Provided proper attribution is provided, Google hereby grants permission to reproduce the tables and figures in this paper solely for use in journalistic or scholarly works.

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

Łukasz Kaiser[*] Google Brain lukaszkaiser@google.com

Illia Polosukhin[*] [‡] illia.polosukhin@gmail.com

arXiv:1706.03762v7 [cs.CL] 2 Aug 2023

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. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

[*]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.

Desde que seja fornecida a devida atribuição, o Google concede permissão para reproduzir as tabelas e figuras deste artigo exclusivamente para uso em trabalhos jornalísticos ou acadêmicos.

Atenção é Tudo que Você Precisa

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain oam@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* Universidade de Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* illia.polosukhin@gmail.com

arXiv:1706.03762v7 [cs.CL] 2 Aug 2023

Resumo

Os modelos dominantes de transdução de sequência baseiam-se em redes neurais recorrentes ou convolucionais complexas que incluem um codificador e um decodificador. Os melhores modelos também conectam o codificador e o decodificador por meio de um mecanismo de atenção. Propomos uma nova arquitetura de rede simples, o Transformer, baseada exclusivamente em mecanismos de atenção, dispensando completamente a recorrência e as convoluções. Experimentos em duas tarefas de tradução automática mostram que esses modelos são superiores em qualidade, sendo mais paralelizáveis e exigindo significativamente menos tempo de treinamento. Nosso modelo alcança 28,4 BLEU na tarefa de tradução inglês-alemão do WMT 2014, superando os melhores resultados existentes, incluindo ensembles, em mais de 2 BLEU. Na tarefa de tradução inglês-francês do WMT 2014, nosso modelo estabelece uma nova pontuação state-of-the-art de modelo único BLEU de 41,8 após 3,5 dias de treinamento em oito GPUs, uma pequena fração dos custos de treinamento dos melhores modelos da literatura. Mostramos que o Transformer generaliza bem para outras tarefas, aplicando-o com sucesso à análise sintática do inglês tanto com grandes quanto com dados de treinamento limitados.

*Contribuição igual. A ordem de listagem é aleatória. Jakob propôs substituir as RNNs por autoatenção e iniciou o esforço para avaliar essa ideia. Ashish, com Illia, projetou e implementou os primeiros modelos Transformer e esteve crucialmente envolvido em todos os aspectos deste trabalho. Noam propôs a atenção de produto ponto escalonado, a atenção de múltiplas cabeças e a representação de posição livre de parâmetros, tornando-se a outra pessoa envolvida em quase todos os detalhes. Niki projetou, implementou, ajustou e avaliou inúmeras variantes de modelos em nossa base de código original e tensor2tensor. Llion também experimentou com variantes de modelos inovadoras, foi responsável pela nossa base de código inicial e pelas inferências e visualizações eficientes. Lukasz e Aidan passaram incontáveis longos dias projetando várias partes e implementando o tensor2tensor, substituindo nossa base de código anterior, melhorando significativamente os resultados e acelerando massivamente nossa pesquisa.

Trabalho realizado enquanto no Google Brain.

Trabalho realizado enquanto no Google Research.

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

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].

Recurrent models typically factor computation along the symbol positions of the input and output sequences. Aligning the positions to steps in computation time, they generate a sequence of hidden states h_t, as a function of the previous hidden state h_{t-1} and the input for position t. This inherently sequential nature precludes parallelization within training examples, which becomes critical at longer sequence lengths, as memory constraints limit batching across examples. Recent work has achieved significant improvements in computational efficiency through factorization tricks [21] and conditional computation [32], while also improving model performance in case of the latter. The fundamental constraint of sequential computation, however, remains.

Attention mechanisms have become an integral part of compelling sequence modeling and transduction models in various tasks, allowing modeling of dependencies without regard to their distance in the input or output sequences [2, 19]. In all but a few cases [27], however, such attention mechanisms are used in conjunction with a recurrent network.

In this work we propose the Transformer, a model architecture eschewing recurrence and instead relying entirely on an attention mechanism to draw global dependencies between input and output. The Transformer allows for significantly more parallelization and can reach a new state of the art in translation quality after being trained for as little as twelve hours on eight P100 GPUs.

2 Background

The goal of reducing sequential computation also forms the foundation of the Extended Neural GPU [16], ByteNet [18] and ConvS2S [9], all of which use convolutional neural networks as basic building block, computing hidden representations in parallel for all input and output positions. In these models, the number of operations required to relate signals from two arbitrary input or output positions grows in the distance between positions, linearly for ConvS2S and logarithmically for ByteNet. This makes it more difficult to learn dependencies between distant positions [12]. In the Transformer this is reduced to a constant number of operations, albeit at the cost of reduced effective resolution due to averaging attention-weighted positions, an effect we counteract with Multi-Head Attention as described in section 3.2.

Self-attention, sometimes called intra-attention is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence. Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations [4, 27, 28, 22].

End-to-end memory networks are based on a recurrent attention mechanism instead of sequence-aligned recurrence and have been shown to perform well on simple-language question answering and language modeling tasks [34].

To the best of our knowledge, however, the Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs or convolution. In the following sections, we will describe the Transformer, motivate self-attention and discuss its advantages over models such as [17, 18] and [9].

3 Model Architecture

Most competitive neural sequence transduction models have an encoder-decoder structure [5, 2, 35]. Here, the encoder maps an input sequence of symbol representations $(x_1, ..., x_n)$ to a sequence of continuous representations $\mathbf{z} = (z_1, ..., z_n)$. Given $\mathbf{z}$, the decoder then generates an output sequence $(y_1, ..., y_m)$ of symbols one element at a time. At each step the model is auto-regressive [10], consuming the previously generated symbols as additional input when generating the next.

1 Introdução

Redes neurais recorrentes, memória de curto prazo de longo termo [13] e redes neurais recorrentes com portas [7], em particular, têm se consolidado como abordagens state of the art em problemas de modelagem e transdução de sequências, como modelagem de linguagem e tradução automática [35, 2, 5]. Diversos esforços têm continuado a expandir os limites dos modelos de linguagem recorrentes e das arquiteturas encoder-decoder [38, 24, 15].

Modelos recorrentes tipicamente fatoram o cálculo ao longo das posições dos símbolos nas sequências de entrada e saída. Ao alinhar as posições com etapas no tempo de computação, eles geram uma sequência de estados ocultos $\boldsymbol{h_t}$, em função do estado oculto anterior $\boldsymbol{h_{t-1}}$ e da entrada para a posição $\boldsymbol{t}$. Essa natureza intrinsecamente sequencial impede a paralelização dentro dos exemplos de treinamento, o que se torna crítico em comprimentos de sequência maiores, pois restrições de memória limitam o batching entre exemplos. Trabalhos recentes alcançaram melhorias significativas na eficiência computacional por meio de truques de fatoração [21] e computação condicional [32], enquanto também melhoram o desempenho do modelo no caso deste último. A restrição fundamental da computação sequencial, no entanto, permanece.

Os mecanismos de atenção tornaram-se parte integrante de modelos de sequências e de transdução convincentes em diversas tarefas, permitindo a modelagem de dependências sem considerar sua distância nas sequências de entrada ou saída [2, 19]. No entanto, em todos, exceto poucos casos [27], esses mecanismos de atenção são utilizados em conjunto com uma rede recorrente.

Neste trabalho, propomos o Transformer, uma arquitetura de modelo que evita a recorrência e, em vez disso, baseia-se inteiramente em um mecanismo de atenção para capturar dependências globais entre entrada e saída. O Transformer permite uma paralelização significativamente maior e pode atingir um novo estado da arte na qualidade de tradução após ser treinado por tão pouco quanto doze horas em oito GPUs P100.

2 Contexto

O objetivo de reduzir a computação sequencial também constitui a base da Extended Neural GPU [16], ByteNet [18] e ConvS2S [9], todos os quais utilizam redes neurais convolucionais como bloco de construção básico, calculando representações ocultas em paralelo para todas as posições de entrada e saída. Nesses modelos, o número de operações necessárias para relacionar sinais de duas posições de entrada ou saída arbitrárias cresce com a distância entre as posições, linearmente para o ConvS2S e logaritmicamente para o ByteNet. Isso torna mais difícil aprender dependências entre posições distantes [12]. No Transformer, isso é reduzido a um número constante de operações, embora à custa de uma resolução efetiva reduzida devido à média das posições ponderadas por atenção, um efeito que contrabalançamos com Atenção de Múltiplas Cabeças, conforme descrito na seção 3.2.

Auto-atenção, às vezes chamada de intra-atenção, é um mecanismo de atenção que relaciona diferentes posições de uma única sequência a fim de computar uma representação da sequência. A auto-atenção tem sido usada com sucesso em uma variedade de tarefas, incluindo compreensão de leitura, sumarização abstrativa, implicação textual e aprendizado de representações de sentenças independentes da tarefa [4, 27, 28, 22].

As redes de memória end-to-end baseiam-se em um mecanismo de atenção recorrente em vez de recorrência alinhada a sequências e demonstraram bom desempenho em tarefas de resposta a perguntas com linguagem simples e modelagem de linguagem [34].

No que nos é conhecido, no entanto, o Transformer é o primeiro modelo de transdução que depende inteiramente de auto-atenção para calcular representações de sua entrada e saída sem utilizar RNNs alinhadas a sequências ou convolução. Nas seções seguintes, descreveremos o Transformer, apresentaremos a motivação para a auto-atenção e discutiremos suas vantagens em relação a modelos como [17, 18] e [9].

