Laboratório de Engenharia de Sistemas e Robótica

UFPB

Modelos de Linguagem Larga (LLM)

3. Transformers

Prof.: Alisson Brito (alisson@ci.ufpb.br)

LASER

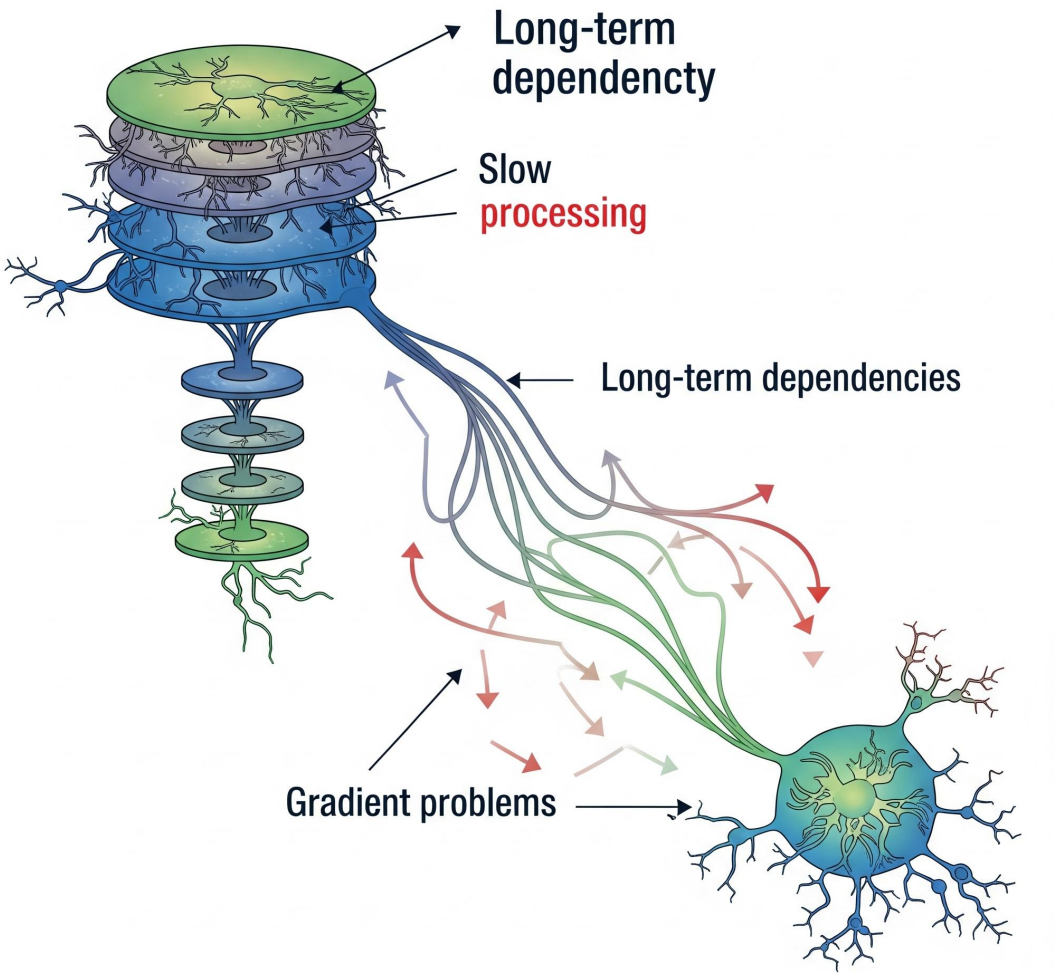
Laboratório de Engenharia de Sistemas e Robótica

UFPB

Agenda

- Revisão
- Introdução à Atenção
- Arquitetura Transformers
- Treinamento de LLMs
- Escala e Avaliação
  - Pré-processamento de datasets massivos.
  - Treinamento distribuído em larga escala.
  - Métricas de avaliação
- Atividade Prática:
  - Simulação de treinamento em ambiente controlado, ajuste de hiperparâmetros

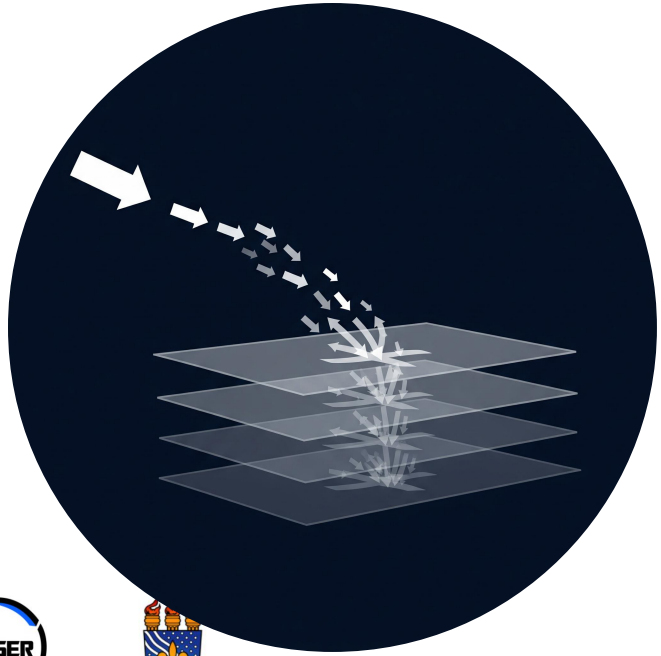
Antes de Atenção e Transformers…



RNN: Limitações

- As RNNs processam as sequências palavra por palavra.
- Treinamento lento para sequências longas.
- Dificuldade em lidar com dependências de longo prazo.
- Problema de desvanecimento ou explosão de gradientes.

Desvanecimento dos Gradientes



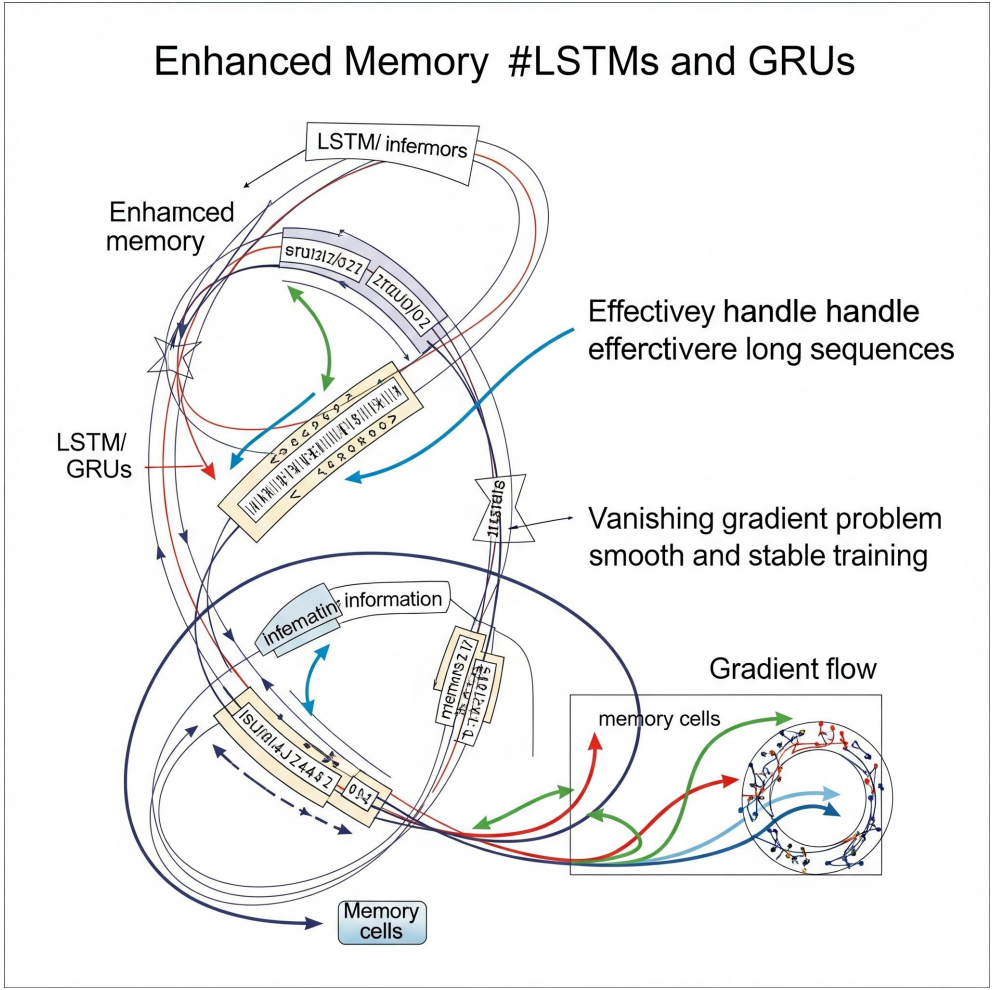
- Problema comum em redes neurais recorrentes.
- Gradientes ficam muito pequenos durante o treinamento.
- Dificulta o aprendizado de dependências de longo prazo.
- Prejudica a capacidade da RNN de capturar informações importantes.

Exemplo de Desvanecimento dos Gradientes

- Sentenças longas prejudicam o treinamento de RNNs.
- Dificuldade em reter informações iniciais da frase.
- Exemplo: "O **menino** que jogava bola na rua escorregou e caiu."
- O modelo pode "esquecer" o sujeito antes do final.

RNN, LSTM, GRU

- LSTMs e GRUs: Soluções para Limitações de RNNs
- Memória Aprimorada para Dependências Longas
- Resolvem o Problema de Desvanecimento do Gradiente
- Treinamento Mais Eficiente e Estável



Laboratório de Engenharia de Sistemas e Robótica

UFPB

Gates: portas para controle

01 — LSTMs e GRUs usam "gates" para controlar o fluxo de informação.

02 — Exemplos de gates: input, forget e output gates.

03 — Gates permitem adicionar ou remover informações da memória da rede.

04 — Isso ajuda a resolver o desvanecimento de gradientes em sequências longas.

LSTM X Atenção

- LSTMs e GRUs são **sequenciais**.
- Atenção permite processamento **paralelo**.
- LSTMs e GRUs têm gargalo de informação.
- Mecanismo de atenção captura **dependências**.

