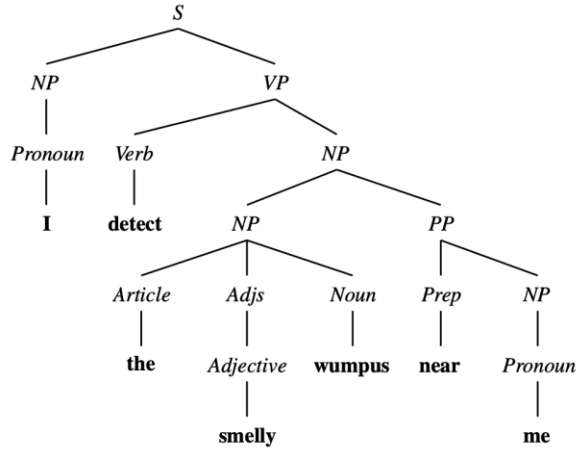
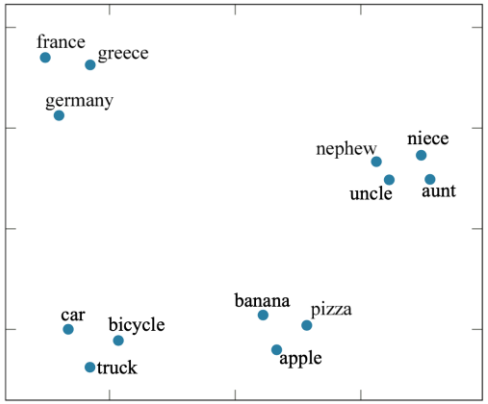
# Natural language processing




- Traditional approaches
  - Grammar, lexicon, parsing
  - Bag of word, n-grams matching
  - Tree-based parsing

- Deep learning approaches



# Why natural language processing?

- To **communicate** with humans. In many situations it is convenient for humans to use speech to interact with computers, and in most situations it is more convenient to use natural language rather than a formal language such as first-order predicate calculus.
- To **learn**. Humans have written down a lot of knowledge using natural language. Wikipedia alone has 30 million pages of facts such as "Bush babies are small nocturnal primates," whereas there are hardly any sources of facts like this written in formal logic. If we want our system to know a lot, it had better understand natural language.
- To advance the **scientific understanding** of languages and language use, using the tools of AI in conjunction with linguistics, cognitive psychology, and neuroscience.

# Why natural language processing?

- AI tasks that require NLP:
  - Speech recognition (from sound to text)
  - Text to speech synthesis
  - Machine translation
  - Information extraction (e.g., fill a structured database entry from an email or web page)
  - Information retrieval (find documents relevant or important to a query)
  - Question answering
  - Human-computer interaction
  - Etc

# Formal languages

- A grammar defines
  - Syntax (form of legal sentences)

- Semantic rules
  - Define meaning

- E.g., first-order logic

# Issues with natural languages

- Language judgments vary from person to person and time to time. Everyone agrees that "Not to be invited is sad" is a grammatical sentence of English, but people disagree on the grammaticality of "To be not invited is sad."
- Natural language is **ambiguous** ("He saw her duck" can mean either that she owns a waterfowl, or that she made a downwards evasive move) and **vague** ("That's great!" does not specify precisely how great it is, nor what it is).
- The mapping from symbols to objects is not formally defined. In first-order logic, two uses of the symbol "Richard" must refer to the same person, but in natural language two occurrences of the same word or phrase may refer to different things in the world.

# Language model

- Probability distribution describing the likelihood of any string

- E.g.,
  - "Do I dare disturb the universe?" should have much higher probability than
  - "Universe dare the I disturb do?"

# Bag of words model

- Goal: capture meaning of a fragment of text by counting which words appear in it.

- E.g., categorize news headlines into different sections on a web page
  - "Stocks rallied on Monday, with major indexes gaining 1% as optimism persisted over the first quarter earnings season." -> **business** section
  - "Heavy rain continued to pound much of the east coast on Monday, with flood warnings issued in New York City and other locations." -> **weather** section

# Bag of words model

- Formulate as a classification problem,
- then use a Naïve Bayes classifier (returns the class with highest probability):

$$\mathbf{P}(Class|w_{1:N}) = \alpha\, \mathbf{P}(Class) \prod_j \mathbf{P}(w_j|Class).$$

Sequence of words

Each word

- Learn prior probabilities on a large collection **(corpus)** of text.
- Complication: what is a word? E.g., aren't vs. are / n't vs. aren / ' / t

# N-gram model

- Bag of word completely ignores word order, but that usually affects semantics and syntax.
- N-gram model: focus on probability of a word given n preceding words

$$P\left(w_{1:N}\right) = \prod_{j=1}^{N} P\left(w_j | w_{1:j-1}\right)$$

Sequence of words

Each word given those before it

- Skip-gram model: same idea but skip some preceding words (e.g., every other word)

# N-gram model

- How to chose N?
  - with a vocabulary of 100,000=$10^5$ words,
  - and a sentence length of N=40,
  - this model would have $10^{200}$ parameters to estimate.

- Markov assumption: only consider a small preceding context (small n).

$$P(w_j|w_{1:j-1}) = P(w_j|w_{j-n+1:j-1})$$ Depends only on n-1 previous words

$$P(w_{1:N}) = \prod_{j=1}^{N} P(w_j|w_{j-n+1:j-1})$$

n=1: unigram
n=2: digram
n=3: trigram
etc

# N-gram model smoothing

- Some n-grams are very frequent, e.g., "of the"
  - Want to reduce their influence on classification
- Some may be so rare that they were not in the training set and hence have probability 0
  - Use special <UNK> word
  - E.g., consider only the 50,000 most common words, treat all others as <UNK>
  - May still encounter at test time n-grams that were never seen in training, e.g., "colorless aquamarine ideas" -> **smoothing**
  - Laplace smoothing: add 1 count to every n-gram
  - Back-off smoothing: reduce n for infrequent n-grams
  - Linear interpolation smoothing: $\widehat{P}\left(c_i|c_{i-2:i-1}\right) = \lambda_3 P\left(c_i|c_{i-2:i-1}\right) + \lambda_2 P\left(c_i|c_{i-1}\right) + \lambda_1 P\left(c_i\right)$
  - etc

$$\lambda_3 + \lambda_2 + \lambda_1 = 1$$

# N-gram model

Many applications

- News headlines grouping
- Spam detection
- Sentiment analysis
- Author attribution
- Etc

# Analyzing structure

N-gram model is atomic.

