



Natural Language Processing

DSECL ZG565



BITS Pilani
Pilani Campus

Dr. Vijayalakshmi Anand

BITS-Pilani



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

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

Agenda

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

Objective of course

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

M1 Natural Language Understanding and Generation

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

Text books and Reference books

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

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

Tentative Evaluation Plan

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

Students are requested to check the canvas announcements for the details

What is Natural Language Processing?

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

What is it..

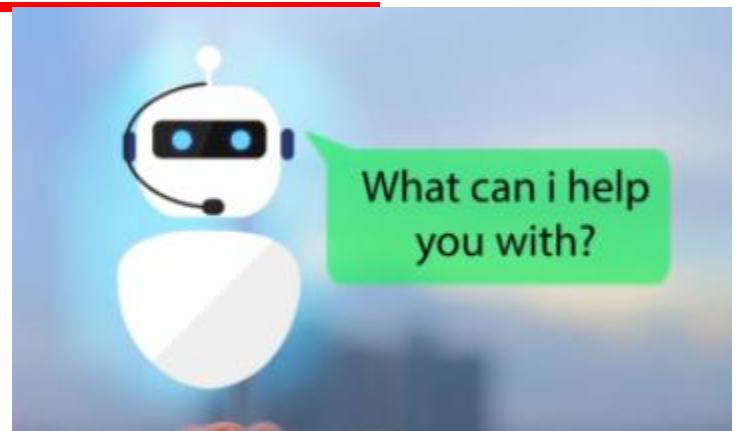
- Analyze, understand and generate human languages just like humans do.
- Applying computational techniques to language domain..
- To explain linguistic theories, to use the theories to build systems that can be of social use..
- Started off as a branch of Artificial Intelligence..
- Borrows from Linguistics,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

Natural Language processing



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

Application of NLP



English ▾

Hindi ▾

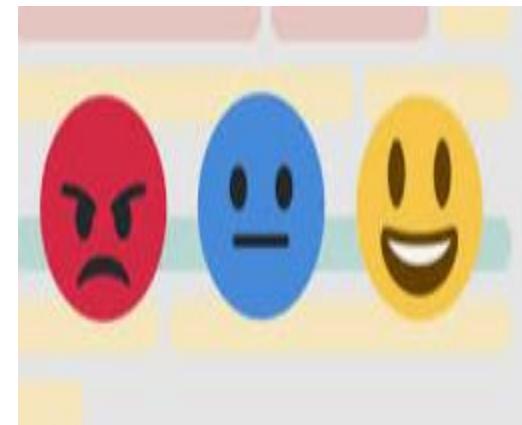
i am fine

×

मैं ठीक हूँ
main theek hoon

1

Community verified



History of NLP

1950-Alan turing published paper

1957-Chomsky published his book, Syntactic Structures

1958-LISP

1964-ELIZA

1966-AI,NLP ,Machine transalation

1980-IBM

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

2020 – Conversational Agents

2022 -- ChatGPT

Example

Dave: Open the pod bay doors, HAL.

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

Dave: What's the problem.

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

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

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

General speech and language understanding and generation capabilities

Politeness: emotional intelligence

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

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

Contd..



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

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

HAL: Can you hold it a bit closer?

[Dave does so].

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

Dave: Yes.

Recognition of emotion from speech

Vision capability including visual recognition of emotions and faces

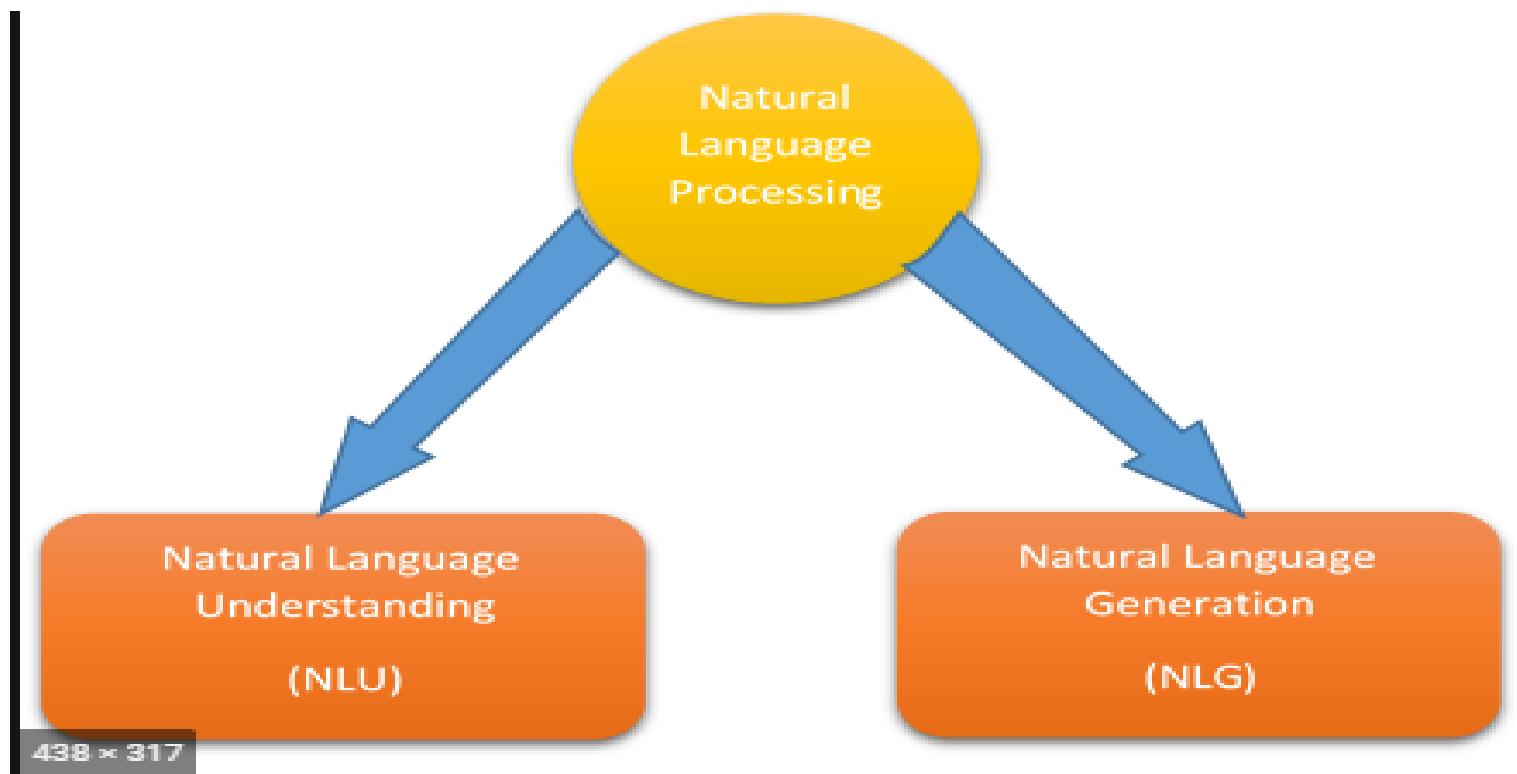
Also: situational ambiguity

Contd..

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

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

Main components of NLP



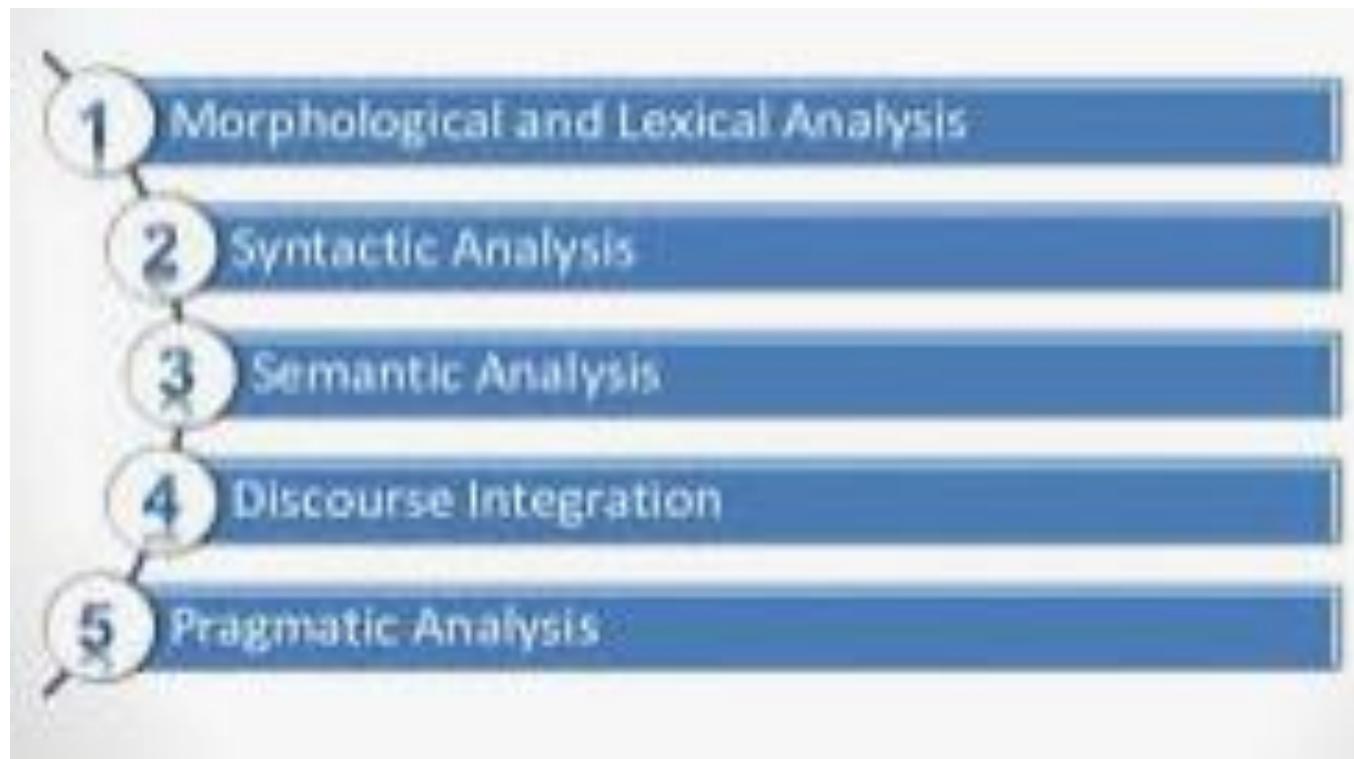
Natural language understanding

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

Natural language generation

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

Steps used for NLU



Morphology analysis

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

Lexical analysis

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

Syntactic analysis

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

➤ Eg:

“the girl go to the school “

Agra goes to the Poonam

Semantic analysis

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

Discourse integration

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

Pragmatic analysis

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

Introduction to Natural Language Processing



Different Levels of Language Analysis

Examples:

Syntax, Semantics, and Pragmatics

1. Green frogs have large noses.
Pragmatically wrong

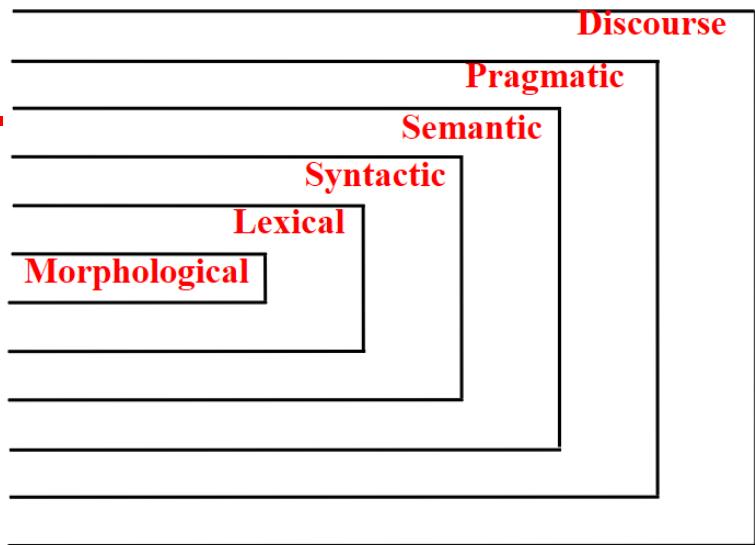
2. Green ideas have large noses.
Semantics, and Pragmatics

3. Large have green ideas nose.
All three wrong

4. I are fond in animals (exercise)

5. My cat is studying Linguistics (exercise)

6. Spain is larger than China (exercise)



Natural Language Generation

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

Why NLP is hard

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

- **Ambiguities at all level**

- I.** Structural Ambiguities

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

- II.** Lexical Ambiguities:

- I saw bats

Contd..

Non Standard english

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

Segmentation issues

- Eg The old city-bus stop

Idioms

- Eg:
 - dark horse
 - lose face

World knowledge

Eg:

Mary and Sue are sisters.

Mary and Sue are mothers.

Tricky entity names

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

Some of the tasks in NLP

Sentence segmentation

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

Contd..

➤ Tokenization

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

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

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

➤ Lemmatization

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

Contd..

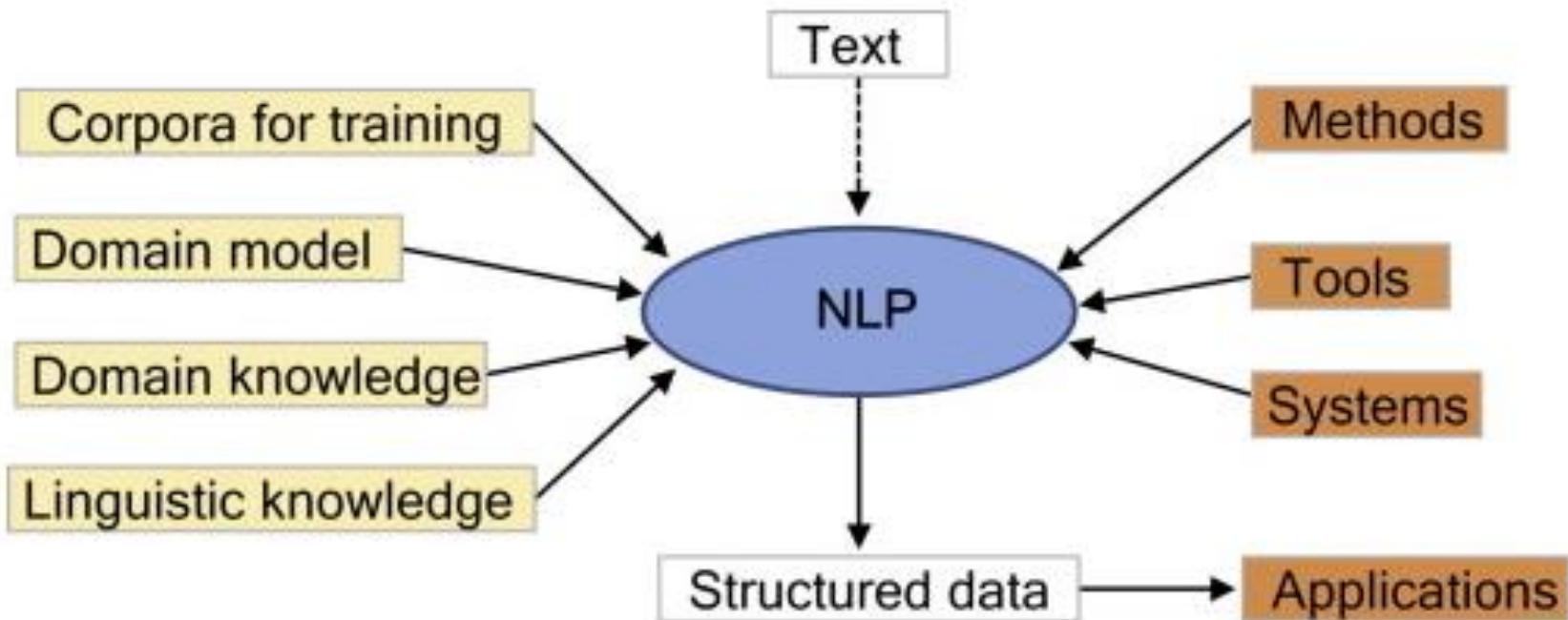
➤ **Stemming**

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

➤ **Stop Words**

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

Schematic representation NLP



Representations and Understanding

Syntactic representations of language

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

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

Representations and Understanding

Allen 1995: Natural Language Understanding - Introduction

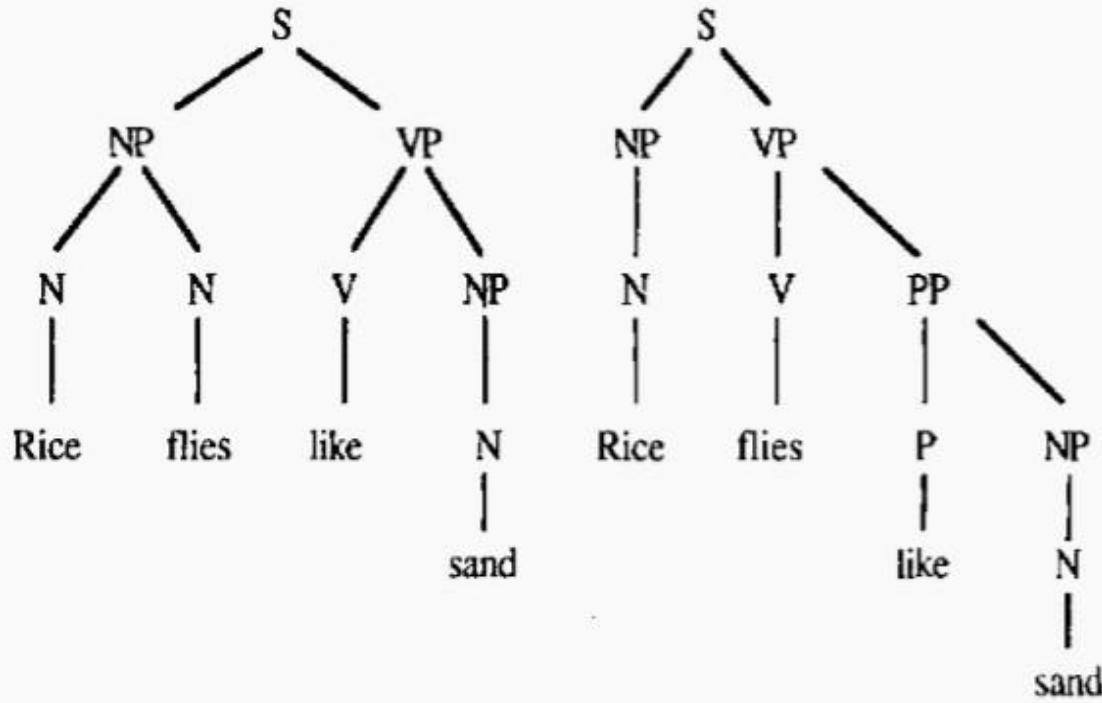


Figure 1.4 Two structural representations of *Rice flies like sand*.

NP – Noun Phrases

VP – Verb Phrases

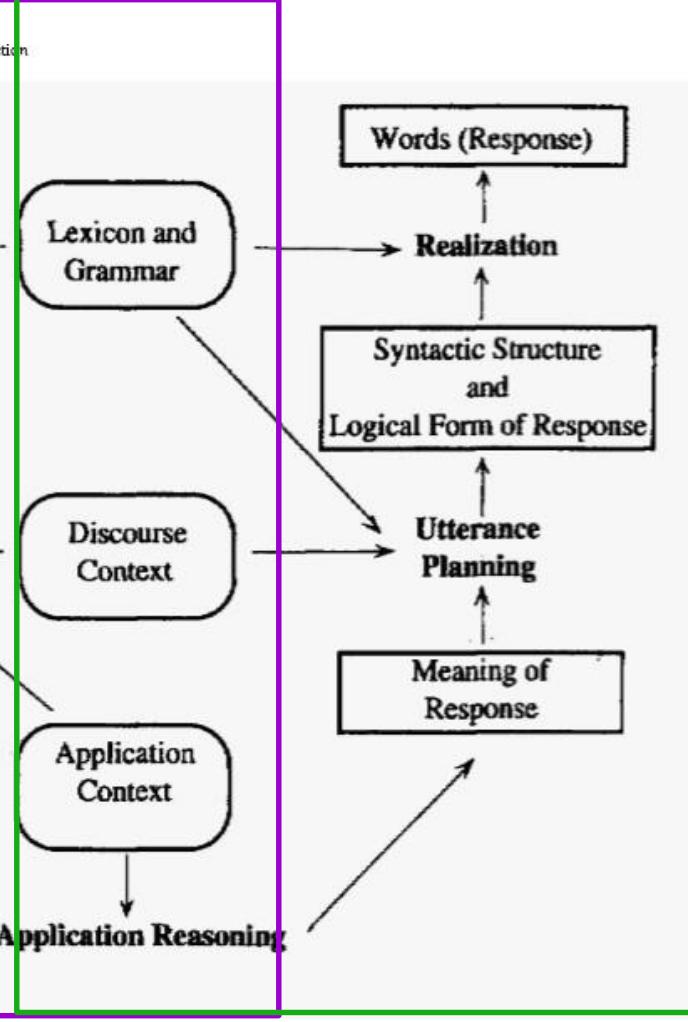
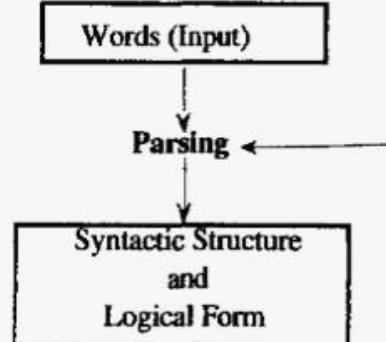
PP – Prepositional Phrases

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

The Organization of Natural Language Processing Systems

**Natural
Language
Understanding**

Allen 1995: Natural Language Understanding - Introduction



**Natural
Language
Generation**

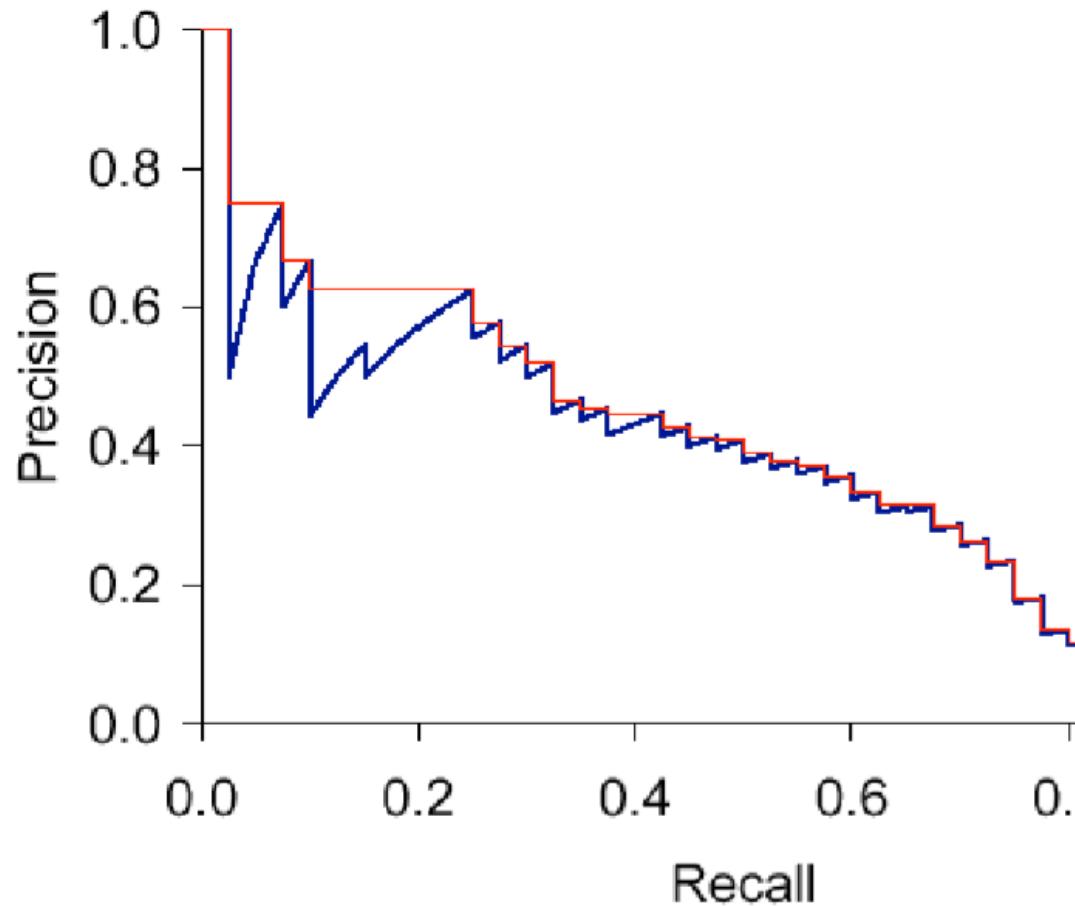
Evaluating Language Understanding Systems

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

Contingency Table

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

Tradeoff between Precision and Recall



NLTK Installation

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

Open Anaconda terminal, run

pip install nltk.

Anaconda and Jupiter are best and popular data science tools

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

! pip install nltk

[NLTK book](#)

[NLTK discussion forum](#)

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

NLP Tools

Some commercial tools

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

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

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

NLP Tools

Open Source Tools

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

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

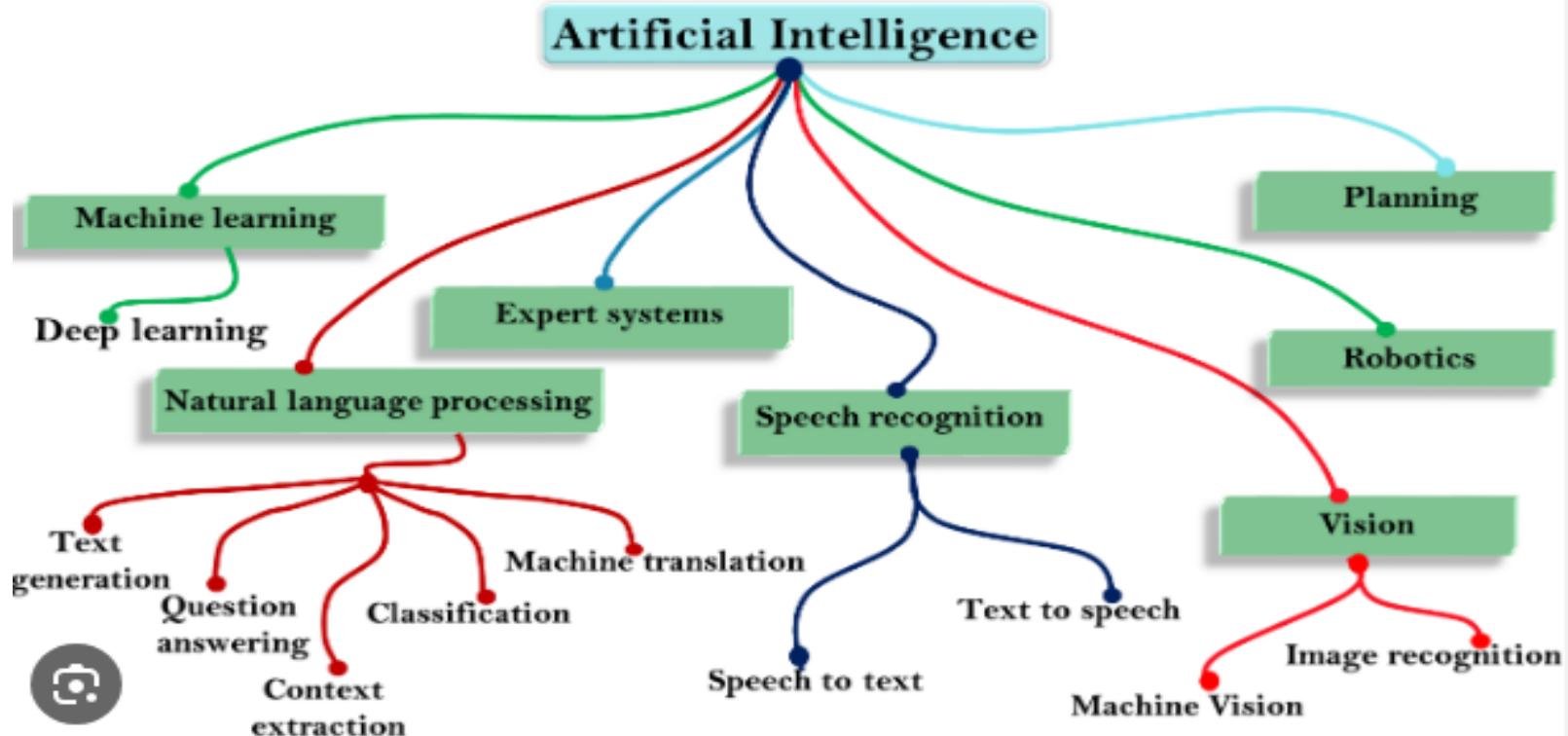
[Gensim](#) is a highly specialized Python library that largely deals with topic modeling tasks using algorithms like Latent Dirichlet Allocation (LDA)

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

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

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

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



References

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

Thank You... 😊

- Q&A
- Suggestions / Feedback



Natural Language Processing

DSECL ZG565



BITS Pilani
Pilani Campus

Dr.Vijayalakshmi Anand

BITS-Pilani



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

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

Content Sequence Plan

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

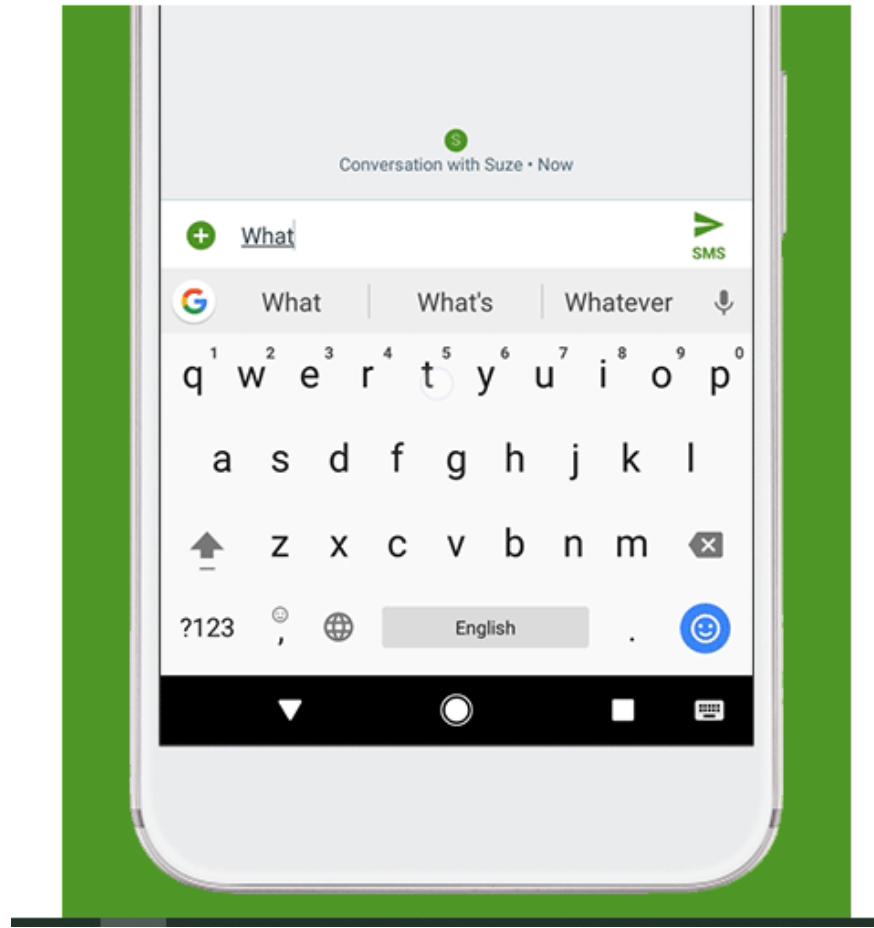
Session#2 – N gram Language model

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

Human word Prediction

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

Example



Language modelling



A model that computes either of these:

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

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

Simply we can say that

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

Language Models



1. Machine Translation:

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



Continued..

2.Spell correction

Example:

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

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

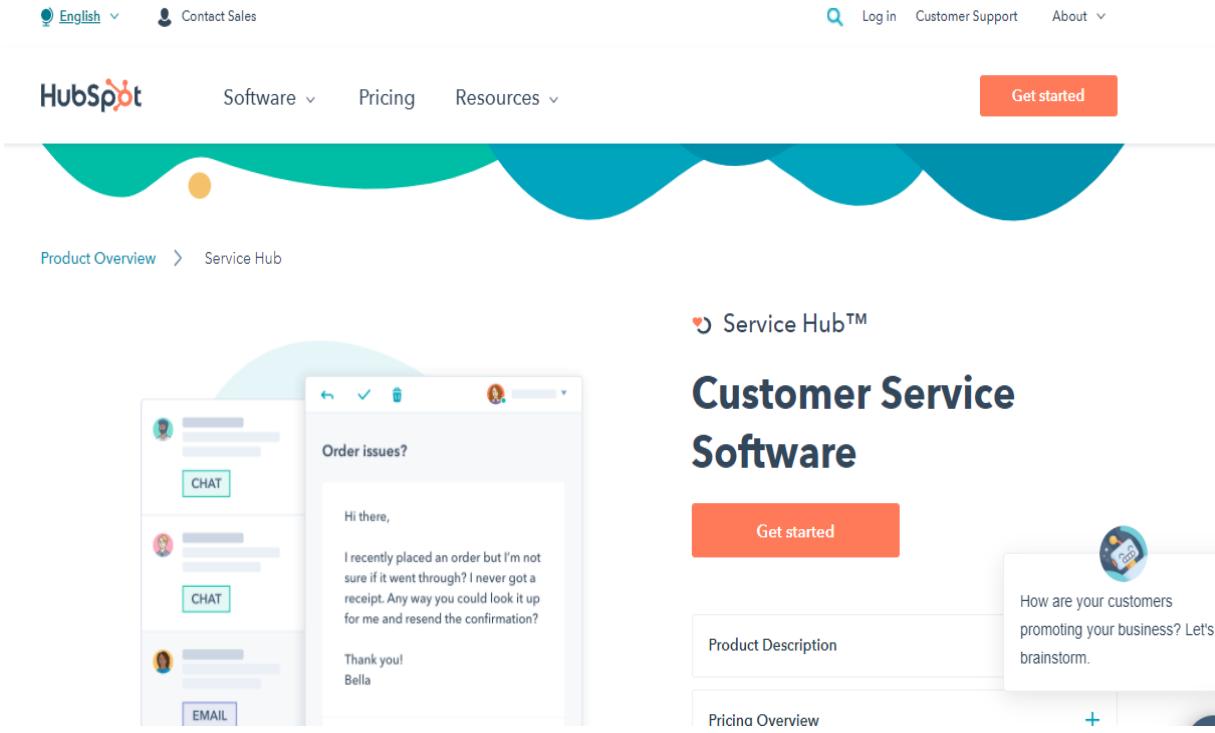
3.Speech Recognition

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



Sentiment analysis

Example :Hubspot's Service



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

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

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

Order issues?

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

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

How to build a language model



- Recall the definition of conditional probabilities

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

- More variables:

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

- The **Chain Rule** in General

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

Example

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

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

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

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

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

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

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

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

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

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

Markov Assumption

- Simplifying assumption:

limit history of fixed number of words N1



Andrei Markov

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

or

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

Markov Assumption

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

In other words, we approximate each component in the product

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

N -Gram Language models

N-gram Language models

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

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

Example:

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

Bigram model

N=2

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

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

Example:

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

N-gram models

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

Advantages

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

Disadvantage:

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

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

Estimating N-gram Probabilities

Estimating bigram probabilities

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

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

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

An example

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

<s> I am Sam </s>

<s> Sam I am </s>

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

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

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

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

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

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

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

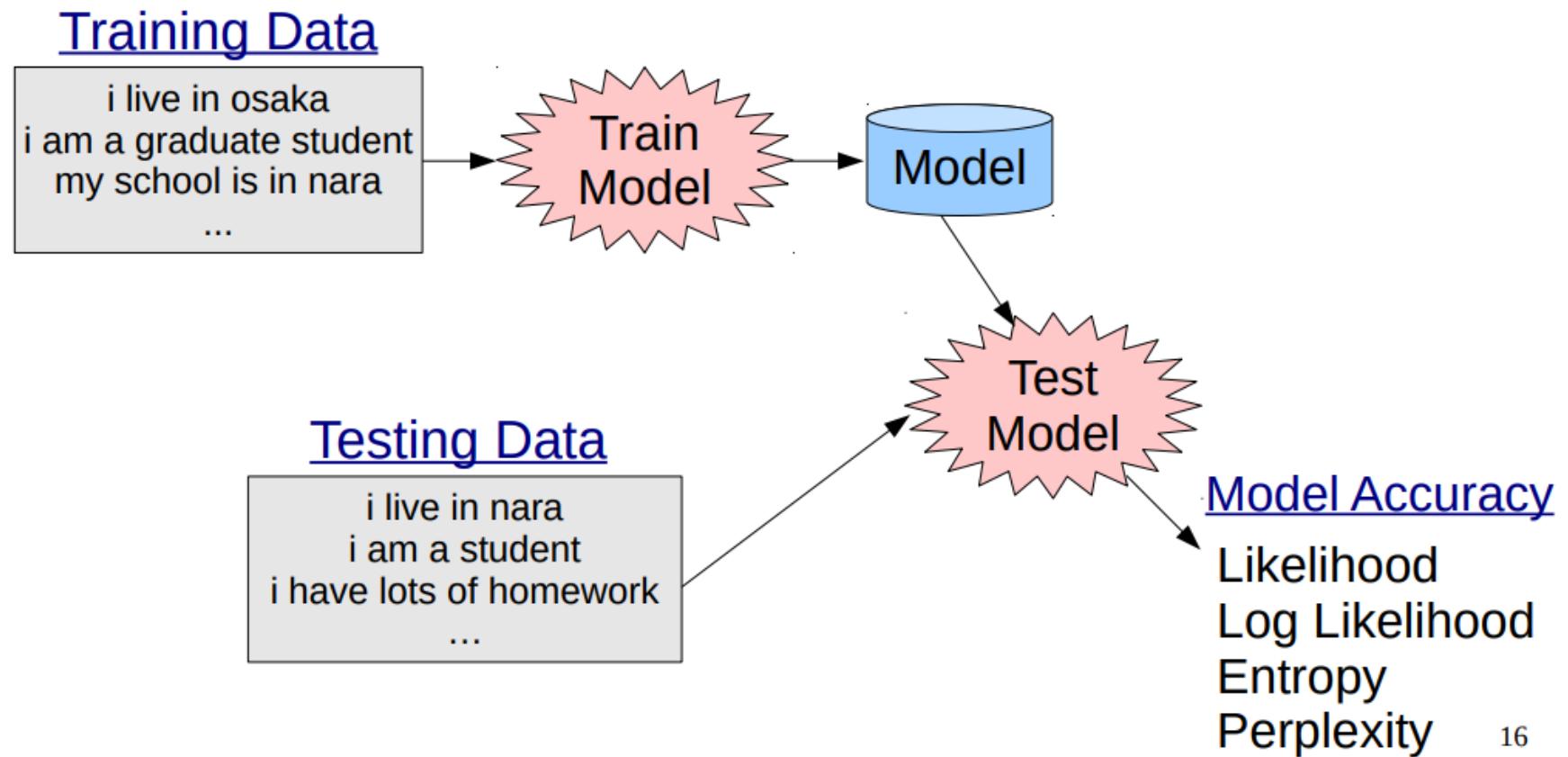
Evaluation



How good is our model?

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

Experimental setup



Example1:

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

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

Extrinsic evaluation of N-gram models



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

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

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

Intuition of Perplexity



- The Shannon Game:

- How well can we predict the next word?

I always order pizza with cheese and....

The president of India is.....

I wrote a

{

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

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

Perplexity



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

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

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

Chain rule:

For bigrams:

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

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

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

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

Minimizing perplexity is the same as maximizing probability

Example1

Using bigram model

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

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

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

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

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

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

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

$$= 2$$

Example2

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

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

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

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

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

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

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

Corpora

Examples of corpora (in chronological order)

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

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

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

ECI: European Corpus Initiative, multilingual.

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

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

Gigaword: 1B words of news text.

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

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

Flickr 8K: images with NL captions

English Visual Genome: Images, bounding boxes ⇒ NL descriptions

Lower perplexity = better model

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

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Generalization and Zeros

The perils of overfitting

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

Problems with simple MLE estimations

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

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

Zero probability bigrams

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

Laplace Smoothing (Add 1 smoothing)

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

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

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

Example

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

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

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

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

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

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

After Add-1 smoothing

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

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

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

$$\approx 0.0001$$

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

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

$$\approx 0.00003$$

Berkeley Restaurant Project corpus

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

Raw Bigram Counts

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

Converting to probabilities

- Obtain likelihoods by dividing bigram counts by unigram counts.

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

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

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

Contd...

- Obtain likelihoods by dividing bigram counts by unigram counts.

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

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

Berkeley Restaurant Corpus: Laplace smoothed bigram



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

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

Laplace-smoothed bigrams

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

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

$V=1446$ in the Berkeley restaurant project corpus

Reconstituted counts



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

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

Compare with raw bigram counts

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

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

Add-1 estimation is a blunt instrument

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

Backoff and Interpolation

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

Linear Interpolation

- Involves using different pieces of information to derive a probability

Simple interpolation

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

$$\sum_i \lambda_i = 1$$

How to set the lambdas?

- Use a **held-out** corpus

Training Data

Held-Out
Data

Test
Data

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

Unknown words: Open versus closed vocabulary tasks

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

Huge web-scale n-grams

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

Smoothing for Web-scale N-grams

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

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

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

Exercise 1

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

Given Corpus

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

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

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

Exercise2

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

Given Corpus

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

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

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

Thank You... 😊

- Q&A
- Suggestions / Feedback



Natural Language Processing

DSECL ZG565



BITS Pilani
Pilani Campus

Dr.Vijayalakshmi Anand

BITS-Pilani



Session 2

Date – 9DEC 2023

Time – 1.40pm to 3.40pm

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

Content Sequence Plan

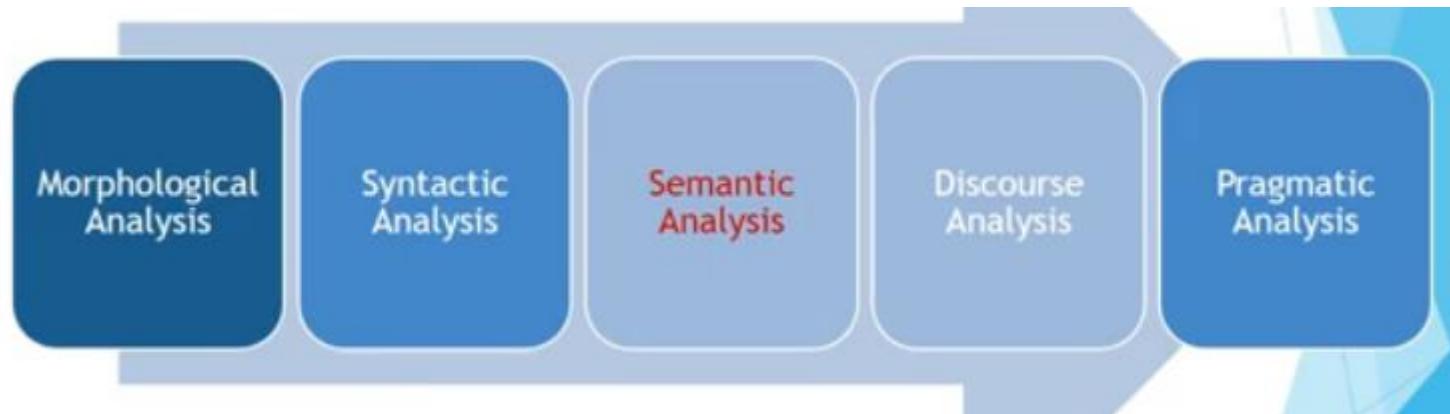
-
- M1 Natural Language Understanding and Generation
 - M2 N-gram Language Modelling
 - M7 Vector semantics and Embedding
 - M3 Neural networks and Neural language Models
 - M4 Part-of-Speech Tagging
 - M5 Hidden Markov Models and MEMM
 - M6 Topic Modelling
 - M8 Grammars and Parsing
 - M9 Statistical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

Session contents

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

Lexical Semantics

Stages in Natural language processing



Semantic analysis



What is semantic analysis ?

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

Why semantic analysis?

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

Applications

Question answering:

Q: “How **tall** is Mt. Everest?”

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

“tall” is similar to “height”

Plagiarism detection

MAINFRAMES

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

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

MAINFRAMES

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

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

What do words mean?

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

Lexical semantics

Lemmas and senses



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

mouse (N) : **lemma**

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

sense

Lexical semantics Relations

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

Synonyms

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

- filbert / hazelnut
- couch / sofa
- big / large
- automobile / car
- vomit / throw up
- Water / H₂O

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 H₂O

Synonymy is a relation between senses rather than words

Consider the words *big* and *large*

Are they synonyms?

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

How about here:

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

Why?

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

Antonyms

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

Otherwise, they are very similar!

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

More formally: antonyms can

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

Relation: Similarity

- Words with similar meanings.

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

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

Relation: Word relatedness

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

Semantic field

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

Example

hospitals

surgeon, scalpel, nurse, anaesthetic, hospital

restaurants

waiter, menu, plate, food, menu, chef

houses

door, roof, kitchen, family, bed

Semantic frame

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

Connotation (sentiment)

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

Connotation

Words seem to vary along 3 affective dimensions:

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

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

Values from NRC VAD Lexicon (Mohammad 2018)

Vector Semantics

What does recent English borrowing *ongchoi* mean?

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

Vector Semantics

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

Word embeddings

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



Word embeddings: Why



Consider sentiment analysis:

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

Word embeddings: types

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

Frequency based Embedding

Count vector

What is count vectorization?

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

Example

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

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

Co-occurrence matrix

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

Term Document Matrix

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

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

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

Two documents are similar if their vectors are similar

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

The words in a term-document matrix



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

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

The words in a term-document matrix



Two words are similar if their vectors are similar

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

Term-context matrix for word similarity

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

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

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

Term frequency tf

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

Frequency in document vs. frequency in collection



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

Desired weight for rare terms

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

Desired weight for frequent terms

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

Document frequency

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

idf

weight

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

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

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

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

Examples for idf

- Compute idf_t using the formula:

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

term	df_t	idf_t
calpurnia	1	6
animal	100	4
sunday	1000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

Effect of idf on ranking

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

Collection frequency vs. Document frequency

word	collection frequency	document frequency
INSURANCE	10440	3997
TRY	10422	8760

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

tf-idf

weighting

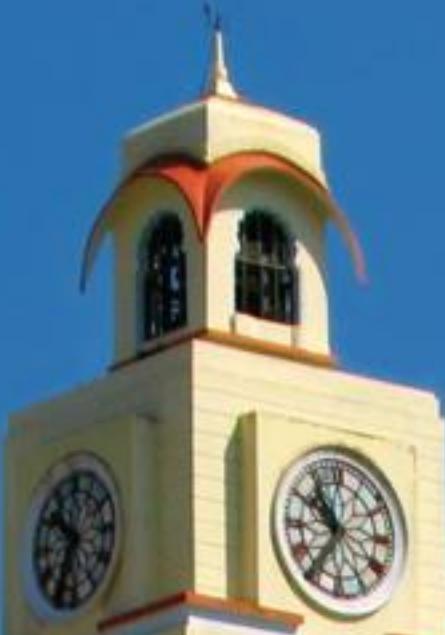
- The tf-idf weight of a term is the product of its **tf weight** and its **idf weight**.

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

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

Thank You... 😊

- Q&A
- Suggestions / Feedback



Natural Language Processing

DSECL ZG565



BITS Pilani
Pilani Campus

Dr.Vijayalakshmi Anand

BITS-Pilani



Session 2

Date – 16DEC 2023

Time – 1.40pm to 3.40pm

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

Content Sequence Plan

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



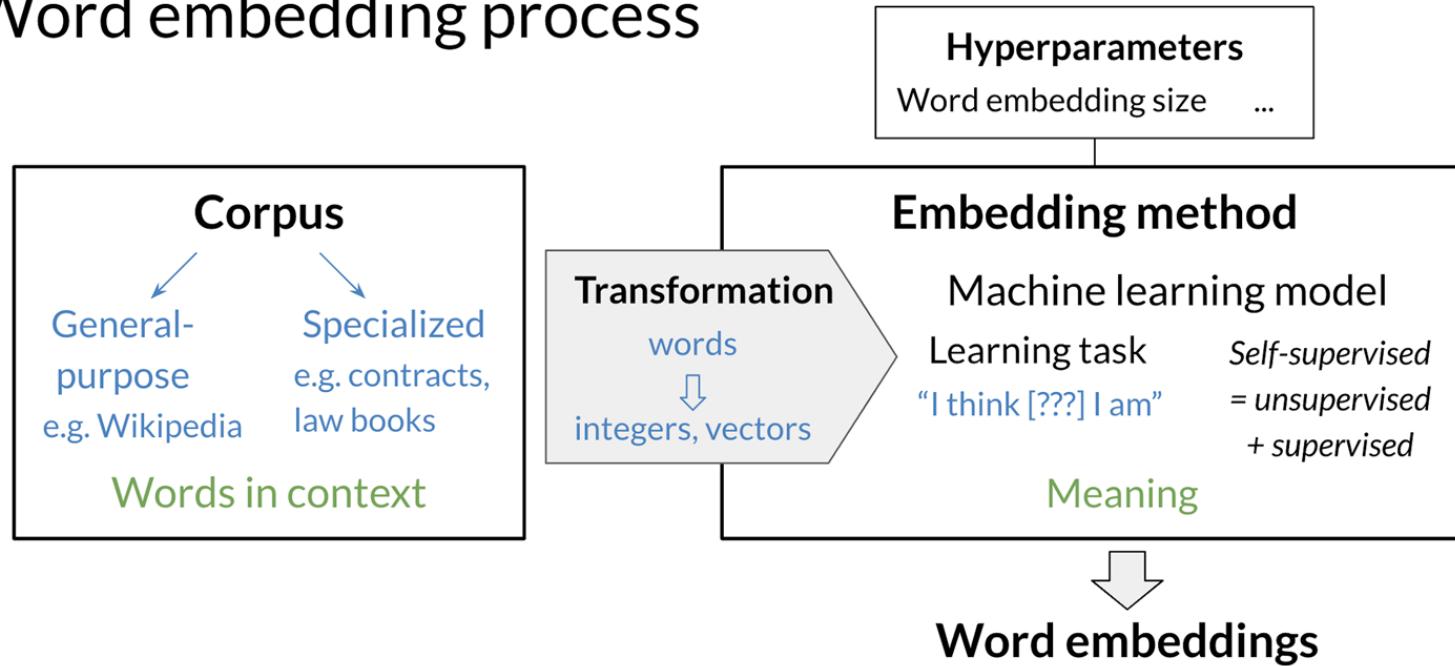
Prediction based Embedding

Sparse versus dense vectors

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

Word embeddings

Word embedding process



Common methods for getting short dense vectors

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

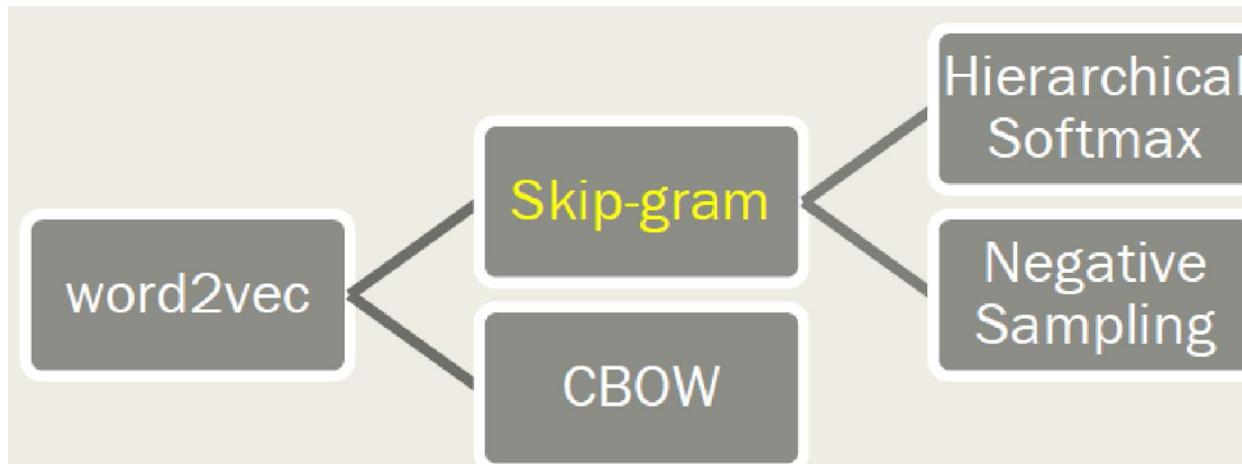
Word2vec -What is word2vec

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

Word2vec

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

Word2vec -Types of word2vec



Basic word representation

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

1-hot representation

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

I want a glass of orange _____

I want a glass of apple _____.