1. Introdução à Atenção

- Conforme visto na prática da aula passada, a codificação simples de word embeddings estáticos não resolve alguns problemas
  - Mesma palavra com significados diferentes
  - Dependência de longo prazo
  - Semântica do contexto
  - Ordem das palavras
  - Etc
- Na análise de sentenças podemos utilizar os mesmos tokens, mas ter semânticas distintas
  - O estudante consome o café **!=** O café consome o estudante

1. Introdução à Atenção

- Um dos modelos utilizados na prática foi o BERT - é um Transformer
  - Bidirectional Encoder Representations from Transformers

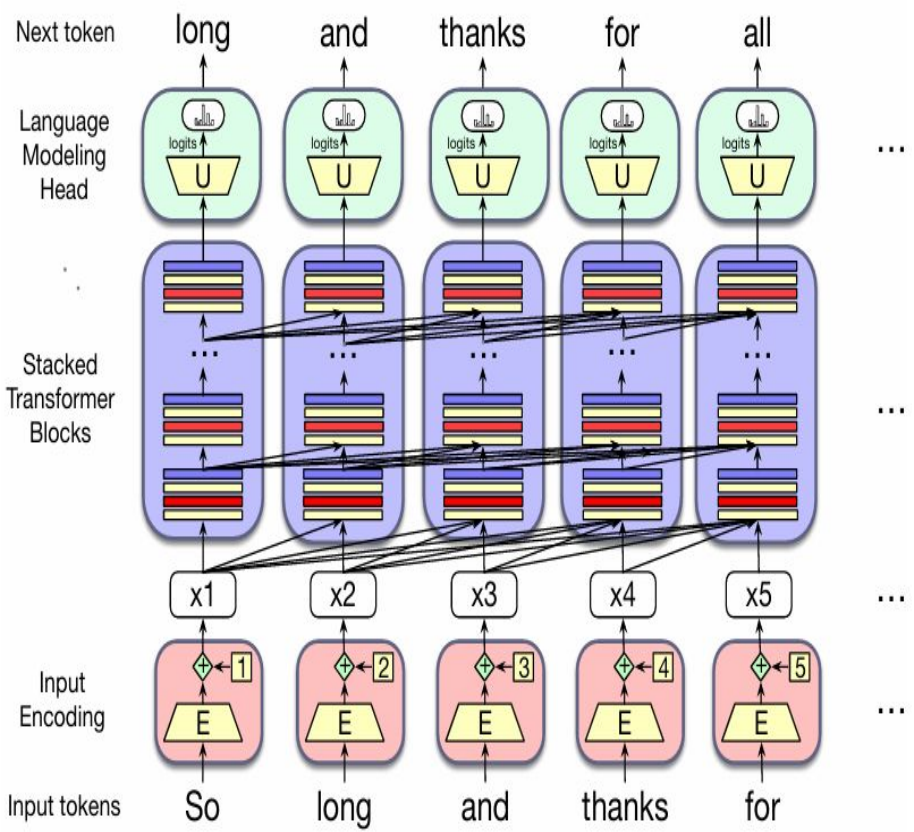
- Os modelos Transformer tem um mecanismo chamado "**Attention**" que resolve muitos dos problemas de forma eficiente (altamente paralelizável)

- Ao invés de processar palavra por palavra, considera todas elas – Tornando possível o "entendimento" do contexto e a relação que cada palavra tem na sentença

1. Introdução à Atenção

A atenção é uma forma de construir representações contextuais do significado de um token, observando e integrando informações dos tokens circundantes

Isso ajuda o modelo a aprender como os tokens se relacionam entre si em grandes extensões.



1. Introdução à Atenção

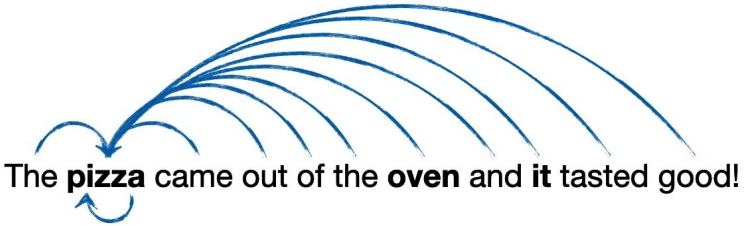
- No contexto de Processamento de Linguagem Natural (PLN), **atenção** é um mecanismo que permite ao modelo **focar nas partes mais importantes** de uma frase ou texto.

- Ele distribui **pesos de relevância** entre as palavras, destacando quais são mais úteis para responder ou prever.

    - Foi proposto em 2014 em modelos de tradução automática (Bahdanau et al.).
    - Depois, virou peça central dos Transformers (2017, "*Attention is All You Need*").
    - Hoje, é a base de modelos como GPT, BERT, LLaMA e outros.

1. Introdução à Atenção

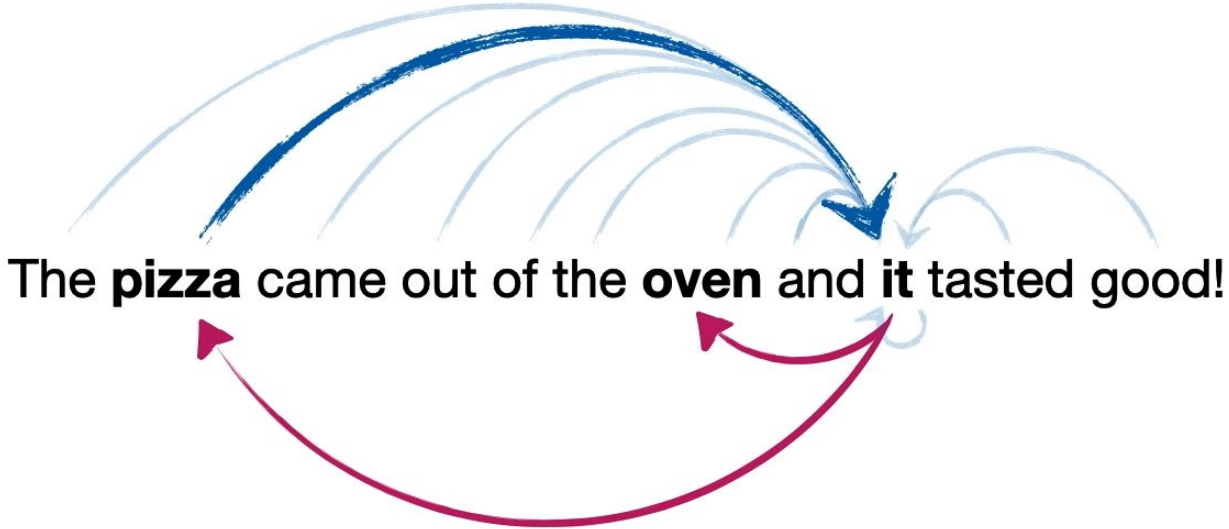
- Self-Attention
- Calcula similaridades entre cada palavra em uma sentença
- …incluindo a palavra de referência com ela mesma



...

Self-Attention

- A ideia é calcular score de similaridade para estabelecer relacionamento entre as palavras
- No exemplo, a palavra "pizza" ter um score maior em relação à palavra "it", terá grande impacto em como o embedding de "it" será calculado



Atenção Escalar (scaled dot-product attention)

- Forma matemática básica de calcular atenção mede o quanto uma palavra deve prestar atenção em outra, usando produto interno (dot product) entre vetores.

- Fórmula principal: $$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- Observando primeiramente as variáveis Q, K e V – São provenientes da terminologia de bancos de dados
  a. Q → Query
  b. K → Key
  c. V → Value

Atenção Escalar (scaled dot-product attention)

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- Query é o vetor de consulta para procurar informações relevantes
- Key é o vetor de chaves, que é comparado com a Query para determinar o quão é relevante cada Value
- Query, Key e Value são computados antecipadamente
- Query e Key são comparados para determinar os pesos
- Os pesos são aplicados aos Value, gerando um grau de similaridade

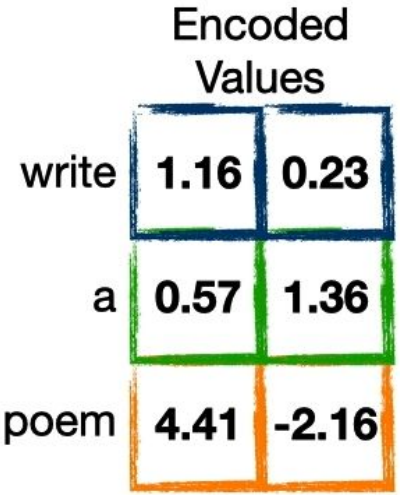
Atenção Escalar (scaled dot-product attention)

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- Q (Query): O que eu quero saber?
- K (Key): Onde estão as informações possíveis?
- V (Value): Qual informação vou recuperar?
- Similaridade (QK^T): Produto interno entre Query e Key para medir quão parecidas são.
- d_k: É a dimensão de k.
- Escala ($\sqrt{d_k}$): Evita valores muito grandes no produto escalar
- Softmax: Converte os valores em probabilidades, cuja soma será 1.

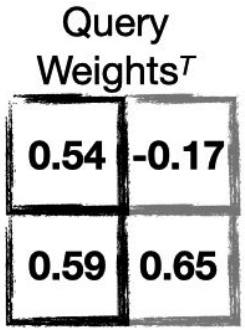
Self-Attention

- Inicialmente, para calcular **Q**, **K** e **V**, precisaremos dos valores de entrada e pesos respectivos (w)
- Vejamos um exemplo utilizando 2 embeddings para cada palavra e empilhamos para criar uma matriz (ótimo para paralelizar)
  - Na prática, conforme visto no exercício da aula passada, essa quantidade é bem maior



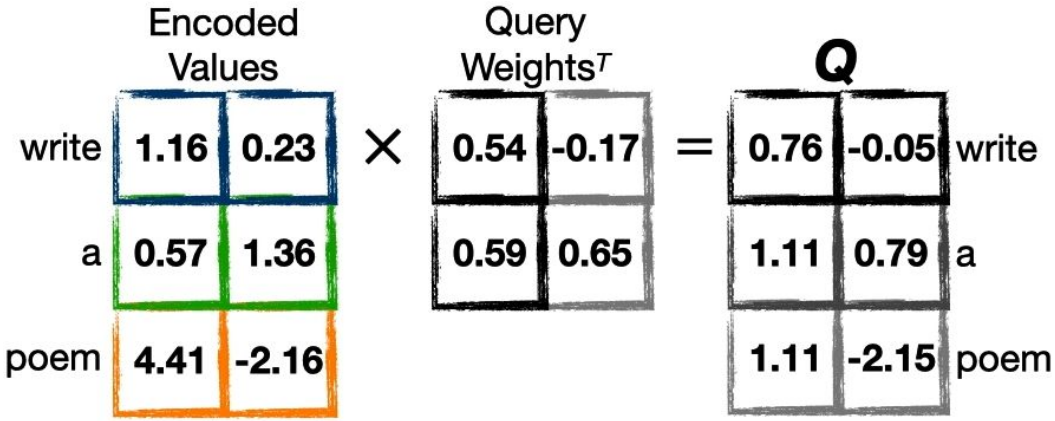
Self-Attention

- **Q** tem relação direta com a entrada (embeddings), portanto sua dimensão (d) será 2
  - O que nos possibilita criar uma matriz de pesos de 2x2
  - A regra aqui é basicamente não inviabilizar a multiplicação de matrizes (normalmente em redes neurais)



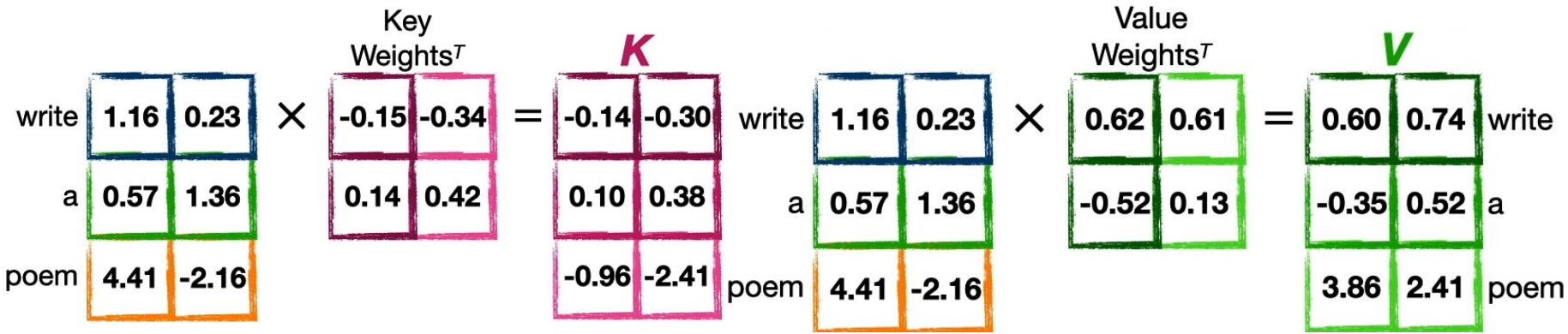
Self-Attention

- Com a multiplicação das matrizes, obtemos os valores de **Q**
  - A matriz 2x2 possibilita obter 2 valores de **Q** para cada palavra