May miss correct but previously unseen phrases, e.g., "the fulvous kitten" because adjective "fulvous" (brownish yellow) was not in the training corpus.

We could infer that fulvous is likely an adjective given its position in the phrase and because it ends in **–ous** as do many adjectives.

➔ **Structured word model (dictionary)**

e.g., WordNet

```
"kitten" <noun.animal> ("young domestic cat") IS A: young_mammal

"kitten" <verb.body> ("give birth to kittens")
  EXAMPLE: "our cat kittened again this year"
```

# Grammar

A **grammar** is a set of rules that defines the tree structure of allowable phrases, and

a **language** is the set of sentences that follow those rules.

**Probabilistic context-free grammar (PCFG):** A probabilistic grammar that assigns a probability to each string, and "context-free" means that any rule can be used in any context.

e.g., the rules for a noun phrase at the beginning of a sentence are the same as for another noun phrase later in the sentence, and if the same phrase occurs in two locations, it must have the same probability each time.

# Grammar example

| | | | | |
|---|---|---|---|---|
| Sentence | $S$ | $\rightarrow$ | *NP VP* | [0.90] I + feel a breeze |
| | | \| | *S Conj S* | [0.10] I feel a breeze + and + It stinks |
| Noun phrase | $NP$ | $\rightarrow$ | *Pronoun* | [0.25] I |
| | | \| | *Name* | [0.10] Ali |
| | | \| | *Noun* | [0.10] pits |
| | | \| | *Article Noun* | [0.25] the + wumpus |
| | | \| | *Article Adjs Noun* | [0.05] the + smelly dead + wumpus |
| | | \| | *Digit Digit* | [0.05] 3 4 |
| | | \| | *NP PP* | [0.10] the wumpus + in 1 3 |
| | | \| | *NP RelClause* | [0.05] the wumpus + that is smelly |
| | | \| | *NP Conj NP* | [0.05] the wumpus + and + I |
| Verb phrase | $VP$ | $\rightarrow$ | *Verb* | [0.40] stinks |
| | | \| | *VP NP* | [0.35] feel + a breeze |
| | | \| | *VP Adjective* | [0.05] smells + dead |
| | | \| | *VP PP* | [0.10] is + in 1 3 |
| | | \| | *VP Adverb* | [0.10] go + ahead |
| | $Adjs$ | $\rightarrow$ | *Adjective* | [0.80] smelly |
| | | \| | *Adjective Adjs* | [0.20] smelly + dead |
| Prepositional phrase | $PP$ | $\rightarrow$ | *Prep NP* | [1.00] to + the east |
| | $RelClause$ | $\rightarrow$ | *RelPro VP* | [1.00] that + is smelly |

# Lexicon

A **lexicon** is a list of allowable words, i.e., the usable vocabulary.

| | | |
|---|---|---|
| *Noun* | → | **stench** [0.05] \| **breeze** [0.10] \| **wumpus** [0.15] \| **pits** [0.05] \| . . . |
| *Verb* | → | **is** [0.10] \| **feel** [0.10] \| **smells** [0.10] \| **stinks** [0.05] \| . . . |
| *Adjective* | → | **right** [0.10] \| **dead** [0.05] \| **smelly** [0.02] \| **breezy** [0.02] . . . |
| *Adverb* | → | **here** [0.05] \| **ahead** [0.05] \| **nearby** [0.02] \| . . . |
| *Pronoun* | → | **me** [0.10] \| **you** [0.03] \| **I** [0.10] \| **it** [0.10] \| . . . |
| *RelPro* | → | **that** [0.40] \| **which** [0.15] \| **who** [0.20] \| **whom** [0.02] \| . . . |
| *Name* | → | **Ali** [0.01] \| **Bo** [0.01] \| **Boston** [0.01] \| . . . |
| *Article* | → | **the** [0.40] \| **a** [0.30] \| **an** [0.10] \| **every** [0.05] \| . . . |
| *Prep* | → | **to** [0.20] \| **in** [0.10] \| **on** [0.05] \| **near** [0.10] \| . . . |
| *Conj* | → | **and** [0.50] \| **or** [0.10] \| **but** [0.20] \| **yet** [0.02] \| . . . |
| *Digit* | → | **0** [0.20] \| **1** [0.20] \| **2** [0.20] \| **3** [0.20] \| **4** [0.20] \| . . . |

# Parsing

**Parsing** is the process of analyzing a string of words to uncover its phrase structure, according to the rules of a grammar.

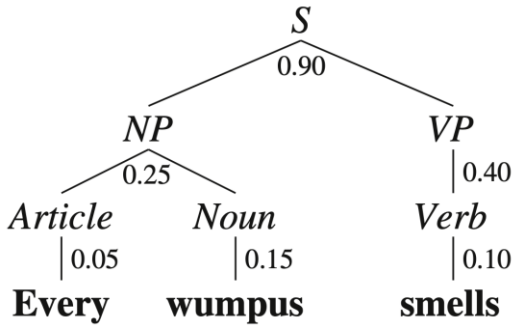
➔ search for a valid parse tree whose leaves are the words of the string.

Example: parsing using the grammar from previous slides:

"the Wumpus is dead"

| List of items | Rule |
|---|---|
| *S* | |
| *NP VP* | *S* → *NP VP* |
| *NP VP Adjective* | *VP* → *VP Adjective* |
| *NP Verb Adjective* | *VP* → *Verb* |
| *NP Verb* **dead** | *Adjective* → **dead** |
| *NP* **is dead** | *Verb* → **is** |
| *Article Noun* **is dead** | *NP* → *Article Noun* |
| *Article* **wumpus is dead** | *Noun* → **wumpus** |
| **the wumpus is dead** | *Article* → **the** |

"Every Wumpus smells"



# CYK parsing algorithm (Ali Cocke, Daniel Younger, and Tadeo Kasami)

Requires a grammar with all rules in one of two very specific formats:

lexical rules of the form $X \rightarrow \mathbf{word}\ [p]$

and syntactic rules of the form $X \rightarrow Y\ Z\ [p]$

with exactly two categories on the right-hand side.