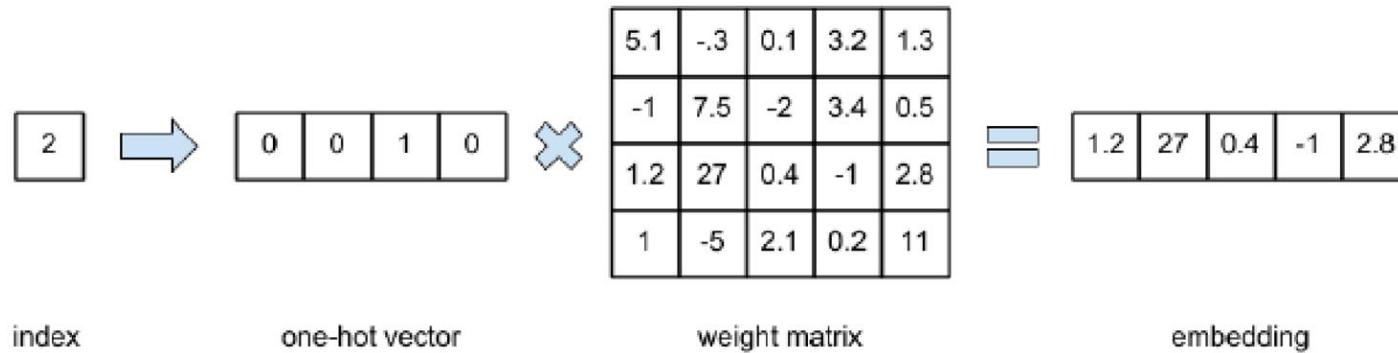
Featurized representation: word embedding

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

I want a glass of orange

I want a glass of apple

Word embeddings



CBOW [Continuous bag of words]

What is CBOW?

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

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

Working of CBOW with example

Corpus

The product is really good

The product is wonderful

The product is awful

Step1:

combine the sentences

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

Step2:

Select window size

Contd..

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



Window Size

1

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



Window Size

2

Contd..

Step3: Get one hot encoding for each word

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

Step4:Training data

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

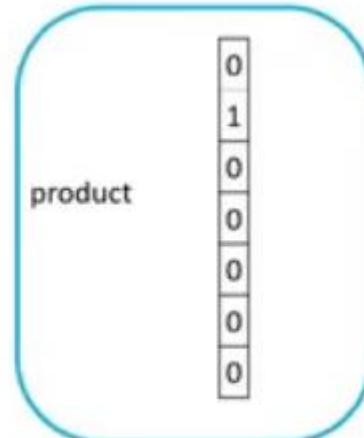
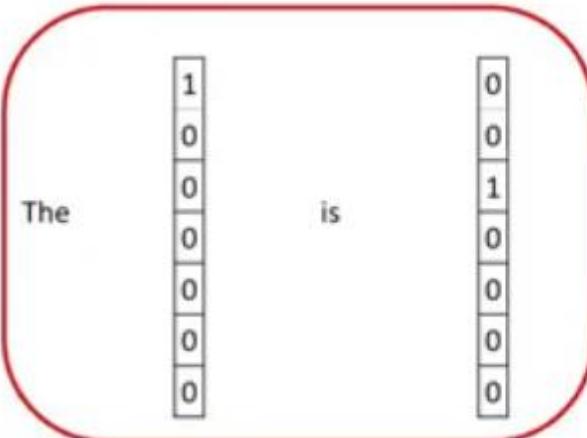
Contd..

One Hot Encoding

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

Training Data

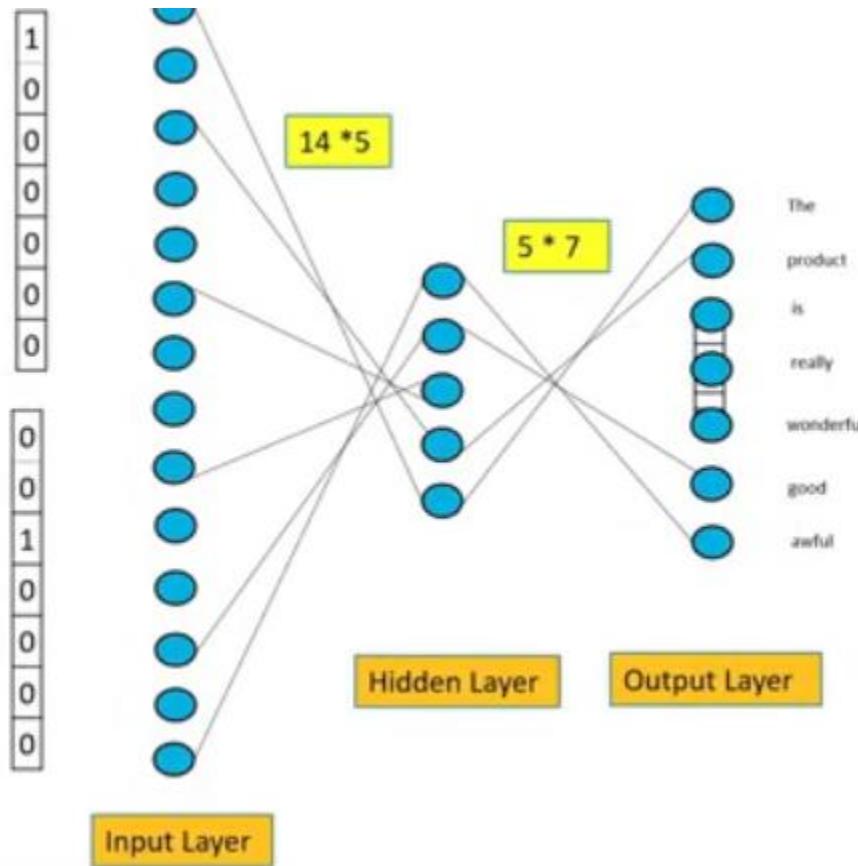
Context Words	Target Word
The,is	product
product,really	is
Is,good	really
Good,product	the



Contd..

One hot encoding of - The

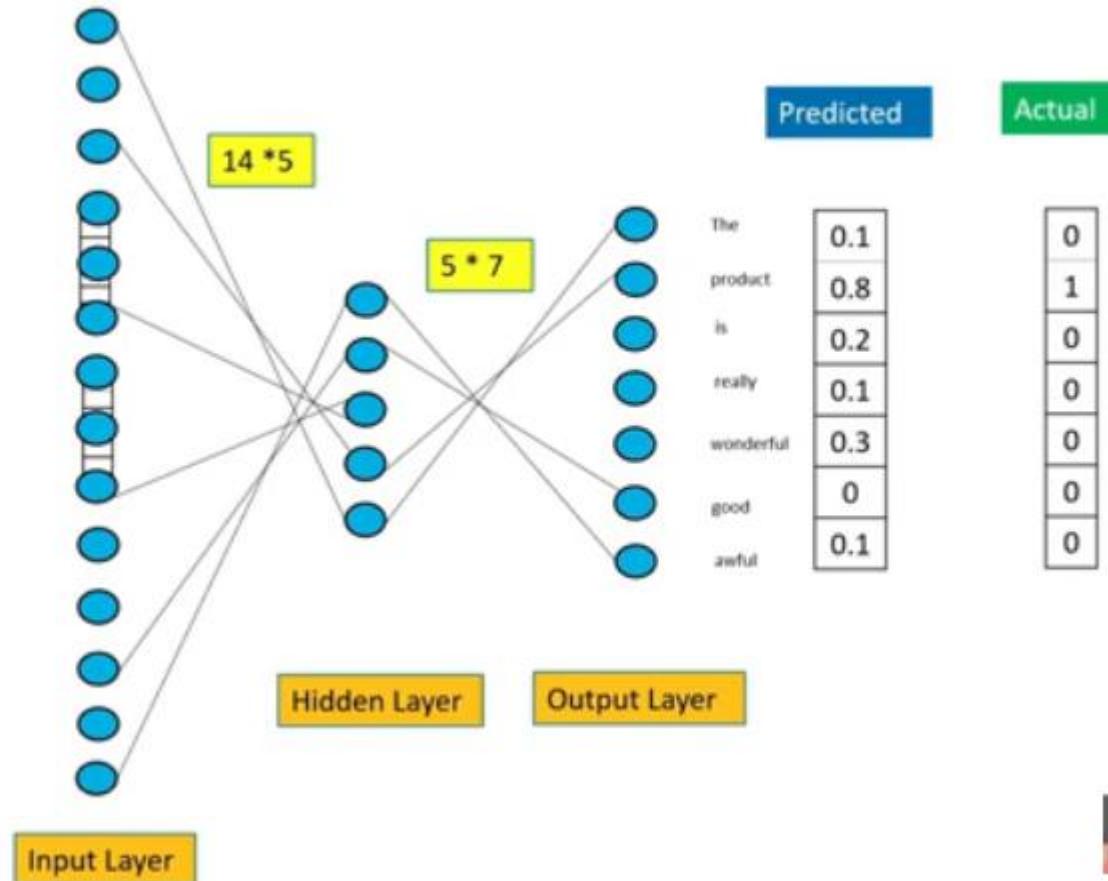
One hot encoding of - is



Cond..

One hot encoding of - The

One hot encoding of - is



Word Embedding matrix

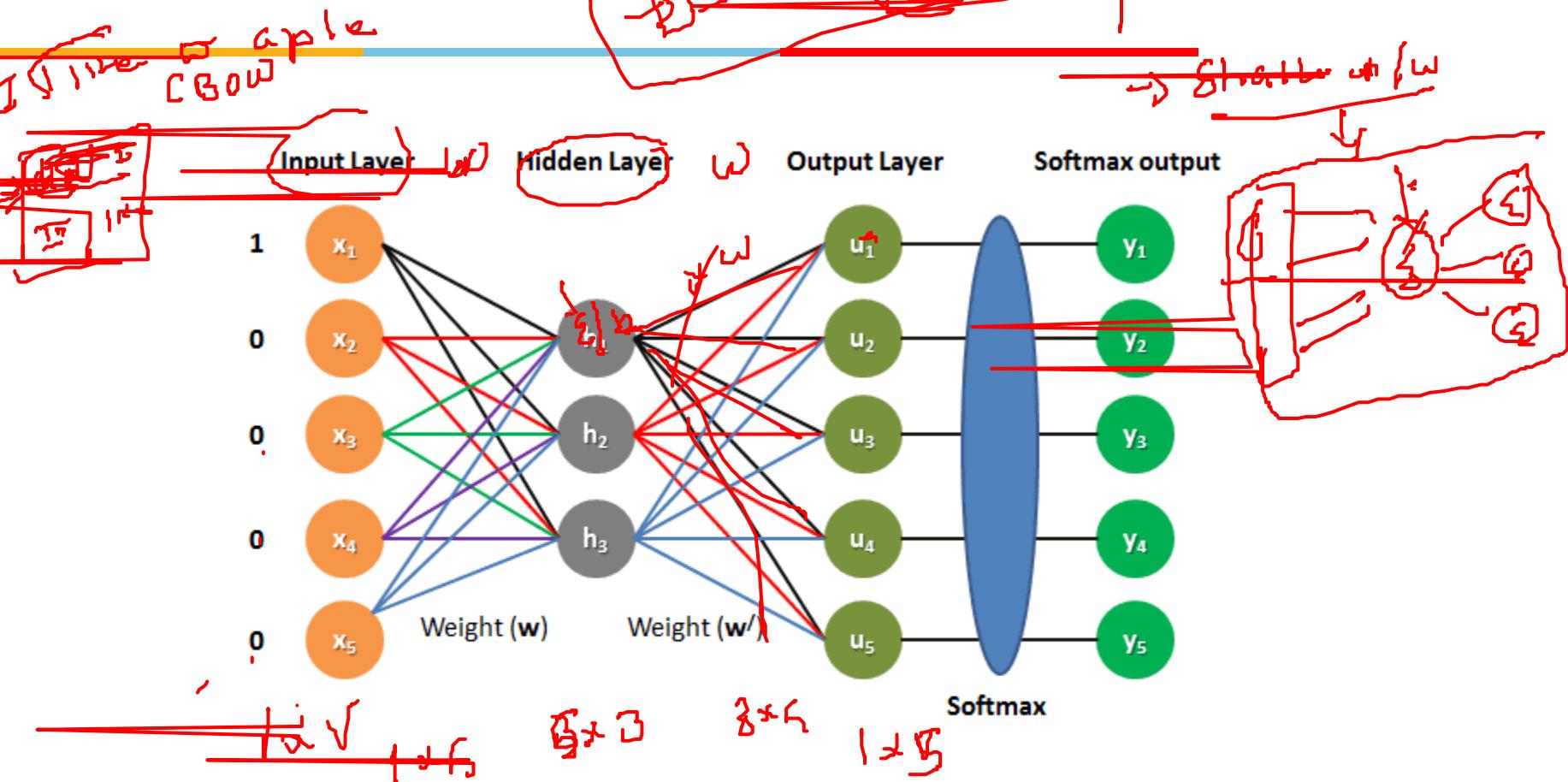
5 * 7

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

Training model

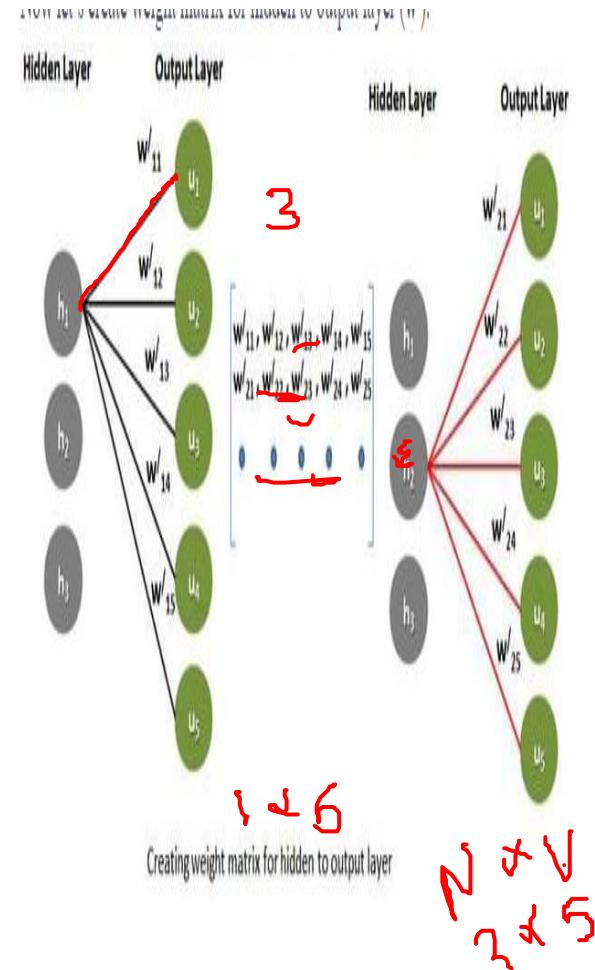
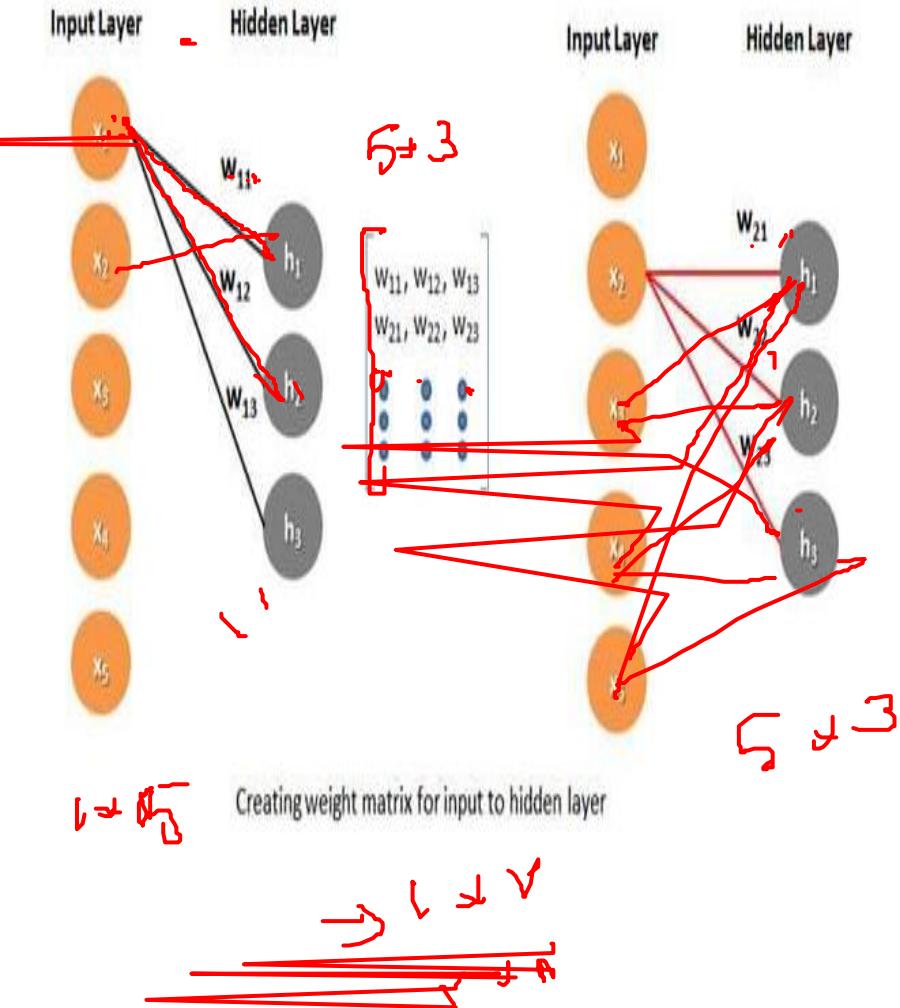
1. Create model Architecture
 2. Forward Propagation
 3. Error Calculation
 4. Weight tuning using backward pass
-

Model architecture



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

Weighed matrix



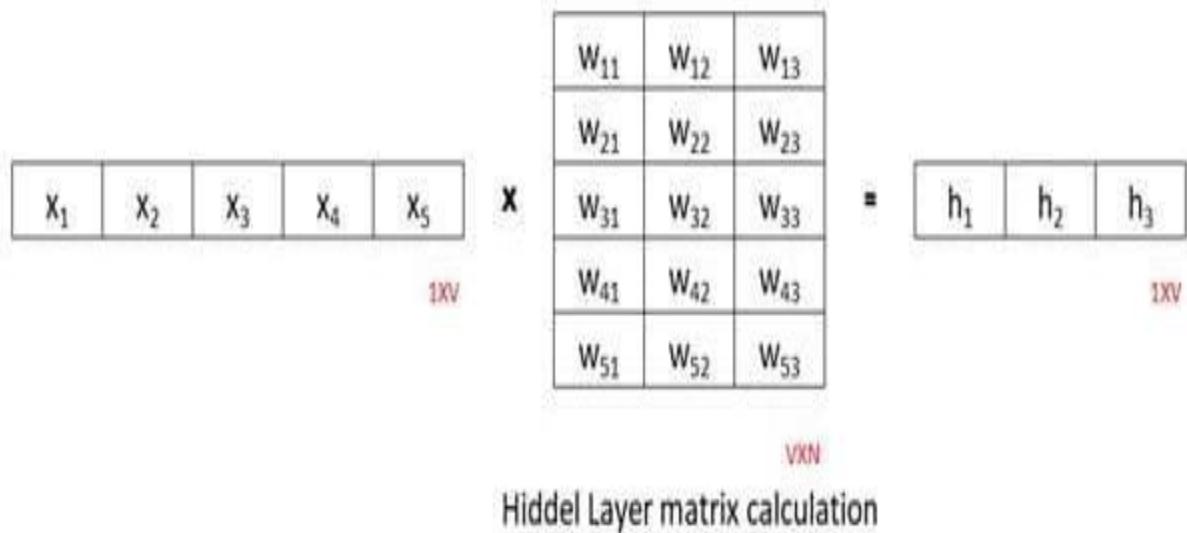
CBOW Vectorized form:

$$\begin{array}{c|c}
 \begin{matrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{matrix} & \times \\
 \begin{matrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \\ w_{51} & w_{52} & w_{53} \end{matrix} & \times \\
 \begin{matrix} h_1 \\ h_2 \\ h_3 \end{matrix} & \times \\
 \begin{matrix} w'_{11} & w'_{12} & w'_{13} & w'_{14} & w'_{15} \\ w'_{21} & w'_{22} & w'_{23} & w'_{24} & w'_{25} \\ w'_{31} & w'_{32} & w'_{33} & w'_{34} & w'_{35} \end{matrix} & \times \\
 \begin{matrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{matrix} & \times \\
 \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{matrix} &
 \end{array}$$

1XV VVN 1XN NXV 1XV 1XV

CBOW vectorized form

Forward Propagation



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

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

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

⋮ ⋮ ⋮

Contd..

Calculate output layer matrix (u):

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

Output Layer matrix calculation

So now:

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

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

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

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

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

contd

Calculate final Softmax output (y):

u_1	y_1
u_2	y_2
u_3	y_3
u_4	y_4
u_5	y_5

Softmax

1XV

1XV

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

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

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

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

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

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

...

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

Error calculation – log loss function

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

w_t = Target word

w_c = Context word

out out

I

~~1, 2, 3~~

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

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

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

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

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

Back propagation

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

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

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

Now as we know

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

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

Derivative of log function with chain rule.

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

Contd..

Taking derivative of E with respect to u_j :

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

$$= (y_j - t_j)$$

$$= \textcircled{e_j}$$

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

Contd..

So for first iteration,

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

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

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

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

So generalized form will looks like below:

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

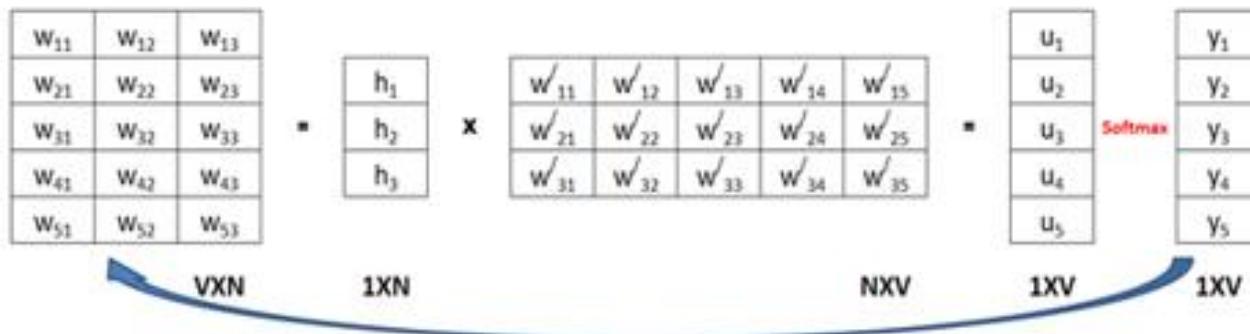
Step2: Updating the weight of w'_{11} :

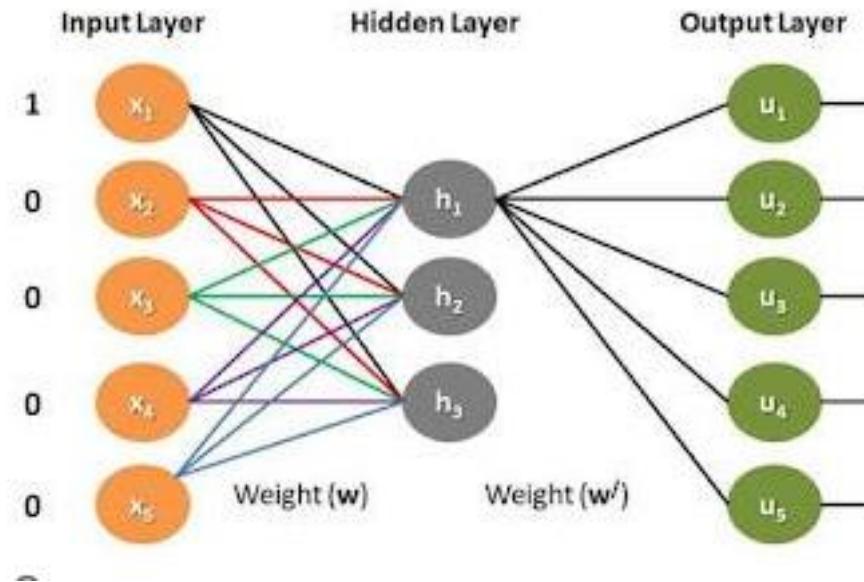
Step2: Updating the weight of w'_{11} :

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

Now Updating the first weight

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





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

As for u_1 and h_1 ,

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

As, h_2 and h_3 are constant with respect to h_1

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

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

Contd..

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

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

Step2: Updating the weight of w_{11} :

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

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

Skip gram

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

Step - 2 Window Size 1

Step - 3 Get one hot encoding for each word

Contd..

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

Step - 1

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

Step - 4

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

One Hot Encoding

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

Training Data

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

product

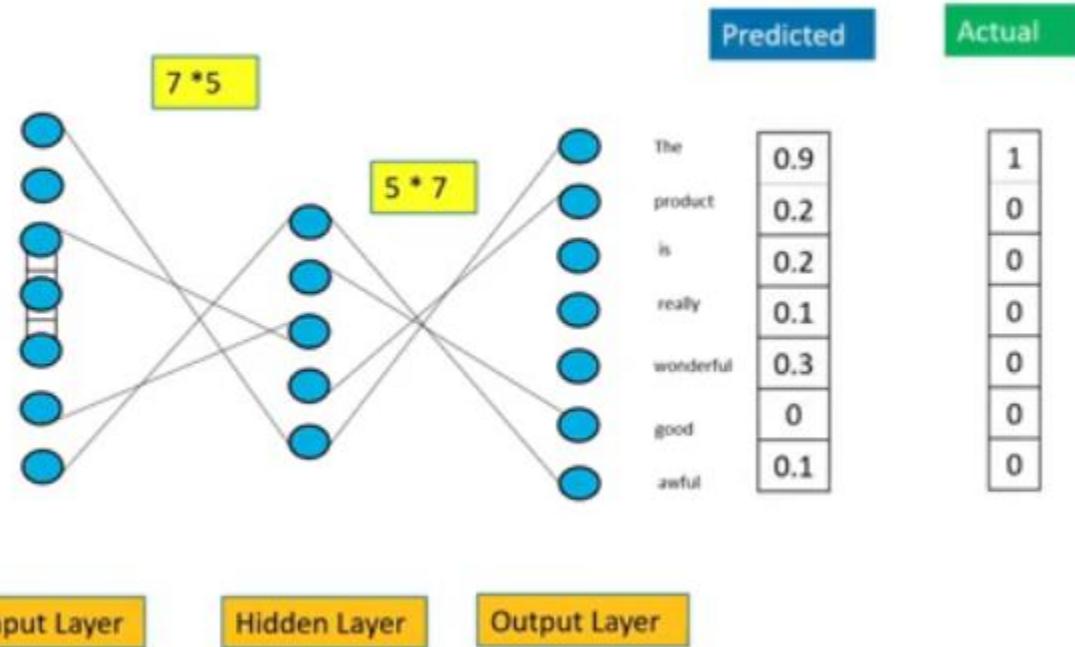
0
1
0
0
0
0
0

The

1
0
0
0
0
0
0

Contd..

One hot encoding of - Product



Thank You... 😊

- Q&A
- Suggestions / Feedback



Natural Language Processing

DSECL ZG565



BITS Pilani
Pilani Campus

Dr.Vijayalakshmi Anand

BITS-Pilani



Session 2

Date – 16DEC 2023

Time – 1.40pm to 3.40pm

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

Content Sequence Plan

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



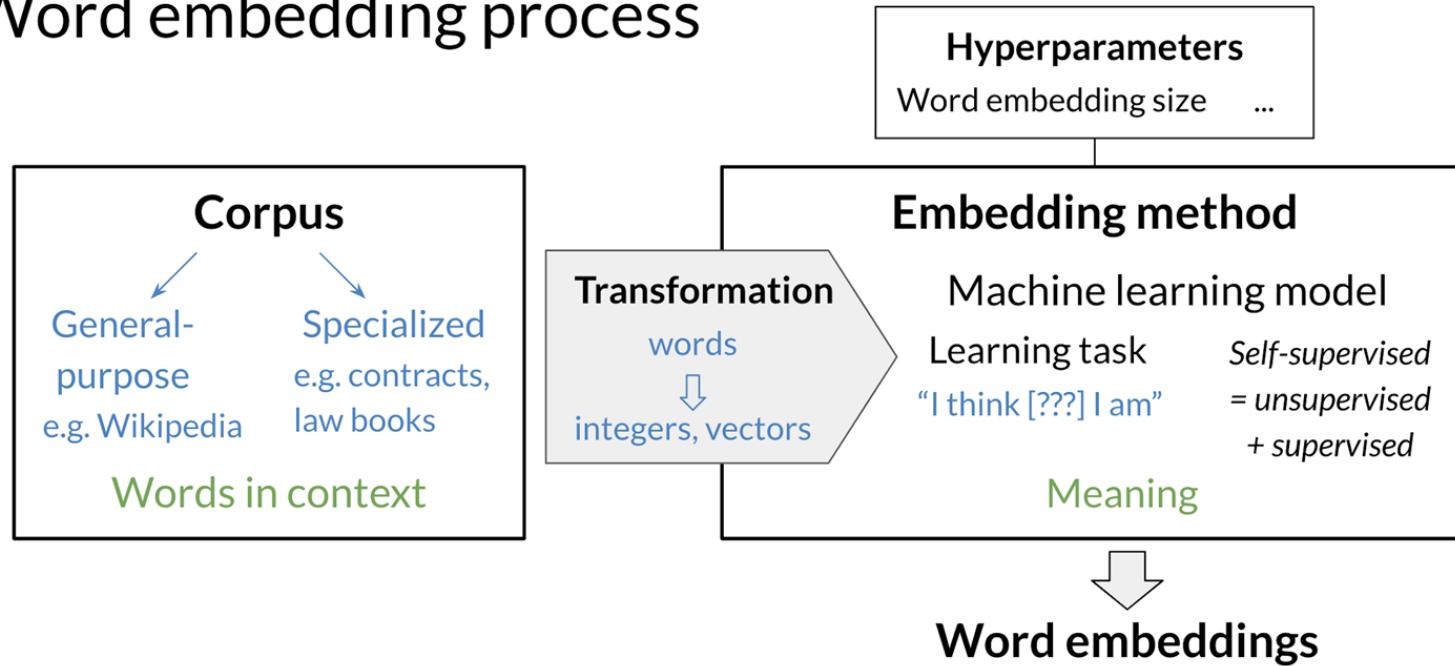
Prediction based Embedding

Sparse versus dense vectors

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

Word embeddings

Word embedding process



Common methods for getting short dense vectors

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

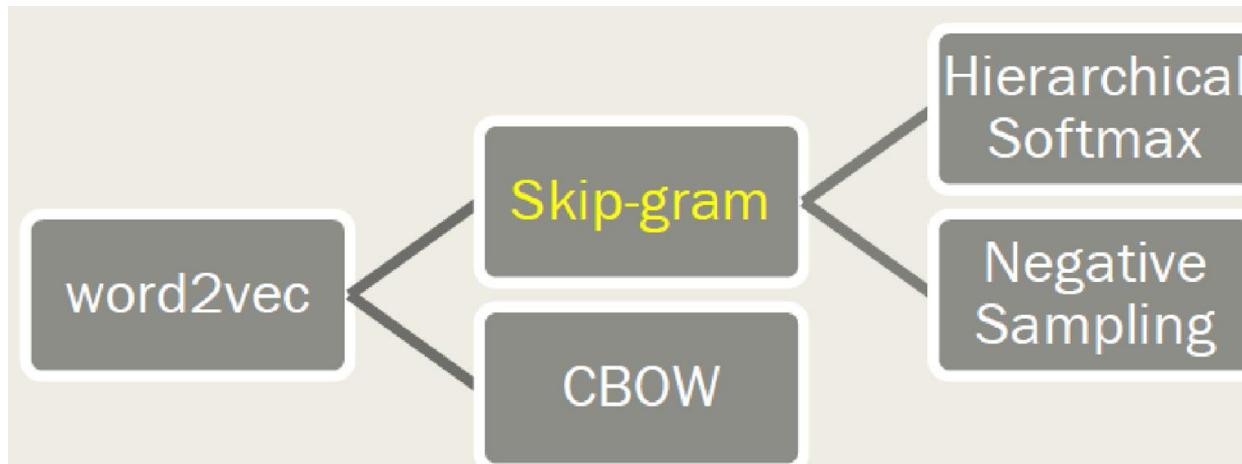
Word2vec -What is word2vec

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

Word2vec

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

Word2vec -Types of word2vec



Basic word representation

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

1-hot representation

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

I want a glass of orange _____

I want a glass of apple _____.

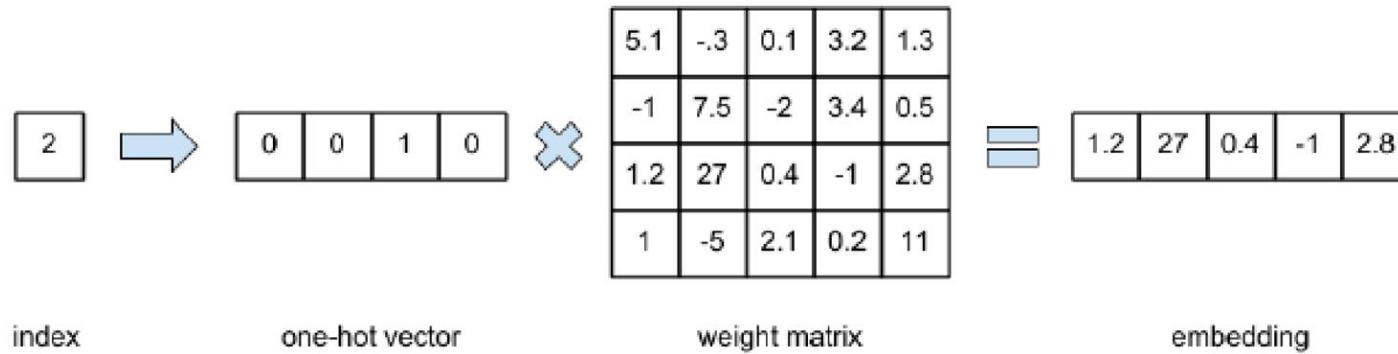
Featurized representation: word embedding

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

I want a glass of orange

I want a glass of apple

Word embeddings



CBOW [Continuous bag of words]

What is CBOW?

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

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

Working of CBOW with example

Corpus

The product is really good

The product is wonderful

The product is awful

Step1:

combine the sentences

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

Step2:

Select window size

Contd..

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



Window Size

1

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



Window Size

2

Contd..

Step3: Get one hot encoding for each word

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

Step4:Training data

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

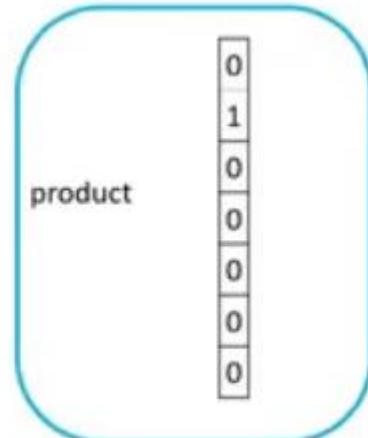
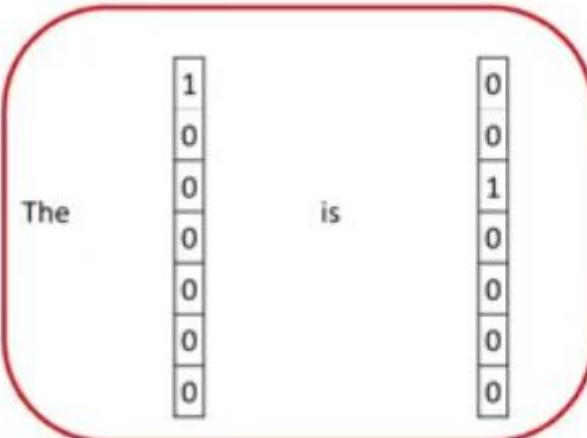
Contd..

One Hot Encoding

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

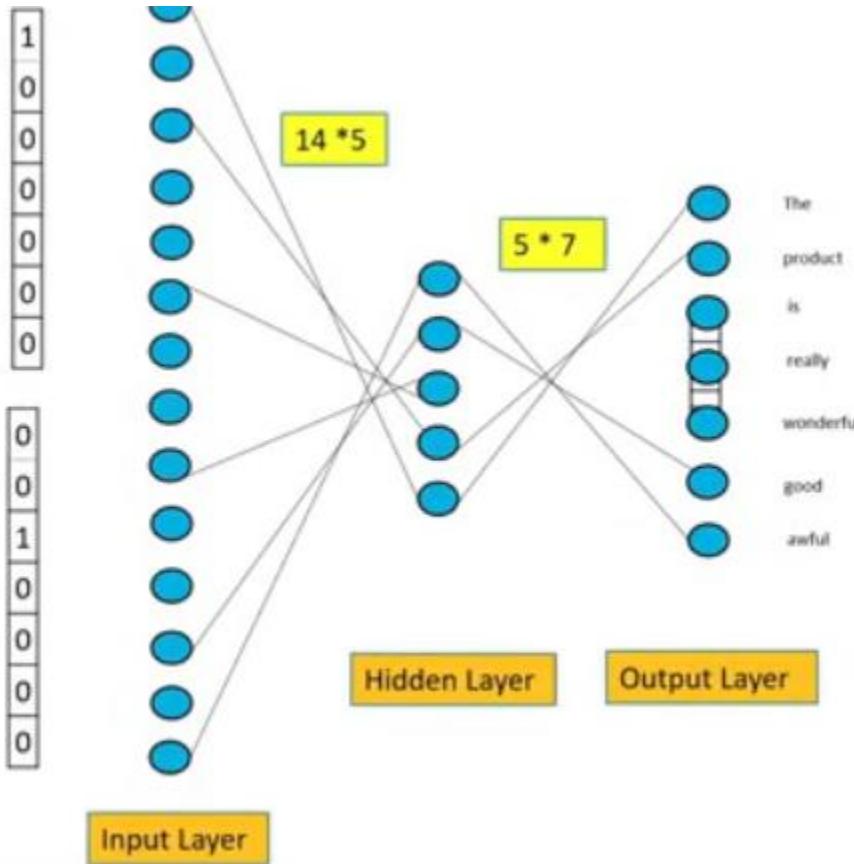
Training Data

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



Contd..

One hot encoding of - The

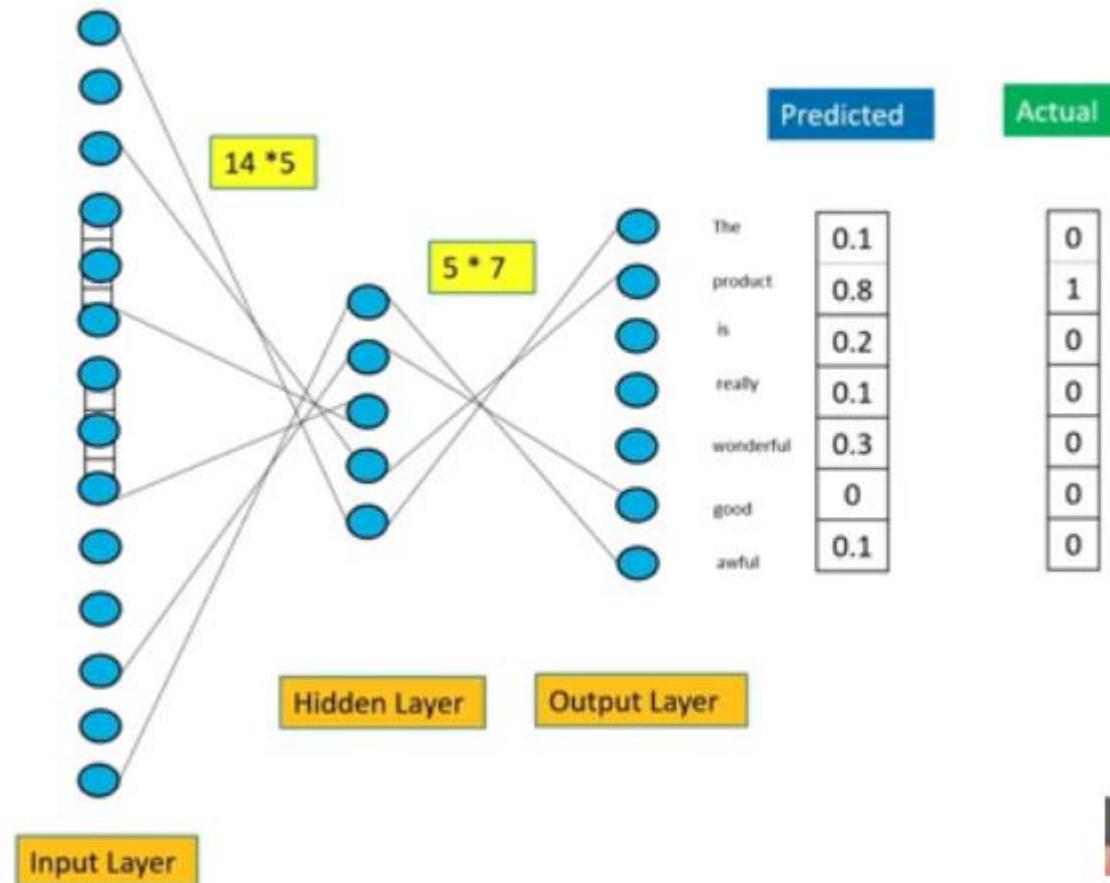


One hot encoding of - is

Cond..

One hot encoding of - The

One hot encoding of - is



Word Embedding matrix

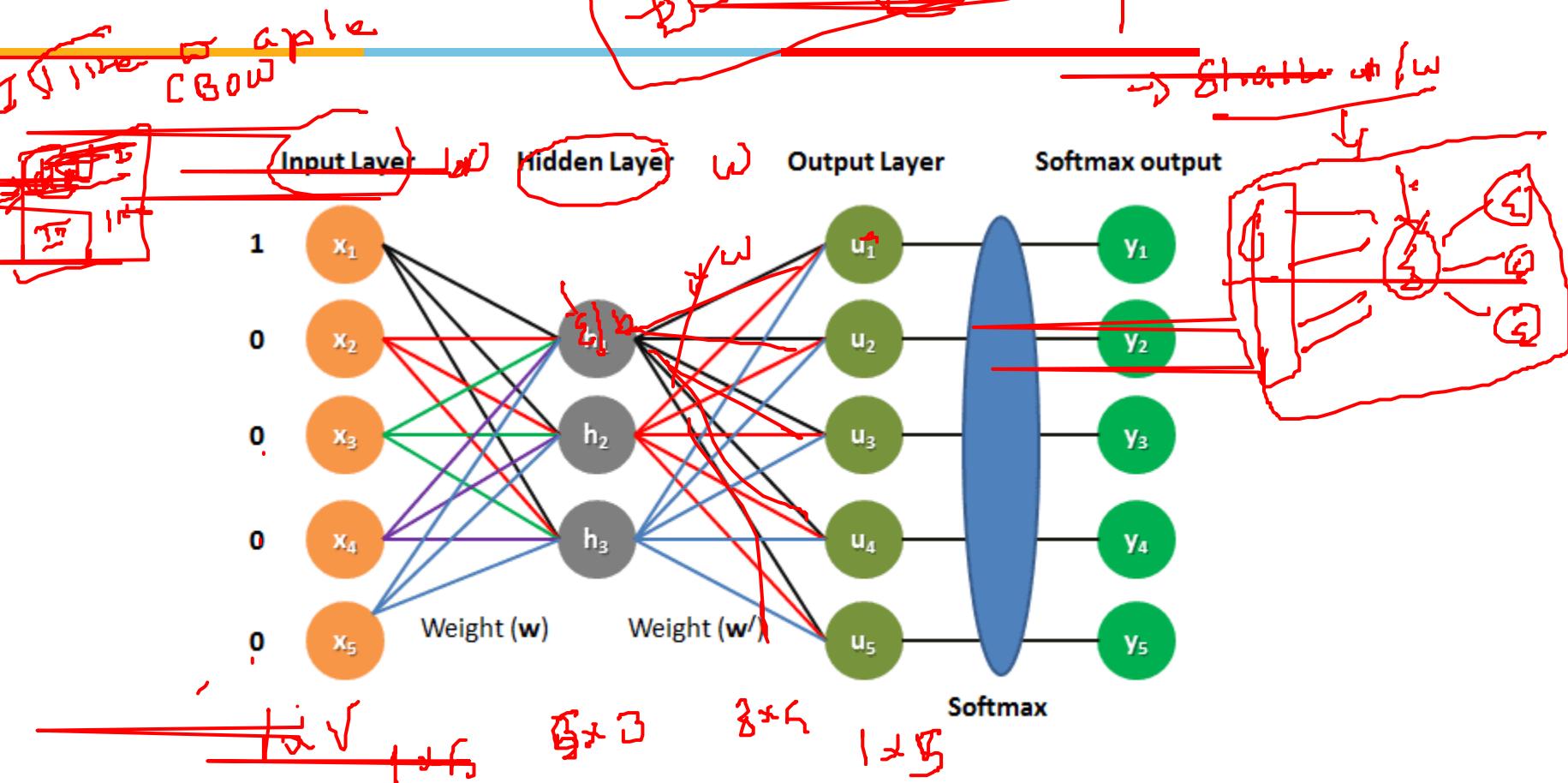
5 * 7

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

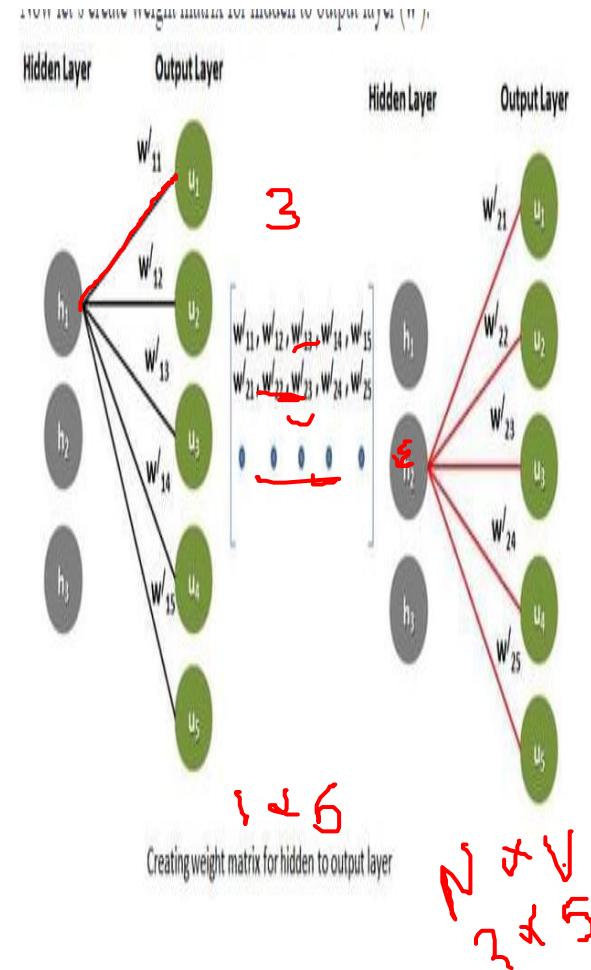
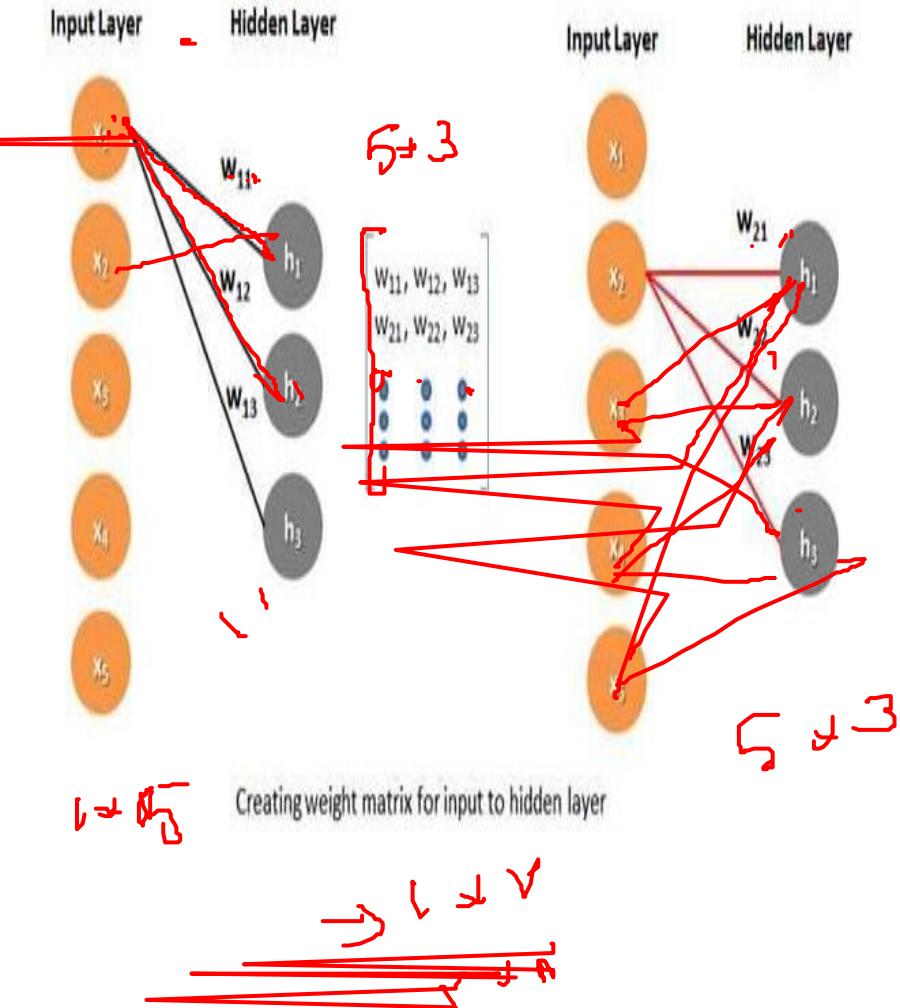
Training model

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

Model architecture



Weighed matrix



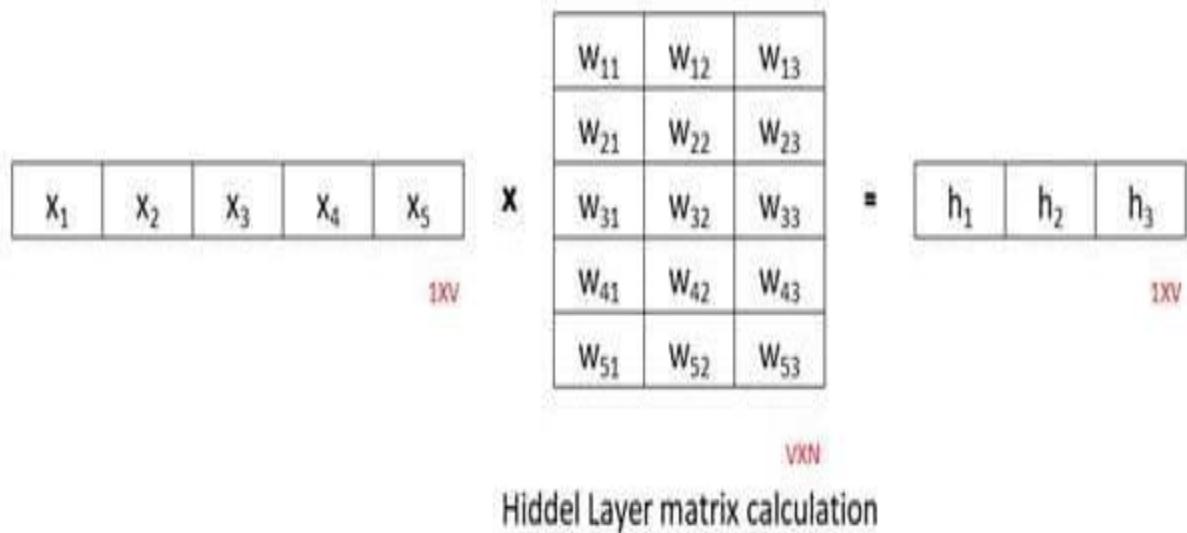
CBOW Vectorized form:

$$\begin{array}{c|c}
 \begin{matrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{matrix} & \times \\
 \begin{matrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \\ w_{51} & w_{52} & w_{53} \end{matrix} & \times \\
 \begin{matrix} h_1 \\ h_2 \\ h_3 \end{matrix} & \times \\
 \begin{matrix} w'_{11} & w'_{12} & w'_{13} & w'_{14} & w'_{15} \\ w'_{21} & w'_{22} & w'_{23} & w'_{24} & w'_{25} \\ w'_{31} & w'_{32} & w'_{33} & w'_{34} & w'_{35} \end{matrix} & \times \\
 \begin{matrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{matrix} & \times \\
 \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{matrix} &
 \end{array}$$

1XV VVN 1XN NXV 1XV 1XV

CBOW vectorized form

Forward Propagation



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

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

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

⋮ ⋮ ⋮

Contd..

