Introduction to Transformers

CS60216: Safety Fundamentals for Generative AI

Jan 9th, 2024

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

Great Results with Transformers: NMT

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

3

[Test sets: WMT 2014 English-German and English-French]

https://web.stanford.edu/class/cs224n/

Great Results with Transformers: Rise of LLMs

Today, Transformer-based models dominate LMSYS Chatbot Arena Leaderboard!

Rank	Model	Arena Elo	95% CI	Votes	Organization	License	Knowledge Cutoff
1	GPT-4-Turbo-2024-04-09	1258	+4/-4	26444	OpenAI	Proprietary	2023/12
1	GPT-4-1106-preview	1253	+3/-3	68353	OpenAI	Proprietary	2023/4
1	Claude 3 Opus	1251	+3/-3	71500	Anthropic	Proprietary	2023/8
2	Gemini 1.5 Pro API-0409-Preview	1249	+4/-5	22211	Google	Proprietary	2023/11
3	GPT-4-0125-preview	1248	+2/-3	58959	OpenAI	Proprietary	2023/12
6	Meta Llama 3 70b Instruct	1213	+4/-6	15809	Meta	Llama 3 Community	2023/12
6	Bard (Gemini Pro)	1208	+7/-6	12435	Google	Proprietary	Online
7	Claude 3 Sonnet	1201	+4/-2	73414	Anthropic	Proprietary	2023/8

Gemini / Bard (Google)

ChatGPT / GPT-4 (OpenAI)

Claude 3 (Anthropic)

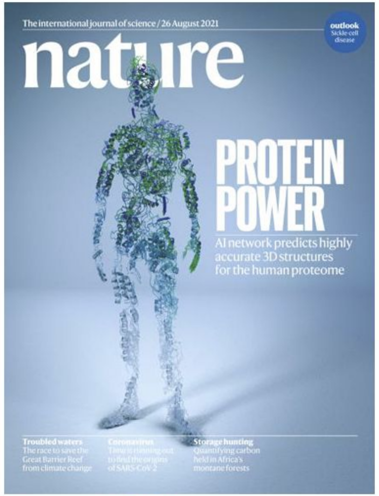
Llama 3 (Meta)

[Chiang et al., 2024]

https://web.stanford.edu/class/cs224n/

Transformers have shown promise outside NLP

Protein Folding



[Jumper et al. 2021] aka AlphaFold2!

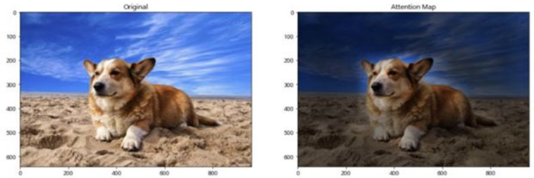
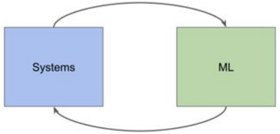
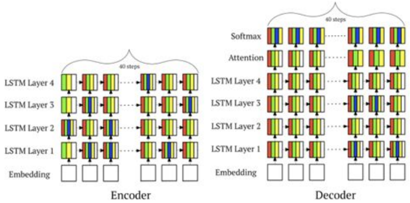


Image Classification

[Dosovitskiy et al. 2020]: Vision Transformer (ViT) outperforms ResNet-based baselines with substantially less compute.

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	**90.72** ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	–
CIFAR-100	**94.55** ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	–
Oxford-IIIT Pets	**97.56** ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	–
Oxford Flowers-102	99.68 ± 0.02	**99.74** ± 0.00	99.61 ± 0.02	99.63 ± 0.03	–
VTAB (19 tasks)	**77.63** ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k





ML for Systems

[Zhou et al. 2020]: A Transformer-based compiler model (GO-one) speeds up a Transformer model!

Model (#devices)	GO-one (s)	HP (s)	METIS (s)	HDP (s)	Run time speed up over HP / HDP	Search speed up over HDP
2-layer RNNLM (2)	0.173	0.192	0.355	0.191	9.9% / 9.4%	2.95x
4-layer RNNLM (4)	0.210	0.239	0.503	0.251	13.8% / 16.3%	1.76x
8-layer RNNLM (8)	0.320	0.332	OOM	0.764	3.8% / 58.1%	27.8x
2-layer GNMT (2)	0.301	0.384	0.344	0.327	27.6% / 14.3%	30x
4-layer GNMT (4)	0.350	0.469	0.466	0.432	34% / 23.4%	58.8x
[illegible]	0.440	0.562	OOM	0.693	21.7% / 36.5%	7.35x
$\boxed{\text{2-layer Transformer-XL (2)}}$	0.223	0.268	0.37	0.262	20.1% / 17.4%	40x
$\boxed{\text{4-layer Transformer-XL (4)}}$	0.230	0.27	OOM	0.259	17.4% / 12.6%	26.7x
$\boxed{\text{8-layer Transformer-XL (8)}}$	0.350	0.46	OOM	0.425	23.9% / 16.7%	16.7x
[illegible]	0.229	0.312	OOM	0.301	26.6% / 23.9%	13.5x
Inception (2) b64	0.423	0.731	OOM	0.498	42.1% / 29.3%	21.0x
AmoebaNet (4)	0.394	0.44	0.426	0.418	26.1% / 6.1%	58.8x
2-stack 18-layer WaveNet (2)	0.317	0.376	OOM	0.354	18.6% / 11.7%	6.67x
4-stack 36-layer WaveNet (4)	0.659	0.988	OOM	0.721	50% / 9.4%	20x
GEOMEAN	-	-	-	-	**20.5% / 18.2%**	**15x**

https://web.stanford.edu/class/cs224n/

Transformer Encoder-Decoder

- Transformer is introduced as an **encoder-decoder architecture**; later we will see **encoder-only** & **decoder-only** transformers
- **Encoder** produces a sophisticated representation of the source sequence that the decoder will use to condition its generation process
- **Decoder** generates one token at the time to produce a target sequence; in the produces, it produces representations that combine the history and a new token

