NPTEL ONLINE CERTIFICATION COURSES

DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING

Lecture 21 : Introduction to Transformers

PROF. PAWAN GOYAL
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

CONCEPTS COVERED

- Attention: More General Framework
- Transformer Encoder-Decoder
- Self-Attention

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	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

[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)

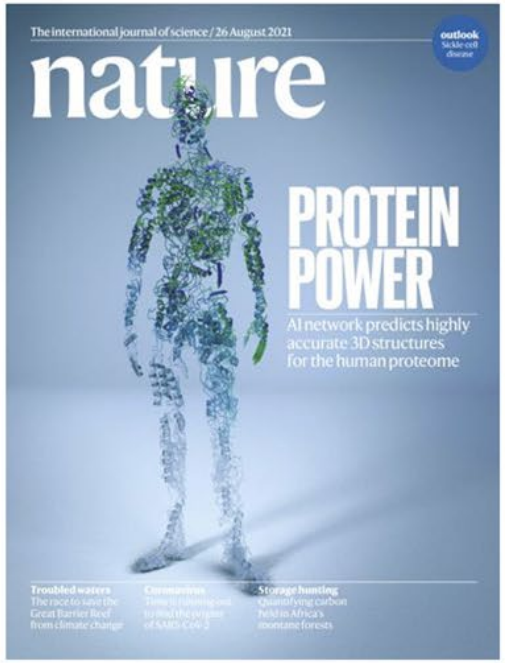
Llama 3 (Meta)

[Chiang et al., 2024]

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

Transformers have shown promise outside NLP

Protein Folding



[Jumper et al. 2021] aka AlphaFold2!

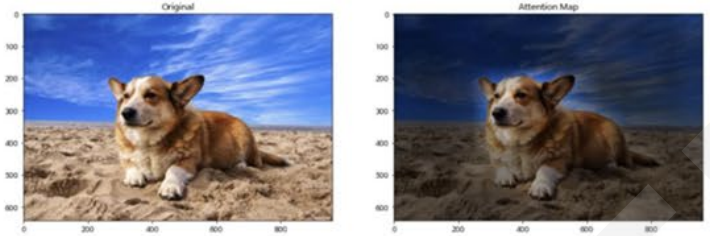
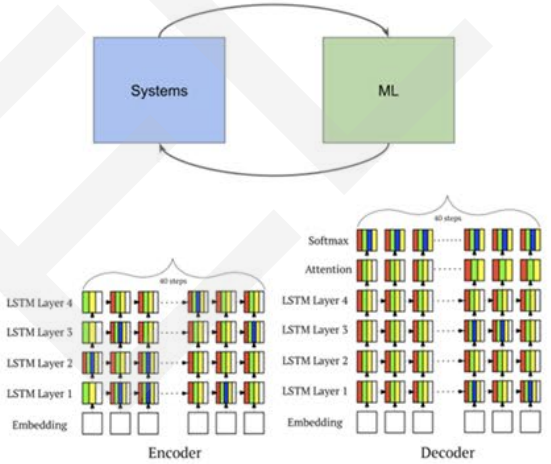


Image Classification

[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
8-layer GNMT (8)	0.440	0.562	OOM	0.693	21.7% / 36.5%	7.35x
2-layer Transformer-XL (2)	0.223	0.268	0.37	0.262	20.1% / 17.4%	40x
4-layer Transformer-XL (4)	0.230	0.27	OOM	0.259	17.4% / 12.6%	26.7x
8-layer Transformer-XL (8)	0.350	0.46	OOM	0.425	23.9% / 16.7%	16.7x
Inception (2) b32	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/

Recap: Attention is a general technique

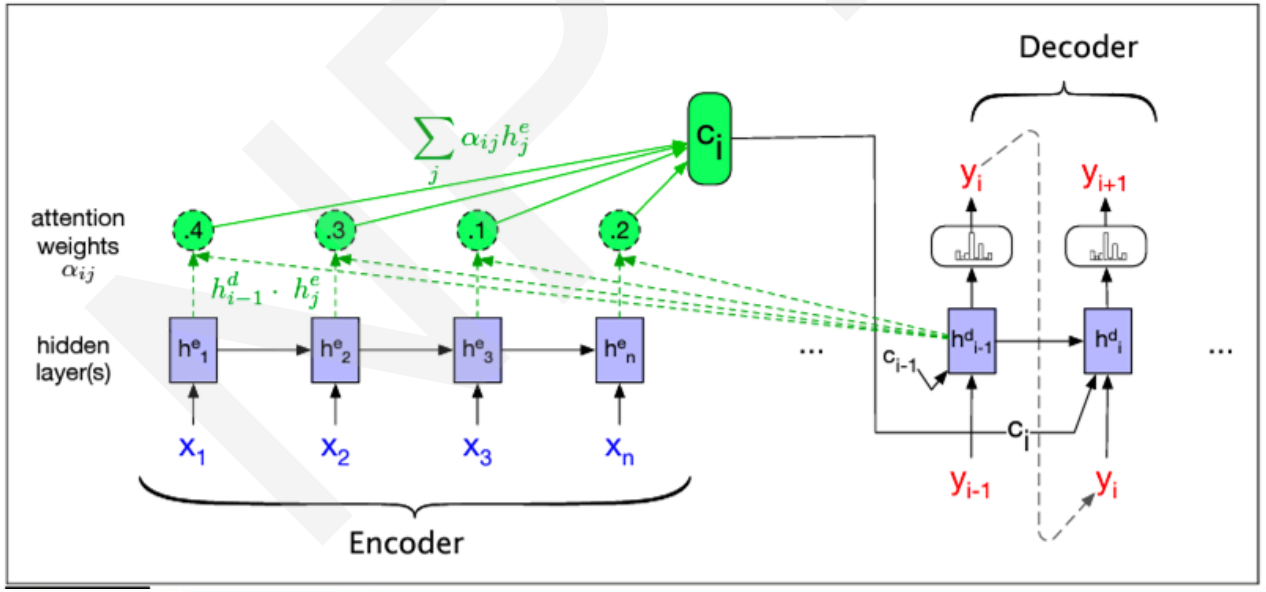
We can use attention in many architectures not just seq2seq and many tasks (not just MT)

More general definition of attention:

➔ Given a set of vector **values**, and a vector **query**, **attention** is a technique to compute a weighted sum of the values, dependent on the query

We say that the **query attends to the values**

In the seq2seq+attention model, each decoder hidden state (query) *attends to* all the encoder hidden state (values)



Attention is a general deep learning technique

More general definition of attention:

➔ Given a set of vector **values**, and a vector **query**, **attention** is a technique to compute a weighted sum of the values, dependent on the query

Intuition:

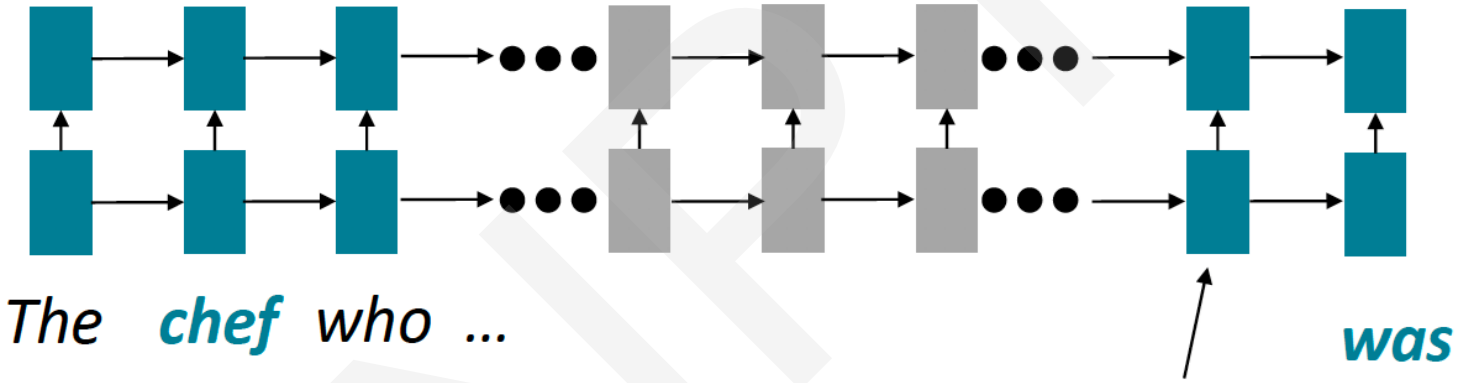
- The weighted sum is a **selective summary** of the information contain in the values, where the query determines which values to focus on
- Attention is a way to obtain a **fixed-size representation of an arbitrary set of representations**, *dependent on some other representation*

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

Issues with recurrent models

O(sequence length) steps for distant word pairs

- Hard to learn long-distance dependencies (gradient problems!)
- Linear order of words is "baked in"; not the right way to think about sentences



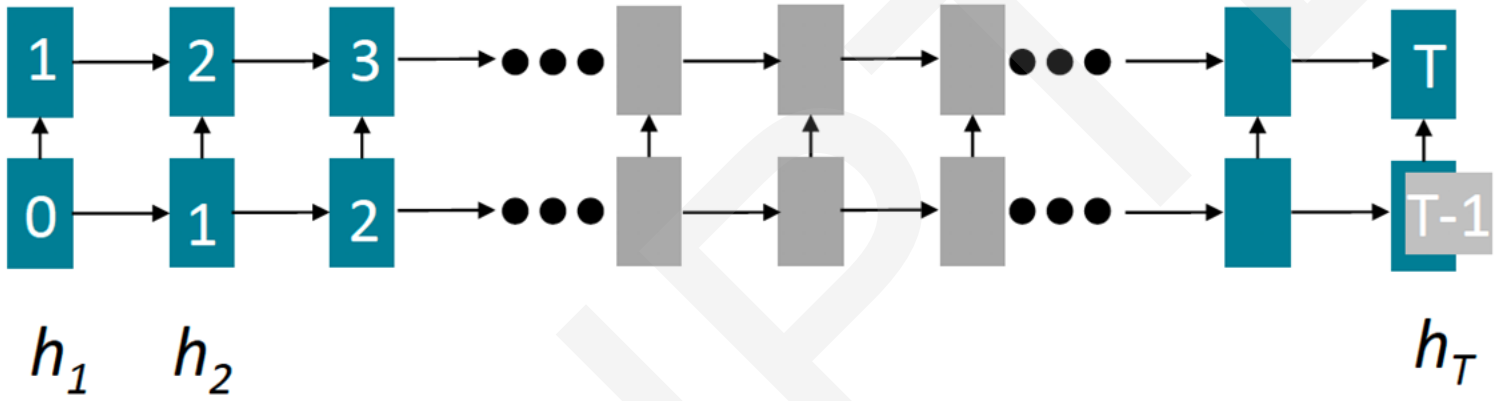
Info of ***chef*** has gone through O(sequence length) many layers!

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

Issues with recurrent models

Lack of parellilizability

- Future RNN hidden states can't be computed in full before past RNN hidden states have been computed



Inhibits training on very large datasets!

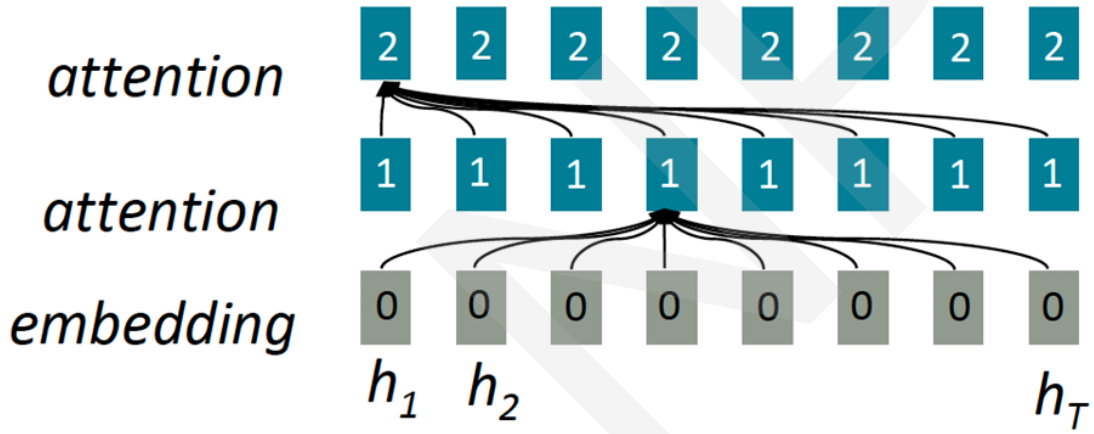
Numbers indicate min # of steps before a state can be computed

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

If not recurrence, then what?

Attention

- Given a word as query, attention can be used to access and incorporate information from a set of values (other words)
- Can we do this within a single sentence?
- All words can interact with each other and computation can be done in parallel!!



All words attend to all words in previous layer; most arrows here are omitted

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

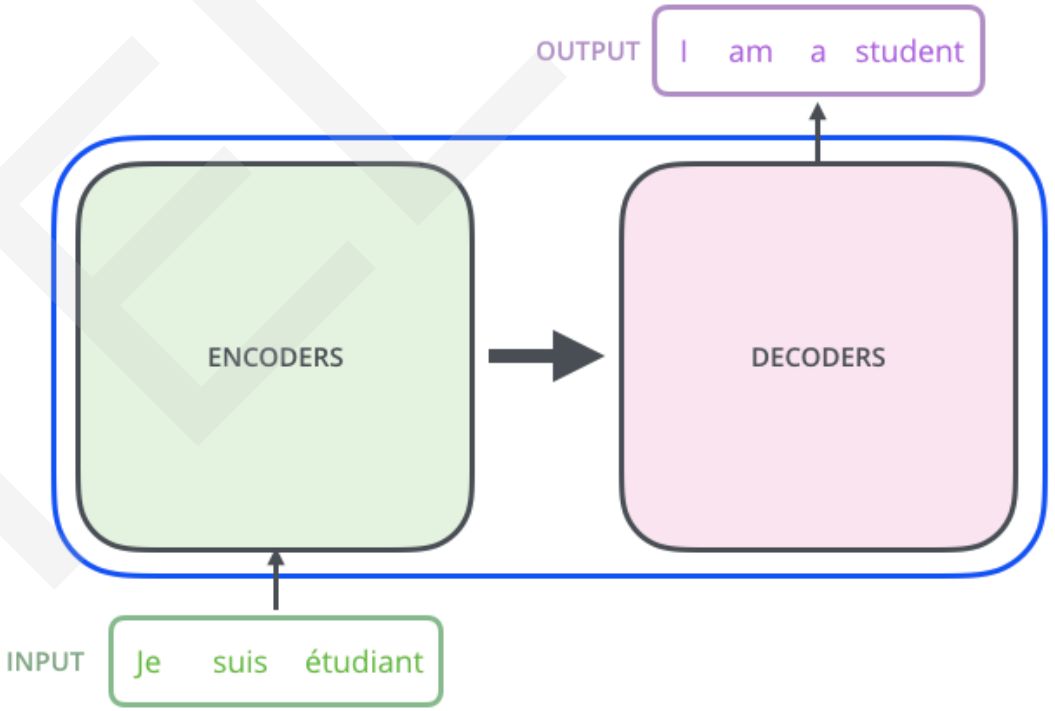
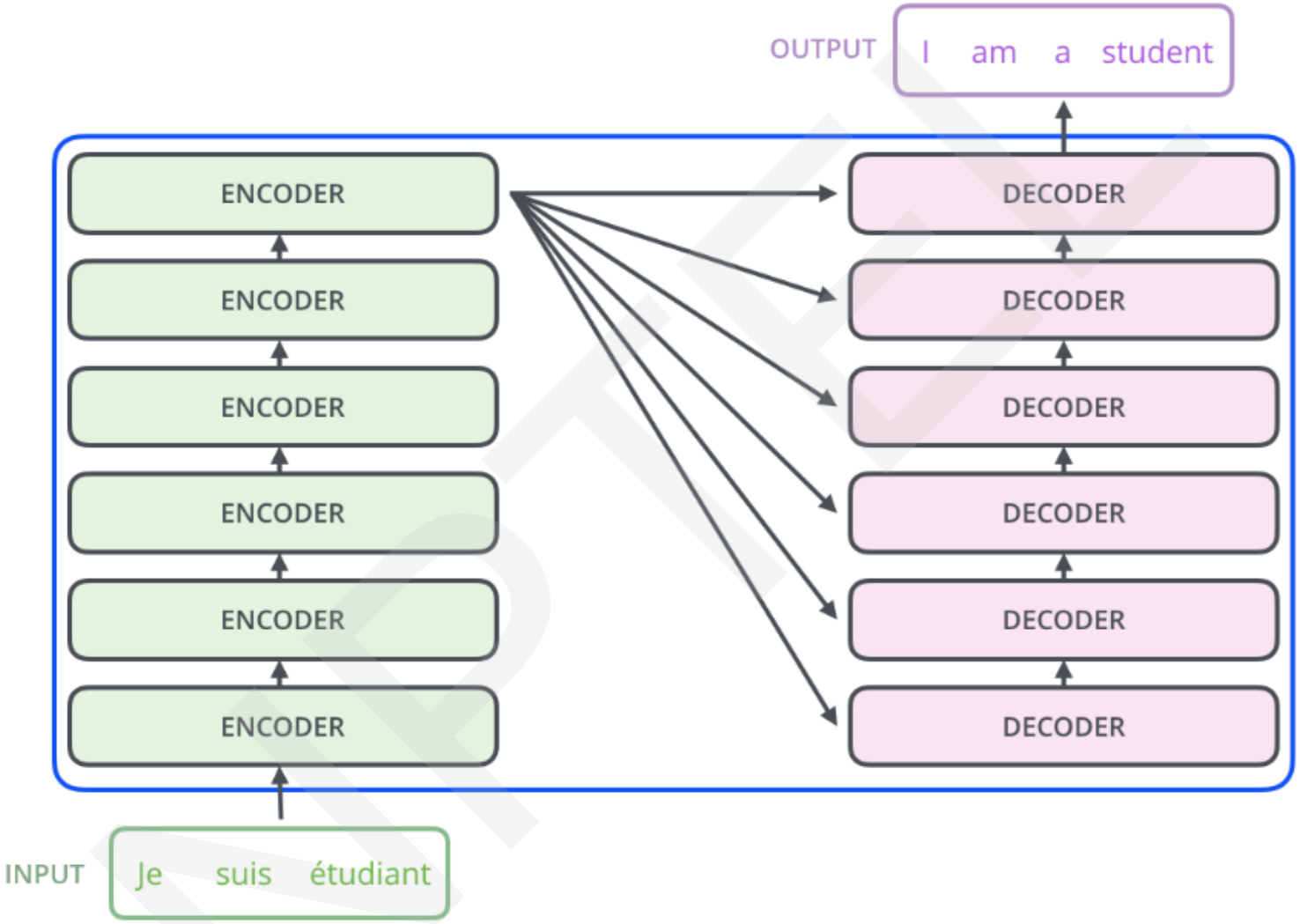


Figure: Jay Alammar

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

A stack of encoder blocks



A stack of decoder blocks

Figure: Jay Alammar

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

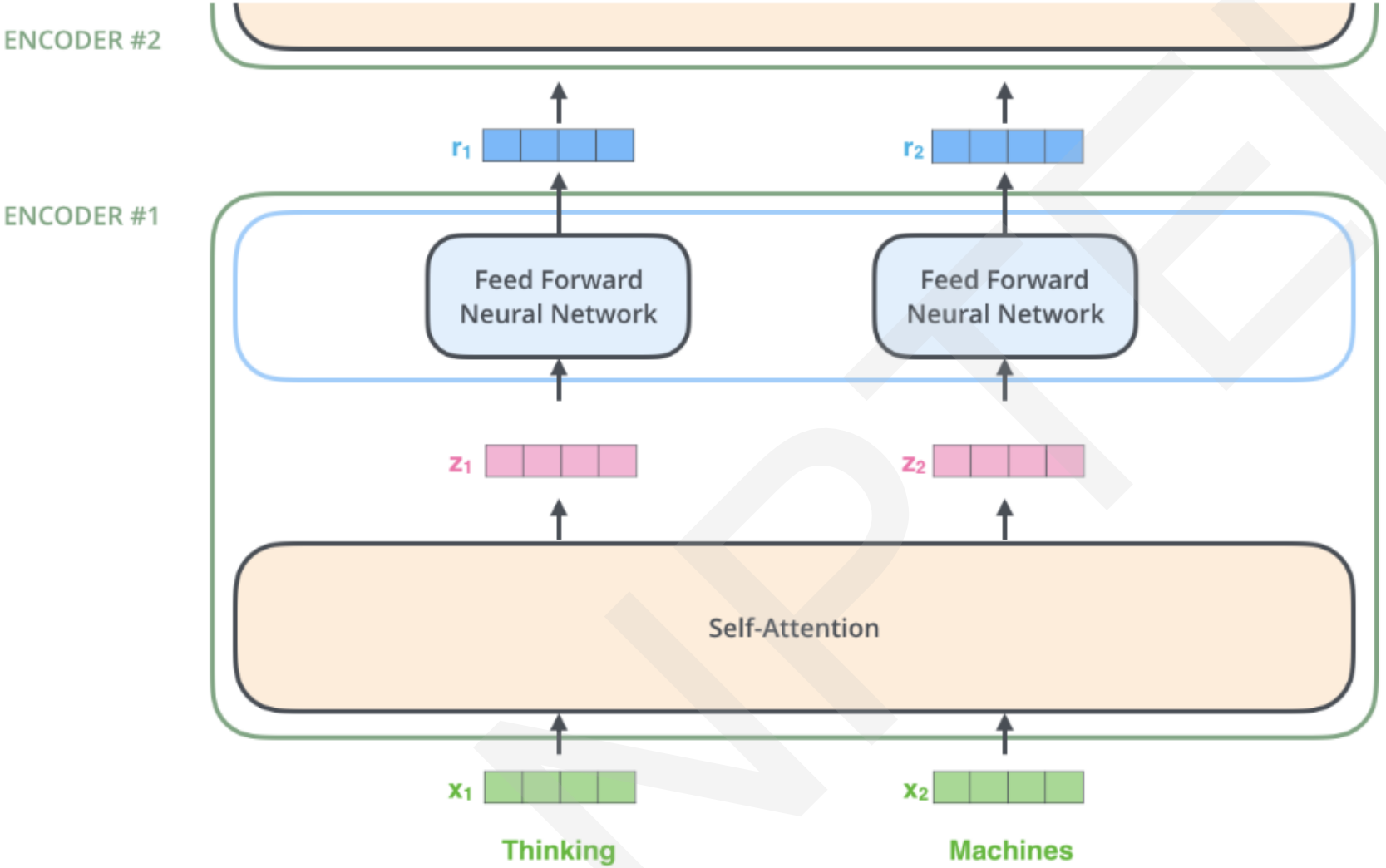


Figure: Jay Alammar

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 Self-Attention: 1

Problem with static embeddings (word2vec)

They are static! The embedding for a word doesn't reflect how its meaning changes in context.

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

What is the meaning represented in

the static embedding for "it"?

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

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**

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

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 animal or the street

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

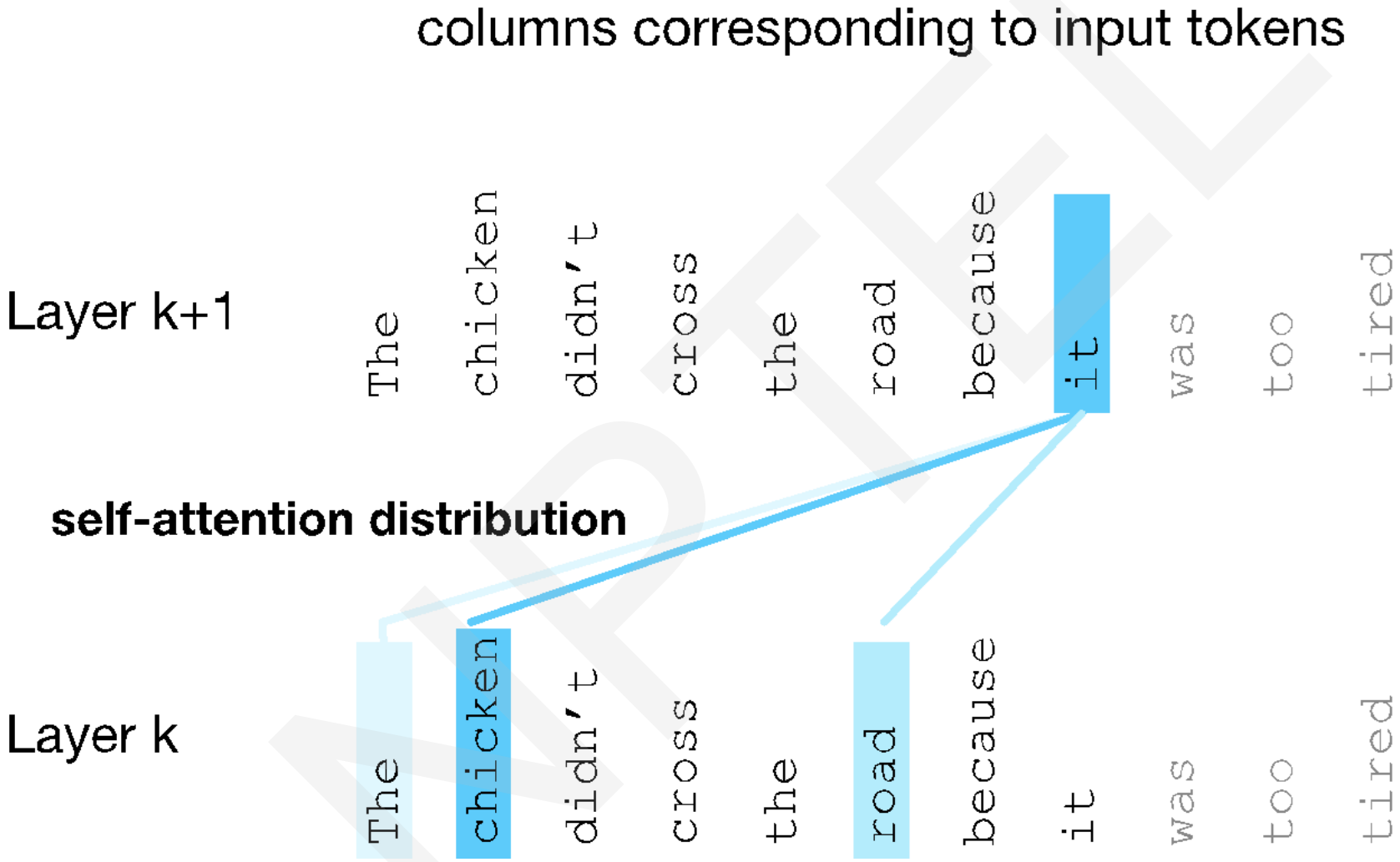
Intuition of attention

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/

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

Given a sequence of token embeddings:

$$\mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_3 \quad \mathbf{x}_4 \quad \mathbf{x}_5 \quad \mathbf{x}_6 \quad \mathbf{x}_7 \quad \mathbf{x}_i$$

Produce: $\mathbf{a}_i$ = a weighted sum of $\mathbf{x}_1$ through $\mathbf{x}_7$ (and $\mathbf{x}_i$)
Weighted by their similarity to $\mathbf{x}_i$

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

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

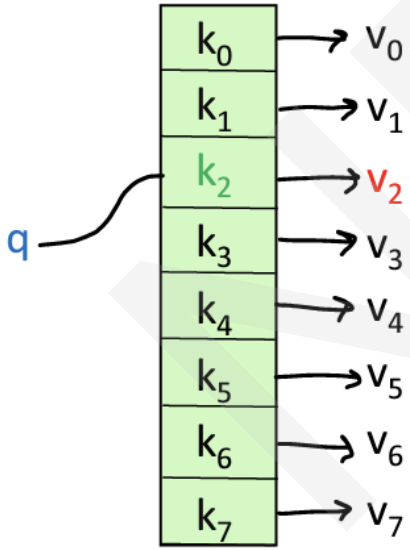
$$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$$

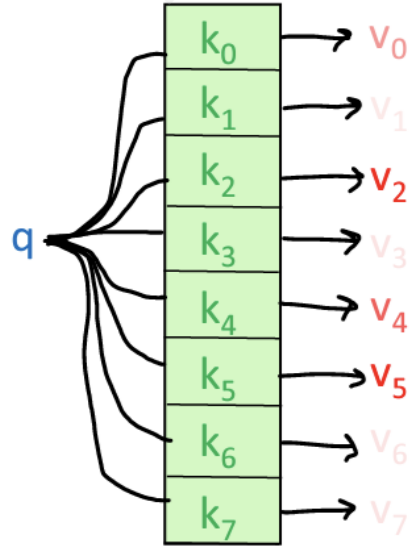
https://web.stanford.edu/~jurafsky/slp3/

Intuition for Self-attention: 2

Let's think of attention as a "fuzzy" or approximate hashtable:

- To look up a **value**, we compare a **query** against **keys** in a table.
- In a hashtable (shown on the bottom left):
  - Each **query** (hash) maps to exactly one **key-value** pair.
- In (self-)attention (shown on the bottom right):
  - Each **query** matches each **key** to varying degrees.
  - We return a sum of **values** weighted by the **query-key** match.





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

Intuition for Self-attention: 3

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).

