The Transformer Model

Beatriz Lourenço

Universidade de Lisboa, Instituto Superior Técnico

beatriz.p.lourenco@tecnico.ulisboa.pt

Lecture Plan

- Introduction to the problem
- From recurrence (RNN) to attention-based NLP models
- Introducing the Transformer model
- Great results with Transformers
- Drawbacks of the Transformers

References

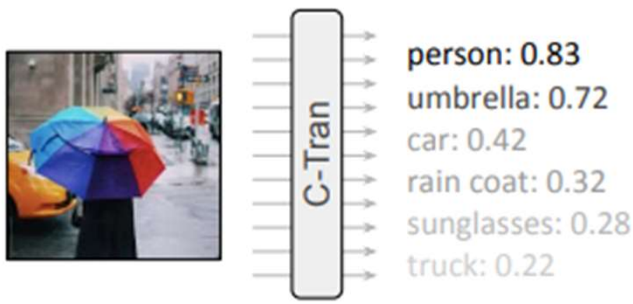
- John Hewitt's lecture on self-attention and transformers
- Stanford Seminar: Introduction to Transformers
- LSTM is dead. Long Live Transformers!
- Illustrated Transformer
- The annotated Transformer
- The original Transformer paper
- BLEU (BiLingual Evaluation Understudy)
- IEEE blog

Lecture Plan

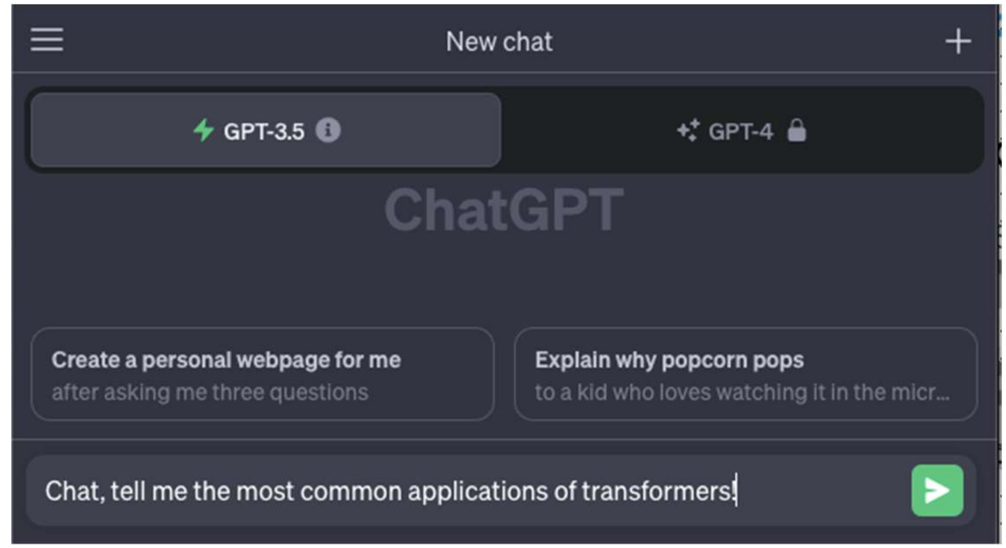
- Introduction to the problem
- From recurrence (RNN) to attention-based NLP models
- Introducing the Transformer model
- Great results with Transformers
- Drawbacks of the Transformers

TÉCNICO LISBOA

What are Transformers?



C-Tran model. Image: [paper]



GPT-3 model

DALL-E model. Image: [OpenAI website]

Other Applications

- Natural Language Processing
  1. Machine Translation
  2. Language Generation
  3. Sentiment Analysis
- Speech Recognition
- Computer Vision
- Reinforcement Learning
- Etc.

Original Problem: Machine Translation

Goal: Translate a **source sentence** x into a **target sentence** y in another language

Example (Portuguese to English):

x: “De grão em grão a galinha enche o papo”

y: “Grain by grain the chicken fills her belly”

Neural Machine Translation: the first big success story of NLP Deep Learning

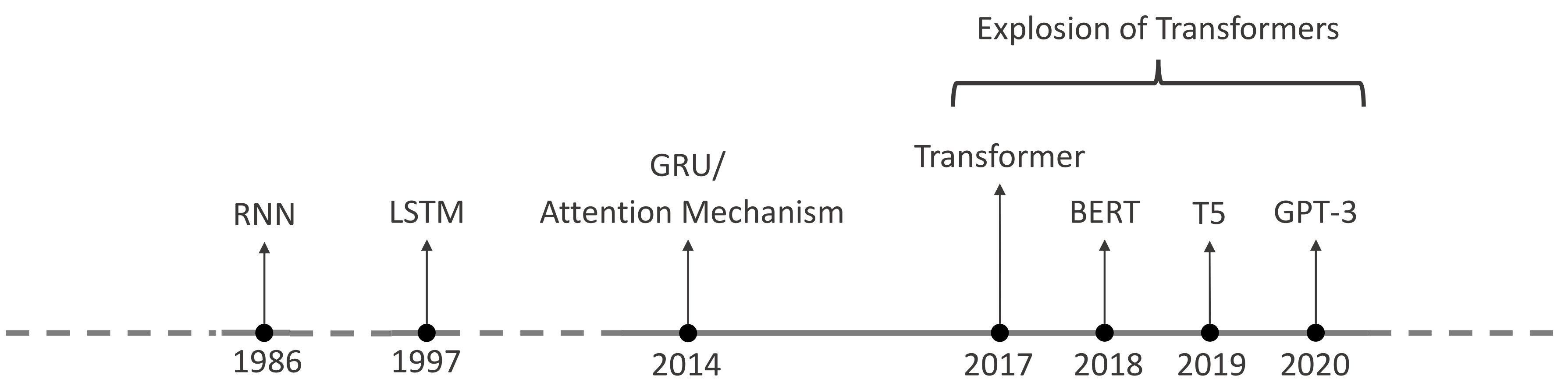
From Statistical Machine Translation (SMT) to Neural Machine Translation (NMT)

- 2014: First seq2seq paper published
- 2016: Google Translate switches from SMT to NMT (and by 2018 everyone has)
- SMT systems, built by **hundreds of engineers over many years**, outperformed by
- NMT systems trained by **small groups of engineers** in a few months

Lecture Plan

- Introduction to the problem
- From recurrence (RNN) to attention-based NLP models
- Introducing the Transformer model
- Great results with Transformers
- Drawbacks and variants of Transformers
- Other Applications

Timeline


Explosion of Transformers
RNN
LSTM
GRU/
Attention Mechanism
Transformer
BERT
T5
GPT-3
1986
1997
2014
2017
2018
2019
2020

Timeline

RNN

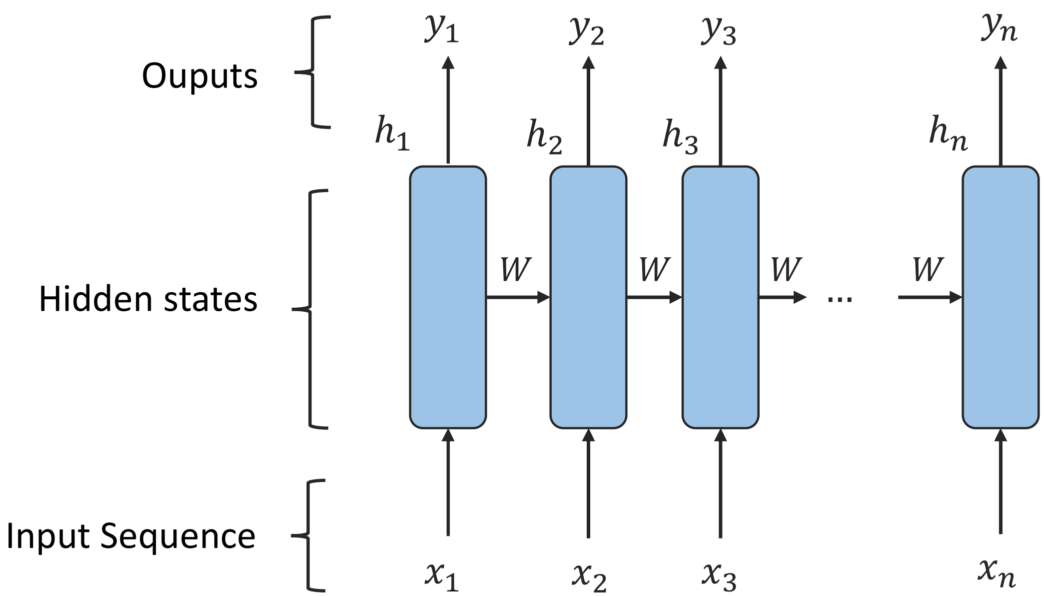
Gated Units: LSTMs and GRUs

Sequence-to-Sequence Architecture

Sequence-to-Sequence with Attention

Transformer Model

Recurrent Neural Networks



Main problem of the approach?

Vanishing/Exploding Gradient Problem

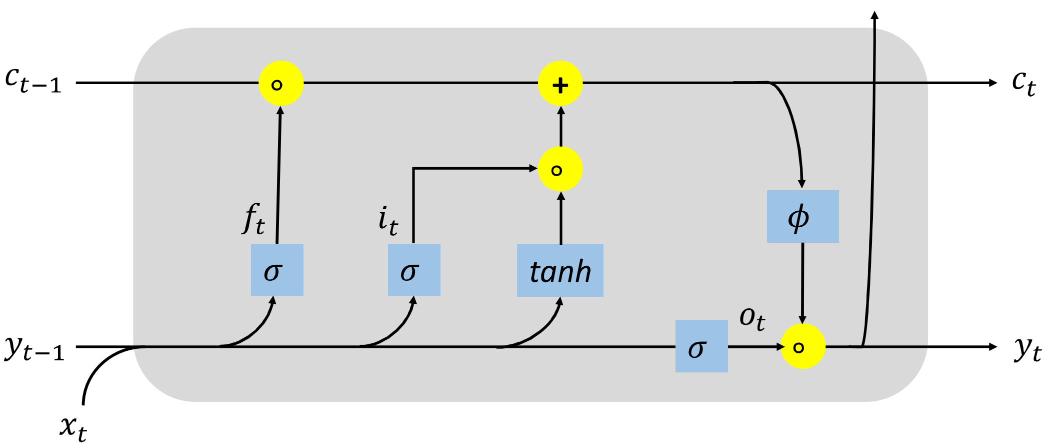
$$w_{i+1} = w_i + \eta \frac{\partial E}{\partial w}$$

Low $\frac{\partial E}{\partial w}$ and η ➡ Insignificant weight update

High $\frac{\partial E}{\partial w}$ and η ➡ Can cause bad updates

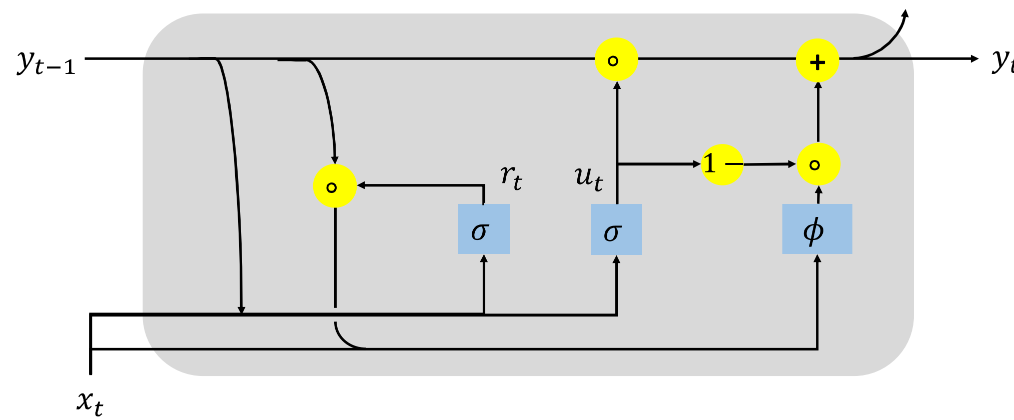
LSTM and GRU unit

Long Short-Term Memory



$o_t^{(l)}$ Output gate
$i_t^{(l)}$ Input gate
$f_t^{(l)}$ Forget gate
$c_t^{(l)}$ Update cell

