

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

Lukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

Carolina Dias
Claudio Fortier

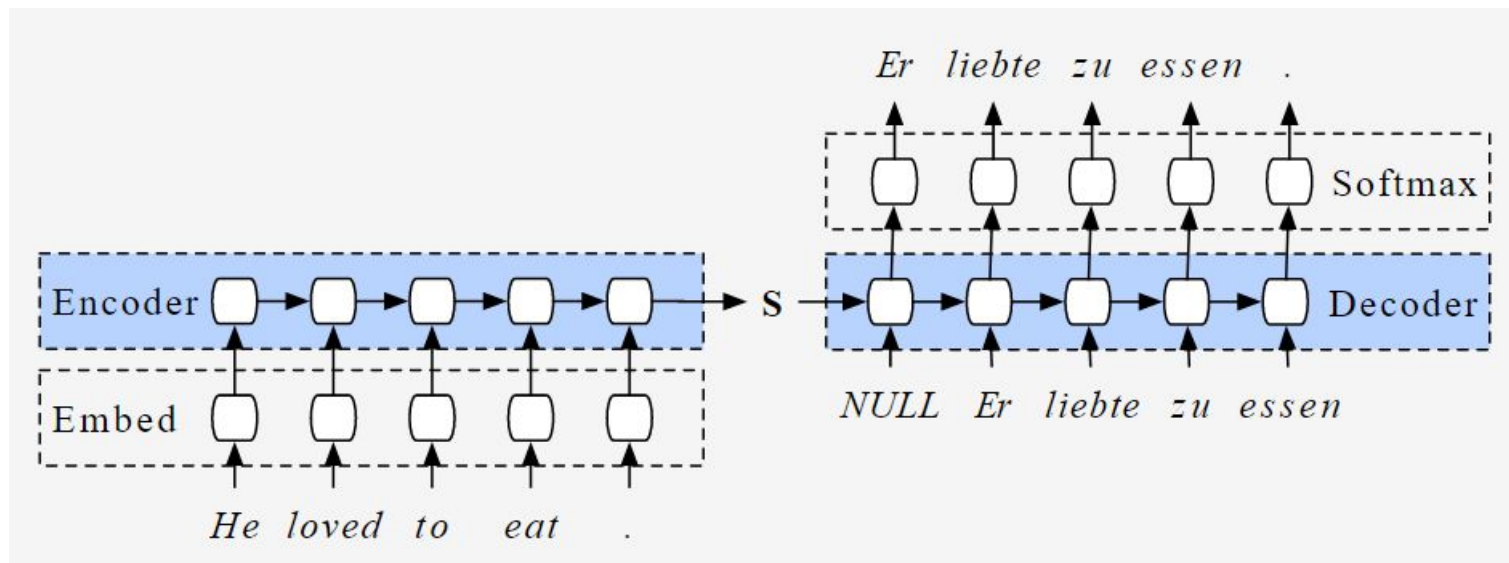
Programa de Pós-Graduação em Ciência da Computação
Universidade Estadual do Ceará

O que veremos hoje...

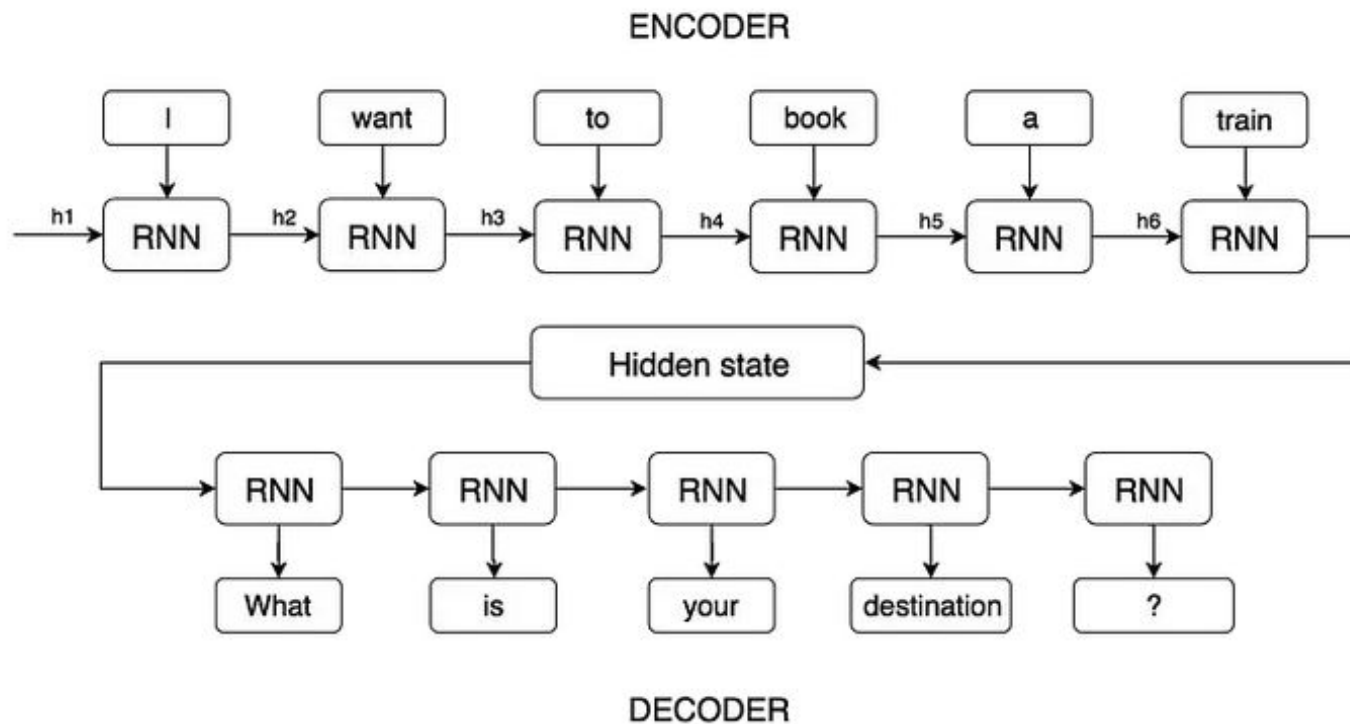
- Porquê Atenção é tudo que você precisa?
- Incorporação de palavras – Word2Vec
- A arquitetura Transformers:
 - Visão Geral
 - Encoders
 - Auto atenção
 - Codificação posicional
 - Conexão Residual
 - Normalização
 - Decoders
 - Fluxo de informações
 - Atenção Encoder / Decoder
 - Função de Perda
 - Retropropagação

Porque atenção é tudo que você precisa?

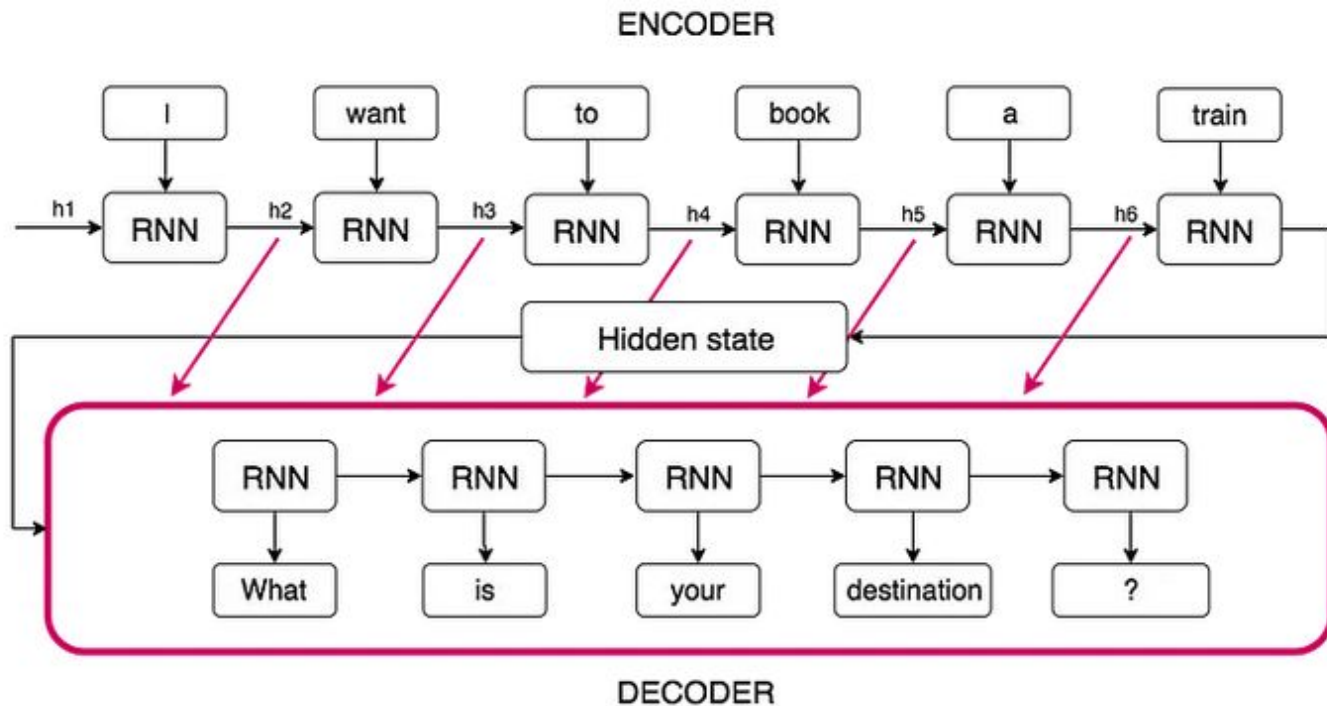
- Modelos **Seq2Seq** já implementavam o conceito de codificadores e decodificadores
- Para traduções, o estado da arte baseava-se em LSTM, GRU e CNN
- Algumas já usavam o conceito de atenção internamente



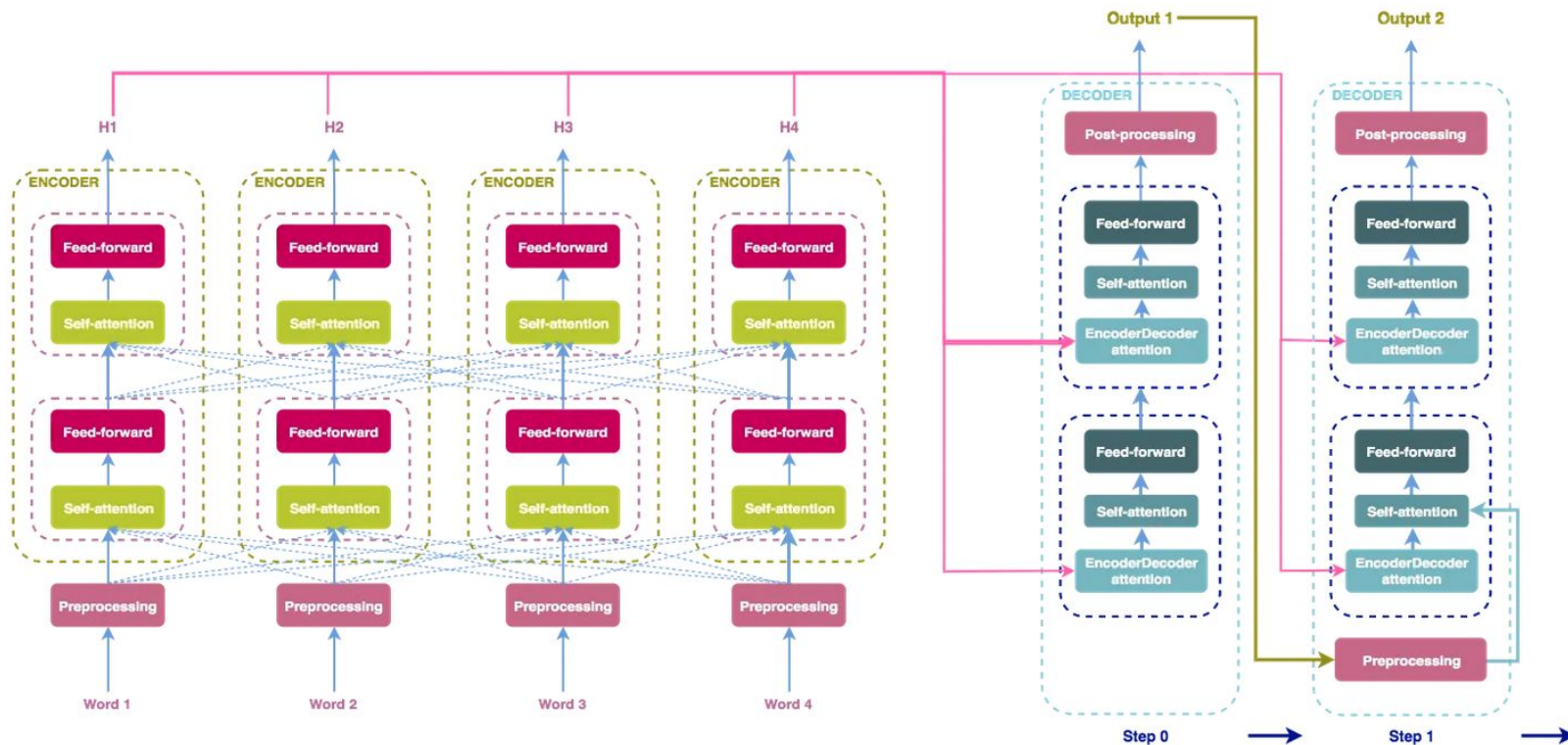
Arquitetura do modelo Sequência a Sequência



