[Tradução] Artigo: “Attention Is All You Need”

19 min read · Jun 13, 2024

Murilo Mazzotti Silvestrini Follow

Share

https://arxiv.org/abs/1706.03762

> *O artigo “Attention Is All You Need” revolucionou o campo da inteligência artificial ao introduzir o conceito de Transformers. Esta inovação transformou a maneira como modelos de linguagem natural (LLMs) são desenvolvidos, proporcionando melhorias significativas em termos de eficiência e desempenho.*
>
> *Este post é uma tradução livre do artigo. Gostaria de enfatizar que não sou um tradutor profissional. Compartilho essa tradução para ajudar a disseminar este importante estudo com a comunidade. Sugestões e correções são bem vindas.*

> *Convido todos a conhecerem mais sobre os autores do artigo e a lerem o estudo original. Todas as referências incluídas possuem links (texto sublinhado) para os artigos citados, e recomendo fortemente a leitura desses materiais para um entendimento mais profundo. Também inseri links em alguns termos chaves.*
>
> *As imagens, textos e fórmulas apresentados aqui são todos retirados do artigo original.*
>
> *Boa Leitura!*

Atenção é Tudo o Que Você Precisa

Attention Is All You Need

Autores: Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

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 de melhor desempenho também conectam o codificador e o decodificador por meio de um mecanismo de atenção. Propomos uma nova arquitetura de rede simples, o Transformer, baseada unicamente em mecanismos de atenção, eliminando completamente 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 de treinamento. Nosso modelo alcança 28,4 BLEU na tarefa de tradução de inglês para alemão do WMT 2014, melhorando em mais de 2 BLEU sobre os melhores resultados existentes, incluindo ensembles. Na tarefa de tradução de inglês para francês do WMT 2014, nosso modelo estabelece um novo estado da arte de BLEU para modelo único com uma pontuação de 41,8 após treinar por 3,5 dias em oito GPUs, uma fração dos custos de treinamento dos melhores modelos da literatura. Mostramos que o Transformer se generaliza bem para outras tarefas, aplicando-o com sucesso à análise de constituição do inglês tanto com dados de treinamento grandes quanto limitados.

1. Introdução

Redes neurais recorrentes, como a *Long Short-Term Memory (*LSTM) [13] e a *Gated Recurrent Unit* (GRU) [7], em particular, estabeleceram-se firmemente como abordagens de ponta em modelagem de sequências e problemas de transdução, como modelagem de linguagem e aprendizado de máquina [35, 2, 5]. Numerosos esforços têm continuado a expandir os limites dos modelos de linguagem recorrentes e das arquiteturas codificador-decodificador [38, 24, 15].

Os modelos recorrentes normalmente distribuem 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*. Esta natureza sequencial inerente impede a paralelização dentro dos exemplos de treinamento, o que se torna crítico em comprimentos de sequência mais longos, já que as limitações de memória restringem o agrupamento entre exemplos. Trabalhos recentes conseguiram melhorias significativas na eficiência computacional através de truques de fatoração [21] e computação condicional [32], enquanto também melhoram o desempenho do modelo no caso deste último. No entanto, a restrição fundamental da computação sequencial permanece.

Os mecanismos de atenção tornaram-se uma parte integral de modelos convincentes de modelagem de sequências e transdução em várias tarefas, permitindo a modelagem de dependências sem levar em conta a sua distância nas sequências de entrada ou saída [2,19]. Em quase todos os casos [27], no entanto, tais mecanismos de atenção são usados em conjunto com uma rede recorrente.

2. Antecedentes

O objetivo de reduzir a computação sequencial também forma a base do Extended Neural GPU [16], ByteNet [18] e ConvS2S [9], os quais usam redes neurais convolucionais como bloco de construção básico, computando representações ocultas em paralelo para todas as posições de entrada e saída. Nestes 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 para um número constante de operações, embora com o custo de uma resolução efetiva reduzida devido à média das posições ponderadas pela atenção, um efeito que

contrabalançamos com Atenção Multi-Cabeça (*Multi-Head Attention*), conforme descrito na seção 3.2.

A auto-atenção (*self-attention*), 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, implicação textual e aprendizado de representações de sentenças independentes de tarefas [4, 27, 28, 22].

Redes de memória de ponta a ponta (*end-to-end memory*) são baseadas em um mecanismo de atenção recorrente em vez de recorrência alinhada à sequência e demonstraram um bom desempenho em tarefas de questionamento de linguagem simples e modelagem de linguagem [34].