3 Arquitetura do Modelo

A maioria dos modelos competitivos de transdução sequencial neural possui uma estrutura codificador-decodificador [5, 2, 35]. Aqui, o codificador mapeia uma sequência de entrada de representações de símbolos $(x_1, \ldots, x_n)$ para uma sequência de representações contínuas $\mathbf{z} = (z_1, \ldots, z_n)$. Dado $\mathbf{z}$, o decodificador gera então uma sequência de saída $(y_1, \ldots, y_m)$ de símbolos um elemento por vez. Em cada passo, o modelo é autoregressivo [10], consumindo os símbolos gerados anteriormente como entrada adicional ao gerar o próximo.

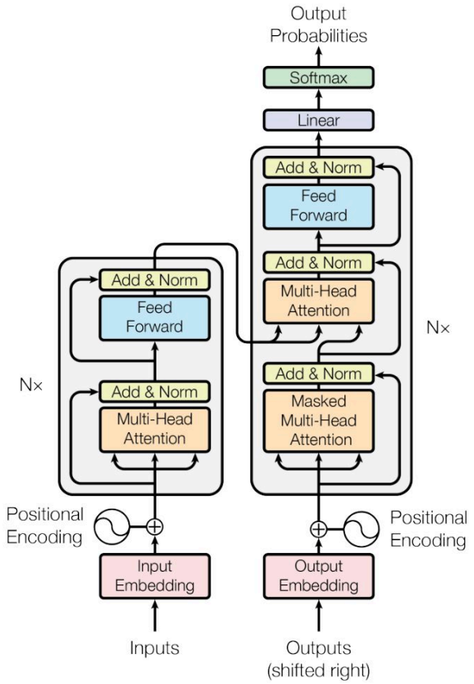


Figure 1: The Transformer - model architecture.

The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, shown in the left and right halves of Figure 1, respectively.

3.1 Encoder and Decoder Stacks

Encoder: The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. We employ a residual connection [11] around each of the two sub-layers, followed by layer normalization [1]. That is, the output of each sub-layer is $\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$, where $\mathrm{Sublayer}(x)$ is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension $d_{\text{model}} = 512$.

Decoder: The decoder is also composed of a stack of $N = 6$ identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

3.2 Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum

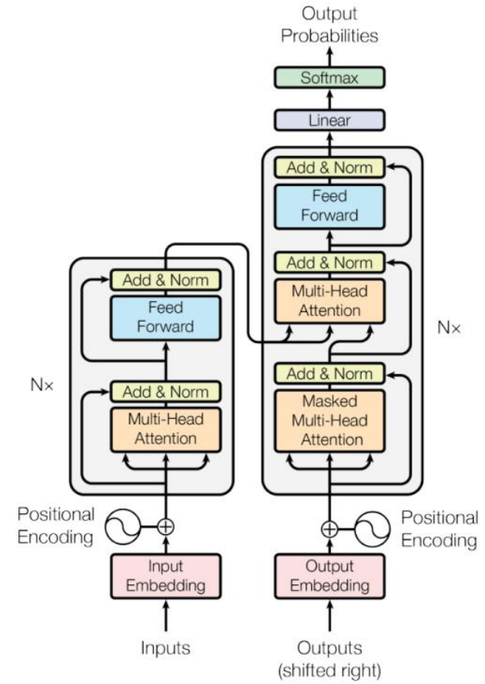


Figura 1: A arquitetura do modelo Transformer.

O Transformer segue esta arquitetura geral, utilizando autoatenção empilhada e camadas totalmente conectadas ponto a ponto tanto para o codificador quanto para o decodificador, mostradas nas metades esquerda e direita da Figura 1, respectivamente.

3.1 Pilhas de Codificador e Decodificador

Codificador: O codificador é composto por uma pilha de $N=6$ camadas idênticas. Cada camada possui dois sub-camadas. A primeira é um mecanismo de autoatenção multicanal, e a segunda é uma rede feed-forward totalmente conectada simples, ponto a ponto. Empregamos uma conexão residual [11] ao redor de cada uma das duas sub-camadas, seguida por normalização de camada [1]. Ou seja, a saída de cada sub-camada é LayerNorm(x + Subcamada(x)), onde Subcamada(x) é a função implementada pela própria sub-camada. Para facilitar essas conexões residuais, todas as sub-camadas no modelo, bem como as camadas de embedding, produzem saídas de dimensão $d_{\text{model}}=512$.

Decoder: O decoder também é composto por uma pilha de $N=6$ camadas idênticas. Além das duas sub-camadas em cada camada do encoder, o decoder insere uma terceira sub-camada, que realiza atenção multi-head sobre a saída da pilha do encoder. Similarmente ao encoder, empregamos conexões residuais ao redor de cada uma das sub-camadas, seguidas pela normalização da camada. Nós também modificamos a sub-camada de auto-atenção na pilha do decoder para impedir que posições atendam a posições subsequentes. Essa máscara, combinada com o fato de que os embeddings de saída estão defasados em uma posição, garante que as previsões para a posição i possam depender apenas das saídas conhecidas em posições menores que i.

3.2 Atenção

Uma função de atenção pode ser descrita como mapear uma consulta e um conjunto de pares chave-valor em uma saída, onde a consulta, as chaves, os valores e a saída são todos vetores. A saída é computada como uma soma ponderada

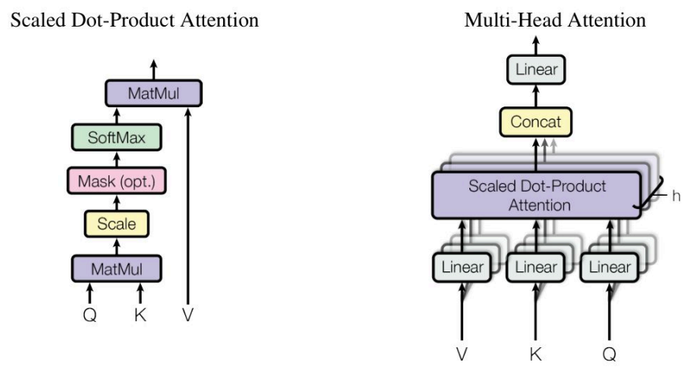


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

3.2.1 Scaled Dot-Product Attention

We call our particular attention "Scaled Dot-Product Attention" (Figure 2). The input consists of queries and keys of dimension d_k, and values of dimension d_v. We compute the dot products of the query with all keys, divide each by $\sqrt{d_k}$, and apply a softmax function to obtain the weights on the values.

In practice, we compute the attention function on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V. We compute the matrix of outputs as:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \quad (1)$$

The two most commonly used attention functions are additive attention [2], and dot-product (multiplicative) attention. Dot-product attention is identical to our algorithm, except for the scaling factor of $\frac{1}{\sqrt{d_k}}$. Additive attention computes the compatibility function using a feed-forward network with a single hidden layer. While the two are similar in theoretical complexity, dot-product attention is much faster and more space-efficient in practice, since it can be implemented using highly optimized matrix multiplication code.

While for small values of d_k the two mechanisms perform similarly, additive attention outperforms dot product attention without scaling for larger values of d_k [3]. We suspect that for large values of d_k, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients [4]. To counteract this effect, we scale the dot products by $\frac{1}{\sqrt{d_k}}$.

3.2.2 Multi-Head Attention

Instead of performing a single attention function with d_{model}-dimensional keys, values and queries, we found it beneficial to linearly project the queries, keys and values h times with different, learned linear projections to d_k, d_k and d_v dimensions, respectively. On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding d_v-dimensional

[4]To illustrate why the dot products get large, assume that the components of q and k are independent random variables with mean 0 and variance 1. Then their dot product, $q \cdot k = \sum_{i=1}^{d_k} q_i k_i$, has mean 0 and variance d_k.

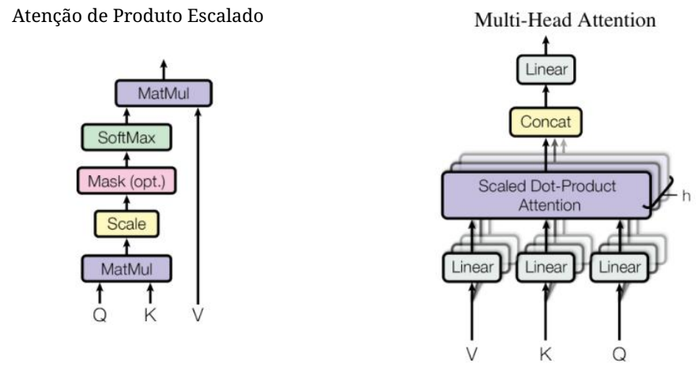


Figura 2: (esquerda) Atenção de Produto Escalar. (direita) Atenção Multicabeça consiste em várias camadas de atenção executando em paralelo.

dos valores, em que o peso atribuído a cada valor é computado por uma função de compatibilidade da consulta com a chave correspondente.

3.2.1 Atenção de Produto Escalar

Chamamos a nossa atenção para o "Atenção de Produto Escalado" (Figura 2). A entrada consiste em queries e keys de dimensão d_k, e values de dimensão d_v. Computamos os produtos escalares da query com todos os keys, dividimos cada um por $\sqrt{d_k}$, e aplicamos uma função softmax para obter os pesos sobre os *values$.