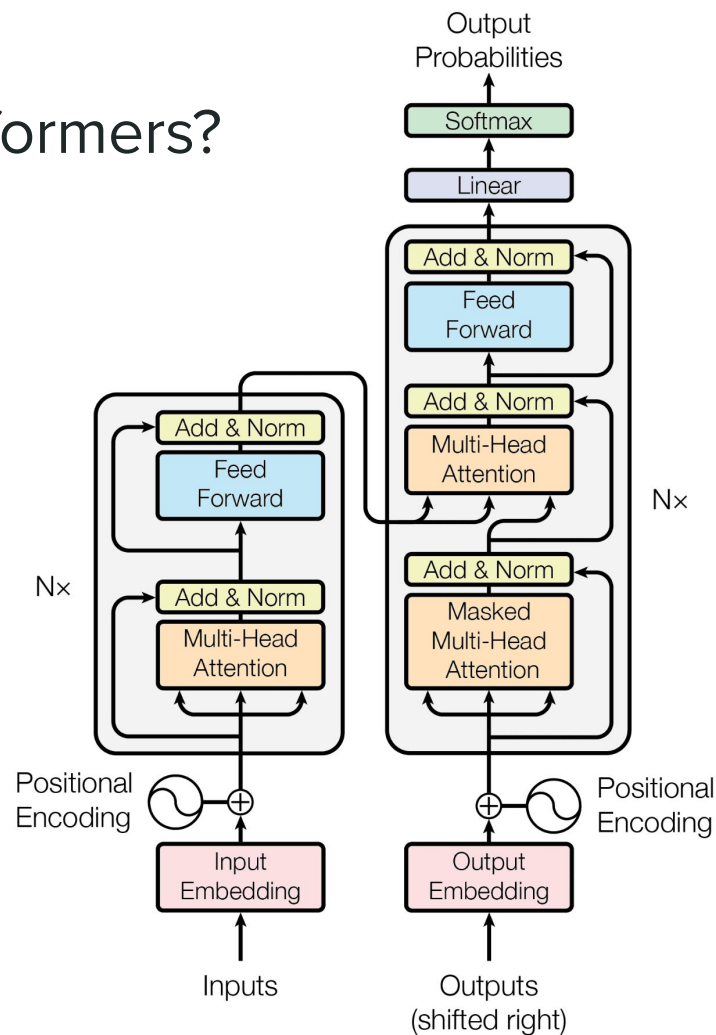
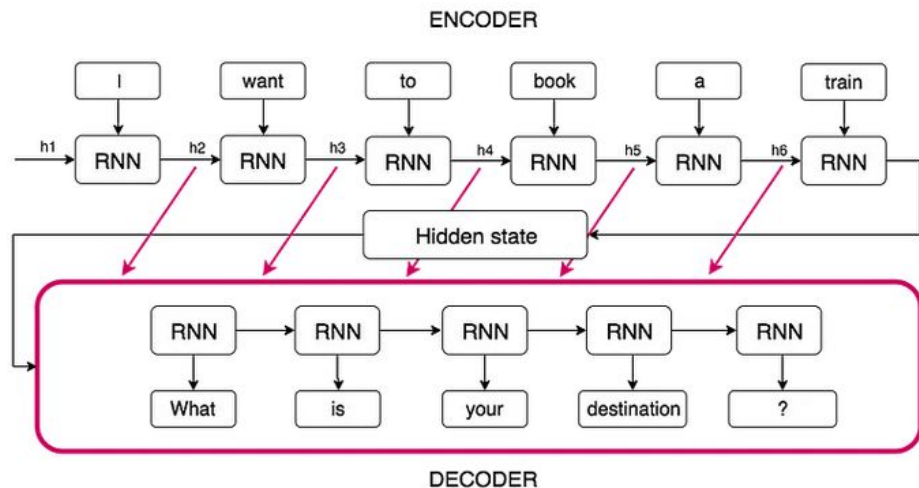
Arquitetura com mecanismo de atenção



Arquitectura Transformer



Você trocaria RNN Seq2Seq por Transformers?



Sim, porquê:

1. **Paralelização:** No Transformer, especialmente no encoder, os tokens são processados em paralelo, o que acelera o treinamento.
2. **Manipulação de Dependências de Longo Alcance:** Os mecanismos de atenção permitem que o Transformer lide eficazmente com dependências de longo alcance entre palavras, o que é crítico para tarefas como tradução.
3. **Escalabilidade:** O modelo escala bem com o tamanho do conjunto de dados e a dimensionalidade dos embeddings, oferecendo desempenho de ponta em grandes conjuntos de dados.
4. **Menor Caminho de Dependência:** O Transformer reduz o número de "hops" ou operações sequenciais necessárias para conectar palavras distantes, o que pode tornar o treinamento mais eficiente e eficaz.

Sim, porquê:

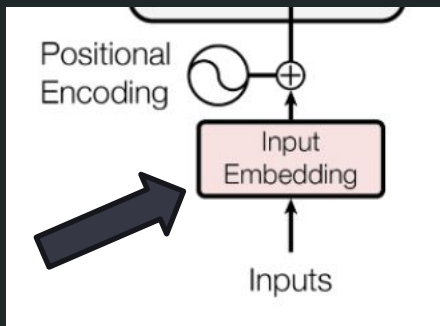
- 5. Generalização para Outras Tarefas:** A arquitetura do Transformer provou ser altamente eficaz não apenas para tradução, mas também para uma variedade de outras tarefas de processamento de linguagem natural, tornando-a mais versátil.
- 6. Treinamento mais Estável:** RNNs, especialmente LSTMs e GRUs, são conhecidos por serem difíceis de treinar eficazmente devido a problemas como desaparecimento e explosão do gradiente, que são menos prevalentes em Transformers.
- 7. Melhor Aprendizado de Representações:** Devido aos mecanismos de atenção e à capacidade de capturar dependências de longo alcance, os Transformers geralmente aprendem representações de tokens mais ricas.

Uso de Transformers

1. **Tradução Automática:** Foi a primeira aplicação para a qual os Transformers foram desenvolvidos.
2. **Processamento de Linguagem Natural (PLN):**
 1. Classificação de Texto
 2. Análise de Sentimento
 3. Resumo Automático
 4. Pergunta e Resposta
 5. Reconhecimento de Entidades Nomeadas
3. **Geração de Texto:** Utilizados em chatbots, escrita assistida e outras tarefas de geração de linguagem.
4. **Análise de Série Temporal:** Embora não seja uma aplicação clássica, os Transformers têm mostrado eficácia nesse campo.
5. **Visão Computacional:** Modelos como o ViT (Vision Transformer) adaptam o Transformer para tarefas de classificação e segmentação de imagem.

Uso de Transformers

6. **Reinforcement Learning:** Em combinação com técnicas de aprendizado por reforço, para tarefas como jogos e navegação autônoma.
7. **Biologia Computacional:** Utilizado em problemas como predição da estrutura de proteínas.
8. **Música e Áudio:** Para geração de música ou classificação de sinais de áudio.
9. **Interpretação de Código e Programação Assistida:** Ajuda no autocomplete de código e detecção de bugs.
10. **Multimodalidade:** Combinação de texto, imagem e outros tipos de dados para tarefas mais complexas.



cat =>

1.2	-0.1	4.3	3.2
-----	------	-----	-----

mat =>

0.4	2.5	-0.9	0.5
-----	-----	------	-----

on =>

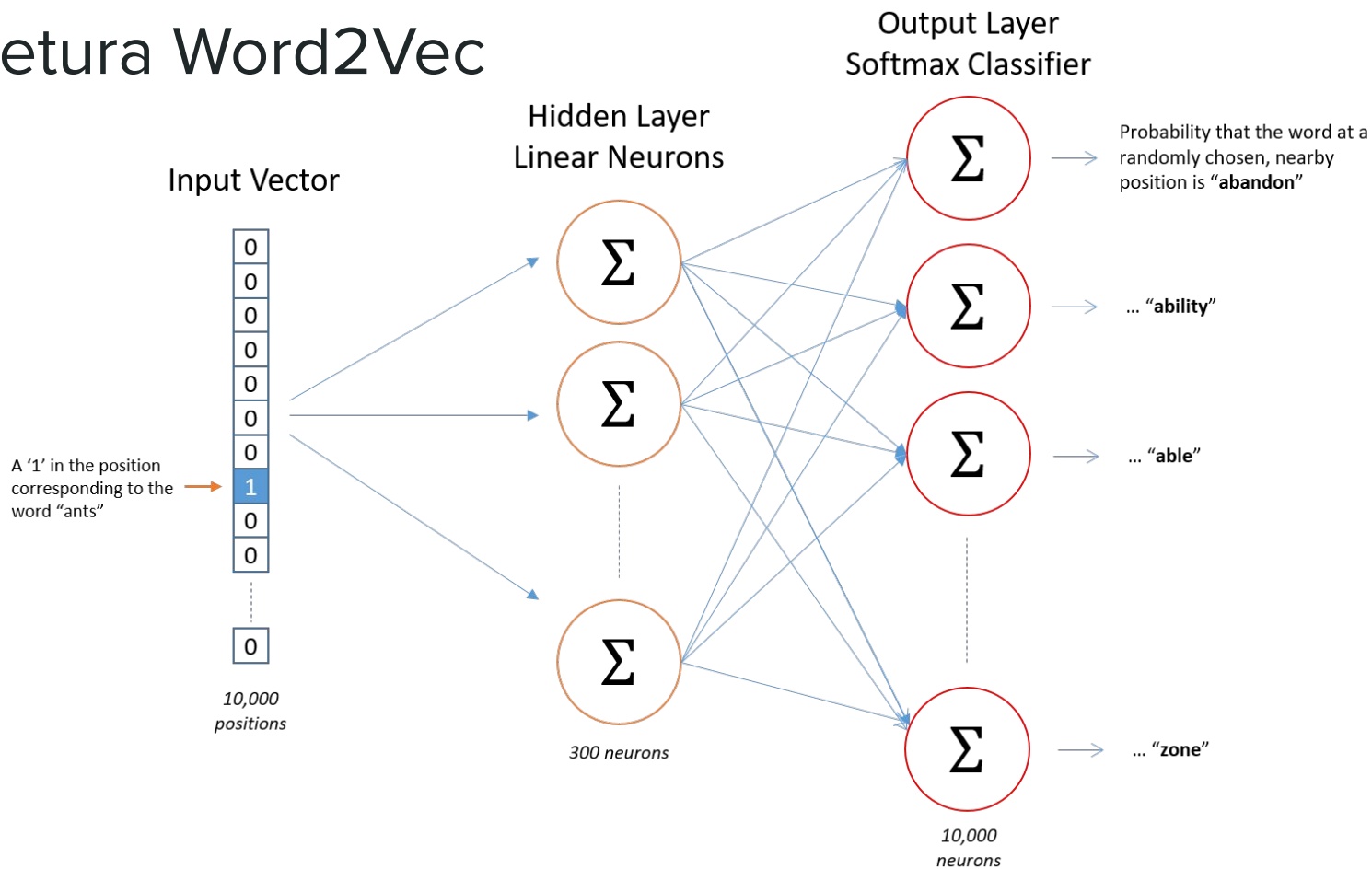
2.1	0.3	0.1	0.4
-----	-----	-----	-----

Incorporação de Palavras – Word2Vec

Não é obrigatório mas...

