# Introduction to Deep Learning

## Lecture: Transformers and the Attention Mechanism

**Francis Engelmann,** Computer Vision and Geometry Group, ETH Zurich

ETH AI Center Postdoctoral Fellow | June 7th, 2024

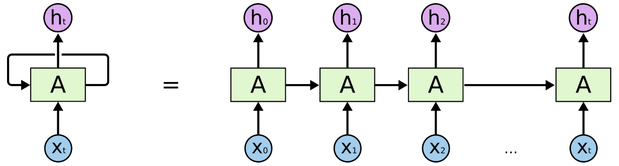
ETH zürich

ETH AI CENTER

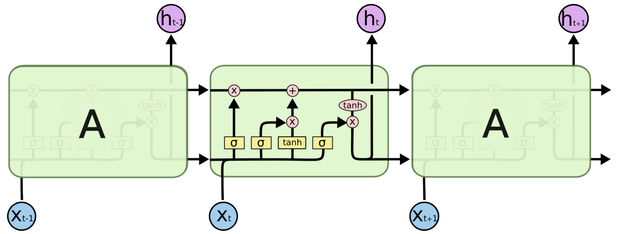
D INFK

# **Last Week:** Sequence Modeling (Recap)

- Recurrent Neural Networks (RNN)




- Long Short Term Memory (LSTM)




- Gated Recurrent Unit (GRU)

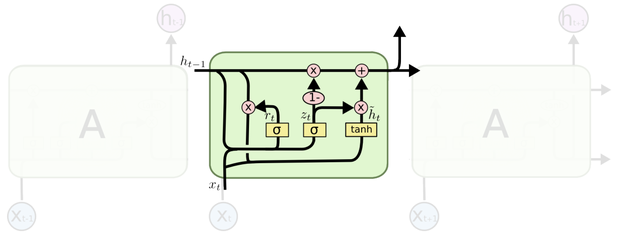
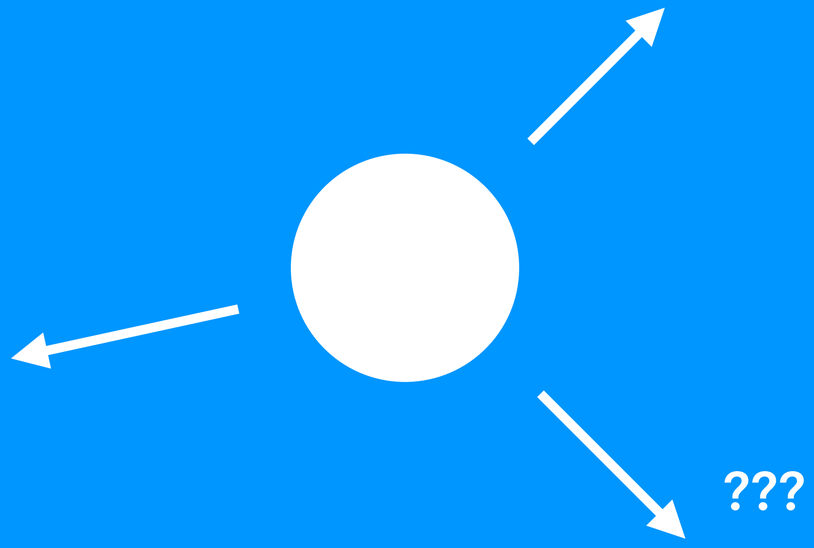


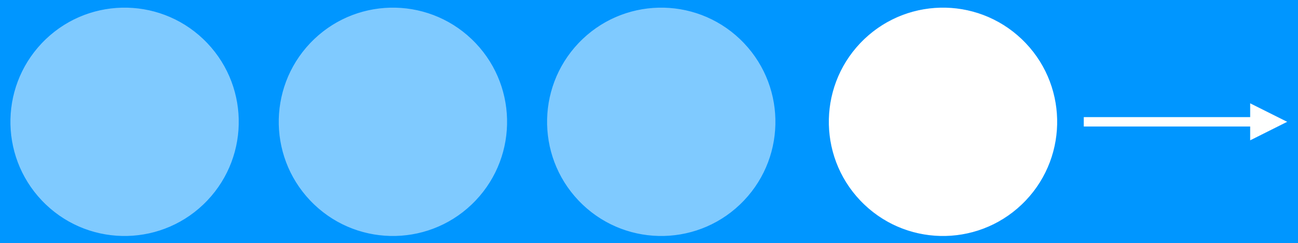

Image credit: Chris Olah

Given an image of a ball,
can you predict where it will go next?

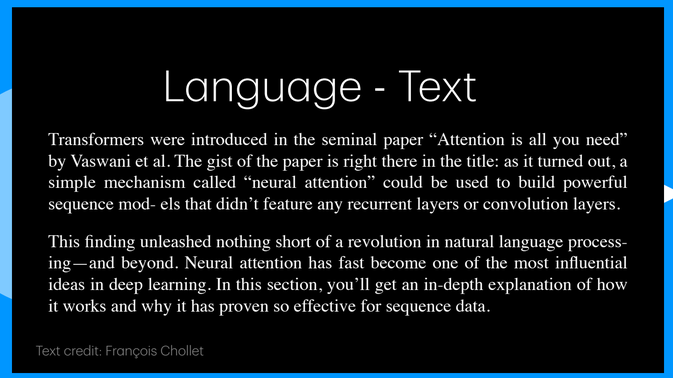
# Given an image of a ball, can you predict where it will go next?



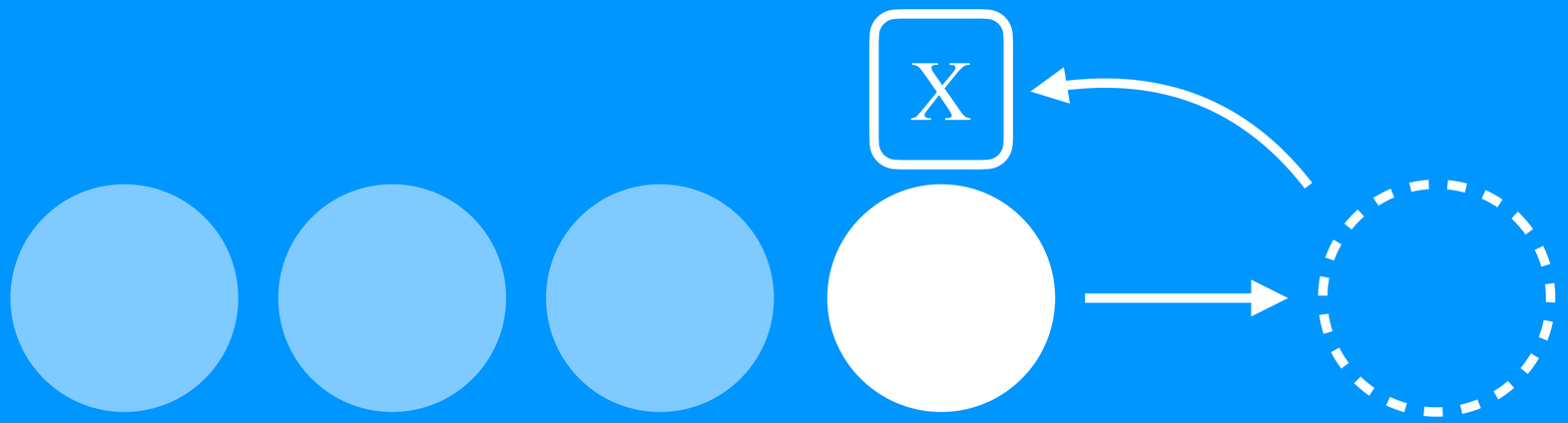
Given an image of a ball,
can you predict where it will go next?



