A INTELIGÊNCIA ARTIFICIAL CONTADA

Desafios, pessoas, modelos
e um pouco de matemática

MAGNAGO, RODRIGO

A INTELIGÊNCIA ARTIFICIAL CONTADA

Desafios, pessoas, modelos
e um pouco de matemática

MAGNAGO, RODRIGO

A INTELIGÊNCIA ARTIFICIAL CONTADA
Desafios, pessoas, modelos e um pouco de matemática

© 2025 Rodrigo C. Magnago
Todos os direitos reservados.

Esta obra encontra-se registrada junto ao Escritório de Direitos Autorais da Fundação Biblioteca Nacional e é protegida por direitos autorais.
Registro nº 483834 - 18 de novembro de 2025.

Nenhuma parte desta publicação poderá ser reproduzida, armazenada ou transmitida, por quaisquer meios existentes ou que venham a ser criados, sem a autorização prévia e por escrito do autor.

FICHA TÉCNICA
Projeto gráfico: Likewise Branding & Digital
Diagramação: Fabianne Bettega e Lilianne Caputo
Revisão:

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Magnago, Rodrigo
A inteligência artificial contada [livro eletrônico] : desafios, pessoas, modelos e um pouco de matemática / Rodrigo Magnago. -- Curitiba, PR : Ed. do Autor, 2026.
ePub
ISBN 978-65-02-21236-3

1. Inovação tecnológica 2. Inteligência artificial 3. Inteligência artificial - História 4. Linguagem 5. Matemática 6. Máquinas 7. Redes de computadores I. Título.

CDD-006.3

Índices para catálogo sistemático:
1. Inteligência artificial : Ciência da computação 006.3

Maria Alice Ferreira - Bibliotecária - CRB-8/7964

Obrigado pela leitura.

PREFÁCIO

Esta obra foi elaborada a partir de ampla pesquisa bibliográfica e da disponibilidade de artigos científicos relacionados ao tema, procurando organizar e tornar acessíveis conhecimentos que vêm sendo construídos e aperfeiçoados por inúmeras contribuições ao longo do tempo.

Contou com a revisão do professor Carlos Oliveira, atualmente docente da NYU. Por compreender que o conhecimento ganha maior valor quando compartilhado, este material é disponibilizado de forma gratuita e aberta, com a expectativa de que possa estimular novas reflexões, aplicações e desdobramentos úteis aos seus leitores e usuários.

SUMÁRIO

CAPÍTULO 1 — 8
O pano de fundo da linguagem transformada em tecnologia.

CAPÍTULO 2 — 22
Os acontecimentos no tempo, a matemática real aplicada, as estruturas de linguagem e os efeitos práticos que sentimos hoje.

CAPÍTULO 3 — 30
1980s-2010s: Hidden Markov Models (HMMs).

CAPÍTULO 4 — 46
1990s-2000s: Smoothed N-Gram Models.

CAPÍTULO 5 — 58
2003-2013: Neural Network Language Models (NNLM) - Yoshua Bengio et al.

CAPÍTULO 6 — 84
2013-2014: A era dos Embedding Models - Word2Vec e GloVe.

CAPÍTULO 7 — 104
2014: Seq2Seq e os esforços em tradução.

CAPÍTULO 8 — 128
2014-2015: Bahdanau, Luong e o nascimento do attention moderno em tradução neural.

CAPÍTULO 9 — 144
2017: Vaswani e o Transformer. Tudo que você precisa é atenção.

CAPÍTULO 10 — 176
2018: ULMFiT e ELMo - a cauda da LSTM e os momentos anteriores aos grandes modelos de linguagem.

CAPÍTULO 1

O PANO DE FUNDO DA LINGUAGEM TRANSFORMADA EM TECNOLOGIA.

Traçar a jornada dos modelos linguísticos modernos, que conhecemos por LLM (Large Language Model), e em tradução livre "Grandes Modelos de Linguagem", assusta.

Na medida que minhas notas para o que era para ser um artigo foram se avolumando, chegando a 150 páginas que abriram espaço para outras 150, e mais delas se eu não me desse ordem de parada, em algum momento "paralisei". Naquele ponto eu não soube como organizar as notas e por onde começar. As notas chegaram a milhares de páginas e deixei acontecer.

Nossa raça é engenhosa, o que torna a jornada emocionante e envolve centenas de super-heróis anônimos. Vou tentar citar alguns deles, em prejuízo a outros que serão injustiçados por não terem seus nomes devidamente agraciados.

O século XX amanheceu com a consolidação dos processos automáticos iniciados na Revolução Industrial, movimento que, embora surgido no século XVIII, transformaria profundamente a produção e a gestão no século "moderno".

Antes da chegada da máquina automática, movida a vapor, eletricidade, combustão ou energia hidráulica, as disciplinas de probabilidade, voltada à modelagem do incerto, e de estatística, voltada à coleta, organização e interpretação de dados observados, ainda tinham uso mais restrito. Com a ascensão dos movimentos taylorista e fordista, tornaram-se centrais: quem não mede, não gerencia.