1. **Inicialização Eficiente:** Iniciar com embeddings pré-treinados pode acelerar a convergência durante o treinamento do Transformer.
2. **Representações Ricas:** Word2Vec captura semântica e relações sintáticas que podem enriquecer as representações usadas pelo Transformer.
3. **Uso de Conhecimento Externo:** Embeddings pré-treinados são geralmente treinados em grandes corpus, capturando informações que podem não estar presentes no conjunto de treinamento específico do Transformer.
4. **Regularização:** Utilizar embeddings pré-treinados pode agir como uma forma de regularização, potencialmente melhorando a generalização do modelo para dados não vistos.
5. **Economia Computacional:** Utilizar embeddings pré-treinados pode economizar recursos computacionais.

A arquitetura Word2Vec



Word2Vec, Skip-Gram

Source Text

Training
Samples

The quick brown fox jumps over the lazy dog. →

(the, quick)
(the, brown)

The quick brown fox jumps over the lazy dog. →

(quick, the)
(quick, brown)
(quick, fox)

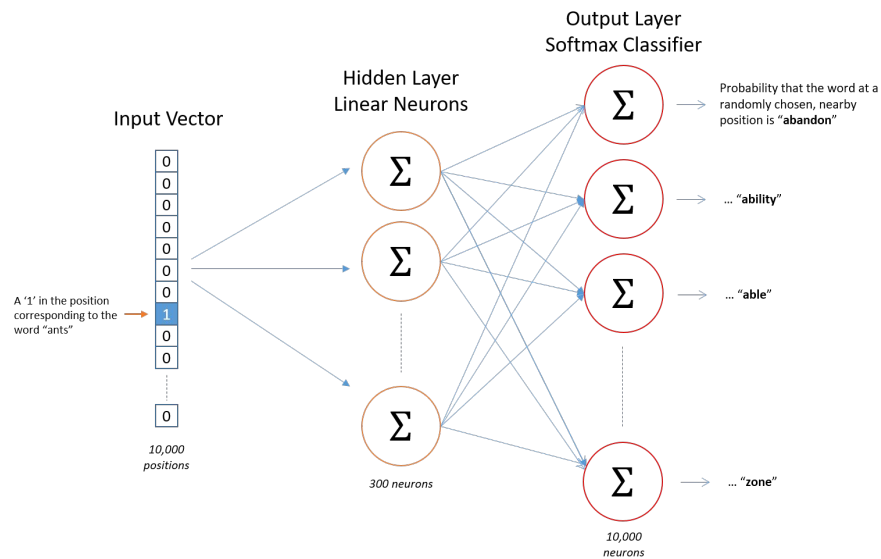
The quick brown fox jumps over the lazy dog. →

(brown, the)
(brown, quick)
(brown, fox)
(brown, jumps)

The quick brown fox jumps over the lazy dog. →

(fox, quick)
(fox, brown)
(fox, jumps)
(fox, over)

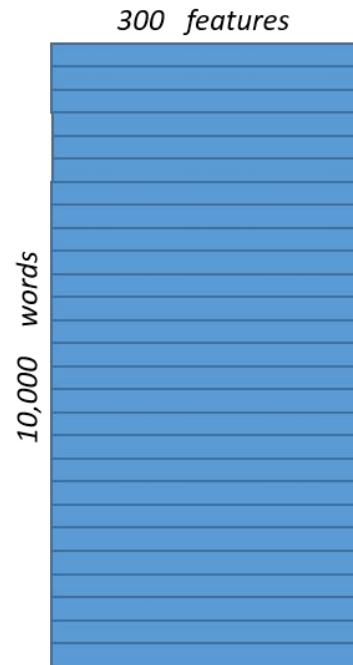
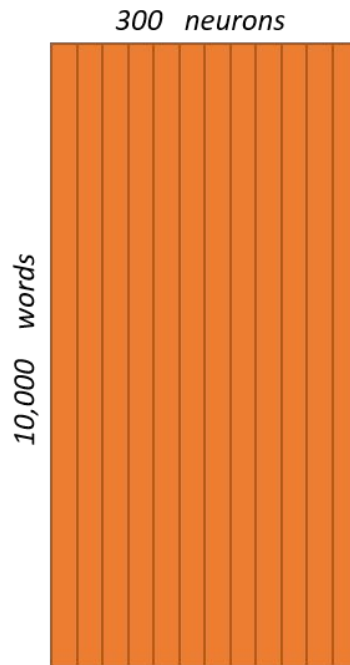
Word2Vec, “O pulo do gato!”



Hidden Layer
Weight Matrix

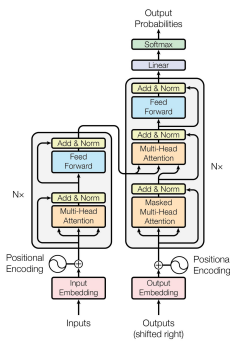


Word Vector
Lookup Table!

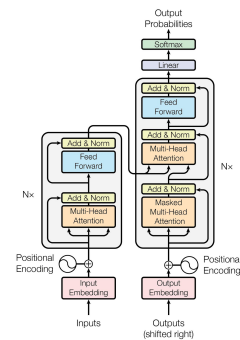
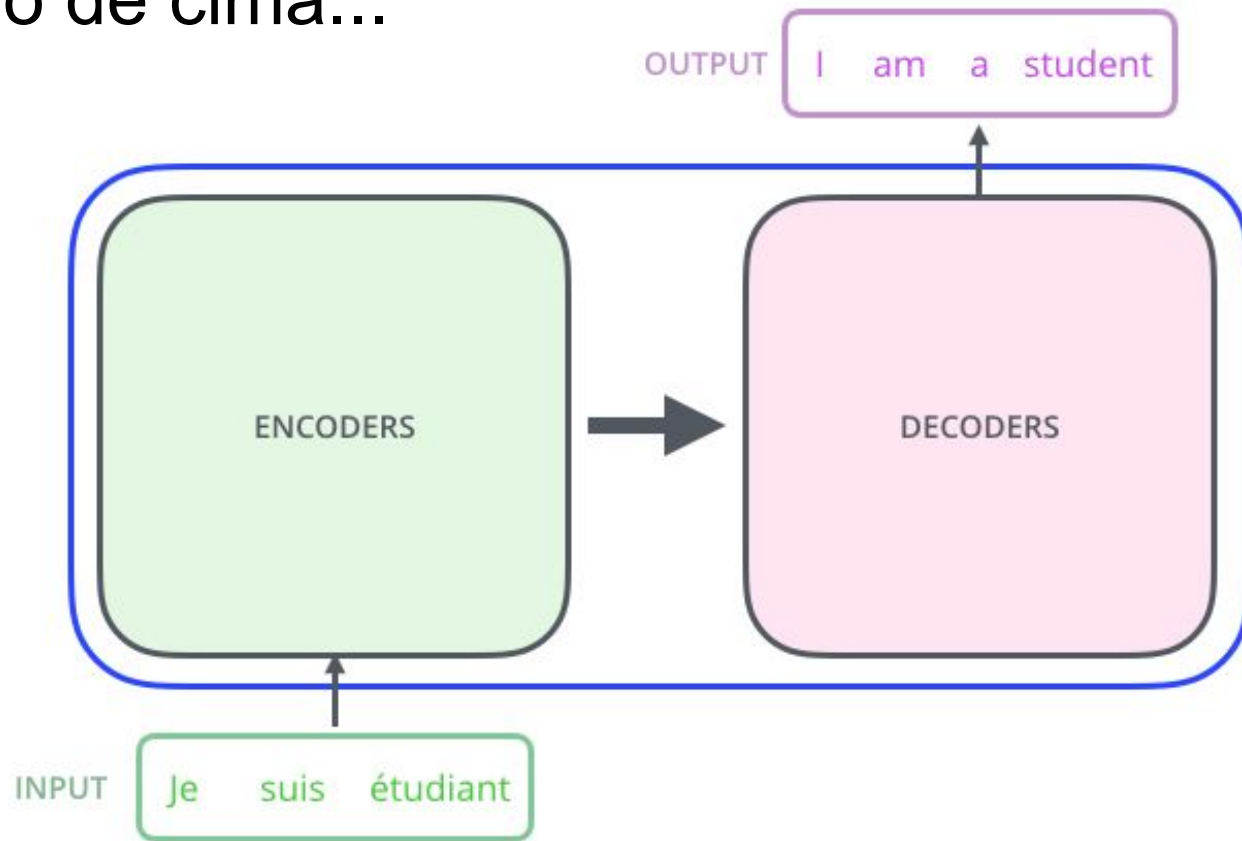


Arquitetura Transformer

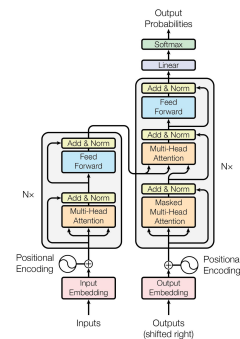
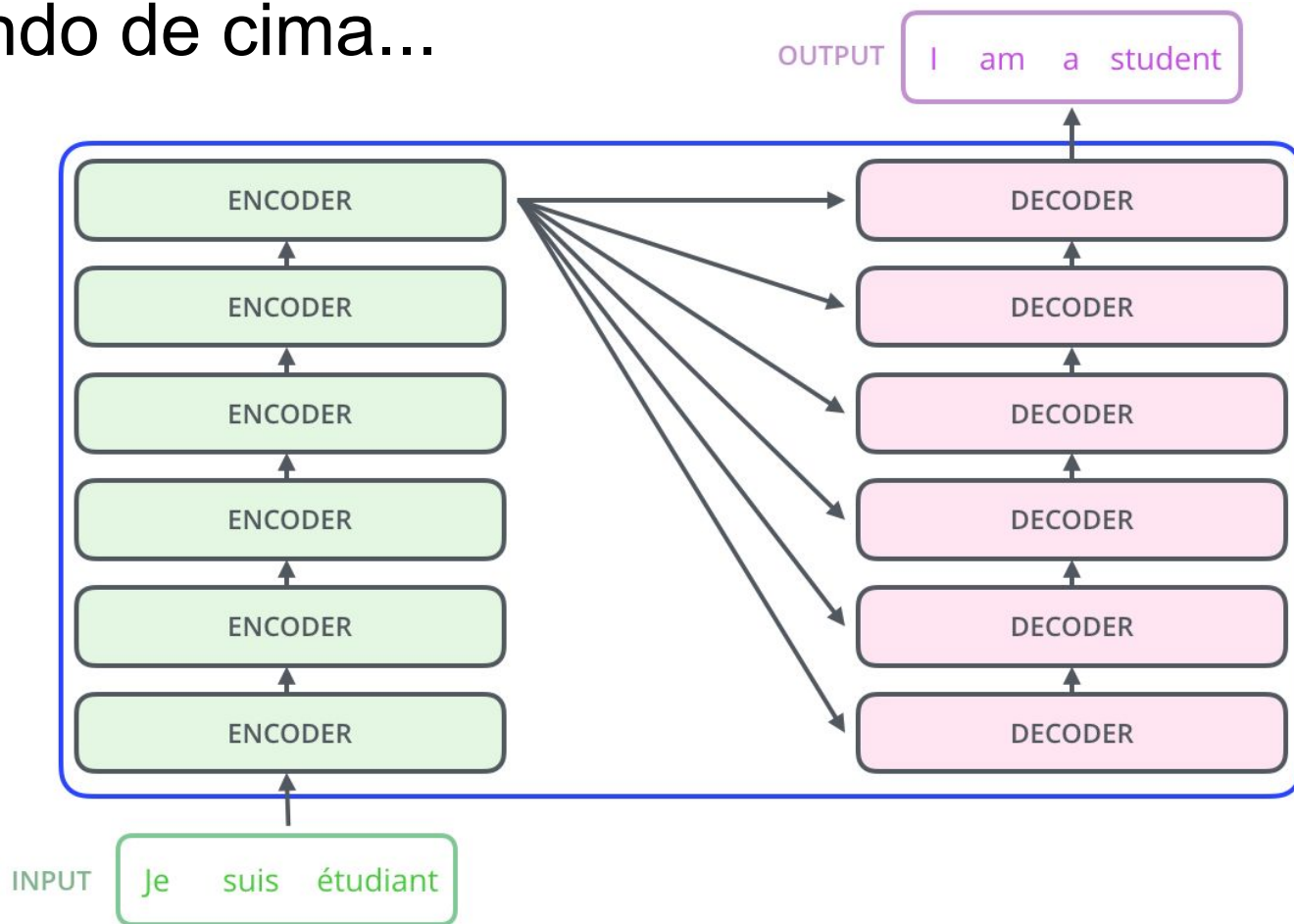
Olhando de cima...



