Atenção é Tudo o que Você Precisa

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

Resumo

Os modelos dominantes de transdução de sequência são baseados em redes neurais recorrentes ou convolucionais complexas que incluem um codificador e um decodificador. Os modelos com melhor desempenho também conectam o codificador e o decodificador através de um mecanismo de atenção. Propomos uma nova arquitetura de rede simples, o Transformer, baseada exclusivamente em mecanismos de atenção, dispensando totalmente a recorrência e as convoluções. Experimentos em duas tarefas de tradução automática mostram que esses modelos são superiores em qualidade, além de serem mais paralelizáveis e exigirem significativamente menos tempo para treinar. Nosso modelo atinge 28.4 BLEU na tarefa de tradução Inglês-para-Alemão do WMT 2014, superando os melhores resultados existentes, incluindo ensembles, em mais de 2 BLEU. Na tarefa de tradução Inglês-para-Francês do WMT 2014, nosso modelo estabelece um novo estado da arte para um único modelo com uma pontuação BLEU de 41.8 após treinar por 3.5 dias 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 de constituintes em inglês, tanto com dados de treinamento grandes quanto limitados.

*Contribuição igual. A ordem da listagem é aleatória. Jakob propôs substituir RNNs por auto-atenção 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 escalar em escala, a atenção multi-cabeça e a representação de posição livre de parâmetros e se tornou a outra pessoa envolvida em quase todos os detalhes. Niki projetou, implementou, ajustou e avaliou inúmeras variantes de modelo em nosso código original e no tensor2tensor. Llion também experimentou com novas variantes de modelo, foi responsável pelo nosso código inicial e pela inferência e visualizações eficientes. Lukasz e Aidan passaram inúmeros longos dias projetando várias partes e implementando o tensor2tensor, substituindo nosso código anterior, melhorando muito os resultados e acelerando massivamente nossa pesquisa.
†Trabalho realizado enquanto estava no Google Brain.
‡Trabalho realizado enquanto estava no Google Research.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.
arXiv:1706.03762v7 [cs.CL] 2 Aug 2023

1 Introdução

Redes neurais recorrentes, em particular a memória de longo e curto prazo (long short-term memory) [**?**] e redes neurais recorrentes com gates (gated recurrent) [**?**], foram firmemente estabelecidas como abordagens de estado da arte em modelagem de sequências e problemas de transdução, como modelagem de linguagem e

tradução automática [**?**, **?**, **?**]. Inúmeros esforços continuaram desde então a expandir os limites dos modelos de linguagem recorrentes e arquiteturas codificador-decodificador [**?**, **?**, **?**].

Modelos recorrentes tipicamente fatoram a computação ao longo das posições dos símbolos das sequências de entrada e saída. Alinhando as posições aos passos no tempo de computação, eles geram uma sequência de estados ocultos h_t, como uma função do estado oculto anterior h_{t-1} e da entrada para a posição t. Essa natureza inerentemente sequencial impede a paralelização dentro dos exemplos de treinamento, o que se torna crítico em comprimentos de sequência mais longos, pois as restrições de memória limitam o processamento em lote (batching) entre os exemplos. Trabalhos recentes alcançaram melhorias significativas na eficiência computacional através de truques de fatorização [**?**] e computação condicional [**?**], ao mesmo tempo em que melhoraram o desempenho do modelo no caso deste último. A restrição fundamental da computação sequencial, no entanto, permanece.

Mecanismos de atenção tornaram-se parte integrante de modelos convincentes de modelagem de sequências e transdução em várias tarefas, permitindo a modelagem de dependências sem levar em conta sua distância nas sequências de entrada ou saída [**?**, **?**]. Em todos, exceto alguns casos [**?**], no entanto, tais mecanismos de atenção são usados em conjunto com uma rede recorrente.

Neste trabalho, propomos o Transformer, uma arquitetura de modelo que evita a recorrência e, em vez disso, depende inteiramente de um mecanismo de atenção para extrair dependências globais entre entrada e saída. O Transformer permite significativamente mais paralelização e pode alcançar um novo estado da arte em qualidade de tradução após ser treinado por apenas doze horas em oito GPUs P100.

2 Contexto

O objetivo de reduzir a computação sequencial também forma a base da Extended Neural GPU [**?**], ByteNet [**?**] e ConvS2S [**?**], todos os quais usam redes neurais convolucionais como bloco de construção básico, computando 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 arbitrárias de entrada ou saída 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 [**?**]. No Transformer, isso é reduzido a um número constante de operações, embora ao custo de uma resolução efetiva reduzida devido à média das posições ponderadas pela atenção, um efeito que combatemos com a Atenção Multi-Cabeça, conforme descrito na seção 3.2.

