Attention is All You Need

(Vaswani et. al. 2017)

Slides and figures when not cited are from:

Mausam, Jay Alammar ‘The Illustrated Transformer’

Attention in seq2seq models (Bahdanau 2014)

The context vector c_i is, then, computed as a weighted sum of these annotations h_i:

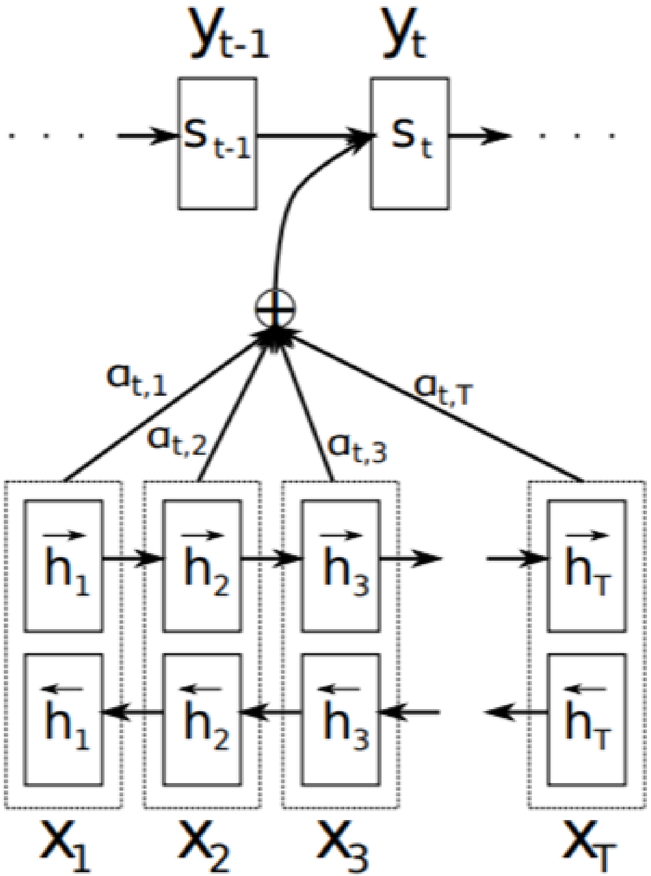
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j. \tag{5}$$

The weight α_{ij} of each annotation h_j is computed by

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

where

$$e_{ij} = a(s_{i-1}, h_j)$$



Attention Functions

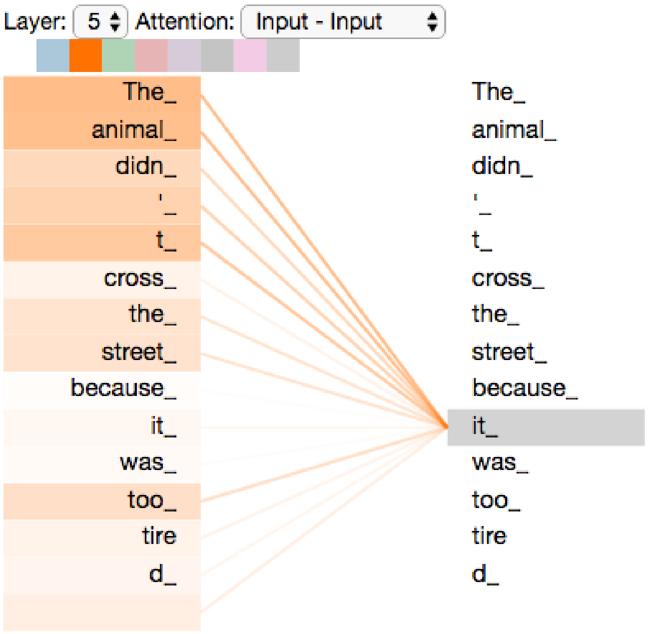
v: attended vec, q: query vec
$\mathrm{MLP}^{att}(q;v)$=

- Additive Attention: $\mathbf{u}g(\mathbf{W^1 v} + \mathbf{W^2 q})$
- Dot Product: $\mathbf{v} \cdot \mathbf{q}$
- Multiplicative Attention: $\mathbf{v}^\top \mathbf{W} \mathbf{q}$

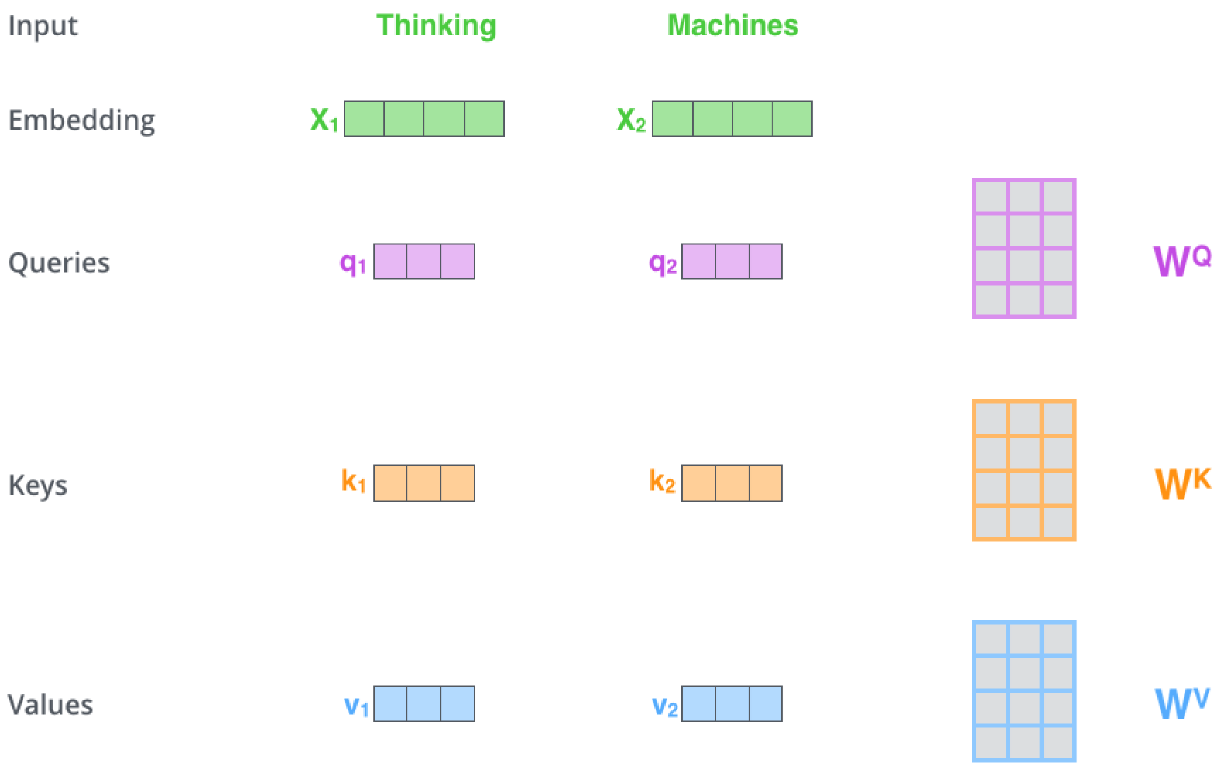
Multi-head attention

Self-attention (single-head, high-level)

"`The animal didn't cross the street because it was too tired`"



Self-attention (single-head, pt. 1)

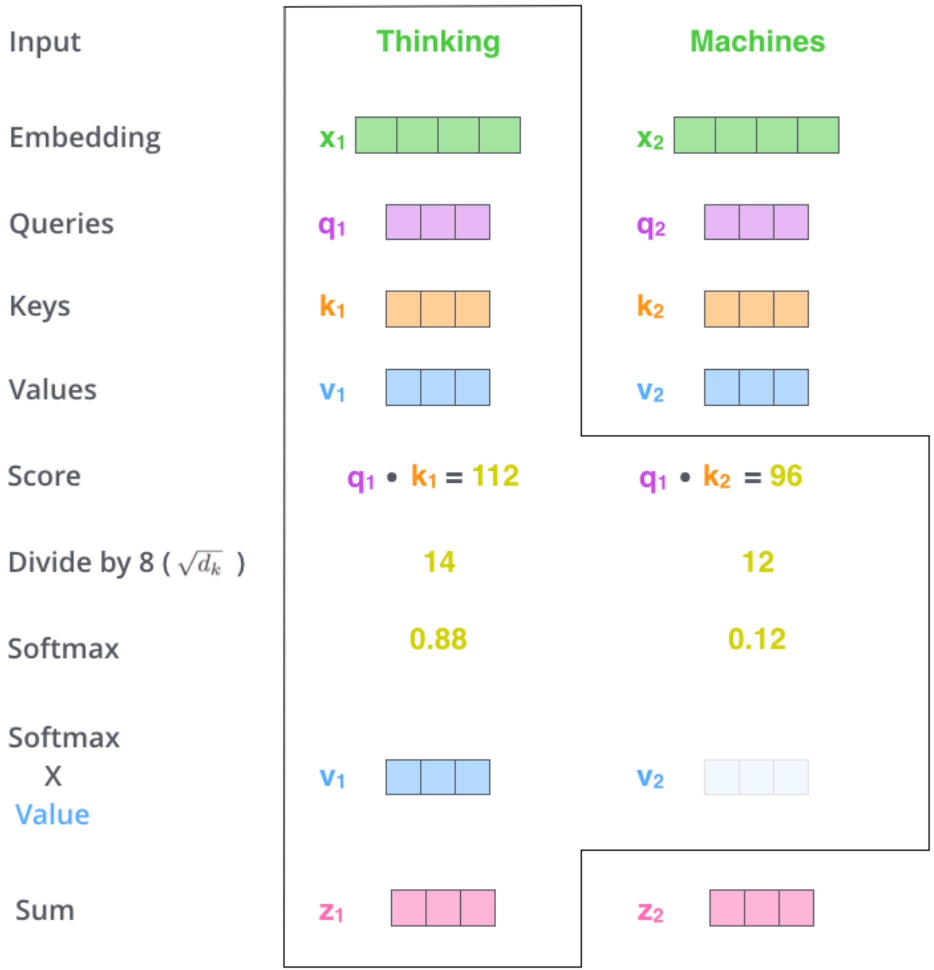


Creation of query, key and value vectors by multiplying by trained weight matrices