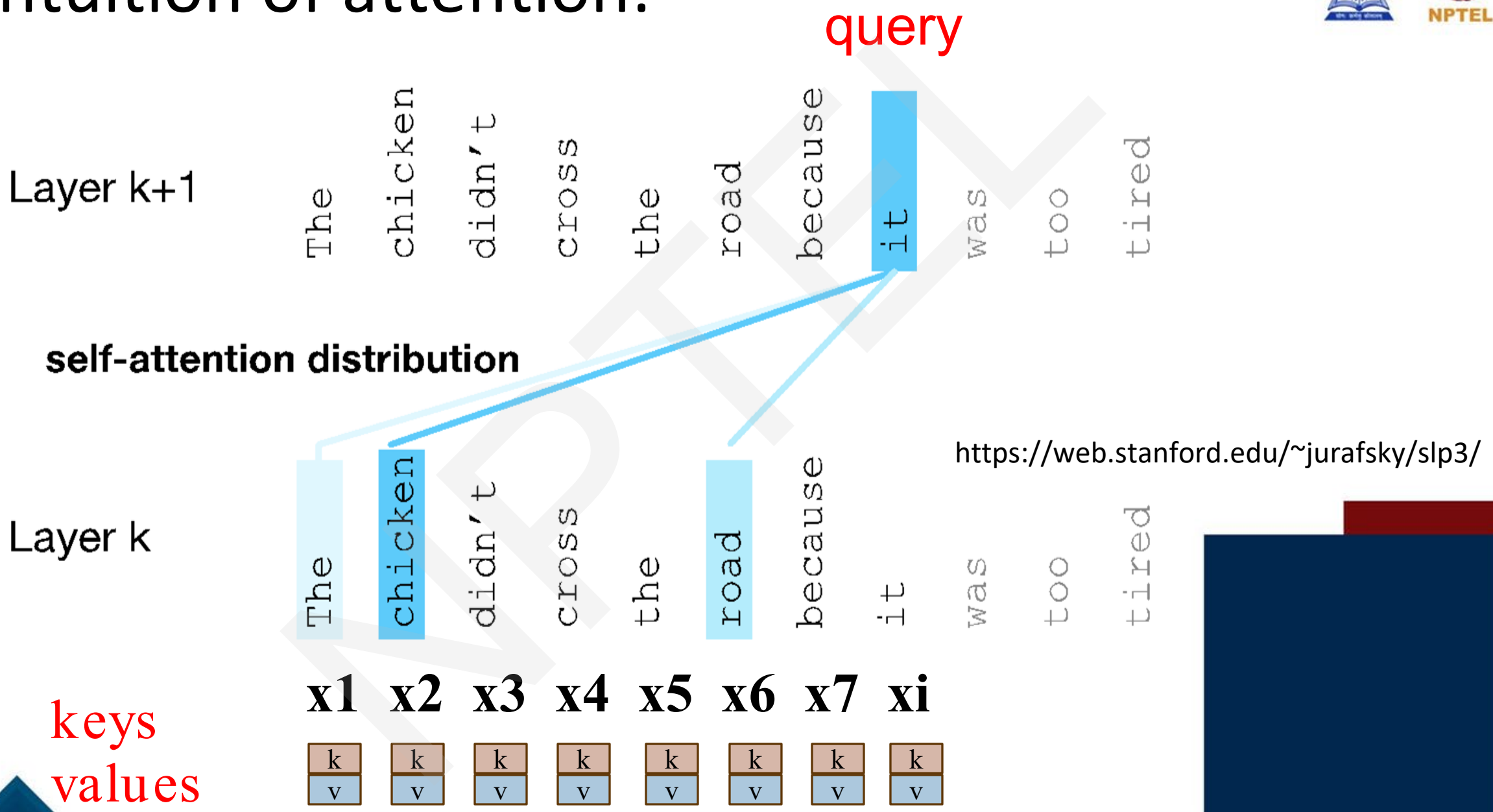
An Actual Attention Head

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

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

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

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

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

An Actual Attention Head: slightly more complicated

Given these 3 representation of x_i

To compute similarity of current element x_i with some prior element x_j

We'll use dot product between q_i and k_j.

And instead of summing up x_j, we'll sum up v_j

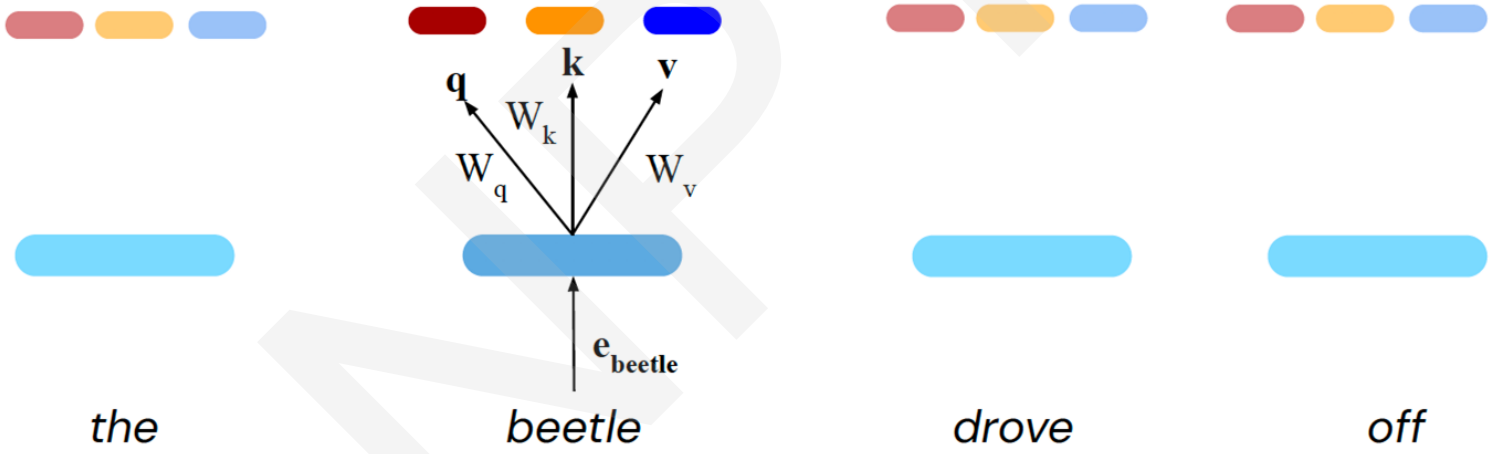
$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^{\mathbf{Q}}; \quad \mathbf{k}_i = \mathbf{x}_i \mathbf{W}^{\mathbf{K}}; \quad \mathbf{v}_i = \mathbf{x}_i \mathbf{W}^{\mathbf{V}}$$

Transformers: Self-attention over input

$$\mathbf{q} = \mathbf{e}_{beetle}\, W_q$$

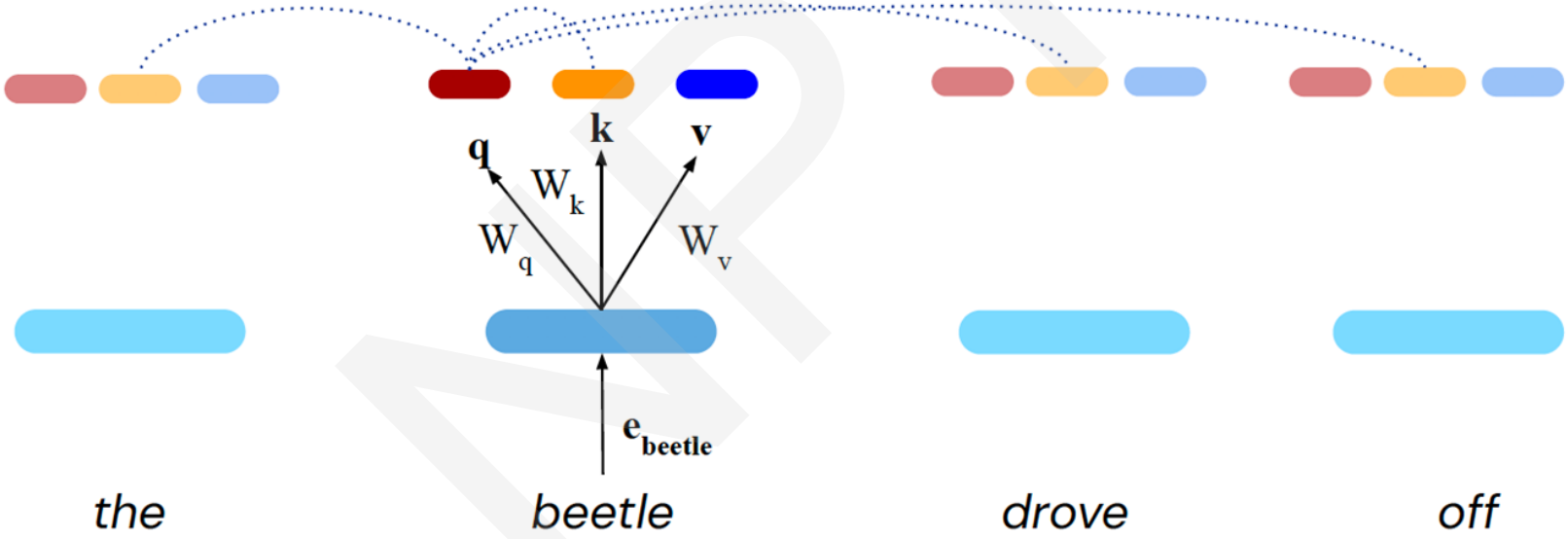
$$\mathbf{k} = \mathbf{e}_{beetle}\, W_k$$

$$\mathbf{V} = \mathbf{e}_{beetle}\, W_v$$



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

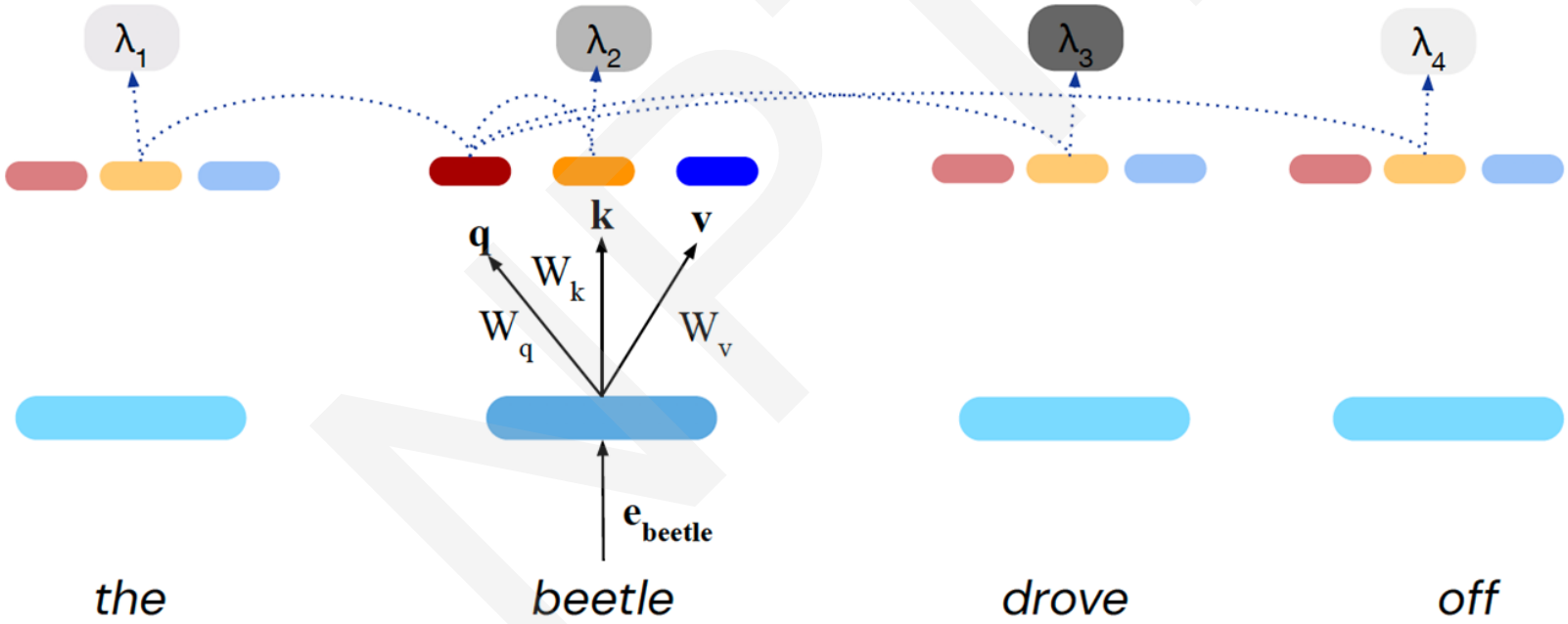
Self-attention over input embeddings



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

Self-attention over input embeddings

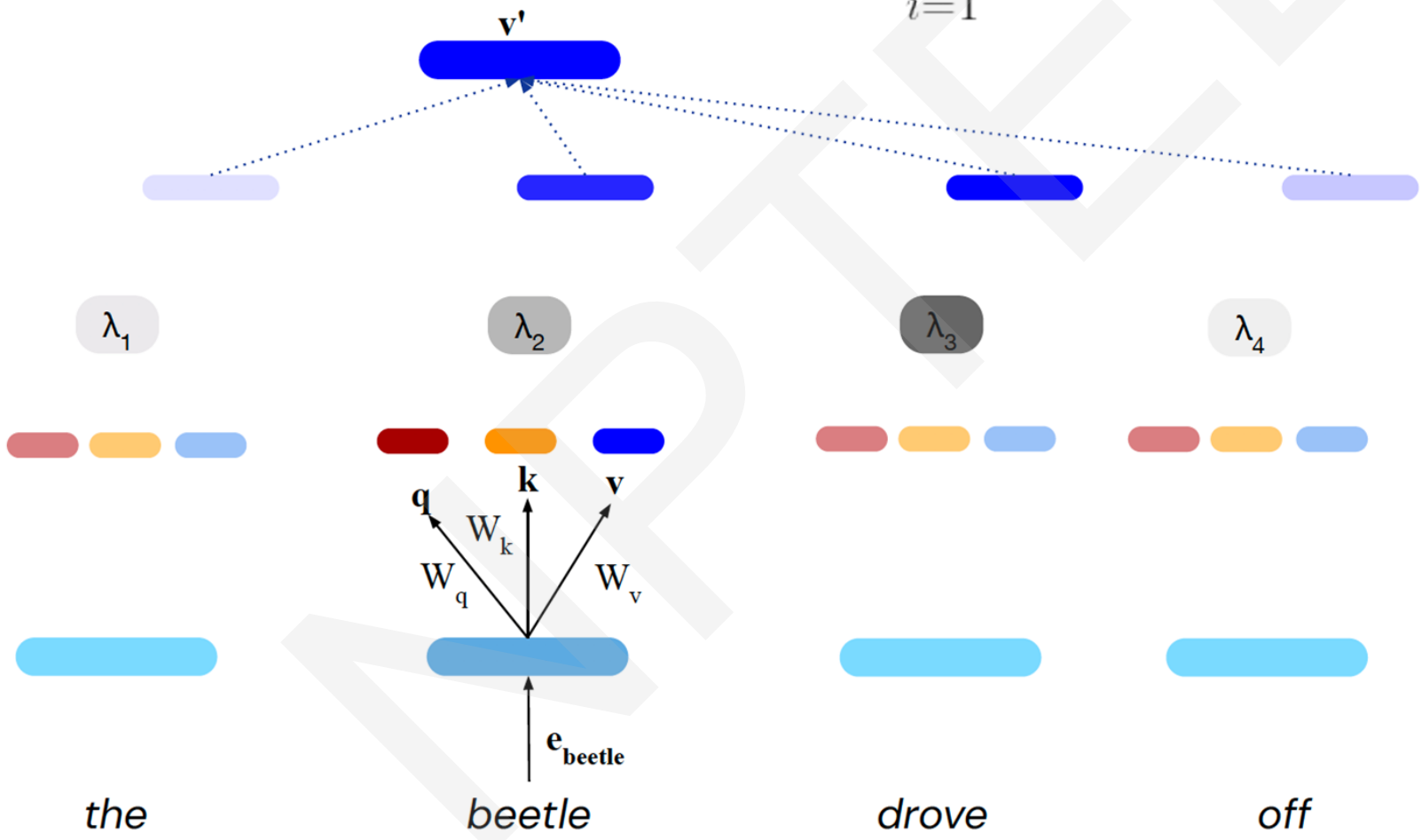
$$\lambda_i = \frac{e^{\mathbf{q}\cdot\mathbf{k_i}}}{\sum_{i=1}^{4} e^{\mathbf{q}\cdot\mathbf{k_i}}}$$



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

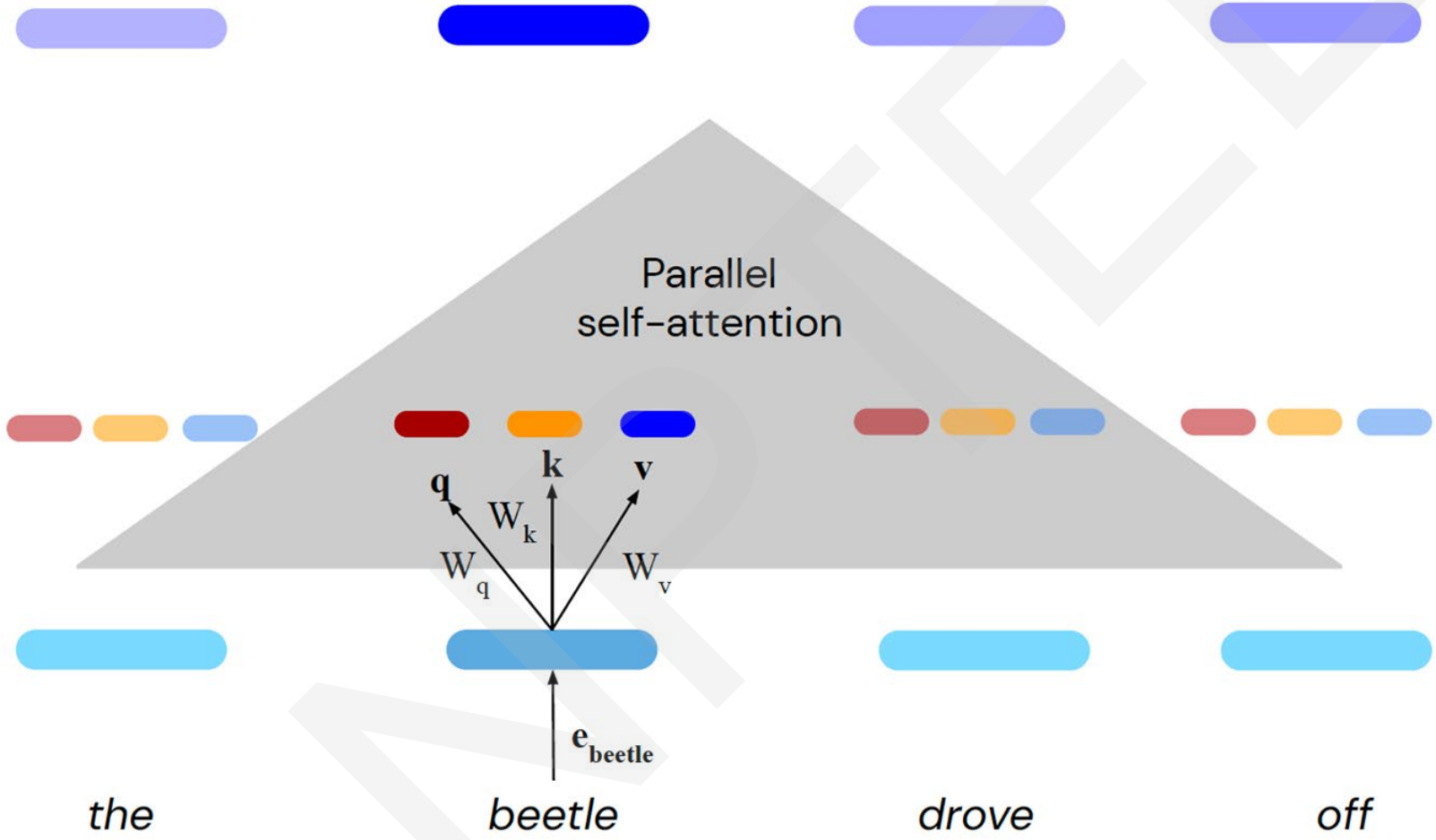
Self-attention over input embeddings

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



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*

REFERENCES

Daniel Jurafsky and James H. Martin. 2024. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models, 3rd edition. Online manuscript released August 20, 2024. [Chapter 9]


THANK YOU

NPTEL ONLINE CERTIFICATION COURSES

DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING

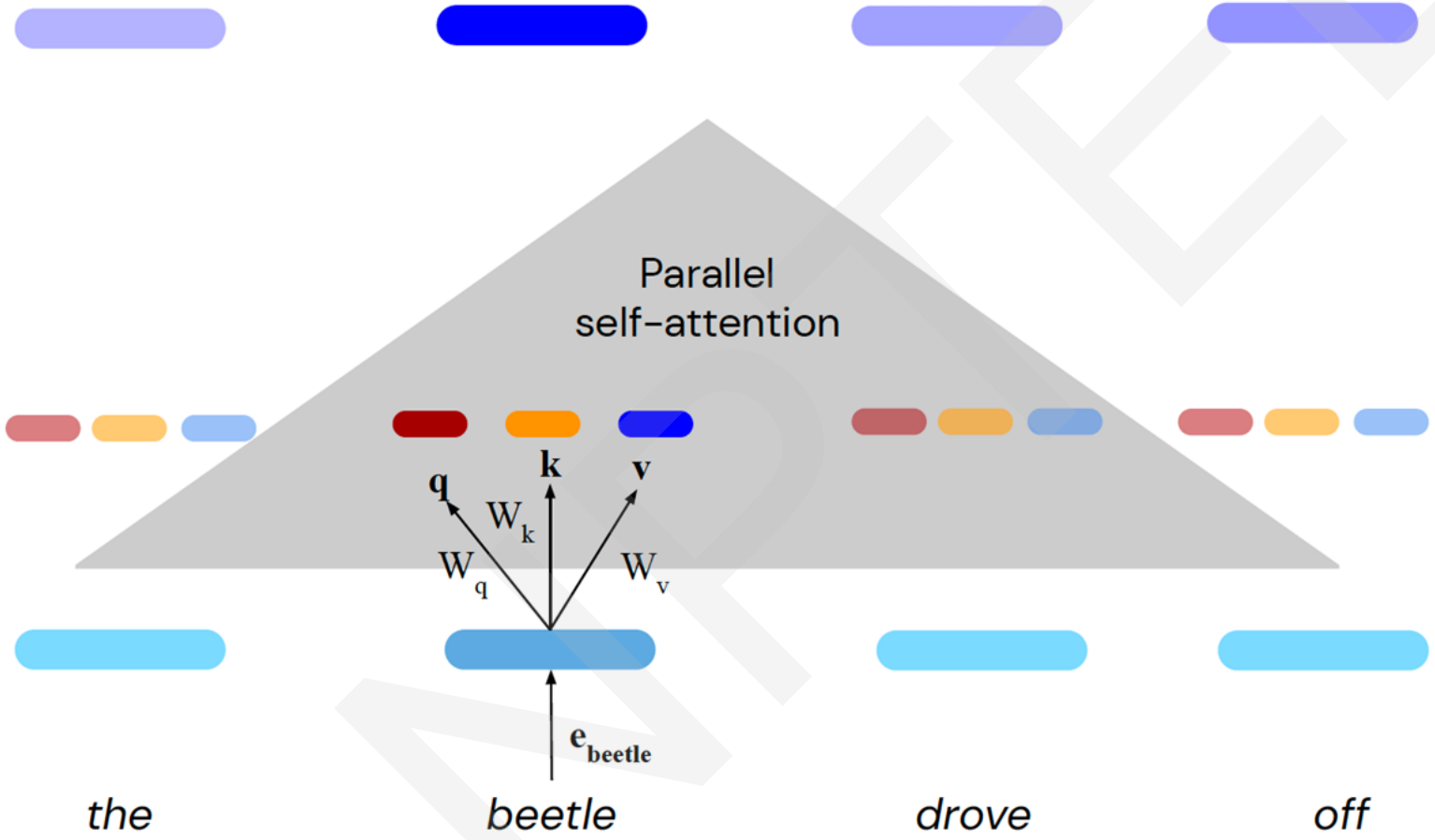
Lecture 22 : Transformers: Part 2

PROF. PAWAN GOYAL
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

CONCEPTS COVERED

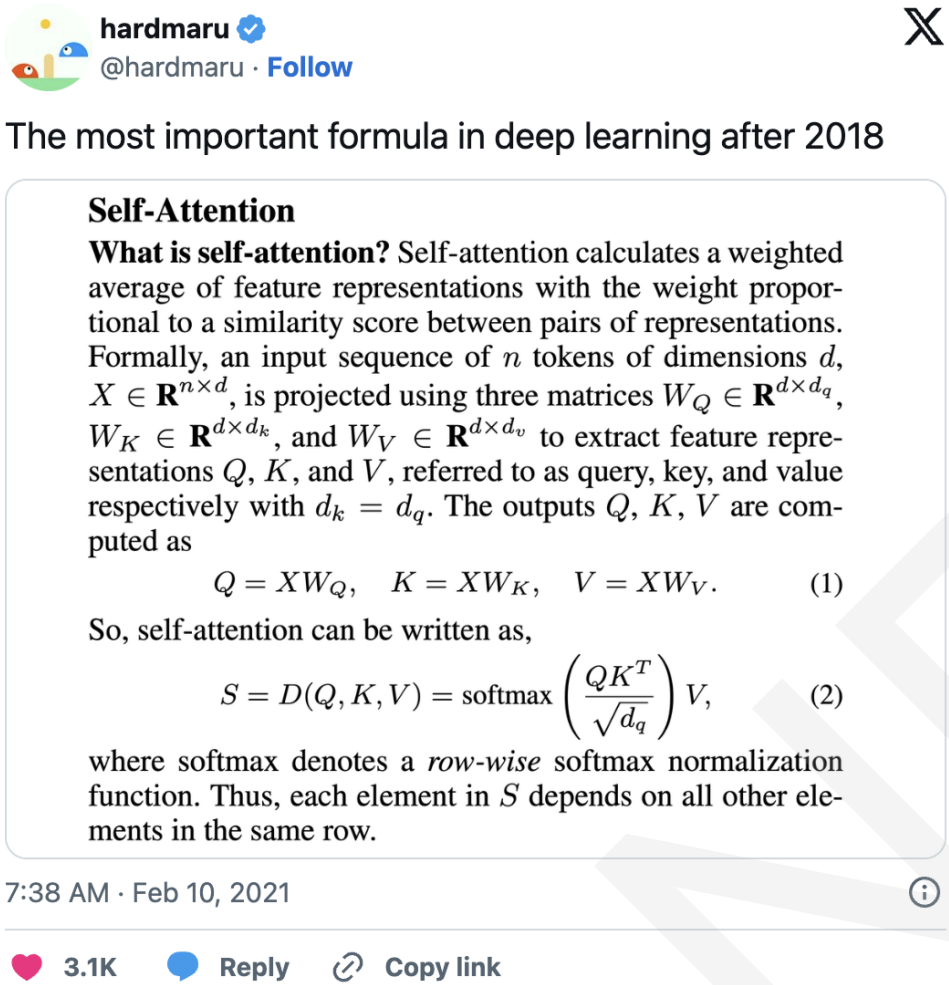
- Self-Attention: More Details
- Multi-head attention
- Feed-forward Network
- Complete Transformer Block

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

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

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

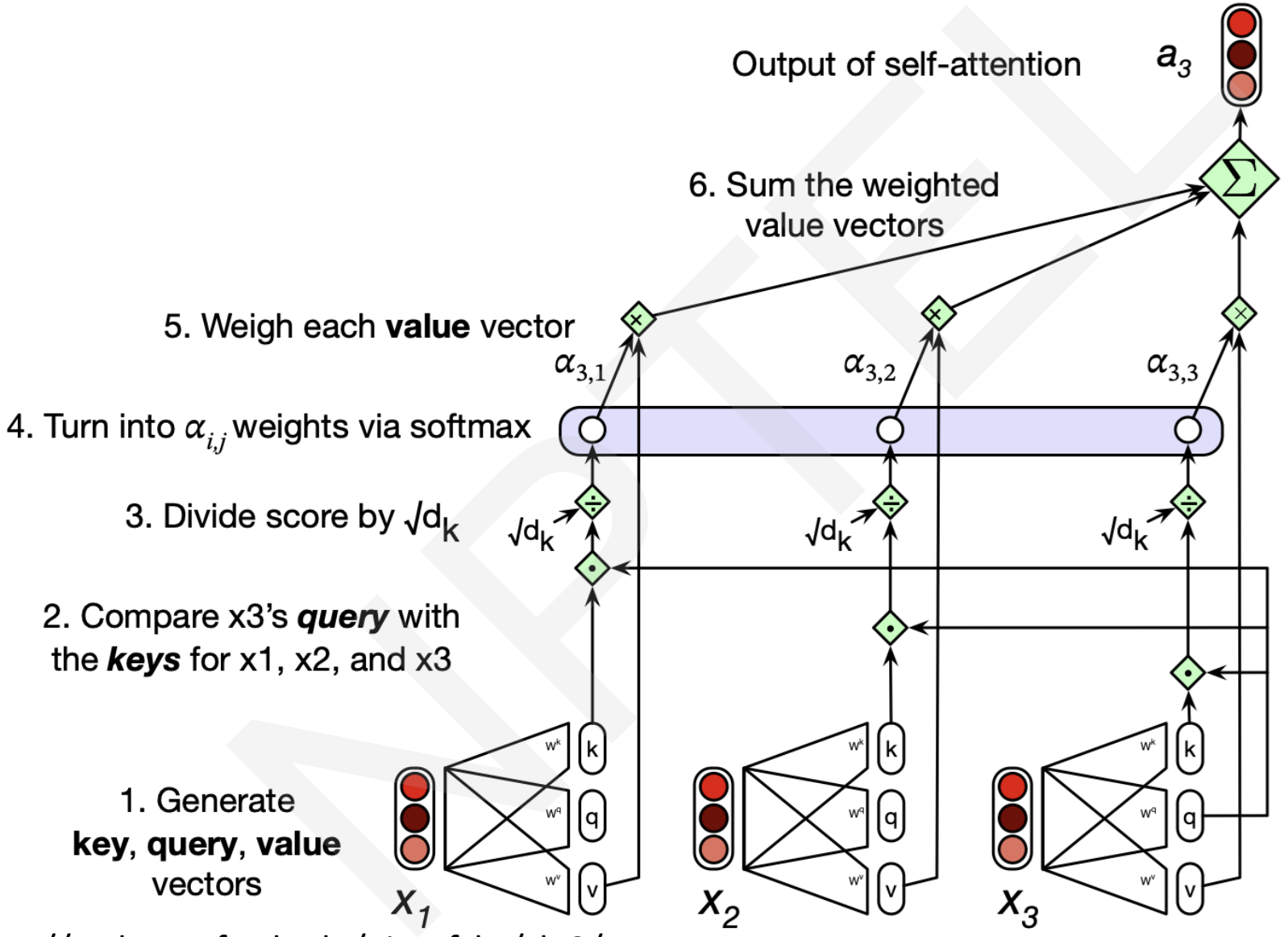
$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j))$$

