Introduction to Transformer

Yu Meng
University of Virginia
yumeng5@virginia.edu

Sep 30, 2024

Join at
slido.com
#1878 355

Announcement

No class/instructor office hours on Wednesday (10/02) due to instructor's travel

Overview of Course Contents

- Week 1: Logistics & Overview
- Week 2: N-gram Language Models
- Week 3: Word Senses, Semantics & Classic Word Representations
- Week 4: Word Embeddings
- Week 5: Sequence Modeling and Neural Language Models
- Week 6-7: Language Modeling with Transformers (Pretraining + Fine-tuning)
- Week 8: Large Language Models (LLMs) & In-context Learning
- Week 9-10: Knowledge in LLMs and Retrieval-Augmented Generation (RAG)
- Week 11: LLM Alignment
- Week 12: Language Agents
- Week 13: Recap + Future of NLP
- Week 15 (after Thanksgiving): Project Presentations

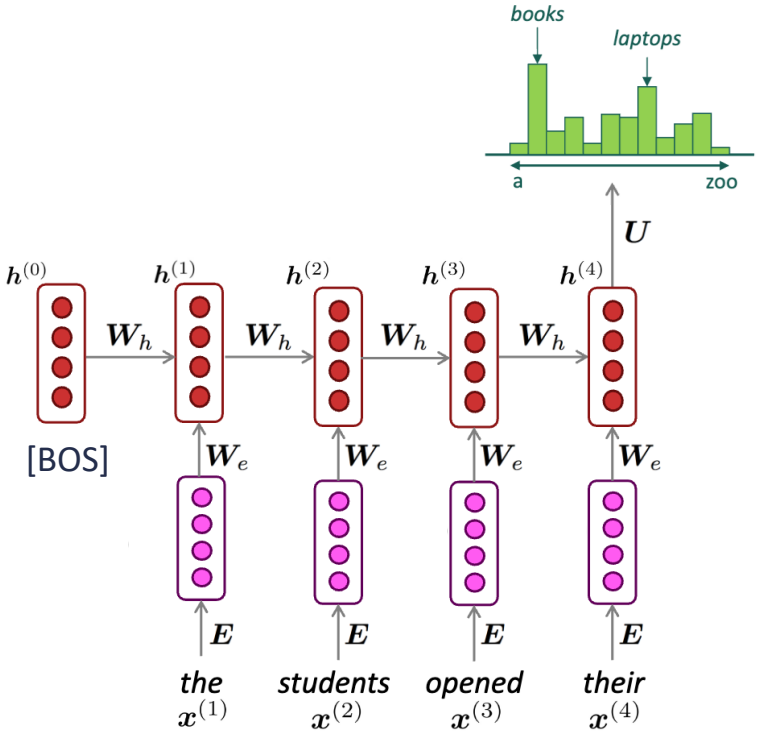
(Recap) RNN Computation

- 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)



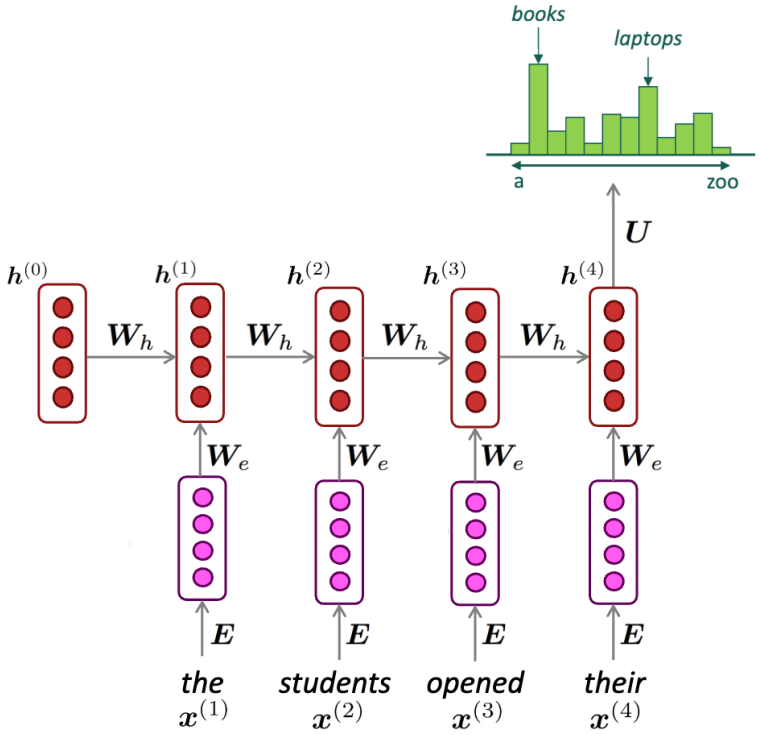
(Recap) RNN Computation

- Input: $\boldsymbol{x} = [\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \cdots, \boldsymbol{x}^{(N)}]$
- Initialize $\boldsymbol{h}^{(0)}$
- For each time step (word) in the input:
  - Compute hidden states:

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

  - Compute output:

$$\boldsymbol{y}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)}\right)$$



(Recap) RNN for Language Modeling

- Recall that language modeling predicts 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: $\boldsymbol{y}^{(t-1)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t-1)}\right) := f\left(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t-2)}, \boldsymbol{x}^{(t-1)}\right)$

$\boldsymbol{h}^{(t-1)}$ 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 $\boldsymbol{x}^{(t-2)}$: $\boldsymbol{h}^{(t-2)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-3)} + \boldsymbol{W}_e\boldsymbol{x}^{(t-2)}\right) := g\left(\boldsymbol{h}^{(t-3)}, \boldsymbol{x}^{(t-2)}\right)$