Olhando de cima...

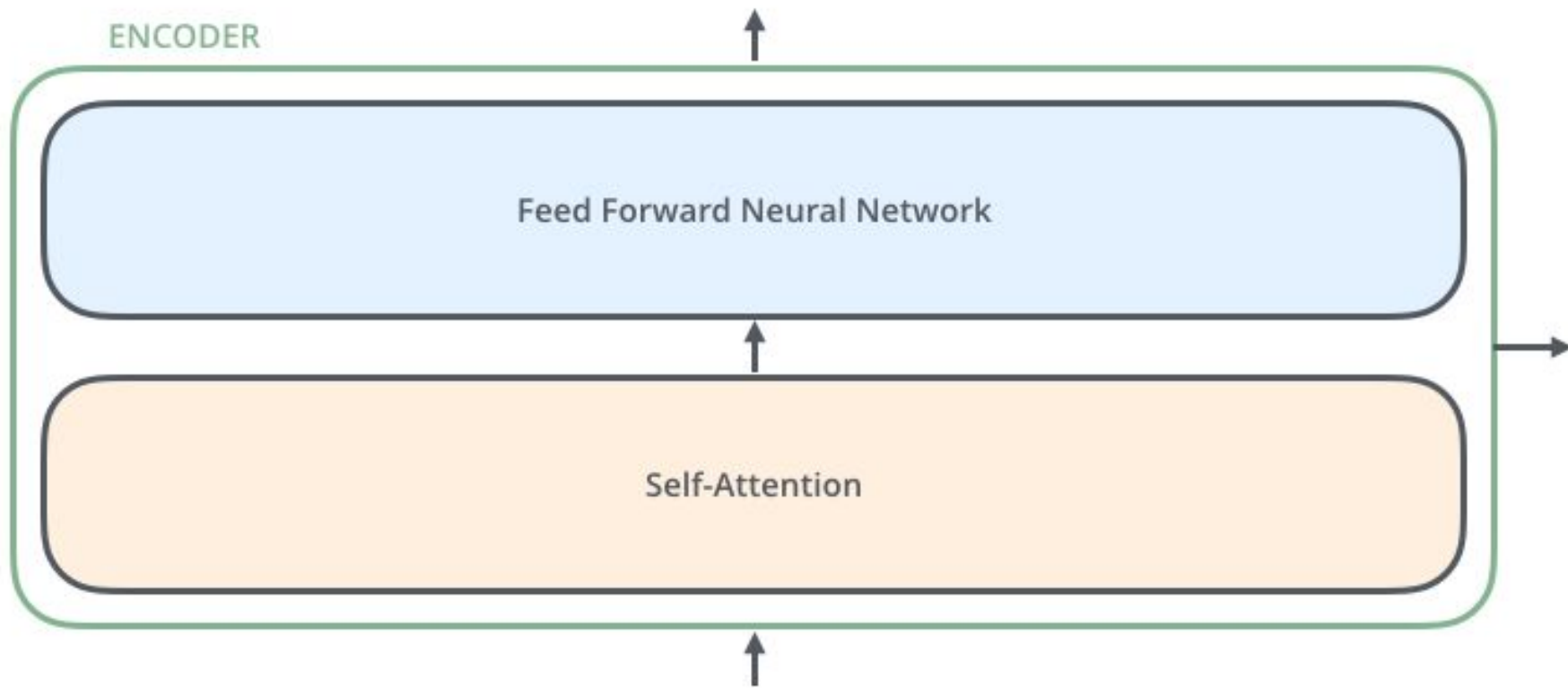


Olhando de cima...

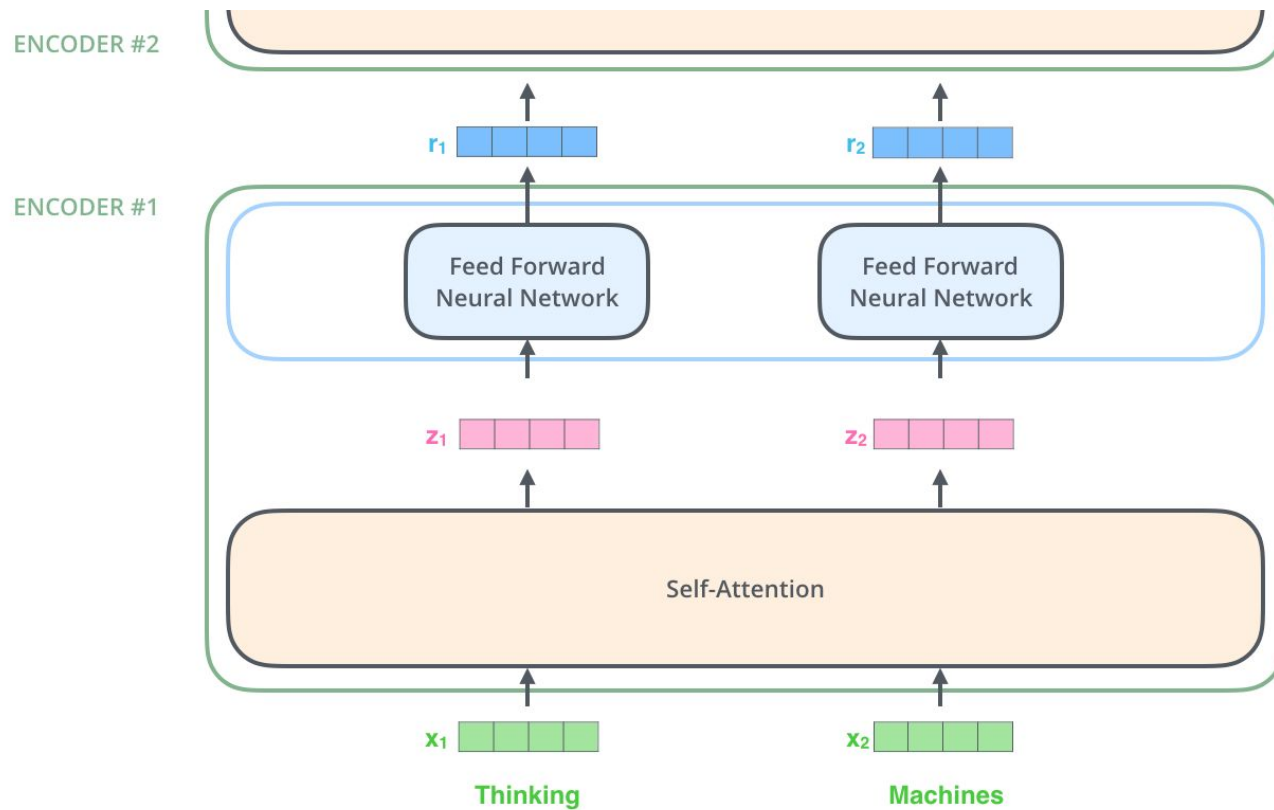


Encoders

Olhando de cima...



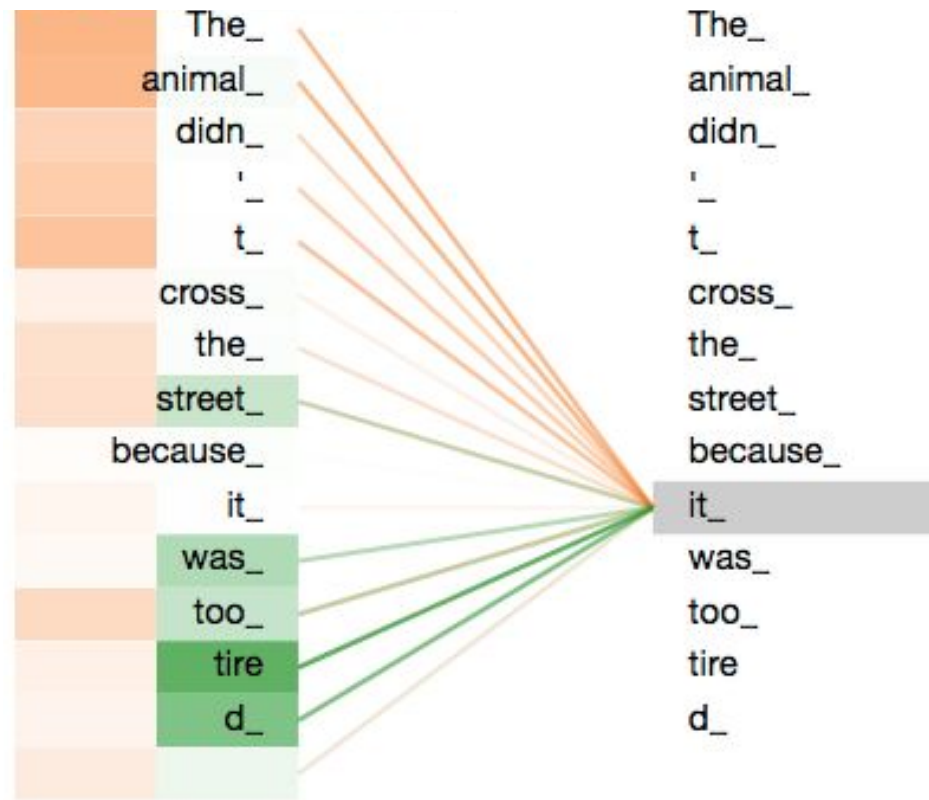
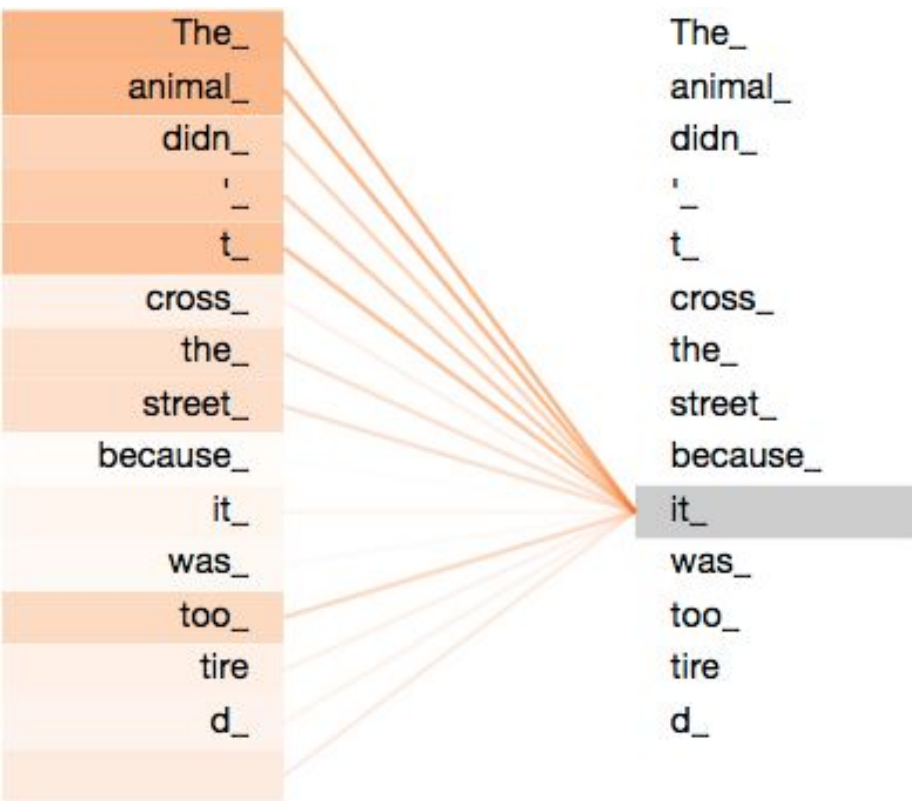
Olhando de cima...