A auto-atenção, às vezes chamada de intra-atenção, é um mecanismo de atenção que relaciona diferentes posições de uma única sequência para computar uma representação da sequência. A auto-atenção tem sido usada com sucesso em uma variedade de tarefas, incluindo compreensão de leitura, sumarização abstrativa, inferência textual e aprendizado de representações de sentenças independentes da tarefa [**?**, **?**, **?**, **?**].

Redes de memória de ponta a ponta (End-to-end memory networks) são baseadas em um mecanismo de atenção recorrente em vez de recorrência alinhada à sequência e demonstraram um bom desempenho em tarefas de resposta a perguntas em linguagem simples e modelagem de linguagem [**?**].

Até onde sabemos, no entanto, o Transformer é o primeiro modelo de transdução que depende inteiramente da auto-atenção para computar representações de sua entrada e saída sem usar RNNs alinhados à sequência ou convolução. Nas seções seguintes, descreveremos o Transformer, motivaremos a auto-atenção e discutiremos suas vantagens sobre modelos como [**?**, **?**] e [**?**].

3 Arquitetura do Modelo

A maioria dos modelos de transdução de sequência neural competitivos tem uma estrutura codificador-decodificador [**?**, **?**, **?**]. Aqui, o codificador mapeia uma sequência de entrada de representações de símbolos $(x_1, \ldots, x_n)$ para uma sequência de representações contínuas $z = (z_1, \ldots, z_n)$. Dado z, o decodificador então gera uma sequência de saída $(y_1, \ldots, y_m)$ de símbolos um elemento de cada vez. A cada passo, o modelo é auto-regressivo [**?**], consumindo os símbolos gerados anteriormente como entrada adicional ao gerar o próximo.

[Imagem removida]
Figura 1: A arquitetura do Transformer.

O Transformer segue essa arquitetura geral usando auto-atenção empilhada e camadas totalmente conectadas ponto a ponto (point-wise) para o codificador e o decodificador, mostrados 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 tem duas sub-camadas. A primeira é um mecanismo de auto-atenção multi-cabeça, e a segunda é uma rede feed-forward simples, totalmente conectada e posicional (position-wise). Empregamos uma conexão residual [**?**] em torno de cada uma das duas sub-camadas, seguida por normalização de camada [**?**]. Ou seja, a saída de cada sub-camada é $\text{LayerNorm}(x + \text{Sublayer}(x))$, onde $\text{Sublayer}(x)$ 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 $d_{\text{model}} = 512$.

Decodificador: O decodificador também é composto por uma pilha de $N = 6$ camadas idênticas. Além das duas sub-camadas em cada camada do codificador, o decodificador insere uma terceira sub-camada, que realiza atenção multi-cabeça sobre a saída da pilha do codificador. Semelhante ao codificador, empregamos conexões residuais em torno de cada uma das sub-camadas, seguidas por normalização de camada. Também modificamos a sub-camada de auto-atenção na pilha do decodificador para impedir que as posições atendam a posições subsequentes. Esse mascaramento, combinado com o fato de que os embeddings de saída são deslocados por 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 o mapeamento de uma consulta (query) e um conjunto de pares chave-valor (key-value) para uma saída, onde a consulta, chaves, valores e saída são todos vetores. A saída é calculada como uma soma ponderada dos valores, onde o peso atribuído a cada valor é calculado por uma função de compatibilidade da consulta com a chave correspondente.

[Imagem removida]

Figura 2: (esquerda) Atenção de Produto Escalar em Escala. (direita) Atenção Multi-Cabeça consiste em várias camadas de atenção executando em paralelo.

3.2.1 Atenção de Produto Escalar em Escala

Chamamos nossa atenção particular de "Atenção de Produto Escalar em Escala"(Figura 2). A entrada consiste em consultas e chaves de dimensão d_k, e valores de dimensão d_v. Calculamos os produtos escalares da consulta com todas as chaves, dividimos cada um por $\sqrt{d_k}$ e aplicamos uma função softmax para obter os pesos sobre os valores.