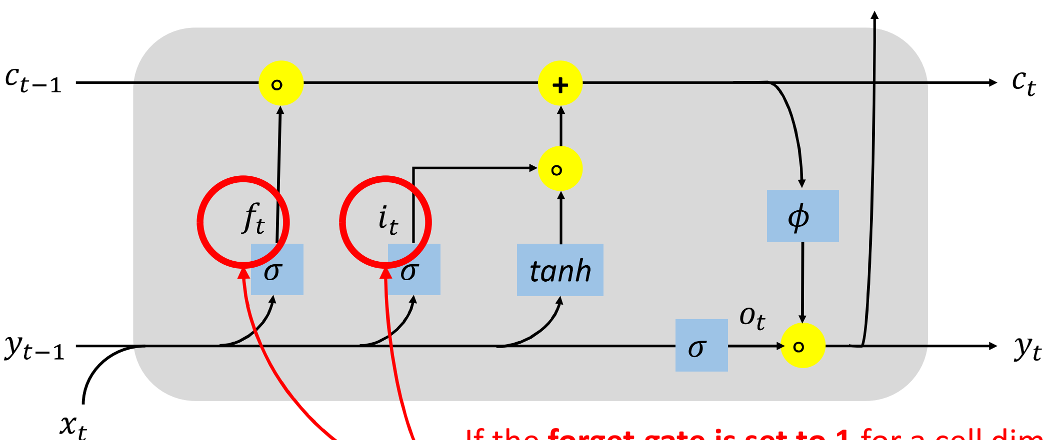
Gated Recurrent Unit



$r_t^{(l)}$ Reset gate
$u_t^{(l)}$ Update gate

LSTM and GRU unit

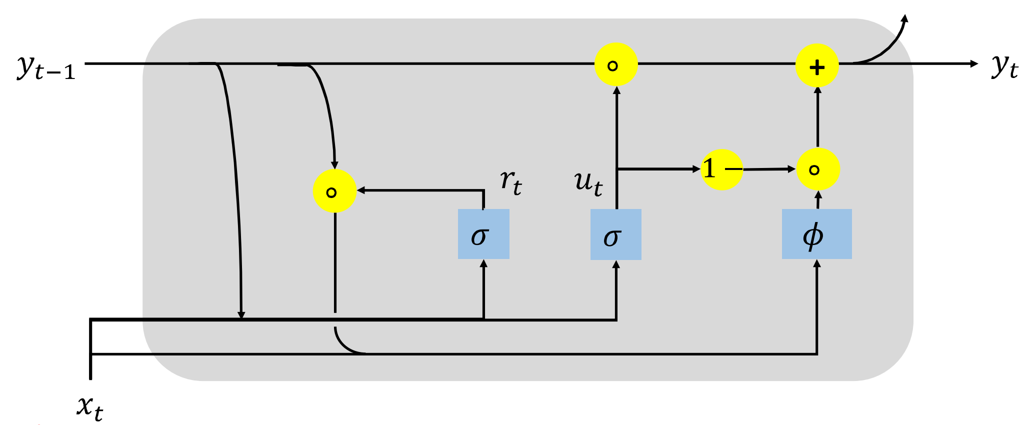
Long Short-Term Memory



If the **forget gate is set to 1** for a cell dimension and the **input gate set to 0**, the information of that cell is **preserved indefinitely**

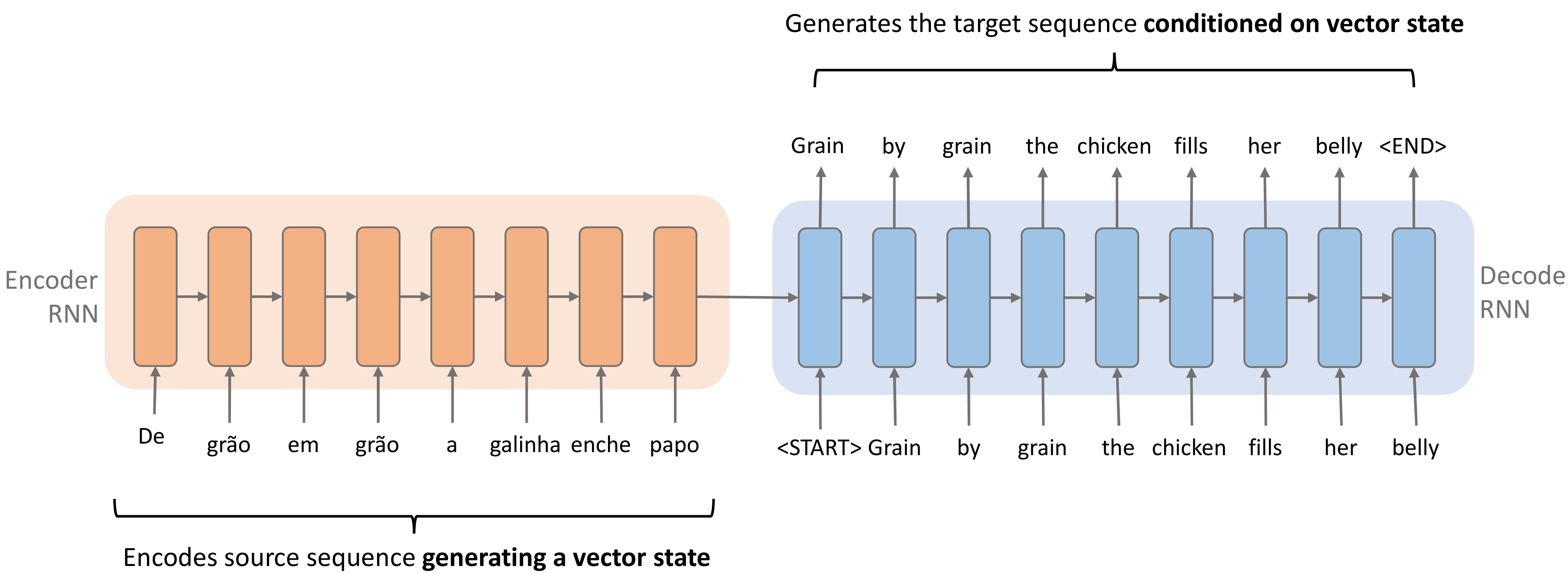
$o_t^{(l)}$ Output gate
$i_t^{(l)}$ Input gate
$f_t^{(l)}$ Forget gate
$c_t^{(l)}$ Update cell

Gated Recurrent Unit

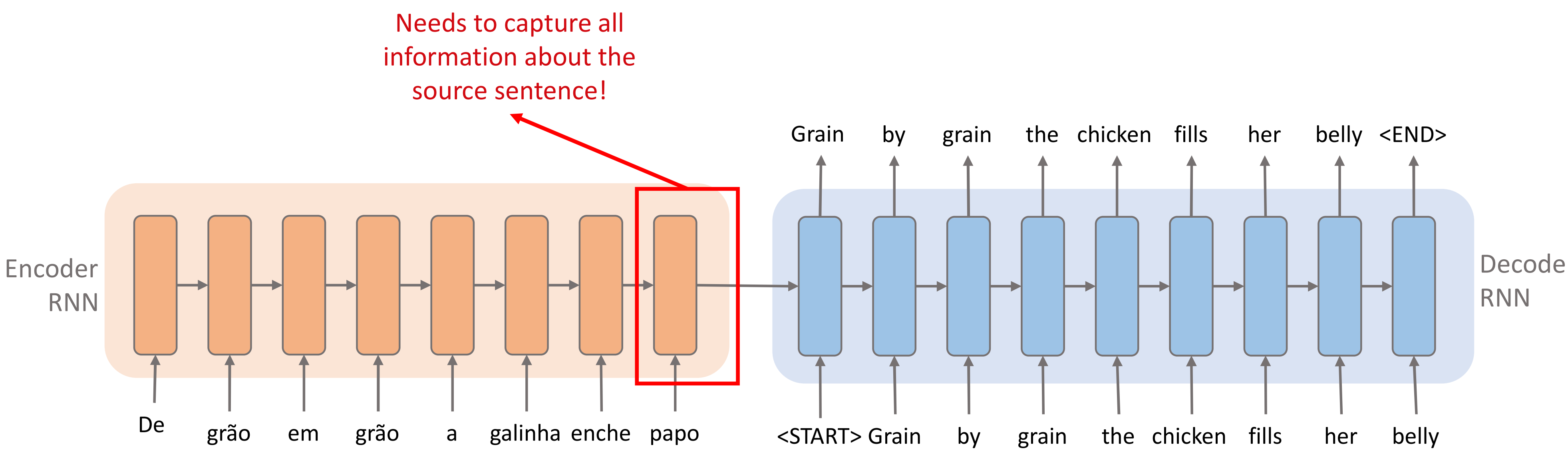


$r_t^{(l)}$ Reset gate
$u_t^{(l)}$ Update gate

Sequence-to-Sequence

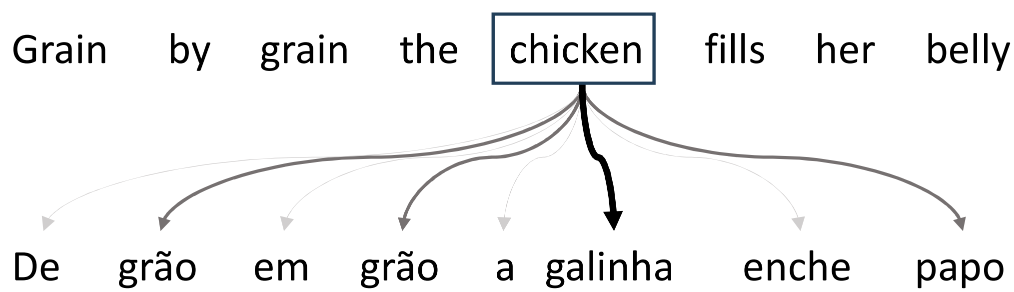


Sequence-to-Sequence: the Botteneck Problem

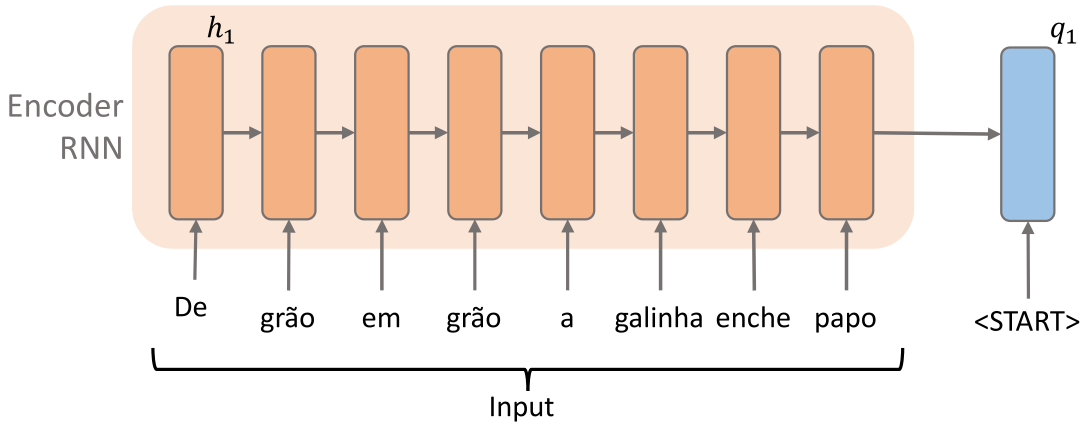


Solution: Attention

- Core idea: Enables the decoder to **focus on different parts of the input data** with **varying importance**