Antes disso, a estatística tinha uso limitado, descritiva, pelo Estado. Censos populacionais, tributação e rotas comerciais. Ainda não havia se fundido com a probabilidade, que se concentrava até então em prever resultados em jogos de azar (sorteios, urnas, da-

dos perfeitos), o que fora um campo fértil para o desenvolvimento da probabilidade moderna.

Essa fusão entre as técnicas matemáticas e os processos automáticos de produção viria a simbolizar a Modernidade: o projeto racional de organização do mundo, baseado em medição, predição, eficiência, ordem técnica e gestão científica.

Esse paradigma não apenas moldou os modelos de eficiência empresarial, como também guiou o esforço acadêmico norte-americano, de caráter tecnocrático, especialmente na primeira metade do século XX.

Contribuições da estatística e da probabilidade ao homem moderno

É importante citarmos os ganhos, especialmente sentidos no século XX, que a modernidade trouxe a partir da estatística e da probabilidade, duas importantes disciplinas da Matemática, a fim de preparar o solo para nossas descobertas nesse livro.

- Censo e planejamento nacional, que a partir do conceito estatístico de amostragem, foram fundamentais para o planejamento urbano, a alocação de recursos e a representação política, entre outros.

- Medicina baseada em evidências, que a partir dos modelos randômicos de testagem, obteve tratamentos mais assertivos e eficazes, e se tornou modelo fundamental para a aprovação de novos tratamentos ou medicamentos.

- Controle de qualidade industrial, a partir do desenvolvimento dos controles estatísticos de processo, que permitiu a produção em massa de itens de alta qualidade, baixando os custos envolvidos e possibilitando a todos nós as condições para obtermos automóveis e eletrodomésticos a preços razoáveis.

- Criação de índices econômicos, como PIB e índices de inflação, que foi central na estabilização das economias e na formação de riqueza pelas famílias.

- Epidemiologia e saúde pública, que a partir de modelos estatísticos pode rastrear os padrões de expansão de doenças como a gripe espanhola e o HIV, entre outros exemplos.

- Projeto genoma humano, que através de bioestatística, conseguiu sequenciar e entender o genoma humano, abrindo espaço para avanços em pesquisa genética que viriam a ser a base para a prevenção e a cura de doenças, tecnologias de reprodução e o tratamento personalizado, com muito mais por vir.

- Exploração espacial, quando comprova a confiabilidade de materiais e conjuntos de engenharia para que suportem condições adversas e desconhecidas. Muitas dessas tecnologias estão disponíveis para o uso cotidiano das pessoas normais, tais como a tecnologia GPS, os painéis solares, mamografias digitais, filtros de água, garrafas térmicas e até mesmo colchões.

Ciências atuariais e as apólices de seguro, assim como os mercados financeiros, podem completar essa lista.

Um encontro acidental

Eram as décadas de 40 e 50, e nas mesmas universidades onde os modelos probabilísticos e estatísticos já ganhavam força há 50 anos em laboratórios de engenharia e departamentos industriais, havia também outro núcleo, mais distante da matemática, interessando-se em ocorrências quantitativas: o bloco das ciências humanas.

Psicólogos, linguistas e neurocientistas já se dedicavam há décadas ao estudo da linguagem, fosse por seu caráter cognitivo e antropológico, fosse pelas bases biológicas ligadas às atividades elétrica e química do cérebro.

Mas algo começava a mudar: crescia o interesse por uma visão estruturada da linguagem, especialmente quanto à sua construção em termos combinatórios. Como o cérebro organiza os fonemas para formar palavras? Como organiza as palavras para criar e manter sentido?

Eram os primeiros flertes entre linguagem, probabilidade e estatística.

Representado por Noam Chomsky, formado na Universidade da Pensilvânia e depois professor no MIT, nascido na Filadélfia e filho de pais judeus, um importante movimento de linguistas, psicólogos, filósofos e lógicos, em diversas universidades, passou a se dedicar à compreensão da formação da linguagem, oral e escrita.

Entre eles estavam Zellig Harris, orientador de Chomsky na universidade da Pensilvânia, Roman Jakobson (MIT e Harvard), russo que trouxe a tradição do Círculo Linguístico de Praga, Morris

Halle (Harvard e Princeton), e sua fonologia generativa, George A. Miller (Harvard e Princeton), e seus modelos formais da mente, e Jerry Fodor, ligado ao MIT e à Rutgers University, com a ideia de que a mente é composta por módulos operacionais.

O livro de Chomsky, Estruturas Sintáticas (1957), é seminal, e propôs que a linguagem é determinada por um conjunto de normas que são naturais no funcionamento cerebral. O livro marca essa nova fase da abordagem linguística, que deixa a abordagem humanista comportamental e se transforma no conceito fundamental em teoria de ciências computacionais e linguagem formal.

Houve tentativas anteriores de interpretar a linguagem pela perspectiva estrutural: Panini na Índia no século V a.C., Al Kindi no século IX no mundo árabe, que contou a frequência de letras em árabe para decifrar mensagens cifradas, Leon Battista Alberti, no século XV onde hoje está o território italiano, que observou a frequência de letras no latim, além de vários criptoanalistas do século XIX e alguns linguistas na virada do século XIX para o XX, como Saussure, que pensou a linguagem como um sistema de símbolos.

