



Text Mining

Martial Luyts

Catholic University of Leuven, Belgium

martial.luyts@kuleuven.be



LEUVEN STATISTICS
RESEARCH CENTRE

Contents

1. Introduction	1
1.1 Introductory material	2
1.2 What is Text Mining	6
1.3 Comparison with other fields	9
1.4 Some text mining applications	13
2. Text mining process	19
2.1 General overview	20
2.2 Text Pre-processing	21

2.2.1. Text cleanup	22
2.2.2. Tokenization	23
2.2.3. Filtering	27
2.2.4. Stemming & lemmatization	30
2.2.5. Chunking & parsing	95
2.2.6. Semantics	117
2.3 Attribute Generation	120
2.3.1. One-hot encoding	123
2.3.2. Bag of Words	126
2.3.3. Term Frequency - Inverse Document Frequency	132
2.3.4. Pointwise Mutual Information	140
2.3.5. Word embeddings	150

2.4 Attribute Selection	172
2.5 Text Mining Techniques	173
2.5.1. Language Models	174
2.5.2. Text classification	261
REFERENCES	301

Part 1:

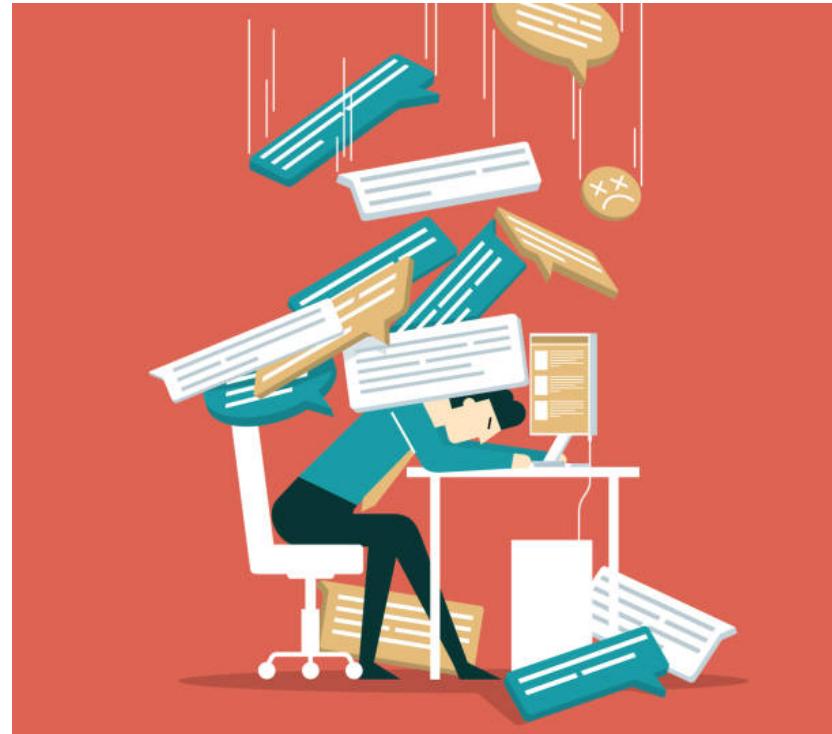
Introduction

1. Introductory material

- Nowadays, we have (and still produce) an extensive amount of electronic books, documents, web pages, emails, blogs, news, chats, memos, research papers, ...
- These subjects/items are immediately accessible, thanks to databases and **Information Retrieval (IR)**
 - ▷ **Information Retrieval (IR):** Process of obtaining information system resources (usually documents) that are relevant to an information need from a collection of those resources. Web search engines are the most visible IR applications.



- **Challenge:** Approximately 80% of all data stored in databases are **natural language texts**.



- Analyze the texts to identify and structure relevant information are extremely relevant.
 - ▷ This task is called **Natural Language Processing (NLP)**.
 - ▷ **Natural Language Processing (NLP):** One of the oldest and most challenging problems in the field of artificial intelligence. It is the study of human language so that computers can understand natural languages as humans do.

- ▷ NLP research pursues the vague question of how we understand the meaning of a sentence or a document. What are the indications we use to understand who did what to whom, or when something happened, or what is fact and what is supposition or prediction? While words - nouns, verbs, adverbs and adjectives - are the building blocks of meaning, it is their correlation to each other within the structure of a sentence in a document, and within the context of what we already know about the world, that provides the true meaning of a text.
- Afterwards, one can discover patterns in the structured information (**Data Mining (DM)**).

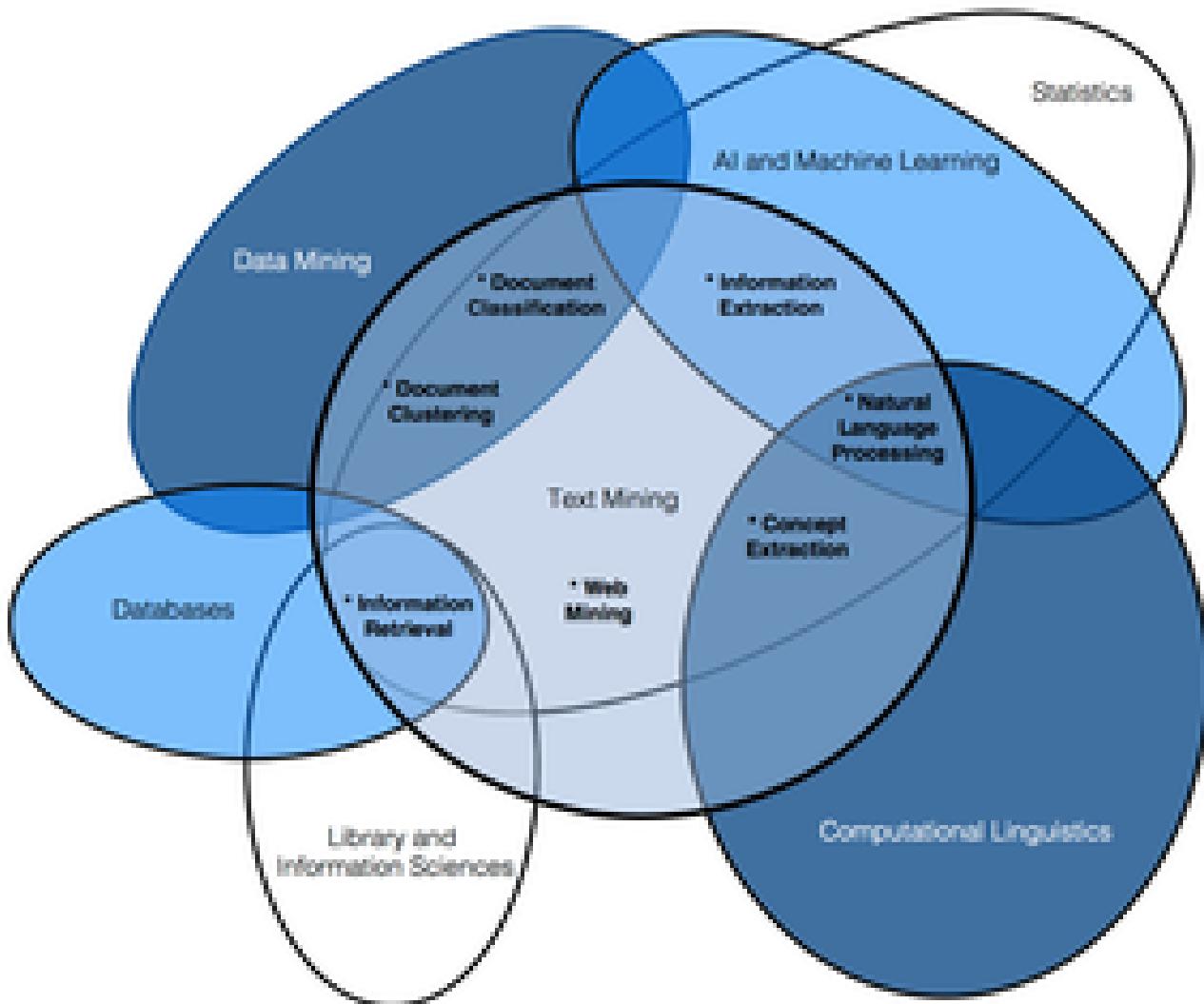
2. What is Text Mining?

In a nutshell:

- Automatic discovery of previously unknown information of high quality in large amounts of mostly unstructured natural language text. The goal is, essentially to turn text (unstructured data) into data (structured format) for analysis, via the use of NLP methods.

Major text mining steps:

- **Information Retrieval (IR)**
- **Information Extraction (IE)**
- **Computational Linguistics (CL) & Natural Language Processing (NLP)**
- **Data & Web Mining**



- Therefore, to derive text mining, one need to have knowledge of many fields such as IE, NLP, IR, etc.
- In what follows, we will make a comparison between text mining and some of these fields

3. Comparison with other fields

1. Text mining vs. IR

- Information retrieval (IR) mostly focused on facilitating information access rather than analyzing information and finding hidden patterns, which is the main purpose of text mining.
- IR has less priority on processing or transformation of text whereas text mining can be considered as going beyond information access to further aid users to analyze and understand information and ease the decision making.

2. Text mining vs. IE

- Information extraction (IE) is the task of automatically extracting information or facts from unstructured or semi-structured document.
- It usually serves as a starting point for other text mining algorithms, e.g., extraction entities, Name Entity Recognition (NER).

3. Text mining vs. CL & NLP

- Computational linguistics (CL) is the scientific discipline concerned with understanding written and spoken language from a computational perspective
- While CL has more of a focus on aspects of language, NLP emphasizes its use of machine learning and deep learning techniques to complete tasks, like language translation or question answering.
- Many of the text mining algorithms extensively make use of NLP techniques, such as part of speech tagging (POG), syntactic parsing and other types of linguistic analysis. Therefore, one can say that NLP is an important aspect in text mining!

4. Text mining vs. data mining

- Text mining is similar to data mining, except that data mining tools are designed to handle structured data from databases, but text mining can also work with unstructured or semi-structured data sets such as emails, text documents and HTML files etc. As a result, text mining is a far better solution.

5. Text mining vs. web mining

- In text mining, the input is commonly free unstructured text, whilst web-mining limits itself to web sources, i.e., structured or semi-structured of nature.

4. Some text mining applications

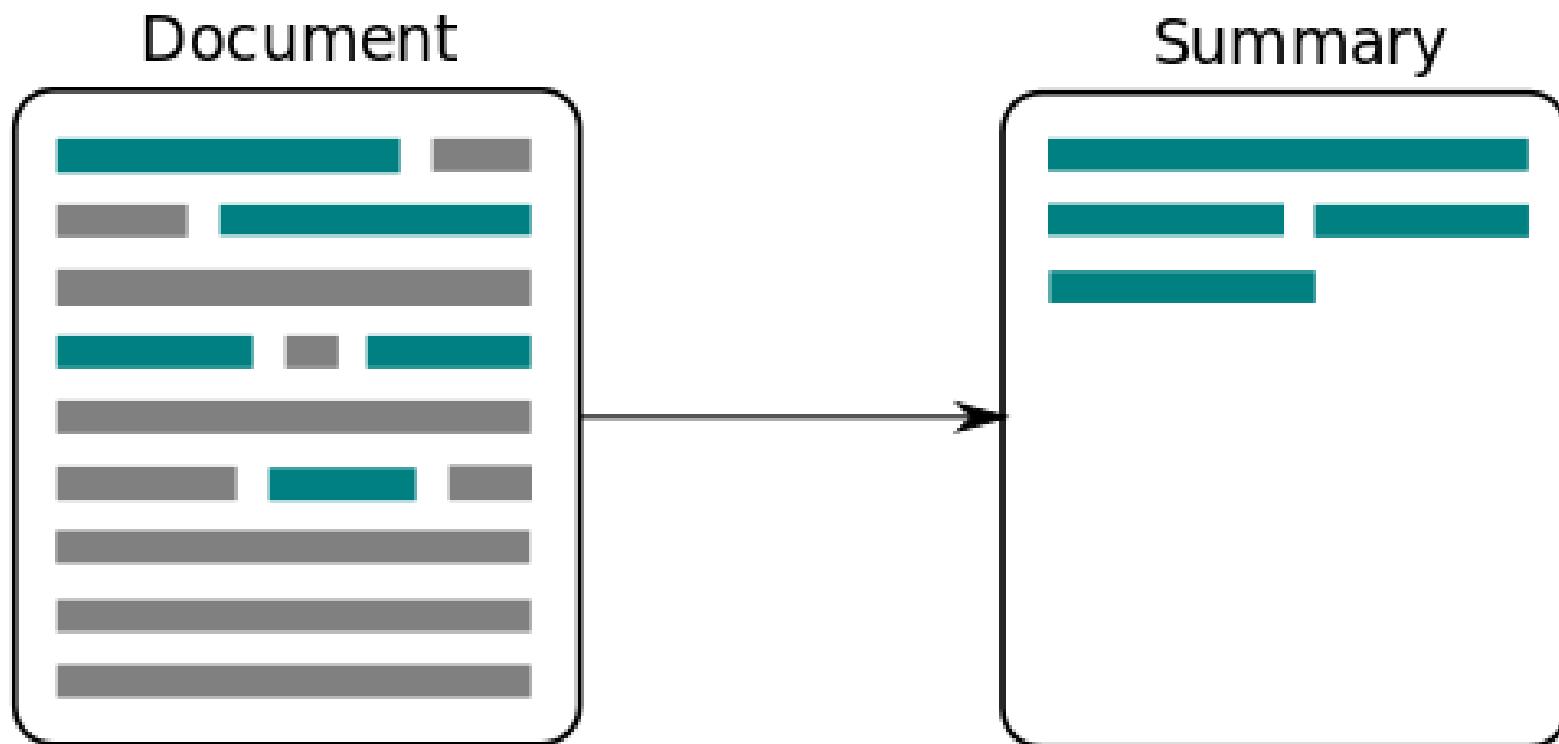
- Classification of news stories

The screenshot shows the Google News homepage. At the top, there is a navigation bar with tabs for Home, For you, Following, U.S., World, Local, Business, Technology, Entertainment, Sports, Science, and Health. A search bar is located above the main content area. Below the search bar, a section titled "Your briefing" displays the date as Wednesday, February 8. On the left, a "Top stories" section features an image from CNN showing Biden at his desk, with the headline "Takeaways from Biden's State of the Union address" and a timestamp of "1 hour ago". To the right, three news articles are listed: one from The Hill, one from The Guardian, and one from USA TODAY, all related to Biden's State of the Union address. On the far right, there is a "Picks for you" section with a "Sign in" button and a weather widget showing "7°C".

- Email and news filtering / SPAM detection



- Automatic text summarization

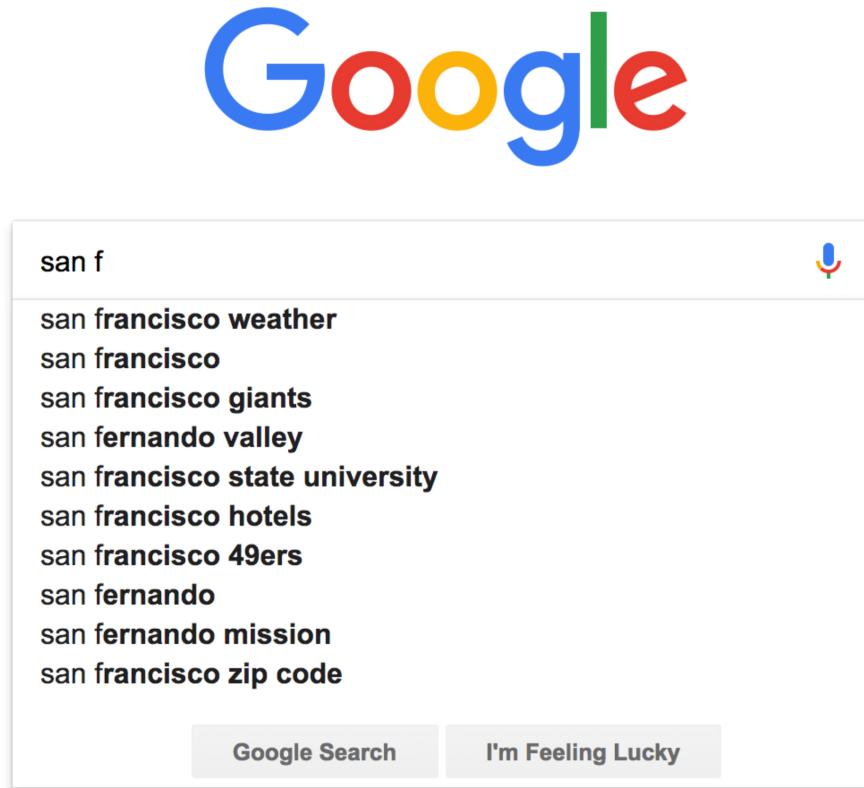


- Sentiment analysis

Out[16]:

		sentence	sentiment
0	Judging from previous posts this used to be a ...		negative
1	We, there were four of us, arrived at noon - t...		negative
2	They never brought us complimentary noodles, i...		negative
3	The food was lousy - too sweet or too salty an...		negative
4	The food was lousy - too sweet or too salty an...		negative

- Search term auto-completion



- Chatbots

The screenshot shows a mobile browser window for chat.openai.com. At the top, there are navigation icons for 'Done' (blue), a lock icon, the URL, a font size 'AA' (blue), and a refresh/circular arrow icon (blue). Below the header is a menu icon (three horizontal lines). The main conversation area starts with a user message from a profile picture of a person with glasses: "Explain zero point energy but in the style of a cat". To the right of this message is a small edit icon. The AI response, preceded by its own profile picture (a green square with a white cat icon), is: "Meow, meow, meow, meow! Zero point energy is like the purr-fect amount of energy that is always present, even in the most still and peaceful moments. It's like the energy that allows us cats to always be ready to pounce and play, even when we're lounging around the house. It's the energy that helps us maintain our balance and poise, no matter what. Meow!" Below the AI message is a page navigation bar showing '2 / 2'. At the bottom of the message card are like and dislike buttons.

Done **chat.openai.com** AA ⏪

≡

 Explain zero point energy but in the style of a cat

 Meow, meow, meow, meow! Zero point energy is like the purr-fect amount of energy that is always present, even in the most still and peaceful moments. It's like the energy that allows us cats to always be ready to pounce and play, even when we're lounging around the house. It's the energy that helps us maintain our balance and poise, no matter what. Meow!

< 2 / 2 >

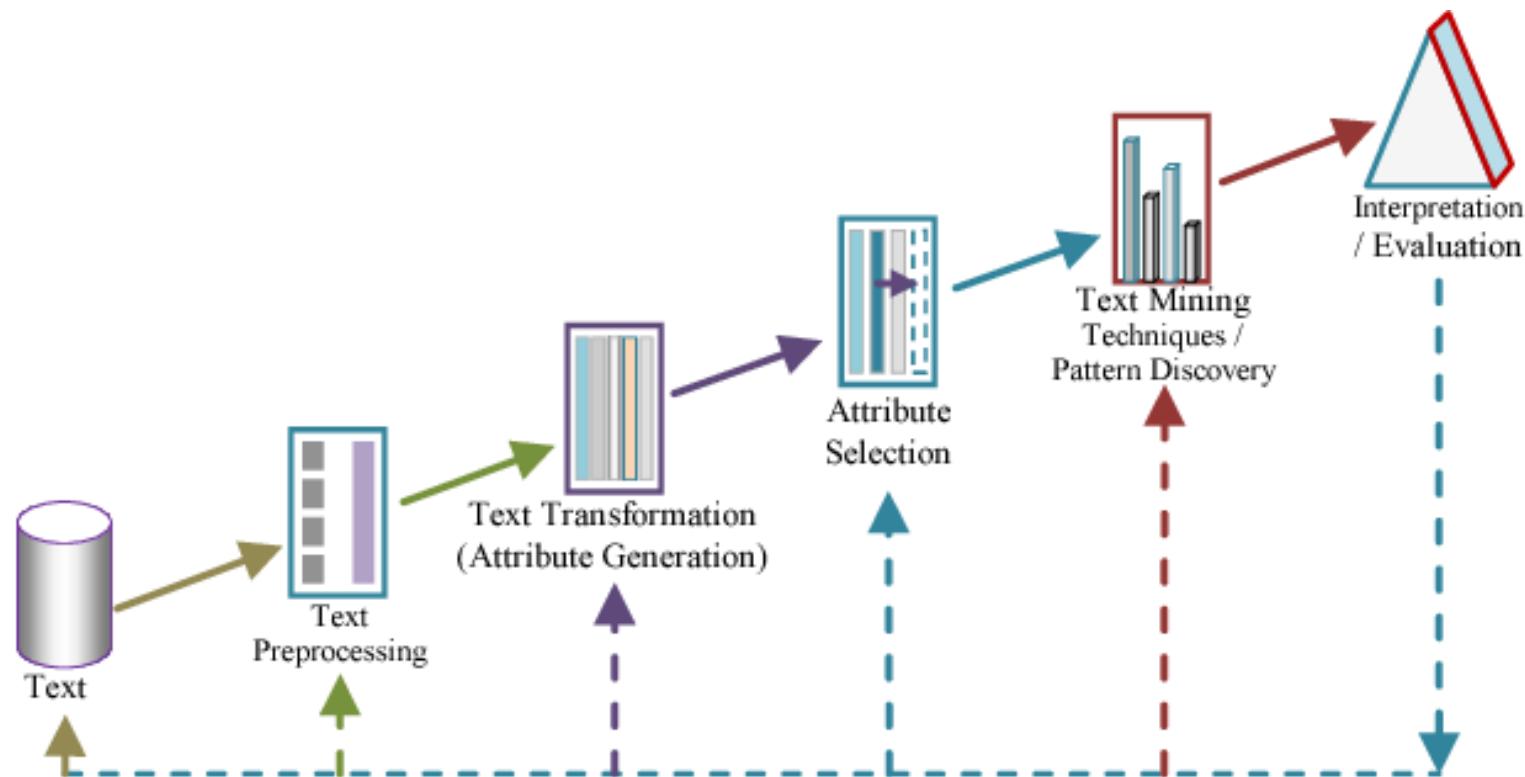
Like Dislike

Part 2:

Text mining process

1. General overview

Text mining involves a series of activities to be performed in order to efficiently mine the information. These activities are:



2. Text Pre-processing

- Text pre-processing is one of the key components in many text mining algorithms.
- It is applied on a (collection of) document(s) containing unstructured or semi-structure data.
- **Task:** Converting a raw text file into a well-defined sequence of linguistically-meaningful units
- To do so, different steps are usually performed, i.e., text cleanup, tokenization, filtering (stop word removal), lemmatizaion, stemming.
- In what follows, we will discuss them briefly.

2.1 Text cleanup

It performs tasks such as removal of:

- advertisement from web pages;
- punctuation and HTML tags;
- tables, figures, etc.

2.2 Tokenization

- **Tokenization** is the process of breaking up a given text into units called tokens, based on certain delimiter(s) and/or rule(s).
- These tokens can be characters, subwords (n-gram characters), words, phrases or even whole sentences.
 - **Example 1:** Consider the sentence "Never give up"
 - The most common way of forming tokens is based on white spaces (delimiter).
 - Word tokens: Never-give-up
 - **Example 2:** Consider the word "smarter"
 - Character tokens: s-m-a-r-t-e-r
 - Subword tokens: smart-er

- As tokens are the building blocks of Natural Language, the most common way of processing the raw text happens at the token level.
- Hence, Tokenization is the foremost step while modeling text data.
- A common next step is the preparation of a vocabulary, often based on the obtained tokens in the corpus.
 - Vocabulary refers to the set of unique tokens in the corpus.
 - At character level, this vocabulary often consists of 26 elements, i.e., the letters within the alphabet

- Unfortunately, even tokenization can be difficult
 - **Example:** Consider "John's sick"
 - **Question:** Is "John's sick" one token or two?
 - **Answer:**
 - If one \Rightarrow Problems in parsing (where's the verb?)
 - If two \Rightarrow What do we do with "John's house" ?
- What to do with hypens (e.g., database vs. data-base vs data base)?
- What to do with "C++", "A/C", ":-)", "..."?
- What to do with languages that don't use whitespace (e.g., Chinese)?

- To avoid/reduce these problems, tokenization is often composed on different rules, next to traditional strategies (e.g., specifying delimiters).

Tokenize on
rules

Let 's tokenize ! Is n't this easy ?

Tokenize on
punctuation

Let ' s tokenize ! Isn ' t this easy ?

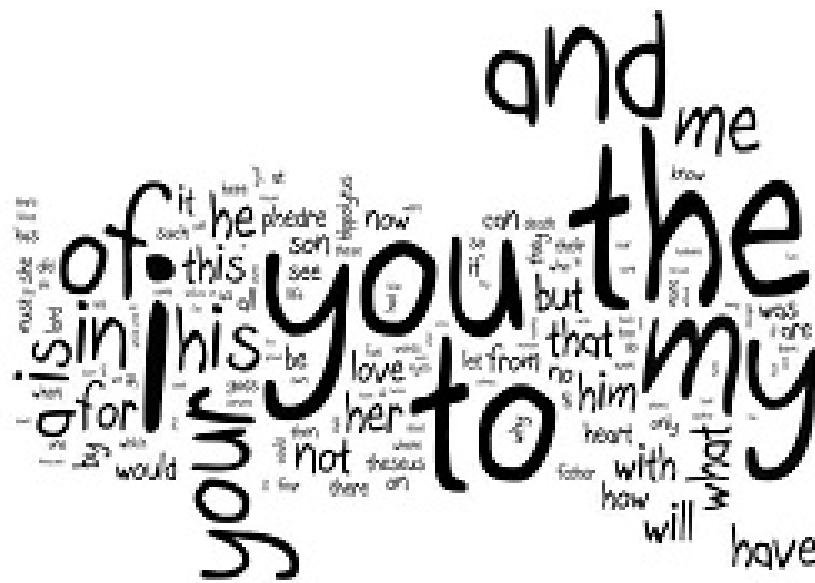
Tokenize on
white spaces

Let's tokenize! Isn't this easy?

Let's tokenize! Isn't this easy?

2.3 Filtering

- Filtering is usually done on text to remove some of the words.
 - A common filtering is stop words removal.
 - Stop words are words that frequently appear in the text without having much content information (e.g., prepositions, conjunctions, etc.)



- Typically text contains about 400 to 500 of such words.

Sample text with Stop Words	Without Stop Words
GeeksforGeeks – A Computer Science Portal for Geeks	GeeksforGeeks , Computer Science, Portal ,Geeks
Can listening be exhausting?	Listening, Exhausting
I like reading, so I read	Like, Reading, read

- For an application, an additional domain specific stopwords list may be constructed.
- **Question:** But why should we remove stopwords?

- **Answer:**

- Reduce data set size (stopwords account for 20-30% of the total word count)
- Improve effectiveness of text mining methods (stopwords may confuse the mining algorithm)

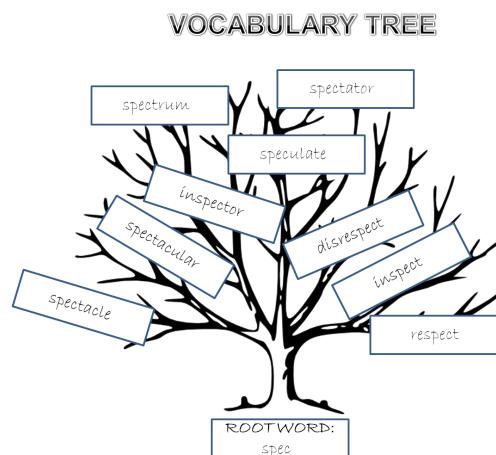


2.4 Stemming & lemmatization

- In addition to tokenization, stemming & lemmatization are the most common text pre-processing techniques in text mining.
- In both cases, these methods try to reduce a given word to its root word.
 - In the stemming process, the root word is called a stem.
 - In the lemmatization process, the root word is called a lemma.
- **Question:** But what is a root word?

- **Answer:**

- Languages such as English, Hindi consists of several words which are often derived from one another.
- Inflected Language is a term used for a language that contains derived words
 - **Example:** The word "historical" is derived from the word "history". Hence, it is a derived word.
- There is always a common root form for all inflected words.

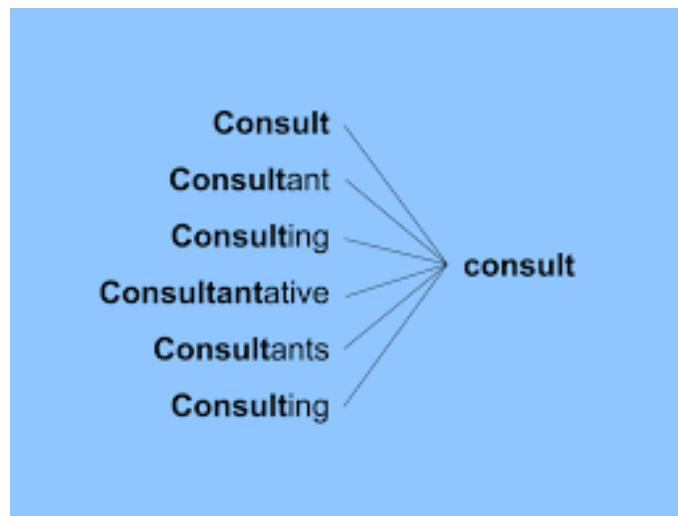


- Depending on the language, the degree of inflection varies from lower to higher.
- Root form of these derived or inflected words are attained using stemming and lemmatization.

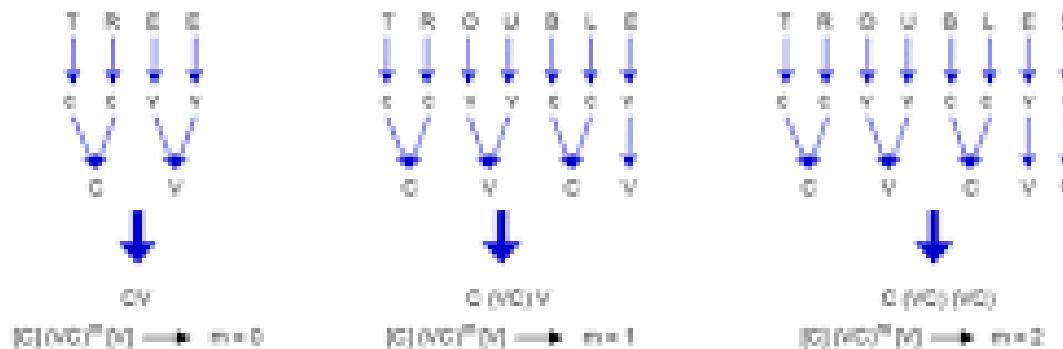
Stemming

- It is the process of removing the last few characters of a given inflected word, to obtain a shorter form, even if that form doesn't have any meaning.

- **Example:**



- Can be achieved with rule-based algorithms, usually based on suffix-stripping
- Standard algorithm for English: the Porter stemmer
 - Overview:



Porter Stemming Algorithm

SSES	\rightarrow	SS	$(m>0)$ ATIONAL	\rightarrow	ATE
IES	\rightarrow	I	$(m>0)$ TIONAL	\rightarrow	TION
SS	\rightarrow	SS	$(m>0)$ ENCI	\rightarrow	ENCE
S	\rightarrow		$(m>0)$ ANCI	\rightarrow	ANCE

- **Advantages:** Simple & fast
- **Disadvantages:**
 - Rules are language dependent;
 - Can create words that do not exist in the language,
e.g., computers ⇒ comput;
 - Often reduces different words to the same stem,
e.g., army, arm ⇒ arm, stocks, stocking ⇒ stock.

- Some basic stemming rules in English: Remove endings & transform words
 - If a word ends with a consonant other than s, followed by an s, then delete s
 - If a word ends in es, drop the s
 - If a word ends in ing, delete the ing unless the remaining word consists only of one letter or of th
 - If a word ends with ies but not eies or aies, then ies -> y
 - ...

Form	Suffix	Stem
studies	-es	studi
study ing	-ing	study
niñ as	-as	niñ
niñ ez	-ez	niñ

Lemmatization:

- It is the process of deriving the linguistic correct root (called lemma) of a word, i.e., base form of the verb, from one of its inflected forms. This requires the morphological analysis of the words, i.e., grouping together the various inflected forms of a word so they can be analyzed as a single item.
- **Morphological analysis:** A field of linguistics that studies the structure of words. It identifies how a word is produced through the use of morphemes. A morpheme is a basic unit of the English language. The morpheme is the smallest element of a word that has grammatical function and meaning.

- **Process:**

- Understand the context
- Determine the Part-of-speech (POS) tagging of a word in a sentence (see later)
- Find the lemma

- **Example:**

Form	Morphological information	Lemma
studies	Third person, singular number, present tense of the verb study	study
studying	Gerund of the verb study	study
niñas	Feminine gender, plural number of the noun niño	niño
niñez	Singular number of the noun niñez	niñez

- **Advantages:**

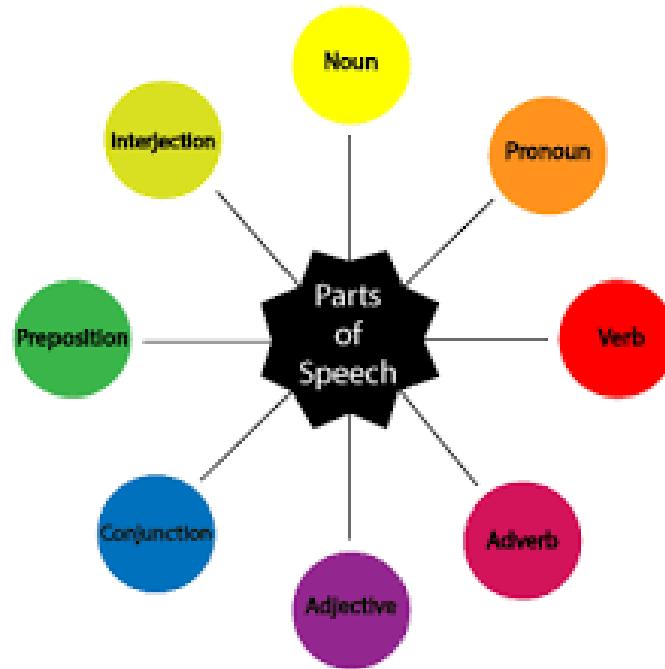
- Identifies the lemma, which is an actual word;
- Less errors than in stemming.

- **Disadvantages:**

- More complex & slower than stemming
- Requires additional language-dependent resources
- While stemming is good enough for IR, TM often requires lemmatization (semantics is more important in this setting)
- **Remark:** Its implementation is difficult because it is related to the semantics and the POS of a sentence, e.g., go is lemma of goes, gone, going, went.
- **Question:** But what is Part-of-speech (POS) tagging?

Part-of-speech (POS) tagging:

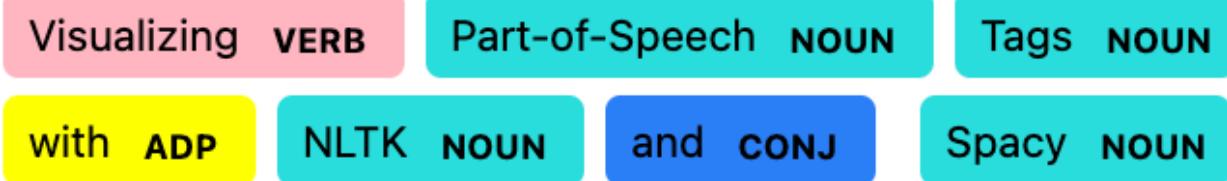
- It determines the linguistic category of word (called tags or parts of speech), and assigns a part of speech to each token.
- In English language, there are eight main tags (which are often collected in a so-called tagset):



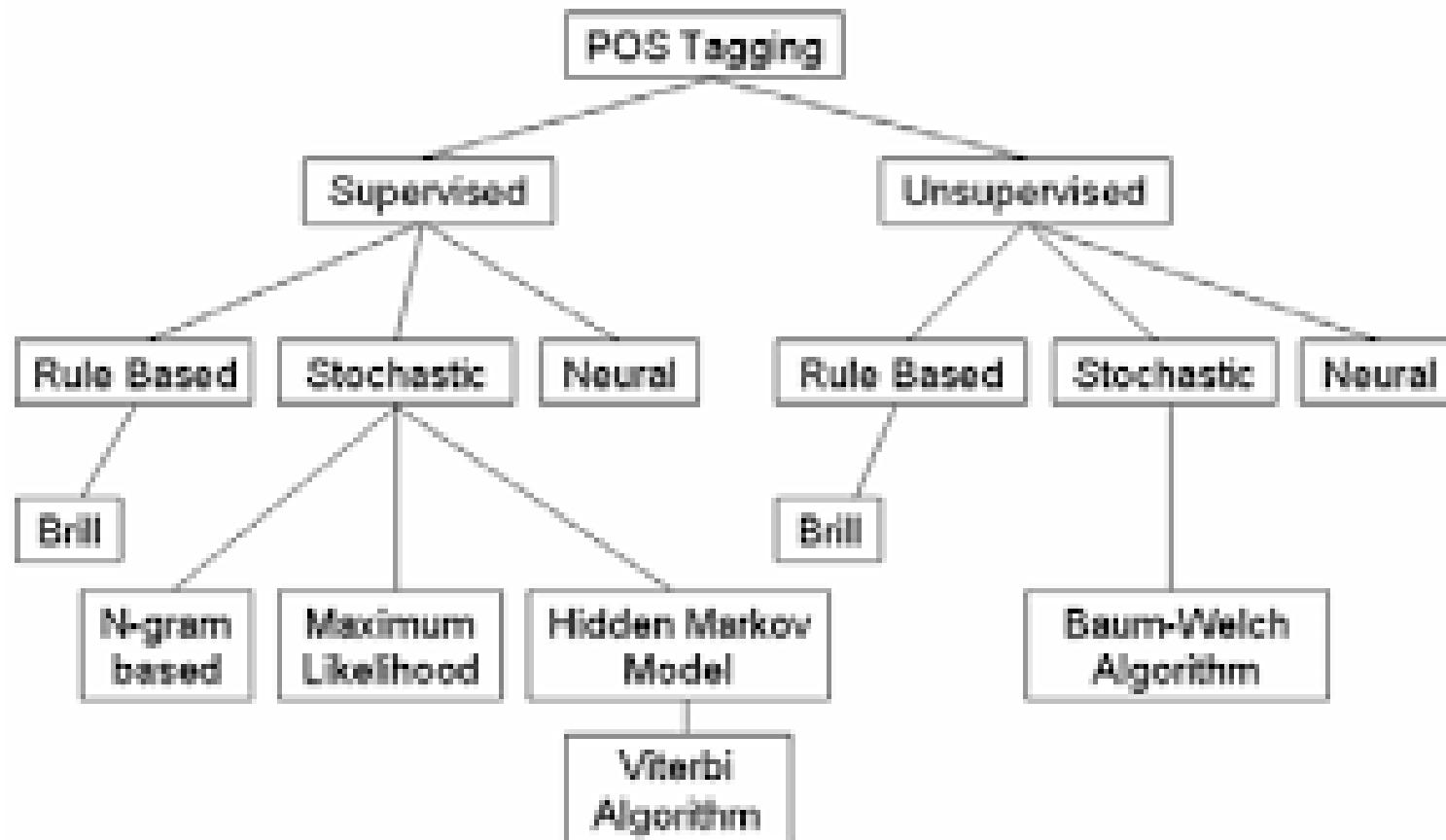
- **noun:** John, London, Table, Teacher, Pen, City, Happiness, Hope
- **pronoun:** I, We, They, You, He, She, It, Me, Us, Them, Him, Her, This
- **adjective:** Big, Happy, Green, Young, Fun, Crazy, Three
- **verb:** Read, Eat, Go, Speak, Run, Play, Live, Have, Like, Are, Is
- **adverb:** Slowly, Quietly, Very, Always, Never, Too, Well, Tomorrow
- **preposition:** At, On, In, From, With, Near, Between, About, Under
- **conjunction:** And, Or, But, Because, So, Yet, Unless, Since, If
- **interjection:** Ouch! Wow! Great! Help! Oh! Hey! Hi!

- Thus, POS tagging can be seen as a learning solution which aims to assign parts of speech tag to each word of a given text based on its context and definition.

- **Example:**



- There are different approaches for POS tagging:



Supervised POS tagging:

- Requires a pre-tagged corpora which is used for training to learn about the tagset, word-tag frequencies, rule sets, etc.
- The performance of these models generally increase with the increase in size of this corpora
- **Examples:**
 - Hidden Markov Models (HMM)
 - Conditional Random Fields (CRF)/Maximum Entropy Markov Models (MEMM)
 - Neural sequence models, e.g., Recurrent Neural Networks (RNN) or Transformers
 - Large Language Models, e.g., BERT, finetuned

Unsupervised POS tagging:

- Do not require a pre-tagged corpora
- Instead, they use advanced computational methods like the Baum-Welch algorithm to automatically induce tagsets, transformation rules, etc.
- In what follows, we will explain a few state-of-the-art techniques for POS tagging, i.e., [2] **Hidden Markov Model (HMM)** with [3] **Viterbi Algorithm**, [4] **Maximum Entropy Markov Model (MEMM)** & [5] **Conditional Random Fields (CRFs)** (all [1] **stochastic taggers**)

[1] Stochastic POS Tagging

- Any POS tagging model which somehow incorporates frequency or probability may be properly labelled stochastic.
- This approach assumes that each word is known and has a finite set of possible tags originating from a specific tagset.
 - These tags can be drawn from a dictionary or a morphological analysis.
 - There exists a lot of tagsets nowadays, for (almost) every language.
 - **Reference:** sketchengine.eu/tagsets/
- **Principle:** When a word has more than one possible tag, statistical methods enable us to determine the optimal sequence of POS tags $T = t_1, t_2, \dots, t_n$, given a sequence of words $W = w_1, w_2, \dots, w_n$

- **In statistical terms:** Find the optimal POS tag sequence, denoted \hat{T} , by maximizing the conditional probability $P(T \mid W)$:

$$\hat{T} = \operatorname{argmax}_{t_1, t_2, \dots, t_n} \underbrace{P(t_1, t_2, \dots, t_n \mid w_1, w_2, \dots, w_n)}_{(*)}$$

- To compute $(*)$, we can:
 - Use the chain rule of probability:

$$\begin{aligned} (*) &= P(t_1 \mid W)P(t_2 \mid t_1; W)P(t_3 \mid t_1, t_2; W) \dots P(t_n \mid t_{1:n-1}; W) \\ &= \prod_{k=1}^n P(t_k \mid t_{1:k-1}; W) \end{aligned}$$

- Use Bayes' theorem:

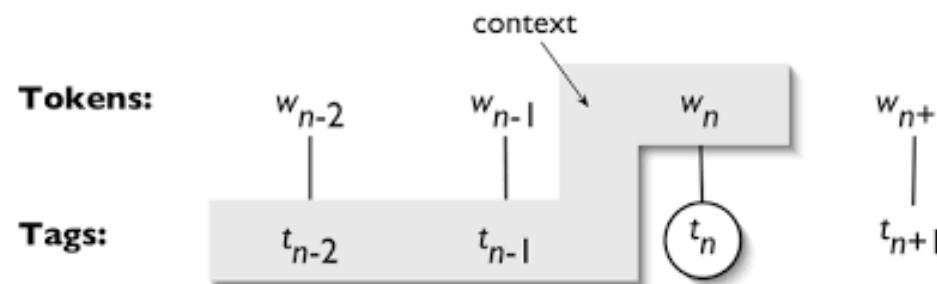
$$(*) = \frac{P(T)P(W \mid T)}{P(W)} \propto P(T)P(W \mid T)$$

- **Problem:** Statistics on sequences of any length are impossible to obtain
 - Consider 5 POS tags in the selected tagset & 10 words in a sentence
 - Number of probabilities to maximize from: $5^{10} \approx 10$ million
- To overcome this problem, Viterbi's algorithm can be used

Viterbi's algorithm

- It is a dynamic programming algorithm for obtaining the maximum posteriori probability estimate of the most likely sequence of hidden states (called the Viterbi path) that results in a sequence of observed event.
- Said differently, Viterbi decoding efficiently determines the most probable path from the exponentially many possibilities.
- We will discuss this algorithm within the HMM POS tagging framework (see later).

- The simplest stochastic taggers disambiguate words based solely on the probability that a word occurs with a particular tag. Said differently, the tag encountered most frequently in the training set with the word is the one assigned to an ambiguous instance of that word.
- **Problem:** While it may yield a valid tag for a given word, it can also yield inadmissible sequences of tags.
- Alternatively to the word frequency approach is to calculate the probability of a given sequence of tags occurring.
- This is often referred as the N-gram approach, i.e., the best tag (t_k) for a given word w_k is determined by the probability that it occurs with the N previous tags (t_{k-1}, t_{k-2}, \dots)



- Here, N can be chosen apriori (1, 2, 3, etc.)
- **Note:** A 1-gram tagger is another term for a unigram tagger; A 2-gram & 3-gram tagger are also called a bigram & trigram tagger

This is Big Data AI Book

<i>Uni-Gram</i>	This	Is	Big	Data	AI	Book
<i>Bi-Gram</i>	This Is	Is Big	Big Data	Data AI	AI Book	
<i>Tri-Gram</i>	This is Big	Is Big Data	Big Data AI	Data AI Book		

- This approach makes much more sense, since it considers the tags for individual words based on context.

- In the bigram tagger, for example, the conditional probability of tag t_k , corresponding to word w_k , is approximated as follows:

$$P(t_k \mid t_{1:k-1}; W) \approx P(t_k \mid t_{k-1}; W)$$

$$\Rightarrow (\star) \approx \prod_{k=1}^n P(t_k \mid t_{k-1}; W)$$

- The assumption that the probability of word depends only on the previous word is called a Markov assumption.
- In the general N-gram tagger, we have:

$$P(t_k \mid t_{1:k-1}; W) \approx P(t_k \mid t_{k-N+1}; W)$$

$$\Rightarrow (\star) \approx \prod_{k=1}^n P(t_k \mid t_{k-N+1}; W)$$

- **Question:** But how do we estimate these N-gram probabilities?
- **Answer: Maximum likelihood estimation**
 - Getting N-gram counts from a trained tagged corpus, i.e.,

$$C(t_{k-N+1:k-1} t_k \mid W)$$

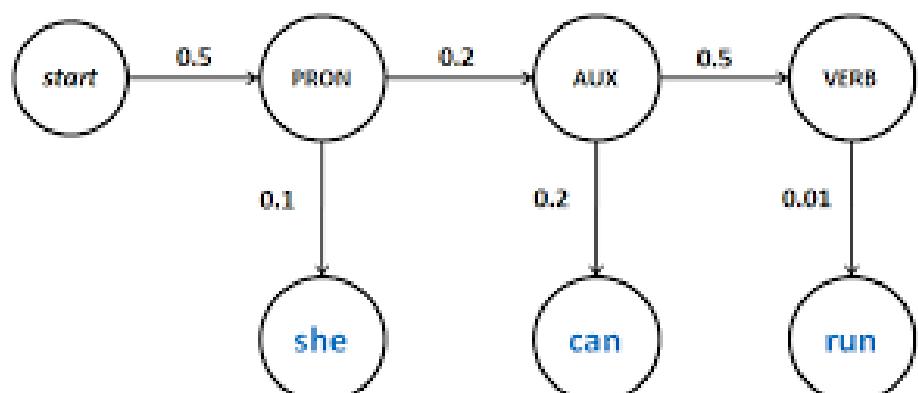
- Normalize these counts so that they lie between 0 and 1, i.e.,

$$\begin{aligned} P(t_k \mid t_{k-N+1}; W) &= \frac{C(t_{k-N+1:k-1} t_k \mid W)}{\sum_t C(t_{k-N+1:k-1} t \mid W)} \\ &\stackrel{(\star\star)}{=} \frac{C(t_{k-N+1:k-1} t_k \mid W)}{C(t_{k-N+1:k-1} \mid W)} \end{aligned}$$

($\star\star$) Since the sum of all N-gram counts that start with a given sequence of tags $t_{k-N+1:k-1}$ must be equal to the N-1-gram count for $t_{k-N+1:k-1}$.

