The Transformer

NLP Week 8

Thanks to Dan Jurafsky for most of the slides this week!

Plan for today

1. Review of notation and matrix multiplication
2. (Multi-head) self-attention
3. Residual stream
4. Position embeddings
5. Putting it all together —> The Transformer
6. GPT and BERT
7. *Group exercises*

This semester

We will build language models adding to each layer of their complexity:

1. **Bag of words models** (basic statistical models of language)
2. **N-gram models** (+ sequential dependencies)
3. **Hidden Markov models** (+ latent categories)
4. **Recurrent neural networks** (+ distributed representations)
5. **LSTM language models** (+ long distance dependencies)
6. **Transformer language models** (+ attention-based dependency learning)

= Today’s language models!

A note on notation

Quick recap on our notation and matrix-matrix and matrix-vector multiplication

- Let $\mathbf{A}$ denote an $p \times d$ matrix
- Let $\mathbf{X}$ denote an $d \times n$ matrix
- Let $\mathbf{x}$ denote a $d \times 1$ vector (column vector)
- $\mathbf{x}^\top$ is a $1 \times d$ vector (row vector)
- Note that: $(\mathbf{AX})^\top = \mathbf{X}^\top \mathbf{A}^\top$

We need to understand:

- $\mathbf{y} = \mathbf{Ax}$
- $\mathbf{y}^\top = \mathbf{x}^\top \mathbf{A}^\top$
- $\mathbf{Y} = \mathbf{AX}$
- $\mathbf{Y}^\top = \mathbf{X}^\top \mathbf{A}^\top$

👉 will do this on the whiteboard …

The paper that started it all

Transformer: a specific kind of network architecture, like a fancier feedforward network, but based on attention

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

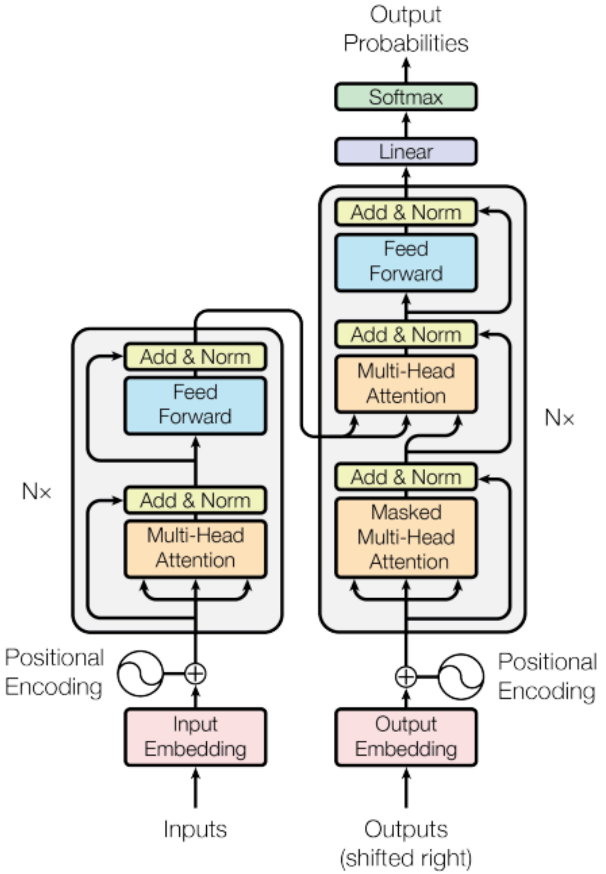
Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

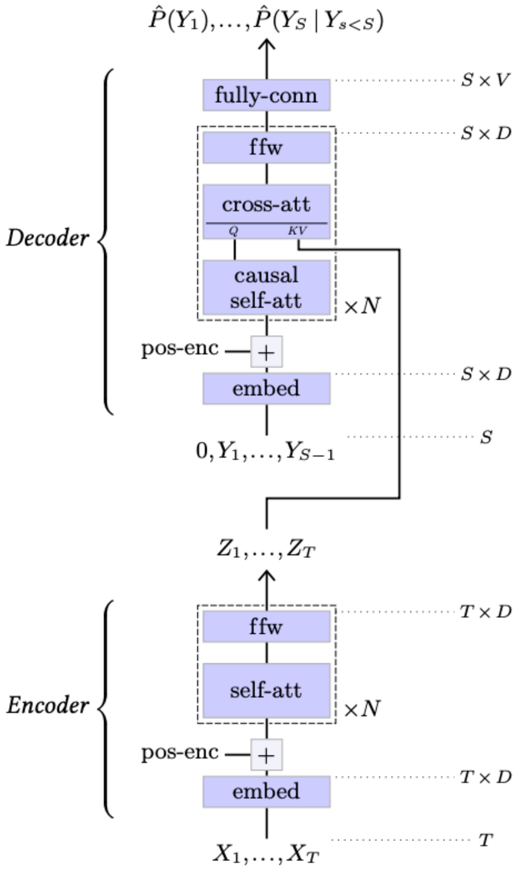
Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

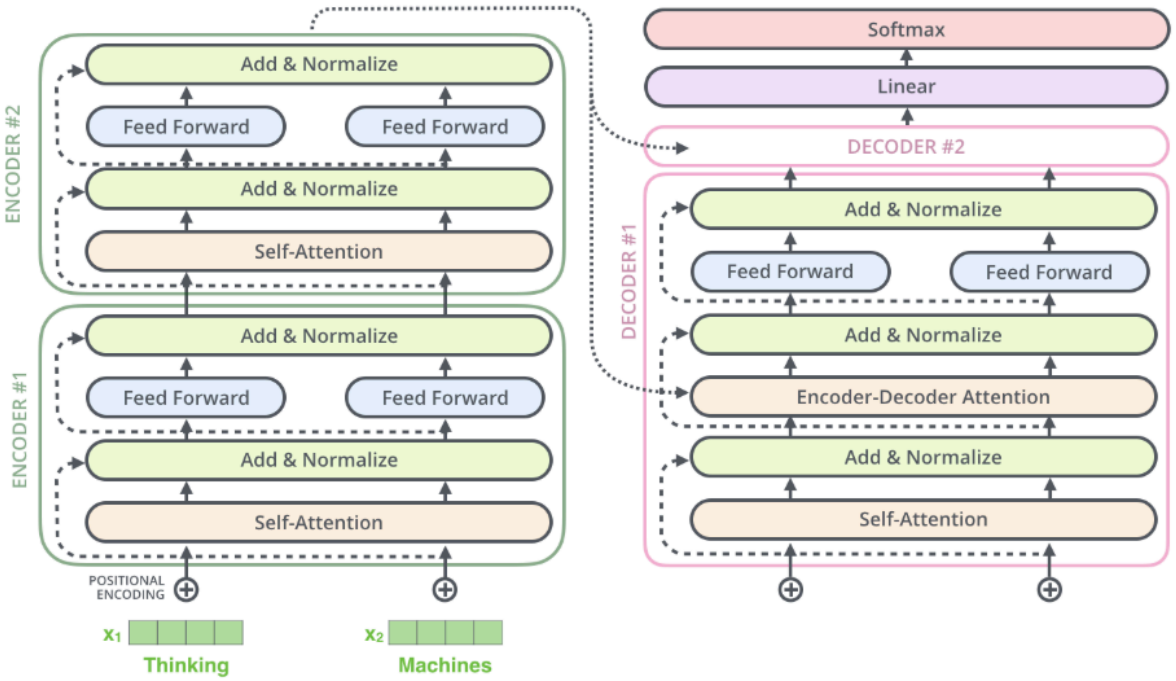
What is a Transformer?



