LVC 1: Attention Mechanism and Transformer Models

Natural Language Processing with Large Language Models

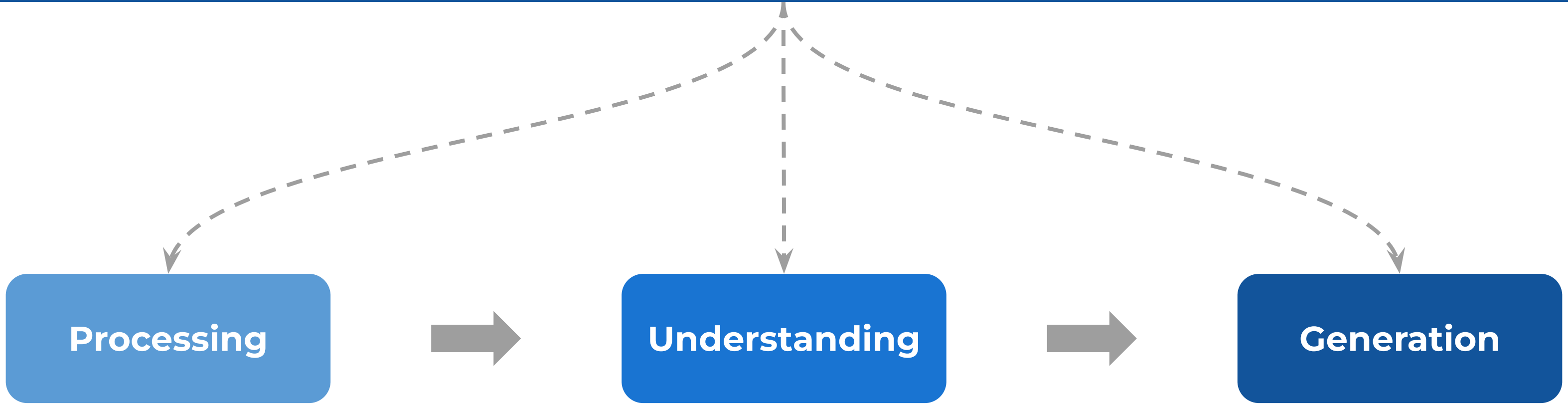
Agenda

- Introduction to **Natural Language Processing**
- Introduction to **Sequential Learning**
- **Attention Mechanism**
- **Transformer Models**

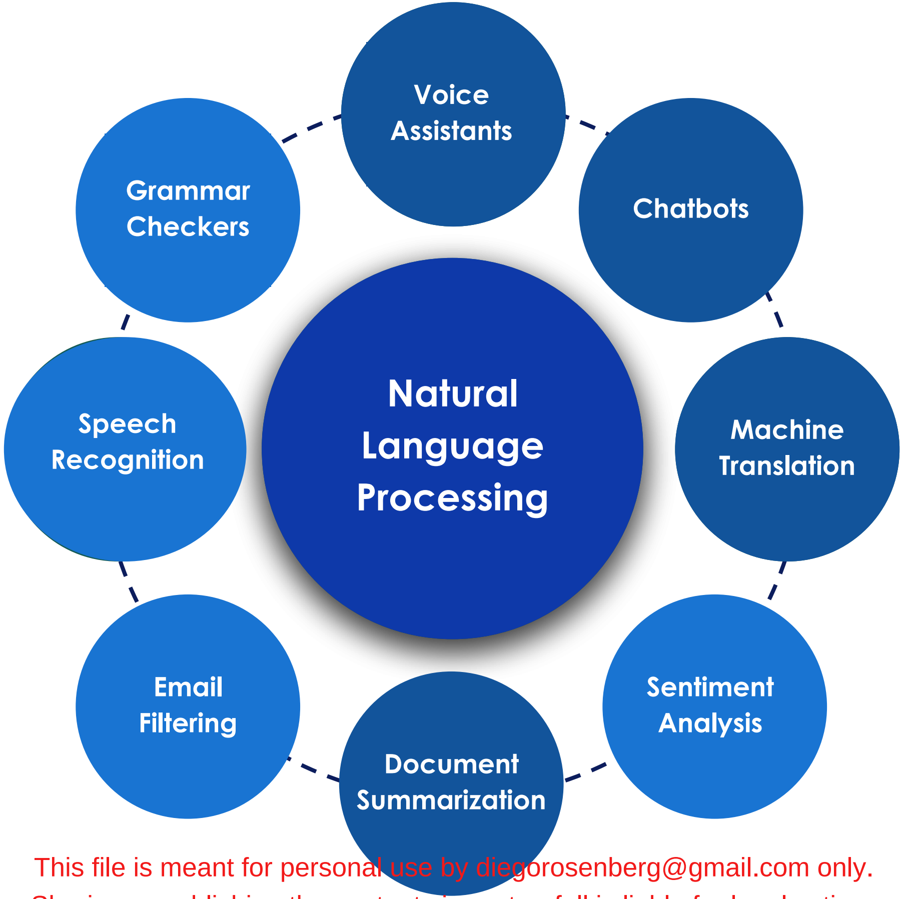
Introduction to **Natural Language Processing**

Introduction to Natural Language Processing

Natural Language Processing (NLP) is a branch of artificial intelligence (AI) that deals with the **interaction between machines and human languages**, with an aim to **automate** the **reading**, **interpretation** and **understanding** of human languages, also called natural language.



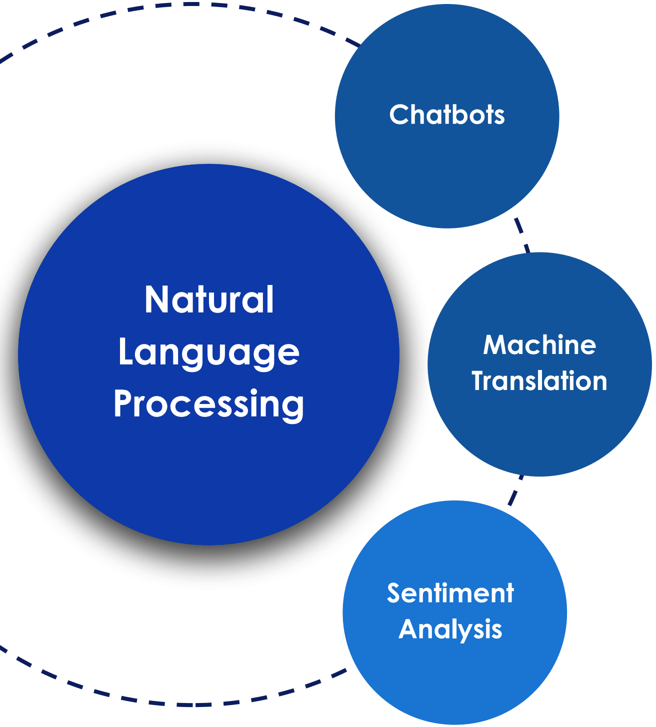
Applications of Natural Language Processing



Introduction to **Sequential Learning**

Sequential Data

Data where **order matters**

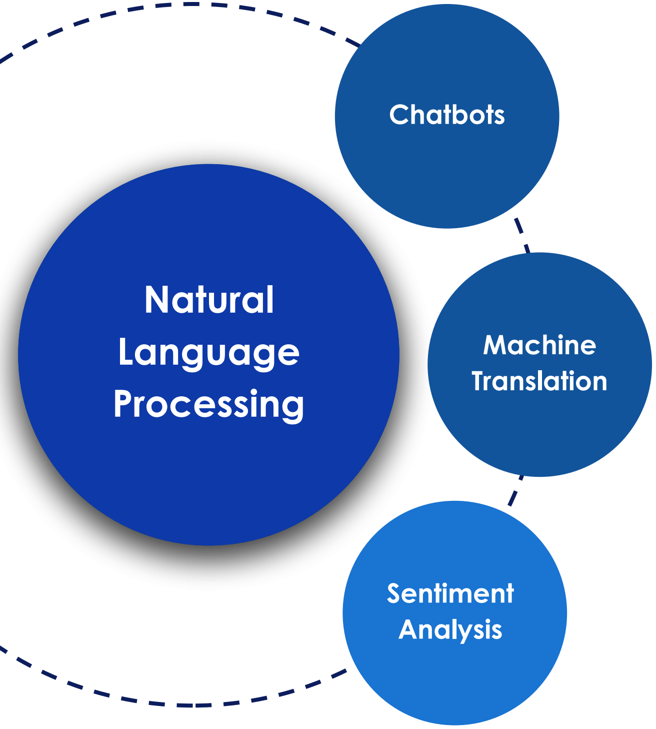


"What time is it now?" => "It is 8:00 pm."

"The cat sat on the wall." => "El gato se sentó en la pared."

"The movie was fun, brisk and imaginative" => **Positive**

Sequential Data



Data where **order matters**

“What is now time it?” => “Did you mean ‘*What time is it now?*’”

Output changes

“The wall sat on the cat.” => ”La pared se sentó sobre el gato.”

Meaning changes

“The imaginative was brisk, fun and movie” => ?????

Difficult to generate output

Sequential Learning

Sequential learning refers to a type of learning where a machine learning (ML) **model learns from sequential** (ordered) **data**

Example: The model has to learn to complete a sentence

I love eating pizza with __________ .

🟩 chilli flakes | 🟩 my friend | 🟥 sand

A model needs to **learn** the patterns in the **ordered data well** to make the right predictions.

I love playing with __________ .

🟥 chilli flakes | 🟩 my friend | 🟩 sand

ANNs for Sequential Learning

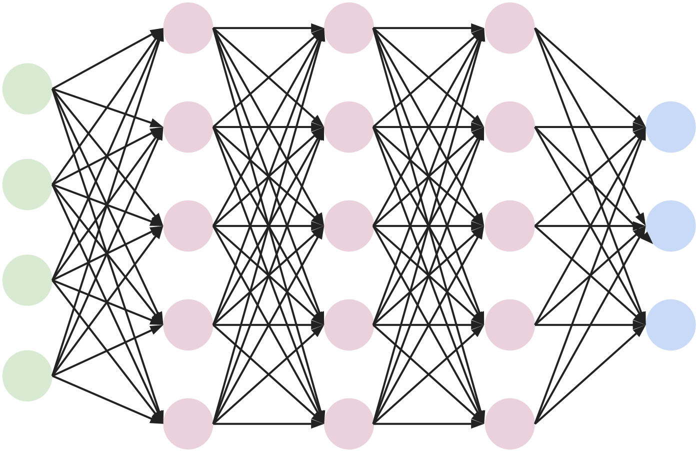
We can train artificial neural networks (ANNs) for sequential learning

I

love

playing

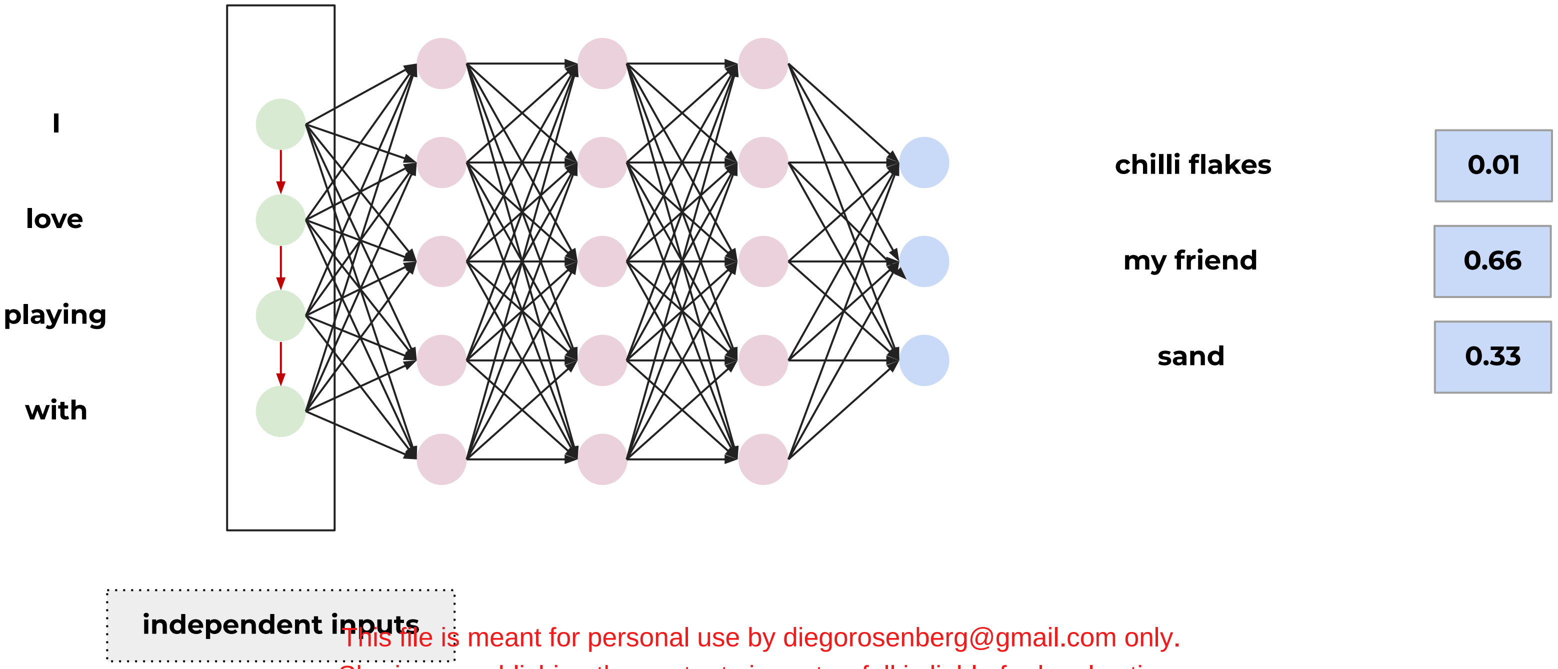
with



chilli flakes	0.01
my friend	0.66
sand	0.33

ANNs for Sequential Learning - Limitations

Each input is treated **independently** ; **no** way to maintain the **order**



ANNs for Sequential Learning - Limitations

Cannot accommodate **inputs** of **different length**

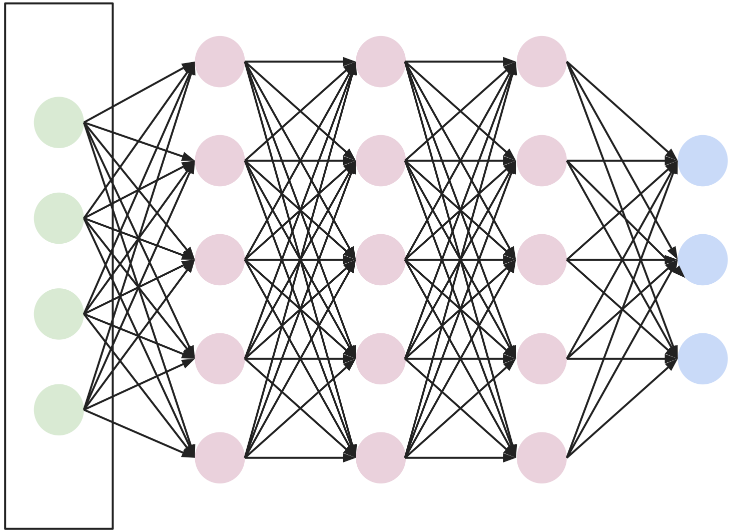
I

love

eating

pizza

with



chilli flakes

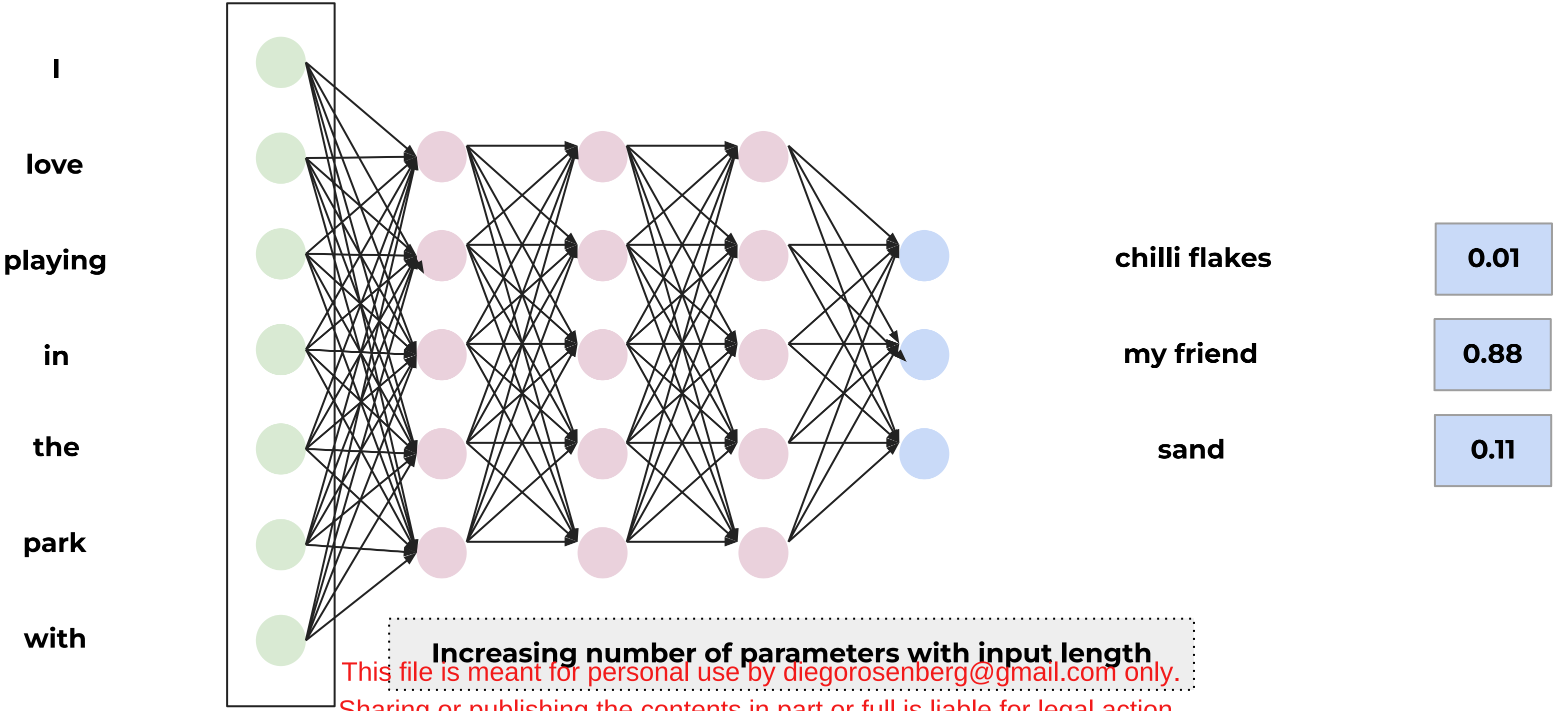
my friend

sand

fixed input length

ANNs for Sequential Learning - Limitations

The number of **parameters** to learn **increases** with **input length** ; more **computational cost**



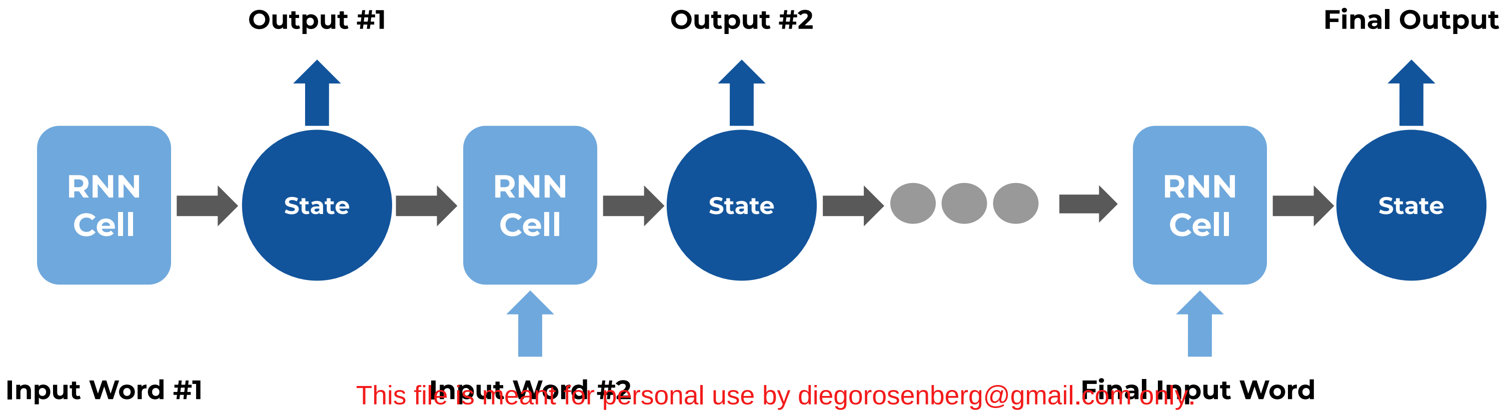
Increasing number of parameters with input length

RNNs for Sequential Learning

Recurrent Neural Networks (RNNs) overcome the problems encountered by ANNs

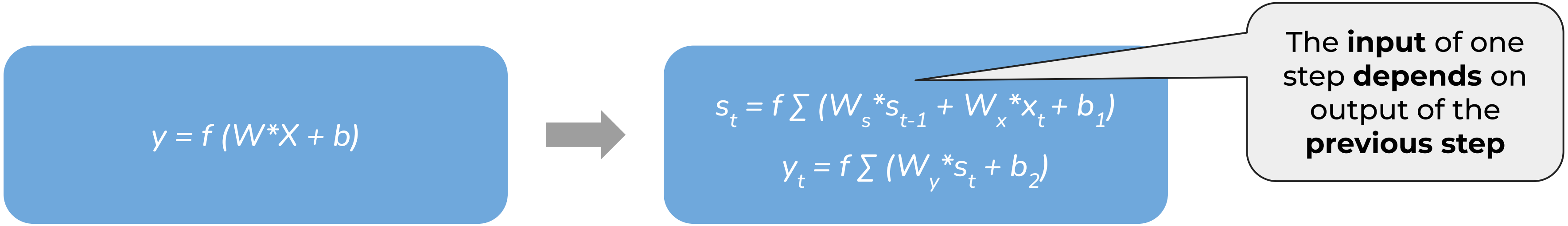
RNNs use **‘modified’ cells** that use a **step-by-step approach** for making predictions

A **‘state’** computed at **each step** is used as an **input** to the **next step**



RNNs for Sequential Learning

The **‘modified’ cells in RNNs** maintain a **hidden state** that implements a **form of memory**



$y = f\ (W*X + b)$

$s_t = f \sum (W_s * s_{t-1} + W_x * x_t + b_1)$

$y_t = f \sum (W_y * s_t + b_2)$

The **input** of one step **depends** on output of the **previous step**

ANN Cell

RNN Cell

Parameters (weights) are **shared** (same) **across all time steps**, so **fewer parameters to learn**

As **same weights** are used at every time step, so the **length of the input doesn’t matter ;** we just **create multiple copies of the same network** and execute them at each time step

Encoder-Decoder Architecture

Consider the example of **machine translation**

“The cat sat on the wall.” => ”El gato se sentó en la pared.”

