Master Degree in Computer Science

Natural Language Processing

Introduction to language models

Prof. Alfio Ferrara

Department of Computer Science, Università degli Studi di Milano
Room 7012 via Celoria 18, 20133 Milano, Italia alfio.ferrara@unimi.it

sed noli modo

A language model in essentially a **probability distribution** over a **sequence of words**

$$p(w_1, w_2, \ldots, w_n)$$

which can be used for a surprisingly high number of tasks, including *document search*, *document classification*, *text summarization*, *text generation*, *machine translation*, and many others

Note: Instead of estimating the probability distribution of words, we can work at a finer granularity on the distribution of substrings of fixed length in words (e.g., characters, 2-chars blocks)

Example 1
A LM may be used to guess the next word in a sequence

$p(w_n \mid w_1, w_2, \ldots, w_{n-1})$

Yesterday → you → studied, → what → are → you → doing → ... ?
→ today

Example 2
Or to guess the author (or any other categorical attribute) of as text

$p(author \mid w_1, w_2, \ldots, w_{n-1})$

"Twenty years from now you will be more disappointed
by the things that you didn't do than by the ones you did do"
→ Mark Twain

Example 3
Or to select the correct translation for a sentence

$p(e_1, e_2, \ldots, e_n \mid w_1, w_2, \ldots, w_n)$

"ci sono molti esempi"
→ "there are many examples"
~~→ "are there many examples"~~

1. **Statistical Language Models**: Estimate the probability distribution of words by enforcing statistical techniques such as n-grams *maximum likelihood estimation (MLE)* or *Hidden Markov Models (HMM)*

2. **Neural Language Models**: Popularized by Bengio et al. 2003, each word is associated with an embedding vector of fixed size and a Neural Network is used to estimate the next word given a sequence of k preceding words

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). *A neural probabilistic language model*. Journal of machine learning research, 3(Feb), 1137-1155

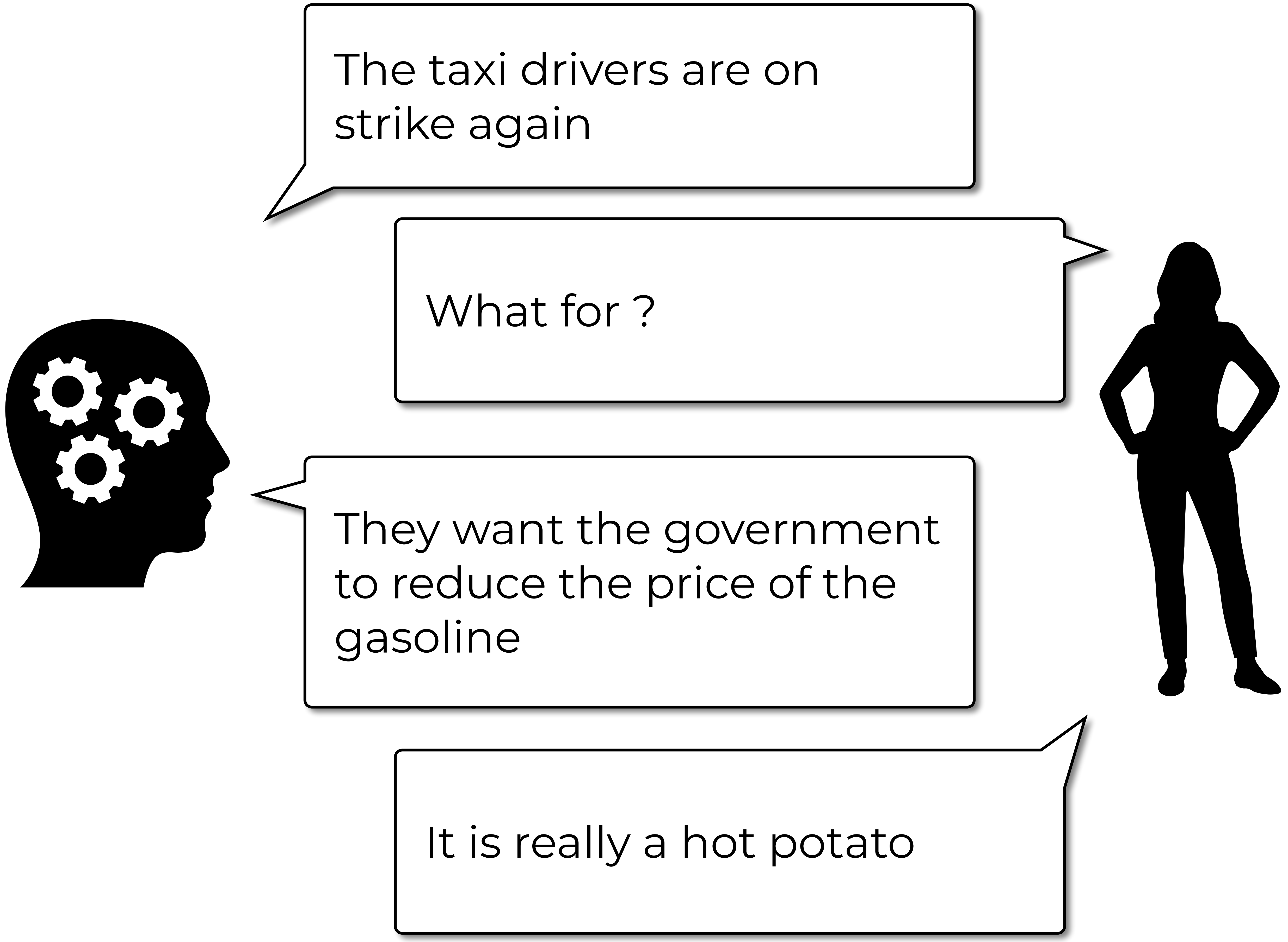
Natural language is often ambiguous and hard to understand

