CS447: Natural Language Processing

http://courses.engr.illinois.edu/cs447

Lecture 29: Envoy

Julia Hockenmaier

juliahmr@illinois.edu

3324 Siebel Center


Envoy

What have you learned in this class?

- What is NLP?

  The core **tasks** (as well as **data sets and evaluation metrics**) that people work on in NLP

- How does NLP work?

  The fundamental **models, algorithms and representations** that have been developed for these tasks

- Why is NLP hard?

  The relevant **linguistic concepts and phenomena** that have to be handled to do well at these tasks

The focus of this class

We want to identify the structure and meaning of words, sentences, texts and conversations

N.B.: we do not deal with speech (no signal processing)

We mainly deal with language analysis/understanding, and less with language generation/production

We focus on fundamental concepts, methods, models, and algorithms, not so much on current research:

Data (natural language): linguistic concepts and phenomena
Representations: grammars, automata, etc.
Neural and statistical models over these representations
Learning & inference algorithms for these models

Why is natural language hard?

Ambiguity:

What does *"I made her duck"* mean?

Ambiguity exists at all levels (lexical, syntactic, referential,…)

Coverage:

What does *"I made her duck cassoulet"* mean?

Zipf's Law: there is a long tail of rare/unknown words

NLP still mostly relies on **supervised learning**,
(which requires large **annotated datasets**)
and/or **pre-training on raw text**
(which requires vast amounts of data)

Why is NLP hard?

NLP still mostly relies on **supervised learning**,
(which requires large **annotated datasets**)
and/or **pre-training on raw text**
(which requires vast amounts of data)

Supervised learning doesn't generalize well
to out-of-domain data

- In NLP: out-of-domain = different genres (news vs. blogs, social media, scientific papers, written vs spoken language), registers (formal vs. colloquial), dialects (British vs. US, AAE, Indian English), or time periods.

NLP annotations have to be designed

- There are many design choices to be made: what do you want to represent, how fine-grained should the labels be?
- We don't know a priori what style of annotation is most useful, or easiest to predict

What is language understanding?

The ability to…

… answer questions about a text

Example: Question answering

... draw (logical/commonsense) inferences:

Example: Entailment recognition

... connect language to the world:

Example: Image Description

... communicate with others to perform a task

Example: Grounded dialogue

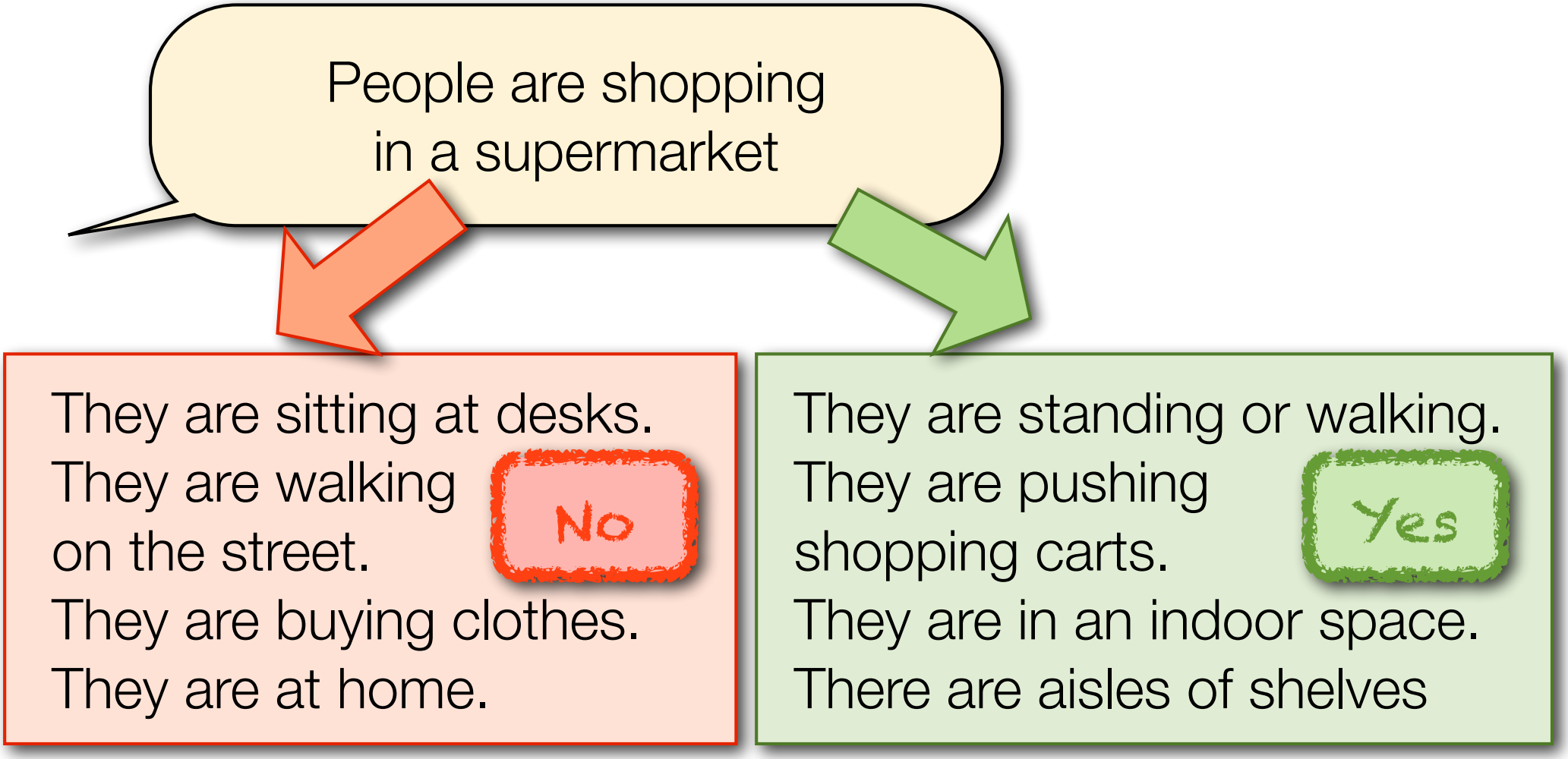
for instruction giving and following

Language understanding as the ability to answer questions about text

More than a decade ago, Carl Lewis stood on the threshold of what was to become the greatest athletics career in history. He had just broken two of the legendary Jesse Owens' college records, but never believed he would become a corporate icon, the focus of hundreds of millions of dollars in advertising. His sport was still nominally amateur. Eighteen Olympic and World Championship gold medals and 21 world records later, Lewis has become the richest man in the history of track and field – a multi-millionaire.

Who is Carl Lewis?
Did Carl Lewis break any world records?
(and how do you know that?)
Is Carl Lewis wealthy? What about Jesse Owens?

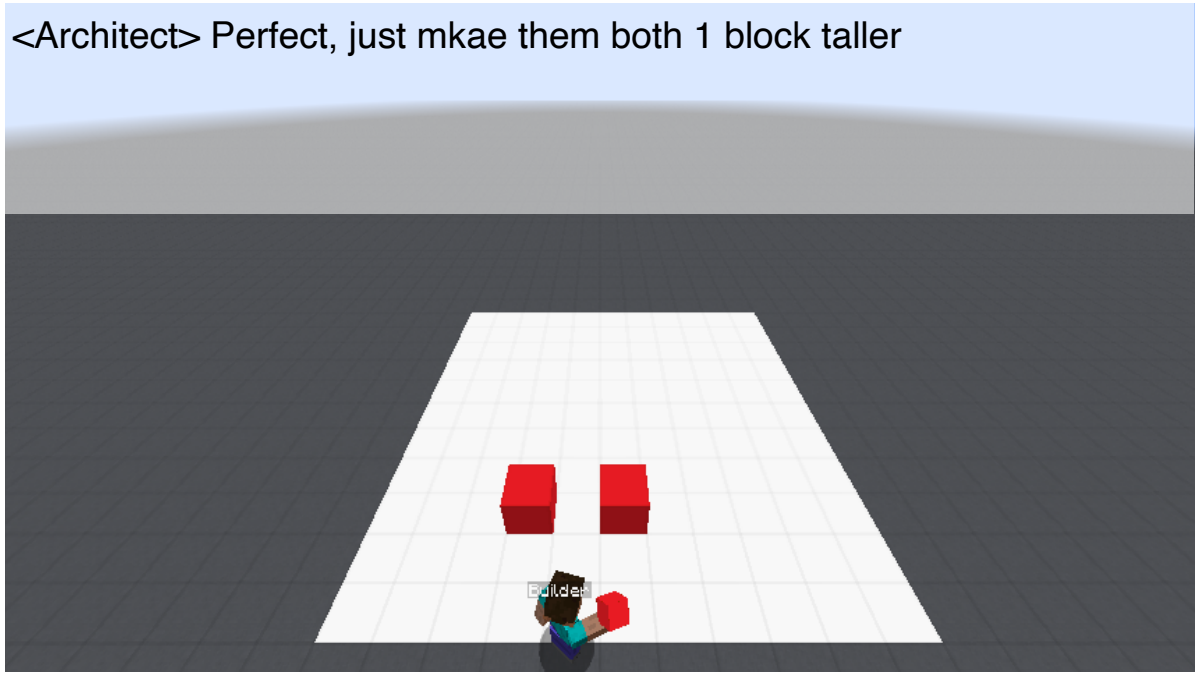
Language understanding as the ability to draw inferences



Language understanding as the ability to describe the world

People are shopping in a supermarket

Language understanding as the ability to give or follow instructions



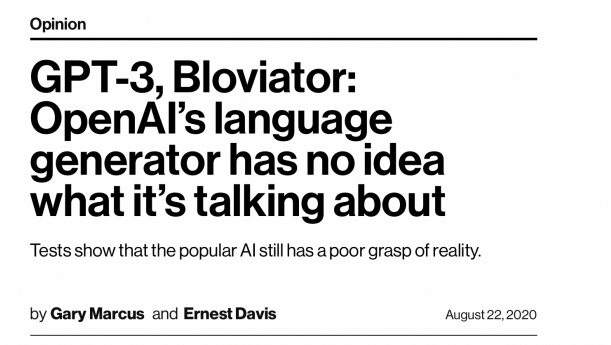
Huge language models solve NLP?

A language model can be used to generate (produce) text

Massive neural language models trained on vast amounts of text have been developed in the last few years

Most recent incarnation: GPT-3 (175B parameters, trained on 300B tokens)

But these models have no access to meaning.
See also Bender & Koller '20 for a critique https://www.aclweb.org/anthology/2020.acl-main.463.pdf



Opinion

GPT-3, Bloviator: OpenAI's language generator has no idea what it's talking about

Tests show that the popular AI still has a poor grasp of reality.

by **Gary Marcus** and **Ernest Davis** August 22, 2020

https://www.technologyreview.com/2020/08/22/1007539/gpt3-openai-language-generator-artificial-intelligence-ai-opinion/

Human Prompt (given to GPT-3)

At the party, I poured myself a glass of lemonade, but it turned out to be too sour, so I added a little sugar.
I didn't see a spoon handy, so I stirred it with a cigarette. But that turned out to be a bad idea because

[GPT-3's generated continuation]

it kept falling on the floor. That's when he decided to start the Cremation Association of North America, which has become a major cremation provider with 145 locations.

from Marcus & Davis '20
https://www.technologyreview.com/2020/08/22/1007539/gpt3-openai-language-generator-artificial-intelligence-ai-opinion/

What are the “next big things”?

Core NLP is still not “solved”

- What is the role of linguistics and/or linguistically inspired annotations?
- A lot of our current techniques (BERT etc.) only work when vast amounts of text and computing resources are available (not true outside of a few big tech companies, or for most languages of the world)
- Situated language understanding/generation are still very difficult
- Our ability to automatically draw inferences from text is still limited

NLP works well enough to have real-world uses

- But applying NLP techniques in any new domain typically requires some amount of manually annotated training and test data (which requires both domain and NLP expertise)
- We need to grapple with ethical implications


Thank you all!
Congrats for making
it through this very
unusual semester!

The remainder of today's slides: What you have learned!

Addendum: NLP toolkits

https://opennlp.apache.org

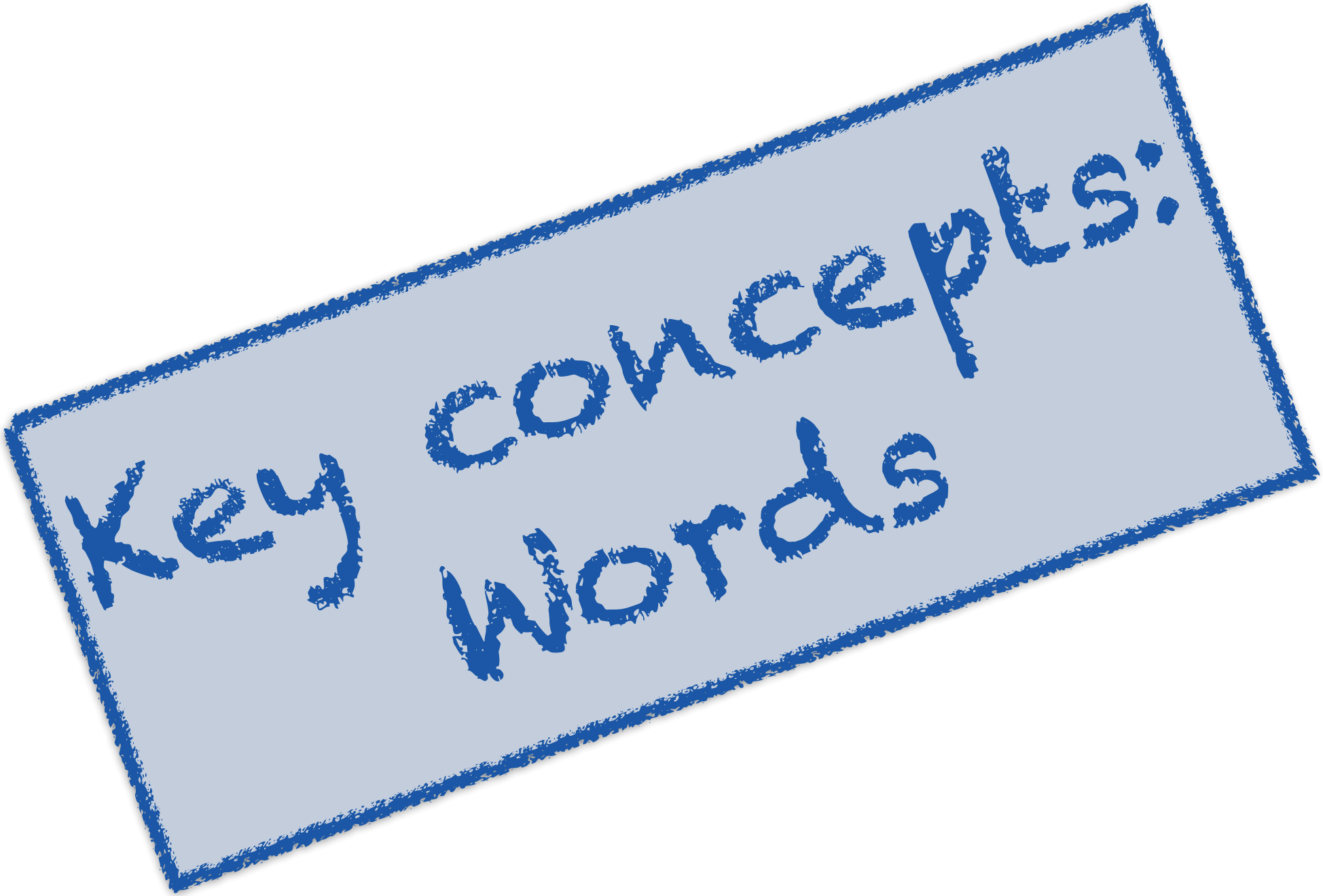
https://spacy.io

https://www.nltk.org

https://allennlp.org

https://stanfordnlp.github.io/CoreNLP/

https://gate.ac.uk


Key concepts:
Words

How many different words are there?

Inflection creates different forms of the same word:

Verbs: to be, being, I am, you are, he is, I was,
Nouns: one book, two books

Derivation creates different words from the same lemma:

grace ⇒ disgrace ⇒ disgraceful ⇒ disgracefully

Compounding combines two words into a new word:

cream ⇒ ice cream ⇒ ice cream cone ⇒ ice cream cone bakery

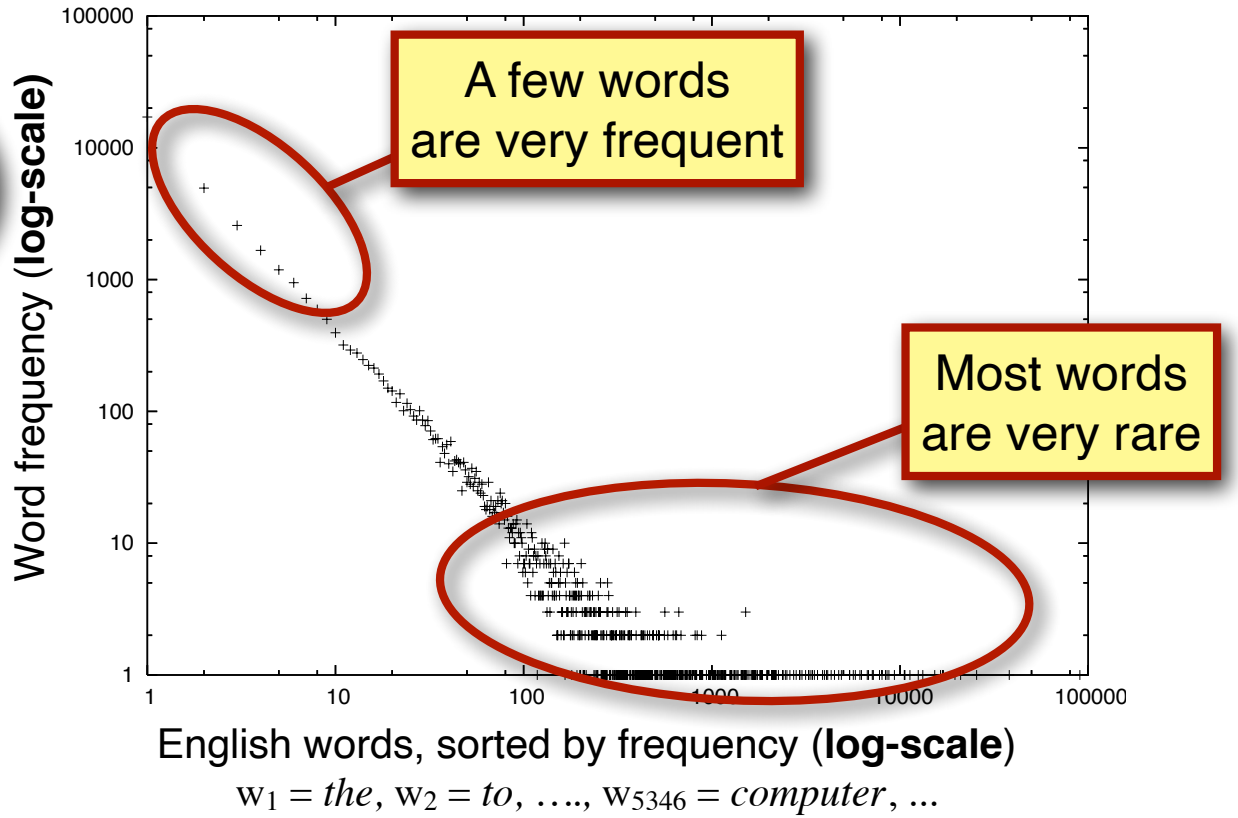
Word formation is productive:

New words are subject to all of these processes:
Google ⇒ Googler, to google, to ungoogle, to misgoogle, googlification, ungooglification, googlified, Google Maps, Google Maps service,...

