Natural Language Understanding

11

11 Natural Language Understanding

11.1 Linguistics

11.2 Syntax

11.3 Semantics

11.4 Language models

11.5 Neural language systems

11.6 Large language models

11.7 Natural language processing

11.8 Conversation AI$^+$

Linguistics

Natural language: any language that has evolved naturally in humans

Linguistics

- **Syntax**: form of sentences from words
- **Semantics**: meaning, and its relation to form
- **Pragmatics**: how context contributes to meaning

Natural language understanding (NLU) or natural language processing (NLP) (computational linguistics, psycholinguistics) concern with the interactions between computers and natural languages

- extracting meaningful information from natural language input
- producing natural language output

NLU vs. NLP

- Natural language is far away from understanding in linguistics and AI, especially in theory
- There are a lot of algorithms for processing natural language, especially in practice

A brief history of NLU#

1940-60s **Foundational Insights**
automaton, McCulloch-Pitts neuron
probabilistic or information-theoretic models
formal language theory (Chomsky, 1956)

1957–70 **The Two Camps**
symbolic and stochastic (parsing algorithms)
Bayesian method (text recognition)
the first online corpora (Brown corpus of English)

1970–83 **Four Paradigms**
stochastic paradigm: Hidden Markov Model
logic-based paradigm: Prolog (Definite Clause Grammars)
natural language understanding: SHRDLU (Winograd, 1972)
discourse modeling paradigm: speech acts, BDI

1983–93 **Empiricism and Finite State Models Redux**

A brief history of NLU

1994–99	**The Field Comes Together** probabilistic and data-driven models
2000–07	**The Rise of Machine Learning** big data (spoken and written) statistical learning Resurgence of probabilistic and decision-theoretic methods
2008–	Deep learning high-performance computing NLP as recognition
2015–	Pretrained language models
2017–	Transformer
2020–	Large language models (GPT3/ChatGPT/GPT4)

Communication

A language is a structured system of communication used by humans

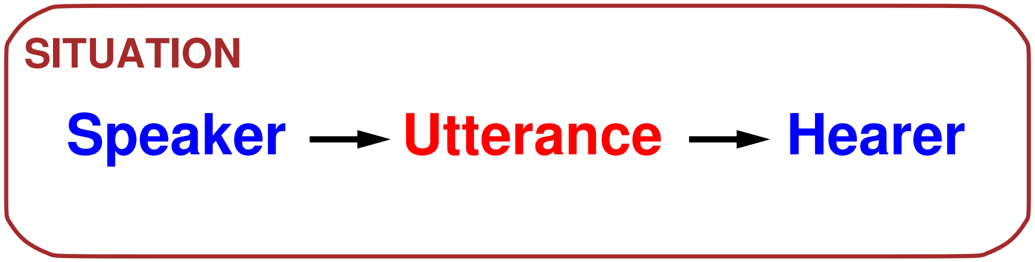
– spoken, written, and sign languages

Philosophy of language

- "Classical" view (pre-1953):
  language consists of sentences that are true/false (cf. logic)
- "Modern" view (post-1953):
  language is a form of action

Wittgenstein (1953), *Philosophical Investigations*
Austin (1962), *How to Do Things with Words*
Searle (1969), *Speech Acts*

Speech acts


SITUATION

Speaker → Utterance → Hearer

Speech acts achieve the speaker's goals:

Inform	"There's a pit in front of you"
Query	"Can you see the gold?"
Command	"Pick it up"
Promise	"I'll share the gold with you"
Acknowledge	"OK"

Speech act planning requires knowledge of

– Situation

– Semantic and syntactic conventions

– Hearer's goals, knowledge base, and rationality

Stages in communication (informing)

Intention	S wants to inform H that P
Generation	S selects words W to express P in context C
Synthesis	S utters words W
Perception	H perceives W' in context C'
Analysis	H infers possible meanings $P_1, \ldots P_n$
Disambiguation	H infers intended meaning P_i
Incorporation	H incorporates P_i into KB

How could this go wrong?

Stages in communication (informing)

Intention	S wants to inform H that P
Generation	S selects words W to express P in context C
Synthesis	S utters words W
Perception	H perceives W' in context C'
Analysis	H infers possible meanings $P_1, \ldots P_n$
Disambiguation	H infers intended meaning P_i
Incorporation	H incorporates P_i into KB

How could this go wrong?

- – Insincerity (S doesn't believe P)
- – Speech wreck ignition failure
- – Ambiguous utterance
- – Different understanding of current context ($C \neq C'$)

Knowledge in language

Engaging in complex language behavior requires various kinds of knowledge of the language

- Linguistic knowledge
  - Phonetics and phonology: the linguistic sounds
  - Morphology: the meaningful components of words
  - Syntax
  - Semantics
  - Pragmatics
  - Discourse: the linguistic units larger than a single utterance
- World knowledge/model: common knowledge, commonsense
  – language cannot be understood without the everyday knowledge that all speakers share about the world

Syntax

Vervet monkeys, antelopes etc. use isolated symbols for sentences
⇒ restricted set of communicable propositions, no generative capacity

Chomsky (1957): Syntactic Structures

Grammar is a set of rules that defines the compositional (tree) structure of allowable phrases
A language is the set of sentences that follow those rules
    e.g., speech (linear), text (linear), music (two-dimensional)

A formal language is a set of strings of terminal symbols
- – impossible to formalize a natural language
- – possible to study a natural language by a formal grammar

Each string in the language can be analyzed/generated by the (formal) grammar

Syntactic structures

The grammar is a set of rewrite rules (Chomsky normal form)

E.g.

$$S \rightarrow NP\ VP$$
$$Article \rightarrow \boldsymbol{the} \mid \boldsymbol{a} \mid \boldsymbol{an} \mid \ldots$$

Here S is the sentence symbol, NP and VP are nonterminals

– lexical rules

– syntactic rules

Grammar types

Regular: $nonterminal \rightarrow \boldsymbol{terminal}[nonterminal]$

$$S \rightarrow \boldsymbol{a}S$$
$$S \rightarrow \Lambda$$

Context-free: $nonterminal \rightarrow anything$

$$S \rightarrow \boldsymbol{a}S\boldsymbol{b}$$

Context-sensitive: more nonterminals on right-hand side

$$ASB \rightarrow AA\boldsymbol{a}BB$$

Recursively enumerable: no constraints

- Related to Post systems and Kleene systems of rewrite rules and can be used as computational models
- Natural languages are probably context-free, parsable in real-time

Example: Wumpus lexicon

$$
\begin{aligned}
\mathit{Noun} &\to \textbf{\textit{stench}} \mid \textbf{\textit{breeze}} \mid \textbf{\textit{glitter}} \mid \textbf{\textit{nothing}} \\
&\mid \textbf{\textit{wumpus}} \mid \textbf{\textit{pit}} \mid \textbf{\textit{pits}} \mid \textbf{\textit{gold}} \mid \textbf{\textit{east}} \mid \ldots \\
\mathit{Verb} &\to \textbf{\textit{is}} \mid \textbf{\textit{see}} \mid \textbf{\textit{smell}} \mid \textbf{\textit{shoot}} \mid \textbf{\textit{feel}} \mid \textbf{\textit{stinks}} \\
&\mid \textbf{\textit{go}} \mid \textbf{\textit{grab}} \mid \textbf{\textit{carry}} \mid \textbf{\textit{kill}} \mid \textbf{\textit{turn}} \mid \ldots \\
\mathit{Adjective} &\to \textbf{\textit{right}} \mid \textbf{\textit{left}} \mid \textbf{\textit{east}} \mid \textbf{\textit{south}} \mid \textbf{\textit{back}} \mid \textbf{\textit{smelly}} \mid \ldots \\
\mathit{Adverb} &\to \textbf{\textit{here}} \mid \textbf{\textit{there}} \mid \textbf{\textit{nearby}} \mid \textbf{\textit{ahead}} \\
&\mid \textbf{\textit{right}} \mid \textbf{\textit{left}} \mid \textbf{\textit{east}} \mid \textbf{\textit{south}} \mid \textbf{\textit{back}} \mid \ldots \\
\mathit{Pronoun} &\to \textbf{\textit{me}} \mid \textbf{\textit{you}} \mid \textbf{\textit{I}} \mid \textbf{\textit{it}} \mid \ldots \\
\mathit{Name} &\to \textbf{\textit{John}} \mid \textbf{\textit{Mary}} \mid \textbf{\textit{Beijing}} \mid \textbf{\textit{UCB}} \mid \textbf{\textit{PKU}} \mid \ldots \\
\mathit{Article} &\to \textbf{\textit{the}} \mid \textbf{\textit{a}} \mid \textbf{\textit{an}} \mid \ldots \\
\mathit{Preposition} &\to \textbf{\textit{to}} \mid \textbf{\textit{in}} \mid \textbf{\textit{on}} \mid \textbf{\textit{near}} \mid \ldots \\
\mathit{Conjunction} &\to \textbf{\textit{and}} \mid \textbf{\textit{or}} \mid \textbf{\textit{but}} \mid \ldots \\
\mathit{Digit} &\to \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \mathbf{4} \mid \mathbf{5} \mid \mathbf{6} \mid \mathbf{7} \mid \mathbf{8} \mid \mathbf{9}
\end{aligned}
$$

Wumpus lexicon

$$
\begin{array}{rcl}
Noun & \rightarrow & \boldsymbol{stench} \mid \boldsymbol{breeze} \mid \boldsymbol{glitter} \mid \boldsymbol{nothing} \\
 & & \mid \boldsymbol{wumpus} \mid \boldsymbol{pit} \mid \boldsymbol{pits} \mid \boldsymbol{gold} \mid \boldsymbol{east} \mid \ldots \\
Verb & \rightarrow & \boldsymbol{is} \mid \boldsymbol{see} \mid \boldsymbol{smell} \mid \boldsymbol{shoot} \mid \boldsymbol{feel} \mid \boldsymbol{stinks} \\
 & & \mid \boldsymbol{go} \mid \boldsymbol{grab} \mid \boldsymbol{carry} \mid \boldsymbol{kill} \mid \boldsymbol{turn} \mid \ldots \\
Adjective & \rightarrow & \boldsymbol{right} \mid \boldsymbol{left} \mid \boldsymbol{east} \mid \boldsymbol{south} \mid \boldsymbol{back} \mid \boldsymbol{smelly} \mid \ldots \\
Adverb & \rightarrow & \boldsymbol{here} \mid \boldsymbol{there} \mid \boldsymbol{nearby} \mid \boldsymbol{ahead} \\
 & & \mid \boldsymbol{right} \mid \boldsymbol{left} \mid \boldsymbol{east} \mid \boldsymbol{south} \mid \boldsymbol{back} \mid \ldots \\
Pronoun & \rightarrow & \boldsymbol{me} \mid \boldsymbol{you} \mid \boldsymbol{I} \mid \boldsymbol{it} \mid \boldsymbol{S/HE} \mid \boldsymbol{Y'ALL} \ldots \\
Name & \rightarrow & \boldsymbol{John} \mid \boldsymbol{Mary} \mid \boldsymbol{Boston} \mid \boldsymbol{UCB} \mid \boldsymbol{PAJC} \mid \ldots \\
Article & \rightarrow & \boldsymbol{the} \mid \boldsymbol{a} \mid \boldsymbol{an} \mid \ldots \\
Preposition & \rightarrow & \boldsymbol{to} \mid \boldsymbol{in} \mid \boldsymbol{on} \mid \boldsymbol{near} \mid \ldots \\
Conjunction & \rightarrow & \boldsymbol{and} \mid \boldsymbol{or} \mid \boldsymbol{but} \mid \ldots \\
Digit & \rightarrow & \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \mathbf{4} \mid \mathbf{5} \mid \mathbf{6} \mid \mathbf{7} \mid \mathbf{8} \mid \mathbf{9}
\end{array}
$$

Example: Wumpus grammar

S	$\rightarrow$	$NP\ VP$	I + feel a breeze
	$\vert$	$S\ Conjunction\ S$	I feel a breeze + and + I smell a wumpus
NP	$\rightarrow$	$Pronoun$	I
	$\vert$	$Noun$	pits
	$\vert$	$Article\ Noun$	the + wumpus
	$\vert$	$Digit\ Digit$	3 4
	$\vert$	$NP\ PP$	the wumpus + to the east
	$\vert$	$NP\ RelClause$	the wumpus + that is smelly
VP	$\rightarrow$	$Verb$	stinks
	$\vert$	$VP\ NP$	feel + a breeze
	$\vert$	$VP\ Adjective$	is + smelly
	$\vert$	$VP\ PP$	turn + to the east
	$\vert$	$VP\ Adverb$	go + ahead
PP	$\rightarrow$	$Preposition\ NP$	to + the east
$RelClause$	$\rightarrow$	$\boldsymbol{that}\ VP$	that + is smelly

Probabilistic grammar#

Probabilistic context-free grammar (PCFG): the grammar assigns a probability to every string

$$\begin{aligned} VP \rightarrow\ & Verb[0.70] \\ \mid\ & VP\ NP[0.03] \end{aligned}$$

With probability 0.70 a verb phrase consists solely of a verb and with probability 0.30 it is a VP followed by an NP

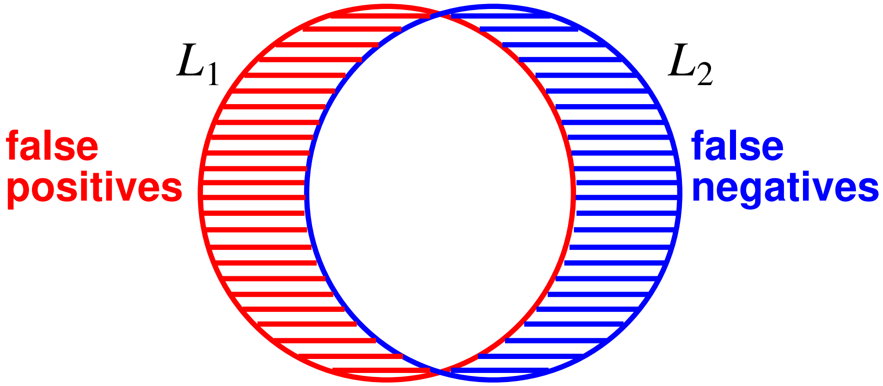
Also, assign a probability to every word (lexicon)

Problems

- Overgenerate: generate sentences that are not grammatical
  e.g., "Me go I."
- Undergenerate: reject correct sentences
  e.g., "I think the wumpus is smelly."