- It’s like throwing the baby out with the bathwater

- Really, Sherlock? No! You are clever

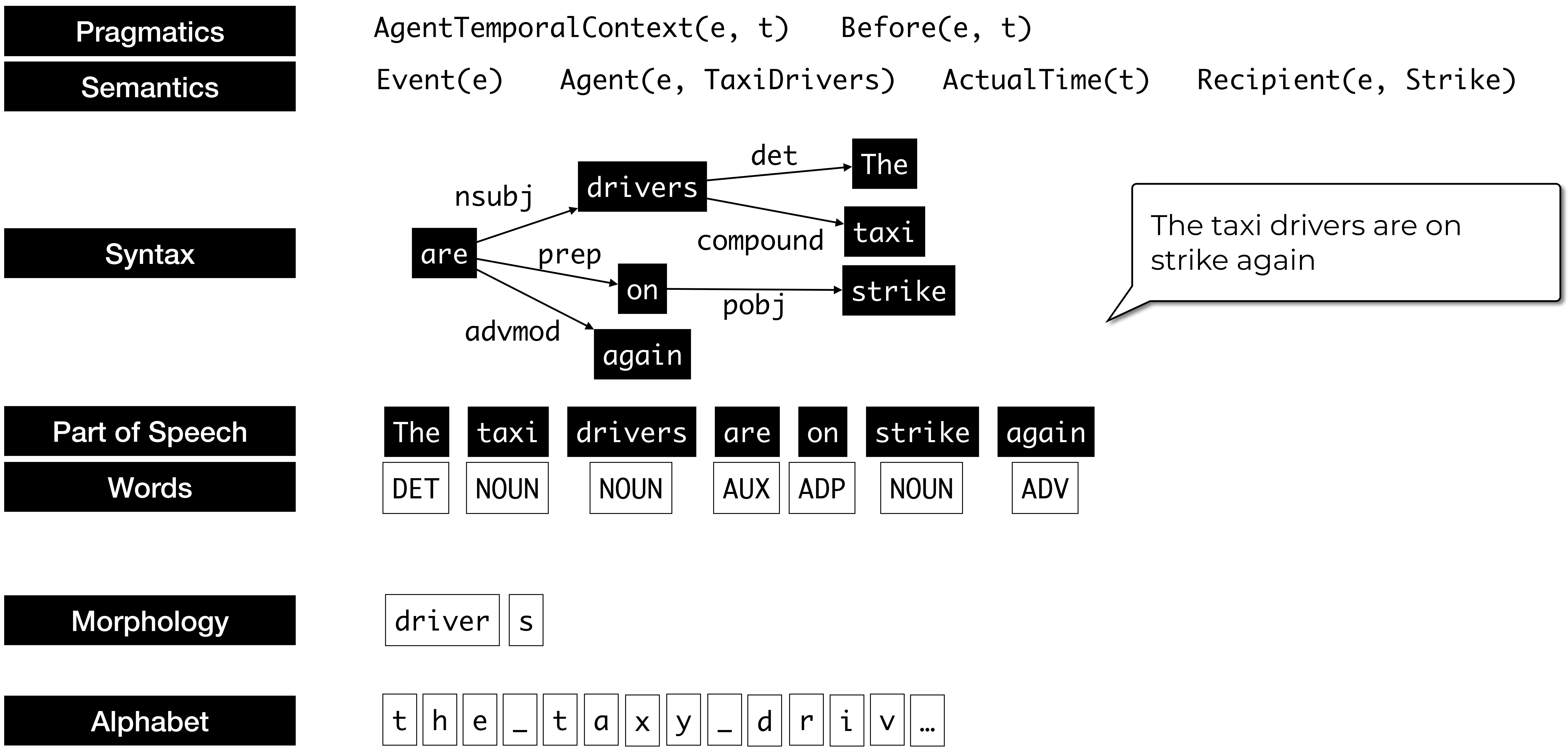
- Leonard: "Hey, Penny. How's work?"
  Penny: "Great! I hope I'm a waitress at the Cheesecake Factory for my whole life!"
  Sheldon: "Was that sarcasm?"
  Penny: "No."
  Sheldon: "Was that sarcasm?"
  Penny: “Yes."