[2] Hidden Markov Model (HMM)

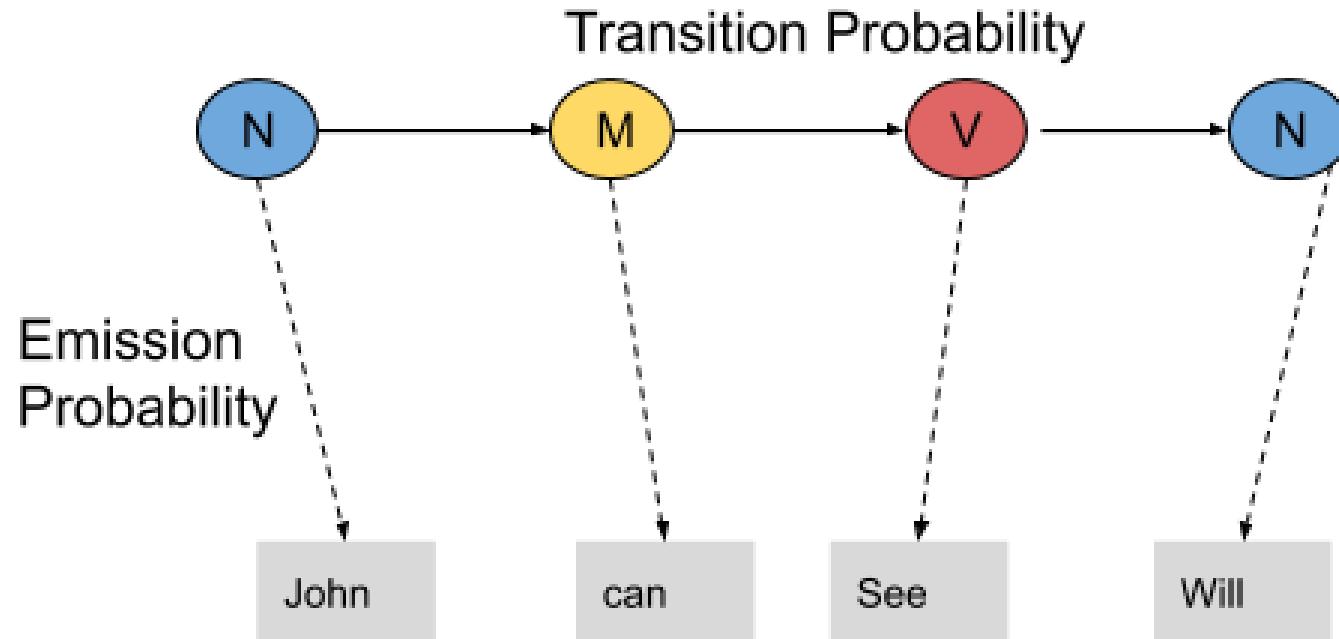
- To model any problem using a HMM, we need a set of observations and a set of possible hidden states.
- In the POS tagging problem:
 - Observations: Words in the given sequence
 - States: POS tags for the words



PRON, AUX, and VERB are hidden states

she, can, and run are observations

- From this representation, we observe two kind of probabilities:
 - Emission probability: Probability of making certain observations given a particular state
 - Transition probability: Probability of transitioning to another state given a particular state

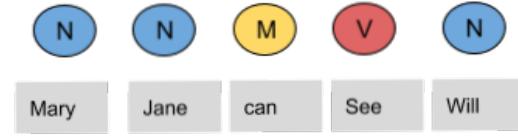


- These probabilities are derived from a trained tagged corpus.

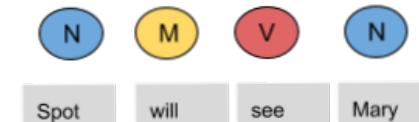
Example:

- Consider the following tagged corpus, consisting of 4 sentences:

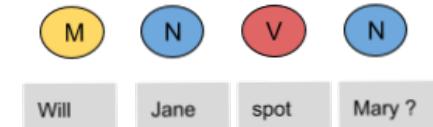
- Mary Jane can see Will



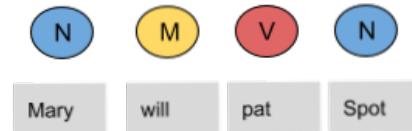
- Spot will see Mary



- Will Jane spot Mary?



- Mary will pat Spot



- **Remark:** For simplicity, only 3 POS are considered in this tagset, i.e., noun (N), model (M) and verb (V).

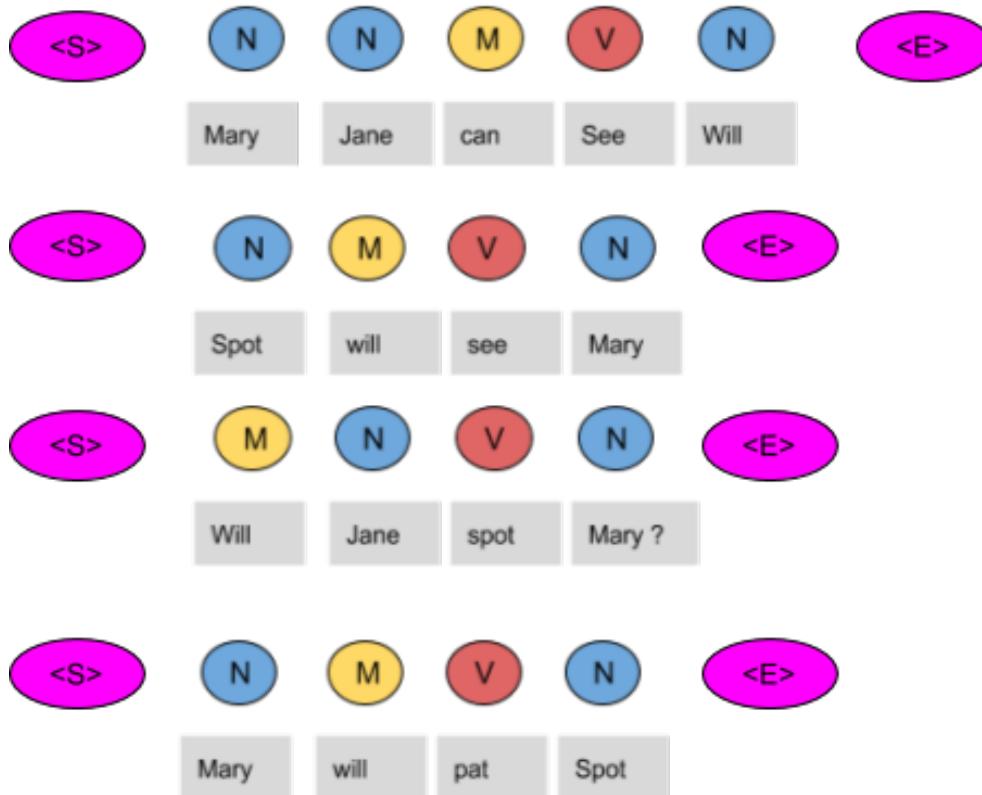
Emission probabilities:

- To calculate the emission probabilities, we first create a frequency table (left), based on the given tagged corpus, and divide each column by the total number of their appearances (right), i.e., the **emission probabilities**:

		Tags					Tags		
		Noun	Model	Verb			Noun	Model	Verb
Word	Mary	4	0	0	Word	Mary	4/9	0	0
	Jane	2	0	0		Jane	2/9	0	0
	Will	1	3	0		Will	1/9	3/4	0
	Spot	2	0	1		Spot	2/9	0	1/4
	Can	0	1	0		Can	0	1/4	0
	See	0	0	2		See	0	0	2/4
	pat	0	0	1		pat	0	0	1/4

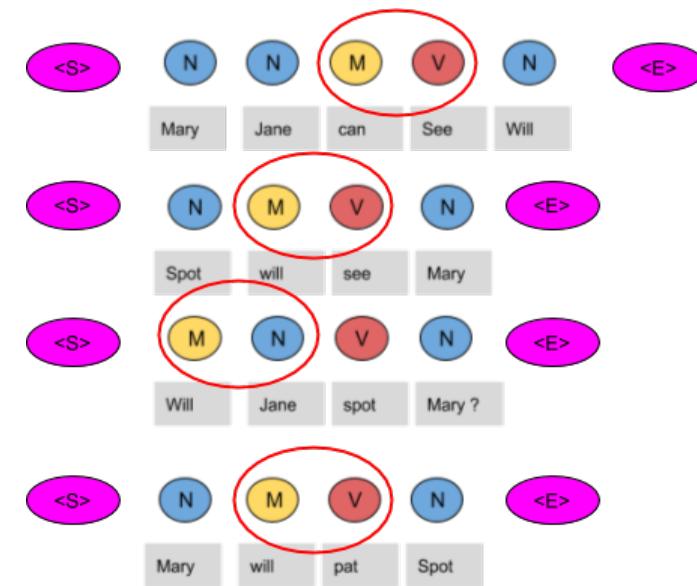
Transition probabilities:

- To calculate the transition probabilities, we first create two more tags $< S >$ and $< E >$, i.e., to define the start and end of a sentence, respectively:



- We can now creating a co-occurrence frequency table, based on the tagged corpus:

		Tags			
		Noun	Model	Verb	< E >
Tags	< S >	3	1	0	0
	Noun	1	3	1	4
Model	1	0	3	0	
Verb	4	0	0	0	



- The **transition probabilities** can now be derived by dividing each term of the co-occurrence frequency table by the total number of co-occurrences of the tag in consideration:

		Tags			
		Noun	Model	Verb	$\langle E \rangle$
$\langle S \rangle$		3/4	1/4	0	0
Tags	Noun	1/9	3/9	1/9	4/9
	Model	1/4	0	3/4	0
	Verb	4/4	0	0	0

- **Question:** Now that we have these probabilities, how does the HMM determine the appropriate sequence of tags for a particular (new) sentence with n words?

- **Reminder:** We wish to find

$$\hat{T} = \operatorname{argmax}_{t_1, t_2, \dots, t_n} \underbrace{P(t_1, t_2, \dots, t_n \mid w_1, w_2, \dots, w_n)}_{(*)}$$

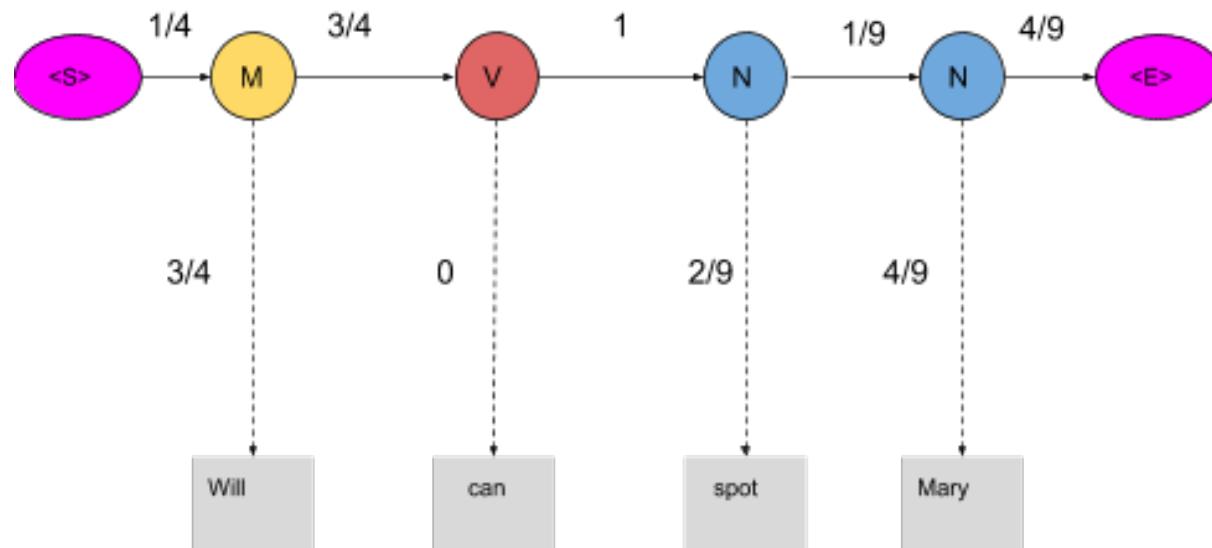
where $(*) \propto \underbrace{P(T)P(W \mid T)}_{(\Delta)}$

- **Answer:** By using the Markov assumption, HMM will approximate (Δ) with

$$Q = \underbrace{\prod_{k=1}^{n+1} P(t_k \mid t_{k-1})}_{\text{TRANSITIONS } (t_{k-1} \rightarrow t_k)} \cdot \underbrace{\prod_{k=1}^n P(w_k \mid t_k)}_{\text{EMISSIONS } (t_k \rightarrow w_k)}$$

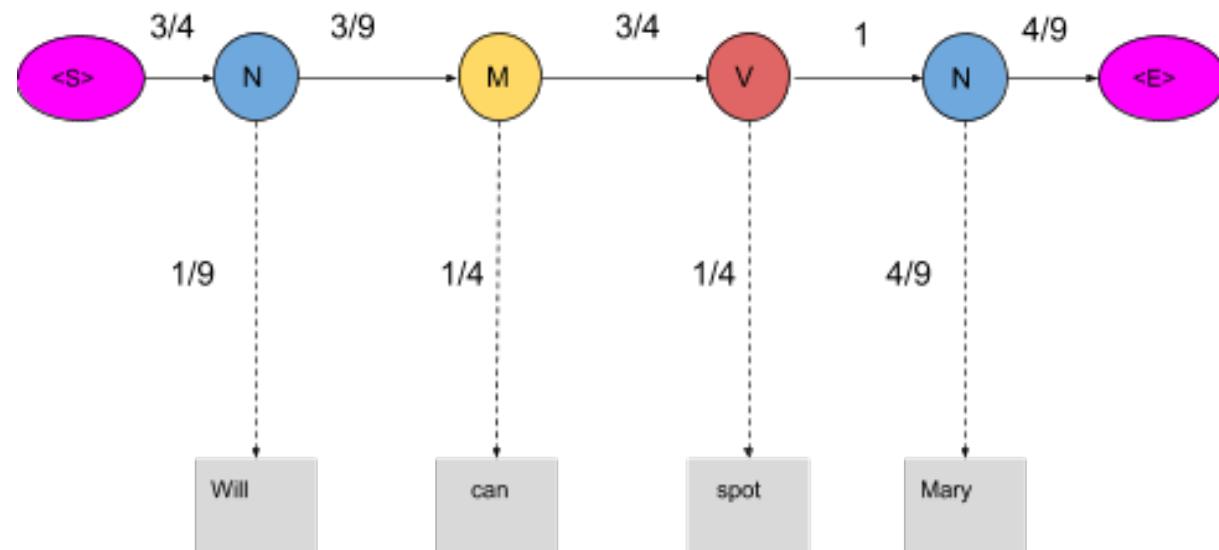
Example: Take a new sentence "Will can spot Mary".

- Consider the following POS tagging for this sentence (obvious wrong): Will as a model; Can as a verb; Spot as a noun; Mary as a noun.
- The probability Q of this sequence being correct equals:



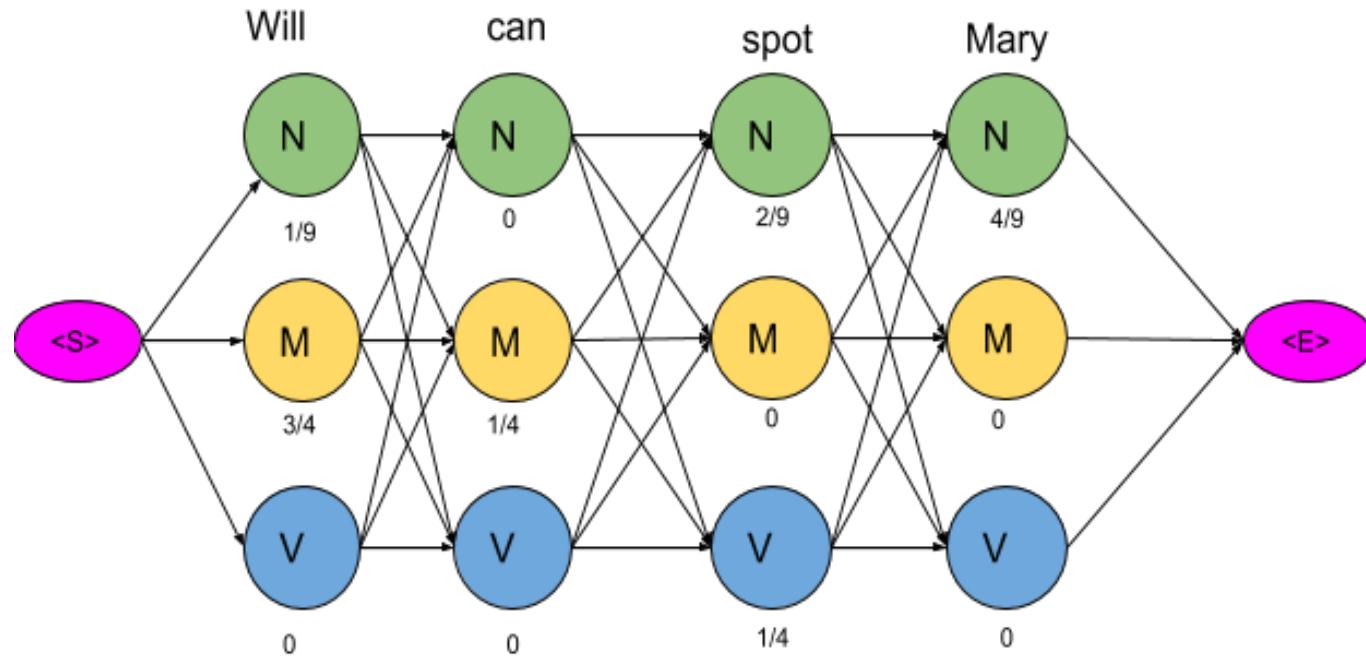
$$\Rightarrow Q = \frac{1}{4} \cdot \frac{3}{4} \cdot 1 \cdot \frac{1}{9} \cdot \frac{4}{9} \cdot \frac{3}{4} \cdot 0 \cdot \frac{2}{9} \cdot \frac{4}{9} = 0$$

- Also consider another POS tagging (obvious right): Will as a noun; Can as a model; Spot as a verb; Mary as a noun.
- The probability Q of this sequence being correct now equals:



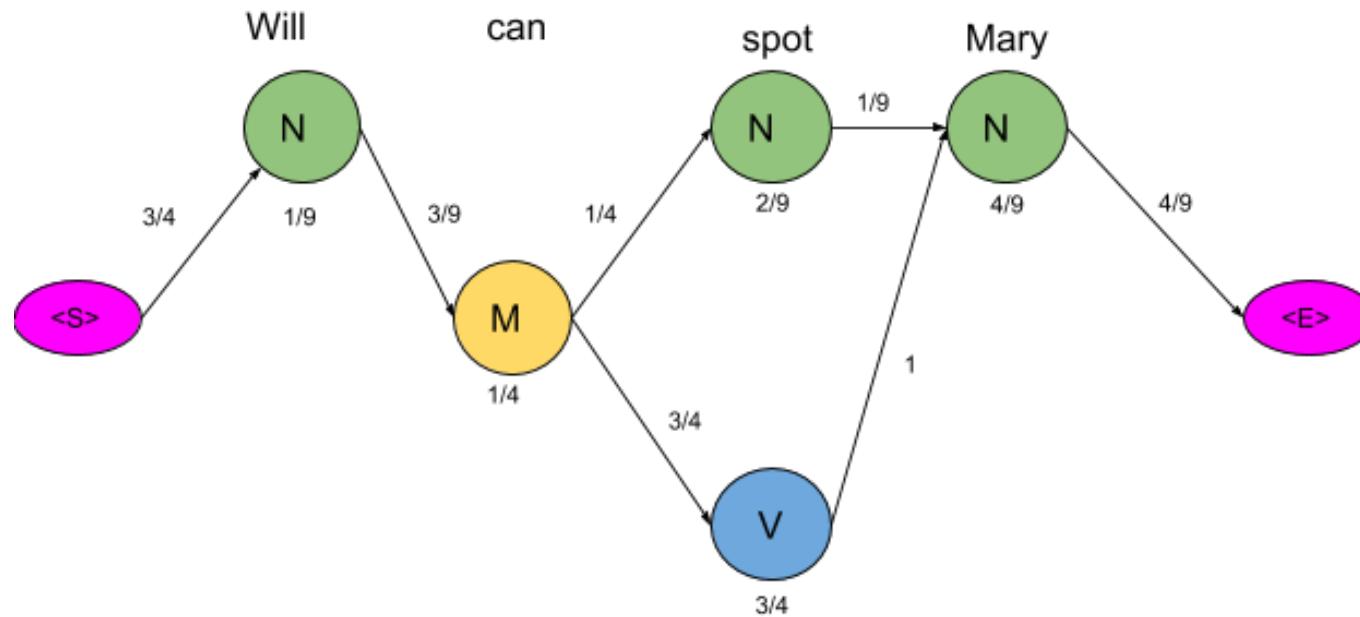
$$\Rightarrow Q = \frac{3}{4} \cdot \frac{3}{9} \cdot \frac{3}{4} \cdot 1 \cdot \frac{4}{9} \cdot \frac{1}{9} \cdot \frac{1}{4} \cdot \frac{1}{4} \cdot \frac{4}{9} = 0.00025720164$$

- By considering only 3 POS tags and a relative small new sentence (with only 4 words), 81 Q's can be calculated, which seems achievable.



- When the task is to tag a larger sentence and all the POS tags in the Penn Treebank project are taken into consideration (which is often preferred in practice), the number of Q's grows exponentially.

- To partly resolve this issue, we can delete all possible tags with 0 probability and all paths which do not lead to the endpoint:



- As result, we reduced the 81 possible combinations to only 2 possible combinations:

$< S > \rightarrow N \rightarrow M \rightarrow N \rightarrow N \rightarrow < E >$:

$$Q = \frac{3}{4} \cdot \frac{3}{9} \cdot \frac{1}{4} \cdot \frac{1}{9} \cdot \frac{4}{9} \cdot \frac{1}{9} \cdot \frac{1}{4} \cdot \frac{2}{9} \cdot \frac{4}{9} = 0.00000846754$$

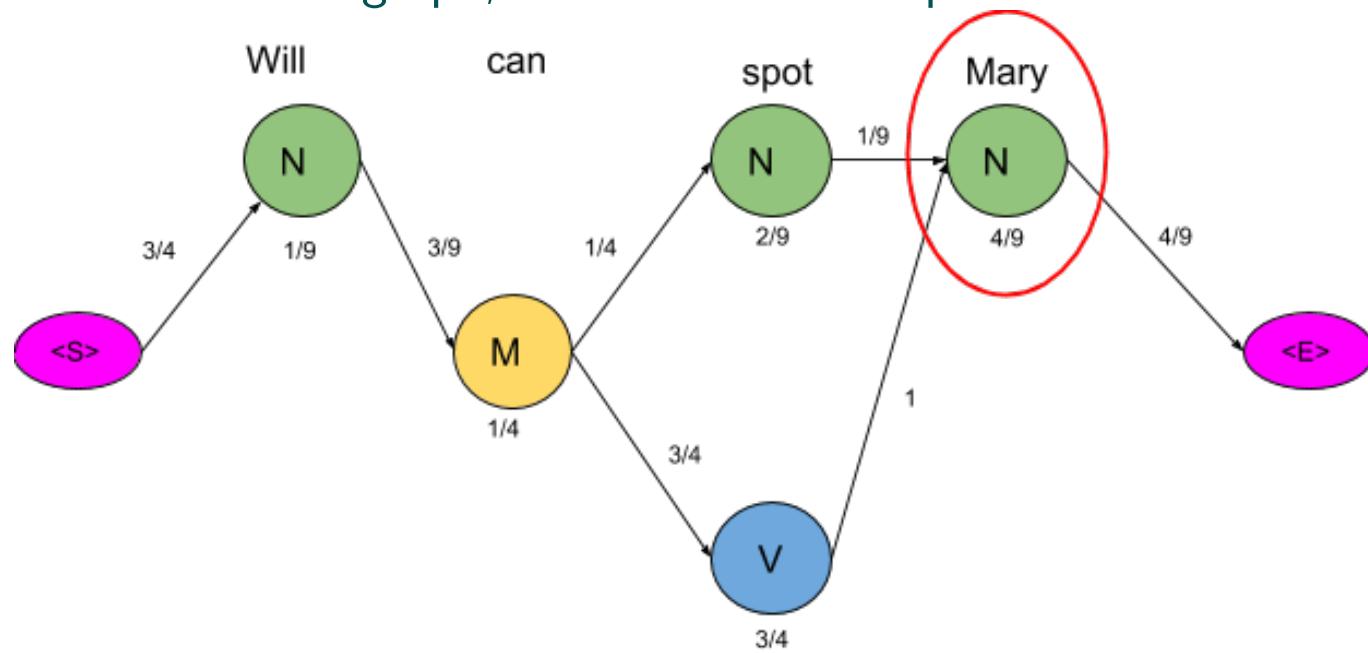
$< S > \rightarrow N \rightarrow M \rightarrow N \rightarrow V \rightarrow < E >$:

$$Q = \frac{3}{4} \cdot \frac{3}{9} \cdot \frac{3}{4} \cdot 1 \cdot \frac{4}{9} \cdot \frac{1}{9} \cdot \frac{1}{4} \cdot \frac{1}{4} \cdot \frac{4}{9} = 0.00025720164$$

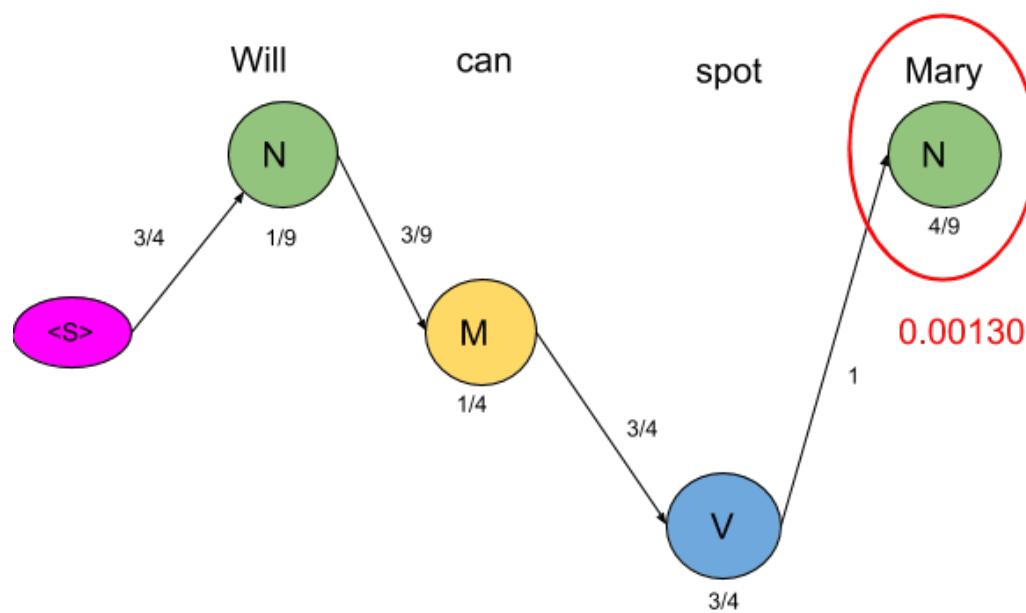
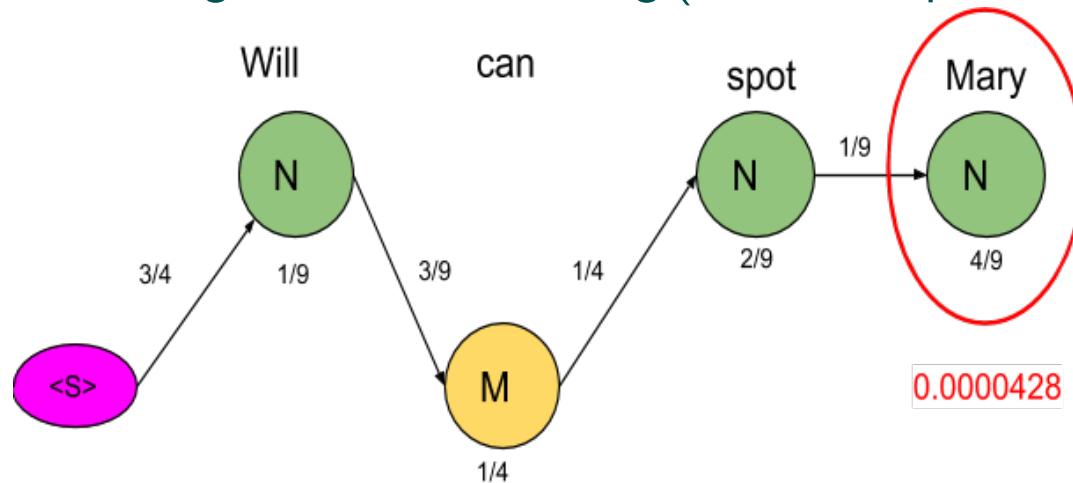
- Alternative/additional to this approach, we can (further) optimize the HMM by using the **Viterbi algorithm**, invented by Andrew James Viterbi.

[3] Viterbi algorithm:

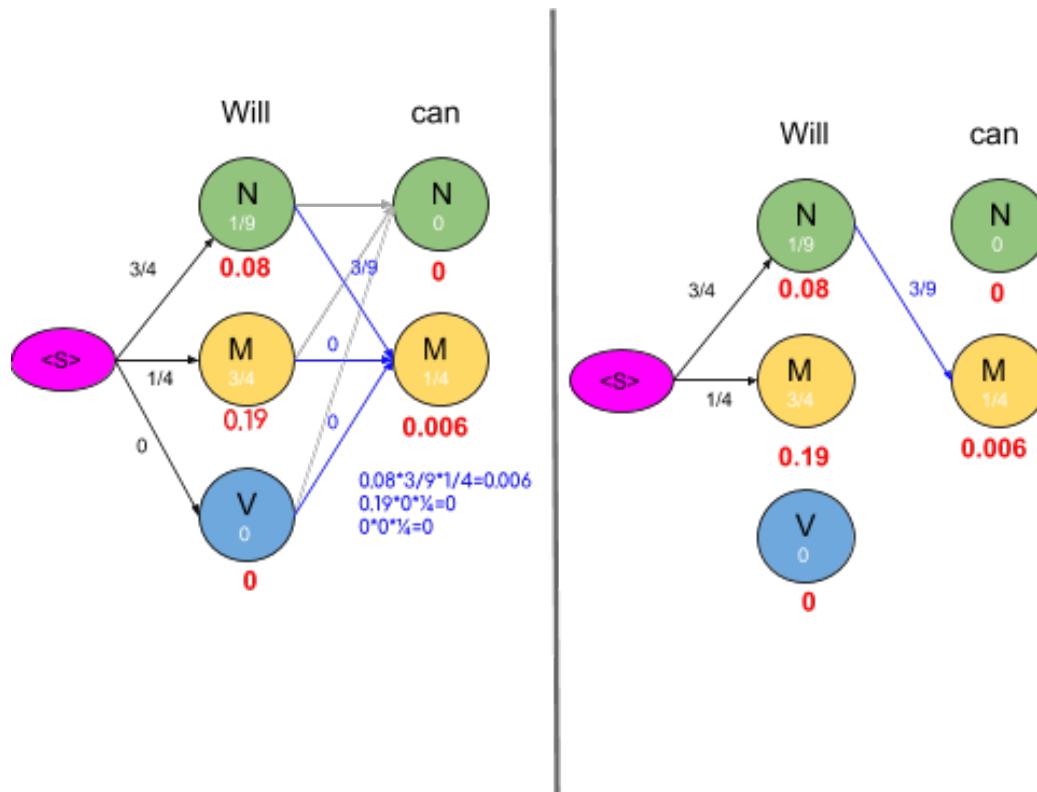
- Reconsider the obtained graph, where our focus is put on the encircled tag:



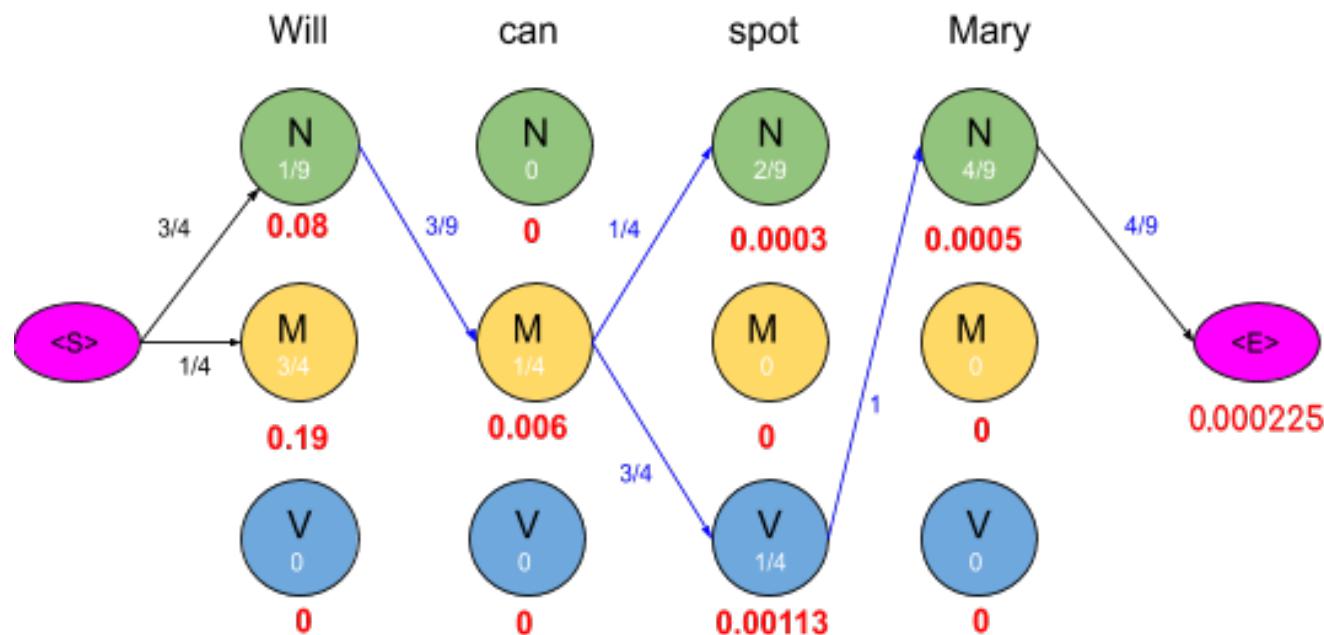
- There are 2 paths leading to this encircled tag (with corresponding Q-value in red):



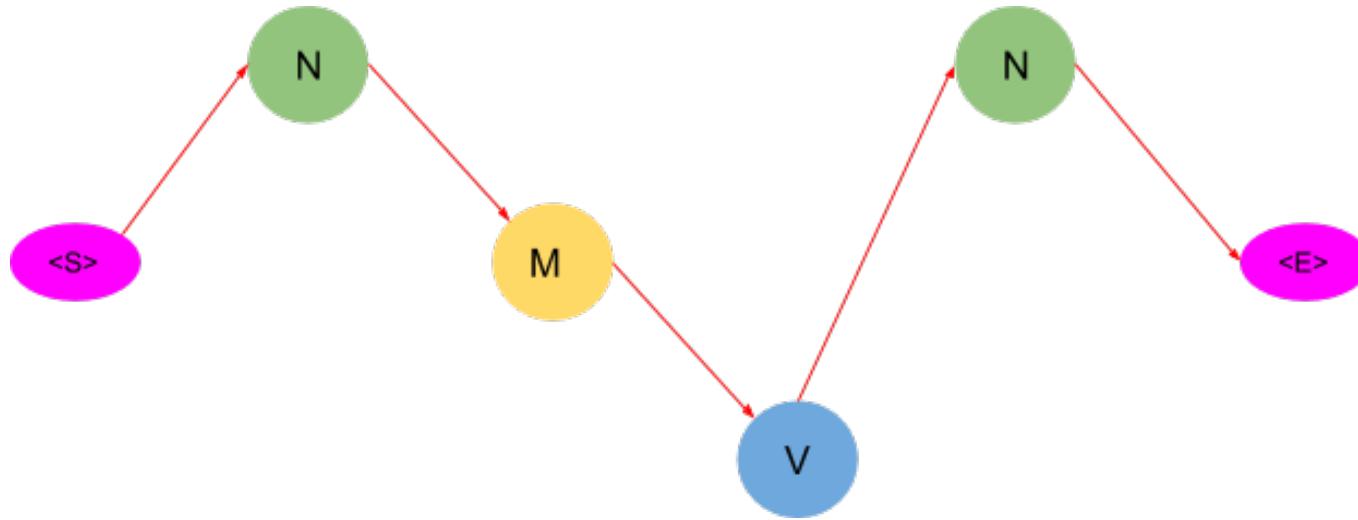
- The Viterbi algorithm now selects the path with the highest probability, and continuous with it.
- As we now already continued from a reduced number of consideration, we can also apply the Viterbi algorithm at the start:



- The graph obtained after computing probabilities of all paths leading to a specific outcome tag is shown below:



- To get an optimal path, we start from the end and trace backward:



- **Remark:** The algorithm returns only one path as compared to the previous method which suggested two paths. Thus by using this algorithm, we saved us a lot of computations.

Remarks on the HMM approach:

- The HMM approach belongs to the family of probabilistic graphical models in machine learning.
- Since HMM models the joint probability of the sequence of words (W) and tags (T), it is called a **generative model**, i.e., they attempt to recreate the original generating process responsible for creating the label-word pairs.
 - **Remark:** If you model on the conditional probability $P(T | W)$, it is a **discriminative model**.
 - **Question:** But how can you see that the HMM is modeling the joint probability $P(T, W)$?

- **Answer:** HMM tries to model $(\Delta) = P(T)P(W | T)$, which correspond to the joint probability $P(T, W)$. By using Bayes' rule, the conditional probability can be obtained afterwards, and maximized.
- While the HMM is a useful and powerful model, it needs a number of augmentations to achieve high accuracy.

Examples:

1. In POS tagging, we often run into **unknown words** in a sentence that are not incorporated in the training corpus:

Profits soared at Boeing Co., easily topping forecasts on Wall Street, as their CEO Alan Mulally announced first quarter results.

In this sentence it is quite likely that the word Mulally has not been seen in training data. Similarly, topping is a relatively infrequent word in English, and may not have been seen in training.

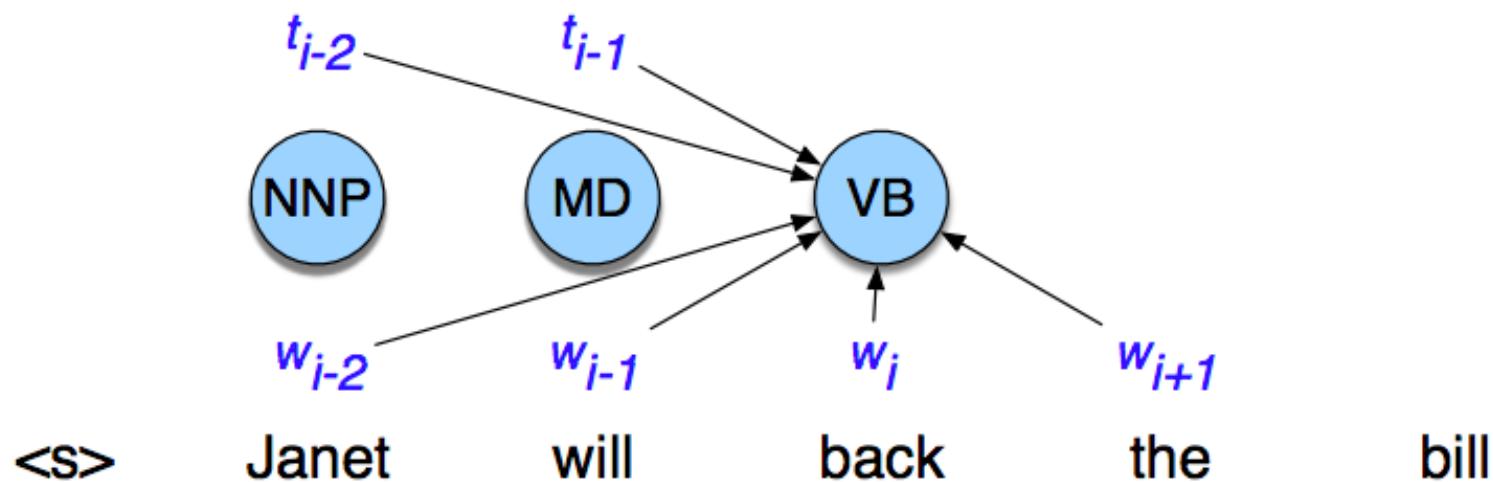
Because of this, for any test sentence W that contains some word(s) that is never seen in training data, it is easily verified that

$$P(T, W) = 0, \quad \forall T.$$

Thus, the HMM model will completely fail on the test sentence.

2. The structure of HMMs are limited in dependency structures.

For example, it would be more helpful if a decision about tag t_i could directly use information about previous and future known words in the sentence, e.g., w_{i+1} , w_{i-1} , w_{i-2} (in addition to w_i), and previous tags, e.g., t_{i-1} and t_{i-2} .



This model framework is often referred as **Maximum Entropy Markov Model (MEMM)**, i.e., a discriminative model.

[4] Maximum Entropy Markov Model (MEMM)

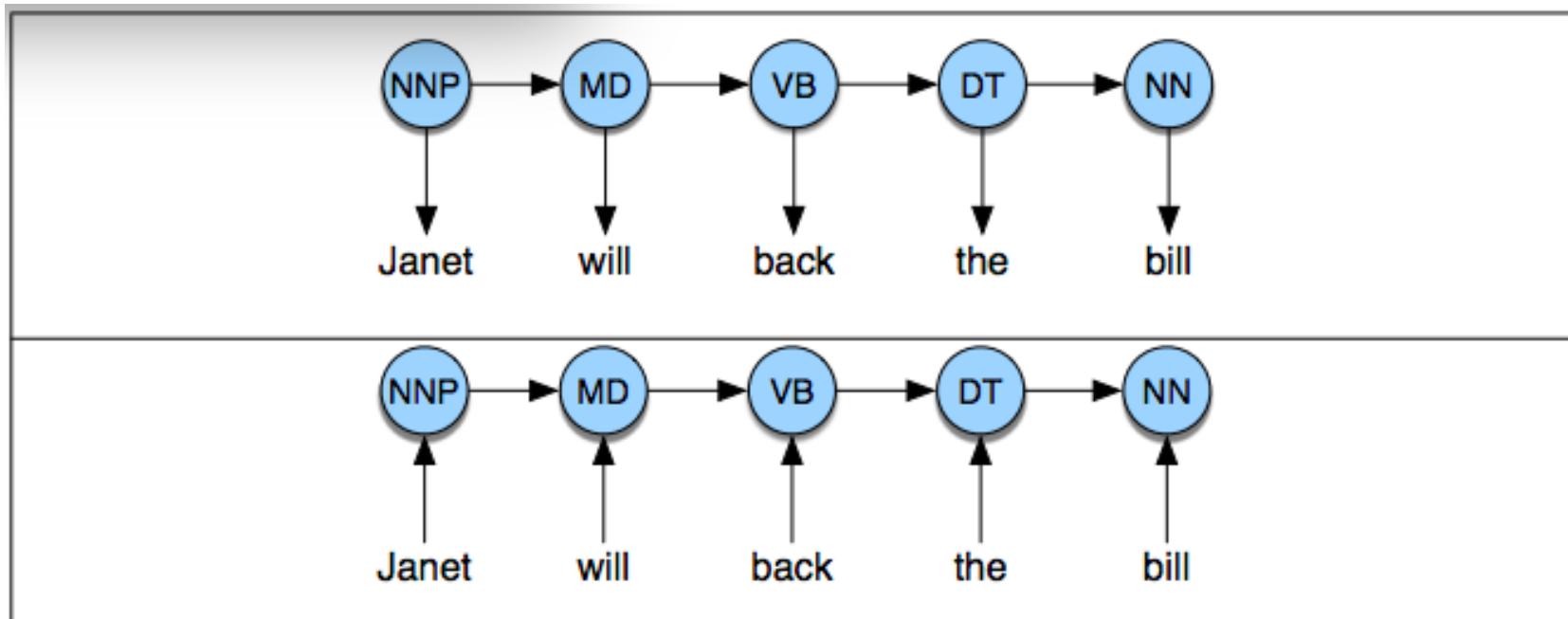
- **Reminder:** In HMMs, the optimal POS tag sequence \hat{T} is obtained by maximizing $P(T \mid W)$, based on the Bayes' rule, in the following sense:

$$\begin{aligned}\hat{T} &= \operatorname{argmax}_T P(T \mid W) = \operatorname{argmax}_T P(W \mid T)P(T) \\ &\approx \operatorname{argmax}_T \prod_{k=1}^{n+1} P(t_k \mid t_{k-1}) \cdot \prod_{k=1}^n P(w_k \mid t_k)\end{aligned}$$

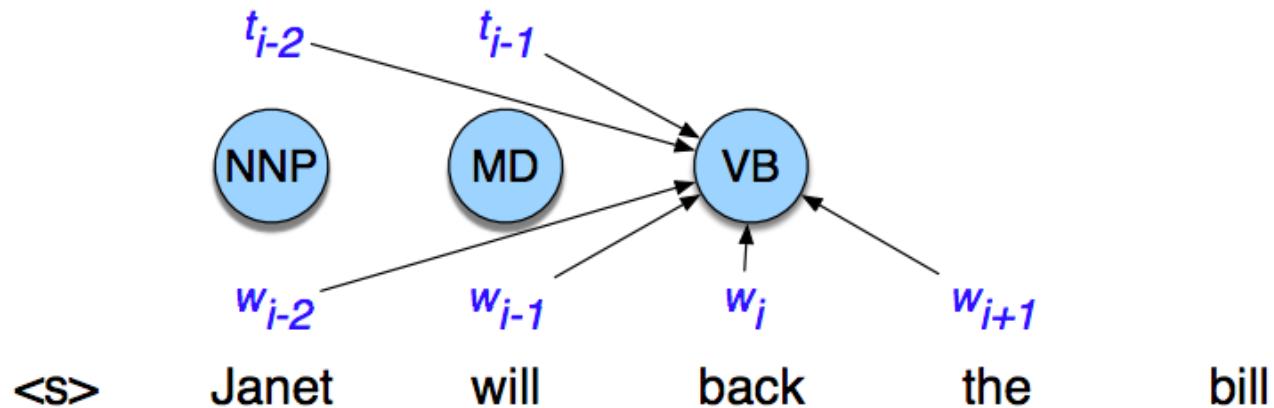
- In a MEMM, by contrast, we compute the conditional probability $P(T \mid W)$ directly, training it to discriminate among the possible tag sequences, e.g.:

$$\begin{aligned}\hat{T} &= \operatorname{argmax}_T P(T \mid W) \\ &\approx \operatorname{argmax}_T \prod_{k=1}^n P(t_k \mid w_k, t_{k-1})\end{aligned}$$

- Said differently, HMMs (top) compute the likelihood, i.e., observation word conditioned on tags, while MEMMs (below) compute the posterior distribution, i.e., tags conditioned on observation words:



- Of course, we don't build MEMMs that condition just on w_k and t_{k-1} .
- A basic MEMM POS tagging for tag t_i conditions on the observation word itself (w_i), neighboring words (e.g., $w_{i-2}, w_{i-1}, w_{i+1}$), and previous tags (e.g., t_{i-2}, t_{i-1}), and various combinations, using so-called feature templates like the following:

$$\begin{aligned} & \langle t_i, w_{i-2} \rangle, \langle t_i, w_{i-1} \rangle, \langle t_i, w_i \rangle, \langle t_i, w_{i+1} \rangle, \langle t_i, w_{i+2} \rangle, \\ & \langle t_i, t_{i-1} \rangle, \langle t_i, t_{i-2}, t_{i-1} \rangle, \langle t_i, t_{i-1}, w_i \rangle, \langle t_i, w_{i-1}, w_i \rangle, \langle t_i, w_i, w_{i+1} \rangle. \end{aligned}$$


- In total, we have $J = 10$ features here, i.e.,

- $j = 1: \langle t_i, w_{i-2} \rangle;$
- $j = 2: \langle t_i, w_{i-1} \rangle;$
- $j = 3: \langle t_i, w_i \rangle;$
- $j = 4: \langle t_i, w_{i+1} \rangle;$
- $j = 5: \langle t_i, w_{i+2} \rangle;$
- $j = 6: \langle t_i, t_{i-1} \rangle;$
- $j = 7: \langle t_i, t_{i-2}, t_{i-1} \rangle;$
- $j = 8: \langle t_i, t_{i-1}, w_i \rangle;$
- $j = 9: \langle t_i, w_{i-1}, w_i \rangle;$
- $j = 10: \langle t_i, w_i, w_{i+1} \rangle;$

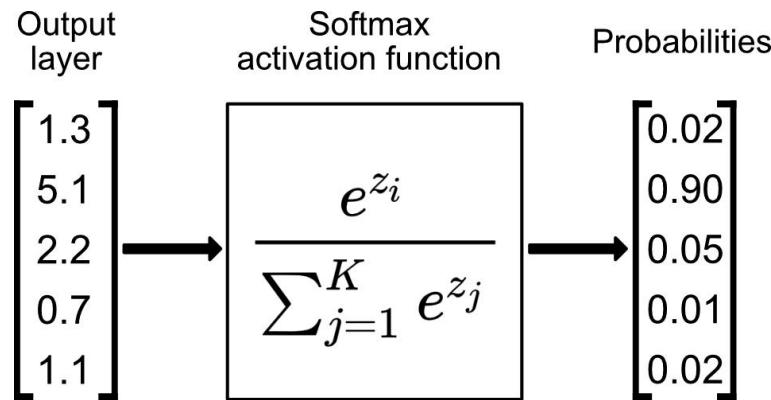
- This, of course, can be extended/changed with other new features, depending on your own choice.
- **Question:** But how will the optimal tag sequence \hat{T} then be calculated in this MEMM formulation, hereby taken into account the different pre-defined features?