Self-Attention

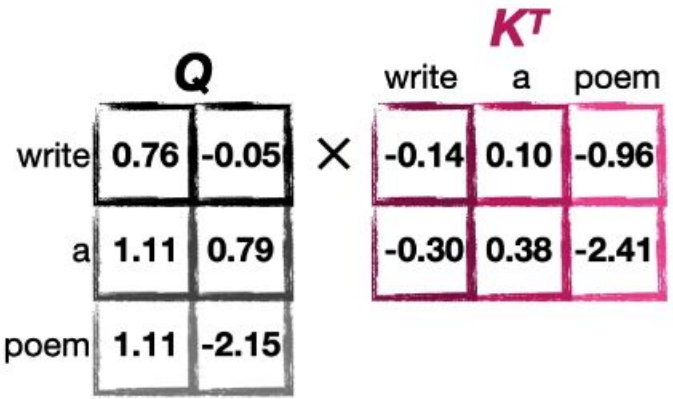
- Similarmente para calcular **K** e **V**



Self-Attention

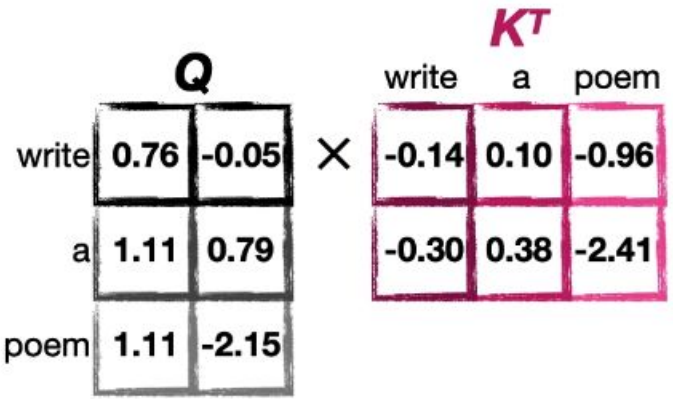
- Em posse dos valores calculados, seguindo a fórmula, podemos calcular **QK^T**:

$$Attention(Q, K, V) = SoftMax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



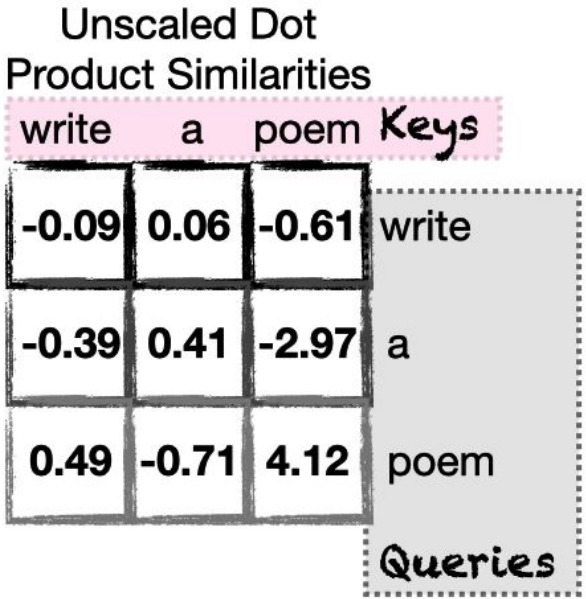
Self-Attention

- Detalhe: A multiplicação das matrizes Q e K^T nos dá um produto escalar entre os pares, o que pode ser usado como uma medida de similaridade



Self-Attention

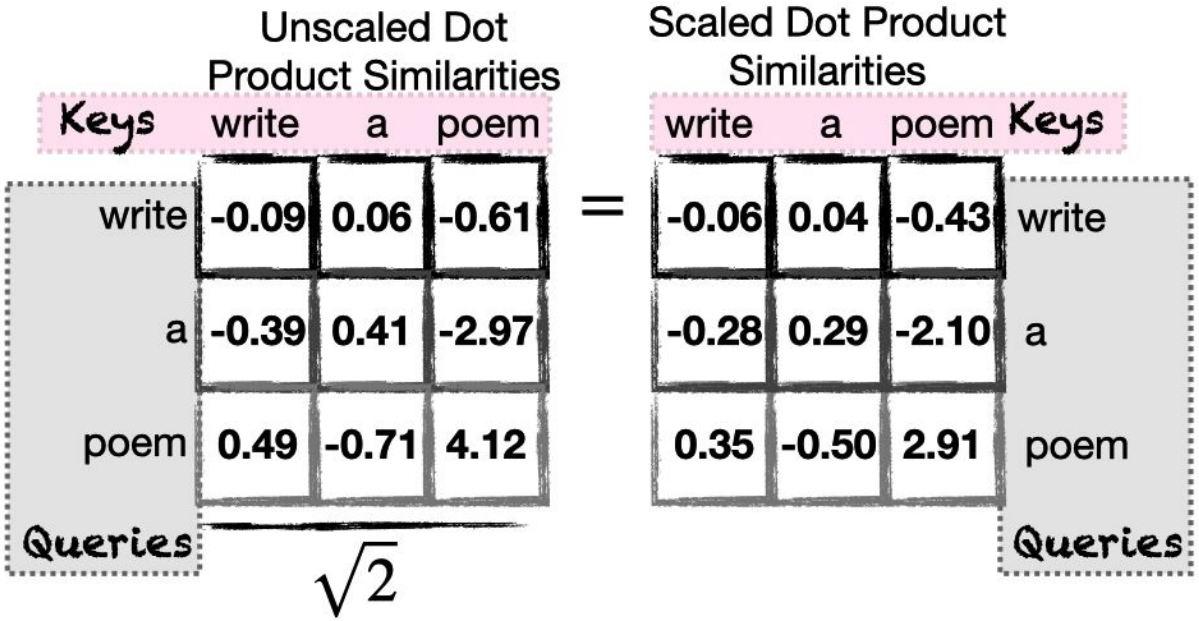
- Assim, através da matriz resultante, obtemos a primeira similaridade (unscaled) entre todas as possíveis combinações entre as palavras



Self-Attention

$$Attention(Q, K, V) = SoftMax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- Seguindo:
  - **K** é a "chave", que deve ter relação direta com a entrada (embeddings), portanto a dimensão (d) de **K** será 2



Self-Attention

- Seguindo:
  - Aplicando SoftMax a cada linha
  - A soma dos valores é igual a 1

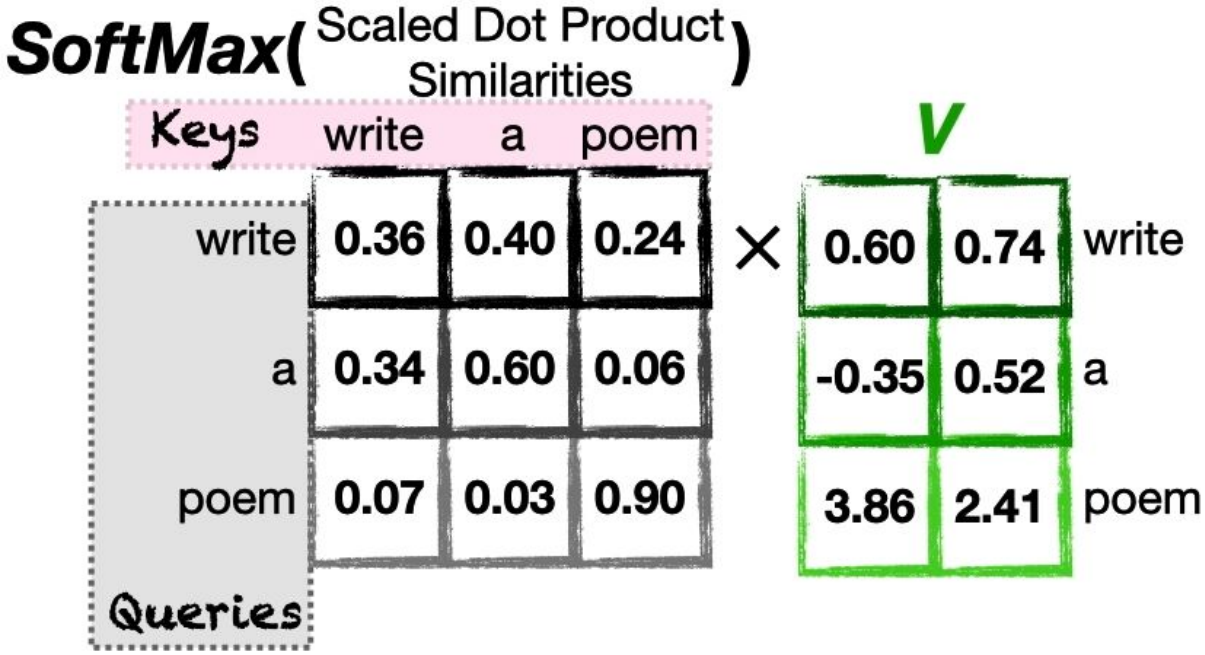
$$Attention(Q, K, V) = SoftMax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

SoftMax(Scaled Dot Product Similarities **)**

Keys \ Queries	write	a	poem
write	0.36	0.40	0.24
a	0.34	0.60	0.06
poem	0.07	0.03	0.90

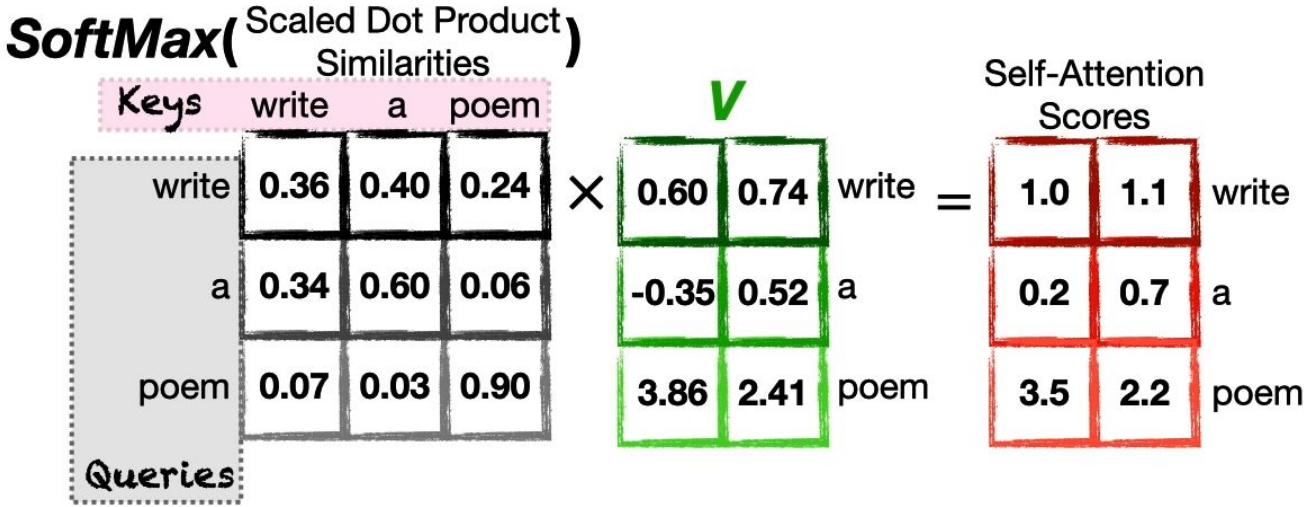
Self-Attention

- Seguindo:
  - Multiplicação por **V**



Self-Attention

- Multiplicação por **V**
  - Ao multiplicar pelos percentuais calculados na matriz, vamos obter os scores de atenção (attention scores)
  - Em outras palavras, a matriz de probabilidades diz o quanto cada palavra influencia no valor final dos embeddings calculados para cada token