# Given an image of a ball, can you predict where it will go next?

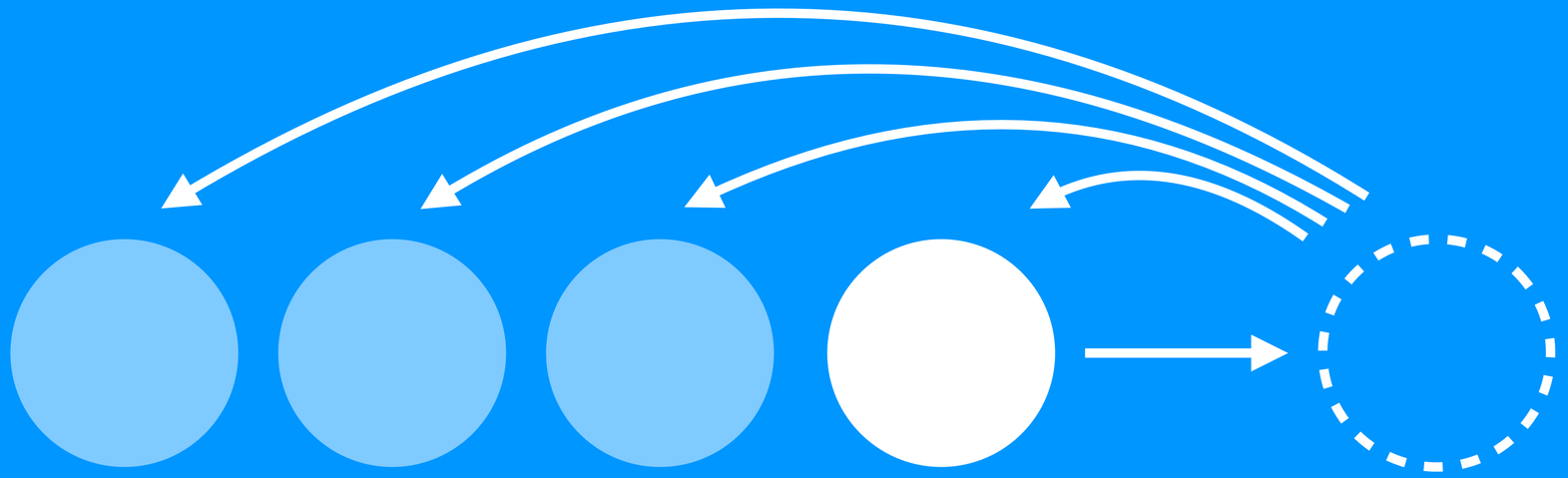




# RNN



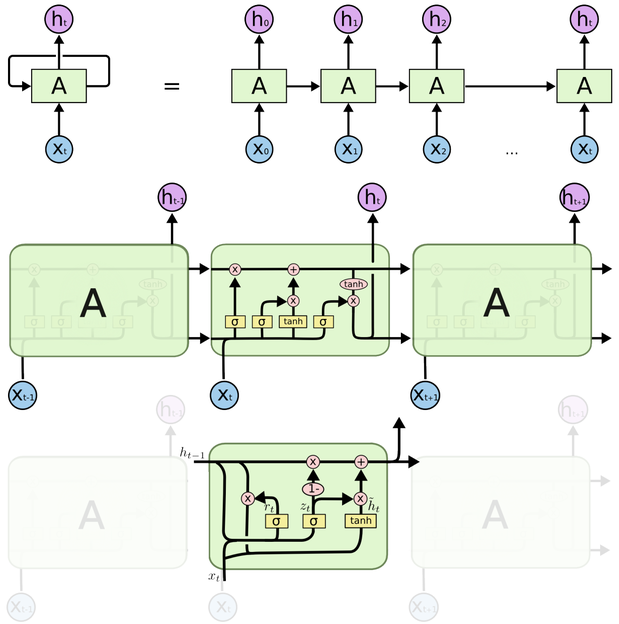

*"**Luxembourg** is where I grew up, but now I live in Zurich. My mother tongue is ____."*

# Attention



# **Last Week:** Sequence Modeling (Recap)

- Recurrent Neural Networks (RNN)
- Long Short Term Memory (LSTM)
- Gated Recurrent Unit (GRU)
- **Today**: Attention and the Transformer model



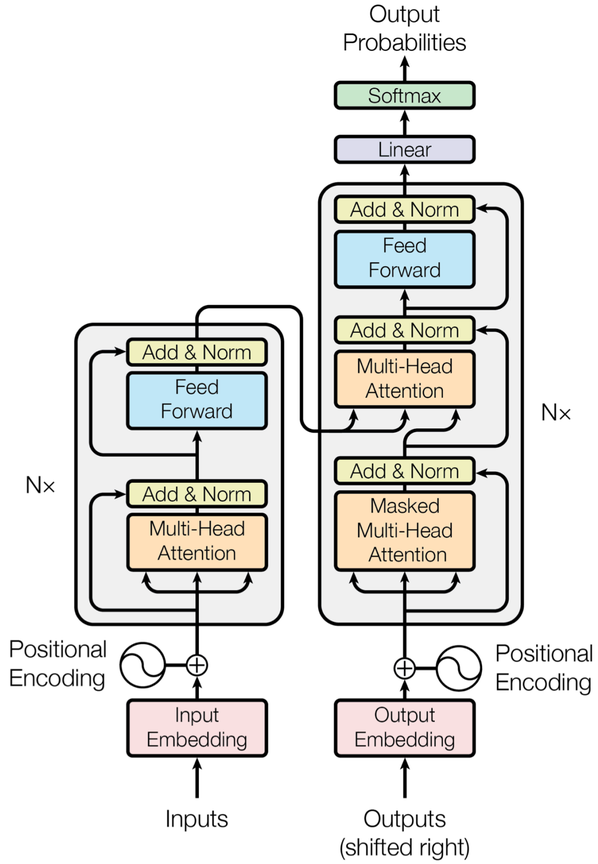
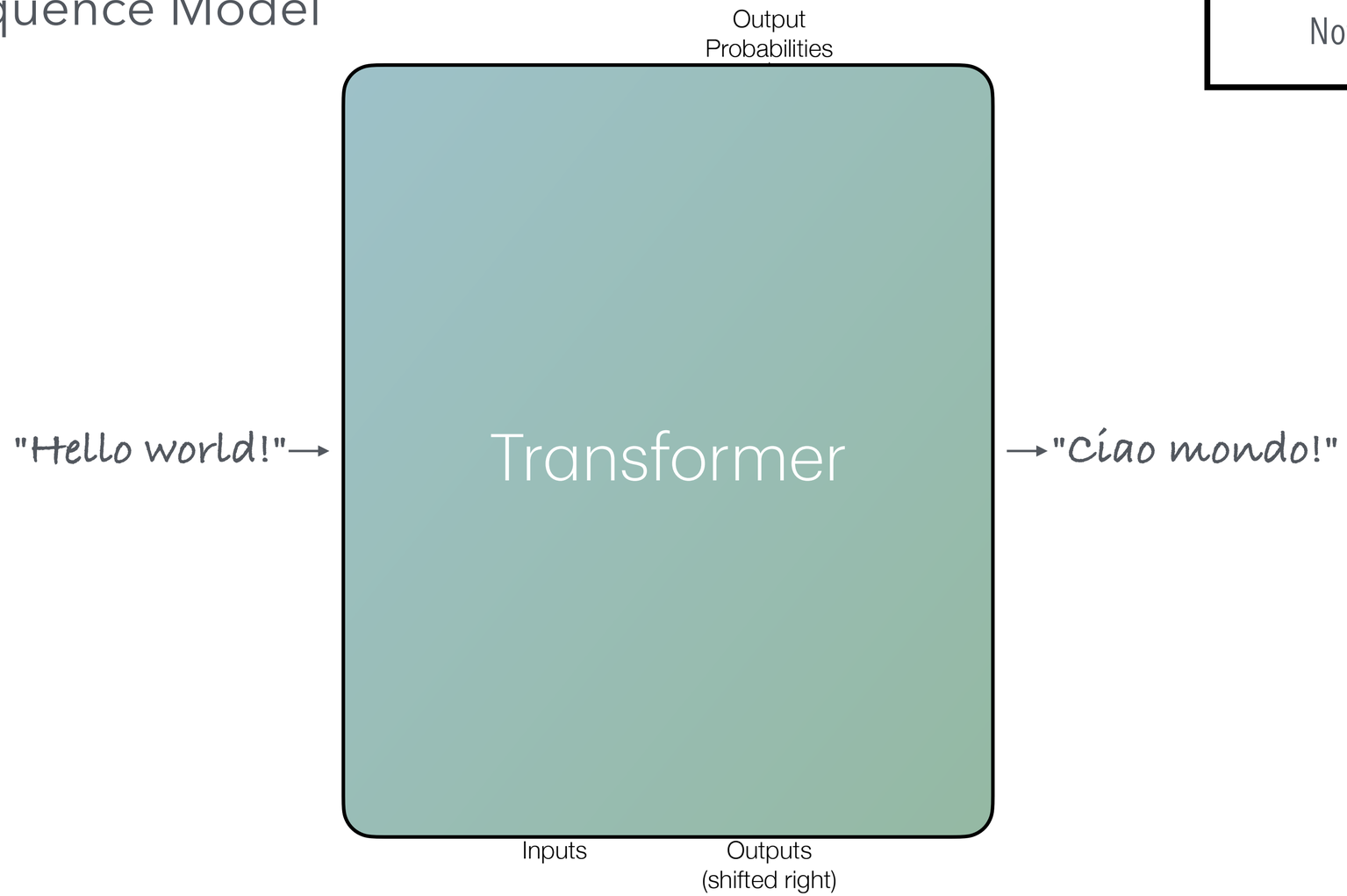




Image credit: Chris Olah, Vaswani et al.