- **Answer:** The most likely sequence of tags \hat{T} is computed by combining these features of the input word w_i , its neighbors within l words w_{i-l}^{i+l} , and the previous v tags t_{i-1}^{i-v} as follows:

$$\begin{aligned}
 \hat{T} &= \operatorname{argmax}_T P(T \mid W) \\
 &\approx \operatorname{argmax}_T \prod_{i=1}^n \underbrace{P(t_i \mid w_i, w_{i-l}^{i+l}, t_{i-1}^{i-v})}_{(\Gamma)} \\
 &\approx \operatorname{argmax}_T \prod_{i=1}^n \frac{\exp \left(\sum_{j=1}^J \theta_j f_j(t_i, w_i, w_{i-l}^{i+l}, t_{i-1}^{i-v}) \right)}{\sum_{t' \in Q} \exp \left(\sum_{j=1}^J \theta_j f_j(t', w_i, w_{i-l}^{i+l}, t_{i-1}^{i-v}) \right)}
 \end{aligned}$$

- Q represents the tagset;
- θ_j is the weight corresponding to the so-called feature function $f_j(\cdot)$.

- As can be seen from the expression, the conditional probability (Γ) is derived by a **softmax function**.
 - In neural networks, for example, this function is often used as last activation function, since it normalizes the output of a network to a probability distribution.

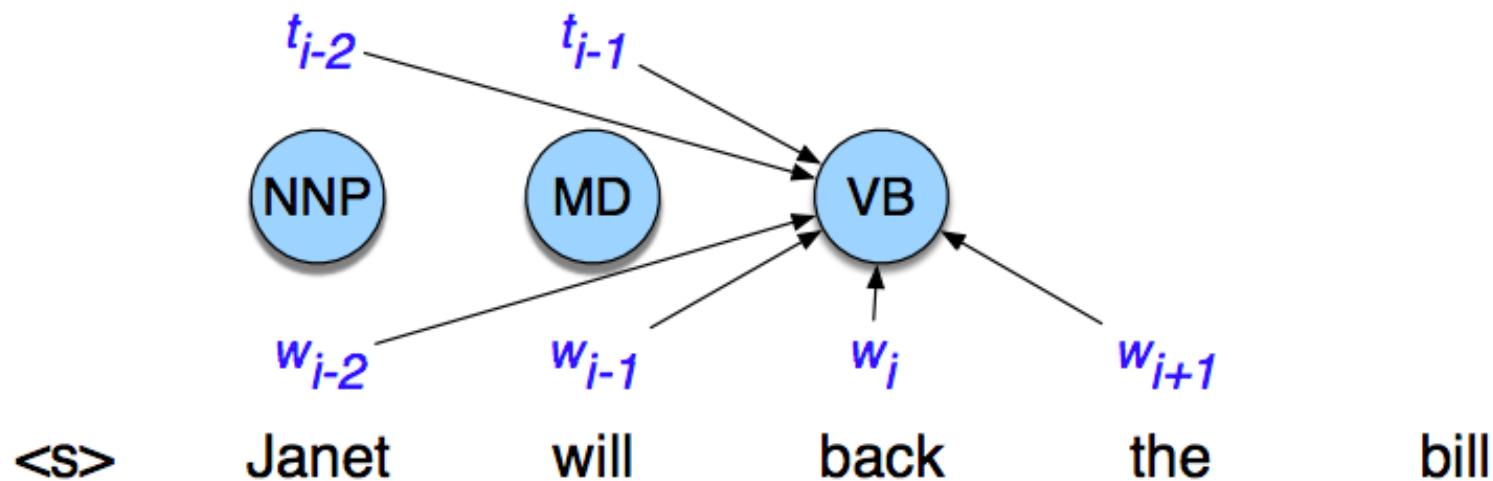


- Within this softmax function, we are using specific weights $\{\theta_j \mid j = 1, \dots, J\}$ corresponding to so-called feature functions $\{f_j(\cdot) \mid j = 1, \dots, J\}$.
- Question:** But what are these feature functions?

- **Answer:**

- Functions that take account of the pre-defined features
- Often, these are formulated as indicator functions.

Our example:



$$f_1(t_i, w_i, w_{i-l}^{i+l}, t_{i-1}^{i-v}) = \begin{cases} 1 & \text{if } t_i = VB \text{ & } w_{i-2} = Janet \\ 0 & \text{otherwise} \end{cases}$$

$$f_2(t_i, w_i, w_{i-l}^{i+l}, t_{i-1}^{i-v}) = \begin{cases} 1 & \text{if } t_i = VB \text{ & } w_{i-1} = will \\ 0 & \text{otherwise} \end{cases}$$

$$f_3(t_i, w_i, w_{i-l}^{i+l}, t_{i-1}^{i-v}) = \begin{cases} 1 & \text{if } t_i = VB \text{ & } w_i = back \\ 0 & \text{otherwise} \end{cases}$$

⋮

$$f_{10}(t_i, w_i, w_{i-l}^{i+l}, t_{i-1}^{i-v}) = \begin{cases} 1 & \text{if } t_i = VB \text{ & } w_i = back \text{ & } w_{i+1} = the \\ 0 & \text{otherwise} \end{cases}$$

- **Question:** And what are these weights?

- **Answer:**
 - The weight θ_j for a feature function $f_j(\cdot)$ captures how closely feature j is related to the given tag t_i .
 - **Question:** How can we find these weights?
 - **Answer:** In the training process, i.e., where all words and corresponding tags are known, $\Theta = \{\theta_j \mid j = 1, \dots, J\}$ is randomly initialized and optimized during training through **gradient descent** by **minimizing** a specific **loss/cost function**.

Example:

- Consider the log-likelihood function $l(\Theta)$, expressed as

$$l(\Theta) = \sum_i \log [P(t_i | w_i, w_{i-l}^{i+l}, t_{i-1}^{i-v}; \Theta)],$$

and assume $-l(\Theta)$ as **loss/cost function**.

- **Maximize** $l(\Theta)$: Maximum-likelihood estimate (MLE)

$$\begin{aligned}\hat{\Theta} &= \operatorname{argmax}_{\Theta} \sum_i \log [P(t_i | w_i, w_{i-l}^{i+l}, t_{i-1}^{i-v}; \theta_j)] \\ &\equiv \operatorname{argmin}_{\Theta} \left\{ - \sum_i \log \left[P(t_i | \underbrace{w_i, w_{i-l}^{i+l}, t_{i-1}^{i-v}}_{F_i}; \theta_j) \right] \right\}\end{aligned}$$

- **Remark:** Maximizing $l(\Theta) \equiv$ minimizing $-l(\Theta)$

- To minimize $-l(\Theta)$, **gradient descent** can be used:
 1. First, initialize Θ to some random values $\hat{\Theta}^{(0)} = \left\{ \hat{\theta}_j^{(0)} \mid j = 1, \dots, J \right\}$;
 2. Then iterate through each input and each iteration you update the weight by finding the derivative of $-l(\Theta)$ with respect to θ_j :

$$\begin{aligned}\frac{\partial [-l(\Theta)]}{\partial \theta_j} &= - \sum_i f_j(t_i, F_i) \\ &+ \sum_i \sum_{t' \in Q} \frac{\exp \left(\sum_{j=1}^J \theta_j f_j(t', F_i) \right)}{\sum_{t' \in Q} \exp \left(\sum_{j=1}^J \theta_j f_j(t', F_i) \right)} f_j(t', F_i)\end{aligned}$$

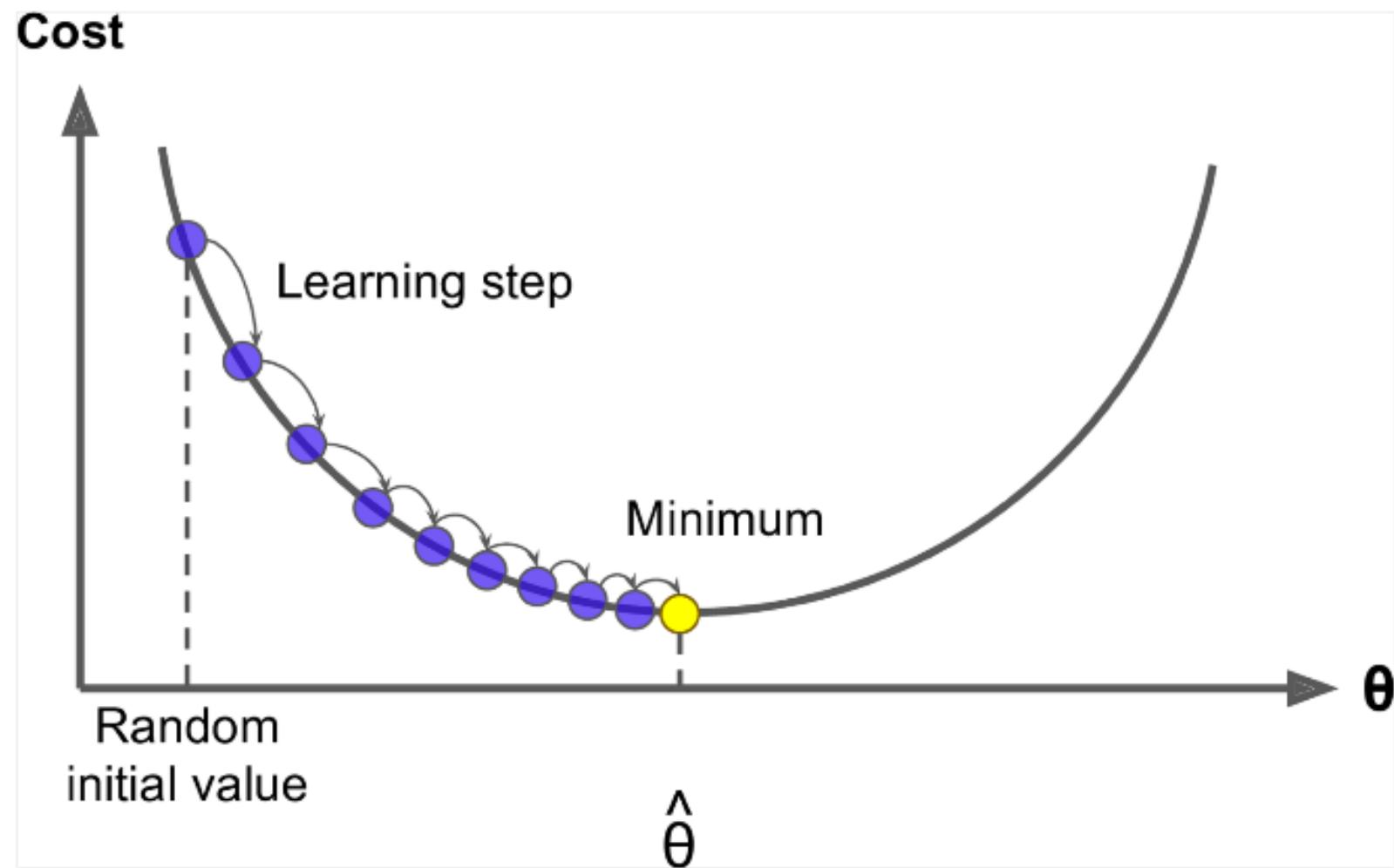
3. Update $\hat{\theta}_j$ as below, and repeat it until converge:

$$\hat{\theta}_j^{(s)} = \hat{\theta}_j^{(s-1)} + \alpha \frac{\partial}{\partial \theta_j} \left[-l(\theta^{(s-1)}) \right].$$

Here,

- α is the learning rate that determine how much to offset;
- $\hat{\theta}_j^{(s)}$ is the estimated weight $\hat{\theta}_j$ at current time step s in the iteration and $s - 1$ is the previous step.

So at each iteration, we move the estimate weights to some distance α in the direction of the gradient.



Remarks on the MEMM approach:

- To partially solve the problem of unknown words in a new sentence (which was a drawback in HMM), additional features are often incorporated that help with unknown words, such as:

- **Word shape features**

- It represents the abstract letter pattern of the word by mapping lower-case letters to 'x', upper-case to 'X', numbers to 'd', and retaining punctuation.
- **Example:** DC10-30 → XXdd-dd

- **Shorter word shape features**

- In these features, consecutive character types are removed, so words in all caps map to X, words with initial-caps map to Xx.
- **Example:** DC10-30 → Xd-d

- **Prefix & suffix features**

Example: The word "well-dressed" might generate the following non-zero valued feature values:

prefix = w
prefix = we
suffix = ed
suffix = d
word-shape = xxxx-xxxxxx
short-word-shape = x-x

These features can be computed for every word seen in the training.

To avoid that you have an enormous amount of features, that are still informative, a **feature cutoff** is generally used in which features are thrown out if they have count < 5 in the training set.

- Similar to HMM, the MEMM approach can be seen as a **directed approach**, as it exclusively run left-to-right.
 - **Limitation:** A decision about word w_i could not directly use information about future tags t_{i+1} and t_{i+2} .

[5] Conditional Random Fields (CRFs)

- **Reminder:** In MEMMs, the optimal POS tag sequence \hat{T} is obtained by maximizing $P(T \mid W)$ in the following sense:

$$\begin{aligned}\hat{T} &= \operatorname{argmax}_T P(T \mid W) \\ &\approx \operatorname{argmax}_T \prod_{i=1}^n \underbrace{P(t_i \mid w_i, w_{i-l}^{i+l}, t_{i-1}^{i-v})}_{(\Gamma)} \\ &\approx \operatorname{argmax}_T \prod_{i=1}^n \frac{\exp \left(\sum_{j=1}^J \theta_j f_j(t_i, w_i, w_{i-l}^{i+l}, t_{i-1}^{i-v}) \right)}{\sum_{t' \in Q} \exp \left(\sum_{j=1}^J \theta_j f_j(t', w_i, w_{i-l}^{i+l}, t_{i-1}^{i-v}) \right)}\end{aligned}$$

- In a CRF, the conditional probability $P(T \mid W)$ is also computed directly, but not by computing a probability for each tag at each time.

- Instead, at each time step the CRF computes log-linear functions over a set of relevant (local) features, and these local features are aggregated and normalized to produce a global probability for the whole sequence.
- More formally, the optimal POS tag sequence \hat{T} in a CRF is obtained by maximizing $P(T | W)$ in the following sense:

$$\begin{aligned}\hat{T} &= \operatorname{argmax}_T P(T | W) \\ &\approx \operatorname{argmax}_T \frac{\exp \left(\sum_{k=1}^K \theta_k F_k(W, T) \right)}{\sum_{T' \in \Upsilon} \exp \left(\sum_{k=1}^K \theta_k F_k(W, T') \right)}\end{aligned}$$

- Υ represent all possible tag sequences;
- θ_k is the weight corresponding to the so-called global feature function $F_k(\cdot)$.
- **Question:** What are those global feature functions $F_k(\cdot)$?

- **Answer:**

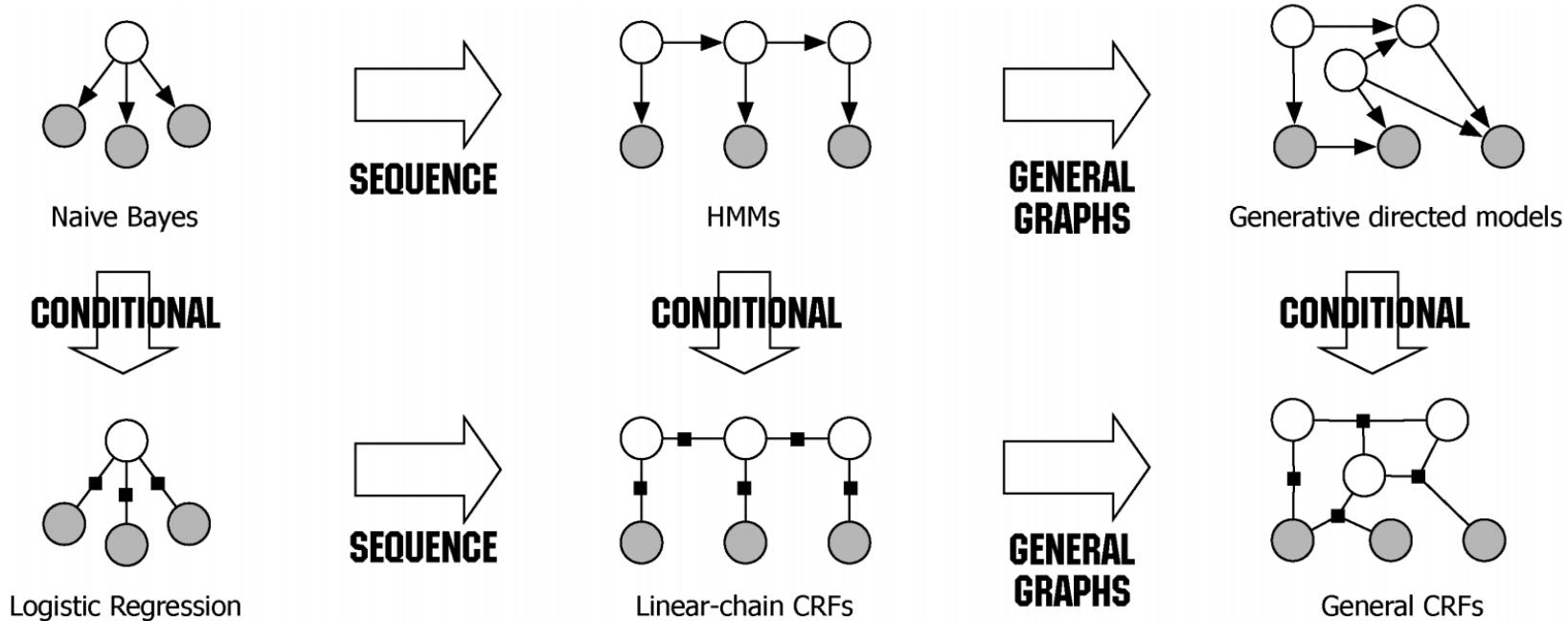
- In a CRF, the function $F_k(W, T)$ maps an entire word sequence W and an entire tag sequence T to a feature vector
 - **Note:** Therefore, the set of $F_k(W, T)$ are called global features since each one is a property of the entire word sequence W and tag sequence T .
- We compute them by decomposing into a sum of local features for each position i in T :

$$F_k(W, T) = \sum_i^n f_k(t_{i-1}, t_i, W, i).$$

- Here, we choose that each local features f_k only depend on the current and previous tags t_i and t_{i-1} , also known as the so-called linear-chain CRF.
- A general CRF allows a feature to make use of any tags, and are thus necessary for tasks in which the decision depend on distant tags, like t_{i-4} .

Remarks on the CRF approach:

- Similar to MEMM, CRF is a discriminative model.
- While HMM and MEMM can be seen as a directed graph, the (linear-chain) CRF can be expressed as an undirected graph.



- CRF calculates the normalizing probability in the global scope, rather than in the local scope as is the case with MEMM.
- Hence, it is an optimal global solution and overcomes the labeling bias issue in MEMMs.
- General CRFs require more complex inference, and are therefore less commonly used for language processing.

2.5 Syntax: Chunking & parsing

- We can now start a syntactic analysis of a sentence, i.e., the analysis that focuses on understanding the logical meaning of sentences or of parts of sentences
- In other words, syntactic analysis analyzes the relationship between words and the grammatical structure of sentences.
- This can be done by using two methods:
 - **Full parsing:** Produce a full parse tree for a sentence using a parser, a grammar, and a lexicon;
 - **Chunking (also called partial, shallow or robust parsing):** Find **syntactic constituents (chunks)** like Noun Phrases (NPs) or Verb Phrases (VPs) within a sentence.
 - **Chunks:** Non-overlapping regions of text

Example:

1) Part of speech

N = noun

V = verb

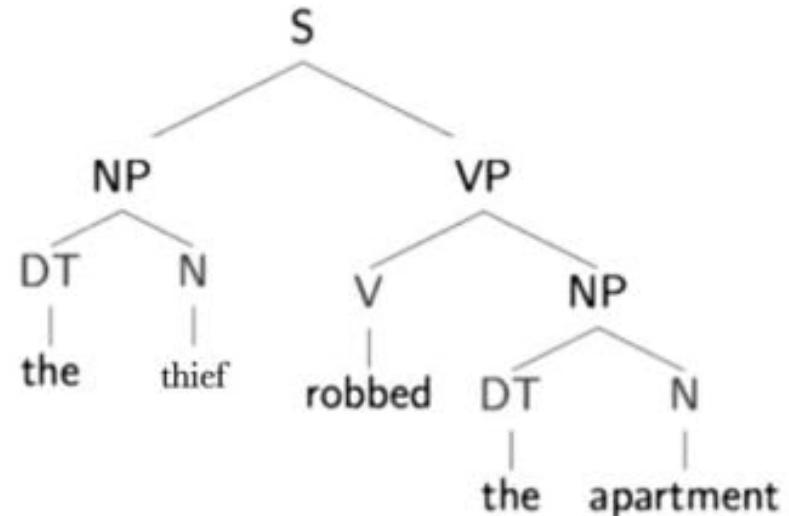
DT = determiner

2) Phrases

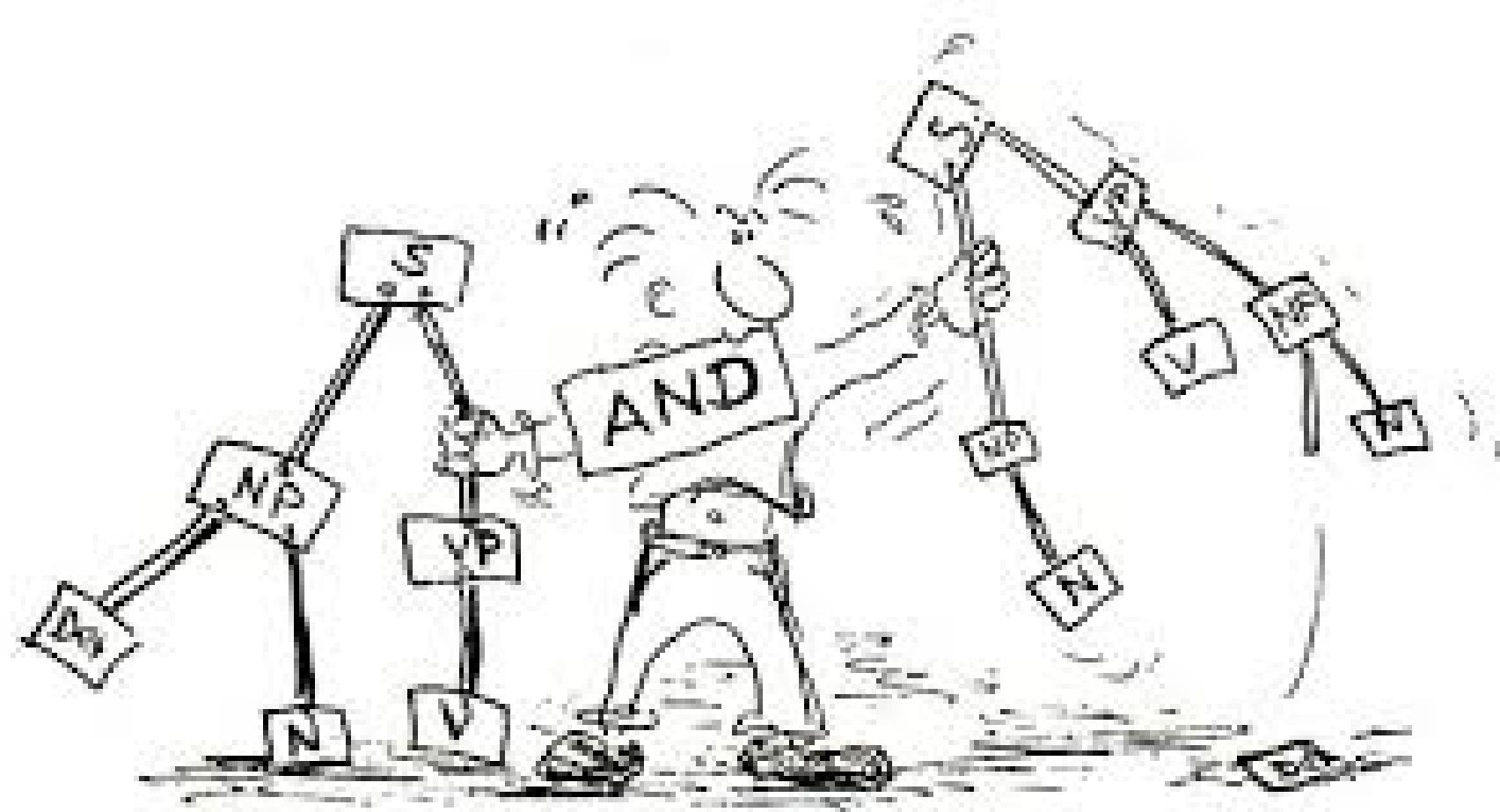
Noun Phrases: : "the thief", "the apartment"

Verb Phrases: "robbed the apartment"

Sentence: "the burglar robbed the apartment"



3) Relationships

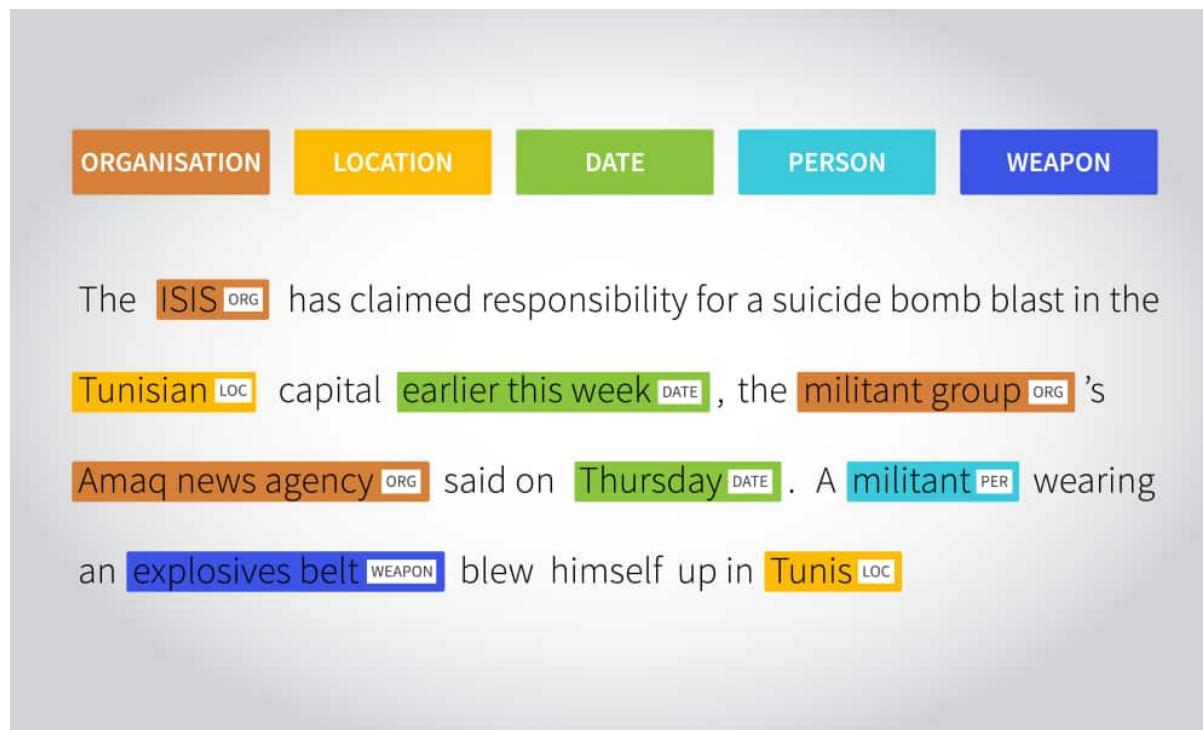


- **Remark:** Producing a full parse tree often fails due to
 - grammatical inaccuracies;
 - novel words;
 - bad tokenization;
 - wrong sentence splits;
 - errors in POS tagging;
 - etc.

Hence, chunking are more commonly used in practice.

- **Question:** But what can we do with chunks?

- **Answer:** Chunking is very important when you want to extract information from text such as person names, locations, organisations, dates, etc.
 - In text mining, this is called **Named Entity Recognition (NER)**.



- In what follows, a brief overview is given on NER.

Named Entity Recognition (NER)

- NER (also known as entity chunking, extraction or identification) is a subtask of information extraction.
 - It seeks to identify and classify key information (entities) mentioned in text into pre-defined categories such as person names, organizations, locations, percentages, time expressions, etc.

I hear Berlin is wonderful in the winter

- To achieve NER, two steps are standardly performed:
 1. **Detect a named entity**, i.e., a single word (chink) or string of words (chunk);
 2. **Categorize the entity**, e.g., as a person name, organisation, etc.

Example: Consider the sentence

" Jim bought 300 shares of Acme Corp. in 2006"



[Jim] bought 300 shares of [Acme Corp.] in [2006]



[Jim]PERSON bought 300 shares of [Acme Corp.]ORG. in [2006]TIME

- **Question:** What is the difference between POS tagging and NER?
- **Answer:**
 - In POS tagging, each POS tag is attached to a single word, while NER tags can be attached to multiple words.
 - Thus NER involves not only detecting the type of named entity, but also the named entity boundaries. Therefore, NER is often seen as a more difficult task to perform compared to POS tagging.
 - Generally, NER uses certain POS tags as an input feature.
- **Question:** But what POS tags are standard used that accounts for (1) the different named entity categories and (2) the named entity boundaries?

- **Answer: Inside-outside-beginning (IOB) tagging**
 - It is a tagging format that is used for tagging tokens in a chunking task such as NER;
 - These tags are similar to POS tags but give us information about the location of the word in the chunk;
 - The IOB tagging system contains tags of the form:
 - B-{CHUNK TYPE} - for the word in the Beginning chunk;
 - I-{CHUNK TYPE} - for the word Inside the chunk;
 - O - Outside any chunk.

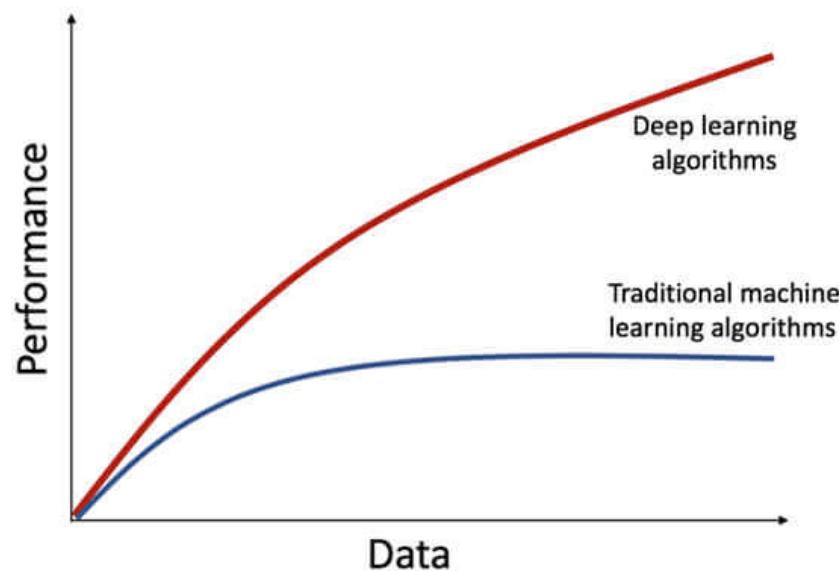
Kevin Durant plays basketball for the Brooklyn Nets

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
B-per I-per O O O O B-org I-org

- There, of course, exists other annotation schemes, e.g.,
 - **IO**: The simplest scheme. In this scheme, each token from the dataset is assigned one of two tags: An inside tag (I) and an outside tag (O). The I tag is for named entities, whereas the O tag is for normal words. This scheme has a limitation, as it cannot correctly encode consecutive entities of the same type.
 - **IOE**: This scheme works nearly identically to IOB, but it indicates the end of the entity (E tag) instead of its beginning.
 - **IOBES**: An alternative to the IOB scheme is IOBES, which increases the amount of related to the boundaries of named entities. In addition to tagging words at the beginning (B), inside (I), end (E), and outside (O) of a named entity. It also labels single-token entities with tag S.

- **BI**: This scheme tags entities in a similar method to IOB. Additionally, it labels the beginning of non-entity words with the tag B-O and the rest as I-O.
- **IE**: This scheme works exactly like IOE with the distinction that it labels the end of non-entity words with the tag E-O and the rest as I-O.
- **BIES**: This scheme encodes the entities similar to IOBES. In addition, it also encodes the non-entity words using the same method. It uses B-O to tag the beginning of non-entity words, I-O to tag the inside of non-entity words, and S-O for single non-entity tokens that exist between two entities.

- NER can be done using rule-based methods alongside a number of sequence labelling methods, e.g.,
 - **Stochastic:** Linear-chain CRF & MEMM
 - **Neural:** Bidirectional Long Short Term Memory (Bi-LSTM) networks, i.e., a special kind of Recurrent Neural Network (RNN, a deep learning technique), capable of learning long-term dependencies.
- See later.



- **Remark:** A CRF for NER makes use of very similar features to a POS tagger, e.g.,

identity of w_i , identity of neighboring words

embeddings for w_i , embeddings for neighboring words

part of speech of w_i , part of speech of neighboring words

presence of w_i in a **gazetteer**

w_i contains a particular prefix (from all prefixes of length ≤ 4)

w_i contains a particular suffix (from all suffixes of length ≤ 4)

word shape of w_i , word shape of neighboring words

short word shape of w_i , short word shape of neighboring words

gazetteer features

- One feature that is especially useful for locations is a **gazetteer**, i.e., a list of place names, often providing millions of entries for locations with detailed geographical and political informations. This can be implemented as a binary feature indicating a phrase appears in the list.

Example:

Words	POS	Short shape	Gazetteer	BIO Label
Jane	NNP	Xx	0	B-PER
Villanueva	NNP	Xx	1	I-PER
of	IN	x	0	O
United	NNP	Xx	0	B-ORG
Airlines	NNP	Xx	0	I-ORG
Holding	NNP	Xx	0	I-ORG
discussed	VBD	x	0	O
the	DT	x	0	O
Chicago	NNP	Xx	1	B-LOC
route	NN	x	0	O
.	.	.	0	O

- **Question:** And how can we evaluate these NER systems?

- Traditional answer: Evaluation metrics

- For evaluation, custom NER uses the following metrics:

- **Precision:**

$$\text{Precision} = \frac{\#\text{True Positives}}{\#\text{True Positives} + \#\text{False Positives}};$$

- **Recall:**

$$\text{Recall} = \frac{\#\text{True Positives}}{\#\text{True Positives} + \#\text{False Negatives}};$$

- **F1 score:**

$$\text{F1 score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}};$$

- These can be calculated for each entity separately, i.e., **entity-level evaluation**, or for the model collectively, i.e., **model-level evaluation**.

Example: Consider the sentence

"The first party of this contract is John Smith, resident of 5678 Main Rd., City of Frederick, state of Nebraska. And the second party is Forrest Ray, resident of 123-345 Integer Rd., City of Corona, state of New Mexico. There is also Fannie Thomas resident of 7890 River Road, city of Colorado Springs, State of Colorado."

The model extracting entities from this text could have the following predictions:

Entity	Predicted as	Actual type
John Smith	Person	Person
Frederick	Person	City
Forrest	City	Person
Fannie Thomas	Person	Person
Colorado Springs	City	City

- Entity-level evaluation for the person entity:

Key	Count	Explanation
True Positive	2	<i>John Smith and Fannie Thomas</i> were correctly predicted as <i>person</i> .
False Positive	1	<i>Frederick</i> was incorrectly predicted as <i>person</i> while it should have been <i>city</i> .
False Negative	1	<i>Forrest</i> was incorrectly predicted as <i>city</i> while it should have been <i>person</i> .

- Precision = $\frac{2}{(2+1)} = 0.67$

- Recall = $\frac{2}{(2+1)} = 0.67$

- F1 score = $\frac{(2 \cdot 0.67 \cdot 0.67)}{(0.67 + 0.67)} = 0.67$

- Entity-level evaluation for the city entity:

Key	Count	Explanation
True Positive	1	<i>Colorado Springs</i> was correctly predicted as <i>city</i> .
False Positive	1	<i>Forrest</i> was incorrectly predicted as <i>city</i> while it should have been <i>person</i> .
False Negative	1	<i>Frederick</i> was incorrectly predicted as <i>person</i> while it should have been <i>city</i> .

- Precision = $\frac{1}{(1+1)} = 0.5$

- Recall = $\frac{1}{(1+1)} = 0.5$

- F1 score = $\frac{(2 \cdot 0.5 \cdot 0.5)}{(0.5 + 0.5)} = 0.5$

- Model-level evaluation for the collective model

Key	Count	Explanation
True Positive	3	<i>John Smith</i> and <i>Fannie Thomas</i> were correctly predicted as <i>person</i> . <i>Colorado Springs</i> was correctly predicted as <i>city</i> . This is the sum of true positives for all entities.
False Positive	2	<i>Forrest</i> was incorrectly predicted as <i>city</i> while it should have been <i>person</i> . <i>Frederick</i> was incorrectly predicted as <i>person</i> while it should have been <i>city</i> . This is the sum of false positives for all entities.
False Negative	2	<i>Forrest</i> was incorrectly predicted as <i>city</i> while it should have been <i>person</i> . <i>Frederick</i> was incorrectly predicted as <i>person</i> while it should have been <i>city</i> . This is the sum of false negatives for all entities.

- Precision = $\frac{3}{(3+2)} = 0.6$

- Recall = $\frac{3}{(3+2)} = 0.6$

- F1 score = $\frac{(2 \cdot 0.6 \cdot 0.6)}{(0.6 + 0.6)} = 0.6$

- Interpretation of these metrics:

Recall	Precision	Interpretation
High	High	This entity is handled well by the model.
Low	High	The model cannot always extract this entity, but when it does it is with high confidence.
High	Low	The model extracts this entity well, however it is with low confidence as it is sometimes extracted as another type.
Low	Low	This entity type is poorly handled by the model, because it is not usually extracted. When it is, it is not with high confidence.

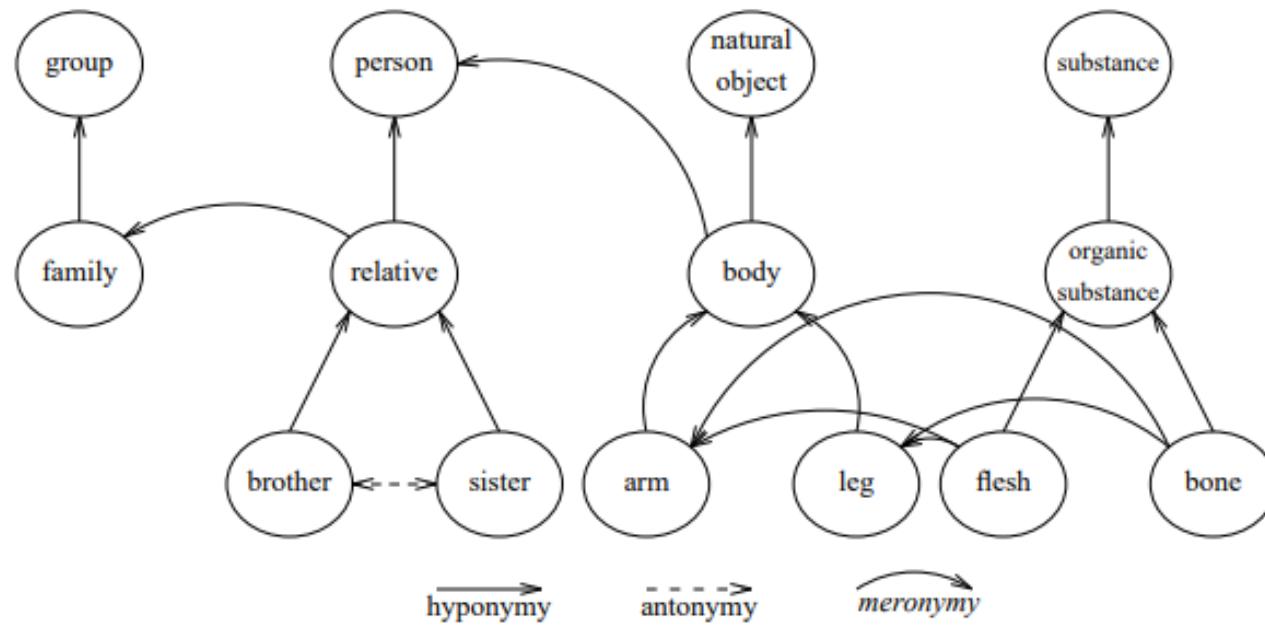
- **Remark:** These metrics don't tell you anything about the strengths & weaknesses of the model and what characteristics of the data are most influential on the model performance, e.g. is the model suffering due to
 - long sentence length?
 - too many (or too few) entities?
 - too highly ambiguous entities?

Therefore, alternative/additional novel evaluation techniques have been proposed in the literature, e.g., (*), that tackles some of these questions

- (*) Fu, J., Liu, P., and Neubig, G. (2020). Interpretable Multi-dataset Evaluation for Named Entity Recognition. *In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 6058–6069, Online. Association for Computational Linguistics.

2.6 Semantics

- Now that we have synthetic information, we can start to address the meaning of words;
- A useful tool to address this: **WordNet**



- WordNet is a large lexical database of English.
- Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept;
- Synsets are interlinked by means of conceptual-semantic and lexical relations;

WordNet Search - 3.1
[- WordNet home page](#) - [Glossary](#) - [Help](#)

Word to search for:

Display Options:

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations
 Display options for sense: (gloss) "an example sentence"

Noun

- S: (n) [statistic](#) (a datum that can be represented numerically)
- S: (n) [statistics](#) (a branch of applied mathematics concerned with the collection and interpretation of quantitative data and the use of probability theory to estimate population parameters)
 - [direct hyponym](#) / [full hyponym](#)
 - [part meronym](#)
 - [domain term category](#)
 - [direct hypernym](#) / [inherited hypernym](#) / [sister term](#)
 - S: (n) [applied mathematics](#), [applied math](#) (the branches of mathematics that are involved in the study of the physical or biological or sociological world)
 - S: (n) [mathematics](#), [math](#), [maths](#) (a science (or group of related sciences) dealing with the logic of quantity and shape and arrangement)
 - S: (n) [science](#), [scientific discipline](#) (a particular branch of scientific knowledge) "*the science of genetics*"
 - S: (n) [discipline](#), [subject](#), [subject area](#), [subject field](#), [field](#), [field of study](#), [study](#), [bailiwick](#) (a branch of knowledge) "*in what discipline is his doctorate?*";

- It superficially resembles a thesaurus, in that groups words together based on their meaning;
- However, there are some important distinctions:
 - WordNet interlinks not just word forms - strings of letters - but specific senses of words. As a result, words that are found in close proximity to one another in the network are semantically disambiguated;
 - WordNet labels the semantic relations among words, whereas the groupings of words in a thesaurus does not follow any explicit pattern other than meaning similarity;
- It is freely and publicly available:
<http://wordnetweb.princeton.edu/perl/webwn>

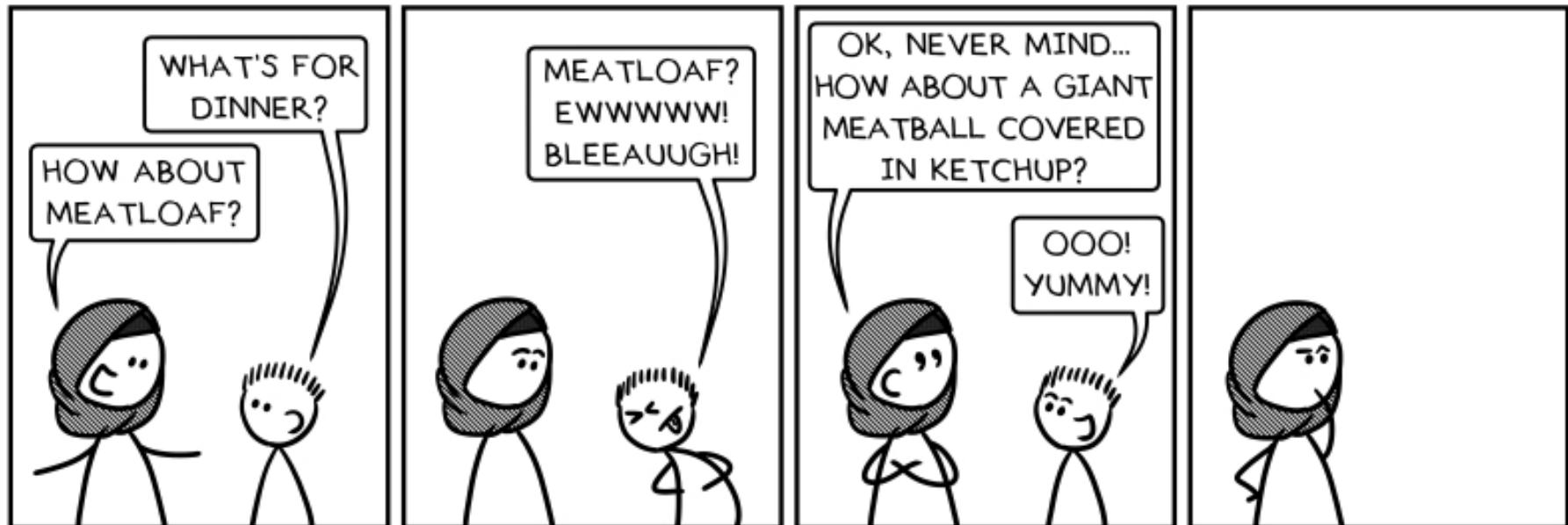
3. Attribute Generation

- Since traditional machine learning and data mining techniques are generally not designed to deal directly with textual data, attribute generation (also known as feature construction or engineering) is an important preliminary step in text mining;
- Feature construction in text mining consists of various techniques and approaches which convert textual data into a feature-based representation;
- The choice hereby depends on
 - the task that is being addressed;
 - the data mining algorithms used;
 - the nature of the dataset in question.