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 ou convoluções alinhadas à sequência. Nas seções seguintes, descreveremos o Transformer, motivaremos o uso da auto-atenção e discutiremos suas vantagens sobre modelos como [17], [18] e [9].

3. Arquitetura do Modelo

A maioria dos modelos neurais competitivos de transdução de sequência possuem uma estrutura de codificador-decodificador [5, 2, 35]. Aqui, o codificador mapeia uma sequência de representações de símbolos $(x_1, ..., x_n)$ para uma sequência de representações contínuas $z = (z_1, ..., z_n)$. Dada z, o decodificador então gera uma sequência de saída $(y_1, ..., y_m)$ de símbolos, um elemento de cada vez. Em cada passo, o modelo é auto-regressivo [10], consumindo os símbolos gerados anteriormente como entrada adicional ao gerar o próximo.

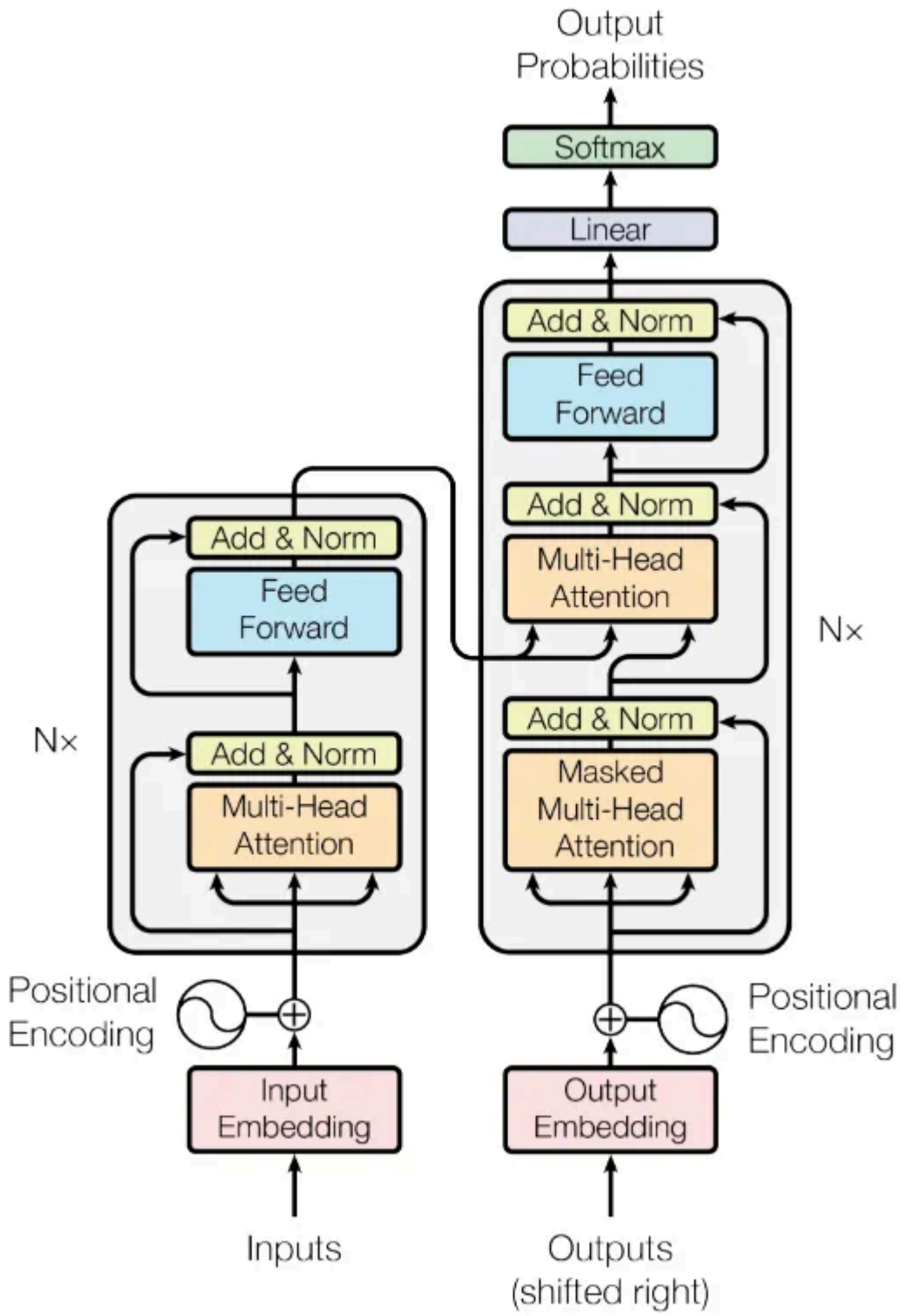

Figura 1. O Transformer — Arquitetura do modelo

O Transformer segue essa arquitetura geral, usando camadas empilhadas de auto-atenção e camadas totalmente conectadas ponto a ponto para ambos o codificador e o decodificador, mostrados nas metades esquerda e direita da Figura 1, respectivamente.

3.1 Pilhas de Codificador e Decodificador

Codificador (***Encoder***): O codificador é composto por uma pilha de $N = 6$ camadas idênticas. Cada camada tem dois sub-níveis. O primeiro é um mecanismo de auto-

