Self-Attention

Yu Meng
University of Virginia
yumeng5@virginia.edu

Oct 04, 2024

Join at
slido.com
#1107 551

Announcement

- Assignment 2 grades posted; reference answer released
- Contact Zhepei (tqf5qb@virginia.edu) if you have questions about your grade

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) 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

(Recap) LSTM w/ Three Gates

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

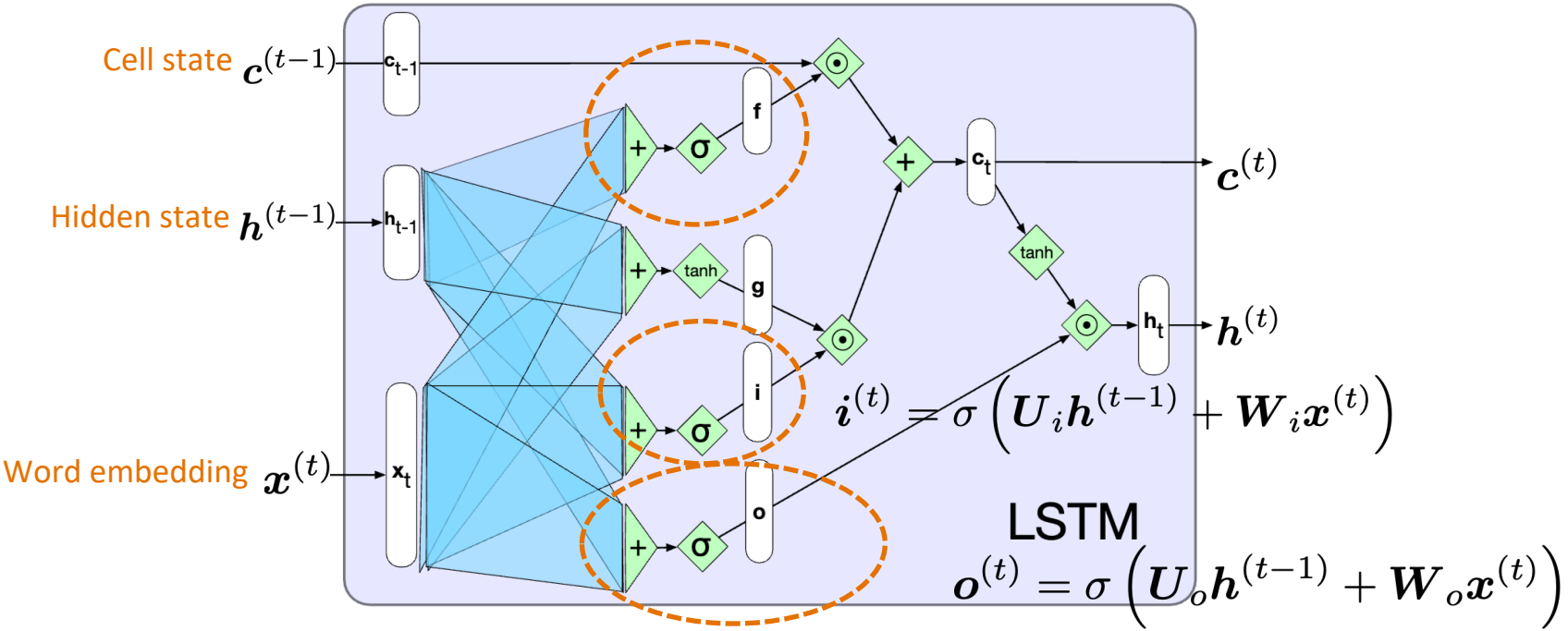


Figure source: https://web.stanford.edu/~jurafsky/slp3/8.pdf

Join at
slido.com
#1107 551