Masked Self-Attention

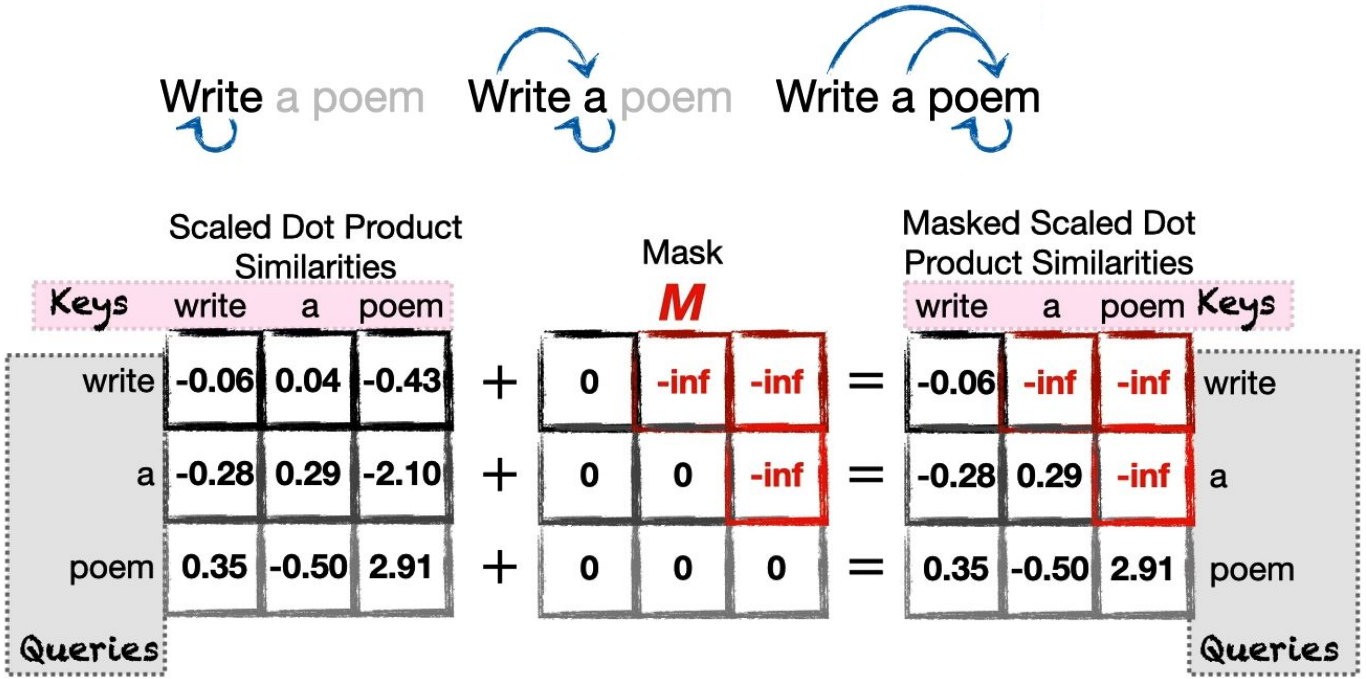
- A principal diferença é não calcular similaridades para tokens posteriores

Masked Self-Attention

- Adicionamos uma matriz **M** que funciona como máscara, não permitindo acesso aos tokens subsequentes no cálculo da atenção

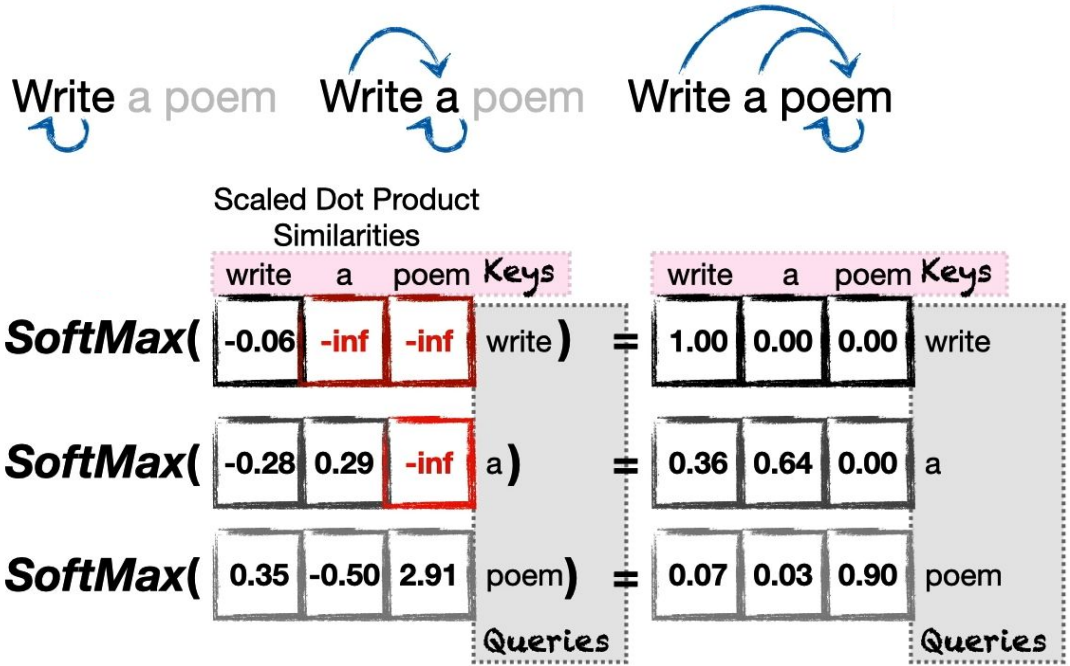
$$MaskedAttention(Q, K, V, M) = SoftMax\left(\frac{QK^T}{\sqrt{d_k}} + M\right)V$$

Masked Self-Attention



Masked Self-Attention

- Simples, mas eficaz!



Attention

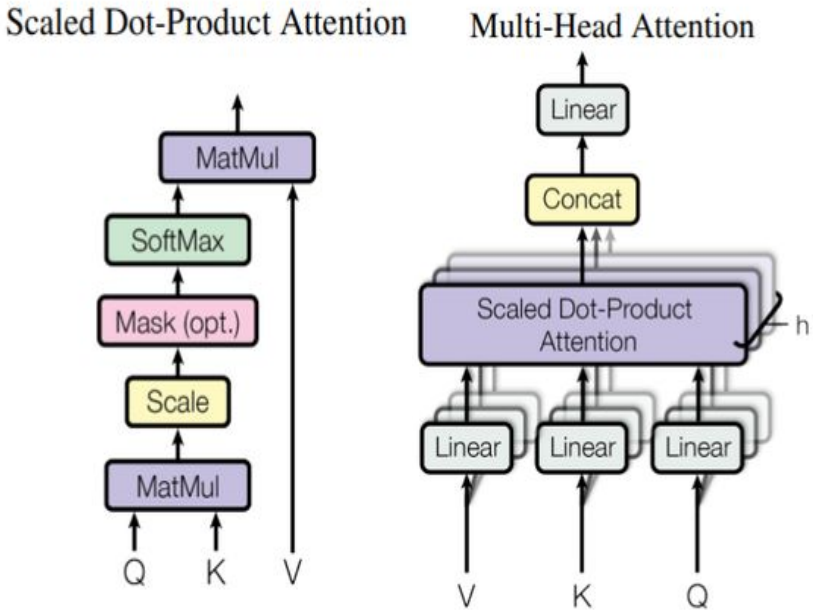
- A técnica de Atenção não resolve todos os problemas
- Apesar de melhorar o entendimento entre os tokens, do jeito que foi calculado, o problema da ordem dos word tokens ainda permanece
  - O estudante consome o café **!=** O café consome o estudante

- Adentrando mais na arquitetura dos Transformers…

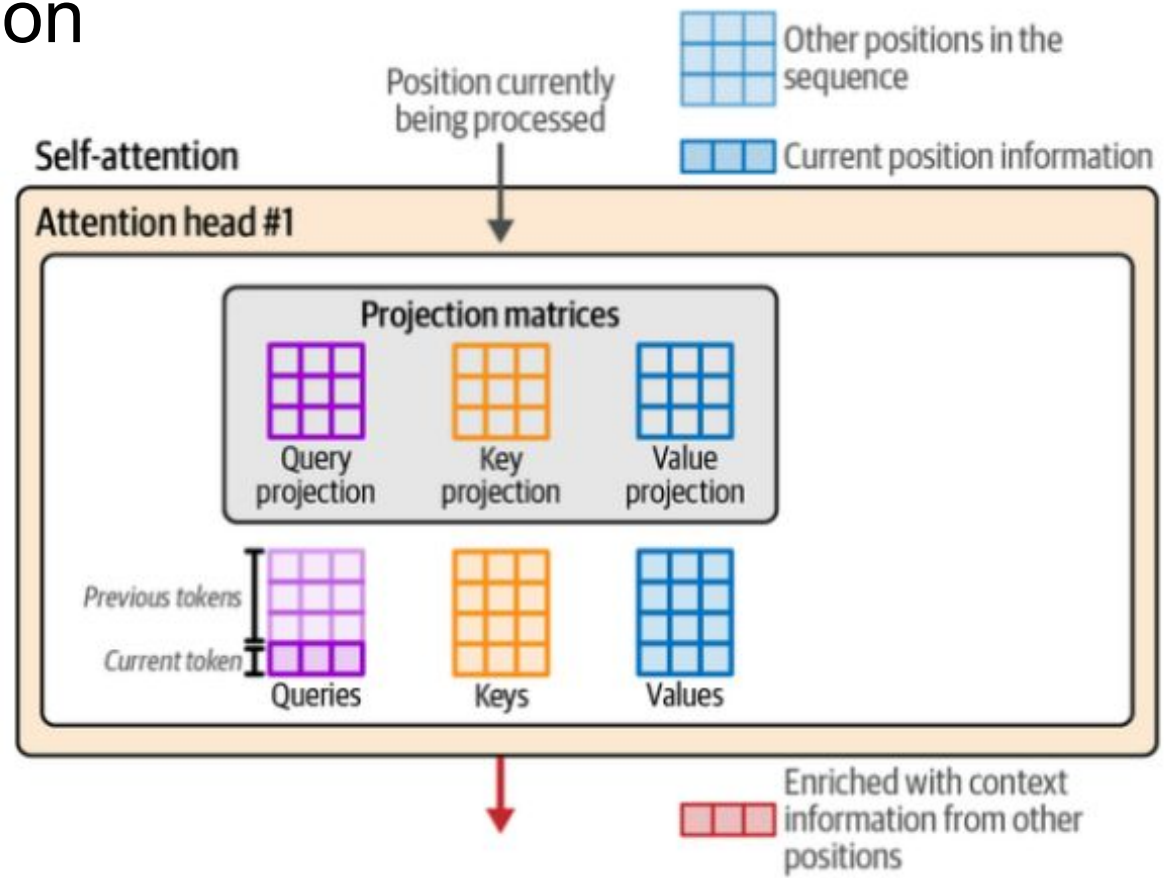
Attention

Self-Attention: Permite que o modelo pondere dinamicamente a importância de cada patch em relação a todos os outros, possibilitando a captura de dependências complexas e informações contextuais em toda a imagem.