- Finding a good man is like finding a needle in a haystack



Li, Y., Su, H., Shen, X., Li, W., Cao, Z., & Niu, S. (2017). Dailydialog: A manually labelled multi-turn dialogue dataset. *arXiv preprint arXiv:1710.03957.*

Language has many possible levels of interpretation

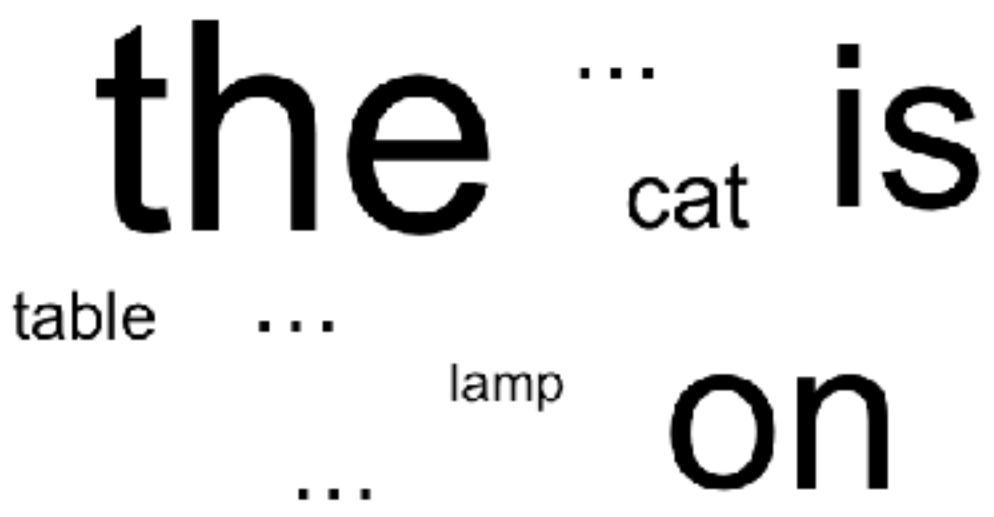


Paradigm shift: human language as a prediction problem

Instead of thinking the language as a set of predefined expressions, we can think it as a generative game

Ingredient 1: **Vocabulary**

- It's a collection of words we can use
- The biggest they are the most frequently they are used in language



Tech

We can associate each word *w* with a probability in the dictionary, based on their frequency in a corpus of texts, by dividing the frequency of w by the sum of the frequencies of all the words in the corpus

Formally

$$P(w) = \frac{count(w)}{\sum_{i=0}^{n} count(w_i)}$$

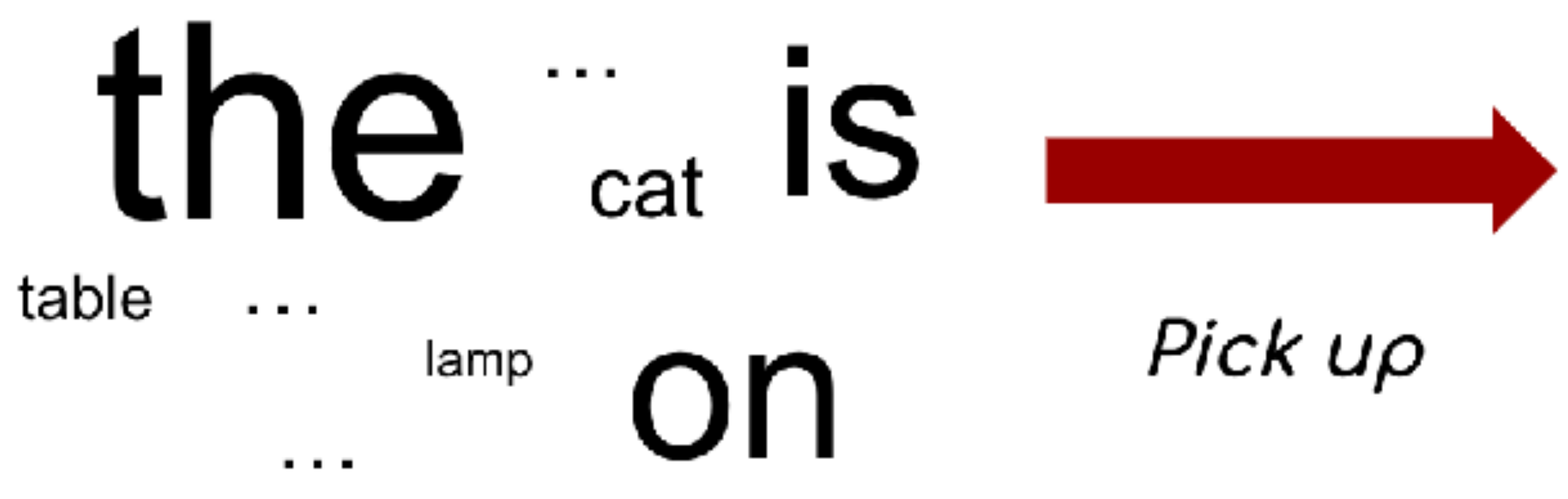
Paradigm shift: human language as a prediction problem

Instead of thinking the language as a set of predefined expressions, we can think it as a generative game

Ingredient 2: **Pick-up procedure**

- We select a word in the vocabulary according to their distribution of probability
- In other terms, we pick up more frequently the most probable words and less frequently the less probable ones
- This can be done one step at a time, thus generating a text word-by-word

Tech
Since words are chosen independently from the others, the joint probability P("the is the table … on") is just the product of the probabilities we used to select the single words



Step 1 the

Step 2 is

Step 3 the

Step 4 table

… …

Step n on

Formally

$$P(w_1, w_2, \ldots, w_n) = P(w_1)P(w_2)\ldots P(w_n)$$

Paradigm shift: human language as a prediction problem

Now, the text "the is the table … on" is not a good emulation of a human real sentence

One of the reasons is that **humans do not choose words to use in a sentence independently from the words they are chosen previously in the same sentence**

Ingredient 3: **Conditional probability**

- During the generation game, the probability of the words in the vocabulary changes at each step depending on the words that have been chosen previously

Tech

We can still use a corpus this way: the probability of a word w to be the next word after a sequence w1 .. wn is given by the number of times we observe w after w1 .. wn divided by the number of times we observe any word after w1 .. wn

Intuition

Make a bet on the next word I'm going to say if I start my sentence by saying:

The President is called Barack [..?..]

