The History of Language Models

Yu Meng
University of Virginia
yumeng5@virginia.edu

Nov 21, 2024

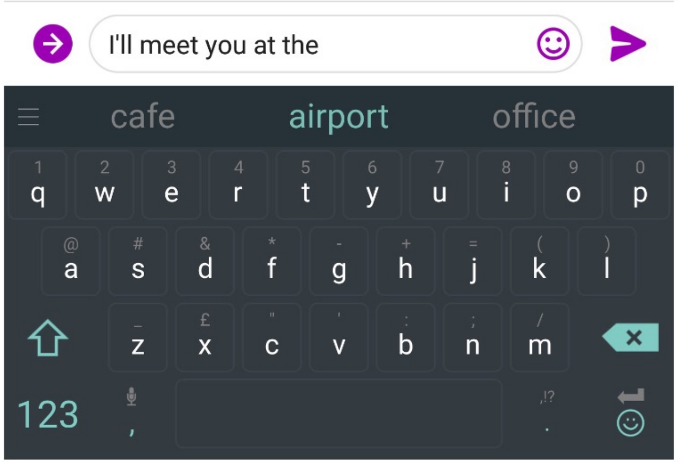
Agenda

- Introduction to Language Models
- Word Embeddings
- Sequence Modeling and Recurrent Neural Networks (RNNs)
- Transformer
- Large language models (LLMs)

UNIVERSITY of VIRGINIA

Overview: Language Modeling

- The core problem in NLP is **language modeling**
- Goal: Assigning probability to a sequence of words
- For text understanding: p("The cat is on the mat") >> p("Truck the earth on")
- For text generation: p(w | "The cat is on the") -> "mat"

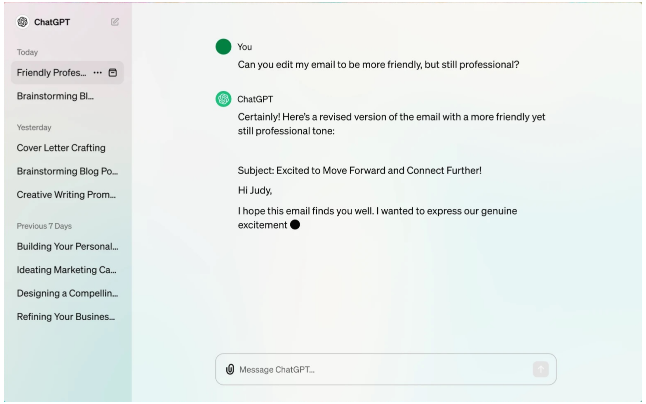


Autocomplete empowered by language modeling

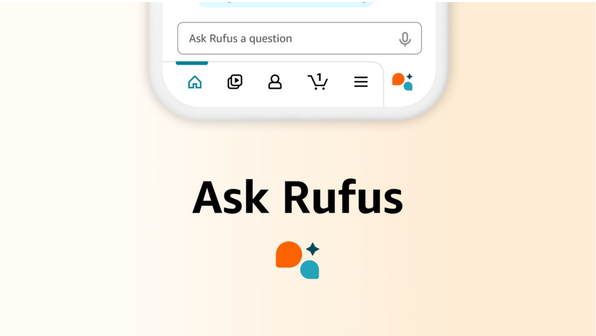
UNIVERSITY of VIRGINIA

Language Model Applications

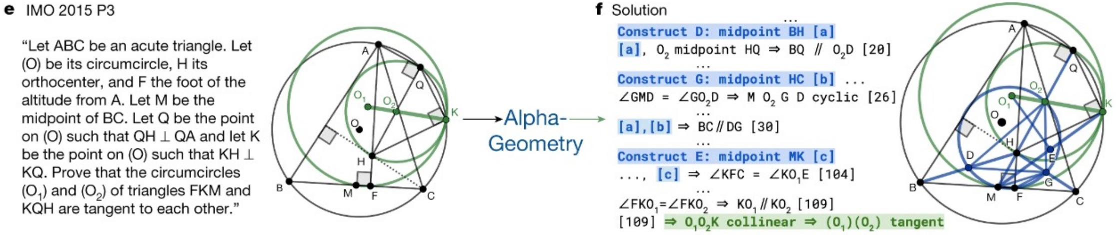
Chatbots



Code Assistants



Shopping Assistants



Generating Math Proofs

UNIVERSITY of VIRGINIA

Language Models = Universal NLP Task Solvers

- Every NLP task can be converted into a text-to-text task!
  - Sentiment analysis: The movie’s closing scene is attractive; it was ___ (good)
  - Machine translation: “Hello world” in French is ___ (Bonjour le monde)
  - Question answering: Which city is UVA located in? ___ (Charlottesville)
  - …
- All these tasks can be formulated as a language modeling problem!

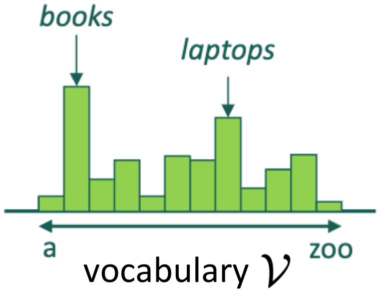
Language Modeling: Probability Decomposition

- Given a text sequence $\boldsymbol{x} = [x_1, x_2, \dots, x_n]$, how can we model $p(\boldsymbol{x})$?
- Autoregressive assumption: the probability of each word only depends on its previous tokens

$$p(\boldsymbol{x}) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2) \cdots p(x_n|x_1, \dots, x_{n-1}) = \prod_{i=1}^{n} p(x_i|x_1, \dots, x_{i-1})$$

Language Models Are Generative Models

- Suppose we have a language model that gives us the estimate of $p(w|x_1, \ldots, x_{i-1})$, we can generate the next tokens one-by-one!
- Sampling: $x_i \sim p(w|x_1, \ldots, x_{i-1})$
- Or greedily: $x_i \leftarrow \arg\max_w p(w|x_1, \ldots, x_{i-1})$
- But how do we know when to stop generation?
- Use a special symbol [EOS] (end-of-sequence) to denote stopping



Example: Language Models for Generation

- Recursively sample $x_i \sim p(w|x_1, \ldots, x_{i-1})$ until we generate [EOS]
- Generate the first word: $\text{“the”} \leftarrow x_1 \sim p(w|\boxed{\text{[BOS]}})$ beginning-of-sequence
- Generate the second word: $\text{“cat”} \leftarrow x_2 \sim p(w|\text{“the”})$
- Generate the third word: $\text{“is”} \leftarrow x_3 \sim p(w|\text{“the cat”})$
- Generate the fourth word: $\text{“on”} \leftarrow x_4 \sim p(w|\text{“the cat is”})$
- Generate the fifth word: $\text{“the”} \leftarrow x_5 \sim p(w|\text{“the cat is on”})$
- Generate the sixth word: $\text{“mat”} \leftarrow x_6 \sim p(w|\text{“the cat is on the”})$
- Generate the seventh word: $\text{[EOS]} \leftarrow x_7 \sim p(w|\text{“the cat is on the mat”})$
- Generation finished!

UNIVERSITY of VIRGINIA

How to Obtain A Language Model?

Learn the probability distribution $p(w|x_1, \ldots, x_{i-1})$ from a training corpus!

Donald Trump

242 languages

Article Talk Read View source View history Tools

Joe Biden

218 languages

Article Talk Read View source View history Tools

From Wikipedia, the free encyclopedia

"Joseph Biden" and "Biden" redirect here. For his first-born son, Joseph Biden III, see Beau Biden. For other uses, see Biden (disambiguation).

Joseph Robinette Biden Jr.[b] (born November 20, 1942) is an American politician serving as the 46th and current president of the United States since 2021. A member of the Democratic Party, he served as the 47th vice president from 2009 to 2017 under President Barack Obama and represented Delaware in the U.S. Senate from 1973 to 2009.