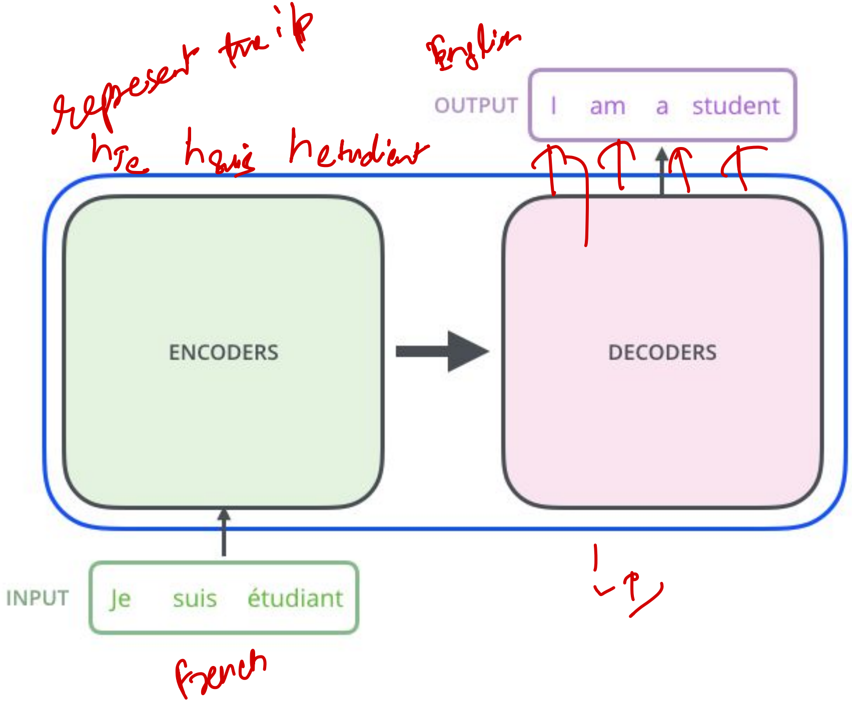
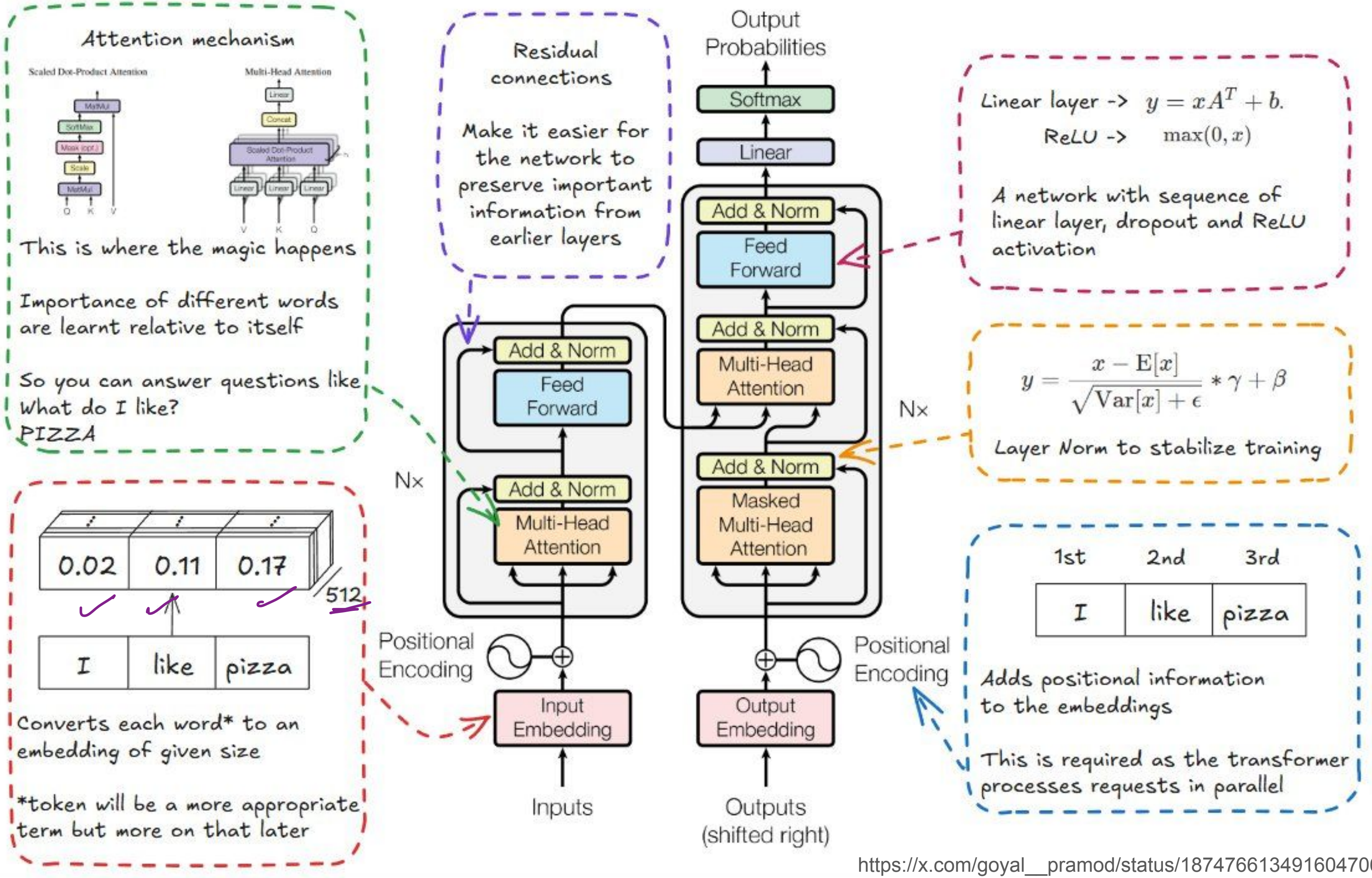


Figure: [Jay Alammar]

https://utah-cs6340-nlp.notion.site/Natural-Language-Processing-bd1a2ca290fc44f69556908ad8d25c70



https://x.com/goyal__pramod/status/1874766134916047061/photo/1

I like pizza

I $[0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0]$

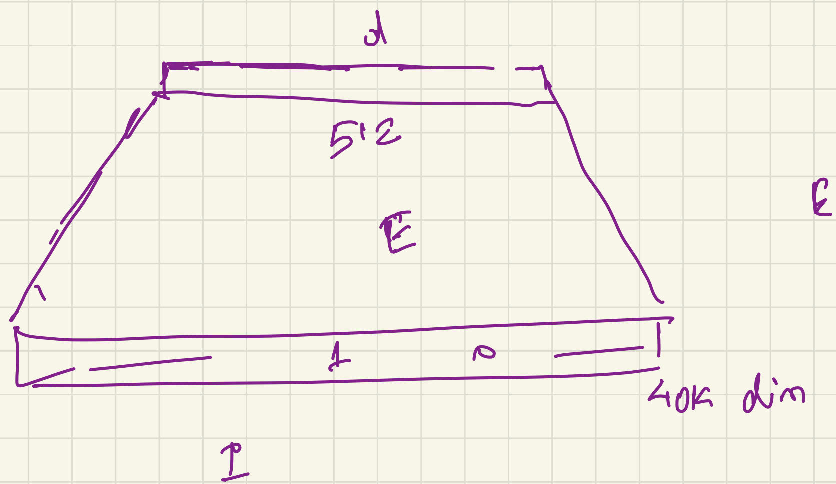
dim = vocab size

How do you decide the vocab size?

~~# distinct words in data~~

May be millions!!

Sub-words [BPE] 40 K <u>vocab</u>



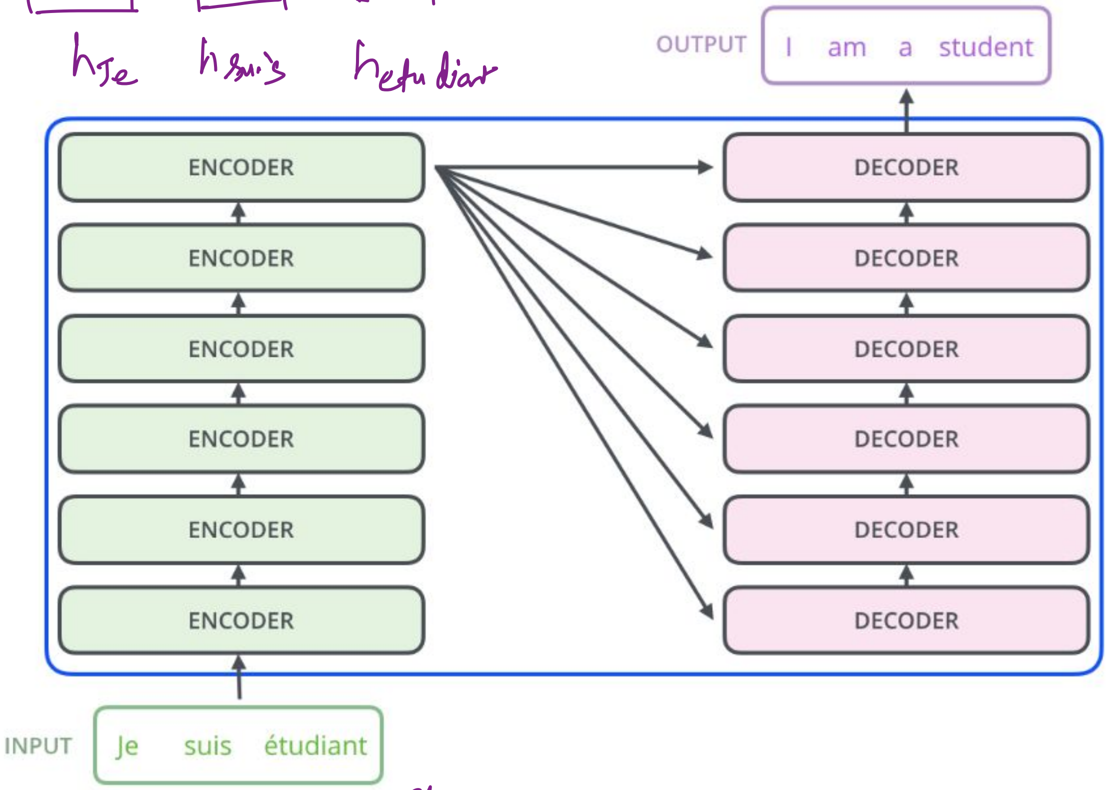
E: 40K × 512

model_dim

512 512 512

h_{Je} h_{suis} $h_{etudiant}$

A stack of encoder blocks



A stack of decoder blocks

512 512 512

Figure: [Jay Alammar]