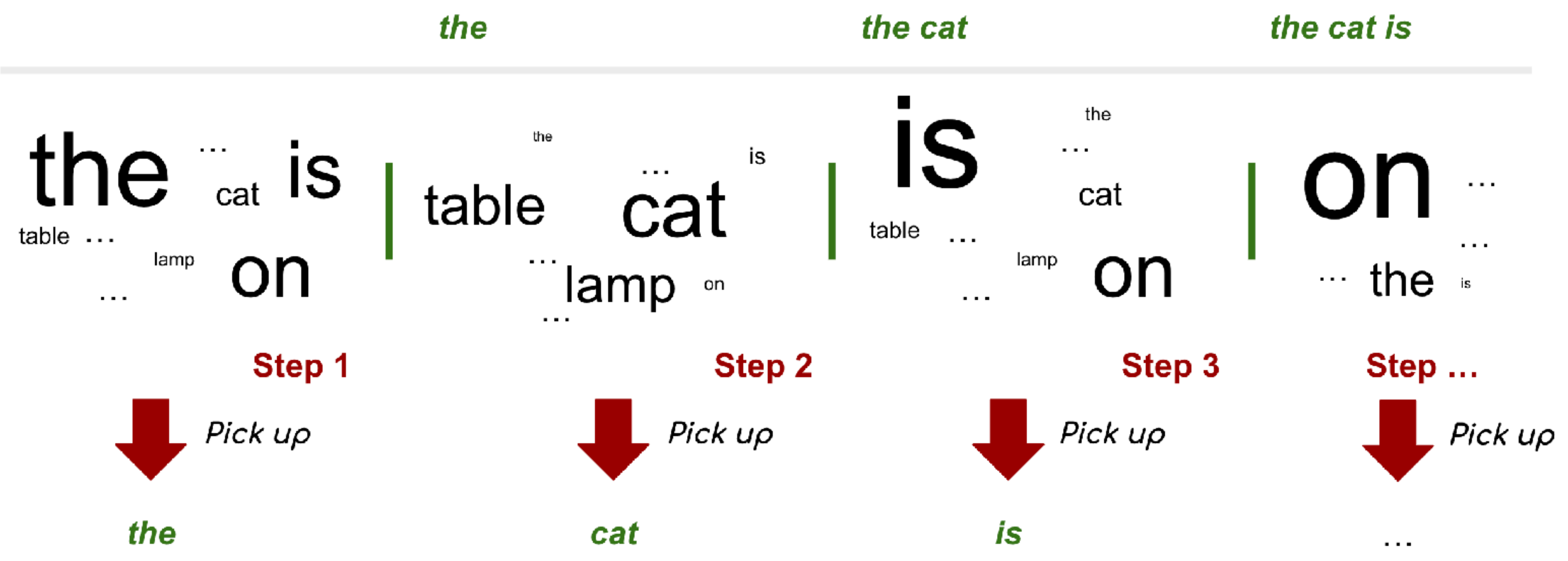
Make a bet of which of the two is my next word when I say:

Good [the] vs **Good [morning]**

How can we use this behaviour?

$$P(w \mid w_1 \dots w_n) = \frac{count(w, w_1, \dots, w_n)}{count(w_1, \dots, w_n)}$$

Paradigm shift: human language as a prediction problem



Tech

The joint probability of the whole sentence now can be split in a chain of conditional probabilities, which is the probability of words at a given step depending on the results of the previous steps:

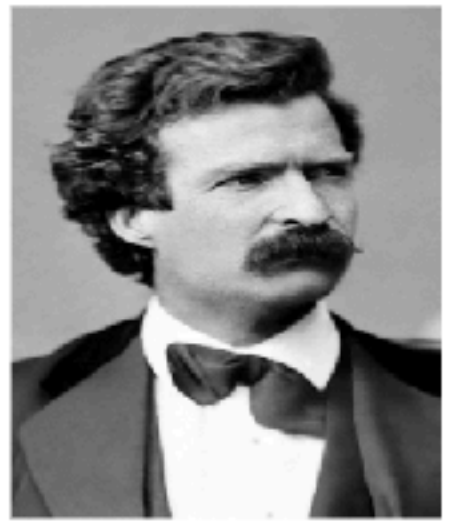
$$P(w_1, \ldots, w_n) = P(w_1)P(w_2 \mid w_1), \ldots, P(w_n \mid w_1, \ldots, w_{n-1})$$

Language models

These types of machines that are able to estimate the probability of the next word in a sequence are called **language models**

Interesting properties:

- They are **generative models**, that is we can use them to generate a new text
- All the **probabilities are estimated starting from a corpus of texts**, meaning that we can build different models from different corpora (e.g., we can emulate different languages, different authors, different text styles, and so on)
- We can also use them to compute the **probability of a text given a model**, which means that, if we have different models from different corpora, we can estimate the probability of a text to be in a given language, written by a given author, be compliant with a given style, and so on)



Mark Twain corpus

Mark Twain LM
MLM

James Joyce corpus

James Joyce LM
JML

T = *"It is better to keep your mouth closed and let people think you are a fool than to open it and remove all doubt"*

MLM(T) > JML(T)