This grammar format, called **Chomsky Normal Form**, may seem restrictive, but it is not:

any context-free grammar can be automatically transformed into Chomsky Normal Form (see https://en.wikipedia.org/wiki/Chomsky_normal_form )

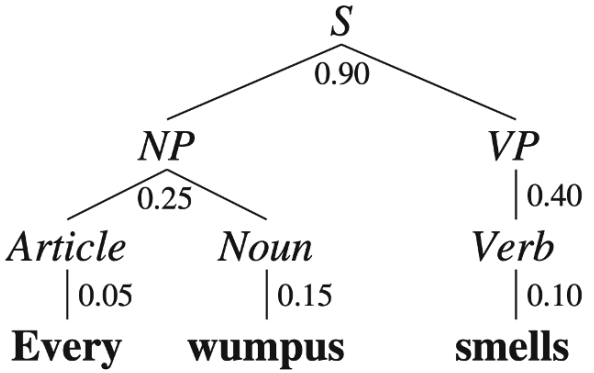
CYK parsing algorithm (Ali Cocke, Daniel Younger, and Tadeo Kasami)

```
function CYK-PARSE(words, grammar) returns a table of parse trees
  inputs: words, a list of words
          grammar, a structure with LEXICALRULES and GRAMMARRULES
  T ← a table        // T[X, i, k] is most probable X tree spanning words_{i:k}
  P ← a table, initially all 0        // P[X, i, k] is probability of tree T[X, i, k]
  // Insert lexical categories for each word.
  for i = 1 to LEN(words) do
    for each (X, p) in grammar.LEXICALRULES(words_i) do
      P[X, i, i] ← p
      T[X, i, i] ← TREE(X, words_i)
  // Construct X_{i:k} from Y_{i:j} + Z_{j+1:k}, shortest spans first.
  for each (i, j, k) in SUBSPANS(LEN(words)) do
    for each (X, Y, Z, p) in grammar.GRAMMARRULES do
      PYZ ← P[Y, i, j] × P[Z, j+1, k] × p
      if PYZ > P[X, i, k] do
        P[X, i, k] ← PYZ
        T[X, i, k] ← TREE(X, T[Y, i, j], T[Z, j + 1, k])
  return T

function SUBSPANS(N) yields (i, j, k) tuples
  for length = 2 to N do
    for i = 1 to N + 1 − length do
      k ← i + length − 1
      for j = i to k − 1 do
        yield (i, j, k)
```

Returns all (i,j,k) tuples with i<=j<k, by increasing length

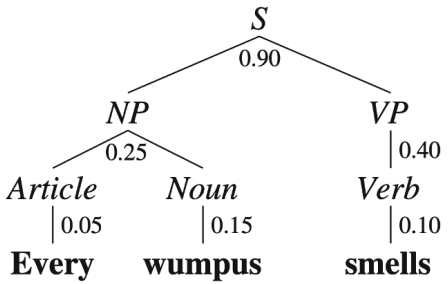
"Every Wumpus smells"



# CYK parsing algorithm (Ali Cocke, Daniel Younger, and Tadeo Kasami)

The CYK algorithm for parsing. Given a sequence of words, it finds the most probable parse tree for the sequence and its subsequences. The table $P[X, i, k]$ gives the probability of the most probable tree of category $X$ spanning $words_{i:k}$. The output table $T[X, i, k]$ contains the most probable tree of category $X$ spanning positions $i$ to $k$ inclusive. The function SUBSPANS returns all tuples $(i, j, k)$ covering a span of $words_{i:k}$, with $i \leq j < k$, listing the tuples by increasing length of the $i : k$ span, so that when we go to combine two shorter spans into a longer one, the shorter spans are already in the table. LEXICALRULES(*word*) returns a collection of $(X, p)$ pairs, one for each rule of the form $X \rightarrow word$ [p], and GRAMMARRULES gives $(X, Y, Z, p)$ tuples, one for each grammar rule of the form $X \rightarrow Y\ Z$ $[p]$.

"Every Wumpus smells"



https://courses.engr.illinois.edu/cs373/sp2009/lectures/lect_15.pdf

# CYK parsing example

Parse "Jeff trains  geometry students"

grammar

$S \rightarrow N\ V_P$
$N \rightarrow N\ N$
$V_P \rightarrow V\ N$
$N \rightarrow$ `students` | `Jeff` | `geometry` | `trains`
$V \rightarrow$ `trains`

(note: probabilities omitted for simplicity)

| len | | | | |
|---|---|---|---|---|
| 4 | Jeff trains geometry students | | | |
| 3 | Jeff trains geometry | trains geometry students | | |
| 2 | Jeff trains | trains geometry | geometry students | |
| 1 | Jeff | trains | geometry | students |
| | Jeff | trains | geometry | students |

first word in substring

https://courses.engr.illinois.edu/cs373/sp2009/lectures/lect_15.pdf

# CYK parsing example

CYK builds a table containing a cell for each substring. The cell for a substring $x$ contains a list of variables $V$ from which we can derive $x$ (in one or more steps).

| length | | | | |
|---|---|---|---|---|
| 4 | | | | |
| 3 | | | | |
| 2 | | | | |
| 1 | | | | |
| | Jeff | trains | geometry | students |

first word in substring

$S \rightarrow N\ V_P$
$N \rightarrow N\ N$
$V_P \rightarrow V\ N$
$N \rightarrow$ `students` | `Jeff` | `geometry` | `trains`
$V \rightarrow$ `trains`

grammar

The bottom row contains the variables that can derive each substring of length 1. This is easy to fill in:

| length | | | | |
|---|---|---|---|---|
| 4 | | | | |
| 3 | | | | |
| 2 | | | | |
| 1 | N | N,V | N | N |
| | Jeff | trains | geometry | students |

first word in substring

https://courses.engr.illinois.edu/cs373/sp2009/lectures/lect_15.pdf

# CYK parsing example

Now we fill the table row-by-row, moving upwards. To fill in the cell for a 2-word substring $x$, we look at the labels in the cells for its two constituent words and see what rules could derive this pair of labels. In this case, we use the rules N → N N and $V_P$ → V N to produce:

| length | | | | | |
|---|---|---|---|---|---|
| | 4 | | | | |
| | 3 | | | | |
| | 2 | N | N,$V_P$ | N | |
| | 1 | N | N,V | N | N |
| | | Jeff | trains | geometry | students |

first word in substring

$S \rightarrow N\ V_P$
$N \rightarrow N\ N$
$V_P \rightarrow V\ N$
N → `students` | `Jeff` | `geometry` | `trains`
V → `trains`

grammar

https://courses.engr.illinois.edu/cs373/sp2009/lectures/lect_15.pdf

# CYK parsing example

For each longer substring $x$, we have to consider all the ways to divide $x$ into two shorter substrings. For example, suppose $x$ is the substring of length 3 starting with "trains". This can be divided into divided into (a) "trains geometry" plus "students" or (b) "trains" plus "geometry students."