Na prática, calculamos a função de atenção em um conjunto de consultas simultaneamente, empacotadas em uma matriz Q. As chaves e os valores também são empacotados em matrizes K e V. Calculamos a matriz de saídas como:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \quad (1)$$

As duas funções de atenção mais comumente utilizadas são a atenção aditiva [2] e a atenção por produto ponto (multiplicativa). A atenção por produto ponto é idêntica ao nosso algoritmo, exceto pelo fator de escalonamento de $\frac{1}{\sqrt{d_k}}$. A atenção aditiva calcula a função de compatibilidade usando uma rede feed-forward com uma única camada oculta. Embora as duas sejam similares em complexidade teórica, a atenção por produto ponto é muito mais rápida e eficiente em termos de espaço na prática, uma vez que pode ser implementada utilizando código de multiplicação de matrizes altamente otimizado.

Enquanto para pequenos valores de d_k os dois mecanismos apresentam desempenho semelhante, a atenção aditiva supera a atenção de produto escalar sem escalonamento para maiores valores de d_k [3]. Suspeitamos que, para grandes valores de d_k, os produtos escalares crescem muito em magnitude, empurrando a função softmax para regiões onde ela possui gradientes extremamente pequenos [4]. Para contrariar esse efeito, escalonamos os produtos escalares por $\frac{1}{\sqrt{d_k}}$.

3.2.2 Atenção de Múltiplas Cabeças

Em vez de realizar uma única função de atenção com chaves, valores e consultas dimensionais d_{model}, verificamos que é benéfico projetar linearmente as consultas, chaves e valores h vezes com projeções lineares aprendidas diferentes para as dimensões d_k, d_k e d_v, respectivamente. Em cada uma dessas versões projetadas de consultas, chaves e valores, realizamos então a função de atenção em paralelo, resultando em [texto cortado] dimensionais d_v.

[4]Para ilustrar por que os produtos escalares tornam-se grandes, assuma que as componentes de d_k e são variáveis aleatórias independentes com média 0 e variância 1. Então seu produto escalar, $q \cdot k = \sum_{i=1} q_i k_i$, tem média 0 e variância d_k.

output values. These are concatenated and once again projected, resulting in the final values, as depicted in Figure 2.

Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.

$$\begin{aligned}\text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O \\ \text{where head}_\text{i} &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned}$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_\text{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_\text{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_\text{model} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_\text{model}}$.

In this work we employ $h = 8$ parallel attention layers, or heads. For each of these we use $d_k = d_v = d_\text{model}/h = 64$. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

3.2.3 Applications of Attention in our Model

The Transformer uses multi-head attention in three different ways:

- In "encoder-decoder attention" layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence. This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models such as [38, 2, 9].
- The encoder contains self-attention layers. In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder.
- Similarly, self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position. We need to prevent leftward information flow in the decoder to preserve the auto-regressive property. We implement this inside of scaled dot-product attention by masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections. See Figure 2.

3.3 Position-wise Feed-Forward Networks

In addition to attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2}$$

While the linear transformations are the same across different positions, they use different parameters from layer to layer. Another way of describing this is as two convolutions with kernel size 1. The dimensionality of input and output is $d_\text{model} = 512$, and the inner-layer has dimensionality $d_{ff} = 2048$.

3.4 Embeddings and Softmax

Similarly to other sequence transduction models, we use learned embeddings to convert the input tokens and output tokens to vectors of dimension d_model. We also use the usual learned linear transformation and softmax function to convert the decoder output to predicted next-token probabilities. In our model, we share the same weight matrix between the two embedding layers and the pre-softmax linear transformation, similar to [30]. In the embedding layers, we multiply those weights by $\sqrt{d_\text{model}}$.

valores de saída. Estes são concatenados e, uma vez mais, projetados, resultando nos valores finais, conforme ilustrado na Figura 2.

A atenção multi-cabeça permite que o modelo atenda conjuntamente a informações de diferentes subespaços de representação em diferentes posições. Com uma única cabeça de atenção, a média inibe isso.

$$\begin{aligned}\text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O \\ \text{where head}_\text{i} &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned}$$

Onde as projeções são matrizes de parâmetros $W_i^Q \in \mathbb{R}^{d_\text{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_\text{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_\text{model} \times d_v}$ e $W^O \in \mathbb{R}^{hd_v \times d_\text{model}}$.

Neste trabalho, employamos $h = 8$ camadas de atenção paralelas, ou cabeças. Para cada uma delas, usamos $d_k = d_v = d_\text{model}/h = 64$. Devido à dimensão reduzida de cada cabeça, o custo computacional total é similar ao de atenção de cabeça única com dimensionalidade completa.

3.2.3 Aplicações de Atenção em Nosso Modelo

O Transformer utiliza atenção de múltiplas cabeças de três maneiras diferentes:

Nas camadas de "atenção encoder-decoder", as consultas provêm da camada anterior do decoder, e as chaves de memória e os valores provêm da saída do encoder. Isso permite que cada posição no decoder atenda a todas as posições na sequência de entrada. Isso imita os mecanismos típicos de atenção encoder-decoder em modelos de sequência-para-sequência, como [38, 2, 9].

O encoder contém camadas de autoatenção. Em uma camada de autoatenção, todas as chaves, valores e consultas provêm do mesmo local, neste caso, a saída da camada anterior no encoder. Cada posição no encoder pode atender a todas as posições na camada anterior do encoder.

Da mesma forma, as camadas de autoatenção no decodificador permitem que cada posição no decodificador atenda a todas as posições no decodificador até e incluindo aquela posição. Precisamos impedir o fluxo de informações da esquerda para a direita no decodificador para preservar a propriedade autoregressiva. Implementamos isso dentro da atenção de produto escalar escalonada mascarando (definindo como $-\infty$) todos os valores na entrada do softmax que correspondem a conexões ilegais. Veja a Figura 2.

3.3 Redes Feed-Forward Posicionais

Além das sub-camadas de atenção, cada uma das camadas em nosso codificador e decodificador contém uma rede feed-forward totalmente conectada, que é aplicada a cada posição separadamente e de forma idêntica. Esta consiste em duas transformações lineares com uma ativação ReLU no meio.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2}$$

Embora as transformações lineares sejam as mesmas em diferentes posições, elas utilizam parâmetros distintos de camada a camada. Outra forma de descrever isso é como duas convoluções com tamanho de kernel 1. A dimensionalidade de entrada e saída é $d_\text{model} = 512$, e a camada interna possui dimensionalidade $d_{ff} = 2048$.

3.4 Embeddings e Softmax

Da mesma forma que outros modelos de transdução de sequência, usamos embeddings aprendidos para converter os tokens de entrada e os tokens de saída em vetores de dimensão d_model. Também usamos a transformação linear aprendida e a função softmax usuais para converter a saída do decodificador em probabilidades preditas para o próximo token. Em nosso modelo, compartilhamos a mesma matriz de pesos entre as duas camadas de embedding e a transformação linear pré-softmax, similar a [30]. Nas camadas de embedding, multiplicamos esses pesos por $\sqrt{d_\text{model}}$.

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

3.5 Positional Encoding

Since our model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence. To this end, we add "positional encodings" to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension d_{model} as the embeddings, so that the two can be summed. There are many choices of positional encodings, learned and fixed [9].

In this work, we use sine and cosine functions of different frequencies:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}.

We also experimented with using learned positional embeddings [9] instead, and found that the two versions produced nearly identical results (see Table 3 row (E)). We chose the sinusoidal version because it may allow the model to extrapolate to sequence lengths longer than the ones encountered during training.

4 Why Self-Attention

In this section we compare various aspects of self-attention layers to the recurrent and convolutional layers commonly used for mapping one variable-length sequence of symbol representations $(x_1, ..., x_n)$ to another sequence of equal length $(z_1, ..., z_n)$, with $x_i, z_i \in \mathbb{R}^d$, such as a hidden layer in a typical sequence transduction encoder or decoder. Motivating our use of self-attention we consider three desiderata.

One is the total computational complexity per layer. Another is the amount of computation that can be parallelized, as measured by the minimum number of sequential operations required.

The third is the path length between long-range dependencies in the network. Learning long-range dependencies is a key challenge in many sequence transduction tasks. One key factor affecting the ability to learn such dependencies is the length of the paths forward and backward signals have to traverse in the network. The shorter these paths between any combination of positions in the input and output sequences, the easier it is to learn long-range dependencies [12]. Hence we also compare the maximum path length between any two input and output positions in networks composed of the different layer types.

As noted in Table 1, a self-attention layer connects all positions with a constant number of sequentially executed operations, whereas a recurrent layer requires $O(n)$ sequential operations. In terms of computational complexity, self-attention layers are faster than recurrent layers when the sequence

Tabela 1: Comprimentos máximos de caminho, complexidade por camada e número mínimo de operações sequenciais para diferentes tipos de camada. n é o comprimento da sequência, d é a dimensão da representação, k é o tamanho do kernel das convoluções e r o tamanho do vizinhança na autoatenção restrita.

