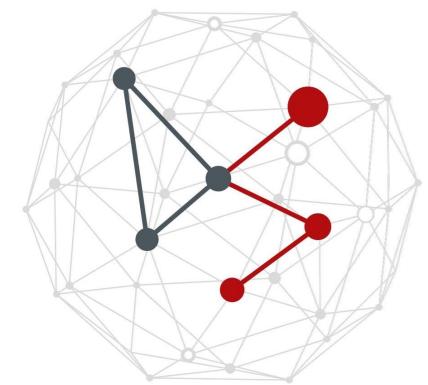


ADVANCED NEURAL NETWORK DESIGNS: SELF-ATTENTION & TRANSFORMERS

Michele Rossi

michele.rossi@unipd.it

Dept. of Information Engineering
University of Padova, IT



Outline (1/2)

- Reference papers
- General picture
 - RNN vs transformers
 - The transformer building blocks (high level)
 - Encoder/decoder and full picture
- Block-by-block analysis
 - Word embedding
 - Self-attention
 - Multihead self-attention
 - Positional encoding
 - Skip connections and normalization
 - Decoder output: dense layer + softmax
 - Masked self-attention @decoder



Outline (2/2)

- Selected applications & trends
 - Learning to attend and diagnose
 - Large language Models
 - Automatic speech translation
 - Automatics code completion
 - Code generation from descriptive text
 - ...
- Concluding remarks



[Vaswani2017]

61799 cit, Jan 2023

Attention Is All You Need

Conference on Neural Information Processing Systems, NeurIPS 2017

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

Lukasz Kaiser*

Google Brain

lukaszkaiser@google.com

Illia Polosukhin* ‡

illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

Transformers in Time-series Analysis: A Tutorial

Sabeen Ahmed¹, Ian E. Nielsen¹, Aakash Tripathi¹

Shamoon Siddiqui¹, Ghulam Rasool¹, Ravi P. Ramachandran¹

¹ Rowan University

{ahmedsa, nielseni6, tripat67, siddiq76, rasool, ravi}@rowan.edu

Abstract

Transformer architecture has widespread applications, particularly in Natural Language Processing and computer vision. Recently Transformers have been employed in various aspects of time-series analysis. This tutorial provides an overview of the Transformer architecture, its applications, and a collection of examples from recent research papers in time-series analysis. We delve into an explanation of the core components of the Transformer, including the self-attention mechanism, positional encoding, multi-head, and encoder/decoder. Several enhancements to the initial, Transformer architecture are highlighted to tackle time-series tasks. The tutorial also provides best practices and techniques to overcome the challenge of effectively training Transformers for time-series analysis.

1 Introduction and Background

Transformers belong to a class of machine learning models that use self-attention or the scaled dot-product operation as their primary learning mechanism. Transformers were initially proposed for neural machine translation - one of the most challenging natural language processing (NLP) tasks [64]. Recently, transformers have been successfully employed to tackle various problems in machine learning and achieve state-of-the-art performance. Apart from classical NLP tasks, examples from other areas include image classification, object detection & segmentation, image & language generation, sequential decision making in reinforcement learning, multi-modal (text, speech, and image) data processing, and analysis of tabular and time-series data. This tutorial paper focuses on time-series analysis using Transformers.

The time-series data consist of ordered samples, observations, or features recorded sequentially over time. Time-series datasets often arise naturally in many real-world applications where data is recorded over a fixed sampling interval. Examples include stock prices, digitized speech signals, traffic measurements, sensor data for weather patterns, biomedical measurements, and various kinds of population data recorded over time. The time series analysis may include processing the numeric data for multiple tasks, including

Transformers in Time-series Analysis: A Tutorial

Sabeen Ahmed¹, Ian E. Nielsen¹, Aakash Tripathi¹
Shamoon Siddiqui¹, Ghulam Rasool¹, Ravi P. Ramachandran¹

¹ Rowan University

{ahmedsa, nielseni6, tripat67, siddiq76, rasool, ravi}@rowan.edu

[Sabeem2022] arXiv:2205.01138v1 [cs.LG] 28 Apr 2022

Transformer architecture has widespread applications, particularly in Natural Language Processing and computer vision. Recently Transformers have been employed in various aspects of time-series analysis. This tutorial provides an overview of the Transformer architecture, its applications, and a collection of examples from recent research papers in time-series analysis. We delve into an explanation of the core components of the Transformer, including the self-attention mechanism, positional encoding, multi-head, and encoder/decoder. Several enhancements to the initial, Transformer architecture are highlighted to tackle time-series tasks. The tutorial also provides best practices and techniques to overcome the challenge of effectively training Transformers for time-series analysis.

1 Introduction and Background

Transformers belong to a class of machine learning models that use self-attention or the scaled dot-product operation as their primary learning mechanism. Transformers were initially proposed for neural machine translation - one of the most challenging natural language processing (NLP) tasks [64]. Recently, transformers have been successfully employed to tackle various problems in machine learning and achieve state-of-the-art performance. Apart from classical NLP tasks, examples from other areas include image classification, object detection & segmentation, image & language generation, sequential decision making in reinforcement learning, multi-modal (text, speech, and image) data processing, and analysis of tabular and time-series data. This tutorial paper focuses on time-series analysis using Transformers.

The time-series data consist of ordered samples, observations, or features recorded sequentially over time. Time-series datasets often arise naturally in many real-world applications where data is recorded over a fixed sampling interval. Examples include stock prices, digitized speech signals, traffic measurements, sensor data for weather patterns, biomedical measurements, and various kinds of population data recorded over time. The time series analysis may include processing the numeric data for multiple tasks, including

Acknowledgments

Gerard I. Gállego, Universitat Politècnica de Catalunya, Barcelona, Spain – “The Transformer Revolution,” CTTC, November 2022.

[Alammar2018] Jay Alammar, “[The Illustrated Transformer](#),” (blog) June 27, 2018.

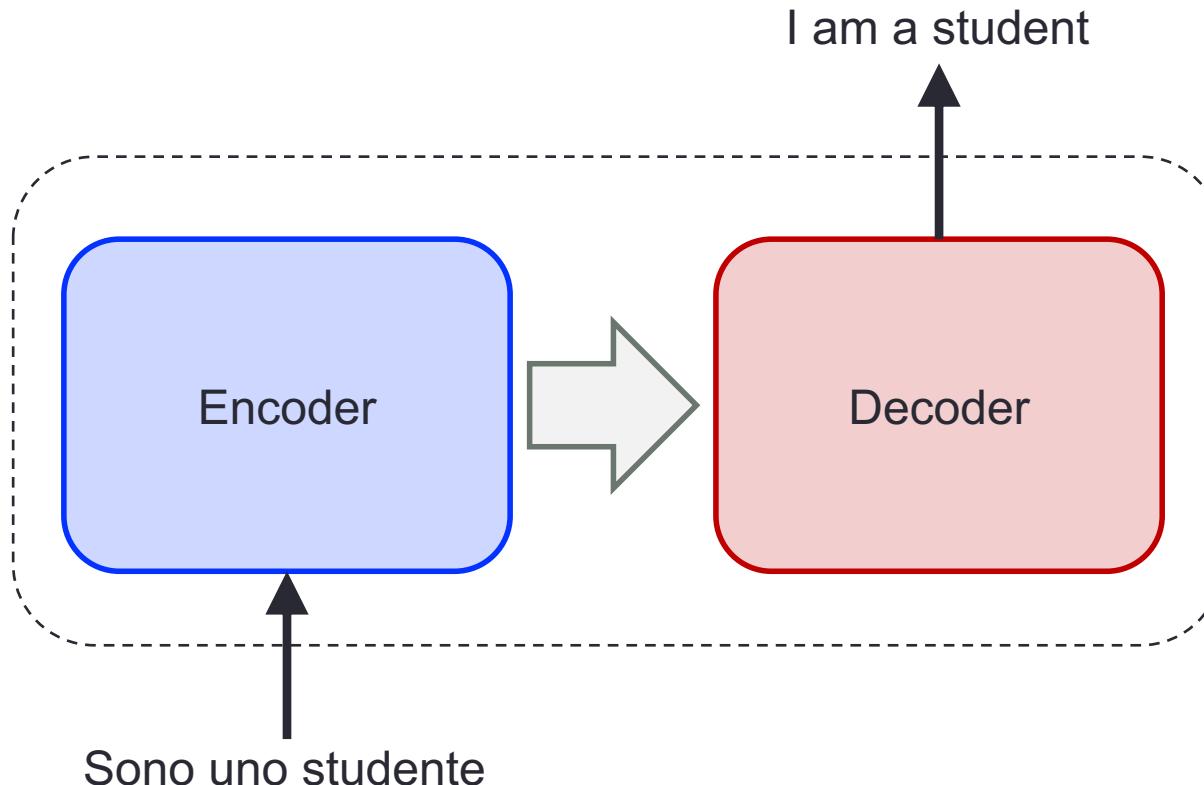
Transformer – what it is

- Sequence-to-sequence architecture
- Implemented via an encoder-decoder model
- Designed based on FFNNs (*no recursion, no memory cells*)
- Takes a sequence S_I of data items as input
- Returns a sequence S_O of data items as output
- Sequences do not necessarily have the same length
- S_I is then compressed into a fixed-length code
- The code is used by the decoder to generate S_O
- Original transformer [Vaswani2017]
 - Proposed for language translation
 - Takes a sequence of words from one language
 - Returns the corresponding sentence in another language

RNN vs transformers (high level)

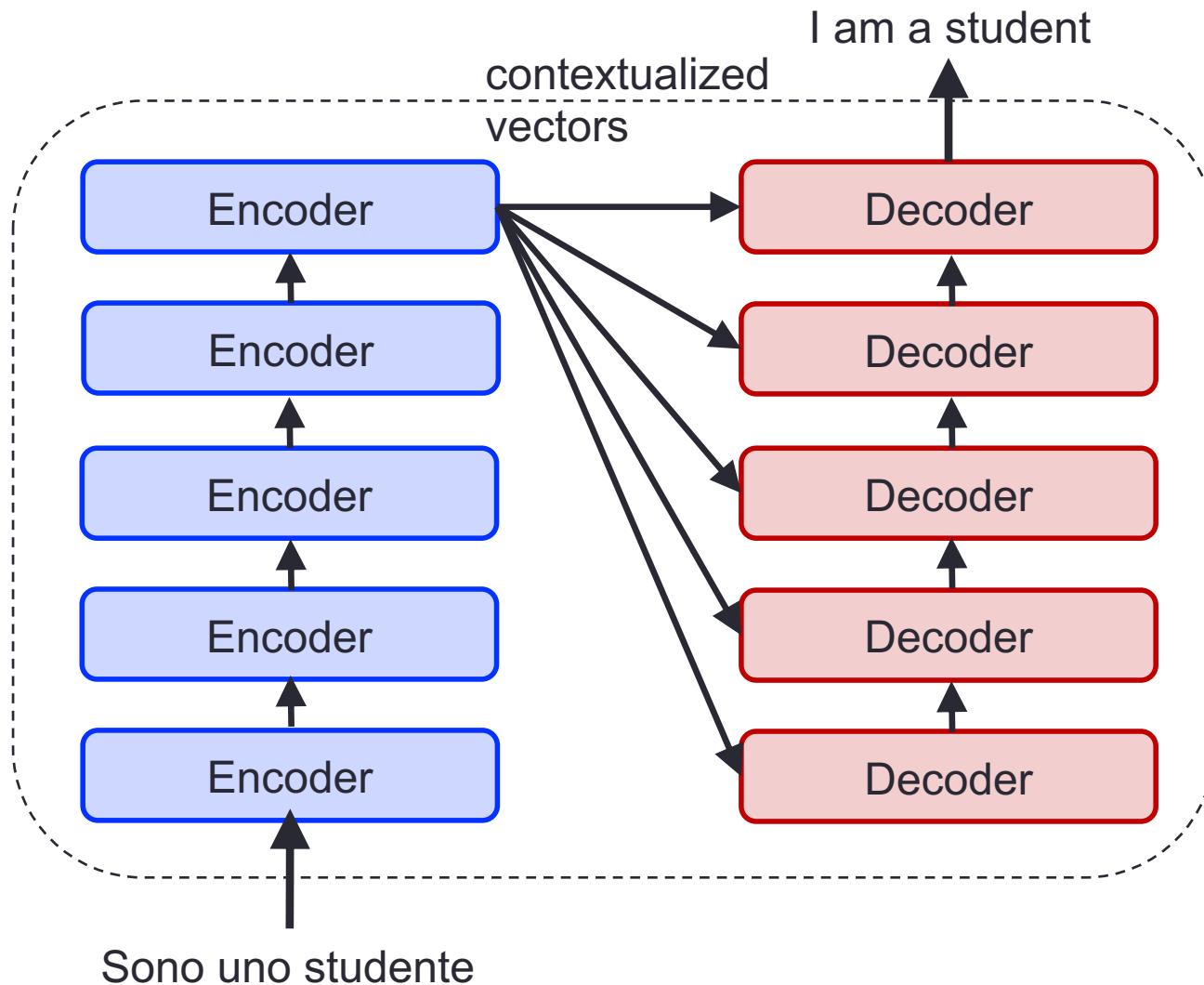
- RNNs
 - Implicitly capture position of words/items in a sequence by **passing one item at a time to the RNN** – past memory is used to process the current input
 - Time correlation is modeled and retained **by the internal cell memory**
- Transformers
 - Are FFNNs
 - They **do not use RNN memory cells**
 - The sentence/data sequence **is fed all at once** to the NN
- ADVANTAGES
 - Getting rid of RNN cells (dense, slow to train, memory expensive)
 - Transformers better exploit the parallelism of modern GPUs

The transformer (very high-level)



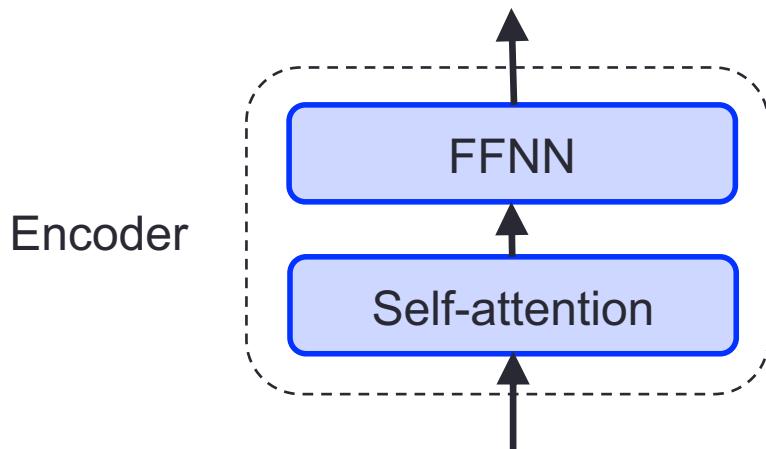
Encoder and decoder are a stack of components

The transformer (high-level)



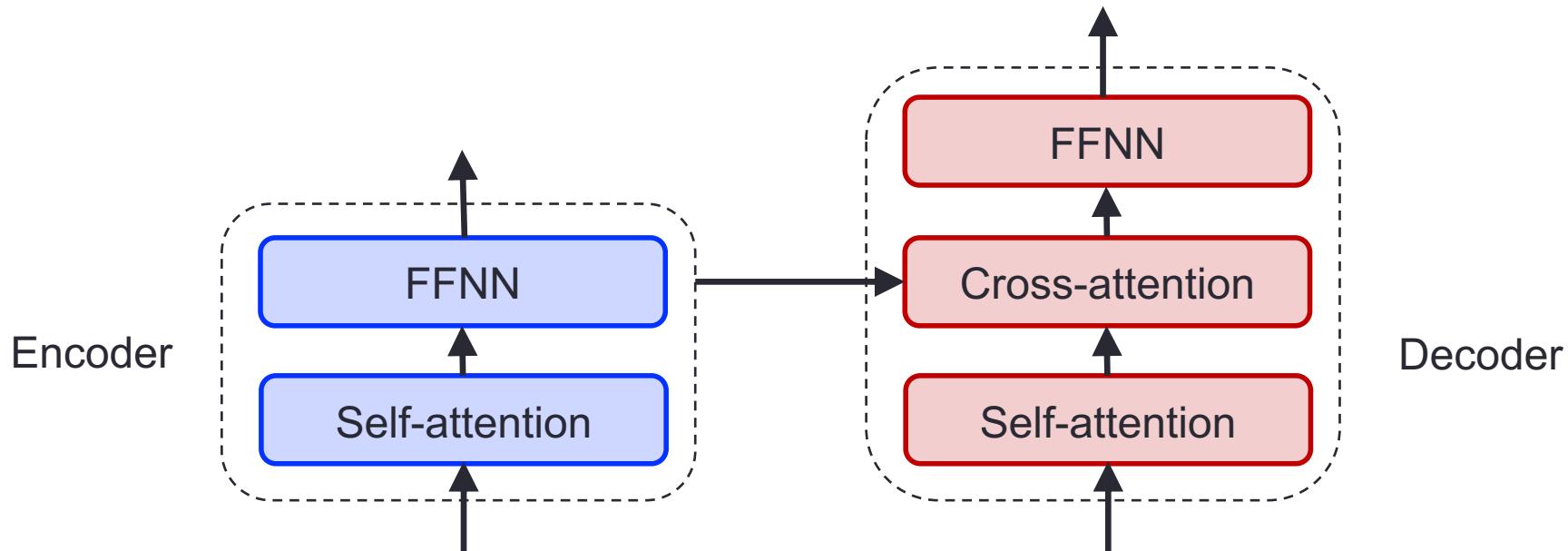
Encoder

- The encoders are **all identical**
 - They **do not share weights** among them
 - Each one is composed of 2 sub-layers
 - **Bottom:** **Self-attention** (**linear**, correlation)
 - **Top:** **Feed forward neural networks** (**non-linear**) – it is *independently applied* to each vector from the underlying attention block (see shortly)

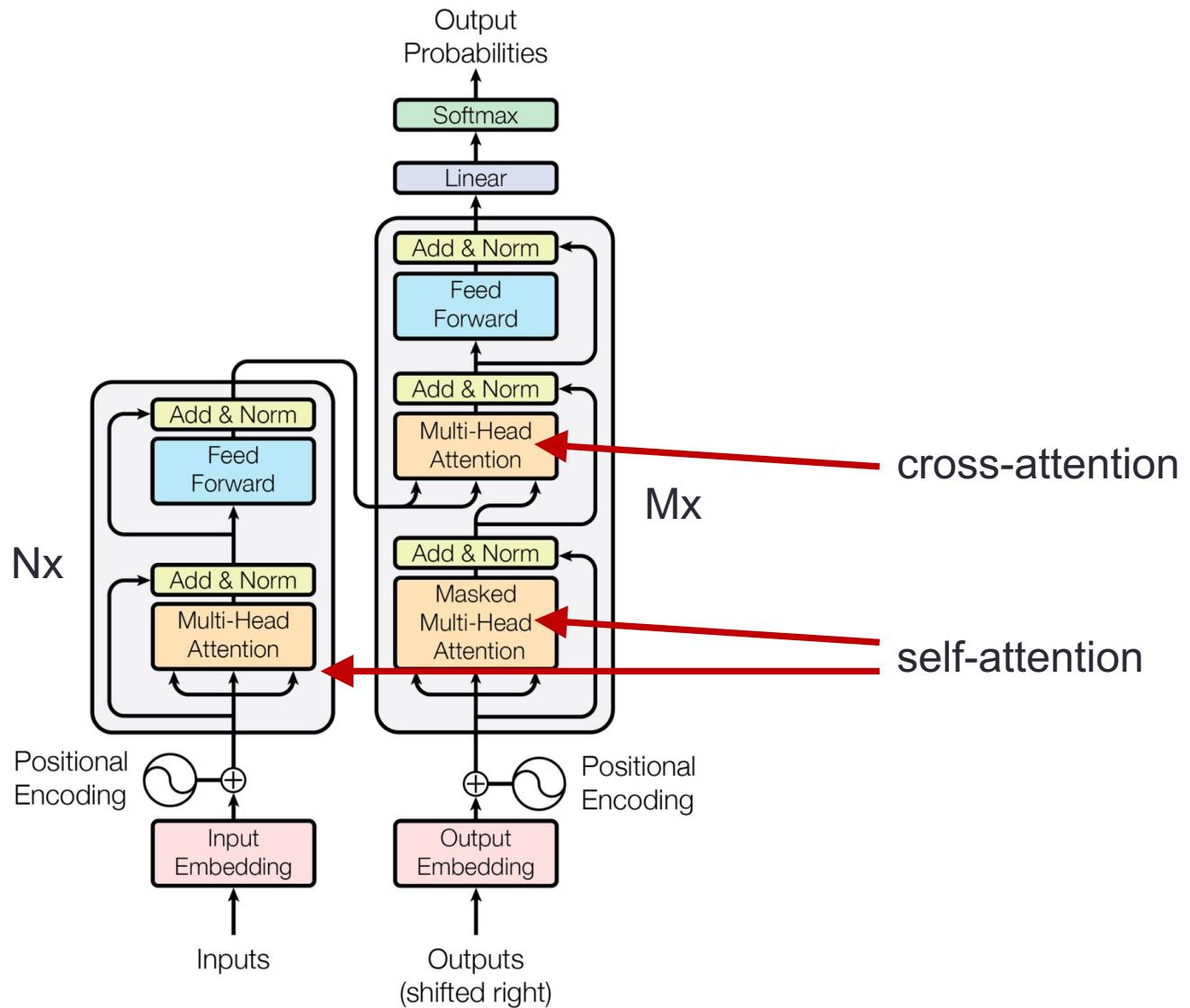


Decoder

- The decoder additionally uses a **cross-attention layer**
- To exploit the **contextualized info** learned by the encoder



The full picture [Vaswani2017]