Vaswani et al. 2017 - Attention is all you need

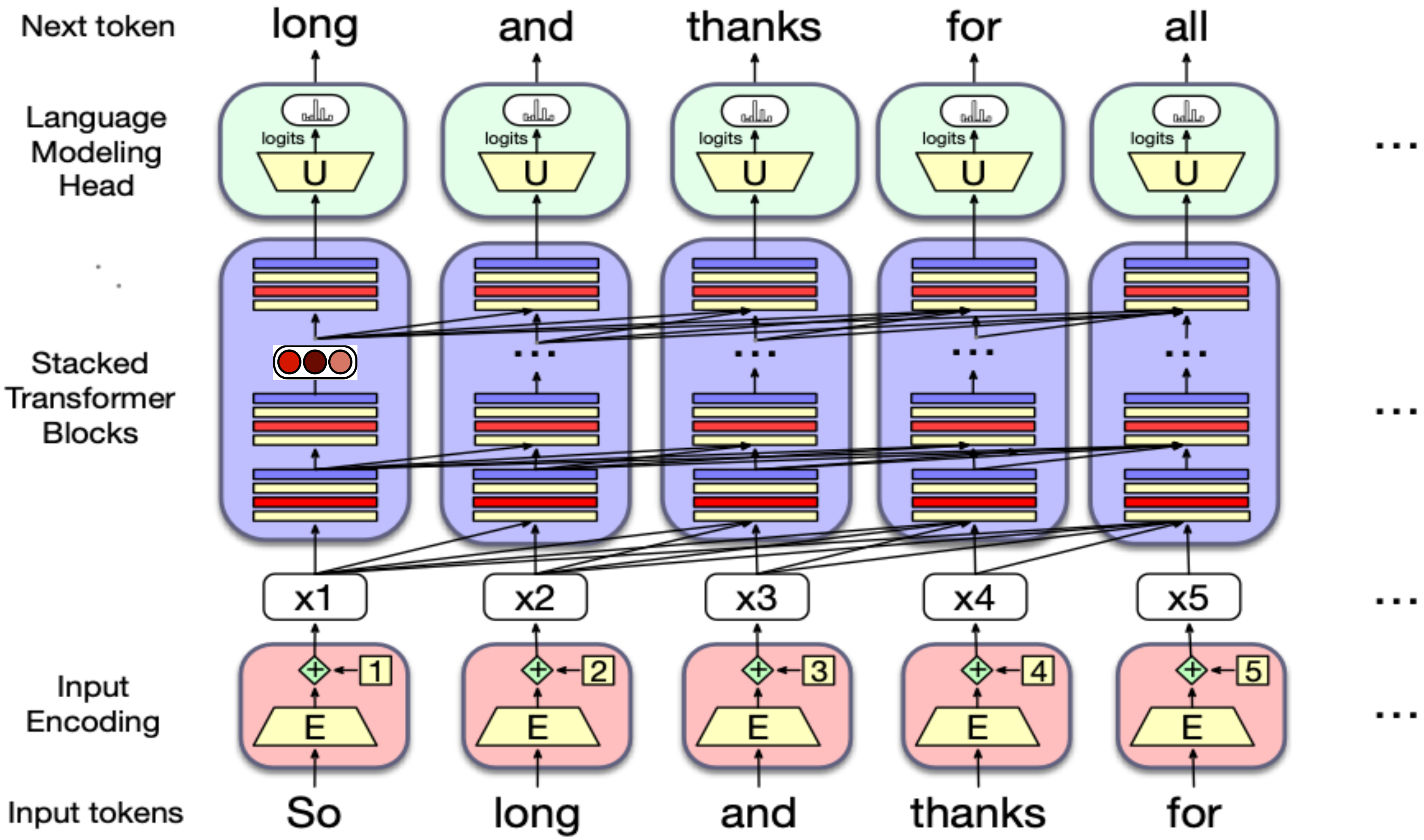


Fleuret et al. 2024 - The little book of DL

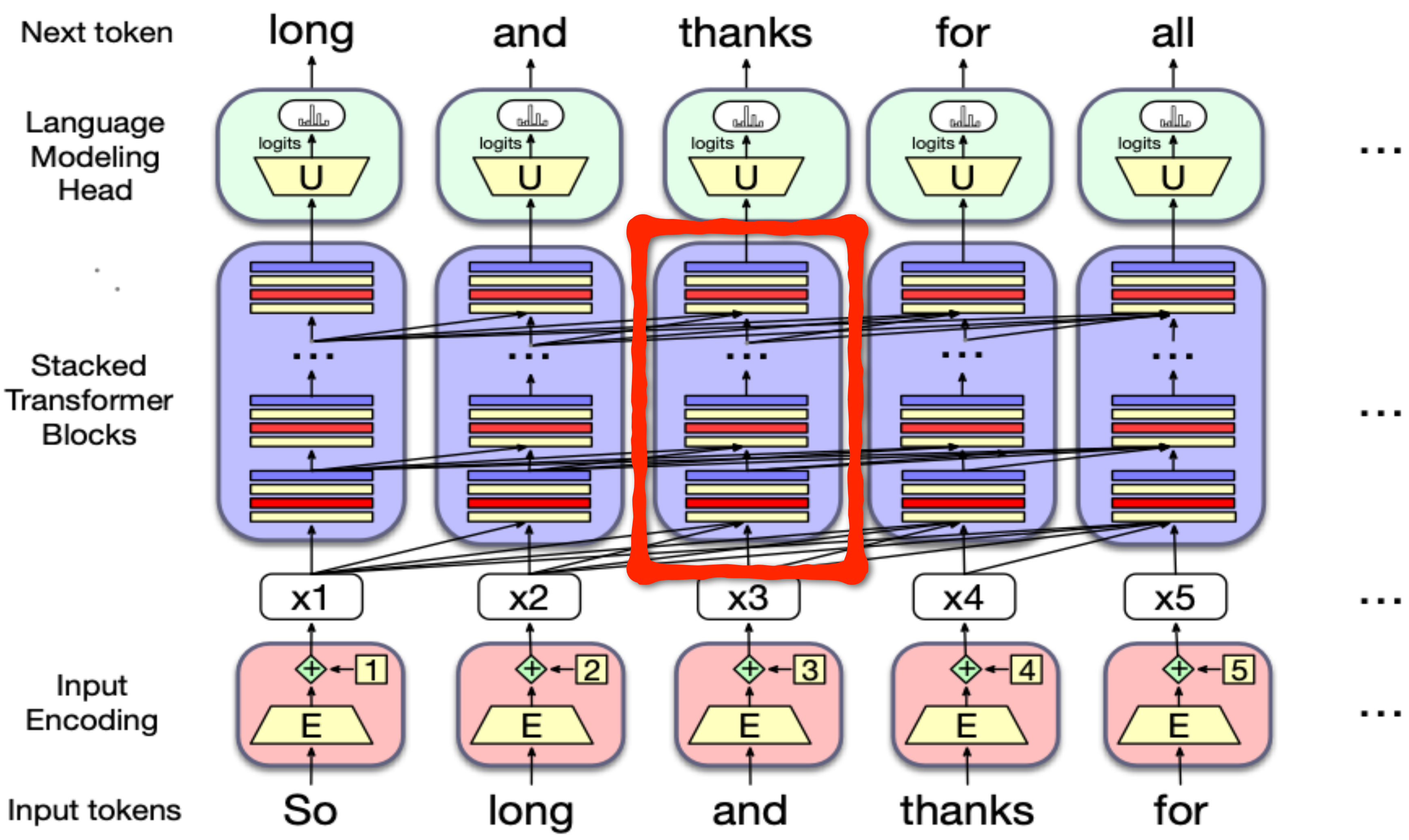


Alammar et al. 2018 - The Illustrated Transformer

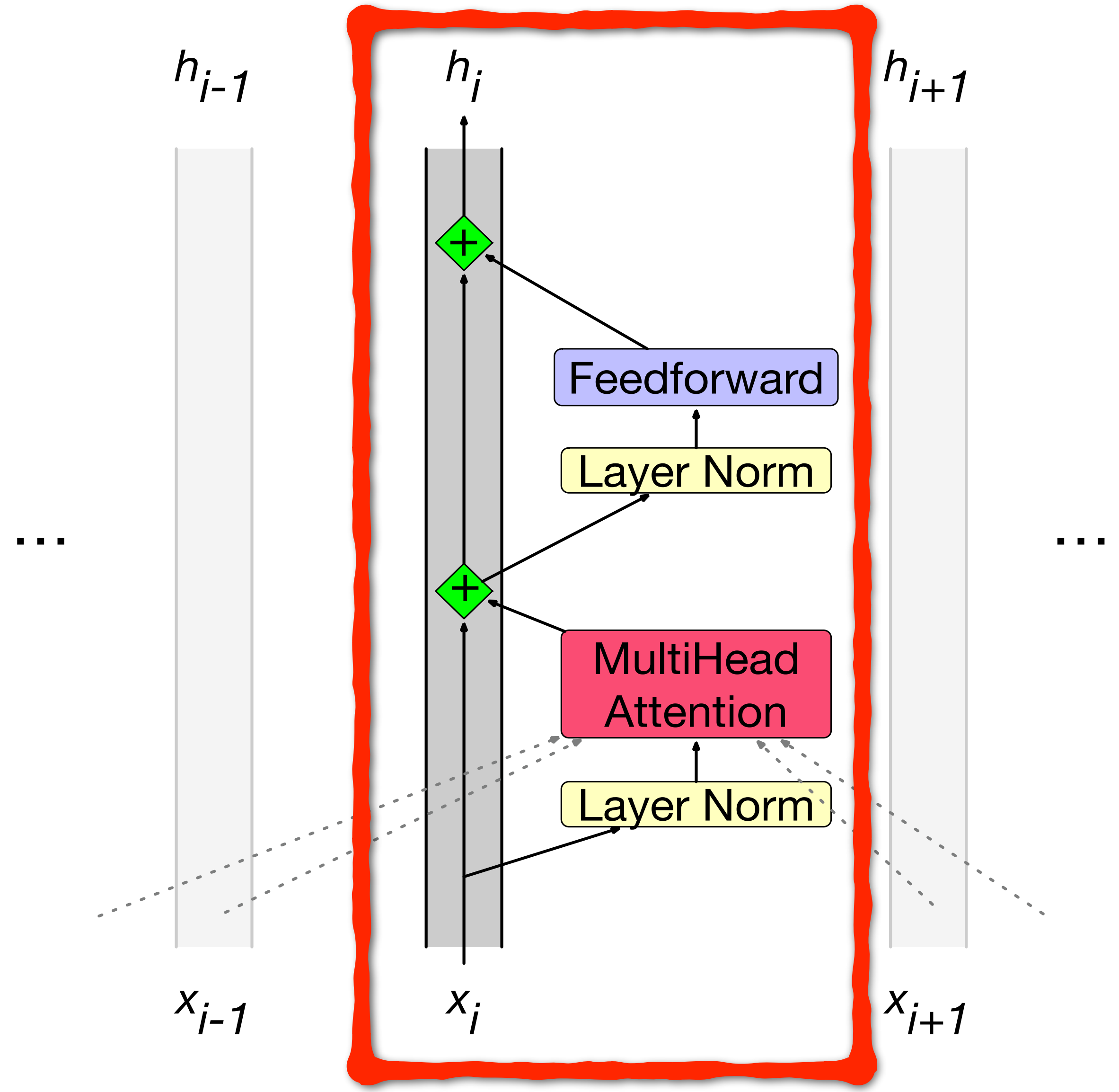
We will stick to the following illustration



The Transformer

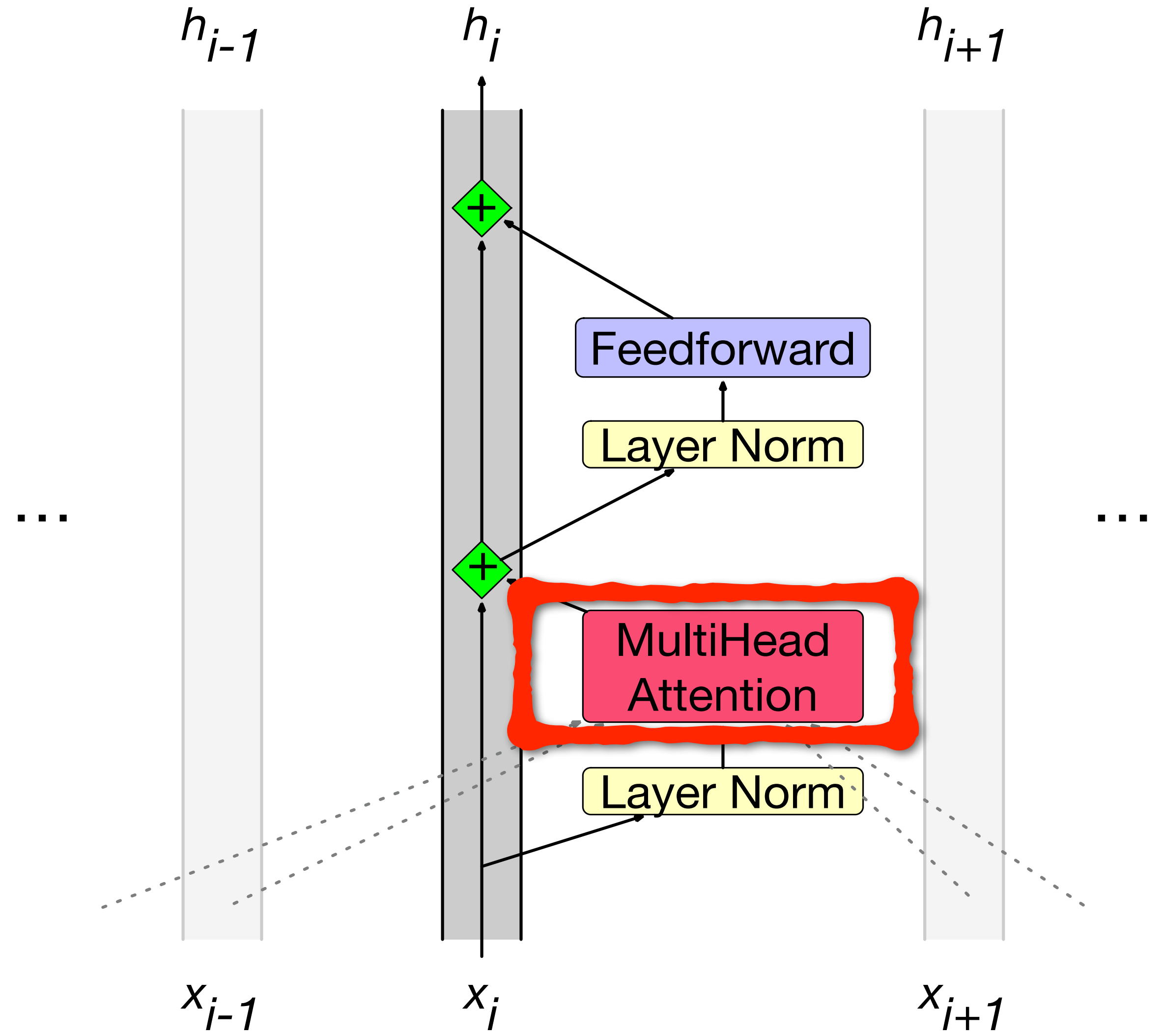


Zooming in



h_{i-1}
h_i
h_{i+1}
Feedforward
Layer Norm
...
...
MultiHead Attention
Layer Norm
x_{i-1}
x_i
x_{i+1}

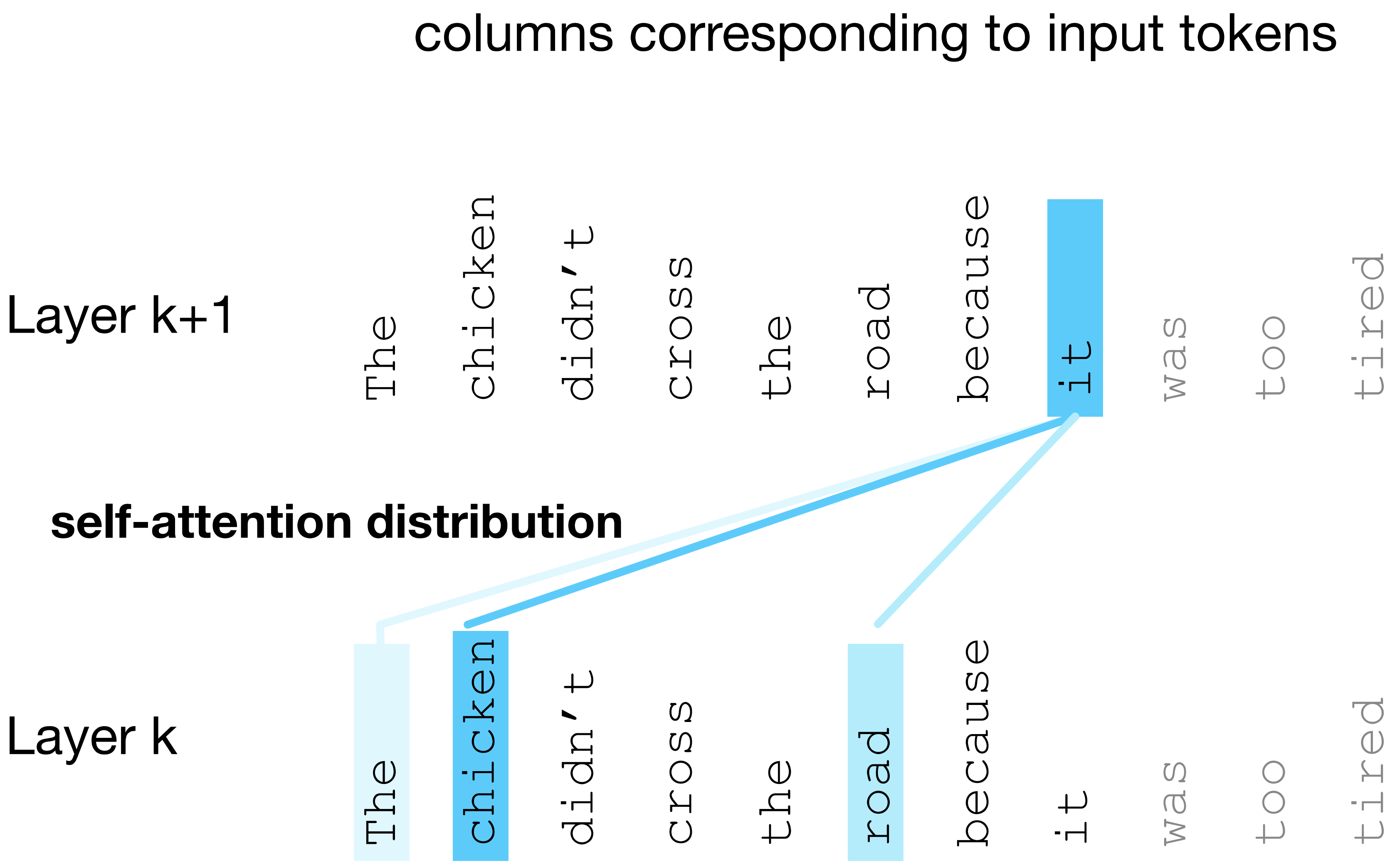
Zooming in


h_{i-1}
h_i
h_{i+1}
Feedforward
Layer Norm
MultiHead Attention
Layer Norm
...
...
x_{i-1}
x_i
x_{i+1}

Intuition of attention

- Build up the representation of a word by selectively integrating information from all the neighbouring words
- We say that a word "attends to" some neighbouring words more than others

Intuition of attention



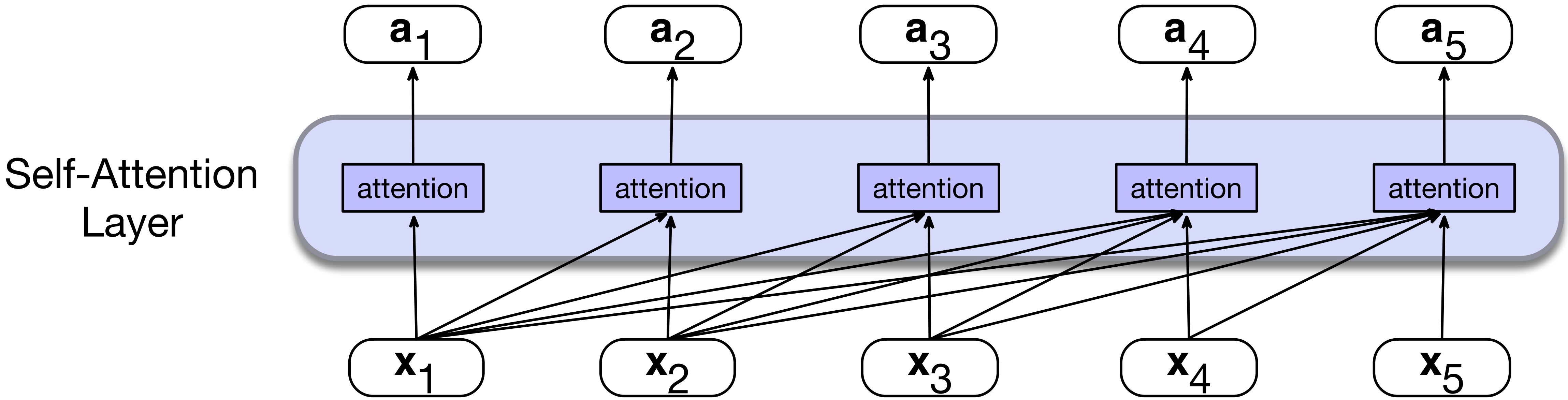
Attention definition

A mechanism for helping compute the embedding for a token by selectively attending to and integrating information from surrounding tokens (at the previous layer).

