"Any A.I. smart enough to pass a Turing test is smart enough to know to fail it" (Ian McDonald).

Large Language Models

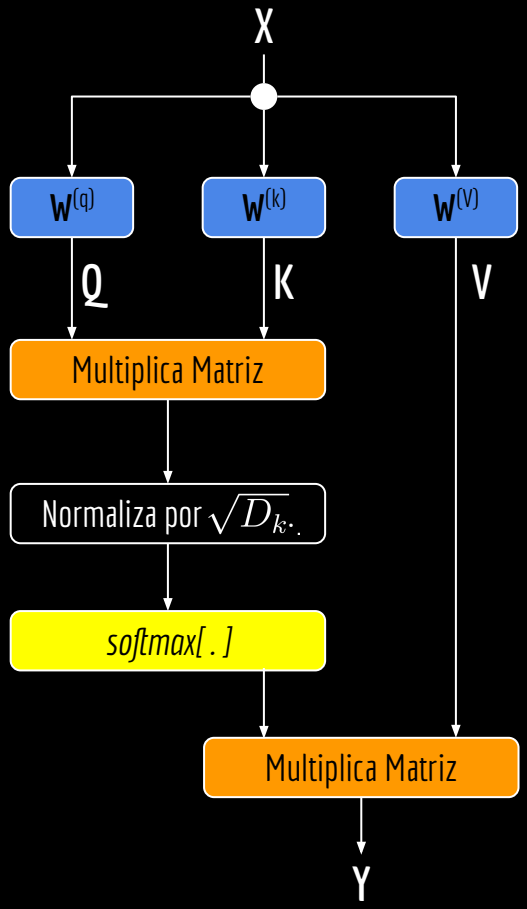
Paulo Ricardo Lisboa de Almeida

DSBD

UFPR

Self-Attention Head

$$Y = softmax\left[\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{D_k}}\right]\mathbf{V}$$



Múltiplas cabeças

Da mesma forma que com redes convolucionais onde é útil ter multiplos filtros, em uma rede baseada em transformers, é útil ter múltiplas cabeças.

Com *H* cabeças, concatenamos as respostas das cabeças, na forma:

$$\mathbf{H}_h = Atencao(\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h) = softmax\left[\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{D_k}}\right]\mathbf{V}$$

$$\mathbf{Y} = Concatenar[\mathbf{H}_1, \ldots, \mathbf{H}_H]\mathbf{W}^{(o)}$$

Múltiplas cabeças

$$\mathbf{Y} = Concatenar[\mathbf{H}_1, \ldots, \mathbf{H}_H]\mathbf{W}^{(o)}$$

Cada cabeça gera uma saída $\mathbf{H}_i$ de dimensão $N \times D_V$.

Concatenadas as saídas, temos uma matriz de $N \times HD_V$.

Múltiplas cabeças

$$\mathbf{Y} = Concatenar[\mathbf{H}_1, \ldots, \mathbf{H}_H]\mathbf{W}^{(o)}$$

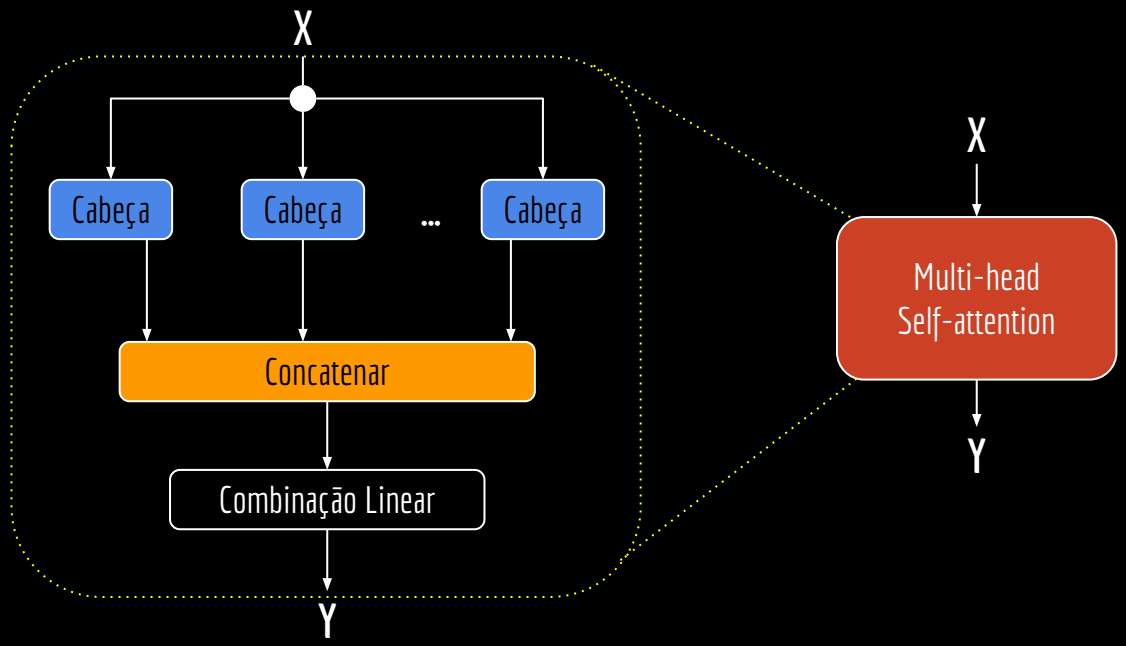
Cada cabeça gera uma saída $\mathbf{H}_i$ de dimensão $N \times D_V$.