Autoatenção

Intuição

Na frase "The animal didn't cross the street because it was too tired" a quem "it" se refere?



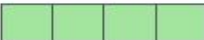
Passo 1 – Vetores de Entrada

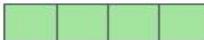
Input

Thinking

Machines


Embedding

x_1 

x_2 

Queries

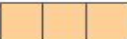
q_1 


q_2 

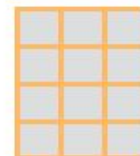


W^Q

Keys


k_1 


k_2 



W^K

Values

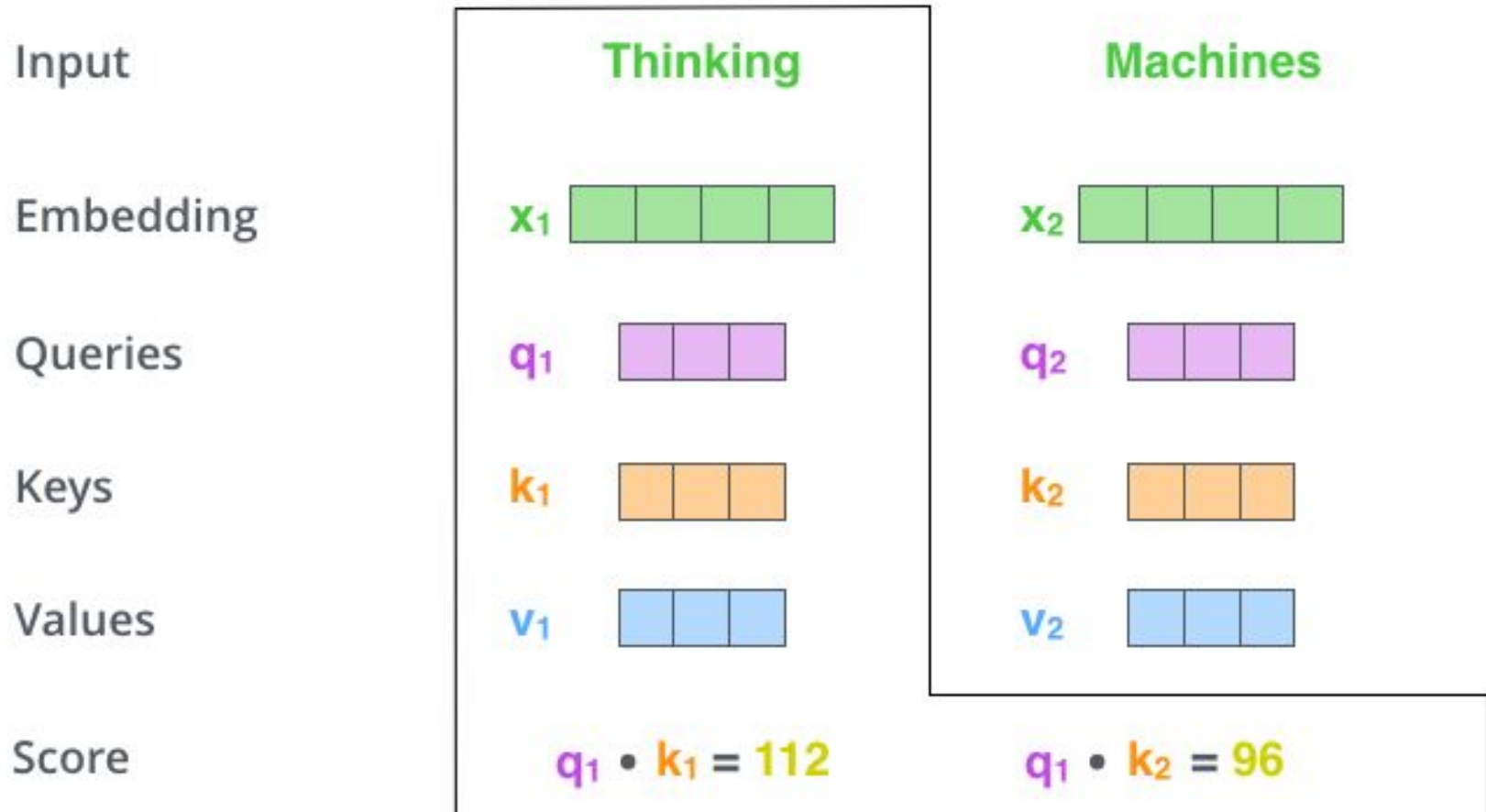
v_1 

v_2 

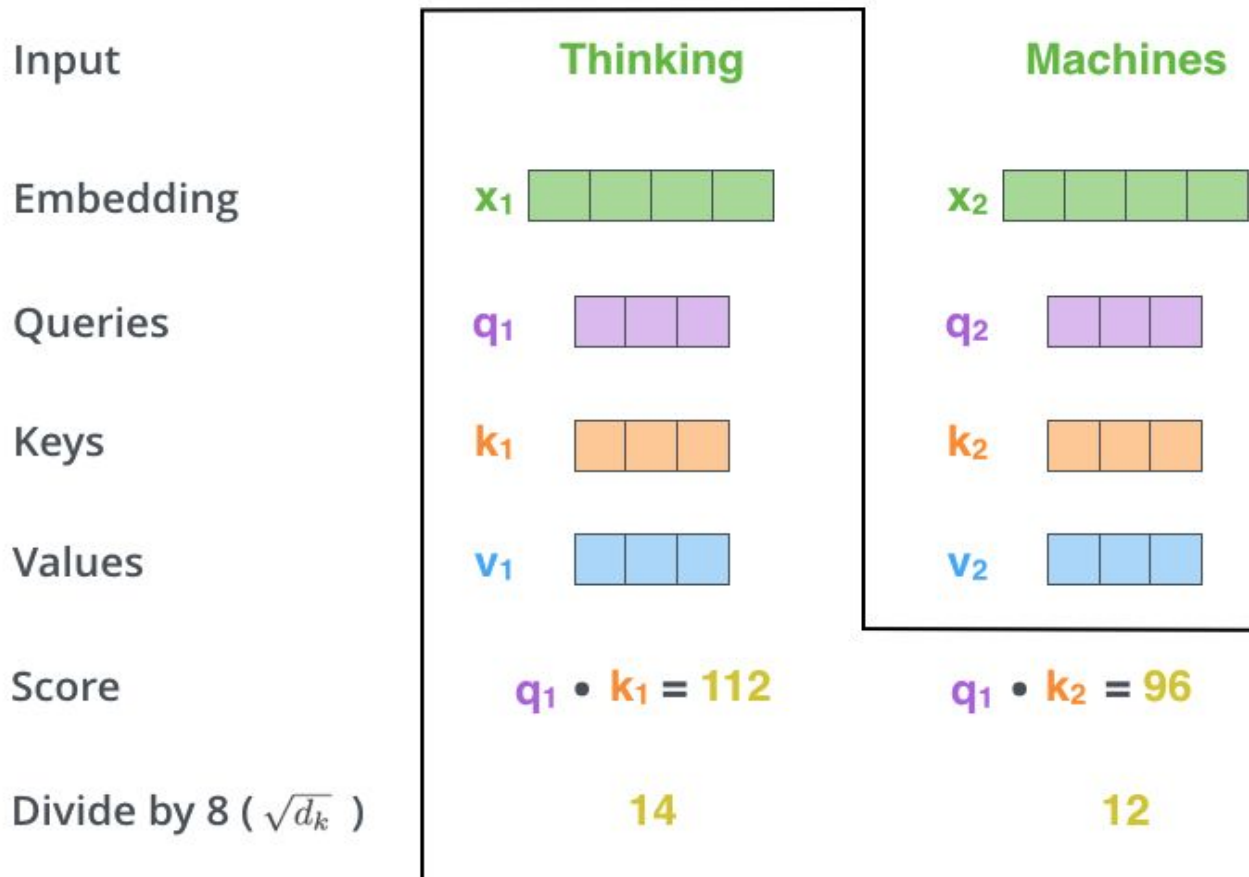


W^V

Passo 2 – Cálculo do Score



Passo 3 – Divisão por $\sqrt{d_k}$



Passo 4 – Softmax

Input

Embedding

Queries

Keys

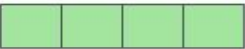
Values

Score

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

Softmax

Thinking

x_1 

q_1 

k_1 

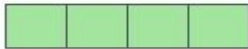
v_1 

$$q_1 \cdot k_1 = 112$$

14

0.88

Machines

x_2 

q_2 

k_2 

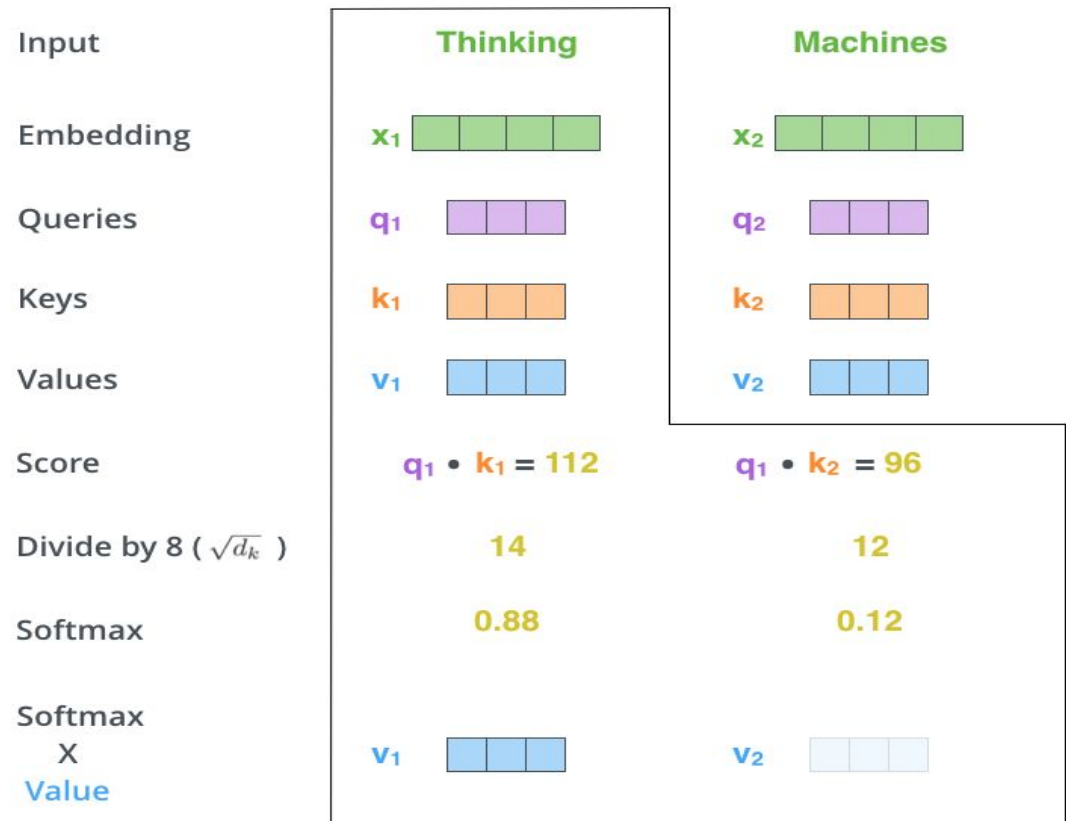
v_2 

$$q_1 \cdot k_2 = 96$$

12

0.12

Passo 5 – Softmax X Value



Passo 5 – Sum(Vn)