atenção multi-cabeça (*multi-head self-attention mechanism*), e o segundo é uma rede feed-forward simples, totalmente conectada e ponto a ponto (*position-wise fully connected feed-forward network*). Utilizamos uma conexão residual [11] ao redor de cada um dos dois sub-níveis, seguida por normalização de camada [1]. Ou seja, a saída de cada sub-nível é $LayerNorm(x + Sublayer(x))$ onde $Sublayer(x)$ é a função implementada pelo próprio sub-nível. Para facilitar essas conexões residuais, todos os sub-níveis do modelo, bem como as camadas de incorporação, produzem saídas de dimensão $dmodel = 512$.

Decodificador (*Decoder*): O decodificador também é composto por uma pilha de $N = 6$ camadas idênticas. Além dos dois sub-níveis em cada camada do codificador, o decodificador insere um terceiro sub-nível, que realiza a atenção multi-cabeça sobre a saída da pilha do codificador. Similar ao codificador, utilizamos conexões residuais ao redor de cada um dos sub-níveis, seguidas por normalização de camada. Também modificamos o sub-nível de auto-atenção na pilha do decodificador para evitar a atenção a posições subsequentes. Esse mascaramento, combinado com o fato de que as incorporações de saída são deslocadas por uma posição, assegura 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 um mapeamento de uma consulta (*query*) e um conjunto de pares chave-valor (*key-value*) para uma saída, onde a consulta, as chaves, os valores e a 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.