- **Examples:**

- When your task/algorithm requires you to work with the structure of the language, the use of **grammatical features** like **POS tagging**, **NER** and **parsing** can be used;
- When you have a large corpus and want to simplify them into fewer words for your algorithm, **statistical features** like **TF-IDF** can be considered;
- When your algorithm require to work with the semantics of the language, i.e., the study of the meaning of words, phrases and sentences, **vectorized embeddings** like **word2vec** can be used.

SEMANTICS



©2019 Mya Lixian Gosling & Andrea Annaba

keepcalmandmuslimon.com

- In what follows, we will focus on the latter two, i.e., **statistical features & vectorized embeddings**, in the context of word semantics.

3.1 One-hot encoding

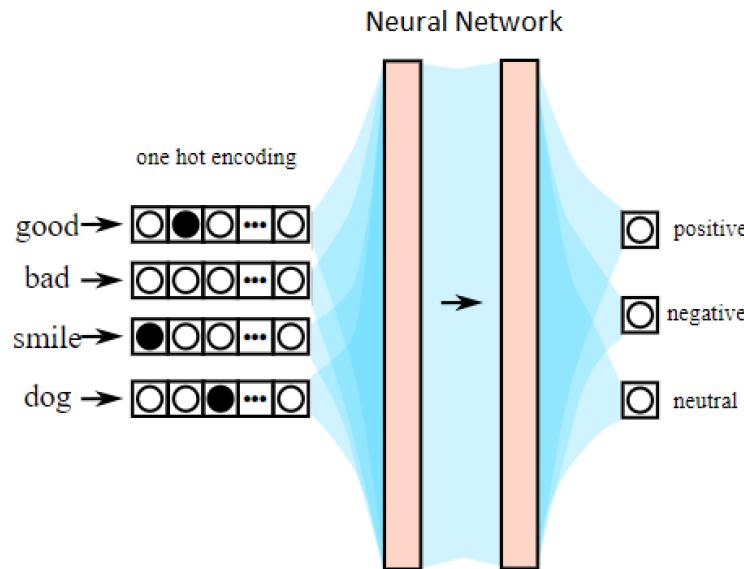
- **One-hot encoding** = Represent every word of your document as a $R^{|V| \times 1}$ vector with all 0s and one 1 at the index of that word in the sorted language;
- In this notation, $|V|$ presents the size of the vocabulary V
 - **Reminder:** Vocabulary = a total number of distinct words which form your corpus
- Word vectors in this type of encoding would appear as the following:

$$w^{aardvark} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, w^a = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, w^{at} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \dots w^{zebra} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

- As result, each word is represented as a completely independent entity, and the list of words in a sentence can be denoted as an array of vectors or a matrix.

- **Advantages:**

- Highly intuitive and straightforward to compute;
- Can be used as input layer within deep learning techniques like Neural Networks

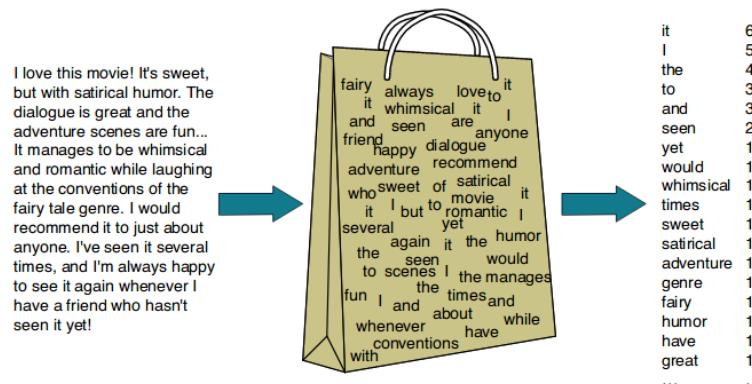


- **Disadvantages:**
 - High-dimensional and sparse representations
 - No fixed size (each document is of different length which creates an array of vectors of different sizes)
 - Does not capture semantics, in sense that it does not give any notion of similarity, e.g.,

$$(w^{hotel})^T w^{motel} = (w^{hotel})^T w^{cat} = 0$$

3.2 Bag of Words

- **Bag of Words (BoW)** = Representation of text that describes the occurrence of words within a document;
- It involves two aspects:
 - A vocabulary of known words
 - A measure of the presence of known words, e.g., counts



- **Example:** Consider the following lines of text from the book "A Tale of Two Cities" by Charles Dickens:

It was the best of times,
it was the worst of times,
it was the age of wisdom,
it was the age of foolishness.

For this small example, let's treat each line as a separate "document" and the 4 lines as our entire corpus of documents.

Step 1: Design the vocabulary

The unique words here (ignoring case and punctuation) are:

- "it"
- "times"
- "was"
- "worst"
- "the"
- "age"
- "best"
- "wisdom"
- "of"
- "foolishness"

⇒ Vocabulary of 10 words from a corpus containing 24 words.

Step 2: Create document vectors

The next step is to score the words in each document. Because $|V| = 10$, a fixed-length document representation of 10 can be used, with one position in the vector to score each word. The simplest scoring method is to mark the presence of words as a boolean value, i.e., 1 for present and 0 for absent.

The scoring of document 1 ("It was the best of times") would look as follows:

- "it" = 1
- "was" = 1
- "the" = 1
- "best" = 1
- "of" = 1
- "times" = 1
- "worst" = 0
- "age" = 0
- "wisdom" = 0
- "foolishness" = 0

Some additional simple scoring methods include:

- **Frequencies:** Calculate the frequency that each words appears in a document out of all words in the document.
- **Counts:** Count the number of times each word appears in a document;



This can be stored in a so-called **Word-Document Matrix**:

		Document			
		D_1	D_2	\dots	D_N
Vocabulary V	v_1	$n_{1,1}$	$n_{1,2}$	\dots	$n_{1,N}$
	v_2	$n_{2,1}$	$n_{2,2}$	\dots	$n_{2,N}$
	\vdots	\vdots	\vdots	\ddots	\vdots
		$v_{ V }$	$n_{ V ,1}$	$n_{ V ,2}$	\dots
					$n_{ V ,N}$

- **Advantages:**

- Simple and intuitive;
- Fix size vector

- **Disadvantages:**

- Out of vocabulary situation, i.e., (new) words in a new sentence that are not present in the predefined vocabulary are simply ignored;
- * Possibility for sparsity, e.g., when having a large vocabulary and few repeated words;
- * Difficult to estimate the semantics of the document.

→ **Possible solution for (*):** A vocabulary of grouped words, i.e., **N-grams**

3.3 Term Frequency - Inverse Document Frequency

- A problem with scoring word frequency is that highly frequent words start to dominate in the document, but may not contain as much "informational content" as rarer but perhaps more domain specific words

A word cloud visualization showing the frequency of words in a document. The most prominent word is "the" in large green font. Other significant words include "climate", "change", "greenhouse", "summit", "nations", "convention", "warming", "agreement", "rise", "sea", "are", "parties", "wollen", "co2", "as", "a", "of", "on", "level", "earth", "in", "for", "with", "countries", "about", "by", "to", "global", "ice", "be", "from", "at", "paris", "is", "change", "have", "would", "it", "report", "action", "united", "that", "emissions", and "this". The size of each word corresponds to its term frequency in the document.

- **Solution: Term Frequency - Inverse Document Frequency (TF-IDF)**
 - **Idea:** Rescale the frequency of words by how often they appear in all documents, such that the scores for frequent words like "the" (that are also frequent across all documents) are penalized;
 - **TF-IDF** rescales these frequencies by assigning weights to the words based on two aspects:
 - **Term Frequency (TF):** A scoring of the frequency of the word in the current document;
 - **Inverse Document Frequency (IDF):** A scoring of how rare the word is across documents

- The **TF-IDF** weightage of a word in the document is calculated as the product of TF and IDF of that word:

$$w_{i,j} = \underbrace{\text{tf}_{i,j}}_{TF} \cdot \underbrace{\log\left(\frac{N}{\text{df}_i}\right)}_{IDF},$$

where:

- $\text{tf}_{i,j}$ = number of times a specific word v_i appeared in document D_j , i.e., $n_{i,j}$, divided by the total number of words (say K_j) in document D_j :

$$\text{tf}_{i,j} = \frac{n_{i,j}}{\sum_{k=1}^{K_j} n_{k,j}}.$$

- N = total number of documents;
- df_i = number of documents containing word v_i .

- **Interpretation:** The IDF of a rare term is high, whereas the IDF of a frequent term is likely to be low.
- **Our example:**

(Document 1) It was the best of times,
(Document 2) it was the worst of times,
(Document 3) it was the age of wisdom,
(Document 4) it was the age of foolishness.

- Step 1: Word-Document Matrix^T

		Vocabulary									
		it	was	the	best	of	times	worst	age	wisdom	foolishness
Document	1	1	1	1	1	1	1	0	0	0	0
	2	1	1	1	1	0	1	1	1	0	0
	3	1	1	1	1	0	1	0	0	1	1
	4	1	1	1	1	0	1	0	0	1	0

- **Step 2:** Find TF

		Vocabulary									
		it	was	the	best	of	times	worst	age	wisdom	foolishness
TF Doc.	1	0.16	0.16	0.16	0.16	0.16	0.16	0	0	0	0
	2	0.16	0.16	0.16	0	0.16	0.16	0.16	0	0	0
	3	0.16	0.16	0.16	0	0.16	0	0	0.16	0.16	0
	4	0.16	0.16	0.16	0	0.16	0	0	0.16	0	0.16

- **Step 3:** Find IDF

		Vocabulary									
		it	was	the	best	of	times	worst	age	wisdom	foolishness
IDF	0	0	0	$\log(4)$	0	$\log(2)$	$\log(4)$	$\log(2)$	$\log(4)$	$\log(4)$	$\log(4)$
	1	1	1	1	1	1	1	1	1	1	1

- **Step 4: TF-IDF**

		Vocabulary									
		it	was	the	best	of	times	worst	age	wisdom	foolishness
TF-IDF Doc.	1	0	0	0	0.10	0	0.05	0	0	0	0
	2	0	0	0	0	0	0.05	0.10	0	0	0
	3	0	0	0	0	0	0	0	0.05	0.10	0
	4	0	0	0	0	0	0	0	0.05	0	0.10

- **Advantages:**

- Simple and computational inexpensive to calculate;
- Simple starting point for similarity calculations, like cosine similarity

- **Disadvantages:**

- Memory-inefficiency (TF-IDF can suffer from the curse of dimensionality);
- Does not capture position in text, semantics, co-occurrences in different documents.

3.4 Pointwise Mutual Information

- While TF-IDF can be used to address the importance of words in documents, hereby penalizing too frequent unimportant words, the **co-occurrence of words** across documents still remains unaddressed;
- To account for the co-occurrences of words in different documents, a **Word-Word Co-occurrence Matrix** can be considered;
 - Our example:

(Document 1) It was the best of times,
(Document 2) it was the worst of times,
(Document 3) it was the age of wisdom,
(Document 4) it was the age of foolishness.

		Vocabulary									
		it	was	the	best	of	times	worst	age	wisdom	foolishness
Vocabulary	it	0	4	0	0	0	0	0	0	0	0
	was	4	0	4	0	0	0	0	0	0	0
	the	0	4	0	1	0	0	1	2	0	0
	best	0	0	1	0	1	0	0	0	0	0
	of	0	0	0	1	0	2	0	0	1	1
	times	0	0	0	0	2	0	0	0	0	0
	worst	0	0	1	0	0	0	0	0	0	0
	age	0	0	2	0	0	0	0	0	0	0
	wisdom	0	0	0	0	1	0	0	0	0	0
	foolishness	0	0	0	0	1	0	0	0	0	0

- Oftentimes, two extra characters, i.e., $< S >$ and $< E >$, are added to this matrix, to address the start and end of a sentence, respectively.

E.g., $< S >$ - *it*: 4; $< S >$ - *wisdom*: 0; *times* - $< E >$: 2

- **Question:** But how can we measure semantic similarity between words?
- **First (Naive) Answer: Word-Word Co-occurrence Matrix**
- This, however, is not the best measure of similarity between 2 words since it is based on raw frequency, and hence is very skewed
 - **Our example:** "it" and "is" are very frequent, but maybe not the most discriminative

- We'd rather have a quantity which measures how much likely is it for 2 words to occur in a window, compared with pure chance.
- **Second Answer: Pointwise Mutual Information (PPMI)?**
 - **Pointwise Mutual Information (PMI)** = one of the most important concepts in NLP;
 - It measures of how often a target word w and a context word c occur, compared with what you would expect if they were independent:

$$\text{PMI}(w, c) = \ln \left[\frac{P(w, c)}{P(w) \cdot P(c)} \right].$$

- $P(w, c)$ tells us how often we observed the two words together;
- $P(w) \cdot P(c)$ tells us how often we would expect the two words to co-occur assuming they each occurred independently.

- **Characteristics:**

- PMI is symmetric, i.e., $\text{PMI}(w, c) = \text{PMI}(c, w)$;
- Due to Bayes' theorem:

$$\begin{aligned}\text{PMI}(w, c) &= \ln \left[\frac{P(w | c)}{P(w)} \right] \\ &= \ln \left[\frac{P(c | w)}{P(c)} \right].\end{aligned}$$

- Range of PMI:

$$\left[-\infty; \underbrace{\min \{-\ln [P(w)], -\ln [P(c)]\}}_{(*)} \right]$$

(*) PMI maximizes when w and x are perfectly associated, i.e., $P(w | c) = P(c | w) = 1$.

- PMI follows the chain rule, i.e.,

$$\text{PMI}(w, cb) = \text{PMI}(w, c) + \text{PMI}(w, b | c).$$

Proof:

$$\begin{aligned}
\text{PMI}(w, c) + \text{PMI}(w, b | c) &= \ln \left[\frac{P(w, c)}{P(w) \cdot P(c)} \right] + \ln \left[\frac{P(w, b | c)}{P(w | c) \cdot P(b | c)} \right] \\
&= \ln \left[\frac{P(w, c)}{P(w) \cdot P(c)} \cdot \frac{P(w, b | c)}{P(w | c) \cdot P(b | c)} \right] \\
&= \ln \left[\frac{P(w | c) \cdot P(c) \cdot P(w, b | c)}{P(w) \cdot P(c) \cdot P(w | c) \cdot P(b | c)} \right] \\
&= \ln \left[\frac{P(w, cb)}{P(w) \cdot P(cb)} \right] \\
&= \text{PMI}(w, cb)
\end{aligned}$$

- **Main limitations:**

- PMI can take both positive and negative values, and has no fixed bounds, which makes it harder to interpret;
- PMI has the problem of being bias toward infrequent event, i.e., very rare words tend to have very high PMI values, but in applications such as measuring the word similarity, it is preferable to have "a higher score to pairs whose relatedness is supported by more evidence".
- To address these, several variations have been proposed, i.e., [1] **Positive PMI (PPMI)**, [2] **Normalize PMI (NPMI)**, & [3] **PMI^k family**;
- In what follows, we will discuss these!

[1] Positive PMI (PPMI):

- The **Positive PMI (PPMI)** measure is defined by setting negative values of PMI to zero, i.e.,

$$\text{PPMI}(w, c) = \max \left\{ \ln \left[\frac{P(w, c)}{P(w) \cdot P(c)} \right]; 0 \right\}.$$

- **Motivation:**

1. Negative PMI values (implying words are co-occurring less often than we would expect by chance) tend to be unreliable unless our corpora are "enormous";

2. It's not clear whether it's even possible to evaluate such scores of "unrelatedness" with human judgments;
3. It avoids to deal with $-\infty$ values for events that never occur together.

[2] Normalize PMI (NPMI)

- PMI can be normalized between $[-1; 1]$ resulting in -1 for never occurring together, 0 for independence, and $+1$ for complete co-occurrence, resulting in the so-called **Normalize PMI (NPMI)**:

$$\text{NPMI}(w, c) = \frac{\text{PMI}(w, c)}{-\ln[P(w, c)]}.$$

[3] PMI^k family

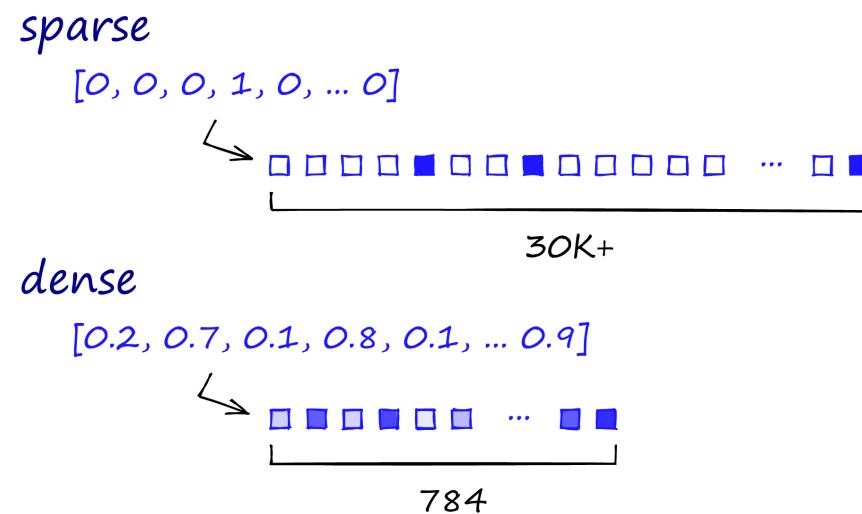
- To account for the PMI problem of being bias toward infrequent events, the **PMI^k measure** (for $k = 2, 3, \dots$) can be used:

$$\text{PMI}^k(w, c) = \ln \left[\frac{P(w, c)^k}{P(w) \cdot P(c)} \right].$$

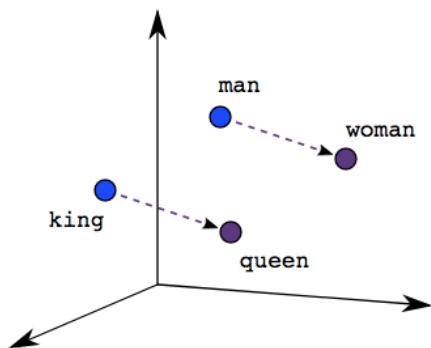
- **Motivation:** The additional factors of $P(w, c)$ inside the logarithm are intended to correct the bias of PMI toward low-frequency events, by boosting the scores of frequent pairs, e.g., Role and Nadif (2011) Handling the impact of low frequency events on co-occurrence-based measures of word similarity: A case study of Pointwise Mutual Information.

3.5 Word embeddings

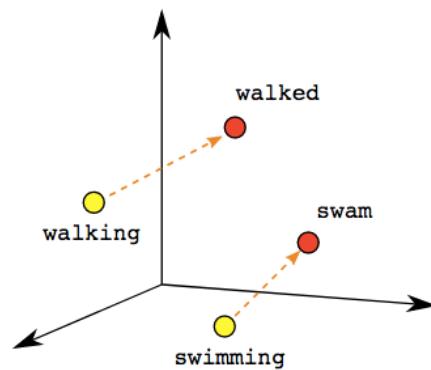
- In the previous sections, e.g., Section 3.1, words are represented as sparse, long vectors with dimensions corresponding to words in the vocabulary of documents in a collection;
- To overcome the problem of high-dimensionality and sparse word representations, a more powerful representation can be considered, i.e., **embeddings**, short dense vectors.



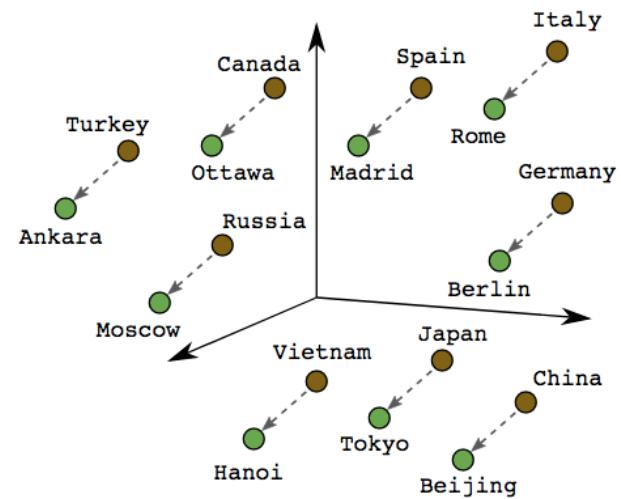
- **Embeddings** translates sparse vectors into a low-dimensional space that preserves semantic relationships.
- **Word embeddings** is a type of word representation that allows words with similar meaning to have a similar representation.



Male-Female

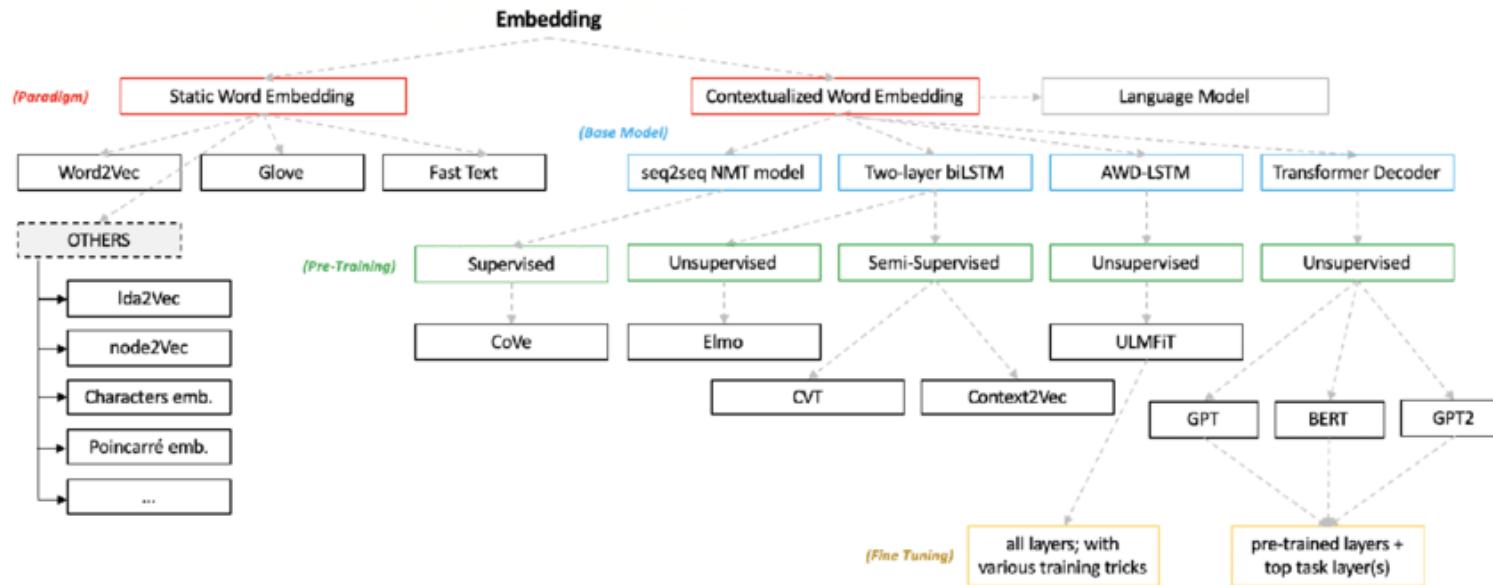


Verb Tense

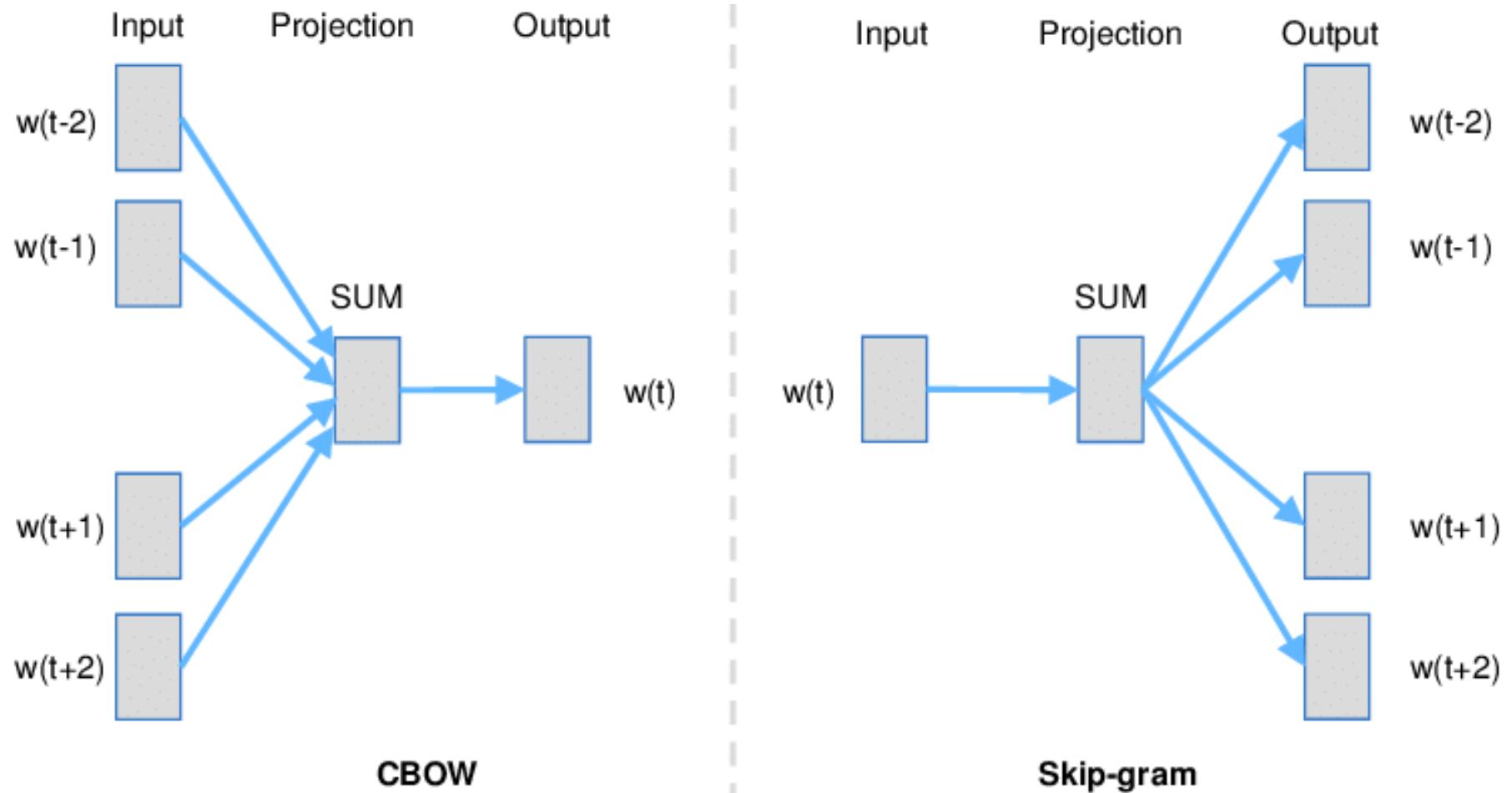


Country-Capital

- There exists two kinds of embeddings:
- **Static embeddings** = One fixed embedding for each word in the vocabulary, with size Q ($Q \ll |V|$);
- **Contextual embeddings** = The vector for each word is different in different contexts.

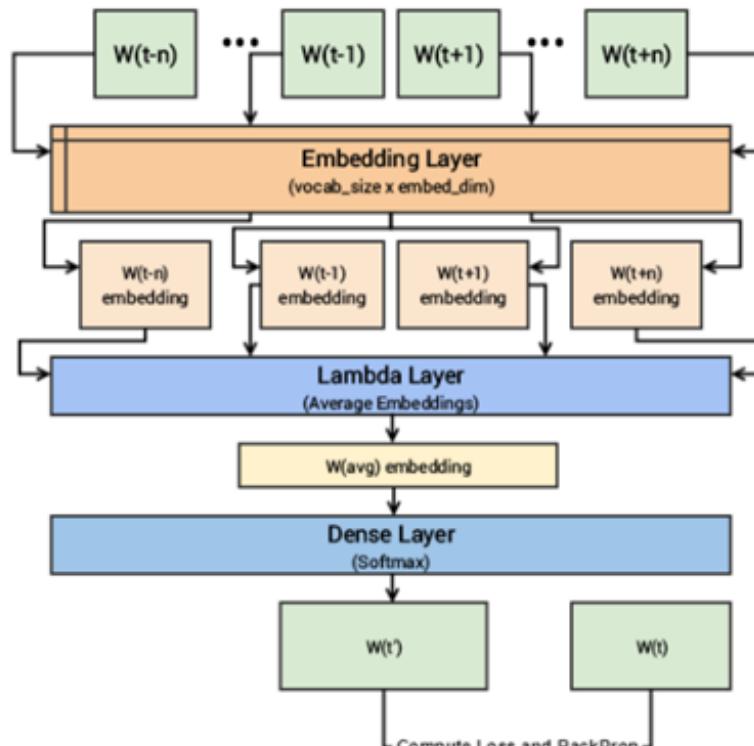


- In this course, the **Word2Vec (static) embeddings** will be explored in detail, including two methods for learning representations of words:
 - **Continuous Bag-of-Words (CBOW) model:**
 - Predicts the middle word w^t based on surrounding context words $w^{t-n}, \dots, w^{t-1}, w^{t+1}, \dots, w^{t+n}$;
 - The context consists of a few words before and after the current (middle) word, specified by a so-called context window size n ;
 - This architecture is called a bag-of-words model as the order of words in the context is not essential.
 - **Skip-gram model:** Predicts words within a certain range before and after the current word in the same sentence.



1) Continuous Bag-of-Words (CBOW) model

- CBOW is a neural network(NN)-based algorithm;
- **Architecture of the NN:**



Visual depiction of the CBOW deep learning model

We breakdown the way this model works in steps:

1. The context words $(w^{t-n}, \dots, w^{t-1}, w^{t+1}, \dots, w^{t+n})$ for context window size n are first encoded as one-hot word vectors $(x^{t-n}, \dots, x^{t-1}, x^{t+1}, \dots, x^{t+n})$, i.e., x^{t-j} represent a $R^{|V| \times 1}$ vector ($\forall j = 1, \dots, n$);
2. The one-hot encoding vectors are passed as an input to an embedding layer (initialized with some random weights), to obtain our embedded word vectors for the context words:

- | | |
|---|---|
| <ul style="list-style-type: none">• $u^{t-n} = W^{(1)} \times x^{t-n}$• ...• $u^{t-1} = W^{(1)} \times x^{t-1}$ | <ul style="list-style-type: none">• $u^{t+1} = W^{(1)} \times x^{t+1}$• ...• $u^{t+n} = W^{(1)} \times x^{t+n}$ |
|---|---|

Here, $W^{(1)}$ represent a $Q \times |V|$ "weight" matrix, which we will be optimized during training;

3. The word embeddings are then passed to a lambda layer where we average out the word embeddings, i.e.,

$$h = \frac{u^{t-n} + \cdots + u^{t-1} + u^{t+1} + \cdots + u^{t+n}}{2n}.$$

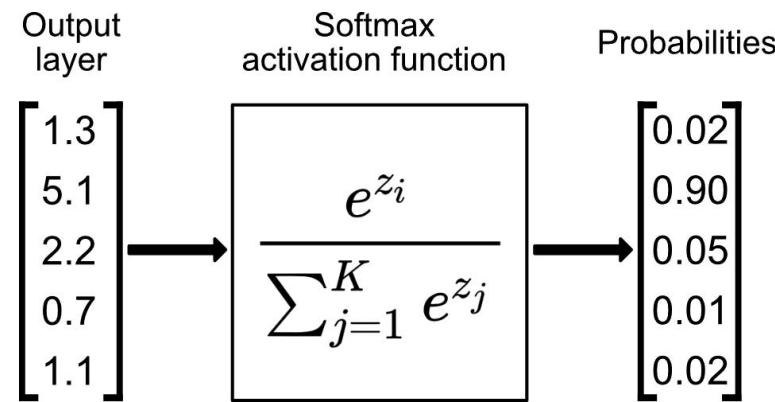
Evidently, h denotes a $R^{Q \times 1}$ vector.

4. Next, a score vector is computed, to ensure that the output will be a $R^{|V| \times 1}$ vector:

$$z = W^{(2)} \times h.$$

Here, $W^{(2)}$ represent a second $R^{|V| \times Q}$ "weight" matrix, which we again (together with $W^{(1)}$) will optimize during training;

5. To ensure that we have an output with probabilities, these scores are passed to a dense SoftMax layer as follows: $\hat{y} = \text{softmax}(z)$:



6. **Goal:** $\hat{y} = x^t$, i.e., the one-hot word vector of target word w^t

- We can extract out the embeddings of the needed words from our embedding layer, once the training of $W^{(1)}$ and $W^{(2)}$ is completed.
- **Question:** How do we train $W^{(1)}$ and $W^{(2)}$ such that the goal above is reached?

- **Answer:** Minimize a specific **loss/cost function** through **gradient descent**

Example:

- Consider the cross entropy $H(\hat{y}, x^t)$ as **loss/cost function**, expressed as

$$H(\hat{y}, x^t) = - \sum_{j=1}^{|V|} x_j^t \cdot \log(\hat{y}_j) = -x_i^t \cdot \log(\hat{y}_i),$$

where i is the index where the correct word's one-hot vector is 1.

Interpretation:

- For a perfect prediction (i.e., $\hat{y}_i = 1$)

$$\rightarrow H(\hat{y}, x^t) = -1 \cdot \log(1) = 0$$

- For a very bad prediction (e.g., $\hat{y}_i = 0.01$)

$$\rightarrow H(\hat{y}, x^t) = -1 \cdot \log(0.011) \approx 4.605$$

- Thus, the objective is to minimize this loss function, in order to optimize $W = (W^{(1)}; W^{(2)})$:

$$\begin{aligned}\hat{W} &= \operatorname{argmin}_W \left\{ -\log [P(w^i | w^{i-n}, \dots, w^{i-1}, w^{i+1}, \dots, w^{i+n})] \right\} \\ &= \operatorname{argmin}_W \left\{ -\log \left[\frac{\exp(W^{i(2)} h)}{\sum_{j=1}^{|V|} \exp(W_j^{i(2)} h)} \right] \right\} \\ &= \operatorname{argmin}_W \left\{ -W^{i(2)} h + \log \left[\sum_{j=1}^{|V|} \exp(W_j^{i(2)} h) \right] \right\}\end{aligned}$$

- To minimize this expression, **gradient descent** can be used to (see Section 2.4).

1) Skip-gram model

- In contrary to the CBOW, the **skip-gram model** will be able to predict the surrounding words $w^{t-n}, \dots, w^{t-1}, w^{t+1}, \dots, w^{t+n}$, given the middle word w^t ;
- Therefore, the setup is largely the same as CBOW, but we essentially swap the input and output;
- **Steps (similar to CBOW steps):**
 1. The middle word (w^t) is first encoded as a one-hot word vector ($x^t \in R^{|V| \times 1}$);

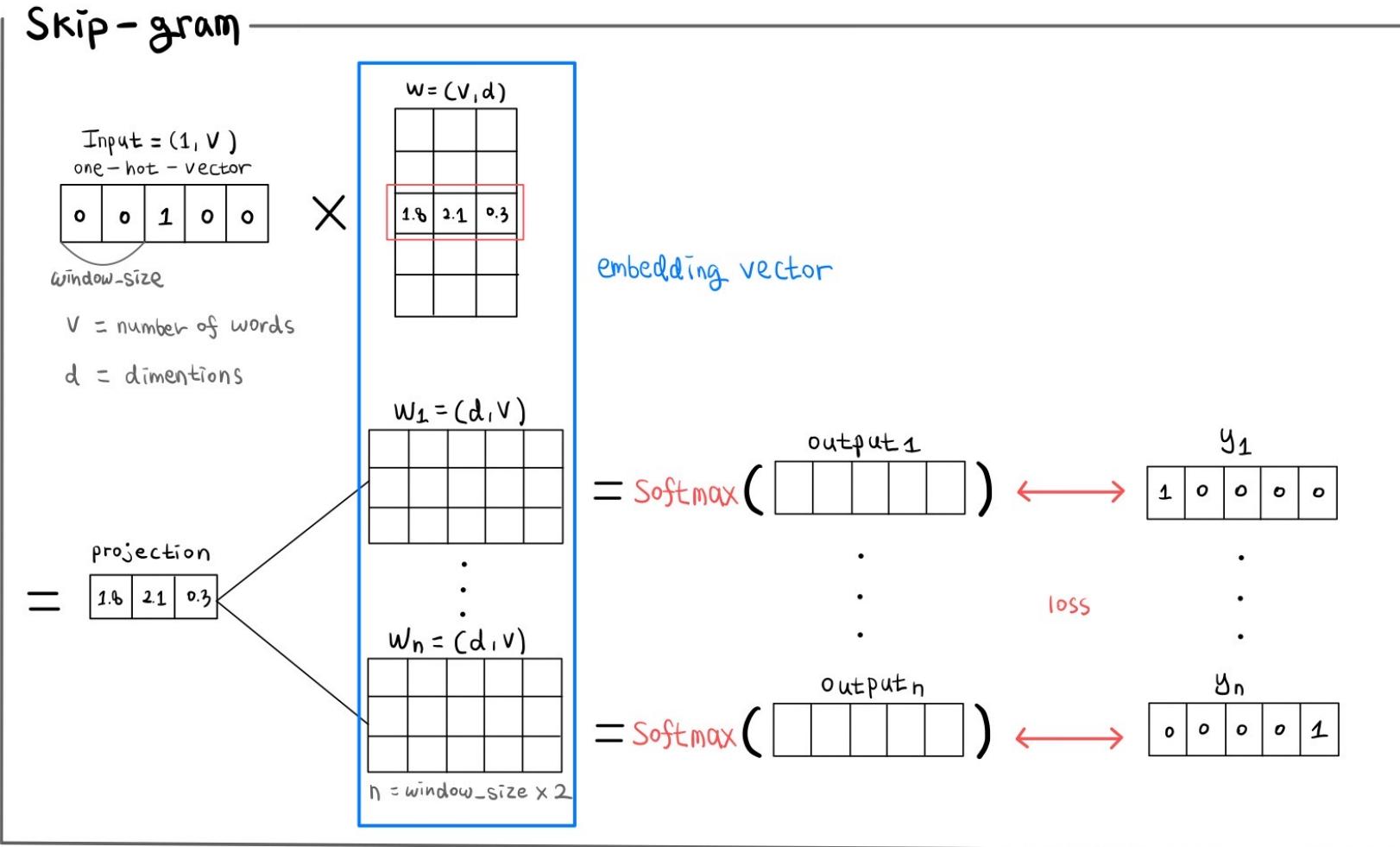
2. The one-hot encoding vector is passed as an input to an embedding layer (initialized with some random weights), to obtain our embedded word vector:

$$u^t = W^{(1)} \times x^t,$$

with similar notations as before;

3. Since there is no averaging, just set: $h = u^t$;
4. Generate $2n$ score vectors $v^{t-n}, \dots, v^{t-1}, v^{t+1}, \dots, v^{t+n}$ using $v = W^{(2)} \times h$;
5. Turn each of the scores into probabilities: $\hat{y} = \text{softmax}(z)$;
6. **Goal:** $(\hat{y}^{t-n}; \dots; \hat{y}^{t-1}; \hat{y}^{t+1}; \dots; \hat{y}^{t+n}) = (x^{t-n}; \dots; x^{t-1}; x^{t+1}; \dots; x^{t+n})$.

- Schematic overview:



- As in CBOW, the following objective function needs to be minimized:

$$\begin{aligned}
 \hat{W} &= \operatorname{argmin}_W \left\{ -\log [P(w^{i-n}, \dots, w^{i-1}, w^{i+1}, \dots, w^{i+n} | w^i)] \right\} \\
 &\stackrel{(\star)}{=} \operatorname{argmin}_W \left\{ -\log \left[\prod_{j=0, j \neq n}^{2n} P(w^{i-n+j} | w^i) \right] \right\} \\
 &= \operatorname{argmin}_W \left\{ -\log \left[\prod_{j=0, j \neq n}^{2n} \frac{\exp(W^{i-n+j(2)} h)}{\sum_{k=1}^{|V|} \exp(W_k^{i-n+j(2)} h)} \right] \right\} \\
 &= \operatorname{argmin}_W \left\{ - \sum_{j=0, j \neq n}^{2n} \left[W^{i-n+j(2)} h + \log \left(\sum_{k=1}^{|V|} \exp(W_k^{i-n+j(2)} h) \right) \right] \right\}
 \end{aligned}$$

- Note:** The **Naive Bayes assumption** is invoked in (\star) , i.e., given the center word w^i , all output words $w^{i-n}, \dots, w^{i-1}, w^{i+1}, \dots, w^{i+n}$ are assumed completely independent.

- **Constraint:** Gradient computation for each step contains the sum of $|V|$ terms, which becomes **computationally huge**, especially when $|V|$ is large.
- **Idea:** Approximate it, such that it becomes computationally less heavy to optimize!
- **Question:** How can we do this?
- **Answer:** Negative sampling!

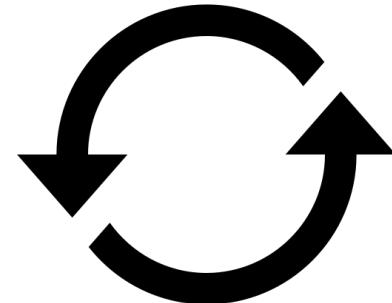
Skipgram

shaft	not	make	a	machine
input		output		
make		shaft		
make		not		
make		a		
make		machine		

Negative Sampling

input word	output word	target
make	shaft	1
make	aaron	0
make	taco	0

- **Principle:** For every training step, instead of looping over the entire vocabulary V , we can just sample several "negative" samples, say S_1, \dots, S_B , each consisting of the target w^t plus a "noise word" w^x ;
- To incorporate this technique into the Skip-gram model, for example, we need to update the:
 - Objective function;
 - Gradients;
 - Update rules.
- It is in fact **optimizing a different objective**, i.e., try to (1) maximize the probability of a word w^t and context w^c being in the corpus data if it indeed is, and (2) maximize the probability of w^t and w^x not being in the corpus data if it indeed is not.



- Consider a word-context pair (w^t, w^c) , $t, c = 1, \dots, |V|$, $t \neq c$, and denote the probability that (w^t, w^c) came from the corpus data by $P(D = 1 | w^t, w^c)$;
- Logically, $P(D = 0 | w^t, w^c)$ will be the probability that (w^t, w^c) did not come from the corpus data;
- **Question:** How to compute these probabilities?
- **Answer:** We can rely on **embedding similarity**, i.e., a word is likely to occur near the target if its embedding vector is similar to the target embedding
 - **Intuition:** Two embedding vectors u^t and u^c are similar if they have a high **dot product**:

$$\text{Similarity}(w^t, w^c) \approx u^t \cdot u^c.$$

- To turn the dot product into a probability, the **sigmoid function** $\sigma(w^t \cdot w^c)$ will be used:

$$P(D = 1 \mid w^t, w^c) = \sigma(w^t \cdot w^c) = \frac{1}{1 + \exp(-w^t \cdot w^c)}$$

$$P(D = 0 \mid w^t, w^c) = \sigma(-w^t \cdot w^c) = \frac{1}{1 + \exp(w^t \cdot w^c)}$$

- Skip-gram relies on the simplifying assumption that all context words are independent, allowing us to calculate the following probability:

$$P(D = 1 \mid w^t; w^{t-n}, \dots, w^{t-1}, w^{t+1}, \dots, w^{t+n}) = \prod_{j=0, j \neq n}^{2n} \sigma(w^t \cdot w^{t-n+j})$$

$$P(D = 0 \mid w^t; w^{t-n}, \dots, w^{t-1}, w^{t+1}, \dots, w^{t+n}) = \prod_{j=0, j \neq n}^{2n} \sigma(-w^t \cdot w^{t-n+j})$$

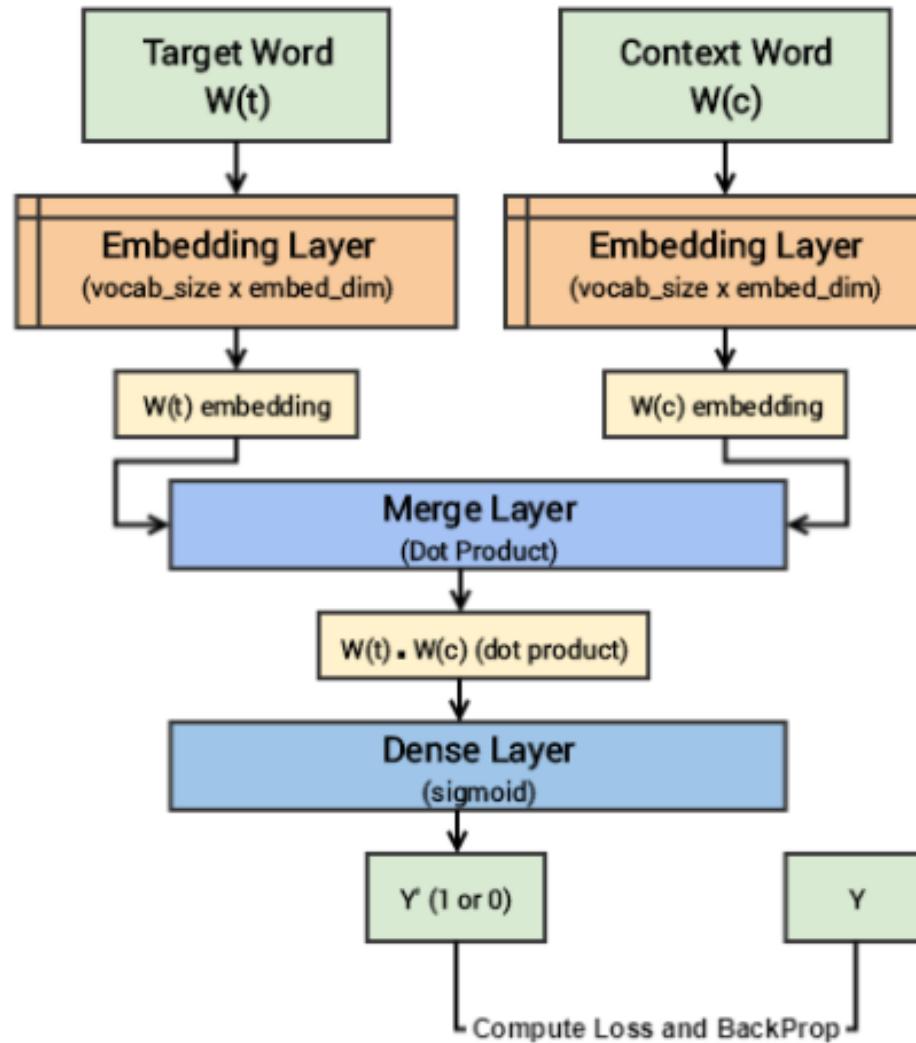
- Taking a simple maximum likelihood approach of (1) and (2), the MLE is found as follow:

$$\hat{W} = \operatorname{argmax}_W \left\{ \underbrace{\left[\prod_{j=0, j \neq n}^{2n} P(D = 1 \mid w^t, w^{t-n+j}) \right]}_{(1)} \cdot \underbrace{\left[\prod_{(w^t, w^x) \in S_{1, \dots, B}} P(D = 0 \mid w^t, w^x) \right]}_{(2)} \right\}$$

$$\begin{aligned}
&= \operatorname{argmax}_W \left\{ \left[\sum_{j=0, j \neq n}^{2n} \log (P(D = 1 \mid w^t, w^{t-n+j})) \right] \right. \\
&\quad \cdot \left. \left[\sum_{(w^t, w^x) \in S_{1, \dots, B}} \log (1 - P(D = 1 \mid w^t, w^x)) \right] \right\} \\
&= \operatorname{argmax}_W \left\{ \left[\sum_{j=0, j \neq n}^{2n} \log \left(\frac{1}{1 + \exp(-u^t \cdot u^{t-n+j})} \right) \right] \right. \\
&\quad \cdot \left. \left[\sum_{(w^t, w^x) \in S_{1, \dots, B}} \log \left(\frac{1}{1 + \exp(u^t \cdot u^x)} \right) \right] \right\}
\end{aligned}$$

- To maximize this expression, **gradient descent** can be used (Section 2.4).
- After training $W = (W^{(1)}; W^{(2)})$, the embeddings of the needed words can be extracted from the embedding layer.