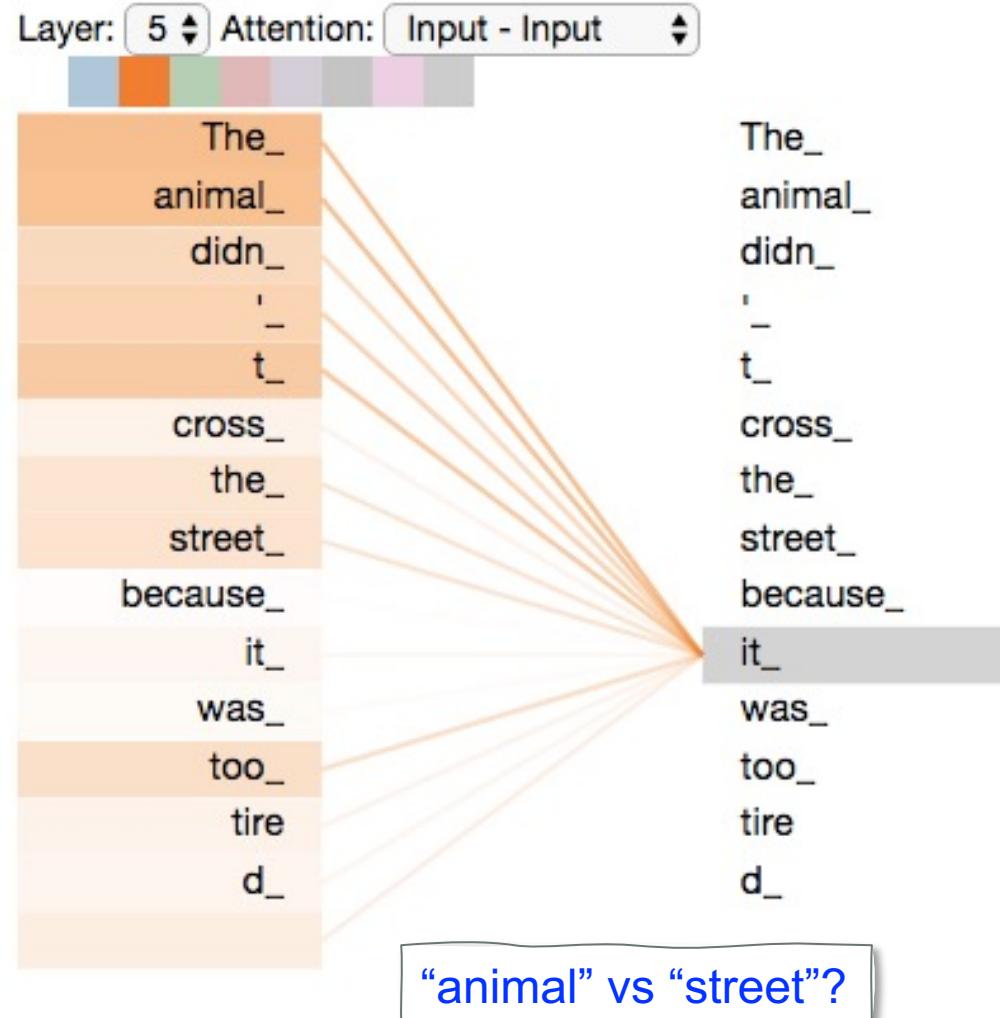
BLOCK-BY BLOCK ANALYSIS

Key ingredients

- Self attention
 - Queries, keys and values are the input data themselves
 - Obtained via dot products (basically, to automatically discover correlations within the input sequence)
- Positional encoding
 - Needed as data sequence is fed *all at once*
 - Desiderata: 1) unique encoding value for each time step in a sequence of items, 2) consistent distance across time steps for seqs of various length 3) encoding results must be independent of the length of the sequence, 4) the encoding must be deterministic
 - Allows the model to learn the data temporal correlation structure
- New things to look at wrt RNNs
 - Self-attention, multi-head attention, positional encoding, masked multi-head attention

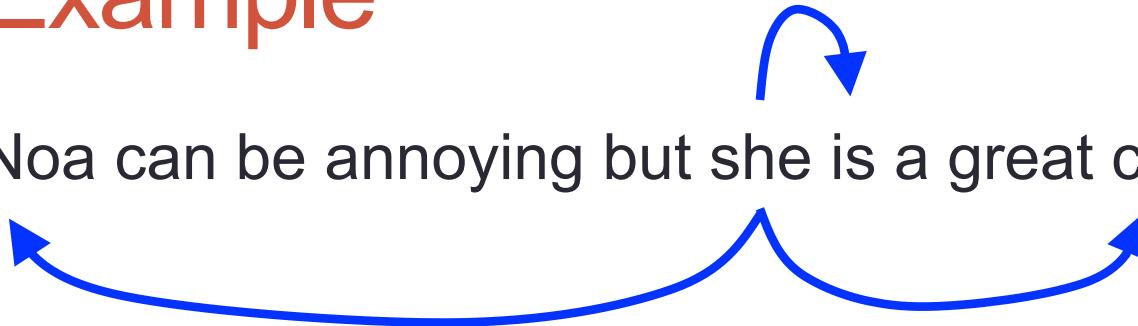
Self-attention (intuition)

- Let's take the word "it" as the **target word** in a sentence
- A relation is built (correlation) between **each target word** and **any other word** in the same sentence
- This relation is stored in the form of a **probability distribution** (softmax)
- A single word **attends to** multiple words in the same sentence (sequence)



Example

Noa can be annoying but she is a great cat



The word “she” in this sentence

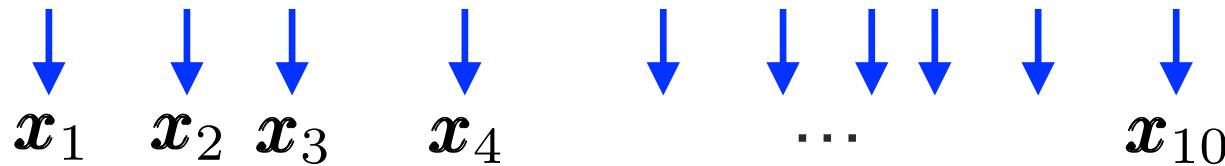
- Is semantically connected with the subject “Noa”
- Is connected with the noun “cat”
- And also with the state of being verb “is”

We would like to re-express words

- Contextualizing them wrt other words data items
- A Proximity based metrix cannot be the way

Self-attention

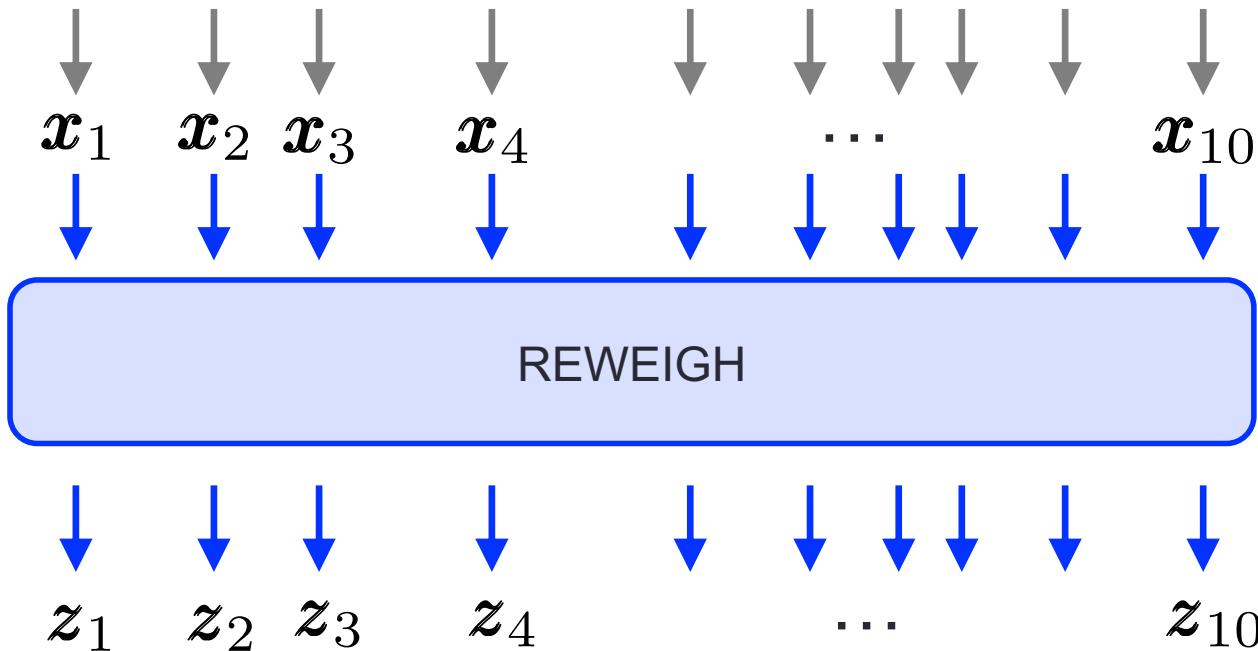
Noa can be annoying but she is a great cat



STEP 1: from words to vectors
→ embeddings

Self-attention

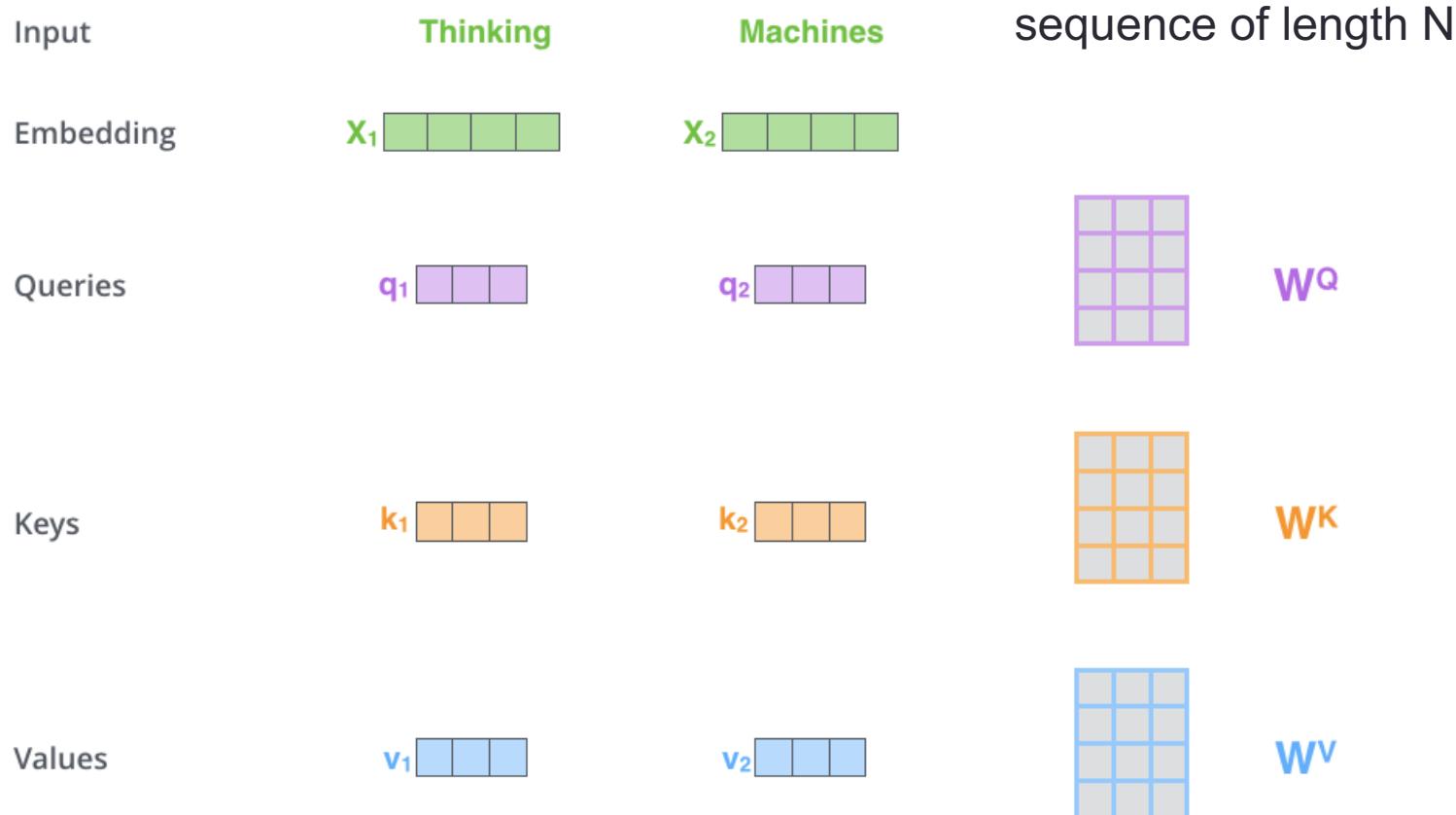
Noa can be annoying but she is a great cat



STEP 2: reweigh words

- Reweighting process is done via **self-attention**
- The new vectors/words are **contextualized representations**

Self-attention: queries, keys, values



Multiply the x_i vectors by 3 different matrices W^Q , W^K , W^V to obtain **Query**, **Key** and **Value** vectors

Self-attention: queries, keys, values

$$\begin{matrix} \mathbf{x} \\ \mathbf{x}_1 \\ \mathbf{x}_2 \end{matrix} \times \begin{matrix} \mathbf{W}^Q \\ \mathbf{W}^Q \end{matrix} = \mathbf{Q}$$

in matrix form

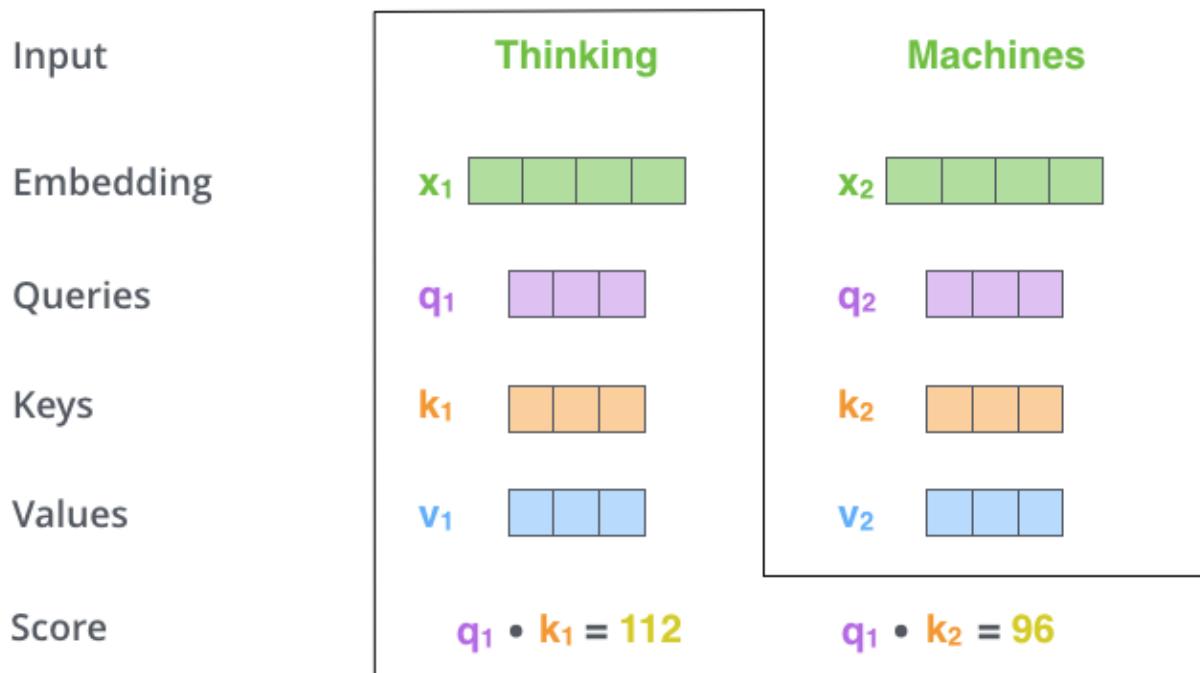
$$\begin{matrix} \mathbf{x} \\ \mathbf{x}_1 \\ \mathbf{x}_2 \end{matrix} \times \begin{matrix} \mathbf{W}^K \\ \mathbf{W}^K \end{matrix} = \mathbf{K}$$

Note: for this slideset we use row vectors – the same calculations apply verbatim with column vectors

$$\begin{matrix} \mathbf{x} \\ \mathbf{x}_1 \\ \mathbf{x}_2 \end{matrix} \times \begin{matrix} \mathbf{W}^V \\ \mathbf{W}^V \end{matrix} = \mathbf{V}$$

Queries, Keys and Values: are 3 linearly weighed vectors of the input. The weights in these matrices **are to be learned**

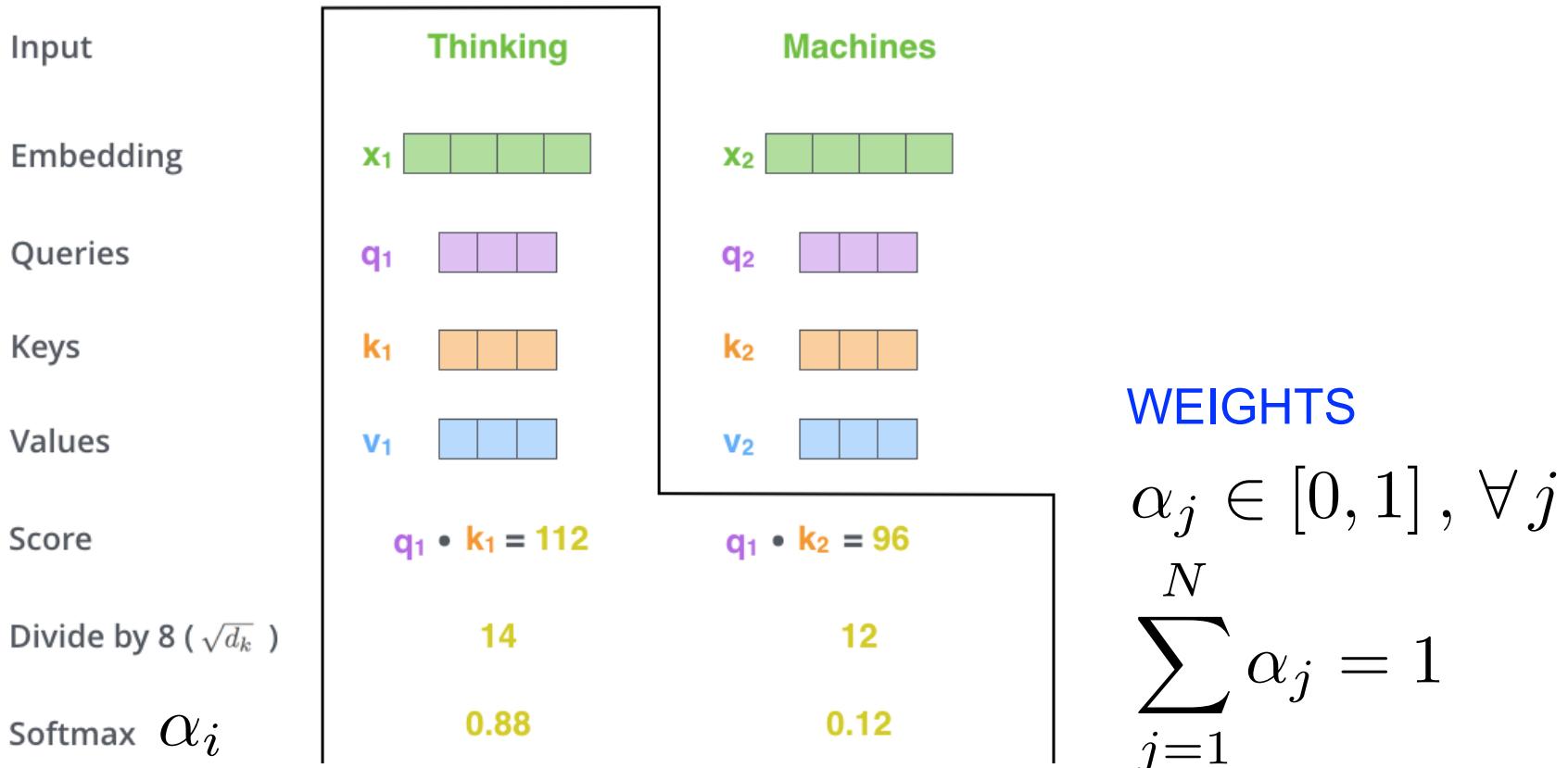
Self-attention: scores



For every x_i in the sequence: compute the **dot product** of q_i and all the N keys k_j , this returns the so called **scores**

- N scores (one for each word)
- It is a **correlation measure**

Self-attention: normalized scores



Normalize (see later), this returns a vector of N real values (one for each element). Use softmax to obtain weights in [0,1]

Input

Embedding

Queries

Keys

Values

Score

Divide by 8 ($\sqrt{d_k}$)

Softmax α_i

Softmax

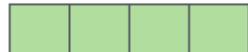
X

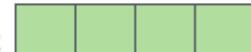
Value

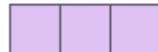
Sum

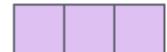
Thinking

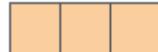
Machines

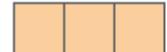
x_1 

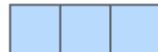
x_2 

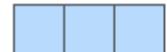
q_1 

q_2 

k_1 

k_2 

v_1 

v_2 

$$q_1 \cdot k_1 = 112$$

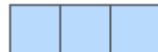
$$q_1 \cdot k_2 = 96$$

14

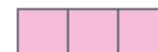
12

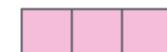
0.88

0.12

v_1 

v_2 

z_1 

z_2 

Reweigh

REWEIGH

$$z_i = \sum_{j=1}^N \alpha_j v_j$$

Input

Embedding

Queries

Keys

Values

Score

Divide by 8 ($\sqrt{d_k}$)

Softmax α_i

Softmax

X

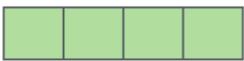
Value

Sum

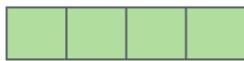
Thinking

Machines

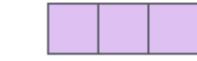
x_1



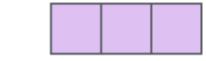
x_2



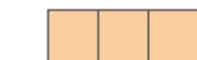
q_1



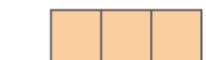
q_2



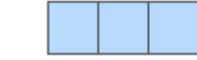
k_1



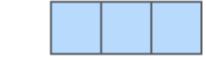
k_2



v_1



v_2



$$q_1 \cdot k_1 = 112$$

$$q_1 \cdot k_2 = 96$$

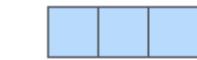
14

12

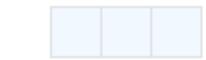
0.88

0.12

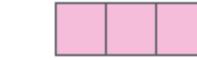
v_1



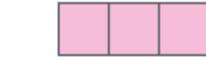
v_2



z_1



z_2



Reweigh

REPEAT FOR ALL
N WORDS x_i

In matrix form

$$\text{softmax} \left(\frac{\begin{matrix} Q & \times & K^T \\ \hline & \sqrt{d_k} & \end{matrix}}{V} \right) = Z$$

The diagram illustrates the computation of the attention matrix Z from input vectors Q , K^T , and V . The input vector Q is represented by a purple 3x3 grid. It is multiplied by the transpose of the input vector K^T , represented by an orange 3x3 grid. The result is then divided by the square root of the dimension d_k . Finally, the softmax function is applied to produce the output matrix Z , represented by a pink 3x3 grid.