More formally: a method for doing a weighted sum of vectors.

$$\mathbf{v}^{k+1} = \sum_{i=1}^{n} \alpha_i \cdot \mathbf{v}_i^k$$

Attention can respect time (causal)



$$\mathbf{a}_j = \sum_{i=1}^{n} (\alpha_i \cdot \mathbb{M}(i,j)) \cdot \mathbf{x}_i \qquad \mathbb{M} = \begin{cases} 1 & \text{if } \; i \leq j \\ 0 & \text{else} \end{cases}$$

Simplified version of attention: a sum of prior words weighted by their similarity with the current word

Given a sequence of token embeddings:

$$\mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_3 \quad \mathbf{x}_4 \quad \mathbf{x}_5 \quad \mathbf{x}_6 \quad \mathbf{x}_7 \quad \mathbf{x}_\mathrm{i}$$

Produce: $\mathbf{a}_\mathrm{i}$ = a weighted sum of $\mathbf{x}_1$ through $\mathbf{x}_7$ (and $\mathbf{x}_\mathrm{i}$)

Weighted by their similarity to $\mathbf{x}_\mathrm{i}$

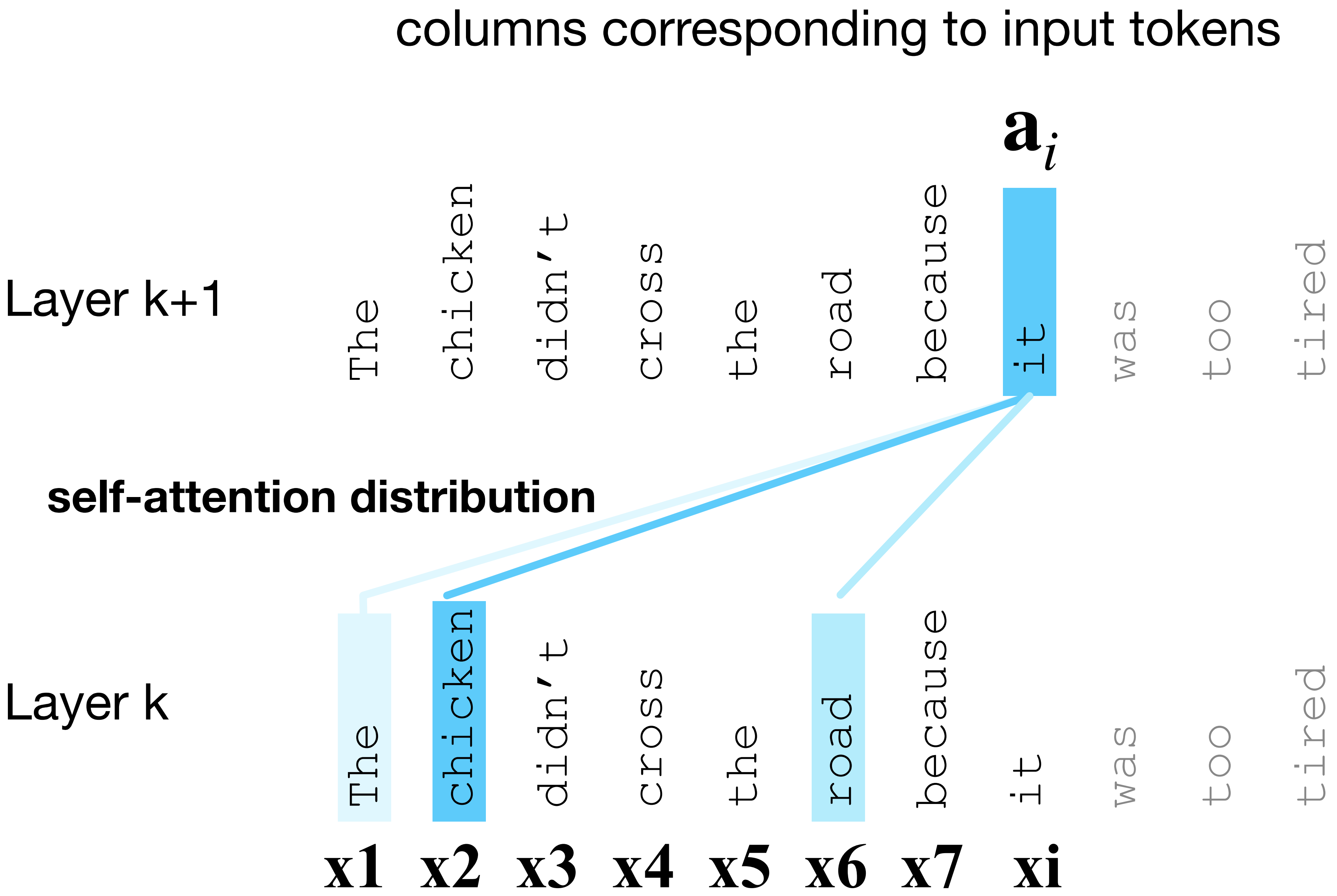
$$\mathbf{a}_i = \sum_{j \le i} \alpha_j \cdot \mathbf{x}_j$$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$

$$\mathbf{a}_i = \left(\sum_{j=1}^{7} \alpha_j \cdot \mathbf{x}_j\right) + \alpha_i \cdot \mathbf{x}_i$$

$$\alpha = \text{softmax}([\text{score}(\mathbf{x}_i, \mathbf{x}_j) \text{ for j in } 1...7, i])$$

Intuition of attention



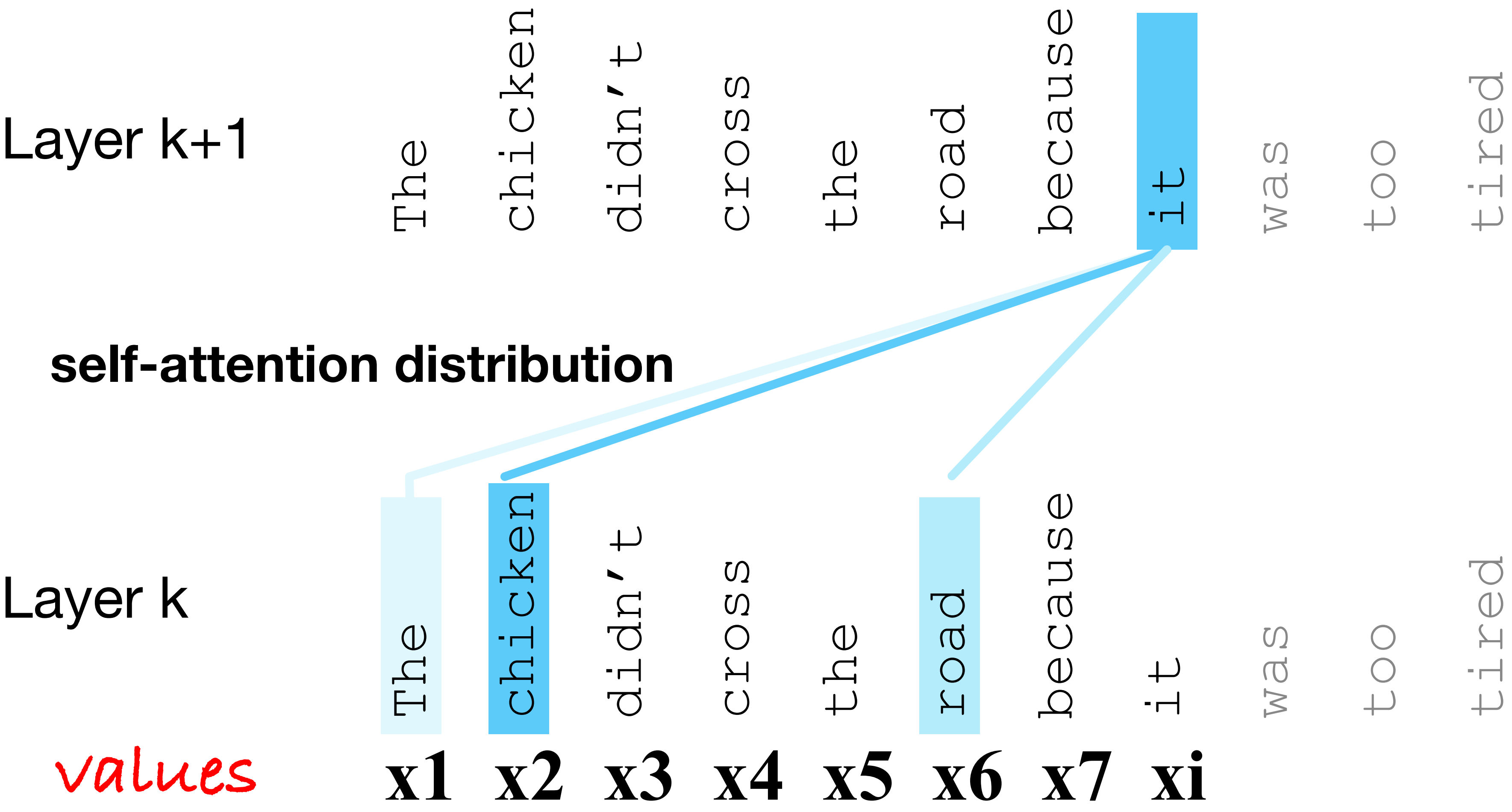
An Actual Attention Head is slightly more complicated

High-level idea: instead of using vectors (like x_i and x_4) directly, we'll represent 3 separate roles each vector $\mathbf{x}_i$ plays:

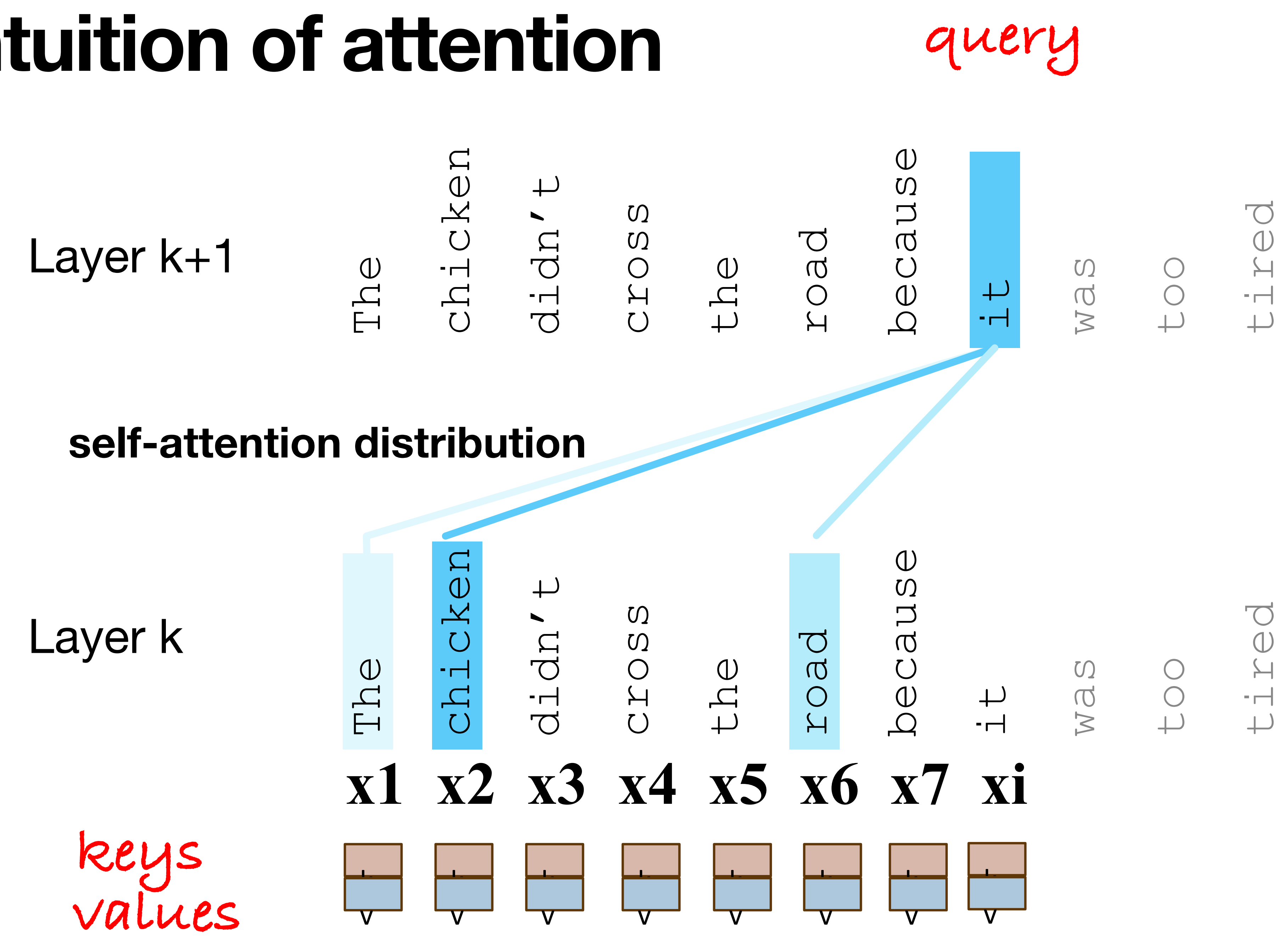
- **query**: As *the current element* being compared to the preceding inputs.
- **key**: as *a preceding input* that is being compared to the current element to determine a similarity
- **value**: a value of a preceding element that gets weighted and summed

Intuition of attention

query



Intuition of attention



An Actual Attention Head is slightly more complicated

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

- **query**: $\mathbf{W}^{\mathbf{Q}}$
- **key**: $\mathbf{W}^{\mathbf{K}}$
- **value**: $\mathbf{W}^{\mathbf{V}}$

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

Note: $\mathbf{x}_i, \mathbf{q}_i, \mathbf{k}_i, \mathbf{v}_i$ are row vectors here

An Actual Attention Head is slightly more complicated

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

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

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 $\mathbf{v}_j$

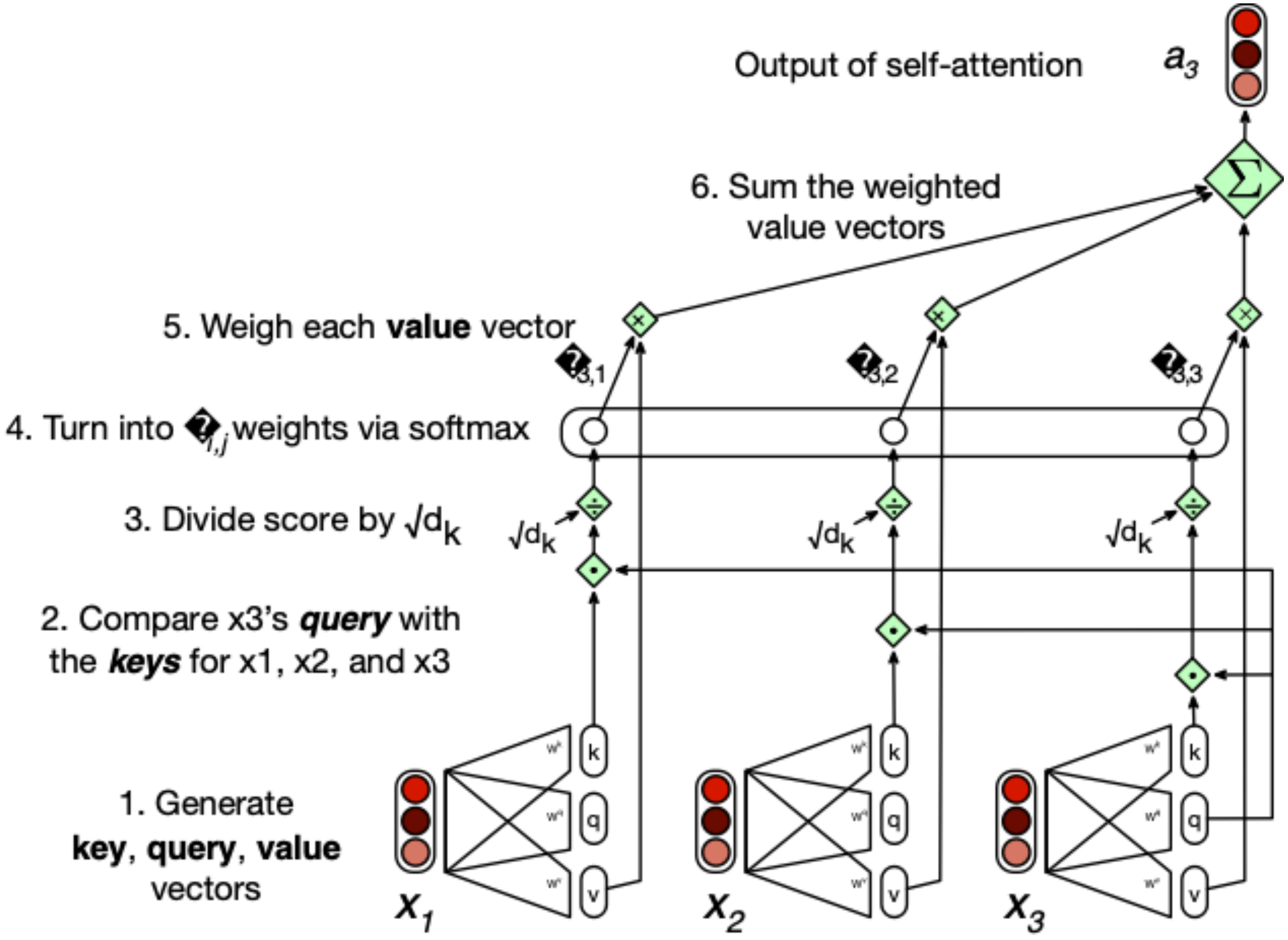
Final equations for one attention head

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

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i\mathbf{k}_j^\top}{\sqrt{d^k}} \qquad \alpha = \text{softmax}([\text{score}(\mathbf{x}_i, \mathbf{x}_j)\forall\ j \le i])$$

$$\mathbf{a}_i = \sum_{j \le i} \alpha_j \cdot \mathbf{v}_j$$

Example: calculating the value of a3



An Actual Attention Head is 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
    - E.g., different linguistic relationships or patterns in the context

$$\mathbf{q}_i^c = \mathbf{x}_i \mathbf{W}^{Qc} \quad \mathbf{k}_i^c = \mathbf{x}_i \mathbf{W}^{Kc} \quad \mathbf{v}_i^c = \mathbf{x}_i \mathbf{W}^{Vc}$$

$$\text{score}^c(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i^c \mathbf{k}_j^{c\top}}{\sqrt{d^k}} \qquad \alpha_i^c = \text{softmax}([\text{score}^c(\mathbf{x}_i, \mathbf{x}_j) \forall\, j \le i])$$

$$\text{head}_i^c = \sum_{j \le i} \alpha_{i,j}^c \cdot \mathbf{v}_j^c \qquad \mathbf{a}_i = (\text{head}^1 \oplus \text{head}^2 \ldots \oplus \text{head}^h)\mathbf{W}^O$$

