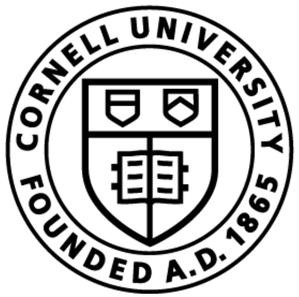
Lecture 15: Transformers



Cornell Bowers CIS
Computer Science

Claire Cardie, Tanya Goyal

CS 4740 (and crosslists): Introduction to Natural Language Processing

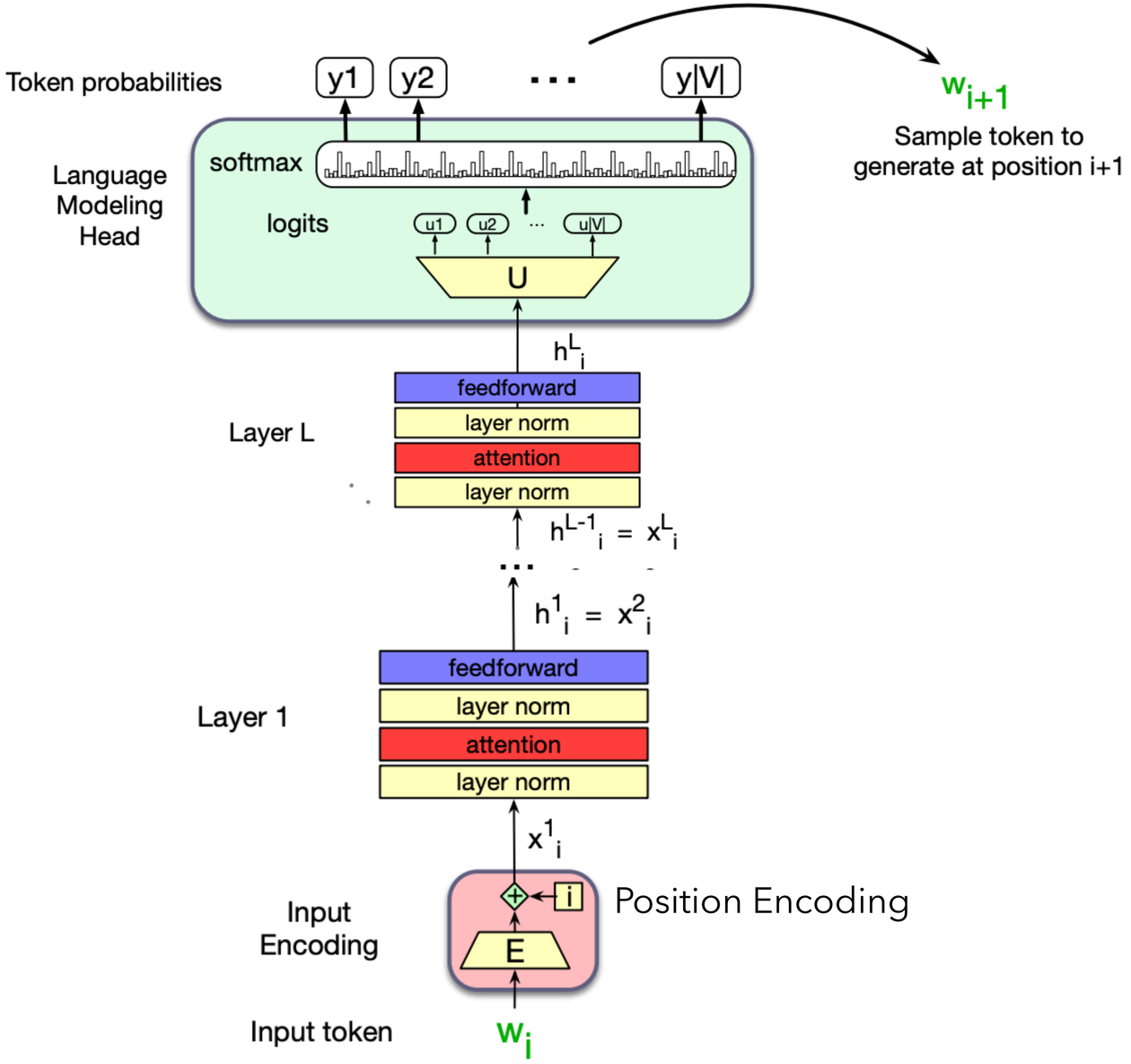
Announcements

- Midterm regrades are open (closes on Sun, 11.59 p.m.)
- HW2 due on Friday, 11.59 p.m.

Today

- Recap: Self-Attention in Transformers
- Transformers
  - Position Embedding
  - Layer Norm
  - Feed forward Layer
- [Maybe] Training Transformer-based Decoder only LMs

Transformer Architecture (Decoder-only)



- We will build up to this!!
- Main components of a transformer model
  - **(Multi-head) Attention**
  - Feed forward
  - Layer Norm
  - Position Encoding

Recap: Attention in Transformer models

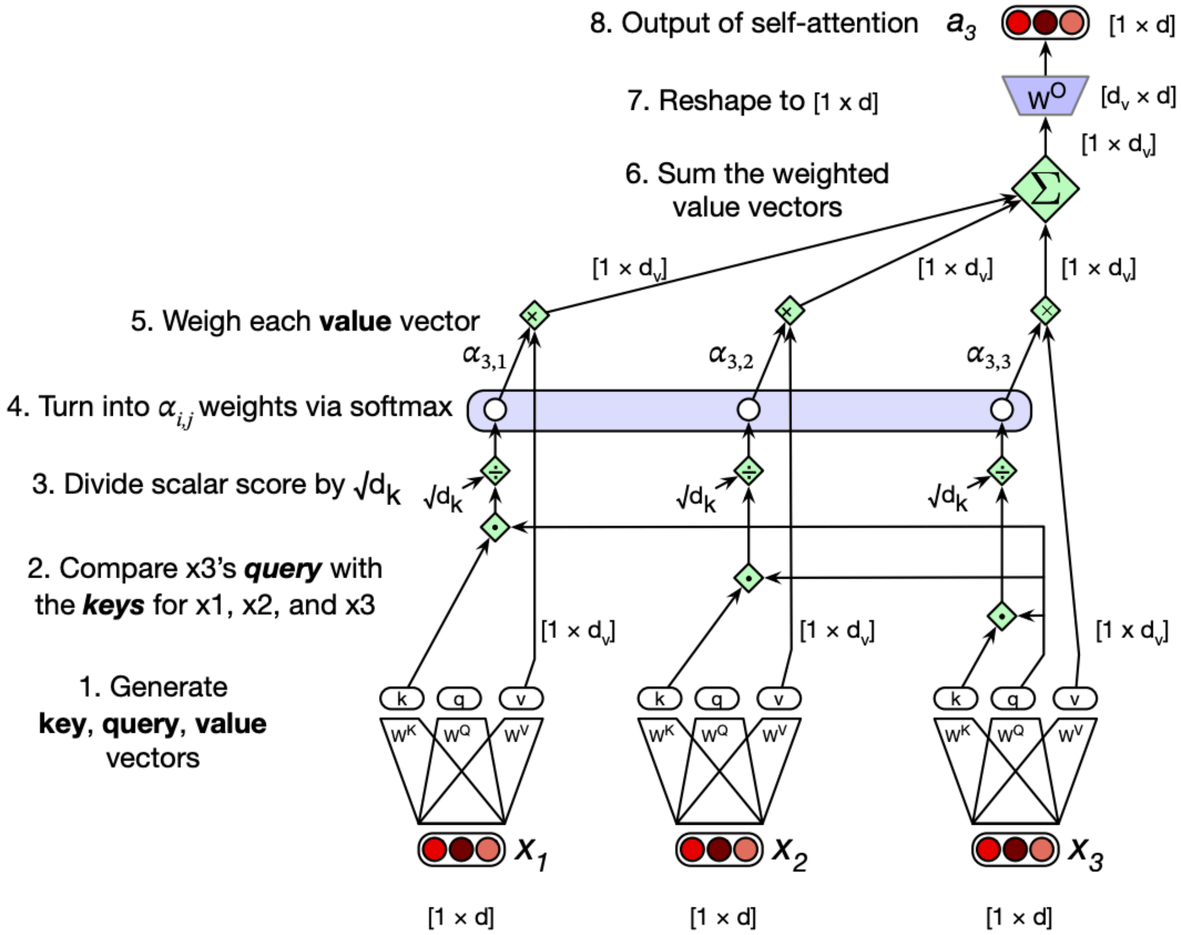


Figure 9.4 Calculating the value of $\mathbf{a}_3$, the third element of a sequence using causal (left-to-right) self-attention.