Each encoder block consists of **self-attention & FFNN**

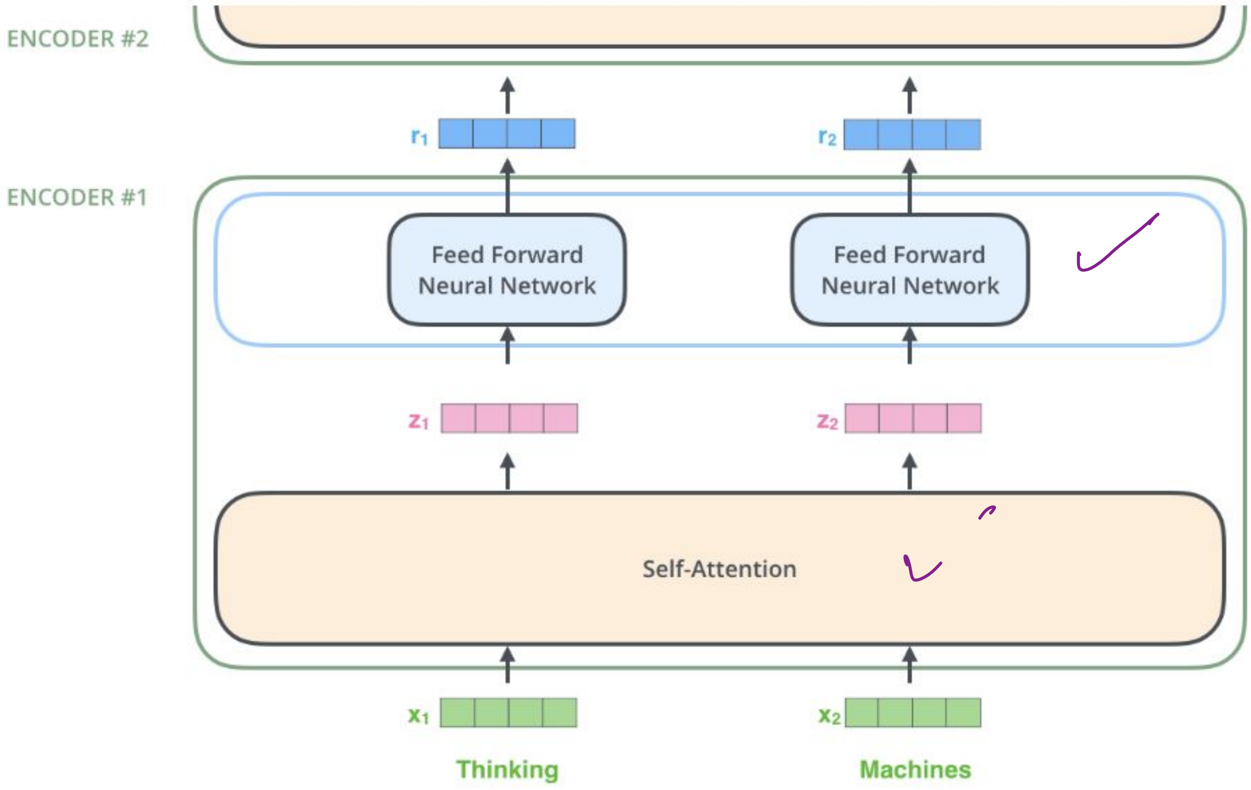


Figure: [Jay Alammar]()

Deeper layers get outputs of the previous layers as inputs

Input to each encoder block has the same size as original token embeddings

In the first layer, inputs are static token emeddings

Intuition for attention

`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 animal or the street

https://web.stanford.edu/~jurafsky/slp3/

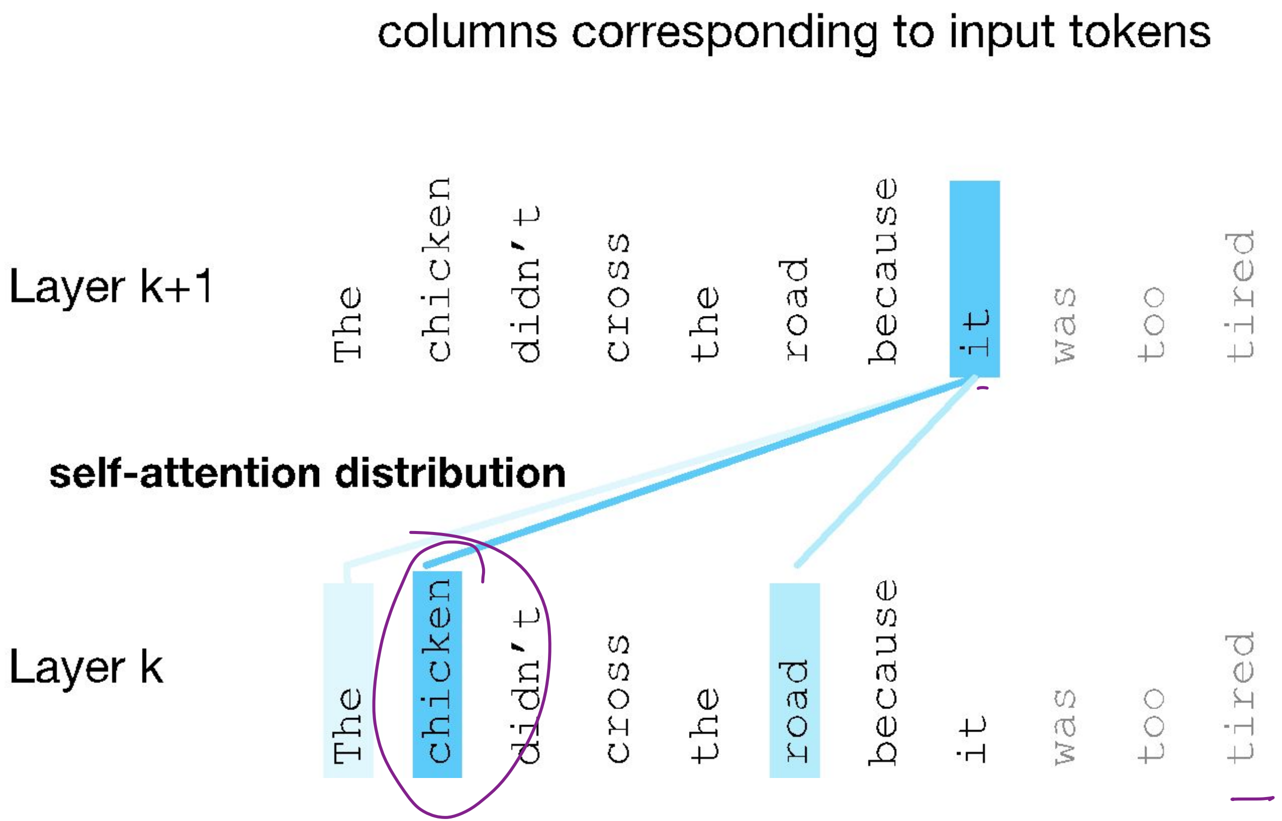
Intuition of attention

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

https://web.stanford.edu/~jurafsky/slp3/

Intuition of attention



https://web.stanford.edu/~jurafsky/slp3/

Intuition for Self-attention

Attention is based on key/value/query concept -- analogous to retrieval systems.

