Transformer Language Models

Hongbo Li, Postdoc Scholar
Time: 2:00pm – 3:00pm, June 12, 2025

Slide Credits

1. *Generative AI*, 10-423/10-623, by Pat Virtue and Matt Gormley, Carnegie Mellon University: https://www.cs.cmu.edu/~mgormley/courses/10423/

2. *Speech and Language Processing*, by Dan Jurafsky and James H. Martin, Stanford University: https://web.stanford.edu/~jurafsky/slp3/

Index

- Part I: History of Large Language Models
- Part II: Attention Mechanism
- Part III: Transformer Language Models
- Part IV: Implementing a Transformer LM

Part I: History of LLMs

A Very Approximate Timeline

- 1990 Static Word Embeddings
- 2003 Neural Language Model
- 2008 Multi-Task Learning
- 2015 Attention
- 2017 Transformer
- 2018 Contextual Word Embeddings and Pretraining
- 2019 Prompting
- …

Large (n-Gram) Language Models

- The earliest (truly) large language models were Google n-grams:
  - 2006: first release, English n-grams
    - Trained on **1 trillion tokens** of web text (95 billion sentences)
    - Included 1-grams, 2-grams, 3-grams, 4-grams, and 5-grams
  - 2009-2010: n-grams in Japanese, Chinese, Swedish, Spanish, Romanian, Portuguese, Polish, Dutch, Italian, French, German, Czech

Number of unigrams:	13,588,391
Number of bigrams:	314,843,401
Number of trigrams:	977,069,902
Number of fourgrams:	1,313,818,354
Number of fivegrams:	1,176,470,663

English n-gram model is ~3 billion parameters

Q: Is this a large Training set?

Yes!

Q: Is this a large model?

Yes!

How Large are LLMs?

Model	Creators	Year of release	Training Data (# tokens)	Model Size (# parameters)
GPT-2	OpenAI	2019	~10 billion	1.5 billion
GPT-3	OpenAI	2020	300 billion	175 billion
LLaMA	Meta	2023	1.4 trillion	70 billion
LLaMA-2	Meta	2023	2 trillion	70 billion
GPT-4	OpenAI	2023	13 trillion	1.2 trillion
Gemini (Ultra)	Google	2023	3.6 trillion	? (1 trillion+)
DeepSeek-v1	DeepSeek	2024	2 trillion	67 billion
DeepSeek-v2	DeepSeek	2024	8.1 trillion	236 billion
LLaMA-3	Meta	2024	15 trillion	405 billion
DeepSeek-v3	DeepSeek	2024	14.8 trillion	671 billion
Gemini 2.5	Google	2025	?	?
GPT-4.5	OpenAI	2025	?	? (5-10 trillion)

Data source: Google

Part II: Attention Mechanism

Noisy Channel Models

- Prior to 2017, two tasks relied heavily on language models:
  - Speech recognition
  - Machine translation

- Definition: a **noisy channel model** combines a transduction model (probability of converting **y** to **x**) with a language model (probability of **y**)

$$\hat{y} = \arg\max_y p(y|x) = \arg\max_y \underbrace{p(x|y)}_{\text{transduction model}} \underbrace{p(y)}_{\text{language model}}$$

- Goal: to recover y from x
  - For speech: x is acoustic signal, y is transcription
  - For machine translation: x is sentence in source language, y is sentence in target language

Recurrent Neural Networks (RNNs)

Key idea:

1) Convert all previous words to a fixed length vector
2) Define distribution $p(w_t | f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $h_t = f_\theta(w_{t-1}, \ldots, w_1)$.

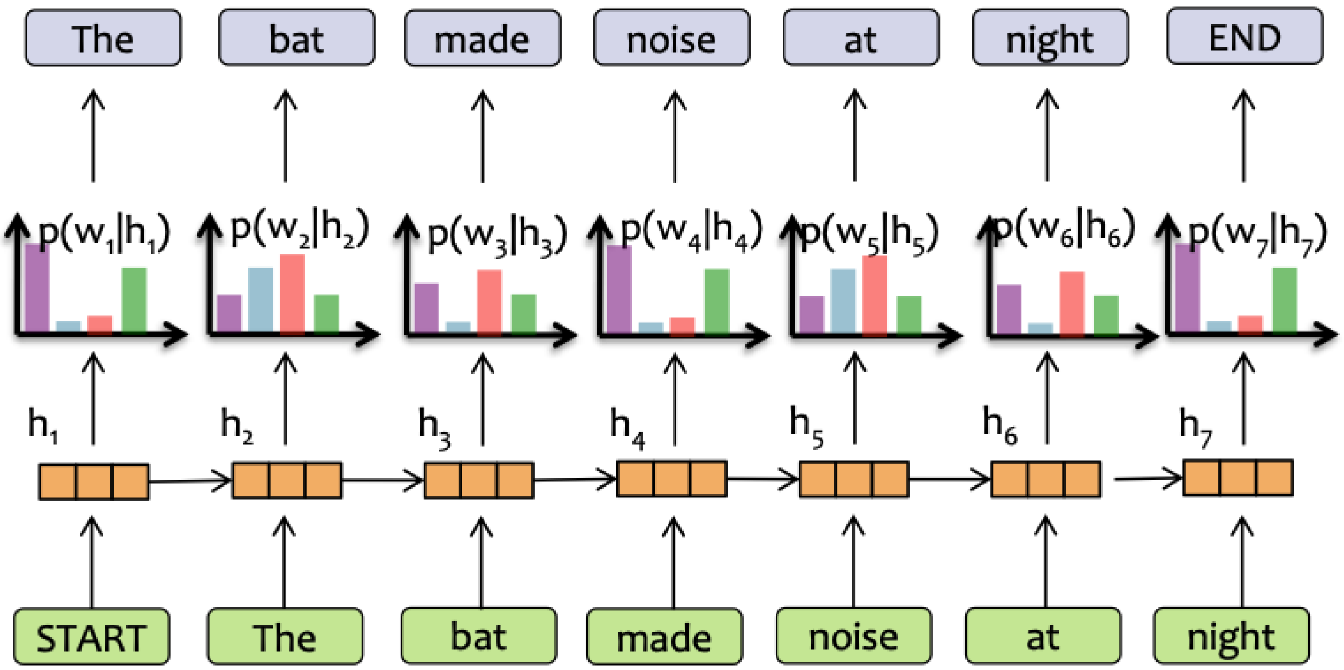


Figure from: Virtue and Gormley (2025)

Problem with Static Embeddings (word2vec)

They are static!

The embedding for a word doesn't reflect how its meaning changes in text.

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

What is the meaning represented in the static embedding for "it"?

