Transformers

CS 6120 Natural Language Processing

Northeastern University

Si Wu

Some slides borrowed from Jurafsky & Martin Chapter 8

Logistics

- Coding assignment 2 due tonight.
- Coding assignment 1 grade will be released tonight.

- Today: the architecture of transformers and self-attention
- Next lecture: more in-depth discussion about different kinds of transformer models and masked language models

More on the class project

Eventually in your final report, there will be these components at least:

- Problem statement,
- Prior /related work
- Data
- Models
- Experiment
- Analysis
- Conclusion

You are welcome to ask instructors or TAs about your project idea, but best think through these before asking!

Make sure you pinpoint the exact problem in your problem statement

So your project needs to be sensible and have depth:

- Sensible/reasonable: could be finished before December, not a full-on research project.
- Interesting: it could be an open research project (e.g. poetry generation) that's difficult to evaluate; or, if you work on a very traditional, well-researched area, you need to provide some insight: comparing different approach, testing your own novel method, come up with new models or efficient implementation, or better evaluation that you come up with
- In short, you should have new insight from working on this project, that other people don't have.

Introduction

- Some of you already raised concerns in the last lecture about the “efficiency” of RNN:
  - vanishing gradient problem, memory, **parallelizability**.
- And we discussed these drawbacks of RNN, and said that’s why we keep inventing new LMs!
  - Parallelizability especially at scale
    - Sure, it’s not completely unparallelizable, but due to its sequential nature
  - Vanishing gradient: problematic for long-term dependency

- Today: a much better LM → transformer!
  - Mainly about autoregressive LMs, more will unfold over the next few lectures, e.g. masked language models and bidirectional transformers.

Components of a transformer

- 3 components:
  - Input encoding components
  - Transformer blocks (each consists of multi-head attention layer, FFNN, layer normalization steps)
  - Language modeling heads

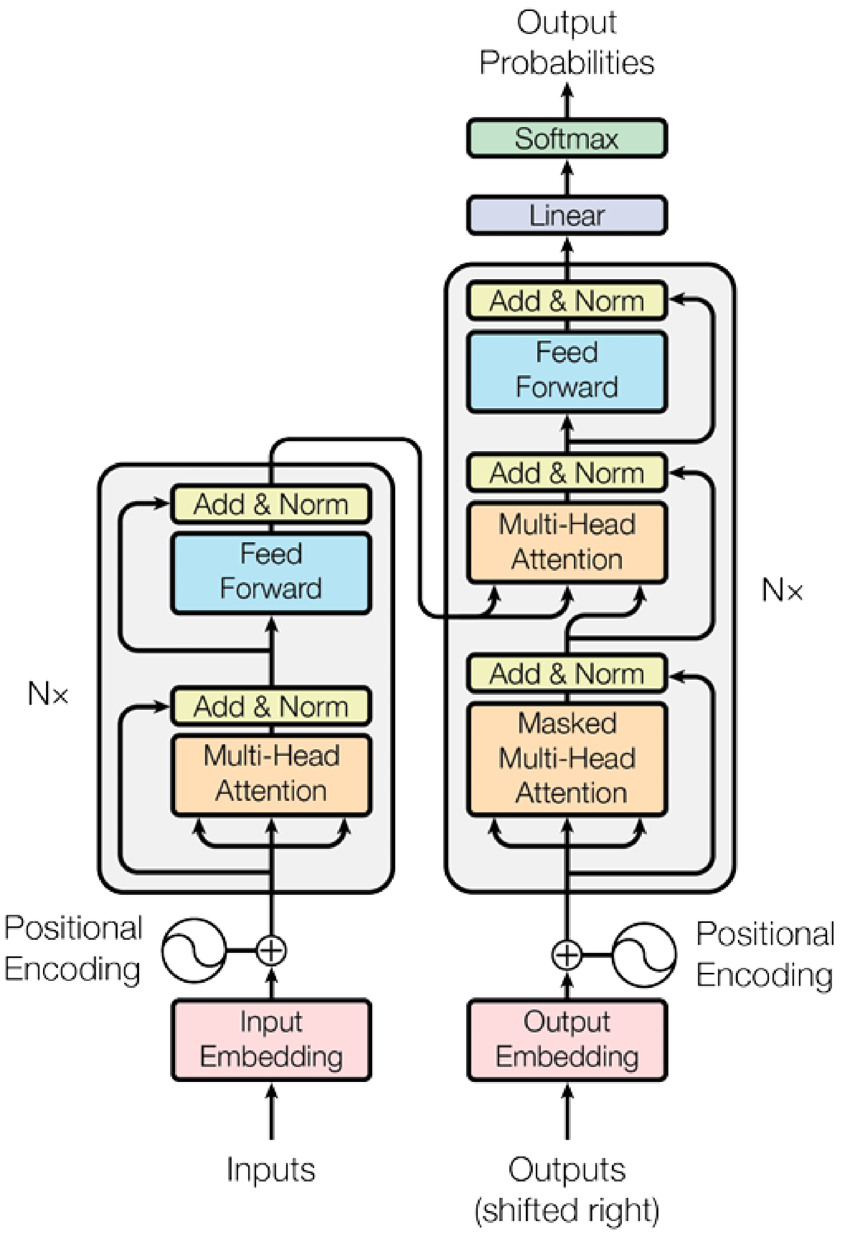


Figure from the original transformer paper Attention Is All You Need

Attention

Examples

Generating from left to right, use **are** instead of **is** to refer to keys. (grammar)

The keys to the cabinet are on the table

Imagine if we use "they" to refer to a group of people after a whole paragraph describing a party.

I walked along the pond, and noticed one of the tress along the bank

Which word sense of bank?

Recall lecture on linguistics where we talked about ambiguity, dependencies, polysemy. But also long-term dependency from RNN lecture.

Many lecture ago

When talking about embeddings, we mentioned those were **static embeddings**, and that we would talk about **contextual embeddings**

Autoregressive generation (left to right)

The chicken didn’t cross the road because **it …**

The chicken didn’t cross the road because **it** was too tired

The chicken didn’t cross the road because **it** was too wide

Attention from last lecture

In the RNN lecture, we introduced attention

Solution: Attention!

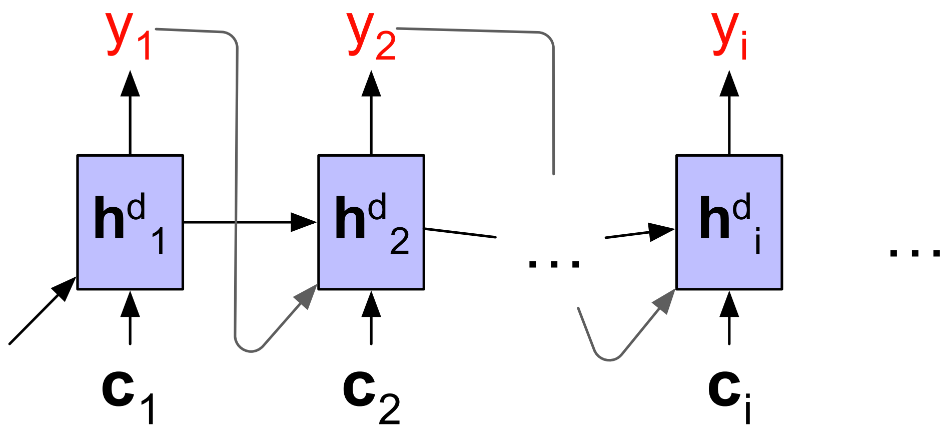
- Instead of being taken from the last hidden state, the **context** it’s a *weighted average* of all the hidden states of the encoder.
- This weighted average is also informed by the state of the decoder right before the current token *i*.

$$\mathbf{c}_i = f(\underline{\mathbf{h}^e_1 \ldots \mathbf{h}^e_n}, \boxed{\mathbf{h}^d_{i-1}})$$

“weighted average” meaning, c_i can attend to a particular part of the input text that is relevant to token I, which is what the decoder is trying to produce

Attention

$$\mathrm{h}_i^d = g(\hat{y}_{i-1}, \mathrm{h}_{i-1}^d, \mathrm{c}_i)$$



How to compute c_i ? How to decide what to pay attention to?

- One way is similarity!
  - Using similarity as a scoring function between last decoder state and each encoder hidden state
  - Simplest such score is **dot-product atten**tion.