Grammaticality judgements

Formal language L_1 may differ from natural language L_2



Adjusting L_1 to agree with L_2 is a <u>learning</u> problem

* the gold grab the wumpus
* I smell the wumpus the gold
I give the wumpus the gold

Intersubjective agreement reliable, independent of semantics
Real grammars 10–500 pages, insufficient even for “proper” English

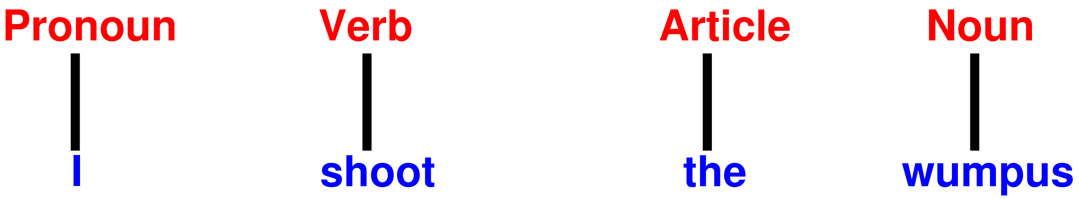
Syntactic analysis

Exhibit the grammatical structure of a sentence

I shoot the wumpus

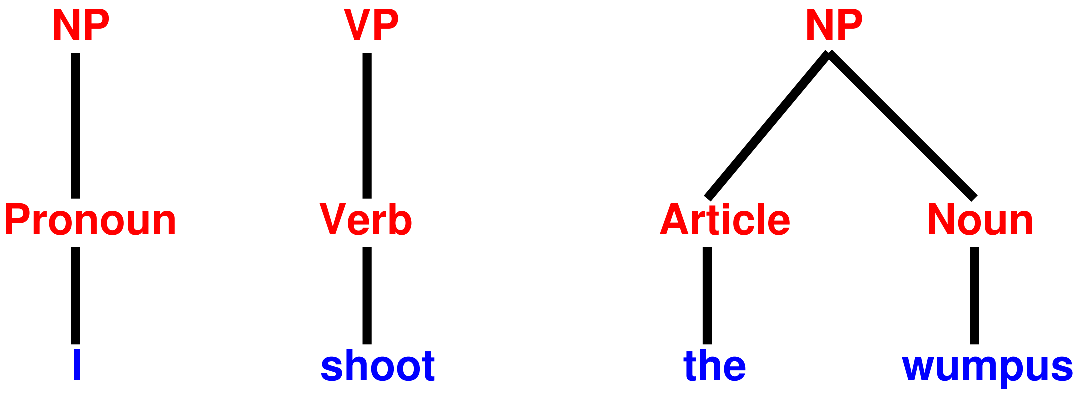
Parse trees

Exhibit the grammatical structure of a sentence



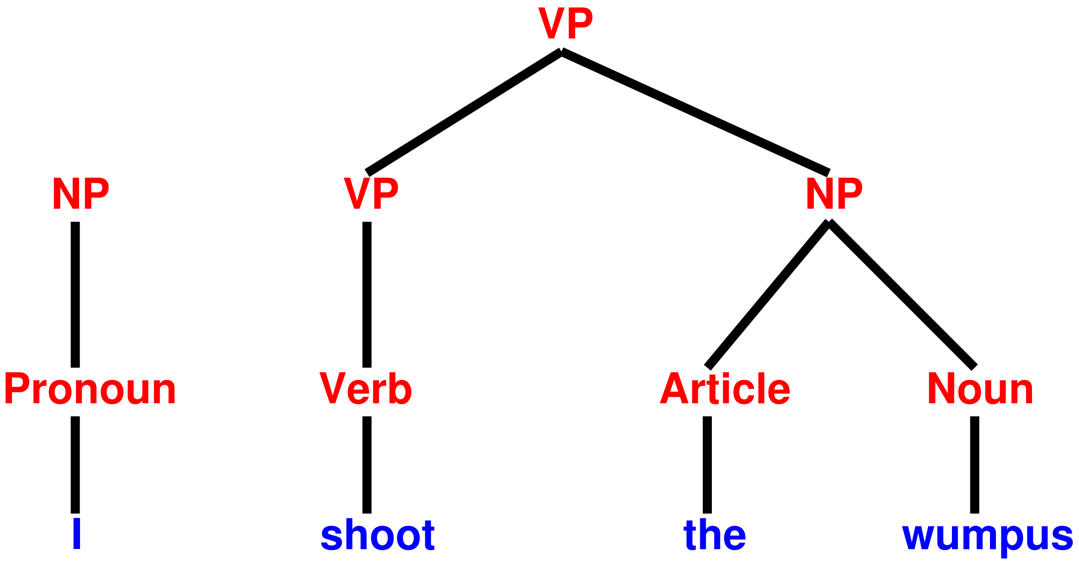
Parse trees

Exhibit the grammatical structure of a sentence



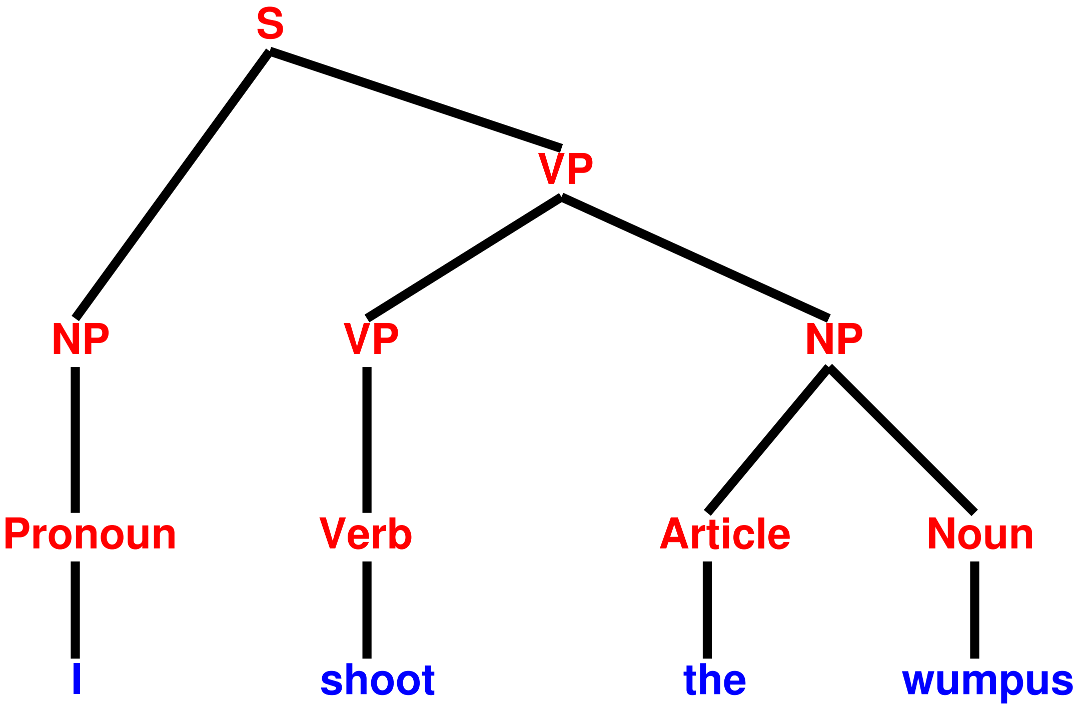
Parse trees

Exhibit the grammatical structure of a sentence



Parse trees

Exhibit the grammatical structure of a sentence



Parsing

Bottom-up: replacing any substring that matches RHS of a rule with the rule's LHS

```
def BottomUpParse(words, grammar)

    forest ← words
    loop do
        if Length(forest) = 1 and Category(forest[1]) = Start(grammar) then
            return forest[1]
        else
            i ← choose from {1...Length(forest)}
            rule ← choose from Rules(grammar)
            n ← Length(Rule-RHS(rule))
            subsequence ← Subsequence(forest, i, i+n-1)
            if Match(subsequence, Rule-RHS(rule)) then
                forest[i...i+n-1] ← [Make-Node(Rule-LHS(rule), subsequence)]
            else failure
    return a parse tree
```

Chart parser$^{\#}$

E.g.
"Have the students in AI course ***take*** *the exam."*
"Have the students in AI course ***taken*** *the exam?"*

Problem: Parsing won't be able to tell if a word is correct until the later word and will have to backtrack all the way to the first word

Chart parsers: to avoid the inefficiency of repeated parsing, every time we analyze a substring, store the results in a data structure known as a chart, so we won't have to reanalyze it later
$\Leftarrow$ dynamic programming

CYK algorithm
– a Chart parser
– bottom-up PCFG

Context-free parsing

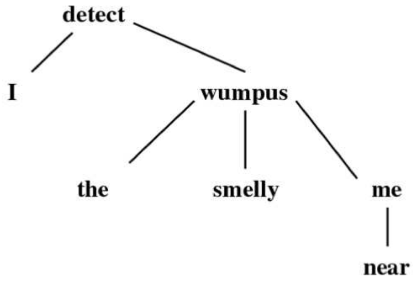
Efficient algorithms (e.g., CYK algorithm) $O(n^2m)$ for context-free
(time, n – number of words, m – number of nonterminal symbols)
run at several thousand words/sentences for real grammar

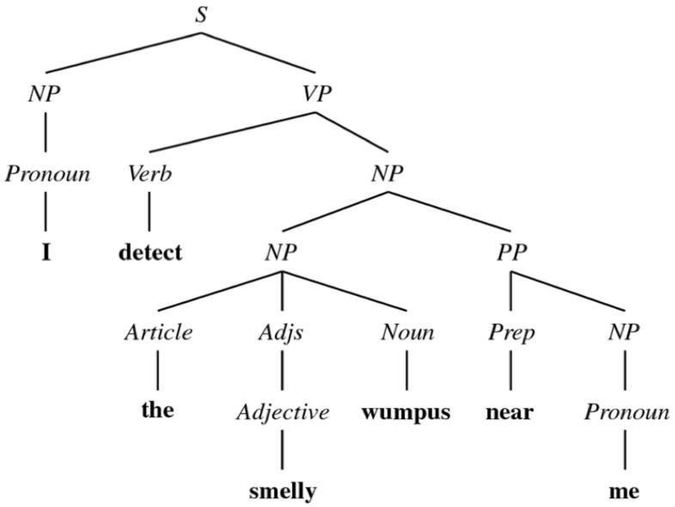
Context-free parsing $\equiv$ Boolean matrix multiplication

Can be improved by search, say A*, beam search etc.
$\Rightarrow$ is it possible $O(n)$?

Dependency grammars*

Syntactic structure is formed by binary relations between lexical items, without a need for syntactic constituents





(Left) dependency parsing (Right) phrase structure parsing

More suitable to data-oriented parsing by machine learning

Augmented grammars*

The nonterminals are not just atomic symbols but are structured representations

E.g., "I": $NP(Sbj, 1S, Speaker)$

"a noun phrase that is in the subjective case, first person singular, and whose meaning is the speaker of the sentence"

"Me": $NP(Obj, 1S, Speaker)$

Subcategory: a category has been augmented with features

E.g., $Pronoun$ augmented with "subjective case, first person singular"

Head: a word is the most important in a phrase or a sentence

E.g., "banana" is the head of the "a banana"

"ate" is the head of the "ate a banana"

$VP(v)$: a phrase with category VP whose headword is v

Logical grammars

BNF notation for grammar too restrictive
- difficult to add "side conditions" (number agreement, etc.)
- difficult to connect syntax to semantics

Idea: express grammar rules in logic

$X \rightarrow YZ$ becomes $Y(s_1) \land Z(s_2) \Rightarrow X(Append(s_1, s_2))$
$X \rightarrow \boldsymbol{word}$ becomes $X([\text{“}\boldsymbol{word}\text{”}])$
$X \rightarrow Y \mid Z$ becomes $Y(s) \Rightarrow X(s) \quad Z(s) \Rightarrow X(s)$

Here, $X(s)$ means that string s can be interpreted as an X

Logical grammars

It's easy to augment the rules

$$\begin{aligned} NP(s_1) \;\wedge\; & EatsBreakfast(Ref(s_1)) \wedge VP(s_2) \\ & \Rightarrow NP(Append(s_1, [\text{“}\boldsymbol{who}\text{”}], s_2)) \end{aligned}$$

$$\begin{aligned} NP(s_1) \;\wedge\; & Number(s_1, n) \wedge VP(s_2) \wedge Number(s_2, n) \\ & \Rightarrow S(Append(s_1, s_2)) \end{aligned}$$

Parsing is reduced to logical inference

$\textsc{Ask}(KB, S([\text{“}\boldsymbol{I}\text{”}\ \text{“}\boldsymbol{am}\text{”}\ \text{“}\boldsymbol{a}\text{”}\ \text{“}\boldsymbol{wumpus}\text{”}]))$

Can add extra arguments to return the parse structure

– logical semantics

Logical grammars

Generation simply requires a query with uninstantiated variables

$\textsc{Ask}(KB,\ S(x))$

If we add arguments to nonterminals to construct sentence semantics, NLP generation can be done from a given logical sentence

$\textsc{Ask}(KB,\ S(x, At(Robot, [1,1]))$

Complications of natural language

Real human languages provide many problems for NLU/NLP

- ambiguity
- anaphora
- indexicality
- vagueness
- discourse structure
- metonymy
- metaphor
- noncompositionality etc.