Born in Scranton, Pennsylvania, Biden moved with his family to Delaware in 1953. He graduated from the University of Delaware in 1965 and from Syracuse University in 1968. He was elected to the New Castle County Council in 1970 and the U.S. Senate in 1972. As a senator, Biden drafted and led the effort to pass the Violent Crime Control and Law Enforcement Act and the Violence Against Women Act. He also oversaw six U.S. Supreme Court confirmation hearings, including the contentious hearings for Robert Bork and Clarence Thomas. Biden ran unsuccessfully for the 1988 and 2008 Democratic presidential nominations. In 2008, Obama chose Biden as his running mate, and he was a close counselor to Obama during his two terms as vice president. In the 2020 presidential election, the Democratic Party nominated Biden for president. He selected Kamala Harris as his running mate, and they defeated Republican incumbents Donald Trump and Mike Pence. He is the oldest president in U.S. history and the first to have a female vice president.

As president, Biden signed the American Rescue Plan Act in response to the COVID-19 pandemic and subsequent recession. He signed bipartisan bills on infrastructure and manufacturing. He proposed the Build Back Better Act, which failed in Congress, but aspects of which were incorporated into the Inflation Reduction Act that he signed into law in 2022. Biden appointed Ketanji Brown Jackson to the Supreme Court. He worked with congressional Republicans to resolve the 2023 debt-ceiling crisis by negotiating a deal to raise the debt ceiling. In foreign policy, Biden restored America's membership in the Paris Agreement. He oversaw the complete withdrawal of U.S. troops from Afghanistan that ended the war in Afghanistan, leading to the collapse of the Afghan government and the Taliban seizing control. He responded to the Russian invasion of

Joe Biden	
Official portrait, 2021	
46th President of the United States	
Incumbent	
Assumed office January 20, 2021	
Vice President	Kamala Harris
Preceded by	Donald Trump
47th Vice President of the United States	
In office January 20, 2009 – January 20, 2017	
President	Barack Obama
Preceded by	Dick Cheney
Succeeded by	Mike Pence
United States Senator from Delaware	
In office January 3, 1973 – January 15, 2009	
Preceded by	J. Caleb Boggs

Trump

trait, 2017

the United States

fice

- January 20, 2021

ence

Obama

den

l details

l John Trump

4, 1946 (age 78)

s, New York City, U.S.

lican (1987–1999, 2011, 2012–present)

(1999–2001)

ratic (2001–2009)

ndent (2011–2012)

Zelníčková

77; div. 1990)

Learning target:

$$p(w|x_1, \ldots, x_{i-1})$$

Text corpora contain rich distributional statistics!

History of Language Models

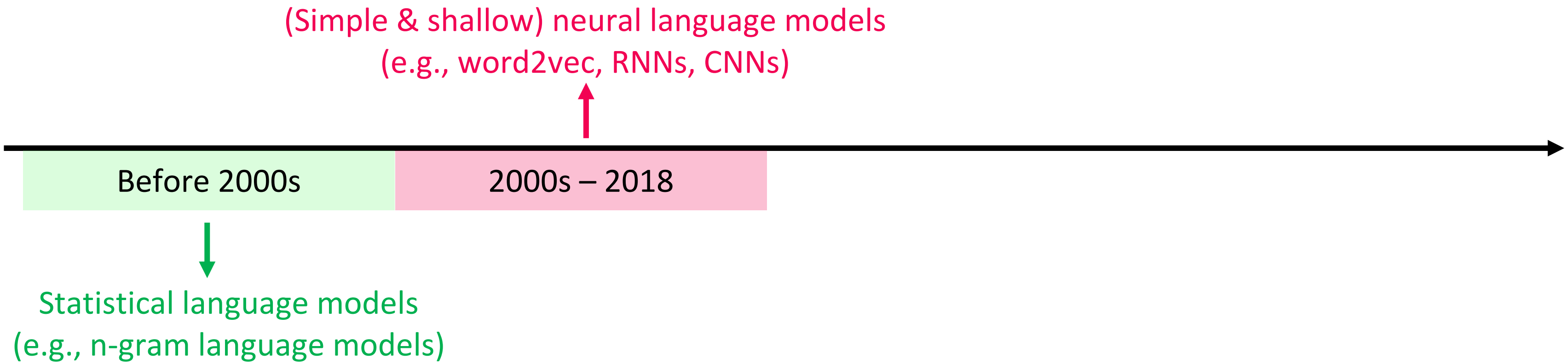
- Language models started to be built with statistical methods
  - Sparsity
  - Poor generalization

Before 2000s

Statistical language models
(e.g., n-gram language models)

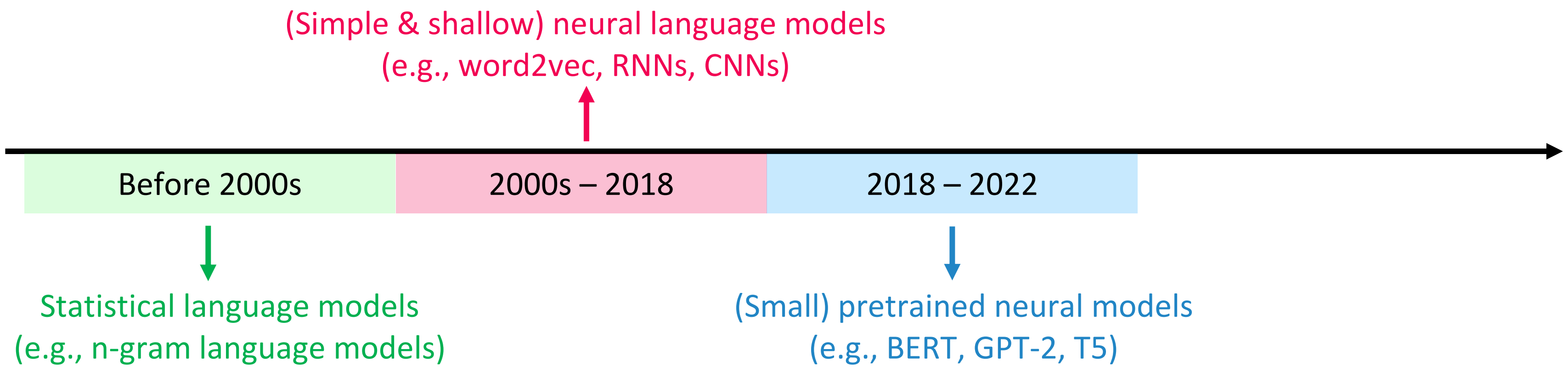
History of Language Models

- The introduction of neural networks into language models mitigated sparsity and improved generalization
  - Neural networks for language models were small-scale and inefficient for a long time
  - Task-specific architecture designs required for different NLP tasks
  - These language models were trained on individual NLP tasks as task-specific solvers



History of Language Models

- Transformer became the dominant architecture for language modeling; scaling up model sizes and (pretraining) data enabled significant generalization ability
  - Transformer demonstrated striking scalability and efficiency in sequence modeling
  - One pretrained model checkpoint fine-tuned to become strong task-specific models
  - Task-specific fine-tuning was still necessary



History of Language Models

- Generalist large language models (LLMs) became the universal task solvers and replaced task-specific language models
  - Real-world NLP applications are usually multifaceted (require composite task abilities)
  - Tasks are not clearly defined and may overlap
  - Single-task models struggle to handle complex tasks

(Simple & shallow) neural language models
(e.g., word2vec, RNNs, CNNs)

Large language models
(e.g., ChatGPT, GPT-4)

Before 2000s	2000s – 2018	2018 – 2022	2022 – Now

Statistical language models
(e.g., n-gram language models)

(Small) pretrained neural models
(e.g., BERT, GPT-2, T5)

Agenda

- Introduction to Language Models
- Word Embeddings
- Sequence Modeling and Recurrent Neural Networks (RNNs)
- Transformer
- Large language models (LLMs)

Vector Semantics

- Represent a word as a point in a multi-dimensional semantic space
- A desirable vector semantic space: words with similar meanings are nearby in space

2D visualization of a desirable high-dimensional vector semantic space

Figure source: https://web.stanford.edu/~jurafsky/slp3/6.pdf

UNIVERSITY of VIRGINIA

How to Represent Words as Vectors?