Multi-Head Attention: Em vez de depender de um único mecanismo de atenção, utilizam múltiplas "cabeças" de atenção em paralelo. Cada cabeça pode focar em diferentes aspectos ou regiões da imagem, permitindo que o modelo aprenda um conjunto de relações mais rico e diverso.



Self-attention



Atenção Escalar (scaled dot-product attention)

["Eu", "gosto", "de", "café"]

Passo 1: Vetores de Query (Q), Key (K) e Value (V)

Token	Q	K	V
Eu	[1,0]	[1,0]	[1,0]
gosto	[0,1]	[0,1]	[0,1]
de	[1,1]	[1,1]	[1,1]
café	[0,1]	[0,1]	[0,1]

Atenção Escalar (scaled dot-product attention)

Passo 2: Produto escalar Q·K

```
score("Eu","Eu")    = Q("Eu") · K("Eu")    = [1,0]·[1,0] = 1
score("Eu","gosto") = [1,0]·[0,1] = 0
score("Eu","de")    = [1,0]·[1,1] = 1
score("Eu","café")  = [1,0]·[0,1] = 0
```

Score total: [1, 0, 1, 0]

Atenção Escalar (scaled dot-product attention)

Passo 3: Softmax

Transformamos os scores em **probabilidades** para somarem 1:

softmax([1, 0, 1, 0]) ≈ [0.42, 0.15, 0.42, 0.15]

Passo 4: Multiplicação pelo Value (V)

Multiplicamos cada vetor V pelo score correspondente e somamos:

```
Atenção("Eu") = 0.42*[1,0] + 0.15*[0,1] + 0.42*[1,1] + 0.15*[0,1]
              = [0.42+0+0.42, 0+0.15+0.42+0.15]
              = [0.84, 0.72]
```

```python
import numpy as np  #calcular atenção escalar com numpy.
from scipy.special import softmax

# Vetores de exemplo (dimensão 3)
Q = np.array([[1, 0, 1]])
K = np.array([[1, 0, 1], [0, 1, 0], [1, 0, 1], [0, 1, 0]])
V = np.array([[0.1, 0.2], [0.0, 0.1], [0.3, 0.0], [0.2, 0.2]])

# Produto escalar QK^T
scores = Q @ K.T / np.sqrt(K.shape[ 1])
weights = softmax(scores)

# Multiplicação pelos valores
output = weights @ V
print("Scores:", scores)
print("Pesos:", weights)
print("Saída da atenção:", output)
```

```
Scores: [[1.15470054 0.         1.15470054 0.        ]]
Pesos: [[0.38018422 0.11981578 0.38018422 0.11981578]]
Saída da atenção: [[0.17603684 0.11198158]]
```

2. Transformers

- Arquitetura que revolucionou o processamento de linguagem natural.
- Transformers entendem texto olhando para todas as palavras de uma vez e ponderando a importância de cada uma, permitindo aprender relações complexas em grandes volumes de dados.
- Captura de dependências longas com precisão.
- Permite treinar LLMs gigantes com eficiência.

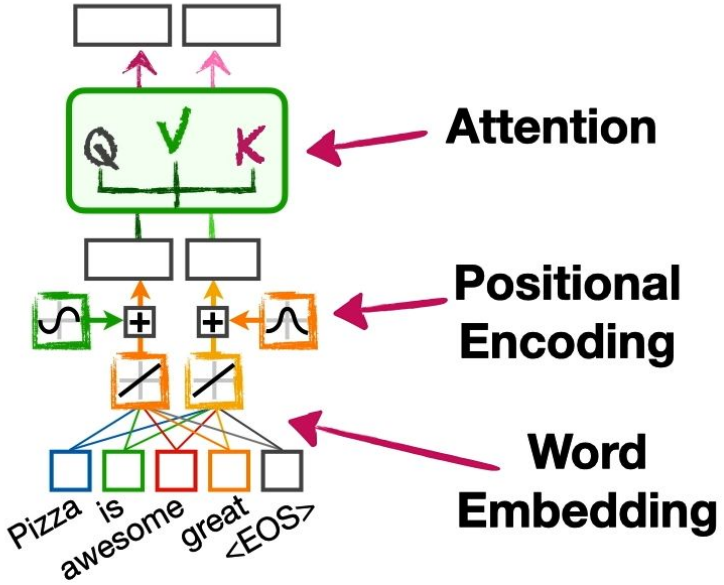
Arquitetura Transformers

Para preservar informações sobre a ordem dos elementos, o Transformer emprega **positional encodings**, adicionados aos embeddings de entrada.

A arquitetura básica é composta por um **encoder** (responsável por mapear a entrada para uma representação contextual) e um **decoder** (responsável por gerar a saída, etapa a etapa)

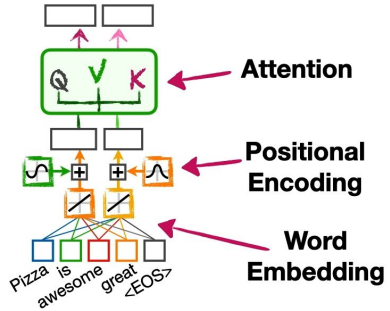
2. Transformers

- Fundamentos da arquitetura Transformer:



2. Transformers

- A camada de Positional Encoding resolve justamente o problema da ordem de palavras
  - Positional Encoding é independente da Attention
  - Permite levar em consideração a ordem das palavras



Embeddings e Positional Encoding

- Cada palavra/token é transformado em um vetor (embedding).

```
"gato" → [0.12, -0.33, 0.58, ...]
"cachorro" → [0.10, -0.30, 0.60, ...]
```

- Como o Transformer não é sequencial, adiciona-se o Positional Encoding para indicar a posição de cada palavra na frase
- Positional Encoding adiciona informação de posição, porque o Transformer **não é sequencial**.

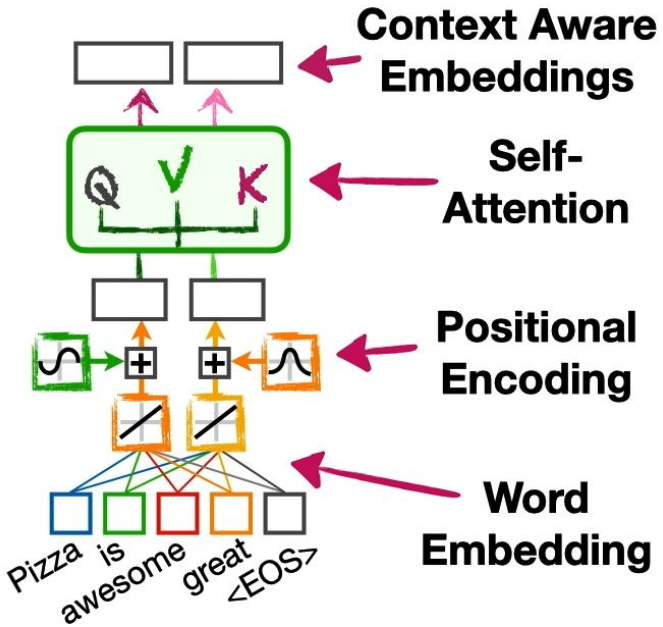
```
Frase: "O gato dorme"
Embeddings + Positional Encoding:
"O" → [0.2, 0.1, 0.5] + pos0
"gato" → [0.12, -0.33, 0.58] + pos1
"dorme" → [0.3, 0.0, 0.7] + pos2
```

```python
import numpy as np

def positional_encoding(max_len, d_model):
    pos = np.arange(max_len)[:, np.newaxis]
    i = np.arange(d_model)[np.newaxis, :]
    angle_rates = 1 / np.power(10000, (2 * (i // 2)) /
np.float32(d_model))
    angle_rads = pos * angle_rates
    angle_rads[:, 0::2] = np.sin(angle_rads[:, 0::2])
    angle_rads[:, 1::2] = np.cos(angle_rads[:, 1::2])
    return angle_rads

pos_enc = positional_encoding(5, 8)
# 5 tokens, 8   dimensões
print(pos_enc)
```

```
[[ 0.00000000e+00  1.00000000e+00  0.00000000e+00  1.00000000e+00
   0.00000000e+00  1.00000000e+00  0.00000000e+00  1.00000000e+00]
 [ 8.41470985e-01  5.40302306e-01  9.98334166e-02  9.95004165e-01
   9.99983333e-03  9.99950000e-01  9.99999833e-04  9.99999500e-01]
 [ 9.09297427e-01 -4.16146837e-01  1.98669331e-01  9.80066578e-01
   1.99986667e-02  9.99800007e-01  1.99999867e-03  9.99998000e-01]
 [ 1.41120008e-01 -9.89992497e-01  2.95520207e-01  9.55336489e-01
   2.99955002e-02  9.99550034e-01  2.99999550e-03  9.99995500e-01]
 [-7.56802495e-01 -6.53643621e-01  3.89418342e-01  9.21060994e-01
   3.99893342e-02  9.99200107e-01  3.99998933e-03  9.99992000e-01]]
```

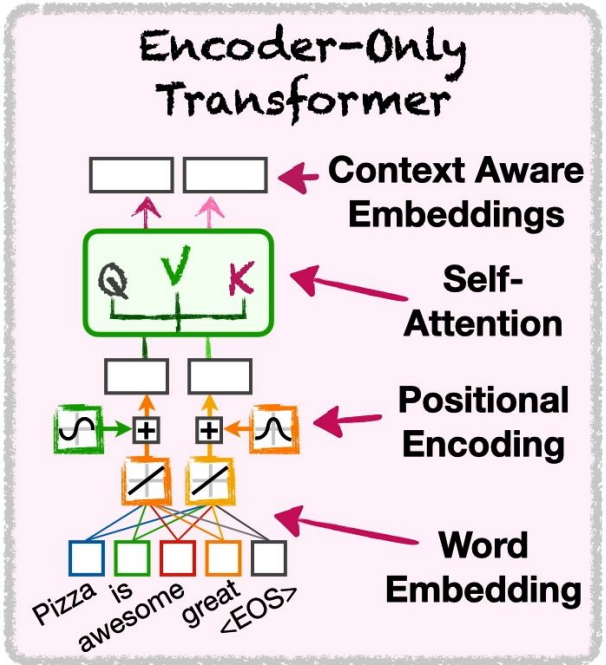
2. Transformers

- Transformers com apenas Self-Attention
  - Bons em produzir embeddings com contexto

