Universidade Federal da Paraíba

Centro de Informática

Departamento de Informática

Aprendizado Profundo

Redes Neurais Transformers

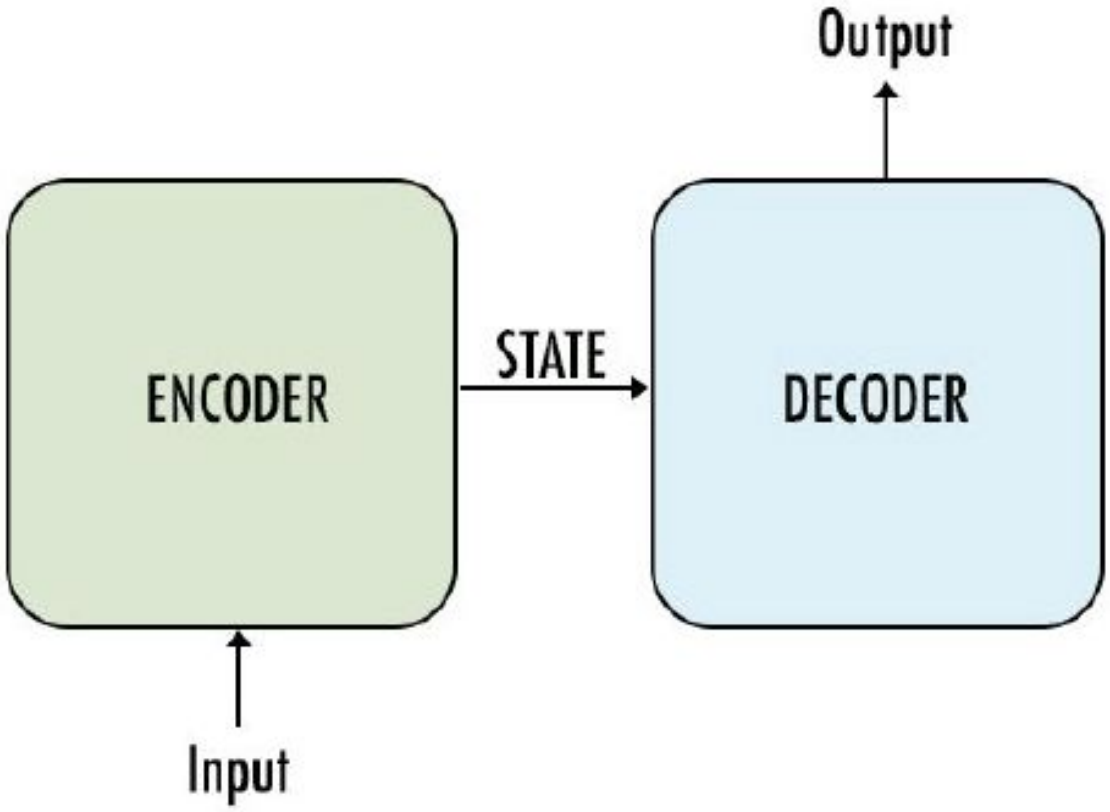
(Material Baseado em @CodeEmporium)

Tiago Maritan

(tiago@ci.ufpb.br)

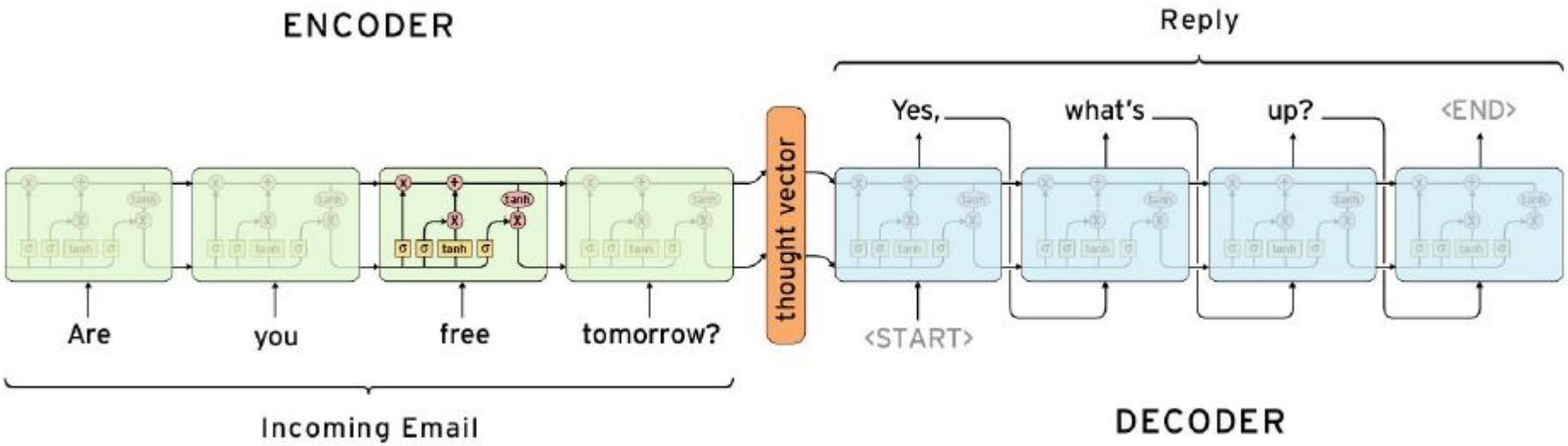
Modelo Seq2Seq - Visão Alto Nível

- Utiliza **2 RNNs separadas**: **Encoder** e **Decoder**

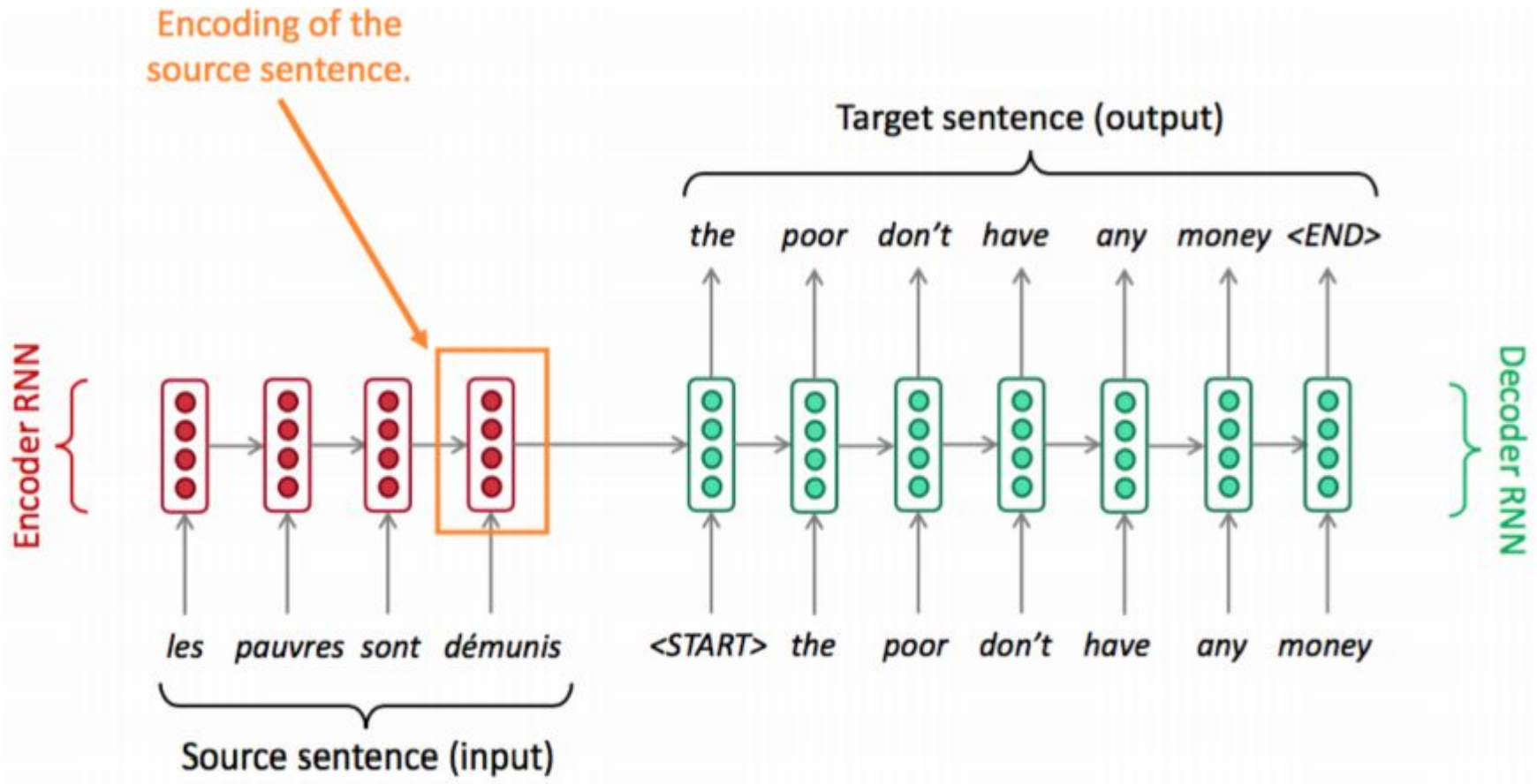


Modelo Seq2Seq - Visão Alto Nível

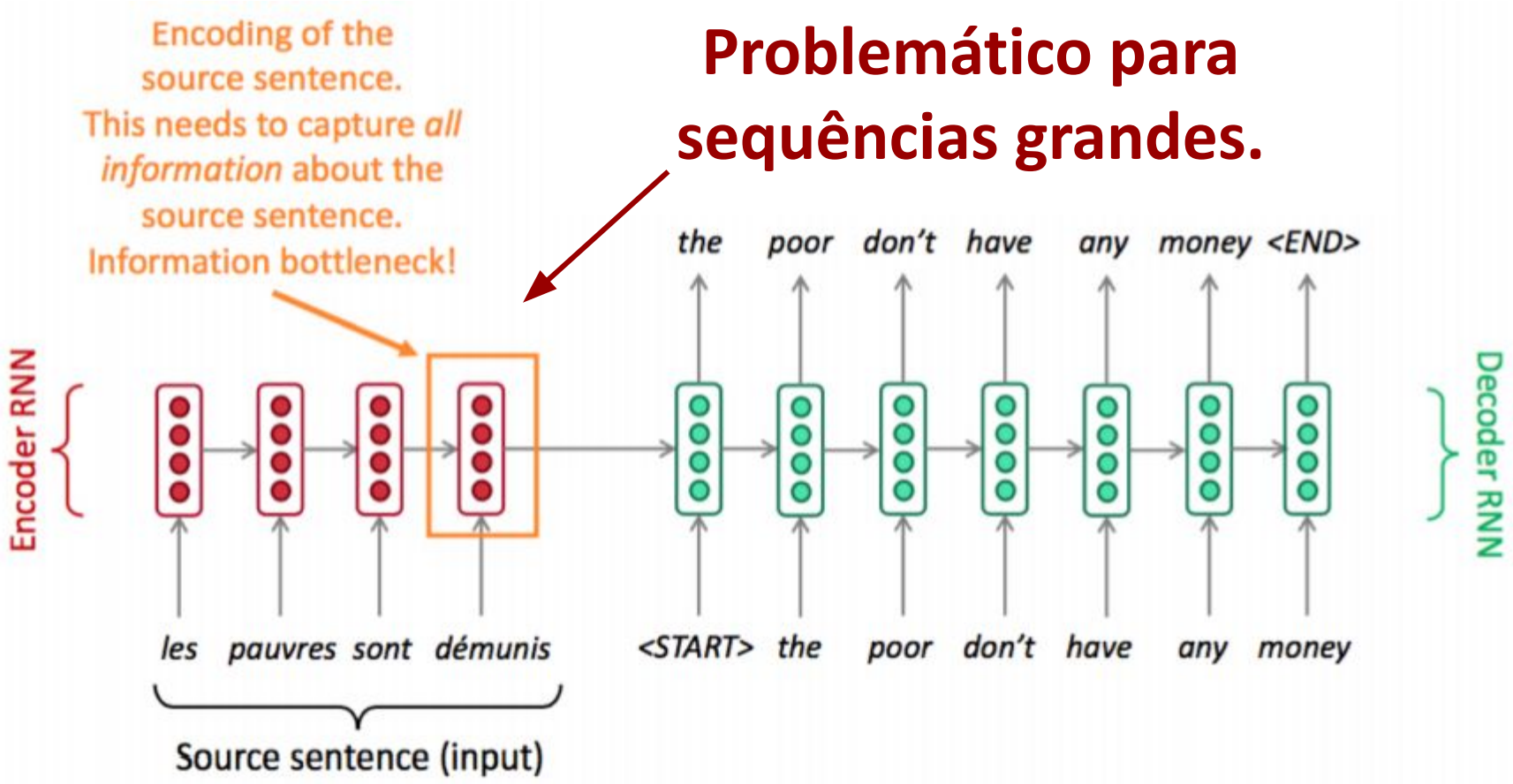
- Utiliza **2 RNNs separadas**: **Encoder** e **Decoder**



Seq2Seq - Problema do Gargalo (*Bootleneck Problem*)



Seq2Seq - Problema do Gargalo (*Bootleneck Problem*)



Mecanismo de Atenção (*Attention Models*)

Mecanismo de Atenção (*Attention*)

- **Attention:** fornece uma solução para o problema do gargalo.

- Ideia principal: em cada passo do **Decoder**, ele **foca num pedaço específico** da sequência inicial.

- Ex: Em uma tradução automática:
  - Na 1a iteração, o **Decoder** foca mais na 1a palavra da entrada (ou 1o estado do Encoder),
  - Na 2a iteração, o **Decoder** foca mais na 2a palavra,
  - E assim por diante...

Mecanismo de Atenção (*Attention*)

- Fazendo um paralelo com imagens

- Ex: Na imagem, uma garota arremessa um frisbee.

  - Para gerar a legenda da imagem, não precisamos prestar atenção em todas as regiões (pixels) da imagem para gerar cada palavra da saída;
  - Podemos focar no que é mais importante a cada passo do tempo

Mecanismo de Atenção (*Attention*)

- É isso que o mecanismo de atenção faz!!!

- **Aprende a <u>focar nas partes mais importantes da entrada</u>**… "borrando" o resto (dando menos importância).

Mecanismo de Atenção (*Attention*)

- **A <u>atenção</u>** seria maior na região da **garota** quando o decoder estiver gerando a saída da palavra “garota”;

- **A <u>atenção</u>** seria maior na região do **frisbee** quando o decoder estiver gerando a saída “frisbee”;

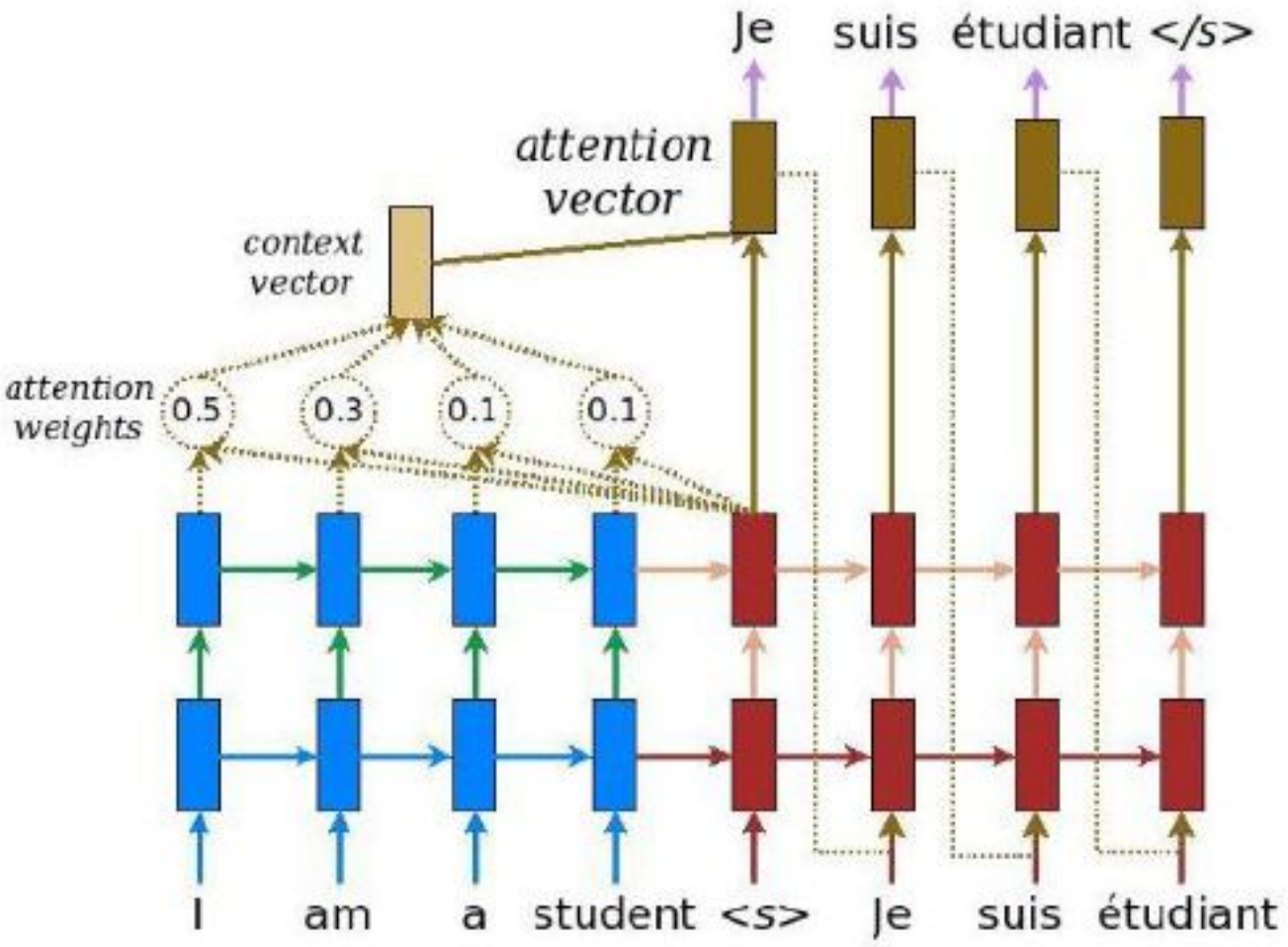
Mecanismo de Atenção - Como funciona?

- Dá acesso ao Decoder a todas as saídas do Encoder (não apenas a última saída);
  - Interessante porque mostra como o Encoder evolui depois de ver cada novo token

- Contudo, não usa as saídas brutas do Encoder.
  - Calculamos **pesos (*attention weigths*)** que representam a importância das saídas do Encoder para a decisão do Decoder na etapa *t*.