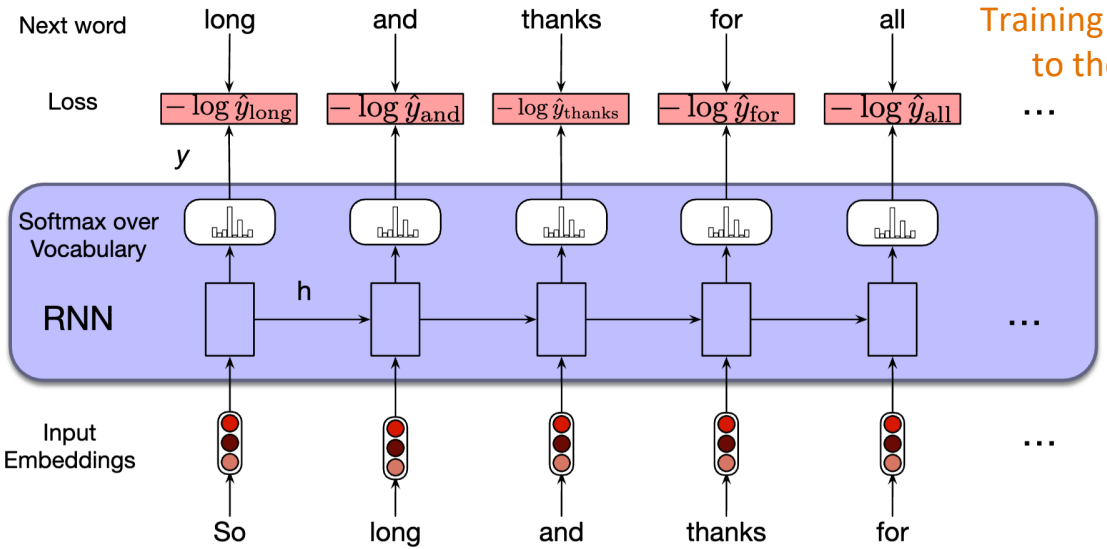
$\vdots \qquad\qquad \vdots$

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

(Recap) RNN Language Model Training

Train the output probability at each time step to predict the next word

$$\mathcal{L}_{\text{LM}}(\boldsymbol{x}) = \frac{1}{n}\sum_{t=1}^{n}\mathcal{L}_{\text{CE}}\left(\hat{\boldsymbol{y}}^{(t)}, \boldsymbol{y}^{(t)}\right) = \frac{1}{n}\sum_{t=1}^{n} -\log \hat{y}^{(t)}_{x^{(t)}} = \frac{1}{n}\sum_{t=1}^{n} -\log \frac{\exp\left(x^{(t)}\right)}{\sum_{w' \in \mathcal{V}} \exp(w')}$$



Training target is the input shifted to the left by one time step

Figure source: https://web.stanford.edu/~jurafsky/slp3/8.pdf

(Recap) RNN for Text Generation

- Input [BOS] (beginning-of-sequence) token to the model
- Sample a word from the softmax distribution at the first time step
- Use the word embedding of that first word as the input at the next time step
- Repeat until the [EOS] (end-of-sequence) token is generated

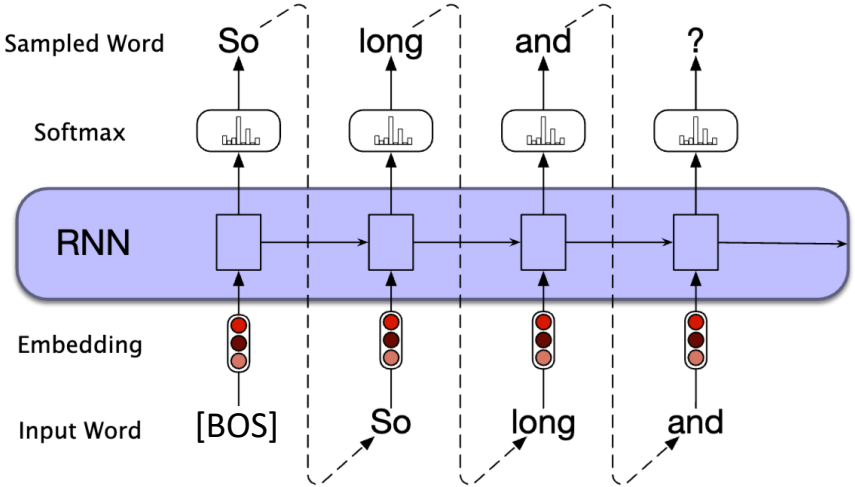


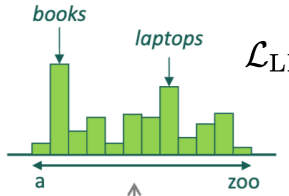
Figure source: https://web.stanford.edu/~jurafsky/slp3/8.pdf

Agenda

- RNN Limitations
- Advanced RNNs
- Transformer Overview

(Recap) Vanishing & Exploding Gradient

- Gradient signal from far away can be unstable!
- Vanishing gradient = many small gradients multiplied together
- Exploding gradient = many large gradients multiplied together