(Recap) Bidirectional RNNs

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

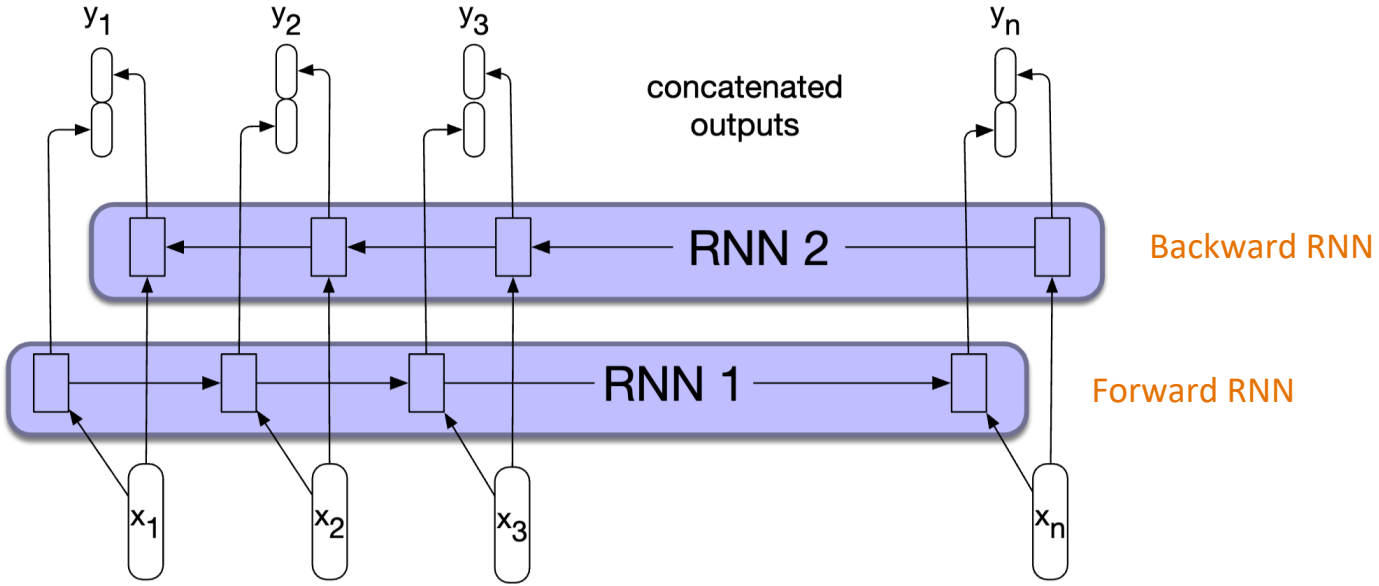


Figure source: https://web.stanford.edu/~jurafsky/slp3/8.pdf

(Recap) 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

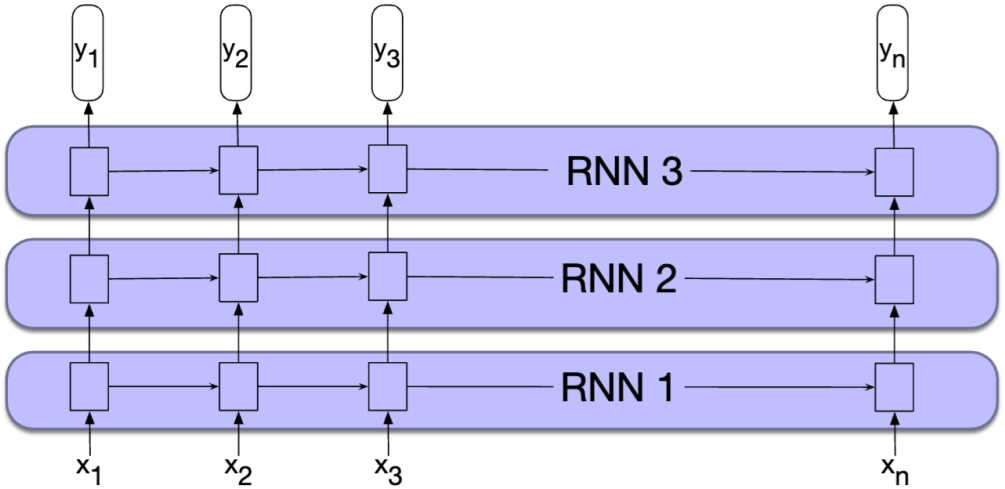


Figure source: https://web.stanford.edu/~jurafsky/slp3/8.pdf

Join at
slido.com
#1107 551

(Recap) 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

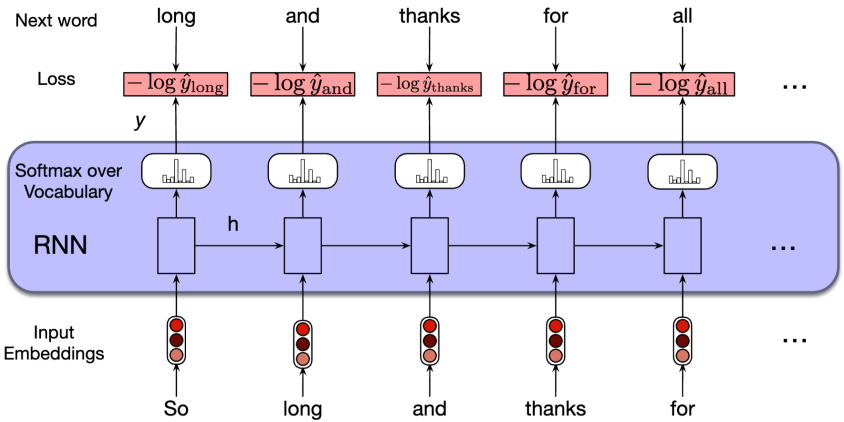
Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser*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