The quote is from Mark Twain

Markov assumption

The idea of language models is pretty good, but there is a **big issue**:

- Everything is based on the idea that at **step n** we need to estimate the probability distribution of words based on the **previous sequence w 1, w 2, ..., w n-1**
- To do so, we need to count how many times in a corpus we observe **w 1, w 2, ..., w n-1** and **w 1, w 2, ..., w n**

It is better to keep your mouth closed and let people think you are a fool than to open it and remove all | doubt

There are two issues:

- Keep track of all the sequences and sub sequences in large corpora is unfeasible
- Even in very large corpora, observing a long sequence like "It is better to keep your mouth closed and let people think you are a fool than to open it and remove all" a sufficient number of times to have a reliable estimation is almost impossible

Markov assumption

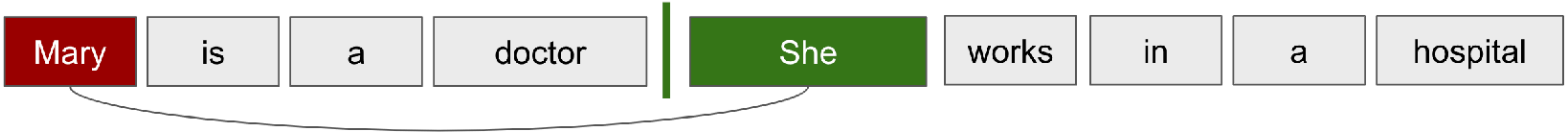
A possible workaround, is to assume that language generation can be modeled as a **Markov process**, where the probability of a word does not depend on the whole previous sequence but only on a (small) previous subsequence

P(the cat is on the table) = P(the) x P(cat | the) x P(is | the, cat) x **P(on | cat, is)** x **P(the | is, on)** x **P(table | on, the)**

here the model *forgets* what was before in the sentence and only relies on the **last two words**

This assumption is effective in many cases ...

... but not when we deal with **long term dependencies**



Chain rule and Markov chain models

$$P(w_1 w_2 w_3 \ldots w_m) = \prod_i^m P(w_i \mid w_1, w_2, \ldots, w_{i-1})$$

$$P(\textbf{the taxi drivers are}\ldots) = P(\textbf{the}) \times P(\textbf{taxi} \mid \textbf{the}) \times P(\textbf{drivers} \mid \textbf{the}, \textbf{taxi}) \times P(\textbf{are} \mid \textbf{the}, \textbf{taxi}, \textbf{drivers}) \times \ldots$$

Markov process: words are generated one at a time until the end of the sentence is generated

$$P(w_1 w_2 w_3 \ldots w_m) = \prod_i^m P(w_i \mid w_{i-n}, \ldots, w_{i-2}, w_{i-1})$$

N-order Markov assumption: we assume that a word depends only on the previous *n* words

$$\textit{with } n = 2 : P(w_1 w_2 w_3 \ldots w_m) = \prod_i^m P(w_i \mid w_{i-2}, w_{i-1})$$

$$P(\textbf{the taxi drivers are}\ldots) = P(\textbf{the}) \times P(\textbf{taxi} \mid \textbf{the}) \times P(\textbf{drivers} \mid \textbf{taxi}) \times P(\textbf{are} \mid \textbf{drivers}) \times \ldots$$

Intuitively, we want to measure how **surprised** the model is to observe an event, given its probability. The surprise is inverse to the probability of the event.

$$S(x) = \log\left(\frac{1}{p(x)}\right) = -\log(p(x))$$

Surprise is a measure of how unlikely a single outcome of a possible event is. **Entropy** generalizes surprise as the expected value of the surprise across every possible outcome, that is the sum of the surprise of every outcome multiplied by the probability it happens

$$H(e) = -\sum_i p(e)_i \log(p(e)_i)$$

The **perplexity** is then defined as the exponential of the entropy

$$PP(e) = 2^{H(e)}$$

We can use this idea for evaluating a test set and comparing the words there with the probabilities estimated by the model, to see how much *surprised* (how much perplexity) the model gets observing unseen data

Perplexity

Perplexity is a form of intrinsic evaluation for a model. In particular, we aim at evaluating the model performance independent of the specific tasks its executing.
Perplexity measures how uncertain a model is about the predictions it makes. Low perplexity means only that a model is **confident**, **not accurate**.

Text as a sequence

Textual data may be modeled as a sequence in many ways

Sequence of characters P → e → r → s → o → n

Sequence of words a → person → in → a → blue → shirt

Sequence of syntax tags DT → NN → IN → DT → JJ → NN

Sequences in text are informative

Sequential information is not informative. The events that compose the sequence are independent one from the other.

a → person → in → a → blue → shirt

Sequential information is informative. The order of words depends on the previous words.