Consider option (a). Looking at the lower rows of the table, "students" has label N. One label for "trains geometry" is $\mathsf{V_P}$, but we don't have any rule whose righthand side contains $\mathsf{V_P}$ followed by N. The other label for "trains geometry" is N. In this case, we find the rule $\mathsf{N} \rightarrow \mathsf{N\ N}$. So one label for $x$ is N. (That is, $x$ is one big long compound noun.)

Now consider option (b). Again, we have the possibility that both parts have label N. But we also find that "trains" could have the label V. We can then apply the rule $\mathsf{V_P} \rightarrow \mathsf{V\ N}$ to add the label $\mathsf{V_P}$ to the cell for $x$.

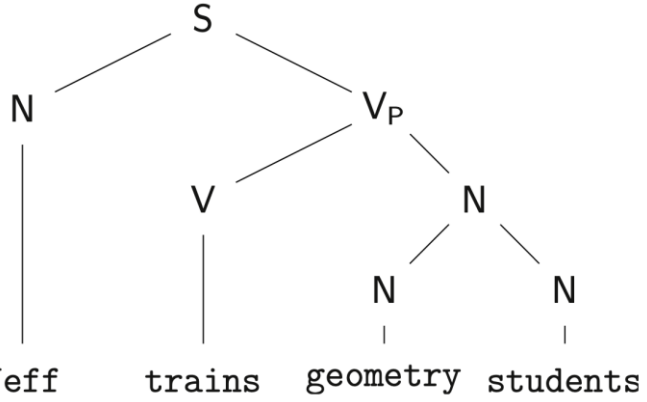
| length | | first word in substring | | | |
|---|---|---|---|---|---|
| 4 | | | | | |
| 3 | | N,$\mathsf{V_P}$ | | | |
| 2 | N | N,$\mathsf{V_P}$ | N | | |
| 1 | N | N,V | N | N | |
| | Jeff | trains | geometry | students | |

https://courses.engr.illinois.edu/cs373/sp2009/lectures/lect_15.pdf

# CYK parsing example

Repeating this procedure for the remaining two cells, we get:

| length | | | | | |
|---|---|---|---|---|---|
| | 4 | N,S | | | |
| | 3 | N,S | $N,V_P$ | | |
| | 2 | N | $N,V_P$ | N | |
| | 1 | N | N,V | N | N |
| | | Jeff | trains | geometry | students |
| | | first word in substring | | | |



# CYK parsing algorithm (Ali Cocke, Daniel Younger, and Tadeo Kasami)

**Space complexity: $O(n^2m)$** for the P and T tables, where n is the number of words in the sentence, and m is the number of nonterminal symbols in the grammar.

**Time complexity: $O(n^3m)$**

Guaranteed to work for all possible context-free grammars.

But natural languages have evolved to be easy to understand in real time, not to be as tricky as possible, so it seems that they should be amenable to a faster parsing algorithm (e.g., a modified A* search)

## Complications of natural language processing

The grammar of real English is endlessly complex (and other languages too).

**Quantification:** Example: "Every agent feels a breeze."

$$\forall a \; a \in Agents \;\Rightarrow\; \exists b \; b \in Breezes \wedge Feel(a, b);$$

or

$$\exists b \; b \in Breezes \wedge \forall a \; a \in Agents \;\Rightarrow\; Feel(a, b).$$

## Complications of natural language processing

The grammar of real English is endlessly complex (and other languages too).

**Quantification:** Example: "Every agent feels a breeze."

$$\forall a \; a \in Agents \;\Rightarrow\; \exists b \; b \in Breezes \wedge Feel(a, b);$$

each agent feels its own breeze

or

$$\exists b \; b \in Breezes \wedge \forall a \; a \in Agents \;\Rightarrow\; Feel(a, b).$$

all agents feel the same breeze

## Complications of natural language processing

**Pragmatics:** Example: "I am in Boston today."

Need context information to fill-in the details:

- who is "I"
- what day is "today"
- etc

**Long-distance dependencies:** Example "she didn't hear or even see him"

- who/what did she not hear? Missing word…
- use special symbols to represent them:

"she didn't [hear ␣ or even see ␣] him."

## Complications of natural language processing

**Time and tense:** Example: "Ali loves Bo" vs. "Ali loved Bo"

In FOL: no ambiguity using event calculus

events

$$\text{Ali loves Bo: } E_1 \in Loves(Ali,\ Bo) \ \wedge During(Now,\ Extent(E_1))$$
$$\text{Ali loved Bo: } E_2 \in Loves(Ali, Bo) \ \wedge After(Now, Extent(E_2)).$$

This suggests that our two lexical rules for the words "loves" and "loved" should be these:

$$Verb\,(\lambda y\ \lambda x\ e \in Loves\,(x,\ y) \wedge During\,(Now,\ e)) \rightarrow \textbf{loves}$$
$$Verb\,(\lambda y\ \lambda x\ e \in Loves\,(x,\ y) \wedge After\,(Now,\ e)) \rightarrow \textbf{loved}.$$

# Complications of natural language processing

**Ambiguity:** not necessarily deficiencies, rather efficiencies!

Squad helps dog bite victim.

Police begin campaign to run down jaywalkers.

Helicopter powered by human flies.

Once-sagging cloth diaper industry saved by full dumps.

Include your children when baking cookies.

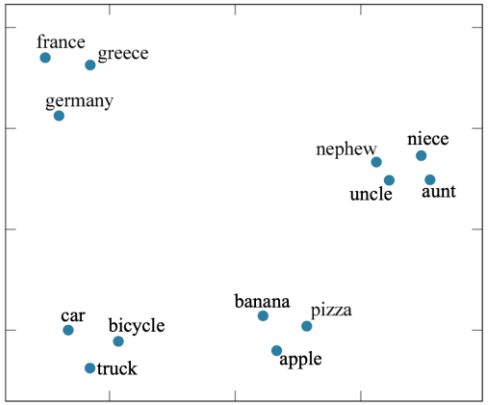
Portable toilet bombed; police have nothing to go on.

Milk drinkers are turning to powder.

Two sisters reunited after 18 years in checkout counter.