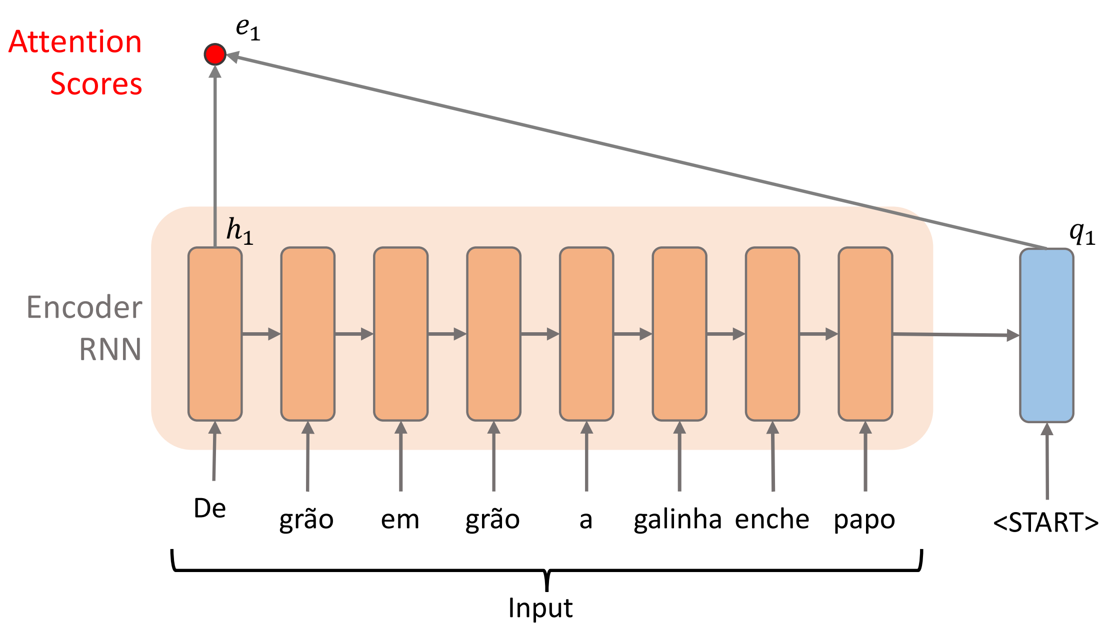


Sequence-to-Sequence with Attention



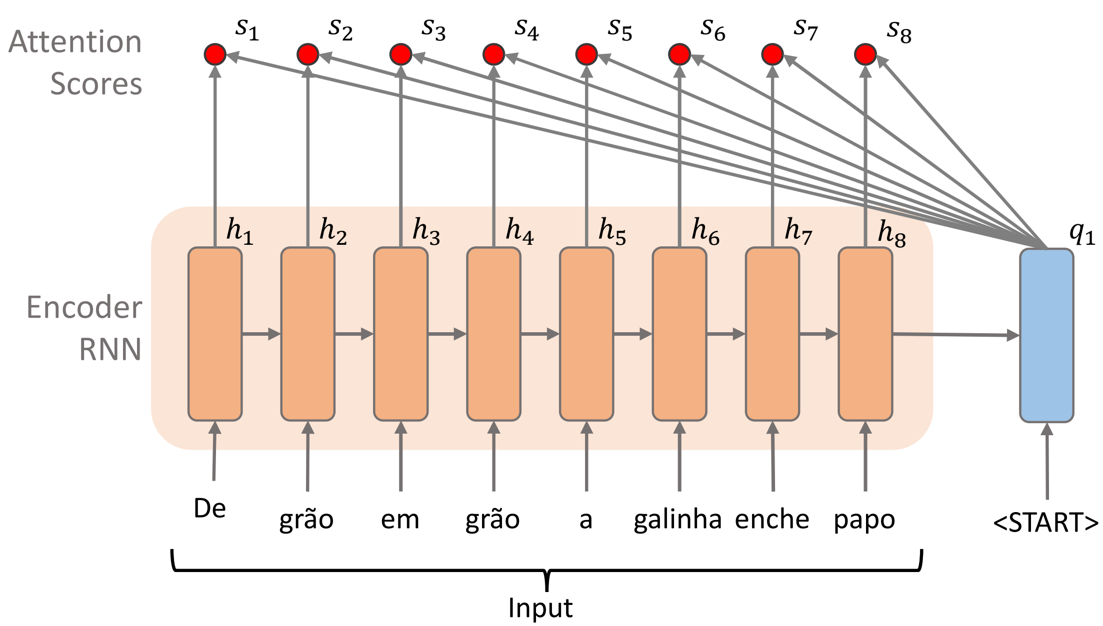
Sequence-to-Sequence with Attention

1. Computing the attention scores, $e \in \mathbb{R}^N$.
(You can use the basic dot-product attention)



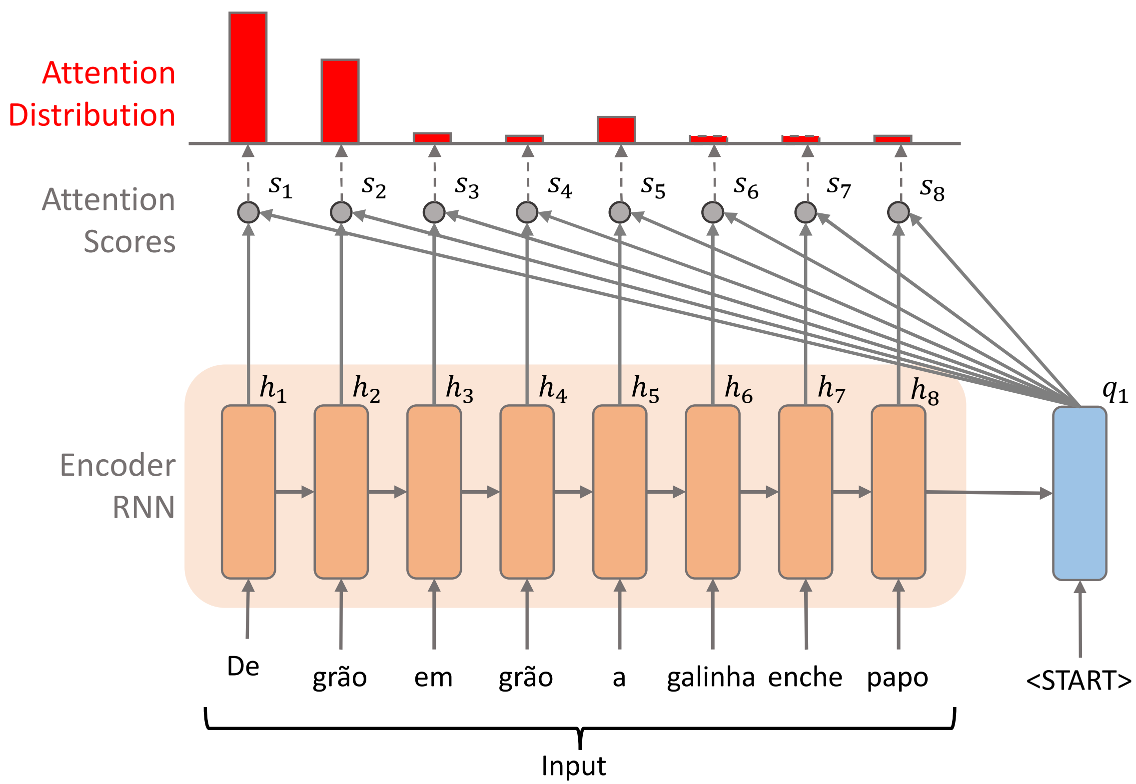
TÉCNICO LISBOA

Sequence-to-Sequence with Attention



1. Computing the attention scores, $s \in \mathbb{R}^L$.
(You can use the basic dot-product attention)

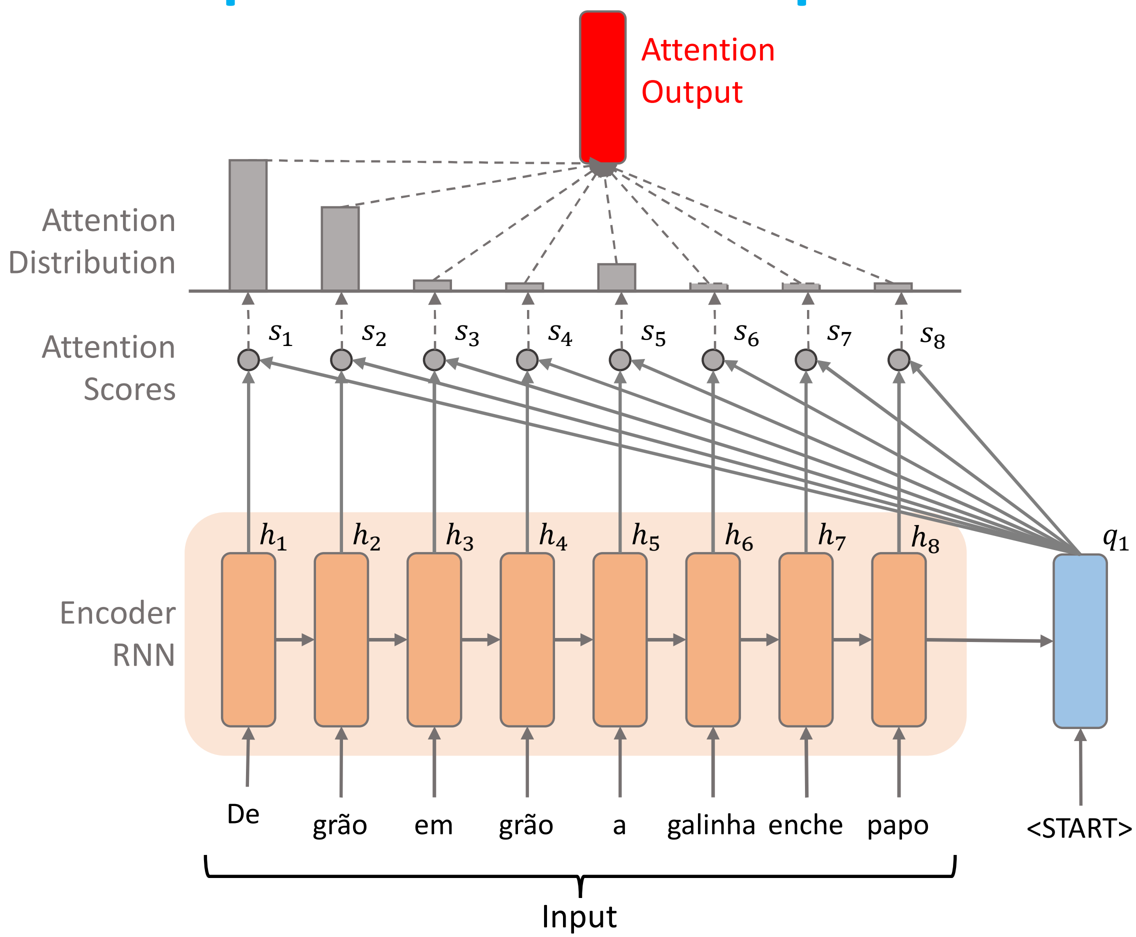
Sequence-to-Sequence with Attention



1. Computing the attention scores, $s \in \mathbb{R}^L$.
(You can use the basic dot-product attention)

2. Apply the softmax to get attention distribution, $\alpha \in \mathbb{R}^L$

Sequence-to-Sequence with Attention

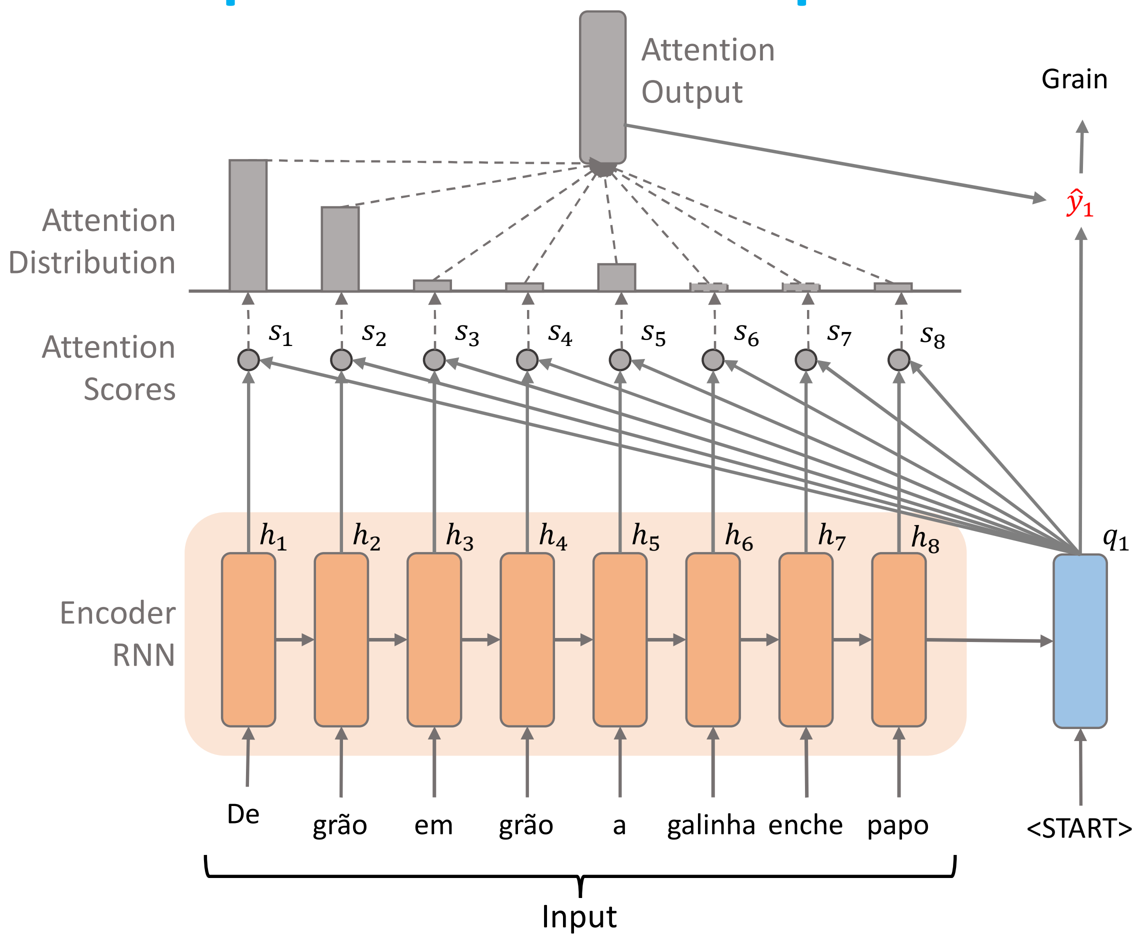


1. Computing the attention scores, $s \in \mathbb{R}^L$.
(You can use the basic dot-product attention)

2. Apply the softmax to get attention distribution, $\alpha \in \mathbb{R}^L$

3. Use the attention distribution to calculate the weighted sum of the values and get the attention output, $a \in \mathbb{R}^d$

Sequence-to-Sequence with Attention

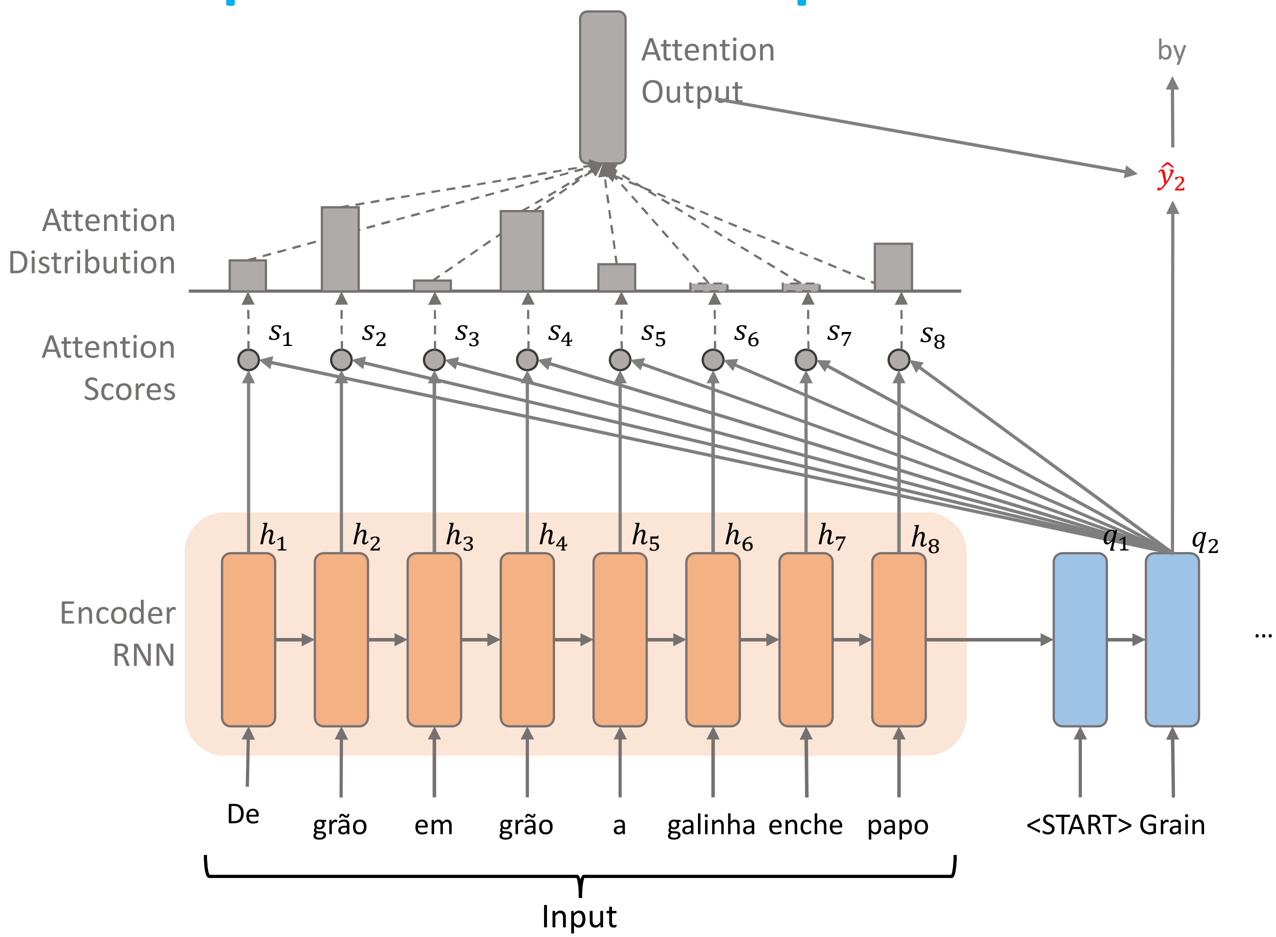


1. Computing the attention scores, $e \in \mathbb{R}^N$.
(You can use the basic dot-product attention)

2. Apply the softmax to get attention distribution, $\alpha \in \mathbb{R}^N$

3. Use the attention distribution to calculate the weighted sum of the values and get the attention output, $a \in \mathbb{R}^d$

