DATASCI 347 Machine Learning

Lecture 21: Introduction to Language Models

Ruoxuan Xiong

EMORY

Language modeling

- Language Modeling is the task of predicting what word comes next

- Example: the students opened their _____
  A. books
  B. laptops
  C. exams
  D. minds

Language modeling

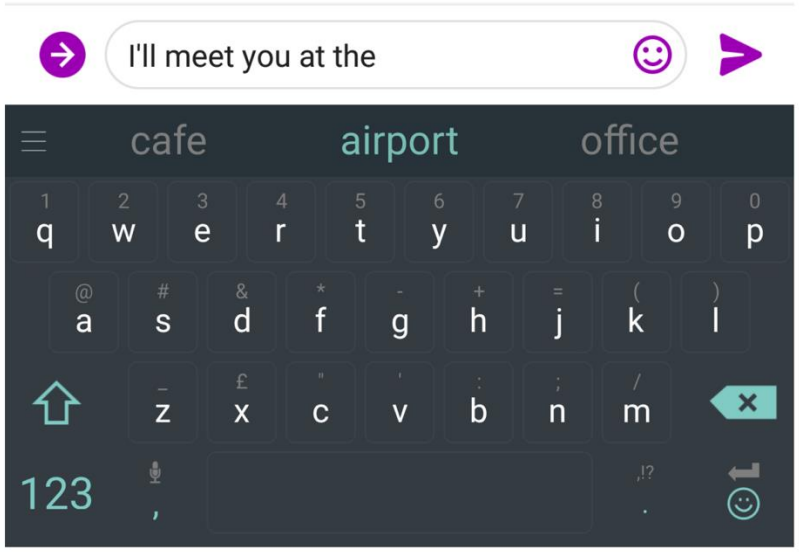
- More formally, given a sequence of words $x^{(1)}, x^{(2)}, \cdots, x^{(t)}$ compute the probability distribution of the next word $x^{(t+1)}$:

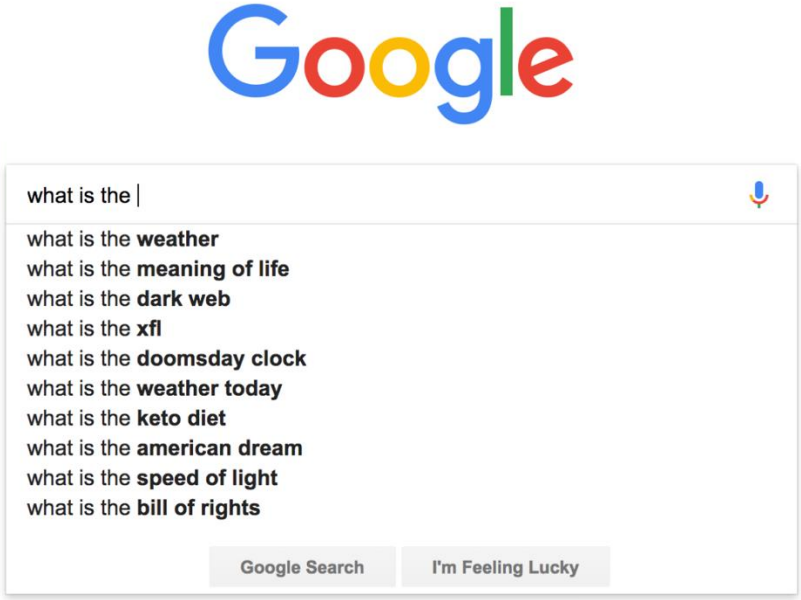
$$P\left(x^{(t+1)} \mid x^{(t)}, \cdots, x^{(1)}\right)$$

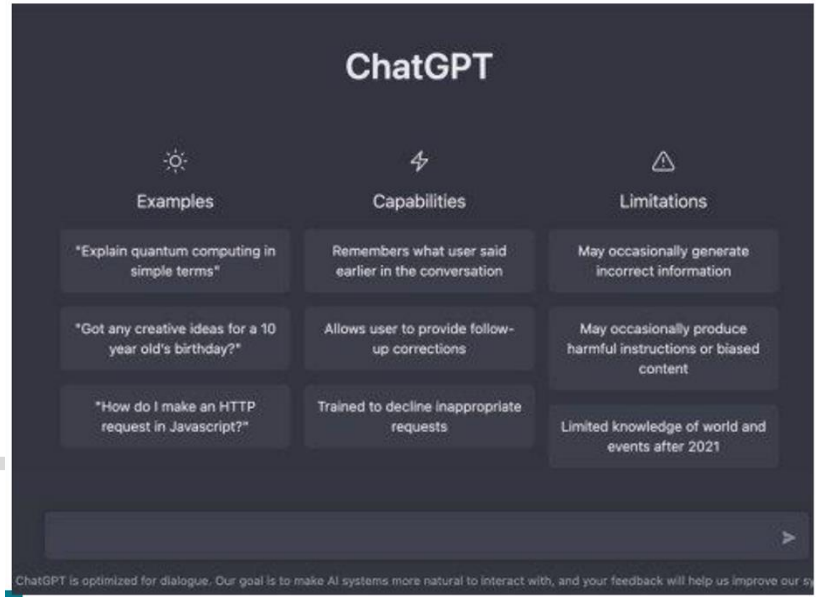
where $x^{(t+1)}$ can be any word in the vocabulary $V = \{w_1, \cdots, w_{|V|}\}$