- Schematic overview of Skip-gram model with Negative sampling:



4. Attribute Selection

- Depending on the **task**, and therefore the **chosen text mining technique** used for it, different attributes can be chosen;
- In language models (Section 5.1), for example, one often rely on word embeddings attributes, for reflecting the semantic nature of words;
- In sentiment analysis, i.e., a part within topic modelling (Section 5.2), one could rely on data attributes (e.g., number of words in a sentence, number positive lexicon words in a sentence, etc.), or even word embeddings.

5. Text Mining Techniques

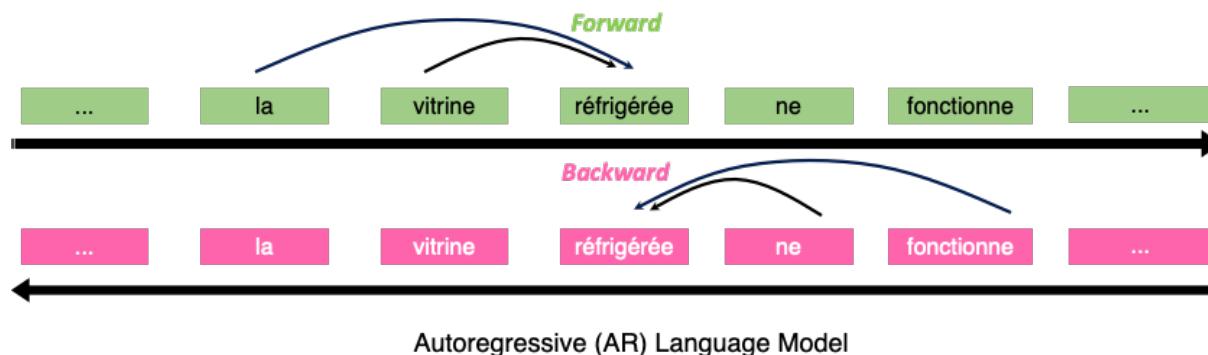
- As seen in the introduction part of this course (Section 1.4), text mining is used in many applications, ranging from **text summarization**, **sentiment analysis**, to **chatbots** like ChatGPT;
- In what follows, we will describe 2 important applications of text mining, i.e.,
 - **Language Models**
 - **Topic modelling**

5.1 Language Models

- While CBOW and Skip-grams model are primarily used to derive word embeddings from their hyperparameters, they essentially predict word in context;
- Therefore, they can be seen as simple language models
 - **Language model (LM):** Probabilistic model of word sequences used for word prediction

Examples:

- N -gram models predicting the next word from the previous $n - 1$ words = **AUTOREGRESSIVE LM**;



- Predicting a words given left and right context (e.g., CBOW, BERT) = **MASKED LM**;



- LMs are often used in practice, to simplify our lifes, e.g.,
 - Speech recognition;



- Language generation (e.g., ChatGPT; Machine translation);

The image shows a dark-themed chat interface. At the top, a user icon (a person's face) is followed by the text "explain quantum theory to a child in the style of snoop dogg". Below this, a AI icon (a green square with a white swirl) is followed by a multi-line response poem:

Yo, little homie, let me break it down for you
Quantum theory, it's a crazy thing, it's true
See, the world is made up of tiny, tiny particles
That are always moving, and changing, and actin' wild

These particles, they don't always behave like we think
Sometimes they're waves, and sometimes they're points, it's a trip
They can be in two places at once, and they can be in two states
It's a mind-blowing concept, but that's the quantum fate

So don't be scared, little homie, it's all good
Quantum theory, it's just the way the world works, it's understood
It's a wild ride, but it's a cool one too
Quantum theory, it's the way the world does what it do.

At the bottom left is a small button labeled "ALT". At the bottom right are two small icons: a thumbs-up and a thumbs-down.

- Spell correction

The screenshot shows a 'Commonly confused words' section from Grammarly. At the top left is the text 'Commonly confused words'. At the top right is a teal 'G' logo. Below this, a sentence is displayed: 'I wasnt sure what to except.' The word 'wasnt' and 'except' are highlighted with red boxes. A callout box below the sentence contains the corrected text: 'I wasn't sure what to expect.'

- In this chapter, several types of LMs will be considered, i.e., **(1) classical LMs based on word frequency count: N-gram models & (2) LM based on NN architectures.**

(1) Classical LMs based on word frequency count: N-gram models

- Similar within POS tagging (Section 2.4), the N-gram approach can also be used to predict words;
- Assume a sentence, composed of a sequence of words $W = w_1, w_2, \dots, w_n$
- **Goal:** Model the joint density of an entire sequence of words, i.e., $P(w_1, w_2, \dots, w_n) (= (\star))$
- Using the chain rule of probability, we get:

$$\begin{aligned} P(w_1, w_2, \dots, w_n) &= P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1, w_2)\dots P(w_n \mid w_1, w_2, \dots, w_{n-1}) \\ &= \prod_{k=1}^n P(w_k \mid w_{1:k-1}) \end{aligned}$$

- **Intuition of the N-gram model:** Instead of computing the probability of a word given its ENTIRE history, i.e., $P(w_k \mid w_{1:k-1})$, we can APPROXIMATE the history by just the last few $N - 1$ words;

Examples:

- **2-gram or bigram model:**

$$(\star) \approx \prod_{k=1}^n P(w_k \mid w_{k-1}) \stackrel{(Sec.2.4)}{=} \prod_{k=1}^n \frac{C(w_{k-1}w_k)}{\sum_{w \in W} C(w_{k-1}w)}$$

- **3-gram or trigram model:**

$$(\star) \approx \prod_{k=1}^n P(w_k \mid w_{k-2}, w_{k-1}) \stackrel{(Sec.2.4)}{=} \prod_{k=1}^n \frac{C(w_{k-2}w_{k-1}w_k)}{\sum_{w \in W} C(w_{k-2}w_{k-1}w)}$$

- In a bigram model, for example, counts $C(w_{k-1}w_k)$ ($k=1, \dots, n$) can easily be observed in the **Word-Word Co-occurrence Matrix** (see Section 3.4)

Example: Berkeley Restaurant Project, i.e., a dialogue system from the last century that answered questions about a database of restaurants in Berkeley, California (Jurafsky et al., 1994)

	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

Table 1 Bigram counts for eight of the words (out of $|V|=1446$) in the Berkeley Restaurant Project corpus of 9332 sentences.

- The conditional probabilities $P(w_k | w_{k-1})$ are given by the **Word-Word Probability Co-occurrence Matrix**:

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

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

Table 2 Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences.

- In the bigram model, the probability $P(< S > \text{ i want chinese food } < E >)$ is approximated as follows:

$$\begin{aligned}
 P(< S > \text{ i want chinese food } < E >) &= P(\text{i} | < S >) \cdot P(\text{want} | \text{i}) \cdot \\
 &\quad P(\text{chinese} | \text{want}) \cdot \\
 &\quad P(\text{food} | \text{chinese}) \cdot \\
 &\quad P(< E > | \text{food}) \\
 &= 0.25 \cdot 0.33 \cdot \dots \cdot 0.68
 \end{aligned}$$

- **Question:** How can we derive a "new" sentence from the N-gram model?

- **Answer: Sampling!**

- Sampling from a distribution means to choose random points according to their likelihood;
- **Sampling for a N-gram model** - representing a distribution over sentences - means to generate some sentences, choosing each sentence according to its likelihood as defined by the model;

Example: Consider a vocabulary V of words with size $|V|$, i.e., $w_1, w_2, \dots, w_{|V|}$, and we have already given the sentence part

” This was a ”
 $w_{100} \quad w_5 \quad w_{67}$

- In the bigram model, we sample from the **Word-Word Probability Co-occurrence Matrix** row corresponding to word w_{67} , i.e., "a":

Vocabulary	w_1	w_2	\dots	$w_{ V }$	
\dots	\dots	\dots	\dots	\dots	
Vocabulary	w_{67}	$P(w_1 \mid w_{67})$	$P(w_2 \mid w_{67})$	\dots	$P(w_{ V } \mid w_{67})$
\dots	\dots	\dots	\dots	\dots	

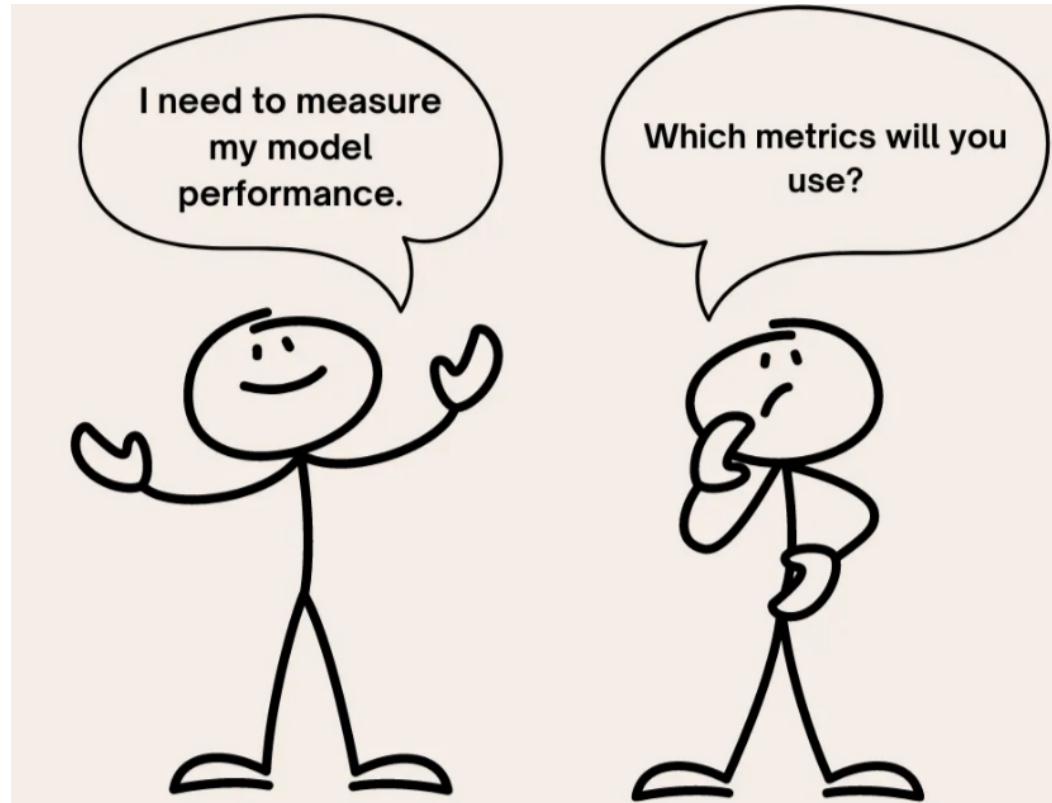
- It is essentially **drawing a random sample** from a **multinomial distribution** with probabilities $\{P(w_1 \mid w_{67}), P(w_2 \mid w_{67}), \dots, P(w_{|V|} \mid w_{67})\}$

- **Remark:** The quality of sampling new sentences from N-gram models depends on two aspects:
 - **Training corpus**, i.e., the conditional probabilities often encode specific facts about a given training corpus;
 - **Value N** , i.e., new sampled sentences are more coherent if we increase the value of N .

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; –It cannot be but so.

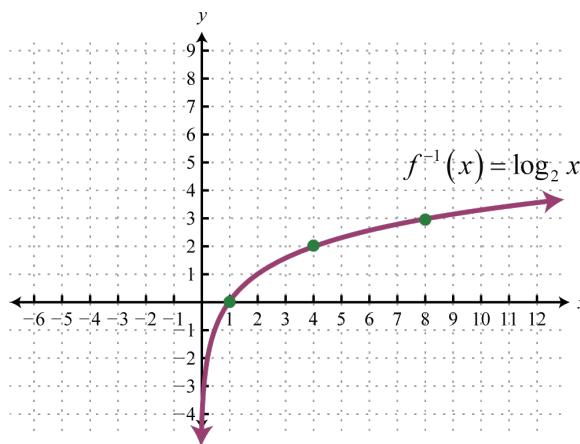
Table 3 Eight sentences randomly generated from four n-grams computed from Shakespeare's works. All characters were mapped to lower-case and punctuation marks were treated as words. Output is hand-corrected for capitalization to improve readability.

- **Question:** But how do we evaluate the quality of the model?
- In other words, does there exist a certain measure that indicates the performance of the model for the training corpus?



- **Answer:** Yes, likelihood principles!

- When the model fits the training corpus perfect, all conditional probabilities equal 1;
- As result, the log-likelihood (denoted by l) should be equal to 0 for obtaining a perfect fit;
- **Reminder:** Log-likelihood = sum of the logarithm of all conditional probabilities;
- For a non-perfect fit of the model, $l < 0$.



- For convenience reasons, i.e., the lower, the better, $-l$ will be used as measure of performance
- **Question:** And what do we do with words listed in V that do not occur in the training corpora?
- As consequence, these words will never be chosen when sampling new sentences, even though they can be valid and coherent (for example, when the training corpus is small).

- **Answer: Smoothing!**

- Smoothing assigns some non-zero probability to any N-gram, even if it is not seen in the training corpus;
- Smoothing can be viewed as discounting (lowering) some non-zero counts in order to get the probability mass that will be assigned to the zero counts.
- Simple and easy smoothing techniques to consider are the **(1) Laplace (add-one) smoothing** and **(2) add-k smoothing**.
- In what follows, we will discuss these techniques!

- **Note:** In the literature, other (more advanced) smoothing techniques exist, e.g., **stupid backoff, Kneser-Ney smoothing**
- Out of scope of this course!



(1) Laplace (add-one) smoothing

- **Simple idea:** Add one to all the N-gram counts, before normalizing them into probabilities;

Example: Bigram model

$$P_{Laplace}(w_k \mid w_{k-1}) = \frac{C(w_{k-1}w_k) + 1}{\sum_{w \in W} [C(w_{k-1}w) + 1]} = \frac{C(w_{k-1}w_k) + 1}{\sum_{w \in W} C(w_{k-1}w) + |V|}$$

- All the counts that used to be 0 will now have count of 1, the counts of 1 will be 2, etc.

- Our example: Berkeley Restaurant Project (BeRP)

1. Counts $C(w_{k-1}w_k) + 1$:

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

Table 4 Add-one smoothed bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences.

2. Probabilities $P_{Laplace}(w_k \mid w_{k-1})$:

	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	0.00078	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

Table 5 Add-one smoothed bigram probabilities for eight of the words (out of $V = 1446$) in the BeRP corpus of 9332 sentences.

- **Note:** By adding 1 to all counts, we see that some original non-zero probabilities, e.g., $P(\text{to} \mid \text{want}) = 0.66$, decreased dramatically, i.e., $P_{Laplace}(\text{to} \mid \text{want}) = 0.26$;

→ This sharp change occurs because too much probability mass is moved to all the zeros!

(2) Add-k smoothing

- **Idea:** Move a bit less of the probability mass from the observed to the unobserved N-grams;
- To do this, instead of adding 1 to each count, a fraction count k is added;

Example: Bigram model

$$P_{Add-k}(w_k \mid w_{k-1}) = \frac{C(w_{k-1}w_k) + k}{\sum_{w \in W} [C(w_{k-1}w) + k]} = \frac{C(w_{k-1}w_k) + k}{\sum_{w \in W} C(w_{k-1}w) + k \mid V \mid}$$

(2) LM based on NN architectures

- Neural LMs (NLMs) have many advantages over the N-gram LMs, i.e.,
 - Handle much longer histories;
 - Generalize better over contexts of similar words;
 - More accurate at word prediction;
- Nevertheless, NLMs are more complex, slower and need more energy to train, and are less interpretable than N-gram models;

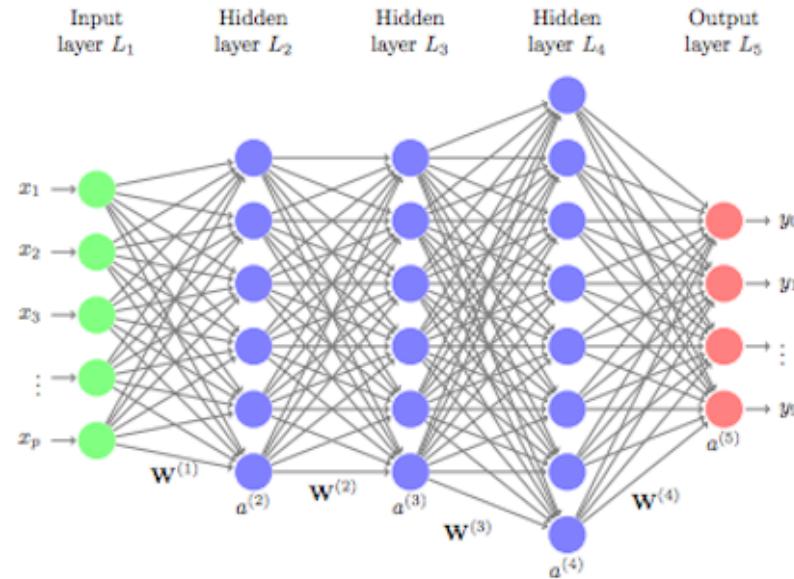
- Nowadays, neural network frameworks are considered as golden standard for LM, especially when having with a huge amount of training corpus;



- In what follow, we will discuss three important NLM frameworks, i.e., **(1) Feedforward NLMs**, **(2) Recurrent NLMs**, and **(3) Transformers**;

(1) Feedforward NLMs

- **Feedforward NLM** = Feedforward NN that takes as (1) input at time t a representation of some number of previous words (w_{t-1}, w_{t-2} , etc.) and (2) outputs a probability distribution over possible next words in vocabulary V .
- **Feedforward NN** = Multilayer neural network in which the units are connected with no cycles, i.e., the outputs from units in each layer are passed to units in the next higher layer; no outputs are passed back to lower layers



- In literature, feedforward NN are also referred as **multilayer perceptrons**
- Thus, like N-gram models, feedforward NLMs approximate the probability of a word w_t given the entire prior context, i.e., $P(w_t | w_{1:t-1})$, by approximating based on the $t - 1$ previous words:
$$P(w_t | w_{1:t-1}) \approx P(w_t | w_{t-n+1}, \dots, w_{t-1})$$
- CBOW (Section 3.5) is an example of a feedforward NLM.
- In what follows, we will explain the general steps taken in a feedforward NLM, with $n = 4$ and only 1 hidden layer.

- **Architecture of a feedforward NLM with $n = 4$ & 1 hidden layer:**

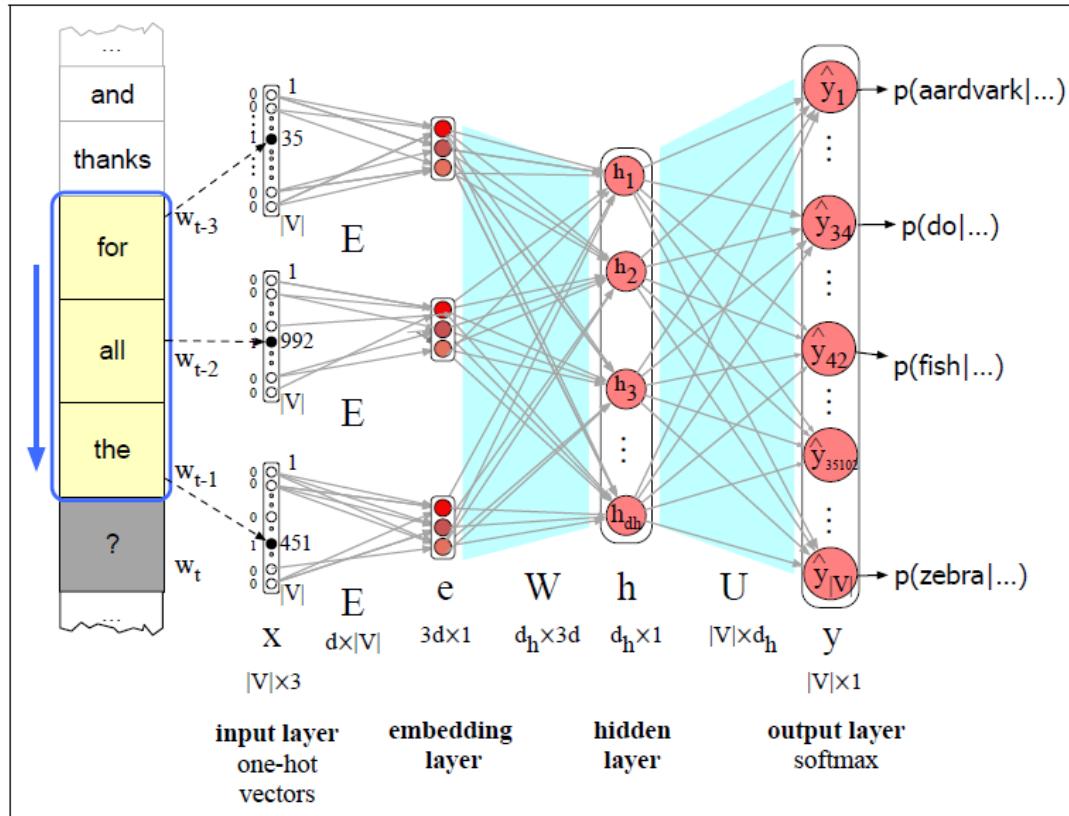


Figure 1 Forward inference in a feedforward neural language model. At each timestep t the network computes a d -dimensional embedding for each context word (by multiplying a one-hot vector by the embedding matrix E), and concatenates the 3 resulting embeddings to get the embedding layer e . The embedding vector e is multiplied by a weight matrix W and then an activation function is applied element-wise to produce the hidden layer h , which is then multiplied by another weight matrix U . Finally, a softmax output layer predicts at each node i the probability that the next word w_t will be vocabulary word V_i .

- In summary, the equations for a feedforward NLM with $n = 4$ & 1 hidden layer are:

$$\mathbf{e} = [\mathbf{Ex}_{t-3}; \mathbf{Ex}_{t-2}; \mathbf{Ex}_{t-1}]$$

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

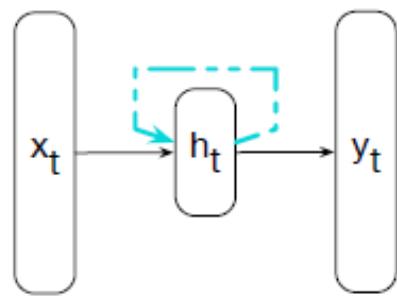
$$\mathbf{z} = \mathbf{Uh}$$

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

- To optimize the hyperparameters within this NN, i.e., \mathbf{E} , \mathbf{W} , \mathbf{U} , and \mathbf{b} , a specific loss function will be minimized through gradient descent (Section 3.5).

(2) Recurrent NLMs

- Before diving into recurrent NLMs, we need to have some notion about recurrent NN.
- **Recurrent NN (RNN)** = Any network that contains a cycle within its network connections, i.e., the value of some unit is directly, or indirectly, dependent on its own earlier outputs as an input
- For simplicity and starting point in this discussion, we first consider a class of RNNs referred as **Elman Networks** (Elman, 1990) or **simple recurrent networks**



- From the figure, a **recurrent link** is now added to a feedforward NN, presented by the blue dashed line.
- This link augments the input to the computation at the hidden layer with the value of the hidden layer from the preceding point in time

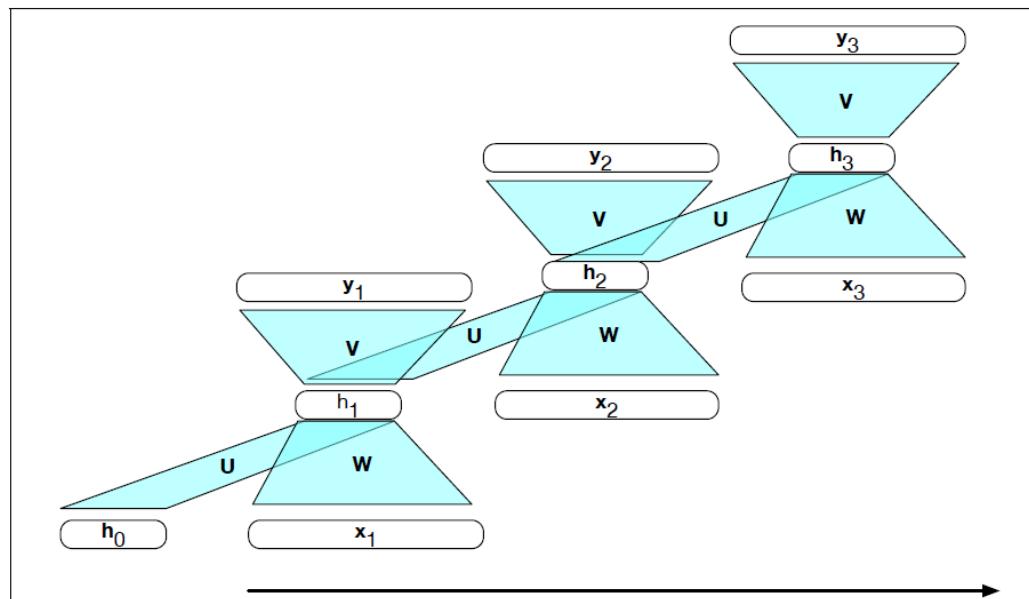


Figure 2 A simple recurrent neural network shown unrolled in time. Network layers are recalculated for each time step, while the weights U , V and W are shared across all time steps.

- Let's see how to apply RNNs to the language modeling task, i.e., recurrent NLMs

- **Recurrent NLMs = Recurrent NN** that process the input sequence one word at a time, attempting to predict the next word from the current word and the previous hidden state;

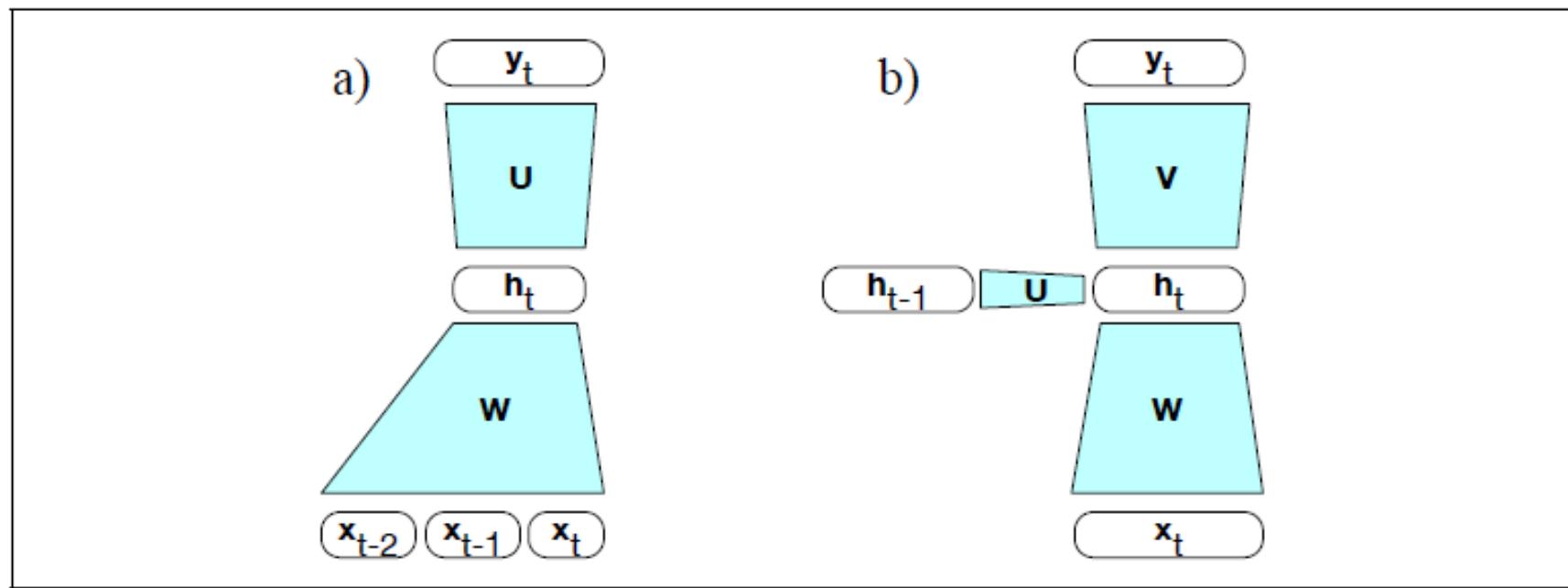


Figure 3 Simplified sketch of (a) a feedforward neural language model versus (b) an RNN language model moving through a text.

- **Note:** $\mathbf{y}_t = P(w_{t+1} \mid w_1, \dots, w_t)$

- The equations for a simple recurrent network are:

$$\begin{aligned}\mathbf{e}_t &= \mathbf{Ex}_t \\ \mathbf{h}_t &= \sigma(\mathbf{Uh}_{t-1} + \mathbf{We}_t) \\ \mathbf{z}_t &= \mathbf{Vh}_t \\ \hat{\mathbf{y}}_t &= \text{softmax}(\mathbf{z}_t)\end{aligned}$$

- **Comparison with N-gram & feedforward NLMs:**

Since the hidden state can represent information about all of the preceding words all the way back to the beginning of the sequence,

- Recurrent NLMs don't have the limited context problem that N-gram models have;
- Recurrent NLMs don't limit to the fixed context that feedforward NLMs have;

- Similar to N-gram models, generating "new" sentences from NLMs can be achieved by sampling from the softmax distribution, as follows:
 - Sample a word in the output from the softmax distribution that results from using the beginning of sentence marker $< S >$, as the first input;
 - Use the word embedding for that word as the input to the network at the next time step, and then sample the next word in the same fashion;
 - Continue generating until the end of the sentence marker $< E >$ is sampled or a fixed length limit is reached.
- This is also referred as **autoregressive generation!**

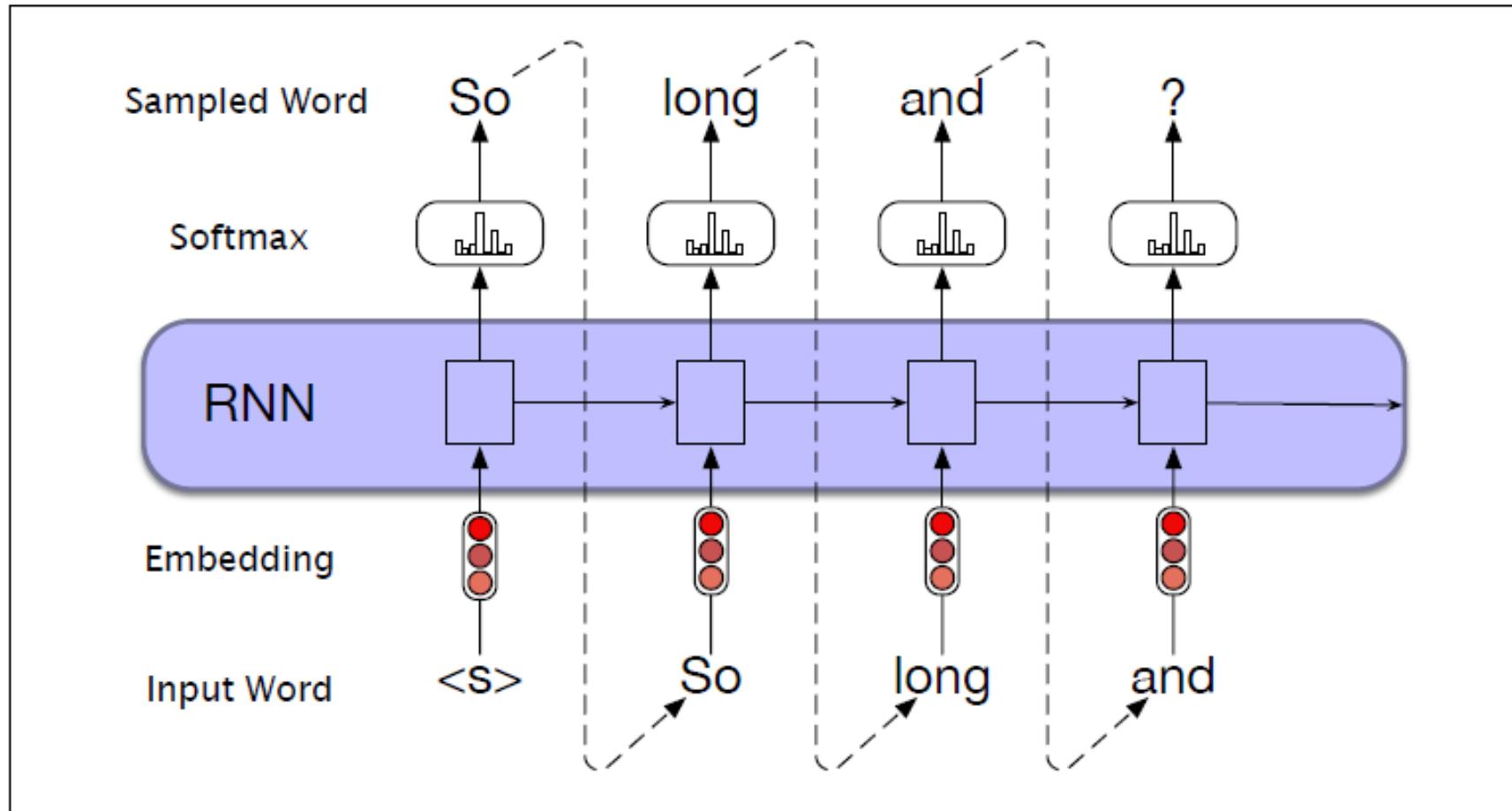


Figure 4 Autoregressive generation with an RNN-based neural language model.

- Let's now look at a popular application in NLP, i.e., **machine translation**, where a recurrent NLM architecture is used as core technology
- **Machine translation** = Prediction of a sequence (in a target language) from another sequence (in a source language)

(SOURCE LANGUAGE)

It is great being here.



Es genial estar aquí.

(TARGET LANGUAGE)

- **Question:** What kind of (recurrent) NLM architecture is standardly used to address this task?

- **Answer: Encoder-decoder NLMs** (also referred as **sequence-to-sequence NLMs**)

- **Key idea of the architecture:** The use of an encoder network that takes an input sequence and created a contextualized representation of it, called the context. This representation is then passed to a decoder which generates a task-specific output sequence

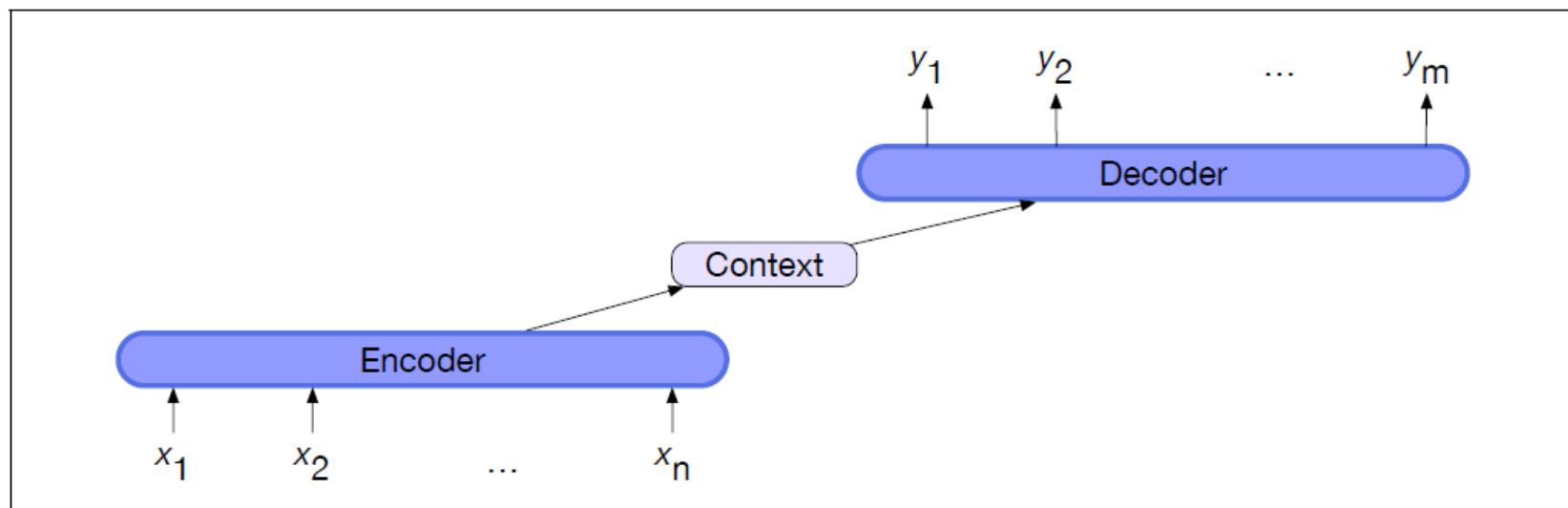
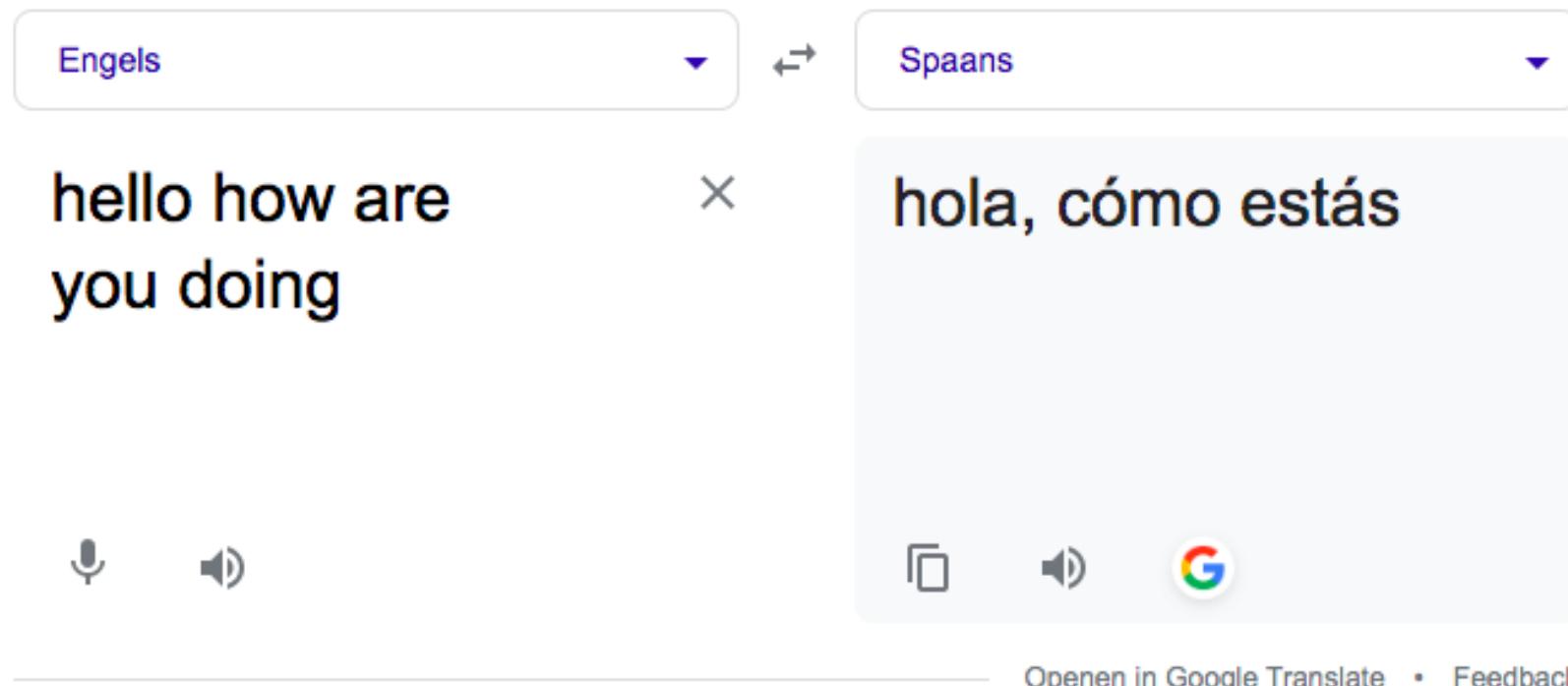


Figure 5 The encoder-decoder architecture. The context is a function of the hidden representations of the input, and may be used by the decoder in a variety of ways.

- This architecture can be seen as the core technology inside Google's translate service



- To begin, we first consider the setup for a simplified encoder-decoder network based on a pair of RNNs
- Let $\mathbf{x} = (x_1, \dots, x_n)$ refer to the source text (e.g., "the green witch arrived", English) plus the separator token $< S >$, and assume $\mathbf{y} = (y_1, \dots, y_v)$ to be the target text (e.g., "llegó la bruja verde", Spanish);
- The basic RNN version of encoder-decoder approach to machine translation can then presented as follows:

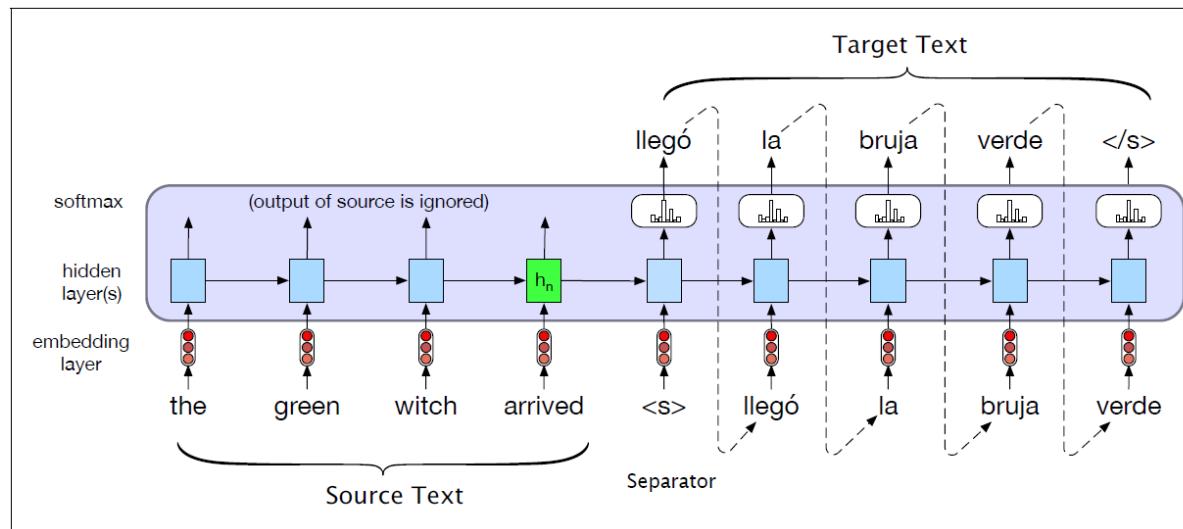


Figure 6 Translating a single sentence (inference time) in the basic RNN version of encoder-decoder approach to machine translation. Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state.

- **Step 1:** The source text is run through the network to generate hidden states, until we get to the end of the source, i.e., "arrived"
- **Step 2:** Starting from input $< S >$ and previous hidden state \mathbf{h}_n , autoregressive generation is used to sample words from the target language
- The equations of the decoder part are defined as:

$$\begin{aligned}\mathbf{h}_n^e &= \mathbf{h}_0^d \\ \mathbf{h}_t^d &= g(\mathbf{h}_{t-1}^d; \hat{y}_{t-1}) \\ \mathbf{z}_t &= f(\mathbf{h}_t^d) \\ \hat{y}_t &= \text{softmax}(\mathbf{z}_t)\end{aligned}$$

- **Remark:** Superscript e and d denote the encoder and decoder part, respectively;

- **Purpose of the encoder:** Generate a contextualized representation of the input, embodied in the final hidden state \mathbf{h}_n^e
 - This representation, also called **c** for **context**, is then passed to the decoder.
- **Limitation 1:** The influence of context vector **c** will wane as the output sequence is generated;
- **Possible solution:** Make the context vector **c** available at each step in the decoding process;
 - This result in a more generalized presentation of the framework;

- **Generalization:**

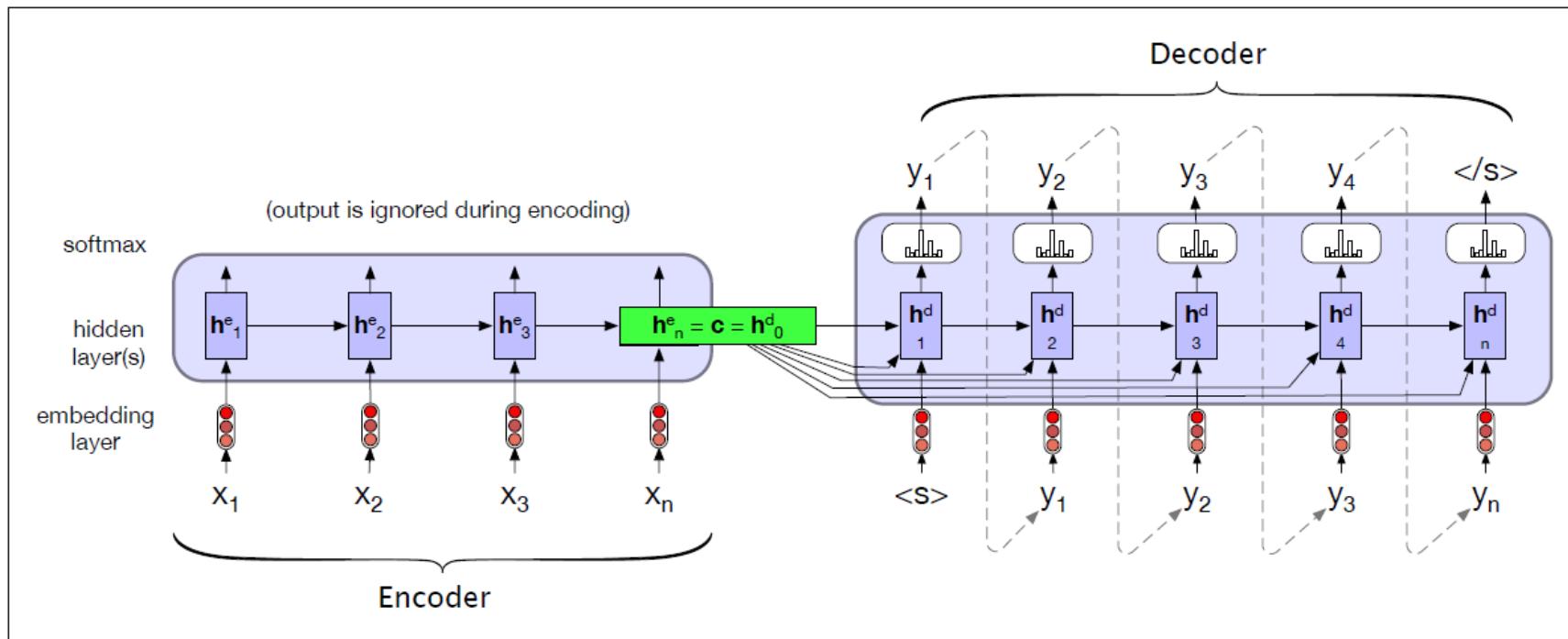


Figure 7 A more formal version of translating a sentence at inference time in the basic RNN-based encoder-decoder architecture. The final hidden state of the encoder RNN, h_e^n , serves as the context for the decoder in its role as h_d^0 in the decoder RNN.