When you search for videos on Youtube

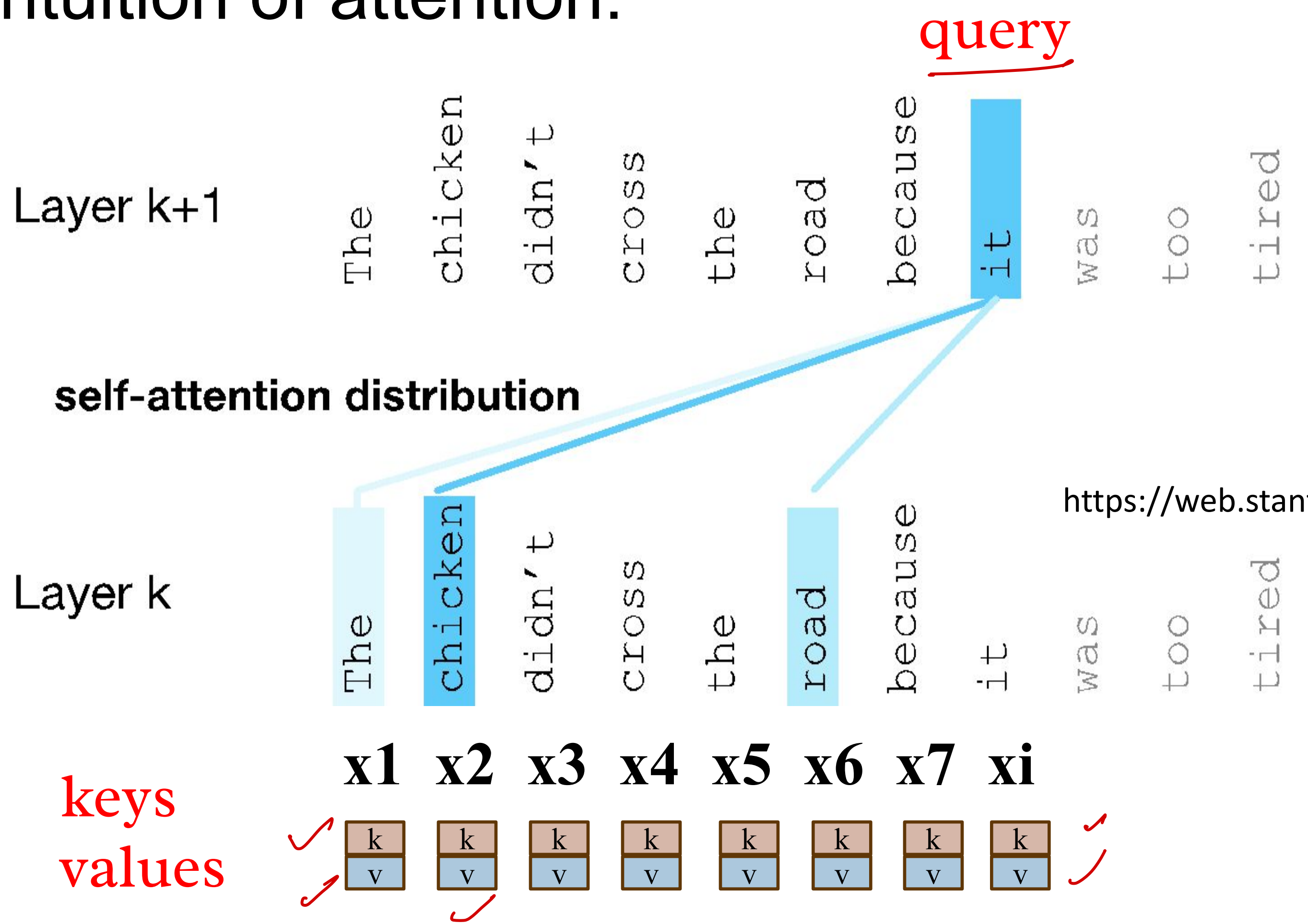
- The search engine will map your query (text in the search bar) against a set of keys (video title, description, etc.) associated with candidate videos in their database
- It will then present you the best matched videos (values).

K_1

$q \rightarrow \boxed{K_2} \rightarrow v_2$

K_n

Intuition of attention:



https://web.stanford.edu/~jurafsky/slp3/

An Actual Attention Head: 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**: $\mathbf{W}^{\mathbf{V}}$

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

https://web.stanford.edu/~jurafsky/slp3/

An Actual Attention Head: slightly more complicated

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

To compute similarity of current element $\boldsymbol{x}_i$ with some prior element $\boldsymbol{x}_j$

We'll use dot product between $\boldsymbol{q}_i$ and $\boldsymbol{k}_j$.

And instead of summing up $\boldsymbol{x}_j$, we'll sum up $\boldsymbol{v}_j$

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

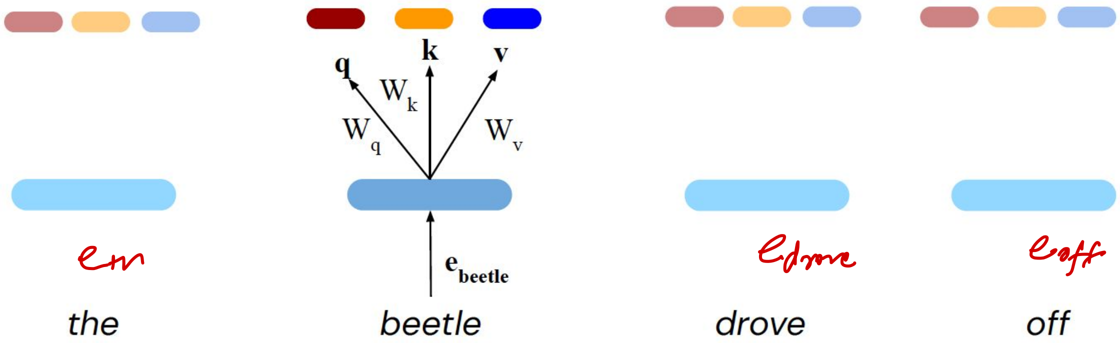
https://web.stanford.edu/~jurafsky/slp3/

Transformers: Self-attention over input

$$\mathbf{q} = \mathbf{e}_{beetle}\ \underline{\mathrm{W}_q}$$

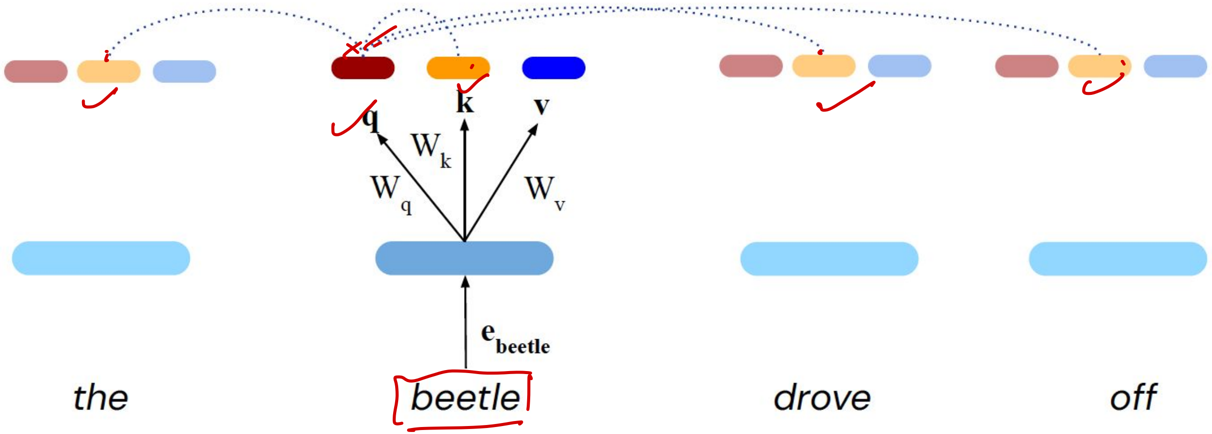
$$\mathbf{k} = \mathbf{e}_{beetle}\ \underline{\mathrm{W}_k}$$

$$\mathbf{V} = \mathbf{e}_{beetle}\ \underline{\mathrm{W}_v}$$



Source: *https://deepmind.com/learning-resources/deep-learning-lecture-series-2020*

Self-attention over input embeddings

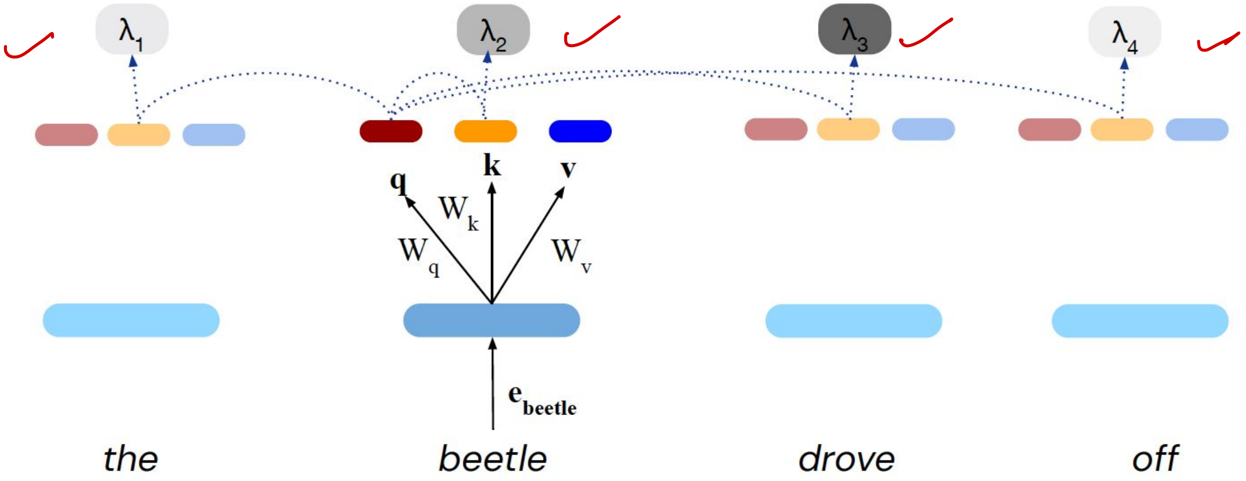


Source: *https://deepmind.com/learning-resources/deep-learning-lecture-series-2020*