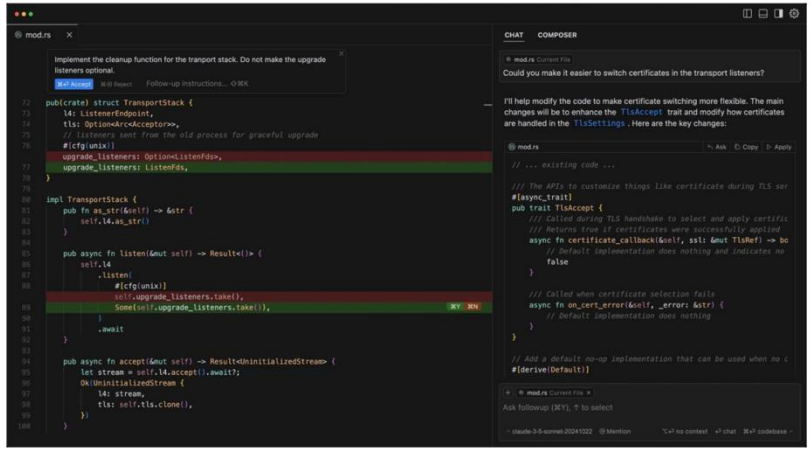
- A system that does this is called a language model
  - Assign a probability to a piece of text

We use language models everyday!