We want to **predict a ‘sequence’** using the ‘learning’ **from another ‘sequence’**

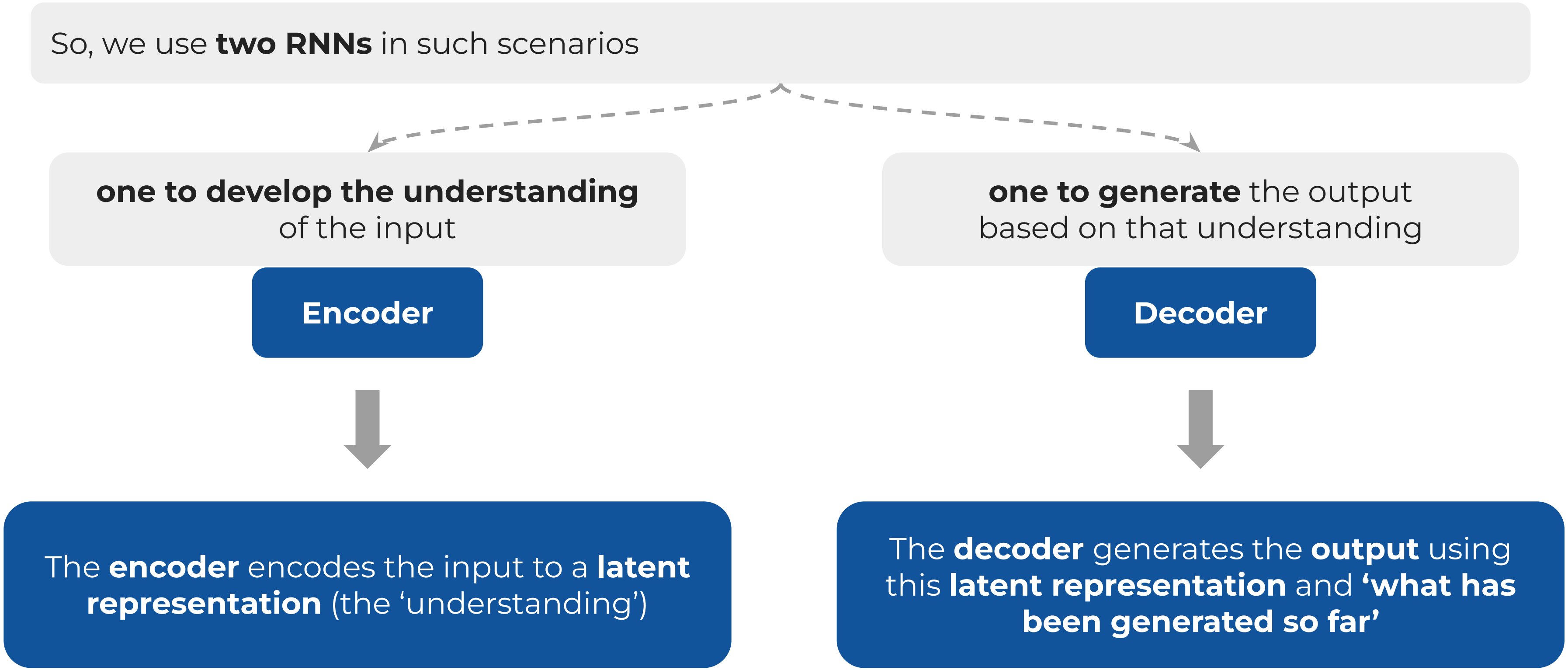
This is known as **sequence-to-sequence learning**

The model has to **first develop an understanding** of the input

Then it has to **generate** the output **based on this understanding**

In practice, using one RNN for such tasks doesn’t yield good results

Encoder-Decoder Architecture



So, we use **two RNNs** in such scenarios

one to develop the understanding of the input

Encoder

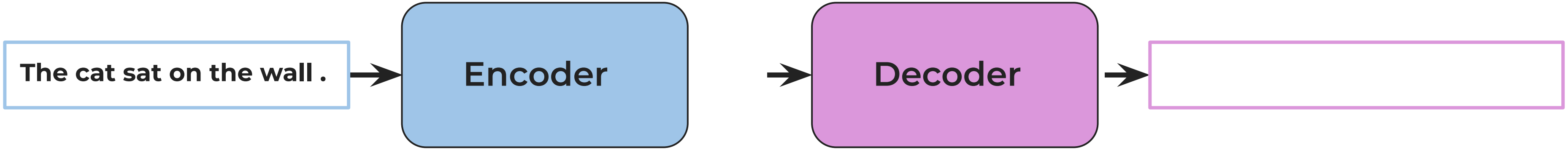
The **encoder** encodes the input to a **latent representation** (the ‘understanding’)

one to generate the output based on that understanding

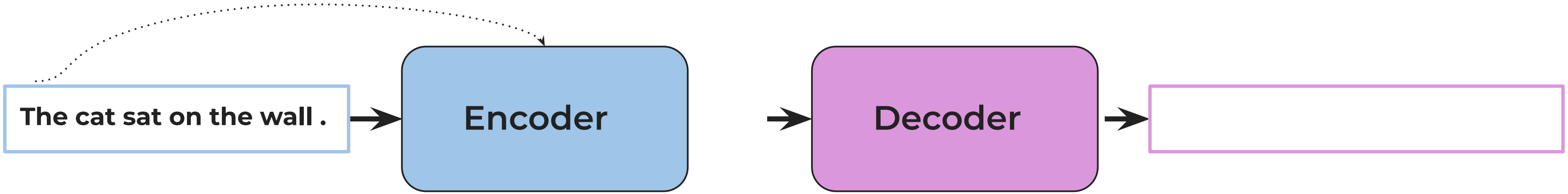
Decoder

The **decoder** generates the **output** using this **latent representation** and **‘what has been generated so far’**

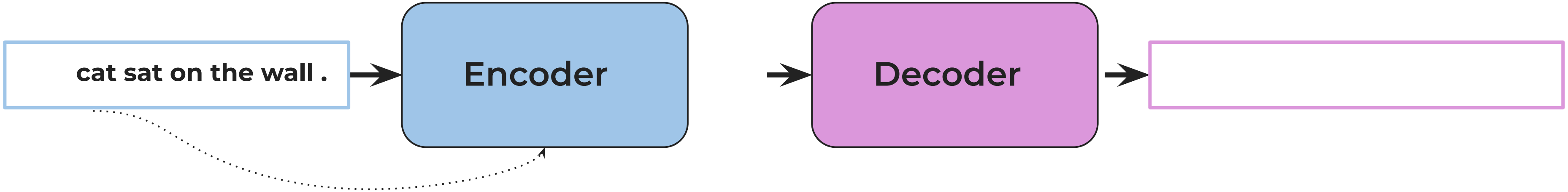
Encoder-Decoder Architecture - Example



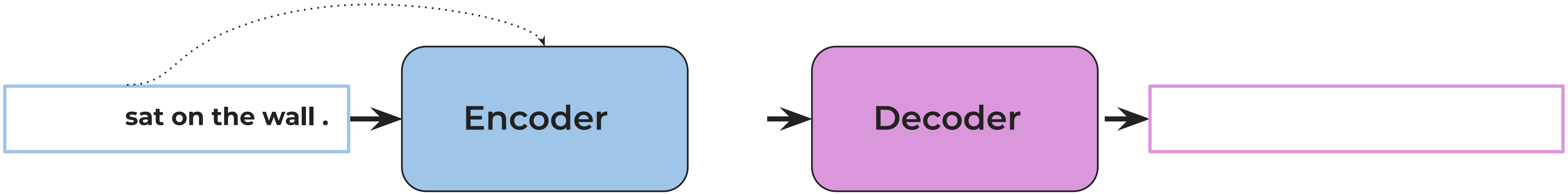
Encoder-Decoder Architecture - Example



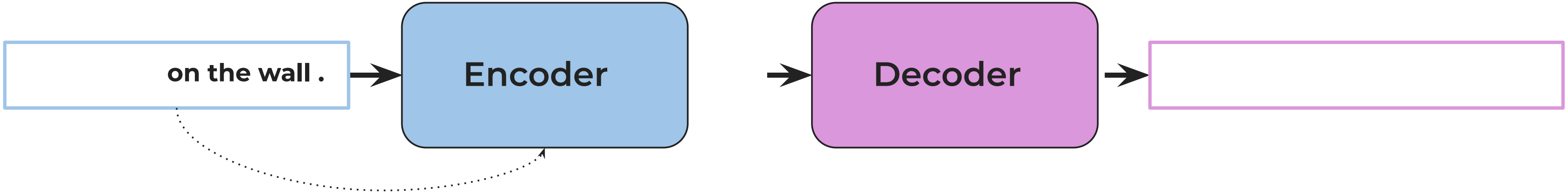
Encoder-Decoder Architecture - Example



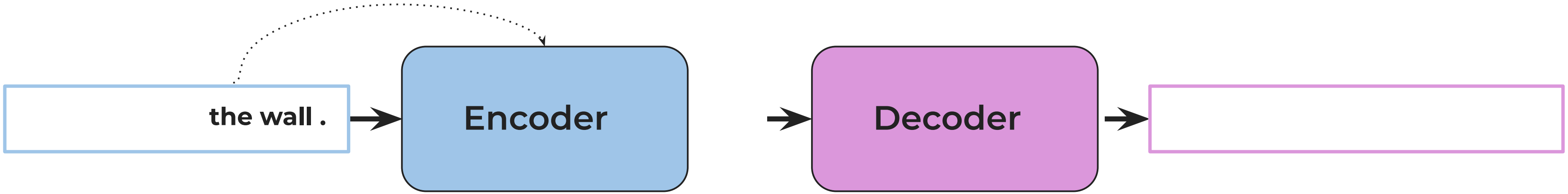
Encoder-Decoder Architecture - Example



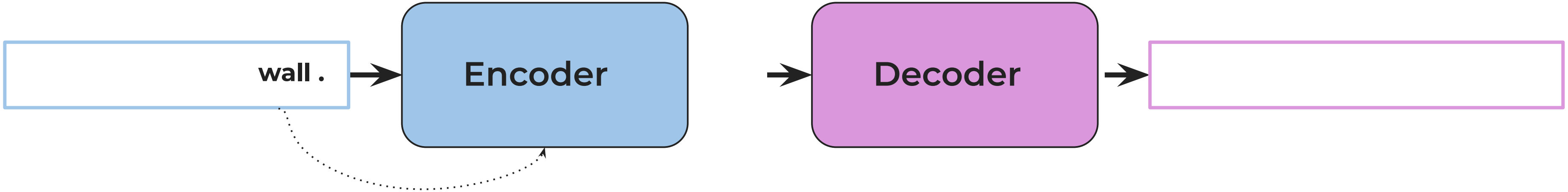
Encoder-Decoder Architecture - Example



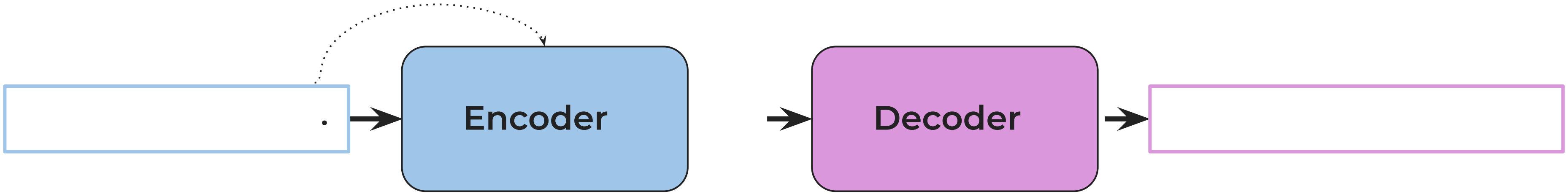
Encoder-Decoder Architecture - Example



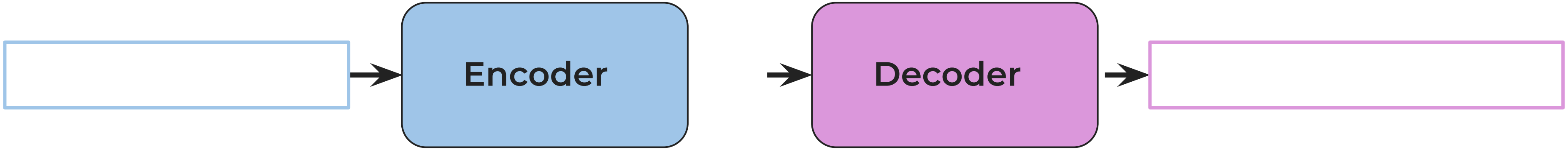
Encoder-Decoder Architecture - Example



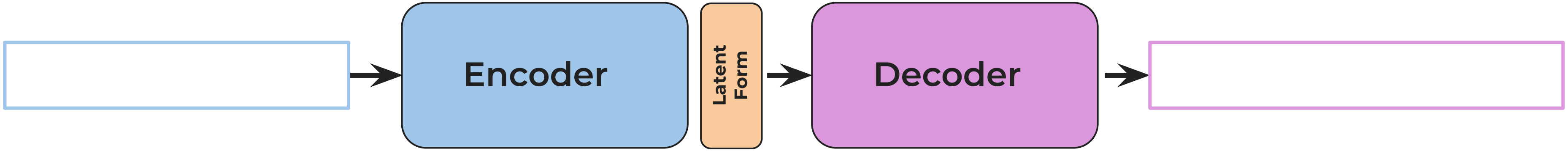
Encoder-Decoder Architecture - Example



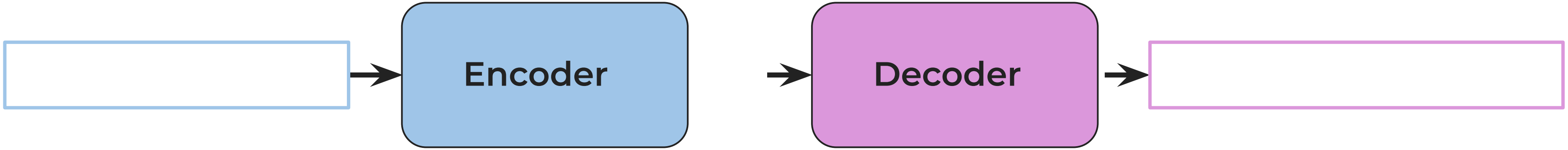
Encoder-Decoder Architecture - Example



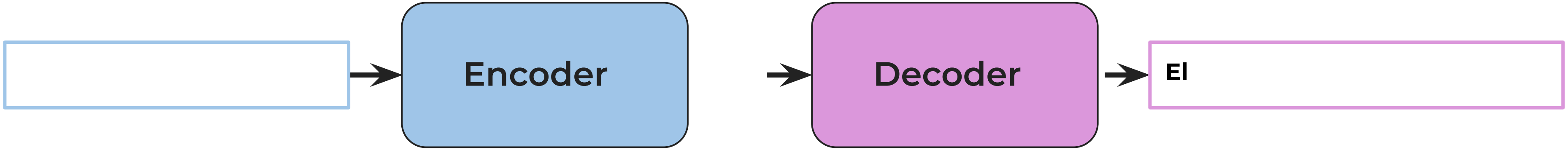
Encoder-Decoder Architecture - Example



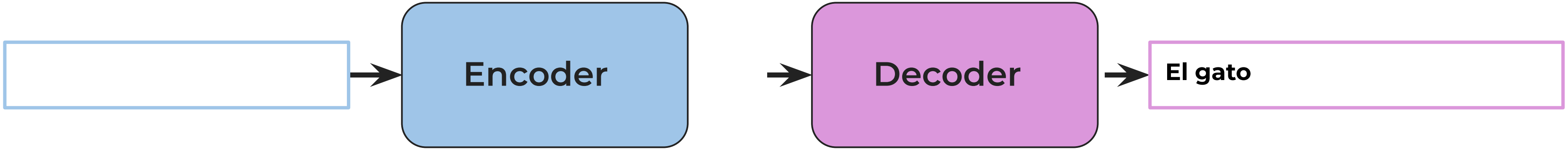
Encoder-Decoder Architecture - Example



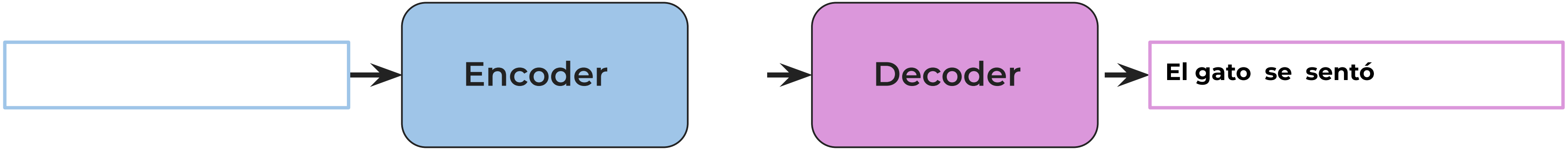
Encoder-Decoder Architecture - Example



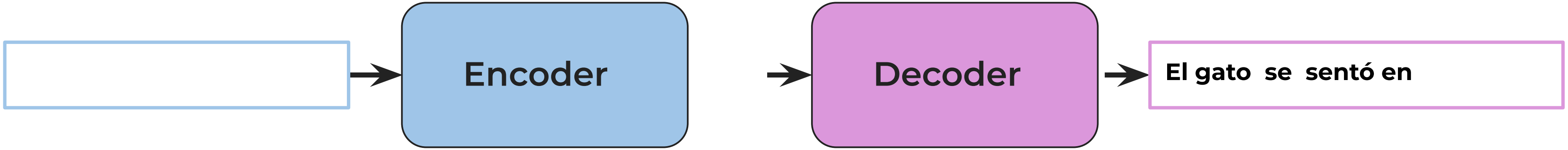
Encoder-Decoder Architecture - Example



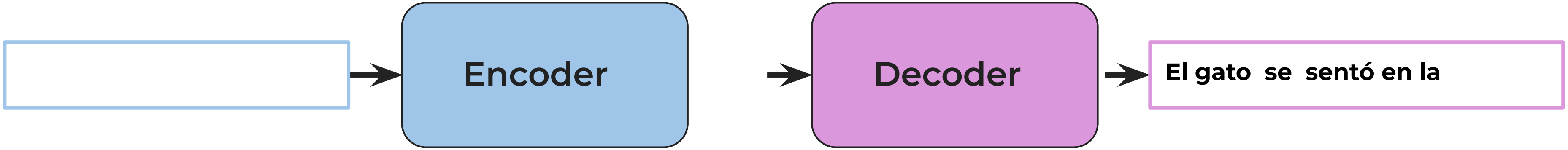
Encoder-Decoder Architecture - Example



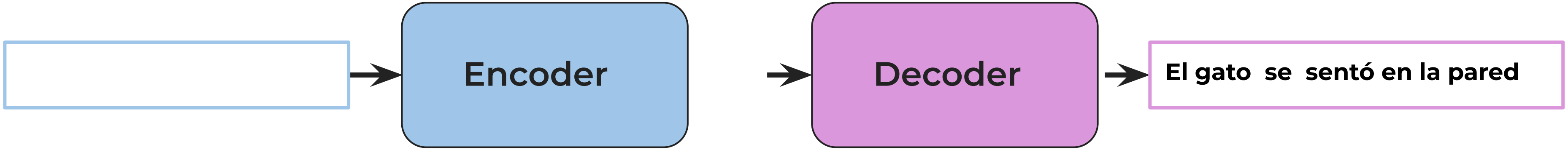
Encoder-Decoder Architecture - Example



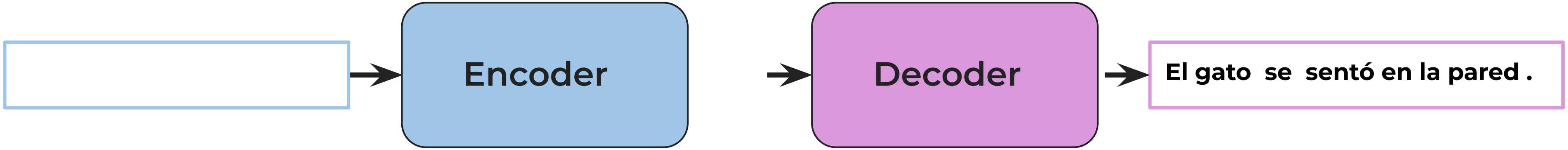
Encoder-Decoder Architecture - Example



Encoder-Decoder Architecture - Example



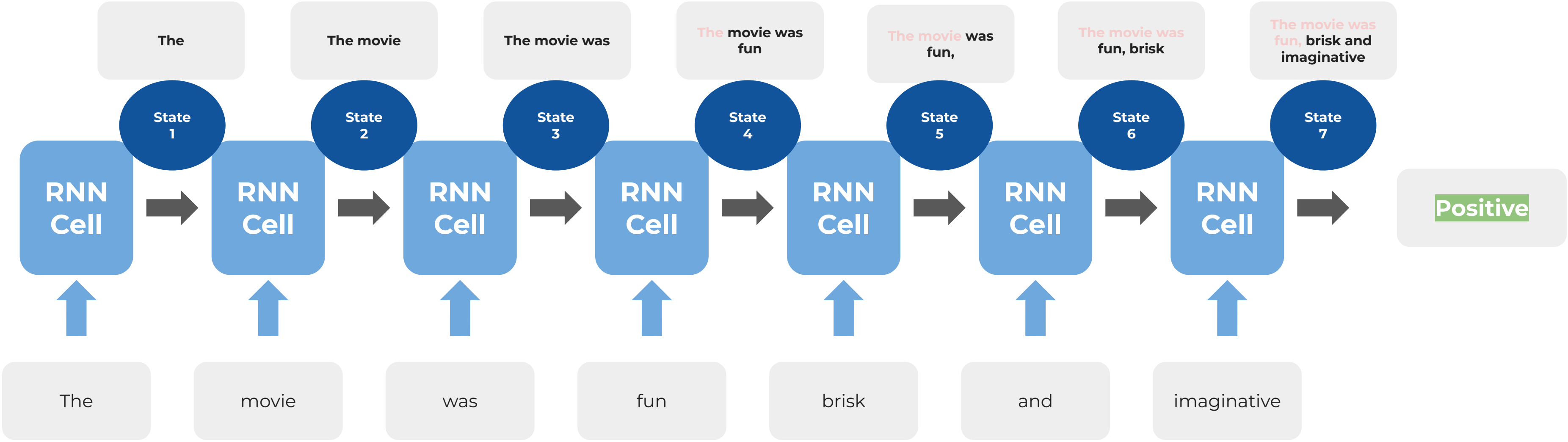
Encoder-Decoder Architecture - Example



RNNs for Sequential Learning - Limitations

Consider the following example of sentiment analysis

"The movie was fun, brisk and imaginative."



RNNs for Sequential Learning - Limitations

RNNs **cannot** effectively **capture long-term dependencies**

The model starts to **‘forget’ information** as new information keeps getting added

For the below example, we are still okay with the ‘memory loss’ ; we still got the desired output

“The movie was fun, brisk and imaginative.” => **Positive**

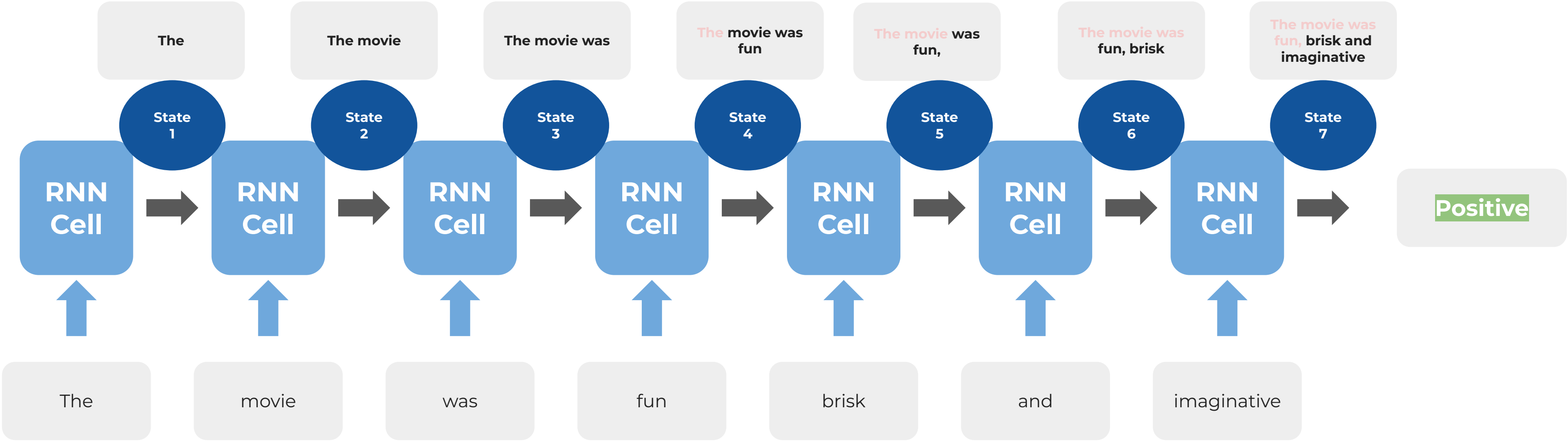
But what about the following example?

“The first half of the movie was great, but then it was a bit of a mess.”

This review is **neutral** in nature

If the model **‘forgets’ the initial part**, it will probably tag this as a **negative** review

RNNs for Sequential Learning - Limitations



The

The movie

The movie was

The movie was fun

The movie was fun,

The movie was fun, brisk

The movie was fun, brisk and imaginative

State 1

State 2

State 3

State 4

State 5

State 6

State 7

RNN Cell

RNN Cell

RNN Cell

RNN Cell

RNN Cell

RNN Cell

RNN Cell

Positive

The

movie

was

fun

brisk

and

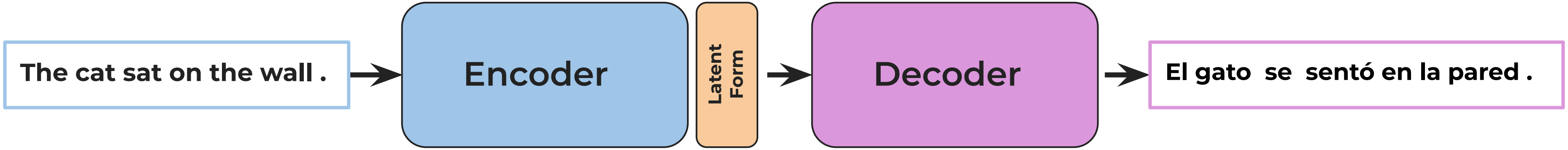
imaginative

RNNs compute one state at a time - State 2 depends on State 1, State 3 depends on State 2, ...

This **increases training time** and **computation cost**

Attention Mechanism

The Need for Attention



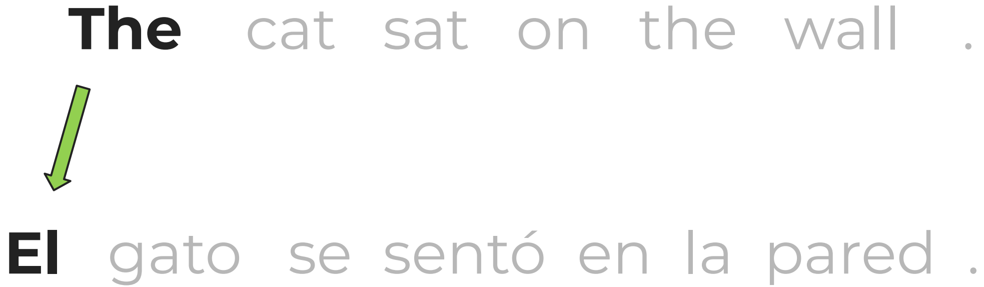
The **encoder** processed the **whole input** sentence **at once**, encoding it into a fixed representation

The **decoder** then **decoded** the output **word by word**, using the **encoded information** from the **encoder** to generate the translation or output.

Is that how humans translate?

The Need for Attention

We **focus** on **individual words or phrases** in the **input**, translating them while **considering specific contexts** rather than processing the entire input sentence at once.