Recap: Attention in Transformer models

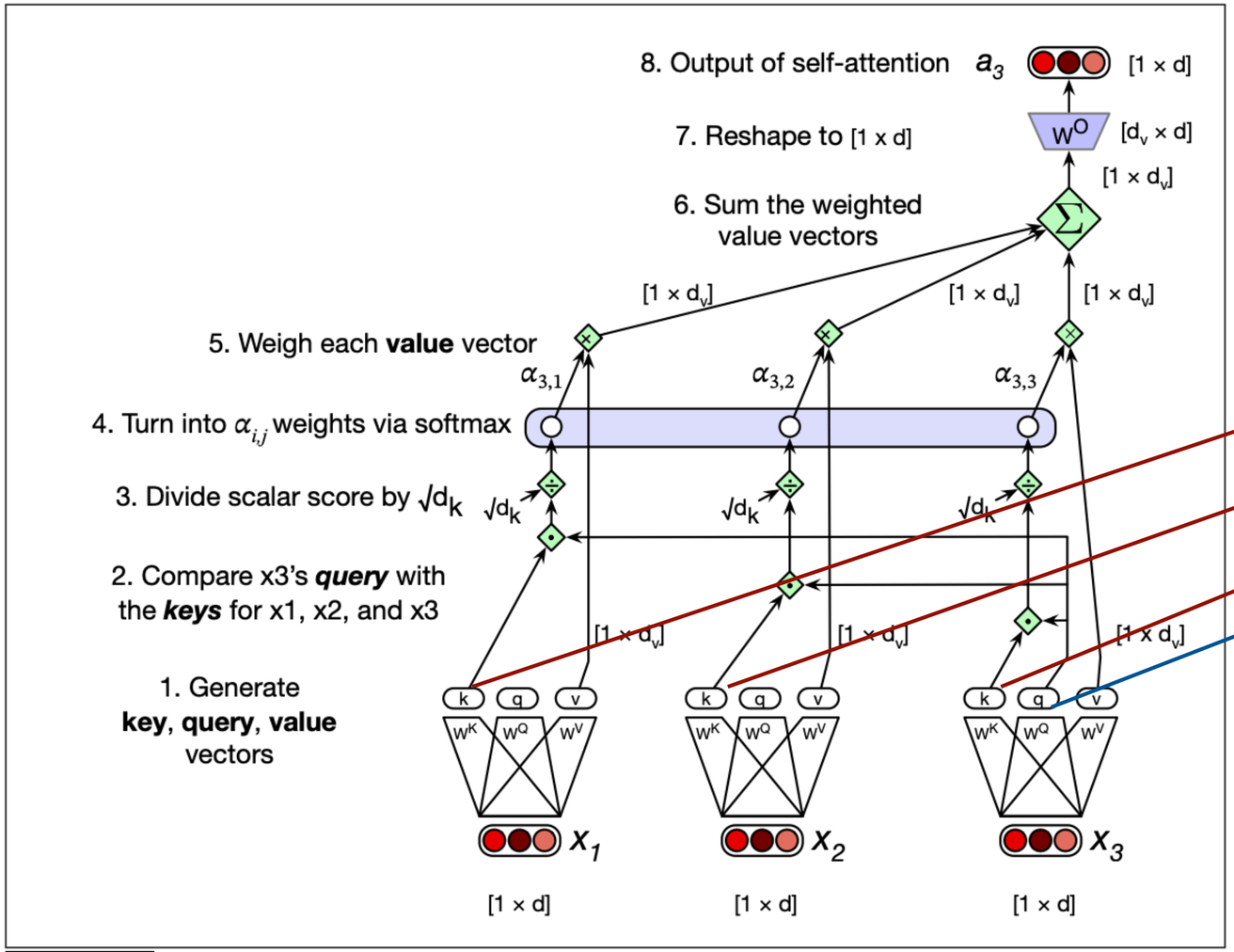
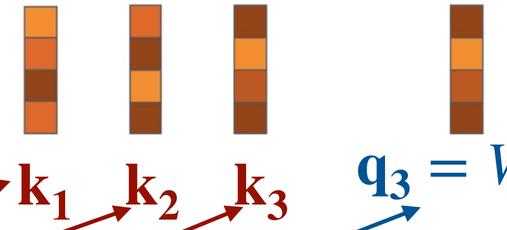


Figure 9.4 Calculating the value of $\mathbf{a}_3$, the third element of a sequence using causal (left-to-right) self-attention.

Computation at time step 3, ie. $\mathbf{a}_3$

Step1: prepare inputs



$\mathbf{k}_1$ $\mathbf{k}_2$ $\mathbf{k}_3$ $\mathbf{q}_3 = W^Q\mathbf{x}_3$

Recap: Attention in Transformer models

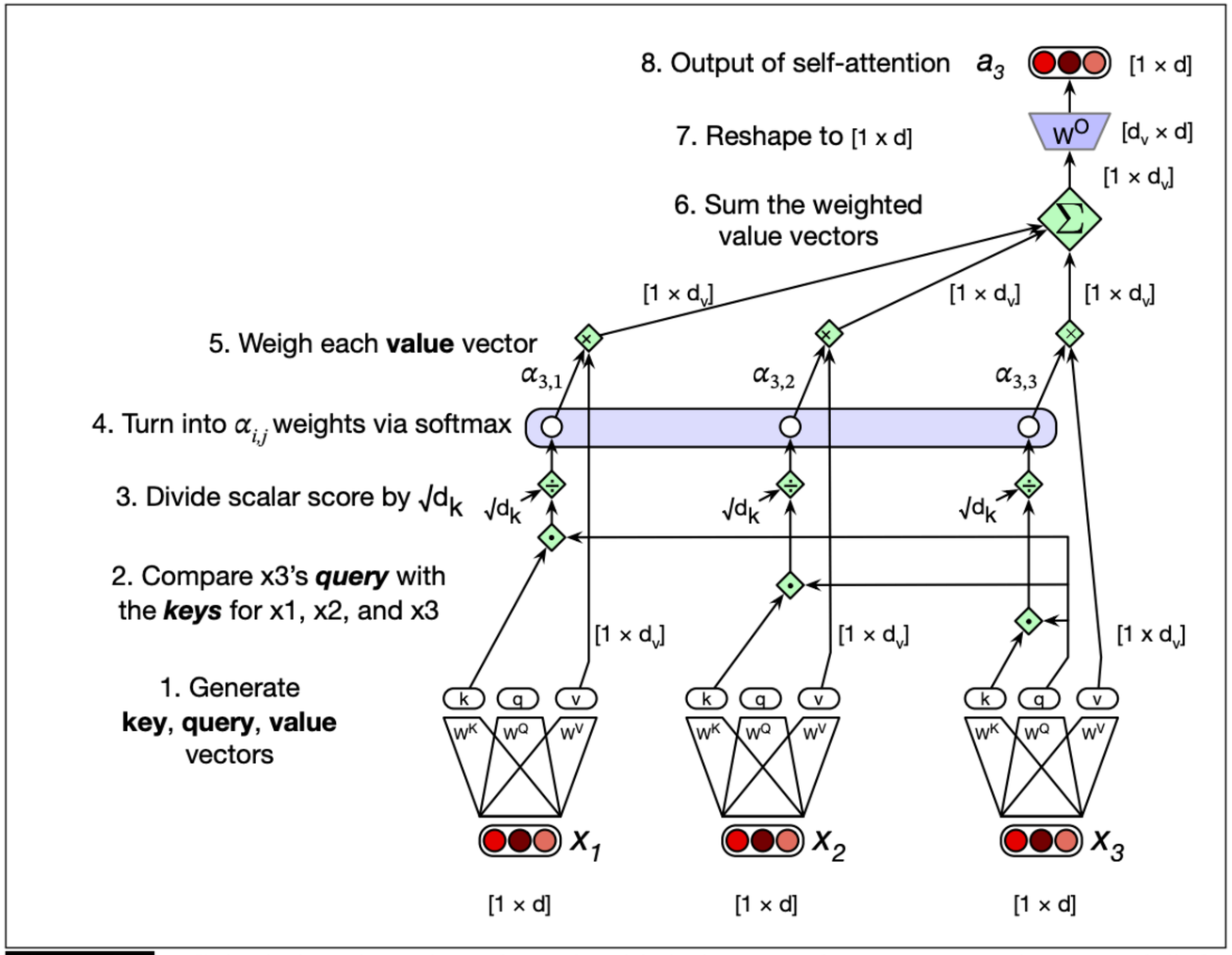
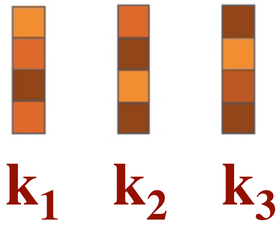


Figure 9.4 Calculating the value of $\mathbf{a}_3$, the third element of a sequence using causal (left-to-right) self-attention.

Computation at time step 3, ie. $\mathbf{a}_3$

Step1: prepare inputs



$\mathbf{k}_1$ $\mathbf{k}_2$ $\mathbf{k}_3$ $\mathbf{q}_3 = W^Q\mathbf{x}_3$

Step2: compute scores

9	9	13

$$\frac{\mathbf{q}_3 \cdot \mathbf{k}_1}{\sqrt{d_k}}$$

d_k is dim of query, key vectors

Step3: softmax scores (Attention weights)

0.02	0.02	0.96

Recap: Attention in Transformer models

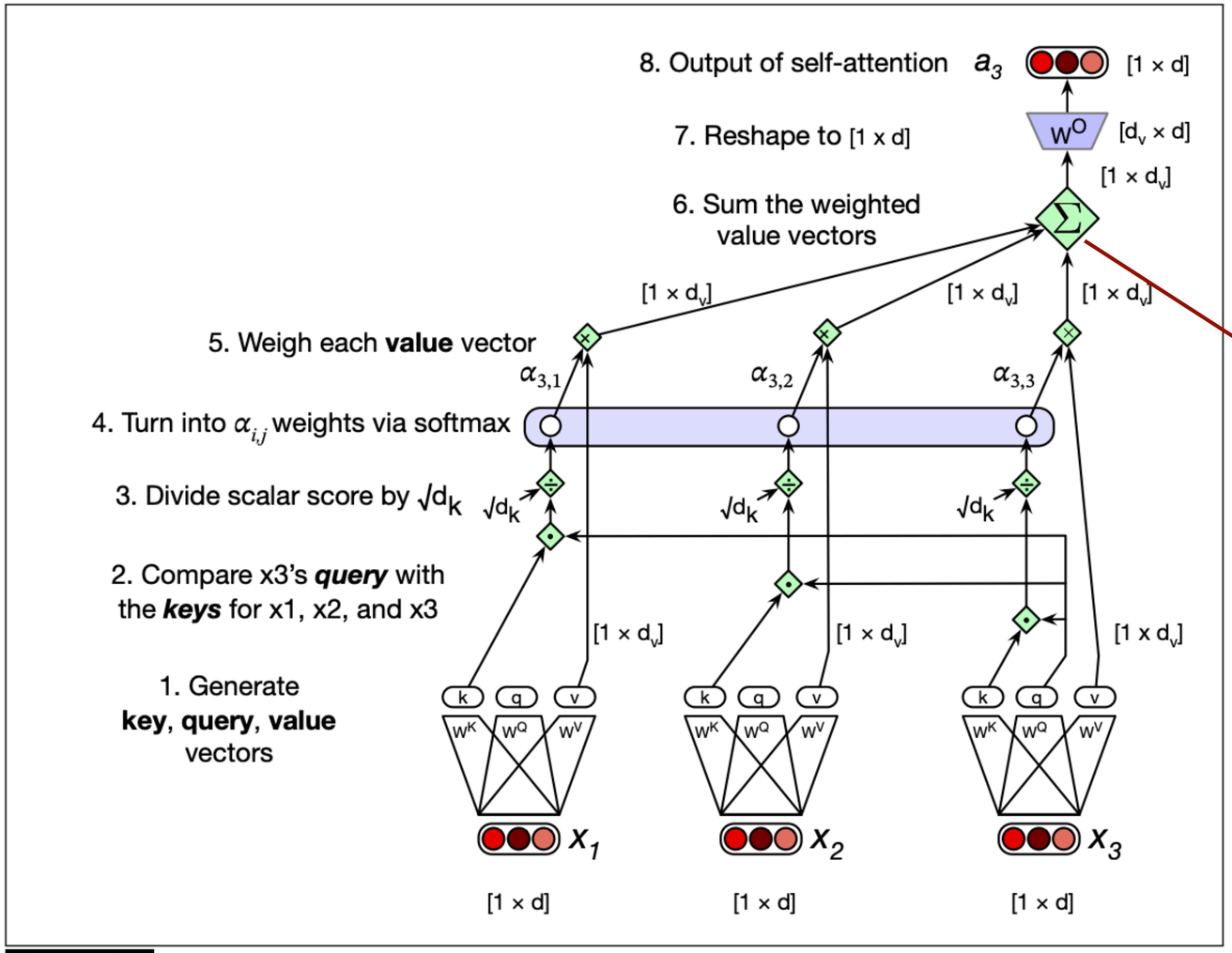
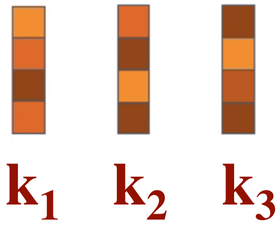


Figure 9.4 Calculating the value of $\mathbf{a}_3$, the third element of a sequence using causal (left-to-right) self-attention.