Self-attention over input embeddings

$$\lambda_i = \frac{e^{\mathbf{q}\cdot\mathbf{k_i}}}{\sum_{i=1}^{4} e^{\mathbf{q}\cdot\mathbf{k_i}}}$$

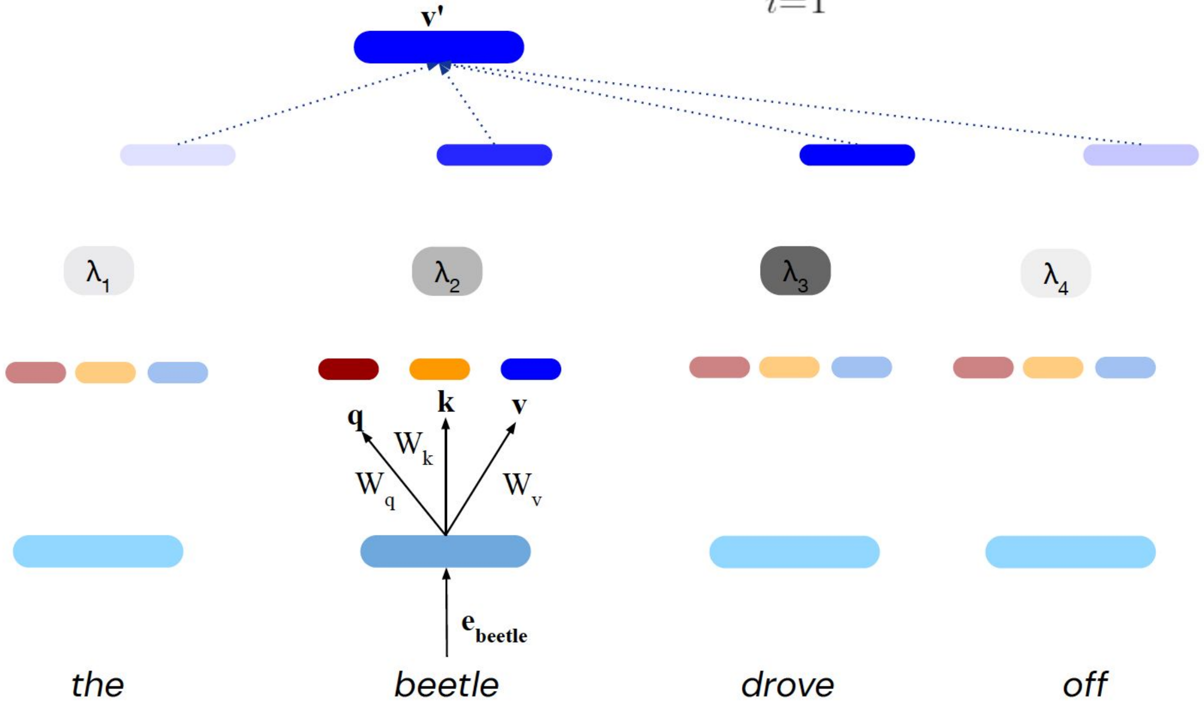
$\lambda_1 v_1 + \lambda_2 v_2 + \lambda_3 v_3$
$+ \lambda_4 v_4$



Source: *https://deepmind.com/learning-resources/deep-learning-lecture-series-2020*

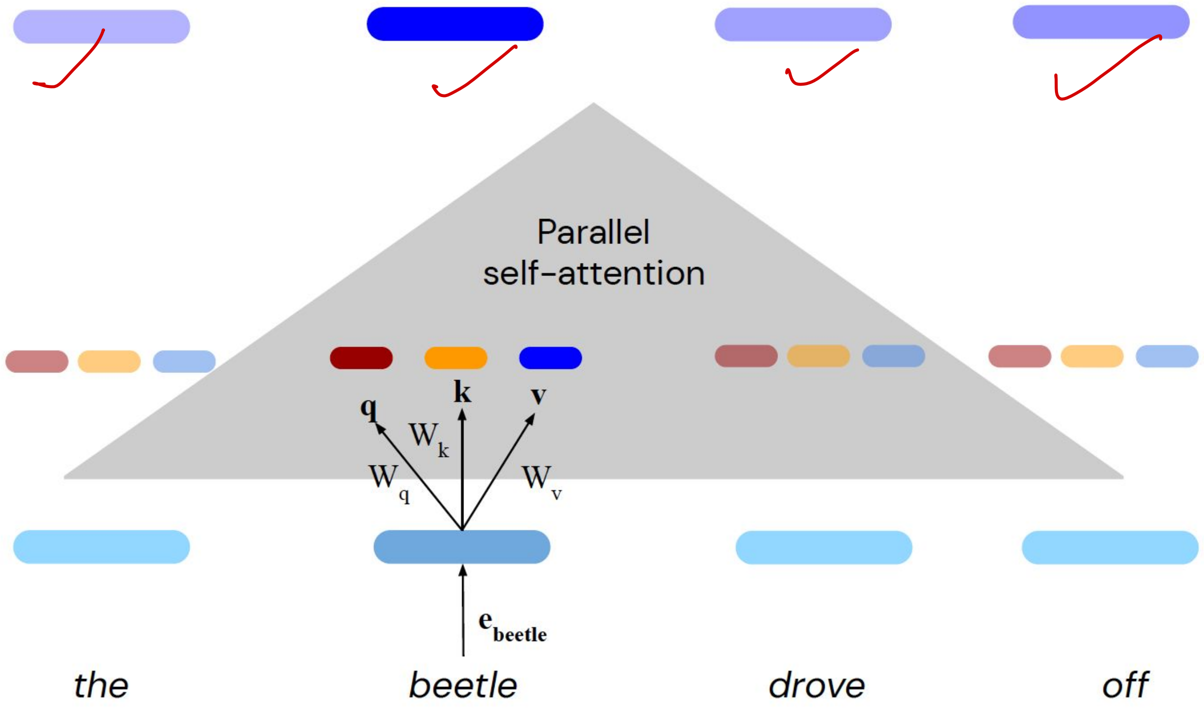
Self-attention over input embeddings

$$\mathbf{v}' = \sum_{i=1}^{4} \lambda_i \mathbf{v}_i$$



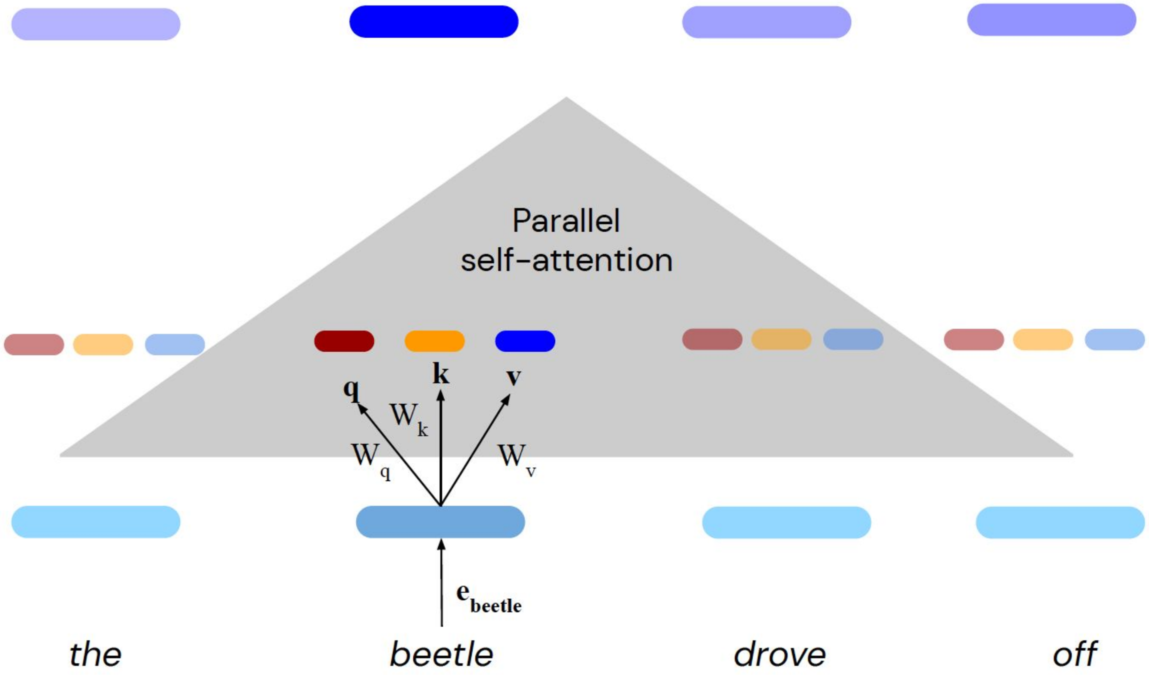
Source: *https://deepmind.com/learning-resources/deep-learning-lecture-series-2020*