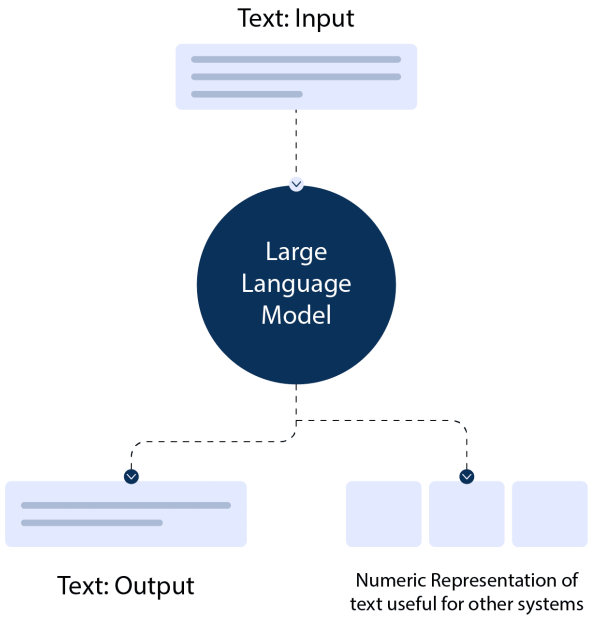


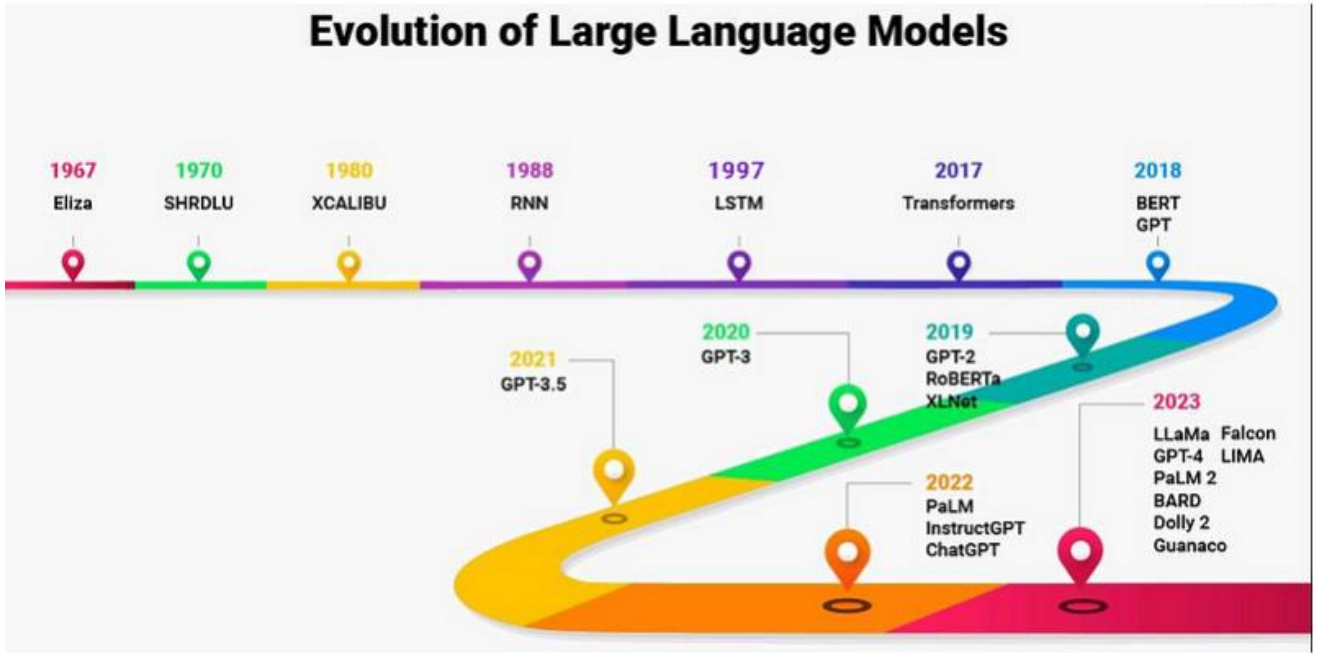




Language models

- A language model enables computers to understand, generate, and interact with human language effectively
- It evolves from simple probabilistic approaches to advanced transformer-based models like GPT and BERT





History of language models

- **Stage 1**: Early foundations, rule-based systems (1950s-1970s)

- Relied on manually crafted rules for syntax and grammar, lacking statistical or probabilistic components

- Examples:
  - Eliza (1966): A rule-based chatbot that simulates a psychotherapist using simple pattern matching
  - SHRDLU (1970s): A system that used logic-based rules to understand and manipulate blocks in a simulated environment

History of language models

- **Stage 2**: Statistical language models (1980s-1990s)

- Probabilistic models to estimate the probability of word sequences based on observed frequencies in text corpora

- **Key techniques**: n-grams, hidden Markov models (HMM)

N-gram language models

- Example: the students opened their _____

- An n-gram is a chunk of n consecutive words
  - Unigram ($n = 1$): "the", "students", "opened", "their"
  - Bigram ($n = 2$): "the students", "students opened", "opened their"
  - Trigram ($n = 3$): "the students opened", "students opened their"
  - Four-gram ($n = 3$): "the students opened their"

- Key idea: Collect statistics about how frequent different n-grams are and use these to predict next word

N-gram language models

- First we assume $x^{(t+1)}$ only depends on the preceding $n-1$ words

$$P\left(x^{(t+1)} \mid x^{(t)}, \cdots, x^{(1)}\right) = P\left(x^{(t+1)} \mid x^{(t)}, \cdots, x^{(t-n+2)}\right)$$

- Question: How do we get these probabilities?

- Answer: We count them in some large corpus of text!

N-gram language models

- Example: Suppose we are learning a 4-gram language model
- Count on "students opened their"
- Suppose in the corpus:
  - "students opened their" occurred 1000 times
  - "students opened their books" occurred 400 times
    - $P(\text{books} \mid \text{students opened their}) = 0.4$
  - "students opened their exams" occurred 100 times
    - $P(\text{exams} \mid \text{students opened their}) = 0.1$

Generating text with an n-gram language model

- We can use a language model to generate text
- Example of bigram model: today the _____
- Condition on "today the", get the probability distribution
  - company: 0.153
  - bank: 0.153
  - price: 0.055 sample
  - italian: 0.039
  - emirate: 0.039

Generating text with an n-gram language model

- We can use a language model to generate text
- Example of bigram model: today the price _____
- Condition on "the price", get the probability distribution
  - of: 0.308 sample
  - for: 0.050
  - it: 0.046
  - to: 0.046
  - is: 0.031

Generating text with an n-gram language model

- We can use a language model to generate text
- Example of bigram model: today the price of _____
- Condition on "price of", get the probability distribution
  - the: 0.072
  - 18: 0.043
  - oil: 0.043
  - its: 0.036
  - gold: 0.018 sample

Generating text with an n-gram language model

- We can use a language model to generate text

- *today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .*

- **Limitation:** It's grammatical, but incoherent! We need to consider more than three words at a time!