# Transformer: Overview

## Sequence-to-Sequence Model

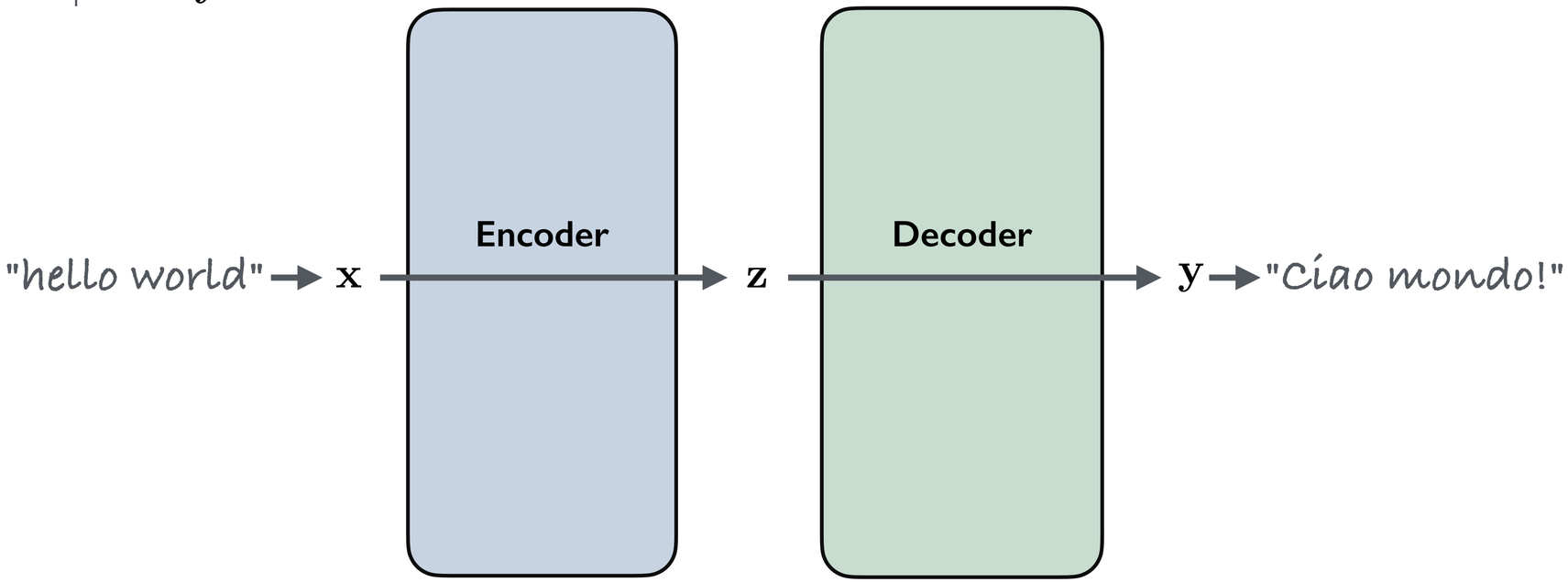
PyTorch Code Notebook



# Transformer: Overview

## Encoder-Decoder Architecture

Mapping from input sequence of tokens $\mathbf{x}$ to sequence of learned encodings $\mathbf{z}$, decoded into output sequence $\mathbf{y}$.



# Transformer: Overview

## Transformer Encoder

Mapping from input sequence of tokens $\mathbf{x}$ to sequence of learned encodings $\mathbf{z}$.

**Main Goal**: "high level, contextualized" representation of input tokens.

Meaning: each $z_i$ can "look at" all $x_i$.

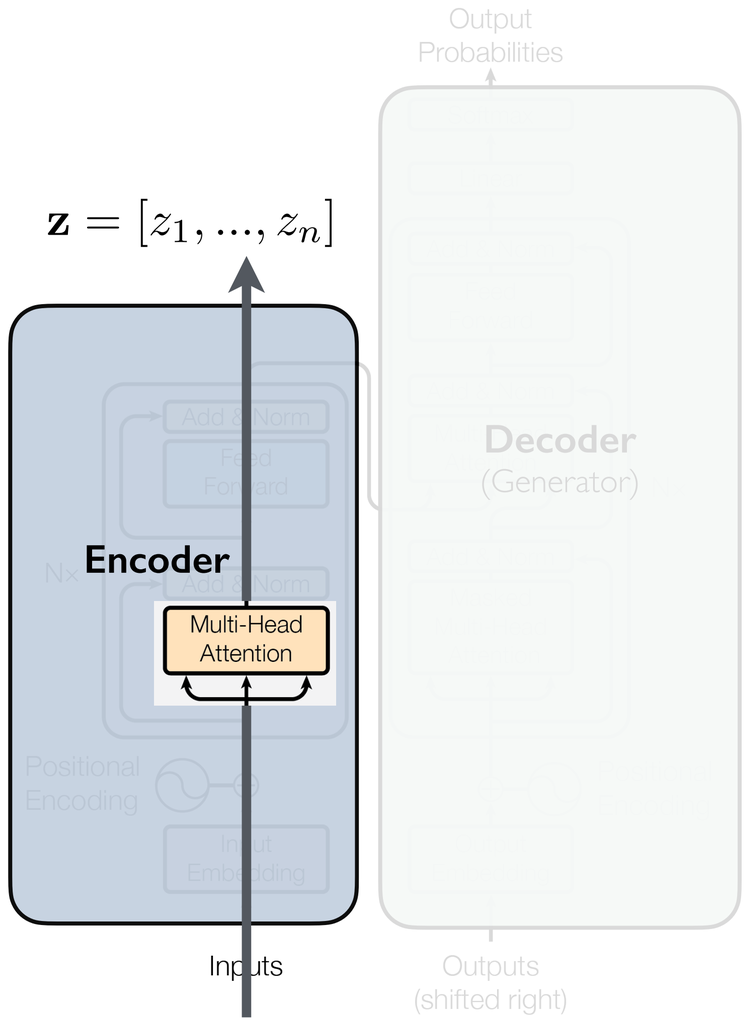



Image credit: Vaswani et al. "Attention is all you need"

# Transformer: Overview

## Transformer Decoder

output → $p(\mathbf{y}|\mathbf{z})$ ← encoded input

**Goal**: $p(\mathbf{y}|\mathbf{z})$

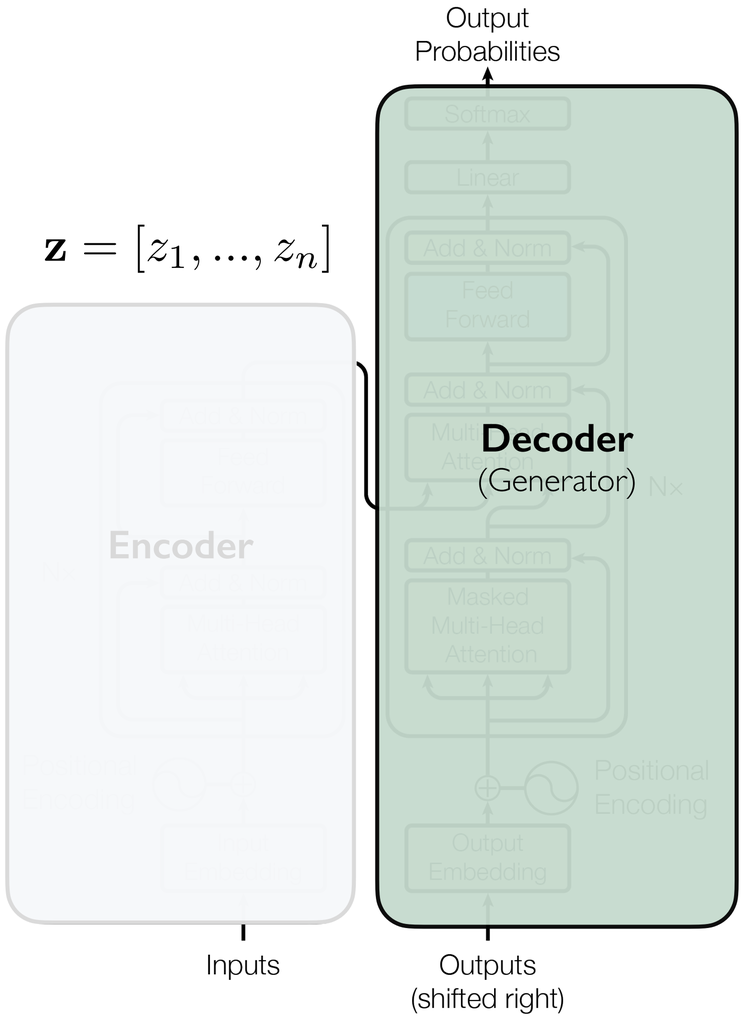
E.g., predict probability distribution over all possible sentences $\mathbf{y}$ in the target language, conditioned on the input sentence $\mathbf{z}$ in the source language.