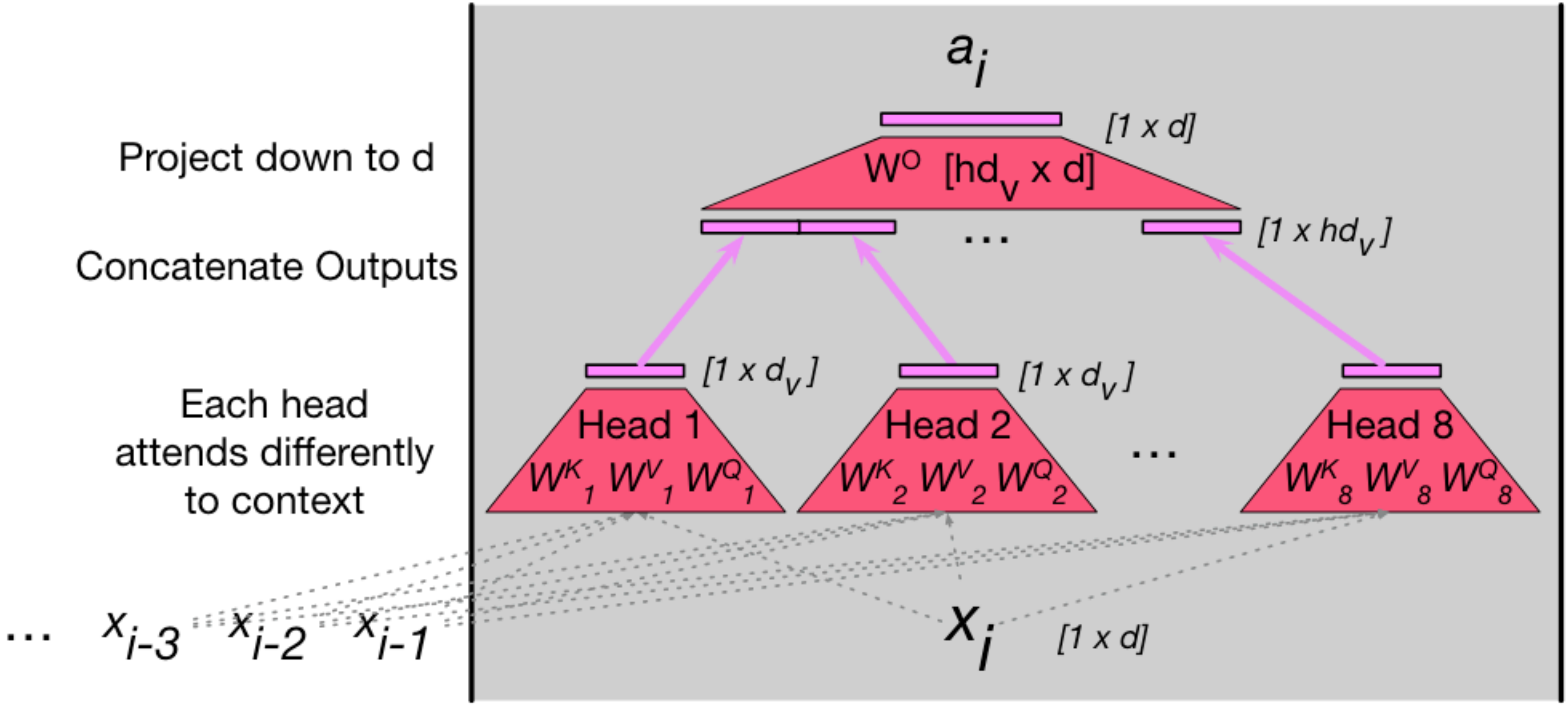
$$\text{MultiHeadAttention}(\mathbf{x}_i, [\mathbf{x}_1, \ldots, \mathbf{x}_n]) = \mathbf{a}_i$$

Multi-head attention

Project down to d

Concatenate Outputs

Each head
attends differently
to context



Parallelizing computation using X

For attention/transformer block we've been computing a **single** output at a **single** time step *i* in a **single** residual stream.

But we can pack the N tokens of the input sequence into a single matrix **X** of size $[N \times d]$.

Each row of X is the embedding of one token of the input.

X can have 1K-32K rows, each of the dimensionality of the embedding d (the **model dimension**)

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

QK^T

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

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \mathbf{k}_j^\top}{\sqrt{d^k}}$$

$$\mathbf{S} = \frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d^k}}$$

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

Parallelizing attention

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

$$\mathbf{A} = \text{softmax}\left(\mathbb{M}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d^k}}\right)\right)\mathbf{V}$$

Masking out the future

- What is this mask function?
  QK^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!

$$\mathbf{A} = \mathrm{softmax}\left(\mathbb{M}\left(\frac{\mathbf{QK}^\top}{\sqrt{d^k}}\right)\right)\mathbf{V}$$

Masking out the future

Add –∞ to cells in upper triangle

The softmax will turn it to 0

$$\mathbf{A} = \text{softmax}\left(\mathbb{M}\left(\frac{\mathbf{QK}^\top}{\sqrt{d^k}}\right)\right)\mathbf{V}$$

N

q1•k1	−∞	−∞	−∞
q2•k1	**q2•k2**	−∞	−∞
q3•k1	**q3•k2**	**q3•k3**	−∞
q4•k1	**q4•k2**	**q4•k3**	**q4•k4**

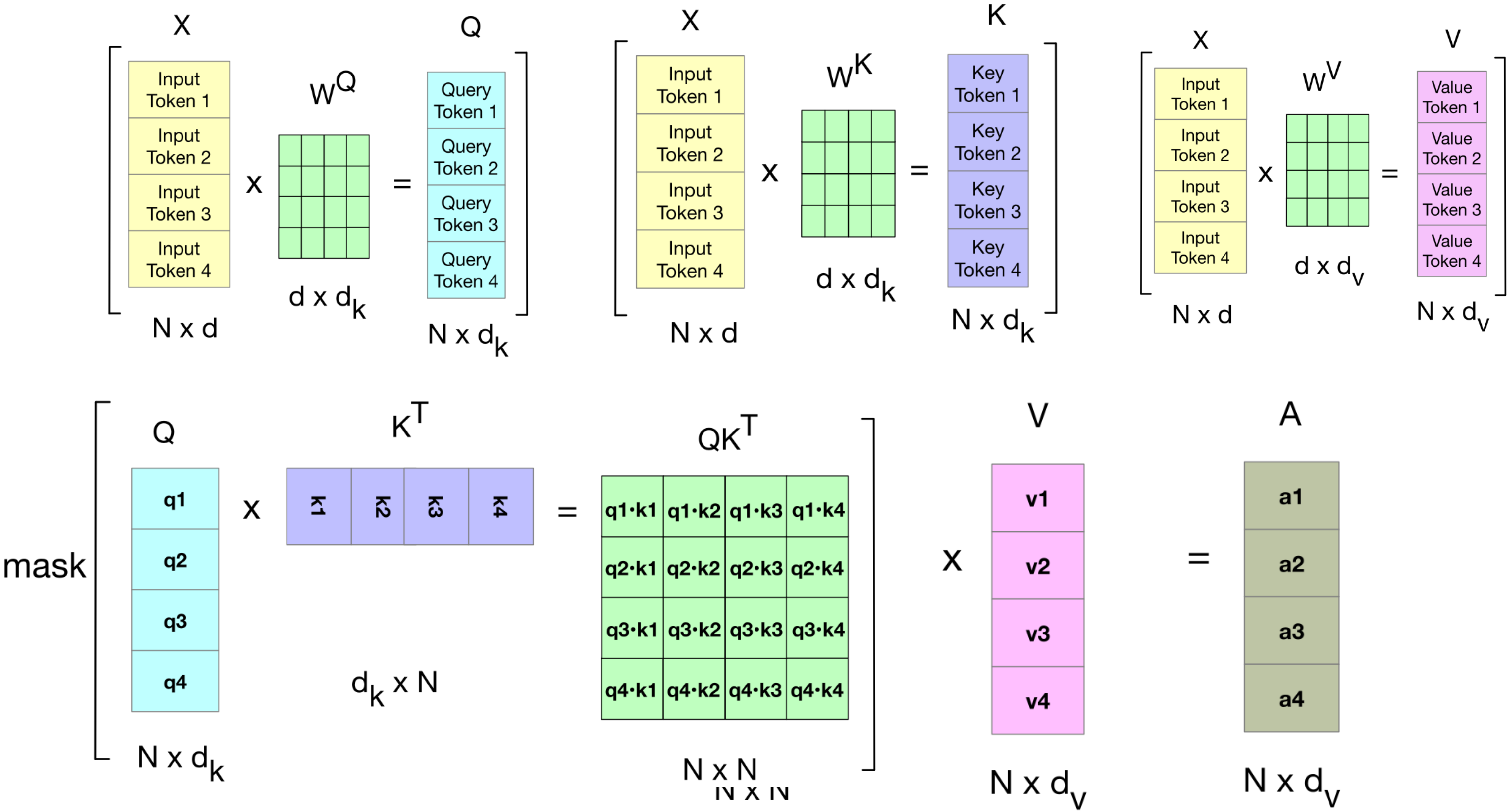
N

Another point: Attention is quadratic in length

$$\mathbf{A} = \mathrm{softmax}\left(\mathbb{M}\left(\frac{\mathbf{QK}^\top}{\sqrt{d^k}}\right)\right)\mathbf{V}$$

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

Attention again



Parallelizing Multi-head Attention

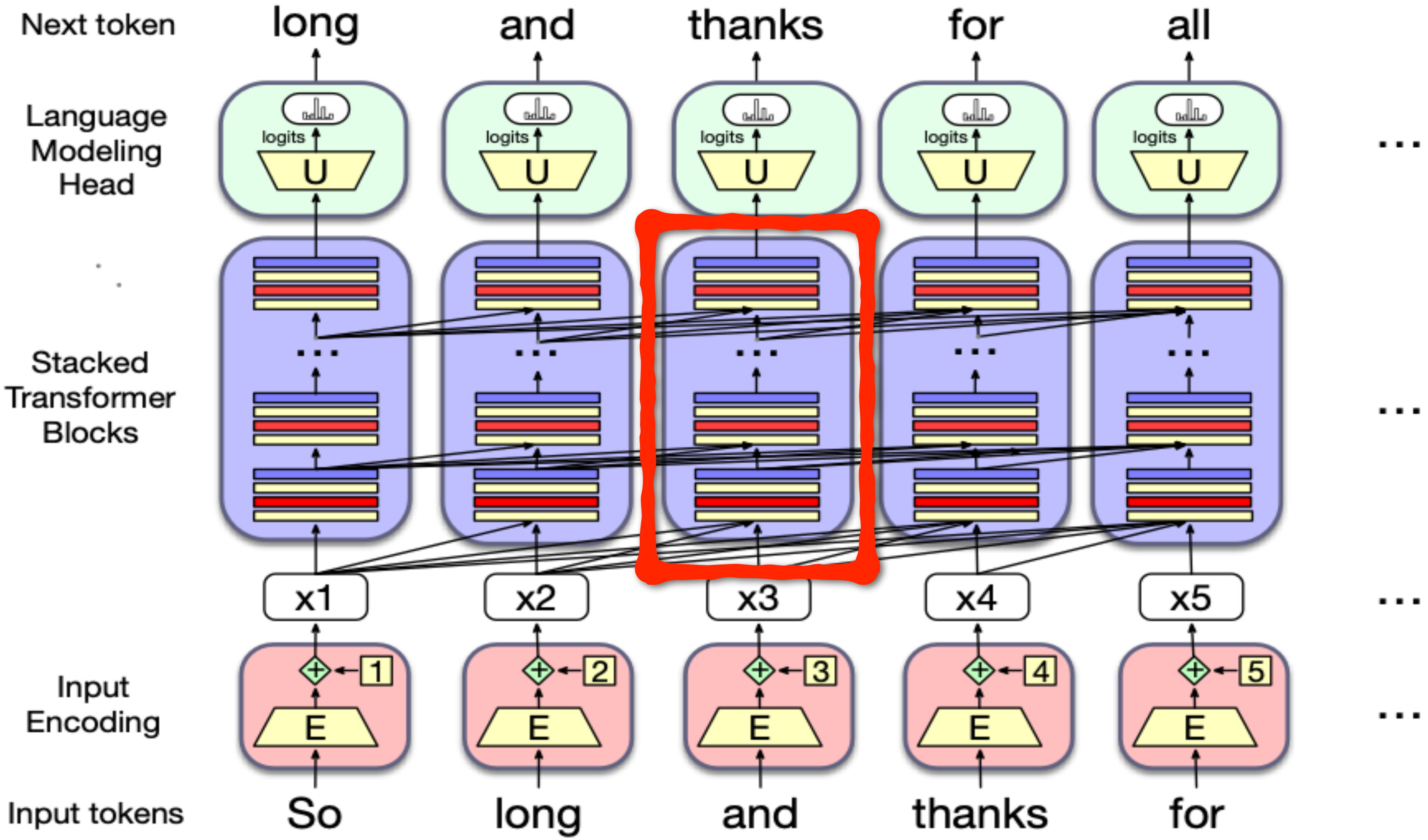
$$\mathbf{Q^i} = \mathbf{XW^{Qi}}\ ; \quad \mathbf{K^i} \; = \; \mathbf{XW^{Ki}}\ ; \quad \mathbf{V^i} = \mathbf{XW^{Vi}}$$