Sequences in text are informative

Learning a sequence means that we can predict the next element of the sequence exploiting the sequence order assuming to keep a memory of the sequence elements

	g	g a	g a m
r	0.28	0.07	0
o	0.20	0	0
e	0.17	0	0.75
a	0.10	0	0
l	0.10	0.19	0
m	0	0.15	0
#END	0	0	0.25

#START →

a	0.59
two	0.11
the	0.04
an	0.03
three	0.03
people	0.02

#START a person in →

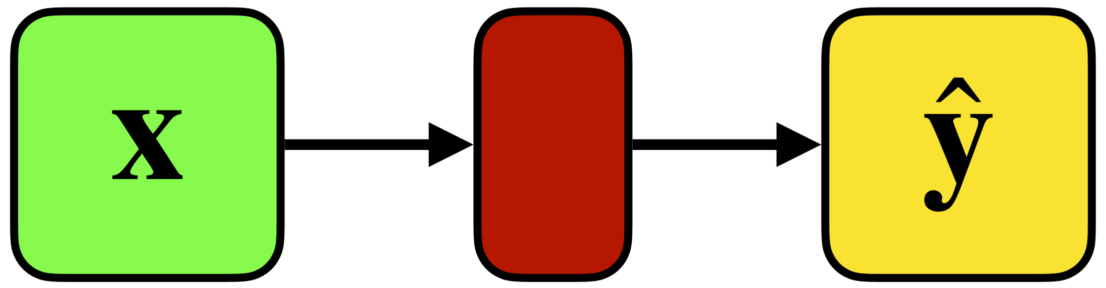
a	0.70
blue	0.03
black	0.03
an	0.03
red	0.02
the	0.02

#START a person in a blue →

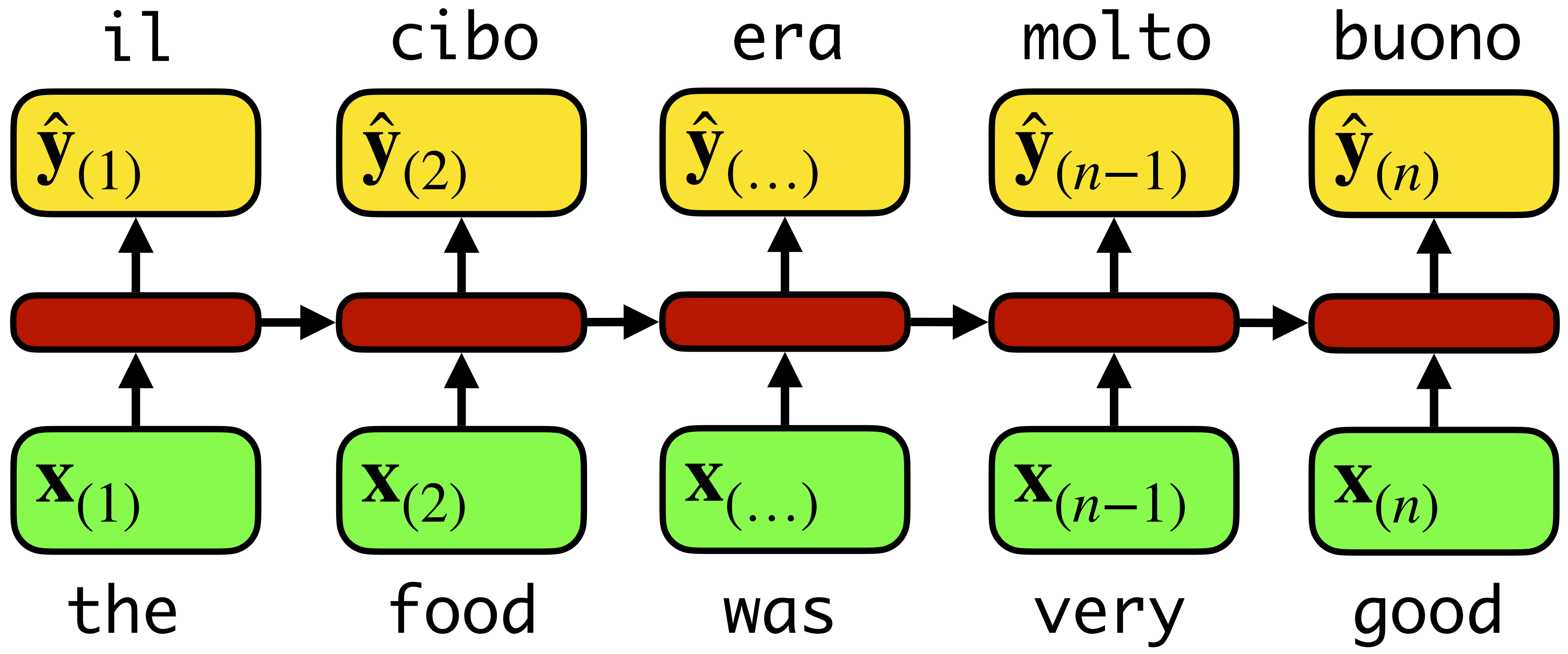
shirt	0.37
jacket	0.20
suit	0.08
hat	0.07
kayak	0.03
outfit	0.03