Zipf's law: the long tail

How many words occur once, twice, 100 times, 1000 times?

the *r*-th most common word w_r has $P(w_r) \propto 1/r$



In natural language:

A small number of events (e.g. words) occur with high frequency

A large number of events occur with very low frequency

Implications of Zipf's Law for NLP

The good:

Any text will contain a number of words that are very **common**.

We have seen these words often enough that we know (almost) everything about them. These words will help us get at the structure (and possibly meaning) of this text.

The bad:

Any text will contain a number of words that are **rare**.

We know *something* about these words, but haven't seen them often enough to know everything about them. They may occur with a meaning or a part of speech we haven't seen before.

The ugly:

Any text will contain a number of words that are **unknown** to us.

We have *never* seen them before, but we still need to get at the structure (and meaning) of these texts.

Dealing with the bad and the ugly

Our systems need to be able to **generalize** from what they have seen to unseen events.

There are two (complementary) approaches to generalization:

— **Linguistics** provides us with insights about the rules and structures in language that we can exploit in the (symbolic) representations we use

E.g.: a finite set of grammar rules is enough to describe an infinite language

— **Machine Learning/Statistics** allows us to learn models (and/or representations) from real data that often work well empirically on unseen data

E.g. most statistical or neural NLP

How do we represent words?

Option 1: Words are **atomic symbols**

- Each (surface) word form is its own symbol
- Add some generalization by mapping different forms of a word to the same symbol
  - **Normalization:** map all variants of the same word (form) to the same canonical variant (e.g. lowercase everything, normalize spellings, perhaps spell-check)
  - **Lemmatization**: map each word to its lemma (esp. in English, the lemma is still a word in the language, but lemmatized text is no longer grammatical)
  - **Stemming**: remove endings that differ among word forms (no guarantee that the resulting symbol is an actual word)

How do we represent words?

Option 2: Represent the **structure** of each word

"books" => `"book N pl"` (or `"book V 3rd sg"`)

This requires a **morphological analyzer** (more later today)

The output is often a lemma (`"book"`)
plus morphological information (`"N pl"` i.e. plural noun)

This is particularly useful for highly inflected languages, e.g. Czech, Finnish, Turkish, etc. (less so for English or Chinese):

In Czech, you might need to know that *nejnezajímavějším*
is a regular, feminine, plural, dative adjective in the superlative.

How do we represent unknown words?

Many NLP systems assume a fixed vocabulary, but still have to handle **out-of-vocabulary (OOV)** words.

Option 1: the **UNK** token

Replace all rare words (with a frequency at or below a given threshold, e.g. 2, 3, or 5) in your training data with an UNK token (UNK = "Unknown word").

Replace *all* unknown words that you come across after training (including rare training words) with the same UNK token

Option 2: **substring-based** representations

[often used in neural models]

Represent (rare and unknown) words ["Champaign"] as sequences of characters ['C', 'h', 'a',…,'g', 'n'] or substrings ["Ch", "amp", "ai", "gn"]

Byte Pair Encoding (BPE): learn which character sequences are common in the vocabulary of your language, and treat those common sequences as atomic units of your vocabulary

What do words **mean**, and how do we **represent** that?

Do we want to represent that…

… “cassoulet” is a French dish?

… “cassoulet” contains meat?

… “cassoulet” is a stew?

What do words mean, and how do we represent that?

Do we want to represent…

… that a “bar” are places to have a drink?

… that a “bar” is a long rods?

… that to “bar” something means to block it?

What does this word mean?

This **plant** needs to be watered each day.

⇒ **living plant**

This **plant** manufactures 1000 widgets each day.

⇒ **factory**

Word Sense Disambiguation (WSD):

Identify the sense of content words (nouns, verbs, adjectives) in context (assuming a fixed inventory of word senses).

Presumes the words to classify have a discrete set of senses.

Different approaches to lexical semantics

Roughly speaking, NLP draws on two different types of approaches to capture the meaning of words:

The lexicographic tradition aims to capture the information represented in lexicons, dictionaries, etc.

The distributional tradition aims to capture the meaning of words based on large amounts of raw text

WordNet

Very large lexical database of English:

110K nouns, 11K verbs, 22K adjectives, 4.5K adverbs

(WordNets for many other languages exist or are under construction)

Word senses grouped into synonym sets ("synsets") linked into a conceptual-semantic hierarchy

81K noun synsets, 13K verb synsets, 19K adj. synsets, 3.5K adv synsets

Avg. # of senses: 1.23 nouns, 2.16 verbs, 1.41 adj, 1.24 adverbs

Conceptual-semantic relations: hypernym/hyponym

also holonym/meronym

Also lexical relations, in particular lemmatization

Available at http://wordnet.princeton.edu

Language understanding requires knowing when words have similar meanings

Question answering:

Q: "How ***tall*** is Mt. Everest?"

Candidate A: "The official ***height*** of Mount Everest is 29029 feet"

"tall" is similar to *"height"*

Plagiarism detection:

MAINFRAMES

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

Consisting of advanced components, mainframes have the capability of running multiple large applications required by many and most enterprises and organizations. This is one of its advantages. Mainframes are also suitable to cater for those applications

MAINFRAMES

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

Due to the advanced components mainframes have, these computers have the capability of running multiple large applications required by most enterprises, which is one of its advantage. Mainframes are also suitable to cater for those applications

The Distributional Hypothesis

Zellig Harris (1954):

"oculist and eye-doctor … occur in almost the same environments"

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

John R. Firth 1957:

You shall know a word by the company it keeps.

The **contexts** in which a word appears
tells us a lot about what it means.

Words that appear in similar contexts have similar meanings

Two ways NLP uses context for semantics

Distributional similarities (vector-space semantics):
Use the set of all contexts in which words
(= word types) appear to measure their similarity

Assumption: Words that appear in similar contexts (*tea, coffee*) have similar meanings.

Word sense disambiguation
Use the context of a *particular occurrence* of a word (token) to identify which sense it has.

Assumption: If a word has multiple distinct senses (e.g. *plant*: *factory* or *green plant*), each sense will appear in different contexts.

How do we represent words to capture word similarities?

As **atomic symbols**?

[e.g. as in a traditional n-gram language model, or when we use them as explicit features in a classifier]

This is equivalent to very high-dimensional **one-hot vectors:**

aardvark=[1,0,...,0], *bear*=[0,1,000],..., *zebra*=[0,...,0,1]

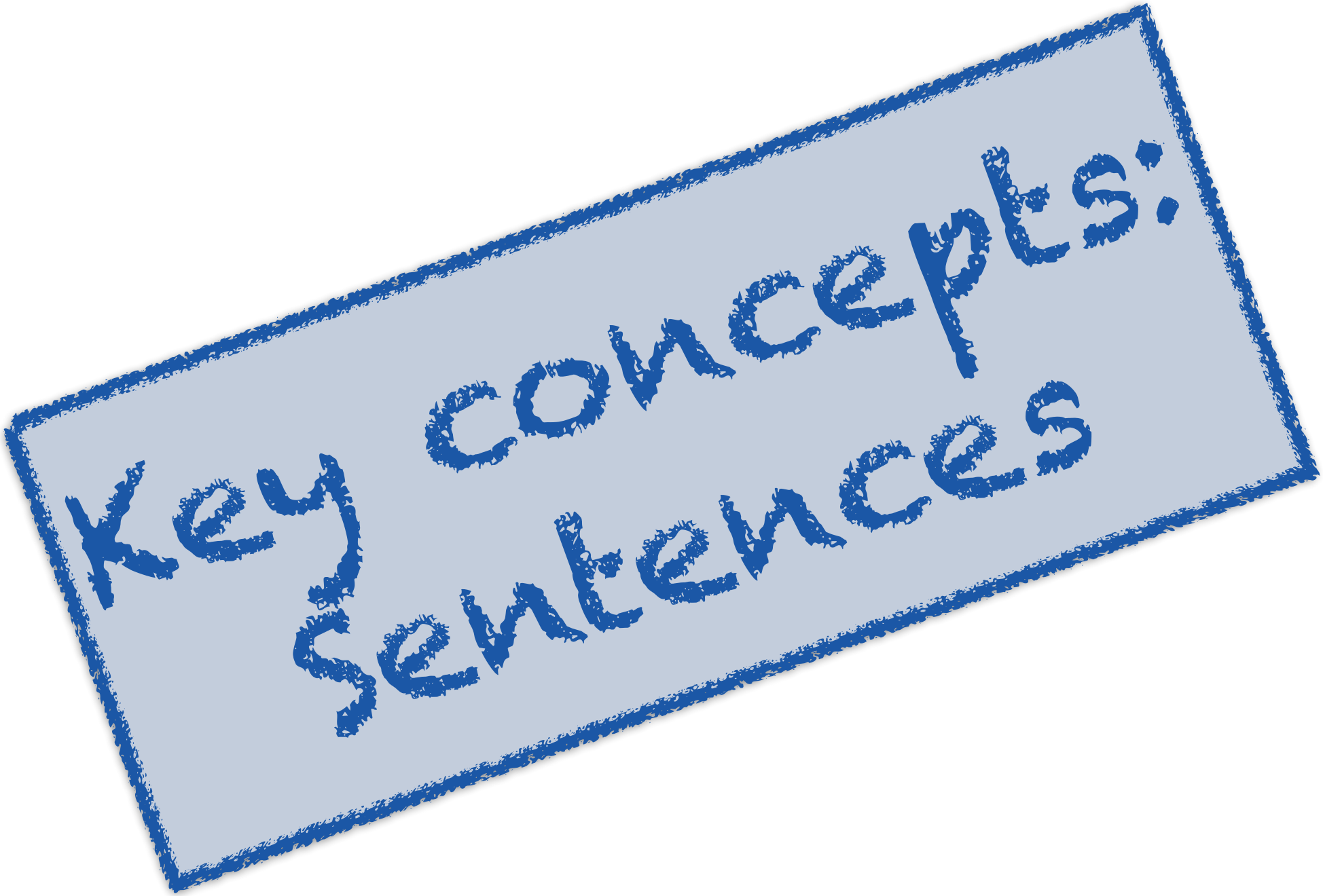
No: *height/tall* are as different as *height/cat*

As very high-dimensional **sparse vectors**?

[to capture so-called distributional similarities]

As lower-dimensional **dense vectors**?

["word embeddings" — important prerequisite for neural NLP]

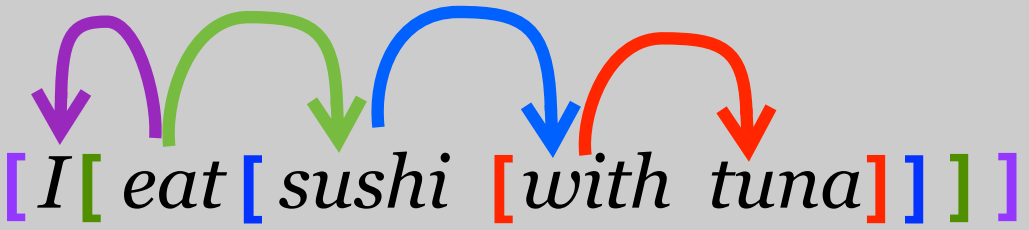

Key concepts:
Sentences

What is the structure of a sentence?

Sentence structure is **hierarchical**:

A sentence consists of **words** (I, eat, sushi, with, tuna)
…which form phrases or **constituents**: “sushi with tuna”

Sentence structure defines **dependencies** between words or phrases:



Constituents: Heads and dependents

There are different kinds of constituents:

Noun phrases: the man, a girl with glasses, Illinois

Prepositional phrases: with glasses, in the garden

Verb phrases: eat sushi, sleep, sleep soundly

NB: this is an oversimplification. Some phrases (**John, Kim and Mary**) have multiple heads, others (I like coffee and [**you tea**]) perhaps don't even have a head

Every phrase has one **head**:

Noun phrases: the man, a girl with glasses, Illinois

Prepositional phrases: with glasses, in the garden

Verb phrases: eat sushi, sleep, sleep soundly

The other parts are its **dependents**.

Dependents are either **arguments** or **adjuncts**

NB: some linguists think the argument-adjunct distinction isn't always clear-cut, and there are some cases that could be treated as either, or something in-between

Arguments are obligatory

Words **subcategorize** for specific sets of arguments:

Transitive verbs (sbj + obj): [John] likes [Mary]
The set/list of arguments is called a **subcat frame**

All **arguments** have to be **present**:

*[John] likes. *likes [Mary].

No argument slot can be **occupied multiple times**:

*[John] [Peter] likes [Ann] [Mary].

Words can have **multiple subcat frames**:

Transitive eat (sbj + obj): [John] eats [sushi].
Intransitive eat (sbj): [John] eats

Adjuncts (modifiers) are optional

Adverbs, PPs and adjectives can be adjuncts

Adverbs: John runs [fast].
a [very] heavy book.

PPs: John runs [in the gym].
the book [on the table]

Adjectives: a [heavy] book

There can be an arbitrary number of adjuncts:

John saw Mary.

John saw Mary [yesterday].

John saw Mary [yesterday] [in town]

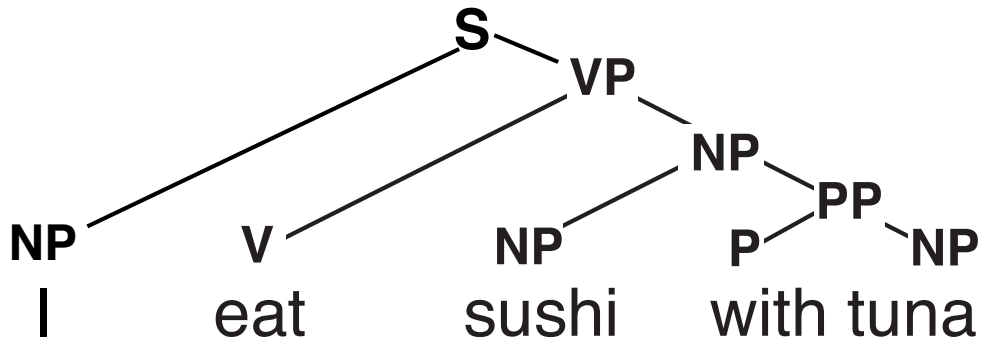
John saw Mary [yesterday] [in town] [during lunch]

[Perhaps] John saw Mary [yesterday] [in town] [during lunch]

Context-free grammars (CFGs) define **phrase structure trees**

```
NP ⟶ I
NP ⟶ sushi
NP ⟶ tuna
NP ⟶ NP PP
P  ⟶ with
PP ⟶ P NP
S  ⟶ NP VP
V  ⟶ eat
VP ⟶ V  NP
```

NP: Noun Phrase
P: Preposition
S: Sentence
PP: Prepositional Phrase
V: Verb
VP: Verb Phrase



Leaf nodes (I, eat, …) correspond to the words in the sentence

Intermediate nodes (NP, VP, PP) span substrings (= the *yield* of the node), and correspond to nonterminal *constituents*

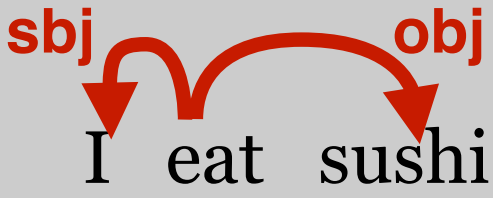
The root spans the entire sentence and is labeled with the start symbol of the grammar (here, S)

Dependency grammar

DGs describe the structure of sentences as a **directed acyclic graph**.

- The **nodes** of the graph are the **words**
- The **edges** of the graph are the **dependencies**.
- **Edge labels** indicate different **dependency types**.

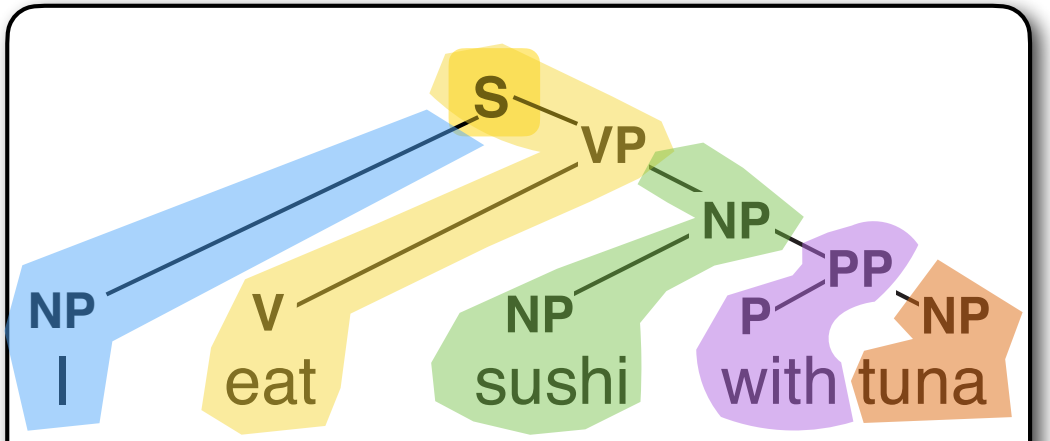
Typically, the graph is assumed to be a **tree**.



Note: the relationship between DG and CFGs:

If a CFG phrase structure tree is translated into DG, the resulting dependency graph has no crossing edges.

From CFGs to dependencies



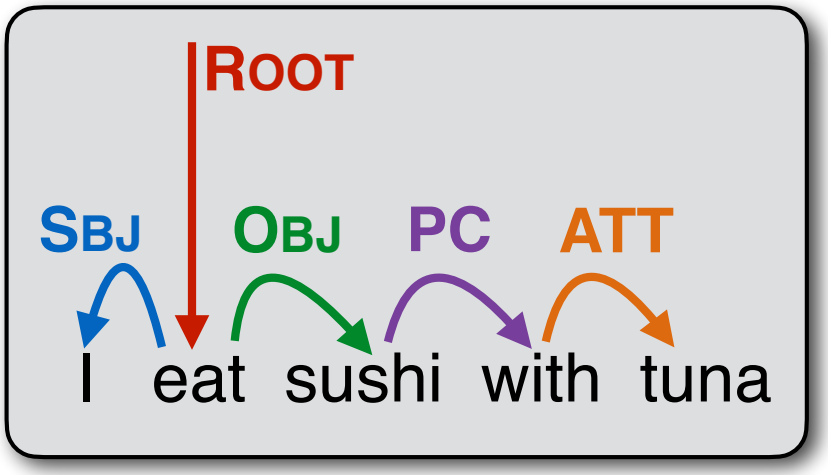
CFG (bold = head child):

S → **NP** **VP**

VP → **V** NP

NP → **NP** PP

PP → **P** NP



Start at the **root** of the tree (S)

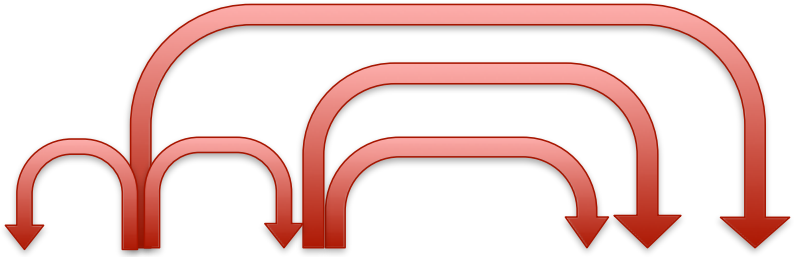
Follow the **head path** ('**spine**' of the tree) to the **head word** of the sentence ('eat').