For each encoder state j

$$\mathrm{score}(\mathrm{h}^d_{i-1}, \mathrm{h}^e_j) = \mathrm{h}^d_{i-1} \cdot \mathrm{h}^e_j$$

How to compute c_i ? How to decide what

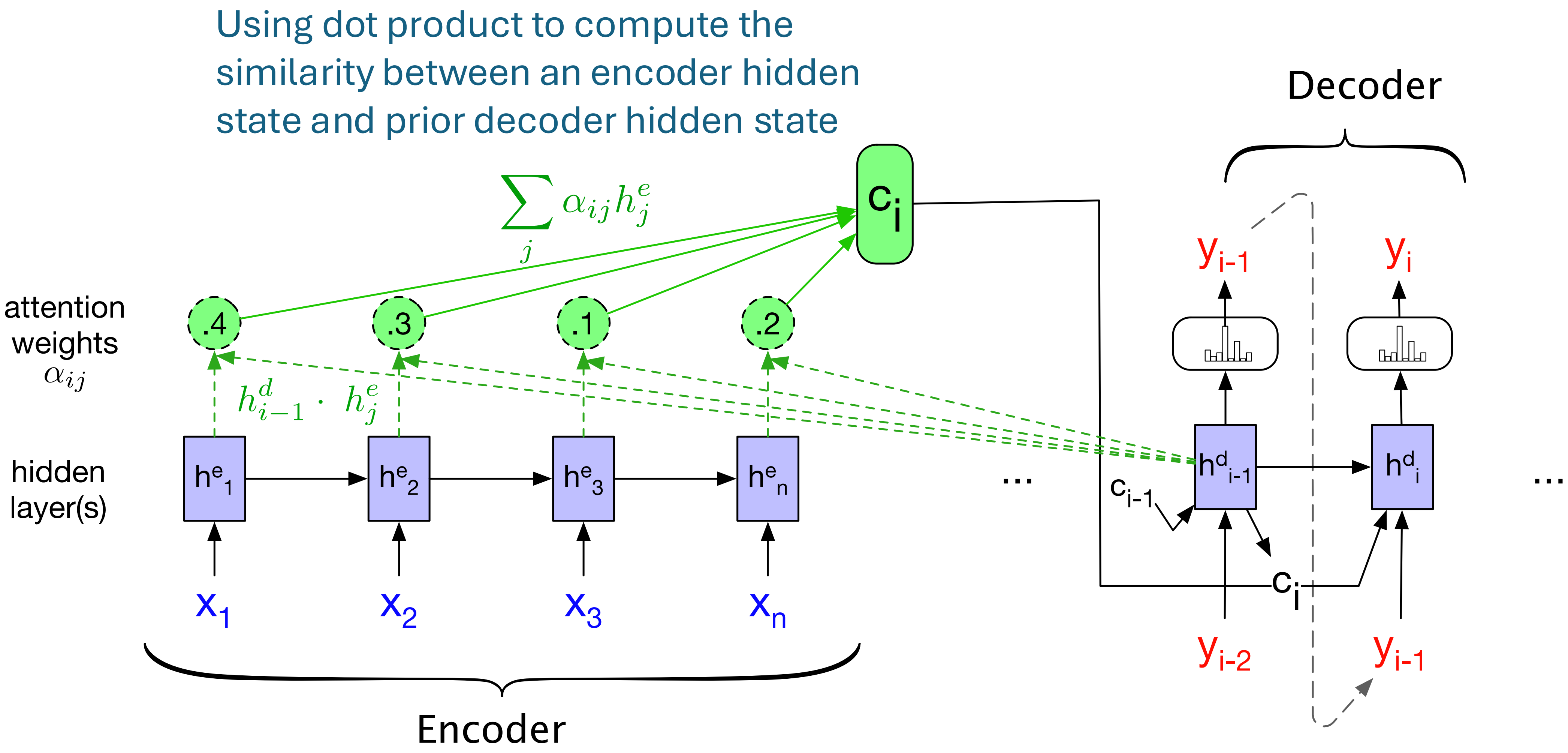
- We'll normalize these similarity scores of each encoder hidden states with a softmax to create weights $\alpha_{i\,j}$, that tell us the relevance of encoder hidden state *j* to hidden decoder state, h^d_{i-1}

$$a_{ij} = \text{softmax}(\text{score}(h^d_{i-1}, h^e_j))$$

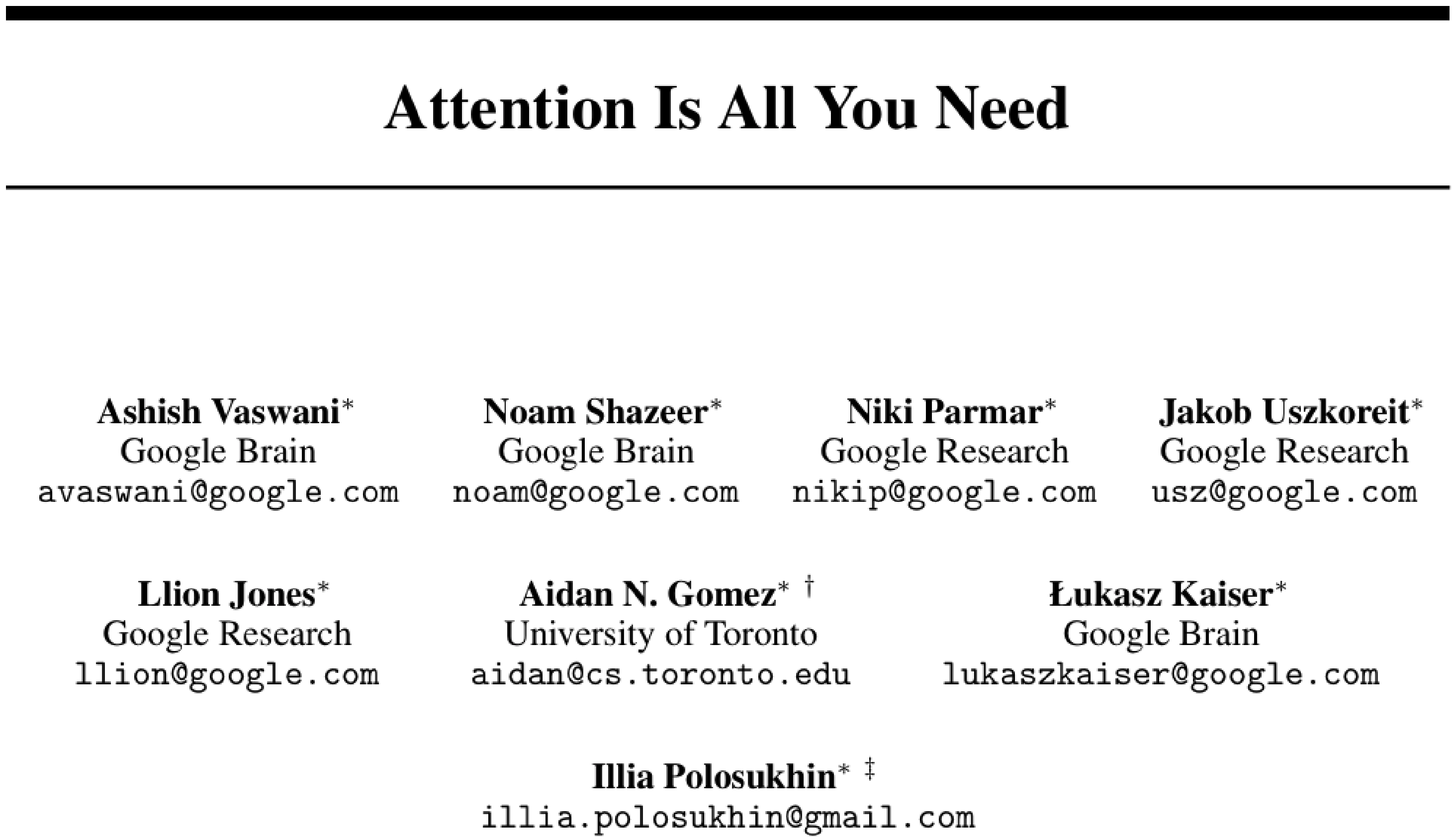
- And then use this to help create a weighted average of all the encoder hidden states:

$$\mathbf{c}_i = \sum_j \alpha_{i\,j}\,\mathbf{h}^e_j$$

Encoder-decoder with attention, focusing on the computation of c



Now, enter



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

At the very beginning of the paper abstract:

Exactly what we talked about in the last lecture

"The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism.

We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train."

Attention head

Recall the simple intuition of attention: begin by computing the similarity between current word and all previous words using their dot product. The similarity will weight the importance of that previous word to the current word.

The chicken didn’t cross the road because **it**

x_i

Compute similarity with each previous word:

$$\text{score}(x_i, x_j) = x_i \cdot x_j$$

α_{ij} is the similarity between current word and a previous word at position j, it will become the weight that signifies its importance/relatedness → how much should we attend to this