Self-attention over all words (in parallel)



Source: *https://deepmind.com/learning-resources/deep-learning-lecture-series-2020*

Self-attention over all words (in parallel)



Source: *https://deepmind.com/learning-resources/deep-learning-lecture-series-2020*

Self-attention: In equations

hardmaru
@hardmaru · Follow

The most important formula in deep learning after 2018

Self-Attention

What is self-attention? Self-attention calculates a weighted average of feature representations with the weight proportional to a similarity score between pairs of representations. Formally, an input sequence of n tokens of dimensions d, $X \in \mathbf{R}^{n \times d}$, is projected using three matrices $W_Q \in \mathbf{R}^{d \times d_q}$, $W_K \in \mathbf{R}^{d \times d_k}$, and $W_V \in \mathbf{R}^{d \times d_v}$ to extract feature representations Q, K, and V, referred to as query, key, and value respectively with $d_k = d_q$. The outputs Q, K, V are computed as

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V. \tag{1}$$

So, self-attention can be written as,

$$S = D(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_q}}\right)V, \tag{2}$$

where softmax denotes a *row-wise* softmax normalization function. Thus, each element in S depends on all other elements in the same row.

7:38 AM · Feb 10, 2021

3.1K Reply Copy link

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

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

→ scaled dot-product

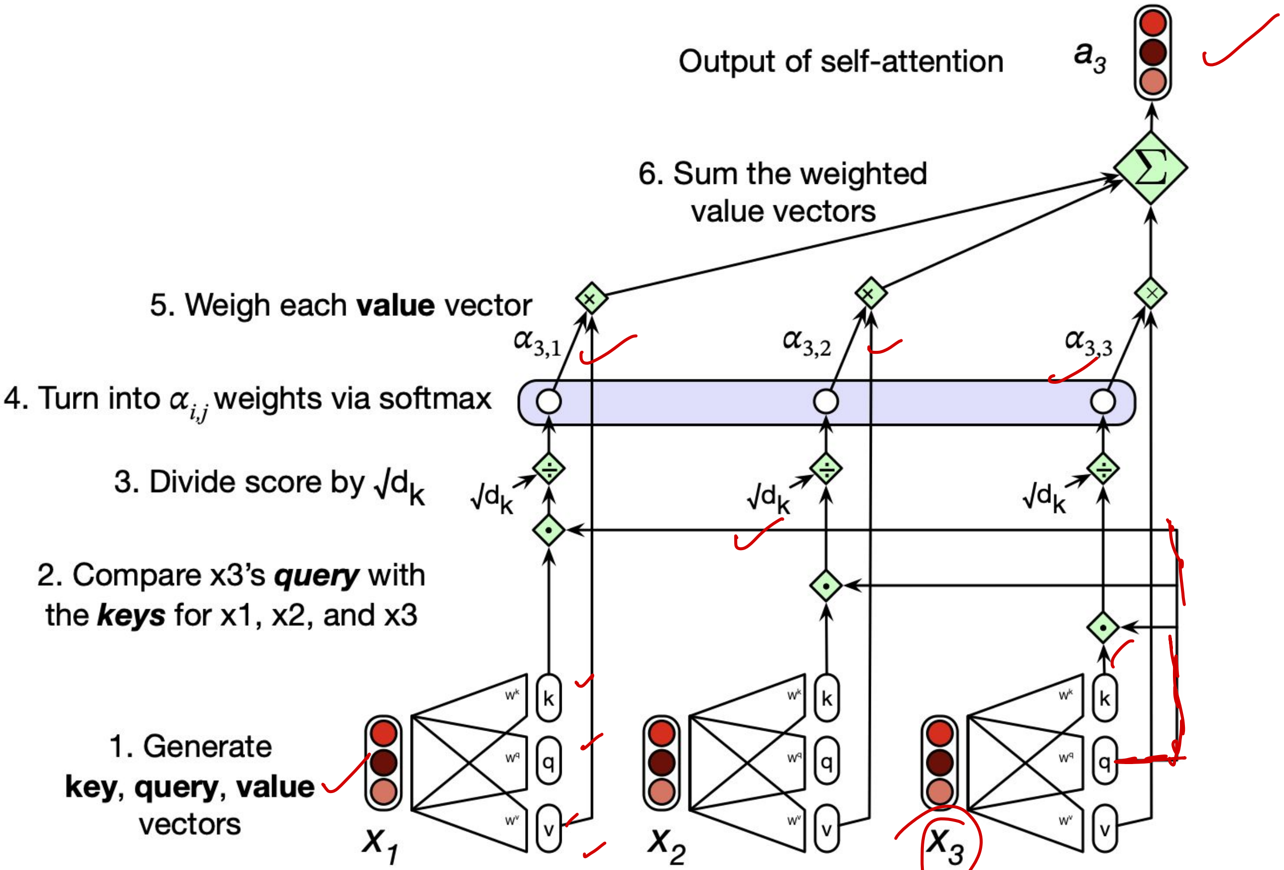
$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j))$$

$$\mathbf{a}_i = \sum \alpha_{ij} \mathbf{v}_j$$

Scaled dot-product: *more on this later*

Source: *https://theaisummer.com/self-attention/* *https://web.stanford.edu/~jurafsky/slp3/*

Calculating the self-attention output



https://web.stanford.edu/~jurafsky/slp3/

Try this problem

softmax $\left[\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\right] \approx [0.8\ \ 0.2]$

Suppose, you give the following input to your transformer encoder: {flying, arrows} The input embeddings for these two words are **[0,1,1,1,1,0] and [1,1,0,-1,-1,1]**, respectively. Suppose you are trying to represent the first word ‘flying’ with the help of self-attention in the first encoder. For the first attention head, the query, key and value matrices just take the 2 dimensions from the input each. Thus, the first 2 dimensions define the query vector, and so on. What will be the self-attention output for the word ‘flying’ corresponding to this attention head. You are using the scaled dot vector.