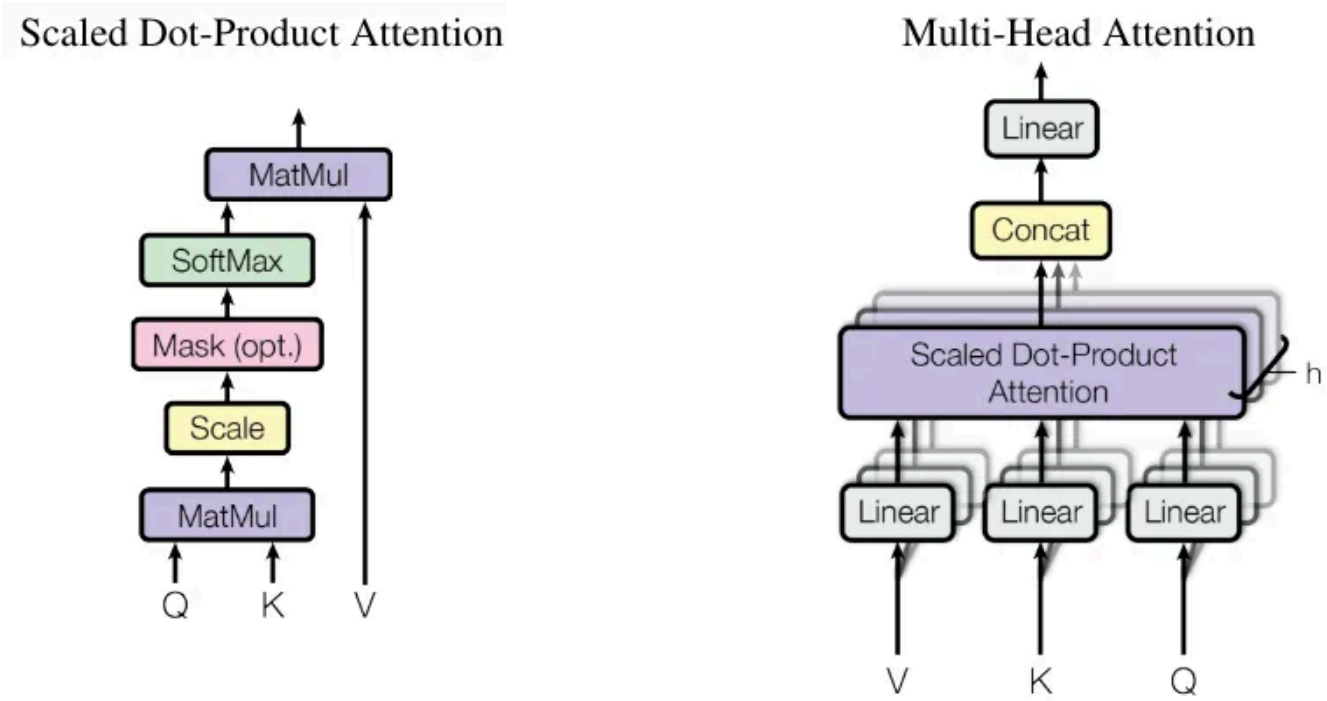

Figura 2: (esquerda) Atenção por Produto Escalar Escalonado. (direita) Atenção Multi-Cabeça consiste em várias camadas de atenção executando em paralelo.

3.2.1 Atenção por Produto Escalar Escalonado

Chamamos nossa atenção particular de “Atenção por Produto Escalar Escalonado” (*Scaled Dot-Product Attention*) (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, agrupadas em uma matriz Q. As chaves e os valores também são agrupados em matrizes K e V. Calculamos a matriz de saída como:

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

Equação 1

As duas funções de atenção mais comumente usadas são atenção aditiva [2] e atenção por produto escalar (multiplicativa). A atenção por produto escalar é idêntica ao nosso algoritmo, exceto pelo fator de escala $1/\sqrt{d_k}$. A atenção aditiva calcula a função de compatibilidade usando uma rede feed-forward com uma única camada oculta. Embora as duas tenham complexidade teórica semelhante, a

atenção por produto escalar é muito mais rápida e eficiente em termos de espaço na prática, pois pode ser implementada usando código altamente otimizado de multiplicação de matrizes.

Para valores pequenos de d_k, os dois mecanismos têm desempenho semelhante, mas a atenção aditiva supera a atenção por produto escalar sem escalonamento para valores maiores de d_k [3]. Suspeitamos que, para grandes valores de d_k, os produtos escalares aumentem em magnitude, empurrando a função softmax para regiões onde ela tem gradientes extremamente pequenos.* Para combater esse efeito, escalonamos os produtos escalares por $1/\sqrt{d_k}$.

**para ilustrar porque 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,*

$$q \times k = \sum_{i=1}^{d_k} q_i k_i$$

, tem média 0 e variância $1/\sqrt{d_k}$.

3.2.2 Atenção Multi-Cabeça

Em vez de realizar uma única função de atenção com chaves, valores e consultas de dimensão *dmodel*, descobrimos que é benéfico projetar linearmente as consultas, chaves e valores *h* vezes com diferentes projeções lineares aprendidas para dimensões d_k, d_k e d_v, respectivamente. Em cada uma dessas versões projetadas de consultas, chaves e valores, realizamos 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 mostrado na Figura 2.

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

$$\begin{aligned}\text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \\ \text{where} \quad \text{head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned}$$

Onde as projeções são matrizes de parâmetros

$$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v} \text{ and } W_i^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$$

Neste trabalho, empregamos $h = 8$ camadas de atenção paralelas, ou cabeças. Para cada uma delas, usamos $d_k = d_v = dmodel / h = 64$. Devido à dimensão reduzida de cada cabeça, o custo computacional total é semelhante ao da atenção de uma única cabeça com dimensionalidade total.

3.2.3 Aplicações da Atenção em nosso Modelo

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

- Nas 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 de sequência-para-sequência [38, 2, 9].

- 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, da saída da camada anterior no codificador. Cada posição no codificador pode atender a todas as posições na camada anterior do codificador.

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

3.3 Redes Feed-Forward Ponto a Ponto

(Position-wise Feed-Forward Networks)

Além dos sub-níveis 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. Isso consiste em duas transformações lineares com ativação ReLU entre elas:

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

Equação 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 *dmodel=512*, e a camada interna tem dimensionalidade *dff = 2048*.