Calculate output layer matrix (u):

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

Output Layer matrix calculation

So now:

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

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

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

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

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

contd

Calculate final Softmax output (y):

u_1	y_1
u_2	y_2
u_3	y_3
u_4	y_4
u_5	y_5

Softmax

1XV

1XV

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

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

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

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

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

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

...

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

Error calculation – log loss function

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

w_t = Target word

w_c = Context word

out out

I

~~1, 2, 3~~

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

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

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

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

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

Back propagation

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

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

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

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

Now as we know

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

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

Derivative of log function with chain rule.

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

Contd..

Taking derivative of E with respect to u_j :

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

$$= (y_j - t_j)$$

$$= \textcircled{e_j}$$

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

Contd..

So for first iteration,

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

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

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

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

So generalized form will looks like below:

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

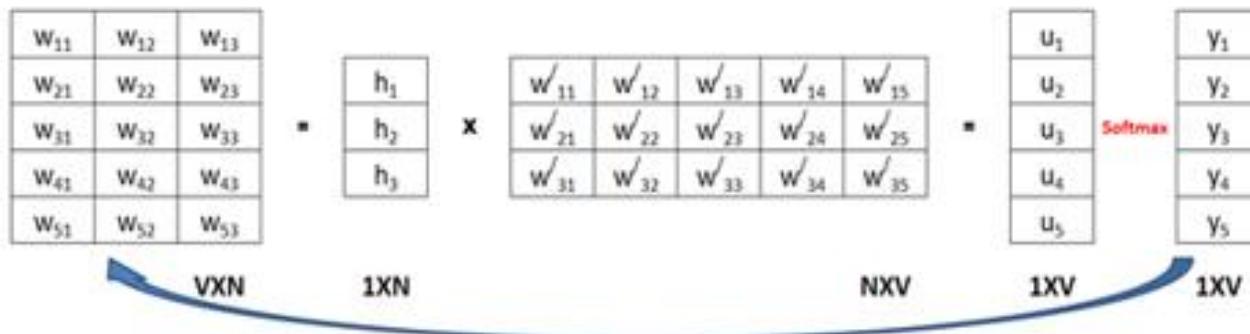
Step2: Updating the weight of w'_{11} :

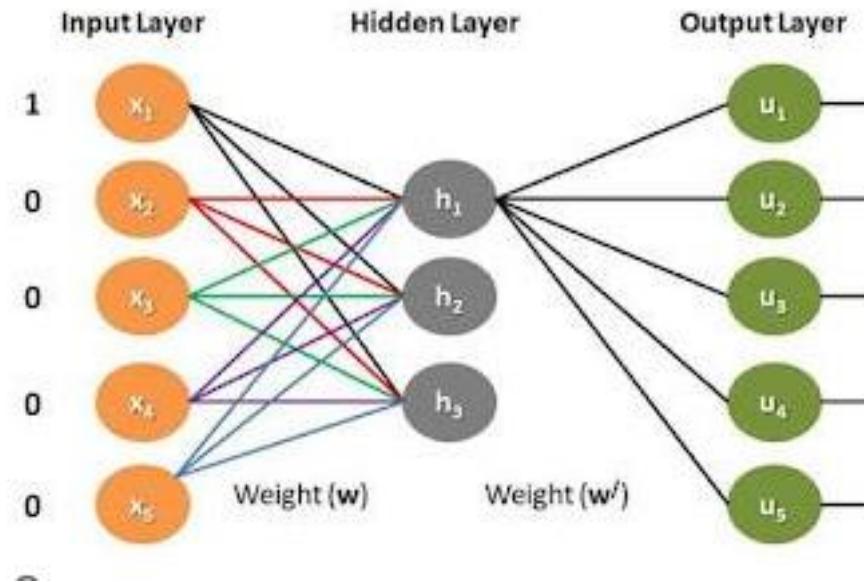
Step2: Updating the weight of w'_{11} :

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

Now Updating the first weight

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





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

As for u_1 and h_1 ,

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

As, h_2 and h_3 are constant with respect to h_1

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

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

Contd..

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

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

Step2: Updating the weight of w_{11} :

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

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

Skip gram

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

Step - 2 Window Size 1

Step - 3 Get one hot encoding for each word

Contd..

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

Step - 1

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

Step - 4

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

One Hot Encoding

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

Training Data

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

product

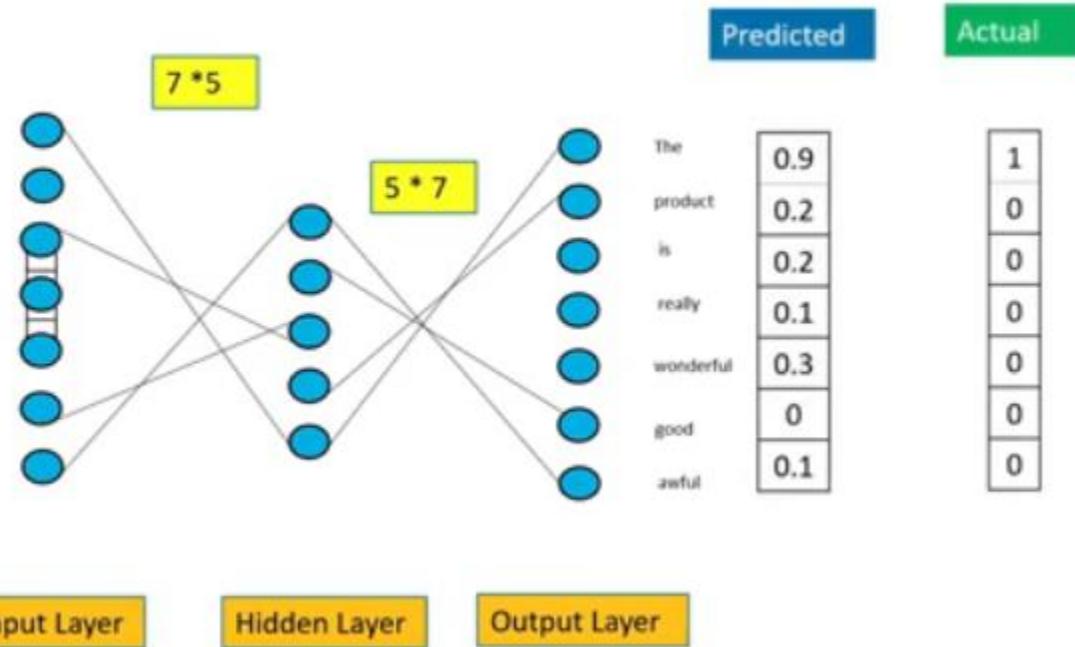
0
1
0
0
0
0
0

The

1
0
0
0
0
0
0

Contd..

One hot encoding of - Product



Skip gram negative example

Why negative sampling?

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

How it works?

K negative samples are randomly drawn from a noise distribution

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

Algorithm

The intuition of skip-gram is:

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

Skip gram training data

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

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

c1 c2 **target** c3 c4

Contd..

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

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

c1 c2 **[target]** c3 c4

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

positive examples +

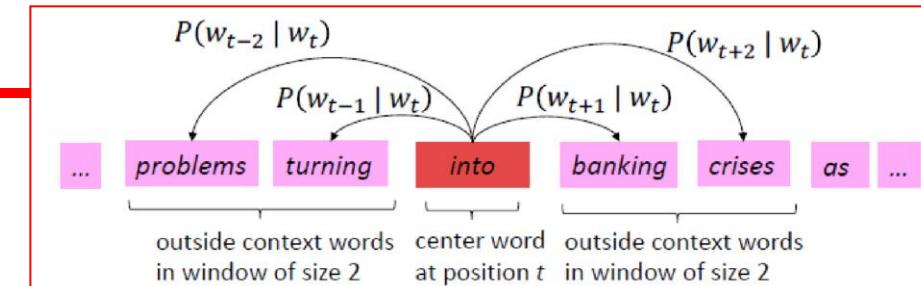
t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

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

How does negative sampling work

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



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

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

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

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

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

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

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

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

Contd..

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

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

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

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

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

↗ Negative examples

Skip gram goal

Our goal is to train a classifier such that,

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

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

Probability that word c is nota real context word for t

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

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

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

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

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

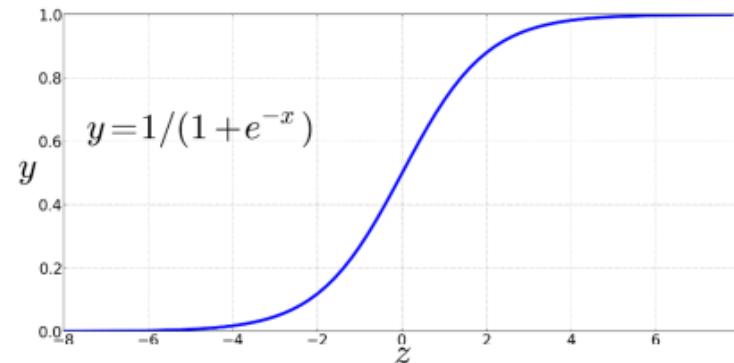
How to Compute $P(+|t,c)$?

Turning dot product into a probability



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

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



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

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

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

Learning skip-gram embeddings

Skip-Gram Training



- **Training sentence:**

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

c1 c2 **t** c3 c4

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

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

Contd..

- **Training sentence:**

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

c1 c2 **t** c3 c4

Create k (=2) negative examples.

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

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

How to select negative samples

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

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

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

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

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

Word2Vec Algorithm

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

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

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

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

Word2Vec Algorithm

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

- Stochastic gradient descent!

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

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

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

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

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

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

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

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

Learning skip-gram embeddings

Train using gradient descent

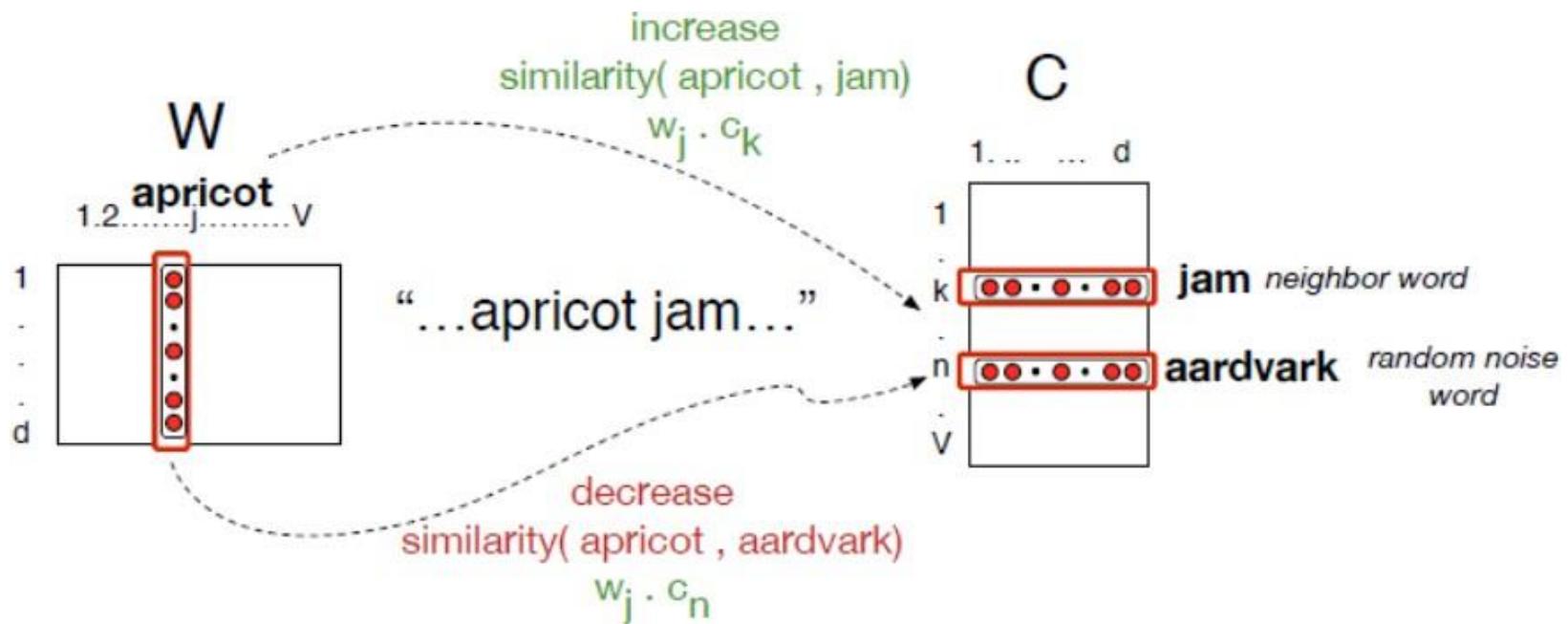


- We can then **use stochastic gradient descent to train to this objective**,
 - iteratively modifying the parameters (the embeddings for each target word t and each context word or noise word c in the vocabulary) to maximize the objective.
- **Skip-gram model** actually learns **two separate embeddings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

Learning skip-gram embeddings

Train using gradient descent

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



Learning skip-gram embeddings

Train using gradient descent

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



Thank you



Natural Language Processing



BITS Pilani

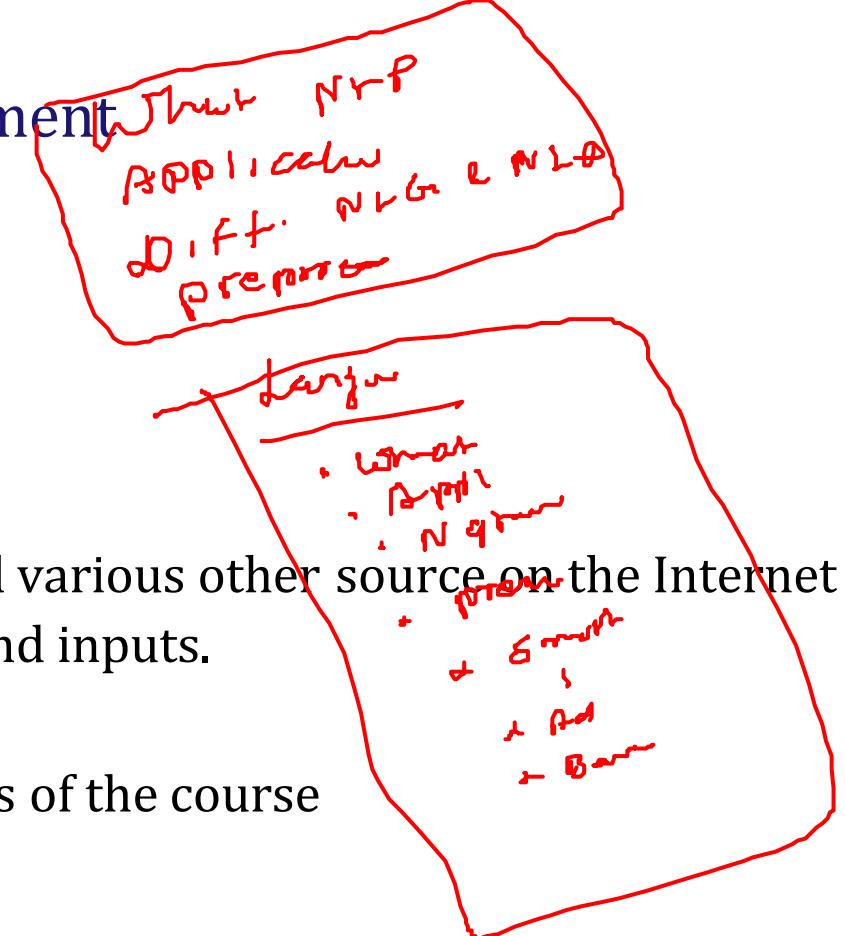
Pilani Campus

Dr. Vijayalakshmi Anand
BITS Pilani

Natural Language Processing

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

1) F → **Disclaimer and Acknowledgement**



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

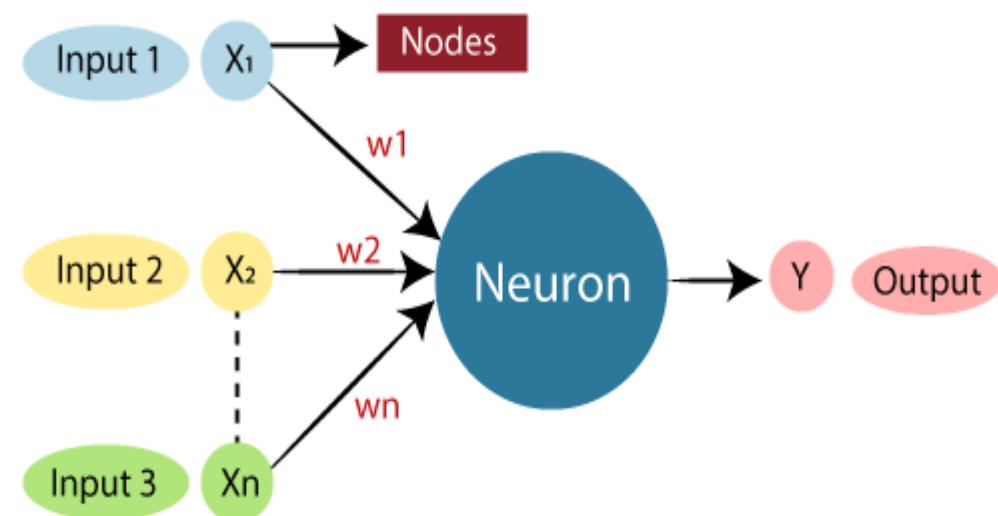
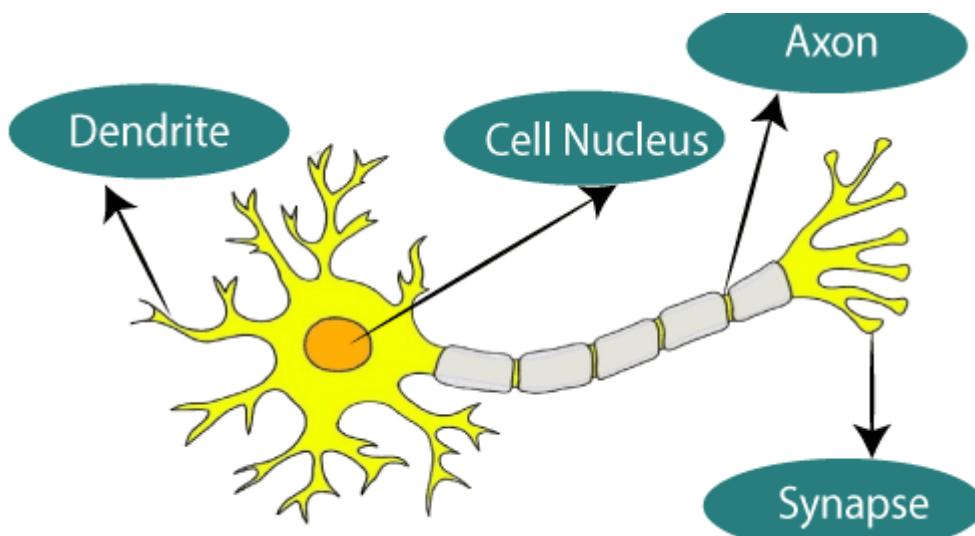
Session Contents

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

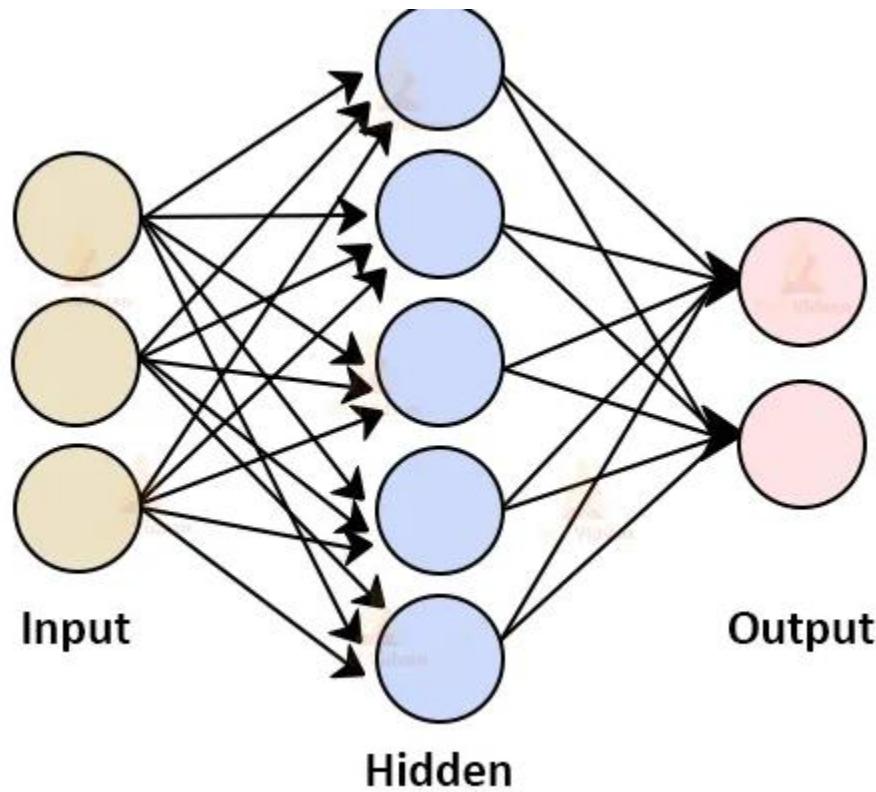
Neural network -Introduction

- What is neural network

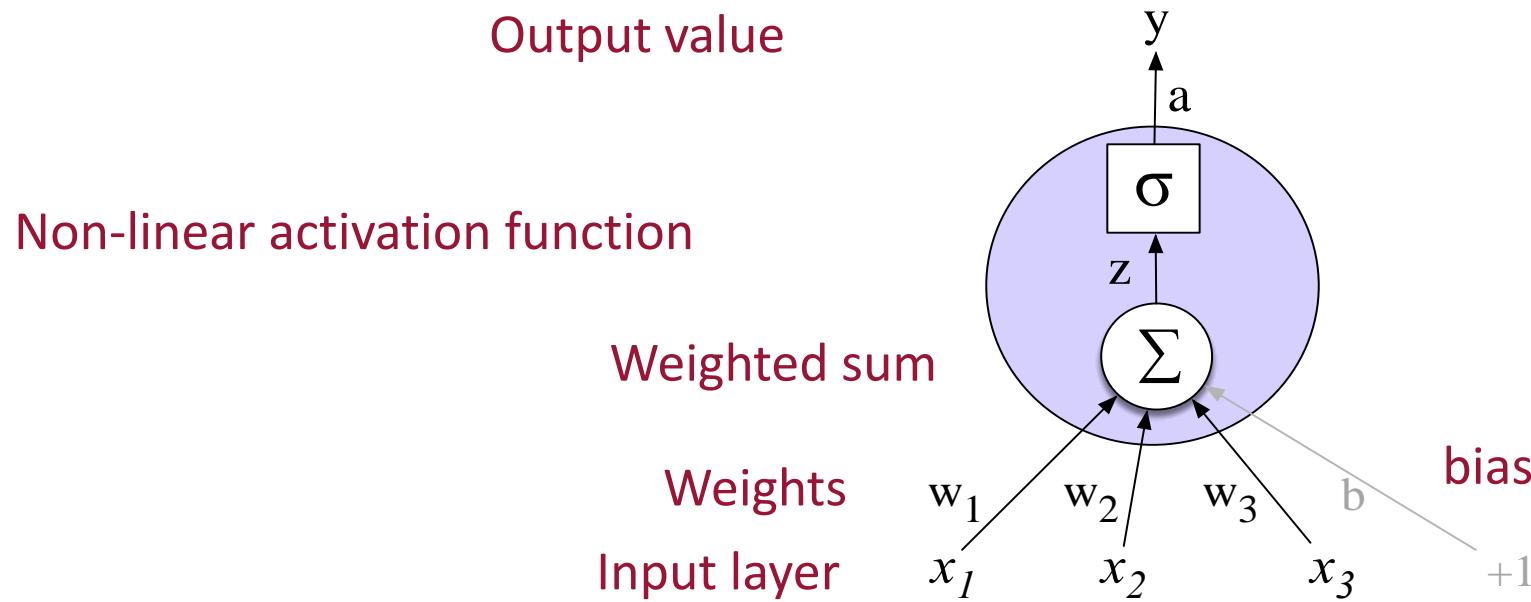
Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks. These neurons are known as nodes.



Architecture of neural networks

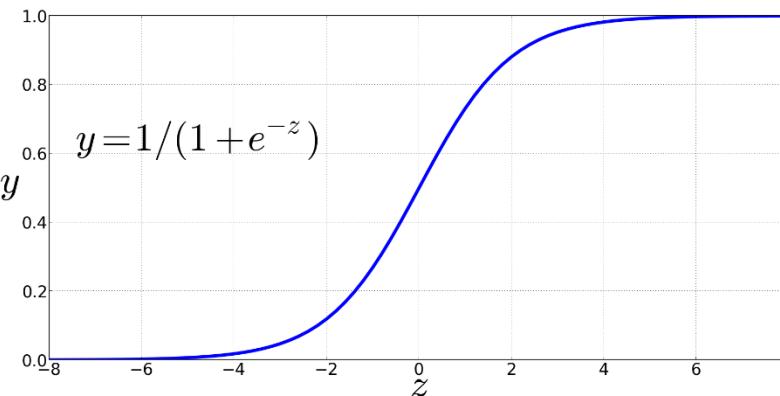


Neural Network-How it works



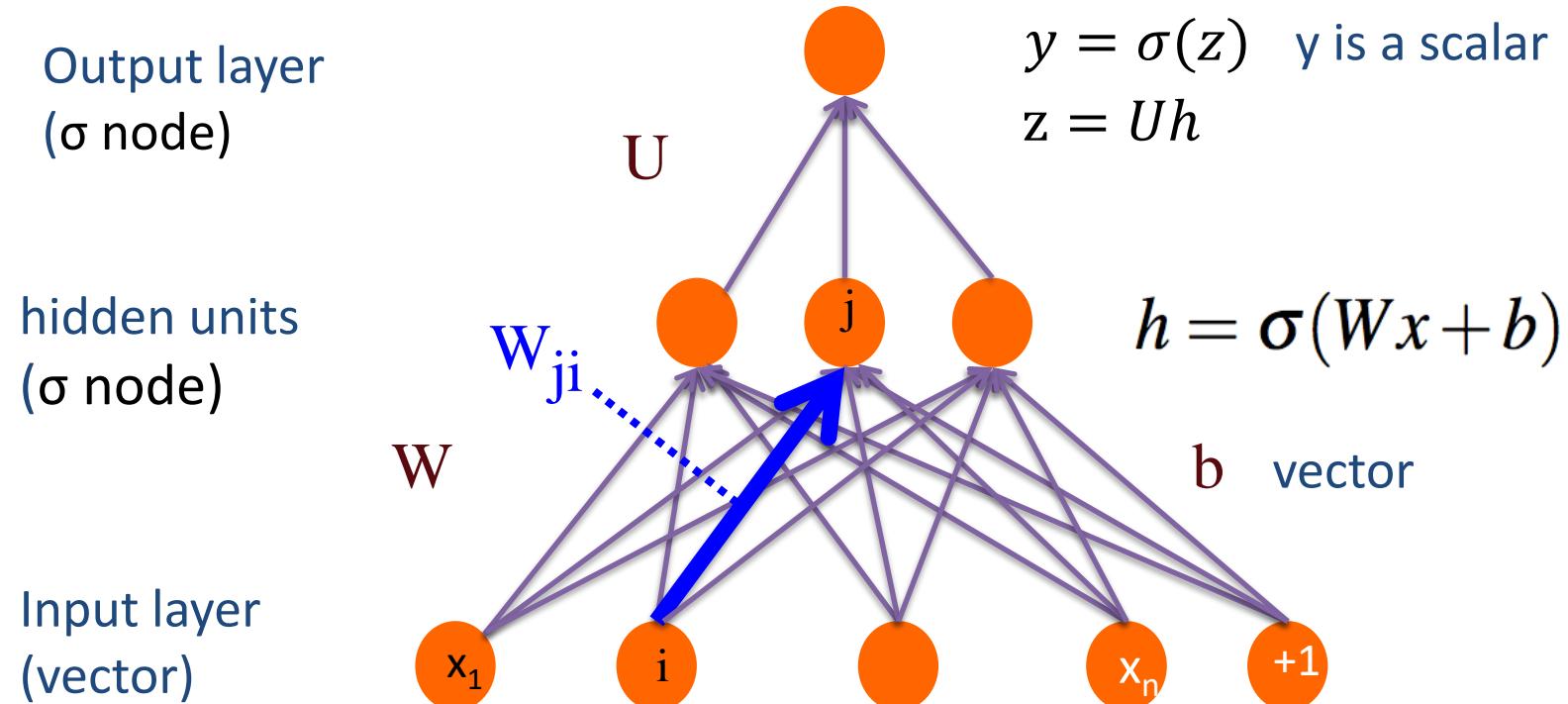
Sigmoid

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

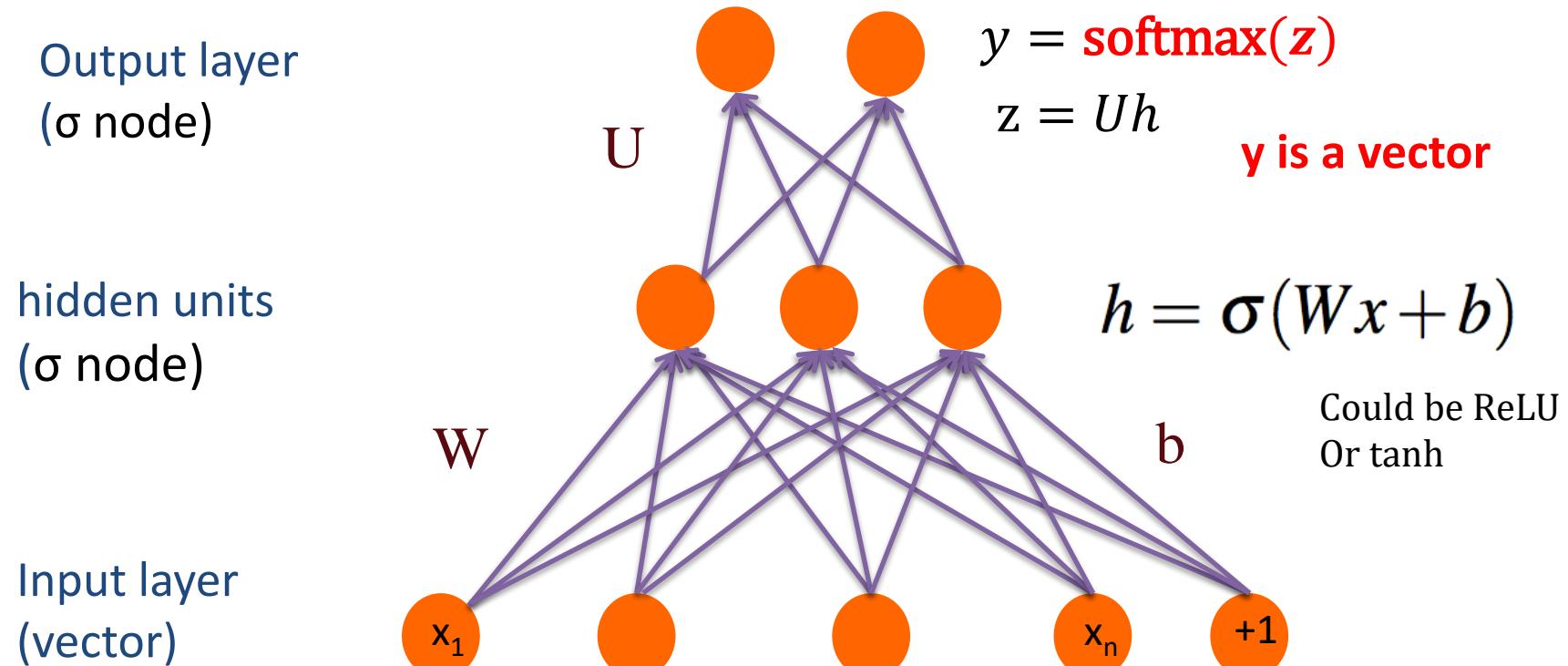


6

Two-Layer Network with scalar output



Two-Layer Network with softmax output



Reminder: softmax: a generalization of sigmoid

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

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

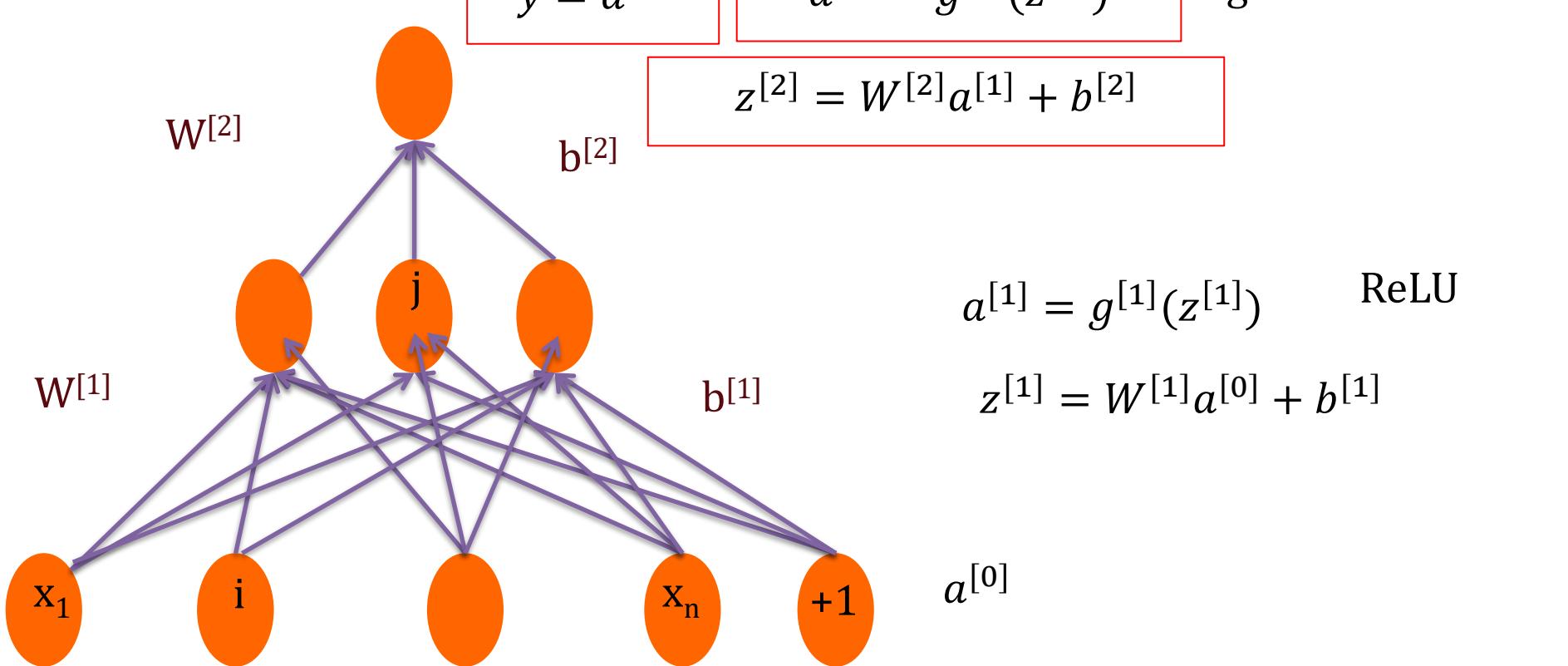
$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^k \exp(z_j)} \quad 1 \leq i \leq k$$

- Example:

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

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

Multi-layer Notation



Multi-layer Notation

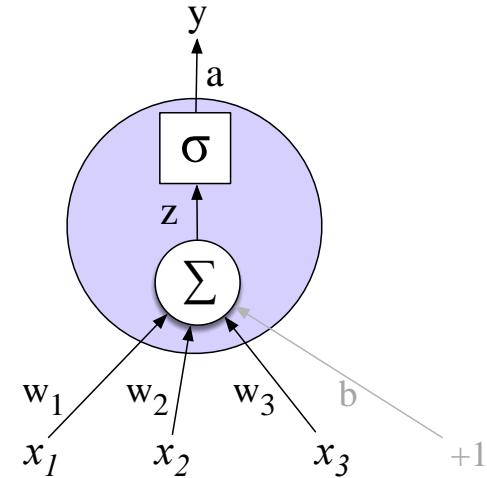
$$z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$$

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

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

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

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



for i in 1..n

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

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

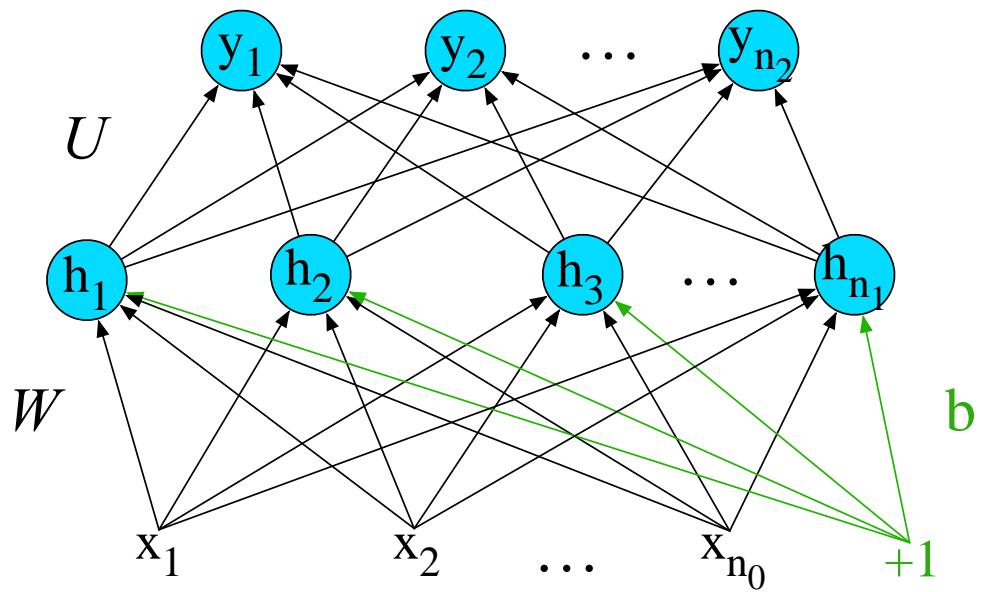
$$\hat{y} = a^{[n]}$$

Replacing the bias unit

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

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

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

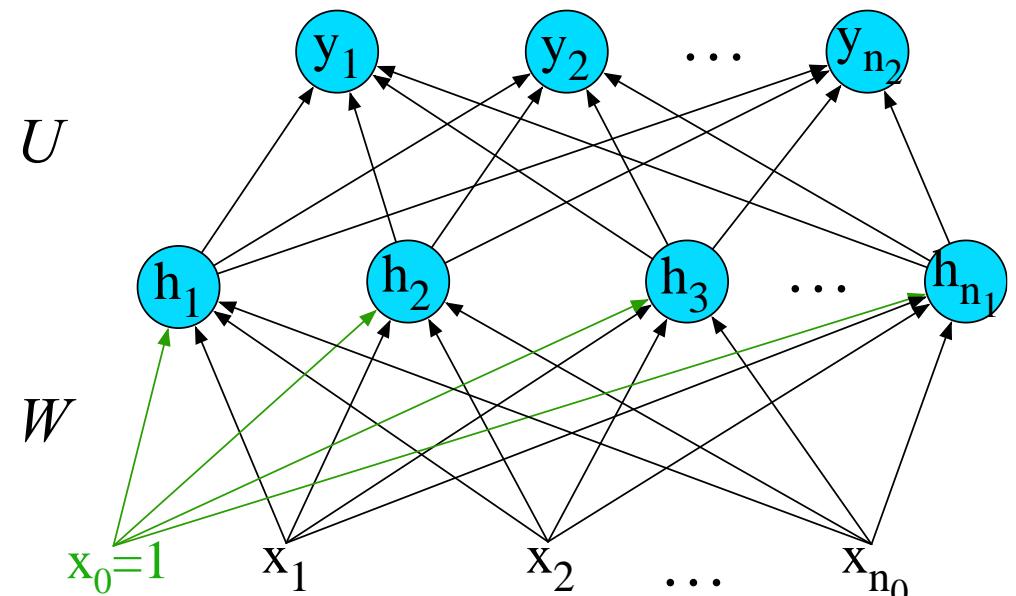


- We'll do this:

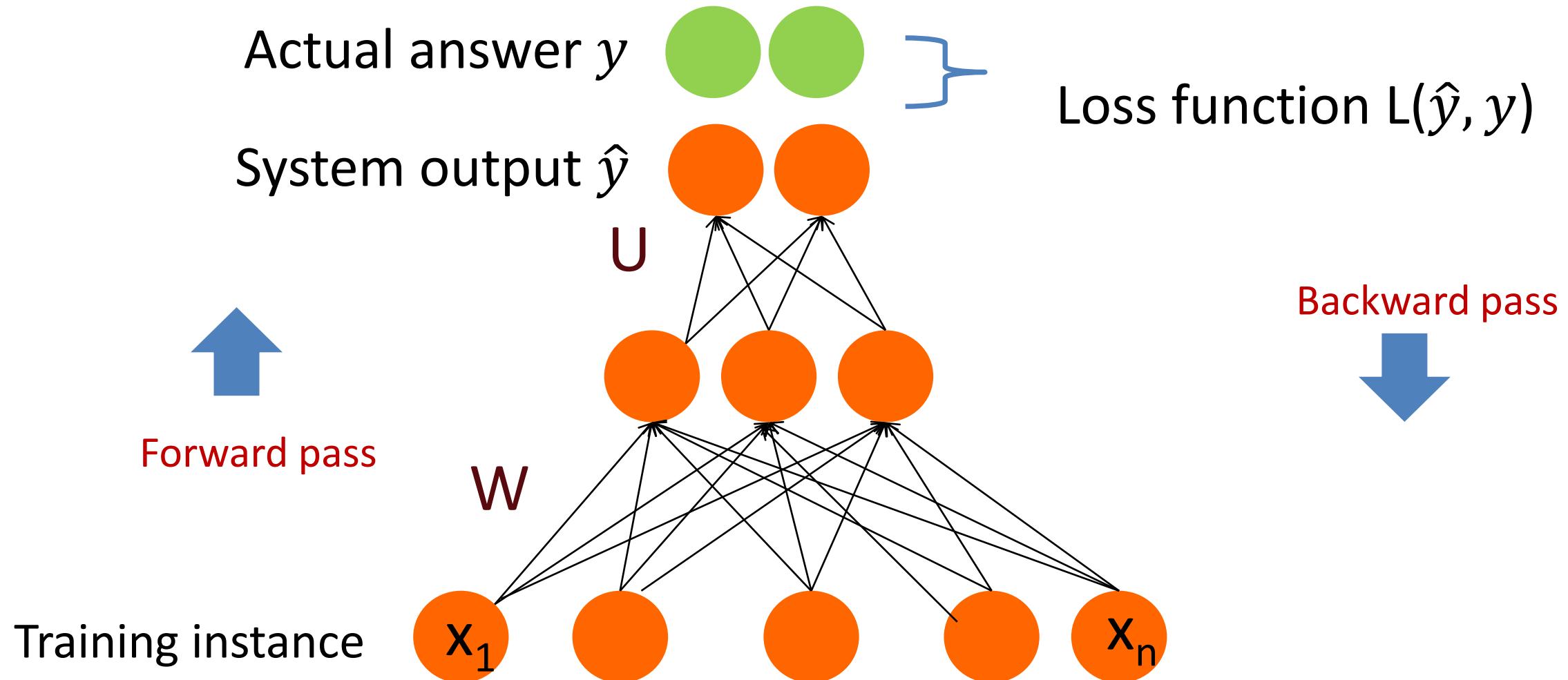
$$x = x_0, x_1, x_2, \dots, x_{n_0}$$

$$h = \sigma(Wx)$$

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



Intuition: training a 2-layer Network



Intuition: Training a 2-layer network

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

Reminder: Loss Function for binary logistic regression

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

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

Reminder: gradient descent for weight updates

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

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

- For logistic regression

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

Backward differentiation

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

Backprop

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



Application : Sentiment Analysis

Use cases for feedforward networks

Let's consider 2 (simplified) sample tasks:

1.Text classification → Sentiment analysis

2.Language modeling → News language model

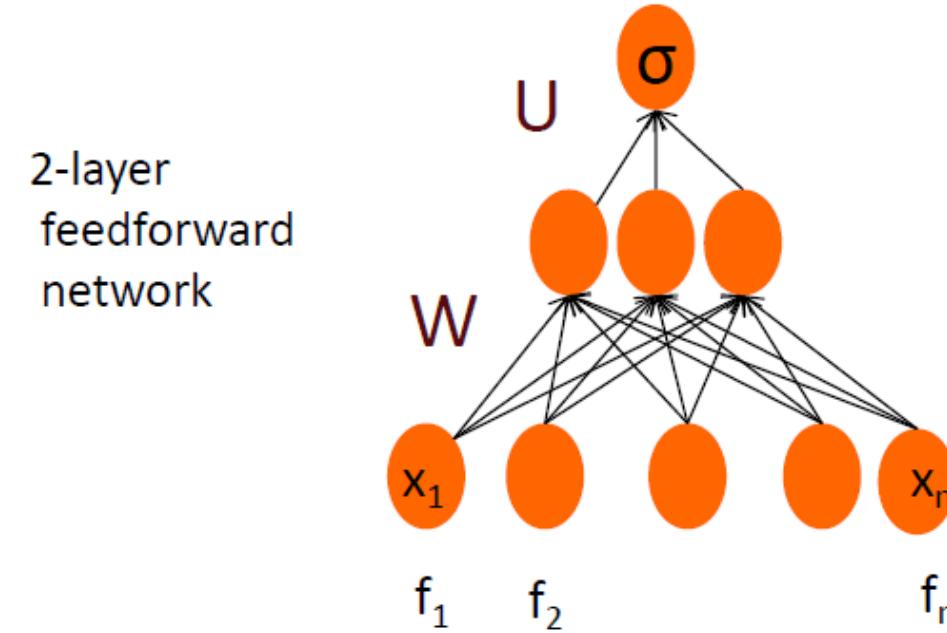
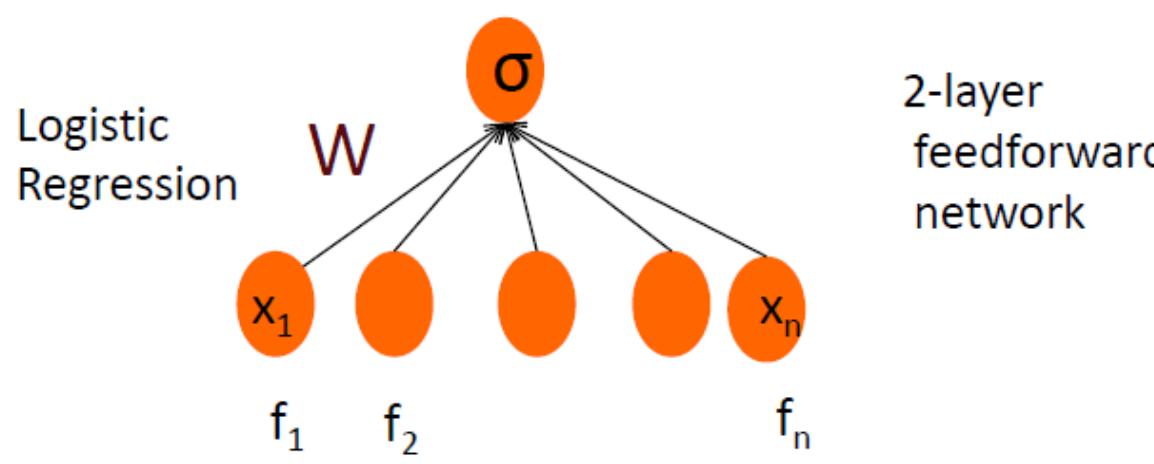
Sentiment example: does $y=1$ or $y=0$?