$1/\sqrt{2}$ $\quad$ $-1/\sqrt{2}$

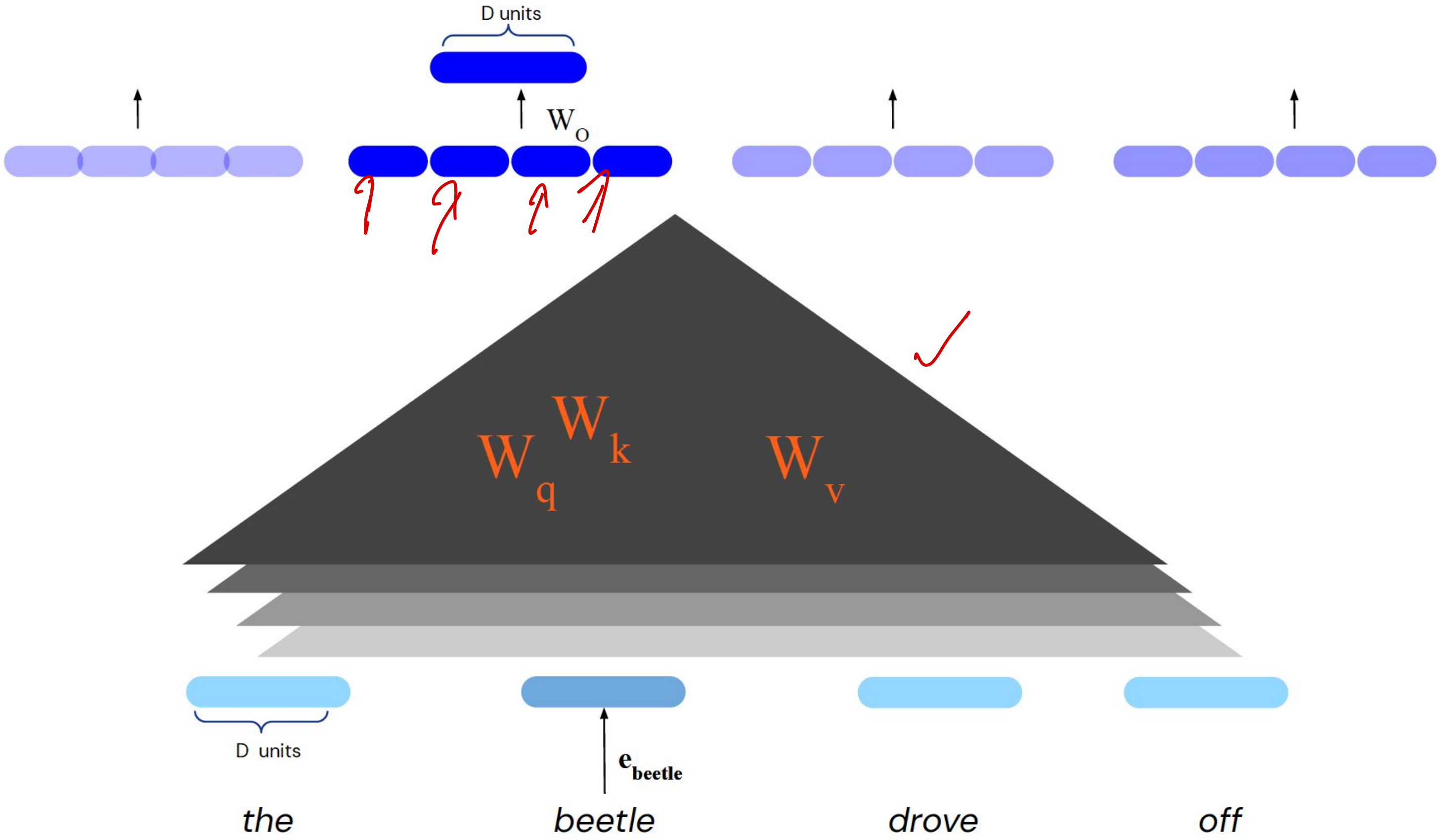
$\underbrace{0\ 1}_{q_1}$ $\quad \underbrace{1\ 1}_{K_1}$ $\quad \underbrace{1\ 0}_{v_1}$ $\qquad \underbrace{1\ 1}_{q_2}$ $\quad \underbrace{0\ -1}_{K_2}$ $\quad \underbrace{-1\ 1}_{v_2}$

$$0.8\,[1\ \ 0] + 0.2\,[-1\ \ 1] = [0.6\ \ 0.2]$$

Try this problem

q1: [0,1], k1: [1,1],, v1: [1,0]
q2: [1,1], k2: [0,-1], v2: [-1,1]
a1?

Multi-head Attention



Source: *https://deepmind.com/learning-resources/deep-learning-lecture-series-2020*

Why Multi-head attention?

- What if we want to look in multiple places in the sentence at once?
  - For word i, maybe we want to focus on different j for different reasons?

- We'll define multiple attention "heads" through multiple Q,K,V matrices

- Each attention head performs attention independently

- Then the outputs of all the heads are combined!

- Each head gets to "look" at different things, and construct value vectors differently.

Why Multi-head attention?

Prior work identified three important types of heads by looking at attention matrices

[Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned: https://arxiv.org/abs/1905.09418]

1. **Positional** heads that attend mostly to their neighbor.
2. **Syntactic** heads that point to tokens with a specific syntactic relation.
3. Heads that point to **rare words** in the sentence.

Source: https://theaisummer.com/self-attention/

Multi-Head Attention: In Equations

- Each head might be attending to the context for different purposes
  - Different linguistic relationships or patterns in the context

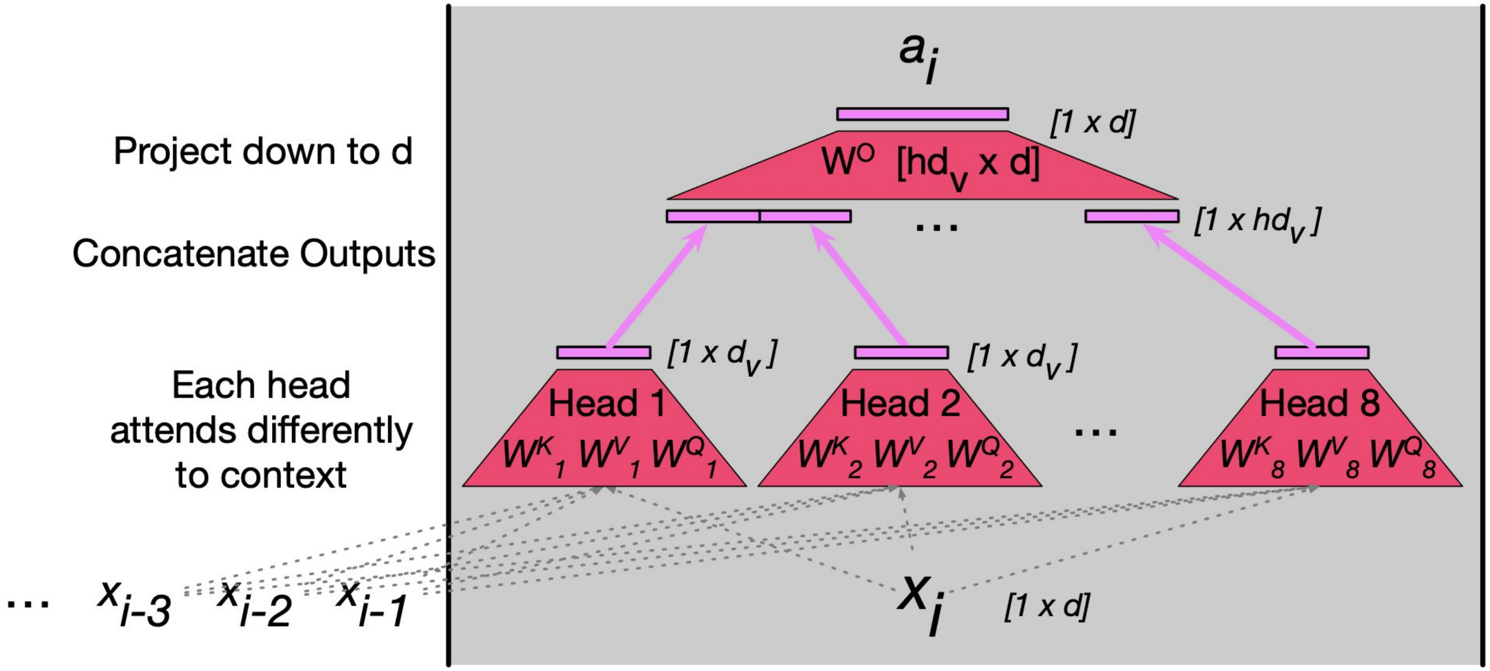
$$
\begin{aligned}
\mathbf{q}_i^c = \mathbf{x}_i \mathbf{W}^{\mathbf{Q}c}; \quad \mathbf{k}_j^c = \mathbf{x}_j \mathbf{W}^{\mathbf{K}c}; \quad \mathbf{v}_j^c &= \mathbf{x}_j \mathbf{W}^{\mathbf{V}c}; \quad \forall c \quad 1 \le c \le \boxed{h} \\
\text{score}^c(\mathbf{x}_i, \mathbf{x}_j) &= \frac{\mathbf{q}_i^c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

