

FUNDAÇÃO GETULIO VARGAS  
ESCOLA de MATEMÁTICA APLICADA

Leonardo Santana Dantas

# Transformers: Teoria e Viabilização

Rio de Janeiro  
2021

Leonardo Santana Dantas

# Transformers: Teoria e Viabilização

Trabalho de conclusão de curso para obtenção do grau de bacharel apresentado à Escola de Matemática Aplicada

Área de concentração: Aprendizado de Máquina

Orientador: Renato Rocha Souza

Rio de Janeiro  
2021

Dantas, Leonardo S.

Transformers: Teoria e Viabilização

57 páginas

Trabalho de Conclusão de Curso (Bacharelado) - Escola  
de Matemática Aplicada

1. Aprendizado de Máquina
2. Redes Neurais Artificiais
3. Processamento de Linguagem Natural

Leonardo Santana Dantas

# Transformers: Teoria e Viabilização

Trabalho de conclusão de curso para obtenção do grau de bacharel  
apresentado à Escola de Matemática Aplicada

E aprovado em 02/07/2021  
Pela comissão organizadora

---

Prof. Dr. Renato Rocha Souza

---

Prof. Dr. Flávio Codeço Coelho

---

Prof. Dr. Suemi Higuchi

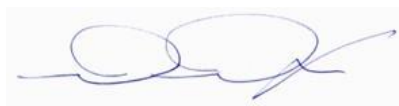
**LEONARDO SANTANA DANTAS**

**“TRANSFORMERS: TEORIA E VIABILIZAÇÃO”**

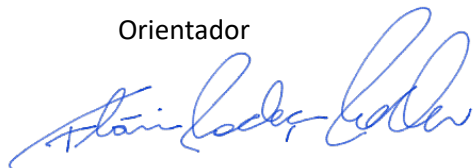
Trabalho de Conclusão de Curso - TCC apresentado ao Curso de Graduação em Matemática Aplicada da Escola de Matemática Aplicada para obtenção do grau de Bacharel (a) em Matemática Aplicada.

Data da Defesa: 02/07/2021

**ASSINATURA DOS MEMBROS DA BANCA EXAMINADORA**



Renato Rocha Souza  
Orientador



Flávio Codeço Coelho  
Membro



Suemi Higuchi  
Membro

Nos termos da Lei nº 13.979 de 06/02/20 - DOU nº 27 de 07/02/20 e Portaria MEC nº 544 de 16/06/20 - DOU nº 114 de 17/06/20 que dispõem sobre a suspensão temporária das atividades acadêmicas presenciais e a utilização de recursos tecnológicos face ao COVID-19, as apresentações dos Trabalhos de Conclusão de Curso, de forma excepcional, serão realizadas de forma remota e síncrona, incluindo-se nessa modalidade membros da banca e discente.

# Agradecimentos

Agradeço a Deus pela oportunidade de completar este trabalho. Agradeço à minha família pelo amplo apoio aos meus estudos, em particular aos meus pais. Agradeço aos meus amigos pelo encorajamento ao longo desse processo. Por fim, agradeço aos professores que contribuíram para minha formação nesta instituição, em especial o Professor Renato Rocha Souza por aceitar ser o meu orientador nesta grande jornada.

# Resumo

Transformers atualmente representam uma das classes mais poderosas de modelos de aprendizado de máquina para processamento de linguagem natural (PLN). Neste trabalho, exploramos os conceitos teóricos por trás dessa arquitetura de redes neurais, os desafios do cenário presente e os aprimoramentos de eficiência propostos na literatura desde a introdução desses modelos.

**Palavras-chave:** aprendizado de máquina, redes neurais artificiais, processamento de linguagem natural

# Abstract

Transformers currently represent one of most powerful classes of machine learning models for natural language processing (NLP). In this work, we explore the theoretical concepts behind this neural network architecture, the challenges in the present scenario, and efficiency improvements proposed in the literature since this introduction of these models.

**Keywords:** machine learning, artificial neural networks, natural language processing



# Lista de Figuras

2.1	Funcionamento de um computador. . . . .	3
3.1	Arquitetura do Transformer conforme apresentada em (VASWANI et al., 2017) .	13
3.2	Exemplificação em (RAFFEL et al., 2020) de entradas do T5. . . . .	16
4.1	Número de parâmetros em modelos de PLN ao longo do tempo. Figura retirada de (SANH et al., 2019). . . . .	21
5.1	Funcionamento de um Transformer-XL com $L = 4$ durante a fase de treino (a) e a fase de avaliação (b), conforme ilustrado em (DAI et al., 2019). . . . .	24
5.2	Visualização em (CHILD et al., 2019) comparando o funcionamento da atenção no Transformer original (a), na abordagem por passo largo (b) e na abordagem por padrão fixo (c). . . . .	26
5.3	Particionamento binário conforme ilustrado em (YE et al., 2019). . . . .	27
5.4	Visualização da construção de $\mathcal{G}$ , conforme apresentada em (YE et al., 2019). Cores diferentes indicam níveis diferentes. Linhas tracejadas são arestas conectando folhas a nós internos. Linhas sólidas são arestas que levam a folhas. . . . .	28
5.5	Critérios de seleção conforme ilustrados em (ZAHEER et al., 2020). É possível identificar os critérios de atenção aleatória (a), janela de contexto (b), atenção global (c) e a combinação das técnicas em (d). . . . .	31
5.6	Modificações do Linformer ao mecanismo de atenção conforme ilustrado em (WANG; LI et al., 2020). . . . .	32
5.7	Funcionamento de atenção LSH conforme ilustrado em (KITAEV; KAISER; LEVSKAYA, 2020) . . . . .	34
5.8	Complexidade computacional após aproximação via núcleos conforme ilustrado em (CHOROMANSKI et al., 2020). . . . .	36
5.9	Relação entre pontuação LRA e velocidade, conforme apresentado em (TAY et al., 2020). . . . .	40

# Lista de Tabelas

3.1	Tarefas de Referência para Modelos de PLN . . . . .	17
5.1	Modelos com Propostas Novas para Atenção . . . . .	38
5.2	Desempenho de modelos em tarefas da LRA, conforme apresentado em (TAY et al., 2020). . . . .	39
5.3	Velocidade e consumo de modelos testados via LRA, conforme apresentado em (TAY et al., 2020). . . . .	40

# Nomenclatura

CNN	Convolutional Neural Network
ELU	Exponential Linear Unit
ETC	Extended Transformer Construction
FAVOR+	Fast Attention via Positive Orthogonal Random features
GELU	Gaussian Error Linear Unit
GPT	Generative Pre-Training
GRU	Gated Recurrent Unit
GSA	Graph Self-Attention
ITC	Internal Transformer Construction
KD	Knowledge Distillation
LED	Longformer-Encoder-Decoder
LM	Language Modeling
LRA	Long Range Arena
LSH	Locality-Sensitive Hashing
LSTM	Long Short-Term Memory
MLM	Masked Language Modeling
MSE	Mean Squared Error
NSP	Next Sentence Prediction
PLN	Processamento de Linguagem Natural
ReLU	Rectified Linear Unit
RNA	Rede Neural Artificial
RNN	Recurrent Neural Network
SVD	Singular Value Decomposition
T5	Text-To-Text-Transfer Transformer

# Sumário

<b>Capa</b>	<b>i</b>
<b>Sumário</b>	<b>xi</b>
<b>1 Introdução</b>	<b>1</b>
<b>2 Fundamentos</b>	<b>2</b>
2.1 Aprendizado de Máquina . . . . .	2
2.2 Redes Neurais Artificiais . . . . .	4
2.2.1 Neurônio Artificial . . . . .	4
2.2.2 Redes Neurais Multicamada . . . . .	5
2.2.3 Funções de Ativação . . . . .	5
2.2.4 Funções Objetivo . . . . .	6
2.2.5 Outras Operações Matemáticas em Redes Neurais . . . . .	7
2.2.6 Arquiteturas e Modelos . . . . .	7
2.3 Processamento de Linguagem Natural . . . . .	7
2.3.1 Modelagem de Linguagem . . . . .	8
2.3.2 Word Embeddings . . . . .	8
2.3.3 Redes Neurais para PLN . . . . .	9
2.3.4 Modelos Sequência-a-Sequência . . . . .	9
<b>3 Transformers</b>	<b>10</b>
3.1 Visão Geral . . . . .	10
3.1.1 Atenção . . . . .	10
3.1.2 Arquitetura . . . . .	12
3.1.3 Regime de Treino . . . . .	14
3.2 Implementações . . . . .	14
3.2.1 AI-Rfou . . . . .	14
3.2.2 Série GPT . . . . .	14
3.2.3 BERT e Derivados . . . . .	15
3.2.4 T5 e mT5 . . . . .	15
3.3 Aspectos Técnicos e Aprimoramentos . . . . .	16
3.3.1 Aprendizado por Transferência . . . . .	16
3.3.2 Regime de Treino . . . . .	16
3.3.3 Tarefas de Referência . . . . .	17
3.4 Situação Atual . . . . .	20
<b>4 Desafios de Transformers</b>	<b>21</b>
4.1 Consumo de Memória . . . . .	21
4.2 Complexidade Quadrática . . . . .	22
4.3 Propostas de Soluções . . . . .	22

<b>5</b>	<b>Alternativas para Atenção</b>	<b>23</b>
5.1	Uso Limitado de Atenção . . . . .	23
5.1.1	Transformer-XL . . . . .	23
5.1.2	Sparse Transformer . . . . .	25
5.1.3	BP-Transformer . . . . .	26
5.1.4	Longformer . . . . .	29
5.1.5	BigBird . . . . .	30
5.2	Alternativas para a Pontuação de Similaridade . . . . .	31
5.2.1	Linformer . . . . .	31
5.2.2	Hashing Sensível a Localidade . . . . .	33
5.2.3	Métodos de Núcleo . . . . .	35
5.3	Comparação . . . . .	38
<b>6</b>	<b>Redução de Modelos</b>	<b>42</b>
6.1	Destilação . . . . .	42
6.1.1	Visão Geral . . . . .	43
6.1.2	Modelo Professor . . . . .	43
6.1.3	Modelo Aluno . . . . .	43
6.1.4	Tarefa e Conjunto de Dados . . . . .	44
6.1.5	Funções de Comportamento . . . . .	44
6.1.6	Resultados e Desempenho . . . . .	47
6.2	Poda . . . . .	47
6.2.1	Procedimento Geral . . . . .	48
6.2.2	Literatura de Poda . . . . .	48
6.3	Quantização . . . . .	49
6.3.1	Quantização Escalar . . . . .	49
6.3.2	Quantização Vetorial . . . . .	49
6.4	Outros Métodos . . . . .	50
6.5	Justificativa e Teoria . . . . .	51
<b>7</b>	<b>Conclusões</b>	<b>52</b>
	<b>Referências</b>	<b>53</b>

# Capítulo 1

## Introdução

O objetivo deste trabalho é introduzir o leitor ao conceito de Transformers, uma classe de modelos matemáticos para processamento de linguagem natural (PLN). Construídos a partir de técnicas de aprendizado de máquina, como redes neurais artificiais, tais modelos vêm desfrutando de grande popularidade nos últimos anos. Hoje, Transformers representam o estado da arte em PLN.

Com o crescimento na popularidade de Transformers, novos desafios surgem. Implementações desses modelos revelam diversas dificuldades e uma vasta literatura começa a surgir sobre como mitigar esses problemas. Sendo assim, este trabalho concentrará em cinco questões chave:

1. O que são Transformers?
2. Quais conceitos estão por trás desses modelos?
3. Como funcionam Transformers?
4. Quais são os desafios de implementação de Transformers?
5. Como mitigamos esses problemas?

Portanto, a contribuição deste trabalho será uma revisão da literatura recente sobre o assunto. Os capítulos estão estruturados da seguinte maneira.

- **Capítulo 2:** Introduz conceitos básicos de aprendizado de máquina (em particular, aprendizado profundo) e processamento de linguagem natural.
- **Capítulo 3:** Visita o artigo seminal de Transformers e os aprimoramentos iniciais à classe de modelos.
- **Capítulo 4:** Considera desafios da teoria e tecnologia apresentadas no Capítulo 3.
- **Capítulos 5 e 6:** Analisam a literatura de soluções aos desafios expostos no Capítulo 4 e o atual estado da arte.
- **Capítulo 7:** Agrega conclusões sobre o estado atual da teoria e tecnologia.

Começaremos o capítulo a seguir com uma motivação para essa discussão e, em seguida, progrediremos para aspectos técnicos.

## Capítulo 2

# Fundamentos

Podem máquinas pensar?

---

Alan Turing

O que é um computador? Em essência, uma máquina de automação do pensamento. Como as máquinas da Revolução Industrial que substituíram o músculo humano em trabalho físico, um computador executa tarefas que ocorrem em nossas mentes – como contas matemáticas.

Todo computador pode ser caracterizado como uma máquina de manipulação de dados. Um dado (que representa uma ideia) é recebido, manipulado segundo um conjunto de operações e retornado ao usuário como resultado. Nesse sentido, existem duas maneiras de automatizar tarefas usando computadores:

1. Instrução Direta (i.e. programação)
2. Aprendizado de Máquina

A primeira será familiar a muitos. Um programador escreve um conjunto de instruções (um algoritmo) sobre como manipular o dado recebido e transformá-lo no dado desejado. Através de linguagens de programação, um programador pode, por exemplo, instruir a máquina sobre como calcular a média das notas de uma turma escolar.

Mas no que consiste o aprendizado de máquina?

### 2.1 Aprendizado de Máquina

Embora determinadas tarefas possam facilmente ser descritas como uma lista de instruções, outras nem tanto. Da mesma forma que é difícil verbalmente instruir uma pessoa sobre como andar de bicicleta, escrever instruções para certas tarefas pode representar um desafio para programadores. Alguns exemplos de tais tarefas são:

- **Visão Computacional:** *Dada uma imagem, como identificar se há um rosto humano nela? Como identificar quantos rostos há numa imagem? Como identificar onde estão os rostos em uma imagem?*
- **Reconhecimento de Voz:** *Microfones captam oscilações na pressão do ar. Como, a partir de medições de pressão, é possível transcrever o que uma pessoa está falando? Como fazer isso levando em consideração a variação dialética de idiomas?*
- **Processamento de Linguagem Natural:** *Dada uma frase em um idioma, como traduzi-la para outro? Dada uma frase, como identificar o sujeito da frase?*

Tal dificuldade em descrever certas tarefas motiva uma nova abordagem para automação. A proposta do aprendizado de máquina é instruir o computador através de exemplos. Formalizando a caracterização previamente apresentada, um computador recebe um dado  $A$ , o manipula segundo uma função  $f$  e retorna um resultado  $B$  (Figura 2.1).

Figura 2.1: Funcionamento de um computador.



Todo aprendizado de máquina busca aproximar a função  $f$  através de exemplos. Quando pares  $(A,B)$  estão previamente disponíveis para nós, chamamos o procedimento de *aprendizado supervisionado*. Quando temos apenas  $A$  e é suficiente para a tarefa que padrões sejam encontrados no dado de entrada, chamamos o procedimento de *aprendizado não-supervisionado*.

Genericamente, o procedimento para aproximar  $f$  em aprendizado supervisionado pode ser descrito como:

1. Criar uma função genérica  $g$ . A função  $g$  deve ser flexível e modificável, mas não precisa ser inicialmente próxima de  $f$ .
2. Alimentar um dado  $A$  à função  $g$  para receber um dado  $\tilde{B}$ .
3. Comparar o dado  $\tilde{B}$  obtido ao dado  $B$  esperado e quantificar o erro.
4. Modificar  $g$  para minimizar o erro cometido.
5. Repetir o procedimento a partir do Passo 2.

Dado um número suficientemente alto de iterações, é esperado que  $g$  se aproxime de  $f$ . Observe que, no fundo,  $g$  está sendo usada para modelar um fenômeno fundamental a partir de exemplos. Portanto, usaremos frequentemente o termo *modelo matemático*, ou simplesmente *modelo*, para nos referir à função  $g$  usada em uma determinada técnica de aprendizado de máquina. Por fim, o procedimento descrito acima é conhecido como o *treino* do modelo.

Tarefas de aprendizado de máquina são geralmente divididas em duas categorias de acordo com seu objetivo:

- **Regressão**
- **Classificação**

A nomenclatura reflete o tipo de dado de saída do modelo: quantitativo ou qualitativo (também chamado de categórico). Chamamos de *regressão* as tarefas quantitativas, cujas saídas representam um valor numérico (contínuo). Já *classificação* se refere às tarefas qualitativas, cujo dado de saída indica a qual de  $K$  categorias o dado de entrada pertence.

Como sistemas de aprendizado de máquina são construídos a partir de dados, a terminologia para descrevê-los também será relevante. Os conjuntos de dados são divididos nas seguintes categorias.

- **Treino:** Dados usados para construir o modelo.
- **Validação:** Dados usados para avaliar o desempenho do modelo durante o processo de treino. Devem ser distintos dos dados de treino.



- **Teste:** Dados usados para avaliar o desempenho do modelo após o processo de treino. Devem ser distintos dos dados de treino.
- **Transferência:** Dados usados para adaptar um modelo previamente treinado para uma nova tarefa.

Embora a distinção entre dados de validação e de teste seja sutil e muitos praticantes englobem as duas categorias, essa diferenciação é importante. Dados de validação servem para orientar o processo de treino, permitindo que o desempenho em dados nunca antes vistos guie ajustes e modificações tanto no regime de treino como no próprio modelo. Tipicamente, uma pequena parcela do conjunto original de treino é removida para servir como conjunto de validação. Nesse sentido, dados de validação podem vir da mesma distribuição do que os de treino. Essa prática não é recomendada para dados de teste, que servem para determinar o desempenho que o modelo terá quando se defrontar com dados completamente novos em sua aplicação final.

Em suma, a abordagem do aprendizado de máquina oferece uma nova maneira de automatizar tarefas. Conforme consideramos, essa maneira é aproximada e construída em cima de grandes volumes de dados. Contudo, o aprendizado de máquina continua a ser responsável por avanços significantes na automação de tarefas de difícil descrição algorítmica, particularmente ao longo da última década. Um ramo do aprendizado de máquina, em particular, se destaca: as redes neurais artificiais.

## 2.2 Redes Neurais Artificiais

Ao longo da última década, *redes neurais artificiais* têm contribuído de maneira significativa ao desenvolvimento do aprendizado de máquina. Redes neurais artificiais (RNAs) são modelos matemáticos usados para aproximar funções. Tais modelos são modulares e flexíveis. Além disso, podem ser usados tanto para tarefas de regressão como de classificação. O estudo das redes neurais artificiais é conhecido como *aprendizado profundo* (em inglês, *deep learning*).

### 2.2.1 Neurônio Artificial

O *neurônio artificial* constitui a unidade básica das redes neurais. Trata-se de uma função matemática simples com parâmetros que podem ser modificados.

Dada uma entrada  $x \in \mathbb{R}^n$ , um vetor de pesos  $w \in \mathbb{R}^n$  e um termo aditivo  $b \in \mathbb{R}$ , chamamos de neurônio artificial a função

$$y(x) = \phi \left( b + \sum_{i=1}^n w_i x_i \right)$$

onde  $y(x)$  representa a saída do modelo e  $\phi$  é conhecida como *função de ativação*. As opções para função de ativação serão consideradas em detalhe posteriormente. Por ora, basta saber que  $\phi$  é sempre uma função não-linear diferenciável (ou aproximadamente diferenciável).

Através de uma função objetivo, é possível comparar a saída do neurônio com a saída esperada. Tornamos, assim, a busca por parâmetros (pesos e termo aditivo) ideais em um problema de otimização matemática. Examinaremos as possíveis escolhas para função objetivo numa subseção futura. Por enquanto, considere uma função objetivo diferenciável.

Como o neurônio é inteiramente diferenciável (ou aproximadamente diferenciável) e a função objetivo é diferenciável, é possível calcular derivadas parciais relativas aos parâmetros para determinar a contribuição de cada parâmetro ao erro. Deste ponto em diante, é possível otimizar o modelo através do método do gradiente ou versões modificadas dele.

Há uma ressalva importante. Como estamos ajustando o modelo a cada exemplo, na realidade estamos executando o método do gradiente *estocástico*. Contudo, explorando o paralelismo em

hardware contemporâneo, é possível ajustar os parâmetros para todos os exemplos de uma vez e executar o método do gradiente tradicional. Na prática contemporânea, é comum ajustar o modelo em lotes (em inglês, *batches*) de exemplos.

### 2.2.2 Redes Neurais Multicamada

Um neurônio isolado constitui o caso mais simples de uma rede neural. No entanto, o verdadeiro poder das redes neurais surge nos grandes agrupamentos de neurônios.

Observe que a saída de um neurônio pode ser passada a outro. Dessa maneira, neurônios podem ser encadeados. Otimização de tais arranjos via métodos de gradiente continua sendo possível por conta da regra da cadeia, num algoritmo conhecido como *retropropagação* (em inglês, *backpropagation* ou simplesmente *backprop*) (RUMELHART; HINTON; WILLIAMS, 1988; RUMELHART; HINTON; WILLIAMS, 1986).

Também é possível executar neurônios em paralelo. Dados  $m$  neurônios, definimos uma matriz  $W \in \mathbb{R}^{n \times m}$  representando os pesos dos neurônios e  $B \in \mathbb{R}^m$  representando os seus termos aditivos. Com isso, obtemos a generalização

$$\text{FFN}(x) = \phi(W^\top x + B)$$

Tal arranjo é chamado *camada*. Como no caso de neurônios isolados, é possível encadear camadas e otimizar através do algoritmo de backpropagation. Uma sequência de camadas encadeadas é conhecida como uma *rede neural multicamada*. Dizemos que a primeira camada é a *camada de entrada* e a última camada é a *camada de saída*. As camadas intermediárias são conhecidas como *camadas ocultas* (em inglês, *hidden layers*).

Note que o número de camadas e o número de neurônios em cada uma delas são flexíveis. É possível construir redes neurais que começam com camadas largas (contendo muitos neurônios) e aos poucos progridem para camadas com menos neurônios. A versatilidade na construção das redes neurais torna essa classe de modelos particularmente flexível.

Observe também que na construção acima a saída de cada camada é passada para cada neurônio da camada seguinte. Sempre que a rede operar desse modo, dizemos que a rede é *densa*.