Input

Embedding

Queries

Keys

Values

Score

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

Softmax

Softmax
X
Value

Sum

Thinking

x₁ 

q₁ 

k₁ 

v₁ 

q₁ • k₁ = 112

14

0.88

v₁ 

z₁ 

Machines

x₂ 

q₂ 

k₂ 

v₂ 

q₂ • k₂ = 96


12


0.12

v₂ 

z₂ 

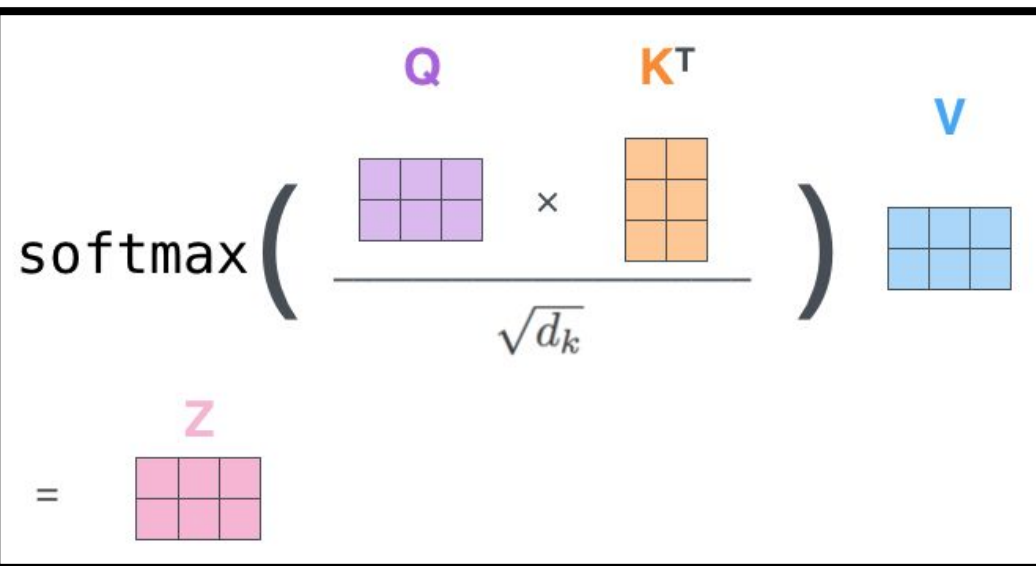
Cálculo Matricial

$$X \times W^Q = Q$$


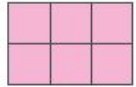
$$X \times W^K = K$$


$$X \times W^V = V$$

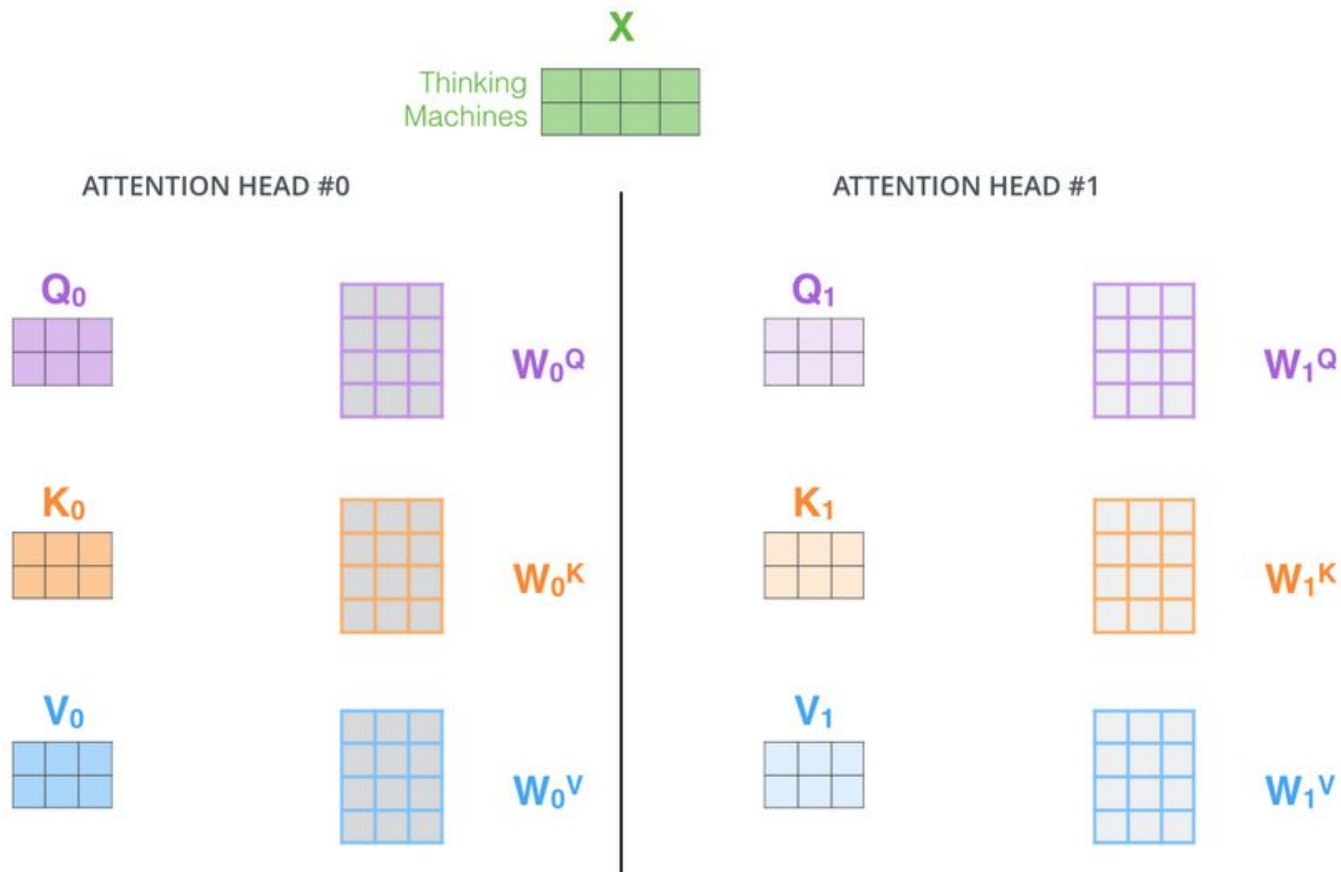

$$A(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

$$\text{softmax} \left(\frac{Q \times K^T}{\sqrt{d_k}} \right) V$$


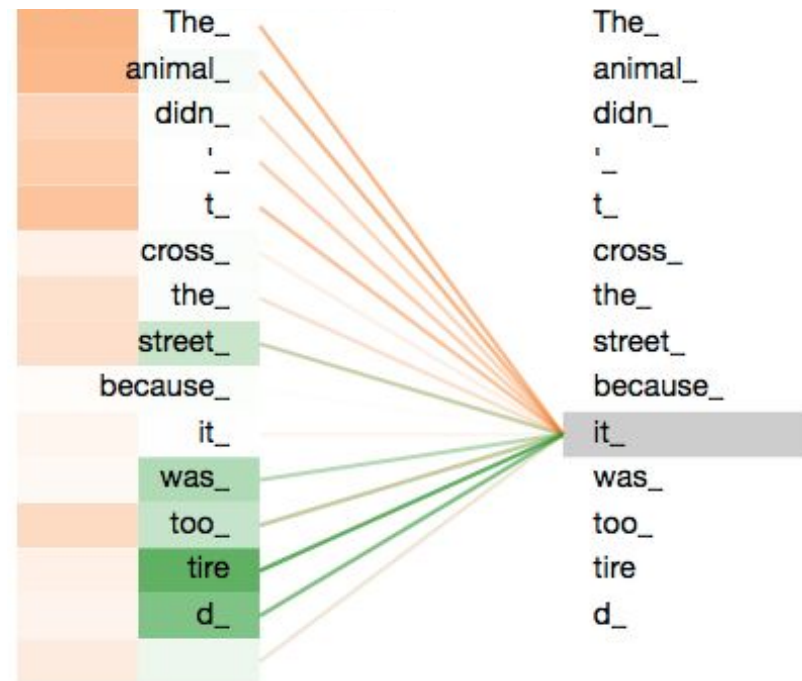
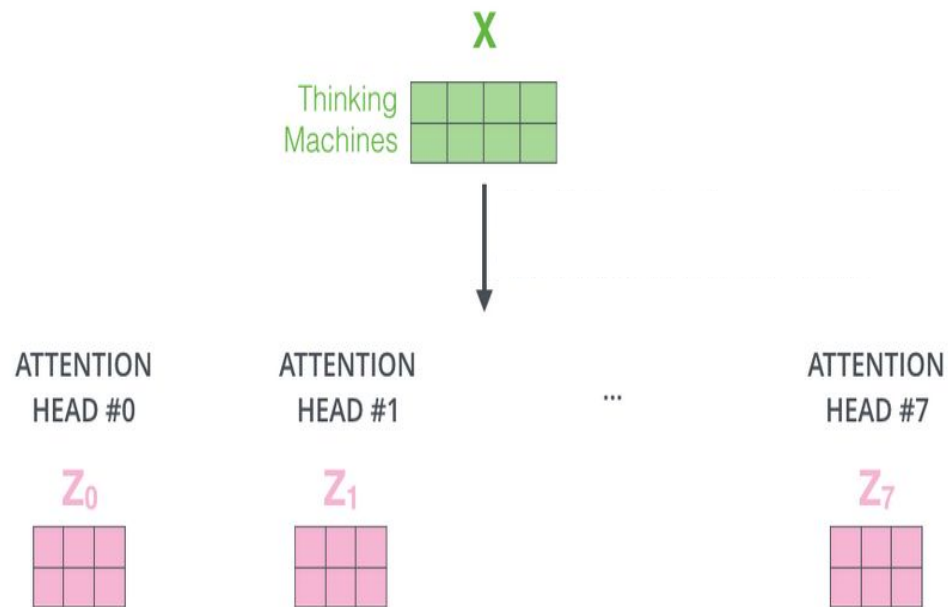
=