As for the normalization inside the **softmax**
• “Leap of faith” for now

Self-attention block – recap

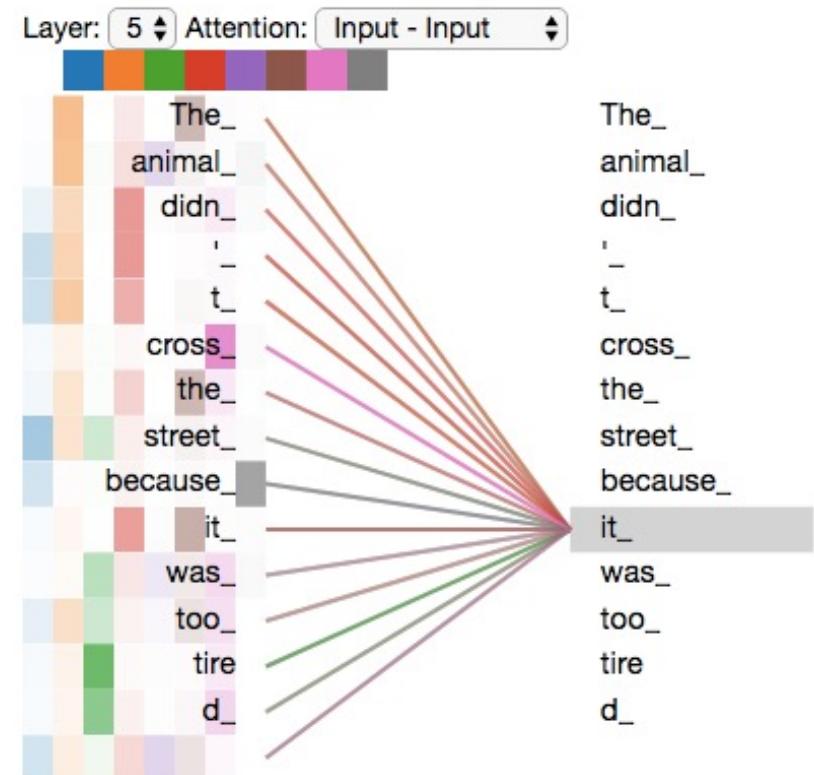
- So, attention captures **the degree of dependence** ("correlation") among the data items (e.g., words) in a data sequence (e.g., sentence)
- This dependence can be related to **semantic meaning in the language**, similarity in an embedded/feature space, proximity of points in the data space
- It is captured by
 - **Computing a distribution of weights** (probabilities) expressing how much each item attends to (relates to) each other item in the same sequence, and then,
 - **Reweighting the data items** themselves using the weights (the probabilities)

Aspects worth of note

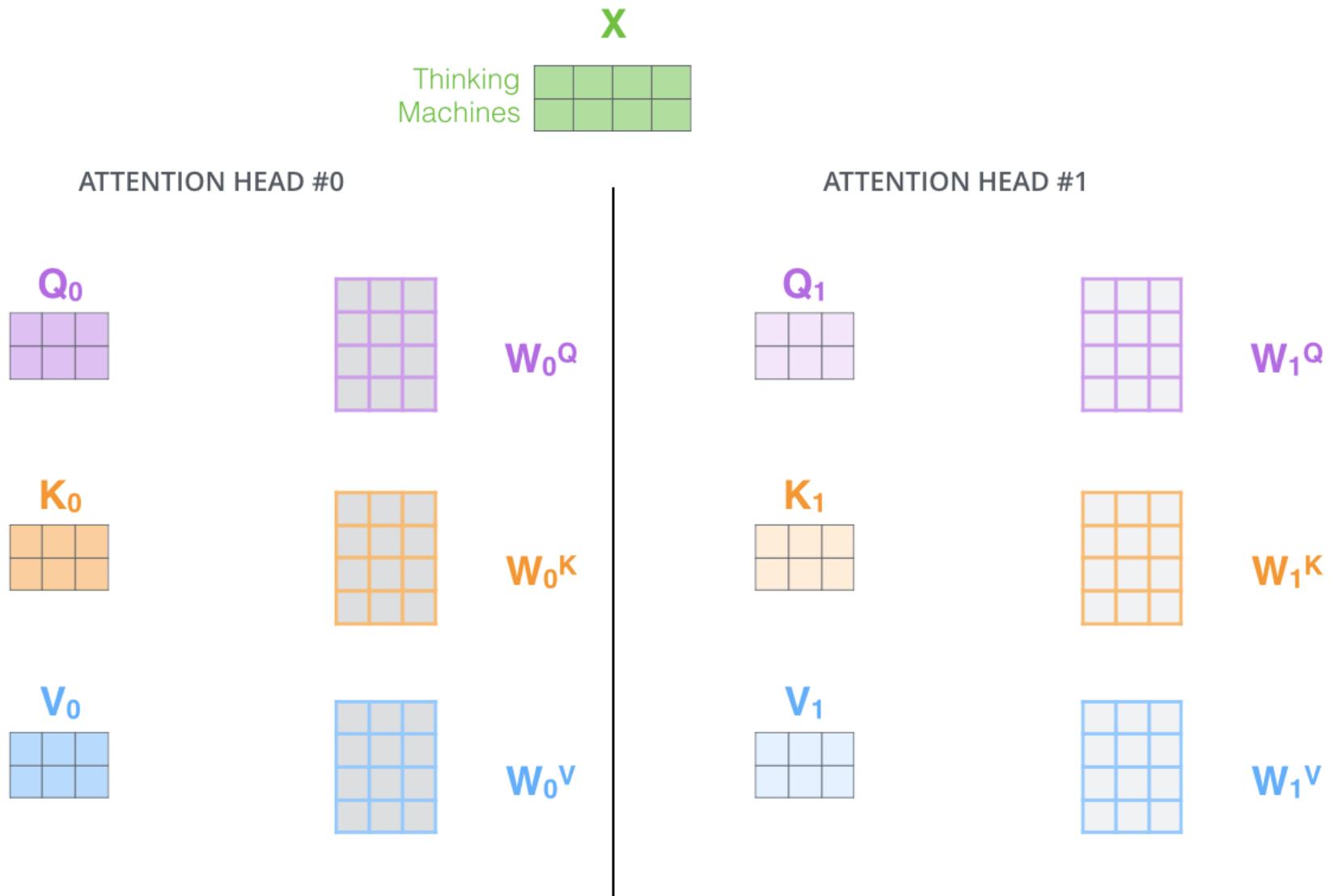
- Order has no influence on the calculation of the z_i
- Proximity has no influence in the calculation of the z_i
- The approach does not depend on the sequence length N
- Complexity scales as $O(N^2)$
 - For each data item $x_i \rightarrow$ compute N weights to reexpress it into z_i
 - Repeat for N data items
 - This is a drawback for long sequences (price to pay)

Multihead attention

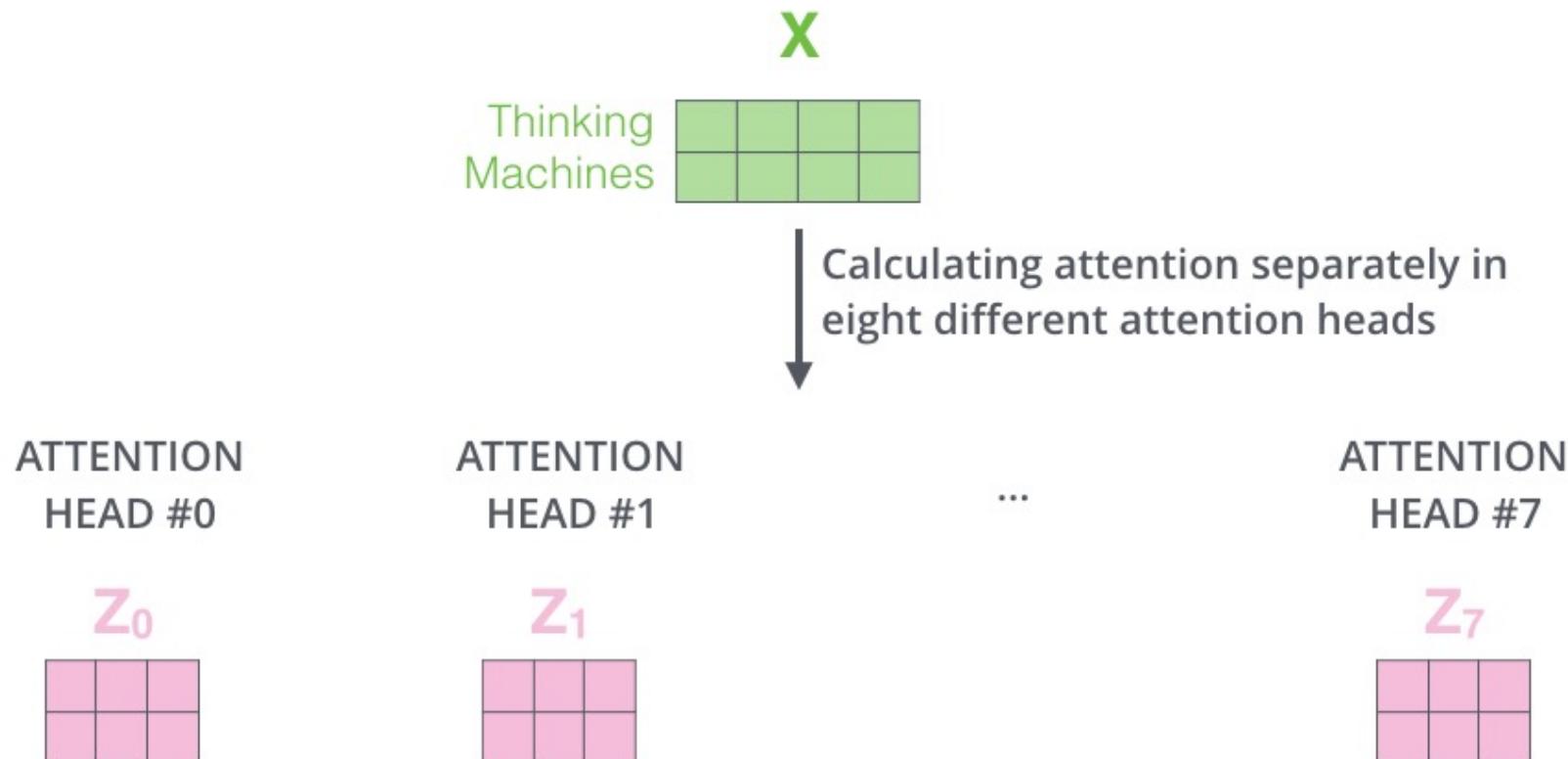
- Merely: using multiple self-attention in parallel
- Same idea as adding multiple CNN filters for a convolutional layer
- Each head may be (possibly) capturing different correlation properties of the input – at different semantic levels



Multihead attention (2 heads)

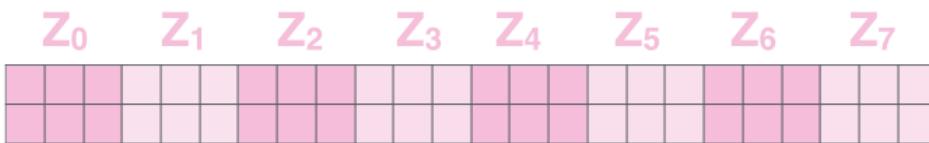


Multihead attention (8 heads)



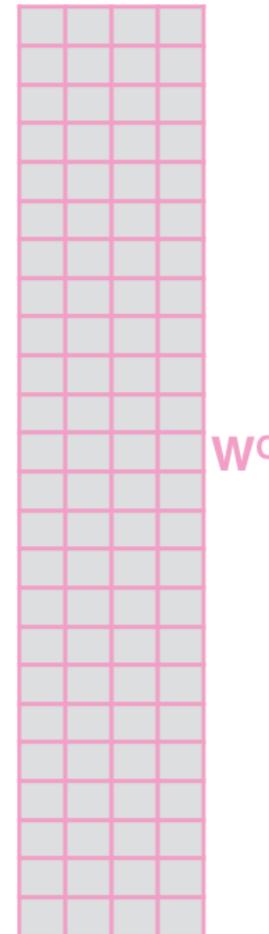
Multihead attention (8 heads)

1) Concatenate all the attention heads



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

\times



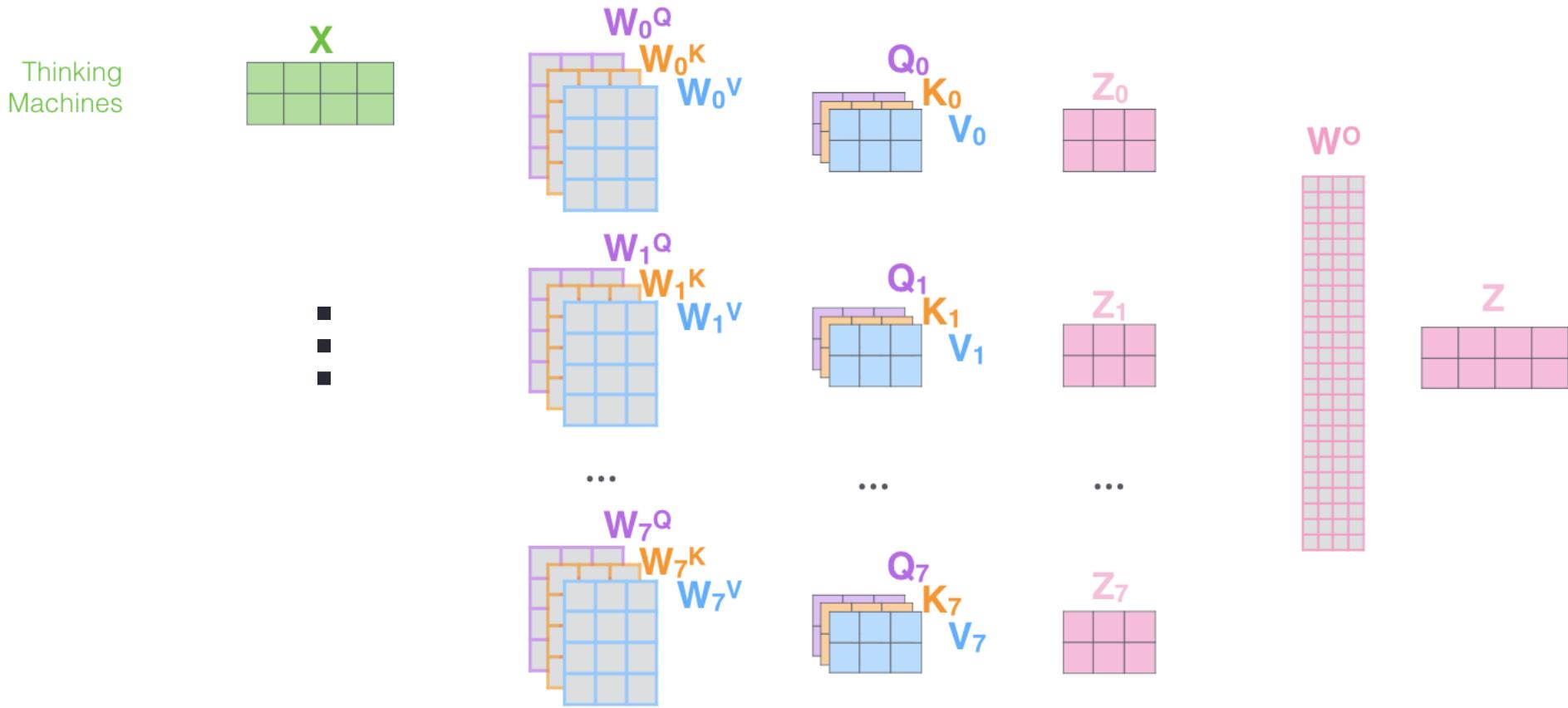
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=



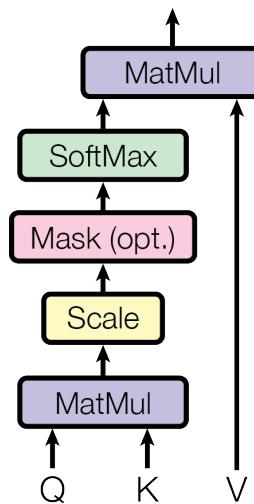
Multihead attention: full pipeline

- 1) This is our input sentence* X
- 2) We embed each word*
- 3) Split into 8 heads. Multiply X by 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

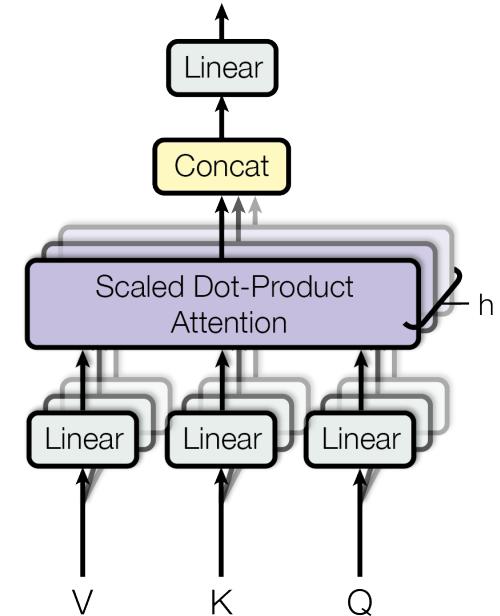


Attention mechanism

Scaled Dot-Product Attention



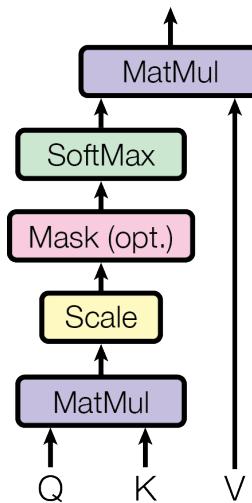
Multi-Head Attention



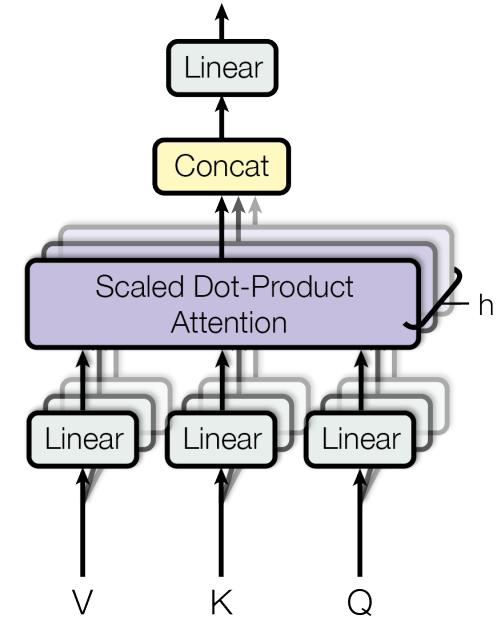
- Wait a minute... **attention does not contain any weights!!!**
- But it will make it so the weights in matrices \mathbf{W}^Q , \mathbf{W}^K , \mathbf{W}^V will be learned so as **to extract meaningful projections** of the input data – projections that will make sense when self-attention is used with them

Attention mechanism

Scaled Dot-Product Attention



Multi-Head Attention



- Wait a minute... **attention does not contain any weights!!!**
- But it will make it so the weights in matrices \mathbf{W}^Q , \mathbf{W}^K , \mathbf{W}^V will be learned so as **to extract meaningful projections** of the input data – **projections that will make sense when self-attention is used with them**

Problem

- **Nothing** of what we have seen so far (self-attention) depends on the ordering of data items (words)
- The output representations z_i are order-independent
- However, we are processing time series and, as such, we would like to **track temporal dependencies**

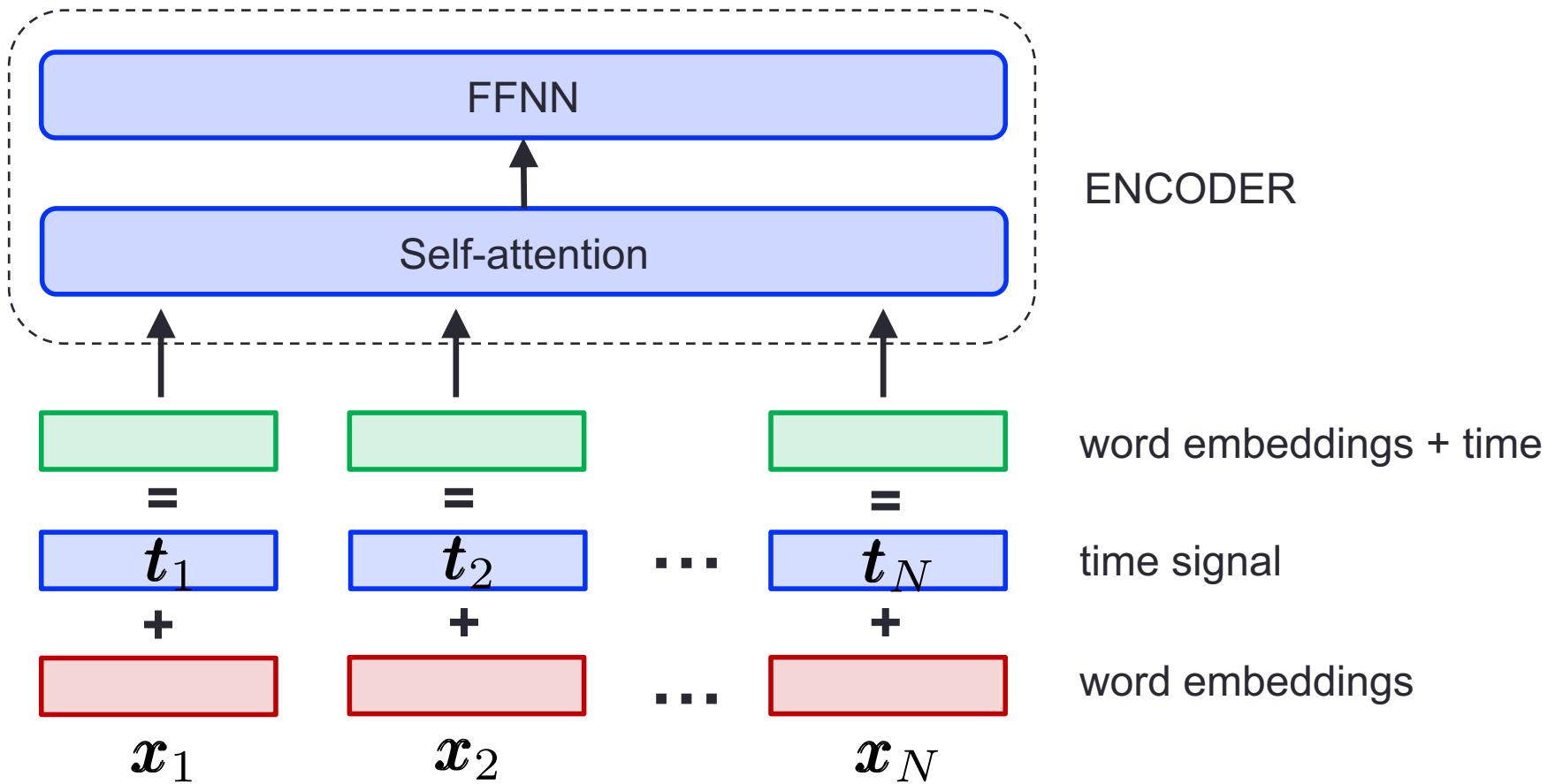
Solution: time signal

- For the model to make use of the order of the sequence
- We add timing information to the data
 - Generating a (deterministic) time signal and adding it to the input items
 - Such time signal is called **positional encoding (PE)**
- PE vectors have the same dimension of word embeddings
 - So that the two can be summed
 - Let **m** be the size of the input vectors $|\mathbf{x}_n|=m$
 - Let $t(n,i)$ be the **element i** of the **PE vector** for data **item n**
 - With $n=1,\dots,N$ and $i=1,\dots,m$