Separation of Value and Key

Matrix multiplications are quite efficient and can be done in aggregated manner

Self-attention (single-head, pt. 2)

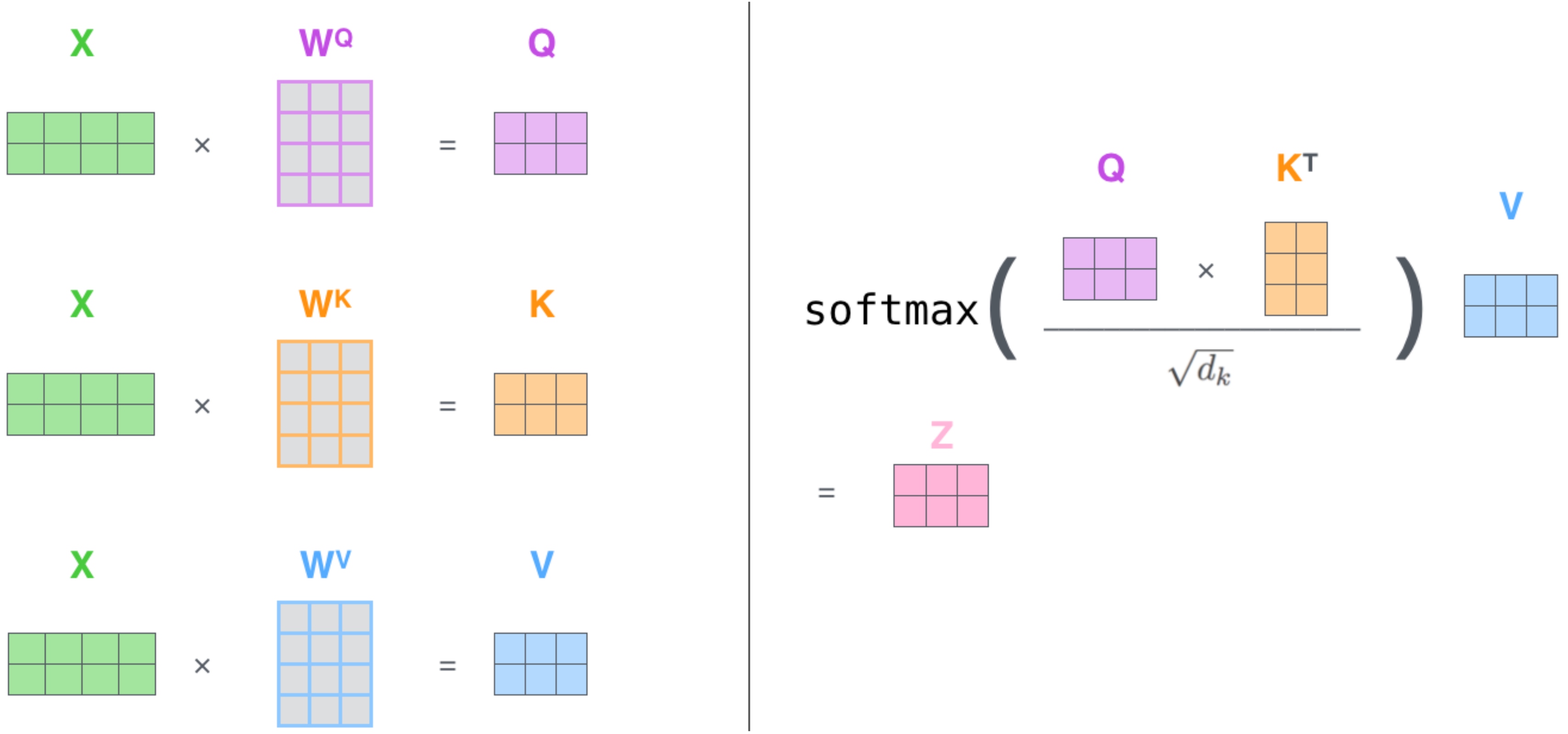


Mechanism similar to regular attention except for division factor

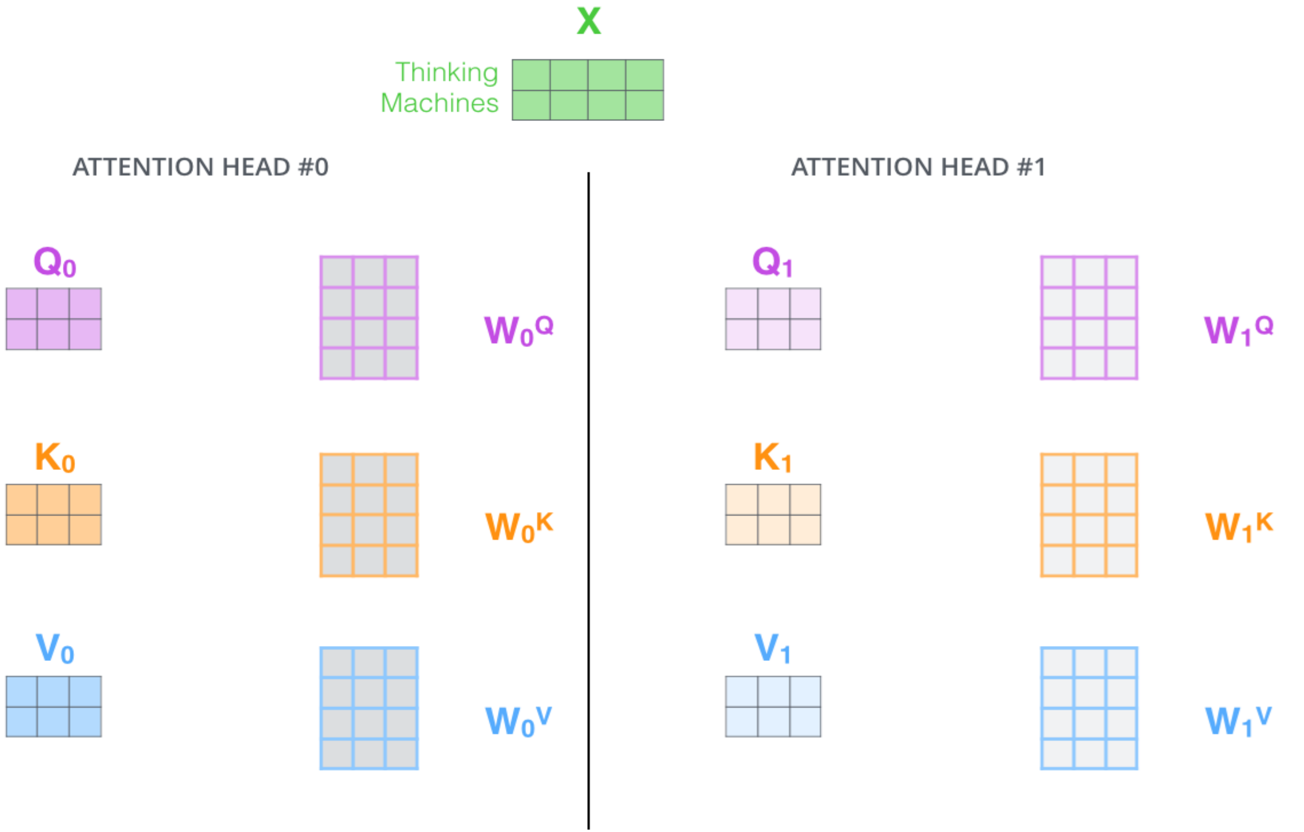
Paper's Justification:

To illustrate why the dot products get large, assume that the components of q and k are independent random variables with mean 0 and variance 1. Then their dot product, $q \cdot k$ has mean 0 and variance d_k

