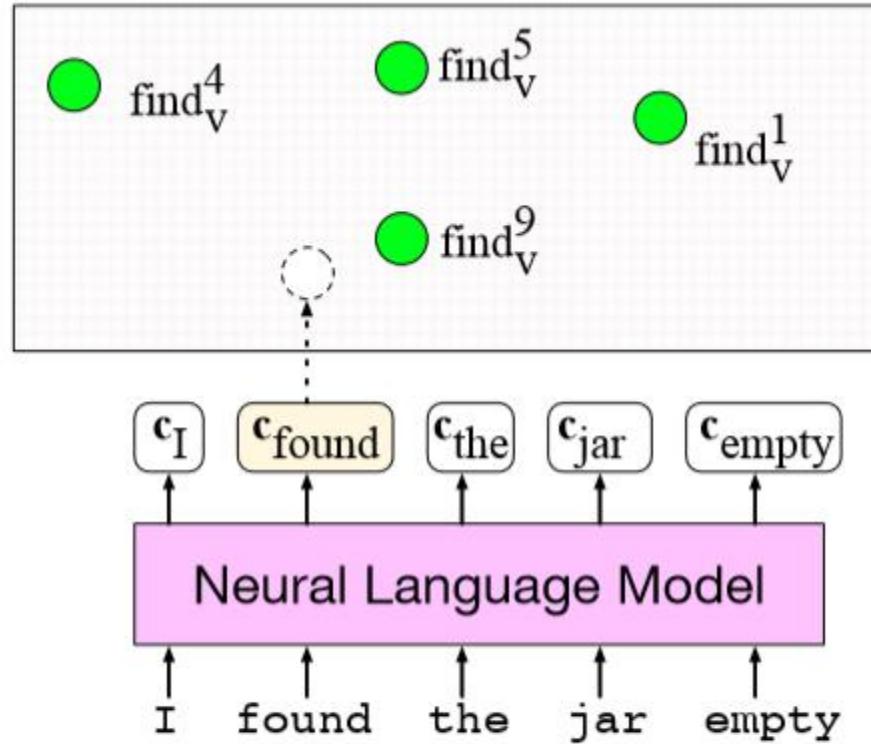
| context words | v(astronomers) | v(bodies) | v(objects) |
|---------------|----------------|-----------|------------|
| 't            |                |           | 1          |
| ,             |                | 2         | 1          |
| .             | 1              |           | 1          |
| 1             |                |           | 1          |
| And           |                |           | 1          |
| Belt          |                |           | 1          |
| But           | 1              |           |            |
| Given         |                |           | 1          |
| Kuiper        |                |           | 1          |
| So            | 1              |           |            |
| and           |                | 1         |            |
| are           |                | 2         | 1          |
| between       |                |           | 1          |
| beyond        |                | 1         |            |
| can           |                |           | 1          |
| contains      |                | 1         |            |
| from          | 1              |           |            |
| hypothetical  |                |           | 1          |
| ice           |                | 1         |            |
| including     |                | 1         |            |
| is            | 1              |           |            |
| larger        |                | 1         |            |
| now           | 1              |           |            |
| of            | 1              |           |            |
| only          |                |           | 1          |
| out           |                | 1         |            |
| potential     |                | 1         |            |
| the           | 1              |           | 1          |
| these         |                | 2         | 1          |
| they          | 1              |           |            |
| think         | 2              |           |            |
| those         |                |           | 1          |
| thought       |                | 2         |            |
| what          | 1              |           |            |

# Contextual Embedding

$$\text{cosine\_similarity}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \cdot \|\mathbf{v}\|}$$

|             | astronomers                                | bodies                                     | objects   |
|-------------|--|--|---|
| astronomers | $\frac{14}{\sqrt{14} \cdot \sqrt{14}} = 1$ | $\frac{0}{\sqrt{24} \cdot \sqrt{14}} = 0$  | $\frac{1+1}{\sqrt{14} \cdot \sqrt{16}} \approx 0.134$   |
| bodies      |  | $\frac{24}{\sqrt{24} \cdot \sqrt{24}} = 1$ | $\frac{2+2+2}{\sqrt{24} \cdot \sqrt{16}} \approx 0.306$ |
| objects     |  |  | $\frac{16}{\sqrt{16} \cdot \sqrt{16}} = 1$              |

# Nearest-neighbor algorithm for WSD



**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).

# The Lesk Algorithm as WSD Baseline

---



- Knowledge-based algorithms, rely solely on knowledge based WordNet or other such resources and don't require labeled data.
  - While supervised algorithms generally work better, knowledge-based methods can be used in languages or domains where thesauruses or dictionaries but not sense labeled corpora are available.
  - 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.***

- Wordnet senses of “bank”:

|                   |           |   |
|-------------------|-----------|---|
| bank <sup>1</sup> | Gloss:    | a financial institution that accepts <b>deposits</b> and channels the money into lending activities |
|                   | Examples: | “he cashed a check at the bank”, “that bank holds the <b>mortgage</b> on my home”                   |
| bank <sup>2</sup> | 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.

# Corpus Lesk Algorithm

---

- Assumes we have some sense-labeled data (like SemCor)
- Take all the sentences with the relevant word sense:

*These short, "streamlined" meetings usually are sponsored by local **banks**<sup>1</sup>, Chambers of Commerce, trade associations, or other civic organizations.*
- Now add these to the gloss + examples for each sense, call it the “signature” of a sense.
- Choose sense with most word overlap between context and signature.

# 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.

# Unsupervised Learning: 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.

# What we covered in todays session



- Word sense is the locus of word meaning; definitions and meaning relations are defined at the level of the word sense rather than word forms.
- Many words are polysemous, having many senses.
- Relations between senses include synonymy, antonymy, meronymy, and taxonomic relations hyponymy and hypernymy.
- WordNet is a large database of lexical relations for English, and exist for a variety of languages.
- WSD is the task of determining the correct sense of a word in context.

# What we covered in todays session



- Supervised approaches make use of a corpus of sentences in which individual words (lexical sample task) or all words (all-words task) are hand-labeled with senses from a resource like WordNet.
- SemCor is the largest corpus with WordNet-labeled senses.
- The standard supervised algorithm for WSD is nearest neighbors with contextual embeddings.
- Feature-based algorithms using parts of speech and embeddings of words in the context of the target word also work well

# What we covered in todays session

---

- An important baseline for WSD is the most frequent sense, equivalent, in WordNet, to take the first sense.
- Another baseline is a knowledge-based WSD algorithm called the Lesk algorithm which chooses the sense whose dictionary definition shares the most words with the target word's neighborhood.
- Word sense induction is the task of learning word senses using unsupervised learning

# References

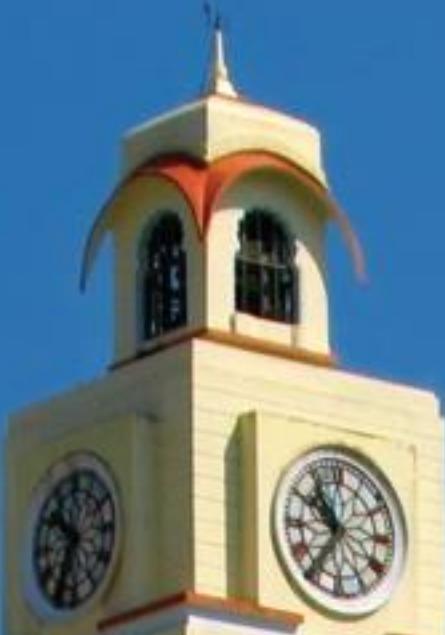
---

- <https://wordnet.princeton.edu/>
  - <https://babelnet.org/>
  - <https://aclanthology.org/2022.coling-1.368.pdf>
  - <https://aclanthology.org/2022.wildre-1.4.pdf>
  - <https://www.sciencedirect.com/science/article/abs/pii/S0885230821001303>
-



---

# Thank You



# Natural Language Processing

## DSECL ZG565



**BITS** Pilani  
Pilani Campus

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



## **Session 13: Semantic Web Ontology**

### **Date – 3rd September 2023**

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Philip Cimiano , Johanna Völker , Paul Buitelaar and many others who made their course materials freely available online.

# Session Content

- Introduction
- Semantic Web and Ontology
- Ontology Languages
- Ontology Engineering
- State of the Art
  - Ontology Learning
  - Knowledge Graphs
  - Linked open data

# A Brief History of Web Search

- 1991: Tim Berners-Lee “invents” the World Wide Web
- First Web search engines:
  - Archie: Query file names by regular expressions
  - Architext/Excite: Full text search, simple ranking (1993)
- Until 1998, web search meant information retrieval
- 1998: Google was founded – Exploits link structure using the PageRank algorithm



Copyright ©1998 Google Inc.

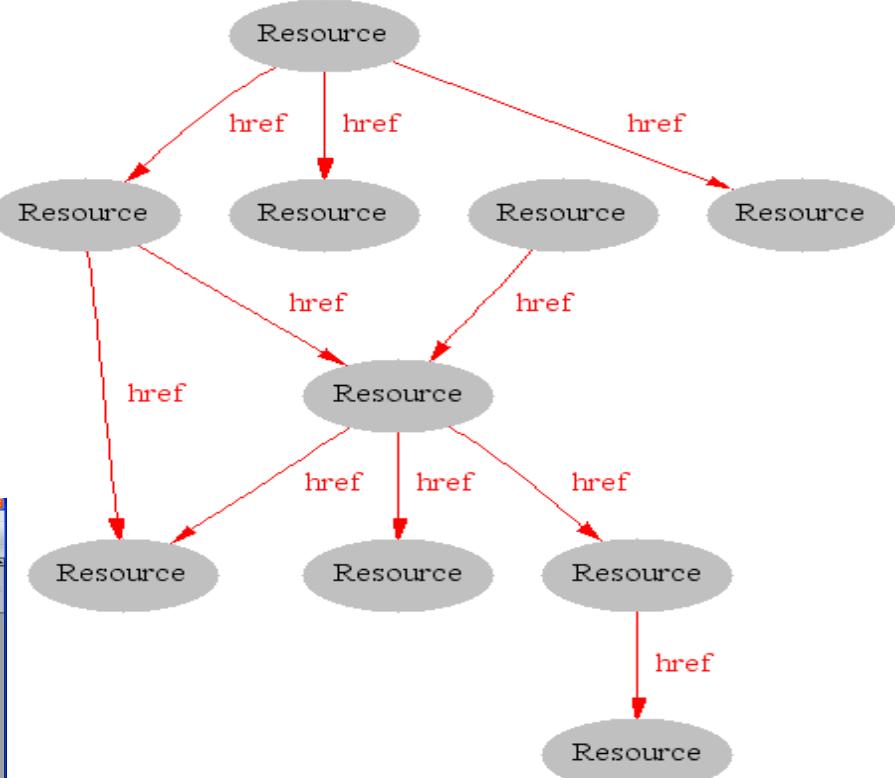
# Problem: Computers don't understand Meaning

*“My mouse is broken. I need a new one”*



# The Syntactic Web

The image shows two screenshots. The top screenshot is the homepage of the "WWW 2002" conference, titled "THE ELEVENTH INTERNATIONAL WORLD WIDE WEB CONFERENCE" held at the Sheraton Waikiki Hotel in Honolulu, Hawaii, USA from May 7-11, 2002. It features a logo with a stylized figure, a map of the world, and sections for "CONFERENCE HIGHLIGHTS" and "REGISTER NOW". The bottom screenshot is a screenshot of Tim Berners-Lee's homepage, showing his biography, publications, and other links.



[Hendler & Miller 02]

# The Syntactic Web is...

- A hypermedia, a digital library
  - A library of documents called (web pages) interconnected by a hypermedia of links
- A database, an application platform
  - A common portal to applications accessible through web pages, and presenting their results as web pages
- A platform for multimedia
  - BBC Radio anywhere in the world! Terminator 3 trailers!
- A naming scheme
  - Unique identity for those documents

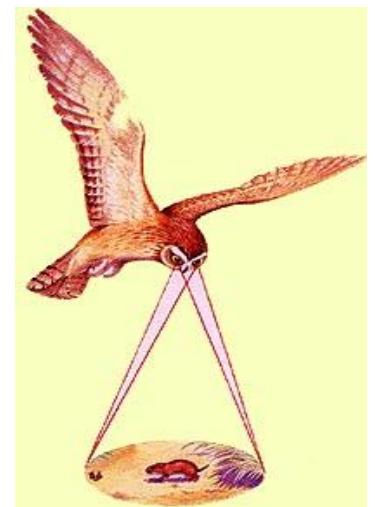
A place where computers do the presentation (easy) and people do the linking and interpreting (hard).

Why not get computers to do more of the hard work?

# 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”
  
- Locating information in **data repositories**
  - Results of human genome experiments

e.g., Barn Owl



# What is the Problem?

Consider a typical web page:

The screenshot shows the homepage of the WWW 2002 conference. At the top, it says "WWW 2002 THE ELEVENTH INTERNATIONAL WORLD WIDE WEB CONFERENCE" and "Sheraton Waikiki Hotel, Honolulu, Hawaii, USA 7-11 May 2002". Below this, a banner reads "1 LOCATION. 5 DAYS. LEARN. INTERACT.". On the left, there's a sidebar with links: "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 features a "REGISTER NOW" button and information about registered participants from various countries. It also mentions the organizing committee (IWC) and partners (University of Hawaii and PTC). A section titled "FEATURED SPEAKERS (CONFIRMED)" lists Tim Berners-Lee and Richard A. DeMillo.

- 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 bernes-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”



*Photo credit Alex (ajagendorf25), Flickr*

# Web data issues

## Many challenges in data management

---

- ▶ Heterogeneous data
- ▶ Implicit semantics
- ▶ No background knowledge
- ▶ Data quality difficult to measure
- ▶ Proprietary schemas
- ▶ Unstructured data
- ▶ Ambiguity
- ▶ Multilinguality
- ▶ Various units and currencies

# 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...

The image shows three overlapping browser windows from Mozilla Firefox:

- CoCoDat - Collation of Cortical Data - Mozilla Firefox**: A database for cortical microcircuitry. It lists morphology, firing properties, currents, conductances, synaptic currents, and connectivity. It also mentions a search board for relaxation and a manual or automatic search.
- Cell Centered Database - Mozilla Firefox**: A national center for microscopy and imaging research. It features a "Gallery" section with various microscopy images and links to data, search, dictionary, publications, and contact us.
- NeuronDB - Thalamic relay neuron - Overview (A) () - Mozilla Firefox**: A detailed view of a thalamic relay neuron. It includes a diagram of the neuron with labels for Soma, Dem, Dep, Ded, and S. Below the diagram, a list of compartments is shown with "Show other" buttons:
  1. Equivalent dendrite
  2. Distal equivalent dendrite
  3. Middle equivalent dendrite
  4. Proximal equivalent dendrite
  5. SomaA sidebar on the right contains a log of recent activity.

# 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  
inFavourOf ?song) AND (?vote madeAt ?date)  
AND (Obama electedAt ?date) ORDER BY  
?likes DESC*

# A Web of Data unleashes new applications

The screenshot shows the homepage of data.gov.uk. At the top, there's a navigation bar with links like Home, Blog, Data, SPARQL, Apps, Ideas, Forum, Wiki, Resources, and About. Below this is a main banner with the text "Unlocking innovation" and "Working with UK Public Sector information and data". To the right of the banner is a large blue 3D molecular model. On the left, there's a search bar and a "Search Data" button. On the right, there's a "User login" form with fields for Username and Password, and buttons for Log In and Request new password. Below the login is a box titled "What is the Semantic Web?" with a brief description and a "Read more" button. At the bottom, there's a "Most Recent Apps" section and a "View all apps" button.

Unlocking innovation | data.gov.uk

http://data.gov.uk/ Google

Netvibes Feedly Social Private Mailing lists SW Python RDFa it! Bookmarklets Add Zemanta bit.ly To Mendeley TinyURL To Faviki

HM Government data.gov.uk

Home Blog Data SPARQL Apps Ideas Forum Wiki Resources About

Unlocking innovation

Working with UK Public Sector information and data

Advised by Sir Tim Berners-Lee and Professor Nigel Shadbolt and others, government are opening up data for reuse. This site seeks to give a way into the wealth of government data and is under constant development. We want to work with you to make it better.

We're very aware that there are more people like you outside of government who have the skills and abilities to make wonderful things out of public data. These are our first steps in building a collaborative relationship with you.

Search Data

Browse for Data

Enter keyword(s)  Search

e.g. education, NHS, crime, transport, environment

Powered by: CKAN

List all datasets

Common tags

Most Recent Apps

View all apps +

Subscribe by RSS

SHARE

User login

Username: \*

Password: \*

Log In

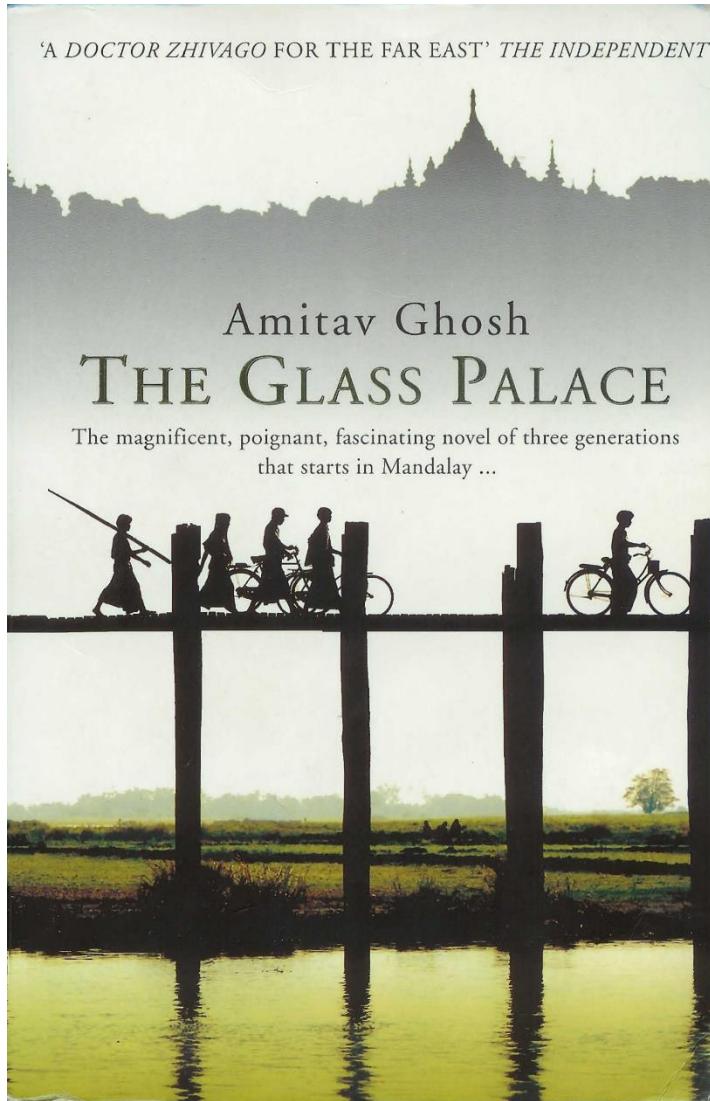
Request new password

What is the Semantic Web?

Combining different data sources has never been easy but the Semantic Web will enable data to be joined easily across boundaries.

Read more

# We start with a example of a book...



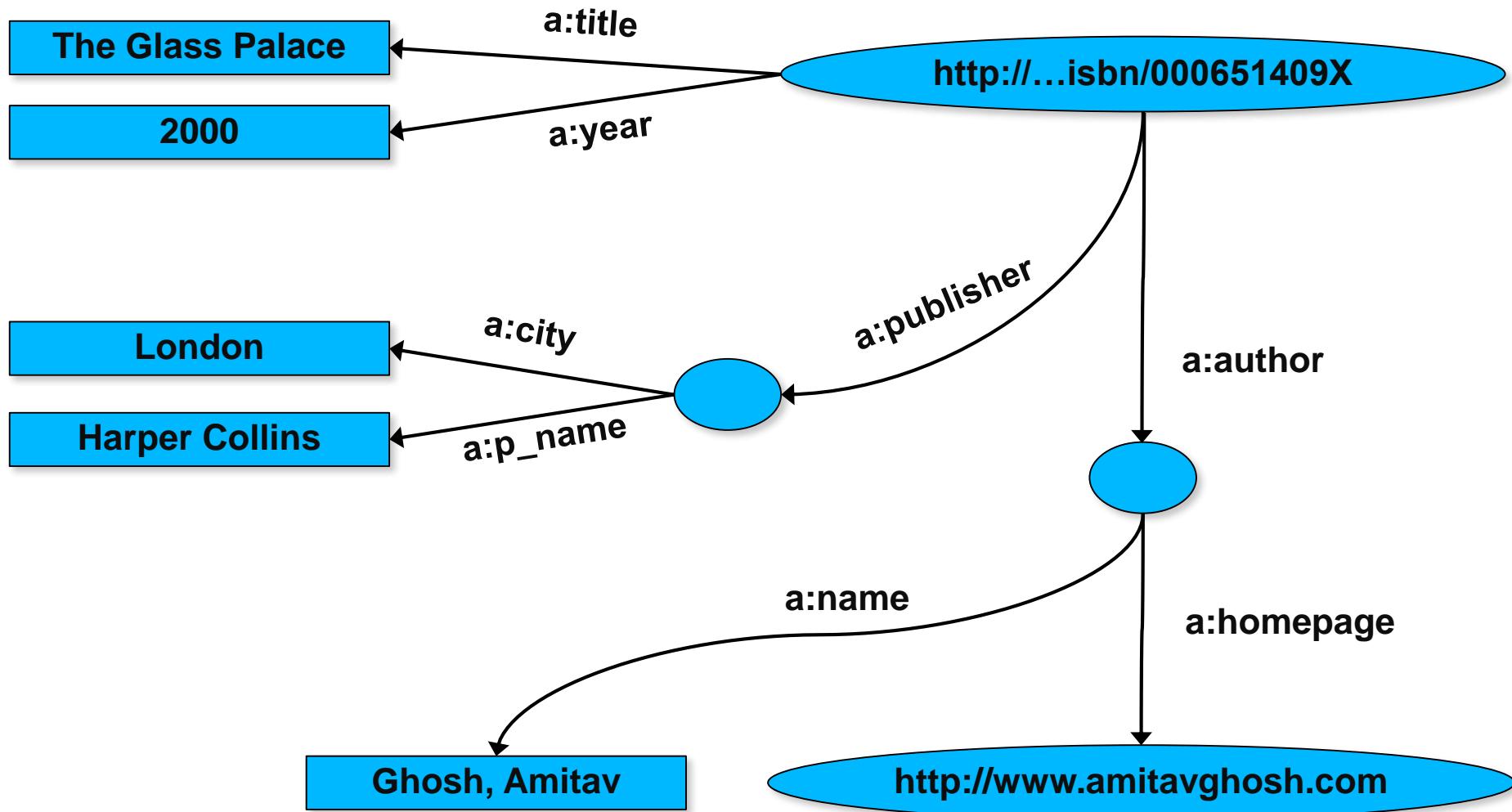
# A simplified bookstore data (dataset “A”)

| ID                  | Author | Title            | Publisher | Year |
|---------------------|--------|------------------|-----------|------|
| ISBN 0-00-6511409-X | id_xyz | The Glass Palace | id_qpr    | 2000 |

| ID     | Name          | Homepage  |
|--------|---------------|---|
| id_xyz | Ghosh, Amitav | <a href="http://www.amitavghosh.com">http://www.amitavghosh.com</a> |

| ID     | Publisher's name | City   |
|--------|------------------|--------|
| id_qpr | Harper Collins   | London |

# 1<sup>st</sup>: export your data as a set of relations



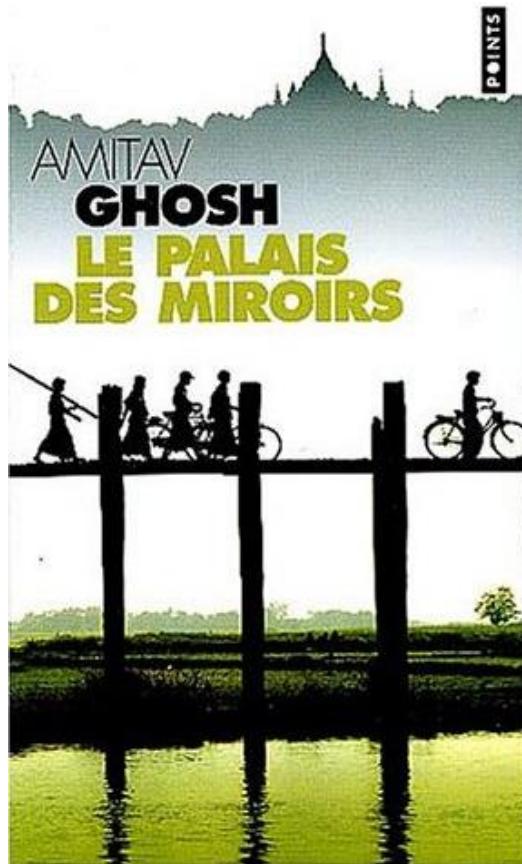
# Some notes on the exporting the data

- 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

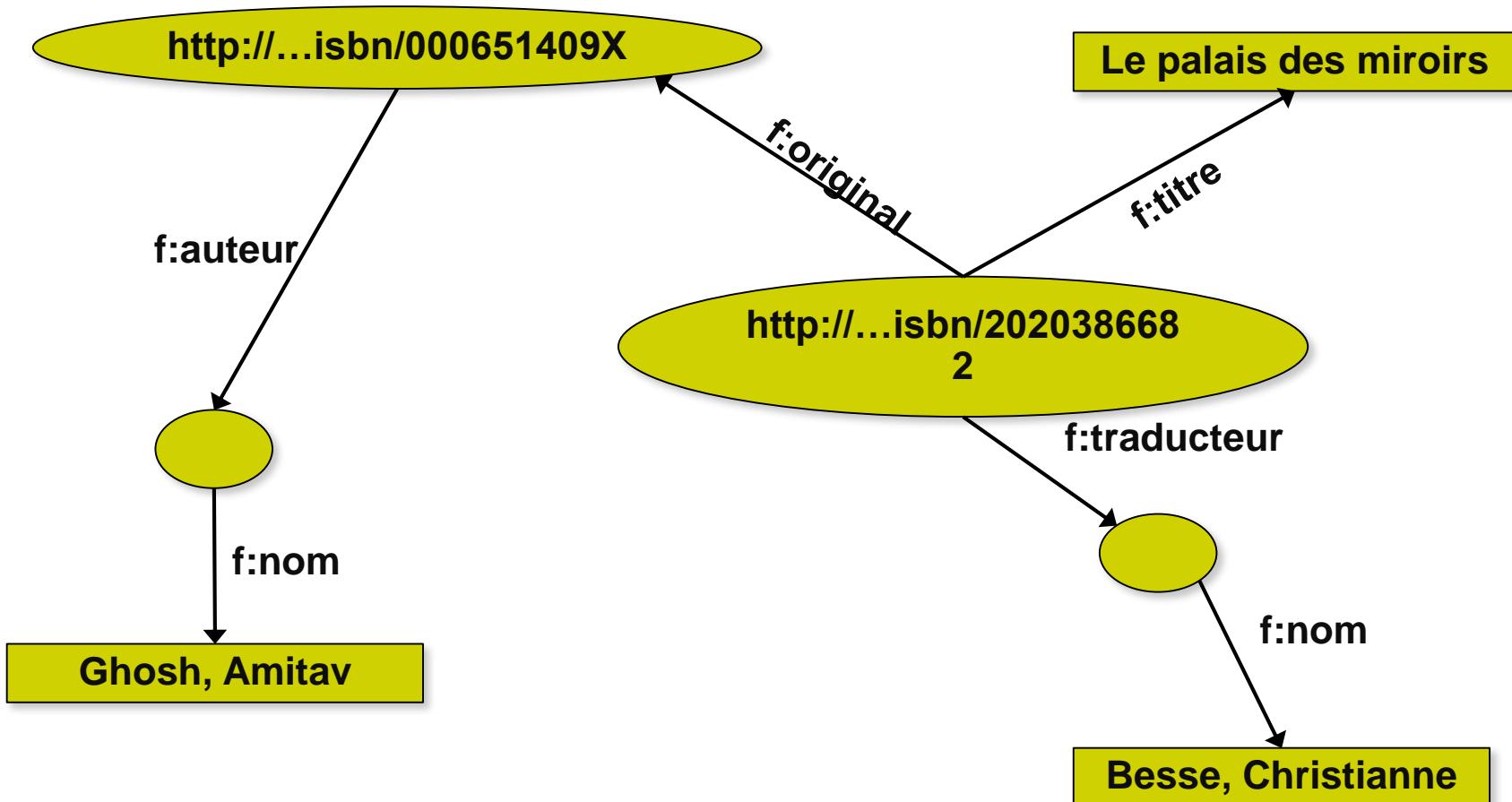
# Same book in French...



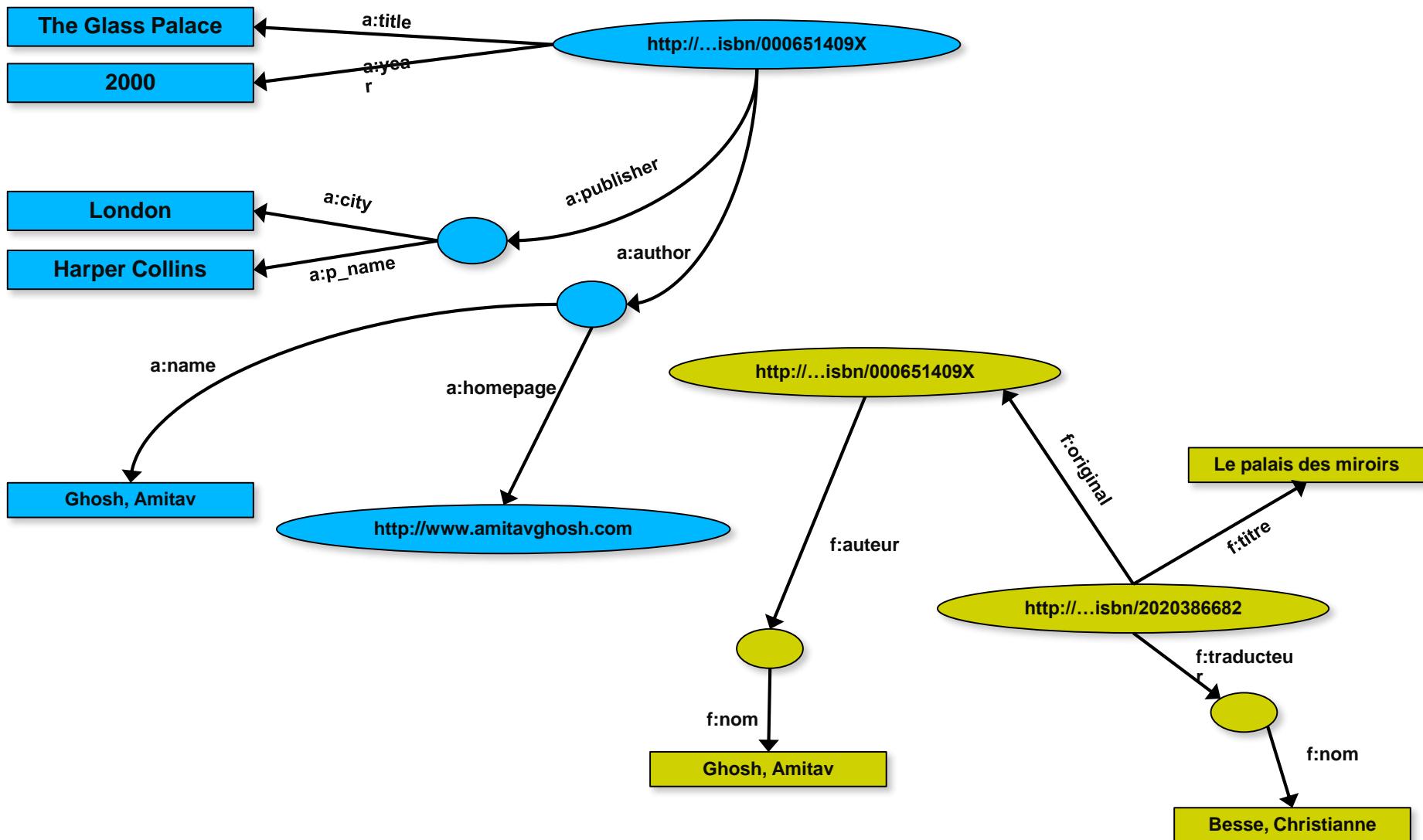
# Another bookstore data (dataset “F”)

| 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  |                       |            |                     |

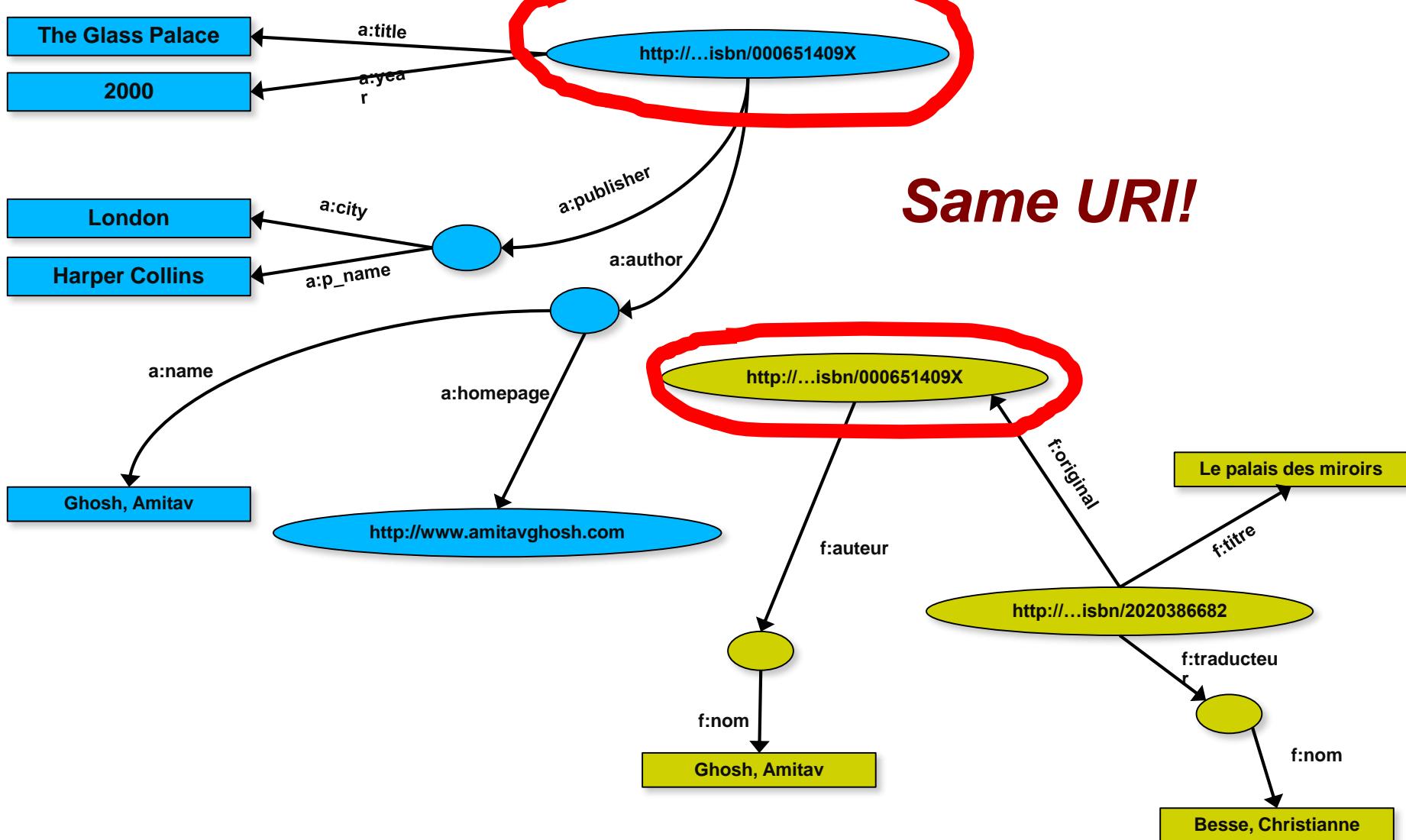
# 2<sup>nd</sup>: export your second set of data



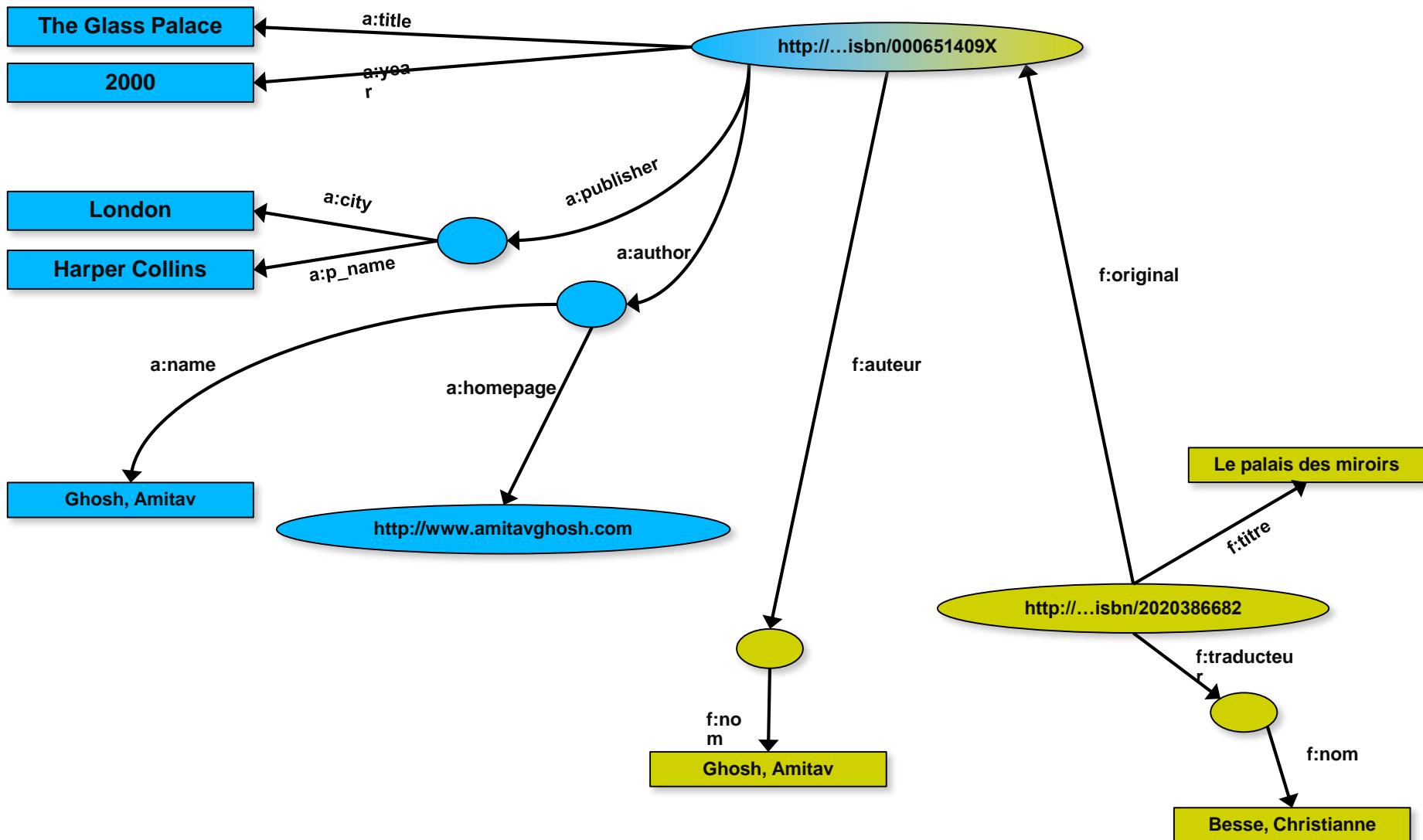
# 3<sup>rd</sup>: start merging your data



# 3<sup>rd</sup>: start merging your data (cont)

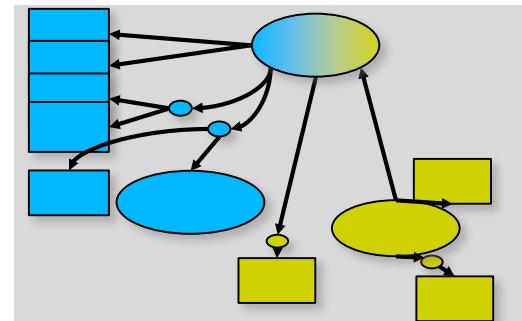


# 3<sup>rd</sup>: start merging your data



# Start making queries...

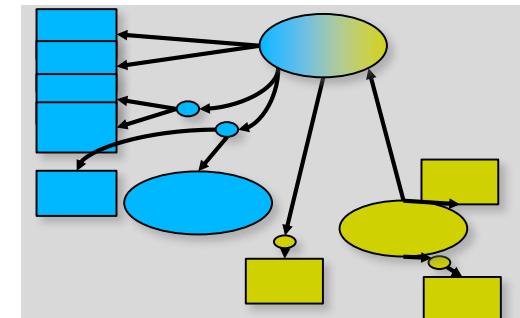
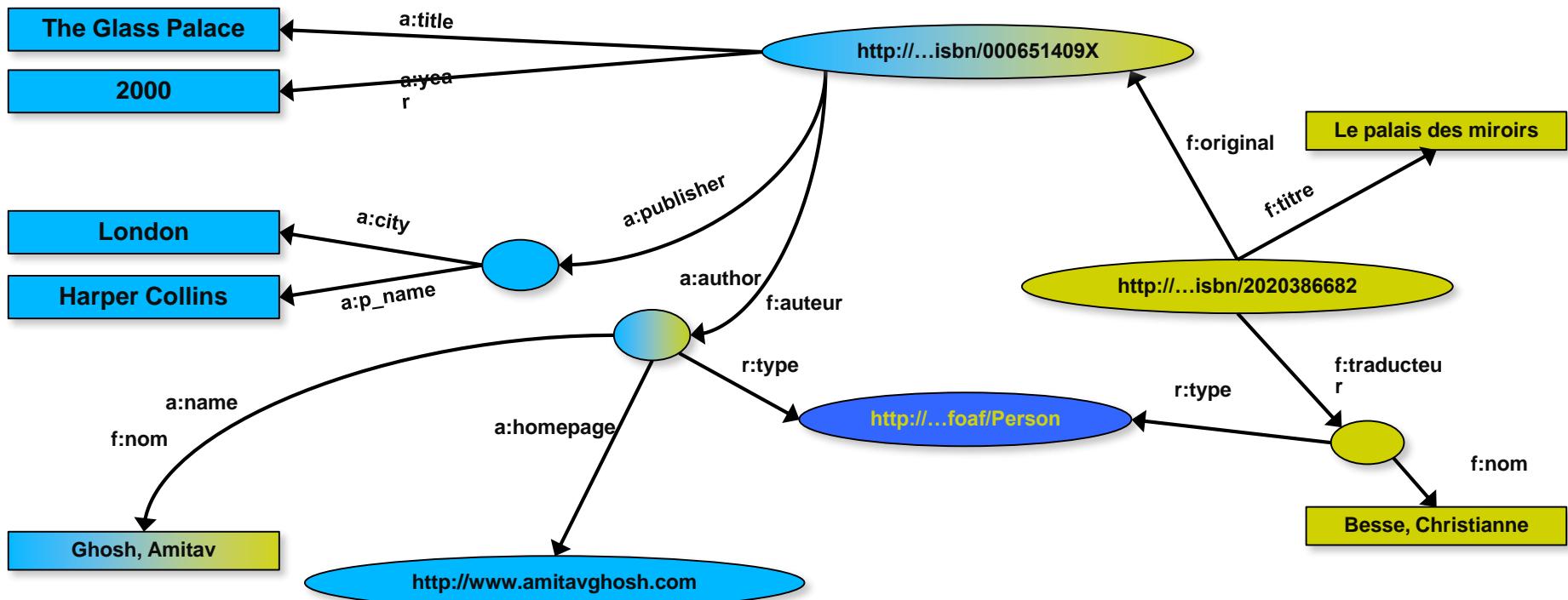
- User of data “F” can now ask queries like:
  - “give me the title of the original”
    - well, ... « donne-moi le titre de l’original »
- This information is not in the dataset “F”...
- ...but can be retrieved by merging with dataset “A”!