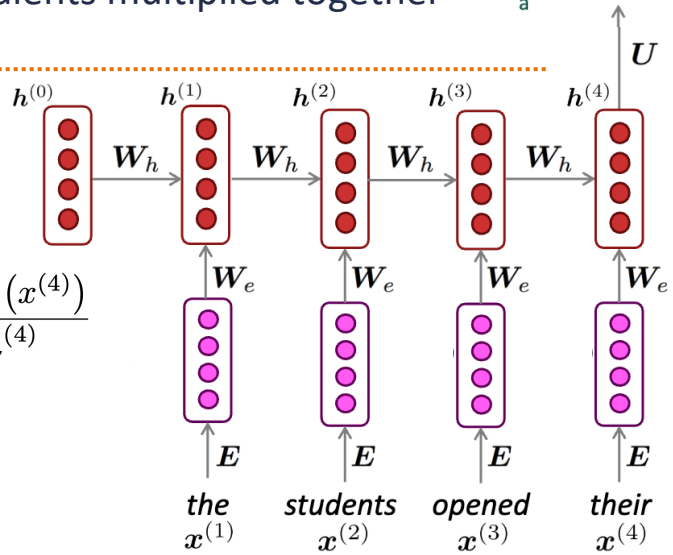


$\mathcal{L}_{\text{LM}}\left(x^{(4)}\right)$

Gradient backpropagation

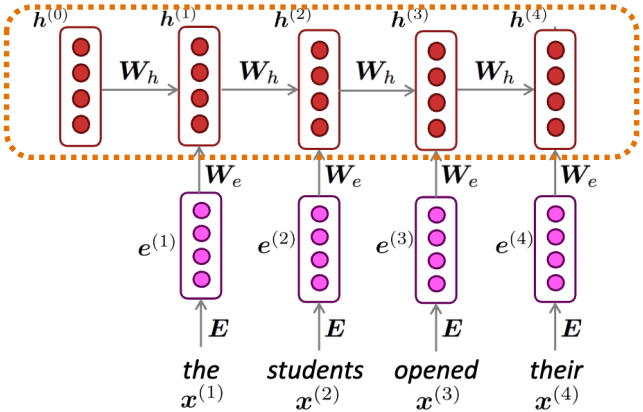
Lots of gradient multiplications!

$$\frac{\partial \mathcal{L}_{\text{LM}}\left(x^{(4)}\right)}{\partial \boldsymbol{h}^{(0)}} = \frac{\partial \boldsymbol{h}^{(1)}}{\partial \boldsymbol{h}^{(0)}} \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}} \frac{\partial \mathcal{L}_{\text{LM}}\left(x^{(4)}\right)}{\partial \boldsymbol{h}^{(4)}}$$



(Recap) Long-Term Dependencies

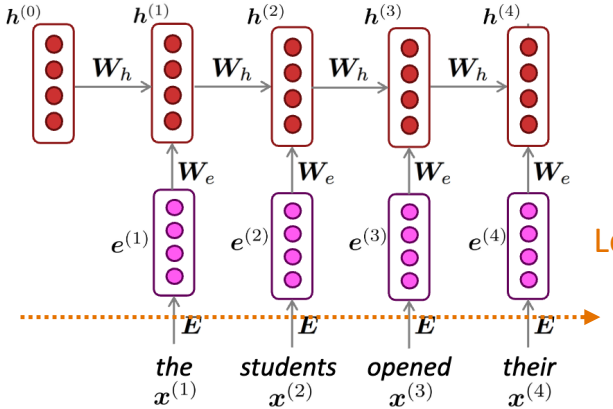
- RNNs are theoretically capable of remembering information over arbitrary lengths of input, but they struggle in practice with long-term dependencies
- RNNs use a fixed-size hidden state to encode an entire sequence of variable length; the hidden state is required to compress a lot of information
- RNNs might give more weight to the most recent inputs and may ignore or "forget" important information at the beginning of the sentence while processing the end



Fixed size hidden states!

(Recap) Lack of Bidirectionality

- RNNs process the input sequence step by step from the beginning to the end (left to right for English)
- At each time step, the hidden state only has access to the information from the past without being able to leverage future contexts
- Example: “The bank is on the river” the word “bank” can be correctly disambiguated only if the model has access to the word “river” later in the sentence

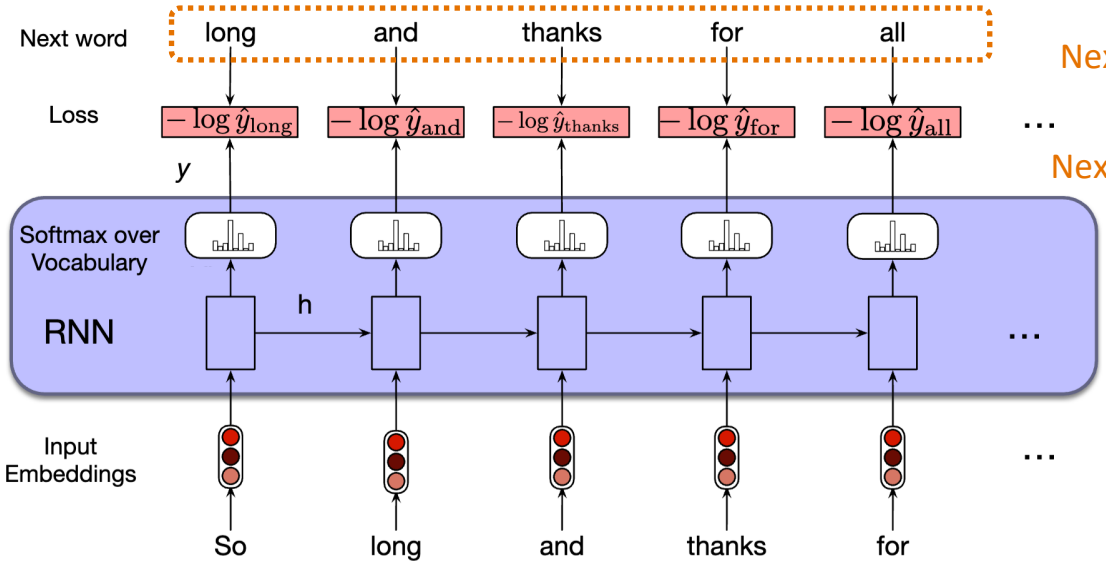


Left to right processing

Join at
slido.com
#1878 355

Exposure Bias

- **Teacher forcing/exposure bias**: during RNN training, the model always receives the **correct** next word from the training data as input for the next step
- When the model has to predict sequences on its own, it may perform poorly if it hasn't learned how to correct its own mistakes