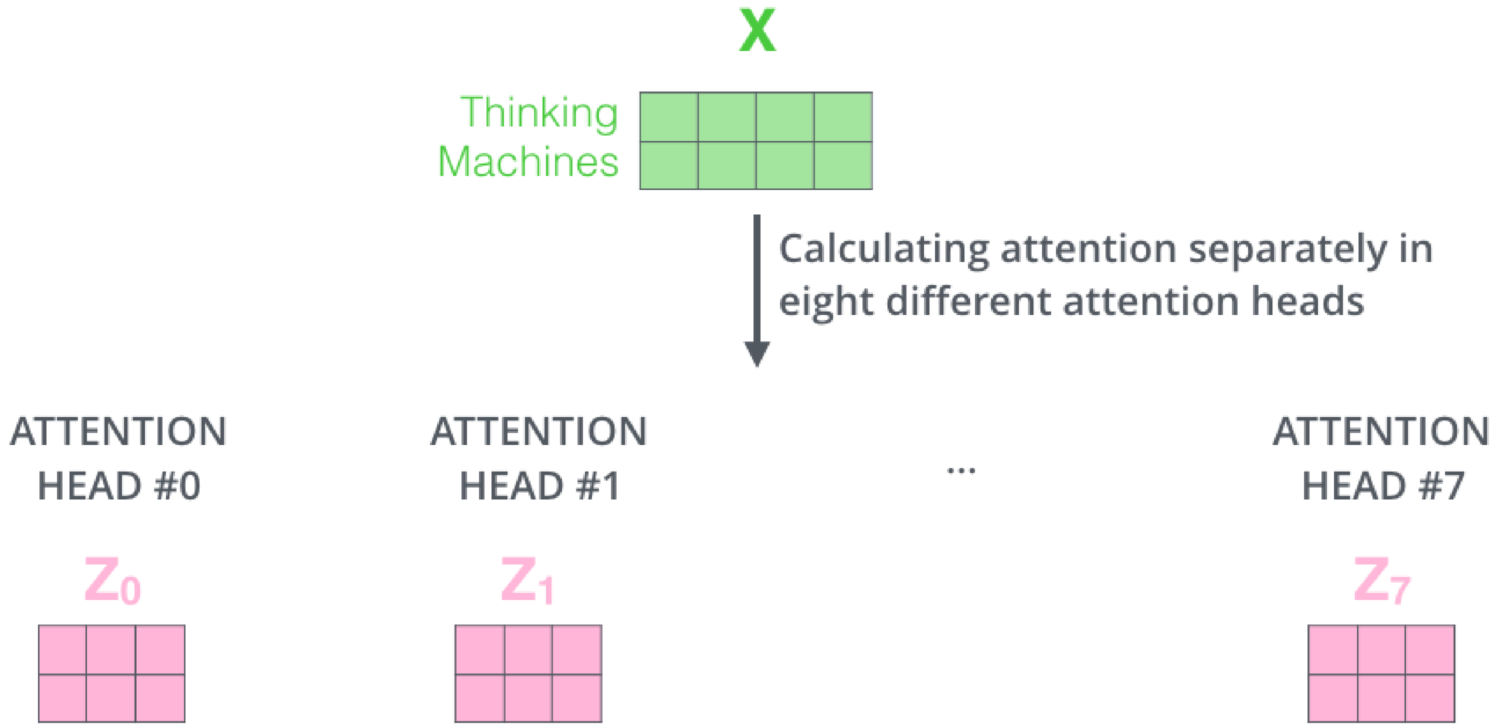
Self-attention (single-head, pt. 3)



Self-attention (multi-head)

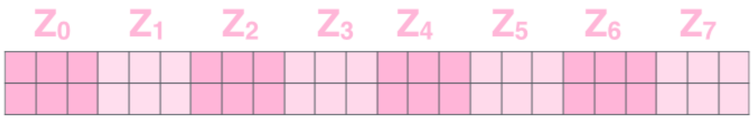


Self-attention (multi-head)



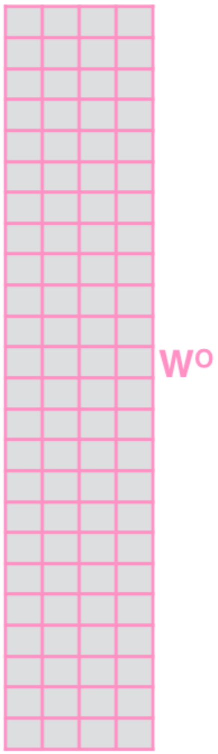
Self-attention (multi-head)

1) Concatenate all the attention heads



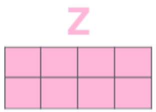
2) Multiply with a weight matrix W^O that was trained jointly with the model