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

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

As duas funções de atenção mais comumente usadas são a atenção aditiva [**?**] e a atenção de produto escalar (multiplicativa). A atenção de produto escalar é idêntica ao nosso algoritmo, exceto pelo fator de escala de $1/\sqrt{d_k}$. A atenção aditiva calcula a função de compatibilidade usando uma rede feed-forward com uma única camada oculta. Embora as duas sejam semelhantes em complexidade teórica, a atenção de produto escalar é muito mais rápida e eficiente em termos de espaço na prática, pois pode ser implementada usando código de multiplicação de matrizes altamente otimizado.

Embora para valores pequenos de d_k os dois mecanismos tenham desempenho semelhante, a atenção aditiva supera a atenção de produto escalar sem escala para valores maiores de d_k [**?**]. Suspeitamos que para valores grandes de d_k, os produtos escalares crescem muito em magnitude, empurrando a função softmax

para regiões onde ela tem gradientes extremamente pequenos[1]. Para neutralizar esse efeito, escalamos os produtos escalares por $1/\sqrt{d_k}$.

3.2.2 Atenção Multi-Cabeça

Em vez de realizar uma única função de atenção com chaves, valores e consultas de dimensão d_{model}, descobrimos que é benéfico projetar linearmente as consultas, chaves e valores h vezes com diferentes projeções lineares aprendidas para as dimensões d_k, d_k e d_v, respectivamente. Em cada uma dessas versões projetadas de consultas, chaves e valores, executamos a função de atenção em paralelo, resultando em valores de saída de dimensão d_v. Estes são concatenados e novamente projetados, resultando nos valores finais, como representado 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.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

$$\text{onde head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

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, empregamos $h = 8$ 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 semelhante ao de uma atenção de cabeça única com dimensionalidade completa.

3.2.3 Aplicações da Atenção em nosso Modelo

O Transformer usa atenção multi-cabeça de três maneiras diferentes:

- Em camadas de "atenção codificador-decodificador", as consultas vêm da camada anterior do decodificador, e as chaves e valores de memória vêm da saída do codificador. Isso permite que cada posição no decodificador atenda a todas as posições na sequência de entrada. Isso imita os mecanismos típicos de atenção codificador-decodificador em modelos sequência-a-sequência como [**?**, **?**, **?**].

- O codificador contém camadas de auto-atenção. Em uma camada de auto-atenção, todas as chaves, valores e consultas vêm do mesmo lugar, neste caso, a saída da camada anterior no codificador. Cada posição no codificador pode atender a todas as posições na camada anterior do codificador.

- Da mesma forma, camadas de auto-atenção no decodificador permitem que cada posição no decodificador atenda a todas as posições no decodificador até e incluindo essa posição. Precisamos impedir o fluxo de informação para a esquerda no decodificador para preservar a propriedade auto-regressiva. Implementamos isso dentro da atenção de produto escalar em escala, 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 Ponto a Ponto

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 separada e identicamente. Isso consiste em duas transformações lineares com uma ativação ReLU entre elas.

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

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

[1]Para ilustrar por que os produtos escalares ficam grandes, suponha que os componentes de q e k sejam variáveis aleatórias independentes com média 0 e variância 1. Então, seu produto escalar, $q \cdot k = \sum_{i=1}^{d_k} q_i k_i$, tem média 0 e variância d_k.

3.4 Embeddings e Softmax

De forma semelhante a 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 usual e a função softmax para converter a saída do decodificador em probabilidades previstas do próximo token. Em nosso modelo, compartilhamos a mesma matriz de pesos entre as duas camadas de embedding e a transformação linear pré-softmax, semelhante a [**?**]. Nas camadas de embedding, multiplicamos esses pesos por $\sqrt{d_{\text{model}}}$.

3.5 Codificação Posicional

Como nosso modelo não contém recorrência nem convolução, para 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 este fim, adicionamos "codificações posicionais"aos embeddings de entrada na base das pilhas do codificador e do decodificador. As codificações posicionais têm a mesma dimensão d_{model} que os embeddings, de modo que os dois podem ser somados. Existem muitas opções de codificações posicionais, aprendidas e fixas [**?**].

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

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

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

onde pos é a posição e i é a dimensão. Ou seja, cada dimensão da codificação posicional corresponde a uma sinusoide. 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, $PE_{\text{pos}+k}$ pode ser representado como uma função linear de PE_{pos}.

Também experimentamos o uso de embeddings posicionais aprendidos [**?**] e descobrimos que as duas versões produziram resultados quase idênticos (ver Tabela 3, linha (E)). Escolhemos a versão sinusoidal porque ela pode permitir que o modelo extrapole para comprimentos de sequência maiores do que os encontrados durante o treinamento.