Computation at time step 3, ie. $\mathbf{a}_3$

Step1: prepare inputs



$\mathbf{k}_1$ $\mathbf{k}_2$ $\mathbf{k}_3$ $\mathbf{q}_3 = W^Q\mathbf{x}_3$

Step2: compute scores

9	9	13

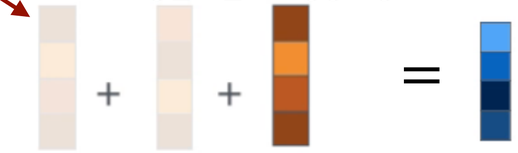
$$\frac{\mathbf{q}_3 \cdot \mathbf{k}_1}{\sqrt{d_k}}$$

Step3: softmax scores (Attention weights)

0.02	0.02	0.96

Step4: multiply each vector by softmax scores

$\alpha_{31}\mathbf{v}_1$ $\alpha_{32}\mathbf{v}_2$ $\alpha_{33}\mathbf{v}_3$



Attention in Transformer models

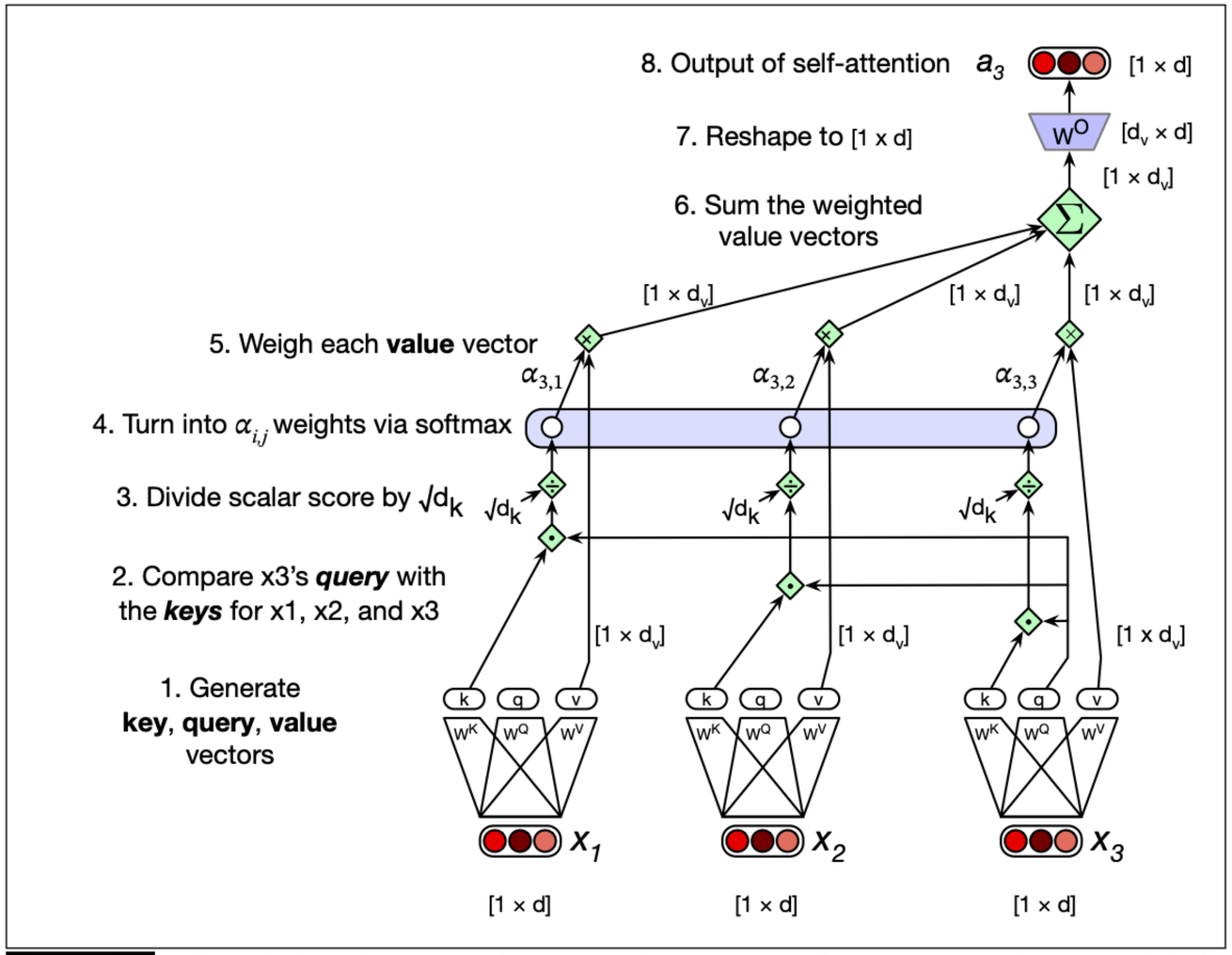
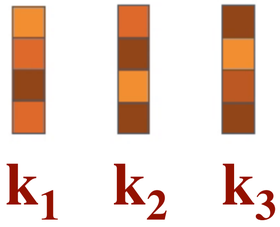


Figure 9.4 Calculating the value of $\mathbf{a}_3$, the third element of a sequence using causal (left-to-right) self-attention.

Computation at time step 3, ie. $\mathbf{a}_3$

Step1: prepare inputs



$\mathbf{q}_3 = W^Q\mathbf{x}_3$

Step2: compute scores

9	9	13

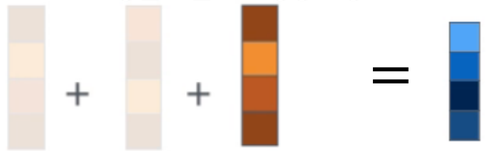
$$\frac{\mathbf{q}_3 \cdot \mathbf{k}_1}{\sqrt{d_k}}$$

Step3: softmax scores (Attention weights)

0.02	0.02	0.96

Step4: multiply each vector by softmax scores

$\alpha_{31}\mathbf{v}_1$ $\alpha_{32}\mathbf{v}_2$ $\alpha_{33}\mathbf{v}_3$



Step5: sum up the weighted vectors **and project**

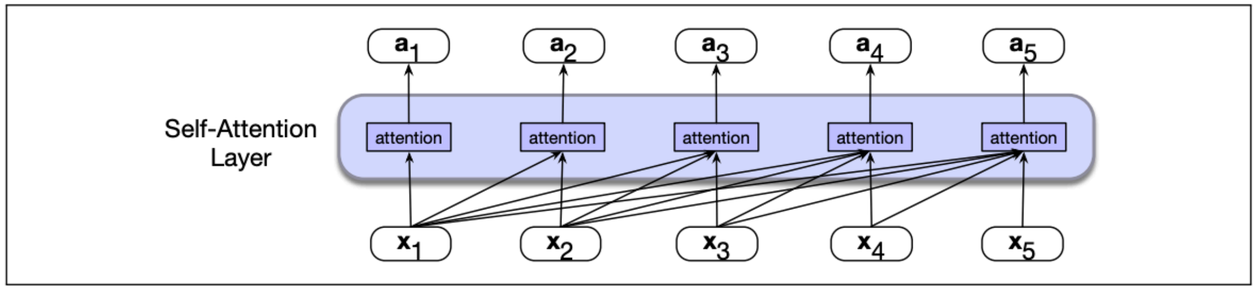
$W^O = \mathbf{a}_3$

Attention Visualization

BERTViz: https://github.com/jessevig/bertviz

[Caveat] BERT is an encoder, all tokens attend to all other tokens (no causal mask)