History of language models

- **Stage 3**: Neural language models (NLMs) (2000s)

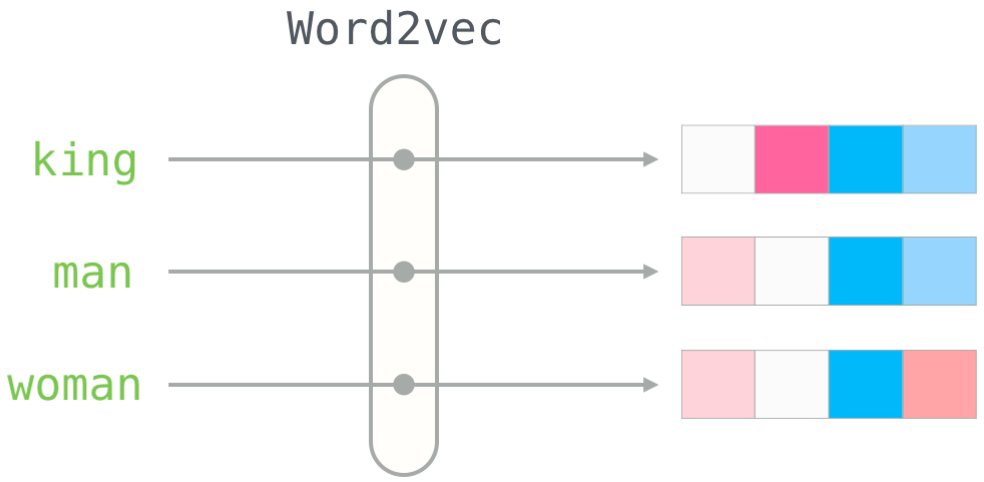
- Use neural networks to learn distributed representations of words, known as word embeddings

- NLMs could generalize better by capturing semantic and syntactic relationships

- **Key innovations**: Word2Vec (2013), Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM, 1997)

Word2Vec

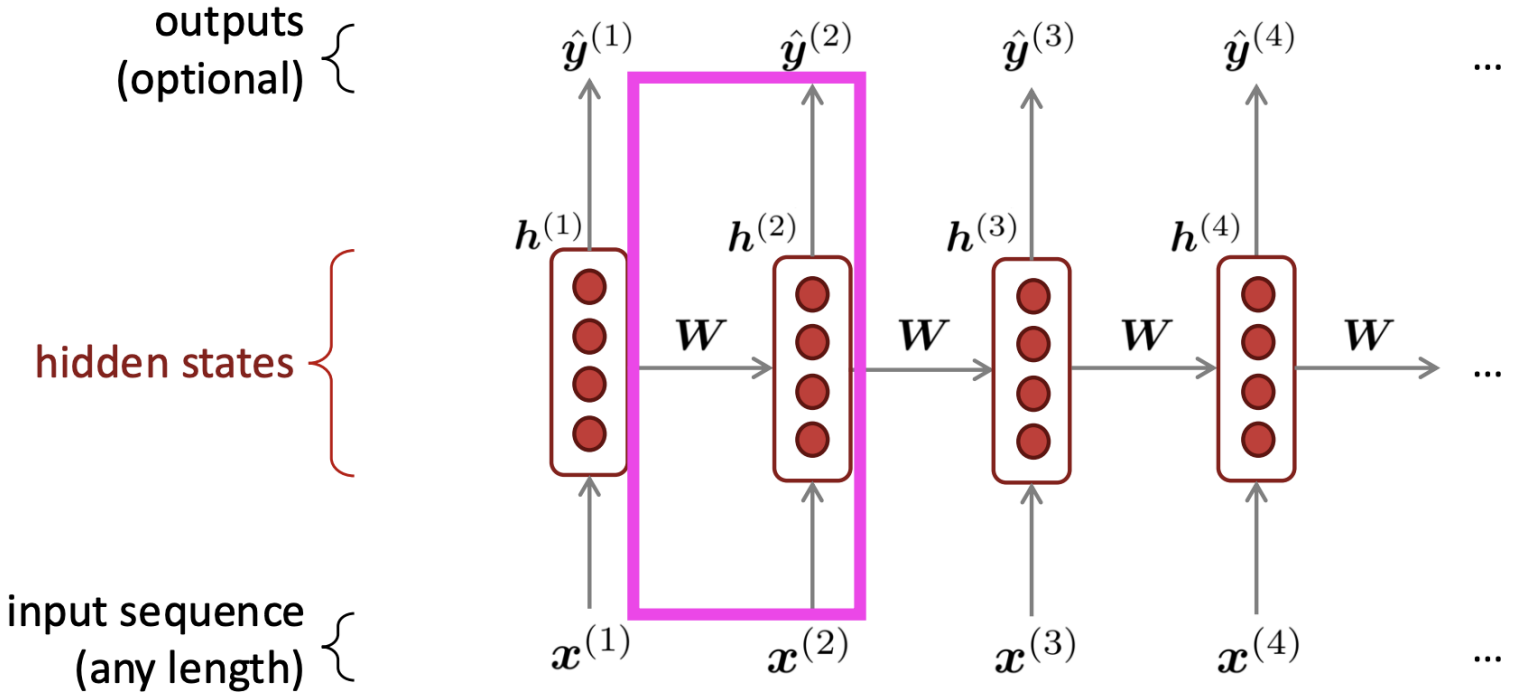
- A neural network-based model developed by Google in 2013 to generate word embeddings
- Word2Vec maps each word to a fixed-size vector in a high-dimensional space. Words with similar meanings or contexts are placed close together in this space
- Example: "woman" and "man" will be closer than "king" and "woman"