Contextual Embeddings

- Intuition: a representation of meaning of a word should be different in different contexts!

- **Contextual Embedding**: each word has a different vector that expresses different meanings depending on the surrounding words

- How to compute contextual embeddings?
    - **Attention**

Contextual Embeddings

The chicken didn’t cross the road because it

What should be the properties of “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**

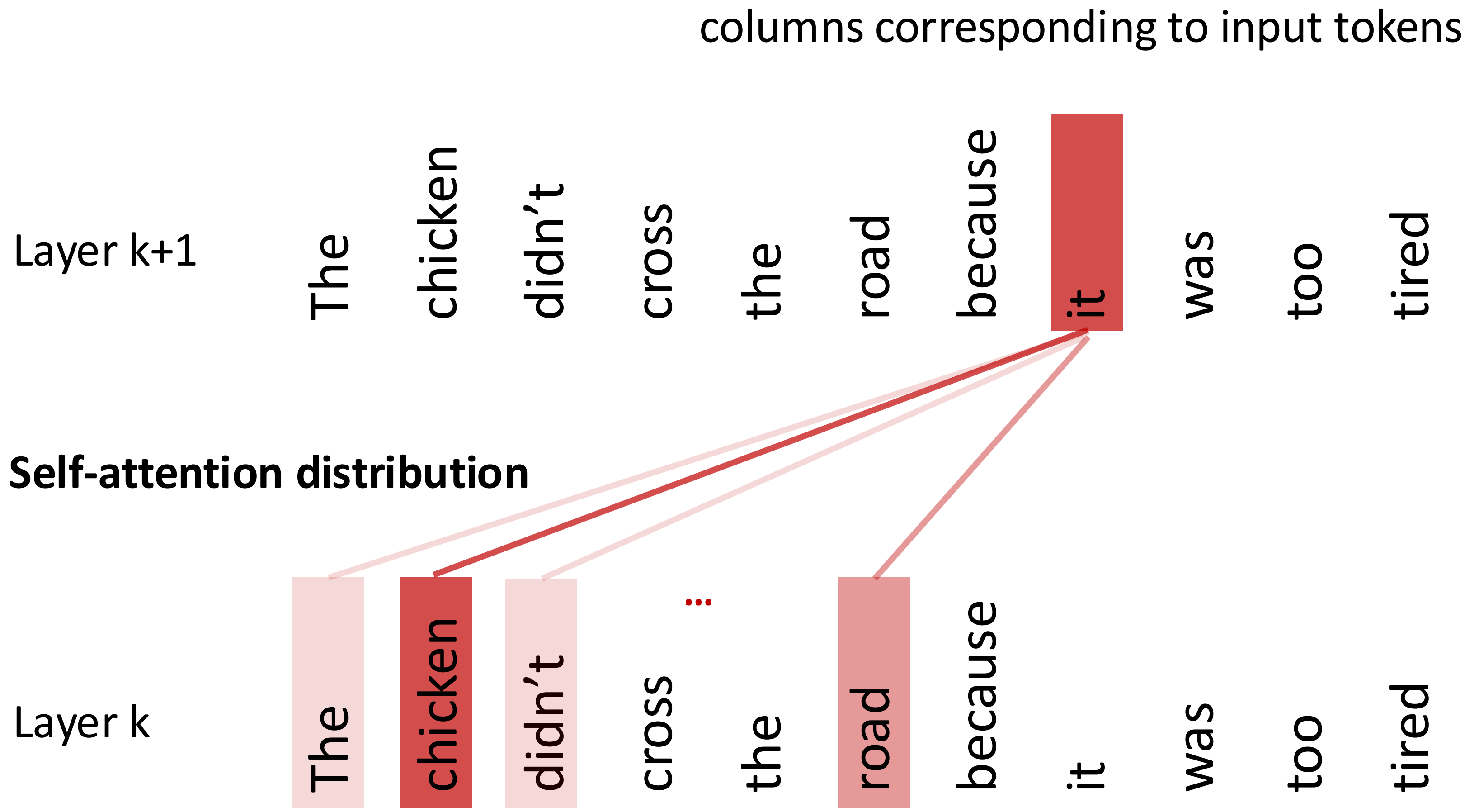
At this point in the sentence, it's probably referring to either the chicken or the street

Intuition of Attention

- Build up the contextual embedding from a word by selectively integrating information from all the neighboring words

- We say that a word "attends to" some neighboring words more than others

Intuition of Attention

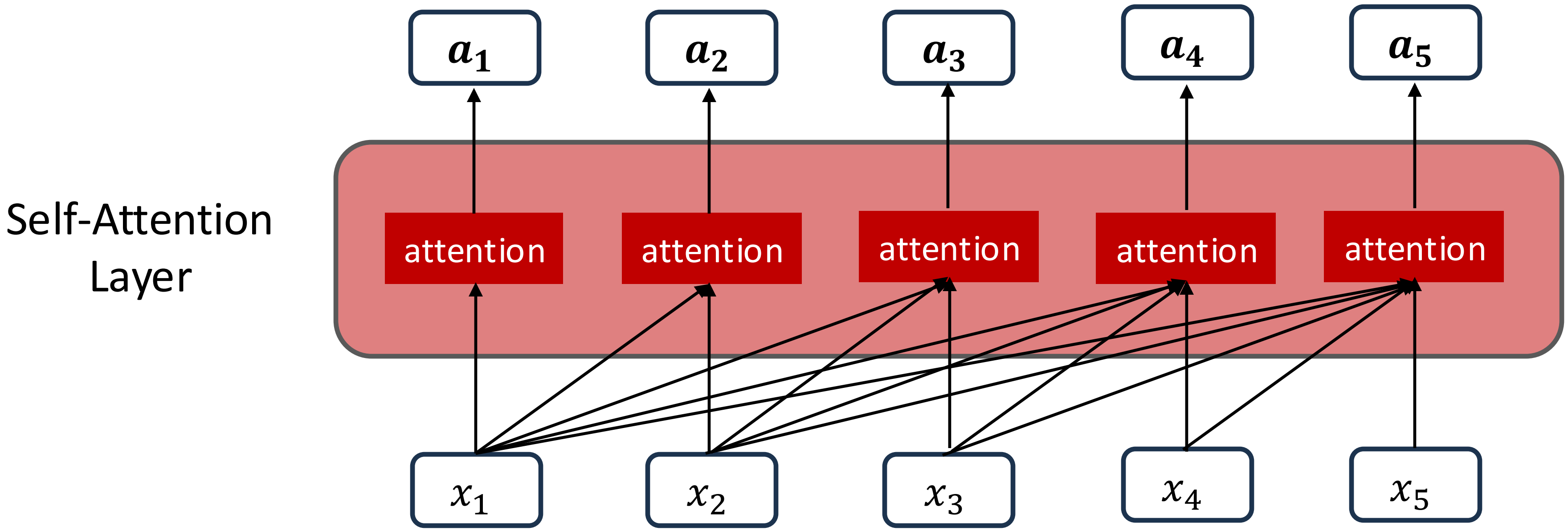


Attention Definition

- 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.