Zooming out

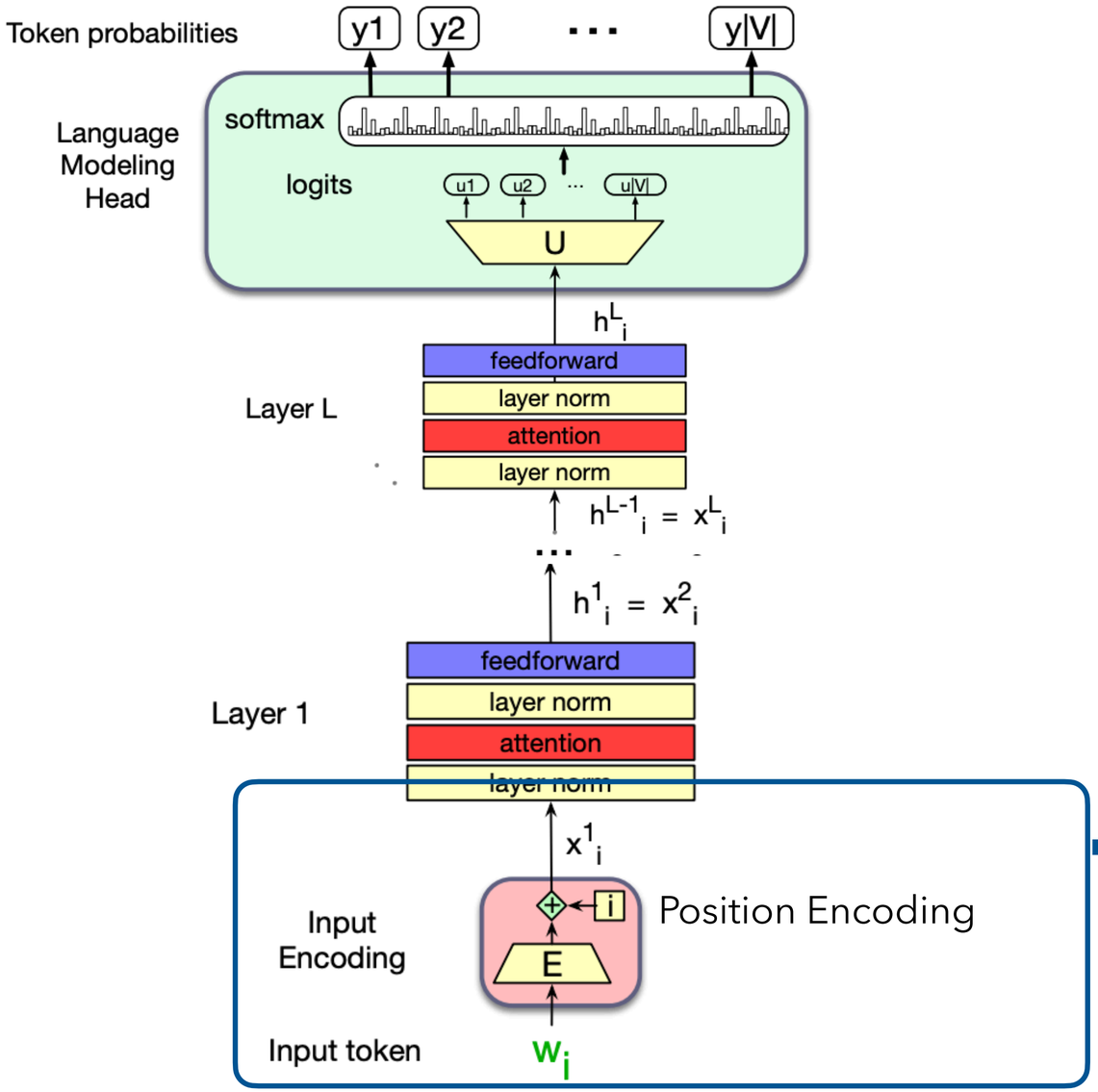


- Self-attention layer transformed the input $\mathbf{x_i}$ to output $\mathbf{a_i}$
- Word order information is lost!

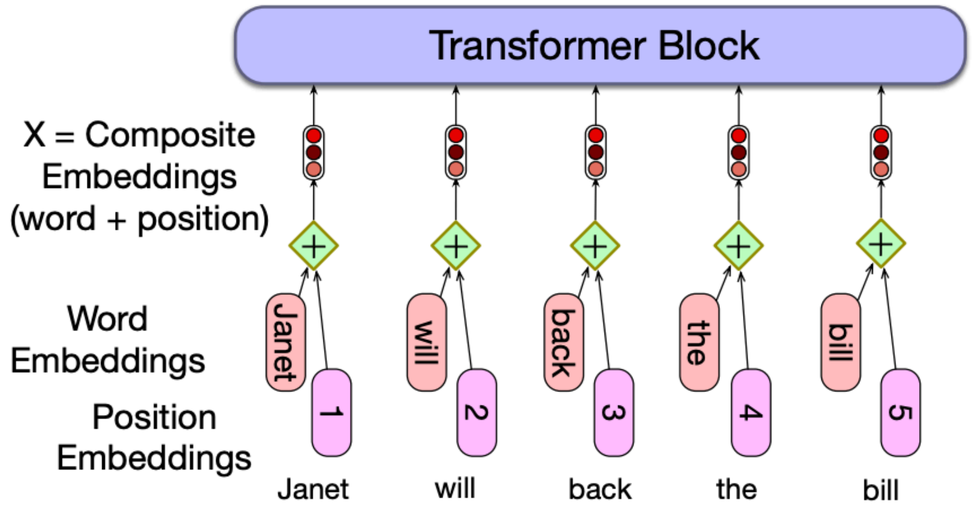
An old dog and a young boy ….

- *boy* attends to both *old* and *young*. We want *young* to have a higher influence on *boy*'s hidden representation than *old*. Attention does not ensure this.

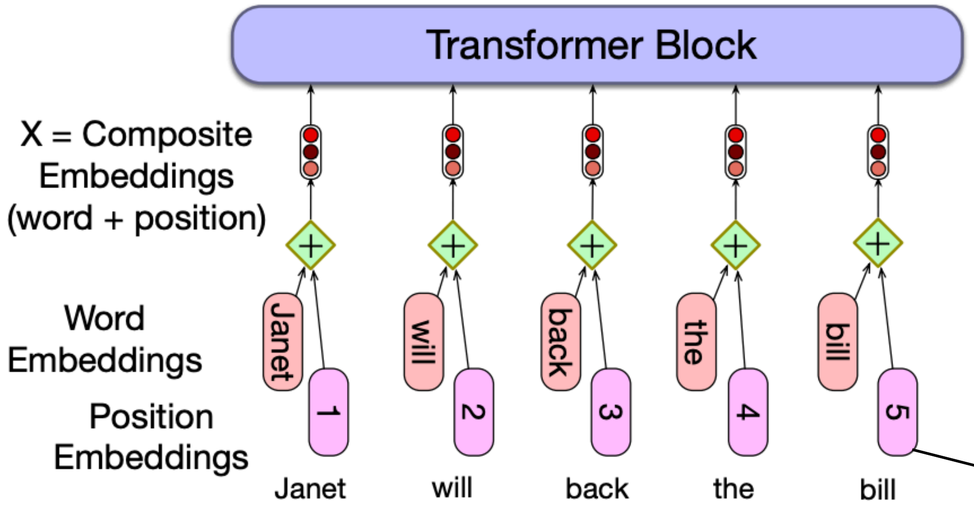
Position Embeddings



- Solution: Element-wise add a “position" embedding to the word embedding to produce a new embedding of the same dimension.



Position Embeddings



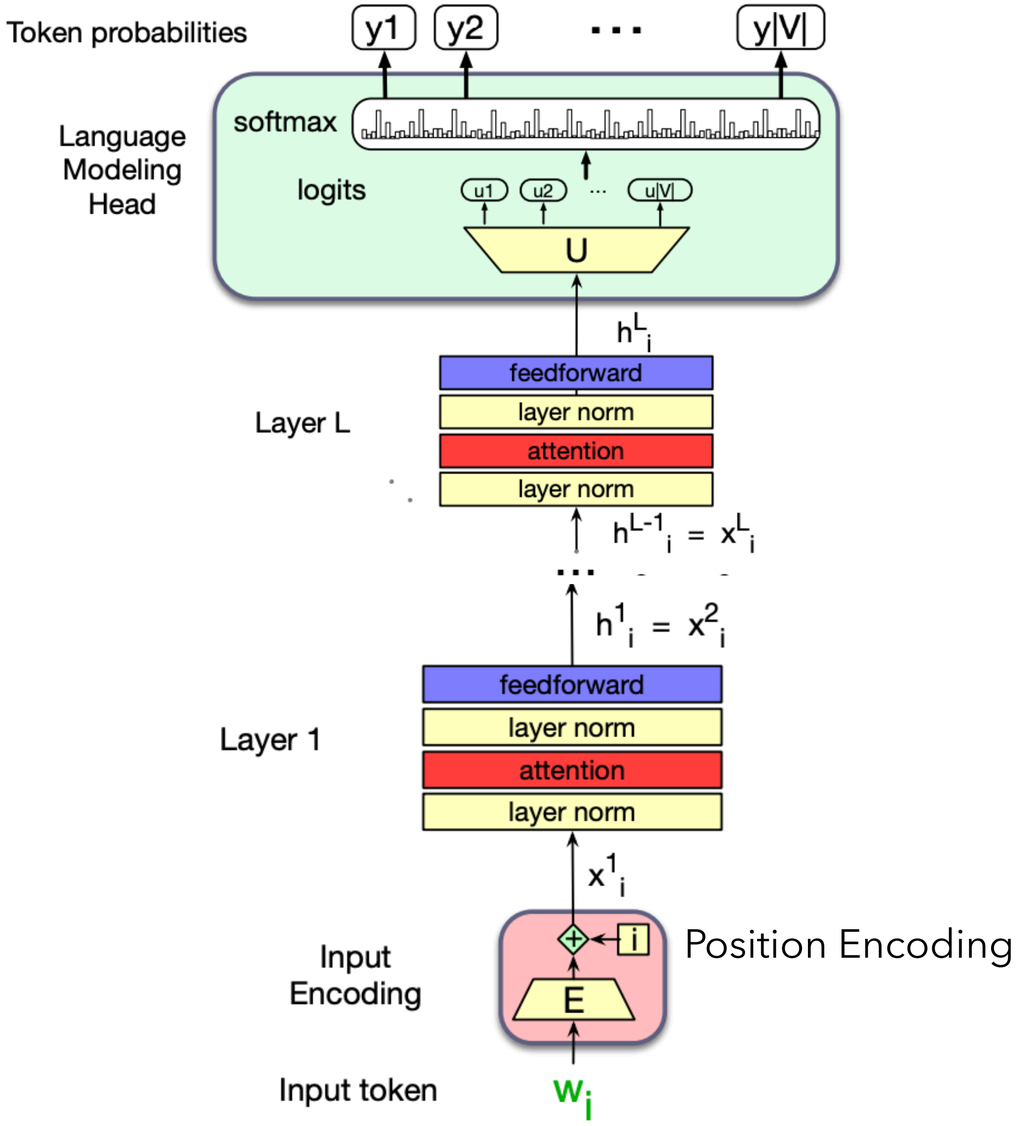
- Solution: Element-wise a "position" embedding to the word embedding to produce a new embedding of the same dimension.

Vectors of real numbers with the same size as word embeddings.

How do we get these positional embeddings?

- Assume all sequences will have length between 0 to N (say 512). Randomly initialize embeddings for each position.
- These will get trained with other transformer parameters.