h heads

d-dim

https://web.stanford.edu/~jurafsky/slp3/

Multi-head attention



https://web.stanford.edu/~jurafsky/slp3/

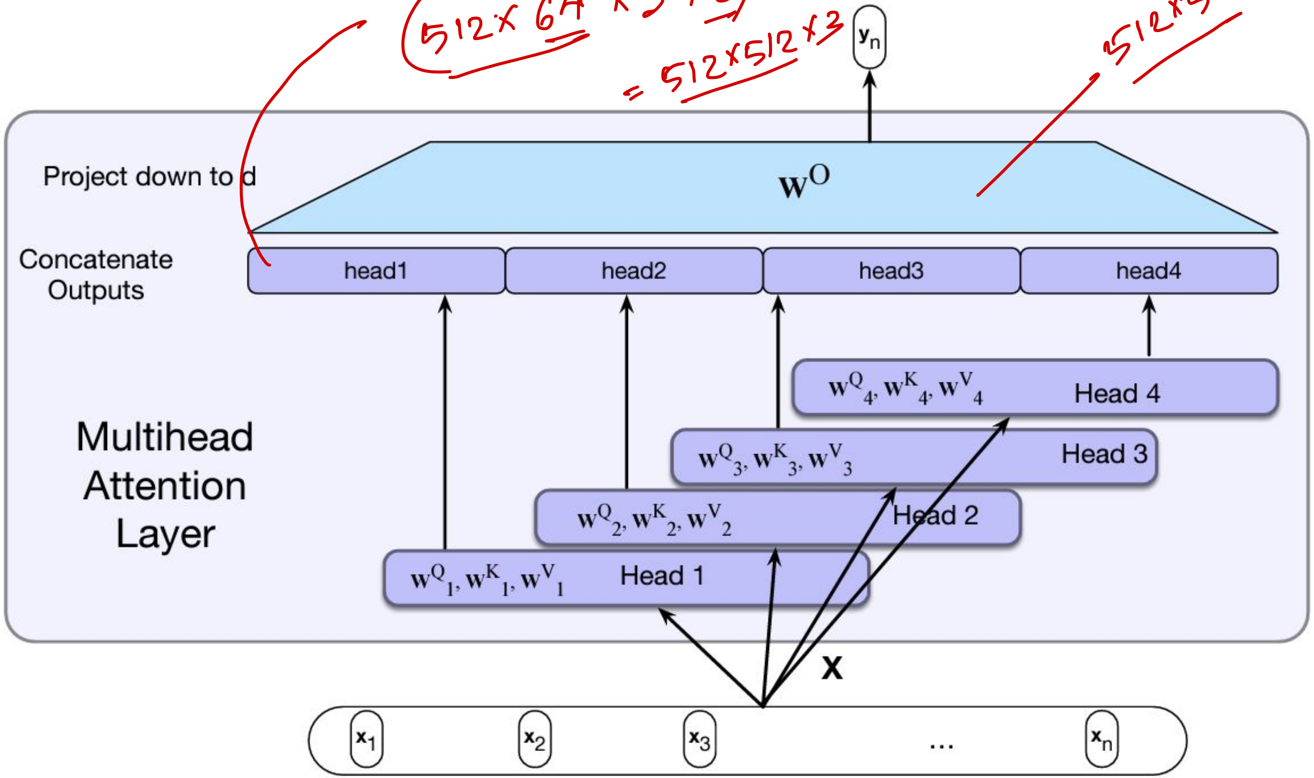
Multi-head Attention Layer

$\boxed{4 \times 512 \times 512} \approx 1\ m$

$(512 \times 64 \times 3 \times 8)$

$= 512 \times 512 \times 3$

512×512



More on dimensions

Model dimension: d (=512)

Query, Key, Value dimensions: d_q, d_k, d_v (=64 each)

Projection matrices: $d \times d_k$ (= $\boxed{512 \times 64}$ each)

The output at each head: $\boxed{d_v}$

For "h" (=8) multi-heads: hd_v

To project it back to model dimension: W^O: $d \times hd_v$

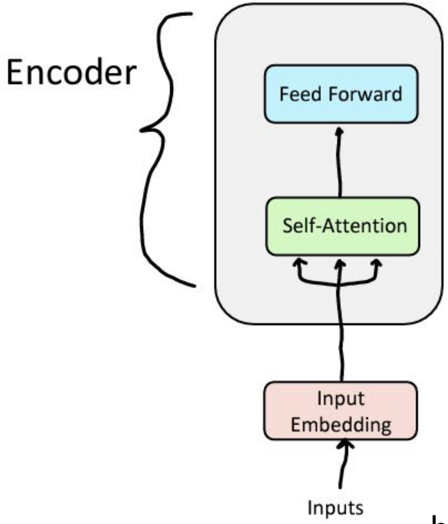
$\boxed{512 \times 512}$

https://web.stanford.edu/~jurafsky/slp3/

Feed-forward Layer

Problem: Since there are no element-wise non-linearities, self-attention is simply performing a re-averaging of the value vectors.

Easy fix: Apply a feedforward layer to the output of attention, providing non-linear activation (and additional expressive power).



https://web.stanford.edu/class/cs224n/

Feed-forward Layer

$W_1: \underline{512 \times 2048}$

$+ \; W_2: 2048 \times 512$

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

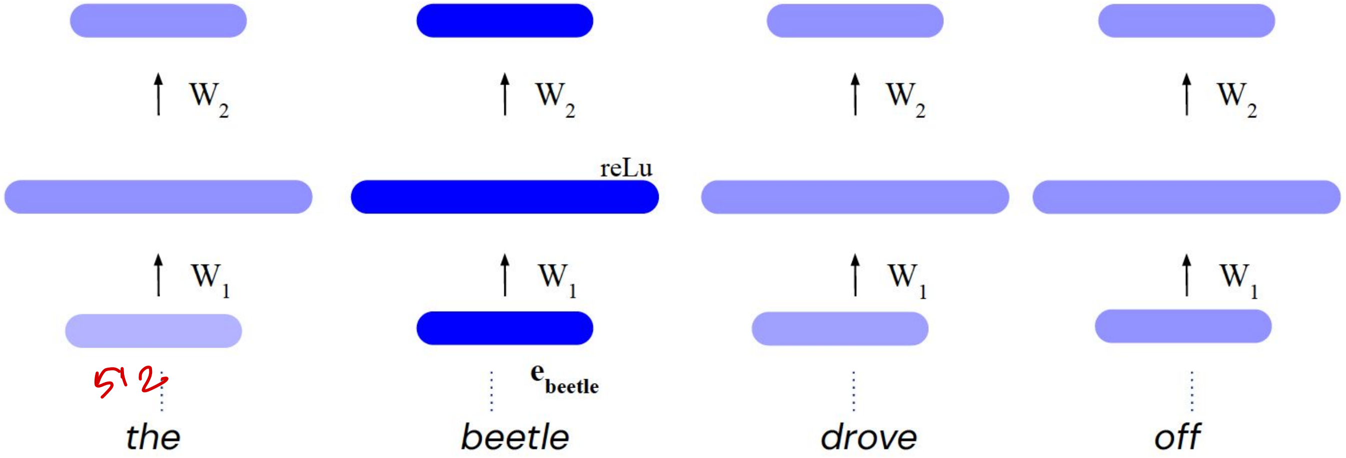
$\underline{512 \times 512} \, [4+4]$

$= 8 \times 512 \times 512$

$\underline{\approx 2m}$

512

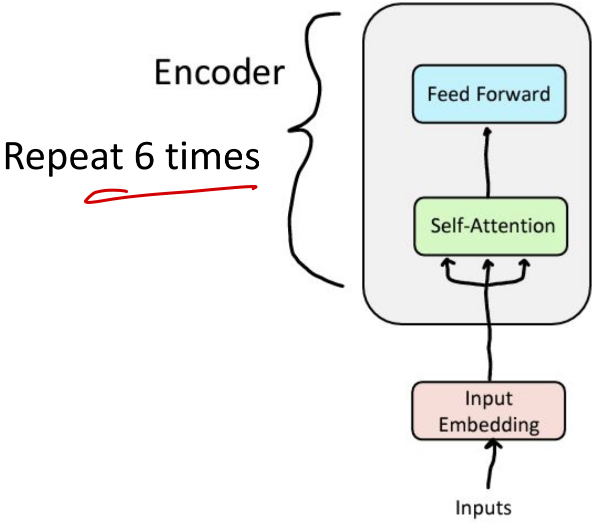
2048



Source: *https://deepmind.com/learning-resources/deep-learning-lecture-series-2020*

How to make this work for deep networks?

3m



3m

Training Trick #1: Residual Connections
Training Trick #2: LayerNorm
Training Trick #3: Scaled Dot Product Attention

https://web.stanford.edu/class/cs224n/