# 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



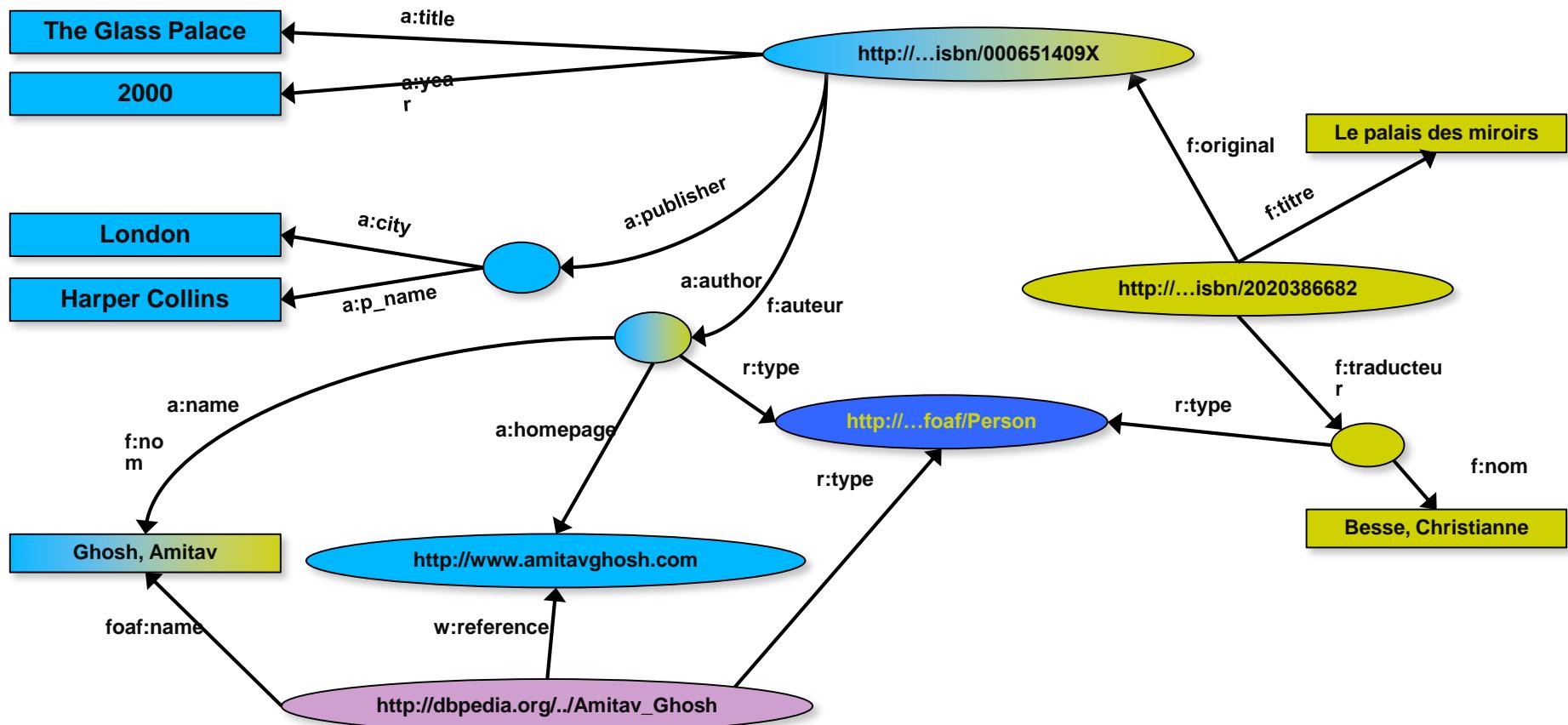
# Start making richer queries!

- 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”

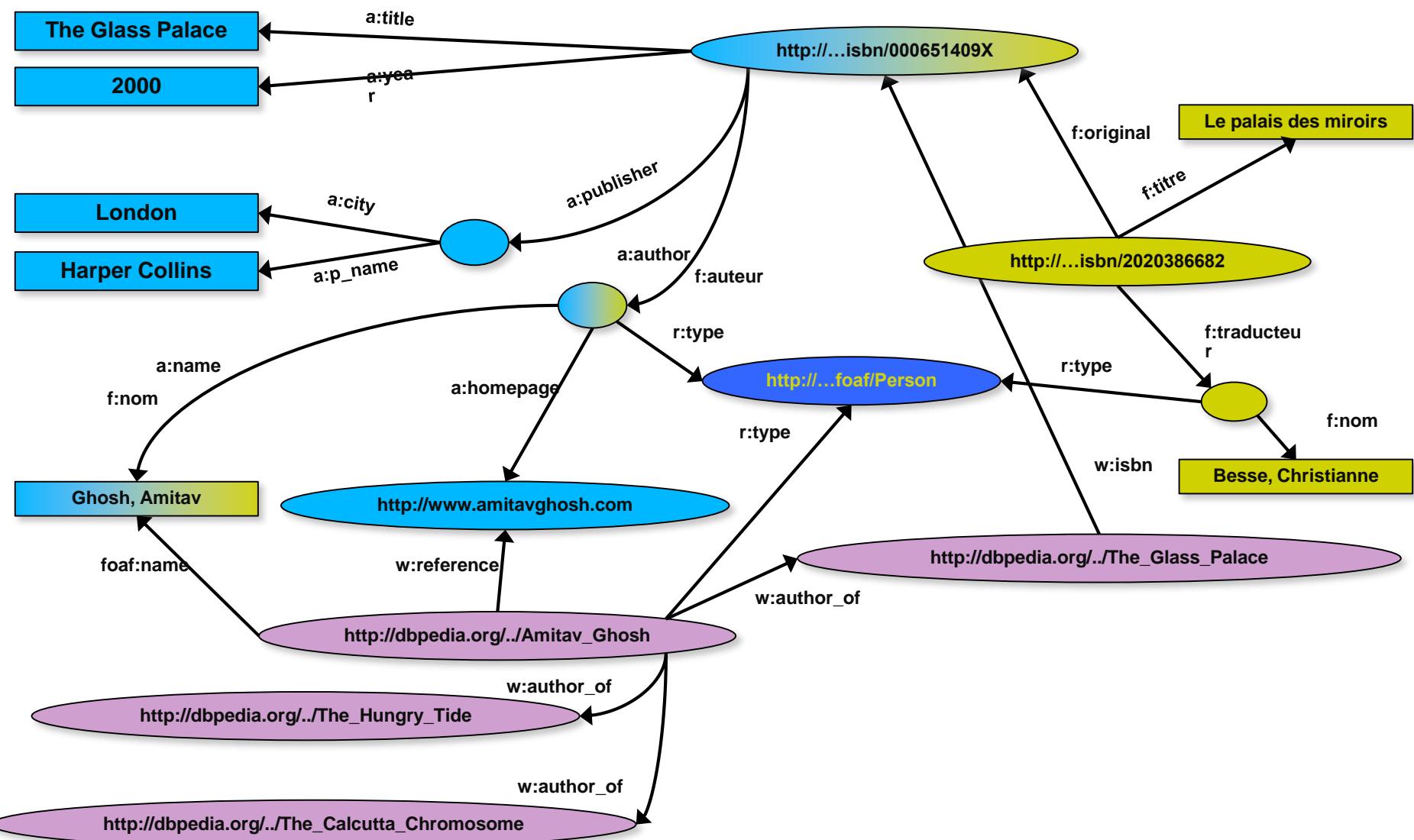
# Combine with different datasets

- 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...

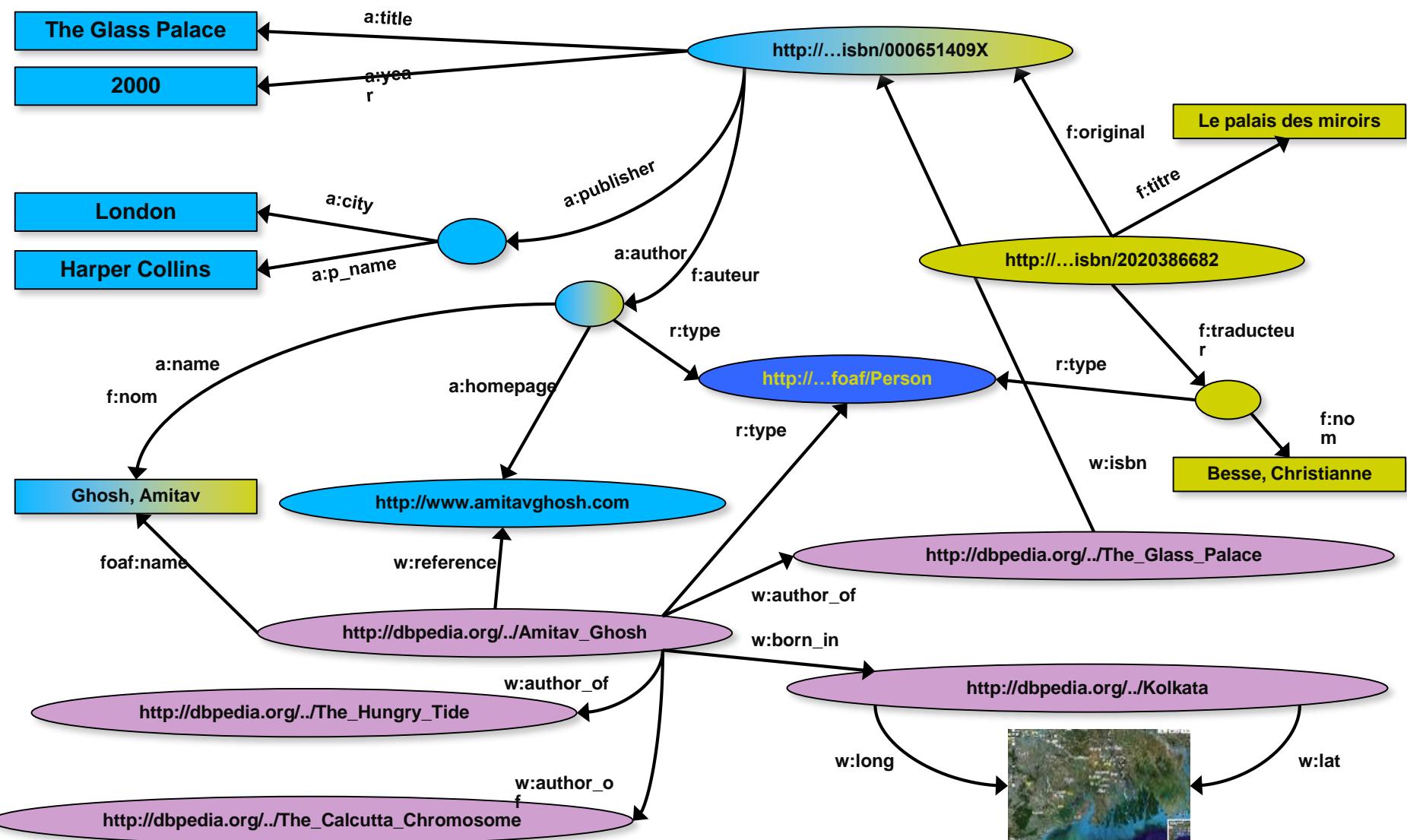
# Merge with Wikipedia data



# Merge with Wikipedia data



# Merge with Wikipedia data

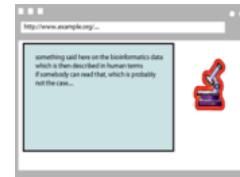
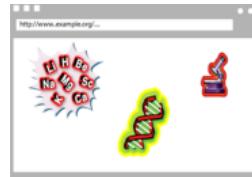


# It could become even more powerful

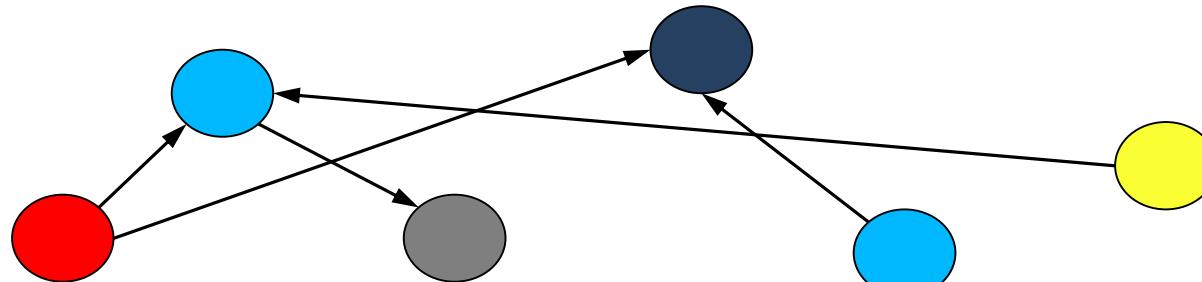
- We could add extra knowledge to the merged datasets
  - e.g., a full classification of various types of library data
  - geographical information etc
- This is where ontologies come in
  - ontologies sets can be relatively simple and small, or huge, or anything in between...
- Even more powerful queries can be asked as a result

# What did we do?

Applications

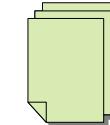
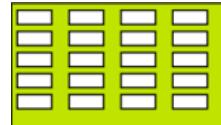
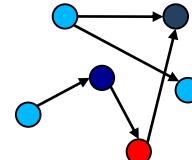
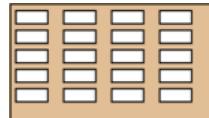
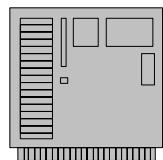


Manipulate  
Query  
...



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/>



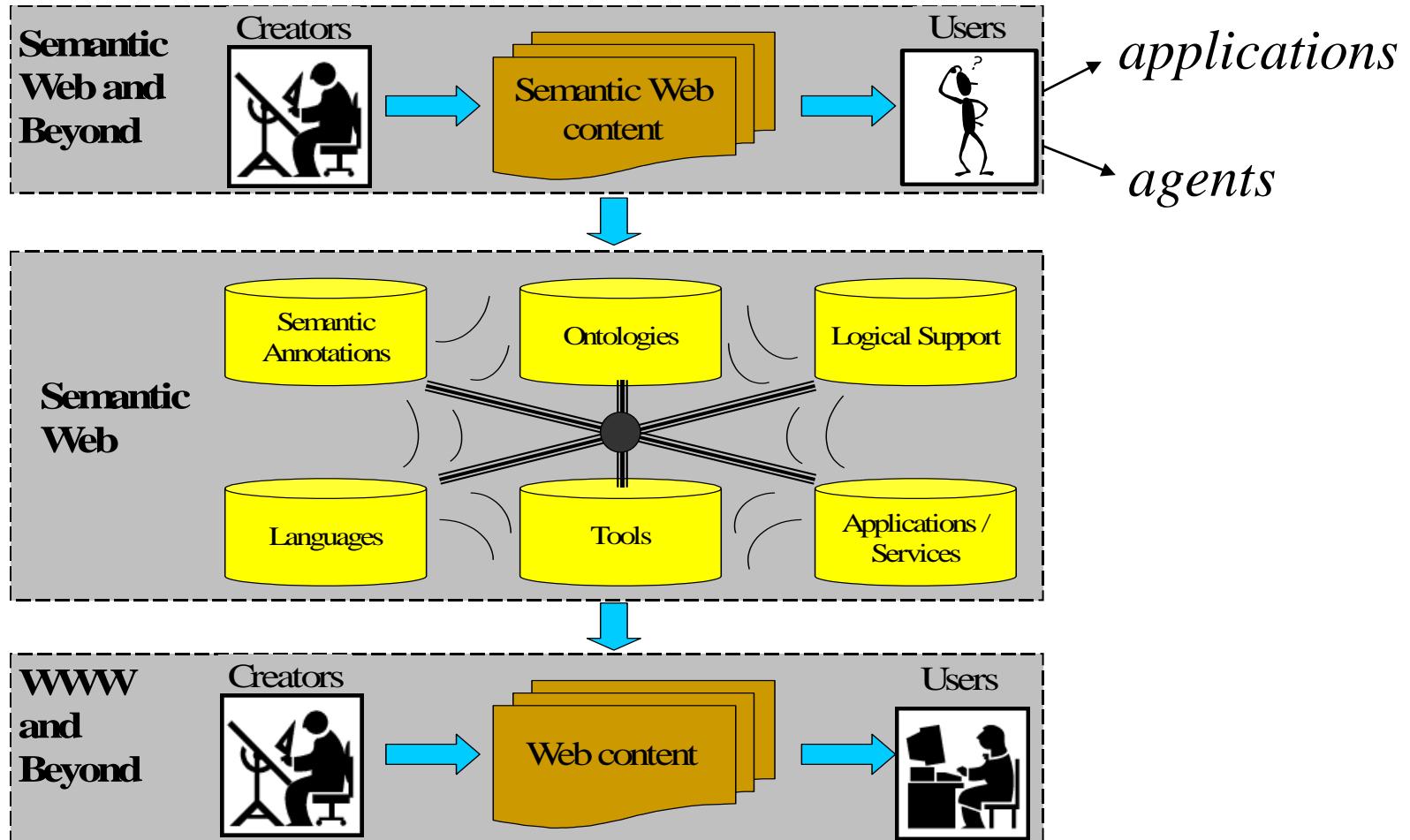
# Semantic Web

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.

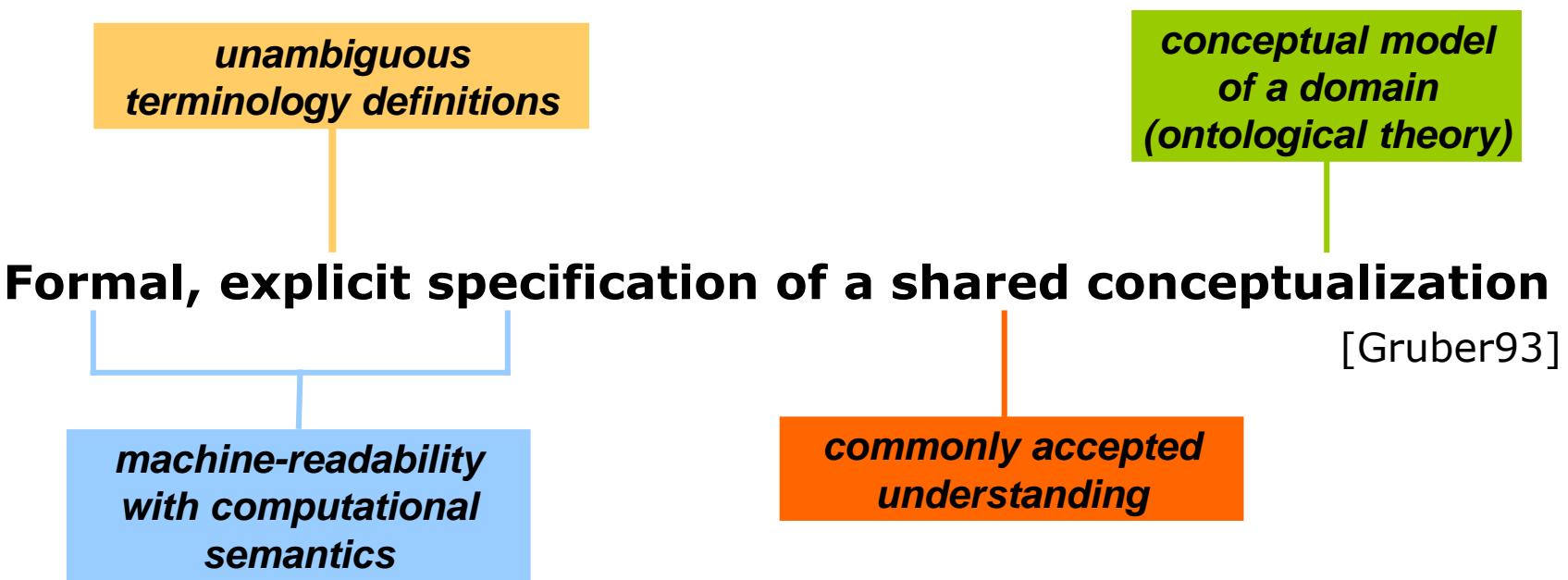
# Semantic Web – New Users



# 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



# Structure of an Ontology

Ontologies typically have two distinct components:

Names for important concepts in the domain

- **Elephant** is a concept whose members are a kind of animal
  - **Herbivore** is a concept whose members are exactly those animals who eat only plants or parts of plants
  - **Adult\_Elephant** is a concept whose members are exactly those elephants whose age is greater than 20 years
- Background knowledge/constraints on the domain
- **Adult\_Elephants** weigh at least 2,000 kg
  - All **Elephants** are either **African\_Elephants** or **Indian\_Elephants**
  - No individual can be both a **Herbivore** and a **Carnivore**

# Ontology Example

## Concept

conceptual entity of the domain

## Attribute

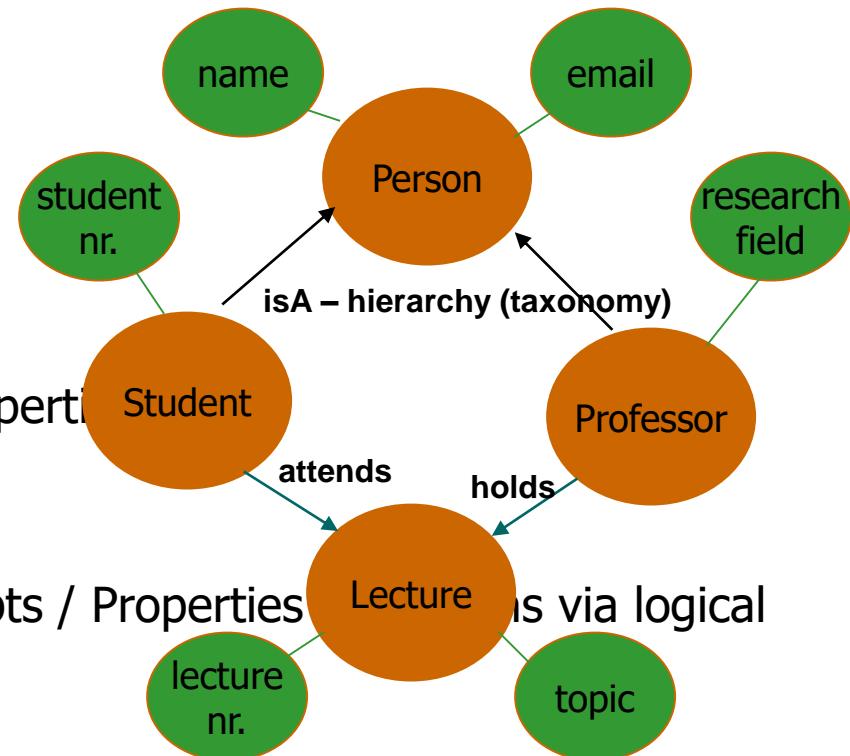
property of a concept

## Relation

relationship between concepts or properties

## Axiom

coherent description between Concepts / Properties  
expressions



$\text{holds}(\text{Professor}, \text{Lecture}) \Rightarrow \text{Lecture}.\text{topic} \in \text{Professor}.\text{researchField}$

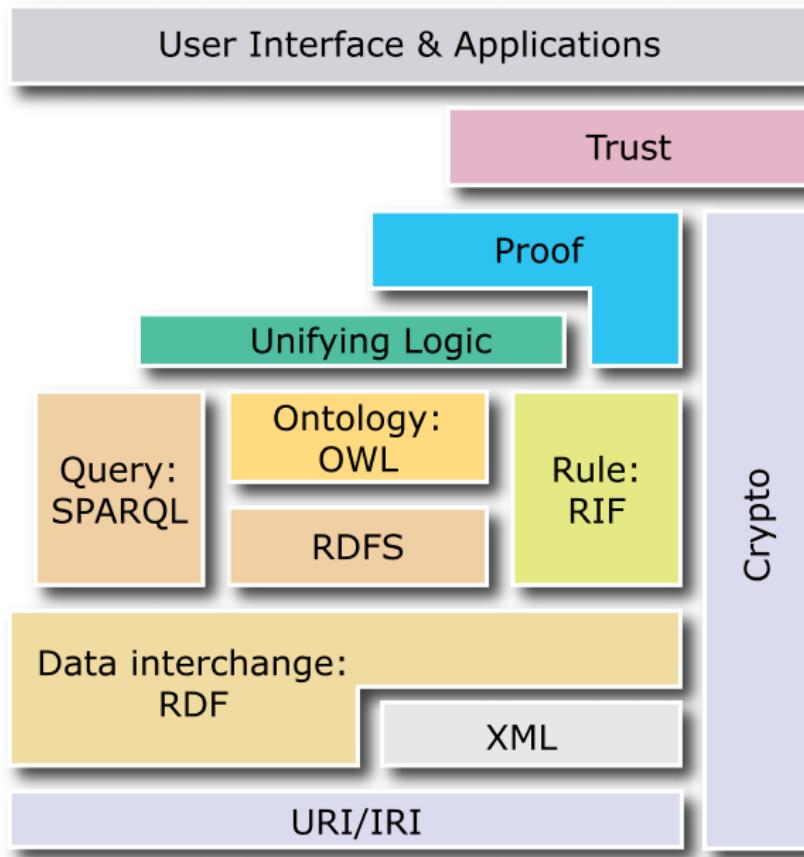
# Ontology Elements

- Concepts (classes) + their hierarchy
- Concept properties (slots/attributes)
- Property restrictions (type, cardinality, domain)
- Relations between concepts (disjoint, equality)
- Instances

# 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

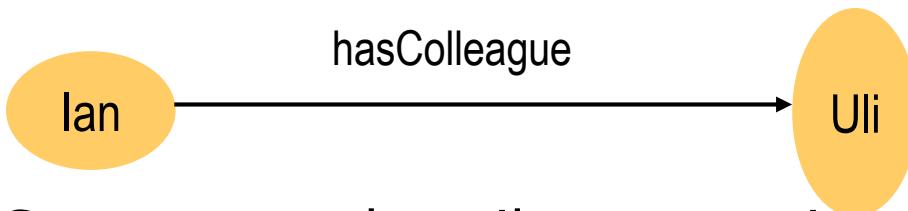
- RDFS extends RDF with “schema vocabulary”, e.g.:
  - Class, Property
  - type, subClassOf, subPropertyOf
  - range, domain

# The RDF Data Model

- Statements are  
<subject, predicate, object> triples:

<Ian, hasColleague, Uli>

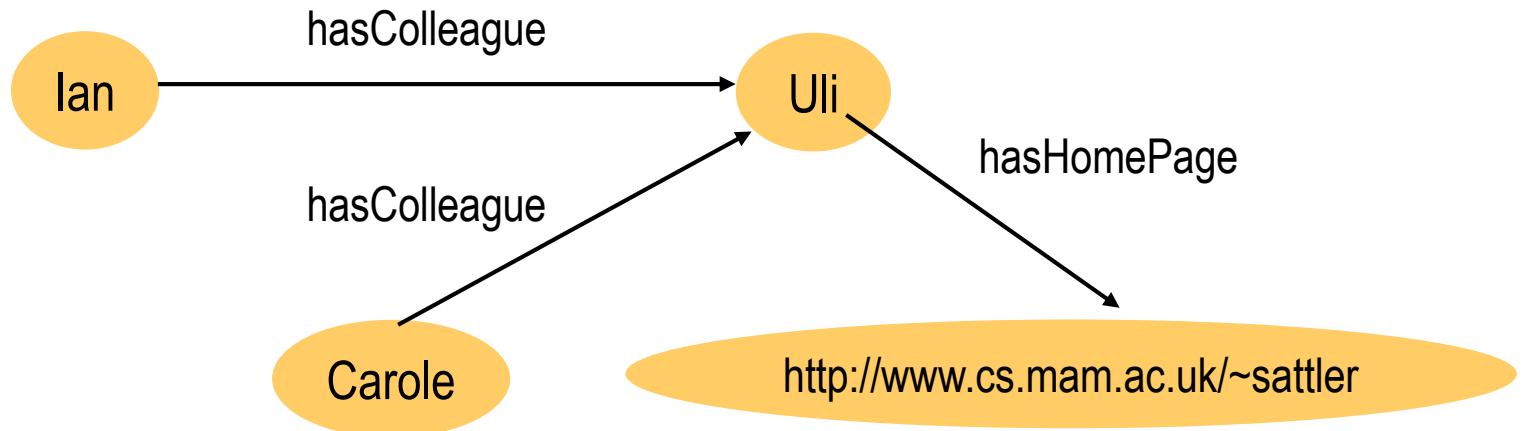
- Can be represented as a graph:



- Statements describe properties of resources
- A resource is any object that can be pointed to by a URI:
  - a document, a picture, a paragraph on the Web;
  - <http://www.cs.man.ac.uk/index.html>
  - isbn://5031-4444-3333
  - ...
- Properties themselves are also resources (URIs)

# Linking Statements

- The subject of one statement can be the object of another
- Such collections of statements form a directed, labelled graph



# 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

OWLClasses Properties Forms Individuals Ontology

Class Hierarchy

- :THING
- Person
  - Woman
  - Man

V C

**Woman** (type=owl:Class)

| Name  | Labels | SameAs | Different | Annotations    |
|-------|--------|--------|-----------|----------------|
| Woman |        |        |           | Property Value |

Documentation

**Properties at Class**

| Name     | Type     | Cardinality | Other Facets  |
|----------|----------|-------------|---------------|
| isWifeOf | Instance | single      | classes={Man} |

**Restrictions**

| Property | Restriction | Filler |
|----------|-------------|--------|
|          |             |        |

**Definition**

**Disjoint classes**

- Man

V C

# 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

**isHusbandOf** (type=owl:ObjectProperty)

Name Labels SameAs DifferentFrom

isHusbandOf

Documentation

Annotations

| Property | Value |
|----------|-------|
|----------|-------|

Cardinality

required  at least

multiple  at most



Instance

Some Values From

Domain defined

Domain

**Woman**

**Man**

Inverse Property

Symmetric  AnnotationProperty

Transitive  InverseFunctional

**isWifeOf**

# Ontology Languages Summary

- 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

# 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

# Semantic Search

**Renewable Energy Glossary**

- energy (12)
- wind (7)
- windpower (6)
- wind farms (5)
- wind turbines (5)
- generators (4)
- photovoltaic power (4)
- ocean energy (2)

[More](#)

**Climate Compatible Development Glossary**

- renewable energies (6)
- wind power (6)
- projects (4)
- economic cost (3)
- fossil energy (3)
- greenhouse gas emissions (3)

wind farms

 Show tagged content only

**Wind farms**



A wind farm is a group of wind turbines in the same location used for production of ... For example, Gansu Wind Farm, the largest wind farm in the world, has several thousand turbines ...

[WIND FARMS](#)

[SHOW PARENT](#)

**Basics of Wind Energy**



What is a wind farm ... The turbines in a wind farm are connected so the electricity they generate can travel from the wind farm to the power grid ...

[ELECTRICITY GENERATION](#) [ENERGY](#) [GRIDS](#) [WIND](#) [WIND POWER](#) [WIND TURBINES](#) [WINDPOWER](#)

[SHOW PARENT](#)

**Wind\_test**



Wind farms consist of many individual wind turbines which are connected to the electric ... Small onshore wind farms can feed some energy into the grid or provide electricity to ...

[ECONOMIC COST](#) [ELECTRICITY GENERATION](#) [ENERGY](#) [WIND](#) [WIND FARMS](#) [WIND POWER](#) [WIND TURBINES](#) [WINDPOWER](#)

[SHOW PARENT](#)

**France Gears Up for Floating Wind**



This project intends to quickly put floating wind farms on the map as a competitive energy ... Engie has stated: "Floating wind turbines are an up-and-coming technology that can be ...

[SHOW PARENT](#)

[SHOW PARENT](#)

**wind farms**



**Description**

A group of wind turbines interconnected to a common power provider system through a system of transformers, distribution lines, and (usually) one substation. Operation, control, and maintenance functions are often centralized through a network of computerized monitoring systems, supplemented by visual inspection.