- Given a vocabulary $\mathcal{V} = \{\text{good}, \text{feel}, \text{I}, \text{sad}, \text{cats}, \text{have}\}$
- Most straightforward way to represent words as vectors: use their indices
- One-hot vector: only one high value (1) and the remaining values are low (0)
- Each word is identified by a unique dimension
- Issue? Fail to capture word semantics!

$$
\begin{aligned}
\boldsymbol{v}_{\text{good}} &= [1, 0, 0, 0, 0, 0] \\
\boldsymbol{v}_{\text{feel}} &= [0, 1, 0, 0, 0, 0] \\
\boldsymbol{v}_{\text{I}} &= [0, 0, 1, 0, 0, 0] \\
\boldsymbol{v}_{\text{sad}} &= [0, 0, 0, 1, 0, 0] \\
\boldsymbol{v}_{\text{cats}} &= [0, 0, 0, 0, 1, 0] \\
\boldsymbol{v}_{\text{have}} &= [0, 0, 0, 0, 0, 1]
\end{aligned}
$$

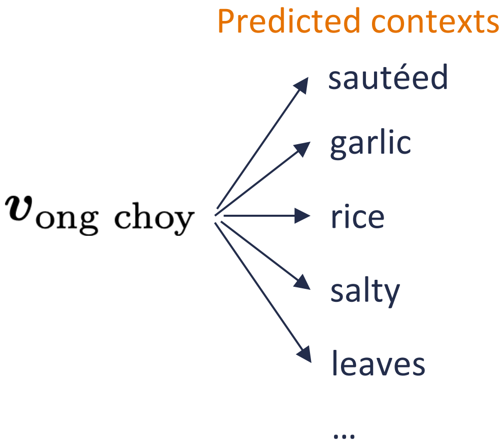
Distributional Hypothesis

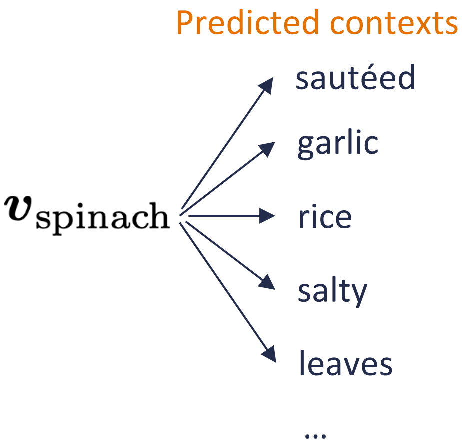
- Words that occur in similar contexts tend to have similar meanings
- A word's meaning is largely defined by the company it keeps (its context)
- Example: suppose we don't know the meaning of "Ong choy" but see the following:
  - Ong choy is delicious **sautéed with garlic**
  - Ong choy is superb **over rice**
  - … ong choy **leaves** with **salty** sauces
- And we've seen the following contexts:
  - … spinach **sautéed with garlic over rice**
  - … chard stems and **leaves** are **delicious**
  - … collard greens and other **salty** leafy greens
- Ong choy = water spinach!

Example source: https://web.stanford.edu/~jurafsky/slp3/slides/vectorsemantics2024.pdf

Learning Word Embeddings

- Assume a large text collection (e.g., Wikipedia)
- Hope to learn similar word embeddings for words occurring in similar contexts
- Construct a prediction task: use a center word's embedding to predict its contexts!
- Intuition: If two words have similar embeddings, they will predict similar contexts, thus being semantically similar!





Word Embedding Is Self-Supervised Learning

- **Self-supervised learning**: a model learns to predict parts of its input from other parts of the same input

Input: *Ong choy is superb over rice*

Prediction task: Ong choy → is, superb, over, rice

$$\max_{\boldsymbol{\theta}} \prod_{(w,c)\in\mathcal{D}} p_{\boldsymbol{\theta}}(c|w)$$

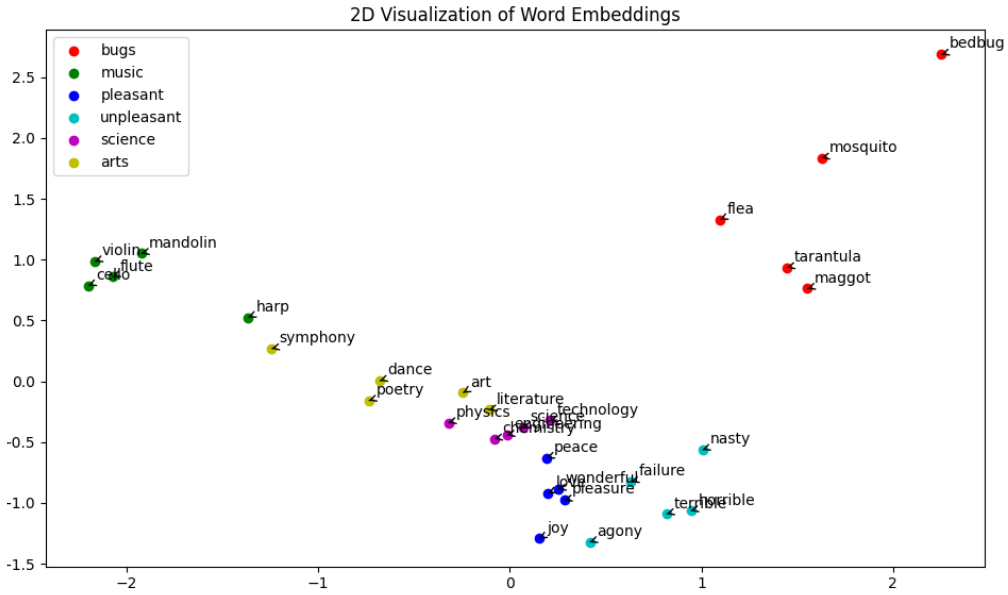
context word — center word

$$p_{\boldsymbol{\theta}}(c|w) = \frac{\exp(\boldsymbol{v}_c \cdot \boldsymbol{v}_w)}{\sum_{c'\in|\mathcal{V}|} \exp(\boldsymbol{v}_{c'} \cdot \boldsymbol{v}_w)}$$

Word Similarity

- Measure word similarity with cosine similarity between embeddings $\cos(\boldsymbol{v}_{w_1}, \boldsymbol{v}_{w_2})$
- Higher cosine similarity = more semantically close

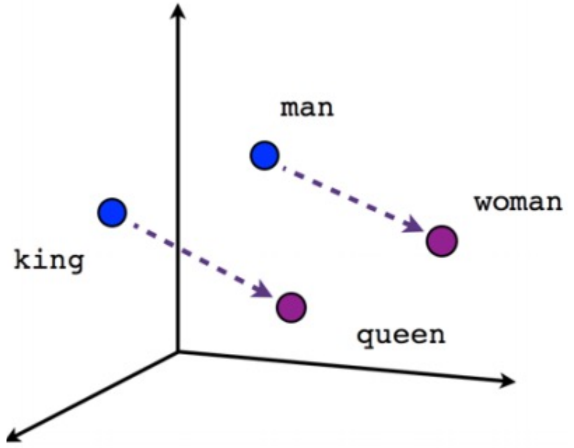
$$\cos(\boldsymbol{v}, \boldsymbol{w}) = \frac{\boldsymbol{v} \cdot \boldsymbol{w}}{|\boldsymbol{v}||\boldsymbol{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$



Word Analogy

- Word embeddings reflect intuitive semantic and syntactic analogy
- Example: man : woman :: king : ? $\boldsymbol{v}_{\text{queen}} \approx \boldsymbol{v}_{\text{woman}} - \boldsymbol{v}_{\text{man}} + \boldsymbol{v}_{\text{king}}$
- General case: find the word such that a : b :: c : ?
- Find the word that maximizes the cosine similarity

$$
\begin{aligned}
w &= \arg\max_{w' \in \mathcal{V}} \cos(\boldsymbol{v}_b - \boldsymbol{v}_a + \boldsymbol{v}_c, \boldsymbol{v}_{w'}) \\
&= \arg\max_{w' \in \mathcal{V}} \frac{(\boldsymbol{v}_b - \boldsymbol{v}_a + \boldsymbol{v}_c) \cdot \boldsymbol{v}_{w'}}{|\boldsymbol{v}_b - \boldsymbol{v}_a + \boldsymbol{v}_c||\boldsymbol{v}_{w'}|}
\end{aligned}
$$