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

Join at
slido.com
#1107 551

Transformer vs. RNN

RNN
(recurrent computations)



Transformer
(self-attention computations)

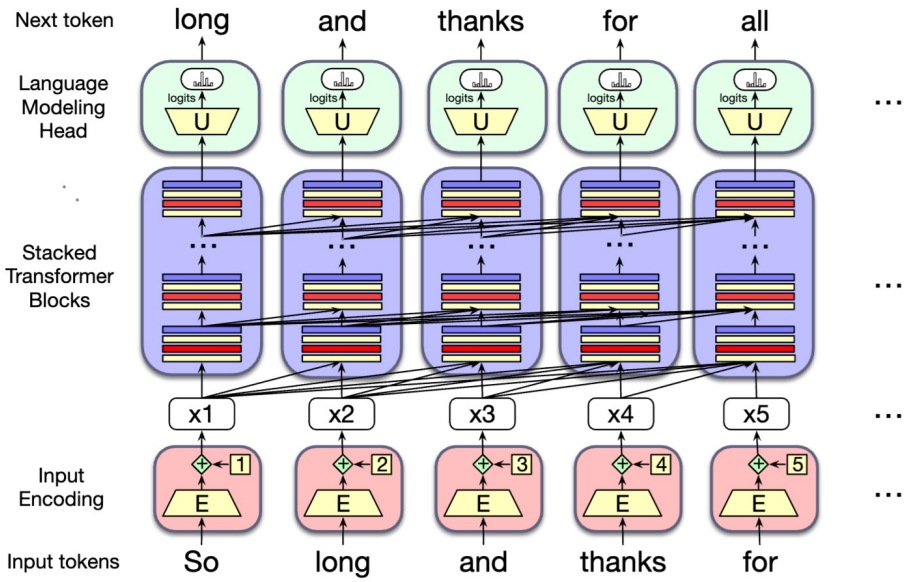


Figure source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

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

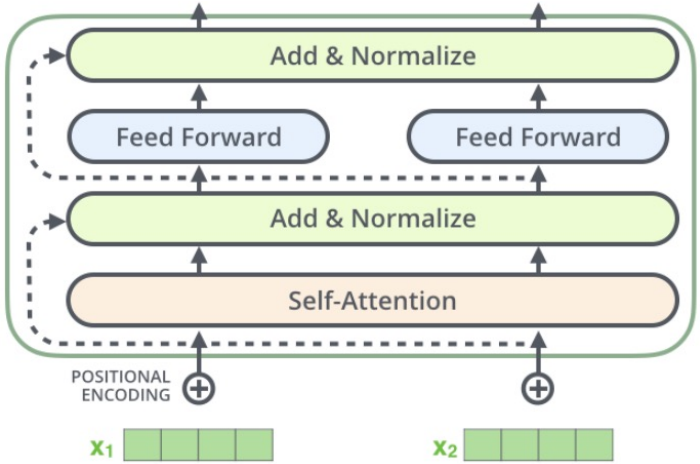


Figure source: https://jalammar.github.io/illustrated-transformer/

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 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"