During training:
Next word = actual next word

During generation:
Next word = model's prediction

Agenda

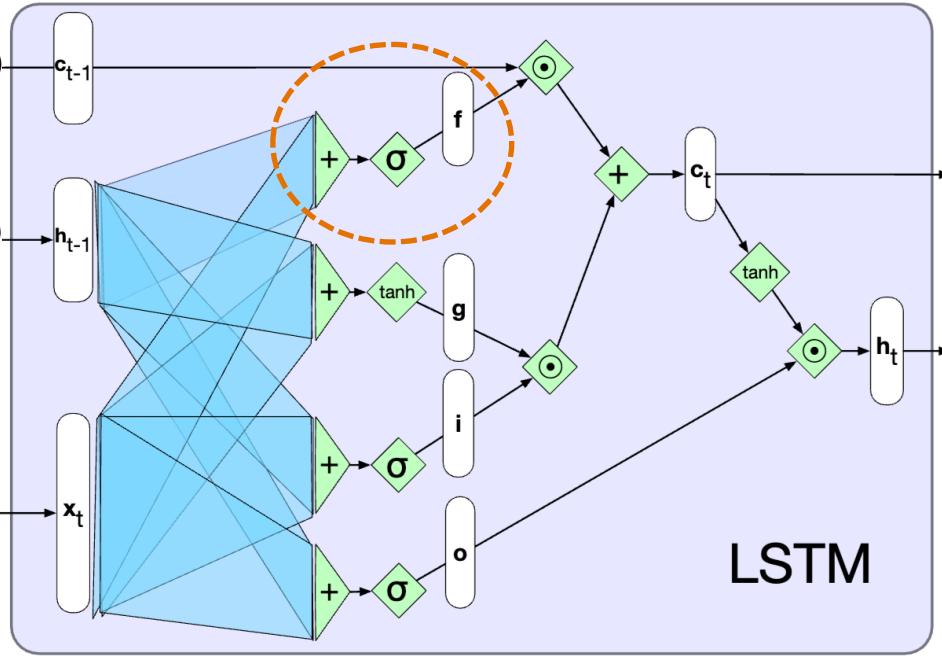
- RNN Limitations
- Advanced RNNs
- Transformer Overview

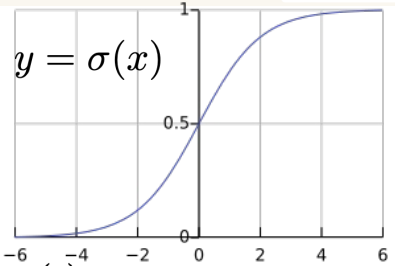
Long Short-Term Memory (LSTM)

- Challenge in RNNs: information encoded in hidden states tends to be local; distant information gets lost
- LSTM design intuition:
  - Remove information no longer needed from the context
  - Add information likely to be needed for future time steps
- Inputs at each time step:
  - Word embedding of the current word
  - Hidden state from the previous time step
  - **Memory/cell state**
- Three gates:
  - Forget gate
  - Add/input gate
  - Output gate

LSTM Computation (Forget Gate)

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{U}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_f \boldsymbol{x}^{(t)}\right)$$





$\boldsymbol{c}^{(t)}$

Decides what information to discard from the cell state

$\boldsymbol{h}^{(t)}$

Figure source: https://web.stanford.edu/~jurafsky/slp3/8.pdf

LSTM Computation (Add/Input Gate)

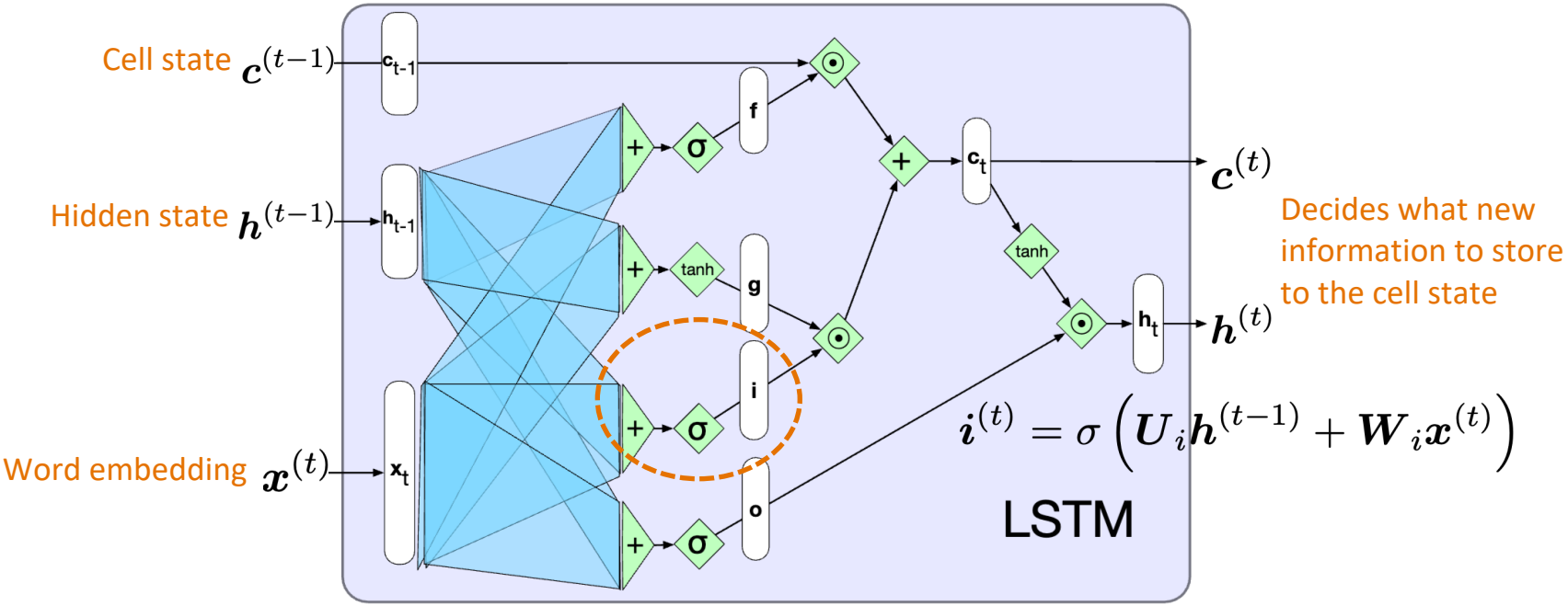
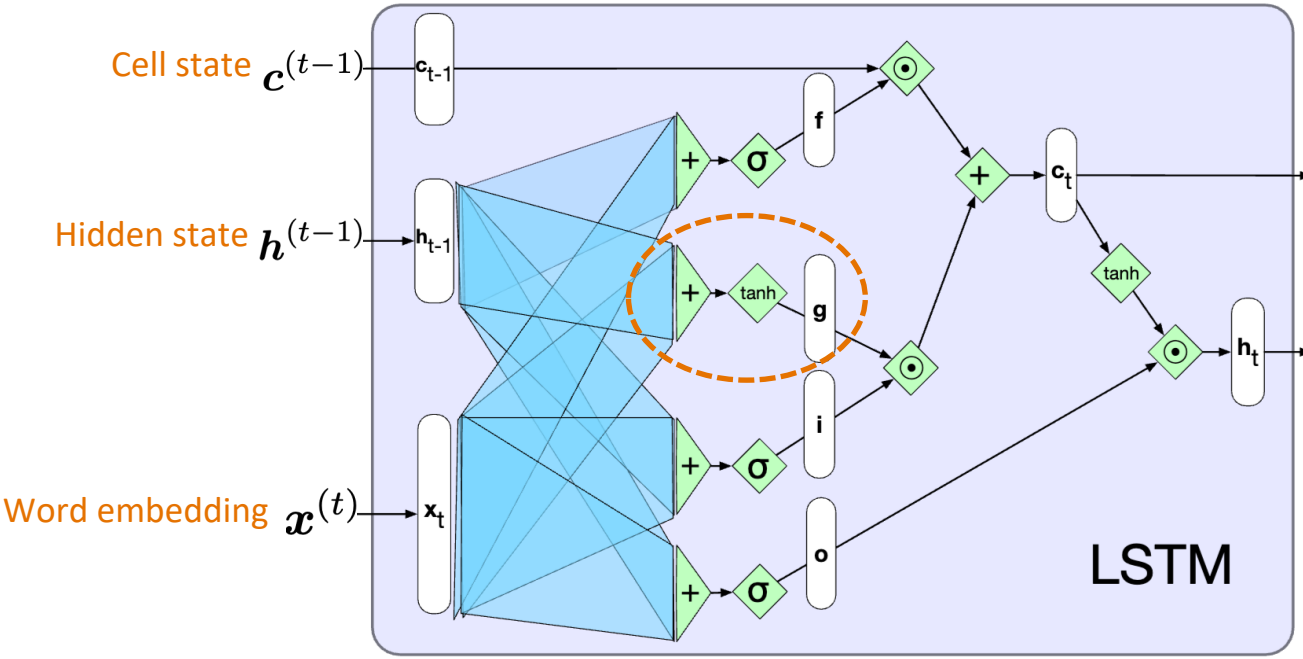
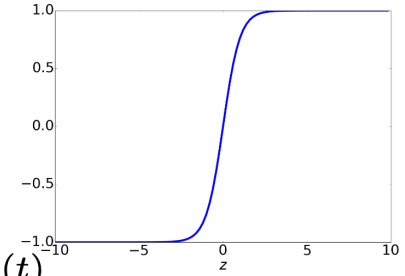