$$a_{ij} = \text{softmax}(\text{score}(x_i, x_j)) \;\; 8j \leq i$$

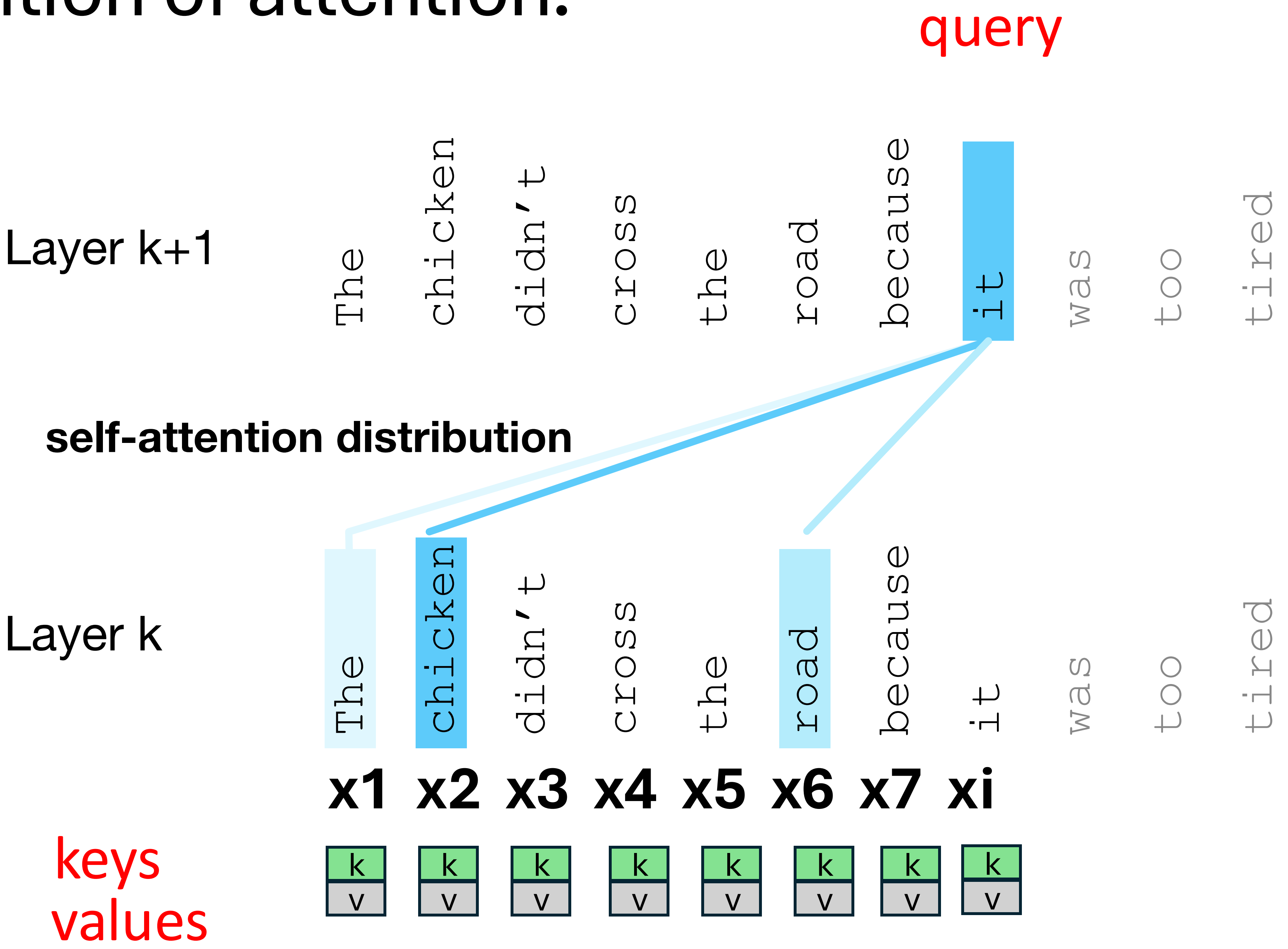
$$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$$

Weight sum

Attention head

- In transformer, we have a slightly more complicated version of this
  - Attention head
    - "Head" in transformers: refer to specific structured layers
- A single attention head use **query, key, value** matrices.
  - 3 different roles each vector x_i can play
    - Intuitively: current word x_i (query), and key and value of preceding word x_j
  - Query: current element
  - Key: a preceding input that's being compared to (for similarity)
  - Value: value of a preceding element (that gets weighted and summed), the actual information, semantic contribution

Intuition of attention:


query
Layer k+1
The chicken didn't cross the road because it was too tired
self-attention distribution
Layer k
The chicken didn't cross the road because it was too tired
x1 x2 x3 x4 x5 x6 x7 xi
keys
values
k k k k k k k k
v v v v v v v v

Single-head attention

- A single attention head uses these three matrices:
  - Query W^Q
  - Key W^K
  - Value W^V

We'll use matrices to project each vector $\mathbf{x}_i$ into a representation of its role as query, key, value:

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

An Actual Attention Head: slightly more complicated

- Given these 3 representation of $\mathbf{x}_i$

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

- To compute similarity of current element $\mathbf{x}_i$ with some prior element $\mathbf{x}_j$
- We’ll use dot product between $\mathbf{q}_i$ and $\mathbf{k}_j$.
- And instead of summing up $\mathbf{x}_j$, we'll sum up $\mathbf{v}_j$

Final equations for one attention head

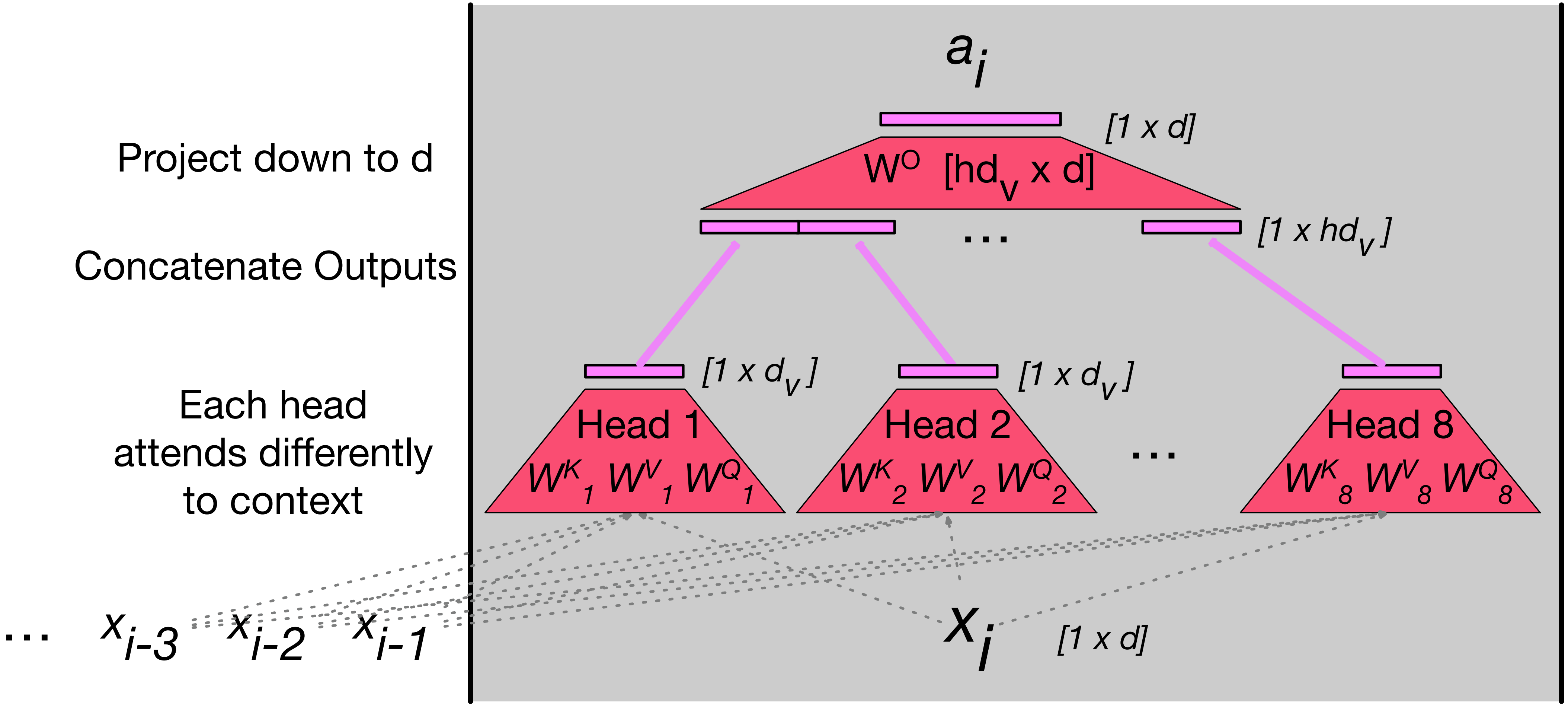
$$
\begin{aligned}
\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^{\mathbf{Q}};\quad \mathbf{k}_j &= \mathbf{x}_j \mathbf{W}^{\mathbf{K}};\quad \mathbf{v}_j = \mathbf{x}_j \mathbf{W}^{\mathbf{V}} \\
\text{score}(\mathbf{x}_i, \mathbf{x}_j) &= \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}} \\
\alpha_{ij} &= \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \;\; \forall j \leq i \\
\mathbf{head}_i &= \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j \\
\mathbf{a}_i &= \mathbf{head}_i \mathbf{W}^{\mathbf{O}}
\end{aligned}
$$