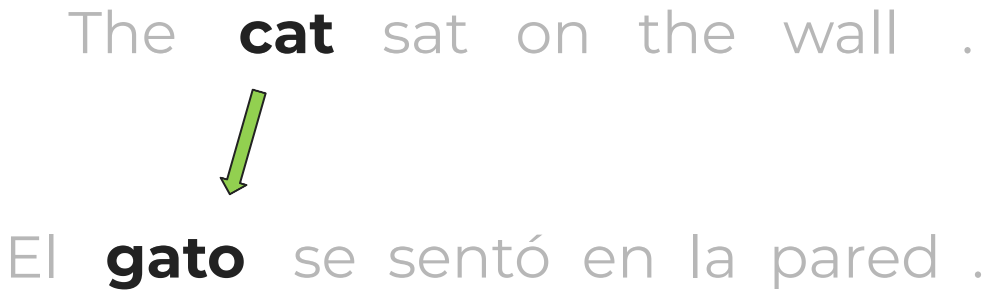


The cat sat on the wall .

El gato se sentó en la pared .

The Need for Attention

We **focus** on **individual words or phrases** in the **input**, translating them while **considering specific contexts** rather than processing the entire input sentence at once.

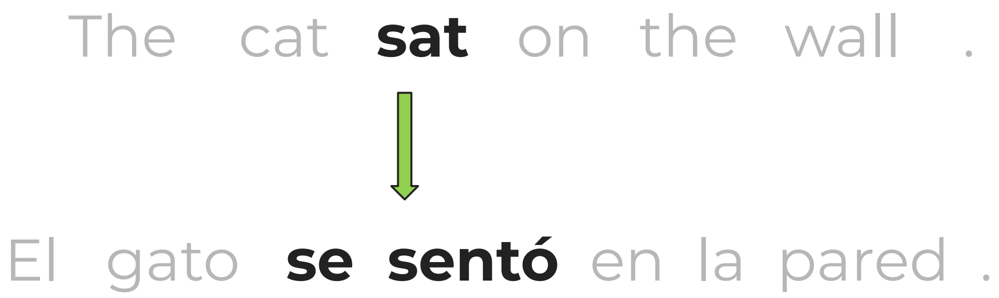


The **cat** sat on the wall .

El **gato** se sentó en la pared .

The Need for Attention

We **focus** on **individual words or phrases** in the **input**, translating them while **considering specific contexts** rather than processing the entire input sentence at once.

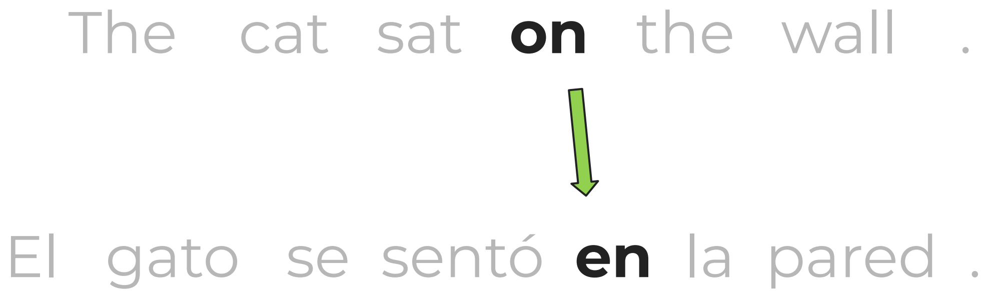


The cat **sat** on the wall .

El gato **se sentó** en la pared .

The Need for Attention

We **focus** on **individual words or phrases** in the **input**, translating them while **considering specific contexts** rather than processing the entire input sentence at once.

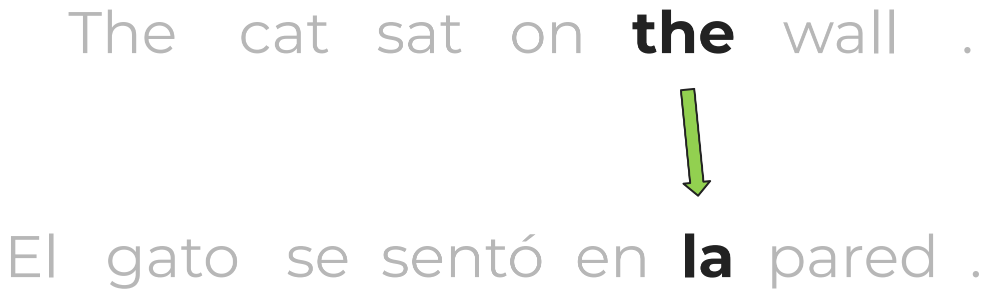


The cat sat **on** the wall .

El gato se sentó **en** la pared .

The Need for Attention

We **focus** on **individual words or phrases** in the **input**, translating them while **considering specific contexts** rather than processing the entire input sentence at once.

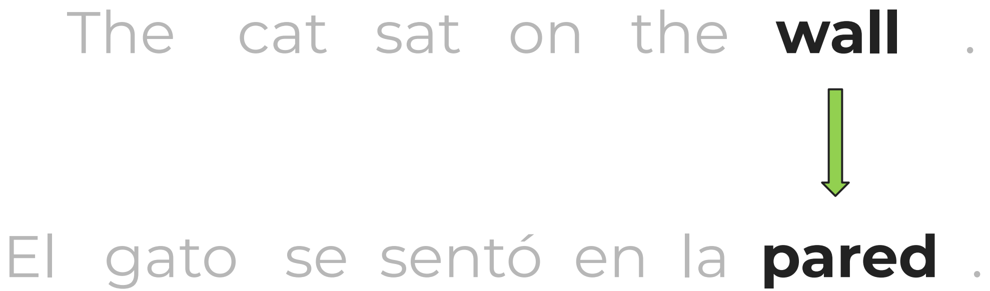


The cat sat on **the** wall .

El gato se sentó en **la** pared .

The Need for Attention

We **focus** on **individual words or phrases** in the **input**, translating them while **considering specific contexts** rather than processing the entire input sentence at once.

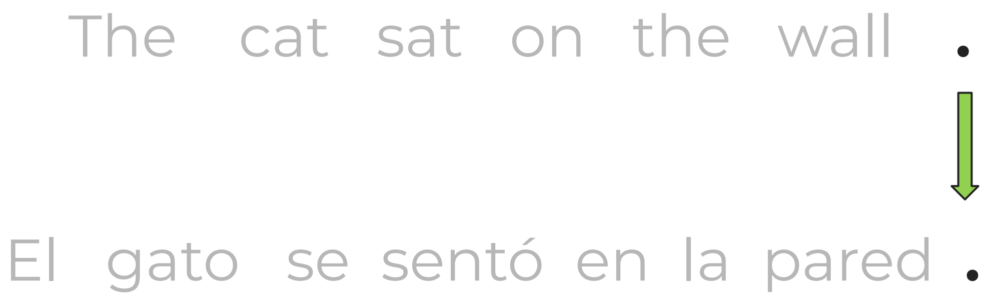


The cat sat on the **wall** .

El gato se sentó en la **pared** .

The Need for Attention

We **focus** on **individual words or phrases** in the **input**, translating them while **considering specific contexts** rather than processing the entire input sentence at once.



The cat sat on the wall .

El gato se sentó en la pared .

So, we need a way to **focus** on **specific parts** of the **input** when **generating** the **output**

In other words, the **model** needs to **learn** to '**pay attention'**

The Need for Attention

One of the **limitations** of **RNNs** was their **inability** to effectively capture **long-term dependencies**

For example, if want to translate the sentence below using an RNN

“The animal didn't cross the street because it was too tired. “

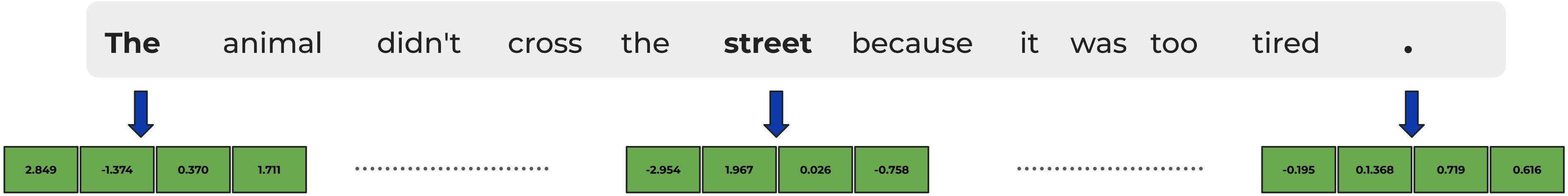
The **model** needs to **understand** that **‘it’** here refers to **‘animal’** and **not** to **‘street’**

If it doesn’t, it will result in a **translation** that completely **changes** the sentence’s **meaning**

So, we need to **understand context** and **pay attention**

Computing Attention - Intuition

Each word in the sentence can be **represented** by a **vector**



The animal didn't cross the **street** because it was too tired .

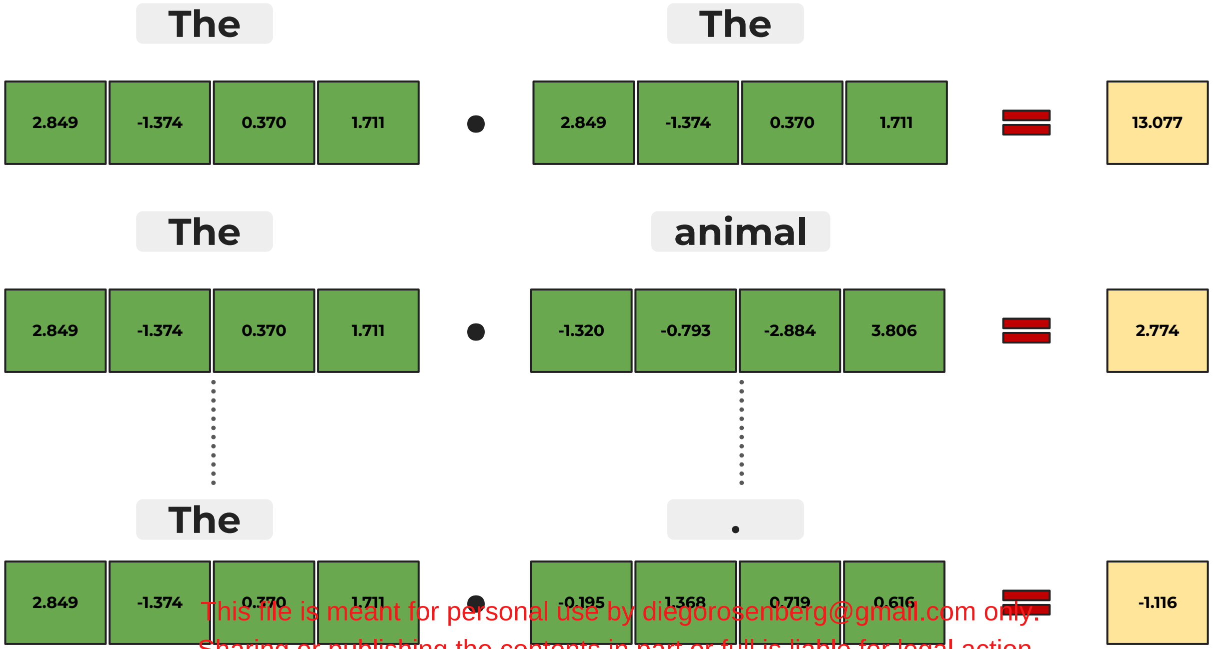
2.849	-1.374	0.370	1.711

-2.954	1.967	0.026	-0.758

-0.195	0.1.368	0.719	0.616

Computing Attention - Intuition

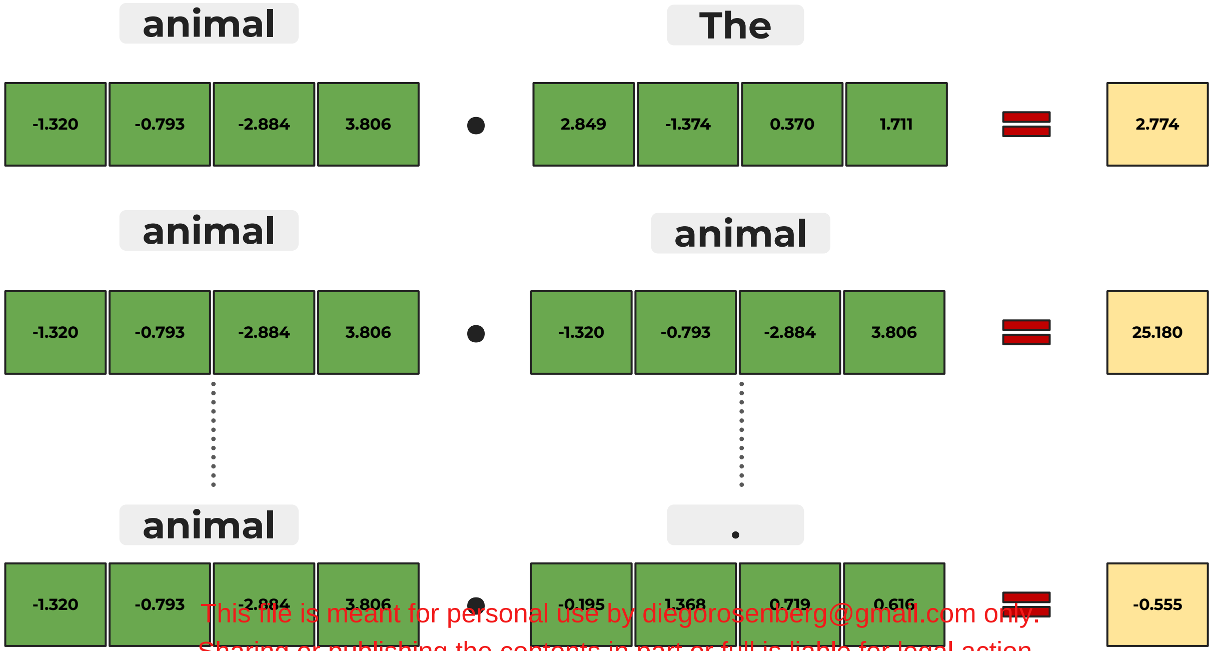
One way to **compute** the **attention score** would be to take a ‘**dot product**’ of a **word** in the sentence with **all other words**



The					The						
2.849	-1.374	0.370	1.711	●	2.849	-1.374	0.370	1.711	=	13.077	
The					animal						
2.849	-1.374	0.370	1.711	●	-1.320	-0.793	-2.884	3.806	=	2.774	
The					.						
2.849	-1.374	0.370	1.711	●	-0.195	1.368	0.719	0.616	=	-1.116	

Computing Attention - Intuition

We can **repeat** the same for **all the words** in the sentence



animal · The

-1.320 -0.793 -2.884 3.806 • 2.849 -1.374 0.370 1.711 = 2.774

animal · animal

-1.320 -0.793 -2.884 3.806 • -1.320 -0.793 -2.884 3.806 = 25.180

animal · .

-1.320 -0.793 -2.884 3.806 • -0.195 1.368 0.719 0.616 = -0.555

Computing Attention - Intuition

If you observe the dot product values for the above words, you'll notice that the values vary across different ranges

This makes it difficult to compare them and draw interpretations

It’ll be better to have probability distributions instead - fixed range of values, interpretable

How to do this?

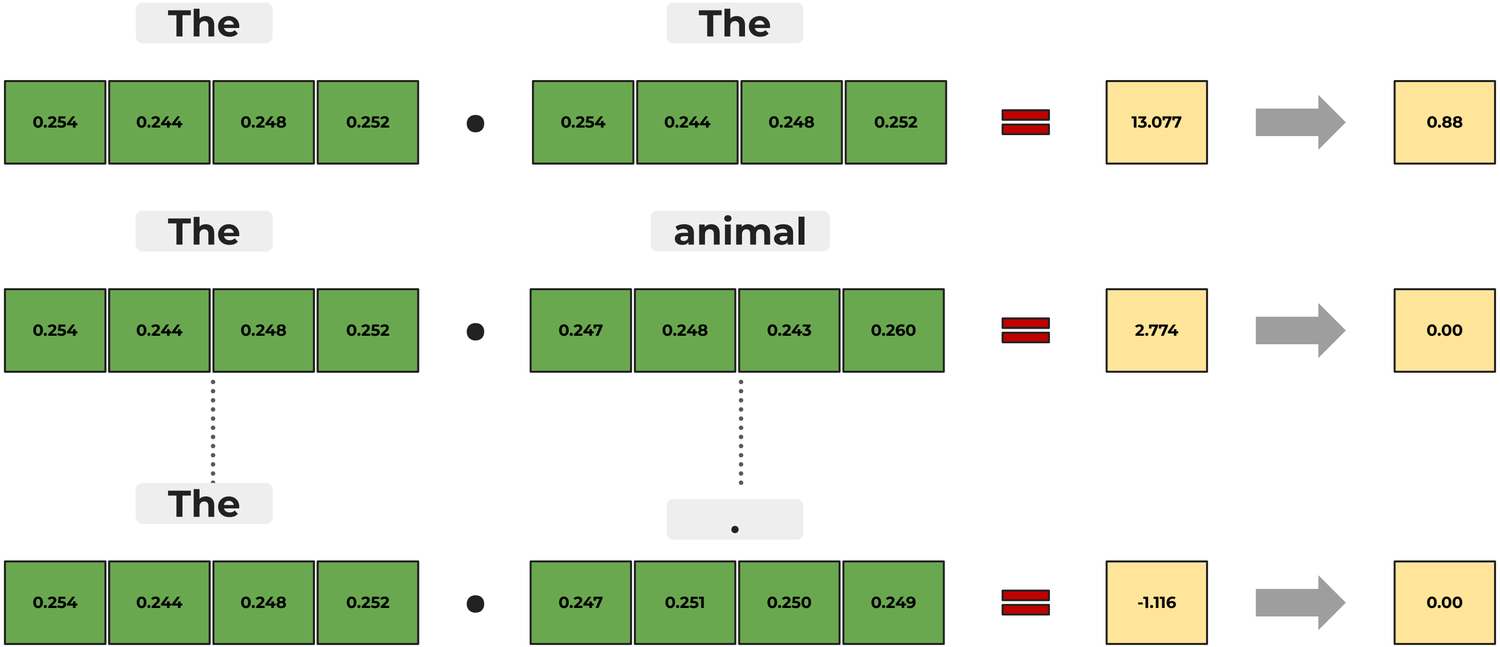
We can use the **softmax function**!

RECALL!

We did a similar thing in the last layer of a neural network for classification problems.

Computing Attention - Intuition

If you observe the below values, they are in the same range.



The · The

0.254	0.244	0.248	0.252	•	0.254	0.244	0.248	0.252	=	13.077	→	0.88

The · animal

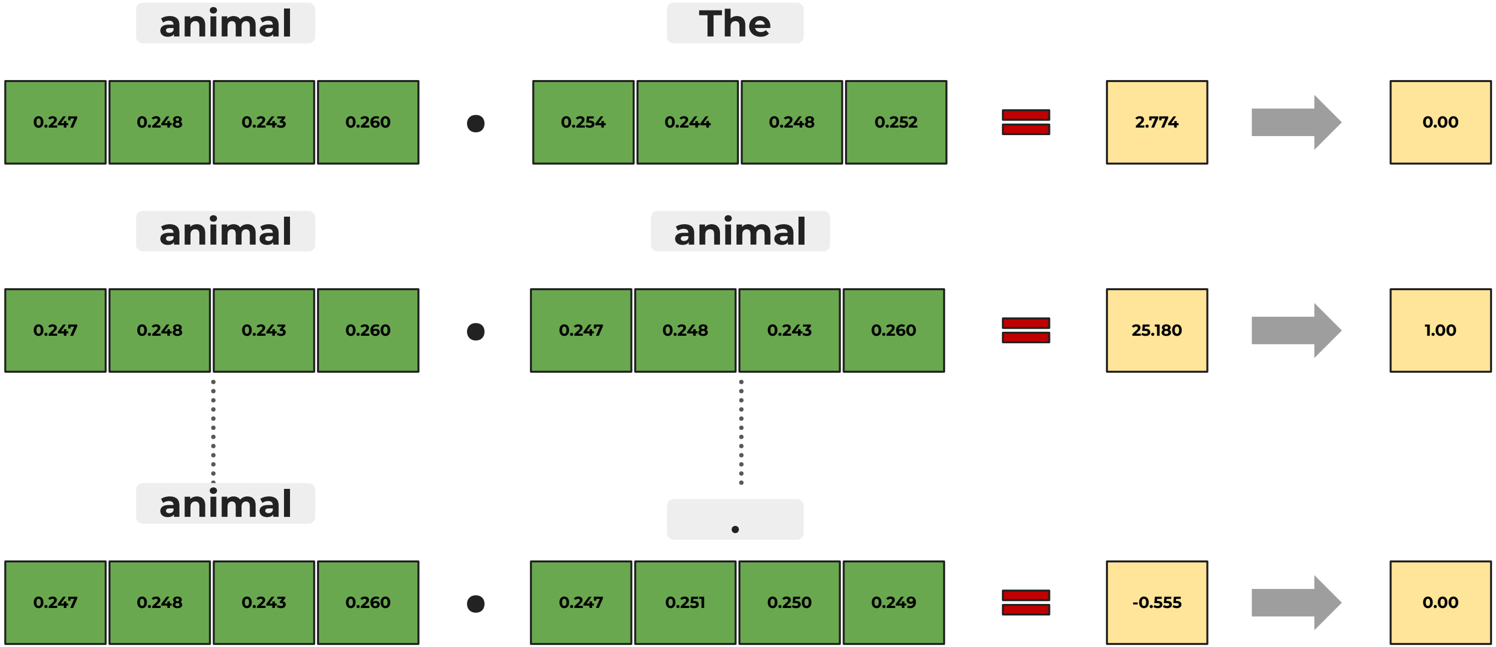
0.254	0.244	0.248	0.252	•	0.247	0.248	0.243	0.260	=	2.774	→	0.00

The · .

0.254	0.244	0.248	0.252	•	0.247	0.251	0.250	0.249	=	-1.116	→	0.00

Computing Attention - Intuition

We repeat the same for all words in the sentence



animal · The

0.247 | 0.248 | 0.243 | 0.260 ● 0.254 | 0.244 | 0.248 | 0.252 = 2.774 → 0.00

animal · animal

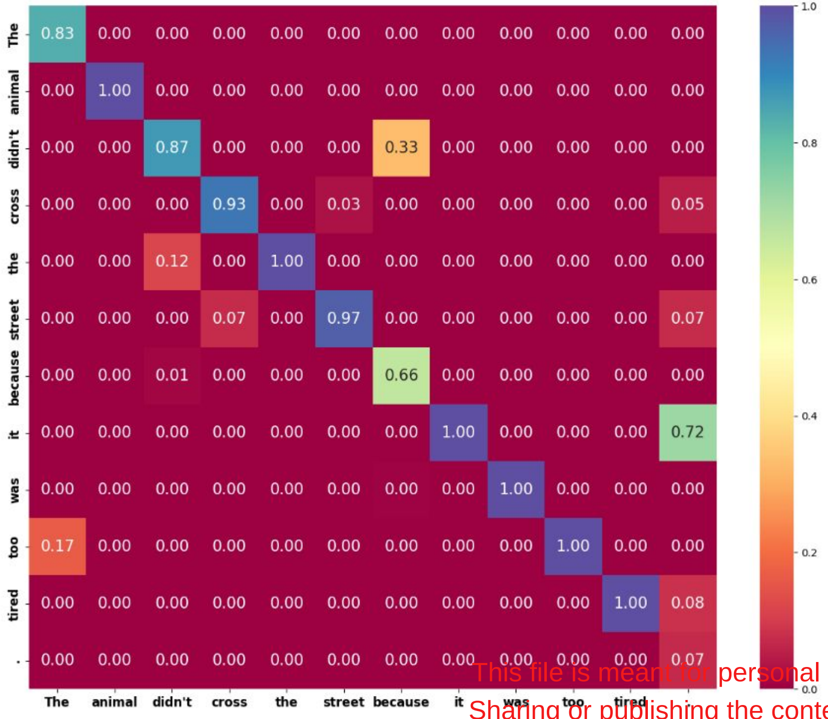
0.247 | 0.248 | 0.243 | 0.260 ● 0.247 | 0.248 | 0.243 | 0.260 = 25.180 → 1.00

animal · .