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

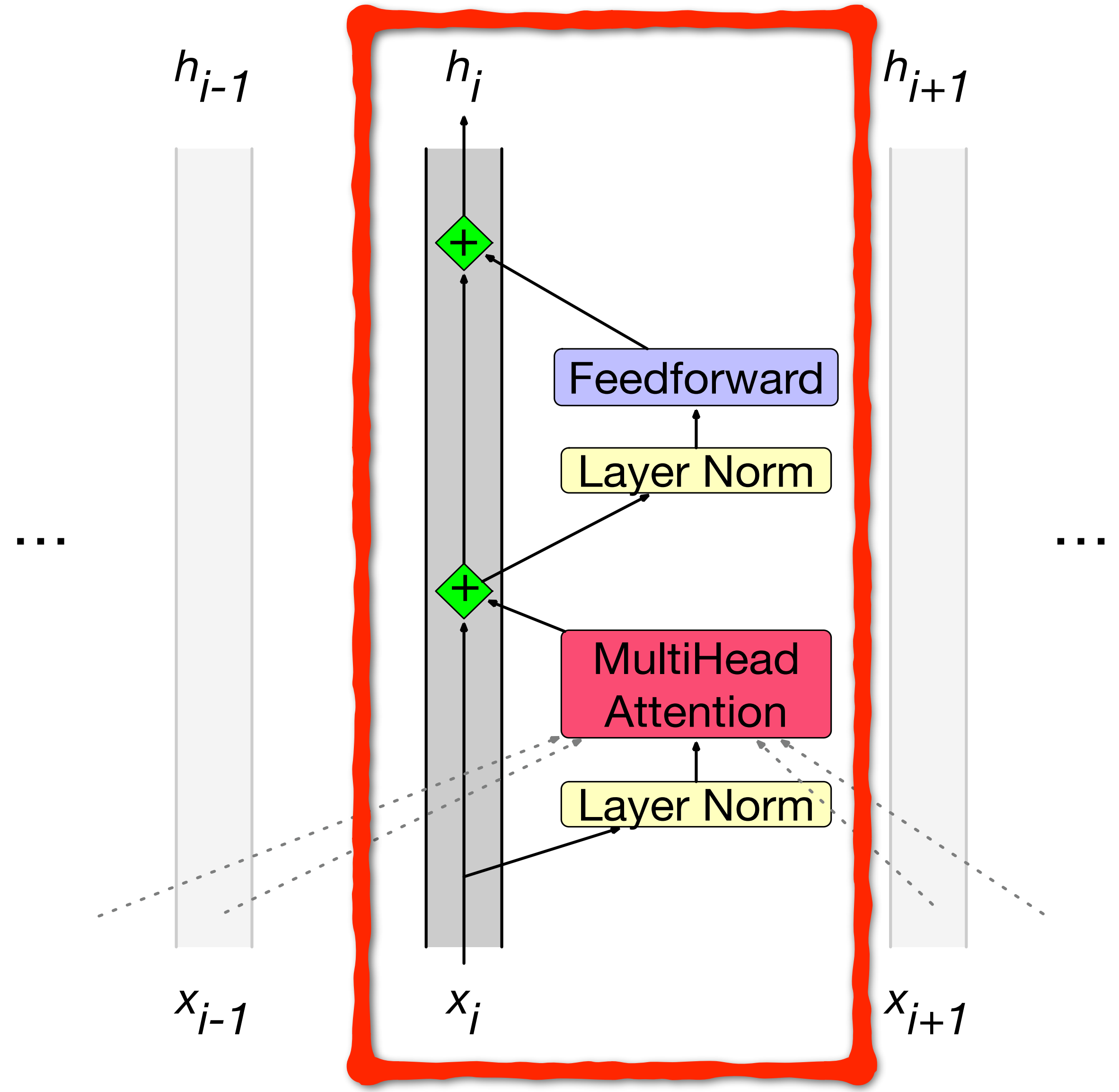
$$\mathrm{MultiHeadAttention}(\mathbf{X}) \; = \; (\mathbf{head}_1 \oplus \mathbf{head}_2 \ldots \oplus \mathbf{head}_A)\mathbf{W^O}$$

This is equivalent to running the attention heads in parallel and adding their results back to the residual stream (Whiteboard)

Reminder: transformer architecture

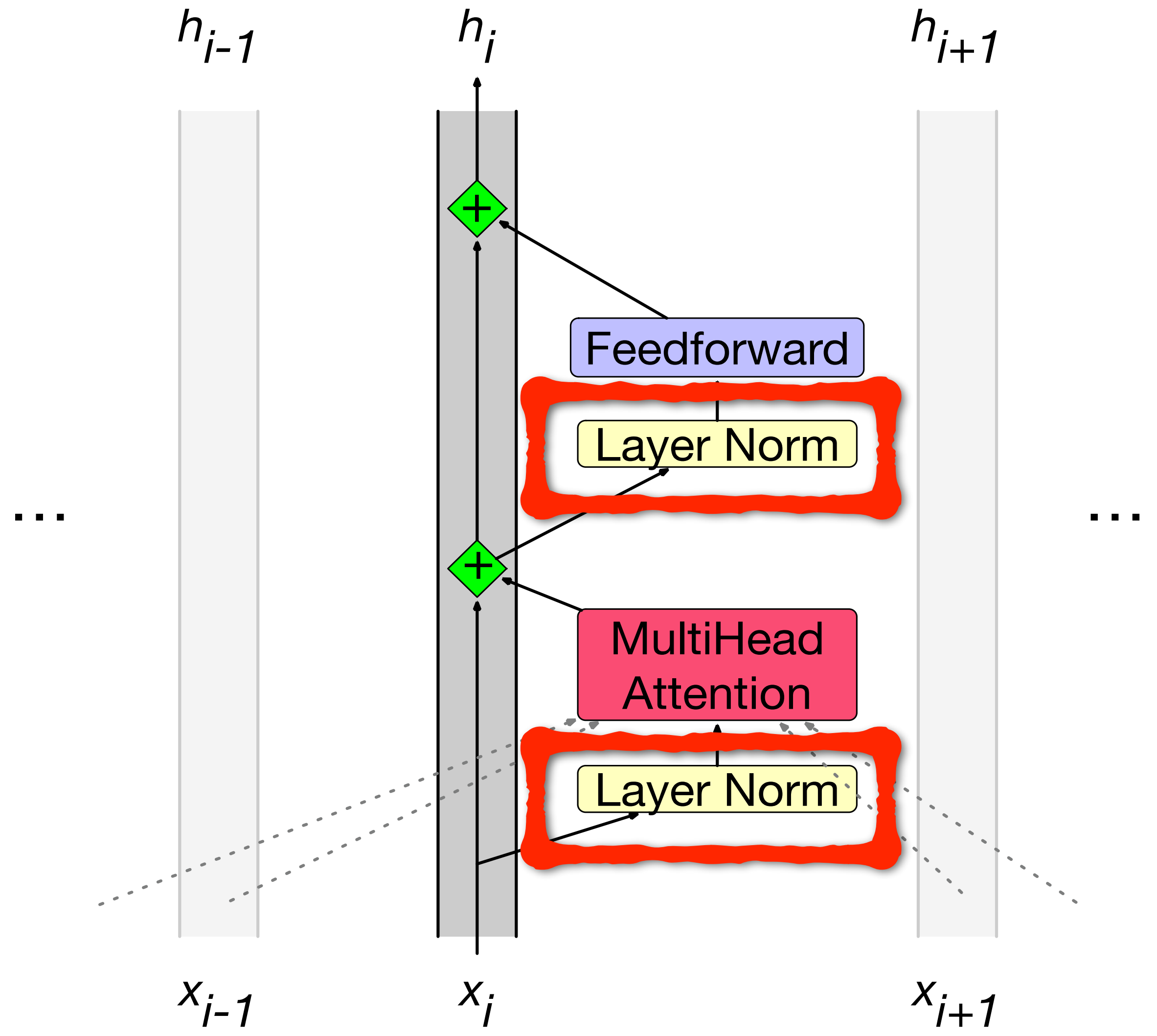


A single transformer block



Sublayers of the transformer block: Layer Norm

$$\mathrm{LayerNorm}(\mathbf{x}_i) = \ldots$$



Layer Norm

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

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

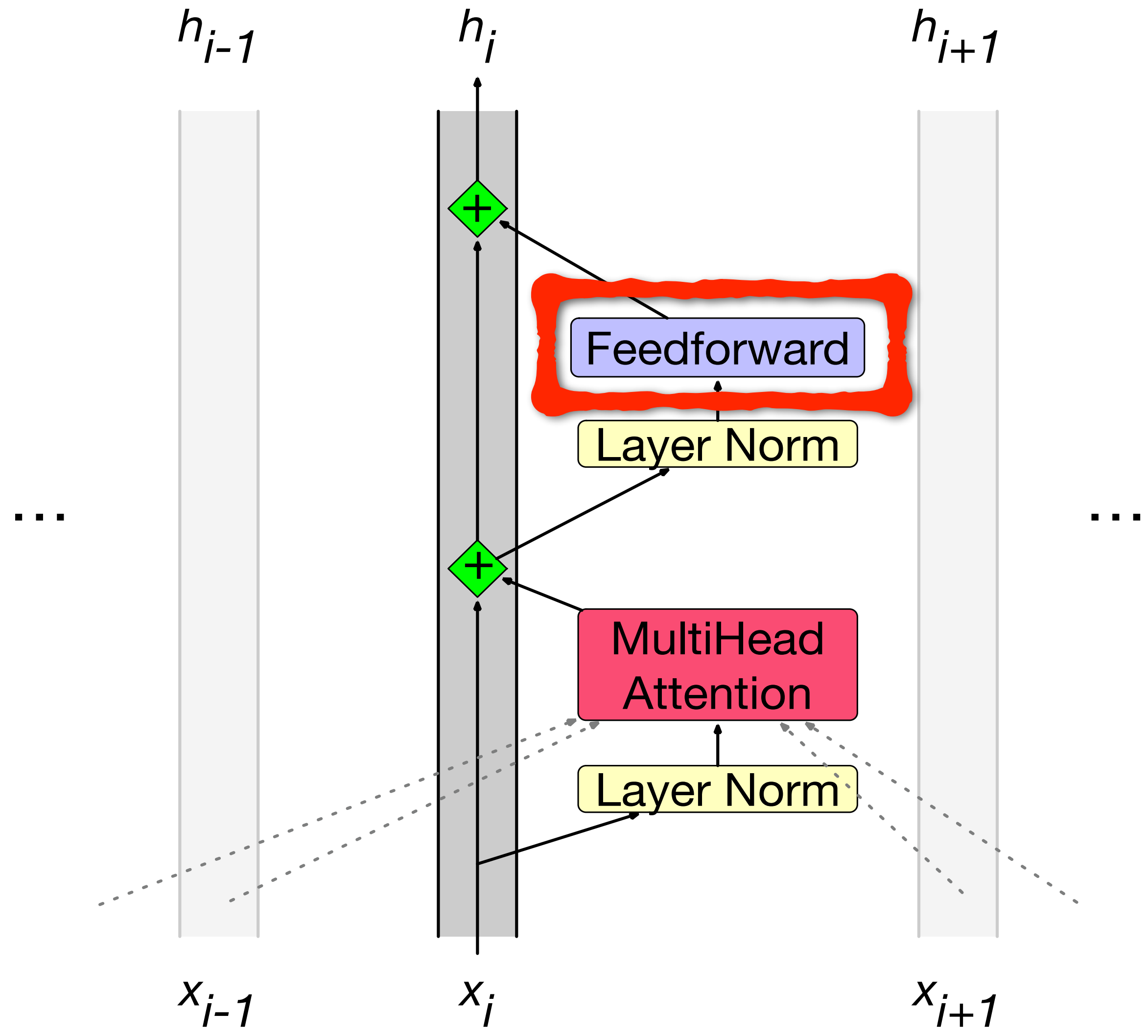
$$\sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2}$$

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

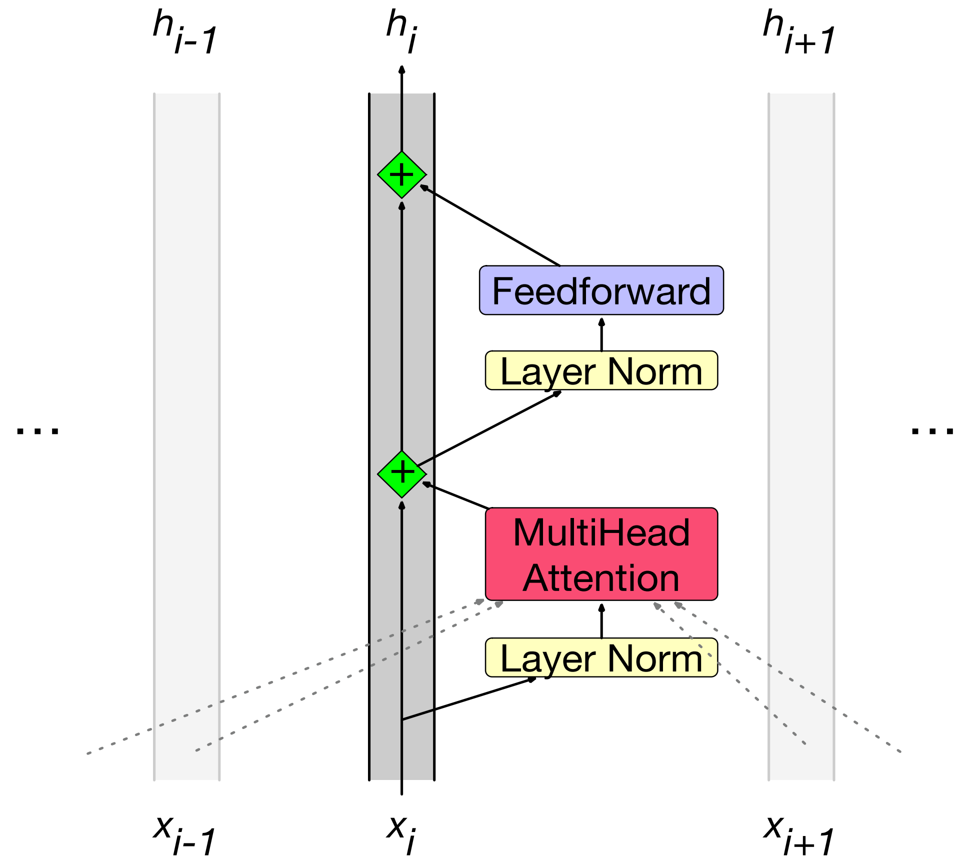
$$\text{LayerNorm}(\mathbf{x}) = \gamma \frac{(\mathbf{x} - \mu)}{\sigma} + \beta$$

Sublayers of the transformer block: FFN

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



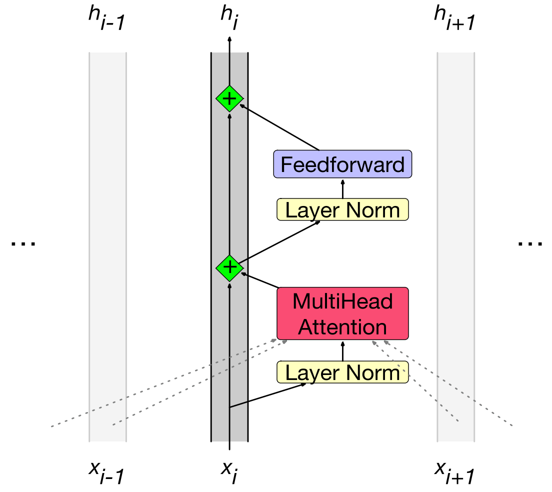
Putting together a single transformer block