Actual Attention: slightly more complicated

- Instead of one attention head, we'll have lots of them!
- Intuition: each head might be attending to the context for different purposes
  - Different linguistic relationships or patterns in the context

$$
\begin{aligned}
\mathbf{q}_i^c = \mathbf{x}_i\mathbf{W}^{\mathbf{Qc}};\quad \mathbf{k}_j^c = \mathbf{x}_j\mathbf{W}^{\mathbf{Kc}};\quad \mathbf{v}_j^c &= \mathbf{x}_j\mathbf{W}^{\mathbf{Vc}};\quad \forall\, c \quad 1 \le c \le h \\
\text{score}^c(\mathbf{x}_i,\mathbf{x}_j) &= \frac{\mathbf{q}_i^c \cdot \mathbf{k}_j^c}{\sqrt{d_k}} \\
\alpha_{ij}^c &= \text{softmax}(\text{score}^c(\mathbf{x}_i,\mathbf{x}_j)) \;\; \forall j \le i \\
\mathbf{head}_i^c &= \sum_{j \le i} \alpha_{ij}^c \mathbf{v}_j^c \\
\boxed{\mathbf{a}_i = (\mathbf{head}^1 \oplus \mathbf{head}^2 \ldots \oplus \mathbf{head}^h)\mathbf{W}^O} \\
\text{MultiHeadAttention}(\mathbf{x}_i,[\mathbf{x}_1,\cdots,\mathbf{x}_N]) &= \mathbf{a}_i
\end{aligned}
$$

Multi-head attention

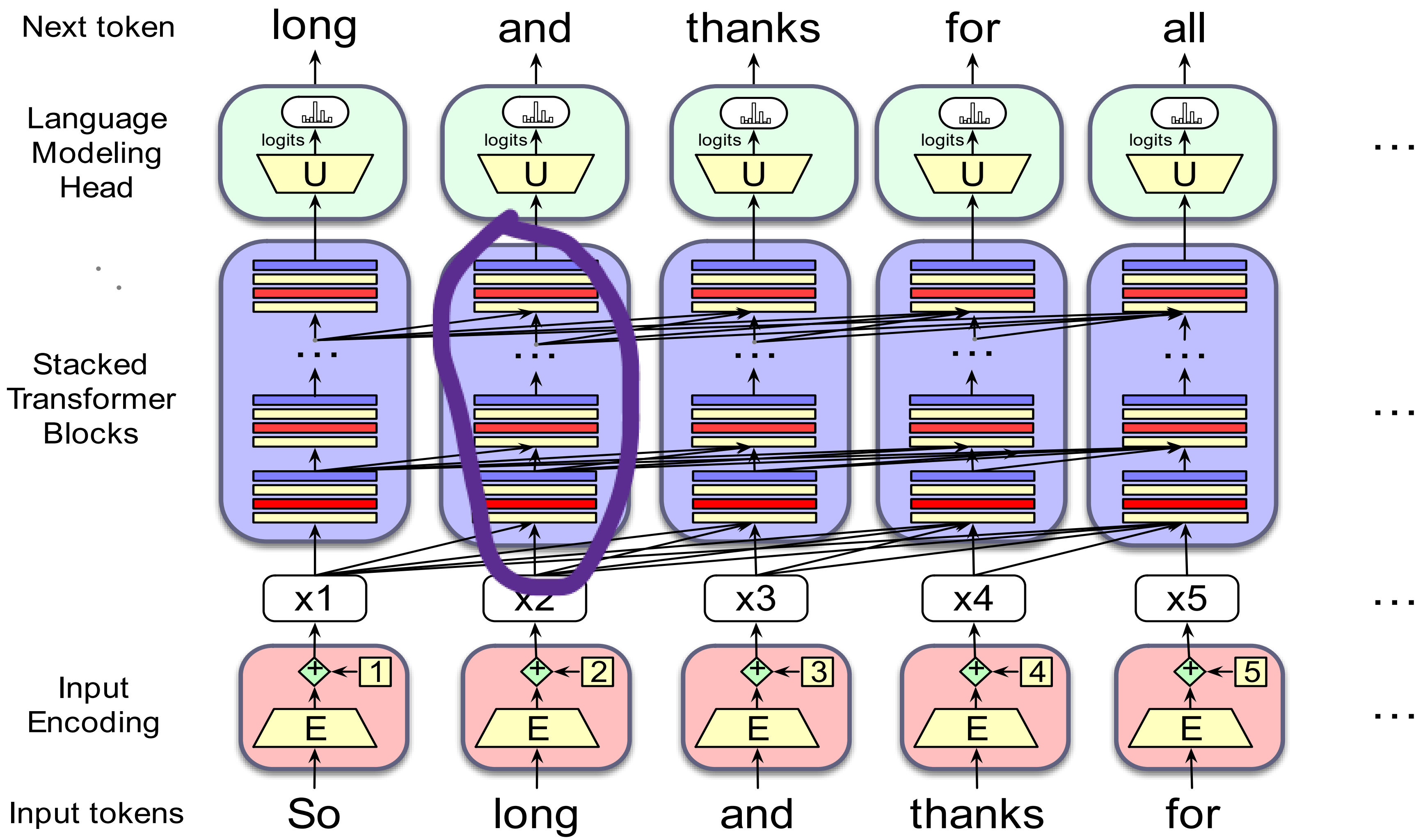


Summary

- Attention is a method for enriching the representation of a token by incorporating contextual information
- The result: the embedding for each word will be different in different contexts!
- Contextual embeddings: a representation of word meaning in its context.

Transformer blocks

Transformer language model



In a transformer block

- In addition to the self-attention layer, there are 3 other layers: feedforward layer, residual connections, and a normalizing layer (aka. Layer norm)
- And we can stack these transformer blocks, but each block is consists of these things.

- One way of thinking about the block is called the **residual stream** (Elhage et al, 2021).

The residual stream

- Each individual token has their own stream for processing
- A single stream starts with the original input vector, then branch out to different components (layer norm, MHA, feedforward), but we add their output back into the stream
- Initial embedding of a token $\boldsymbol{x_i}$ at position i is of dimensionality d.
- The final output of the transformer block (end of this stream) for token i is $\boldsymbol{h_i}$

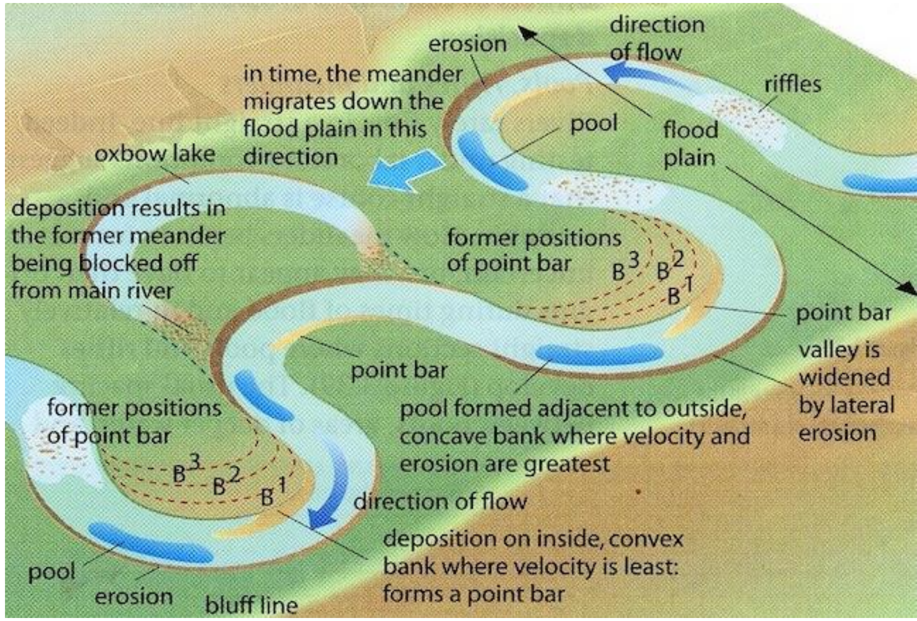
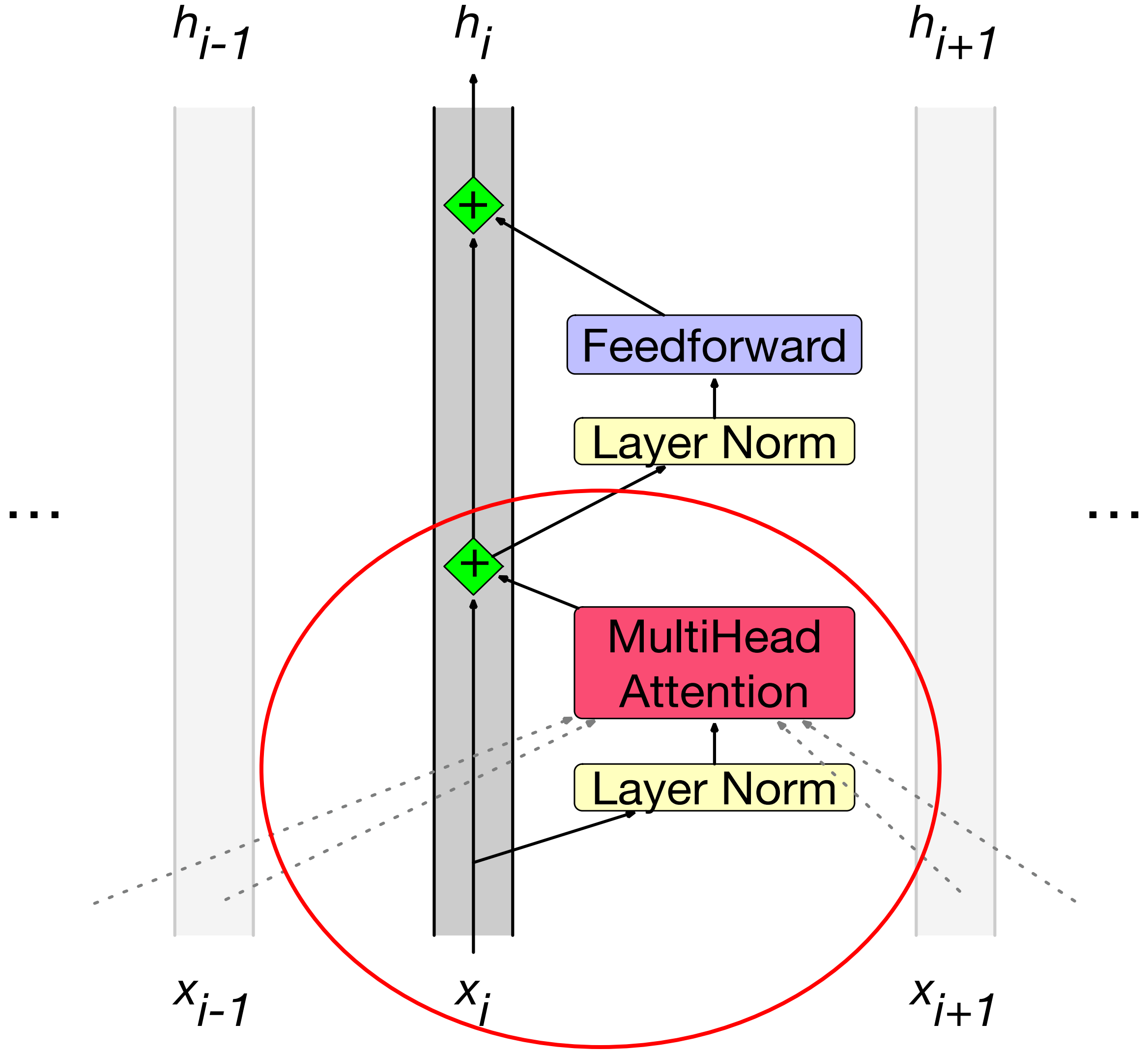