Concatenadas as saídas, temos uma matriz de $N \times HD_V$.

A matriz $\mathbf{W}^{(o)}$ comumente tem dimensões $HD_V \times D_V$ para que o resultado final tenha dimensão $N \times D_V$.

Os pesos de $\mathbf{W}^{(o)}$ são calculados durante o treinamento.

Multi-head Self-attention



Faça você mesmo

Descubra (usando o código do Google Colab) quantas cabeças a versão da rede Llama 3.2 que estamos usando possui.

Múltiplas Camadas

Ideia: transformar o conjunto de múltiplas cabeças visto anteriormente em uma camada de uma rede.

Criar um rede profunda, composta de múltiplas dessas camadas.

Múltiplas Camadas

Primeiro, para facilitar a convergência vamos:

Adicionar conexões residuais.

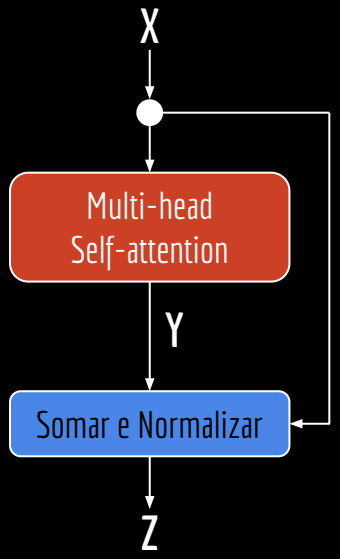
- Ideia: fazer um bypass dos dados da camada diretamente para sua saída.
  - Ideia usada em outros tipos de redes, como CNNs. Veja na literatura.

Normalizar a camada.

- Vamos usar *Layer Normalization*.
  - Ideia similar ao batch normalization de aulas passadas. Veja na literatura.

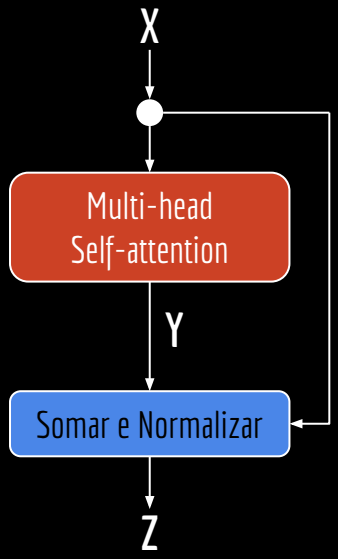
Múltiplas Camadas

$$\mathbf{Z} = LayerNorm[\mathbf{Y} + \mathbf{X}]$$



Múltiplas Camadas

A única não linearidade até então é dada pela função *softmax[.]* aplicada na *multi-head self-attention*.



Múltiplas Camadas

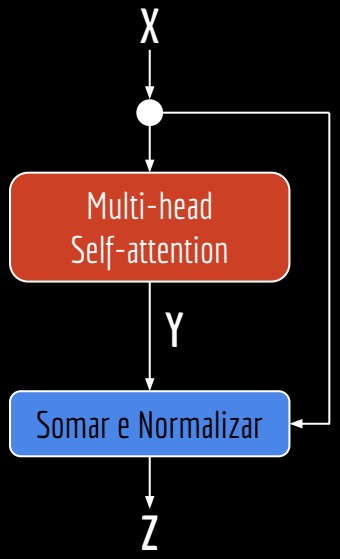
A única não linearidade até então é dada pela função *softmax[.]* aplicada na *multi-head self-attention*.