0.247 | 0.248 | 0.243 | 0.260 ● 0.247 | 0.251 | 0.250 | 0.249 = -0.555 → 0.00

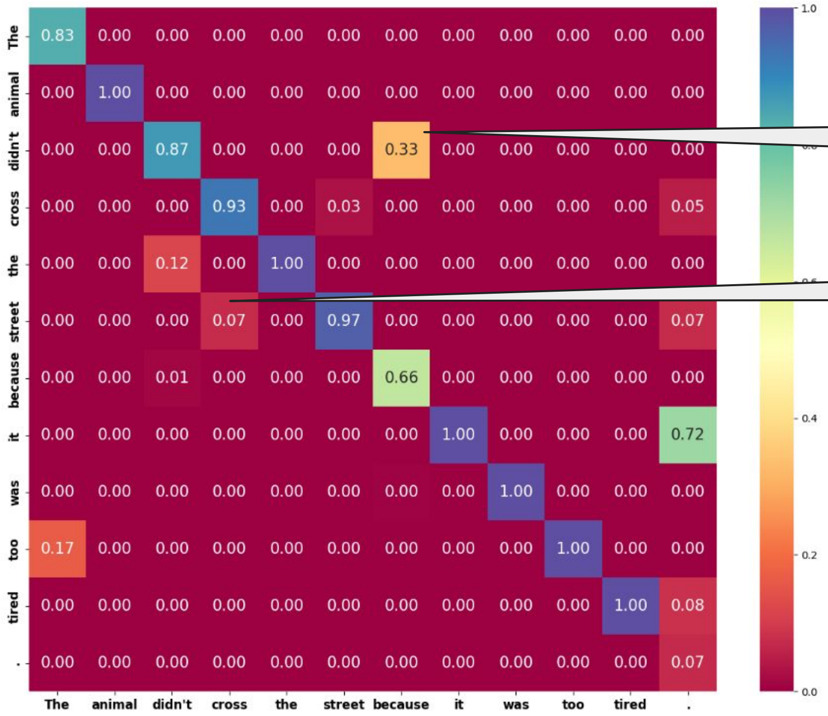
Computing Attention - Intuition

These **attention scores** for all words can be represented using a **matrix**



	The	animal	didn't	cross	the	street	because	it	was	too	tired	.
The	0.83	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
animal	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
didn't	0.00	0.00	0.87	0.00	0.00	0.00	0.33	0.00	0.00	0.00	0.00	0.00
cross	0.00	0.00	0.00	0.93	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.05
the	0.00	0.00	0.12	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
street	0.00	0.00	0.00	0.07	0.00	0.97	0.00	0.00	0.00	0.00	0.00	0.07
because	0.00	0.00	0.01	0.00	0.00	0.00	0.66	0.00	0.00	0.00	0.00	0.00
it	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.72
was	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
too	0.17	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
tired	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.08
.	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.07

Computing Attention - Intuition



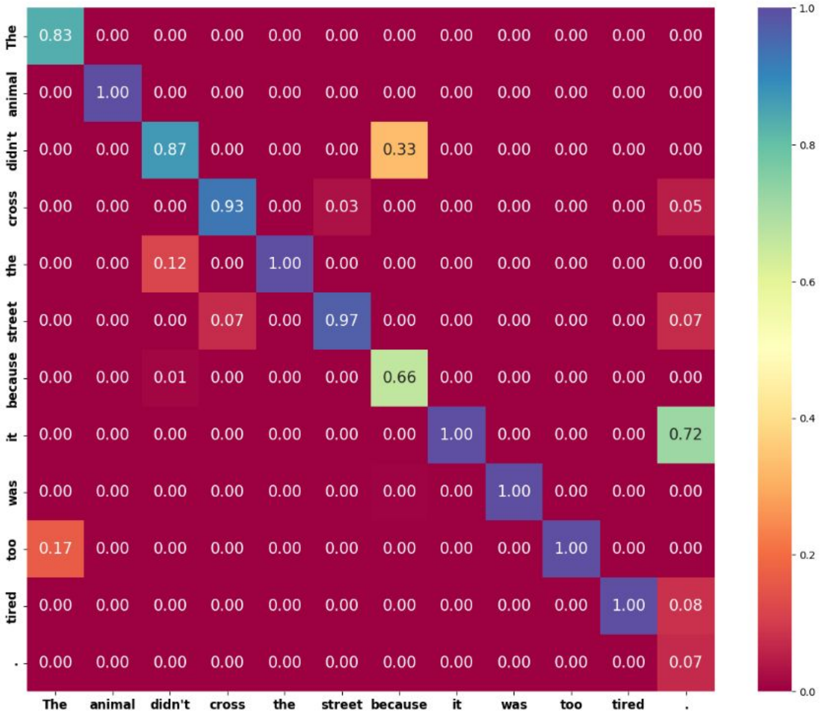
	The	animal	didn't	cross	the	street	because	it	was	too	tired	.
The	0.83	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
animal	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
didn't	0.00	0.00	0.87	0.00	0.00	0.00	0.33	0.00	0.00	0.00	0.00	0.00
cross	0.00	0.00	0.00	0.93	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.05
the	0.00	0.00	0.12	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
street	0.00	0.00	0.00	0.07	0.00	0.97	0.00	0.00	0.00	0.00	0.00	0.07
because	0.00	0.00	0.01	0.00	0.00	0.00	0.66	0.00	0.00	0.00	0.00	0.00
it	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.72
was	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
too	0.17	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
tired	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.08
.	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.07

‘**because**’ and ‘**didn’t**’ are associated with each other

‘**Cross**’ and ‘**street’** are associated with each other

But most of the words are not associated with others

Computing Attention - Intuition



	The	animal	didn't	cross	the	street	because	it	was	too	tired	.
The	0.83	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
animal	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
didn't	0.00	0.00	0.87	0.00	0.00	0.00	0.33	0.00	0.00	0.00	0.00	0.00
cross	0.00	0.00	0.00	0.93	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.05
the	0.00	0.00	0.12	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
street	0.00	0.00	0.00	0.07	0.00	0.97	0.00	0.00	0.00	0.00	0.00	0.07
because	0.00	0.00	0.01	0.00	0.00	0.00	0.66	0.00	0.00	0.00	0.00	0.00
it	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.72
was	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
too	0.17	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
tired	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.08
.	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.07

Note that this is a simple dot product

We have **not** done any **‘learning’**

On **‘learning’** these **associations**, we should be able to get **better results**

As we computed the **association of words within the sentence**, this is known as **self attention**

Computing Self Attention - Intuition

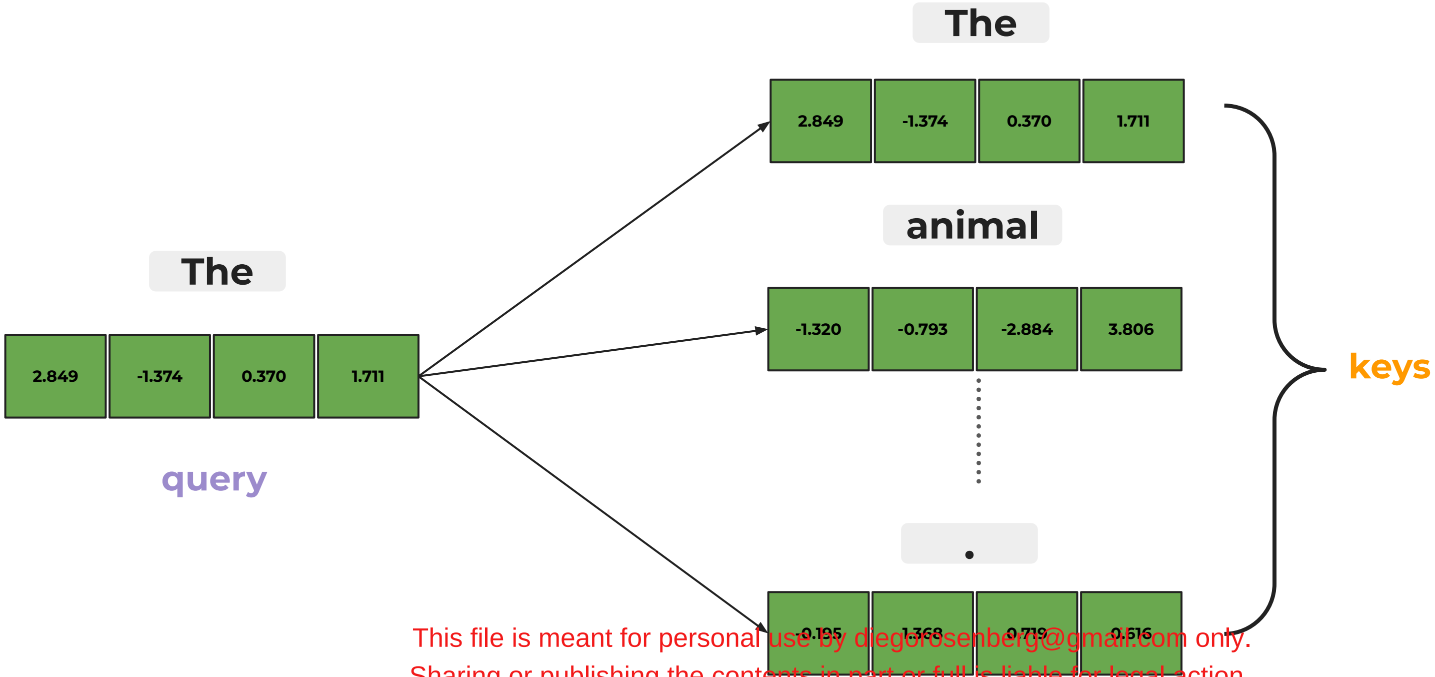
Let’s consider the word ‘The’ in the sentence

The

2.849	-1.374	0.370	1.711

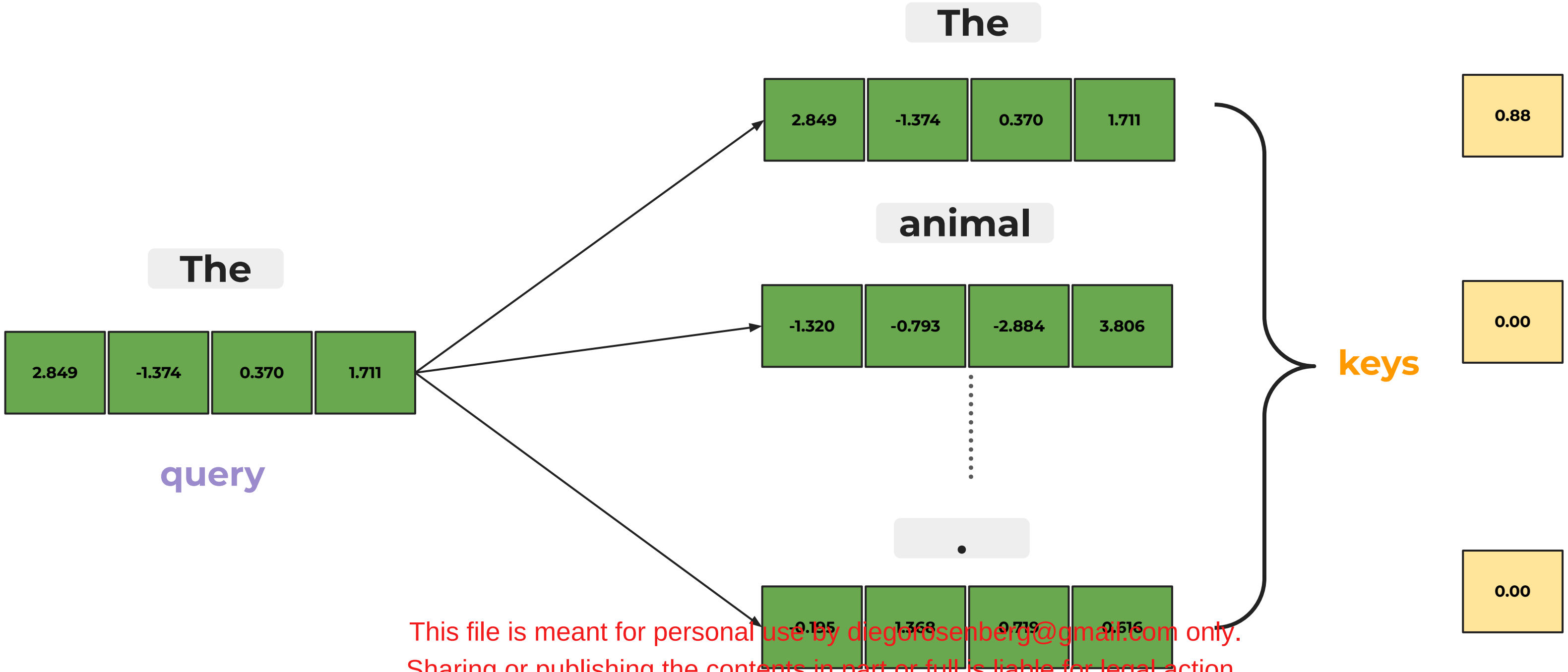
Computing Self Attention - Intuition

When computing self attention for this word, we are trying to **‘query’** information for this ‘key’ (word) against all available **‘keys’** (words)



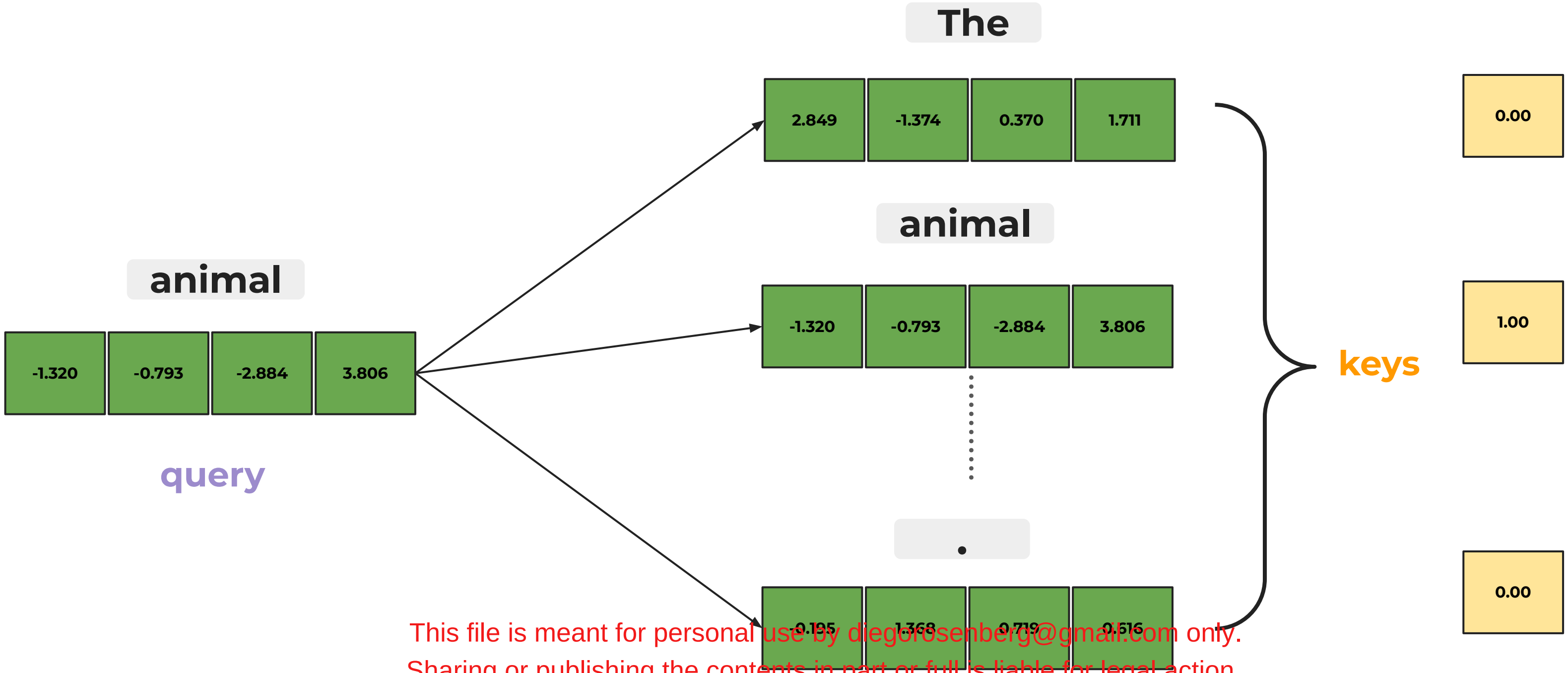
Computing Self Attention - Intuition

This will give the **self attention scores** for the **word** ‘The’ wrt **all other words** in the sentence



Computing Self Attention - Intuition

We can then **compute** this for **all words** in the sentence to get the self attention scores for each word wrt the sentence



animal (query): -1.320 | -0.793 | -2.884 | 3.806

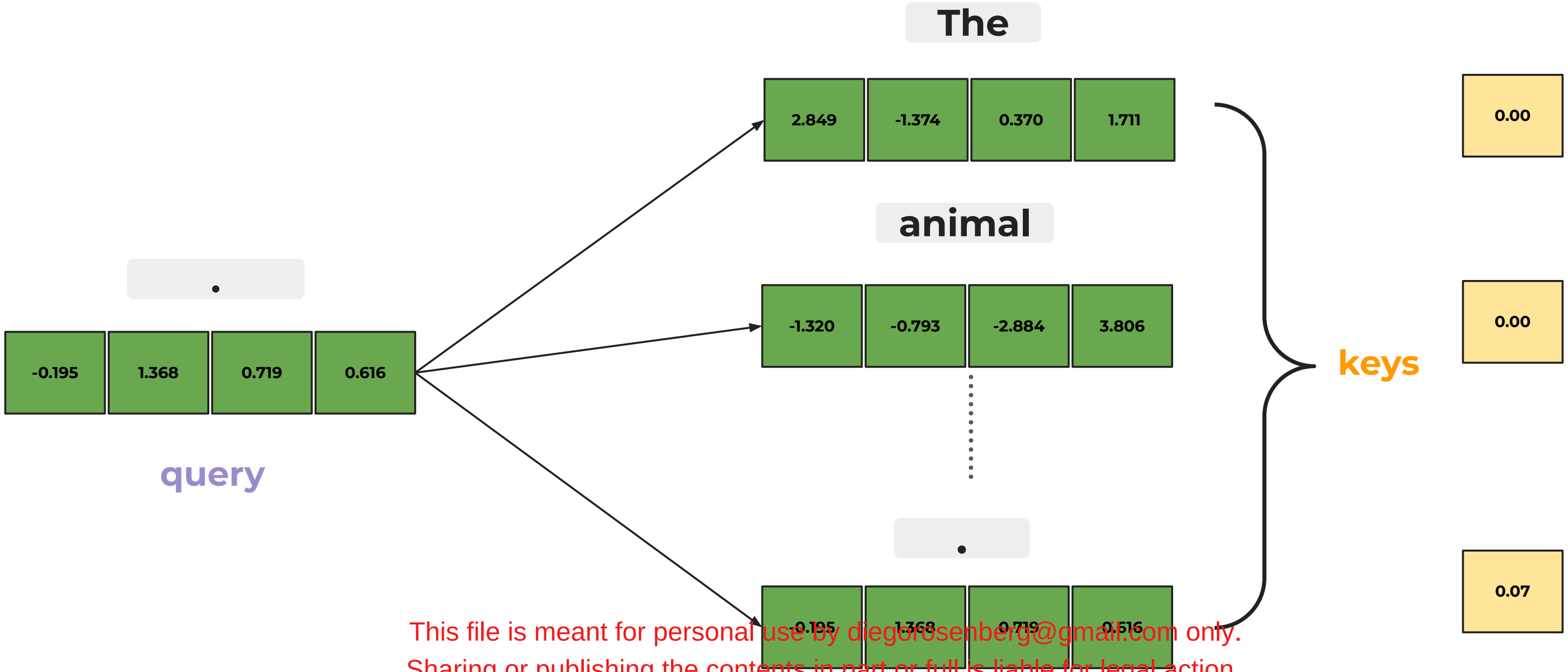
The: 2.849 | -1.374 | 0.370 | 1.711 → 0.00

animal: -1.320 | -0.793 | -2.884 | 3.806 → 1.00

.: -0.195 | 1.368 | 0.719 | 0.616 → 0.00

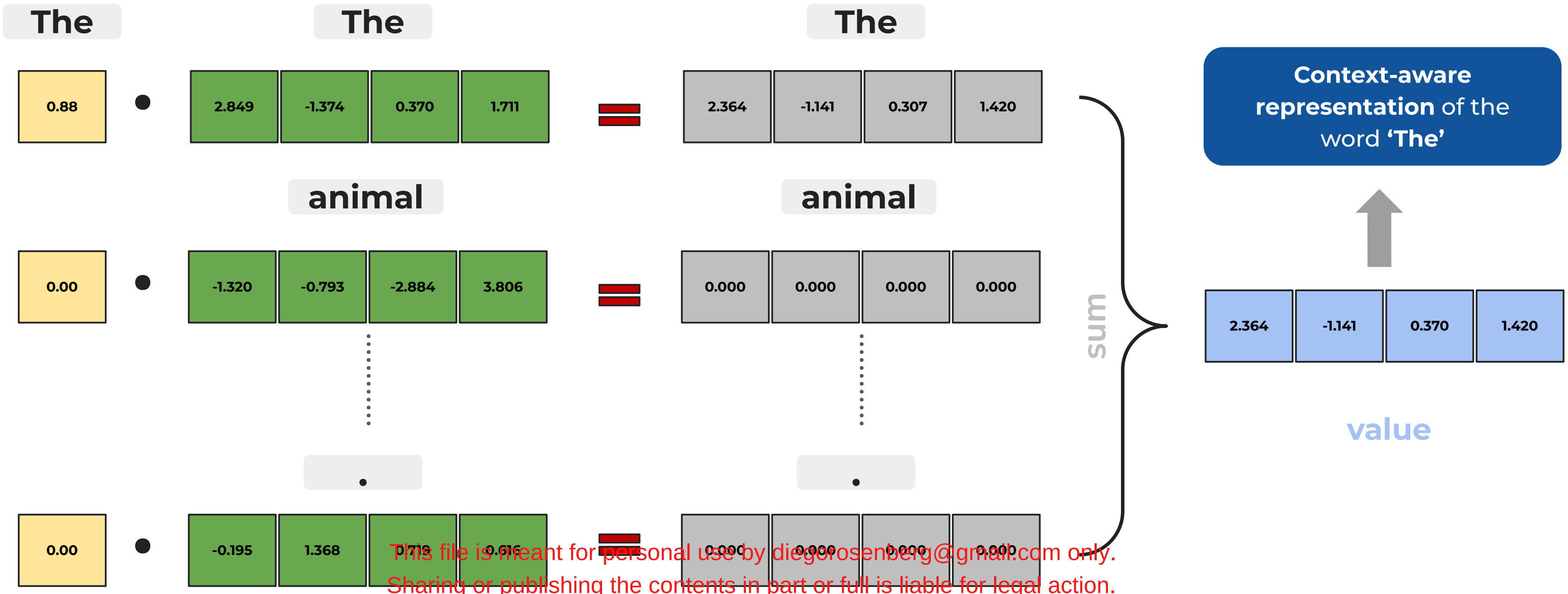
Computing Self Attention - Intuition

We can then **compute** this for **all words** in the sentence to get the self attention scores for each word wrt the sentence



Computing Self Attention - Intuition

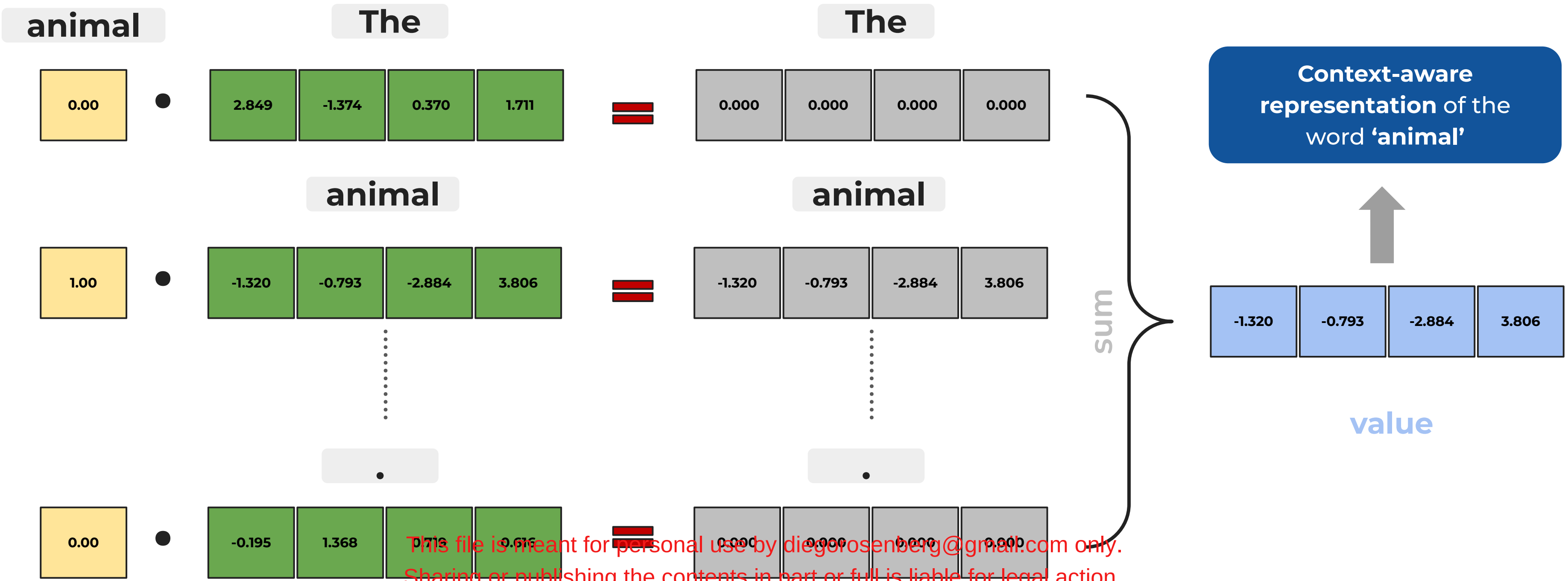
We can then combine these **self attention scores** with the original **'values'** (words in the sentence) to get a **context-aware representation** for **each word** in the sentence