**Also known as**

wind parks, wind power plants, wind power stations, windfarm

**More general**

windpower concepts

**More specific**

grid-connected wind power systems, on-shore windparks, off-shore windparks

**Related**

wind turbines, wind power capacity installed, wind feed-in tariffs, stand-alone wind power turbines

# Research in Life Sciences

As a researcher in pharmaceutical industry, I want to plan new experiments more efficiently. I want to know what's already available. I'm interested in former experiments where

- certain genes were tested
- under specific treatment conditions
- in a target therapeutic area
- with help from categorisation systems like 'disease hierarchies'

→ Linking Structured to Unstructured Data and to Industry Knowledge Graphs

UniProt, ChEMBL



Experiments Documentation



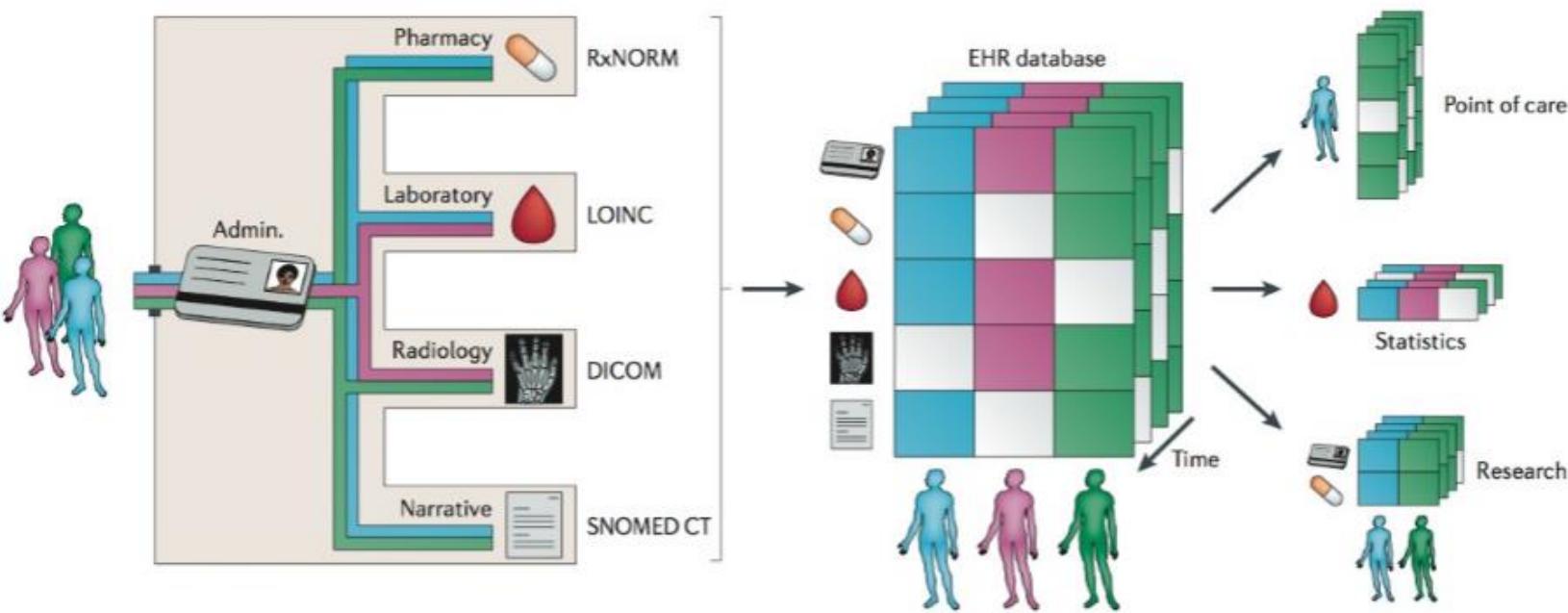
MeSH



DrugBank

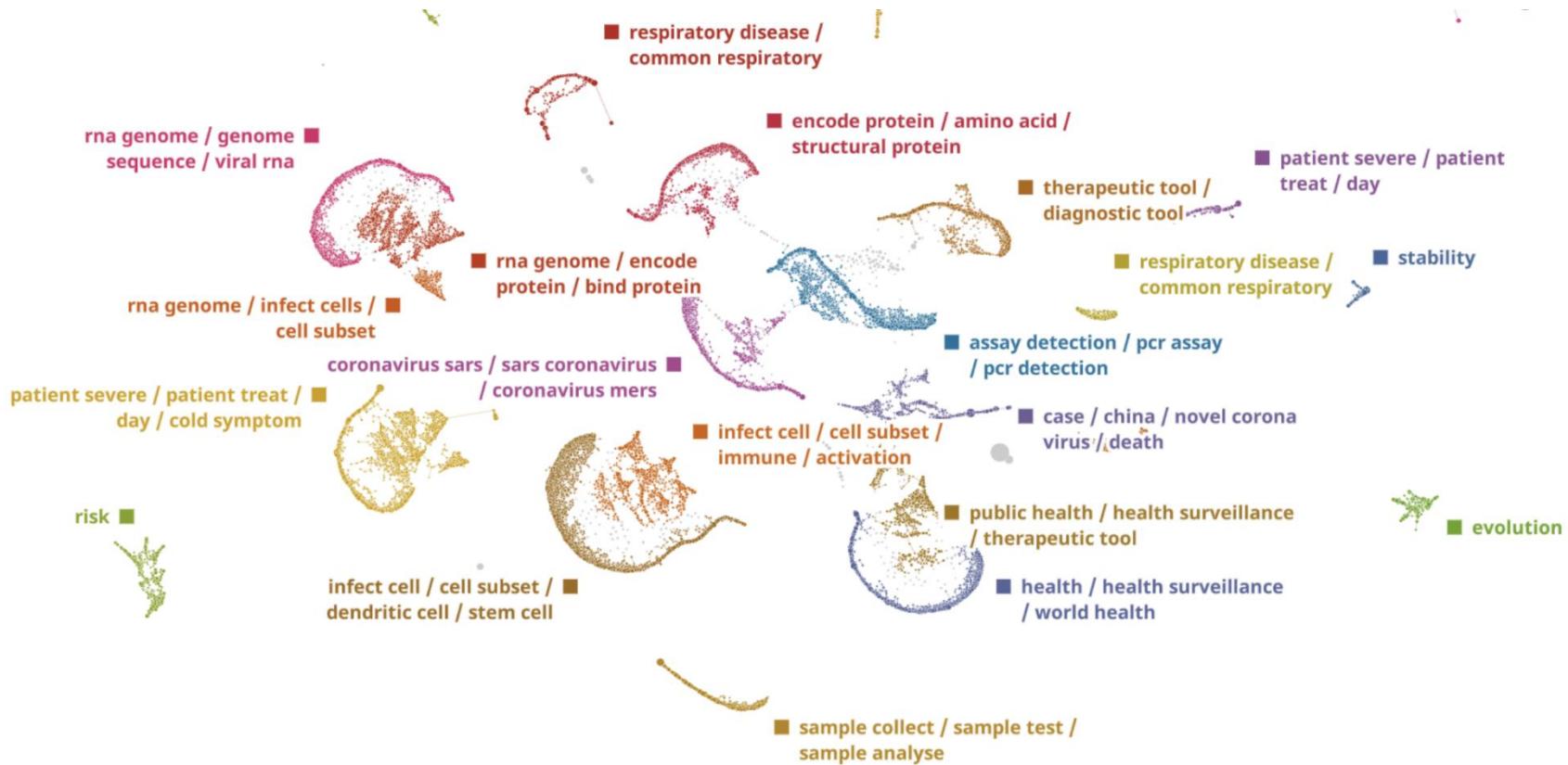


# Data and text mining of electronic health records



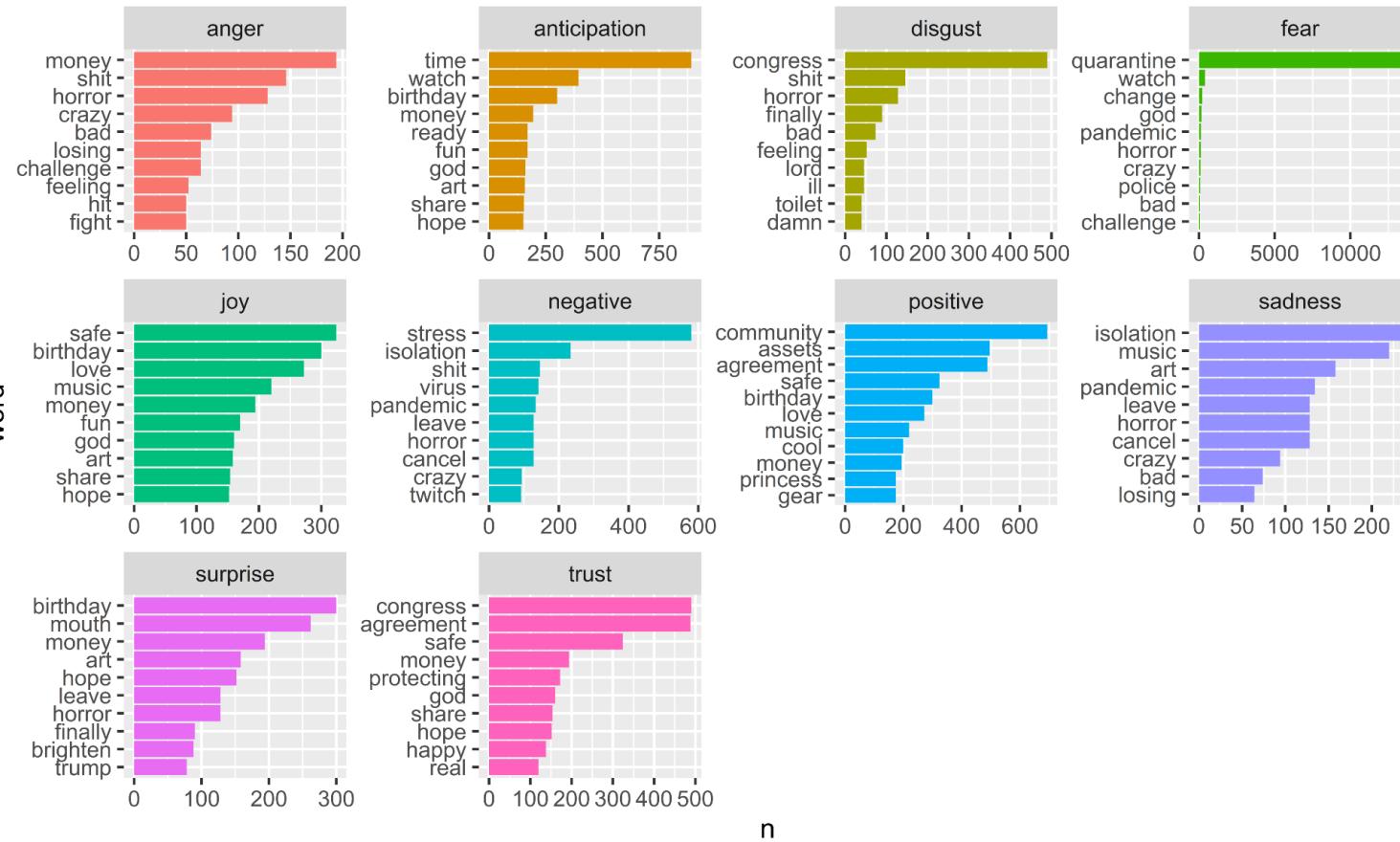
<https://www.slideshare.net/larsjuhljensen/data-and-text-mining-of-electronic-health-records-55027555>

# Covid 19 datasets

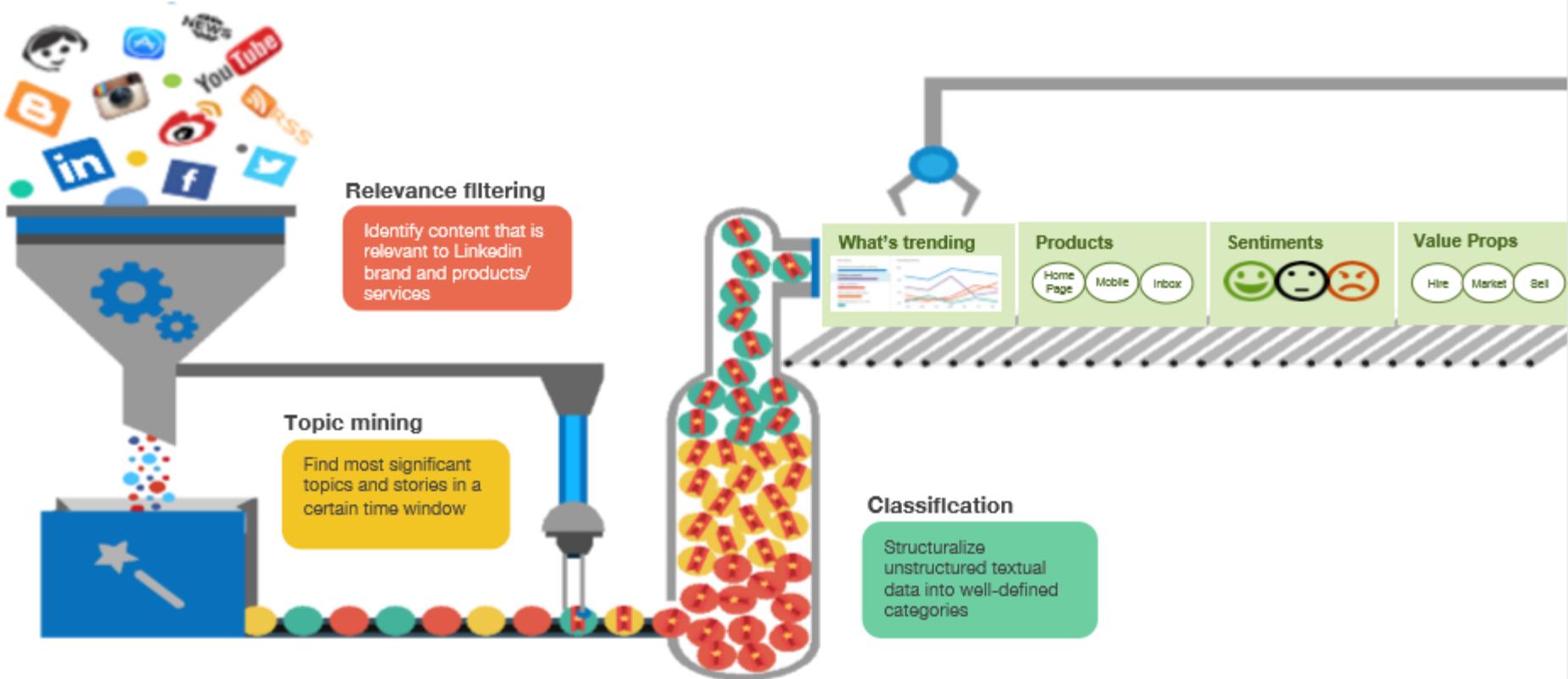


# Sentiment Analysis

Word frequency based on emotion



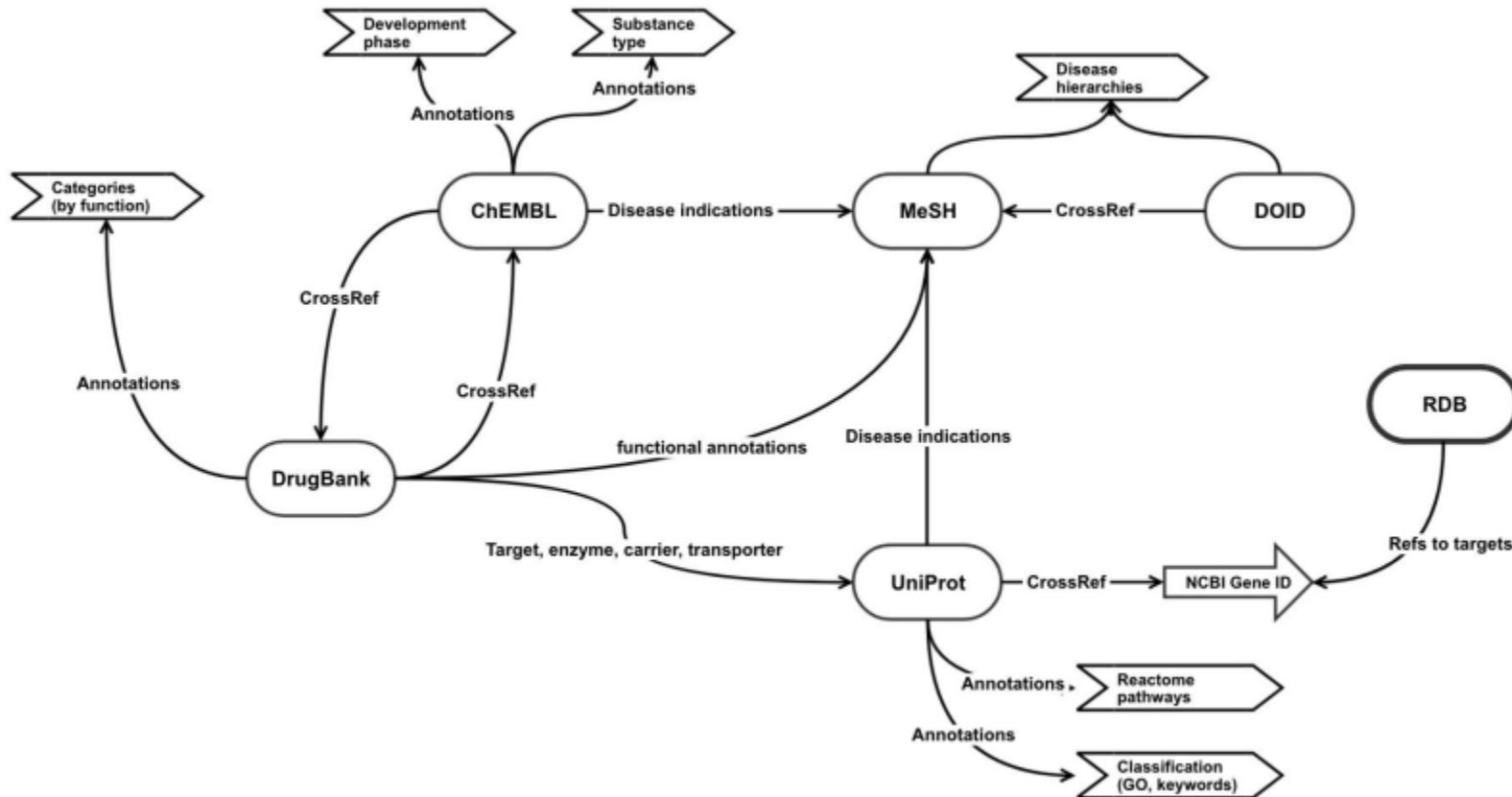
# Voices: drive actionable intelligence from member voices...



# 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

# Reusing Ontologies



# **State of art**

- Ontology Learning
- Knowledge graph
- Linked open data

# 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

# Knowledge is the key



# Knowledge Graph

- **ontology + data = knowledge graph**
- Information extraction techniques for performing named entity recognition. Another subfield which has gained much interest from the community is keywords extraction.
- A **knowledge graph** is a way of storing data that resulted from an information extraction task. Many basic implementations of knowledge graphs make use of a concept we call **triple**, that is a set of three items(a subject, a predicate and an object)

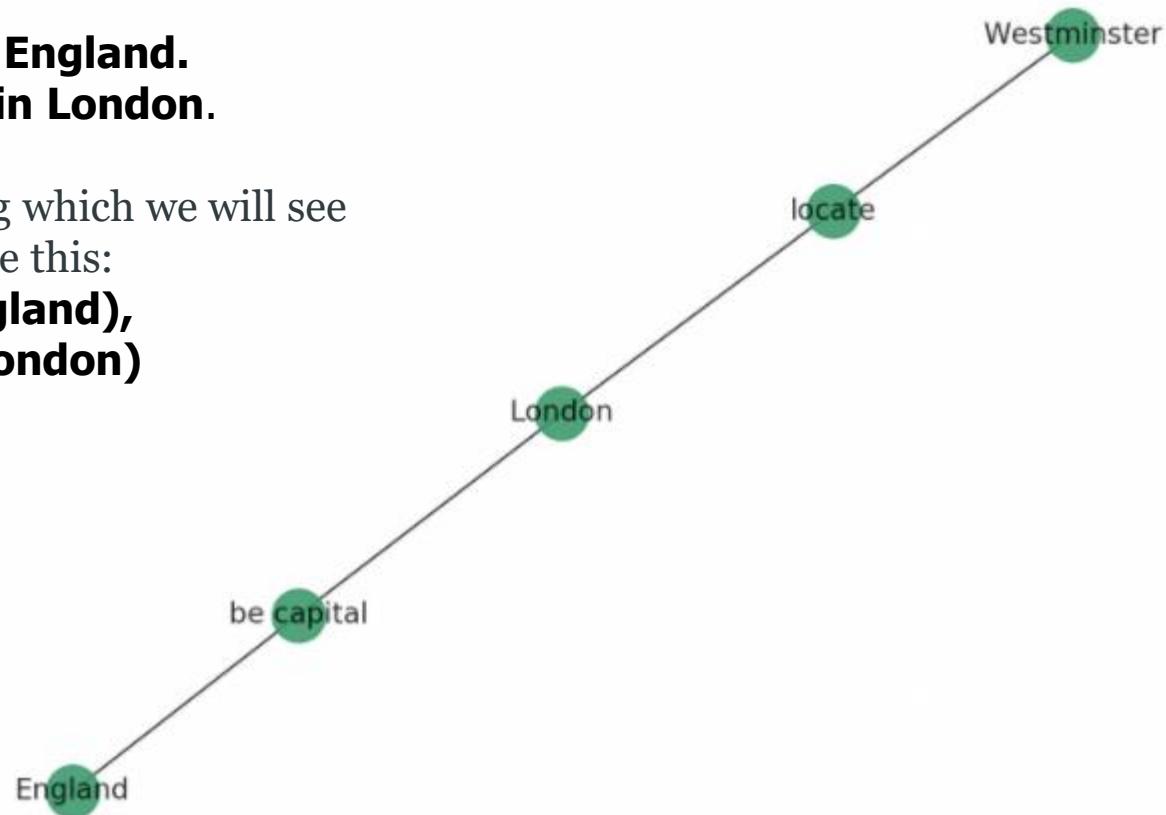
# Example

**London is the capital of England.**

**Westminster is located in London.**

After some basic processing which we will see later, we would 2 triples like this:

**(London, be capital, England),  
(Westminster, locate, London)**

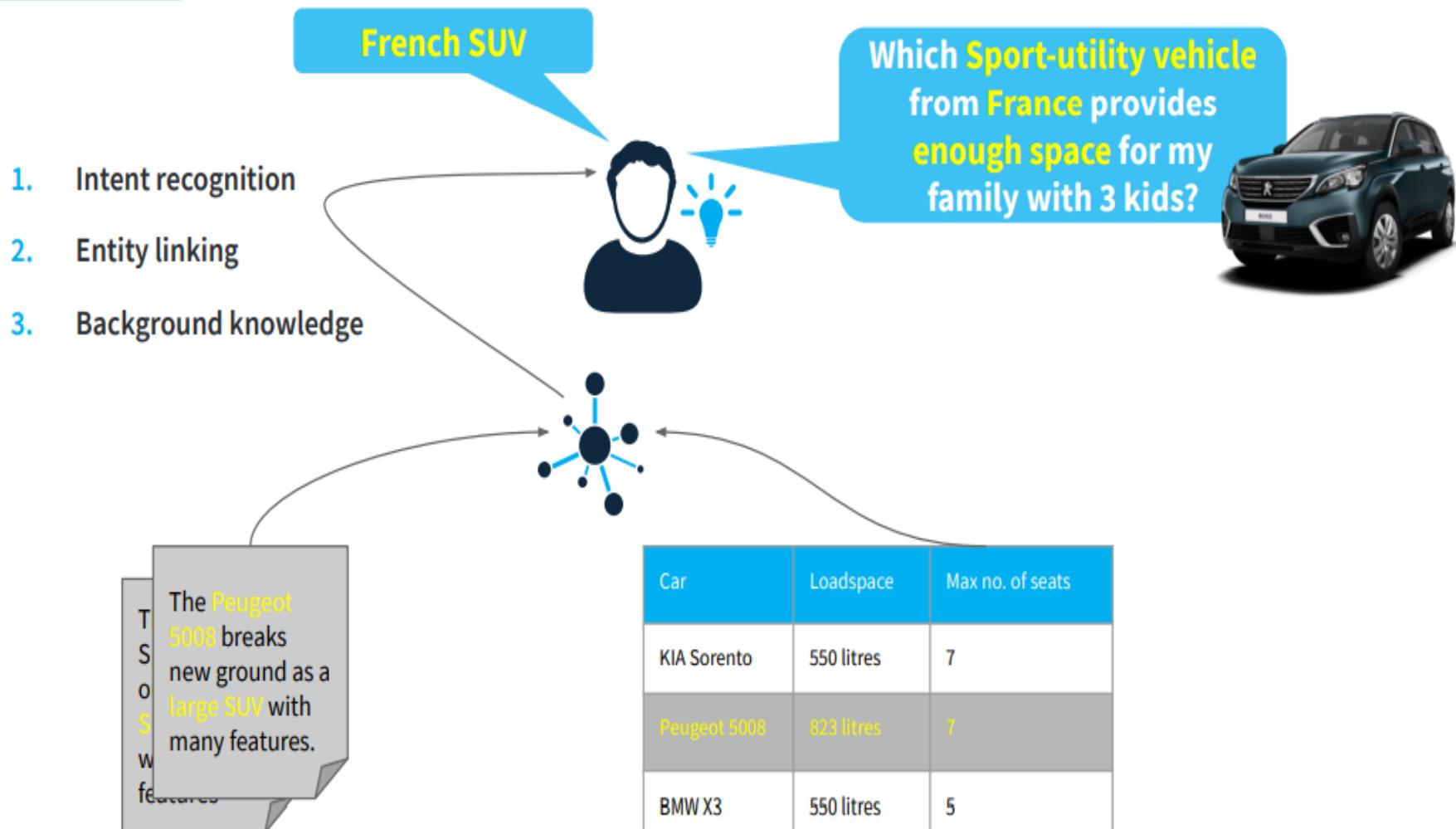


Knowledge graph representation

# Building Knowledge Graph

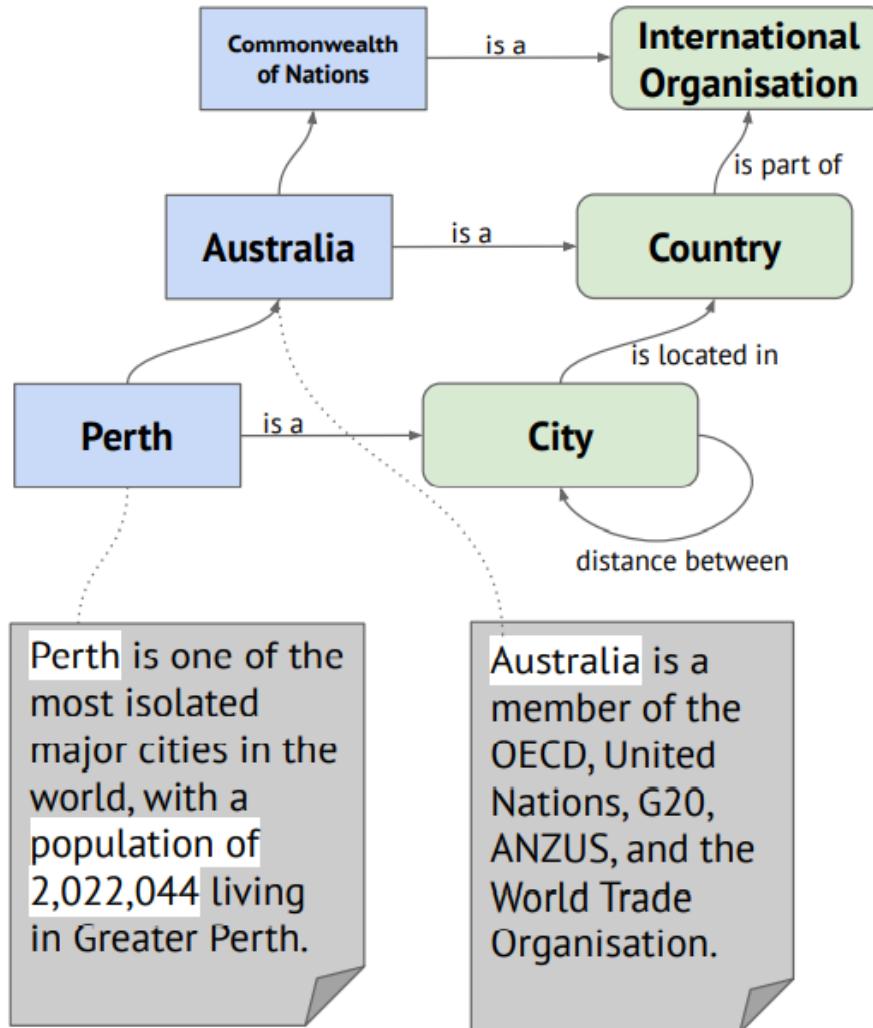
- Identifying the entities and the relation between them is not a difficult task for us. However, **manually building a knowledge graph is not scalable.**
- Nobody is going to go through thousands of documents and extract all the entities and the relations between them!
- **To build a knowledge graph from the text, it is important to make our machine understand natural language.**
- This can be done by using NLP techniques such as sentence segmentation, dependency parsing, parts of speech tagging, and entity recognition

Do machines understand user intent? Do they have enough context?



# Artificial “Intelligence”

Background knowledge is key



## Support complex Q&A:

Which cities located in the Commonwealth of Nations have a population of more than 2 mio. people?

## Avoid illogical answers:



# Knowledge is the key

- **Semantic descriptions** of entities and their **relationships**
  - Uses a **knowledge representation formalism**  
(Focus here: RDF, RDF-Schema, OWL)
  - **Entities**: real world objects (things, places, people) and abstract concepts (genres, religions, professions)
  - **Relationships**: graph-based data model where relationships are first-class
  - **Semantic descriptions**: types and properties with a well-defined meaning (e.g. through an ontology)
  - Possibly axiomatic knowledge (e.g. rules) to support automated reasoning
-

# Knowledge Graph

## Applications

- Semantic Search
- Question Answering
- Analytics
- Dashboards
- Knowledge Sharing
- Knowledge Management

## Algorithms

- Inferencing
- Machine Learning
- Entity Recognition
- Disambiguation
- Text Understanding
- Recommendations

## Knowledge Graph



- Entities
- Relationships
- Semantic Descriptions

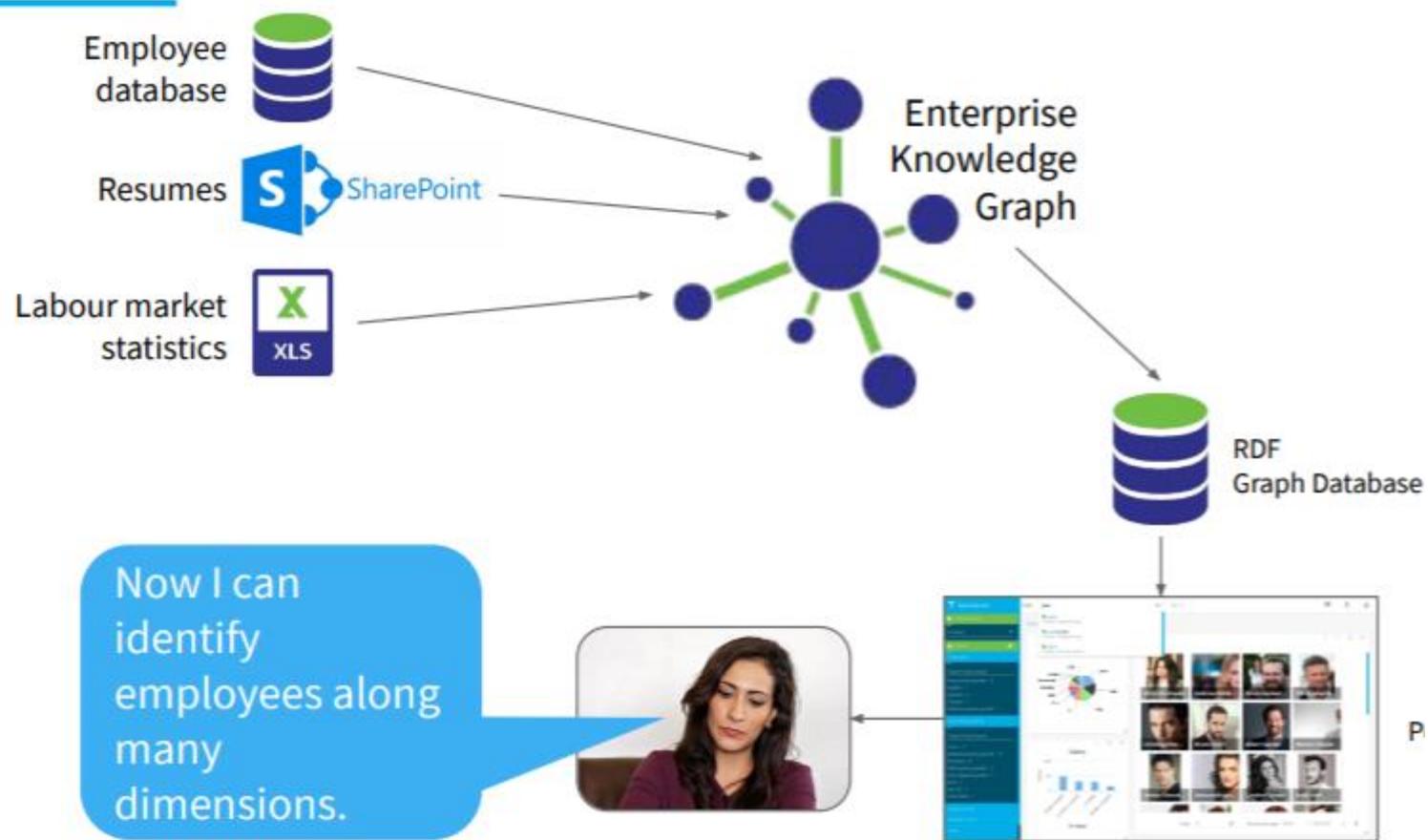
## Data Sources

Data Transformation, Integration  
Natural Language Processing

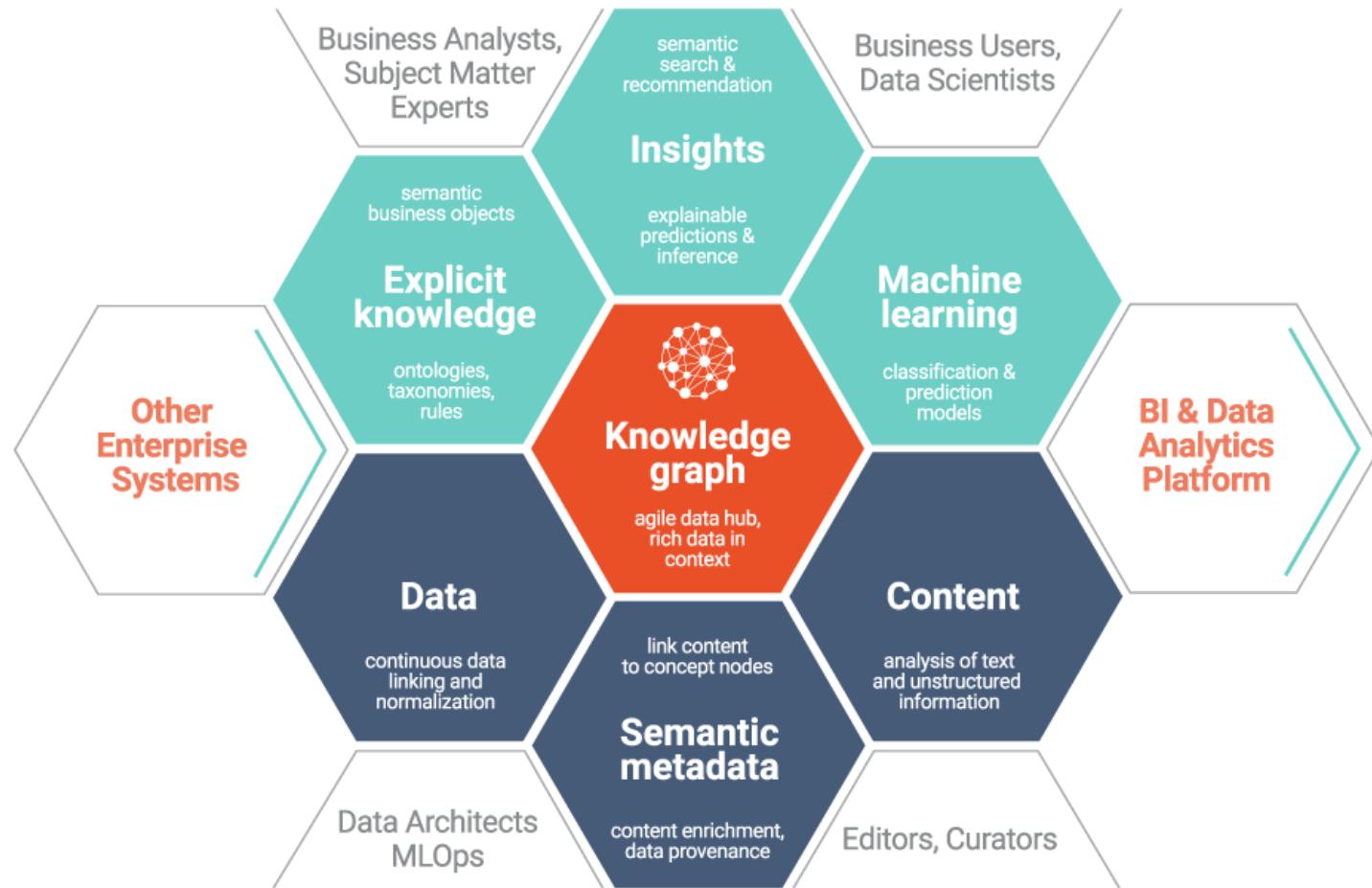


# Knowledge Graphs for **Data Integration & Analytics**

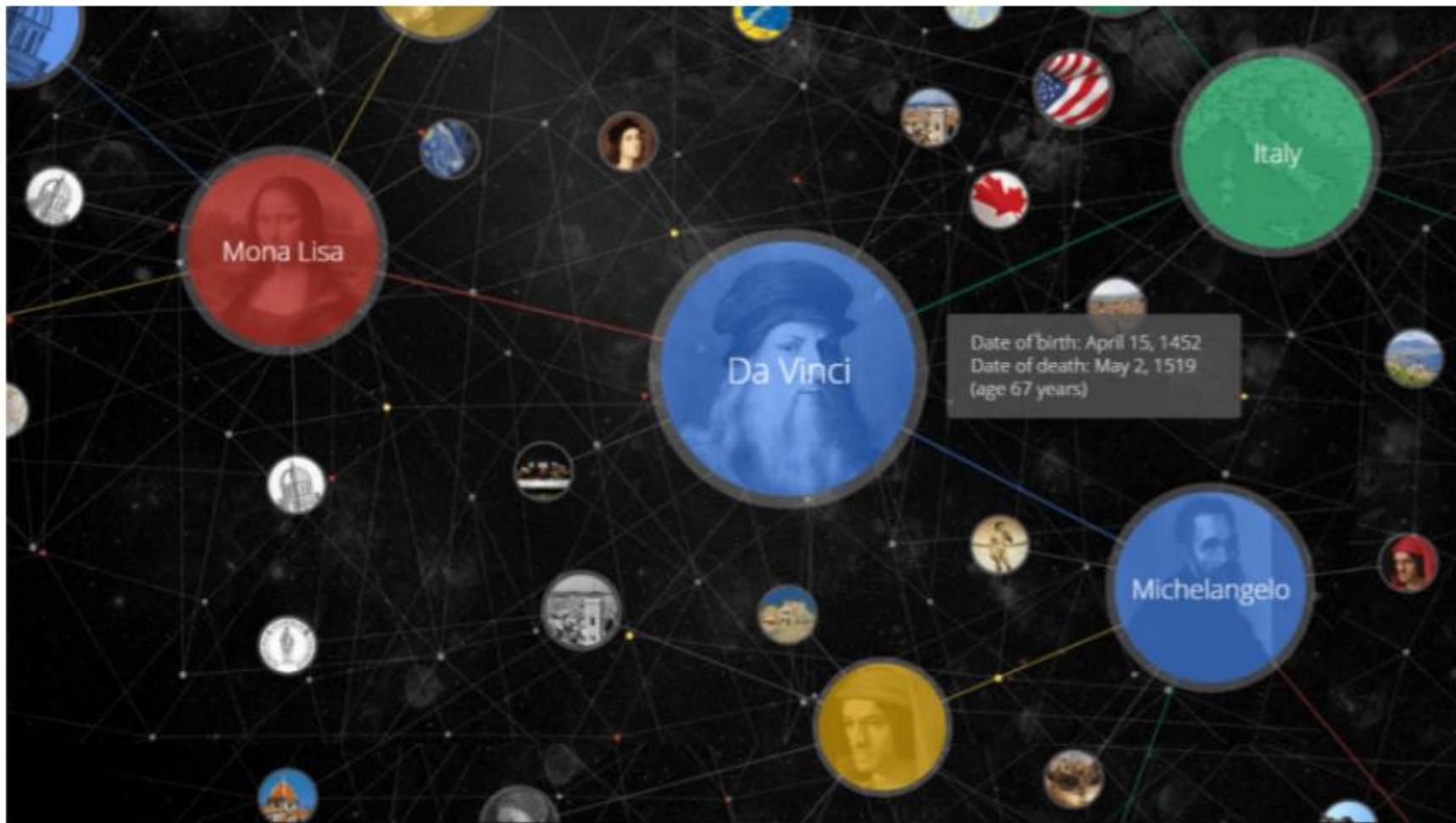
Metadata enrichment, linked data, text mining, entity-centric search, agile reporting



# Knowledge Graph



# Google Knowledge Graph



# Knowledge Graph Adaption

Largest change in market cap by company (2009 to 31 March 2018)

|    | Company name       | Location      | Industry          | Change in market cap<br>2009-2018 (\$bn) | Market cap 2018 (\$bn) |
|----|--------------------|---------------|-------------------|--|------------------------|
| 1  | Apple              | United States | Technology        | 757                                      | 851                    |
| 2  | Amazon.Com         | United States | Consumer Services | 670                                      | 701                    |
| 3  | Alphabet           | United States | Technology        | 609                                      | 719                    |
| 4  | Microsoft Corp     | United States | Technology        | 540                                      | 703                    |
| 5  | Tencent Holdings   | China         | Technology        | 483                                      | 496                    |
| 6  | Facebook           | United States | Technology        | 383(1)                                   | 464                    |
| 7  | Berkshire Hathaway | United States | Financial         | 358                                      | 492                    |
| —  | Alibaba            | China         | Consumer Services | 302(1)                                   | 470                    |
| 9  | JPMorgan Chase     | United States | Financials        | 275                                      | 375                    |
| 10 | Bank of America    | United States | Financials        | 263                                      | 307                    |

Known knowledge graph builders

Operator of Taobao and KG builder

Known KG builders

# 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  |  |
| - Type  | Municipality   |
| - Mayor   | Job Cohen <sup>[1]</sup> (PvdA)  |
| - Aldermen  | Lodewijk Asscher<br>Carolien Gehrels<br>Tjeerd Herrema<br>Maarten van Poelgeest<br>Marijke Vos |
| - Secretary   | Erik Gerritsen   |
| Area <small>[2][3]</small>  |  |
| - 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 <small>[4]</small>  | 2 m (7 ft)   |
| Population <small>(1 October 2008)[5][6]</small>                                    |  |
| - 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>                     |  |