Attention is Left-to-Right



Simplified Version of Attention

- Given a sequence of token embeddings:
  - $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_i$
- Produce: a_i =a weighted sum of x_1 through x_7 (and x_i)
- Weighted by their **similarity to $\boldsymbol{x_i}$**

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

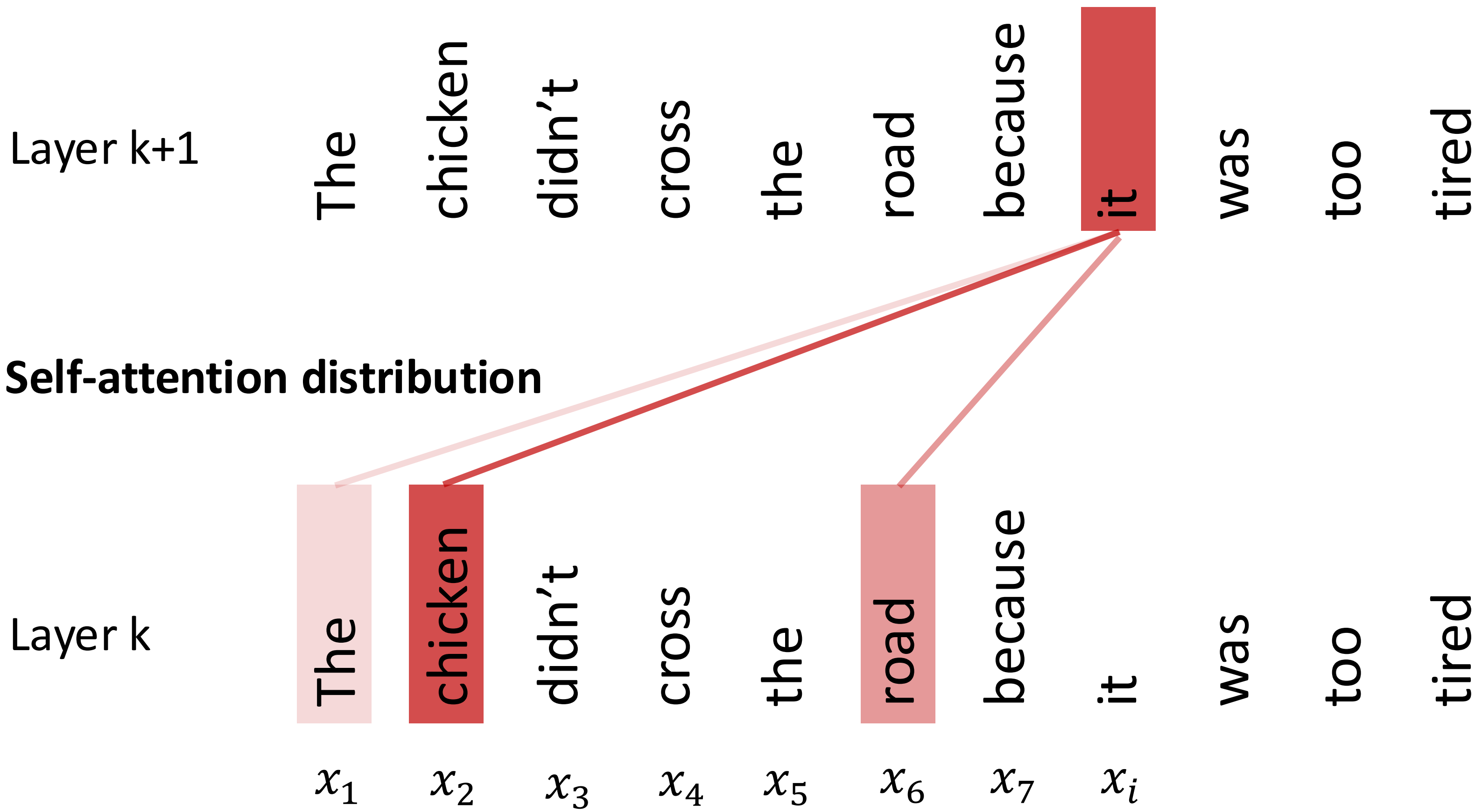
$$\alpha_{ij} = \text{softmax}\left(\text{score}(x_i, x_j)\right), \ \forall j \leq i$$

$$\boldsymbol{a_i} = \sum_{j \leq i} \alpha_{ij} \, x_j$$

Given a vector $\mathbf{z} = [z_1, z_2, \ldots, z_n]$, the softmax is: $\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$.

This is applied **element-wise** to produce: $\text{softmax}(\boldsymbol{z}) = \left[\frac{e^{z_1}}{\sum_j e^{z_j}}, \frac{e^{z_2}}{\sum_j e^{z_j}}, \ldots, \frac{e^{z_n}}{\sum_j e^{z_j}}\right]$