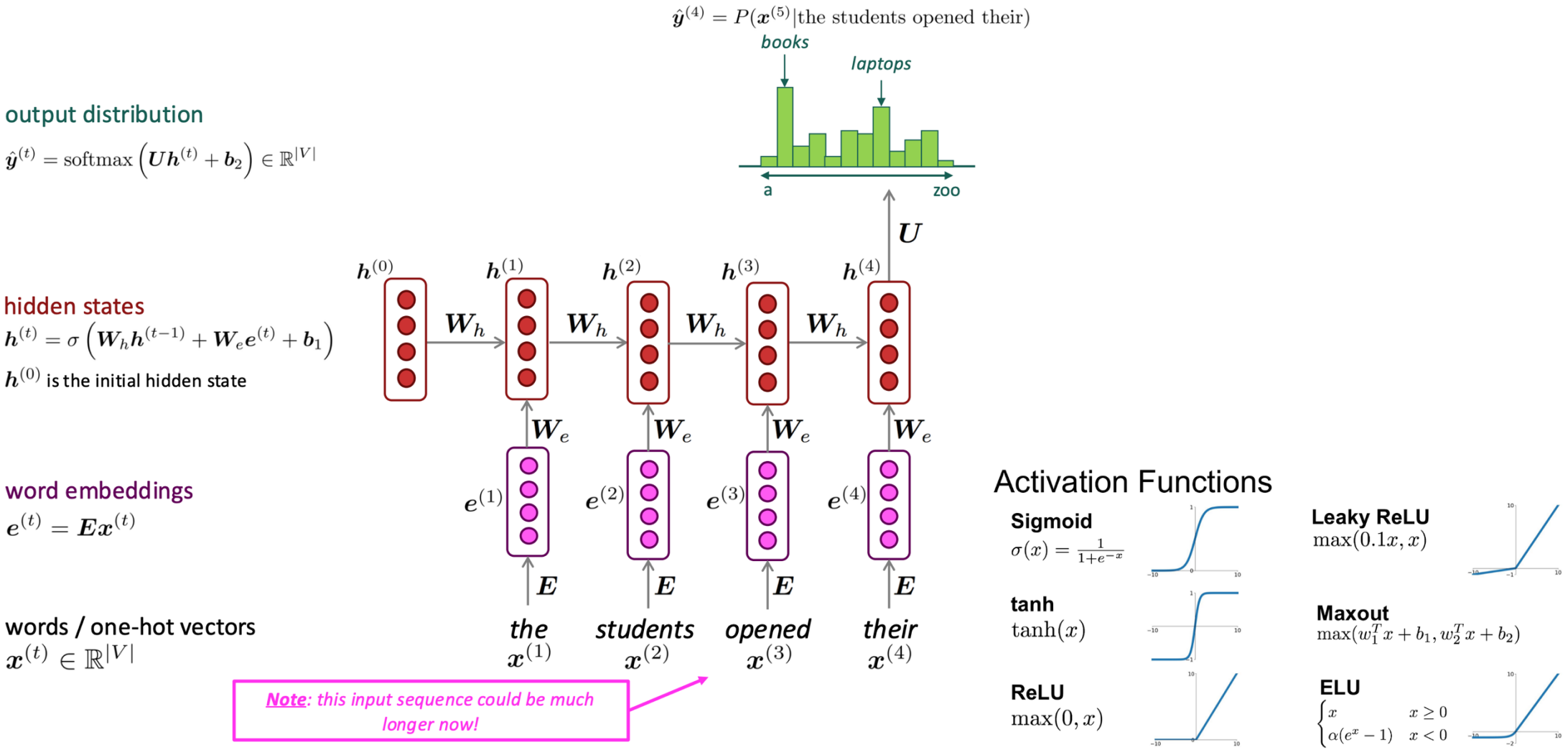
Mikolov, Tomas. "Efficient estimation of word representations in vector space." *arXiv preprint arXiv:1301.3781* 3781 (2013).