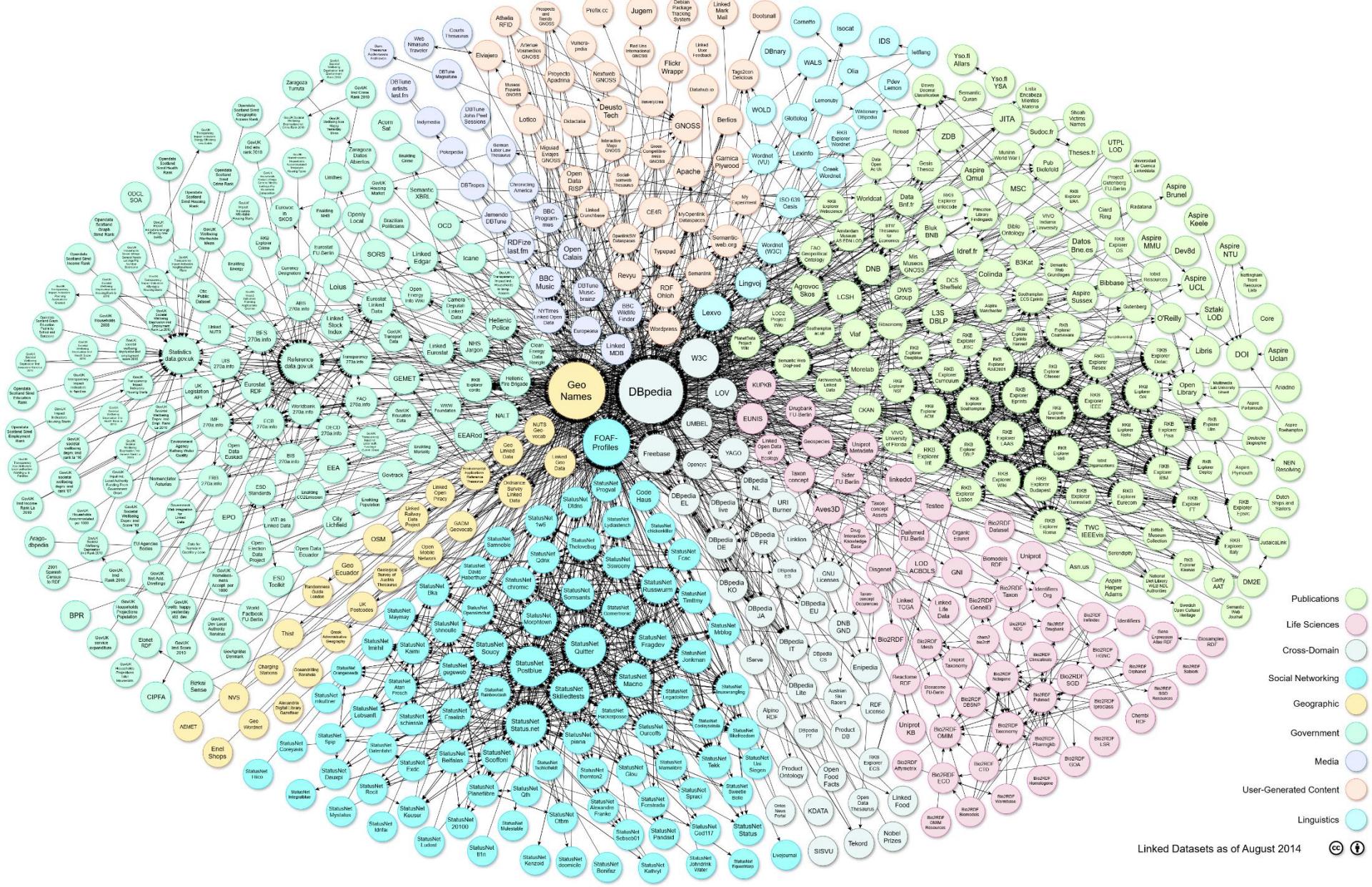
# Automatic links among open datasets

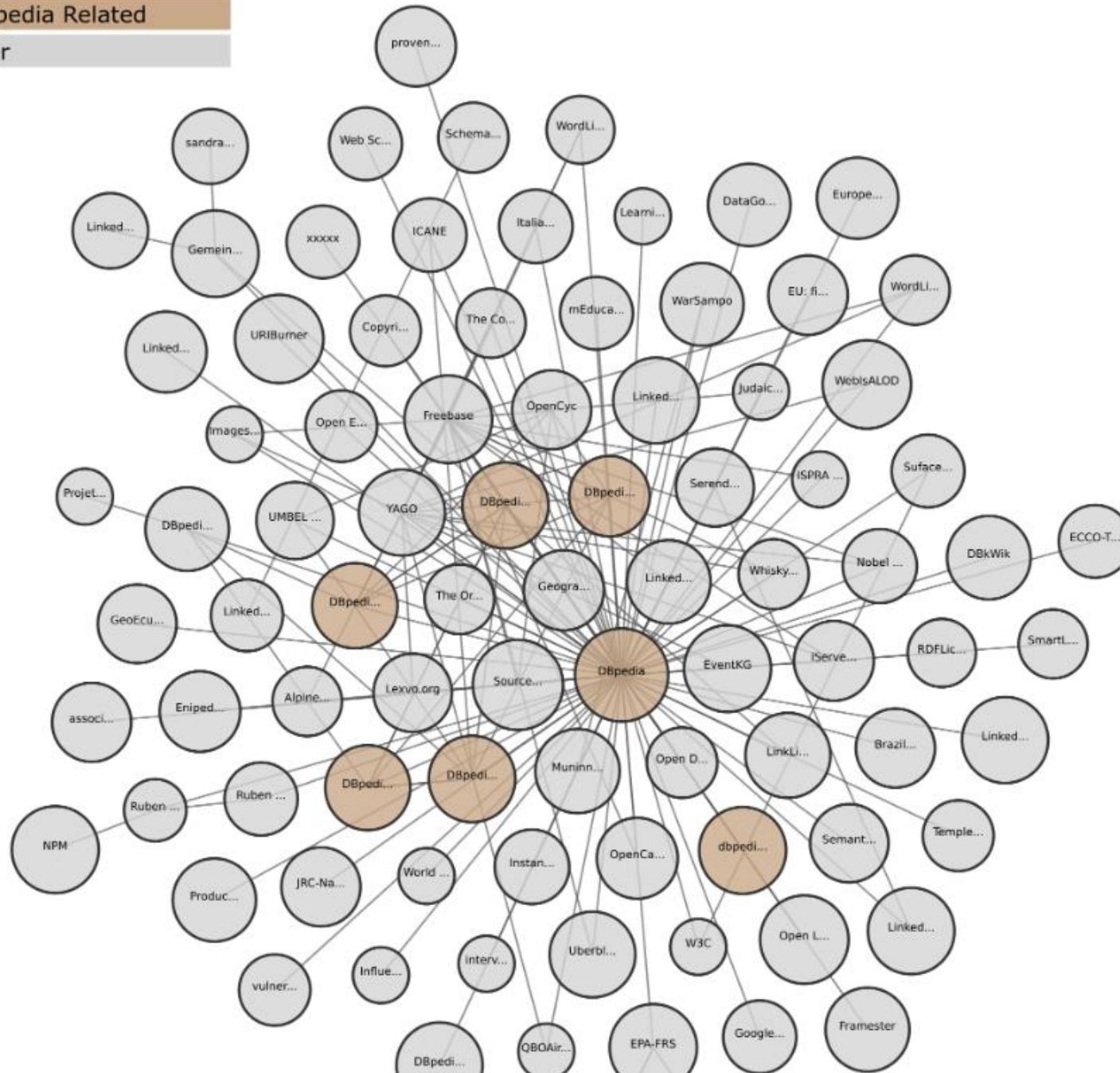
```
<http://dbpedia.org/resource/Amsterdam>
owl:sameAs <http://rdf.freebase.com/ns/...> ;
owl:sameAs <http://sws.geonames.org/2759793> ;
...
```

```
<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”





# Lots of Tools (*not* an exhaustive list!)

- Categories:

- Triple Stores
- Inference engines
- Converters
- Search engines
- Middleware
- CMS
- Semantic Web browsers
- Development environments
- Semantic Wikis
- ...

- Some names:

- Jena, AllegroGraph, Mulgara, Sesame, flickurl, ...
- TopBraid Suite, Virtuoso environment, Falcon, Drupal 7, Redland, Pellet, ...
- Disco, Oracle 11g, RacerPro, IODT, Ontobroker, OWLIM, Talis Platform, ...
- RDF Gateway, RDFLib, Open Anzo, DartGrid, Zitgist, Ontotext, Protégé, ...
- Thetus publisher, SemanticWorks, SWI-Prolog, RDFStore...
- ...

# References

- Berners-Lee, Hendler, Lassila (May 17, 2001). The Semantic Web. Scientific American Magazine. @ <http://bit.ly/2z2LZIE>
- 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/2xIWKou>
- <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>

# References

- <https://www.ontotext.com/knowledgehub/fundamentals/what-is-a-knowledge-graph/>
- [Google Knowledge Graph](#)
- [Building Knowledge graph](#)
- <https://www.youtube.com/watch?v=lUc0woFX16M>
- [https://www2.slideshare.net/jeffpan\\_sw/linked-data-and-knowledge-graphs-constructing-and-understanding-knowledge-graphs?qid=ab5124c7-a15a-4f77-9fc9-914feac00a6a&v=&b=&from\\_search=1](https://www2.slideshare.net/jeffpan_sw/linked-data-and-knowledge-graphs-constructing-and-understanding-knowledge-graphs?qid=ab5124c7-a15a-4f77-9fc9-914feac00a6a&v=&b=&from_search=1)
- [https://www2.slideshare.net/phaae/getting-started-with-knowledge-graphs?from\\_action=save](https://www2.slideshare.net/phaae/getting-started-with-knowledge-graphs?from_action=save)
- <https://lod-cloud.net/>
- <https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/>



**BITS** Pilani  
Pilani Campus



# Natural Language Processing DSECL ZG565

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



# **Encoder-Decoder Models, Attention and Contextual Embedding's**

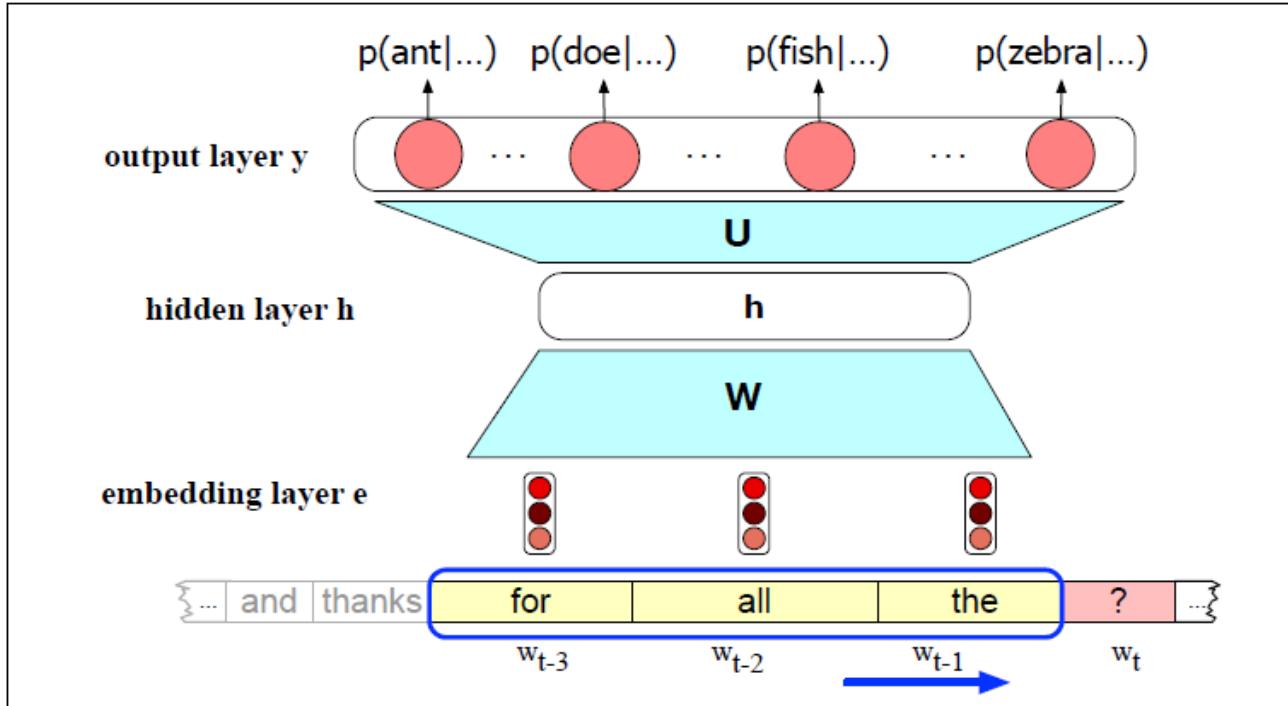
## **Date – 16th September 2023**

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

# Session Content

- Encoder-Decoder with RNN
- Issues with recurrent models
- Attention
- Transformer architecture
  - Self attention and Multihead-attention
  - Position encoding
  - Masked Self attention
  - Transformer encoder-decoder
  - Layer normalization
  - Residual connection
- Pre training
- Contextual Embeddings
  - BERT

# Simple feedforward sliding-window as a language model



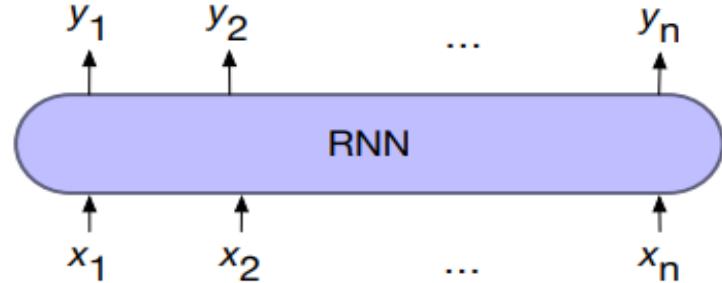
**Figure 9.1** Simplified sketch of a feedforward neural language model moving through a text. At each time step  $t$  the network converts  $N$  context words, each to a  $d$ -dimensional embedding, and concatenates the  $N$  embeddings together to get the  $Nd \times 1$  unit input vector  $x$  for 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.

# Deep Learning Architectures for Sequence Processing

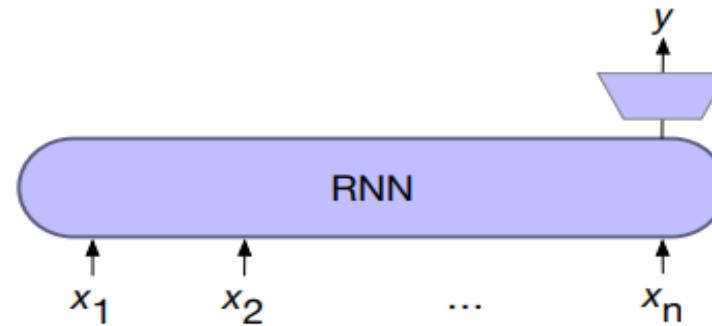
- **Recurrent neural networks and transformer networks**
- Both capture and exploit the **temporal** nature of language
- use the **prior context**, allowing the model's decision to depend on information from words in the past.
- **The transformer uses mechanisms (self-attention and positional encodings) that help focus on how words relate to each other over long distances**

# RNN Architectures for NLP Tasks



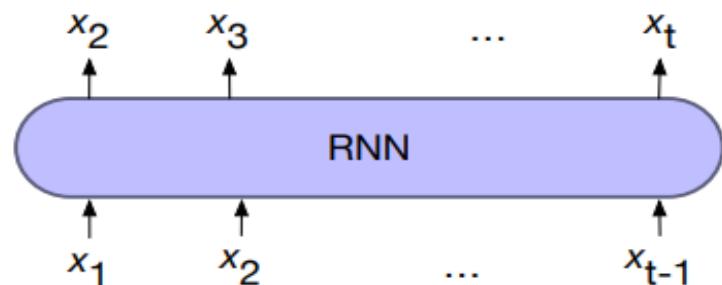
a) sequence labeling

Ex: POS Tagging, Named Entity Tagging



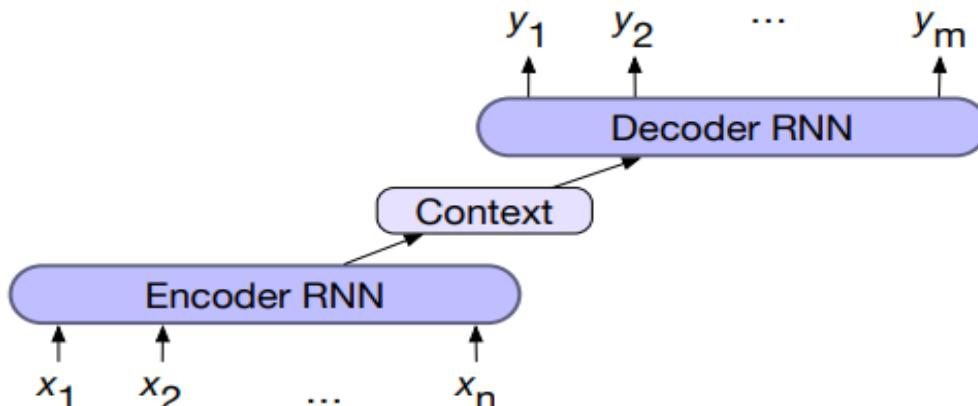
b) sequence classification

Ex: Sentiment Analysis



c) language modeling

Ex: Predict Next Word



d) encoder-decoder

Ex: Language Translation

# Encoder-Decoder or Sequence-to-Sequence Networks

---

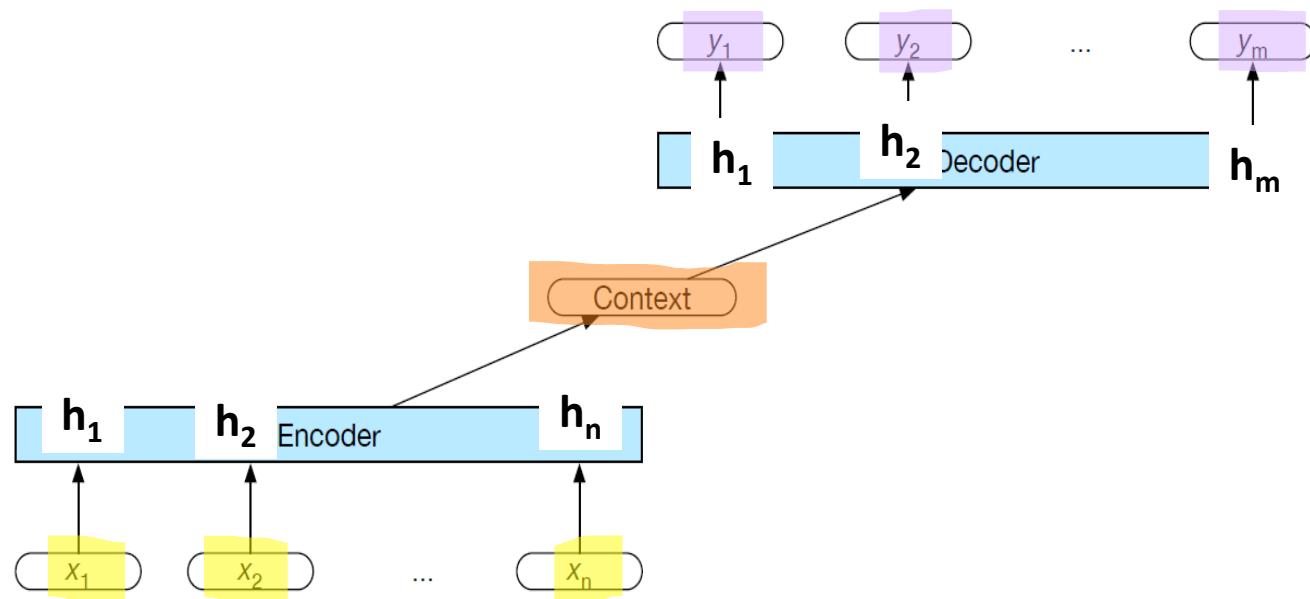
- **Models capable of generating contextually appropriate, arbitrary length, output sequences**
- One neural network takes input and produces a neural representation
- Another network produces output based on that neural representation
- Many NLP tasks can be phrased as sequence-to-sequence:
  - Summarization (long text → short text)
  - Dialogue (previous utterances → next utterance)
  - Parsing (input text → output parse as sequence)
  - Code generation (natural language → Python code)

# Encoder-Decoder architecture

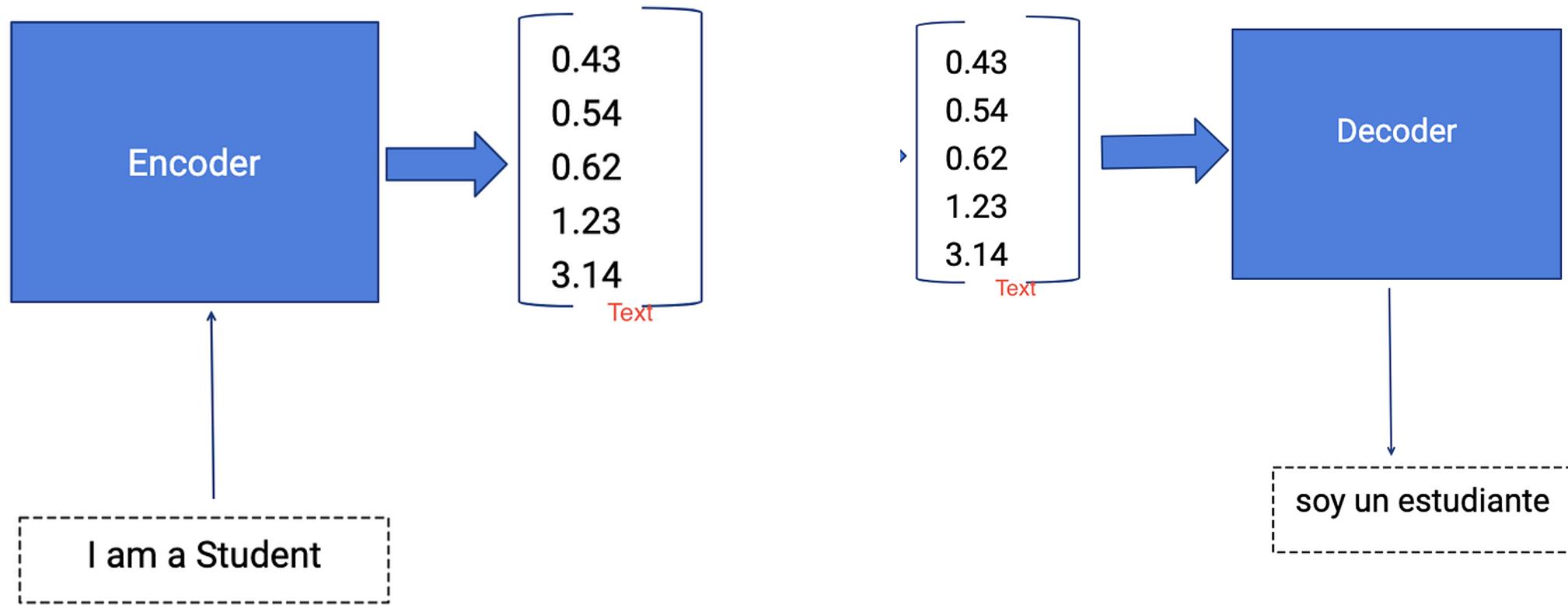


1. **Encoder:** accepts an input sequence,  $x_{1:n}$  and generates a corresponding sequence of contextualized representations,  $h_{1:n}$
2. **Context vector c:** function of  $h_{1:n}$  and conveys the essence of the input to the decoder.
3. **Decoder:** accepts  $c$  as input and generates an arbitrary length sequence of hidden states  $h_{1:m}$  from which a corresponding sequence of output states  $y_{1:m}$  can be obtained.

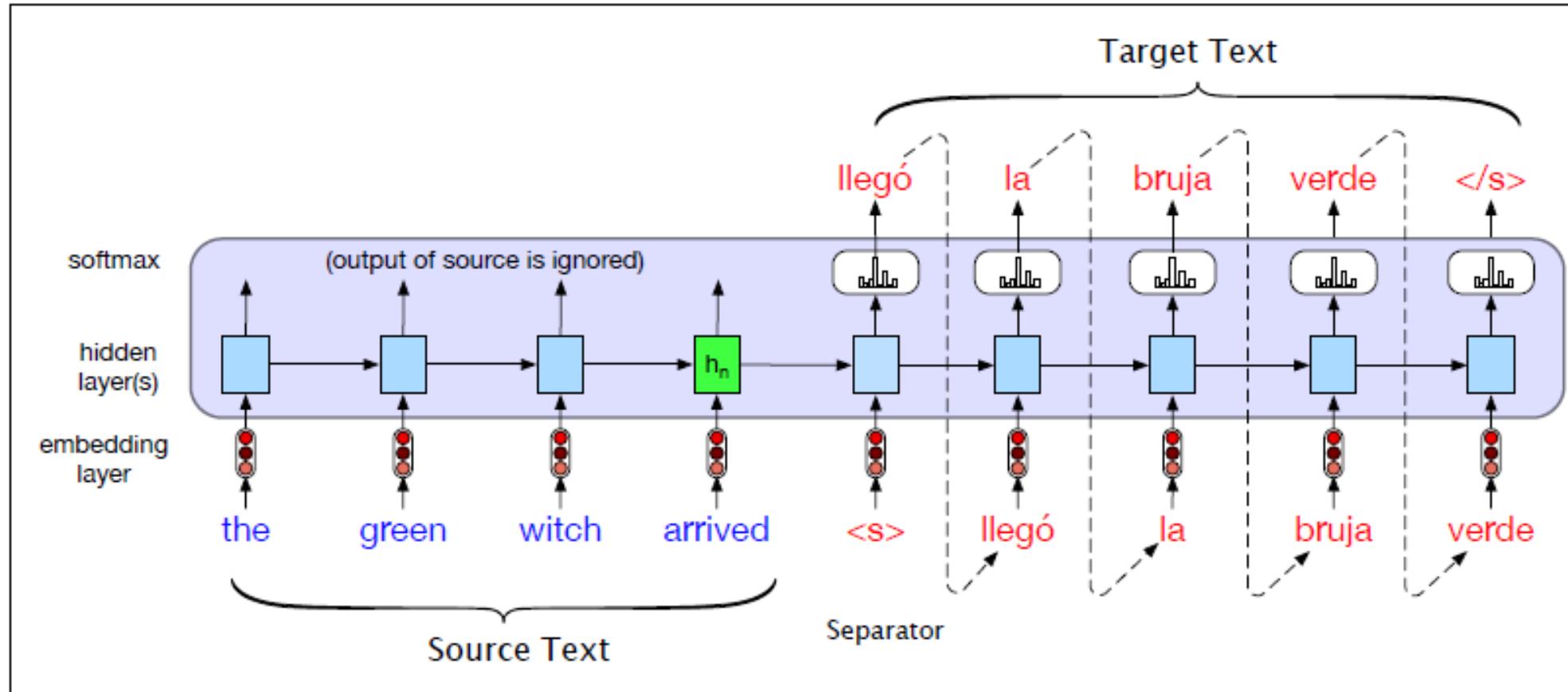
LSTMs, convolutional networks, and Transformers can all be employed as encoders/decoders



# Simple MT example using encoder-decoder



# MT with RNN based encoder-decoder



**Figure 10.4** Translating a single sentence (inference time) in the basic RNN version of encoder-decoder approach to machine translation. Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state.

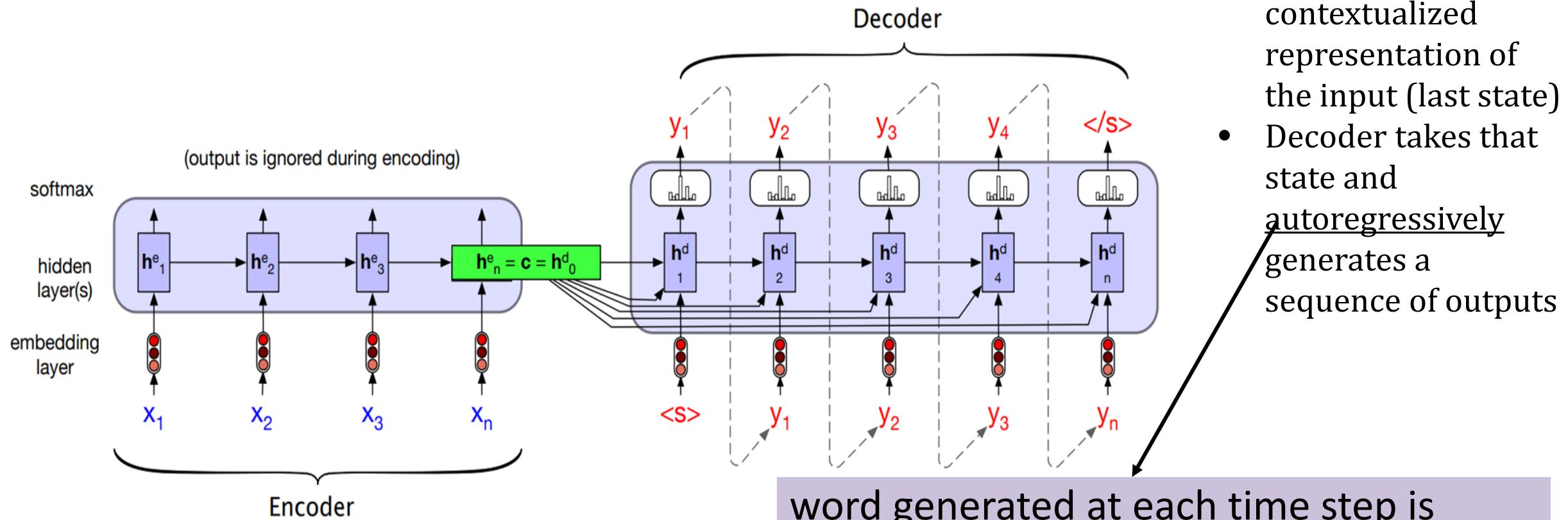
# Encoder - Decoder using RNN

innovate

achieve

lead

## Encoder - Decoder for Language Translation

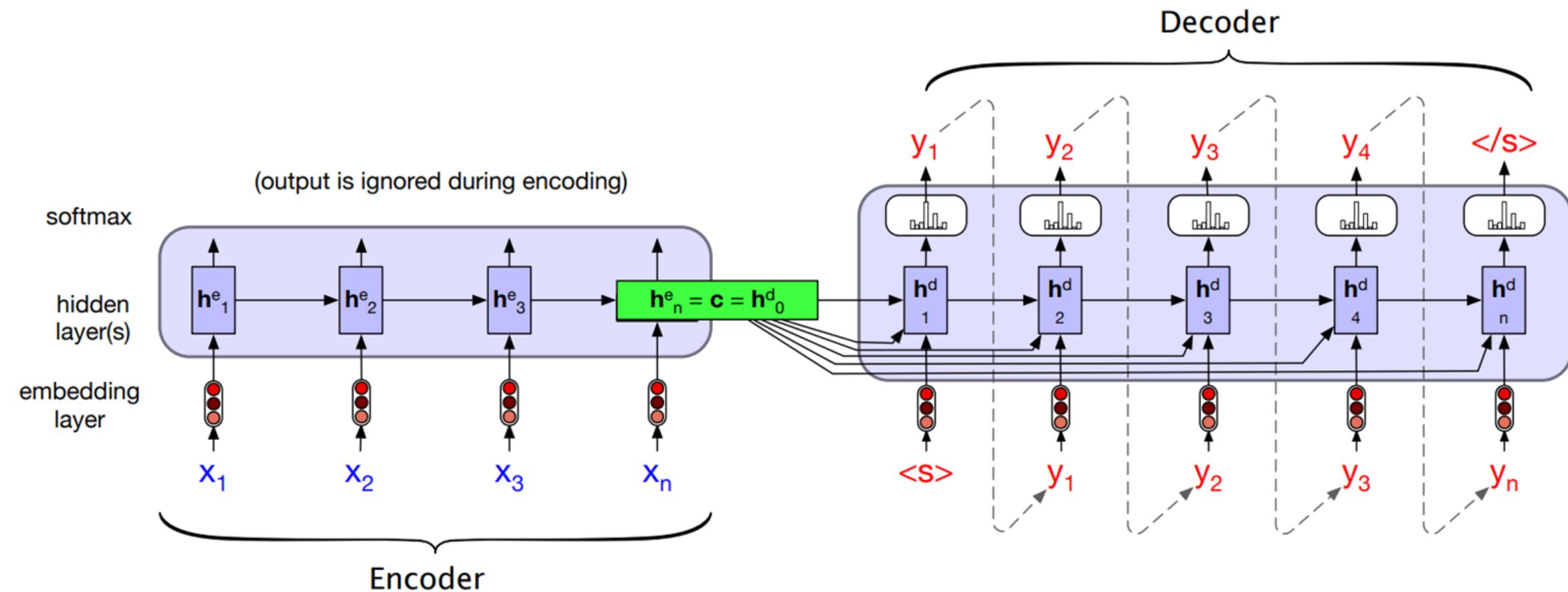


word generated at each time step is conditioned on word from previous step.

# Encoder - Decoder

## Encoder - Decoder for Language Translation

$$\begin{aligned}\mathbf{c} &= \mathbf{h}_n^e \\ \mathbf{h}_0^d &= \mathbf{c} \\ \mathbf{h}_t^d &= g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c}) \\ \mathbf{z}_t &= f(\mathbf{h}_t^d) \\ y_t &= \text{softmax}(\mathbf{z}_t)\end{aligned}$$

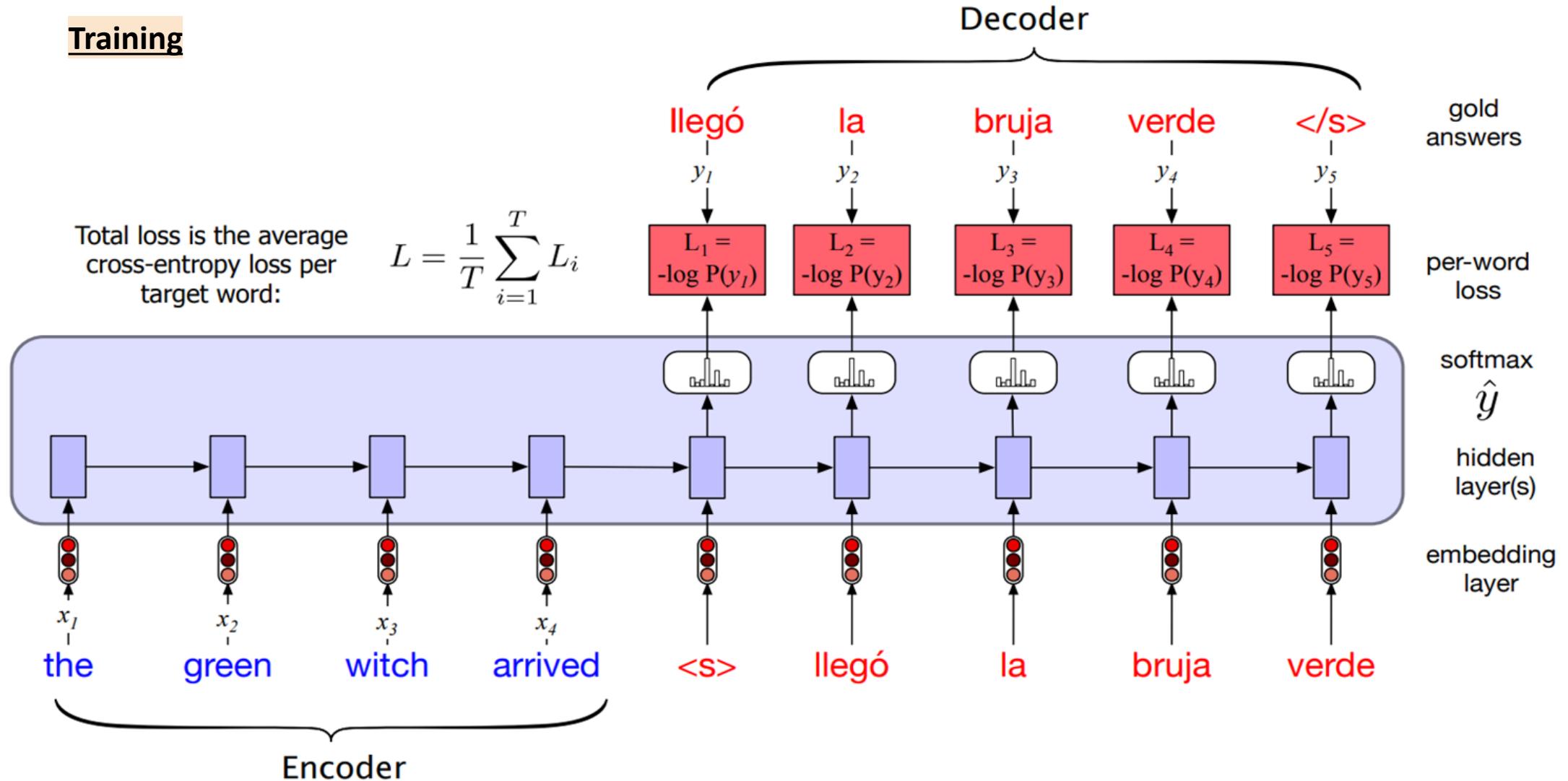


# Encoder - Decoder

## Training

Total loss is the average cross-entropy loss per target word:

$$L = \frac{1}{T} \sum_{i=1}^T L_i$$



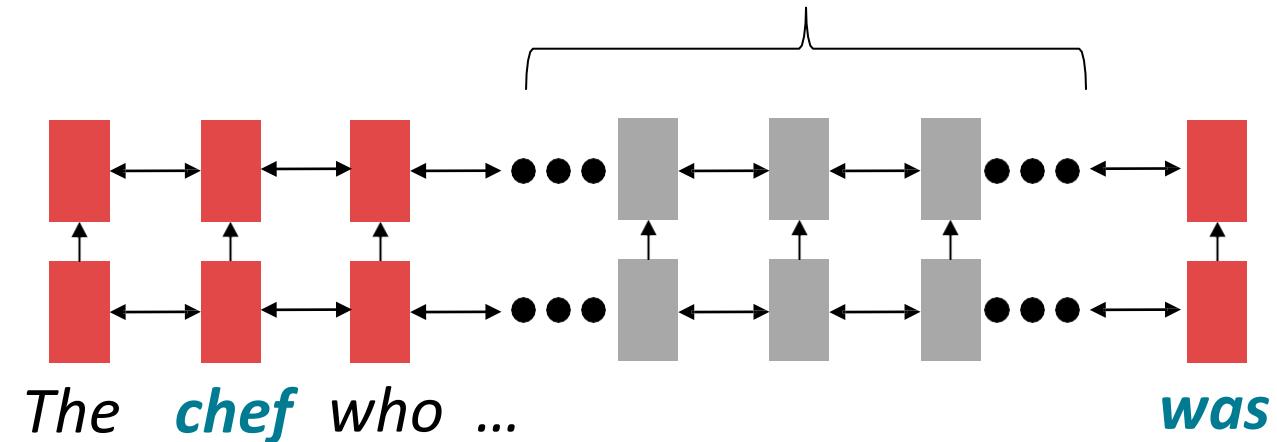
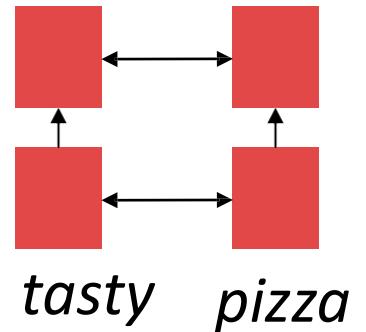
# Encoder - Decoder

## Teaching Forcing

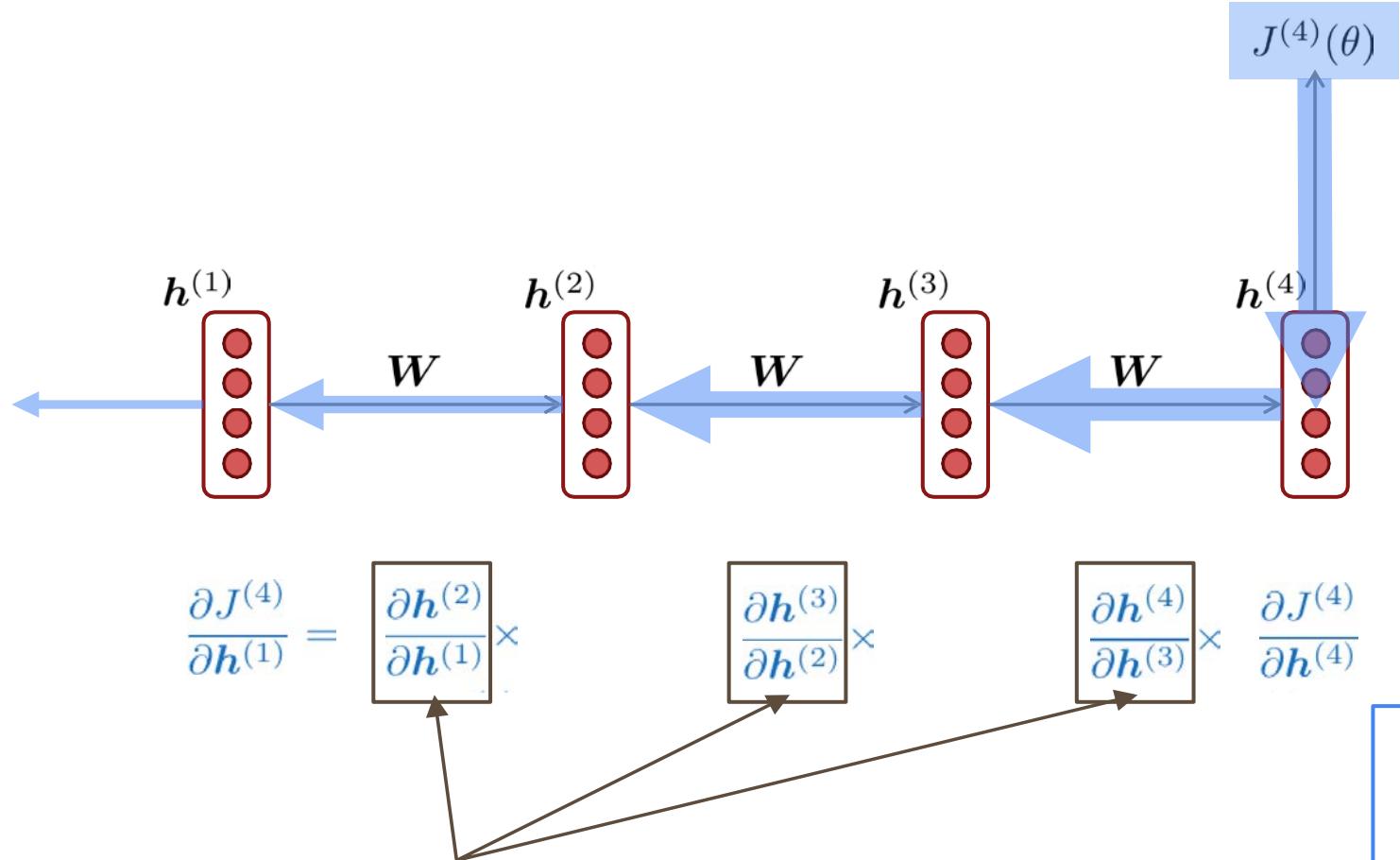
- Force the system to use the gold target token from training as the next input  $x_{t+1}$ , rather than allowing it to rely on the (possibly erroneous) decoder output  $\hat{y}_t$ .
- Speeds up training

# Issues with recurrent models

- **O(sequence length)** steps for distant word pairs to interact means:
  - Forward and backward passes have **O(sequence length)** unparallelizable operations
- RNNs are unrolled “left-to-right”.
- This encodes linear locality: a useful heuristic!
  - Nearby words often affect each other’s meanings
  - Hard to learn long-distance dependencies (because gradient problems!)



# Vanishing gradient intuition

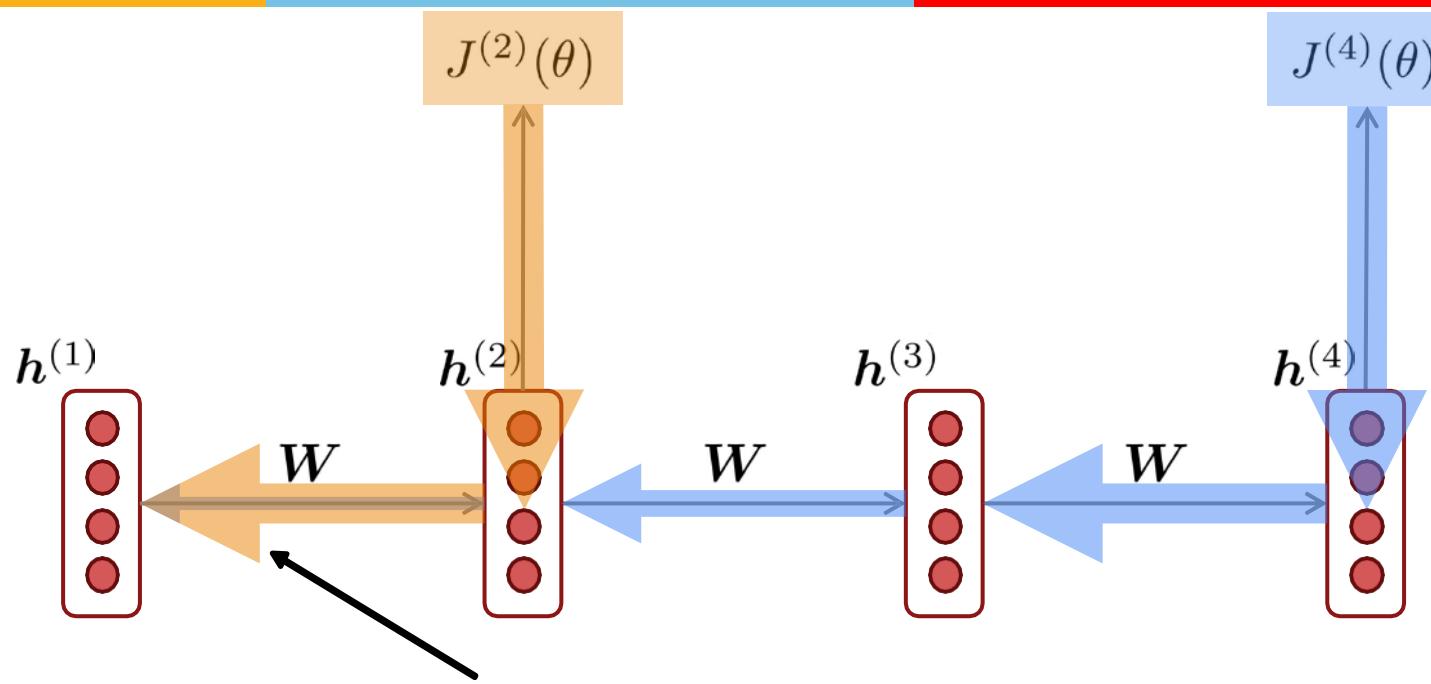


What happens if these are small?

chain rule!

**Vanishing gradient problem:**  
When these are small, the gradient signal gets smaller and smaller as it backpropagates further

# Why is vanishing gradient a problem?



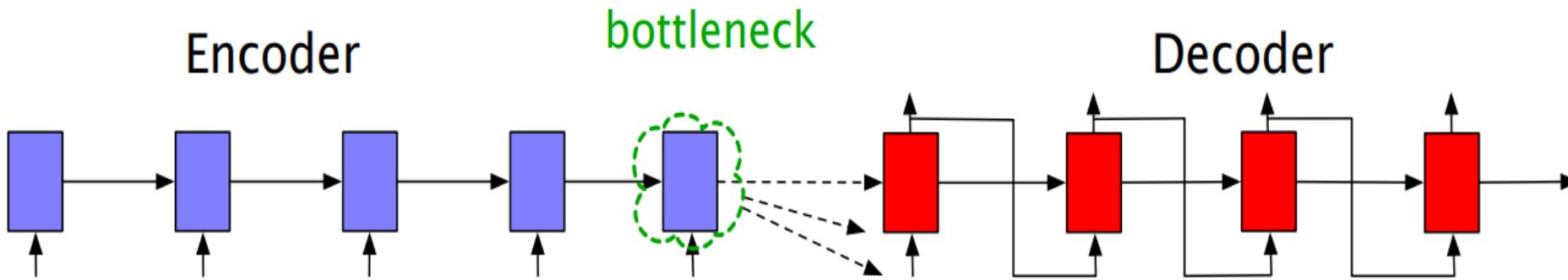
Gradient signal from far away is lost because it's much smaller than gradient signal from close-by.

So, model weights are basically updated only with respect to near effects, not long-term effects.

# Effect of vanishing gradient on RNN-LM

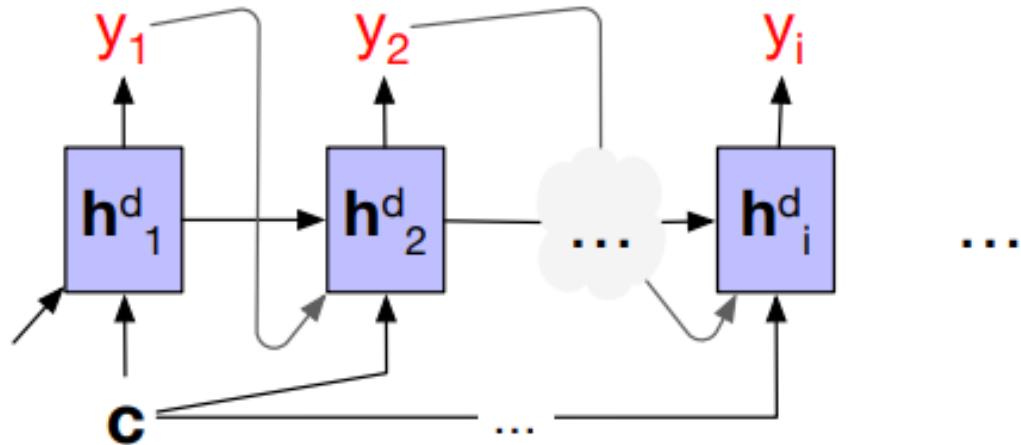
- **LM task:** *When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her \_\_\_\_\_*
- To learn from this training example, the RNN-LM 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



- Final state of the **E** is the only context available to **D**
- 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

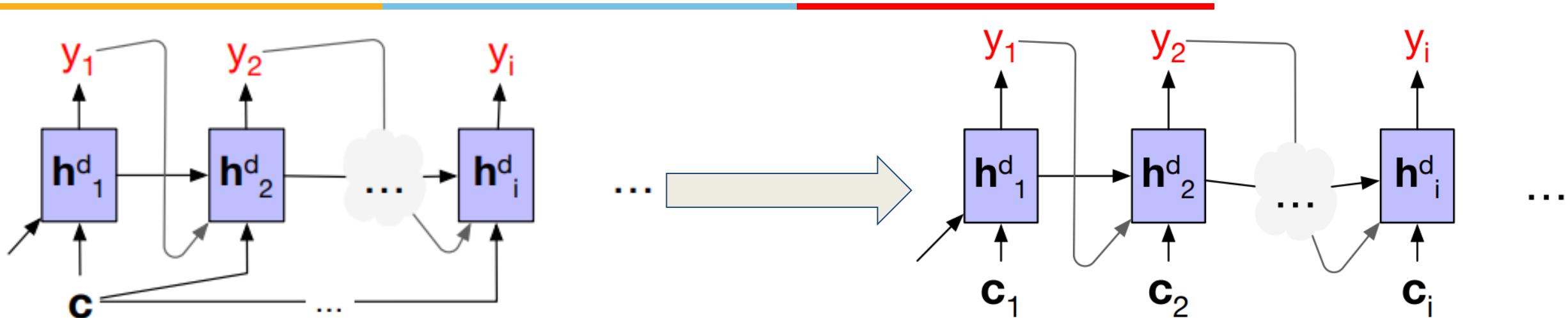
# Attention ! (in RNN based encoder-decoder)



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 ! (in RNN based encoder-decoder)



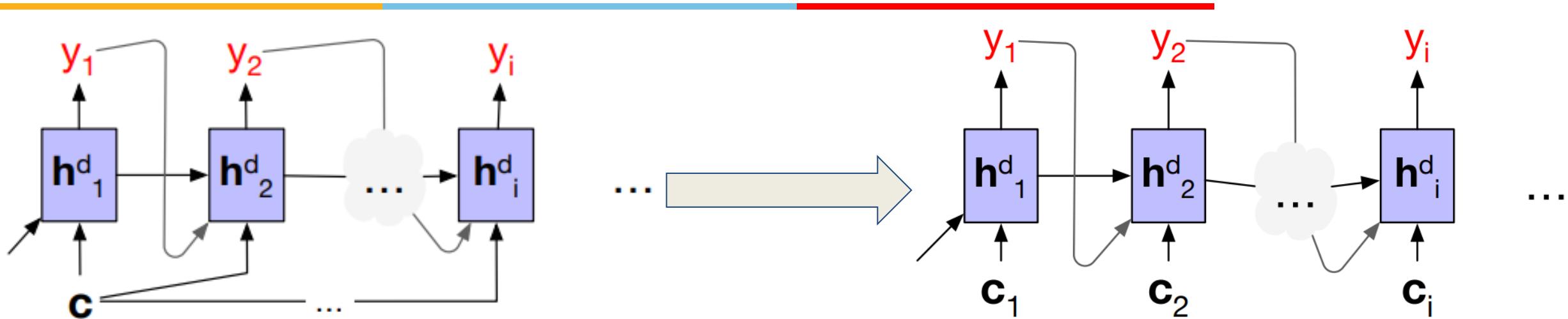
Without attention, a decoder sees the same context vector ,  
which is a static function of all the encoder hidden states

$$\mathbf{h}_t^d = g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c})$$

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

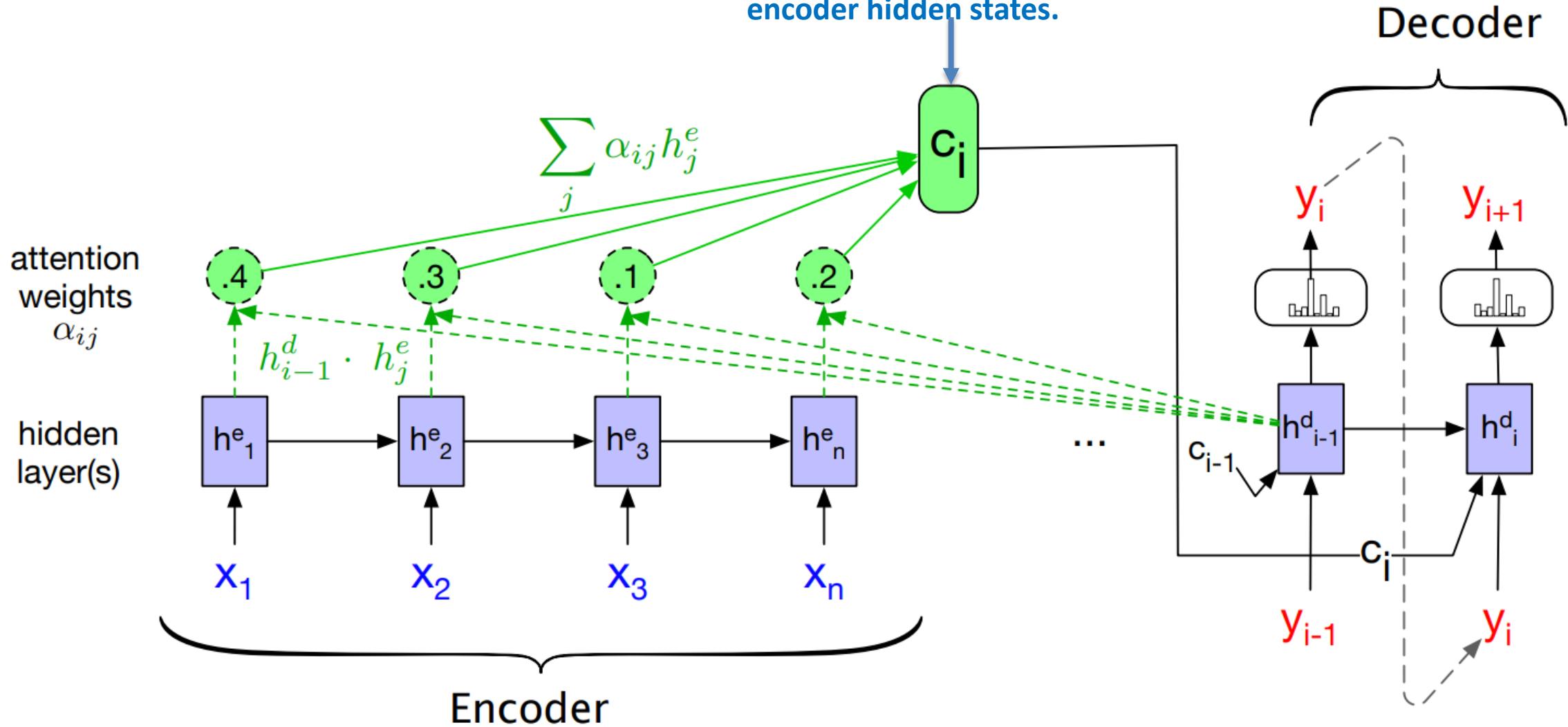
$$\mathbf{c}_i = \sum_j \alpha_{ij} \mathbf{h}_j^e$$

# Attention !



compute a fixed-length context vector for the current decoder state by taking a weighted average over all the encoder hidden states.

$$\mathbf{h}_i^d = g(\hat{y}_{i-1}, \mathbf{h}_{i-1}^d, \mathbf{c}_i)$$

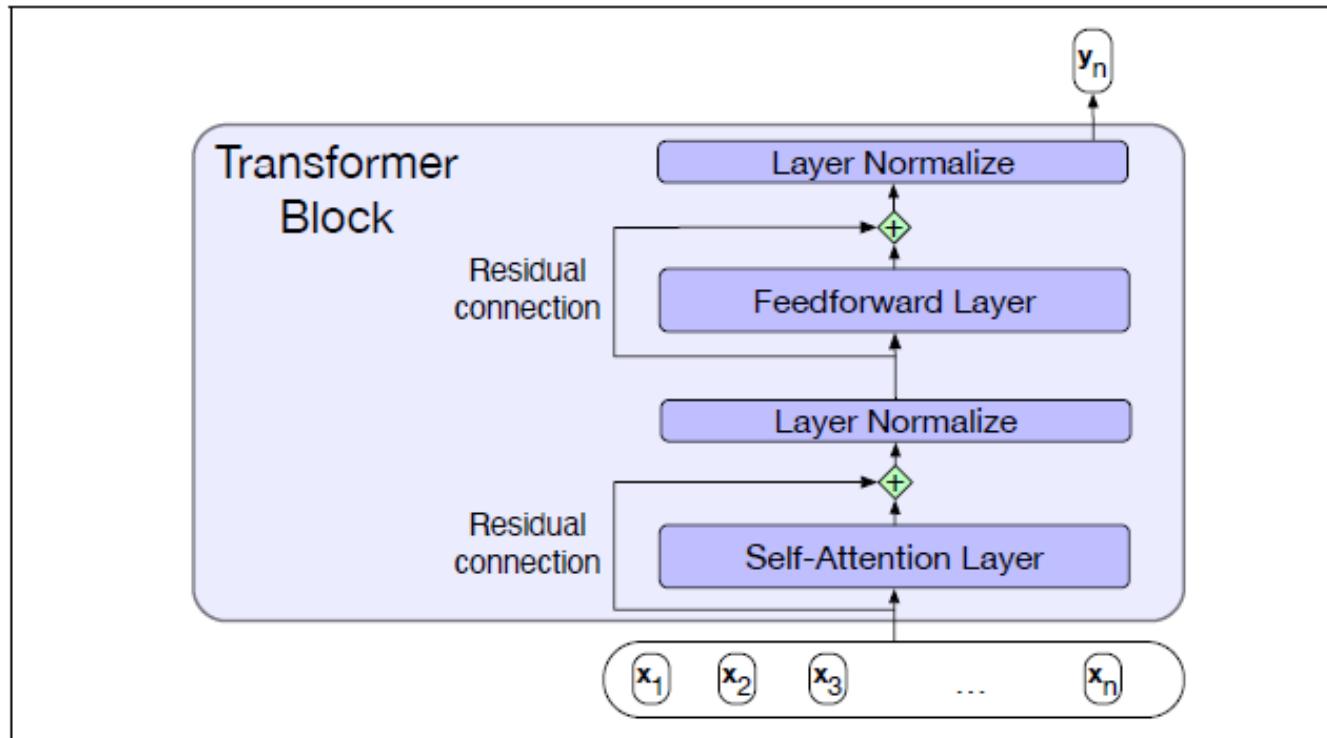


# Attention is a *general* Deep Learning technique

- Attention significantly **improves** 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
- Attention ⇒ 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 ⇒
  - > Set of **comparisons** are to other elements ***within a given sequence***
  - > Use these comparisons to compute an output for the current input

# Transformers

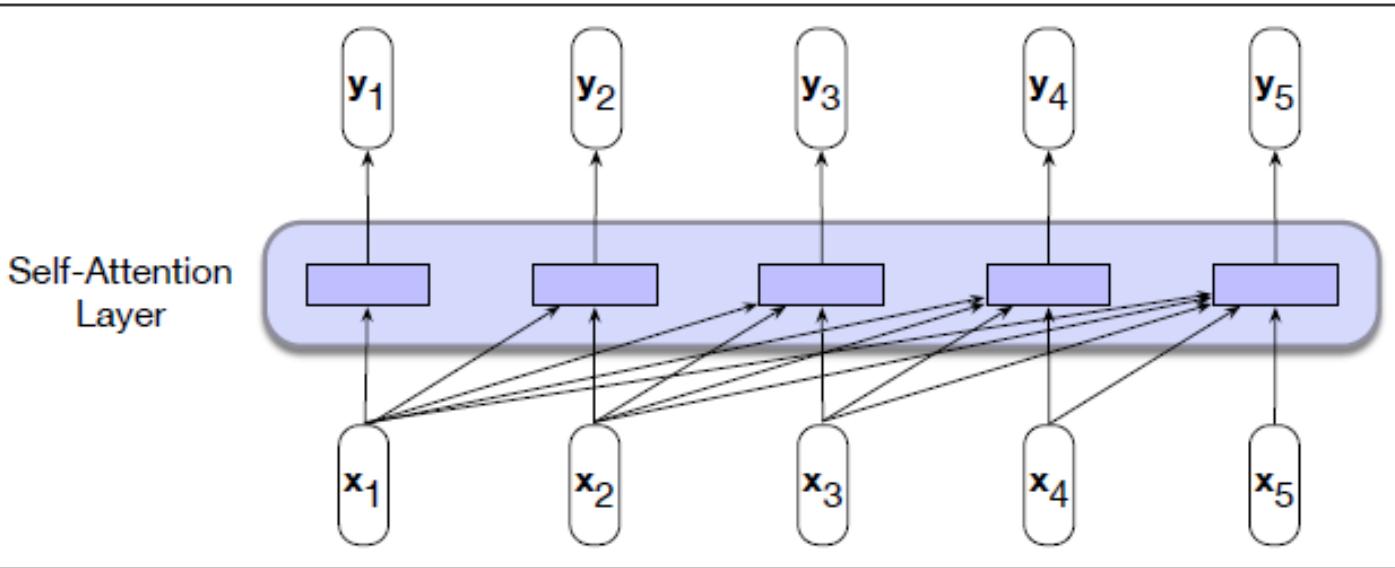
- 2017, NIPS, Vaswani et. al., **Attention Is All You Need !!!**
- Made up of **transformer blocks** in which the key component is **self-attention** layers
- Transformers are **not based on recurrent connections**  $\Rightarrow$  Parallel implementations possible  $\Rightarrow$  Efficient to scale ( comparing LSTM)