Intuition of Attention



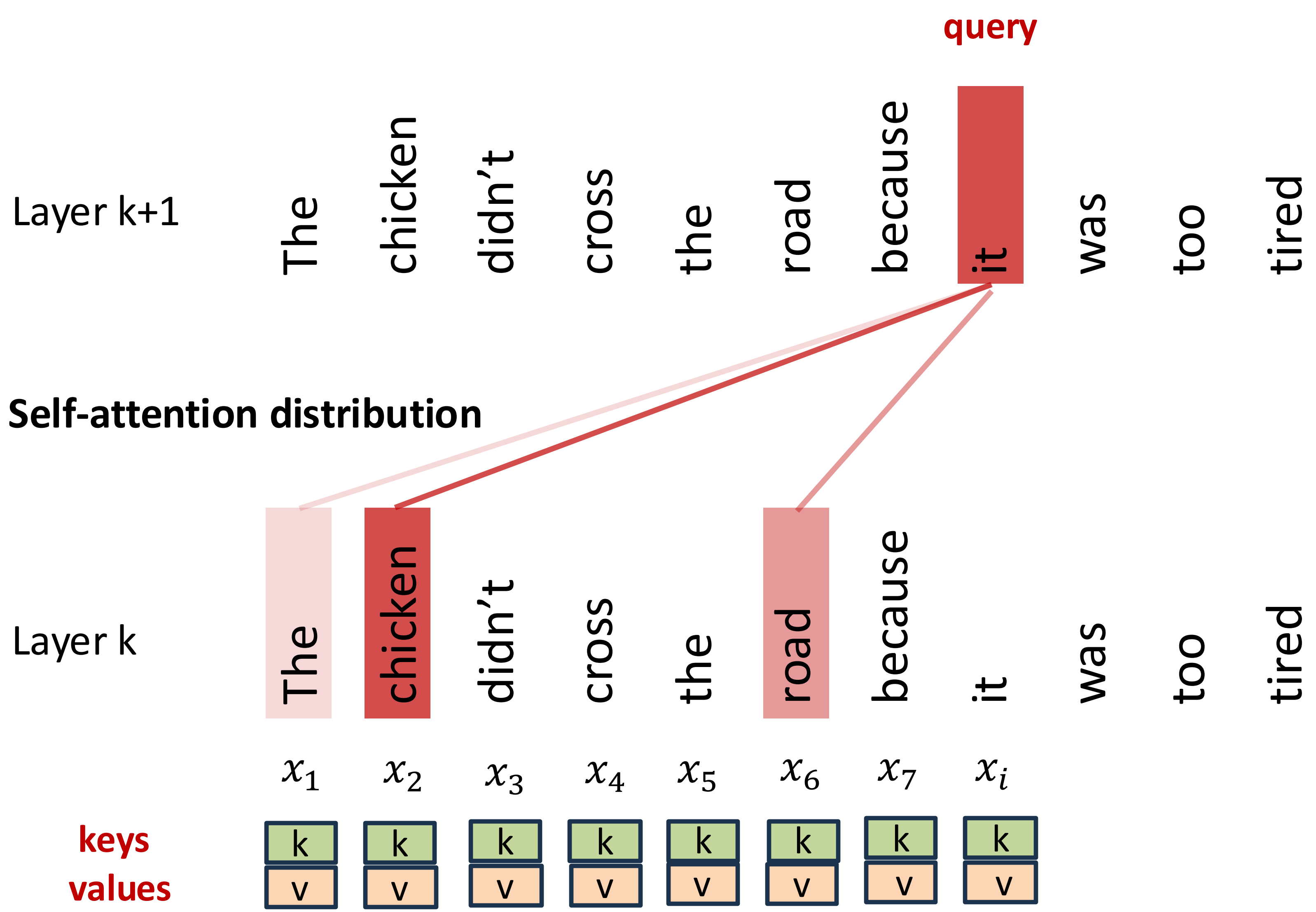
An Actual Attention Head

- High-level idea: instead of using vectors (like x_i and x_4) directly, we'll represent 3 separate roles each vector x_i players:
  - **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.

Information routing:

- You can think of **keys** as addresses, **queries** as questions, and **values** as answers.
- You match the query to the key (to get a weight), and then retrieve the value.

An Actual Attention Head



An Actual Attention Head

- We will use matrices to project each vector x_i into a representation of its role as query, key, value:
  - **Query**: $\boldsymbol{W}^Q$
  - **Key**: $\boldsymbol{W}^K$
  - **Value**: $\boldsymbol{W}^V$

$$\boldsymbol{q}_i = x_i \boldsymbol{W}^Q; \qquad \boldsymbol{k}_i = x_i \boldsymbol{W}^K; \qquad \boldsymbol{v}_i = x_i \boldsymbol{W}^V$$

An Actual Attention Head

Given these 3 representation of x_i

$$\boldsymbol{q}_i = x_i \boldsymbol{W}^Q; \qquad \boldsymbol{k}_i = x_i \boldsymbol{W}^K; \qquad \boldsymbol{v}_i = x_i \boldsymbol{W}^V$$

To compute similarity of current element x_i with some prior element x_j, we will use dot product between q_i and k_j.

Instead of summing up x_j, we will sum up v_j.

Q: Why?

A: Because we don't want to directly mix the **raw inputs $\boldsymbol{x}_j$**. We want to mix **task-specific transformed representations**— that's what the **values $\boldsymbol{v}_j$** are.

An Actual Attention Head: Final Equations

$$\boldsymbol{q}_i = x_i \boldsymbol{W}^Q; \qquad \boldsymbol{k}_j = x_j \boldsymbol{W}^K; \qquad \boldsymbol{v}_j = x_j \boldsymbol{W}^V$$

$$\text{score}(x_i, x_j) = \frac{\boldsymbol{q}_i \cdot \boldsymbol{k}_j}{\sqrt{d_k}}$$

$$\alpha_{ij} = \text{softmax}\left(\text{score}(x_i, x_j)\right), \qquad \forall j \le i$$

Q: Why divide by $\sqrt{d_k}$?

$$\boldsymbol{a}_i = \sum_{j \le i} \alpha_{ij} \, \boldsymbol{v}_j$$

1. Dot product grows with vector dimension:
   - Scaling keeps values small and controlled.
2. Softmax becomes too peaky without scaling:
   - Scaling softens softmax output.
3. Variance of scores too high:
   - Normalizes score magnitude to stabilize training.

Calculate the Value of a_3

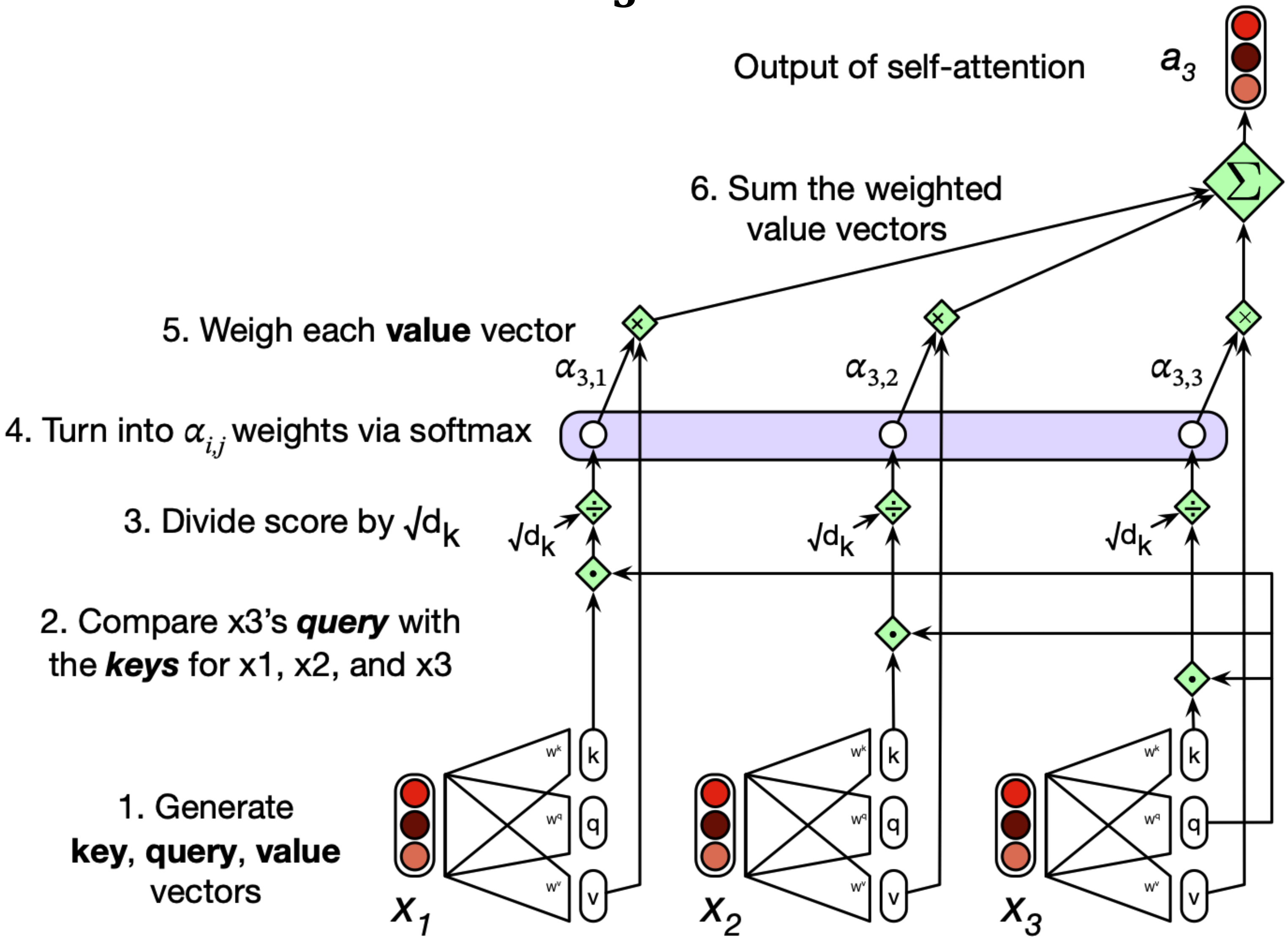


Figure from: Jurafsky and Martin (2024)

Multi-Head Attention

- 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

$$\boldsymbol{q}_i^c = x_i \boldsymbol{W}^{\boldsymbol{Q}c}; \quad \boldsymbol{k}_j^c = x_j \boldsymbol{W}^{\boldsymbol{K}c}; \quad \boldsymbol{v}_j^c = x_j \boldsymbol{W}^{\boldsymbol{V}c}; \ \forall c \in \{1, \dots, h\}$$

$$\text{score}^c(x_i, x_j) = \frac{\boldsymbol{q}_i^c \cdot \boldsymbol{k}_j^c}{\sqrt{d_k}}$$

$$\alpha_{ij} = \text{softmax}\left(\text{score}^c(x_i, x_j)\right), \qquad \forall j \leq i$$

$$\textbf{head}_i^c = \sum_{j \leq i} \alpha_{ij}^c \boldsymbol{v}_j^c$$

$$\boldsymbol{a_i} = (\textbf{head}^1 \oplus \textbf{head}^2 \oplus \dots \oplus \textbf{head}^h)\boldsymbol{W^O}$$

$$\text{MultiHeadAttention}(x_j, [x_1, \dots, x_N]) = \boldsymbol{a_i}$$

Part III: Transformer Language Models

Transformer Language Model

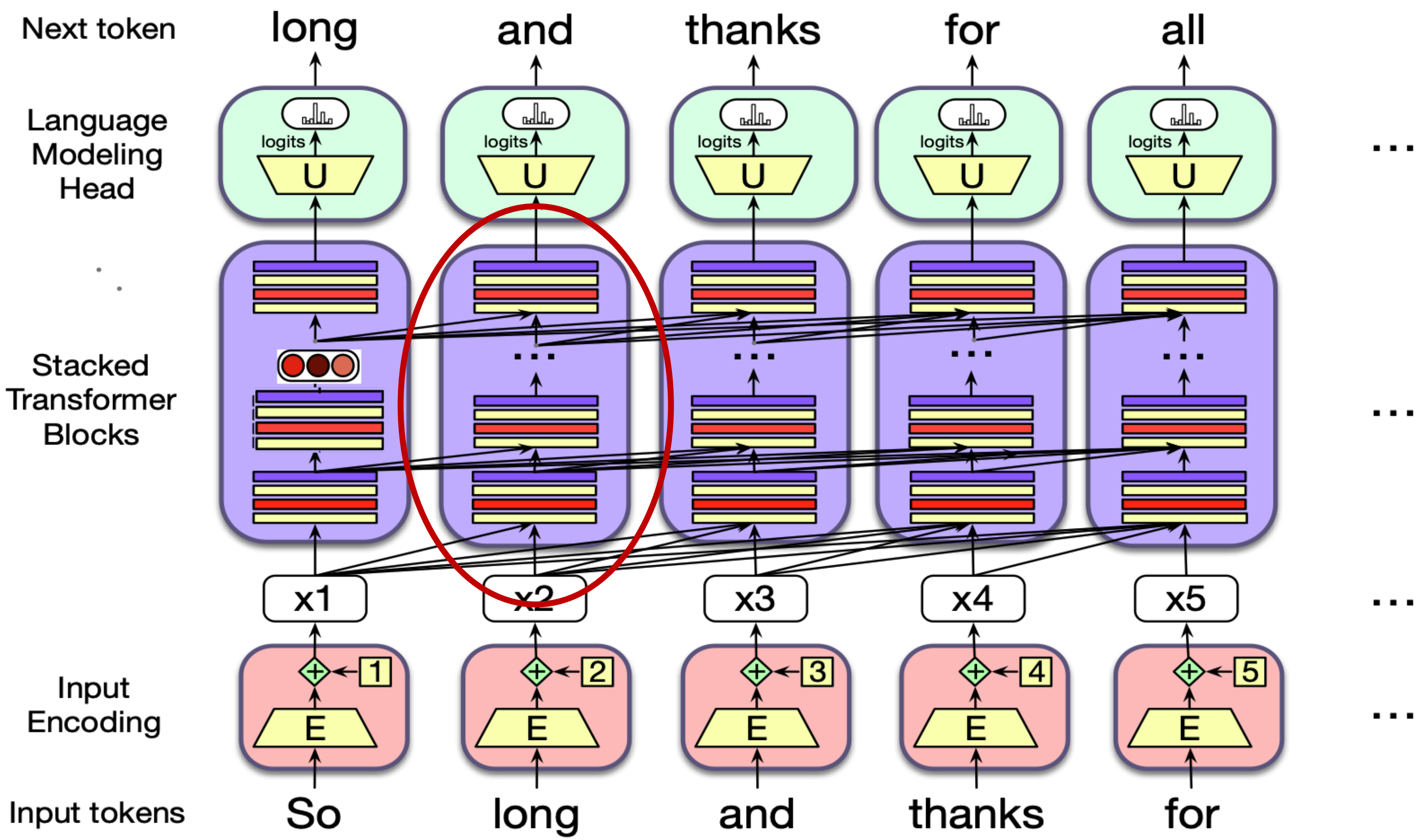


Figure from: Jurafsky and Martin (2024)

Transformer Language Model

- The residual stream: each token gets passed up and modified

Q: Why is residual stream important?