Podemos adicionar um MLP na saída **Z** para aumentar a flexibilidade da rede.

O *MLP[.]* deve preservar as dimensões dos dados.

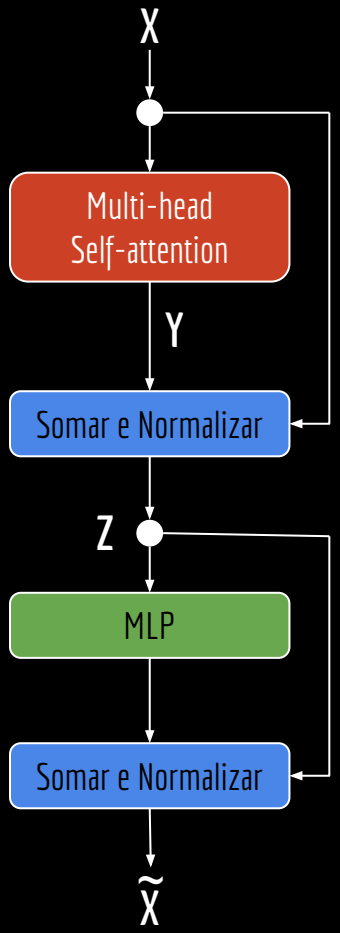
Entrada de tamanho *D*, e saída de tamanho *D*.

Incorporamos ainda conexões residuais e normalização no MLP.