**Approach**: decompose using chain rule

$$p(\mathbf{y}|\mathbf{z}) = p(y_1|\mathbf{z}) \cdot p(y_2|y_1, \mathbf{z}) \cdot p(y_3|y_2, y_1, \mathbf{z}) \cdot ...$$

$\mathbf{y} = [y_1, ..., y_m]$ → "Ciao mondo!"

$\mathbf{z} = [z_1, ..., z_n]$




"Hello world!" → $\mathbf{x} = [x_1, ..., x_n]$

Image credit: Vaswani et al.
Slide inspired by Lucas Beyer

# Transformer: Overview

## Transformer Decoder

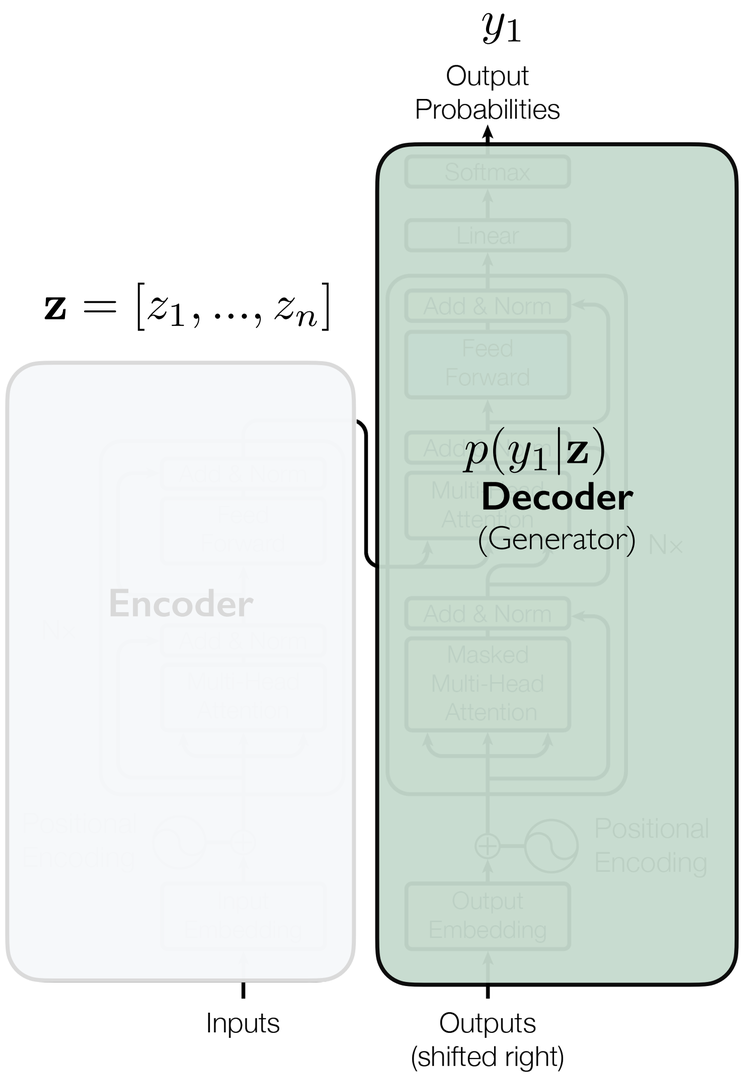
output → $\mathbf{y}$ ← encoded input $\mathbf{z}$

**Goal**: $p(\mathbf{y}|\mathbf{z})$

E.g., predict probability distribution over all possible sentences $\mathbf{y}$ in the target language, conditioned on the input sentence $\mathbf{z}$ in the source language.

**Approach**: decompose using chain rule

$$p(\mathbf{y}|\mathbf{z}) = p(y_1|\mathbf{z}) \cdot p(y_2|y_1, \mathbf{z}) \cdot p(y_3|y_2, y_1, \mathbf{z}) \cdot \ldots$$




$\mathbf{y} = [y_1, ..., y_m]$ → "Ciao mondo!"

"Hello world!" → $\mathbf{x} = [x_1, ..., x_n]$

Image credit: Vaswani et al.
Slide inspired by Lucas Beyer

# Transformer: Overview

## Transformer Decoder

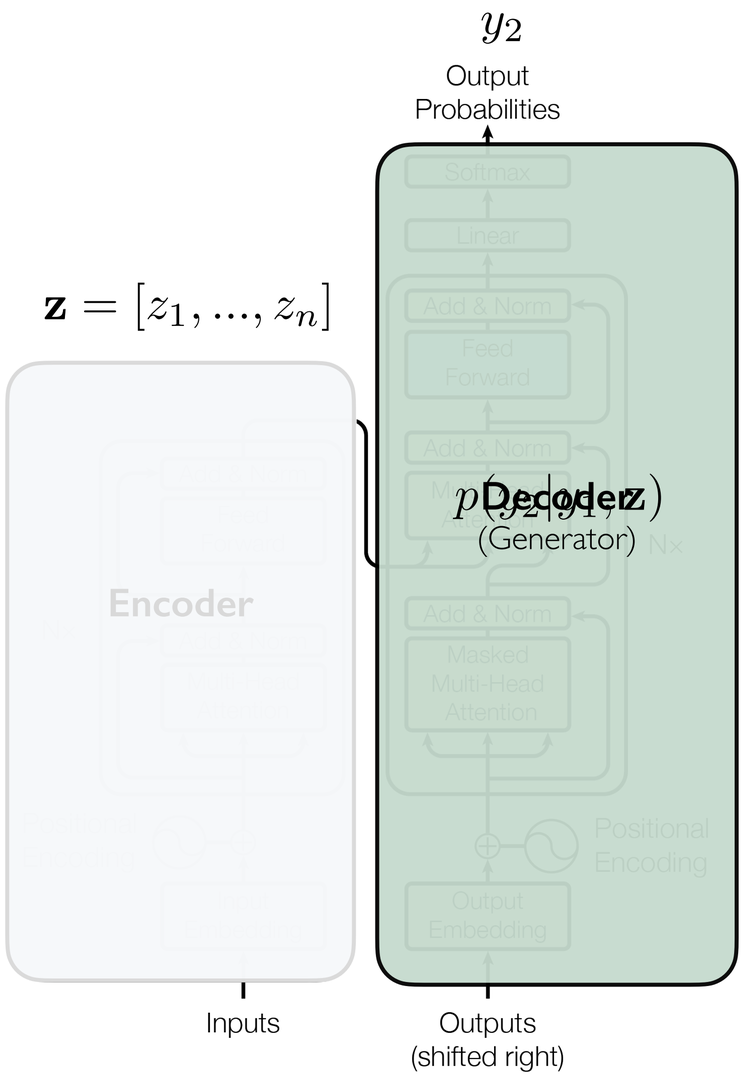
output → $\mathbf{y}$ ← encoded input $\mathbf{z}$

**Goal**: $p(\mathbf{y}|\mathbf{z})$

E.g., predict probability distribution over all possible sentences $\mathbf{y}$ in the target language, conditioned on the input sentence $\mathbf{z}$ in the source language.

**Approach**: decompose using chain rule

$$p(\mathbf{y}|\mathbf{z}) = p(y_1|\mathbf{z}) \cdot p(y_2|y_1, \mathbf{z}) \cdot p(y_3|y_2, y_1, \mathbf{z}) \cdot \ldots$$




$\mathbf{y} = [y_1, ..., y_m]$ → "Ciao mondo!"