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

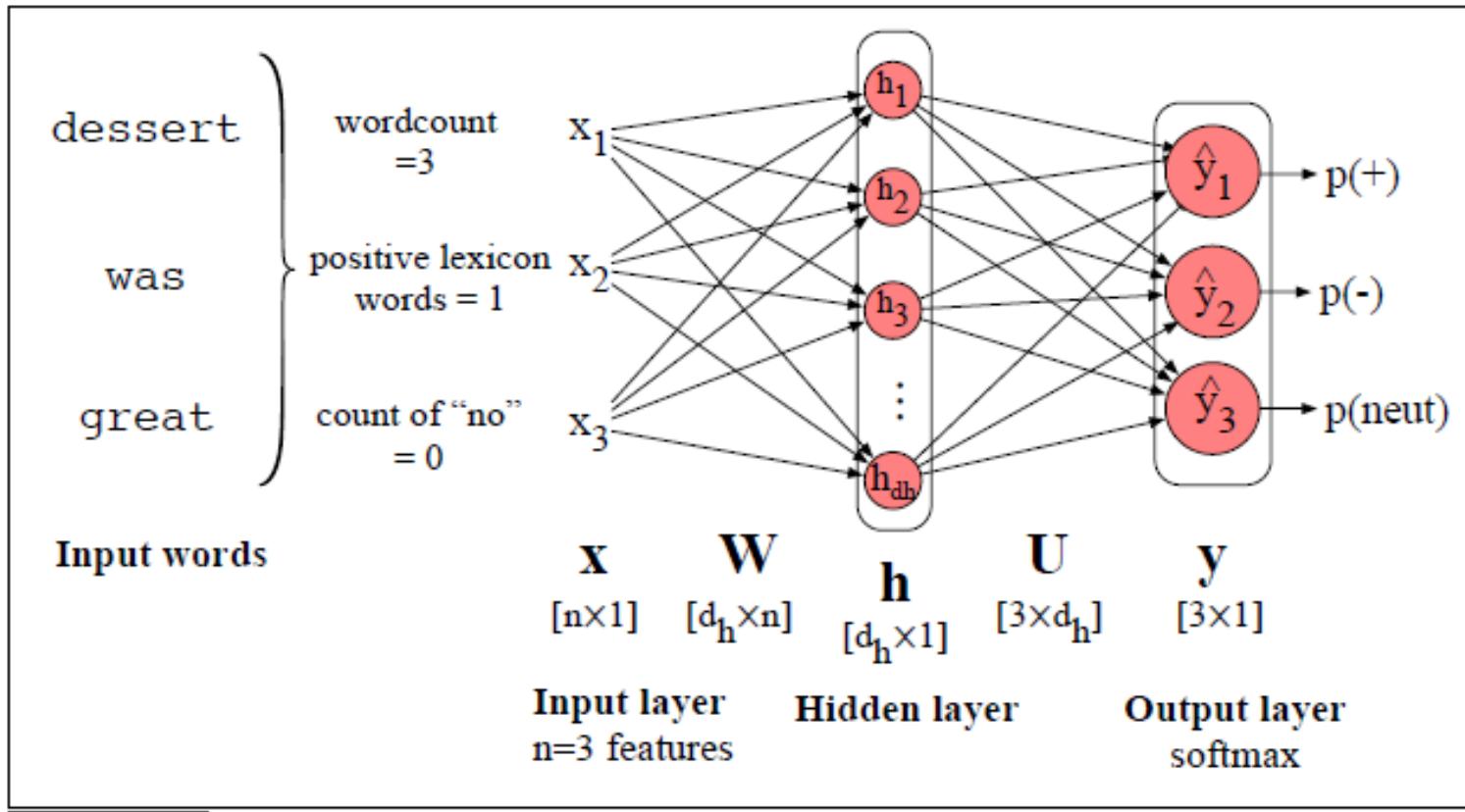
Feedforward nets for simple classification



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



Feedforward networks for NLP: Classification



(each x_i is a hand-designed feature)

$$\begin{aligned} \mathbf{x} &= [x_1, x_2, \dots, x_N] \\ \mathbf{h} &= \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \\ \mathbf{z} &= \mathbf{U}\mathbf{h} \\ \hat{\mathbf{y}} &= \text{softmax}(\mathbf{z}) \end{aligned}$$

$$\begin{aligned} (3 \times 1) \quad (4 \times 1) \quad 3 \\ &\times \quad &\times \\ 3 \times 1 & \quad (5 \times 3) \\ &\times \quad &\times \\ (5 \times 1) & \quad (5 \times 5) \\ &\times \quad &\times \end{aligned}$$

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

Classification: Sentiment Analysis



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

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

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

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

Sentiment Features

Classifying sentiment using logistic regression

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

$b = 0.1$

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

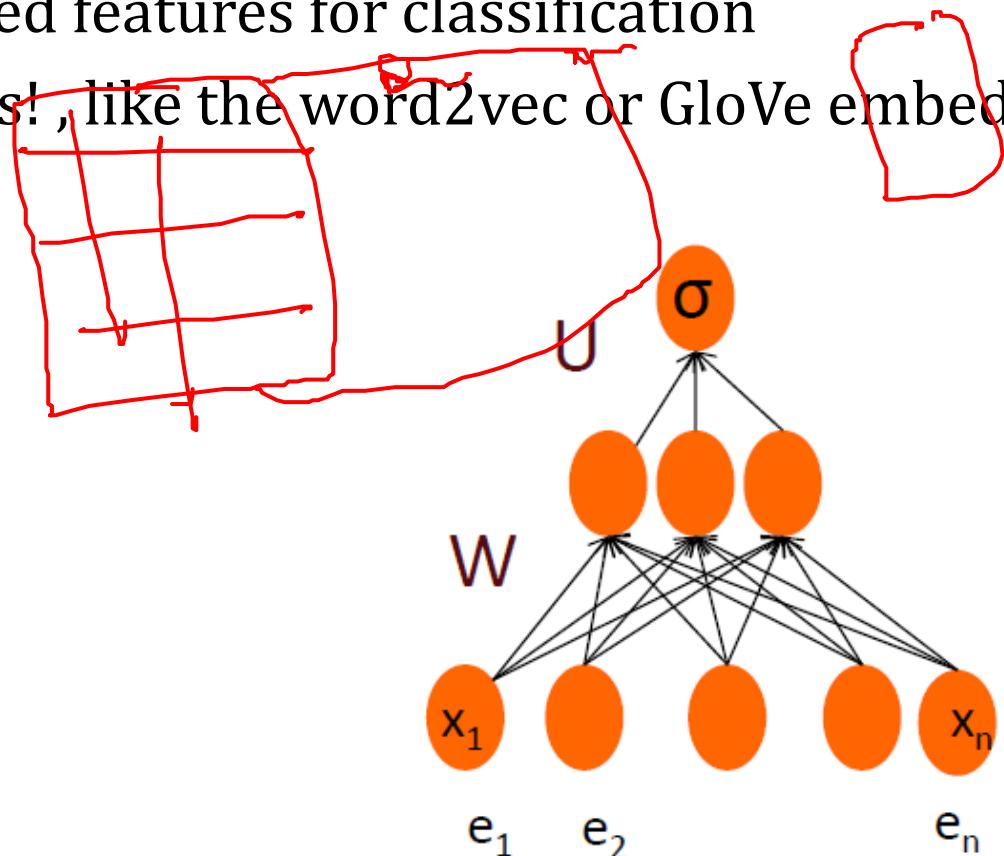
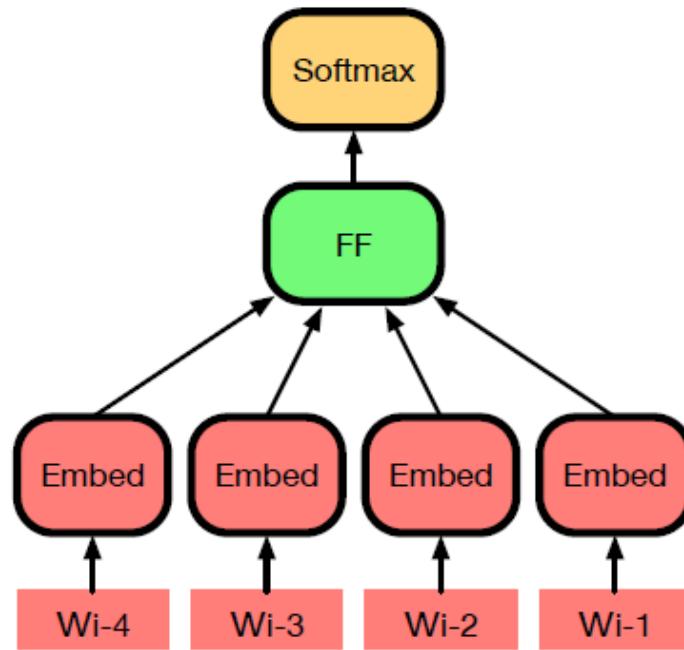
$$\in \mathcal{Z}_t$$

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

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

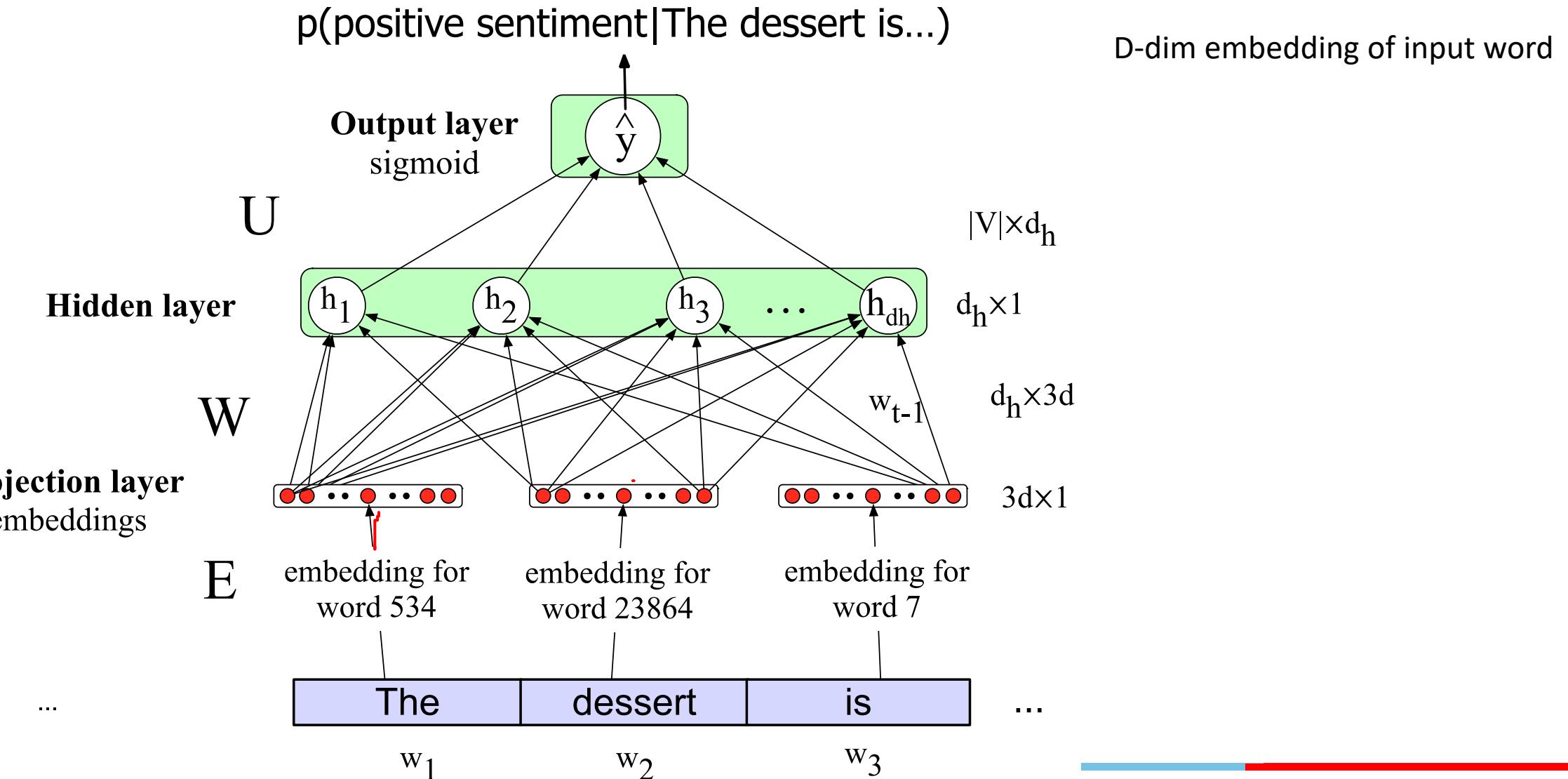
Representation learning

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



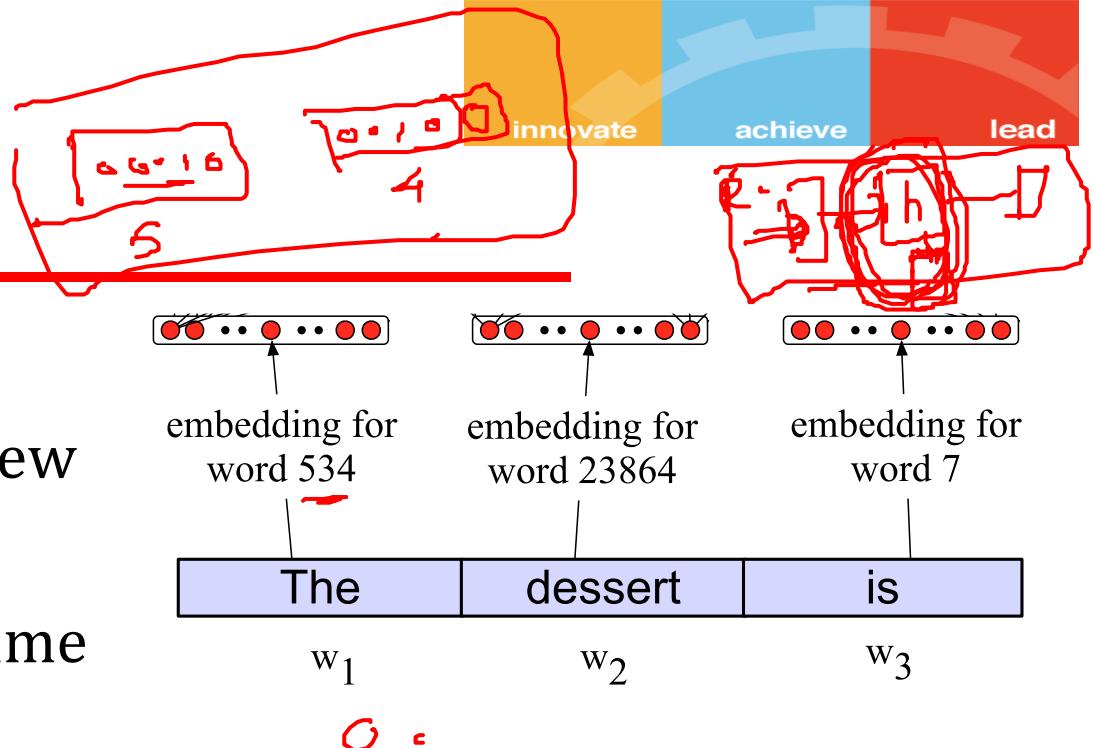
Neural Net Classification with embeddings as input features

lead



Issue: texts come in different sizes

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



Using Pool embeddings

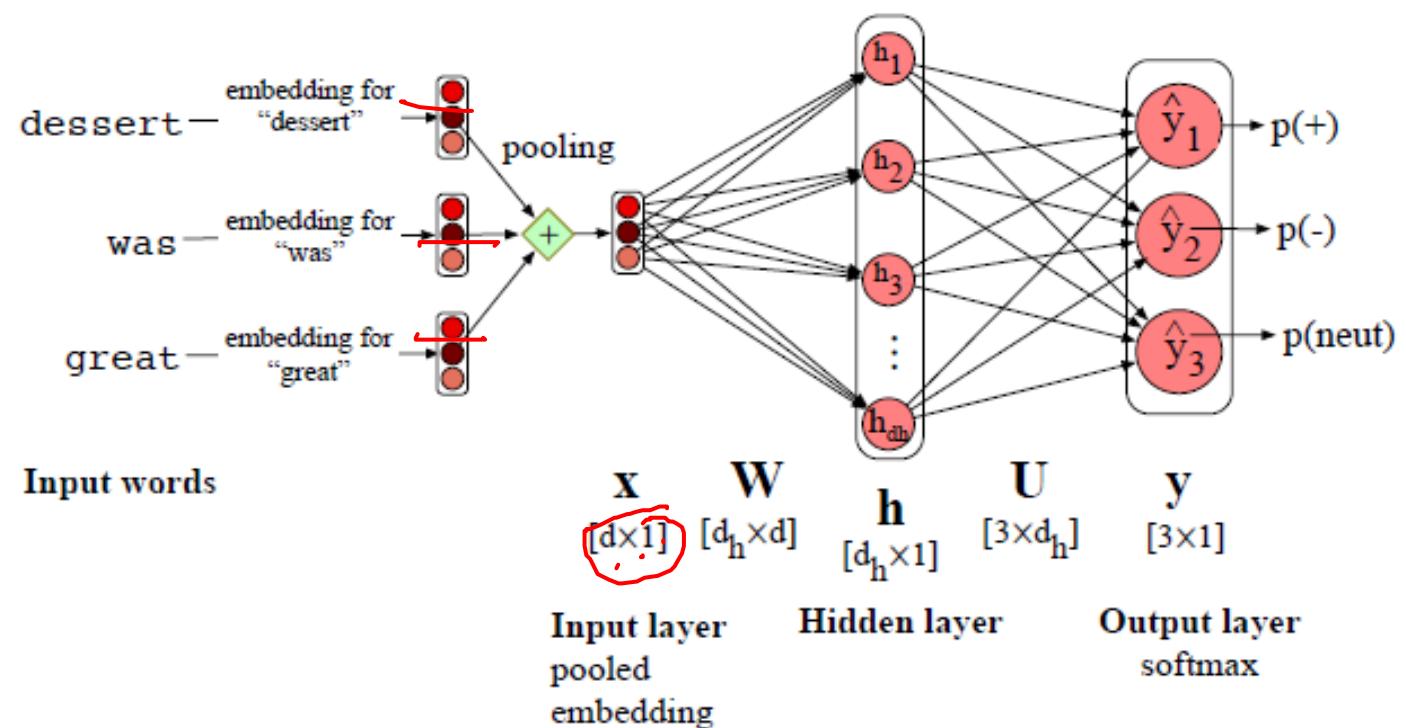
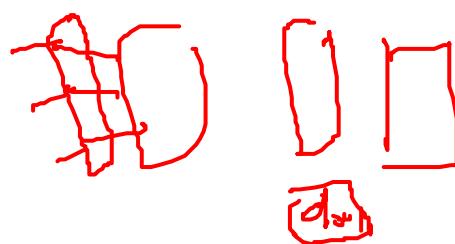


Figure 7.11 Feedforward sentiment analysis using a pooled embedding of the input words.

w₁...w_n: n input tokens
 e(w₁)....e(w_n) : n embedding (each d-dim)
 X: single embedding of d-dim, mean of all embedding

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

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

vectorising

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



Neural Language Model

Types of neural language model

- Statistical model
- Neural language model



The day will be

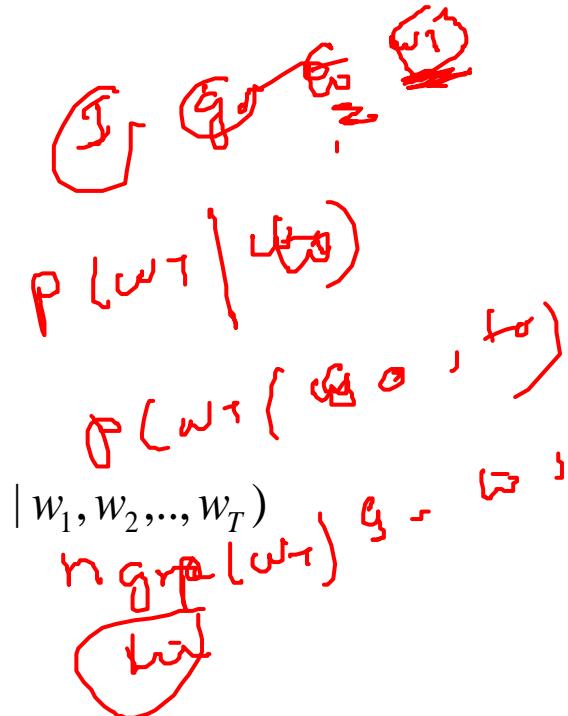
Probabilistic Language Models

innovate

achieve

lead

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



Example

day

Have a good ↗

weather

terrible

time

4-gram language model:

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

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

$\cancel{P(P | B)}$ X

$\cancel{P(P, B)}$

$C(B)$

Curse of dimensionality

Consider this sentences

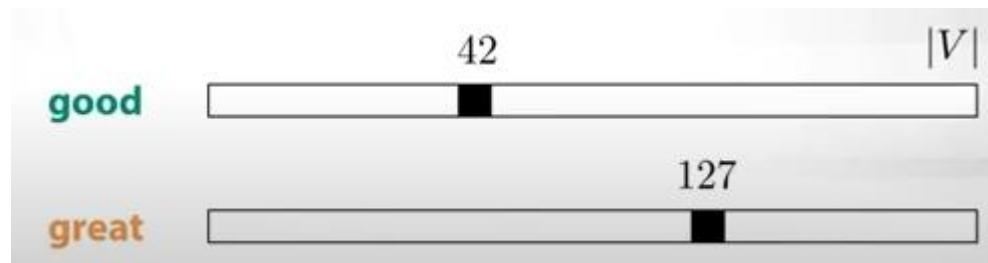
Have a good day

Have a great day

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

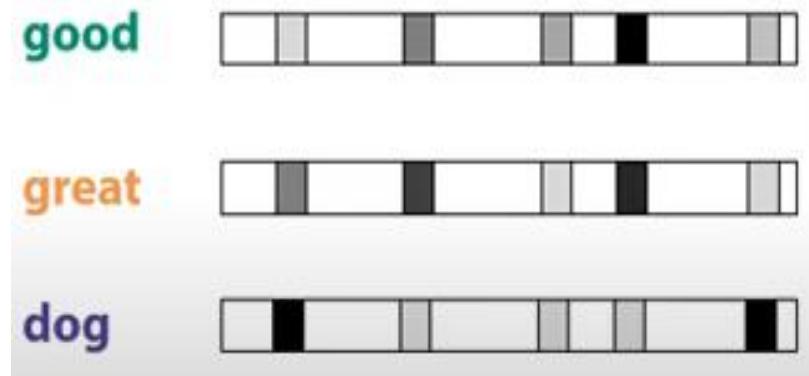
What happens in smoothing ?

Since
= 0

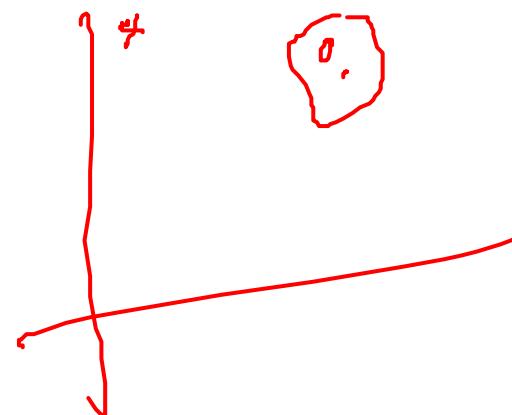


How to generalize better

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



$R - \text{man} + \text{woman} = \text{Queen}$



Neural Probabilistic Language Models



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

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

Neural Language Models (LMs)

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

Simple feedforward Neural Language Models

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

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

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

Word Embeddings

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

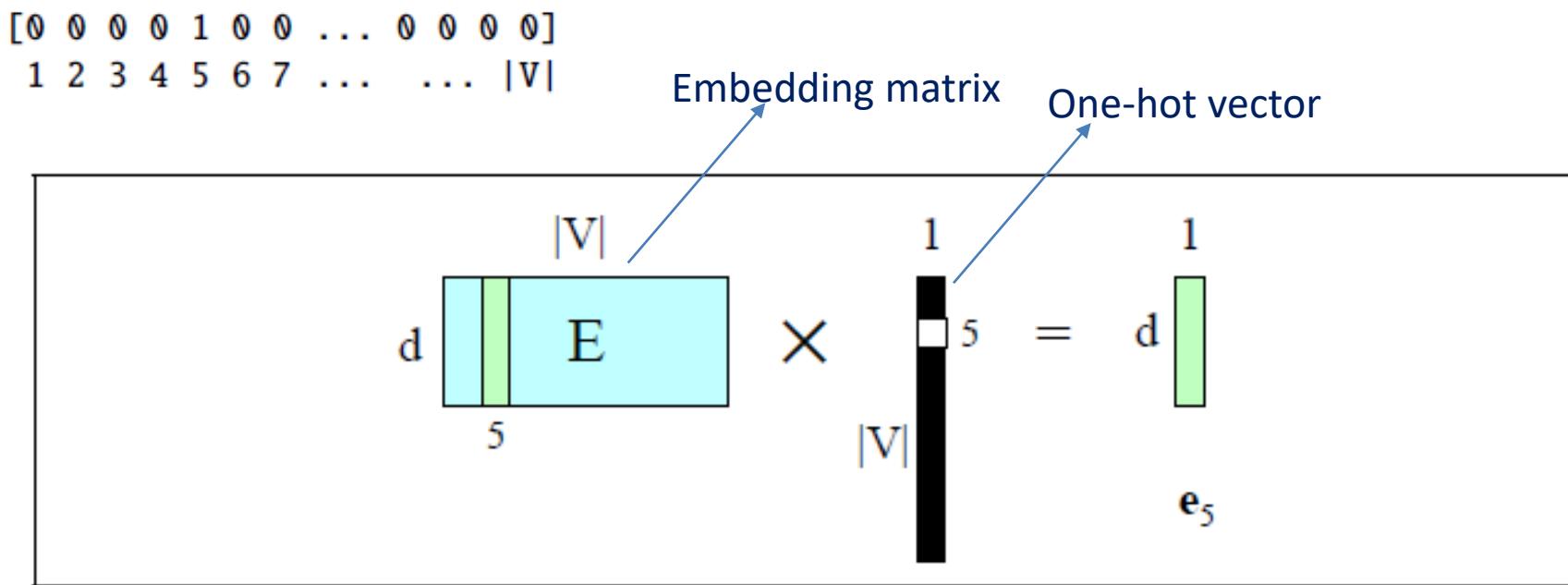


Figure 7.12 Selecting the embedding vector for word V_5 by multiplying the embedding matrix E with a one-hot vector with a 1 in index 5.

Why Neural LMs work better than N-gram LMs

embeddings

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

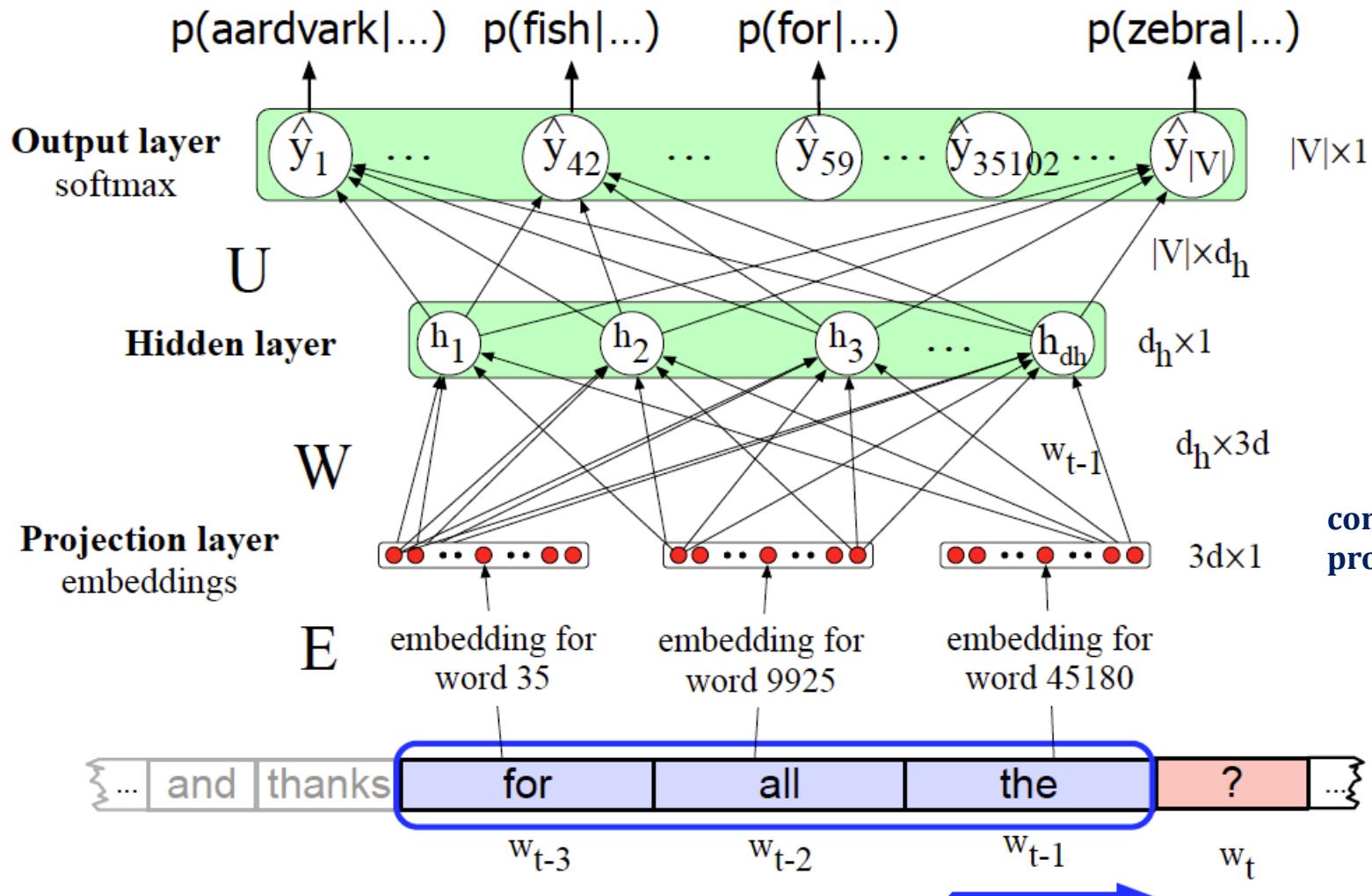
Training data:

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

Test data:

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

Neural Language Model



Equations:

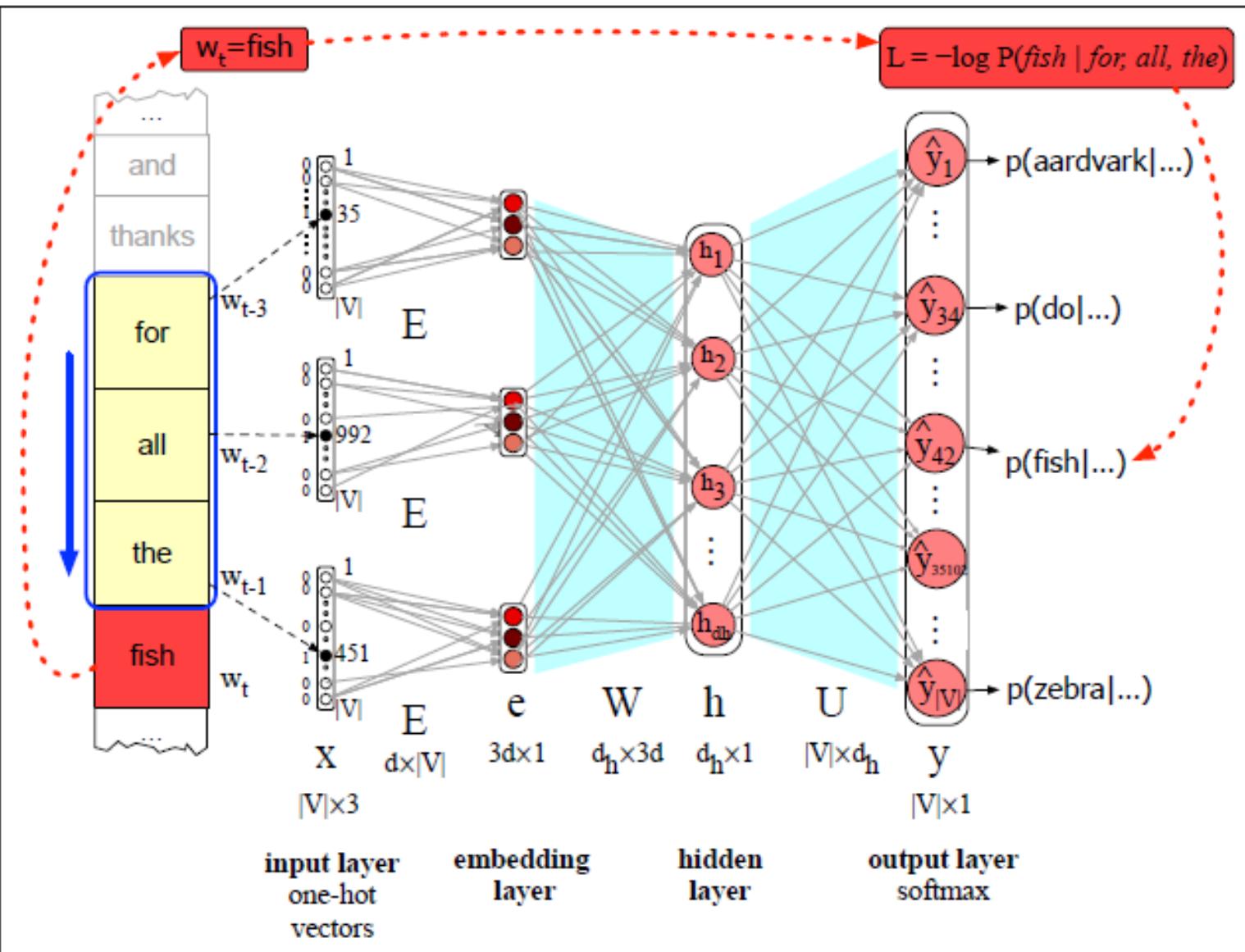
$$\begin{aligned} e &= [E_{x_{t-3}}; E_{x_{t-2}}; E_{x_{t-1}}] \\ h &= \sigma(We + b) \\ z &= Uh \\ \hat{y} &= \text{softmax}(z) \end{aligned}$$

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

Training the neural language model

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

Training the neural language model



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

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

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

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

Training the neural language model

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

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

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

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

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



References

CH-7 - Speech and Language Processing by Daniel Jurafsky



BITS Pilani
Pilani Campus

Natural Language Processing

DSECL ZG565

Prof.Vijayalakshmi
BITS-Pilani



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

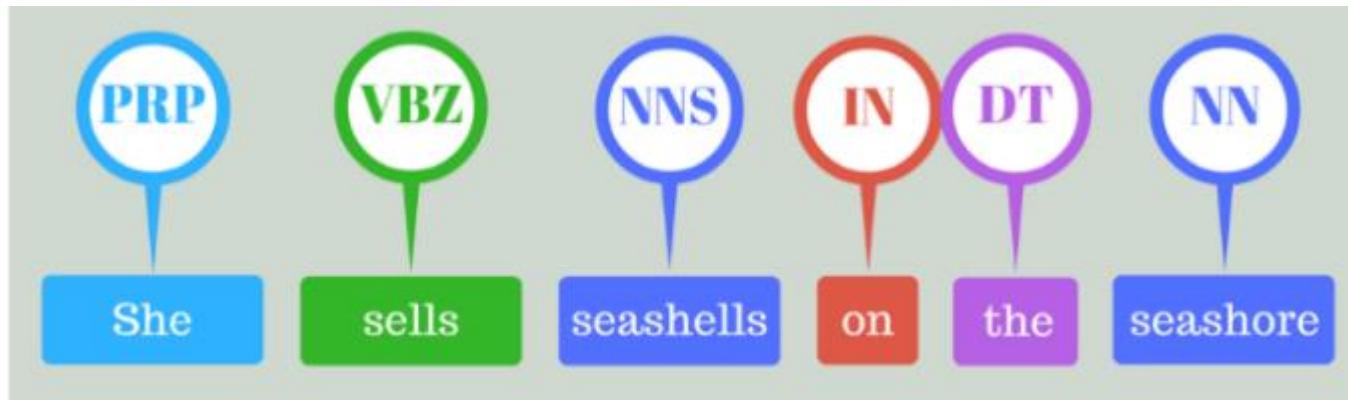
These slides are prepared by the instructor, with grateful acknowledgement of James Allen and many others who made their course materials freely available online.

Session3:POS tagging

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

What is POS Tagging

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



Process

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

- ❖ Eg:

"People jump high".

People : Noun/Verb

jump : Noun/Verb

high : Noun/Verb/Adjective

- ❖ Choose best suitable tag sequence

start with probabilities.

Why POS?

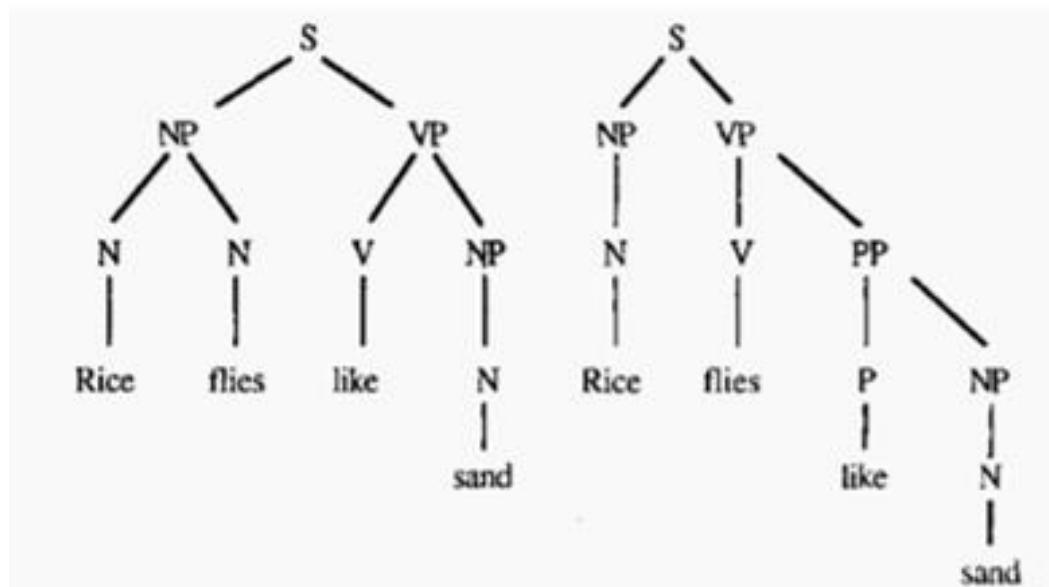
- ❖ First step in many applications
 - ❖ POS tell us a lot about word
 - ❖ Pronunciation depends on POS
 - ❖ To find named entities
 - ❖ Stemming
-

Application of POS

❖ Parsing

Recovering syntactic structures requires correct
POS tags

Eg:



Information retrieval

❖ Word Disambiguation

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

Eg:

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

Question answering system

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

Why POS tagging is hard?

❖ Ambiguity

Eg:

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

Parts of Speech

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

POS examples

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

POS Tagging

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

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

Open and Closed Classes

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

Closed Class Words

Examples:

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

Tagsets

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

Penn tree bank

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

Penn TreeBank POS Tagset



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

Just for Fun...

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

Just for Fun...

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

POS Tagging-

Choosing a Tag set

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

Approaches to POS Tagging

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

Rule based POS tagging

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

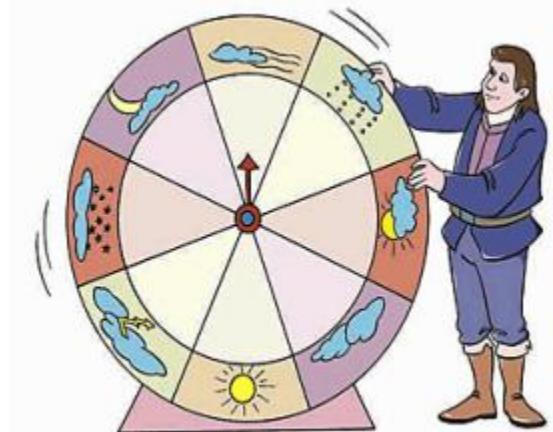
Hidden Markov model

Markov model /Markov chain

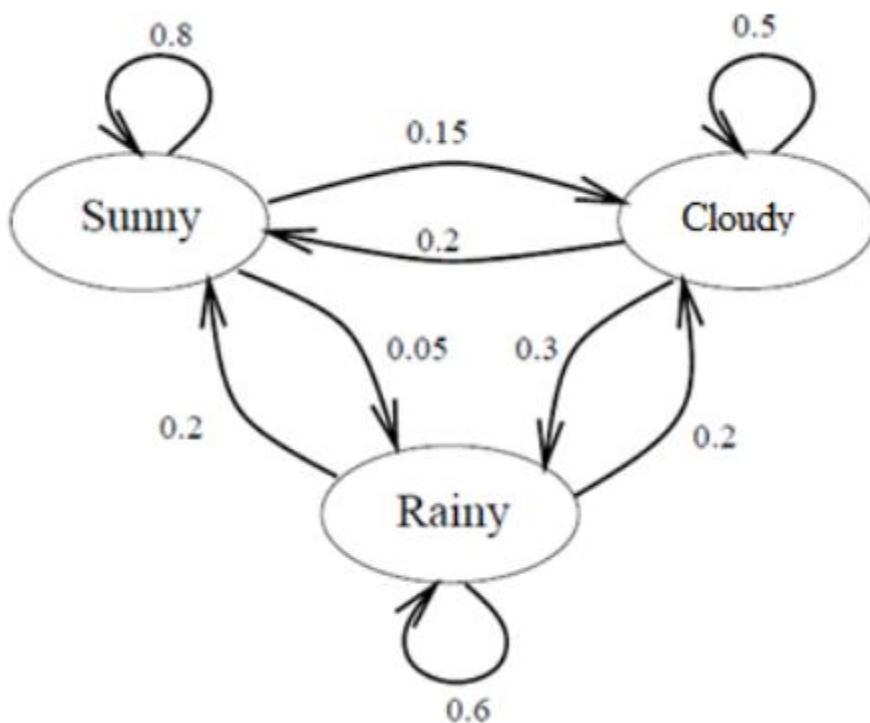
A Markov process is a process that generates a sequence of outcomes in such a way that the probability of the next outcome depends only on the current outcome and not on what happened earlier.

MARKOV CHAIN: WEATHER EXAMPLE

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



Contd..



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

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

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

Contd..

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

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

Example

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

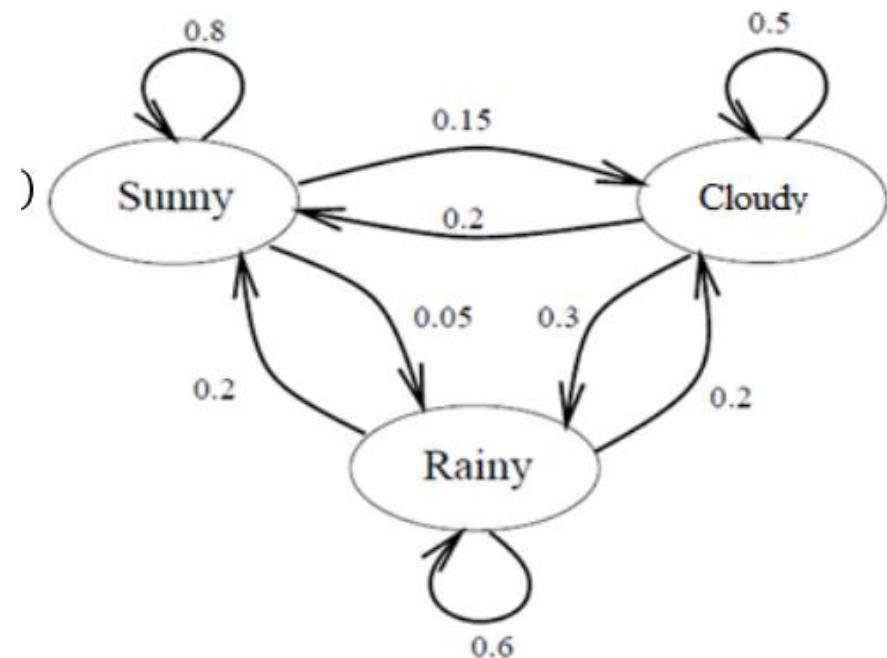
$$P(q_2, q_3 | q_1) = P(q_2 | q_1)P(q_3 | q_1, q_2)$$

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

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

$$= (0.8)(0.05)$$

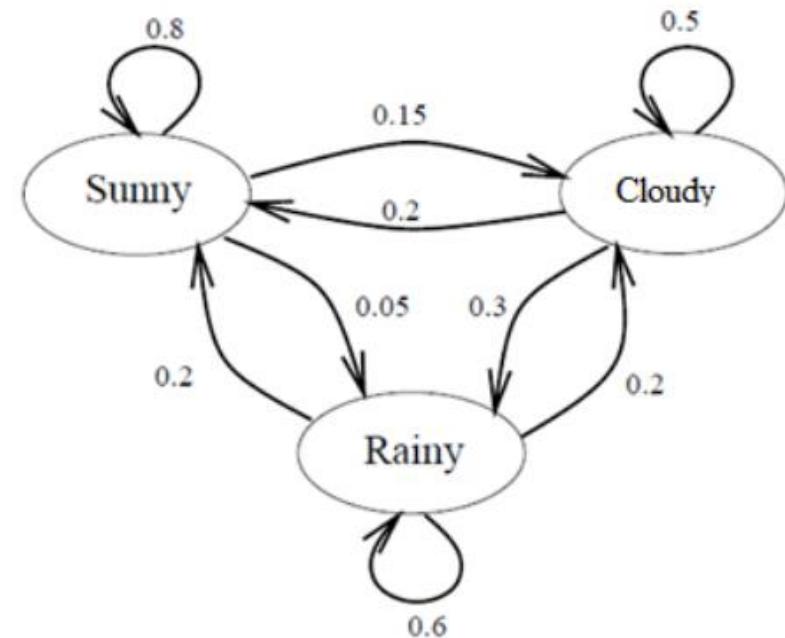
$$= 0.04$$



Example2

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

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



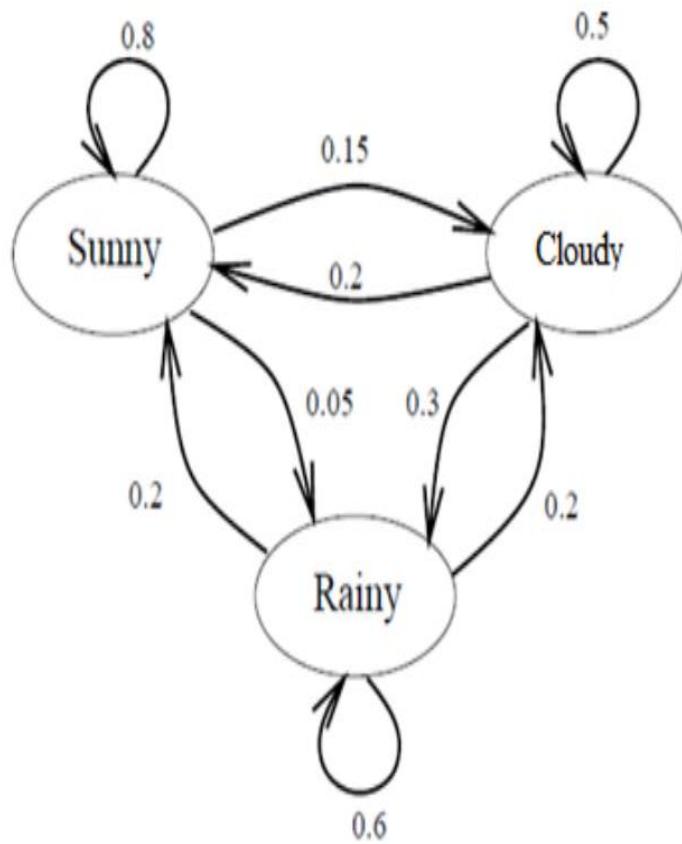
WHAT IS A HIDDEN MARKOV MODEL (HMM)?



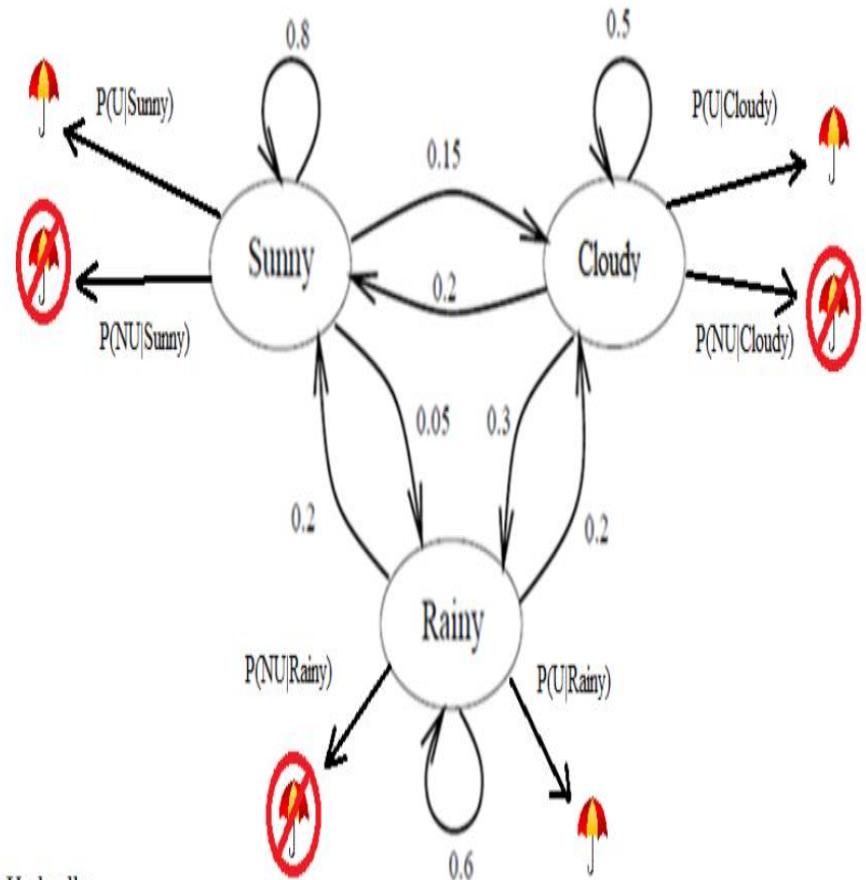
- ❖ A Hidden Markov Model, is a stochastic model where the states of the model are hidden. Each state can emit an output which is observed.
 - ❖ Imagine: You were locked in a room for several days and you were asked about the weather outside. The only piece of evidence you have is whether the person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

Markov chain Vs HMM

Markov chain



HMM



Hidden Markov Models (Formal)

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

First-Order HMM Assumptions

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

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

How to build a second-order HMM?

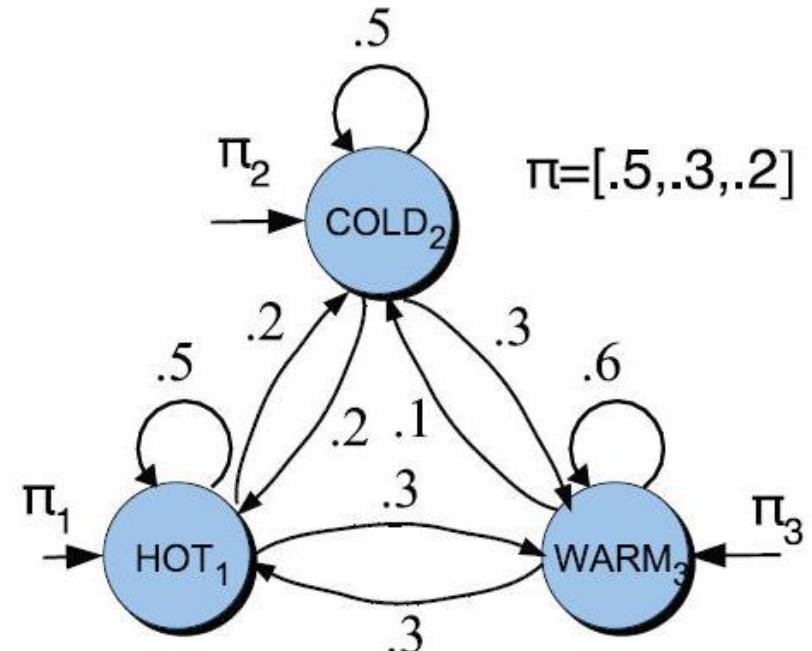
- Second-order HMM
 - Current state only depends on previous 2 states
- Example
 - Trigram model over POS tags
 - $P(\mathbf{t}) = \prod_{i=1}^n P(t_i | t_{i-1}, t_{i-2})$
 - $P(\mathbf{w}, \mathbf{t}) = \prod_{i=1}^n P(t_i | t_{i-1}, t_{i-2})P(w_i | t_i)$

Markov Chain for Weather

- What is the probability of 4 consecutive warm days?

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

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



POS Tagging as Sequence Classification



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

Probabilistic Sequence Models

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

How you predict the tags?

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

Statistical POS Tagging

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

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

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

- Hat ^ means “our estimate of the best one”
- Argmax_x f(x) means “the x such that f(x) is maximized”

- ❖ This equation should give us the best tag sequence

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

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

Using Bayes Rule

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

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

Bayes Rule:

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

Rewriting:

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

Contd..