Add a ROOT dependency to this word.

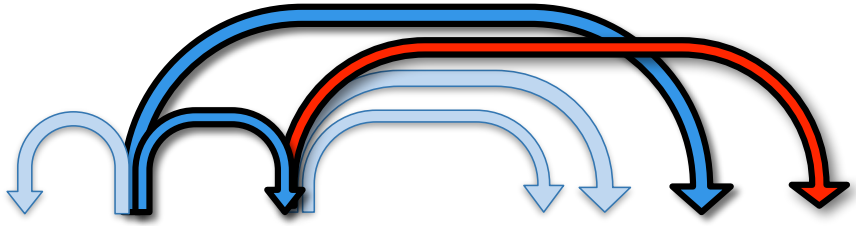
For all other **maximal projections**: follow their head paths to get their head words and add the corresponding dependencies

Dependency structures in general

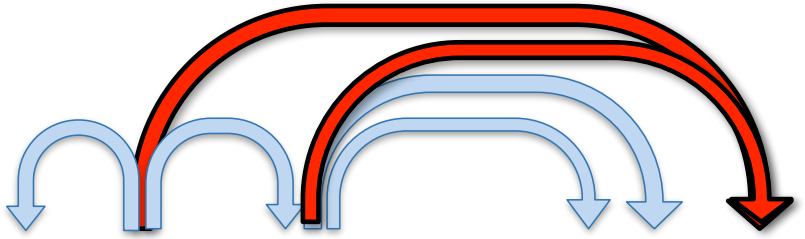
Nested (projective) dependency trees (CFGs)



Non-projective dependency trees



Non-local dependency graphs



Probabilistic Context-Free Grammars

For every nonterminal X, define a probability distribution $P(X \rightarrow \alpha \mid X)$ over all rules with the same LHS symbol X:

S	$\rightarrow$	NP VP	0.8
S	$\rightarrow$	S conj S	0.2
NP	$\rightarrow$	Noun	0.2
NP	$\rightarrow$	Det Noun	0.4
NP	$\rightarrow$	NP PP	0.2
NP	$\rightarrow$	NP conj NP	0.2
VP	$\rightarrow$	Verb	0.4
VP	$\rightarrow$	Verb NP	0.3
VP	$\rightarrow$	Verb NP NP	0.1
VP	$\rightarrow$	VP PP	0.2
PP	$\rightarrow$	P NP	1.0

CKY chart parsing algorithm

Bottom-up parsing:

start with the words

Dynamic programming:

save the results in a table/chart

re-use these results in finding larger constituents

Complexity: $O(n^3|G|)$

n: length of string, $|G|$: size of grammar)

Presumes a CFG in **Chomsky Normal Form**:

Rules are all either **A → B C** (RHS = two nonterminals)
or **A → a** (RHS = a single terminal)
(with **A, B, C** nonterminals and **a** a terminal)

Parsing algorithms for DG

'Transition-based' parsers:

Learn a sequence of actions to parse sentences

Models:

State = stack of partially processed items
+ queue/buffer of remaining tokens
+ set of dependency arcs that have been found already

Transitions (actions) = add dependency arcs; stack/queue operations

'Graph-based' parsers:

Learn a model over dependency graphs

Models:

a function (typically sum) of local attachment scores

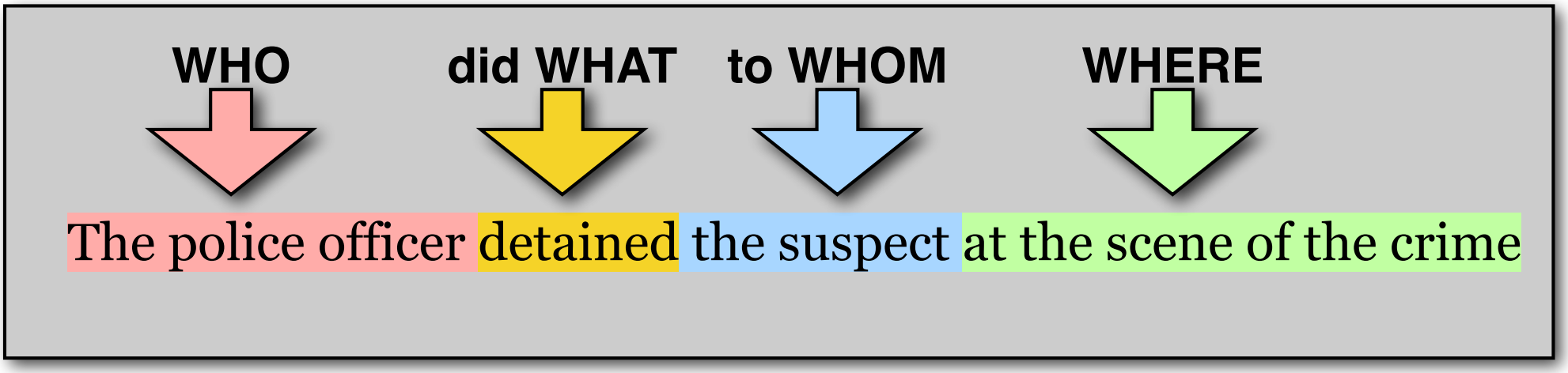
For dependency trees, you can use a minimum spanning tree algorithm

Predicate-argument structure

Understanding a sentence = knowing who did what (to whom, when, where, why…)

Verbs corresponds to predicates (what was done)

Their **arguments** (and modifiers) identify who did it, to whom, where, when, why, etc.



What do verbs mean?

Verbs describe **events** or **states** ('eventualities'):

Tom broke the window with a rock.

The window broke.

The window was broken by Tom/by a rock.

If we *naively* translate verbs to **(logical) predicates…**

(subject = first argument, object = second argument, etc.)

```
break(Tom, window, rock)
break(window)
break(window, Tom)
break(window, rock)
```

… we don't really capture that these sentences describe the same event.

There are many different ways to describe the same event

Grammatical roles ≠ Semantic roles

Tom **broke the window** with a rock.

The window broke.

The window was broken by Tom/by a rock.

Related verbs/nouns can describe the **same event**:

XYZ corporation **bought** the stock.

They **sold** the stock to XYZ corporation.

The stock was **bought** by XYZ corporation.

The **purchase** of the stock by XYZ corporation…

The stock **purchase** by XYZ corporation…

Can we map sentences describing the same event to **the same representation?**

Semantic/Thematic roles

Verbs describe **events** or **states** ('eventualities'):

Tom broke the window with a rock.

The window broke.

The window was broken by Tom/by a rock.

Thematic roles refer to **participants** of these events:

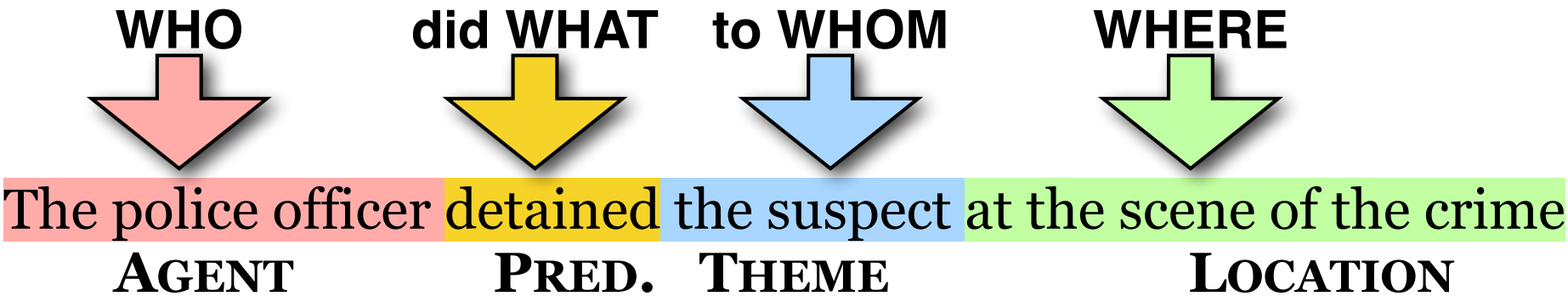
Agent (who performed the action): Tom

Patient (who was the action performed on): window

Tool/Instrument (what was used to perform the action): rock

Semantic/thematic roles (agent, patient) are different from **grammatical roles** (subject or object).

Semantic Role Labeling (SRL)



The task of identifying…

— all **predicates** in a sentence

— the **arguments of each predicate** and their **semantic role**

SRL systems for English are typically trained on **PropBank** or **FrameNet**

FrameNet

Baker et al. 1998, Fillmore et al. 2003, Fillmore and Baker 2009, Ruppenhofer et al. 2006

[You] can't [blame] [the program] [for being unable to identify it]
COGNIZER PRED. EVALUEE REASON

A **FrameNet frame** defines a set of **frame-specific semantic roles** (called **frame elements**), and includes a set of **predicates** (e.g verbs) that take these roles.

It also includes **example sentences** (not shown below)

Frame: `Change-position-on-a-scale`
Predicates: rise, increase,…
Frame Elements: ITEM, ATTRIBUTE, INITIAL VALUE, FINAL VALUE
This frame consists of words that indicate the change of an ITEM's position on a scale (the ATTRIBUTE) from a starting point (INITIAL VALUE) to an end point (FINAL VALUE)

PropBank Frames and Annotations

agree.01 Arg0: Agreer Arg1: Proposition
Arg2: Other entity agreeing

[Arg0 The group] agreed [Arg1 it wouldn't make an offer]
[Arg0 John] agrees with [Arg2 Mary]

fall.01 Arg1: patient/thing falling Arg2: extent/amount fallen
Arg3: start point Arg4: end point

[Arg1 Sales] fell [Arg4 to $251 million]
[Arg1 Junk bonds] fell [Arg2 by 5%]


Key concepts:
Discourse and
Dialogue

How can we understand discourse?

> On Monday, John went to Einstein's. He wanted to buy lunch. But the cafe was closed. That made him angry, so the next day he went to Green Street instead.

Understanding discourse requires (among other things):

1) doing **coreference** resolution:

'the cafe' and '*Einstein's'* **refer to the same entity**

He and *John* refer to the same person.
That refers to *'the cafe was closed'*.

2) identifying **discourse ('coherence') relations**:

'He wanted to buy lunch' is the ***reason*** for
'John went to Bevande.'

The coreference resolution task

Victoria Chen, Chief Financial Officer of Megabucks Banking Corp since 2004, saw her pay jump 20%, to $1.3 million, as the 37-year-old also became the Denver-based financial services company's president. It has been ten years since she came to Megabucks from rival Lotsabucks.

Return **Coreference Chains**
(sets of mentions that refer to the same entities)

1. {Victoria Chen, Chief Financial Officer...since 2004, her, the 37-year-old, the Denver-based financial services company's president}
2. {Megabucks Banking Corp, Denver-based financial services company, Megabucks}
3. {her pay}
4. {rival Lotsabucks}

The importance of world knowledge

Coreference resolution often needs
world ("commonsense") knowledge.

Compare:

The city councilmen refused **the demonstrators** a permit because **they feared violence**.

The city councilmen refused **the demonstrators** a permit because **they advocated violence.**

CF: The Winograd Schema Challenge

https://cs.nyu.edu/faculty/davise/papers/WinogradSchemas/WS.html

World knowledge may capture bias

Preferred attachments (both by humans and systems) often reflect stereotypes (e.g. about occupations and gender)

A man and his son get into a terrible car crash. The father dies, and the boy is badly injured. In the hospital, the surgeon looks at the patient and exclaims, "I can't operate on this boy, he's my son!" https://www.aclweb.org/anthology/N18-2002/

Entity-based coherence

John wanted to buy a piano for his living room.
Jenny also wanted to buy a piano.
He went to the piano store.
It was nearby.
The living room was on the second floor.
She didn't find anything she liked.
The piano he bought was hard to get up to that floor.

This is incoherent because the sentences switch back and forth between **entities** (John, Jenny, the piano, the store, the living room)

Topical coherence

Before winter I **built** a **chimney**, and **shingled** the **sides** of my **house**...
I have thus a tight **shingled** and **plastered house**... with a **garret** and a **closet**, a large **window** on each **side**....

These sentences clearly talk about the same topic: both contain a lot of words having to do with the structures of houses and building (they belong to the same 'semantic field').

When nearby sentences talk about the same topic, they often exhibit **lexical cohesion** (they use the same or semantically related words).

Rhetorical coherence

John took a train from Paris to Istanbul.
He likes spinach.

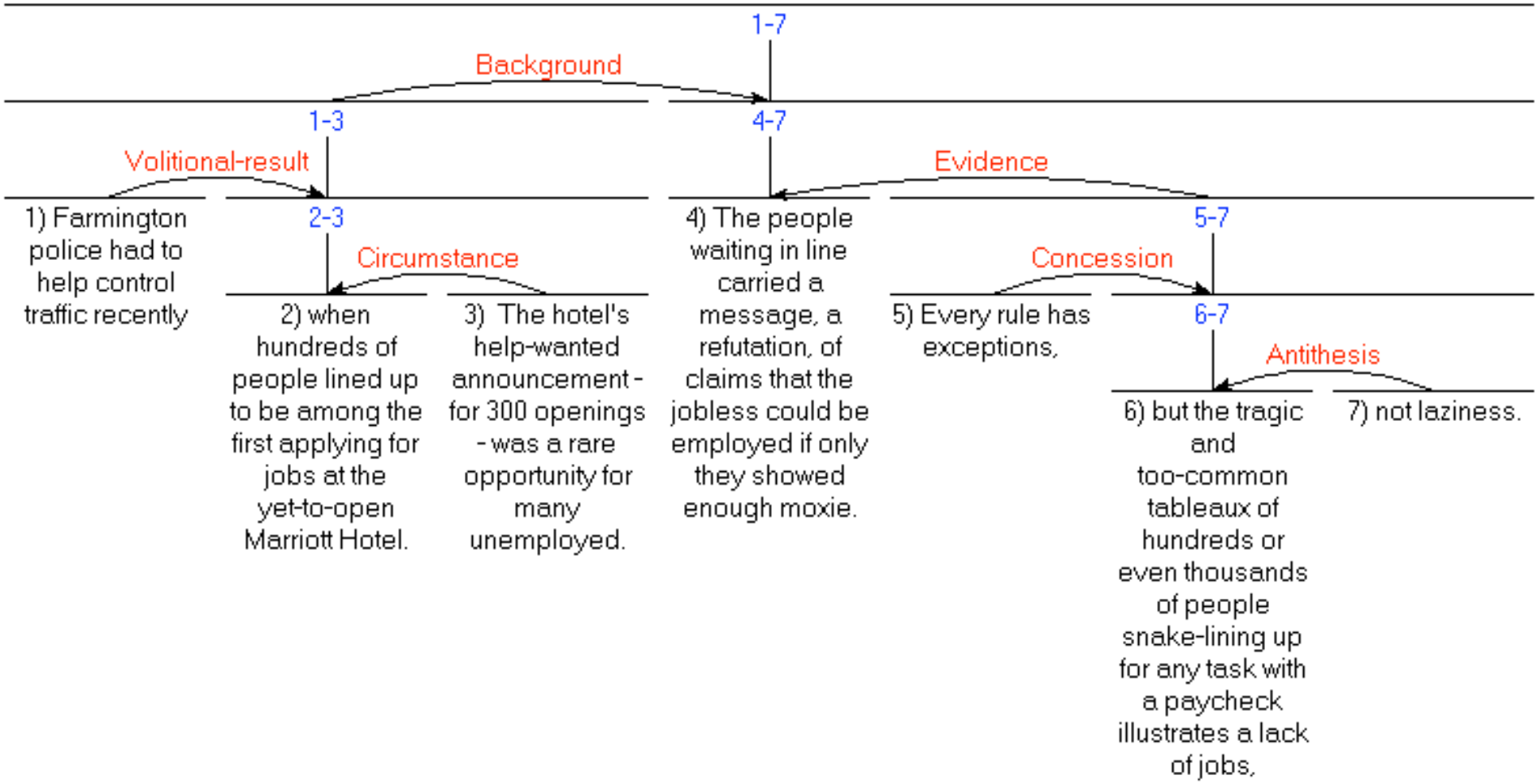
This discourse is incoherent because there is no apparent rhetorical relation between the two sentences.

(Did you try to construct some explanation, perhaps that Istanbul has exceptionally good spinach, making the very long train ride worthwhile?)

Jane took a train from Paris to Istanbul.
She had to attend a conference.

This discourse is coherent because there is clear rhetorical relation between the two sentences.
The second sentence provides a REASON or EXPLANATION for the first.

Discourse structure is hierarchical



RST website: http://www.sfu.ca/rst/

Penn Discourse Treebank (PDTB)

Miltsakaki et al. 2004, Prasad et al. 2008, 2014

The PDTB annotates explicit and implicit discourse connectives and their argument spans.

Explicit connective ("as a result")

[arg1 *Jewelry displays in department stores were often cluttered and uninspired. And the merchandise was, well, fake*].
As a result, [arg2 marketers of faux gems steadily lost space in department stores to more fashionable rivals—cosmetics makers]

Implicit connective (no lexical item)

[arg1 *In July, the Environmental Protection Agency imposed a gradual ban on virtually all uses of asbestos.*]
[arg2 By 1997, almost all remaining uses of cancer-causing asbestos will be outlawed]

From discourse to dialogue

Discourse:
The **speaker** communicates to an **absent, passive listener** (or audience), and attempts to get them to construct a similar model of the state of affairs.

The speaker does not receive any feedback from the audience.

Dialogue:
Both parties are **present** and **active** participants.
They each bring **their own mental model** of the state of affairs.
Communication succeeds if both parties understand each other's mental models (and perhaps even get their models to agree).

Both parties provide feedback to each other.

A dialogue between a customer (C) and a travel agent (A)

C_1: ...I need to travel in May.
A_1: And, what day in May did you want to travel?
C_2: OK uh I need to be there for a meeting that's from the 12th to the 15th.
A_2: And you're flying into what city?
C_3: Seattle.

A **dialogue** is a conversation between two speakers that consists of a sequence of **turns**

Turn = an **utterance** by one of the two speakers

Turn-taking requires the ability to detect when the other speaker has finished

Multiparty dialogue: A conversation among more than two speakers

Dialogues have structure too

Dialogues have (hierarchical) structure:

"Adjacency pairs": Some acts (first pair part) typically followed by (set up expectation for) another (second pair part):

Question → Answer,
Proposal → Acceptance/Rejection, etc.

Sometimes, a **subdialogue** is required (e.g. for clarification questions):

A: I want to book a ticket for tomorrow

B: Sorry, I didn't catch where you want to go?
A: To Chicago
B: And where do you want to leave from?
…

B: Okay, I've got the following options: …

Chatbots vs Dialogue Systems

Chatbots: Chitchat, often used for entertainment, originally as testbed for clinical therapy

Dialogue Systems: Typically to perform specific tasks (e.g. customer service, reservations, etc., smart devices, cars, etc.)

Task-driven dialog as slot filling

If the purpose of the dialog is to complete a specific **task** (e.g. book a plane ticket), that task can often be represented as a **frame** with a number of **slots** to fill.

The task is completed if all necessary slots are filled.