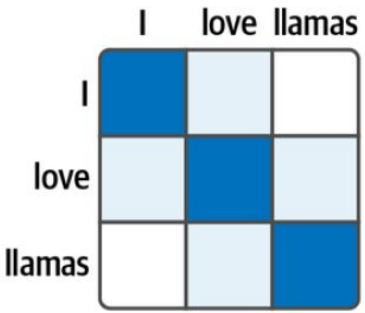


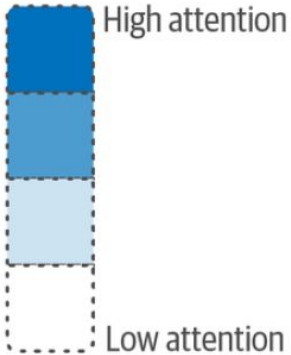
2. Transformers

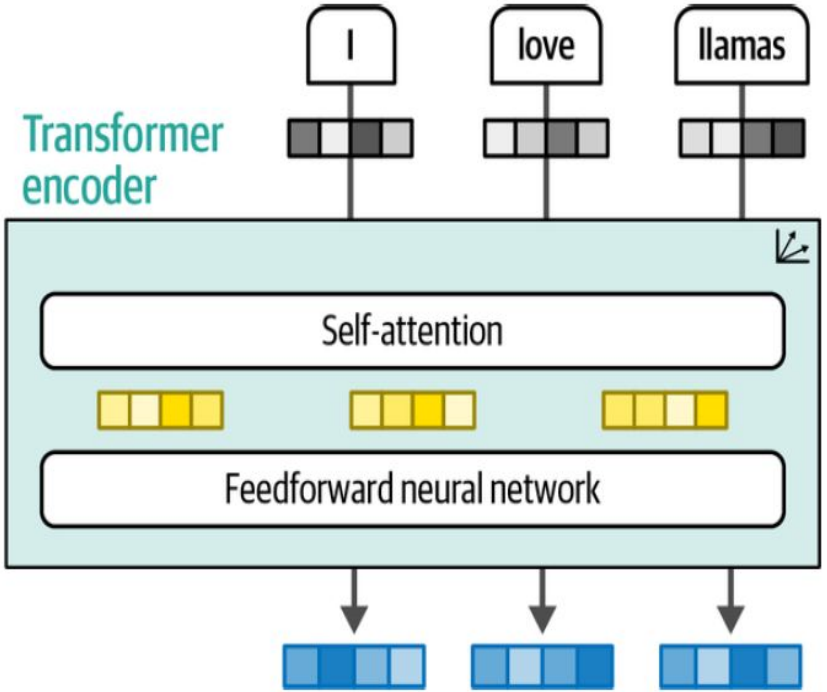
- Esses modelos podem ser chamados de **Encoder-Only Transformers**



Encoder

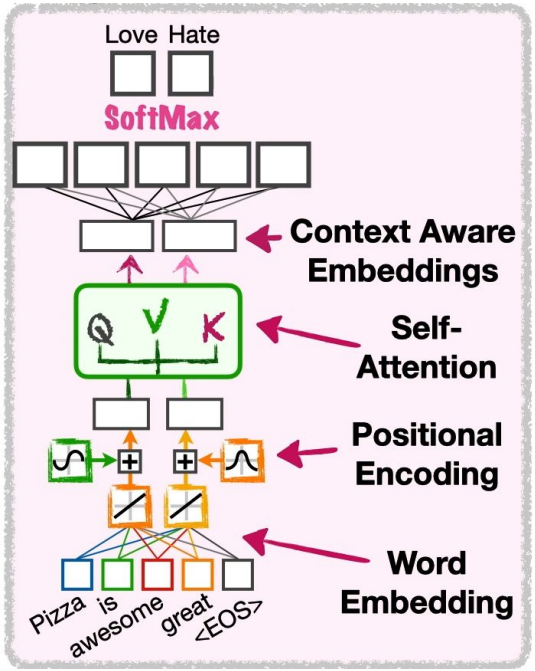

I
love
llamas
I
love
llamas


High attention
Low attention


I
love
llamas
Transformer encoder
Self-attention
Feedforward neural network

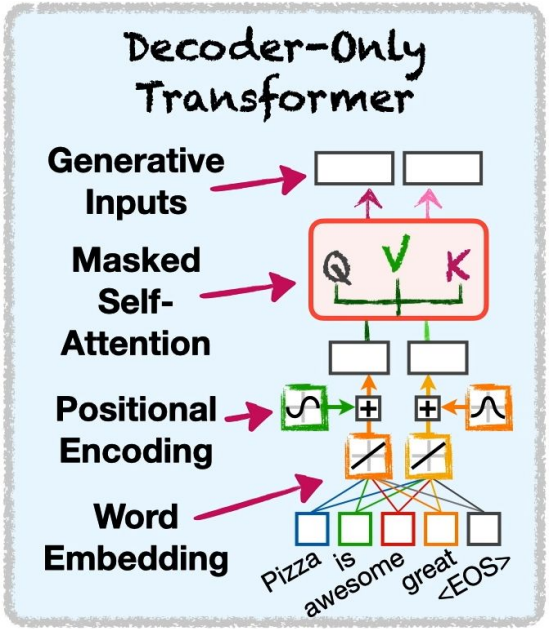
2. Transformers

- Comum utilizá-los como base para treinar modelos classificadores, entre outros

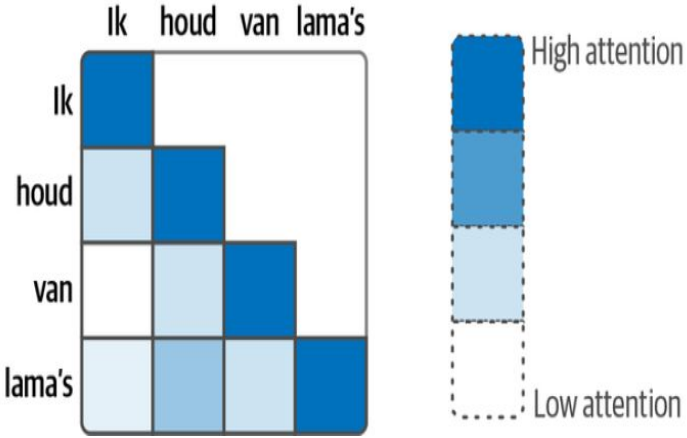


2. Transformers

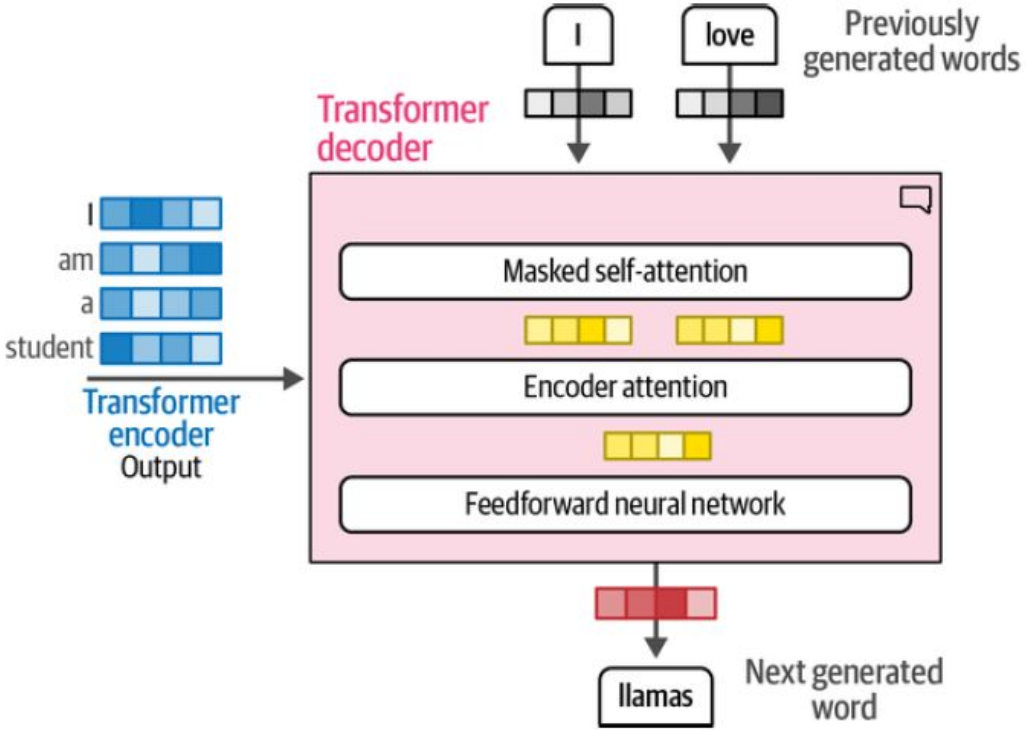
- Por outro lado, modelos que utilizam apenas Masked Self-Attention são os **Decoder-Only Transformers**
  - Produzem **generative inputs**



Decoder


Ik
houd
van
lama's
Ik
houd
van
lama's

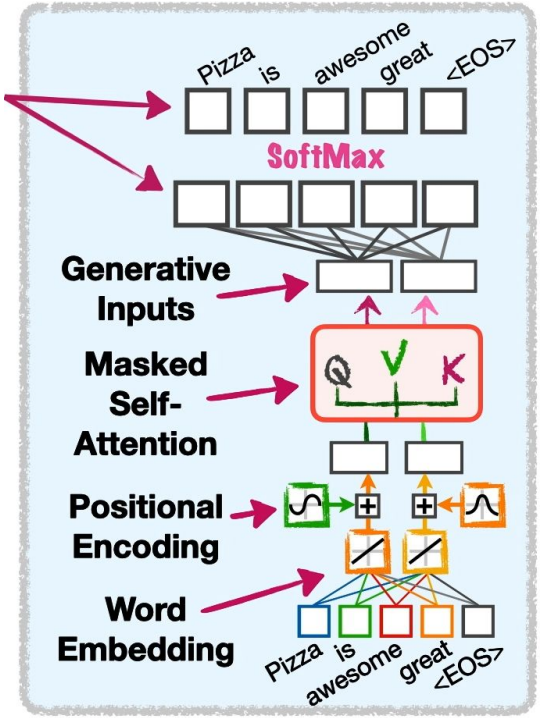
High attention
Low attention


I
love
Previously generated words
Transformer decoder
I
am
a
student
Transformer encoder Output
Masked self-attention
Encoder attention
Feedforward neural network
llamas
Next generated word

2. Transformers

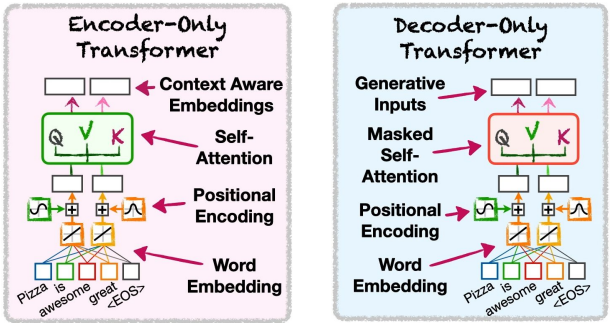
- Ao acoplar rede neural na saída, é possível gerar novos tokens