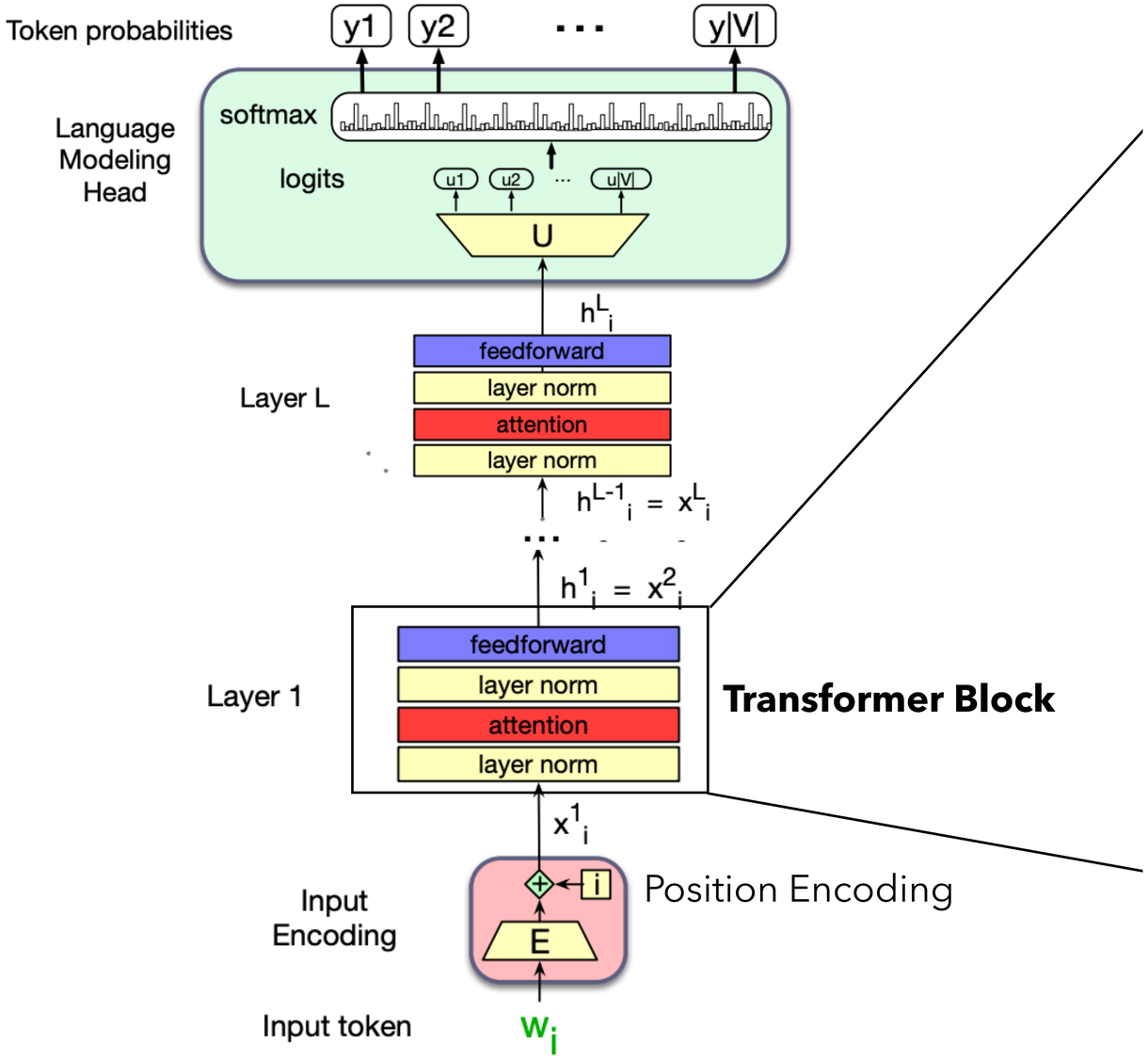
Let's go back to our transformer arch



- Last Lecture:
  - Multi-head self-attention
  - Position Embeddings

- This Lecture:
  - Residual connections
  - Layer Norm
  - Feedforward layer
  - Putting it all together
  - Encoder Decoder

Let's go back to our transformer arch



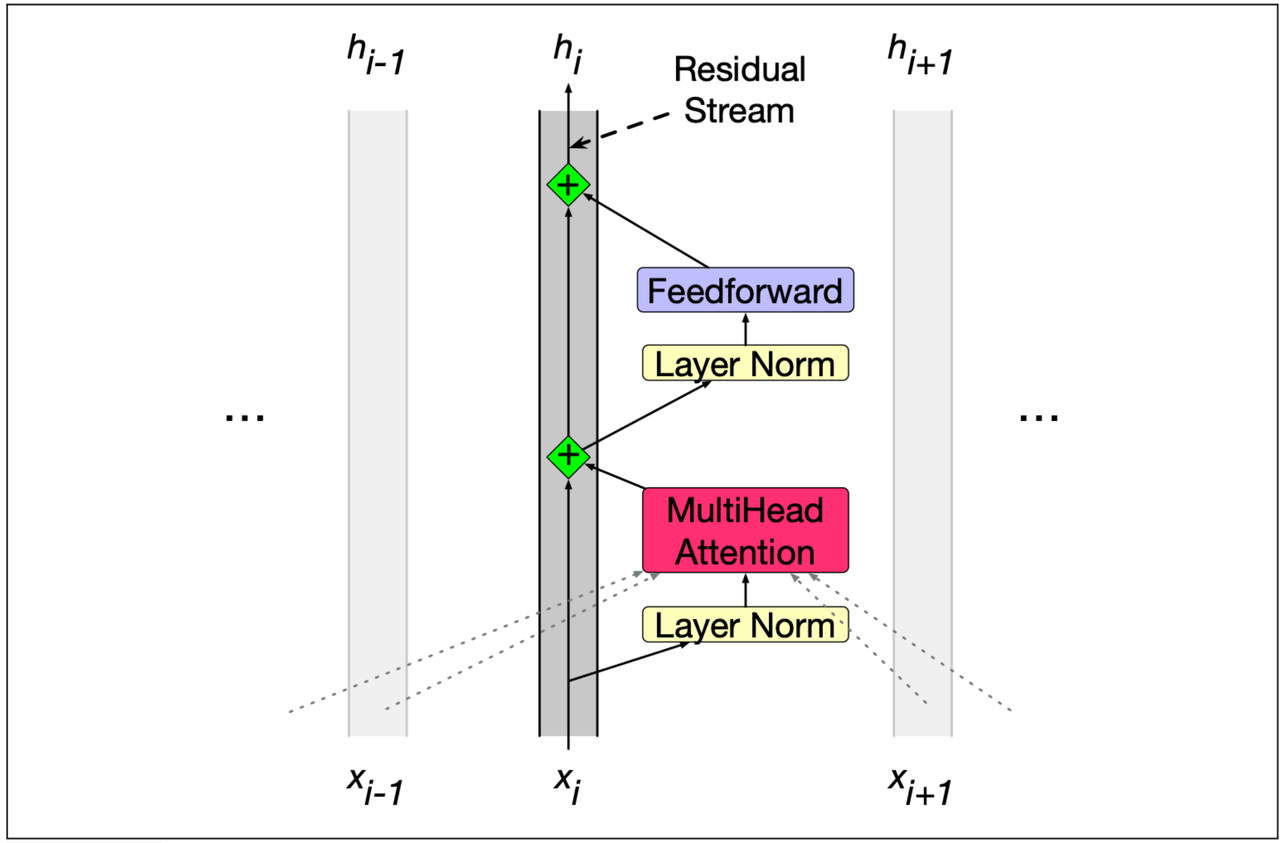


Figure 9.6 **Residual stream view of Transformer Block**

Residual Stream view

- Input x_i (d-dim vector)
  - Layer Norm + Multi-head Attention perform some computation, produce another d-dim vector.
  - Add to the residual stream $x_i \rightarrow x'_i$
- x'_i (d-dim vector)
  - Layer Norm + Feedforward produces another d-dim vector
  - Add to the residual stream $x'_i \rightarrow h_i$
- Output h_i