This assumes a "**domain ontology**":

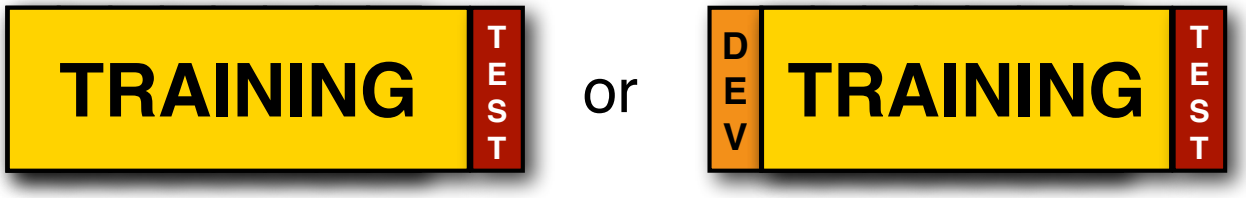
A knowledge structure representing possible user intentions for the given task


Key concepts:
Evaluation

Evaluating Classifiers

Evaluation setup:

Split data into separate **training,** (**dev**elopment) and **test** sets.



Better setup: **n-fold cross validation**:

Split data into *n* sets of equal size

Run *n* experiments, using set *i* to test and remainder to train

This gives average, maximal and minimal accuracies

When **comparing two classifiers**:

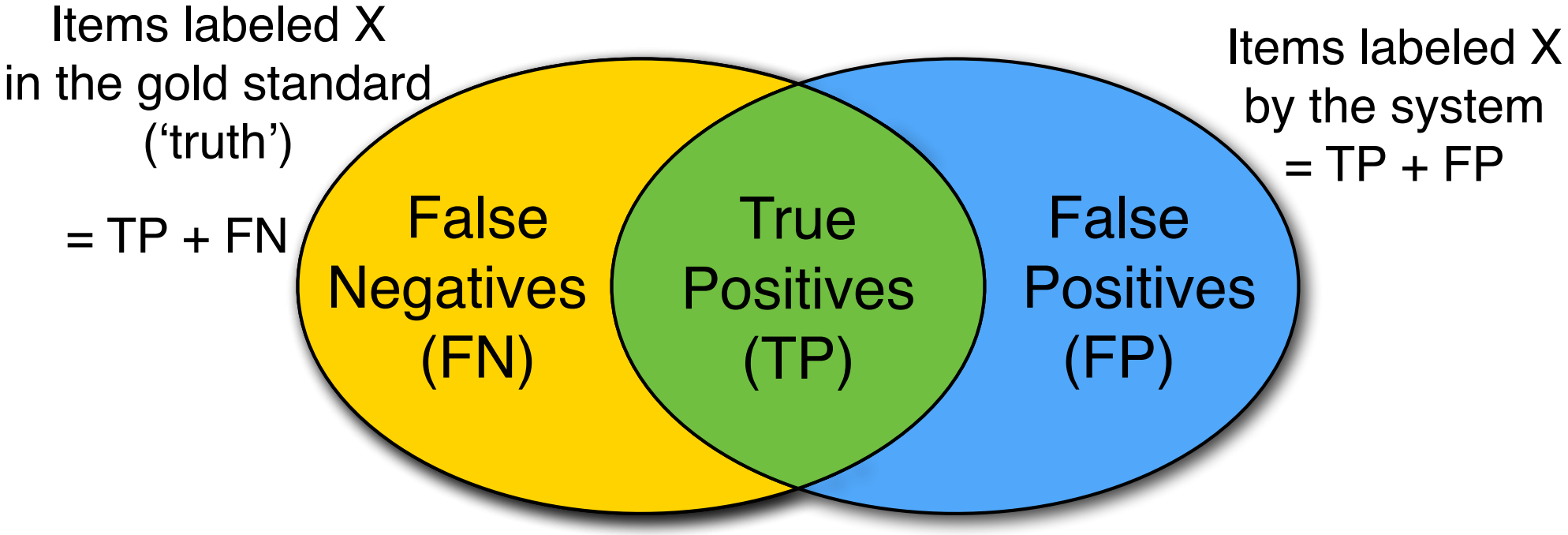
Use the **same** test and training data with the same classes

Confusion Matrices

A confusion matrix tabulates how many items that are labeled with class y in the gold data are labeled with class y' by the classifier.

		gold labels		
		urgent	normal	spam
system output	urgent	8	10	1
	normal	5	60	50
	spam	3	30	200

Precision, Recall, F-Measure



Precision: $P = TP / (TP + FP)$

Recall: $R = TP / (TP + FN)$

F-measure: harmonic mean of precision and recall

$$F = (2 \cdot P \cdot R) / (P + R)$$

Macro-average vs. Micro-average

Which average should you report?

Macro-average (average P/R of all classes):

Useful if performance on all *classes*
is equally important.

Micro-average (average P/R of all items):

Useful if performance on all *items*
is equally important.

Intrinsic vs extrinsic evaluation

How do we know whether one model
is better than another?

There are two ways to evaluate models:

- intrinsic evaluation (on an unseen test set)
  measures how well the model captures what it is supposed to capture (e.g. probabilities, precision/recall or accuracies on unseen test data)
- extrinsic (task-based) evaluation measures how useful the model when used as a component in a particular task.

Both cases require an evaluation metric
that allows us to measure and compare
the performance of different models.

Human vs automatic evaluation Example: Machine Translation

What do we need to evaluate in machine translation?

- Correctness of the translation
- Fluency of the translation, appropriateness, …

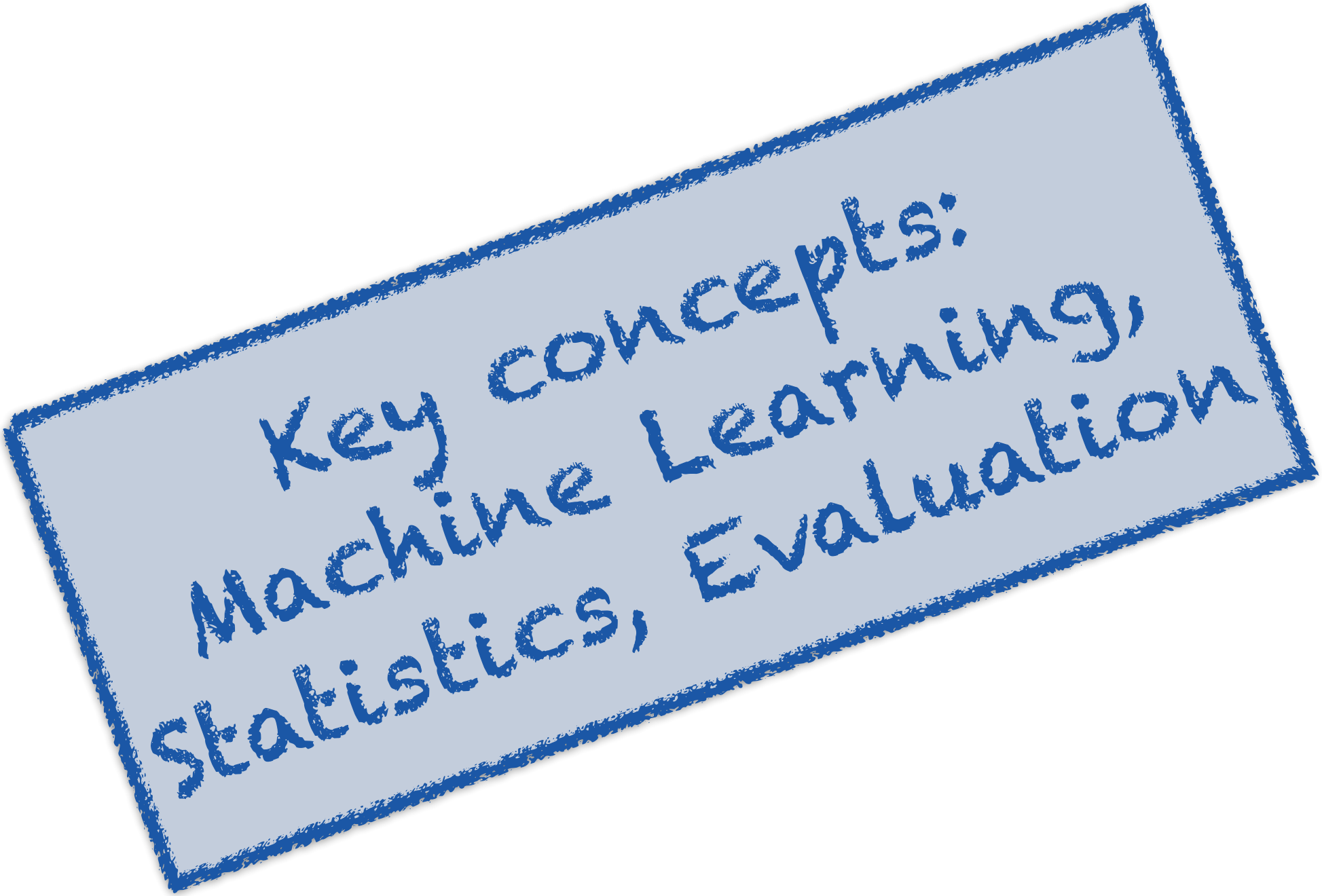
We need appropriate evaluation metrics

Automatic evaluation (on an unseen test set)

Inexpensive, can be done on a large scale,
but may not capture what we want to evaluate.

Human evaluation (also on an unseen test set)

Expensive, and not easily reproducible or comparable across evaluations (different judges, different questions, …)
But human evaluation is often required to actually measure what we want the evaluation to capture.


Key concepts:
Machine Learning,
Statistics; Evaluation

Classification and classifiers

A **classifier** is a function $f(\mathbf{x})$ that maps input items $\mathbf{x} \in X$ to class labels $y \in Y$

(X is a vector space, Y is a finite set)

Binary classification:
Each input item is mapped to exactly one of 2 classes

Multi-class classification:
Each input item is mapped to exactly one of K classes (K > 2)

Multi-label classification:
Each input item is mapped to N of K classes
(N ≥1, varies per input item)

Classification as supervised machine learning

Classification tasks: Map inputs to a fixed set of class labels

Underlying assumption: Each input *really* has one (or N) correct labels
Corollary: The correct mapping is a function (aka the 'target function')

How do we obtain a classifier (model) for a given task?

— If the target function is very simple (and known), implement it directly

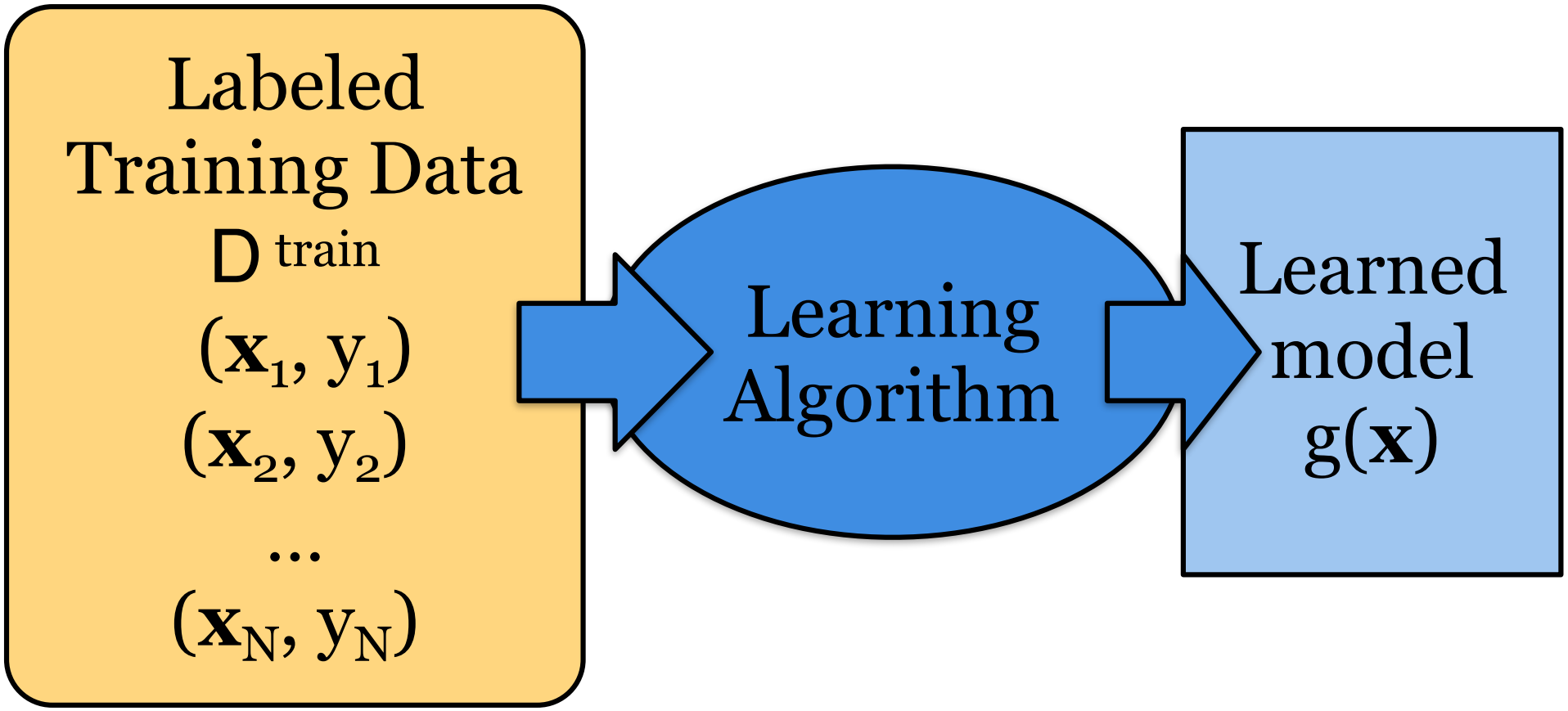
— Otherwise, if we have enough correctly labeled data, estimate (aka. learn/train) a classifier based on that labeled data.

Supervised machine learning:

Given (correctly) ***labeled* training data**, obtain a classifier that predicts these labels as accurately as possible.

Learning is supervised because the learning algorithm can get **feedback** about how accurate its predictions are **from the labels in the training data.**

Supervised learning: **Training**



Give the learning algorithm examples in D train

The learning algorithm returns a model g(**x**)

Learning = Optimization = Loss Minimization

Learning = parameter estimation = optimization:

Given a particular class of model (logistic regression, Naive Bayes, …) and data D_{train}, find **the *best* parameters** for this class of model on D_{train}

If the model is a probabilistic classifier, think of optimization as Maximum Likelihood Estimation (**MLE**)

*"Best" = return (among all possible parameters for models of this class) parameters that assign the **largest probability** to* D_{train}

In general (incl. for probabilistic classifiers), think of optimization as **Loss Minimization**:

*"Best" = return (among all possible parameters for models of this class) parameters that have the **smallest loss** on* D_{train}

"**Loss**": how bad are the predictions of a model?

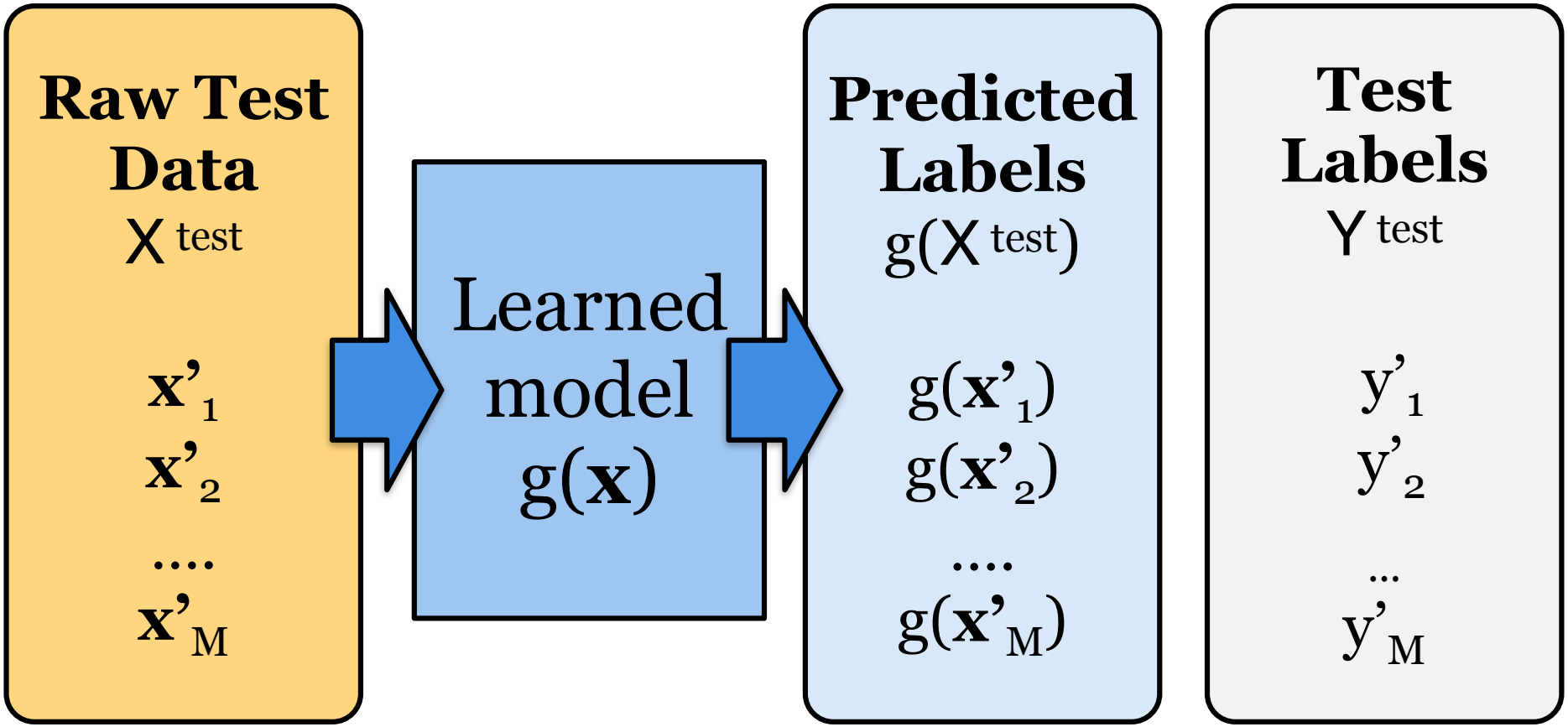
*The **loss function** we use to measure loss depends on the class of model*

$L(\hat{y}, y)$: how bad is it to predict $\hat{y}$ if the correct label is y ?

Supervised learning: **Testing**

Reserve some data for testing.

Apply the learned model to the raw test data to obtain **predicted labels** for the test data and compare against the actual test labels.