4 Por que Auto-Atenção?

Nesta seção, comparamos vários aspectos das camadas de auto-atenção com as camadas recorrentes e convolucionais comumente usadas para mapear uma sequência de comprimento variável de representações de símbolos $(x_1, \ldots, x_n)$ para 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 de transdução de sequência típico. Motivando nosso uso da auto-atenção, consideramos três desejos.

Um é a complexidade computacional total por camada. Outro 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 chave em muitas tarefas de transdução de sequência. Um fator chave que afeta a capacidade de aprender tais dependências é o comprimento dos caminhos que os sinais para frente e para trás têm que percorrer na rede. Quanto mais curtos esses caminhos entre qualquer combinação de posições nas sequências de entrada saída, mais fácil é aprender dependências de longo alcance [**?**]. Portanto, também comparamos o comprimento máximo do caminho entre quaisquer duas posições de entrada e saída em redes compostas pelos diferentes tipos de camada.

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 da vizinhança na auto-atenção restrita.

- **Auto-Atenção:** Complexidade por Camada $O(n^2 \cdot d)$, Operações Sequenciais $O(1)$, Comprimento Máximo do Caminho $O(1)$.

- **Recorrente:** Complexidade por Camada $O(n \cdot d^2)$, Operações Sequenciais $O(n)$, Comprimento Máximo do Caminho $O(n)$.
- **Convolucional:** Complexidade por Camada $O(k \cdot n \cdot d^2)$, Operações Sequenciais $O(1)$, Comprimento Máximo do Caminho $O(\log_k(n))$.
- **Auto-Atenção (restrita):** Complexidade por Camada $O(r \cdot n \cdot d)$, Operações Sequenciais $O(1)$, Comprimento Máximo do Caminho $O(n/r)$.

Como observado na Tabela 1, uma camada de auto-atenção 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, as camadas de auto-atenção são mais rápidas que as camadas recorrentes quando o comprimento da sequência n é menor que a dimensionalidade da representação d, o que é mais frequente com representações de sentenças usadas por modelos de estado da arte em tradução automática, como representações word-piece [**?**] e byte-pair [**?**]. Para melhorar o desempenho computacional para tarefas envolvendo sequências muito longas, a auto-atenção 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 $O(n/r)$. Planejamos investigar essa abordagem mais a fundo em trabalhos futuros.

Uma única camada convolucional com largura de kernel $k < n$ 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

Como benefício adicional, a auto-atenção poderia produzir modelos mais interpretáveis. Inspecionamos as distribuições de atenção de nossos modelos e apresentamos e discutimos exemplos no apêndice. Não apenas as cabeças de atenção individuais aprendem claramente a realizar diferentes tarefas, muitas parecem exibir comportamento relacionado à estrutura sintática e semântica das sentenças.

5 Treinamento

Esta seção descreve o regime de treinamento para nossos modelos.

5.1 Dados de Treinamento e Batching

Treinamos no conjunto de dados padrão WMT 2014 Inglês-Alemão, consistindo em cerca de 4.5 milhões de pares de sentenças. As sentenças foram codificadas usando a codificação byte-pair [**?**], que tem um vocabulário fonte-alvo compartilhado de cerca de 37000 tokens. Para Inglês-Francês, usamos o conjunto de dados significativamente maior WMT 2014 Inglês-Francês, consistindo em 36M de sentenças e dividimos os tokens em um vocabulário word-piece de 32000 [**?**]. Os pares de sentenças foram agrupados em lotes (batched) por comprimento aproximado da sequência. Cada lote de treinamento continha um conjunto de pares de sentenças contendo aproximadamente 25000 tokens de origem e 25000 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 passo de treinamento levou cerca de 0.4 segundos. Treinamos os modelos base por um total de 100.000 passos ou 12 horas. Para nossos modelos grandes (descritos na linha inferior da tabela 3), o tempo de passo foi de 1.0 segundo. Os modelos grandes foram treinados por 300.000 passos (3.5 dias).

5.3 Otimizador

Usamos o otimizador Adam

5.4 Regularização

Empregamos três tipos de regularização durante o treinamento:

Residual Dropout Aplicamos dropout

Suavização de Rótulos Durante o treinamento, empregamos suavização de rótulos com valor $\epsilon_{ls} = 0.1$

6 Resultados

6.1 Tradução Automática

Na tarefa de tradução Inglês-para-Alemão do WMT 2014, o modelo transformer grande (Transformer (big) na Tabela 2) supera os melhores modelos relatados anteriormente (incluindo ensembles) em mais de 2.0 BLEU, estabelecendo uma nova pontuação BLEU de estado da arte 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, com uma fração do custo de treinamento de qualquer um dos modelos competitivos.