"Hello world!" → $\mathbf{x} = [x_1, ..., x_n]$

Image credit: Vaswani et al.
Slide inspired by Lucas Beyer

# Transformer: Overview

## Transformer Decoder

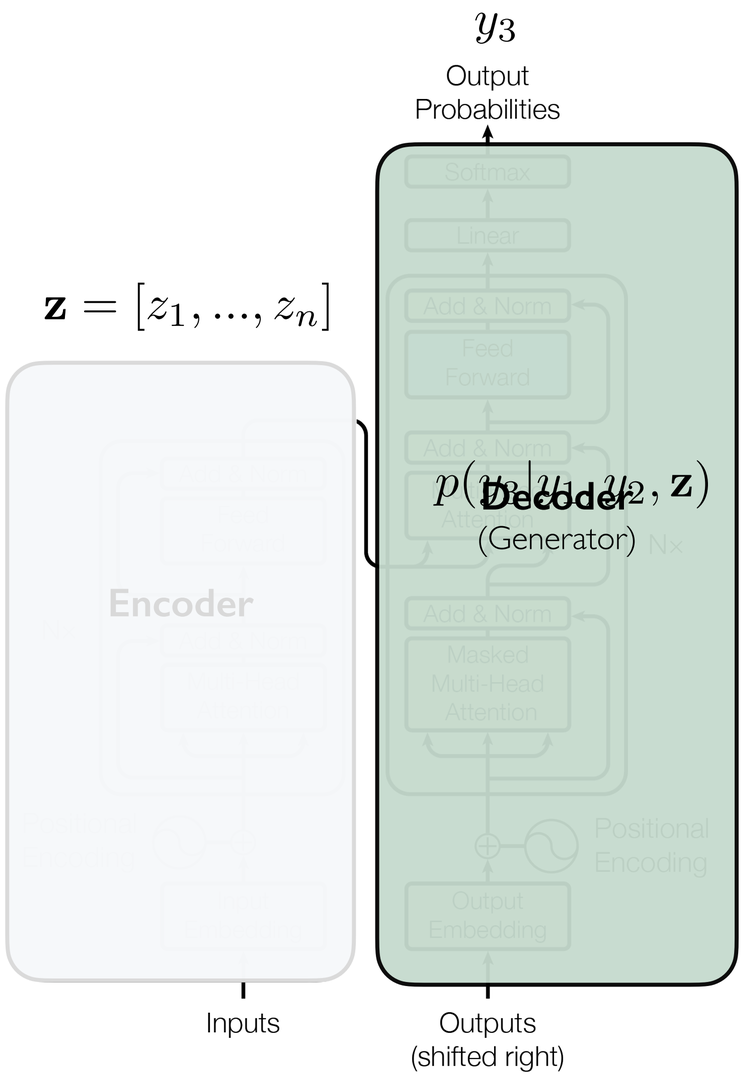
output → $p(\mathbf{y}|\mathbf{z})$ ← encoded input

**Goal**: $p(\mathbf{y}|\mathbf{z})$

E.g., predict probability distribution over all possible sentences $\mathbf{y}$ in the target language, conditioned on the input sentence $\mathbf{z}$ in the source language.

**Approach**: decompose using chain rule

$$p(\mathbf{y}|\mathbf{z}) = p(y_1|\mathbf{z}) \cdot p(y_2|y_1, \mathbf{z}) \cdot p(y_3|y_2, y_1, \mathbf{z}) \cdot \ldots$$




$\mathbf{y} = [y_1, ..., y_m]$ → "Ciao mondo!"

"Hello world!" → $\mathbf{x} = [x_1, ..., x_n]$

Image credit: Vaswani et al.
Slide inspired by Lucas Beyer

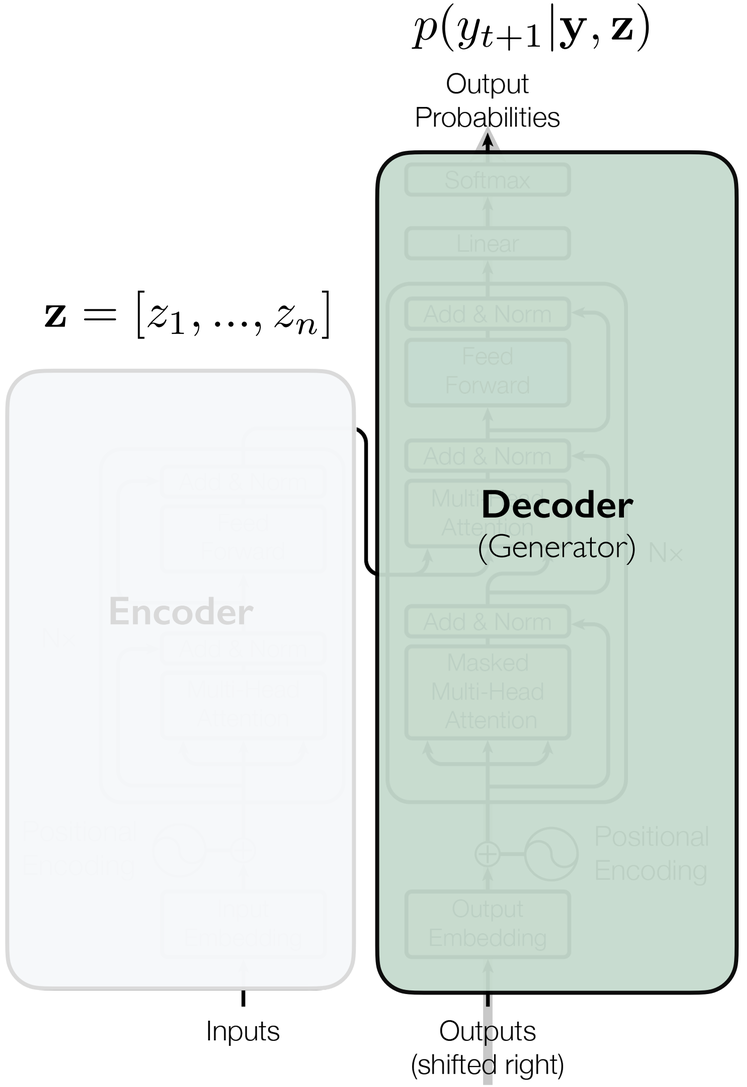
# Transformer: Overview

## Transformer Decoder

output → $\mathbf{y}$ ← encoded input $\mathbf{z}$

**Goal**: $p(\mathbf{y}|\mathbf{z})$
E.g., predict probability distribution over all possible sentences $\mathbf{y}$ in the target language, conditioned on the input sentence $\mathbf{z}$ in the source language.

**Approach**: decompose using chain rule
$p(\mathbf{y}|\mathbf{z}) = p(y_1|\mathbf{z}) \cdot p(y_2|y_1, \mathbf{z}) \cdot p(y_3|y_2, y_1, \mathbf{z}) \cdot ...$




$\mathbf{y} = [y_1, ..., y_m]$ → "Ciao mondo!"

**Decoder**: Auto-regressive prediction of next token $y_{t+1}$ from the so far predicted tokens $\mathbf{y}$ while cross-attending to the encodings $\mathbf{z}$

Each $p$ is a full pass through the model (can be slow for large sequences)

Image credit: Vaswani et al.
Slide inspired by Lucas Beyer

# Transformer: Overview

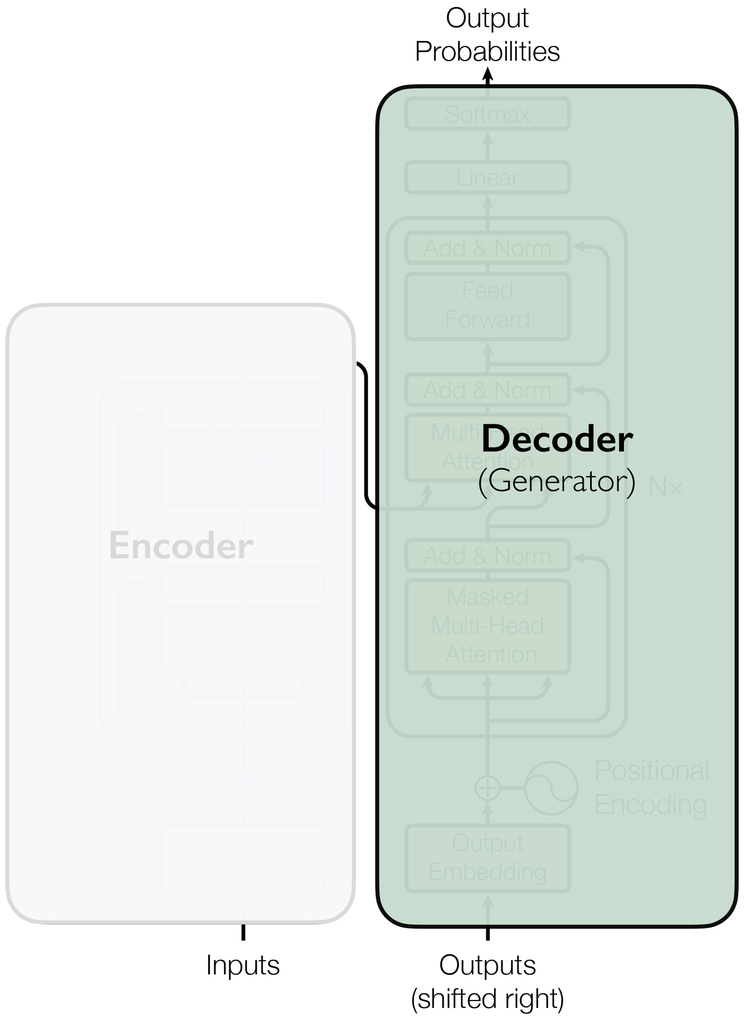





Image credit: Vaswani et al. "Attention is all you need"