- The equations of the decoder part are adapted accordingly as follows:

$$\begin{aligned}
 \mathbf{c} &= \mathbf{h}_n^e \\
 \mathbf{h}_0^d &= \mathbf{c} \\
 \mathbf{h}_t^d &\stackrel{(\star)}{=} g(\mathbf{h}_{t-1}^d; \hat{y}_{t-1}; \mathbf{c}) \\
 \mathbf{z}_t &= f(\mathbf{h}_t^d) \\
 \hat{y}_t &= \text{softmax}(\mathbf{z}_t)
 \end{aligned}$$

- To compute the most likely output in the decoder at each time t , the argmax over the softmax output is taken, i.e.,

$$\hat{y}_t = \text{argmax}_{w \in V_{target}} P(w \mid \mathbf{x}, y_1, \dots, y_{t-1})$$

- **Limitation 2:** The only thing the decoder knows about the source text \mathbf{x} is embedded in the context vector \mathbf{c}
- As result, information at the beginning of the source text \mathbf{x} , especially for long sentences, may not be equally well represented in \mathbf{c}
- **Possible solution: Attention mechanism;**
 - **Principle:** Allow the decoder to get information from all the hidden states of the encoder, i.e., $\mathbf{h}_1^e, \mathbf{h}_2^e, \dots, \mathbf{h}_n^e$, not just the last hidden state \mathbf{h}_n^e
 - Since the number of hidden states varies with the size of the input n , the entire tensor of encoder hidden state vectors cannot be used directly as the context for the decoder

- Therefore, a function of all hidden states of the encoder can be considered, i.e., $\mathbf{c} = f(\mathbf{h}_1^e, \mathbf{h}_2^e, \dots, \mathbf{h}_n^e)$
- **Idea of attention:**
 - Create a single fixed-length vector \mathbf{c} by taking a weighted sum of $\mathbf{h}_1^e, \mathbf{h}_2^e, \dots, \mathbf{h}_n^e$
 - The weights focus on ("attend to") a particular part of the source text that is relevant for the word the decoder is currently producing
 - Thus, the static context vector \mathbf{c} is actually replaced with dynamically ones, derived from $\mathbf{h}_1^e, \mathbf{h}_2^e, \dots, \mathbf{h}_n^e$ and different for each word in decoding, i.e., \mathbf{c}_t , for every decoding step t

- **Mathematically:**

$$\mathbf{c}_t = \sum_{j=1}^n \alpha_{tj} \mathbf{h}_j^e$$

- As consequence, equation (\star) in the decoder part is adapted as follows:

$$\mathbf{h}_t^d = g(\mathbf{h}_{t-1}^d; \hat{y}_{t-1}; \mathbf{c}_t)$$

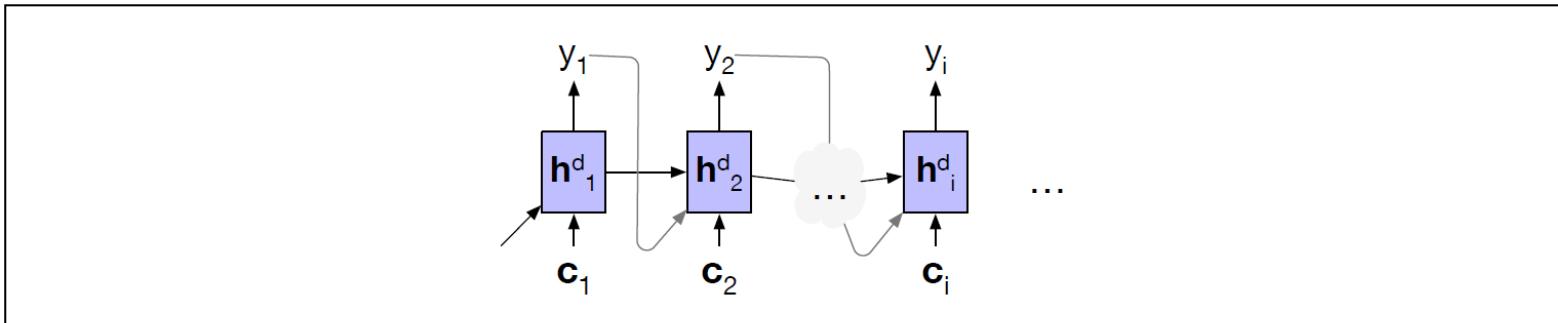


Figure 8 The attention mechanism allows each hidden state of the decoder to see a different, dynamic, context, which is a function of all the encoder hidden states.

- **Question:** But how are these weights α_{tj} calculated?
- **Reminder:** The weights $\alpha_{tj} \in [0, 1]$ focus on ("attend to") a particular part of the source text that is relevant for the word the decoder is currently producing
- In other words, we need to know how relevant each encoder state j is to the decoder state captured in \mathbf{h}_{t-1}^d
- Attention capture relevance by computing - at each state t during decoding - a $score(\mathbf{h}_{t-1}^d, \mathbf{h}_j^e)$, for each encoder state j

- **Simple proposal:** Score relevance by similarity, through a so-called **dot-product attention**:

$$\text{score}(\mathbf{h}_{t-1}^d, \mathbf{h}_j^e) = \mathbf{h}_{t-1}^d \cdot \mathbf{h}_j^e$$

- Since $\alpha_{tj} \in [0, 1]$, the dot-product attention will be normalized through the softmax function:

$$\begin{aligned}\alpha_{tj} &= \text{softmax}(\mathbf{h}_{t-1}^d \cdot \mathbf{h}_j^e) \\ &= \frac{\exp(\mathbf{h}_{t-1}^d \cdot \mathbf{h}_j^e)}{\sum_{k=1}^n \mathbf{h}_{t-1}^d \cdot \mathbf{h}_k^e}\end{aligned}$$

- Thus, α_{tj} tells us the proportional relevance of each encoder hidden state j to the prior hidden decoder state \mathbf{h}_{t-1}^d .

- Graphical representation of an encoder-decoder network with attention mechanism:

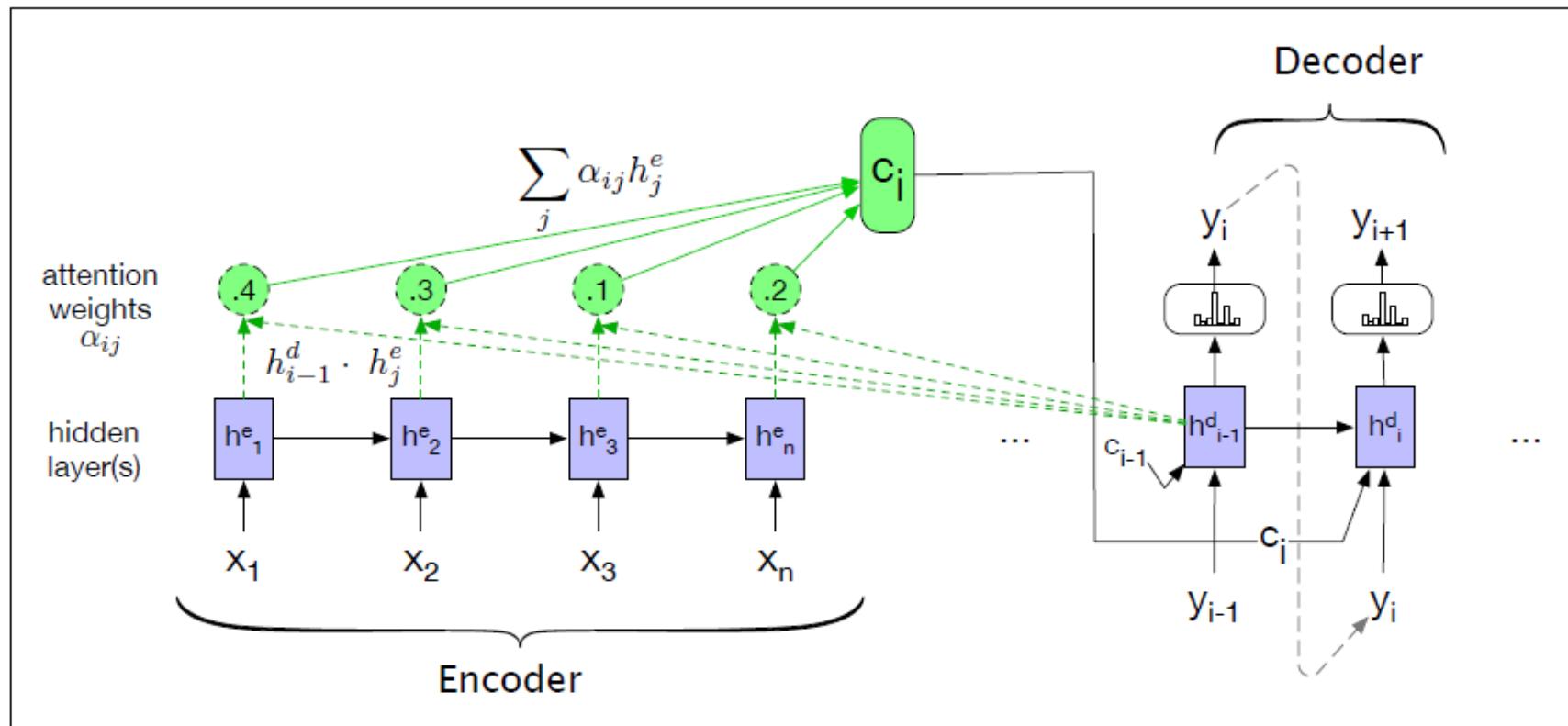
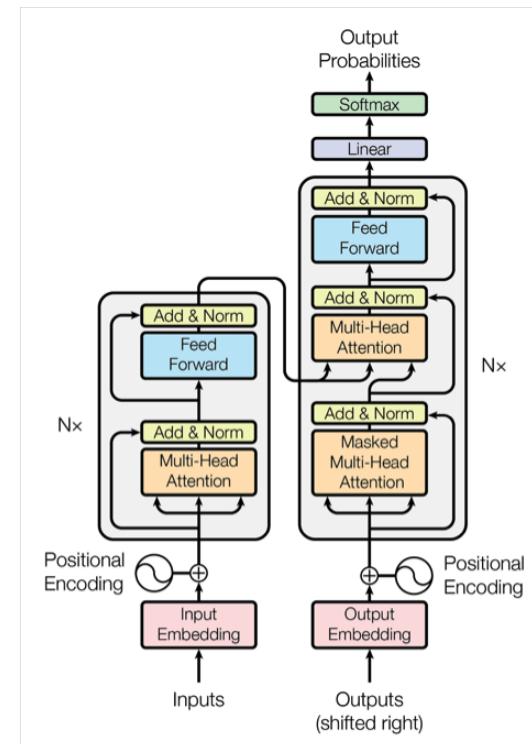


Figure 9 A sketch of the encoder-decoder network with attention, focusing on the computation of c_i . The context value c_i is one of the inputs to the computation of h^d_i . It is computed by taking the weighted sum of all the encoder hidden states, each weighted by their dot product with the prior decoder hidden state h^d_{i-1} .

(3) Transformers

- The usage of attention mechanisms within RNN has led to the development of transformers, i.e., a state-of-the-art NLM developed by Google.
- **Transformers** = Stacks of "**transformer blocks**", each of which is a multilayer NN made by combining

- simple linear layers;
- feedforward networks;
- self-attention layers, the key innovation of transformers.



- The transformer offers new mechanisms (**self-attention** and **positional encodings**) that help represent time and focus on how words relate to each other over *long distances*.
- This **self-attention mechanism** can be seen as a slight modification of the attention mechanism discussed before.
- In what follows, we will explain the underlying mechanisms and construction within Transformers.

Self-attention mechanism:

- **Reminder: Core of an attention-based approach**

- The ability to compare an item of interest (say \mathbf{x}_i) to a collection of other items (say \mathbf{x}_j , $j \leq i$) in a way that reveals their relevance in the current context. In the case of self-attention, the set of comparisons are to other elements within a given sequence. The result of these comparisons is then used to compute an output for the current input (say \mathbf{y}_j).

$$\begin{aligned}\text{score}(\mathbf{x}_i, \mathbf{x}_j) &= \mathbf{x}_i \cdot \mathbf{x}_j \\ \alpha_{ij} &= \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \quad \forall j \leq i \\ &= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^i \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))} \quad \forall j \leq i \\ \mathbf{y}_i &= \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j\end{aligned}$$

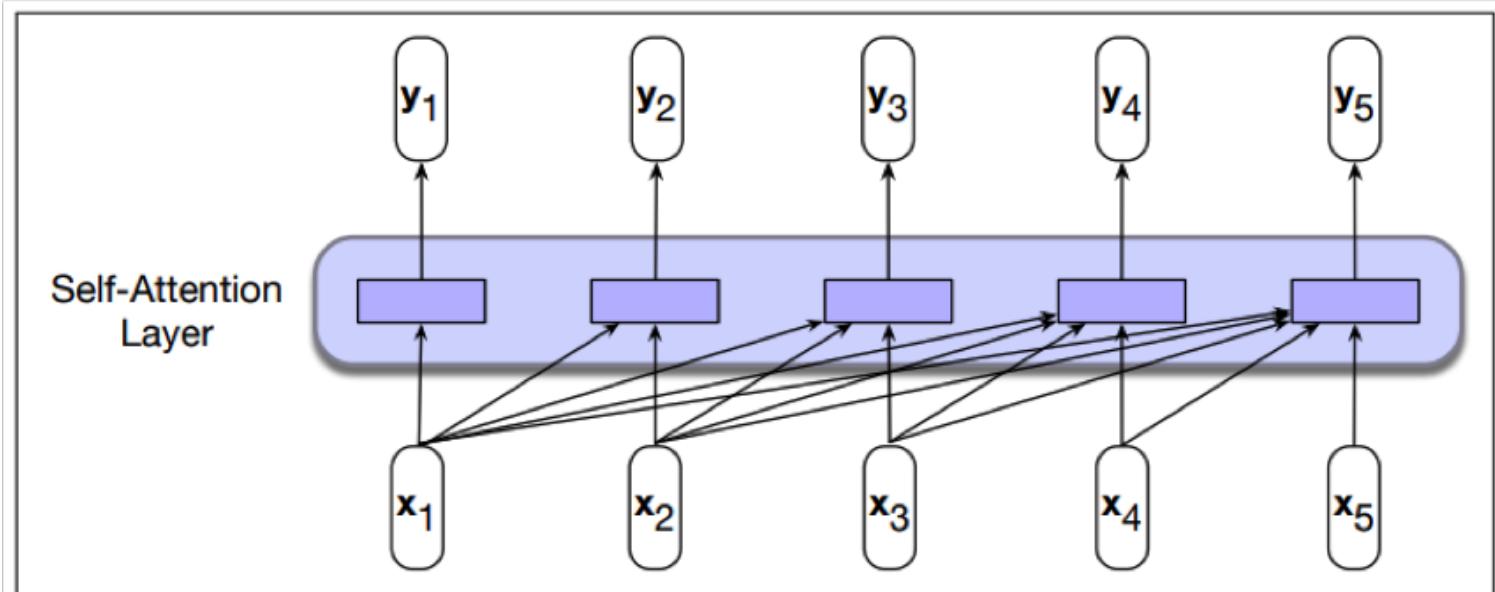


Figure 10.1 Information flow in a causal (or masked) self-attention model. In processing each element of the sequence, the model attends to all the inputs up to, and including, the current one. Unlike RNNs, the computations at each time step are independent of all the other steps and therefore can be performed in parallel.

- Transformers allow us to create a more sophisticated way of representing how words can contribute to the representation of longer inputs.
- **Question:** How will it do this?

- **Answer: Attention-based approach with QUERY, KEY and VALUE**

Consider three different roles that each input embedding plays during the course of the attention process.

- As *the current focus of attention* when being compared to all of the other preceding inputs (**QUERY**);
- In its role as *a preceding input* being compared to the current focus of attention (**KEY**);
- As a **VALUE** used to compute the output for the current focus of attention.

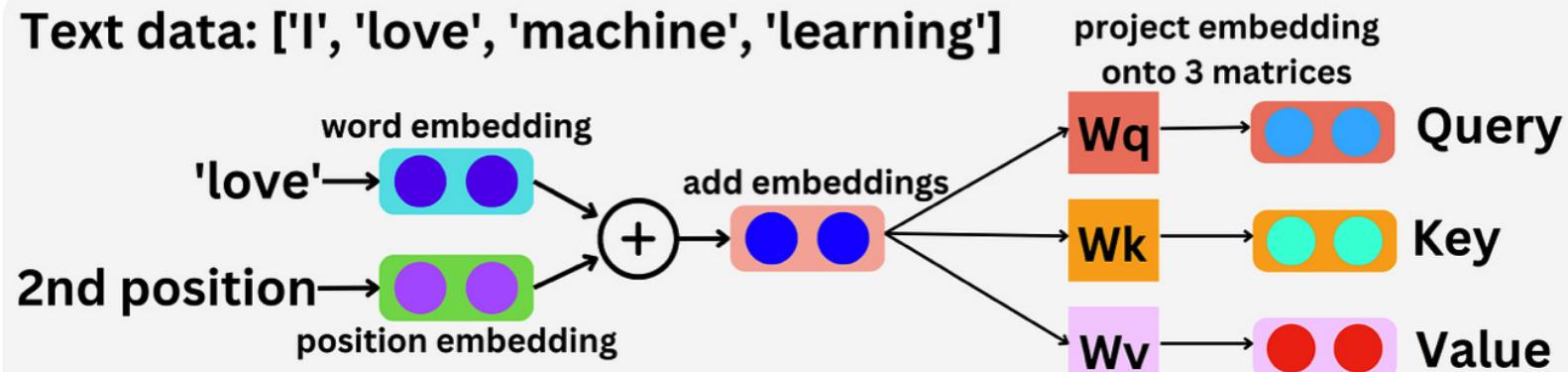
- To capture these three different roles, weight matrices \mathbf{W}^Q , \mathbf{W}^K , and \mathbf{W}^V are introduced, used to project each input embedding vector \mathbf{x}_i into a representation of its role as a QUERY, KEY, and VALUE, respectively.

$$\mathbf{q}_i = \mathbf{W}^Q \mathbf{x}_i, \quad \mathbf{k}_i = \mathbf{W}^K \mathbf{x}_i, \quad \mathbf{v}_i = \mathbf{W}^V \mathbf{x}_i$$

Step 1: Create the Query, Key, Value

TheAiEdge.io

Text data: ['I', 'love', 'machine', 'learning']



- Given these projections, the score between a current focus of attention, \mathbf{x}_i , and an element in the preceding context, \mathbf{x}_j , consists of a dot product between its query vector \mathbf{q}_i and the preceding elements key vectors \mathbf{k}_j .

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{q}_i \cdot \mathbf{k}_j$$

- Remark:** The result of a dot product can be an arbitrarily large (positive or negative) value. Exponentiating such large values can lead to numerical issues and to an effective loss of gradients during training. Therefore, NLP researchers often use the following score function:

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}};$$

d_k = dimensionality of the query and key value.

- The softmax calculation remains the same:

$$\begin{aligned}\alpha_{ij} &= \text{softmax(score}(\mathbf{x}_i, \mathbf{x}_j)) \quad \forall j \leq i \\ &= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^i \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))} \quad \forall j \leq i\end{aligned}$$

- The output calculation for \mathbf{y}_i is now based on a weighted sum over the value vectors \mathbf{v} .

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$$

- Graphical representation of the attention-based approach with KEY, QUERY and VALUE:

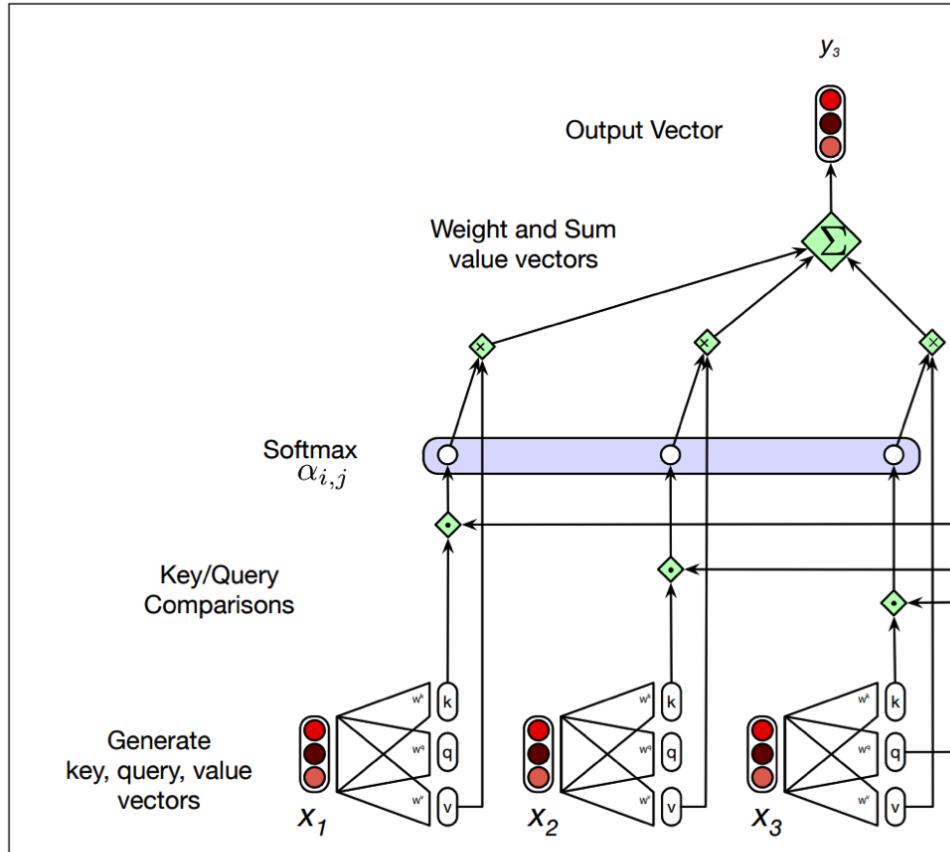


Figure 10.2 Calculating the value of y_3 , the third element of a sequence using causal (left-to-right) self-attention.

- **Generalization for N words in a sentence:**

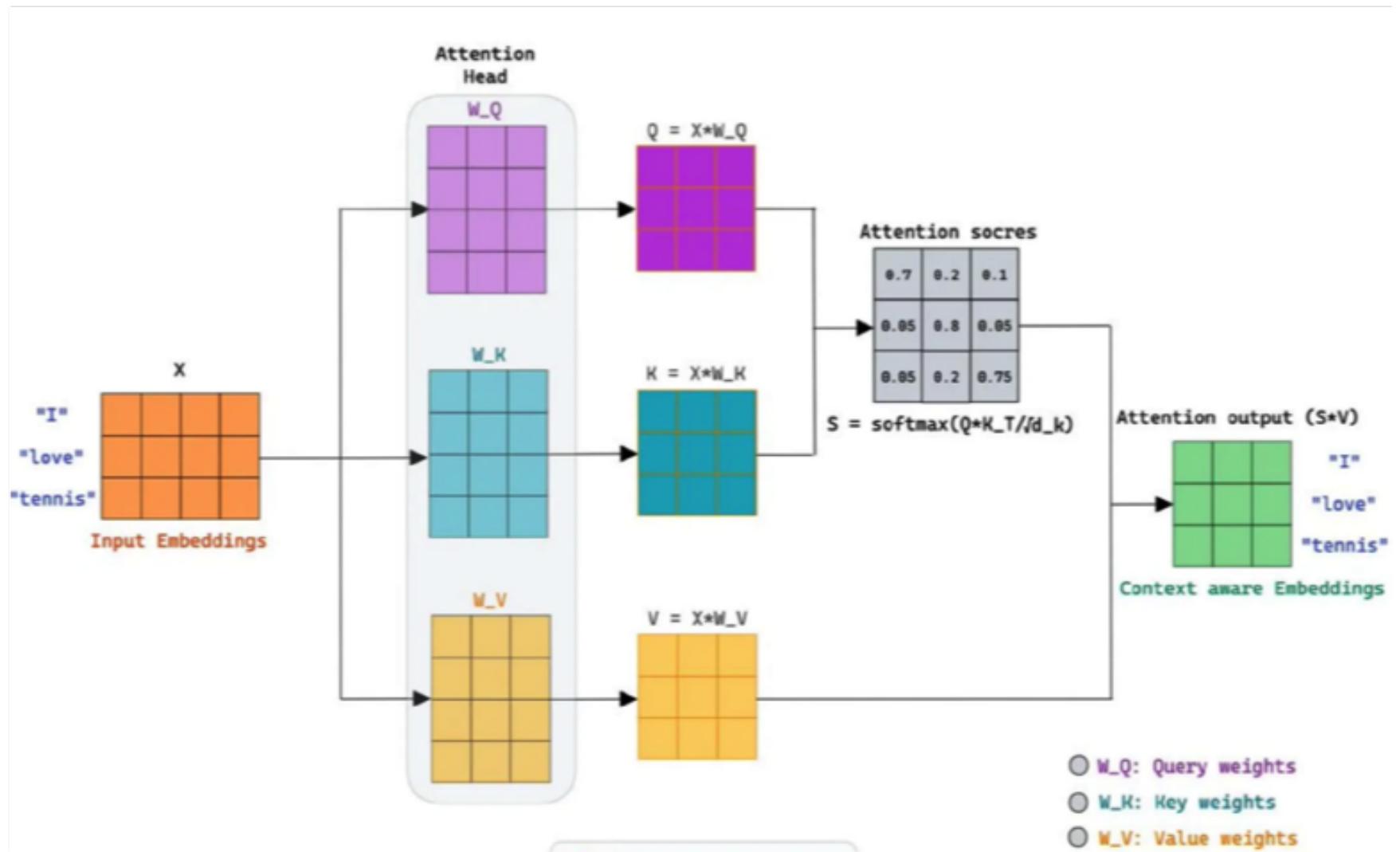
- Since each output y_i is computed independently, this entire process can be parallelized by taking advantage of efficient matrix multiplication routines by packing the input embeddings of the N tokens of the input sequence into a single matrix $\mathbf{X} \in \mathbf{R}^{N \times d}$. That is, each row of \mathbf{X} is the embedding of one token of the input.

$$\mathbf{Q} = \mathbf{XW}^Q, \quad \mathbf{K} = \mathbf{XW}^K, \quad \mathbf{V} = \mathbf{XW}^V,$$

$$\mathbf{Q} \in \mathbf{R}^{N \times d}, \quad \mathbf{K} \in \mathbf{R}^{N \times d}, \quad \mathbf{V} \in \mathbf{R}^{N \times d},$$

- Thus, the entire self-attention step for an entire sequence of N tokens reduces to the following computation:

$$\text{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax} \left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} \right) \mathbf{V}.$$



- **Problem:** \mathbf{QK}^T results in a score for each query value to every key value, *including those that follow the query*. This is inappropriate in the setting of language modeling since guessing the next word is pretty simple if you already know it.
- **Solution:** Elements in the upper-triangular portion of the matrix are zeroed out (set to $-\infty$), thus eliminating any knowledge of words that follow in the sequence.

	$q_1 \cdot k_1$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
	$q_2 \cdot k_1$	$q_2 \cdot k_2$	$-\infty$	$-\infty$	$-\infty$
N	$q_3 \cdot k_1$	$q_3 \cdot k_2$	$q_3 \cdot k_3$	$-\infty$	$-\infty$
	$q_4 \cdot k_1$	$q_4 \cdot k_2$	$q_4 \cdot k_3$	$q_4 \cdot k_4$	$-\infty$
	$q_5 \cdot k_1$	$q_5 \cdot k_2$	$q_5 \cdot k_3$	$q_5 \cdot k_4$	$q_5 \cdot k_5$

Figure 10.3 The $N \times N$ \mathbf{QK}^T matrix showing the $q_i \cdot k_j$ values, with the upper-triangle portion of the comparisons matrix zeroed out (set to $-\infty$, which the softmax will turn to zero).

Transformer blocks:

- The self-attention calculation lies at the core of what's called a "**transformer block**", which, in addition to the *self-attention layer*, includes additional feedforward layers, **residual connections**, and **normalizing layers**.

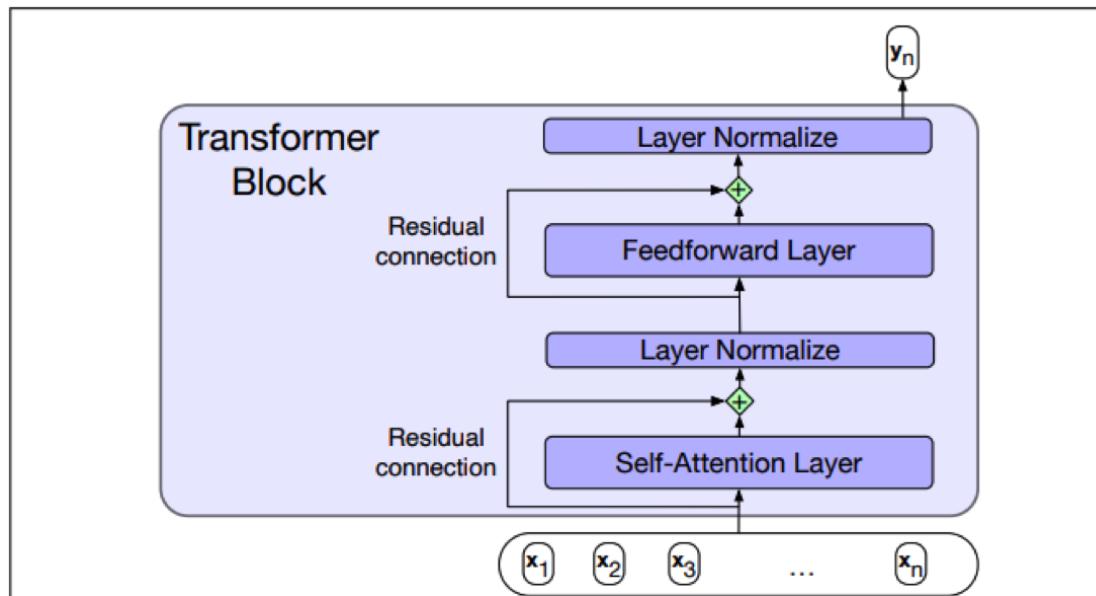


Figure 10.4 A transformer block showing all the layers.

- **Question:** What are (1) **residual connections** and (2) **normalizing layers**?

- **Answer: (1) Residual connections**

- **Residual connections** = Connections that pass information from a lower layer to a higher layer without going through the intermediate layer.
- Allowing information from the activation going forward and the gradient going backwards to skip a layer **improves learning** and gives higher level layers **direct access to information** from lower layers (He et al., 2016).
- **Residual connections** in transformers are implemented by adding a layers input vector to its output vector before passing it forward.

- In Transformers, these **connections** are used with both the attention and feedforward sublayers:

$$z = \text{LayerNorm}(x + \text{SelfAttention}(x))$$

$$y = \text{LayerNorm}(z + \text{FFN}(z))$$

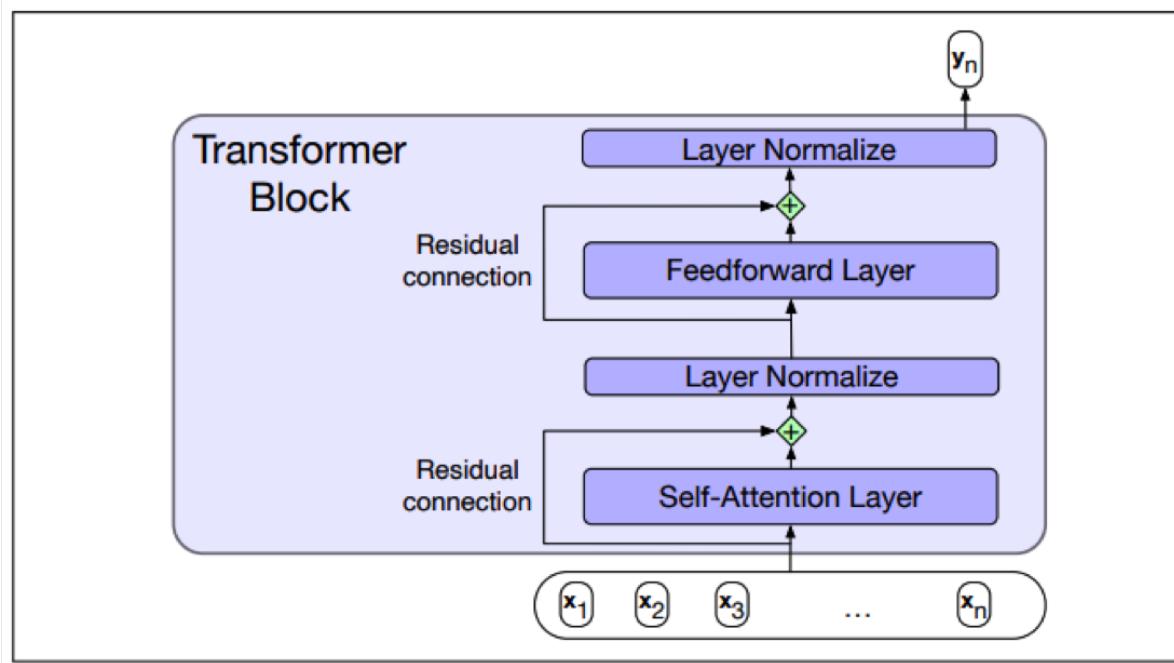


Figure 10.4 A transformer block showing all the layers.

(2) Normalizing layers

- Layer normalization (or layer norm) is one of many forms of normalization that can be used to improve training performance in deep neural networks by keeping the values of a hidden layer in a range that facilitates gradient-based training.
- Layer norm is a variation of the standard score, or z-score, from statistics applied to a single hidden layer.

- **Procedure:**

1. Given a hidden layer with dimensionality d_h , we first calculate its mean and standard deviation:

$$\mu = \frac{1}{d_h} \sum_{i=1}^{d_h} x_i, \quad \sigma = \sqrt{\frac{1}{d_h} \sum_{i=1}^{d_h} (x_i - \mu)^2}$$

2. The vector components are normalized by subtracting this mean from each and dividing by the standard deviation:

$$\hat{\mathbf{x}} = \frac{(\mathbf{x} - \mu)}{\sigma}$$

3. In the standard implementation of layer normalization, two learnable parameters γ and β are introduced:

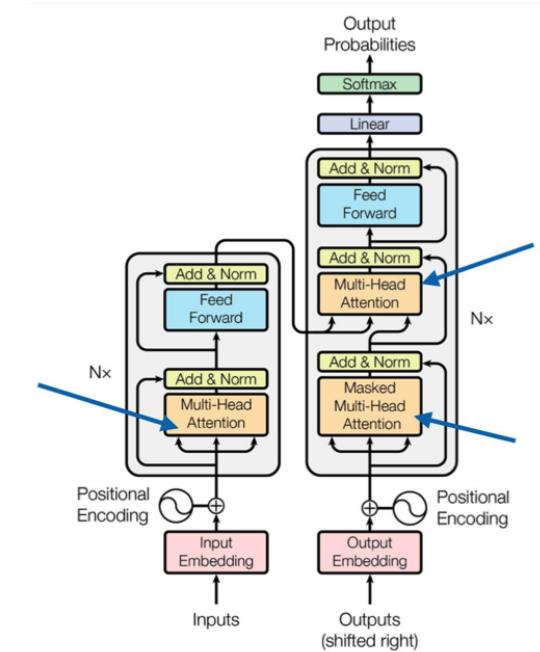
$$\text{LayerNorm}(\mathbf{x}) = \gamma \hat{\mathbf{x}} + \beta$$

Multi-head attention:

- Different words in a sentence can relate to each other in many different ways simultaneously. For example, distinct syntactic, semantic, and discourse relationships can hold between verbs and their arguments in a sentence.
- **Problem:** It would be difficult for a single transformer block to learn to capture all of the different kinds of parallel relations among its inputs

- **Solution:** Multi-head attention layers

- Sets of self-attention layers, called **heads**, that reside in parallel layers at the same depth in a model, each with its own set of parameters.
- Given these distinct sets of parameters, each head can learn different aspects of the relationships that exist among inputs at the same level of abstraction.



- **Procedure:**

- Each head i in a self-attention layer is provided with its own set of KEY, QUERY and VALUE matrices:

$$\mathbf{W}_i^K \in \mathbf{R}^{d \times d_k}, \quad \mathbf{W}_i^Q \in \mathbf{R}^{d \times d_k}, \quad \mathbf{W}_i^V \in \mathbf{R}^{d \times d_v}$$

- These get multiplied by the inputs packed into \mathbf{X} to produce

$$\mathbf{K}_i \in \mathbf{R}^{N \times d_k}, \quad \mathbf{Q}_i \in \mathbf{R}^{N \times d_k}, \quad \mathbf{V}_i \in \mathbf{R}^{N \times d_v}$$

- The output of each of the h heads, i.e.,

$$\text{SelfAttention}(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i) = \text{softmax}\left(\frac{\mathbf{Q}_i \mathbf{K}_i^T}{\sqrt{d_k}}\right) \mathbf{V}_i, \forall i \in \{1, \dots, h\},$$

is of shape $N \times d_v$, and so the output of the multi-head layer with h heads consists of h vectors of shape $N \times d_v$.

- To make use of these vectors in further processing, they are combined and then reduced down to the original input dimension d . This is accomplished by concatenating the outputs from each head and then using yet another linear projection,

$$\mathbf{W}^O \in \mathbf{R}^{hd_v \times d},$$

to reduce it to the original output dimension for each token, or a total $N \times d$ output.

- **Mathematically:**

$$\begin{aligned}\text{MultiHeadAttention}(\mathbf{X}) &= (\text{head}_1 \oplus \dots \oplus \text{head}_h) \mathbf{W}^O; \\ \mathbf{Q}_i &= \mathbf{X} \mathbf{W}_i^Q; \mathbf{K}_i = \mathbf{X} \mathbf{W}_i^K; \mathbf{V}_i = \mathbf{X} \mathbf{W}_i^V; \\ \text{head}_i &= \text{SelfAttention}(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i)\end{aligned}$$

- **Graphical representation:**

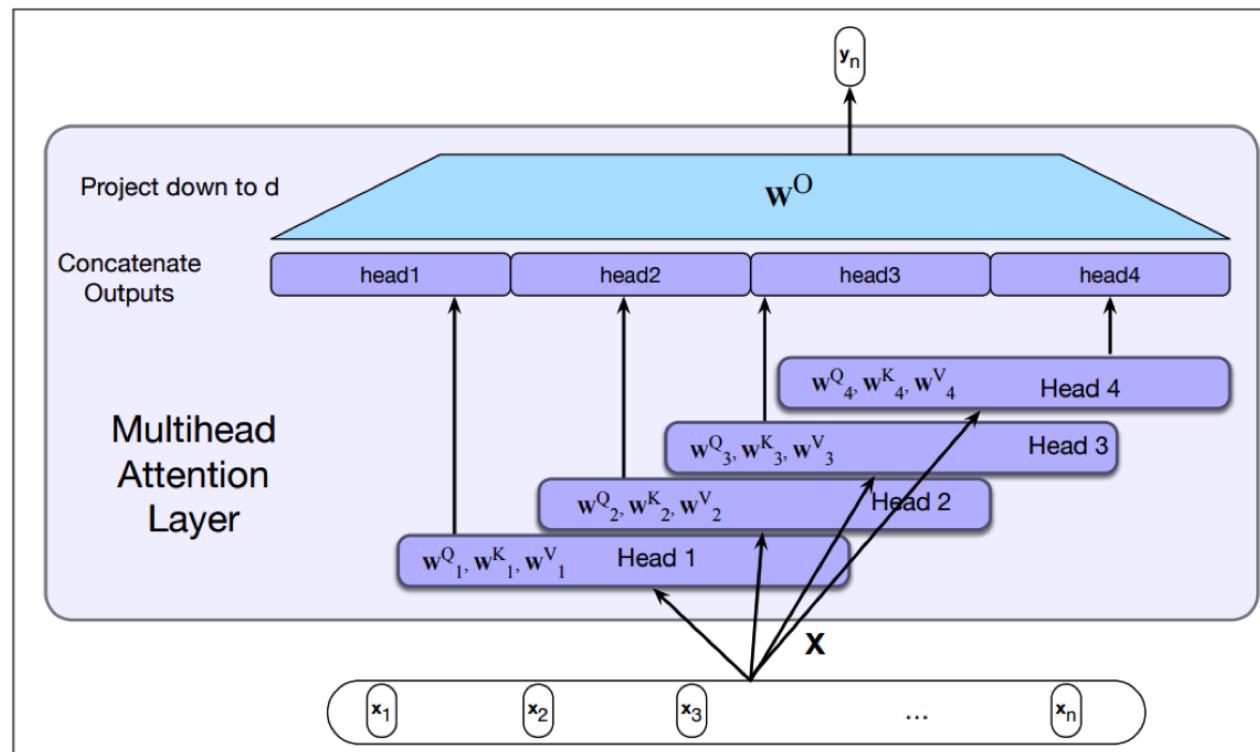


Figure 10.5 Multihead self-attention: Each of the multihead self-attention layers is provided with its own set of key, query and value weight matrices. The outputs from each of the layers are concatenated and then projected down to d , thus producing an output of the same size as the input so layers can be stacked.

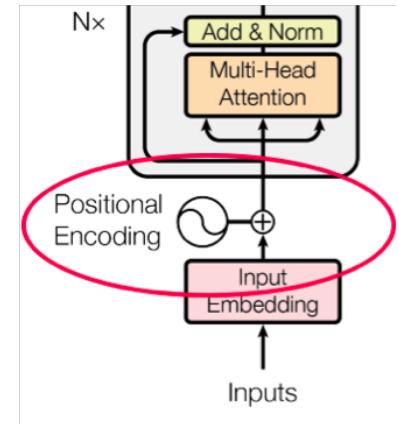
- **Remark:** This multi-head layer replaces the single self-attention layer in the transformer block (Figure 10.4).

Positional encoding:

- With RNNs, information about the order of the inputs was built into the structure of the model.
- Unfortunately, the same isn't true for transformers.
 - The models as we've described them so far don't have any notion of the relative, or absolute, positions of the tokens in the input.
 - If you scramble the order of the inputs in the attention computation in Fig. 10.2 you get exactly the same answer.
- **Question:** But how is this information then included in the framework?

- **Answer: Positional encoding**

- Modify the input embeddings by combining them with positional embeddings specific to each position in an input sequence.

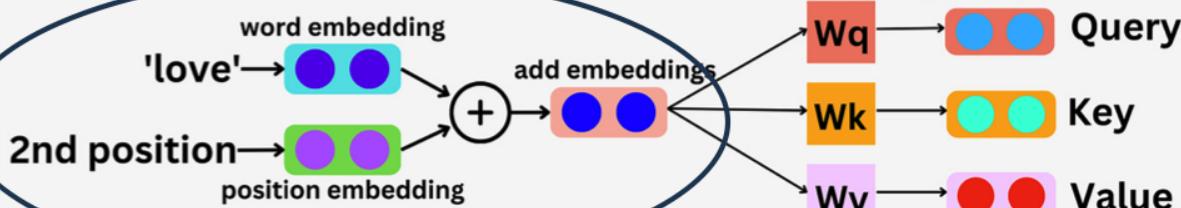


- In the original work of transformers, the word embedding for each input are **simply added** to its corresponding positional embedding.

Step 1: Create the Query, Key, Value

TheAiEdge.io

Text data: ['I', 'love', 'machine', 'learning']



- As result, the positional embedding needs to have the same dimensionality as the word embedding.
- While word embeddings can be achieved with the classical CBOW or skip-gram models, a **static function** is originally proposed that maps integer inputs to real valued vectors in a way that *captures the inherent relationships among the positions*.
- **Question:** How is this achieved?

- **Answer: Sine and cosine functions with differing frequencies**

- Suppose you have an input sequence of length L and require the position of the k^{th} object in the sequence.
- The **positional encoding** is given by sine and cosine functions of varying frequencies:

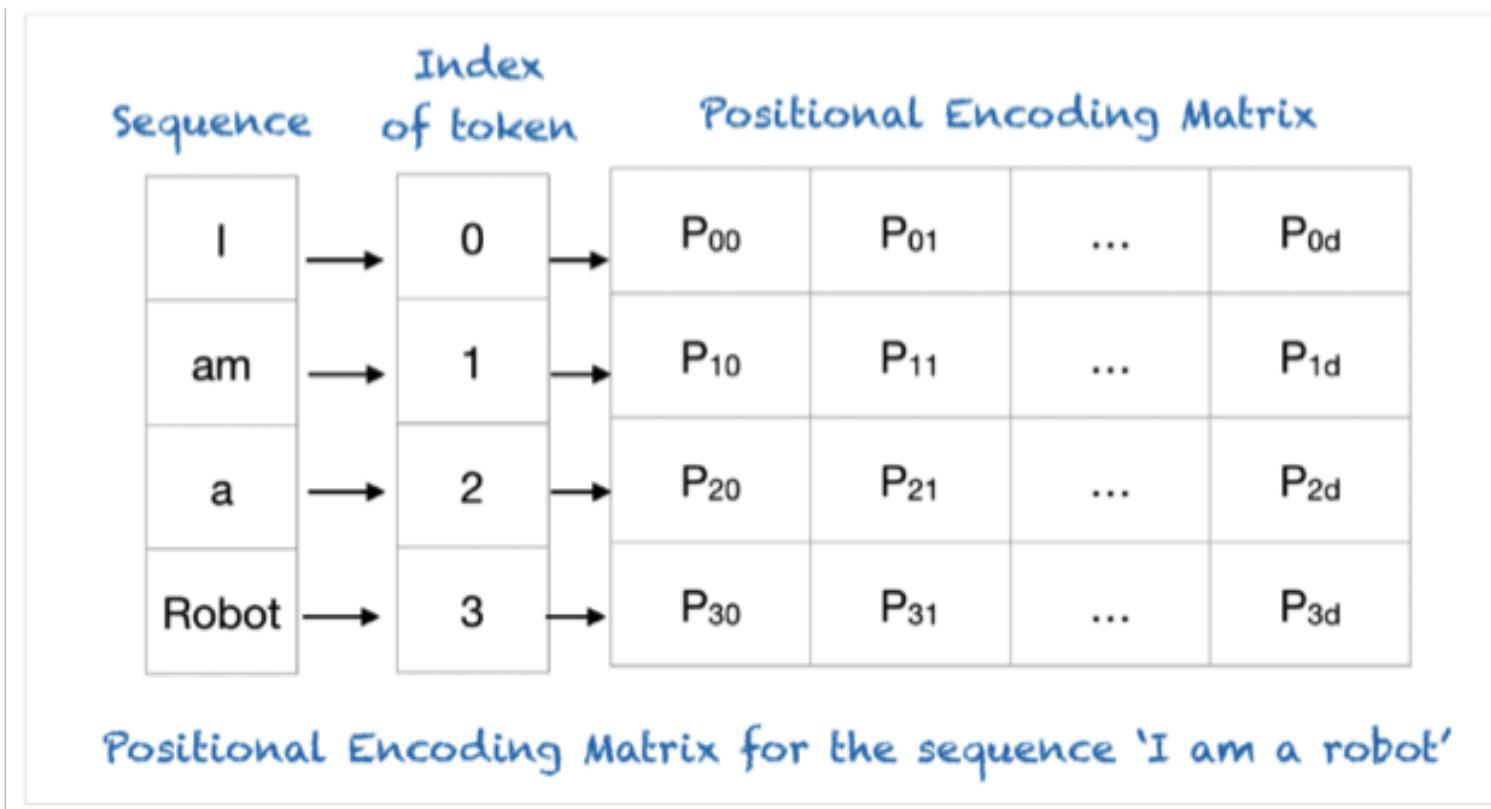
$$P_{k,2i} = \sin\left(\frac{k}{n^{2i/d}}\right);$$
$$P_{k,2i+1} = \cos\left(\frac{k}{n^{2i/d}}\right).$$

Here:

- k : Position of an object in the input sequence;
- d : Dimension of the embedding space (i.e., same as the word embedding);
- $P_{k,j}$: Position function for mapping a position k in the input sequence to index (k, j) of the positional matrix;
- n : User-defined scalar (e.g., set to 10000 by the authors of "Attention is all you need");
- i : Used for mapping to column indices with a single value of i maps to both sine and cosine functions.