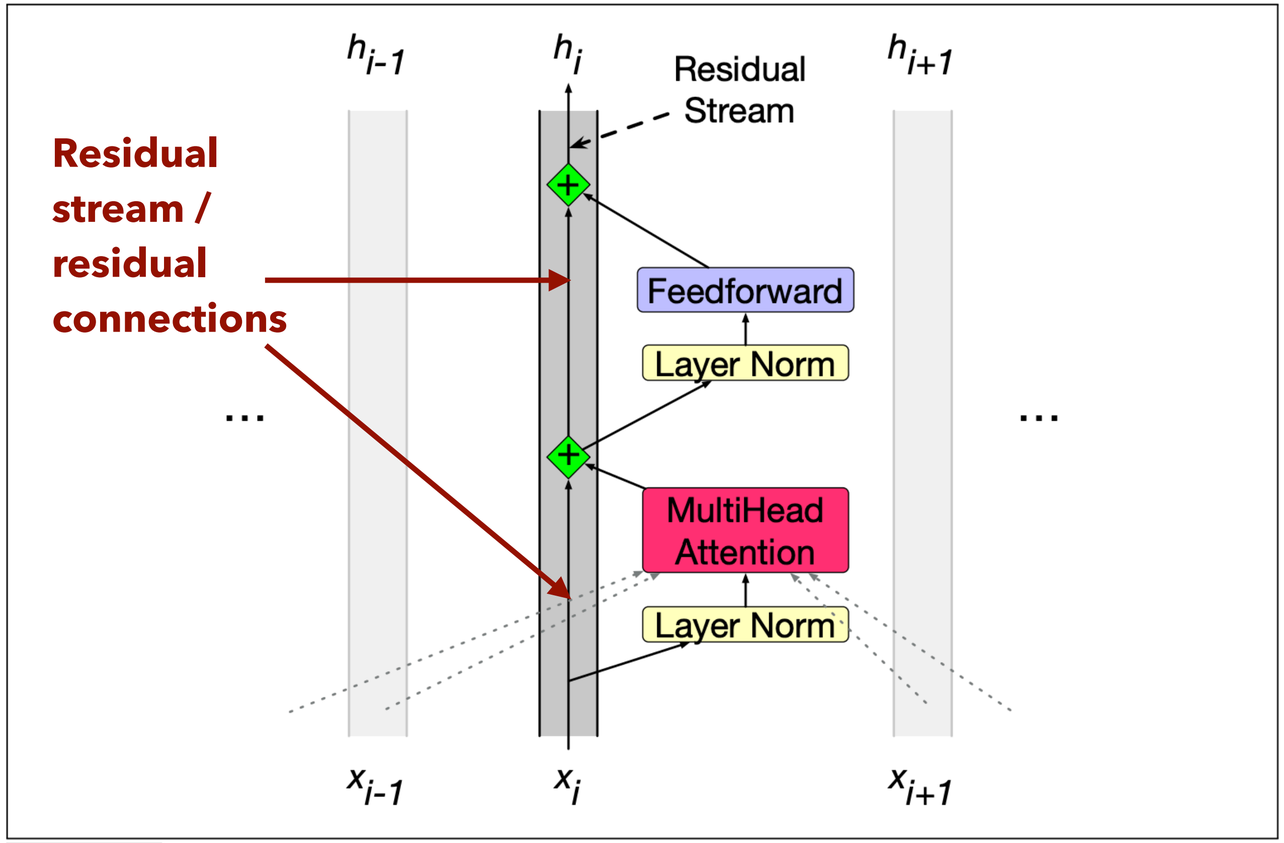


Figure 9.6 **Residual stream view of Transformer Block**

Feedforward layers

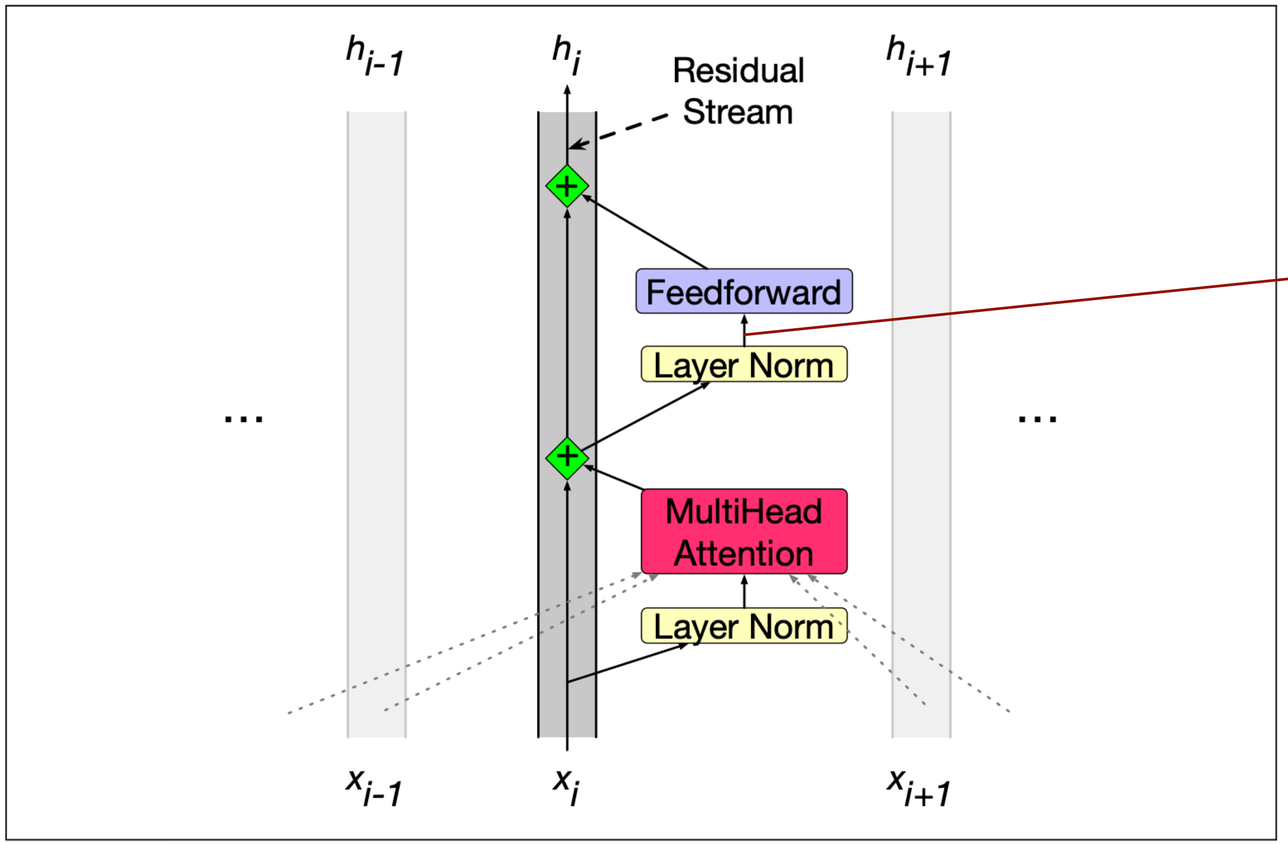


Figure 9.6 **Residual stream view of Transformer Block**

- Fully-connected 2-layer network

$$\text{FFNN}(x_i'') = \text{RELU}(x_i''\mathbf{W_1} + b_1)\mathbf{W_2} + b_2$$

Can we draw this?

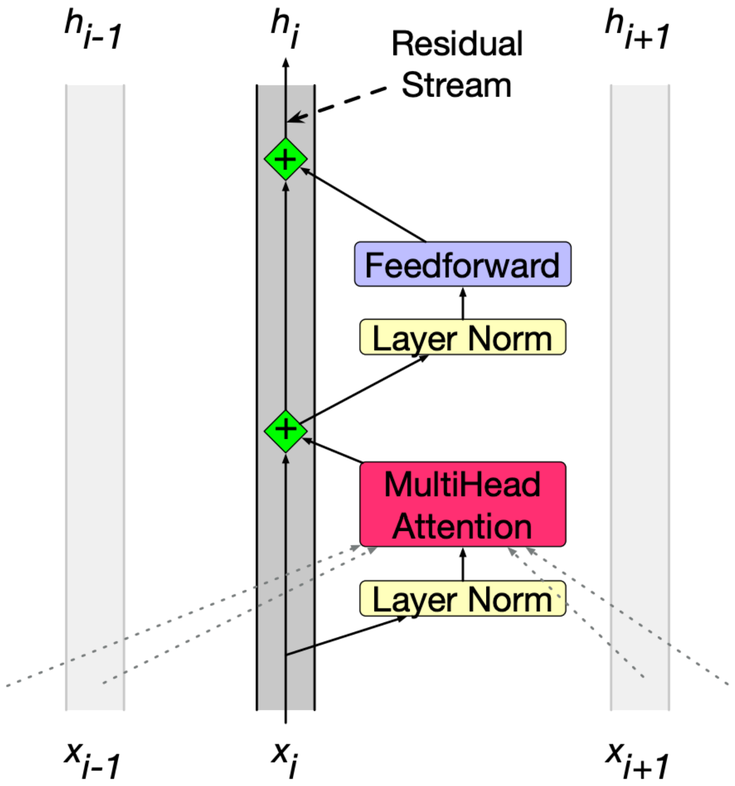
Layer Norm(alization)

- *Normalize* the input vector.
- **Not** applied to the entire transformer layer, applied to single token vectors in isolation.
- Given input a, $\mu = \frac{1}{d}\sum_{i=1}^{d} a_i \quad \sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(a_i - \mu)^2}$

$$\hat{a} = \frac{a - \mu}{\sigma}$$

$$\text{LayerNorm}(a) = \gamma\frac{(a - \mu)}{\sigma} + \beta \qquad \gamma, \beta \text{ are learnable parameters}$$

Transformer Block: Putting it all together



...