Agenda

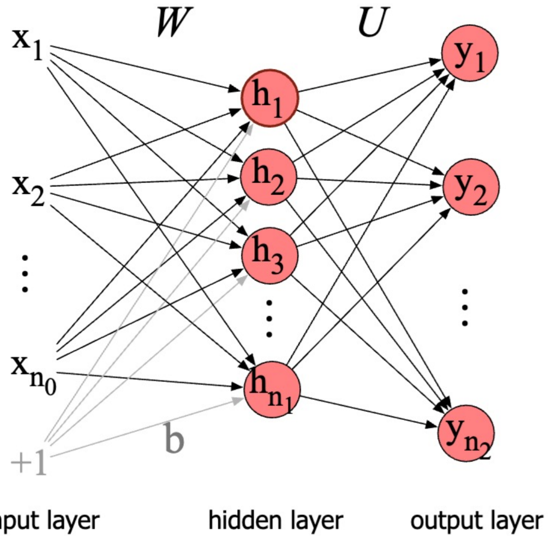
- Introduction to Language Models
- Word Embeddings
- Sequence Modeling and Recurrent Neural Networks (RNNs)
- Transformer
- Large language models (LLMs)

Sequence Modeling: Overview

- Use deep learning methods to understand, process, and generate **text sequences**
- Goals:
  - Learn context-dependent representations
  - Capture long-range dependencies
  - Handle complex relationships among large text units
- Sequence modeling architectures are based on deep neural networks (DNNs)!
  - Language exhibits hierarchical structures (e.g., letters form words, words form phrases, phrases form sentences)
  - DNNs learn multiple levels of abstraction across layers, allowing them to capture low-level patterns (e.g., word relations) in lower layers and high-level patterns (e.g., sentence meanings) in higher layers
  - Each layer in DNNs refines the word representations by considering contexts at different granularities (shorter & longer-range contexts), allowing for contextualized understanding of words and sequences

Simple Neural Language Model

Instantiate Feedforward network (FFN) as a neural language model



2-layer FFN

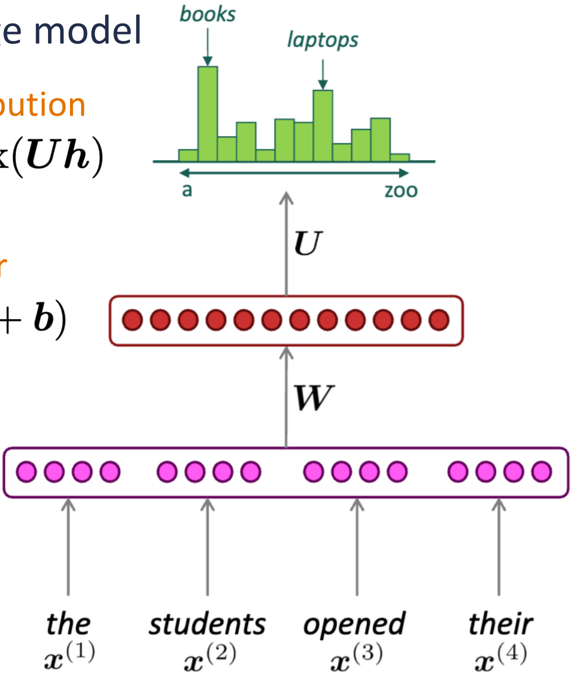
Output distribution

$$\boldsymbol{y} = \mathrm{softmax}(\boldsymbol{U}\boldsymbol{h})$$

Hidden layer

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

Word embeddings



2-layer neural language model

Figure source: https://web.stanford.edu/class/cs224n/slides/cs224n-spr2024-lecture05-rnnlm.pdf

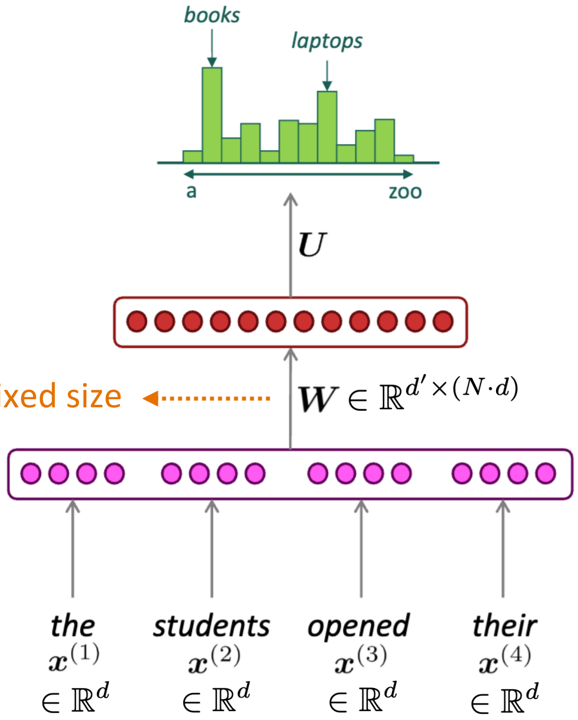
Limitations of (Simple) Neural Language Models

- Context window is fixed
- Increasing N will enlarge $\boldsymbol{W}$

Concatenated word embeddings

$\boldsymbol{x} = \boldsymbol{x}^{(1)} \oplus \boldsymbol{x}^{(2)} \oplus \cdots \oplus \boldsymbol{x}^{(N)} \in \mathbb{R}^{N \cdot d}$

Fixed size

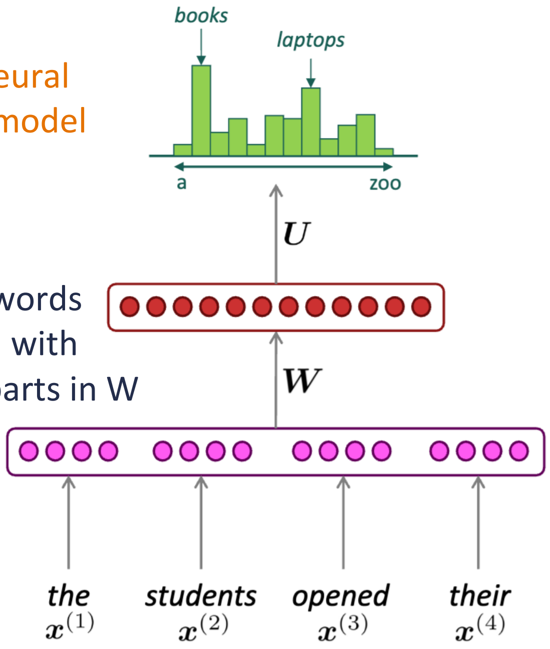


Recurrent Neural Network (RNN) Overview

A neural language model that can process inputs of arbitrary lengths

Simple neural language model

Different words multiplied with different subparts in W



Recurrent neural language model

Reuse the same weights for all words

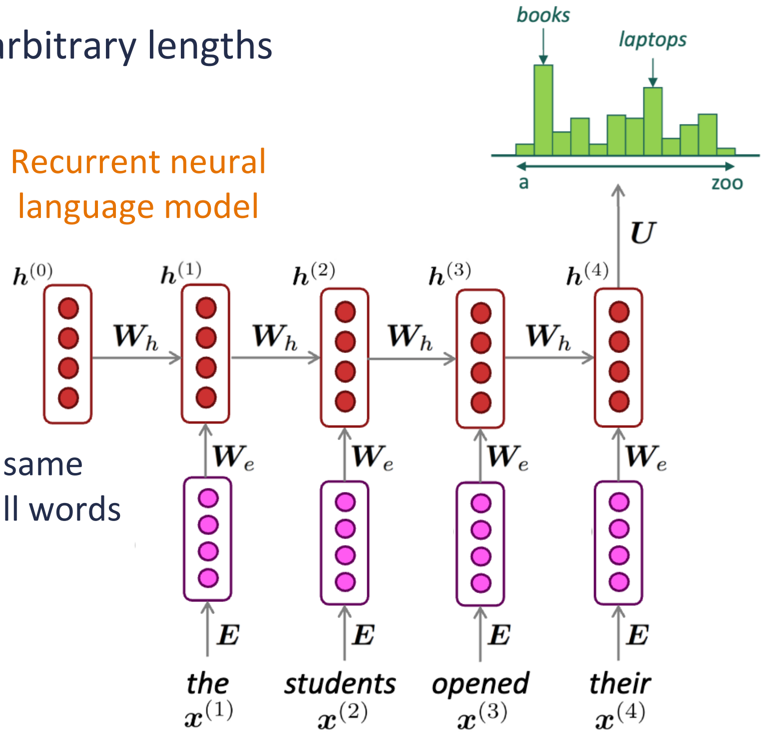


Figure source: https://web.stanford.edu/class/cs224n/slides/cs224n-spr2024-lecture05-rnnlm.pdf