- **Example:** Consider the sentence "I am a Robot"

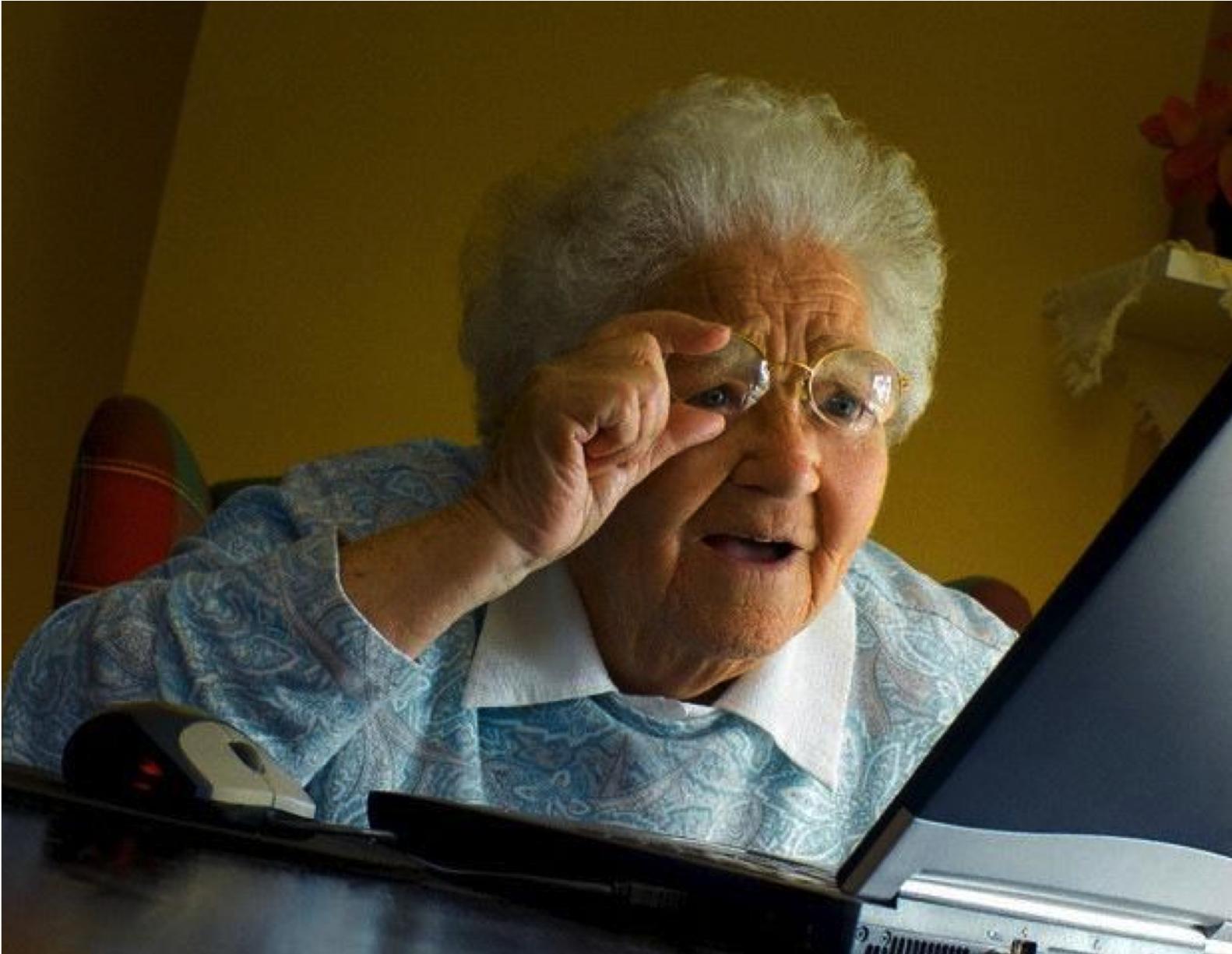
- First index the word, based on the position within the sentence, and choose the dimensionality of the word embedding (say d) to create the initial positional encoding matrix:



- Choosing $n=100$ (user-defined), and assume that $d=4$, the following positional encoding is obtained:

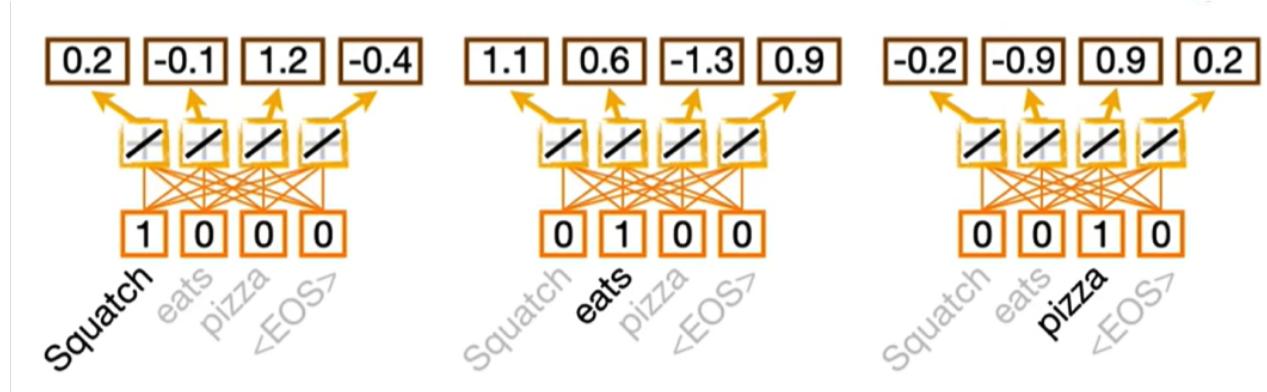
Sequence	Index of token, k	Positional Encoding Matrix with $d=4, n=100$			
		$i=0$	$i=0$	$i=1$	$i=1$
I	0	$P_{00}=\sin(0) = 0$	$P_{01}=\cos(0) = 1$	$P_{02}=\sin(0) = 0$	$P_{03}=\cos(0) = 1$
am	1	$P_{10}=\sin(1/1) = 0.84$	$P_{11}=\cos(1/1) = 0.54$	$P_{12}=\sin(1/10) = 0.10$	$P_{13}=\cos(1/10) = 1.0$
a	2	$P_{20}=\sin(2/1) = 0.91$	$P_{21}=\cos(2/1) = -0.42$	$P_{22}=\sin(2/10) = 0.20$	$P_{23}=\cos(2/10) = 0.98$
Robot	3	$P_{30}=\sin(3/1) = 0.14$	$P_{31}=\cos(3/1) = -0.99$	$P_{32}=\sin(3/10) = 0.30$	$P_{33}=\cos(3/10) = 0.96$

Positional Encoding Matrix for the sequence 'I am a robot'

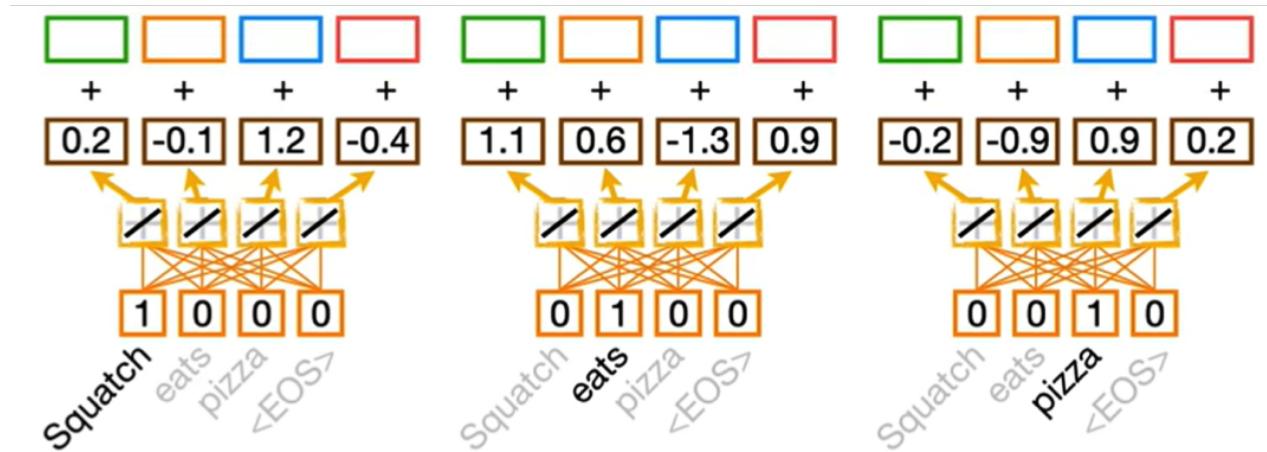


- **Intuitively:**

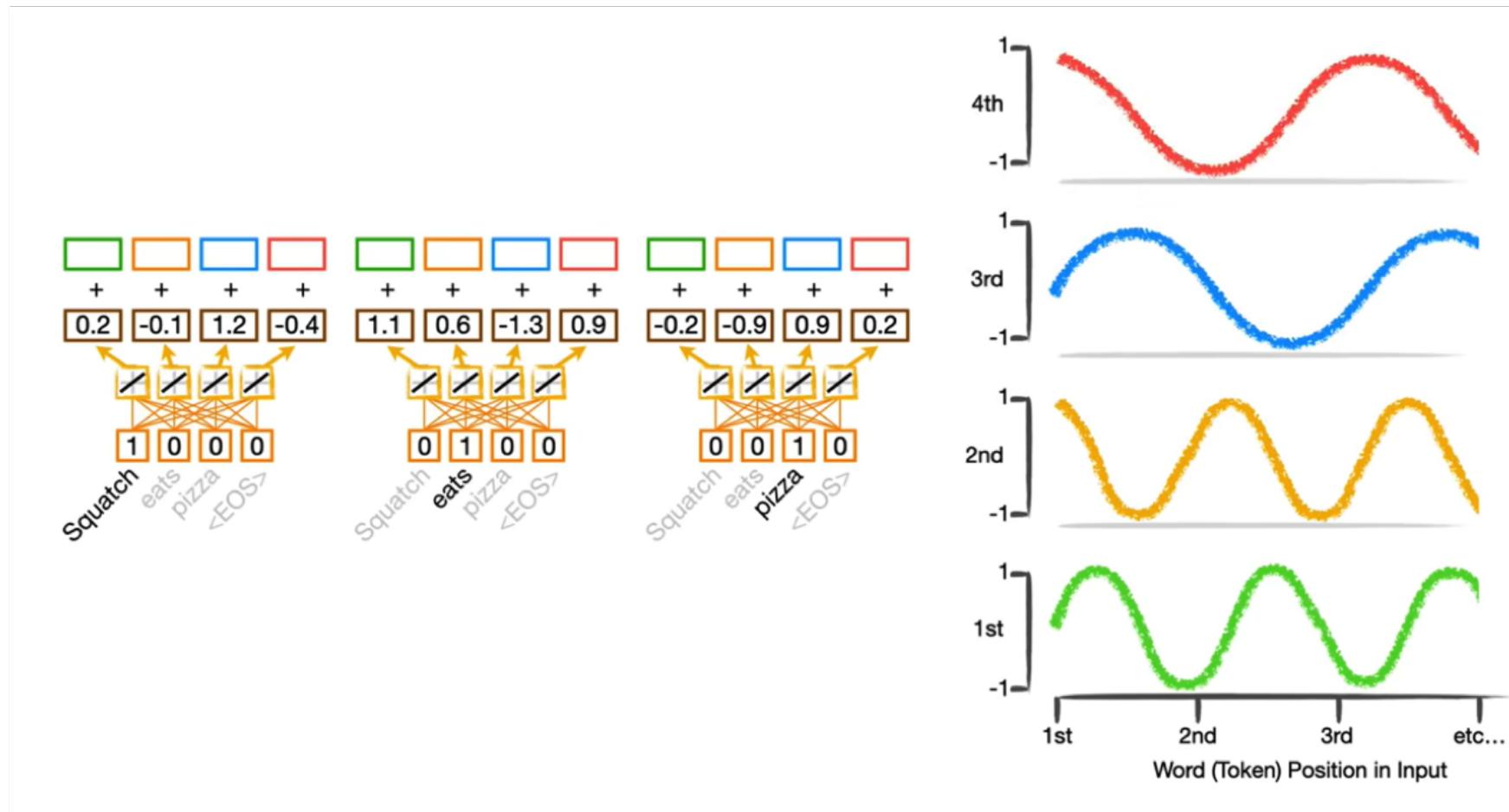
- Assume the following word embeddings for the sentence "Squatch eats pizza", originating from CBOW with $d=4$:



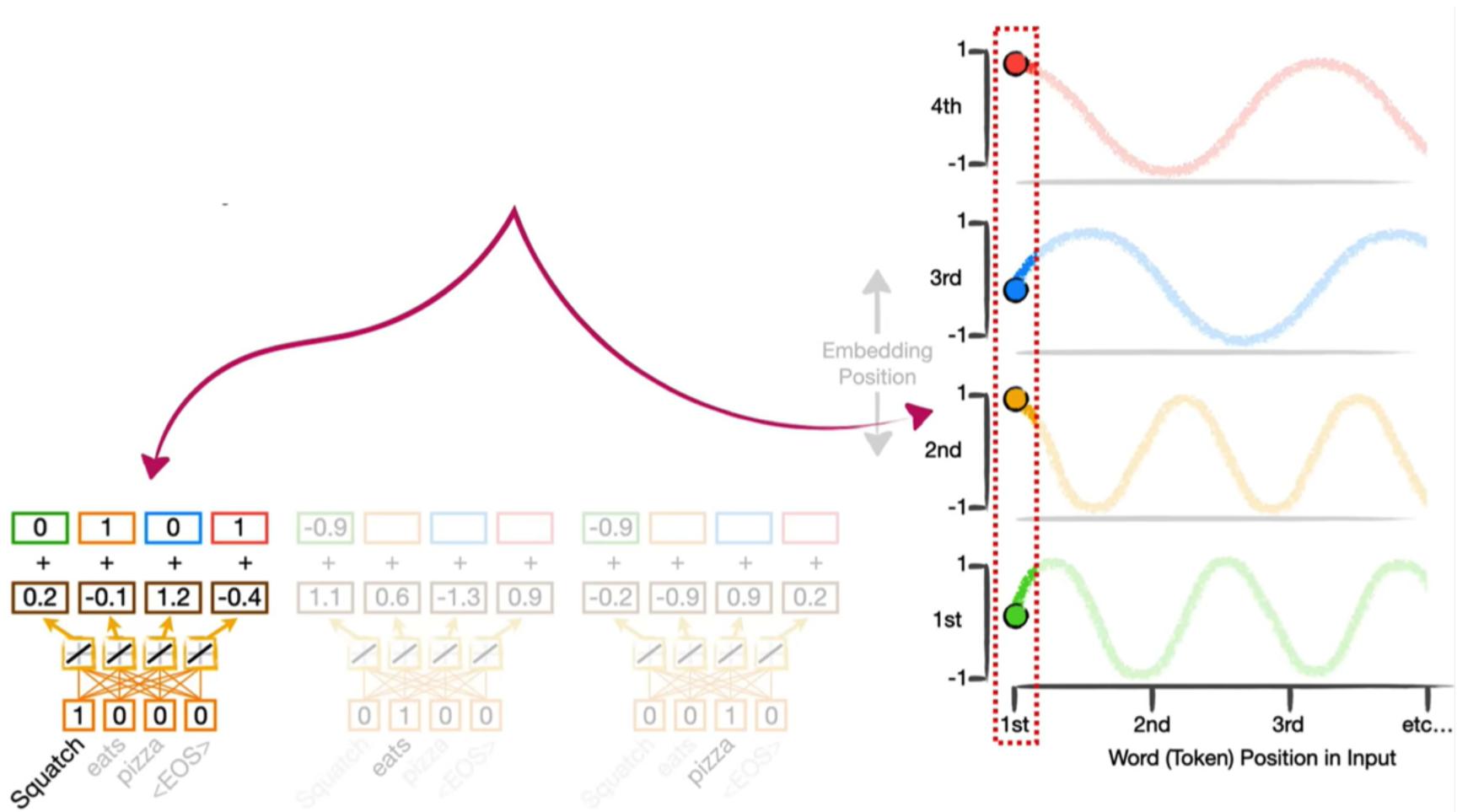
- A set of numbers are added that correspond to the word order:



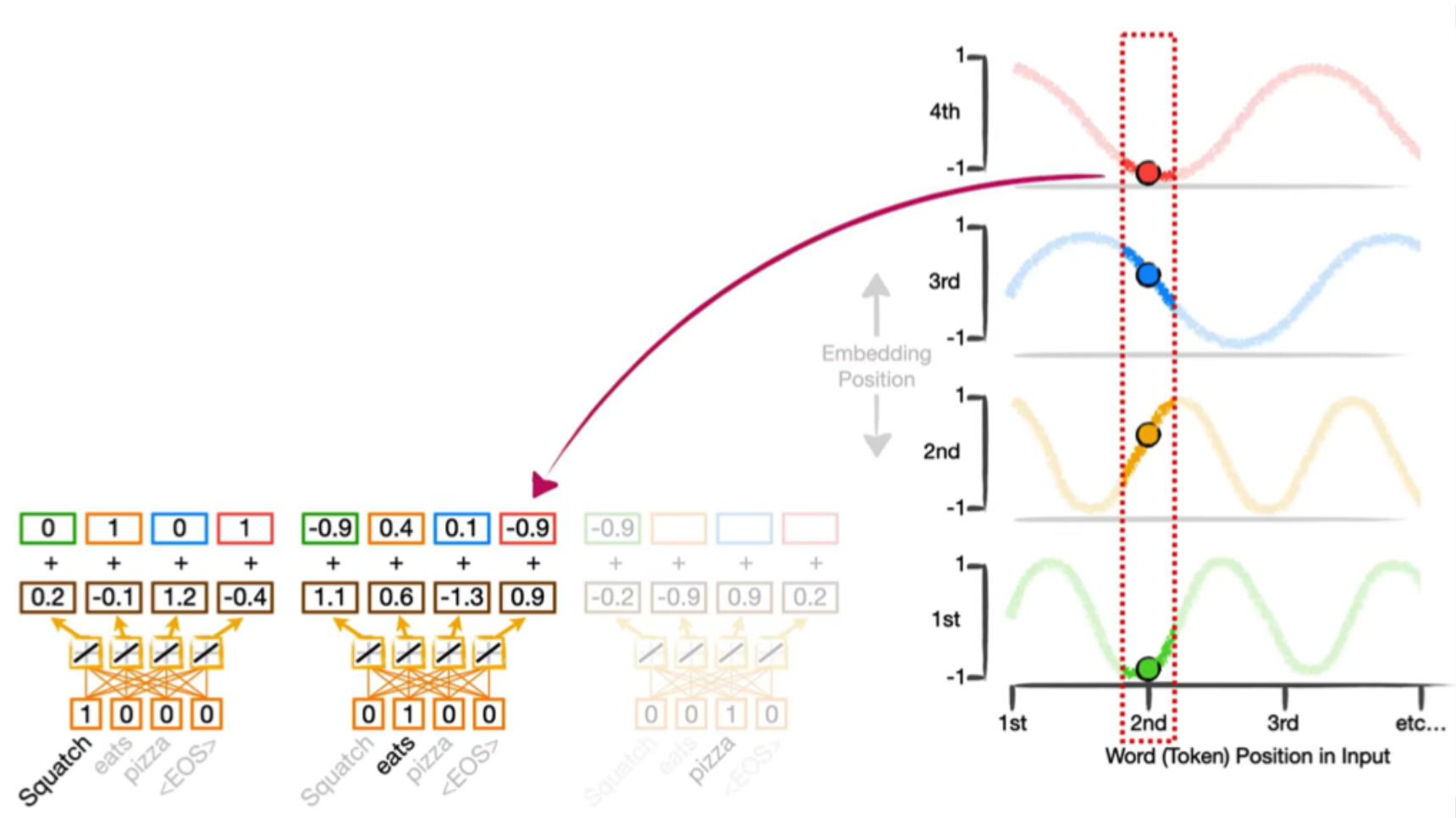
- The numbers that represent the word order come from a sequence of alternating sine and cosine functions:

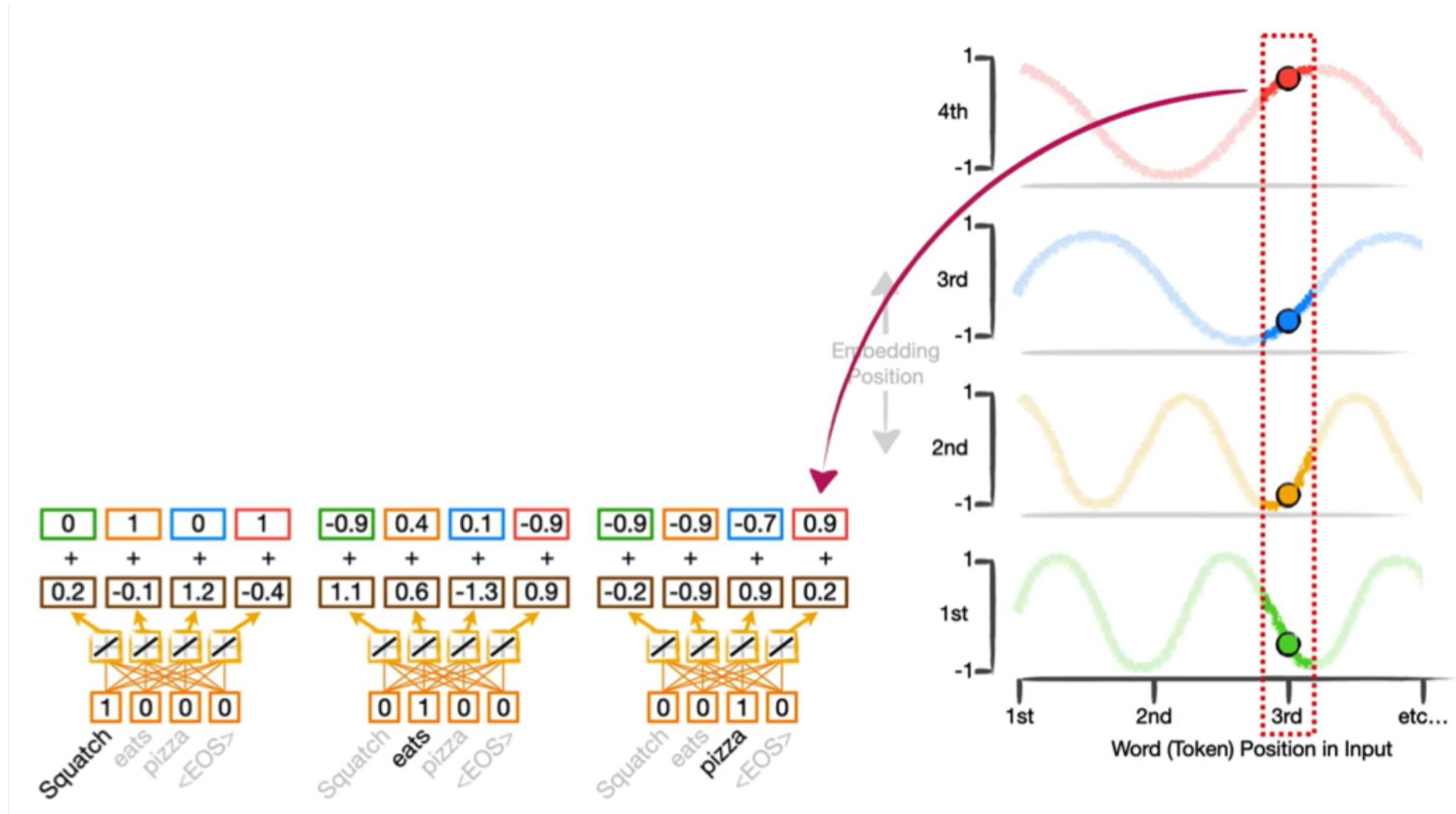


- Each function gives us specific values for each words embedding. For example, the position values for the first word come from the corresponding y-axis coordinates on the functions:



- For the second and third word, a similar approach is followed:



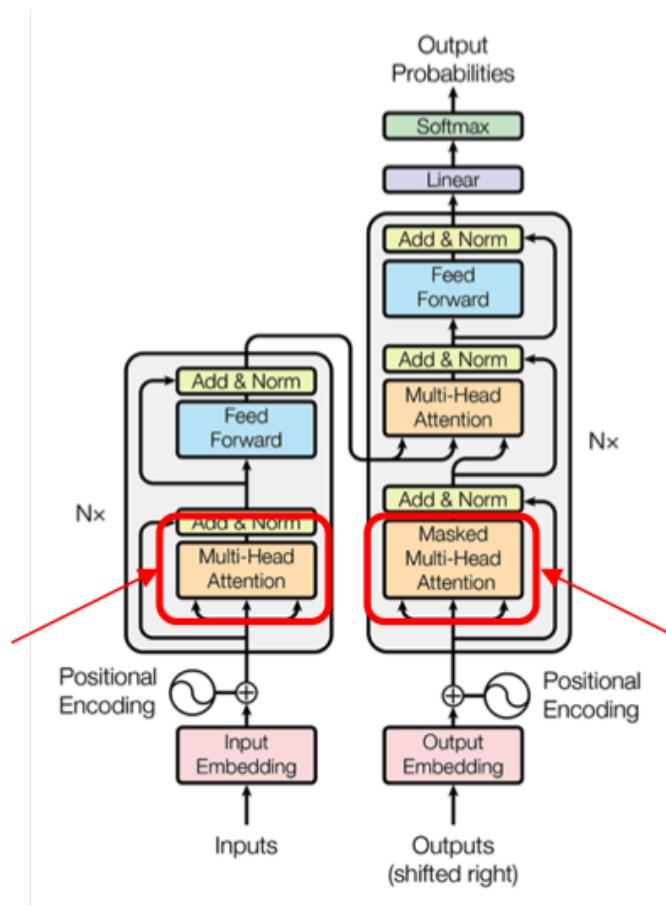


- **Question:** But why are alternating sine and cosine functions suitable for this task?
- **Answer:**
 - Output of sine and cosine are between -1 and 1, which is normalized. As consequence, output values don't become exponentially large in size, which is often more manageable for complex NN.
 - By choosing a reasonable embedding dimensionality d , **unique representations for each position** will be obtained due to the varying wave lengths of the functions.

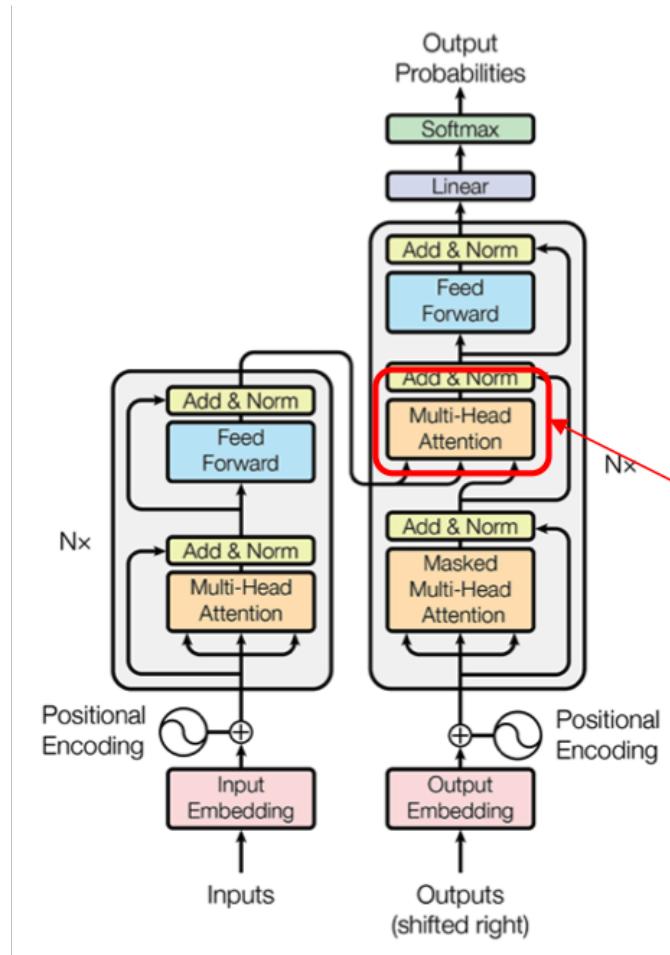
SUMMARY:

- The **self-attention mechanism** makes sure that the similarity between a specific word, and all other words in a sentence is taken into account.
- **Residual connections** and **normalizing layers** make it easier to train the model by allowing the Self-attention layer to establish relationships among the words *without having to also preserve the word embedding and position encoding information*, and keeping the values of a hidden layer in a range that facilitates gradient-based training.

- To capture word similarity and different relationships within the input and output sentence, separately, **multi-head attention layers** are both present in the encoder and decoder part:

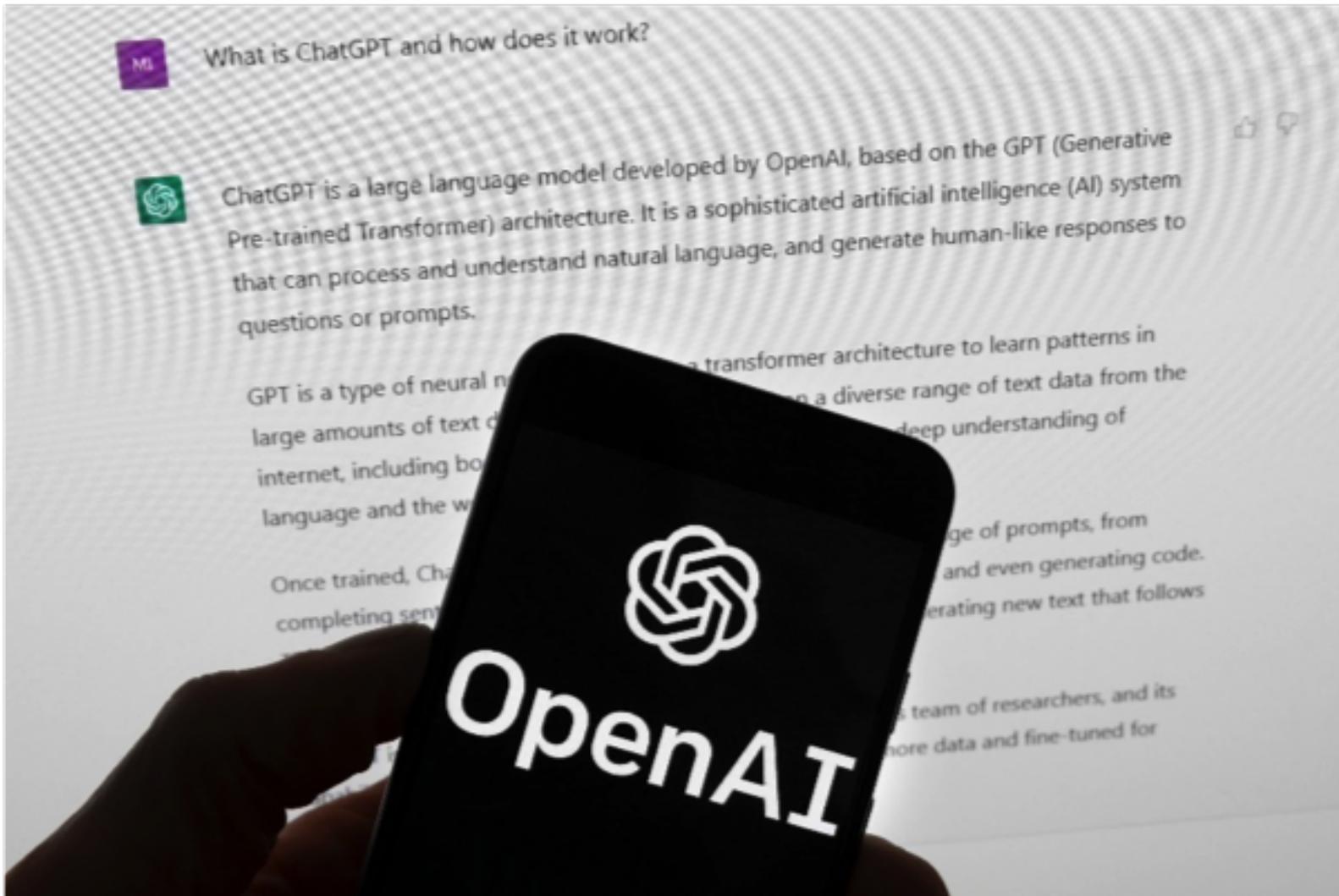


- To capture word similarity and different relationships between the input and output sentence, a **multi-head attention (Encoder-Decoder) layer** is present to allow the decoder to keep track of the significant words in the input:



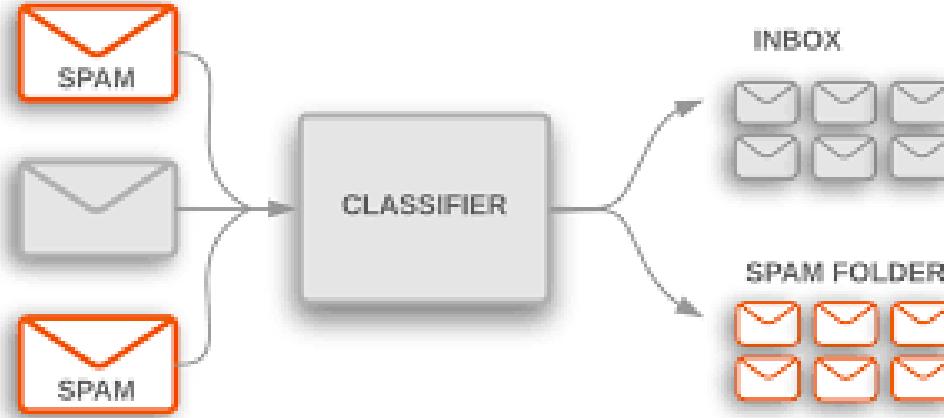
Application of Transformers: A few examples

- In 2018, an encoder-only transformer was used in the (more than 1B-sized) BERT model, improving upon ELMo.
- In 2020, vision transformer and speech-processing convolution-augmented transformer outperformed recurrent neural networks, previously used for vision and speech.
- In 2020, difficulties with converging the original transformer were solved by normalizing layers before (instead of after) multiheaded attention by Xiong et al. This is called pre-LN Transformer.
- In 2023, uni-directional ("autoregressive") transformers were being used in the (more than 100B-sized) GPT-3 and other OpenAI GPT models.



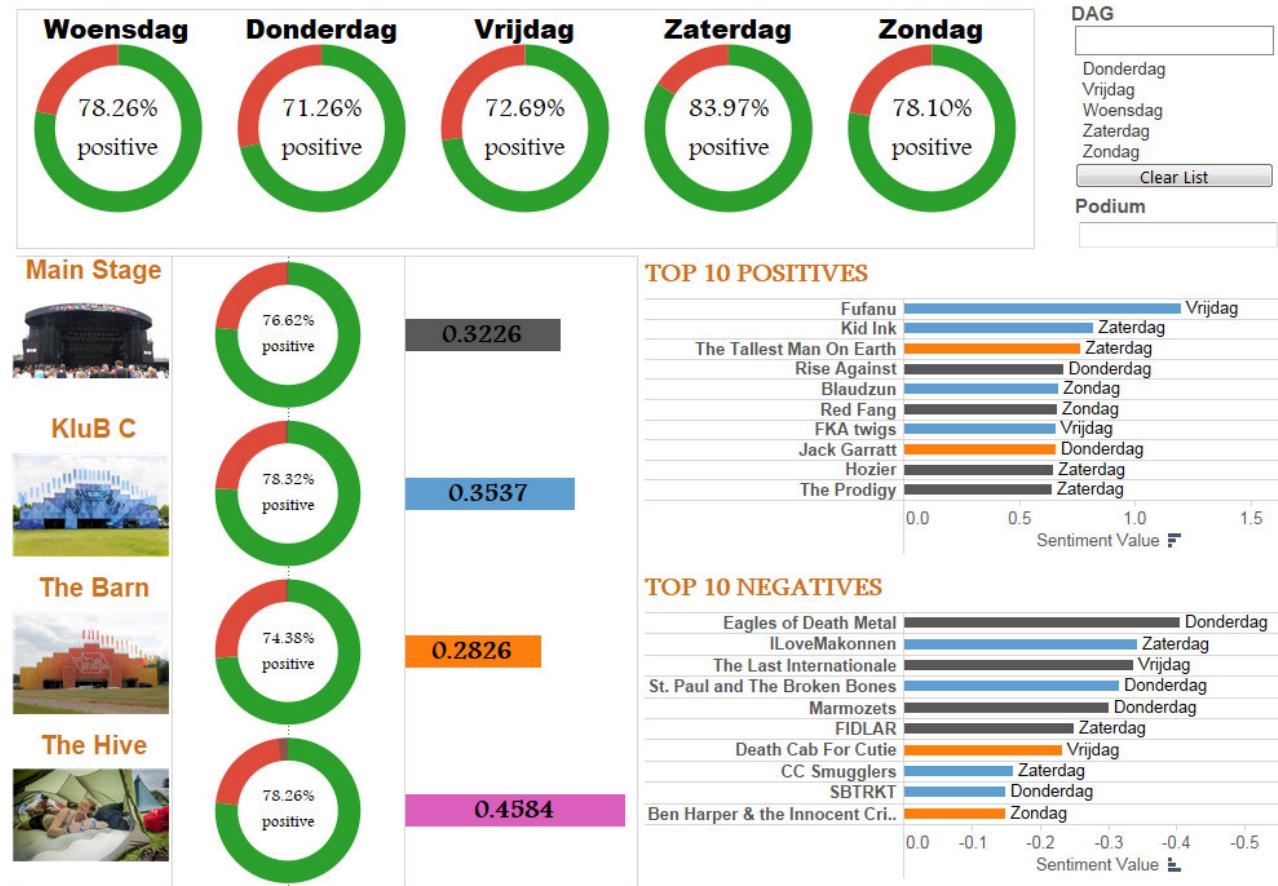
5.2 Text classification

- It is essential in NLP to organize, structure and categorize complex text, turning it into meaningful data
- This is necessary in a variety of applications, e.g.,
 - **Email filtering**



• Sentiment analysis/opinion mining

Rock Werchter: Was it amazing or not? (Sentiment Analysis on Twitter Tweets)



● Topic analysis

Topics

gene	0.04
dna	0.02
genetic	0.01
...	

life	0.02
evolve	0.01
organism	0.01
...	

brain	0.04
neuron	0.02
nerve	0.01
...	

data	0.02
number	0.02
computer	0.01
...	

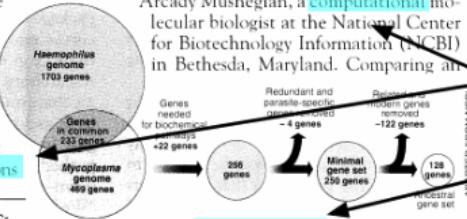
Documents

Seeking Life's Bare (Genetic) Necessities

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

Although the numbers don't match precisely, those predictions

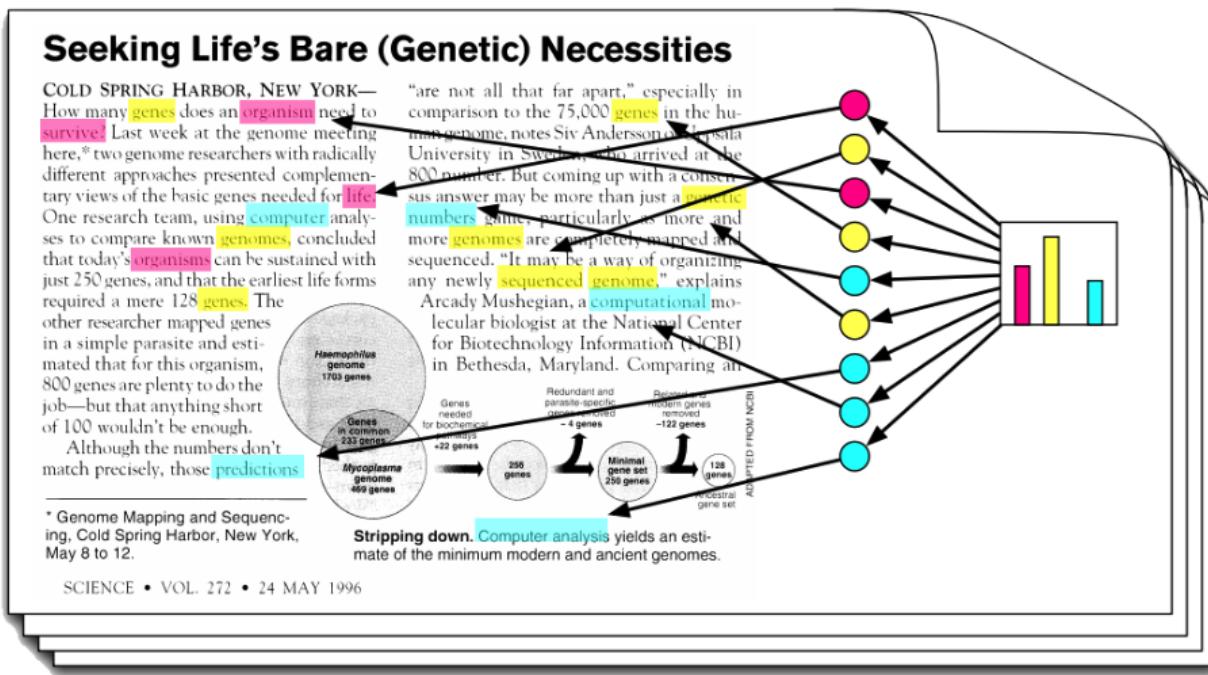
"are not all that far apart," especially in comparison to the 75,000 genes in the human genome, notes Siv Andersson of Uppsala University in Sweden, who arrived at the 800 number. But coming up with a conservative answer may be more than just a genetic numbers game, particularly as more and more genomes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



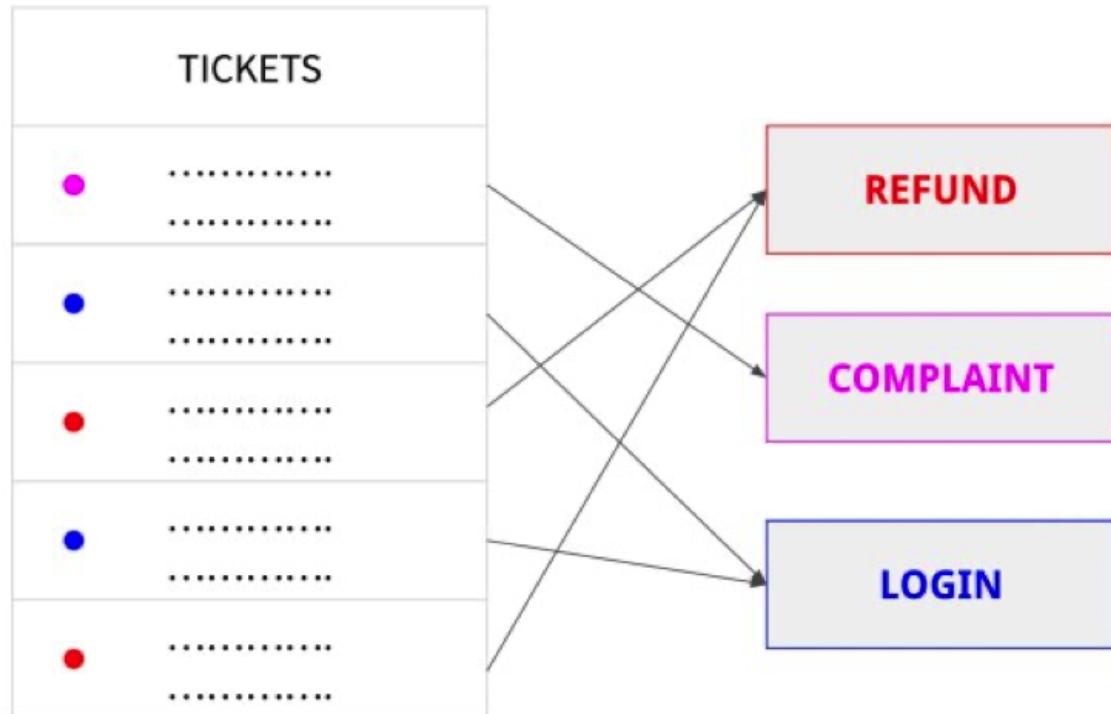
Stripping down. Computer analysis yields an estimate of the minimum modern and ancient genomes.

SCIENCE • VOL. 272 • 24 MAY 1996

Topic proportions & assignments



- Intent detection



- **Question:** How can we derive these outcomes?

- **Answer: Text classification**
- **Text classification** = Process of assigning categories (tags) to unstructured text data
- In what follows, we will discuss two commonly used practices of text classification, i.e., **(1) sentiment analysis** (often referred as **opinion mining**) and **(2) topic modelling**, respectively.

(1) Sentiment analysis

- **Sentiment analysis** = Identify and extract sentiment or opinion from within text (positive, negative, neutral, etc.);
- It can focus on **polarity** of opinion (positive, negative, neutral), personal feelings (angry, happy, sad, etc.), and intentions or objectives (interested/not interested);

Emotion Mood Face Chart

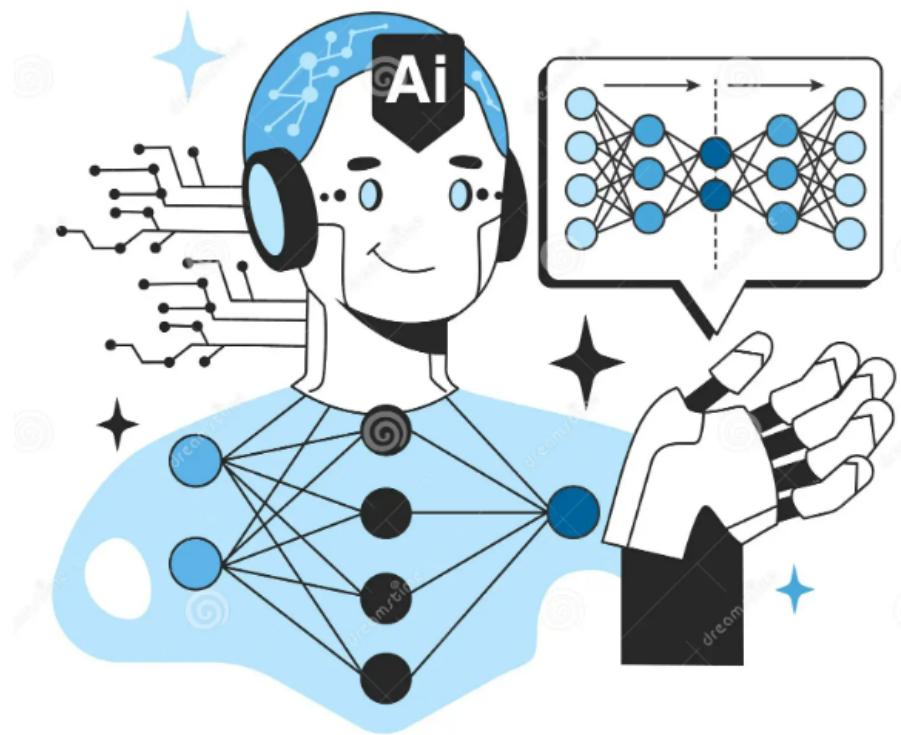


- **Important application area: Marketing & Retail**

→ Based on sentiments, marketeers & retailers can create products and services that meet their client needs!



- To perform this task, several **techniques** exist, e.g., logistic & multinomial regression (statistical), feedforward NN (deep learning);
- Here, we will discuss opinion classification (positive, negative, neutral) with a **feedforward NN**, based on **(1) features** and **(2) word embeddings**!