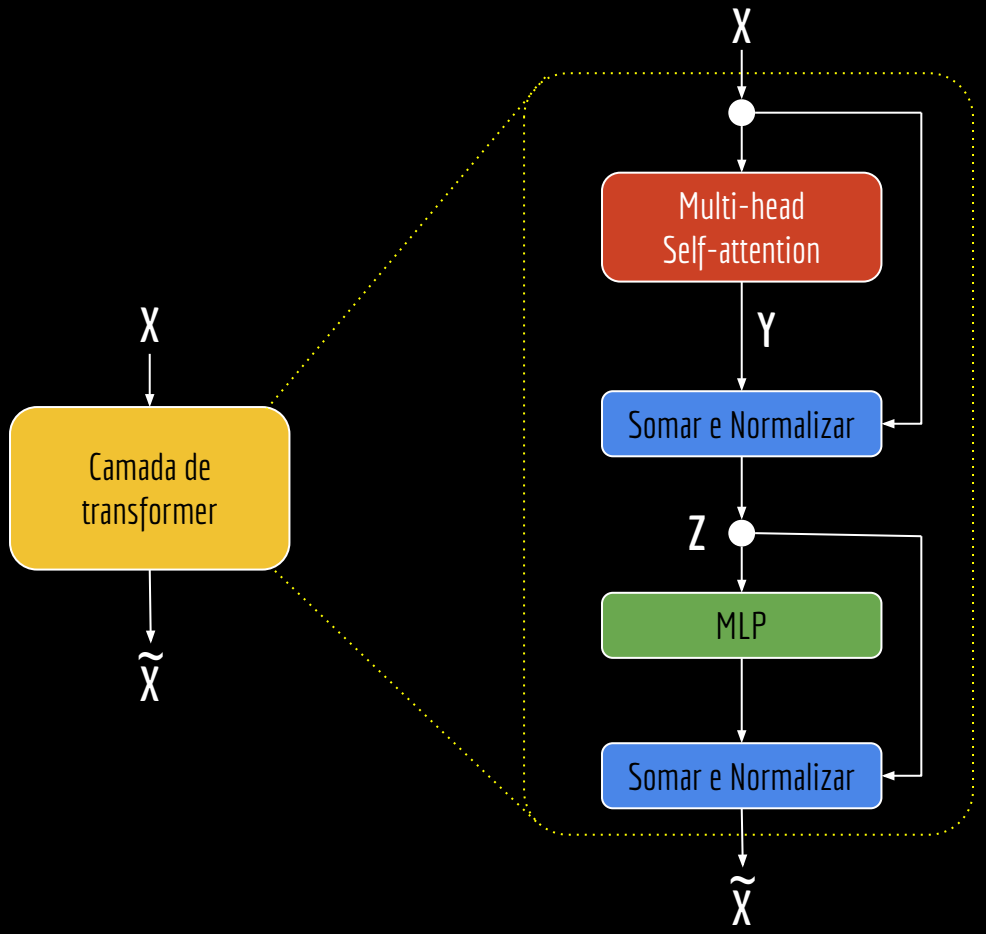


Múltiplas Camadas

$$\mathbf{Z} = LayerNorm[\mathbf{Y} + \mathbf{X}]$$
$$\tilde{\mathbf{X}} = LayerNorm[MLP[\mathbf{Z}] + \mathbf{Z}]$$



Múltiplas Camadas



Faça você mesmo

Descubra (usando o código do Google Colab) quantas camadas de *transformer* a rede Llama 3.2 que estamos usando possui.

Positional encoding

Até agora um transformer é equivariante a permutações da entrada.

Em muitos casos os dados são sequenciais, e a ordem importa.

Exemplo:

“Ele disse que viria, mas não veio.”

“Ele disse que não viria, mas veio.”

Positional encoding

Criar um vetor de posição $\mathbf{r}_n$, com as mesmas dimensões de $\mathbf{x}_n$.

O vetor a ser enviado para a rede, com as informações posicionais fica:

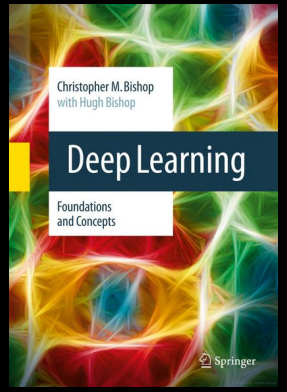
$$\tilde{\mathbf{x}}_n = \mathbf{x}_n + \mathbf{r}_n$$

Positional encoding

Criar um vetor de posição $\mathbf{r}_n$, com as mesmas dimensões de $\mathbf{x}_n$.

O vetor a ser enviado para a rede, com as informações posicionais fica:

$$\tilde{\mathbf{x}}_n = \mathbf{x}_n + \mathbf{r}_n$$



$\mathbf{r}_n$ geralmente é uma combinação de valores de senos e cossenos.

Veja em Bishop (2024) e Vaswani et al. (2017) como a inclusão de $\mathbf{r}_n$ não corrompe os vetores de entrada, e como a combinação de senos e cossenos consegue codificar a posição dos *tokens*.


Attention Is All You Need

Abstract

1 Introduction

Self-supervised learning

Considerando modelos de linguagem, uma das formas que podemos fazer o treinamento é mascarando trechos das entradas, e computando o loss baseado na saída gerada para a próxima palavra.

A previsão do tempo é de chuva para hoje, é melhor levar um ‹máscara›.

Dessa forma, grandes volumes de dados textuais podem ser processados para realizar o treinamento.

Inferência

A inferência em LLMs comumente é feita como um modelo **auto-regressivo**.

Considerando as *n-L* palavras anteriores, prever a *enésima* palavra na sequência.

Depois, utilizar a palavra recém gerada como parte da entrada, e prever a palavra *n+1*.

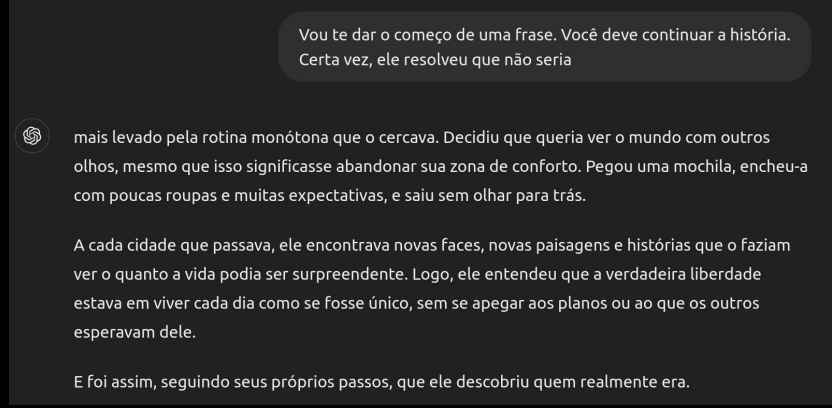
Quanto maior o valor de L, mais informação passada pode ser levada em consideração, ao preço de um custo computacional mais alto.

$$p(x_n | x_{n-1}, x_{n-2}, \ldots, x_{n-L})$$

Inferência

Vou te dar o começo de uma frase. Você deve continuar a história.