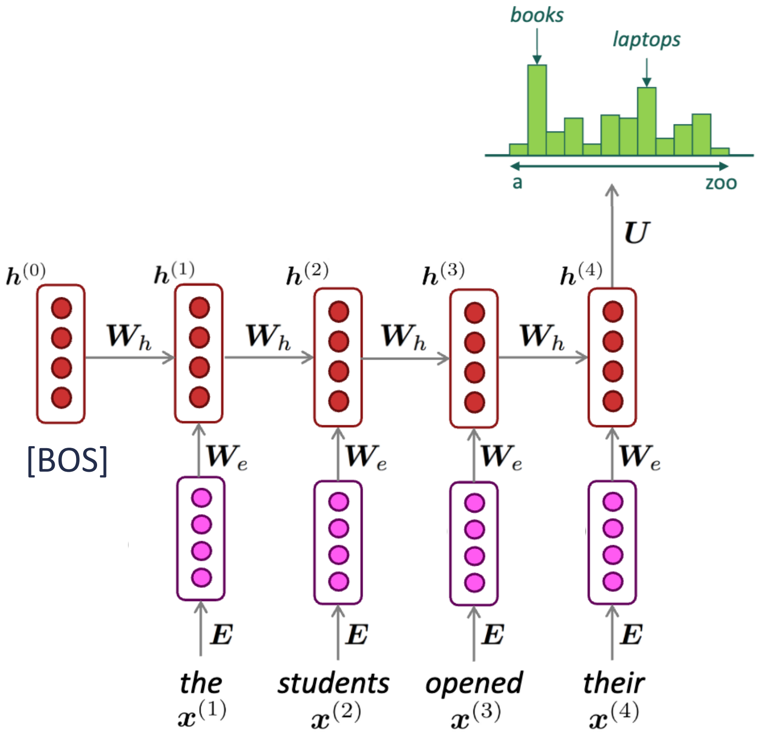
RNN Computation

- Hidden states in RNNs are computed based on
  - The hidden state at the previous step (memory)
  - The word embedding at the current step
- Parameters:
  - $\boldsymbol{W}_h$: weight matrix for the recurrent connection
  - $\boldsymbol{W}_e$: weight matrix for the input connection

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e \boldsymbol{x}^{(t)}\right)$$

Hidden states at the previous word (time step)

Word embedding of the current word (time step)



RNN for Language Modeling

- Recall that language modeling predict the next word given previous words

$$p(\boldsymbol{x}) = p\left(x^{(1)}\right) p\left(x^{(2)} | x^{(1)}\right) \cdots p\left(x^{(n)} | x^{(1)}, \ldots, x^{(n-1)}\right) = \prod_{t=1}^{n} p\left(x^{(t)} | x^{(1)}, \ldots, x^{(t-1)}\right)$$

- How to use RNNs to represent $p\left(x^{(t)} | x^{(1)}, \ldots, x^{(t-1)}\right)$?

Output probability at (t-1) step: $\boldsymbol{y}^{(t-1)} = \mathrm{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t-1)}\right) := f\left(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t-2)}, \boldsymbol{x}^{(t-1)}\right)$

$\boldsymbol{h}^{(t-1)}$ is a function of $\boldsymbol{h}^{(t-2)}$ and $\boldsymbol{x}^{(t-1)}$: $\boldsymbol{h}^{(t-1)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-2)} + \boldsymbol{W}_e \boldsymbol{x}^{(t-1)}\right) := g\left(\boldsymbol{h}^{(t-2)}, \boldsymbol{x}^{(t-1)}\right)$

$\boldsymbol{h}^{(t-2)}$ is a function of $\boldsymbol{h}^{(t-3)}$ and $\boldsymbol{x}^{(t-2)}$: $\boldsymbol{h}^{(t-2)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-3)} + \boldsymbol{W}_e \boldsymbol{x}^{(t-2)}\right) := g\left(\boldsymbol{h}^{(t-3)}, \boldsymbol{x}^{(t-2)}\right)$

$\vdots \qquad\qquad \vdots$

$\boldsymbol{h}^{(1)}$ is a function of $\boldsymbol{h}^{(0)}$ and $\boldsymbol{x}^{(1)}$: $\boldsymbol{h}^{(1)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(0)} + \boldsymbol{W}_e \boldsymbol{x}^{(1)}\right) := g\left(\boldsymbol{h}^{(0)}, \boldsymbol{x}^{(1)}\right)$

Agenda

- Introduction to Language Models
- Word Embeddings
- Sequence Modeling and Recurrent Neural Networks (RNNs)
- Transformer
- Large language models (LLMs)

Transformer: Overview

- Transformer is a specific kind of sequence modeling architecture (based on DNNs)
- Use attention to replace recurrent operations in RNNs
- The most important architecture for language modeling (almost all LLMs are based on Transformers)!

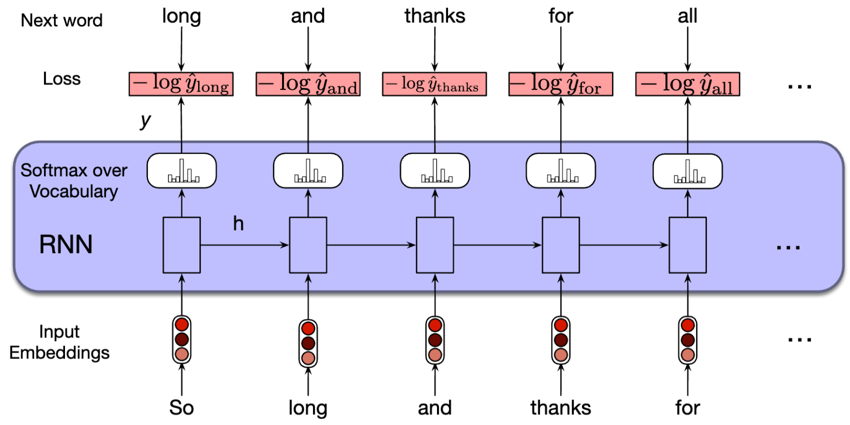
Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Transformer: https://arxiv.org/pdf/1706.03762

Transformer vs. RNN

RNN
(recurrent computations)



Transformer
(self-attention computations)

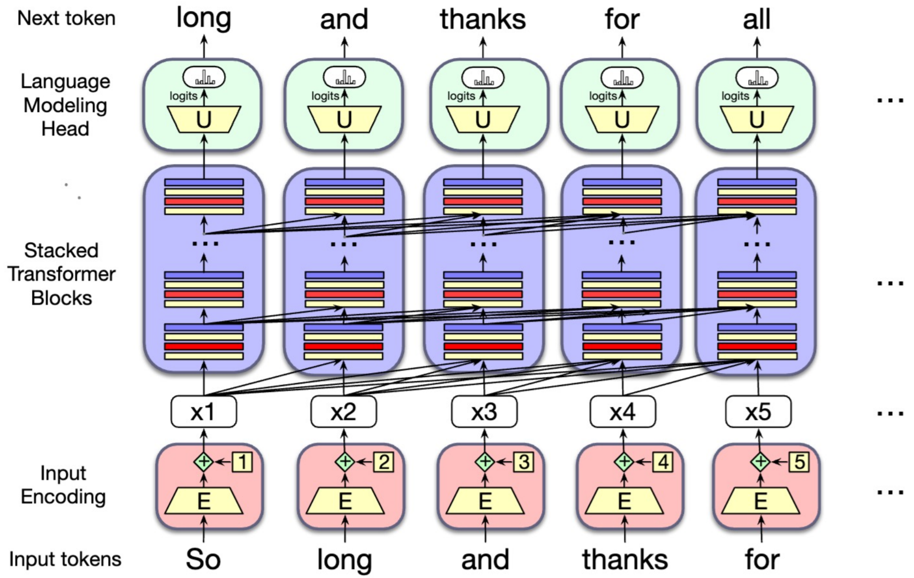


Figure source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

Transformer: Motivation

- Parallel token processing
  - RNN: process one token at a time (computation for each token depends on previous ones)
  - Transformer: process all tokens in a sequence in parallel
- Long-term dependencies
  - RNN: bad at capturing distant relating tokens (vanishing gradients)
  - Transformer: directly access any token in the sequence, regardless of its position
- Bidirectionality
  - RNN: can only model sequences in one direction
  - Transformer: inherently allow bidirectional sequence modeling via attention