# Transformer: Encoder

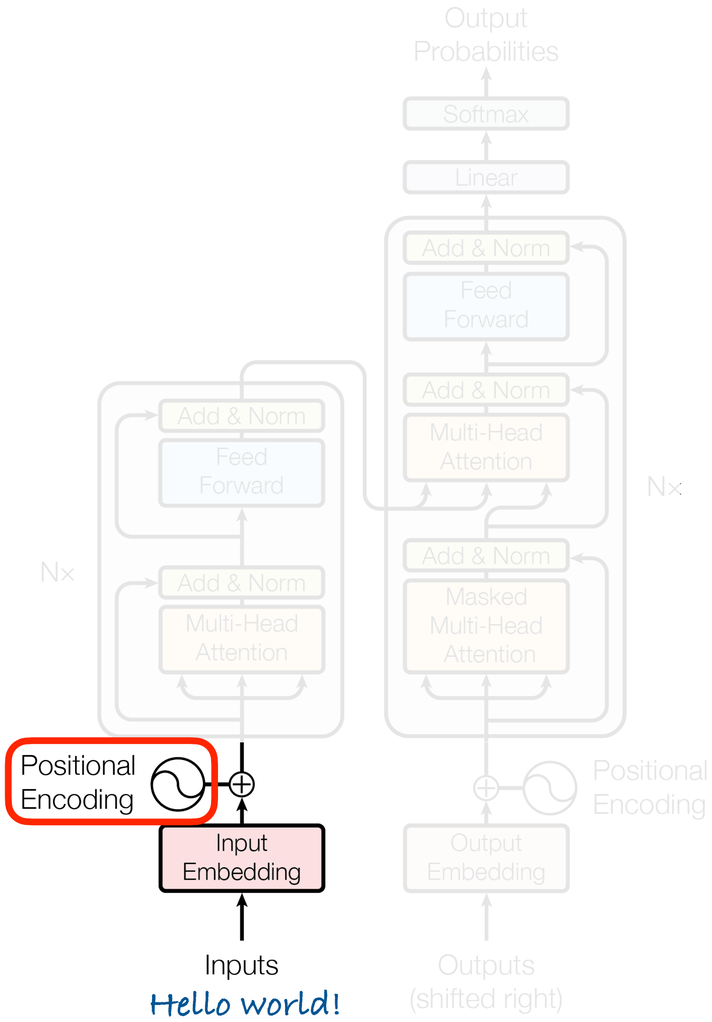



Image credit: Vaswani et al. "Attention is all you need"
Slide inspired by Lucas Beyer

Another (trivial) Vocabulary based on characters

| char | a | b | c | ... |
|---|---|---|---|---|
| index | 0 | 1 | 2 | ... |

hello -> [7,4,11,11,14]

## Input Embedding

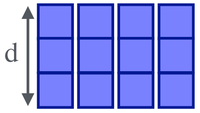
1) Split the input text into pieces, e.g., characters, words, "tokens":

Hello world! --> [Hel, lo_, wor, ld!]

2) Tokens are *indices* into the model's "vocabulary":

[Hel, lo_, wor, ld!] --> [34, 23, 89, 42]

3) Each vocabulary entry corresponds to a learned d-dimensional vector:

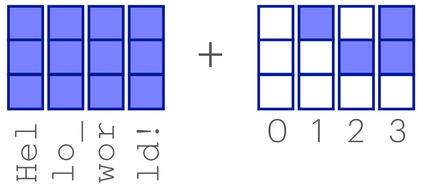
[34, 23, 89, 42] --> d  = [[-0.789, -0.708, ...], [...], [...], [...]]

## Positional Encoding

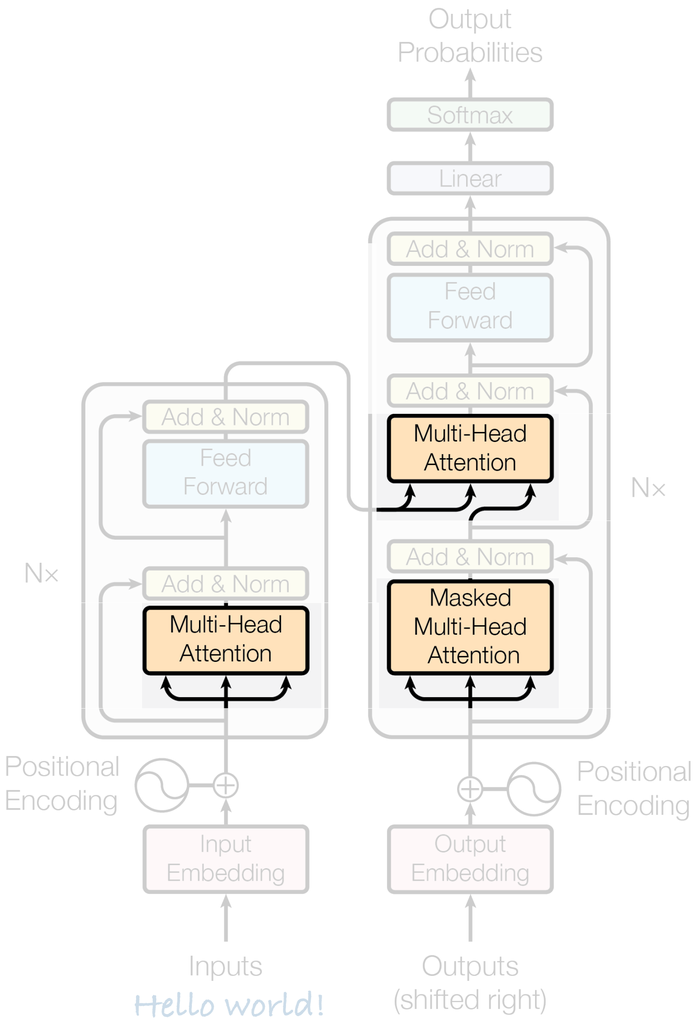
**Problem:** Attention is permutation invariant (like a dict), but language is not!

The food was good, not bad at all. ≠ The food was bad, not good at all.

**Solution**: Need to encode position of each token, and add:

Input x = 

# Transformer: Encoder

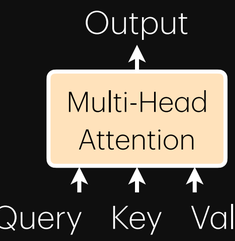



Attention

Image credit: Vaswani et al. "Attention is all you need"

Slide inspired by Lucas Beyer

# Transformer: Attention




YouTube CH

Attention Query


SEONG-JIN CHO – Piano Concerto in E minor, Op. 11 (final stage of the Chopin Competition 2015)

17 Mio. Aufrufe • vor 8 Jahren

17TH INTERNATIONAL FRYDERYK **CHOPIN PIANO** COMPETITION 18 October 201...

Allegro maestoso | Romance. Larghetto | Rondo.... 3 Stellen

Key 1

Score 1 = 0.05


Attention mechanism: Overview

116.164 Aufrufe • vor 11 Monaten

This video introduces **you** to the **attention** mechanism, a powerful technique that allows neural networks to focus on specific parts ...

Untertitel

Key 2

Score 2 = 0.91


Day 1 Highlights // ROCKWOOL Canada Sail Grand Prix | SailGP

13.849 Aufrufe • vor 19 Stunden

Subscribe today https://bit.ly/2IZR3n4 Sign up to the SailGP fan loyalty program, The Dock at sailgp.com/dock Follow SailGP On ...