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

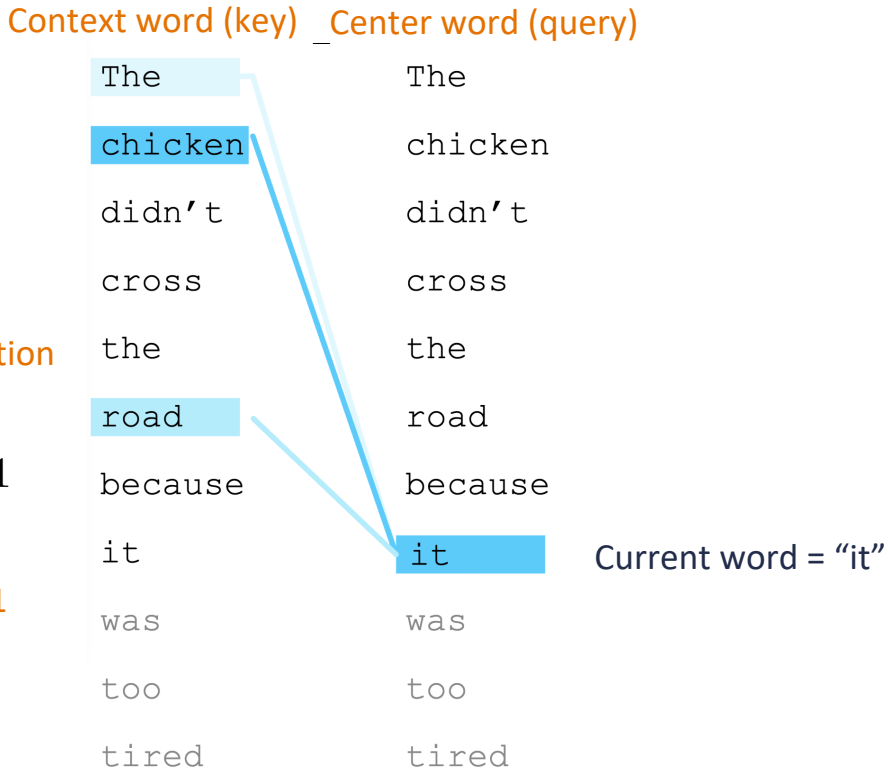


Figure source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

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

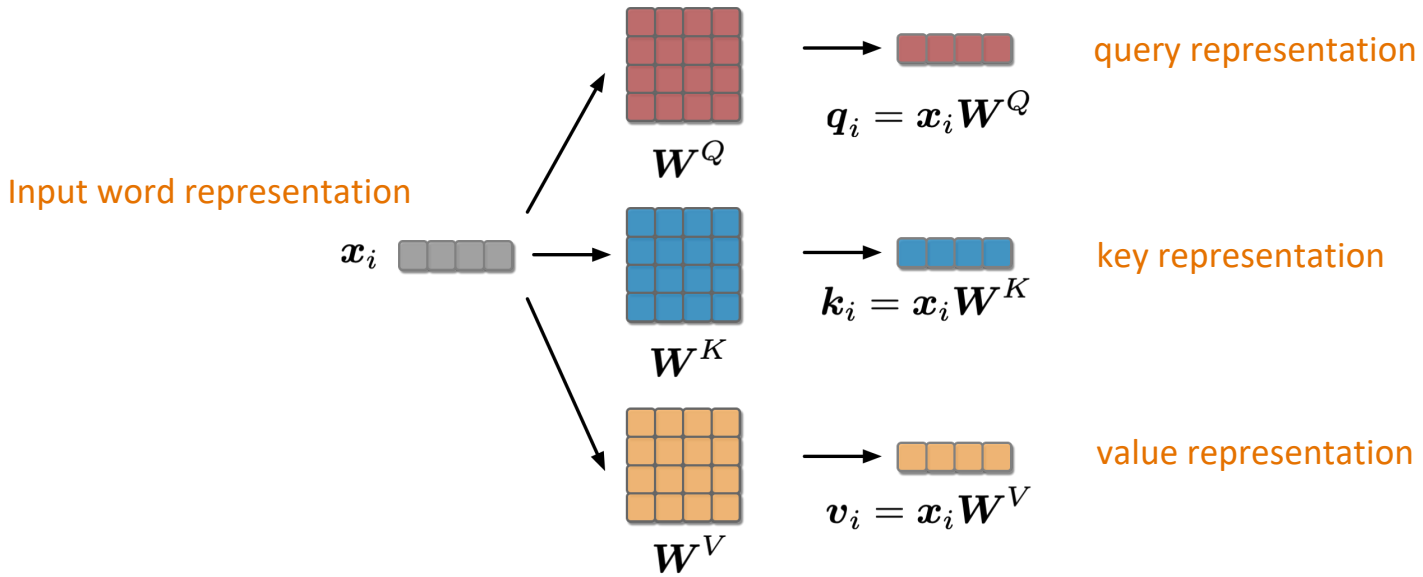
- Why use two copies of word representations for attention computation?
  - We want to reflect the different roles a word plays (as the target word being compared to others, or as the context word being compared to the target word)
  - If using the same copy of representations for attention calculation, a word will (almost) always attend to itself heavily due to high dot product with itself!