Transformer Layer

Each Transformer layer contains the following important components:

- Self-attention
- Feedforward network
- Residual connections + layer norm

Transformer layer

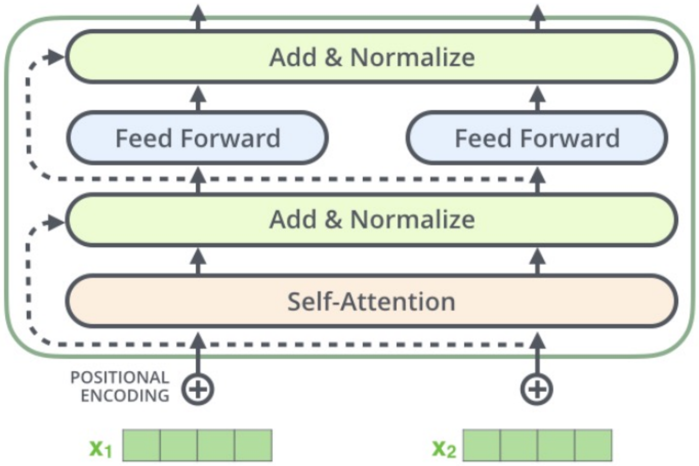


Figure source: https://jalammar.github.io/illustrated-transformer/

Self-Attention: Intuition

- Attention: weigh the importance of different words in a sequence when processing a specific word
  - "When I'm looking at this word, which other words should I pay attention to in order to understand it better?"
- **Self-attention**: each word attends to other words in the **same** sequence
- Example: "The chicken didn't cross the road because it was too tired"
  - Suppose we are learning attention for the word "**it**"
  - With self-attention, "**it**" can decide which other words in the sentence it should focus on to better understand its meaning
  - Might assign high attention to "**chicken**" (the subject) & "**road**" (another noun)
  - Might assign less attention to words like "the" or "didn't"

UNIVERSITY of VIRGINIA

Self-Attention: Example

Derive the center word representation as a weighted sum of context representations!

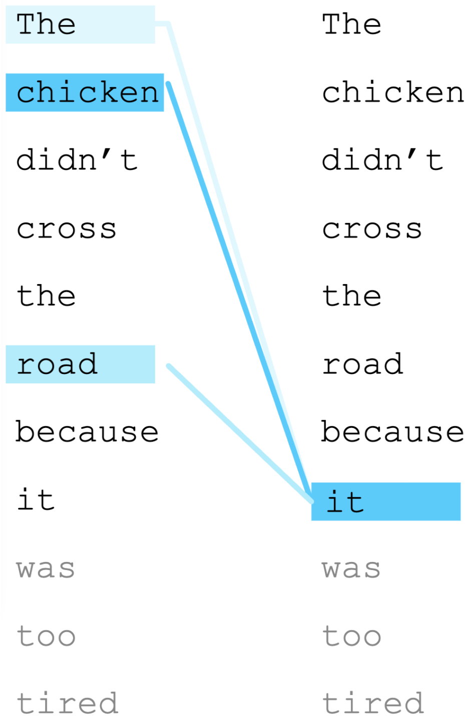
Center word representation

Context word representation

$$\boldsymbol{a}_i = \sum_{x_j \in \boldsymbol{x}} \alpha_{ij} \boldsymbol{x}_j, \quad \sum_{x_j \in \boldsymbol{x}} \alpha_{ij} = 1$$

Attention score $i \rightarrow j$, summed to 1

Context word (key) Center word (query)



Current word = "it"

Figure source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

Self-Attention: Attention Score Computation

- Attention score is given by the softmax function over vector dot product

$$\boldsymbol{a}_i = \sum_{x_j \in \boldsymbol{x}} \alpha_{ij} \boldsymbol{x}_j, \quad \sum_{x_j \in \boldsymbol{x}} \alpha_{ij} = 1$$

$$\alpha_{ij} = \text{Softmax}(\boldsymbol{x}_i \cdot \boldsymbol{x}_j)$$

Center word (query) representation

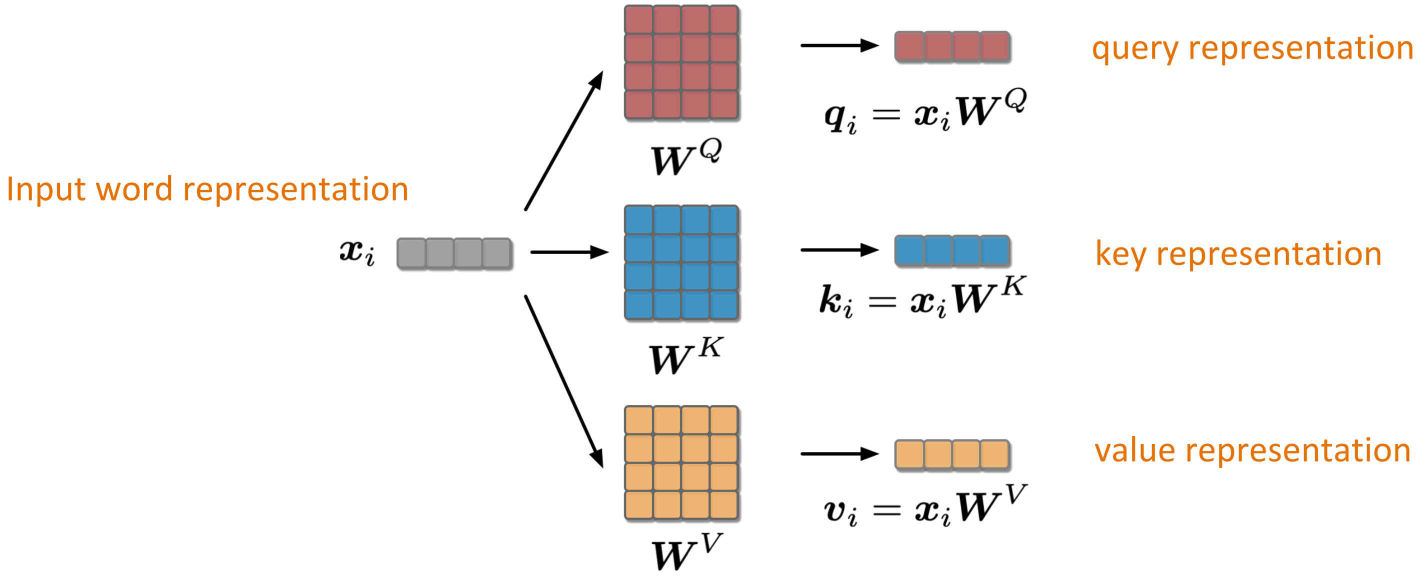
Context word (key) representation

Self-Attention: Query, Key, and Value

- Each word in self-attention is represented by three different vectors
  - Allow the model to flexibly capture different types of relationships between tokens
- **Query (Q)**:
  - Represent the current word seeking information about
- **Key (K)**:
  - Represent the reference (context) against which the query is compared
- **Value (V)**:
  - Represent the actual content associated with each token to be aggregated as final output

Self-Attention: Query, Key, and Value

Each self-attention module has three weight matrices applied to the input word vector to obtain the three copies of representations



Self-Attention: Overall Computation

- Input: single word vector of each word $\boldsymbol{x}_i$
- Compute Q, K, V representations for each word:

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

- Compute attention scores with Q and K
  - The dot product of two vectors usually has an expected magnitude proportional to $\sqrt{d}$
  - Divide the attention score by $\sqrt{d}$ to avoid extremely large values in softmax function

$$\alpha_{ij} = \mathrm{Softmax}\left(\frac{\boldsymbol{q}_i \cdot \boldsymbol{k}_j}{\sqrt{d}}\right)$$

Dimensionality of q and k

- Sum the value vectors weighted by attention scores

$$\boldsymbol{a}_i = \sum_{x_j \in \boldsymbol{x}} \alpha_{ij} \boldsymbol{v}_j$$

Self-Attention: Illustration

- Example: an input sequence with three words $[x_1, x_2, x_3]$
- Suppose we want to compute the self-attention for x_3

Output of self-attention

Sum the weighted value vectors

Obtain attention scores via softmax

Divide the dot product by vector dimension

Compare x_3's query with the keys of all words

Compute query, key, value

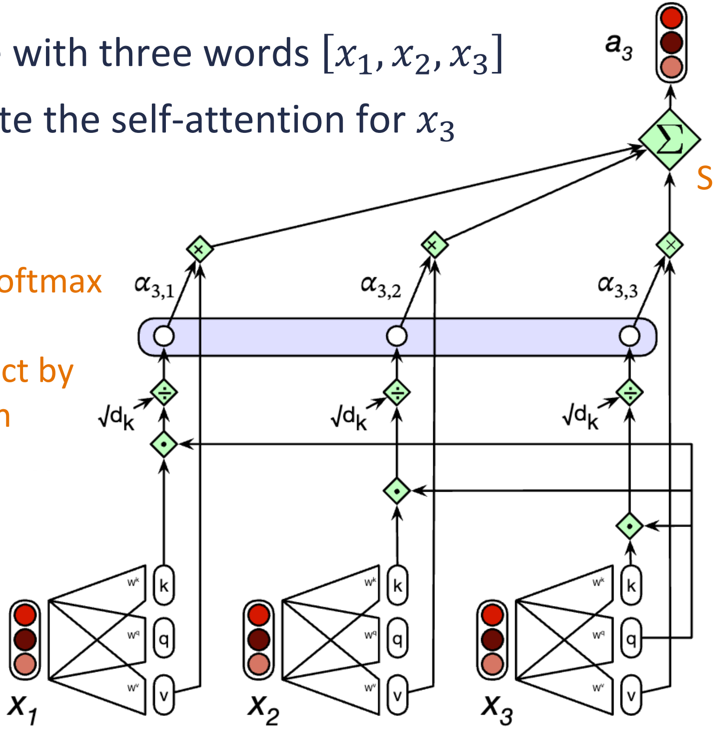
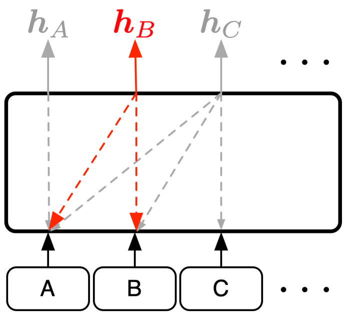


Figure source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

Unidirectional Self-Attention

- Self-attention can capture context dependencies
- **Unidirectional** (or **causal**) self-attention:
  - Each position can only attend to earlier positions in the sequence (including itself).
  - Transformers with unidirectional self-attention are called Transformer **decoders** (e.g., GPT)
  - Use case: natural language generation (NLG) where the model generates output sequentially



every token attends to its previous tokens

Unidirectional Self-Attention

$$\alpha_{ij} = \text{Softmax}\left(\frac{\boldsymbol{q}_i \cdot \boldsymbol{k}_j}{\sqrt{d}}\right)$$

upper-triangle portion set to -inf

q1•k1	$-\infty$	$-\infty$	$-\infty$
q2•k1	q2•k2	$-\infty$	$-\infty$
q3•k1	q3•k2	q3•k3	$-\infty$
q4•k1	q4•k2	q4•k3	q4•k4

UNIVERSITY of VIRGINIA

Agenda

- Introduction to Language Models
- Word Embeddings
- Sequence Modeling and Recurrent Neural Networks (RNNs)
- Transformer
- Large language models (LLMs)

Pretraining: Motivation

- Before pretraining became prevalent in NLP, most NLP models were trained from scratch on downstream task data
- **Data scarcity**: many NLP tasks do not have large labeled datasets available (costly to obtain)
- **Poor generalization**: models trained from scratch on specific tasks do not generalize well to unseen data or other tasks
- **Sensitivity to noise and randomness**: models are more likely to learn spurious correlations or be affected by annotation errors/randomness in training

Pretraining: Motivation

- There are abundant text data on the web, with rich information of linguistic features and knowledge about the world
- Learning from these easy-to-obtain data greatly benefits various downstream tasks

Pretraining: Multi-Task Learning

- In my free time, I like to **{run, banana}** (*Grammar*)
- I went to the zoo to see giraffes, lions, and **{zebras, spoon}** (*Lexical semantics*)
- The capital of Denmark is **{Copenhagen, London}** (*World knowledge*)
- I was engaged and on the edge of my seat the whole time. The movie was **{good, bad}** (*Sentiment analysis*)
- The word for “pretty” in Spanish is **{bonita, hola}** (*Translation*)
- 3 + 8 + 4 = **{15, 11}** (*Math*)
- …

Examples from: https://docs.google.com/presentation/d/1hQUd3pF8_2Gr2Obc89LKjmHL0DlH-uof9M0yFVd3FA4/edit#slide=id.g28e2e9aa709_0_1

UNIVERSITY of VIRGINIA

Transformer for Pretraining

- Transformer is the common backbone architecture for language model pretraining
- **Efficiency**: Transformer processes all tokens in a sequence simultaneously – fast and efficient to train, especially on large datasets
- **Scalability**: Transformer architectures have shown impressive scaling properties, with performance improving as model size and training data increase (more on this later!)
- **Versatility**: Transformer can be adapted for various tasks and modalities beyond just text, including vision, audio, and other multimodal applications

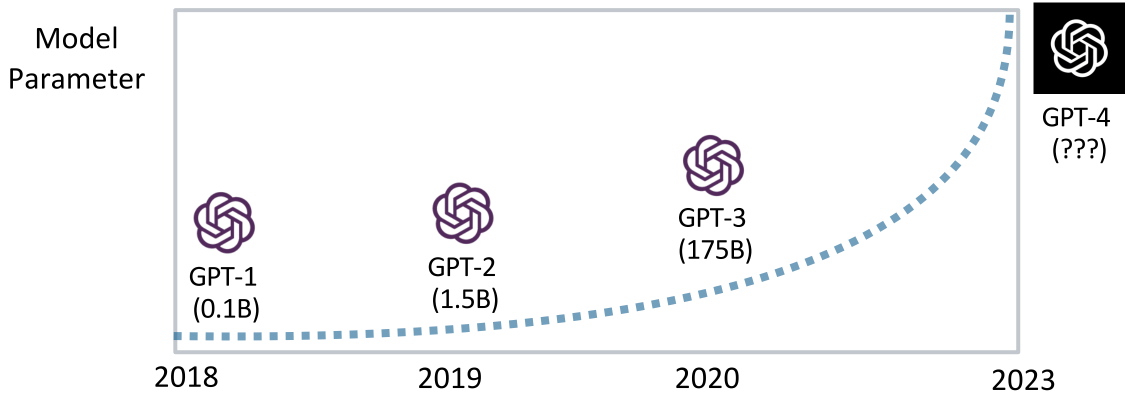
Transformer Decoder Pretraining

- Decoder architecture is the prominent choice in large language models
- Pretraining decoders is first introduced in GPT (generative pretraining) models
- Follow the standard language modeling (cross-entropy) objective

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N}\log p_{\boldsymbol{\theta}}(x_i|x_1, x_2, \ldots, x_{i-1})$$

GPT Series

- GPT-1 (2018): 12 layers, 117M parameters, trained in ~1 week
- GPT-2 (2019): 48 layers, 1.5B parameters, trained in ~1 month
- GPT-3 (2020): 96 layers, 175B parameters, trained in several months



Papers: (GPT-1) https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf
(GPT-2) https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf
(GPT-3) https://arxiv.org/pdf/2005.14165.pdf