Input x_i at time step i

$$t_i^1 = \text{LayerNorm}(x_i)$$

$$t_i^2 = \text{MultiHead-Attention}(t_i^1, [t_1^1, t_2^1, \ldots t_N^1])$$

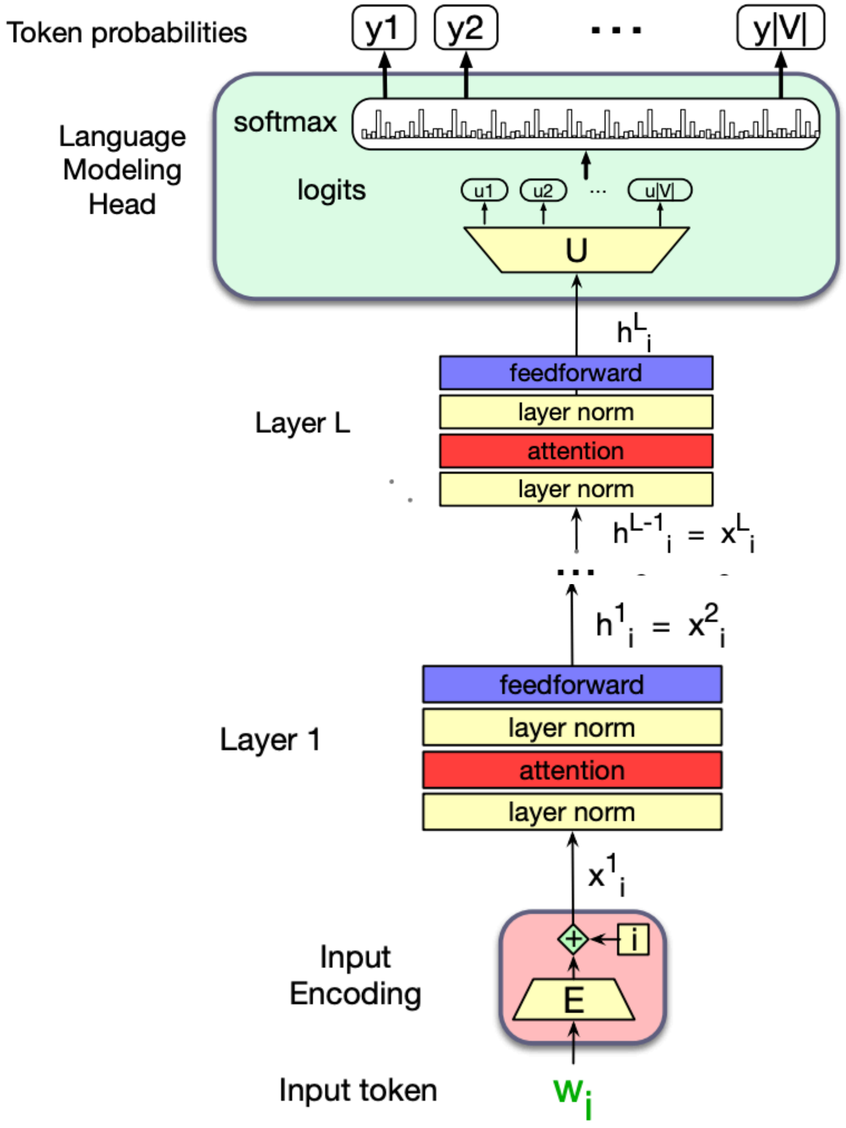
$$t_i^3 = t_i^2 + x_i$$

$$t_i^4 = \text{LayerNorm}(t_i^3)$$

$$t_i^5 = \text{FFNN}(t_i^4)$$

$$h_i = t_i^5 + t_i^3$$

Zooming out



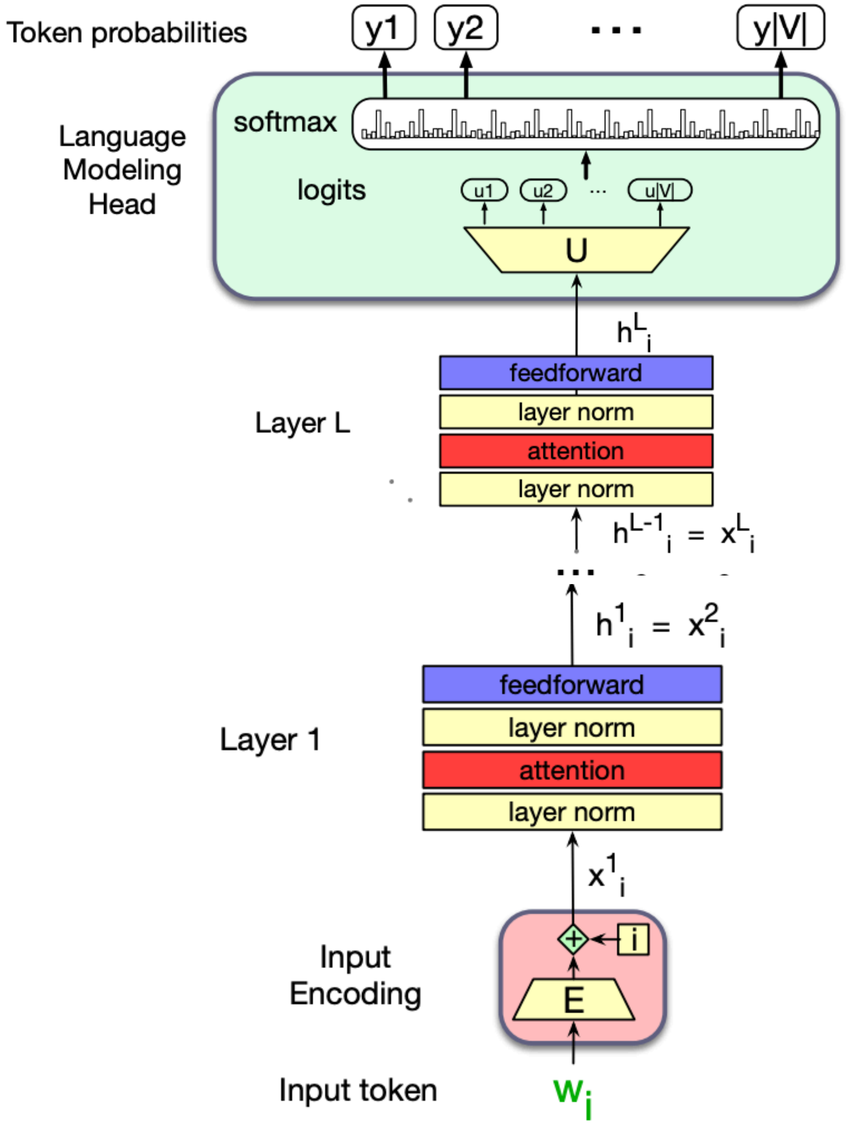
Same transformer blocks repeated N times.

- GPT2 was a family of 4 different models with the same architecture.

Parameters	Layers	d_{model}
117M	12	768
345M	24	1024
762M	36	1280
1542M	48	1600

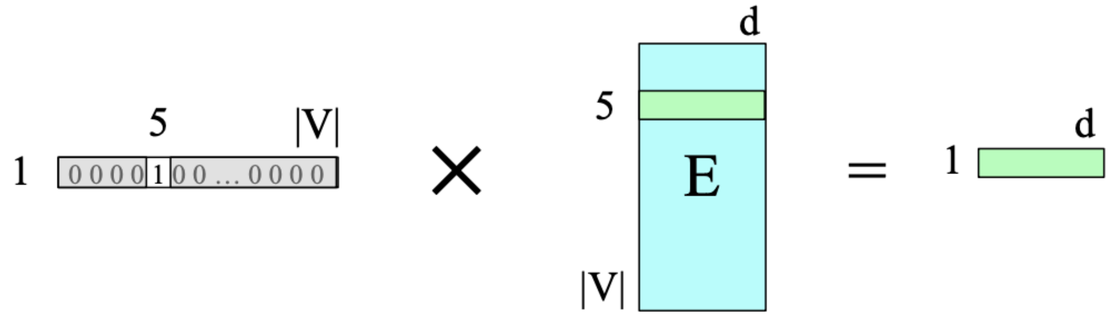
- Last piece: Input and Output Layers

Input Layer



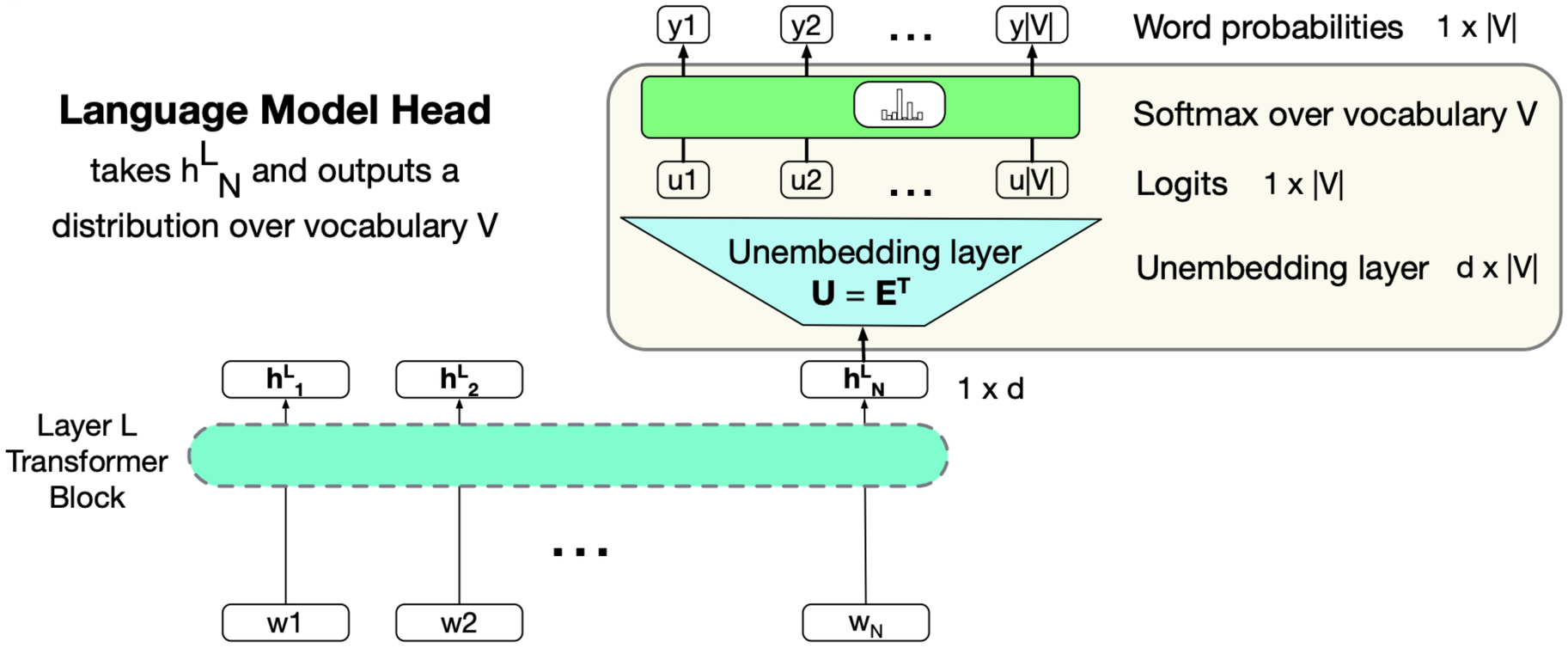
Input token w_i represented as a one-hot vector.

Matrix-multiplication with the Embedding matrix E gives d-dim vector representation of w_i



Output Layer

- Remember: We are studying a decoder-only Language Model.
- Training objective: Predict the next token, given preceding tokens.
- Final output: probability distribution over the vocabulary. Next word sampled this distribution.



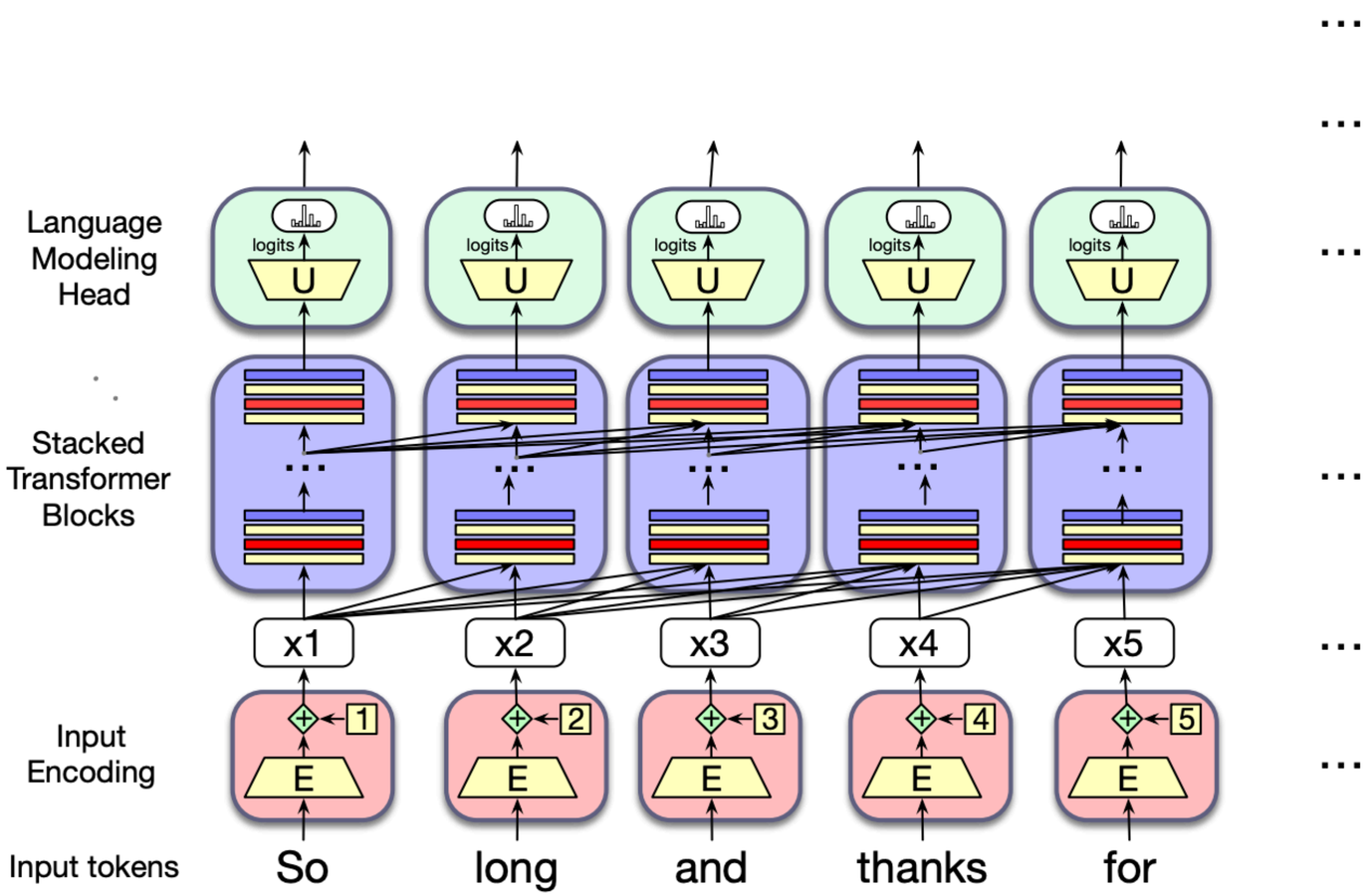
Language Modeling with Decoder-only Transformers