...

1. It prevents **vanishing gradients** during training (deep networks).
2. It lets each layer **refine** the representation rather than overwrite it.
3. Residual stream is also a **memory trace** of the input that's updated step-by-step.

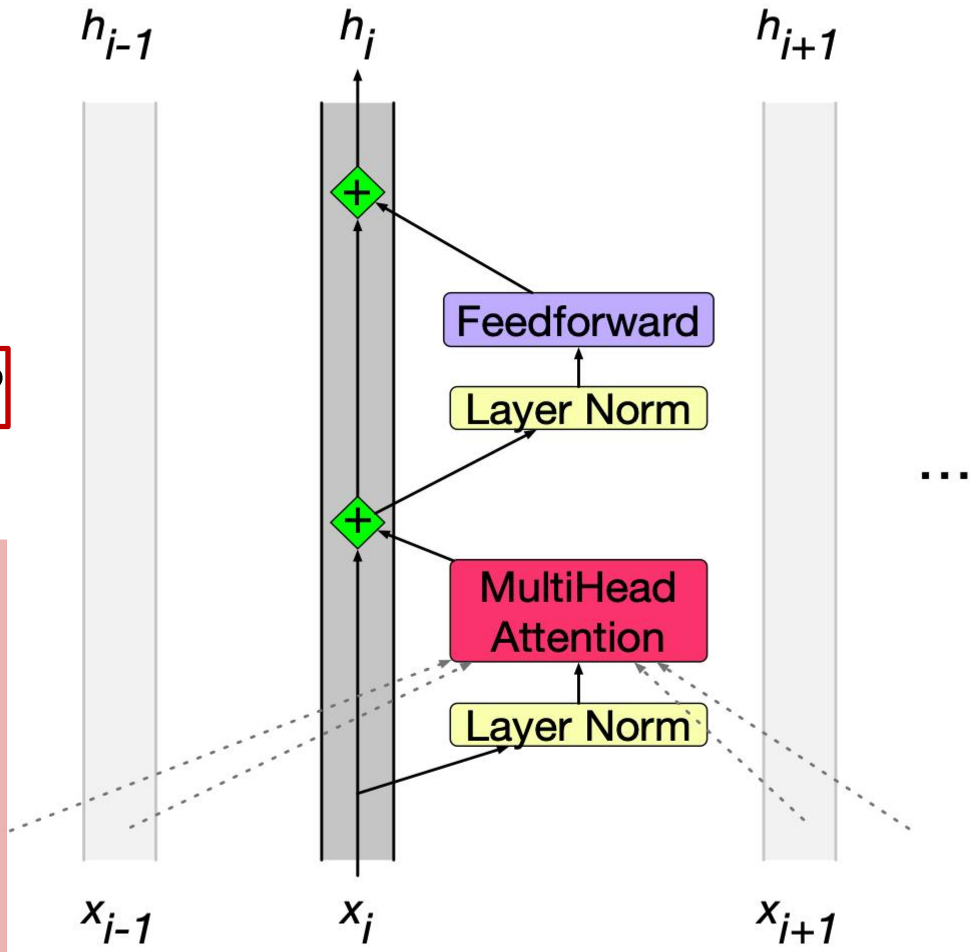


Figure from: Jurafsky and Martin (2024)

Transformer Language Model

- The residual stream: each token gets passed up and modified

- We'll need non-linearities, so a feedforward layer:

$$\mathrm{FFN}(x_i) = \mathrm{ReLU}(x_i \boldsymbol{W}_1 + b_1)\boldsymbol{W}_2 + b_2$$