Ambiguity

Ambiguity at all levels

- Lexical

“You **held** your breath and the door for me”

- Syntactic

“Put the book in the box on the table”

[**the book**] in the box

[**the book in the box**]

- Semantic: sentence can have more than one meaning

“Alice wants a dog like Bob’s”

- Pragmatic

“Alice: Do you know who’s going to the party?
Bob: Who?”

Disambiguation: recovering the most probable intended meaning of an utterance

Example: ambiguity

Garden-path sentence (Bever 1970): a grammatically correct but most likely interpreting the incorrect sentence

“The old man the boat.”
[The old are the persons who man the boat.]

Try this sentence
“The complex houses married and single soldiers and their families.”

“研究鲁迅的文章”
[“我研究鲁迅的文章。”
“我研究鲁迅的文章发表不了。”]

Example: ambiguity

Most view syntactic structure as an essential step towards meaning

e.g., “Mary hit John” $\neq$ “John hit Mary”

Fish-and-Chip

“Wouldn’t the sentence ’I want to put a hyphen between the words Fish and And and And and Chips in my Fish-And-Chips sign’ have been clearer if quotation marks had been placed before Fish, and between Fish and and, and and and And, and And and and, and and and And, and And and and, and and and Chips, as well as after Chips?”

Example: ambiguity in speech

施氏食狮史 (赵元任)

“石室诗士施氏，嗜狮，誓食十狮。施氏时时适市视狮。十时，适十狮适市。是时，适施氏适市。施氏视是十狮，恃矢势，使是十狮逝世。氏拾是十狮尸，适石室。石室湿，氏使侍拭石室。石室拭，施氏始试食是十狮尸。食时，始识是十狮尸，实十石狮尸。试释是事。”

ChatGPT: translating classical Chinese$^{\#}$

石室里的诗人施氏，痴迷于狮子，发誓要吃十只狮子。施氏时常去市场观察狮子。十点钟时，他带着十只狮子去市场。就在这时，他和十只狮子去了市场。施氏看到了这十只狮子，依仗自己的力量，使这十只狮子死去。施氏拾起了这十只狮子的尸体，回到了石室。石室很湿，施氏叫人把石室擦干净。石室擦干后，施氏开始尝试吃这十只狮子的尸体。在吃的时候，他才意识到这十只狮子的尸体实际上是十块石头。他试图解释这件事。

Semantics

Formal semantics

– logical grammars = logical semantics

Computatiolnal semantics

– Distributional semantics
foundation of deep learning for NLP

Formal semantics

Montague grammar → formal semantics for English

R. Montague, *English as a Formal Language*, 1970
(Formal Philosophy, 1974)
I. Heim and A. Kratzer, *Semantics in Generative Grammar*, 1998
C. Potts, Logic of Conventional Implicatures, 2005

– Chomsky: Minimalist Program
– Discourse Representation Theory
– Situation Semantics/Situation Theory
– Game-theoretic Semantics

Distributional semantics

Context is essential in the similarity of the words

Distributional hypothesis: the link between similarity in how words are distributed and similarity in what they mean

– a word's distribution is the set of contexts

– two words that occur in very similar distributions (that occur together with very similar words) are likely to have the same meaning

Distributional semantics (vector semantics): instantiating the distributional hypothesis by automatically learning representations of the meaning of words directly from their distributions in texts in unsupervised ways

Embeddings

Word embedding: the meaning of a word can be defined as a vector (or a list of numbers, a point) in low dimensional space

– based in some way on counts of neighboring words

– embedding: assign the vectors for representing words in a vector space

One-shot vector: the i-th word in the dictionary V with a 1 bit in the i-th input position and a 0 in all the other positions

– can't capture the similarity between words

Consider a vector of N neighboring word (N-gram, see later)

– with a $100,000$-word vocabulary, there are 10^{25} 5-grams (although vectors in this 10^{25}-dimensional space would (sparse)

A word embedding: a low-dimensional vector representing a word

$\Leftarrow$ learning automatically from the data

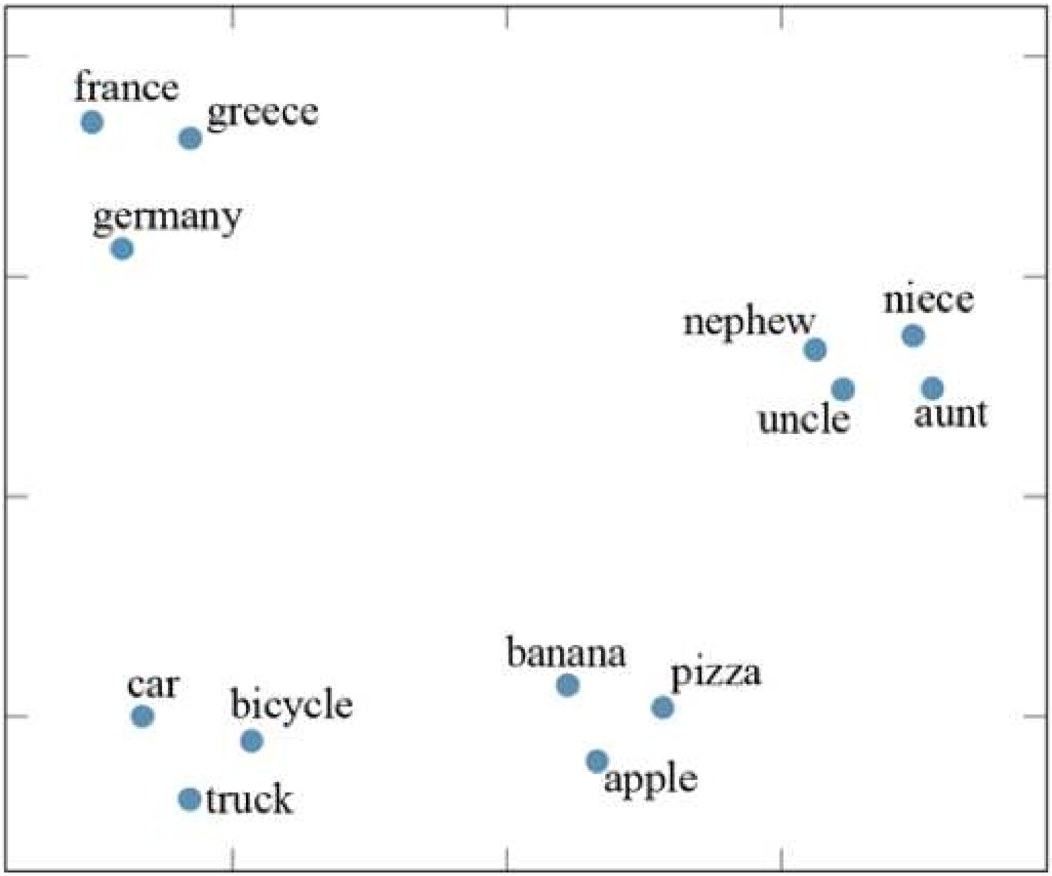
Example: embeddings

Each vector doesn't have "meanings"

$$\begin{aligned}
\text{“aardvark”} &= [-0.7, +0.2, -3.2, \cdots] \\
\text{“abacus”} &= [+0.5, +0.9, -1.3, \cdots] \\
&\vdots \\
\text{“zyzzyva”} &= [-0.1, +0.8, -0.4, \cdots]
\end{aligned}$$

Example: embeddings

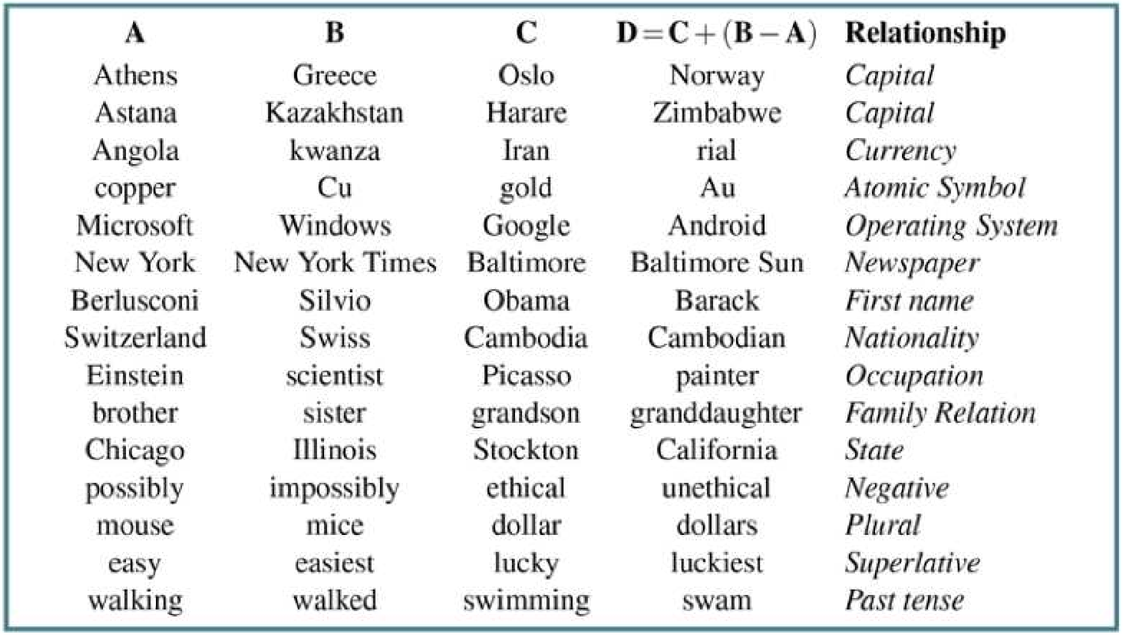
The vector (feature) space: similar words having similar vectors



Example: embeddings

Word analogy problems: assume $\mathbf{B} - \mathbf{A} = \mathbf{D} - \mathbf{C}$

“Athens is to Greece as Oslo is to [what]?”



A	B	C	$\mathbf{D} = \mathbf{C} + (\mathbf{B} - \mathbf{A})$	Relationship
Athens	Greece	Oslo	Norway	*Capital*
Astana	Kazakhstan	Harare	Zimbabwe	*Capital*
Angola	kwanza	Iran	rial	*Currency*
copper	Cu	gold	Au	*Atomic Symbol*
Microsoft	Windows	Google	Android	*Operating System*
New York	New York Times	Baltimore	Baltimore Sun	*Newspaper*
Berlusconi	Silvio	Obama	Barack	*First name*
Switzerland	Swiss	Cambodia	Cambodian	*Nationality*
Einstein	scientist	Picasso	painter	*Occupation*
brother	sister	grandson	granddaughter	*Family Relation*
Chicago	Illinois	Stockton	California	*State*
possibly	impossibly	ethical	unethical	*Negative*
mouse	mice	dollar	dollars	*Plural*
easy	easiest	lucky	luckiest	*Superlative*
walking	walked	swimming	swam	*Past tense*

Contextualized word embeddings (by pretraining, see later) can be used for downstream NLP tasks

Word embeddings

Let V be a vocabulary. A mapping f from any $w \in V$ to a real vector $f(w) = e \in \mathbb{R}^D$, i.e., $V \to \mathbb{R}^D$

– D is a hyperparameter of the dimension

– f is usually represented by a $|V| \times D$ matrix of free parameters and shared across all the words in the context

– representing the distributed feature vectors associated with each word in the vocabulary

Vector semantic models are extremely practical because they can be learned automatically from text without any labeling or supervision

Language models

By a language $\mathcal{L}$ we mean a natural language that is a set of sequences of discrete tokens, or any set of sequences of discrete things that can be discretized and encoded as tokens

– e.g., the dialects (including internet slang), the ancient languages, the sign language, the lip language, the cortical activity, the musical languages, the animal songs, mathematical (and scientific expressions), the signals and the images (representing images by language) and games, gesture and feeling (such as emotion) and even cognitive behaviour (such as gaze)

– Artificial languages, such as programming languages, are formal languages, but the source code of a programming language is a special kind of language

Vocabularies

Let V be the vocabulary of $\mathcal{L}$ that is a large but finite set

– $w_n \in V^*$ (finite strings of elements in V) is a token, $n \in \mathbb{N}$

– $\boldsymbol{w} = (w_1, w_2 \cdots, w_N)$ is a sequence (finite string of tokens), where N is the length of sequence

Write $\mathcal{L} = \{\boldsymbol{w} \mid \boldsymbol{w}$ is a sequence$\}$, that is a finite set of sequences

A discourse $\mathcal{D} \subset \mathcal{L}$ is a subset of $\mathcal{L}$

– can be made as a single sequence by concatenating all the sequences in the discourse

Tokenization

A token may be a character, a sub-word a word, or even a sequence that is either a sentence or a text

– without loss of generality, usually mention the level of the tokens as words and the sequences as sentences

Tokenization: the process of dividing a text into a sequence of token

Token IDs: A piece of text is represented as a sequence of indices, corresponding to its (sub)words, preceded by bos_token and followed by eos_token