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

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/

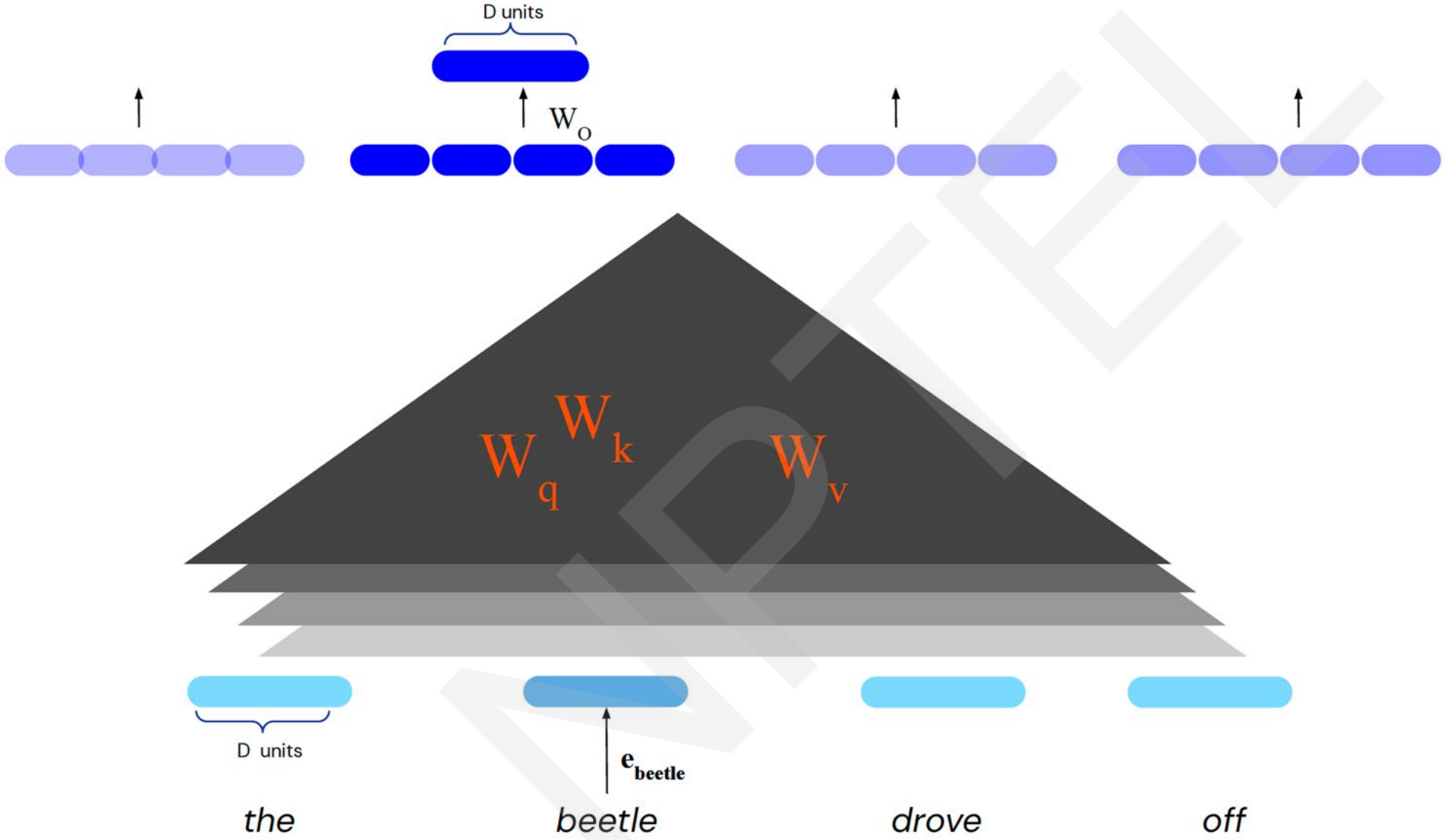
Try this problem

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.

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

$$\mathbf{q}_i^c = \mathbf{x}_i \mathbf{W}^{\mathbf{Qc}}; \quad \mathbf{k}_j^c = \mathbf{x}_j \mathbf{W}^{\mathbf{Kc}}; \quad \mathbf{v}_j^c = \mathbf{x}_j \mathbf{W}^{\mathbf{Vc}}; \quad \forall\, c \;\; 1 \le c \le h$$

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

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

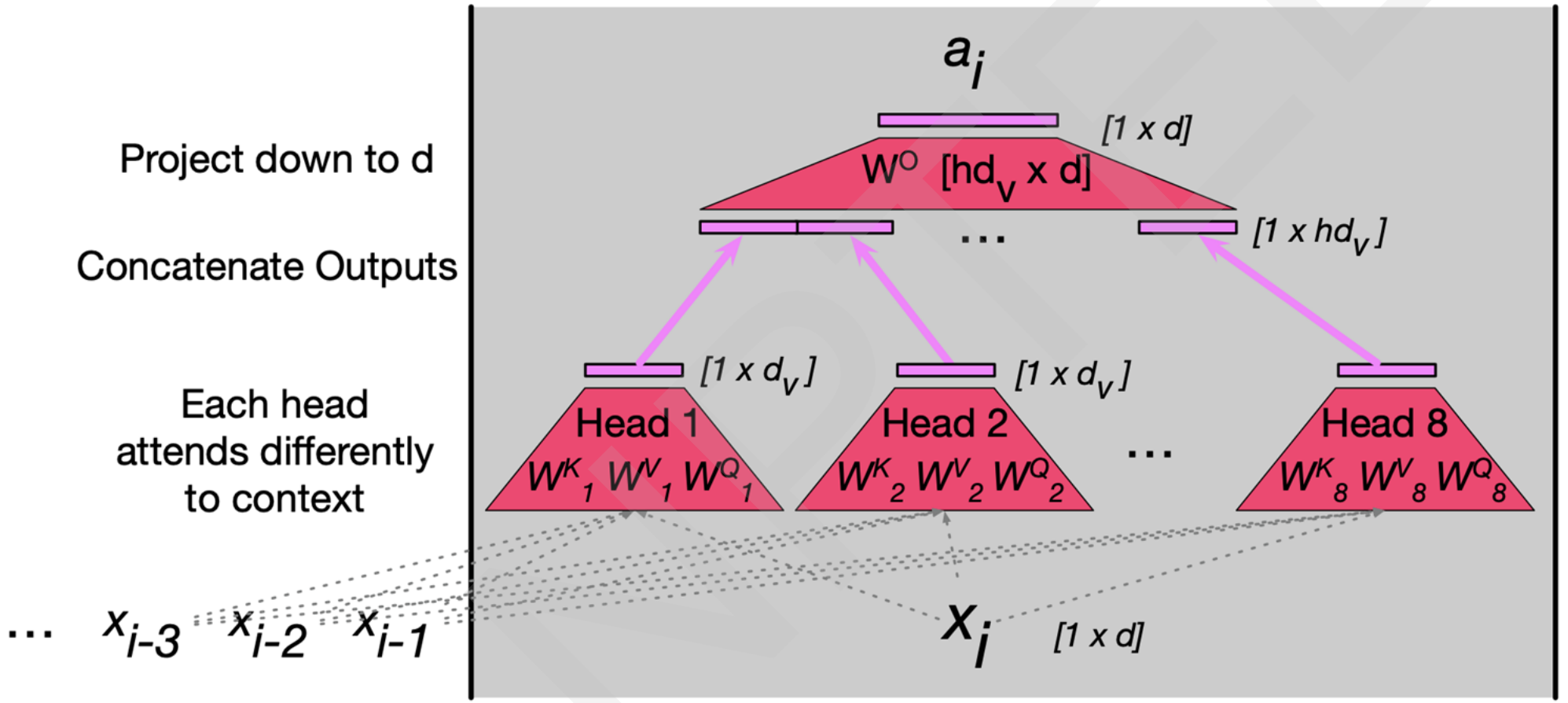
$$\mathbf{head}_i^c = \sum \alpha_{ij}^c \mathbf{v}_j^c$$

$$\mathbf{a}_i = (\mathbf{head}^1 \oplus \mathbf{head}^2 \ldots \oplus \mathbf{head}^h)\mathbf{W}^O$$

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

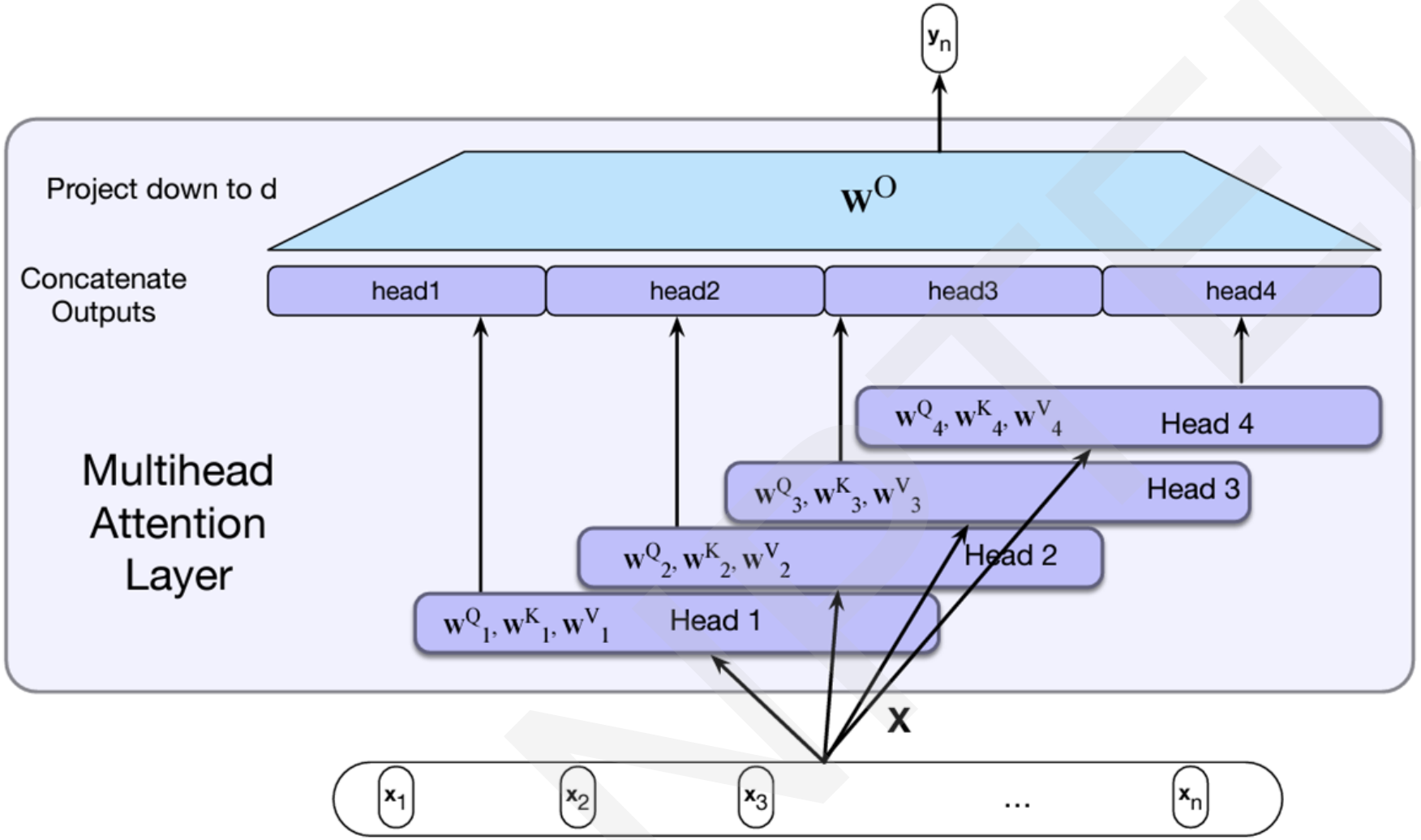
https://web.stanford.edu/~jurafsky/slp3/

Multi-head attention



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

Multi-head Attention Layer



More on dimensions

Model dimension: d (=512)

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

Projection matrices: $d \times d_k$ (= 512 x 64 each)

The output at each head: d_v

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

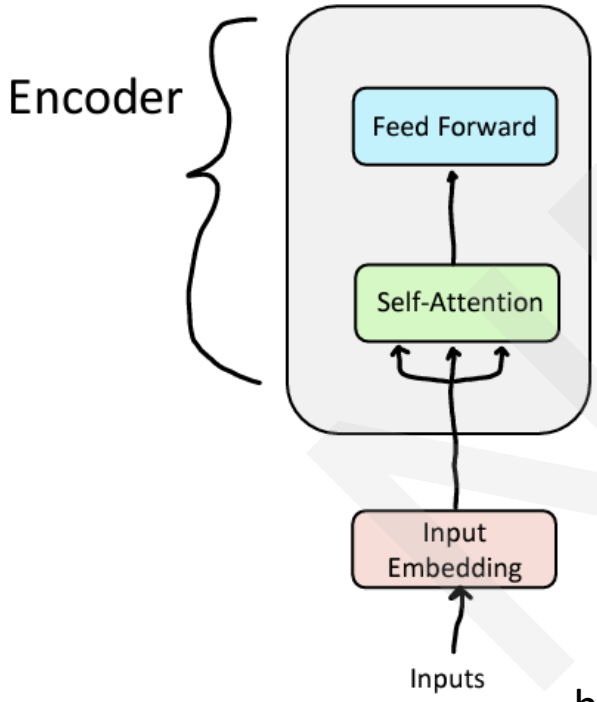
To project it back to model dimension: W^O: $d \times hd_v$

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.

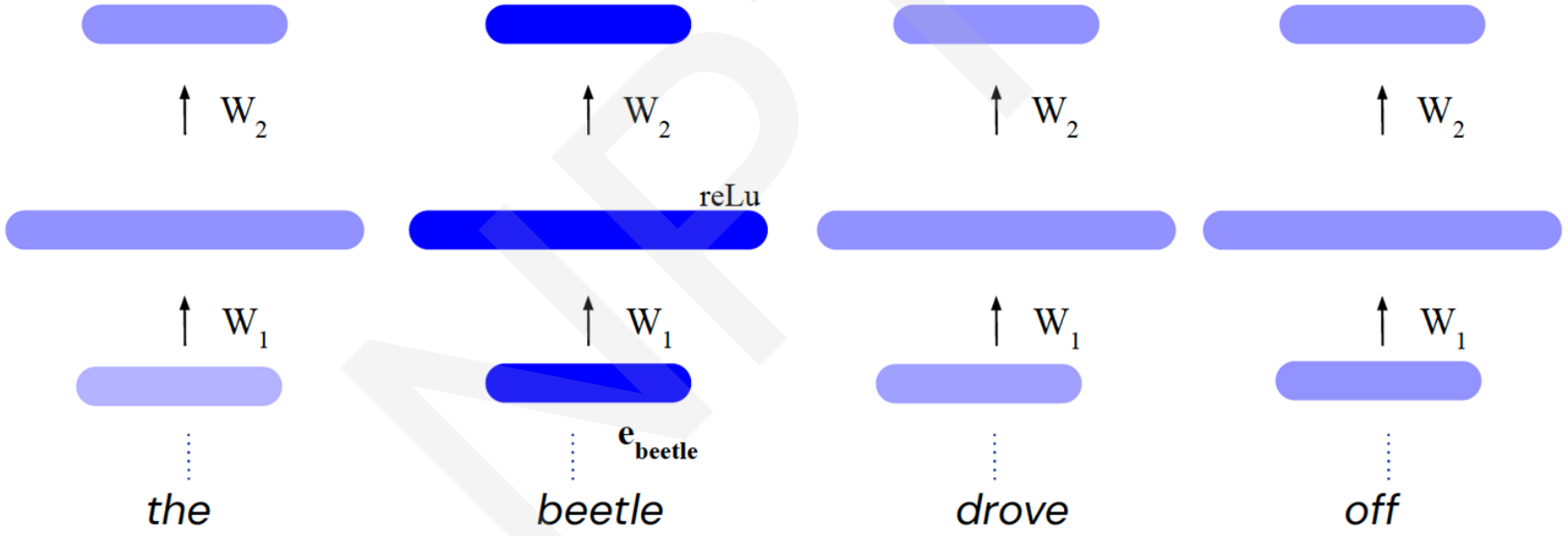
Easy fix: Apply a feedforward layer to the output of attention, providing non-linear activation (and additional expressive power).



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

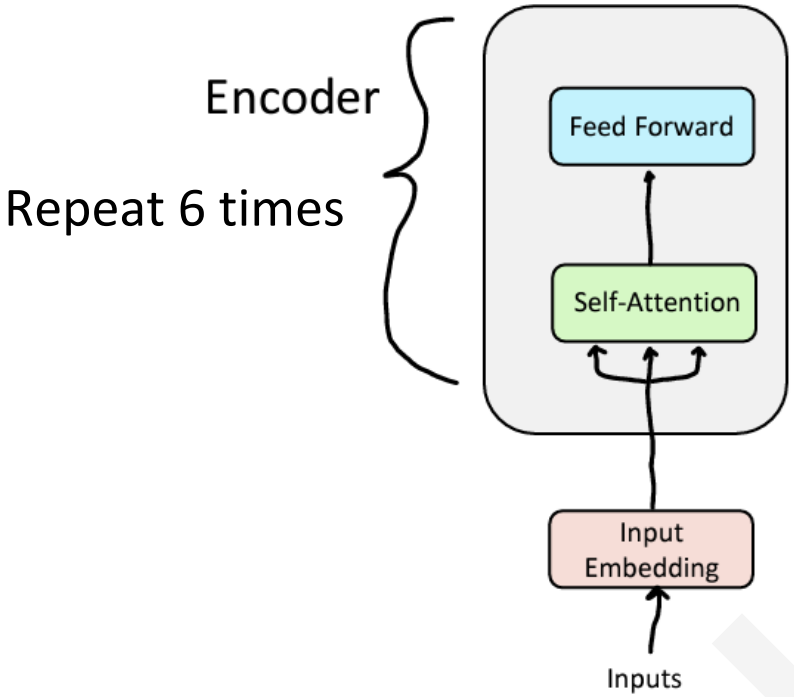
Feed-forward Layer

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



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

How to make this work for deep networks?



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

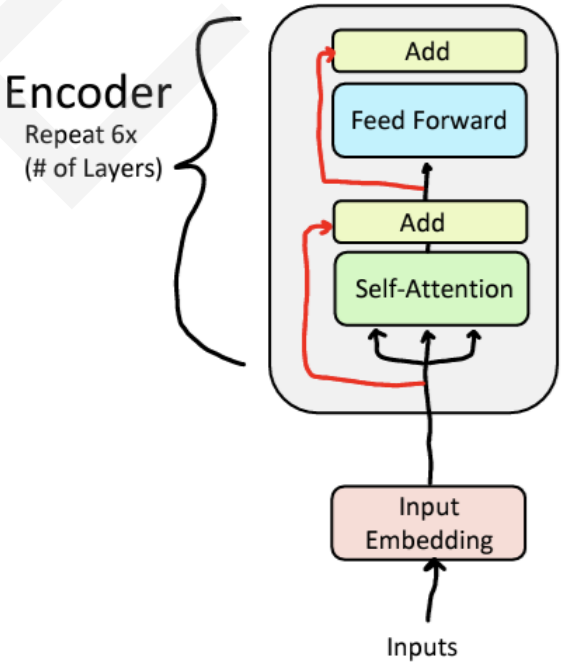
https://web.stanford.edu/class/cs224n/

Training Trick #1: Residual Connections

- Residual connections are a simple but powerful technique from computer vision.
- Deep networks are surprisingly bad at learning the identity function!
- Therefore, directly passing "raw" embeddings to the next layer can actually be very helpful!

$$x_\ell = F(x_{\ell-1}) + x_{\ell-1}$$

- This prevents the network from "forgetting" or distorting important information as it is processed by many layers.



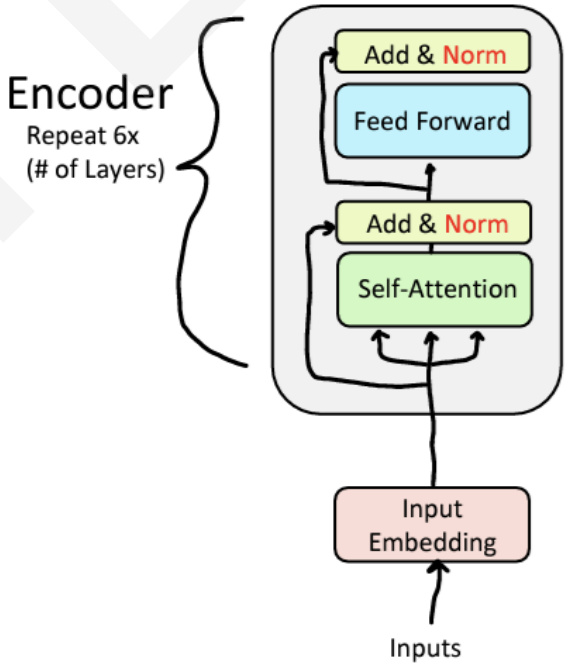
https://web.stanford.edu/class/cs224n/

Training Trick #2: Layer Normalization

- **Problem:** Difficult to train the parameters of a given layer because its input from the layer beneath keeps shifting.
- **Solution:** Reduce variation by **normalizing** to zero mean and standard deviation of one within each **layer**.

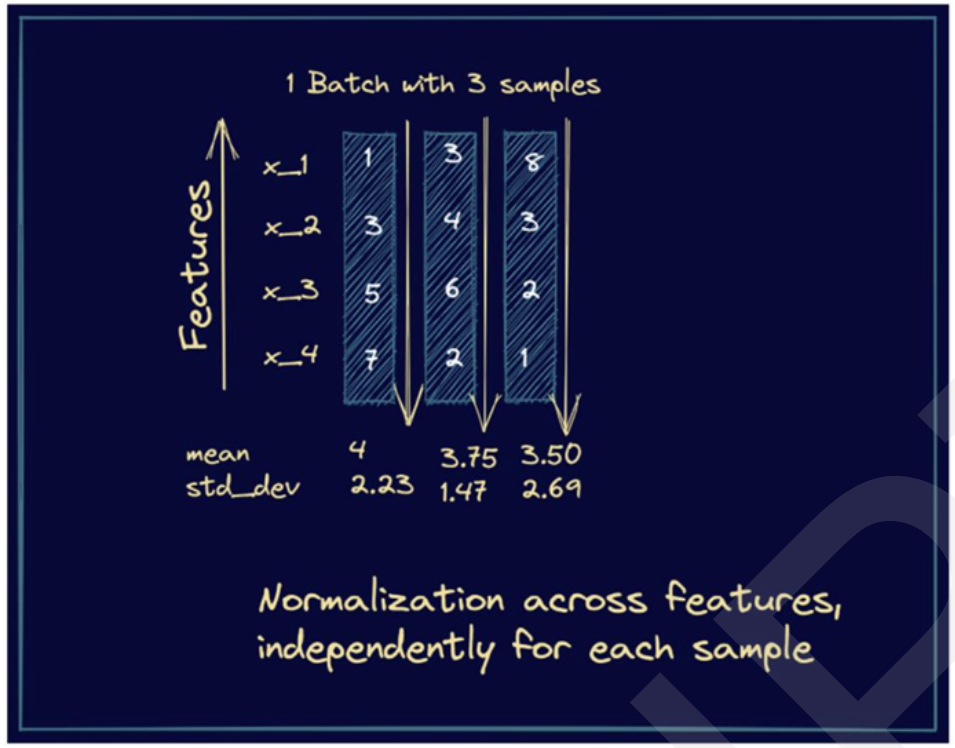
Mean: $\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l$ Standard Deviation: $\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2}$

$$x^{\ell\prime} = \frac{x^\ell - \mu^\ell}{\sigma^\ell + \epsilon}$$

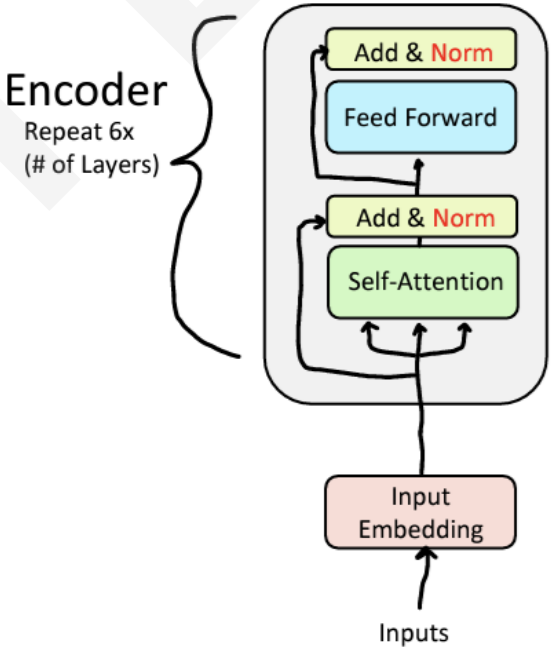


https://web.stanford.edu/class/cs224n/

Training Trick #2: Layer Normalization



An Example of How LayerNorm Works (Image by Bala Priya C, Pinecone)



Mean: $\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l$ Standard Deviation: $\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2}$

$$x^{\ell\prime} = \frac{x^\ell - \mu^\ell}{\sigma^\ell + \epsilon}$$

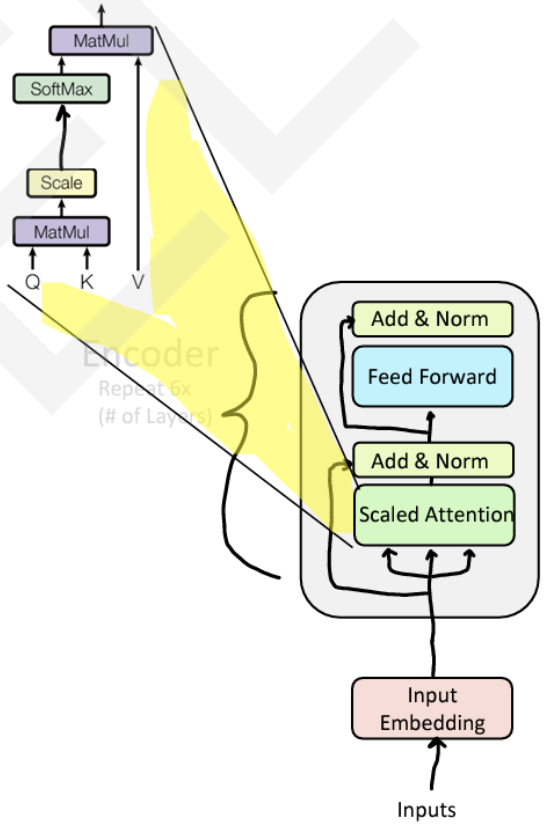
https://web.stanford.edu/class/cs224n/

Training Trick #3: Scaled Dot Product Attention

- After LayerNorm, the mean and variance of vector elements is 0 and 1, respectively. (Yay!)
- However, the dot product still tends to take on extreme values, as its variance scales with dimensionality d_k

Quick Statistics Review:

- Mean of sum = sum of means = $d_k * 0 = 0$
- Variance of sum = sum of variances = $d_k * 1 = d_k$
- To set the variance to 1, simply divide by $\sqrt{d_k}$!