X

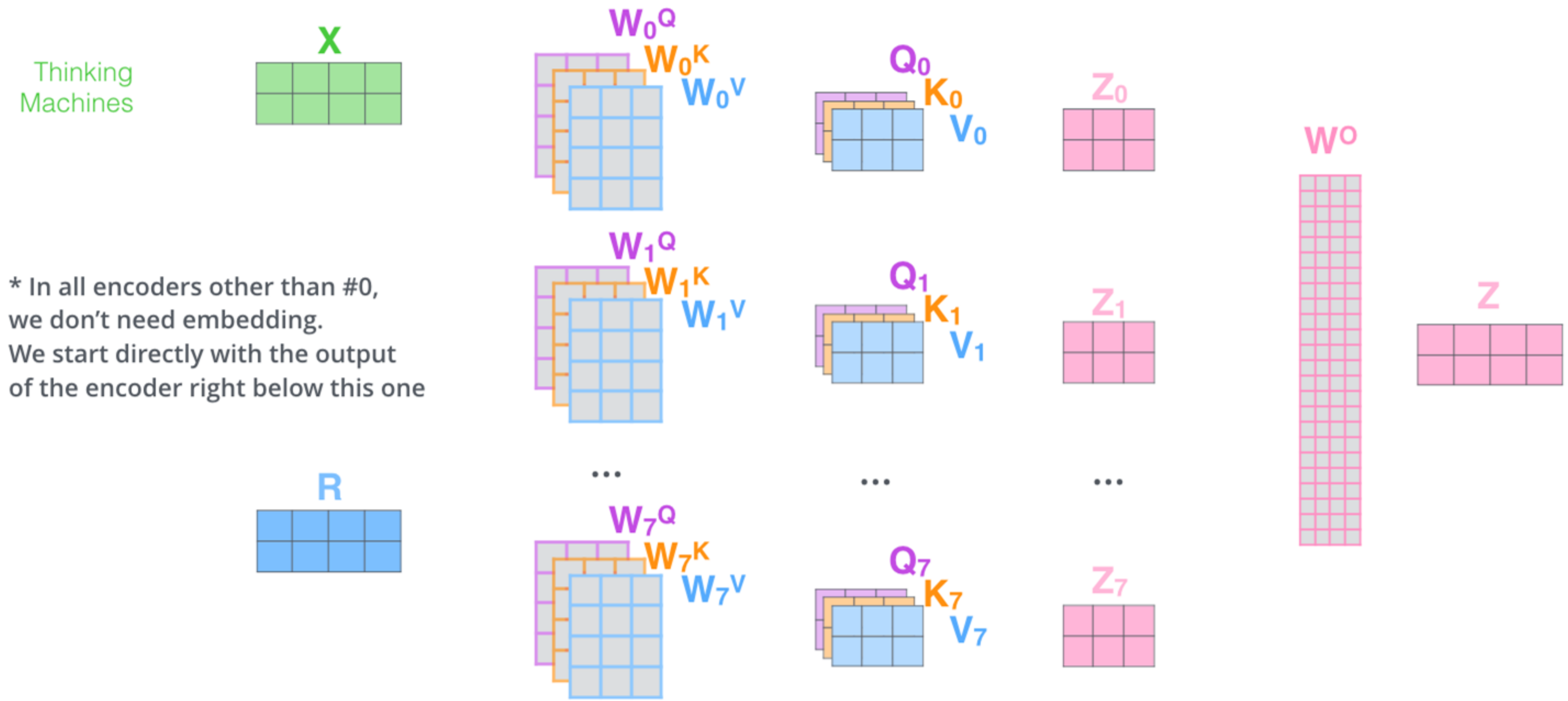


3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

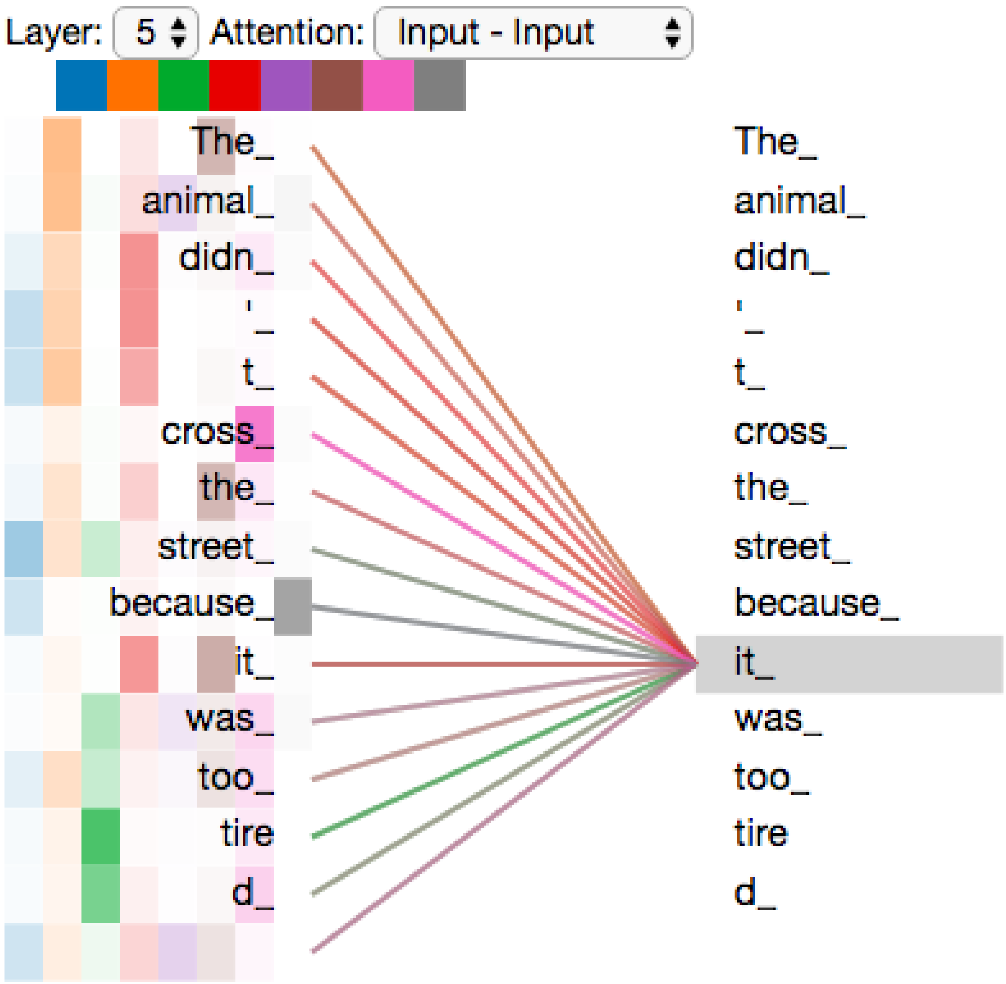
=



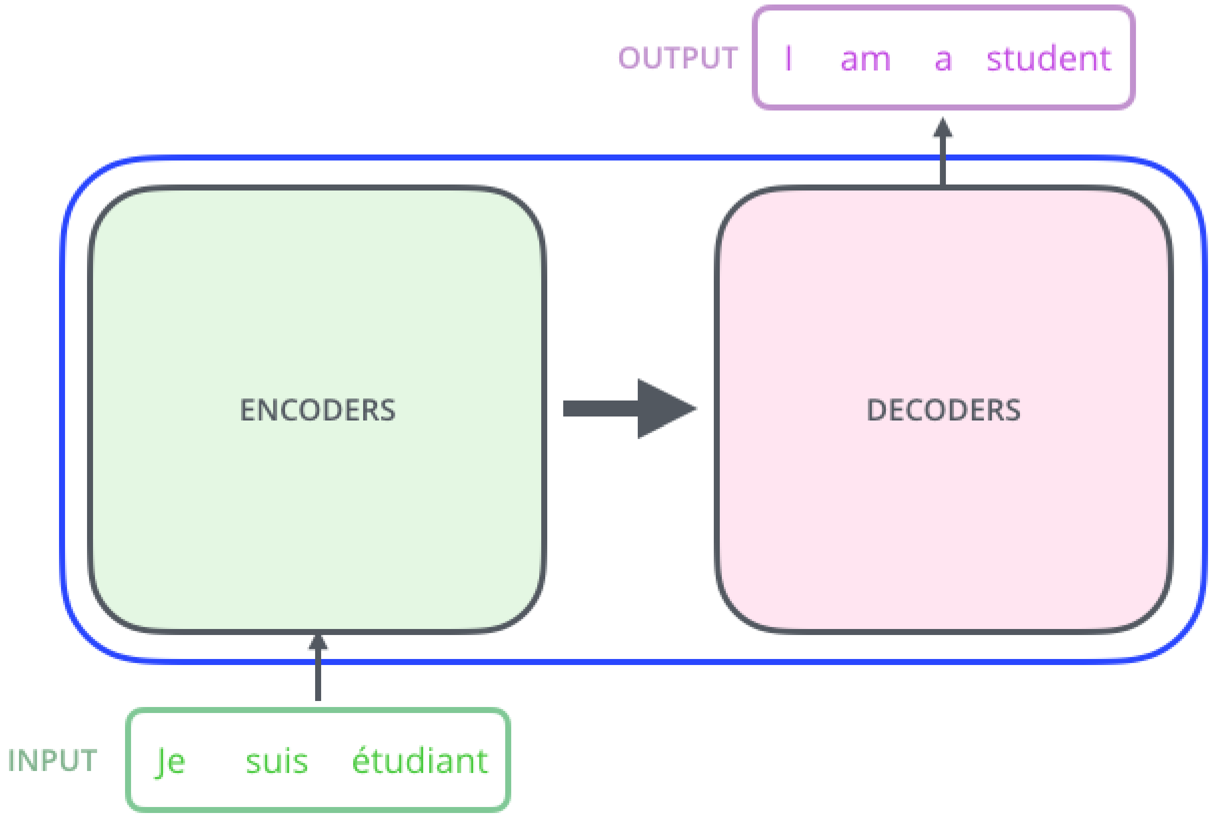
Self attention summary

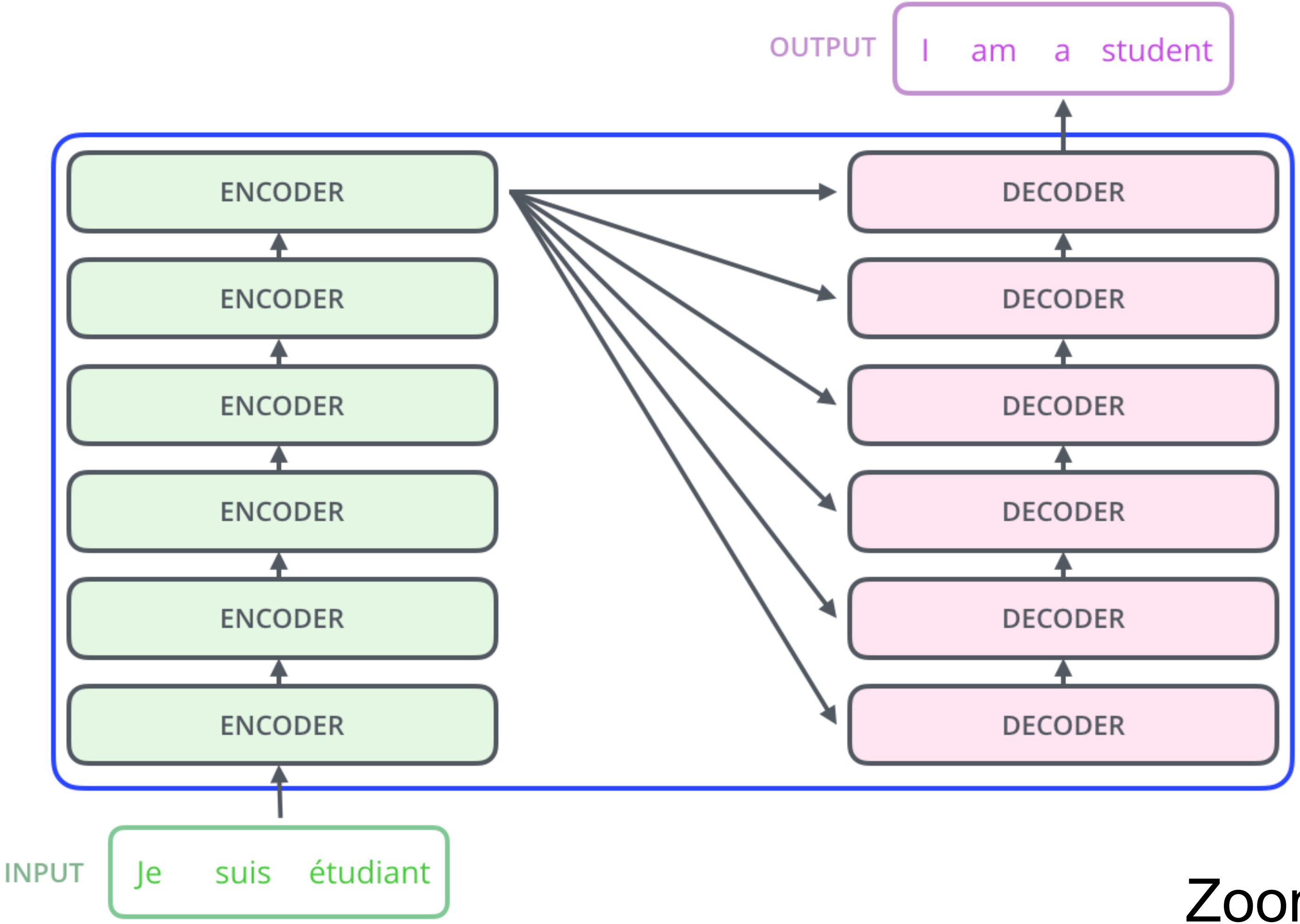


Self attention visualisation (Interpretable?!)