3.4 Embeddings e Softmax

Semelhante a outros modelos de transdução de sequência, usamos embeddings aprendidos para converter os tokens de entrada e de saída em vetores de dimensão *dmodel*. Também usamos a transformação linear usual aprendida e a função softmax para converter a saída do decodificador em probabilidades do próximo token previsto. 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 √*dmodel*.

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)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Tabela 1. Comprimento máximo do caminho, complexidade por camada e número mínimo de operações sequenciais para diferentes tipos de camadas. n é o comprimento da sequência, d é a dimensão de representação, k é o tamanho do kernel das convoluções e r o tamanho da vizinhança em auto-atenção restrita.

3.5 Codificação Posicional

Como nosso modelo não contém recorrência nem convolução, para que o modelo faça uso da ordem da sequência, devemos injetar alguma informação sobre a posição relativa ou absoluta dos tokens na sequência. Para isso, adicionamos “codificações posicionais” aos embeddings de entrada nas bases das pilhas do codificador e do decodificador. As codificações posicionais têm a mesma dimensão *dmodel* que os embeddings, para que possam ser somadas. Há muitas opções de codificações posicionais, aprendidas e fixas [9].

Neste trabalho, usamos 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 da codificação posicional corresponde a uma senoide. Os comprimentos de onda formam uma progressão geométrica de 2π a $10000 \times 2\pi$. Escolhemos essa função porquê hipotetizamos que 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 usar embeddings posicionais aprendidos [9] em vez disso, e descobrimos que as duas versões produziram resultados quase idênticos (ver Tabela 3, linha (E)). Escolhemos a versão senoidal porque 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 representações de símbolos de comprimento variável $(x_1, ..., x_n)$ para outra sequência de comprimento igual $(z_1, ..., z_n)$, com $x_i, z_i \in R^d$, como uma camada oculta em um codificador ou decodificador típico de transdução de sequência. Para motivar o uso da auto-atenção, consideramos três aspectos desejáveis.

Um deles 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 fundamental 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 de avanço e retrocesso têm que percorrer na rede. Quanto mais curtos 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 camada.

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 frequentemente o caso com representações de sentenças usadas por modelos de ponta em traduções automáticas, como as representações *word-piece* [28] e *byte-pair* [31]. Para melhorar o desempenho computacional em tarefas que envolvem sequências muito longas, a auto-atenção poderia ser restringida a considerar apenas uma vizinhança de tamanho *r* na sequência de entrada centrada na 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 $k < n$ 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 caras do que camadas recorrentes, por um fator de *k*. Convoluções separáveis [6], no entanto, diminuem consideravelmente a complexidade, para $O(k \times n \times d + n \times d^2)$. Mesmo com $k = n$, porém, a complexidade de uma convolução separável é igual à combinação de uma camada de auto-atenção e uma camada feed-forward ponto a ponto, a abordagem que adotamos em nosso modelo.

Como benefício adicional, a auto-atenção pode resultar em modelos mais interpretáveis. Inspecionamos as distribuições de atenção de nossos modelos e apresentamos e discutimos exemplos no apêndice. Não apenas cabeças de atenção

individuais claramente aprendem a realizar diferentes tarefas, muitas parecem exibir comportamentos relacionados 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 Agrupamento

Treinamos no conjunto de dados padrão WMT 2014 de inglês-alemão, consistindo em cerca de 4,5 milhões de pares de sentenças. As sentenças foram codificadas usando codificação *byte-pair* [3], que tem um vocabulário de origem-alvo compartilhado de cerca de 37.000 tokens. Para inglês-francês, usamos o conjunto de dados significativamente maior WMT 2014 de inglês-francês, consistindo em 36 milhões de sentenças e dividimos os tokens em um vocabulário de 32.000 palavras [38]. Os pares de sentenças foram agrupados por comprimento aproximado da sequência. Cada lote de treinamento continha um conjunto de pares de sentenças 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 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 última linha 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 com $\beta_1=0.9$, $\beta_2=0.98$ e $\epsilon = 10^{-9}$. Variamos a taxa de aprendizado ao longo do treinamento, de acordo com a fórmula:

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$$

Equação 3

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

5.4 Regularização

Empregamos três tipos de regularização durante o treinamento:

1. Dropout Residual: Aplicamos dropout [33] à saída de cada sub-camada, antes de ser adicionada à entrada da sub-camada e normalizada. Além disso, aplicamos dropout às somas dos embeddings e das codificações posicionais nas pilhas do codificador e do decodificador. Para o modelo base, usamos uma taxa de $P_{drop} = 0,1$.