Figure source: https://web.stanford.edu/~jurafsky/slp3/8.pdf

LSTM Computation (Candidate Cell State)

$$\boldsymbol{g}^{(t)} = \tanh\left(\boldsymbol{U}_g \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_g \boldsymbol{x}^{(t)}\right)$$





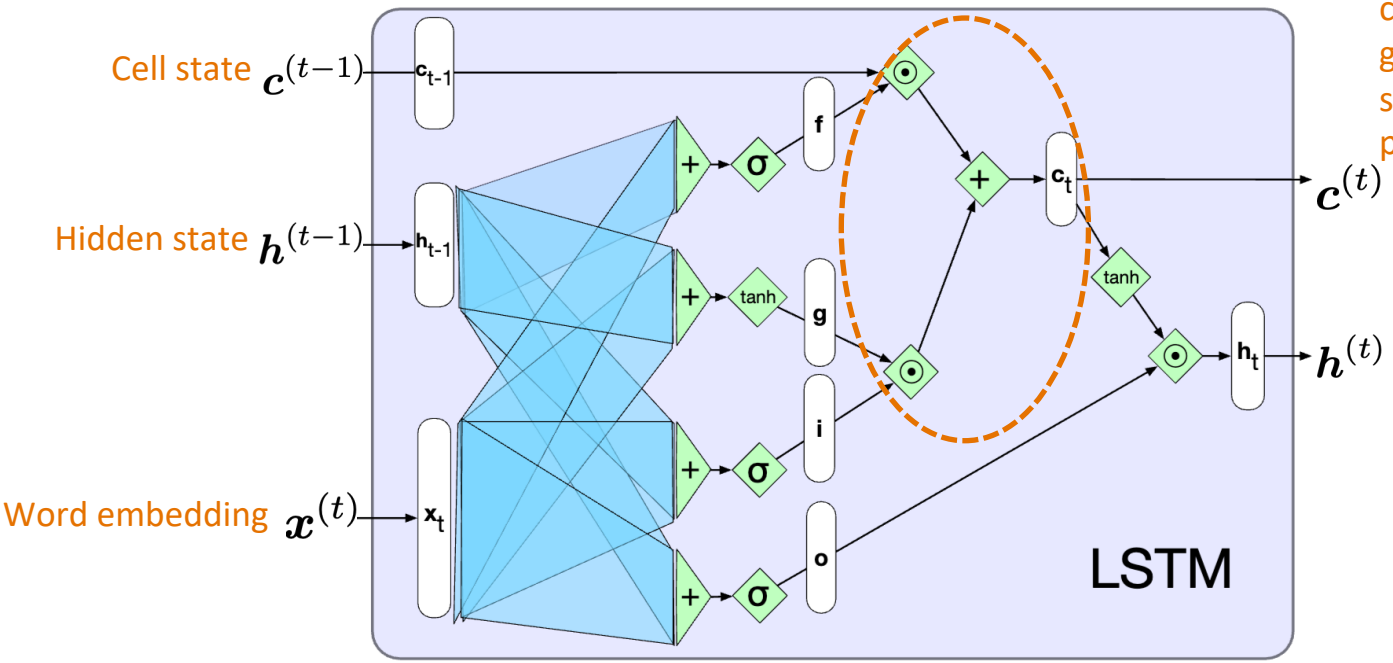
$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Compute information needed from the previous hidden state and current inputs