Tipo de Camada	Complexidade por Camada	Sequencial Operações	Comprimento Máximo do Caminho
Autoatenção	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recorrente	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolucional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Autoatenção (restrita)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

3.5 Codificação Posicional

Como nosso modelo não contém recorrência nem convolução, a fim de que o modelo possa fazer uso da ordem da sequência, devemos injetar alguma informação sobre a posição relativa ou absoluta dos tokens na sequência. Para esse fim, adicionamos "codificações posicionais" aos embeddings de entrada nas bases das pilhas do codificador e do decodificador. As codificações posicionais possuem a mesma dimensão d_{model} dos embeddings, para que as duas possam ser somadas. Existem muitas opções de codificações posicionais, aprendidas e fixas [9].

Neste trabalho, utilizamos funções seno e cosseno de diferentes frequências:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

onde pos é a posição e i é a dimensão. Ou seja, cada dimensão do codificação posicional corresponde a um sinusóide. Os comprimentos de onda formam uma progressão geométrica de 2π a $10000 \cdot 2\pi$. Escolhemos esta função porque hipotetizamos que ela permitiria ao modelo aprender facilmente a atender por posições relativas, já que para qualquer deslocamento fixo k, PE_{pos+k} pode ser representado como uma função linear de PE_{pos}.

Também experimentamos o uso de embeddings posicionais aprendidos [9] em vez disso, e verificamos que as duas versões produziram resultados quase idênticos (veja Tabela 3, linha (E)). Escolhemos a versão senoidal porque ela pode permitir que o modelo extrapole para comprimentos de sequência maiores do que aqueles encontrados durante o treinamento.

4 Por que Autoatenção

Nesta seção, comparamos vários aspectos das camadas de autoatenção com as camadas recorrentes e convolucionais comumente usadas para mapear uma sequência de comprimento variável de representações de símbolos $(x_1, \ldots, x_n)$ em outra sequência de igual comprimento $(z_1, \ldots, z_n)$, com $x_i, z_i \in \mathbb{R}^d$, como uma camada oculta em um codificador ou decodificador típico de transdução de sequências. Para motivar o uso de autoatenção, consideramos três requisitos.

Uma é a complexidade computacional total por camada. A outra é a quantidade de computação que pode ser paralelizada, medida pelo número mínimo de operações sequenciais necessárias.

O terceiro é o comprimento do caminho entre dependências de longo alcance na rede. Aprender dependências de longo alcance é um desafio crucial em muitas tarefas de transdução de sequências. Um fator chave que afeta a capacidade de aprender tais dependências é o comprimento dos caminhos que os sinais forward e backward precisam percorrer na rede. Quanto menores forem esses caminhos entre qualquer combinação de posições nas sequências de entrada e saída, mais fácil é aprender dependências de longo alcance [12]. Portanto, também comparamos o comprimento máximo do caminho entre quaisquer duas posições de entrada e saída em redes compostas pelos diferentes tipos de camadas.

Como observado na Tabela 1, uma camada de autoatenção conecta todas as posições com um número constante de operações executadas sequencialmente, enquanto uma camada recorrente requer $O(n)$ operações sequenciais. Em termos de complexidade computacional, camadas de autoatenção são mais rápidas do que camadas recorrentes quando a sequência

length n is smaller than the representation dimensionality d, which is most often the case with sentence representations used by state-of-the-art models in machine translations, such as word-piece [38] and byte-pair [31] representations. To improve computational performance for tasks involving very long sequences, self-attention could be restricted to considering only a neighborhood of size r in the input sequence centered around the respective output position. This would increase the maximum path length to $O(n/r)$. We plan to investigate this approach further in future work.

A single convolutional layer with kernel width $k < n$ does not connect all pairs of input and output positions. Doing so requires a stack of $O(n/k)$ convolutional layers in the case of contiguous kernels, or $O(log_k(n))$ in the case of dilated convolutions [18], increasing the length of the longest paths between any two positions in the network. Convolutional layers are generally more expensive than recurrent layers, by a factor of k. Separable convolutions [6], however, decrease the complexity considerably, to $O(k \cdot n \cdot d + n \cdot d^2)$. Even with $k = n$, however, the complexity of a separable convolution is equal to the combination of a self-attention layer and a point-wise feed-forward layer, the approach we take in our model.

As side benefit, self-attention could yield more interpretable models. We inspect attention distributions from our models and present and discuss examples in the appendix. Not only do individual attention heads clearly learn to perform different tasks, many appear to exhibit behavior related to the syntactic and semantic structure of the sentences.

5 Training

This section describes the training regime for our models.

5.1 Training Data and Batching

We trained on the standard WMT 2014 English-German dataset consisting of about 4.5 million sentence pairs. Sentences were encoded using byte-pair encoding [3], which has a shared source-target vocabulary of about 37000 tokens. For English-French, we used the significantly larger WMT 2014 English-French dataset consisting of 36M sentences and split tokens into a 32000 word-piece vocabulary [38]. Sentence pairs were batched together by approximate sequence length. Each training batch contained a set of sentence pairs containing approximately 25000 source tokens and 25000 target tokens.

5.2 Hardware and Schedule

We trained our models on one machine with 8 NVIDIA P100 GPUs. For our base models using the hyperparameters described throughout the paper, each training step took about 0.4 seconds. We trained the base models for a total of 100,000 steps or 12 hours. For our big models,(described on the bottom line of table 3), step time was 1.0 seconds. The big models were trained for 300,000 steps (3.5 days).

5.3 Optimizer

We used the Adam optimizer [20] with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$. We varied the learning rate over the course of training, according to the formula:

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step_num^{-0.5}, step_num \cdot warmup_steps^{-1.5}) \quad (3)$$

This corresponds to increasing the learning rate linearly for the first $warmup_steps$ training steps, and decreasing it thereafter proportionally to the inverse square root of the step number. We used $warmup_steps = 4000$.

5.4 Regularization

We employ three types of regularization during training:

o comprimento n é menor que a dimensionalidade da representação d, o que ocorre com maior frequência em representações de frases usadas por modelos state-of-the-art em traduções automáticas, como as representações word-piece [38] e byte-pair [31]. Para melhorar o desempenho computacional em tarefas envolvendo sequências muito longas, a autoatenção poderia ser restrita a considerar apenas uma vizinhança de tamanho r na sequência de entrada centrada em torno da respectiva posição de saída. Isso aumentaria o comprimento máximo do caminho para $O(n/r)$. Prevemos investigar essa abordagem mais detalhadamente em trabalhos futuros.

Uma única camada convolucional com largura de kernel $k < n$ não conecta todos os pares de posições de entrada e saída. Fazer isso requer uma pilha de $O(n/k)$ camadas convolucionais no caso de kernels contíguos, ou $O(\log_k(n))$ no caso de convoluções dilatadas [18], aumentando o comprimento dos caminhos mais longos entre quaisquer duas posições na rede. Camadas convolucionais são geralmente mais custosas do que camadas recorrentes, em um fator de k. Convoluções separáveis [6], no entanto, reduzem consideravelmente a complexidade, para $O(k \cdot n \cdot d + n \cdot d^2)$. Mesmo com $k = n$, however, a complexidade de uma convolução separável é igual à combinação de uma camada de autoatenção e uma camada de feed-forward ponto a ponto, a abordagem que adotamos em nosso modelo.

Como benefício adicional, a autoatenção poderia produzir modelos mais interpretáveis. Analisamos as distribuições de atenção dos nossos modelos e apresentamos e discutimos exemplos no apêndice. Não apenas os cabeçotes de atenção individuais aprendem claramente a realizar tarefas diferentes, mas muitos parecem exibir comportamento relacionado à estrutura sintática e semântica das frases.

5 Treinamento

Esta seção descreve o regime de treinamento para nossos modelos.

5.1 Dados de Treinamento e Loteamento

Treinamos no conjunto de dados WMT 2014 padrão Inglês-Alemão, composto por cerca de 4,5 milhões de pares de frases. As frases foram codificadas usando codificação byte-pair [3], que possui um vocabulário compartilhado fonte-destino de cerca de 37.000 tokens. Para Inglês-Francês, utilizamos o conjunto de dados WMT 2014 Inglês-Francês, significativamente maior, composto por 36 milhões de frases, e dividimos os tokens em um vocabulário word-piece de 32.000 elementos [38]. Os pares de frases foram agrupados em lotes com base no comprimento aproximado da sequência. Cada lote de treinamento continha um conjunto de pares de frases contendo aproximadamente 25.000 tokens de origem e 25.000 tokens de destino.

5.2 Hardware e Cronograma

Treinamos nossos modelos em uma máquina com 8 GPUs NVIDIA P100. Para nossos modelos base usando os hiperparâmetros descritos ao longo do artigo, cada etapa de treinamento levou cerca de 0,4 segundos. Treinamos os modelos base por um total de 100.000 etapas ou 12 horas. Para nossos grandes modelos (descritos na última linha da tabela 3), o tempo por etapa foi de 1,0 segundo. Os grandes modelos foram treinados por 300.000 etapas (3,5 dias).

5.3 Otimizador

Utilizamos o otimizador Adam [20] com $\beta_1 = 0,9$, $\beta_2 = 0,98$ e $\varepsilon = 10^{-9}$. Variamos a taxa de aprendizado ao longo do treinamento, de acordo com a fórmula:

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step_num^{-0.5}, step_num \cdot warmup_steps^{-1.5}) \quad (3)$$

Isso corresponde a aumentar a taxa de aprendizado linearmente para os primeiros warmup_steps passos de treinamento, e diminuí-la posteriormente proporcionalmente à raiz quadrada inversa do número do passo. Usamos warmup_steps = 4000.