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

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

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

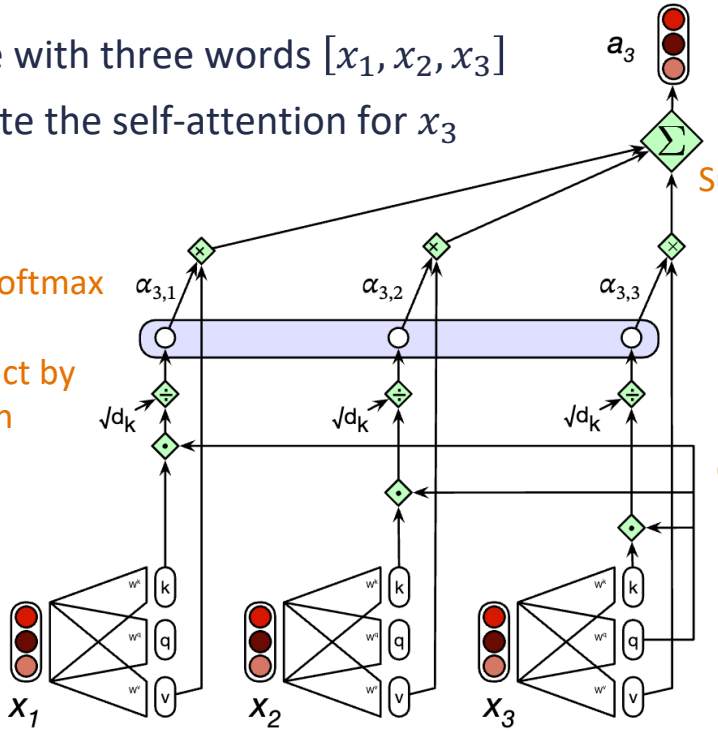
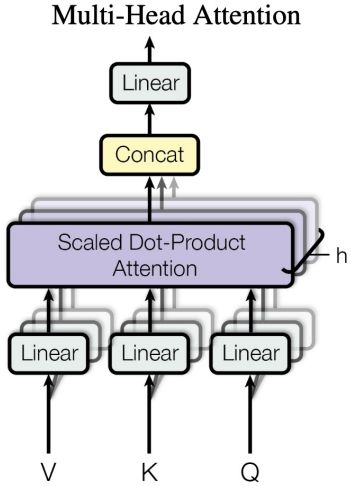


Figure source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

Join at
slido.com
#1107 551

Multi-Head Self-Attention

- Transformers use multiple attention heads for each self-attention module
- Intuition:
  - Each head might attend to the context for different purposes (e.g., particular kinds of patterns in the context)
  - Heads might be specialized to represent different linguistic relationships