erosion
direction of flow
in time, the meander migrates down the flood plain in this direction
riffles
pool
flood plain
oxbow lake
deposition results in the former meander being blocked off from main river
former positions of point bar
B^3
B^2
B^1
point bar
valley is widened by lateral erosion
point bar
pool formed adjacent to outside, concave bank where velocity and erosion are greatest
former positions of point bar
B^3
B^2
B^1
direction of flow
deposition on inside, convex bank where velocity is least: forms a point bar
pool
erosion
bluff line

Figure from https://www.ausableriver.org/blog/why-do-streams-meander

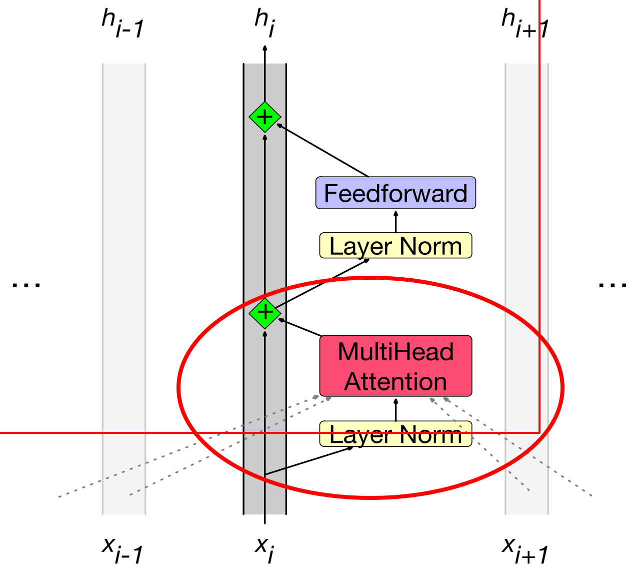
The residual stream: each token gets passed up and modified

There are many variations of transformers. Here's we are using the pre-norm version which is the most common today, instead of the OG transformer which is post-norm



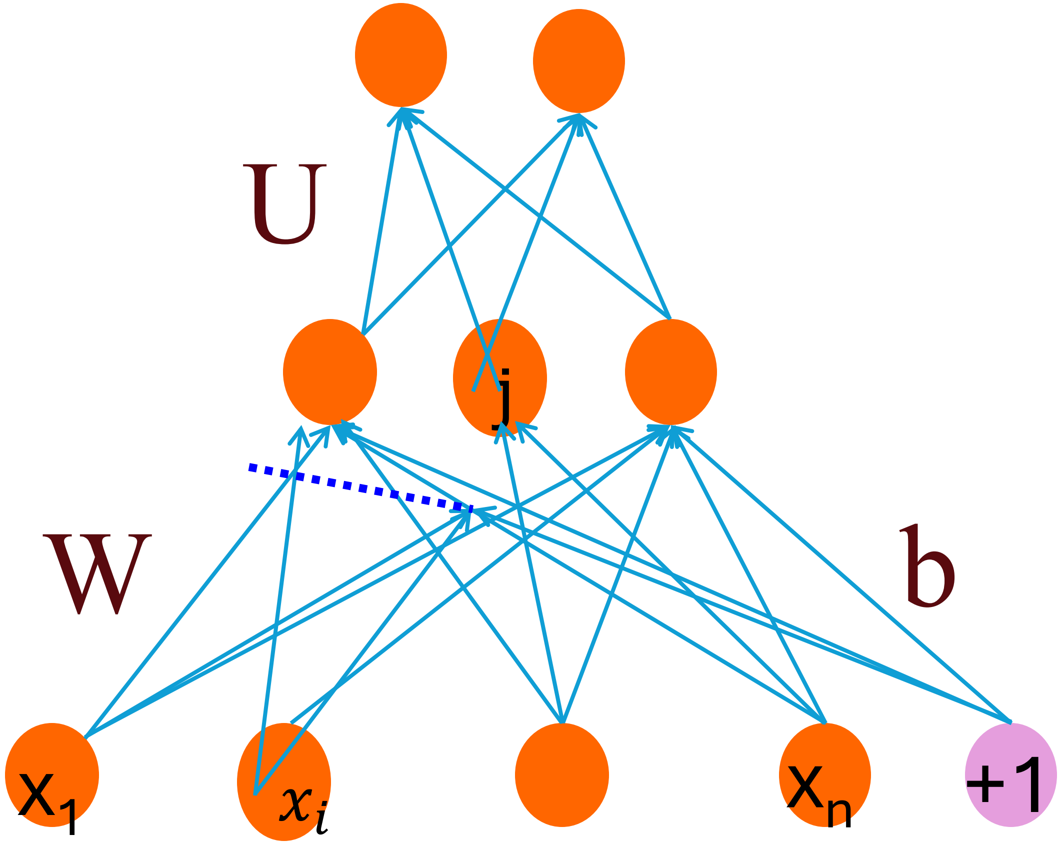
Minor clarification: Pre-norm vs post-norm

- Just different variations
- Post-norm: input → MHA → Residual Add → layer norm
  - OG transformer in their paper (Vaswani et al, 2017)
  - Takes raw hidden states of token, no layer norm yet
- Pre-norm: input → layer norm → MHA → residual add
  - GPT 2/3, some BERT, etc.
  - Take the layer normalized hidden states of tokens



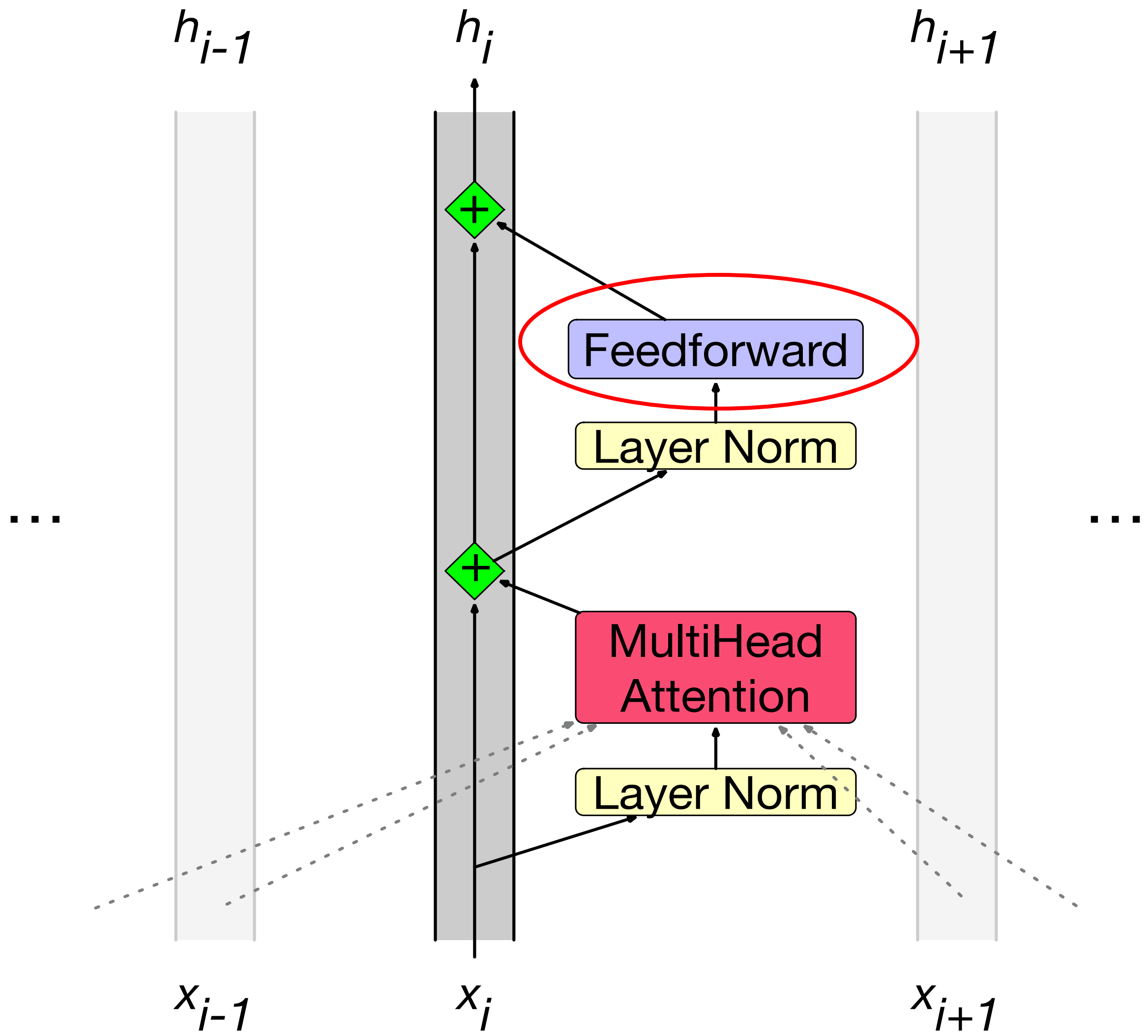
The feedforward layer in residual stream/ a transformer block