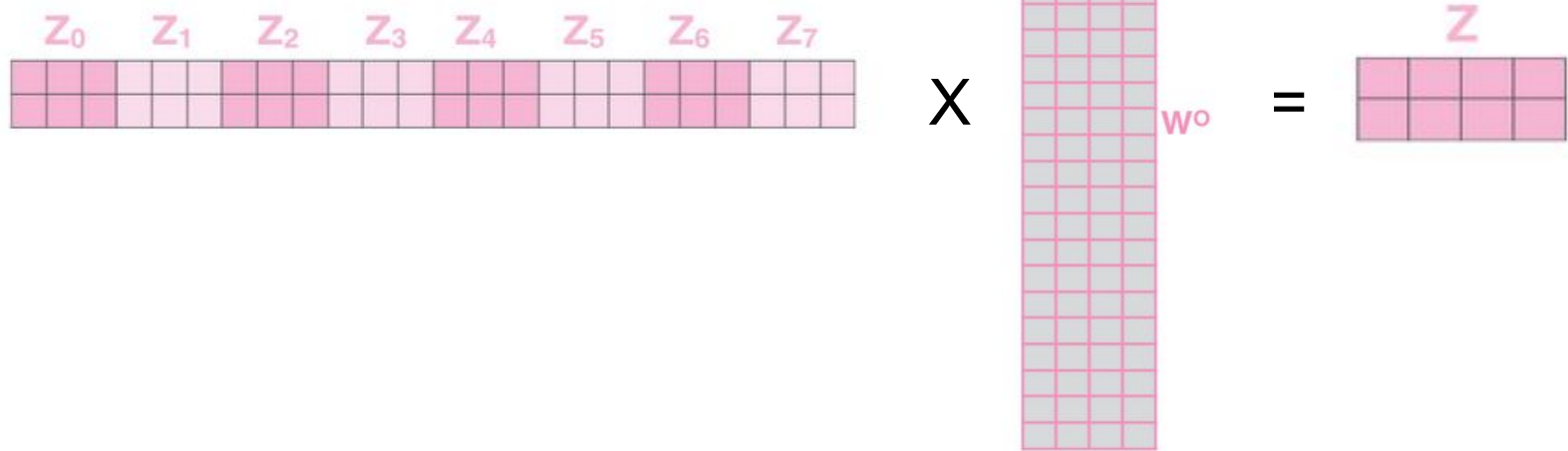
Atenção Multi-cabeças



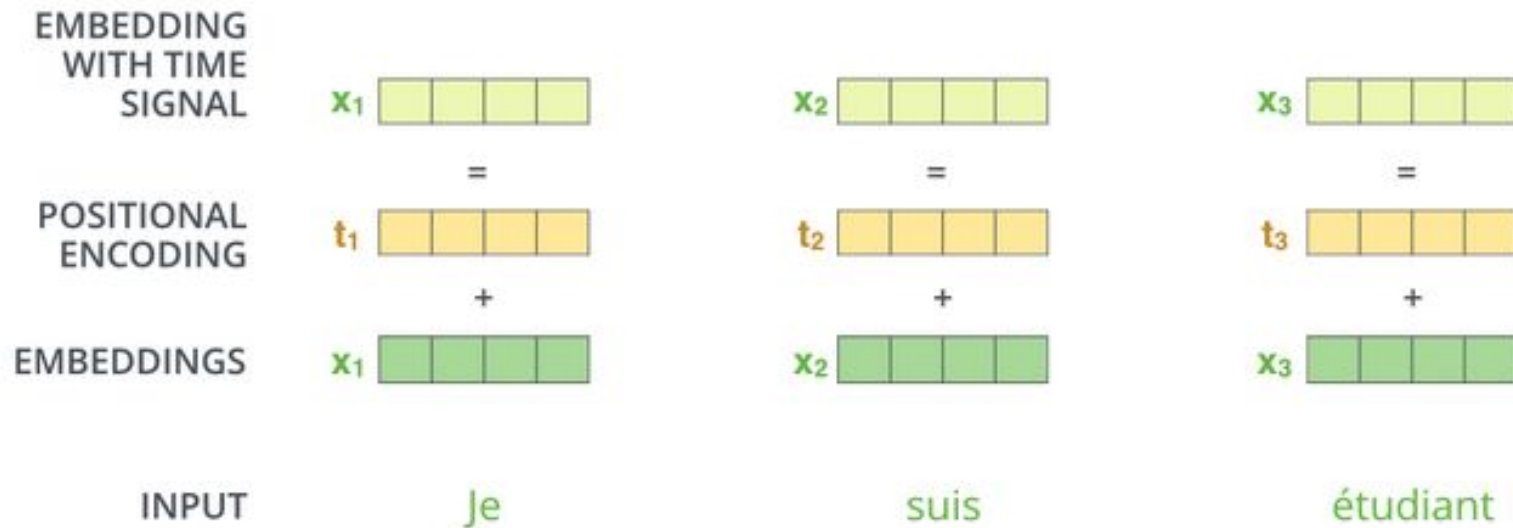
Atenção Multi-cabeças



Atenção Multi-cabeças



Codificação Posicional



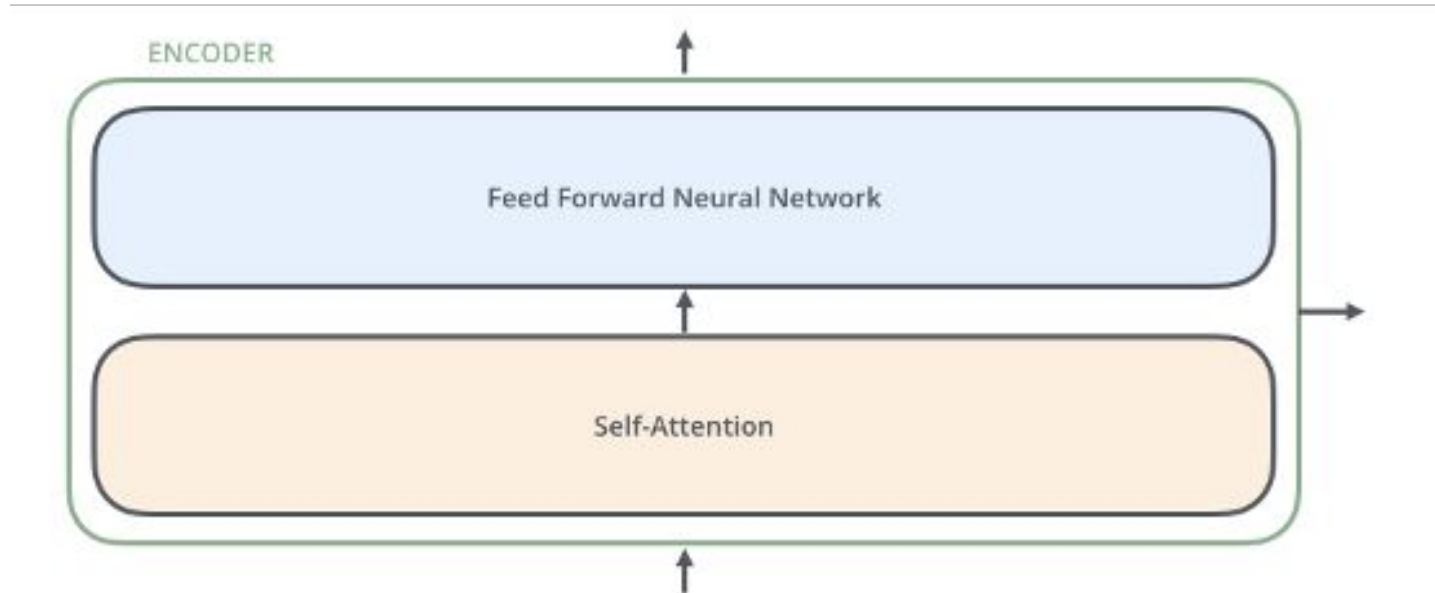
Codificação Posicional



$$PE(p, 2i) = \sin\left(\frac{p}{10000^{2i/d}}\right)$$
$$PE(p, 2i + 1) = \cos\left(\frac{p}{10000^{2i/d}}\right)$$

Feed-forward

$$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2$$

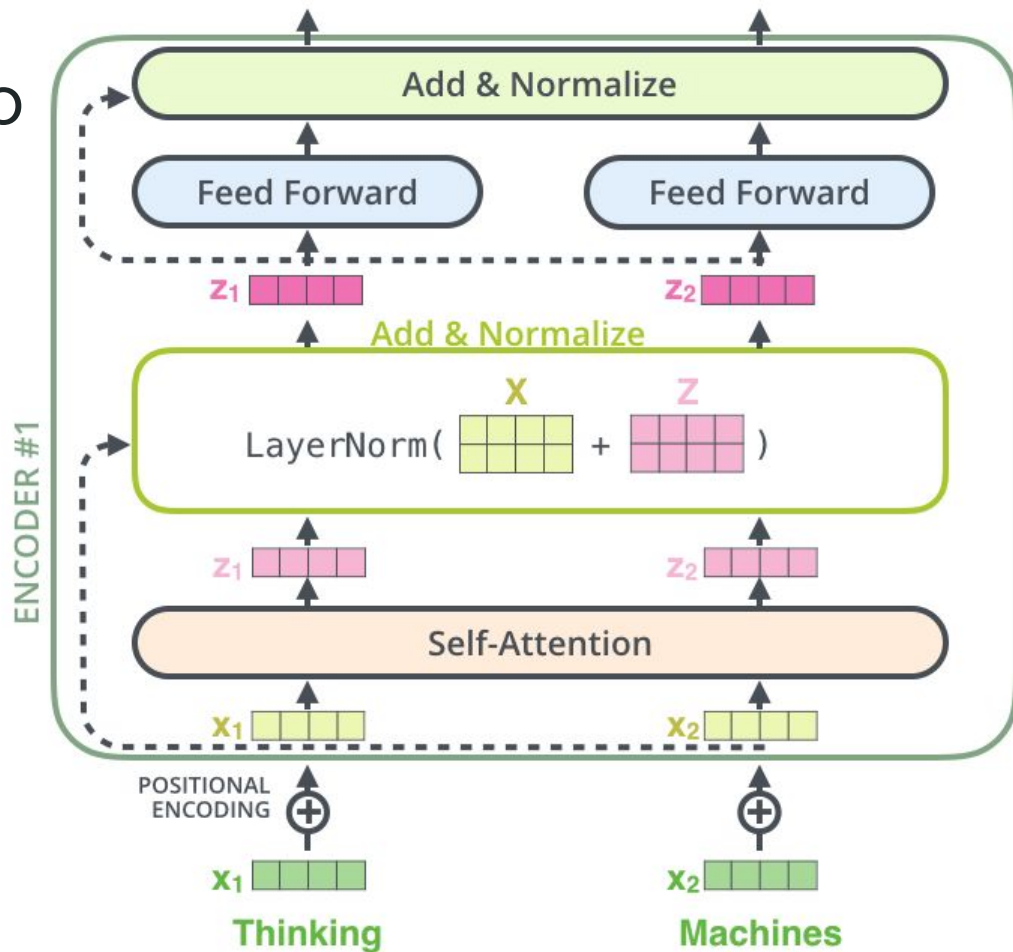


Residuais e Normalização

$$H(x) = F(x) + x$$

$$\text{Norm}(x) = \frac{x - \mu}{\sigma} \times \gamma + \beta$$

Onde μ e σ são a média e o desvio padrão das saídas da subcamada, e γ e β são parâmetros treináveis.