Corpora

A corpus is a body of text

Considered a corpus of sequences (sentences) of tokens that is a large but finite set

$$W = \{\boldsymbol{w}^{(1)}, \boldsymbol{w}^{(2)}, \ldots, \boldsymbol{w}^{(M)}\}$$

– $\boldsymbol{w}^{(m)} = \left(w_{m_1}, w_{m_2} \cdots, w_{m_N}\right) \in V^*$ is a (finite) sequence of tokens $w_n \in V$

– independent and identically distributed (*iid*) data, i.e. sampled iid from some distribution P over V^*

– used as the training set in machine learning

Wordnet: dictionary of about 100,000 words and phrases

– parts of speech, semantic relations (synonym, antonym)

Penn Treebank: parse trees for a 3-million-word corpus (English)

The British National Corpus: 100 million words

Web: trillion words (say, Wikipedia)

Statistical language models

(Statistical) language model (LM): modeling a natural language as a probability distribution over sentences and possible meaning

– assign probabilities to sequences of tokens

$$\begin{aligned} P(\boldsymbol{w}) &= P(w_{1:N}) \\ &= P(w_1) P(w_2 \mid w_1) P(w_3 \mid w_{1:2}) \cdots P(w_n \mid w_{1:N-1}) \\ &= \prod_{n=1}^{N} P(w_n \mid w_{<n}) \\ &= \sum_{n=1}^{N} \log P(w_n \mid w_{<n}) \end{aligned}$$

– the sub-sequence $w_{i \le n \le j} = w_{i:j} = (w_i, w_{i+1} \cdots, w_{j-1}, w_j)$ is the context of the neighboring tokens or grammatical environments

Statistical language models

- The chain rule (factorization) that indicates the link between the joint probability of a sequence and the conditional probability of a token given previous tokens
- $P\left(w_k \mid w_{<k}\right)$ <u>predicts</u> (generates) the next word given previous words — forward LM
- Similarly, — backward LM

$$P\left(\boldsymbol{w}\right)=P\left(w_{1:N}\right)\ =\sum_{n=1}^{N}\log P\left(w_n \mid w_{>n}\right) \tag{1}$$

Bi-directional LM $=$ Forward LM $+$ Backward LM

$\Leftarrow$ generative and autoregressive models

N-gram model

N-gram (letters or units) model $P(c_{1:N})$: probability distribution of n-letter (or word) sequences, defined as Markov chain of order $n-1$
Say, a trigram (3-gram) model

$$P(c_i|c_{1:i-1}) = P(c_i|c_{i-2:i-1})$$

In a language with 100 characters, the distribution has a million entries, and can be accurately estimated by counting character sequences in a corpus with 10 million characters
With a vocabulary of 10^5 words, there are 10^{15} trigram probabilities to estimate

e.g., books.google.com/ngram

Skip-gram model: counting words that are near each other, but skip N words between them

Neural language models

Neural LMs (NLM) implement LMs by neural networks (NNs) that learn to predict the next word from prior words

$$P(w_{1:N}) = \sum_{n=1}^{N} \log P(w_n \mid w_{<n}; f_\theta)$$

- The conditional probabilities are computed by an NN, which is a function f with parameters θ, written as f_θ
- f_θ may be implemented by any deep NN, such as FFN (MLP), RNN etc.
  - FFN: a window of N words with many parameters $O(N)$ (N-gram $O(v^N)$), difficult due to the context, relearning the same word at different positions (parameter)

RNN LM: RNNs-based LM is suitable, one word at a time with a parameter $O(1)$, the parameter is the same for every word positions

Neural language models

Write $f_\theta(h_t) = f_\theta(h_t, w_t, \boldsymbol{v}_t)$ to indicate the t-th hidden layer of f_θ with the deepness T ($t \in \mathbb{N}$)

– $(h_{t-1}(w_1), \cdots, h_{t-1}(w_{t-1}))$ are the input token vectors from the previous layer

– $\boldsymbol{v}_t = h_t(w_t)$ is the output vector

— at every time-step, updating the internal hidden state h_{t-2}, which summarizes $(w_1, \cdots, w_{t-2})$, with a new token w_{t-1}, resulting in h_{t-1}

— the resulting hidden state h_{t-1} is used to compute $P(w_t \mid w_1, \cdots, w_{t-1})$

— the output of f_θ is a vector whose the t-th element $\boldsymbol{v}_t$ estimates the probability $P\left(w_t = t \mid w_{<t}\right)$

The initial h_0 at the start is generally an arbitrary constant vector. The readout function is generally a softmax layer

Pretrained language models

Pretrained LMs (PLMs): given an unsupervised corpus

$$W = \{\boldsymbol{w}^{(1)}, \boldsymbol{w}^{(2)}, \ldots, \boldsymbol{w}^{(M)}\}$$

Use a standard objective to maximize the (log) likelihood

$$L(W) = \frac{1}{M} \sum_{m} \log P \left(\boldsymbol{w}^{(m)} \mid \boldsymbol{w}^{(m-1)}, \cdots, \boldsymbol{w}^{(m-c+1)}; g_\theta \right)$$

– c is the size of the context in which it occurs
– PLM can be pretrained from scratch (raw texts) in self-supervised learning

Pretrain: self-supervised learning

Self-supervised learning is the unsupervised learning by unlabelled data in a supervised learning manner

that predicts only a subset of information using the rest

– it usually doesn't care about the final performance of the task, rather learns intermediate representation with the expectation that the representation can carry good semantic or structural meanings and can be beneficial to a variety of downstream tasks

PLMs that are language "understanding" can be applied for downstream NLP tasks by finetuning

Pretraining: self-supervised learning

Pretraining algorithm

1. **Initialization**: Initialize LM parameters either randomly or with pre-trained weights

2. **Training loop**:

    2.1 Iterate over the training data for a certain number of epochs

    2.2 Divide the corpus into batches and tokenize the input text

    2.3 Forward pass: Pass the tokenized input through the neural network layers to compute logits

    2.4 Calculate the loss between predicted logits and actual targets

    2.5 Backward pass: Compute gradients of the loss concerning model parameters

    2.6 Update parameters using gradient descent

3. **Evaluation** (optional): Optionally, evaluate the model on a validation set to monitor performance

4. **Return**: trained LM after the specified number of epochs

Finetuning: supervised learning

Finetuning: adjusting the parameters of a pretrained LM on a specific dataset or task to improve its performance ⇒ transfer learning

– the knowledge gained from pre-training on a large corpus is leveraged to adapt the model to a more specific task

Finetuning algorithm

1. **Select** a target dataset/task: a specific task
2. **Adapt** the model architecture: modify the LM architecture if necessary to suit the target task
3. **Initialize** from pretrained weights: initialize the parameters of the model using the pretrained weights
4. **Train** on target task data: finetune the model on the target dataset by updating its parameters through BP by a loss function
5. **Monitor** performance: on a validation set
6. **Iterate**: finetune the model iteratively until satisfactory performance is achieved on the target task

Prompting: in-context learning

In-context learning: dynamically adapt and personalize models based on specific user interactions or contexts

Prompting: providing specific instructions or cues to guide the generation of text

– a way to steer the model's output toward a desired direction

Both finetuning and prompting are used to adapt PLMs to specific tasks, but they differ in their approach and purpose

– Approach: finetuning involves updating/modifying the parameters of a PLM using supervised learning; prompting does not involve updating the model's parameters but instead influences the output

– Purpose: fine-tuning is to adapt the PLM to perform well on a specific task; prompting is to control and influence the generation of text from a PLM

Masked language model*

MLM are trained by masking (hiding) individual words in the input and asking the model to predict the masked word

– considered to enhance the similarity of the distributed feature vectors from the mask information

$$P\left(w_{1:N}\right)=\sum_{n=1}^{N}\log P\left(w_n \mid \tilde{\boldsymbol{w}}; f_\theta(\overleftrightarrow{h}_t)\right)$$

– $f_\theta(\overleftrightarrow{h}_t)$: the NN for bidirectional LM – $\tilde{\boldsymbol{w}}$ is the partially masked version of the initial input $\boldsymbol{w}$ – a mask sequence is the sub-sequence

$w_{i:j/a:b}=(w_i, w_{i+1}, \cdots, \tilde{w}_a, \tilde{w}_{a+1}, \cdots, \tilde{w}_{b-1}, \tilde{w}_b, \cdots, w_{j-1}, w_j)$

which masks the tokens $w_a, w_{a+1}, \cdots, w_{b-1}, w_b$ in $w_{i:j}$

where $\tilde{w}_k$s are the mask sign depicted as blank (or some special symbol)

Pretrained word embeddings

Embedding (representation): representing knowledge (say words) into low-dimensional (continuous) vector spaces

Pretrained word embeddings: word embeddings are pretrained in a self-supervised way over a large corpus of text

Contextualized word embeddings: a model is pretrained to generate contextual representations of each word in a sentence, instead of just learning a word-to-embedding table

– mapping both a word and the surrounding context of words into a word embedding vector

Neural language systems

Neural language systems: neural models (deep learning) for NLP

- Encoder-decoder framework
- Transformer model

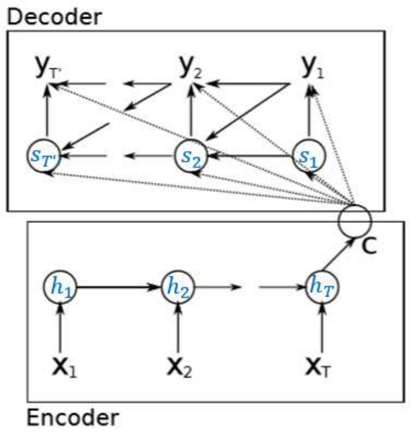
Encoder-decoder framework

Encoder-decoder (E-D) consists of, considering RNN

– an encoder (reader, input) RNN processes the input sequence $\boldsymbol{x}$

– A decoder (writer, output) RNN is conditioned on that fixed-length vector to generate the output sequence $\boldsymbol{y}$

The last state of the encoder RNN is used as a representation (context) $\boldsymbol{c}$ of the input sequence that is provided as input to the decoder RNN



Sequence-sequence models

Sequence-sequence (seq-seq) models are the E-Ds that are trained jointly to maximize the average of the likelihood over all the pairs of $\boldsymbol{x}$ and $\boldsymbol{y}$ sequences in the training set

– maybe text-text (discourse-discourse, document-document)

– is a natural choice of the E-Ds, i.e., using the same model, objective, training procedure and decoding process for every NLP task

E.g., machine translation (MT) is to translate a sentence from a source language to a target language

Usually, the decoder in E-Ds generates a sequence by greedy decoding, such as beam search

Sequence-sequence formulations

Input sequence: $\boldsymbol{x} = \{x_1, x_2, \cdots, x_T\}$
Output sequence: $\boldsymbol{y} = \{y_1, y_2, \cdots, y_{T'}\}$

Encoder: $h_t = f(x_t, h_{t-1})$

– h_t is hidden state of input when f is a nonlinear activation function

Context: $\boldsymbol{c} = q(\{h_1, h_2, \cdots, h_T\})$

– a vector connecting encoder-decoder

Decoder: $s_t = f'(s_{t-1}, y_{t-1}, \boldsymbol{c})$

s_t is hidden state of output

$$P(y_t|y_{t-1}, \cdots, y_1, c) = g(y_{t-1}, s_t, \boldsymbol{c})$$

$$P(\mathbf{y}) = \prod_{t=1}^{T'} P(y_t|y_{t-1}, \cdots, y_2, y_1, \boldsymbol{c})$$

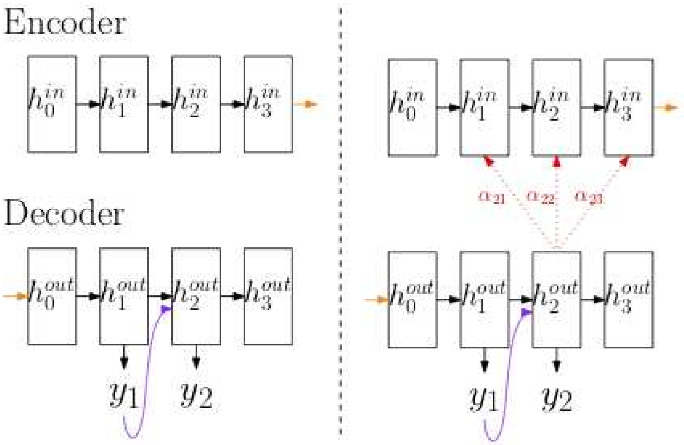
Transformer model

The bottleneck problem of the E-D connector

– encoding: capturing all information about the source sentence

– performance of E-D deteriorates rapidly as the length of an input sentence increases

Attention mechanism: maintains the RNN encoder, for each step j during decoding, computes an attention score α_{ji} for hidden representation $\boldsymbol{h}_i^{in}$ of each input token to obtain a context vector



Attention mechanism

Decoder

$$e_{ji} = a(\boldsymbol{h}_i^{in}, \boldsymbol{h}_j^{out})$$

$$\alpha_{ji} = \frac{\exp(e_{ji})}{\sum\limits_{i=1}^{T} \exp(e_{ji})}$$

$$\boldsymbol{c}_j = \sum_i^T \alpha_{ji} \boldsymbol{h}_i^{in}$$

Alignment function a: measures similarity between two tokens

$$\boldsymbol{y}_j = f_y(\boldsymbol{h}_j^{out}, \boldsymbol{y}_{j-1}, \boldsymbol{c}_j)$$

$$\boldsymbol{h}_{j+1}^{out} = f_n(\boldsymbol{h}_j^{out}, \boldsymbol{y}_{j-1})$$

f_y,f_h: output layer and hidden layer in RNN

a can be implemented by an FFN, besides RNNs, e.g., a single-layer neural network

Self-attention

External attention: the basic attention acore $e_i = a(\boldsymbol{u}, \boldsymbol{v}_i)$ is computed by matching an external pattern $\boldsymbol{u}$ with each element $\boldsymbol{v}_i$, and each score e_i indicates quality of match

Idea: self-attention (internal attention) is to replace $\boldsymbol{u}$ by (internal) parts of the sequence itself

– modeling long-distance context without a sequential dependency

E.g., “The math match is in progress between classes”, match is the sentence head on which all other tokens depend (subcatogorization)
Self-attention captures the intrinsic dependency
⇒ Transformer

Transformer

Transformers: stacked self-attention and point-wise fully connected layers for both the encoder and decoder

- eschewing recurrence (RNNs) instead relying entirely on an attention mechanism to draw global dependencies between input and output
  - The attention matrix is directly formed by the dot product of the input vectors
- both encoder and decoder are parallel
- No long connections: $O(1)$ for all tokens

Transformer

Best practice for the LMs, say contextualized word embeddings, not only neural MT task

E.g., word sense disambiguation

"I arrived at the bank after crossing the street."