Probabilistic classifiers

A **probabilistic classifier** returns the *most likely* class y^* for input $\mathbf{x}$:

$$y^* = \text{argmax}_y P(Y = y \mid \mathbf{X} = \mathbf{x})$$

Naive Bayes uses Bayes Rule:

$$y^* = \text{argmax}_y P(y \mid \mathbf{x}) = \text{argmax}_y P(\mathbf{x} \mid y) P(y)$$

Naive Bayes models the joint distribution of the class and the data:

$$P(\mathbf{x} \mid y) P(y) = P(\mathbf{x}, y)$$

Joint models are also called **generative** models because we can view them as stochastic processes that *generate* (labeled) items:

Sample/pick a label y with $P(y)$, and then an item $\mathbf{x}$ with $P(\mathbf{x} \mid y)$

Logistic Regression models $P(y \mid \mathbf{x})$ directly

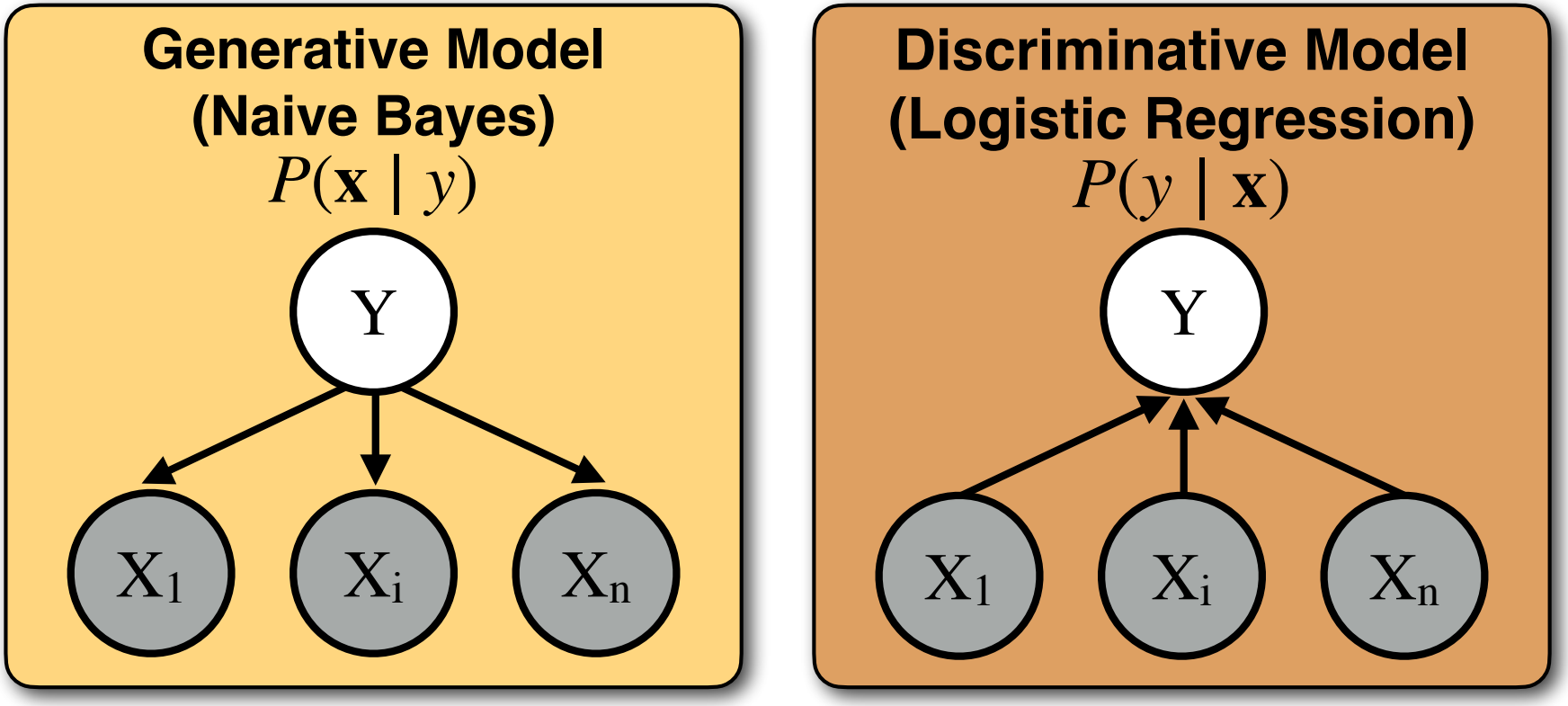
This is also called a **discriminative** or **conditional** model, because it only models the probability of the class given the input, and not of the raw data itself.

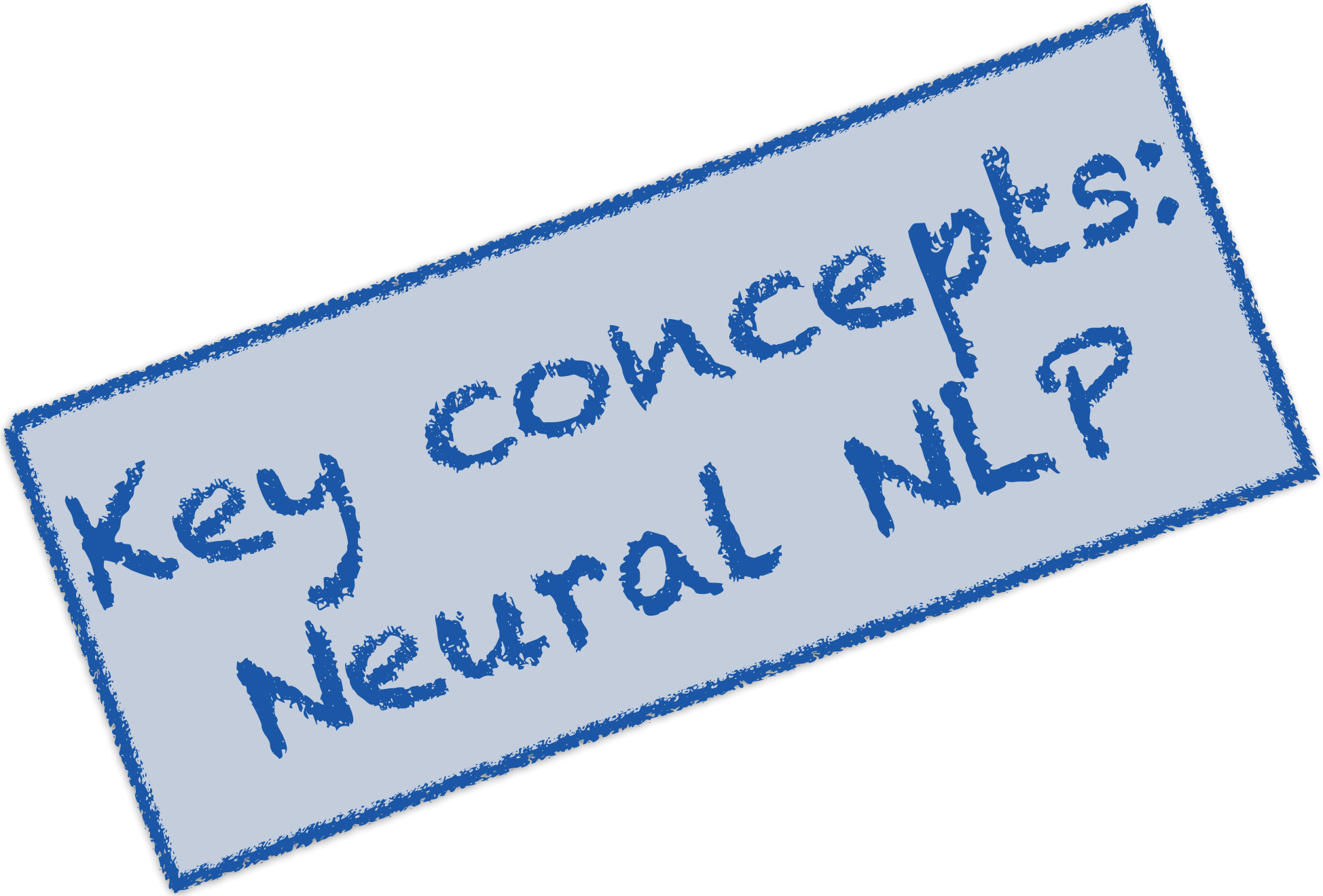
Generative vs Discriminative Models

In classification:

— The data $\mathbf{x} = (x_1, \ldots, x_n)$ is observed (shaded nodes).

— The label y is hidden (and needs to be inferred)




Key concepts:
Neural NLP

What are neural networks?

A family of machine learning models that was originally inspired by how neurons (nerve cells) process information and learn.

In NLP, neural networks are now widely used, e.g. for

- **Classification**
  (e.g. sentiment analysis)
- **(Sequence) generation**
  (e.g. in machine translation, response generation for dialogue, etc.
- **Representation Learning** (neural embeddings)
  (word embeddings, sequence embeddings, graph embeddings,…)
- **Structure Prediction** (incl. sequence labeling)
  (e.g. part-of-speech tagging, named entity recognition, parsing,…)

The Perceptron (Rosenblatt 1958)

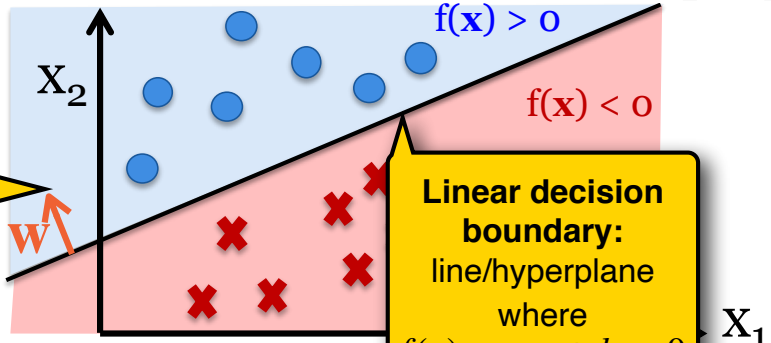
A **linear classifier** based on a **threshold activation** function:

Return $y = +1$ iff $f(\mathbf{x}) = \mathbf{w}\mathbf{x} + b > 0$

$y = -1$ iff $f(\mathbf{x}) = \mathbf{w}\mathbf{x} + b \leq 0$

$y \in \{-1, +1\}$ makes the **update rule** easier to write than $y \in \{0,1\}$

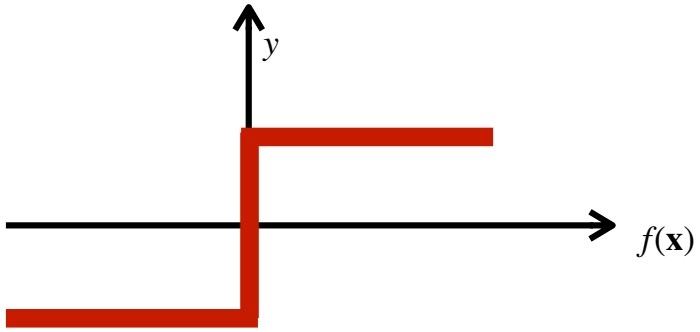
Linear classifier for $\mathbf{x} = (x_1, x_2)$



w is **orthogonal** to the **decision boundary**

Linear decision boundary: line/hyperplane where $f(\mathbf{x}) = \mathbf{w}\mathbf{x} + b = 0$

Threshold Activation



Threshold activation is inspired by the "all-or-none character" (McCulloch & Pitts, 1943) of how neurons process information

Perceptron update rule: (online stochastic gradient descent)

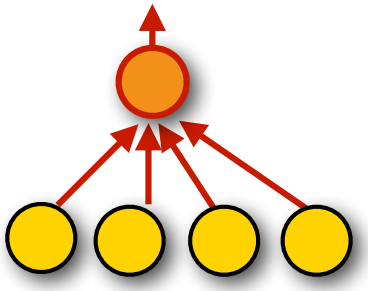
If the predicted $\hat{y}^{(i)} \neq y^{(i)}$: $\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} + \eta y^{(i)} \mathbf{x}^{(i)}$

Increment w (**lower the slope** of the decision boundary) when y **should be** +1, decrement **w** when it should be -1)

Training: Change weights when the model makes a **mistake**

From Perceptrons to (Feedforward) Neural Nets

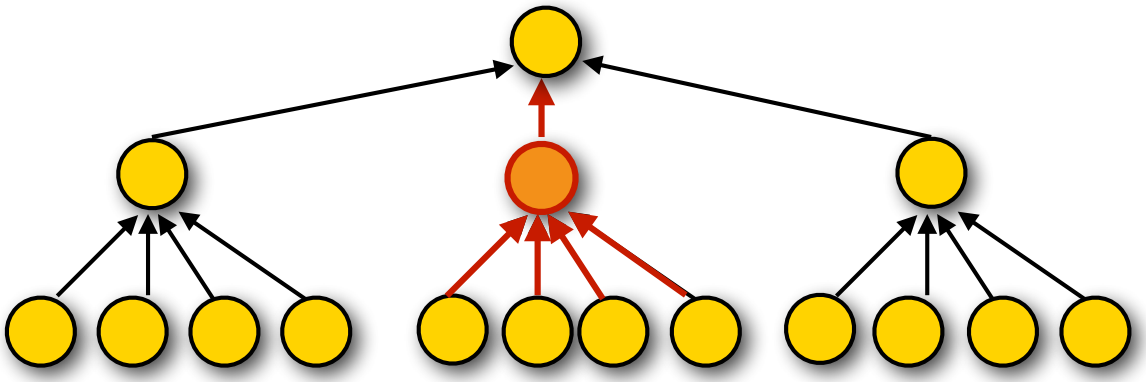
A **perceptron** can be seen as a single neuron (one output unit with a vector or **layer** of input units):

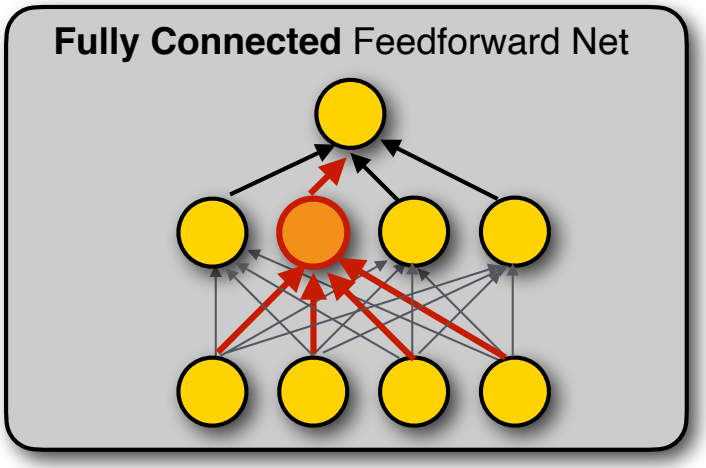


Output unit: scalar $y = f(\mathbf{x})$

Input layer: vector $\mathbf{x}$

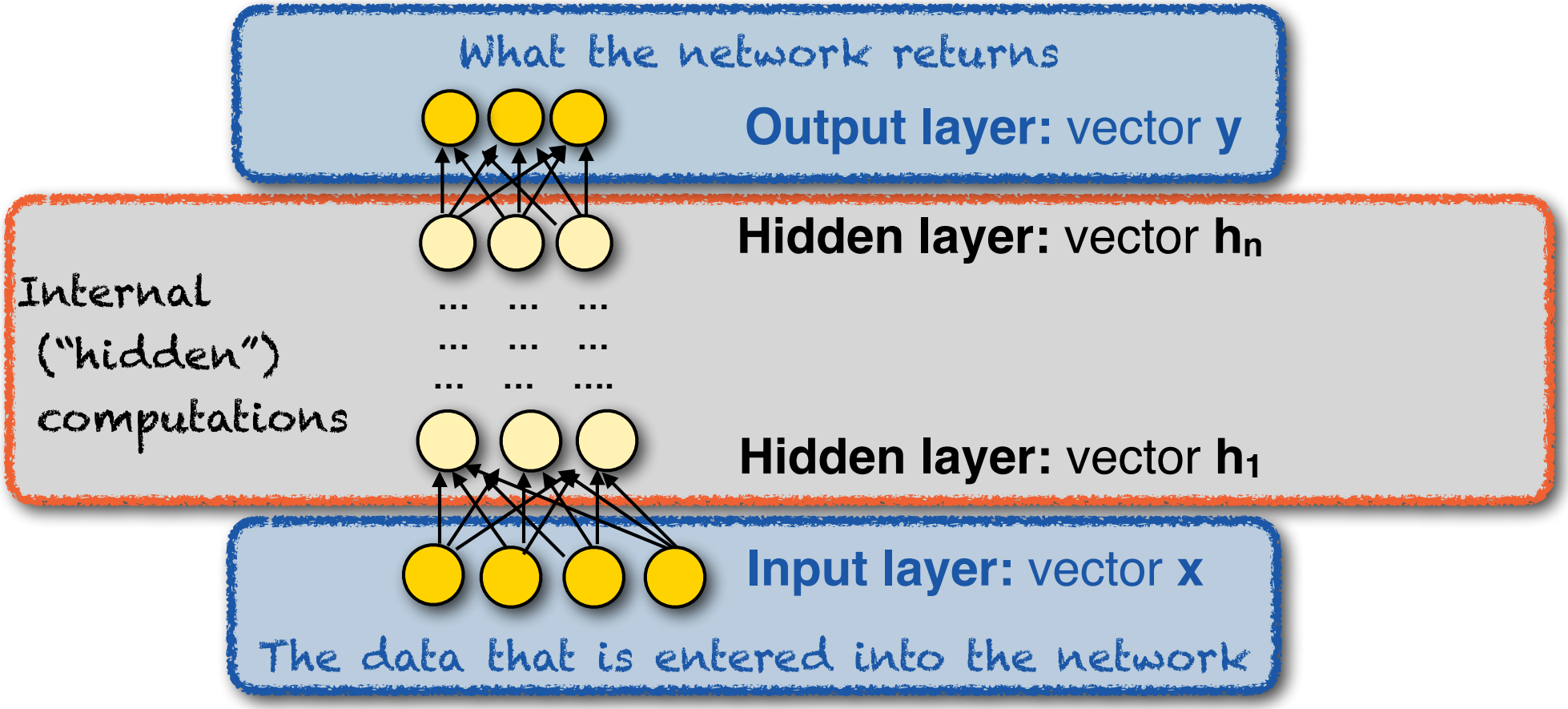
But each element of the input can be a neuron itself:





Multi-layer feedforward networks

We typically assume feedforward networks are organized into layers:



Nonlinear Activation Functions $g()$

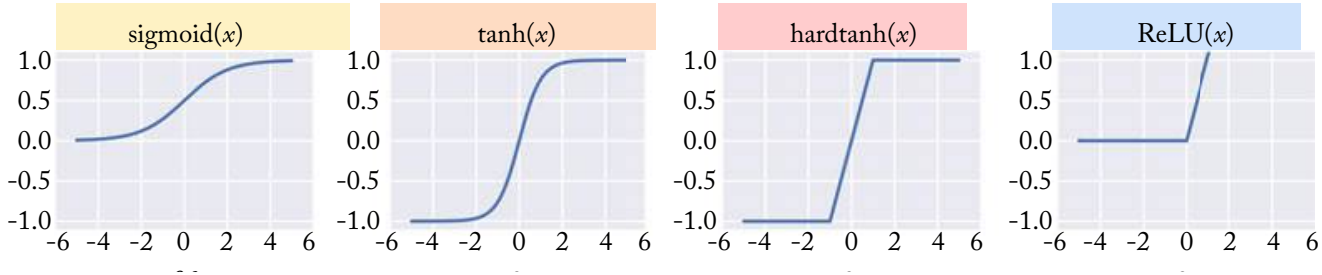


Fig.:Y. Goldberg (2017) Neural Network Methods for Natural Language Processing

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

Outputs in $[0,1]$ range. Useful for output units (**probabilities**), interpolation

Hyperbolic tangent: $\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$

Outputs in $[-1,1]$ range. Useful for **internal** units

Hard tanh $\text{htanh}(x) = -1 \text{ for } x < -1,\ 1 \text{ for } x > 1,\ x \text{ otherwise}$

Outputs in $[-1,1]$ range. Approximates $\tanh$

Rectified Linear Unit: $ReLU(x) = \max(0,x)$

Outputs in $[0, +\infty]$. Works very well for **internal** units.