Por fim, quando dados são encaminhados de uma camada a outra de maneira unidirecional (i.e. a saída de uma camada nunca retorna a uma camada anterior), a rede é chamada de uma *rede de propagação* (em inglês, *feedforward network*). Redes multicamada densas de propagação constituem um dos moldes mais tradicionais em aprendizado profundo e são frequentemente chamadas simplesmente de *redes neurais*, ficando subentendido a qual tipo se referem.

### 2.2.3 Funções de Ativação

Funções de ativação introduzem não-linearidade ao modelo. Sem elas, redes neurais tratariam apenas dados linearmente separáveis. Em uma rede neural, a escolha de função de ativação geralmente depende de dois fatores:

- Desempenho
- Propósito

Tradicionalmente, redes neurais para classificação binária empregam uma função logística padrão (denotada por  $\sigma$ ) ou tangente hiperbólica como função de ativação para a sua camada final.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Observe que tais redes fornecem uma saída escalar. Para classificação em  $K$  classes, a escolha natural é a função *softmax* na camada final. Para  $z \in \mathbb{R}^K$

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

Nesse caso, a saída da rede neural é um vetor de tamanho  $K$  referente às probabilidades de cada classe. No caso de tarefas de regressão, uma escolha popular para a ativação da camada final é a *unidade linear retificada* (em inglês, *rectified linear unit*), melhor conhecida como ReLU.

$$\text{ReLU}(x) = \max(0, x)$$

Historicamente, neurônios em camadas ocultas ou de entrada usam funções como  $\sigma$  ou ReLU. Dessas duas, a segunda é mais recente e mais popular na literatura graças a certos ganhos de desempenho. Outras funções de ativação (incluindo diversas variantes da função ReLU) existem, mas fogem do escopo deste trabalho. Ocasionalmente, a última camada simplesmente não recebe uma função de ativação. Quando uma camada não recebe uma função de ativação dizemos que ela é uma *camada linear*.

## 2.2.4 Funções Objetivo

Como no caso de funções de ativação, funções objetivo são escolhidas de acordo com a tarefa.

Em tarefas de regressão, é comum usar o erro quadrático médio (em inglês, *mean squared error* ou *MSE*). Considerando um dado esperado  $y \in \mathbb{R}^n$  e um dado predito  $\hat{y} \in \mathbb{R}^n$ , o MSE é dado por

$$\mathcal{L}_{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Note que a função acima é diferenciável.

Com relação a tarefas de classificação, é necessário primeiro introduzir o conceito de codificação *one-hot*. Considere  $K$  categorias. A *codificação one-hot* da categoria  $i$  é o vetor de tamanho  $K$  cujo único elemento não-nulo é o da  $i$ -ésima posição, que é igual a 1. Por exemplo, se  $K = 5$ , a representação *one-hot* da categoria 2 é

$$y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Em tarefas de classificação, a escolha mais popular de função objetivo é a entropia cruzada (em inglês, *cross-entropy*), oriunda da teoria da informação. Seja  $y \in \mathbb{R}^K$  a codificação *one-hot* da categoria correta e  $\hat{y} \in \mathbb{R}^K$  o vetor de probabilidades preditas para cada categoria. Então, a função de entropia cruzada é dada por

$$\mathcal{L}_{CE} = - \sum_{i=1}^K y_i \log(\hat{y}_i)$$

Observe que essa função objetivo é também diferenciável. Casos específicos dessa função recebem uma terminologia diferenciada. Quando  $K = 2$ , chamamos a função de *entropia cruzada binária*. Já para  $K > 2$ , nos referimos à função como *entropia cruzada categórica*.

### 2.2.5 Outras Operações Matemáticas em Redes Neurais

Consideramos acima a apresentação tradicional de redes neurais. No entanto, redes neurais podem ser modificadas para conter outras operações matemáticas. Tal prática é prevalente na literatura. Além de camadas convencionais, redes neurais podem incluir camadas de convolução, *pooling*, normalização de lote (em inglês, *batch normalization*) e assim por diante. Em geral, tais operações matemáticas adicionais não serão relevantes para este trabalho. No entanto, duas são de atenção.

#### Normalização de Lote

Introduzidas em (IOFFE; SZEGEDY, 2015), camadas de normalização de lote visam otimizar o processo de treino de uma rede neural. Matematicamente, essa camada normaliza a sua entrada de acordo com a média e o desvio padrão do lote de dados atualmente sendo processado. Na implementação canônica, parâmetros treináveis  $\gamma$  e  $\beta$  são também introduzidos para reter o poder de expressividade da rede. Não entraremos em maiores detalhes sobre esse aspecto, mas formalmente temos a seguinte fórmula para a operação.

$$BN(x) = \frac{x - \mathbb{E}[x]}{\text{Var}(x) + \epsilon} * \gamma + \beta$$

Em certas bibliotecas de aprendizado profundo, os parâmetros treináveis são fixados por padrão, com  $\gamma$  assumindo um vetor de 1s e  $\beta$  assumindo um vetor nulo.

#### Conexões Residuais

Apresentadas em (HE et al., 2016), as *conexões residuais* são frequentemente usadas em redes neurais contemporâneas. Em essência, uma conexão residual consiste em adicionar a saída de uma determinada camada à saída de alguma camada futura. Formalmente, seja  $\mathcal{F}(x)$  uma sequência de camadas ou operações matemáticas em uma rede neural, com  $x \in \mathbb{R}^n$ . Adicionamos uma conexão residual após  $\mathcal{F}$  alimentando o seguinte valor à camada após  $\mathcal{F}$ .

$$\mathcal{H}(x) = \mathcal{F}(x) + x$$

Na prática, conexões residuais propagam valores previamente encontrados a seções futuras da rede. Tal técnica é utilizada para aprimorar o desempenho da rede, mas os detalhes técnicos fogem o escopo desse trabalho.

### 2.2.6 Arquiteturas e Modelos

Dentro do contexto de redes neurais, determinadas nomenclaturas serão úteis. As definições propostas em (BLALOCK et al., 2020) servem como uma boa referência. Uma *arquitetura* consiste na configuração de parâmetros de uma rede neural e o conjunto de operações usadas por ela para produzir saídas a partir de entradas, que inclui: o arranjo dos parâmetros em convoluções, funções de ativação, pooling e normalização de lote. Quando aplicado a redes neurais, o termo *modelo* significa uma parametrização particular de uma arquitetura.

## 2.3 Processamento de Linguagem Natural

Processamento de linguagem natural (PLN) se refere ao estudo da automação de tarefas envolvendo linguagem humana. Tal ramo busca construir máquinas capazes de ler, compreender e extrair significado de texto escrito em linguagem humana. Alguns exemplos de tarefas de PLN são:

- **Tradução Automática:** Converter texto de um idioma para outro.

- **Predição de Texto:** Antecipar as próximas palavras ou expressões de um texto incompleto.
- **Assistentes Virtuais:** Interagir com usuários através de linguagem humana.
- **Análise de Sentimento:** Avaliar se um comentário demonstra uma atitude positiva ou negativa.

A temática central deste trabalho será a análise de novas técnicas e abordagens para realizar tarefas como as acima.

Consideraremos a seguir uma breve história da aplicação de redes neurais artificiais a tarefas de PLN. Interseções entre PLN e essa técnica notável de aproximação de funções são abundantes e de longa data. Portanto, esta seção se limitará a destacar alguns dos principais aspectos dessa trajetória. Grande parte da análise a seguir será baseada nas observações de (RUDER, 2018) em *A Review of the Neural History of Natural Language Processing*. Por fim, optaremos por examinar esses aspectos por temática ao invés de uma estrita ordem cronológica.

### 2.3.1 Modelagem de Linguagem

Há mais de 15 anos, modelos neurais são usados em PLN. Como um dos importantes desenvolvimentos iniciais, Ruder destaca o uso de redes neurais para *modelagem de linguagem*.

Modelagem de linguagem (em inglês, *language modeling* ou *LM*) é a expressão usada na literatura para descrever tarefas de predição de texto. Por exemplo, dado uma frase incompleta, o sistema deve prever a mais provável próxima palavra. Formalmente, a tarefa é frequentemente caracterizada da seguinte maneira. Dado uma sequência  $x$  composta de termos  $(s_1, s_2, \dots, s_n)$ , temos que

$$p(x) = \prod_{i=1}^n p(s_i | s_1, \dots, s_{i-1})$$

O cerne da modelagem de linguagem consiste em modelar as probabilidades condicionais acima. Dentro deste contexto, note que “sequência” se refere a algum trecho de texto: uma frase, um parágrafo ou até um livro completo. Da mesma forma, “termo” se refere a alguma parte constituinte da sequência maior: uma letra, uma sílaba, ou uma palavra, por exemplo.

Em (BENGIO; DUCHARME; VINCENT, 2001) e posteriormente (BENGIO; DUCHARME; VINCENT; JANVIN, 2003), os autores propõem o uso de uma rede neural para realizar a tarefa acima. Conforme colocado por Ruder, a ideia de modelagem de linguagem se mostrou central para diversos desenvolvimentos posteriores em PLN neural.

### 2.3.2 Word Embeddings

Word embeddings<sup>1</sup> (em livre tradução, “imersões de palavras”) são representações de palavras como vetores em um espaço vetorial. Por trás dessa abordagem, há uma expectativa que é possível obter um espaço que não só comporta um certo vocabulário, mas cujas características, como ângulo e distância, capturam relações semânticas.

Particularmente emblemático desse esforço é o trabalho de (MIKOLOV; CHEN et al., 2013; MIKOLOV; SUTSKEVER et al., 2013) com o sistema *word2vec*. Nesses artigos, autores constroem uma word embedding usando redes neurais e demonstram diversas propriedades úteis do espaço resultante. Por exemplo, o vetor que interliga os pontos referentes às palavras “Homem” e “Rei”, respectivamente, é aproximadamente o mesmo que interliga os pontos das palavras “Mulher” e “Rainha”. Tais resultados refletem a captura de uma informação semântica na própria estrutura do espaço.

<sup>1</sup>Pela ausência de termos que capturem a plenitude da ideia, persistiremos no uso da expressão inglesa.

### 2.3.3 Redes Neurais para PLN

Outro destaque da história do PLN neural são as arquiteturas específicas. As Redes Neurais Recorrentes (em inglês, *Recurrent Neural Networks* ou *RNNs*) (MIKOLOV; KARAFIÁT et al., 2010) representam uma arquitetura notadamente eficaz em processamento de texto e razoavelmente popular no ramo. Com a introdução de mecanismos como a *GRU* (*Gated Recurrent Unit*, em inglês) (CHO et al., 2014) e *LSTM* (*Long Short-Term Memory*, em inglês) (HOCHREITER; SCHMIDHUBER, 1997; GRAVES, 2013), as RNNs foram aprimoradas e desenvolveram novas subarquiteturas ainda mais eficientes. Embora os detalhes de funcionamento dessa arquitetura fujam do escopo deste trabalho, as RNNs desempenharam um papel predominante na última década.

### 2.3.4 Modelos Sequência-a-Sequência

*Modelos sequência-a-sequência* (em inglês, *sequence-to-sequence models*) representam uma união das ideias de RNNs e word embeddings. Como o nome sugere, essa abordagem é direcionada a tarefas que envolvem a transformação de uma sequência em outra. Aplicações características incluem tradução automática e resumo de texto.

Apresentada inicialmente em (SUTSKEVER; VINYALS; LE, 2014), a abordagem consiste em criar um modelo *codificador* (em inglês, *encoder*) que consome a sequência de entrada e produz uma representação vetorial da sequência. Em seguida, outro modelo, chamado de *decodificador* (em inglês, *decoder*), recebe a representação vetorial da entrada e produz uma sequência de saída. Como escolha de modelo, os autores do artigo original optaram por uma RNN baseada em LSTM. Contudo, outras escolhas são possíveis.

Notavelmente, os modelos sequência-a-sequência são exemplo de uma *arquitetura codificador-decodificador*, presente em outras áreas do aprendizado profundo.

## Capítulo 3

# Transformers

Em 2017, um artigo intitulado *Attention Is All You Need* ([VASWANI et al., 2017](#)) introduziu uma nova arquitetura neural para PLN. Conhecida como o Transformer, essa abordagem reuniu diversas ideias presentes nos ramos de pesquisa de RNNs, modelos sequência-a-sequência e modelagem de linguagem. Hoje, Transformers constituem umas das abordagens mais populares e poderosas em PLN. Desde a publicação do artigo original, aprimoramentos à arquitetura continuam a ser propostos, embora a essência desses sistemas permaneça a mesma. Neste capítulo, consideraremos o que são Transformers e quais são as grandes ideias por trás deles.

### 3.1 Visão Geral

Transformers surgem dentro de um contexto de modelos neurais sequência-a-sequência. No momento de publicação do artigo original, o cenário de PLN era dominado por modelos baseados em RNNs (particularmente com LSTM) ou redes neurais convolucionais (em inglês, *convolutional neural networks* ou *CNNs*). Uma das grandes ideias desse período é o conceito de *atenção*. Conforme veremos, essa ideia será central para os Transformers. Tão central que os autores de Vaswani et al. demonstrarão que a recorrência e as camadas de convolução que definem as RNNs e CNNs, respectivamente, não são necessárias para um modelo neural de PLN. Realmente, *atenção* é tudo que você precisa. Nesta seção, consideraremos:

1. O que é atenção?
2. Qual é a arquitetura geral do Transformer?

Ambas perguntas serão respondidas pela perspectiva do artigo seminal. Em seções seguintes, discutiremos implementações e aprimoramentos ao Transformer original.

#### 3.1.1 Atenção

##### Motivação

Métodos iterativos são prevalentes em modelos sequência-a-sequência. A abordagem iterativa pode ser exemplificada com uma tarefa de tradução. Uma frase é traduzida do idioma de origem para o idioma destino palavra-a-palavra. Isto é, a cada passo, o modelo é fornecido com a frase original inteira para produzir a próxima palavra no idioma destino. Intuitivamente, nem toda palavra na frase original é igualmente relevante para a tradução da próxima palavra, mas um modelo de implementação ingênua as trata igualmente. O conceito de atenção surge da necessidade de discriminar trechos diferentes do texto de entrada. Desejamos modelar as dependências do segmento sendo gerado naquela iteração. Diversas propostas para alcançar esse objetivo existem e são conhecidas como *mecanismos de atenção*. Em particular, os autores de Vaswani et al. apresentam a seguinte abordagem.

## Queries, Keys e Values

O mecanismo de atenção de Vaswani et al. trabalha na base de três entradas. A escolha dessas entradas ficará clara quando considerarmos a arquitetura do Transformer. Por ora, considere  $x_q, x_k \in \mathbb{R}^{d_k}$  e  $x_v \in \mathbb{R}^{d_v}$ . Antes da aplicação da operação de atenção, cada uma dessas entradas é passada por uma camada linear distinta.

$$\begin{aligned} q &= W_Q^\top x_q + b_Q \\ k &= W_K^\top x_k + b_K \\ v &= W_V^\top x_v + b_V \end{aligned}$$

Os vetores  $q$ ,  $k$  e  $v$  resultantes são conhecidos como *consulta*, *chave* e *valor* (do inglês, *query*, *key* e *value*), respectivamente. Tais expressões surgem da terminologia de pesquisa em banco de dados, que inspira o trabalho.

Quando as entradas são fornecidas em lotes, como é típico, a operação acima pode ser expressa em forma matricial. Dados  $X_Q, X_K \in \mathbb{R}^{d_k \times m}$  e  $X_V \in \mathbb{R}^{d_v \times m}$ , temos

$$\begin{aligned} Q &= W_Q^\top X_Q + B_Q \\ K &= W_K^\top X_K + B_K \\ V &= W_V^\top X_V + B_V \end{aligned}$$

A caracterização em matrizes é a mais típica para explicar a operação de atenção que segue.

## Operação Atenção

Propriamente dita, a operação de atenção consiste em

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

Consideraremos a interpretação da expressão acima. Observe que a função softmax à esquerda determina a similaridade entre as *consultas* e as *chaves* fornecidas. Quando computamos  $QK^\top$ , estamos calculando o produto interno de pares de *consultas* e *chaves* dos quais essas matrizes são compostas. Tal operação nos informa o quão alinhados esses vetores estão. Em seguida, dividimos o resultado por um termo de normalização  $\sqrt{d_k}$  (para estabilidade numérica) e depois passamos esse novo resultado por uma função softmax, que coloca todos elementos em uma mesma escala. Nesse sentido, a função softmax representa, a grosso modo, uma pontuação de similaridade. Por fim, essa pontuação de similaridade é multiplicada com os *valores*, produzindo uma certa ponderação deles. A interpretação geral do mecanismo de atenção é que estamos comparando duas das entradas e propagando os valores da terceira de acordo.

A operação de atenção apresentada acima é conhecida como *atenção de produto escalar escalonado* (em inglês, *scaled dot-product attention*). Com isso, concluímos a apresentação do mecanismo de atenção.

## Atenção Multicabeça

Os autores de Vaswani et al. observam que há um ganho em usar mais de um mecanismo de atenção para o mesmo conjunto de entradas. Cada um desses mecanismos fica conhecido como uma *cabeça* (em inglês, *head*). Sob este esquema, as saídas das cabeças são concatenadas e passadas por uma camada linear. Esta abordagem é conhecida como *atenção multicabeça* (em inglês, *multi-head attention*) e pode ser expressa da seguinte forma.

$$\begin{aligned} \text{MultiCabeça}(Q, K, V) &= \text{Concatenação}(\text{cabeça}_1, \dots, \text{cabeça}_h) W^O \\ \text{onde } \text{cabeça}_i &= \text{Atenção}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned}$$



A atenção multicabeça representa a abordagem padrão dos Transformers. Nesse sentido, a escolha do número de cabeças constitui um dos hiperparâmetros de modelos desse tipo.

### Autoatenção

Há um caso particular de atenção quando  $d_k = d_v$ . Sob estas circunstâncias, o mesmo vetor pode ser fornecido como as três entradas do mecanismo de atenção. Quando a consulta, chave e valor são calculados a partir de um mesmo vetor, chamamos o mecanismo de um mecanismo de *autoatenção* (em inglês, *self-attention*). Tal terminologia é motivada pelo fato do mecanismo estar comparando um vetor de entrada consigo mesmo e determinando quais valores dele devem ser passados adiante.

### Aplicações de Atenção

Tradicionalmente, atenção é aplicada à entrada geral do modelo. Consideramos um exemplo nesses conformes na motivação desta subseção. No entanto, note que o mecanismo de atenção é uma operação matemática que pode ser aplicada a qualquer dado que admita uma partição em *consulta*, *chave* e *valor*. Conforme veremos, Transformers são icônicos pela aplicação de atenção a dados distintos da entrada geral.

#### 3.1.2 Arquitetura

Transformers geram uma sequência a partir de outra de forma iterativa. Como entrada, o modelo recebe a sequência de origem e a sequência de saída produzida até a iteração atual (inicialmente vazia). Com esses dados, o modelo produz uma distribuição de probabilidades para o próximo elemento na sequência de saída. O elemento com a maior probabilidade é então adicionado à sequência de saída e o processo é repetido. Um diagrama completo da arquitetura está representado na Figura 3.1.

Com esse funcionamento geral em mente, consideraremos em detalhes a construção e operação dos componentes do Transformer.

### Entradas e Codificação

Antes de serem alimentadas ao modelo, as entradas do Transformer são primeiro convertidas em *embeddings*. Nesse procedimento, o conteúdo das sequências é codificado em forma vetorial. Contudo, as ordens dos elementos nessas sequências são perdidas. A fim de sanar esse problema, os autores de Vaswani et al. adicionam uma codificação posicional que fornece informações sobre a posição de cada elemento da sequência.

### Codificador e Decodificador

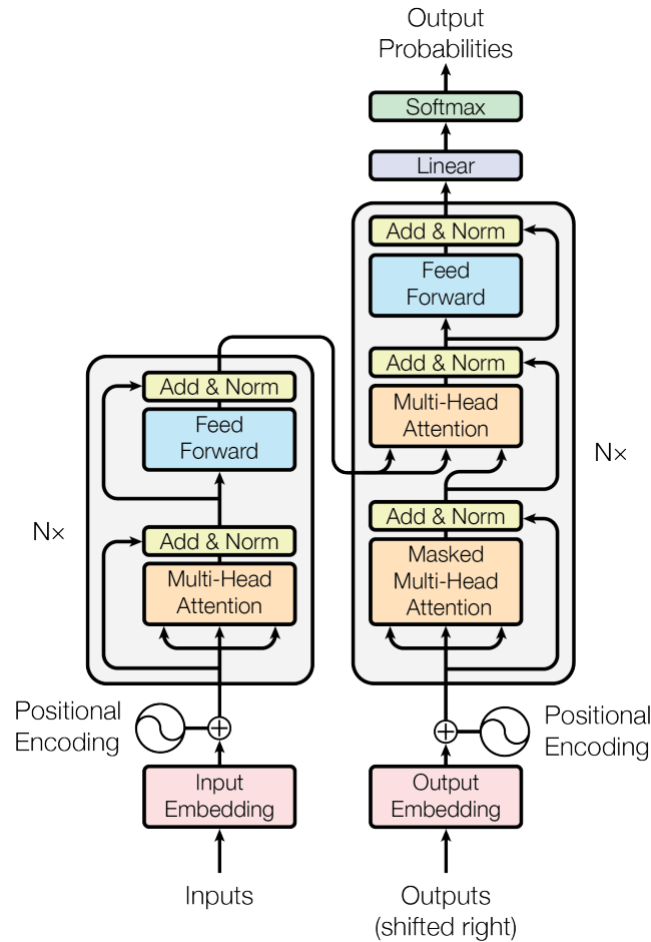
Inspirado por outros modelos sequência-a-sequência neurais, o Transformer é composto por um codificador e um decodificador.

Como entrada, o codificador recebe a embedding da sequência de origem somada à sua codificação posicional. Essa entrada é então propagada por uma sucessão de  $N$  blocos de codificador. Cada um desses blocos de codificador é composto por dois segmentos:

1. **Segmento de Autoatenção:** Composto por um mecanismo de autoatenção multicabeça.
2. **Segmento de Propagação:** Composto por uma pequena rede neural. No artigo original, os autores usam uma rede neural de duas camadas com uma ativação ReLU na primeira camada e nenhuma ativação na segunda. Dado uma entrada  $x \in \mathbb{R}^n$ , essa implementação do segundo segmento pode ser expressa como

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

Figura 3.1: Arquitetura do Transformer conforme apresentada em (VASWANI et al., 2017)



Ao final de cada segmento, os autores empregam uma conexão residual e uma normalização de camada.