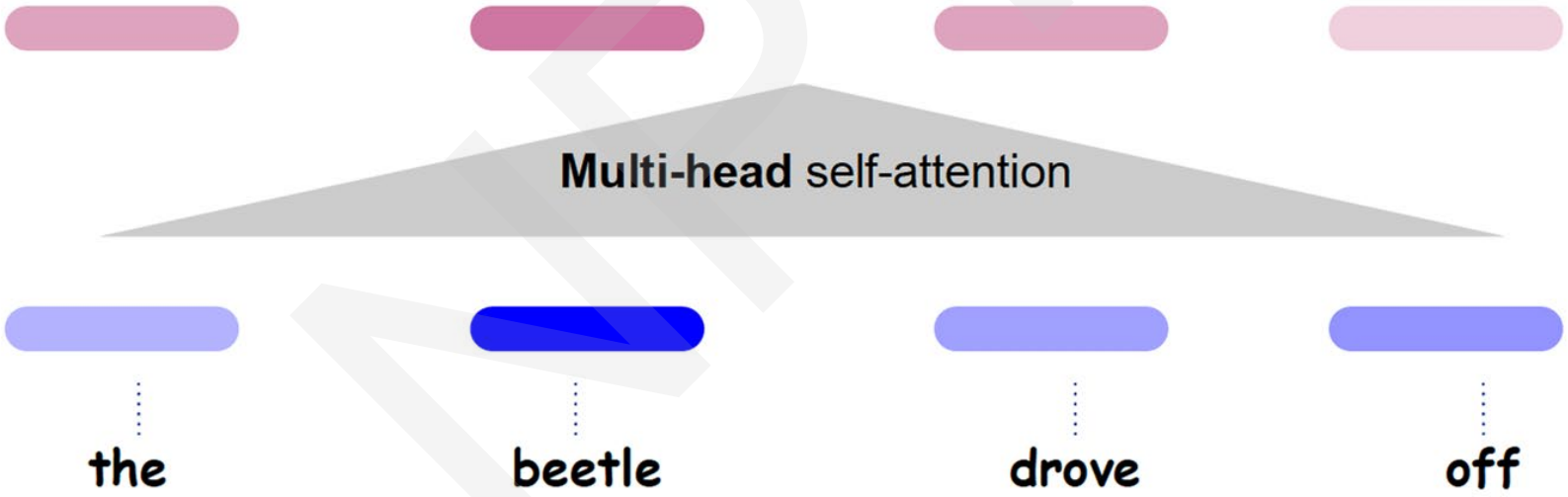


https://web.stanford.edu/class/cs224n/

Training Trick #3: Scaled Dot Product Attention

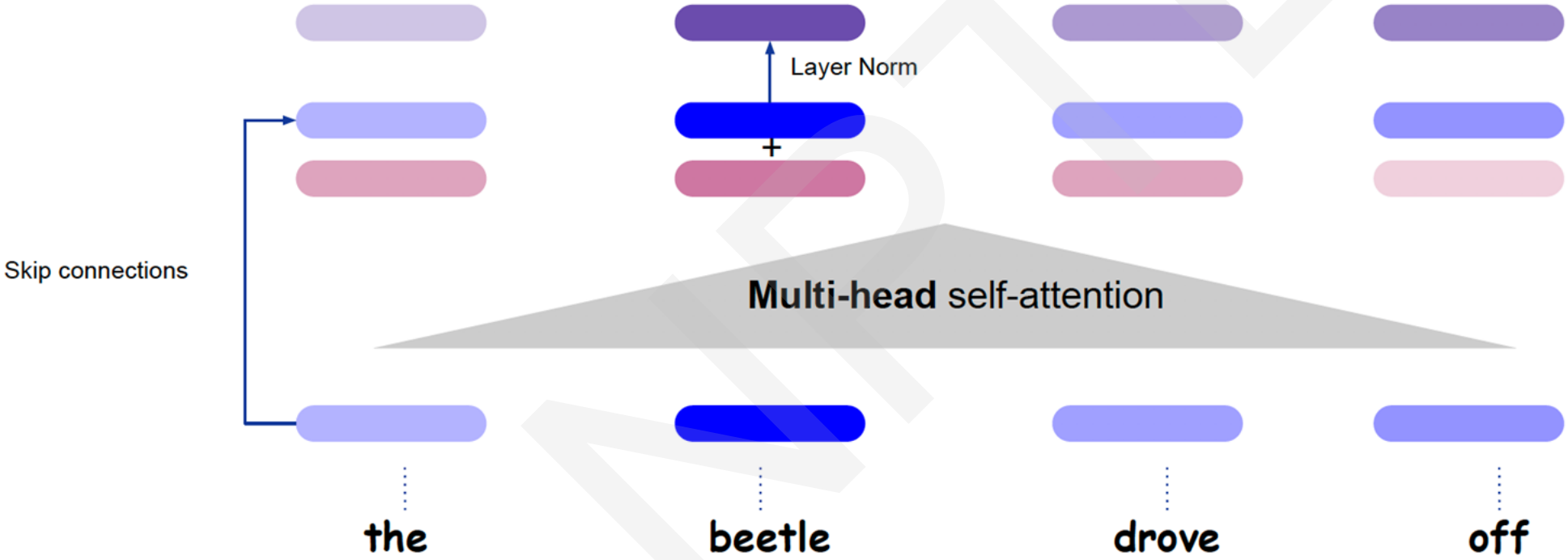
- Assume that the components of q and k are independent random variables with mean 0 and variance 1.
- Then their dot product $q \cdot k = \sum_{i=1}^{d_k} q_i k_i$ has mean 0 and variance d_k.
- Hence the scaling ensures that the resultant dot product has mean 0 and variance 1.

A Complete Transformer Block



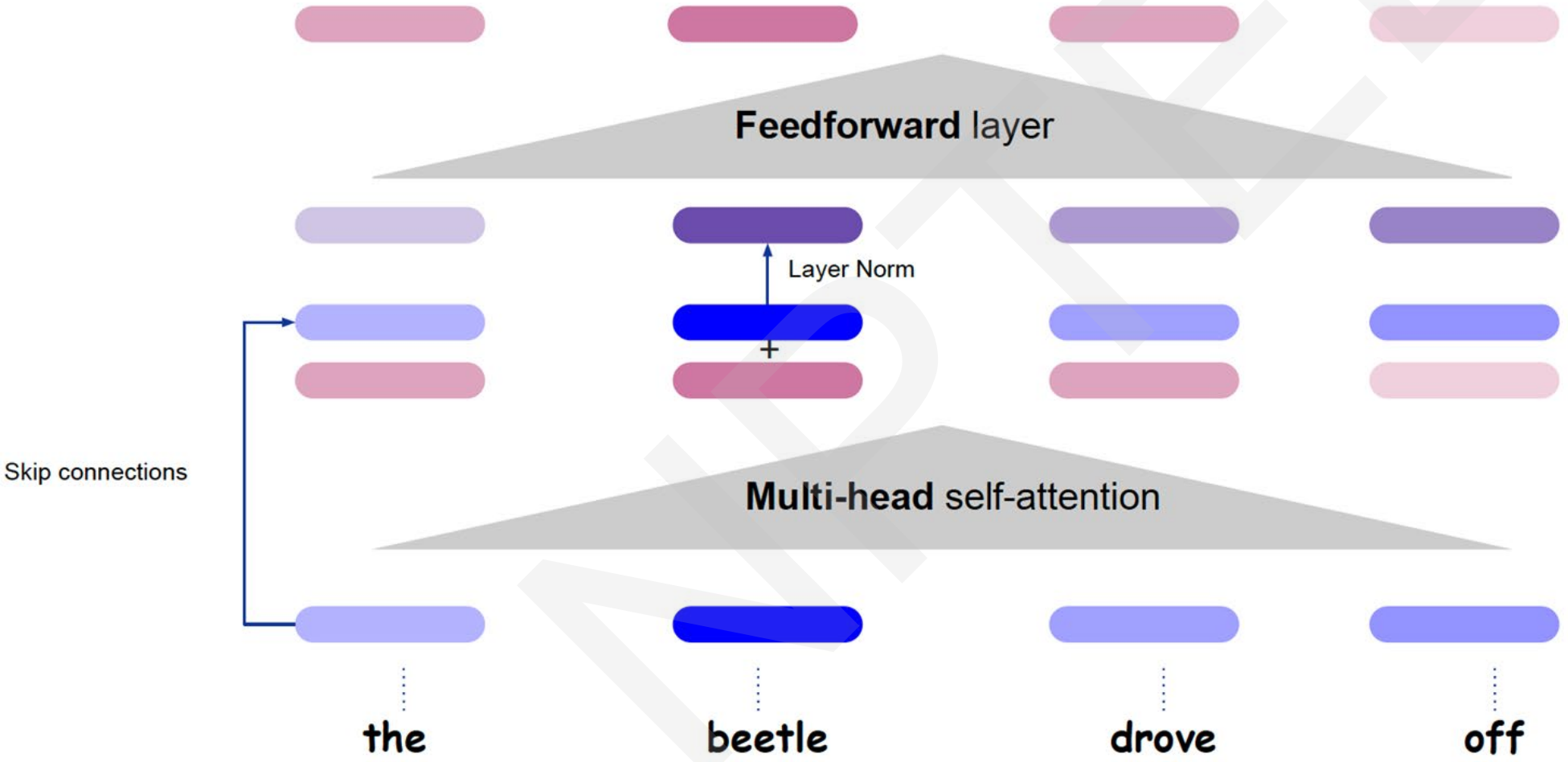
Source: *https://deepmind.com/learning-resources/deep-learning-lecture-series-2020*

A Complete Transformer Block



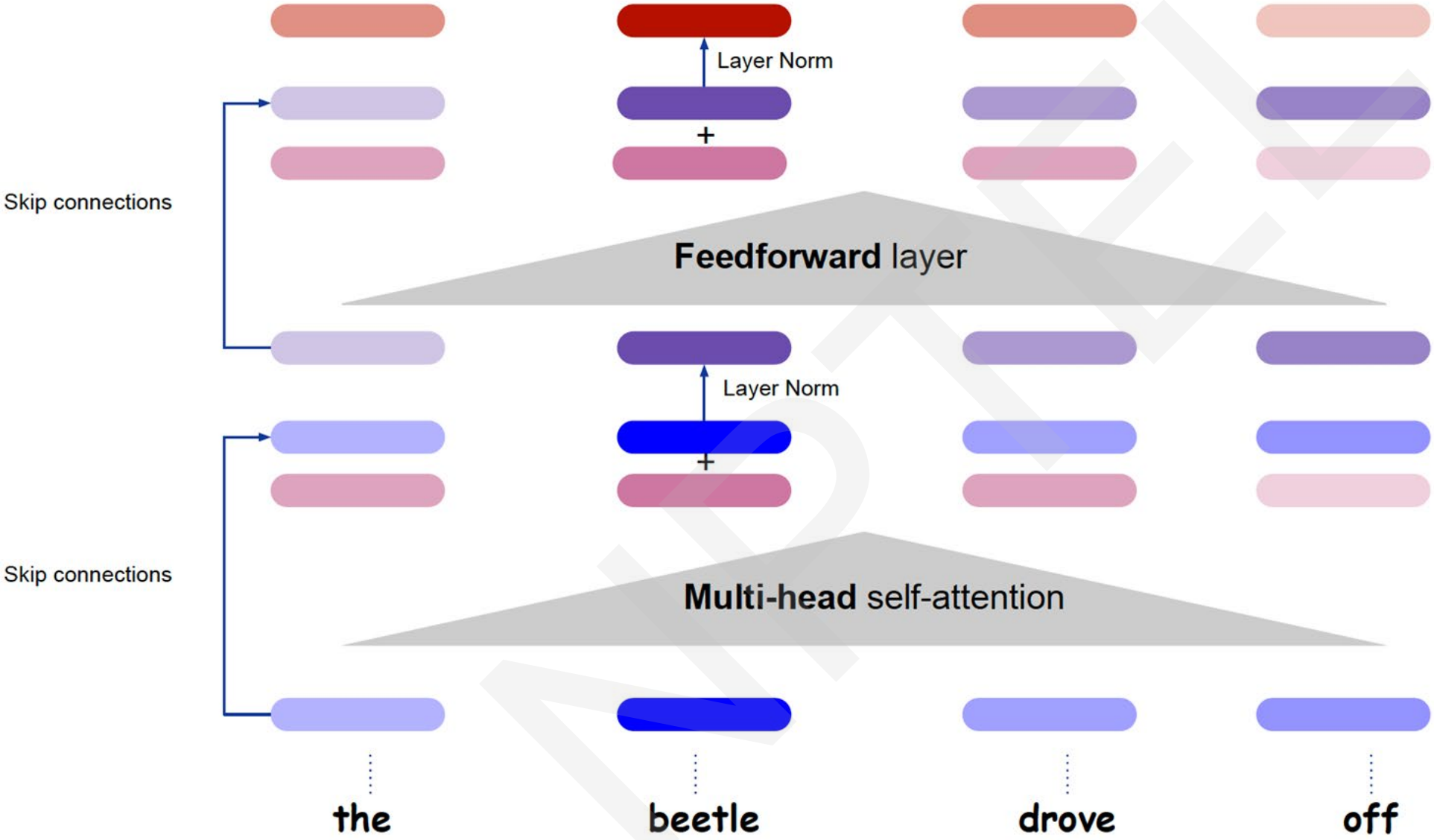
Source: *https://deepmind.com/learning-resources/deep-learning-lecture-series-2020*

A Complete Transformer Block



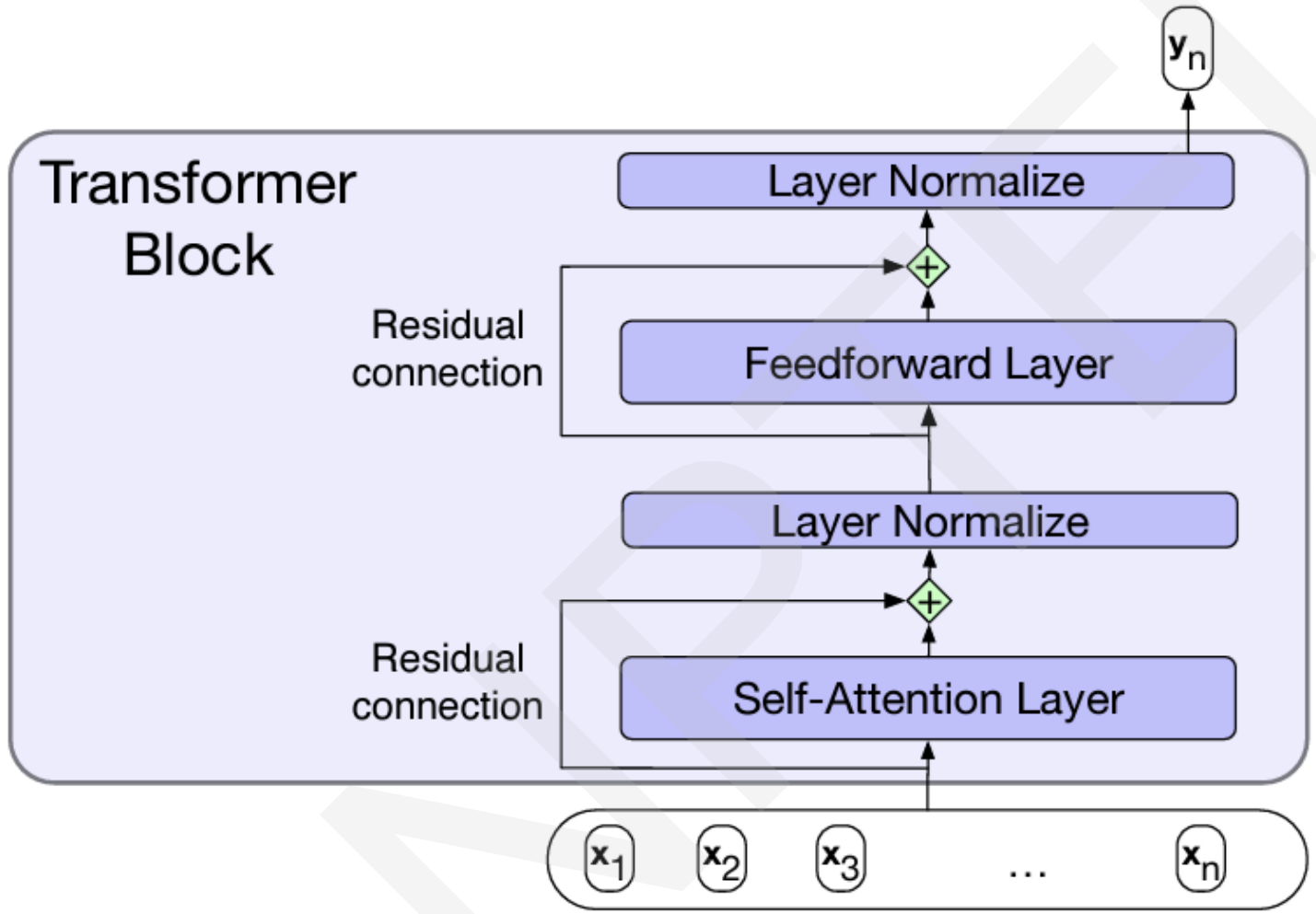
Source: *https://deepmind.com/learning-resources/deep-learning-lecture-series-2020*

A Complete Transformer Block



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

Transformer Block: Another visualization



$$\mathbf{z} = \text{LayerNorm}(\mathbf{x} + \text{SelfAttention}(\mathbf{x}))$$
$$\mathbf{y} = \text{LayerNorm}(\mathbf{z} + \text{FFN}(\mathbf{z}))$$

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

REFERENCES

Daniel Jurafsky and James H. Martin. 2024. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models, 3rd edition. Online manuscript released August 20, 2024. [Chapter 9]


THANK YOU

NPTEL ONLINE CERTIFICATION COURSES

DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING

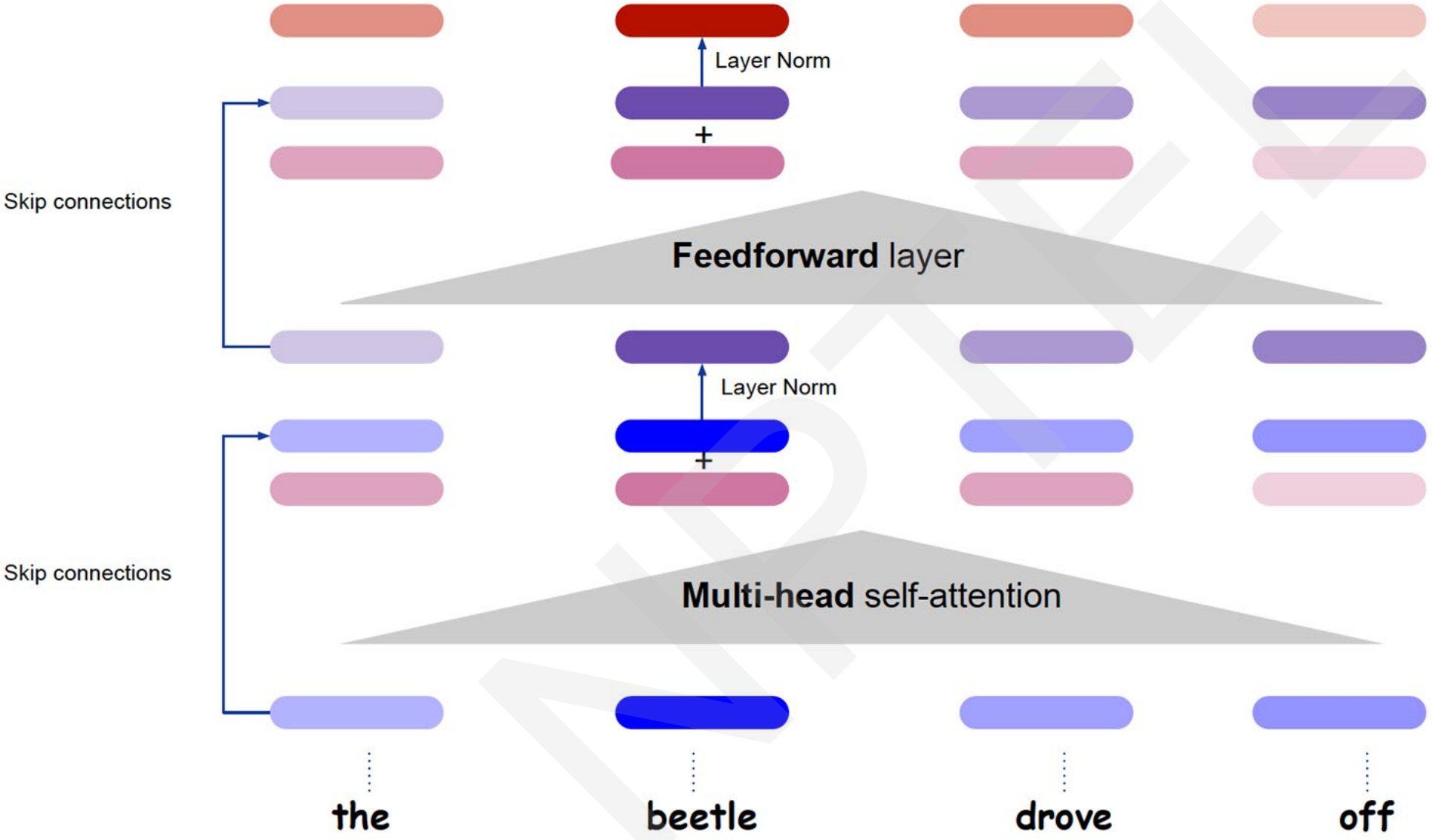
Lecture 23 : Transformers: Part 3

PROF. PAWAN GOYAL
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

CONCEPTS COVERED

- Positional Encodings
- Matrix Calculation of Self-Attention
- Encoding Images

A Complete Transformer Block



There is still a major problem!

Order does not matter!!

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

Position Encoding of words

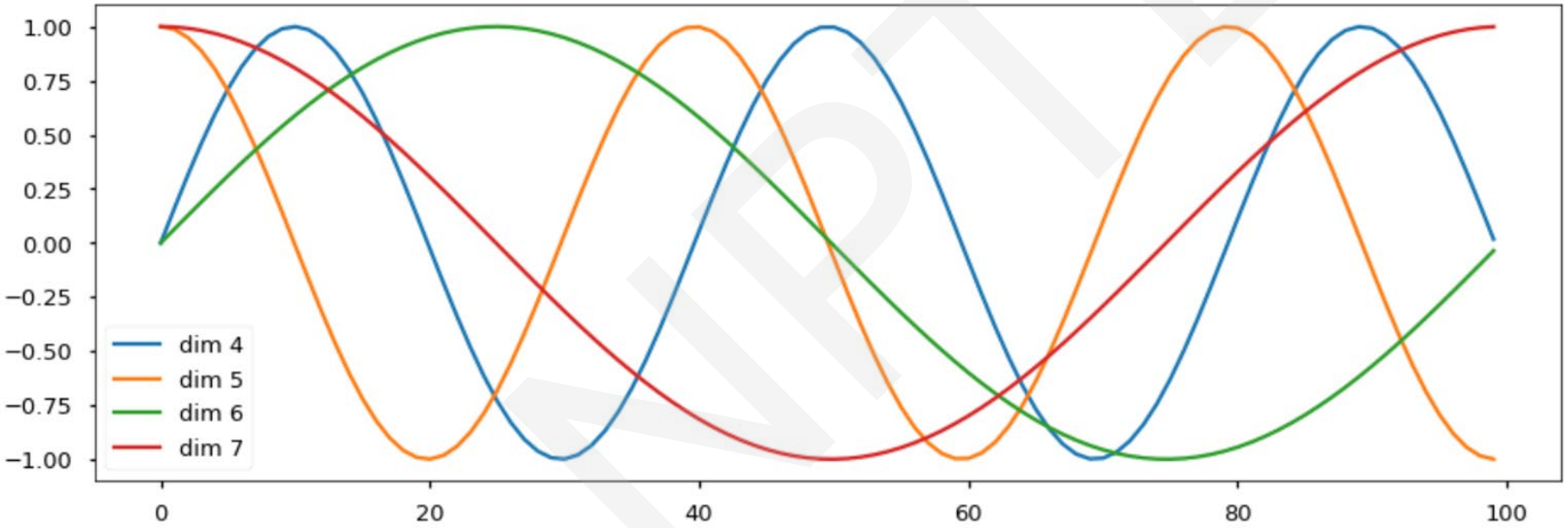
- Add fixed quantity to embedding activations
- The quantity added to each input embedding unit $\in [-1, 1]$ depends on:
  - The dimension of the unit within the embedding
  - The (absolute) position of the words in the input

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

Understanding Positional Encoding

Sinusoidal position representations: concatenate sinusoidal functions of varying periods

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



Understanding Positional Encoding

Suppose you have an input sequence of length L

Defining the positional encoding of the **k^{th} word** within the sequence. The positional encoding is given by **sine** and **cosine** functions of varying frequencies:

even p.e. $$P(k, 2i) = \sin\left(\frac{k}{n^{2i/d}}\right)$$

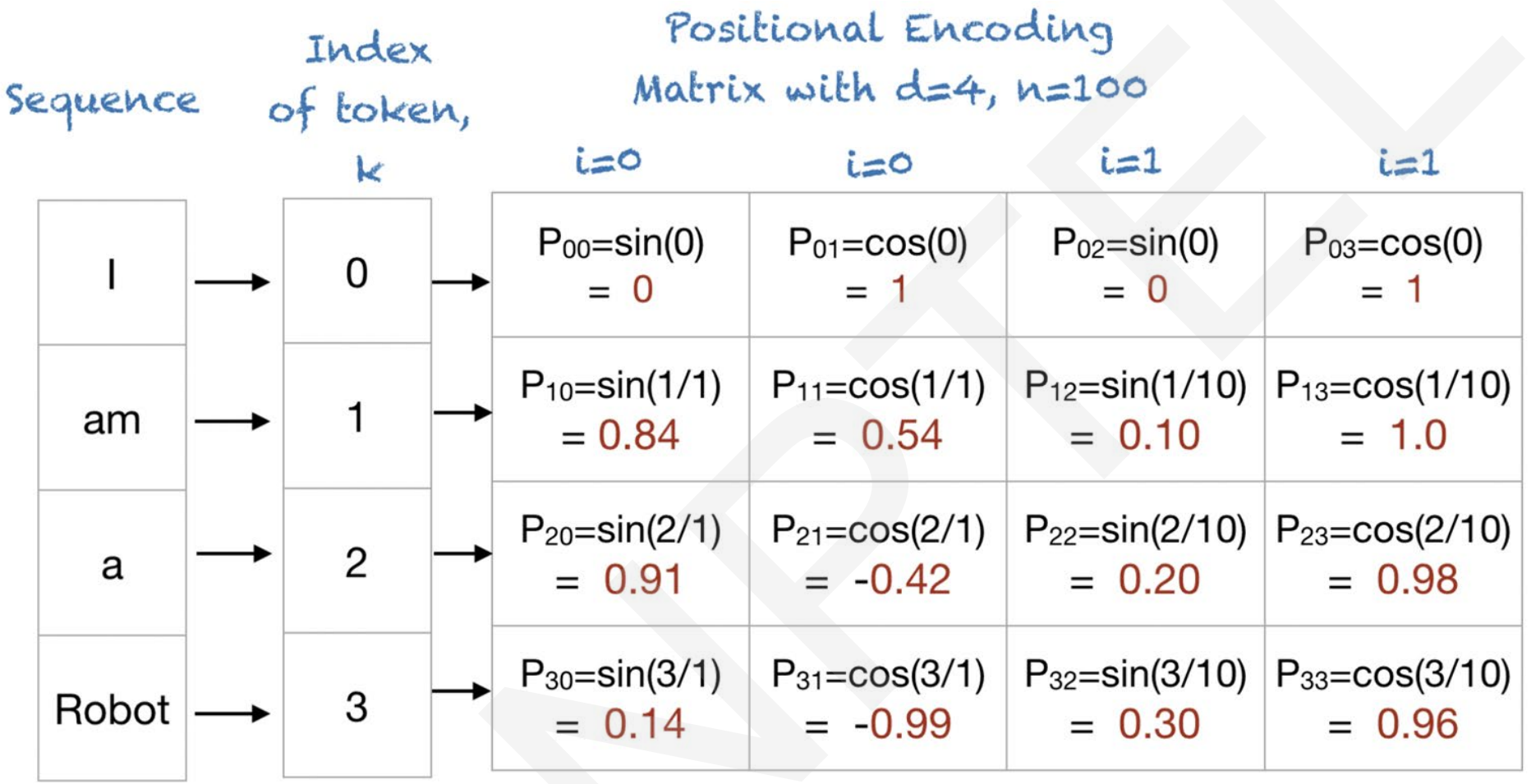
odd p.e. $$P(k, 2i+1) = \cos\left(\frac{k}{n^{2i/d}}\right)$$

Here:

k: Position of a word in the input sequence
i: Used for mapping to indices in the positional encoding of a particular word $0 \leq i < d/2$
d: Dimension of the output embedding space
P(k,j): Position function for mapping a position k in the input sequence to index (k,j) of the positional matrix
n: User-defined scalar, set to 10,000 by the authors of Attention Is All You Need.

https://rycolab.io/classes/llm-s24/

Understanding Positional Encoding



$$P(k, 2i) = \sin\left(\frac{k}{n^{2i/d}}\right)$$

$$P(k, 2i+1) = \cos\left(\frac{k}{n^{2i/d}}\right)$$

https://rycolab.io/classes/llm-s24/

Other variants: Learned Positional embedding

Goal: learn a position embedding matrix E_{pos} of shape $[1 \times N]$.

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.