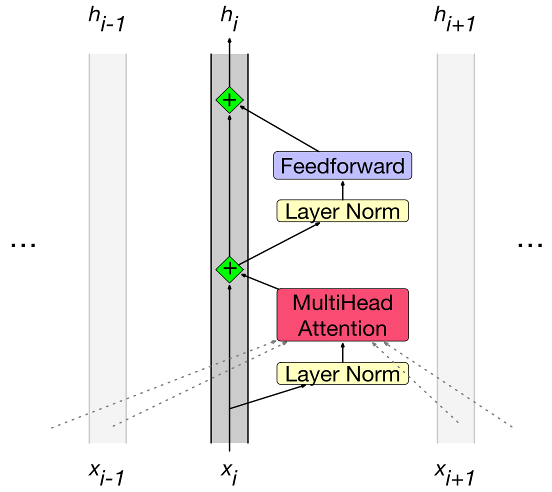
$$
\begin{aligned}
\mathbf{t}_i^1 &= \mathrm{LayerNorm}(\mathbf{x}_i) \\
\mathbf{t}_i^2 &= \mathrm{MultiHeadAttention}(\mathbf{t}_i^1, [\mathbf{t}_1^1, \cdots, \mathbf{t}_N^1]) \\
\mathbf{t}_i^3 &= \mathbf{t}_i^2 + \mathbf{x}_i \\
\mathbf{t}_i^4 &= \mathrm{LayerNorm}(\mathbf{t}_i^3) \\
\mathbf{t}_i^5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

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

Block 2

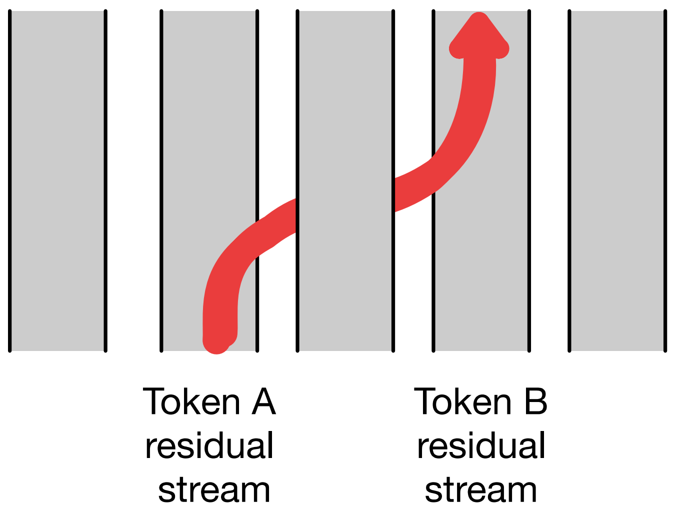


Block 1



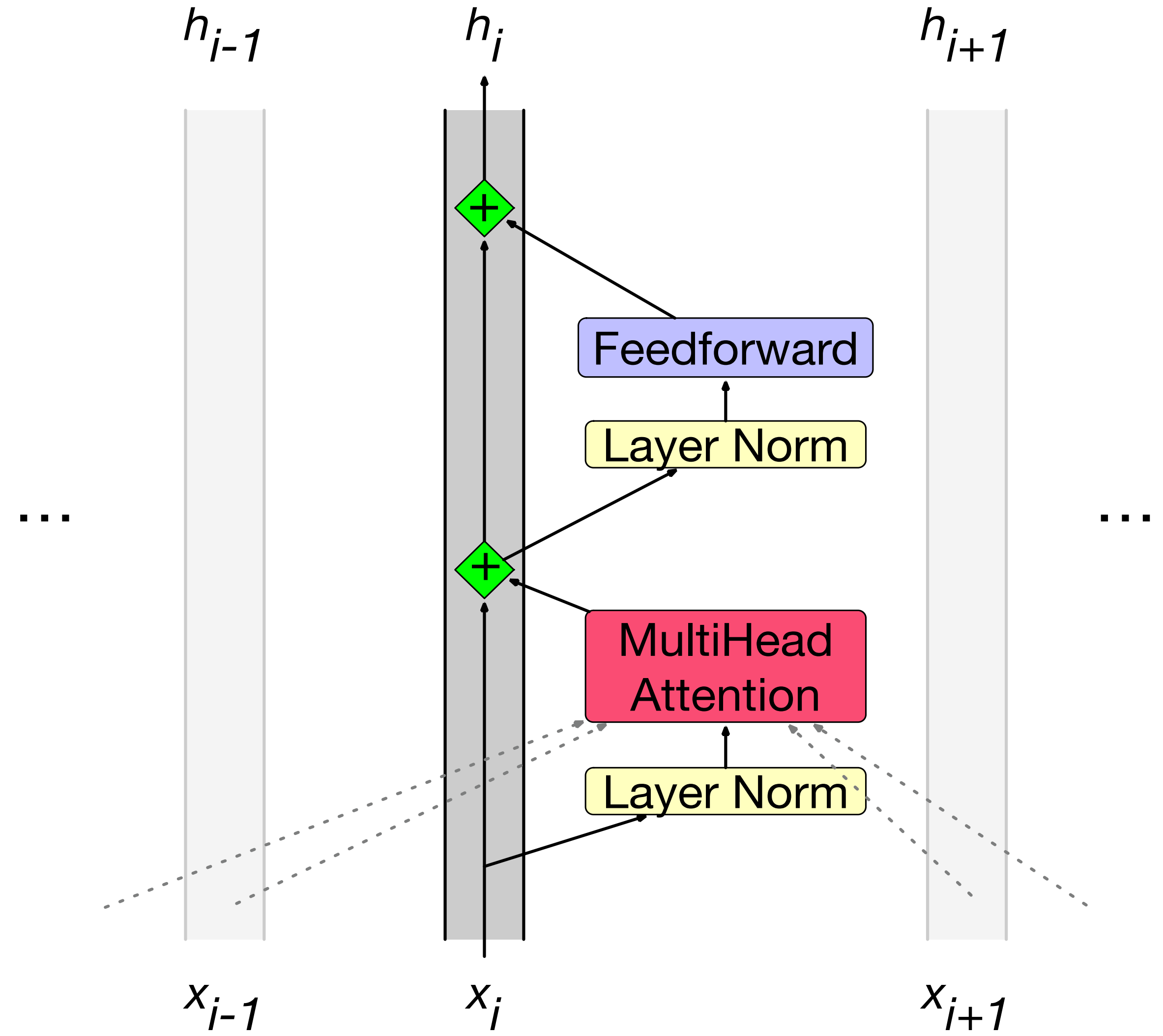
Residual streams and attention

- Notice that all parts of the transformer block apply to 1 residual stream except attention, which takes information from other tokens
- 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



Elhage et al. (2021) - A Mathematical Framework for Transformer Circuits

Residual stream view



- FFN and attention layers **read** from and **write** to the residual stream

- FFN layers have access to “one lane” only. Same computation applied on every “lane”

- Attention layers can read from from other “lanes” too

- $\mathbf{x}_i$ is transformed into $\mathbf{h}_i^L$ through a sequence of non-linear transformations

Putting together a single transformer block

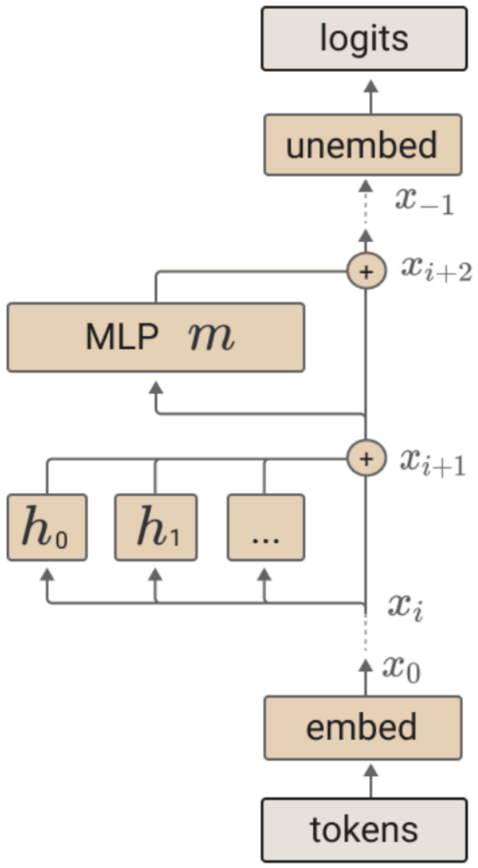
Single vector:

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

Matrix of inputs:

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

Residual stream view



The final logits are produced by applying the unembedding.

$$T(t) = W_U x_{-1}$$

An MLP layer, m, is run and added to the residual stream.

$$x_{i+2} = x_{i+1} + m(x_{i+1})$$

Each attention head, h, is run and added to the residual stream.

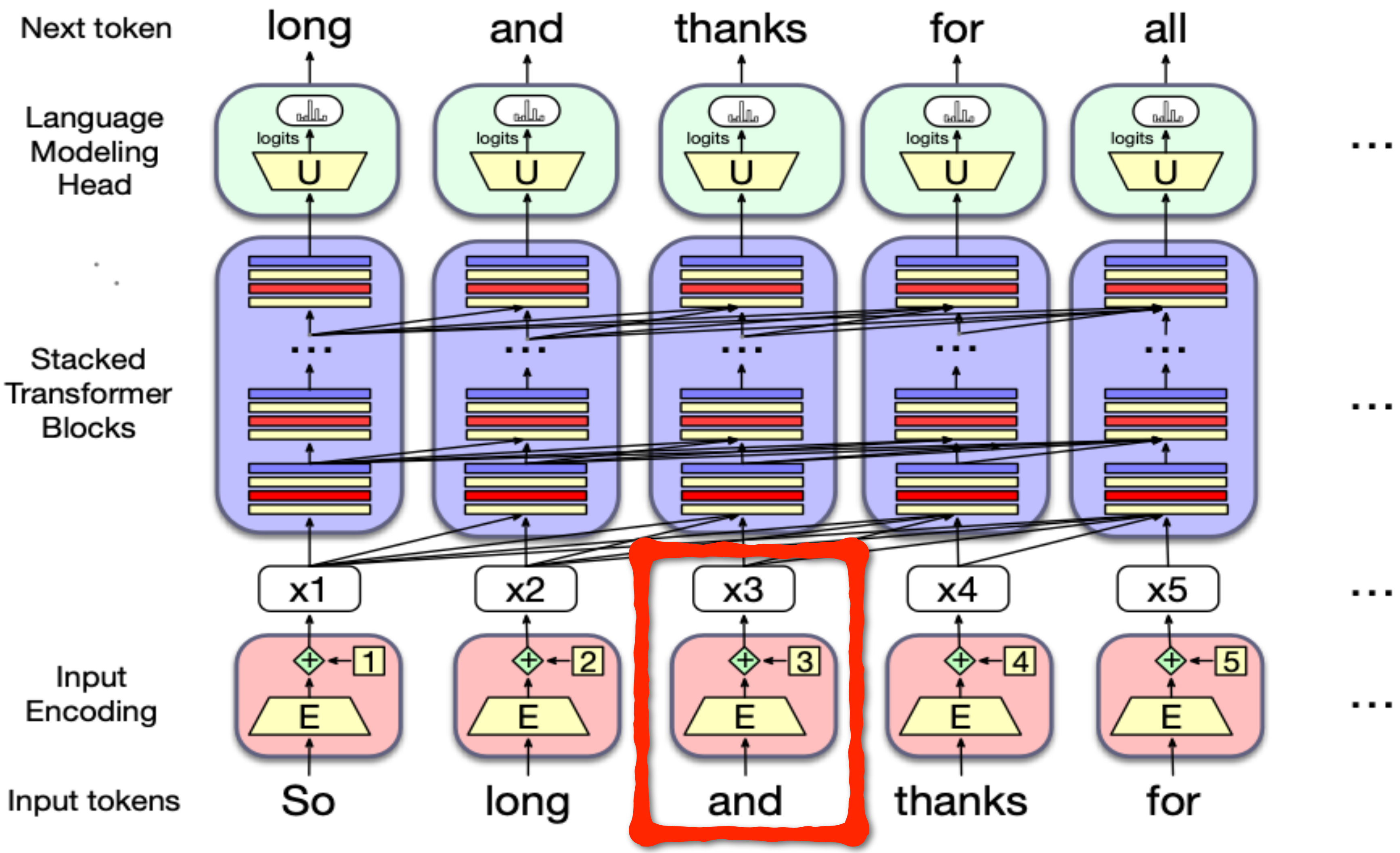
$$x_{i+1} = x_i + \sum_{h \in H_i} h(x_i)$$

Token embedding.

$$x_0 = W_E t$$

Elhage et al. (2021) - A Mathematical Framework for Transformer Circuits

Reminder: transformer architecture

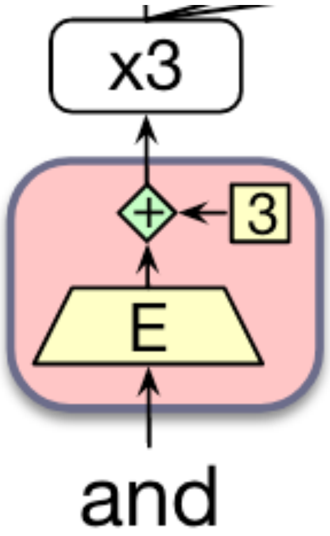


Token and Position Embeddings

The matrix $\mathbf{X}$ (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



Token Embeddings

Embedding matrix $\mathbf{E}$ has shape $|\mathscr{V}| \times d$.

- One row for each of the $|\mathscr{V}|$ tokens in the vocabulary.
- Each word is a row vector of *d* dimensions

Given: string "*Thanks for all the*"

1. Tokenize with BPE and convert into vocab indices

input_ids = [5,4000,10532,2224]

2. Select the corresponding rows from $\mathbf{E}$, each row an embedding

- (row 5, row 4000, row 10532, row 2224).

Position Embeddings

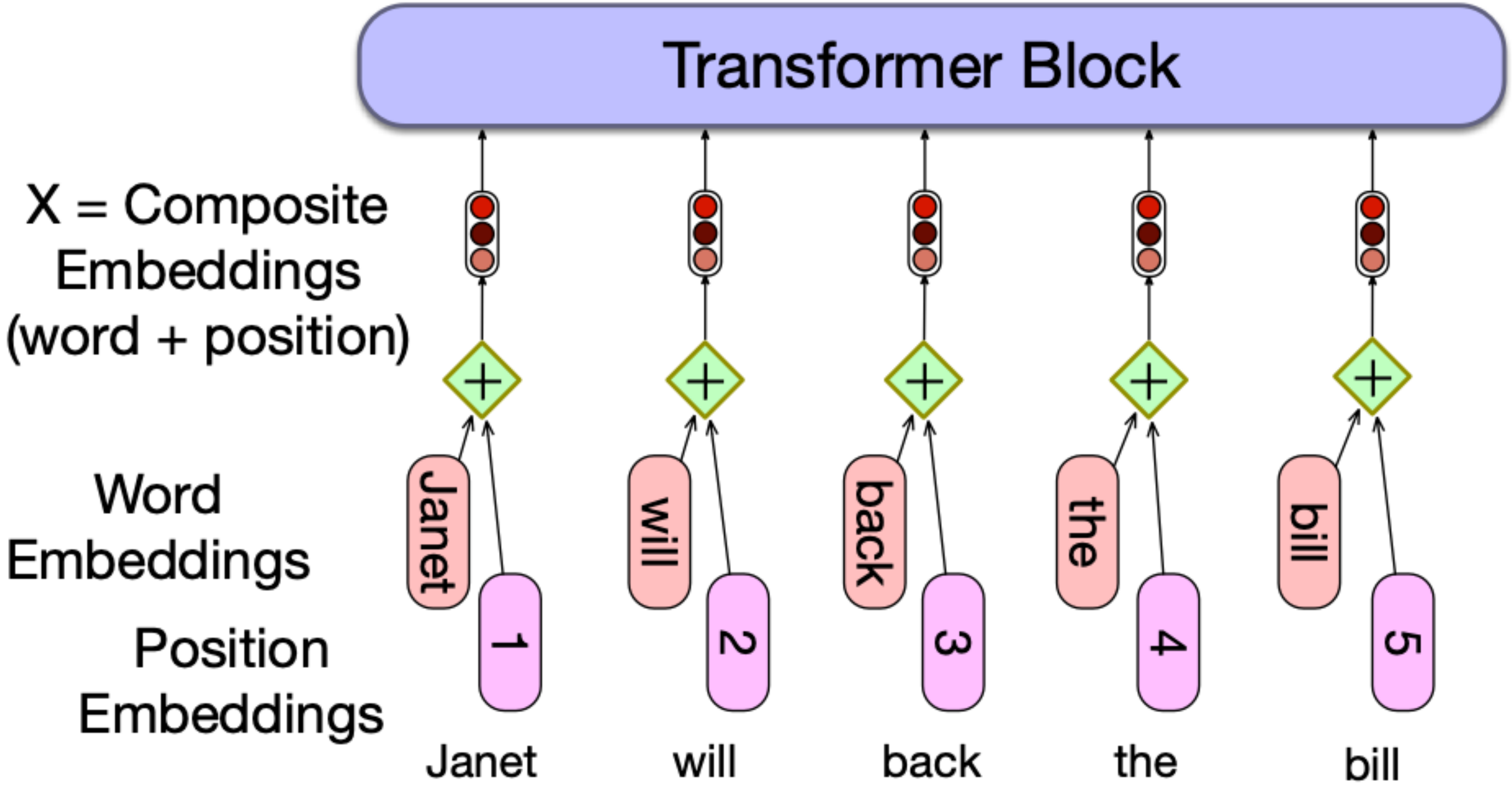
There are many methods, but we'll just describe the simplest: absolute position.