O decodificador opera de maneira semelhante ao codificador, porém distinta. Como entrada, o decodificador recebe a embedding da sequência de saída produzida até a iteração atual somada à sua codificação posicional. De forma análoga ao codificador, o decodificador é composto por  $N$  blocos. Contudo, os seus blocos são formados por três segmentos, estruturados de maneira diferente do codificador. Esses segmentos são:

1. **Segmento de Autoatenção:** Composto por um mecanismo de autoatenção multicabeça. Especificamente no decodificador, esse mecanismo é adaptado para que uma máscara possa ser aplicada sobre a entrada. A adaptação permite que, ao longo do processo de treino, sequências destino sejam alimentadas *parcialmente* ao decodificador.
2. **Segmento de Atenção:** Composto por um mecanismo de atenção multicabeça cujas chaves e valores são calculados a partir da saída do codificador e cujas consultas são calculadas a partir da saída do segmento anterior.
3. **Segmento de Propagação:** Composto por uma pequena rede neural no mesmo esquema que o segmento de propagação do codificador.

Como no caso do codificador, após cada segmento há uma conexão residual seguida por uma normalização de camada. Após a propagação por todos os blocos, a saída do decodificador é passada por uma camada linear seguida por uma função softmax. Tal saída é interpretada como a distribuição de probabilidades da próxima chave na sequência de saída.

### 3.1.3 Regime de Treino

Como a saída do Transformer é uma distribuição de probabilidade, o Transformer funciona essencialmente como um classificador. Dessa maneira, ele é treinado com uma função objetivo de entropia cruzada categórica.

Em termos de tarefas, o Transformer original de Vaswani et al. foi treinado em tarefas de tradução de inglês para alemão e inglês para francês. No entanto, outras aplicações são possíveis.

## 3.2 Implementações

Desde a publicação do artigo seminal, implementações de Transformers continuam a surgir. Nesta seção, consideraremos alguns modelos e famílias de modelos prevalentes na literatura.

De modo algum exaustiva, a listagem a seguir possui caráter expositivo. Aspectos técnicos serão considerados em maior detalhe na próxima seção. Omissos dessa listagem estão modelos especificamente projetados para enfrentar desafios arquiteturais que serão abordados no Capítulo 4. Dentro de cada arquitetura proposta, há diversos modelos de tamanhos distintos. Sendo assim, nomes próprios serão frequentemente usados nessa seção para se referir a uma série de modelos de origem comum ou ao modelo base de um artigo.

### 3.2.1 AL-Rfou

Citado frequentemente na literatura, (AL-RFOU et al., 2019) aplica a abordagem geral do Transformer a tarefas de PLN a nível de caractere. Demais modelos no ramo frequentemente operam a nível de palavra ou pequenas sequências de símbolos.

Notavelmente, os Transformers propostos nesse artigo seguem uma modificação da arquitetura original. Os modelos T12 e T64 apresentados consistem em uma cadeia de blocos consecutivos, sem a partição entre codificador e decodificador. Cada bloco é composto por:

1. **Segmento de Autoatenção:** Construído nos conformes do *decodificador* do Transformer original, ou seja, com a aplicação de uma máscara.
2. **Segmento de Propagação:** Nos moldes do Transformer original.

Além disso, há uma conexão residual e uma normalização de camada após cada um desses segmentos. Com isso, os autores introduzem um novo tipo de modelo Transformer: o *modelo autoregressivo* (em inglês, *autoregressive model*). Nessa abordagem, o modelo recebe uma sequência e produz as probabilidades condicionais para o próximo elemento. A sequência resultante pode então ser realimentada ao modelo.

Os Transformers introduzidos nesse artigo são particularmente profundos comparados ao original. As quantias nos nomes desses modelos indicam a quantidade blocos utilizados, 12 e 64 respectivamente. Os autores ainda propõem algumas pequenas modificações à arquitetura base. Avaliados em tarefas de predição de texto, os modelos T12 e T64 apresentados atingem um desempenho estado-da-arte segundo os autores.

### 3.2.2 Série GPT

Desenvolvido pela OpenAI, o GPT (RADFORD; NARASIMHAN, 2018) é o primeiro de uma série impactante de modelos. O nome surge do procedimento de *pré-treino generativo* (em inglês, *generative pre-training*) proposto pelos autores para Transformers. Como em AL-Rfou et al., os autores optam por um Transformer autoregressivo, com 12 blocos, e introduzem diversas modificações pequenas à arquitetura original. O GPT é construído sob uma filosofia de modelos flexíveis e reutilizáveis que inspirará o resto da série e será analisada em mais detalhes na próxima seção.

Em (RADFORD; WU et al., 2019), os pesquisadores da OpenAI introduzem o GPT-2. Fruto de aprimoramentos à arquitetura do GPT, o GPT-2 possui parâmetros mais de uma ordem de magnitude mais numerosos que seu antecessor e é treinado em um conjunto de dados especialmente preparado. Além de apresentar um modelo mais poderoso, o artigo instiga reflexões sobre o desenvolvimento de modelos a base de Transformers para PLN.

Concluindo a série, (BROWN et al., 2020) divulga o GPT-3. Com 175 bilhões de parâmetros, o GPT-3 representa possivelmente um dos maiores modelos neurais jamais treinados. Construído sobre a arquitetura do GPT-2, o GPT-3 dá continuidade às discussões iniciadas no artigo do GPT-2 e demonstra tremendos ganhos de desempenho.

### 3.2.3 BERT e Derivados

Apresentado em (DEVLIN et al., 2019), o BERT representa um dos modelos mais populares baseados em Transformers. Forma abreviada de *Bidirectional Encoder Representations from Transformers* (em português, *Representações de Codificador Bidirecionais a partir de Transformers*), seu nome indica a ideia central por trás da construção do modelo. Como consideramos, o Transformer original segue uma arquitetura codificador-decodificador. Por contrapartida, o BERT é composto apenas pela unidade codificadora do Transformer. Adaptando a camada final do codificador, é possível adequar o BERT a uma variedade de tarefas de PLN. Além disso, a escolha de arquitetura permite que o BERT processe sequências em ambos sentidos, diferentemente do Transformer original que as processa da esquerda para a direita. Como exemplo, o BERT é capaz de produzir probabilidades condicionais para um elemento desconhecido no meio de uma sequência de entrada com base nos elementos que o antecedem e o sucedem. Essa abordagem é conhecida como *modelagem de linguagem mascarada* (em inglês, *masked language modeling - MLM*) e rendeu ao modelo grande sucesso, além de diversos resultados estado-da-arte em tarefas de PLN na época de publicação.

Em (LIU et al., 2019), os autores propõem um aprimoramento do BERT conhecido como RoBERTa. Conhecido por extenso como *Robustly Optimized BERT Pretraining Approach* (em português, *Abordagem de Pré-treino para o BERT Robustamente Otimizada*), o modelo apresentado explora um ajuste fino de hiperparâmetros do BERT como também novas estratégias de treino. Em particular, os autores descobrem o benefício de treinos mais longos com lotes maiores e mais dados, além de treinos com aritmética de ponto flutuante de precisão mista. Segundo os autores, RoBERTa resulta em um ganho significativo de desempenho sobre o BERT original como também resultados estado-da-arte em tarefas de PLN.

Outros derivados do BERT existem. Até o fim deste trabalho, consideraremos diversos variantes como: MobileBERT (SUN et al., 2020), DistilBERT (SANH et al., 2019) e TinyBERT (JIAO et al., 2020).

Análises de literatura como *Revealing the Dark Secrets of BERT* (KOVALEVA et al., 2019) e *A Primer in BERTology: What We Know About How BERT Works* (ROGERS; KOVALEVA; RUMSHISKY, 2020) evidenciam o ramo de pesquisa ativo em torno do BERT.

### 3.2.4 T5 e mT5

Proposto em (RAFFEL et al., 2020), o *Text-To-Text-Transfer Transformer* (T5) representa uma nova abordagem para Transformers. Longe de ser confinado a um objetivo, o T5 é especificamente projetado para realizar diversas tarefas e também poder aprender novas tarefas. Incorporando aprimoramentos recentes em Transformers, os autores exploram as modificações necessárias para obter um modelo multitarefa e demonstram os seus resultados positivos. Tal visão de modelos reutilizáveis será analisada em maiores detalhes na próxima seção. Além da introdução de uma nova abordagem arquitetural, os autores apresentam um novo conjunto de dados massivo preparado especificamente para o treino de Transformers.

Estendendo a abordagem do T5, (XUE et al., 2020) introduz um variante multilíngue chamado mT5. Como o conjunto original de treino do T5 era exclusivamente de língua inglesa, os autores compilaram um conjunto de dados em vários idiomas para o desenvolvimento de um modelo nos moldes do T5.

### 3.3 Aspectos Técnicos e Aprimoramentos

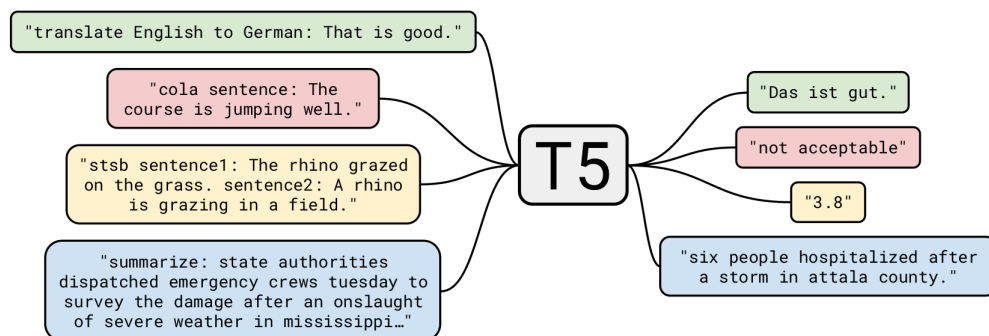
Ao longo do desenvolvimento dos Transformers, alguns conceitos chave se consolidaram na comunidade. Nas subseções a seguir, consideraremos tópicos relevantes para a compreensão contemporânea desses modelos de processamento de linguagem natural.

#### 3.3.1 Aprendizado por Transferência

Transformers herdam a tradição de reutilização de modelos em aprendizado profundo. Diante dos custos computacionais significantes para treinar um modelo neural, é proveitoso adaptar um modelo previamente treinado para uma nova tarefa. O processo de retrainar um modelo pronto usando um novo conjunto de dados é conhecido como *aprendizado por transferência* (em inglês, *transfer learning*). Visando a reutilização de modelos, diversos artigos oferecem modelos de uma mesma arquitetura em tamanhos diferentes. A disponibilidade pública de modelos como BERT e RoBERTA justifica, parcialmente, as suas popularidades.

Modelos multitarefa também se unem aos esforços de economia de poder computacional. O T5, em particular, admite uma entrada adicional que especifica qual tarefa deve ser realizada com a sequência fornecida. A Figura 3.2 ilustra como entradas referentes a diferentes tarefas são alimentadas ao modelo T5. Nesse sentido, o modelo reaproveita a sua estrutura para aplicações distintas. Essa construção também está presente no mT5.

Figura 3.2: Exemplificação em (RAFFEL et al., 2020) de entradas do T5.



Não obstante, veremos que avanços em regimes de treino representam algumas das maiores contribuições à reutilização de modelos.

#### 3.3.2 Regime de Treino

Embora o Transformer original tenha sido treinado diretamente em uma tarefa de tradução, um outro regime de treino se popularizou no ramo. Em (HOWARD; RUDER, 2018), os autores sugerem uma abordagem geral de treino para modelos neurais em PLN. A técnica consiste em dois passos:

1. Pré-treinar o modelo em uma tarefa de modelagem de linguagem em um corpo de domínio geral.
2. Ajustar o modelo para a tarefa objetivo.

Encontrada em todos os Transformers citados na seção anterior (exceto os de Al-Rfou et al., cujos objetivos já são modelagem de linguagem), a abordagem proposta é hoje dominante na literatura. O procedimento exato de pré-treino varia entre implementações. Contudo, como toda modelagem de linguagem essencialmente consiste em prever trechos faltantes de um texto maior, não analisaremos em detalhes as diferenças entre implementações. Ainda assim, o conceito de pré-treino é fundamental.

Segundo Howard e Ruder, o estágio de pré-treino resulta em grandes melhorias no desempenho de um modelo neural e possibilita o aprendizado usando poucos exemplos (em inglês, *low-shot learning* ou *few-shot learning*). Exponente da abordagem, o GPT foi uma das primeiras aplicações da técnica a Transformers e constatou os ganhos com o pré-treino. Seus sucessores, GPT-2 e GPT-3, foram além, demonstrando a capacidade de modelos pré-treinados de realizar tarefas com mínimo ajuste. No momento de publicação, ambos os modelos obtiveram resultados estado-da-arte em diversas tarefas utilizando poucos exemplos e, em alguns casos, nenhum exemplo (chamado em inglês de *zero-shot learning*). Diante desses resultados de destaque, (RADFORD; WU et al., 2019) e (BROWN et al., 2020) propõem a redução das diversas tarefas de PLN a mera modelagem de linguagem. Segue o argumento que um modelo suficientemente pré-treinado será capaz de realizar tarefas tipicamente alcançadas através de treino supervisionado direto. Nesse caso, o passo 1 da abordagem de Howard e Ruder seria suficiente. Tal visão ainda sugere que modelos pré-treinados são multitarefa por essência e formam bons candidatos para reuso.

Com a prevalência do procedimento de pré-treino, conjuntos de dados especialmente preparados para o propósito passaram a surgir. Esforços de mineração de texto na Internet resultaram em conjuntos como o *WebText* (GPT-2), *OpenWebText* (RoBERTa) e versões filtradas ou aprimoradas do conjunto *Common Crawl* como *Colossal Clean Crawled Corpus* (T5), *mC4* (mT5), *CC-News* (RoBERTa) e a versão filtrada do Common Crawl usada pelo GPT-3.

### 3.3.3 Tarefas de Referência

Há uma grande diversidade de tarefas de PLN que podem ser abordadas através de Transformers. Avaliar as diferenças de desempenho entre modelos nessas tarefas nem sempre é trivial. Como critério de comparação, a comunidade acadêmica possui uma série de tarefas de referência para avaliação. Tais tarefas são frequentemente citadas em artigos. A tabela a seguir contém algumas aplicações de Transformers prevalentes na literatura e exemplos de tarefas de referência para cada uma delas.

Tabela 3.1: Tarefas de Referência para Modelos de PLN

Tarefa	Descrição	Exemplos de Tarefas de Referência
Modelagem de Linguagem	Inferir trechos faltantes de um texto.	<ul style="list-style-type: none"> <li>• <b>LAMBADA</b> - LAn-guage Modeling Broa-dened to Account for Discourse Aspects (PA-PERNO et al., 2016)</li> </ul>

Tradução	Converter texto de um idioma para outro.	<ul style="list-style-type: none"> <li>• Desafios <b>WMT</b>: English-German, English-French, English-Romanian, etc.</li> </ul>
Respostas a Perguntas	Dado um texto, responder perguntas sobre ele.	<ul style="list-style-type: none"> <li>• <b>SQuAD</b> - Stanford Question Answering Dataset (<a href="#">RAJPURKAR</a>; <a href="#">ZHANG et al., 2016</a>)</li> <li>• <b>SQuAD 2.0</b> - Stanford Question Answering Dataset 2.0 (<a href="#">RAJPURKAR</a>; <a href="#">JIA</a>; <a href="#">LIANG, 2018</a>)</li> <li>• <b>TriviaQA</b> (<a href="#">JOSHI et al., 2017</a>)</li> <li>• <b>CoQA</b> - Conversational Question Answering Challenge (<a href="#">REDDY</a>; <a href="#">CHEN</a>; <a href="#">MANNING, 2019</a>)</li> <li>• <b>RACE</b> - Large-scale Reading Comprehension Dataset From Examinations (<a href="#">LAI et al., 2017</a>)</li> </ul>

Inferência de Linguagem Natural	Tarefas envolvendo raciocínio lógico em cima de um texto.	<ul style="list-style-type: none"> <li>• <b>SWAG</b> (<a href="#">ZELLERS; BISK et al., 2018</a>)</li> <li>• <b>HellaSwag</b> (<a href="#">ZELLERS; HOLTZMAN et al., 2019</a>)</li> <li>• <b>QNLI</b> - Question-answering NLI (<a href="#">WANG; SINGH et al., 2018</a>)</li> <li>• <b>MNLI</b> - Multi-Genre Natural Language Inference Corpus (<a href="#">WILLIAMS; NANGIA; BOWMAN, 2018</a>)</li> <li>• <b>SNLI</b> - Stanford Natural Language Inference Corpus (<a href="#">BOWMAN et al., 2015</a>)</li> </ul>
Resumo de Texto	Dado um texto, produzir uma versão sintetizada do conteúdo original.	<ul style="list-style-type: none"> <li>• <b>CNN/Daily Mail Summarization Task</b> (<a href="#">SEE; LIU; MANNING, 2017</a>)</li> </ul>
Classificação Gramatical	Identificar a categoria gramatical de uma palavra em uma frase.	<ul style="list-style-type: none"> <li>• <b>PTB</b> - Penn Treebank (<a href="#">MARCUS; SANTORINI; MARCINKIEWICZ, 1993</a>)</li> </ul>
Compreensão de Linguagem Natural	Tarefas envolvendo a interpretação de um texto.	<ul style="list-style-type: none"> <li>• <b>Story Cloze Test</b> (<a href="#">MOSTAFAZADEH; CHAMBERS et al., 2016</a>) e (<a href="#">MOSTAFAZADEH; ROTH et al., 2017</a>)</li> </ul>



Identificação de Similaridade Semântica	Determinar se dois textos capturam a mesma informação.	<ul style="list-style-type: none"> <li>• <b>MRPC</b> - Microsoft Research Paraphrase Corpus (<a href="#">DOLAN; BROCKETT, 2005</a>)</li> <li>• <b>QQP</b> - Quora Question Pairs (<a href="#">ILYER; DANDEKAR; CSERNAI, 2017</a>)</li> </ul>
Aceitabilidade Linguística	Verificar se um trecho está gramaticalmente correto.	<ul style="list-style-type: none"> <li>• <b>CoLA</b> - Corpus of Linguistic Acceptability (<a href="#">WARSTADT; SINGH; BOWMAN, 2018</a>)</li> </ul>
Avaliações Gerais	Agrupamentos de diversas tarefas para determinar o desempenho geral do modelo.	<ul style="list-style-type: none"> <li>• <b>GLUE</b> - General Language Understanding Evaluation (<a href="#">WANG; SINGH et al., 2018</a>)</li> <li>• <b>SuperGLUE</b> (<a href="#">WANG; PRUKSACHATKUN et al., 2019</a>)</li> </ul>

### 3.4 Situação Atual

Neste capítulo, consideramos o que são Transformers e como funcionam. Desde a sua introdução, esses modelos têm demonstrado uma grande capacidade para tarefas de PLN. A arquitetura é hoje considerada o estado da arte no ramo e novos aprimoramentos e estratégias continuam a ser propostos. Somada às inovações arquiteturais, a crescente disponibilidade de dados e poder computacional potencializa ainda mais os Transformers. No entanto, o sucesso traz novos desafios e instiga discussões sobre a viabilidade dessa arquitetura no futuro. No próximo capítulo, consideraremos os desafios enfrentados por Transformers e a busca por melhorias.

## Capítulo 4

# Desafios de Transformers

Com a popularidade dos Transformers, o futuro da arquitetura é hoje um tema de grande interesse. Nos últimos anos, implementações revelaram tanto o alto desempenho dos Transformers como as fraquezas da arquitetura. Como consequência, há hoje uma vasta literatura voltada à viabilidade desses modelos no futuro. Naturalmente, essa discussão é fundamentada pelos desafios enfrentados pelos Transformers. Dois são de destaque:

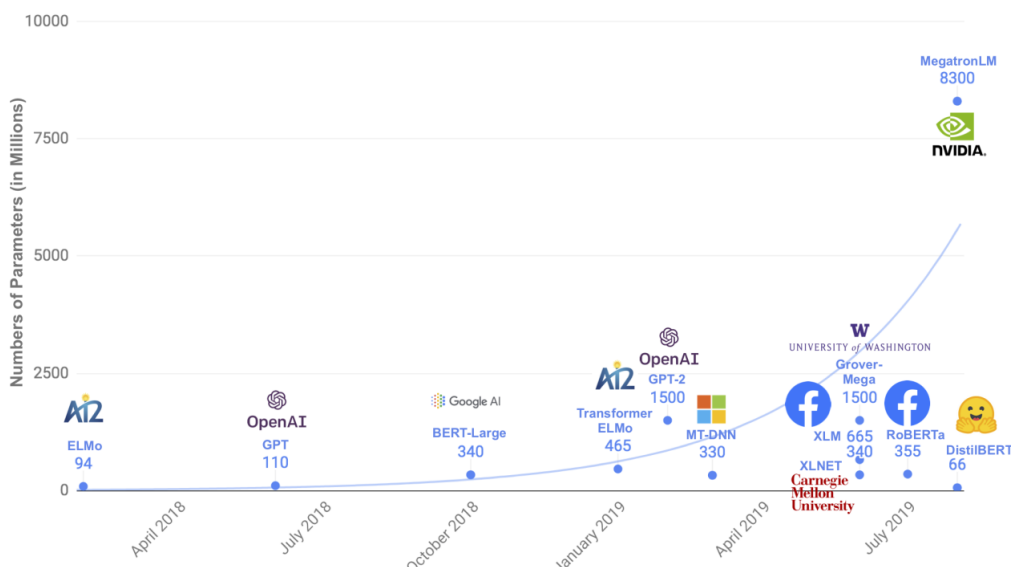
- Consumo crescente de recursos de memória.
- Complexidade computacional quadrática (com relação ao tamanho da sequência).

Nesse capítulo, analisaremos cuidadosamente cada uma dessas dificuldades. Nos capítulos seguintes, consideraremos as propostas na literatura para sanar ou mitigar esses desafios.