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

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

Independent assumptions

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

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

- ❖ For Tags – use bigram probabilities

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

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

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

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

POS formula

- Find the sequence of tags that maximizes:

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

POS Tagging using HMM

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

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

- State-to-state transition probabilities

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

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

- Probabilities of observing various values at a given state

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

Two Kinds of Probabilities

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

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

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

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

Two Kinds of Probabilities

2. Word likelihood/emission probabilities

$p(w_i|t_i)$

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

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

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

Sample Probabilities



Bigram Probabilities

Bigram(T_i, T_j)	Count($i, i + 1$)	Prob($T_j T_i$)
\emptyset, ART	213	.71 (213/300)
\emptyset, N	87	.29 (87/300)
\emptyset, 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
300	633	1102	358	366

Sample Lexical Generation Probabilities

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

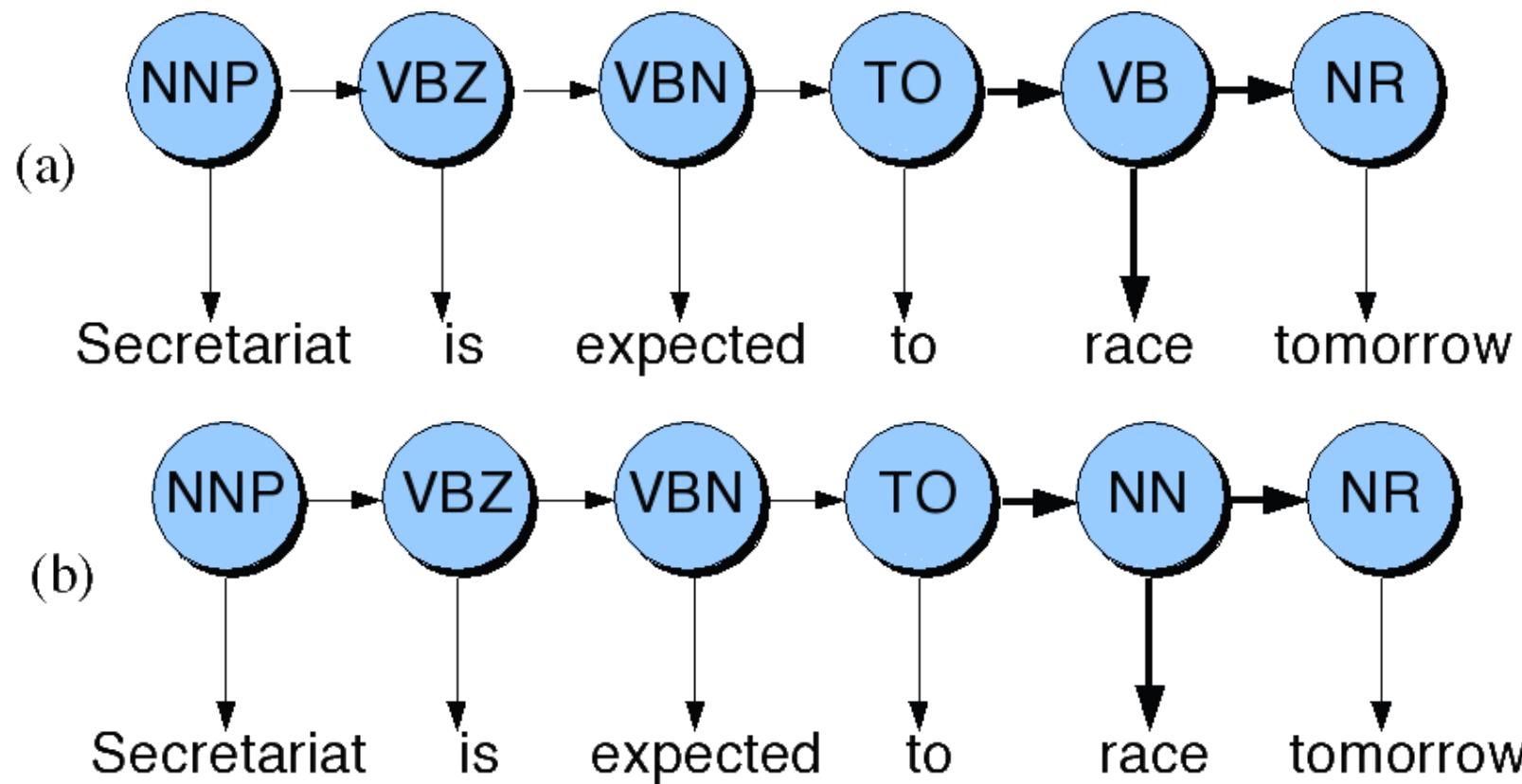
Two types of stochastic POS tagging

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

Example1-Some Data on race

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

Disambiguating **to race tomorrow**



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

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

Example2:Statistical POS tagging- Whole tag sequence

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

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

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

Tag Transition Probability

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

Markov assumption

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

Word likelihood /Emission Probability

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

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

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

Put them together

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

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

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

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

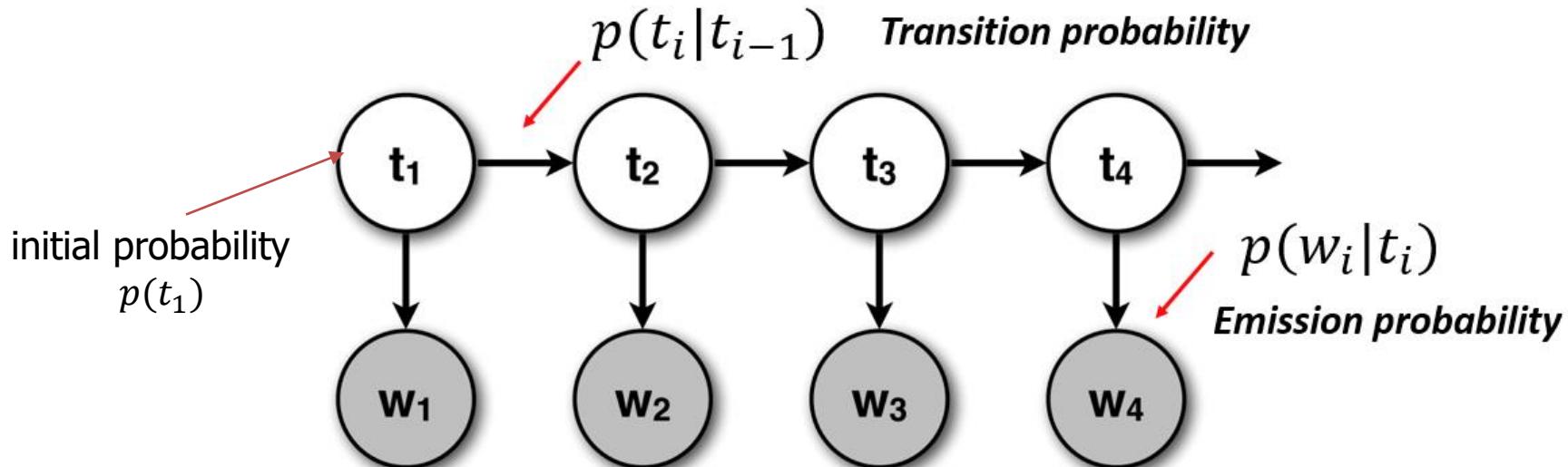
e.g., $P(\text{a smart dog , DD JJ NN })$

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

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

Put them together

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



Example1-HMM

Will can spot Mary

M V N N

Will Can spot Mary

N M V N

Emission probabilities

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

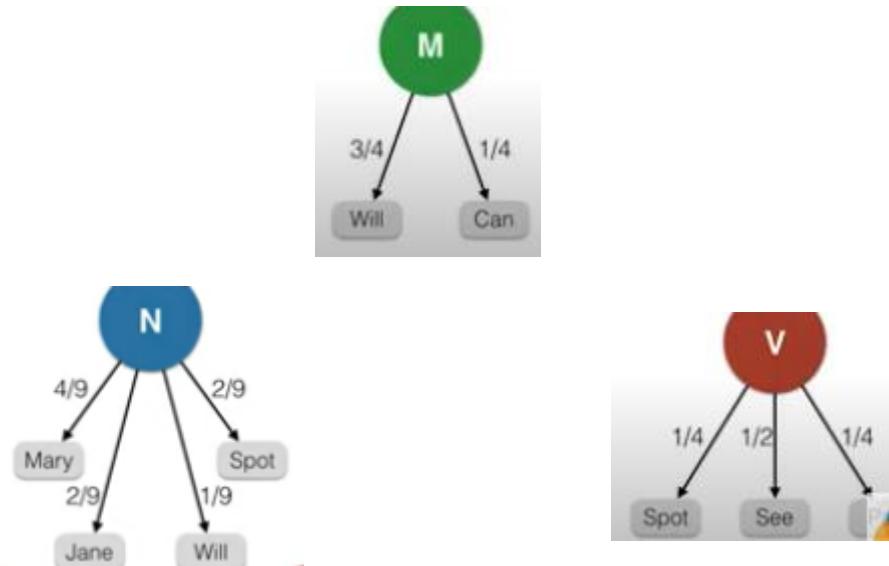
Mary Jane can see Will.

Spot will see Mary.

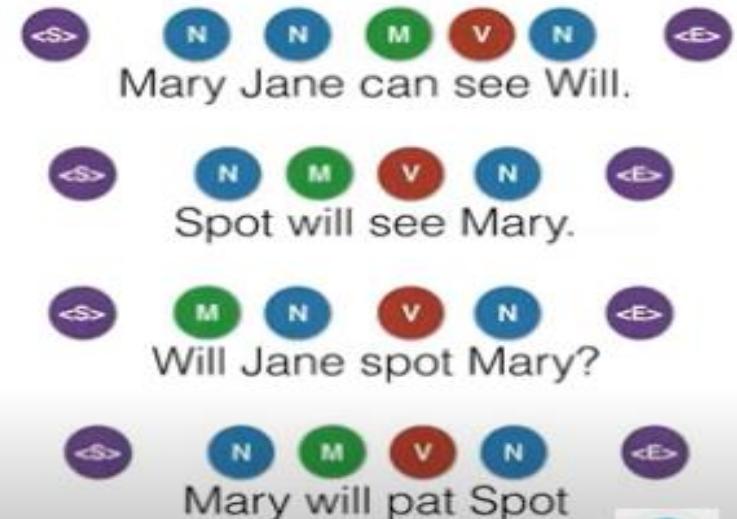
Will Jane spot Mary?

Mary will pat Spot

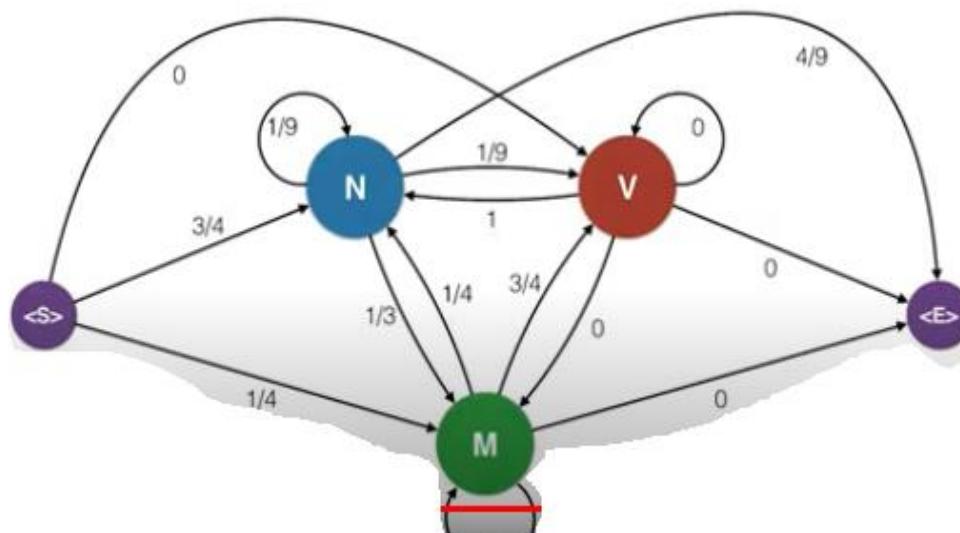
	N	M	V
Mary	4/9	0	0
Jane	2/9	0	0
Will	1/9	3/4	0
Spot	2/9	0	1/4
Can	0	1/4	0
See	0	0	1/2
Pat	0	0	1/4



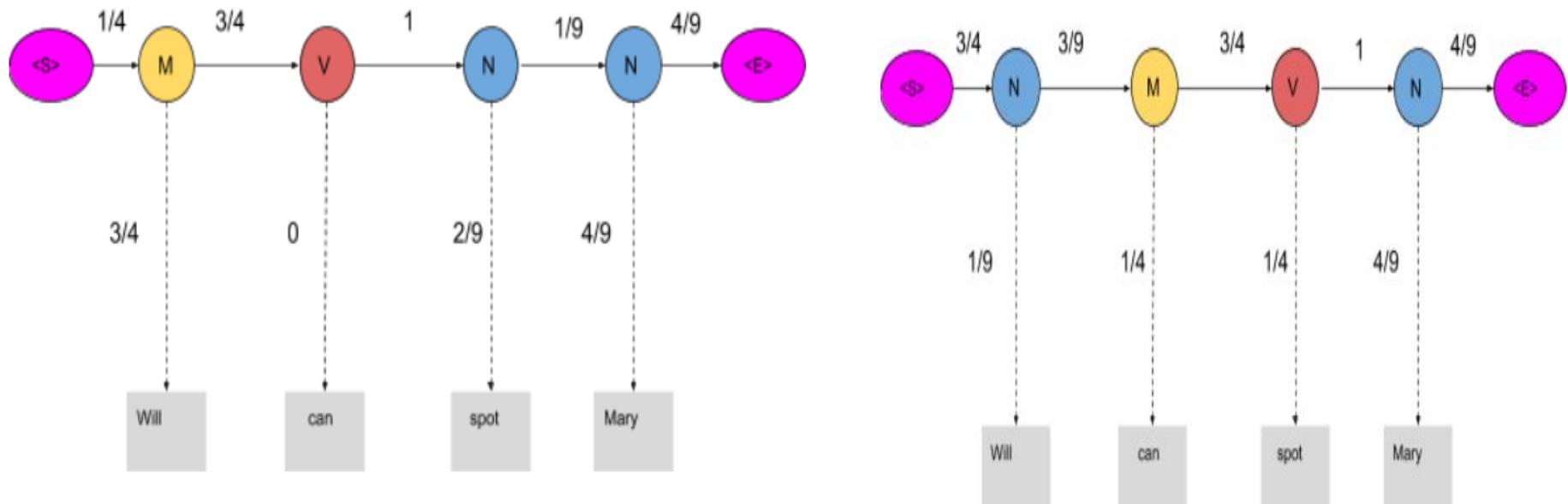
Transition probabilities



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



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



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

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

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

Thank You... 😊

- Q&A
- Suggestions / Feedback



Natural Language Processing

DSECL ZG565



BITS Pilani
Pilani Campus

Prof.Vijayalakshmi Anand

BITS-Pilani



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

Date – 5th Dec 2021

Time – 9 am to 11am

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



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

Session4-POS tagging(Viterbi ,Maximum entropy model)

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

Hidden Markov Models

It is a **sequence model**.

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

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

This is a kind of *generative* model.

Hidden Markov Model (HMM)

Oftentimes we want to know what produced the sequence

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

For example,

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

Definition of HMM

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

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

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

Transition probabilities

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

Observation likelihoods

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

Special initial probability vector π

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

Three Problems

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

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

Problem 1:

Observation Likelihood

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

Problem 2:



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

Problem 3:

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

Solutions

Problem 1: Forward (learn observation sequence)

Problem 2: Viterbi (learn state sequence)

Problem 3: Forward-Backward (learn probabilities)

- An instance of EM (Expectation Maximization)

Example :HMMs for Ice Cream

You are a climatologist in the year 2799 studying global warming

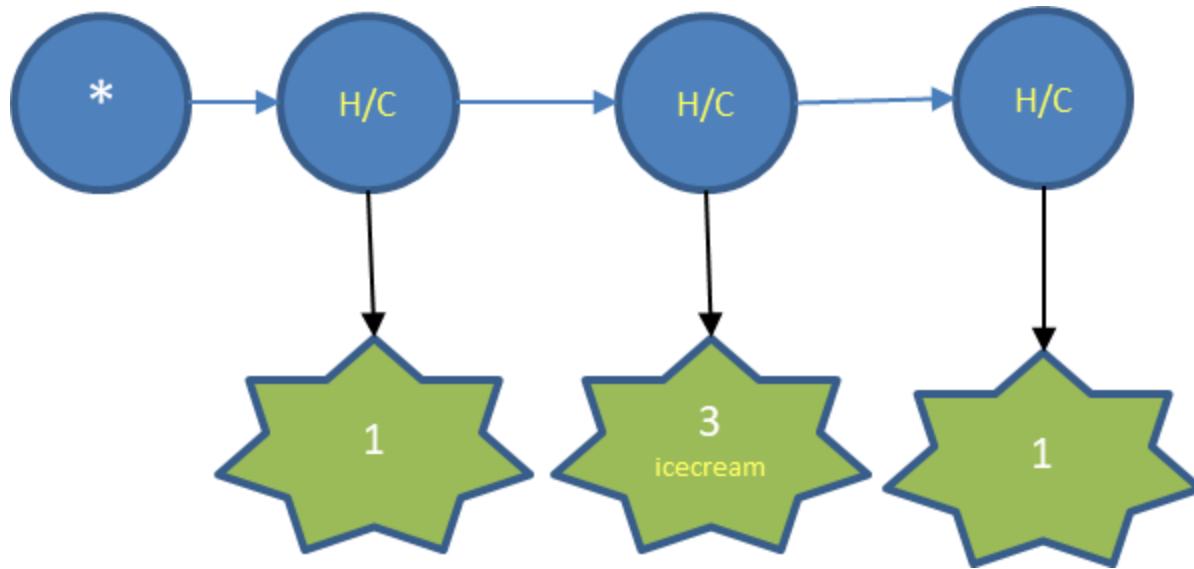
You can't find any records of the weather in Baltimore for summer of 2007

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



Your job: figure out how hot it was each day

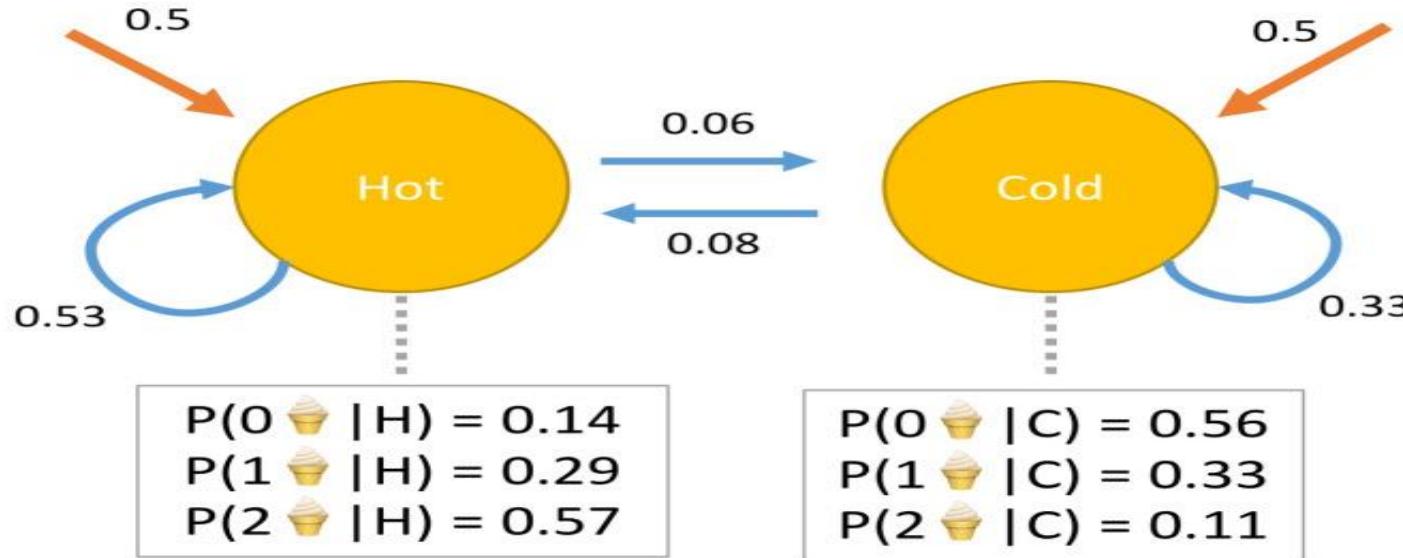
Problem 1



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

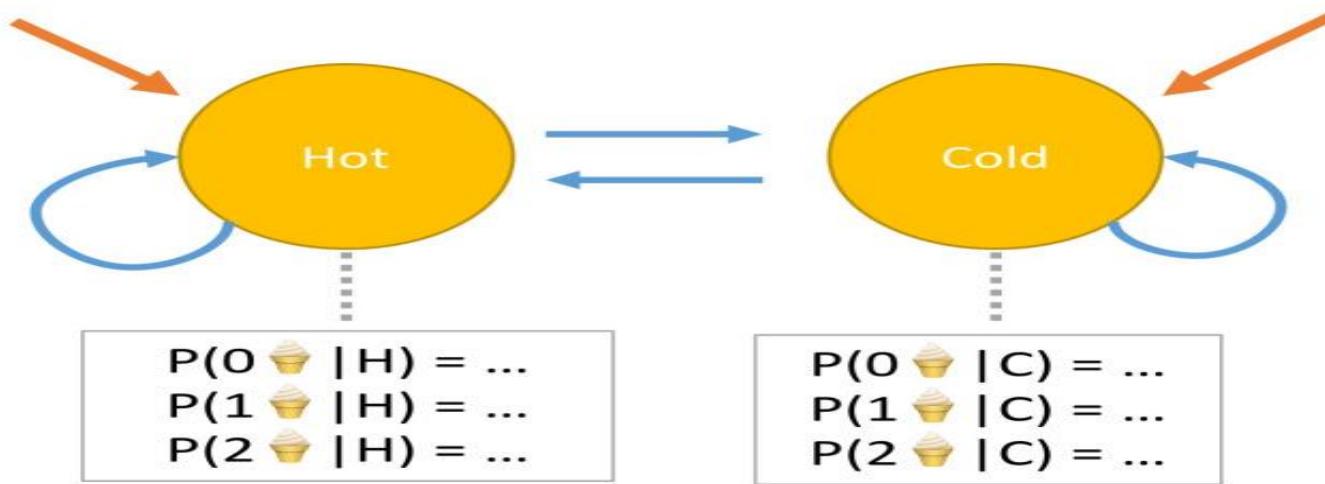
Problem2

1 2 3



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

Problem3



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

Getting to HMMs

This equation should give us the best tag sequence

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

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

Intuition of Bayesian inference:

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

Using Bayes Rule

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

Know this.

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

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

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

Likelihood and Prior

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

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

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

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

HMM for Ice Cream

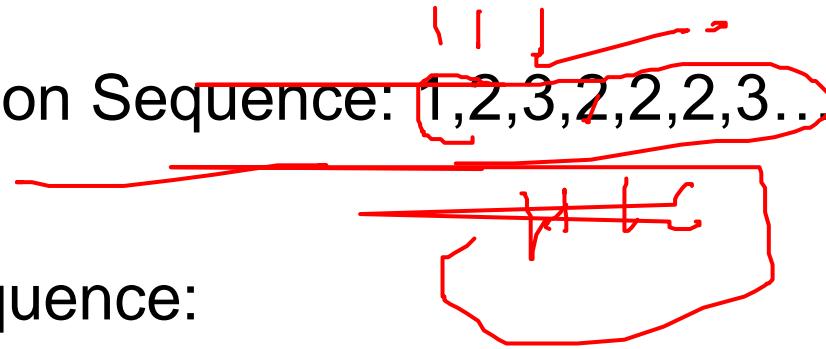
Given

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

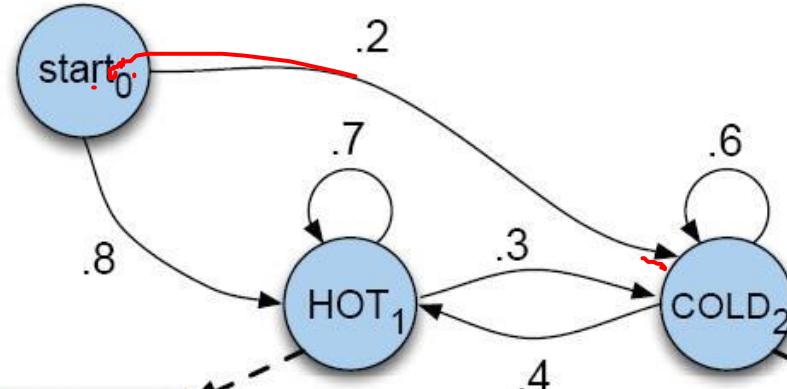
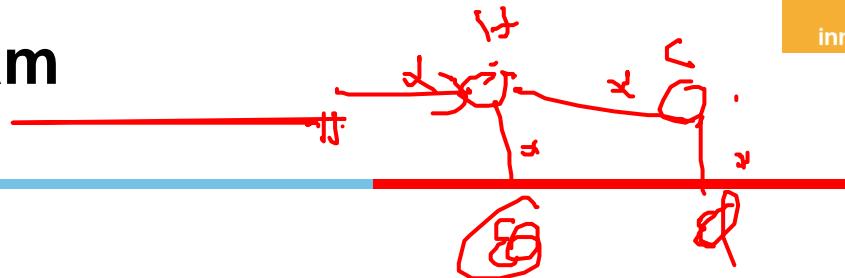
Produce:

- Hidden Weather Sequence:

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



HMM for Ice Cream



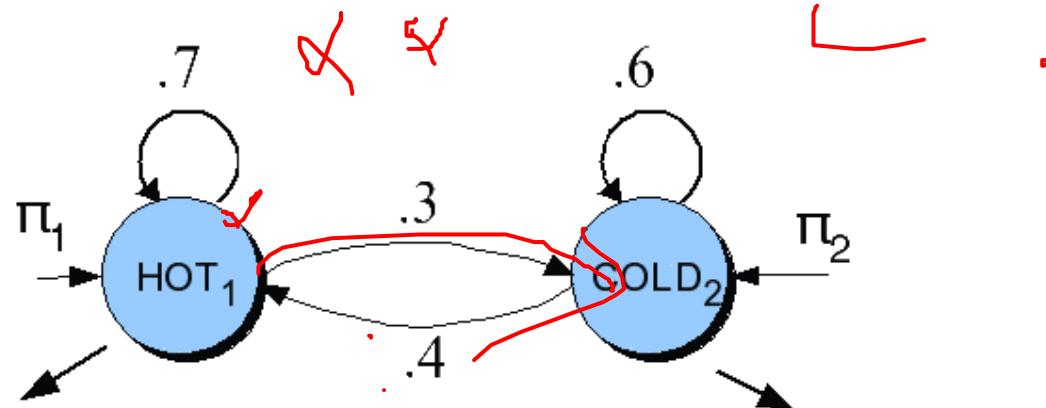
$$B_1 \quad \begin{bmatrix} P(1 | HOT) \\ P(2 | HOT) \\ P(3 | HOT) \end{bmatrix} = \begin{bmatrix} .2 \\ .4 \\ .4 \end{bmatrix}$$

the observed sequence
“1 2 3”

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

HMM for Ice Cream

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



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

Transition probability

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

Emission probabilities

	H	C
H	0.7	0.3
C	0.4	0.6

	1	2	3
H	0.2	0.4	0.4
C	0.5	0.4	0.1

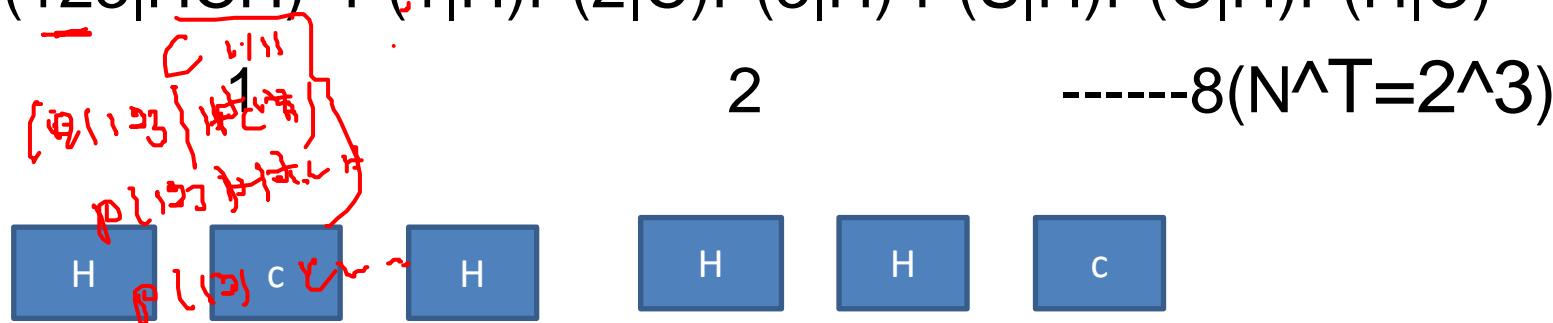
Find 123 sequence

(1 2 3)

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

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

\cdot



Decoding

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

Contd..

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

Viterbi intuition

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

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

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

probabilities

	2	3
2	0.4	0.4
3	0.4	0.1

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

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

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

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

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

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

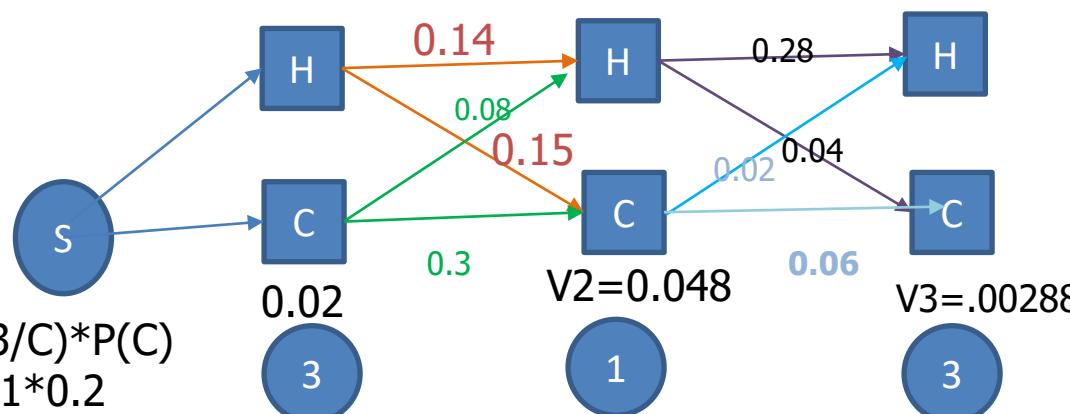
$$V3 = \text{Max} [(0.28 * 0.0448 = 0.013), (0.02 * 0.048 = 0.00096)]$$

$$V1 = 0.32$$

$$V2 = 0.0448$$

$$V3 = 0.013$$

Transition probability



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

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

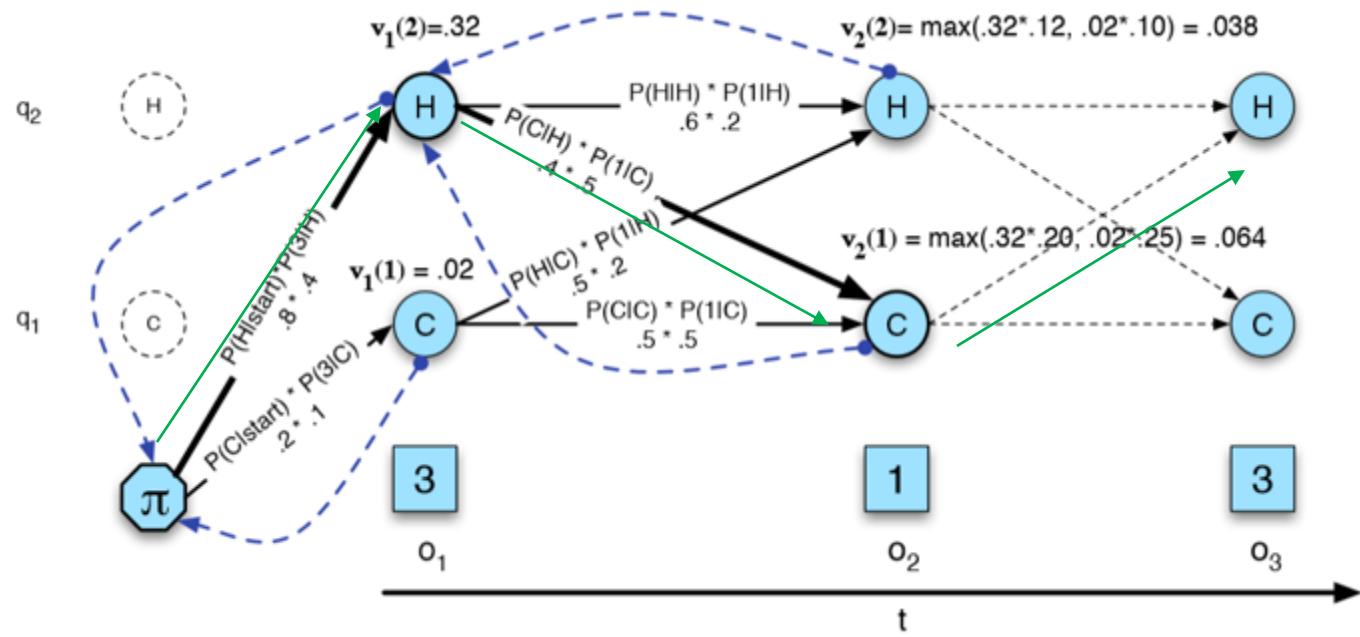
$$\begin{aligned} P(1,C) &= P(1/C) * P(C/C) \\ &= 0.5 * 0.6 \\ &= 0.30 \end{aligned}$$

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

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

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

	H	C
H	0.7	0.3
C	0.4	0.6



Example: Rainy and Sunny Days

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

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

Two hidden states: rainy and sunny

Two observables: walking and driving

Assume equal likelihood of the first day being rainy or sunny

Transitional probabilities

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

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

Output (emission) probabilities

sunny given walking = .1, driving = .9

rainy given walking = .8, driving = .2

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

State1:Walk

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

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

State2: Drive

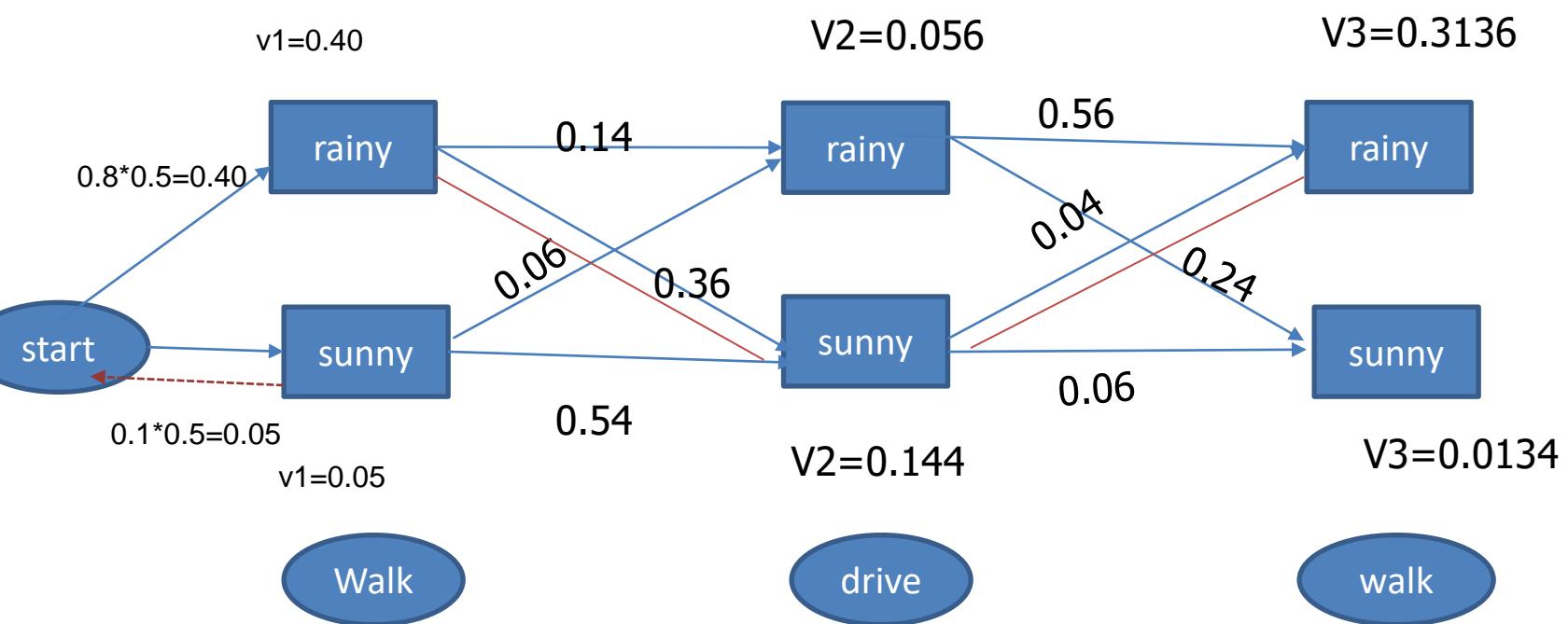
$$\begin{aligned} P(\text{rainy, drive}) &= P(\text{drive} | \text{rainy}) * P(\text{rainy}) \\ &= 0.2 * 0.7 = 0.14 \\ &= P(\text{drive} | \text{rainy}) * P(\text{rainy} | \text{sunny}) \\ &= 0.2 * 0.3 = 0.06 \end{aligned}$$

$$\begin{aligned} P(\text{sunny, drive}) &= P(\text{drive} | \text{sunny}) * P(\text{sunny}) \\ &= 0.9 * 0.6 = 0.54 \\ &= P(\text{drive} | \text{sunny}) * P(\text{sunny} | \text{rainy}) \\ &= 0.9 * 0.4 = 0.36 \end{aligned}$$

State 3:Walk

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

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



The Viterbi Algorithm

function VITERBI(*observations* of len T ,*state-graph* of len N) **returns** *best-path*

create a path probability matrix $viterbi[N+2,T]$

for each state s **from** 1 **to** N **do** ; initialization step

$viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$

$backpointer[s,1] \leftarrow 0$

for each time step t **from** 2 **to** T **do** ; recursion step

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

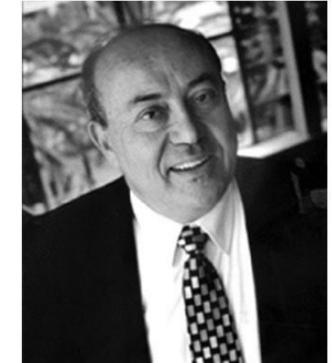
$viterbi[s,t] \leftarrow \max_{s'=1}^N viterbi[s',t-1] * a_{s',s} * b_s(o_t)$

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

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

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

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



Viterbi Example: Ice Cream

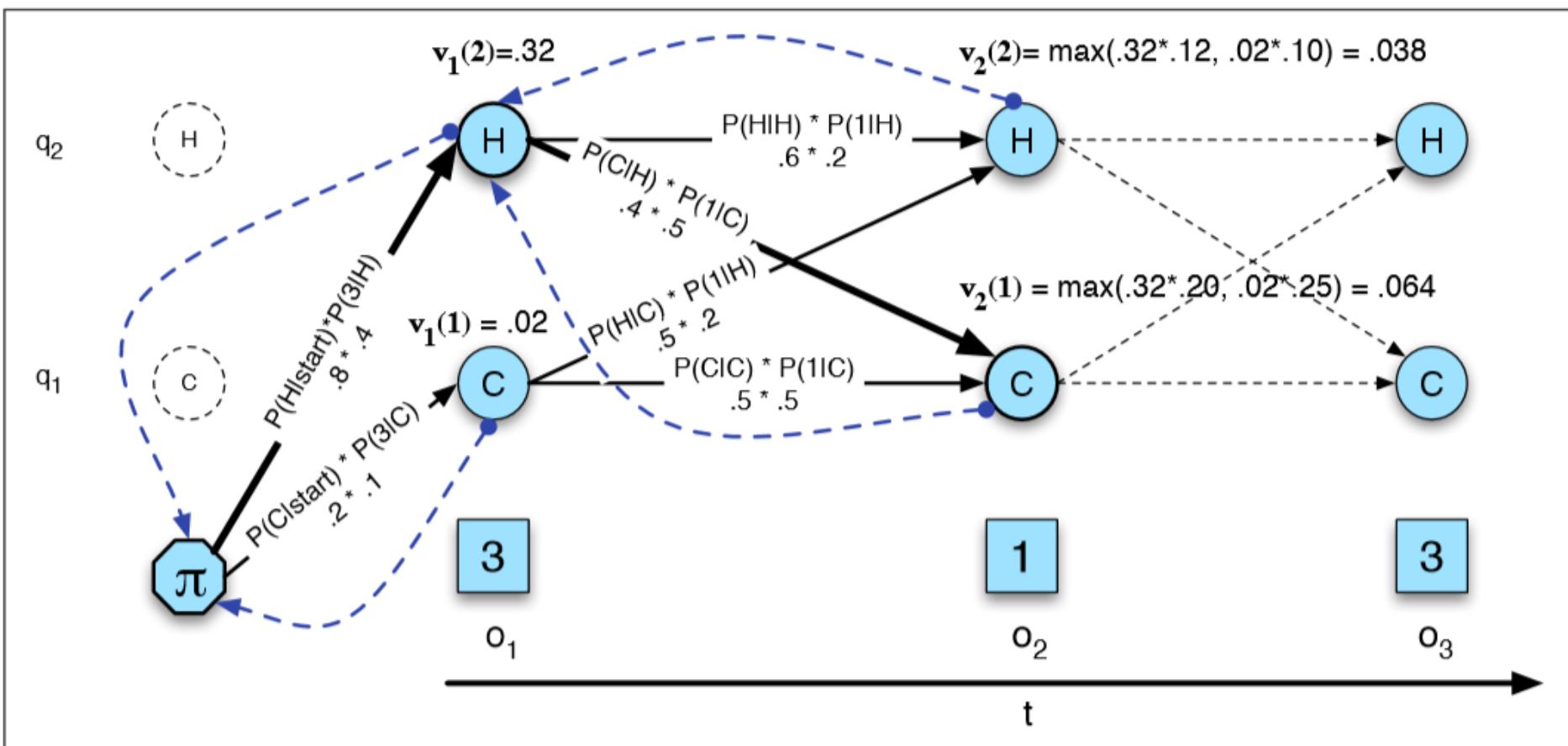


Figure A.10 The Viterbi backtrace. As we extend each path to a new state account for the next observation, we keep a backpointer (shown with broken lines) to the best path that led us to this state.

Example3

Janet will back the bill

The correct series of tags is:

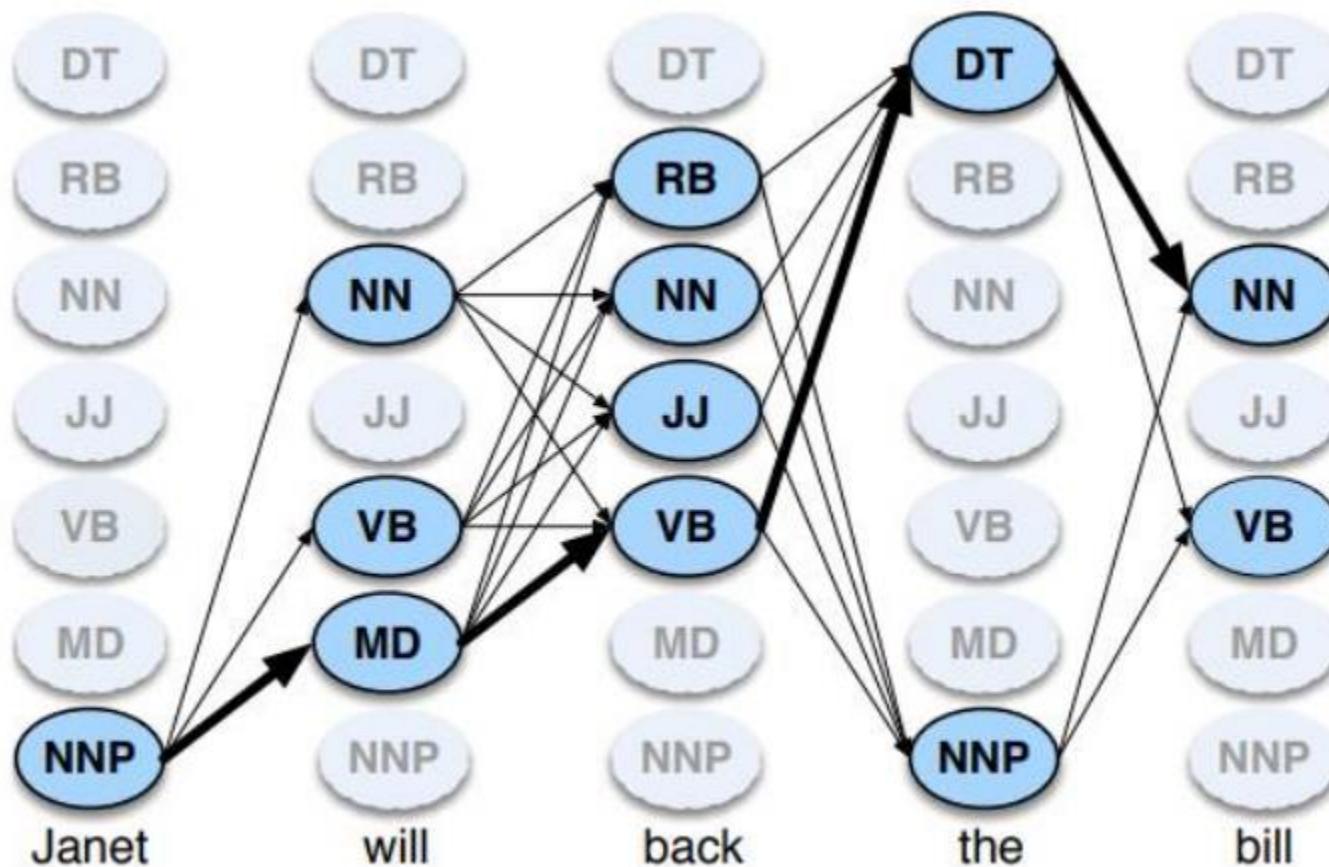
Janet/NNP will/MD back/VB the/DT bill/NN

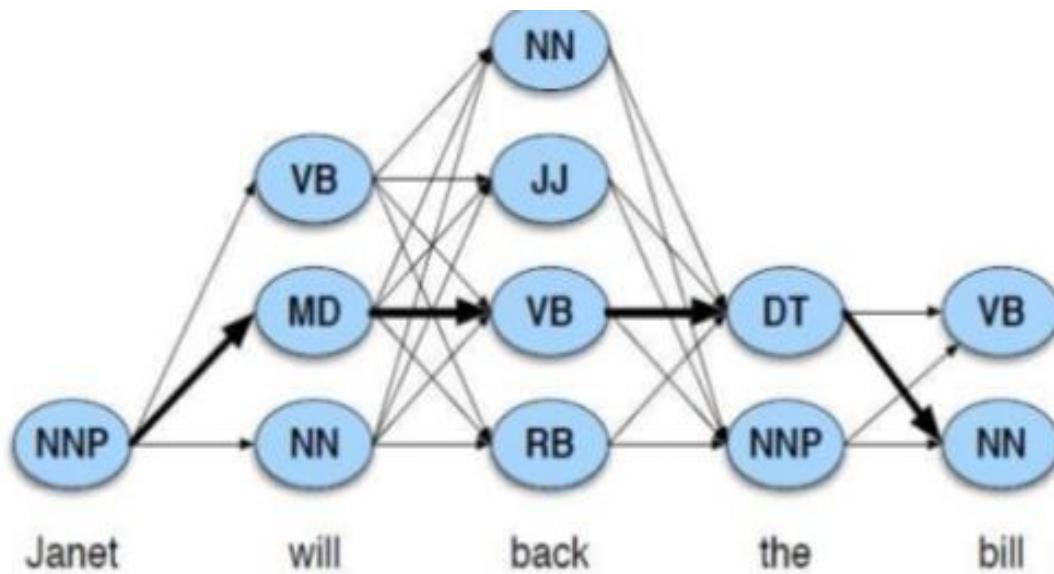
	NNP	MD	VB	JJ	NN	RB	DT
<i>< s ></i>	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096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

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

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

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





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

Practice question

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

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

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

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

Forward

Efficiently computes the probability of an observed sequence given a model

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

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

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

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

The Forward Algorithm

function FORWARD(*observations* of len T , *state-graph* of len N) **returns** *forward-prob*

create a probability matrix $forward[N+2,T]$

for each state s **from** 1 **to** N **do** ; initialization step

$forward[s,1] \leftarrow a_{0,s} * b_s(o_1)$

for each time step t **from** 2 **to** T **do** ; recursion step

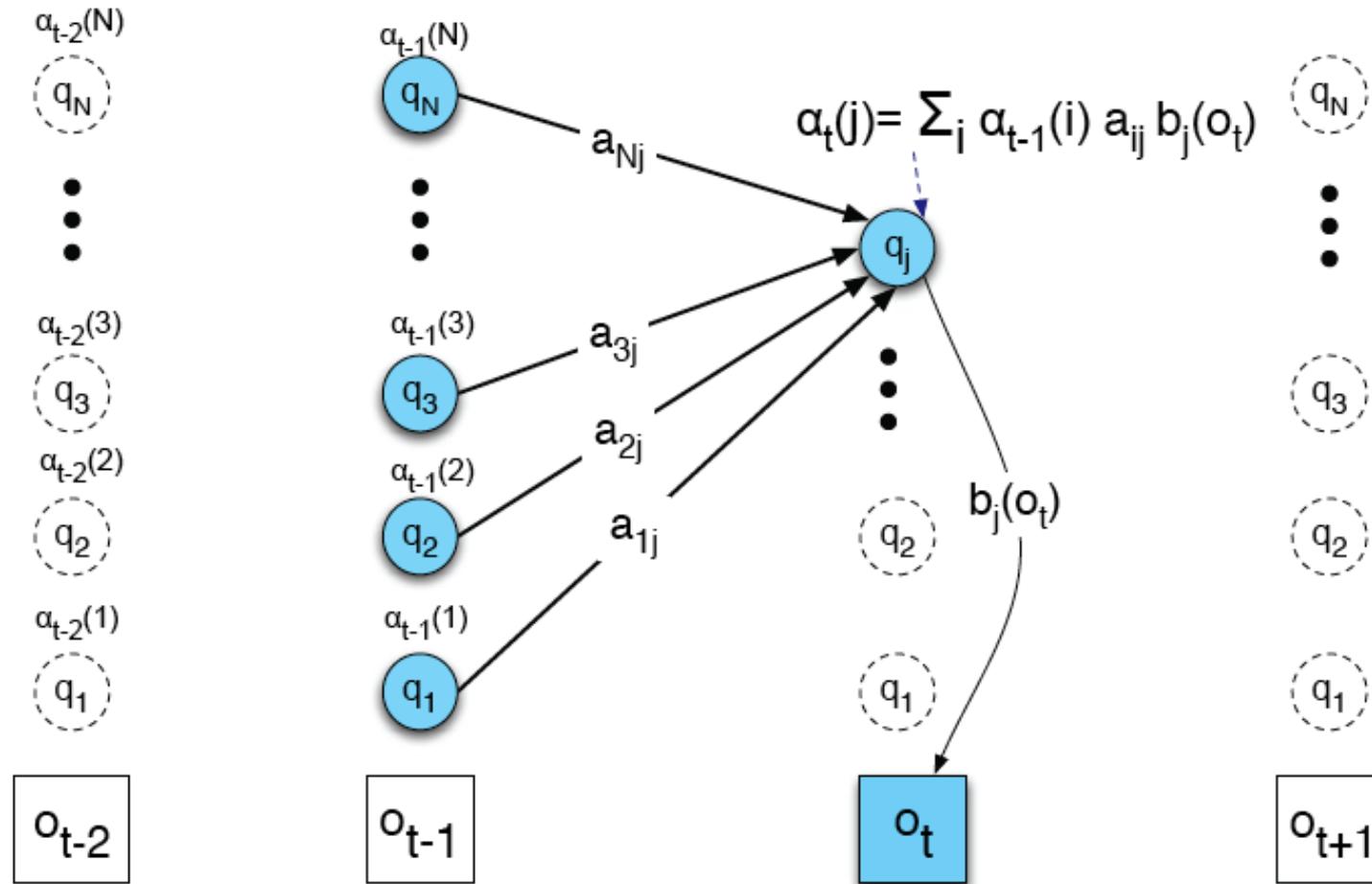
for each state s **from** 1 **to** N **do**

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

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

return $forward[q_F, T]$

Visualizing Forward



Forward Algorithm: Ice Cream

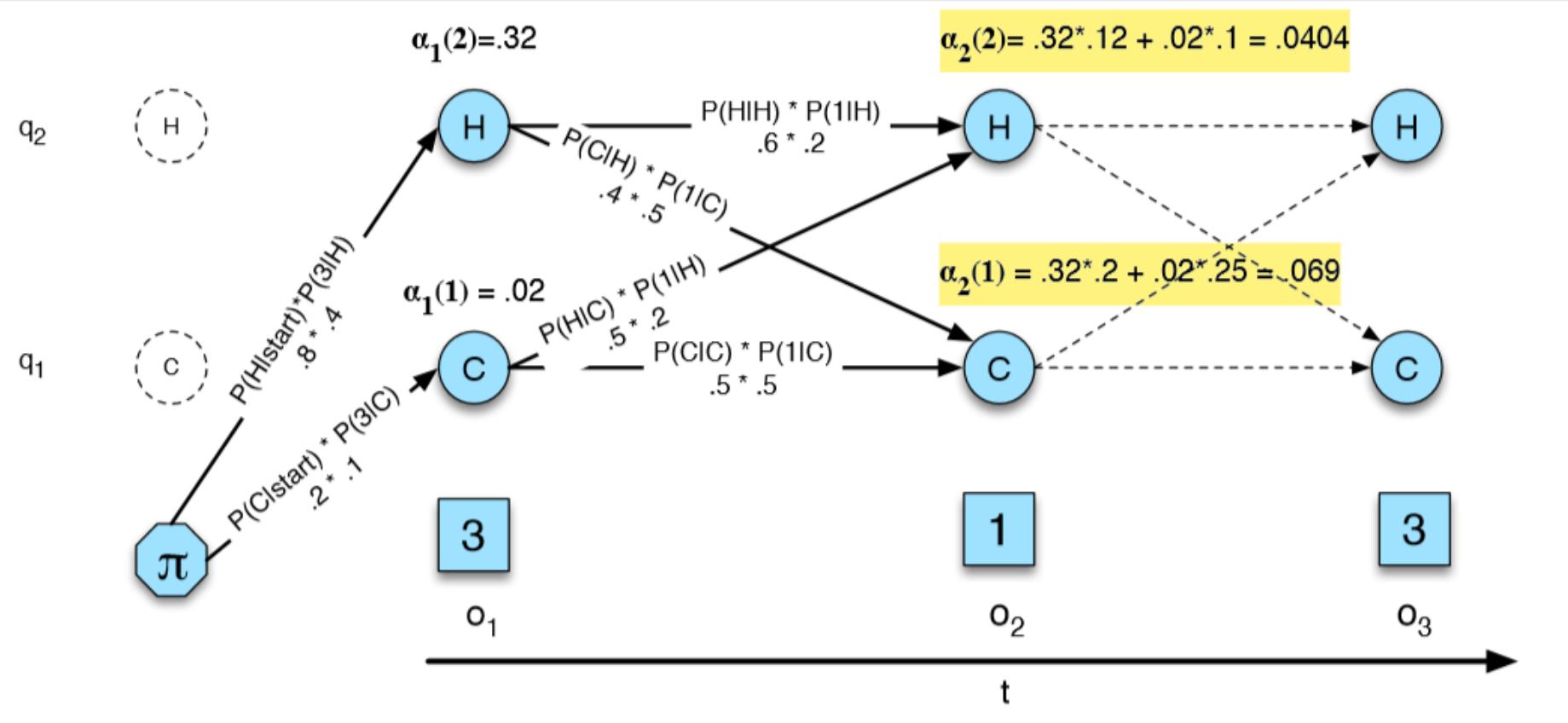


Figure A.5 The forward trellis for computing the total observation likelihood for the ice-cream events 3 1 3. Hidden states are in circles, observations in squares. The figure shows the computation of $\alpha_t(j)$ for two states at two time steps. The computation in each cell follows Eq. A.12: $\alpha_t(j) = \sum_{i=1}^N \alpha_{t-1}(i) a_{ij} b_j(o_t)$. The resulting probability expressed in each cell is Eq. A.11: $\alpha_t(j) = P(o_1, o_2 \dots o_t, q_t = j | \lambda)$.

Problem 3

Infer the best model parameters, given a skeletal model and an observation sequence...

- That is, fill in the A and B tables with the right numbers...
 - The numbers that make the observation sequence most likely
- Useful for getting an HMM without having to hire annotators...