$$t(n, 2i) = \sin(n/10000^{2i/m})$$

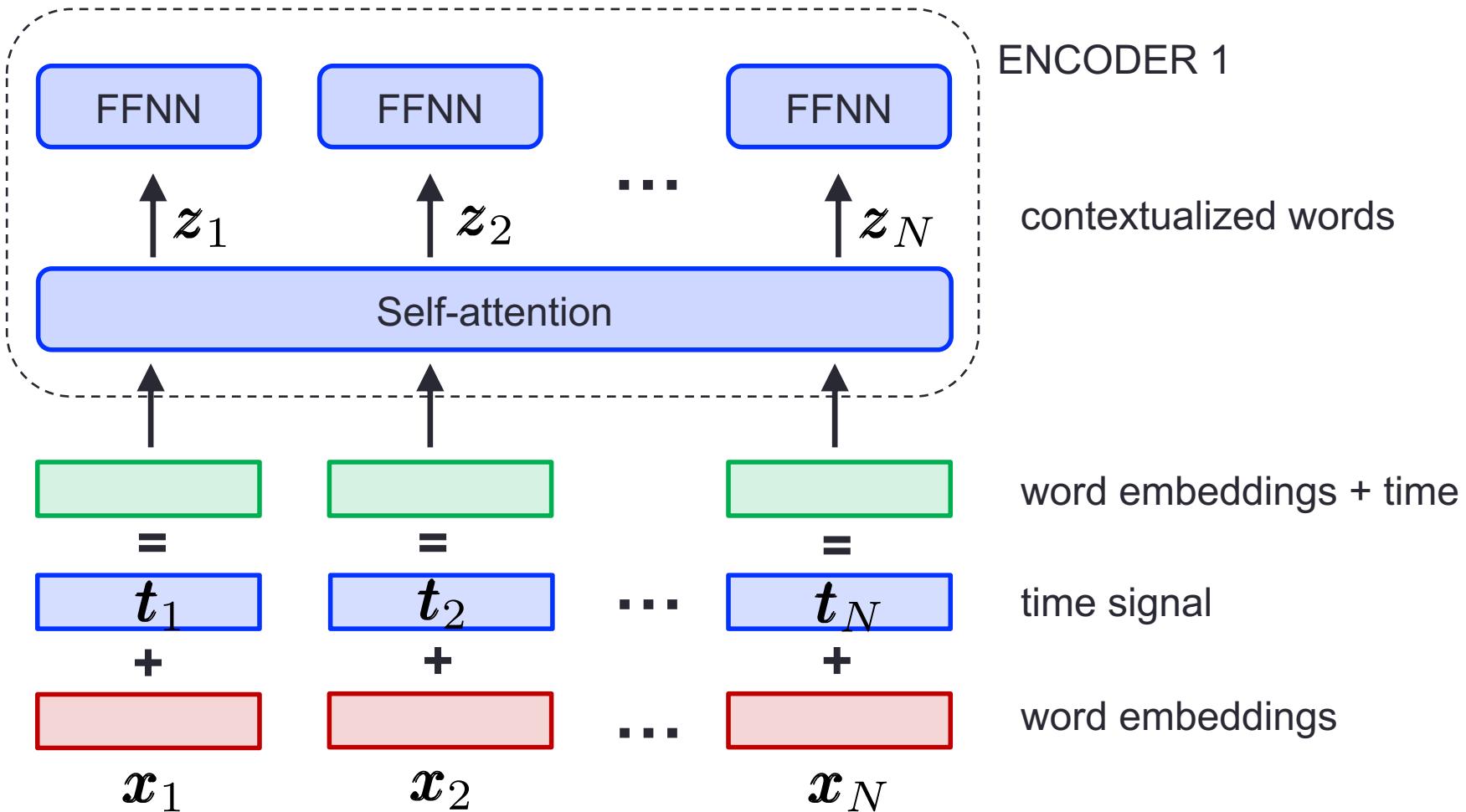
$$t(n, 2i + 1) = \cos(n/10000^{2i/m})$$

Word embeddings + time signal

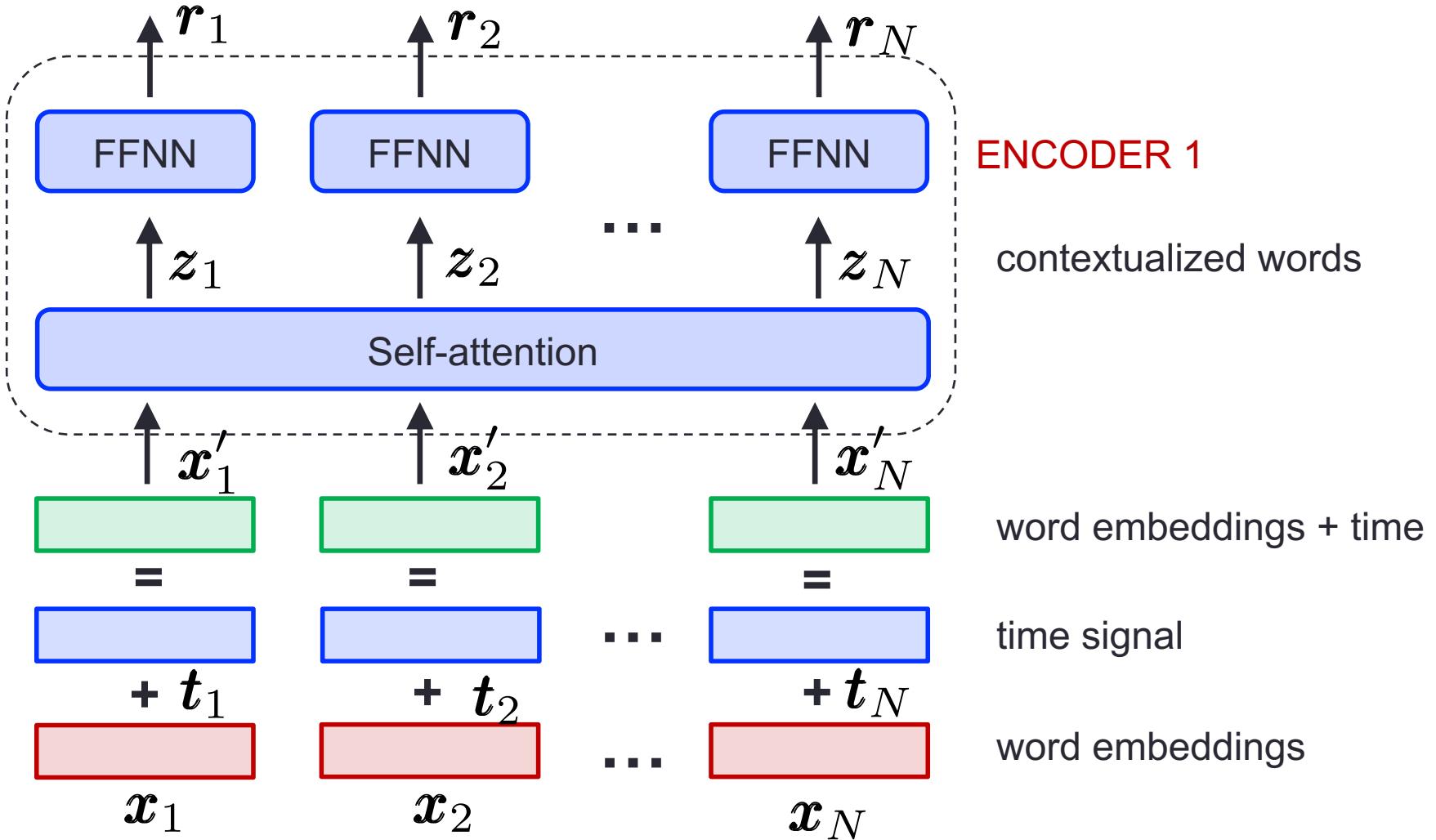


FFNN

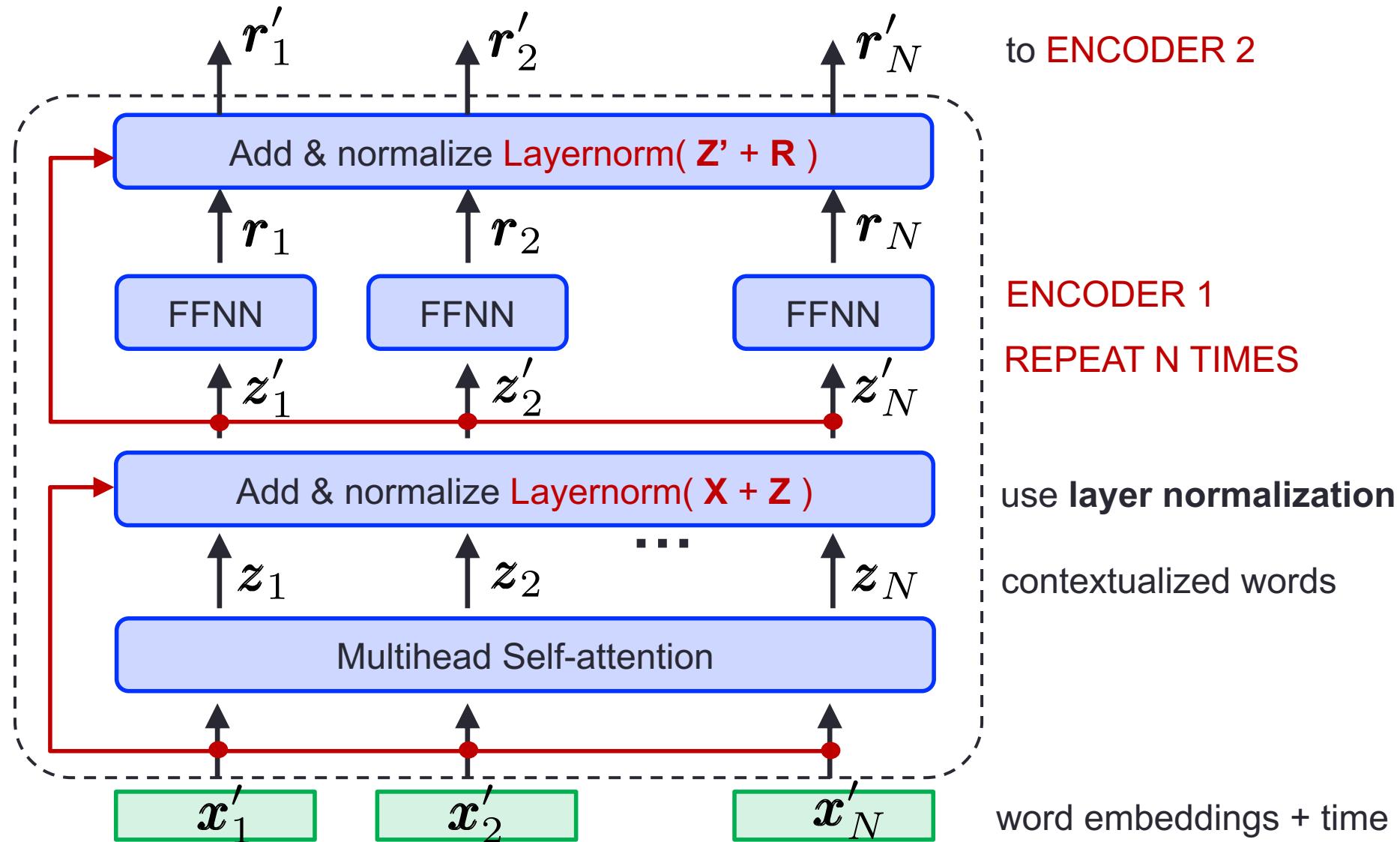
The same FFNN (same weights) is applied to each z_i independently. It consists of two linear transformations with a **ReLU non-linearity** in between. FFNN weights are shared across z_i but differ from layer to layer



$$\mathbf{r}_i = \text{FFNN}(\mathbf{z}_i) = \max(0, \mathbf{z}_i \mathbf{W}_1 + \mathbf{b}_1) \mathbf{W}_2 + \mathbf{b}_2$$



Residual connection and normalization



Batchnorm (BN)

- It is a technique proposed by Google in **[Ioffe2015]**
- Rescales vectors by first bringing them to
 - Zero mean (remove mean over a mini-batch)
 - Unit variance (over a mini-batch)
- Means and variances are computed on mini-batches
 - Two scaling parameters are learned to reshape the output vector
 - So that it works best (range and stdev) with the non-linearities at the following layer
- It largely improves **learning speed** and **stability**
- Often used in combination with residual (skip) connections

[Ioffe2015] S. Ioffe, C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” Proceedings of the 32nd Int. Conf. on Machine Learning (ICML), Lille, France, July 2015.

BN transform (1/2)

- At training time t , BN transforms input \mathbf{z}_t into output \mathbf{z}'_t



- Input vector $\mathbf{z}_t = [z_t^{(1)} \ z_t^{(2)} \ \dots \ z_t^{(H)}]^\top$ has H activations
- The BN transform is applied to each activation $z_t^{(k)}$ independently
- Next, we refer to activation k , by omitting the apex (k)
- For this specific activation, we refer to a mini-batch

$$\mathcal{B} = \{z_{1\dots m}\}$$

BN transform (2/2)

- Let us consider any activation k (from $1, \dots, H$)
- Input:** values of for such activation over a mini-batch are:
- Output values for the mini-batch are:** $\mathcal{B} = \{z_1 \dots m\}$

$$\{z'_i = \text{BN}_{\gamma, \beta}(z_i), i = 1, \dots, m\}$$

$$1) \quad \mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^m z_i \quad 2) \quad \sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{i=1}^m (z_i - \mu_{\mathcal{B}})^2$$

$$3) \quad \hat{z}_i \leftarrow \frac{z_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad 4) \quad z'_i \leftarrow \gamma \hat{z}_i + \beta \triangleq \text{BN}_{\gamma, \beta}(z_i)$$

normalize

scale and shift

Layer normalization (LN) [Ba2016]

- For the transformer, LN is used in place of BN
- Same concept: **rescale vectors**
- However, mean and variance are *solely* computed **using the current vector instead** of using all the vectors in the mini-batch
- If a hidden layer has H units (activations) with current vector

$$\mathbf{z} = [z^{(1)} \ z^{(2)} \ \dots \ z^{(H)}]^\top$$

- LN computes

$$\mu = \frac{1}{H} \sum_{h=1}^H z^{(h)} \quad \sigma = \sqrt{\frac{1}{H} \sum_{h=1}^H (z^{(h)} - \mu)^2}$$

[Ba2016] J. L. Ba, J. R. Kiros, G. E. Hinton, “Layer normalization,”
Tech. Report, arXiv:1607.06450v1, July 21, 2016.

Layer normalization (LN) [Ba2016]

- LN input $\mathbf{z} = [z^{(1)} \ z^{(2)} \ \dots \ z^{(H)}]^\top$

- LN computes

$$\mu = \frac{1}{H} \sum_{h=1}^H z^{(h)} \quad \sigma = \sqrt{\frac{1}{H} \sum_{h=1}^H (z^{(h)} - \mu)^2}$$

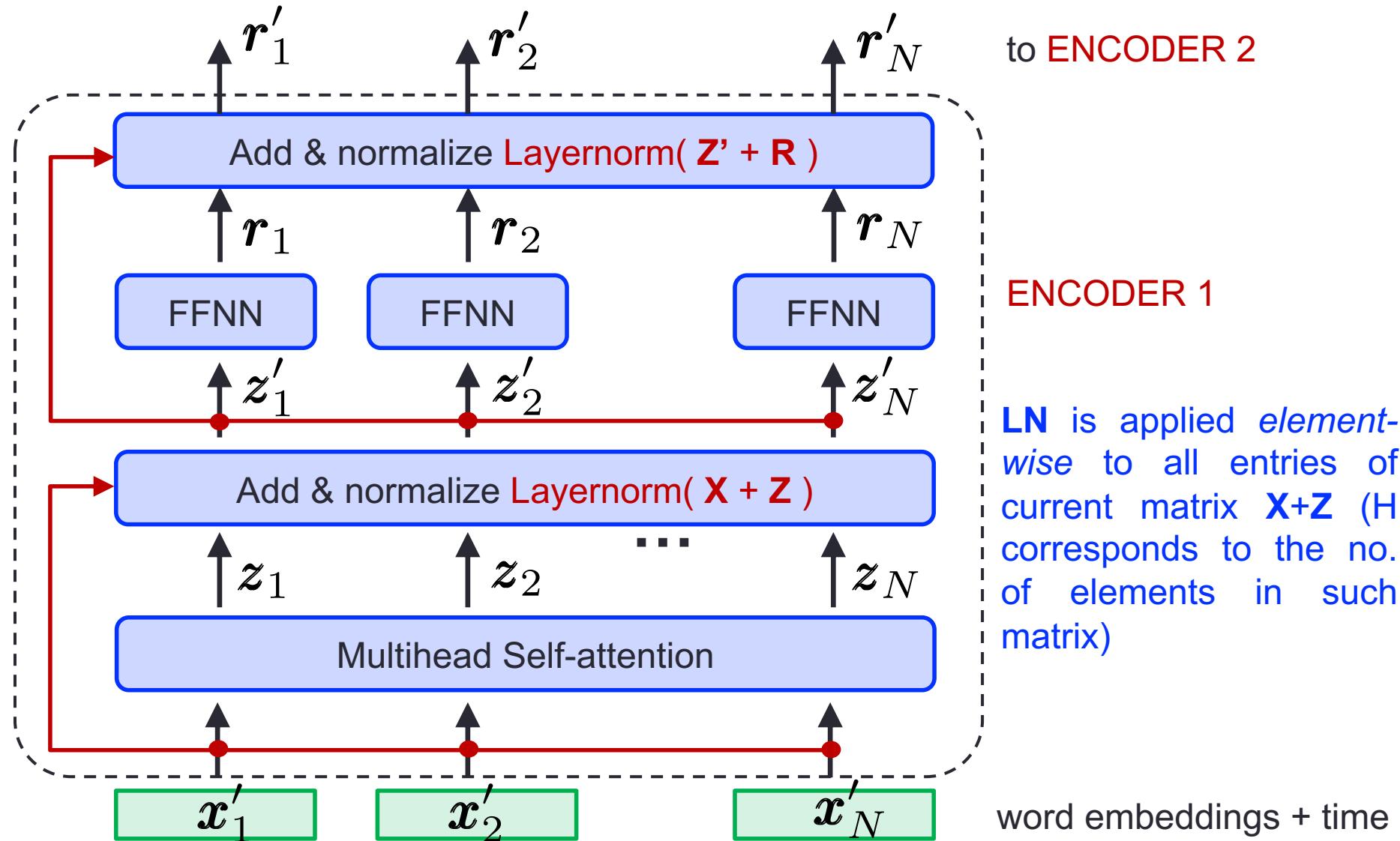
- LN output is

$$\mathbf{z}' = \frac{\mathbf{g}}{\sigma} \odot (\mathbf{z} - \mu \mathbf{1}) + \mathbf{b}$$

where \odot is the **elementwise product**, \mathbf{g} and \mathbf{b} are **gain** and **bias** vectors of the same dimension of \mathbf{z} ($\mathbf{1}$ is the vector of all 1s)

[Ba2016] J. L. Ba, J. R. Kiros, G. E. Hinton, “Layer normalization,” *Tech. Report*, arXiv:1607.06450v1, July 21, 2016.

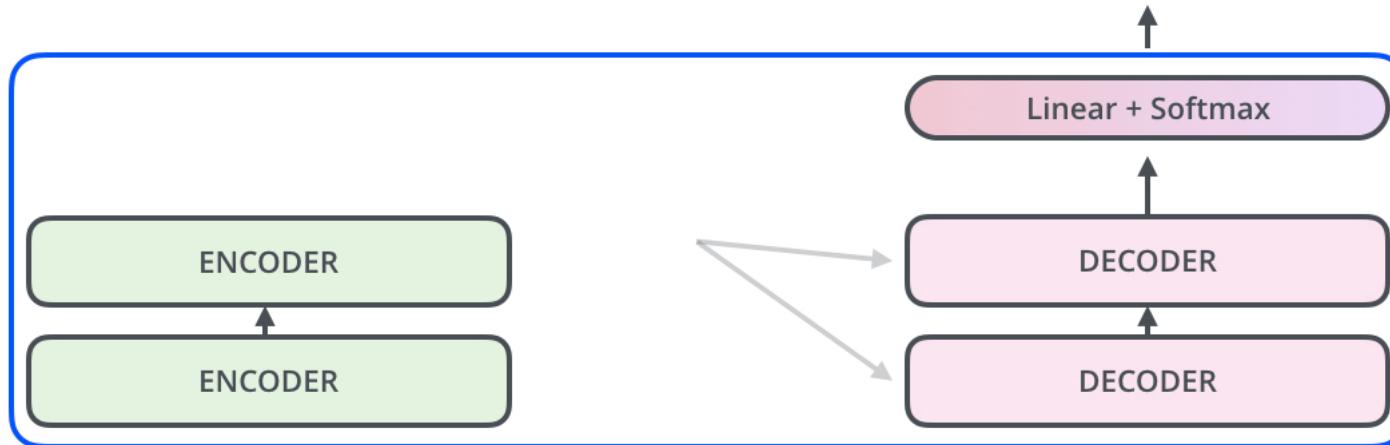
LN in the transformer



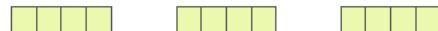
Encoder + decoder

Decoding time step: 1 2 3 4 5 6

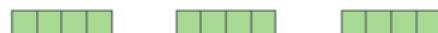
OUTPUT



EMBEDDING
WITH TIME
SIGNAL



EMBEDDINGS



INPUT Je suis étudiant

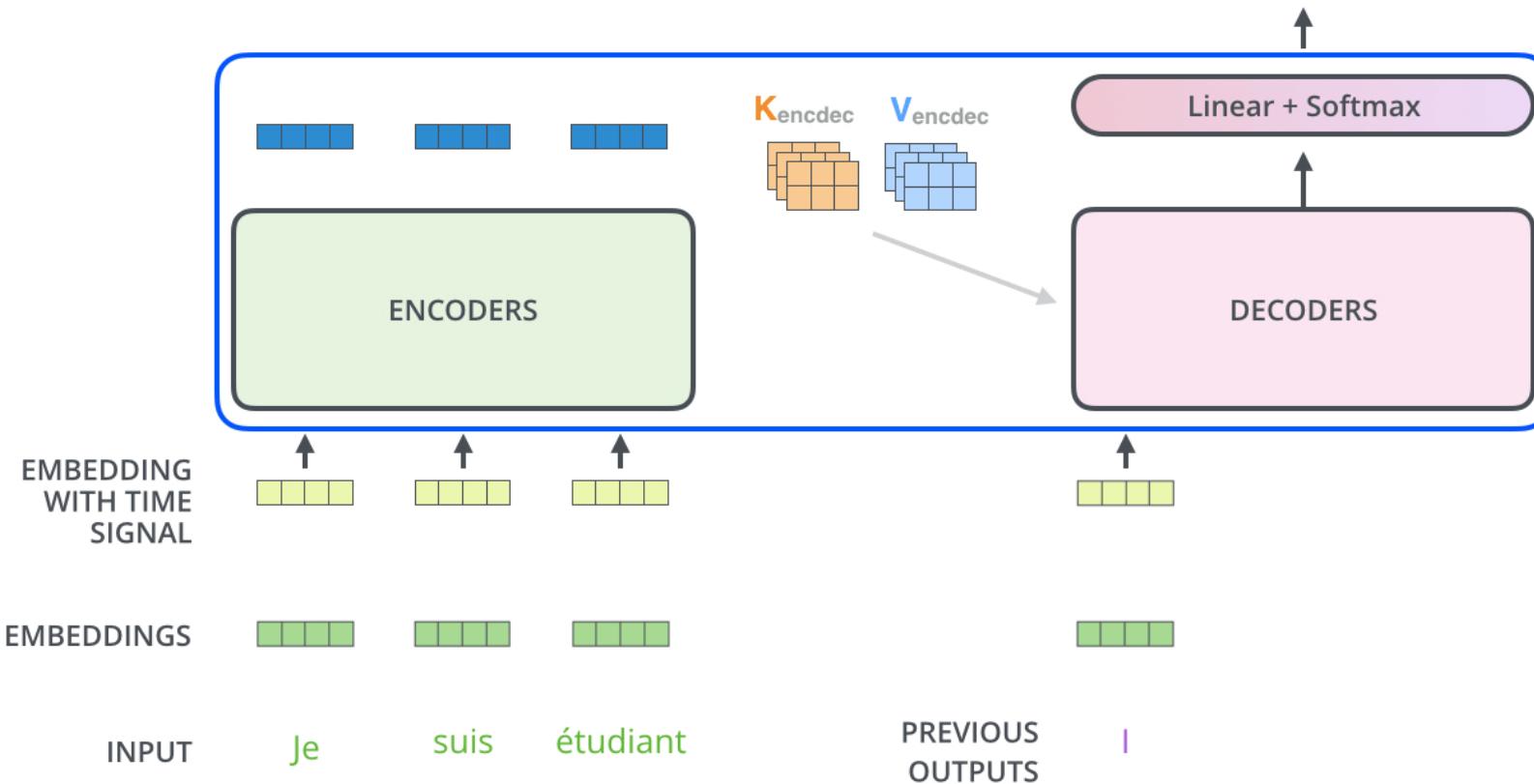
The **encoder** starts by processing the input sequence. The output of the **top encoder** is transformed into a set of attention vectors **K** and **V** (via linear projections). These are to be used by each decoder in its “encoder-decoder attention” layer which helps the decoder focus on appropriate places in the input sequence

Decoding: autoregressive process

Decoding time step: 1 2 3 4 5 6

OUTPUT

I



Decoder output

- The decoder stack outputs vectors of floats
- How do we turn it into a **final word**?
- We use a **linear (dense) layer**
 - It is a simple fully connected NN: projects the vector produced by the stack of decoders **into a much larger vector** called the **logits vector**
 - If our vocabulary is, e.g., 10,000 words: this will be the size of the logits vector
- After that: a **softmax layer** turns the logits vector
 - into a **vector of probabilities** (one for each word in dictionary)
 - the cell **with the highest probability is chosen as the output** for the next time step

Decoder output

Which word in our vocabulary
is associated with this index?