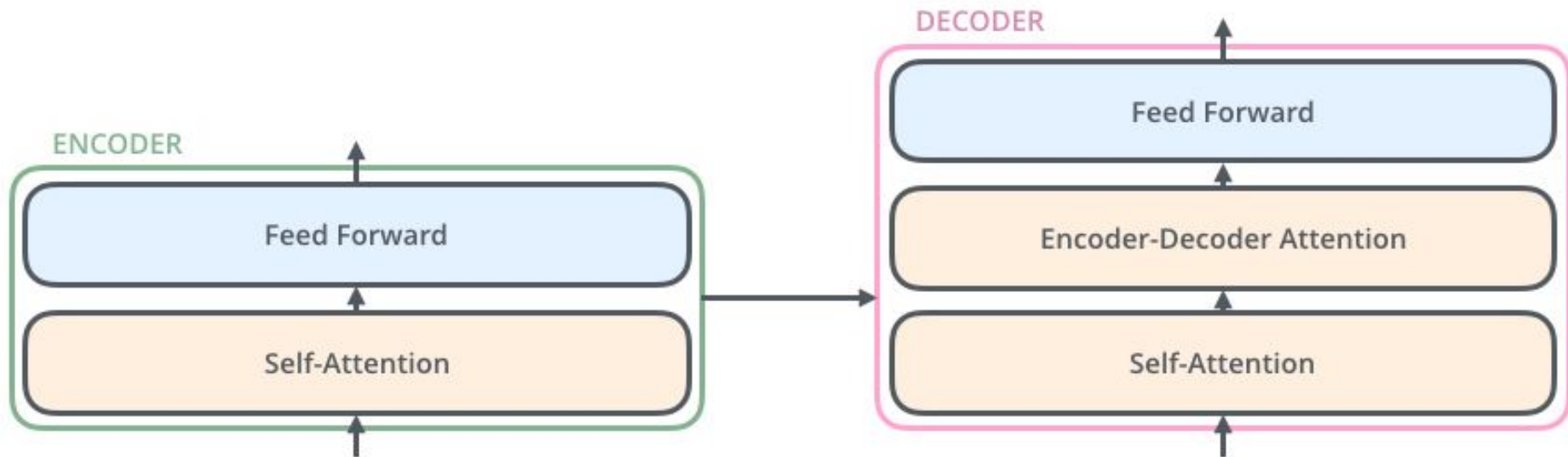


Parâmetros treináveis - Encoder

1. **Matriz de Embedding de Palavras:** Usada para converter tokens de entrada em vetores de embedding.
2. **Matrizes de Pesos para Queries (Q), Keys (K) e Values (V) na Camada de Atenção:**
 1. **WQ** : para transformar a entrada em queries
 2. **WK** : para transformar a entrada em keys
 3. **WV** : para transformar a entrada em values
3. **Matriz de Pesos de Saída da Multi-Cabeça de Atenção (O):**
 1. **WO**: para combinar as saídas das diferentes cabeças de atenção
4. **Parâmetros da Camada de Normalização de Atenção:** Geralmente são escalares beta e gama para cada dimensão do vetor.
5. **Matrizes de Pesos das Camadas Feed-Forward:**
 1. **W1** e **b1**: para a primeira camada linear
 2. **W2** e **b2**: para a segunda camada linear
6. **Parâmetros da Camada de Normalização Feed-Forward:** Geralmente são escalares beta e gama para cada dimensão do vetor.

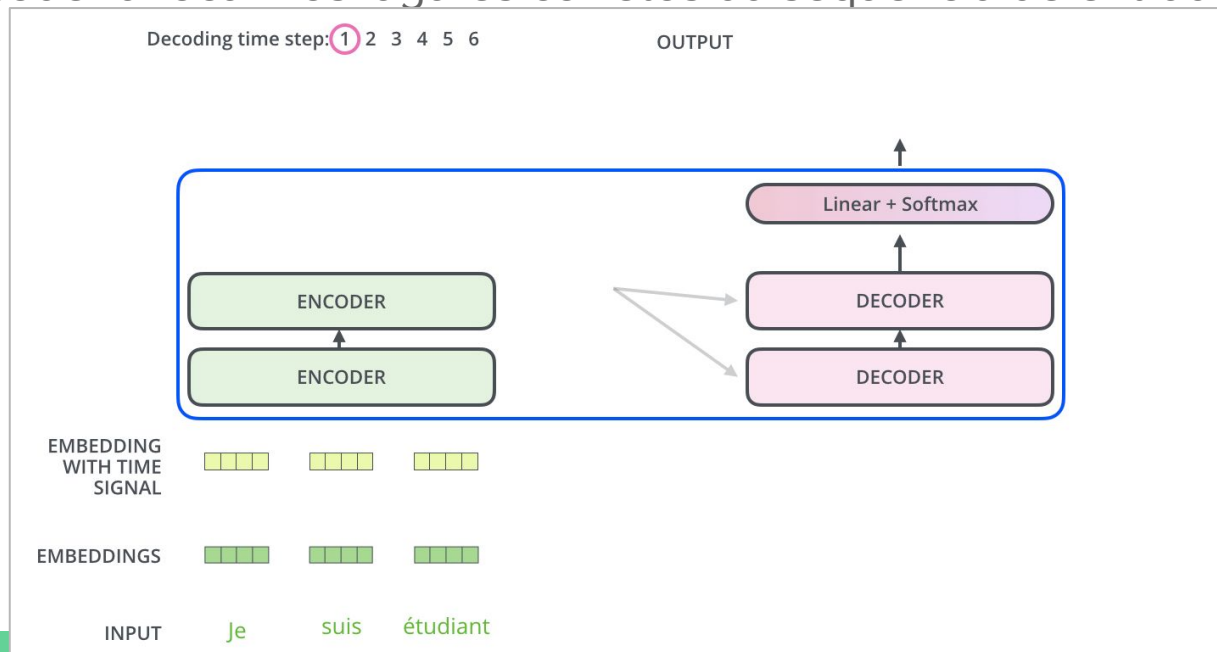
Decoders

Olhando de cima...



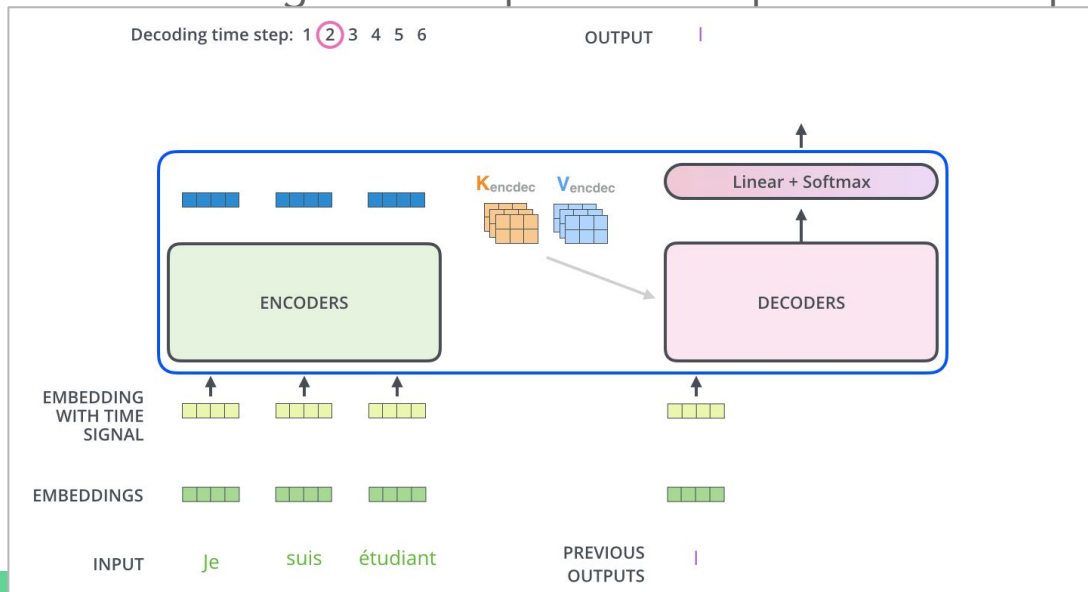
Conexão entre os encoders e os decoders

Após o encoder realizar o processamento da sequência de entrada, a saída do último encoder é transformada em um conjunto de vetores K e V. Esses serão usados por cada decoder na camada de "atenção do encoder-decoder", que auxilia o decoder a focar nos lugares corretos da sequência de entrada.



Conexão entre os decoders

Os próximos passos repetem o processo até que o símbolo de fim ou início da sequência seja atingido, indicado que o decoder completou sua saída. A saída de cada passo vira a entrada do passo seguinte, como acontecia com os encoders. Também como nos encoders, nos decoders são realizados os embeddings dos inputs, juntamente com os argumentos posicionais para indicar a posição de cada palavra.



1ª Subcamada: Multi-headed Attention (com máscara)

Scaled Scores

0.7	0.1	0.1	0.1
0.1	0.6	0.2	0.1
0.1	0.3	0.6	0.1
0.1	0.3	0.3	0.3

+

Look-Ahead Mask

0	-inf	-inf	-inf
0	0	-inf	-inf
0	0	0	-inf
0	0	0	0

=

Masked Scores

0.7	-inf	-inf	-inf
0.1	0.6	-inf	-inf
0.1	0.3	0.6	-inf
0.1	0.3	0.3	0.3

Softmax(

0.7	-inf	-inf	-inf
0.1	0.6	-inf	-inf
0.1	0.3	0.6	-inf
0.1	0.3	0.3	0.3

)

=

	eu	estou	bem	<fim>	
eu	1	0	0	0	eu
estou	0.37	0.62	0	0	estou
bem	0.26	0.31	0.43	0	bem
<fim>	0.21	0.26	0.26	0.26	<fim>

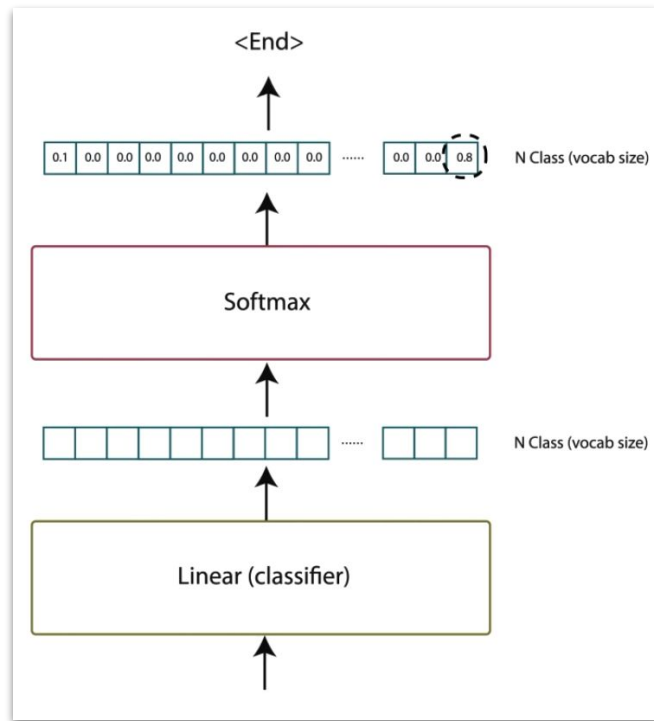
2ª Subcamada: Multi-headed Attention (entre camadas)

- Depois que o decoder processa sua própria sequência com a atenção autoregressiva, ele precisa "consultar" a sequência de entrada para gerar a saída correta. É aqui que entra o módulo de atenção inter-camadas. Esse módulo funciona como um módulo de atenção regular, mas as Queries (Q) vêm da saída do módulo anterior do decoder, enquanto as Keys (K) e Values (V) vêm da saída do encoder*.
- Isso permite que o decoder decida quais partes da entrada são relevantes para cada token que está sendo gerado.

*Esse processo é análogo ao processo de pesquisa no google. Digitamos uma pesquisa (a *query*) e isso vira uma *key* para retornar um *value* que buscamos.

3ª Subcamada: Rede Neural Feed-forward

- O decoder nos retorna como saída um vetor de números. Como transformar isso em uma palavra?
Passando pela camada linear e pela camada softmax.
- A camada linear é uma rede neural completamente conectada que transforma esse vetor em outro vetor de números com o tamanho do vocabulário usado no treinamento (normalização).
- A camada softmax transforma esse vetor de números em um vetor de *probabilidades* (todas positivas e que somam a 1). **Assim, a posição com a maior probabilidade é a palavra escolhida como a saída do passo atual.**



Parâmetros Treináveis - Decoder

1. **Matrizes de Pesos para Queries (Q), Keys (K) e Values (V) na Camada de Atenção:**
 1. **WQ** : para transformar a entrada em queries
 2. **WK** : para transformar a entrada em keys
 3. **WV** : para transformar a entrada em values
2. **Matriz de Projeção de Saída:** para retornar a saída para as dimensões originais.
3. **Feed-forward Neural Network:** pesos e bias para cada uma das duas camadas densas.
4. **Camadas de Normalização:** vetor de escala (gama) e vetor de deslocamento (beta).

Resultados do Modelo Apresentado

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	$3.3 \cdot 10^{18}$	
Transformer (big)	28.4	41.8	$2.3 \cdot 10^{19}$	

Referências

- O artigo seminal "*Attention Is All You Need*", usado como principal fonte de informação para o entendimento dos transformers.
- O guia "*The Illustrated Transformers*" como principal auxílio para a criação das apresentações.
- O artigo "*A Survey of Transformers*" foi utilizado como referência extra.

Dúvidas?