- It’s a fully-connected 2-layer network
  - 1 hidden layer
  - 2 weight matrices

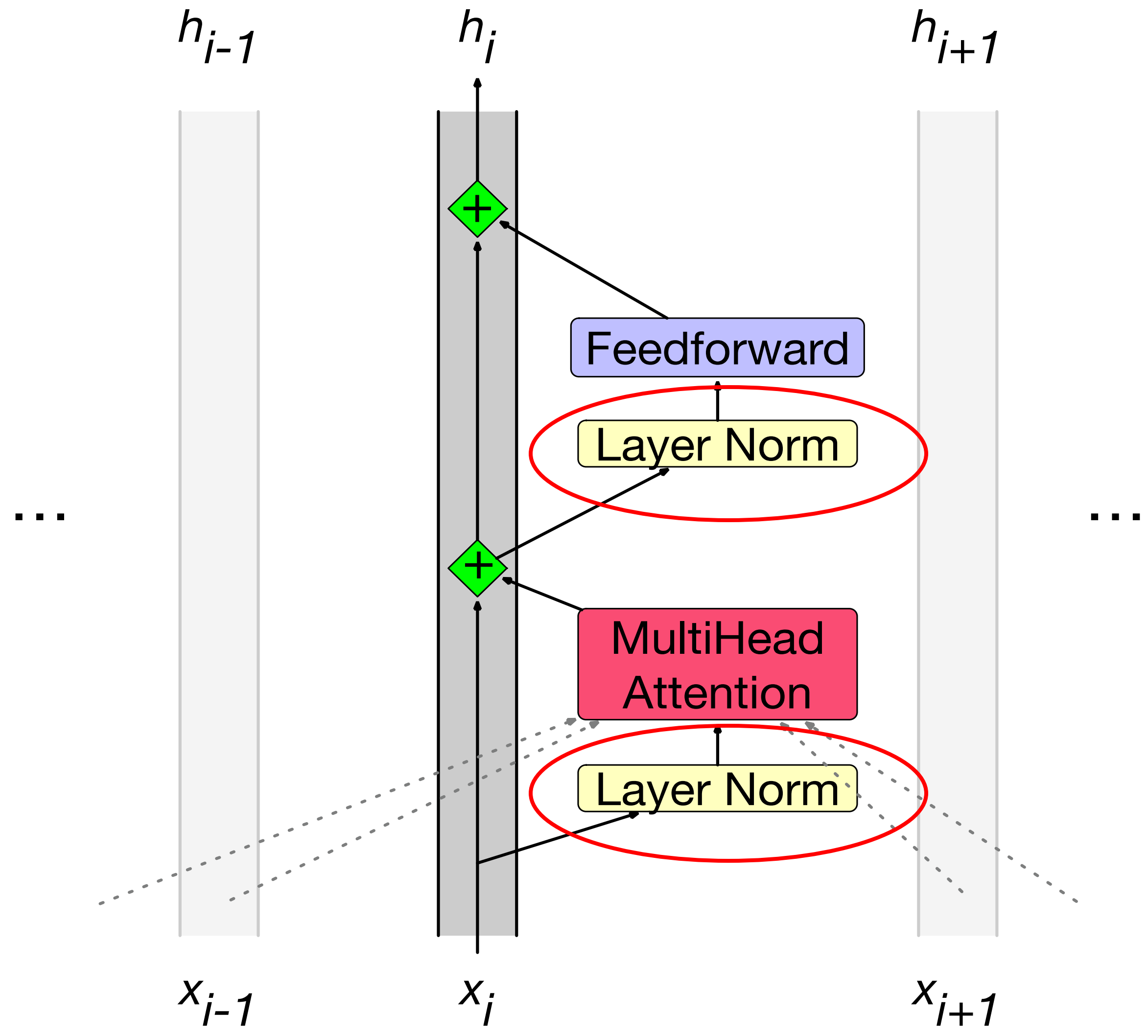


We'll need nonlinearities, so a feedforward layer

$$\mathrm{FFN}(\mathbf{x}_i) = \mathrm{ReLU}(\mathbf{x}_i\mathbf{W}_1 + b_1)\mathbf{W}_2 + b_2$$



Layer norm: the vector $\mathbf{x}_i$ is normalized twice



Layer Norm

Layer norm is a variation of the z-score from statistics, applied to a single vector in a hidden layer

$$\mu = \frac{1}{d}\sum_{i=1}^{d} x_i$$

$$\sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d} (x_i - \mu)^2}$$

mean

$$\hat{\mathbf{x}} = \frac{(\mathbf{x} - \mu)}{\sigma}$$

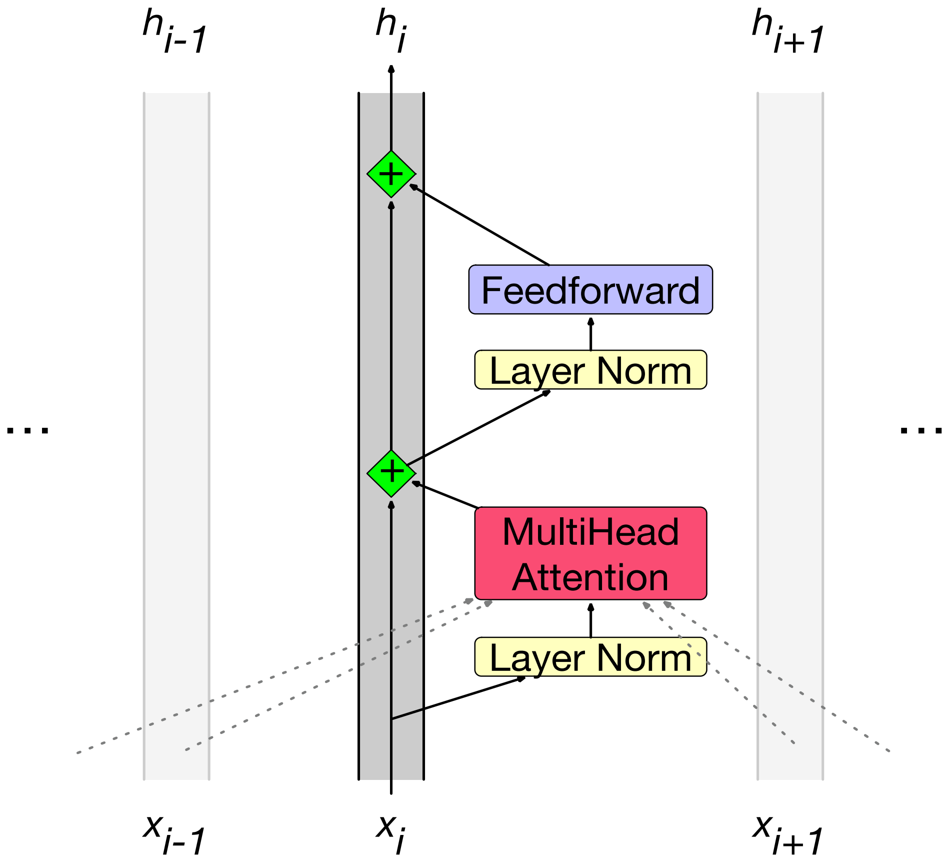
Standard deviation

$$\text{LayerNorm}(\mathbf{x}) = \gamma \frac{(\mathbf{x} - \mu)}{\sigma} + \beta$$

One of many form of normalization that can help improve training performance in deep neural network.