Sequence-to-Sequence with Attention



Attention: in Equations

Query vector: $q \in \mathbb{R}^d$ (decoder state)

Input vectors: $\mathrm{H} = [h_1, \cdots, h_L]^T \in \mathbb{R}^{L \times d}$ (encoder states e.g. one state per source word)

1. Computing the **afinity scores**, $\mathrm{s} \in \mathbb{R}^L$, by *comparing* q and H.

There are multiple ways to do this!

2. Apply the softmax to get **attention distribution**, $\alpha \in \mathbb{R}^L$

$$\alpha = softmax(s)$$

3. Calculate the weighted sum of H with α as weights to get the **attention output**, $\mathrm{a} \in \mathbb{R}^d$

$$a = H^T \alpha = \sum_{i=1}^{N} \alpha_i h_i$$

TÉCNICO LISBOA

Affinity Scores

There are **several ways** of computing the scores, $\boldsymbol{s}$, with $\boldsymbol{h_1}, \cdots, \boldsymbol{h_L}$ and $\boldsymbol{q}$

- "Basic" dot-product attention: $s_i = q^T h_i$
- Bilinear attention: $s_i = q^T U h_i$
  - U is a weight matrix
- Additive attention: $s_i = u^T \tanh(A h_i + B q)$
  - A and B are weight matrices and u is a weight vector

Keys and Values

The **input matrix** $\mathrm{H} = [h_1, \cdots, h_L]^T \in \mathbb{R}^{L \times d}$ appears in two places:

1. Used as keys to “compare” them with the query vector q to obtain the **affinity scores**
2. Used as values in the weighted sum to get the **attention output**

However, they don’t have to be the same - we can have:

1. A **key matrix** $\mathrm{K} = [k_1, \cdots, k_L]^T \in \mathbb{R}^{L \times d_k}$
2. A **value matrix** $\mathrm{V} = [v_1, \cdots, v_L]^T \in \mathbb{R}^{L \times d_v}$

Attention: More General Version

Query vector: $q \in \mathbb{R}^d$
Key vectors: $\mathrm{K} = [k_1, \cdots, k_L]^T \in \mathbb{R}^{L \times d_k}$
Value vectors: $\mathrm{V} = [v_1, \cdots, v_L]^T \in \mathbb{R}^{L \times d_v}$

1.

1. Computing the **afinity scores**, $s \in \mathbb{R}^L$, by *comparing* q and K.

2. Apply the softmax to get **attention distribution**, $\alpha \in \mathbb{R}^L$

$$\alpha = softmax(s)$$

2.

3. Calculate the weighted sum of V with α as weights to get the **attention output**, $a \in \mathbb{R}^d$

$$a = V^T \alpha = \sum_{i=1}^{N} \alpha_i h_i$$

TÉCNICO LISBOA

Attention: More General Version

Consider a sequence of $\boxed{\textbf{length L}}$

Query vector: $\mathrm{Q} = [q_1, \cdots, q_L]^T \in \mathbb{R}^{L \times d_q}$

Key vectors: $\mathrm{K} = [k_1, \cdots, k_L]^T \in \mathbb{R}^{L \times d_k}$

Value vectors: $\mathrm{V} = [v_1, \cdots, v_L]^T \in \mathbb{R}^{L \times d_v}$

1. Computing the query-key **afinity scores,** $\mathrm{S} \in \mathbb{R}^{L \times L}$

$$\mathrm{S} = QK^T \in \mathbb{R}^{L \times L} \qquad \text{(dot-product affinity)}$$

2. Convert these scores to **probabilities** (row-wise), $P \in \mathbb{R}^{L \times L}$

$$P = softmax(s) \in \mathbb{R}^{L \times L}$$

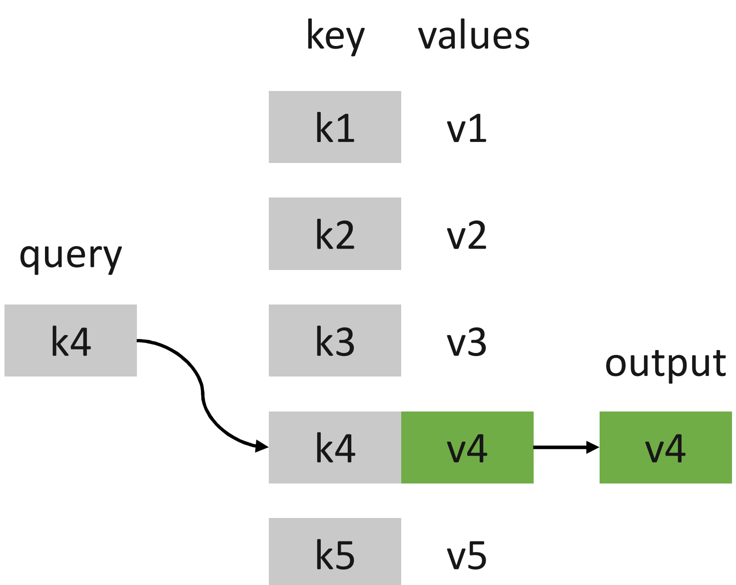
3. Calculate the weighted average to get the **attention output**, $\mathrm{A} \in \mathbb{R}^{L \times d_v}$

$$A = PV = softmax(QK^T)V$$

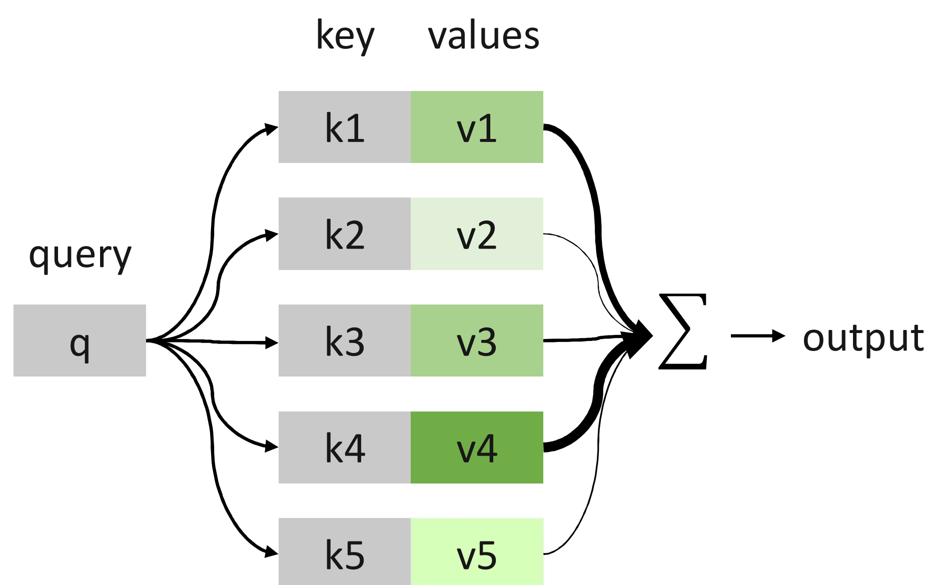
TÉCNICO LISBOA

Attention: a lookup table?

In a **lookup table**, the **query** matches a **key** and returns its **value**



In **attention**, the **query** matches all **keys** softly with a weight between 0 and 1 and returns the weighted sum of its **values**



Attention: Intuition

- Attention is a way to obtain a **representation** of an arbitrary set of **representations** (the values), dependent on some other **representation** (the query).

Fun Fact! There is a relation between the concept of **self-attention** and **convolutional layers** (paper)

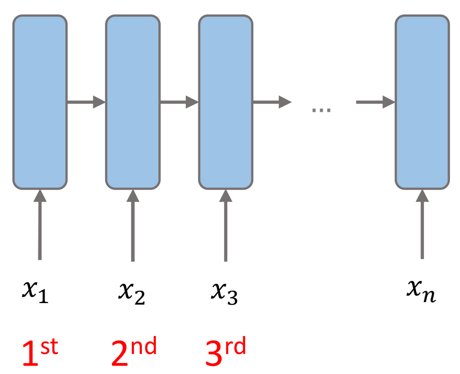
Attention is great!

- Attention significantly **improves model performance**!
- It’s very useful to allow **decoder to focus** on certain **parts of the source**
- Attention solves the **bottleneck problem** (by allowing the decoder to look directly at source)
- Attention helps with **vanishing gradient problem** (provides shortcut to faraway states)
- Attention provides **some interpretability** (by inspecting the attention distribution, we can see what the decoder was focusing on)

Issues with recurrent models

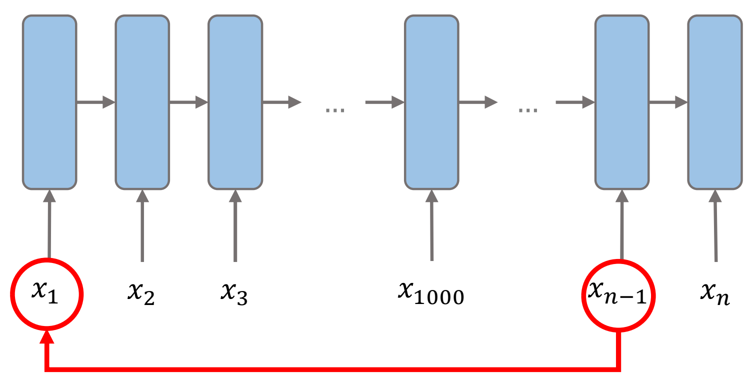
Lack of parallelization

- Forward and backward passes have **O(sequence length)** unparallelizable operations
- Inhibits training on **very large** datasets!



Long-range dependencies are tricky! (even with LSTMS)

- RNNs are unrolled "left-to-right"
- **Hard to learn long-distance** dependencies (because of gradient problems!)



If not recurrence then what?

What about **attention**?