Get the index of the cell
with the highest value
(`argmax`)

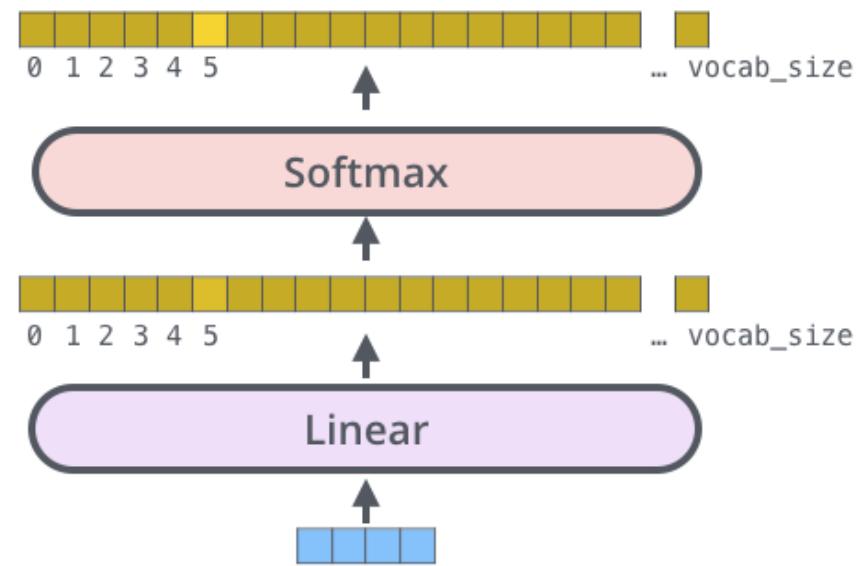
`log_probs`

`logits`

Decoder stack output

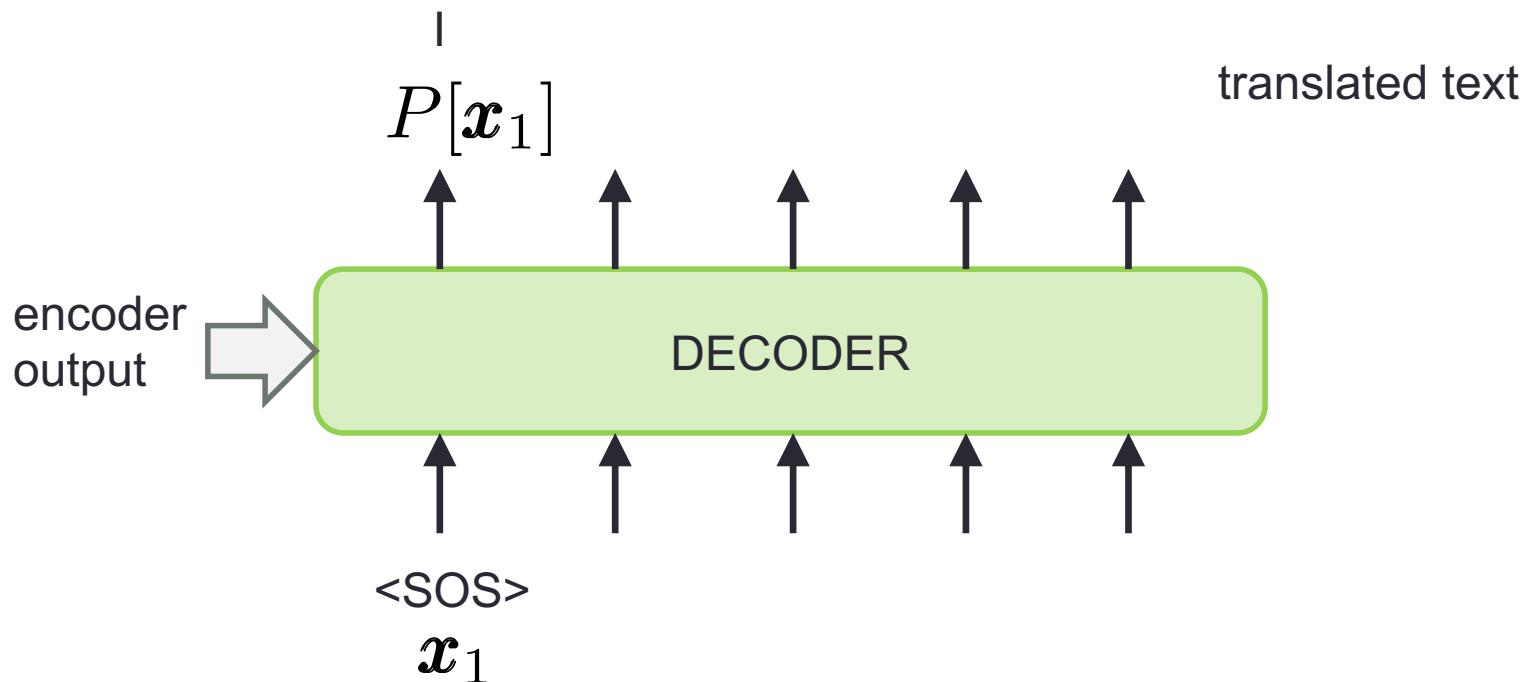
am

5



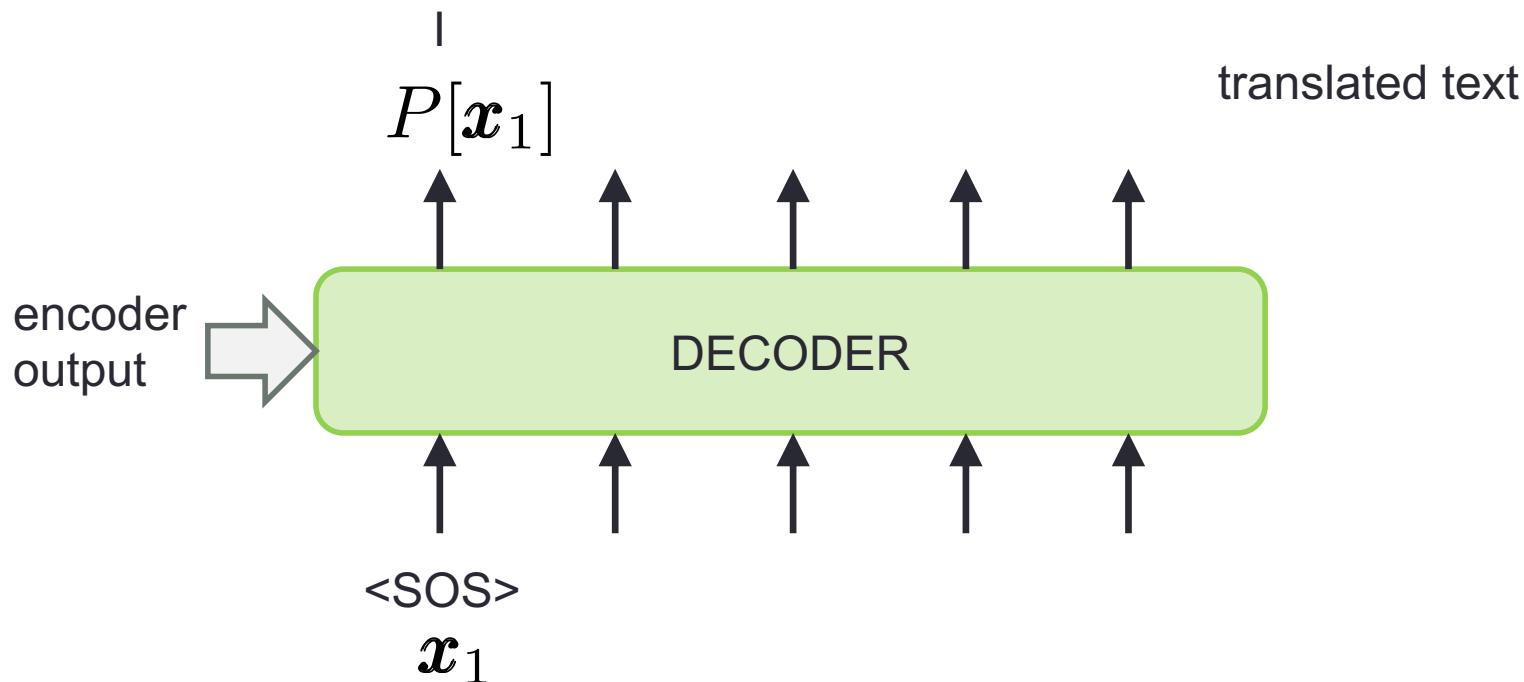
Decoder: masked self-attention

- The encoders obtain contextualized information over the whole sequence - it is used by the **cross-attention blocks** at the decoder to infer the next word in the translation
- Let's look at the traslation of the italian sequence “io ho una macchina” into English “I have a car”



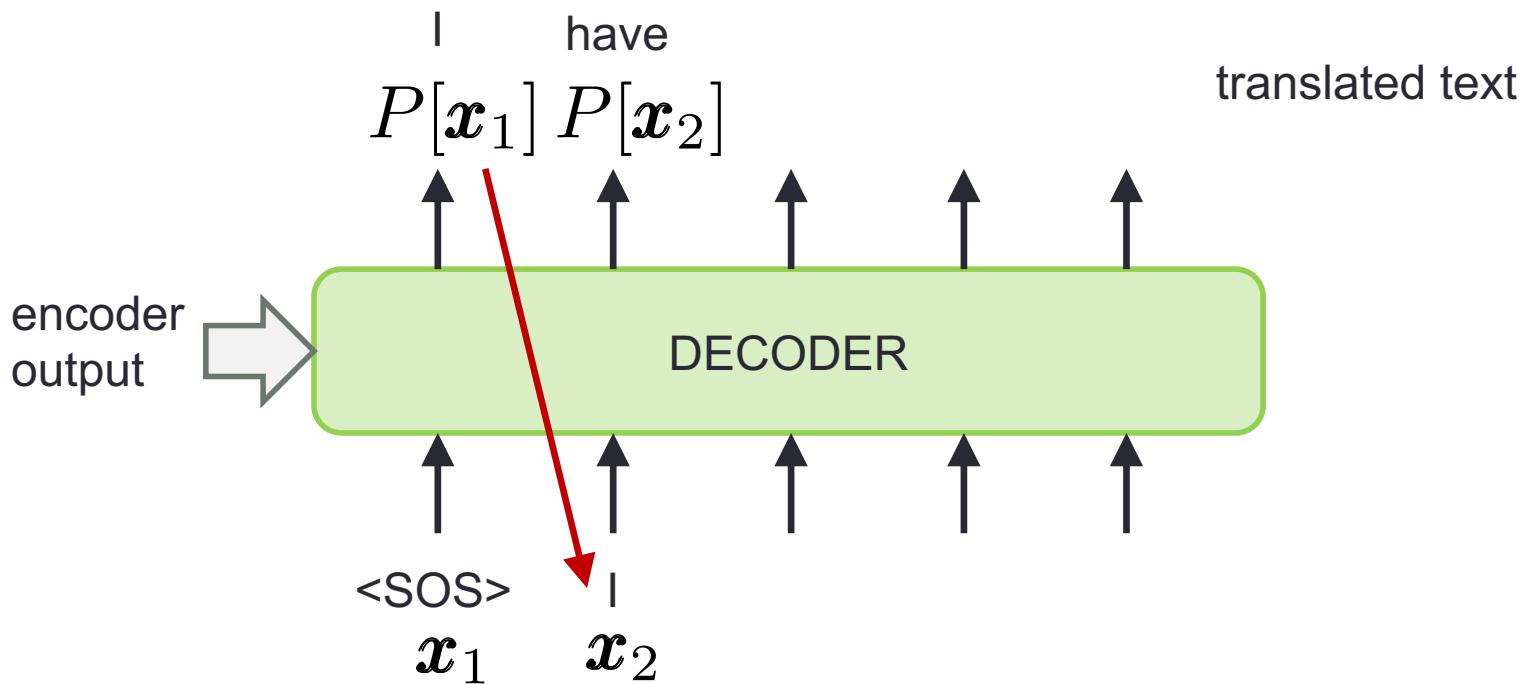
Decoder: masked self-attention

- As a first step, we input the start of sequence <SOS> symbol to the decoder: the first output should be “I” → the softmax should peak at the “I” word from the dictionary
- The output in this case is **solely inferred from the encoder output**



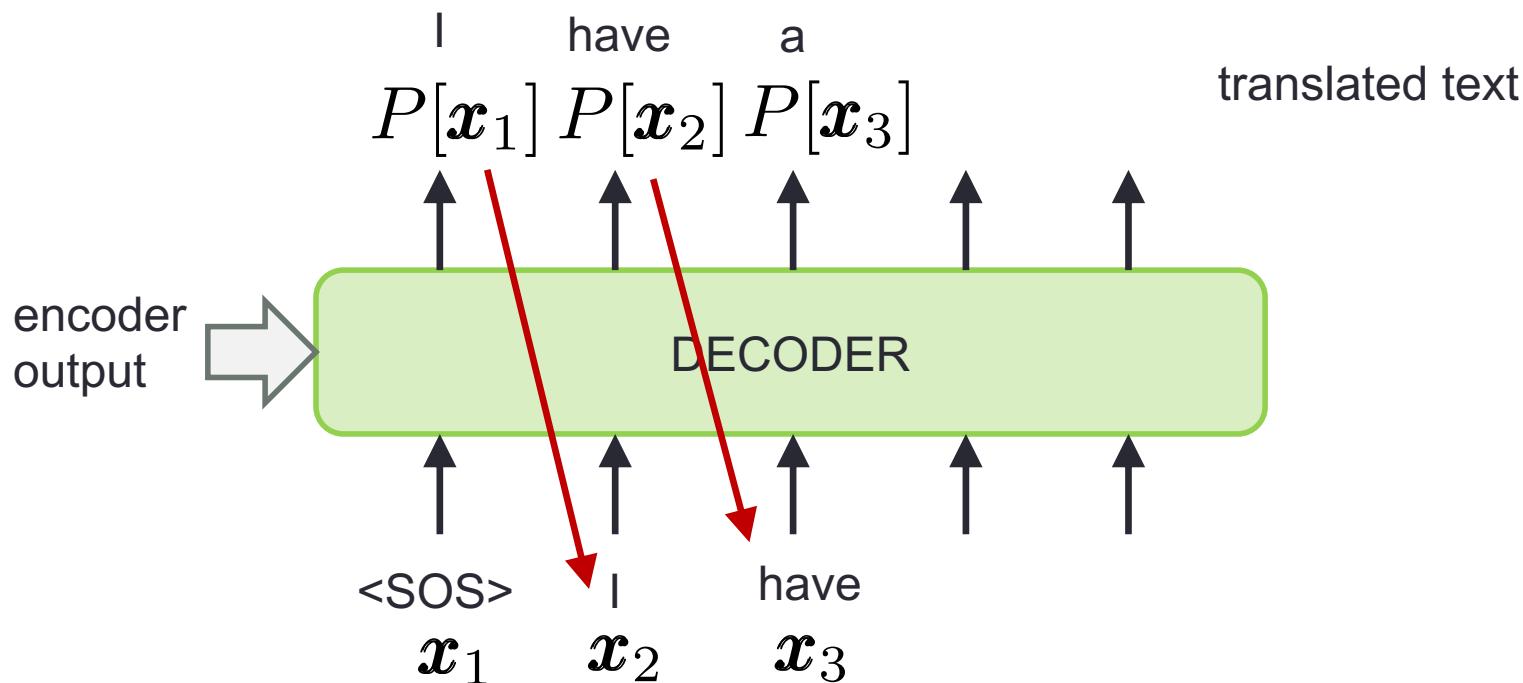
Decoder: masked self-attention

- **Step 2:** we autoregressively copy the word “I” to the next decoder input – the word “have” should now be inferred from
 - 1) the context from the encoder
 - 2) the previous decoded words <SOS> and “I”



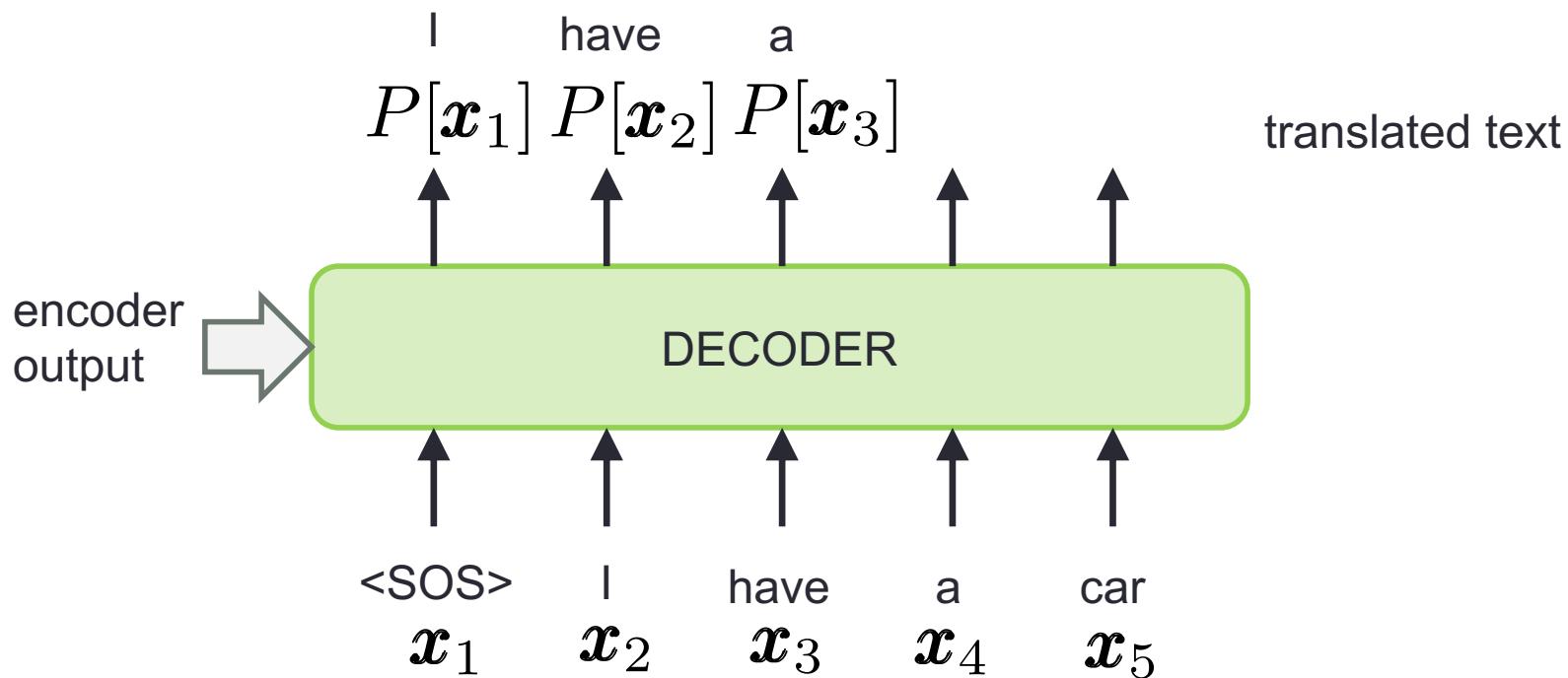
Decoder: masked self-attention

- **Step 3:** we autoregressively copy the word “have” to the next decoder input – the word “a” should now be inferred from
 - 1) the context from the encoder
 - 2) the previous decoded words <SOS>, “I” and “have”