"I arrived at the bank after crossing the river."

to learn contextual word embeddings that can capture semantic information from their surrounding contexts

Transformer operations

Problem: dot product between a vector and itself will always be high, so each hidden state will be biased towards attending to itself

Solution: first projecting the input into three different representations $\mathbf{QKV}$, using three different weight matrices $\mathbf{W}_Q\mathbf{W}_K\mathbf{W}_V$

- Query vector $\mathbf{q}_i = \mathbf{W}_q\mathbf{x}_i$: attended from, like the target in the attention mechanism

- Key vector $\mathbf{k}_i = \mathbf{W}_k\mathbf{x}_i$: attended to, like the source

- Value vector $\mathbf{v}_i = \mathbf{W}_v\mathbf{x}_i$: the context that is being generated

Note: In the attention mechanism, the key and value networks are identical

but separate representations make sense and are flexible

Transformer formulations

$$r_{ij} = (\mathbf{q}_i \cdot \mathbf{k}_j) / \sqrt{d}$$

$$a_{ij} = e^{r_{ij}} / \left(\sum_k e^{r_{ik}} \right)$$

$$\mathbf{c}_i = \sum_j a_{ij} \cdot \mathbf{v}_j$$

d: the dimension of $\mathbf{k}$, $\mathbf{q}$; $\sqrt{d}$ is scale factor (numerical stability)

i, j: indexes

r_{ij} is different from r_{ji} (asymmetric)

- The choice of $\mathbf{c}_i$ to use is learned from training examples
- In each transformer layer, self-attention uses the hidden vectors from the previous layer
- Input is initially the embedding layer

Transformer formulations

Encoding for all words in a sentence can be calculated simultaneously, using matrix operations

can be computed efficiently in parallel on GPU

$$\mathbf{A} = Softmax(\frac{(\mathbf{X}\mathbf{W}^Q)(\mathbf{X}\mathbf{W}^K)^\top}{\sqrt{d_{out}}}),$$

$$\mathbf{c} = \mathbf{A}^\top(\mathbf{X}\mathbf{W}^V)$$

Write as $Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V})$

Transformer formulations#

$$
\begin{aligned}
attn(h_t, h_s) &= Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \\
&= softmax(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_{out}}})\mathbf{V} \\
&= softmax(\frac{(\mathbf{X}\mathbf{W}^Q)(\mathbf{X}W^K)^\top}{\sqrt{d_{out}}})(\mathbf{X}\mathbf{W}^V),
\end{aligned}
$$

– $\mathbf{W}^Q$, $\mathbf{W}^K$, $\mathbf{W}^V \in \mathbb{R}^{d_{in} \times d_{out}}$ (d_{in},d_{out} are the input dimension and the output dimension) are matrices to be learned for transforming $\mathbf{X}$ to its $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$ representations

Cross attention

For seq-seq, the attention is applied to each token of the primary token sequence $\mathbf{X}$ treating the second token sequence $\mathbf{Z}$ as the context

Define the Softmax function for matrix arguments and a Mask matrix

$$\mathrm{Softmax}(\mathbf{A})\left[t_z, t_x\right] := \frac{\exp \mathbf{A}\left[t_z, t_x\right]}{\sum_t \exp \mathbf{A}\left[t, t_z\right]}$$

$$\mathrm{Mask}\left[t_z, t_x\right] = \begin{cases} 1 & \text{for bidirectional attention} \\ [[t_\mathrm{z} \leq t_\mathrm{x}]] & \text{for unidirectional attention} \end{cases}$$

Self-attention can be viewed as a special cross-attention

–Bidirectional unmasked self-attention: attend to each token, treating all tokens in the sequence as the context (Mask≡1)

–Unidirectional masked self-attention: attend to each token, treating all preceding tokens (including itself) as the context (autoregressive). Future tokens are masked out ($\mathbf{Z} = \mathbf{X}$)

Multihead attention

The context-based summarization $\mathbf{c}$ is a sum over all previous positions in the sentence

– sometimes important information gets lost because it is essentially averaged out over the whole sentence

Multihead attention: dividing the sentence up into m equal pieces and apply the attention model to each of the m pieces

– Each piece has its own set of weights

The results are concatenated together to form $\mathbf{c}$

Transformer layers

Each transformer layer consists of several sub-layers

– self-attention layer

– feedforward layers, where the same FFN weight matrices are applied independently at each position

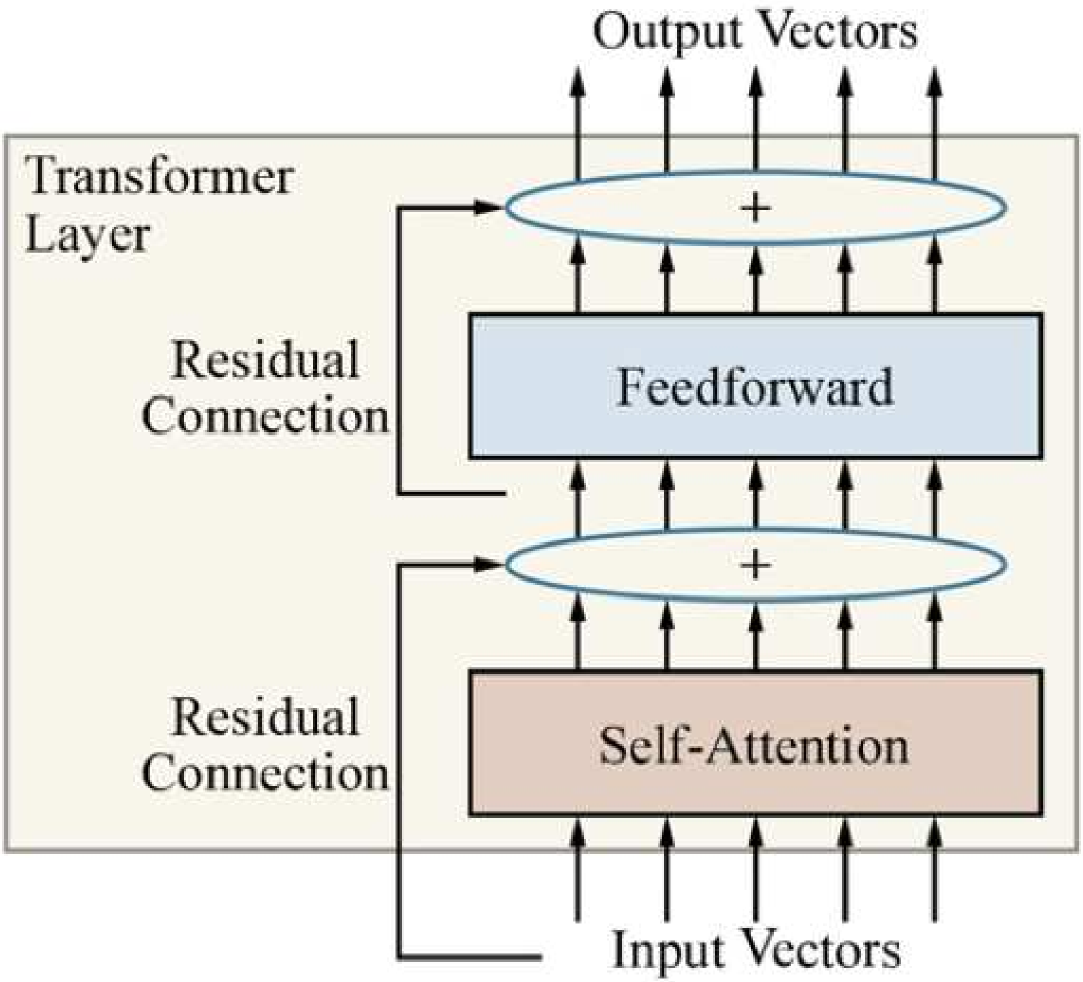
– – a nonlinear activation function (ReLU) is applied after the first feedforward layer for better representation

– residual connections (resolving vanishing gradient)

Typically, transformer models have six or more layers

– the output of layer i is used as the input to layer $i+1$

Transformer layers

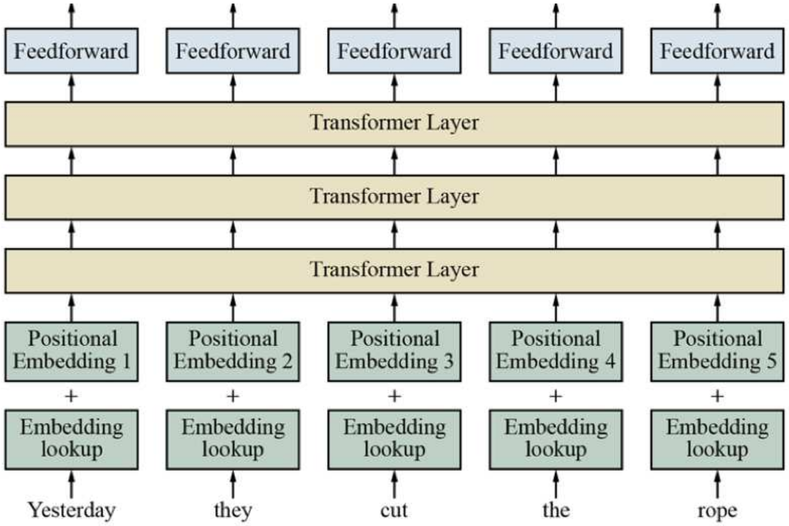


Positional embedding

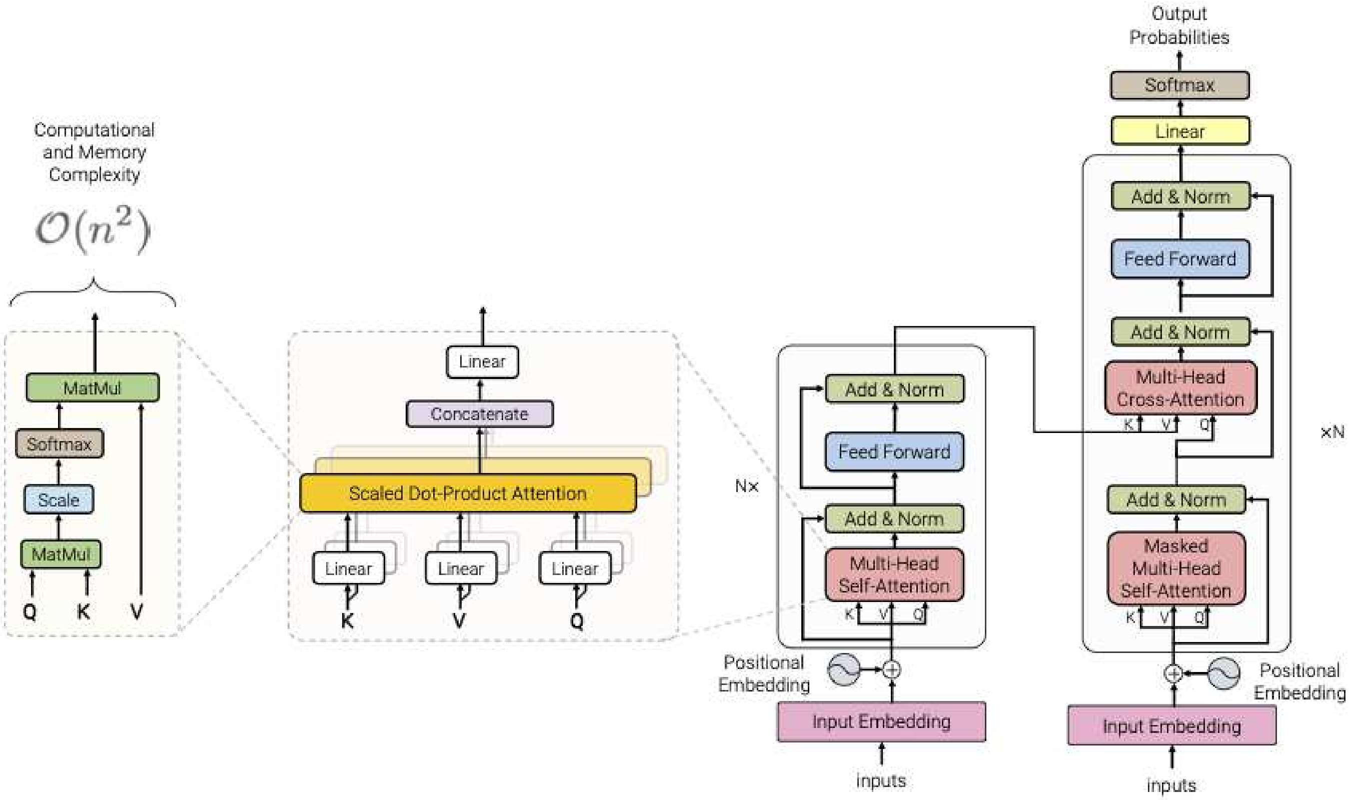
Positional embedding: if the sequence has a maximum length of l_{max}, we learn l_{max} new embedding vectors for each word position, capture the ordering of the words