- Q: What is the Language Modeling Task?
  - Predict the next token given preceding tokens
  - $\hat{w}_i = \arg\max_{w \in V} P(w \mid w_1 \ldots w_{i-1})$

- Next Lecture:
  - Training a decoder-only Language Model
    - Cross Entropy Loss
  - Encoder Models
  - Encoder-Decoder Models

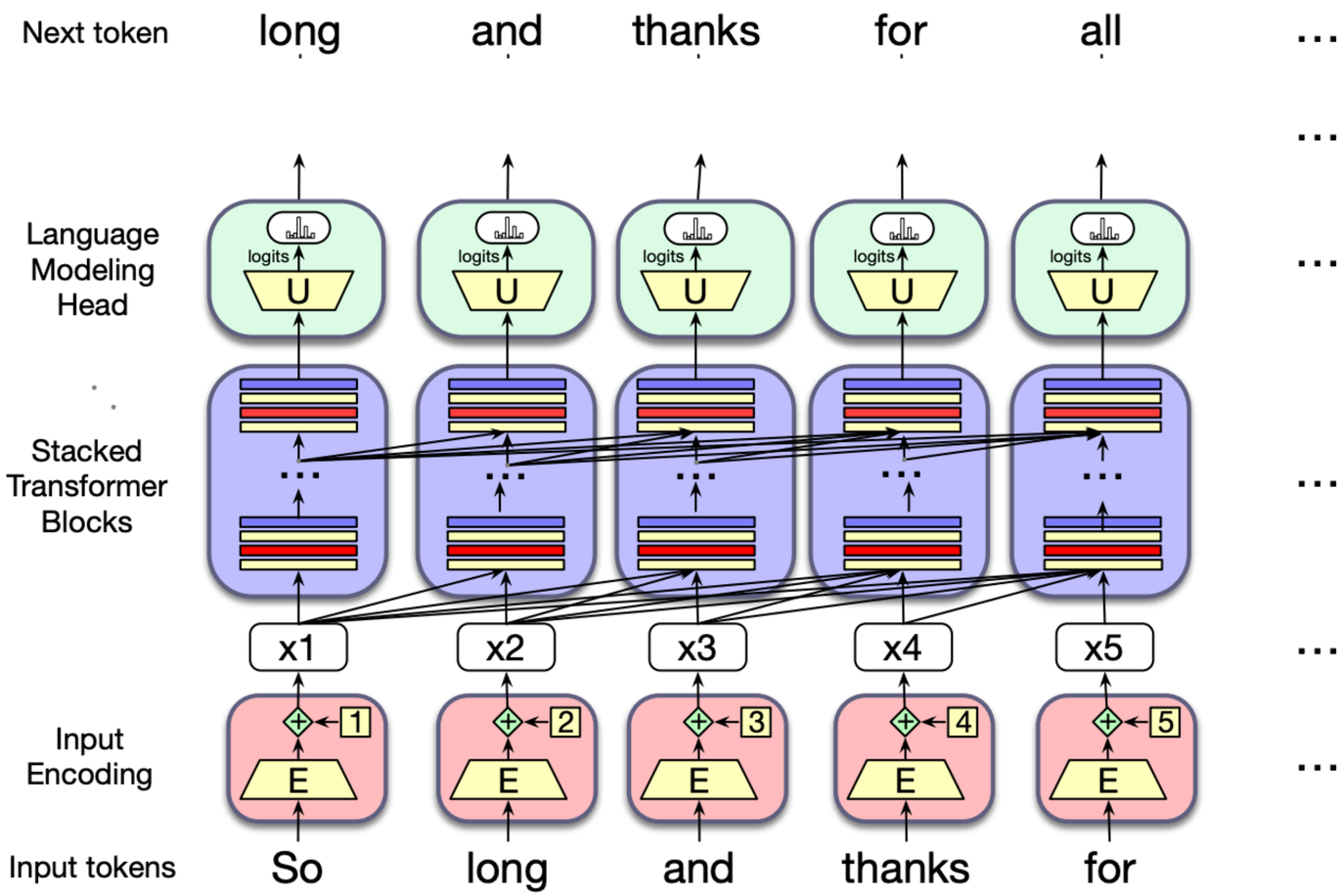
Language Modeling with Decoder-only Transformers

- Q: What should be the output at each step?



Language Modeling with Decoder-only Transformers

- Q: What should be the output at each step?
  - This is called **self-supervision** or **self-training** because no additional labels are used.



Language Modeling with Decoder-only Transformers

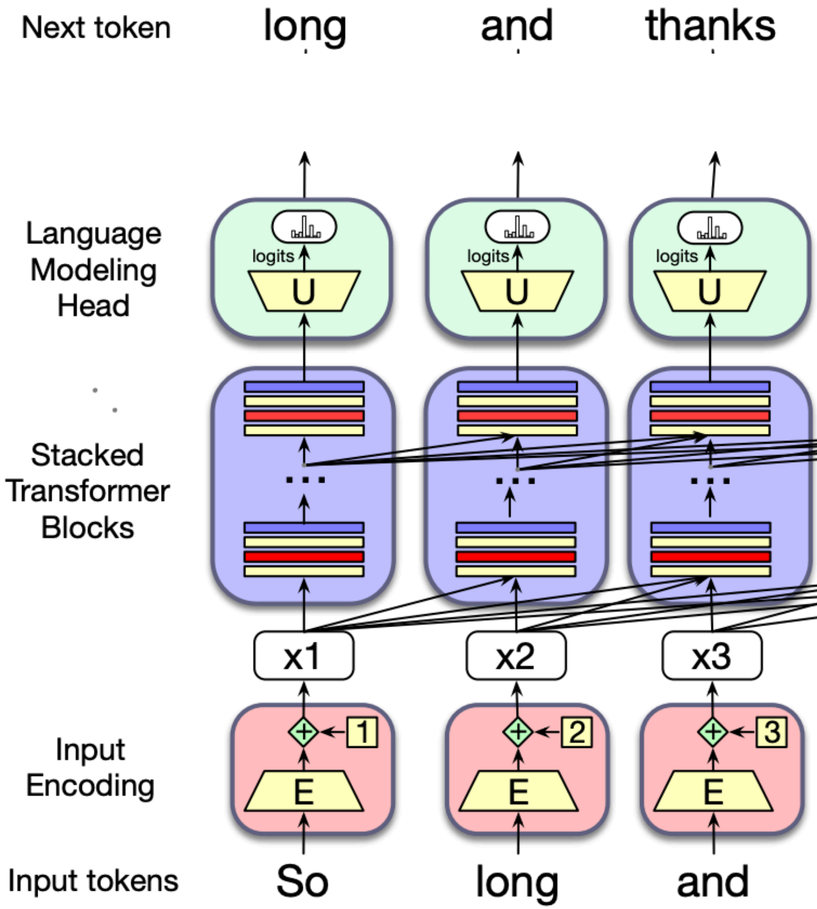
- Q: How should we model loss?
  - Cross Entropy Loss

$$L_{CE} = -\sum_{w \in V} \mathbf{y}_t[w] \log \hat{\mathbf{y}}_t[w]$$

$\mathbf{y}_t$ is the gold / desired output at time step t.
One-hot vector with index of gold token = 1, all else 0

$\hat{\mathbf{y}}_t$ is model predicted probabilities at time step t

[w] square brackets refer to index of the vector



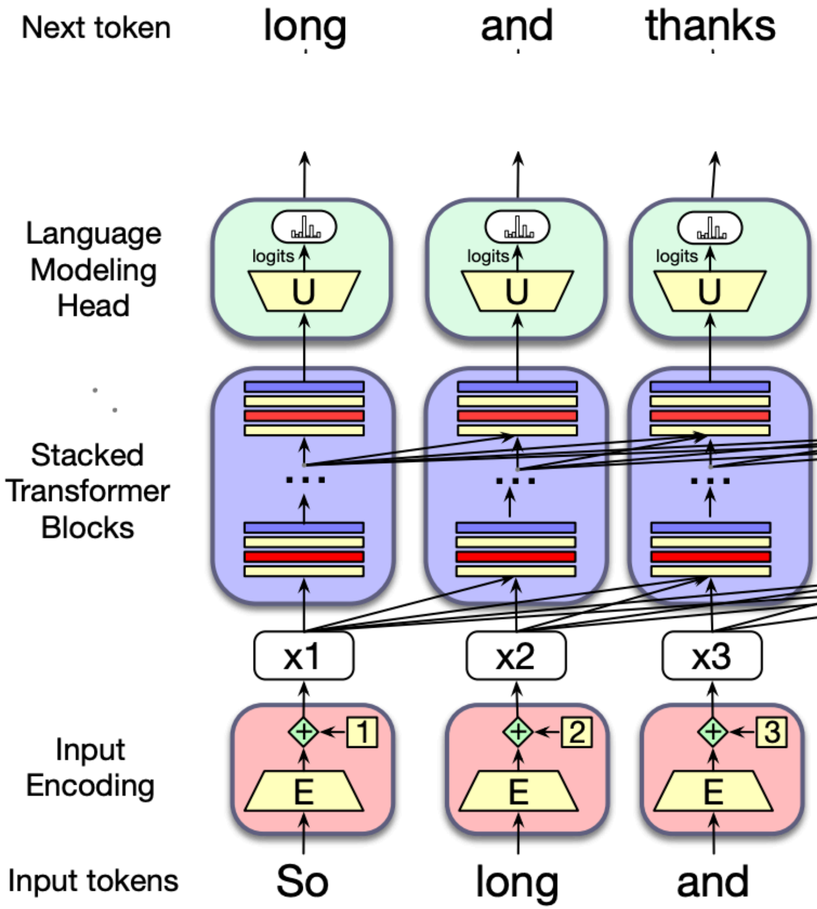
Language Modeling with Decoder-only Transformers

- Q: How should we model loss?
  - Cross Entropy Loss

$$L_{CE} = -\sum_{w \in V} \mathbf{y}_t[w] \log \hat{\mathbf{y}}_t[w]$$

  - What is $\mathbf{y}_t$ at time step t?
  - Simplifies to:

$$L_{CE} = -\log \hat{\mathbf{y}}_t[w_{t+1}]$$



Slide Acknowledgements

- Earlier versions of this course offerings including materials from Claire Cardie, Marten van Schijndel, Lillian Lee.