- Each Block consists of:
  - Self-attention
  - Add & Norm
  - Feed-Forward
  - Add & Norm

**Figure 9.18** A transformer block showing all the layers.

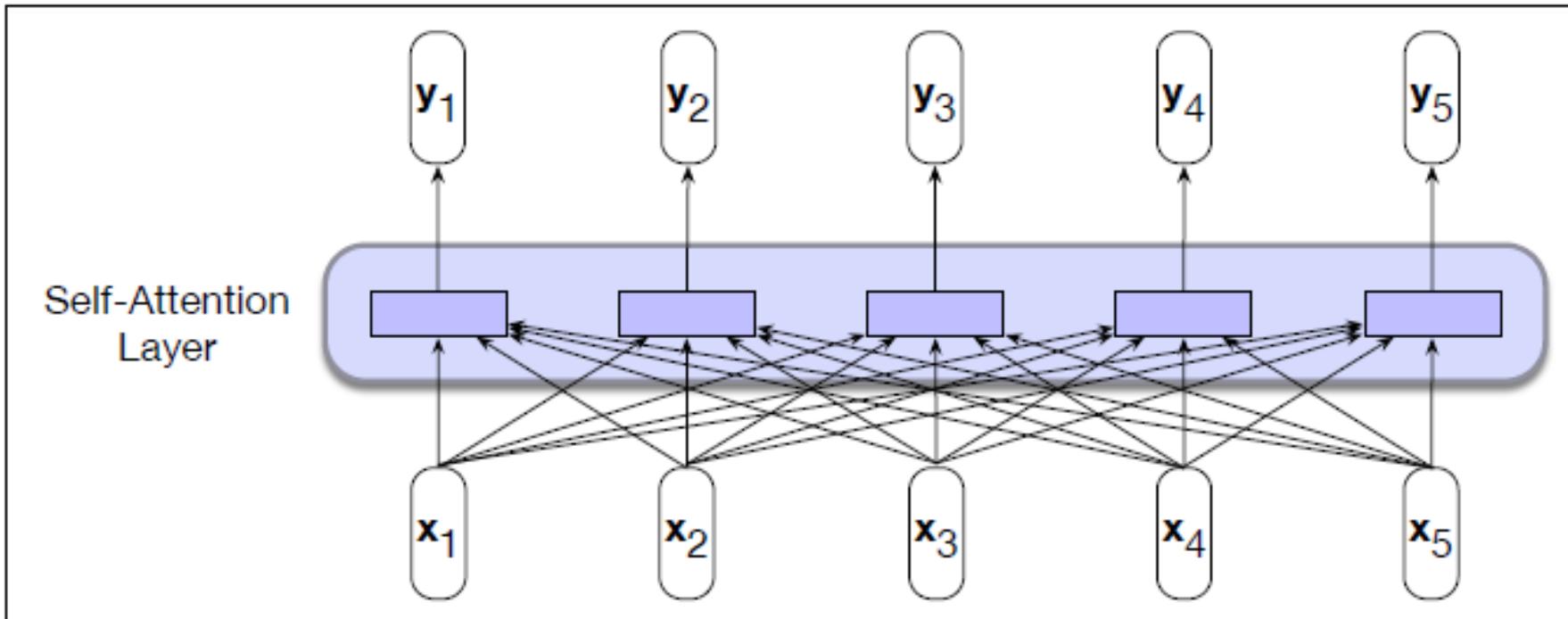
# Self attention layer



**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**

# Bidirectional self-attention model

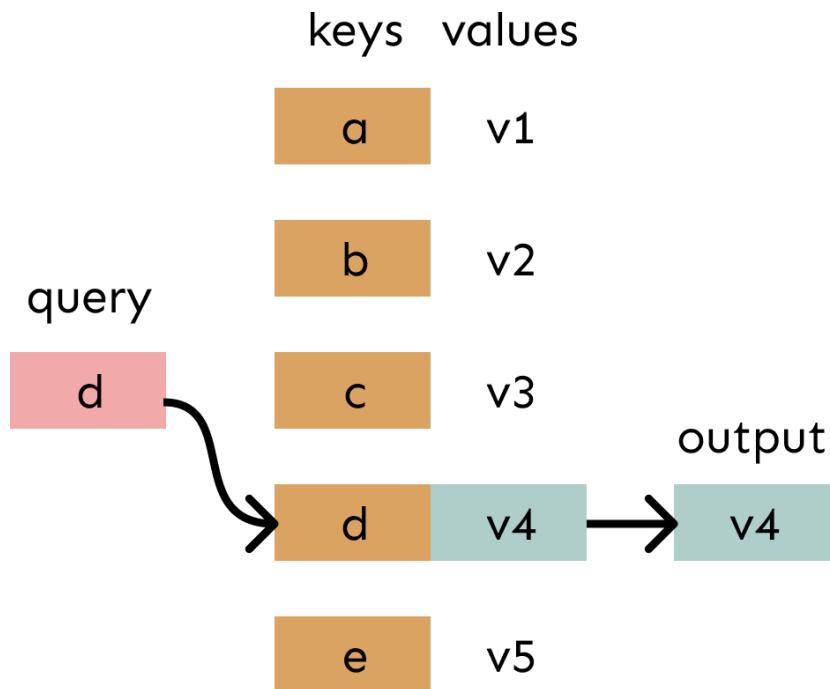


**Figure 11.2** Information flow in a bidirectional self-attention model. In processing each element of the sequence, the model attends to all inputs, both before and after the current one.

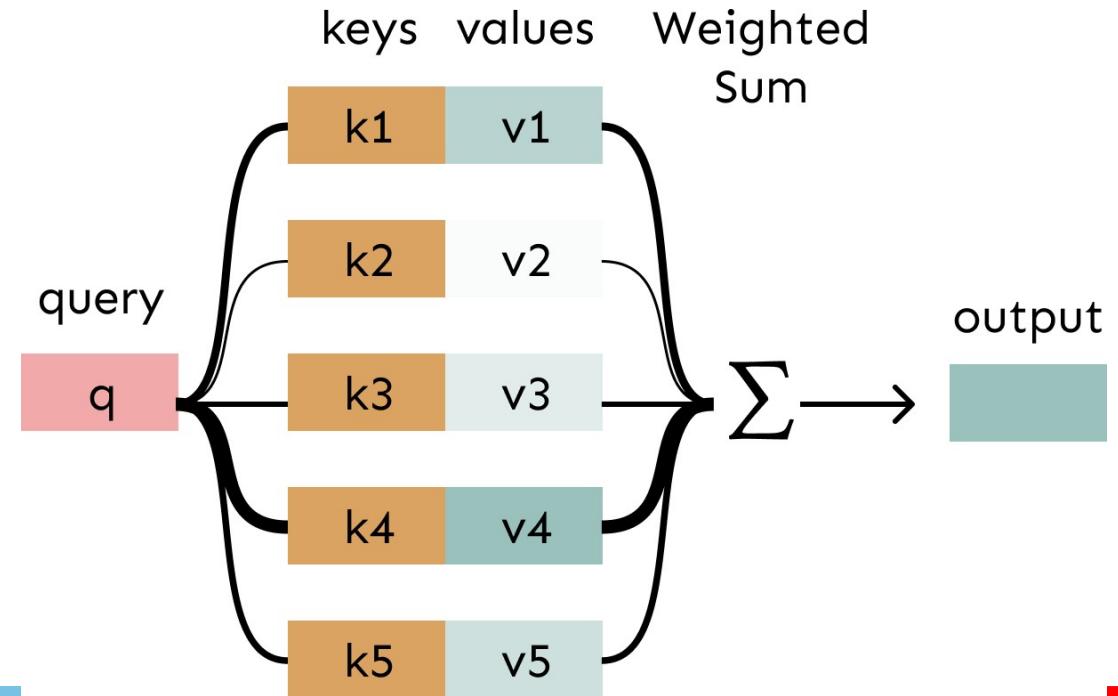
# Attention as a soft, averaging lookup table

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.

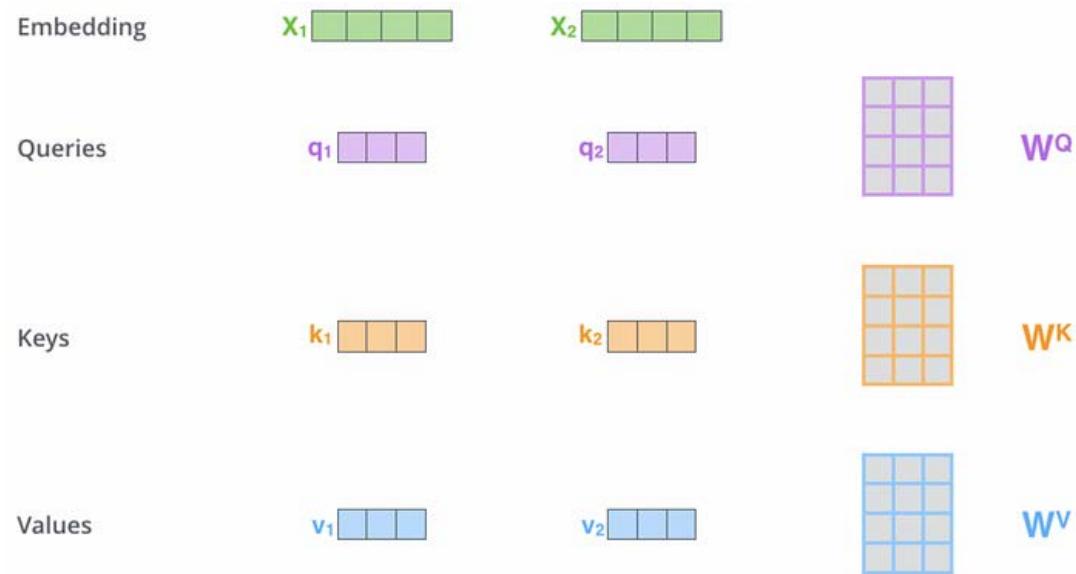


In **attention**, the **query** matches all **keys** *softly*, to a weight between 0 and 1. The keys' **values** are multiplied by the weights and summed.



# Parallelized using efficient matrix multiplication

Create three vectors from each of the encoder's input values (query, key, value)



## 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**. E.g. in the seq2seq + attention model, each decoder hidden state (query) *attends to* all the encoder hidden states (values).

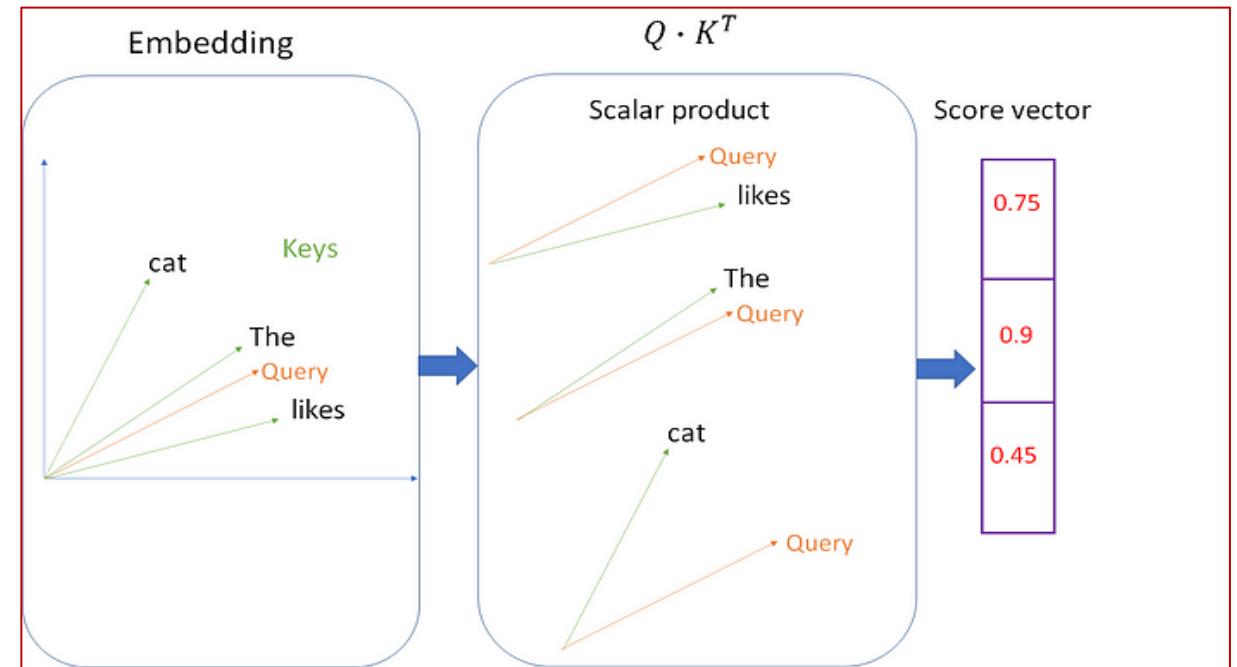
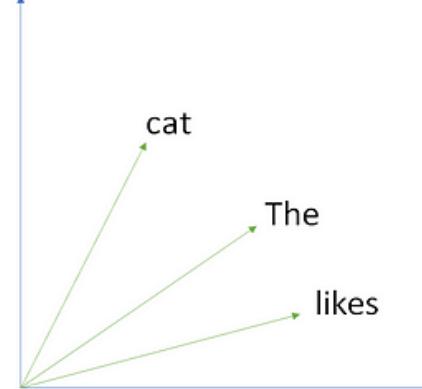
Self-attention step for an entire sequence of N tokens:

$$\text{SelfAttention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

$D_k$ =dimensionality of the query and key vectors  
acts as regularization and improve performance of larger models

# Self-Attention Hypothetical Example

The  
cat  
likes



# Self-Attention: keys, queries, values from the same sequence

computing a single output at a single time step i

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

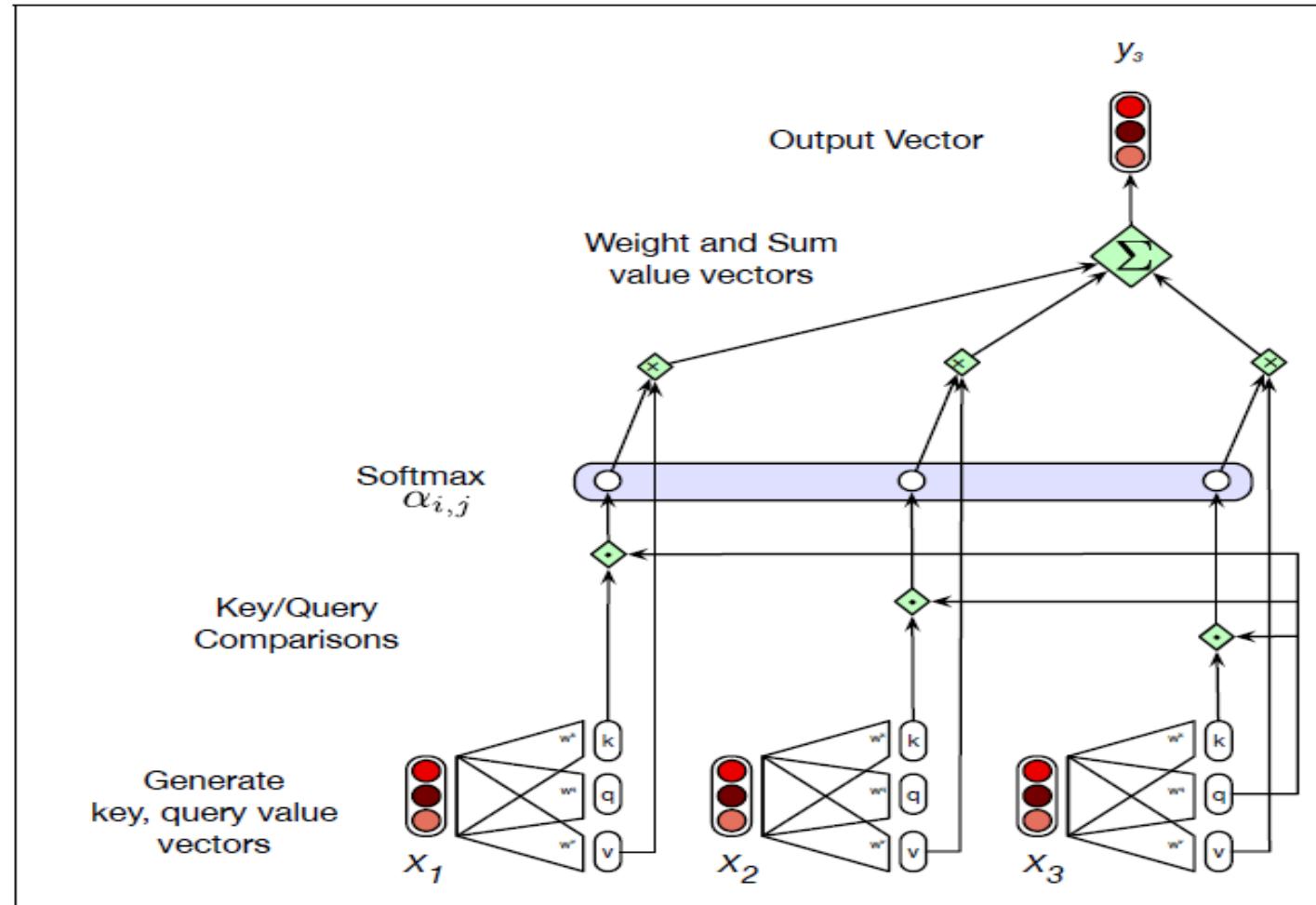
$$e_{ij} = q_i^T k_j \quad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

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.



**Figure 9.16** Calculating the value of  $y_3$ , the third element of a sequence using causal (left-to-right) self-attention.

# Self attention Example

3 input vectors  $x_i$

[1, 0, 1, 0]  
[0, 2, 0, 2]  
[1, 1, 1, 1]

innovate

achieve

lead

Weights for Key (K)

[[0, 0, 1],  
[1, 1, 0],  
[0, 1, 0],  
[1, 1, 0]]

Key representation for Input

[1, 0, 1, 0] . [0, 0, 1]  
[1, 1, 0] . [1, 1, 0] = [0, 1, 1]  
[0, 1, 0] . [0, 1, 0]  
[1, 1, 0] . [1, 1, 0]

Key for all inputs

[0 1 1]  
[4 4 0]  
[2 3 1]

$k_i = Kx_i$  (keys)

Weights for query (Q)

[[1, 0, 1],  
[1, 0, 0],  
[0, 0, 1],  
[0, 1, 1]]

Query representation for Input

[1, 0, 1, 0] . [1, 0, 1]  
[1, 0, 0] . [1, 0, 0] = [1, 0, 2]  
[0, 0, 1] . [0, 0, 1]  
[0, 1, 1] . [0, 1, 1]

Query for all inputs

[1, 0, 2]  
[2, 2, 2]  
[2, 1, 3]

$q_i = Qx_i$  (queries)

Weights for Value (V)

[[0, 2, 0],  
[0, 3, 0],  
[1, 0, 3],  
[1, 1, 0]]

Value representation for Input

[1, 0, 1, 0] . [0, 2, 0]  
[1, 0, 0] . [0, 3, 0] = [1, 2, 3]  
[1, 0, 3] . [1, 0, 3]  
[1, 1, 0] . [1, 1, 0]

Value for all inputs

[1, 2, 3]  
[2, 8, 0]  
[2, 6, 3]

$v_i = Vx_i$  (values)

3 attention scores for first input  $i$  :  $(e_{ij})$

$$[0, 1, 1]$$

$$[1, 0, 2] \cdot [4, 4, 0] = [2, 4, 4]$$

$$[2, 3, 1]$$

3 softmax attention scores ( $\alpha_{ij}$ )

$$\text{softmax}([2, 4, 4]) = [0.0, 0.5, 0.5]$$

$$e_{ij} = q_i^T k_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_j \exp(e_{ij'})}$$

weighted values

$$1: 0.0 * [1, 2, 3] = [0.0, 0.0, 0.0]$$

$$2: 0.5 * [2, 8, 0] = [1.0, 4.0, 0.0]$$

$$3: 0.5 * [2, 6, 3] = [1.0, 3.0, 1.5]$$

$$\alpha_{ij} v_i$$

Output 1, which is based on the query representation from Input 1 interacting with all other keys, including itself

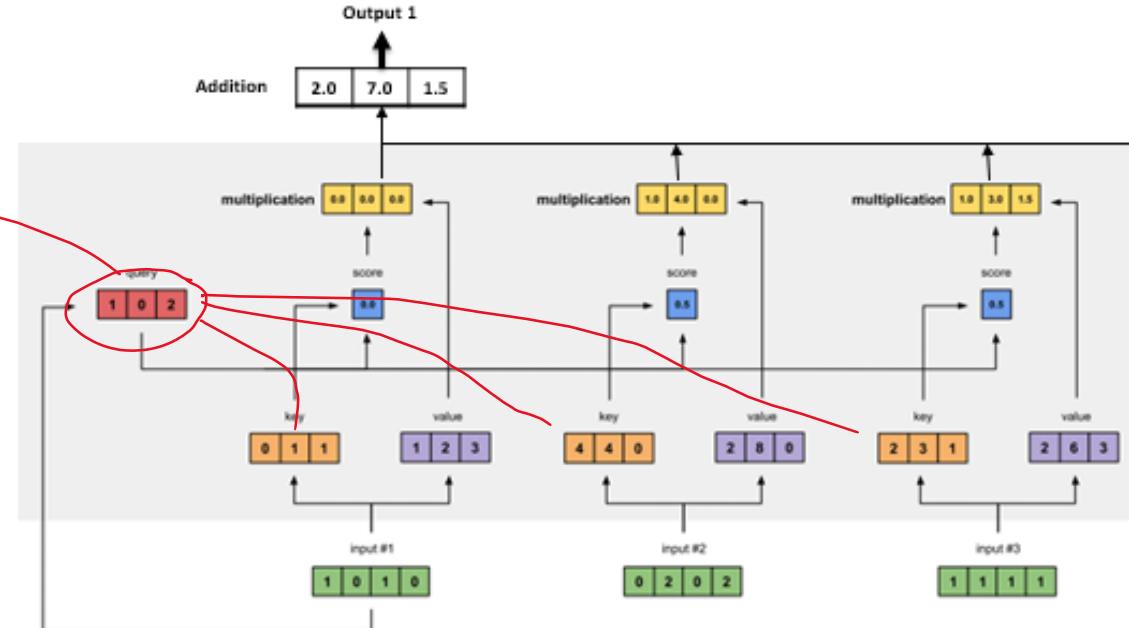
$$[0.0, 0.0, 0.0]$$

$$+ [1.0, 4.0, 0.0]$$

$$+ [1.0, 3.0, 1.5]$$

$$o_i = \sum_j \alpha_{ij} v_i$$

$$= [2.0, 7.0, 1.5]$$



All the outputs after first iteration  
 $[2.0, 7.0, 1.5]$ , # Output 1

$[2.0000, 8.0, 0.0]$ , # Output 2

$[2.0, 7.8, 0.3]$  # Output 3

# Multihead-Attention

---

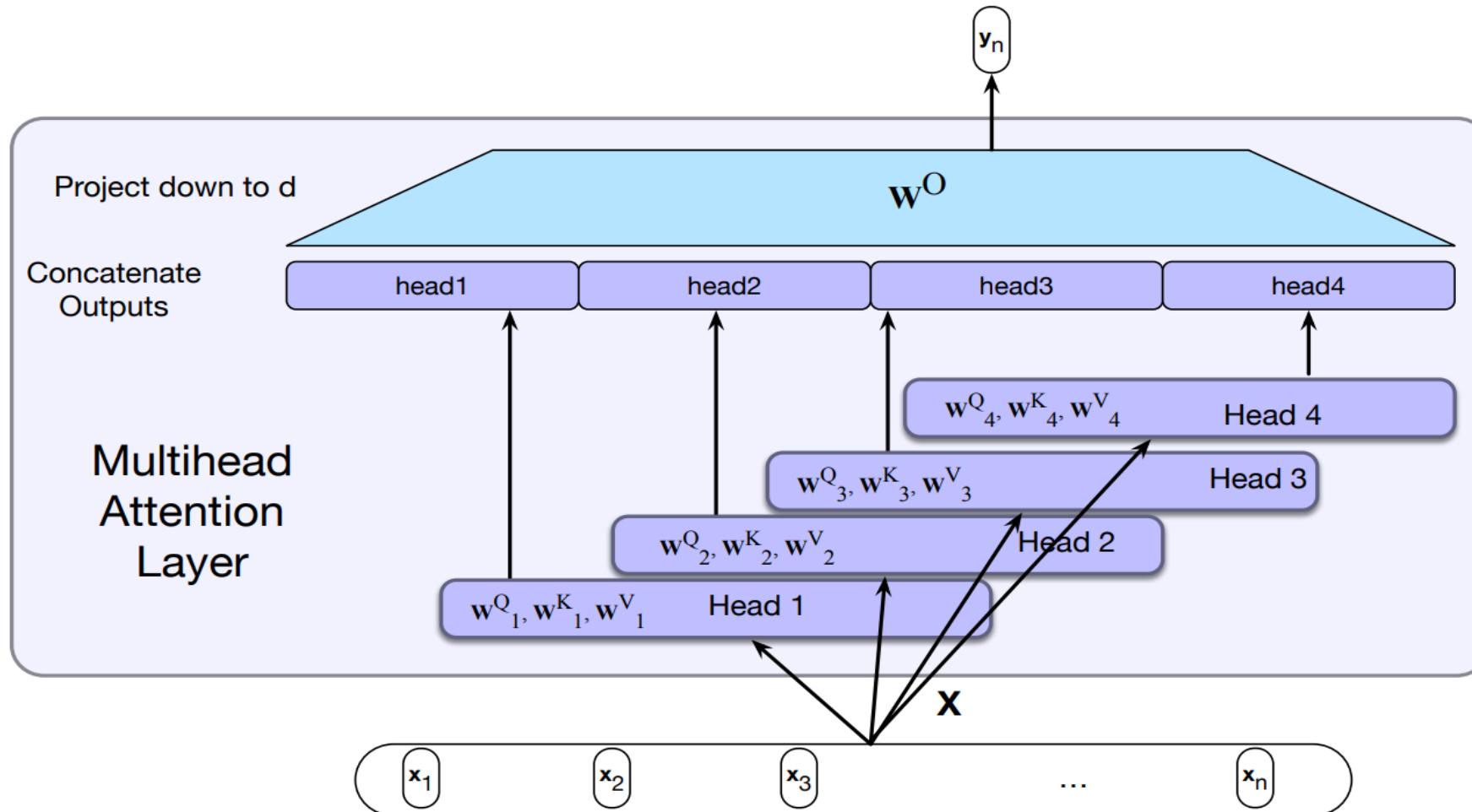
- Different words in a sentence can relate to each other in many different ways simultaneously
  - >> A single transformer block to learn to capture all of the different kinds of parallel relations among its inputs is inadequate.
- **Multihead** self-attention layers
  - >> **Heads** ⇒ sets of self-attention 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

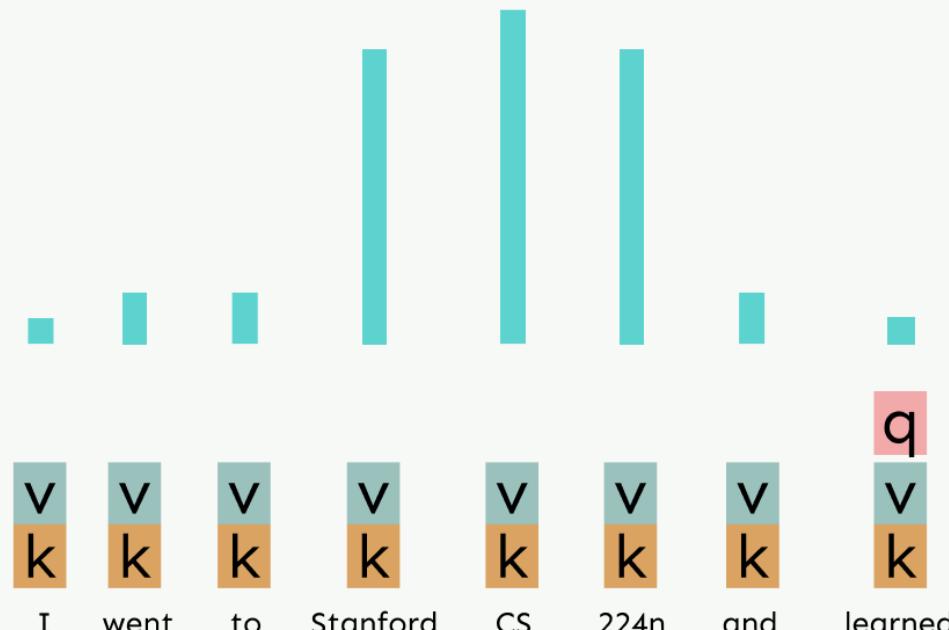


Each of the multihead self-attention layers is provided with its own set of key, query and value weight matrices.

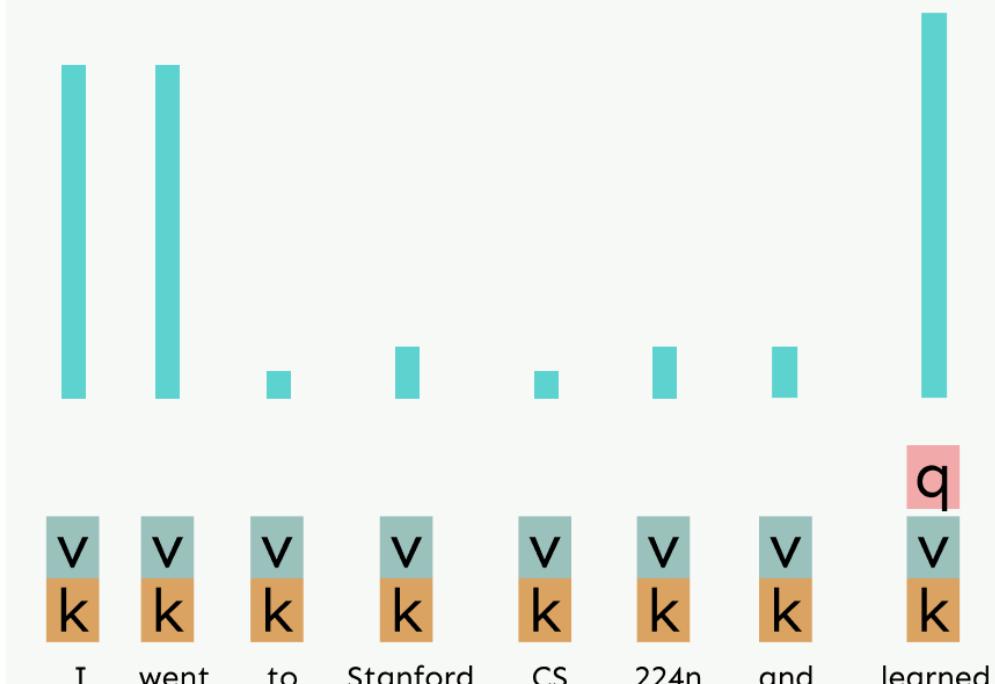
The outputs from each of the layers are concatenated and then projected down to  $d$ , thus producing an output of the same size as the input so layers can be stacked.

# Hypothetical Example of Multi-Head Attention

Attention head 1  
attends to entities



Attention head 2 attends to  
syntactically relevant words

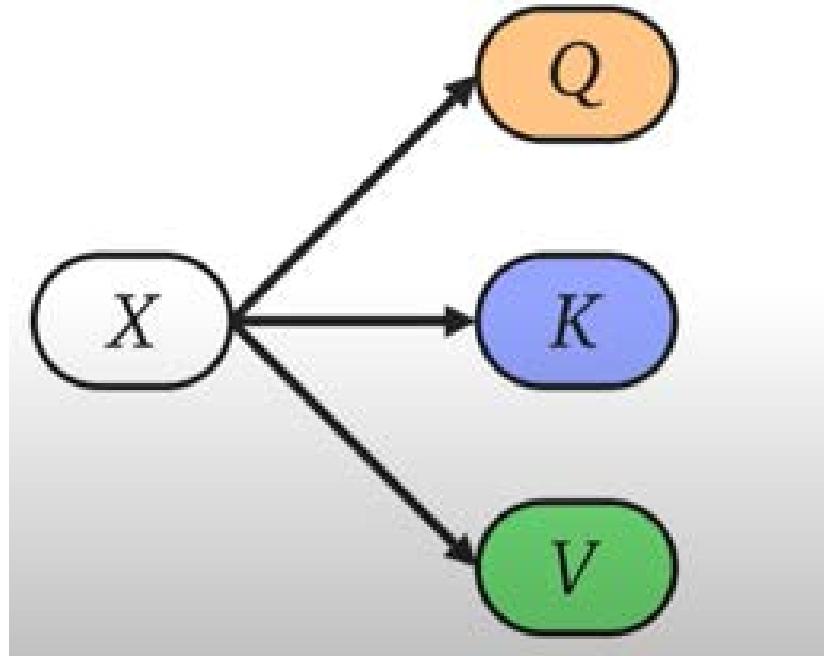


I      went      to      Stanford      CS      224n      and      learned

# Self Attention Vs Cross Attention

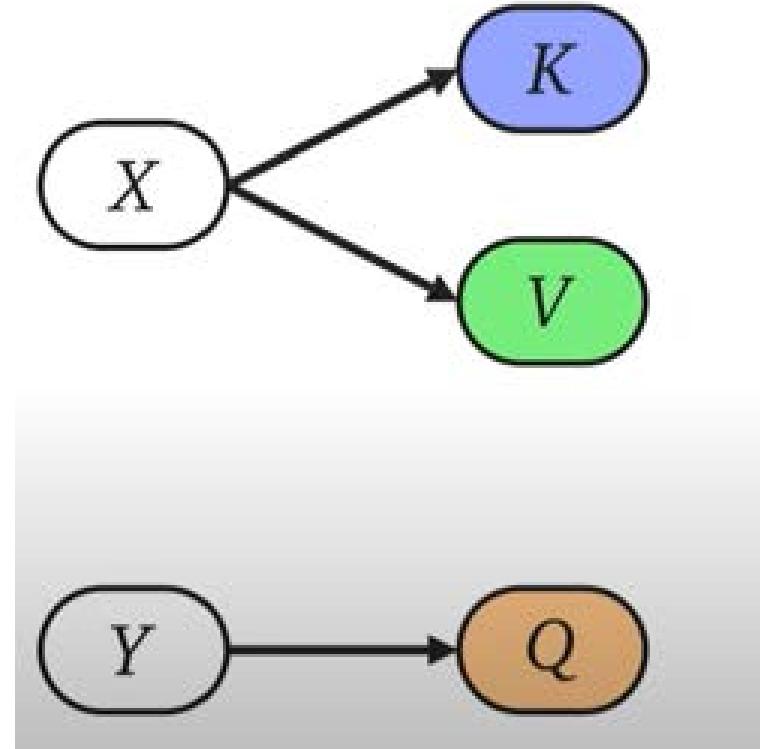
$$Q = XW^Q; \quad K = XW^K; \quad V = XW^V$$

$$\text{SelfAttention}(Q, K, V) = \text{softmax} \left( \frac{QK^\top}{\sqrt{d_k}} \right) V$$



$$Q = W^Q H^{dec[i-1]}; \quad K = W^K H^{enc}; \quad V = W^V H^{enc}$$

$$\text{CrossAttention}(Q, K, V) = \text{softmax} \left( \frac{QK^\top}{\sqrt{d_k}} \right) V$$



# 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

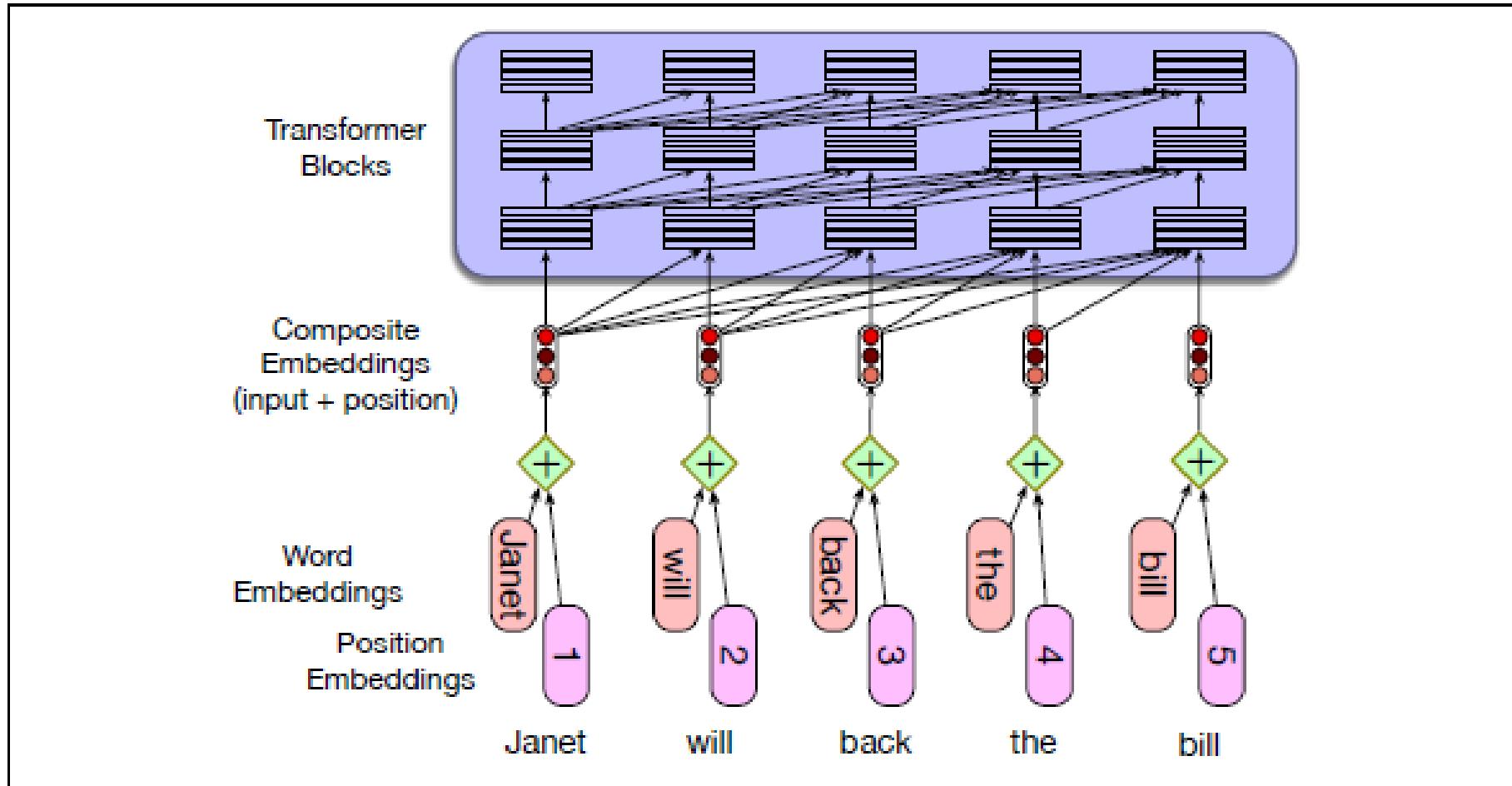
- Add position representations to the inputs
- Easy fix: apply the same feedforward network to each self- attention output.
- Mask out the future by artificially setting attention weights to 0!

# 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

Each position/index is mapped to a vector

| Sequence | Index of token, $k$ | Positional Encoding Matrix with $d=4$ , $n=100$ |                            |                            |                            |
|----------|---------------------|---|----------------------------|----------------------------|----------------------------|
|          |                     | $i=0$   | $i=0$                      | $i=1$                      | $i=1$                      |
| I        | 0                   | $P_{00}=\sin(0) = 0$                            | $P_{01}=\cos(0) = 1$       | $P_{02}=\sin(0) = 0$       | $P_{03}=\cos(0) = 1$       |
| am       | 1                   | $P_{10}=\sin(1/1) = 0.84$                       | $P_{11}=\cos(1/1) = 0.54$  | $P_{12}=\sin(1/10) = 0.10$ | $P_{13}=\cos(1/10) = 1.0$  |
| a        | 2                   | $P_{20}=\sin(2/1) = 0.91$                       | $P_{21}=\cos(2/1) = -0.42$ | $P_{22}=\sin(2/10) = 0.20$ | $P_{23}=\cos(2/10) = 0.98$ |
| Robot    | 3                   | $P_{30}=\sin(3/1) = 0.14$                       | $P_{31}=\cos(3/1) = -0.99$ | $P_{32}=\sin(3/10) = 0.30$ | $P_{33}=\cos(3/10) = 0.96$ |

Positional Encoding Matrix for the sequence 'I am a robot'

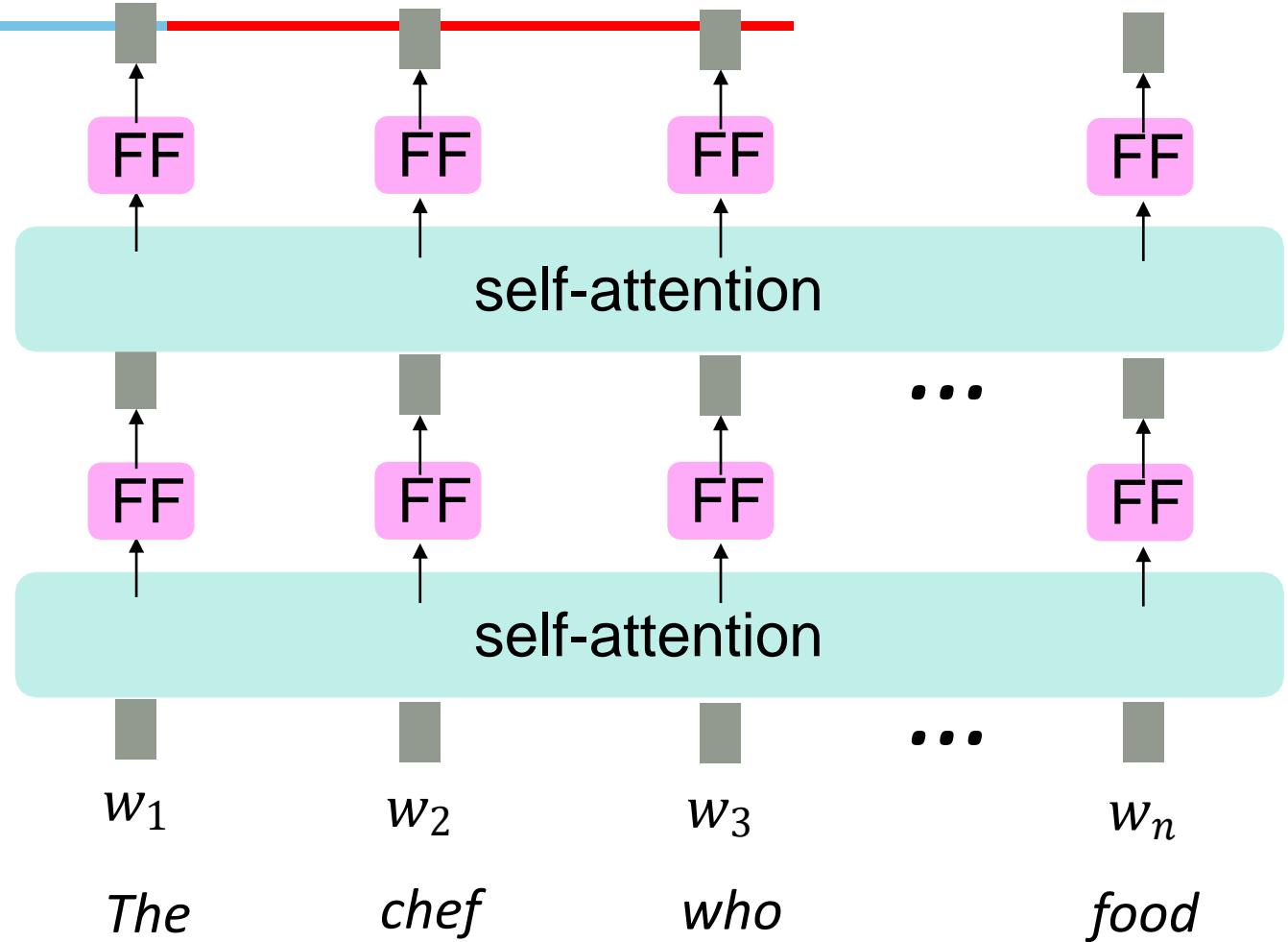
$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$

$d$ = Dimension of the output embedding space

$i$ = Used for mapping to column indices  $0 \leq i < d/2$ , with a single value of  $i$  maps to both sine and cosine functions

# 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     |           |           |           |

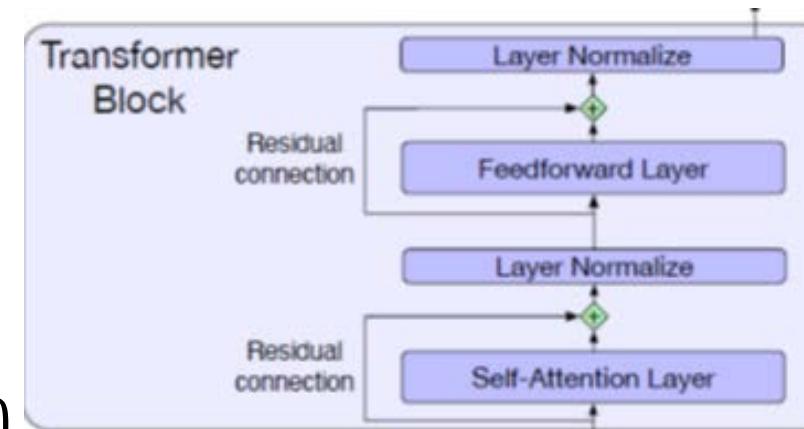
# Residual connections

- **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)



# 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.

$$\hat{x} = \frac{(x - \mu)}{\sigma}$$

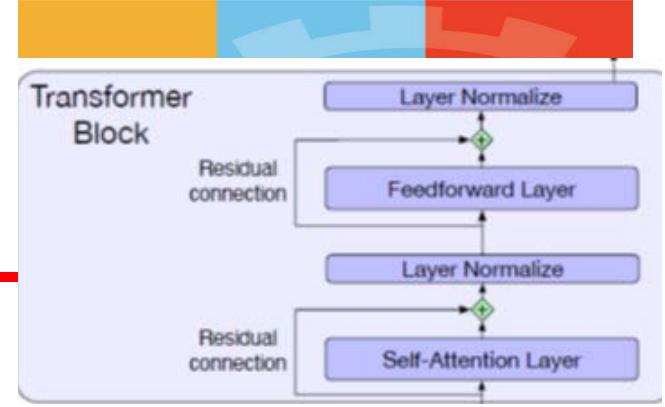
$$\begin{aligned}z &= \text{LayerNorm}(x + \text{SelfAttn}(x)) \\y &= \text{LayerNorm}(z + \text{FFNN}(z))\end{aligned}$$

$$\mu = \frac{1}{d_h} \sum_{i=1}^{d_h} x_i$$

$$\sigma = \sqrt{\frac{1}{d_h} \sum_{i=1}^{d_h} (x_i - \mu)^2}$$

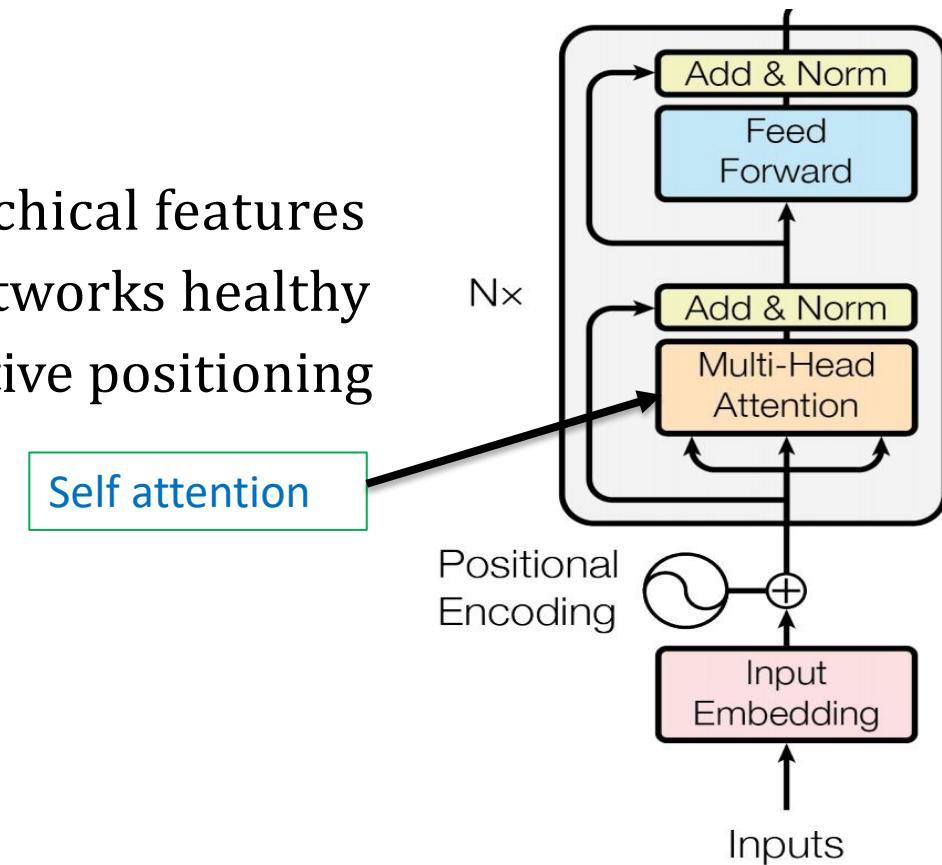
$\gamma$  and  $\beta$  are learnable parameters

$$\text{LayerNorm} = \gamma \hat{x} + \beta$$



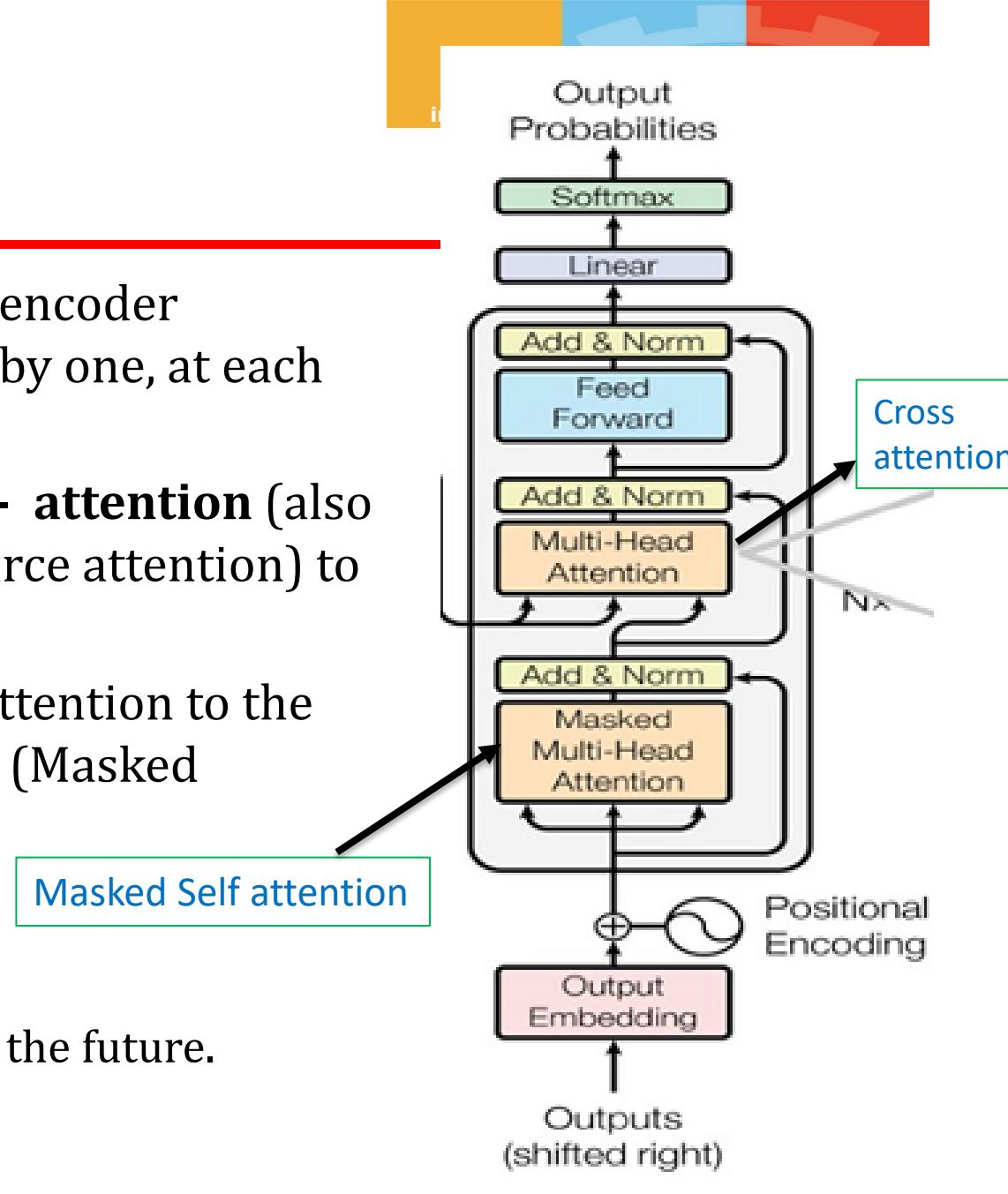
# The Transformer Encoder

- It receives an input and **builds a representation of it (its features): contextual embeddings.**
- **Bidirectional** model
- A stack of 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

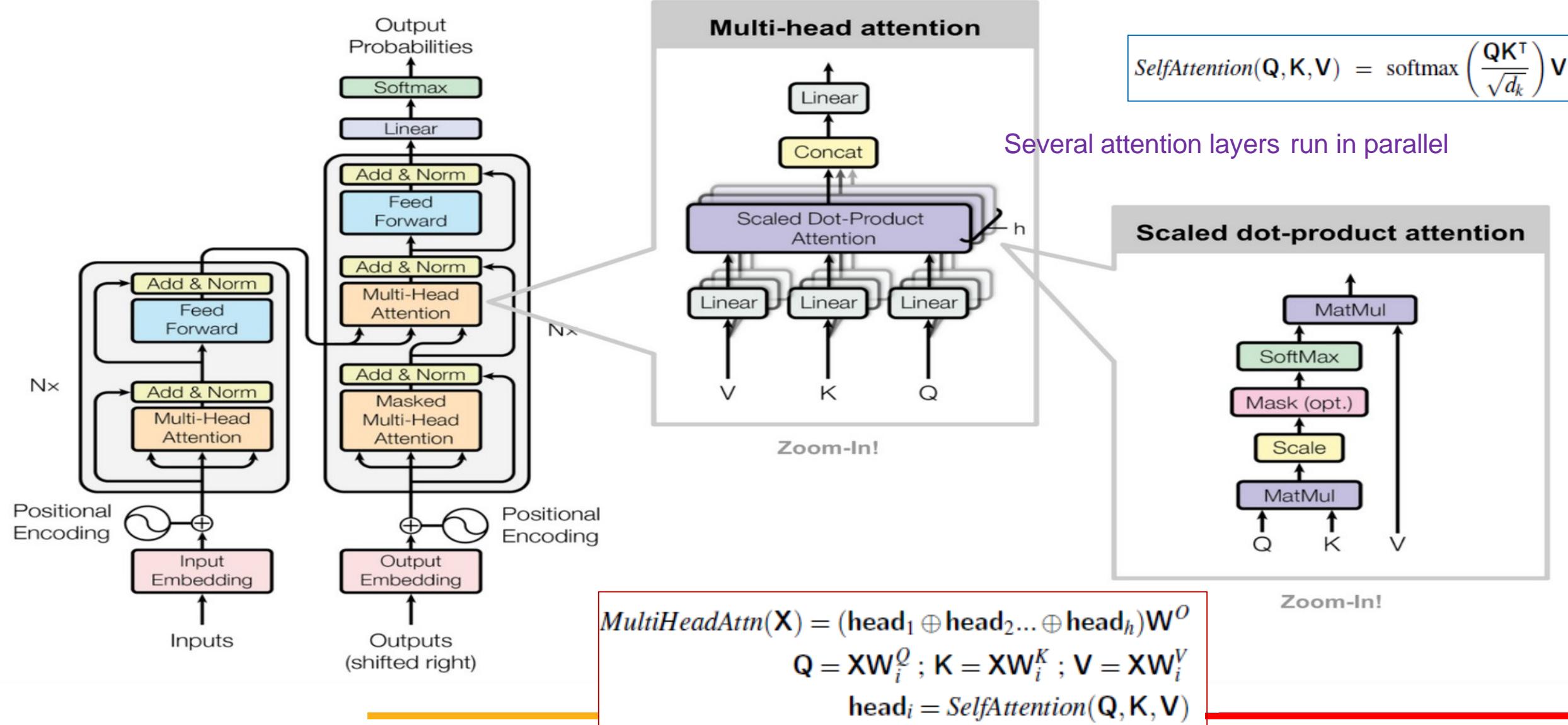


# The Transformer Decoder

- A **conditional language model** that attends to the encoder representation and generates the target words one by one, at each timestep
- 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**
- **Masking:**
  - In order to parallelize operations while not looking at the future.
  - Keeps information about the future from “leaking” to the past.



# The Transformer Encoder-Decoder

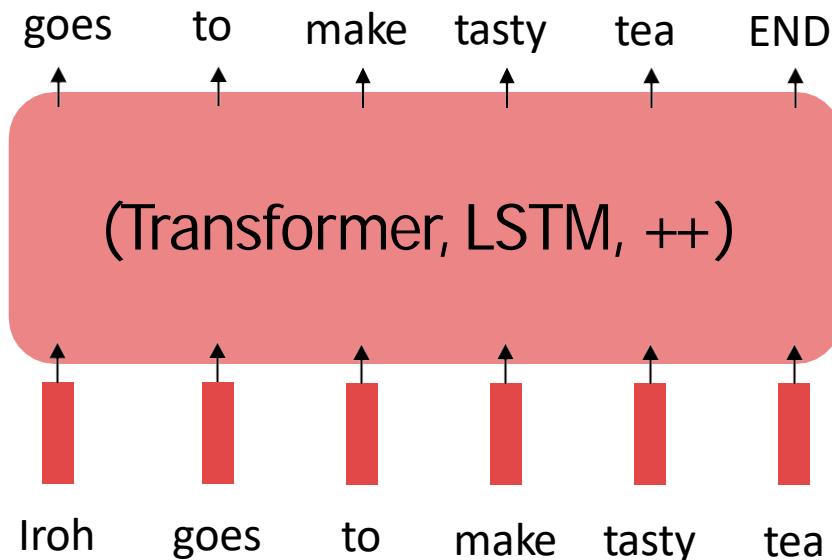


# The Pre training / Fine tuning Paradigm

Pretraining can improve NLP applications by serving as parameter initialization.

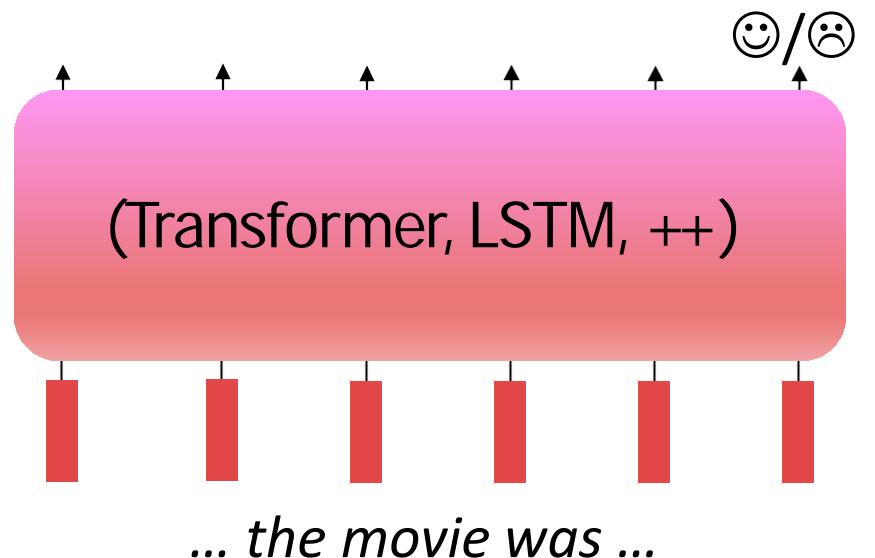
## Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



## Step 2: Finetune (on your task)

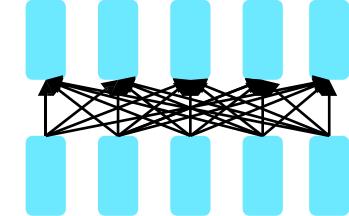
Not many labels; adapt to the task!



# Pre-training for three types of architectures

## Encoder-only models

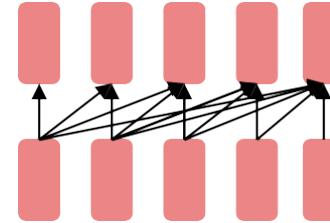
- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?
- Also known as *auto-encoding models*
- For tasks that require understanding of the input, e.g sentence classification and named entity recognition
- Representatives of this family of models include:
  - ALBERT (have smaller parameter sizes when compared to the corresponding BERT models),
  - BERT,
  - DistilBERT (DistilBERT is a small, fast, cheap and light Transformer model trained by distilling Bert base)



# Pre-training for three types of architectures

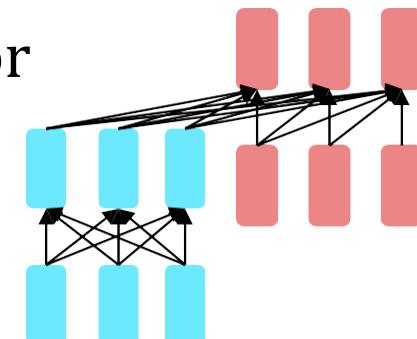
## Decoder-only models

- 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:
  - GPT (Generative Pre-trained Transformer), GPT-2, GPT-3 etc



## Encoder-decoder models or sequence-to-sequence models

- For generative tasks that require an input, such as translation or summarization.
- BART (Bidirectional and Auto-Regressive Transformer)



# Contextual embeddings

- Word embeddings (word2vec, GloVe) are often *pre-trained* on text corpus from co-occurrence statistics
- Problem: Word embeddings are applied in a context free manner



[ 0.3, 0.2, -0.8, ... ]

- Solution: Train contextual representations on text corpus

[ 0.9, -0.2, 1.6, ... ]

↑  
open a bank account

[ -1.9, -0.4, 0.1, ... ]

↑  
on the river bank

- Given a pre trained language model and a novel input sentence, model outputs contextual embeddings for each token in the input.

# 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.

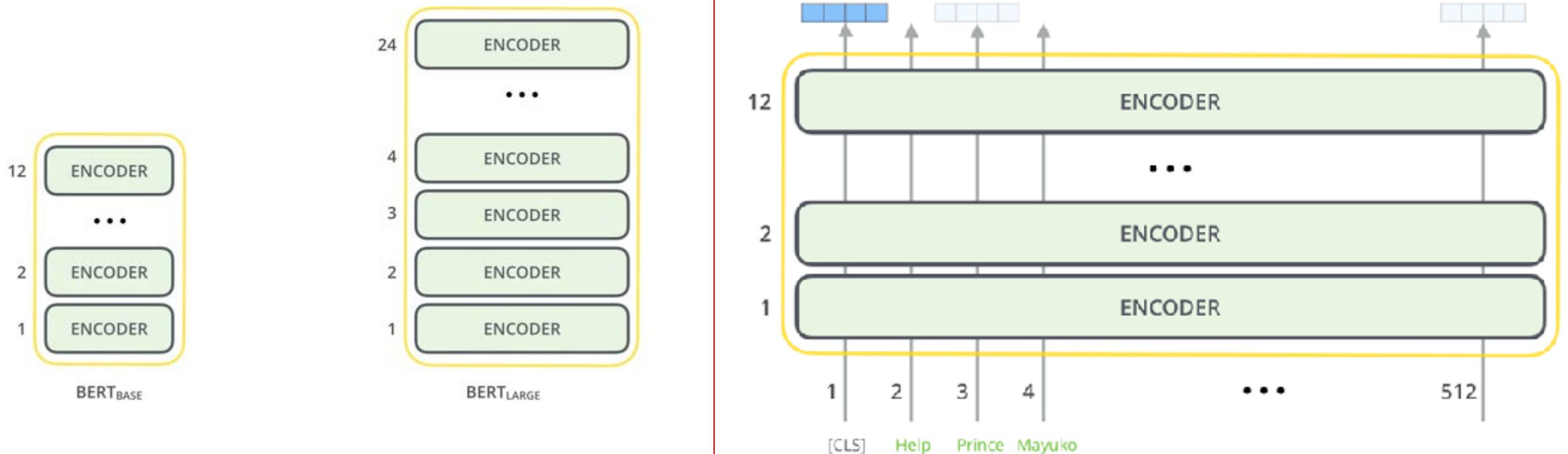
# BERT: Bidirectional Encoder Representations from Transformers



## 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



Model input dimension 512

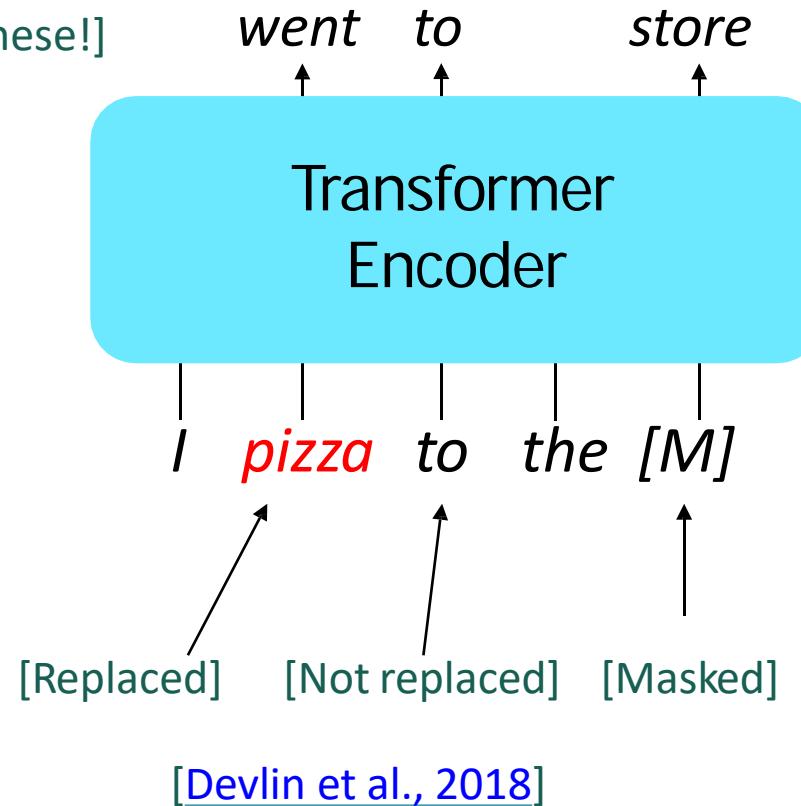
# BERT Pre training Task 1: masked words

- Instead of trying to predict the next word, the model learns to perform a **fill-in-the-blank task**, [Predict these!] technically called the **cloze task** e.g.

Please turn your homework\_\_\_\_\_.

Please turn \_\_\_\_\_ homework in.

- 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

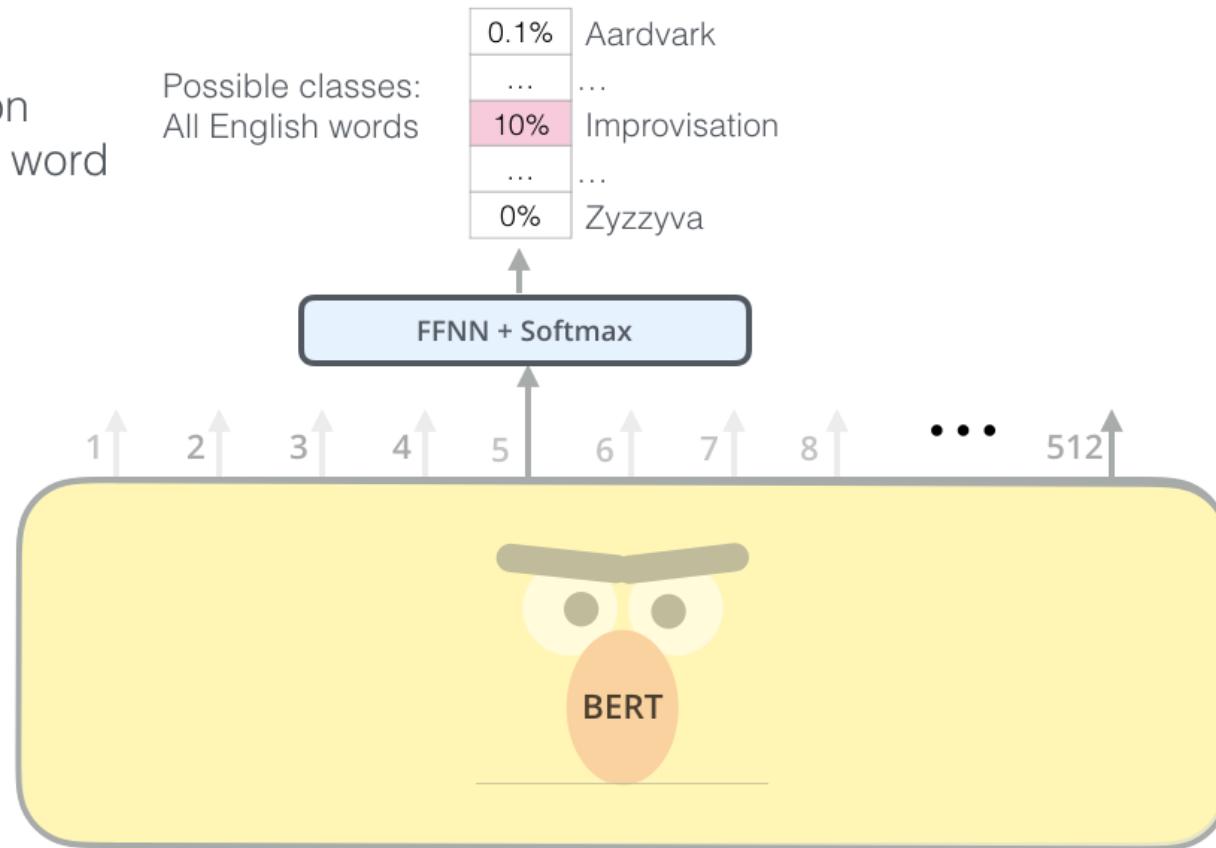


Use the output of the masked word's position to predict the masked word

Possible classes:  
All English words

|      |               |
|------|---------------|
| 0.1% | Aardvark      |
| ...  | ...           |
| 10%  | Improvisation |
| ...  | ...           |
| 0%   | Zyzyva        |

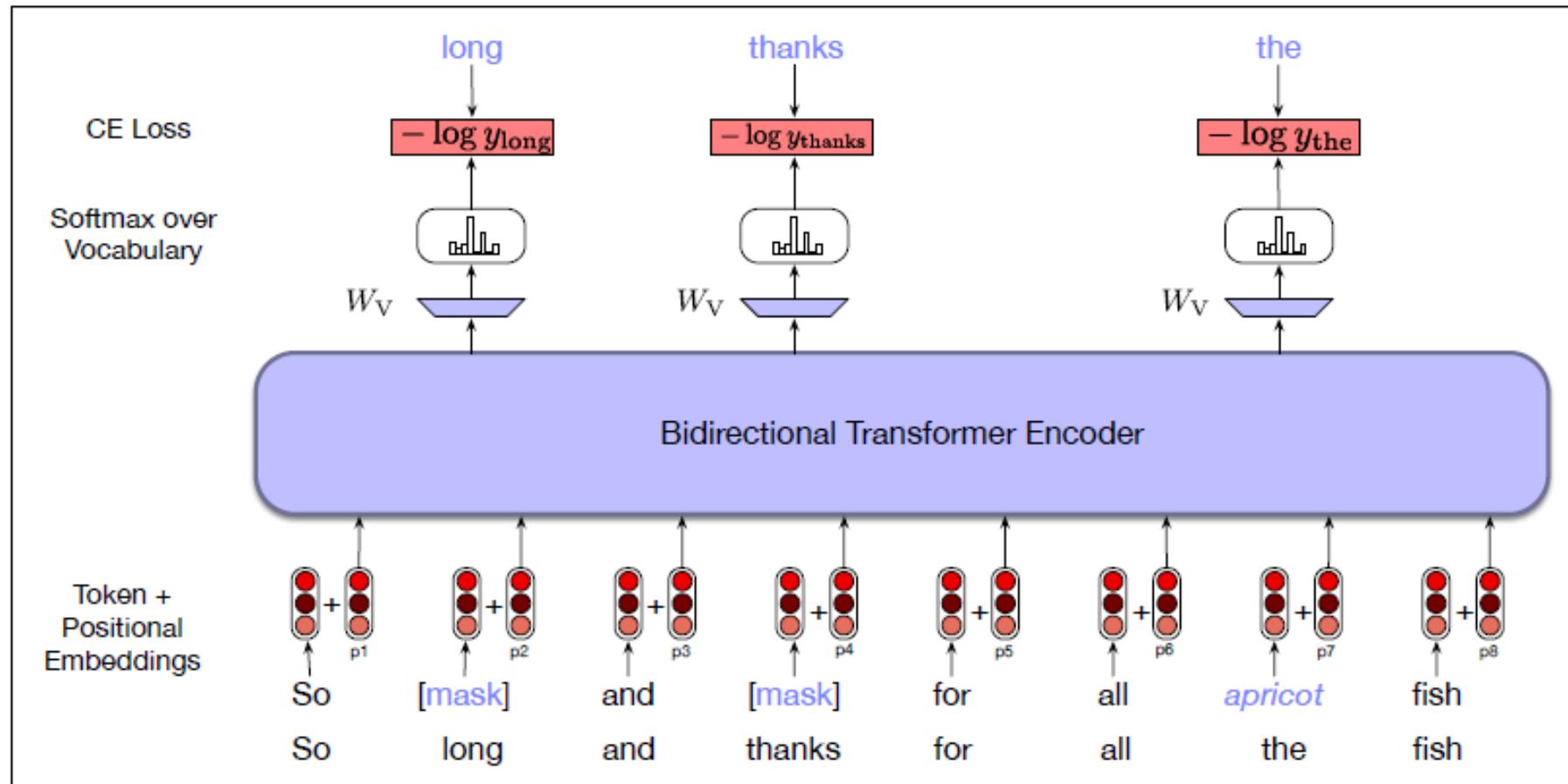
Randomly mask  
15% of tokens



Input

[CLS] Let's stick to improvisation in this skit

# Masked Language Modeling (MLM) output for BERT



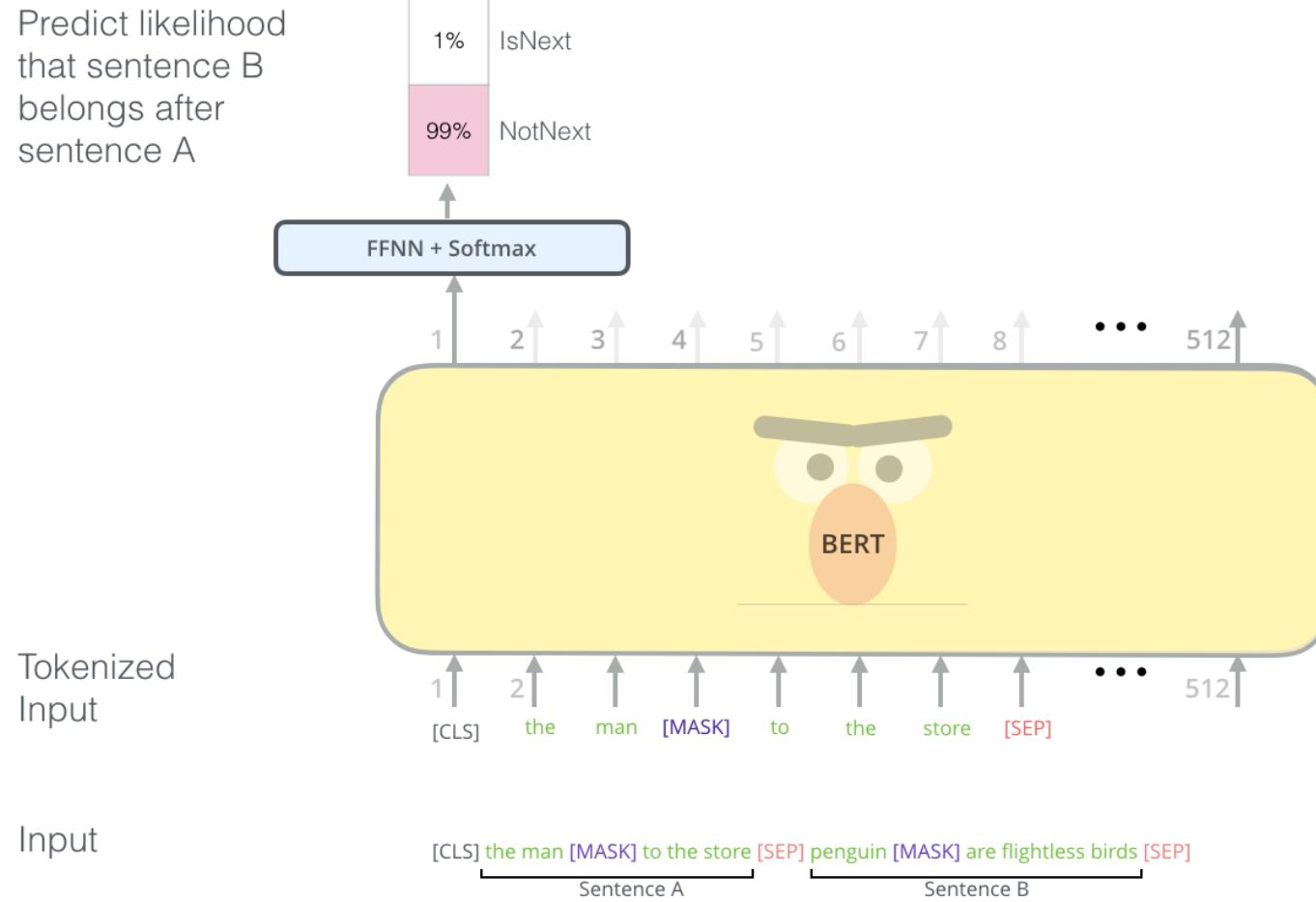
**Figure 11.5** Masked language model training. In this example, three of the input tokens are selected, two of which are masked and the third is replaced with an unrelated word. The probabilities assigned by the model to these three items are used as the training loss. (In this and subsequent figures we display the input as words rather than subword tokens; the reader should keep in mind that BERT and similar models actually use subword tokens instead.)

# BERT Pretraining Task 2: Next Sentence Prediction (NSP)



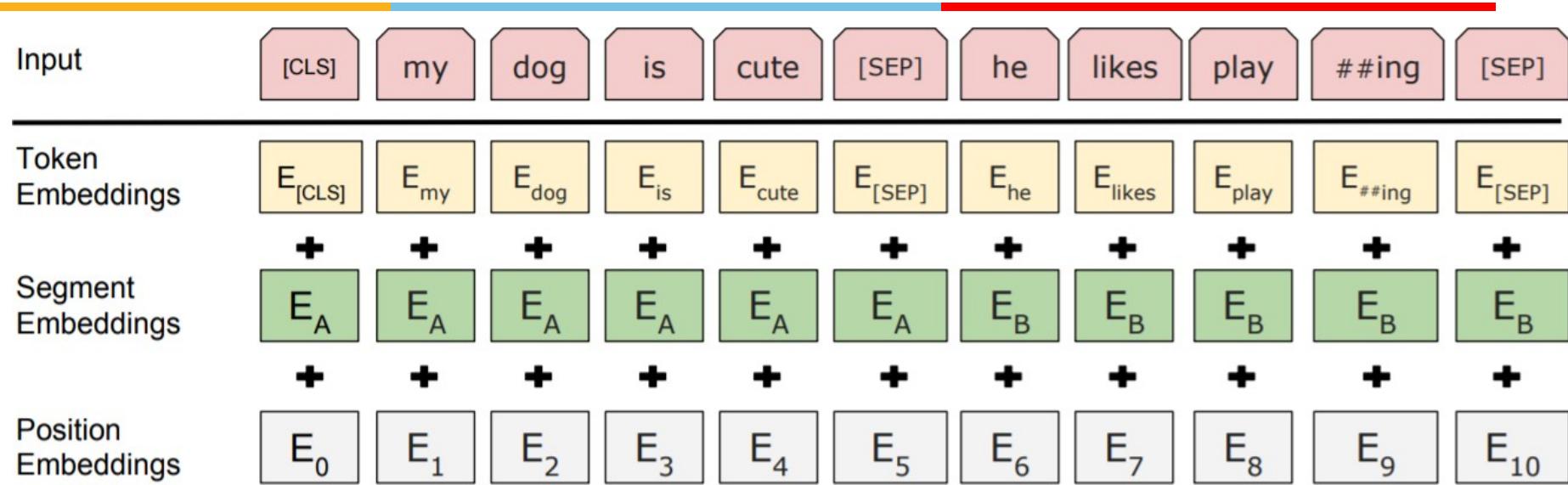
- Determining the relationship between pairs of sentences
  - paraphrase detection (detecting if two sentences have similar meanings)
  - entailment (detecting if the meanings of two sentences entail or contradict each other)
  - discourse coherence (deciding if two neighboring sentences form a coherent discourse)
- 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?**

# BERT Pretraining Task 2: Next Sentence Prediction (NSP)



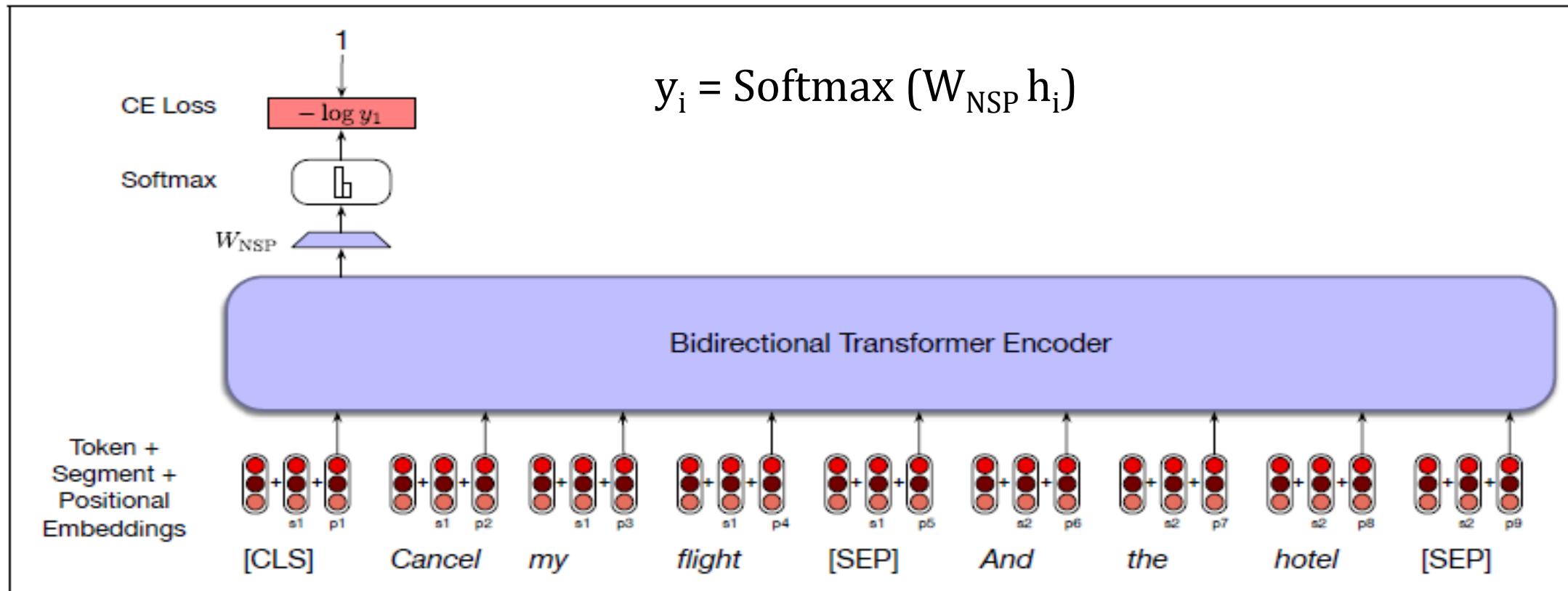
- Two new tokens to the input representation : CLS and SEP
  - **50% the second sentence of a pair was randomly selected from elsewhere in the corpus**
  - Embeddings representing the first and second segments of the input are added to the word and positional embeddings to allow the model to more easily distinguish the input sentences.

# 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

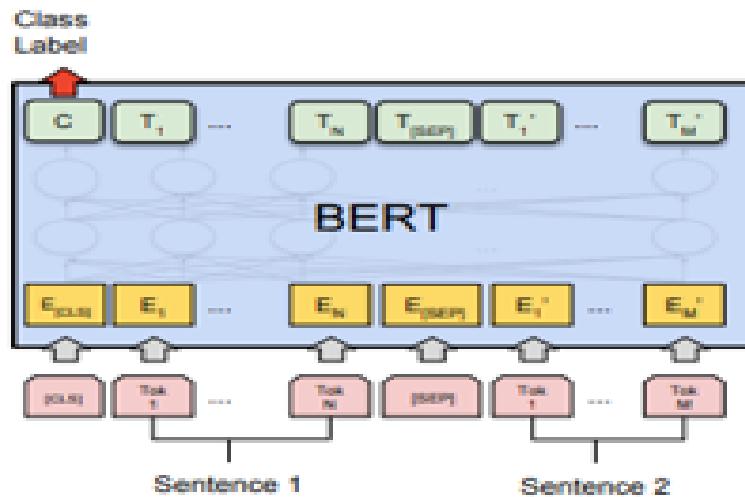
# NSP output for BERT



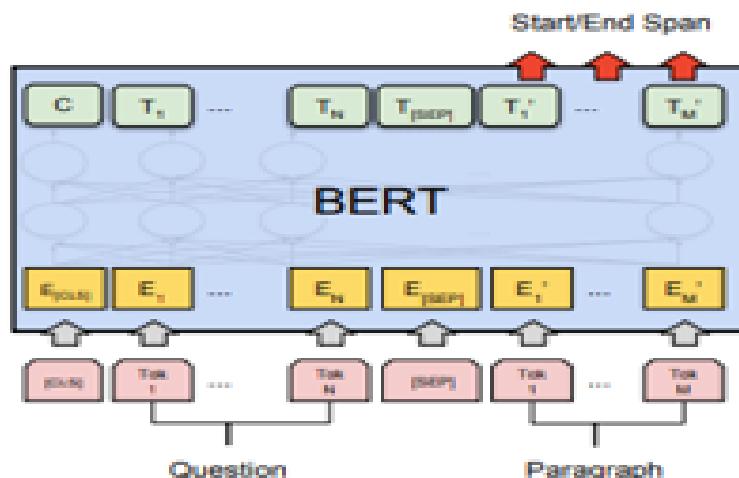
**Figure 11.7** An example of the NSP loss calculation.

output vector from the final layer associated with the [CLS] token represents the next sentence prediction

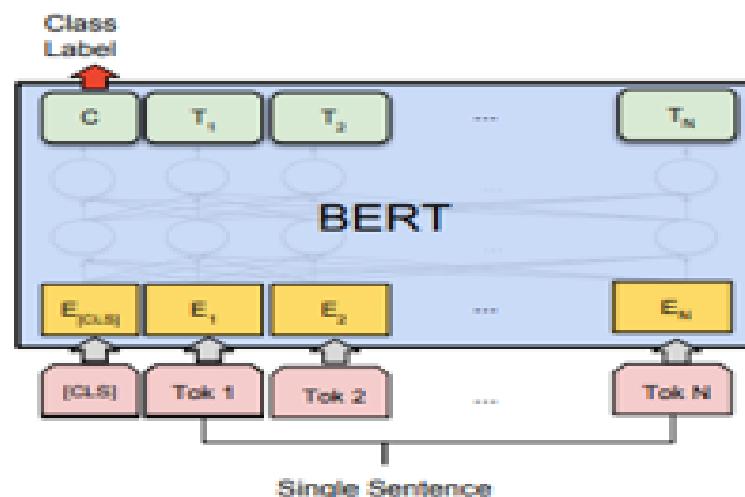
# BERT for different task



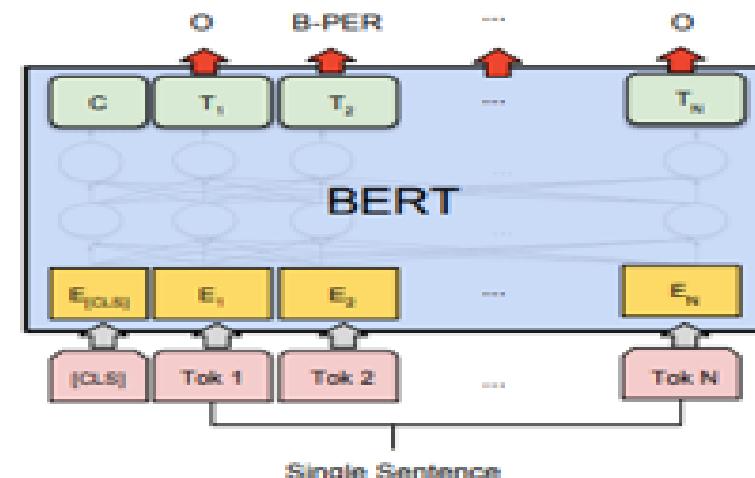
(a) Sentence Pair Classification Tasks:  
MNLI, QQP, QNLI, STS-B, MRPC,  
RTE, SWAG



(c) Question Answering Tasks:  
SQuAD v1.1



(b) Single Sentence Classification Tasks:  
SST-2, CoLA

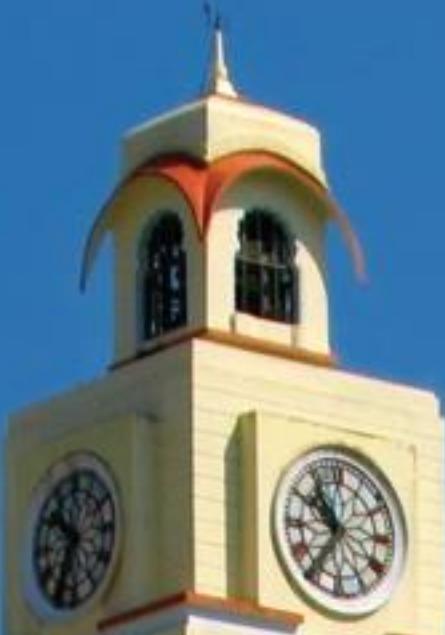


(d) Single Sentence Tagging Tasks:  
CoNLL-2003 NER

# 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



**BITS** Pilani  
Pilani Campus

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay Monash University Australia  
[Chetana.gavankar@pilani.bitspilani.ac.in](mailto:Chetana.gavankar@pilani.bitspilani.ac.in)



## **Session 15 Text Summarization**

### **Date – 23<sup>rd</sup> September 2023**

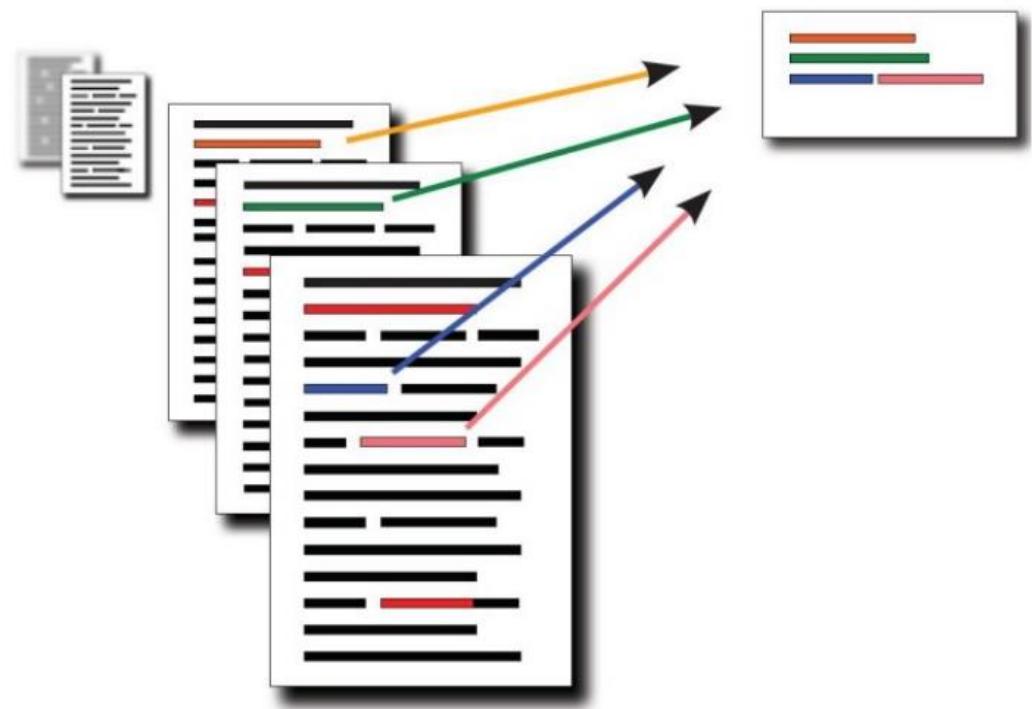
These slides are prepared by the instructor, with grateful acknowledgement of Prof. Dan Jurafsky and many others who made their course materials freely available online.

# Session Content

- What is Text Summarization?
- Applications
- Type of Summarization
  - Single Document Summarization
  - Multidocument Summarization
  - Extractive Summarization
  - Abstractive Summarization
  - Generic Summarization
  - Query focused summarization
- Stages of Summarization
  - Content Selection
  - Information Ordering
  - Sentence Realization
- Neural Text Summarization

# 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

# 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 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

# 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 for Question Answering Snippets



- Create **snippets** summarizing a web page for a query
  - Google: 156 characters (about 26 words) plus title and link

Google what is die brücke?

Search About 5,910,000 results (0.28 seconds)

---

|              |   |
|--------------|---|
| Everything   | <a href="#">Die Brücke - Wikipedia, the free encyclopedia</a><br>en.wikipedia.org/wiki/Die_Brücke   |
| Images       | <a href="#">Die Brücke</a> (The Bridge) was a group of German expressionist artists formed in Dresden in 1905, after which the Brücke Museum in Berlin was named. Founding ...<br>You've visited this page 5 times. Last visit: 4/16/12 |
| Maps         |   |
| Videos       |   |
| News         | <a href="#">Die Brücke (film) - Wikipedia, the free encyclopedia</a><br>en.wikipedia.org/wiki/Die_Brücke_(film)   |
| Shopping     | <a href="#">Die Brücke</a> (English: The Bridge) is a 1959 West German film directed by Austrian filmmaker Bernhard Wicki. It is based on the eponymous 1958 novel by ...   |
| Applications |   |
| More         | <a href="#">Die Brücke - Die Brücke Art</a><br>www.huntfor.com/arhistory/c20th/diebrucke.htm  |

# 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

# Simple baseline take the first sentence



Google what is die brücke?

Search About 5,910,000 results (0.28 seconds)

Everything [Die Brücke - Wikipedia, the free encyclopedia](#)  
[en.wikipedia.org/wiki/Die\\_Brücke](http://en.wikipedia.org/wiki/Die_Brücke)  
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 ...

Images

Maps

## Die Brücke

From Wikipedia, the free encyclopedia

*For other uses, see [Die Brücke \(disambiguation\)](#).*

**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 members were [Fritz Bleyl](#), [Erich Heckel](#), [Ernst Ludwig Kirchner](#) and [Karl Schmidt-Rottluff](#). Later members were [Emil Nolde](#), [Max Pechstein](#) and [Otto Mueller](#). The seminal group had a major impact on the evolution of modern art in the 20th century and the creation of expressionism.<sup>[1]</sup>

Die Brücke is sometimes compared to the [Fauves](#). Both movements shared interests in primitivist art. Both

# Query focused summary

Was cast-metal movable type invented in korea?

About 591,000 results (0.14 seconds)

## [Movable type - Wikipedia, the free encyclopedia](#)

[en.wikipedia.org/wiki/Movable\\_type](https://en.wikipedia.org/wiki/Movable_type)

Jump to [Metal movable type](#): Transition from wood type to **metal** type occurred in 1234 ... The following description of the **Korean** font **casting** ... In the early fifteenth century, however, the **Koreans invented** a form of **movable type** that has ...

## [History of printing in East Asia - Wikipedia, the free encyclopedia](#)

[en.wikipedia.org/wiki/History\\_of\\_printing\\_in\\_East\\_Asia](https://en.wikipedia.org/wiki/History_of_printing_in_East_Asia)

The following description of the **Korean** font **casting** process was recorded by the ... While **metal movable type** printing was **invented in Korea** and the oldest ...

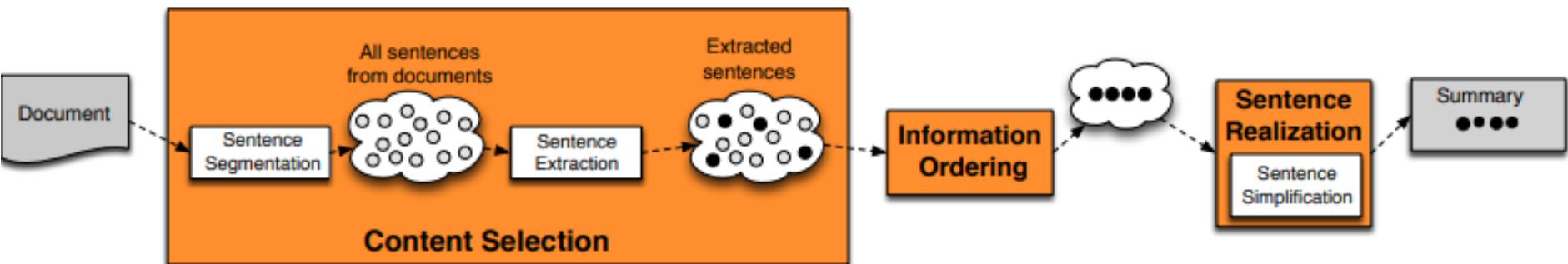
## [Korea, 1000–1400 A.D. | Heilbrunn Timeline of Art History | The ...](#)

[www.metmuseum.org/toah/ht/?period=07&region=eak](https://www.metmuseum.org/toah/ht/?period=07&region=eak)

The **invention** and use of **cast-metal movable type in Korea** in the early thirteenth century predates by two centuries Gutenberg's **invention** of metal **movable type** ...

# 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



---

# Stage 1: Content Selection

# Frequency as indicator of importance

---

The topic of a document will be repeated many times

In multi-document summarization, important content is repeated in different sources

# Greedy frequency method

---

Compute word probability from input

Compute sentence weight as function of word probability