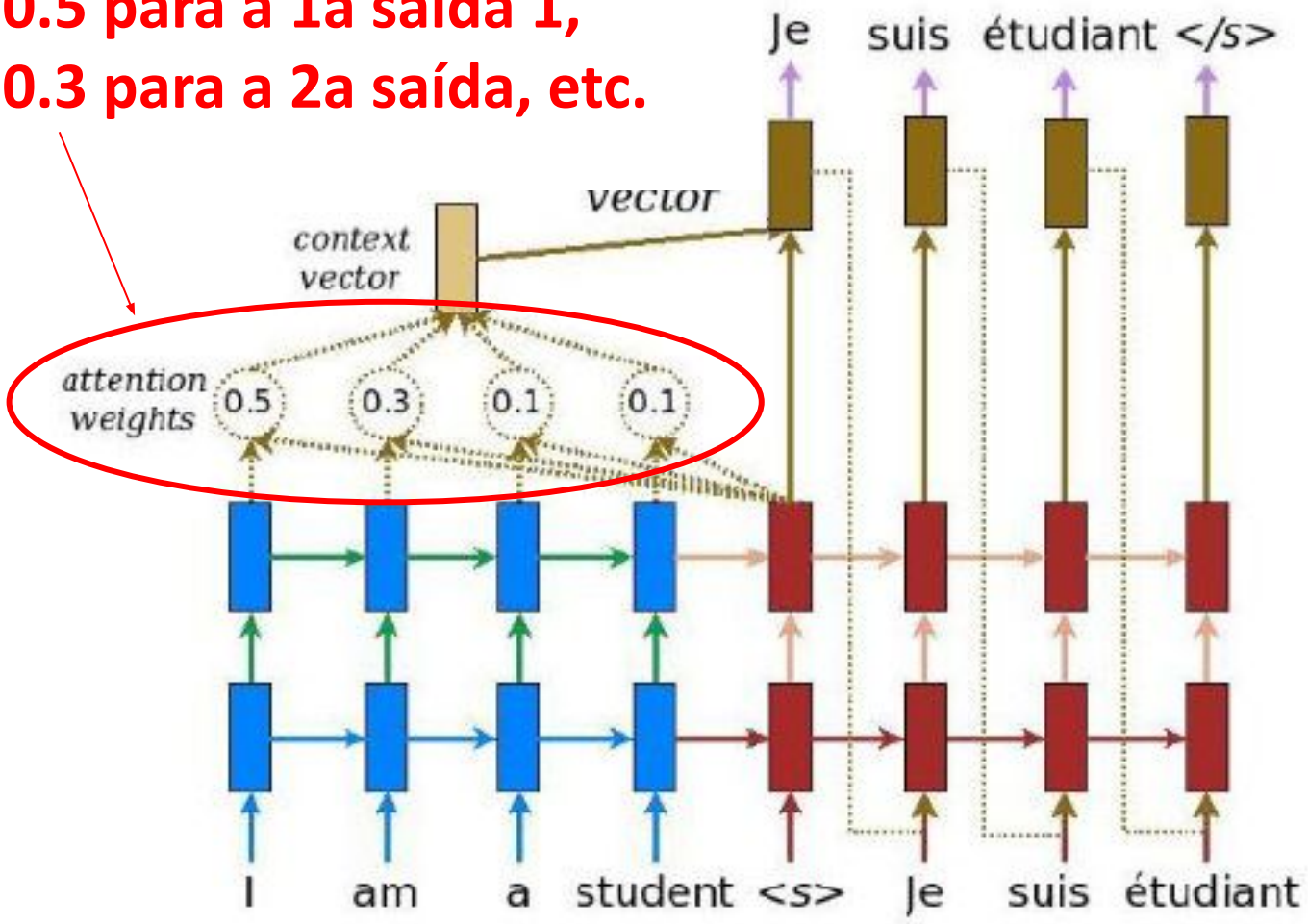
- **Decoder** pode então dinamicamente prestar mais atenção a um subconjunto das saídas do Encoder;

Mecanismo de Atenção - Como funciona?



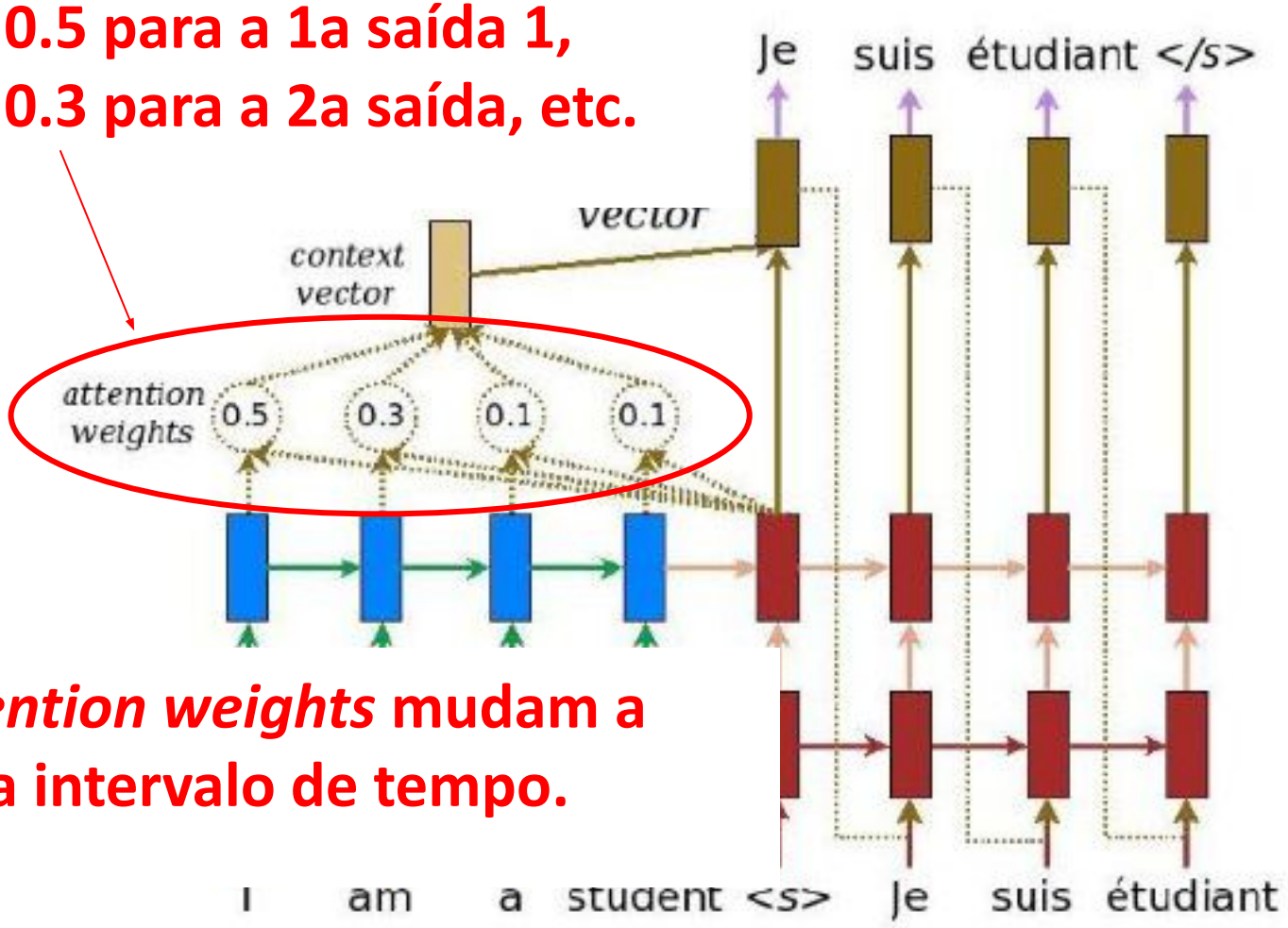
Mecanismo de Atenção - Como funciona?

Peso 0.5 para a 1a saída 1,
Peso 0.3 para a 2a saída, etc.



Mecanismo de Atenção - Como funciona?

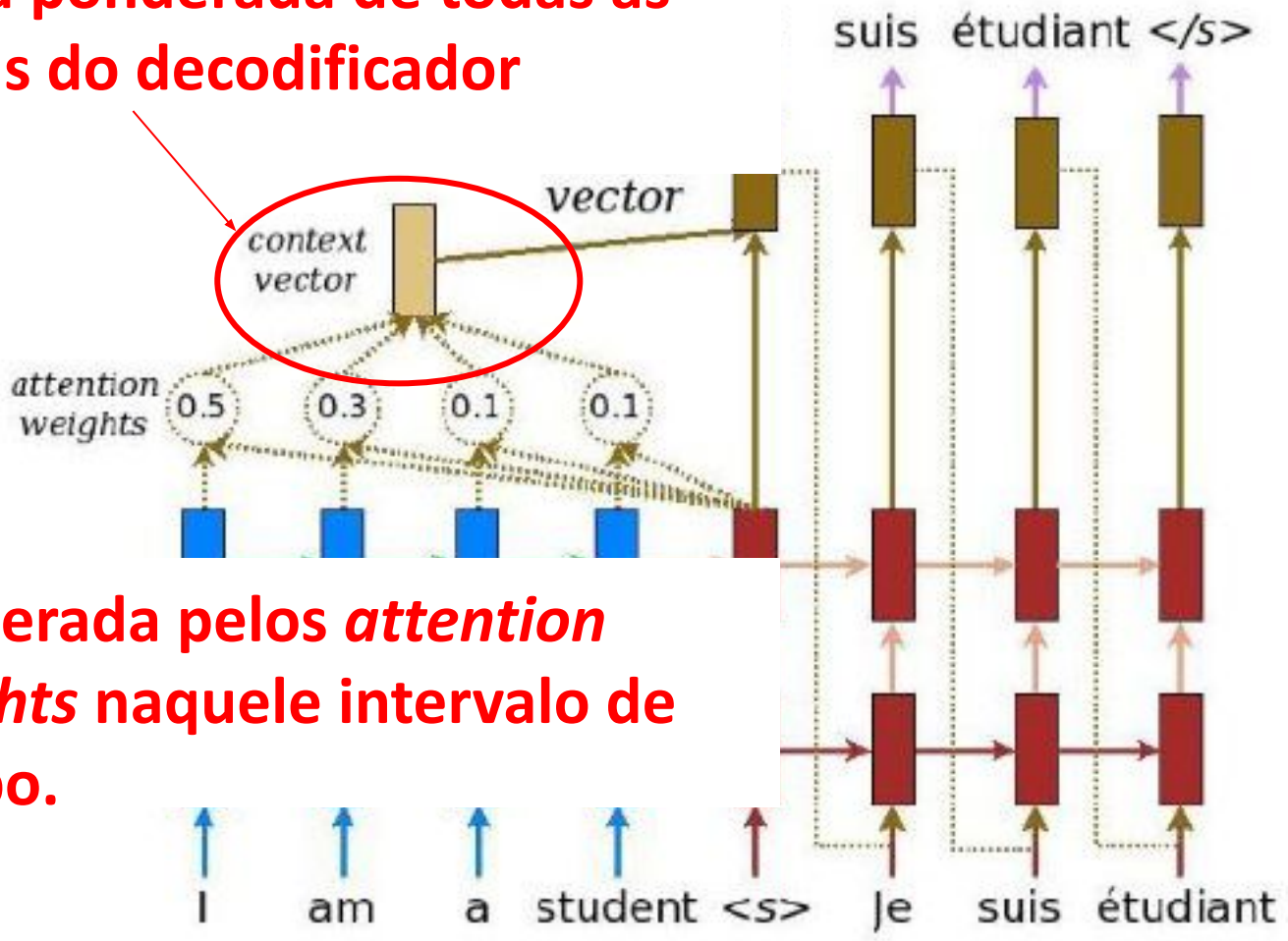
Peso 0.5 para a 1a saída 1,
Peso 0.3 para a 2a saída, etc.



***Attention weights* mudam a cada intervalo de tempo.**

Mecanismo de Atenção - Como funciona?

Soma ponderada de todas as saídas do decodificador



Ponderada pelos *attention weights* naquele intervalo de tempo.

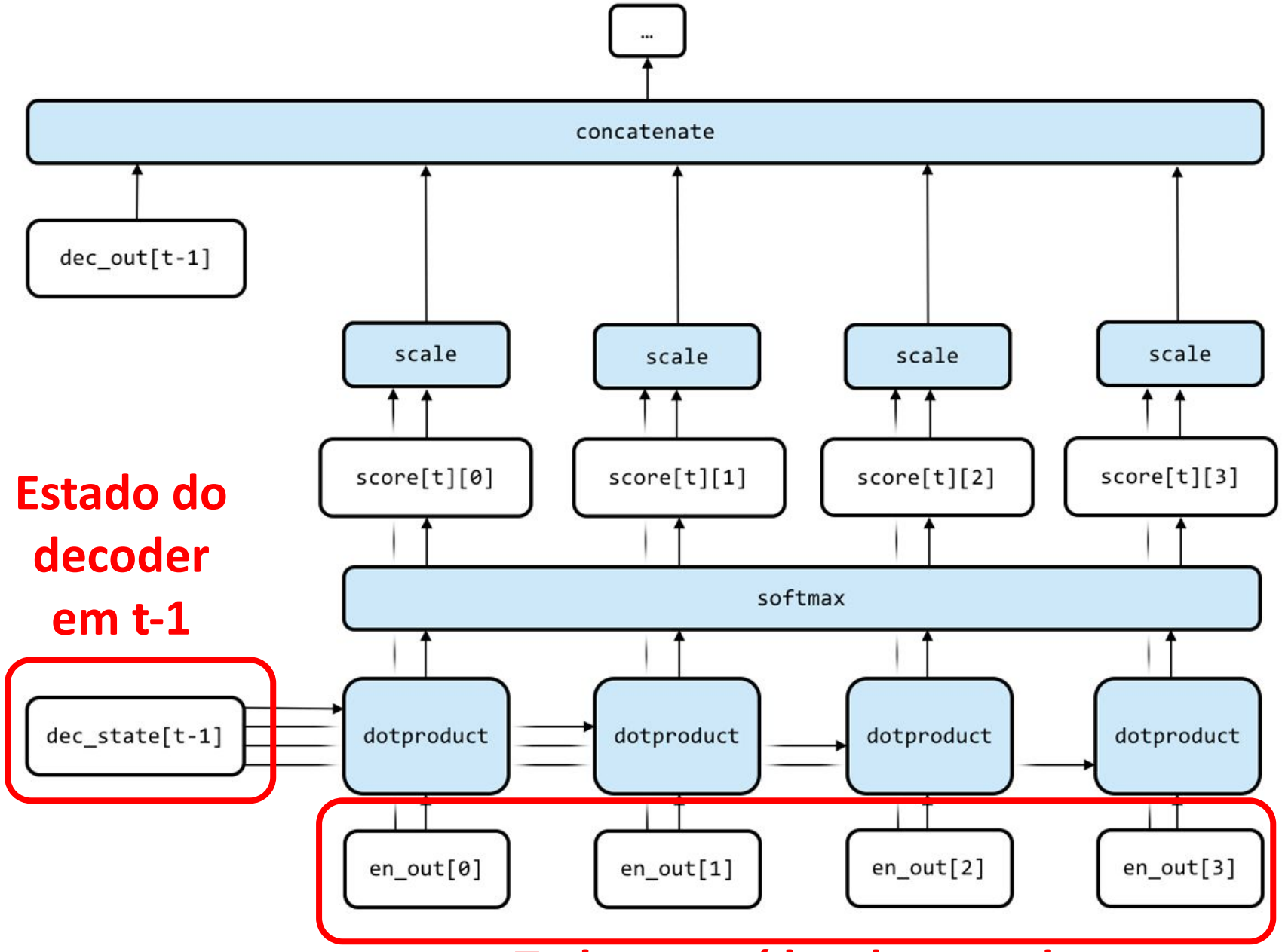
Mecanismo de Atenção - Como funciona?

- Os **attention weigths** são gerados por uma rede neural pequena chamada **camada de atenção**

- Treinada com o resto do modelo Encoder-Decoder

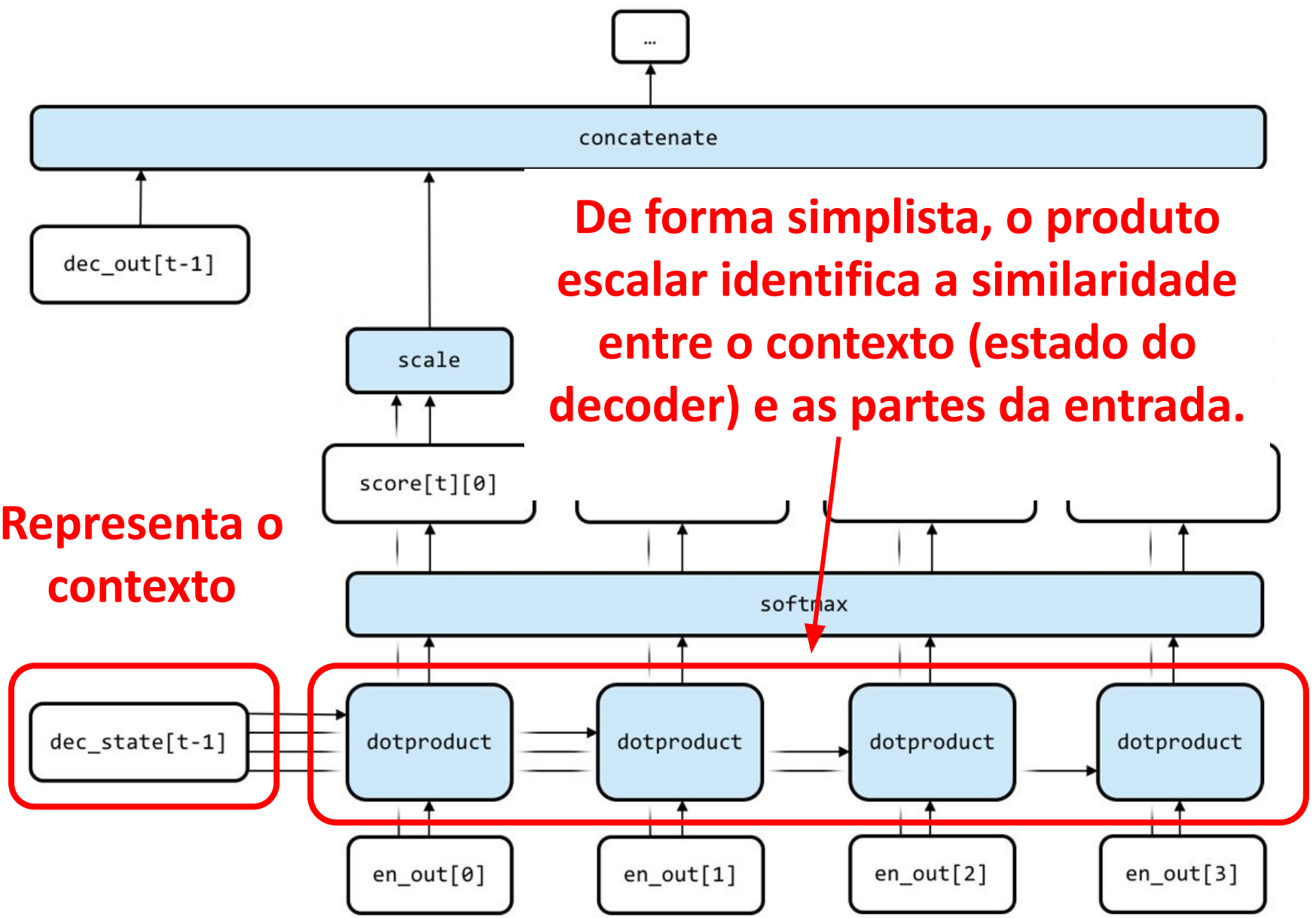
- **Mecanismo de Atenção de Luong[1]:**
  - Similaridade entre as saídas do Encoder e o estado oculto anterior do Decoder é calculado usando o **produto escalar**
  - Produto escalar é uma boa medida de similaridade;

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 1412–1421, Lisbon, Portugal.



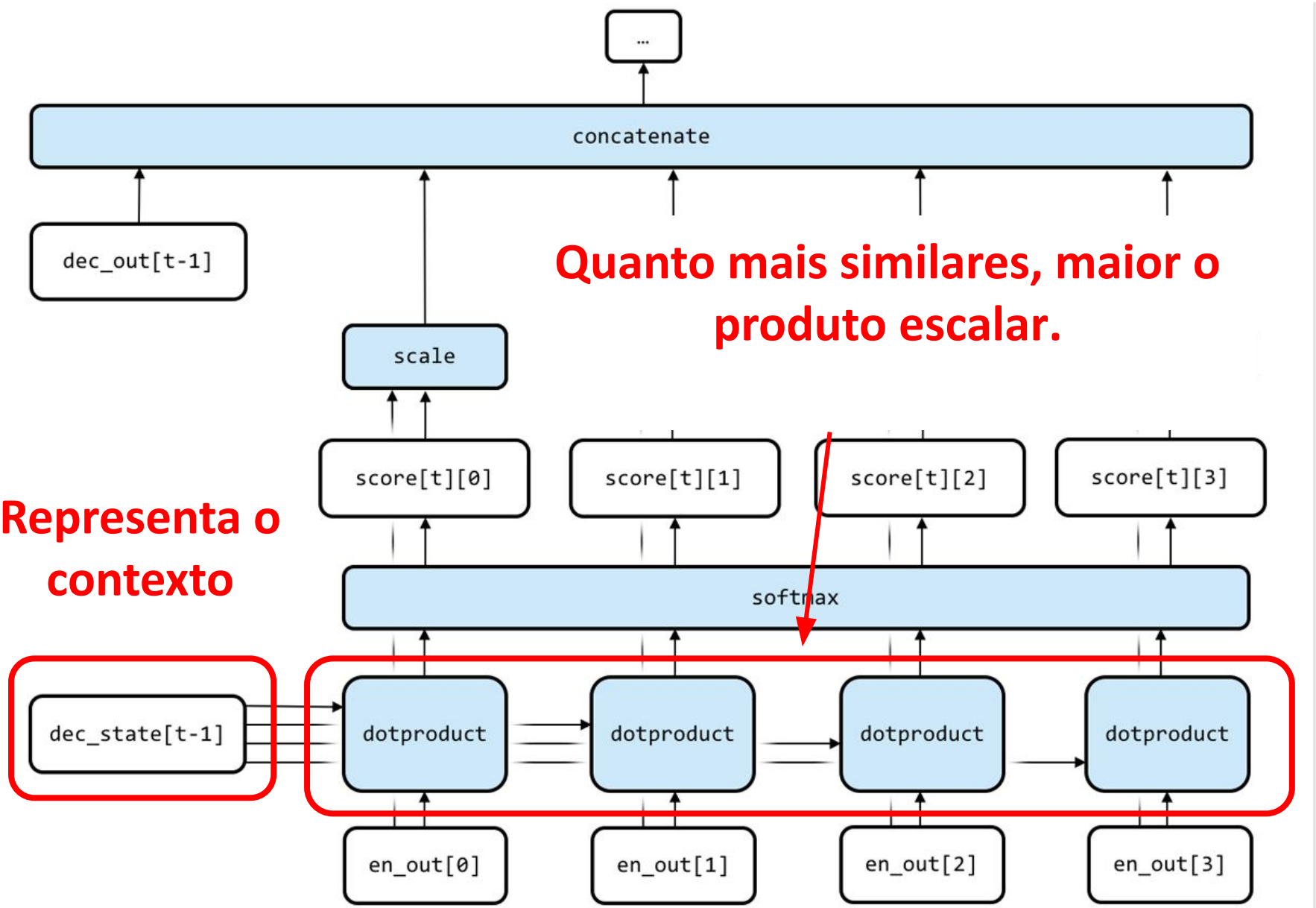
Estado do decoder em t-1

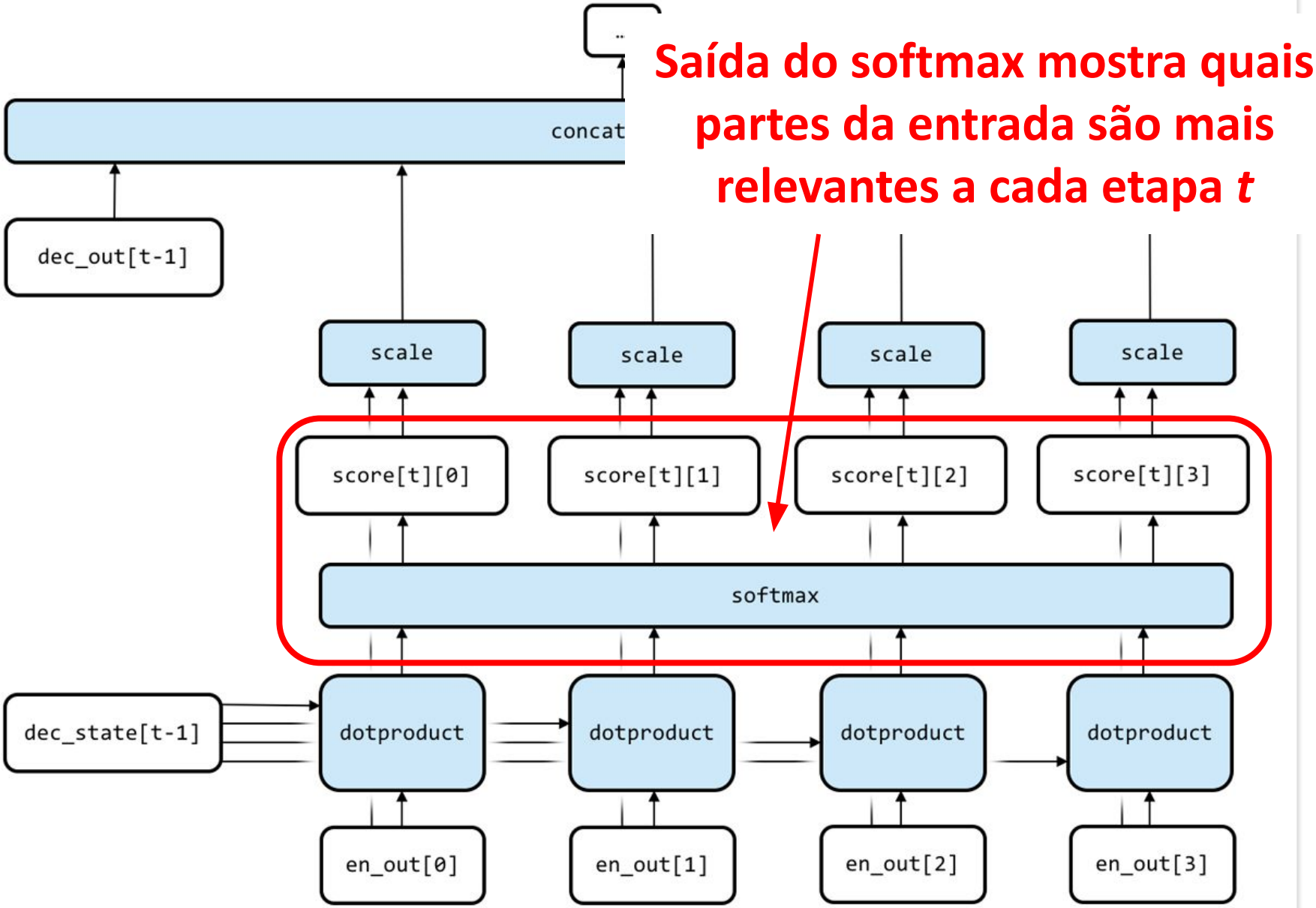
Todas as saídas do encoder

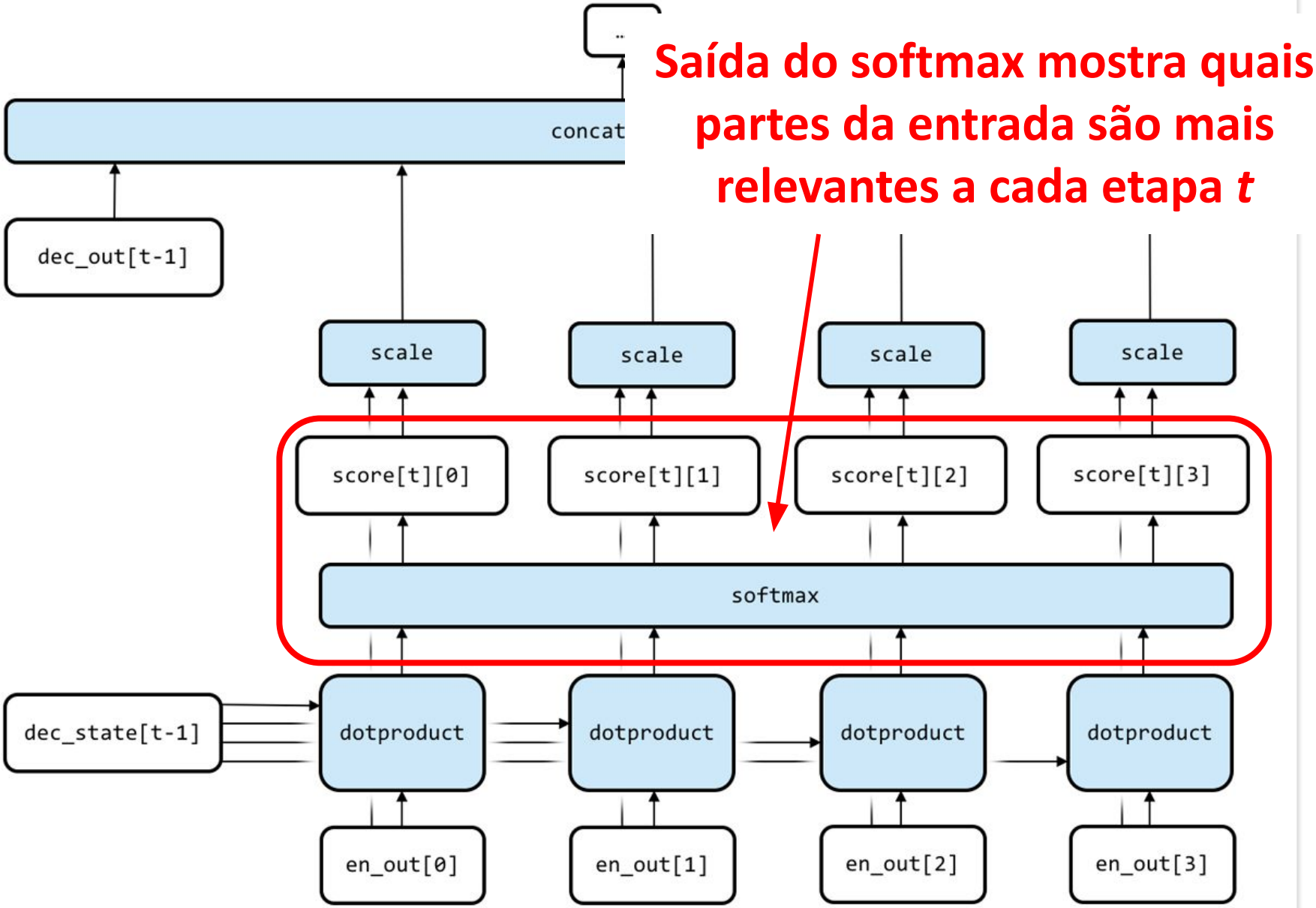


De forma simplista, o produto escalar identifica a similaridade entre o contexto (estado do decoder) e as partes da entrada.

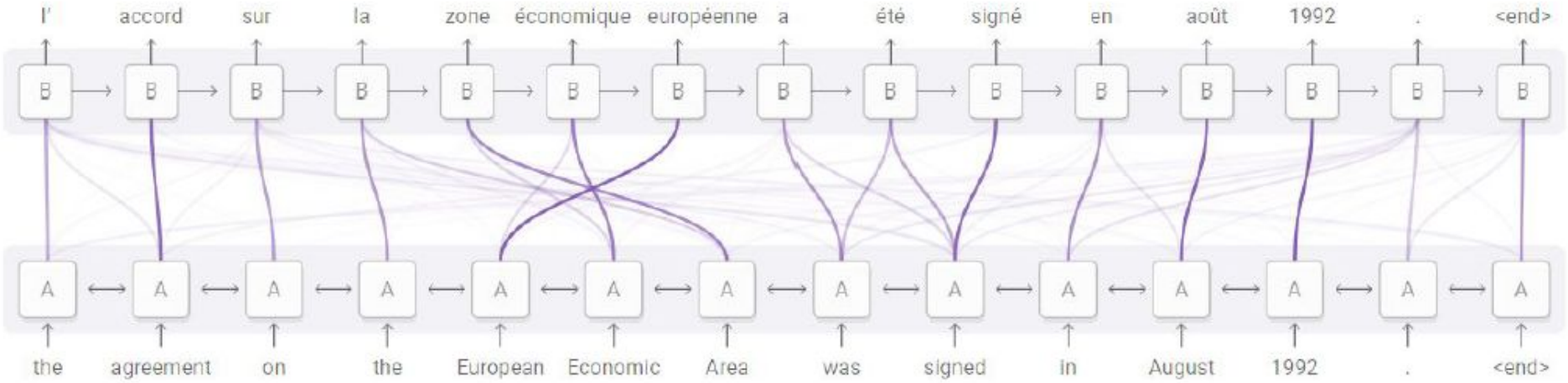
Representa o contexto


...
concatenate
dec_out[t-1]
Quanto mais similares, maior o produto escalar.
scale
score[t][0]
score[t][1]
score[t][2]
score[t][3]
Representa o contexto
softmax
dec_state[t-1]
dotproduct
dotproduct
dotproduct
dotproduct
en_out[0]
en_out[1]
en_out[2]
en_out[3]


Saída do softmax mostra quais partes da entrada são mais relevantes a cada etapa *t*
concat
dec_out[t-1]
scale
scale
scale
scale
score[t][0]
score[t][1]
score[t][2]
score[t][3]
softmax
dec_state[t-1]
dotproduct
dotproduct
dotproduct
dotproduct
en_out[0]
en_out[1]
en_out[2]
en_out[3]

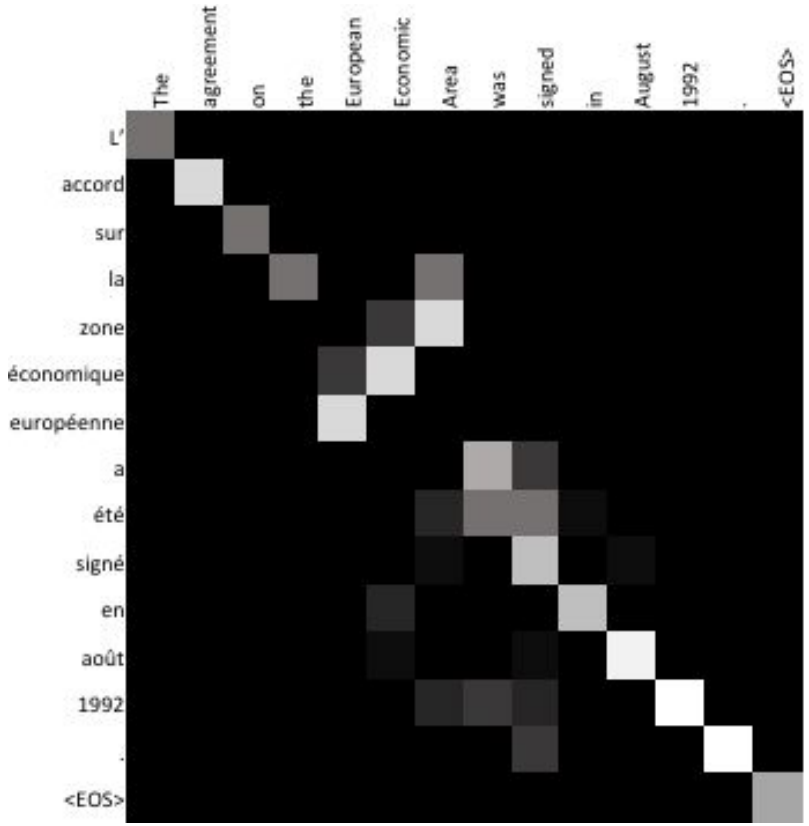

Saída do softmax mostra quais partes da entrada são mais relevantes a cada etapa *t*
concat
dec_out[t-1]
scale
scale
scale
scale
score[t][0]
score[t][1]
score[t][2]
score[t][3]
softmax
dec_state[t-1]
dotproduct
dotproduct
dotproduct
dotproduct
en_out[0]
en_out[1]
en_out[2]
en_out[3]

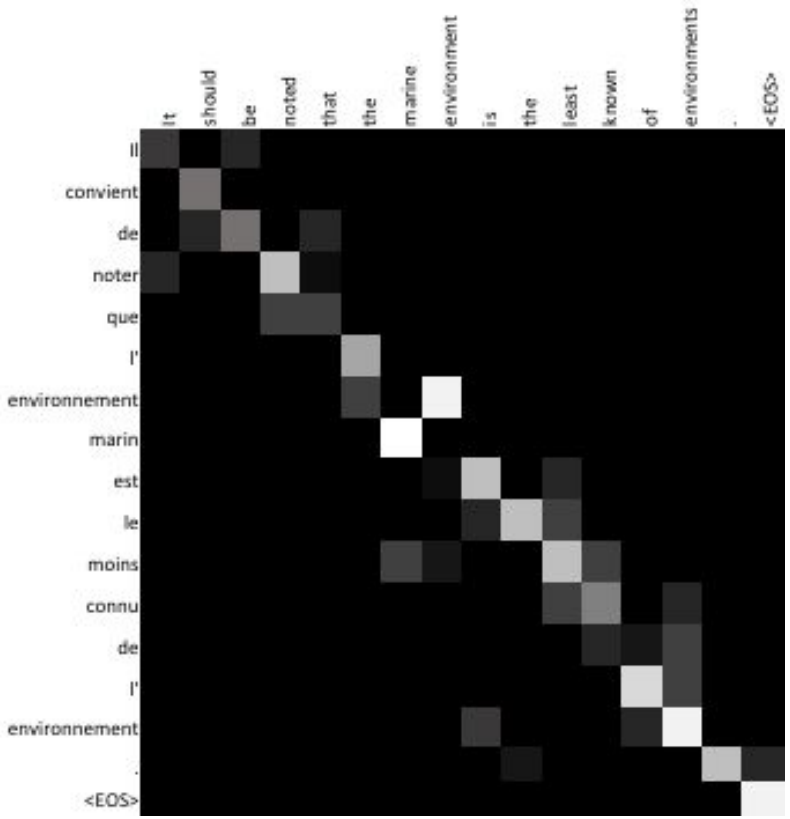
Atenção - Exemplo



Atenção - Exemplo

- Quanto mais claro o quadrado, mais atenção (attention) naquele elemento





Redes Neurais
Transformers

Artigo: https://arxiv.org/abs/1706.03762

Motivação

- Redes recorrentes e LSTM são **lentas para treinar;**
- Dados precisam ser passados sequencialmente na rede, um após o outro
- Necessário que entradas do estado anterior sejam processadas para realizar operações no estado atual;
- Esse fluxo sequencial das LSTMs não explora bem o poder de processamento paralelo das GPUs

Como utilizar paralelização em dados sequenciais?

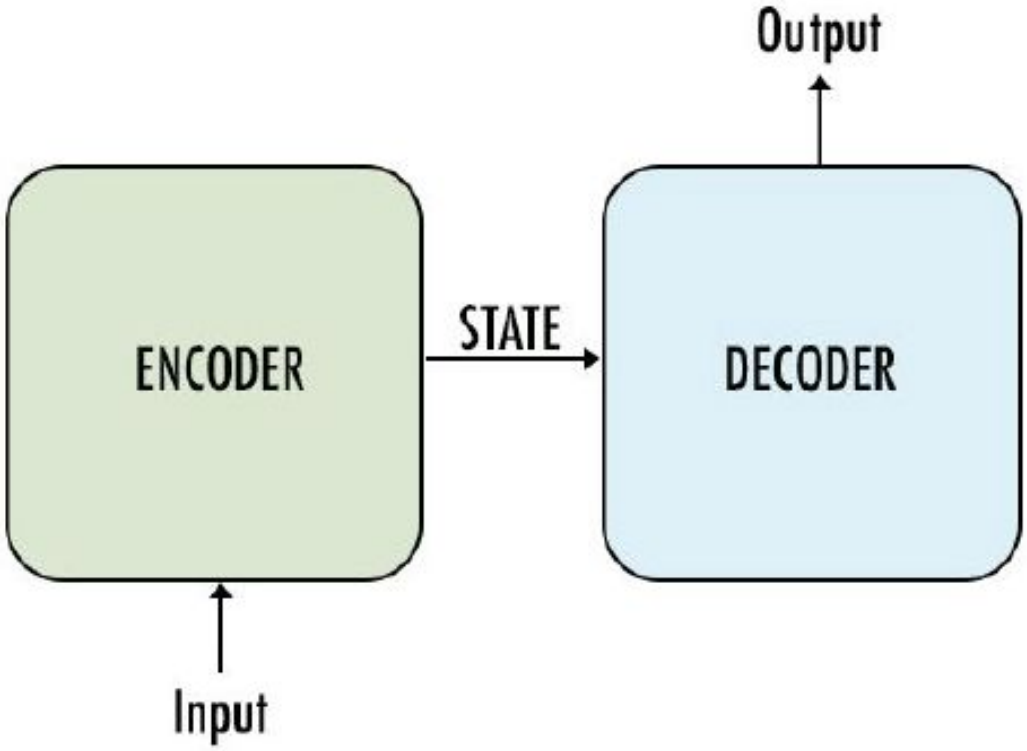
- Em 2017, foram criadas as **Redes Neurais Transformers.**

- Elas permitem que a sequência de entrada possa ser processada em paralelo

- Não existe o conceito de passo de tempo (**time step**) para os dados de entrada;
  - Ex: Numa tradução de sentenças, as palavras são passadas simultaneamente, e os words embeddings são determinados simultaneamente;

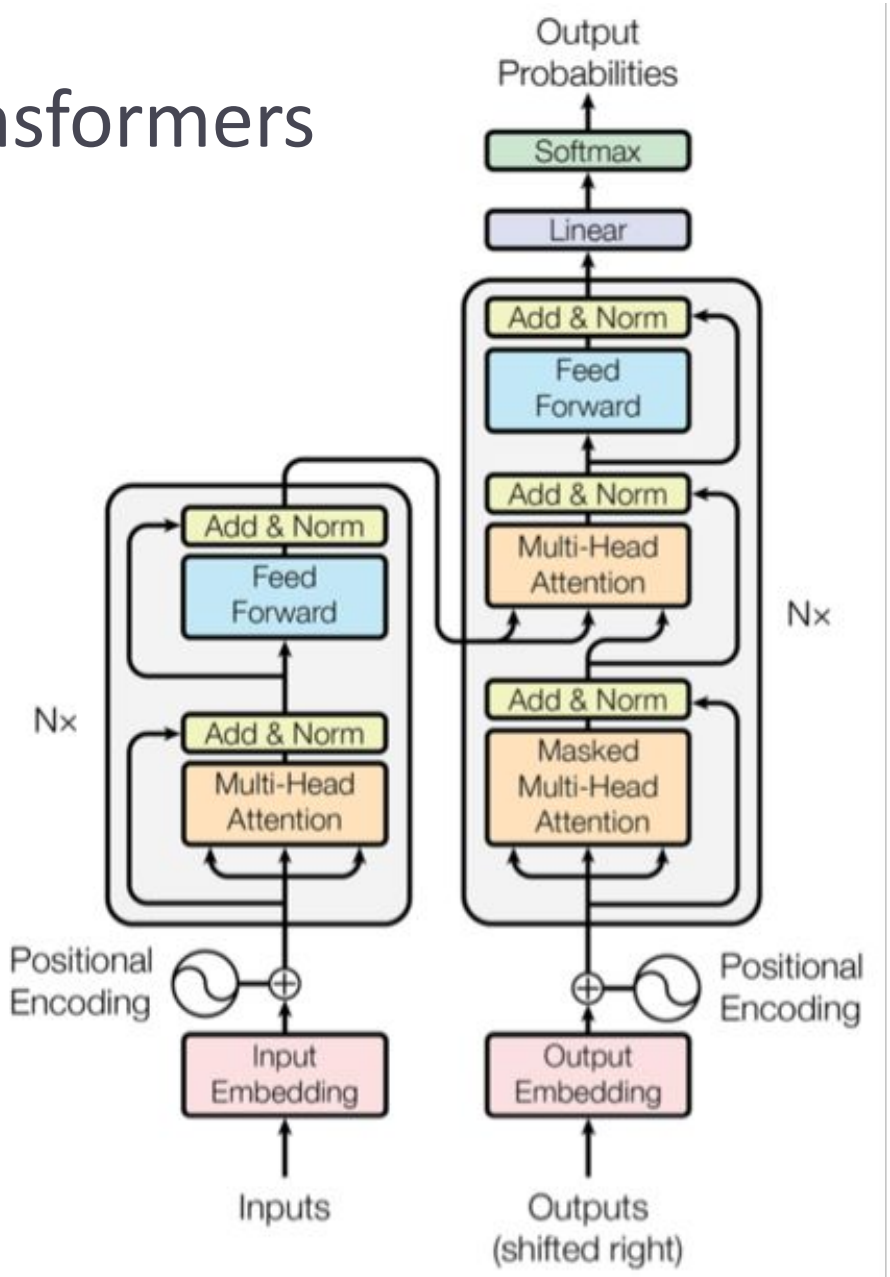
Redes Neurais Transformers

- Arquitetura do tipo Encoder-Decoder
  - Fortemente baseada no conceito de **Atenção**!!!
  - "**Attention is all we need!**"



Artigo: https://arxiv.org/abs/1706.03762

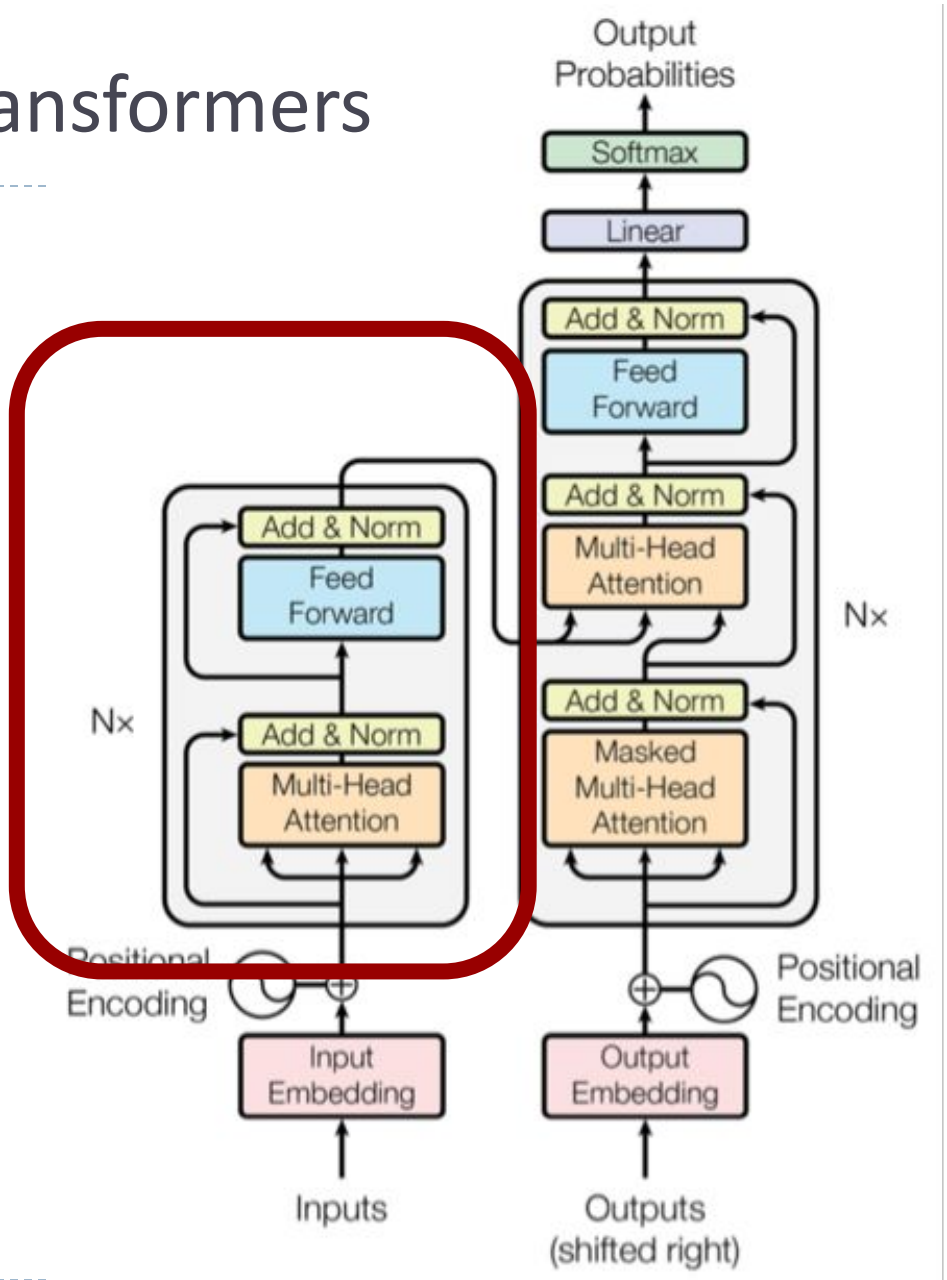
Redes Transformers



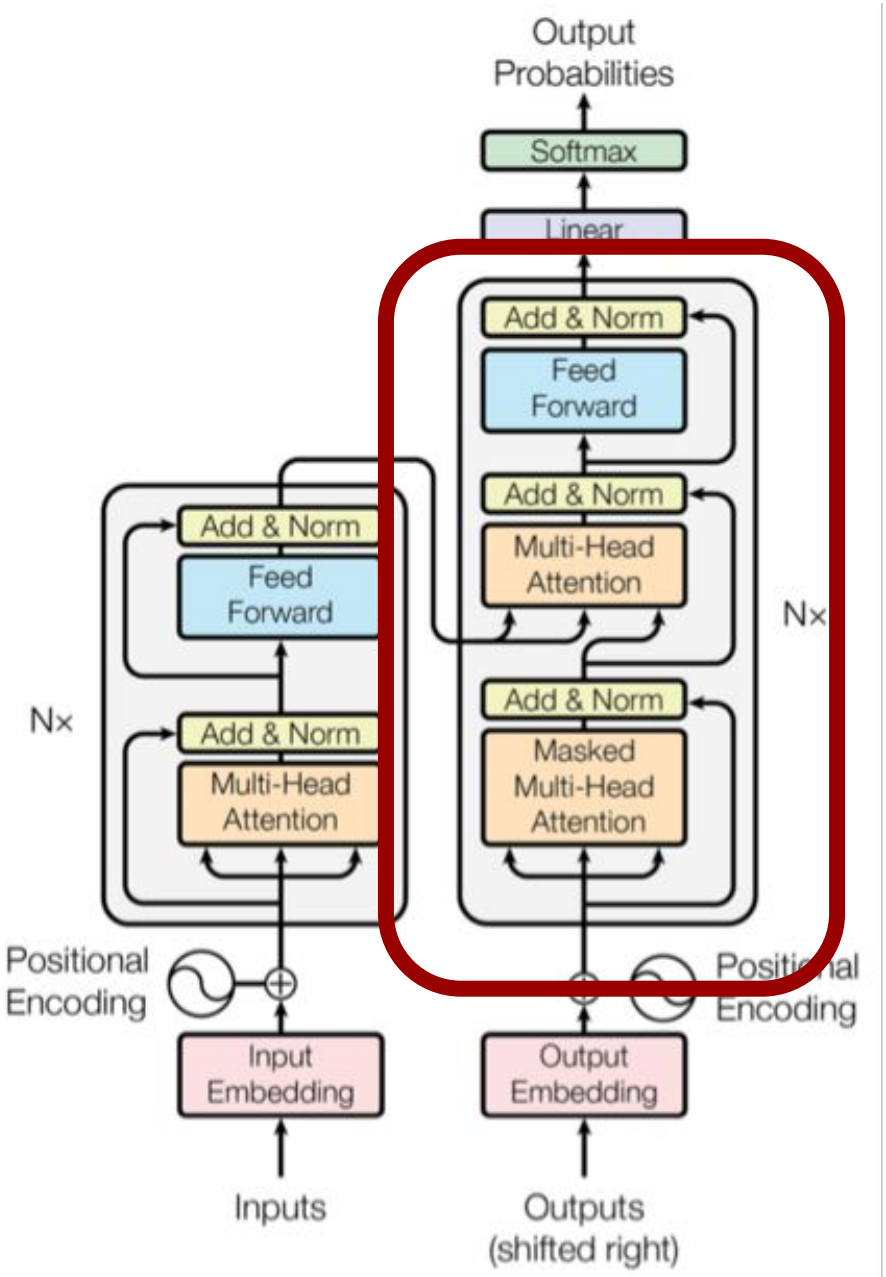
Redes Transformers

Encoder

6 camadas iguais



Redes Trar



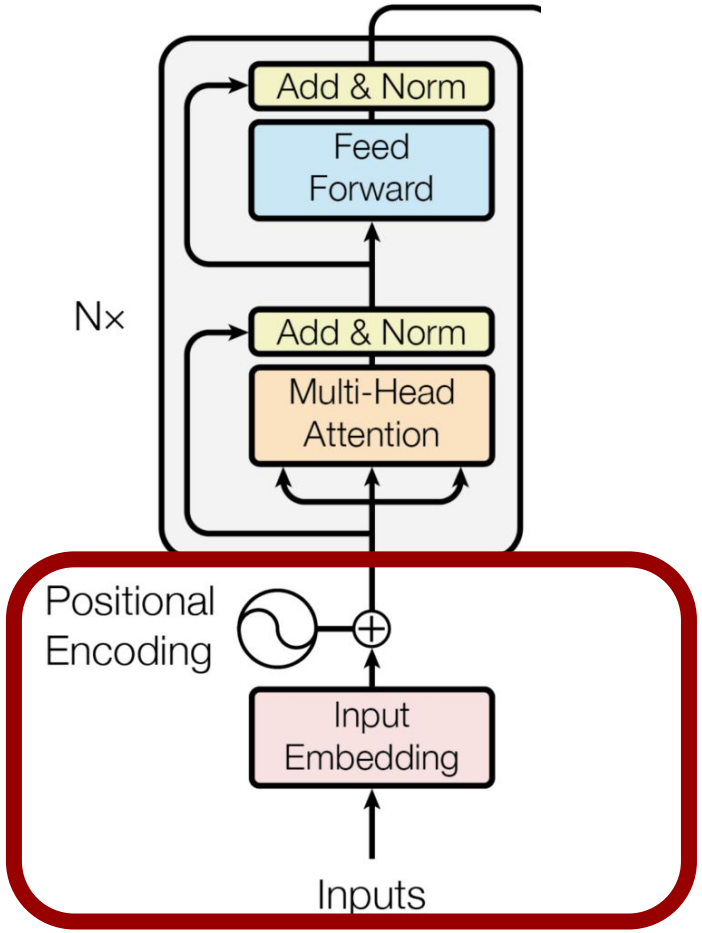
Decoder

6 camadas iguais

Consideremos, como exemplo,
uma Rede para Traduzir Inglês->Francês…

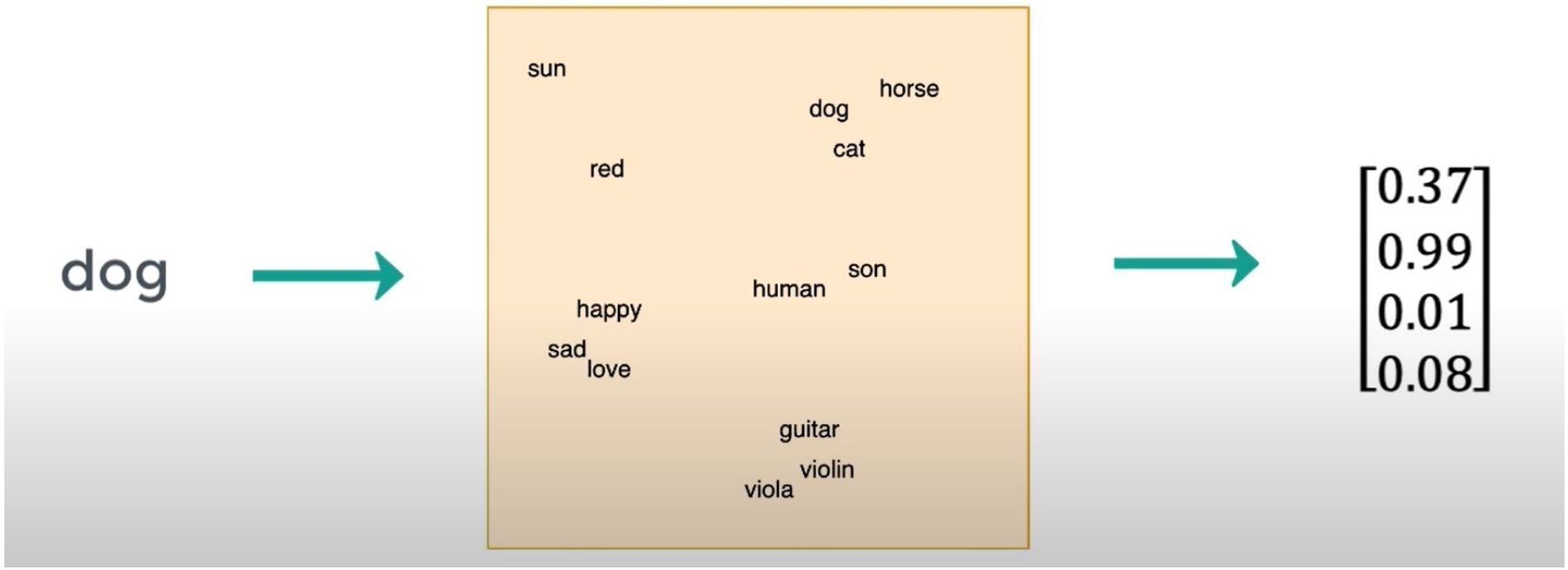
Input Embedding + Positional Encoding

- Na Rede Transformer, inicialmente a entrada passa por um processo de **Input embedding (com Positional Encoding).**



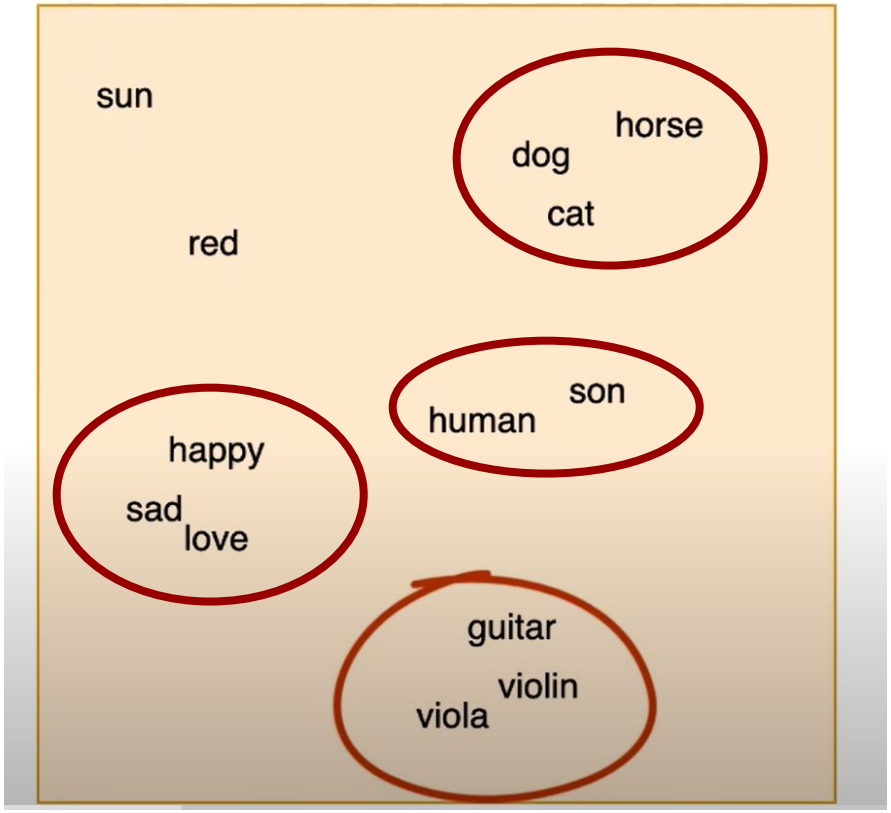
Input Embedding

- Para ser processada por uma rede neural, as palavras são normalmente convertidas para um vetor numérico
  - Ex: Word2Vec, GloVe



Input Embedding

- Normalmente palavras com sentidos/contextos próximos possuem vetores com valores próximos



Input Embedding + Positional Encoding

- Uma mesma palavra pode ter diferentes significados em sentenças diferentes.

- **Positional encoding:** Gera um vetor que representa o contexto da palavra de acordo com a sua posição na sentença.

Input Embedding + Positional Encoding

Codificar a posição do token é importante porque tokens são alimentados simultaneamente em redes Transformers

Embedding de "dog"

Vetor que codifica da posição da palavra na sentença

Embedding de "dog" (com info de contexto)

Positional Encoding

- Embeddings posicionais poderiam ser aprendidos pelo modelo;

- Mas no artigo original, os autores preferiram usar **embeddings posicionais fixos**, calculado usando funções seno e cosseno de frequências diferentes

***PE* - matriz do embedding posicional**

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/512}}\right)$$

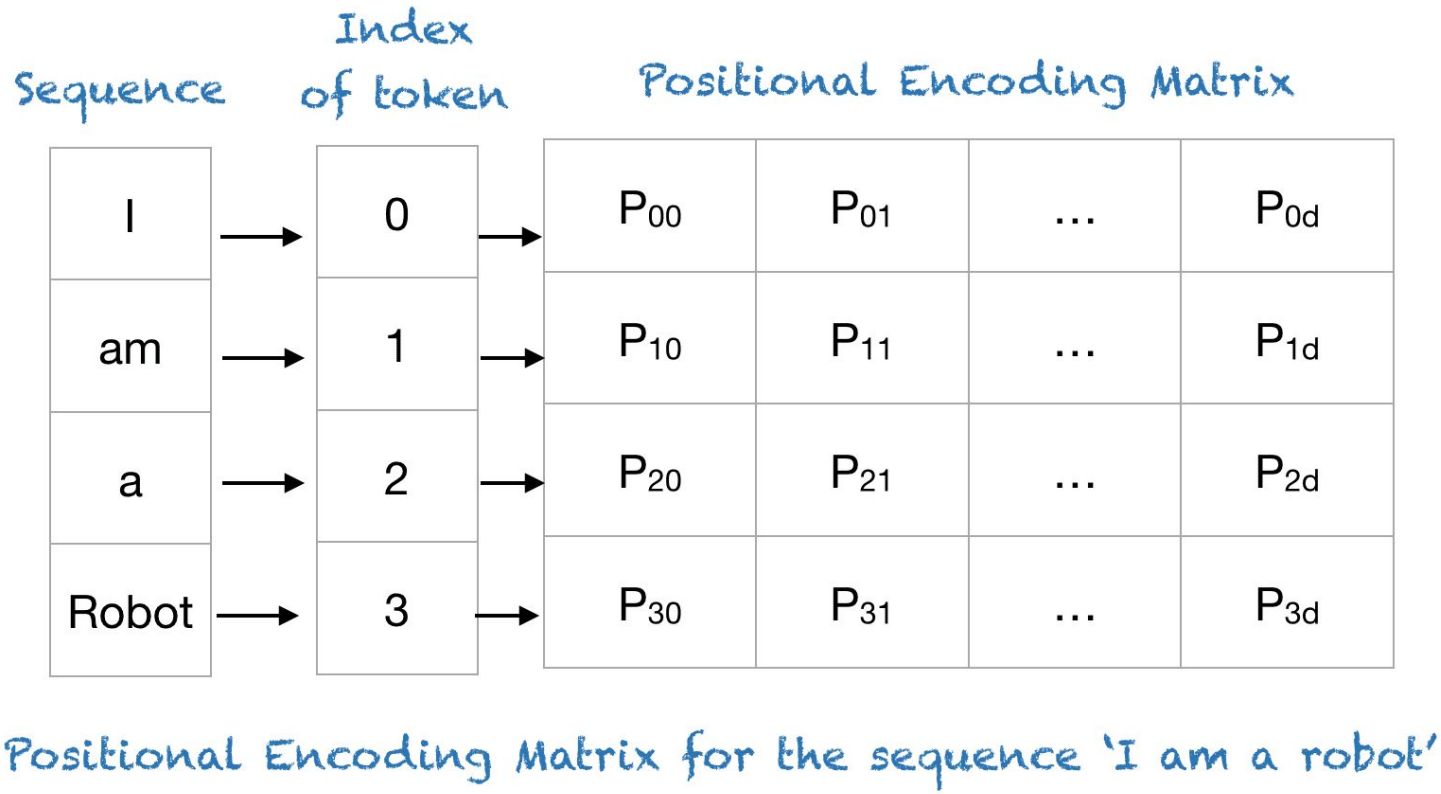
$$PE_{(pos,2*i+1)} = \cos\left(\frac{pos}{10000^{2i/512}}\right)$$

***pos* - posição da palavra na sentença**

***i* - dimensão dentro do vetor (diferencia posições pares de ímpares)**

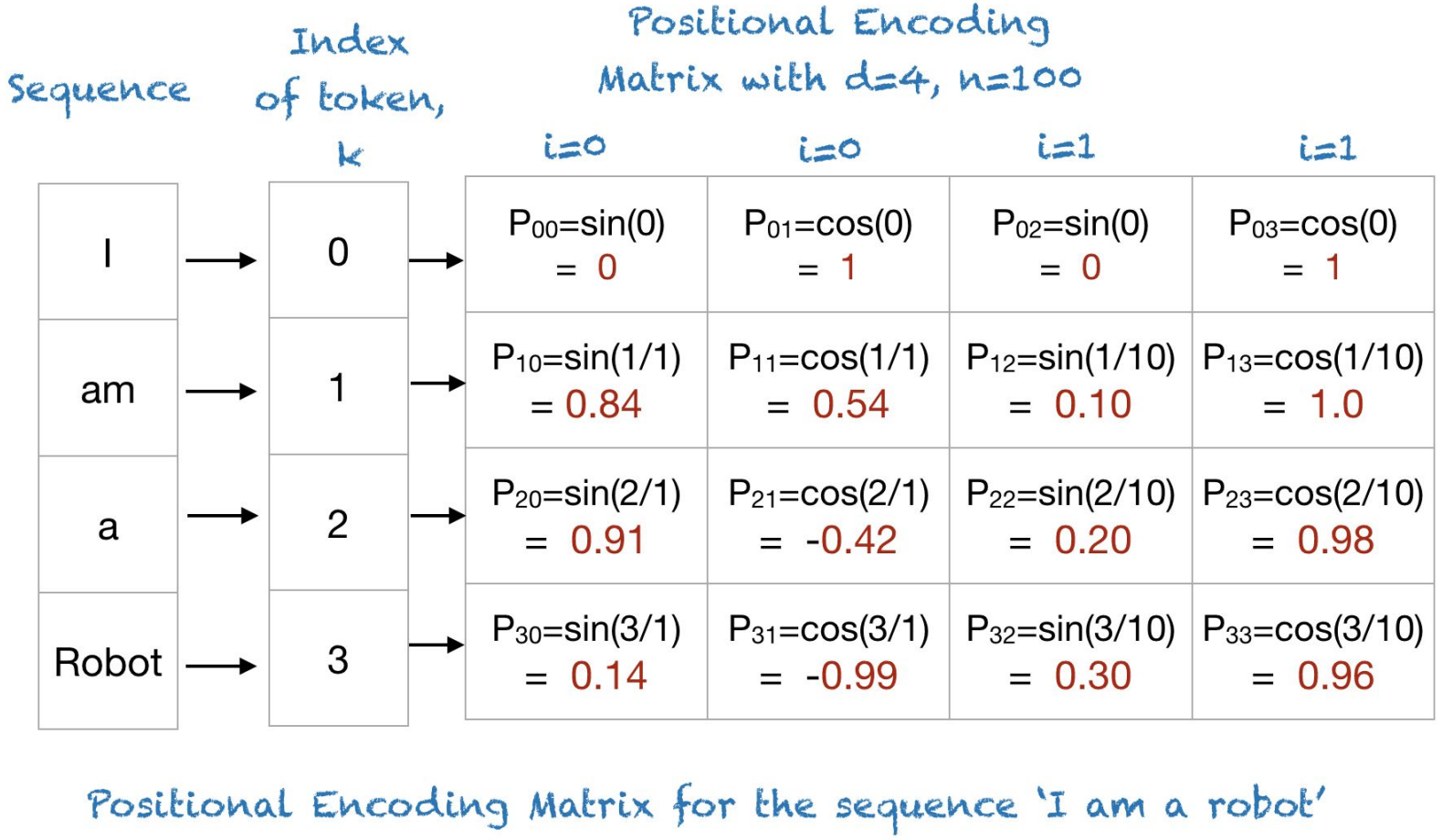
Positional Encoding

- Exemplo: **P == *PE***



Exemplo extraído de:
https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/

Positional Encoding



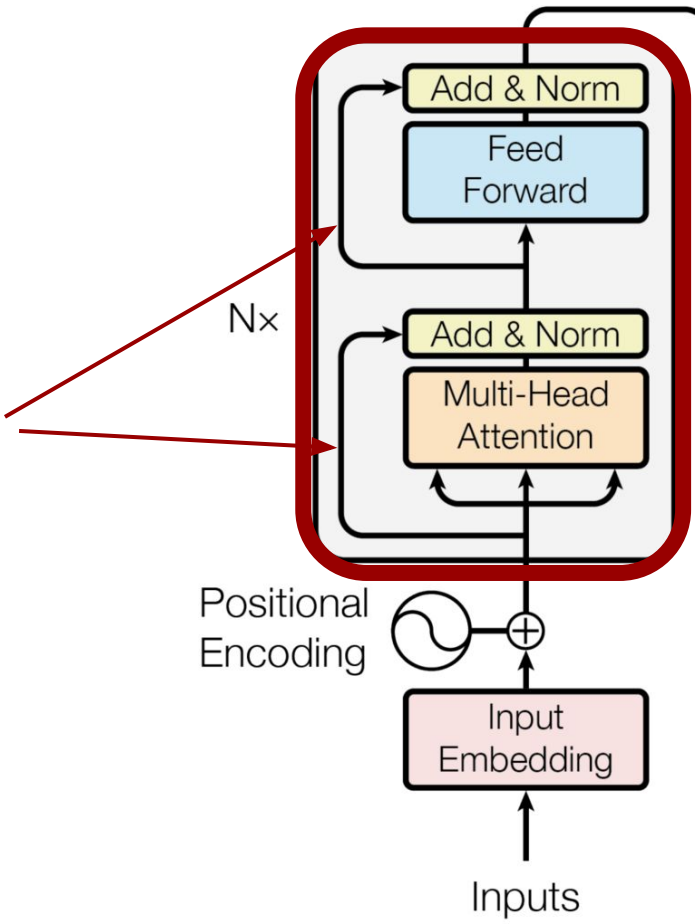
Positional Encoding Matrix for the sequence 'I am a robot'

Exemplo extraído de:
https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/

Encoder Block

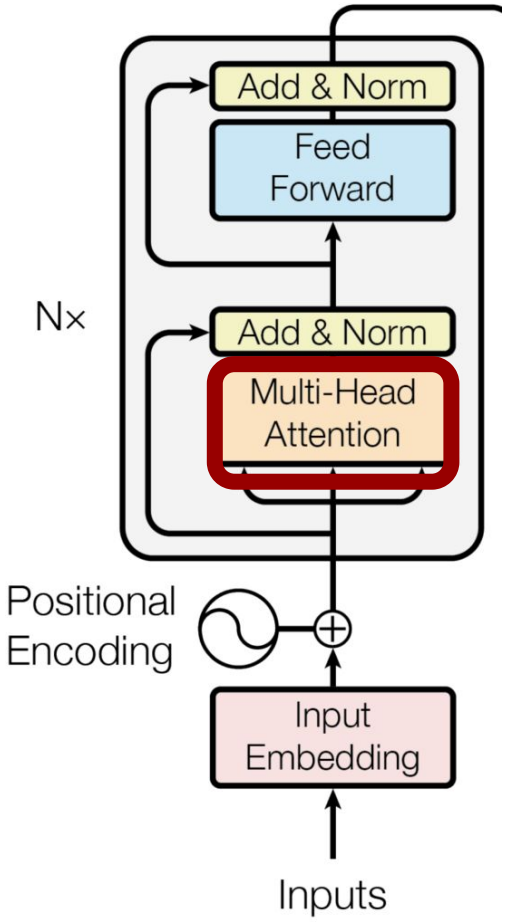
- Estruturado em 2 unidades principais:
  - **Multi-Head Attention**
  - **Feed Forward**

- Utiliza **skip connections** em cada unidade



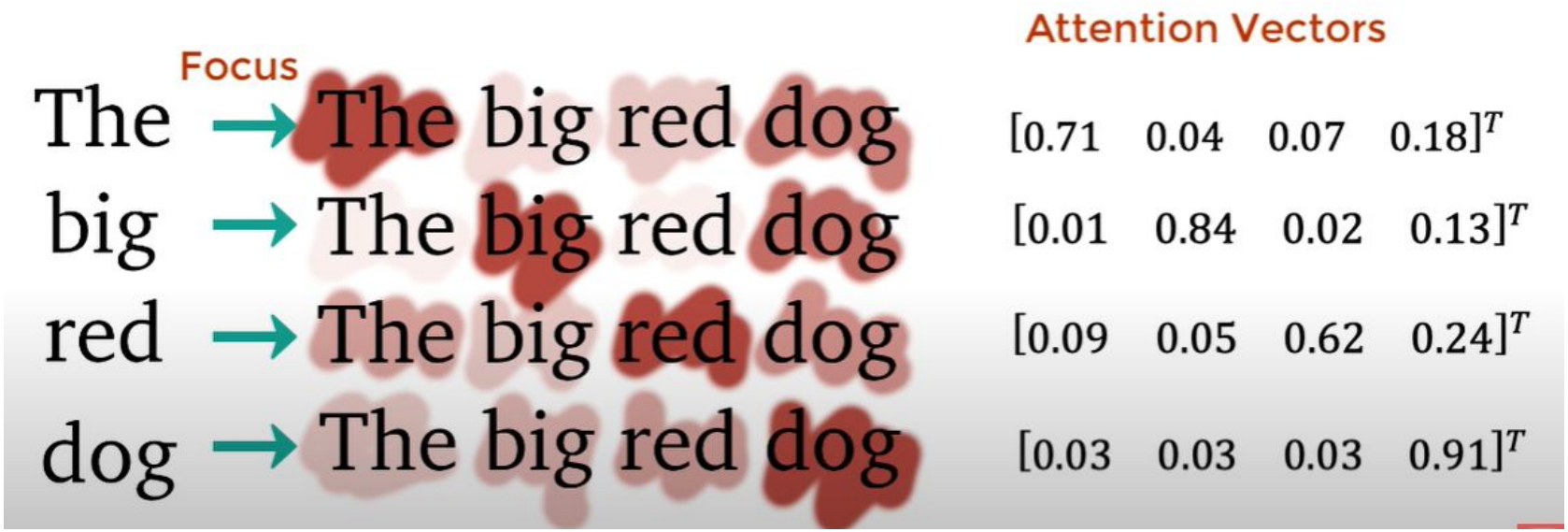
Encoder Block

- Estruturado em 2 unidades principais:
  - **Multi-Head Attention**
  - Feed Forward



Multi-Head Attention

- Utiliza o conceito de **Self-attention**
  - Aprende a relação entre os elementos da própria sentença.
  - Quão relevante é a i-ésima palavra da sentença, com relação às outras palavras da sentença?
  - Ajudam a capturar a estrutura sintática e contextual da sentença

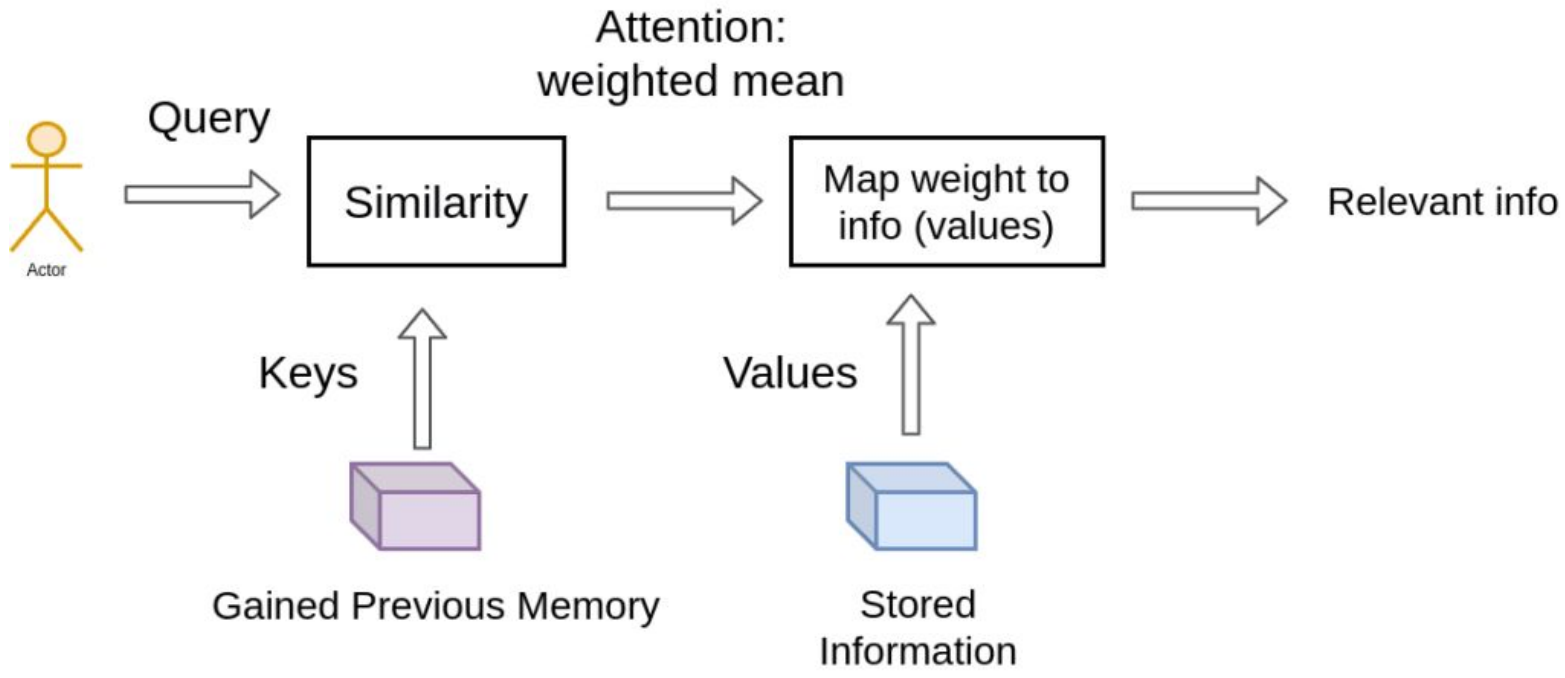


Self-Attention - Como funciona?

- Inspira-se no funcionamento de um sistema de recuperação de informações (consulta chave-valor)
  - Exemplo: Suponha a busca de um vídeo no youtube.
    - Quando você Pesquisa (**Query**) um vídeo específico, o mecanismo de pesquisa calcula uma similaridade entre a **Query** e um conjunto de chaves (**Keys**) (ex: título do vídeo, descrição, etc.) associado a possíveis vídeos armazenados.
    - Em seguida, o algoritmo apresenta os vídeos (Valores or **Values**) com as melhores correspondências.

Self-Attention - Como funciona?

- Inspira-se no funcionamento de um sistema de recuperação de informações (consulta chave-valor)

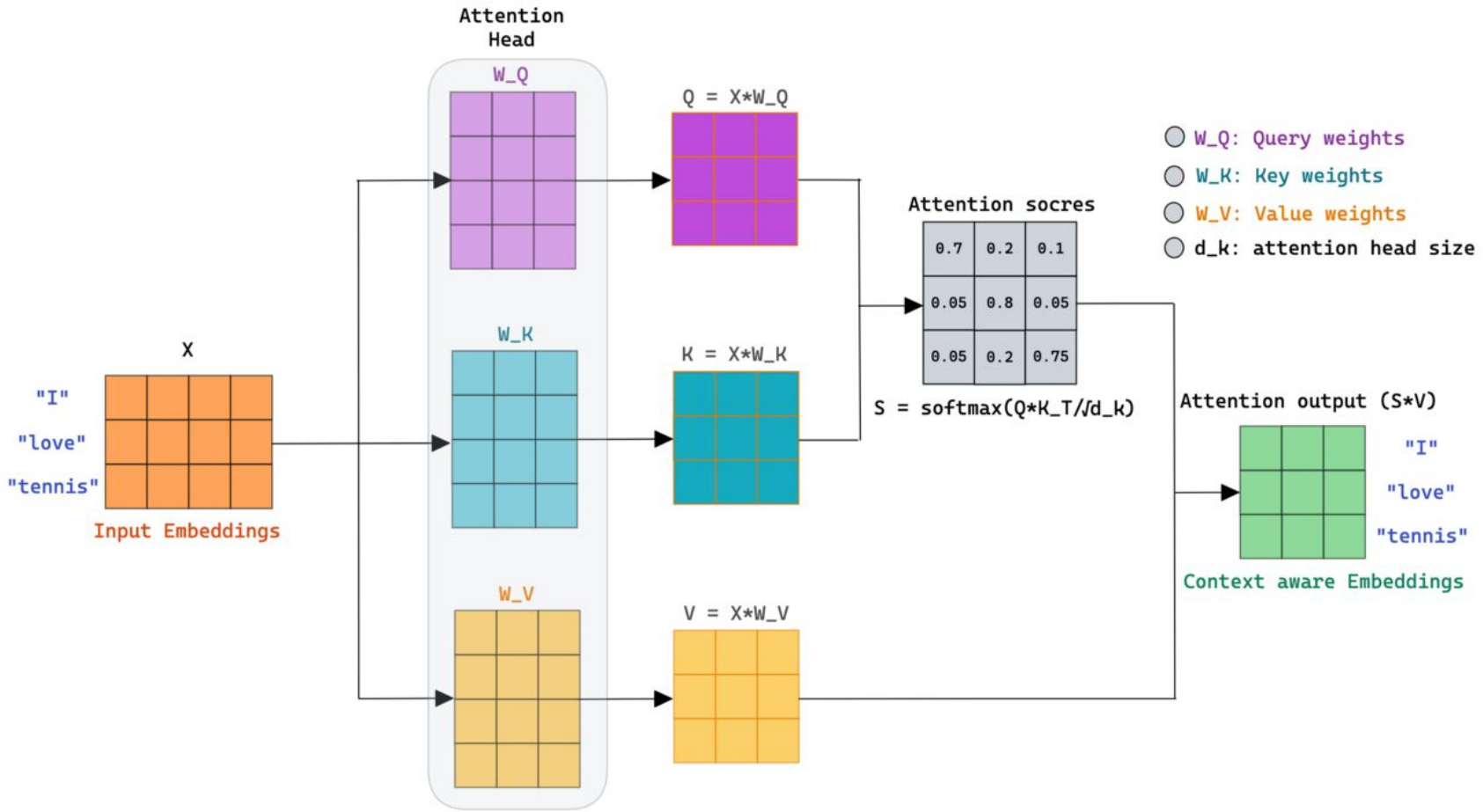


Self-Attention - Como funciona?

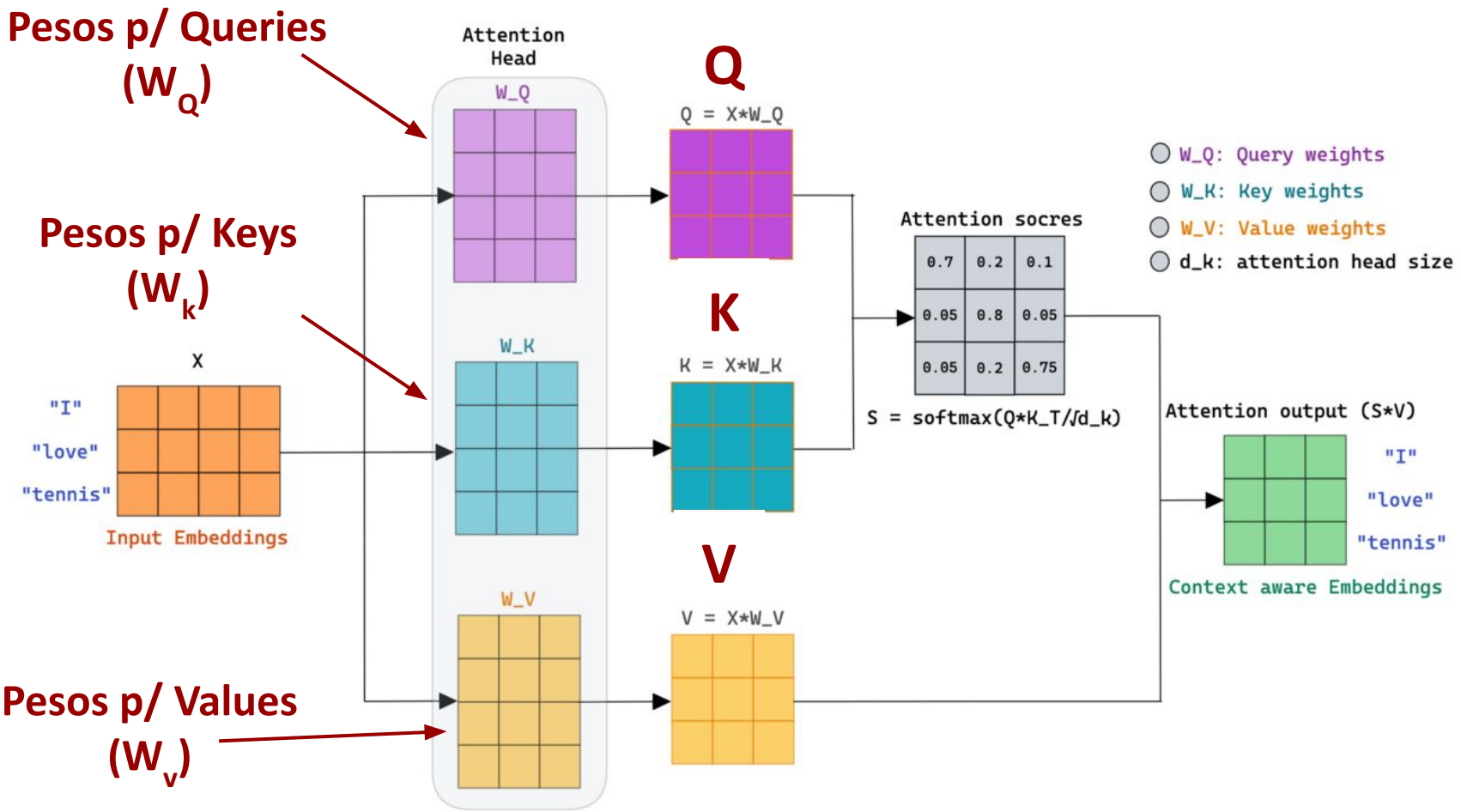
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d\_k}}\right)\mathbf{V}$$

- **Q** - Matriz de Querys; **K** - Matriz de Keys
- **V** - Matriz de Values
- **QK^T** - Contém um score de similaridade para para cada par <**query**, **key**>, baseado no **produto escalar**
- **$\sqrt{d\_k}$** - Reduz os scores de similaridade para evitar a saturação da função softmax

Self-Attention - Como funciona?

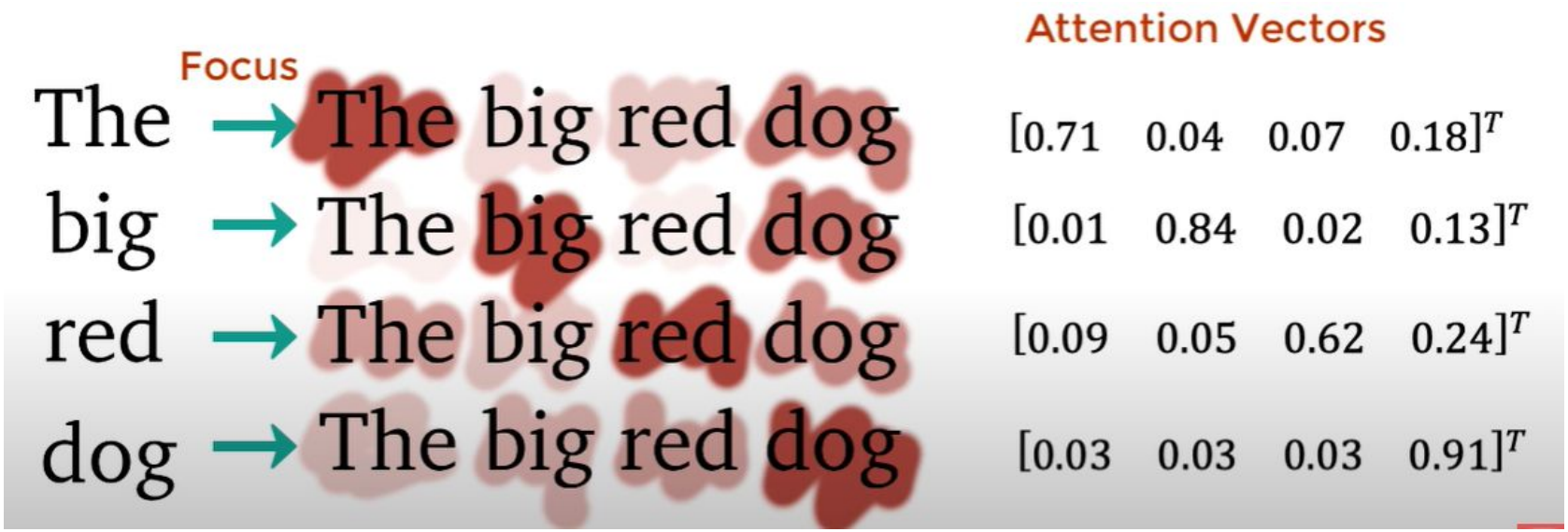


Self-Attention - Como funciona?



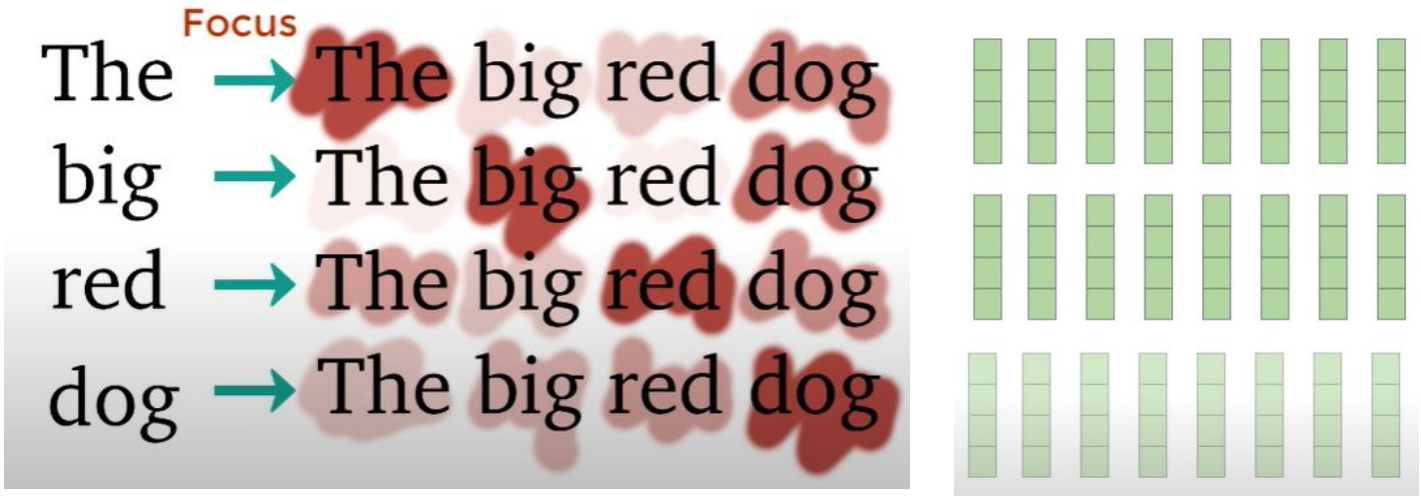
Self-Attention

- No Encoder, **Q**, **K** e **V** são iguais a lista de palavras na sentença
  - Assim cada palavra será comparada a todas as palavras na sentença, incluindo ela mesma



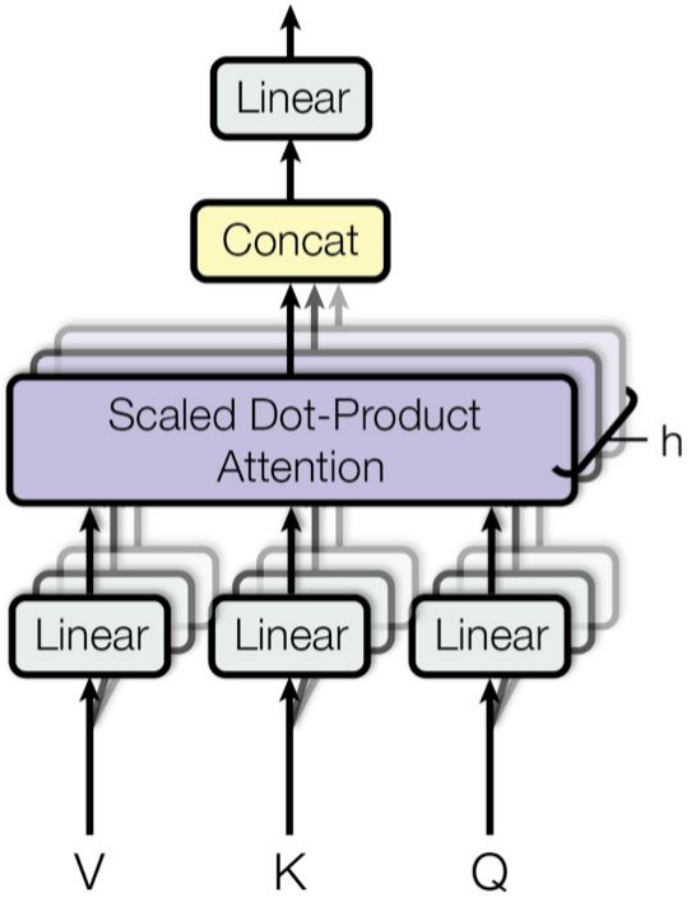
Multi-Head Attention

- Termo **multi-head attention** é usado porque são gerados **8 vetores de atenção para cada palavra**.
  - Esses 8 vetores são concatenados, formando um único vetor de dimensão maior.
  - Permite codificar múltiplas características diferentes da palavra

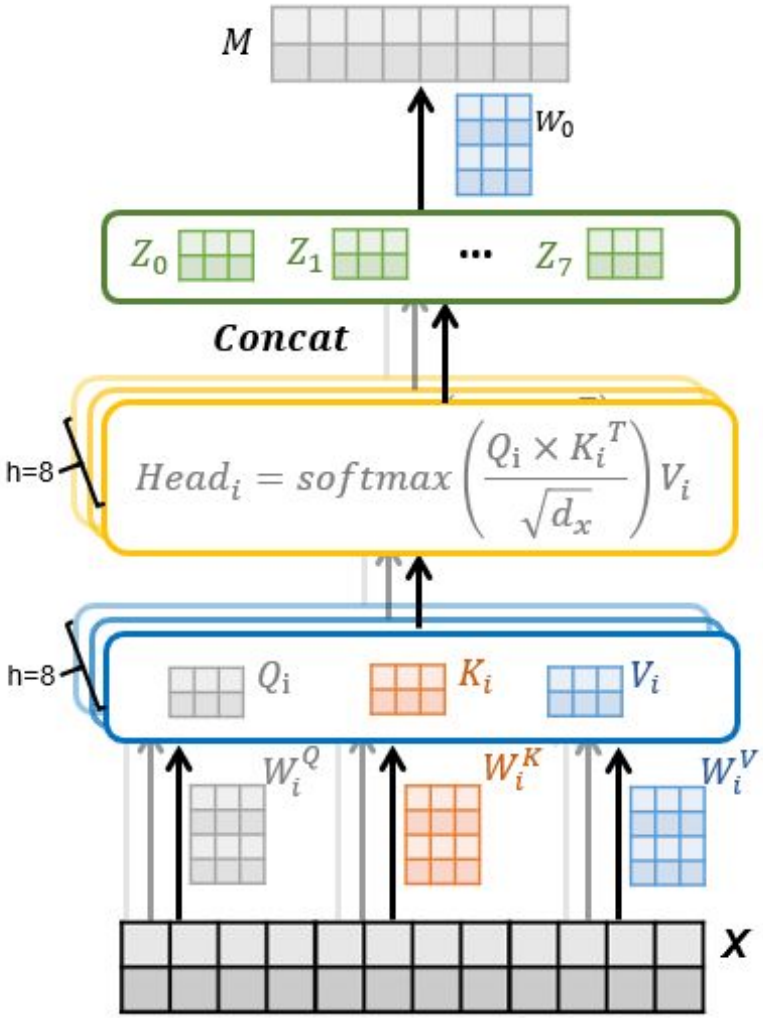


Multi-Head Attention

- Camada **multi-head attention** aplica diversas transformações lineares diferentes de Q, K e V
  - Possibilita que o modelo aplique projeções diferentes da representação da palavra em diferentes subespaços;
  - Ex: uma camada linear pode projetar a palavra com relação ao radical do verbo; outra camada pode projetar a palavra com relação ao tempo verbal (ex: passado, futuro).

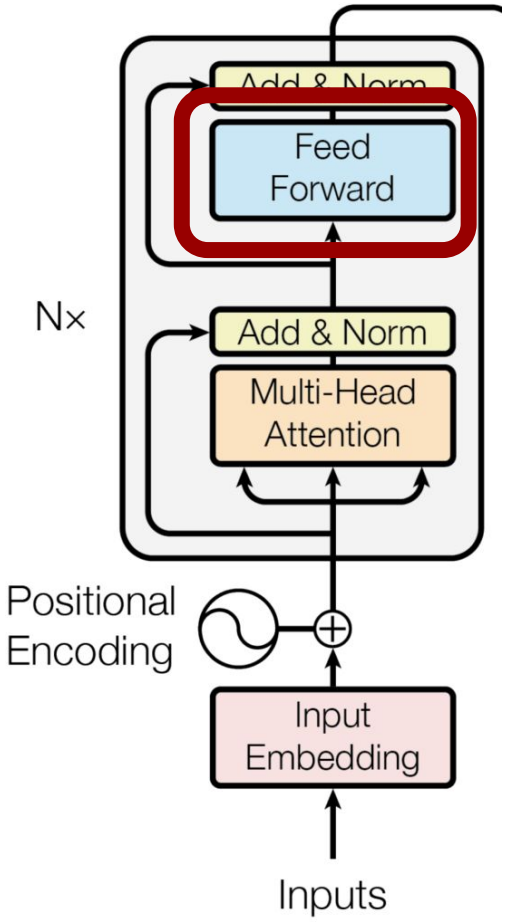


Multi-Head Attention



Encoder Block

- Estruturado em 2 unidades principais:
  - Multi-Head Attetion
  - **Feed-Forward**

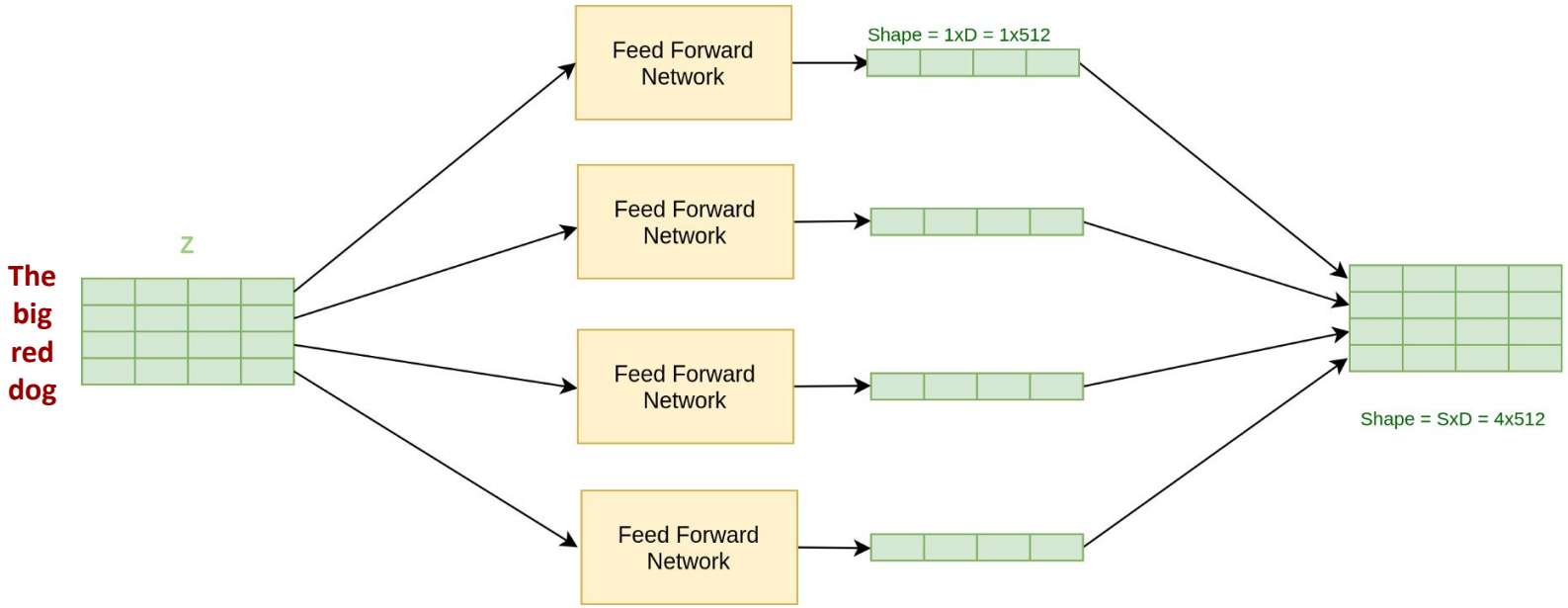


Feed Forward

- Aplica uma rede densa para o vetor de cada palavra;
- Rede densa com duas camadas de neurônios
  - A primeira camada usando função de ativação ReLU;
  - A segunda camada com função de ativação linear;
- Saída é um **conjunto de vetores**, um para cada palavra

Feed Forward

- Cada rede densa é independente da outra.
  - Com isso é possível **paralelizar**!
  - Passar todas as palavras ao mesmo tempo no bloco Encoder

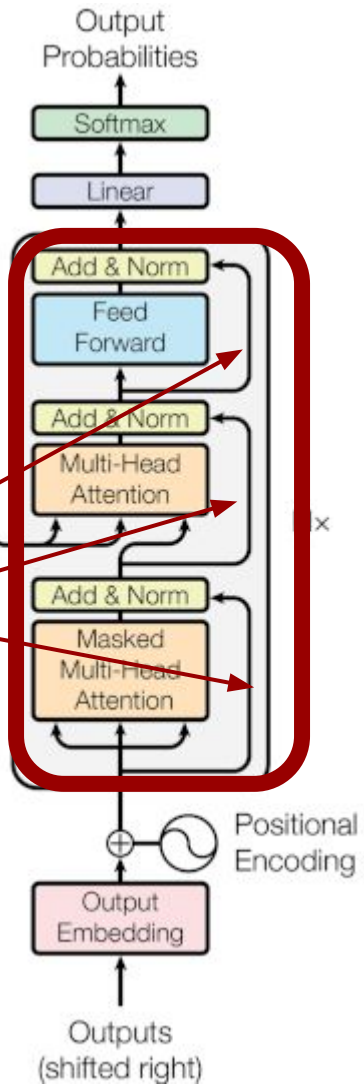


Decoder Block

- Estruturado em 3 unidades principais:
  - Masked Multi-Head Attention
  - Multi-Head Attention
  - Feed Forward

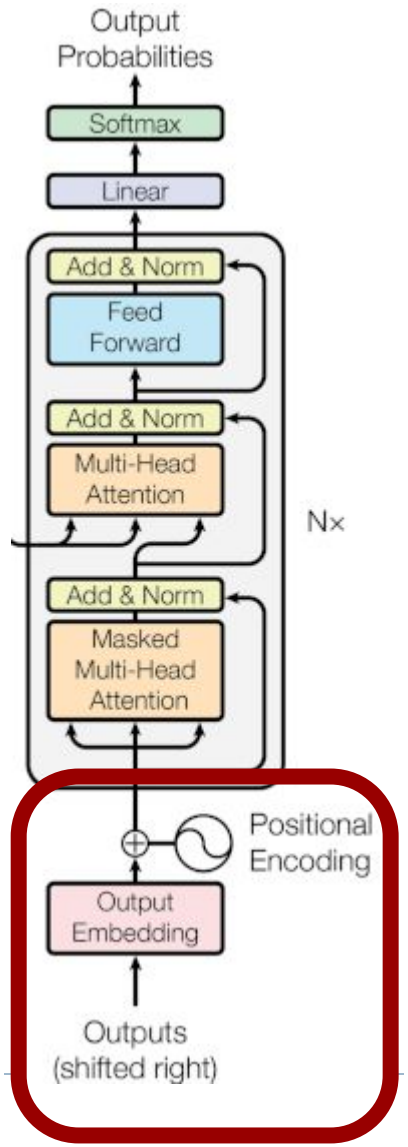
- Também utiliza **skip connections** em cada unidade

- Duas unidades semelhantes ao Encoder (Multi-Head Attention e Feed Forward)



Decoder Block

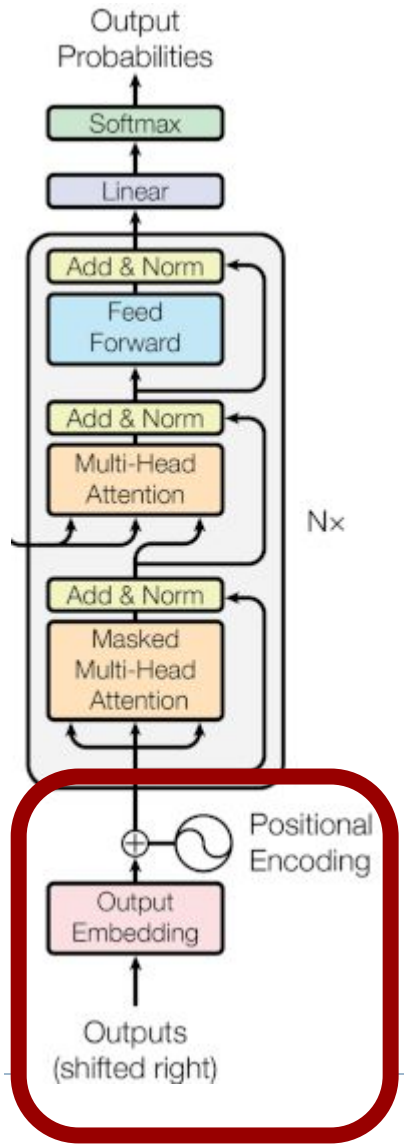
- Durante o treinamento, o Decoder é também alimentado com a "sentença em francês"



Decoder Block

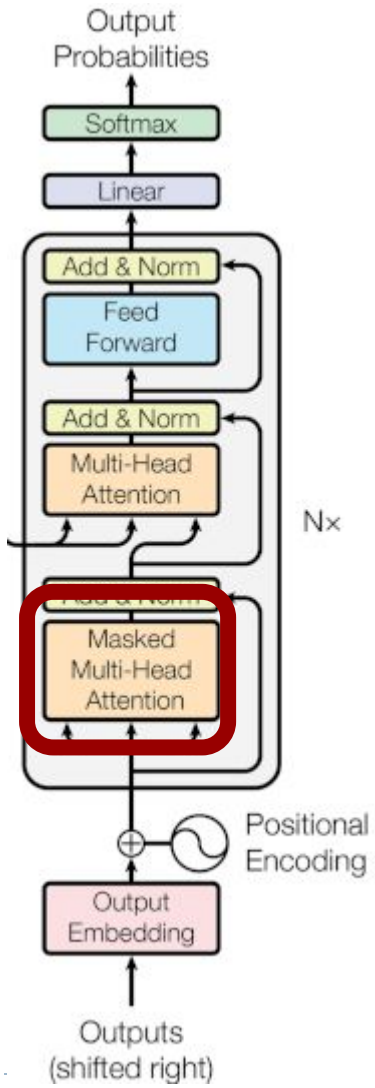
- Inicialmente, passa-se a sentença em francês pelo processo de **Embedding + Positional Encoding**
  - Gerando **vetores de embedding posicional** da sentença em Francês.

- Em seguida, esses vetores são encaminhados ao Decoder.



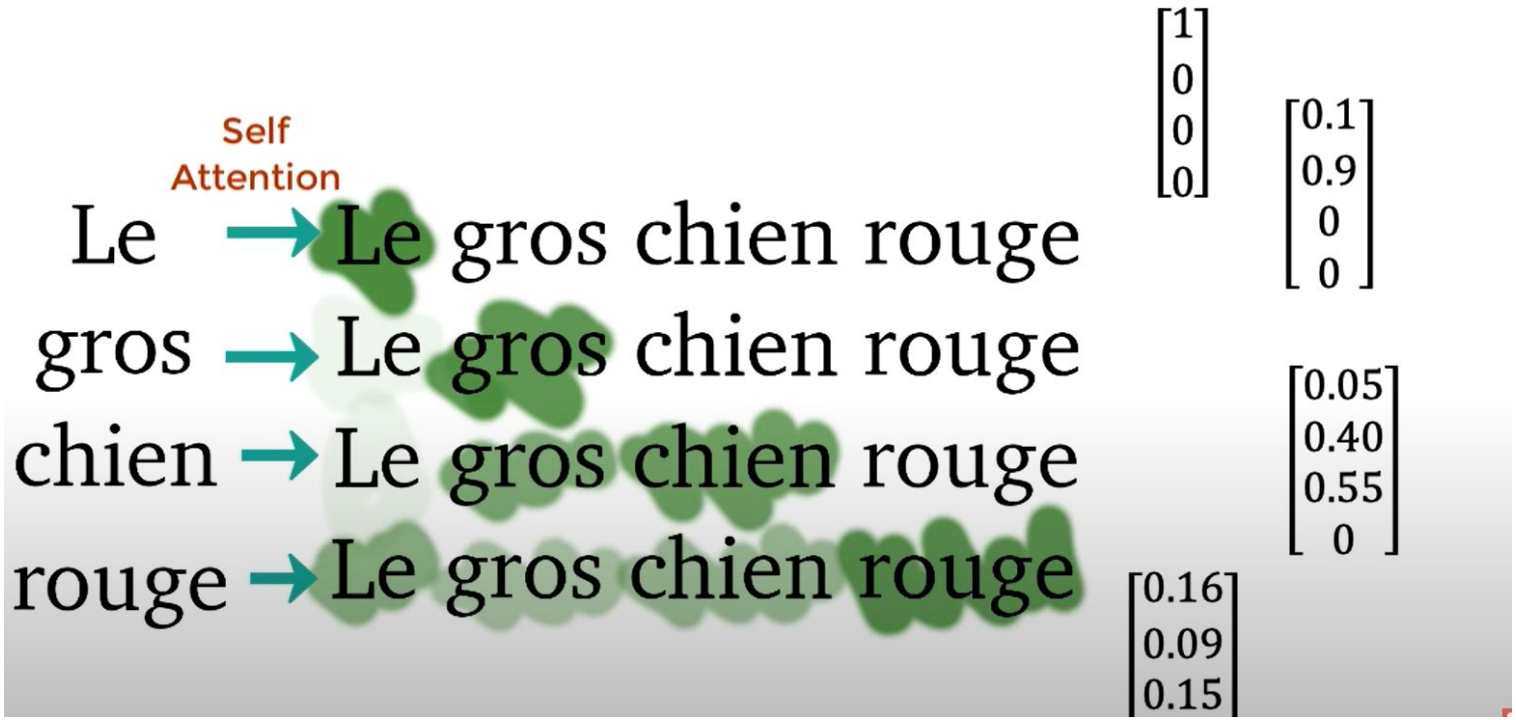
Decoder Block

- Estruturado em 3 unidades principais:
  - **Masked Multi-Head Attention**
  - Multi-Head Attention
  - Feed Forward



Masked Multi-Head Attention

- O **Masked Multi-Head Attention** (1o bloco de atenção) gera **vetores de atenção** para cada palavra em francês
  - **Similar ao Multi-Head Attention do Encoder**;

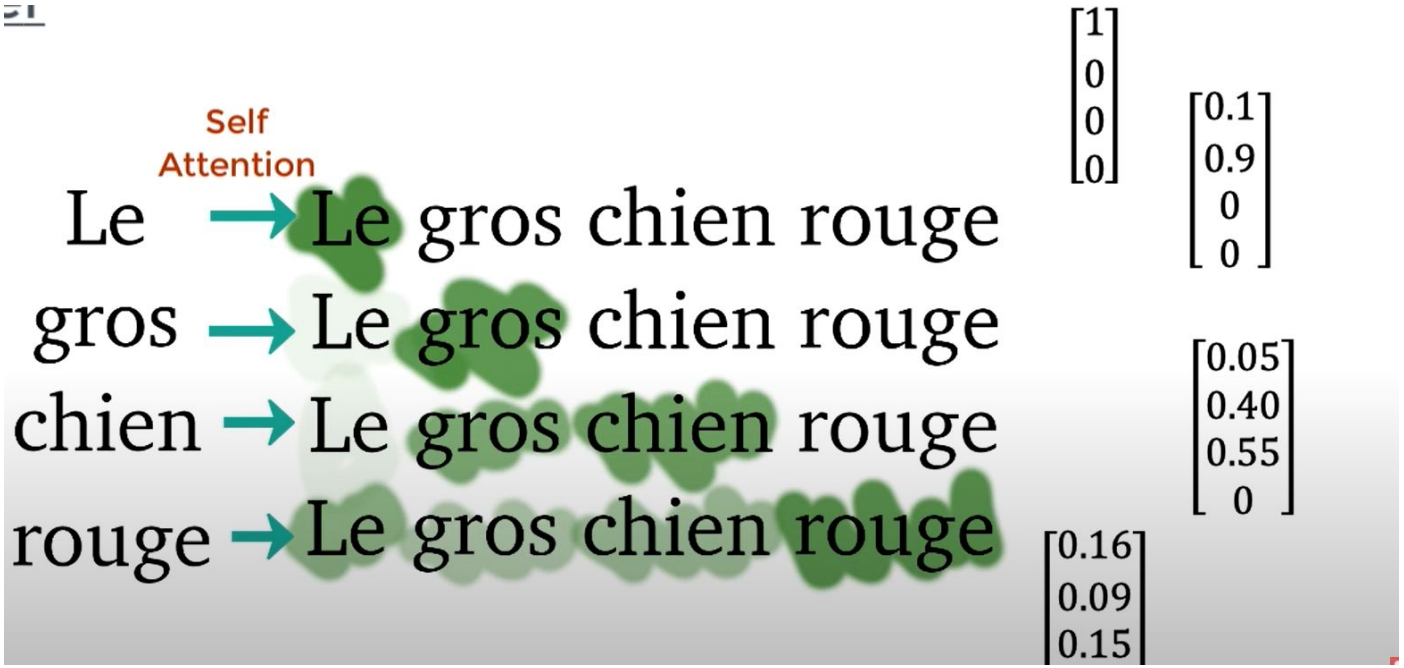


Masked Multi-Head Attention

- Porque ele é chamado de **Masked attention block?**
  - Enquanto tentamos prever a próxima palavra em francês, podemos usar **todas as palavras em inglês**, mas apenas **palavras anteriores em francês.**
  - Ex: para prever a 3a palavra da saída, podemos usar apenas a 1a e a 2a palavra em francês.
  - Para permitir paralelização, o bloco **mascara os vetores das palavras posteriores em francês com zeros**
    - Caso contrário a rede de atenção não poderia utilizá-la.

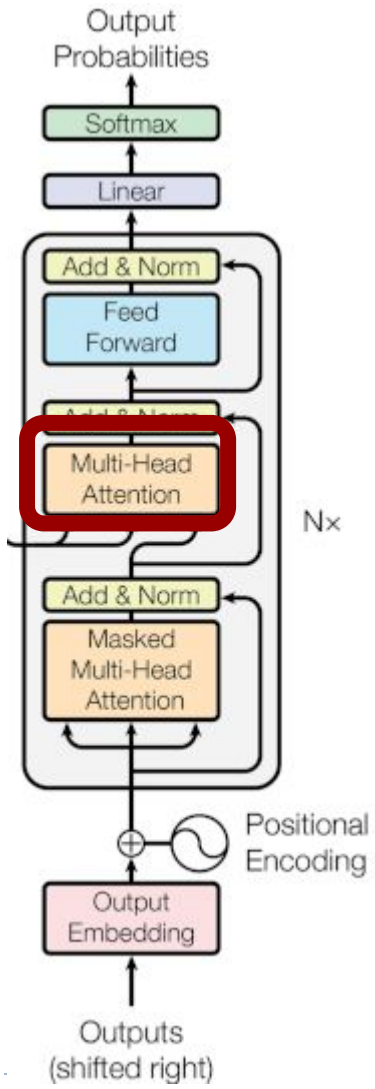
Masked Multi-Head Attention

- Nesse caso, **Q**, **K** e **V** são iguais a lista de palavras na sentença-alvo (ex: francês), mas usando a máscara
  - Para evitar que as palavras se comparem com palavras localizadas depois dela



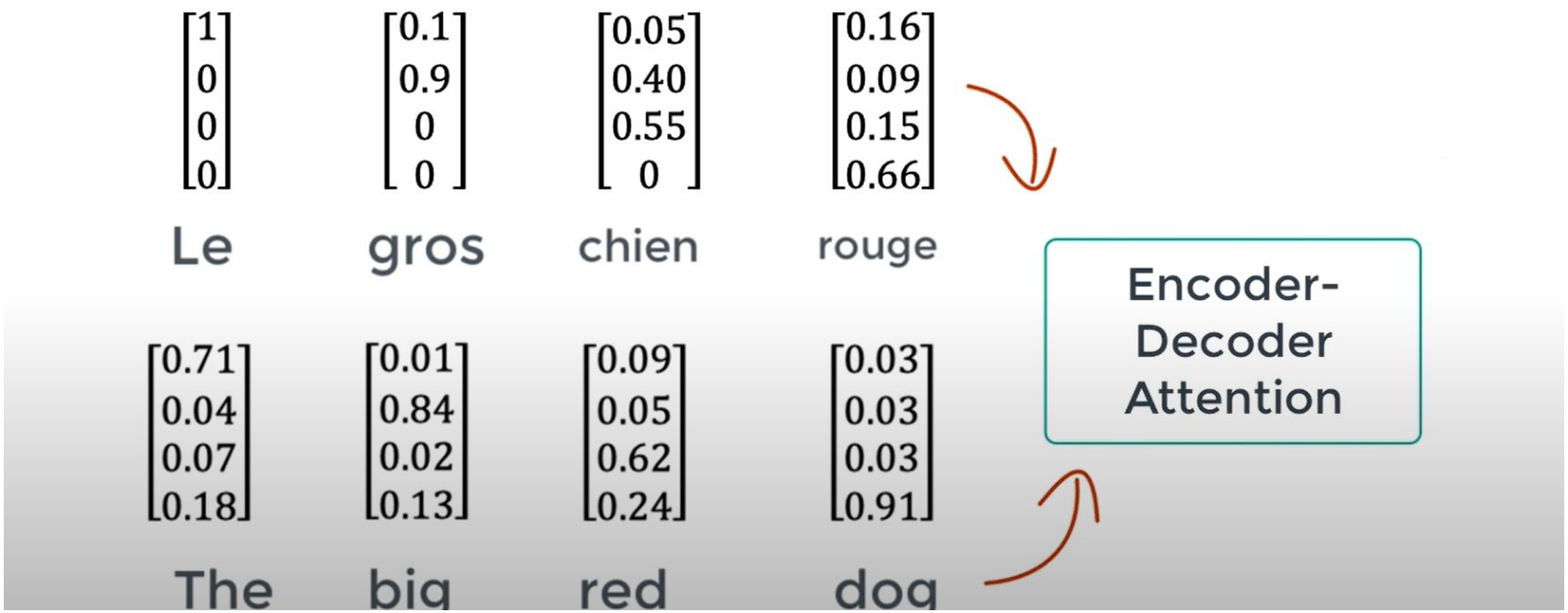
Decoder Block

- Estruturado em 3 unidades principais:
  - Masked Multi-Head Attention
  - **Multi-Head Attention**
  - Feed Forward



Multi-Head Attention
(2o Bloco de Atenção do Decoder)

- Tenta determinar **quanto os vetores de atenção inglês-francês estão relacionados** entre si.



Multi-Head Attention
(2o Bloco de Atenção do Decoder)

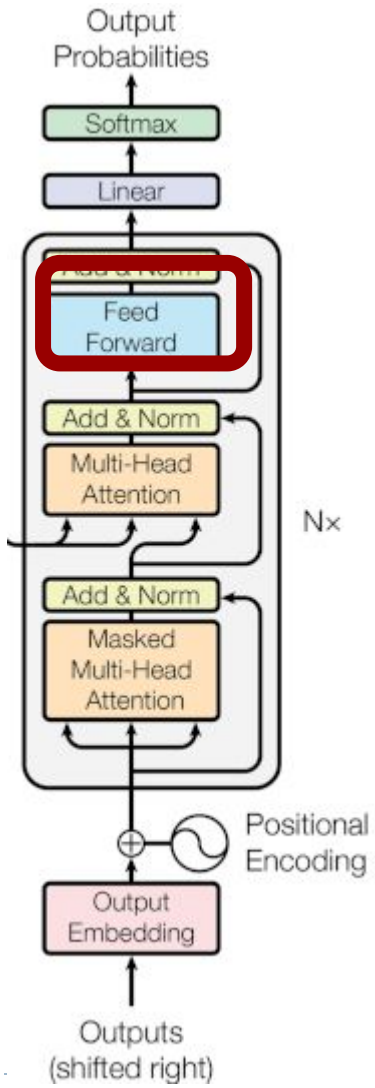
- Realiza o mapeamento entre palavras em inglês e francês

- Nesta unidade:
  - **Q** representa a codificação de palavras geradas pelo **Decoder**
  - **K** e **V** representam a codificação de palavras geradas pelo **Encoder**

- Cada vetor da saída representa a relação com outras palavras em ambas as línguas.

Decoder Block

- Estruturado em 3 unidades principais:
  - Masked Multi-Head Attention
  - Multi-Head Attention
  - **Feed-forward**



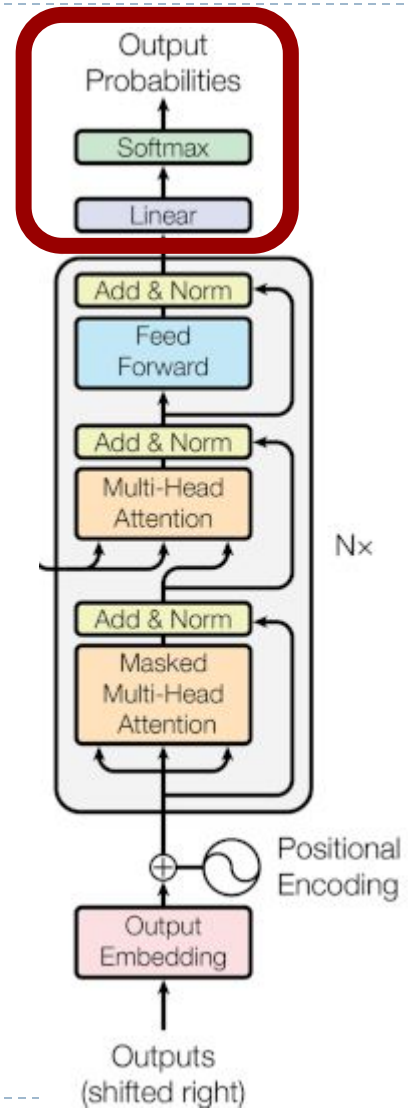
Feed Forward

- Similar ao bloco Feed Forward do Encoder

- Rede densa com duas camadas de neurônios
  - A primeira camada usando função de ativação ReLU;
  - A segunda camada com função de ativação linear;

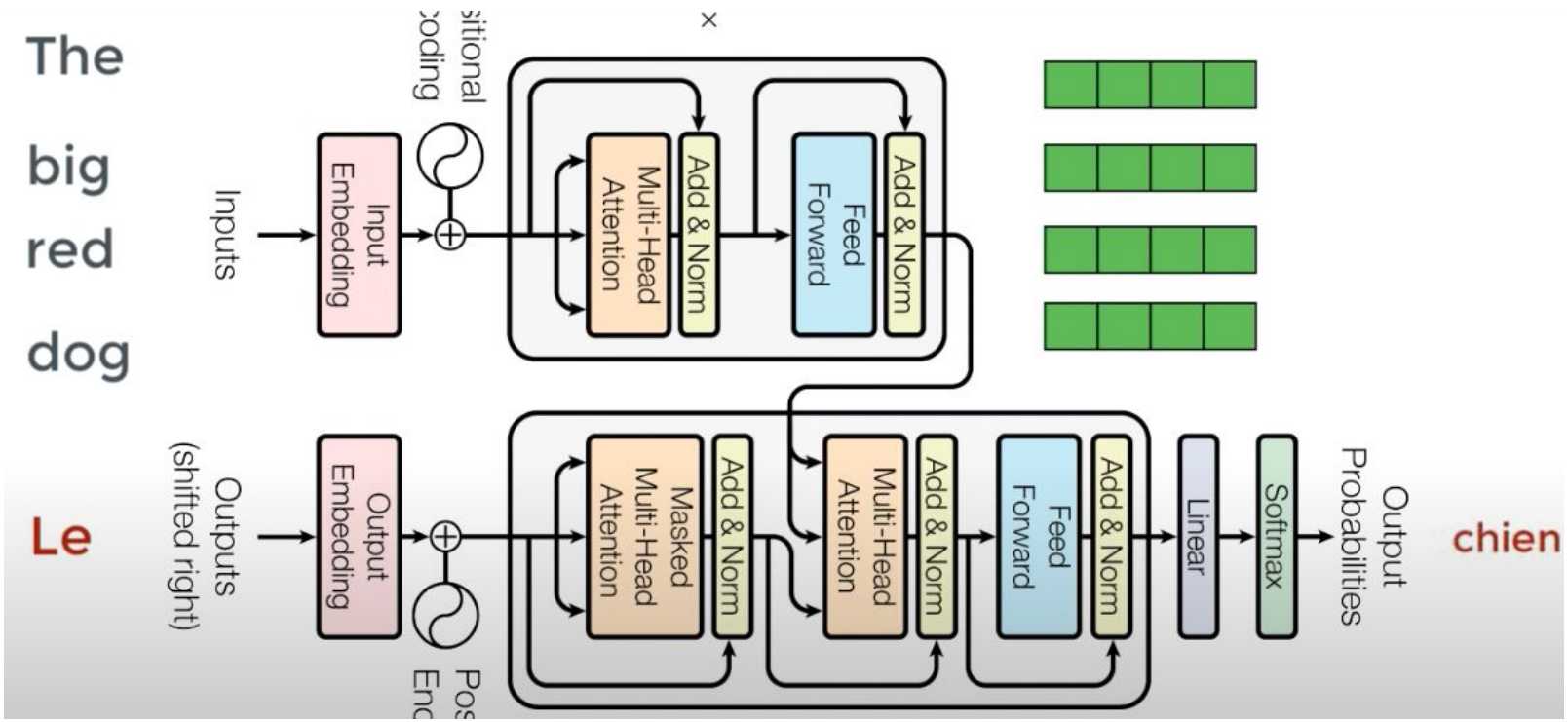
Camada Linear + Softmax

- A parte final da rede possui uma camada **Linear** com a função de ativação **softmax**
  - Transforma a saída da camada Linear em uma distribuição de probabilidade
  - Palavra gerada na saída é aquela que possui a probabilidade mais alta.



Decoder Block

- Geralmente, o **<u>Decoder prediz a próxima palavra</u>**
  - Ele é executado em múltiplos passos até o token de fim de sentença ser gerado.



Universidade Federal da Paraíba

Centro de Informática

Departamento de Informática

Aprendizado Profundo

Redes Neurais Transformers

(Material Baseado em @CodeEmporium)

Tiago Maritan

(tiago@ci.ufpb.br)