Context-aware representation of the word **'The'**

Computing Self Attention - Intuition

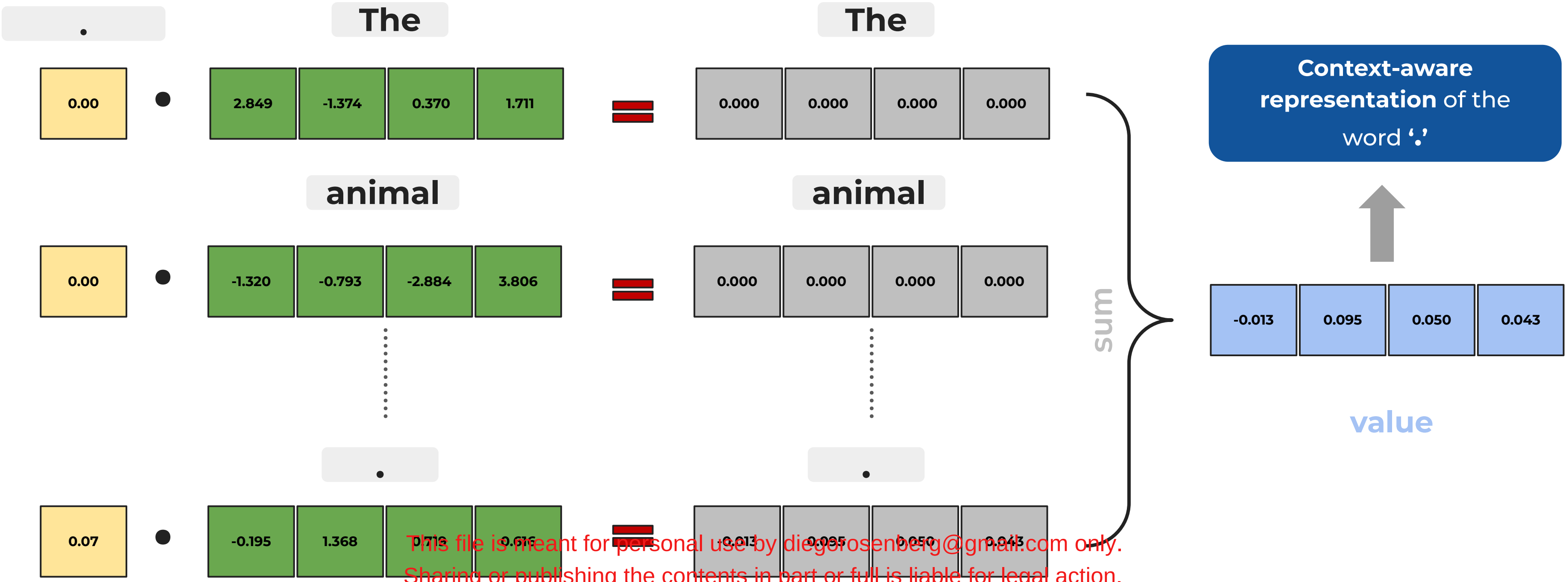
We then repeat this for all words in the sentence to get the **context-aware representations**



Context-aware representation of the word **'animal'**

Computing Self Attention - Intuition

We then repeat this for all words in the sentence to get the **context-aware representations**



Computing Self Attention - Intuition

The initial **word embedding** and the **context-aware representation** of ‘**The**’ are **very similar** - this is because the self attention score of ‘**The**’ is highest wrt to ‘**The**’

The

Embedding			
2.849	-1.374	0.370	1.711

The

Context-aware representation			
2.364	-1.141	0.370	1.420

We observe a similar pattern for the word ‘**animal**’

animal

Embedding			
-1.320	-0.793	-2.884	3.806

animal

Context-aware representation			
-1.320	-0.793	-2.884	3.806

Computing Self Attention - Intuition

However, for the word **‘.’**, the initial **word embedding** and the **context-aware representation** are **very different**

This is because there are **multiple words** in the sentence **related to ‘.’** - the context-aware representation captures this information

	.			
Embedding	-0.195	1.368	0.719	0.616

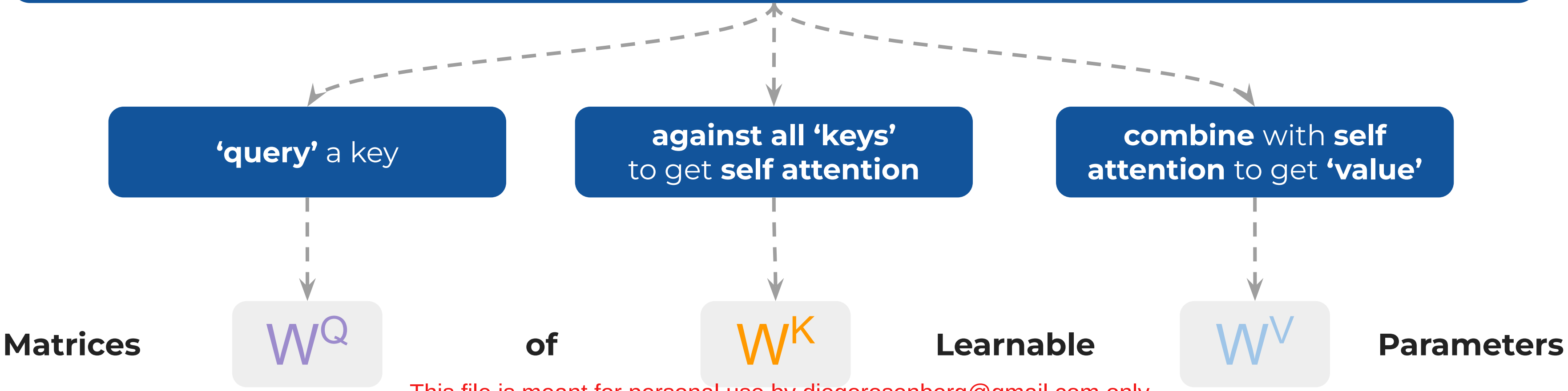
	.			
Context-aware representation	-0.013	0.095	0.050	0.043

Computing Self Attention

Note that we have **just** taken a **dot-product** of the words **so far**

There is nothing to **'learn'** here - we need to introduce some **parameters** (weights)

Remember the **steps** we talked about to get the **context-aware representation** for the **each** of the **word** in the **sentence**



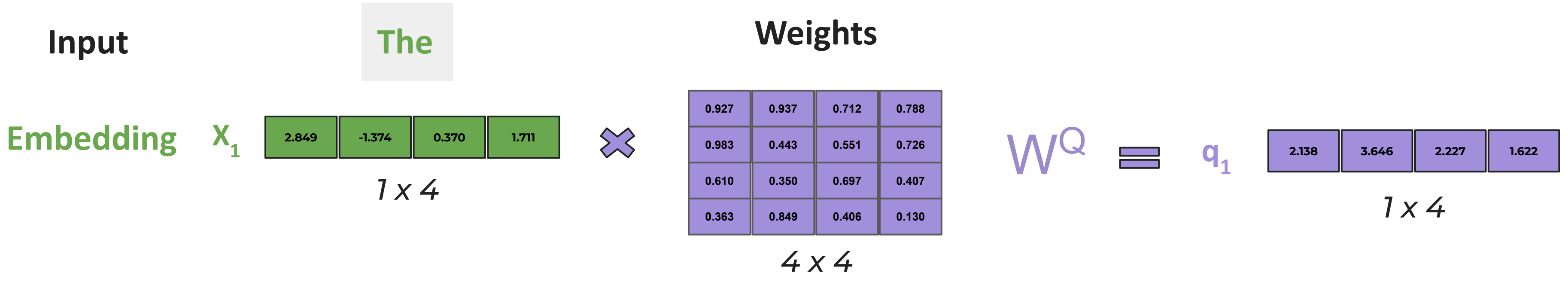
'query' a key

against all 'keys' to get **self attention**

combine with **self attention** to get **'value'**

Matrices W^Q **of** W^K **Learnable** W^V **Parameters**

Computing Self Attention



Input The

Embedding X_1

2.849	-1.374	0.370	1.711

1×4

×

Weights

0.927	0.937	0.712	0.788
0.983	0.443	0.551	0.726
0.610	0.350	0.697	0.407
0.363	0.849	0.406	0.130

4×4

W^Q = q_1

2.138	3.646	2.227	1.622

1×4

Why is the weight matrix *4 x 4*? Why not *4 x 2* or *4 x 6*?

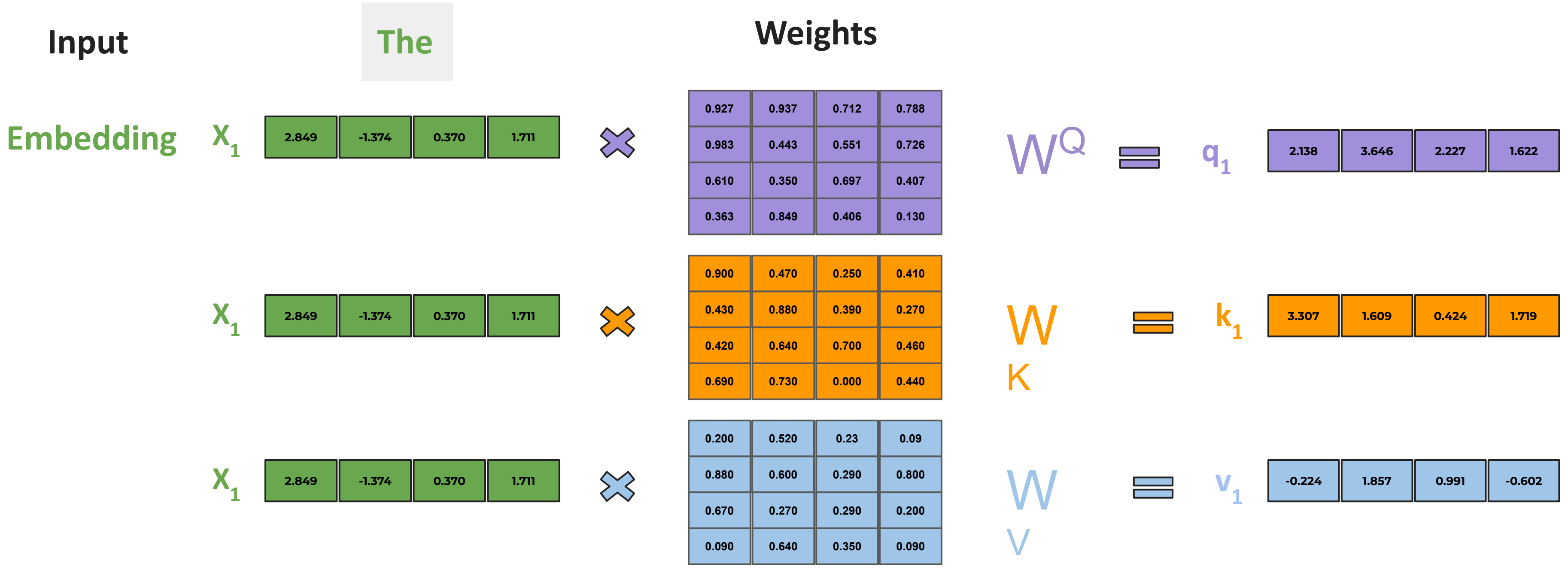
We want the **output** to be of the **same size** as the **input** here

In case we want to shrink or expand the output, we can change the weight matrix dimension

Shrink => use *4 x 2* | Expand => use *4 x 6*

Note: The **weights** assigned above are **randomly chosen** since we need to start at some point

Computing Self Attention



Input

The

Weights

Embedding

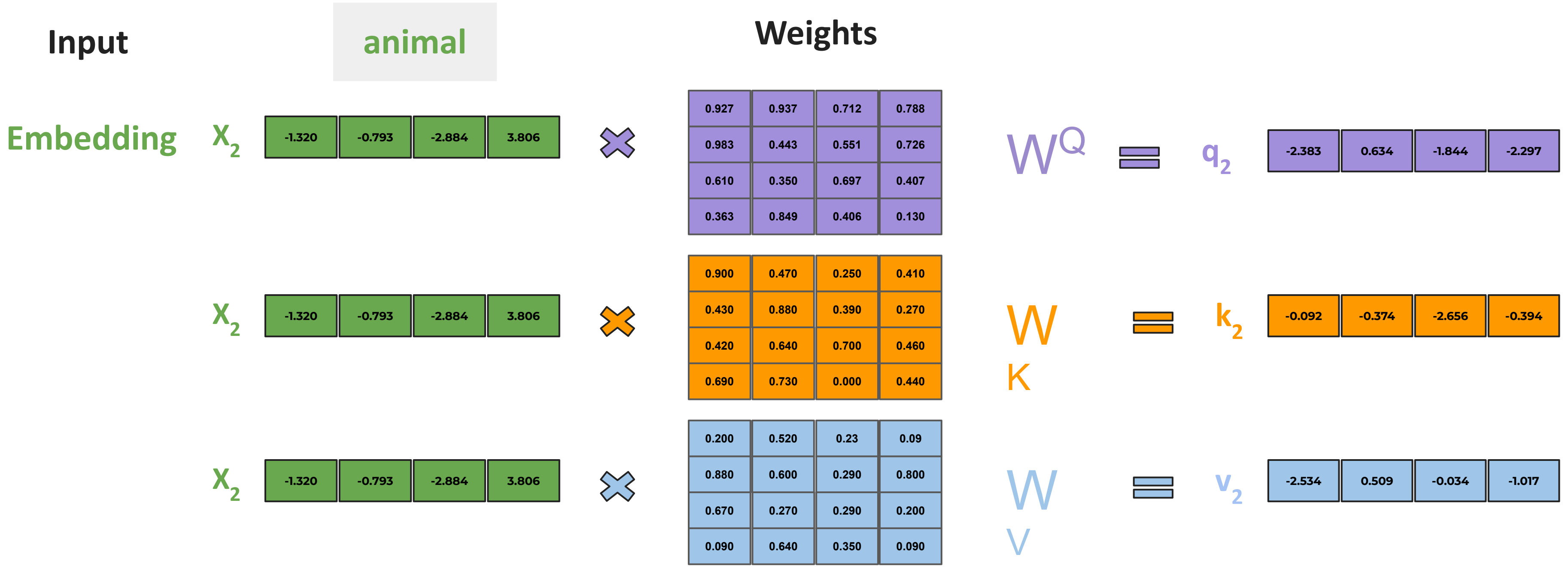
$$x_1 \begin{bmatrix} 2.849 & -1.374 & 0.370 & 1.711 \end{bmatrix} \times \begin{bmatrix} 0.927 & 0.937 & 0.712 & 0.788 \\ 0.983 & 0.443 & 0.551 & 0.726 \\ 0.610 & 0.350 & 0.697 & 0.407 \\ 0.363 & 0.849 & 0.406 & 0.130 \end{bmatrix} W^Q = q_1 \begin{bmatrix} 2.138 & 3.646 & 2.227 & 1.622 \end{bmatrix}$$

$$x_1 \begin{bmatrix} 2.849 & -1.374 & 0.370 & 1.711 \end{bmatrix} \times \begin{bmatrix} 0.900 & 0.470 & 0.250 & 0.410 \\ 0.430 & 0.880 & 0.390 & 0.270 \\ 0.420 & 0.640 & 0.700 & 0.460 \\ 0.690 & 0.730 & 0.000 & 0.440 \end{bmatrix} W^K = k_1 \begin{bmatrix} 3.307 & 1.609 & 0.424 & 1.719 \end{bmatrix}$$

$$x_1 \begin{bmatrix} 2.849 & -1.374 & 0.370 & 1.711 \end{bmatrix} \times \begin{bmatrix} 0.200 & 0.520 & 0.23 & 0.09 \\ 0.880 & 0.600 & 0.290 & 0.800 \\ 0.670 & 0.270 & 0.290 & 0.200 \\ 0.090 & 0.640 & 0.350 & 0.090 \end{bmatrix} W^V = v_1 \begin{bmatrix} -0.224 & 1.857 & 0.991 & -0.602 \end{bmatrix}$$

Note: The **weights** W^Q, W^K, and W^V are **shared** (same) across all words

Computing Self Attention



Input animal

Weights

Embedding

$$X_2 \begin{bmatrix}-1.320 & -0.793 & -2.884 & 3.806\end{bmatrix} \times \begin{bmatrix}0.927 & 0.937 & 0.712 & 0.788\\ 0.983 & 0.443 & 0.551 & 0.726\\ 0.610 & 0.350 & 0.697 & 0.407\\ 0.363 & 0.849 & 0.406 & 0.130\end{bmatrix} W^Q = q_2 \begin{bmatrix}-2.383 & 0.634 & -1.844 & -2.297\end{bmatrix}$$

$$X_2 \begin{bmatrix}-1.320 & -0.793 & -2.884 & 3.806\end{bmatrix} \times \begin{bmatrix}0.900 & 0.470 & 0.250 & 0.410\\ 0.430 & 0.880 & 0.390 & 0.270\\ 0.420 & 0.640 & 0.700 & 0.460\\ 0.690 & 0.730 & 0.000 & 0.440\end{bmatrix} W^K = k_2 \begin{bmatrix}-0.092 & -0.374 & -2.656 & -0.394\end{bmatrix}$$

$$X_2 \begin{bmatrix}-1.320 & -0.793 & -2.884 & 3.806\end{bmatrix} \times \begin{bmatrix}0.200 & 0.520 & 0.23 & 0.09\\ 0.880 & 0.600 & 0.290 & 0.800\\ 0.670 & 0.270 & 0.290 & 0.200\\ 0.090 & 0.640 & 0.350 & 0.090\end{bmatrix} W^V = v_2 \begin{bmatrix}-2.534 & 0.509 & -0.034 & -1.017\end{bmatrix}$$

We repeat this for all the words in the sentence

Computing Self Attention

Input

Weights

Embedding

$$X_{12}\ \begin{bmatrix} -0.195 & 1.368 & 0.719 & 0.616 \end{bmatrix} \times \begin{bmatrix} 0.927 & 0.937 & 0.712 & 0.788 \\ 0.983 & 0.443 & 0.551 & 0.726 \\ 0.610 & 0.350 & 0.697 & 0.407 \\ 0.363 & 0.849 & 0.406 & 0.130 \end{bmatrix} W^Q = q_{12}\ \begin{bmatrix} 1.827 & 1.199 & 1.368 & 1.212 \end{bmatrix}$$

$$X_{12}\ \begin{bmatrix} -0.195 & 1.368 & 0.719 & 0.616 \end{bmatrix} \times \begin{bmatrix} 0.900 & 0.470 & 0.250 & 0.410 \\ 0.430 & 0.880 & 0.390 & 0.270 \\ 0.420 & 0.640 & 0.700 & 0.460 \\ 0.690 & 0.730 & 0.000 & 0.440 \end{bmatrix} W_K = k_{12}\ \begin{bmatrix} 1.134 & 2.025 & 0.995 & 0.894 \end{bmatrix}$$

$$X_{12}\ \begin{bmatrix} -0.195 & 1.368 & 0.719 & 0.616 \end{bmatrix} \times \begin{bmatrix} 0.200 & 0.520 & 0.23 & 0.09 \\ 0.880 & 0.600 & 0.290 & 0.800 \\ 0.670 & 0.270 & 0.290 & 0.200 \\ 0.090 & 0.640 & 0.350 & 0.090 \end{bmatrix} W_V = v_{12}\ \begin{bmatrix} 1.703 & 1.312 & 0.774 & 1.270 \end{bmatrix}$$

We repeat this for all the words in the sentence

Computing Self Attention

Input			The		
Embedding	X_1	2.849	-1.374	0.370	1.711
Query	q_1	2.138	3.646	2.227	1.622
Key	k_1	3.307	1.609	0.424	1.719
Value	v_1	-0.224	1.857	0.991	-0.602

Dot Product $q_1 * k^T_1 = 16.676$

Dot product of ‘The’ wrt ‘The’

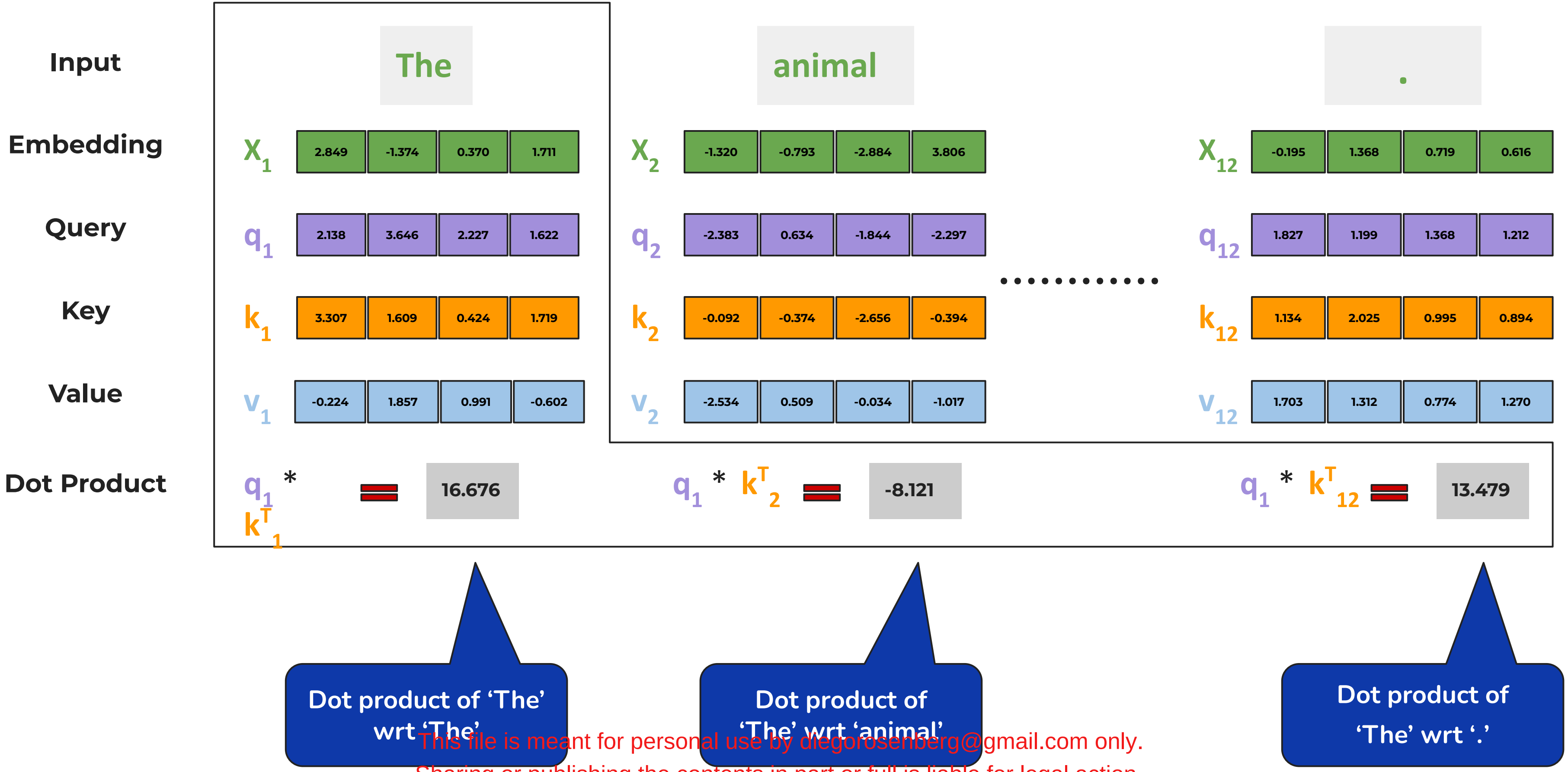
Computing Self Attention

Input		The					animal			
Embedding	x_1	2.849	-1.374	0.370	1.711	x_2	-1.320	-0.793	-2.884	3.806
Query	q_1	2.138	3.646	2.227	1.622	q_2	-2.383	0.634	-1.844	-2.297
Key	k_1	3.307	1.609	0.424	1.719	k_2	-0.092	-0.374	-2.656	-0.394
Value	v_1	-0.224	1.857	0.991	-0.602	v_2	-2.534	0.509	-0.034	-1.017
Dot Product	$q_1 * k^T_1$	=	16.676			$q_1 * k^T_2$	=	-8.121		

Dot product of ‘The’ wrt ‘The’

Dot product of ‘The’ wrt ‘animal’

Computing Self Attention



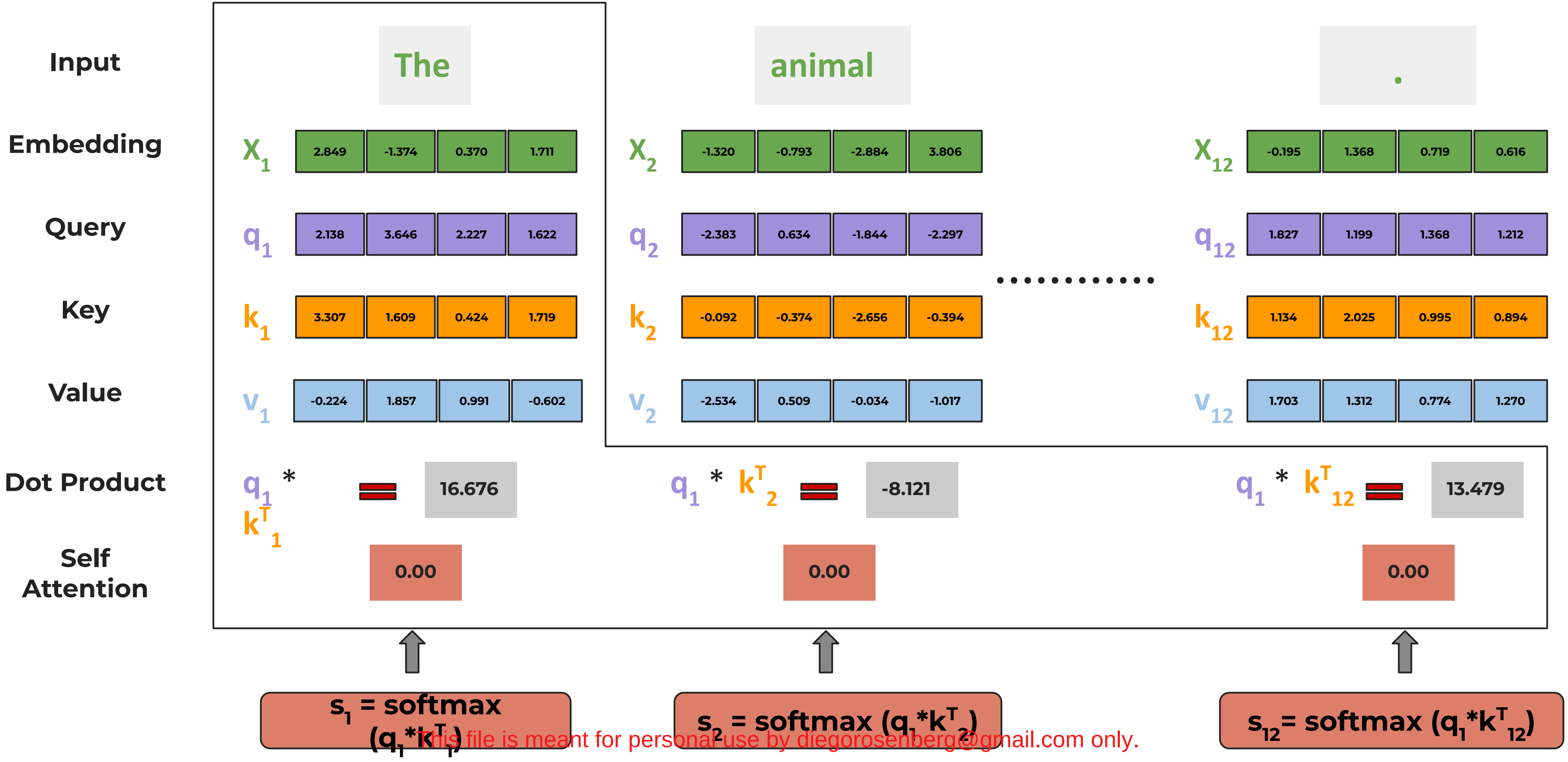
Input		The					animal						.			
Embedding	x_1	2.849	-1.374	0.370	1.711	x_2	-1.320	-0.793	-2.884	3.806		x_{12}	-0.195	1.368	0.719	0.616
Query	q_1	2.138	3.646	2.227	1.622	q_2	-2.383	0.634	-1.844	-2.297		q_{12}	1.827	1.199	1.368	1.212
Key	k_1	3.307	1.609	0.424	1.719	k_2	-0.092	-0.374	-2.656	-0.394	············	k_{12}	1.134	2.025	0.995	0.894
Value	v_1	-0.224	1.857	0.991	-0.602	v_2	-2.534	0.509	-0.034	-1.017		v_{12}	1.703	1.312	0.774	1.270
Dot Product	$q_1 * k^T_1$	=	16.676			$q_1 * k^T_2$	=	-8.121				$q_1 * k^T_{12}$	=	13.479		

Dot product of 'The' wrt 'The'

Dot product of 'The' wrt 'animal'

Dot product of 'The' wrt '.'