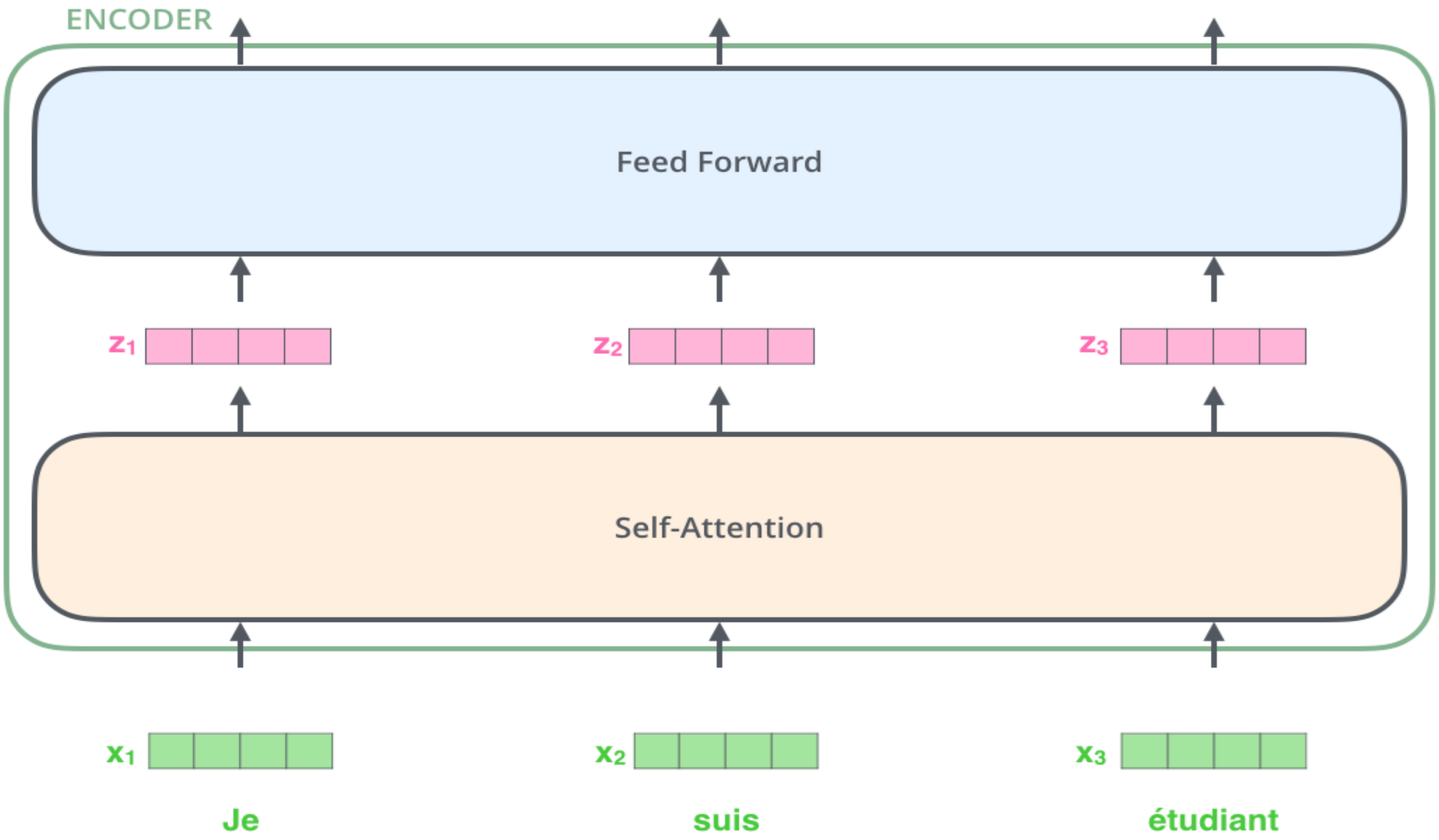


Transformer Architecture

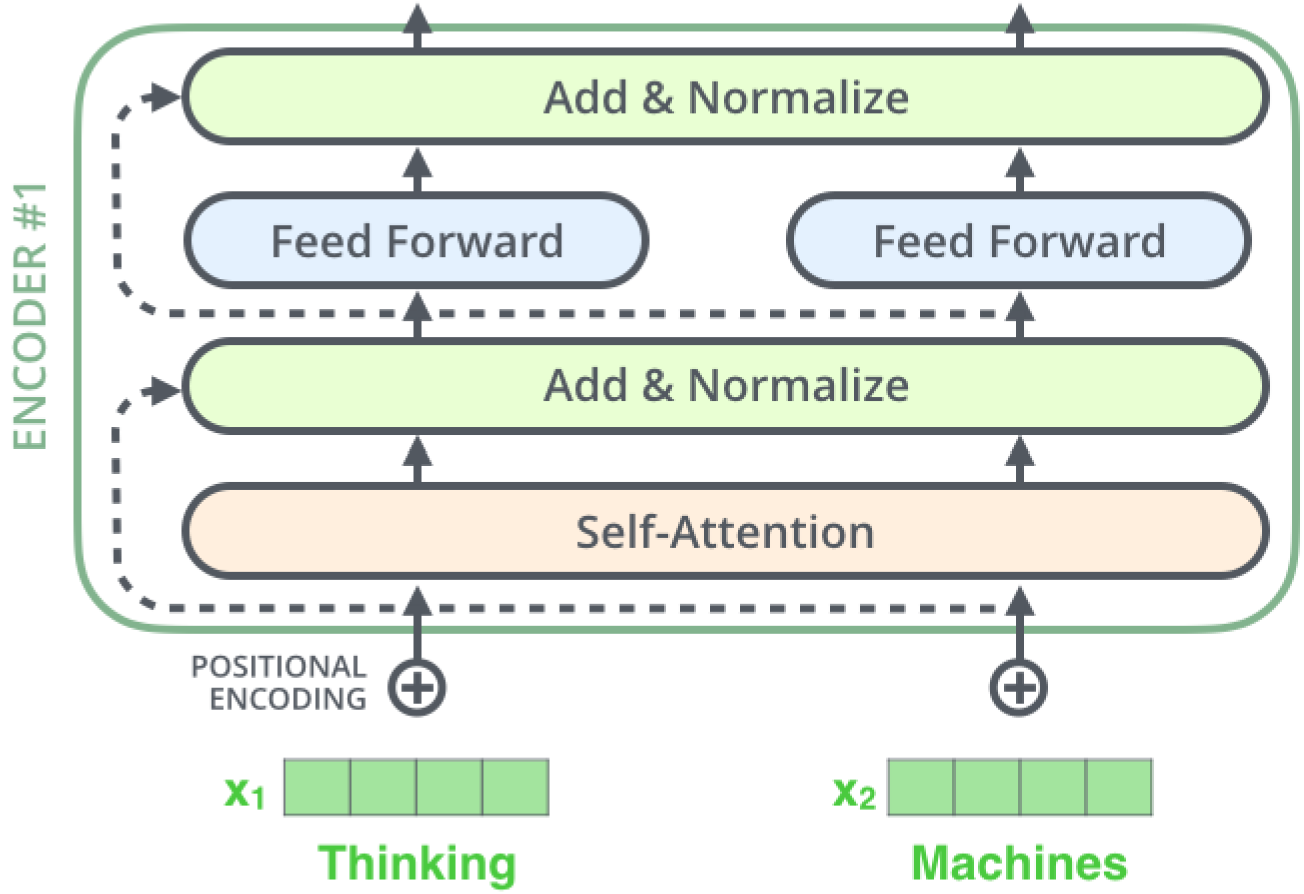

OUTPUT
I am a student
ENCODERS
DECODERS
INPUT
Je suis étudiant


OUTPUT
I am a student
ENCODER
ENCODER
ENCODER
ENCODER
ENCODER
ENCODER
DECODER
DECODER
DECODER
DECODER
DECODER
DECODER
INPUT
Je suis étudiant

Zooming in...

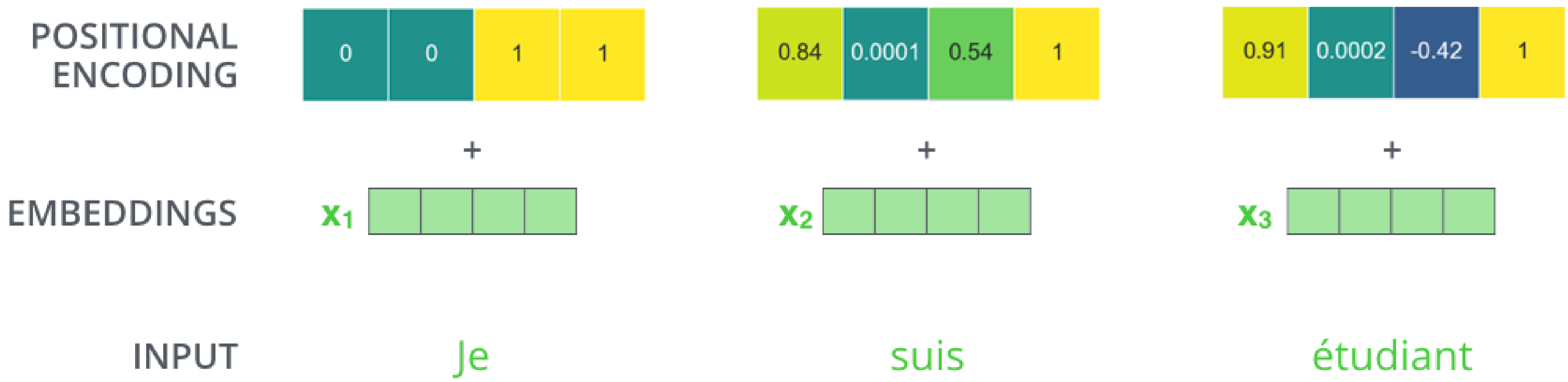


Zooming in further...



Adding residual connections...

A note on Positional embeddings

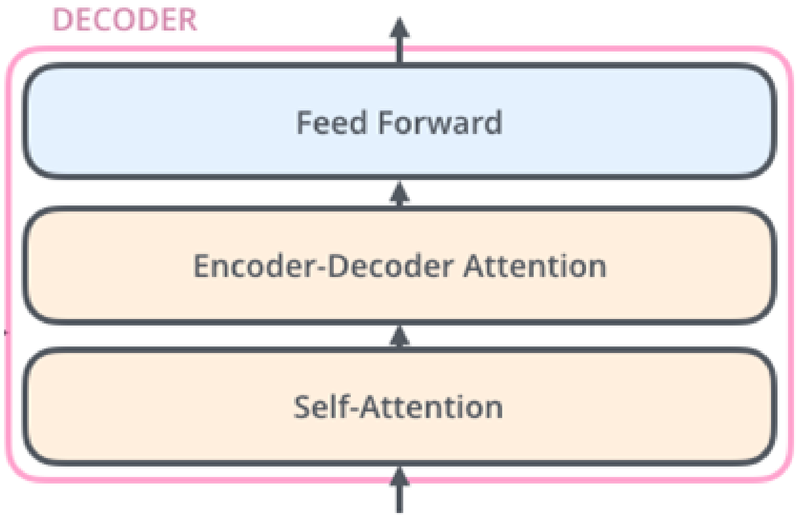


Positional embeddings can be extended to any sentence length but if any test input is longer than all training inputs then we will face issues.