5.4 Regularização

Empregamos três tipos de regularização durante o treinamento:

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

Residual Dropout We apply dropout [33] to the output of each sub-layer, before it is added to the sub-layer input and normalized. In addition, we apply dropout to the sums of the embeddings and the positional encodings in both the encoder and decoder stacks. For the base model, we use a rate of $P_{drop} = 0.1$.

Label Smoothing During training, we employed label smoothing of value $\epsilon_{ls} = 0.1$ [36]. This hurts perplexity, as the model learns to be more unsure, but improves accuracy and BLEU score.

6 Results

6.1 Machine Translation

On the WMT 2014 English-to-German translation task, the big transformer model (Transformer (big) in Table 2) outperforms the best previously reported models (including ensembles) by more than 2.0 BLEU, establishing a new state-of-the-art BLEU score of 28.4. The configuration of this model is listed in the bottom line of Table 3. Training took 3.5 days on 8 P100 GPUs. Even our base model surpasses all previously published models and ensembles, at a fraction of the training cost of any of the competitive models.

On the WMT 2014 English-to-French translation task, our big model achieves a BLEU score of 41.0, outperforming all of the previously published single models, at less than $1/4$ the training cost of the previous state-of-the-art model. The Transformer (big) model trained for English-to-French used dropout rate $P_{drop} = 0.1$, instead of 0.3.

For the base models, we used a single model obtained by averaging the last 5 checkpoints, which were written at 10-minute intervals. For the big models, we averaged the last 20 checkpoints. We used beam search with a beam size of 4 and length penalty $\alpha = 0.6$ [38]. These hyperparameters were chosen after experimentation on the development set. We set the maximum output length during inference to input length + 50, but terminate early when possible [38].

Table 2 summarizes our results and compares our translation quality and training costs to other model architectures from the literature. We estimate the number of floating point operations used to train a model by multiplying the training time, the number of GPUs used, and an estimate of the sustained single-precision floating-point capacity of each GPU [5].

6.2 Model Variations

To evaluate the importance of different components of the Transformer, we varied our base model in different ways, measuring the change in performance on English-to-German translation on the

[5] We used values of 2.8, 3.7, 6.0 and 9.5 TFLOPS for K80, K40, M40 and P100, respectively.

Tabela 2: O Transformer obtém melhores escores BLEU do que modelos anteriores state-of-the-art nos testes English-to-German e English-to-French newstest2014, a uma fração do custo de treinamento.

Modelo	BLEU		Custo de Treinamento (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 (modelo base)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (grande)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Dropout Residual Aplicamos dropout [33] à saída de cada subcamada, antes que ela seja somada à entrada da subcamada e normalizada. Além disso, aplicamos dropout às somas dos embeddings e dos codificadores posicionais tanto nas pilhas do encoder quanto do decoder. Para o modelo base, utilizamos uma taxa de $P_{drop} = 0.1$.

Suavização de Rótulos Durante o treinamento, employamos suavização de rótulos com valor $\varepsilon_{ls} = 0.1$ [36]. Isso prejudica a perplexidade, pois o modelo aprende a ser mais inseguro, mas melhora a precisão e a pontuação BLEU.

6 Resultados

6.1 Tradução Automática

Na tarefa de tradução do inglês para o alemão WMT 2014, o grande modelo transformer (Transformer (big) na Tabela 2) supera os melhores modelos anteriormente reportados (incluindo ensembles) em mais de 2.0 BLEU, estabelecendo uma nova pontuação state-of-the-art BLEU de 28.4. A configuração deste modelo está listada na linha inferior da Tabela 3. O treinamento levou 3.5 dias em 8 GPUs P100. Mesmo nosso modelo base supera todos os modelos e ensembles publicados anteriormente, a uma fração do custo de treinamento de qualquer um dos modelos concorrentes.

Na tarefa de tradução do inglês para o francês do WMT 2014, nosso modelo grande alcança uma pontuação BLEU de 41,0, superando todos os modelos single anteriores publicados, a menos de 1/4 do custo de treinamento do modelo state-of-the-art anterior. O modelo Transformer (grande) treinado para inglês para francês utilizou taxa de dropout $P_{drop} = 0,1$, em vez de 0,3.

Para os modelos base, utilizamos um único modelo obtido pela média dos últimos 5 checkpoints, que foram escritos em intervalos de 10 minutos. Para os grandes modelos, médiamos os últimos 20 checkpoints. Utilizamos busca em feixe com tamanho de feixe de 4 e penalidade de comprimento $\alpha = 0.6$ [38]. Esses hiperparâmetros foram escolhidos após experimentação no conjunto de desenvolvimento. Definimos o comprimento máximo de saída durante a inferência como comprimento de entrada + 50, mas terminamos precocemente quando possível [38].

A Tabela 2 resume nossos resultados e compara nossa qualidade de tradução e custos de treinamento com outras arquiteturas de modelo da literatura. Estimamos o número de operações de ponto flutuante usadas para treinar um modelo multiplicando o tempo de treinamento, o número de GPUs usadas e uma estimativa da capacidade sustentada de ponto flutuante de precisão simples de cada GPU [5].

6.2 Variações do Modelo

Para avaliar a importância de diferentes componentes do Transformer, variamos nosso modelo base de diferentes maneiras, medindo a mudança no desempenho na tradução do inglês para o alemão na

[5] Utilizamos valores de 2,8, 3,7, 6,0 e 9,5 TFLOPS para K80, K40, M40 e P100, respectivamente.

Table 3: Variations on the Transformer architecture. Unlisted values are identical to those of the base model. All metrics are on the English-to-German translation development set, newstest2013. Listed perplexities are per-wordpiece, according to our byte-pair encoding, and should not be compared to per-word perplexities.

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

development set, newstest2013. We used beam search as described in the previous section, but no checkpoint averaging. We present these results in Table 3.

In Table 3 rows (A), we vary the number of attention heads and the attention key and value dimensions, keeping the amount of computation constant, as described in Section 3.2.2. While single-head attention is 0.9 BLEU worse than the best setting, quality also drops off with too many heads.

In Table 3 rows (B), we observe that reducing the attention key size d_k hurts model quality. This suggests that determining compatibility is not easy and that a more sophisticated compatibility function than dot product may be beneficial. We further observe in rows (C) and (D) that, as expected, bigger models are better, and dropout is very helpful in avoiding over-fitting. In row (E) we replace our sinusoidal positional encoding with learned positional embeddings [9], and observe nearly identical results to the base model.

6.3 English Constituency Parsing

To evaluate if the Transformer can generalize to other tasks we performed experiments on English constituency parsing. This task presents specific challenges: the output is subject to strong structural constraints and is significantly longer than the input. Furthermore, RNN sequence-to-sequence models have not been able to attain state-of-the-art results in small-data regimes [37].

We trained a 4-layer transformer with $d_{model} = 1024$ on the Wall Street Journal (WSJ) portion of the Penn Treebank [25], about 40K training sentences. We also trained it in a semi-supervised setting, using the larger high-confidence and BerkleyParser corpora from with approximately 17M sentences [37]. We used a vocabulary of 16K tokens for the WSJ only setting and a vocabulary of 32K tokens for the semi-supervised setting.

We performed only a small number of experiments to select the dropout, both attention and residual (section 5.4), learning rates and beam size on the Section 22 development set, all other parameters remained unchanged from the English-to-German base translation model. During inference, we

Tabela 3: Variações na arquitetura Transformer. Valores não listados são idênticos aos do modelo base. Todas as métricas estão no conjunto de desenvolvimento de tradução do inglês para o alemão, newstest2013. As perplexidades listadas são por wordpiece, de acordo com nossa codificação byte-pair, e não devem ser comparadas a perplexidades por palavra.

	N	dmodel	dff	h	dk	dv	Pdrop	ϵ_{ls}	treino etapas	PPL (dev)	BLEU (dev)	parâmetros $\times 10^6$
base	6	512	2048	8	64	64	0,1	0,1	100K	4,92	25,8	65
(A)				1	512	512				5,29	24,9	
				4	128	128				5,00	25,5	
				16	32	32				4,91	25,8	
				32	16	16				5,01	25,4	
(B)					16					5,16	25,1	58
					32					5,01	25,4	60
(C)	2									6,11	23,7	36
	4									5,19	25,3	50
	8									4,88	25,5	80
		256			32	32				5,75	24,5	28
		1024			128	128				4,66	26,0	168
			1024							5,12	25,4	53
			4096							4,75	26,2	90
(D)							0,0			5,77	24,6	
							0,2			4,95	25,5	
								0,0		4,67	25,3	
								0,2		5,47	25,7	
(E)	incorporação posicional em vez de senoides									4,92	25,7	
grande	6	1024	4096	16			0,3		300K	4,33	26,4	213

conjunto de desenvolvimento, newstest2013. Utilizamos busca em feixe conforme descrito na seção anterior, mas sem média de checkpoints. Apresentamos esses resultados na Tabela 3.

Na Tabela 3, linhas (A), variamos o número de cabeças de atenção e as dimensões da chave e do valor de atenção, mantendo a quantidade de computação constante, conforme descrito na Seção 3.2.2. Embora a atenção de uma única cabeça seja 0,9 BLEU pior que a melhor configuração, a qualidade também cai com muitas cabeças.