Geração de tokens na saída



2. Transformers

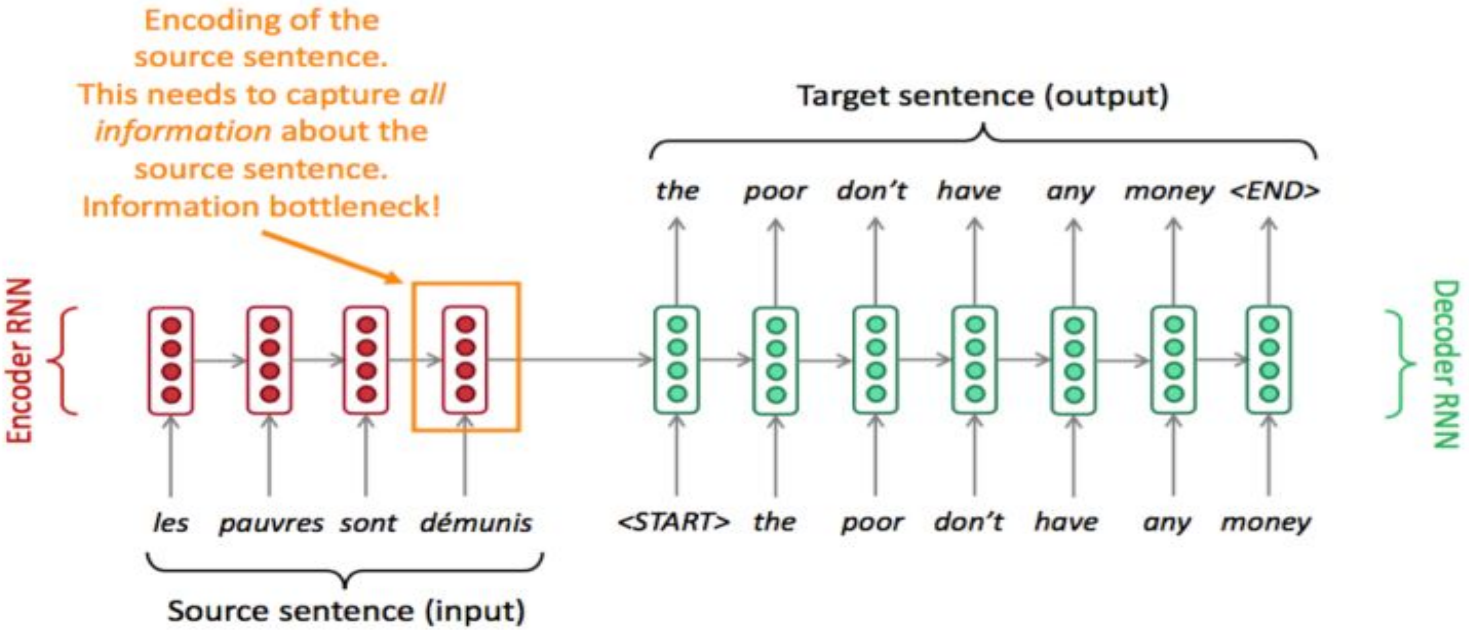
- Mas de onde vem essa definição de **Encoder-Only** e **Decoder-Only**?



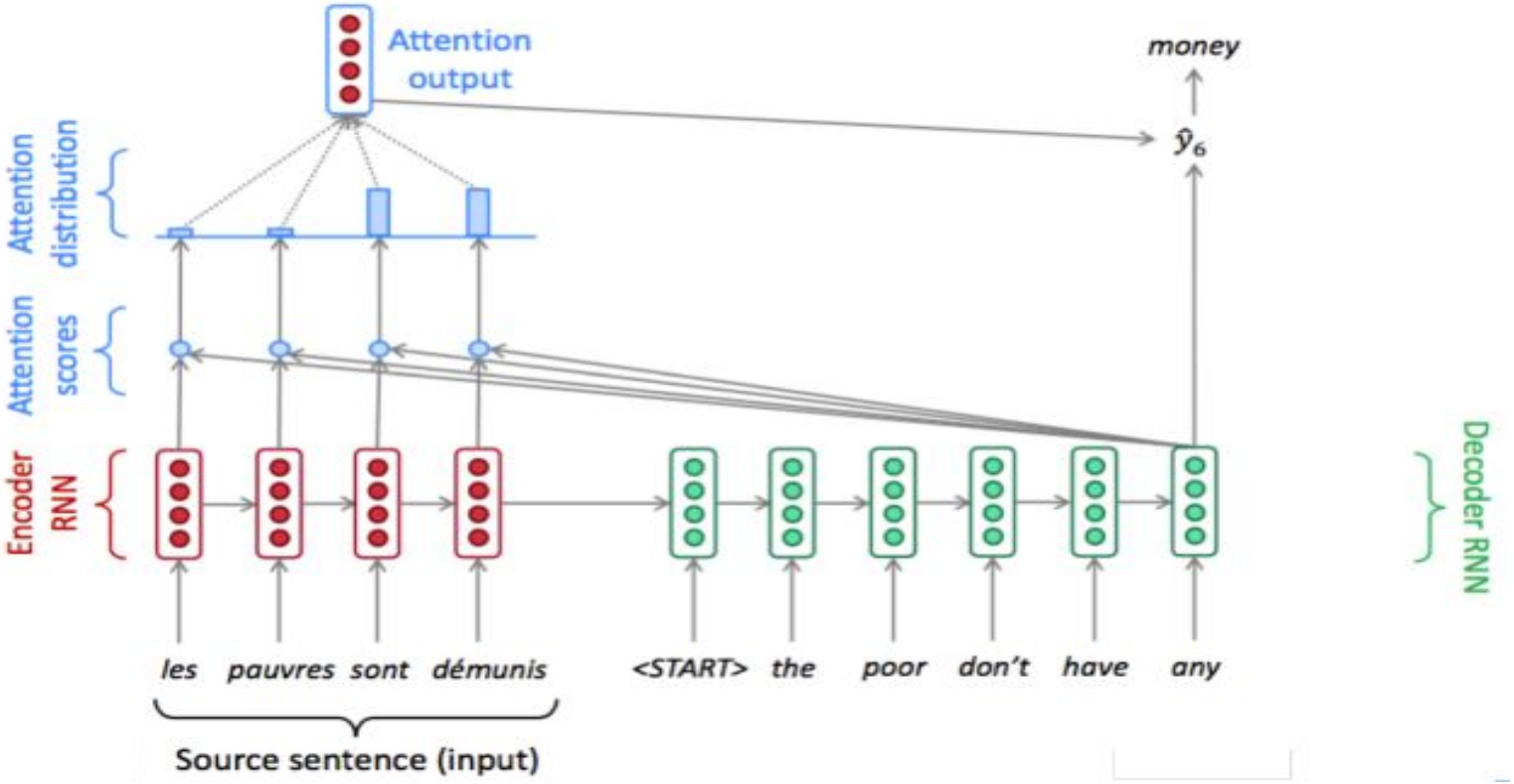
2. Transformers

- A arquitetura original do artigo publicado (Vaswani et al., 2017) foi baseada em **Seq2Seq**, ou arquitetura **Encoder-Decoder**
  - Utilizado para tarefas de tradução
  - Encoder-Decoder Attention

Seq2Seq

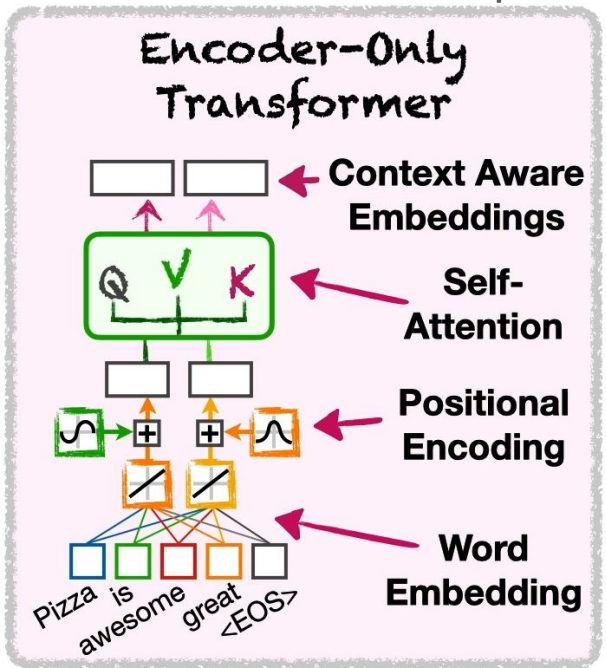

Encoding of the source sentence. This needs to capture *all information* about the source sentence. Information bottleneck!
Target sentence (output)
the poor don't have any money <END>
Encoder RNN
Decoder RNN
les pauvres sont démunis
<START> the poor don't have any money
Source sentence (input)

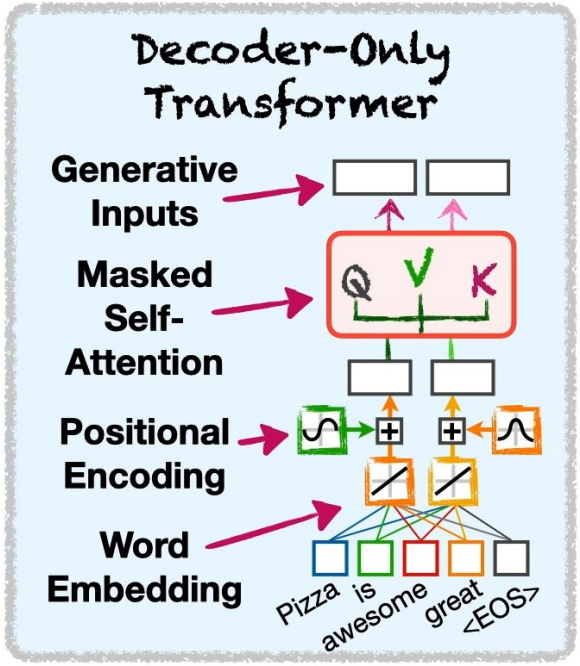
Seq2Seq com atenção



2. Transformers

- A comunidade percebeu que havia aplicações tanto para apenas utilizar o bloco **Encoder**, como também apenas o **Decoder**





Arquitetura Transformers

"We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely." (Vaswani et al., 2017)