### 4.1 Consumo de Memória

Ganho de desempenho com escala é um fenômeno amplamente reconhecido em aprendizado profundo. Longe de fugirem do padrão, Transformers acompanham os demais modelos neurais. Esforços para aumentar Transformers continuam a render resultados de destaque e sugerem uma tendência geral de melhoria com escala. Artigos como os dos modelos GPT-2, GPT-3, T5, BERT e RoBERTa reafirmam tais ganhos.

Figura 4.1: Número de parâmetros em modelos de PLN ao longo do tempo. Figura retirada de (SANH et al., 2019).



Naturalmente, tais constatações motivam modelos cada vez maiores em busca de desempenho. Como resultado, pesquisadores observam um crescimento vertiginoso de parâmetros nos Transformers ao longo dos últimos anos, conforme ilustrado na Figura 4.1. Potencialmente, as tendências apontam para um futuro no qual modelos baseados na arquitetura são cada vez mais difíceis de construir e executar devido à escassez de recursos computacionais.

## 4.2 Complexidade Quadrática

A questão de complexidade computacional é outro desafio significativo da arquitetura. Conforme consideramos no Capítulo 3, mecanismos de atenção representam componentes essenciais dos Transformers. Tanto blocos de codificador como de decodificador contêm mecanismos de atenção.

No entanto, segundo os autores de Vaswani et al., o mecanismo de autoatenção possui complexidade temporal de  $O(n^2 \cdot d)$ , onde  $n$  é o tamanho da sequência e  $d$  a dimensão de representação. Tal complexidade surge do cálculo da pontuação de similaridade, mais especificamente do produto das matrizes de chaves e consultas. Portanto, quanto maior a entrada de um Transformer, mais computacionalmente exigente será o seu processo de treino e a inferência no momento de aplicação. Tal característica impõem um obstáculo notável para o uso de Transformers para processamento de texto.

## 4.3 Propostas de Soluções

Inovações de arquitetura e implementação para Transformers são apresentadas como as principais soluções para os desafios descritos. Dentro da literatura, artigos tipicamente divulgam novos modelos incorporando tais inovações e demonstrando seu desempenho através de tarefas de referência.

Deste ponto em diante, este trabalho considerará as diferentes propostas encontradas na literatura para os desafios apresentados. No Capítulo 5, consideraremos inovações propostas para mecanismo de atenção dos Transformers, visando primariamente atacar o desafio de complexidade computacional. Em seguida, no Capítulo 6, analisaremos técnicas de compressão, ou redução, de modelos que visam reduzir a pegada de memória de modelos do tipo Transformer.

## Capítulo 5

# Alternativas para Atenção

Desde a sua introdução no Transformer original, o mecanismo de atenção recebe diversas propostas de aprimoramento para reduzir sua complexidade computacional. Na literatura, duas vertentes gerais de pesquisa existem:

1. Redução no uso da operação de atenção.
2. Substituição da pontuação de similaridade.

Predominam na primeira vertente técnicas de *atenção esparsa*. Atenção esparsa consiste em usar apenas uma parcela das consultas, chaves e valores recebidos para computar a operação de atenção. Em contraste com o Transformer original que aplica atenção sobre todos os elementos da sequência, modelos desse tipo buscam reduzir a quantidade de elementos contemplados a uma quantidade tratável, sem perder considerável desempenho. Determinar quais elementos devem ser contemplados ou não representa o grande desafio dessa abordagem.

Outra vertente consiste em substituir diretamente a pontuação de similaridade do Transformer original. Chamamos de pontuação de similaridade o seguinte segmento da operação de atenção.

$$\text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)$$

Na literatura, a expressão acima é também frequentemente conhecida como *matriz de atenção*. Note que a expressão compara vetores e fornece pontuações de similaridade na escala de 0 a 1. Operações matemáticas que realizam a mesma tarefa (ou aproximam a operação de atenção original) com complexidade sub-quadrática são candidatas para essa abordagem.

Em ambos os casos, técnicas novas são tipicamente introduzidas através de novos modelos. A listagem a seguir busca capturar as ideias predominantes na literatura contemporânea através de uma análise cuidadosa de modelos propostos. Por facilidade de compreensão, os modelos estão organizados de acordo com a vertente principal que seguem.

### 5.1 Uso Limitado de Atenção

#### 5.1.1 Transformer-XL

Pioneiro na viabilização de Transformers, (DAI et al., 2019) constrói sobre a ideia de janelas de contexto. Janelas de contexto consistem em restringir o acesso de um modelo sequência-a-sequência a elementos próximos da posição atual na sequência de entrada. Uma limitação dessa abordagem é que o modelo se torna incapaz de modelar dependências distantes, diferentemente do Transformer original.

Como solução, Dai e outros propõem construir um Transformer baseado em janelas de contexto, mas com estados ocultos. A estratégia pode ser descrita da seguinte forma. Dada uma

sequência de entrada  $x$  de tamanho arbitrário, os autores particionam  $x$  em segmentos consecutivos de tamanho  $L$ . Dessa maneira, é possível representar o  $t$ -ésimo segmento e o  $t + 1$ -ésimo segmento consecutivos como

$$s_t = [x_{t,1}, \dots, x_{t,L}]$$

$$s_{t+1} = [x_{t+1,1}, \dots, x_{t+1,L}]$$

Em seguida, os autores montam um Transformer com tamanho de entrada  $L$ , mas que retém alguma informação no modelo após o processamento de cada segmento. Como o Transformer é composto por blocos, a saída de cada bloco pode ser utilizada como informação residual. Denotamos por  $h_t^n \in \mathbb{R}^{L \times d}$  a saída do  $n$ -ésimo bloco na iteração  $t$ . A cada iteração, armazenamos o seguinte estado oculto:

$$\tilde{h}_{t+1}^{n-1} = [SG(h_t^{n-1}); h_{t+1}^{n-1}]$$

onde  $[u; v]$  indica a concatenação de  $u$  e  $v$ , enquanto  $SG(\cdot)$  indica o valor do argumento antes de sua atualização via gradiente. Através desse estado oculto, é possível gerar as consultas, chaves e valores a serem usados no próximo bloco.

$$q_{t+1}^n = h_{t+1}^{n-1} W_q^\top$$

$$k_{t+1}^n = \tilde{h}_{t+1}^{n-1} W_k^\top$$

$$v_{t+1}^n = \tilde{h}_{t+1}^{n-1} W_v^\top$$

Consequentemente temos,

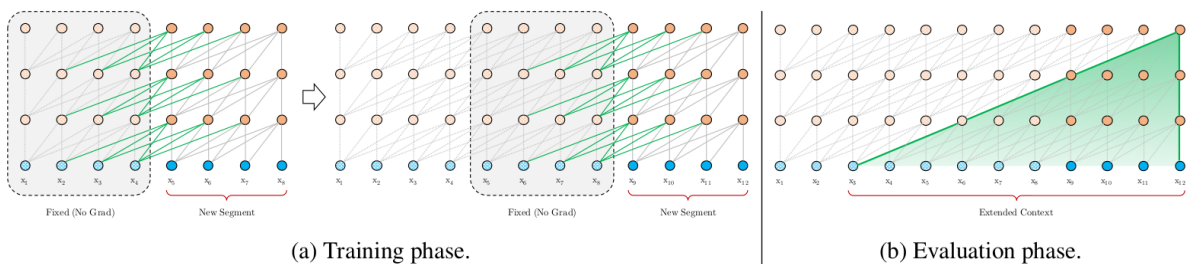
$$h_{t+1}^n = \text{Bloco-Transformer}(q_{t+1}^n, k_{t+1}^n, v_{t+1}^n)$$

Por meio da construção acima, é possível ter um modelo que processa a sequência de entrada por partes, enquanto preserva informação de dependências distantes. A operação desse derivado do Transformer nas fases de treino e avaliação, respectivamente, é exemplificada na Figura 5.1. Note que, na formulação acima, armazenamos apenas a informação do segmento anterior no estado oculto. No entanto, é possível reter os estados ocultos de uma quantidade arbitrária de segmentos anteriores. Nesse caso, basta substituir  $SG(h_t^{n-1})$  por  $SG(m_t^{n-1})$  com  $m_t^n \in \mathbb{R}^{M \times d}$  e  $M$  hiperparâmetro.

O modelo resultante é o **Transformer-XL**. Em termos de complexidade, o Transformer-XL retém um mecanismo de atenção de complexidade quadrática, mas agora relativo ao tamanho da janela  $L$  ao invés do tamanho  $n$  da sequência de entrada. Essa modificação traduz em um ganho de velocidade do modelo, embora o consumo de memória cresça devido à necessidade de armazenar os estados ocultos. No momento de publicação, o Transformer-XL obteve resultados estado-da-arte em uma série de experimentos.

Como observação, a abordagem proposta exige modificações à codificação posicional das embeddings. Em particular, Dai e outros usam uma codificação de posição relativa para viabilizar a técnica. Devido ao baixo impacto dessa codificação na complexidade computacional da técnica, não entraremos em maiores detalhes.

Figura 5.1: Funcionamento de um Transformer-XL com  $L = 4$  durante a fase de treino (a) e a fase de avaliação (b), conforme ilustrado em (DAI et al., 2019).



### 5.1.2 Sparse Transformer

Apresentado em (CHILD et al., 2019), o **Sparse Transformer** (em português, *Transformer Esperso*) explora o uso de atenção esparsa sem depender de estados ocultos. Como mencionado, o cerne dessa estratégia consiste em determinar quais elementos da sequência serão contemplados. Portanto, os autores propõem a seguinte notação para definir essas parcelas. Denotamos por  $S_i$  o conjunto dos índices das entradas a serem usadas na  $i$ -ésima iteração. No Transformer original,

$$S_i = \{j : j \leq i\}$$

para todas as  $p$  cabeças de atenção. A expressão acima indica que, a cada iteração, todo elemento atende a todas as posições anteriores e à sua própria posição. Por contrapartida, o Sparse Transformer propõe escolhas diferentes de  $S_i$  para cada cabeça. Os autores denotam a escolha de índices da  $m$ -ésima cabeça como

$$\begin{aligned} A_i^{(m)} &\subset \{j : j \leq i\} \\ S_i &= A_i^{(m)} \end{aligned}$$

Em seguida, Child e outros apresentam duas propostas de escolha de índices para o caso específico  $p = 2$ . Segundo os autores, o raciocínio dessas propostas generaliza para dimensões maiores. Ambas as propostas envolvem definir um tamanho de passo  $l$  próximo a  $\sqrt{n}$ . Propriamente ditas, as propostas são:

#### 1. Atenção por Passo Largo:

A proposta consiste em tomar

$$\begin{aligned} A_i^{(1)} &= \{t, t+1, \dots, i\} \\ A_i^{(2)} &= \{j : (i-j) \bmod l = 0\} \end{aligned}$$

com  $t = \max(0, i-l)$ . De acordo com os autores, essa abordagem é particularmente útil para dados com alguma natureza periódica, como imagens e alguns tipos de música.

#### 2. Atenção por Padrão Fixo:

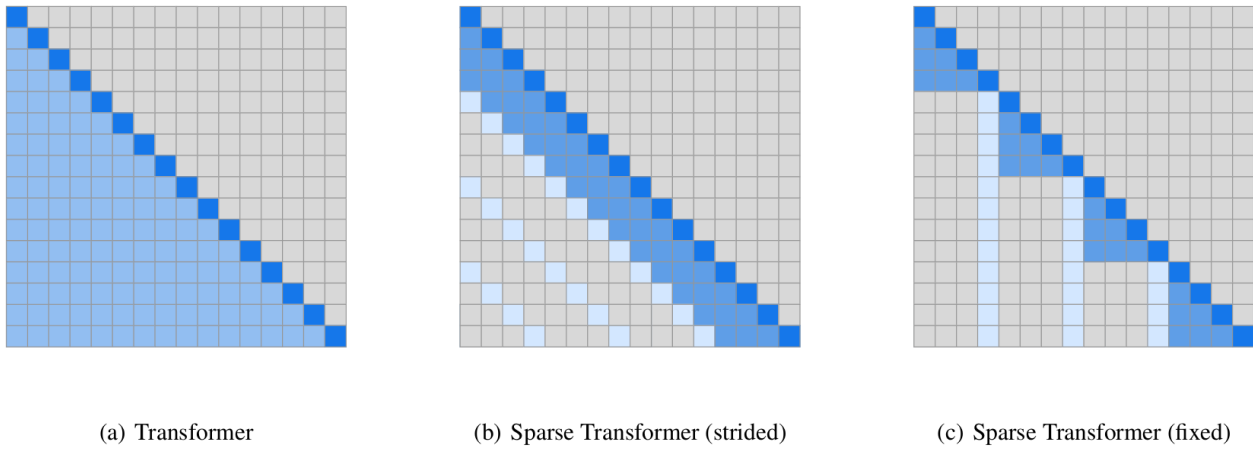
Nessa proposta, os autores tomam

$$\begin{aligned} A_i^{(1)} &= \left\{ j : \left( \left\lfloor \frac{j}{l} \right\rfloor = \left\lfloor \frac{i}{l} \right\rfloor \right) \right\} \\ A_i^{(2)} &= \{j : j \bmod l \in \{t, t+1, \dots, l\}\} \end{aligned}$$

com  $t = l - c$  e  $c$  hiperparâmetro.

Exemplificadas na Figura 5.2 estão as diferenças de operação entre os dois padrões propostos e o Transformer original. Aplicados, os critérios acima resultam em uma complexidade de  $O(n\sqrt{n})$ . Naturalmente, a complexidade da abordagem é fortemente guiada pela escolha dos hiperparâmetros  $l$  e  $c$ .

Figura 5.2: Visualização em (CHILD et al., 2019) comparando o funcionamento da atenção no Transformer original (a), na abordagem por passo largo (b) e na abordagem por padrão fixo (c).



Além da adaptação ao mecanismo de atenção, o artigo discute detalhes de implementação e outros aperfeiçoamentos. Modificações de destaque incluem: reestruturação das conexões residuais, alterações na inicialização de parâmetros, troca da função de ativação das camadas densas pela *Gaussian Error Linear Unit* – GELU (em português, *Unidade Linear de Erro Gaussiano*) (HENDRYCKS; GIMPEL, 2016), embeddings construídas a partir de aprendizado, uso de pontos de controle de gradiente (em inglês, *gradient checkpointing*) (CHEN et al., 2016) (GRUSLYS et al., 2016) e treino com representação em ponto flutuante de precisão mista (MICIKEVICIUS et al., 2018). Como o objetivo deste capítulo é introduzir inovações em atenção, não entraremos em maiores detalhes sobre esses outros aprimoramentos.

### 5.1.3 BP-Transformer

Introduzido em (YE et al., 2019), o Binary Partitioning Transformer ou **BP-Transformer** (em português, *Transformer de Paricionamento Binário*) introduz uma abordagem baseada em grafos para aplicar atenção de maneira seletiva à sequência de entrada. Conceitualmente, a abordagem consiste em definir:

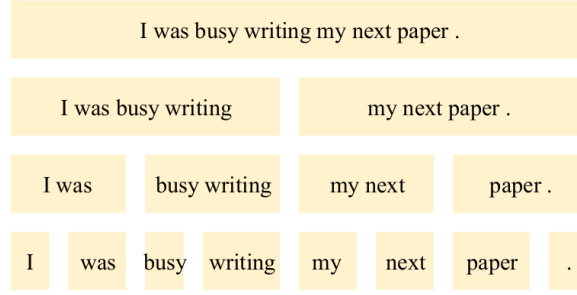
- Uma árvore binária de hierarquia.
- Um grafo de atenção.

Analisaremos cada um desses elementos e a implementação de atenção resultante do modelo.

#### Árvore Binária de Hierarquia

O primeiro passo da abordagem consiste em representar a sequência de entrada como uma árvore binária. Como raiz, a árvore tem um nó referente à sequência de entrada inteira. Os demais nós na árvore são referentes a subsequências da sequência original. Construímos esses nós recursivamente através da seguinte regra: cada nó tem dois filhos e cada filho é referente a uma metade da subsequência do pai. Quando cada nó é composto de apenas um elemento, a geração recursiva termina. O resultado é uma árvore binária perfeita na qual as folhas têm todas a mesma profundidade. A Figura 5.3 traz um exemplo de uma tal árvore.

Figura 5.3: Particionamento binário conforme ilustrado em (YE et al., 2019).



Como a árvore pode ser interpretada em níveis, introduziremos a seguinte notação. Dado uma árvore de profundidade  $r$ , adotaremos a convenção de chamar o nível das folhas de nível 0 e o nível da raiz de nível  $r$ . Denotaremos por  $u_{l,m}$  o  $m$ -ésimo elemento da esquerda para a direita do  $l$ -ésimo nível da árvore. A partir da hierarquia dessa árvore podemos construir um grafo direcionado de atenção.

### Grafo de Atenção

Construímos o grafo de atenção, denotado por  $\mathcal{G}$ , adicionando *arestas direcionadas de atenção* aos nós da árvore binária. Cada aresta de atenção indica que o nó destino aplica atenção sobre o nó de origem. Consideraremos posteriormente como exatamente atenção é aplicada sob essa estrutura. Por ora, considere que nós em níveis diferentes serão interligados. Os autores então apresentam duas categorias de arestas de atenção a serem adicionadas:

#### 1. Arestas Afiadas

Dada um nó interno  $u_{l,m}$ , adicionamos uma aresta direcionada a partir de cada folha de  $u_{l,m}$  na árvore binária. Formalmente, essas são

$$u_{0,2^l * m + 1}, \dots, u_{0,2^l * (m+1)}$$

O objetivo dessas *arestas afiliadas* (em inglês, *affiliated edges*) é associar toda subsequência a seus elementos base.

#### 2. Arestas Contextuais

*Arestas contextuais* (em inglês, *contextual edges*) capturam o contexto de um elemento na sequência de entrada. Como principal contribuição, o artigo propõe definir esse contexto em graus variados de granularidade através da árvore binária de hierarquia. Nessa abordagem, cada folha é interligada com nós em diferentes níveis da árvore de acordo com sua distância até esse trecho da sequência de entrada. Visando controlar a densidade de conexões em  $\mathcal{G}$ , os autores introduzem um hiperparâmetro  $k$  que indica a quantidade de arestas adicionadas por nível.

Considere agora um nó  $u_{0,i}$ . Sem perda de generalidade, construiremos as arestas contextuais do lado direito de  $u_{0,i}$ . Tomamos os *nós de contexto* de acordo com o seguinte padrão:

$$u_{0,p_0}, \dots, u_{0,p_0+k-1}$$

$$u_{1,p_1}, \dots, u_{1,p_1+k-1}$$

$$\dots$$

$$u_{l,p_l}, \dots, u_{l,p_l+k-1},$$

$$\dots$$



onde  $p_l$  é o índice de início no nível  $l$ , que pode ser computado recursivamente através da regra

$$\begin{aligned} p_0 &= i + 1 \\ p_l &= \text{pai}(p_{l-1} + k) \end{aligned}$$

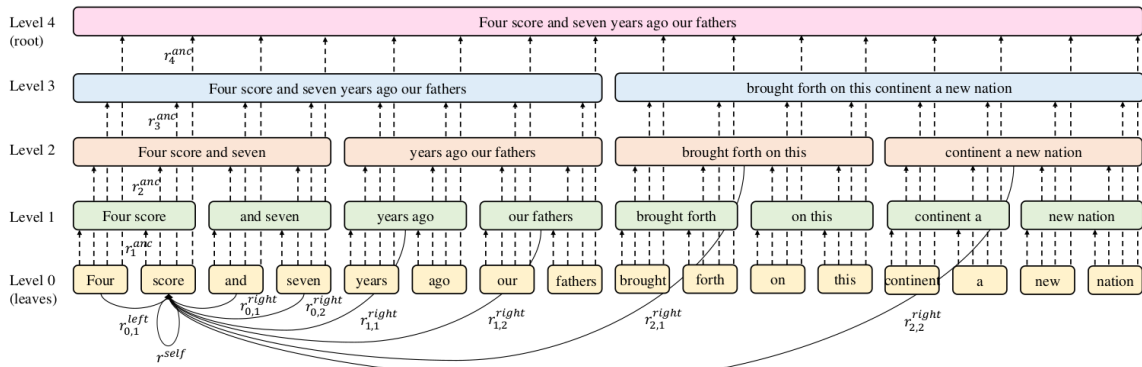
Como detalhe, os autores observam que quando o índice  $p_l + k - 1$  é ímpar, o próximo nó na camada também é adicionado como nó de contexto, por eficiência computacional. Nesse caso teríamos

$$p_{l+1} = \text{pai}(p_l + k + 1)$$

Por fim, construímos arestas com origens em cada nó de contexto e destinos em  $u_{0,i}$ . A construção das arestas do lado esquerdo é análoga.

Um exemplo da construção desse grafo pode encontrada na Figura 5.4.

Figura 5.4: Visualização da construção de  $\mathcal{G}$ , conforme apresentada em (YE et al., 2019). Cores diferentes indicam níveis diferentes. Linhas tracejadas são arestas conectando folhas a nós internos. Linhas sólidas são arestas que levam a folhas.



### Autoatenção de Grafo

Através do grafo de atenção  $\mathcal{G}$ , é possível construir um mecanismo de atenção multicabeça conhecido como *Autoatenção de Grafo* (em inglês, *Graph Self-Attention* - GSA). Nessa abordagem, cada nó  $u$  possui um valor  $h^u \in \mathbb{R}^d$  associado, com  $d$  sendo a dimensão de representação do modelo. Considerando  $\mathcal{A}(u)$  como o conjunto de nós vizinhos de  $u$  em  $\mathcal{G}$ , substituímos a atenção multicabeça convencional por

$$\begin{aligned} A^u &= \text{concat}(\{h_v | \mathcal{A}(u)\}) \\ Q_i^u &= H_k W_i^Q \\ K_i^u &= A^u W_i^K \\ V_i^u &= A^u W_i^V \\ \text{cabeça}_i^u &= \text{softmax}\left(\frac{Q_i^u K_i^{u\top}}{K_i^{u\top} K_i^u}\right) \\ GSA(\mathcal{G}, h_u) &= [\text{cabeça}_1^u, \dots, \text{cabeça}_h^u] W^O \end{aligned}$$

com  $H_k \in \mathbb{R}^{n \times d}$  a entrada,  $h$  o número de cabeças e  $W_i^Q, W_i^K, W_i^V, W^O$  parâmetros treináveis. Como observação, quando  $\mathcal{G}$  é construído, os nós interiores são inicializados com vetores nulos e as

folhas com suas word embeddings correspondentes (conforme mencionado, cada folha é referente a um elemento da sequência de entrada). Atualizamos as representações dos nós atribuindo

$$h^u \leftarrow GSA(\mathcal{G}, h^u)$$

A partir desse ponto, é possível processar a sequência elemento a elemento como no mecanismo original. O resultado é um mecanismo de atenção que aplica atenção sobre uma quantidade de elementos consideravelmente inferior ao tamanho da sequência de entrada.