Figure source: https://web.stanford.edu/~jurafsky/slp3/8.pdf

LSTM Computation (Cell State Update)

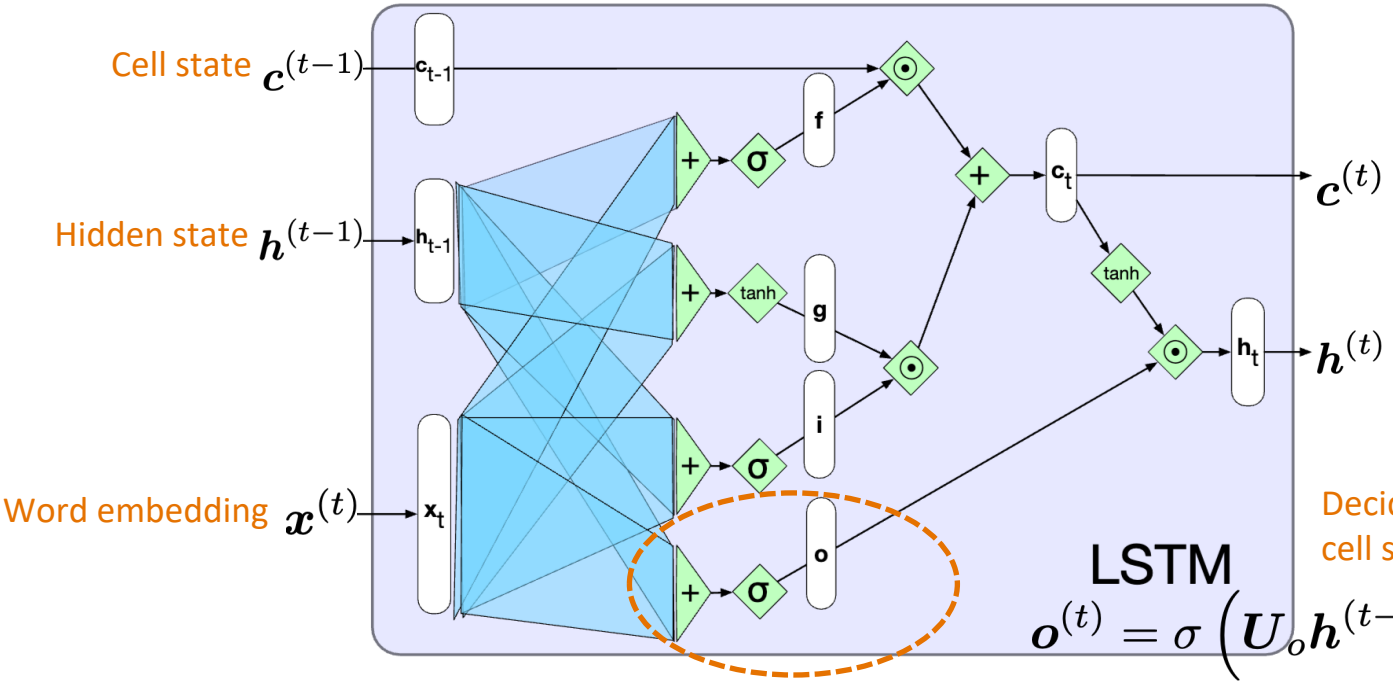
$$\boldsymbol{c}^{(t)} = \boldsymbol{i}^{(t)} \odot \boldsymbol{g}^{(t)} + \boldsymbol{f}^{(t)} \odot \boldsymbol{c}^{(t-1)}$$



Cell state updated by combining the input gate, candidate cell state, forget gate & previous cell state

Figure source: https://web.stanford.edu/~jurafsky/slp3/8.pdf

LSTM Computation (Output Gate)



Decides what parts of the cell state will be output

Figure source: https://web.stanford.edu/~jurafsky/slp3/8.pdf

LSTM Computation (Hidden State Update)