2. Suavização de Rótulo: Durante o treinamento, empregamos suavização de rótulo de valor $\epsilon_{ls} = 0,1$. Isso prejudica a perplexidade, pois o modelo aprende a ser mais incerto, mas melhora a precisão e a pontuação BLEU.

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

Tabela 2. O Transformer obtém melhores pontuações BLEU do que os modelos de última geração anteriores nos testes newstest2014 de inglês para alemão e de inglês para francês por uma fração do custo de treinamento.

6. Resultados

6.1 Tradução Automática

Na tarefa de tradução de inglês para alemão do WMT 2014, o modelo Transformer grande (Transformer (big) na Tabela 2) supera os melhores modelos anteriormente relatados (incluindo conjuntos de modelos) em mais de 2,0 BLEU, estabelecendo uma nova pontuação BLEU de 28,4. A configuração deste modelo está listada na última linha da Tabela 3. O treinamento levou 3,5 dias em 8 GPUs P100. Mesmo nosso modelo base supera todos os modelos e conjuntos publicados anteriormente, a uma fração do custo de treinamento de qualquer um dos modelos competitivos.

Na tarefa de tradução de inglês para francês do WMT 2014, nosso modelo grande atinge uma pontuação BLEU de 41,0, superando todos os modelos individuais anteriormente publicados, a menos de ¼ do custo de treinamento do modelo de

estado da arte anterior. O modelo Transformer (big) treinado para inglês-francês usou uma taxa de dropout *Pdrop = 0,1*, em vez de 0,3.

Para os modelos base, usamos um único modelo obtido pela média dos últimos 5 checkpoints, que foram gravados em intervalos de 10 minutos. Para os modelos grandes, fizemos a média dos últimos 20 checkpoints. Usamos busca em feixe com um tamanho de feixe de 4 e penalidade de comprimento $\alpha = 0,6$. Esses hiperparâmetros foram escolhidos após experimentação no conjunto de desenvolvimento. Definimos o comprimento máximo de saída durante a inferência como o comprimento de entrada +50, mas terminamos mais cedo quando possível [38].

A Tabela 2 resume nossos resultados e compara nossa qualidade de tradução e custos de treinamento com outras arquiteturas de modelos 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 de ponto flutuante de precisão única sustentada de cada GPU (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 de desempenho na tradução de inglês para alemão no conjunto de desenvolvimento, newstest2013. Usamos busca em feixe como descrito na seção anterior, mas sem a média dos checkpoints. Apresentamos esses resultados na Tabela 3.

	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

Tabela 3. Variações na arquitetura do Transformer. Os valores não listados são idênticos aos do modelo base. Todas as métricas estão no conjunto de desenvolvimento de tradução de inglês para alemão, newstest2013. As perplexidades listadas são por palavra, de acordo com nossa codificação de pares de bytes, e não devem ser comparadas às perplexidades por palavra.

Na Tabela 3, linhas (A), variamos o número de cabeças de atenção e as dimensões das chaves e valores de atenção, mantendo constante a quantidade de computação, conforme descrito na Seção 3.2.2. Embora a atenção de cabeça única seja 0,9 BLEU pior do que a melhor configuração, a qualidade também diminui 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, como esperado, modelos maiores são melhores, e o dropout é muito útil para evitar o overfitting. Na linha (E), substituímos nossa codificação posicional senoidal por embeddings posicionais aprendidos [9] e observamos resultados quase idênticos ao modelo base.

6.3 Análise Constituinte do Inglês

Para avaliar se o Transformer pode se generalizar para outras tarefas, realizamos experimentos na análise constituinte do 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 de sequência-para-sequência baseados em RNNs não conseguiram atingir resultados de estado da arte em regimes de dados pequenos [37].

Treinamos um Transformer de 4 camadas com *dmodel = 1024* na parte do Wall Street Journal (WSJ) do Penn Treebank [25], cerca de 40.000 sentenças de treinamento. Também o treinamos em um cenário semi-supervisionado, usando os corpora de alta confiança e BerkeleyParser, com aproximadamente 17 milhões de sentenças [37]. Usamos um vocabulário de 16.000 tokens para o cenário apenas WSJ e um vocabulário de 32.000 tokens para o cenário semi-supervisionado.