Pick best sentence

## *Unsupervised content selection; Luhn (1958)*

### *Intuition*

Choose sentences that have salient or informative words

stop word removal

2

3

4

5

### *Two approaches to define salient words*

- *tf-idf*: weigh each word  $w_i$  in document  $j$  by tf-idf

$$\text{weight}(w_i) = \text{tf}_{ij} \times \text{idf}_i$$

- *Topic signatures*: choose a smaller set of salient words, specific to that domain

$$\text{weight}(w_i) = 1 \text{ if } w_i \text{ is a specific term (use mutual information)}$$

### *Weighing a sentence*

$$\text{weight}(s) = \frac{1}{|S|} \sum_{w \in S} \text{weight}(w)$$

# 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  $T_k$

$$idf_k = \log\left(\frac{N}{n_k}\right)$$

# Using graph representations

---

## Nodes

- Sentences
- Discourse entities

## Arcs

- Between similar sentences
- Between related entities

# Using graph representations

## *LexRank: A Graph-based approach*

### *Text Document*

Computation is a process following  
a well defined model ...  
A computation can be seen as a  
purely physical phenomena ...  
...

processing

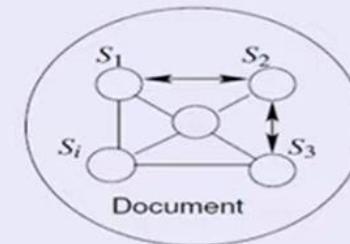
$S_1 \rightarrow \{(computation, 0.1), (process, 0.15), \dots\}$   
 $S_2 \rightarrow \{(computation, 0.1), (seen, 0.05), \dots\}$   
 $S_3 \rightarrow \dots$

Machine-readable format

### Document Representation

### *Underlying Hypothesis*

Sentences that convey the  
theme of the document are  
more similar to each other



### Finding the most salient sentences

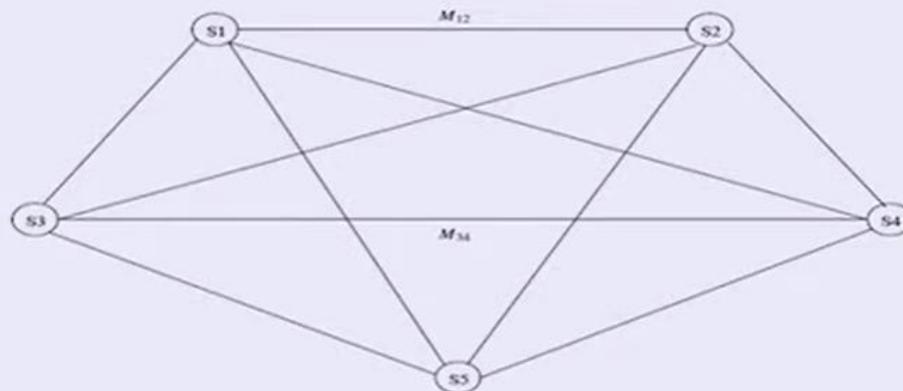


# Using graph representations

## *Sentence Centrality Measure*

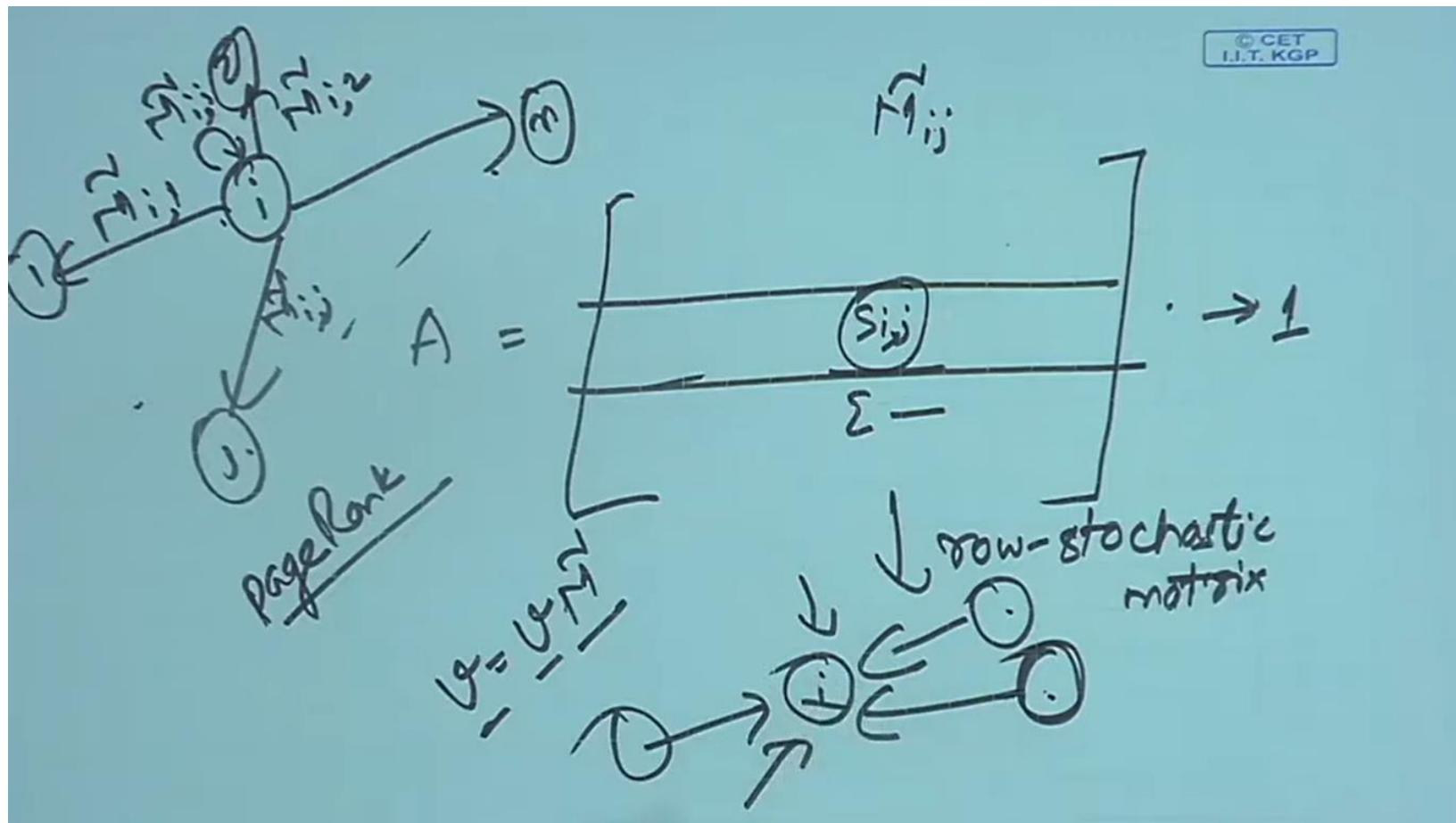
*Finding the most salient sentences*

PageRank based algorithm is used to compute the sentence centrality vector  $I$ .



$$\tilde{M} = \begin{bmatrix} 0.0 & 0.5 & 0.0 & 0.4 & 0.1 \\ 0.5 & 0.0 & 0.5 & 0.0 & 0.0 \\ 0.0 & 0.5 & 0.0 & 0.5 & 0.0 \\ 0.4 & 0.0 & 0.4 & 0.0 & 0.2 \\ 0.3 & 0.0 & 0.0 & 0.7 & 0.0 \end{bmatrix}$$

# Using graph representations



<https://www.youtube.com/watch?v=1XBOK-I8Gc8&t=133s>

# 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**

# How to deal with redundancy?

---

Author JK Rowling has won her legal battle in a New York court to get an unofficial Harry Potter encyclopaedia banned from publication.

A U.S. federal judge in Manhattan has sided with author J.K. Rowling and ruled against the publication of a Harry Potter encyclopedia created by a fan of the book series.

- Shallow techniques not likely to work well

# Global optimization for content selection

---

What is the best summary? vs What is the best sentence?

Form all summaries and choose the best

- What is the problem with this approach?

# LLR+MMR: Choosing informative yet non redundant sentences

---

One of many ways to combine the intuitions of MMR:

1. Score each sentence based on MMR(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

# Maximal Marginal Relevance MMR



- An iterative method for content selection from multiple documents
- Iteratively (greedily) choose the best sentence to insert in the summary/answer so far:
  - **Relevant:** Maximally relevant to the user's query
    - high cosine similarity to the query
  - **Novel:** Minimally redundant with the summary/answer so far
    - low cosine similarity to the summary
- Stop when desired length

$$\hat{s}_{MMR} = \max_{s \in D} \lambda sim(s, Q) - (1-\lambda) \max_{s \in S} sim(s, S)$$

# Optimization based approach for summarization

---

- Let us define document  $D$  with  $t_n$  textual units

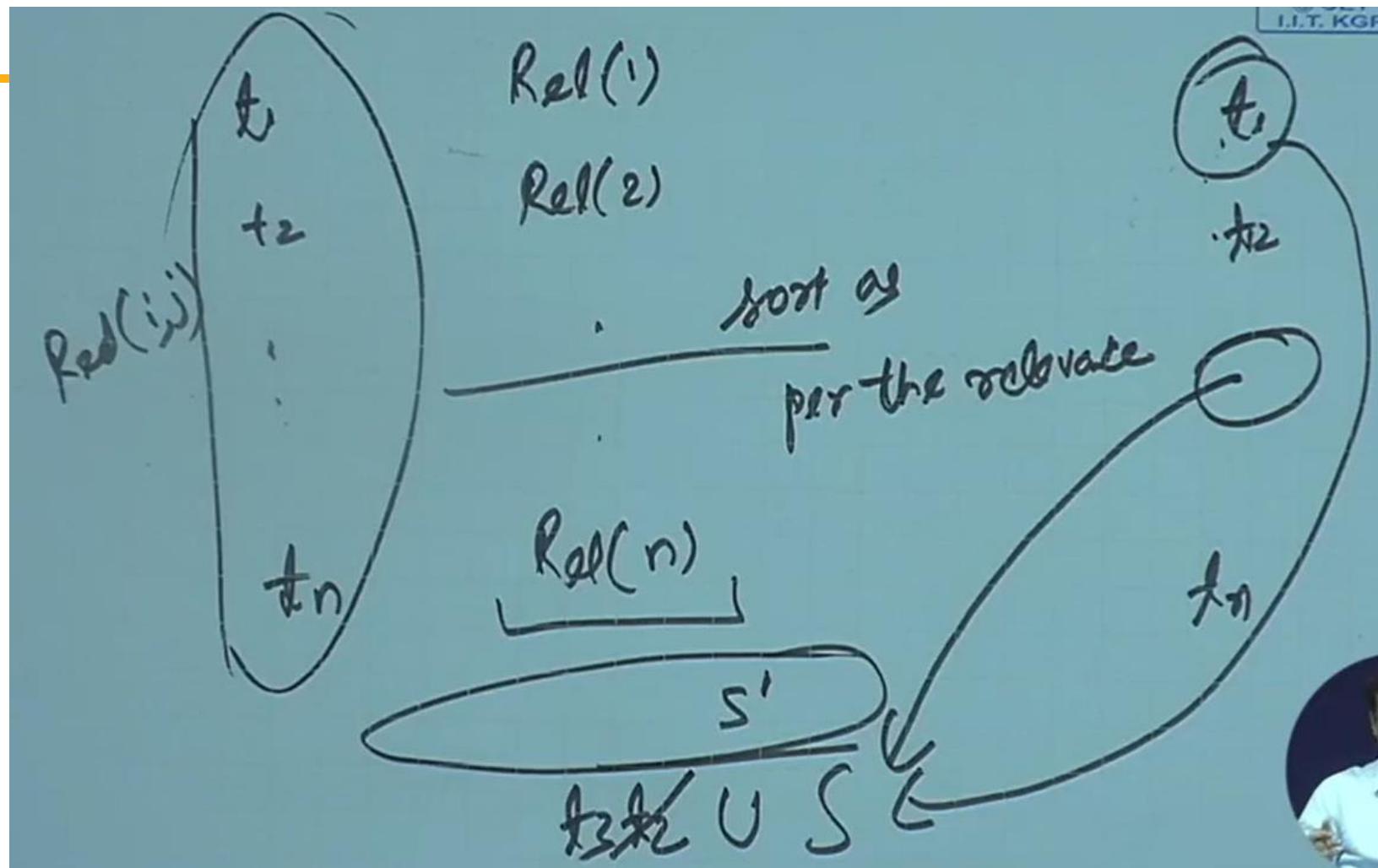
$$D = t_1, t_2, \dots, t_{n-1}, t_n$$

- Let  $Rel(i)$  be the relevance of  $t_i$  to be in the summary
- Let  $Red(i, j)$  be the redundancy between  $t_i$  and  $t_j$
- Let  $I(i)$  be the length of  $t_i$

# Optimization based approach for summarization

---

- The inference problem is to select a subset  $S$  of textual units from  $D$  such that summary score of  $S$ , i.e.,  $s(S)$ , is maximized.
- $$S = \arg \max_{S \subseteq D} \left[ \sum_{i \in S} Rel(i) - \sum_{i, j \in S, i < j} Red(i, j) \right]$$
such that  $\sum_{i \in S} l(i) \leq K$ , where  $k$  denotes the maximum length of the summary



# Algorithm

1. Sort  $D$  so that  $Rel(i) > Rel(i + 1) \forall i$
2.  $S = \{t_1\}$
3. while  $\sum_{t_i \in S} l(i) < K$
4.      $t_j = \arg \max_{t_j \in D - S} s(S \cup \{t_j\})$
5.      $S = S \cup \{t_j\}$
6. return  $S$

---

## Stage 2: Information Ordering

# Information ordering

---

In what order to present the selected sentences?

- An article with permuted sentences will not be easy to understand

Very important for multi-document summarization

- Sentences coming from different documents

# 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 |

# Answering harder questions: Query focused multi-document summarization

---



- The (bottom up) snippet method
  - • Find a set of relevant documents
  - • Extract informative sentences from the documents
  - • Order and modify the sentences into an answer
- The (top down) information extraction method
  - build specific answers for different question types:
    - definition questions
    - biography questions
    - certain medical questions

# Definition questions

---

Q: What is water spinach?

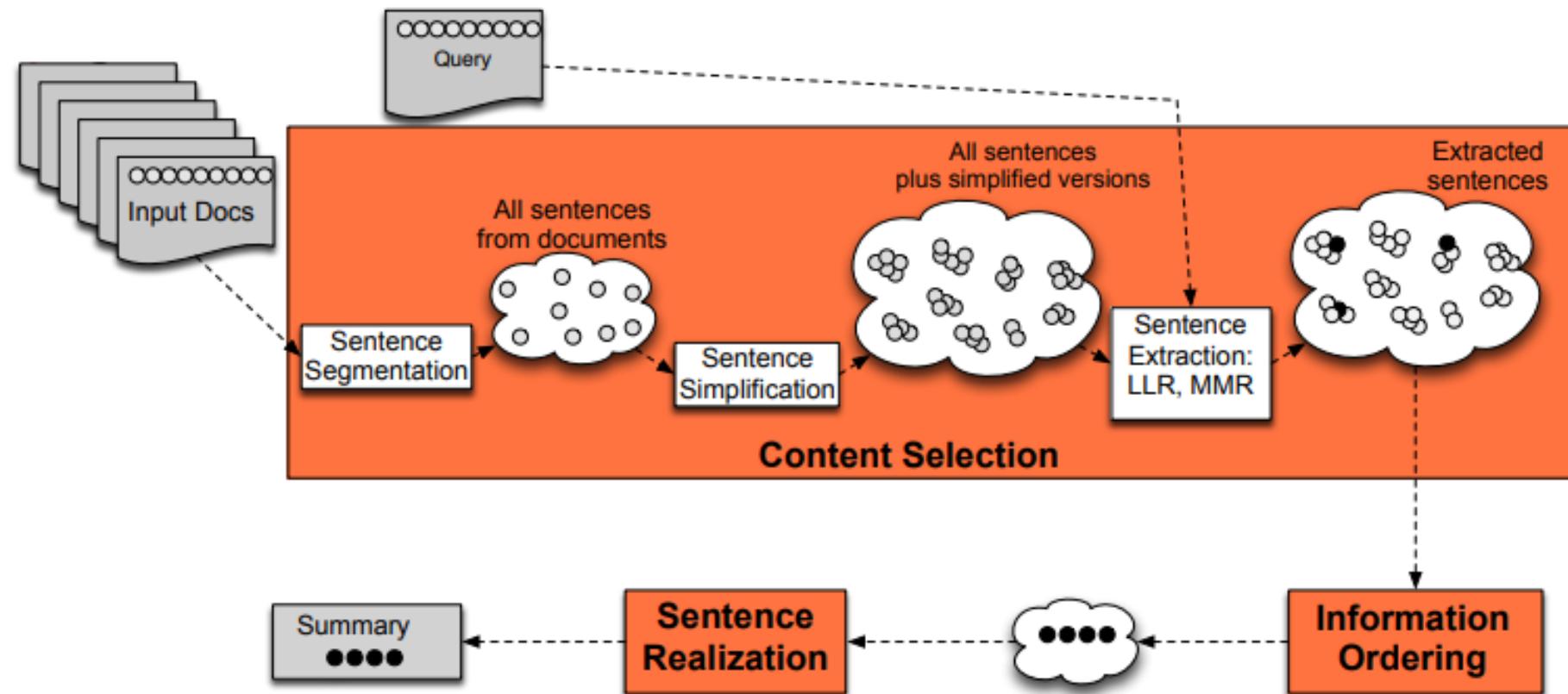
A: Water spinach (*Ipomoea aquatica*) is a semiaquatic leafy green plant with long hollow stems and spear or heart shaped leaves, widely grown throughout Asia as a leaf vegetable. The leaves and stems are often eaten fried flavored with salt or in soups. Other common names include morning glory vegetable, kangkong (Malay). It is not related to spinach, but is closely related to sweet potato and convolvulus.

# Complex Questions

---

1. How is compost made and used for gardening (including different types of compost, their uses, origins and benefits)?
  2. What causes train wrecks and what can be done to prevent them?
  3. Where have poachers endangered wildlife, what wildlife has been endangered and what steps have been taken to prevent poaching?
  4. What has been the human toll in death or injury of tropical storms in recent years?
-

# Query Focused Multi Document Summarization



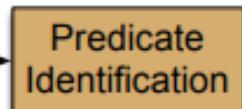
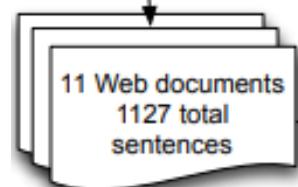
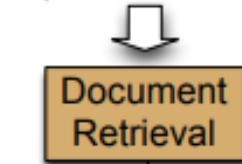
# Simplifying sentences

Simplest method: parse sentences, use rules to decide which modifiers to prune  
(more recently a wide variety of machine-learning methods)

|                                   |   |
|-----------------------------------|---|
| <b>appositives</b>                | Rajam, <del>28, an artist who was living at the time in Philadelphia</del> , found the inspiration in the back of city magazines.                   |
| <b>attribution clauses</b>        | Rebels agreed to talks with government officials, <del>international observers said Tuesday</del> .   |
| <b>PPs without named entities</b> | The commercial fishing restrictions in Washington will not be lifted unless the salmon population increases <del>[PP to a sustainable number]</del> |
| <b>initial adverbials</b>         | <del>"For example", "On the other hand", "As a matter of fact", "At this point"</del>   |

# Architecture for complex question answering: definition questions

"What is the Hajj?"  
(Ndocs=20, Len=8)

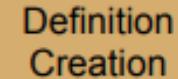


The Hajj, or pilgrimage to Makkah [Mecca], is the central duty of Islam. More than two million Muslims are expected to take the Hajj this year. Muslims must perform the hajj at least once in their lifetime if physically and financially able. The Hajj is a milestone event in a Muslim's life. The annual hajj begins in the twelfth month of the Islamic year (which is lunar, not solar, so that hajj and Ramadan fall sometimes in summer, sometimes in winter). The Hajj is a week-long pilgrimage that begins in the 12th month of the Islamic lunar calendar. Another ceremony, which was not connected with the rites of the Ka'ba before the rise of Islam, is the Hajj, the annual pilgrimage to 'Arafat, about two miles east of Mecca, toward Mina...

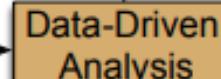
## 9 Genus-Species Sentences

The Hajj, or pilgrimage to Makkah (Mecca), is the central duty of Islam.  
The Hajj is a milestone event in a Muslim's life.  
The hajj is one of five pillars that make up the foundation of Islam.

383 Non-Specific Definitional sentences



Sentence clusters,  
Importance ordering



# Automatic summary edits

---

Some expressions might not be appropriate in the new context

- References:
  - he
  - Putin
  - Russian Prime Minister Vladimir Putin
- Discourse connectives
  - However, moreover, subsequently

Requires more sophisticated NLP techniques

# Before

---

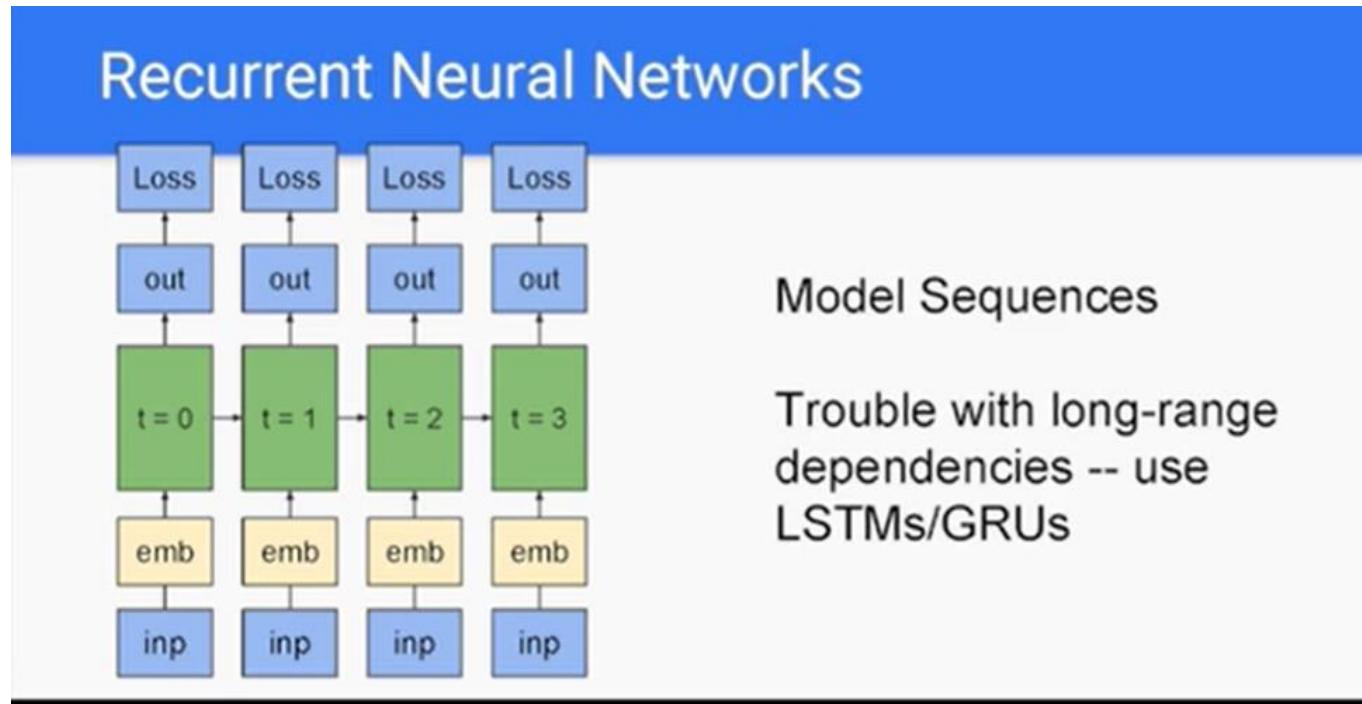
Pinochet was placed under arrest in London Friday by British police acting on a warrant issued by a Spanish judge. Pinochet has immunity from prosecution in Chile as a senator-for-life under a new constitution that his government crafted. Pinochet was detained in the London clinic while recovering from back surgery.

# After

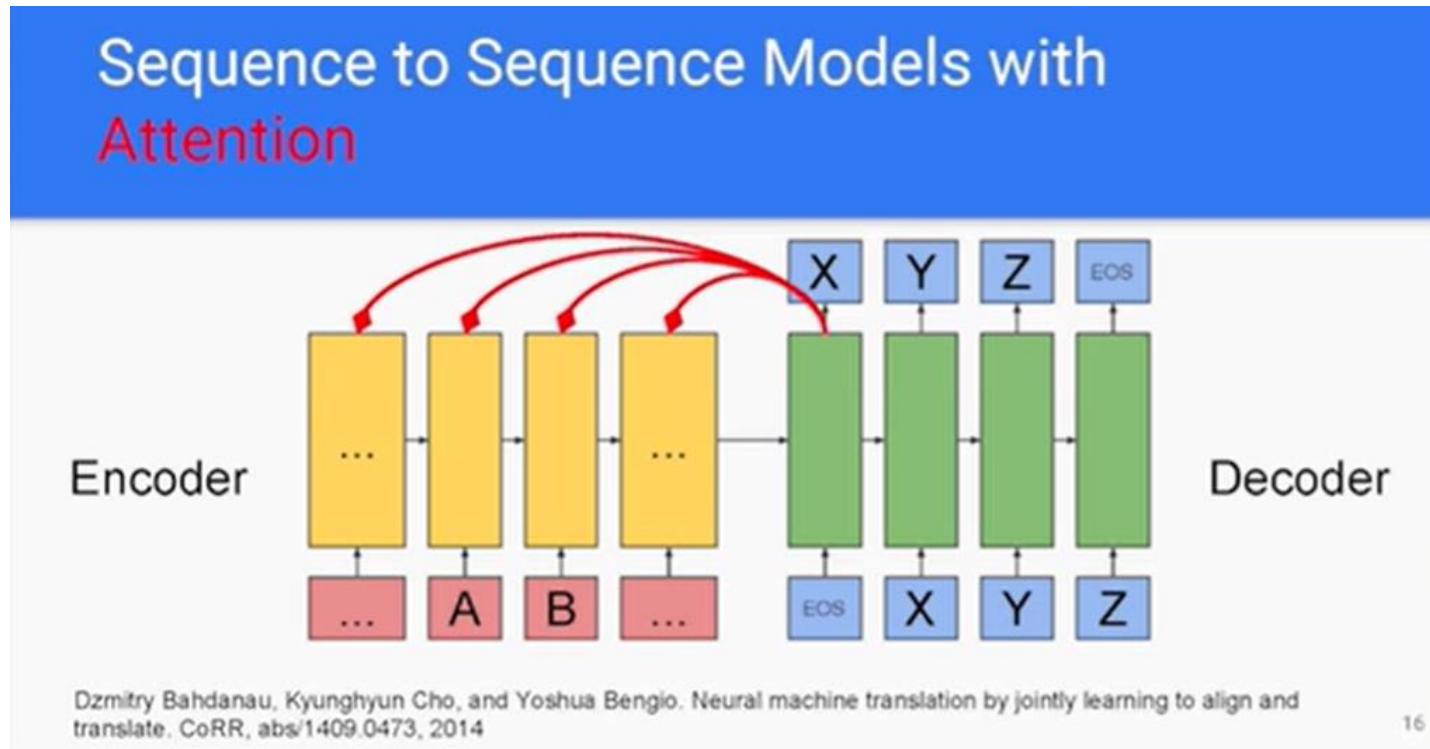
---

Gen. Augusto Pinochet, the former Chilean dictator, was placed under arrest in London Friday by British police acting on a warrant issued by a Spanish judge. Pinochet has immunity from prosecution in Chile as a senator-for-life under a new constitution that his government crafted. Pinochet was detained in the London clinic while recovering from back surgery.