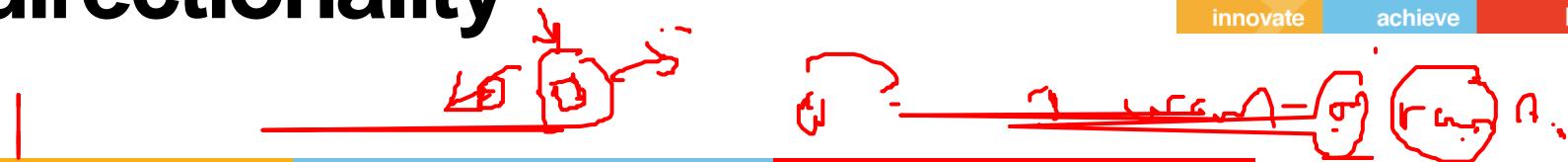
Forward-Backward

Learning: Given an observation sequence O and the set of possible states in the HMM, learn the HMM parameters A and B .

Is a special case of the EM or Expectation-Maximization algorithm

The algorithm will let us learn the transition probabilities $A = \{a_{ij}\}$ and the emission probabilities $B = \{b_i(o_t)\}$ of the HMM

Bidirectionality



- One problem with the HMM models as presented is that they are exclusively run left-to-right. $b_{1,1}$ $b_{2,2}$
- Viterbi algorithm still allows present decisions to be influenced indirectly by future decisions,
- It would help even more if a decision about word w_i could directly use information about future tags t_{i+1} and t_{i+2} .

Bidirectionality



- Any sequence model can be turned into a bidirectional model by using multiple passes.
- For example, the first pass would use only part-of-speech features from already-disambiguated words on the left. In the second pass, tags for all words, including those on the right, can be used.
- Alternately, the tagger can be run twice, once left-to-right and once right-to-left.
- In Viterbi decoding, the classifier chooses the higher scoring of the two sequences (left-to-right or right-to-left).
- Modern taggers are generally run bidirectionally.

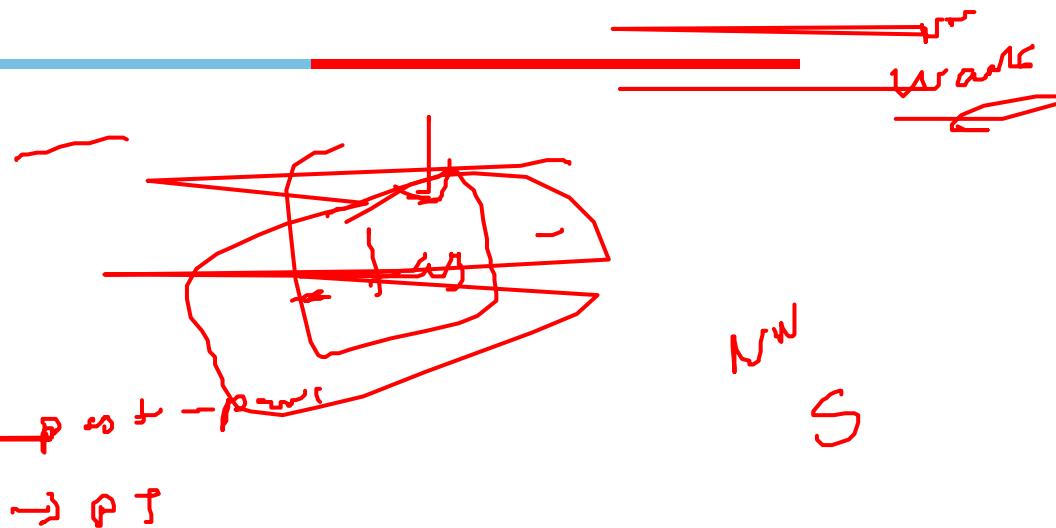
Some limitation of HMMS

- Unknown words
- First order HMM

Eg:

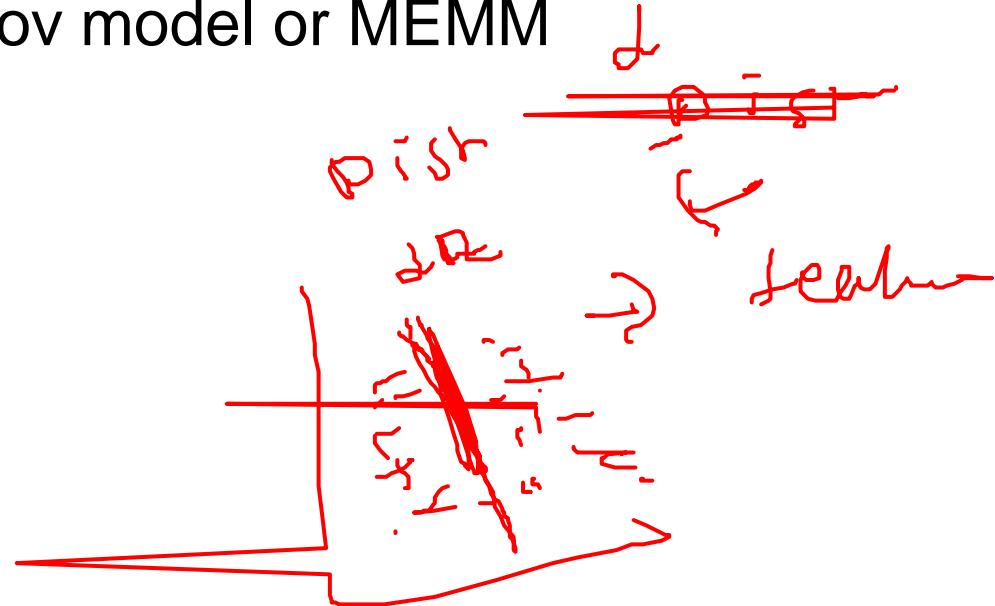
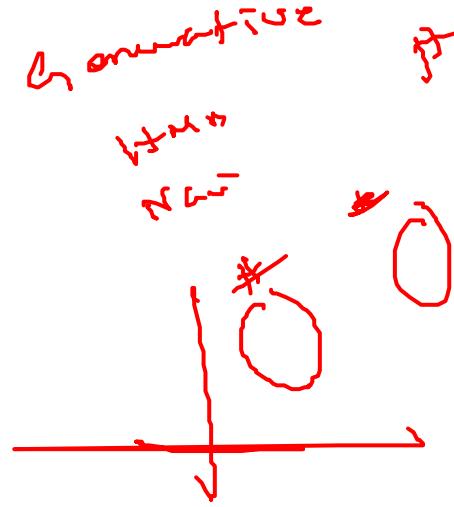
Is clearly marked

He clearly marked



Maximum Entropy Markov Model

- Turn logistic regression into a discriminative sequence model simply by running it on successive words, using the class assigned to the prior word as a feature in the classification of the next word.
- When we apply logistic regression in this way, it's called maximum entropy Markov model or MEMM



Maximum Entropy Markov Model



- Let the sequence of words be $W = w^n_1$ and the sequence of tags $T = t^n_1$.
- In an HMM to compute the best tag sequence that maximizes $P(T|W)$ we rely on Bayes' rule and the likelihood $P(W|T)$:

$$\hat{T} = \operatorname{argmax}_T P(T|W)$$

$$= \operatorname{argmax}_T P(W|T)P(T)$$

$$= \operatorname{argmax}_T \prod_i P(\text{word}_i | \underline{\text{tag}}_i) \prod_i P(\underline{\text{tag}}_i | \underline{\text{tag}}_{i-1})$$

$P((\underline{B})) =$
 ~~$P(B_1, B_2, B_3, B_4)$~~

$\rightarrow P(B_1)$

Maximum Entropy Markov Model

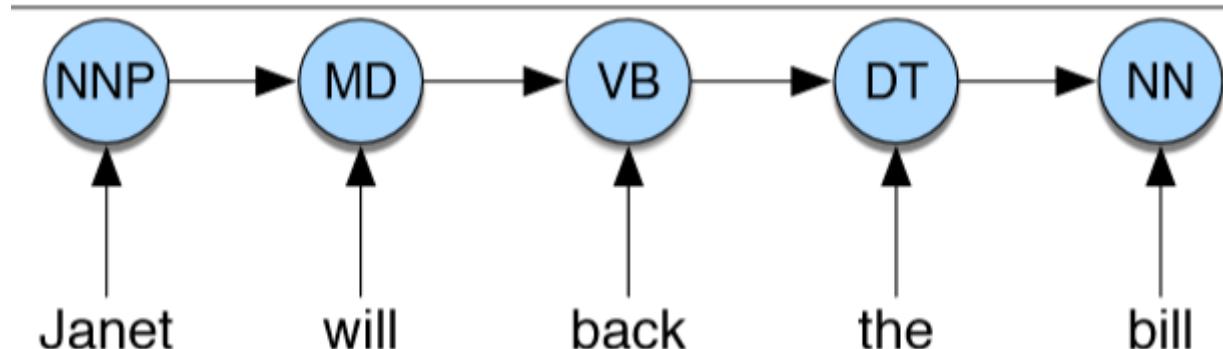
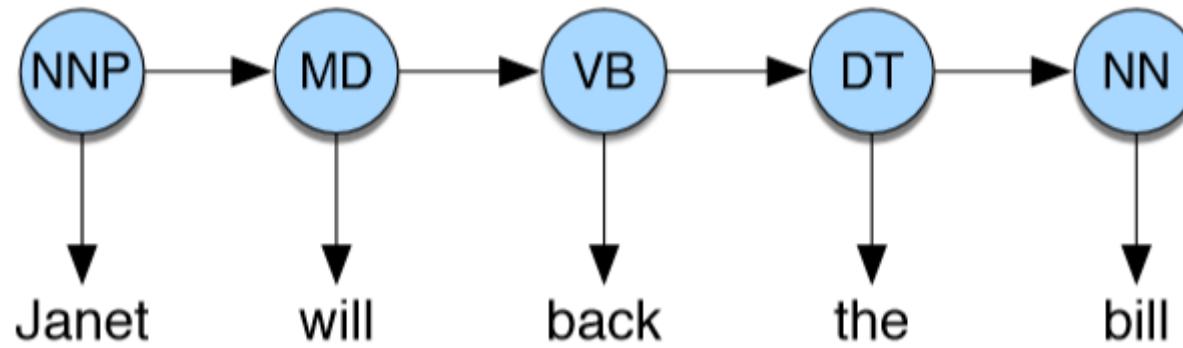


- In an MEMM, by contrast, we compute the posterior $P(T|W)$ directly, training it to discriminate among the possible tag sequences:

$$\begin{aligned}\hat{T} &= \operatorname{argmax}_T P(T|W) \\ &= \operatorname{argmax}_T \prod_i P(t_i|w_i, t_{i-1})\end{aligned}$$
A red circle highlights the term T in the first equation, and a red oval encloses the entire second equation, highlighting the conditional probability expression.

Maximum Entropy Markov Model

- Consider tagging just one word. A multinomial logistic regression classifier could compute the single probability $P(t_i|w_i, t_{i-1})$ in a different way than an HMM
- HMMs compute likelihood (observation word conditioned on tags) but MEMMs compute posterior (tags conditioned on observation words).



Learning MEMM

- Learning in MEMMs relies on the same supervised learning algorithms we presented for logistic regression.
- Given a sequence of observations, feature functions, and corresponding hidden states, we use gradient descent to train the weights to maximize the log-likelihood of the training corpus.

Maximum Entropy Markov Model

- Reason to use a discriminative sequence model is that it's easier to incorporate a lot of features

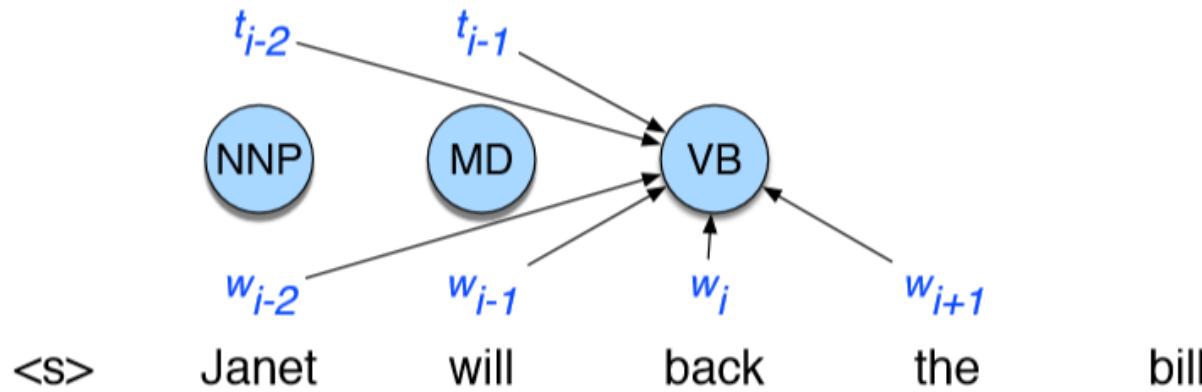


Figure 8.13 An MEMM for part-of-speech tagging showing the ability to condition on more features.

MEMM

- Janet/NNP will/MD back/VB the/DT bill/NN, when w_i is the word back, would generate the following features

$t_i = \text{VB}$ and $w_{i-2} = \text{Janet}$

$t_i = \text{VB}$ and $w_{i-1} = \text{will}$

$t_i = \text{VB}$ and $w_i = \text{back}$

$t_i = \text{VB}$ and $w_{i+1} = \text{the}$

$t_i = \text{VB}$ and $w_{i+2} = \text{bill}$

$t_i = \text{VB}$ and $t_{i-1} = \text{MD}$

$t_i = \text{VB}$ and $t_{i-1} = \text{MD}$ and $t_{i-2} = \text{NNP}$

$t_i = \text{VB}$ and $w_i = \text{back}$ and $w_{i+1} = \text{the}$

Training MEMMs

- The most likely sequence of tags is then computed by combining these features of the input word w_i , its neighbors within l words w_{i-l}^{i+l} , and the previous k tags t_{i-k}^{i-1} as follows (using θ to refer to feature weights instead of w to avoid the confusion with w meaning words):

$$\begin{aligned}
 \hat{T} &= \operatorname{argmax}_T P(T|W) \\
 &= \operatorname{argmax}_T \prod_i P(t_i | w_{i-l}^{i+l}, t_{i-k}^{i-1}) \\
 &= \operatorname{argmax}_T \prod_i \frac{\exp \left(\sum_j \theta_j f_j(t_i, w_{i-l}^{i+l}, t_{i-k}^{i-1}) \right)}{\sum_{t' \in \text{tagset}} \exp \left(\sum_j \theta_j f_j(t', w_{i-l}^{i+l}, t_{i-k}^{i-1}) \right)}
 \end{aligned}$$

How to decode to find this optimal tag sequence \hat{T} ?

- Simplest way to turn logistic regression into a sequence model is to build a local classifier that classifies each word left to right, making a hard classification on the first word in the sentence, then a hard decision on the second word, and so on.
- This is called a greedy decoding algorithm

```

function GREEDY SEQUENCE DECODING(words W, model P) returns tag sequence T
    for  $i = 1$  to  $\text{length}(W)$ 
         $\hat{t}_i = \underset{t' \in T}{\text{argmax}} P(t' | w_{i-l}^{i+l}, t_{i-k}^{i-1})$ 

```

Figure 8.14 In greedy decoding we simply run the classifier on each token, left to right, each time making a hard decision about which is the best tag.

Issue with greedy algorithm

- The problem with the greedy algorithm is that by making a hard decision on each word before moving on to the next word, the classifier can't use evidence from future decisions.
- Although the greedy algorithm is very fast, and occasionally has sufficient accuracy to be useful, in general the hard decision causes too great a drop in performance, and we don't use it.
- MEMM with the Viterbi algorithm just as with the HMM, Viterbi finding the sequence of part-of-speech tags that is optimal for the whole sentence

MEMM with Viterbi algorithm

- Finding the sequence of part-of-speech tags that is optimal for the whole sentence. Viterbi value of time t for

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

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

Open Source POS Tagger

- Stanford PoS Tagger python bind- Java based, but can be used in python but difficult to install
 - Flair - POS tagger available for python.
 - NLTK implementation to be very precise, around 97% and its quite fast.
 - SPACY
-

Q1.

Introduction- 3 marks

N-gram LM- 4 marks

Q2 Neural LM- 3 marks

Word embeddings- - 4/5 marks

Q3 Vector semantics- 6 marks

Q4. POS tagging- 5 marks

Q5 HMM/Viterbi - 4/5 marks

References

- <https://www.nltk.org/>
- <https://likegeeks.com/nlp-tutorial-using-python-nltk/>
- <https://www.guru99.com/pos-tagging-chunking-nltk.html>
- <https://medium.com/greyatom/learning-pos-tagging-chunking-in-nlp-85f7f811a8cb>
- <https://nlp.stanford.edu/software/tagger.shtml>
- <https://www.forbes.com/sites/mariyayao/2020/01/22/what-are--important-ai--machine-learning-trends-for-2020/#601ce9623239>
- <https://medium.com/fintechexplained/nlp-text-processing-in-data-science-projects-f083009d78fc>

Maximum entropy classification

From: LeChuck@yahoo.com

Har har!

Allow mes to introduce myself. My name is Lechuck ,and I got your email from the interwebs mail directory. I threw a dart at the directory and hit your name. You seam like a good lad based on your name, so I here I am writing this email.

I live out here in this faraway island, and I have some moneys (\$122K USD, to be exact) that I need to send overseas. Could you do mes a favor and help me with my moneys transfer?

- 1) Provide mes a bank account where this money would be transferred to.
- 2) Send me a picture of yourslefs so I know who to look for and thank when I sail to the US. Click heres to my Facebook
[www.lechuck.myfacebook.com/fake.link/give_me_money] and post me your pic.

Monkeys bananas money rich

As reward, I are willing to offer you 15% of the moneys as compensation for effort input after the successful transfer of this fund to your designate account overseas.
please feel free to contact ,me via this email address
lechuck@yahoo.com

Features and weights

F1=Email contains spelling/grammatical errors

F2=Email asks for money to be transferred

F3=Email mentions account holder's name

Weights for spam

W1 =Email contains spelling/grammatical errors): 0.5

W2=Email asks for money to be transferred): 0.2

W3=Email mentions account holder's name): -0.5

Weights for not spam

W1=(Email contains spelling/grammatical errors): -0.2

W2=(Email asks for money to be transferred): 0

W3 =(Email mentions account holder's name): 1

Function

$$f(x) = \begin{cases} 1, & \text{if the feature is present in } x \\ 0, & \text{otherwise} \end{cases}$$

Formula

$$Score_d(x) = \frac{\exp(\sum_{i=1}^N w_i(d)*f_i(x))}{\sum_d \exp(\sum_{i=1}^N w_i(d)*f_i(x))}$$

F1=Email contains spelling/grammatical errors -Yes

F2=Email asks for money to be transferred-Yes

F3=Email mentions account holder's name-No

Spam score

$$Score_{spam}(email_{LeChuck}) = \frac{\exp(\sum_{i=1}^N w_i(spam)*f_i(email_{LeChuck}))}{\sum_d \exp(\sum_{i=1}^N w_i(d)*f_i(email_{LeChuck}))} =$$

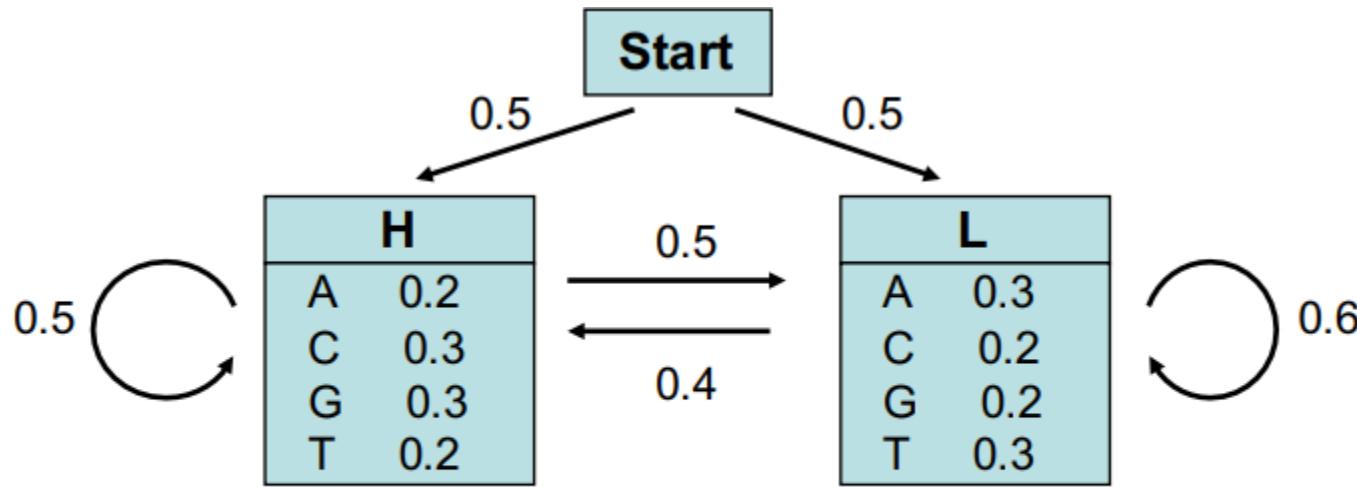
$$\frac{\exp(0.5*1+0.2*1-0.5*0)}{\exp(0.5*1+0.2*1-0.5*0)+\exp(-0.2*1+0*1+1*0)} = 0.71$$

Not spam score

$$Score_{notspam}(email_{LeChuck}) = \frac{\exp(\sum_{i=1}^N w_i(notspam)*f_i(email_{LeChuck}))}{\sum_d \exp(\sum_{i=1}^N w_i(d)*f_i(email_{LeChuck}))} =$$

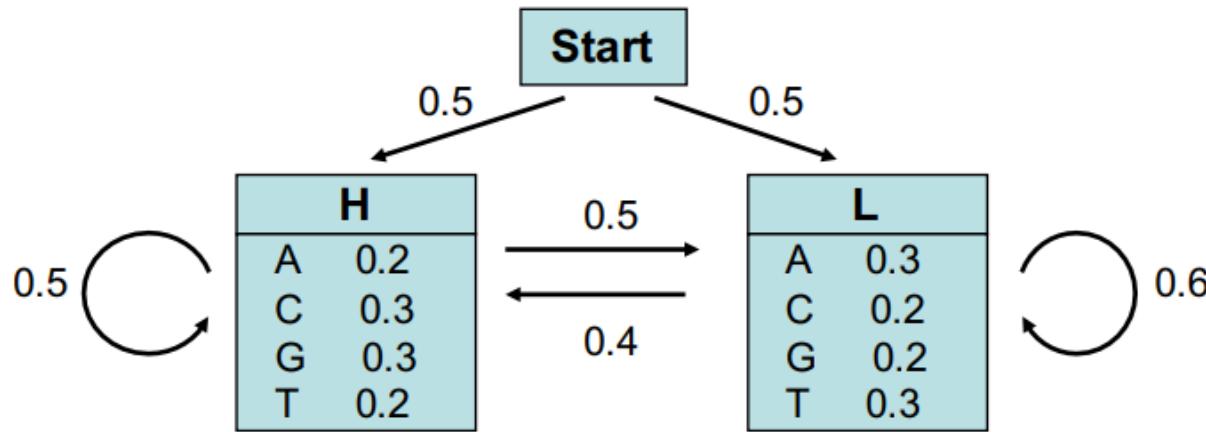
$$\frac{\exp(-0.2*1+0*1+1*0)}{\exp(0.5*1+0.2*1-0.5*0)+\exp(-0.2*1+0*1+1*0)} = 0.29$$

Forward algorithm-example



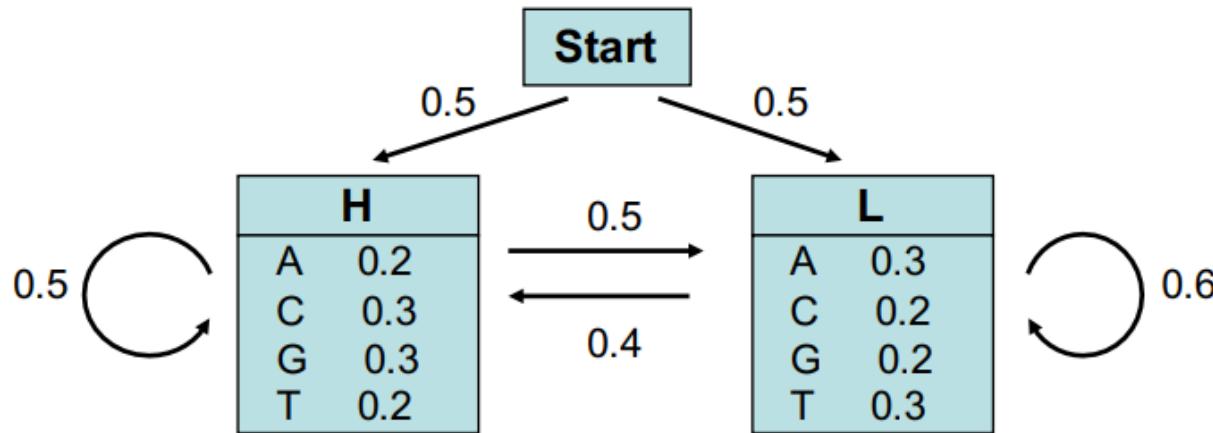
Consider now the sequence S= **GGCA**

	Start	G	G	C	A
H	0	$0.5 * 0.3 = 0.15$			
L	0	$0.5 * 0.2 = 0.1$			



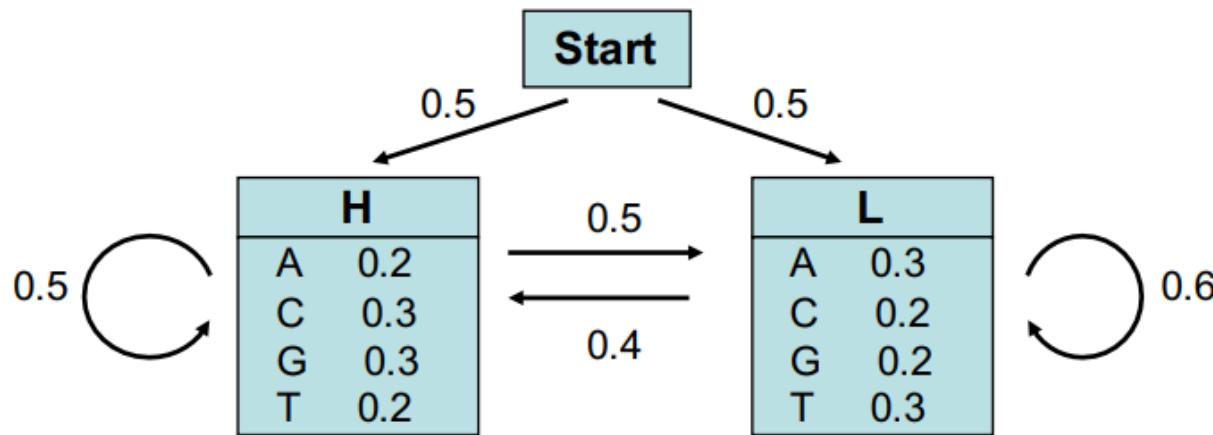
Consider now the sequence S= **GGCA**

	Start	G	G	C	A
H	0	$0.5 \cdot 0.3 = 0.15$	$0.15 \cdot 0.5 \cdot 0.3 + 0.1 \cdot 0.4 \cdot 0.3 = 0.0345$		
L	0	$0.5 \cdot 0.2 = 0.1$			



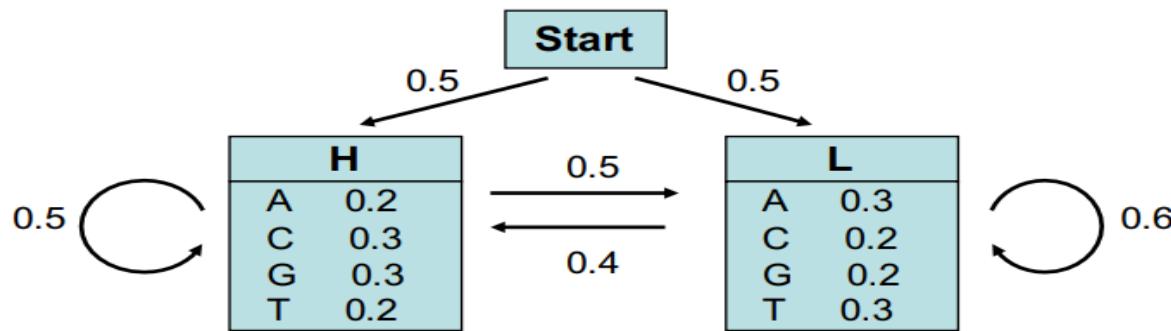
Consider now the sequence S= **GGCA**

	Start	G	G	C	A
H	0	$0.5 \cdot 0.3 = 0.15$	$0.15 \cdot 0.5 \cdot 0.3 + 0.1 \cdot 0.4 \cdot 0.3 = 0.0345$		
L	0	$0.5 \cdot 0.2 = 0.1$	$0.1 \cdot 0.6 \cdot 0.2 + 0.15 \cdot 0.5 \cdot 0.2 = 0.027$		



Consider now the sequence S= **GGCA**

	Start	G	G	C	A
H	0	$0.5 \cdot 0.3 = 0.15$	$0.15 \cdot 0.5 \cdot 0.3 + 0.1 \cdot 0.4 \cdot 0.3 = 0.0345$	$\dots + \dots$	
L	0	$0.5 \cdot 0.2 = 0.1$	$0.1 \cdot 0.6 \cdot 0.2 + 0.15 \cdot 0.5 \cdot 0.2 = 0.027$	$\dots + \dots$	



Consider now the sequence S= **GGCA**

	Start	G	G	C	A
H	0	$0.5 * 0.3 = 0.15$	$0.15 * 0.5 * 0.3 + 0.1 * 0.4 * 0.3 = 0.0345$	$\dots + \dots$	0.0013767
L	0	$0.5 * 0.2 = 0.1$	$0.1 * 0.6 * 0.2 + 0.15 * 0.5 * 0.2 = 0.027$	$\dots + \dots$	0.0024665

=> The probability that the sequence S was generated by the HMM model is thus $P(S)=0.0038432$.

$$\Sigma = 0.0038432$$



Natural Language Processing

DSECLZG530
Hidden Markov Models(MEMM)

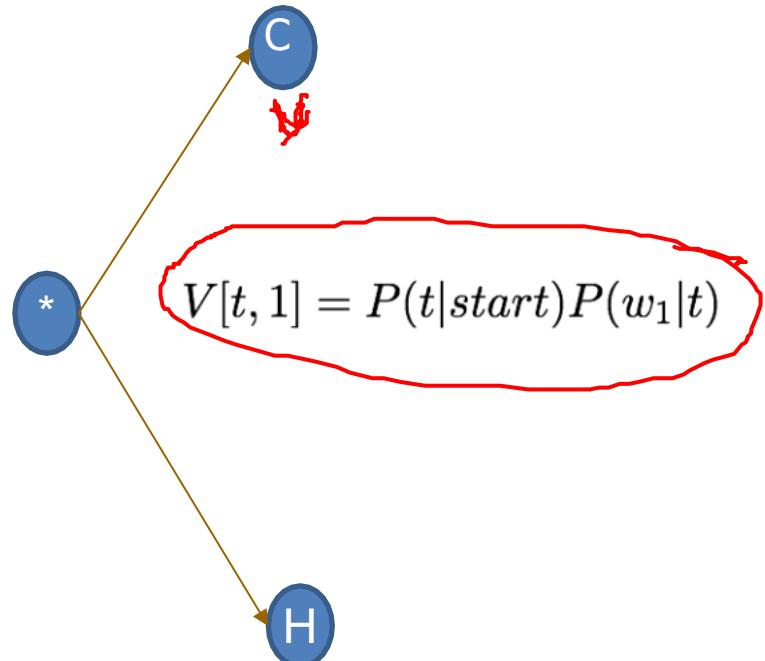


BITS Pilani
Pilani Campus

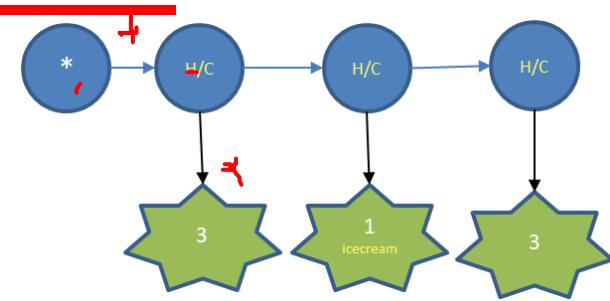
Hidden Markov Model

Example -1 – Viterbi Algorithm - Initialization

$$P(C) * P(3|C) = 0.2 * 0.1 = 0.02$$



$$P(H) * P(3|H) = 0.8 * 0.4 = 0.32$$



	Hot	Cold
<S>	0.8	0.2
A		
Hot	0.6	0.4
Cold	0.5	0.5

A	Hot	Cold
Hot	0.6	0.4
Cold	0.5	0.5

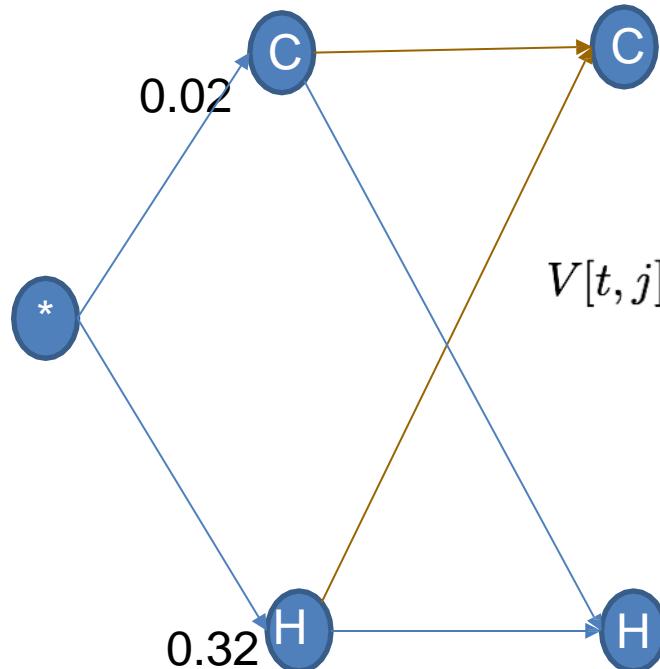
B	1	2	3
Hot	.2	.4	.4
Cold	.5	.4	.1

Hidden Markov Model

Example -1 – Viterbi Algorithm - Recursion

$$P(C)*P(C|C)*P(1|C) = 0.02*0.5*0.5 = 0.005$$

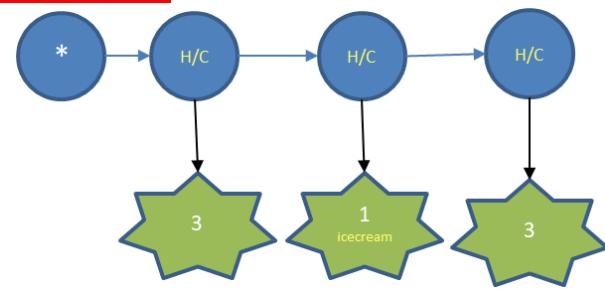
$$P(H)*P(C|H)*P(1|C) = 0.32*0.4*0.5 = 0.064$$



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

$$P(C)*P(H|C)*P(1|H) = 0.02*0.5*0.2 = 0.002$$

$$P(H)*P(H|H)*P(1|H) = 0.32*0.6*0.2 = 0.0384$$



	Hot	Cold
<S>	0.8	0.2
A		

A	Hot	Cold
Hot	0.6	0.4
Cold	0.5	0.5

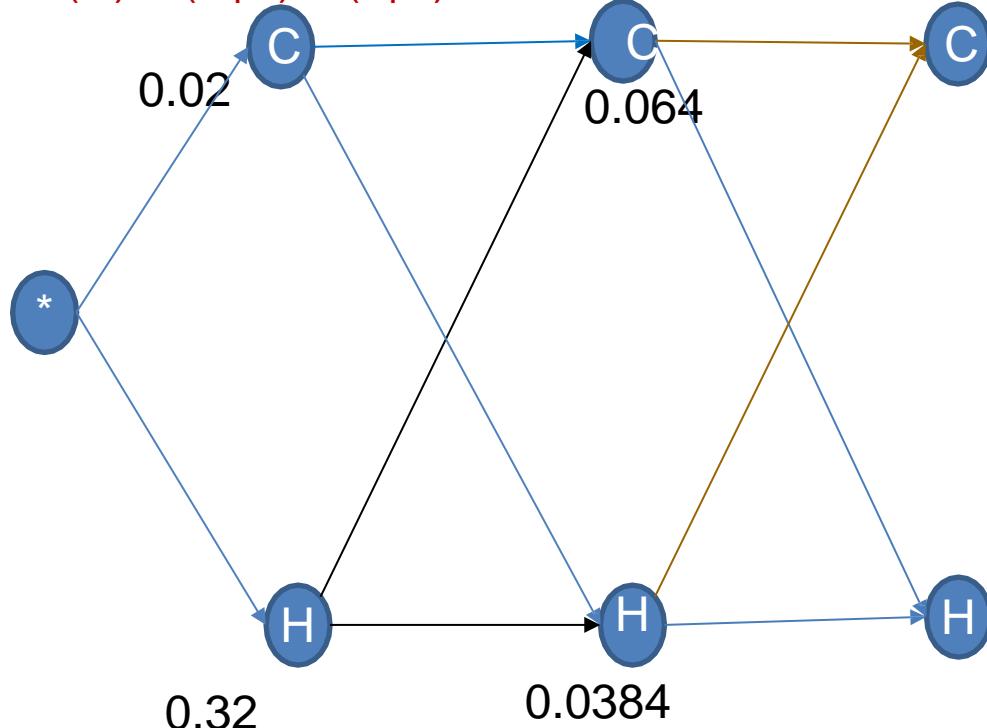
B	1	2	3
Hot	.2	.4	.4
Cold	.5	.4	.1

Hidden Markov Model

Example -1 – Viterbi Algorithm – Termination through Back Trace

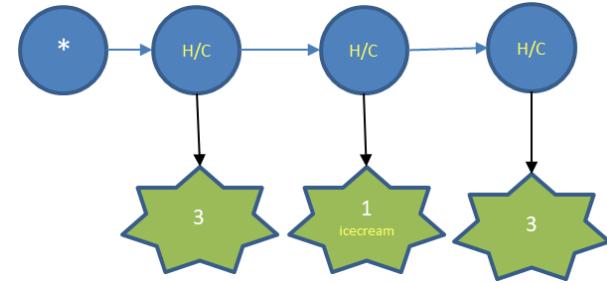
$$P(C)*P(C|C)*P(3|C) = 0.064 * 0.5 * 0.1 = \textcolor{red}{0.032}$$

$$P(H)*P(C|H)*P(3|C) = 0.0384 * 0.4 * 0.1 = \textcolor{green}{0.0015}$$



$$P(C)*P(H|C)*P(3|H) = 0.064 * 0.5 * 0.2 = 0.0064$$

$$P(H)*P(H|H)*P(3|H) = 0.0384 * 0.6 * 0.2 = 0.0046$$



Best Sequence:
Hot Cold

	Hot	Cold
<S>	0.8	0.2
A	Hot	Cold

	Hot	Cold
Hot	0.6	0.4
Cold	0.5	0.5

B	1	2	3
Hot	.2	.4	.4
Cold	.5	.4	.1

Hidden Markov Model

Example -1 – Viterbi Algorithm

Handwritten notes:

States: H, C
Observations: 1, 2, 3
Transitions: H → H, H → C, C → H, C → C
Emissions: P(H|1) = .12, P(H|2) = .10, P(C|1) = .25, P(C|2) = .20, P(C|3) = .10

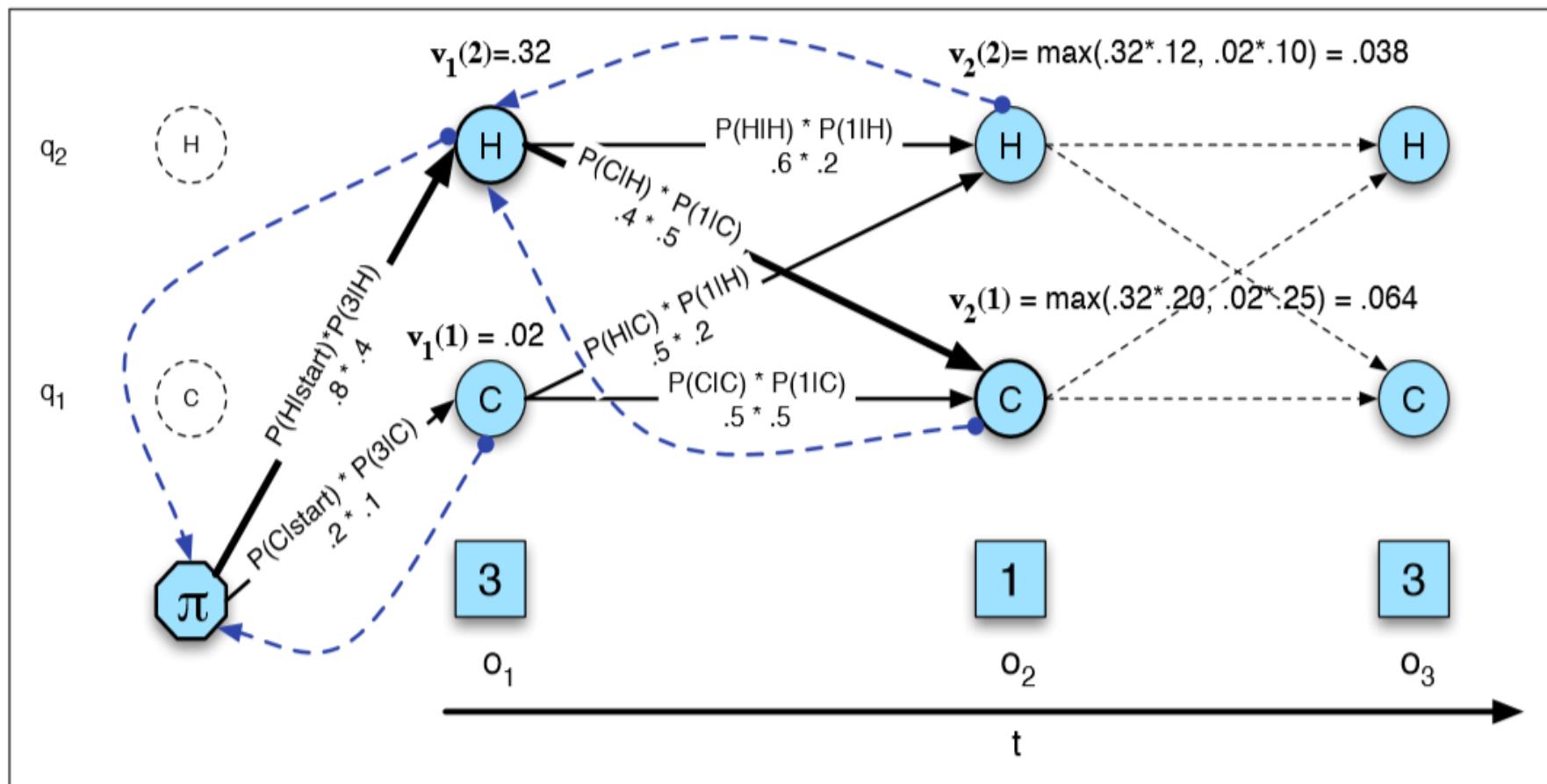


Figure A.10 The Viterbi backtrace. As we extend each path to a new state account for the next observation, we keep a backpointer (shown with broken lines) to the best path that led us to this state.

Source Credit : Speech and Language Processing - Jurafsky and Martin

POS Tagging – Viterbi Algorithm

function VITERBI(*observations* of len T ,*state-graph* of len N) **returns** *best-path*

create a path probability matrix $viterbi[N+2,T]$

for each state s **from** 1 **to** N **do** ; initialization step

$viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$

$backpointer[s,1] \leftarrow 0$

for each time step t **from** 2 **to** T **do** ; recursion step

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

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

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

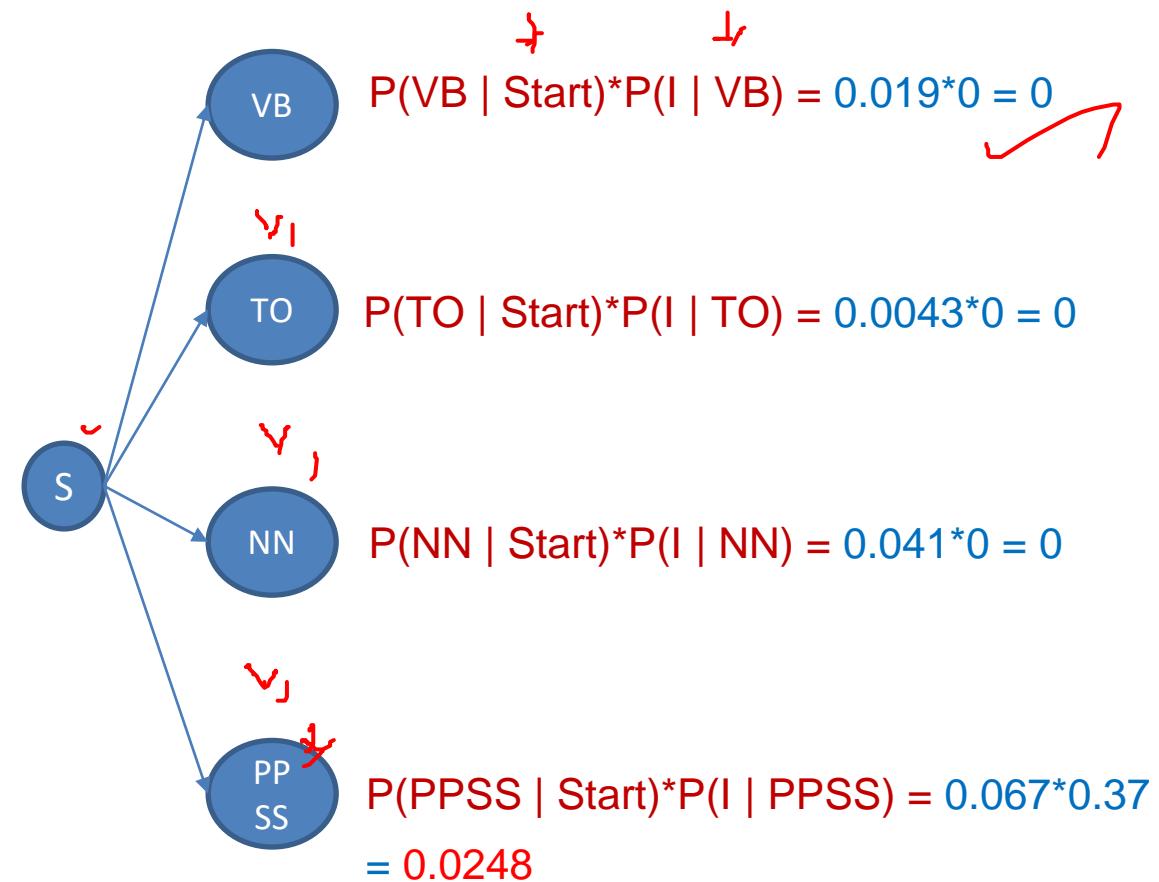
$viterbi[q_F, T] \leftarrow \max_{s=1}^N viterbi[s, T] * a_{s,q_F}$; termination step

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

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

Hidden Markov Model

POS Tagging – Example -2 – Viterbi Algorithm - Initialization



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

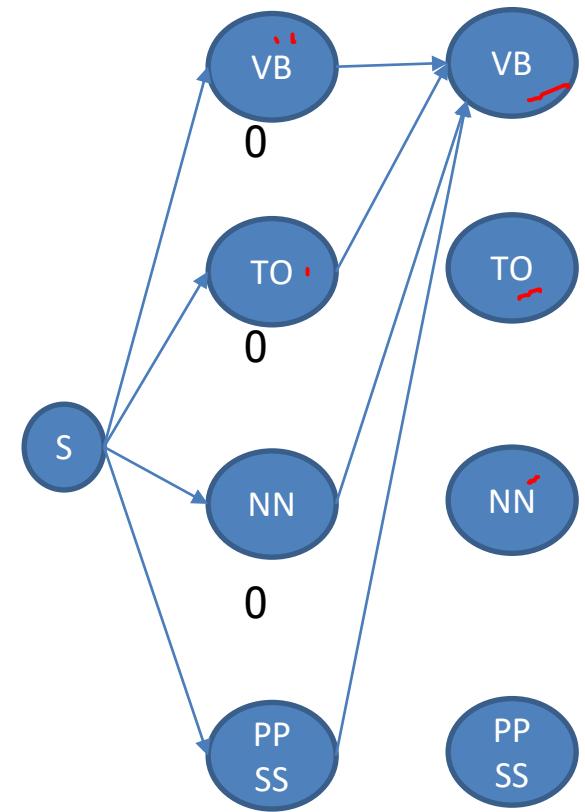
	VB	TO	NN	PPSS
<S>	0.019	0.0043	0.041	0.067
A	VB	TO	NN	PPSS
VB	0.0038	0.035	0.047	0.0070
TO	0.83	0	0.00047	0
NN	0.0040	0.016	0.087	0.0045
PPSS	0.23	0.00079	0.0012	0.00014

B	I	want	to	race
VB	0	0.0093	0	0.00012
TO	0	0	0.99	0
NN	0	0.000054	0	0.00057
PPSS	0.37	0	0	0

Hidden Markov Model

POS Tagging – Example -2 – Viterbi Algorithm - Recursion

~~0.0248~~



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

	VB	TO	NN	PPSS
<S>	0.019	0.0043	0.041	0.067
A	VB	TO	NN	PPSS
VB	0.0038	0.035	0.047	0.0070
TO	0.83	0	0.00047	0
NN	0.0040	0.016	0.087	0.0045
PPSS	0.23	0.00079	0.0012	0.00014

B	I	want	to	race
VB	0	0.0093	0	0.00012
TO	0	0	0.99	0
NN	0	0.000054	0	0.00057
PPSS	0.37	0	0	0

$$P(VB) * P(VB | VB)*P(want | VB) = 0*0.0038*0.0093= 0$$

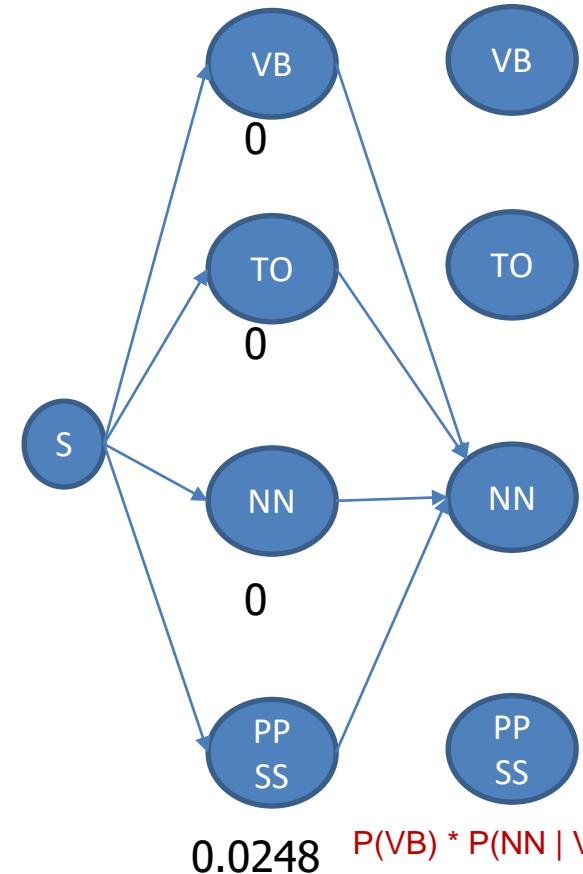
$$P(TO) * P(VB | TO)*P(want | VB) = 0*0.83*0.0093= 0$$

$$P(NN) * P(VB | NN)*P(want | VB) = 0*0.004*0.0093= 0$$

$$P(PPSS) * P(VB | PPSS)*P(want | VB) = 0.0248*0.23*0.0093= 0.0005$$

Hidden Markov Model

POS Tagging – Example -2 – Viterbi Algorithm - Recursion



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

	VB	TO	NN	PPSS
<S>	0.019	0.0043	0.041	0.067
A	VB	TO	NN	PPSS
VB	0.0038	0.035	0.047	0.0070
TO	0.83	0	0.00047	0
NN	0.0040	0.016	0.087	0.0045
PPSS	0.23	0.00079	0.0012	0.00014