– self-attention is agnostic to word order (permutation invariant)

– the input to the first transformer layer is the sum of the word embedding at position plus the positional embedding corresponding to position



The architecture of transformer



Reading: Vaswani, A et al. Attention is all you need, arXiv, 2017

Transformer formulations$^{\#}$

Transformer networks (TRMs) are the E-D that stack the self-attention and (element-wise fully connected) FFN layers for both the encoder and decoder

$$\begin{aligned} \text{Encoder}: h_t &= ffn(attn(h_{t-1}, h_t(x_t))) \\ \mathbf{c}_s &= attn(h_t, g_{s-1})) \\ \text{Decoder}: g_s &= ffn(attn(attn(g_{s-1}, g_s(y_s)), \mathbf{c}_s)) \end{aligned}$$

– $h_t(x_t), g_s(y_s)$: the attentive position of the hidden layer
– $attn((\cdot), \boldsymbol{c}_s)$: encoder-decoder (cross) attention that is the E-D connector by the attention mechanism

– – similar to self-attention, except it creates its queries matrix from the layer below it, and takes the keys and values matrix from the output of the encoder

Transformer algorithms$^{\#}$

```
def $\textsc{Transformer}(\mathbf{w} \mid \theta)$   // Encoder-decoder transformer forward pass
    Input $\mathbf{w} \in V^*$, a sequence of token IDs
    Parameters $\theta$: $\mathbf{W}_e, , \mathbf{W}_p, \mathbf{W}_{QKV}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{W}_u$
    Hyperparameters $l_{max}, H$            // See below
      // — Encoder —
    for length($\mathbf{w}$) do
        $\mathbf{w}_e \leftarrow \textsc{TokenEmbedding}(\mathbf{w}, \mathbf{W}_e)$
        $\mathbf{w}_p \leftarrow \textsc{PositionalEmbedding}(l, \mathbf{W}_p)$
        $\mathbf{X} \leftarrow \mathbf{w}_e + \mathbf{w}_p$    // Input embedding
    for length of the encoder do
        $\mathbf{X} \leftarrow \mathbf{X} + \textsc{MultiheadAttention}(\mathbf{X}, \mathbf{X})$ //self-attention ($\mathrm{Mask} \equiv 1$)
        for length($\mathbf{X}$) do $\mathbf{X} \leftarrow \textsc{Norm}(\mathbf{c})$
        $\mathbf{X} \leftarrow \textsc{FeedForward}(\mathbf{X})$
        for length($\mathbf{X}$) do $\mathbf{X} \leftarrow \textsc{Norm}(\mathbf{X})$    // send to the decoder
```

Transformer algorithms$^{\#}$

```
  // — Decoder —
$\mathbf{Z} \leftarrow \mathbf{Y}.\textsc{Left}$
for length($\mathbf{Z}$) do
    $\mathbf{w}_e \leftarrow \textsc{TokenEmbedding}(\mathbf{Z}, \mathbf{W}_e)$
    $\mathbf{w}_p \leftarrow \textsc{PositionalEmbedding}(l, \mathbf{W}_p)$
    $\mathbf{Z} \leftarrow \mathbf{w}_e + \mathbf{w}_p$
for length of the decoder do
    $\mathbf{Z} \leftarrow \mathbf{Z} + \textsc{MultiheadAttention}(\mathbf{Z}, \mathbf{Z})$
      // self-attention with parameter sharing $(\mathrm{Mask}\,[[t_{\mathbf{z}} \leq t_{\mathrm{x}}])$
    for length($\mathbf{Z}$) do $\mathbf{Z} \leftarrow \textsc{Norm}(\mathbf{Z})$
    $\mathbf{Z} \leftarrow \textsc{MultiheadAttention}(\mathbf{Z}, \mathbf{X})$
      // cross-attention, $\mathbf{Z}$ is query sequence $(\mathrm{Mask} \equiv 1)$
    for length($\mathbf{Z}$) do $\mathbf{Z} \leftarrow \textsc{Norm}(\mathbf{Z})$
    $\mathbf{Z} \leftarrow \textsc{FeedForward}(\mathbf{Z})$
    for length($\mathbf{Z}$) do
        $\mathbf{Y} \leftarrow \textsc{Unembedding}(\textsc{Linear}(\textsc{Norm}(\mathbf{Z})))$
return $\mathbf{Y}$  // Output probabilities
```

Transformer algorithms#

```
def TokenEmbedding($\mathbf{w} \mid \mathbf{W}_e$)
    Input $w_i \in V$, a token ID
    Parameters $\mathbf{W}_e \in \mathbb{R}^{D\times|V|}$, the token embedding matrix
    return $\mathbf{w}_e = \mathbf{W}_e[:, w_i]$
      // output $\mathbf{w}_e \in \mathbb{R}^D$, the vector representation of the token
def PositionalEmbedding($l \mid \mathbf{W}_p$)
    Input $l \in [l_{max}]$, position of a token in the sequence
    Output $\mathbf{w}_p \in \mathbb{R}^D$
    Parameters $\mathbf{W}_p \in \mathbb{R}^{D\times l_{max}}$, the positional embedding matrix
    return $\mathbf{w}_p = \mathbf{W}_p[:, l]$
      // output $\mathbf{w}_p \in \mathbb{R}^D$, the positional embedding matrix
```

Transformer algorithms$^{\#}$

```
def ATTENTION($\mathbf{X}, \mathbf{Z} \mid \mathbf{W}_{QKV}$, Mask)
    // Computes a single (masked) self- or cross- attention head
    Input $\mathbf{X} \in \mathbb{R}^{d_x \times\ l_x}$,$\mathbf{Z} \in \mathbb{R}^{d_z \times\ l_z}$
    vector representation of the current and context sequences
    Parameters $\mathbf{W}_{QKV}$: $\mathbf{W}_Q \in \mathbb{R}^{d_{attn} \times d_x}$,$\mathbf{W}_K \in \mathbb{R}^{d_{attn} \times d_z}$, $\mathbf{W}_V \in \mathbb{R}^{d_{out} \times d_z}$
    the query, key and value linear projections
    Hyperparamter $\mathrm{Mask} \in \{0,1\}^{l_z \times l_x}$
    $\mathbf{A} \leftarrow \mathrm{Softmax}(\frac{(\mathbf{X}\mathbf{W}^Q)(\mathbf{Z}\mathbf{W}^K)^\top}{\sqrt{d_{attn}}})$
    if $\neg\mathrm{Mask}[t_x, t_z]$ then $\mathbf{A}[t_x, t_z] \leftarrow -\infty$
    return $\mathbf{c} = \mathbf{A}^\top(\mathbf{Z}\mathbf{W}^V)$
      // output $\mathbf{c} \in \mathbb{R}^{d_{out} \times d_{l_x}}$,updated representations of tokens in $\mathbf{X}$
    in the context of tokens in $\mathbf{Z}$
```

Transformer algorithms#

```
def $\textsc{MultiheadAttention}(\mathbf{X}, \mathbf{Z} \mid \mathbf{W}^h_{QKV}, \mathrm{Mask}\,, H)$
  // Computes a multihead (masked) self- or cross- attention head
  Input $\mathbf{X} \in \mathbb{R}^{d_x \times\ l_x}$,$\mathbf{Z} \in \mathbb{R}^{d_z \times\ l_z}$
  Parameters $\mathbf{W}^h_{QKV}$: for each $h \in\ [H]$
    $\mathbf{W}^h_Q \in \mathbb{R}^{d_{attn} \times d_x}$,$\mathbf{W}^h_K \in \mathbb{R}^{d_{attn} \times d_z}$, $\mathbf{W}^h_V \in \mathbb{R}^{d_{mid} \times d_z}$
    $\mathbf{W}_o \in \mathbb{R}^{d_{out} \times H\, d_{mid}}$
  Hyperparamters $\mathrm{Mask} \in \{0, 1\}^{l_z \times l_x}$
    $H$, number of attention heads
  for each $h \in\ [H]$ do
    $\mathbf{c}^h \leftarrow \textsc{Attention}(\mathbf{X}, \mathbf{Z} \mid \mathbf{W}^h_{QKV}, \mathrm{Mask})$
  $\bar{\mathbf{c}} \leftarrow [\mathbf{c}^1, \cdots, \mathbf{c}^H]$  // concatenation
  return $\mathbf{c} = \mathbf{W}_o \bar{\mathbf{c}}$
   // output $\mathbf{c} \in \mathbb{R}^{d_{out} \times d_{l_x}}$
```

Transformer algorithms$^{\#}$

```
def NORM($\mathbf{X} \mid \boldsymbol{\beta}, \boldsymbol{\gamma}, \mathbf{W}_u$)
    // Normalizes layer activations $\mathbf{X}$
    Input $\mathbf{X} \in \mathbb{R}^{d_x}$, NN activations
    Parameters $\boldsymbol{\beta}, \boldsymbol{\gamma} \in \mathbb{R}^{d_x}$, element-wise scale and offset
    $\mathbf{m} \leftarrow \sum_{i=1}^{d_x} \mathbf{X}[i] \ / \ d_x$
    $\mathbf{n} \leftarrow \sum_{i=1}^{d_x} (\mathbf{X}[i] - \mathbf{m})^2 \ / \ d_x$
    return $\mathbf{X} = \frac{\mathbf{X}-\mathbf{m}}{\sqrt{\mathbf{n}}} \odot \ \boldsymbol{\gamma} + \boldsymbol{\beta}$
      // output $\mathbf{X} \in \mathbb{R}^{d_x}$, normalized activations

def UNEMBEDDING($\mathbf{c} \mid \mathbf{W}_u$)
    Input $\mathbf{Z} \in \mathbb{R}^{d_x}$, a token encoding
    Parameters $\mathbf{W}_u \in \mathbb{R}^{|V| \times d_x}$, the unembedding matrix
    return $\mathbf{Y} = \mathrm{Softmax}(\mathbf{W}_u \mathbf{Z})$
      // output $\mathbf{Y} \in \ P(V)$, a probability distribution over the vocabulary
```

Transformer training algorithms$^{\#}$

```
def TransformerTraining($\mathbf{d}, \theta$)
    // Training a seq-seq model
    Input $\mathbf{d} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N}$, a dataset of pairwise sequence
        $\theta$, initial transformer parameters
    Hyperparamter Epochs $\in \mathbb{N}, \eta \in \ (0, \infty)$
    for $i = 1, 2, \cdots, \text{Epochs}$ do
        for $n = 1, 2, \cdots, N$ do
            $l \leftarrow \text{length}(\mathbf{x}_n)$
            $\mathbf{P}(\theta) \leftarrow \text{Transformer}(\mathbf{x}_n, \mathbf{y}_n \mid \theta)$
            $\text{loss}(\theta) \leftarrow -\sum_{t=1}^{l-1} \ \log \ \mathbf{P}(\theta) \left[\mathbf{x_n}[t+1], \ t\right]$
            $\theta \leftarrow \theta - \eta \cdot \ \nabla \text{loss}(\theta)$
    return $\hat{\theta} = \theta$
      // $\hat{\theta}$, the trained parameters
```

Transformer inference algorithms#

```
def TRANSFORMERINFERENCE($\mathbf{x}, \hat{\theta}$)
    // Using a trained seq-seq model for prediction
    Input A transformer with trained parameters $\theta$
        $\mathbf{x}$, an input sequence
    Hyperparamter $\tau \in \ (0, \infty)$
    $\mathbf{y} \leftarrow [\text{bos\_token}]$
    $y \leftarrow 0$
    while $s \neq \text{eos\_token}$ do
        $\mathbf{P} \leftarrow \text{TRANSFORMER}(\mathbf{x}, \mathbf{y} \mid \theta)$
        $\mathbf{P} \leftarrow \mathbf{P}[:, length(x)]$
        sample a token $y$ from $q \ \propto \ P^{1\ /\ \tau}$
        $\mathbf{y} \leftarrow [\mathbf{y}, y]$
    return $\mathbf{y}$
```

Variants of Transformer#

Decoder-only Transformer: a variant of the Transformer architecture that consists only of the decoder, without the encoder

where only generation or autoregressive prediction is required, using a decoder-only Transformer can be more efficient

there is no need for an encoder to process the input sequence

Encoder-only Transformer: only the encoder, excluding the decoder

where bidirectional context is sufficient or where only encoding is required (e.g. text classification etc.)