Na tarefa de tradução Inglês-para-Francês do WMT 2014, nosso modelo grande atinge uma pontuação BLEU de 41.0, superando todos os modelos únicos publicados anteriormente, com menos de 1/4 do custo de treinamento do modelo de estado da arte anterior. O modelo Transformer (big) treinado para Inglês-para-Francês usou uma taxa de dropout $P_{\text{drop}} = 0.1$, em vez de 0.3.

Para os modelos base, usamos um único modelo obtido pela média dos últimos 5 checkpoints, que foram salvos a cada 10 minutos. Para os modelos grandes, fizemos a média dos últimos 20 checkpoints. Usamos busca em feixe (beam search) com um tamanho de feixe de 4 e penalidade de comprimento $\alpha = 0.6$

Tabela 2: O Transformer alcança melhores pontuações BLEU do que modelos de estado da arte anteriores nos testes newstest2014 de Inglês-para-Alemão e Inglês-para-Francês, com uma fração do custo de treinamento.

- **ByteNet [18]:** BLEU EN-DE: 23.75
- **Deep-Att + PosUnk [39]:** BLEU EN-FR: 39.2, Custo de Treinamento (FLOPs) EN-FR: $1.0 \cdot 10^{20}$
- **GNMT + RL [38]:** BLEU EN-DE: 24.6, BLEU EN-FR: 39.92, Custo de Treinamento EN-DE: $2.3 \cdot 10^{19}$, Custo de Treinamento EN-FR: $1.4 \cdot 10^{20}$
- **ConvS2S [9]:** BLEU EN-DE: 25.16, BLEU EN-FR: 40.46, Custo de Treinamento EN-DE: $9.6 \cdot 10^{18}$, Custo de Treinamento EN-FR: $1.5 \cdot 10^{20}$
- **MoE [32]:** BLEU EN-DE: 26.03, BLEU EN-FR: 40.56, Custo de Treinamento EN-DE: $2.0 \cdot 10^{19}$, Custo de Treinamento EN-FR: $1.2 \cdot 10^{20}$
- **Deep-Att + PosUnk Ensemble [39]:** BLEU EN-FR: 40.4, Custo de Treinamento EN-FR: $8.0 \cdot 10^{20}$
- **GNMT + RL Ensemble [38]:** BLEU EN-DE: 26.30, BLEU EN-FR: 41.16, Custo de Treinamento EN-DE: $1.8 \cdot 10^{20}$, Custo de Treinamento EN-FR: $1.1 \cdot 10^{21}$
- **ConvS2S Ensemble [9]:** BLEU EN-DE: 26.36, BLEU EN-FR: 41.29, Custo de Treinamento EN-DE: $7.7 \cdot 10^{19}$, Custo de Treinamento EN-FR: $1.2 \cdot 10^{21}$
- **Transformer (modelo base):** BLEU EN-DE: 27.3, BLEU EN-FR: 38.1, Custo de Treinamento EN-DE: $3.3 \cdot 10^{18}$
- **Transformer (grande):** BLEU EN-DE: 28.4, BLEU EN-FR: 41.8, Custo de Treinamento EN-DE: $2.3 \cdot 10^{19}$

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[2].

[2]Usamos valores de 2.8, 3.7, 6.0 e 9.5 TFLOPS para K80, K40, M40 e P100, respectivamente.

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 Inglês-para-Alemão no conjunto de desenvolvimento.

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

- **base**: $N = 6, d_{\text{model}} = 512, d_{ff} = 2048, h = 8, d_k = 64, d_v = 64, P_{\text{drop}} = 0.1, \epsilon_{ls} = 0.1$. Passos de treino: 100K. PPL(dev): 4.92. BLEU(dev): 25.8. Parâmetros $\times 10^6$: 65.
- **(A) Variação em h (número de cabeças), com d_k, d_v fixos**:
  - $h = 1, d_k = 512, d_v = 512$: PPL(dev): 5.29, BLEU(dev): 24.9.
  - $h = 4, d_k = 128, d_v = 128$: PPL(dev): 5.00, BLEU(dev): 25.5.
  - $h = 16, d_k = 32, d_v = 32$: (continuação da tabela original não fornecida).

(Nota: A Tabela 3 original parece estar incompleta no texto fornecido. A conversão reflete as informações disponíveis.)