Concatenate the outputs of all heads

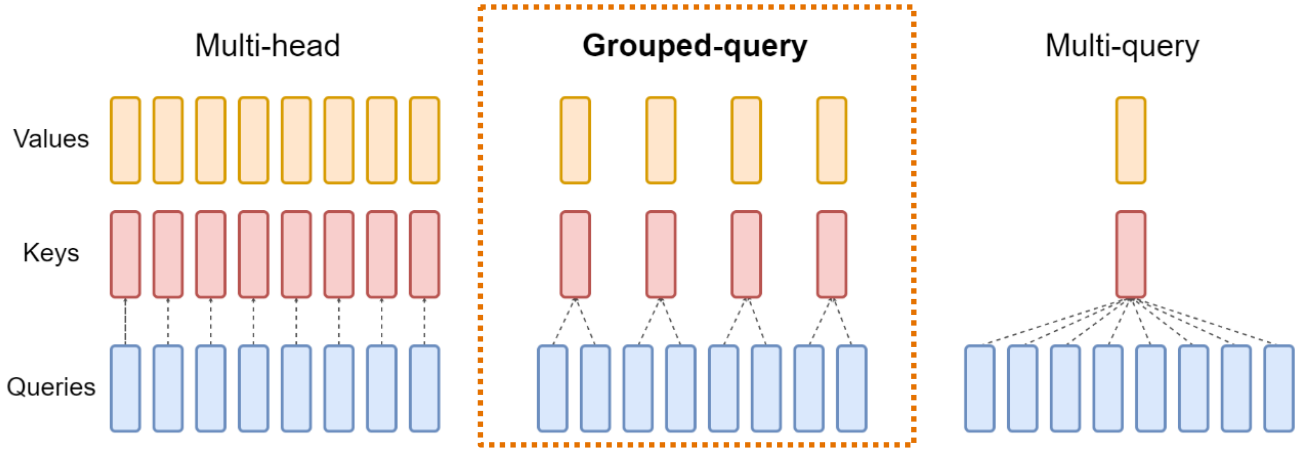
h attention heads are computed separately

Figure source: https://arxiv.org/pdf/1706.03762

Join at
slido.com
#1107 551

Multi-Head Self-Attention Variants

- Multi-query attention (Fast Transformer Decoding: One Write-Head is All You Need): share keys and values across all attention heads
- Grouped-query attention (GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints): share keys and values within groups of heads



Used in latest LLMs (e.g., Llama3)

Figure source: https://arxiv.org/pdf/2305.13245

Parallel Computation of QKV

- Self-attention computation performed for each token is independent of other tokens
- Easily parallelize the entire computation, taking advantage of the efficient matrix multiplication capability of GPUs
- Process an input sequence with N words in parallel

Compute QKV for one word: $\boldsymbol{q}_i = \boldsymbol{x}_i \boldsymbol{W}^Q \quad \boldsymbol{k}_i = \boldsymbol{x}_i \boldsymbol{W}^K \quad \boldsymbol{v}_i = \boldsymbol{x}_i \boldsymbol{W}^V \in \mathbb{R}^d$

Stacking N input vectors: $\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^Q \quad \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^K \quad \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^V \in \mathbb{R}^{N \times d}$

$$\boldsymbol{X} = \begin{bmatrix} \text{——} & \boldsymbol{x}_1 & \text{——} \\ \text{——} & \boldsymbol{x}_2 & \text{——} \\ \cdots & \cdots & \cdots \\ \text{——} & \boldsymbol{x}_N & \text{——} \end{bmatrix}$$

Parallel Computation of Attention

Attention computation can also be written in matrix form

Compute attention for one word: $\boldsymbol{a}_i = \text{Softmax}\left(\frac{\boldsymbol{q}_i \cdot \boldsymbol{k}_j}{\sqrt{d}}\right) \cdot \boldsymbol{v}_j$

Compute attention for one N words: $\boldsymbol{A} = \text{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}}\right)\boldsymbol{V}$

Attention matrix

q1·k1	q1·k2	q1·k3	q1·k4
q2·k1	q2·k2	q2·k3	q2·k4
q3·k1	q3·k2	q3·k3	q3·k4
q4·k1	q4·k2	q4·k3	q4·k4