Recurrent Neural Network (RNN)

- A type of artificial neural network designed to handle sequential data by maintaining a memory of past inputs
- Core idea: Apply the same weights repeatedly



Recurrent Neural Network



Activation Functions

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

tanh
$\tanh(x)$

ReLU
$\max(0, x)$

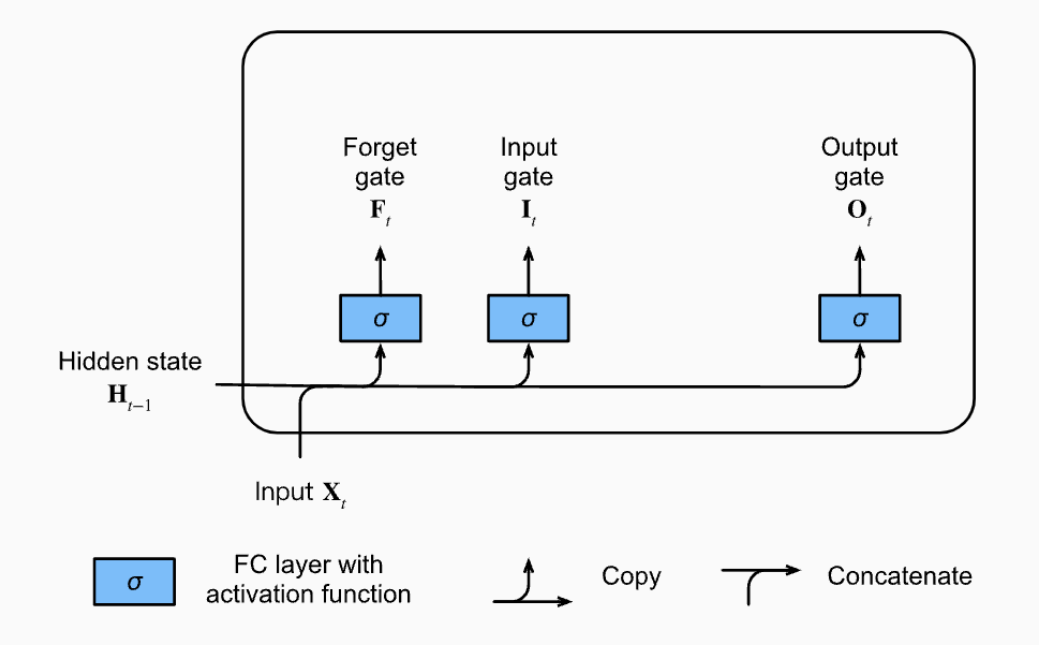
Leaky ReLU
$\max(0.1x, x)$

Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$

Long Short-Term Memory (LSTM)

- LSTM is a type of RNN designed to maintain information over long sequences
- LSTM has three gates to control the flow of information
  - Forget gate: Determines what information to discard
  - Input gate: Determines what new information to store in memory
  - Output gate: Determines what information to output
- LSTM also has hidden states



History of language models

- **Stage 4**: Transformer revolution (2017-present)
- Transformer architecture in the paper "Attention is All You Need" revolutionized language modeling
- Transformers use **self-attention mechanisms** to model long-range dependencies efficiently
- **Key Milestones**: BERT (2018), GPT (2018-now)

Transformer

- **Key feature**: **Self-Attention Mechanism**

- It allows the model to focus on particular parts of the input sequence, regardless of their distance from the current token

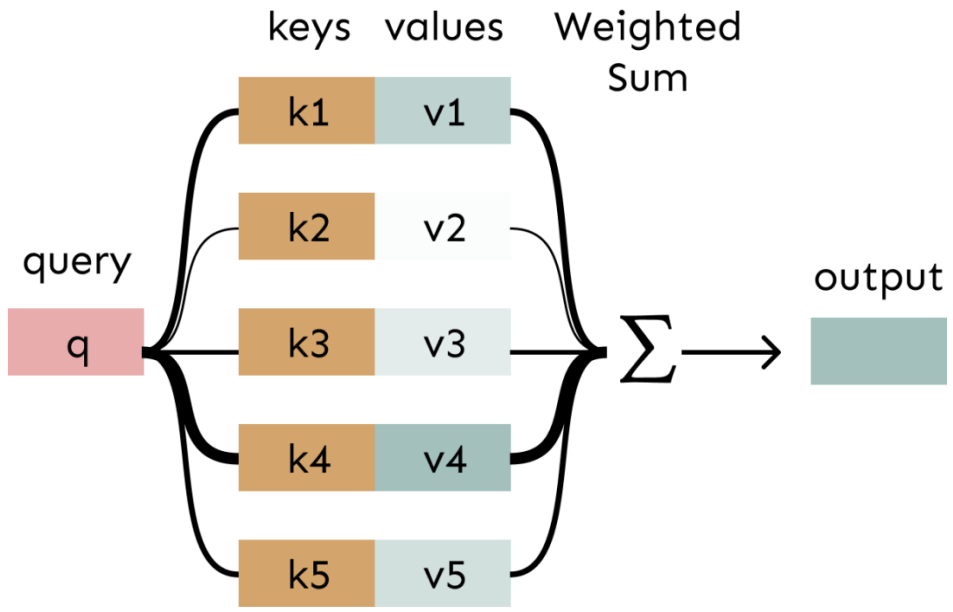
- Example #1: "*The cat chased the mouse, and it ran away*."
  - "It" can refer to "mouse" or "cat"
  - Self-attention helps the model focus on "mouse" based on semantic relationship