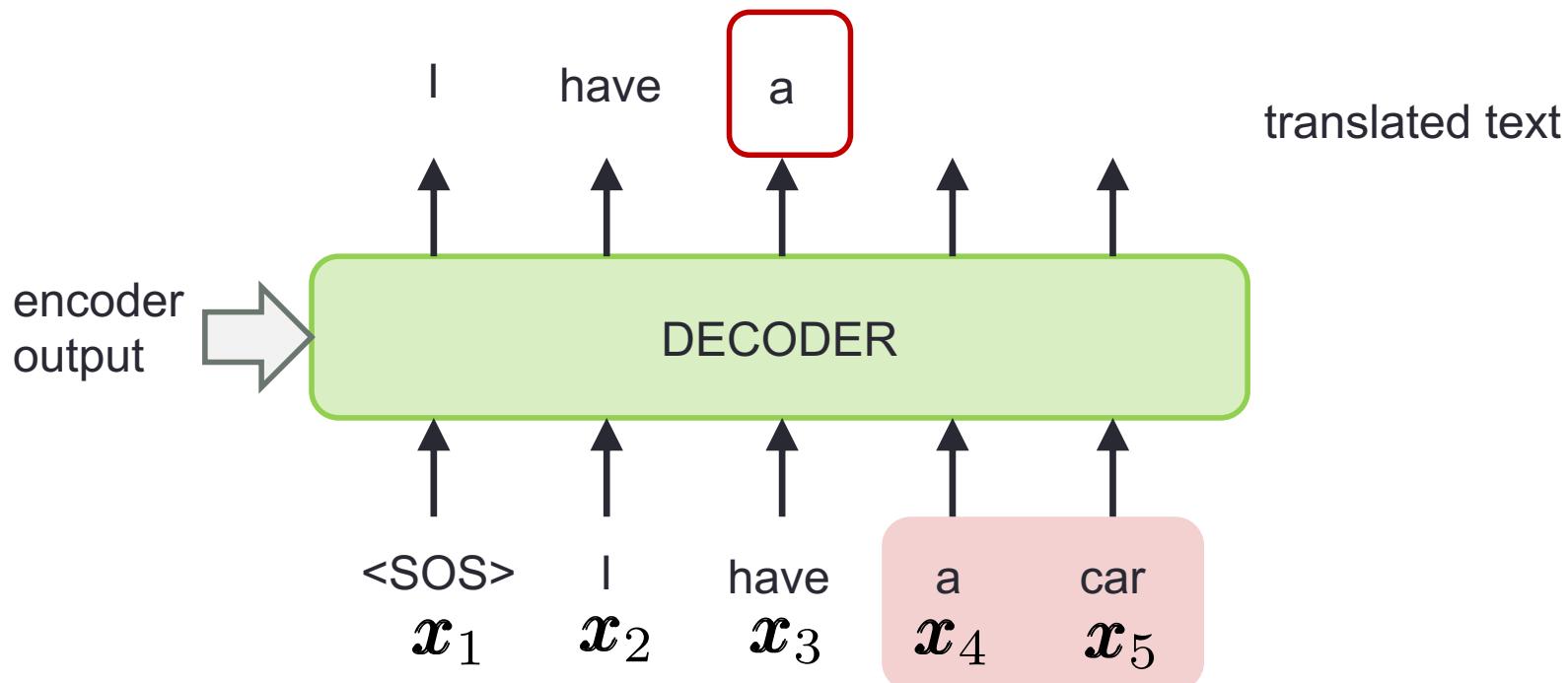
Decoder: masked self-attention

- At training time, the full translated sequence is available (label)
- However, we should train the decoder **in order not to cheat**



Decoder: masked self-attention

- Let training be at time 3 at the decoder side
- For this, only x_1 , x_2 and x_3 should be used
→ **no access** to x_4 and x_5



Decoder: masked self-attention

- Compute queries, keys, values. The scores are:

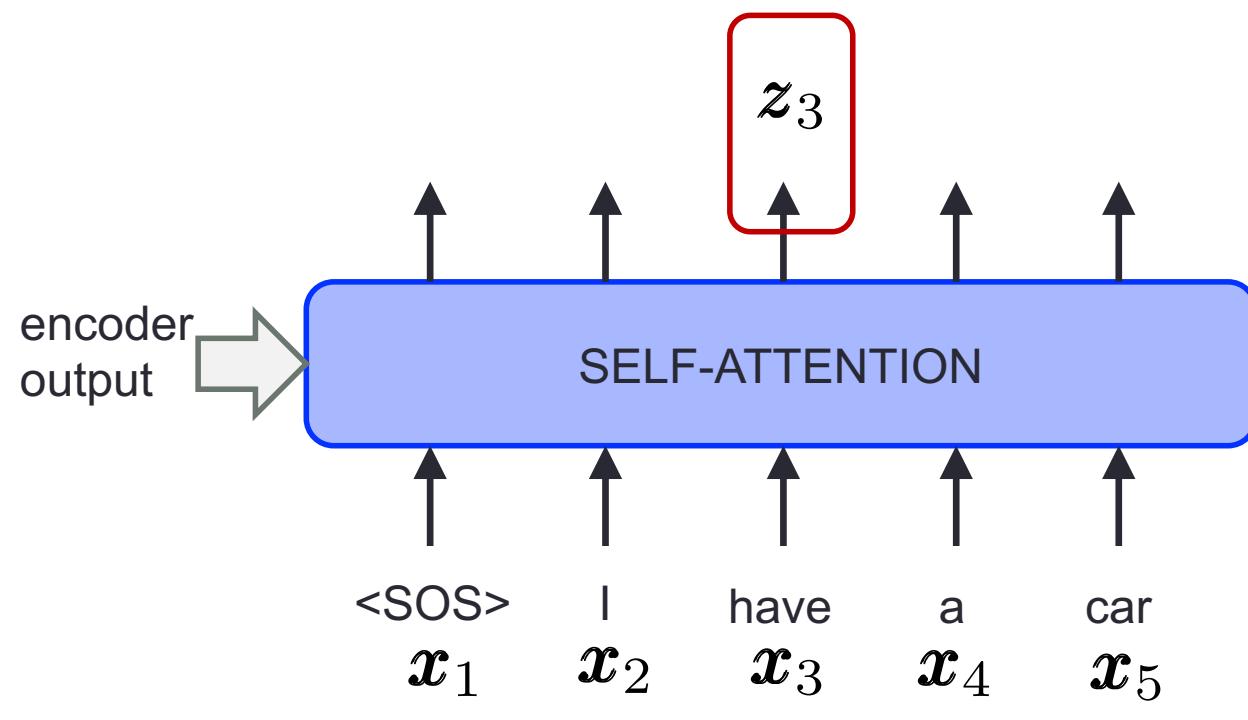
$$s_{31} = \frac{\mathbf{k}_1 \mathbf{q}_3^\top}{\sqrt{d}}, \dots, s_{35} = \frac{\mathbf{k}_5 \mathbf{q}_3^\top}{\sqrt{d}}$$

Masked (normalized) weights

$$\begin{bmatrix} \alpha_{31} \\ \alpha_{32} \\ \alpha_{33} \\ \alpha_{34} \\ \alpha_{35} \end{bmatrix}^\top = \begin{bmatrix} \exp(s_{31})/S \\ \exp(s_{32})/S \\ \exp(s_{33})/S \\ 0 \\ 0 \end{bmatrix}^\top$$

$$S = \sum_{i=1}^3 \exp(s_{3i})$$

$$\mathbf{z}_3 = \sum_{j=1}^5 \alpha_{3j} \mathbf{v}_j$$



Decoder: masked self-attention

- Repeating for all \mathbf{z}_i
- The softmax weights expressed in terms of matrix product are (sequence length = 5)

$$\mathbf{K} \times \mathbf{Q}^T = \begin{bmatrix} \alpha_{11} & 0 & 0 & 0 & 0 \\ \alpha_{21} & \alpha_{22} & 0 & 0 & 0 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0 & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} & 0 \\ \alpha_{51} & \alpha_{52} & \alpha_{53} & \alpha_{54} & \alpha_{55} \end{bmatrix}$$

- It is implemented as a **mask** applied to the **softmax weights**
- by setting the weights $s_{ij} = -\infty$ in the positions with 0 in the above matrix

Discussion

- The paper title says “Attention is all you need”
- This may be replaced with “**Correlation** is all you need”
- It looks like the correlation structure that is embedded into data sequences, images, sentences is the most important aspect to capture
- There are signs that (with advanced transformer architectures) this correlation also **enables computational thinking** and **reasoning** as a (rather unexpected) result
- Future results will unveil whether this is the case

Large Language Models (LLMs) Landscape

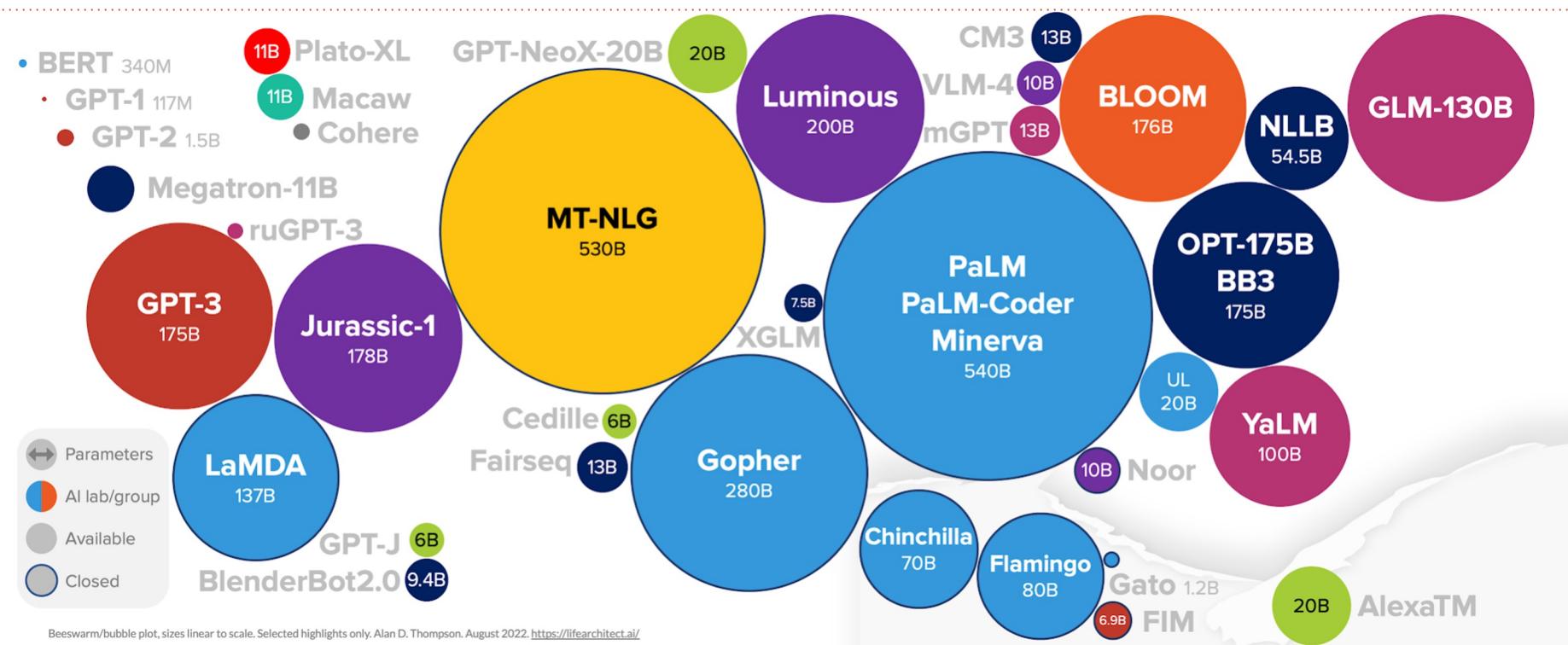


Figure: Alan D. Thompson, “[Inside language models \(from GPT-3 to PaLM\)](#)” (2022).

Optimal Large Language Models



Training Compute-Optimal Large Language Models

Jordan Hoffmann*, Sebastian Borgeaud*, Arthur Mensch*, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals and Laurent Sifre*

*Equal contributions

We investigate the optimal model size and number of tokens for training a transformer language model under a given compute budget. We find that current large language models are significantly under-trained, a consequence of the recent focus on scaling language models whilst keeping the amount of training data constant. By training over 400 language models ranging from 70 million to over 16 billion parameters on 5 to 500 billion tokens, we find that for compute-optimal training, the model size and the number of training tokens should be scaled equally: for every doubling of model size the number of training tokens should also be doubled. We test this hypothesis by training a predicted compute-optimal model, *Chinchilla*, that uses the same compute budget as *Gopher* but with 70B parameters and 4 \times more data. *Chinchilla* uniformly and significantly outperforms *Gopher* (280B), GPT-3 (175B), Jurassic-1 (178B), and Megatron-Turing NLG (530B) on a large range of downstream evaluation tasks. This also means that *Chinchilla* uses substantially less compute for fine-tuning and inference, greatly facilitating downstream usage. As a highlight, *Chinchilla* reaches a state-of-the-art average accuracy of 67.5% on the MMLU benchmark, greater than a 7% improvement over *Gopher*.

Optimal Large Language Models



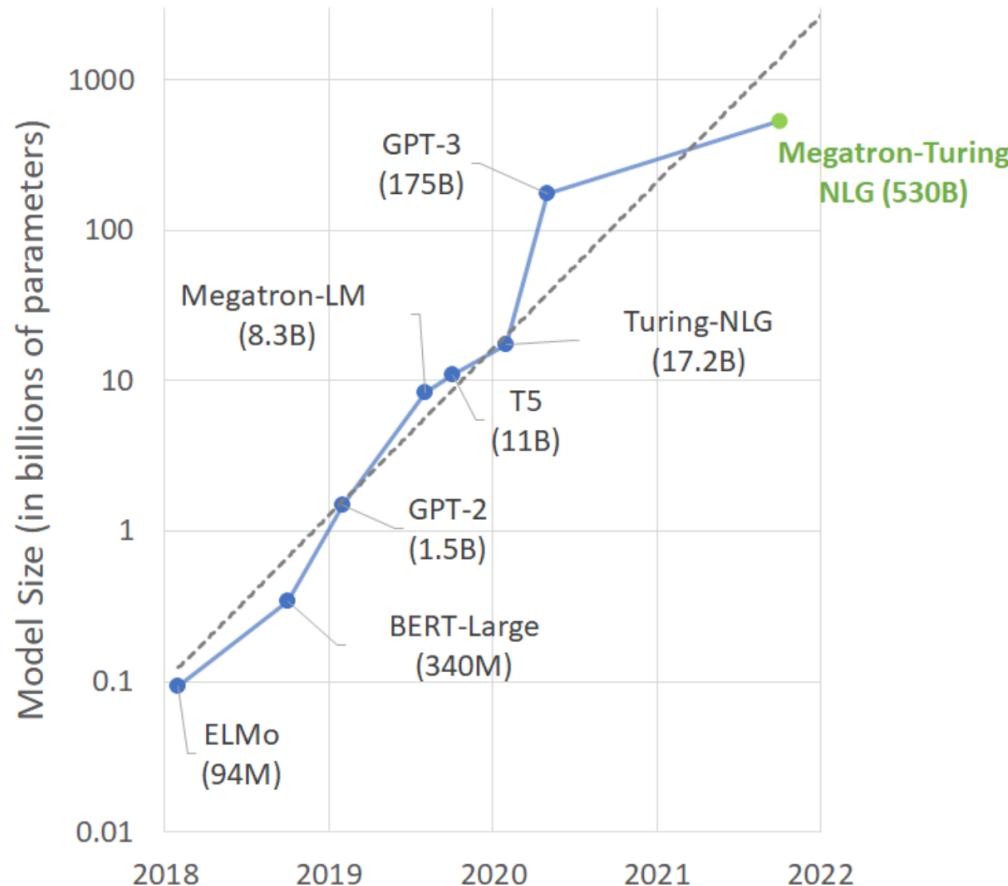
Training Compute-Optimal Large Language Models

Jordan Hoffmann*, Sebastian Borgeaud*, Arthur Mensch*, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals and Laurent Sifre*

Table 1 | Current LLMs. We show five of the current largest dense transformer models, their size, and the number of training tokens. Other than LaMDA ([Thoppilan et al., 2022](#)), most models are trained for approximately 300 billion tokens. We introduce *Chinchilla*, a substantially smaller model, trained for much longer than 300B tokens.

Model	Size (# Parameters)	Training Tokens
LaMDA (Thoppilan et al., 2022)	137 Billion	168 Billion
GPT-3 (Brown et al., 2020)	175 Billion	300 Billion
Jurassic (Lieber et al., 2021)	178 Billion	300 Billion
Gopher (Rae et al., 2021)	280 Billion	300 Billion
MT-NLG 530B (Smith et al., 2022)	530 Billion	270 Billion
<i>Chinchilla</i>	70 Billion	1.4 Trillion

A new Moore law?



[Brown2022] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., et al. "[Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model](#)," arXiv 2201.11990v3 [cs.CL] 4 Feb 2022.

WHY SCALED DOT PRODUCTS?

Scaled dot products

- In papers and books, you often read something like:

“As you increase the number of dimensions, the dot products grows larger in magnitue → pushing the softmax function into regions where it has extreme gradients” [Kamath2022]

EXTREME GRADIENTS?

Scaled dot products: softmax

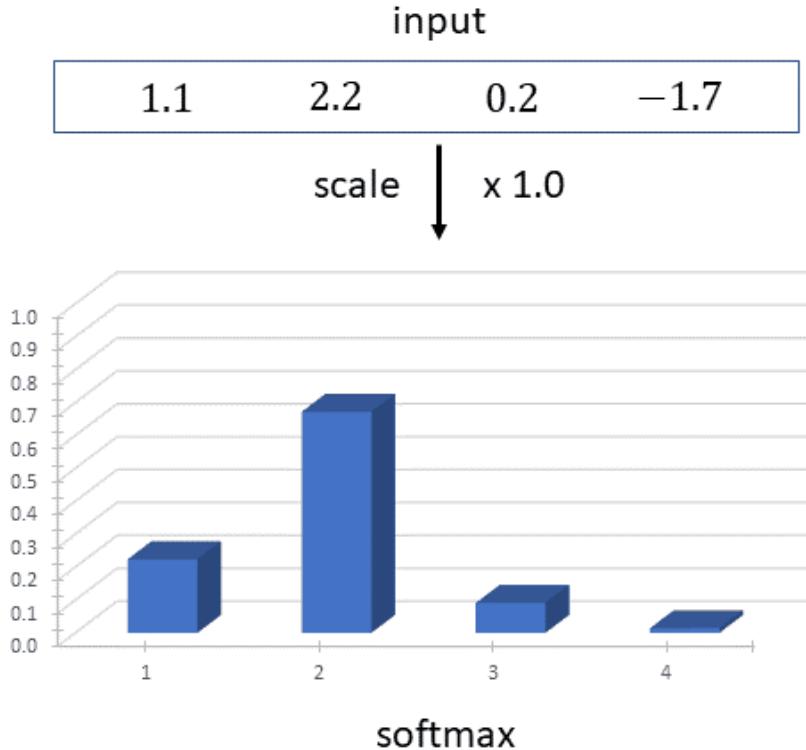
- To see this, let us take a look at the softmax function
- It is a vector function $\text{softmax} : \mathbb{R}^n \rightarrow \mathbb{R}^n$

softmax

$$a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} \begin{array}{l} 1.1 \\ 2.2 \\ 0.2 \\ -1.7 \end{array} \rightarrow \boxed{b_i = \frac{e^{a_i}}{\sum_j e^{a_j}}} \rightarrow \begin{array}{l} 0.224 \\ \color{red}{0.672} \\ 0.090 \\ 0.014 \end{array} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = b$$
$$b_i \in [0, 1], \sum_i b_i = 1$$

Scaled dot products: softmax

- softmax is NOT scale invariant, as shown here below



The higher we scale the input, the more the largest input dominates the softmax output → getting close to 1

If we reduce the scale factor, the outputs become very similar

Let's verify this

- Vectors of size **n**, pick an **index j, $\gamma=\text{constant}$** →

$$a_j = \gamma, a_i = \gamma - \varepsilon, \text{ for } i \neq j, \text{ with } 0 < \varepsilon \ll \gamma$$

- Now consider a **scaled version** (by factor $K>0$) of vector **a**

$$a'_i = K a_i, i = 1, \dots, n, \text{ with } K > 0$$

- The **softmax** applied to this scaled vector leads to

$$b_i = \frac{e^{-K\varepsilon}}{1 + (n-1)e^{-K\varepsilon}} \xrightarrow[K \rightarrow \infty]{} 0, i \neq j$$

$$b_j = \frac{1}{1 + (n-1)e^{-K\varepsilon}} \xrightarrow[K \rightarrow \infty]{} 1$$

- This means that small differences gets amplified for large K

Scaled dot products: softmax

- softmax
 - computing the Jacobian (gradient for a vector function)

$$J = \begin{bmatrix} \frac{\partial b_1}{\partial a_1} & \frac{\partial b_1}{\partial a_2} & \cdots & \frac{\partial b_1}{\partial a_n} \\ \frac{\partial b_2}{\partial a_1} & \frac{\partial b_2}{\partial a_2} & \cdots & \frac{\partial b_2}{\partial a_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial b_n}{\partial a_1} & \frac{\partial b_n}{\partial a_2} & \cdots & \frac{\partial b_n}{\partial a_n} \end{bmatrix}$$

- It can be proven than (see Appendix B)

$$\frac{\partial b_i}{\partial a_j} = b_i(\delta_{i,j} - b_j), \quad i, j = 1, \dots, n$$

Scaled dot products: softmax

- softmax
 - computing the Jacobian (gradient for a vector function), with $n=4$

$$J = \begin{bmatrix} b_1(1 - b_1) & -b_1b_2 & -b_1b_3 & -b_1b_4 \\ -b_2b_1 & b_2(1 - b_2) & -b_2b_3 & -b_2b_4 \\ -b_3b_1 & -b_3b_2 & b_3(1 - b_3) & -b_3b_4 \\ -b_4b_1 & -b_4b_2 & -b_4b_3 & b_4(1 - b_4) \end{bmatrix}$$

- Where it holds:

$$\frac{\partial b_i}{\partial a_j} = b_i (\delta_{i,j} - b_j), \quad i, j = 1, \dots, n$$

Kronecker delta

Scaled dot products: softmax

- softmax
 - computing the Jacobian (gradient for a vector function), with $n=4$

$$J = \begin{bmatrix} b_1(1 - b_1) & -b_1b_2 & -b_1b_3 & -b_1b_4 \\ -b_2b_1 & b_2(1 - b_2) & -b_2b_3 & -b_2b_4 \\ -b_3b_1 & -b_3b_2 & b_3(1 - b_3) & -b_3b_4 \\ -b_4b_1 & -b_4b_2 & -b_4b_3 & b_4(1 - b_4) \end{bmatrix}$$

- The Jacobian is **symmetric**
- Diagonal elements **become zero** when any of the b_j takes the value 0 or 1