Neu

Key 3

Score 3 = 0.04

# Transformer: Attention

## Attention as "soft" dictionary lookup

1) Attention weights a are query-key similarities:

$$\hat{a}_i = \mathbf{q} \cdot \mathbf{k}_i$$

Normalize via softmax: $a_i = e^{\hat{a}_i} / \sum_j e^{\hat{a}_j}$

2) Output z is the attention-weighted sum of values v:

$$\mathbf{z} = \sum_i a_i \mathbf{v}_i = \mathbf{a} \cdot \mathbf{v}$$

3) Usually, k and v are derived from the same input x:

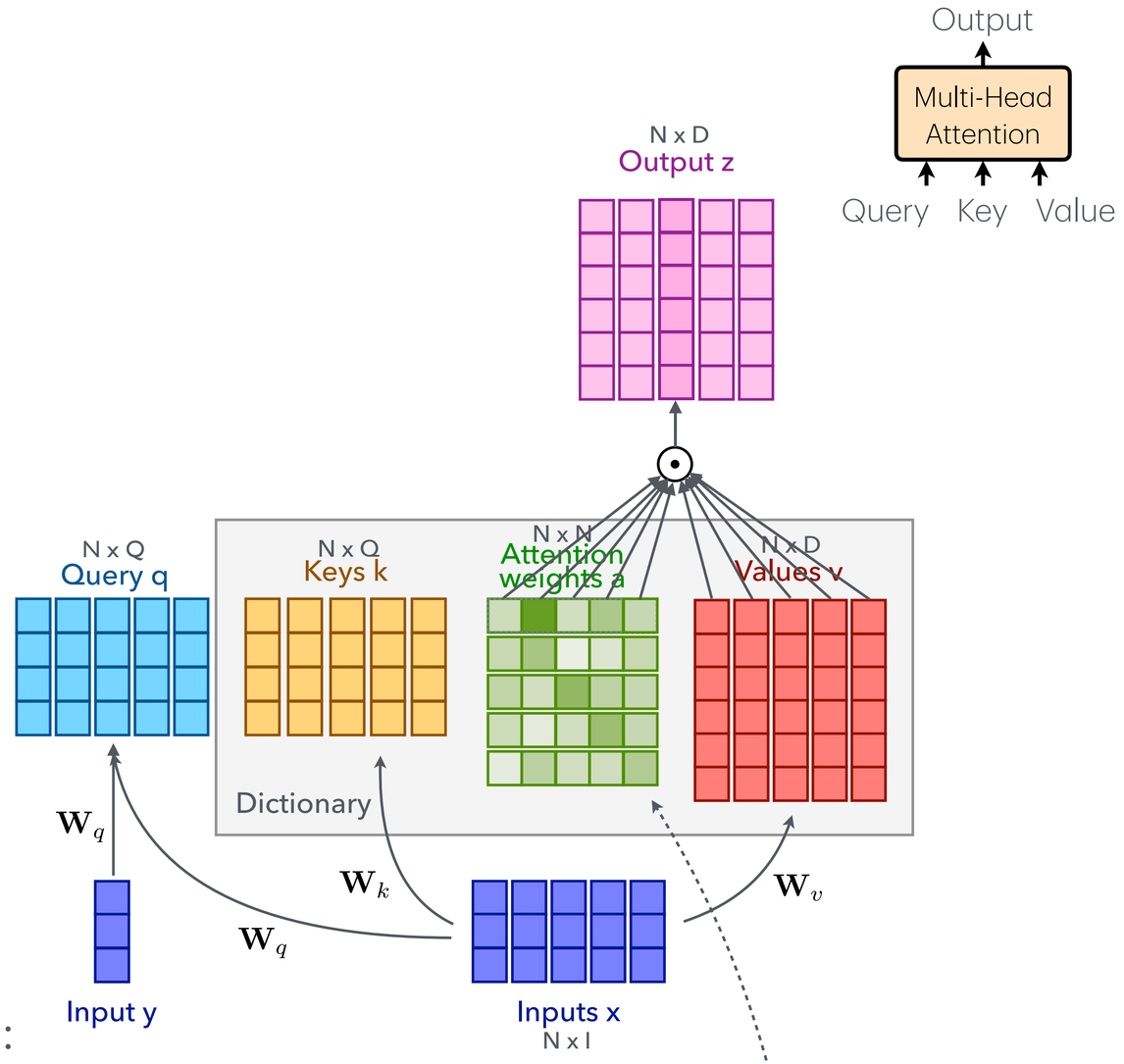
$$\mathbf{k} = \mathbf{W}_k \cdot \mathbf{x} \qquad \mathbf{v} = \mathbf{W}_v \cdot \mathbf{x}$$

4) Query q comes from a separate input y *(cross-attention)*:

$$\mathbf{q} = \mathbf{W}_q \cdot \mathbf{y}$$

or from the same input x *(self-attention)*:

$$\mathbf{q} = \mathbf{W}_q \cdot \mathbf{x}$$



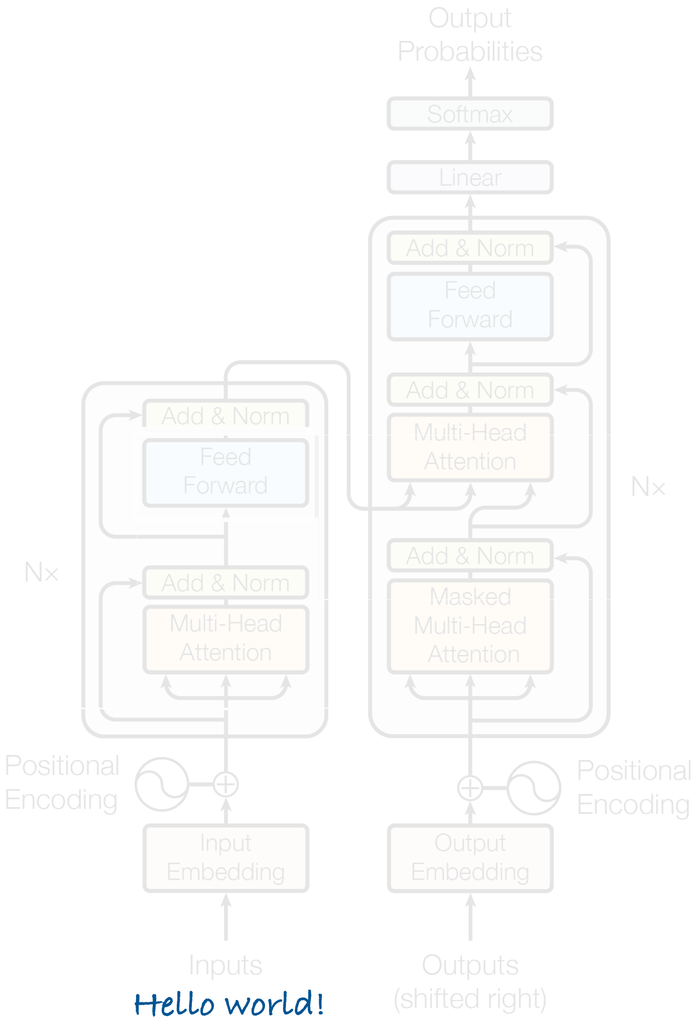

Note that the complexity is $\mathcal{O}(N^2)$

($N$: #input tokens / context length)



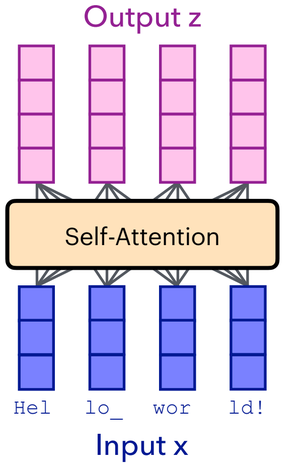
Slide inspired by Lucas Beyer

# Transformer: Encoder




## Self Attention

The **input sequence** us used to create **queries**, **keys** and **values**. Each token can "look around" the whole input, and decide how to update the **output representation** based on what is sees.




## Feed Forward

A simple MLP applied to each token individually:

$$z_i = W_2 \mathrm{ReLU}(W_1 x + b_1) + b_2$$

Meaning: each token is pondering for itself about what it has observed previously.