## Implementação

Visando a eficiência computacional, a implementação do GSA adota uma construção diferente, porém equivalente à descrita acima. Em particular, os autores aplicam algoritmos eficientes para encontrar nós contextuais e atualizar o gráfico (presentes no artigo), e não usam diretamente a árvore para construir  $\mathcal{G}$ . O modelo resultante possui uma complexidade de  $O(k \cdot n \log(\frac{n}{k}))$ . Como outros modelos de atenção esparsa, a abordagem também usa uma embedding posicional relativa.

### 5.1.4 Longformer

O **Longformer** de (BELTAGY; PETERS; COHAN, 2020) combina uma abordagem de janela de contexto com padrões adicionais de atenção. De modo remanescente ao BERT, o Longformer é composto apenas pelo codificador e opera de maneira bidirecional. No entanto, os autores também fornecem um variante nos moldes codificador-decodificador chamado de Longformer-Encoder-Decoder (LED).

Como um modelo de atenção esparsa, o Longformer aplica atenção apenas sobre uma parcela das entradas. Três aspectos guiam essa escolha de parcelas:

1. **Janela de Contexto:** O Longformer limita o uso de entradas a uma janela de contexto centrada na posição atual  $i$ . Dado um tamanho de janela fixo  $w$ , o modelo tem acesso à posição  $i$ , aos  $\frac{w}{2}$  elementos anteriores e aos  $\frac{w}{2}$  elementos posteriores para um hiperparâmetro  $w$ .
2. **Janela Dilatada:** Os autores propõem uma modificação à janela de contexto conhecida como uma janela dilatada. Nesse sistema, a janela possui vãos periódicos de tamanho  $d$  a partir da posição central  $i$ . Por exemplo: se  $d = 2$ , então as posições  $(i + 1)$ ,  $(i + 2)$ ,  $(i - 1)$  e  $(i - 2)$  estão indisponíveis para a aplicação de atenção.
3. **Atenção Global:** Reconhecendo as limitações da janela dilatada, os autores introduzem um critério adicional de seleção conhecido como *atenção global*. De acordo com a aplicação final do modelo, os autores selecionam determinados elementos da sequência de entrada como fichas globais (em inglês, *global tokens*). Fichas globais são elementos que sempre aplicam atenção à sequência inteira. Além disso, todo outro elemento na sequência sempre aplica atenção sobre as fichas globais, independentemente da janela dilatada. Como exemplo de ficha global, os autores fornecem o símbolo especial [CLS] nas entradas do BERT.

Como outros Transformers, o Longformer usa atenção multicabeça. Os autores aproveitam essa característica para escolher diferentes valores de  $w$  e  $d$  para cada cabeça. Usando a notação de

Child e outros, descrevemos o critério final de seleção para cada cabeça do Longformer como

$$A_i^{(m)} = \begin{cases} A^* \cup \{j : a \leq j \leq b\} & \text{caso } d_m = 0 \\ A^* \cup \{j : a \leq j \leq b, (i - j - 1) \bmod d_m = 0\} & \text{caso } d_m > 0 \end{cases}$$

$$a = \max\left(0, i - \frac{w_m}{2}\right)$$

$$b = \min\left(n, i + \frac{w_m}{2}\right)$$

onde  $A^*$  representa os índices dos elementos de atenção global e  $n$  é o tamanho da sequência. No artigo, tais escolhas são feitas de acordo com a aplicação final e em muitos casos a janela não é dilatada.

Em termos de implementação, os autores indicam que uma implementação eficiente exige operações matriciais ainda não suportadas nas bibliotecas PyTorch e Tensorflow no momento de publicação do artigo. O Longformer é implementado através de código customizado na plataforma CUDA. Além disso, os autores usam as embeddings posicionais relativas propostas em (DAI et al., 2019).

### 5.1.5 BigBird

Inspirado pelo BERT, o **BigBird** de (ZAHEER et al., 2020) apresenta novos critérios para atenção esparsa. Os autores argumentam que muitos métodos anteriores são baseados em heurísticas e não tão versáteis e robustos, empiricamente, como o Transformer original. Segundo os autores, o BigBird satisfaz todas as propriedades teóricas de um Transformer pleno, incluindo aproximação universal de funções de sequências e Turing-completude. Como atenção esparsa consiste em restringir a aplicação de atenção a determinados elementos da sequência de entrada, Zaheer e outros propõem os seguintes critérios:

#### 1. Janela de Contexto

Prevalente na literatura, a janela de contexto está presente no BigBird. Como o Longformer, o BigBird implementa janelas de contexto simétricas em torno da posição atual na sequência de entrada. Dessa maneira, cada elemento  $i$  na sequência aplica atenção sobre si mesmo, os  $\frac{w}{2}$  elementos anteriores e os  $\frac{w}{2}$  elementos posteriores.

#### 2. Atenção Global

Outro critério previamente visto na literatura é a atenção global. Conforme considerado anteriormente, as fichas globais sempre aplicam atenção sobre todos elementos da sequência e todo elemento da sequência sempre aplica atenção sobre as fichas globais. No BigBird, os autores constroem um conjunto  $g$  de fichas globais usando uma de duas técnicas:

- **Construção de Transformer Interna** (em inglês, *Internal Transformer Construction* - ITC)
- **Construção de Transformer Estendida** (em inglês, *Extended Transformer Construction* - ETC)

Não entraremos em maiores detalhes sobre essas construções. No entanto, a ETC obtém resultados superiores em experimentos e é nela que fichas especiais como [CLS] são também utilizadas como fichas globais.

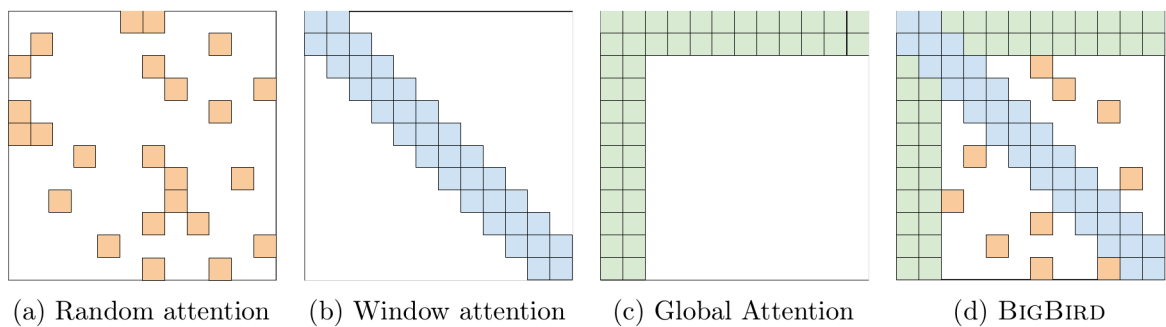
#### 3. Atenção Aleatória

Motivados por resultados na teoria de grafos aleatórios, os autores apresentam a atenção aleatória. Nessa abordagem, para cada elemento na sequência, há um conjunto  $r$  de elementos escolhidos aleatoriamente aos quais atenção será aplicada.

Como em outros modelos de atenção esparsa, os critérios descritos acima são aplicados dentro de um arcabouço de atenção multicabeça. A Figura 5.5 exemplifica os diferentes padrões de atenção propostos e como podem ser combinados. Grande parte do artigo consiste em fornecer justificativas teóricas para a construção do modelo, que fogem do escopo deste trabalho.

Em experimentos, o artigo demonstra empiricamente o alto desempenho do BigBird em diversas tarefas de PLN, superior a modelos como RoBERTa e o Longformer. Além disso, o BigBird atinge resultados estado-da-arte em tarefas de respostas a perguntas e resumo de documentos.

Figura 5.5: Critérios de seleção conforme ilustrados em (ZAHEER et al., 2020). É possível identificar os critérios de atenção aleatória (a), janela de contexto (b), atenção global (c) e a combinação das técnicas em (d).



## 5.2 Alternativas para a Pontuação de Similaridade

### 5.2.1 Linformer

Em (WANG; LI et al., 2020), os autores apresentam o **Linformer**. Por trás desse modelo, há o argumento que a matriz de atenção é de posto baixo. Oriunda de observações tanto empíricas como teóricas, essa afirmação implica que a matriz de atenção poderia ser aproximada através de técnicas de projeção. Nessa subseção, consideraremos a linha de raciocínio por trás desse argumento e o mecanismo de atenção proposto.

#### Argumentos para Baixo Posto da Matriz de Atenção

Como base das avaliações empíricas, os autores pré-treinam dois modelos do tipo RoBERTa (um de 12 blocos e outro de 24 blocos). Em seguida, Wang e outros produzem as decomposições em valores singulares (em inglês, *singular value decompositions* - SVD) das matrizes de atenção de diferentes blocos e cabeças no modelo. Plotando os valores singulares acumulados normalizados referentes a cada matriz, é possível identificar distribuições de cauda longa pelos diferentes blocos, cabeças e tarefas. Com isso, os autores concluem que a maior parte da informação nas matrizes provém de alguns poucos dos maiores valores singulares. Nos blocos superiores, é possível ainda observar que mais informação está concentrada nos maiores valores singulares, o que indicaria um posto menor do que nas matrizes em blocos inferiores.

Em seguida, os autores consideram aspectos teóricos. Usando o lema de Johnson-Lindenstrauss, os autores demonstram que uma matriz de baixo posto suficientemente próxima da matriz de atenção deve existir com alta probabilidade.

#### Mecanismo de Atenção baseado em Projeções

Com essas avaliações, o artigo então progride para as possíveis alterações no mecanismo de atenção. Dada uma matriz de atenção de posto baixo, é possível aproximá-la através dos seus  $i$

maiores valores singulares. No entanto, tal abordagem exigiria computar a SVD de cada matriz de atenção.

Os autores apresentam uma alternativa. Antes de alimentar as chaves e valores à operação de atenção tradicional, elas são projetadas em um espaço menor. Invocando novamente o lema de Johnson-Lindenstrauss, os autores argumentam que devem existir com alta probabilidade projeções  $E$  e  $F$  tais que a operação de atenção com as chaves e valores projetados é suficientemente próxima da operação de atenção convencional. Com isso, o artigo propõe

$$\text{Atenção}_{\text{Linformer}} = \text{Atenção}(Q, EK, FV)$$

onde  $E$  e  $F$  são matrizes de projeção linear que levam as chaves e valores em uma dimensão  $(k \times d)$ . Tal mecanismo de atenção é ilustrado na Figura 5.6. No artigo, os autores escolhem

$$E = \delta R$$

$$F = e^{-\delta} R$$

onde  $R \in \mathbb{R}^{k \times n}$  com entradas i.i.d. de  $N(0, \frac{1}{k})$  e  $\delta$  é uma pequena constante.

Supondo que a matriz de atenção tem posto  $d$ , os autores demonstram teoricamente que é possível escolher um  $k$  que reduz a complexidade das operações matriciais de forma que ela não dependa mais do tamanho da sequência  $n$ , mas da dimensão de representação  $d$ . Nessa argumentação, há uma premissa implícita que  $d \ll n$  e a troca é vantajosa. Nos seus resultados teóricos, os autores consideram

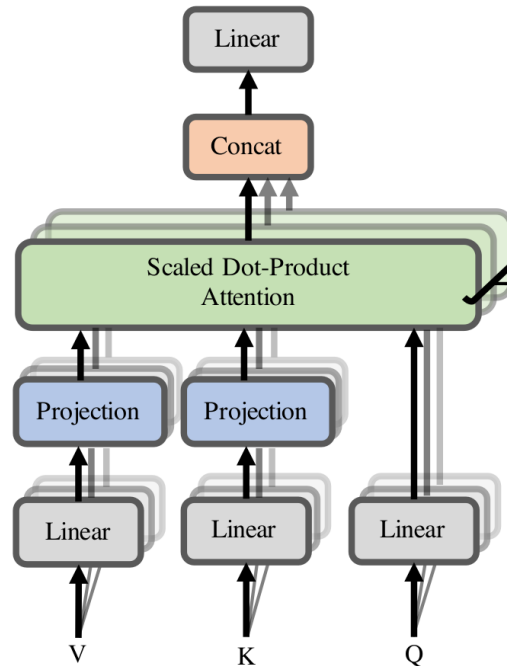
$$k = 9 \frac{\log(d)}{\epsilon^2 - \epsilon^3}$$

$$\delta = \left( \frac{1}{2^n} \right)$$

Na implementação, os autores experimentam com diferentes valores de  $k$ .

Em experimentos, o Linformer alcança desempenho comparável com o Transformer original, o BERT e RoBERTa. No entanto, o seu grande destaque é um tempo inferência significativamente menor que o Transformer original e melhor habilidade para lidar com sequências maiores.

Figura 5.6: Modificações do Linformer ao mecanismo de atenção conforme ilustrado em (WANG; LI et al., 2020).



## Aperfeiçoamentos

Por fim, os autores destacam uma série de aprimoramentos para a abordagem. Otimizações incluem: compartilhamento de parâmetros entre camadas e cabeças, uso de dimensões diferentes de projeção para diferentes camadas e cabeças e dimensões menores encorajadas para camadas maiores. O artigo também considera uso de projeções gerais como *pooling* ou convolução.

### 5.2.2 Hashing Sensível a Localidade

Em (KITAEV; KAISER; LEVSKAYA, 2020), os autores apresentam a abordagem de *atenção por hashing sensível a localidade* (em inglês, *locality-sensitive hashing attention* ou *LSH attention*). A técnica reduz a complexidade do mecanismo de atenção para  $O(n \log n)$ . Como o nome sugere, o conceito de hash forma a base desse trabalho. Nesta subseção, analisaremos as bases matemáticas dessa técnica e sua implementação final em um Transformer.

#### Hashing

Chamamos de *função de hash* uma função que projeta um vetor num espaço de dimensão menor, finita e fixa, cuja saída fica conhecida propriamente como *hash*. Dizemos que uma função de hash é sensível a localidade quando vetores próximos, ao serem passados pela função, obtêm a mesma hash com alta probabilidade. Note que uma função de hash não é reversível. Funções desse tipo podem ser usadas para comparar vetores probabilisticamente.

No artigo, os autores empregam projeções aleatórias nos moldes de (ANDONI et al., 2015) como função de hash sensível a localidade. A técnica pode ser descrita da seguinte forma. Para obter  $b$  hashes diferentes, os autores definem uma matriz aleatória  $R$  de tamanho  $[d, \frac{b}{2}]$ , onde  $d$  é a dimensão de representação do vetor de entrada. Dada uma entrada  $x \in \mathbb{R}^d$ , a função de hash  $h$  é dada por

$$h : \mathbb{R}^d \rightarrow \mathbb{R}^c$$

$$h(x) = \text{argmax}([xR; -xR])$$

onde  $[u; v]$  indica a concatenação de dois vetores. Deste ponto em diante, chamaremos a operação acima simplesmente de LSH.

De natureza estocástica, a LSH representa uma ferramenta aproximada de comparação de vetores. Embora ela não quantifique a relação de vetores distantes, a LSH é capaz de identificar vetores próximos uns dos outros por eles produzirem a mesma hash. Vetores com a mesma hash são agrupados em conjuntos conhecidos como *balde*s (em inglês, *buckets*). Comparada à pontuação de similaridade convencional, a LSH compara vetores com alguma perda de informação, mas não necessariamente uma perda significativa para propósitos práticos.

#### Aplicação no Mecanismo de Atenção

Conforme considerado anteriormente, o mecanismo de atenção convencional recebe chaves, consultas e valores. Como primeiro passo, os autores aplicam a LSH sobre as chaves e consultas. Os resultados da operação são então organizados em baldes da seguinte forma.

$$\mathcal{P}_i = \{j : h(q_i) = h(k_j)\}$$

Interpretamos a expressão acima como o conjunto referente à consulta  $q_i$  com os índices de todas as chaves que obtiveram a mesma hash que  $q_i$ .

Uma estratégia possível, a partir desse ponto, seria aplicar atenção convencional apenas dentro de cada balde. Embora a operação de atenção convencional persista no modelo, a quantidade total de computação é consideravelmente reduzida. No entanto, dois fatos comprometem a implementação dessa estratégia. Primeiramente, os baldes não necessariamente possuem o mesmo

tamanho. Além disso, o número de chaves e consultas dentro de cada balde pode ser desigual. Portanto, os autores propõem uma série de ajustes para viabilizar essa estratégia.

O primeiro ajuste consiste em equiparar chaves e consultas. Isso é feito tomando

$$k_j = \frac{q_j}{||q_j||}$$

Observe que essa modificação garante que  $h(k_j) = h(q_j)$ . Com isso, a nova operação de pontuação de similaridade recebe apenas uma sequência de consultas (que são equivalentes às chaves). Em seguida, os autores ordenam a sequência por balde e ordenam cada balde pela posição na sequência original. Formalmente, os autores apresentam esse procedimento como uma permutação

$$i \mapsto s_i$$

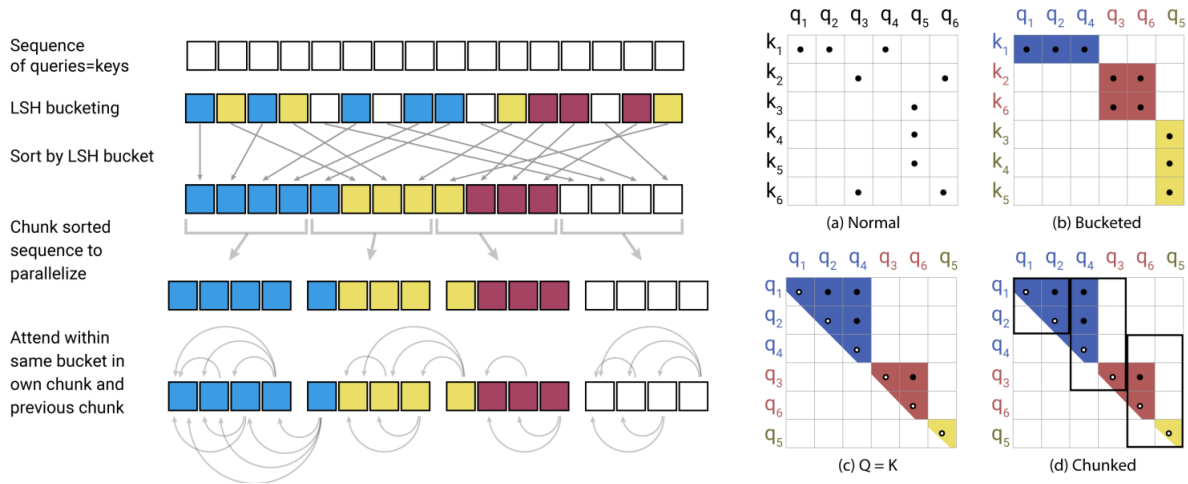
Por fim, a sequência resultante é particionada em fatias (em inglês, *chunks*) de  $m$  consultas consecutivas.

Com essas modificações, os autores definem os critérios para aplicação da operação de atenção convencional. A operação de atenção deve ser aplicada em cada consulta  $q_i$  com referência às demais consultas que estejam simultaneamente

- no mesmo balde que  $q_i$
- na mesma fatia que  $q_i$  ou na fatia anterior

A implementação dessa abordagem resulta numa complexidade significativamente reduzida.

Figura 5.7: Funcionamento de atenção LSH conforme ilustrado em (KITAEV; KAISER; LEVSKAYA, 2020)



## Aperfeiçoamentos

Devido à probabilidade não-nula de dois vetores próximos obterem hashes distintas, o artigo propõe realizar múltiplas rodadas de hashing usando funções de hash  $\{h_1, h_2, \dots\}$  distintas. Dadas  $n$  rodadas, temos

$$\mathcal{P}_i = \bigcup_{r=1}^n \mathcal{P}_i^{(r)} \quad \mathcal{P}_i^{(r)} = \left\{ j : h^{(r)}(q_i) = h^{(r)}(q_j) \right\}$$

Observe que a geração dessas hashes pode ser feito em paralelo. Ilustrado na Figura 5.7 está o procedimento completo de construção desse novo mecanismo de atenção.

## Modelo Resultante

O modelo resultante de (KITAEV; KAISER; LEVSKAYA, 2020) é conhecido como o **Reformer**. Além de modificações à computação de atenção, os autores introduzem alguns outros aprimoramentos. Os principais destaques são técnicas de camadas reversíveis (que serão consideradas no futuro) e particionamento de ativações. Não entraremos em detalhes uma vez que essas características fogem o escopo dessa seção.

### 5.2.3 Métodos de Núcleo

Métodos baseados em núcleos (em inglês, *kernels*) representam ainda outra abordagem para substituir a pontuação de similaridade. Construídos sobre uma base matemática extensa, tais métodos prometem uma complexidade computacional linear, isto é,  $O(n)$ .

Os primórdios da abordagem podem ser encontrados em (TSAI et al., 2019). Nesse artigo, diferentes operações em Transformers são identificadas e caracterizadas através de núcleos. Em (KATHAROPOULOS et al., 2020), os autores apresentam um modelo com atenção baseada em núcleos visando ganhos de complexidade. Por fim, (CHOROMANSKI et al., 2020) constrói sobre a ideia de atenção baseada em núcleos, mas com garantias teóricas mais fortes e menos premissas comparado a outros métodos de atenção.

## Teoria Matemática

Resumidamente, núcleos são funções que comparam vetores. A base de todo núcleo é um *mapa de características* (em inglês, *feature map*), uma função  $\phi$  tal que

$$\begin{aligned}\phi : \mathbb{R}^d &\rightarrow \mathbb{R}_+^r \\ r &> 0\end{aligned}$$

Propriamente dito, o núcleo é a função

$$\begin{aligned}K : \mathbb{R}^d \times \mathbb{R}^d &\rightarrow \mathbb{R}_+ \\ K(x, y) &= \phi(x)^\top \phi(y)\end{aligned}$$

O conceito pode ser estendido para um cenário estocástico. Dado uma mapa  $\phi$  de natureza aleatória, temos

$$\begin{aligned}K : \mathbb{R}^d \times \mathbb{R}^d &\rightarrow \mathbb{R}_+ \\ K(x, y) &= \mathbb{E}[\phi(x)^\top \phi(y)]\end{aligned}$$

Núcleos possuem uma vasta literatura em aprendizado de máquina. Grande parte dela gira em torno de encontrar funções  $\phi$  que produzam núcleos de interesse. No caso do mecanismo de atenção, a busca é por funções  $\phi$  que produzam como núcleo a pontuação de similaridade ou algo semelhante.