# Deep learning for natural language processing

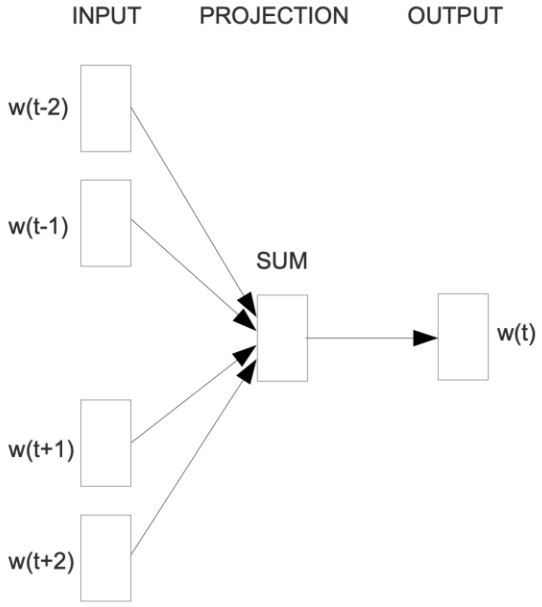
**General goal:** learn a representation ("embedding") of words, without requiring manual feature engineering, which allows for generalization across related words:




- syntactically ("colorless" and "ideal" are both adjectives),
- semantically ("cat" and "kitten" are both felines),
- topically ("sunny" and "sleet" are both weather terms),
- in terms of sentiment ("awesome" has opposite sentiment to "cringeworthy"),
- or otherwise.

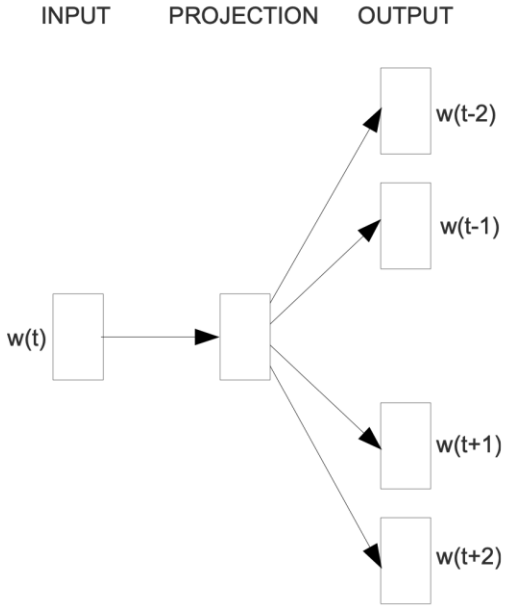
Then analyze sequences of words in the embedding space.

# Word2vec




**CBOW**

Continuous bag of words




**Skip-gram**

Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

# Word2vec

Let $V$ ("vocabulary") be the set of all words appearing in the corpus $C$. Our goal is to learn one vector $v_w \in \mathbb{R}^n$ for each word $w \in V$.

The idea of skip-gram is that the vector of a word should be close to the vector of each of its neighbors. The idea of CBOW is that the vector-sum of a word's neighbors should be close to the vector of the word.

## Continuous Bag of Words (CBOW)

Suppose we want each word in the corpus to be predicted by every other word in a small span of 4 words. We write the neighbor set $N = \{-4, -3, -2, -1, +1, +2, +3, +4\}$.

Then the training objective is to maximize the following quantity:

$$\prod_{i \in C} Pr(w_i | w_j : j \in N + i)$$

That is, we want to maximize the total probability for the corpus, as seen by a probability model that uses word neighbors to predict words.

# Word2vec

Our probability model is as follows: Given words $\{w_j : j \in N + i\}$, it takes their vector sum $v := \sum_{j \in N+i} v_{w_j}$, then take the dot-product-softmax with every other vector sum (this step is similar to the attention mechanism in Transformers), to obtain the probability:

$$Pr(w|w_j : j \in N + i) := \frac{e^{v_w \cdot v}}{\sum_{w \in V} e^{v_w \cdot v}}$$

## Skip-gram

For skip-gram, the training objective is

$$\prod_{i \in C} Pr(w_j : j \in N + i | w_i)$$

That is, we want to maximize the total probability for the corpus, as seen by a probability model that uses words to predict its word neighbors. We predict each word-neighbor independently, thus $Pr(w_j : j \in N + i | w_i) = \prod_{j \in N+i} Pr(w_j | w_i)$.

Arithmetic on word embeddings

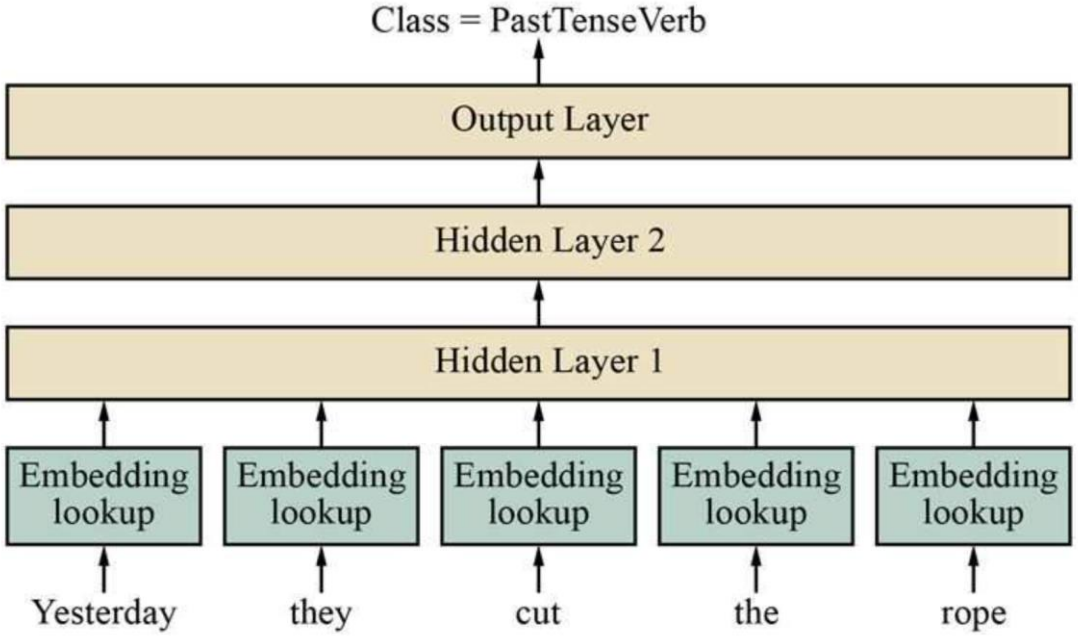
| A | B | C | D = C + (B − A) | Relationship |
|---|---|---|---|---|
| Athens | Greece | Oslo | Norway | *Capital* |
| Astana | Kazakhstan | Harare | Zimbabwe | *Capital* |
| Angola | kwanza | Iran | rial | *Currency* |
| copper | Cu | gold | Au | *Atomic Symbol* |
| Microsoft | Windows | Google | Android | *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* |

**Figure 24.2** A word embedding model can sometimes answer the question "**A** is to **B** as **C** is to [what]?" with vector arithmetic: given the word embedding vectors for the words **A**, **B**, and **C**, compute the vector $\mathbf{D} = \mathbf{C} + (\mathbf{B} - \mathbf{A})$ and look up the word that is closest to **D**. (The answers in column **D** were computed automatically by the model. The descriptions in the "Relationship" column were added by hand.) Adapted from ? (?, ?).