Goal: learn a position embedding matrix $\mathbf{E}_{\text{pos}}$ of shape $N \times d$.

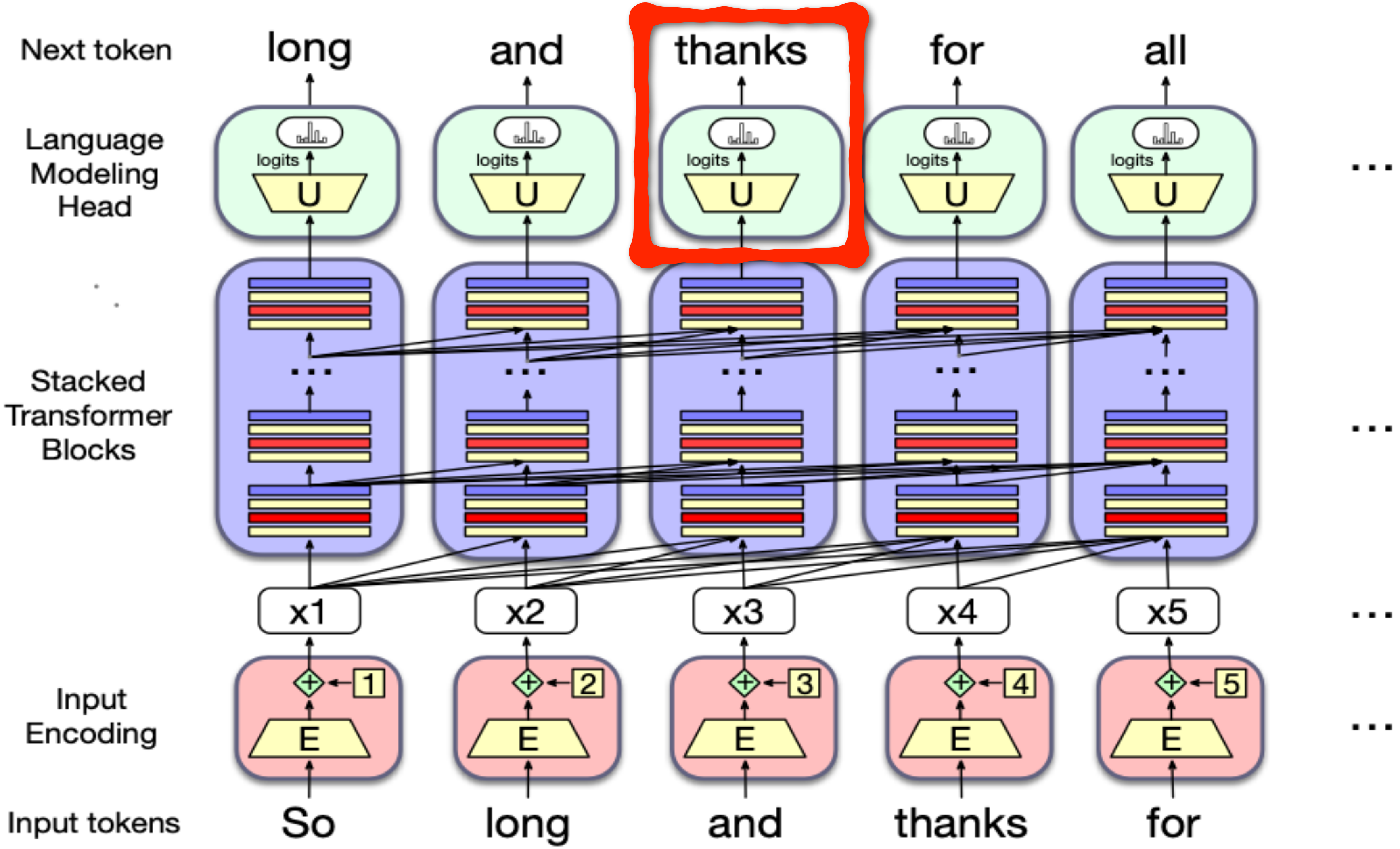
Start with randomly initialized embeddings

- 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.

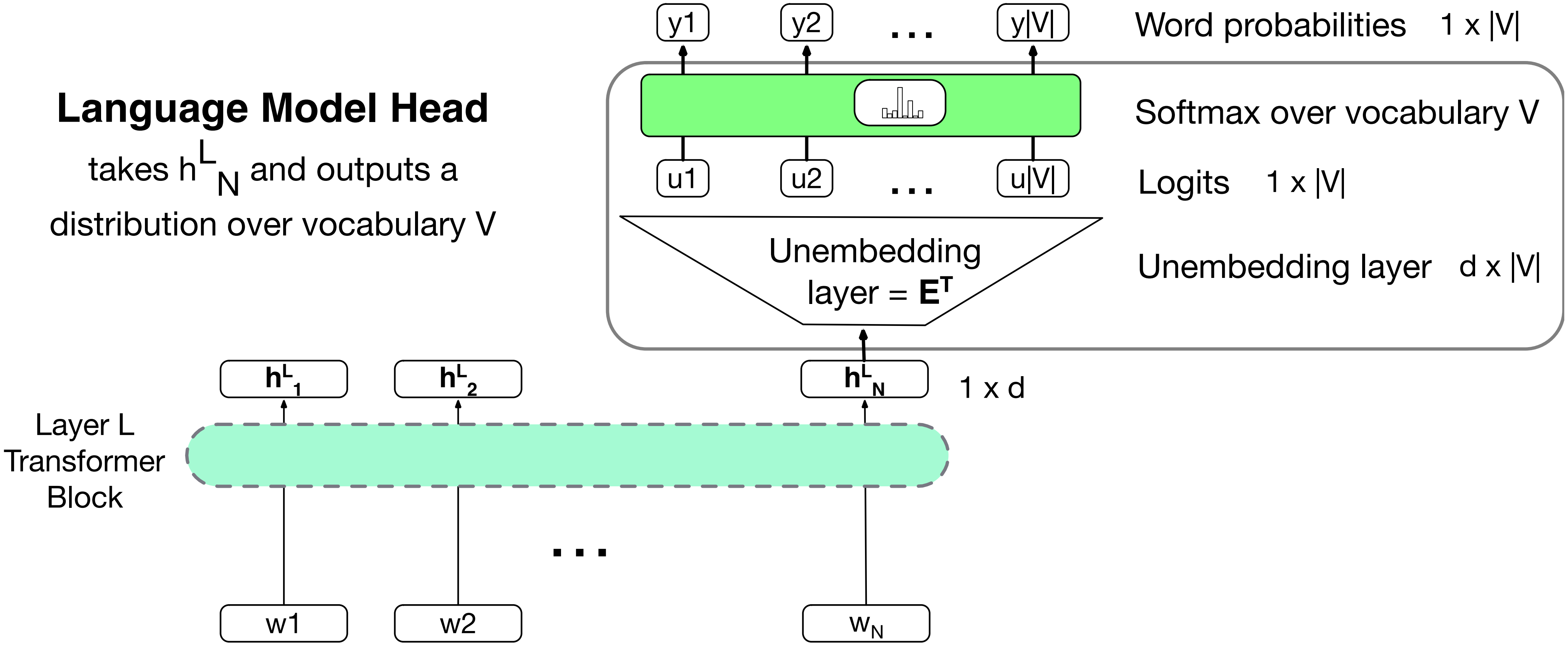
Each $\mathbf{x}_i$ is just the sum of word and position embeddings



Reminder: transformer architecture

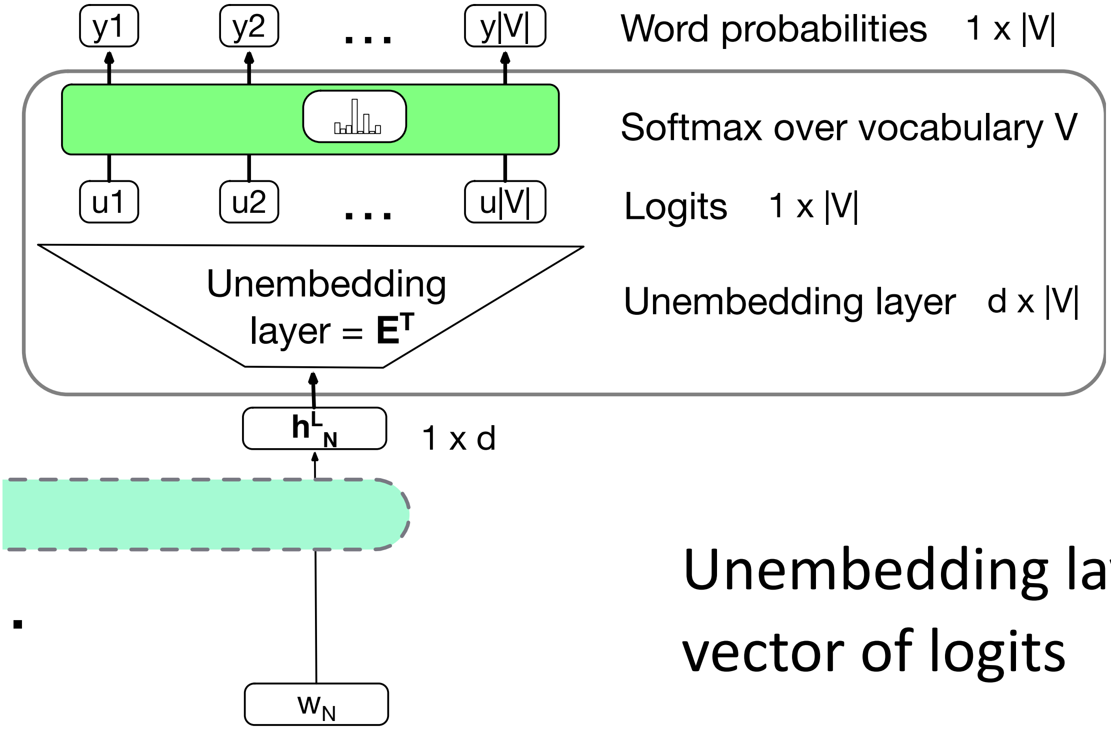


Language modeling head



Language modeling head

Unembedding layer: linear layer projects from $\mathbf{h}_N^L$ (shape $1 \times d$) to logit vector

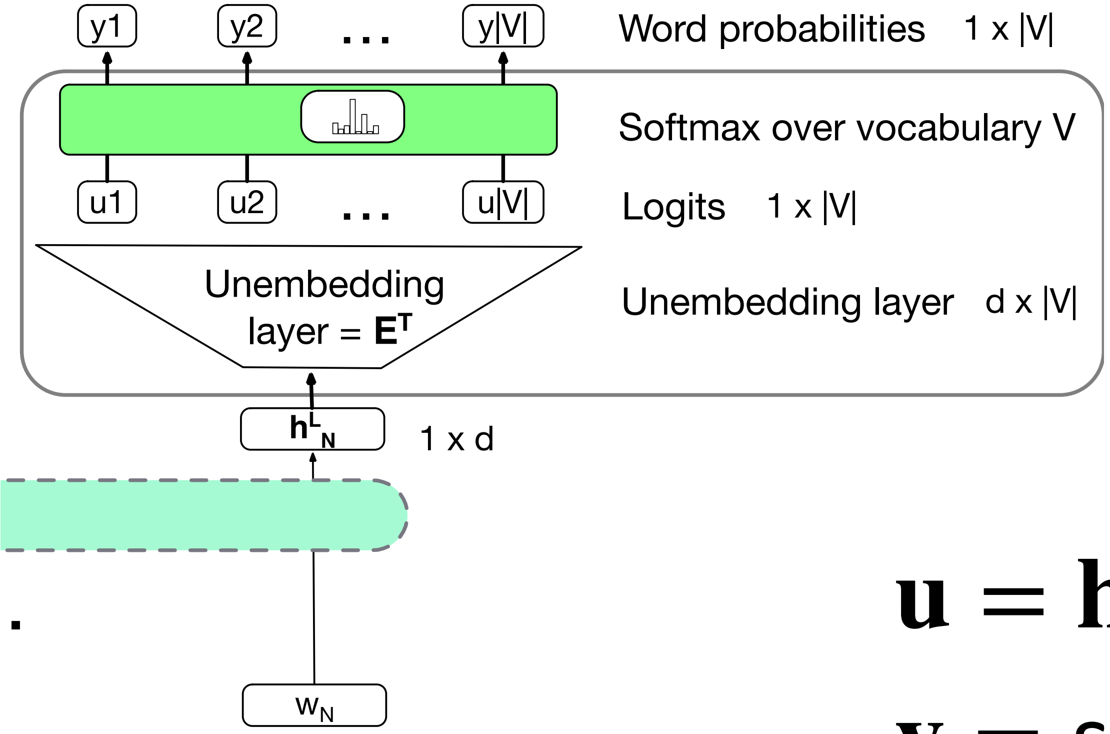


Why "unembedding"? **Tied** to $\mathbf{E}^T$

Weight tying, we use the same weights for two different matrices

Unembedding layer maps from an embedding to a $1 \times |\mathscr{V}|$ vector of logits

Language modeling head



Logits, the score vector $\mathbf{u}$

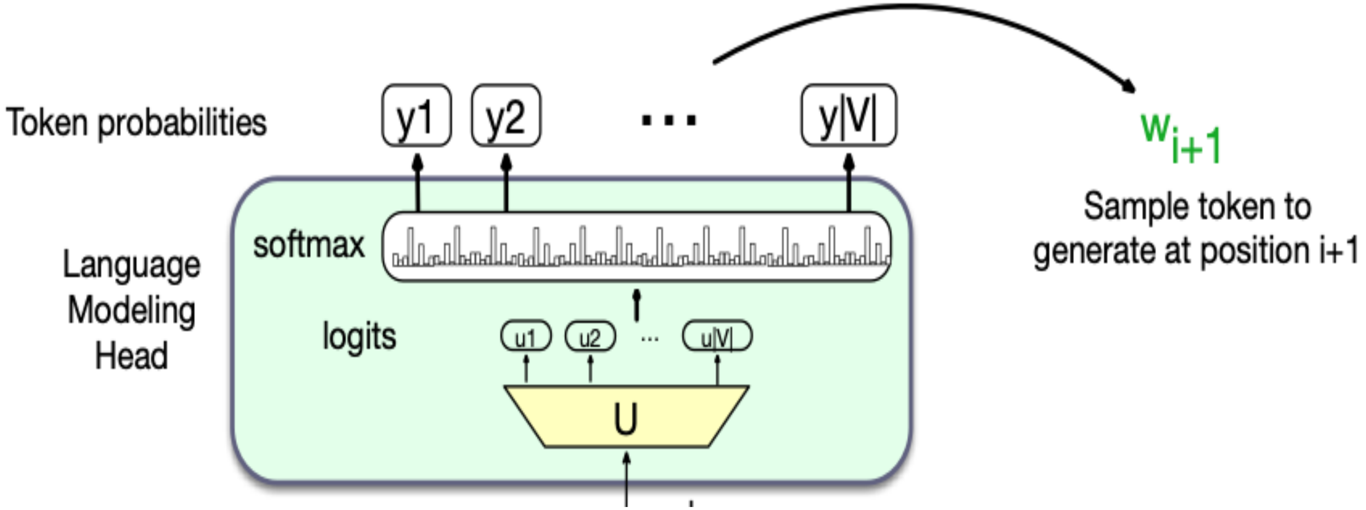
One score for each of the $|\mathcal{V}|$ possible words in the vocabulary $\mathcal{V}$. Shape $1 \times |\mathcal{V}|$.