Na Tabela 3, linhas (B), observamos que reduzir o tamanho da chave de atenção d_k prejudica a qualidade do modelo. Isso sugere que determinar a compatibilidade não é fácil e que uma função de compatibilidade mais sofisticada do que o produto escalar pode ser benéfica. Observamos ainda nas linhas (C) e (D) que, conforme esperado, modelos maiores são melhores, e o dropout é muito útil para evitar overfitting. Na linha (E), substituímos nosso codificação posicional senoidal por embeddings posicionais aprendidos [9], e observamos resultados quase idênticos aos do modelo base.

6.3 Análise Sintática de Constituente em Inglês

Para avaliar se o Transformer pode generalizar para outras tarefas, realizamos experimentos de análise sintática de constituintes em inglês. Esta tarefa apresenta desafios específicos: a saída está sujeita a fortes restrições estruturais e é significativamente mais longa que a entrada. Além disso, modelos RNN de sequência-para-sequência ainda não conseguiram atingir resultados state-of-the-art em regimes de dados limitados [37].

Treinamos um transformador de 4 camadas com $d_{model} = 1024$ na porção do Wall Street Journal (WSJ) do Penn Treebank [25], cerca de 40K frases de treinamento. Também o treinamos em um cenário semi-supervisionado, utilizando os maiores corpora de alta confiança e BerkleyParser, com aproximadamente 17M de frases [37]. Utilizamos um vocabulário de 16K tokens para o cenário apenas WSJ e um vocabulário de 32K tokens para o cenário semi-supervisionado.

Realizamos apenas um pequeno número de experimentos para selecionar o dropout, tanto de atenção quanto residual (seção 5.4), taxas de aprendizado e tamanho do feixe no conjunto de desenvolvimento da Seção 22; todos os outros parâmetros permaneceram inalterados em relação ao modelo base de tradução do inglês para o alemão. Durante a inferência, nós

Table 4: The Transformer generalizes well to English constituency parsing (Results are on Section 23 of WSJ)

Parser	Training	WSJ 23 F1
Vinyals & Kaiser el al. (2014) [37]	WSJ only, discriminative	88.3
Petrov et al. (2006) [29]	WSJ only, discriminative	90.4
Zhu et al. (2013) [40]	WSJ only, discriminative	90.4
Dyer et al. (2016) [8]	WSJ only, discriminative	91.7
Transformer (4 layers)	WSJ only, discriminative	91.3
Zhu et al. (2013) [40]	semi-supervised	91.3
Huang & Harper (2009) [14]	semi-supervised	91.3
McClosky et al. (2006) [26]	semi-supervised	92.1
Vinyals & Kaiser el al. (2014) [37]	semi-supervised	92.1
Transformer (4 layers)	semi-supervised	92.7
Luong et al. (2015) [23]	multi-task	93.0
Dyer et al. (2016) [8]	generative	93.3

increased the maximum output length to input length + 300. We used a beam size of 21 and $\alpha = 0.3$ for both WSJ only and the semi-supervised setting.

Our results in Table 4 show that despite the lack of task-specific tuning our model performs surprisingly well, yielding better results than all previously reported models with the exception of the Recurrent Neural Network Grammar [8].

In contrast to RNN sequence-to-sequence models [37], the Transformer outperforms the Berkeley-Parser [29] even when training only on the WSJ training set of 40K sentences.

7 Conclusion

In this work, we presented the Transformer, the first sequence transduction model based entirely on attention, replacing the recurrent layers most commonly used in encoder-decoder architectures with multi-headed self-attention.

For translation tasks, the Transformer can be trained significantly faster than architectures based on recurrent or convolutional layers. On both WMT 2014 English-to-German and WMT 2014 English-to-French translation tasks, we achieve a new state of the art. In the former task our best model outperforms even all previously reported ensembles.

We are excited about the future of attention-based models and plan to apply them to other tasks. We plan to extend the Transformer to problems involving input and output modalities other than text and to investigate local, restricted attention mechanisms to efficiently handle large inputs and outputs such as images, audio and video. Making generation less sequential is another research goals of ours.

The code we used to train and evaluate our models is available at `https://github.com/tensorflow/tensor2tensor`.

Acknowledgements We are grateful to Nal Kalchbrenner and Stephan Gouws for their fruitful comments, corrections and inspiration.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[3] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc V. Le. Massive exploration of neural machine translation architectures. *CoRR*, abs/1703.03906, 2017.

[4] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*, 2016.

Tabela 4: O Transformer generaliza bem para a análise sintática de constituintes em inglês (Os resultados são da Seção 23 do WSJ)

Analisador	Treinamento	WSJ 23 F1
Vinyals & Kaiser et al. (2014) [37]	Somente WSJ, discriminativo	88.3
Petrov et al. (2006) [29]	Somente WSJ, discriminativo	90.4
Zhu et al. (2013) [40]	Somente WSJ, discriminativo	90.4
Dyer et al. (2016) [8]	Somente WSJ, discriminativo	91.7
Transformer (4 camadas)	Somente WSJ, discriminativo	91.3
Zhu et al. (2013) [40]	semi-supervisionado	91.3
Huang & Harper (2009) [14]	semi-supervisionado	91.3
McClosky et al. (2006) [26]	semi-supervisionado	92.1
Vinyals & Kaiser et al. (2014) [37]	semi-supervisionado	92.1
Transformer (4 camadas)	semi-supervisionado	92.7
Luong et al. (2015) [23]	multitarefa	93.0
Dyer et al. (2016) [8]	generativo	93.3

aumentou o comprimento máximo de saída para comprimento de entrada + 300. Utilizamos um tamanho de feixe de 21 e α=0.3 tanto para o cenário apenas WSJ quanto para o cenário semi-supervisionado.

Nossos resultados na Tabela 4 mostram que, apesar da ausência de ajuste específico para a tarefa, nosso modelo apresenta desempenho surpreendentemente bom, obtendo resultados superiores aos de todos os modelos anteriormente reportados, com exceção da Rede Neural Recorrente Gramatical [8].

Em contraste com os modelos RNN de sequência-para-sequência [37], o Transformer supera o Berkeley-Parser [29] mesmo quando treinado apenas no conjunto de treinamento WSJ de 40K frases.

7 Conclusão

Neste trabalho, apresentamos o Transformer, o primeiro modelo de transdução de sequências baseado inteiramente em atenção, substituindo as camadas recorrentes mais comumente usadas em arquiteturas de codificador-decodificador por autoatenção de múltiplas cabeças.

Para tarefas de tradução, o Transformer pode ser treinado significativamente mais rápido do que arquiteturas baseadas em camadas recorrentes ou convolucionais. Em ambas as tarefas de tradução WMT 2014 Inglês-para-Alemão e WMT 2014 Inglês-para-Francês, alcançamos um novo estado da arte. Na primeira tarefa, nosso melhor modelo supera até mesmo todos os ensembles previamente relatados.

Estamos entusiasmados com o futuro dos modelos baseados em atenção e planejamos aplicá-los em outras tarefas. Planejamos estender o Transformer a problemas que envolvem modalidades de entrada e saída diferentes do texto e investigar mecanismos de atenção locais e restritos para lidar eficientemente com grandes entradas e saídas, como imagens, áudio e vídeo. Tornar a geração menos sequencial é outro objetivo de pesquisa nosso.

O código que utilizamos para treinar e avaliar nossos modelos está disponível em https://github.com/tensorflow/tensor2tensor.

Agradecimentos Somos gratos a Nal Kalchbrenner e Stephan Gouws por seus comentários, correções e inspiração frutíferos.

Referências

[1] Jimmy Lei Ba, Jamie Ryan Kiros e Geoffrey E Hinton. Normalização de camada. arXiv preprint arXiv:1607.06450, 2016.

[2] Dzmitry Bahdanau, Kyunghyun Cho, e Yoshua Bengio. Tradução de máquina neural aprendendo conjuntamente a alinhar e traduzir. CoRR, abs/1409.0473, 2014.

[3] Denny Britz, Anna Goldie, Minh-Thang Luong e Quoc V. Le. Exploração massiva de arquiteturas de tradução automática neural. CoRR, abs/1703.03906, 2017.

[4] Jianpeng Cheng, Li Dong e Mirella Lapata. Redes de memória de curto e longo prazo para leitura de máquina. arXiv preprint arXiv:1601.06733, 2016.

[5] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[6] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016.

[7] Junyoung Chung, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[8] Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. Recurrent neural network grammars. In *Proc. of NAACL*, 2016.

[9] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122v2*, 2017.

[10] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[12] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[14] Zhongqiang Huang and Mary Harper. Self-training PCFG grammars with latent annotations across languages. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 832–841. ACL, August 2009.

[15] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

[16] Łukasz Kaiser and Samy Bengio. Can active memory replace attention? In *Advances in Neural Information Processing Systems, (NIPS)*, 2016.

[17] Łukasz Kaiser and Ilya Sutskever. Neural GPUs learn algorithms. In *International Conference on Learning Representations (ICLR)*, 2016.

[18] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099v2*, 2017.

[19] Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. Structured attention networks. In *International Conference on Learning Representations*, 2017.

[20] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[21] Oleksii Kuchaiev and Boris Ginsburg. Factorization tricks for LSTM networks. *arXiv preprint arXiv:1703.10722*, 2017.

[22] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*, 2017.

[23] Minh-Thang Luong, Quoc V. Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*, 2015.