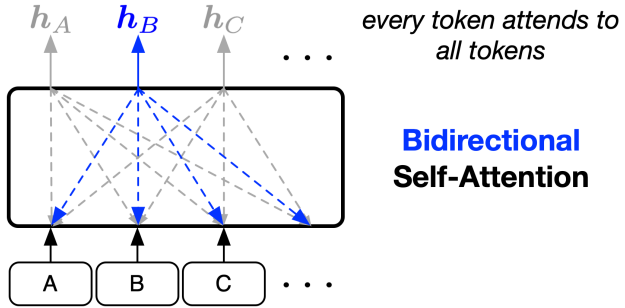
N

N

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

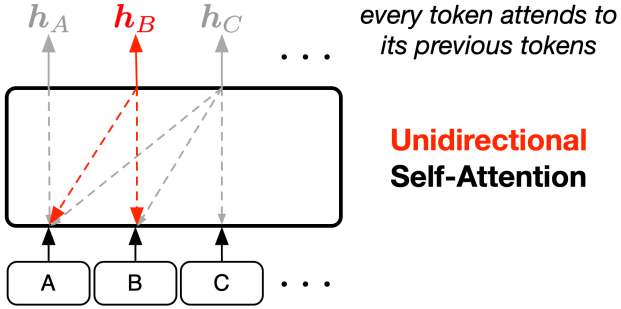
Bidirectional vs. Unidirectional Self-Attention

- Self-attention can capture different context dependencies
- **Bidirectional** self-attention:
  - Each position to attend to all other positions in the input sequence
  - Transformers with bidirectional self-attention are called Transformer **encoders** (e.g., BERT)
  - Use case: natural language understanding (NLU) where the entire input is available at once, such as text classification & named entity recognition



Bidirectional vs. Unidirectional Self-Attention

- Self-attention can capture different 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



upper-triangle portion set to -inf

q1·k1	−∞	−∞	−∞
q2·k1	q2·k2	−∞	−∞
q3·k1	q3·k2	q3·k3	−∞
q4·k1	q4·k2	q4·k3	q4·k4

N

N

Join at
slido.com
#1107 551

Position Encoding

- Motivation: inject positional information to input vectors

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

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

When $\boldsymbol{x}$ is word embedding, $\boldsymbol{q}$ and $\boldsymbol{k}$ do not have positional information!

- How to know the word positions in the sequence? Use position encoding!

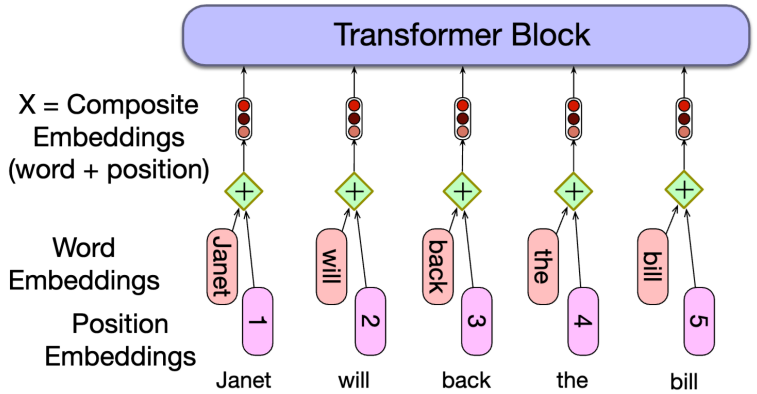


Figure source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

Position Encoding Methods

- Absolute position encoding (the original Transformer paper)
  - Learn position embeddings for each position
  - Not generalize well to sequences longer than those seen in training
- Relative position encoding (Self-Attention with Relative Position Representations)
  - Encode the relative distance between words rather than their absolute positions
  - Generalize better to sequences of different lengths
- Rotary position embedding (RoFormer: Enhanced Transformer with Rotary Position Embedding)
  - Apply a rotation matrix to the word embeddings based on their positions
  - Incorporate both absolute and relative positions
  - Generalize effectively to longer sequences
  - Widely-used in latest LLMs

Summary

- Motivation: weigh the importance of different words in a sequence when processing a specific word
- Implementation: represent each word with three vectors:
  - Query: the current word that seeks information
  - Key: context word to be retrieved information from
  - Value: semantic content to be aggregated as the new word representation
- Allow parallel computation of all input words
- Usually deployed with multiple heads to capture various linguistic relationships
- Can be either unidirectional (only attend to previous words) or bidirectional (attend to all words)
- Need to use position encodings to inject positional information

Thank You!

Yu Meng
University of Virginia
yumeng5@virginia.edu