Realizamos apenas um pequeno número de experimentos para selecionar o dropout, tanto atenção quanto residual (Seção 5.4), taxas de aprendizado e tamanho de feixe no conjunto de desenvolvimento da Seção 2.2, todos os outros parâmetros permaneceram inalterados em relação ao modelo base de tradução inglês-alemão. Durante a inferência, aumentamos o comprimento máximo de saída para o comprimento de entrada +300. Usamos um tamanho de feixe de 21 e $\alpha = 0,3$ para ambos os cenários WSJ e semi-supervisionado.

Nossos resultados na Tabela 4 mostram que, apesar da falta de ajuste específico para a tarefa, nosso modelo apresenta desempenho surpreendentemente bom, obtendo melhores resultados do que todos os modelos anteriormente relatados, com exceção da Gramática de Rede Neural Recorrente [8].

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

Tabela 4. O Transformer generaliza bem para a análise do eleitorado inglês (os resultados estão na Seção 2.3 do WSJ)

Em contraste com os modelos de sequência-para-sequência baseados em RNNs [37], o Transformer supera o Berkeley-Parser [29] mesmo quando treinado apenas no conjunto de treinamento WSJ de 40.000 sentenças.

7. Conclusão

Neste trabalho, apresentamos o Transformer, o primeiro modelo de transdução de sequência baseado inteiramente em atenção, substituindo as camadas recorrentes mais comumente usadas em arquiteturas de codificador-decodificador por auto-atenção multi-cabeça.

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 do WMT 2014 de inglês-para-alemão e inglês-para-francês, alcançamos um novo estado da arte. Na primeira tarefa, nosso melhor modelo supera até mesmo todos os conjuntos de modelos anteriormente relatados.

Estamos entusiasmados com o futuro dos modelos baseados em atenção e planejamos aplicá-los a outras tarefas. Pretendemos estender o Transformer para problemas envolvendo modalidades de entrada e saída além de texto e investigar mecanismos de atenção local e restrita para lidar de maneira eficiente com grandes entradas e saídas, como imagens, áudio e vídeo. Tornar a geração menos sequencial é outro objetivo de nossa pesquisa.

O código que usamos 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 frutíferos, correções e inspiração.

Referências

[1] Layer normalization

[2] Neural machine translation by jointly learning to align and translate

[3] Massive exploration of neural machine translation architectures

[4] Long short-term memory-networks for machine reading

[5] Learning phrase representations using rnn encoder-decoder for statistical machine translation

[6] Xception: Deep learning with depthwise separable convolutions

[7] Empirical evaluation of gated recurrent neural networks on sequence modeling

[8] Recurrent neural network grammars

[9] Convolutional sequence to sequence learning

[10] Generating sequences with recurrent neural networks

[11] Deep residual learning for image recognition

[12] Gradient flow in recurrent nets: the difficulty of learning long-term dependencies

[13] Long short-term memory

[14] Self-training PCFG grammars with latent annotations across languages

[15] Exploring the limits of language modeling

[16] Can active memory replace attention?

[17] Neural GPUs learn algorithms

[18] Neural machine translation in linear time

[19] Structured attention networks

[20] Adam: A method for stochastic optimization

[21] Factorization tricks for LSTM networks

[22] A structured self-attentive sentence embedding

[23] Multi-task sequence to sequence learning

[24] Effective approaches to attention-based neural machine translation

[25] Building a large annotated corpus of English: The Penn Treebank

[26] Effective self-training for parsing

[27] A decomposable attention model

[28] A deep reinforced model for abstractive summarization

[29] Learning accurate, compact, and interpretable tree annotation

[30] Using the output embedding to improve language models

[31] Neural machine translation of rare words with subword units

[32] Outrageously large neural networks: The sparsely-gated mixture-of-experts layer

[33] Dropout: a simple way to prevent neural networks from overfitting

[34] End-to-end memory networks

[35] Sequence to sequence learning with neural networks

[36] Rethinking the inception architecture for computer vision

[37] Grammar as a foreign language

[38] Google’s neural machine translation system: Bridging the gap between human and machine translation

[39] Deep recurrent models with fast-forward connections for neural machine translation

[40] Fast and accurate shift-reduce constituent parsing

Attention Is All You Need | Transformers | NLP | AI

Follow

Written by Murilo Mazzotti Silvestrini

51 followers · 96 following

Deep interest in sciences