[24] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[5] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk e Yoshua Bengio. Aprendendo representações de frases usando codificador-decodificador rnn para tradução automática estatística. CoRR, abs/1406.1078, 2014.

[6] Francois Chollet. Xception: Deep learning com convoluções separáveis por profundidade. arXiv preprint arXiv:1610.02357, 2016.

[7] Junyoung Chung, Çaglar Gülçehre, Kyunghyun Cho, e Yoshua Bengio. Avaliação empírica de redes neurais recorrentes com porta em modelagem de sequências. CoRR, abs/1412.3555, 2014.

[8] Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros e Noah A. Smith. Gramáticas de redes neurais recorrentes. In Proc. of NAACL, 2016.

[9] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats e Yann N. Dauphin. Aprendizado de sequência para sequência convolucional. arXiv preprint arXiv:1705.03122v2, 2017.

[10] Alex Graves. Gerando sequências com redes neurais recorrentes. arXiv preprint arXiv:1308.0850, 2013.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, e Jian Sun. Aprendizado residual profundo para reconhecimento de imagens. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, páginas 770 – 778, 2016.

[12] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi e Jürgen Schmidhuber. Fluxo de gradiente em redes recorrentes: a dificuldade de aprender dependências de longo prazo, 2001.

[13] Sepp Hochreiter e Jürgen Schmidhuber. Memória de longo prazo de curto termo. Cálculo neural, 9(8):1735 – 1780, 1997.

[14] Zhongqiang Huang e Mary Harper. Gramáticas PCFG de autoaprendizado com anotações latentes através de línguas. In Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, páginas 832 – 841. ACL, agosto de 2009.

[15] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer e Yonghui Wu. Explorando os limites da modelagem de linguagem. arXiv preprint arXiv:1602.02410, 2016.

[16] Łukasz Kaiser e Samy Bengio. A memória ativa pode substituir a atenção? In Advances in Neural Information Processing Systems, (NIPS), 2016.

[17] Łukasz Kaiser e Ilya Sutskever. GPUs neurais aprendem algoritmos. In International Conference on Learning Representations (ICLR), 2016.

[18] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, e Koray Kavukcuoglu. Tradução automática neural em tempo linear. arXiv preprint arXiv:1610.10099v2, 2017.

[19] Yoon Kim, Carl Denton, Luong Hoang e Alexander M. Rush. Redes de atenção estruturadas. In International Conference on Learning Representations, 2017.

[20] Diederik Kingma e Jimmy Ba. Adam: Um método para otimização estocástica. In ICLR, 2015.

[21] Oleksii Kuchaiev e Boris Ginsburg. Truques de fatoração para redes LSTM. arXiv preprint arXiv:1703.10722, 2017.

[22] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou e Yoshua Bengio. Um embedding de frase com autoatenção estruturada. arXiv preprint arXiv:1703.03130, 2017.

[23] Minh-Thang Luong, Quoc V. Le, Ilya Sutskever, Oriol Vinyals e Lukasz Kaiser. Aprendizado de sequência para sequência multi-tarefa. arXiv preprint arXiv:1511.06114, 2015.

[24] Minh-Thang Luong, Hieu Pham e Christopher D Manning. Abordagens eficazes para tradução automática neural baseada em atenção. arXiv preprint arXiv:1508.04025, 2015.

[25] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[26] David McClosky, Eugene Charniak, and Mark Johnson. Effective self-training for parsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 152–159. ACL, June 2006.

[27] Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model. In *Empirical Methods in Natural Language Processing*, 2016.

[28] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*, 2017.

[29] Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL*, pages 433–440. ACL, July 2006.

[30] Ofir Press and Lior Wolf. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*, 2016.

[31] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[32] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[33] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[34] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2440–2448. Curran Associates, Inc., 2015.

[35] Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

[36] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[37] Vinyals & Kaiser, Koo, Petrov, Sutskever, and Hinton. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, 2015.

[38] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[39] Jie Zhou, Ying Cao, Xuguang Wang, Peng Li, and Wei Xu. Deep recurrent models with fast-forward connections for neural machine translation. *CoRR*, abs/1606.04199, 2016.

[40] Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st Annual Meeting of the ACL (Volume 1: Long Papers)*, pages 434–443. ACL, August 2013.

[25] Mitchell P Marcus, Mary Ann Marcinkiewicz e Beatrice Santorini. Construindo um grande corpus anotado de inglês: O Penn Treebank. Linguística computacional, 19(2):313 – 330, 1993.

[26] David McClosky, Eugene Charniak e Mark Johnson. Autoentlycagem eficaz para análise sintática. In: Anais da Conferência de Tecnologia da Linguagem Humana da NAACL, Conferência Principal, páginas 152 – 159. ACL, junho de 2006.

[27] Ankur Parikh, Oscar Täckström, Dipanjan Das e Jakob Uszkoreit. Um modelo de atenção decomponível. In Empirical Methods in Natural Language Processing, 2016.

[28] Romain Paulus, Caiming Xiong e Richard Socher. Um modelo de reforço profundo para sumarização abstrativa. arXiv preprint arXiv:1705.04304, 2017.

[29] Slav Petrov, Leon Barrett, Romain Thibaux e Dan Klein. Aprendizado de anotação de árvores precisa, compacta e interpretável. In Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL, páginas 433 – 440. ACL, julho de 2006.

[30] Ofir Press e Lior Wolf. Usando o embedding de saída para melhorar modelos de linguagem. arXiv preprint arXiv:1608.05859, 2016.

[31] Rico Sennrich, Barry Haddow e Alexandra Birch. Tradução automática neural de palavras raras com unidades de subpalavra. arXiv preprint arXiv:1508.07909, 2015.

[32] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton e Jeff Dean. Redes neurais absurdamente grandes: A camada de mistura de especialistas com porta esparsa. arXiv preprint arXiv:1701.06538, 2017.

[33] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever e Ruslan Salakhutdinov. Dropout: uma maneira simples de evitar o sobreajuste em redes neurais. Journal of Machine Learning Research, 15(1):1929 – 1958, 2014.

[34] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston e Rob Fergus. Redes de memória end-to-end. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama e R. Garnett, editores, Advances in Neural Information Processing Systems 28, páginas 2440 – 2448. Curran Associates, Inc., 2015.

[35] Ilya Sutskever, Oriol Vinyals e Quoc VV Le. Aprendizado de sequência para sequência com redes neurais. In Advances in Neural Information Processing Systems, páginas 3104 – 3112, 2014.

[36] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens e Zbigniew Wojna. Repensando a arquitetura inception para visão computacional. CoRR, abs/1512.00567, 2015.

[37] Vinyals & Kaiser, Koo, Petrov, Sutskever e Hinton. Gramática como uma língua estrangeira. In: Advances in Neural Information Processing Systems, 2015.

[38] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. O sistema de tradução neural do Google: Fechando a lacuna entre a tradução humana e a máquina. arXiv preprint arXiv:1609.08144, 2016.

[39] Jie Zhou, Ying Cao, Xuguang Wang, Peng Li e Wei Xu. Modelos recorrentes profundos com conexões de avanço rápido para tradução automática neural. CoRR, abs/1606.04199, 2016.

[40] Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang e Jingbo Zhu. Análise sintática constituinte de deslocamento-redução rápida e precisa. In Proceedings of the 51st Annual Meeting of the ACL (Volume 1: Long Papers), páginas 434 – 443. ACL, agosto de 2013.

Attention Visualizations

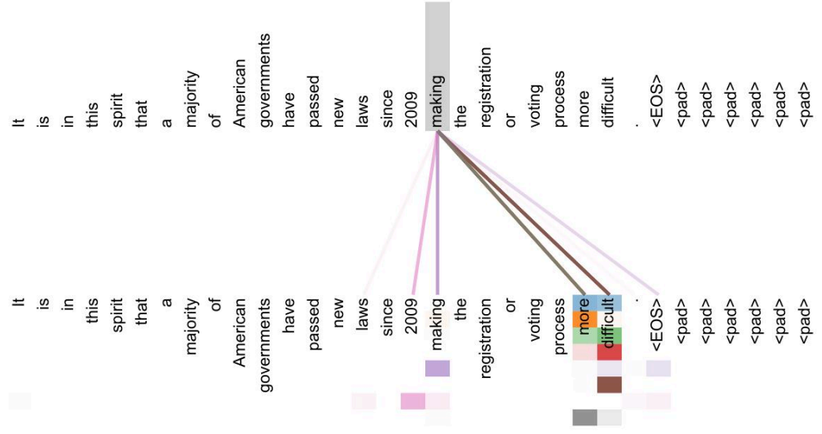


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb ‘making’, completing the phrase ‘making...more difficult’. Attentions here shown only for the word ‘making’. Different colors represent different heads. Best viewed in color.