Lecture Plan

- Introduction to the problem
- From recurrence (RNN) to attention-based NLP models
- Introducing the Transformer model
- Great results with Transformers
- Drawbacks of the Transformers

Transformer

- Encoder-Decoder architecture (similarly to before)
- Released in 2017
- Key Idea: **Attention is all you need!**

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

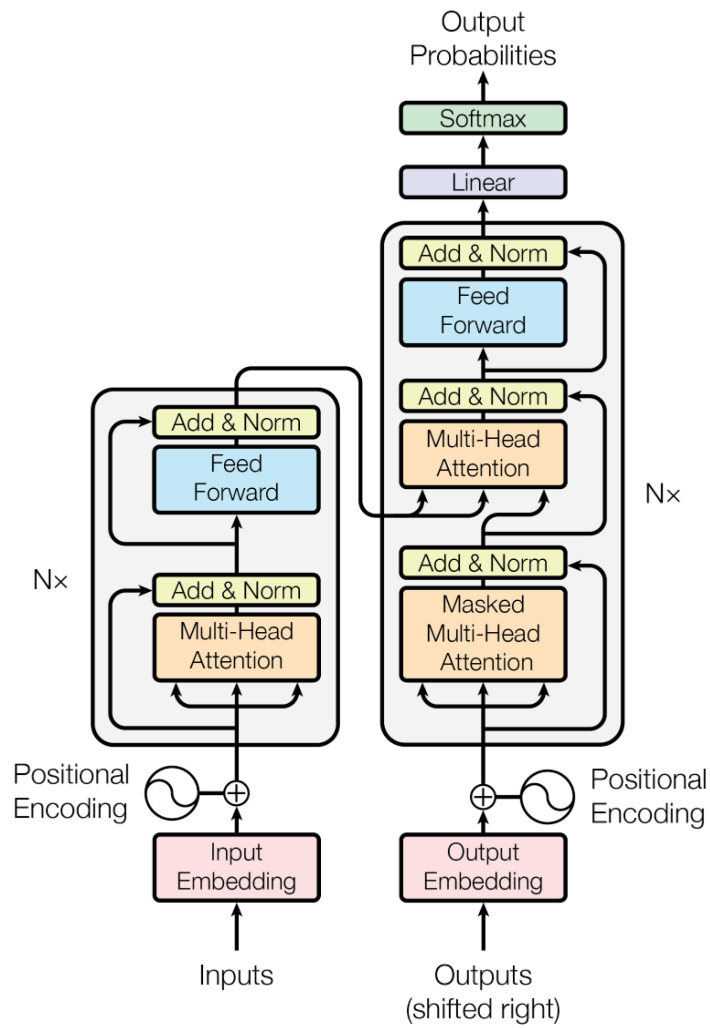
Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract



TÉCNICO LISBOA

Self-Attention: Intuition

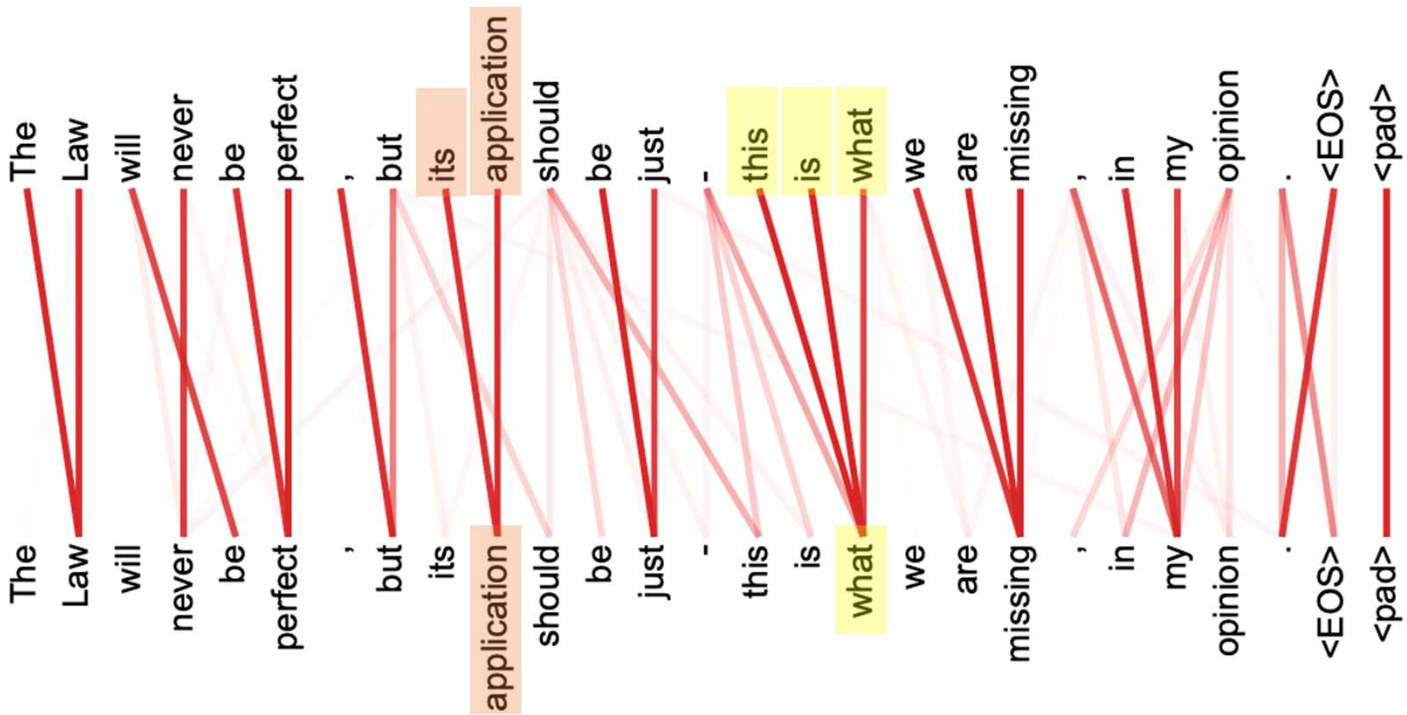


Image by: [The original Transformer paper]

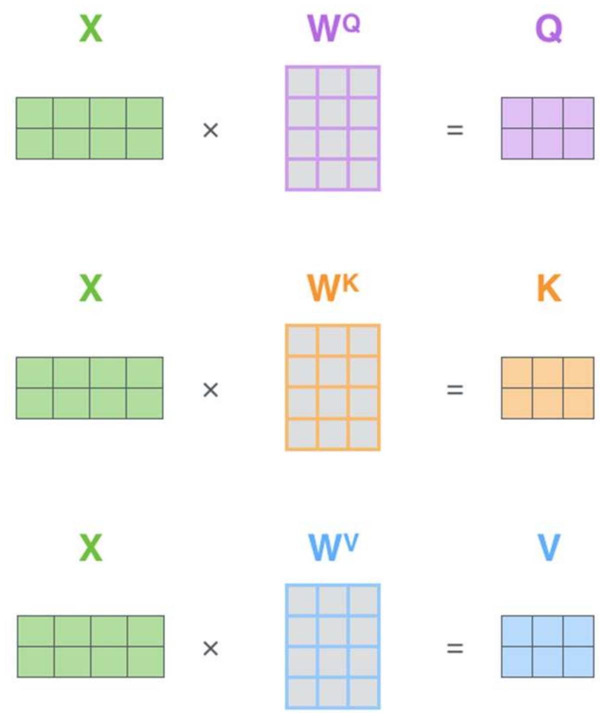
Fun Fact! There is a relation between the concept of **self-attention** and **convolutional layers** ([paper])

TÉCNICO LISBOA

Transformer: Self-Attention

The **original transformer model** does this in a particular way:

> **Query** vector: $\mathrm{Q} = XW^Q$
> **Key** vectors: $\mathrm{K} = XW^k$
> **Value** vectors : $\mathrm{V} = XW^v$



The matrices W^Q , W^k, W^v correspond to model parameters

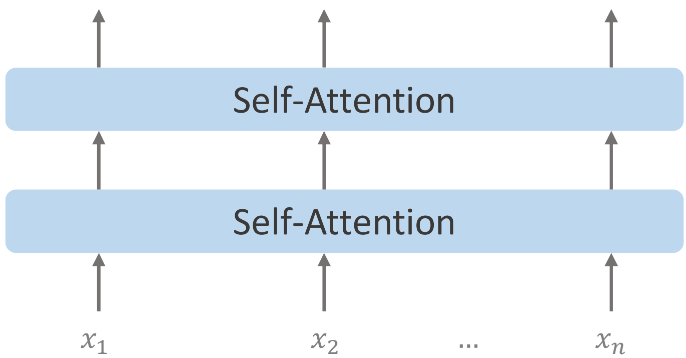
Image by: [Illustrated Transformer]

TÉCNICO LISBOA

Let’s build a transformer!

Self-Attention as a Building Block

- In the diagram at the right, we have **stacked self-attention blocks**, like we might stack LSTM layers.
- Can self-attention be a drop-in replacement for recurrence? **No**. It has a few issues, which we'll go through.



Barriers and solutions for Self-Attention as a building block

Barrier		**Solution**
Doesn't have an inherent **notion of order**!	→	?

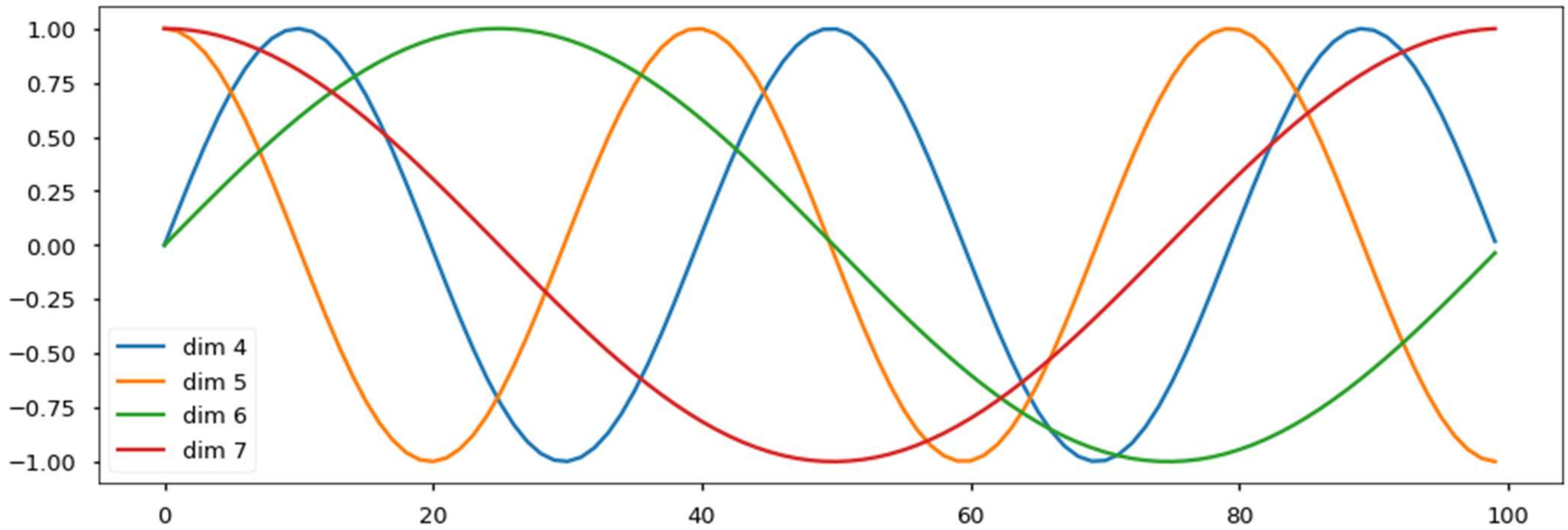
Solution: Positional Encoder

- Since self-attention doesn't build in order information, we need to **encode the order of the sentence**.
- Consider representing each sequence index as a vector
$$p_i \in \mathbb{R}^d, i \in \{1, 2, \dots, T\}$$
- How to define $p_i \in \mathbb{R}^d$?
  1. Use a **sinusoidal positional encoder** (strategy used by the original Transformer paper)
  2. Learn **one embedding for each position**

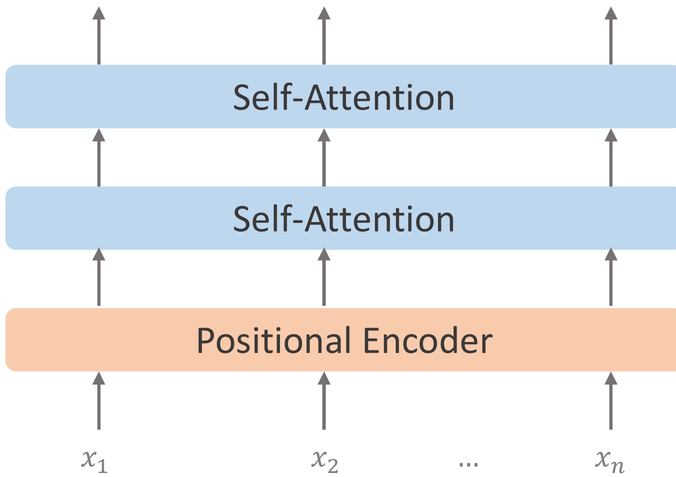
TÉCNICO LISBOA

Sinusoidal Positional Encoder

$$PE_{(pos,2i+1)} = cos\left(\frac{pos}{1000^{2i/d_{model}}}\right) \qquad PE_{(pos,2i)} = sin\left(\frac{pos}{1000^{2i/d_{model}}}\right)$$



Adding Positional Encoder

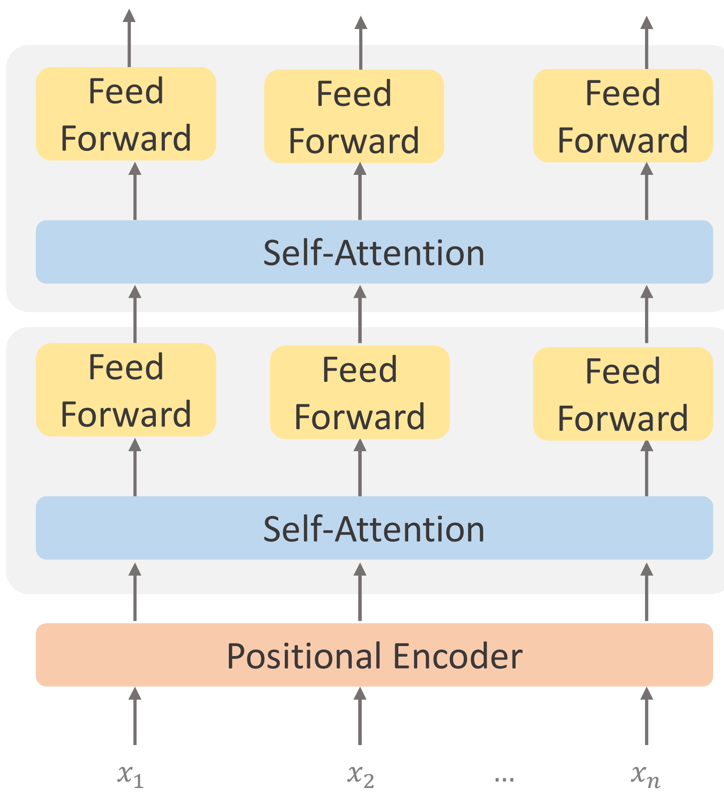


Barriers and solutions for Self-Attention as a building block

Barrier		Solution
Doesn't have an inherent **notion of order**!	→	**Positional Encoding**!
No nonlinearities for learning expression! (It's all just weighted averages)	→	?