(1) Based on features

- Consider the sentence "dessert was great";
- Assume hand-built human-engineered features as input layer, e.g., input element x_i could be scalar features:
 - $x_1 = \#$ words in a sentence;
 - $x_2 = \#$ positive lexicon words in a sentence;
 - $x_3 = \begin{cases} 1 & \text{if "no" is in the sentence} \\ 0 & \text{otherwise} \end{cases}$
- The output layer \hat{y} denotes the estimated probability of the sentiments, i.e., \hat{y}_1 the probability of positive sentiment, \hat{y}_2 the probability of negative sentiment, and \hat{y}_3 the probability of neutral;

- Architecture of the feedforward NN framework:

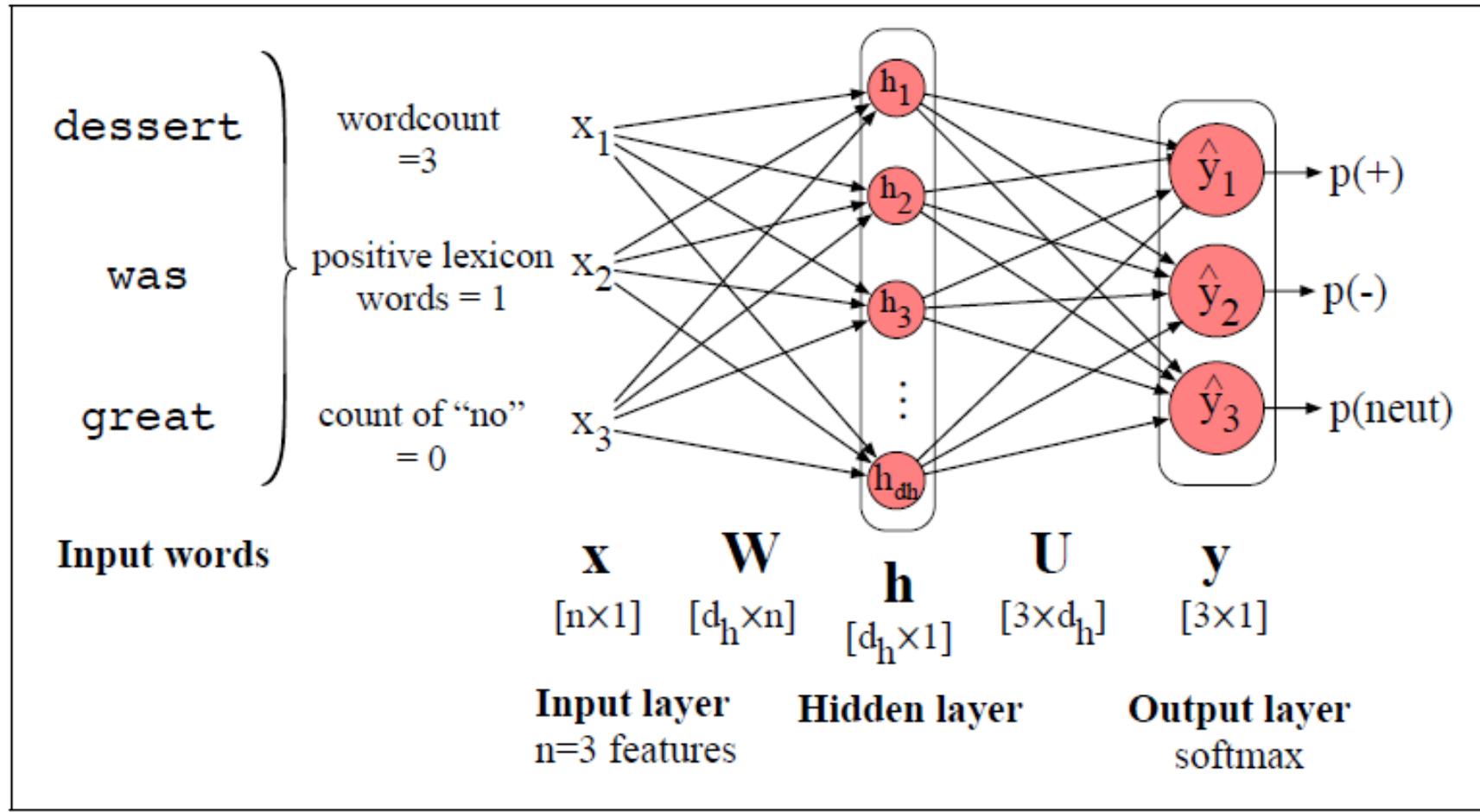


Figure 10 Feedforward network sentiment analysis using traditional hand-built features of the input text.

- The equations of this framework can be formulated as follows:

$$\mathbf{x} = [\mathbf{x}_1; \mathbf{x}_2; \mathbf{x}_3] \quad (\text{each } \mathbf{x}_i \text{ is a hand-designed feature})$$

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

$$\mathbf{z} = \mathbf{U}\mathbf{h}$$

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

- **Limitation:** Since the number of features are limited, a loss of information is present
- **Possible solution: Word embeddings!**

(2) Based on word embeddings

- Reconsider the sentence "dessert was great", with $w_1 = \text{dessert}$, $w_2 = \text{was}$, $w_3 = \text{great}$;
- We can create 3 embeddings from these words, i.e., \mathbf{e}_1 , \mathbf{e}_2 , \mathbf{e}_3 (each of dimensionality d)
- A basic, yet simple idea is to apply some sort of "pooling" function to these embeddings, e.g., by taking their mean:

$$\mathbf{x}_{mean} = \frac{1}{n} \sum_{i=1}^3 \mathbf{e}_i$$

- **Architecture of the feedforward NN framework:**

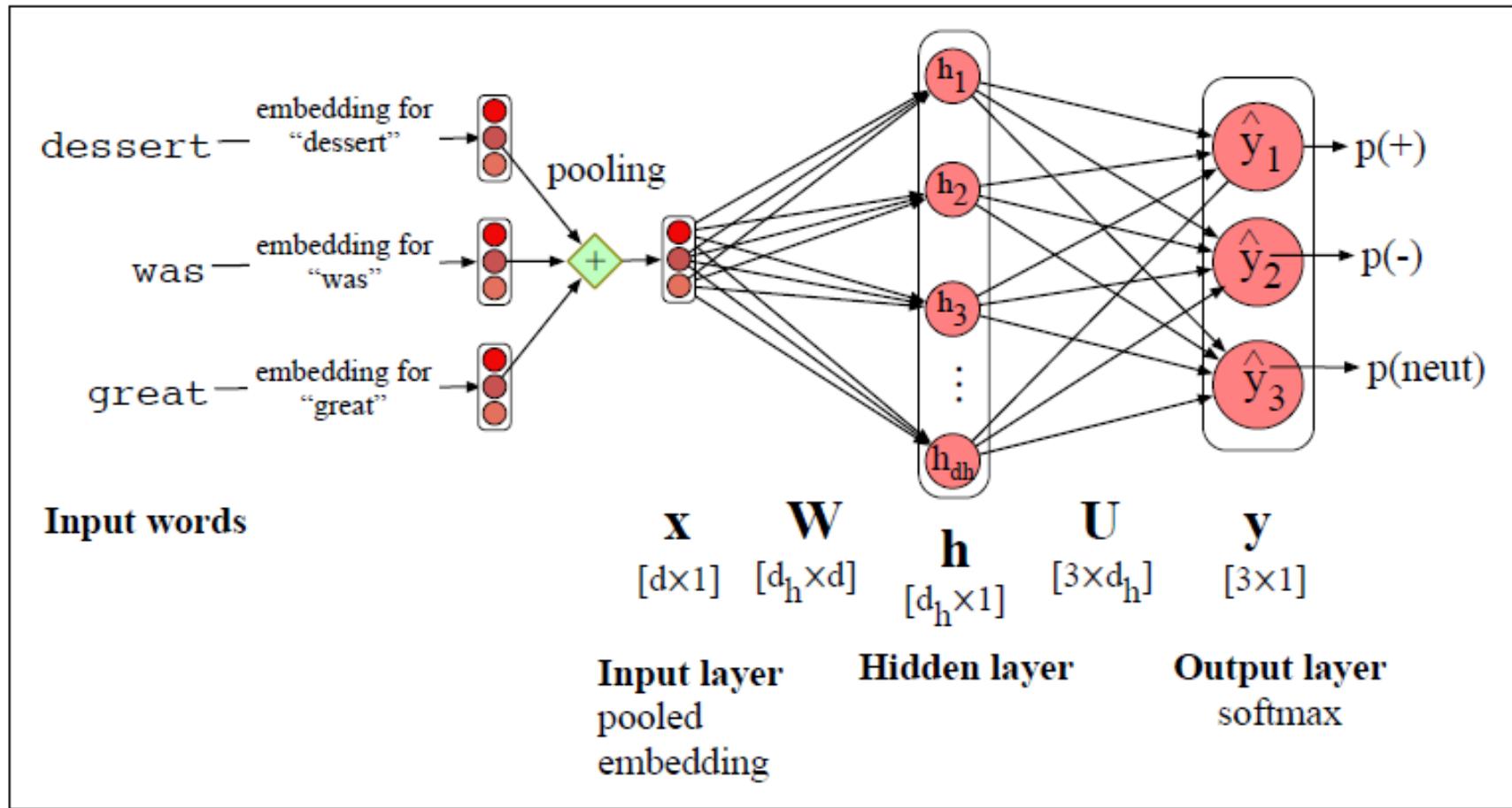


Figure 11 Feedforward network sentiment analysis using a pooled embedding of the input words.

- The equations of this framework can be formulated as follows:

$$\mathbf{x} = \text{mean} [\mathbf{e}_1; \mathbf{e}_2; \mathbf{e}_3]$$

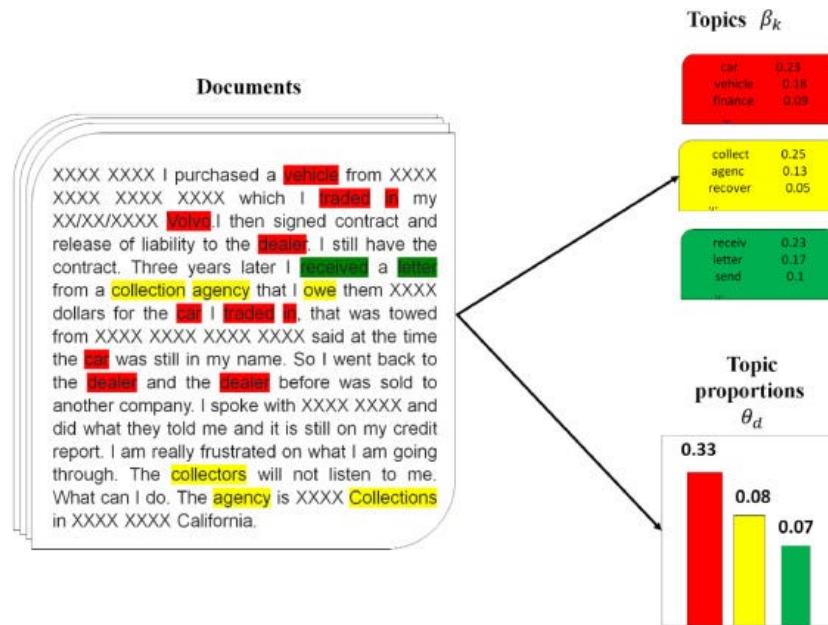
$$\mathbf{h} = \sigma (\mathbf{Wx} + \mathbf{b})$$

$$\mathbf{z} = \mathbf{Uh}$$

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

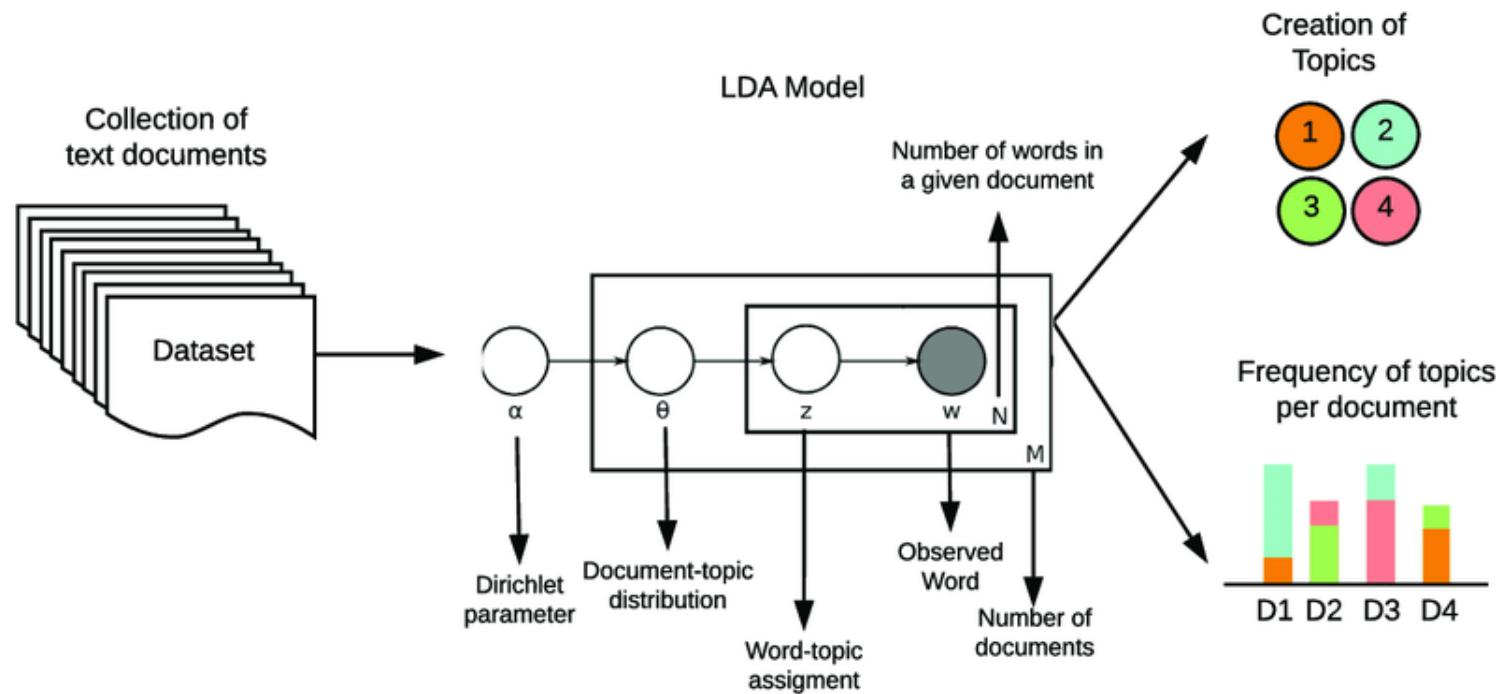
(2) Topic modelling

- **Topic modelling** = A statistical modeling technique to identify clusters or groups of similar words within a body text.
- It uses the semantic structures in text to identify common themes (topics)



- **Purpose of this technique:**
 - Discover hidden (latent) themes in the collection
 - Classifying documents into the discovered themes
 - Use this classification technique to organize/summarize/search documents
- By annotating a document, based on the topics predicted by the modelling method, we are able to optimize our search process
- **Question:** But what topic models exist that enables the prediction of (latent) topics?

- **Answer: Probabilistic latent semantic indexing (PLSI) & latent Dirichlet allocation (LDA)**



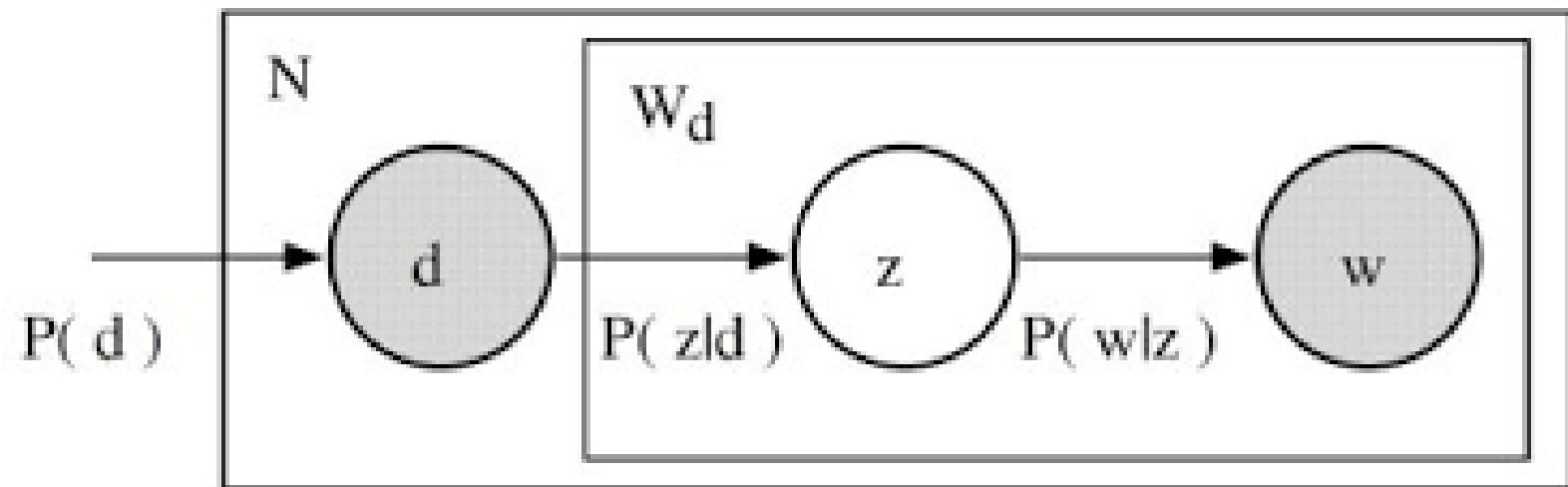
- In what follows, we will discuss the (1) PLSI model & (2) LDA model!

(1) Probabilistic latent semantic indexing (PLSI)

- **Probabilistic latent semantic indexing (PLSI)** = Statistical technique for the analysis of co-occurrence data
- It can be seen as a **latent variable model**, i.e., a statistical model where the **latent/hidden variables** are associated with the **observed variables**
 - Latent/hidden variable: Topic Z ($z \in \{z_1, \dots, z_K\}$)
 - Observed variables:
 - Document D ($d \in \{d_1, \dots, d_N\}$)
 - Word W ($w \in \{w_1, \dots, w_W\}$)

- PLSI associate topic z with observed word-document co-occurrence (w, d) via a **generative process**, as follows:
 1. Sample a document $d \in \{d_1, \dots, d_N\}$ from a multinomial distribution with probability $P(d)$
 2. For each word w_i ($i \in \{1, \dots, W_d\}$) in document d :
 - (a) Select a topic $z \in \{z_1, \dots, z_K\}$ from a multinomial distribution conditioned on the given document with probability $P(z | d)$
 - (b) Select a word w from a multinomial distribution conditioned on the previous chosen topic with probability $P(w | z)$

- Graphical representation of PLSI:



- **Assumptions:**
- **Bag-of-words**, i.e., each document is regarded as an unordered collection of words. More precisely, it means that the joint variable (w, d) is independently sampled and, consequently, the joint distribution of the observed data will factorized as a product:

$$P(D, W) = \prod_{(d,w)} P(d, w)$$

- **Conditional independence**, i.e., words and documents are conditionally independent given the topic:

$$\begin{aligned} P(w, d | z) &= P(w | z) \cdot P(d | z) \\ P(w | d, z) &= P(w | z) \end{aligned}$$

- Mathematically, the PLSI model can be completely defined by specifying the joint distribution $P(d, w)$ as follows, hereby:

- Using the **product rule**:

$$\begin{aligned}
 P(d, w) &= P(d) \cdot P(w | d) \\
 P(w | d) &= \sum_{z \in \{z_1, \dots, z_K\}} P(w, z | d) \\
 &= \sum_{z \in \{z_1, \dots, z_K\}} P(w | d, z) \cdot P(z | d)
 \end{aligned}$$

- Using the **conditional independence assumption**:

$$P(w | d) = \sum_{z \in \{z_1, \dots, z_K\}} P(w | z) \cdot P(z | d)$$

$$\begin{aligned}
 P(d, w) &= P(d) \cdot \sum_{z \in \{z_1, \dots, z_K\}} P(w | z) \cdot P(z | d) \\
 &\stackrel{(\star)}{=} \cancel{P(d)} \cdot \sum_{z \in \{z_1, \dots, z_K\}} P(z) \cdot P(d | z) \cdot P(w | z)
 \end{aligned}$$

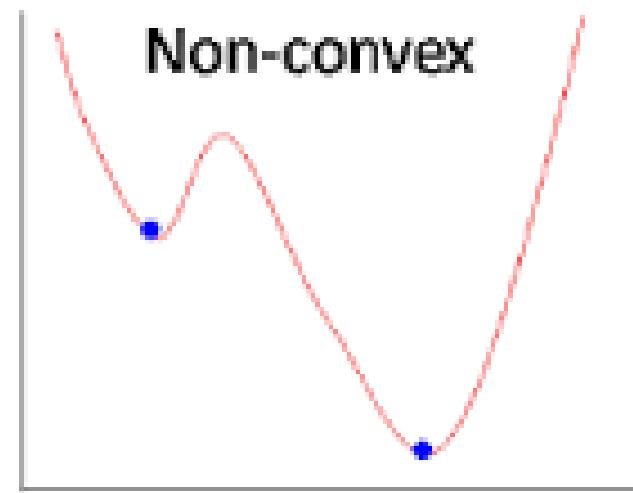
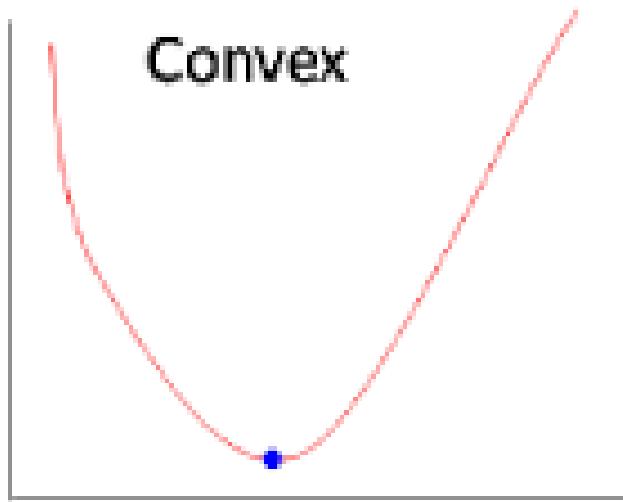
(\star) Using Bayes' rule

- Following the likelihood principle, we can determine $P(d)$, $P(z | d)$ and $P(w | z)$ by minimization of the negative log-likelihood function

$$-l = - \sum_{d \in \{d_1, \dots, d_N\}} \sum_{w \in \{w_1, \dots, w_{W_d}\}} n(d, w) \cdot \log [P(d, w)]$$

- Here, $n(d, w)$ denotes the **term frequency**, i.e., number of times w occurred in document d , observed in the **Word-Document Matrix** (see p.130);

- **Problem:** This is a non-convex optimization problem!



- **Possible solution:** **Expectation-Maximization (EM) algorithm!**

- The **EM algorithm** is used to seek a locally optimal solution;
- **EM** alternates two steps:
 - An **expectation (E) step** computes the posterior probabilities of the latent variable z , based on the current estimates of the parameters:

$$P'(z \mid d, w) = \frac{P(d) \cdot P(z \mid d) \cdot P(w \mid z)}{\sum_{z' \in \{z_1, \dots, z_K\}} P(d) \cdot P(z' \mid d) \cdot P(w \mid z')}$$

where $P'(\cdot)$ indicates the new estimate of the probability for the next step;

- A **maximization (M) step** updates the parameters once the latent variables are known using the posterior estimated in the previous E-step:

$$P'(w \mid z) = \frac{\sum_{d \in \{d_1, \dots, d_N\}} n(d, w) \cdot P'(z \mid d, w)}{\sum_{d \in \{d_1, \dots, d_N\}} \sum_{w' \in \{w_1, \dots, w_{W_d}\}} n(d, w') \cdot P'(z \mid d, w')}$$

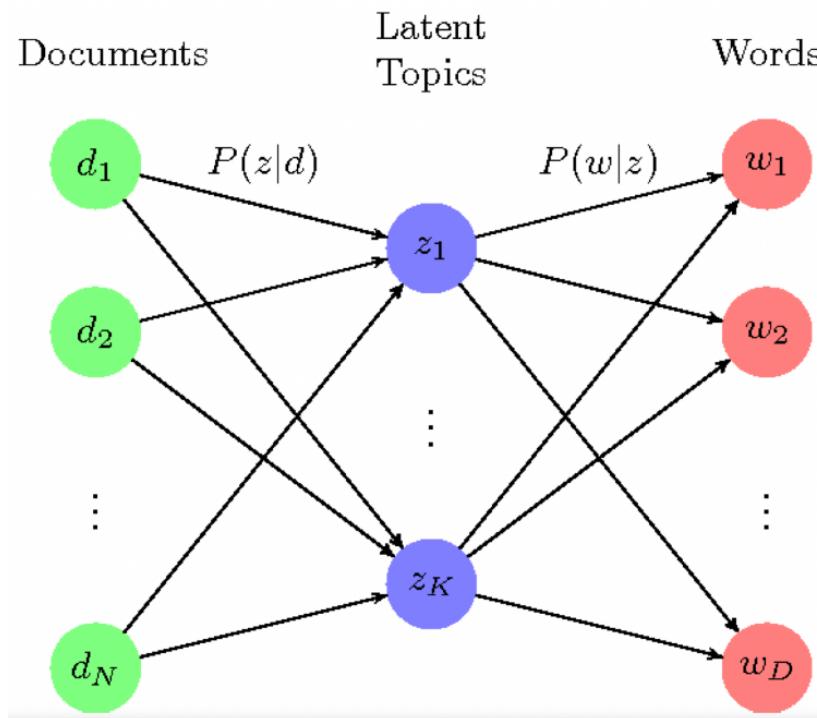
$$P'(d \mid z) = \frac{\sum_{w \in \{w_1, \dots, w_{W_d}\}} n(d, w) \cdot P'(z \mid d, w)}{\sum_{d' \in \{d_1, \dots, d_N\}} \sum_{w \in \{w_1, \dots, w_{W_{d'}}\}} n(d', w) \cdot P'(z \mid d', w)}$$

$$P'(z) = \frac{1}{R} \cdot \sum_{d \in \{d_1, \dots, d_N\}} \sum_{w \in \{w_1, \dots, w_{W_d}\}} n(d, w) \cdot P'(z \mid d, w)$$

$$R = \sum_{d \in \{d_1, \dots, d_N\}} \sum_{w \in \{w_1, \dots, w_{W_d}\}} n(d, w)$$

- **Limitations:**

- The **number of parameters grows linearly with the number of documents**, such that when training over a large number of documents, PLSI tends to overfit the training data



- **Question:** But what are the parameters of this model?

- **Answer:** The two main parameters in the model are as follows:
 - $\frac{P(w \mid z)}{\text{words}}$: $(W_D - 1) \cdot K$ parameters (for every topic z we have W_D words).
 - **Remark:** We subtract 1 from the total number of words per document since the sum of these W_D probabilities should be one, so we lose one degree of freedom!
 - $\frac{P(z \mid d)}{\text{topic}}$: $(K - 1) \cdot N$ parameters (for every document d we have K topics)
- There is no natural way to compute the probability of a "new" document that was not in the training data.
- Therefore, **latent Dirichlet allocation (LDA)** is often used as (better) alternative to overcome these issues!

(2) Latent Dirichlet allocation (LDA)

- Similar to PLSI, LDA assumes that
 - The semantic content of a document is composed by combining one or more words from one or more topics;
 - Certain words are ambiguous, belonging to more than one topic, with different probability

Example: The word "training" can apply to both dogs and cats, but are more likely to refer to dogs;

- LDA, unlike PLSI, takes into account two important concerns:
 - Most documents will contain only a relatively small number of topics. In the collection, individual topics will occur with different frequencies;

- Within a topic, certain words will be used much more frequently than others.
- To account for these concerns, LDA assigns **(prior) probability distributions** to them, i.e.,
 - Topics within a document will have a probability distribution such that a given document is more likely to contain some topics than others
 - Words within a topic will also have their own probability distribution;
- **Question:** What kind of prior distributions are chosen in LDA?

- **Answer: Dirichlet distributions!**

- **Definition:**

Dirichlet distribution $\text{Dir}(\alpha) =$ A family of continuous multivariate probability distributions parametrized by a K -dimensional vector $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_K)$ of positive reals.

The Dirichlet distribution of order $K \geq 2$ with parameters $\alpha_1, \alpha_2, \dots, \alpha_K > 0$ has the following probability density function:

$$f(x_1, x_2, \dots, x_K; \alpha_1, \alpha_2, \dots, \alpha_K) = \frac{1}{B(\alpha)} \prod_{k=1}^K x_k^{\alpha_k - 1},$$

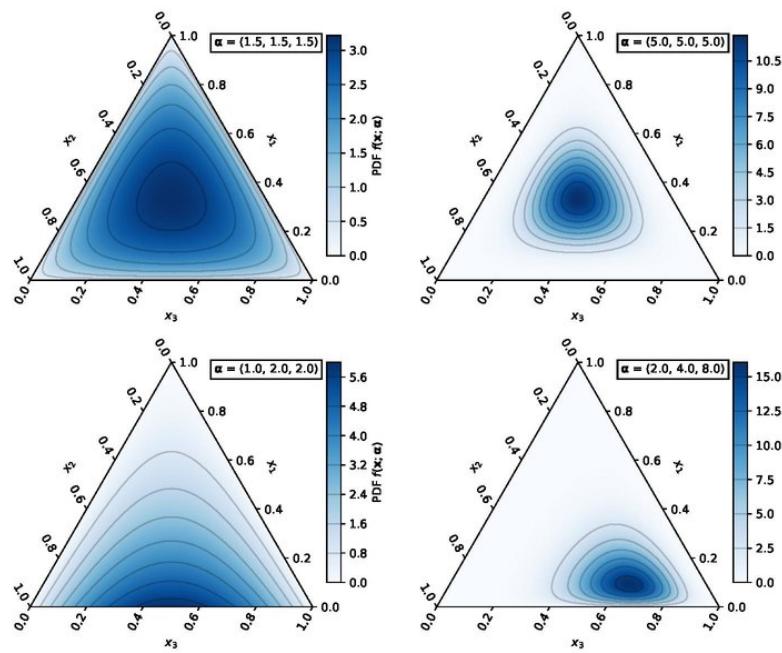
where $\{x_k\}_{k=1}^K$ belong to the standard $K - 1$ simplex, i.e.,

$$\sum_{k=1}^K x_k = 1 \text{ and } x_k \in [0, 1] \text{ for all } k \in \{1, \dots, K\},$$

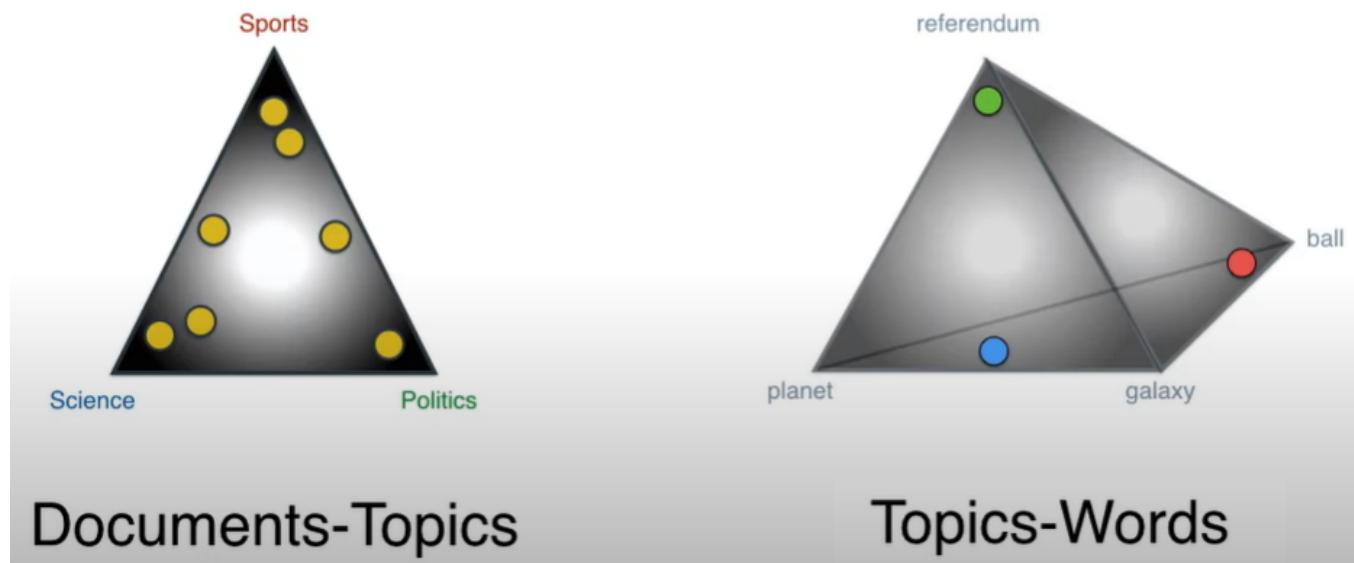
and the normalizing constant $B(\alpha)$ given by

$$B(\alpha) = \frac{\prod_{k=1}^K \Gamma(\alpha_k)}{\Gamma\left(\sum_{k=1}^K \alpha_k\right)}.$$

- **Example:** $K=3$



- It is essentially a multivariate generalization of the Beta distribution
- In LDA, **two Dirichlet distributions are assumed**, i.e., one associating documents with corresponding topics (left) and one associating topics with corresponding words (right):



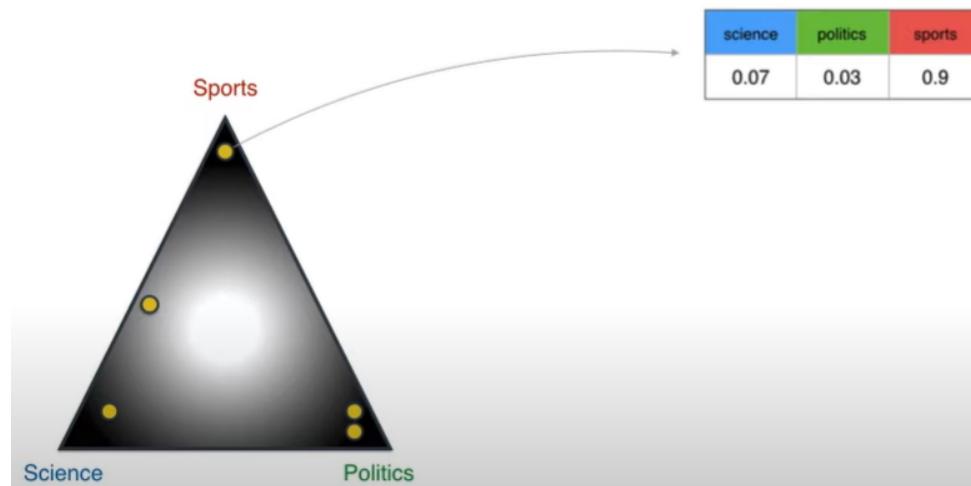
- In this example: 7 documents, 3 topics (science, politics, sports), and 4 words (referendum, ball, planet, galaxy)

- Let's dive in each of these distributions:

- Documents-Topics:**

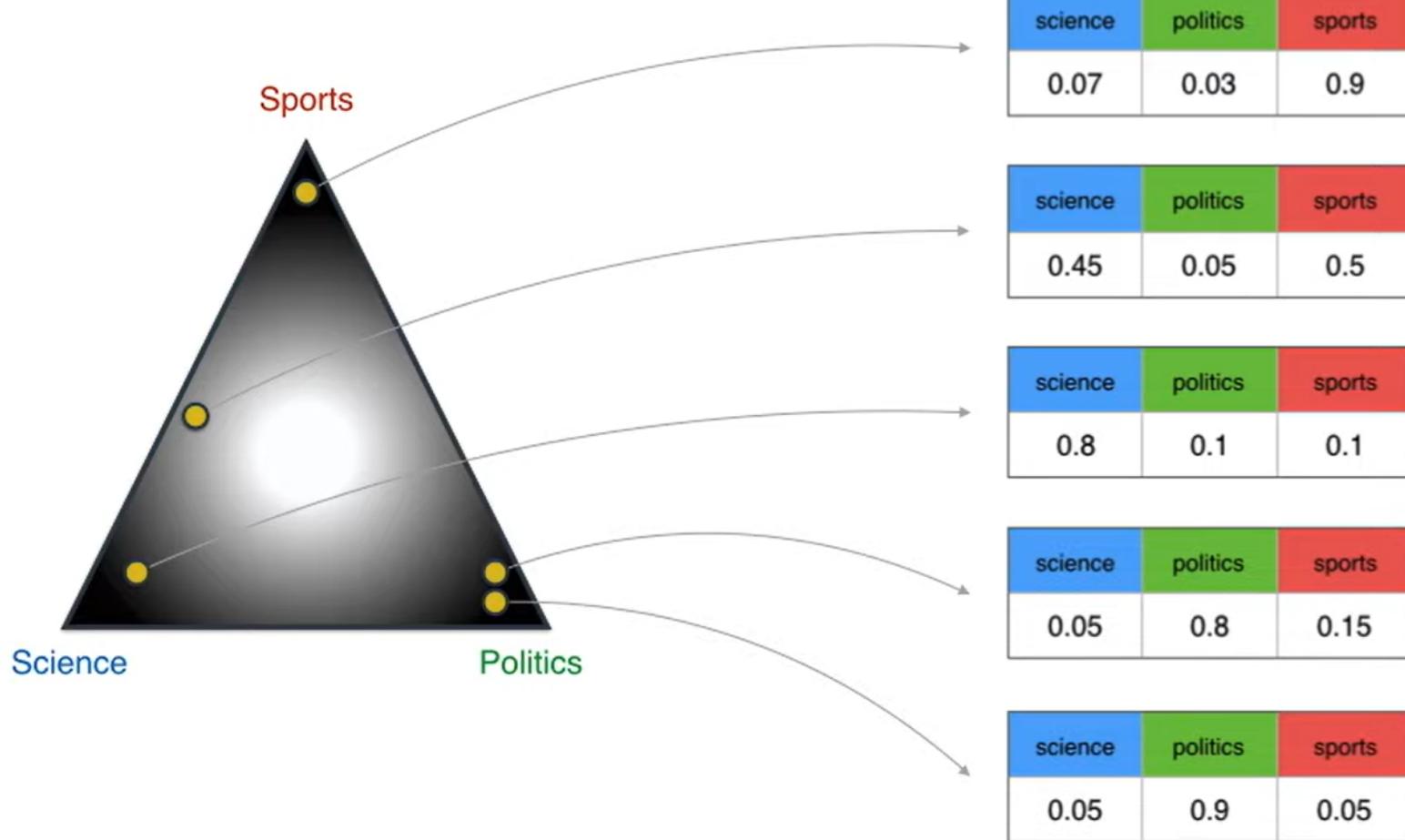
- Consider a specific choose for $\alpha = (\alpha_1, \alpha_2, \alpha_3)$ within the Dirichlet distribution. For every **document**, the inputs $\{x_k\}_{k=1}^3 = \{\text{science}, \text{politics}, \text{sports}\}$ indicates the percentage of "association" within the document, based on the chosen Dirichlet distribution;

- Example of 1 document:**



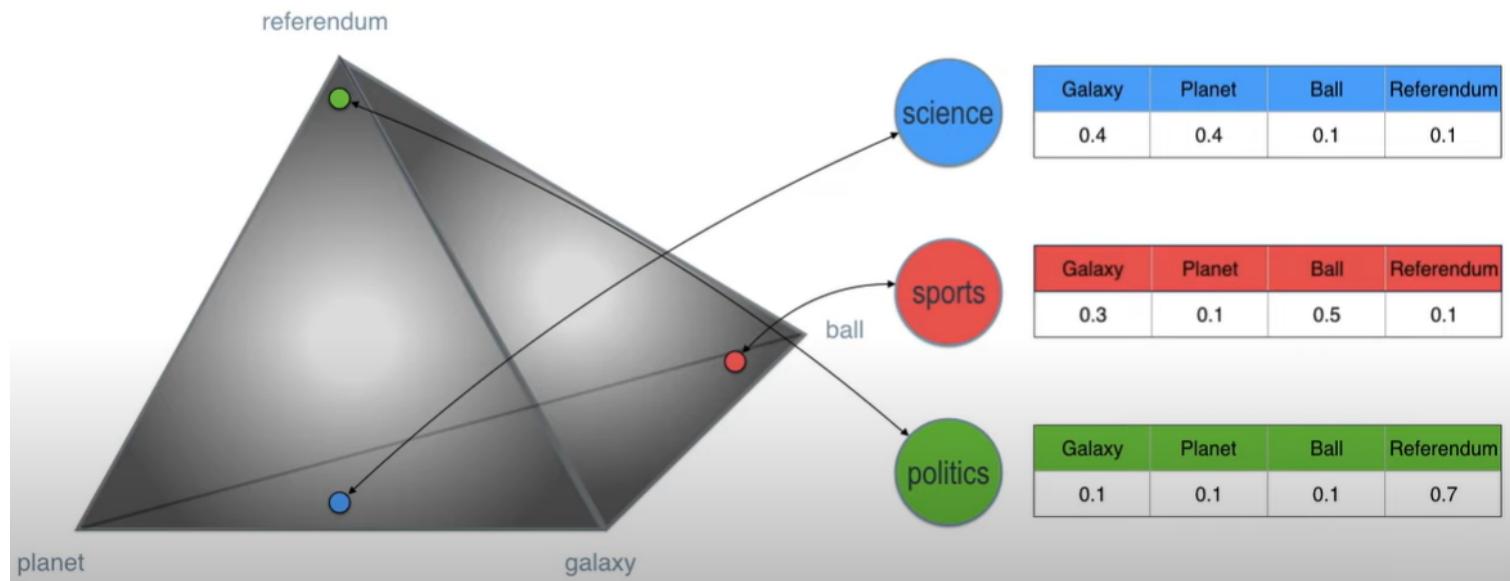
Here, the document consists of 90% **sports**, 7% **science** and 3% **politics**.

- All documents:

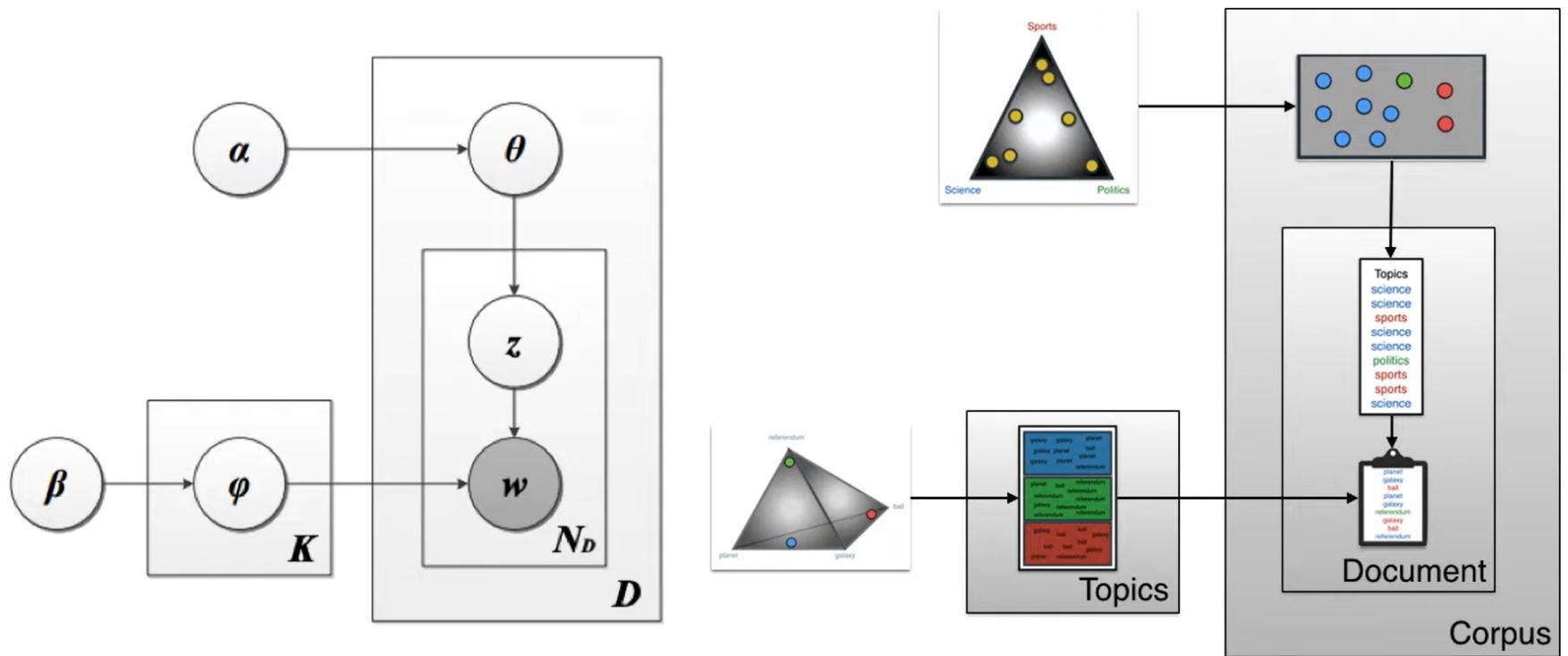


- Topics-Words:

- A similar intuition can be followed here, i.e., for every topic (**sports**, **science**, **politics**), the inputs $\{x_k\}_{k=1}^4 = \{\text{referendum}, \text{ball}, \text{planet}, \text{galaxy}\}$ gives the percentages of "association" within the topic, based on a specific prior Dirichlet distribution choice $\text{Dir}(\beta)$;



- Depending on the given priors, words, and topics, LDA can now develop a new document, based on its following framework:



- **Question:** But why are Dirichlet distributions chosen in this framework?
- **Answer:** Dirichlet distributions are **conjugate priors** for multinomial distributions!
 - **Bayesian theory:** A distributions $p(\theta)$ is said to be a **conjugate prior** if the posterior distribution $p(\theta | \mathbf{x})$ is in the **same probability distribution family** as the prior distribution $p(\theta)$;

- Here,

$$\begin{aligned}
 \alpha &= (\alpha_1, \dots, \alpha_K) \\
 \mathbf{p} | \alpha &= (p_1, \dots, p_K) \sim \text{Dir}(K, \alpha) \\
 \mathbf{X} | \mathbf{p} &= (x_1, \dots, x_K) \sim \text{Multinomial}(K, \mathbf{p}) \\
 &\quad \downarrow \\
 \mathbf{c} &= (c_1, \dots, c_K) = \text{number of occurrences of category } i \\
 \mathbf{p} | X, \alpha &= \sim \text{Dir}(K, \mathbf{c} + \alpha) = \text{Dir}(K, c_1 + \alpha_1, \dots, c_K + \alpha_K)
 \end{aligned}$$

- Similar to PLSI, LDA can be seen as a **generative process**, with following steps:
 1. Choose $\theta_i \sim \text{Dir}(\alpha)$, where $i \in \{1, \dots, D\}$, with $D = \# \text{ documents}$ and α typical sparse (i.e., $\alpha < 1$);
 2. Choose $\varphi_k \sim \text{Dir}(\beta)$, where $k \in \{1, \dots, K\}$, $K = \# \text{ topics}$ and β typical sparse;
 3. For each word positions i, j , where $i \in \{1, \dots, D\}$, $j \in \{1, \dots, N_i\}$, and N_i the length document i :
 - (a) Select a topic $z_{i,j} \sim \text{Multinomial}(\theta_i)$
 - (b) Select a word $w_{i,j} \sim \text{Multinomial}(\varphi_{z_{i,j}})$

- Given the parameters α and β , the joint distribution of a corpus (consisting of D documents) can be expressed as follows:

$$p(\theta, \varphi, \mathbf{z}, \mathbf{w} \mid \alpha, \beta) = \prod_{i=1}^D p(\theta_i \mid \alpha) \prod_{k=1}^K p(\varphi_k \mid \beta) \prod_{j=1}^{N_i} p(z_{i,j} \mid \theta_i) p(w_{i,j} \mid \varphi_{z_{i,j}})$$

- Inference:** To train this model, various techniques exist in the literature, i.e., Gibbs sampling, Variational Bayes, likelihood maximization, etc.

→ Out of scope of this course!

- Remark:** LDA can be seen as a (Bayesian) generalization of the PLSI model, where PLSI is equivalent to LDA under a uniform prior Dirichlet distribution

REFERENCES

- Useful materials can be found on Toledo!
- **Remark:** These materials often go deeper in the highlighted topics of this course, and is available for the interested reader!