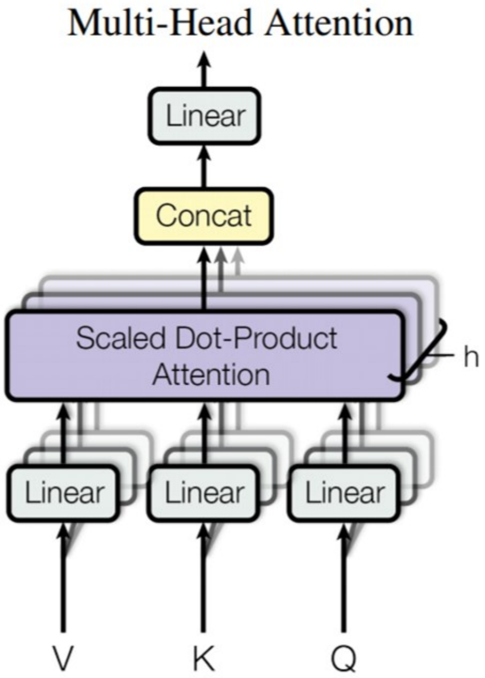
Decoders

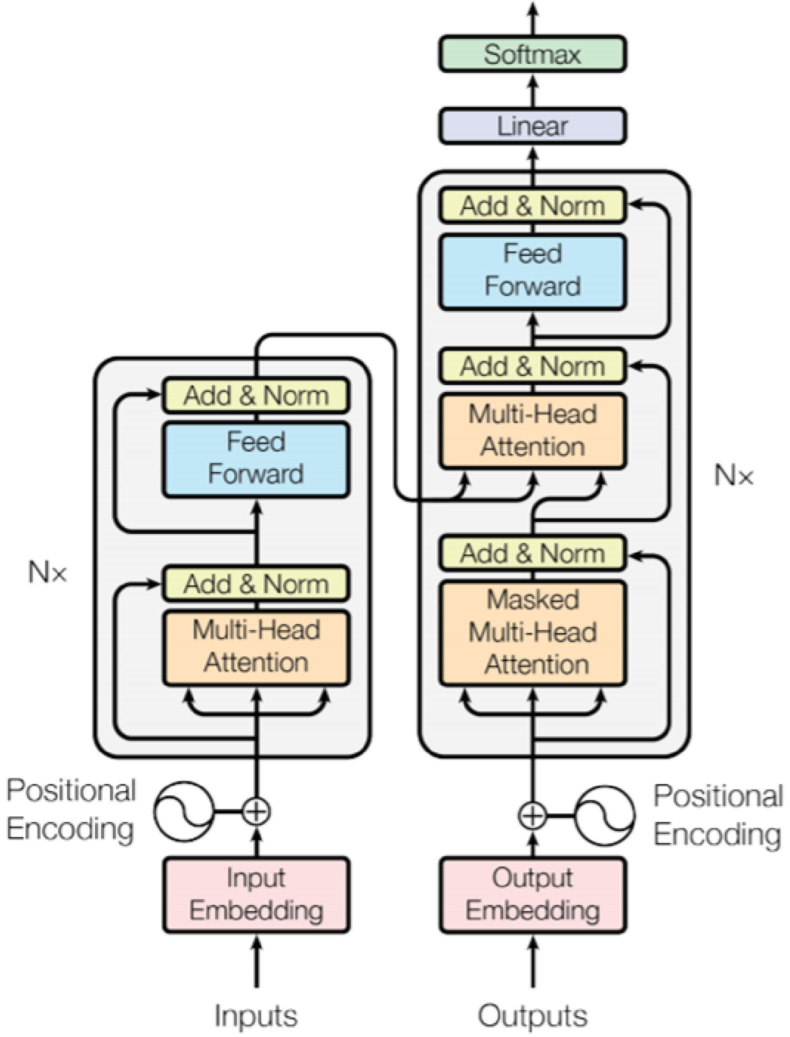
Two key differences from encoder:

- Self-attention only on words generated uptil now, not on whole sentence.
- Additional encoder-decoder attention layer where keys, values come from last encoder layer.



Full architecture with Attention reference





Regularization

Residual dropout: Dropout added to the the output of each sublayer, before it is added to the input of the sublayer and normalized

Label Smoothing: During training label smoothing was employed. This hurts perplexity, as the model learns to be more unsure, but improves accuracy and BLEU score.

Results

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

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**	$2.3 \cdot 10^{19}$	

Results: Parameter Analysis

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

Results: Constituency Parsing

Table 4: The Transformer generalizes well to English constituency parsing (Results are on Section 23 of WSJ)

Parser	Training	WSJ 23 F1
Vinyals & Kaiser el al. (2014) [37]	WSJ only, discriminative	88.3
Petrov et al. (2006) [29]	WSJ only, discriminative	90.4
Zhu et al. (2013) [40]	WSJ only, discriminative	90.4
Dyer et al. (2016) [8]	WSJ only, discriminative	91.7
Transformer (4 layers)	WSJ only, discriminative	91.3
Zhu et al. (2013) [40]	semi-supervised	91.3
Huang & Harper (2009) [14]	semi-supervised	91.3
McClosky et al. (2006) [26]	semi-supervised	92.1
Vinyals & Kaiser el al. (2014) [37]	semi-supervised	92.1
Transformer (4 layers)	semi-supervised	92.7
Luong et al. (2015) [23]	multi-task	93.0
Dyer et al. (2016) [8]	generative	93.3

Continuations and SOTA for Machine Translation

Scaling Neural Machine Translation (Ott et.al. 2018)

model	# gpu	bsz	cumul	BLEU	updates	tkn/sec	time	speedup
Vaswani et al. (2017)	8×P100	25k	1	26.4	300k	~25k	~5,000	–
Our reimplementation	8×V100	25k	1	26.4	192k	54k	1,429	reference
+ `16-bit`	8	25k	1	26.7	193k	143k	495	2.9x
+ `cumul`	8	402k	16	26.7	13.7k	195k	447	3.2x
+ `2x lr`	8	402k	16	26.5	9.6k	196k	311	4.6x
+ `5k tkn/gpu`	8	365k	10	26.5	10.3k	202k	294	4.9x
16 nodes (from `+2x lr`)	128	402k	1	26.5	9.5k	1.53M	37	38.6x
+ `overlap comm+bwd`	128	402k	1	26.5	9.7k	1.82M	32	44.7x

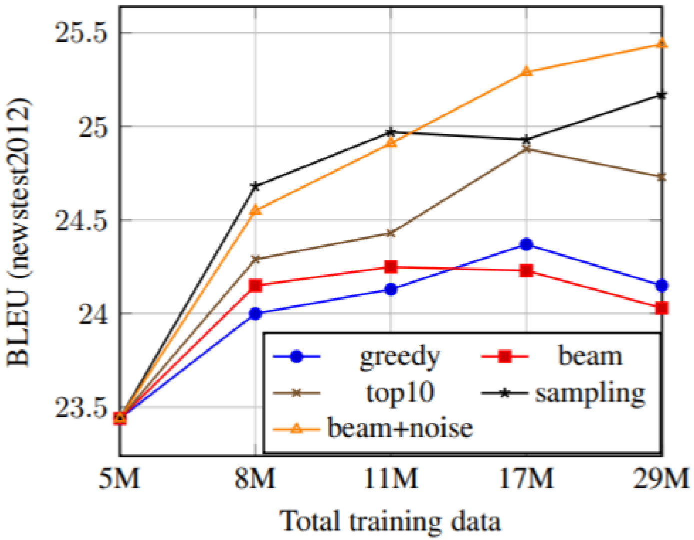
Table 1: Training time (min) for reduced precision (`16-bit`), cumulating gradients over multiple backwards (`cumul`), increasing learning rate (`2x lr`) and computing each forward/backward with more data due to memory savings (`5k tkn/gpu`). Average time (excl. validation and saving models) over 3

Understanding Back-translation at Scale (Edunov et.al. 2018)

This paper augments parallel data corpus with noisy back-translations of monolingual corpora. State of the art for English-German.

Training done on 4.5M bitext and 262M monolingual sentences.

	En–De	En–Fr
a. Gehring et al. (2017)	25.2	40.5
b. Vaswani et al. (2017)	28.4	41.0
c. Ahmed et al. (2017)	28.9	41.4
d. Shaw et al. (2018)	29.2	41.5
DeepL	33.3	**45.9**
Our result	**35.0**	45.6
detok. sacreBLEU[3]	*33.8*	*43.8*



BPE-Dropout: Simple and Effective Subword Regularization (Provilkov et. al. 2019)

This paper adds dropout to Byte-Pair Encoding. State of the art or matching it for syllabic language translation like English-Vietnamese, English-Chinese.