Solution: Adding nonlinearities

- Note that there are no **elementwise nonlinearities** in self-attention: stacking more self-attention layers just re-averages **value** vectors
- **Solution**: add a feed-forward network to post-process each output vector



Barriers and solutions for Self-Attention as a building block

Barrier		Solution
Doesn't have an inherent **notion of order**!	→	**Positional Encoding**!
No nonlinearities for learning expression! It's all just weighted averages	→	Apply **feedforward network** to each self-attention output.
Need to ensure we **don't "look at the future"** when predicting a sequence	→	?

Solution: Masking

- To use self-attention in **decoders**, we need to ensure **we can't peek at the future**.
- At every timestep, we could change the set of **keys and queries** to include only past words. (Inefficient!)
- To enable parallelization, we **mask out attention** to future words by setting attention scores to **-inf**.

$$s_{ij} = \begin{cases} q_i^T k_j, & i \leq j \\ -\infty, & i > j \end{cases}$$

	[START]	Grain	by	Grain
[START]		-inf	-inf	-inf
Grain			-inf	-inf
by				-inf
Grain				

TÉCNICO LISBOA

Barriers and solutions for Self-Attention as a building block

Barrier		Solution
Doesn't have an inherent **notion of order**!	→	**Positional Encoding**!
No nonlinearities for learning expression! It's all just weighted averages	→	Apply **feedforward network** to each self-attention output.
Need to ensure we **don't "look at the future"** when predicting a sequence	→	**Mask out** the future by artificially setting attention weights to 0!

Putting the blocks together…

Transformer: Encoder-Decoder

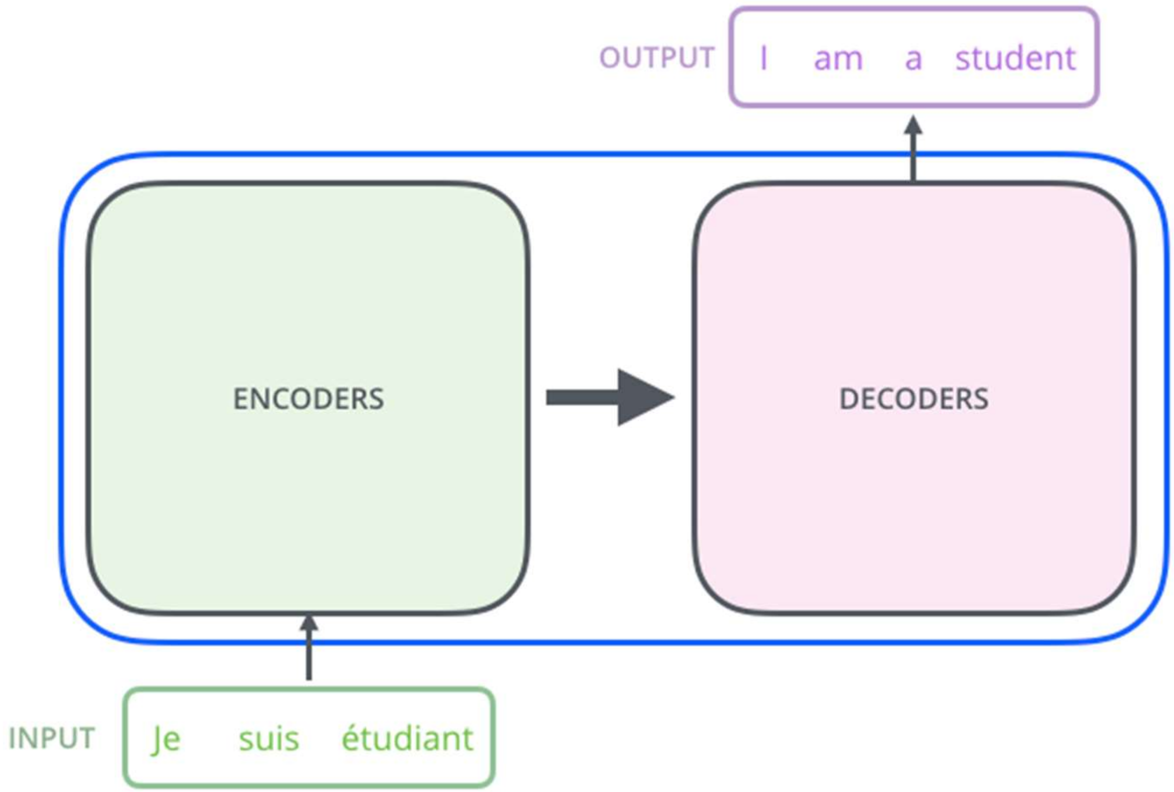


Image by: [Illustrated Transformer]()

Transformer: Basics

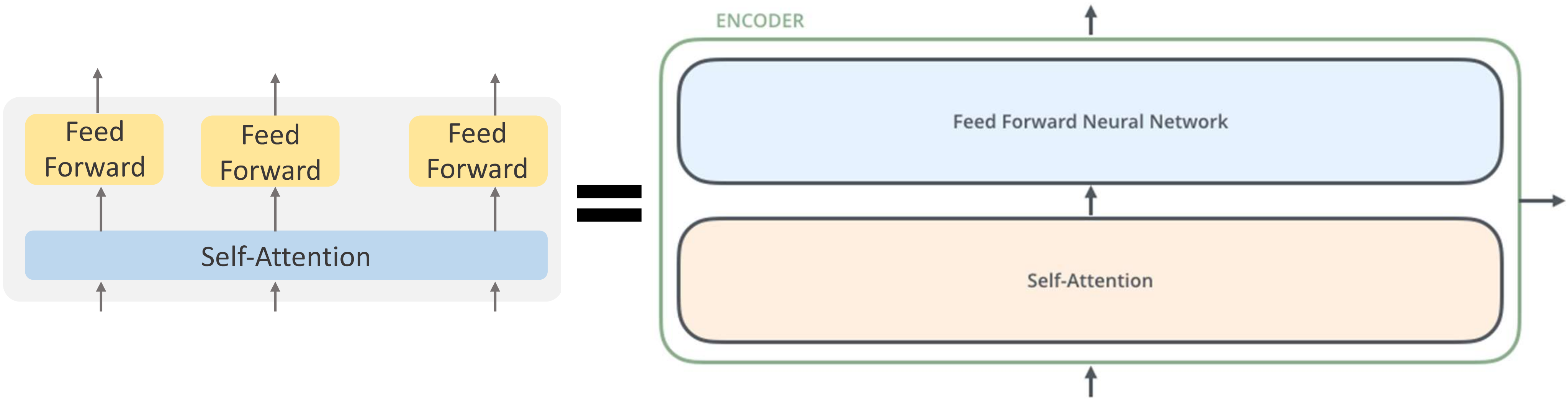


Image by: [Illustrated Transformer]

TÉCNICO LISBOA

Transformer: Encoder-Decoder

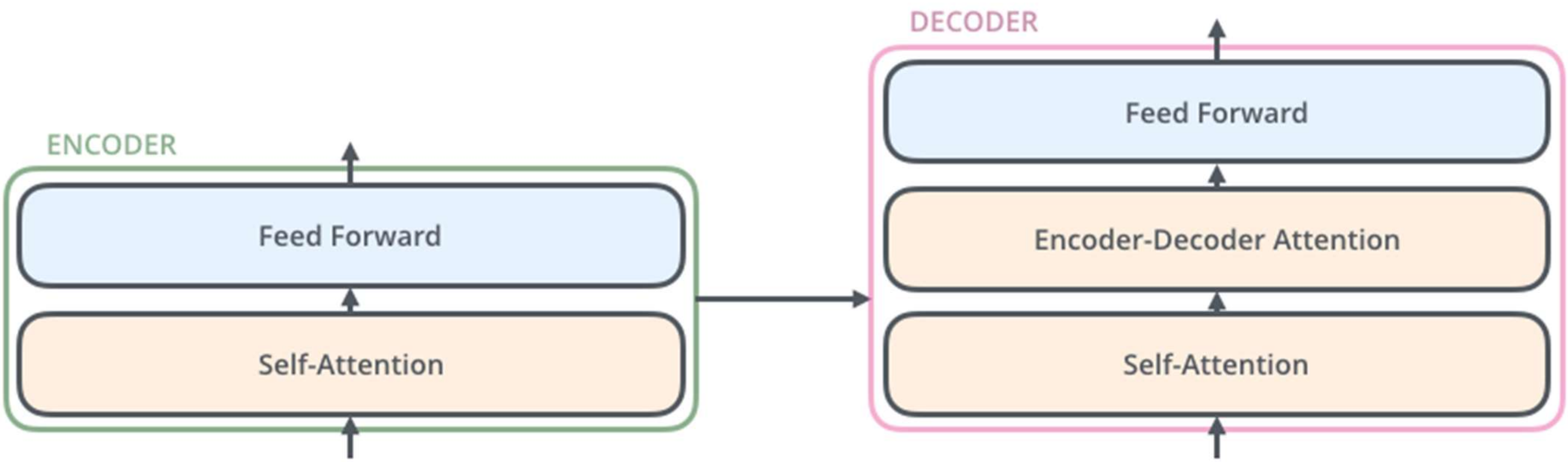


Image by: [Illustrated Transformer]

Transformer: Basics

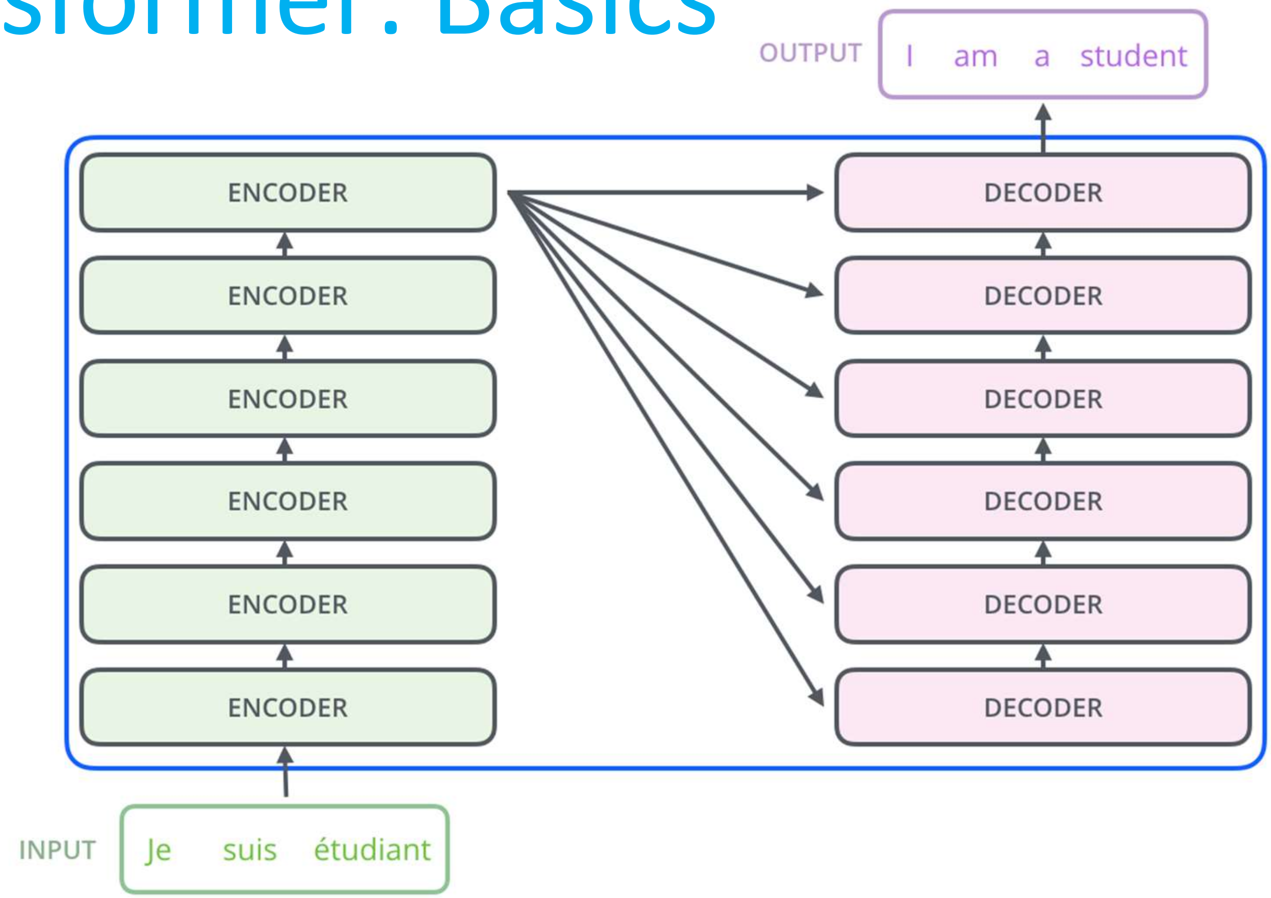


Image by: [Illustrated Transformer]()

TÉCNICO LISBOA

What’s left in a Transformer that we haven’t covered?

What’s left in a Transformer that we haven’t covered?

Multi-head Attention

Tricks to help with Training!