Uma análise da operação de atenção demonstra os potenciais ganhos em complexidade. Estendendo a notação matricial do Capítulo 3, (CHOROMANSKI et al., 2020) reescreve a operação de atenção da seguinte maneira.

$$\begin{aligned}\text{Atenção}(Q, K, V) &= D^{-1}AV \\ A &= \exp\left(\frac{QK^\top}{\sqrt{d}}\right) \\ D &= \text{diag}(A1_L)\end{aligned}$$

onde  $\exp(\cdot)$  é aplicada elemento a elemento,  $1_L$  indica um vetor de uns de tamanho  $L$  e  $\text{diag}(\cdot)$  é uma matriz diagonal com o vetor de entrada como a diagonal. Por sua vez,  $L$  representa o tamanho da sequência de entrada na notação dos autores.



Métodos de núcleo para atenção buscam mapas de características cujos núcleos sejam úteis para representar  $A$ . Suponha que existe uma mapa de características para  $A$ . Denotando por  $i$  subscrito a  $i$ -ésima linha de uma matriz, temos

$$\begin{aligned} Q'_i &= \phi(Q_i^\top)^\top \\ K'_i &= \phi(K_i^\top)^\top \end{aligned}$$

Então, em um cenário determinístico

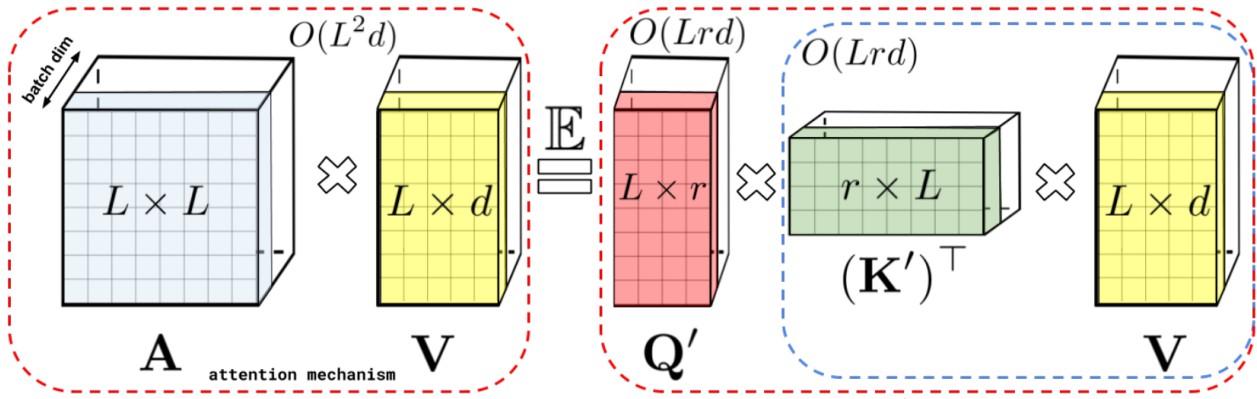
$$\begin{aligned} A &= Q'^\top K' \\ \text{Atenção}(Q, K, V) &= \hat{D}^{-1} (Q' ((K')^\top V)) \\ \hat{D} &= \text{diag} (Q' ((K')^\top 1_L)) \end{aligned}$$

Analogamente, em um cenário estocástico

$$\begin{aligned} A &= \mathbb{E} [Q'^\top K'] \\ \widehat{\text{Atenção}}(Q, K, V) &= \hat{D}^{-1} (Q' ((K')^\top V)) \\ \hat{D} &= \text{diag} (Q' ((K')^\top 1_L)) \end{aligned}$$

Em ambas as expressões de atenção obtidas, o posicionamento dos parenteses indica a ordem de computação. Analisando a complexidade computacional das operações, chegamos a uma complexidade espacial de  $O(Lr + Ld + rd)$  e uma complexidade temporal é de  $O(Lrd)$ , onde  $d$  e  $r$  são dimensões referentes ao mapa de características  $\phi$ . Em outras palavras, um método baseado em núcleos conforme apresentado é linear no tempo com relação ao tamanho de sequência. A melhoria em complexidade temporal é ilustrada na Figura 5.8.

Figura 5.8: Complexidade computacional após aproximação via núcleos conforme ilustrado em (CHOROMANSKI et al., 2020).



O grande desafio da abordagem consiste em encontrar mapas de características viáveis. Analisaremos as escolhas de mapas de dois modelos com mecanismos de atenção baseados em núcleos: o **Linear Transformer** (em português, *Transformer Linear*) apresentado em (KATHAROPOULOS et al., 2020) e o **Performer** apresentado em (CHOROMANSKI et al., 2020).

### Abordagem de Katharopoulos

De natureza mais simples, (KATHAROPOULOS et al., 2020) busca encontrar um mapa de características para substituir  $D^{-1}A$ . Após considerarem mapas apresentados em (TSAI et al., 2019), os autores optam por um mapa de características que resulta em uma função de similaridade positiva. Mais especificamente, o mapa escolhido é

$$\phi(x) = \text{ELU}(x) + 1$$

Na expressão acima, ELU representa a *exponential linear unit* (em português, *unidade linear exponencial*) apresentada em (CLEVERT; UNTERTHINER; HOCHREITER, 2016). Considerando um hiperparâmetro  $\alpha > 0$ , a função ELU é dada por

$$f(x) = \begin{cases} x & \text{se } x > 0 \\ \alpha(\exp(x) - 1) & \text{se } x \leq 0 \end{cases}, \quad f'(x) = \begin{cases} 1 & \text{se } x > 0 \\ f(x) + \alpha & \text{se } x \leq 0 \end{cases}$$

Além de apresentar a abordagem de atenção por núcleos, o artigo considera em detalhes a viabilidade da implementação e analisa empiricamente os resultados dessa estratégia. O modelo resultante, conhecido simplesmente como **Linear Transformer**, apresenta ganhos significantes de velocidade comparado ao Reformer e ao Transformer original e obtém resultados competitivos com esses dois modelos em tarefas de referência (embora por vezes inferiores).

### Abordagem de Choromanski (FAVOR+)

Em (CHOROMANSKI et al., 2020), os autores argumentam que abordagens como as de (KATHAROPOULOS et al., 2020) e (KITAEV; KAISER; LEVSKAYA, 2020) não aproximam plenamente a operação de atenção original, mas simplesmente a substituem por operações mais tratáveis. Como resposta, Choromanski e outros propõem o mecanismo FAVOR+ (*Fast Attention via Positive Orthogonal Random features* ou *Atenção Rápida através de características Aleatórias Ortogonais*, em português).

Como objetivo, o FAVOR+ busca um mapa cujo núcleo substitua

$$\text{SM}(x, y) \stackrel{\text{def}}{=} \exp(x^\top y)$$

Observe que a expressão acima omite o termo de normalização por  $\sqrt{d}$  tipicamente encontrado na operação de atenção. Os autores argumentam que é possível equivalentemente normalizar as chaves e consultas na entrada.

De acordo com os autores, uma grande diversidade de núcleos pode ser modelada através de uma forma geral. Dadas funções  $f_1, \dots, f_l : \mathbb{R} \rightarrow \mathbb{R}$ , uma função  $h : \mathbb{R}^d \rightarrow \mathbb{R}$  e vetores determinísticos  $\omega_i$  ou  $\omega_1, \dots, \omega_m \stackrel{iid}{\sim} D$  para alguma distribuição  $D \in \mathcal{P}(\mathbb{R})^d$ , essa forma consiste em

$$\phi(x) = \frac{h(x)}{\sqrt{m}} (f_1(\omega_1^\top x), \dots, f_1(\omega_m^\top x), \dots, f_l(\omega_1^\top x), \dots, f_l(\omega_m^\top x))$$

Nesse sentido, as escolhas de quantidades, funções e distribuição acima determinam o mapa aleatório de características. Diversas propostas de mapas de características são apresentadas no artigo. Resumidamente, os autores:

1. Identificam três mapas possíveis para estimação de SM, um dos quais é impraticável.
2. Demonstram o benefício de regularização na construção dos mapas.
3. Demonstram o benefício de escolher  $\omega_i$  ortogonais.

Como resultado, Choromanski e outros apresentam o estimador  $\widehat{\text{SMREG}}_m^{\text{ort}+}$  definido por

$$\begin{aligned} h(x) &= \frac{1}{\sqrt{2}} \exp\left(-\frac{\|x\|^2}{2}\right) \\ l &= 2 \\ f_1(u) &= \exp(u) \\ f_2(u) &= \exp(-u) \\ D &= \text{Unif}\left(\sqrt{d}S^{d-1}\right) \end{aligned}$$

Observe que a distribuição  $D$  escolhida é equivalente a amostrar uniformemente numa esfera de raio  $\sqrt{d}$  em  $\mathbb{R}^d$ . Como passo adicional, os vetores amostrados de  $D$  são renormalizados através do processo de Gram-Schmidt padrão, sem prejuízos à estimação.

Resultados teórico revelam um método particularmente robusto. A abordagem não depende de premissas como esparsidade ou posto baixo de  $A$ . Além disso, os autores provam três garantias teóricas fortes:

- Estimação não-enviesada ou quase-não-enviesada de  $A$ .
- Convergência uniforme do estimador.
- Estimação de baixa variância.

Em avaliações empíricas do modelo resultante, o **Performer**, os autores demonstram velocidades e desempenhos melhores que o Reformer e o Transformer original, considerando diferentes variantes do Performer.

### 5.3 Comparação

Neste capítulo, consideramos nove propostas para mitigar o desafio da complexidade quadrática. Analisamos cada um desses modelos qualitativamente através de suas estratégias e motivações. Na tabela a seguir, constam os modelos apresentados de acordo com suas contribuições.

Tabela 5.1: Modelos com Propostas Novas para Atenção

Modelo	Artigo	Categoria	Abordagem
Transformer-XL	(DAI et al., 2019)	Atenção Esparsa	Janela de contexto com estados ocultos.
Sparse Transformer	(CHILD et al., 2019)	Atenção Esparsa	Janela de contexto e uso de elementos periódicos.
BP-Transformer	(YE et al., 2019)	Atenção Esparsa	Autoatenção de Grafo (GSA).
Longformer	(BELTAGEY; PETERS; COHAN, 2020)	Atenção Esparsa	Janela de contexto, janela dilatada e atenção global.
BigBird	(ZAHEER et al., 2020)	Atenção Esparsa	Janela de contexto, atenção global e atenção aleatória.
Linformer	(WANG; LI et al., 2020)	Alternativas para Pontuação de Similaridade	Projeções de chaves e consultas em um espaço menor.
Reformer	(KITAEV; KAISER; LEVSKAYA, 2020)	Alternativas para Pontuação de Similaridade	Hashing sensível a localidade.
Linear Transformer	(KATHAROPOULOS et al., 2020)	Alternativas para Pontuação de Similaridade	Métodos de núcleo.
Performer	(CHOROMANSKI et al., 2020)	Alternativas para Pontuação de Similaridade	Métodos de núcleo.

Comparações entre modelos são notoriamente difíceis. Tipicamente, cada artigo compara seu

modelo a poucos outros e usa sua própria escolha de tarefas de referência. Em geral, há dois critérios possíveis para avaliar um Transformer:

- Velocidade de Processamento
- Desempenho em Tarefas

Estratégias para mitigar o problema da complexidade quadrática contemplam principalmente o primeiro critério. No entanto, ambos são desejáveis. Há uma preocupação frequente na comunidade que velocidade venha ao custo de desempenho. Um modelo ideal é rápido, porém poderoso nas diferentes tarefas de PLN e aprendizado de máquina. Embora diversas tarefas de referência existam, há um argumento para uma métrica própria para avaliação de Transformers eficientes, ou seja, Transformers que mitigam o problema da complexidade quadrática.

Uma proposta é a *Long Range Arena* (em português, *Arena de Longo Alcance*, ou LRA) apresentada em (TAY et al., 2020). No Transformer original, o mecanismo de atenção é capaz de lidar com dependências distantes. Como Transformers eficientes modificam o mecanismo de atenção, tarefas que exijam essa característica representam uma métrica útil. Apropriadamente, a LRA consiste em um conjunto de seis tarefas que envolvem dependências distantes em contextos diferentes. Não entraremos em maior detalhe sobre a construção das tarefas, mas elas são:

- **Long ListOps** (em português, Operações em Listas Longas)
- **Byte-Level Text Classification** (em português, Classificação de Texto a Nível de Byte)
- **Byte-Level Document Retrieval** (em português, Recuperação de Documento a Nível de Byte)
- **Image Classification on Sequence of Pixels** (em português, Classificação de Imagem em Sequências de Pixels)
- **Pathfinder** (em português, Busca de Caminhos)
- **Pathfinder-X** (em português, Busca de Caminhos, versão extrema)

Através da LRA, (TAY et al., 2020) compara sete dos nove modelos analisados acima. Os resultados não são particularmente animadores. Nenhum modelo obtém uma vantagem clara.

Na categoria desempenho (Tabela 5.2), o Sparse Transformer alcança o primeiro posto em duas das seis tarefas – mais do que qualquer outro modelo. Reformer, Performer e Linear Transformer cada um obtém o primeiro posto em uma tarefa. Todos os modelos fracassam na tarefa Pathfinder-X. Em média, o BigBird é o modelo com o melhor desempenho.

Tabela 5.2: Desempenho de modelos em tarefas da LRA, conforme apresentado em (TAY et al., 2020).

Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Avg
Transformer	36.37	64.27	57.46	42.44	71.40	FAIL	<u>54.39</u>
Local Attention	15.82	52.98	53.39	41.46	66.63	FAIL	46.06
Sparse Trans.	17.07	63.58	<b>59.59</b>	<b>44.24</b>	71.71	FAIL	51.24
Longformer	35.63	62.85	56.89	42.22	69.71	FAIL	53.46
Linformer	35.70	53.94	52.27	38.56	<u>76.34</u>	FAIL	51.36
Reformer	<b>37.27</b>	56.10	53.40	38.07	<u>68.50</u>	FAIL	50.67
Sinkhorn Trans.	33.67	61.20	53.83	41.23	67.45	FAIL	51.39
Synthesizer	<u>36.99</u>	61.68	54.67	41.61	69.45	FAIL	52.88
BigBird	36.05	64.02	<u>59.29</u>	40.83	74.87	FAIL	<b>55.01</b>
Linear Trans.	16.13	<b>65.90</b>	53.09	42.34	75.30	FAIL	50.55
Performer	18.01	<u>65.40</u>	53.82	<u>42.77</u>	<b>77.05</b>	FAIL	51.41
Task Avg (Std)	29 (9.7)	61 (4.6)	55 (2.6)	41 (1.8)	72 (3.7)	FAIL	52 (2.4)

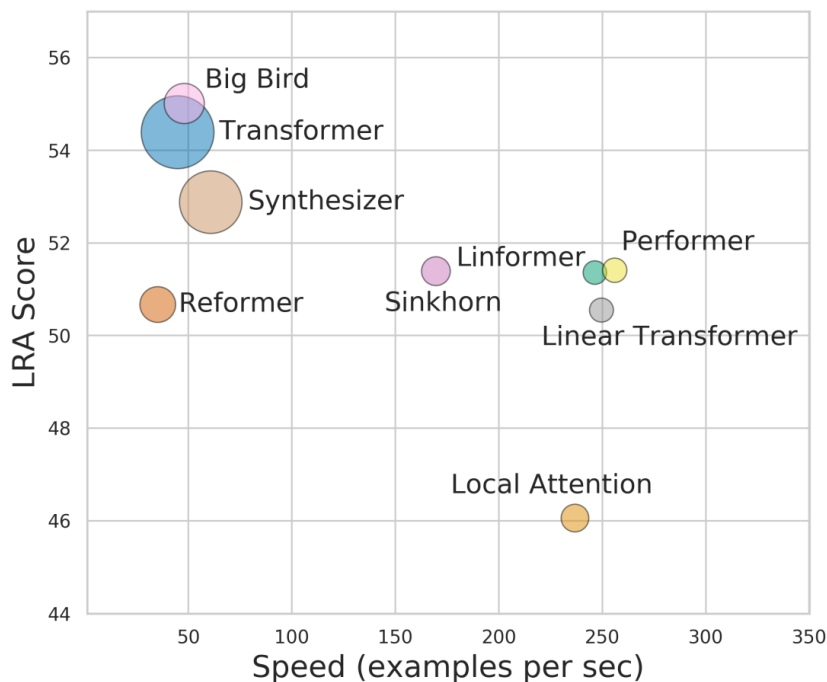
Na categoria velocidade (Tabela 5.3), o Performer toma a liderança. Em termos de consumo de memória, o Linformer é o líder. O Linear Transformer representa um meio termo entre velocidade e consumo de memória, obtendo resultados competitivos com os líderes em ambos.

Tabela 5.3: Velocidade e consumo de modelos testados via LRA, conforme apresentado em (TAY et al., 2020).

Model	Steps per second				Peak Memory Usage (GB)			
	1K	2K	3K	4K	1K	2K	3K	4K
Transformer	8.1	4.9	2.3	1.4	0.85	2.65	5.51	9.48
Local Attention	9.2 (1.1x)	8.4 (1.7x)	7.4 (3.2x)	7.4 (5.3x)	0.42	0.76	1.06	1.37
Linformer	<u>9.3</u> (1.2x)	9.1 (1.9x)	8.5 (3.7x)	7.7 (5.5x)	<b>0.37</b>	<b>0.55</b>	0.99	<b>0.99</b>
Reformer	4.4 (0.5x)	2.2 (0.4x)	1.5 (0.7x)	1.1 (0.8x)	0.48	0.99	1.53	2.28
Sinkhorn Trans	9.1 (1.1x)	7.9 (1.6x)	6.6 (2.9x)	5.3 (3.8x)	0.47	0.83	1.13	1.48
Synthesizer	8.7 (1.1x)	5.7 (1.2x)	6.6 (2.9x)	1.9 (1.4x)	0.65	1.98	4.09	6.99
BigBird	7.4 (0.9x)	3.9 (0.8x)	2.7 (1.2x)	1.5 (1.1x)	0.77	1.49	2.18	2.88
Linear Trans.	9.1 (1.1x)	<u>9.3</u> (1.9x)	<u>8.6</u> (3.7x)	<u>7.8</u> (5.6x)	<b>0.37</b>	<u>0.57</u>	<b>0.80</b>	<u>1.03</u>
Performer	<b>9.5</b> (1.2x)	<b>9.4</b> (1.9x)	<b>8.7</b> (3.8x)	<b>8.0</b> (5.7x)	<b>0.37</b>	0.59	<u>0.82</u>	1.06

Por fim, os autores identificam a relação entre pontuação LRA e velocidade (Figura 5.9). Novamente, os resultados não destacam um líder claro no geral. Na maioria dos casos, há uma troca de desempenho e velocidade. Os autores apontam métodos baseados em núcleos e matrizes de baixo posto como possivelmente o melhor custo-benefício.

Figura 5.9: Relação entre pontuação LRA e velocidade, conforme apresentado em (TAY et al., 2020).



Conforme destacado pelos autores, não é trivial conduzir uma avaliação perfeita de modelos, visto todos os fatores envolvidos. Detalhes de implementação, como ajuste de hiperparâmetros, podem ter um forte impacto e os resultados obtidos estão sujeitos a alteração no futuro.

Em suma, diversas estratégias para mitigação da complexidade quadrática existem na literatura. Até o momento, não existe uma estratégia ideal clara. Métodos baseados em atenção

esparsa vêm evoluindo ao longo dos anos. Além disso, métodos que aproximam a pontuação de similaridade via núcleos ou matrizes de baixo posto parecem promissores.

## Capítulo 6

# Redução de Modelos

Disponibilidade de memória impõe um limite rígido sobre modelos baseados em Transformers. Preocupações existem tanto com relação a memória primária (i.e. RAM) como secundária (i.e. armazenamento a longo prazo). Reduzir um modelo consiste, assim, em diminuir o seu consumo de memória a ponto que o modelo possa ser armazenado e posteriormente executado dentro dos recursos computacionais disponíveis.

Técnicas eficazes de redução resultam em modelos menores que retêm um desempenho razoável. Eventuais quedas de desempenho são admitidas contanto que não expressivas. O que constitui um desempenho razoável e uma queda expressiva varia com a aplicação. Ainda assim, veremos que proponentes de redução de modelos frequentemente almejam desempenho comparável ao modelo original.

Todas as abordagens consideradas a seguir buscam reduzir a pegada de memória do modelo, mas alcançam esse objetivo de maneiras diferentes. Três abordagens, em particular, se destacam na literatura:

- Destilação (em inglês, *distillation*)
- Poda (em inglês, *pruning*)
- Quantização (em inglês, *quantization*)

Dessas, as duas primeiras estratégias exploram uma diminuição do número de parâmetros de um modelo base. Como arquiteturas neurais são flexíveis e podem ser escalonadas, há a perspectiva de construir um modelo com poucos parâmetros a partir de um modelo base maior. Por contrapartida, a terceira abordagem busca formas mais eficazes de armazenar os parâmetros de um modelo.

Observe que as abordagens consideradas são gerais e aplicáveis a modelos baseados em redes neurais como um todo. Contudo, focaremos na aplicação dessas abordagens especificamente para modelos baseados em Transformers.

### 6.1 Destilação

Destilação consiste em treinar um modelo menor a imitar o comportamento de um modelo maior. Introduzida em (BUCILA; CARUANA; NICULESCU-MIZIL, 2006) e aplicada ao contexto de redes neurais em (HINTON; VINYALS; DEAN, 2015), a técnica envolve treinar um modelo leve (conhecido como *aluno*) em paralelo com um modelo já treinado, porém oneroso (conhecido como *professor*). Ambos os modelos são alimentados com entradas iguais, e o objetivo do aluno é replicar valores produzidos pelo professor. Em geral, esses valores envolvem a saída final do professor, mas podem também incluir valores intermediários do processamento. Tal meta é alcançada através de funções de perda conhecidas como *funções de comportamento*. Devido

à analogia de conhecimento estar sendo refinado do professor para o aluno, a abordagem é frequentemente chamada de *destilação de conhecimento* (em inglês, *knowledge distillation* ou KD). Diversos métodos de destilação existem, mas todos empregam a mesma ideia central.