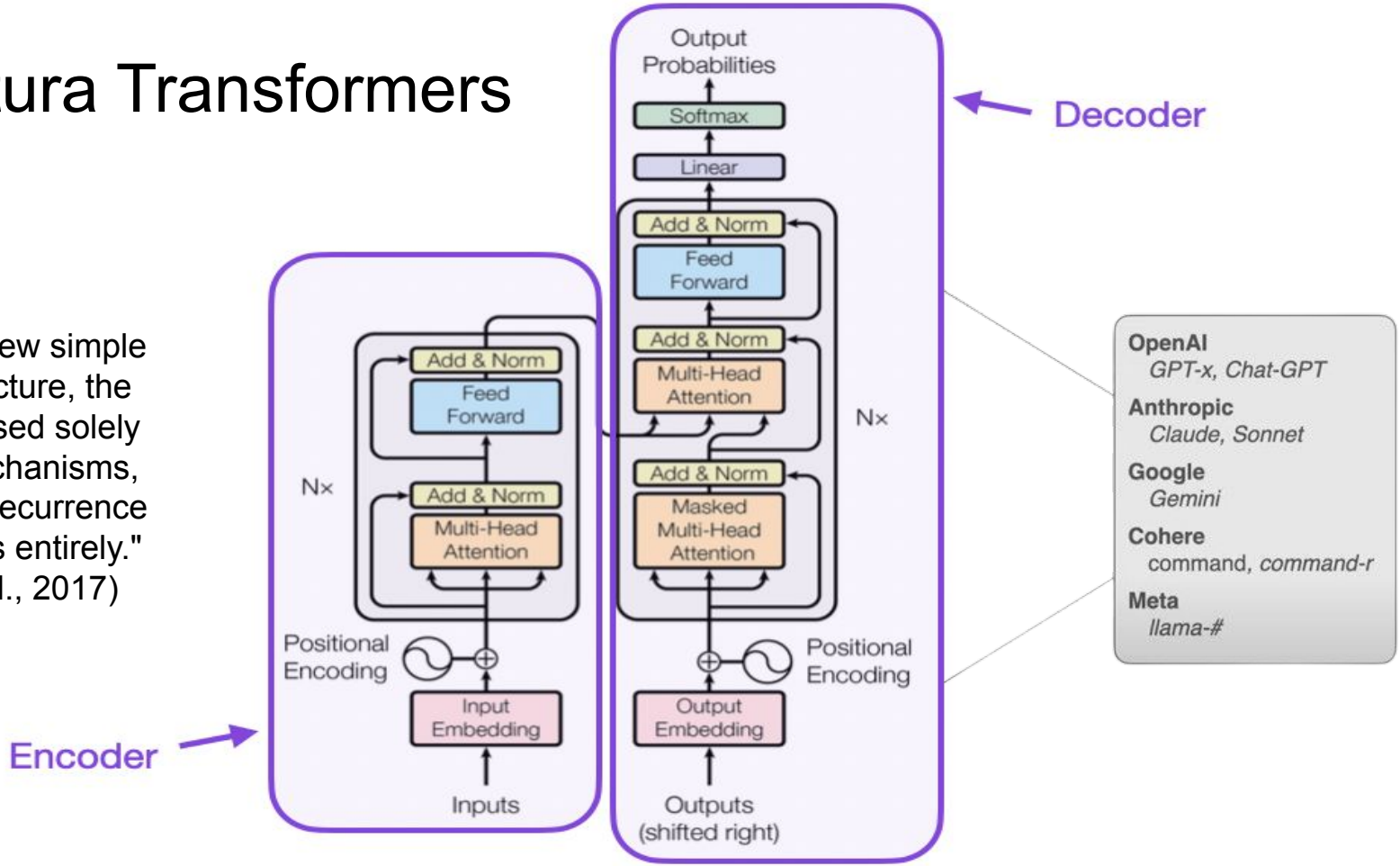
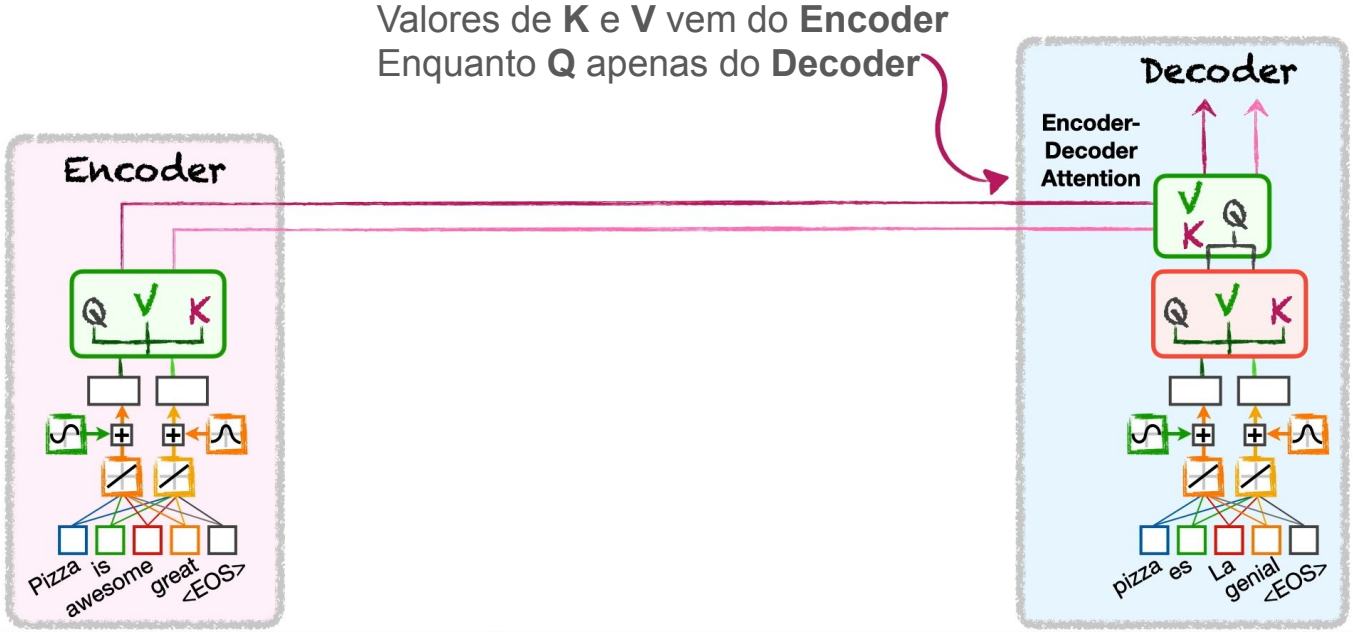


Figure 1: The Transformer - model architecture.

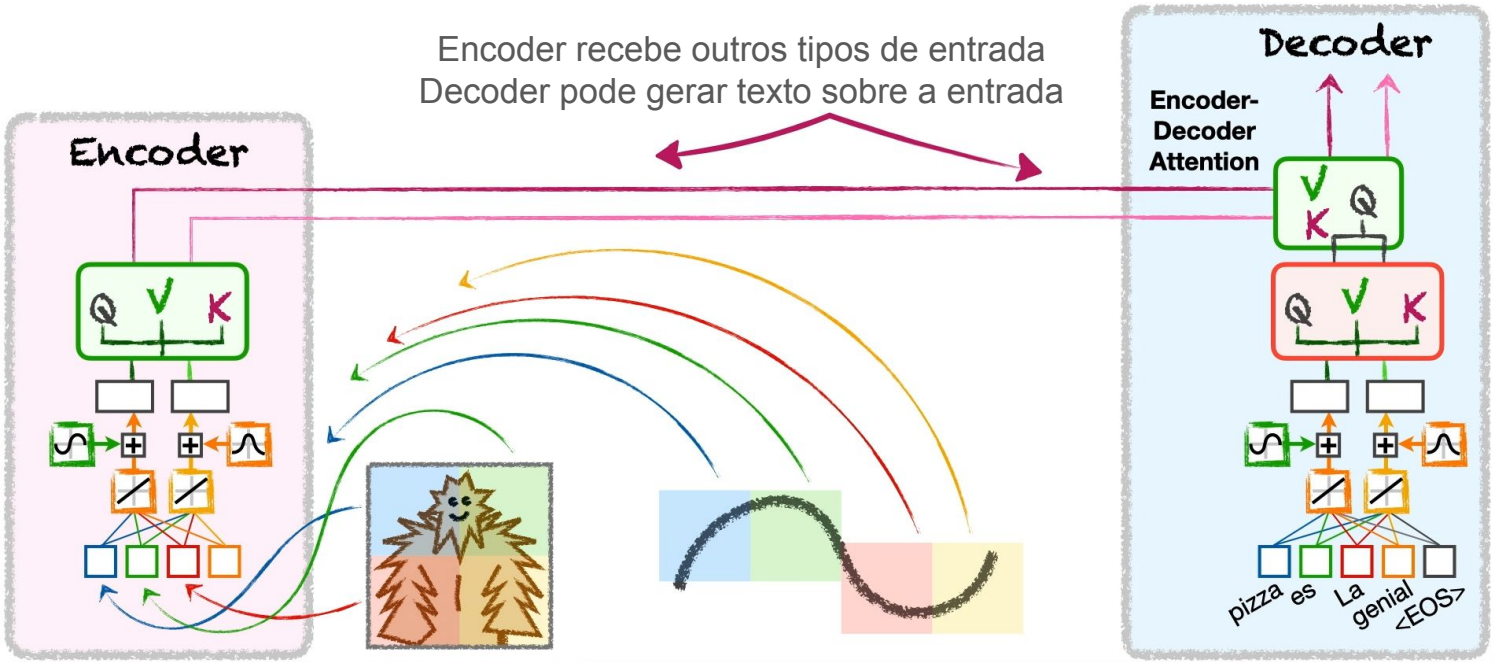
2. Transformers

- Encoder-Decoder Attention
  - Cross Attention



2. Transformers

- Modelos Multi-Modais



Importância dos Transformers

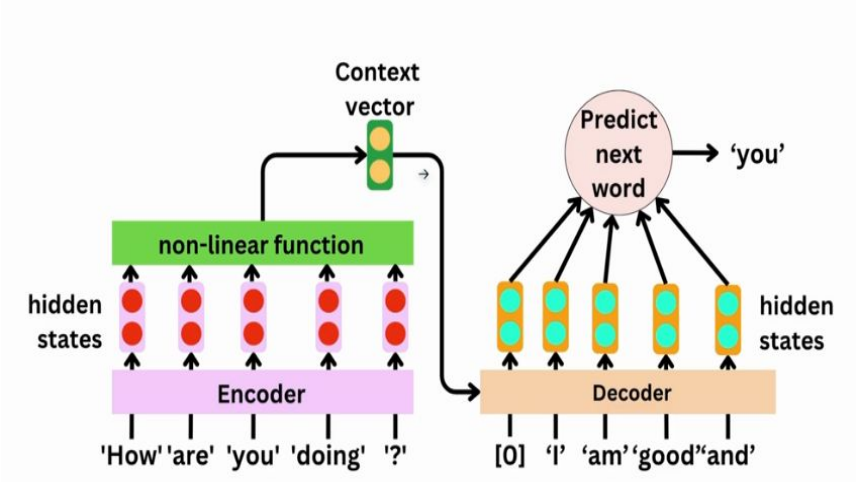
- Transformers surgiram para resolver limitações do seq2seq, dos RNNs e LSTMs em NLP.

- **RNN (Redes Recorrentes)** Processa sequência palavra por palavra.
  Limitações: lento e difícil de treinar em sequências longas (desvanecimento de gradientes).

- **LSTM (Long Short-Term Memory)** Melhora RNN com portas para memorizar informação.
  Limitações: ainda sequencial, pouco paralelismo, treinamento lento.

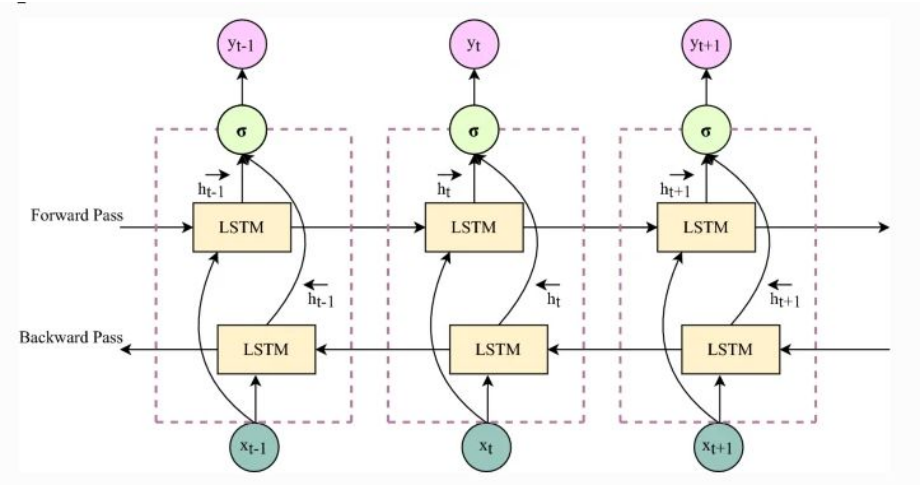
- **Seq2Seq (Sequence-to-Sequence)** Encoder → vetor de contexto → Decoder.
  Limitações: vetor único pode perder informações em frases longas.
  Processamento ainda sequencial, dependência de saída anterior.

RNN



Processa sequência palavra por palavra.

LSTM



Usa portas para memorizar informação

Estrutura dos Transformers

- Transformers em essência: