Multi-Head Attention

Tópicos em Ciência de Dados

Pontifícia Universidade Católica de Campinas

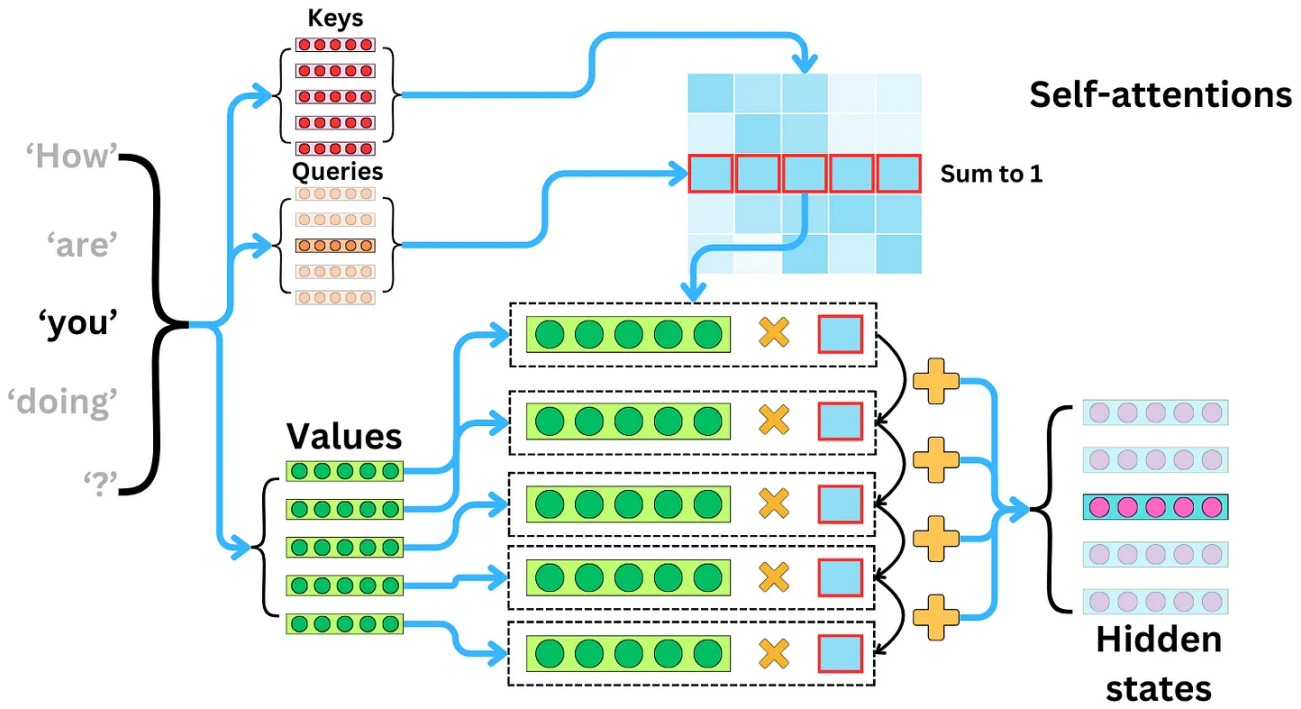
Prof. Dr. Denis M. L. Martins

Objetivos de Aprendizagem

- Compreender a motivação por trás da **multi-head** versus **single-head** attention.
- Compreender matematicamente o mecanismo MHA.
- Analisar exemplos práticos em LLMs (BERT, GPT-3).

Revisão Rápida: Self-Attention

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{Q\,K^\top}{\sqrt{d_k}}\right) V$$

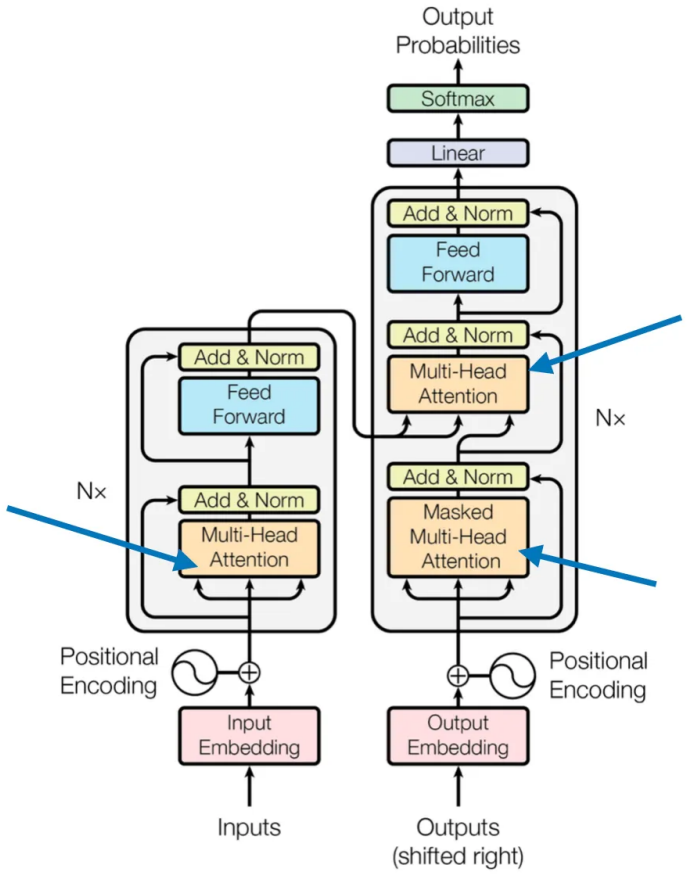


Self-Attention. Fonte: The AI Edge.

- **Limitação**: apenas **uma** "perspectiva" de atenção (um único conjunto de Q, K e V).

Multi-Head Attention: Motivação

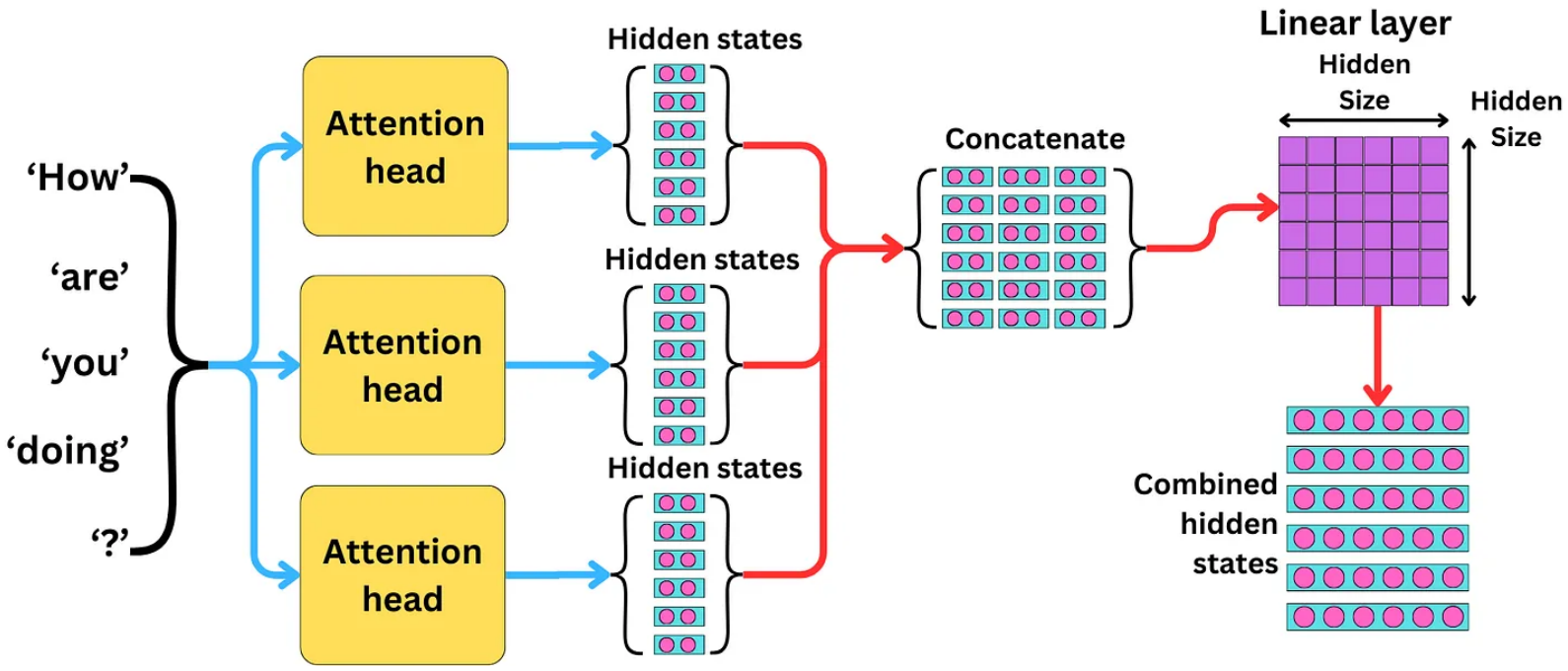
- **Expressividade**: Cada cabeça de atenção pode focar em padrões diferentes (ex.: sintaxe, semântica, dependências a longo prazo).
- **Robustez**: Diversificação de representações reduz risco de over-fitting a um único padrão.
- **Parallelismo**: Cabeças independentes podem ser computadas simultaneamente em GPUs/ TPUs.



Blocos de Multi-Head Attention no transformer original.
Fonte: https://arxiv.org/abs/1706.03762.

Multi-Head Attention (MHA)

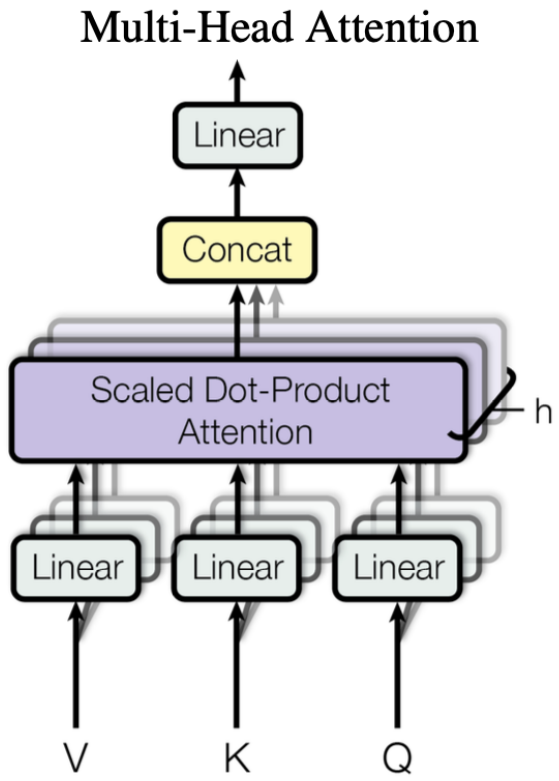
Semelhante ao uso de múltiplos kernels em CNNs para gerar diferentes feature maps.



Multi-Head Attention. Fonte: The AI Edge.

Multi-Head Attention: Formulação Matemática

- Computar Queries: $Q^{(h)} = XW_Q^{(h)}$
- Computar Keys: $K^{(h)} = XW_K^{(h)}$
- Computar Values: $V^{(h)} = XW_V^{(h)}$
- $A^{(h)} = \text{Softmax}\left(\frac{Q^{(h)}K^{(h)\top}}{\sqrt{d_k}}\right)V^{(h)}$
- Concatenar o output de todas as cabeças: $\hat{A} = \text{Concat}(A^{(1)}, \ldots, A^{(H)})$
- Calcular a projeção final: $O = \hat{A}W_O + b$
  - H: número de cabeças.
  - $d_k = d_v = d_{\text{model}}/H$.



Multi-Head Attention. Fonte: Jeremy Jordan.

Implementação simples: Tokenização

```python
import numpy as np

# Exemplos de embeddings para 4 palavras usando One-Hot Encoding
word_embeddings = {
    'the': np.array([1, 0, 0]),
    'cat': np.array([0, 1, 0]),
    'sat': np.array([0, 0, 1]),
    'on': np.array([1, 1, 0])
}

# Converte palavras em embeddings
def get_embedding(word):
    return word_embeddings.get(word)

# Calcula os embeddings
sentence = ['the', 'cat', 'sat', 'on']
embeddings = np.array([get_embedding(word) for word in sentence])
print("Embeddings:\n", embeddings)
```

> Fonte: https://kostyanuman.substack.com/p/understanding-the-attention-mechanism.

Implementação simples: Self-Attention

```python
def softmax(x):
    return np.exp(x) / np.sum(np.exp(x), axis=0)

def self_attention(embeddings):
    # Calcula o produto escalar dos embeddings
    scores = np.dot(embeddings, embeddings.T)

    # Aplica a softmax para obter os pesos de atenção
    attention_weights = softmax(scores)

    # Calcula a soma ponderada dos embeddings
    output = np.dot(attention_weights, embeddings)

    return output, attention_weights

# Calcula a auto-atenção
output, attention_weights = self_attention(embeddings)
print("Pesos de Atenção:\n", attention_weights)
print("Saída da Auto-Atenção:\n", output)
```

> Fonte: https://kostyanuman.substack.com/p/understanding-the-attention-mechanism.

Implementação simples: Multi-Head Attention

```
def multi_head_attention(embeddings, num_heads=2):
    head_outputs = []

    for _ in range(num_heads):
        output, _ = self_attention(embeddings)
        head_outputs.append(output)

    # Concatena as saídas de todas as cabeças
    return np.concatenate(head_outputs)

# Calcula a atenção multi-cabeça
multi_head_output = multi_head_attention(embeddings)
print("Saída da Atenção Multi-Cabeça:\n", multi_head_output)
```

> Fonte: https://kostyanuman.substack.com/p/understanding-the-attention-mechanism.

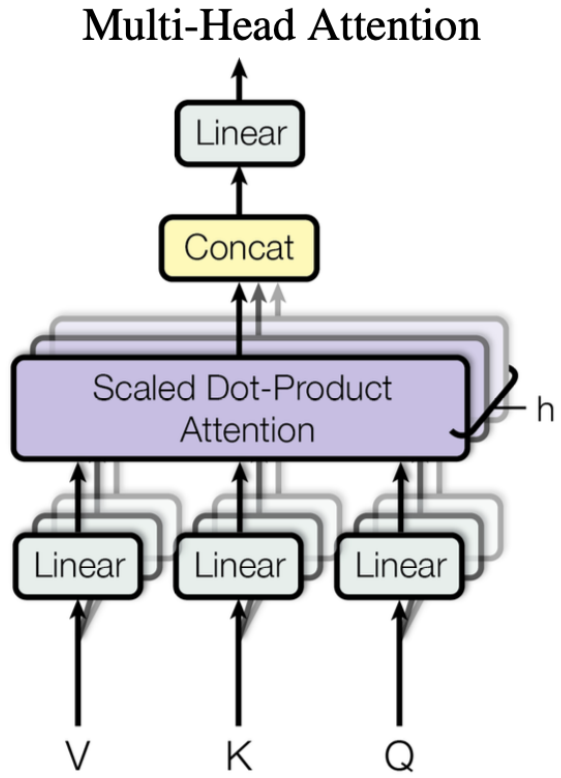
Implementação simples: Saída do Bloco de MHA

```
def feedforward_network(x):
    # Rede feed-forward simples com uma camada oculta
    # Pesos da camada oculta
    W1 = np.random.rand(x.shape[1], x.shape[1] * 2)
    # Viéses da camada oculta
    b1 = np.random.rand(x.shape[1] * 2)
    # Pesos da camada de saída
    W2 = np.random.rand(x.shape[1] * 2, x.shape[1])
    # Viéses da camada de saída
    b2 = np.random.rand(x.shape[1])

    # Ativação ReLU
    hidden_layer = np.maximum(0, np.dot(x, W1) + b1)
    output_layer = np.dot(hidden_layer, W2) + b2

    return output_layer

# Passa as features de MHA pela rede (FC)
final_output = feedforward_network(multi_head_output)
print("Saída Final:\n", final_output)
```

> Fonte: https://kostyanuman.substack.com/p/understanding-the-attention-mechanism.

Resumo e Próximos Passos

- Multi-Head Attention (MHA)
  - Peça-chave na capacidade dos Transformers de capturar relações entre palavras de forma eficiente e paralela.
  - Projeções para Q, K, V em H cabeças.
  - Cada cabeça calcula atenção independentemente.
  - Resultados concatenados → projeção final W_O.
  - As múltiplas cabeças ampliam a expressividade do modelo, permitindo diferentes focos simultâneos.
- Atenção Causal (Masked): Máscara triangular inferior impede acesso a tokens futuros.
- Interpretação Visual: Heat-maps para comparação entre cabeças locais vs globais.



Multi-Head Attention. Fonte: Jeremy Jordan.

Perguntas e Discussão

- Qual a motivação por trás do fator $\sqrt{d_k}$ na fórmula de atenção? Como ele influencia a distribuição dos logits antes da softmax?
- Como o Multi-Head Attention aumenta a capacidade expressiva de um modelo comparado ao Self-Attention? Quais tipos de dependências cada cabeça tende a capturar?
- Como a atenção pode ser interpretada como um mecanismo de explicabilidade em LLMs?
- Em cenários de poucos dados, como a atenção pode ser ajustada para evitar overfitting? Discuta estratégias de regularização.
- Qual o papel da MHA na transferência de conhecimento entre tarefas em um LLM pré-treinado?
- Quando reduzir o número de cabeças pode ser benéfico?