Apresentada originalmente como uma técnica geral para redes neurais, a destilação hoje possui uma literatura especificamente direcionada a Transformers - em particular o BERT. Três modelos destilados propostos chamam a atenção: **DistillBERT** (SANH et al., 2019), **TinyBERT** (JIAO et al., 2020) e **MobileBERT** (SUN et al., 2020). Tais modelos representam a fronteira da tecnologia e introduzem aprimoramentos relevantes à técnica original.

### 6.1.1 Visão Geral

Todo procedimento de destilação parte da escolha de quatro requisitos básicos:

1. Modelo Professor
2. Modelo Aluno
3. Tarefa e Conjunto de Dados
4. Funções de Comportamento e Regime de Treino

Nesta seção, analisaremos as opções disponíveis para cada um desses quesitos e os resultados obtidos através da técnica.

### 6.1.2 Modelo Professor

Bons candidatos para professores geralmente surgem de modelos publicamente disponíveis de alto desempenho. Há uma observação importante, no entanto. Durante a destilação, professor e aluno são carregados simultaneamente na memória. Assim, a escolha de professor deve levar em consideração as limitações de recursos computacionais.

No artigo seminal, Hinton e outros buscam meramente exemplificar a técnica. Por isso, esses autores constroem professores próprios ao invés de reaproveitar modelos prontos. Contudo, os demais modelos considerados nessa seção tomam como professores diferentes versões do BERT. Mais especificamente, o **TinyBERT** e o **DistillBERT** usam o BERT<sub>Base</sub>, enquanto o **MobileBERT** opta pelo BERT<sub>Large</sub>. Nesse sentido, a destilação oferece uma oportunidade de criar versões menores de modelos elaborados com grande poder computacional.

### 6.1.3 Modelo Aluno

Modelos aluno e professor não precisam compartilhar a mesma arquitetura. Contudo, escolhas semelhantes de arquiteturas surgem no artigo seminal e são prevalentes na literatura. Os três modelos destilados considerados nesta análise buscam não só uma dupla aluno-professor do tipo Transformer, mas também alunos e professores arquiteturalmente semelhantes.

Os criadores do **TinyBERT** propõem construir alunos através de uma seleção limitada de camadas do professor. Dentro dessa perspectiva, dado um aluno composto por  $M$  blocos de Transformer e um professor composto por  $N$  blocos de Transformer, com  $M < N$ , é necessário definir uma função  $g$  que mapeie cada bloco do aluno a um bloco do professor, isto é,

$$n = g(m)$$

Os autores ainda impõem condições necessárias para garantir que aluno e professor aceitem as mesmas entradas e forneçam saídas do mesmo tipo. Considerando a camada de *embedding* como a camada de índice 0 e a camada de predição como a camada de índice  $M + 1$ , é essencial que

$$\begin{aligned} 0 &= g(0) \\ N + 1 &= g(M + 1) \end{aligned}$$



As possíveis escolhas de  $g$  são consideradas no artigo. No fim, o TinyBERT opta por

$$g(m)_{\text{TinyBERT}} = 3 \times m$$

Ou seja, o TinyBERT seleciona uma a cada três camadas do modelo professor.

Embora anterior à formalização acima, o **DistillBERT** segue uma linha de raciocínio semelhante. Sanh e outros optam por tomar uma a cada duas camadas do modelo professor, fazendo isso alternadamente. Assim, temos

$$g(m)_{\text{DistillBERT}} = 2 \times m$$

Além disso, os autores removem as embeddings de tipo *token* e poolers.

Outros modelos divergem da estratégia de seleção de camadas. O **MobileBERT** é projetado como uma versão mais “estreita” do BERT<sub>Large</sub>. Em termos práticos, isso significa ter um aluno com a mesma quantidade de blocos do que o professor, mas uma dimensão oculta menor em cada bloco. Através de uma técnica conhecida como gargalo invertido (em inglês, *inverted bottleneck*), os autores garantem que ainda assim as dimensões de entrada e saída dos bloco de aluno e professor sejam as mesmas. Além disso, Sun e outros propõem certas mudanças aos blocos do aluno. Após uma busca complexa de arquitetura, os autores reduzem o número de cabeças de atenção para 4 e aumentam a quantidade de camadas densas também para 4. Outras otimizações operacionais incluem pequenos ajustes, como a troca das funções de ativação e remoção de camadas de normalização. Por fim, os autores reconhecem que encontrar as configurações exatas do modelo envolveu uma busca razoavelmente profunda por hiperparâmetros.

#### 6.1.4 Tarefa e Conjunto de Dados

Como no caso de outros Transformers, o modelo aluno pode ser treinado com uma variedade de tarefas. No entanto, para propósitos de destilação, uma tarefa específica deve ser escolhida. A tarefa, por sua vez, determina o conjunto de dados a ser usado.

No artigo seminal, Hinton e outros optam por uma tarefa de classificação. Notavelmente, em destilação, a escolha de uma tarefa de aprendizado supervisionado não necessariamente exige um conjunto de dados rotulados. Como o aluno busca apenas replicar o comportamento professor, destilação pode ser feita mesmo sem os rótulos verdadeiros. Basta que o aluno produza valores similares ao professor. Ainda assim, um conjunto de dados rotulados pode ser útil caso os desenvolvedores tenham interesse em realizar uma destilação híbrida, como é frequentemente o caso.

Inspirados pelo BERT, **DistillBERT**, **TinyBERT** e **MobileBERT** são todos treinados com uma tarefa de modelagem de linguagem, embora detalhes do treino difiram entre modelos.

#### 6.1.5 Funções de Comportamento

Encontrar funções de comportamento que resultem em um aluno capaz representa um dos grandes desafios de destilação. Conforme considerado, funções de comportamento buscam encorajar o aluno a produzir valores semelhantes ao professor. Em geral, alunos são treinados com uma combinação de funções de comportamento e funções de perda adicionais que treinam o aluno diretamente na tarefa.

Nessa subseção, consideraremos a proposta original de destilação e as abordagens seguidas por diversos Transformers destilados.

#### Abordagem Original

Na proposta original, Hinton e outros sugerem modificar os modelos aluno e professor para realizar a destilação. Os autores introduzem uma variante da função softmax conhecida como

*softmax-temperatura*.

$$\text{SoftmaxTemperatura}(z_i) = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}}$$

O objetivo dessa função é suavizar as saídas de um modelo classificador. O termo  $T$  é conhecido como a *temperatura* da função. Quanto maior for o termo de temperatura, mais suave será a distribuição de probabilidades produzida pelo modelo ao longo das classes. Quando  $T = 1$ , obtemos a forma canônica da função softmax.

Com esse arcabouço, os autores introduzem duas funções de perda:

1. **Perda de Destilação:** Denotada por  $\mathcal{L}_D$ , consiste na entropia cruzada das saídas do professor e do aluno, com ambos operando em alta temperatura. A temperatura deve ser alta o suficiente que a saída do professor seja suave. Observe que a  $\mathcal{L}_D$  constitui uma função de comportamento uma vez que relaciona o aluno ao professor.
2. **Perda do Aluno:** Denotada por  $\mathcal{L}_S$ , consiste na entropia cruzada do aluno, operando sob temperatura 1, com os rótulos verdadeiros.

Como objetivo geral, Hinton e outros propõem uma média ponderada das duas funções de perda acima.

$$\mathcal{L} = \alpha \mathcal{L}_D + \beta \mathcal{L}_S$$

Segundo os autores, melhores resultados são obtidos quando  $\alpha$  é bem maior do que  $\beta$ . Como observação final, Hinton e outros indicam que, após a destilação, o modelo aluno deve ser usado com temperatura 1.

## DistillBERT

Pioneiro nas aplicações de destilação a Transformers, o DistillBERT propõem como objetivo uma combinação linear de três funções de perda.

1. **Perda de Destilação:** Construída nos moldes de Hinton e outros.
2. **Perda de Cossenos de Embeddings:** Denotada por  $\mathcal{L}_{cos}$ , consiste em uma perda de similaridade de cossenos das camadas de embedding de aluno e professor, respectivamente, em cada bloco. Representa a segunda função de comportamento do modelo.
3. **Perda de Modelagem de Linguagem Mascarada:** Denotada por  $\mathcal{L}_{MLM}$ , consiste no objetivo MLM do BERT original.

Com relação ao regime de treino, os criadores do DistillBERT seguem as boas práticas para treinamento de RoBERTa, conforme estipuladas em (LIU et al., 2019).

## TinyBERT

Os criadores do TinyBERT consideram o objetivo do modelo por bloco. Conforme considerado, cada bloco do aluno TinyBERT é associado a um bloco do professor BERT. Assim, os autores propõem quatro funções de comportamento.

1. **Perda de Destilação:** Construída nos moldes de Hinton e outros, embora os autores observem que utilizar uma temperatura 1 para essa perda traz bons resultados.

2. **Perda de Atenção (Variante MSE):** Definida por

$$\mathcal{L}_{\text{atnMSE}} = \frac{1}{h} \sum_{i=1}^h \text{MSE}(A_i^S, A_i^T)$$

onde  $h$  representa o número de cabeças de atenção no bloco,  $A_i$  indica a  $i$ -ésima matriz de atenção,  $S$  sobrescrito relaciona a matriz ao aluno e  $T$  sobrescrito relaciona a matriz ao professor. A função incentiva o aluno a produzir mecanismos de atenção semelhantes ao professor.

3. **Perda de Estado Oculto:** Definida por

$$\mathcal{L}_{\text{ocult}} = \text{MSE}(H^S W_h, H^T)$$

onde  $H^S$  e  $H^T$  são os estados ocultos de aluno e professor, respectivamente. A matriz  $W_h$  representa uma transformação linear aprendida para levar o estado oculto do aluno ao mesmo espaço que o do professor.

4. **Perda de Destilação de Embedding:** Definida por

$$\mathcal{L}_{\text{embd}} = \text{MSE}(E^S W_e, E^T)$$

onde  $E^S$  e  $E^T$  são as camadas de embedding do aluno e do professor, respectivamente. A matriz  $W_e$  desempenha um papel análogo à matriz  $W_h$  previamente apresentada.

Com isso, os autores definem uma perda geral para cada bloco.

$$\mathcal{L}_{\text{bloco}} = \begin{cases} \mathcal{L}_{\text{embd}}, & m = 0 \\ \mathcal{L}_{\text{ocult}} + \mathcal{L}_{\text{atnMSE}}, & M \leq m < 0 \\ \mathcal{L}_D, & m = M + 1 \end{cases}$$

Portanto, a cada exemplo, o objetivo do modelo é minimizar

$$\mathcal{L}_{\text{modelo}} = \sum_{m=0}^{M+1} \lambda_m \mathcal{L}_{\text{bloco}}^{(m)}$$

onde  $(m)$  sobrescrito indica o índice da bloco e  $\lambda_m$  é um hiperparâmetro que indica a importância da bloco.

Além do conjunto de funções de perda descritos acima, o TinyBERT separa o processo de destilação em duas fases. Na primeira fase, o aluno é treinado em um processo de pré-treino junto a um professor não-ajustado. Em seguida, o aluno é treinado em uma tarefa específica junto com um professor ajustado para essa tarefa. O procedimento em duas fases é destacado como de grande importância pelos autores.

## MobileBERT

O MobileBERT segue uma abordagem de destilação um pouco mais complexa. Dois objetivos guiam o treino do aluno: um objetivo de destilação geral, definido a nível de bloco, e um objetivo específico para a fase de pré-treino. Como objetivo de destilação geral, o aluno usa uma combinação linear de duas funções de perda.

1. **Perda de Estado Oculto:** Construída de forma semelhante ao TinyBERT, com a diferença que a matriz  $W_h$  é desnecessária, uma vez que os estados ocultos de aluno e professor estão sempre no mesmo espaço, por construção.

2. **Perda de Atenção (Variante  $D_{KL}$ ):** Denotada por  $\mathcal{L}_{KL}$ , consiste na *divergência de Kullback-Leibler* (denotada por  $D_{KL}$ ) calculada entre as matrizes de atenção do aluno e do professor, respectivamente. Como a perda de atenção do TinyBERT, esta função de comportamento busca incentivar o aluno a produzir mecanismos de atenção semelhantes ao professor.

Por sua vez, para o objetivo de pré-treino, os autores propõem

$$\mathcal{L}_{PD} = \alpha \mathcal{L}_{MLM} + (1 - \alpha) \mathcal{L}_{KD} + \mathcal{L}_{NSP}$$

onde  $\mathcal{L}_{NSP}$  é a perda de *predição de próxima frase* (em inglês, *next sentence prediction* ou NSP) do BERT original com  $\alpha \in (0,1)$  hiperparâmetro.

Em termos de regime de treino, Sun e outros consideram diversas estratégias no artigo. Ao final, os autores concluem que a estratégia de melhor desempenho é a chamada *transferência progressiva de conhecimento* (em inglês, *progressive knowledge transfer* ou PKT), que consiste em treinar um bloco do aluno por vez, começando pelos blocos iniciais. Durante o treinamento de um bloco, os parâmetros de blocos anteriores são fixados ou treinados com uma taxa de aprendizado baixa. Blocos superiores aguardam a iteração apropriada para serem treinados. Dessa maneira, o processo de treino consiste em  $L$  fases, onde  $L$  é a quantidade de blocos. Note que apenas ao final o objetivo de pré-treino é de fato usado.

### 6.1.6 Resultados e Desempenho

Proponentes de destilação defendem que redes neurais destiladas demonstram desempenho superior a redes de mesmo tamanho treinadas diretamente. Sem dúvida, os resultados de aplicações de destilação a Transformers são animadores.

Dos modelos considerados nesta seção, todos registram uma redução significativa de parâmetros. Notavelmente, o **DistillBERT** obtém uma redução de 40% no total de parâmetros, enquanto **TinyBERT** é 7.5x menor do que o modelo professor e **MobileBERT** é 4.3x menor.

Com relação ao desempenho em tarefas de PLN, os modelos registram poucas perdas. O **DistillBERT** retém 97% do desempenho em compreensão de linguagem relativo ao modelo professor, avaliado através do conjunto de tarefas GLUE. Por sua vez, o **TinyBERT** alcança 96.8% do desempenho do BERT<sub>Base</sub> na mesma avaliação. Surpreendentemente, um variante do **MobileBERT** no artigo original registra um desempenho *melhor* que o BERT<sub>Base</sub> nas avaliações SQuAD v1.1, SQuAD v2.0 e GLUE. Embora o **MobileBERT** canônico e uma versão miniatura dele não alcancem o mesmo feito, esses modelos ainda obtêm resultados altamente competitivos com BERT<sub>Base</sub>.

Por sua vez, comparações entre modelos destilados são complexas. Diferenças entre professores significam que nem sempre uma comparação é justa. Além disso, os modelos de referência apresentados nem sempre refletem o professor utilizado. E muito embora conjuntos de tarefas como GLUE sejam prevalentes para avaliação, determinadas métricas podem divergir. Portanto, não entraremos em maiores detalhes sobre comparações de técnicas. Ainda assim, fica evidente que há um imenso potencial para técnicas de destilação no desenvolvimento de Transformers.

## 6.2 Poda

Podar consiste em eliminar diretamente parâmetros do modelo. Diversas heurísticas existem para determinar quais parâmetros devem ser removidos de uma rede neural. (BLALOCK et al., 2020), uma meta-análise de 81 artigos, fundamenta muito do que se entende sobre o estado atual de técnicas de poda em redes neurais artificiais.

### 6.2.1 Procedimento Geral

Como algoritmo geral de poda, Blalock e outros introduzem os seguintes passos:

1. Treinar uma rede neural até convergência.
2. Atribuir uma pontuação a cada parâmetro.
3. Eliminar parâmetros de acordo com a sua pontuação.
4. Ajustar (em inglês, *fine-tune*) a rede neural remanescente.

A abrangência do algoritmo acima reflete a diversidade de heurísticas de pontuação e ajuste. Segundo os autores da meta-análise, há quatro principais quesitos nos quais os algoritmos de poda geralmente divergem.

- **Estrutura:** Parâmetros são removidos individualmente ou em grupo?
- **Pontuação:** A pontuação é absoluta em toda a rede, local ou relativa a algum grupo específico de parâmetros?
- **Agendamento:** Os parâmetros são todos removidos em um passo ou iterativamente? A fração de parâmetros que é removida a cada iteração varia?
- **Ajuste:** A rede é ajustada com os parâmetros já treinados ou é completamente reiniciada para o ajuste?

De acordo com Blalock e outros, o típico algoritmo de poda usa tanto poda como ajuste iterativos.

### 6.2.2 Literatura de Poda

Os autores da meta-análise observam que, no geral, técnicas de poda reduzem substancialmente o número de parâmetros de uma rede neural com mínima redução de precisão. Em determinados casos, redes podadas podem até mesmo atingir uma precisão maior. Além disso, diversas técnicas “espertas” de poda superam eliminação aleatória de parâmetros, pelo menos quando a quantidade de parâmetros eliminados é suficientemente alta. No entanto, comparações de técnicas são difíceis devido a metodologias inconsistentes e falta de padronização na literatura. Blalock e outros constataam que dos artigos analisados:

- 25% dos artigos não comparam seu método a qualquer outro método.
- 50% dos artigos comparam seu método a no máximo um outro método.
- Dezenas de métodos propostos nunca serviram de comparação para trabalhos futuros.
- Vários artigos redescobrem métodos que já haviam sido publicados, particularmente antes de 2010.
- Em um terço dos artigos, nenhum par de dados-rede neural consta.
- Nomenclaturas, métricas de avaliação e metodologias de experimentação divergem.
- Hiperparâmetros e outros fatores de confusão variam ou sequer são especificados.
- Nenhum artigo contundentemente controla todos os fatores de confusão.

Sob tais circunstâncias, é impossível identificar uma técnica que represente o estado da arte em poda. Contudo, Blalock e outros também introduzem em sua meta-análise a ShrinkBench, uma biblioteca de código aberto para comparação padronizada de técnicas de poda. Contidas nas ferramentas de avaliação dessa biblioteca estão cinco estratégias básicas de poda que servem para efeitos comparativos:

- **Poda por Magnitude Global:** Elimina os parâmetros com o menor valor absoluto em toda a rede.
- **Poda por Magnitude por Camada:** Elimina os parâmetros com o menor valor absoluto em cada camada.
- **Poda por Magnitude de Gradiente Global:** Elimina parâmetros cujo produto de peso por gradiente, avaliado num lote de entradas, tem o menor valor absoluto em toda a rede.
- **Poda por Magnitude de Gradiente por Camada:** Elimina parâmetros cujo produto de peso por gradiente, avaliado num lote de entradas, tem o menor valor absoluto em cada camada.
- **Poda Aleatória:** Elimina cada parâmetro independentemente com probabilidade igual à fração de parâmetros a serem eliminados.

Nessa sentido, as opções acima representam estratégias razoável de poda. Como observação, é importante ressaltar que o processo de poda resulta em redes neurais esparsas. Portanto, bibliotecas ou hardware especializados são importantes para garantir uma execução eficiente. No entanto, a técnica possui tremendo potencial na redução de modelos.

## 6.3 Quantização

Quantização consiste em armazenar parâmetros de um modelo em um espaço reduzido de memória. A prática está presente em diversos ramos de modelos matemáticos. Devido à temática de Transformers, concentraremos nesta seção nas diferentes técnicas de quantização para camadas densas de modelos neurais. Parâmetros dessas camadas são responsáveis por uma parcela considerável da pegada de memória de Transformers. Portanto, técnicas que amenizem o consumo dessas camadas são de grande interesse.

### 6.3.1 Quantização Escalar

Tipagem eficiente de parâmetros representa a primeira abordagem para quantização. Bibliotecas populares de aprendizado profundo tipicamente armazenam seus parâmetros em representação de ponto flutuante de 32 (ou, até mesmo, 64) bits. Trabalhos como (HUBARA et al., 2017) e (JACOB et al., 2018) propõem utilizar tipos com menor consumo de memória. Por tratar parâmetros de forma individual, a abordagem é conhecida como *quantização escalar*.

Uma proposta consiste em usar inteiros como forma aproximada de expressar números reais. Ainda assim, até mesmo uma representação em ponto flutuante de precisão reduzida pode ser vantajosa. Outra opção é a binarização, técnica que consiste em expressar cada parâmetro em um bit representando  $\{-1, +1\}$ . Na literatura, algumas otimizações são também introduzidas para minimizar eventuais perdas de desempenho e adequar a técnica a diferentes contextos. Separação entre quantização durante a fase de treino e quantização pós-treino representa ainda uma discussão adicional. Não obstante, a busca por métodos mais eficientes motiva outras abordagens de quantização que consideraremos a seguir.

### 6.3.2 Quantização Vetorial

Grande parte da literatura de quantização busca métodos mais elaborados que reduzam o espaço de armazenamento do modelo sem significativo redução no desempenho. Dessses esforços, nasce a abordagem de *quantização vetorial*. A estratégia aproveita a estrutura inerente dos modelos neurais para realizar uma quantização mais eficiente do que uma abordagem individualizada nos parâmetros. Em (GONG et al., 2014), os autores expõem algumas técnicas voltadas a vetores como:

- **Quantização Escalar via Agrupamento *K-Means*:** Agrega os valores dos parâmetros em diversos grupos e armazena apenas o valor médio de cada grupo.
- **Quantização por Produto:** O espaço vetorial é particionado em diversos subespaços disjuntos e cada subespaço é quantizado via agrupamento K-Means
- **Quantização Residual:** Quantiza os vetores em  $k$  centros via agrupamento K-Means e daí recursivamente quantiza os residuais via agrupamento K-Means.

Uma inspiração para os métodos apresentados é o trabalho de (DENIL et al., 2013). De fato, os autores obtêm resultados semelhantes a esse artigo anterior. Demais métodos foram encontrados, mas considerados inadequados. Além disso, os autores observam que métodos de fatoração de matrizes usando decomposição por valores singular, usados para estruturas convolucionais, são ineficazes para camadas densas. Ao final, Gong e outros encontraram bons resultados com a abordagem de quantização escalar via agrupamento K-means. Por fim, os autores observam que o principal motivo para quantização vetorial é a redução da pegada de memória ao invés de tempo de inferência, uma vez que as operações dos métodos podem não ser eficientemente executadas no hardware.