Multi-head Attention

- Idea: What if we want to **look in multiple places** in the sentence at once?
- Define $\boldsymbol{h}$ **attention heads** and apply attention to each of them:

$$Multihead(X) = Concat(A_1, \ldots, A_h)W^o,$$

where $A_i = Attention\left(XW_i^Q, XW_i^K, XW_i^V\right)$

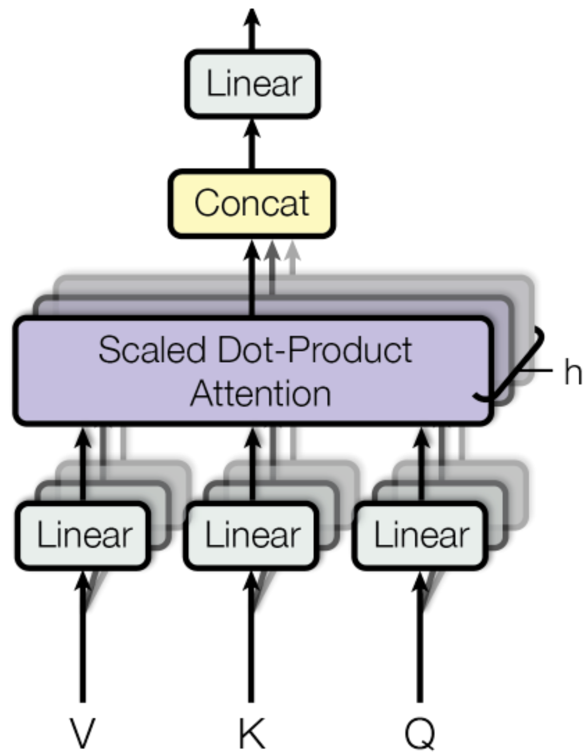


Image by: [The original Transformer paper]

Multi-head Attention

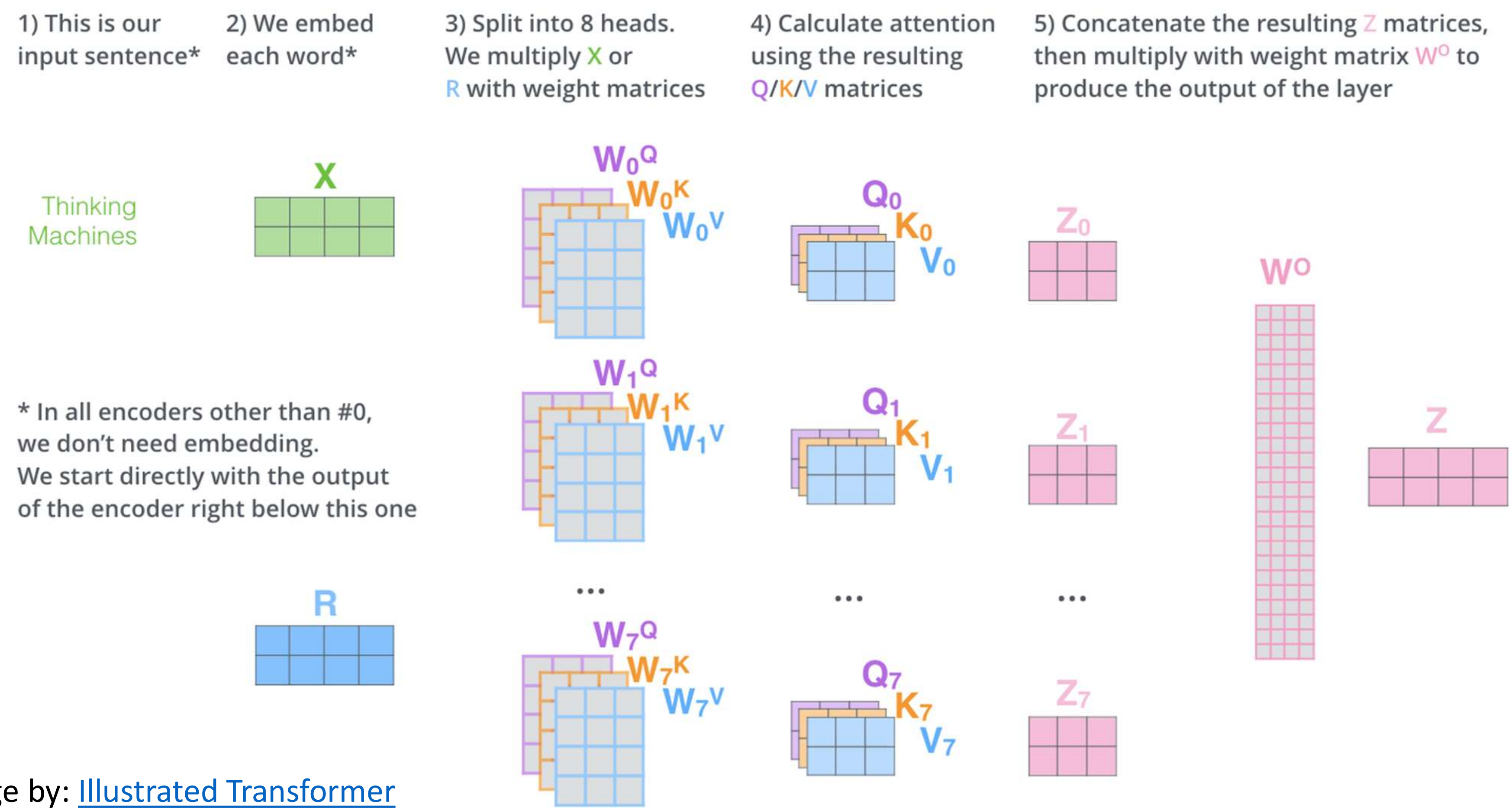


Image by: [Illustrated Transformer]

Tricks to help with training!

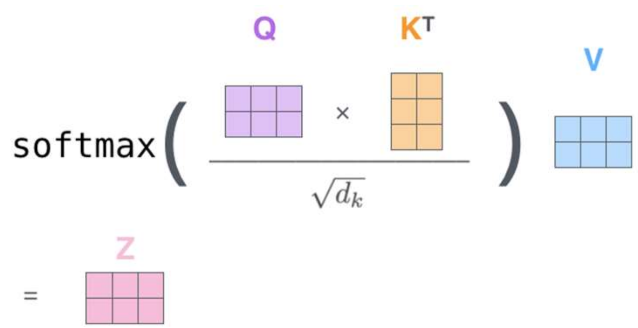
Residual Connections

- Residual connections are thought to make the loss landscape **considerably smoother** (thus easier training!)

Layer Normalization

- Idea: cut down on uninformative variation in hidden vector values by **normalizing to unit mean and standard deviation** within each layer.

Scaled Dot-Product Attention



Residual Connections & Layer Normalization

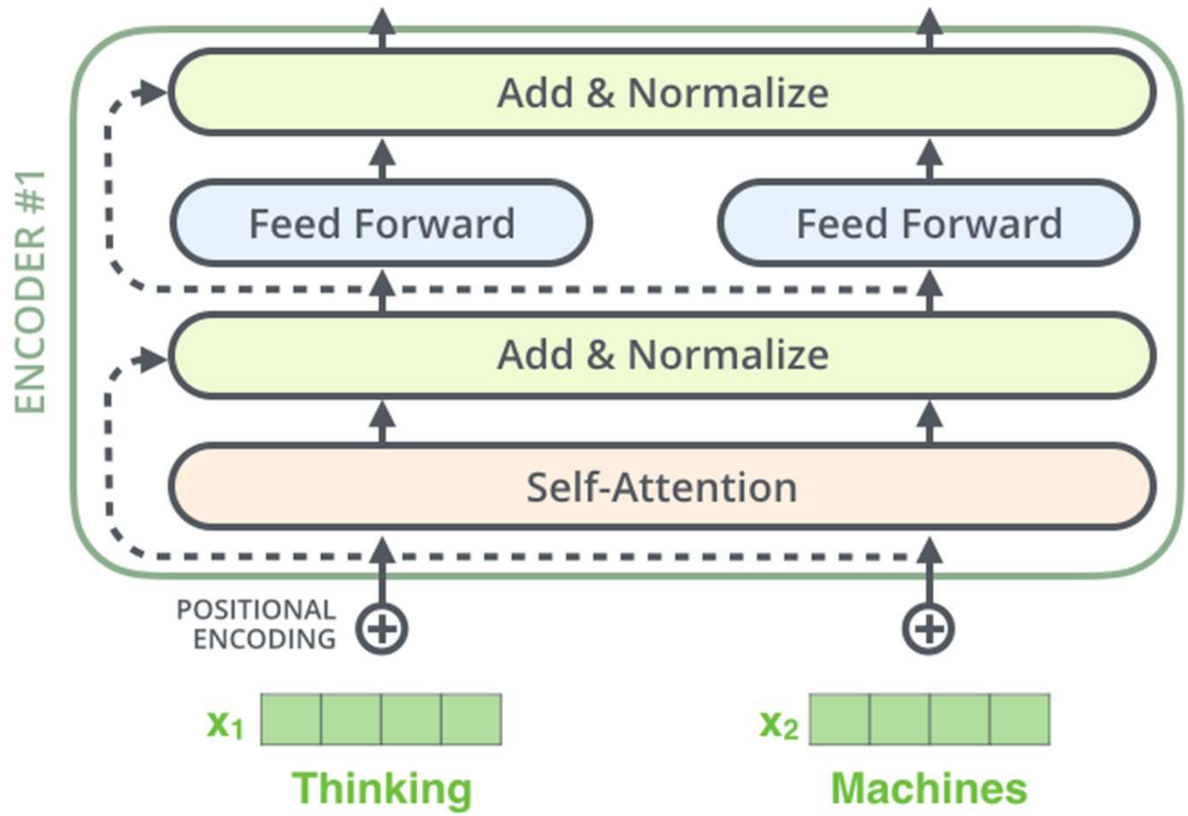


Image by: [Illustrated Transformer]()

TÉCNICO LISBOA

Residual Connections & Layer Normalization

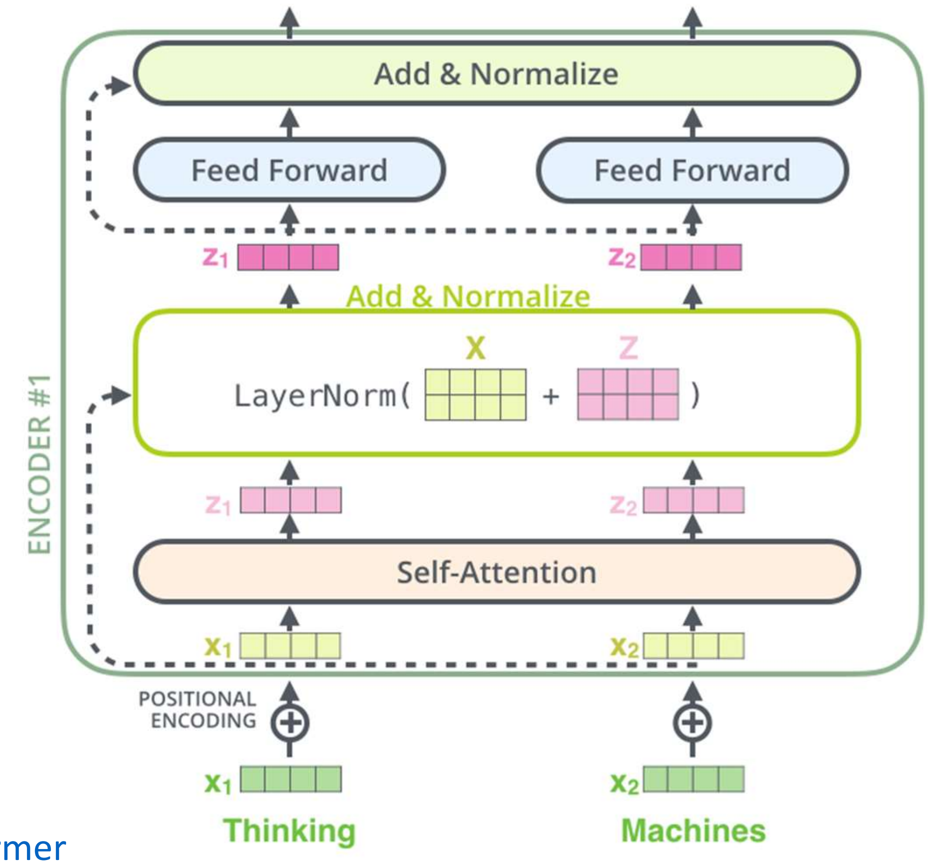


Image by: [Illustrated Transformer]()