Applications of sequence learning

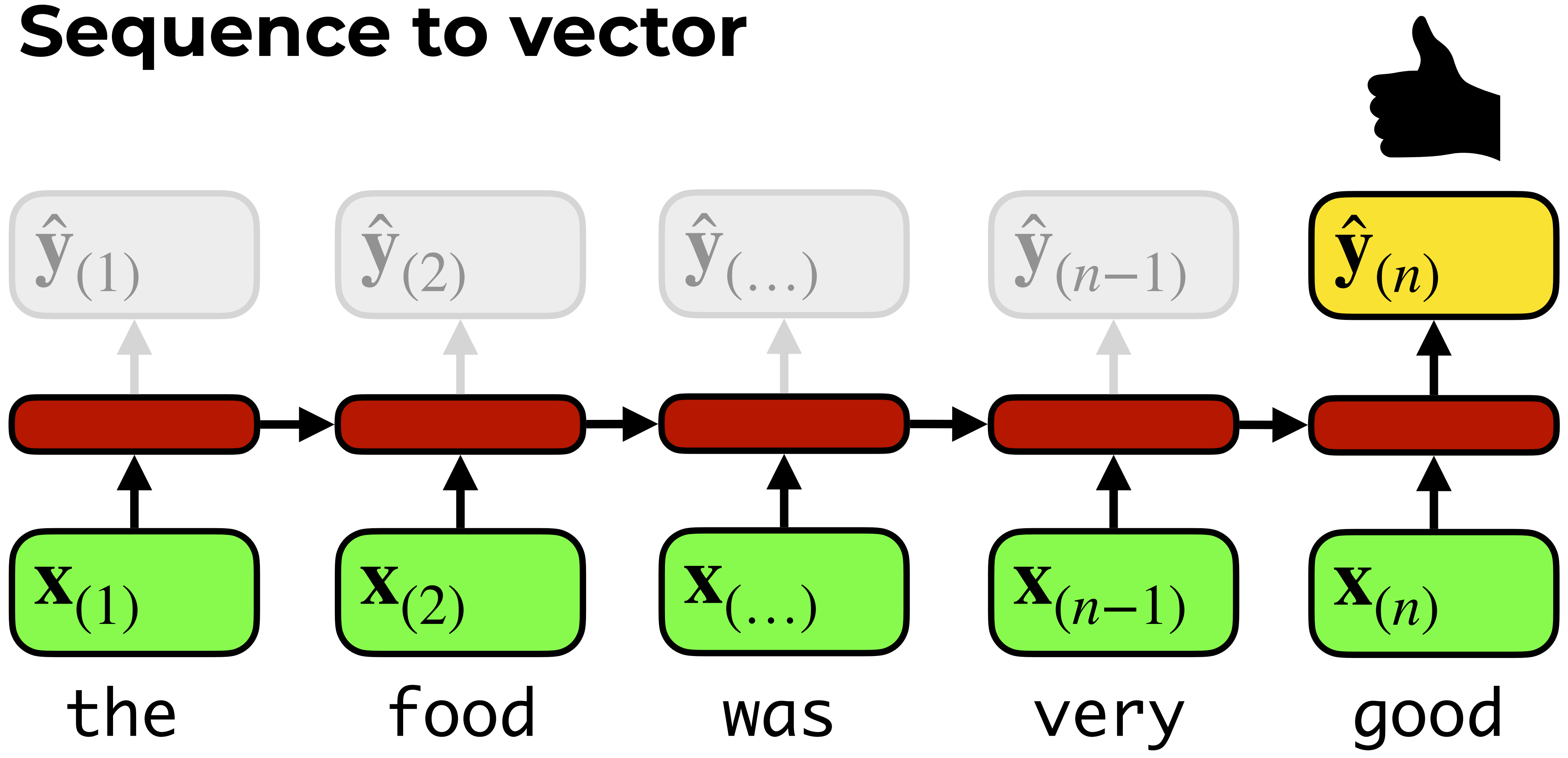
Using the notion of linear transformation as a building block, we can use sequence learning for several different tasks