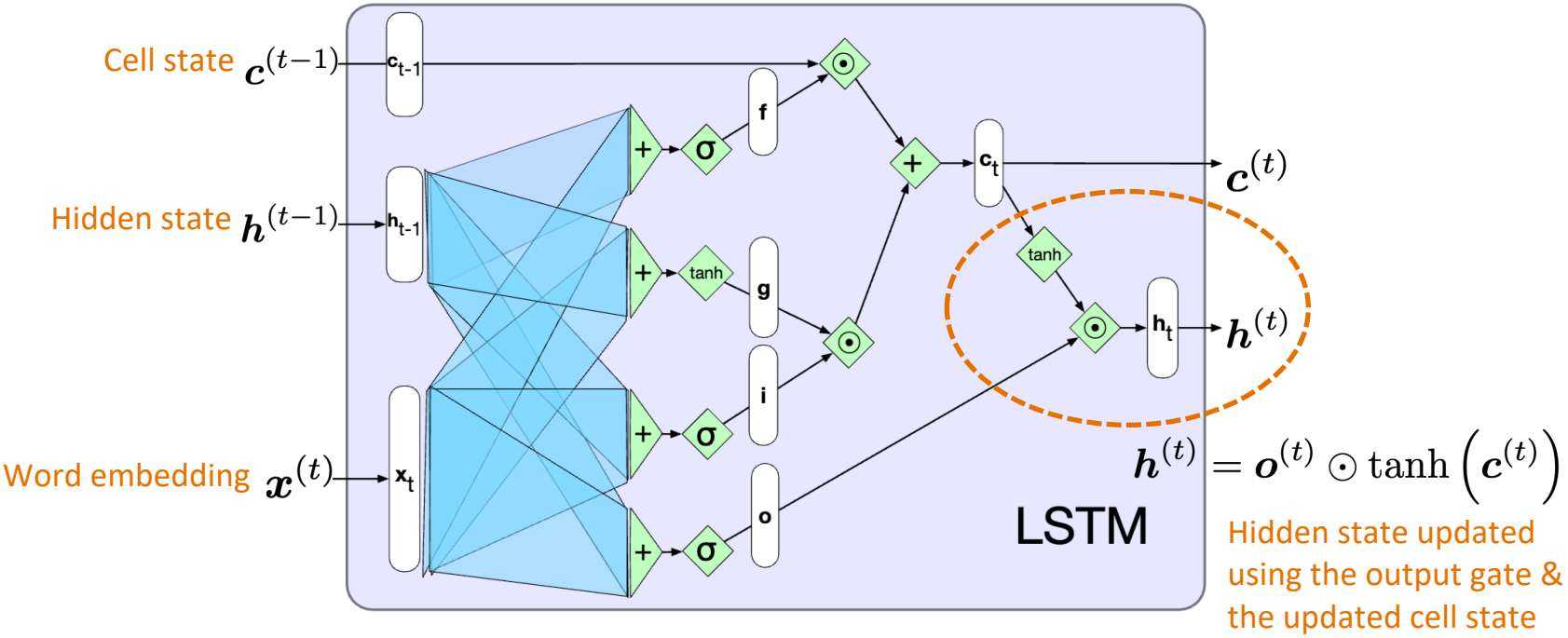


Figure source: https://web.stanford.edu/~jurafsky/slp3/8.pdf

Bidirectional RNNs

- Separate models are trained in the forward and backward directions
- Hidden states from both RNNs are concatenated as the final representations

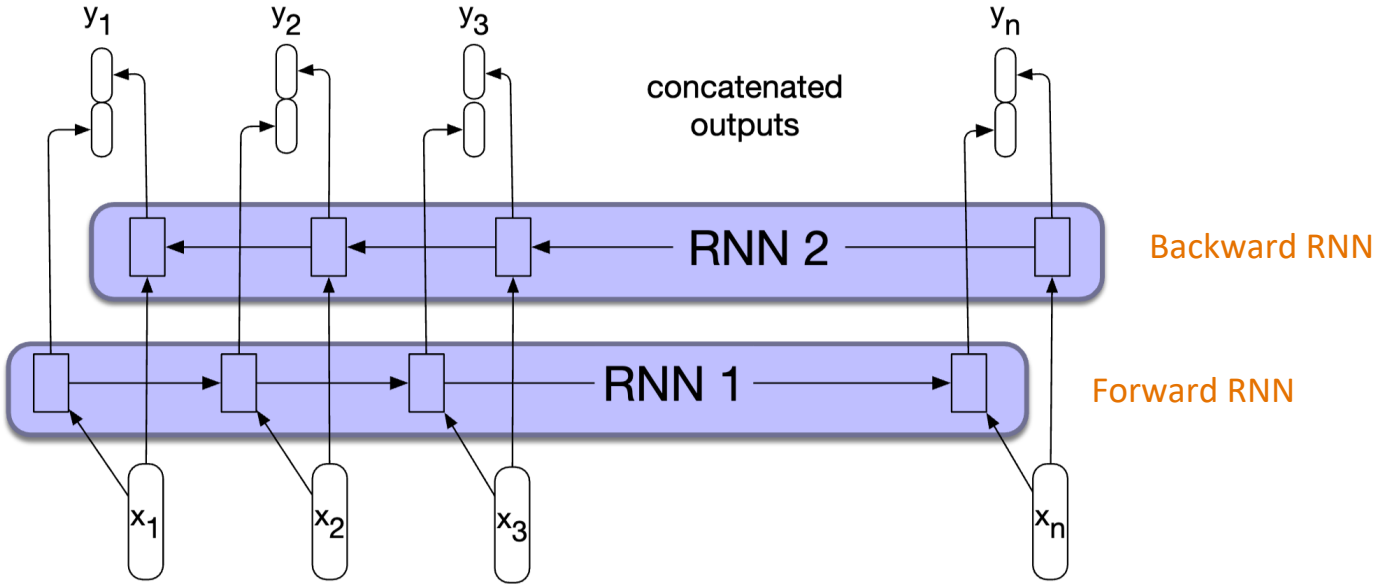


Figure source: https://web.stanford.edu/~jurafsky/slp3/8.pdf

Deep RNNs

- We can stack multiple RNN layers to build deep RNNs
- The output of a lower level serves as the input to higher levels
- The output of the last layer is used as the final output

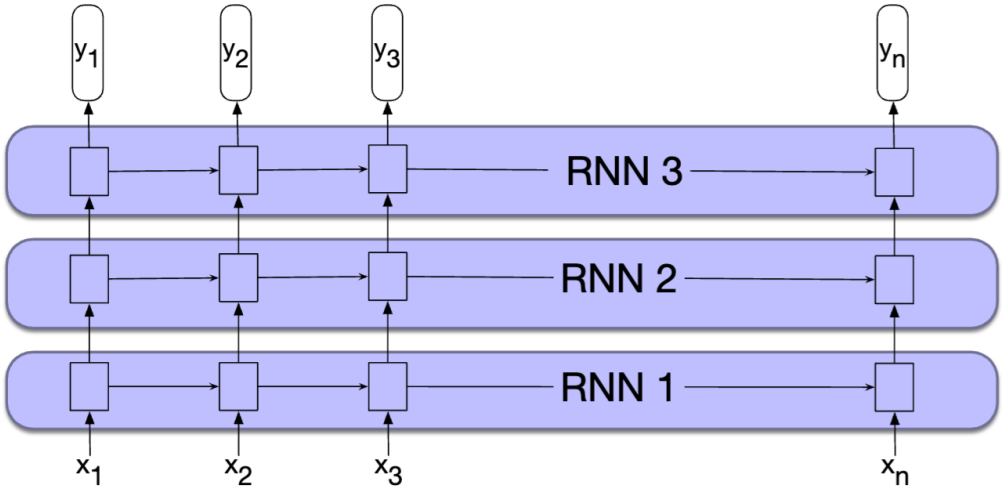


Figure source: https://web.stanford.edu/~jurafsky/slp3/8.pdf

Summary: Sequence Modeling

- Sequence modeling goals:
  - Learn context-dependent representations
  - Capture long-range dependencies
  - Handle complex relationships among large text units
- Use deep learning architectures to understand, process, and generate text sequences
- Why DNNs?
  - The multi-layer structure in DNNs mirrors the hierarchical structures in language
  - DNNs learn multiple levels of semantics across layers: low-level patterns (e.g., relations between words) in lower layers & high-level patterns (e.g., sentence meanings) in higher layers