Visualizações de Atenção

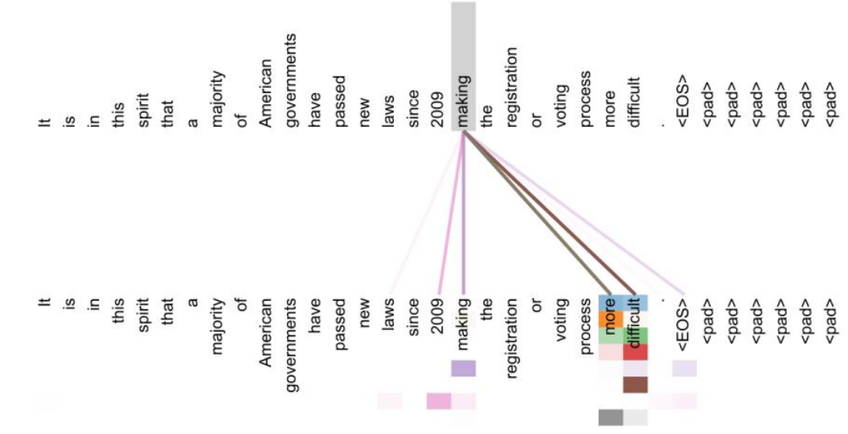
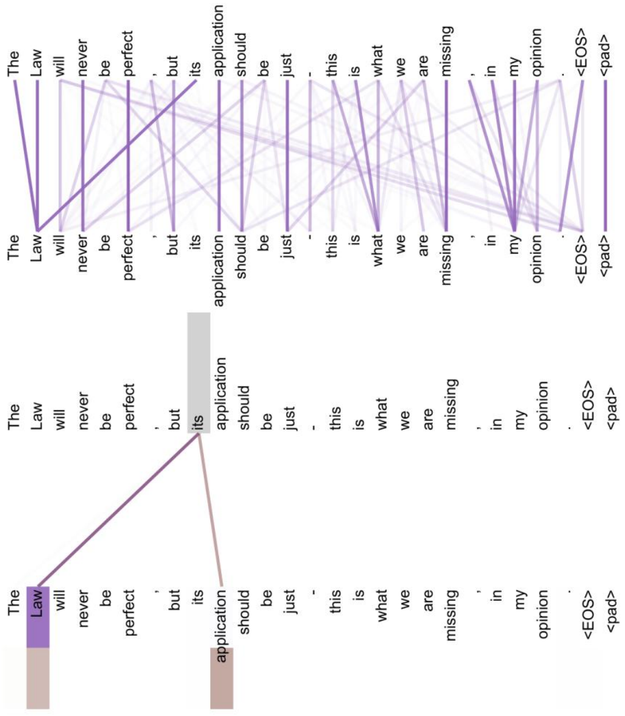


Figura 3: Um exemplo do mecanismo de atenção seguindo dependências de longa distância no encoder de autoatenção na camada 5 de 6. Muitas das cabeças de atenção atendem a uma dependência distante do verbo 'making' (produzindo/fazendo), completando a frase 'making...more difficult' (tornando...mais difícil). As atenções aqui mostradas são apenas para a palavra 'making'. Cores diferentes representam cabeças diferentes. Melhor visualizado em cores.



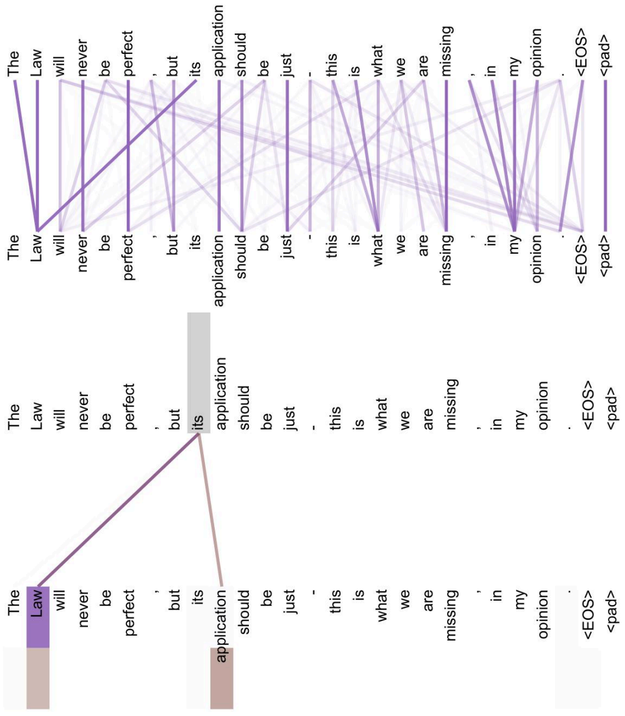


Figure 4: Two attention heads, also in layer 5 of 6, apparently involved in anaphora resolution. Top: Full attentions for head 5. Bottom: Isolated attentions from just the word 'its' for attention heads 5 and 6. Note that the attentions are very sharp for this word.

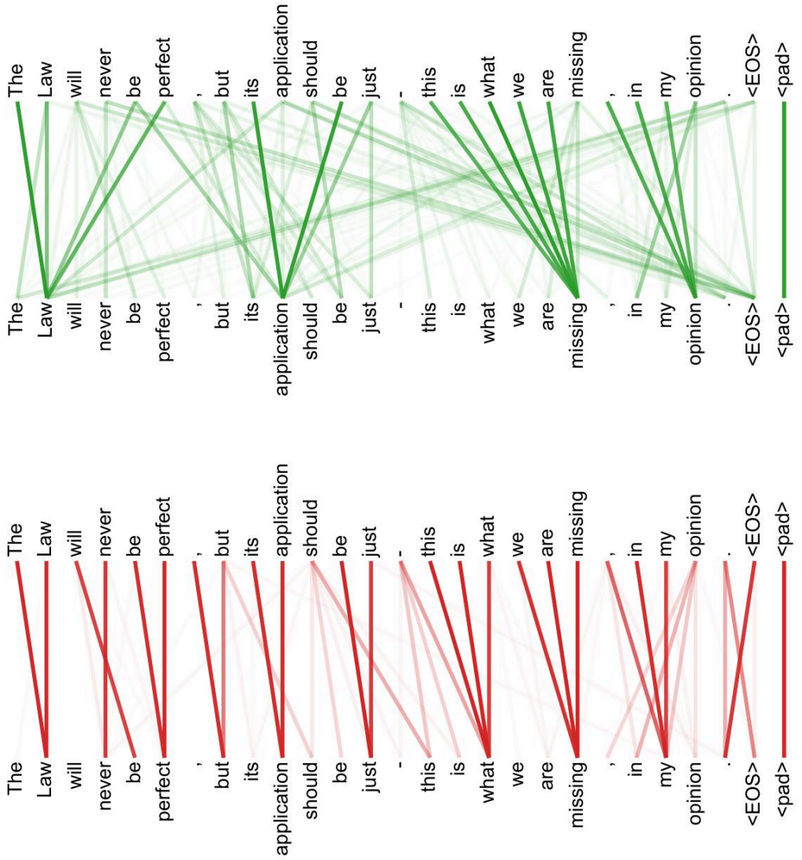


Figure 5: Many of the attention heads exhibit behaviour that seems related to the structure of the sentence. We give two such examples above, from two different heads from the encoder self-attention at layer 5 of 6. The heads clearly learned to perform different tasks.

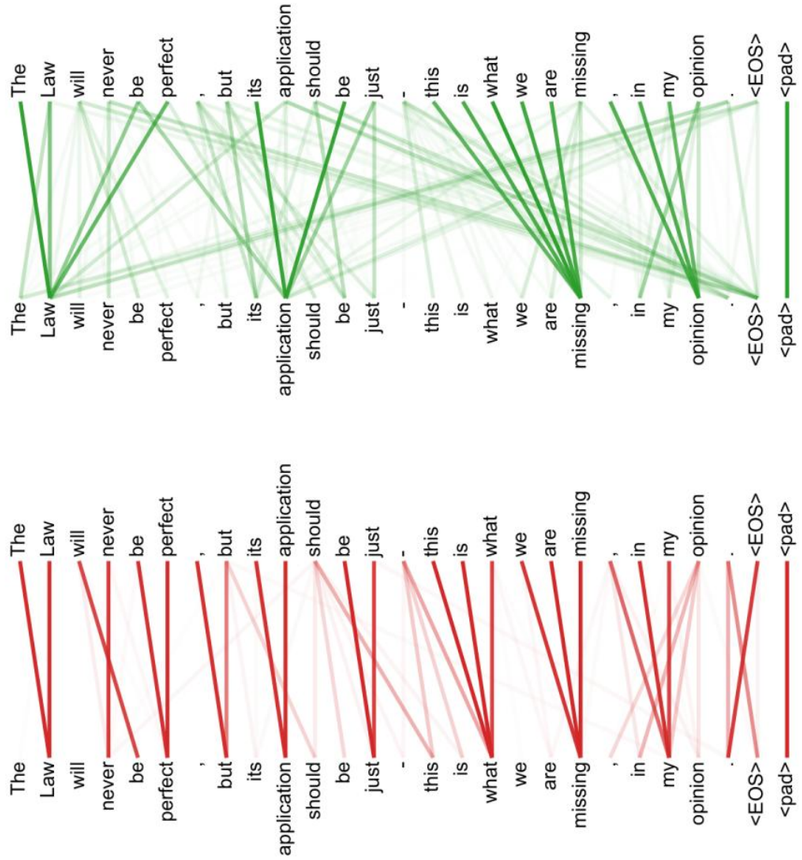


Figura 5: Muitas das cabeças de atenção exibem comportamento que parece estar relacionado à estrutura da frase. Apresentamos dois exemplos desses casos acima, provenientes de duas cabeças diferentes da autoatenção do encoder na camada 5 de 6. As cabeças aprenderam claramente a realizar tarefas distintas.