The **softmax** function

The **softmax** function turns *any* vector of reals $\mathbf{z} = (z_1, \ldots, z_n)$ into a discrete probability distribution $\mathbf{p} = (p_1, \ldots, p_n)$ where $\forall_{j \in \{1,\ldots,n\}}: 0 < p_j < 1$ and $\Sigma_{j=1}^{n} p_j = 1$

$$p_j = \text{softmax}(\mathbf{z})_j = \frac{\exp(z_j)}{\sum_{k=1}^{K} \exp(z_k)}$$

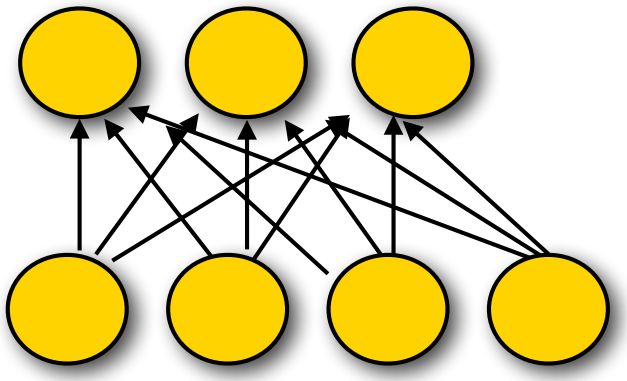
Logistic regression applies the softmax to a linear combination of the input features $\mathbf{x}$: $\mathbf{z} = \mathbf{fx}$

Models based on logistic regression are also known as Maximum Entropy (**MaxEnt**) models

We will see the softmax again when we talk about **neural nets,** but there the input is typically a much more complex, nonlinear function of the input features.

Multi-Class Classification
with a multilayer feedforward net

With K output classes, the output layer has K **units** with a **softmax** activation function:



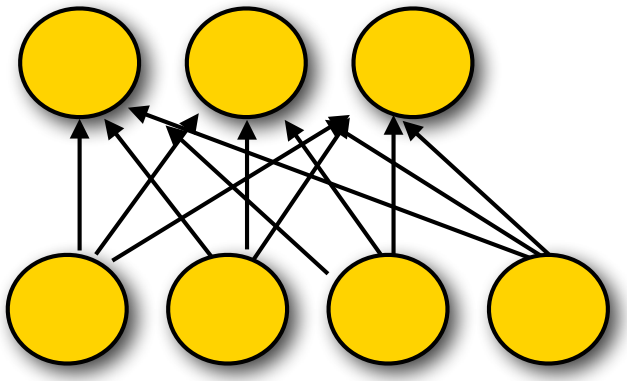
Output layer:
A vector $\mathbf{y} = (y_1, \ldots, y_K)$ where the *i*-th element corresponds to the probability that the input has class *i*:

$$y_i = \mathbf{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{k=1}^{K} \exp(z_k)}$$

such that we get a categorical distribution over all *K* classes

Multi-Label Classification
with a multilayer feedforward net

With K output classes, K output units
with K **sigmoid** activation functions:

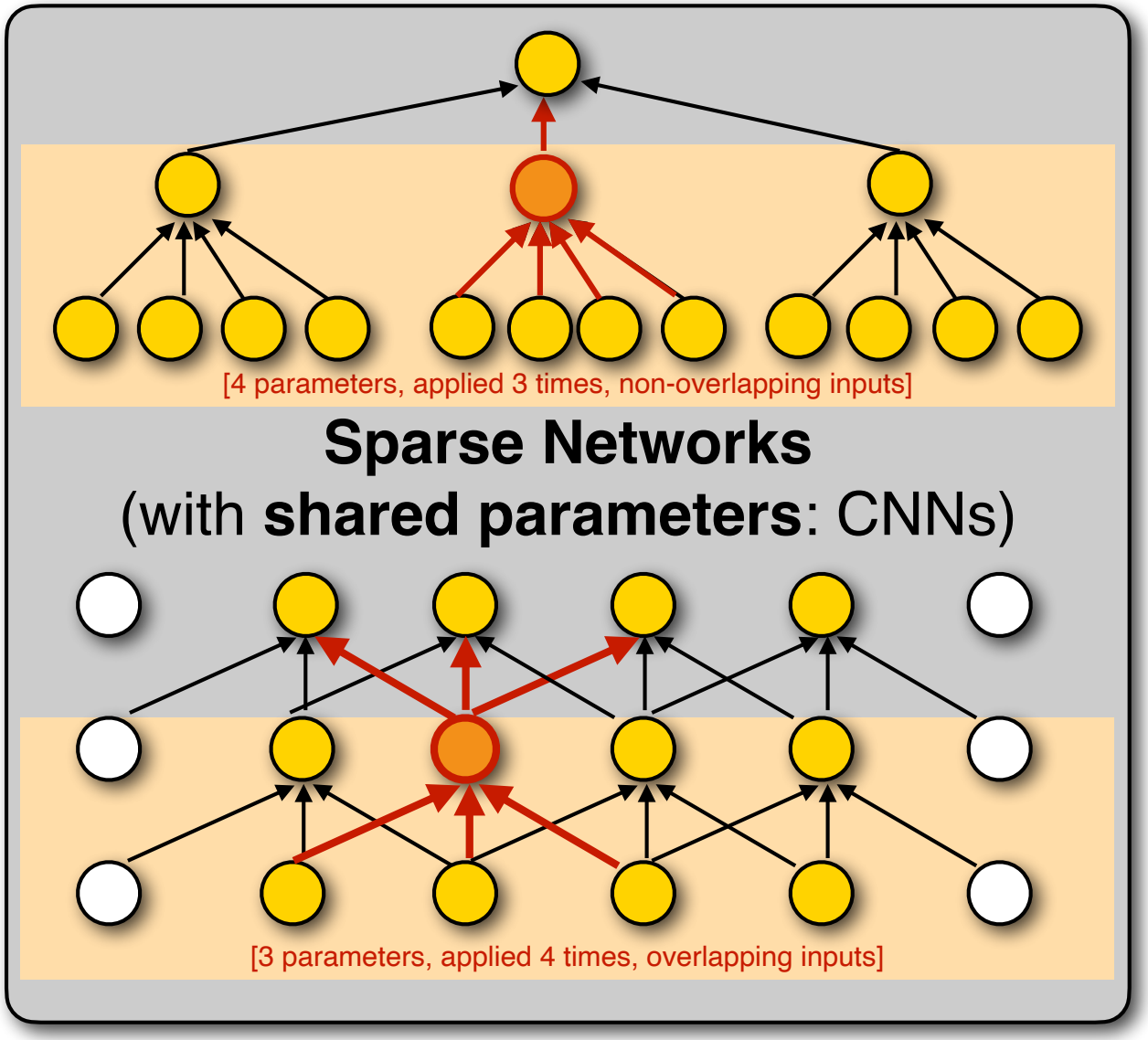


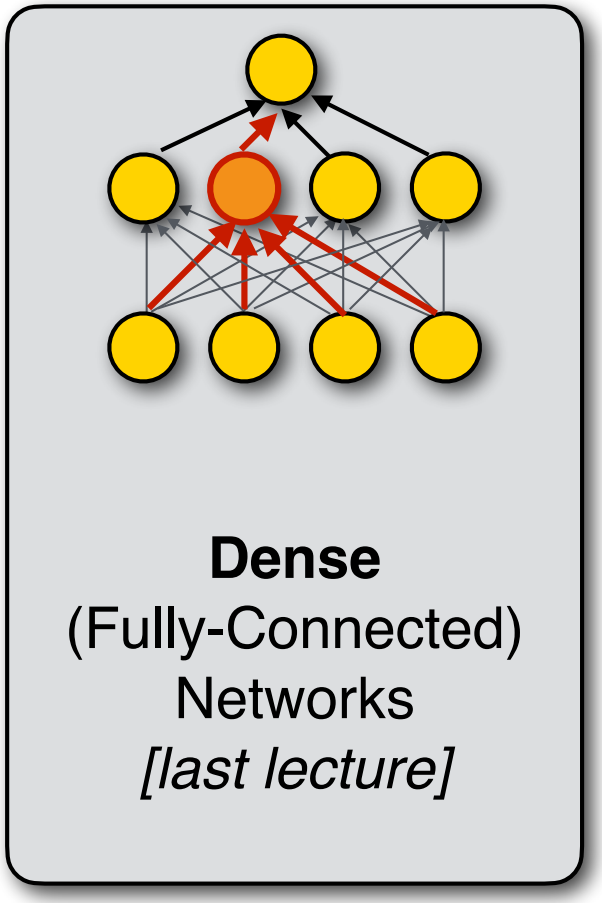
Output layer:
A vector $\mathbf{y} = (y_1, \ldots, y_K)$ where the *i*-th element corresponds to the probability that the input does (or doesn't) have class *i*:

$$y_i = \mathbf{sigmoid}(\mathbf{w}_i \mathbf{x}_i + b_i)$$

We now have a separate probability for each possible class label.

Convolutional Neural Nets (ConvNets, CNNs)





1D CNNs for text

Text is a (variable-length) **sequence** of words (word vectors)

[#channels = dimensionality of word vectors]

We can use a **1D CNN** to slide a window of *n* tokens across:

— Filter size n = 3, stride = 1, no padding

The quick brown fox jumps over the lazy dog
The **quick brown fox** jumps over the lazy dog
The quick **brown fox jumps** over the lazy dog
The quick brown **fox jumps over** the lazy dog
The quick brown fox **jumps over the** lazy dog
The quick brown fox jumps **over the lazy** dog

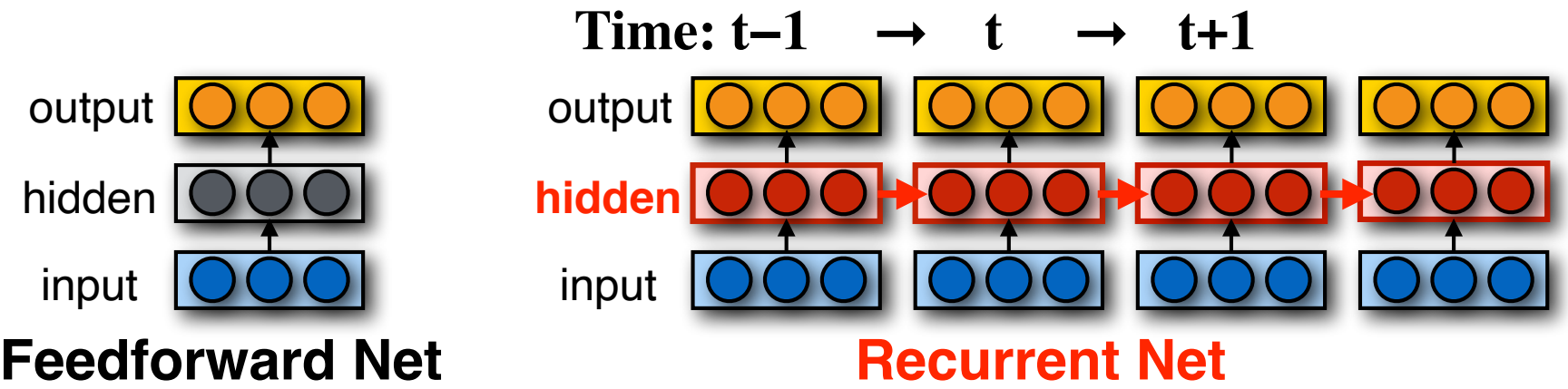
— Filter size n = 2, stride = 2, no padding:

The quick brown fox jumps over the lazy dog
The quick **brown fox** jumps over the lazy dog
The quick brown fox **jumps over** the lazy dog
The quick brown fox jumps over **the lazy** dog

Recurrent neural networks (RNNs)

Basic RNN: Generate a **sequence of *T* outputs**
by running a variant of a feedforward net T times.

Recurrence:
The **hidden state** computed at the **previous step** ($\mathbf{h}^{(t-1)}$)
is fed into **the hidden state** at the **current step** ($\mathbf{h}^{(t)}$)
With H hidden units, this requires additional H^2 parameters



RNN variants: LSTMs, GRUs

Long Short-Term Memory networks (**LSTMs**)
are RNNs with a more complex recurrent architecture

Gated Recurrent Units (GRUs)
are a simplification of LSTMs

Both contain "**Gates**" to control how much of the input or previous hidden state to forget or remember

LSTMs are more expressive than GRUs and basic RNNs (they're better at learning long-range dependencies), but GRUs are easier to train than LSTMs (useful when training data is limited)

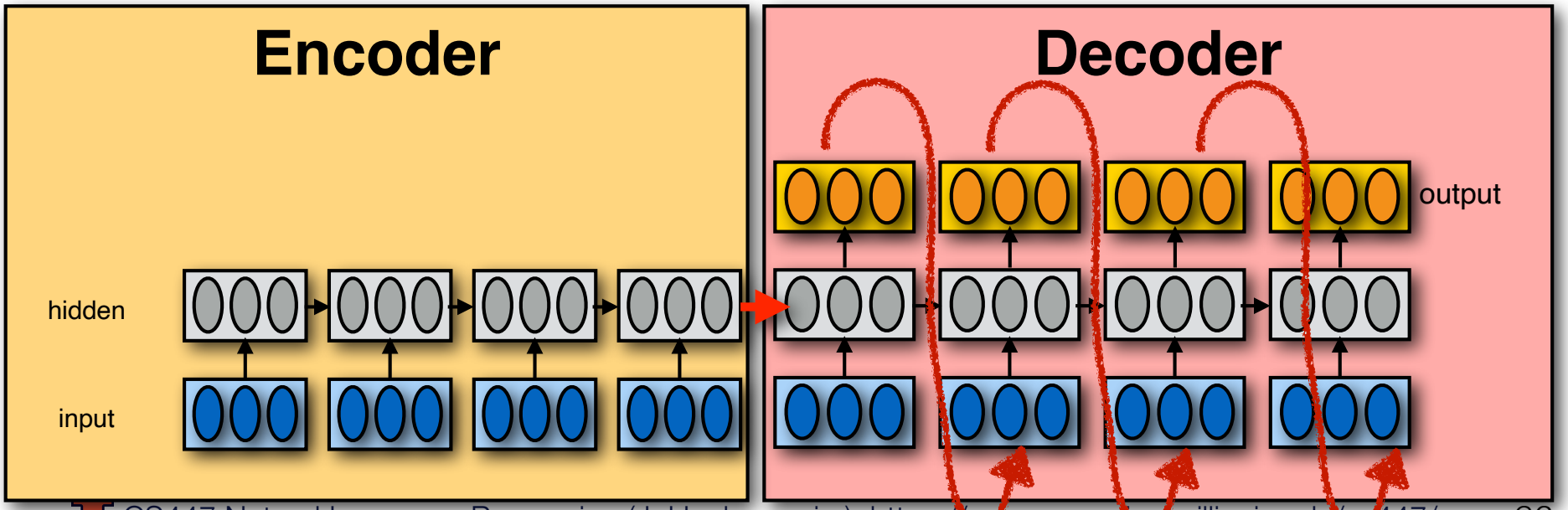
Encoder-Decoder (seq2seq) model

Task: Read an input sequence
and return an output sequence

- Machine translation: translate source into target language
- Dialog system/chatbot: generate a response

Reading the input sequence: **RNN Encoder**

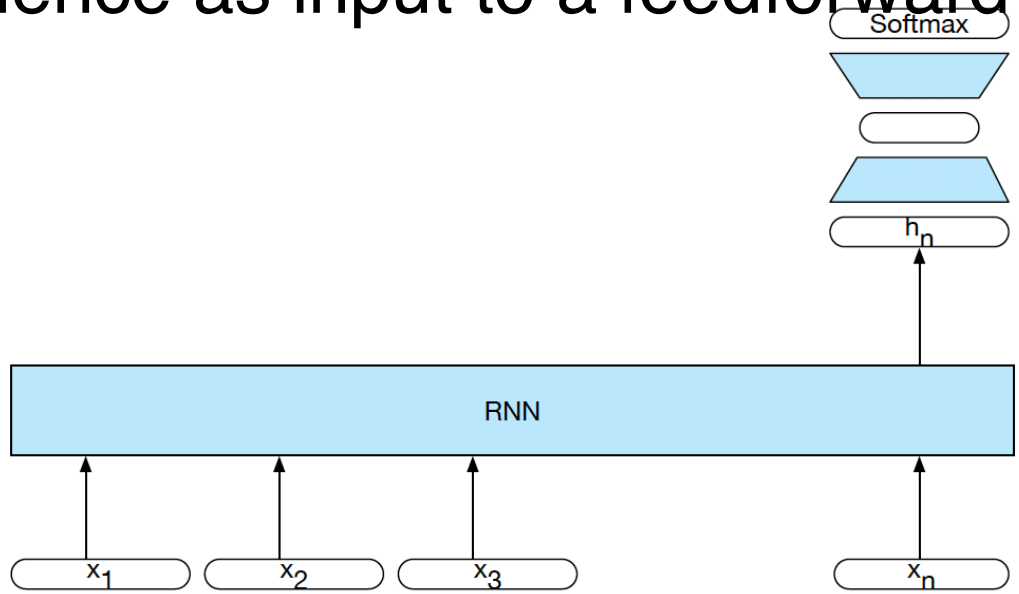
Generating the output sequence: **RNN Decoder**



RNNs for sequence classification

If we just want to assign **one label** to the entire sequence, we don't need to produce output at each time step, so we can use a simpler architecture.

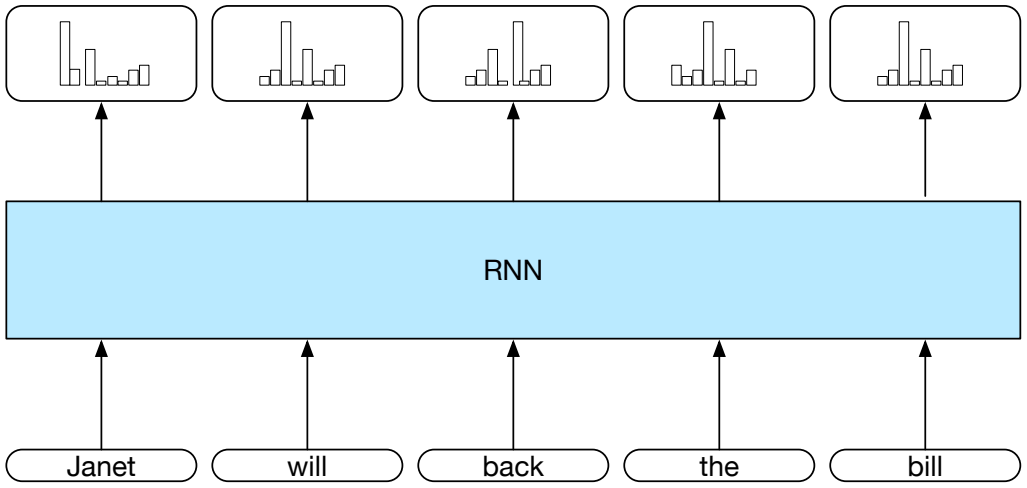
We can use the hidden state of the last word in the sequence as input to a feedforward net:



Basic RNNs for sequence labeling

Sequence labeling (e.g. POS tagging):
Assign **one label to each element** in the sequence.

RNN Architecture:
Each time step has a distribution over output classes



Extension: add a CRF layer to capture dependencies among labels of adjacent tokens.

Adding attention to the decoder

We want to **condition the output** generation of the decoder on a **context-dependent representation of the input** sequence.