Scaled dot products: softmax

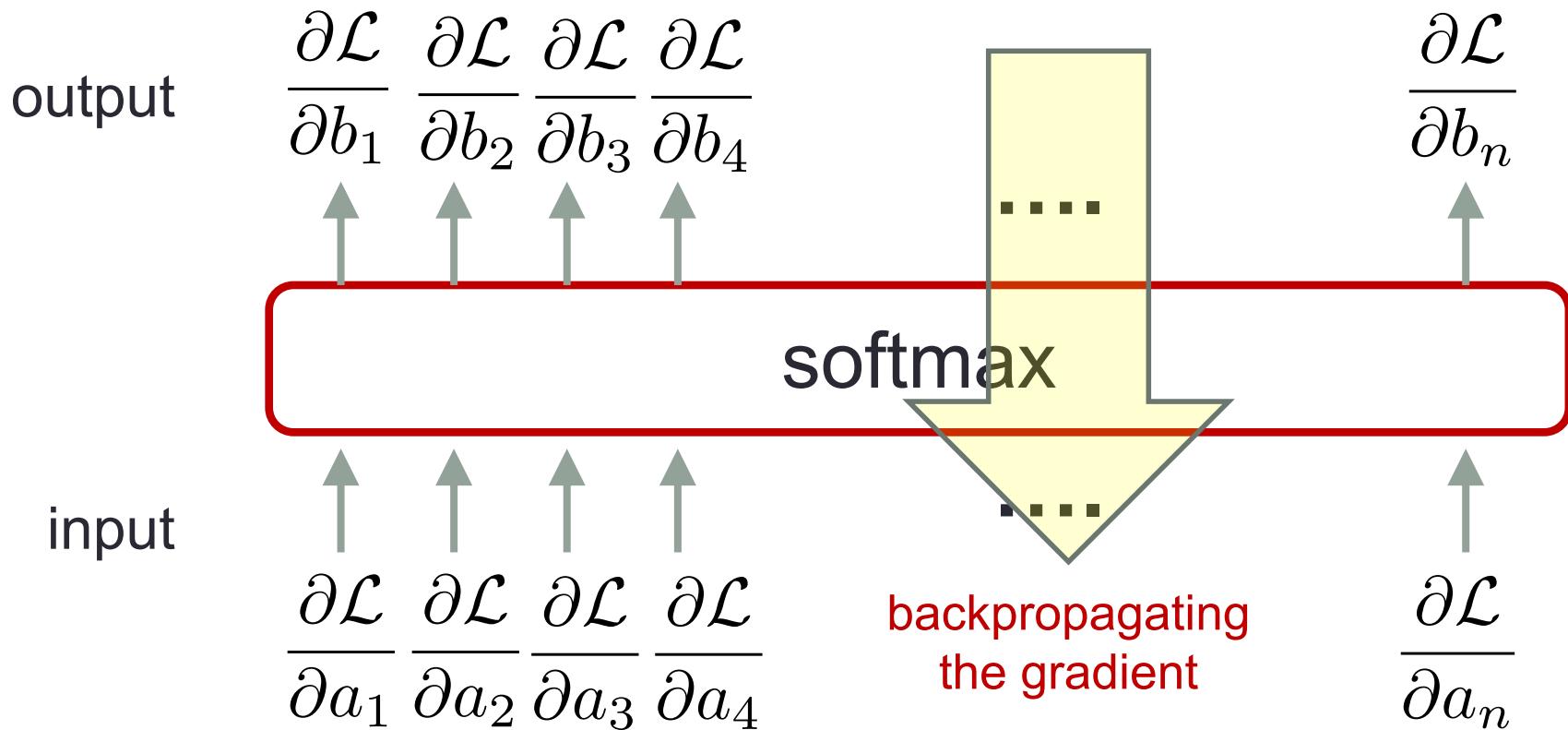
- It follows that
 - **J becomes the zero matrix** in any of the following four output vector cases

$$\mathbf{b}_a = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{b}_b = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{b}_c = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \mathbf{b}_d = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

- For **large input values** → softmax generates outputs that closely resemble the above configurations (the largest input value tends to dominate the softmax output)

Backpropagation through a softmax layer

- During backpropagation
 - We backpropagate the error gradient from output to input of the softmax layer (as we do for any other layer)



Backpropagation through a softmax layer

- Using the **chain rule of derivatives**, we write

$$\frac{\partial \mathcal{L}}{\partial a_j} = \sum_{i=1}^n \frac{\partial \mathcal{L}}{\partial b_i} \times \frac{\partial b_i}{\partial a_j} = \begin{bmatrix} \frac{\partial b_1}{\partial a_j} & \frac{\partial b_2}{\partial a_j} & \cdots & \frac{\partial b_n}{\partial a_j} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial b_1} \\ \frac{\partial \mathcal{L}}{\partial b_2} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial b_n} \end{bmatrix}$$

- The row vector on the right size of this equation is the j-th column (transposed) of the Jacobian matrix \mathbf{J}

- Given this, in vector form it holds

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}} = \mathbf{J}^T \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{b}}$$

Backprop outcome

- Backprop works by moving the error gradient from output layer to input layer according to

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}} = \mathbf{J}^T \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{b}}$$

- When softmax input values grow in magnitude → the output emphasizes the largest input element that gets close to 1 and the remaining elements get close to 0
- In this situation the **J matrix tends to the 0 matrix** and the gradient gets killed by the softmax layer
- As a consequence, all elements coming before the softmax layer **slow down learning** or even **stop learning all together**

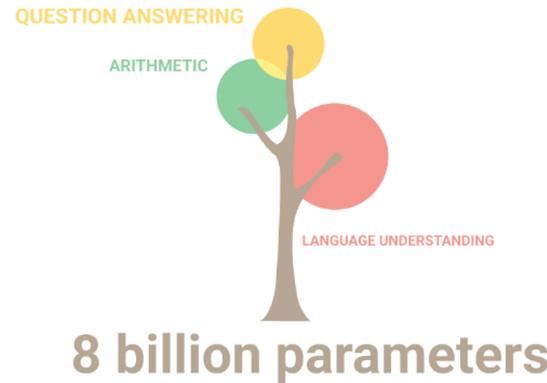
Solution to vanishing softmax gradient

- In the original transformer paper [Vaswani2017]
 - The input to the softmax corresponds to the result of the **dot product** between *key vectors* and *query vectors* (of the same dimension **d=64**)
 - The larger the dimension d of such vectors, the larger the dot product will tend to be
 - The remedy used by the authors is to divide the dot product by the **square root of the dimension d**
 - In this way, learning **will not be affected by the gradient vanishing problem**

This shows how a small detail can **have a huge impact**

MORE APPLICATIONS AND TRENDS

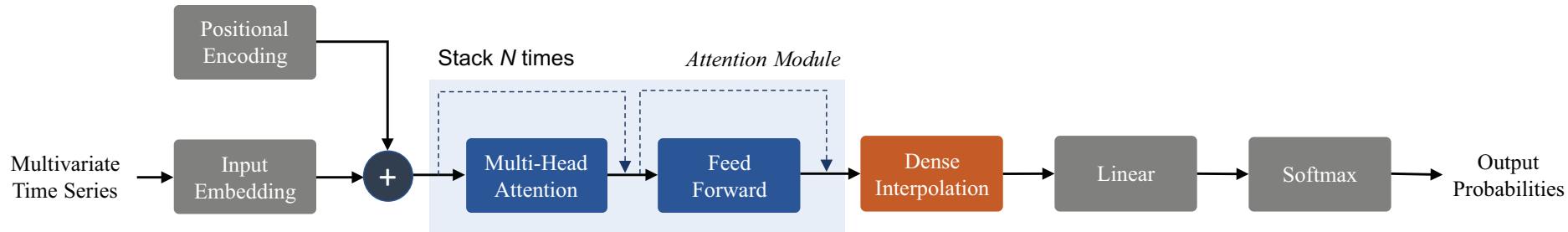
The growing tree of capabilities...



Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., et al. (2022).
[“PaLM: Scaling Language Modeling with Pathways,” arXiv2203..](#)

Attend and diagnose [Song2018]

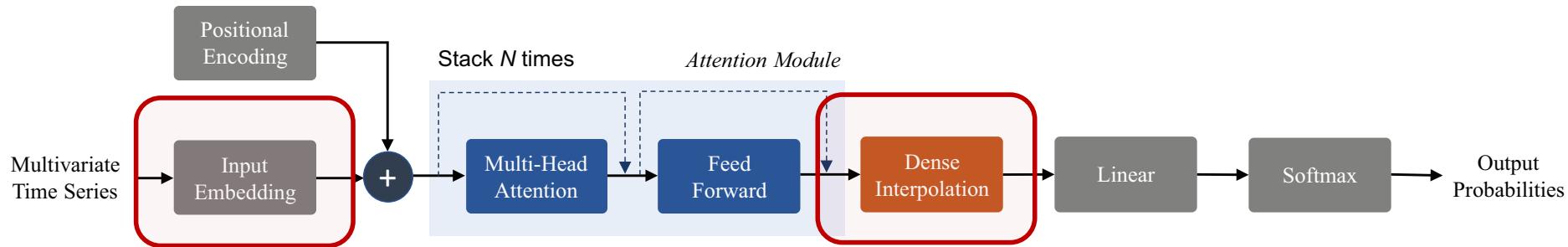
- Uses transformer model: **encoder-only** architecture
- Takes **multivariate time series** (input)
- Returns **class probability estimates** or **regression**
 - Depending on the **configuration** of the **last layer**



[Song2018] H. Song, D. Rajan, J.J. Thiagarajan, A. Spanias, “Attend and diagnose: clinical time-series analysis using attention models,” Proc. of Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18), 2018. Github: <https://github.com/khirotaka/SAnD/>

Attend and diagnose [Song2018]

- Only two differences with respect to transformer are
 - Input embedding
 - Dense interpolation



[Song2018] H. Song, D. Rajan, J.J. Thiagarajan, A. Spanias, “Attend and diagnose: clinical time-series analysis using attention models,” Proc. of Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18), 2018.

Input embedding

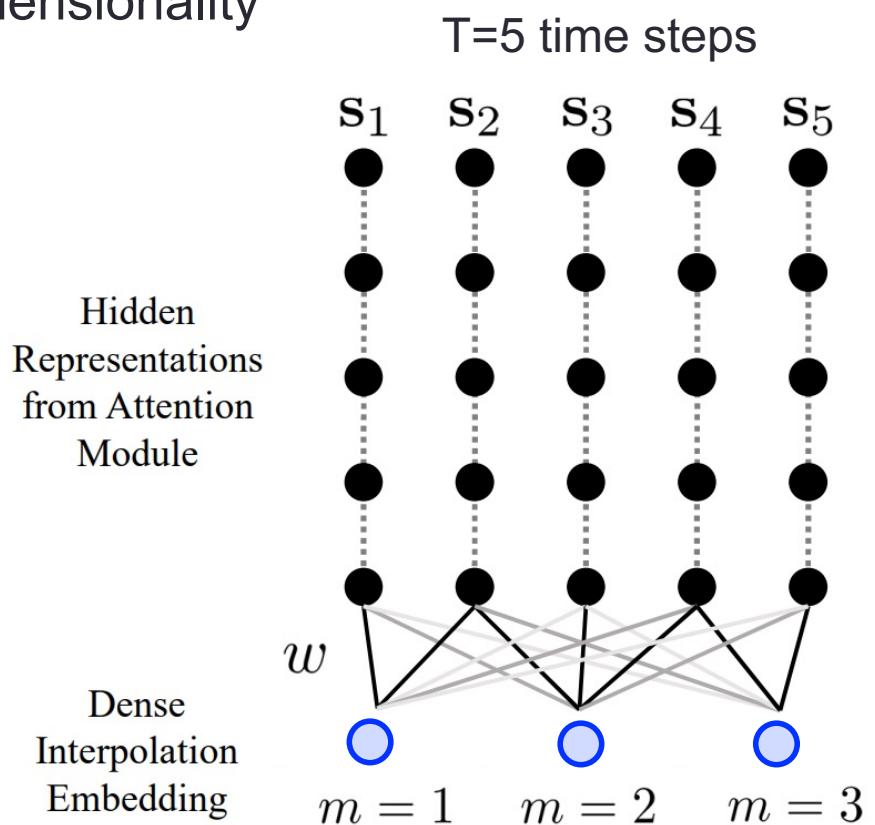
- T time steps
- Input vectors at time t are $\mathbf{x}_t \in \mathbb{R}^m$, with $t=1, \dots, T$
- Input embedding
 - Transform ([1D convolution](#)) from **input vectors** to **output vector**
 - Using a **kernel size of 1**, this is equivalent to projecting the input vector into a higher dimensional space of size **$d > m$**
 - Number of output channels $C_{\text{in}} = m$
 - Number of input channels $C_{\text{out}} = d$

$$\mathbf{y}_t[j] = \text{bias}[j] + \sum_{k=0}^{C_{\text{in}}-1} w(j, k) \star \mathbf{x}_t[k], \quad j = 0, \dots, C_{\text{out}} - 1$$

output vector	bias for out channel j	cross-correlation	convolution weights	input vector k-th element
-------------------------	----------------------------------	--------------------------	-------------------------------	-------------------------------------

Dense interpolation

- Interpolation of hidden vectors from the last attention block
 - Interpolation weights predefined
 - implemented to reduce the dimensionality



The speech transformer

SPEECH-TRANSFORMER: A NO-RECURRENCE SEQUENCE-TO-SEQUENCE MODEL FOR SPEECH RECOGNITION

Linhao Dong^{1,2}, Shuang Xu¹, Bo Xu¹

¹Institute of Automation, Chinese Academy of Sciences, China

²University of Chinese Academy of Sciences, China

{donglinhao2015, shuang.xu, xubo}@ia.ac.cn

ABSTRACT

Recurrent sequence-to-sequence models using encoder-decoder architecture have made great progress in speech recognition task. However, they suffer from the drawback of slow training speed because the internal recurrence limits the training parallelization. In this paper, we present the Speech-Transformer, a no-recurrence sequence-to-sequence model entirely relies on attention mechanisms to learn the positional dependencies, which can be trained faster with more efficiency. We also propose a 2D-Attention mechanism, which can jointly attend to the time and frequency axes of the 2-dimensional speech inputs, thus providing more expressive representations for the Speech-Transformer. Evaluated on the Wall Street Journal (WSJ) speech recognition dataset, our best model achieves competitive word error rate (WER) of 10.9%, while the whole training process only takes 1.2 days on 1 GPU, significantly faster than the published results of recurrent sequence-to-sequence models.

Index Terms— Speech Recognition, Sequence-to-Sequence, Attention, Transformer

1. INTRODUCTION

In speech recognition field, sequence-to-sequence (seq2seq) mod-

Recently, Vaswani et al. [17] proposed a no-recurrence sequence-to-sequence model, called the Transformer, which achieved state-of-the-art performance on WMT 2014 English-to-French translation task with markedly less training cost. Its fundamental module is self-attention, a mechanism relates all the position-pairs of a sequence to extract a more expressive sequence representation. Since the self-attention can draw the dependencies between different positions through the position-pair computation rather than the position-chain computation of RNNs, it just needs to be calculated once to obtain the transformed representation, rather than be computed one by one in RNNs. Therefore, the Transformer relying solely on attention mechanisms can be trained faster with more parallelization, which is exactly needed by the seq2seq models in automatic speech recognition (ASR).

In this paper, we successfully introduce the Transformer to ASR task and we term our model the Speech-Transformer, a new seq2seq model that transforms speech feature sequences to the corresponding character sequences. Additionally, we propose a 2D-Attention mechanism, which is inspired by the time-frequency LSTM in [18] but replaces the time-frequency recurrence with both the temporal and spectral dependencies captured by attention. By deploying the WSJ speech recognition dataset, we find our proposed 2D-Attention mechanism achieves better performance than the convolutional networks in [5] thus providing a more discriminated representation for

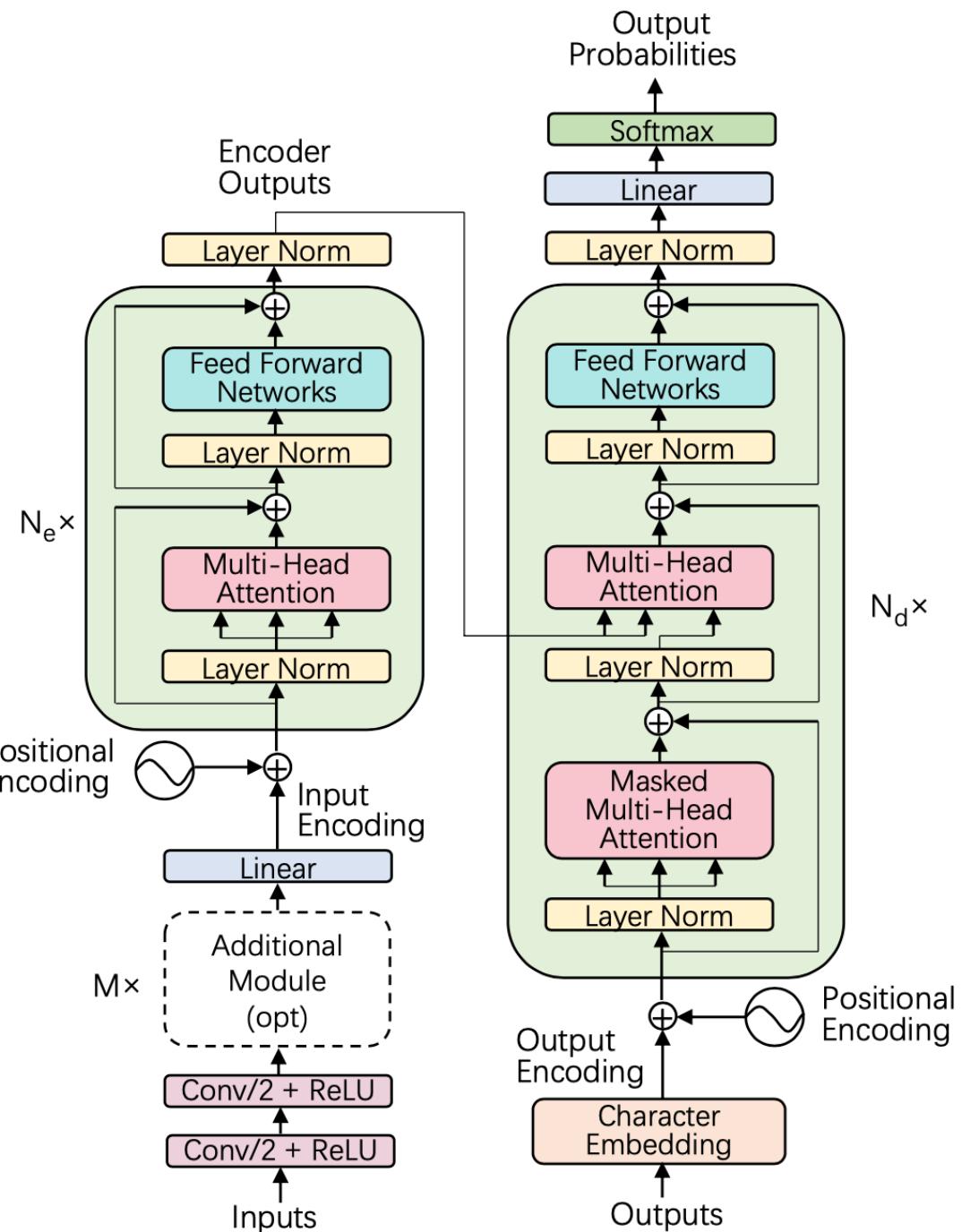
Architecture

Standard transformer model

Only differences are

- Input is a **spectrogram**
- **2D convolutions** used to process it (stride 2 along both time and frequency axes to reduce size and make it comparable with that of output)

[Dong2018] L. Dong, S. Xu, B. Xu, "Speech transformer: a no-recurrence sequence-to-sequence model for speech recognition," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.



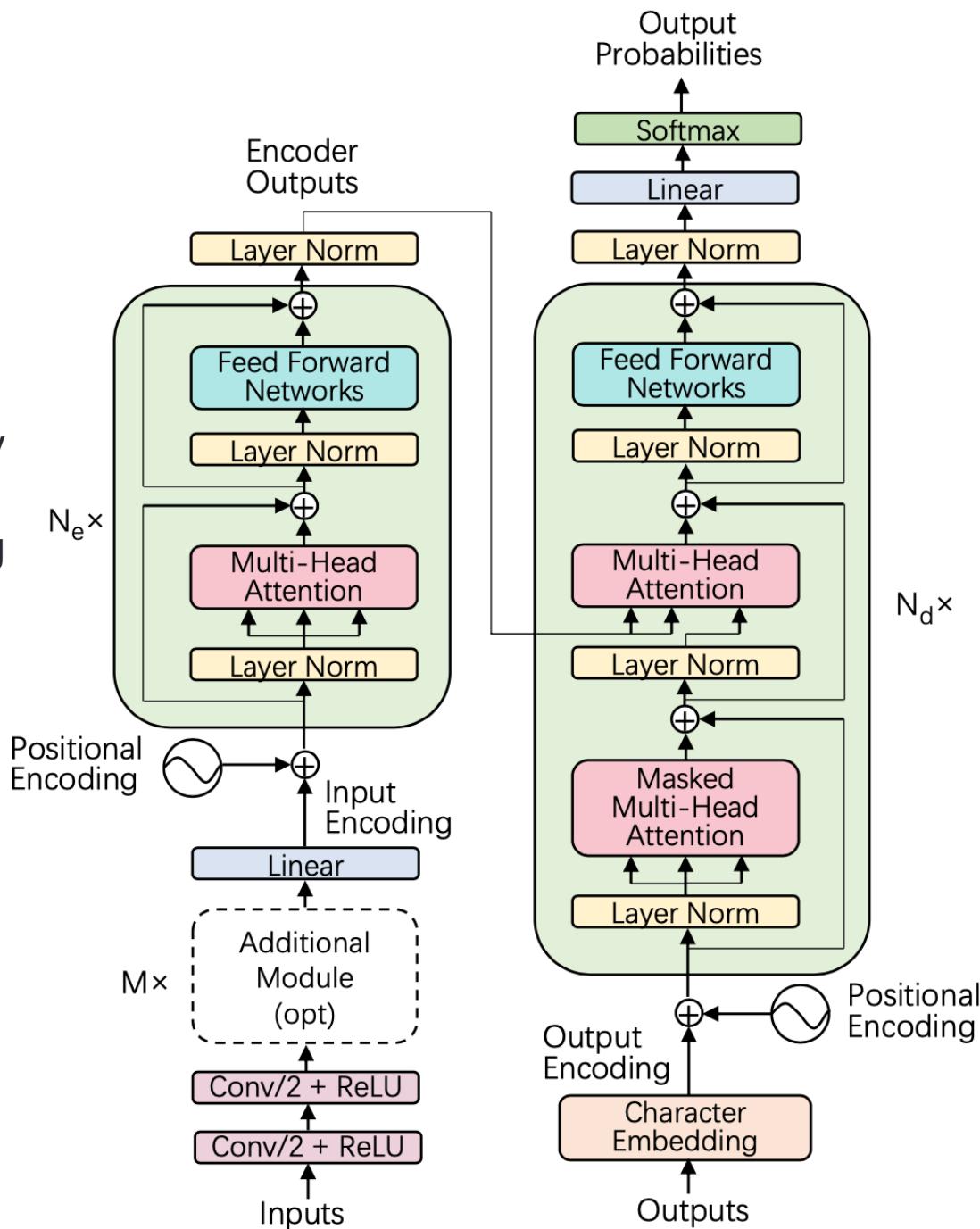
Architecture

Convolution: 3x3 with stride 2

Linear layer:

- Applied to the flattened conv output
- To obtain input encoding vectors (of the same size d_{model})

All the rest is identical to original transformer from [Vaswani2017]



Word error rate (WER) performance

Parameters

- N_e no. of encoder blocks
- N_d no. of decoder blocks
- d_{ff} inner dimension of feed-forward networks (FFN)
- $d_{model} = 256$ model dimension (size of vectors at input of FFN)
- $h=8$ no. of heads for the attention block

Model	N_e	N_d	d_{ff}	WER
6Enc6Dec (base model)	6	6	1024	12.20
12Enc6Dec-wide (big model)	12	6	2048	10.92
4Enc8Dec	4	8	1024	13.26
8Enc4Dec	8	4	1024	11.95
8Enc4Dec-wide	8	4	2048	11.56
10Enc5Dec-wide	10	5	2048	11.01

[Dong2018] L. Dong, S. Xu, B. Xu, “Speech transformer: a no-recurrence sequence-to-sequence model for speech recognition,” *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

INTERSPEECH 2020

October 25–29, 2020, Shanghai, China



Conformer: Convolution-augmented Transformer for Speech Recognition

*Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han,
Shibo Wang, Zhengdong Zhang, Yonghui Wu, Ruoming Pang*

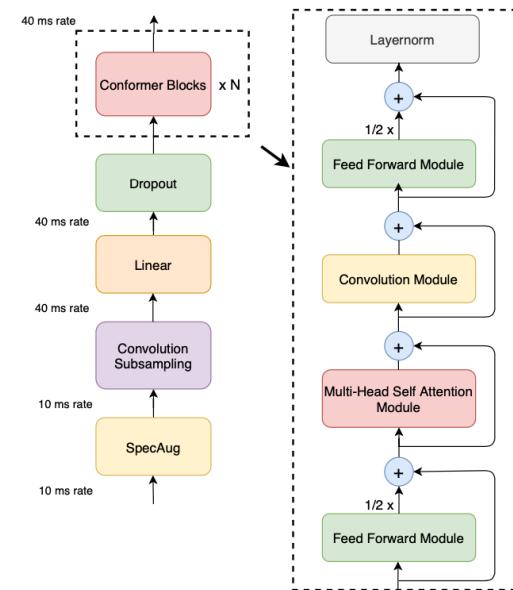
Google Inc.