- Example #2: "*The dog chased the ball, and it was happy*."
  - "It" can refer to "dog" or "ball"
  - Self-attention helps the model focus on "dog" based on semantic relationship

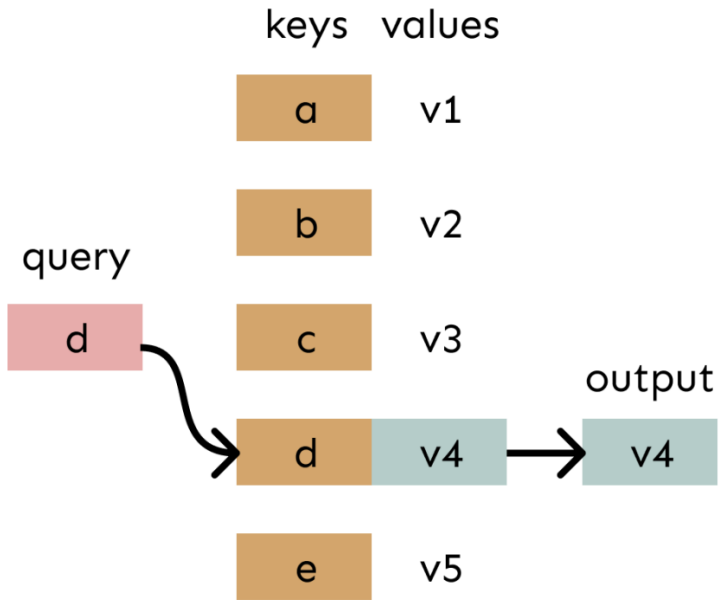
Attention behaves like "soft" lookups

Attention is just a **weighted** average – this is very powerful if the weights are learned!

In **attention**, the **query** matches all **keys** *softly*, to a weight between 0 and 1. The keys' **values** are multiplied by the weights and summed.

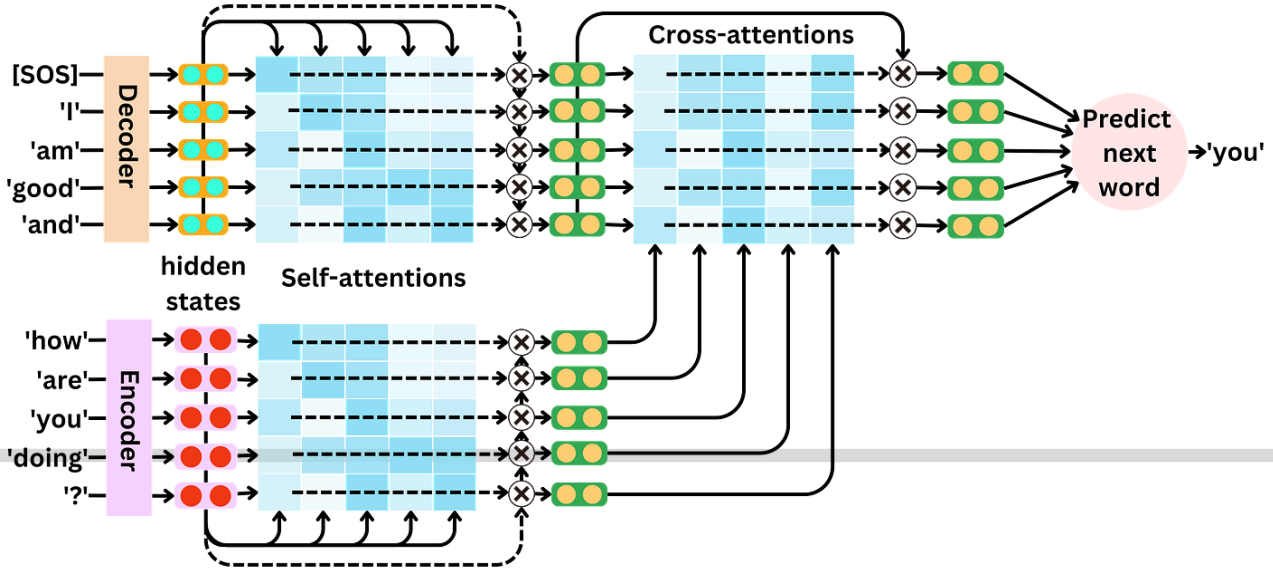


In a **lookup table**, we have a table of **keys** that map to **values**. The **query** matches one of the keys, returning its value.