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

Positional Embeddings

BERT, GPT-2/3

1. Learned Positional Embeddings (Gehring et al., 2017 ,Vaswani et al., 2017)

2. Sinusoidal Embeddings (Vaswani et al., 2017)

Gopher (variant)

3. Relative Positional Embeddings (RPE) (Shaw et al., 2018)

T5

4. Relative Positional Bias (Raffel et al., 2020)

5. Rotary Position Embeddings (RoPE) (Su et al., 2021)

Llama 1/2, Mistral, Falcon, PaLM, GPT-J, OLMo, Gemma

BLOOM, MPT

1. Attention with Linear Biases (ALiBi) (Press et al., 2022)

Slide by Maitrey Mehta

More Details: Self-attention

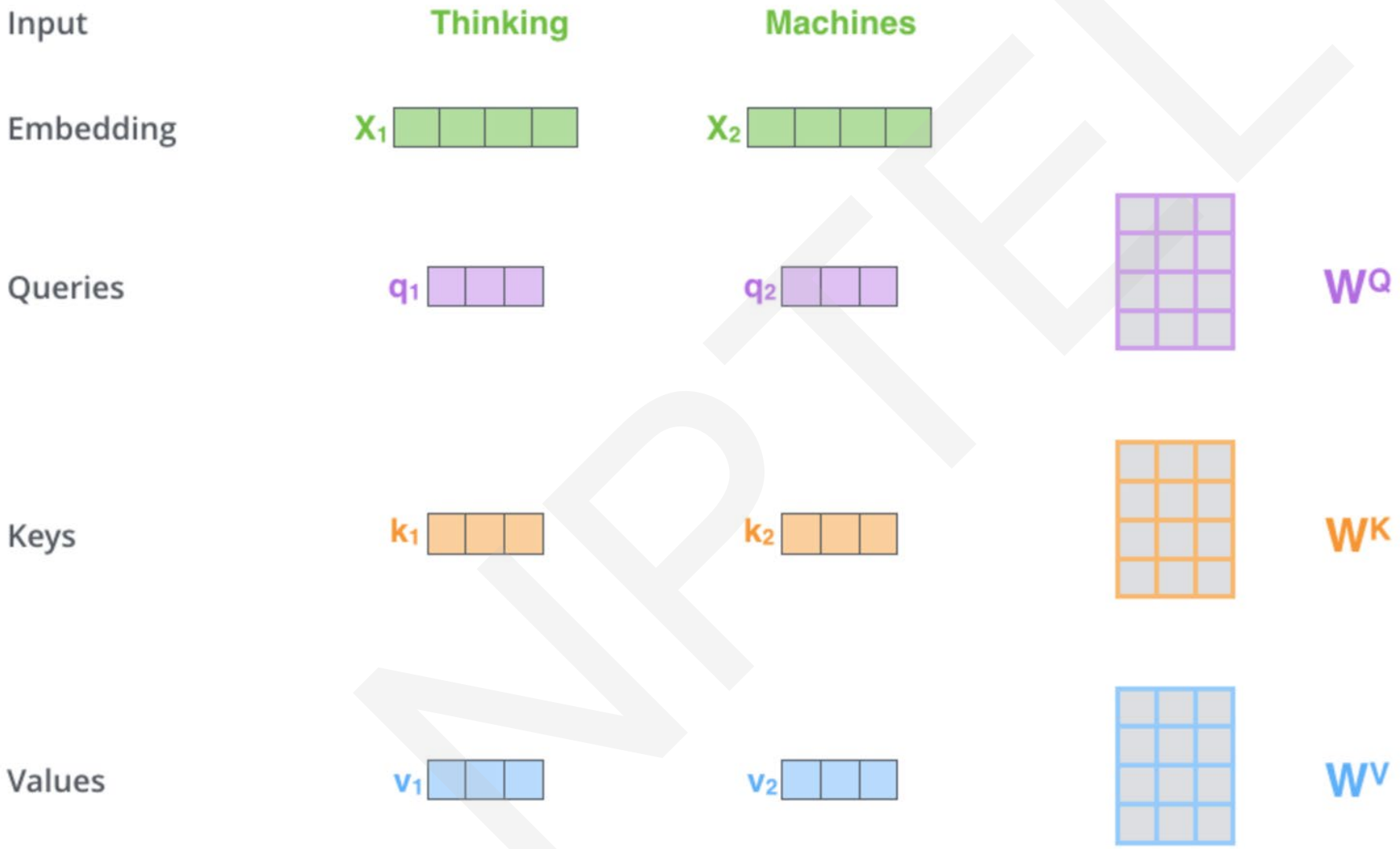


Figure: Jay Alammar

Scaled Dot Product for Attention

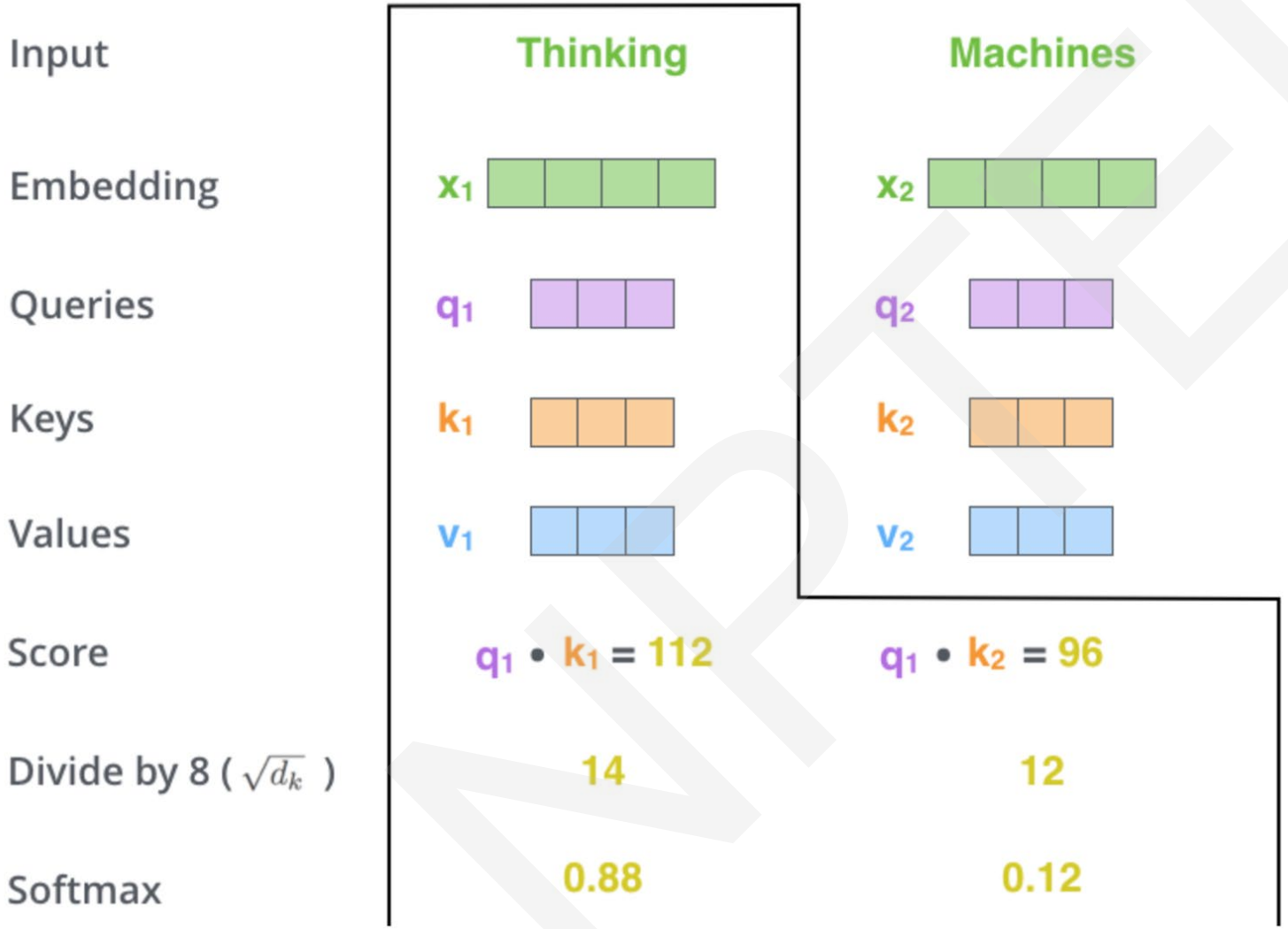


Figure: Jay Alammar

Matrix Calculation of Self Attention

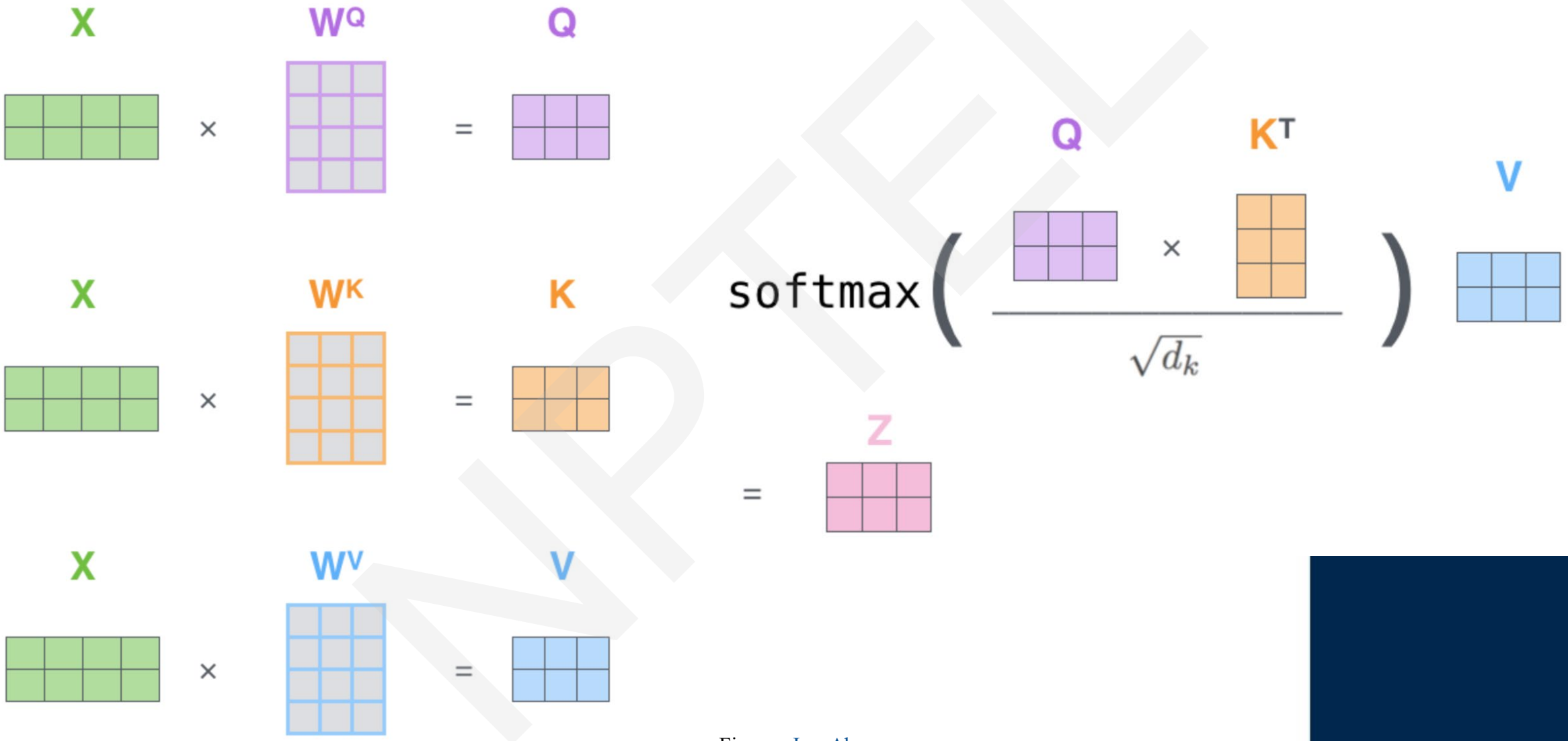


Figure: Jay Alammar

Multi-headed Attention

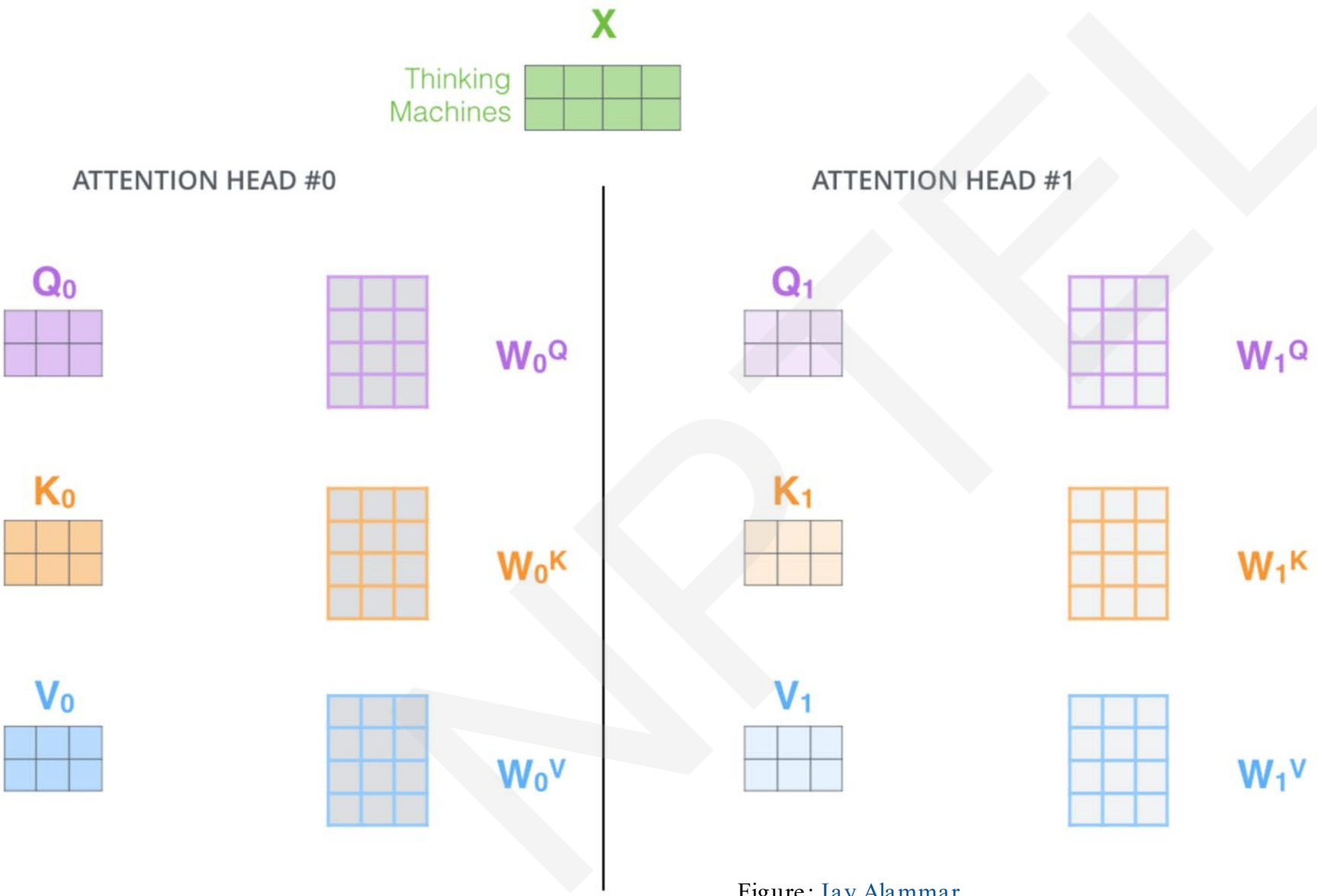


Figure: Jay Alammar

Multi-headed Attention

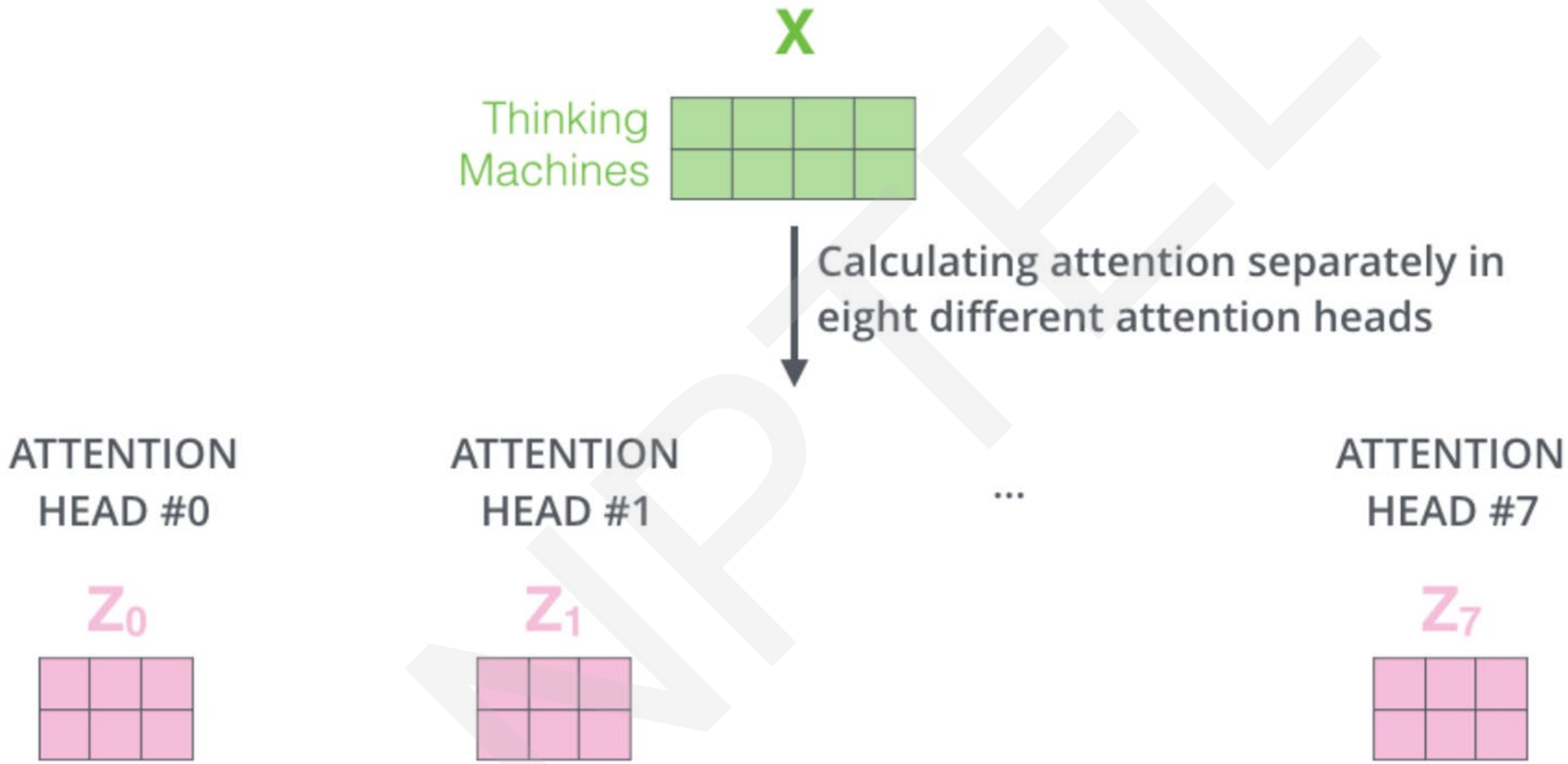
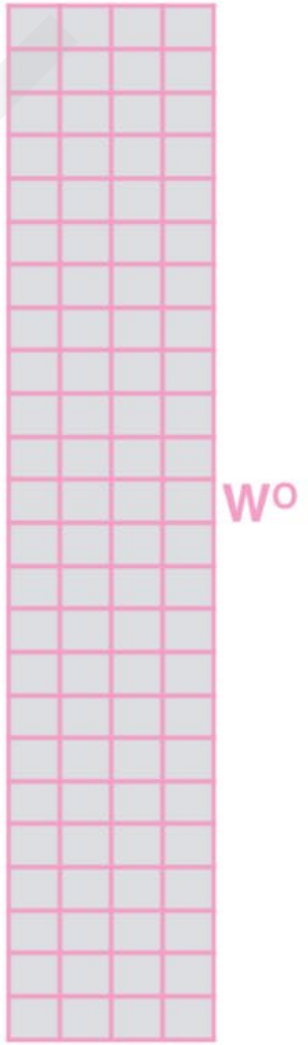


Figure: Jay Alammar

Combining from Multiple Heads

1) Concatenate all the attention heads

2) Multiply with a weight matrix W^O that was trained jointly with the model



3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

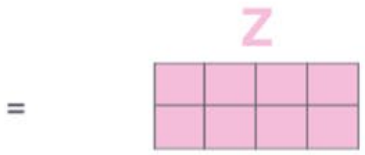


Figure: Jay Alammar

Multi-headed Self Attention Block

1) This is our input sentence*

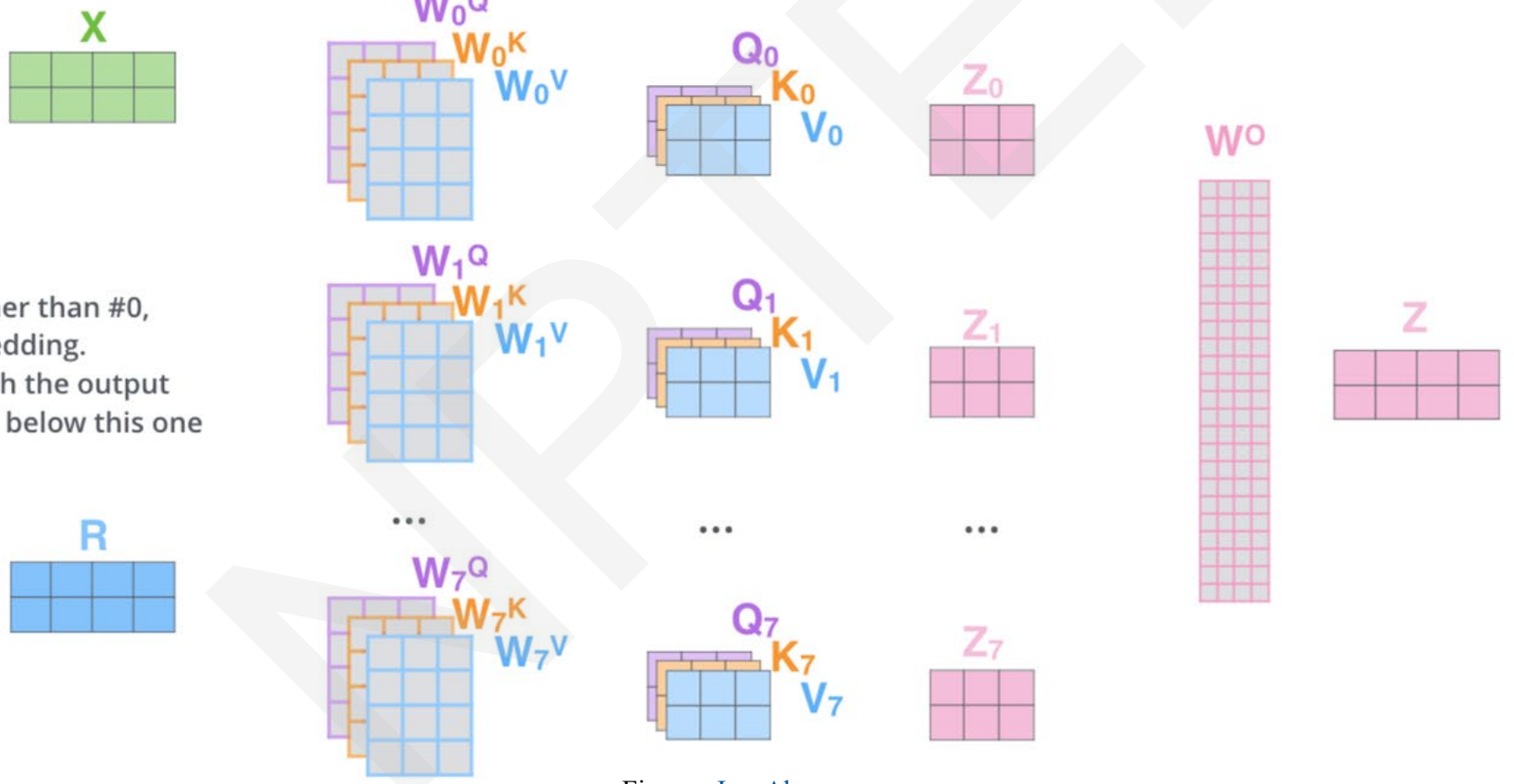
2) We embed each word*

3) Split into 8 heads. We multiply X or R with weight matrices

4) Calculate attention using the resulting Q/K/V matrices

5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

Thinking Machines



* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

Figure: Jay Alammar

Encoder Block

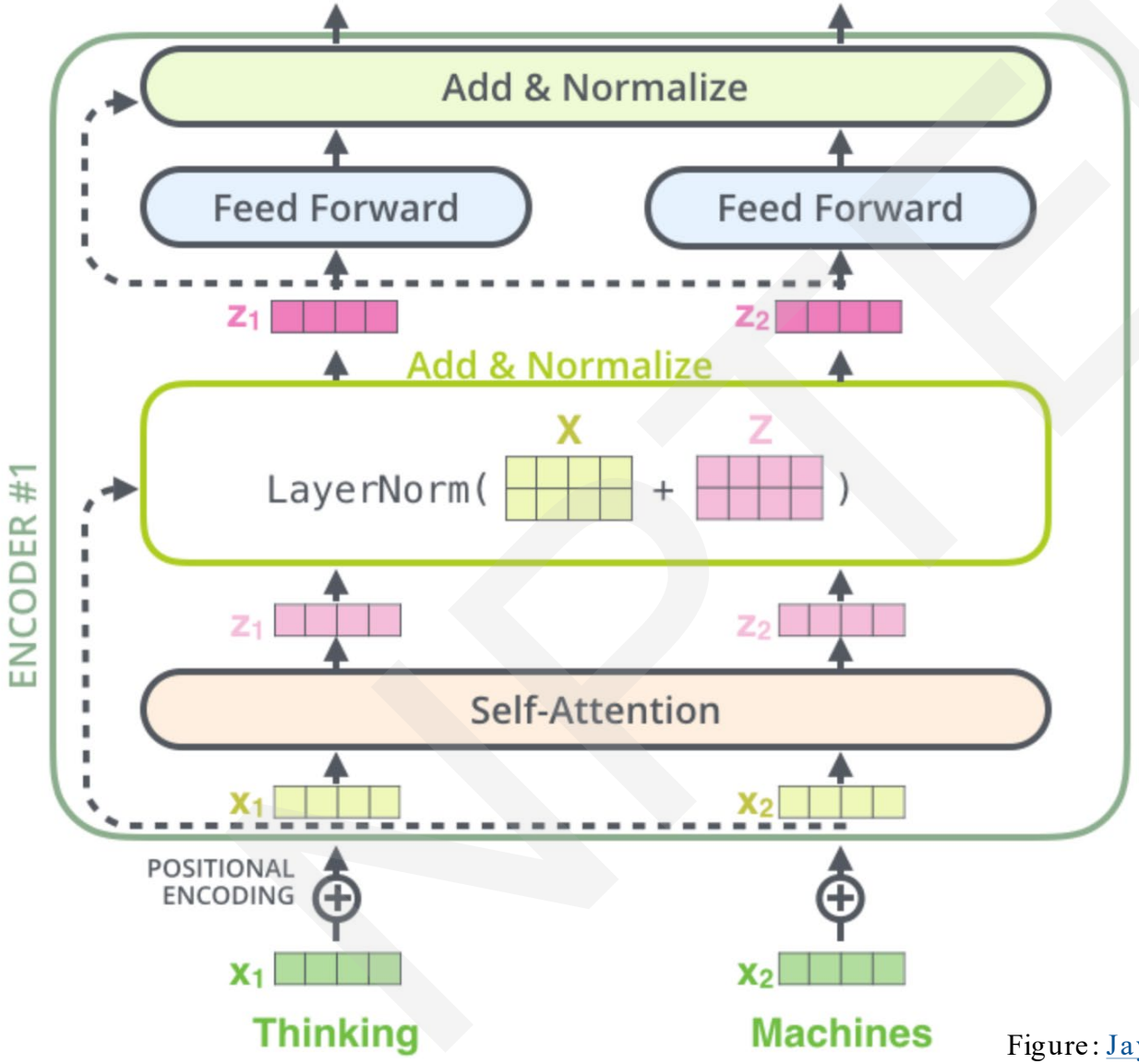


Figure: Jay Alammar

Number of parameters in the encoder?

Encoding Images? Vision Transformer

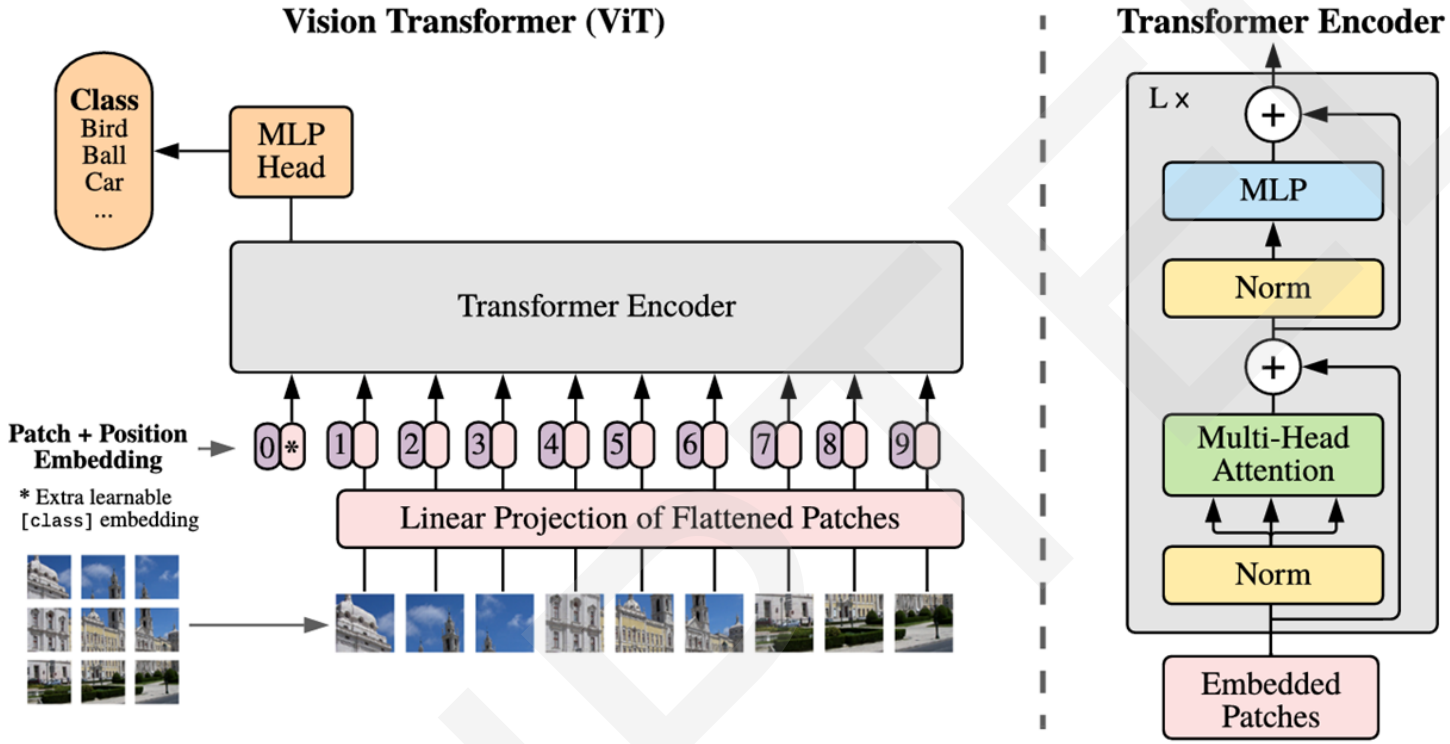


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale: ICLR 2021

How to encode an image as a sequence?