{anmolgulati, jamesqin, chungchengc, nikip, ngyuzh, jiahuiyu, weihan, shibow, zhangzd,
yonghui, rpang}@google.com

Abstract

Recently Transformer and Convolution neural network (CNN) based models have shown promising results in Automatic Speech Recognition (ASR), outperforming Recurrent neural networks (RNNs). Transformer models are good at capturing content-based global interactions, while CNNs exploit local features effectively. In this work, we achieve the best of both worlds by studying how to combine convolution neural networks and transformers to model both local and global dependencies of an audio sequence in a parameter-efficient way. To this regard, we propose the convolution-augmented transformer for speech recognition, named *Conformer*. *Conformer* significantly outperforms the previous Transformer and CNN based models achieving state-of-the-art accuracies. On the widely used LibriSpeech benchmark, our model achieves WER of 2.1%/4.3% without using a language model and 1.9%/3.9% with an external language model on test/testother. We also observe competitive performance of 2.7%/6.3% with a small model of only 10M parameters.

Index Terms: speech recognition, attention, convolutional neu-



Conformer

- **Input**
 - Spectrograms: contain features along **time** and **frequency** dimensions
- **Self-attention (transformer)**
 - good to extract large scale temporal features across time samples
 - does not track local (fine grained features) at each time sample
- **Solution**
 - use convolution operation to track local features
 - applied after self-attention and before the following FF module

[Gulati2020] A. Gulati, C-C. Chiu, J. Qin, J. Yu, N. Parmar, R. Pang, S. Wang, W. Han, Y. Wu, Y. Zhang, Z. Zhang, "Conformer: Convolution-augmented Transformer for Speech Recognition," 21st Annual Conference of the International Speech Communication Association (INTERSPEECH), 2020.
[1097 Citations, Dec. 2022]

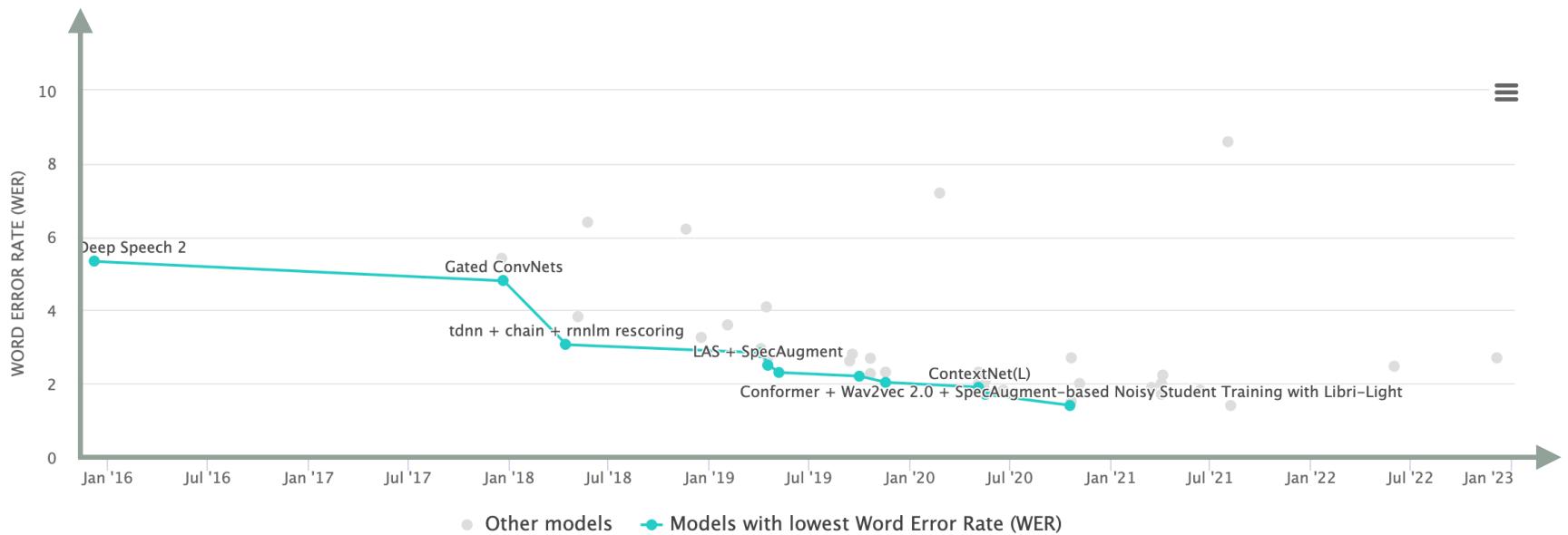
Models and data

- Evaluation on **LibriSpeech** dataset
 - 970 hours of labeled speech data (<http://www.openslr.org/12>)
 - **Cepstral** coeff: 80-filterbanks, windows of 25ms, with 10ms overlap
- Three models were identified
 - Small (10 M pars), medium (30 M), large (118 M)
 - Changing network depth, model dimension, no. of transformer heads
- Subsampling
 - Two 3x3 CNN models with srtride 2 → 4x temporal subsampling
 - This reduces the sampling period from 10ms to 40ms
- Dropout
 - Applied as *regularizer* in each residual unit of the comformer module (at the output of each module, before it is added to the residual input)

Performance

Method	#Params (M)	WER Without LM		WER With LM	
		testclean	testother	testclean	testother
Hybrid					
Transformer [33]	-	-	-	2.26	4.85
CTC					
QuartzNet [9]	19	3.90	11.28	2.69	7.25
LAS					
Transformer [34]	270	2.89	6.98	2.33	5.17
Transformer [19]	-	2.2	5.6	2.6	5.7
LSTM	360	2.6	6.0	2.2	5.2
Transducer					
Transformer [7]	139	2.4	5.6	2.0	4.6
ContextNet(S) [10]	10.8	2.9	7.0	2.3	5.5
ContextNet(M) [10]	31.4	2.4	5.4	2.0	4.5
ContextNet(L) [10]	112.7	2.1	4.6	1.9	4.1
Conformer (Ours)					
Conformer(S)	10.3	2.7	6.3	2.1	5.0
Conformer(M)	30.7	2.3	5.0	2.0	4.3
Conformer(L)	118.8	2.1	4.3	1.9	3.9

Plenty of models are being proposed...



<https://paperswithcode.com/sota/speech-recognition-on-librispeech-test-clean>

Code completion



**GitHub
Copilot**

Code completion

arXiv:2107.03374v2 [cs.LG] 14 Jul 2021

Evaluating Large Language Models Trained on Code

Mark Chen^{*1} Jerry Tworek^{*1} Heewoo Jun^{*1} Qiming Yuan^{*1} Henrique Ponde de Oliveira Pinto^{*1}
Jared Kaplan^{*2} Harri Edwards¹ Yuri Burda¹ Nicholas Joseph² Greg Brockman¹ Alex Ray¹ Raul Puri¹
Gretchen Krueger¹ Michael Petrov¹ Heidy Khlaaf³ Girish Sastry¹ Pamela Mishkin¹ Brooke Chan¹
Scott Gray¹ Nick Ryder¹ Mikhail Pavlov¹ Alethea Power¹ Lukasz Kaiser¹ Mohammad Bavarian¹
Clemens Winter¹ Philippe Tillet¹ Felipe Petroski Such¹ Dave Cummings¹ Matthias Plappert¹
Fotios Chantzis¹ Elizabeth Barnes¹ Ariel Herbert-Voss¹ William Hebgen Guss¹ Alex Nichol¹ Alex Paino¹
Nikolas Tezak¹ Jie Tang¹ Igor Babuschkin¹ Suchir Balaji¹ Shantanu Jain¹ William Saunders¹
Christopher Hesse¹ Andrew N. Carr¹ Jan Leike¹ Josh Achiam¹ Vedant Misra¹ Evan Morikawa¹
Alec Radford¹ Matthew Knight¹ Miles Brundage¹ Mira Murati¹ Katie Mayer¹ Peter Welinder¹
Bob McGrew¹ Dario Amodei² Sam McCandlish² Ilya Sutskever¹ Wojciech Zaremba¹

Abstract

We introduce Codex, a GPT language model fine-tuned on publicly available code from GitHub, and study its Python code-writing capabilities. A distinct production version of Codex powers GitHub Copilot. On HumanEval, a new evaluation set we release to measure functional correctness for synthesizing programs from docstrings, our model solves 28.8% of the problems, while GPT-3 solves 0% and GPT-J solves 11.4%. Furthermore, we find that repeated sampling from the model is a surprisingly effective strategy for producing working solutions to difficult prompts. Using this method, we solve 70.2% of our problems with 100 samples per problem. Careful investigation of our model reveals its limitations, including difficulty with docstrings describing long chains of operations and with binding operations to variables. Finally, we discuss the potential broader impacts of deploying powerful code generation technologies, covering safety, security, and economics.

1. Introduction

Scalable sequence prediction models (Graves, 2014; Vaswani et al., 2017; Child et al., 2019) have become a general-purpose method for generation and representation learning in many domains, including natural language processing (Mikolov et al., 2013; Sutskever et al., 2014; Dai & Le, 2015; Peters et al., 2018; Radford et al., 2018; Devlin et al., 2018), computer vision (Van Oord et al., 2016; Menick & Kalchbrenner, 2018; Chen et al., 2020; Bao et al., 2021), audio and speech processing (Oord et al., 2016; 2018; Dhariwal et al., 2020; Baevski et al., 2020), biology (Alley et al., 2019; Rives et al., 2021), and even across multiple modalities (Das et al., 2017; Lu et al., 2019; Ramesh et al., 2021; Zellers et al., 2021). More recently, language models have also fueled progress towards the longstanding challenge of program synthesis (Simon, 1963; Manna & Waldinger, 1971), spurred by the presence of code in large datasets (Husain et al., 2019; Gao et al., 2020) and the resulting programming capabilities of language models trained on these datasets (Wang & Komatsu, 2021). Popular language modeling objectives like masked language modeling (Devlin et al., 2018) and span prediction (Raffel et al., 2020) have also been adapted to train their programming counterparts CodeBERT (Feng et al., 2020) and PyMTS (Clement et al., 2020).



Generating code from text description

```

def incr_list(l: list):
    """Return list with elements incremented by 1.
    >>> incr_list([1, 2, 3])
    [2, 3, 4]
    >>> incr_list([5, 3, 5, 2, 3, 3, 9, 0, 123])
    [6, 4, 6, 3, 4, 4, 10, 1, 124]
    """
    return [i + 1 for i in l]

def solution(lst):
    """Given a non-empty list of integers, return the sum of all of the odd elements
    that are in even positions.

    Examples
    solution([5, 8, 7, 1]) =>12
    solution([3, 3, 3, 3, 3]) =>9
    solution([30, 13, 24, 321]) =>0
    """
    return sum(lst[i] for i in range(0,len(lst)) if i % 2 == 0 and lst[i] % 2 == 1)

def encode_cyclic(s: str):
    """
    returns encoded string by cycling groups of three characters.
    """
    # split string to groups. Each of length 3.
    groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((len(s) + 2) // 3)]
    # cycle elements in each group. Unless group has fewer elements than 3.
    groups = [(group[1:] + group[0]) if len(group) == 3 else group for group in groups]
    return "".join(groups)

def decode_cyclic(s: str):
    """
    takes as input string encoded with encode_cyclic function. Returns decoded string.
    """
    # split string to groups. Each of length 3.
    groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((len(s) + 2) // 3)]
    # cycle elements in each group.
    groups = [(group[-1] + group[:-1]) if len(group) == 3 else group for group in groups]
    return "".join(groups)

```

Multimodal

DeepMind
2022-5-19

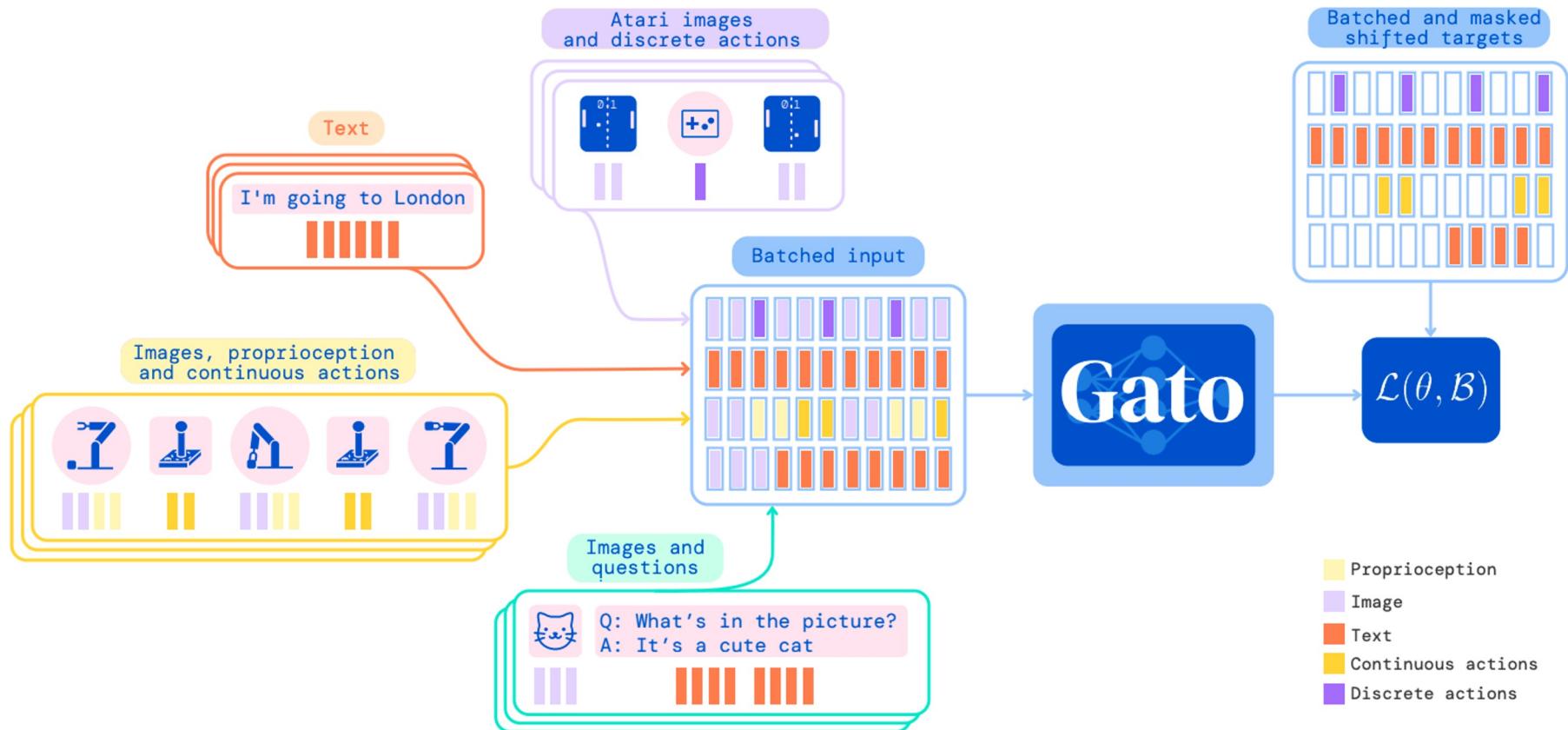
A Generalist Agent

Scott Reed^{*,†}, Konrad Żolna^{*}, Emilio Parisotto^{*}, Sergio Gómez Colmenarejo[†], Alexander Novikov,
 Gabriel Barth-Maron, Mai Giménez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles,
 Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals,
 Mahyar Bordbar and Nando de Freitas[†]

^{*}Equal contributions, [†]Equal senior contributions, All authors are affiliated with DeepMind

Inspired by progress in large-scale language modeling, we apply a similar approach towards building a single generalist agent beyond the realm of text outputs. The agent, which we refer to as Gato, works as a multi-modal, multi-task, multi-embodiment generalist policy. The same network with the same weights can play Atari, caption images, chat, stack blocks with a real robot arm and much more, deciding based on its context whether to output text, joint torques, button presses, or other tokens. In this report we describe the model and the data, and document the current capabilities of Gato.

Gato



S. Reed et al. “[A Generalist Agent](#)”. [Blog]

Repo for models and datasets

<https://huggingface.co/models>

The screenshot shows the Hugging Face website interface. At the top, there is a navigation bar with a logo, a search bar, and links for Models, Datasets, Spaces, Docs, Solutions, Pricing, Log In, and Sign Up. Below the navigation bar, there are sections for Tasks, Libraries, Datasets, and Languages. The main content area is titled "Models" and shows 97,178 results. A "Sort: Most Downloads" button is visible. Below this, several model cards are listed:

- bert-base-uncased**: Updated 21 days ago, 19M downloads, 356 stars.
- gpt2**: Updated 14 days ago, 10.3M downloads, 336 stars.
- distilbert-base-uncased-finetuned-sst-2-english**: Updated 2 days ago, 8.62M downloads, 118 stars.
- openai/clip-vit-large-patch14**: Updated Oct 4, 7.74M downloads, 98 stars.
- distilbert-base-uncased**: Updated 21 days ago, 6.9M downloads, 104 stars.
- bert-base-multilingual-cased**: Updated 21 days ago, 6.73M downloads, 70 stars.
- Jean-Baptiste/camembert-ner**: Updated Oct 13, 6.02M downloads, 46 stars.

At the bottom left, there are buttons for English, French, and Spanish.

Additional resources

[Kamath2022] Uday Kamath, Kenneth L. Graham, Wael Emara, “Transformers for Machine Learning: a Deep Dive,” CRC Press, 2022.

[Ahmed2022] S. Ahmed, I. E. Nielsen, A. Tripathi, S. Siddiqui, G. Rasool, R. P. Ramachandran, “Transformers in Time Series Analysis: a Tutorial,” arXiv:2205.01138, 28 April 2022.

[Cristina2022] Stefania Cristina “[Training the transformer model](#),” (blog and book), Nov. 16, 2022.

APPENDIX A: DERIVATIVE OF SOFTMAX

Derivative of softmax (1/4)

- For softmax output node i , we have

$$y_i = \frac{\exp(a_i)}{\sum_k \exp(a_k)}$$

$\frac{\partial y_i}{\partial a_j} = ?$

- Derivative of a quotient:

$$f(x) = \frac{g(x)}{h(x)} \rightarrow f'(x) = \frac{g'(x)h(x) - h'(x)g(x)}{h(x)^2}$$

- We use:

$$g(x) = \exp(a_i), h(x) = \sum_k \exp(a_k)$$

Derivative of softmax (2/4)

if $i = j$

$$\begin{aligned}
 \frac{\partial y_i}{\partial a_j} &= \frac{\exp(a_i) \sum_k \exp(a_k) - \exp(a_j) \exp(a_i)}{\left(\sum_k \exp(a_k)\right)^2} = \\
 &= \frac{\exp(a_i) (\sum_k \exp(a_k) - \exp(a_j))}{\left(\sum_k \exp(a_k)\right)^2} = \\
 &= \frac{\exp(a_i)}{\sum_k \exp(a_k)} \times \frac{(\sum_k \exp(a_k) - \exp(a_j))}{\sum_k \exp(a_k)} = \\
 &= y_i(1 - y_j) = y_i(1 - y_i) \quad (\text{as } i = j)
 \end{aligned}$$

Derivative of softmax (3/4)

if $i \neq j$

$$\begin{aligned}\frac{\partial y_i}{\partial a_j} &= \frac{0 - \exp(a_j) \exp(a_i)}{\left(\sum_k \exp(a_k)\right)^2} = \\ &= \frac{-\exp(a_j)}{\left(\sum_k \exp(a_k)\right)} \times \frac{\exp(a_i)}{\left(\sum_k \exp(a_k)\right)} = \\ &= -y_j y_i\end{aligned}$$

Derivative of softmax (4/4)

- Thus, we get:

$$\frac{\partial y_i}{\partial a_j} = \begin{cases} y_i(1 - y_i) & i = j \\ -y_j y_i & i \neq j \end{cases}$$

- Compactly, using the **Kronecker delta**

$\delta_{ij} = 1$ if $i = j$, $\delta_{ij} = 0$ otherwise

$$\frac{\partial y_i}{\partial a_j} = y_i(\delta_{ij} - y_j)$$

ADVANCED NEURAL NETWORK DESIGNS: SELF-ATTENTION & TRANSFORMERS

Michele Rossi

michele.rossi@unipd.it

Dept. of Information Engineering
University of Padova, IT