Computing Self Attention



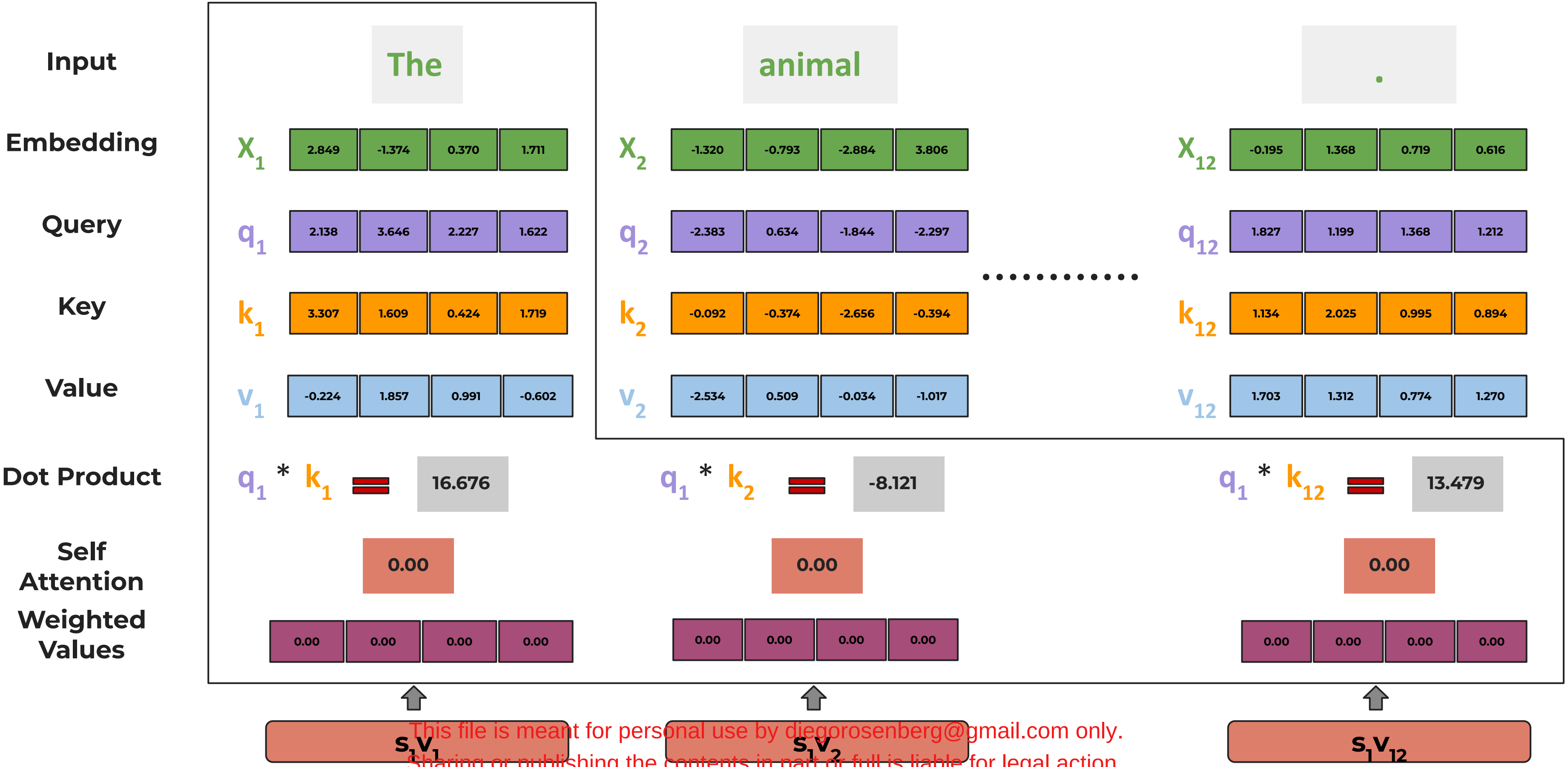
Input		The	animal		.
Embedding		X_1: 2.849, -1.374, 0.370, 1.711	X_2: -1.320, -0.793, -2.884, 3.806		X_{12}: -0.195, 1.368, 0.719, 0.616
Query		q_1: 2.138, 3.646, 2.227, 1.622	q_2: -2.383, 0.634, -1.844, -2.297	....	q_{12}: 1.827, 1.199, 1.368, 1.212
Key		k_1: 3.307, 1.609, 0.424, 1.719	k_2: -0.092, -0.374, -2.656, -0.394		k_{12}: 1.134, 2.025, 0.995, 0.894
Value		v_1: -0.224, 1.857, 0.991, -0.602	v_2: -2.534, 0.509, -0.034, -1.017		v_{12}: 1.703, 1.312, 0.774, 1.270
Dot Product		$q_1 * k^T_1$ = 16.676	$q_1 * k^T_2$ = -8.121		$q_1 * k^T_{12}$ = 13.479
Self Attention		0.00	0.00		0.00

s_1 = softmax $(q_1 * k^T_1)$

s_2 = softmax $(q_1 * k^T_2)$

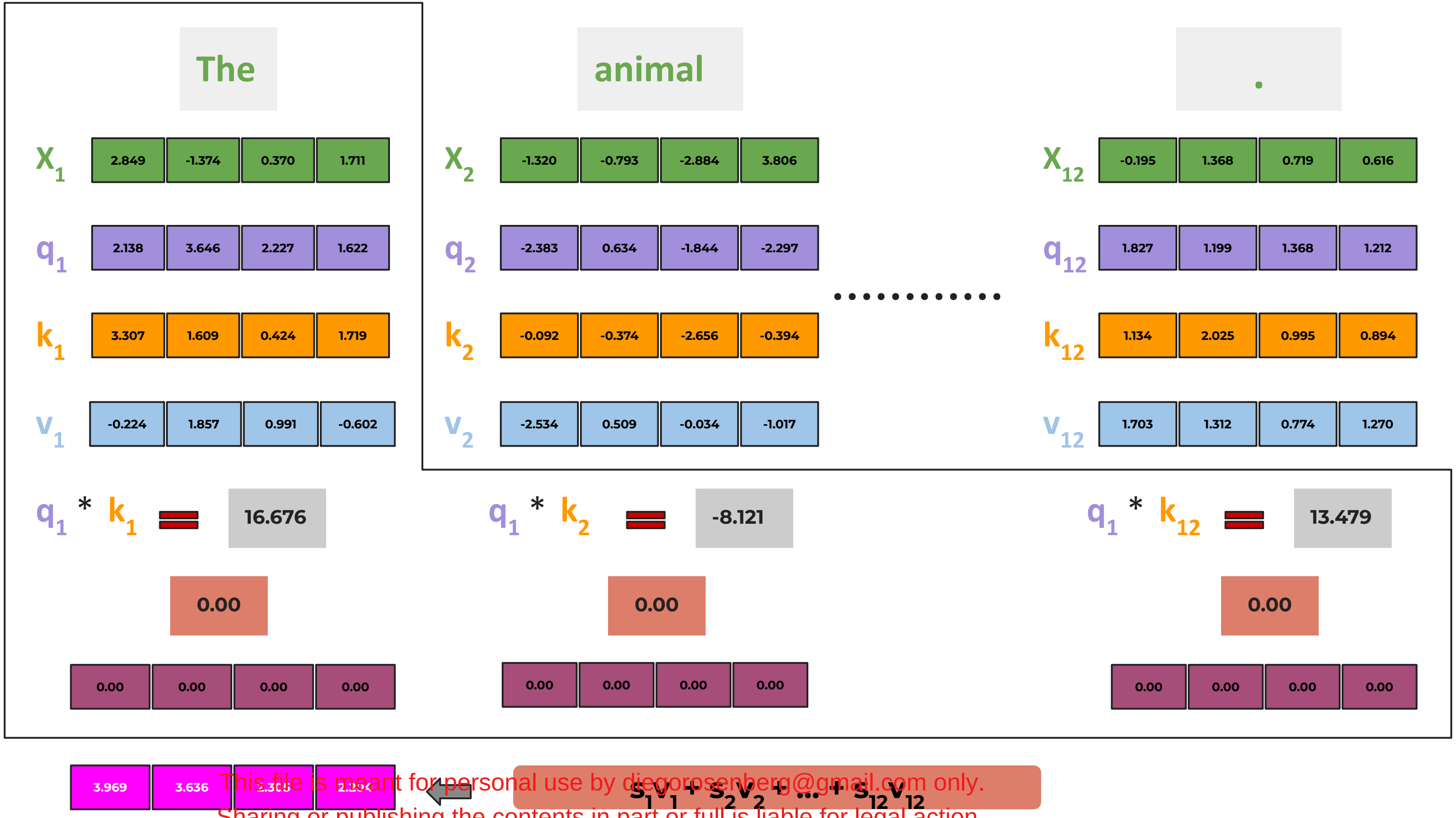
s_{12} = softmax $(q_1 * k^T_{12})$

Computing Self Attention



Input	The	animal		.
Embedding	x_1: 2.849, -1.374, 0.370, 1.711	x_2: -1.320, -0.793, -2.884, 3.806		x_{12}: -0.195, 1.368, 0.719, 0.616
Query	q_1: 2.138, 3.646, 2.227, 1.622	q_2: -2.383, 0.634, -1.844, -2.297		q_{12}: 1.827, 1.199, 1.368, 1.212
Key	k_1: 3.307, 1.609, 0.424, 1.719	k_2: -0.092, -0.374, -2.656, -0.394	....	k_{12}: 1.134, 2.025, 0.995, 0.894
Value	v_1: -0.224, 1.857, 0.991, -0.602	v_2: -2.534, 0.509, -0.034, -1.017		v_{12}: 1.703, 1.312, 0.774, 1.270
Dot Product	$q_1 * k_1 =$ 16.676	$q_1 * k_2 =$ -8.121		$q_1 * k_{12} =$ 13.479
Self Attention	0.00	0.00		0.00
Weighted Values	0.00, 0.00, 0.00, 0.00	0.00, 0.00, 0.00, 0.00		0.00, 0.00, 0.00, 0.00
	s_1v_1	s_1v_2		s_1v_{12}

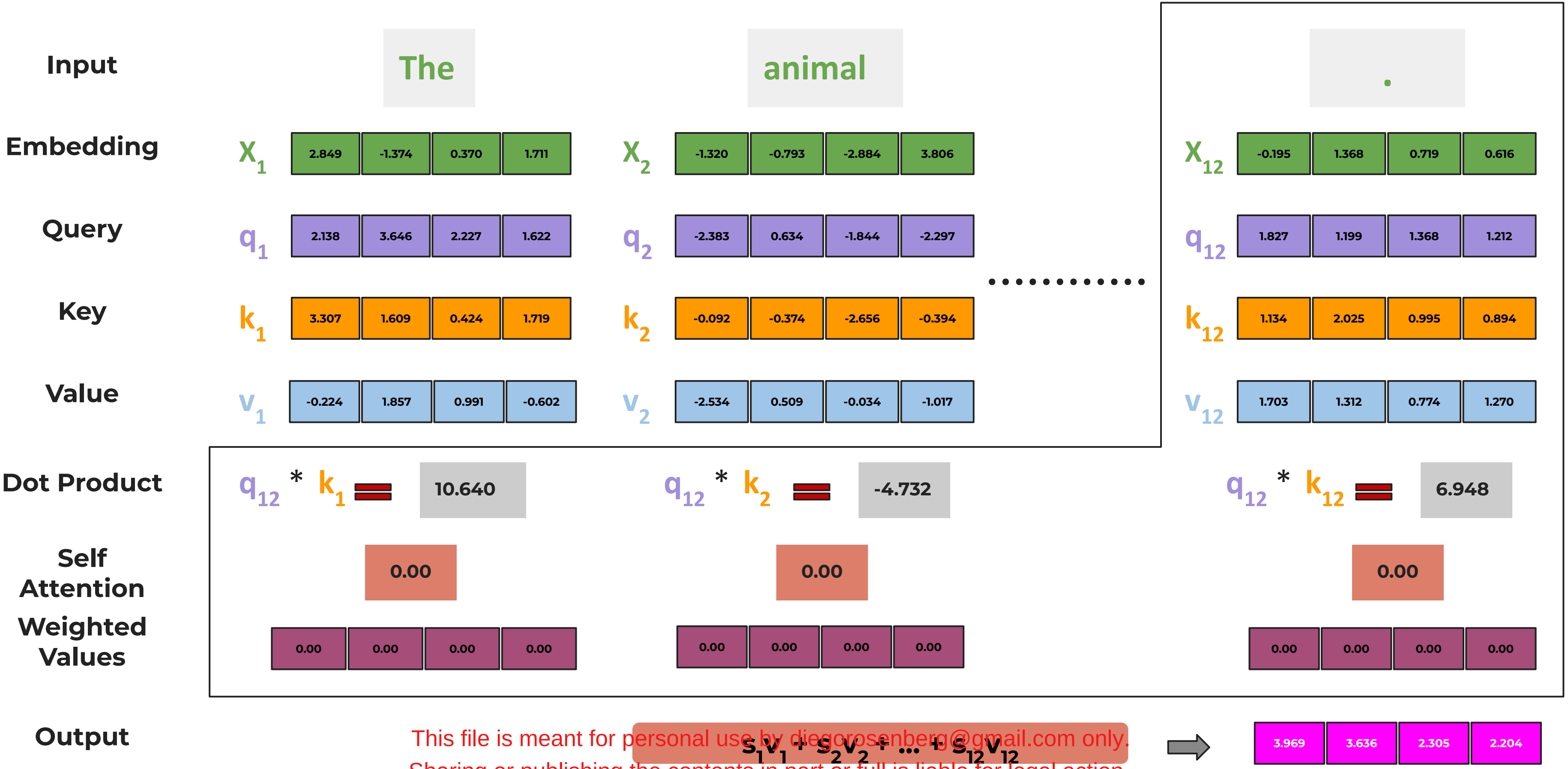
Computing Self Attention



	The	animal	…………	.
Input	The	animal		.
Embedding	x_1: 2.849, -1.374, 0.370, 1.711	x_2: -1.320, -0.793, -2.884, 3.806		x_{12}: -0.195, 1.368, 0.719, 0.616
Query	q_1: 2.138, 3.646, 2.227, 1.622	q_2: -2.383, 0.634, -1.844, -2.297		q_{12}: 1.827, 1.199, 1.368, 1.212
Key	k_1: 3.307, 1.609, 0.424, 1.719	k_2: -0.092, -0.374, -2.656, -0.394		k_{12}: 1.134, 2.025, 0.995, 0.894
Value	v_1: -0.224, 1.857, 0.991, -0.602	v_2: -2.534, 0.509, -0.034, -1.017		v_{12}: 1.703, 1.312, 0.774, 1.270
Dot Product	$q_1 * k_1 =$ 16.676	$q_1 * k_2 =$ -8.121		$q_1 * k_{12} =$ 13.479
Self Attention	0.00	0.00		0.00
Weighted Values	0.00, 0.00, 0.00, 0.00	0.00, 0.00, 0.00, 0.00		0.00, 0.00, 0.00, 0.00

Output: 3.969, 3.636, 2.305, 2.204 ← $s_1 v_1 + s_2 v_2 + \ldots + s_{12} v_{12}$

Computing Self Attention



Input	The	animal	…………	.
Embedding	x_1: 2.849, -1.374, 0.370, 1.711	x_2: -1.320, -0.793, -2.884, 3.806		x_{12}: -0.195, 1.368, 0.719, 0.616
Query	q_1: 2.138, 3.646, 2.227, 1.622	q_2: -2.383, 0.634, -1.844, -2.297		q_{12}: 1.827, 1.199, 1.368, 1.212
Key	k_1: 3.307, 1.609, 0.424, 1.719	k_2: -0.092, -0.374, -2.656, -0.394		k_{12}: 1.134, 2.025, 0.995, 0.894
Value	v_1: -0.224, 1.857, 0.991, -0.602	v_2: -2.534, 0.509, -0.034, -1.017		v_{12}: 1.703, 1.312, 0.774, 1.270
Dot Product	$q_{12} * k_1 =$ 10.640	$q_{12} * k_2 =$ -4.732		$q_{12} * k_{12} =$ 6.948
Self Attention	0.00	0.00		0.00
Weighted Values	0.00, 0.00, 0.00, 0.00	0.00, 0.00, 0.00, 0.00		0.00, 0.00, 0.00, 0.00

Output: $s_1v_1 + s_2v_2 + \dots + s_{12}v_{12}$ → 3.969, 3.636, 2.305, 2.204

Computing Self Attention

Let’s go over the computation once again for the first word in the sentence

$$\text{softmax}\left([q_1] \quad * \quad [k^T{}_1]\right) \quad * \quad v_1$$

q_1	2.138	3.646	2.227	1.622

*

$k^T{}_1$
3.307
1.609
0.424
1.719

*

-0.224	1.857	0.991	-0.602	v_1

Computing Self Attention

This is how we can do it for the first two words in the sentence

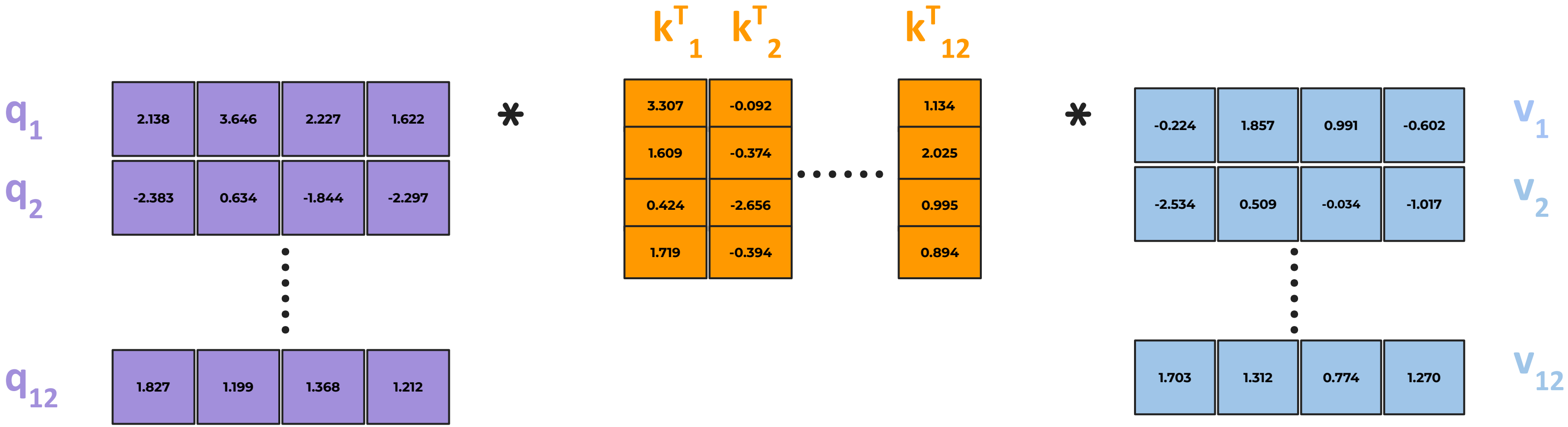
$$\text{softmax}\Big([q_1,q_2] * [k^T_1,k^T_2\Big)\,] * [v_1,v_2]$$

$$\begin{array}{c|cccc} q_1 & 2.138 & 3.646 & 2.227 & 1.622 \\ q_2 & -2.383 & 0.634 & -1.844 & -2.297 \end{array} * \begin{array}{cc} k^T_1 & k^T_2 \\ \hline 3.307 & -0.092 \\ 1.609 & -0.374 \\ 0.424 & -2.656 \\ 1.719 & -0.394 \end{array} * \begin{array}{cccc|c} -0.224 & 1.857 & 0.991 & -0.602 & v_1 \\ -2.534 & 0.509 & -0.034 & -1.017 & v_2 \end{array}$$

Computing Self Attention

This is how we can do it for all the words in the sentence

$$\text{softmax}\left([q_1,q_2,..q_{11}] * [k^T_1,k^T_2,..k^T_{12}]\right) * [v_1,v_2,..v_{12}]$$



q_1	2.138	3.646	2.227	1.622
q_2	-2.383	0.634	-1.844	-2.297
⋮				
q_{12}	1.827	1.199	1.368	1.212

*

k^T_1	k^T_2		k^T_{12}
3.307	-0.092		1.134
1.609	-0.374	……	2.025
0.424	-2.656		0.995
1.719	-0.394		0.894

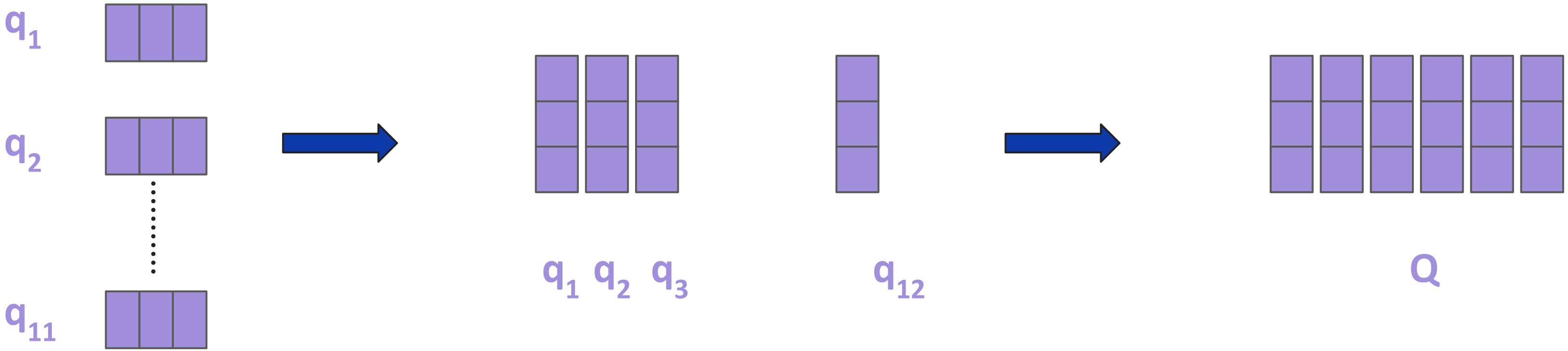
*

-0.224	1.857	0.991	-0.602	v_1
-2.534	0.509	-0.034	-1.017	v_2
⋮				
1.703	1.312	0.774	1.270	v_{12}

Computing Self Attention

We can ‘**stack**’ the vector representations of all words together into a **matrix**