There are a lot of variants of Transformer

Examples: BERT and GPT$^+$

Hints

- BERT is bi-directional LM and the encoder-only for language understanding; GPT (successors GPT2/GPT3/GPT4) is unidirectional LM and the decoder-only for language generation
- There are many variants and applications of BERT and GPT
- There are a lot of PLMs other than BERT/GPT

Readings: Devlin J et al., Bert: Pre-training of deep bidirectional transformers for language understanding, arXiv, 2018

Brown T et al. Language models are few-shot learners, arXiv, 2020 (GPT3)

Example: AlphaCode*

Programming is a general-purpose problem-solving tool

Say, answering university-level math problems

AlphaCode: a system for code generation, top 54.3% in programming competitions on the Codeforces w/ >5,000 participants

- Pretrain an E-D transformer-based LM on GitHub code
- Finetune the model on the dataset of competitive data
- Generate a large number of samples from the models for each problem; filter the samples to obtain a small set of submissions

Readings: Li Y et al., Competition-Level Code Generation with AlphaCode, DeepMind, 2022

Drori I et al., A Neural Network Solves and Generates Mathematics Problems by Program Synthesis: Calculus, Differential Equations, Linear Algebra, and More, PNAS, 2022

Large language models

LLMs are the PLMs trained on vast amounts of data, characterized by their large size in terms of parameters

– Scale: billions or even trillions of data

– Size: a large number of parameters, from billions or even more

– Architectures: mostly Transformers

Multi-modal LLM (MM-LLM): a type of LM/LLM that incorporates information from multiple modalities, such as text, images, audio, or video, into its training and inference processes

– Integration of multi-modalities, cross-modal understanding

– Training Data: paired examples consisting of text along with accompanying images, audio, or video, as well as unimodal data for each modality

– Architecture: Transformers are commonly adapted and extended to handle multi-modal inputs

Scaling laws$^{\#}$

Scaling laws in LLMs refers to the relationship between model size, dataset size, computational resources, and performance

– Larger models have been shown to achieve better performance on various NLP tasks

Compute-resource scaling laws suggests that as the model size increases, so does the amount of computational resources needed to train and deploy the model effectively

– Performance improvement comes at the cost of increased computational resources

— Larger models require more memory, longer training times, and greater computational power for training and inference.

Emergent abilities

Emergent abilities of LLMs are the abilities that are not present in small models but arise in large models

– In-context learning introduced by 175B GPT-3: provided with a natural language instruction and/or several task demonstrations, it can generate the expected output for the test instances by completing the word sequence of input text, without requiring additional training or gradient update

– Instruction following: finetuning with a mixture of multi-task datasets formatted via natural language descriptions (instruction tuning), LLMs perform well on unseen tasks that are also described in the form of instructions

– Step-by-step reasoning: LLMs can solve complex tasks by utilizing the prompting mechanism that involves intermediate reasoning steps for deriving the final answer

Emergent abilities

By analogy, such an emergent pattern has close connections with the phenomenon of phase transition in physics

Do the emergent abilities of LLMs mean some kind of machine intelligence??
– differing from human one?

Ability eliciting: design suitable task instructions or specific in-context learning strategies to elicit the abilities

Example: Prompting engineer$^{\#}$

Prompting engineer refers to a role or skill set of designing effective prompts for LLMs

Designing prompts, controlling model behavior, evaluating prompt effectiveness, domain expertise, ethical considerations, etc.

E.g., designing prompts that prompt the chatbot to ask clarifying questions, and provide relevant information for a call center

Human alignment

Alignment tuning: making LLMs act in line with human expectations (e.g., helpful, honest, and harmless)

– Pretraining lacks the consideration of human values or preferences

– LLMs exhibit unintended behaviors

Reinforcement learning from human feedback (RLHF) proposed to finetune LLMs with the collected human feedback data

RLHF algorithm$^{\#}$

RLHF algorithm

1. **Initialization**: the model
2. **Interaction** with the environment: taking actions based on current policy
3. **Human feedback**: human feedback is collected to evaluate the quality of the actions
4. **Updating** the policy: using the collected human feedback to update the policy
5. **Iterative** learning: continuing to interact and collect human feedback and update policy iteratively
6. **Convergence**: to converge to a policy that maximizes the cumulative reward or satisfies the user's preferences based on the human feedback provided

Hallucination$^{\#}$

Hallucination means that the generated information is either in conflict with the existing source (intrinsic hallucination) or cannot be verified by the available source (extrinsic hallucination)

Partially alleviated approaches
– alignment tuning
– tool utilization: the integration of external tools for the provision of credible information sources
– uncertainty estimation of LLMs to identify hallucinations

Multi-modal LLMs#

Multi-modal integration in LLMs

- Early fusion: the input data from different modalities are combined at the input layer of the LLM

– a single, unified representation for all modalities, preserving the relationships between modalities, but may suffer from modality mismatches

- Late fusion: processing each modality separately through dedicated branches of the LLM and then merging the representations at a higher-level layer

– capturing modality-specific features before combining them, potentially reducing modality mismatches

- Cross-modal attention: enabling the LLM to dynamically attend to relevant information from different modalities, facilitating effective integration of multi-modal information

Multi-modal LLMs

Benefits of MM-LLM

- Richer understanding: richer semantics and contextual cues, e.g., combining text with images enables the model to understand concepts that are difficult to express in words alone
- Improved performance: complementary information from different modalities, e.g., combining textual and visual information in image captioning tasks often results in more descriptive and accurate captions
- Enhanced user experience: more natural and intuitive interactions

E.g., GPT-4V, Gemini 1.5

from millions of tokens of context, including multiple long documents and hours of video and audio

Examples: Multi-modal LLMs$^{\#}$

Visual Question Answering (VQA): LLMs answer questions about images by combining visual and textual information

Image Captioning: LLMs generate descriptive captions for images based on their visual content

Speech Emotion Recognition: LLMs analyze audio signals to detect and classify the emotional states of speakers

Multi-modal Chatbots: LLMs engage in natural conversations using text, images, and audio inputs

Applications of LLMs#

- Healthcare
- Education
- Law
- Finance
- Scientific research (AI4Science)

Most likely, LLMs can be used as a foundation model for NLP

- NLP tasks
  as downstream tasks of LLMs

Natural language processing

NLP encompasses a wide range of tasks, each addressing different aspects of language understanding and generation

- Machine Translation (MT): Translating text from one language to another while preserving its meaning and context
- Speech Recognition: Transcribing spoken language into text

NLP tasks#

- Text Classification: Classifying text into predefined categories or labels based on its content
  - applications: sentiment analysis, topic classification, spam detection
- Named Entity Recognition (NER): Identifying and classifying named entities (e.g., persons, organizations) mentioned in the text
  - information extraction, entity linking
- Part-of-Speech Tagging (POS): Assigning grammatical categories (e.g., noun, verb) to each word in a sentence
  - many downstream NLP tasks
- Dependency Parsing: Analyzing the grammatical structure of a sentence to identify the relationships between words
  - syntactic analysis, information extraction

NLP tasks

- Text Summarization: Generating concise and coherent summaries of longer text documents
  -extract or abstract information
- Question Answering (QA): Answering questions posed in natural language based on a given context or knowledge base
  - information retrieval, reading comprehension
- Sentiment Analysis: Analyzing the sentiment or opinion expressed in the text to determine whether it is positive, negative, or neutral
  - social media monitoring, customer feedback analysis, market research
- Text Generation: Generating coherent and contextually relevant text based on a given prompt or input
  -language modeling, dialogue generation, content creation

NLP tasks

- Named Entity Linking (NEL): Linking named entities mentioned in the text to entries in a knowledge base or reference database
  - Language Understanding: Understanding the meaning and intent behind natural language utterances

    - intent classification, slot filling, dialogue state tracking
- Text Clustering and Similarity: Grouping similar documents or text passages into clusters based on their content or similarity

    - information retrieval, document organization, recommendation systems
- Textual Entailment Recognition: Determining whether one text (the premise) logically entails another text (the hypothesis)

    - question answering, information retrieval, inference
- Text Alignment and Paraphrasing: Aligning and comparing text passages to identify similarities, differences, or paraphrases

    - plagiarism detection, duplicate detection, text simplification

Machine translation

MT: automatic translation of text from one natural language (the source) to another (the target)

Try to translate a passage of a page in a browser by Google translator or ChatGPT in the source Chinese into the target English, and then translate back from English to Chinese

What can you find??

A translator (human or machine) requires an in-depth understanding of the bilingual text

A representation language that makes all the distinctions necessary for a set of languages is called an interlingua

– creating a complete knowledge representation of everything
– parsing into that representation
– generating sentences from that representation

Neural machine translation

NMT (Neural MT): end-to-end (deep) learning approach for MT

– regard MT as a sequence-sequence prediction task and, without using any information from conventional MT systems

– design two deep neural networks ⇒ viewing MT as recognition

– – an encoder: to learn continuous representations of source language sentences

– – a decoder: to generate the target language sentence with source sentence representation

Currently, NMT is the best MT system over rule-based or statistical MT systems

Transformer for machine translation

The best practice NMLs are based transformer: input source $\boldsymbol{x}$, output target $\boldsymbol{y}$

$$h_t = Transformer\_encoder(h_{t-1}, h_t(\boldsymbol{x}_t))$$
$$g_s = Transformer\_decoder(g_{s-1}, g_s(\boldsymbol{y}_s))$$

Decoding: once training is complete, give a source sentence, generate the corresponding target sentence

one word at a time, and then feedback in the word generated the next timestep

- Greedy decoding: selecting the highest probability word

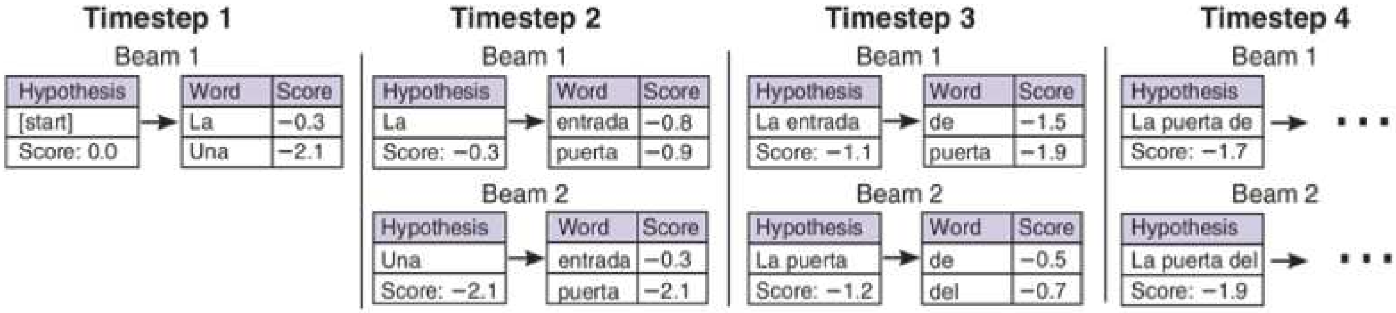
– may not maximize the probability of the entire target sequence

Decoding

Beam search: optimal decoding

– keeping the top k hypotheses at each stage, extending each by one word using the top k choices of words, then choosing the best k of the resulting k^2 new hypotheses

– when all hypotheses in the beam generate the special $\langle \text{end} \rangle$ token, the algorithm outputs the highest scoring hypothesis



Timestep 1

Beam 1

Hypothesis
[start]
Score: 0.0

Word	Score
La	−0.3
Una	−2.1

Timestep 2

Beam 1

Hypothesis
La
Score: −0.3

Word	Score
entrada	−0.8
puerta	−0.9

Beam 2

Hypothesis
Una
Score: −2.1

Word	Score
entrada	−0.3
puerta	−2.1

Timestep 3

Beam 1

Hypothesis
La entrada
Score: −1.1

Word	Score
de	−1.5
puerta	−1.9

Beam 2

Hypothesis
La puerta
Score: −1.2

Word	Score
de	−0.5
del	−0.7

Timestep 4

Beam 1

Hypothesis
La puerta de
Score: −1.7

...

Beam 2

Hypothesis
La puerta del
Score: −1.9

...

Beam size $b = 2$

Machine translation as a downstream task of LLM

Downstream tasks of LLMs involve finetuning pretrained LLMs on specific tasks or domains

Early integration of LLMs in MT tasks, such as pretraining on parallel corpora and finetuning on translation objectives

LLMs revolutionized MT with their ability to capture context, semantics, and stylistic nuances, leading to more fluent and natural translations

E.g., Google's NMT system and OpenAI's GPT-based MT

Adaptability to diverse language pairs and domains

Speech recognition*

Speech recognition: identify a sequence of words uttered by a speaker, given the acoustic signal
It's not easy to wreck a nice beach (recognize speech)
Speech signals are noisy, variable, ambiguous

Use Bayes' rule

$$P(Words|signal) = \alpha P(signal|Words)P(Words)$$

I.e., decomposes into acoustic model + (statistical) language model
$Words$ are the hidden state sequence, $signal$ is the observation sequence

Automatic speech recognition (ASR): pretrained encoder and decoder
End-to-end speech to text translation (ST) and text-to-speech (TTS): pretrained MT
Speech-speech translation: sequence-sequence model
$\Leftarrow$ pretrained transformer $\Rightarrow$ downstream tasks of LLMs

Phones*

All human speech is composed from 40-50 phones, determined by the configuration of articulators (lips, teeth, tongue, vocal cords, air flow)