Certa vez, ele resolveu que não seria mais



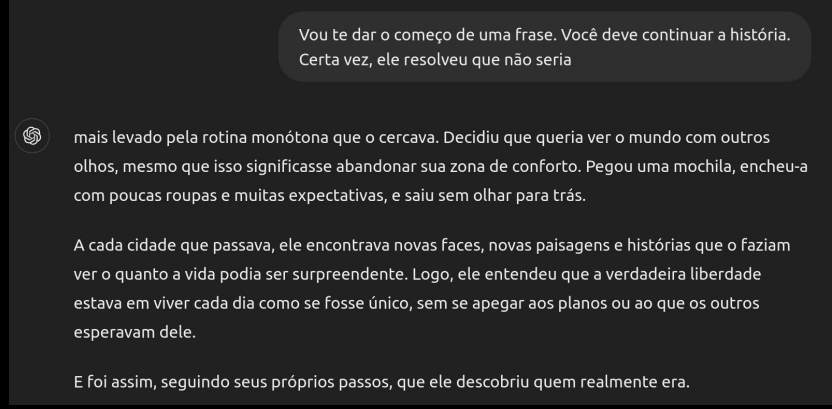
ChatGPT.

Considerando token = palavras e pontuação.
L = 23.
Entrada: N - L Tokens.
Saída: próximo Token.
Tokens que estão fora da janela.

Inferência

Vou te dar o começo de uma frase. Você deve continuar a história.

Certa vez, ele resolveu que não seria mais levado



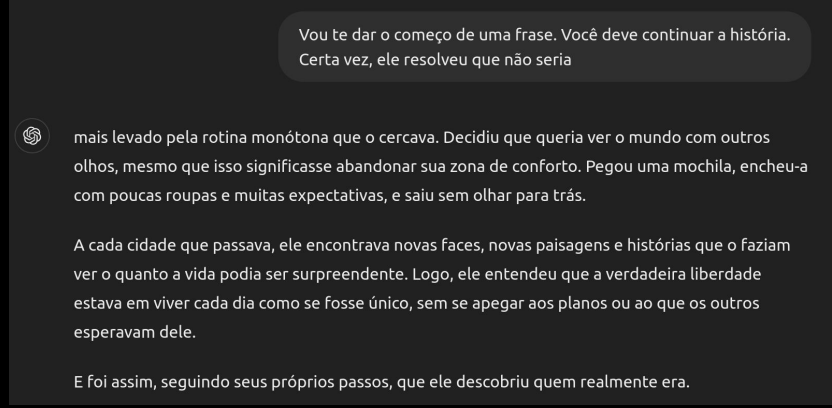
ChatGPT.

Considerando token = palavras e pontuação.
L = 23.
Entrada: N - L Tokens.
Saída: próximo Token.
Tokens que estão fora da janela.

Inferência

Vou te dar o começo de uma frase. Você deve continuar a história.

Certa vez, ele resolveu que não seria mais levado pela



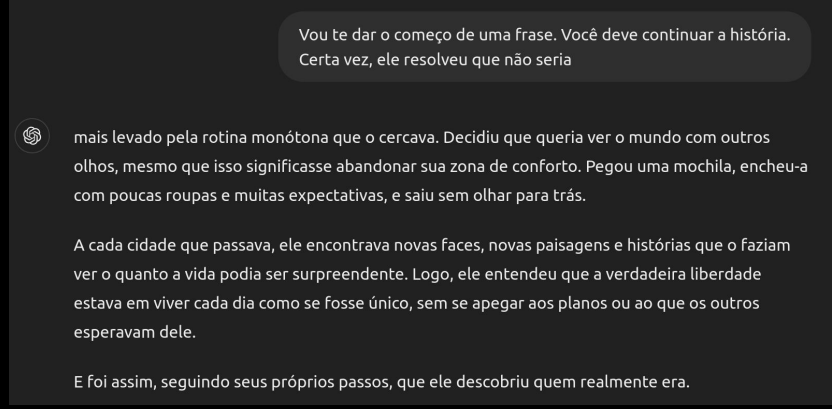
ChatGPT.

Considerando token = palavras e pontuação.
L = 23.
Entrada: N - L Tokens.
Saída: próximo Token.
Tokens que estão fora da janela.

Inferência

Vou te dar o começo de uma frase. Você deve continuar a história.

Certa vez, ele resolveu que não seria mais levado pela rotina



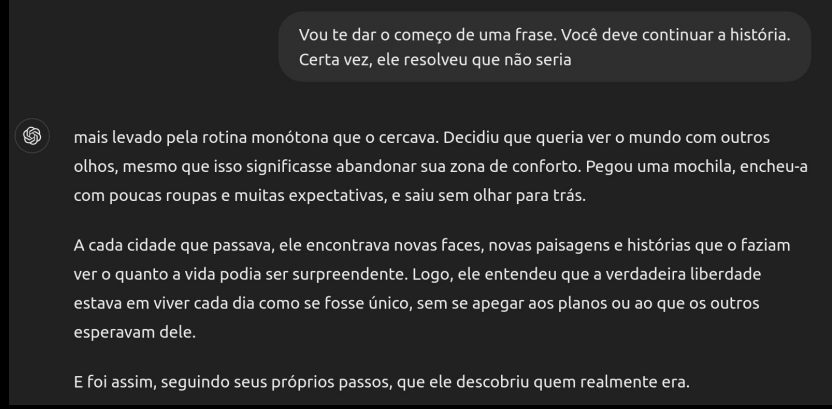
ChatGPT.

Considerando token = palavras e pontuação.
L = 23.
Entrada: N - L Tokens.
Saída: próximo Token.
Tokens que estão fora da janela.

Inferência

Vou te dar o começo de uma frase. Você deve continuar a história.

Certa vez, ele resolveu que não seria mais levado pela rotina monótona



ChatGPT.

Considerando token = palavras e pontuação.
L = 23.
Entrada: N - L Tokens.
Saída: próximo Token.
Tokens que estão fora da janela.

Escolhendo o próximo Token

A rede gera como resposta as probabilidades *a posteriori* (através da função *softmax*) de cada token no dicionário ser a próxima palavra na sentença.

Poderíamos fazer uma busca gulosa (greedy search), e usar o token com a maior probabilidade.

O que você esperaria com isso?

Escolhendo o próximo Token

A rede gera como resposta as probabilidades *a posteriori* (através da função *softmax*) de cada token no dicionário ser a próxima palavra na sentença.

Poderíamos fazer uma busca gulosa (greedy search), e usar o token com a maior probabilidade.

+ Baixo custo.

Resultado determinístico (se isso é bom ou ruim ... depende).

Escolhendo o próximo Token

A rede gera como resposta as probabilidades *a posteriori* (através da função *softmax*) de cada token no dicionário ser a próxima palavra na sentença.

Poderíamos fazer uma busca gulosa (greedy search), e usar o token com a maior probabilidade.

+ Baixo custo.

Resultado determinístico (se isso é bom ou ruim ... depende).

- Uma busca gulosa não garante que a sentença como um todo é a que tem a melhor probabilidade a posteriori combinada (de forma similar a uma busca gulosa em um grafo).

Temperatura

Uma técnica que possui um bom custo x benefício e é geralmente empregada é o uso de uma temperatura *T* na função softmax.

Uma possibilidade é combinar com a seleção *top-p*, onde selecionamos todos os tokens, ordenados da maior probabilidade para a menor, até que a soma das probabilidades seja maior ou igual a p.

Selecionar aleatoriamente um desses tokens.

Saída da rede com temperatura é:

$$y_i = \frac{e^{\frac{\hat{y}_i}{T}}}{\sum_j e^{\frac{\hat{y}_j}{T}}}$$

e é o número de Euler.
$\hat{y}$ é a saída final da rede antes de passar pelo *softmax*.
j é a quantidade de itens no dicionário de *tokens*.

Faça você mesmo

Considere o exemplo disponibilizado no Colab.

No exemplo é dado um vetor de exemplo de *logits* (saídas antes de passar pelo softmax).

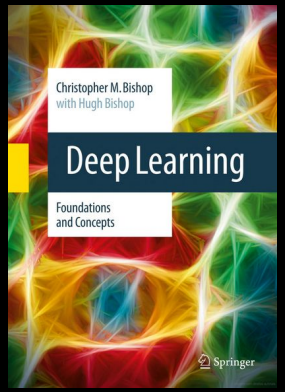
Não está implementado o conceito de temperatura. Implemente, e verifique como a saída é modificada de acordo com a temperatura.

Exercícios

1. Modifique o exemplo do Llama 3 disponibilizado no Google Colab para introduzir o conceito de auto-regressão. O modelo vai gerar texto até encontrar um número máximo de tokens, ou até encontrar um token de fim de texto.
2. Teste diferentes temperaturas para gerar o texto do item 1.

Referências

Bishop, C. M., Bishop, H. Deep Learning: Foundations and Concepts. 2023.



Bahdanau, Cho e Bengio. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473, 2014.


Published as a conference paper at ICLR 2015

NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho Yoshua Bengio*
Université de Montréal

ABSTRACT

Neural machine translation is a recently proposed approach to machine translation. Unlike the traditional statistical machine translation, the neural machine translation aims at building a single neural network that can be jointly tuned to maximize the translation performance. The models proposed recently for neural machine translation often belong to a family of encoder-decoders and encode a source sentence into a fixed-length vector from which a decoder generates a translation. In this paper, we conjecture that the use of a fixed-length vector is a bottleneck in improving the performance of this basic encoder-decoder architecture, and propose to extend this by allowing a model to automatically (soft-)search for parts of a source sentence that are relevant to predicting a target word, without having to form these parts as a hard segment explicitly. With this new approach, we achieve a translation performance comparable to the existing state-of-the-art phrase-based system on the task of English-to-French translation. Furthermore, qualitative analysis reveals that the (soft-)alignments found by the model agree well with our intuition.

1 INTRODUCTION

Neural machine translation is a newly emerging approach to machine translation, recently proposed by Kalchbrenner and Blunsom (2013), Sutskever et al. (2014), and Cho et al. (2014b). Unlike the traditional phrase-based translation system (see, e.g., Koehn et al., 2003) which consists of many small sub-components that are tuned separately, neural machine translation attempts to build and train a single, large neural network that reads a sentence and outputs a correct translation.

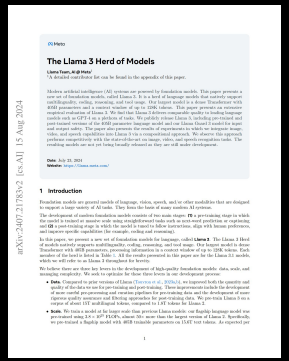
Most of the proposed neural machine translation models belong to a family of encoder-decoders (Sutskever et al., 2014; Cho et al., 2014a), with an encoder and a decoder for each language, or involve a language-specific encoder applied to each sentence whose outputs are then compared (Hermann and Blunsom, 2013). An encoder neural network reads and encodes a source sentence into a fixed-length vector. A decoder then outputs a translation from the encoded vector. The whole encoder-decoder system, which consists of the encoder and the decoder for a language pair, is jointly trained to maximize the probability of a correct translation given a source sentence.

A potential issue with this encoder-decoder approach is that a neural network needs to be able to compress all the necessary information of a source sentence into a fixed-length vector. This may make it difficult for the neural network to cope with long sentences, especially those that are longer than the sentences in the training corpus. Cho et al. (2014b) showed that indeed the performance of a basic encoder-decoder deteriorates rapidly as the length of an input sentence increases.

In order to address this issue, we introduce an extension to the encoder-decoder model which learns to align and translate jointly. Each time the proposed model generates a word in a translation, it (soft-)searches for a set of positions in a source sentence where the most relevant information is concentrated. The model then predicts a target word based on the context vectors associated with these source positions and all the previous generated target words.

*CIFAR Senior Fellow

Dubey, A. et al. The Llama 3 Herd of Models. 2024.



Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser e Polosukhin. Attention is all you need. Advances in Neural Information Processing Systems, 2017.


Attention Is All You Need

Ashish Vaswani* Noam Shazeer* Niki Parmar* Jakob Uszkoreit*
Google Brain Google Brain Google Research Google Research
avaswani@google.com noam@google.com nikip@google.com usz@google.com

Llion Jones* Aidan N. Gomez* Łukasz Kaiser*
Google Research University of Toronto Google Brain
llion@google.com aidan@cs.toronto.edu lukaszkaiser@google.com

Illia Polosukhin*
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

1 Introduction

Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and transduction problems such as language modeling and machine translation [35, 2, 5]. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures [38, 24, 15].

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

†Work performed while at Google Brain.

‡Work performed while at Google Research.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

Licença