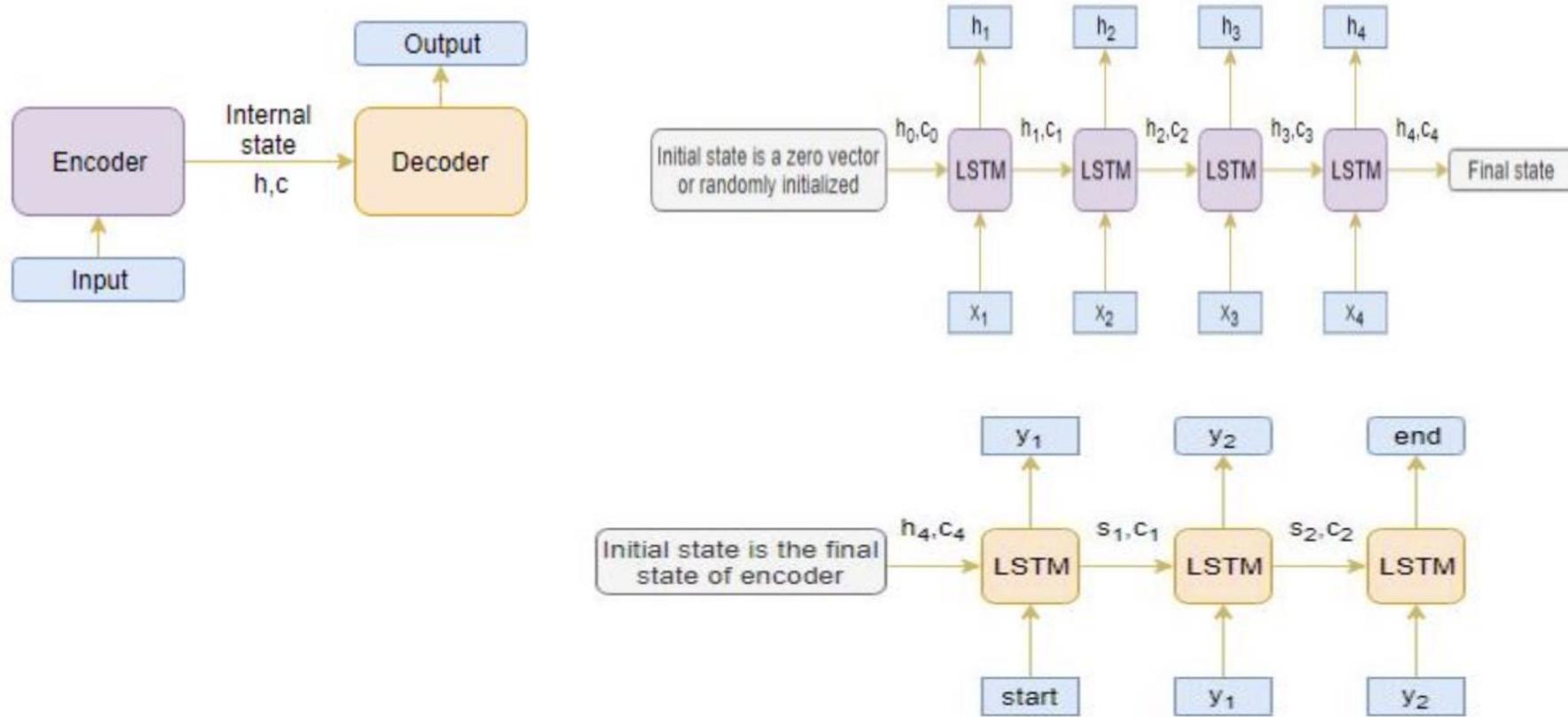
# Neural Text Summarization



# Neural Text Summarization

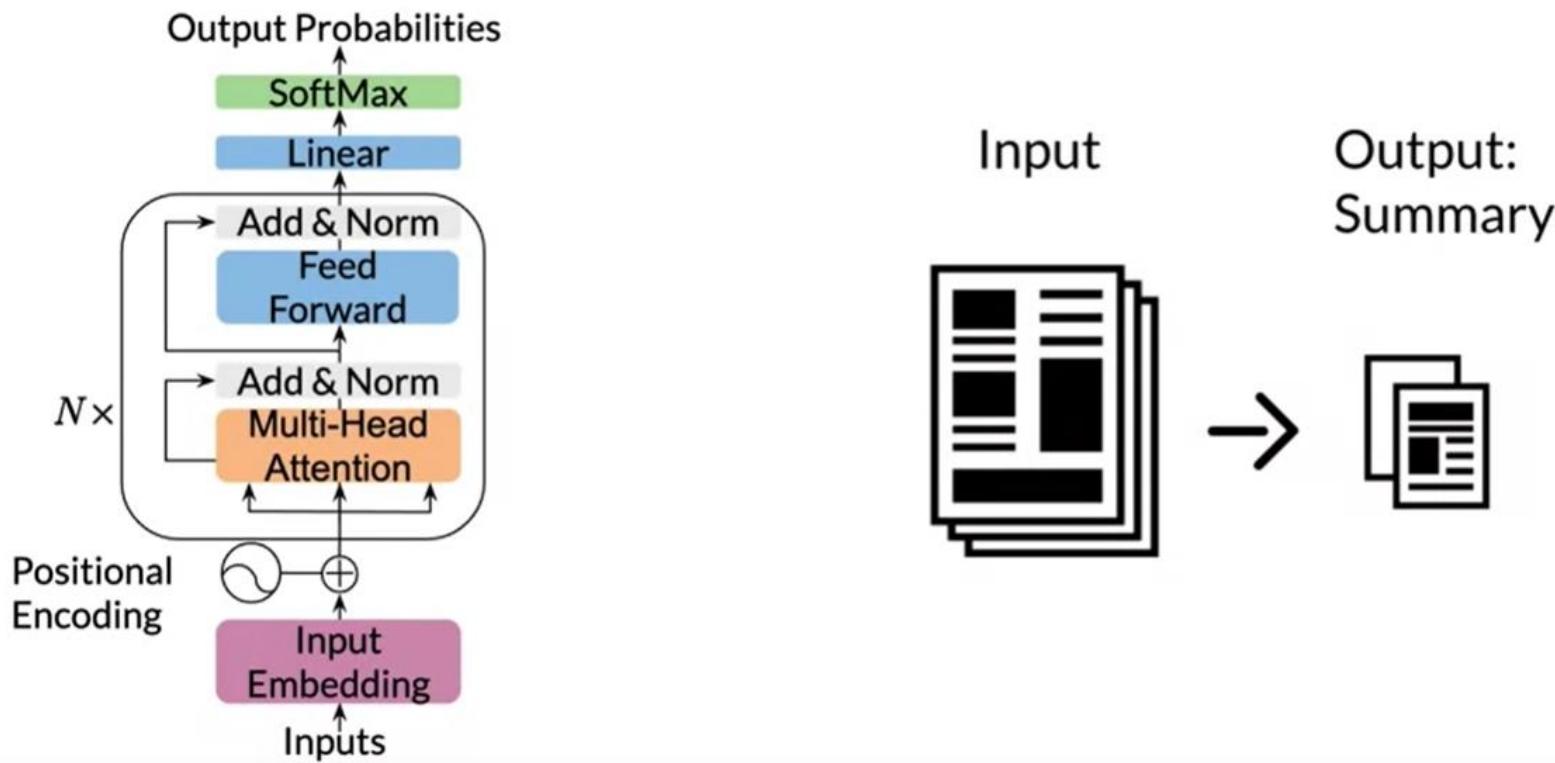


# Neural Text Summarization



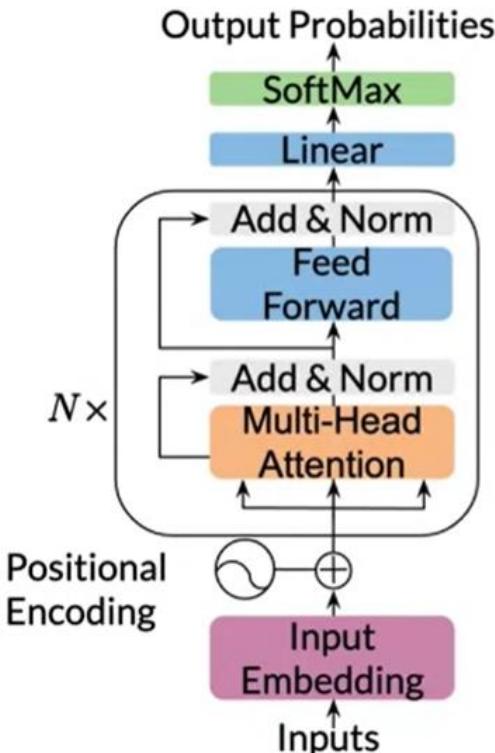
# Neural Text Summarization

## Transformer for summarization



# Neural Text Summarization

## Technical details for data processing



**Model Input:**

ARTICLE TEXT <EOS> SUMMARY <EOS> <pad> ...

**Tokenized version:**

[2, 3, 5, 2, 1, 3, 4, 7, 8, 2, 5, 1, 2, 3, 6, 2, 1, 0, 0]

Loss weights: 0s until the first <EOS> and then 1 on the start of the summary.

# 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



# 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.

Make the bi-grams  
from system words

$$\text{Rouge-2 score} = \frac{\text{Number of matching bi-grams}}{\text{Sum of Number of bi-grams in the 3 human labeled document}} = \frac{3 + 3 + 6}{10 + 9 + 9} = 12/28 = .43$$

Sum of Number of bi-grams in the 3 human labeled document

# 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/l22\\_text\\_summarization.pdf](https://people.engr.tamu.edu/huangrh/Fall16/l22_text_summarization.pdf)
- <https://vimeo.com/193652155>
- <https://www.turing.com/kb/5-powerful-text-summarization-techniques-in-python>



**BITS** Pilani  
Pilani|Dubai|Goa|Hyderabad

# Webinar 1

## NLP-Introduction & Word Embeddings

Presented by : B Radhika  
Email: b.radhika@wilp.bits-pilani.ac.in

# Outline

---

NLP Process

Introduction and Downloading of NLTK

NLTK utilities

# Overview of NLP Process

- Deal with Punctuations and Stop words
- Stemming/Lemmatization
- Tokenization
- Vectorization
- Model building
- Model evaluation and testing

# Some Basics

**Text Pre-processing:** Machine learning algorithms require numerical feature vector to perform any task. Need to pre-process it to make it ideal for work.

- Converting the entire text into **uppercase or lowercase**, so that the algorithm does not treat the same words in different cases as different
- **Tokenization:** Tokenization is just the term used to describe the process of converting the normal **text strings into a list of tokens**.
- Removing **Stop words**, words that **does not add much meaning** to a sentence.
- **Stemming/Lemmatization:** Major difference being, stemming can often create non-existent **root words**, whereas lemmas are actual words.

# Some Basics

**Vectorization:** After the initial preprocessing phase, we need to transform the text into a meaningful vector (or array) of numbers.

- The **Bag-of-words** is a representation of text that describes the occurrence of words within a document. No information about the order or structure of words in the document.
  1. Dictionary contains {Learning, is, the, not, great}
  2. Vectorizing text “Learning is great”, vector: (1, 1, 0, 0, 1).

# Some Basics

- **Term Frequency-Inverse Document Frequency**, or **TF-IDF** :rescales the frequency of words by how often they appear in all documents so that the scores for frequent words like “the” that are also frequent across all documents are penalized.
  1. **Term Frequency**: is a scoring of the frequency of the word in the current document.
  2. **Inverse Document Frequency**: is a scoring of how rare the word is across documents.

# Some Basics

- Consider a document containing 100 words wherein the word ‘phone’ appears 5 times.
- The term frequency (i.e.,  $tf$ ) for phone is then  $(5 / 100) = 0.05$ .
- Assume we have 10 million documents and the word phone appears in 1000 of these.
- The inverse document frequency (i.e.,  $IDF$ ) is calculated as  $\log(10,000,000 / 1,000) = 4$ .
- The  $Tf-IDF$  weight is the product of these quantities:  $0.05 * 4 = 0.20$ .

# NLTK

---

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

Open Anaconda terminal, run **pip install nltk** and other libraries as required

# General Instructions

---

- Check your connectivity and system settings before pinging.

# Pre-processing text data

---

Cleaning (or pre-processing) the data typically consists of a number of steps:

- Remove punctuation
- Tokenization
- Remove stopwords
- Lemmatize/Stem

# NLP

- Loading the data
- Understand the data
- Data Pre-processing
  - Remove Punctuations
  - Remove Stopwords
  - Stemming/Lemmatization
- Vectorizing Raw Data
  - CountVectorizer
  - N-Grams
  - TF-IDF
- Model Building

# Word Embeddings

Run the script to understand!



# Demo

---

# Queries??



**BITS** Pilani

Pilani|Dubai|Goa|Hyderabad

# Webinar 2

## Neural Language Model

Presented by: Radhika B

# What is a Language Model in NLP?

---



- A language model learns to predict the probability of a sequence of words.
- compute the probability of a sentence or sequence of words:

$$P(W) = P(w_1, w_2, w_3, w_4, \dots \dots \dots, w_n)$$

- A model that computes either of these:

$P(W)$  or  $P(w_n | (w_1, w_2, w_3, w_4, \dots \dots \dots, w_{n-1}))$  is called a **language model** or **LM**

# Types of Language Models

---

There are primarily two types of Language Models:

- 1. Statistical Language Models:** These models use traditional statistical techniques like N-grams, Hidden Markov Models (HMM) and certain linguistic rules to learn the probability distribution of words
- 2. Neural Language Models:** These are new players in the NLP town and use different kinds of Neural Networks to model language

# Building an N-gram Language Model

achieve

lead

An N-gram is a sequence of N tokens (or words)

- Let's understand N-gram with an example. Consider the following sentence: **“I love reading blogs about data science.”**
- A 1-gram (or unigram) is a one-word sequence. For the above sentence, the unigrams would simply be: “I”, “love”, “reading”, “blogs”, “about”, “data”, “science”.
- A 2-gram (or bigram) is a two-word sequence of words, like “I love”, “love reading”, or “data science”.

# How do N-gram Language Models work?

Can you please come **here** ?



- An N-gram language model predicts the probability of a given N-gram within any sequence of words in the language.
- If we have a good N-gram model, we can predict  $p(w|h)$  — what is the probability of seeing the word **w** given a history of previous words **h** — where the history contains  $n - 1$  words.

# Building a Basic Language Model

---

- Let's build a basic language model using trigrams of the Reuters corpus.
- Reuters corpus is a collection of 10,788 news documents totaling 1.3 million words. We can build a language model in a few lines of code using the NLTK package:

# Limitations of N-gram approach to Language Modeling

---

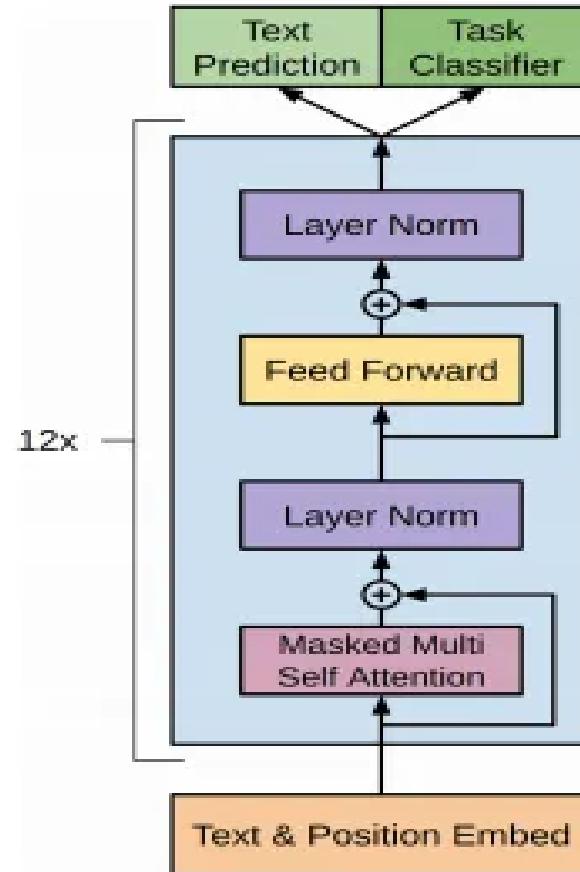
N-gram based language models do have a few drawbacks:

1. The higher the N, the better is the model usually. But this leads to lots of computation overhead that requires large computation power in terms of RAM
2. N-grams are a sparse representation of language. It will give zero probability to all the words that are not present in the training corpus

# Neural Language Model

- Deep Learning has been shown to perform really well on many NLP tasks like
  1. Text Summarization
  2. Machine Translation, etc.
- Since these tasks are essentially built upon Language Modeling, there has been a tremendous research effort with great results to use Neural Networks for Language Modeling.
- The problem statement is to train a language model on the given text and then generate text given an input text in such a way that it looks straight out of this document and is grammatically correct and legible to read.

# Generative Pretrained Transformer 2



Architecture

# Generative Pretrained Transformer 2

---

1. **Input Embeddings:** The initial text inputs (words, or more commonly subwords) are converted into a sequence of vectors using an embedding layer. Each input word corresponds to a unique vector in this embedding layer.
2. **Positional Encoding:** To retain the order information of the input sequence, positional encodings are added to these input embeddings.
3. **Self-Attention Mechanism:** The core component of the transformer model is the self-attention mechanism, which is applied to the input sequence. **Self-attention allows the model to weigh the relevance of a word in the sequence to every other word, including itself.**
4. This is how the model is able to capture dependencies between words regardless of their distance from each other in the text.

# Generative Pretrained Transformer 2

- 
- 5. **Feed-forward Neural Network:** The outputs from the self-attention layers are then passed through a feed-forward neural network, which is the same for each position.
  - 6. **Normalization and Residual Connections:** Layer normalization and residual connections are used to stabilize the learning process and to enable the training of deep networks.
  - 7. **Decoder:** The decoder is designed similarly to the encoder but also includes additional layers to aid in generating output. However, in GPT-2, which is a decoder-only model, we only have this part.
  - 8. **Linear and Softmax Layer:** Finally, the outputs from the transformer layers are passed through a linear layer followed by a softmax activation to generate a probability distribution over the vocabulary for the next word prediction.

This process is repeated for every layer in the model, with the output of one layer serving as the input for the next. In GPT-2, this can be up to 48 transformer layers (for the largest variant of the model).

# DEMO