Input and output of layer norm are both of dimensionality d

Putting together a single transformer block

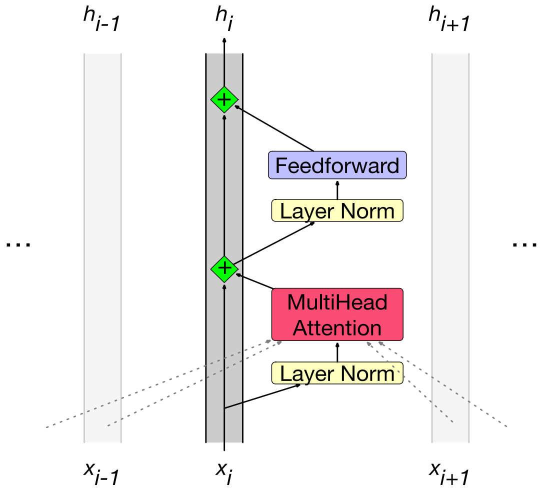


$$
\begin{aligned}
\mathbf{t}_i^1 &= \mathrm{LayerNorm}(\mathbf{x}_i) \\
\mathbf{t}_i^2 &= \mathrm{MultiHeadAttention}(\mathbf{t}_i^1, [\mathbf{x}_1^1, \cdots, \mathbf{x}_N^1]) \\
\mathbf{t}_i^3 &= \mathbf{t}_i^2 + \mathbf{x}_i \\
\mathbf{t}_i^4 &= \mathrm{LayerNorm}(\mathbf{t}_i^3) \\
\mathbf{t}_i^5 &= \mathrm{FFN}(\mathbf{t}_i^4) \\
\mathbf{h}_i &= \mathbf{t}_i^5 + \mathbf{t}_i^3
\end{aligned}
$$

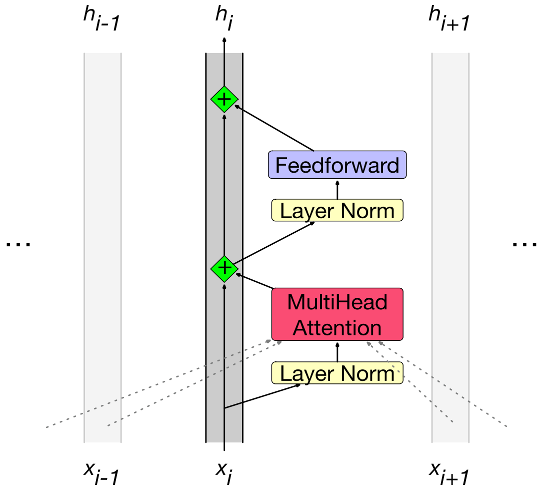
MHA is the only component that takes input from other tokens

A transformer is a stack of these blocks so all the vectors are of the same dimensionality d

Block 2

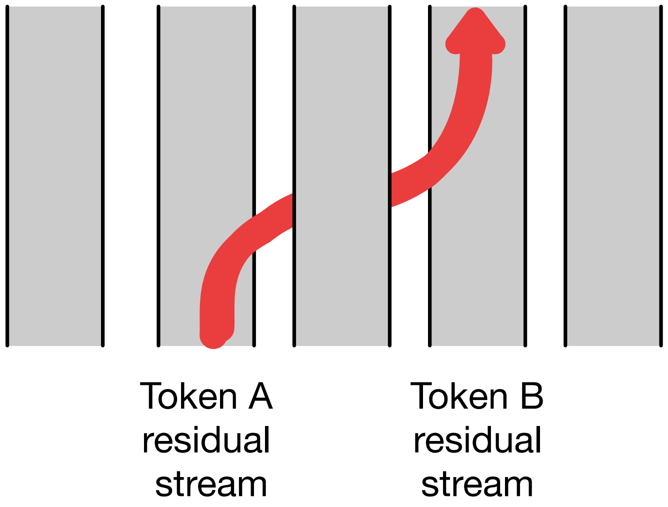


Block 1



Residual streams and attention

- Notice that all parts of the transformer block apply to 1 same token.
- Except multi-head attention component, which takes information from other tokens as well. (because we need context!)
- Elhage et al. (2021) show that we can view attention heads as literally moving information from the residual stream of a neighboring token into the current stream .



LLMs and transformer blocks

- In large language models, there are usually stacks of these transformer blocks
    - From 12 layers: T5, GPT-3-small
    - To 96 layers: GPT-3-large
    - Even more in recent models
- Once we stack many transformer blocks, at the very end of the last transformer block, there's a single extra layer norm after the last $\boldsymbol{h_i}$ (output of that block)

Parallelizing attention

Parallelizing attention

- The computation of a token's $\boldsymbol{a_i}$ is independent of the computation of other token's
- So are all the computations in a transformer block
- That means we can easily parallelize this entire computation and take advantage of the efficient matrix multiplication routines.

Input embeddings for N tokens

- Each row is a token embedding of ***d*** dimensions.
- We have N tokens
- So a matrix ***X*** with *N* tokens is of size *[N x d]* (row x col)

- So a matrix with 32k tokens (in other words, input length 32k) is of size *[32k x d]*

Each weight matrices

- W^Q → model learn how to ask questions
- W^K → model learn how to match relevant information
- W^V → model learn what content to pass on once we matched the keys

- These are trainable matrices. Without them we are just attending to token based on their raw embedding similarity
- You can project the X to lower-dimension space d_k which is cheaper

Let’s start with a single attention head

- We have input matrix X, and we multiply X by query, key, and value matrices

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{\mathbf{Q}};\ \ \mathbf{K} = \mathbf{X}\mathbf{W}^{\mathbf{K}};\ \ \mathbf{V} = \mathbf{X}\mathbf{W}^{\mathbf{V}}$$

QK^T

- Now can do a single matrix multiplication to combine Q and K^T

N

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

Parallelizing attention

- Scale the scores, take the softmax, and then multiply the result by V resulting in a matrix of shape *N* × *d*
  - An attention vector for each input token

$$\mathbf{A} = \text{softmax}\left(\text{mask}\left(\frac{\mathbf{QK}^\top}{\sqrt{d_k}}\right)\right)\mathbf{V}$$

Masking out the future

$$\mathbf{A} = \mathrm{softmax}\left(\mathrm{mask}\left(\frac{\mathbf{QK}^{\mathsf{T}}}{\sqrt{d_k}}\right)\right)\mathbf{V}$$

- What is this mask function?
  QK$^{\mathsf{T}}$ has a score for each query dot every key, *including those that follow the query*.
- Guessing the next word is pretty simple if you already know it!

Masking out the future

$$\mathbf{A} = \mathrm{softmax}\left(\mathrm{mask}\left(\frac{\mathbf{QK}^{\intercal}}{\sqrt{d_k}}\right)\right)\mathbf{V}$$

- set $-\infty$ to cells in upper triangle
- The softmax will turn it to 0

N

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

N

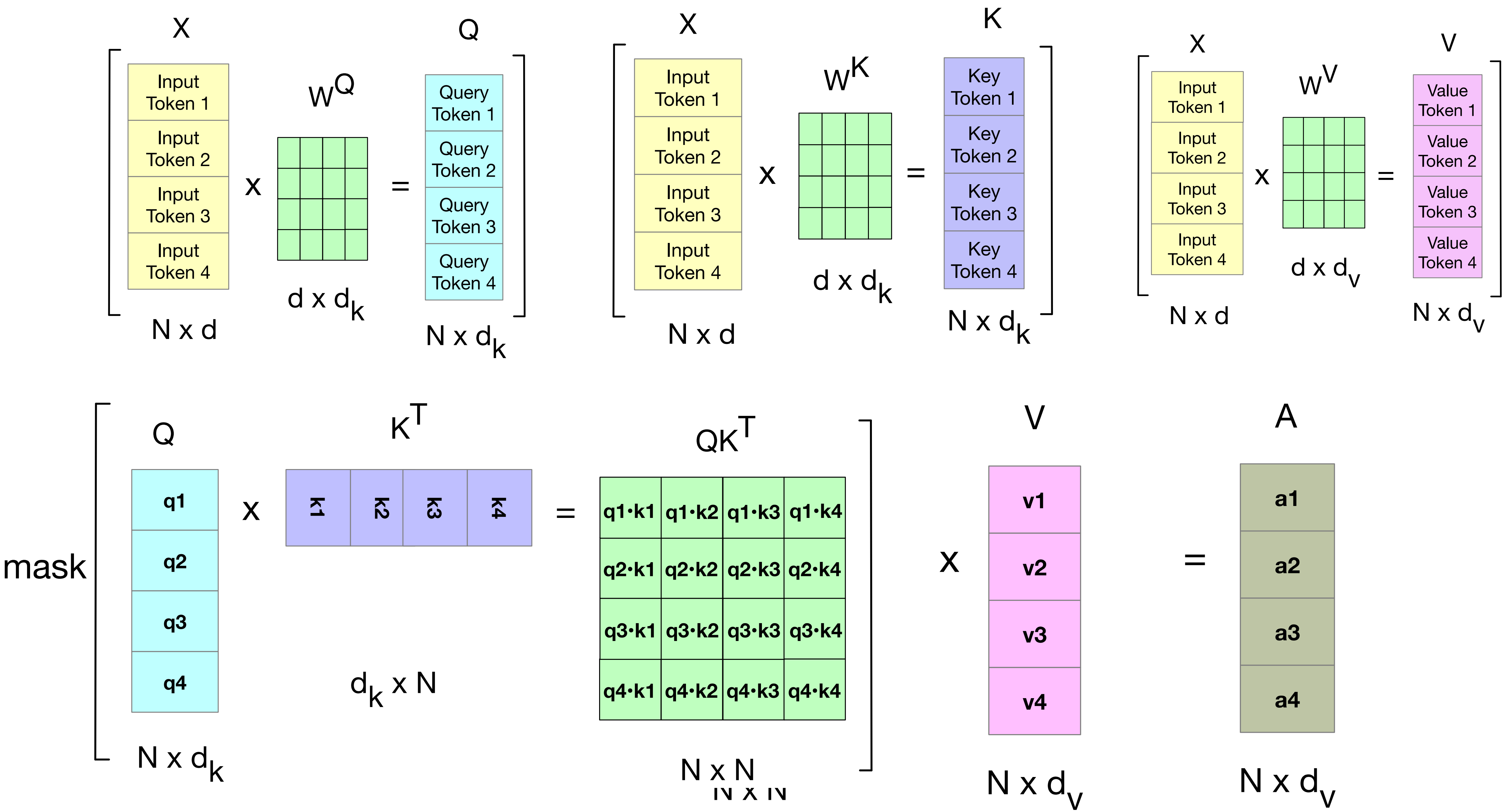
Another point: Attention is quadratic in length