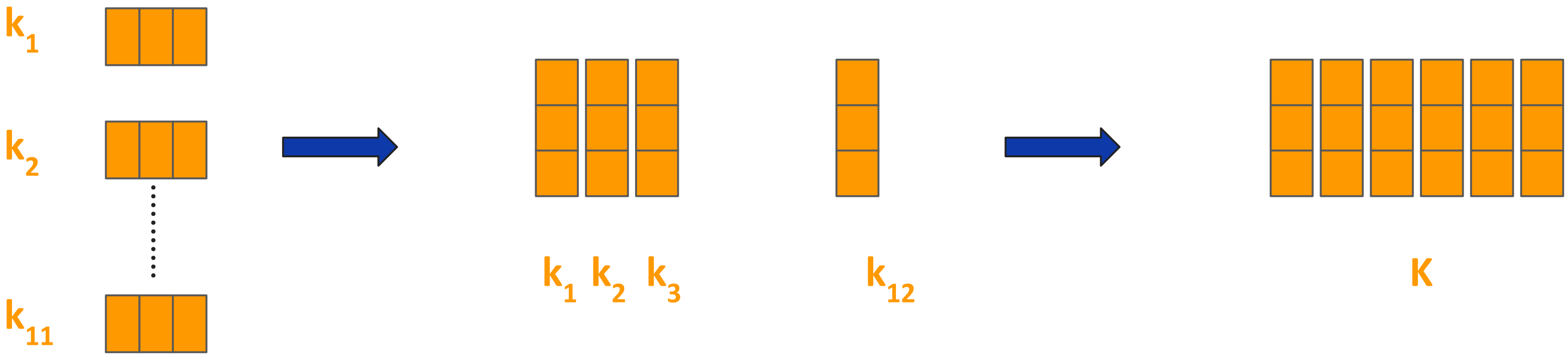
This will allows us to **compute** the context-aware representations of all words **at one go**



Computing Self Attention

We can ‘**stack**’ the vector representations of all words together into a **matrix**

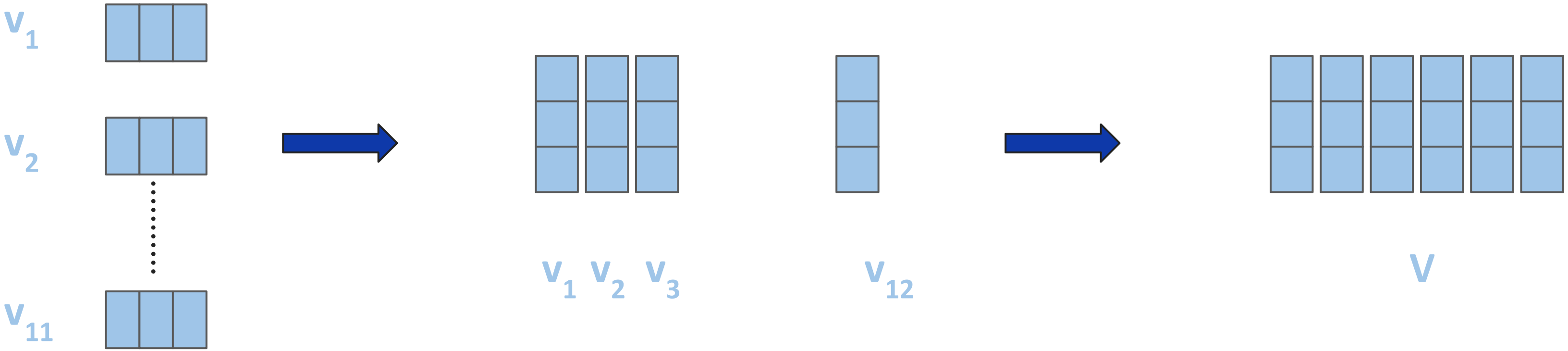
This will allows us to **compute** the context-aware representations of all words **at one go**



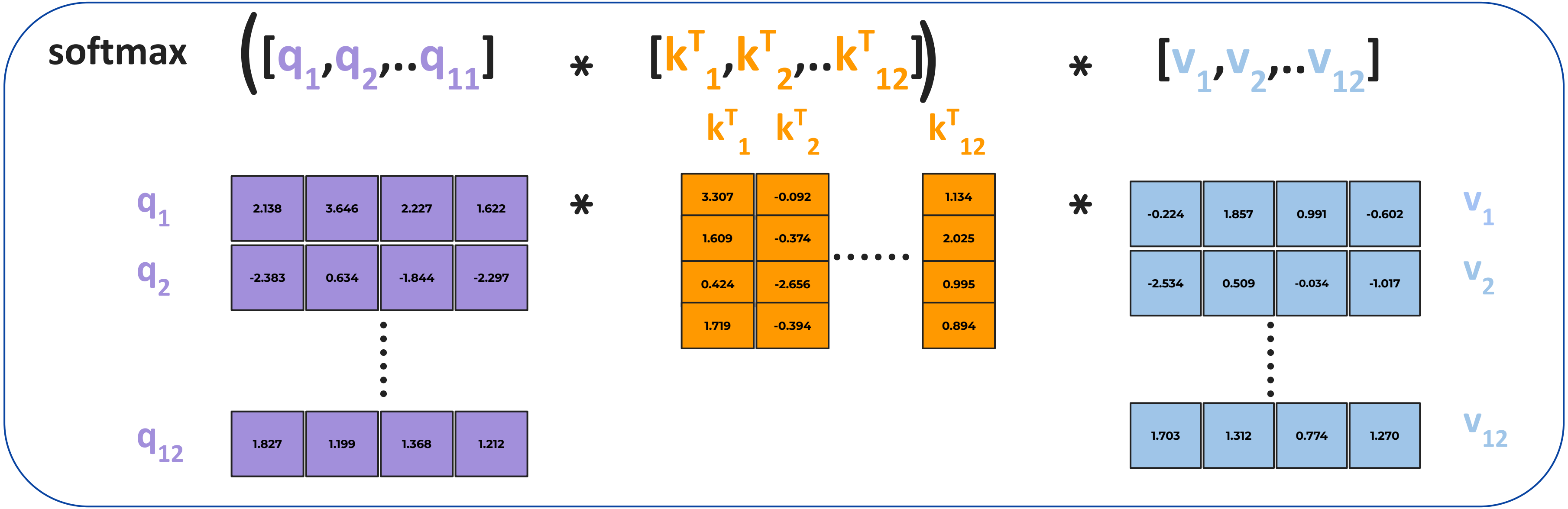
Computing Self Attention

We can ‘**stack**’ the vector representations of all words together into a **matrix**

This will allows us to **compute** the context-aware representations of all words **at one go**



Computing Self Attention



$$\text{softmax}\left([q_1,q_2,..q_{11}] \ * \ [k^T_1,k^T_2,..k^T_{12}]\right) \ * \ [v_1,v_2,..v_{12}]$$

q_1	2.138	3.646	2.227	1.622
q_2	-2.383	0.634	-1.844	-2.297
⋮				
q_{12}	1.827	1.199	1.368	1.212

*

k^T_1	k^T_2	……	k^T_{12}
3.307	-0.092		1.134
1.609	-0.374		2.025
0.424	-2.656		0.995
1.719	-0.394		0.894

*

-0.224	1.857	0.991	-0.602	v_1
-2.534	0.509	-0.034	-1.017	v_2
⋮				
1.703	1.312	0.774	1.270	v_{12}

$$\text{softmax}\left(Q \cdot K^T \right) * V$$

Computing Self Attention

Matrix multiplications are **very fast** and **efficient** way of **computation**

In practice, a **scaling factor d_k** is used for **smoother computation** and **better performance**

$$\mathrm{softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) * V$$

d_k here refers to the dimension of the vectors used for representing the input - we used d_k=4

Self Attention - Summary

Self attention allows us to **focus** on each **part of the sentence**

There is **no form of memory** here like we had in RNNs

Long term dependencies are captured by directly relating words in the sentence

Computing self attention for one word has no dependency on another word

All the computations can be done simultaneously (i.e., in parallel)

The **self-attention mechanism** lies at the **core** of **transformer models**

Transformer Models

The Basics of Transformer Models

Transformers are a **type of neural network architecture**

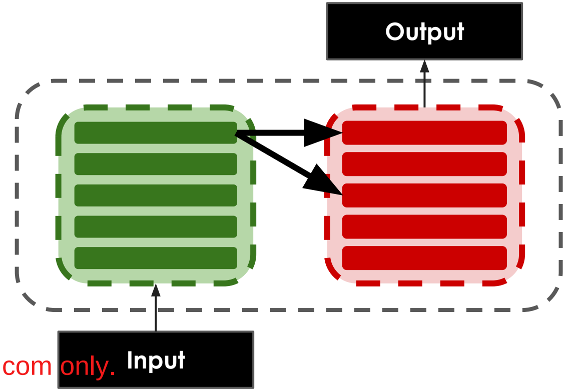
Transformers were introduced in a paper by **Vaswani et al. in 2017**

Transformers are based on the idea of **self-attention**

Transformers consist of an **encoder** and a **decoder**

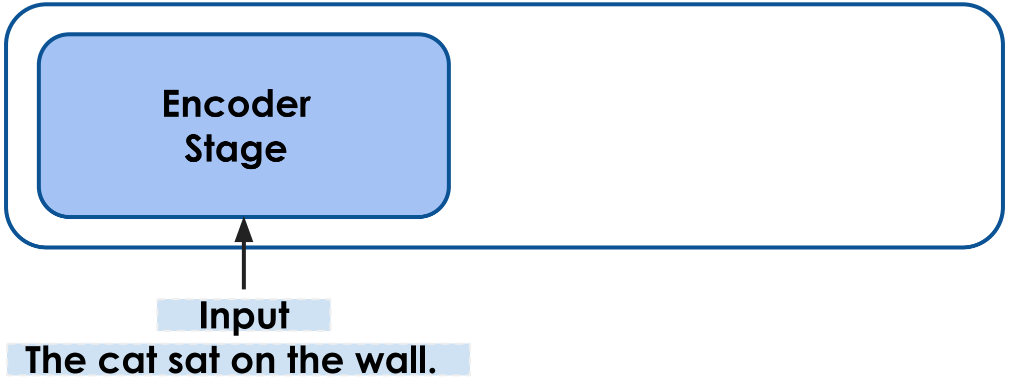
The **encoder** takes in a sequence of tokens (e.g. words or characters) and outputs a **latent representation**

The **decoder** then takes this latent representation as input and outputs a **sequence of tokens**



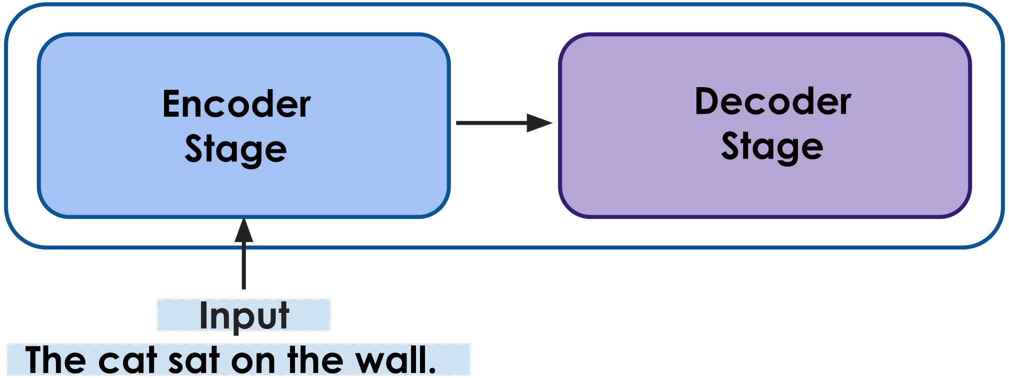
The Transformer Model - High-level Flow

The way this would work is **an input sequence is first passed to the Encoder stage** of the Transformer



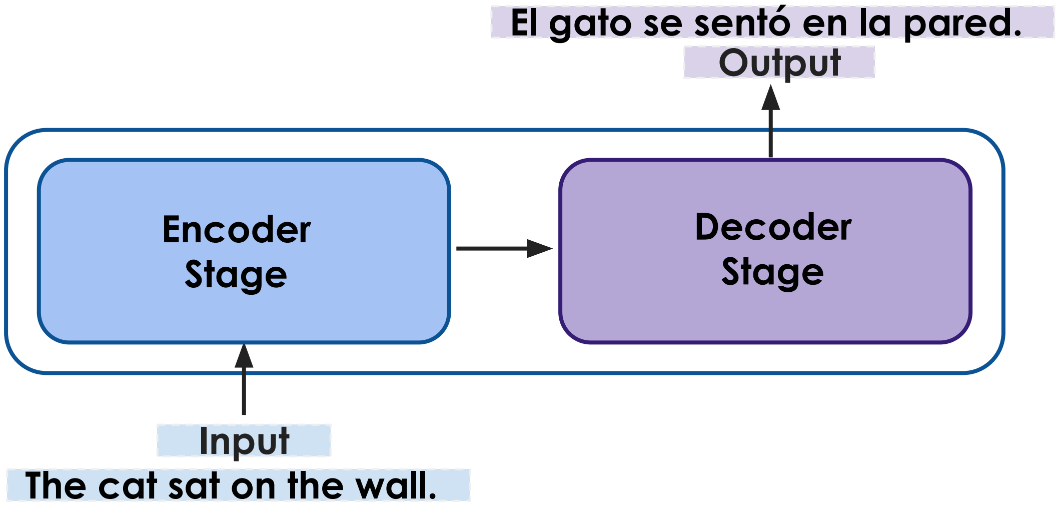
The Transformer Model - High-level Flow

The **Encoder stage's operations** eventually compute a **high-quality representation of the input** sequence, which has captured its **syntactical & semantic meaning**



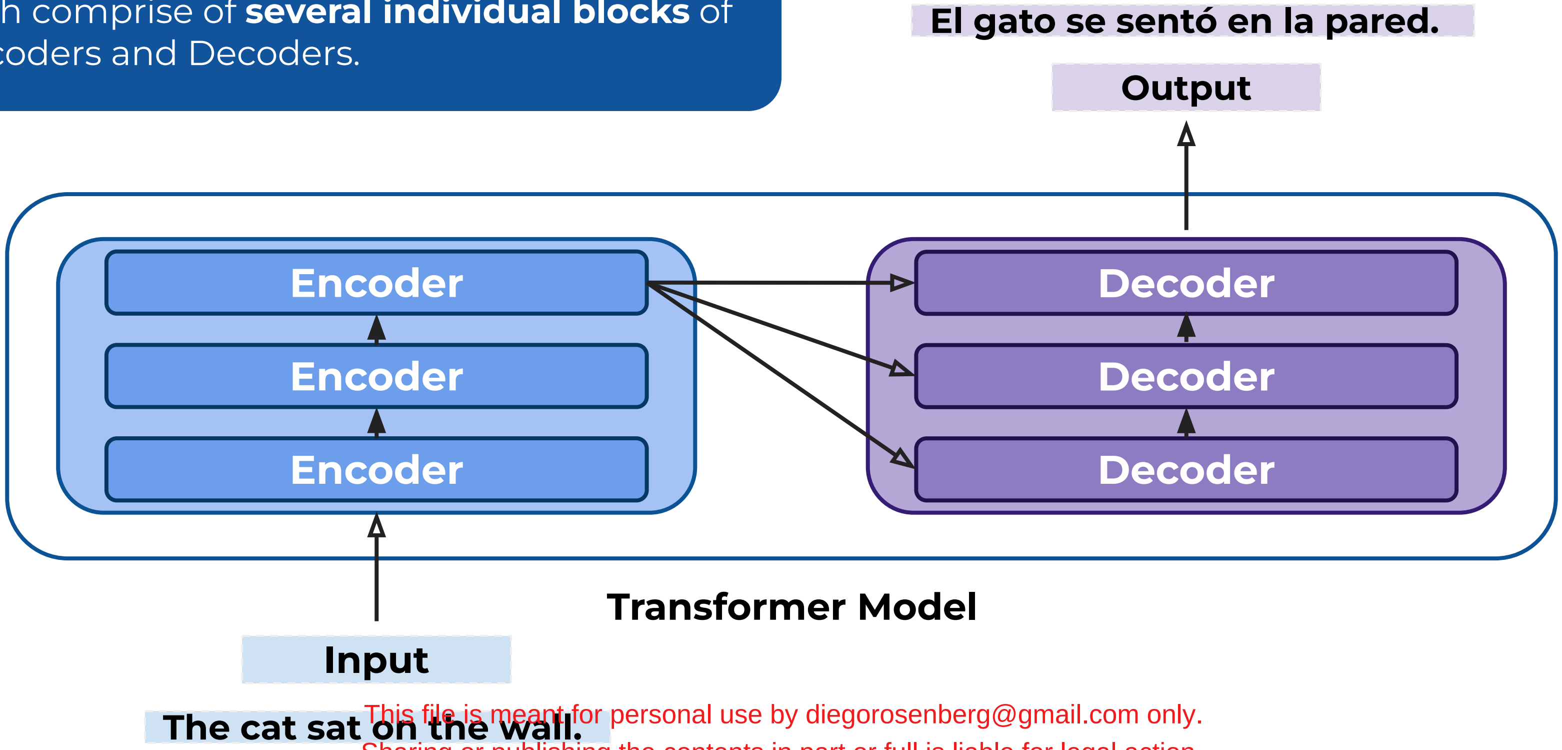
The Transformer Model - High-level Flow

The **Decoder stage** is responsible for eventually **“decoding” this representation** to a different sentence, in other words, converting it to the output needed


El gato se sentó en la pared.
Output
Encoder Stage
Decoder Stage
Input
The cat sat on the wall.

The Transformer Model - High-level Flow

In reality, the **Encoder** and **Decoder** stage each comprise of **several individual blocks** of Encoders and Decoders.



Transformer Architecture - Encoder Block

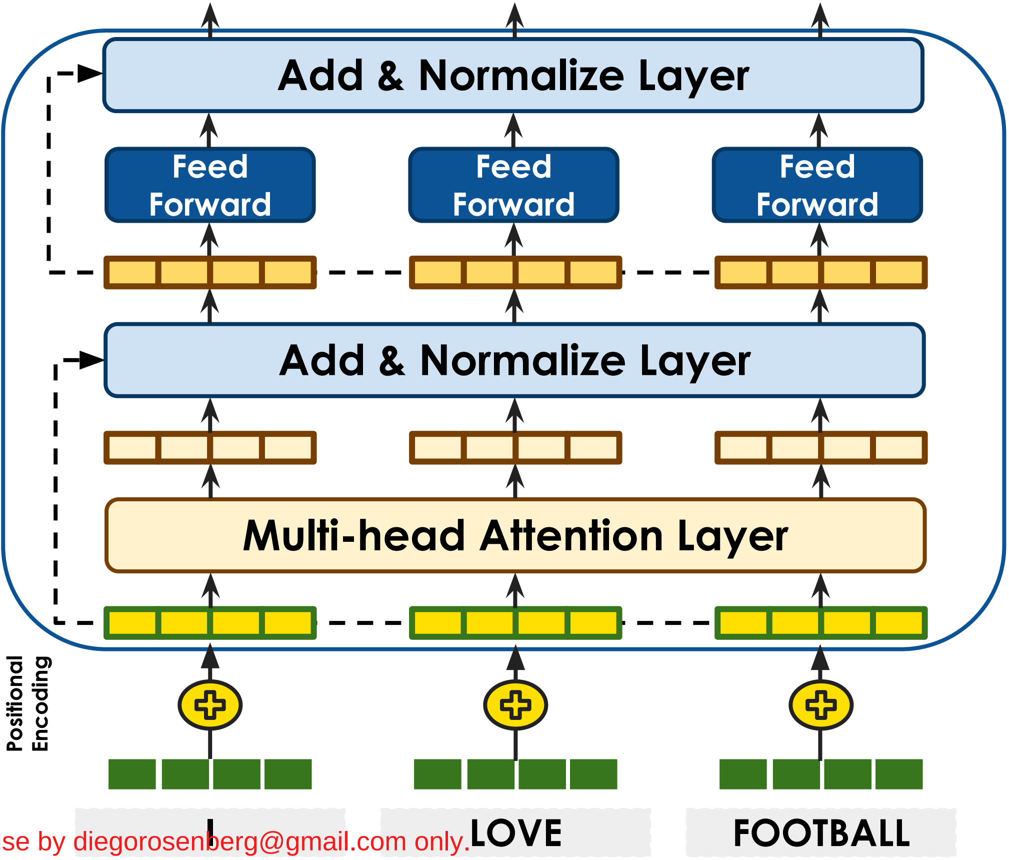
The Encoder block of a Transformer architecture consists of the following components:

1. **Multi-head Attention**: A stack of self-attention layers that allows the Encoder to attend to different parts of the input sequence simultaneously.

2. **Feedforward Neural Network**: Processes the outputs of the Multi-head Attention layer using a standard fully connected neural network with activations like ReLU.

3. **Residual Connections and Layer Normalization**: Improves the flow of information through the Encoder and avoids the vanishing gradient problem. These are added after each sub-layer.

4. **Positional Encoding**: Typically added to the input embeddings of the Encoder to provide positional information for words, using a set of learned sinusoidal functions.



The Need for Multi-Head Attention

Let’s go back to one of our previous examples

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

Now consider the following sentence

The animal didn’t cross the street because it was congested.

In the **first** sentence, **‘it’** is referring to **‘animal’**, while in the **second** one, **‘it’** is referring to **‘street’**

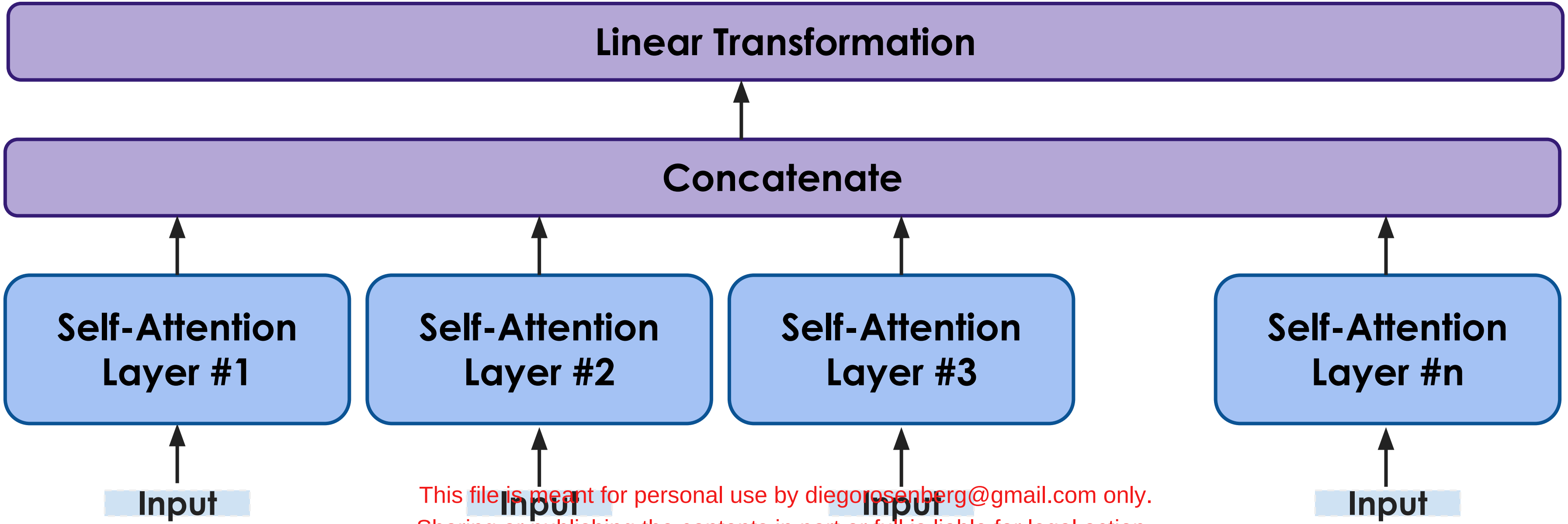
A **single self-attention** layer might **not** be able to **capture these nuances**

So, we use **multiple self-attention** layers - a **multi-head attention** layer

Multi-Head Attention

The output of each self-attention layer is taken and concatenated

The linear transformation layer is merely a fully-connected layer of neurons



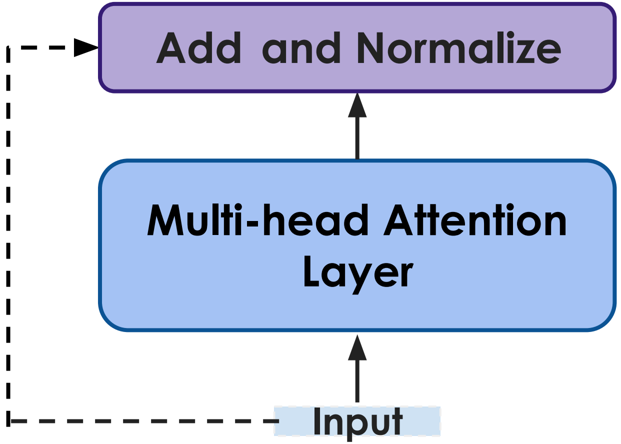
Residual Connections and Layer Normalization

Residual connections, also known as skip connections, are pathways that allow the input of a certain layer to bypass that layer and be directly added to the output of subsequent layers

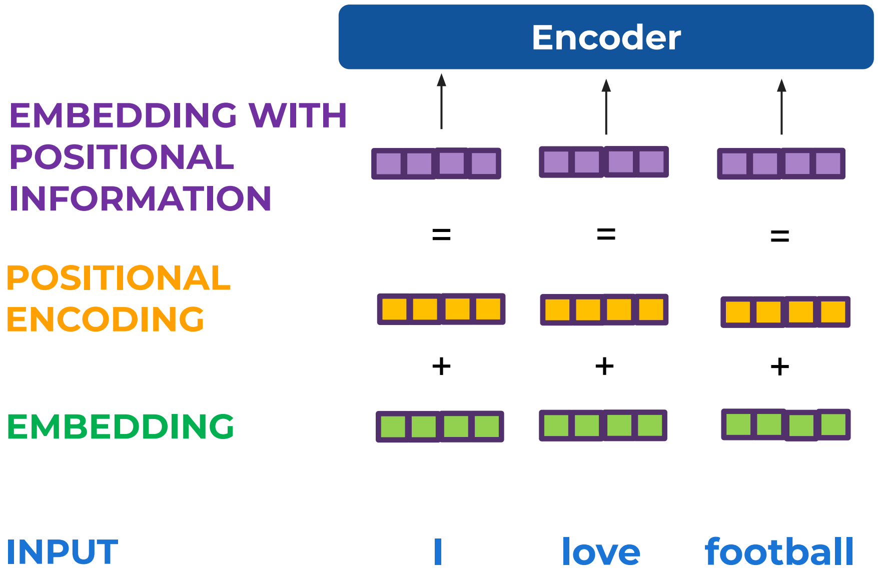
The residual connections always "remind" the representation of what the original state was

This kind of ensures contextual representations of the input tokens really represent the tokens

Normalization ensures that the inputs for each layer is on the same scale - enables **smoother computation** and **better performance**



Positional Encoding



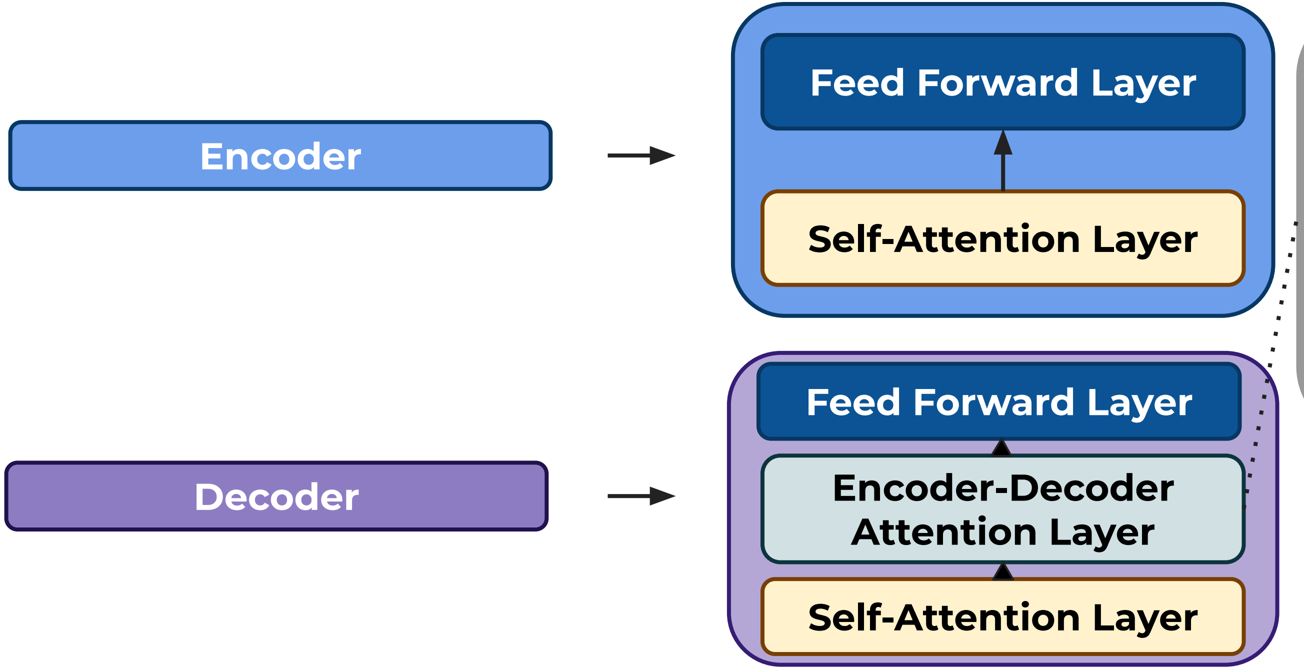
Positional Encoding is a way to account for **the order of the words in the input sequence.**

Positional Encoding is a vector added to each input embedding.