- Layer norm: the vector x_i is normalized twice

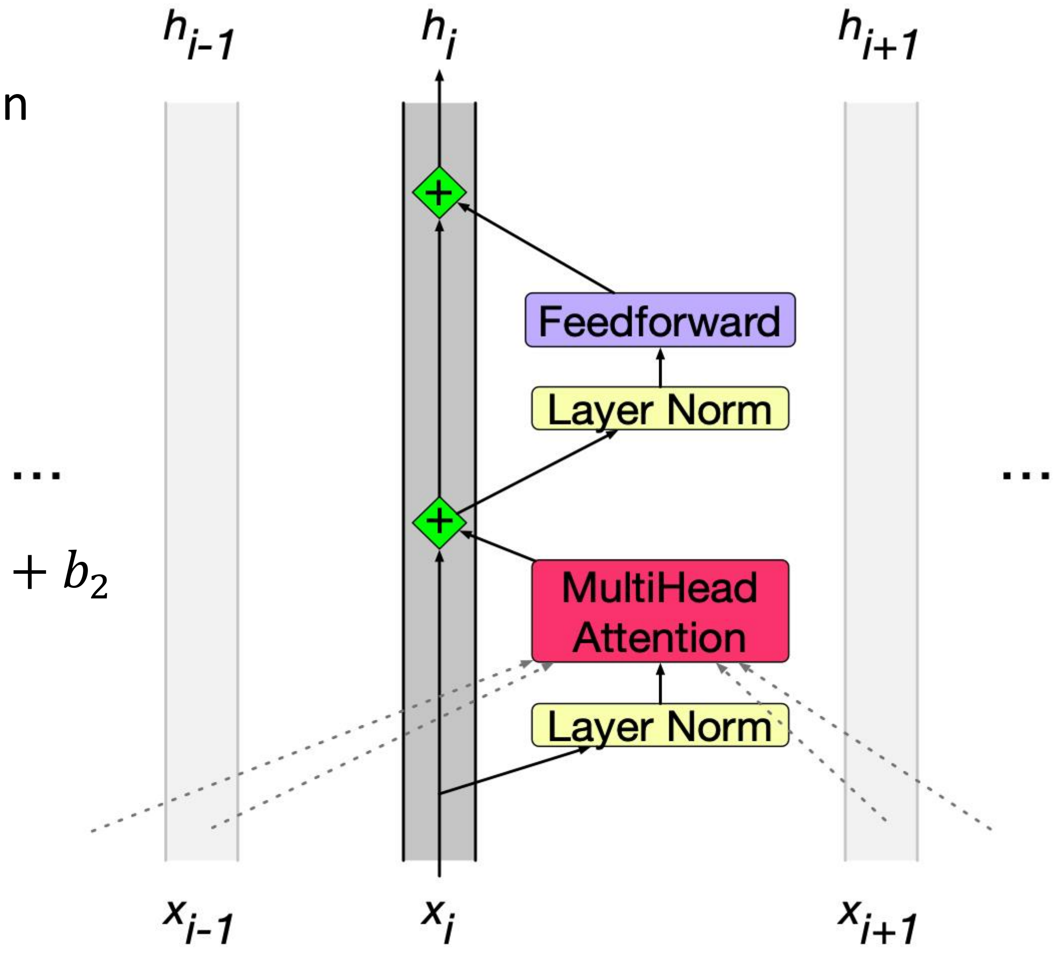


Figure from: Jurafsky and Martin (2024)

Layer Norm

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

$$\mu = \frac{1}{d}\sum_{i=1}^{d} x_i$$ mean of the features

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

$$\hat{\boldsymbol{x}} = \frac{(\boldsymbol{x} - \boldsymbol{\mu})}{\sigma}$$ z-score (standardization)

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

learnable scale and shift parameters

Q: What's the function of Layer norm?

1. Stabilizes training: it prevents exploding or vanishing gradients.
2. Makes the optimization landscape smoother by keeping layer inputs at a consistent scale.
3. Allows deeper networks (like Transformers) to train faster and more reliably.

Putting Together

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

$$t_i^2 = \text{MultiHeadAttention}(t_i^1, [x_1^1, \dots, x_N^1])$$

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

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

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

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

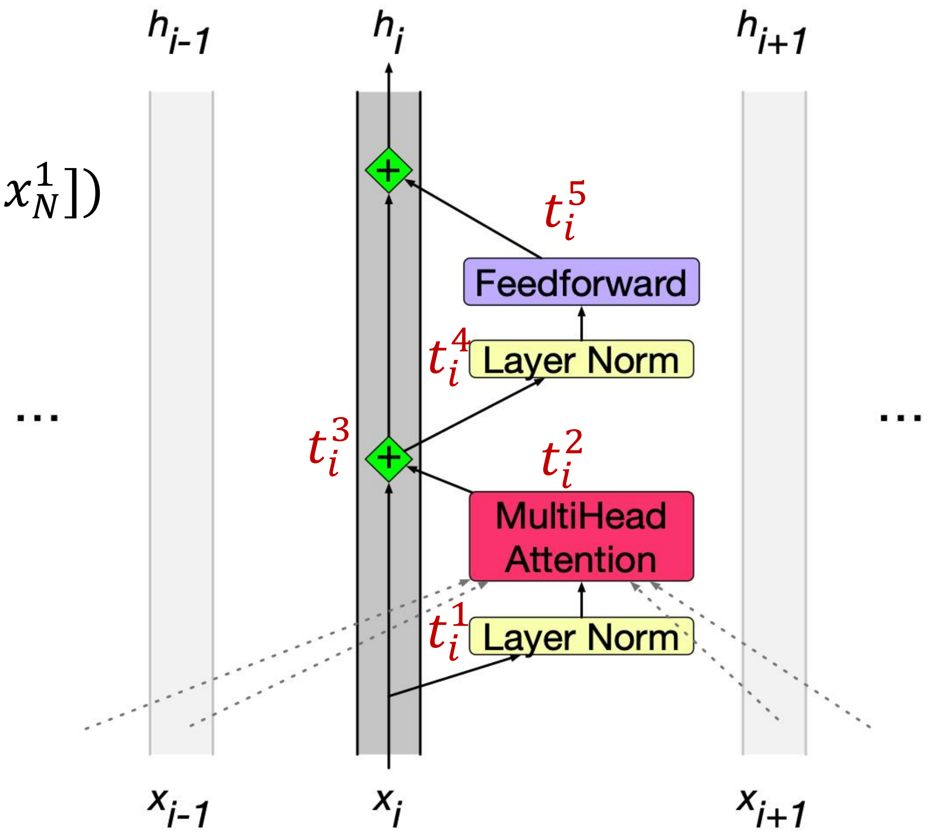


Figure from: Jurafsky and Martin (2024)

Part IV: Implementing a Transformer LM

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 $\boldsymbol{X}$ of size $[N \times d]$.
- Each row of $\boldsymbol{X}$ is the embedding of one token of the input
- $\boldsymbol{X}$ can have 1K-32K rows, each of the dimensionality of the embedding d (the **model dimension**)

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

$\boldsymbol{QK^T}$

- Now can do a single matrix multiply to combine $\boldsymbol{Q}$ and $\boldsymbol{K^T}$

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

Figure from: Jurafsky and Martin (2024)

Masking out the Future

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

- What is the mask function?
  - QK^T has a score for each query dot every key, including those that follow the query
  - Add $-\infty$ to cells in upper triangle.
  - Then the softmax will turn it to 0.

N

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

N

Figure from: Jurafsky and Martin (2024)