$$\mathbf{A} \;=\; \mathrm{softmax}\left(\mathrm{mask}\left(\frac{\mathbf{QK}^{\top}}{\sqrt{d_k}}\right)\right)\mathbf{V}$$

dot products between each pair of tokens in the input.

$$\begin{array}{cc}
N & \begin{array}{|c|c|c|c|}
\hline
\mathbf{q1{\cdot}k1} & -\infty & -\infty & -\infty \\
\hline
\mathbf{q2{\cdot}k1} & \mathbf{q2{\cdot}k2} & -\infty & -\infty \\
\hline
\mathbf{q3{\cdot}k1} & \mathbf{q3{\cdot}k2} & \mathbf{q3{\cdot}k3} & -\infty \\
\hline
\mathbf{q4{\cdot}k1} & \mathbf{q4{\cdot}k2} & \mathbf{q4{\cdot}k3} & \mathbf{q4{\cdot}k4} \\
\hline
\end{array} \\
 & N
\end{array}$$

Single-head attention with N inputs



Parallelizing Multi-head Attention

Each attention head performs attention independently, and we concatenate them at the end, and multiply by W^o

$$\mathbf{Q^i} = \mathbf{XW^{Qi}} \; ; \quad \mathbf{K^i} \; = \; \mathbf{XW^{Ki}} \; ; \quad \mathbf{V^i} = \mathbf{XW^{Vi}}$$

$$\mathbf{head}_i = \text{SelfAttention}(\mathbf{Q^i}, \mathbf{K^i}, \mathbf{V^i}) \; = \; \text{softmax}\left(\frac{\mathbf{Q^i K^{i\top}}}{\sqrt{d_k}}\right)\mathbf{V^i}$$

$$\text{MultiHeadAttention}(\mathbf{X}) \; = \; (\mathbf{head}_1 \oplus \mathbf{head}_2 ... \oplus \mathbf{head}_h)\mathbf{W^O}$$

Attention heads can be parallelized, but not the stacked blocks, these are sequential.

Parallelizing Multi-head Attention

$$\begin{aligned}\mathbf{O} &= \mathrm{LayerNorm}(\mathbf{X} + \mathrm{MultiHeadAttention}(\mathbf{X}))\\ \mathbf{H} &= \mathrm{LayerNorm}(\mathbf{O} + \mathrm{FFN}(\mathbf{O}))\end{aligned}$$

- or

$$\begin{aligned}\mathbf{T}^1 &= \mathrm{MultiHeadAttention}(\mathbf{X})\\ \mathbf{T}^2 &= \mathbf{X} + \mathbf{T}^1\\ \mathbf{T}^3 &= \mathrm{LayerNorm}(\mathbf{T}^2)\\ \mathbf{T}^4 &= \mathrm{FFN}(\mathbf{T}^3)\\ \mathbf{T}^5 &= \mathbf{T}^4 + \mathbf{T}^3\\ \mathbf{H} &= \mathrm{LayerNorm}(\mathbf{T}^5)\end{aligned}$$

Input embeddings

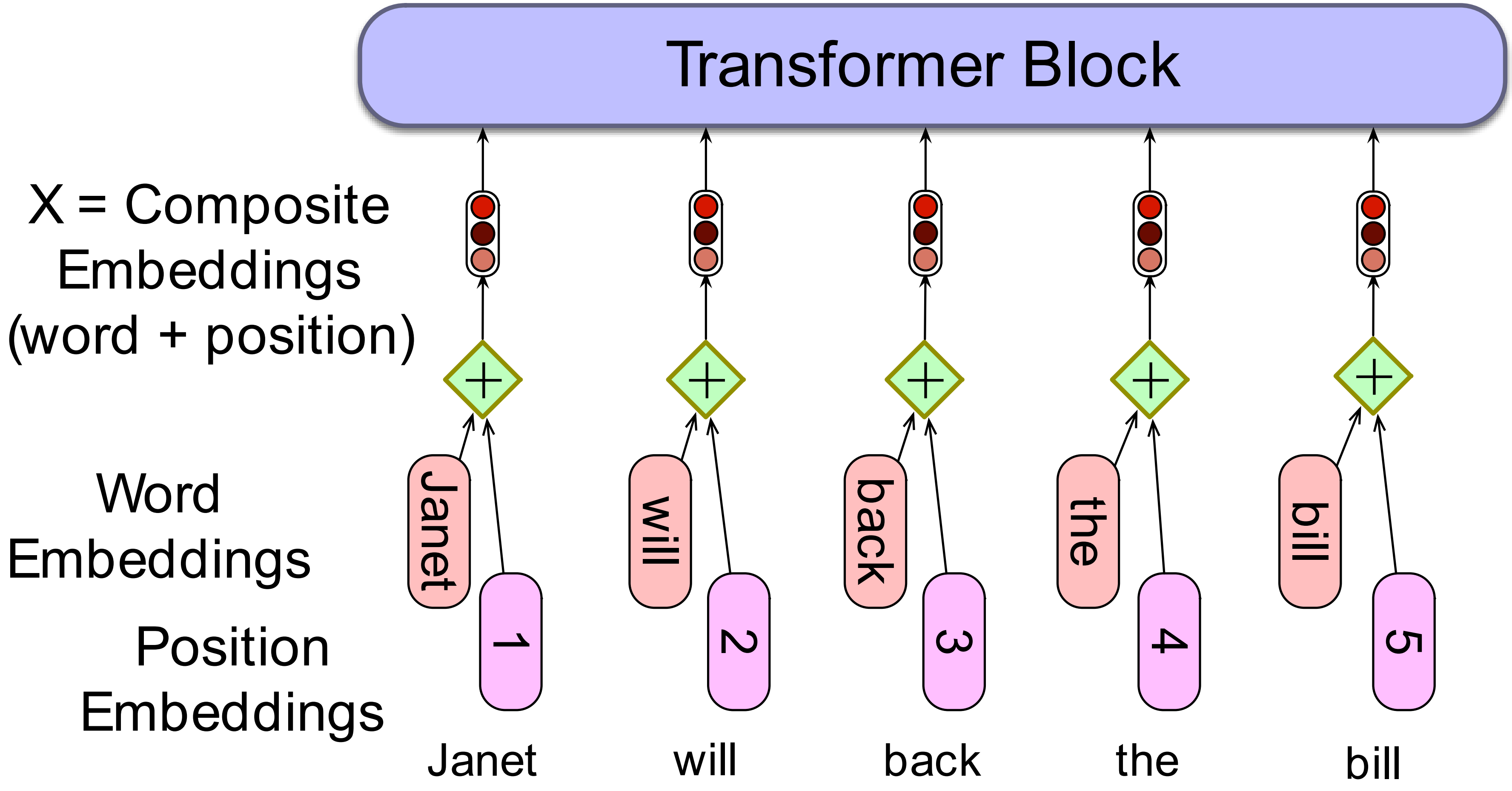
Token and Position Embeddings

- The matrix X (of shape [$N \times d$]) has an embedding for each word in the context.
- This embedding is created by adding two distinct embedding for each input
- token embedding
- positional embedding

Position Embeddings

- There are many methods, but we'll just describe the simplest: absolute position.
- Goal: learn a position embedding matrix *E*pos of shape [1 × *N*].
- Start with randomly initialized embeddings
- one for each integer up to some maximum length.
- i.e., just as we have an embedding for token *fish*, we’ll have an embedding for position 3 and position 17.
- As with word embeddings, these position embeddings are learned along with other parameters during training.

Each **x** is just the sum of word and position embeddings



Pointers

- We will continue this in the next lecture.
- If you haven’t read it in your deep learning class, the original paper is worth reading! Though it is not a required reading for this class.
- Once again, the textbook is wonderful if you want to slowly go over any concepts with more details