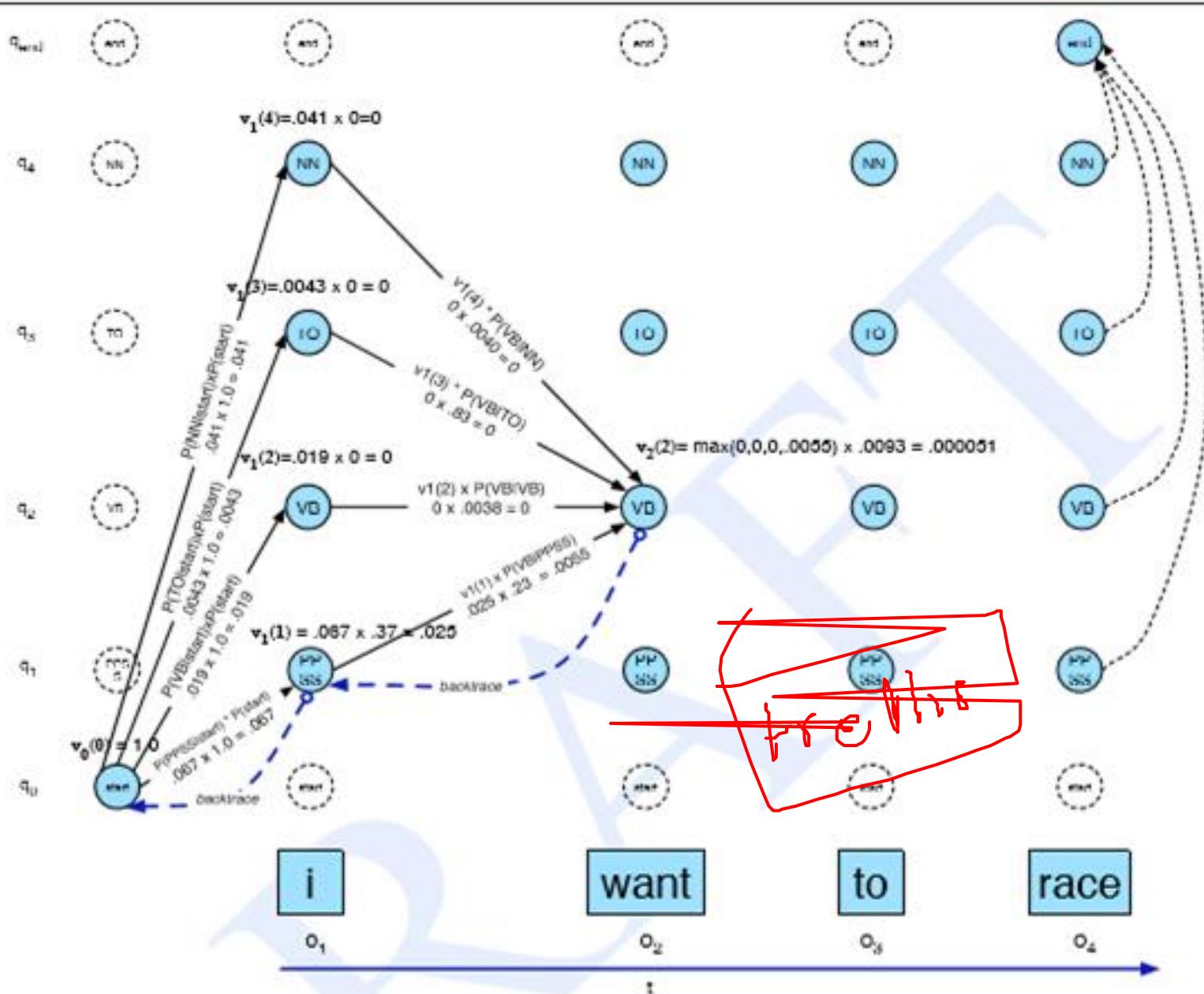
B	I	want	to	race
VB	0	0.0093	0	0.00012
TO	0	0	0.99	0
NN	0	0.000054	0	0.00057
PPSS	0.37	0	0	0

$$P(VB) * P(NN | VB)*P(want | NN) = 0*0.047*0.000054= 0$$

$$P(TO) * P(NN | TO)*P(want | NN) = 0*0.0047*0.000054= 0$$

$$P(NN) * P(NN | NN)*P(want | NN) = 0*0.087*0.000054= 0$$

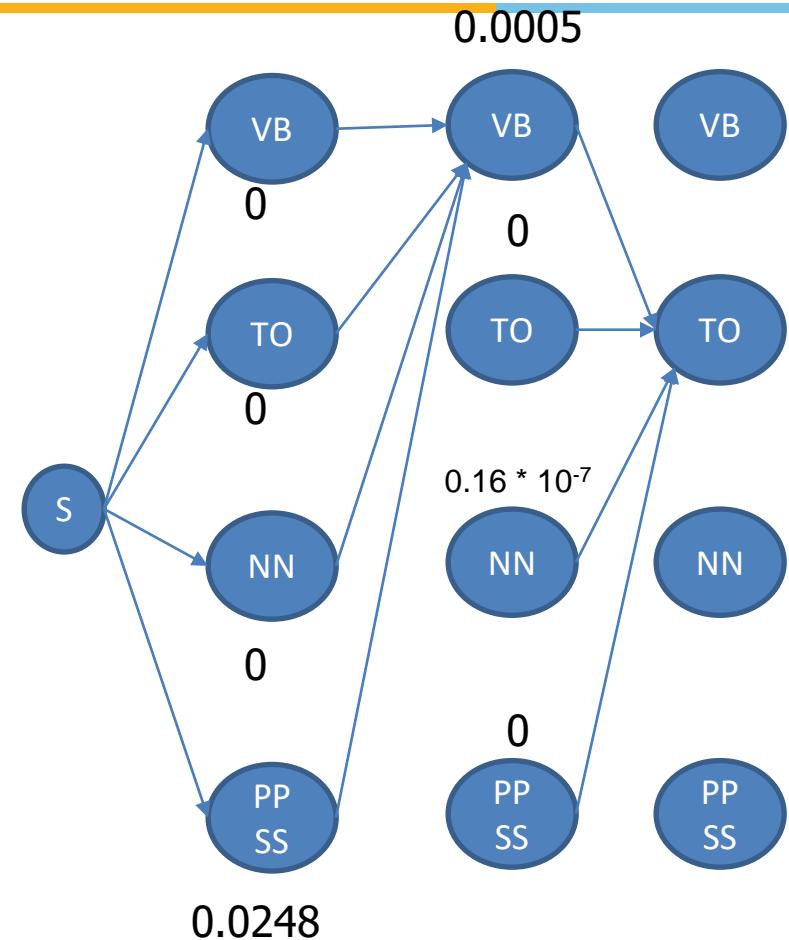
$$P(PPSS) * P(NN | PPSS)*P(want | NN) = 0.0012*0.23*0.000054= 0.16 * 10^{-7}$$



Source Credit : Speech and Language Processing - Jurafsky and Martin

Hidden Markov Model

POS Tagging – Example -2 – Viterbi Algorithm - Recursion



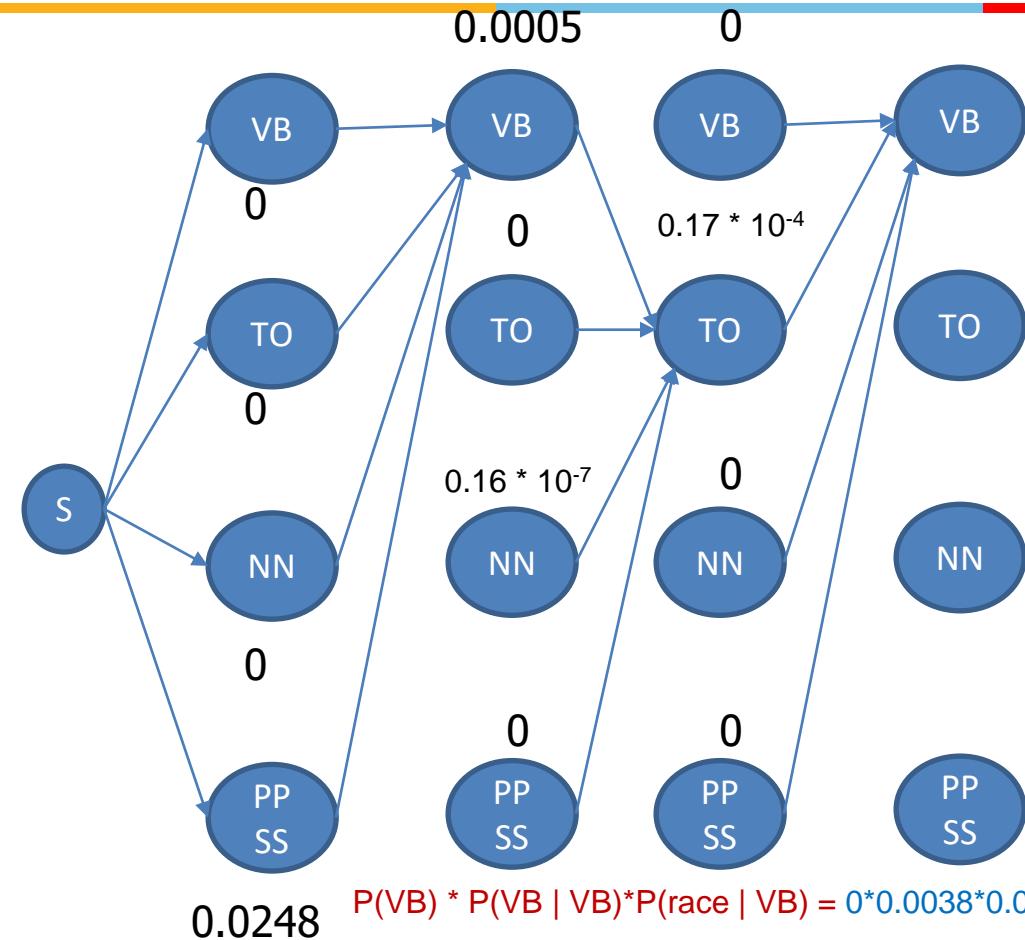
$$V[t, j] = \max_{t'} V[t', j - 1] P(t|t') P(w_j|t)$$

	VB	TO	NN	PPSS
<S>	0.019	0.0043	0.041	0.067
A	VB	TO	NN	PPSS
VB	0.0038	0.035	0.047	0.0070
TO	0.83	0	0.00047	0
NN	0.0040	0.016	0.087	0.0045
PPSS	0.23	0.00079	0.0012	0.00014

B	I	want	to	race
VB	0	0.0093	0	0.00012
TO	0	0	0.99	0
NN	0	0.000054	0	0.00057
PPSS	0.37	0	0	0

Hidden Markov Model

POS Tagging – Example -2 – Viterbi Algorithm - Recursion



$$P(VB) * P(VB | VB) * P(race | VB) = 0 * 0.0038 * 0.00012 = 0$$

$$\begin{aligned} P(TO) * P(VB | TO) * P(race | VB) &= 0.17 * 10^{-4} * 0.83 * 0.00012 \\ &= 0.17 * 10^{-8} \end{aligned}$$

$$P(NN) * P(VB | NN) * P(race | VB) = 0 * 0.0040 * 0.00012 = 0$$

$$P(PPSS) * P(VB | PPSS) * P(race | VB) = 0 * 0.23 * 0.00012 = 0$$

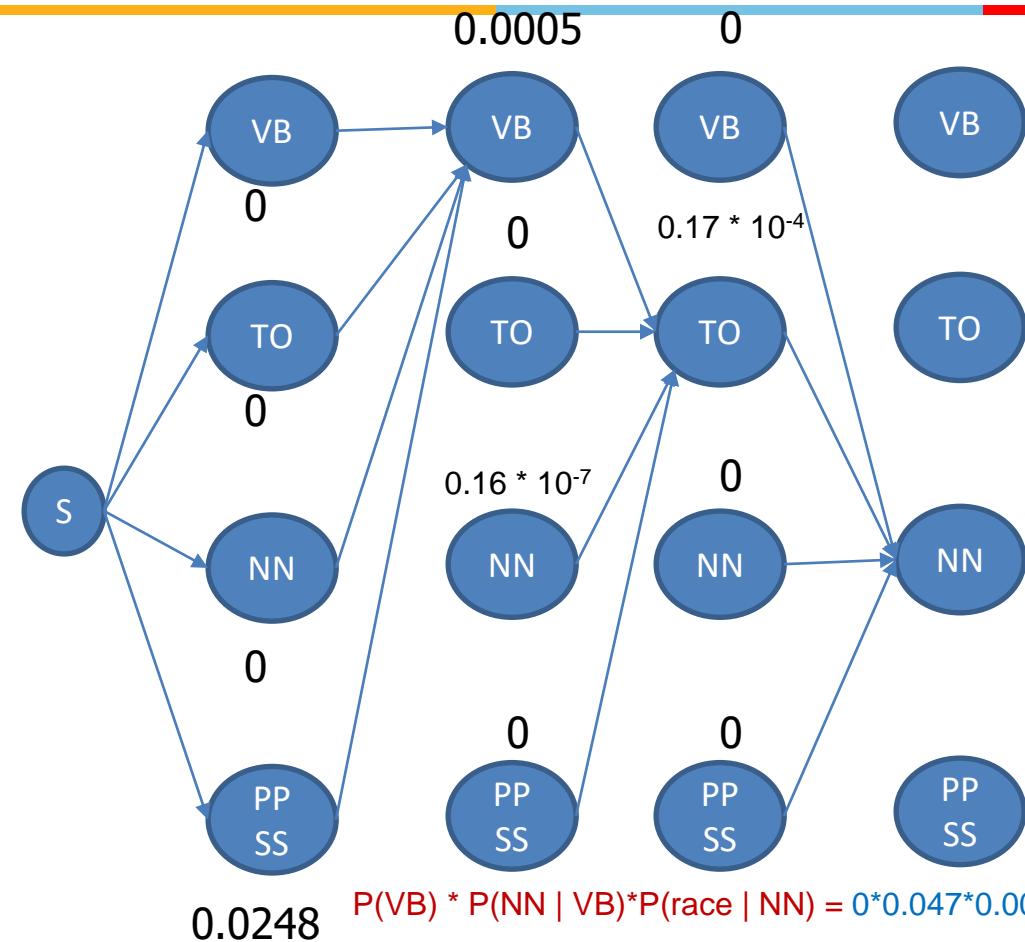
$$V[t, j] = \max_{t'} V[t', j - 1] P(t|t') P(w_j|t)$$

	VB	TO	NN	PPSS
<S>	0.019	0.0043	0.041	0.067
A	VB	TO	NN	PPSS
VB	0.0038	0.035	0.047	0.0070
TO	0.83	0	0.00047	0
NN	0.0040	0.016	0.087	0.0045
PPSS	0.23	0.00079	0.0012	0.00014

B	I	want	to	race
VB	0	0.0093	0	0.00012
TO	0	0	0.99	0
NN	0	0.000054	0	0.00057
PPSS	0.37	0	0	0

Hidden Markov Model

POS Tagging – Example -2 – Viterbi Algorithm - Recursion



$$P(VB) * P(NN | VB) * P(race | NN) = 0 * 0.047 * 0.00057$$

$$P(TO) * P(NN | TO) * P(race | NN) = 0.17 * 10^{-4} * 0.00047 * 0.00057$$

$$= 0.46 * 10^{-11}$$

$$P(NN) * P(NN | NN) * P(race | NN) = 0 * 0.087 * 0.00057 = 0$$

$$P(PPSS) * P(NN | PPSS) * P(race | NN) = 0 * 0.0012 * 0.00057 = 0$$

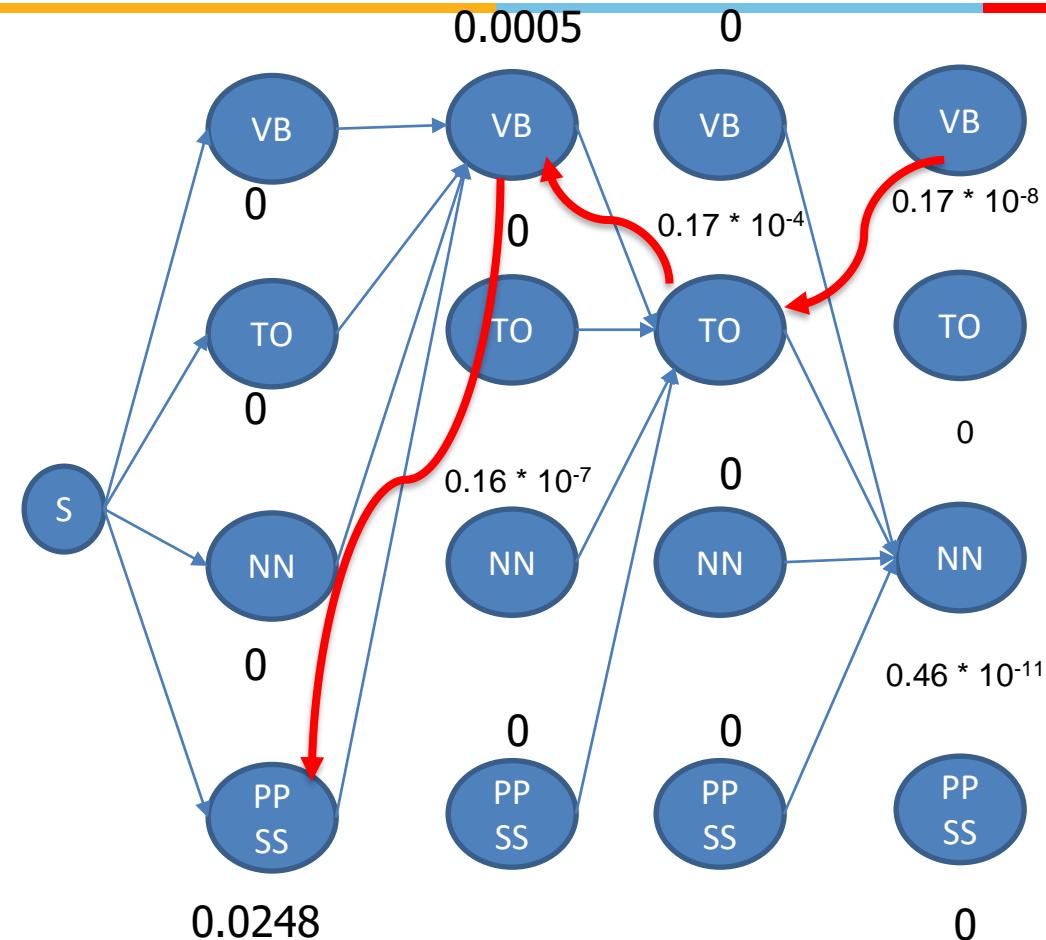
$$V[t, j] = \max_{t'} V[t', j - 1] P(t|t') P(w_j|t)$$

	VB	TO	NN	PPSS
<S>	0.019	0.0043	0.041	0.067
A	VB	TO	NN	PPSS
VB	0.0038	0.035	0.047	0.0070
TO	0.83	0	0.00047	0
NN	0.0040	0.016	0.087	0.0045
PPSS	0.23	0.00079	0.0012	0.00014

B	I	want	to	race
VB	0	0.0093	0	0.00012
TO	0	0	0.99	0
NN	0	0.000054	0	0.00057
PPSS	0.37	0	0	0

Hidden Markov Model

POS Tagging – Example -2 – Viterbi Algorithm - Termination



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

	VB	TO	NN	PPSS
<S>	0.019	0.0043	0.041	0.067
A	VB	TO	NN	PPSS
VB	0.0038	0.035	0.047	0.0070
TO	0.83	0	0.00047	0
NN	0.0040	0.016	0.087	0.0045
PPSS	0.23	0.00079	0.0012	0.00014

B	I	want	to	race
VB	0	0.0093	0	0.00012
TO	0	0	0.99	0
NN	0	0.000054	0	0.00057
PPSS	0.37	0	0	0

Hidden Markov Model



POS Tagging – Example -3 – Viterbi Algorithm – Practice

Transition matrix: $P(t_i|t_{i-1})$

	NOUN	Verb	Det	Prep	ADV	STOP
<S>	.3	.1	.3	.2	.1	0
Noun	.2	.4	.01	.3	.04	.05
Verb	.3	.05	.3	.2	.1	.05
Det	.9	.01	.01	.01	.07	0
Prep	.4	.05	.4	.1	.05	0
Adv	.1	.5	.1	.1	.1	.1

Emission matrix: $P(w_i|t_i)$

	a	cat	doctor	in	is	the	very
Noun	0	.5	.4	0	0.1	0	0
Verb	0	0	.1	0	.9	0	0
Det	.3	0	0	0	0	.7	0
Prep	0	0	0	1.0	0	0	0
Adv	0	0	0	.1	0	0	.9

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

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

$$V(\text{Noun}, \text{the}) = P(\text{Noun}|<\text{S}>)P(\text{the}|\text{Noun}) = .3 \times 0 = 0$$

$$V(\text{Verb}, \text{the}) = P(\text{Verb}|<\text{S}>)P(\text{the}|\text{Verb}) = .1 \times 0 = 0$$

$$V(\text{Det}, \text{the}) = P(\text{Det}|<\text{S}>)P(\text{the}|\text{Det}) = .3 \times .7 = .21$$

$$V(\text{Prep}, \text{the}) = P(\text{Prep}|<\text{S}>)P(\text{the}|\text{Prep}) = .2 \times 0 = 0$$

$$V(\text{Adv}, \text{the}) = P(\text{Adv}|<\text{S}>)P(\text{the}|\text{Adv}) = .2 \times 0 = 0$$

POS Tagging – Example -3 – Viterbi Algorithm – Practice

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

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

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

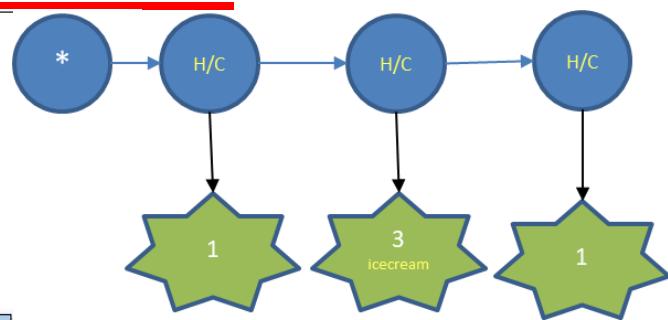
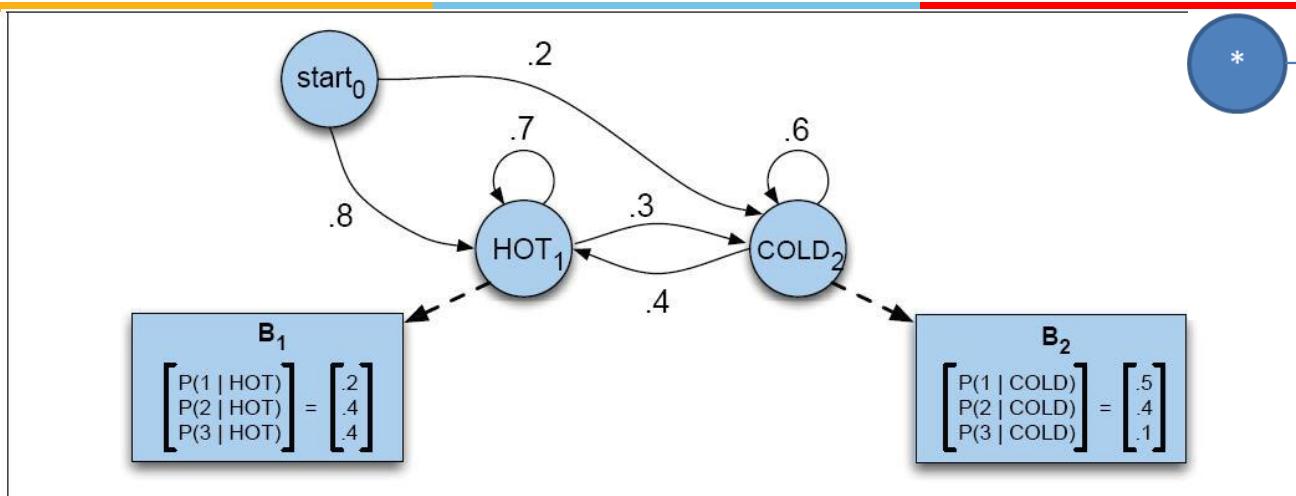
	w1=the	w2=doctor	w3=is	w4=in	STOP
Noun	0	.0756	.001512	0	
Verb	0	.00021	.027216	0	
Det	.21	0	0	0	.0000272
Prep	0	0	0	.005443	
Adv	0	0	0	.000272	

Det Noun Verb Prep

Hidden Markov Model



Example -4 – Naïve Search

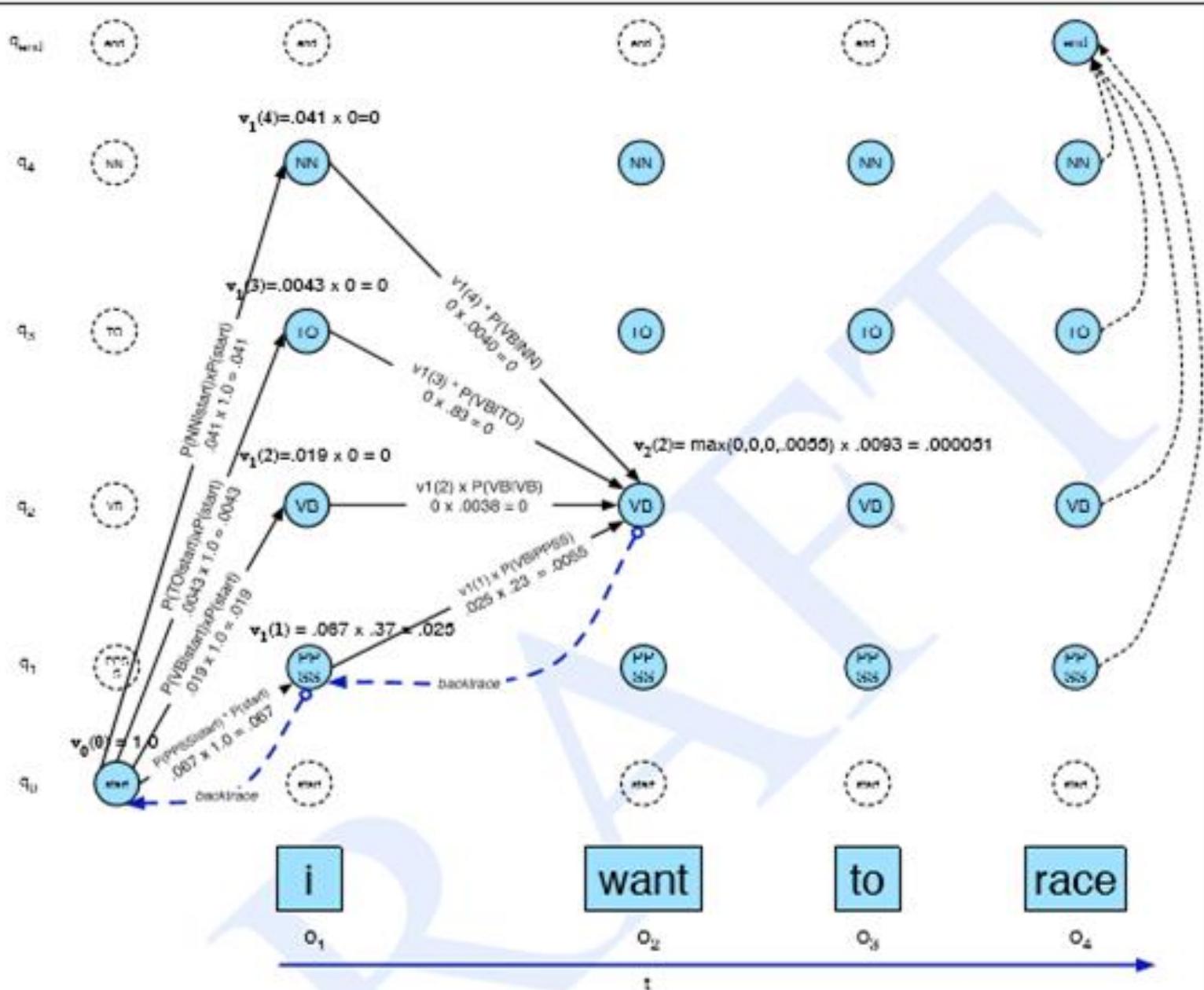


HHH	$P(H)^*P(1 H)^*P(H H)^*P(3 H)^*P(H H)^*P(1 H)$	
HHC		
HCC		
CCC		
CHC	$P(C)^*P(1 C)^*P(H C)^*P(3 H)^*P(C H)^*P(1 C)$ $=0.2^*0.5^*0.4^*0.4^*0.3^*0.5$	0.0024
CCH		
CHH		
HCH		

	Hot	Cold
<S>	0.8	0.2

A	Hot	Cold
Hot	0.7	0.3
Cold	0.4	0.6

B	1	2	3
Hot	.2	.4	.4
Cold	.5	.4	.1

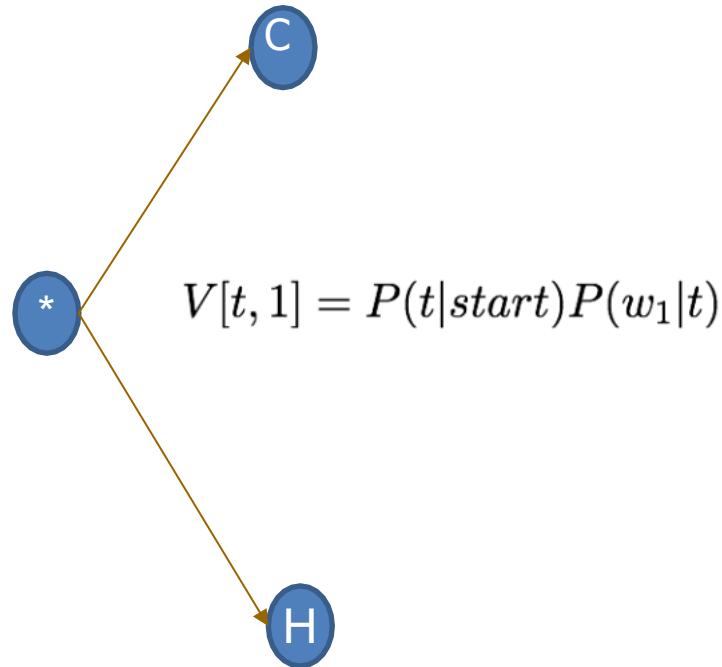


Source Credit : Speech and Language Processing - Jurafsky and Martin

Hidden Markov Model

Example – 5 – Observation Likelihood by Forward Propagation

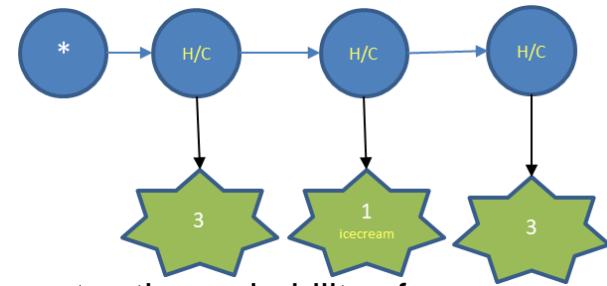
$$P(C)*P(3|C) = 0.2*0.1 = 0.02$$



$$P(H)*P(3|H) = 0.8*0.4 = 0.32$$

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

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



Efficiently computes the probability of an observed sequence given a model

$$P(\text{sequence} \mid \text{model})$$

Identical to Viterbi; **replace the MAX with a SUM**

	Hot	Cold
<S>	0.8	0.2
A		
Hot	0.6	0.4
Cold	0.5	0.5

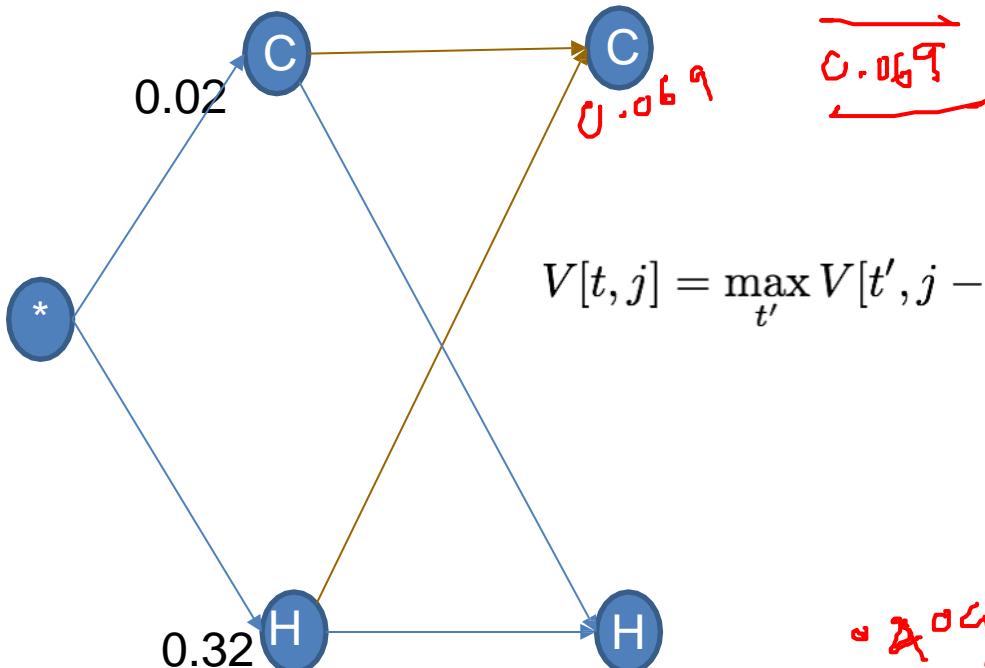
B	1	2	3
Hot	.2	.4	.4
Cold	.5	.4	.1

Hidden Markov Model

Example -5 – Observation Likelihood by Forward Propagation - Recursion

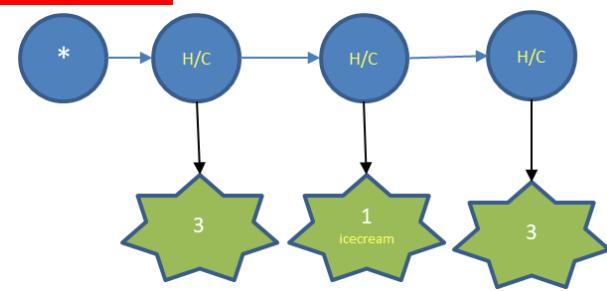
$$P(C) * P(C|C) * P(1|C) = 0.02 * 0.5 * 0.5 = 0.005$$

$$P(H) * P(C|H) * P(1|C) = 0.32 * 0.4 * 0.5 = 0.064$$



$$P(C) * P(H|C) * P(1|H) = 0.02 * 0.5 * 0.2 = 0.002$$

$$P(H) * P(H|H) * P(1|H) = 0.32 * 0.6 * 0.2 = 0.0384$$



	Hot	Cold
<S>	0.8	0.2
A		
Hot	0.6	0.4
Cold	0.5	0.5

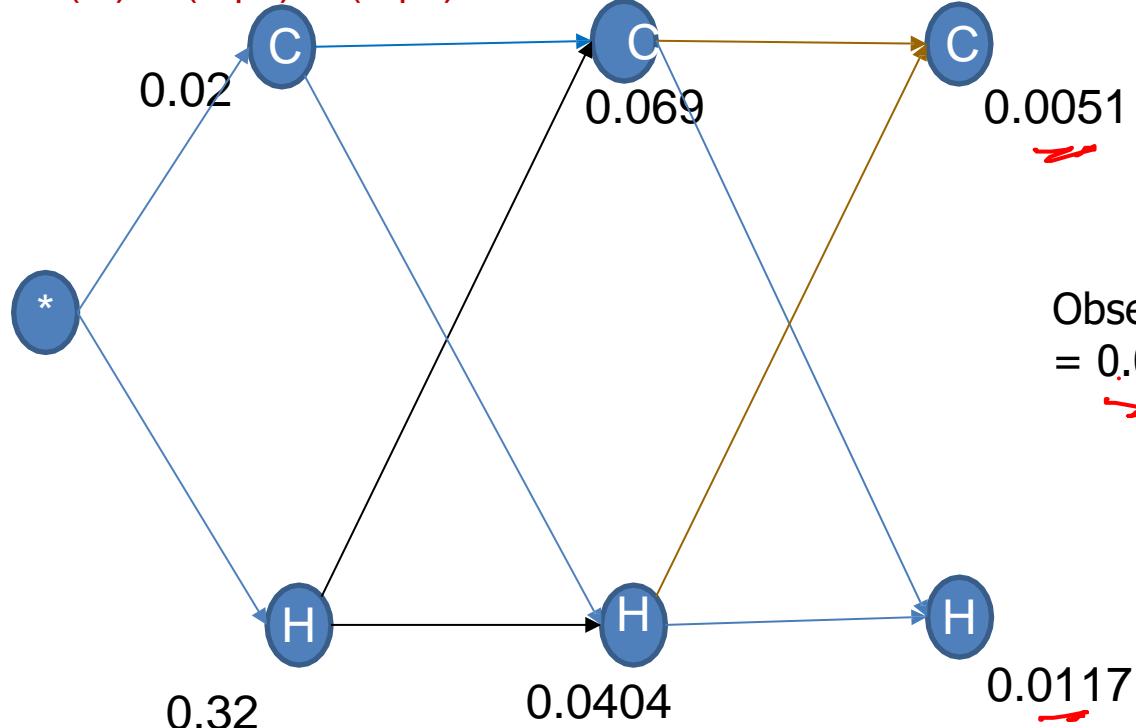
A	1	2	3
B	.2	.4	.4
C	.5	.4	.1

Hidden Markov Model

Example -5 – Observation Likelihood by Forward Propagation

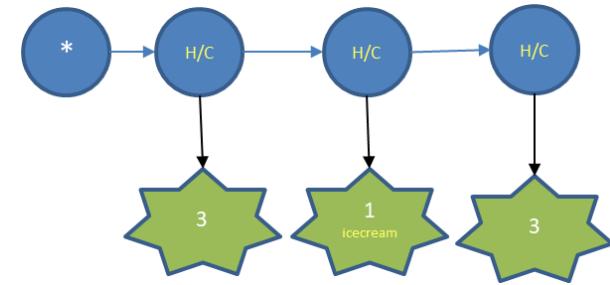
$$P(C)*P(C|C)*P(R|C) = 0.069 * 0.5 * 0.1 = \textcolor{red}{0.0035}$$

$$P(H)*P(C|H)*P(R|C) = 0.0404 * 0.4 * 0.1 = \textcolor{green}{0.0016}$$



$$P(C)*P(H|C)*P(R|H) = 0.069 * 0.5 * 0.2 = 0.0069$$

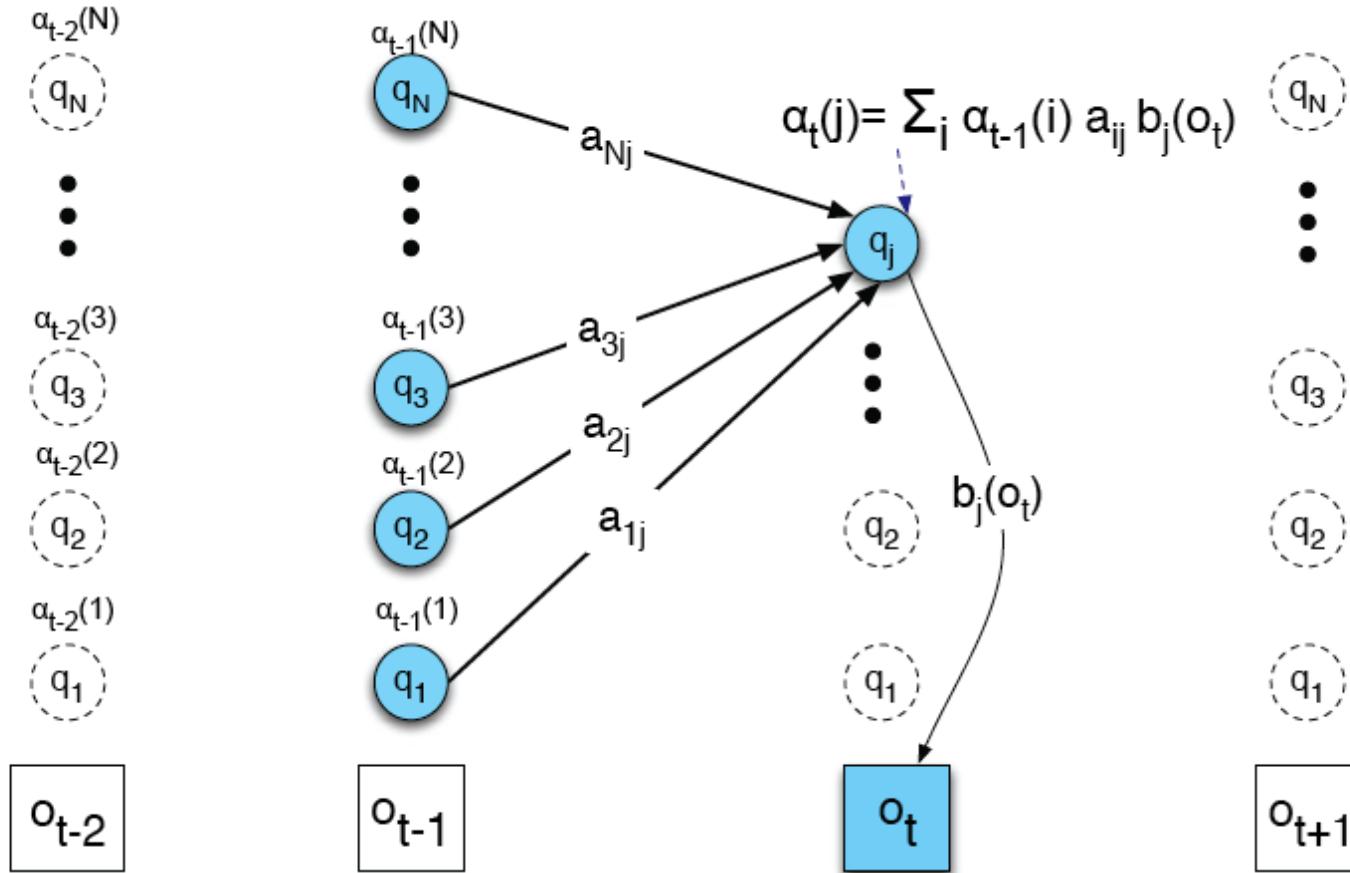
$$P(H)*P(H|H)*P(R|H) = 0.0404 * 0.6 * 0.2 = 0.0048$$



	Hot	Cold
<S>	0.8	0.2
A		
Hot	0.6	0.4
Cold	0.5	0.5

B	1	2	3
Hot	.2	.4	.4
Cold	.5	.4	.1

Observation Likelihood – Forward Propagation



Source Credit : Speech and Language Processing - Jurafsky and Martin

Hidden Markov Model

Observation Likelihood – Forward Propagation

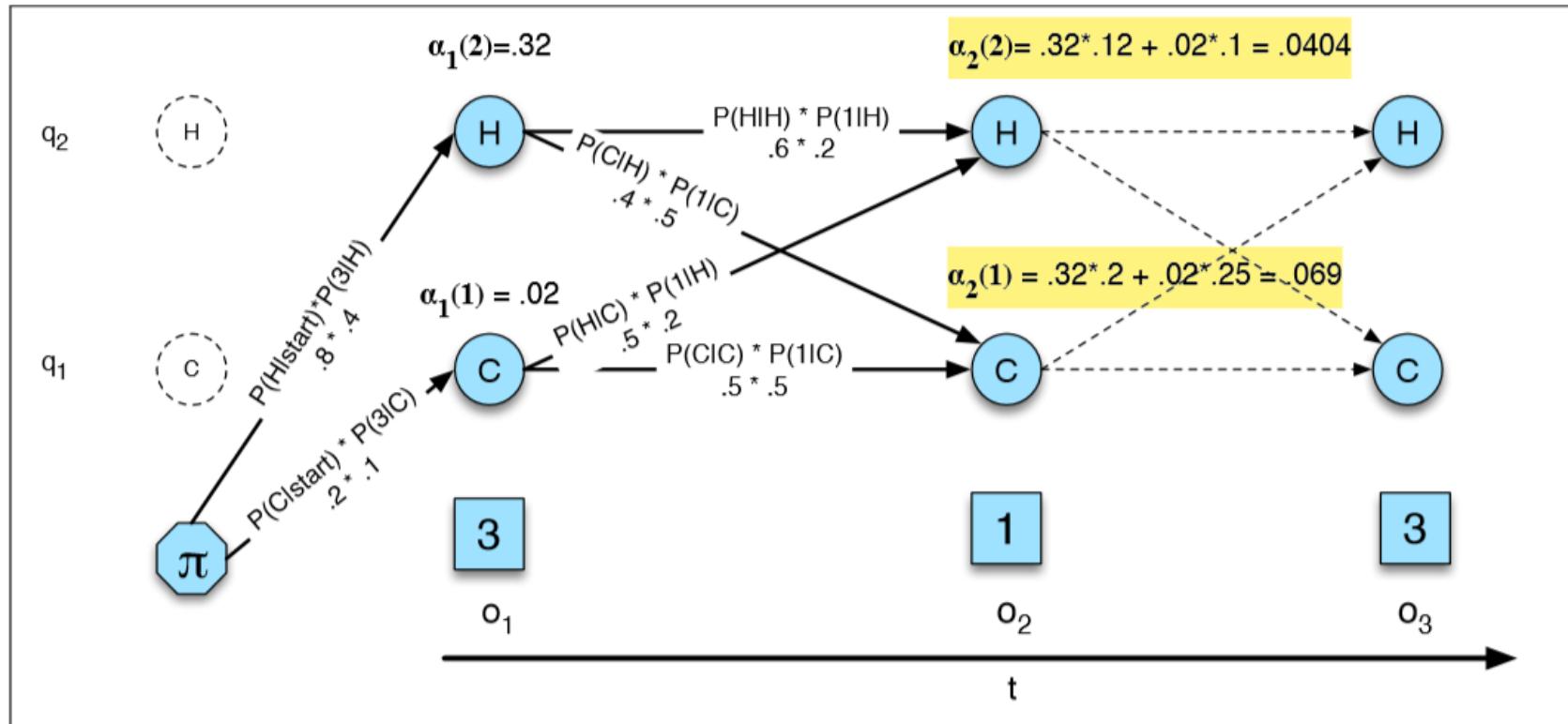


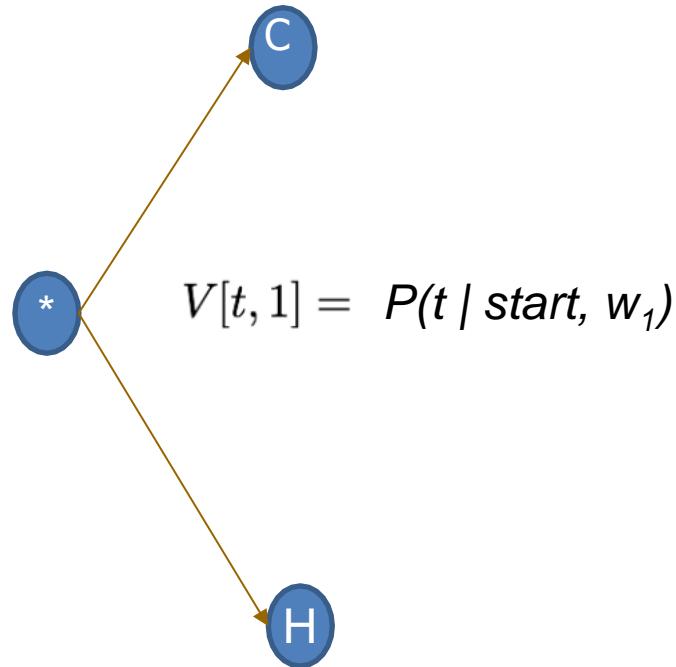
Figure A.5 The forward trellis for computing the total observation likelihood for the ice-cream events 3 1 3. Hidden states are in circles, observations in squares. The figure shows the computation of $\alpha_t(j)$ for two states at two time steps. The computation in each cell follows Eq. A.12: $\alpha_t(j) = \sum_{i=1}^N \alpha_{t-1}(i) a_{ij} b_j(o_t)$. The resulting probability expressed in each cell is Eq. A.11: $\alpha_t(j) = P(o_1, o_2 \dots o_t, q_t = j | \lambda)$.

Source Credit : Speech and Language Processing - Jurafsky and Martin

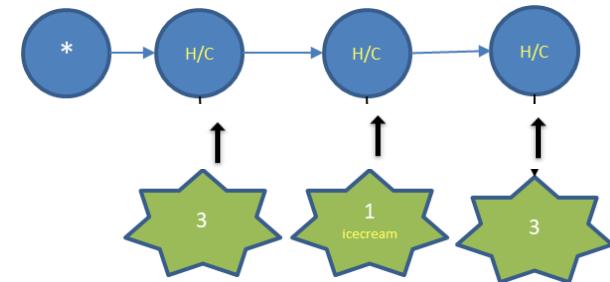
Maximum Entropy Markov Model

Example -6 – Viterbi Algorithm - Initialization

$$P(C \mid \text{Start}, 3) = 0.55$$



$$P(H \mid \text{Start}, 3) = 0.45$$



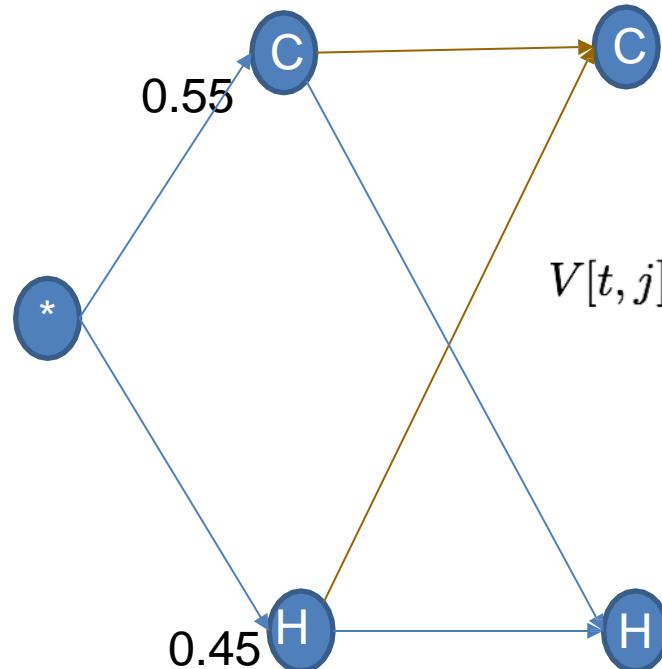
$X_1 = T_{i-1}$	$X_2 = w_i$	$\text{Label } Y = T_i$	$P(T_i \mid T_{i-1} W_i)$
Start	1	Hot	0.3
Start	1	Cold	
Start	3	Hot	0.45
Start	3	Cold	
.....
Hot	1	Hot	0.15
Hot	1	Cold	
Hot	3	Hot	0.6
Hot	3	Cold	
.....
Cold	1	Hot	0.5
Cold	1	Cold	
Cold	3	Hot	0.8
Cold	3	Cold	
....

Maximum Entropy Markov Model

Example -6 – Viterbi Algorithm - Recursion

$$P(C) * P(C|C, 1) = 0.55 * 0.5 = 0.275$$

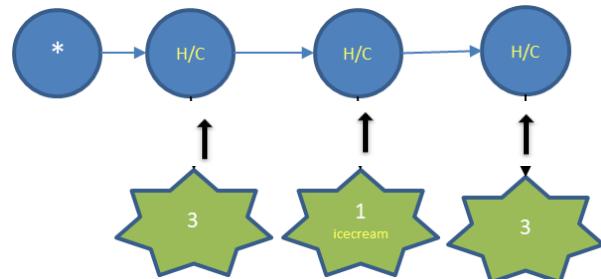
$$P(H) * P(C|H, 1) = 0.45 * 0.85 = 0.3825$$



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

$$P(C) * P(H|C, 1) = 0.55 * 0.5 = 0.275$$

$$P(H) * P(H|H, 1) = 0.45 * 0.15 = 0.0675$$



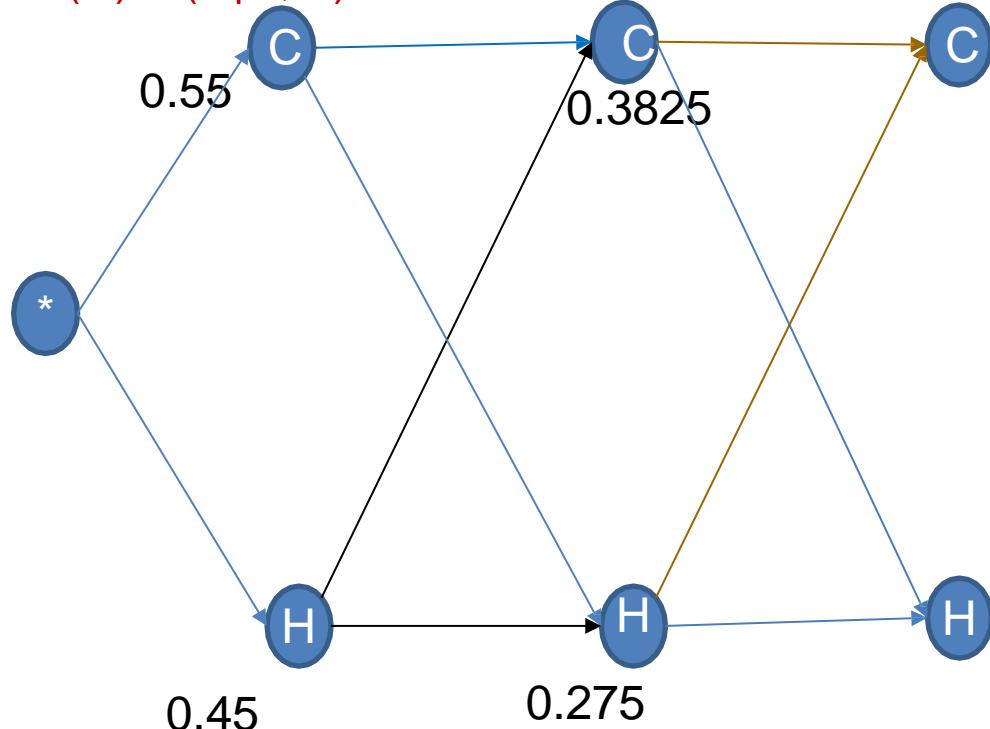
$X_1 = T_{i-1}$	$X_2 = w_i$	Label $Y = T_i$	$P(T_i T_{i-1}, w_i)$
Start	1	Hot	0.3
Start	1	Cold	
Start	3	Hot	0.45
Start	3	Cold	
.....
Hot	1	Hot	0.15
Hot	1	Cold	
Hot	3	Hot	0.6
Hot	3	Cold	
.....
Cold	1	Hot	0.5
Cold	1	Cold	
Cold	3	Hot	0.8
Cold	3	Cold	
....