# Simplest DNNs for NLP: convnets

**Part-of-speech tagging:**

assign a lexical category or tag to each word: noun, verb, adjective, etc



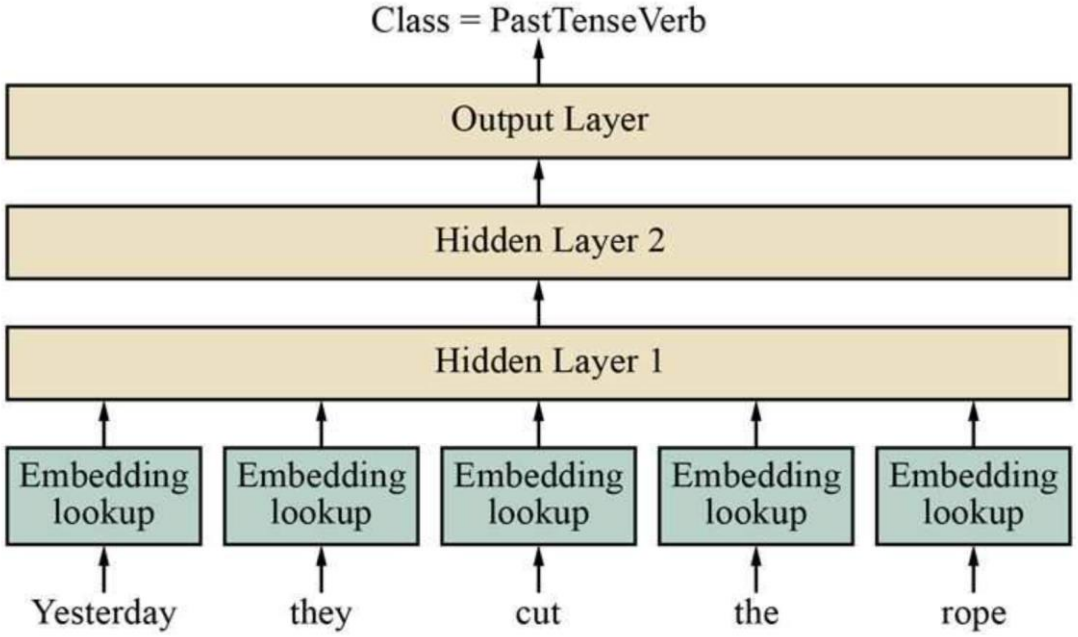

Feedforward part-of-speech tagging model. This model takes a 5-word window as input and predicts the tag of the word in the middle—here, *cut*. The model is able to account for word position because each of the 5 input embeddings is multiplied by a different part of the first hidden layer. The parameter values for the word embeddings and for the three layers are all learned simultaneously during training.

# Simplest DNNs for NLP: convnets

**Part-of-speech tagging:**

assign a lexical category or tag to each word: noun, verb, adjective, etc

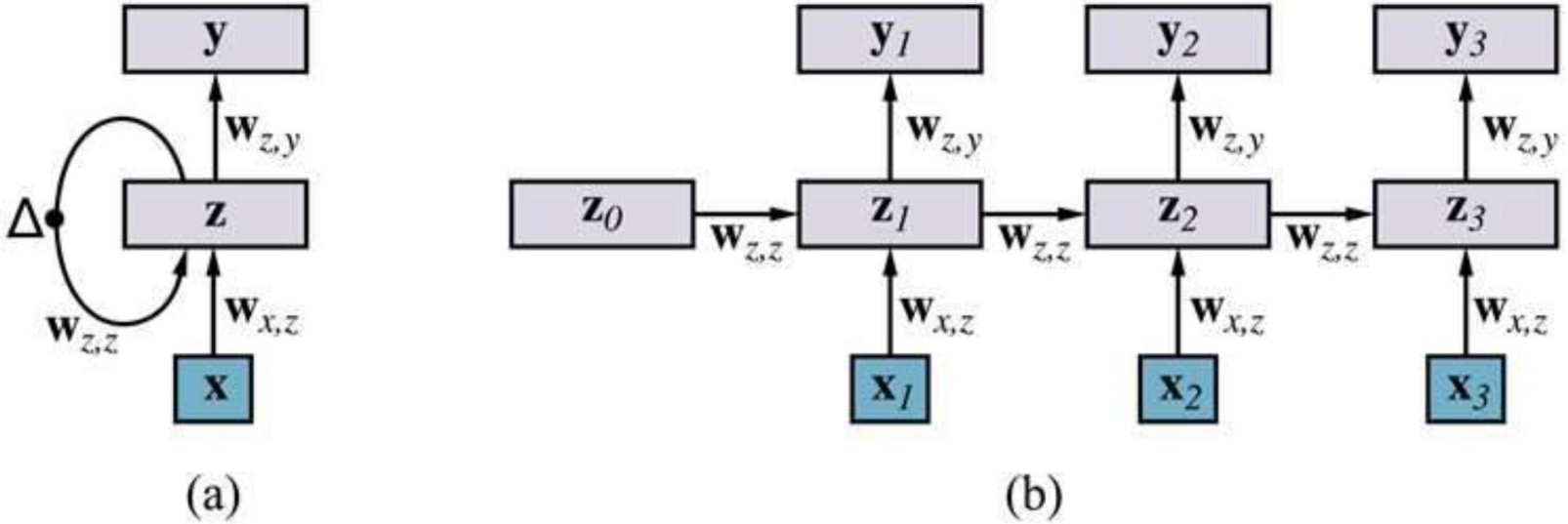
Weakness: limited, fixed-size context




Feedforward part-of-speech tagging model. This model takes a 5-word window as input and predicts the tag of the word in the middle—here, *cut*. The model is able to account for word position because each of the 5 input embeddings is multiplied by a different part of the first hidden layer. The parameter values for the word embeddings and for the three layers are all learned simultaneously during training.