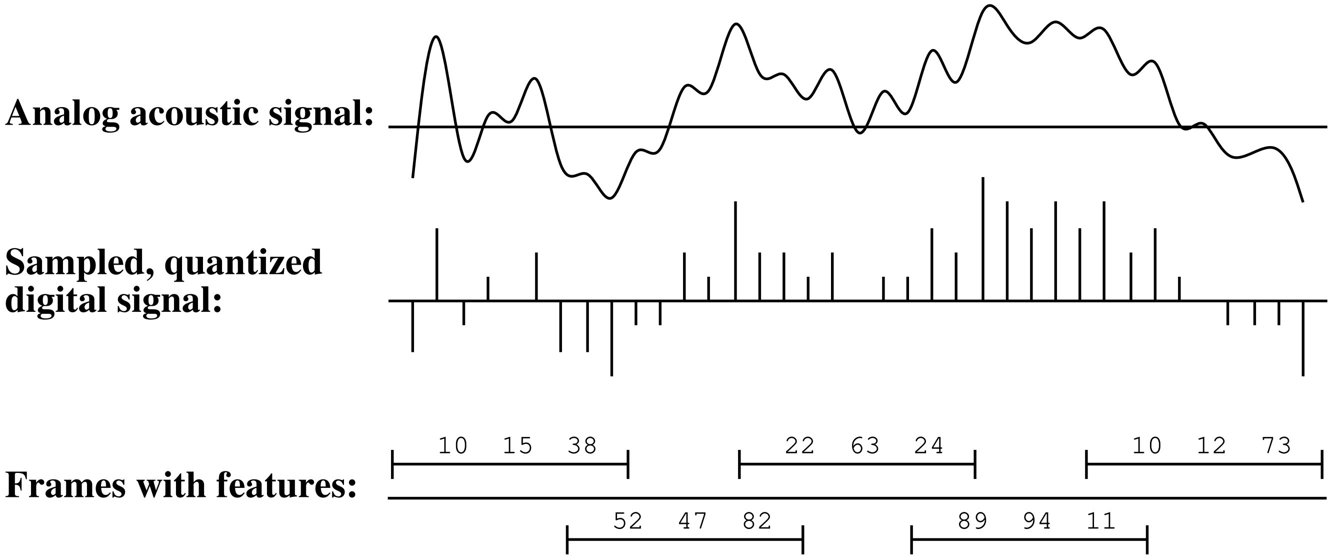
Form an intermediate level of hidden states between words and signal
⇒ acoustic model = pronunciation model + phone model

ARPAbet designed for American English

[iy]	be**a**t	[b]	**b**et	[p]	**p**et
[ih]	b**i**t	[ch]	**Ch**et	[r]	**r**at
[ey]	b**e**t	[d]	**d**ebt	[s]	**s**et
[ao]	b**ough**t	[hh]	**h**at	[th]	**th**ick
[ow]	b**oa**t	[hv]	**h**igh	[dh]	**th**at
[er]	B**er**t	[l]	**l**et	[w]	**w**et
[ix]	ros**e**s	[ng]	si**ng**	[en]	butt**on**
⋮	⋮	⋮	⋮	⋮	⋮

Speech sounds#

Raw signal is the microphone displacement as a function of time; processed into overlapping 30ms frames, each described by features



Frame features are typically formants—peaks in the power spectrum

Audio-language models

ALMs: designed to understand and generate language from spoken audio signals and bridge the gap between audio and text

- Audio processing: transcribe and analyze spoken language from audio recordings
- Audio-language pretraining (ALP)
- learn to associate audio with textual descriptions in a shared latent space
- training objectives capturing semantic alignment between audio and textual modalities
- training on speech-text pairs using self-supervised learning

⇒ MM-LLMs

TTS: Text-to-speech

Consider a dataset $\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=0}^{N}$

- $\mathbf{y}$: an audio sample
- $\mathbf{x} = \{x_1, \cdots, x_T\}$: the corresponding text transcription

The audio $\mathbf{y} = \{y_1, \cdots, y_S\}$ is represented by a sequence of S discrete tokens

– a separately trained speech tokenizer

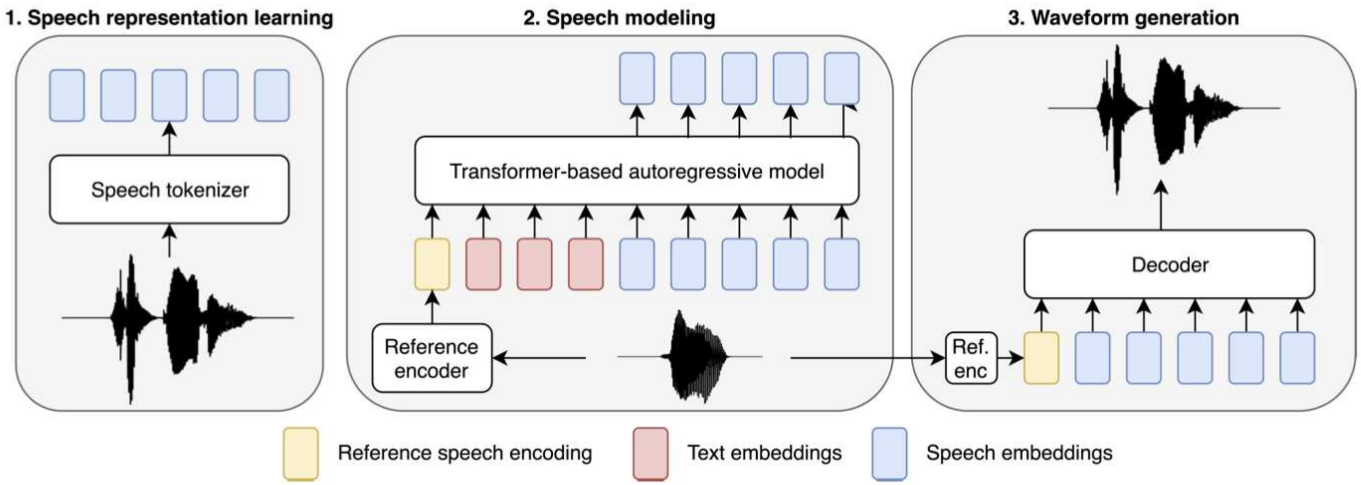
Transformer-based autoregressive model with parameters ϕ to learn the joint probability of the text and audio sequences (i.e. LLM-based)

$$p(\mathbf{y}, \mathbf{x}) = p(\mathbf{y} \mid \mathbf{x})p(\mathbf{x}) = \prod_{s=1}^{S} p\left(\mathbf{y}_s \mid \mathbf{y}_{<s}, \mathbf{x}; \phi\right) \prod_{t=1}^{T} p\left(\mathbf{x}_t \mid \mathbf{x}_{<t}; \phi\right).$$

The predicted speech tokens are concatenated with speaker embeddings and decoded into waveforms

– a separately trained decoder consisting of linear and CNN layers

TTS: Text-to-speech



Emergent abilities: say BASE TTS (2024), built with 10K+ hours and 500M+ parameters begin to demonstrate natural prosody on textually complex sentences

LLM-based ASR*

Massively multilingual ASR refers to the capability of a single system to transcribe speech into text across a wide range of languages

LLMs have emerged as a promising approach for massively multilingual ASR

$$y = \mathrm{LLM}\left(I, \left(x_1, y_1\right), \left(x_2, y_2\right), \ldots, \left(x_k, y_k\right), x\right)$$

where x represents the ASR transcription result, and y is the correct transcription. The pairs $(x_i, y_i)_{i=1}^{k}$ are the k examples given to the LLM, and I is the instruction provided to the LLM (in-context learning)

LLM-based ASR*

1. Data collection and preprocessing

   - Preprocessed to extract acoustic features

2. Training LLMs for ASR

   - LLMs are trained or finetuned to map acoustic features to corresponding textual transcripts in multiple languages

3. Language modeling and adaptation

   - Capture the linguistic characteristics of diverse languages

   - LMs may be shared across languages, leveraging the commonalities between languages to improve performance on low-resource languages

4. Handling multilingual input

   - During inference, the LLM processes incoming speech signals from multiple languages simultaneously

   - The LLM's language identification module identifies the language(s) spoken in the input signal

Conversation AI$^+$

Conversation AI, Artificial Conversation Intelligence (ACI) (conversation systems, conversation agents, dialogue systems, (chat)bots): converse with humans via natural language ⇐ Turing test

– the first kind of NLP, the crown pearl of AI Types of ACI

- QA: single turn conversation
- Task-oriented agents: short conversations to get information from the user for a particular task

  E.g., intelligent assistants, such as intelligent personal assistants (IPAs), on every cellphone or on home controllers (Siri, Cortana, Alexa, Google Home etc.).
- Chatbots: long (unstructured) conversations like human-human interaction (open-ended non-task-oriented, open domain)
- Companionable agents: conversations with users day by day by building long-term relationships with them

Dialogue

A dialogue contains the following phases

1. Input recognizer: converting user's input to plain text
   e.g., speech, gesture, handwriting etc.
2. NLP units
3. Dialog manager: analyzing the semantic information, keeping the history and state of the dialog and managing the general flow of the conversation
4. Task managers: the dialog manager contacts one or more, that have knowledge of the specific task domain
5. Output generator: producing output
   e.g., natural language generator, layout engine etc.
6. Output renderer
   e.g., text-to-speech engine (TTS), talking head, robot, avatar etc.

Example: ELIZA#

The ELIZA program (Weizenbaum, 1966): imitate the responses of a Rogerian psychotherapist

User: You are like my father in some ways.
ELIZA: WHAT RESEMBLANCE DO YOU SEE?
User: You are not very aggressive but I think you don't want me to notice that.
ELIZA: WHAT MAKES YOU THINK I AM NOT AGGRESSIVE?
User: You don't argue with me.
ELIZA: WHY DO YOU THINK I DON'T ARGUE WITH YOU?
User: You are afraid of me.
ELIZA: DOES IT PLEASE YOU TO BELIEVE I AM AFRAID OF YOU?

ELIZA#

Use of pattern-matching to process the input and translate it into suitable outputs

Word	Rank	Pattern	Outputs
alike	10	?X	In what way?

ELIZA doesn't actually need to know anything to mimic a Rogerian psychotherapist

So-called Eliza-style fool

ChatGPT

Trying practical dialogue systems

say, ChatGPT ⇒ say something??

How long have you been asked for a dialogue??

Question answering

Question answering (QA): answering a question, not a ranked list of documents but rather a short response (a sentence or phrase)

– retrieval-based approach
– neural QA

Retrieval-based QA may use either a pre-structured database or a collection of natural language documents (a text corpus such as Web)

Question types: fact, list, definition, how, why, hypothetical, semantically constrained, and cross-lingual questions

AskMSR: Web-based QA system (2002)

Hint: QA can be seen as single turn conversation and implemented by LMs (see later)

Example: DeepQA#

Watson: IBM's DeepQA

– In 2011, competed on the quiz show Jeopardy

– access to 200 million pages of structured and unstructured content consuming four terabytes of disk storage, including the full text of Wikipedia, but was not connected to the Internet during the game

Conversation model

Denote input utterances $\boldsymbol{x}$ for a user and output responses $\boldsymbol{y}$ for a bot

A dialog history (context) samples
$\boldsymbol{d} = (\boldsymbol{x}, \boldsymbol{y}) = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{T} = (\boldsymbol{x}_1, \boldsymbol{y}_1; \boldsymbol{x}_2, \boldsymbol{y}_2; \cdots; \boldsymbol{x}_T, \boldsymbol{y}_T)$,
where at dialog turn t an utterance of tokens $\boldsymbol{x}_t = (x_t^1, x_t^2, \cdots, x_t^{X_t})$ and a response of tokens $\boldsymbol{y}_t = (y_t^1, y_t^2, \cdots, y_t^{Y_t})$, whose lengths allow to vary and $x_t^i, y_t^i \in \mathcal{V}$ for any time step i

$$P(\boldsymbol{y}_t) = \sum_{i=1}^{T_y} \log P\left(y_t^i \mid y_t^{<i}, \boldsymbol{x}_{\leq t}, \boldsymbol{y}_{<t}; f_\theta\right)$$

The distribution of the generated $P(\boldsymbol{y}_t)$ would be indistinguishable from that of the ground truth $P(\boldsymbol{x}_{t+1})$ and $Y_t = X_{t+1}$
⇐ Conversation model (CM)
⇐ transformer-based LM as CM

ChatGPT algorithm

```python
# Initialize the model
chatGPT = GPT3_5()

# Define conversation context
conversation_history = []

# Start the conversation loop
while True:
    # Get user input
    user_input = input("User: ")

    # Add user input to conversation history
    conversation_history.append(user_input)

    # Generate response using GPT-3.5 based on conversation history
    response = chatGPT.generate_response(conversation_history)

    # Add generated response to conversation history
    conversation_history.append(response)

    # Output response to user
    print("ChatGPT:", response)

    # Check for conversation termination signal
    if user_input.lower() == "exit":
        break
```

Understanding*

Levels of understanding

1. **Keyword processing**: limited knowledge of particular words or phrases

   e.g., Chatbots, information retrieval, Web searching

2. **Limited linguistic ability**: appropriate response to simple, highly constrained sentences

   e.g., database queries in NL, simple NL interfaces

3. **Full text comprehension**: multi-sentence text and its relation to the real world

   e.g., conversational dialogue, automatic knowledge acquisition

4. **Emotional understanding/generation**

   e.g., responding to literature, poetry, story narration

Understanding

Why is understanding hard?

– Ambiguity: mapping is one-to-many

– Rich structures than strings: often hierarchical or scope-bearing

– Strong expressiveness: mapping from surface form to meaning is many-to-one

Debate: empiricism vs. rationalism

empiricism argued that it is possible to reliably learn correct context-free grammar

rationalism argued that it is not possible to understand a language by something like ChatGPT

– Chomsky argued that there must be an innate universal grammar that all children have from birth

Did LLMs show the emergencies of intelligence w/o understanding??

Understanding

Goal: a scientific theory of communication by language

- To understand the structure of language and its use as a complex computational system
- To develop the data structures and algorithms that can implement that system

Long way to go

The dream*

Trend: LLMs based on deep learning are a foundation model for NLP, but not for NLU

since deep learning and LLMs are uninterpretable yet

Deep learning: models of how children learn their language just from what they hear and observe

– apply machine learning to show how children can learn

– to map words in a sentence to real-world objects

– the relation between verbs and their arguments ⇐ Understanding??

The dream: "the linguistic computer"

Human-like competence in language ⇐ strong AI

Is LLMs possible to pass through the Turing test??