Attention computes a probability **distribution over the encoder's hidden states** that depends on the **decoder's current hidden state**

(This distribution is **computed anew for each output symbol**)

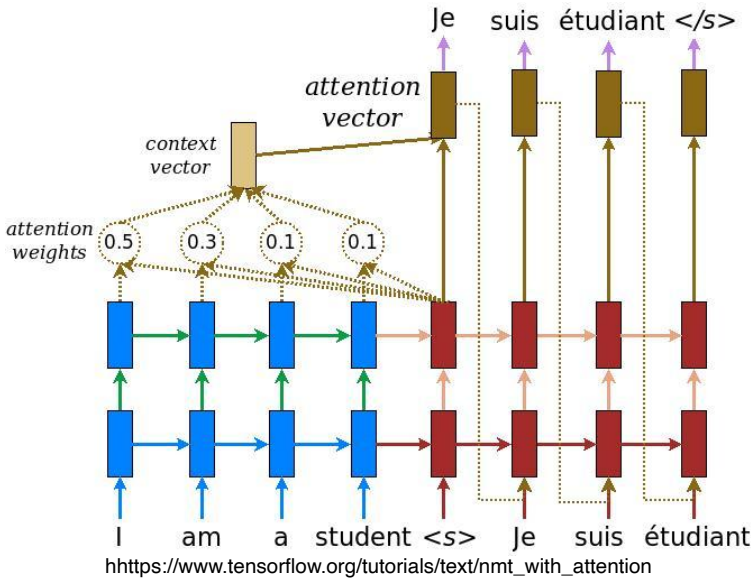
This attention distribution is used to compute **a weighted average of the encoder's hidden state vectors**.

This **context-dependent embedding of the input sequence** is fed into the output of the decoder RNN.

Attention, more formally

Define a **probability distribution** $\alpha^{(t)} = \left(\alpha_1^{(t)}, \ldots, \alpha_S^{(t)}\right)$ over the ***S* elements of the input** sequence **that depends on the current output** element *t*

Use this distribution to compute a **weighted average of the encoder's output** $\sum_{s=1..S} \alpha_s^{(t)} \mathbf{o}_s$ or hidden states $\sum_{s=1..S} \alpha_s^{(t)} \mathbf{h}_s$ and feed that into the decoder.



hhttps://www.tensorflow.org/tutorials/text/nmt_with_attention

Attention, more formally

1. Compute **a probability distribution** $\alpha^{(t)} = \left(\alpha_1^{(t)}, \ldots, \alpha_S^{(t)}\right)$ over the *encoder's* hidden states $\mathbf{h}^{(s)}$ that depends on the *decoder's* current $\mathbf{h}^{(t)}$

$$\alpha_s^{(t)} = \frac{\exp\left(s(\mathbf{h}^{(t)}, \mathbf{h}^{(s)})\right)}{\sum_{s'} \exp\left(s(\mathbf{h}^{(t)}, \mathbf{h}^{(s')})\right)}$$

2. Use $\alpha^{(t)}$ to compute **a weighted avg.** $\mathbf{c}^{(t)}$ of the *encoder's* $\mathbf{h}^{(s)}$:

$$\mathbf{c}^{(t)} = \sum_{s=1..S} \alpha_s^{(t)} \mathbf{h}^{(s)}$$

3. Use both $\mathbf{c}^{(t)}$ and $\mathbf{h}^{(t)}$ to compute **a new output** $\mathbf{o}^{(t)}$, e.g. as

$$\mathbf{o}^{(t)} = \tanh\left(W_1 \mathbf{h}^{(t)} + W_2 \mathbf{c}^{(t)}\right)$$

Defining Attention Weights

Hard attention (degenerate case, non-differentiable):

$\alpha^{(t)} = \left(\alpha_1^{(t)}, \ldots, \alpha_S^{(t)}\right)$ is a **one-hot vector**

(e.g. 1 = most similar element to decoder's vector, 0 = all other elements)

Soft attention (general case):

$\alpha^{(t)} = \left(\alpha_1^{(t)}, \ldots, \alpha_S^{(t)}\right)$ is **not a one-hot**

- Use the **dot product** (no learned parameters):

$$s\left(\mathbf{h}^{(t)}, \mathbf{h}^{(s)}\right) = \mathbf{h}^{(t)} \cdot \mathbf{h}^{(s)}$$

- Learn a **bilinear matrix** W:

$$s\left(\mathbf{h}^{(t)}, \mathbf{h}^{(s)}\right) = \left(\mathbf{h}^{(t)}\right)^T W \mathbf{h}^{(s)}$$

- Learn **separate weights** for the hidden states:

$$s\left(\mathbf{h}^{(t)}, \mathbf{h}^{(s)}\right) = \mathbf{v}^T \tanh\left(W_1 \mathbf{h}^{(t)} + W_2 \mathbf{h}^{(s)}\right)$$

Transformers

Sequence transduction model based on **attention** (**no convolutions or recurrence**)

— easier to parallelize than recurrent nets

— faster to train than recurrent nets

— captures more long-range dependencies than CNNs with fewer parameters

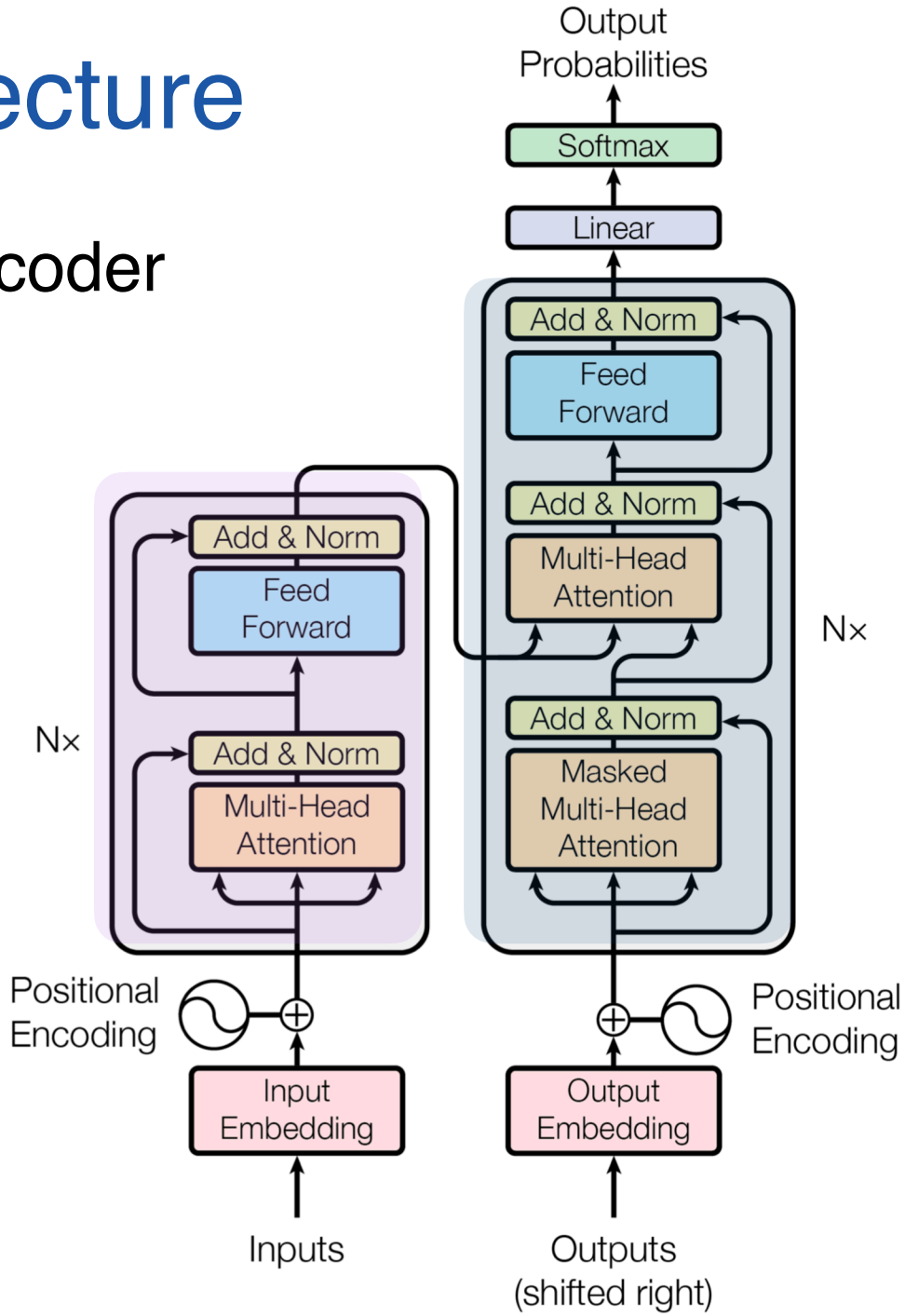
Transformers use stacked self-attention and position-wise, fully-connected layers for the encoder and decoder

Transformers form the basis of BERT, GPT(2-3), and other state-of-the-art neural sequence models.

Transformer Architecture

Non-Recurrent Encoder-Decoder architecture

- No hidden states
- Context information captured via attention and positional encodings
- Consists of stacks of layers with various sublayers



Language models

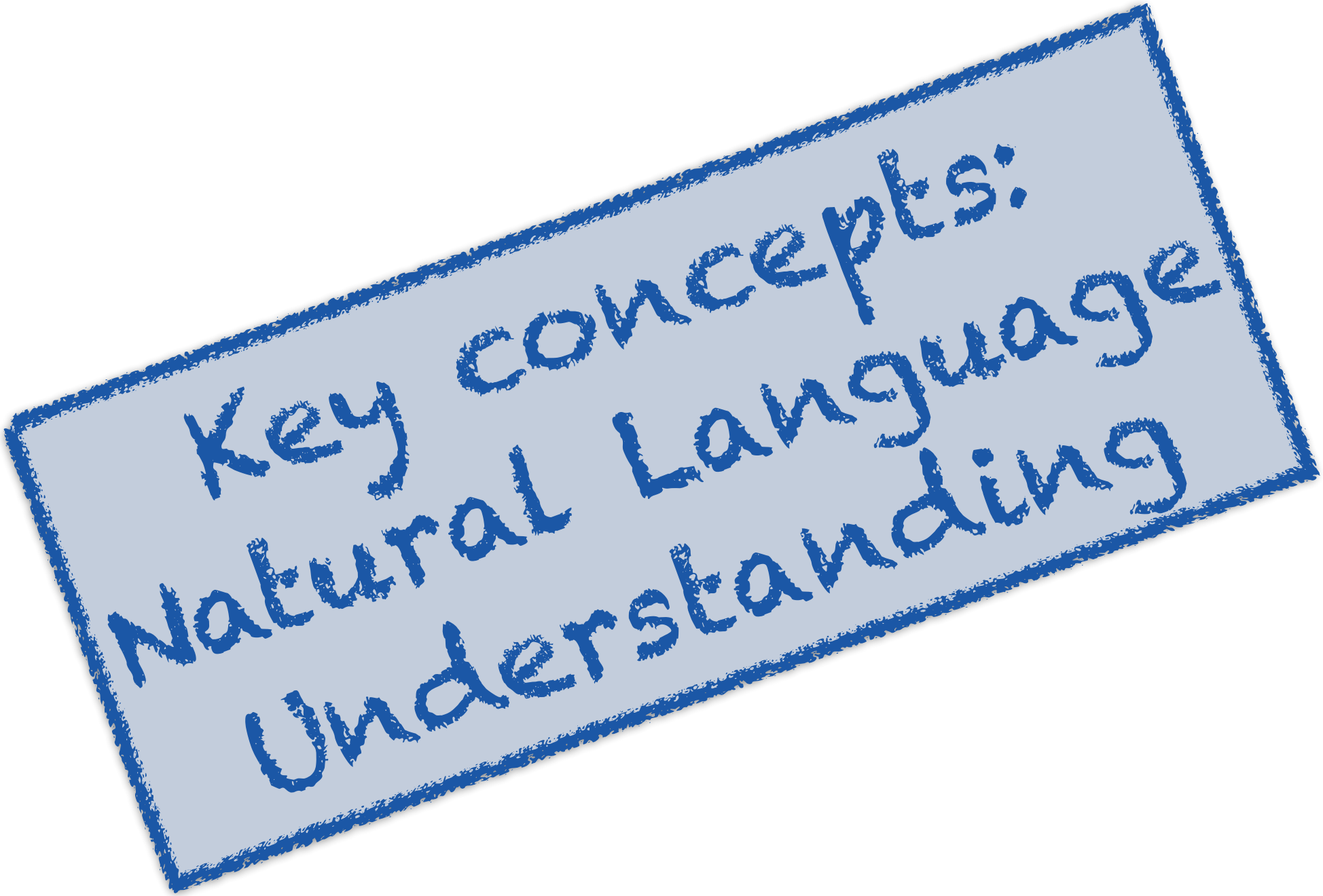
Many NLP tasks require **natural language output**:

- **Machine translation**: return text in the target language
- **Speech recognition**: return a transcript of what was spoken
- **Natural language generation**: return natural language text
- **Spell-checking**: return corrected spelling of input

Language models define **probability distributions over** (natural language) **strings or sentences**.

➔ We can use a language model to generate strings

➔ We can use a language model to score/rank candidate strings so that we can choose the best (i.e. most likely) one:
if $P_{LM}(A) > P_{LM}(B)$, return A, not B


Key concepts:
Natural Language
Understanding

The NLP Pipeline

An NLP system may use some or all of the following steps:

Tokenizer/Segmenter

to identify words and sentences

Morphological analyzer/POS-tagger

to identify the part of speech and structure of words

Word sense disambiguation

to identify the meaning of words

Syntactic/semantic Parser

to obtain the structure and meaning of sentences

Coreference resolution/discourse model

to keep track of the various entities and events mentioned

NLP Pipeline: Assumptions

Each step in the NLP pipeline embellishes the input with **explicit information** about its linguistic structure

- POS tagging: parts of speech of word,
- Syntactic parsing: grammatical structure of sentence,….

Each step in the NLP pipeline requires its own explicit ("symbolic") **output representation**:

- POS tagging requires a POS tag set
  - (e.g. NN=common noun singular, NNS = common noun plural, …)
- Syntactic parsing requires constituent or dependency labels
  - (e.g. NP = noun phrase, or nsubj = nominal subject)

These representations should capture linguistically appropriate **generalizations/abstractions**

- Designing these representations requires linguistic expertise

NLP Pipeline: Shortcomings

Each step in the pipeline relies on a **learned model** that will return the *most likely* representations

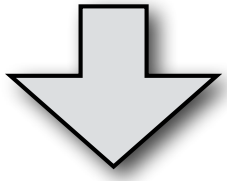
- This requires a lot of **annotated training data** for each step
- Annotation is **expensive** and sometimes **difficult** (people are not 100% accurate)
- These models are **never 100% accurate**
- Models make more mistakes if their input contains mistakes

How do we know that we have captured the "***right***" **generalizations** when designing representations?

- Some representations are **easier to predict** than others
- Some representations are **more useful** for the next steps in the pipeline than others
- But we won't know how easy/useful a representation is until we have a model that we can plug into a particular pipeline

Statistical POS tagging

	She	promised	to	back	the	bill
$\mathbf{w} =$	$w^{(1)}$	$w^{(2)}$	$w^{(3)}$	$w^{(4)}$	$w^{(5)}$	$w^{(6)}$



$\mathbf{t} =$	$t^{(1)}$	$t^{(2)}$	$t^{(3)}$	$t^{(4)}$	$t^{(5)}$	$t^{(6)}$
	PRP	**VBD**	**TO**	**VB**	**DT**	**NN**

What is the most likely sequence of tags $\mathbf{t}= t^{(1)}\ldots t^{(N)}$ for the given sequence of words $\mathbf{w}= w^{(1)}\ldots w^{(N)}$?

$$\mathbf{t}^* = \text{argmax}_t P(\mathbf{t} \mid \mathbf{w})$$

POS tagging with generative models

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

$P(\mathbf{t},\mathbf{w})$: the joint distribution of the labels we want to predict ($\mathbf{t}$) and the observed data ($\mathbf{w}$).

We decompose $P(\mathbf{t},\mathbf{w})$ into $P(\mathbf{t})$ and $P(\mathbf{w} \mid \mathbf{t})$ since these distributions are easier to estimate.

Models based on joint distributions of labels and observed data are called generative models: think of $P(\mathbf{t})P(\mathbf{w} \mid \mathbf{t})$ as a stochastic process that first generates the labels, and then generates the data we see, based on these labels.

Hidden Markov Models (HMMs)

HMMs are the most commonly used generative models for POS tagging (and other tasks, e.g. in speech recognition)

HMMs make specific **independence assumptions** in $P(\mathbf{t})$ and $P(\mathbf{w}|\,\mathbf{t})$:

1) $P(\mathbf{t})$ is an *n*-gram (typically **bigram** or **trigram**) model over tags:

$$P_{\text{bigram}}(\mathbf{t}) = \prod_i P(t^{(i)} \mid t^{(i-1)}) \qquad P_{\text{trigram}}(\mathbf{t}) = \prod_i P(t^{(i)} \mid t^{(i-1)}, t^{(i-2)})$$

$P(t^{(i)} \mid t^{(i-1)})$ and $P(t^{(i)} \mid t^{(i-1)}, t^{(i-2)})$ are called **transition probabilities**

2) In $P(\mathbf{w} \mid \mathbf{t})$, each $w^{(i)}$ depends only on [is generated by/conditioned on] $t^{(i)}$:

$$P(\mathbf{w} \mid \mathbf{t}) = \prod_i P(w^{(i)} \mid t^{(i)})$$

$P(w^{(i)} \mid t^{(i)})$ are called **emission probabilities**

These probabilities don't depend on the position in the sentence $^{(i)}$,
but are defined over word and tag types.
With subscripts $_{i,j,k}$, to index word/tag types, they become $P(t_i \mid t_j)$, $P(t_i \mid t_j, t_k)$, $P(w_i \mid t_j)$

The Viterbi algorithm

A dynamic programming algorithm which finds the best (=most probable) tag sequence $\mathbf{t}^*$ for an input sentence $\mathbf{w}$: $\mathbf{t}^* = \text{argmax}_{\mathbf{t}} P(\mathbf{w} \mid \mathbf{t})P(\mathbf{t})$

Complexity: linear in the sentence length.

With a bigram HMM, Viterbi runs in $O(T^2N)$ steps for an input sentence with N words and a tag set of T tags.

The independence assumptions of the HMM tell us how to break up the big search problem (find $\mathbf{t}^* = \text{argmax}_{\mathbf{t}} P(\mathbf{w} \mid \mathbf{t})P(\mathbf{t})$) into smaller subproblems.

The data structure used to store the solution of these subproblems is the trellis.

Sequence Labeling

Input: a sequence of *n* tokens/words:

```
Pierre Vinken , 61 years old , will join IBM 's board as a
nonexecutive director Nov. 29
```

Output: a sequence of *n* labels, such that each token/word is associated with a label:

POS-tagging: `Pierre_NNP Vinken_NNP ,_, 61_CD years_NNS old_JJ ,_, will_MD join_VB IBM_NNP 's_POS board_NN as_IN a_DT nonexecutive_JJ director_NN Nov._NNP 29_CD ._.`

Named Entity Recognition: `Pierre_B-PERS Vinken_I-PERS ,_O 61_O years_O old_O ,_O will_O join_O IBM_B-ORG 's_O board_O as_O a_O nonexecutive_O director_O Nov._B-DATE 29_I-DATE ._O`

BIO encodings in general

BIO encoding can be used to frame any task that requires the identification of non-overlapping and non-nested text spans as a sequence labeling problem, e.g.:

- NP chunking
- Shallow Parsing
- Named entity recognition

Sequence labeling algorithms

Statistical models:

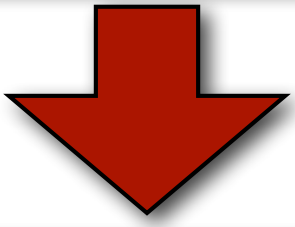
- Maximum Entropy Markov Models (MEMMs)
- Conditional Random Fields (CRFs)

Neural models:

- Recurrent networks (or transformers) that predict a label at each time step, possibly with a CRF output layer.