Softmax turns the logits into probabilities over vocabulary. Shape $1 \times |\mathcal{V}|$.

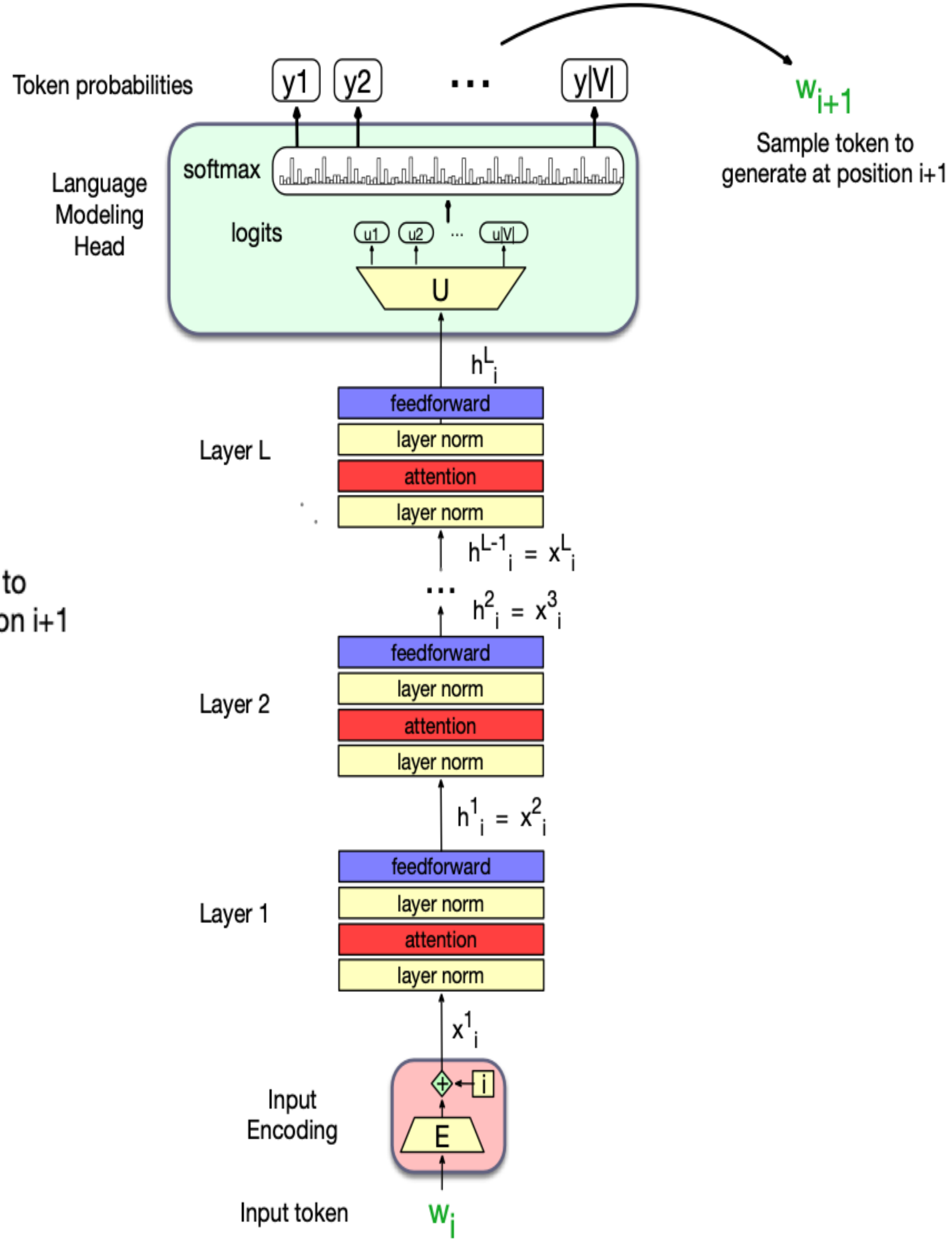
$$\mathbf{u} = \mathbf{h}_N^L \mathbf{E}^\top$$

$$\mathbf{y} = \text{softmax}(\mathbf{u})$$

The final transformer model



~ GPT



LM loss

The LM head takes output of final transformer layer L, multiplies it by unembedding layer and turns into probabilities:

$$\mathbf{u}_i = E\mathbf{h}_i^L \qquad\qquad \mathbf{y}_i = \text{softmax}(\mathbf{u}_i)$$

The loss is the probability of the next word, given output h^L_i :

$$\mathscr{L}_{LM}(x_i) = -\log P(x_{i+1} \mid \mathbf{h_i}^L) = -\log \hat{\mathbf{y}}[\mathbf{x_{i+1}}]$$

We get the gradients by taking the average of this loss over the batch

$$\mathscr{L}_{\text{LM}} = -\frac{1}{|\mathscr{B}|}\sum_{s\in\mathscr{B}}\frac{1}{|s|}\sum_{i\in s}\log P(x_{i+1} \mid \mathbf{h}_i^L)$$

Language Models are Unsupervised Multitask Learners

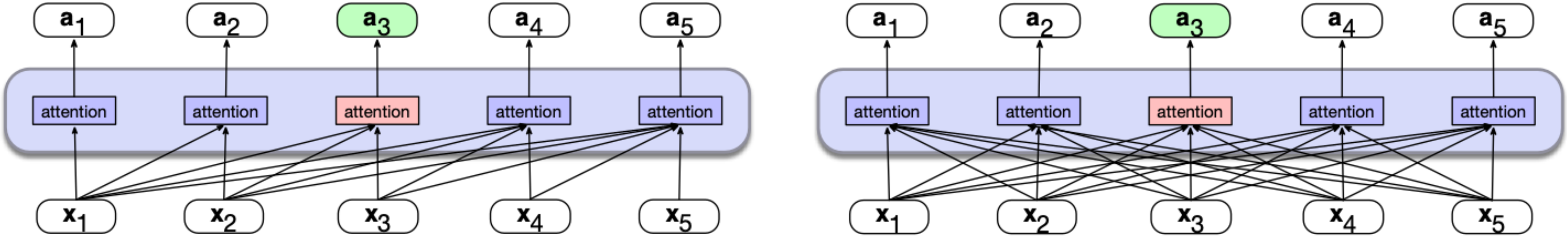
GPT-2 (Radford et al., 2019)

- Trained on ~40GB of text crawled from the internet
- Input context window N=1024 tokens, and model dimensionality {d=768, d=1024, d=1280, d=1600}
- {L=12, L=24, L=36, L=48} layers of transformer blocks
- *The resulting models have around {117M, 335M, 762M, 1542M} parameters*

Masked Language Modeling

- We've seen autoregressive (causal, left-to-right) LMs.
- But what about tasks for which we want to peak at future tokens?
  - Especially true for tasks where we map each input token to an output token
- **Bidirectional encoders** use **unmasked self-attention** to
  - map sequences of input embeddings $\mathbf{x}_1, \ldots, \mathbf{x_n}$
  - to sequences of output embeddings of the same length $\mathbf{h}_1, \ldots, \mathbf{h_n}$
  - where the output vectors have been contextualized using information from the entire input sequence.

Bidirectional Self-Attention



a) A causal self-attention layer

b) A bidirectional self-attention layer

We just remove the mask

Casual self-attention

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

$$\mathbf{A} = \text{softmax}\left(\mathbb{M}\left(\frac{\mathbf{QK}^\top}{\sqrt{d^k}}\right)\right)$$

Bidirectional self-attention

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

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{QK}^\top}{\sqrt{d^k}}\right)$$

Masked training intuition

- For **left-to-right (causal; decoder-only) LMs**, the model tries to predict the last word from prior words:

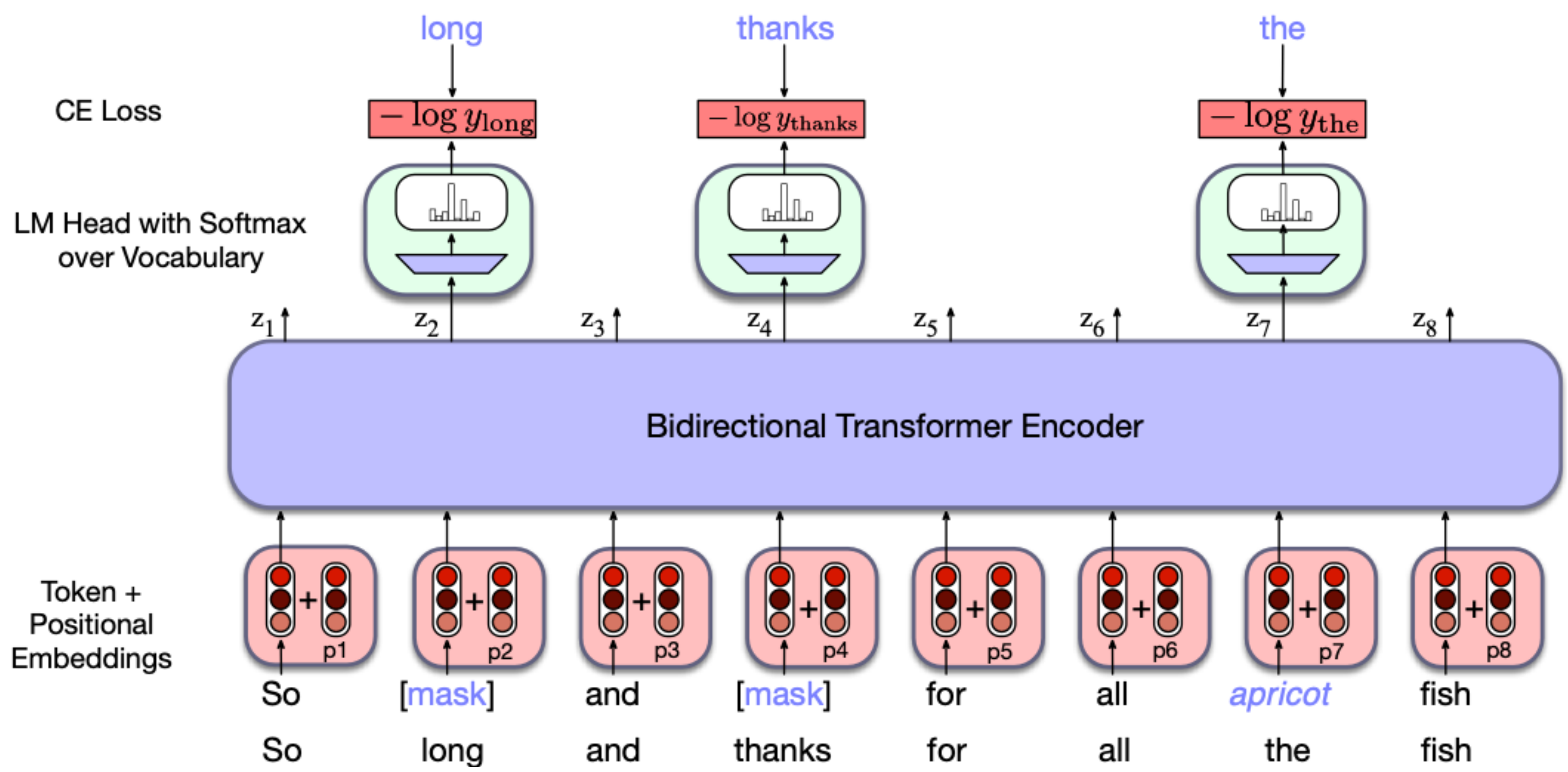
  The water of Walden Pond is so beautifully ______

  - And we train it to improve its predictions.
- For **bidirectional masked LMs**, the model tries to predict one or more missing words from all the rest of the words:

  The ______ of Walden Pond ______ so beautifully blue

  - The model generates a probability distribution over the vocabulary for each missing token
  - We use the cross-entropy loss from each of the model's predictions to drive the learning process.

Bidirectional Transformer $\sim$ BERT



MLM training in BERT

15% of the tokens are randomly chosen to be part of the masking

Example: "Lunch was **delicious**", if delicious was randomly chosen:

Three possibilities:

1. 80%: Token is replaced with special token [MASK]

    Lunch was **delicious ->** Lunch was **[MASK]**

2. 10%: Token is replaced with a random token (sampled from unigram prob)

    Lunch was **delicious ->** Lunch was **gasp**

3. 10%: Token is unchanged

    Lunch was **delicious ->** Lunch was **delicious**

MLM loss

The LM head takes output of final transformer layer L, multiplies it by unembedding layer and turns into probabilities:

$$\mathbf{u}_i = E\mathbf{h}_i^L \qquad\qquad \mathbf{y}_i = \text{softmax}(\mathbf{u}_i)$$

E.g., for the x_i corresponding to "long", the loss is the probability of the correct word *long*, given output h^L_i):

$$\mathscr{L}_{\text{MLM}}(x_i) = -\log P(x_i \mid \mathbf{h}_i^L)$$

We get the gradients by taking the average of this loss over the batch

$$\mathscr{L}_{\text{MLM}} = -\frac{1}{|\mathscr{B}|}\sum_{s\in\mathscr{B}}\frac{1}{|\mathscr{M}_s|}\sum_{i\in\mathscr{M}_s}\log P(x_i \mid \mathbf{h}_i^L)$$

Bidirectional Encoder Representations from Transformers

BERT (Devlin et al., 2019)

- 30,000 English-only tokens (WordPiece tokenizer)
- Input context window N=512 tokens, and model dimensionality d=768
- L=12 layers of transformer blocks, each with A=12 (bidirectional) multihead-attention layers.
- The resulting model has about 100M parameters.

XLM-RoBERTa (Conneau et al., 2020)

- 250,000 multilingual tokens (SentencePiece Unigram LM tokenizer)
- Input context window N=512 tokens,model dimensionality d=1024
- L=24 layers of transformer blocks, with A=16 multihead attention layers each
- The resulting model has about 550M parameters.

[15 minute break]

Implementing the Transformer!

Team up!

Open exercises/week 8 in your course folder and start writing/running code!