TÉCNICO LISBOA

Residual Connections & Layer Normalization

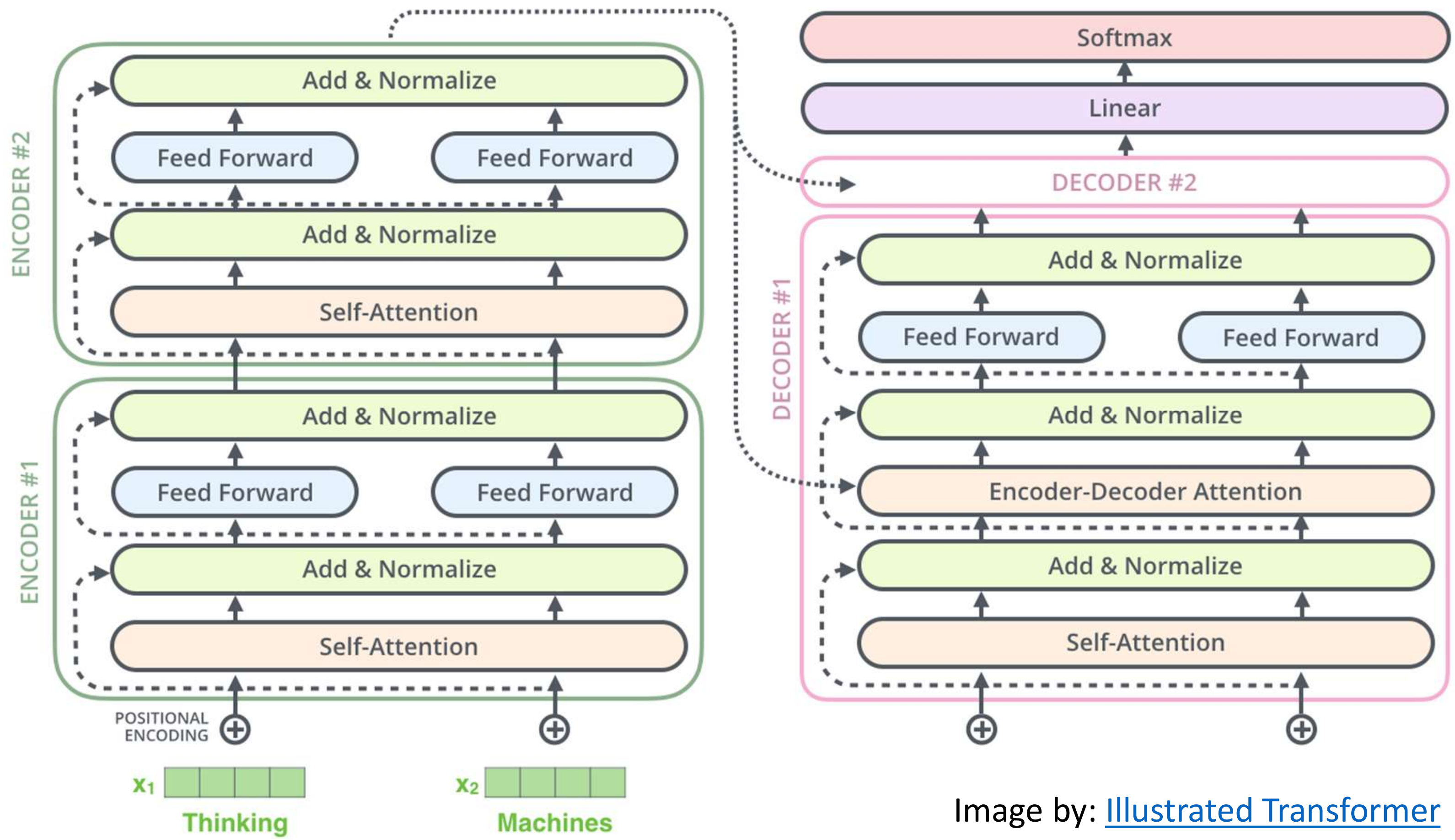
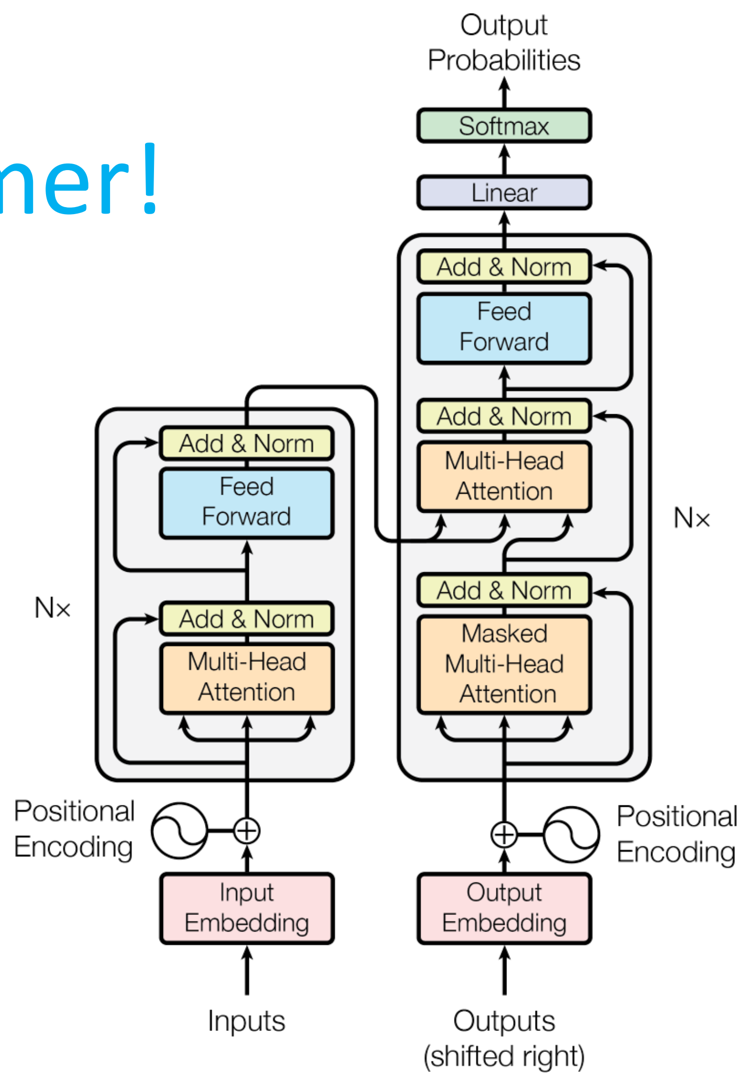


Image by: Illustrated Transformer

TÉCNICO LISBOA

Congrats you build a Transformer!



Implementation in Keras

```
vocab_size = 20000
sequence_length = 600
embed_dim = 256
num_heads = 2
dense_dim = 32

inputs = keras.Input(shape=(None,), dtype="int64")
x = PositionalEmbedding(sequence_length, vocab_size, embed_dim)(inputs)
x = TransformerEncoder(embed_dim, dense_dim, num_heads)(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.summary()
```

Check the implementation on the course page!

Lecture Plan

- Introduction to the problem
- From recurrence (RNN) to attention-based NLP models
- Introducing the Transformer model
- Great results with Transformers
- Drawbacks of the Transformers

Great Results with the Transformers

Results from the original transformer paper

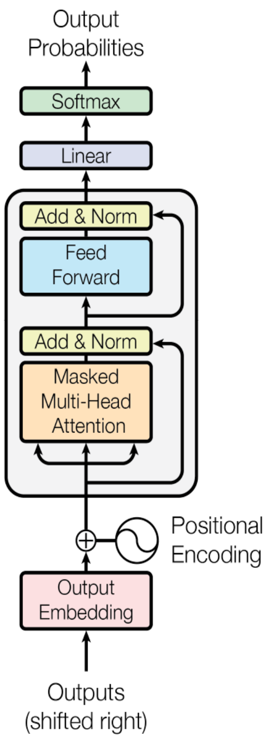
Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

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

Note: BLEU (BiLingual Evaluation Understudy) is a **metric for evaluating machine-translated text**. The score **measures the similarity** of the machine-translated text to a set of high-quality reference translations

GPT (Decoder-only)

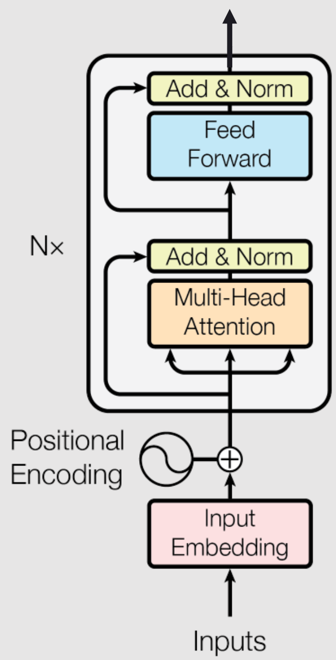
sat



[START] The cat

BERT (Encoder-only)

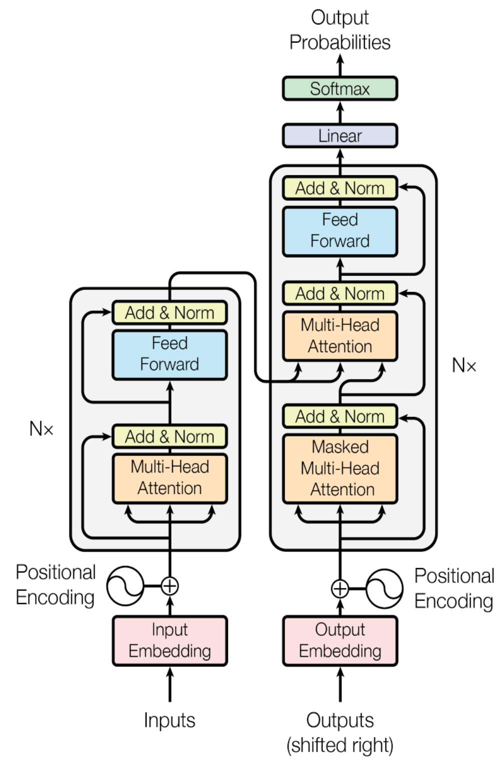
[*] [*] [*] sat [*] the [*]



[START] The cat [MASK] on [MASK] mat

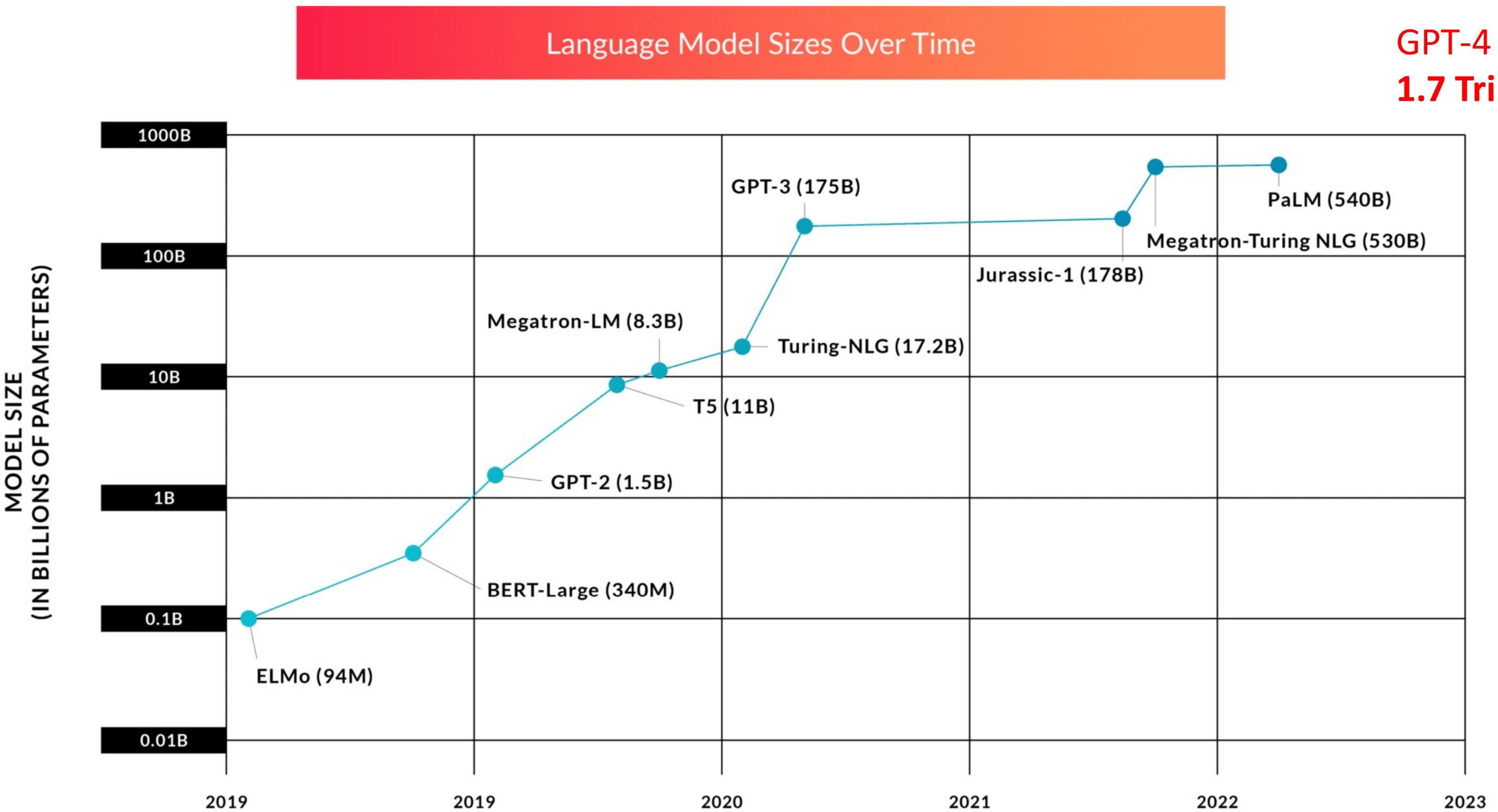
T5 (Encoder-Decoder)

[*] [*] [*] sat on [*] [*]



[START] The cat [X] the mat

Great Results with the Transformers



GPT-4 (march 2023) has
1.7 Trillion parameters !!!

(logarithmic scale)

Source: [IEEE blog]

Lecture Plan

- Introduction to the problem
- From recurrence (RNN) to attention-based NLP models
- Introducing the Transformer model
- Great results with Transformers
- Drawbacks of the Transformers

What would we like to fix about the Transformer?

Quadratic compute in self-attention

- Computing **all pairs** of interactions means that the computation **grows quadratically** with the sequence length!
- (For recurrent models, it only grew linearly)

Position representations

- Are **simple absolute indices** the best we can do to represent position?

Great videos to watch!

- Stanford Seminar: Introduction to Transformers
- LSTM is dead. Long Live Transformers!
- A brief history of the Transformer architecture in NLP
- Positional embeddings in transformers EXPLAINED | Demystifying positional encodings
- Transformer models and BERT model: Overview

TÉCNICO LISBOA