These vectors follow a specific pattern that the model learns, which helps it determine the position of each word, or the distance between different words in the sequence.

The Encoder vs. The Decoder

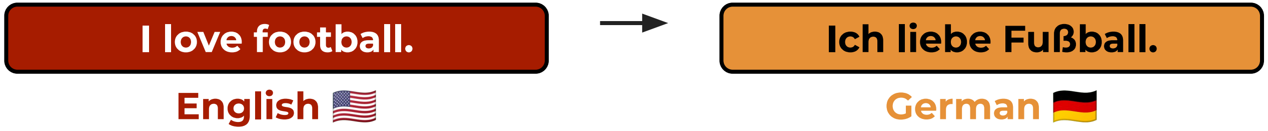
At a high level, the Decoder **only slightly differs** from the constitution of the Encoder.



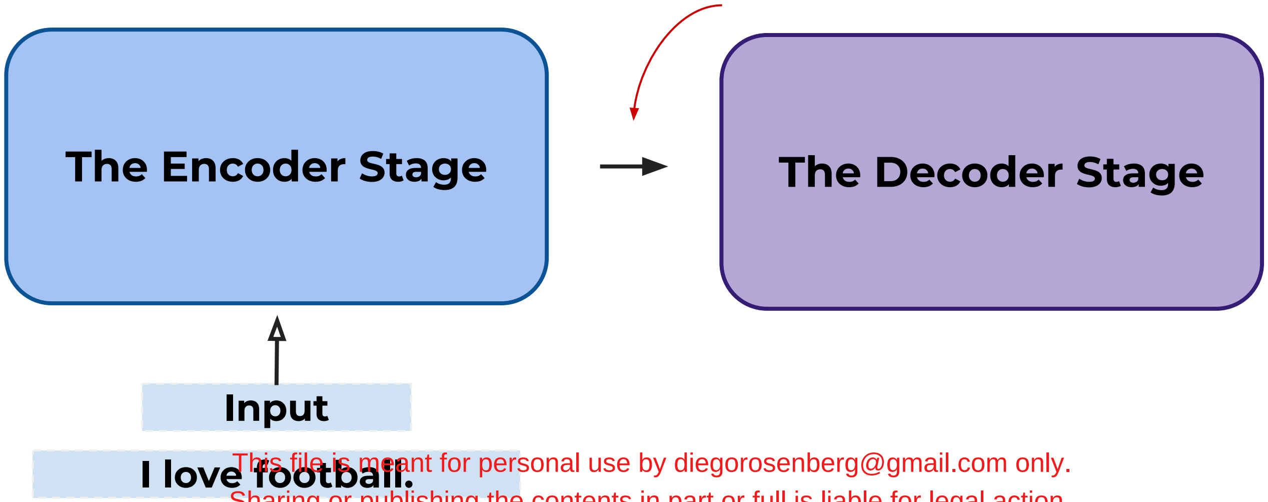
This additional Encoder-Decoder Attention Layer of the Decoder, allows the Decoder to specifically focus on certain parts of the input it receives from the Encoder, in order to generate its output.

A Peek into the Decoder

Let's assume we're creating this Encoder-Decoder architecture for an **English-to-German Machine Translation task.**



Also, let's remember the Decoder operations start at the point **where the pass through the Encoder Stage has been completed.**



A Peek into the Decoder

We see immediately that most of these operations are identical to the Encoder.

Self-Attention Layer

Add & Normalize Layer

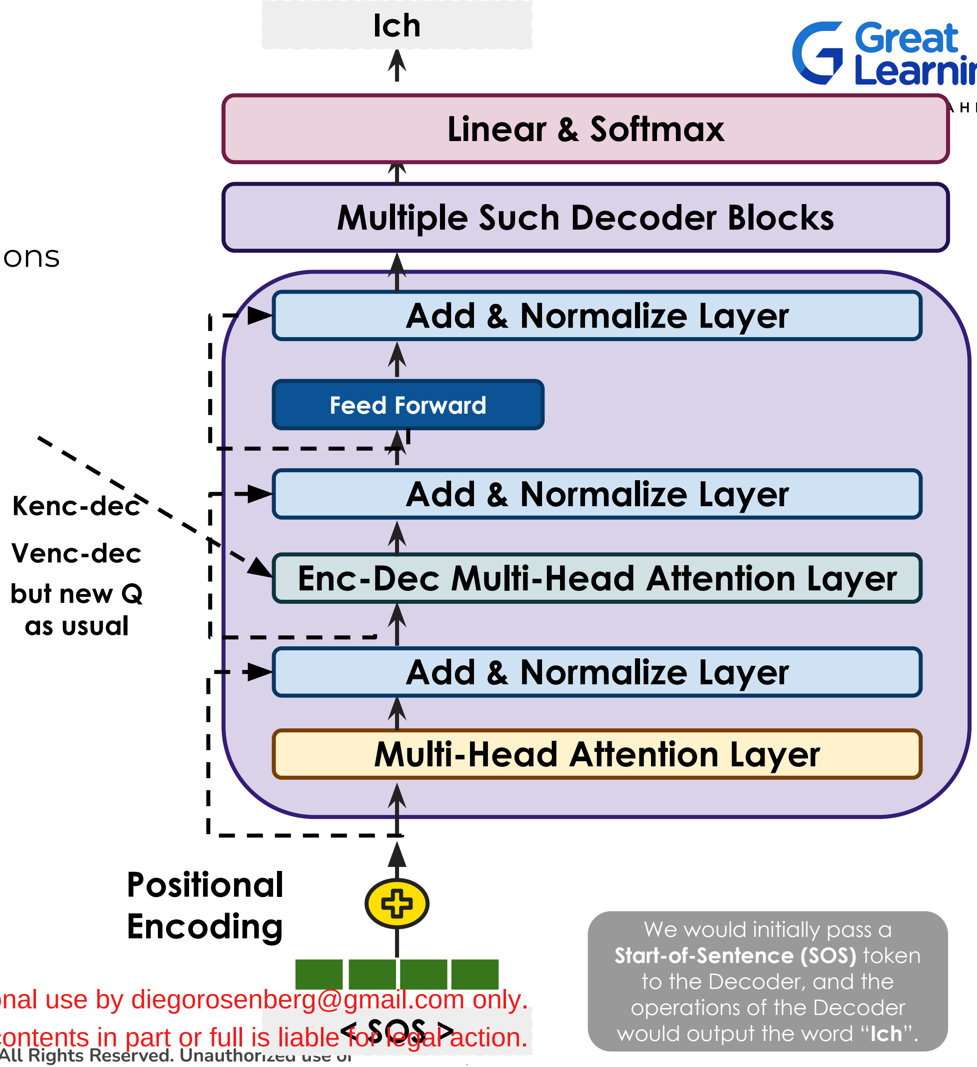
Feed Forward

But there are a few other operations **unique to the Decoder.**

Encoder-Decoder Attention Layer

Linear & Softmax

Let's understand these differences in some more detail.



We would initially pass a **Start-of-Sentence (SOS)** token to the Decoder, and the operations of the Decoder would output the word "**Ich**".

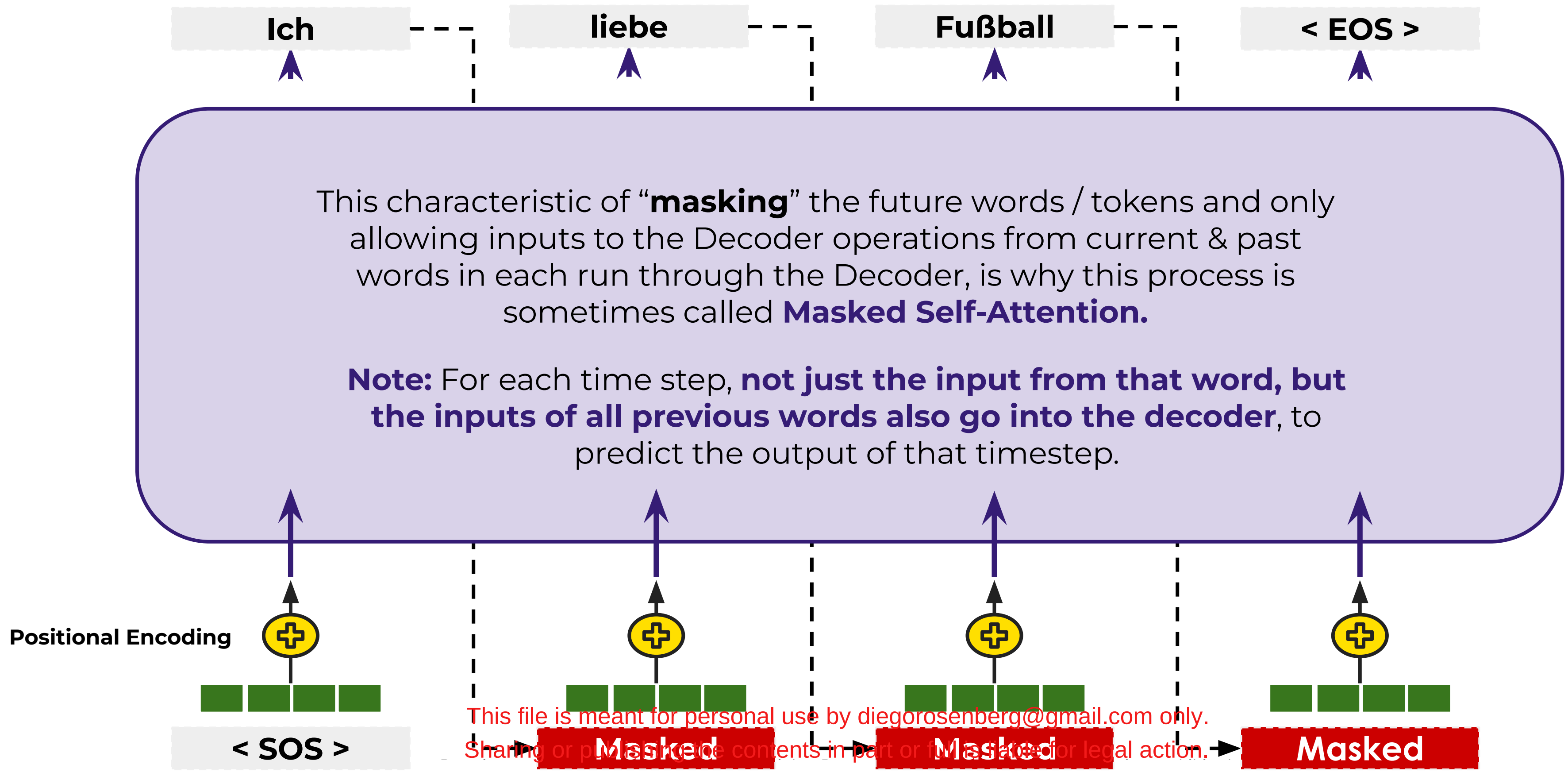
The Decoder’s Sequential Nature (Masked Self-Attention)

The first difference to note is that unlike the Encoder, where all the words pass through the Encoder block in parallel, **the Decoder is Sequential in nature**, similar to how we know RNNs operate.

Starting with the Start of Sentence <SOS> token, **the Decoder takes a previous word & generates one word at a time**, until it understands it has generated the last word of the sentence, in which case it generates the End of Sentence <EOS> token.

This sequential word-by-word process of the Decoder’s text generation makes **the Decoder training stage much more time consuming than that of the Encoder**, and more difficult to parallelize as well.

The Decoder’s Sequential Nature (Masked Self-Attention)

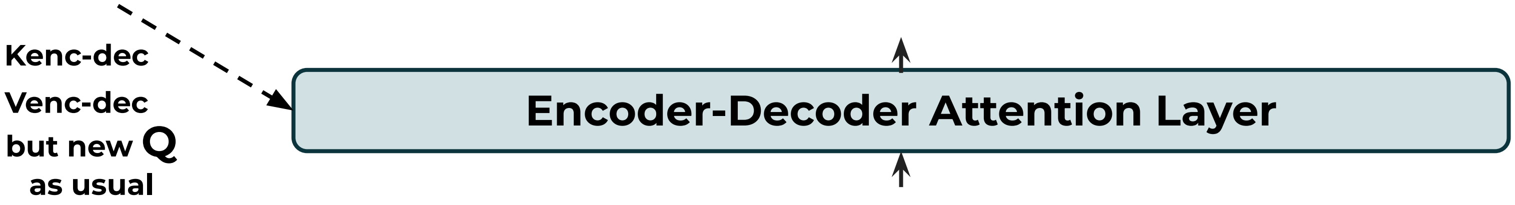


This characteristic of “**masking**” the future words / tokens and only allowing inputs to the Decoder operations from current & past words in each run through the Decoder, is why this process is sometimes called **Masked Self-Attention.**

Note: For each time step, **not just the input from that word, but the inputs of all previous words also go into the decoder**, to predict the output of that timestep.

The Encoder-Decoder Attention Layer

The other major difference is, of course, the **Encoder-Decoder Attention Layer.**



Kenc-dec

Venc-dec

but new **Q**

as usual

Encoder-Decoder Attention Layer

The difference from normal Self-Attention is that in this layer, **the K and V vectors are not generated from the input embeddings to this layer**, the way they were in the normal Self-Attention layer.

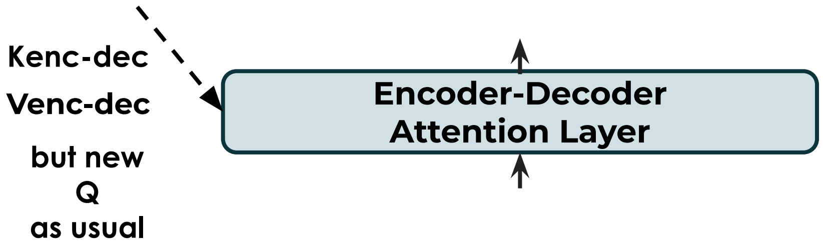
In fact, we utilize a **K encoder-decoder (K enc-dec)** and a **V encoder-decoder (V enc-dec)** in this layer, whose source is from the **final output of the Encoder stage**.

The Encoder-Decoder Attention Layer

We directly utilize **the final embedding vectors** generated at the end of the Encoder stage, and multiply those with weight matrices to get **K enc-dec** & **V enc-dec**. These get used as K and V in this Encoder-Decoder Attention Layer.

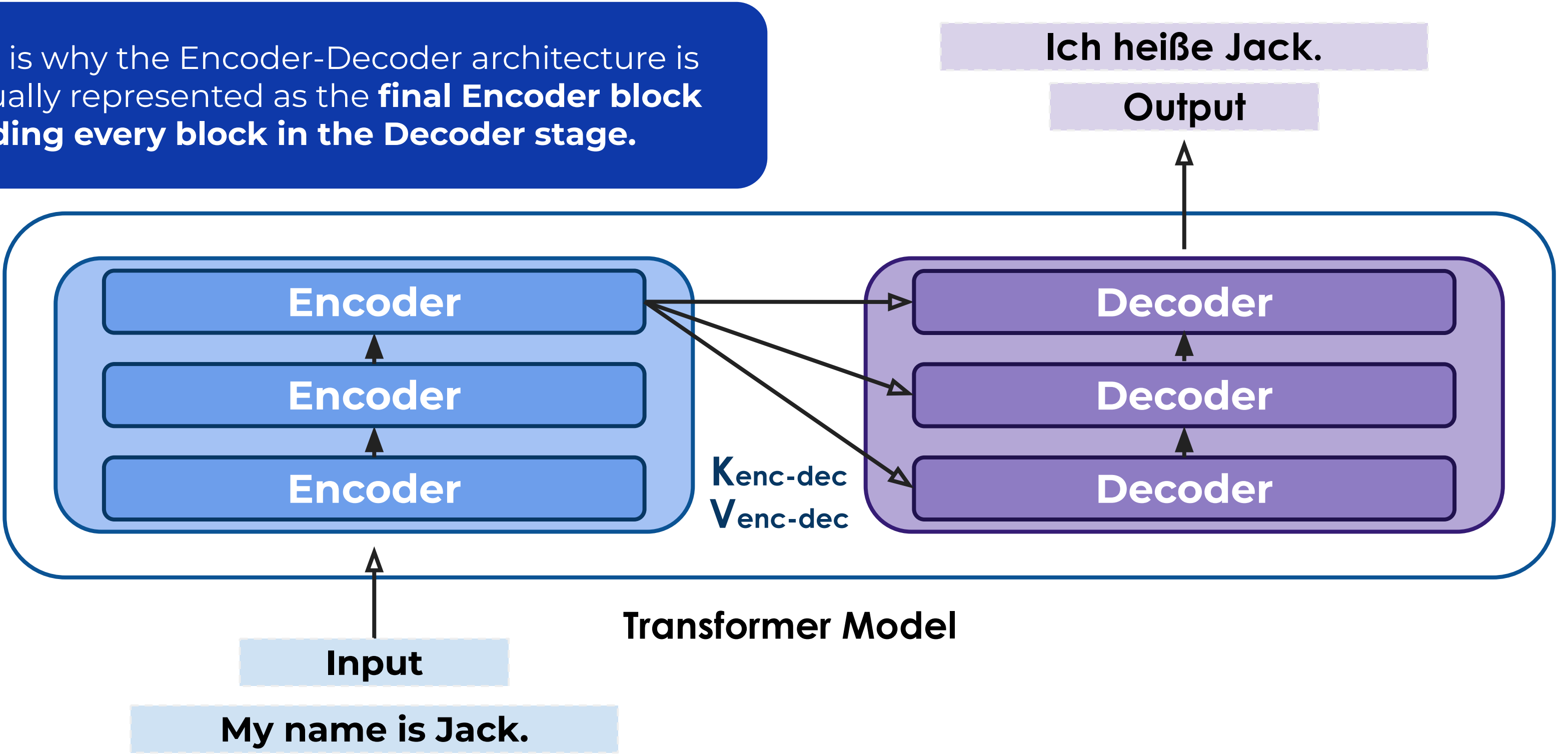
It is only the Q vector that this layer creates from the input to it, the way that normally happens in the Self-Attention Layer (where all three of K, Q & V are directly created from the input embeddings to the layer).

It is also important to mention, that the **Q for < SOS >** (Dec Pos 0) for example, **only relies on the K enc-dec & V enc-dec of the word "I"** (Enc Pos 1) from the input, to predict the word "Ich".
This happens for every Decoder word.



The Encoder-Decoder Attention Layer

This is why the Encoder-Decoder architecture is actually represented as the **final Encoder block feeding every block in the Decoder stage.**



The arrows from the final Encoder block to each Decoder block represent the **K enc-dec & V enc-dec from the final Encoder layer being used in the Encoder-Decoder Attention Layer of each Decoder block** in the Decoder stage.

The Linear & Softmax Layers

At the end of the Decoder stage, there's a **Linear and Softmax** layer that performs a fairly simple operation needed to get the final word prediction.

Ich
Linear & Softmax

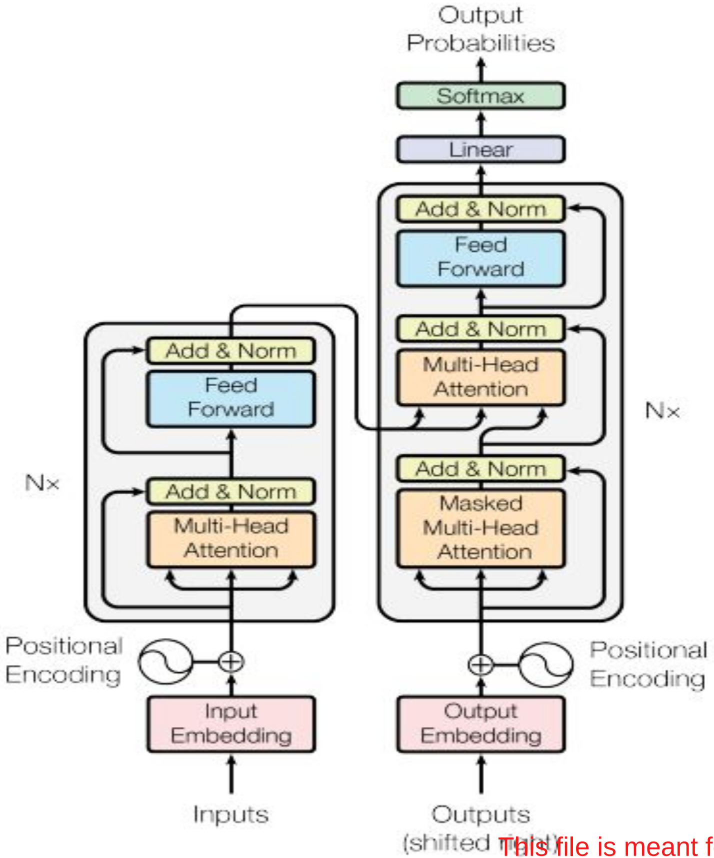
The **c** in addition to some special tokens.

This is then fed to the final Softmax layer, which converts the numerical outputs into probabilities, so that **the word with the highest probability can be selected as the output of the Decoder**, in the style of a **multi-class Classification problem.**

Finally, **Categorical Cross-Entropy** is the loss function used for backpropagation.

This construct is called the **Language Model Head**, and this is how **the Decoder eventually generates a word at each sequential time step**!

Bringing It All Together



The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder

Source: Image from the original research paper Attention Is All You Need

Transformer Models - Summary

Transformers are a **type of neural network architecture,** which consist of an **encoder** and a **decoder.** The **encoder** takes in a sequence of tokens and outputs a **latent representation,** while **decoder** then takes this latent representation as input and outputs a **sequence of tokens.**

The encoder consists of several components - positional encoding (providing positional information for words), multi-head attention (facilitating the transformer's understanding of various relationships between words), residual connections (for smoother computation), and feedforward network (for a linear transformation)

The decoder functions similarly to the encoder, yet it involves some different operations - masking (used to hide relations between next tokens to predict), encoder-decoder attention (where the keys and values are computed from the encoders output), and softmax layer (to select the token with the highest probability as output)

Transformers have revolutionized NLP, demonstrating state-of-the-art performance across multiple tasks like machine translation, sentiment analysis, and document summarization.

Great Learning

Happy Learning !