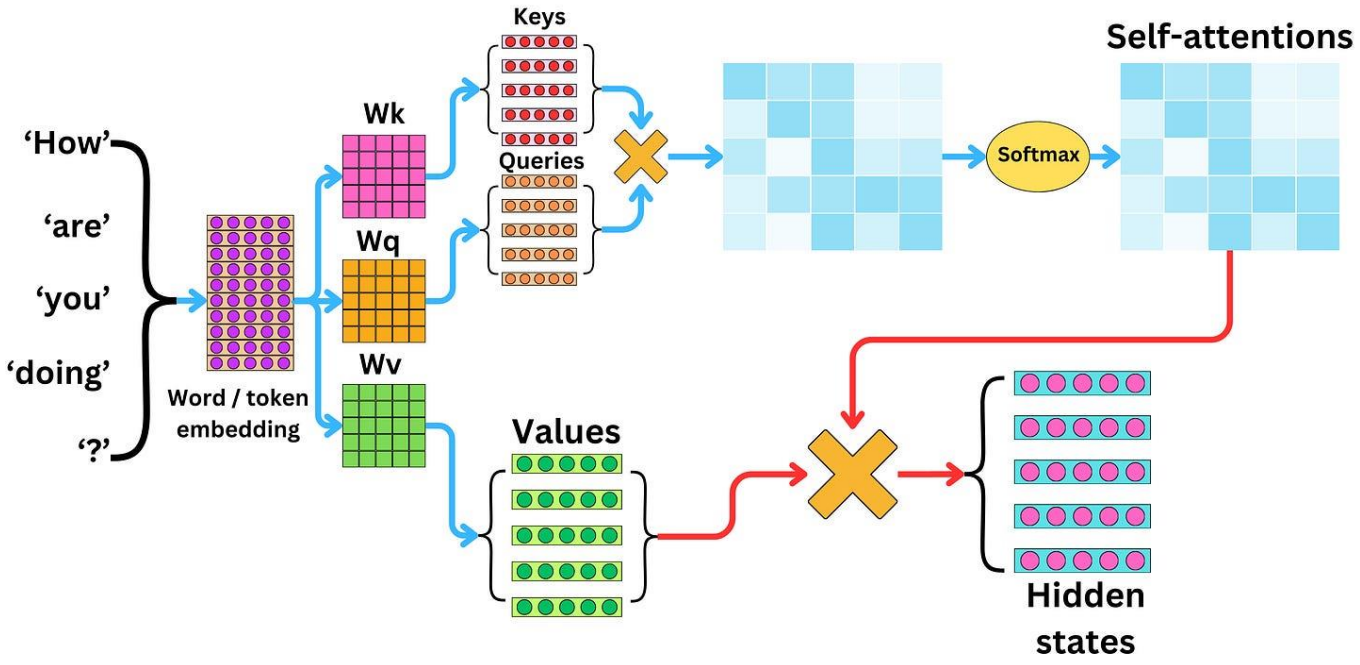
Self-attention

- Input sequence: *How are you doing*?
- Output sequence: *I am good and* _____
- Self-attention can capture the interactions between words within the input sequence and within the output sequence
- Core idea: on each step of the decoder, use direct connection to the encoder to focus on a particular part of the source sequence



Self-attention mechanism

- First compute the **keys**, **queries**, and **values** vectors
- Next compute the **matrix multiplication** between the keys and queries
- After a **softmax transformation**, this matrix of interactions is called the **attention matrix**
- Multiply attention matrix by values vector to obtain **hidden states**



https://newsletter.theaiedge.io/p/understanding-the-self-attention

Bidirectional encoder representations from transformers

- **Bidirectional**: understand the context of words in a sentence by considering both the words before and after the target word
- BERT-base: 12 transformer layers (encoders), 768 hidden dimensions, 12 attention heads; Total parameters: ~110 million
- **Two pre-training objectives**:
  - **Masked language model** (MLN): randomly mask some words in the input sentence and trains the model to predict these words based on context
    - Example: Input: "*The cat eats [MASK]*"; Output: "*fish*"
  - **Next sentence prediction** (NSP): trains the model to determine if a given sentence follows another logically
    - Example: Sentence A: "*I love traveling.*" Sentence B: "*During the Thanksgiving break, I went to Miami.*" (related) Sentence B: "*We have final project presentations this week.*" (unrelated)

Kenton, Jacob Devlin Ming-Wei Chang, and Lee Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." *Proceedings of naacL-HLT*. Vol. 1. 2019.

Generative Pre-trained Transformer (GPT)

- Generate human-like text and perform various natural language processing (NLP) tasks
- **Autoregressive model** (unidirectional): predict the next word in a sequence based on previous words
  - Example: Input: "*The sky is*"; Output: "*blue*"
- Training objective: maximize the probability of next word given preceding words
- Tokenization: Text is broken into small units called tokens (e.g., words, subwords, characters)