## 6.4 Outros Métodos

Determinados métodos de redução no consumo de memória não se encaixam em nenhuma das categorias acima. As camadas reversíveis, apresentadas em (GOMEZ et al., 2017), representam um tal caso. Inspiradas pelas redes residuais, camadas reversíveis livram um modelo neural da necessidade de armazenar todas as ativações de suas camadas em memória ao mesmo tempo. Isso é feito através de um mecanismo que permite reconstruir a ativação de uma camada a partir da camada seguinte. Originária de um contexto de visão computacional, a técnica pode também ser aplicada a Transformers.

Como base, os autores introduzem a noção de um bloco reversível de operações. Note que um bloco reversível não deve ser imediatamente confundido com um bloco de Transformer, trata-se apenas de uma abstração para modelos neurais. Cada bloco reversível recebe entradas  $(x_1, x_2)$  e produz saídas  $(y_1, y_2)$ . A propagação pelo bloco consiste em

$$\begin{aligned} y_1 &= x_1 + \mathcal{F}(x_2) \\ y_2 &= x_2 + \mathcal{G}(y_1) \end{aligned}$$

Tal construção permite recuperar as ativações de uma camada a partir das ativações da próxima camada. Basta tomar

$$\begin{aligned} x_2 &= y_2 - \mathcal{G}(y_1) \\ x_1 &= y_1 - \mathcal{F}(x_2) \end{aligned}$$

Note que a escolha de funções  $\mathcal{F}$  e  $\mathcal{G}$  depende da aplicação. Os criadores do Reformer em (KITAEV; KAISER; LEVSKAYA, 2020) propõem a seguinte implementação para Transformers.

$$\begin{aligned} y_1 &= x_1 + \text{Atenção}(x_2) \\ y_2 &= x_2 + \text{FFN}(y_1) \end{aligned}$$

Notavelmente, a implementação acima alinha o conceito de bloco reversível com bloco de Transformer. Não entraremos em maiores detalhes sobre a derivação da técnica e implementação. No entanto, note que a técnica resulta em uma grande redução no consumo de memória e, ao menos no caso do Reformer, alcança tal resultado sem maiores prejuízos em custo de computação. Ambos artigos citados para essa técnica também constatam que a abordagem não traz prejuízos ao desempenho do modelo neural. Por fim, Kitaev e outros destacam a contribuição da técnica para construir um Transformer mais eficiente.

## 6.5 Justificativa e Teoria

No desenvolvimento de técnicas de redução de modelos, há uma preocupação prevalente que modelos menores talvez não possuam a mesma expressividade de modelos maiores. Desdobramentos recentes na teoria lançam algum luz sobre esse tópico e fornecem até mesmo possíveis justificativas ou explicações. Em contraponto à preocupação mencionada, há uma suspeita difundida na comunidade que o típico modelo é superparametrizado, ou seja, contém muito mais parâmetros do que de fato precisa. Tão popular é essa ideia que (FRANKLE; CARBIN, 2019) apresenta a Hipótese de Bilhete de Loteria.

**(Hipótese de Bilhete de Loteria).** *Uma rede neural densa, aleatoriamente inicializada contém uma subrede que foi inicializada de tal modo que — quando treinada isoladamente — é capaz de alcançar a precisão de teste da rede neural original após treinar por, no máximo, uma mesma quantidade de iterações.*

O argumento de Frankle e Carbin fica mais claro quando se observa a construção dos experimentos empíricos dos autores. Primeiro, uma rede neural densa é aleatoriamente inicializada. Em seguida, essa rede é treinada em uma determinada tarefa por  $N$  iterações e atinge um desempenho  $P$ . Através de uma heurística, os autores então eliminam parâmetros da rede e a treinam novamente. O processo é repetido por múltiplas iterações até que a quantidade de parâmetros tenha sido significantemente reduzida. Por fim, os autores retornam todos os parâmetros remanescentes para seus valores originais, quando a rede original foi inicializada. Frankle e Carbin estabelecem empiricamente que, para centenas de redes de arquiteturas e configurações diferentes, a rede composta pelos parâmetros remanescentes, ao ser treinada, atinge o desempenho  $P$  em no máximo  $N$  iterações. Tais redes são conhecidas como *bilhetes vencedores* e geralmente tem 10-20% do tamanho da rede original. Esses resultados apoiam a tese da prevalência de redes superparametrizadas.

Outras observações curiosas ainda surgem desse artigo. Segundo os autores, a mera estrutura de um bilhete vencedor não explica por si o seu sucesso. A inicialização específica do bilhete parece desempenhar um papel. Quando um bilhete vencedor é reinicializado com valores diferentes da rede original, seu desempenho é consideravelmente pior. Frankle e Carbin ainda descartam a tese que os parâmetros do bilhete vencedor foram inicializados próximos de seu valores finais após o treino. De fato, os autores constataam que os parâmetros de bilhetes vencedores geralmente mudam mais que outros parâmetros da rede original. Os resultados levam os autores a conjecturar que talvez o método do gradiente estocástico encontre um conjunto pequeno de parâmetros bem inicializados e foque o treino neles. Nesse sentido, uma rede superparametrizada poderia ser mais facilmente treinada por possuir uma maior disponibilidade de parâmetros.

Além da Hipótese de Bilhete de Loteria, a análise em (KOVALEVA et al., 2019) apresenta evidências de superparametrização no modelo BERT, que parecem ser confirmadas pelo sucesso dos modelos destilados previamente apresentados. Adicionando a essa análise, (ROGERS; KOVALEVA; RUMSHISKY, 2020) argumenta que o número elevado de cabeças do BERT traz uma contribuição limitada ao modelo.

Embora ainda haja muito a se descobrir sobre Transformers, a perspectiva de redução de modelos parece ter grande potencial na criação de Transformers mais eficientes.



## Capítulo 7

# Conclusões

Desde a sua introdução em 2017, Transformers têm mudado o mundo de PLN. Neste trabalho, exploramos as diferentes facetas dessa arquitetura inovadora, de seu início aos dias de hoje.

Com primórdios em conceitos de processamento de linguagem natural e aprendizado de máquina, Transformers representam uma união de diversas ideias. Entre essas, redes neurais e mecanismos de atenção formam o cerne de tais modelos. Pouco após a introdução, implementações iniciais demonstraram o grande potencial da arquitetura, além de introduzirem diversos aprimoramentos relevantes. No entanto, Transformers possuem também limitações. Problemas de complexidade computacional e número elevado de parâmetros representam desafios significantes para a arquitetura.

Diante de um cenário de preocupação com a viabilidade futura dos Transformers, este trabalho mostra que soluções existem. Alternativas para o mecanismo de atenção original estão disponíveis na literatura e continuam a ser aprimoradas. Outra perspectiva é a de redução de modelos, composta por um conglomerado de técnicas como: destilação, poda e quantização.

Na busca por Transformers eficientes, as técnicas consideradas neste trabalho são efetivas e poderosas. Resultados de destaque em avaliações evidenciam o potencial para aplicação em modelos. Determinados mecanismos de atenção alternativos, como o do Performer, admitem até mesmo a substituição direta em modelos já treinados, com mínimo ajuste posterior necessário. Adicionalmente, as diferentes classes de técnicas consideradas (i.e. atenção alternativa, destilação, quantização, poda, etc.) são ortogonais umas às outras, ou seja, podem ser implementadas em conjunto. O MobileBERT, por exemplo, é construído primeiro destilando um modelo e depois quantizando o resultado, muito embora as principais contribuições do modelo sejam relacionadas a destilação. No horizonte próximo, combinações de técnicas representam uma trajetória possível para Transformers.

Tudo considerado, o progresso dos Transformers não parece ter chegado ao seu limite. Novas propostas continuam a surgir, enquanto outras antigas são reinventadas. Diante de tantas novidades, uma afirmação é certa – a pesquisa atual aponta para um futuro animador para o PLN neural.

# Referências

- ANDONI, Alexandr et al. Practical and Optimal LSH for Angular Distance. In: NIPS. [S.l.: s.n.], 2015.
- BELTAGY, Iz; PETERS, Matthew E.; COHAN, Arman. Longformer: The Long-Document Transformer. **ArXiv**, abs/2004.05150, 2020.
- BENGIO, Yoshua; DUCHARME, Réjean; VINCENT, Pascal. A Neural Probabilistic Language Model. In: \_\_\_\_\_. **Advances in Neural Information Processing Systems**. [S.l.]: MIT Press, 2001. v. 13. Disponível em: <<https://proceedings.neurips.cc/paper/2000/file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf>>.
- BENGIO, Yoshua; DUCHARME, Réjean; VINCENT, Pascal; JANVIN, Christian. A Neural Probabilistic Language Model. **J. Mach. Learn. Res.**, JMLR.org, v. 3, null, p. 1137–1155, mar. 2003. ISSN 1532-4435.
- BISHOP, Chistopher M. **Pattern Recognition and Machine Learning**. [S.l.]: Springer, 2006.
- BLALOCK, Davis W. et al. What is the State of Neural Network Pruning? **ArXiv**, abs/2003.03033, 2020.
- BOWMAN, Samuel R. et al. A large annotated corpus for learning natural language inference. In: PROCEEDINGS of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP). [S.l.]: Association for Computational Linguistics, 2015.
- BROWN, Tom et al. Language Models are Few-Shot Learners. In: \_\_\_\_\_. **Advances in Neural Information Processing Systems**. [S.l.]: Curran Associates, Inc., 2020. v. 33, p. 1877–1901. Disponível em: <<https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf>>.
- BUCILA, Cristian; CARUANA, R.; NICULESCU-MIZIL, Alexandru. Model compression. In: KDD '06. [S.l.: s.n.], 2006.
- CHEN, T. et al. Training Deep Nets with Sublinear Memory Cost. **ArXiv**, abs/1604.06174, 2016.
- CHILD, R. et al. Generating Long Sequences with Sparse Transformers. **ArXiv**, abs/1904.10509, 2019.
- CHO, Kyunghyun et al. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In: PROCEEDINGS of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation. Doha, Qatar: Association for Computational Linguistics, out. 2014. p. 103–111. DOI: [10.3115/v1/W14-4012](https://www.aclweb.org/anthology/W14-4012). Disponível em: <<https://www.aclweb.org/anthology/W14-4012>>.
- CHOROMANSKI, Krzysztof et al. Rethinking Attention with Performers. **ArXiv**, abs/2009.14794, 2020.

- CLEVERT, Djork-Arné; UNTERTHINER, Thomas; HOCHREITER, S. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). **arXiv: Learning**, 2016.
- DAI, Zihang et al. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. In: ACL. [S.l.: s.n.], 2019.
- DENIL, Misha et al. Predicting Parameters in Deep Learning. In: NIPS. [S.l.: s.n.], 2013.
- DEVLIN, J. et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: NAACL-HLT. [S.l.: s.n.], 2019.
- DOLAN, William B.; BROCKETT, Chris. Automatically Constructing a Corpus of Sentential Paraphrases. In: PROCEEDINGS of the Third International Workshop on Paraphrasing (IWP2005). [S.l.: s.n.], 2005. Disponível em: <https://www.aclweb.org/anthology/I05-5002>.
- FRANKLE, Jonathan; CARBIN, Michael. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. **arXiv: Learning**, 2019.
- GOMEZ, Aidan N. et al. The Reversible Residual Network: Backpropagation Without Storing Activations. In: NIPS. [S.l.: s.n.], 2017.
- GONG, Yunchao et al. Compressing Deep Convolutional Networks using Vector Quantization. **ArXiv**, abs/1412.6115, 2014.
- GRAVES, A. Generating Sequences With Recurrent Neural Networks. **ArXiv**, abs/1308.0850, 2013.
- GRUSLYS, Audrunas et al. Memory-Efficient Backpropagation Through Time. In: \_\_\_\_\_. **Advances in Neural Information Processing Systems**. [S.l.]: Curran Associates, Inc., 2016. v. 29. Disponível em: <https://proceedings.neurips.cc/paper/2016/file/a501bebf79d570651ff601788ea9d16d-Paper.pdf>.
- HE, Kaiming et al. Deep Residual Learning for Image Recognition. **2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, p. 770–778, 2016.
- HENDRYCKS, Dan; GIMPEL, Kevin. Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units. **ArXiv**, abs/1606.08415, 2016.
- HINTON, Geoffrey E.; VINYALS, Oriol; DEAN, J. Distilling the Knowledge in a Neural Network. **ArXiv**, abs/1503.02531, 2015.
- HOCHREITER, Sepp; SCHMIDHUBER, Jürgen. Long Short-Term Memory. **Neural Computation**, v. 9, n. 8, p. 1735–1780, nov. 1997. ISSN 0899-7667. DOI: [10.1162/neco.1997.9.8.1735](https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf). eprint: <https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf>. Disponível em: <https://doi.org/10.1162/neco.1997.9.8.1735>.
- HOWARD, J.; RUDER, Sebastian. Universal Language Model Fine-tuning for Text Classification. In: ACL. [S.l.: s.n.], 2018.
- HUBARA, Itay et al. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. **ArXiv**, abs/1609.07061, 2017.
- ILYER, Shankar; DANDEKAR, Nikhil; CSERNAI, Kornél. **First Quora Dataset Release: Question Pairs**. [S.l.: s.n.], 2017. Disponível em: <https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs>.
- IOFFE, Sergey; SZEGEDY, Christian. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. **ArXiv**, abs/1502.03167, 2015.
- JACOB, Benoit et al. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. **2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition**, p. 2704–2713, 2018.

- JAMES, Gareth et al. **An Introduction to Statistical Learning**: with Applications in R. [S.l.]: Springer, 2013.
- JIAO, Xiaoqi et al. TinyBERT: Distilling BERT for Natural Language Understanding. **ArXiv**, abs/1909.10351, 2020.
- JOSHI, Mandar et al. TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension. In: ACL. [S.l.: s.n.], 2017.
- KATHAROPOULOS, Angelos et al. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention. **ArXiv**, abs/2006.16236, 2020.
- KITAEV, Nikita; KAISER, Lukasz; LEVSKAYA, Anselm. Reformer: The Efficient Transformer. **ArXiv**, abs/2001.04451, 2020.
- KOVALEVA, O. et al. Revealing the Dark Secrets of BERT. In: EMNLP/IJCNLP. [S.l.: s.n.], 2019.
- LAI, Guokun et al. RACE: Large-scale ReAding Comprehension Dataset From Examinations. In: PROCEEDINGS of the 2017 Conference on Empirical Methods in Natural Language Processing. Copenhagen, Denmark: Association for Computational Linguistics, set. 2017. p. 785–794. DOI: [10.18653/v1/D17-1082](https://doi.org/10.18653/v1/D17-1082). Disponível em: <https://www.aclweb.org/anthology/D17-1082>.
- LIU, Y. et al. RoBERTa: A Robustly Optimized BERT Pretraining Approach. **ArXiv**, abs/1907.11692, 2019.
- MARCUS, M.; SANTORINI, Beatrice; MARCINKIEWICZ, Mary Ann. Building a Large Annotated Corpus of English: The Penn Treebank. **Comput. Linguistics**, v. 19, p. 313–330, 1993.
- MICIKEVICIUS, P. et al. Mixed Precision Training. **ArXiv**, abs/1710.03740, 2018.
- MIKOLOV, Tomas; CHEN, Kai et al. **Efficient Estimation of Word Representations in Vector Space**. [S.l.: s.n.], 2013. Disponível em: <http://arxiv.org/abs/1301.3781>.
- MIKOLOV, Tomas; KARAFIÁT, M. et al. Recurrent neural network based language model. In: INTERSPEECH. [S.l.: s.n.], 2010.
- MIKOLOV, Tomas; SUTSKEVER, Ilya et al. Distributed Representations of Words and Phrases and their Compositionality. In: \_\_\_\_\_. **Advances in Neural Information Processing Systems**. [S.l.]: Curran Associates, Inc., 2013. v. 26. Disponível em: <https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf>.
- MOSTAFAZADEH, N.; CHAMBERS, Nathanael et al. A Corpus and Evaluation Framework for Deeper Understanding of Commonsense Stories. **ArXiv**, abs/1604.01696, 2016.
- MOSTAFAZADEH, N.; ROTH, Michael et al. LSDSem 2017 Shared Task: The Story Cloze Test. In: LSDSEM@EACL. [S.l.: s.n.], 2017.
- PAPERNO, D. et al. The LAMBADA dataset: Word prediction requiring a broad discourse context. **ArXiv**, abs/1606.06031, 2016.
- RADFORD, A.; NARASIMHAN, Karthik. Improving Language Understanding by Generative Pre-Training. In:
- RADFORD, A.; WU, Jeffrey et al. Language Models are Unsupervised Multitask Learners. In: RAFFEL, Colin et al. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. **J. Mach. Learn. Res.**, v. 21, 140:1–140:67, 2020.
- RAJPURKAR, Pranav; JIA, Robin; LIANG, Percy. Know What You Don’t Know: Unanswerable Questions for SQuAD. **ArXiv**, abs/1806.03822, 2018.

- RAJPURKAR, Pranav; ZHANG, Jian et al. SQuAD: 100, 000+ Questions for Machine Comprehension of Text. In: EMNLP. [S.l.: s.n.], 2016.
- REDDY, Siva; CHEN, Danqi; MANNING, Christopher D. CoQA: A Conversational Question Answering Challenge. **Transactions of the Association for Computational Linguistics**, v. 7, p. 249–266, 2019.
- AL-RFOU, Rami et al. Character-Level Language Modeling with Deeper Self-Attention. In: AAAI. [S.l.: s.n.], 2019.
- ROGERS, Anna; KOVALEVA, O.; RUMSHISKY, Anna. A Primer in BERTology: What We Know About How BERT Works. **Transactions of the Association for Computational Linguistics**, v. 8, p. 842–866, 2020.
- RUDER, Sebastian. **A Review of the Neural History of Natural Language Processing**. [S.l.: s.n.], 2018. <http://runder.io/a-review-of-the-recent-history-of-nlp/>.
- RUMELHART, D.E.; HINTON, G.E.; WILLIAMS, R.J. Learning Internal Representations by Error Propagation. In: COLLINS, Allan; SMITH, Edward E. (Ed.). **Readings in Cognitive Science**. [S.l.]: Morgan Kaufmann, 1988. p. 399–421. ISBN 978-1-4832-1446-7. DOI: <https://doi.org/10.1016/B978-1-4832-1446-7.50035-2>. Disponível em: <https://www.sciencedirect.com/science/article/pii/B9781483214467500352>.
- RUMELHART, David E.; HINTON, Geoffrey E.; WILLIAMS, Ronald J. Learning representations by back-propagating errors. **Nature**, v. 323, n. 6088, p. 533–536, out. 1986. ISSN 1476-4687. DOI: [10.1038/323533a0](https://doi.org/10.1038/323533a0). Disponível em: <https://doi.org/10.1038/323533a0>.
- SANH, Victor et al. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. **ArXiv**, abs/1910.01108, 2019.
- SEE, A.; LIU, Peter J.; MANNING, Christopher D. Get To The Point: Summarization with Pointer-Generator Networks. In: ACL. [S.l.: s.n.], 2017.
- SUN, Zhiqing et al. MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices. **ArXiv**, abs/2004.02984, 2020.
- SUTSKEVER, Ilya; VINYALS, Oriol; LE, Quoc V. Sequence to Sequence Learning with Neural Networks. In: \_\_\_\_\_. **Advances in Neural Information Processing Systems**. [S.l.]: Curran Associates, Inc., 2014. v. 27. Disponível em: <https://proceedings.neurips.cc/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf>.
- TAY, Yi et al. Long Range Arena: A Benchmark for Efficient Transformers. **ArXiv**, abs/2011.04006, 2020.
- TSAL, Yao-Hung Hubert et al. Transformer Dissection: An Unified Understanding for Transformer’s Attention via the Lens of Kernel. **ArXiv**, abs/1908.11775, 2019.
- VASWANI, Ashish et al. Attention is All you Need. In: \_\_\_\_\_. **Advances in Neural Information Processing Systems**. [S.l.]: Curran Associates, Inc., 2017. v. 30. Disponível em: <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf>.
- WANG, Alex; PRUKSACHATKUN, Yada et al. SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems. In: NEURIPS. [S.l.: s.n.], 2019.
- WANG, Alex; SINGH, Amanpreet et al. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In: BLACKBOXNLP@EMNLP. [S.l.: s.n.], 2018.

WANG, Sinong; LI, Belinda Z. et al. Linformer: Self-Attention with Linear Complexity. **ArXiv**, abs/2006.04768, 2020.

WARSTADT, Alex; SINGH, Amanpreet; BOWMAN, Samuel R. Neural Network Acceptability Judgments. **arXiv preprint arXiv:1805.12471**, 2018.

WILLIAMS, Adina; NANGIA, Nikita; BOWMAN, Samuel. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. In: PROCEEDINGS of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers). New Orleans, Louisiana: Association for Computational Linguistics, 2018. p. 1112–1122. Disponível em: <http://aclweb.org/anthology/N18-1101>.

XUE, Linting et al. mT5: A massively multilingual pre-trained text-to-text transformer. **ArXiv**, abs/2010.11934, 2020.

YE, Zihao et al. BP-Transformer: Modelling Long-Range Context via Binary Partitioning. **ArXiv**, abs/1911.04070, 2019.

ZAHEER, M. et al. Big Bird: Transformers for Longer Sequences. **ArXiv**, abs/2007.14062, 2020.

ZELLERS, Rowan; BISK, Yonatan et al. SWAG: A Large-Scale Adversarial Dataset for Grounded Commonsense Inference. In: PROCEEDINGS of the 2018 Conference on Empirical Methods in Natural Language Processing. Brussels, Belgium: Association for Computational Linguistics, out. 2018. p. 93–104. DOI: [10.18653/v1/D18-1009](https://doi.org/10.18653/v1/D18-1009). Disponível em: <https://www.aclweb.org/anthology/D18-1009>.

ZELLERS, Rowan; HOLTZMAN, Ari et al. HellaSwag: Can a Machine Really Finish Your Sentence? In: ACL. [S.l.: s.n.], 2019.