	BPE	Kudo (2018)	*BPE-dropout*
IWSLT15			
En-Vi	31.78	32.43	**33.27**
Vi-En	30.83	32.36	**32.99**
En-Zh	21.07	**23.15**	**23.27**
Zh-En	18.29	21.10	**21.45**
IWSLT17			
En-Fr	39.37	39.45	**40.02**
Fr-En	38.18	38.88	**39.39**
En-Ar	13.89	14.43	**15.05**
Ar-En	31.90	32.80	**33.72**

Multi-agent Learning for Neural Machine Translation (Bi et. al. EMNLP 2019)

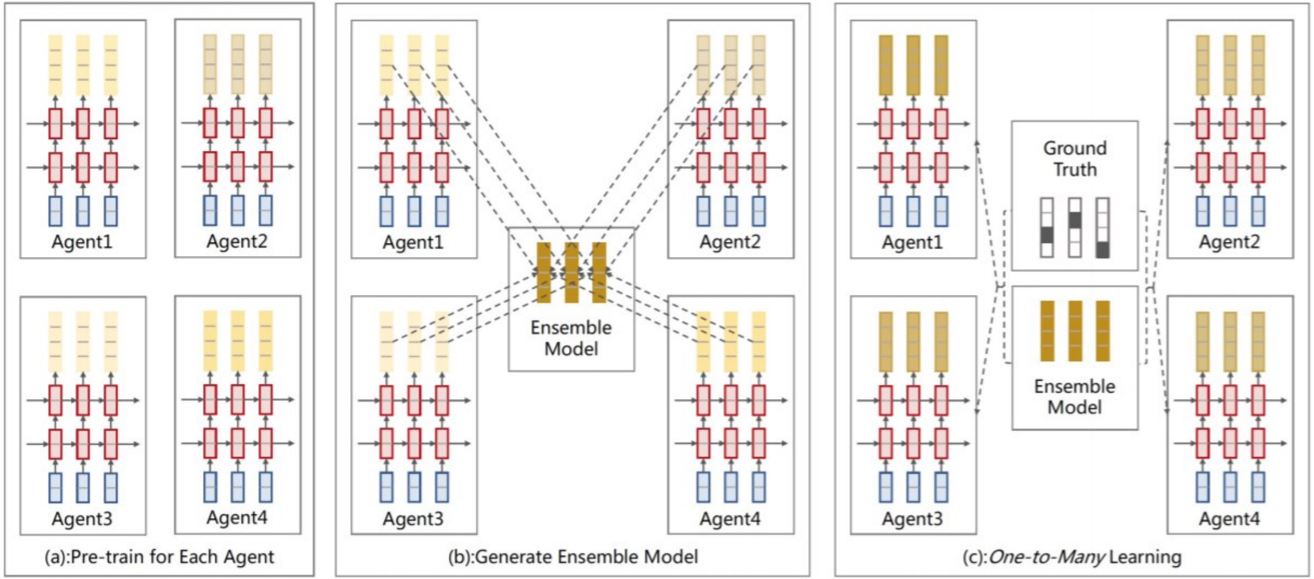


Figure 2: In this example, four agents decode the similar sentence with different model capacity. (a): At first, each agent is pre-trained to generate the translation independently. (b) The ensemble model is generated by the average prediction from each agent. (c): The *One-to-Many* learning distills the knowledge from the ensemble model to each agent as necessary. The performance of each agent is improved explicitly in an interactive updating process, through repeating the process (b) and (c).

These 4 agents are different types of transformers: L2R, R2L, 30-layer encoder, relative position attention

Jointly Learning to Align and Translate with Transformer Models (Garg et. al. EMNLP 2019)

Table 4: Results on the align and translate task. Alignment quality is reported in AER, translation quality in BLEU. †baseline (without back-translation) sacreBLEU results were provided in https://github.com/pytorch/fairseq/issues/506#issuecomment-464411433. ‡Difference in AER w.r.t. GIZA++ (BPE-based) is statistically significant (p<0.001)

	AER[%] (Precision[%], Recall[%])			BLEU[%]	
Model	DeEn	EnDe	Symmetrized	DeEn	EnDe
GIZA++ (word-based)	21.7 (85.4, 72.1)	24.0 (85.8, 68.2)	22.2 (93.5, 66.5)	-	-
GIZA++ (BPE-based)	19.0 (89.1, 74.2)	21.3 (86.8, 71.9)	19.6 (93.2, 70.6)	-	-
Layer average baseline	66.8 (32.0, 34.6)	66.5 (32.5, 34.7)	54.8 (94.2, 29.6)	33.1	28.7
Multi-task	31.1 (67.2, 70.7)	32.2 (66.6, 69.1)	25.8 (88.1, 63.8)	33.1	28.5
+ full-context	21.2 (76.9, 80.9)	23.5 (75.0, 78.0)	19.5 (89.5, 72.9)	33.2	28.5
++ GIZA++ supervised	**17.5**‡ (80.5, 84.7)	**19.8**‡ (78.8, 81.7)	**16.4**‡ (89.6, 78.2)	33.1	28.8
Edunov et al. (2018)†	-	-	-	-	29.0

Pros

- Current state-of-the-art in machine translation and text simplification.
- Intuition of model well explained
- Easier learning of long-range dependencies
- Relatively less computation complexity
- In-depth analysis of training parameters

Cons

Huge number of parameters so-

- Very data hungry
- Takes a long time to train, LSTM comparisons in paper are unfair
- No study of memory utilisation

Other issues

- Keeping sentence length limited
- How to ensure multi-head attention has diverse perspectives.

Reformer: The Efficient Transformer

Kitaev et. al. (January 2020, ICLR)

Concerns about the transformer

“Transformer models are also used on increasingly long sequences. Up to 11 thousand tokens of text in a single example were processed in (Liu et al., 2018) … These large-scale long-sequence models yield great results but **strain resources to the point where some argue that this trend is breaking NLP research**”

“Many large Transformer models **can only realistically be trained in large industrial research laboratories** and such models trained with model parallelism cannot even be fine-tuned on a single GPU as their **memory requirements demand a multi-accelerator hardware setup**"

Memory requirement estimate (per layer)

Largest transformer layer ever: 0.5B parameters = 2GB

Activations for 64K tokens for embedding size 1K and batch size 8

= 64K * 1K * 8 = 2GB

Training data used in BERT = 17GB

Why can't we fit everything in one GPU? 32GB GPUs are common today.

Caveats follow ->>>>>

Caveats

1. There are N layers in a transformer, whose activations need to be stored for backpropagation
2. We have been ignoring the feed-forward networks uptil now, whose depth even exceeds the attention mechanism so contributes to significant fraction of memory use.
3. Dot product attention is $O(L^2)$ in space complexity where L is length of text input.

Solutions

1. Reversible layers, first introduced in Gomez et al. (2017), enable storing only a single copy of activations in the whole model, so the N factor disappears.
2. Splitting activations inside feed-forward layers and processing them in chunks saves memory inside feed-forward layers.
3. Approximate attention computation based on locality-sensitive hashing replaces the $O(L^2)$ factor in attention layers with $O(L \log L)$ and so allows operating on long sequences.

Locality Sensitive Hashing

Hypothesis: Attending on all vectors is approximately same as attending to the 32/64 closest vectors to query in key projection space.

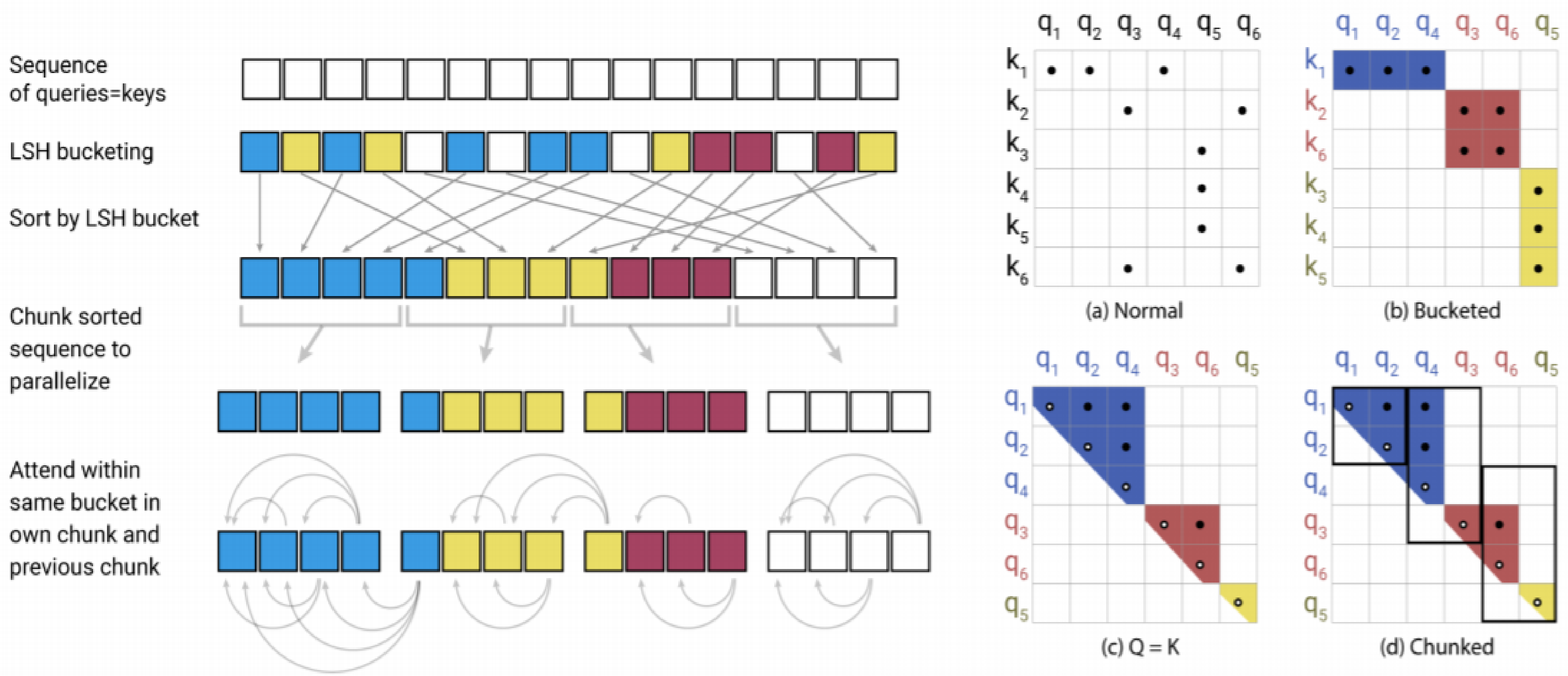
To find such vectors easily we require:

- Key and Query to be in same space
- Locality sensitive hashing i.e. if distance between key and query is less then distance between their hash values is less.