Attention Again

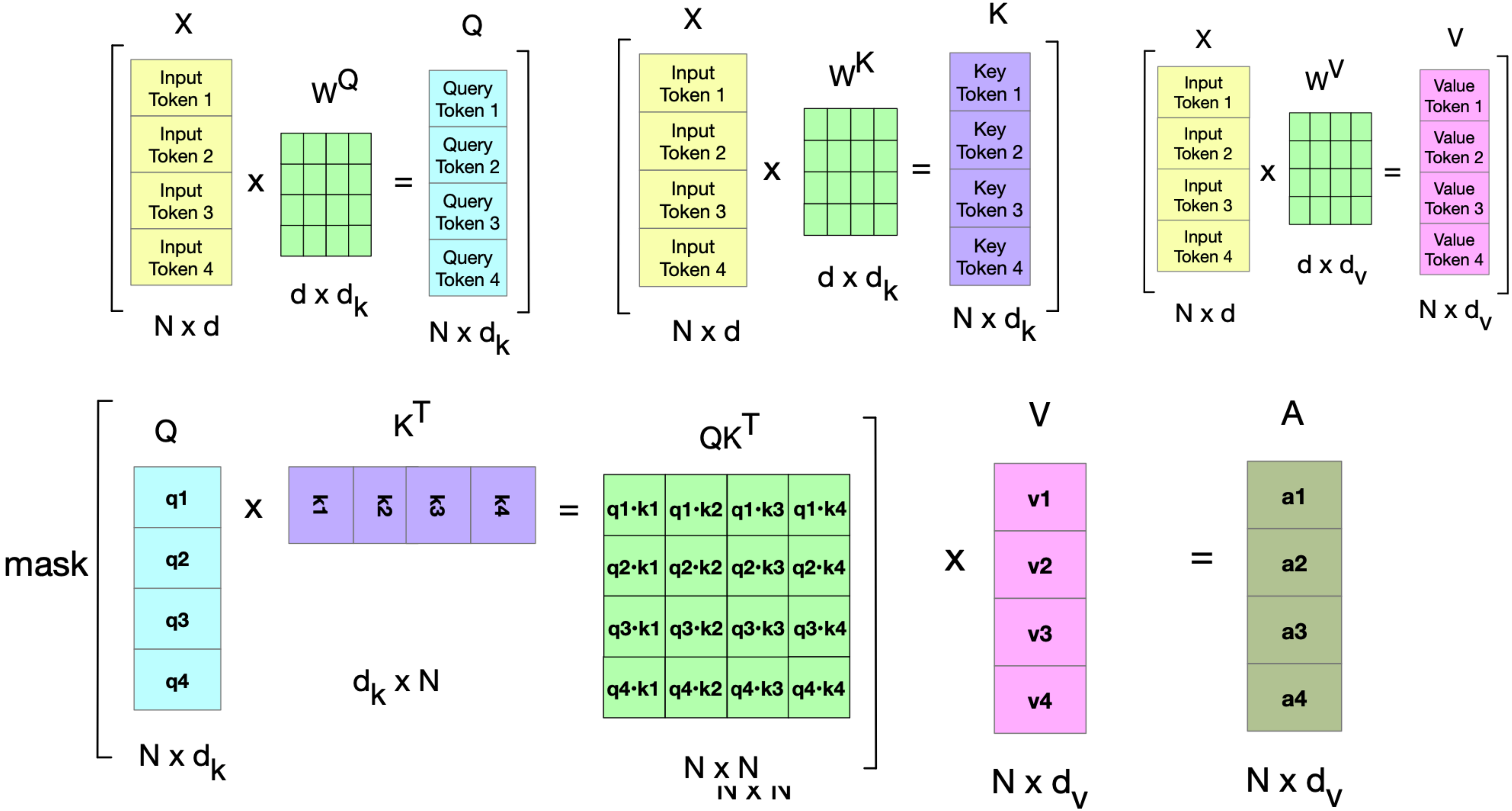


Figure from: Jurafsky and Martin (2024)

Position Embeddings

- There are many methods, but we'll just describe the simplest: absolute position.
- Each $\mathbf{X_i}$ is just the sum of word and position embeddings.

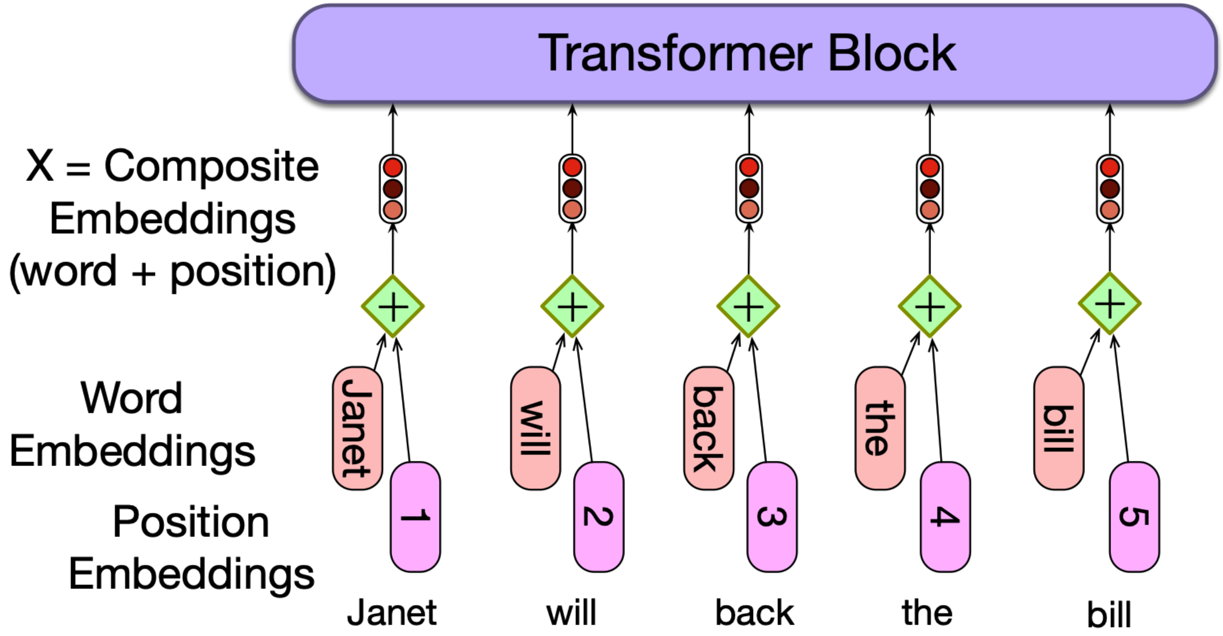


Figure from: Jurafsky and Martin (2024)


рахмат
danke
謝謝
ngiyabonga
spas
teşekkür ederim
Баярлалаа
mersi
kia ora
barka
welalin
tack
mahalo
tapadh leat
спасибо
faafetai lava
vinaka
спасибі
blagodaram
dank je
misaotra
matondo
paldies
grazzi
kiitos
dankie
nanni
nandri
dhanyavad
thank you
gracias
хвала
asante
manana
obrigada
tenki
murakoze
maururu
köszönöm
hvala
bedankt
bayarlalaa
gràcie
akun
dankon
aciū
enkosi
djiere dieuf
tau
mochchakkeram
дякую
mamnun
chokrane
dziękuję
chnorakaloutioun
gratias ago
gràcies
sulpáy
go raibh maith agat
taiku
sobodi
děkuji
sagolun
sukriya
kop khun krap
arigatô
takk
dakujem
trugarez
obrigado
mèsi
didi madloba
najis tuke
ありがとう
tanemirt
grazie
terima kasih
rahmet
kam sah hamnida
rahmat
diolch
dhanyavadagalu
shukriya
mercê
мерси
धन्यवाद
তোমাকে ধন্যবাদ
감사합니다
xièxie
ευχαριστώ
merci