- Original image shape: $H \times W \times C$
- We create N patches of dimensions $P \times P$: $N = HW/(P * P)$ (N is the effective sequence length)
- Each patch is mapped to D dimensions through a trainable linear projection $E \in R^{(P^2 \cdot C) \times D}$
- Standard learnable 1-D positional embeddings are used $E_{pos} \in R^{(N+1) \times D}$
- A learnable class token embedding is prepended, and its representation at the last layer is used as the image representation

REFERENCES

Daniel Jurafsky and James H. Martin. 2024. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models, 3rd edition. Online manuscript released August 20, 2024. [Chapter 9]


THANK YOU

NPTEL ONLINE CERTIFICATION COURSES

DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING

Lecture 24 : Transformers: Part 4

PROF. PAWAN GOYAL
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

CONCEPTS COVERED

- Transformer Decoder
- Transformer LM

Transformer with encoders and decoders

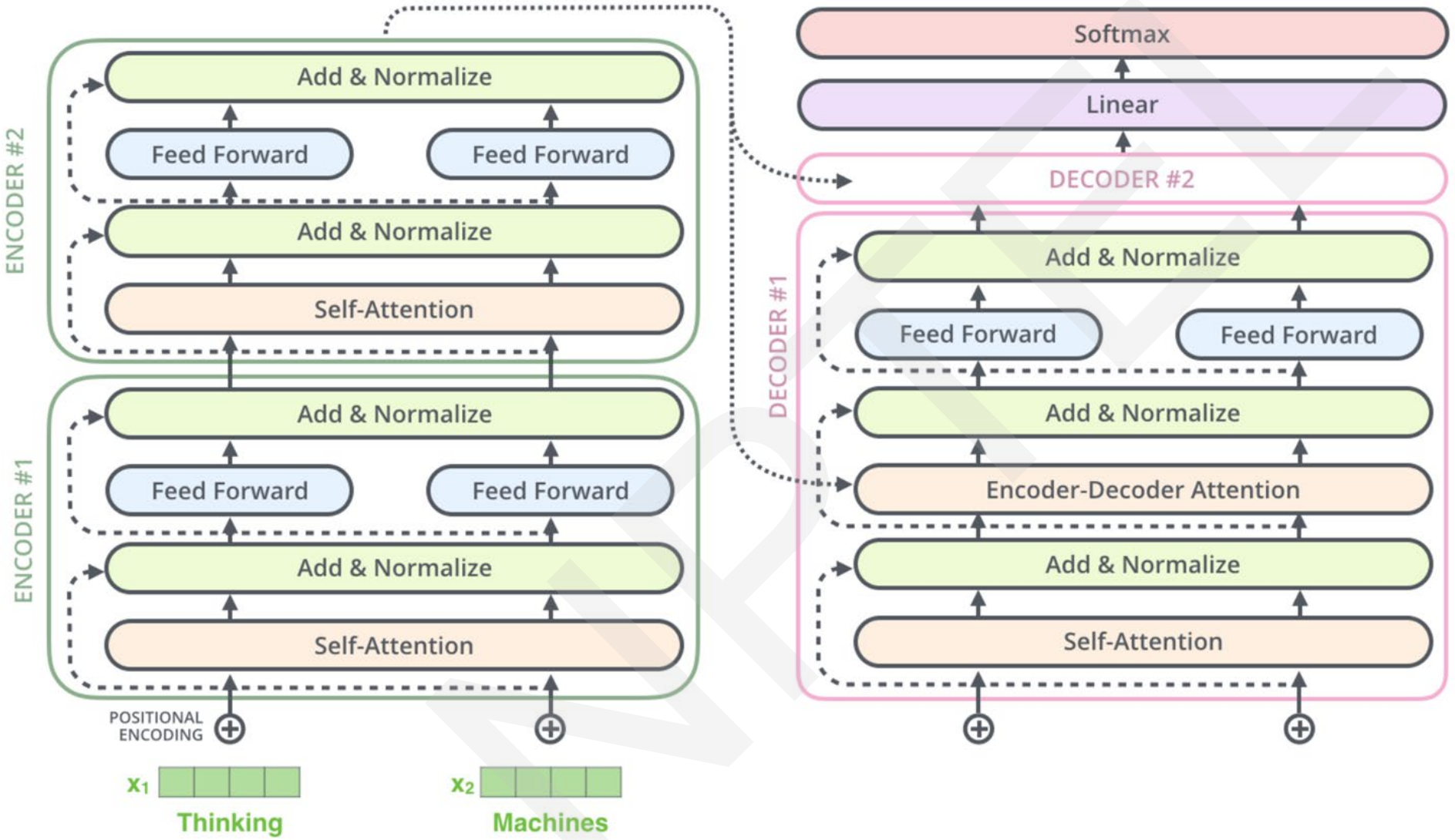


Figure: Jay Alammar

How is the decoder different?

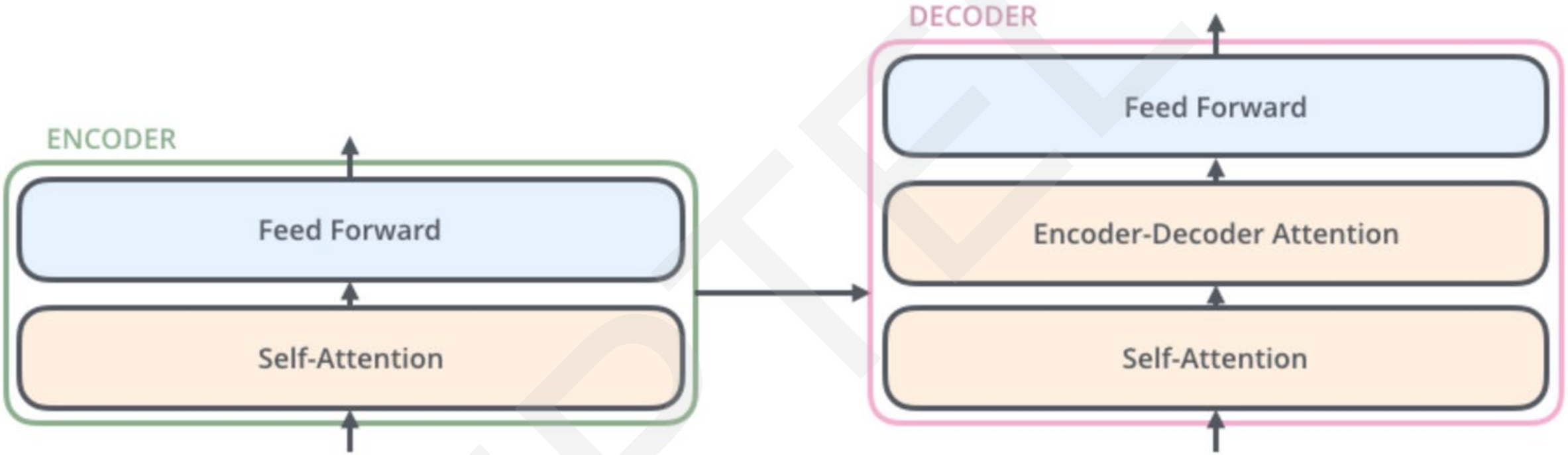
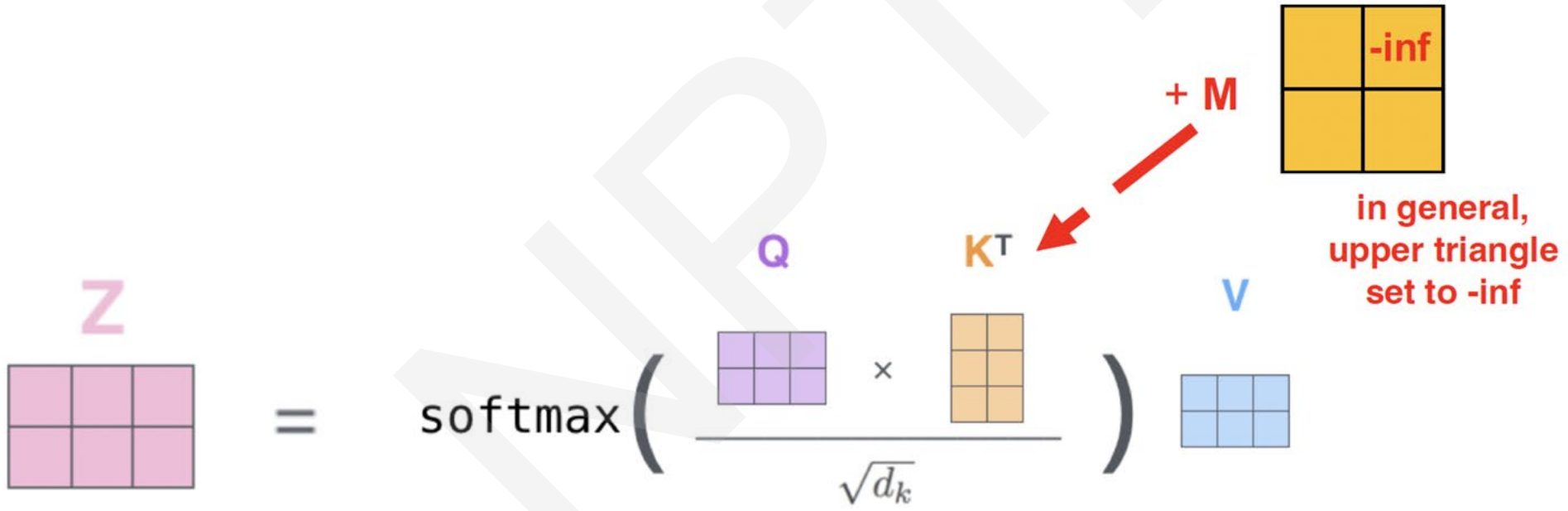


Figure: Jay Alammar

Masked Self-attention

In the **decoder**, the self-attention layer is only allowed to attend to **earlier positions** in the output sequence. Otherwise, we'd be cheating!

This is done by **masking future positions** **(setting the dot product score for future positions to -inf)** before the Softmax step in the self-attention calculation.



https://rycolab.io/classes/llm-s24/

Masked Self-attention for decoder (to avoid seeing the future tokens)

N	q1•k1	−∞	−∞	−∞	−∞
	q2•k1	q2•k2	−∞	−∞	−∞
	q3•k1	q3•k2	q3•k3	−∞	−∞
	q4•k1	q4•k2	q4•k3	q4•k4	−∞
	q5•k1	q5•k2	q5•k3	q5•k4	q5•k5
			N		

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

Encoder-Decoder Attention

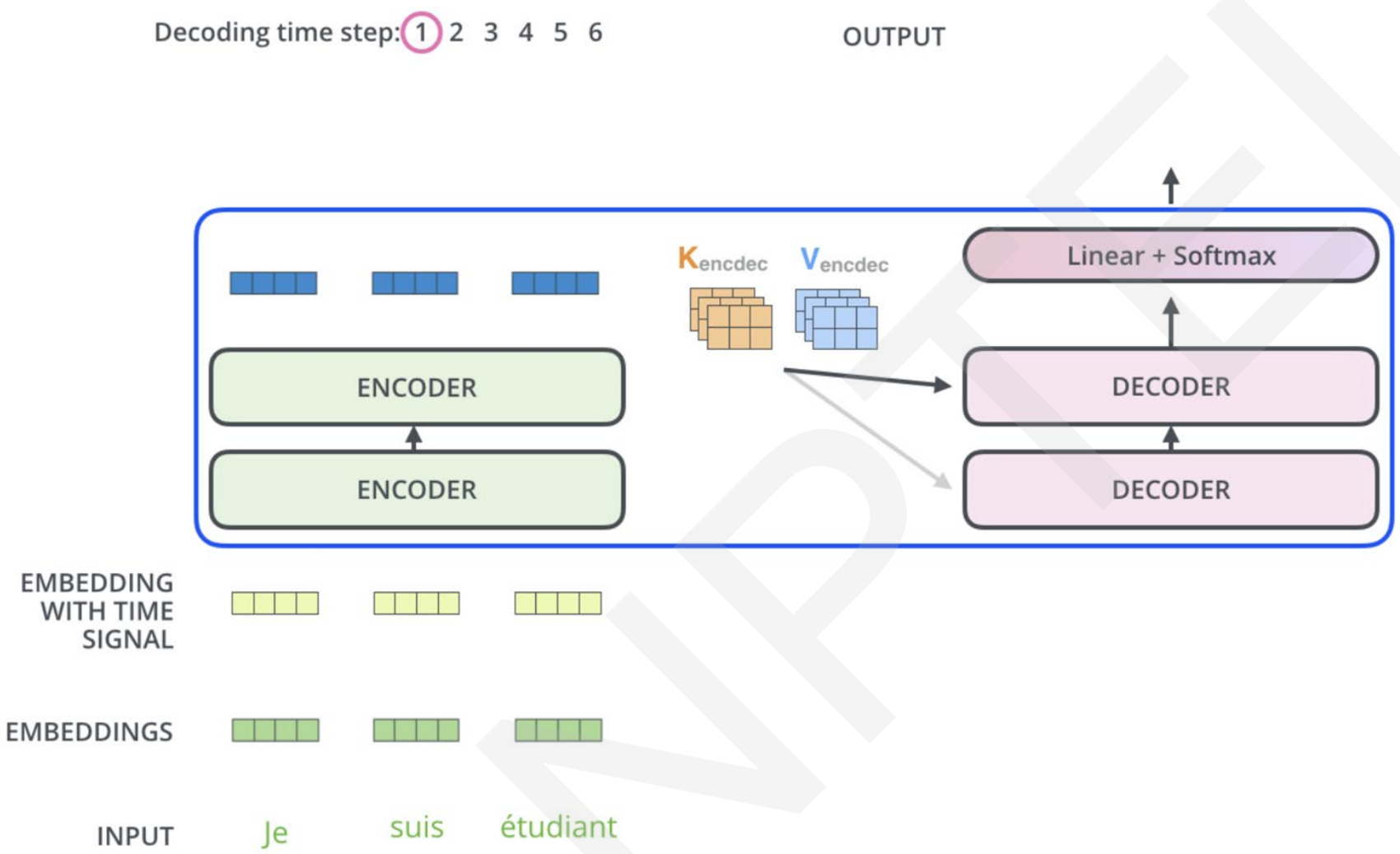
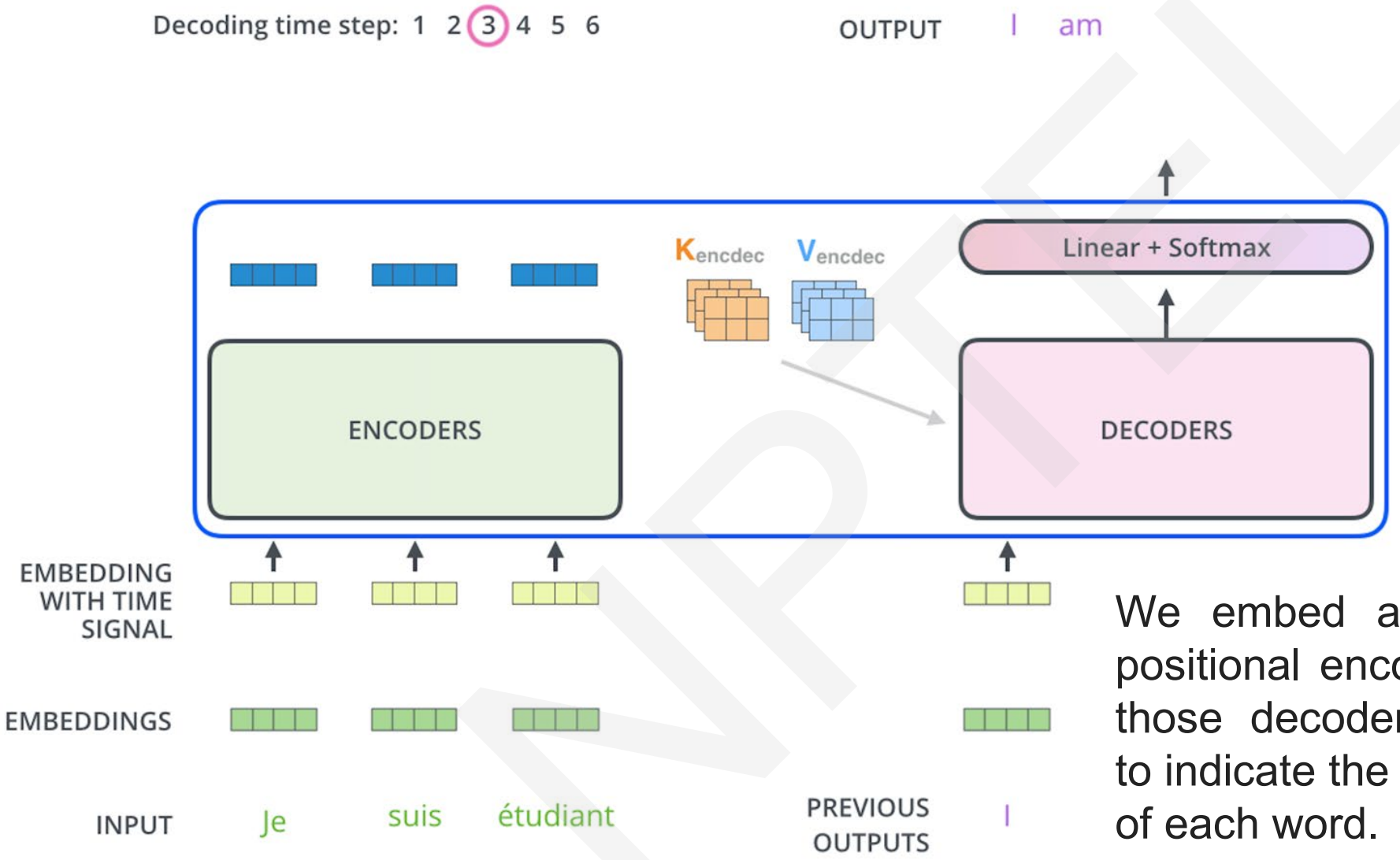


Figure: Jay Alammar

The output of the top encoder is transformed into a set of attention vectors K and V.

These are to be used by each decoder in its "***encoder-decoder attention***" layer which helps the decoder focus on appropriate places in the input sequence:

Decoding

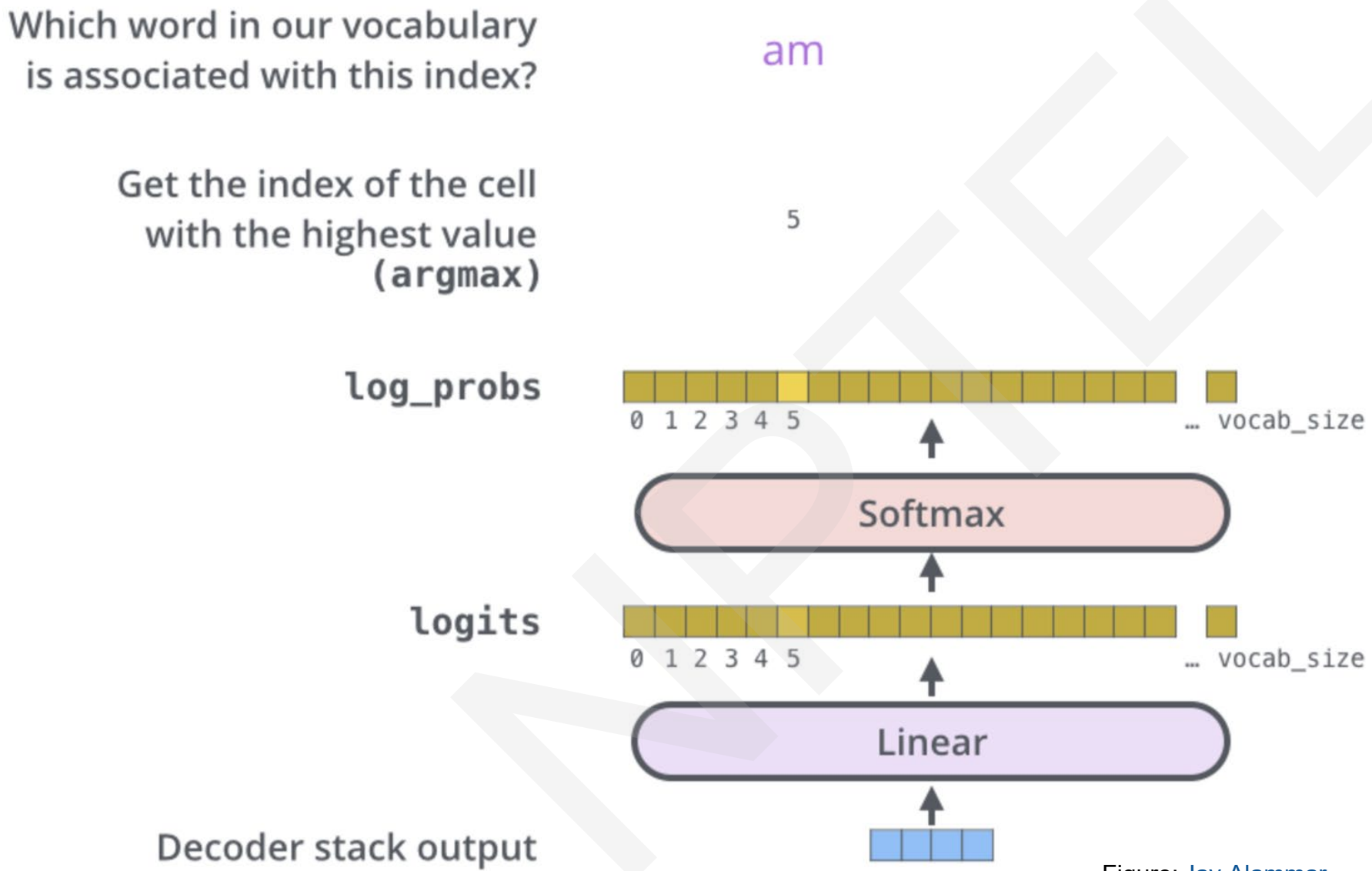


The output of each step is fed to the bottom decoder in the next time step, and the decoders bubble up their decoding results just like the encoders did.

We embed and add positional encoding to those decoder inputs to indicate the position of each word.

Figure: Jay Alammar

Converting decoder stack output to words



That's the job of the final ***Linear layer*** which is followed by a ***Softmax Layer***.

Figure: Jay Alammar

Try this problem

*Suppose, you are training your transformer **decoder** with the Ground truth: '**how are you**', and you are using multi-headed attention with h = 2. The embeddings for these words are the concatenation of in and out embeddings shown in Table below. For both your attention heads, the first and third dimensions of the input define your query vector, and the second and fourth dimensions define your key vector. For the first attention head, the value is the in-vector, and for the second attention head, the value is the out-vector. What will be the output of the multi-headed self-attention block for the word '**are**' (before applying the projection matrix W_0)? You are using the scaled dot vector for self-attention. Ignore the start-of-sequence token.*

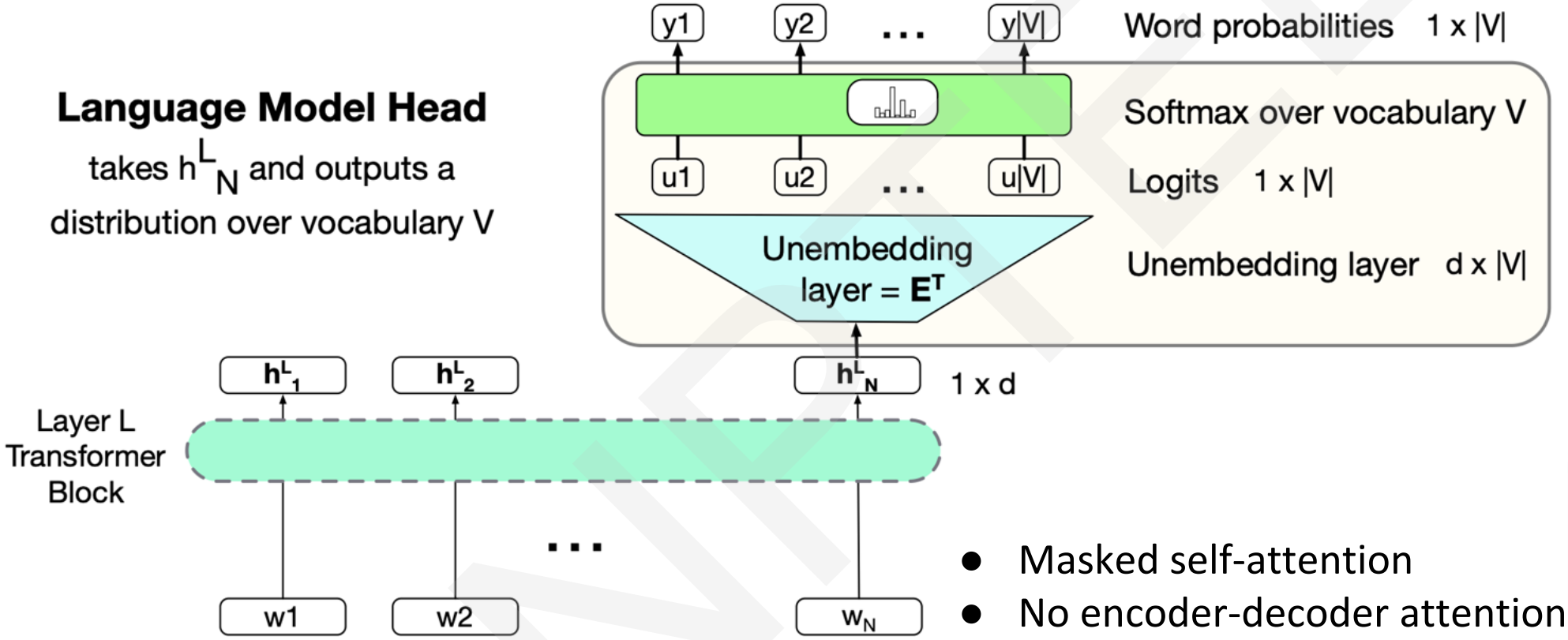
Word	In-vector	Out-vector
hi	(1, -1)	(-2, 2)
you	(-1, 2)	(1, -2)
they	(-2, -2)	(2, 2)
are	(1, 1)	(-1, -1)
am	(2, 1)	(-2, -1)
how	(-1, -1)	(1, 1)
why	(1, 2)	(-1, -2)
there	(2, -1)	(-2, 1)
who	(2, 2)	(-2, -2)
what	(1, 1)	(-1, -1)

Contd..