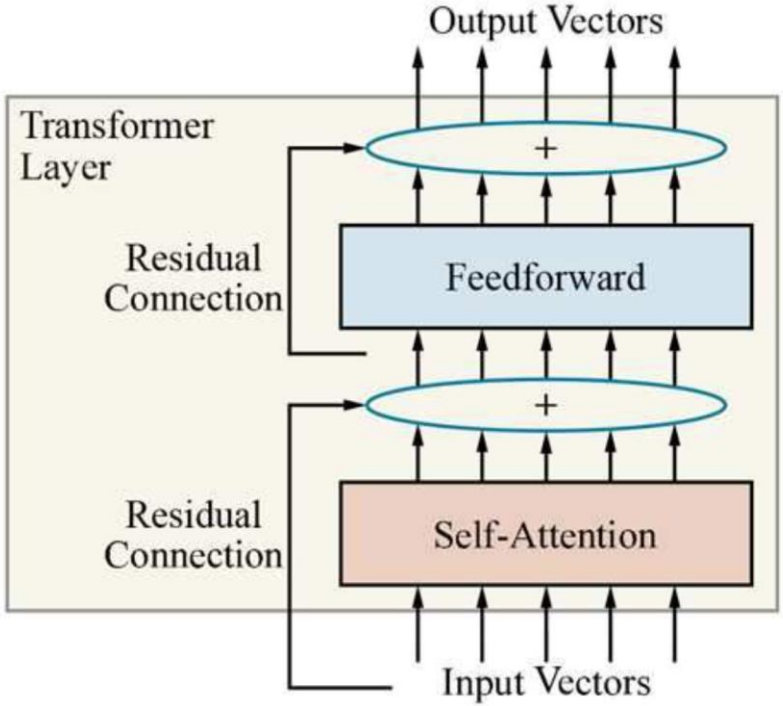
# “Traditional” approach: recurrent networks

## Solves the limited context problem



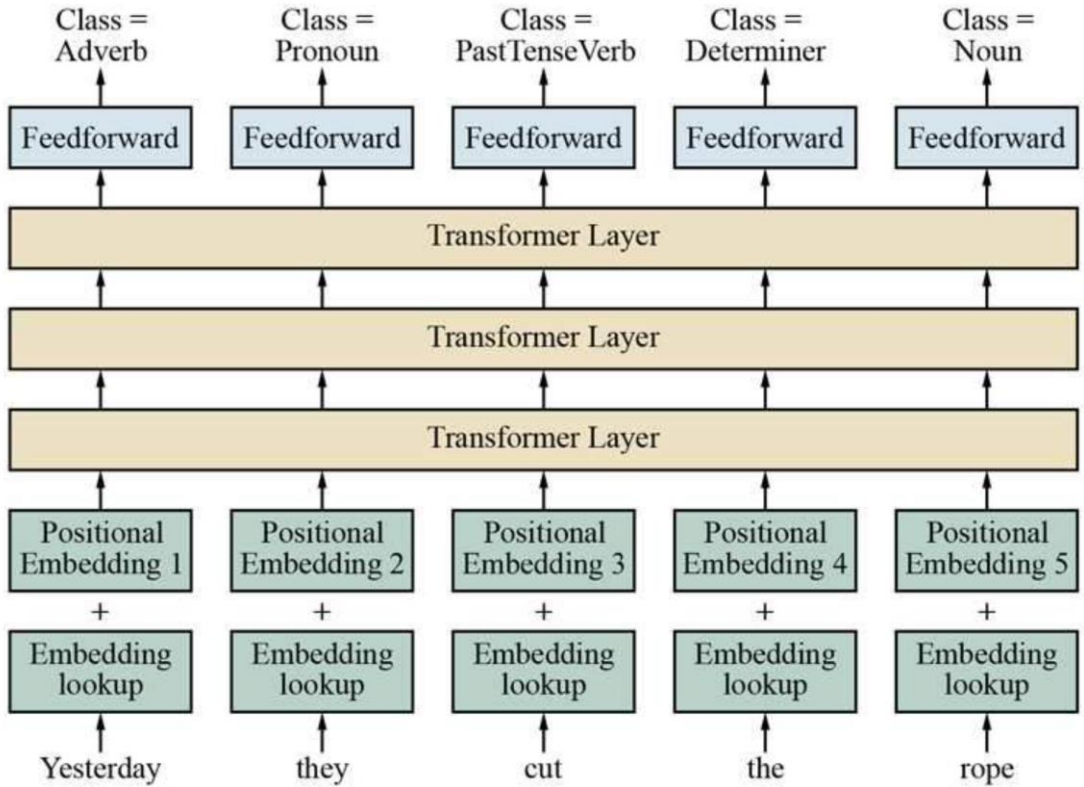

(a) Schematic diagram of an RNN where the hidden layer z has recurrent connections; the Δ symbol indicates a delay. Each input x is the word embedding vector of the next word in the sentence. Each output y is the output for that time step. (b) The same network unrolled over three timesteps to create a feedforward network. Note that the weights are shared across all timesteps.