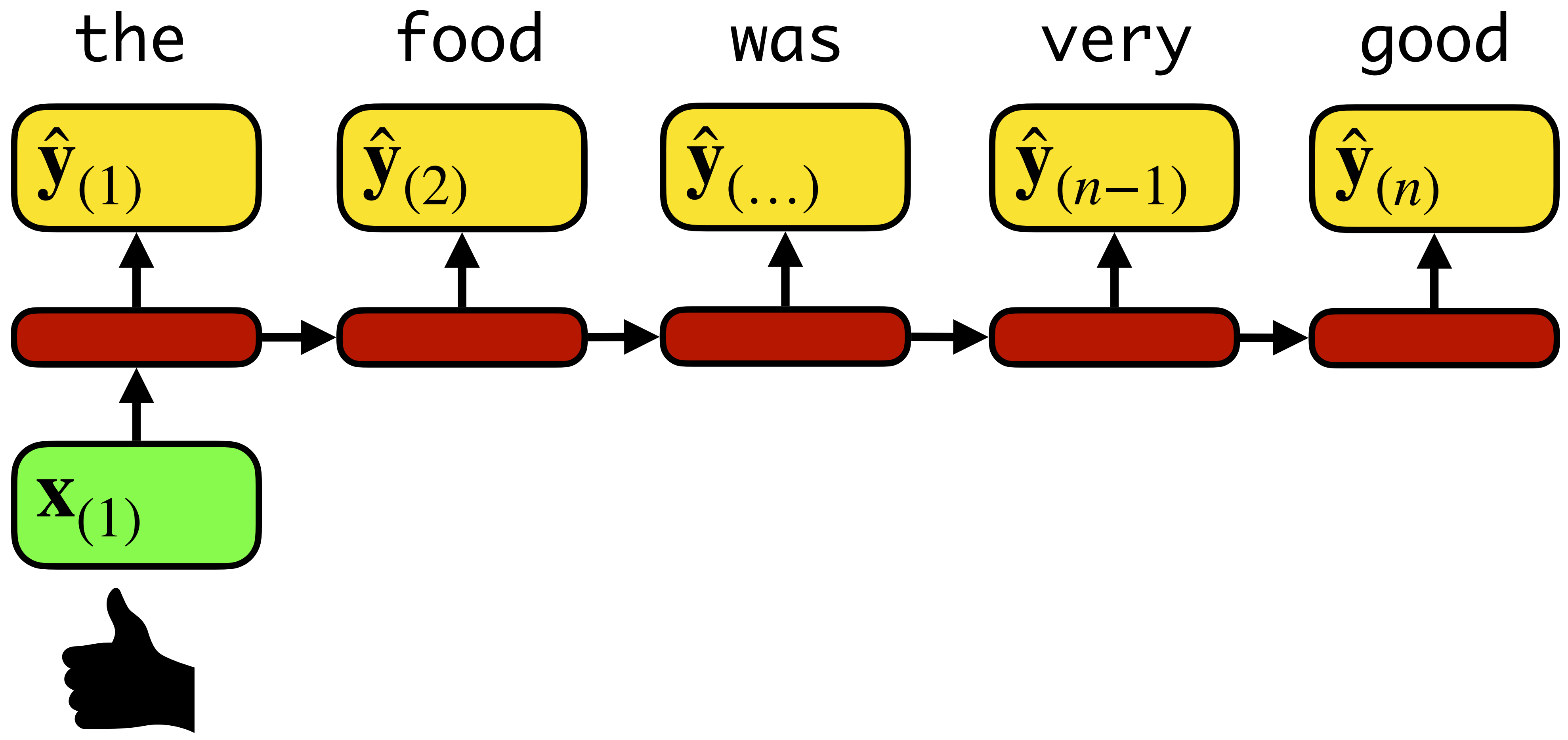
Sequence to sequence



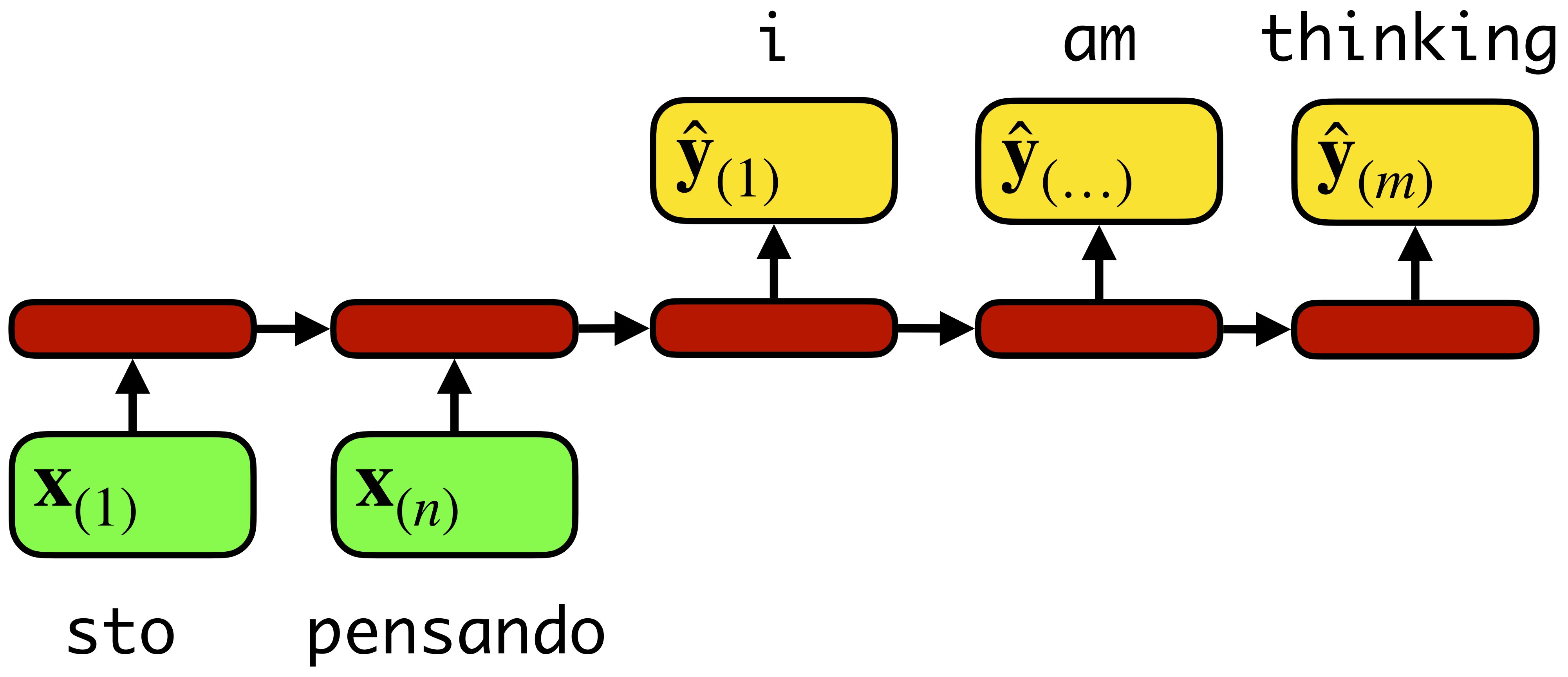
Sequence to vector



Vector to sequence



Encoder-decoder



To deal with text sequences we need to change the learning model

Bag of words learning **is not** sequence learning

```
The food was good,
the service was bad
```

Encoding

In the BOW encoding step, we lose the information provided by the word order in the text

```
The food was bad,
the service was good
```