Word	In-vector	Out-vector
how	(-1, -1)	(1, 1)
are	(1, 1)	(-1, -1)
you	(-1, 2)	(1, -2)

how are you

Transformers as Language models

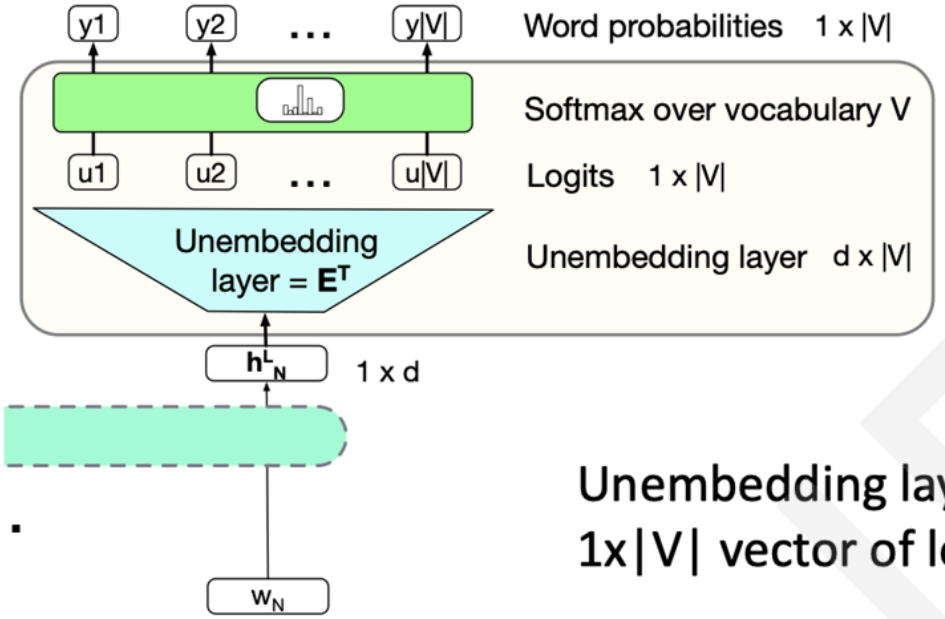


- Masked self-attention
- No encoder-decoder attention

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

Transformers as Language models

Unembedding layer: linear layer projects from h^L_N (shape $[1 \times d]$) to logit vector



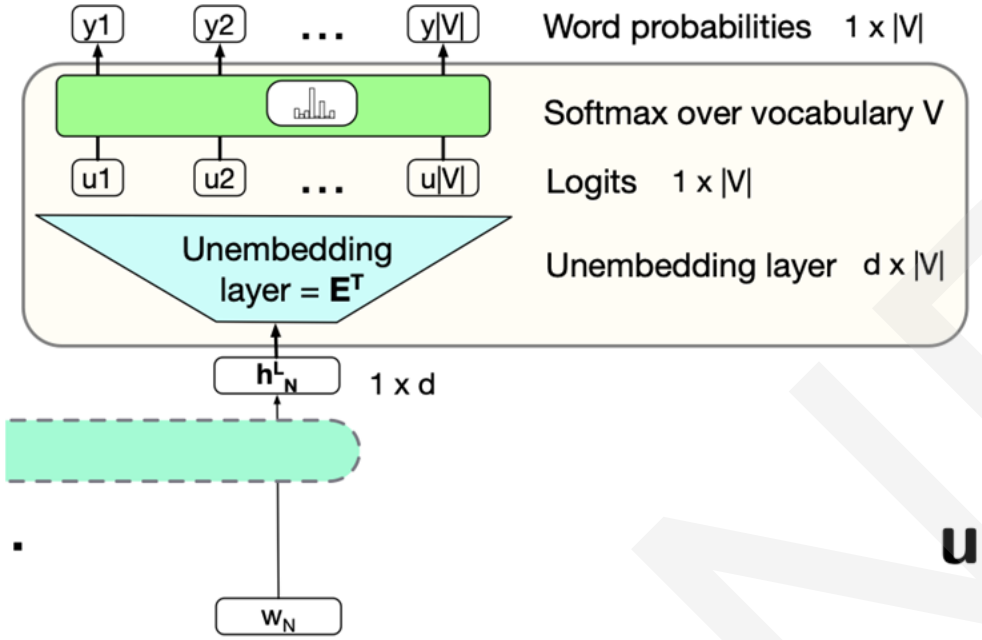
Why "unembedding"? **Tied** to $\mathbf{E}^T$

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

Unembedding layer maps from an embedding to a 1x|V| vector of logits

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

Language modeling head



Logits, the score vector u

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

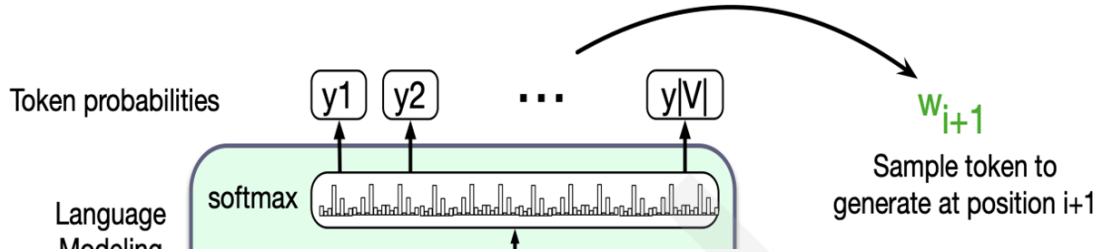
Softmax turns the logits into probabilities over vocabulary. Shape $1 \times |V|$.

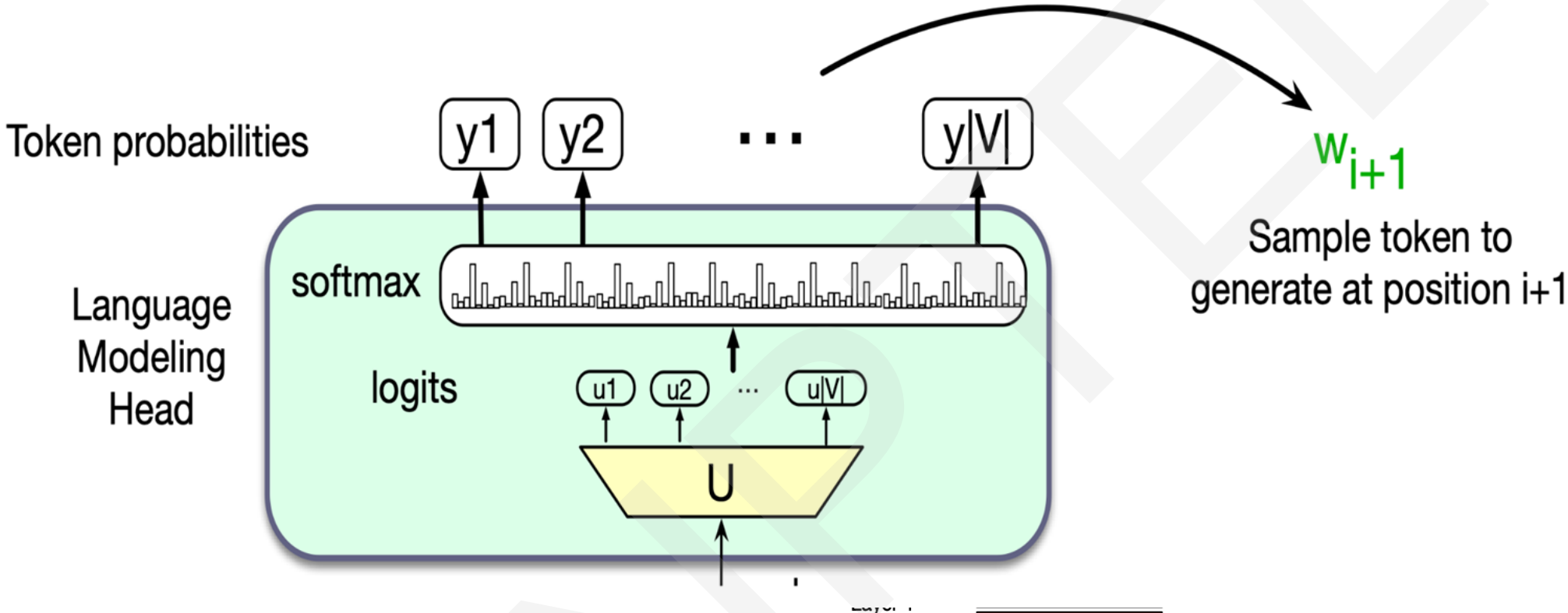
$$\mathbf{u} = \mathbf{h}_N^L \mathbf{E}^T$$

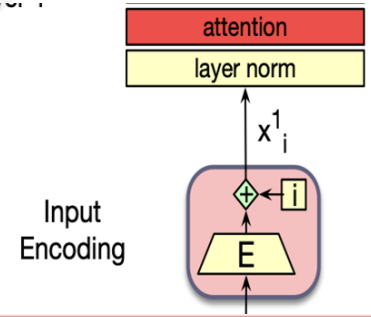
$$\mathbf{y} = \text{softmax}(\mathbf{u})$$

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

Transformers LM







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

Number of parameters in encoder-decoder?

Number of parameters in the decoder only Transformer?

REFERENCES

Daniel Jurafsky and James H. Martin. 2024. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models, 3rd edition. Online manuscript released August 20, 2024. [Chapter 9]


THANK YOU

NPTEL ONLINE CERTIFICATION COURSES

DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING

Lecture 25 : Efficient Transformers

PROF. PAWAN GOYAL
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

CONCEPTS COVERED

- Efficiency in Decoding: KV Cache
- Transformer for Long Sequences: Sparse Attention, Longformer, BIGBIRD
- Grouped Query Attention
- Sparse Mixture-of-Experts

KV Cache

In training, we can compute attention very efficiently in parallel:

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\intercal}{\sqrt{d_k}}\right)\mathbf{V}$$

But not at inference! We generate the next tokens **one at a time!**

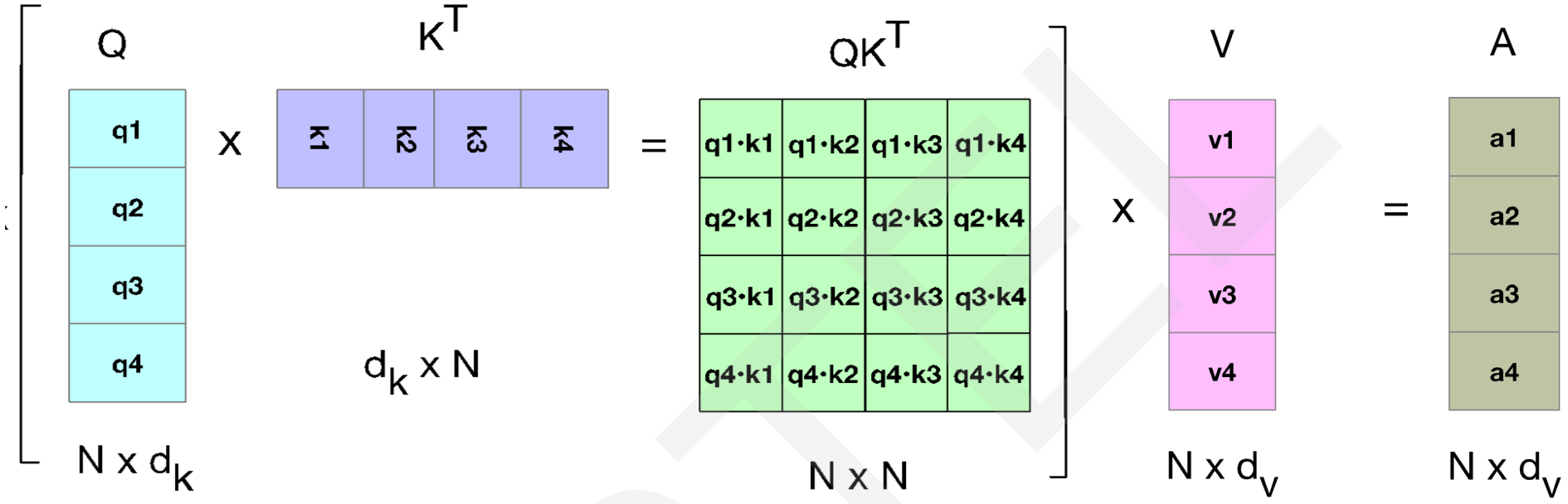
For a new token x, need to multiply by W^Q, W^K, and W^V to get query, key, values

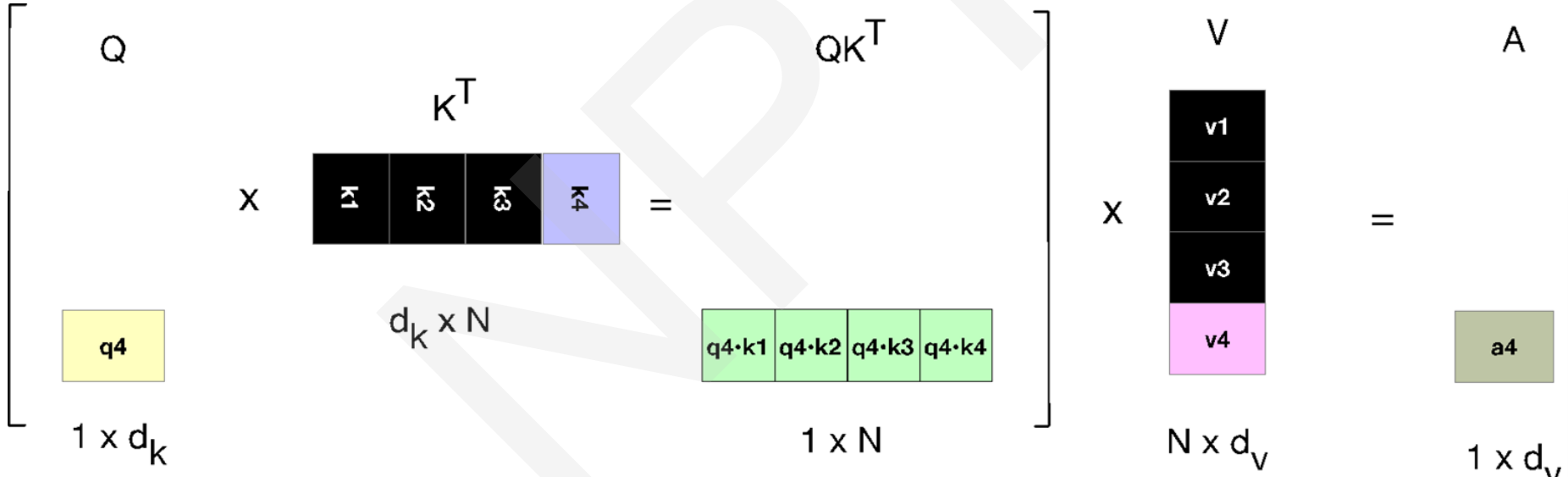
But don't want to **recompute** the key and value vectors for all the prior tokens $x_{<i}$

Instead, store key and value vectors in memory in the KV cache, and then we can just grab them from the cache

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

KV Cache





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

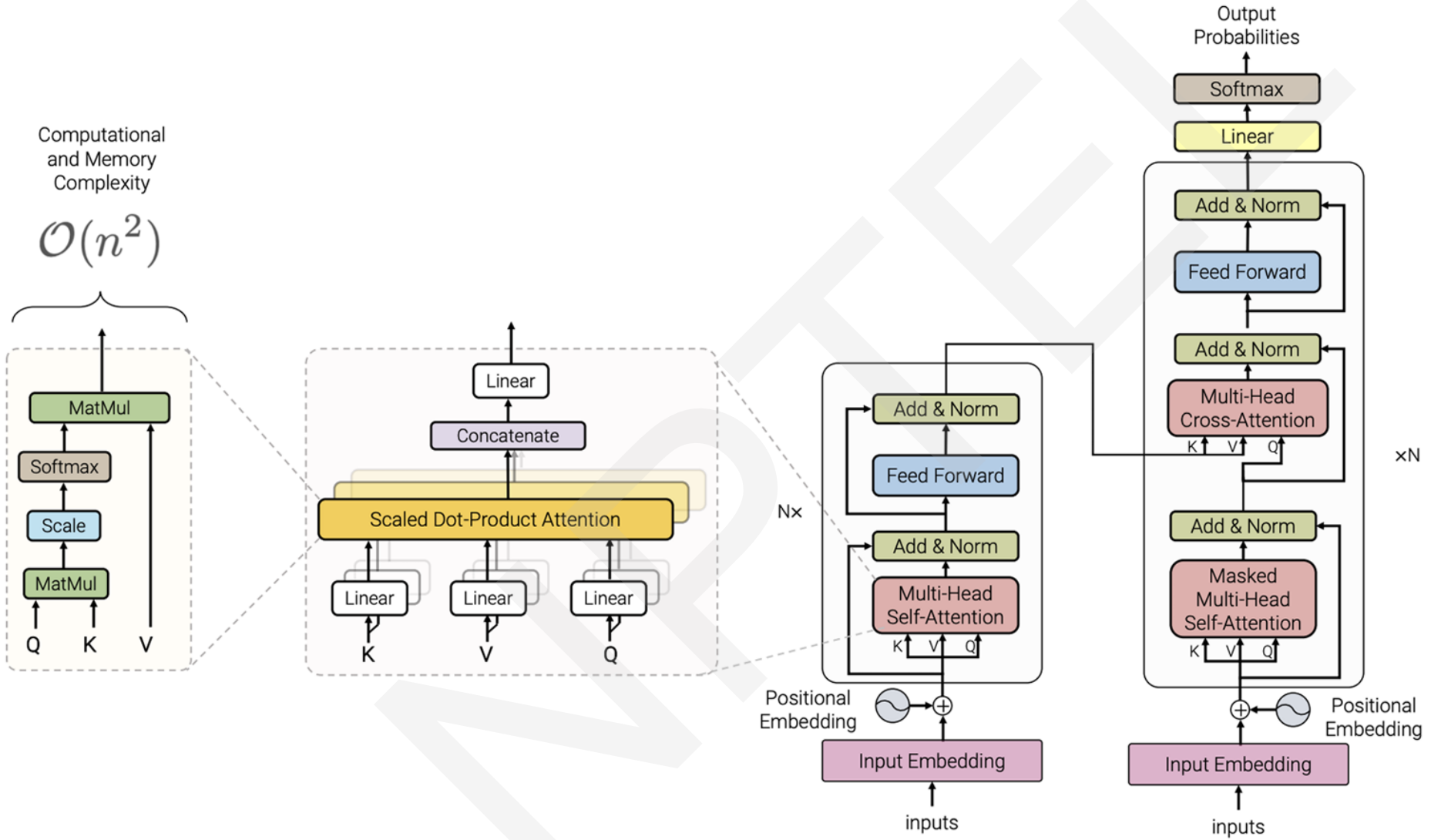
Transformer LMs are slow

	100 context tokens	200 context tokens	300 context tokens
Generate next 100 tokens	2.3s	2.7s	3.1s
Generate next 200 tokens	4.4s	4.6s	5.7s
Generate next 300 tokens	7.3s	8.4	9.7s

GPT-2 medium timed on a single 2080Ti

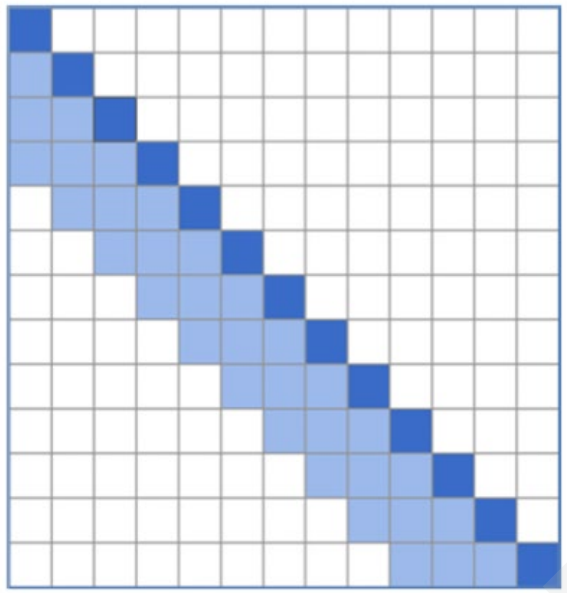
https://people.cs.umass.edu/~miyyer/cs685/

Quadratic Complexity of Self-Attention



https://people.cs.umass.edu/~miyyer/cs685/

Local Attention

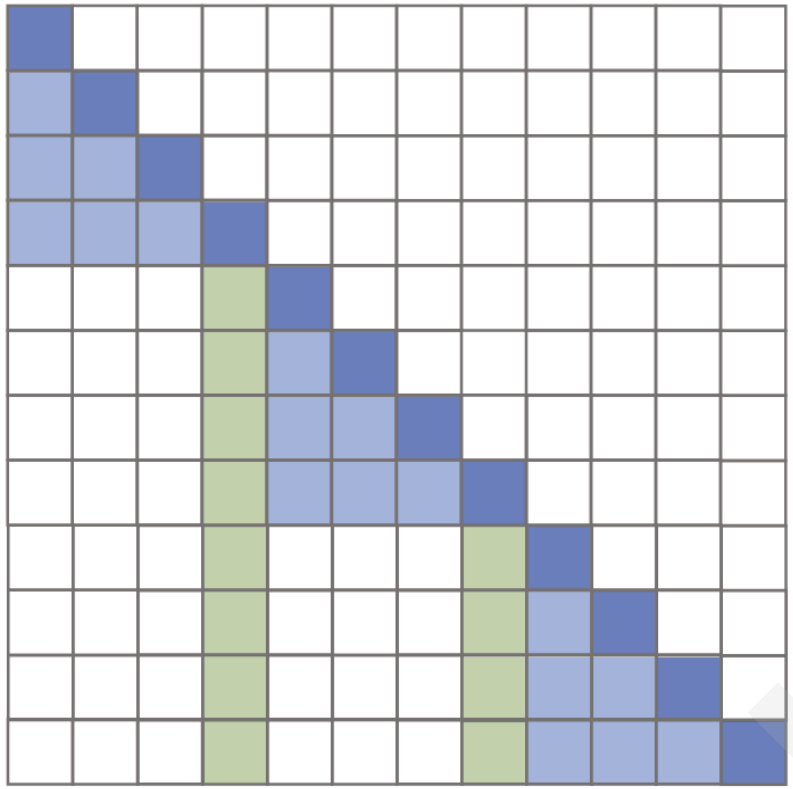


Position based sparse attention: Each token attends to previous *k* tokens (e.g., in decoder)

Other variations: *Attend to tokens in a block, sliding window based attention*, etc.

https://people.cs.umass.edu/~miyyer/cs685/

Sparse Transformer



Half of the heads are used for local attention

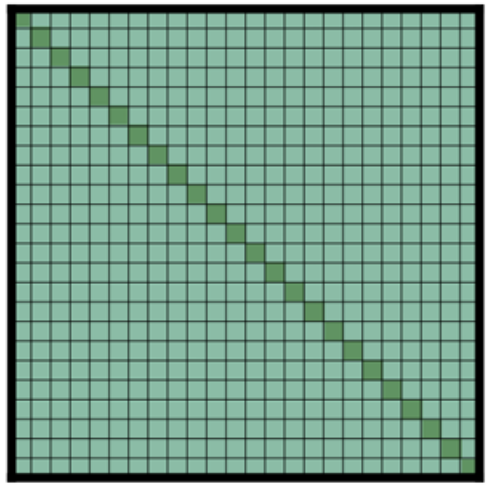
$$\hat{A}_{ij} = \begin{cases} Q_i(K)_j^\top), & \text{if} \lfloor j/N \rfloor = \lfloor i/N \rfloor \\ 0 & \text{otherwise} \end{cases}$$

The remaining half are dedicated to fixed strided patterns

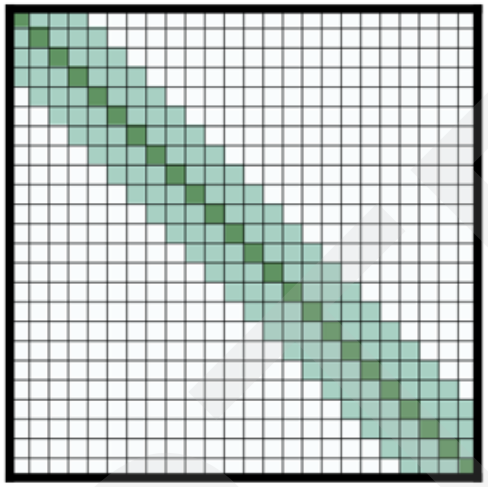
$$\hat{A}_{ij} = \begin{cases} Q_i(K)_j^\top), & \text{if}(i-j) \mod N = 0 \\ 0 & \text{otherwise} \end{cases}$$

Tay et al., 2022. Survey on efficient Transformers

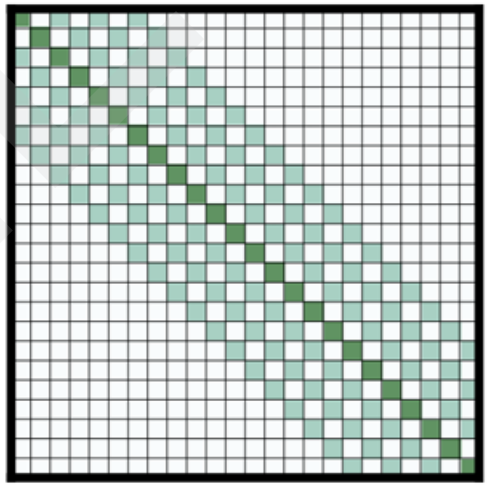
Longformer: Dilated Sliding Window Attention



(a) Full n^2 attention



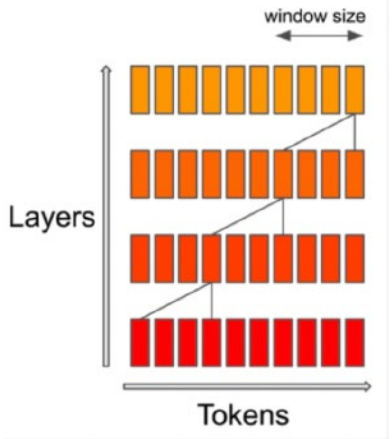
(b) Sliding window attention



(c) Dilated sliding window

Sliding window attention: Attention pattern employs a fixed-size window (*w*) attention surrounding each token

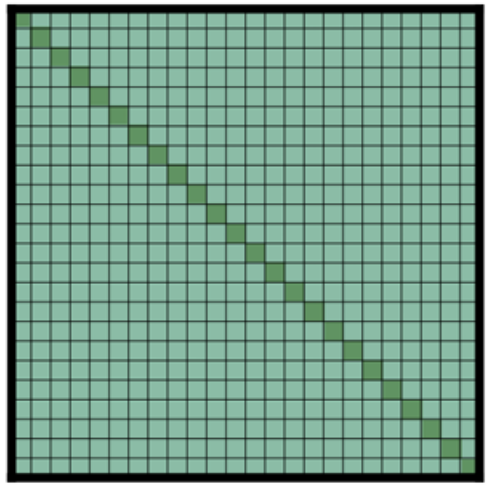
In a transformer with *L* layers, the receptive field size at the top layer is *L x w*



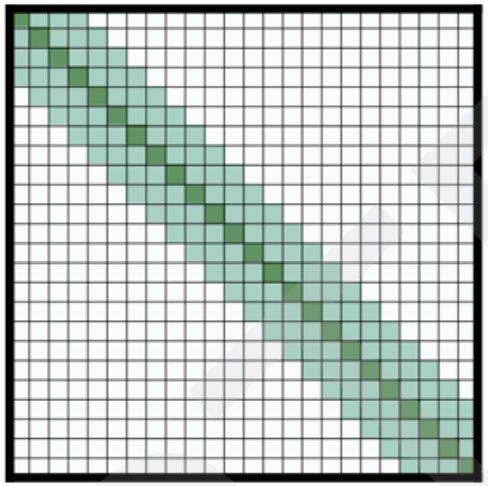
Effective Context Length

Tay et al., 2022. Survey on efficient Transformers

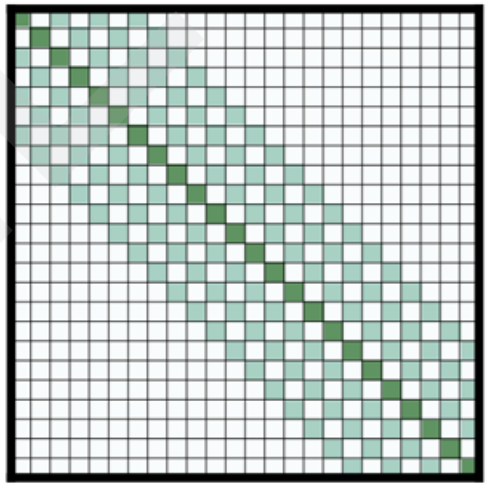
Longformer: Dilated Sliding Window Attention



(a) Full n^2 attention



(b) Sliding window attention

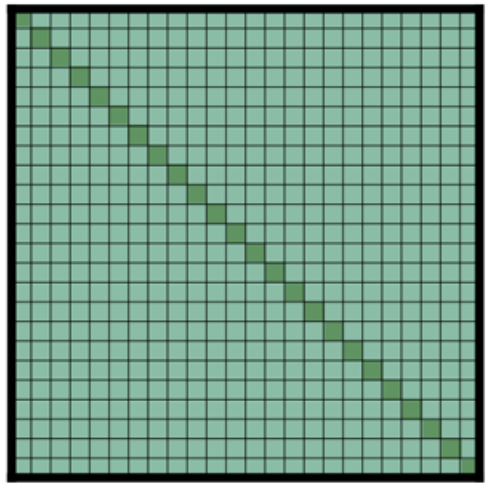


(c) Dilated sliding window

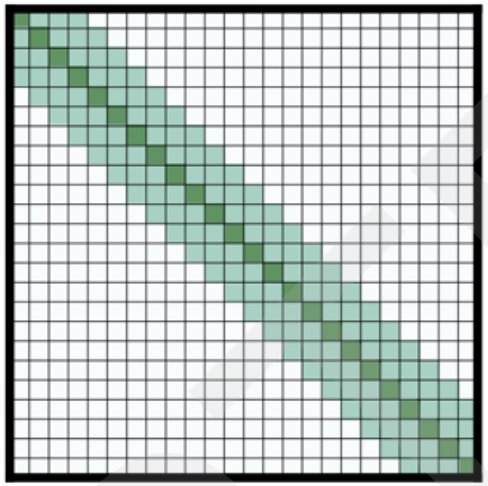
Dilated Sliding window attention: To further increase the receptive field without increasing computation, the sliding window can be "dilated". Assuming dilation *d*, with *L* layers, the receptive field size at the top layer is *L x d x w*

Tay et al., 2022. Survey on efficient Transformers

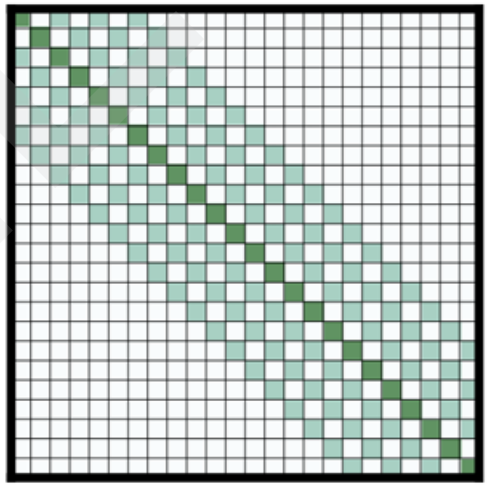
Longformer: Dilated Sliding Window Attention



(a) Full n^2 attention



(b) Sliding window attention

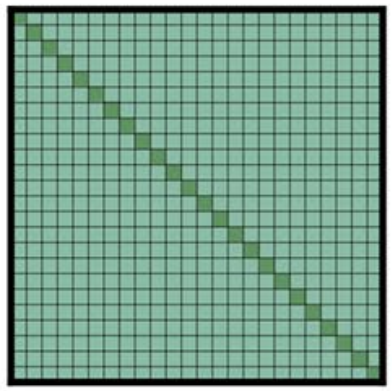


(c) Dilated sliding window

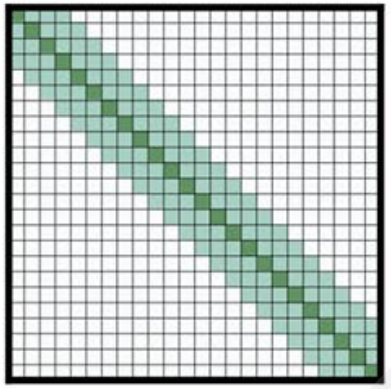
Dilated Sliding window attention: Different dilation configurations per head improves performance by allowing some heads without dilation to focus on local context, while others with dilation focus on longer context.

https://arxiv.org/pdf/2004.05150

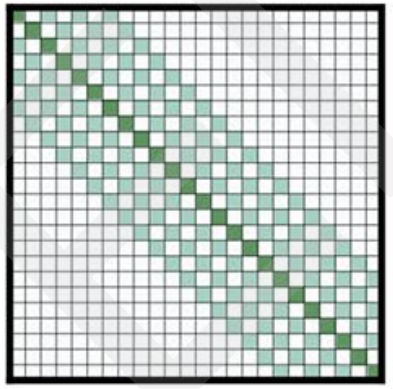
Longformer: Global Attention



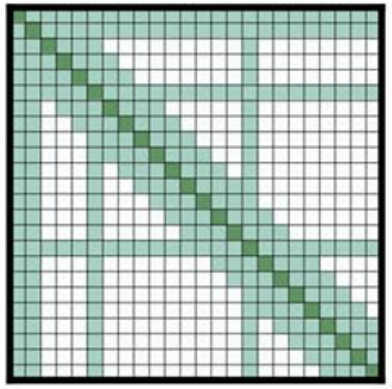
(a) Full n^2 attention



(b) Sliding window attention



(c) Dilated sliding window



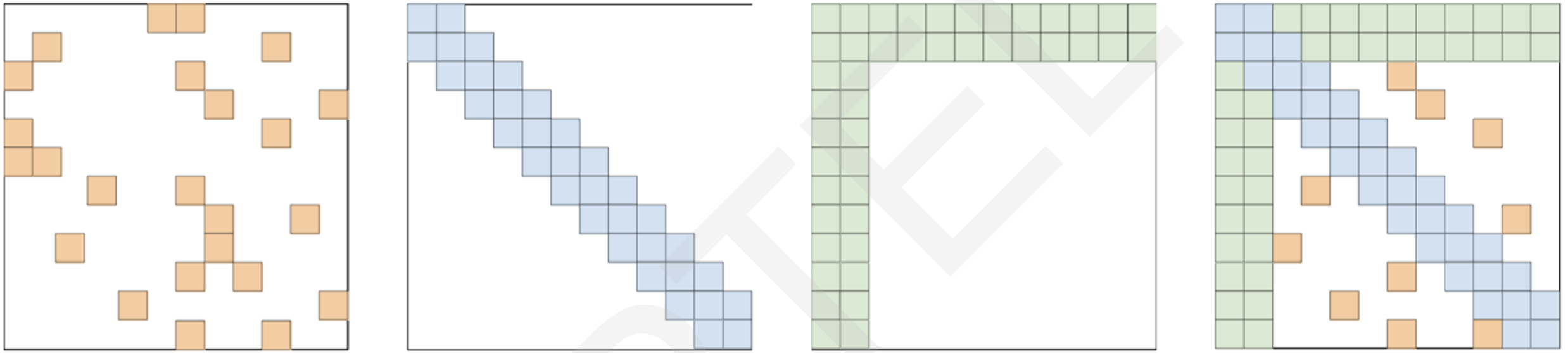
(d) Global+sliding window

Symmetric “global attention” is added on few pre-selected input locations.