Why Scaling Up Language Models? Emergent Ability

- Larger models develop **emergent abilities**
  - Skills or capabilities that were not explicitly learned but arise as a result of model capacity
  - Larger models demonstrate surprising abilities in challenging tasks even when they were not explicitly trained for them
- Emergent capabilities typically become noticeable only when the model size reaches a certain threshold (cannot be predicted by small model's performance)

Performance vs. Model Scale

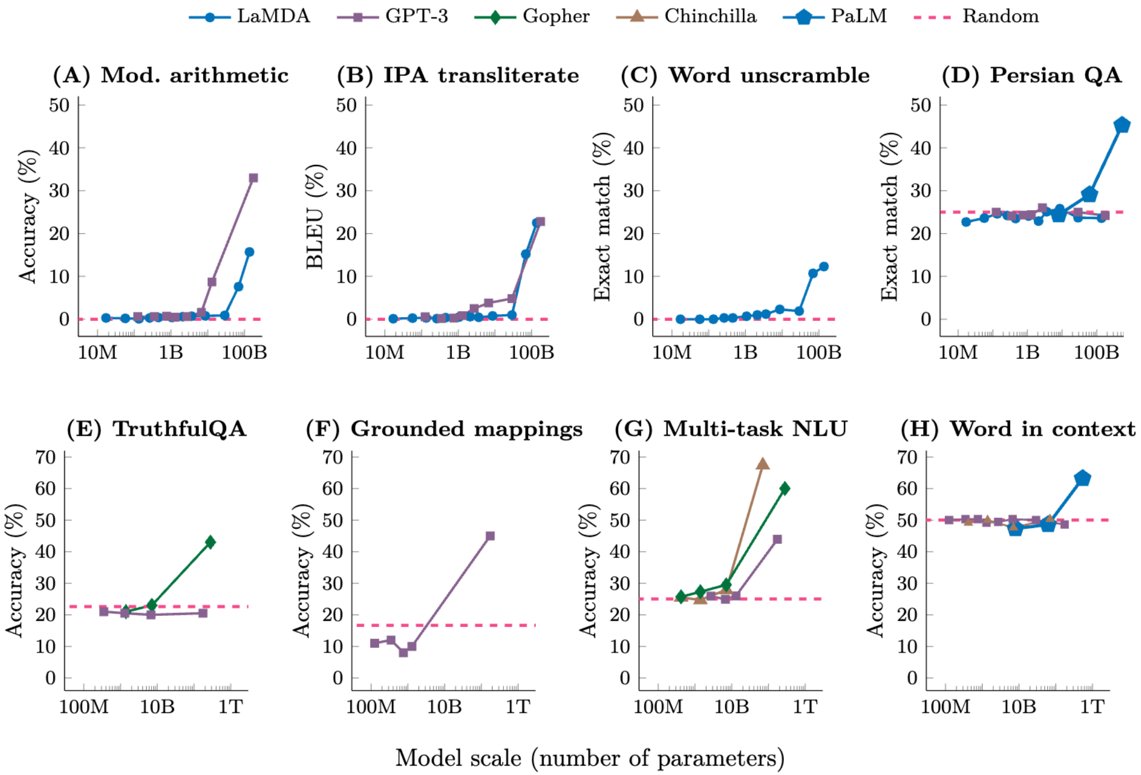
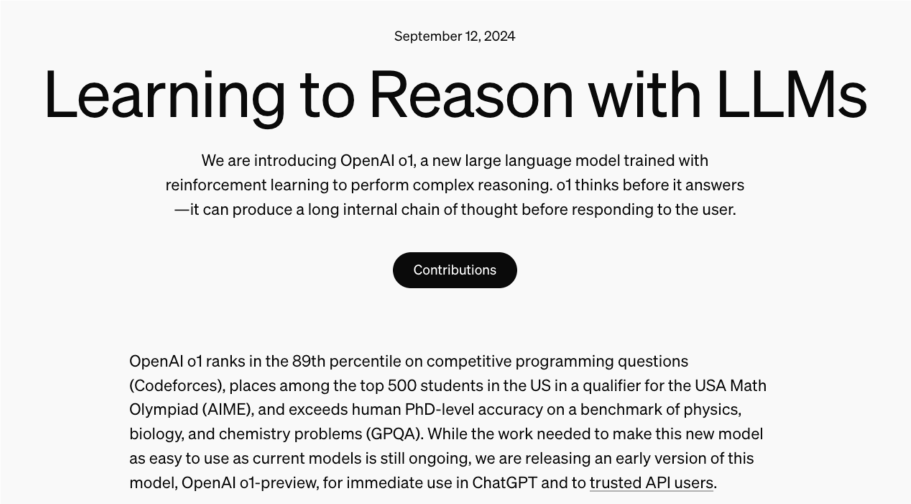


Figure source: https://arxiv.org/pdf/2206.07682

Models exhibit random performance until a certain scale, after which performance significantly increases

Lots of Ongoing Developments!


September 12, 2024

Learning to Reason with LLMs

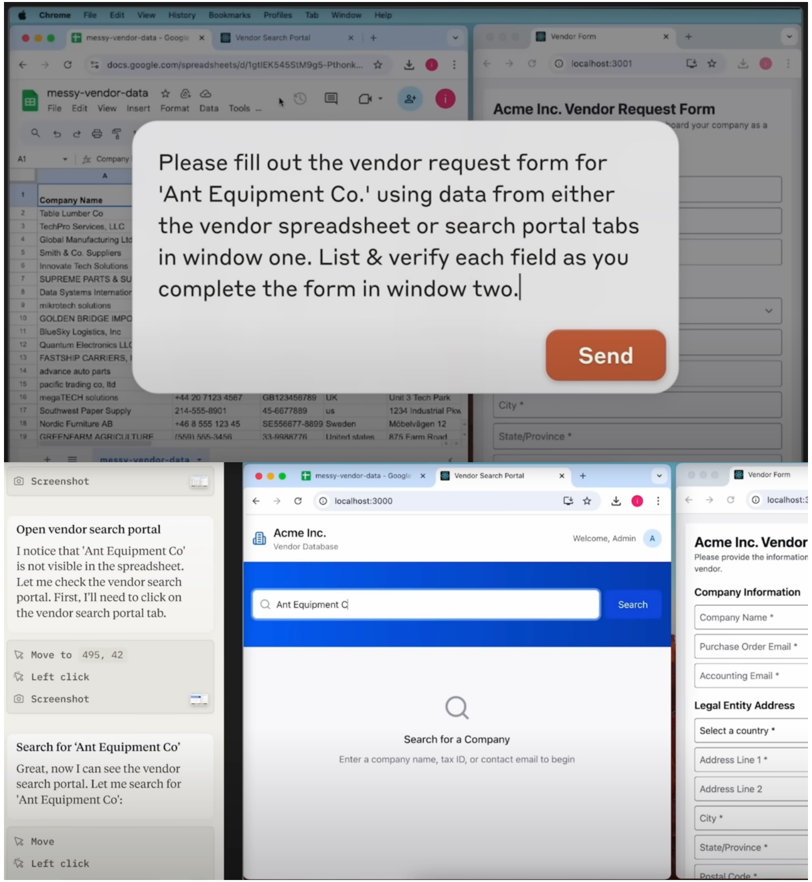
We are introducing OpenAI o1, a new large language model trained with reinforcement learning to perform complex reasoning. o1 thinks before it answers —it can produce a long internal chain of thought before responding to the user.

Contributions

OpenAI o1 ranks in the 89th percentile on competitive programming questions (Codeforces), places among the top 500 students in the US in a qualifier for the USA Math Olympiad (AIME), and exceeds human PhD-level accuracy on a benchmark of physics, biology, and chemistry problems (GPQA). While the work needed to make this new model as easy to use as current models is still ongoing, we are releasing an early version of this model, OpenAI o1-preview, for immediate use in ChatGPT and to trusted API users.

LLM agents for complex reasoning

Figure source: https://openai.com/index/learning-to-reason-with-llms/



LLM agents for computer use

Figure source: https://www.anthropic.com/news/3-5-models-and-computer-use

NLP Courses at UVA

- CS 4501 NLP: Undergraduate NLP course (introductory)
- CS 6501 NLP: Graduate NLP course (advanced)

Thank You!

Yu Meng
University of Virginia
yumeng5@virginia.edu