# Latest: transformers



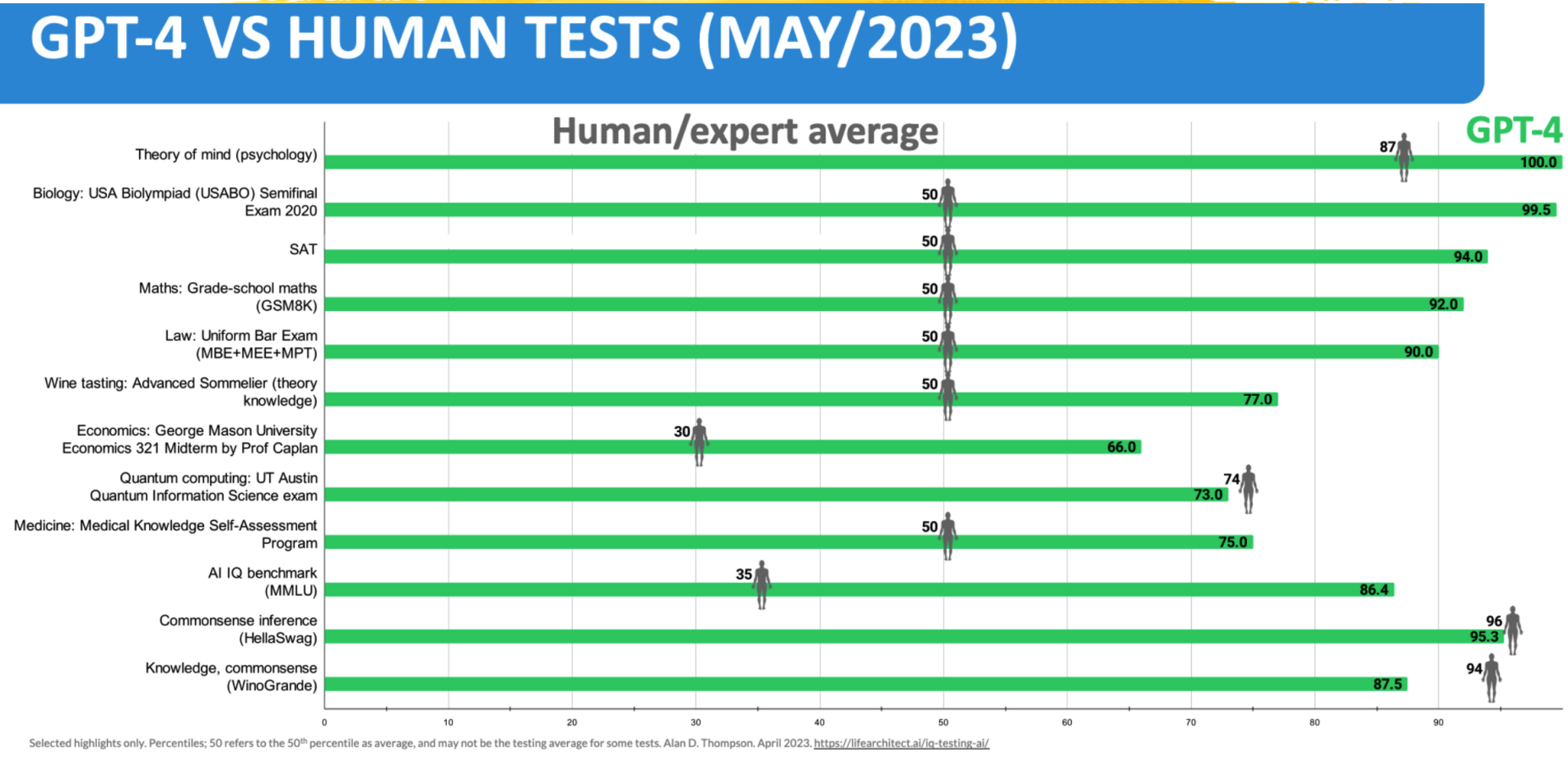

A single-layer transformer consists of self-attention, a feedforward network, and residual connections.

# Latest: transformers




Using the transformer architecture for POS tagging.

# Are LLMs close to achieving general intelligence?



| | Claude 3 Opus | Claude 3 Sonnet | Claude 3 Haiku | GPT-4 | GPT-3.5 | Gemini 1.0 Ultra | Gemini 1.0 Pro |
|---|---|---|---|---|---|---|---|
| Undergraduate level knowledge *MMLU* | **86.8%** 5 shot | 79.0% 5-shot | 75.2% 5-shot | 86.4% 5-shot | 70.0% 5-shot | 83.7% 5-shot | 71.8% 5-shot |
| Graduate level reasoning *GPQA, Diamond* | **50.4%** 0-shot CoT | 40.4% 0-shot CoT | 33.3% 0-shot CoT | 35.7% 0-shot CoT | 28.1% 0-shot CoT | — | — |
| Grade school math *GSM8K* | **95.0%** 0-shot CoT | 92.3% 0-shot CoT | 88.9% 0-shot CoT | 92.0% 5-shot CoT | 57.1% 5-shot | 94.4% Maj1@32 | 86.5% Maj1@32 |
| Math problem-solving *MATH* | **60.1%** 0-shot CoT | 43.1% 0-shot CoT | 38.9% 0-shot CoT | 52.9% 4-shot | 34.1% 4-shot | 53.2% 4-shot | 32.6% 4-shot |
| Multilingual math *MGSM* | **90.7%** 0-shot | 83.5% 0-shot | 75.1% 0-shot | 74.5% 8-shot | — | 79.0% 8-shot | 63.5% 8-shot |
| Code *HumanEval* | **84.9%** 0-shot | 73.0% 0-shot | 75.9% 0-shot | 67.0% 0-shot | 48.1% 0-shot | 74.4% 0-shot | 67.7% 0-shot |
| Reasoning over text *DROP, F1 score* | **83.1** 3-shot | 78.9 3-shot | 78.4 3-shot | 80.9 3-shot | 64.1 3-shot | 82.4 Variable shots | 74.1 Variable shots |
| Mixed evaluations *BIG-Bench-Hard* | **86.8%** 3-shot CoT | 82.9% 3-shot CoT | 73.7% 3-shot CoT | 83.1% 3-shot CoT | 66.6% 3-shot CoT | 83.6% 3-shot CoT | 75.0% 3-shot CoT |
| Knowledge Q&A *ARC-Challenge* | **96.4%** 25-shot | 93.2% 25-shot | 89.2% 25-shot | 96.3% 25-shot | 85.2% 25-shot | — | — |
| Common Knowledge *HellaSwag* | **95.4%** 10-shot | 89.0% 10-shot | 85.9% 10-shot | 95.3% 10-shot | 85.5% 10-shot | 87.8% 10-shot | 84.7% 10-shot |

https://www.anthropic.com/news/claude-3-family

# Are LLMs close to achieving general intelligence?

## AIs ranked by IQ

MAXIMUM TRUTH
MaximumTruth.org

| AI | IQ Score | Questions right (out of 35 per test) | Chance it beats random guessing |
|---|---|---|---|
| Claude-3 | 101 | 18.5 | 99.999999%+ |
| ChatGPT-4 | 85 | 13 | 99.9986% |
| Claude-2 | 82 | 12 | 99.9911% |
| Bing Copilot | 79 | 11 | 99.9314% |
| Gemini (normal) | 77.5 | 10.5 | 99.8212% |
| Gemini Advanced | 76 | 10 | 99.5894% |
| Grok | 68.5 | 7.5 | 87.9402% |
| Llama-2 (Meta) | 67 | 7 | 80.3278% |
| Claude-1 | 64 | 6 | 56.3155% |
| ChatGPT-3.5 | 64 | 6 | 56.3155% |
| Grok Fun | 64 | 6 | 56.3155% |
| Random Guesser | 63.5 | 5.8333 | 50% |

Tests given in March 2024. The IQ test was Mensa Norway, with all questions verbalized as if one were giving the test to a blind person. The right-hand column shows the % of random-guesser simulations that the AI did better than, with ties ignored, over 70 questions (two test administrations.)

https://www.maximumtruth.org/p/ais-ranked-by-iq-ai-passes-100-iq