Named Entity Recognition

```
Pierre Vinken , 61 years old , will join IBM 's board
as a nonexecutive director Nov. 29 .
```



```
[PERS Pierre Vinken] , 61 years old , will join
[ORG IBM] 's board as a nonexecutive director
[DATE Nov. 2] .
```

Task: identify all mentions of named entities (people, organizations, locations, dates)

Named Entity Types

Type	Tag	Sample Categories	Example sentences
People	PER	people, characters	**Turing** is a giant of computer science.
Organization	ORG	companies, sports teams	The **IPCC** warned about the cyclone.
Location	LOC	regions, mountains, seas	The **Mt. Sanitas** loop is in **Sunshine Canyon**.
Geo-Political Entity	GPE	countries, states, provinces	**Palo Alto** is raising the fees for parking.
Facility	FAC	bridges, buildings, airports	Consider the **Golden Gate Bridge**.
Vehicles	VEH	planes, trains, automobiles	It was a classic **Ford Falcon**.

Figure 18.1 A list of generic named entity types with the kinds of entities they refer to.

These types were developed for the news domain as part of NIST's Automatic Content Extraction (ACE) program.

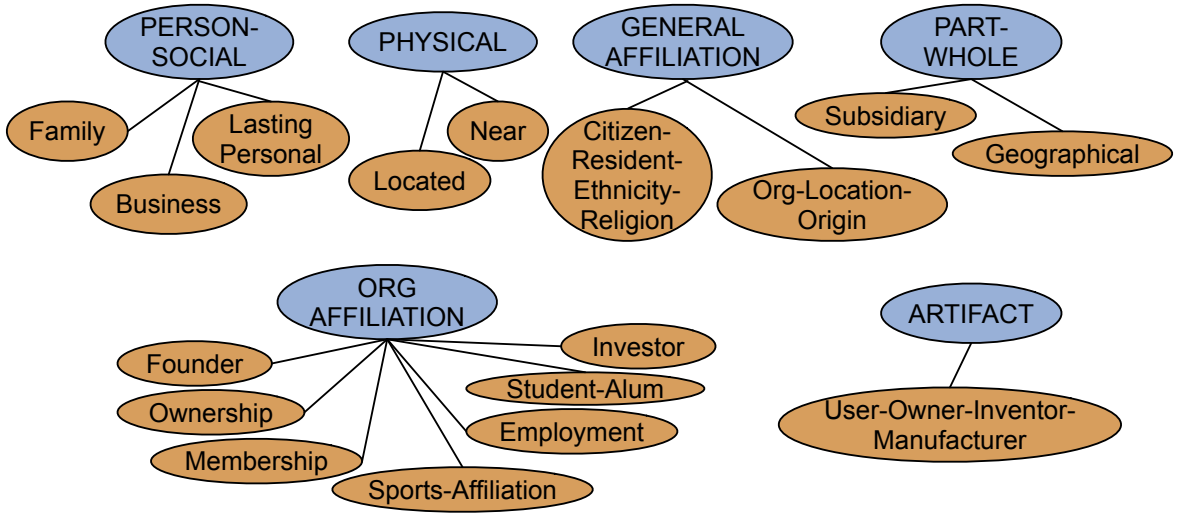
Other domains (e.g. biomedical text) require different types (proteins, genes, diseases, etc.)

Relation Extraction from text

Identify **relations between named entities**, typically from a small set of predefined relations.

Relations	Types	Examples
Physical-Located	PER-GPE	**He** was in **Tennessee**
Part-Whole-Subsidiary	ORG-ORG	**XYZ**, the parent company of **ABC**
Person-Social-Family	PER-PER	**Yoko**'s husband **John**
Org-AFF-Founder	PER-ORG	**Steve Jobs**, co-founder of **Apple**...

The 17 relations (orange) used in ACE:



Question Answering

Question Answering can mean different things:

— Being able to query a knowledge base
(e.g. a database of known facts) in natural language.
This may require a semantic parser
to translate the natural language question into, say, SQL

— Being able to query a collection of documents
that is known (or assumed) to contain answers
(as short text spans in these documents)

— Being able to answer questions about a single document
by returning short text spans in the document that answer
these questions ("reading comprehension")

— Being able to answer knowledge questions about a domain
(e.g. take multiple choice exams on science questions)

Reading: Chapter 25

Question Answering (QA) as an Information Retrieval (IR) task

Answer a user's questions by finding a text snippet in a large document collection that contains the answer.

Questions that can be answered in this way are typically about simple "factoids"

Question	Answer
Where is the Louvre Museum located?	in Paris, France
What's the abbreviation for limited partnership?	L.P.
What are the names of Odin's ravens?	Huginn and Muninn
What currency is used in China?	the yuan
What kind of nuts are used in marzipan?	almonds
What instrument does Max Roach play?	drums
What's the official language of Algeria?	Arabic
How many pounds are there in a stone?	14

Reading comprehension as span-extraction QA

Reading comprehension tests often ask children to answer questions based on a short paragraph.

Although reading comprehension can be formulated as a multiple-choice task, or a free answer task (which is difficult to evaluate), the span-extraction perspective requires that answers correspond to text spans

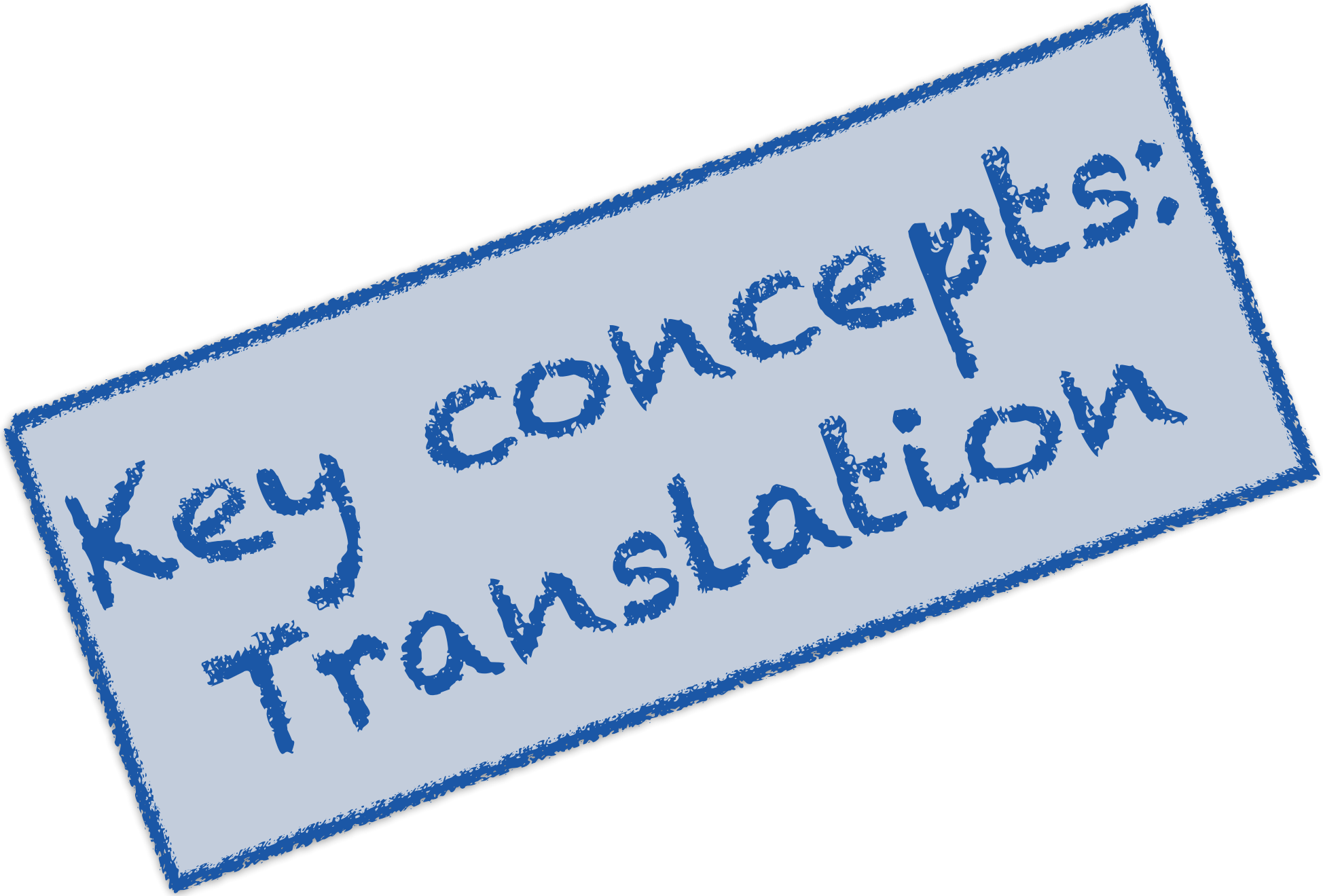
Science exams as testbed for QA

Task: Answer **multiple choice questions**
from 8th-grade science exams

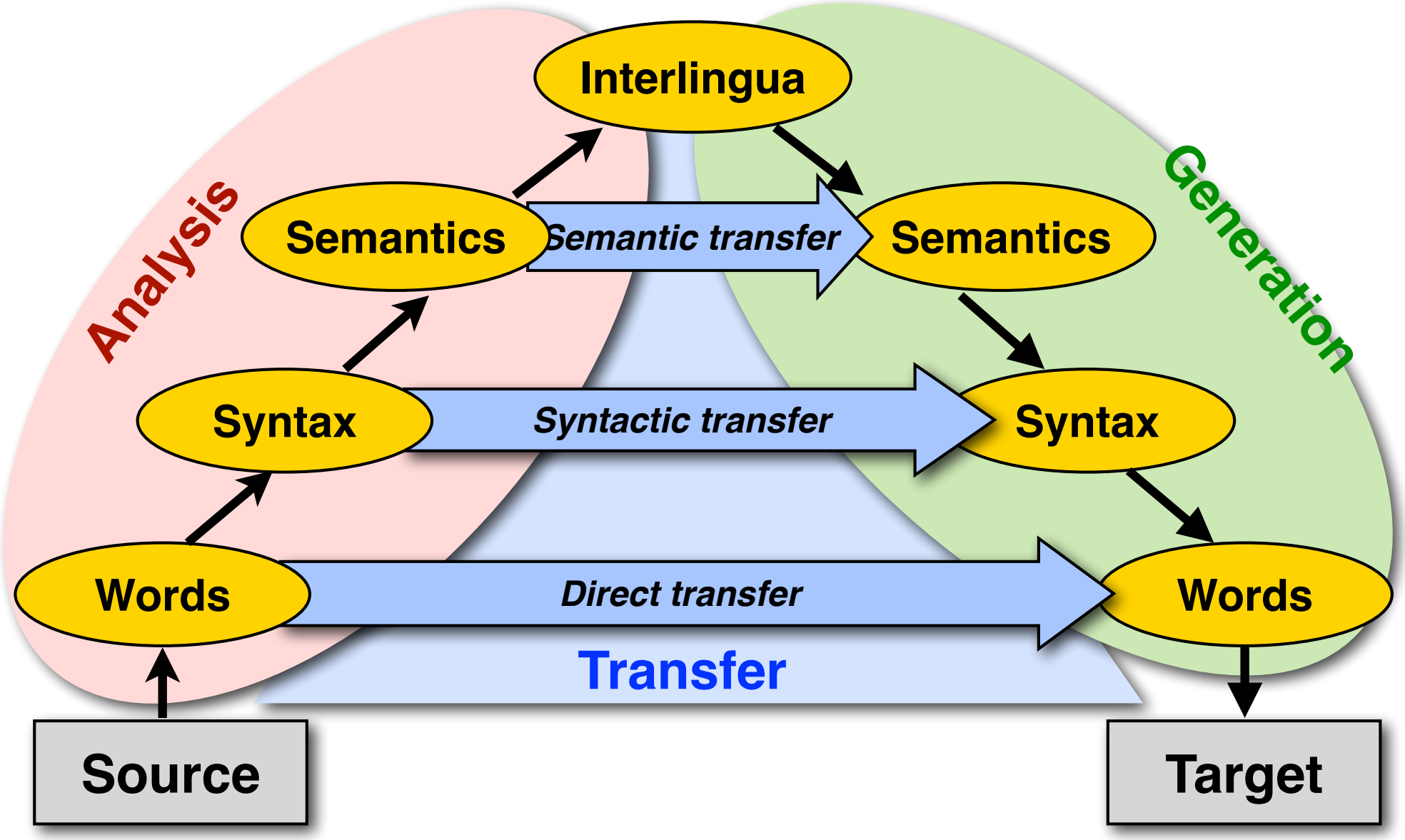
1. Which equipment will best separate a mixture of iron filings and black pepper?
(1) magnet (2) filter paper (3) triple-beam balance (4) voltmeter

This requires a lot of **background knowledge** that has to be acquired from somewhere (e.g. textbooks), and reasoning capabilities

https://allenai.org/content/docs/Aristo_Milestone.pdf


Key concepts:
Translation

The Vauquois triangle



Statistical Machine Translation

Given a Chinese input sentence (**source**)…

主席：各位議員，早晨。

…find the best English translation (**target**)

President: Good morning, Honourable Members.

We can formalize this as $\mathrm{T}^* = \mathrm{argmax}_{\mathrm{T}} P(\mathrm{T} \mid \mathrm{S})$

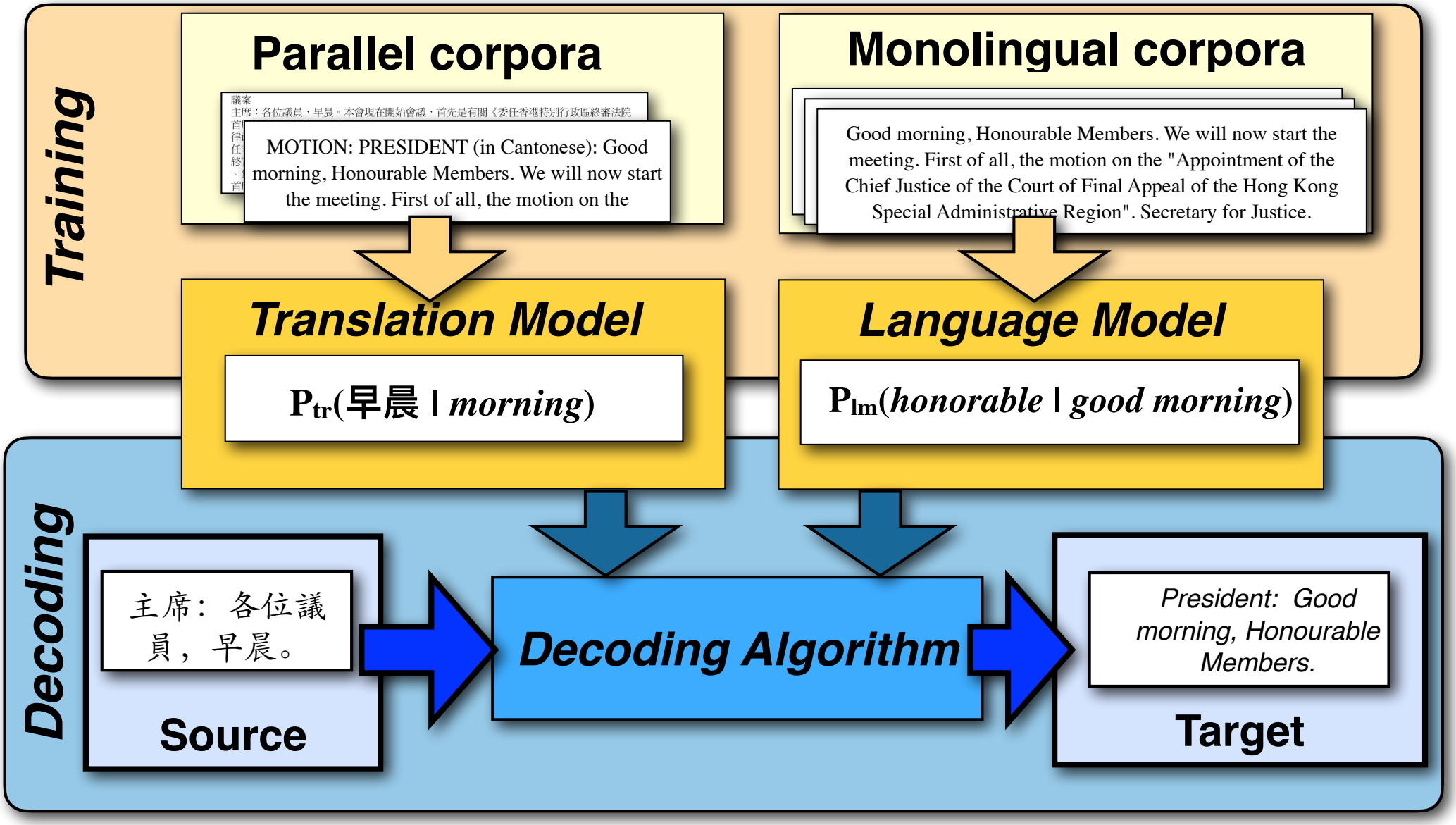
Using **Bayes Rule** simplifies the modeling task, so this was the first approach for statistical MT (the so-called "**noisy-channel model**"):

$$\mathrm{T}^* = \mathrm{argmax}_{\mathrm{T}} P(\mathrm{T} \mid \mathrm{S}) = \mathrm{argmax}_{\mathrm{T}} P(\mathrm{S} \mid \mathrm{T}) P(\mathrm{T})$$

where $P(\mathrm{S} \mid \mathrm{T})$: translation model

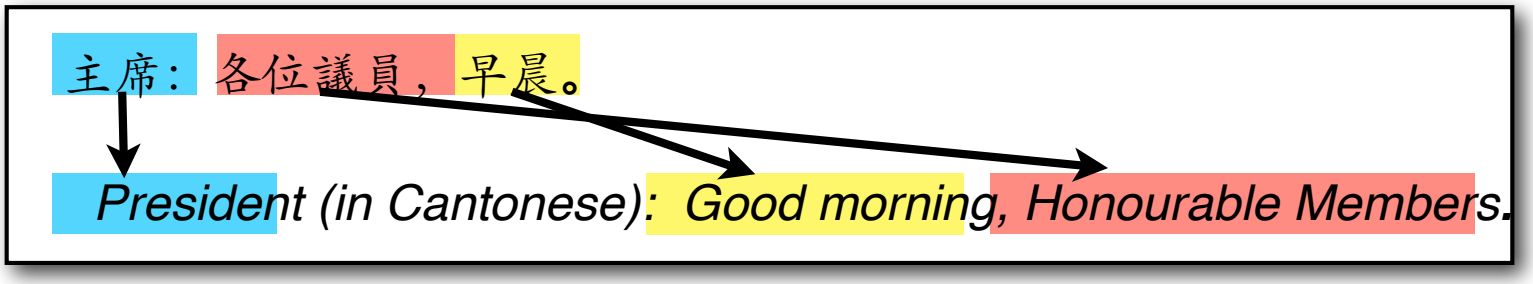
$P(\mathrm{T})$: language model

Statistical MT: Training and Decoding



Phrase-based translation models

Assumption: fundamental units of translation are **phrases**:



Phrase-based model of $P(F \mid E)$:

1. Split target sentence deterministically into phrases $ep_1 \ldots ep_n$
2. Translate each target phrase ep_i into source phrase fp_i with translation probability $\varphi(fp_i \mid ep_i)$
3. Reorder foreign phrases with distortion probability $d(a_i - b_{i-1}) = c^{|a_i - b_{i-1} - 1|}$

a_i = start position of source phrase generated by e_i

b_{i-1} = end position of source phrase generated by e_{i-1}