Locality sensitive hashing scheme taken from Andoni et al., 2015

For simplicity, a bucketing scheme chosen: attend on everything in your bucket

Locality sensitive hashing



Locality sensitive hashing

Table 1: Memory and time complexity of attention variants. We write l for length, b for batch size, n_h for the number of heads, n_c for the number of LSH chunks, n_r for the number of hash repetitions.

Attention Type	Memory Complexity	Time Complexity
Scaled Dot-Product	$\max(bn_hld_k, bn_hl^2)$	$\max(bn_hld_k, bn_hl^2)$
Memory-Efficient	$\max(bn_hld_k, bn_hl^2)$	$\max(bn_hld_k, bn_hl^2)$
LSH Attention	$\max(bn_hld_k, bn_hln_r(4l/n_c)^2)$	$\max(bn_hld_k, bn_hn_rl(4l/n_c)^2)$

We have reduced the second term in the max(...) but the first term still remains a challenge.

Plumbing the depths

the activations before each layer are already of the size $b \cdot l \cdot d_{model}$, so the memory use of the whole model with n_l layers is at least $b \cdot l \cdot d_{model} \cdot n_l$. Even worse: inside the feed-forward layers of Transformer this goes up to $b \cdot l \cdot d_{ff} \cdot n_l$. In a big Transformer it is usual to set $d_{ff} = 4K$ and $n_l = 16$ so with $l = 64K$ this again would use an impractical $16GB$ of memory

For reducing attention activations: RevNets

For reducing feed forward activations: Chunking

RevNets

Reversible residual layers were introduced in Gomez et. al. 2017

Idea: Activations of previous layer can be recovered from activations of subsequent layers, using model parameters.

Normal residual layer: y = x + F(x)

Reversible laver:

$$y_1 = x_1 + F(x_2)$$
$$x_2 = y_2 - G(y_1)$$

$$y_2 = x_2 + G(y_1)$$
$$x_1 = y_1 - F(x_2)$$

So, for transformer:

$$Y_1 = X_1 + \mathrm{Attention}(X_2)$$

$$Y_2 = X_2 + \mathrm{FeedForward}(Y_1)$$

Chunking

$$Y_2 = \left[Y_2^{(1)}; \ldots; Y_2^{(c)}\right] = \left[X_2^{(1)} + \mathrm{FeedForward}(Y_1^{(1)}); \ldots; X_2^{(c)} + \mathrm{FeedForward}(Y_1^{(c)})\right]$$

Operations done a chunk at a time:

- Forward pass of Feed-forward network
- Reversing the activations during backpropagation
- For large vocabularies, chunk the log probabilities

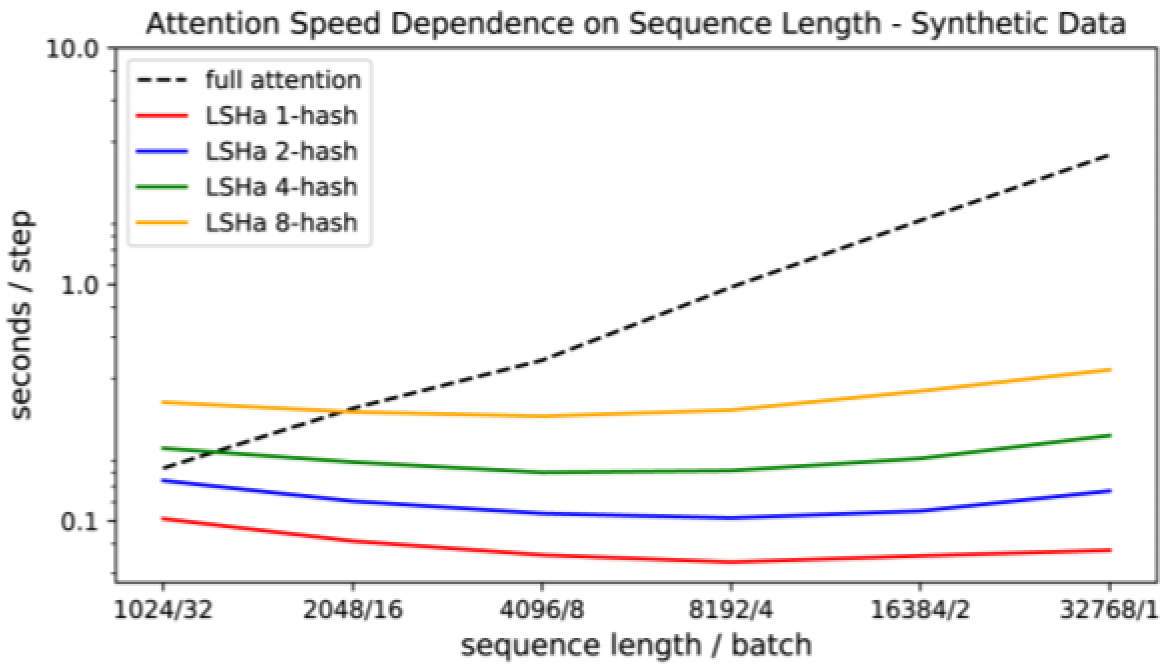
CPU data swaps and conclusion

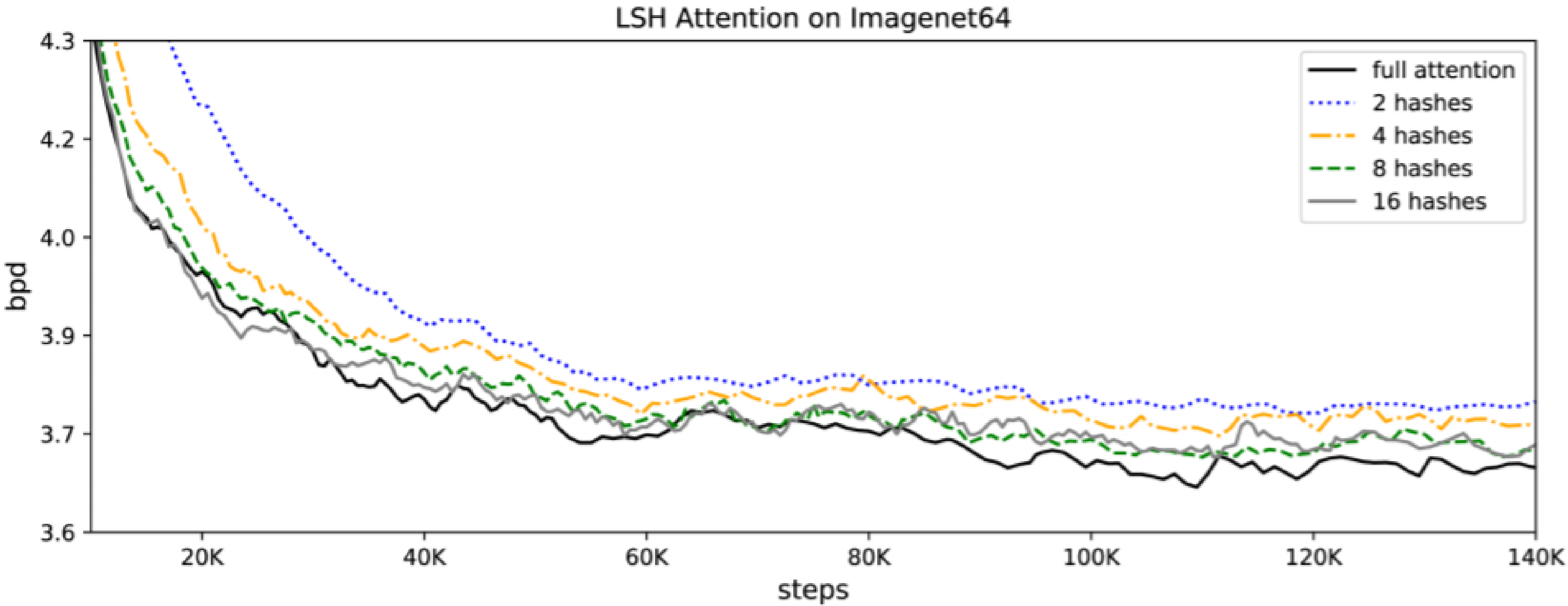
Layer parameters being computed swapped from CPU to GPU and vice versa

Hypothesis: Large batch size and length of input in Reformer so not so inefficient to do such data transfers

Model Type	Memory Complexity	Time Complexity
Transformer	$\max(bld_{ff}, bn_h l^2)n_l$	$(bld_{ff} + bn_h l^2)n_l$
Reversible Transformer	$\max(bld_{ff}, bn_h l^2)$	$(bn_h ld_{ff} + bn_h l^2)n_l$
Chunked Reversible Transformer	$\max(bld_{model}, bn_h l^2)$	$(bn_h ld_{ff} + bn_h l^2)n_l$
LSH Transformer	$\max(bld_{ff}, bn_h ln_r c)n_l$	$(bld_{ff} + bn_h n_r lc)n_l$
Reformer	$\max(bld_{model}, bn_h ln_r c)$	$(bld_{ff} + bn_h n_r lc)n_l$

Experiments





LSH Attention on Imagenet64
4.3
4.2
4.0
3.9
3.7
3.6
bpd
20K
40K
60K
80K
100K
120K
140K
steps
full attention
2 hashes
4 hashes
8 hashes
16 hashes