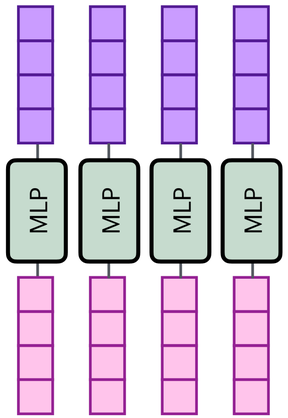
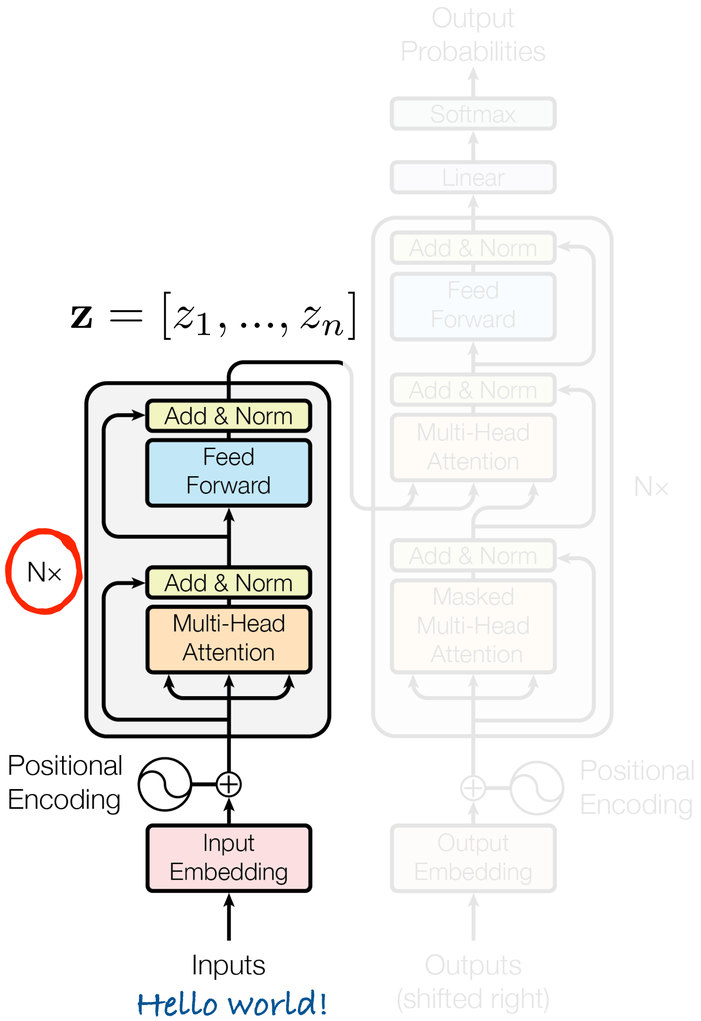



Image credit: Vaswani et al. "Attention is all you need"
Slide inspired by Lucas Beyer

# Transformer: Encoder




## Encoder

Since the input and output dimensions are identical, we can stack N such encoder blocks.

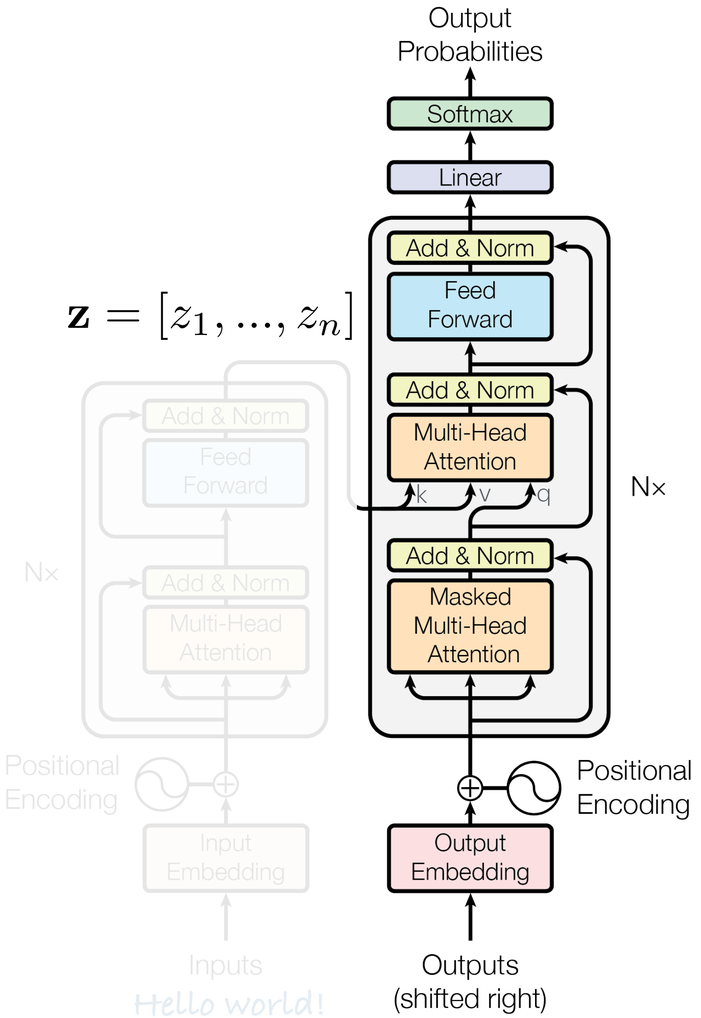
The encoder output is a "processed" version of the input tokens, i.e., an intermediate representation.

The actual transformer output, comes from the decoder.



Image credit: Vaswani et al. "Attention is all you need"

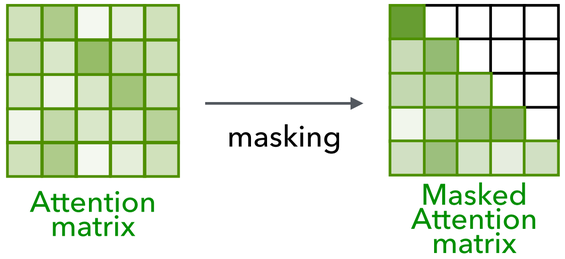
# Transformer: Decoder (next week)




## Masked Self-Attention

Like regular self-attention (as in the encoder), but masked.

Restricts each token to see only the already generated ones.




## Cross-Attention

Each decoded token can "look at" the encoder's output.

Queries come from decoder, keys and values from the encoder.



Image credit: Vaswani et al. "Attention is all you need"

# Transformer: Summary

1. Sequence modeling using RNN/LSTM
2. Motivation for the attention mechanism
3. Attention mechanism
4. Transformer (encoder)

**Next week:** Transformer (decoder)



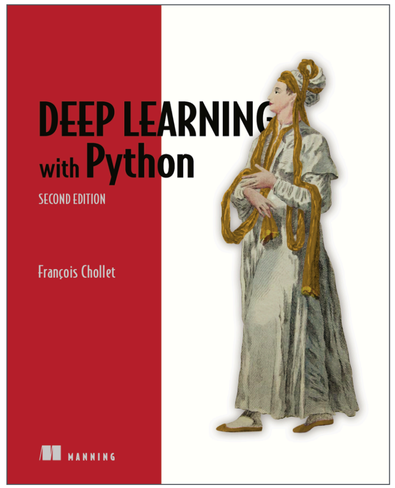
Image credit: Google Research

# References and Further Reading

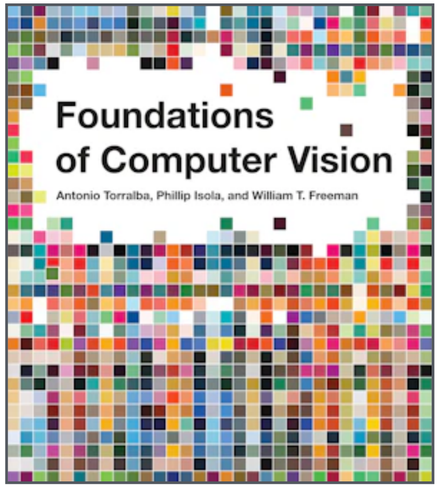
- Slides:

Example Code:

PyTorch Notebook
GPT Decoder




Chapter 11.4, 11.5




Chapter 26, 51

- Book Chapters on Transformers:
  - Deep Learning with Python (François Chollet)
  - Foundations of Computer Vision (Torralba, Isola, Freeman)

- Code example MinGPT by Andrej Karpathy https://github.com/karpathy/minGPT?tab=readme-ov-file
- RNN/LSTM by Chris Olah http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- Lecture slides by Lucas Beyer http://lucasb.eyer.be/transformer