Maximum Entropy Markov Model

Example -6 – Viterbi Algorithm – Termination through Back Trace

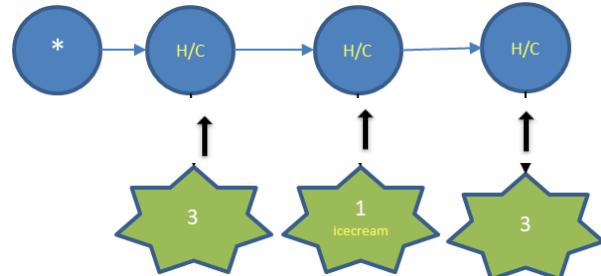
$$P(C) * P(C|C, 3) = 0.3825 * 0.2 = 0.0765$$

$$P(H) * P(C|H, 3) = 0.275 * 0.4 = 0.11$$



$$P(C) * P(H|C, 3) = 0.3825 * 0.8 = 0.306$$

$$P(H) * P(H|H, 3) = 0.275 * 0.6 = 0.165$$



$X_1 = T_{i-1}$	$X_2 = w_i$	Label $Y = T_i$	$P(T_i / T_{i-1} W_i)$
Start	1	Hot	0.3
Start	1	Cold	
Start	3	Hot	0.45
Start	3	Cold	
.....
Hot	1	Hot	0.15
Hot	1	Cold	
Hot	3	Hot	0.6
Hot	3	Cold	
.....
Cold	1	Hot	0.5
Cold	1	Cold	
Cold	3	Hot	0.8
Cold	3	Cold	
....

Maximum Entropy Markov Model

Example -6 – Viterbi Algorithm

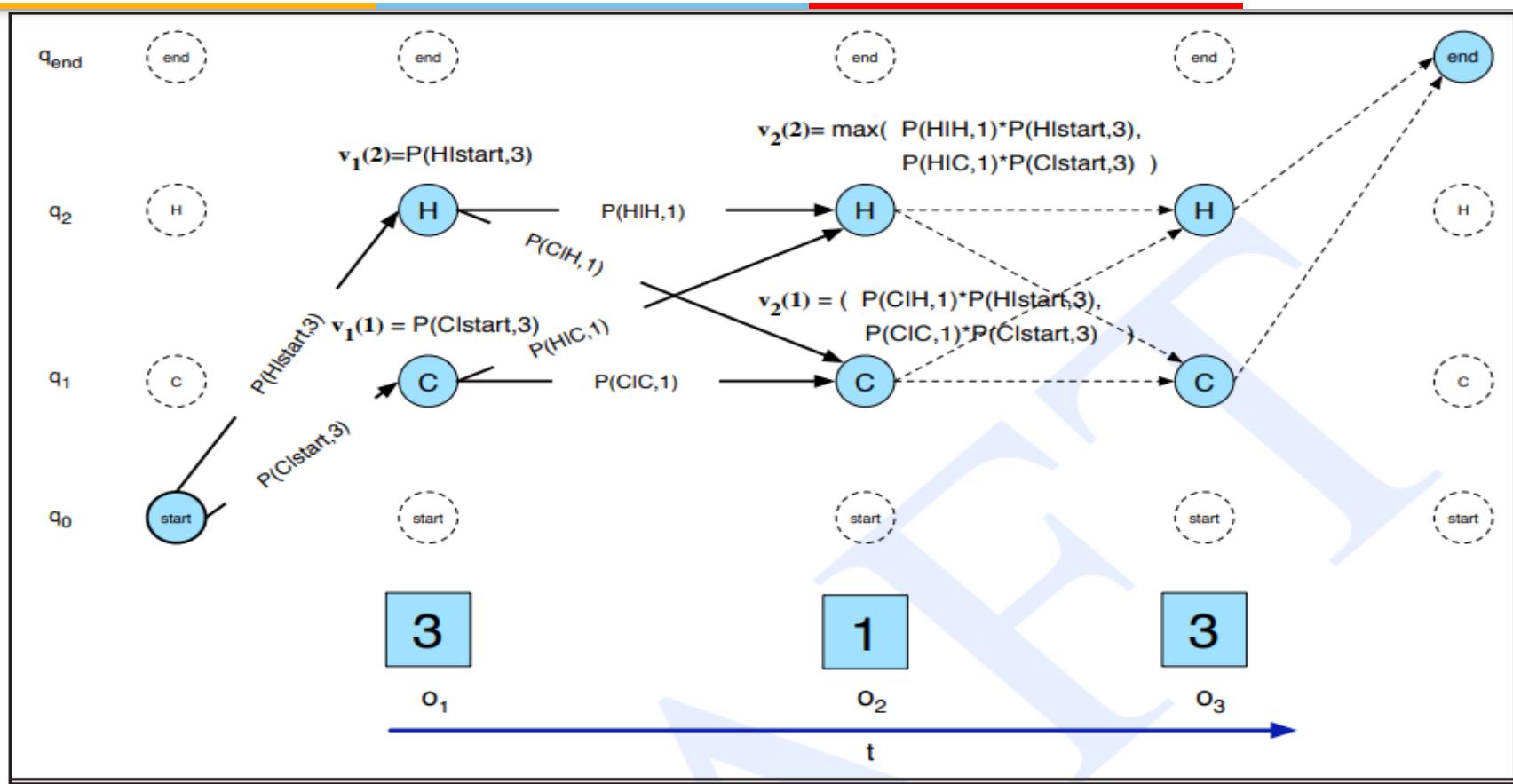


Figure 6.22 Inference from ice-cream eating computed by an MEMM instead of an HMM. The Viterbi trellis for computing the best path through the hidden state space for the ice-cream eating events 3 1 3, modified from the HMM figure in Fig. 6.10.

Source Credit : Speech and Language Processing - Jurafsky and Martin

POS Tagging – Viterbi Algorithm

function VITERBI(*observations* of len T ,*state-graph* of len N) **returns** *best-path*

create a path probability matrix $viterbi[N+2,T]$

for each state s **from** 1 **to** N **do** ; initialization step

$viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$

$backpointer[s,1] \leftarrow 0$

for each time step t **from** 2 **to** T **do** ; recursion step

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

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

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

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

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

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

Modified for Maximum Entropy Markov Model



POS Tagging – Viterbi Algorithm

function VITERBI(*observations* of len T ,*state-graph* of len N) **returns** *best-path*

create a path probability matrix $viterbi[N+2,T]$

for each state s **from** 1 **to** N **do** ; initialization step

$viterbi[s,1] \leftarrow P(s | O_0, O_1)$

$backpointer[s,1] \leftarrow 0$

for each time step t **from** 2 **to** T **do** ; recursion step

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

$viterbi[s,t] \leftarrow \max_{s'=1}^N viterbi[s',t-1] * P(s | S', O_t)$

$backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^N viterbi[s',t-1] * P(s | S', O_t)$

$viterbi[q_F, T] \leftarrow \max_{s=1}^N viterbi[s, T] * P(q_f | S, _)$; termination step

$backpointer[q_F, T] \leftarrow \operatorname{argmax}_{s=1}^N viterbi[s, T] * P(q_f | S, _)$; termination step

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

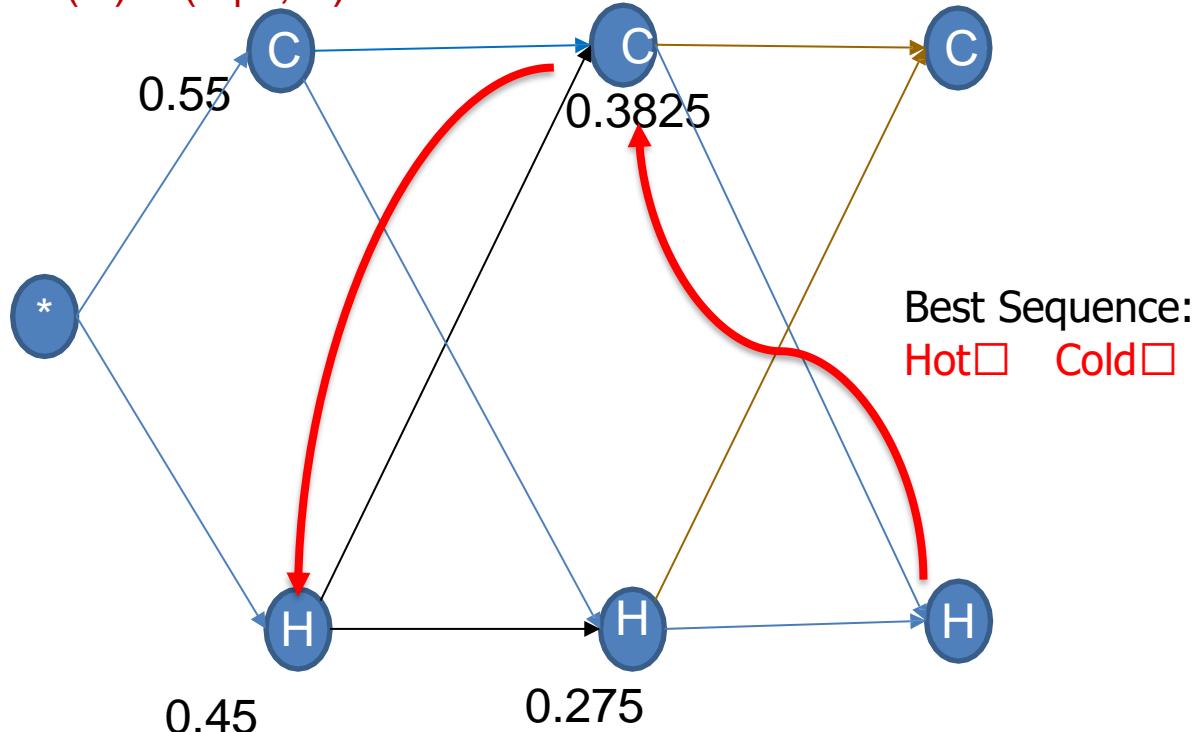
Source Credit : Speech and Language Processing - Jurafsky and Martin

Maximum Entropy Markov Model

Example -6 – Viterbi Algorithm – Termination through Back Trace

$$P(C) * P(C|C, 3) = 0.3825 * 0.2 = 0.0765$$

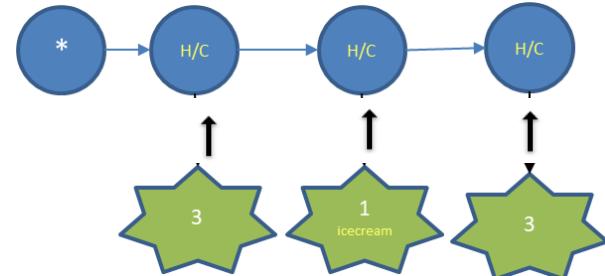
$$P(H) * P(C|H, 3) = 0.275 * 0.4 = 0.11$$



Best Sequence:
Hot Cold

$$P(C) * P(H|C, 3) = 0.3825 * 0.8 = 0.306$$

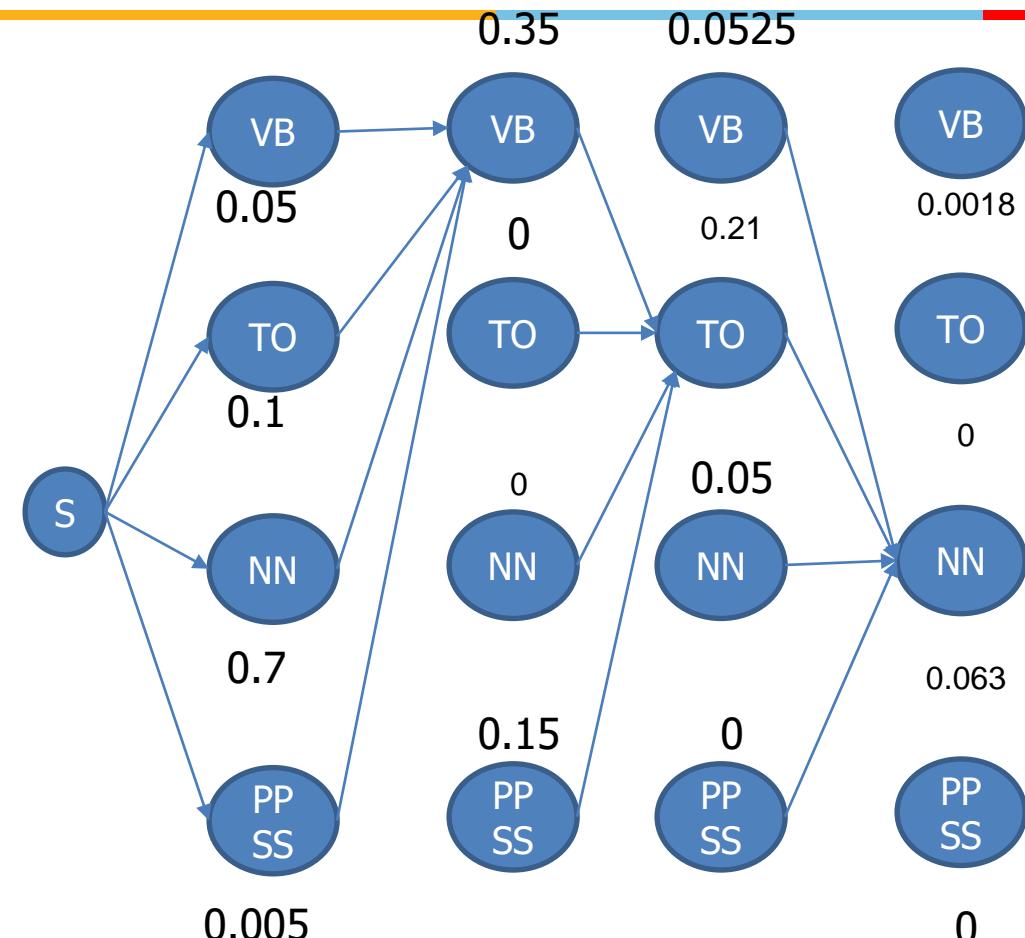
$$P(H) * P(H|H, 3) = 0.275 * 0.6 = 0.165$$



$X_1 = T_{i-1}$	$X_2 = w_i$	Label $Y = T_i$	$P(T_i / T_{i-1} W_i)$
Start	1	Hot	0.3
Start	1	Cold	
Start	3	Hot	0.45
Start	3	Cold	
.....
Hot	1	Hot	0.15
Hot	1	Cold	
Hot	3	Hot	0.6
Hot	3	Cold	
.....
Cold	1	Hot	0.5
Cold	1	Cold	
Cold	3	Hot	0.8
Cold	3	Cold	
....

Maximum Entropy Markov Model

Example -7 – Viterbi Algorithm

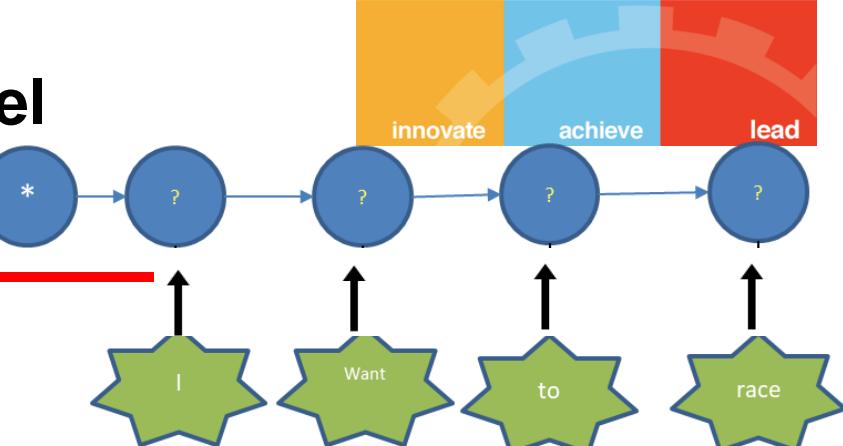


$$P(VB) * P(VB | VB, want) = 0.05 * 0.3 = 0.015$$

$$P(TO) * P(VB | TO, want) = 0.1 * 0.5 = 0.05$$

$$P(NN) * P(VB | NN, want) = 0.7 * 0.5 = 0.35$$

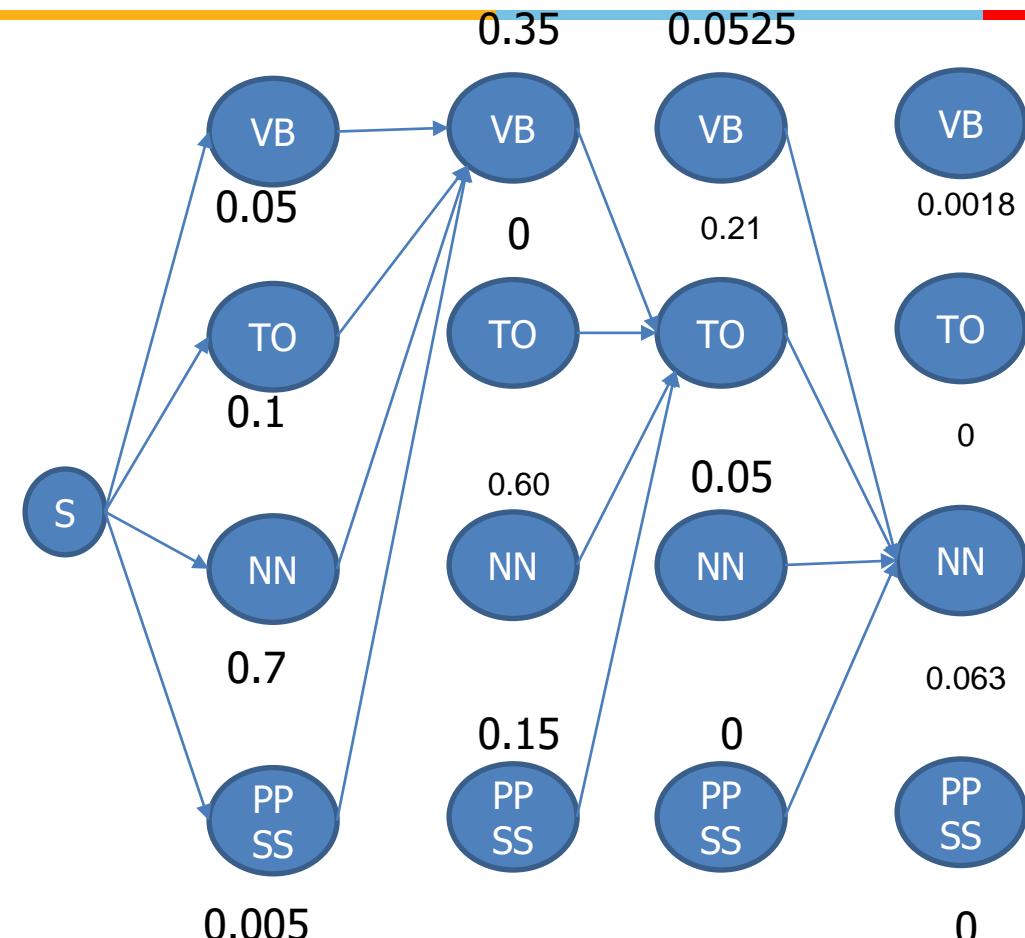
$$P(PPSS) * P(VB | PPSS, want) = 0.005 * 0.5 = 0.0025$$



$X_1 = T_{i-1}$	$X_2 = w_i$	Label $Y = T_i$	$P(T_i / T_{i-1} W_i)$
Start	I	NN	0.7
Start	I	TO	0.1
Start	I	VB	0.05
Start	I	PPSS	0.005
Start	The	NN	0.01
....
NN	want	NN	0.15
NN	want	VB	0.5
VB	want	VB	0.3
VB	to	TO	0.6
VB	to	NN	0.2
....
TO	race	NN	0.3
TO	race	VB	0.7
TO	race	TO	0
....
....

Maximum Entropy Markov Model

Example -7 – Viterbi Algorithm

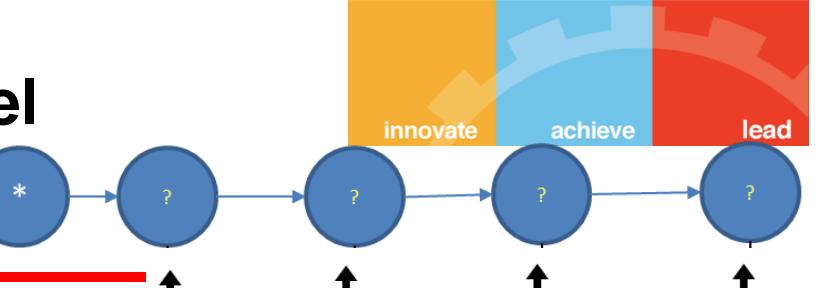


$$P(\text{VB}) * P(\text{TO} | \text{VB}, \text{to}) = 0.35 * 0.15 = 0.0525$$

$$P(\text{TO}) * P(\text{TO} | \text{TO}, \text{to}) = 0 * 0 = 0$$

$$P(\text{NN}) * P(\text{TO} | \text{NN}, \text{to}) = 0.6 * 0 = 0$$

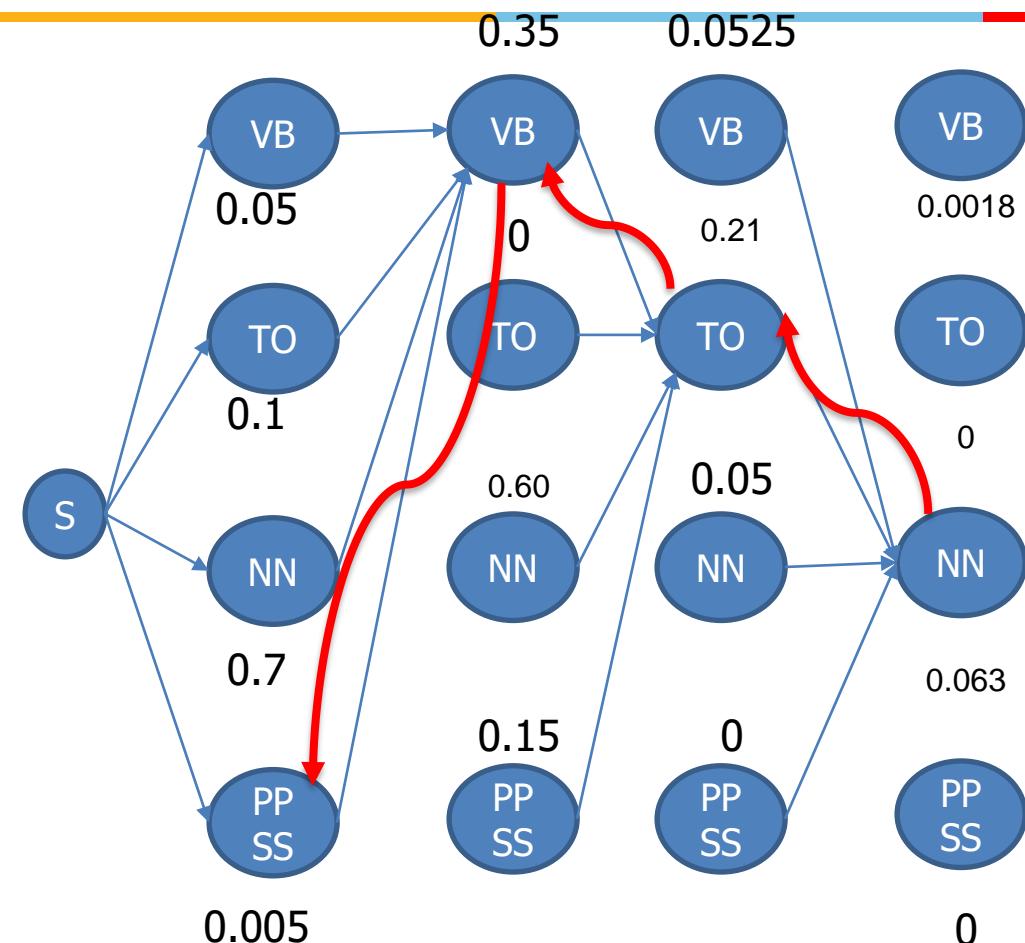
$$P(\text{PPSS}) * P(\text{TO} | \text{PPSS}, \text{to}) = 0.15 * 0 = 0$$



$X_1 = T_{i-1}$	$X_2 = w_i$	Label $Y = T_i$	$P(T_i T_{i-1}, W_i)$
Start	I	NN	0.7
Start	I	TO	0.1
Start	I	VB	0.05
Start	I	PPSS	0.005
Start	The	NN	0.01
....
NN	want	NN	0.15
NN	want	VB	0.5
VB	want	VB	0.3
VB	to	TO	0.6
VB	to	NN	0.2
....
TO	race	NN	0.3
TO	race	VB	0.7
VB	race	TO	0
....
....

Maximum Entropy Markov Model

Example -7 – Viterbi Algorithm

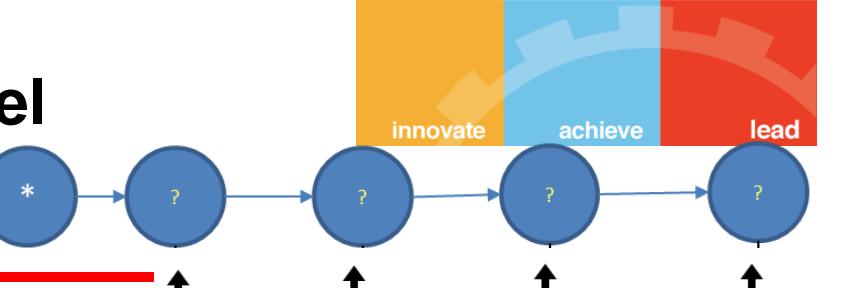


$$P(VB) * P(NN | VB, race) = 0.0525 * 0.1 = 0.00525$$

$$P(TO) * P(NN | TO, race) = 0.21 * 0.3 = 0.063$$

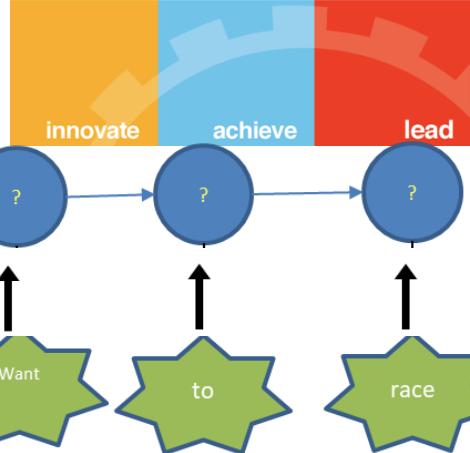
$$P(NN) * P(NN | NN, race) = 0.05 * 0.005 = 0.00025$$

$$P(PPSS) * P(NN | PPSS, race) = 0 * 0 = 0$$



$X_1 = T_{i-1}$	$X_2 = w_i$	Label $Y = T_i$	$P(T_i / T_{i-1} W_i)$
Start	I	NN	0.7
Start	I	TO	0.1
Start	I	VB	0.05
Start	I	PPSS	0.005
Start	The	NN	0.01
....
NN	want	NN	0.15
NN	want	VB	0.5
VB	want	VB	0.3
VB	to	TO	0.6
VB	to	NN	0.2
....
TO	race	NN	0.3
TO	race	VB	0.7
VB	race	TO	0
....
....

Maximum Entropy Markov Model



Learning the $P(T_i | T_{i-1} W_i)$: Posterior Tags

- Training
Multinomial Logistic Regression – MaxEnt

- Features
 - Engineer Features' design
 - “Secretariat is expected to **race** tomorrow”

NNP VBZ VBN TO VB RB

		f1	f2	f3	f4	f5	f6
VB	f	0	1	0	1	1	0
VB	w		.8		.01	.1	
NN	f	1	0	0	0	0	1
NN	w	.8					-1.3

Figure 6.19 Some sample feature values and weights for tagging the word *race* in (6.81).

$$f_1(c, x) = \begin{cases} 1 & \text{if } word_i = "race" \& c = \text{NN} \\ 0 & \text{otherwise} \end{cases}$$

$$f_2(c, x) = \begin{cases} 1 & \text{if } t_{i-1} = \text{TO} \& c = \text{VB} \\ 0 & \text{otherwise} \end{cases}$$

$$f_3(c, x) = \begin{cases} 1 & \text{if } \text{suffix}(word_i) = "ing" \& c = \text{VBG} \\ 0 & \text{otherwise} \end{cases}$$

$$f_4(c, x) = \begin{cases} 1 & \text{if } \text{is_lower_case}(word_i) \& c = \text{VB} \\ 0 & \text{otherwise} \end{cases}$$

$$f_5(c, x) = \begin{cases} 1 & \text{if } word_i = "race" \& c = \text{VB} \\ 0 & \text{otherwise} \end{cases}$$

$$f_6(c, x) = \begin{cases} 1 & \text{if } t_{i-1} = \text{TO} \& c = \text{NN} \\ 0 & \text{otherwise} \end{cases}$$

$$f_{\text{example}}(x^{(*)}, t, y_i) := \begin{cases} 1 & x^{(t-1)} = "the" \wedge y_i = \text{NN} \\ 0 & \text{otherwise} \end{cases}$$

Source Credit : Speech and Language Processing - Jurafsky and Martin

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

Maximum Entropy Markov Model

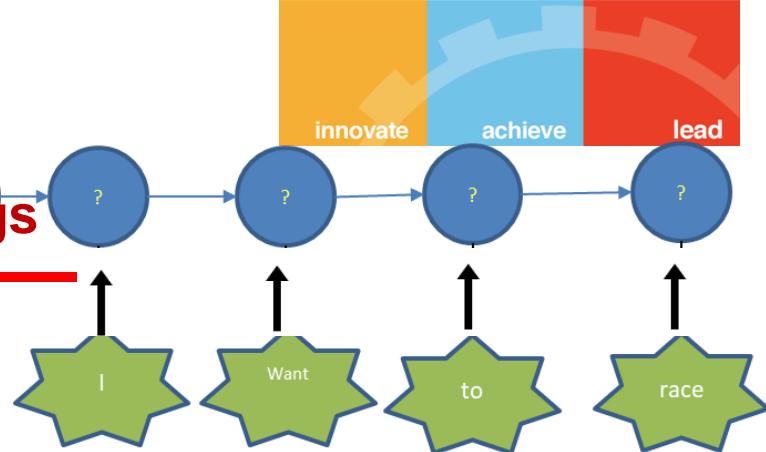
Learning the $P(T_i | T_{i-1} W_i)$: Posterior Tags

- Training
Multinomial Logistic Regression – MaxEnt

- Features
 - Engineer Features' design
 - “Secretariat is expected to **race** tomorrow”
NNP VBZ VBN TO VB RB

- Model : Log Linear / Logistic Regression Equation
- In Practice : Complex Features involving x is designed

$$f_{125}(c, x) = \begin{cases} 1 & \text{if } word_{i-1} = <\text{s}> \text{ & } \text{isupperfirst}(word_i) \text{ & } c = \text{NNP} \\ 0 & \text{otherwise} \end{cases}$$



$$P(NN|x) = \frac{e^{.8}e^{-1.3}}{e^{.8}e^{-1.3} + e^{.8}e^{.01}e^{.1}} = .20$$

$$P(VB|x) = \frac{e^{.8}e^{.01}e^{.1}}{e^{.8}e^{-1.3} + e^{.8}e^{.01}e^{.1}} = .80$$

Source Credit : Speech and Language Processing - Jurafsky and Martin

Maximum Entropy Markov Model

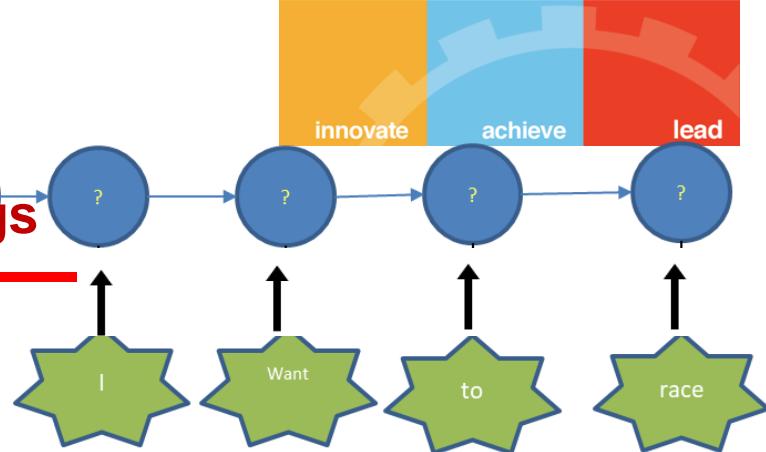
Learning the $P(T_i | T_{i-1} W_i)$: Posterior Tags

- Training
Multinomial Logistic Regression – MaxEnt

- Features
 - Engineer Features' design
 - “Secretariat is expected to **race** tomorrow”
- | | | | | | |
|-----|-----|-----|----|----|----|
| NNP | VBZ | VBN | TO | VB | RB |
|-----|-----|-----|----|----|----|

- Model : Log Linear / Logistic Regression Equation

$$p(c|x) = \frac{\exp\left(\sum_{i=0}^N w_{ci} f_i(c, x)\right)}{\sum_{c' \in C} \exp\left(\sum_{i=0}^N w_{c'i} f_i(c', x)\right)}$$



$$P(NN|x) = \frac{e^8 e^{-1.3}}{e^{-8} e^{-1.3} + e^8 e^{.01} e^{-.1}} = .20$$

$$P(VB|x) = \frac{e^{-8} e^{.01} e^{-.1}}{e^{-8} e^{-1.3} + e^8 e^{.01} e^{-.1}} = .80$$

		f1	f2	f3	f4	f5	f6
VB	f	0	1	0	1	1	0
VB	w		.8		.01	.1	
NN	f	1	0	0	0	0	1
NN	w	.8					-1.3

Figure 6.19 Some sample feature values and weights for tagging the word *race* in (6.81).

In boolean logistic regression, classification involves building one linear expression which separates the observations in the class from the observations not in the class. Classification in MaxEnt, by contrast, involves building a separate linear expression for each of C classes

Source Credit : Speech and Language Processing - Jurafsky and Martin

Maximum Entropy Markov Model

Learning the W_c : Weights not words

- Training
Multinomial Logistic Regression – MaxEnt

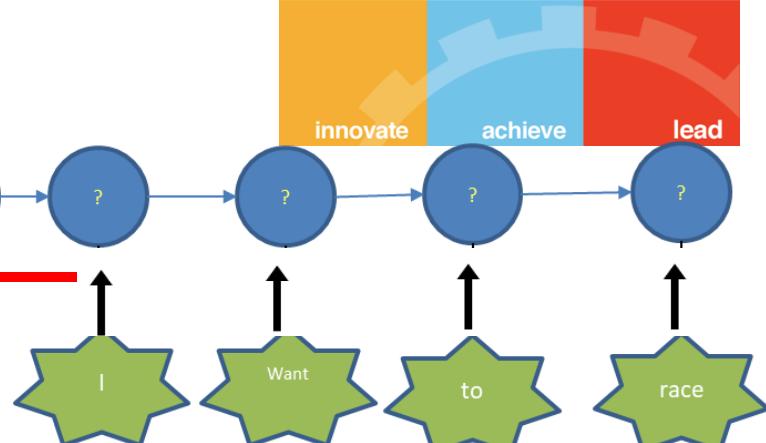
- Features
 - Engineer Features' design
 - “Secretariat is expected to **race** tomorrow”

NNP VBZ VBN TO VB RB

- Model : Log Linear / Logistic Regression Equation
- Learn the weights (with or without Regularization)
 - **Gradient Descent**

$$\hat{w} = \operatorname{argmax}_w \sum_i \log P(y^{(i)} | x^{(i)}) - \sum_{j=1}^N \frac{w_j^2}{2\sigma_j^2}$$

$$\hat{w} = \operatorname{argmax}_w \sum_i \log P(y^{(i)} | x^{(i)}) - \alpha \sum_{j=1}^N w_j^2$$



$X_1 = T_{i-1}$	$X_2 = w_i$	Label $Y = T_i$	$P(T_i / T_{i-1} W_i)$
Start	I	NN	0.7
Start	I	TO	0.1
Start	I	VB	0.05
Start	I	PPSS	0.005
Start	The	NN	0.01
.....
NN	want	NN	0.15
NN	want	VB	0.5
VB	want	VB	0.3
VB	to	TO	0.6
VB	to	NN	0.2
.....
TO	race	NN	0.3
TO	race	VB	0.7
VB	race	TO	0
....
....



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Neural Language Model

Dr. S. Prabakeran

Teaching Assistant
Birla Institute of Technology & Science, Pilani

Agenda

- Neural Language Models
- How Large Language Models Work
- Feedforward Neural Network Basics
- Transformers for beginners - What are they and how do they work?
- Inside the Transformer Architecture
- How a Transformer works in machine translation
- Summary
- Demo Session

Neural Language Models

- Introduction to Neural Language Models
- Applications of Neural Language Models
- Input Representations in Neural Language Models
- Structure of a Simple Neural Language Model
- Pretraining and Learning Word Embeddings
- Embedding Matrix and Model Optimization
- Visualization of Neural Language Model with Embedding Layer

How Large Language Models Work

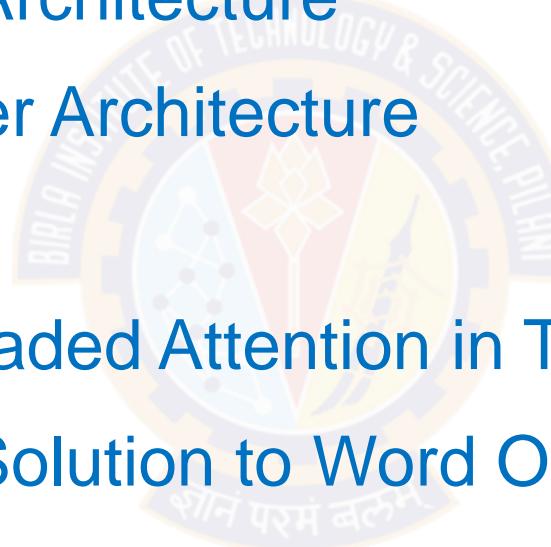
- Introduction to GPT
- Large Language Models and Foundation Models
- Scale of Text Data and Model Parameters
- Components of Large Language Models
- Training Process of Large Language Models
- Fine-Tuning for Specific Tasks
- Business Applications of Large Language Models

Feedforward Neural Network Basics

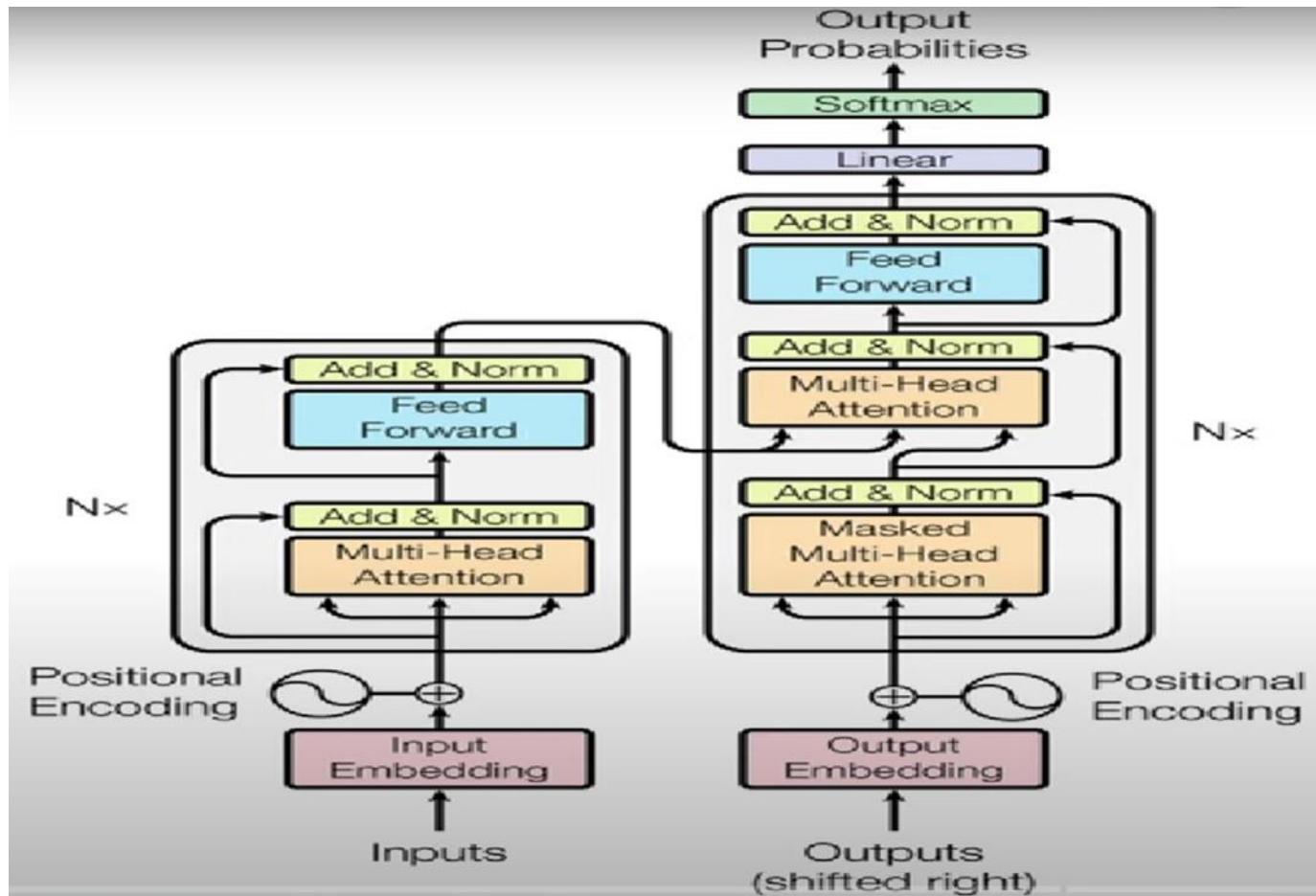
- Intuition Behind Feedforward Neural Network Design
- Structure of Feedforward Neural Networks
- General Flow of a Feedforward Neural Network
- Prediction Process in Feedforward Neural Networks
- Relationship Between Neural Networks and Deep Learning
- Differentiating Neural Networks in Depth
- Power of Neural Networks Over Traditional Models
- Feature Usage in Neural Networks

Transformers for beginners - What are they and how do they work ?

- The Rise of Transformers in NLP
- Inside the Transformer Architecture
- Key Ideas in Transformer Architecture
- Multi-Headed Attention
- Understanding Multi-Headed Attention in Transformers
- Positional Encoding: A Solution to Word Order Ambiguity
- Bringing It All Together: How Transformers Operate
- Final Thoughts on Transformers: Simplicity in Complexity



Inside the Transformer Architecture



Inside the Transformer Architecture...

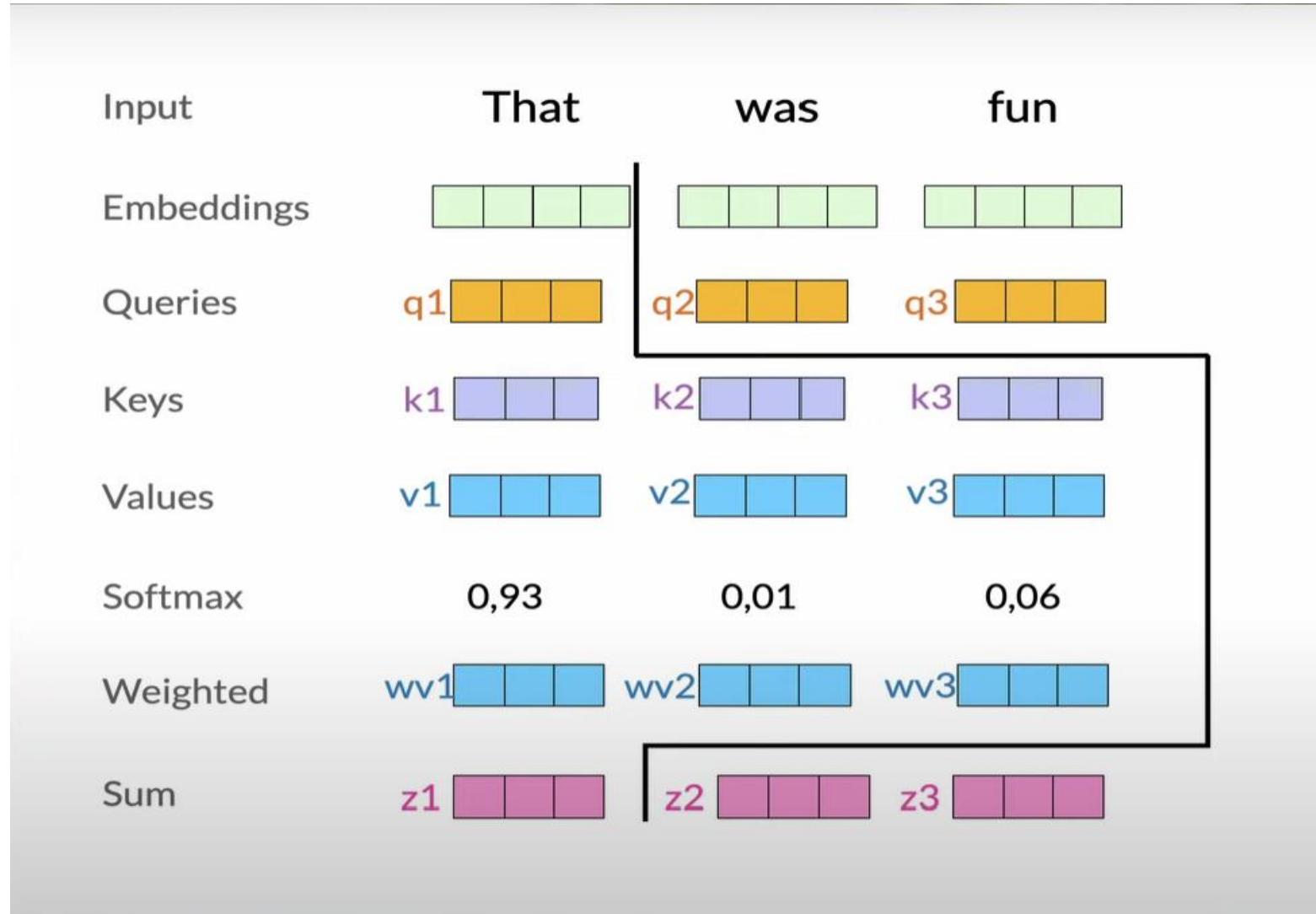
- Input Embedding
- Positional Encoding
- Multi-Head Attention: Query, Key-Value, Attention, Weighted Sum
- Add & Norm
- Feed Forward
- Output Embedding
- SoftMax



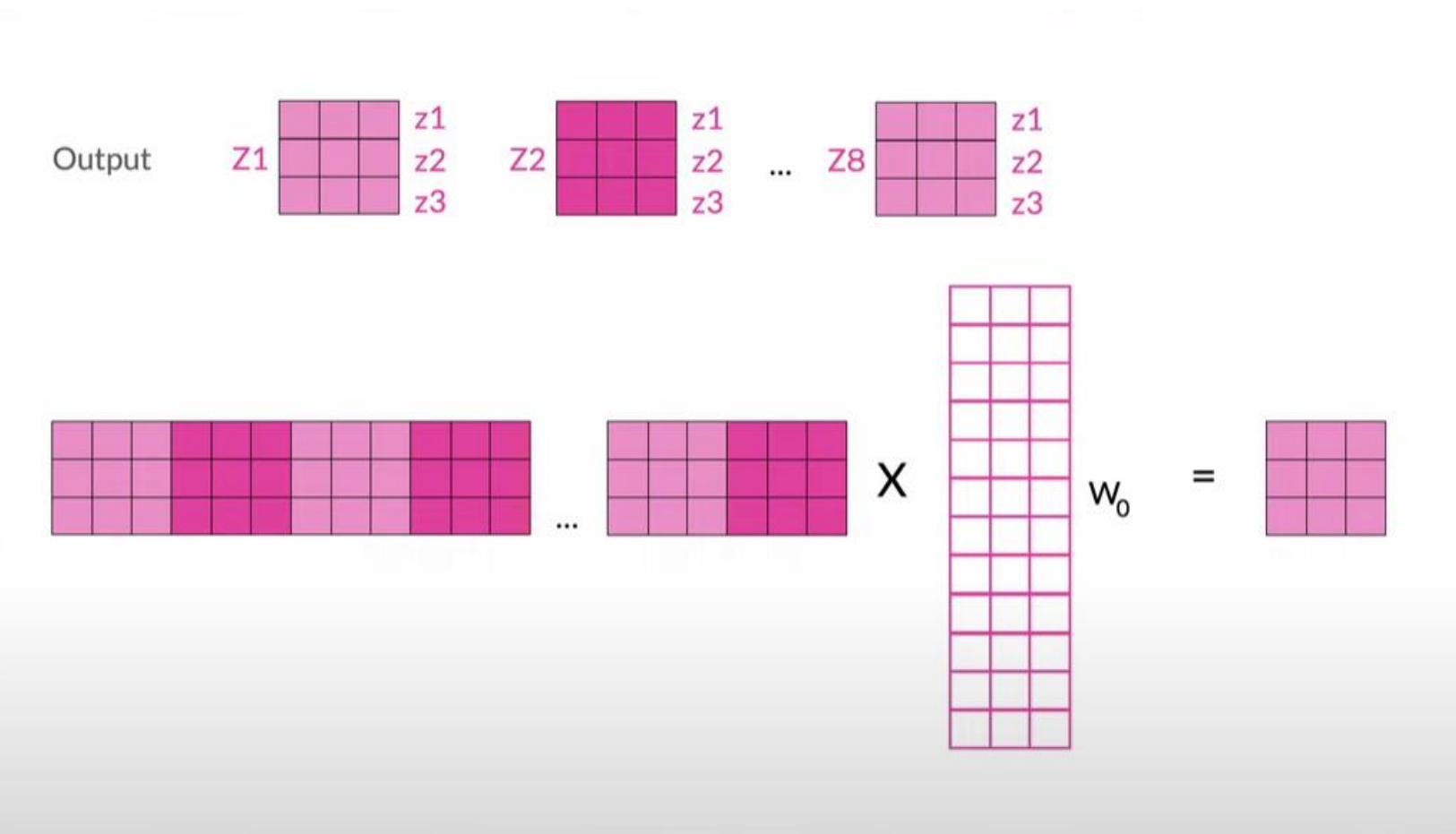
Multi-Head Attention: Query, Key-Value, Attention, Weighted Sum...

Input	That	was	fun
Embeddings			
Queries			
Keys			
Values			
Score	$q1 \bullet k1 = 98$	$q1 \bullet k2 = 46$	$q1 \bullet k3 = 76$
Divide by 8	12,25	5,75	9,5
Softmax	0,93	0,01	0,06

Multi-Head Attention: Query, Key-Value, Attention, Weighted Sum...



Multi-Head Attention: Query, Key-Value, Attention, Weighted Sum...



How a Transformer works in machine translation ?

Translate the English sentence "The cat sat on the mat" into French.

Input Embedding:

- Each word in the English sentence is converted into a numerical representation (embedding).
- "The" becomes a vector of numbers, "cat" becomes a different vector, and so on.

Positional Encoding:

- Information about the word order is added to the embeddings.
- This helps the model understand that "cat" comes before "mat" in the sentence.

Multi-Head Attention:

- The model focuses on the relationships between different words in the sentence.
- It learns to pay more attention to words that are closely related, like "cat" and "mat."

How a Transformer works in machine translation ?...

Translate the English sentence "The cat sat on the mat" into French.

Feed Forward:

- The model applies additional processing to the information it has gathered.
- This helps it make more complex decisions about how to translate the sentence.

Output:

- The model generates the French translation, "Le chat s'est assis sur le tapis."

DEMO Session on Machine Translation



DEMO Session on Text Classification using Neural Networks



Thank you