Summary: Neural Language Models

- Address the sparsity issue in N-gram language models by computing the output distribution based on distributed representations (with semantic information)
- Simple neural language models based on feedforward networks suffer from the fixed context window issue
  - Can only model a fixed number of words (similar to N-gram assumption)
  - Increasing the context window requires adding more model parameters

Summary: Recurrent Neural Networks

- General idea: Use the same set of model weights to process all input words
- RNNs as language models
  - Theoretically able to process infinitely long sequences
  - Practically can only keep track of recent contexts
- Training issues: vanishing & exploding gradients
- LSTM is a prominent RNN variant to keep track of both long-term and short-term memories via multiple gates

Agenda

- RNN Limitations
- Advanced RNNs
- Transformer Overview

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)!

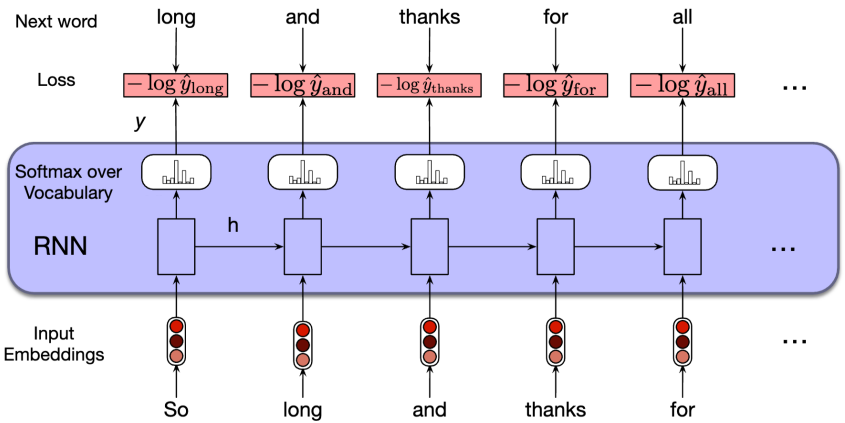
Attention Is All You Need

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)

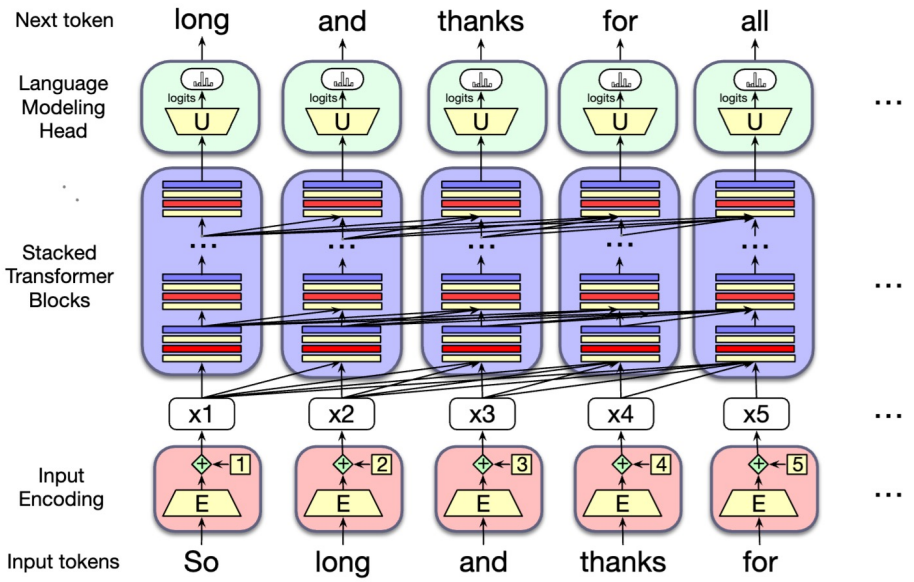


Figure source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

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 attention via attention

Transformer Layer

Each Transformer layer contains the following important components:

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

Transformer layer

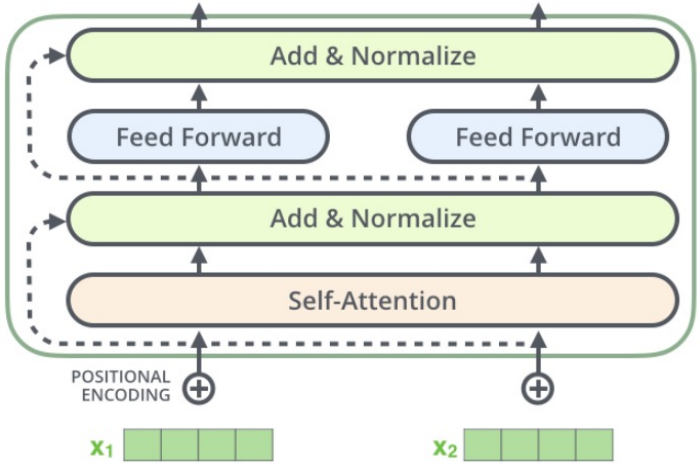


Figure source: https://jalammar.github.io/illustrated-transformer/

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 quick brown fox jumps over the lazy dog."
  - Suppose we are learning attention for the word "**jumps**"
  - With self-attention, "**jumps**" can decide which other words in the sentence it should focus on to better understand its meaning
  - Might assign high attention to "fox" (the subject) & "over" (the preposition)
  - Might assign less attention to words like "the" or "lazy"

Thank You!

Yu Meng
University of Virginia
yumeng5@virginia.edu