Task Specific: For example for classification, global attention is used for the [CLS] token while in QA, global attention is provided on all question tokens.

https://arxiv.org/pdf/2004.05150

BIGBIRD

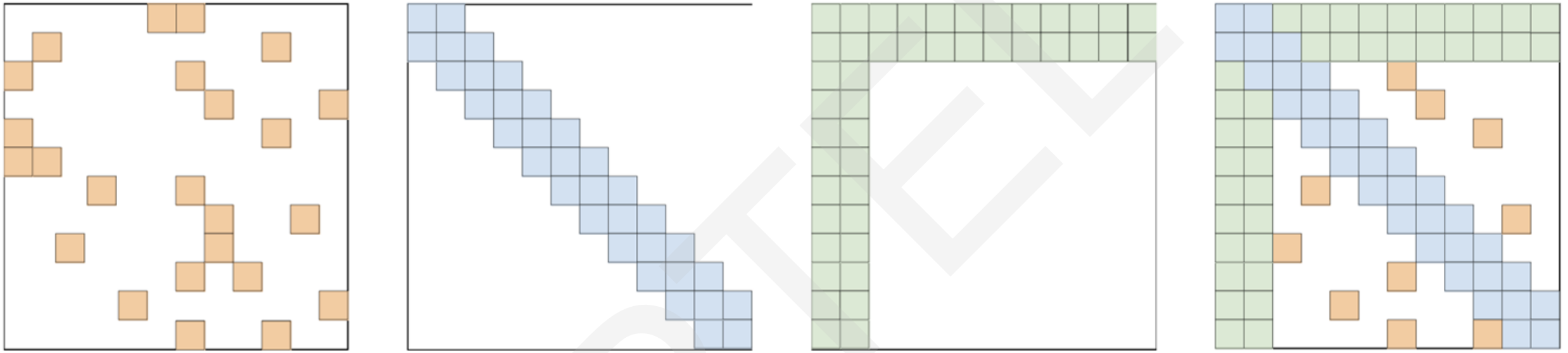


(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Building blocks of the attention mechanism used in BIGBIRD. White color indicates absence of attention. (a) random attention with r = 2, (b) sliding window attention with w = 3, (c) global attention with g = 2, (d) the combined BIGBIRD model

https://arxiv.org/pdf/2007.14062

BIGBIRD



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

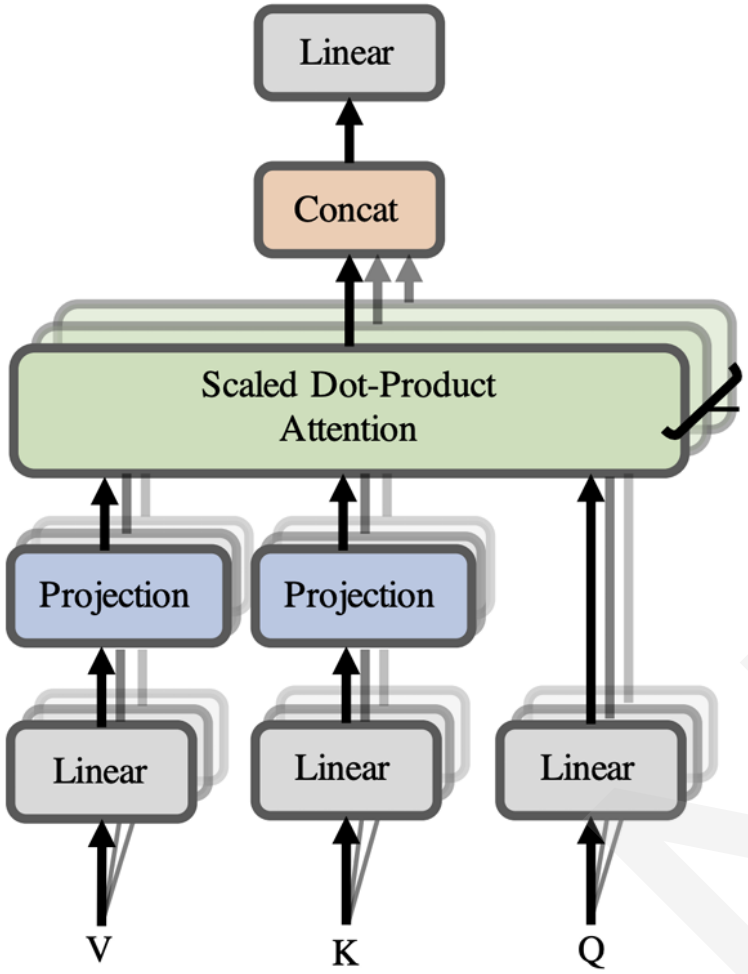
Global tokens can be defined in two ways:

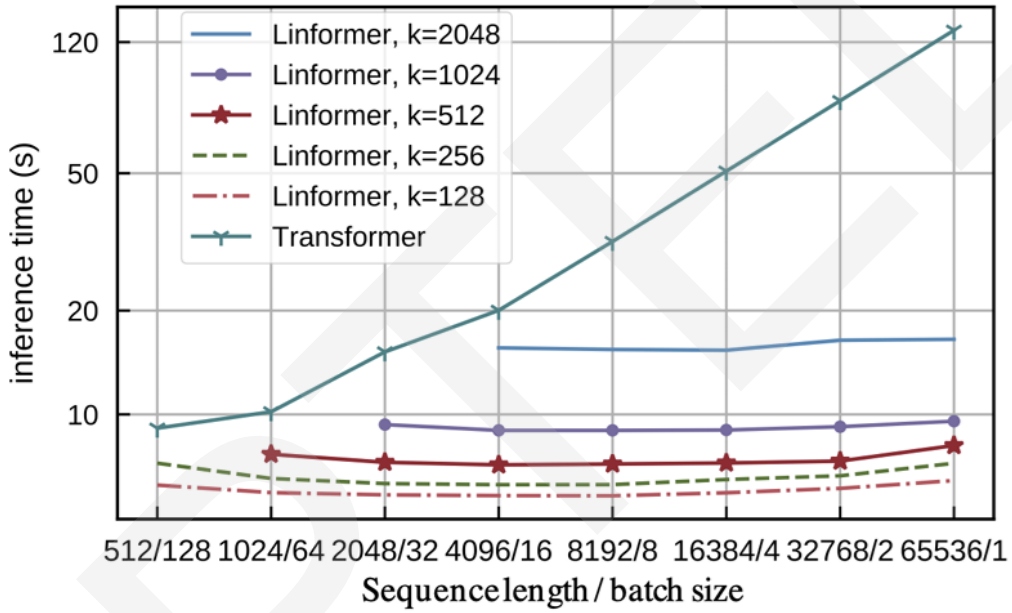
BIGBIRD-ITC: In internal transformer construction (ITC), we make some existing tokens “global”, which attend over the entire sequence.

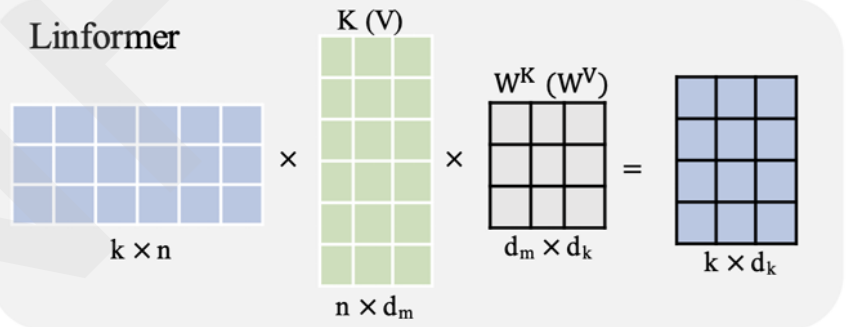
BIGBIRD-ETC: In extended transformer construction (ETC), we include additional “global” tokens such as [CLS].

https://arxiv.org/pdf/2007.14062

Linformer







Main idea: produce a low-rank approximation of the N x N attention matrix by simply projecting the N x d keys and values to N x k where k <<<<< N

https://people.cs.umass.edu/~miyyer/cs685/

Linformer

$$\overline{\text{head}_i} = \text{Attention}(QW_i^Q, E_iKW_i^K, F_iVW_i^V)$$

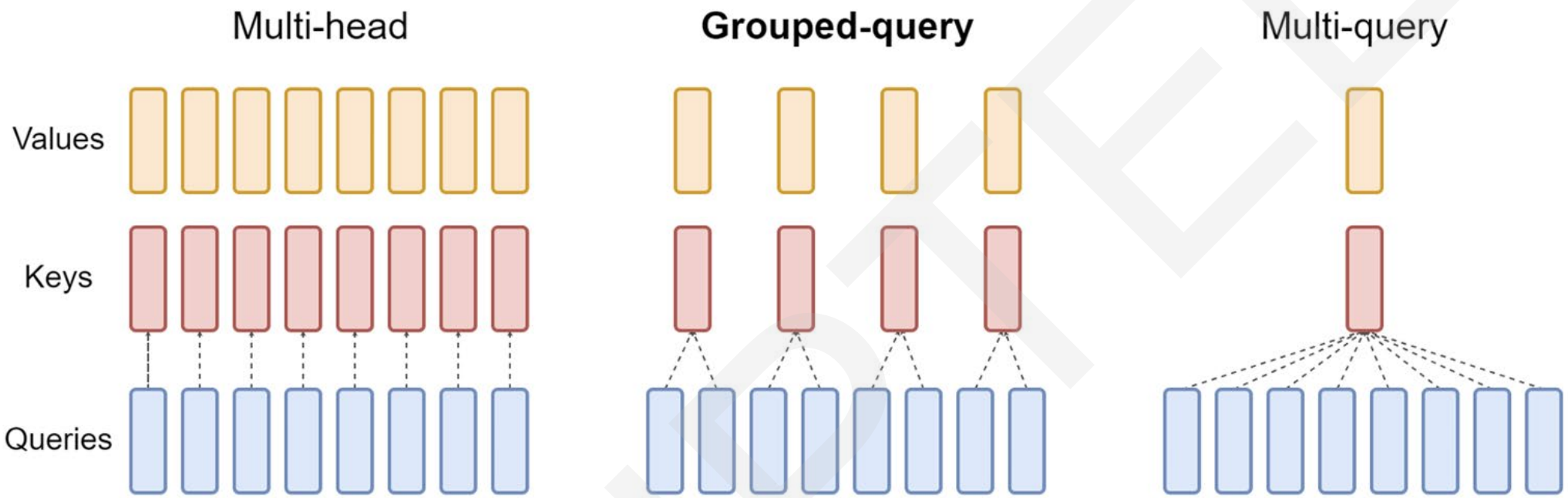
$$= \underbrace{\text{softmax}\left(\frac{QW_i^Q(E_iKW_i^K)^T}{\sqrt{d_k}}\right)}_{\bar{P}:n\times k} \cdot \underbrace{F_iVW_i^V}_{k\times d}$$

The above operations only require *O(nk)* time and space complexity.

Main idea: produce a low-rank approximation of the N x N attention matrix by simply projecting the N x d keys and values to N x k where k <<<<< N

https://arxiv.org/pdf/2006.04768v3

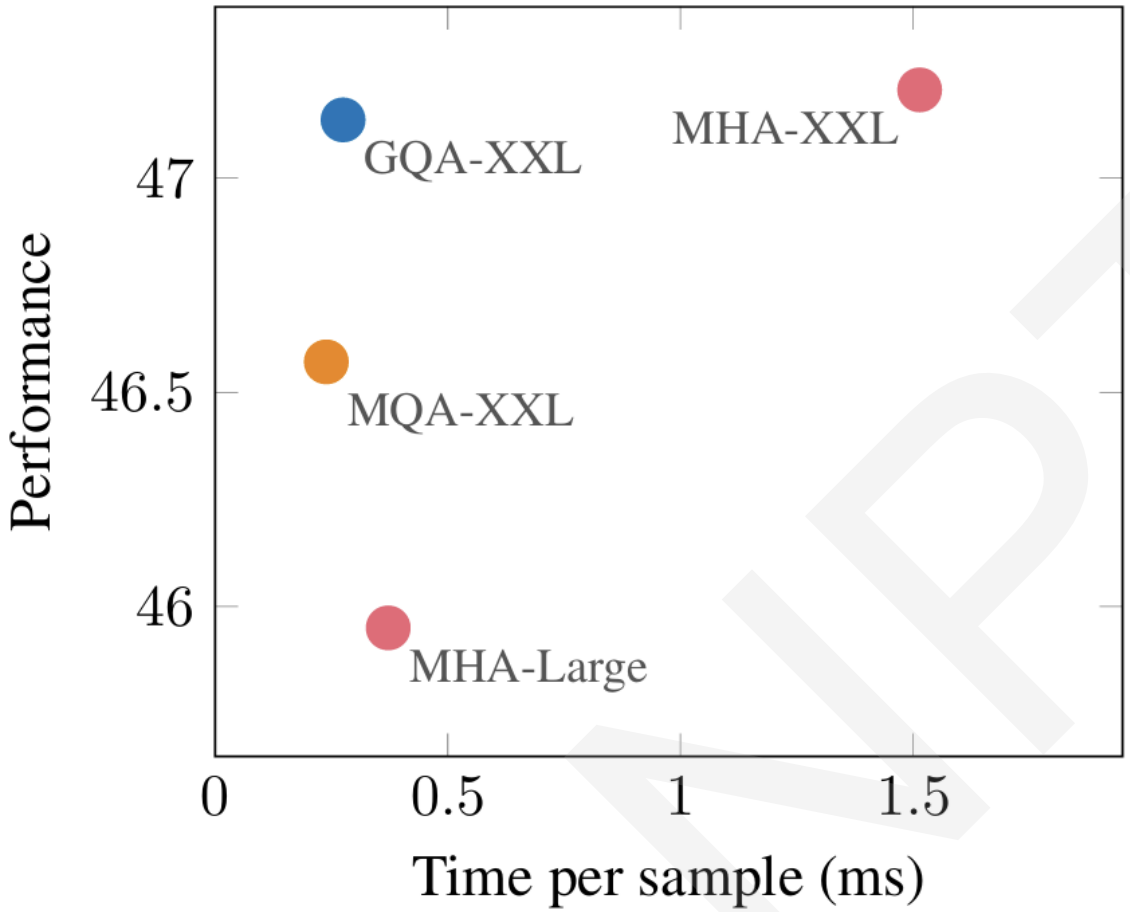
Multi Query Attention (MQA) and Grouped Query Attention (GQA)



Multi-head attention has H query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each group of query heads, interpolating between multi-head and multi-query attention.

https://arxiv.org/pdf/2305.13245

Multi Query Attention (MQA) and Grouped Query Attention (GQA): Time-Performance tradeoff

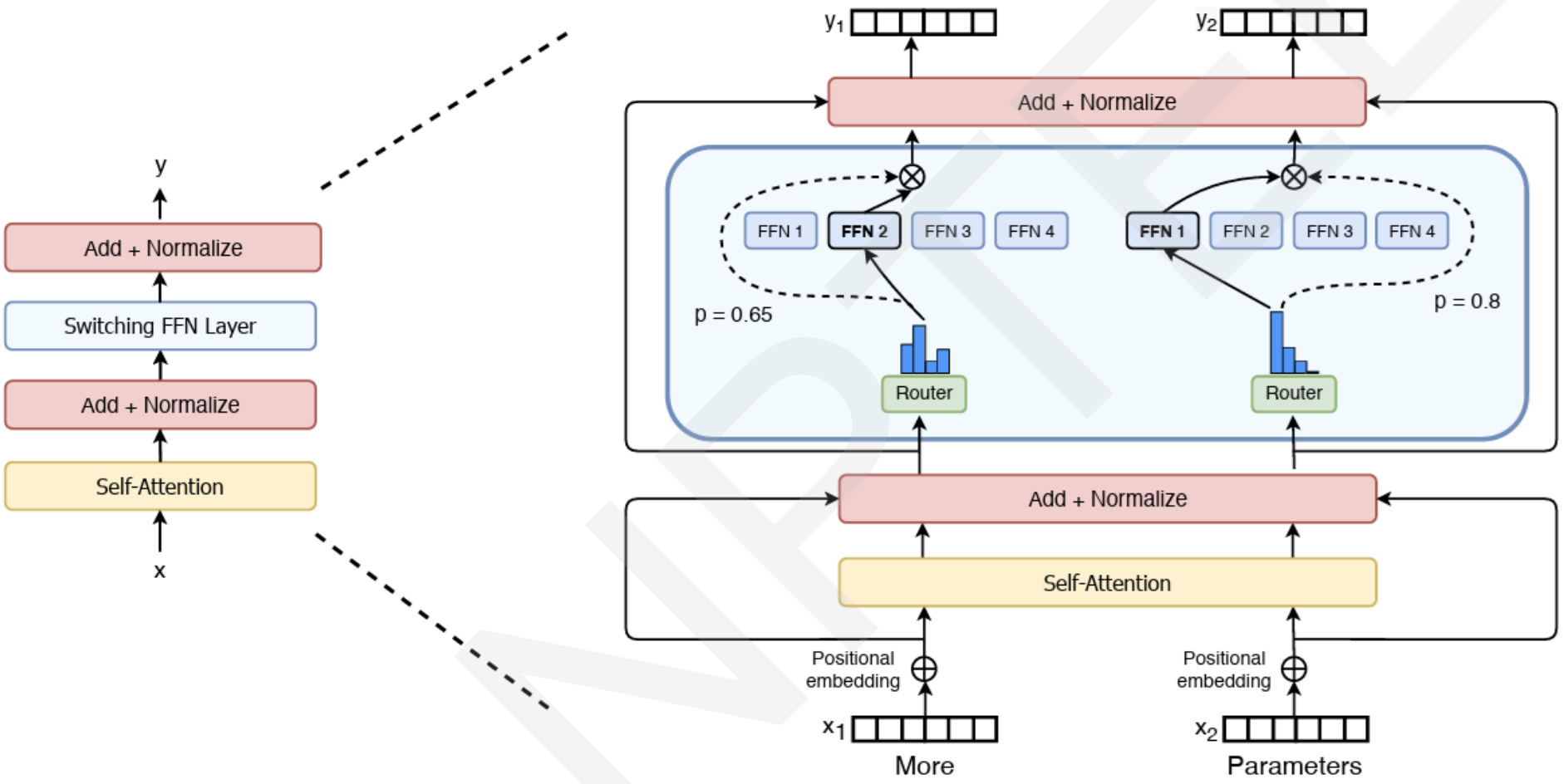


Average performance on all tasks as a function of average inference time per sample for T5-Large and T5-XXL with multihead attention, and 5% uptrained T5-XXL with MQA and GQA-8 attention.

What is uptraining? Will discuss after pretraining

https://arxiv.org/pdf/2305.13245

Sparse Mixture of Experts (MoE)



The MoE layer operates independently on the tokens in the sequence. The router independently routes each token.

Illustration of a Switch Transformer encoder block.

https://arxiv.org/pdf/2101.03961

Sparse Mixture of Experts (MoE)

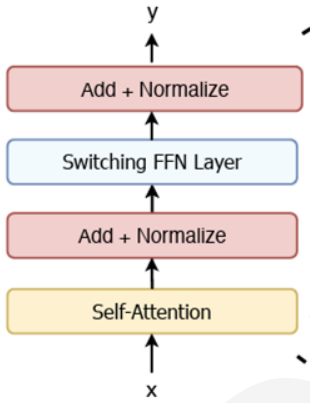
The router variable W_r produces logits $h(x) = W_r \cdot x$

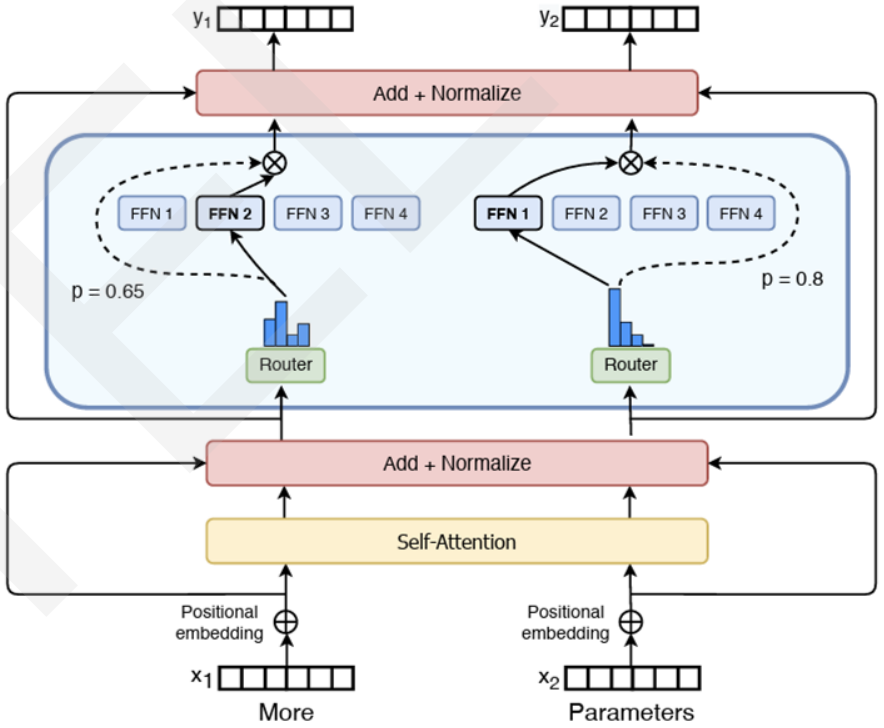
Gate value for expert *i*

$$p_i(x) = \frac{e^{h(x)_i}}{\sum_j^N e^{h(x)_j}}$$

The top-k gate (set *T*) values are selected for routing the token x.
Final output:

$$y = \sum_{i \in \mathcal{T}} p_i(x) E_i(x)$$

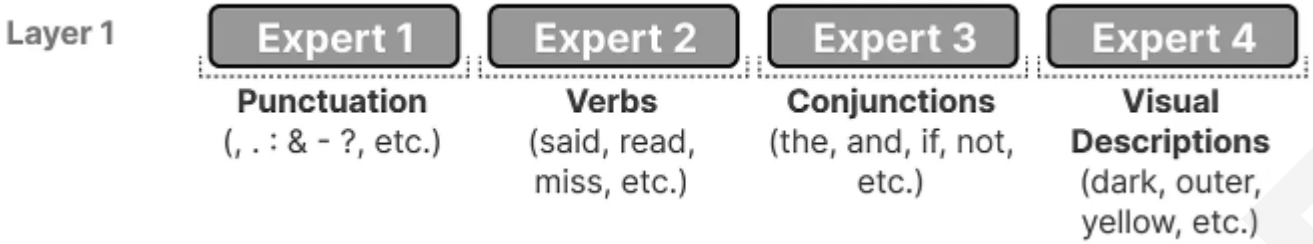




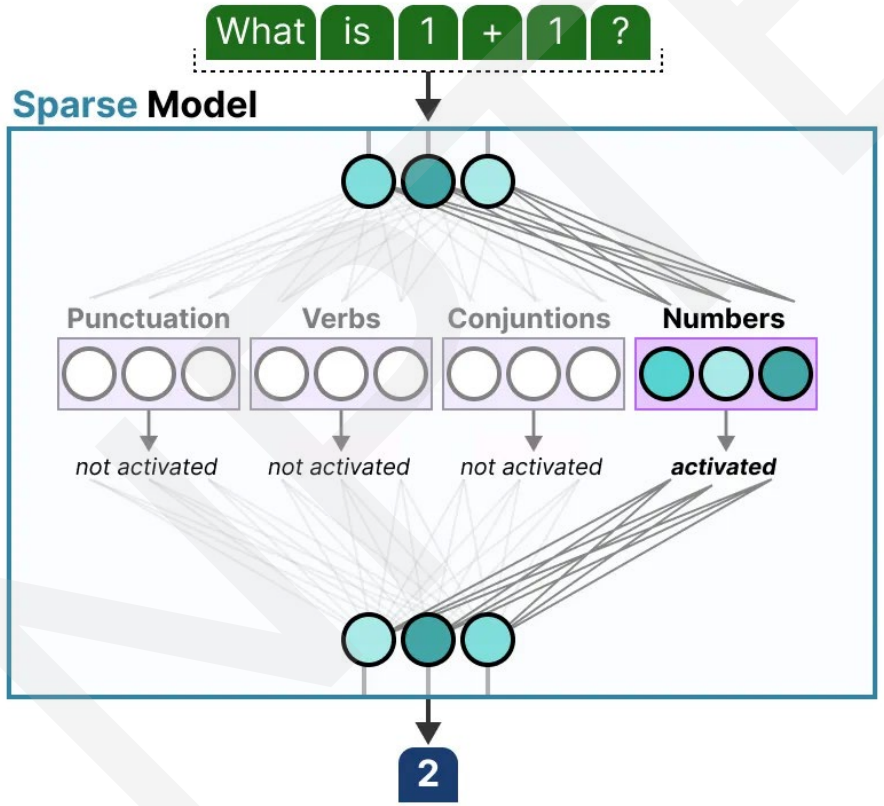
k = 1 routing strategy is later referred to as a Switch layer

https://arxiv.org/pdf/2101.03961

What do the experts learn?



More specifically, their expertise is in handling specific tokens in specific contexts.



https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mixture-of-experts

REFERENCES

Daniel Jurafsky and James H. Martin. 2024. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models, 3rd edition. Online manuscript released August 20, 2024. [Chapter 10]


THANK YOU