Mas seria o movimento de Chomsky o mais estruturado e representativo movimento de compreensão da linguagem a partir dos humanistas.

Já no lado dos "cientifistas", ou "exatistas", e antes de Chomsky e sua turma de humanas, havia amplo interesse pelo tema de codificação da linguagem, principalmente por motivos de segurança nas comunicações de guerra, em vista às mensagens criptografadas de inimigos.

Ocorre que esses esforços começam a transitar por interesses

comerciais.
No centro de um acontecimento que viria a moldar o futuro estava o MIT (Massachusetts Institute of Technology), uma empresa chamada Bell Labs, que era o principal centro de desenvolvimento de tecnologias de comunicação do planeta, e um engenheiro e matemático aplicado chamado Claude Shannon, nascido no interior do Michigan de pai funcionário público e juiz de paz da cidade, e de mãe professora de escola primária.

Shannon trabalhava para o Bell Labs, o braço de pesquisa em telefonia da AT&T, e enquanto a empresa crescia pela crescente demanda em comunicações nos EUA, problemas com a eficiência da infraestrutura (postes, fios e cabos) tentavam ser resolvidos: como transmitir mais dados com menos estrutura?

Codificando, comprimindo, garantindo a transmissão íntegra da informação. Um problema de ordem econômica no setor de comunicações.

Por seus trabalhos, em especial "A Mathematical Theory of Communication", publicado em 1948 no *Bell System Technical Journal*, Shannon passou a ser reconhecido como uma das figuras fundadoras da teoria da informação, com impacto sobre comunicação digital, compressão de dados e modelagem estatística de sequências.

Shannon usaria, de forma indireta, a mesma estrutura de pensamento de outro matemático que outrora havia se interessado pela estrutura de linguagem.

Nascido no interior do império Russo em 1856, filho de pai funcionário público e mãe do lar, Andrey Markov se tornaria mate-

mático com foco em teoria das probabilidades.
Observando a ocorrência de vogais e consoantes no texto de Eugene Onegin, de Pushkin, Markov publicou, em 1913, um experimento estatístico aplicado a símbolos linguísticos, abordando a dependência entre eventos sucessivos, ou seja, a próxima letra ocorrerá em razão da letra anterior, probabilisticamente.

Criava-se nesse momento as Cadeias de Markov, das quais o leitor terá notícias logo adiante.

Até aqui acredito que se demonstrou que o interesse pela linguagem do ponto de vista matemático (como ela se combina) esteve sempre presente na humanidade.

Pode-se até inferir algo ainda mais iluminador: entre nós, estão sempre presentes aqueles que tomam gosto pela observação dos padrões. E a Matemática passa a figurar como ferramenta central na contagem desses padrões na dimensão tempo-espaço.

Galileu e os movimentos celestes, Maxwell e a eletricidade, Mendel e a genética das ervilhas, Mendeleev e sua tabela periódica dos elementos químicos, Darwin e a seleção natural, Kepler e os movimentos planetários, Linnaeus e a taxonomia, Pasteur e teoria dos germes e doenças, Bach e os padrões da estrutura musical.

Não faltariam exemplos.

A linguagem não escapou do escrutínio desses dedicados observadores, dessa vez com Alan Turing, Claude Shannon, Andrey Markov e Noam Chomsky.

Abstraindo a ideia

Mergulhe na seguinte proposição: temos um conjunto finito de coisas, e logo compreenderemos a razão de ser finito.

Pense em uma caixa com peças de madeira com encaixes regulares, macho e fêmea, ou seja, todos os encaixes se combinam entre macho e fêmea de todas as peças. Algumas peças têm 3 encaixes, 1 fêmea e 2 machos, outras têm 5 encaixes, 4 fêmeas e 1 macho, outras têm 2 encaixes, somente machos, e assim por diante.

Quantos modelos de peças são possíveis e de quantas formas podemos combiná-las? Esse é um exercício de análise combinatória e de permutação. Quantos conjuntos posso criar com essas peças? Esse é um exercício de combinação.

Agora considere que 300 pessoas combinaram e criaram conjuntos com essas peças e você obteve os resultados. O que vem logo à cabeça? Quais foram as peças mais utilizadas? Quais as peças que são combinadas mais vezes? Quais peças vêm sempre em seguida de outras tais peças? Se você obtiver essas informações, você obteve dados estatísticos sobre essas peças e suas combinações.

Vamos mais adiante, e pensemos na seguinte situação: convidamos um amigo para combinar essas peças, montando sequências e conjuntos. Podemos nos aventurar a adivinhar as peças que ele mais vai usar e quem sabe as sequências e conjuntos que ele vai montar? Se sua resposta é sim, você acaba de adentrar no território das probabilidades.

Ocorre que, para fins de esclarecer que esse foi um exemplo simplório para criar compreensão intuitiva, 300 ocorrências é uma amostra

pequena perto da população de pessoas que podem montar peças.

Se tivéssemos uma amostragem de 1 bilhão de pessoas e suas montagens, em razão da Lei dos Grandes Números, nos aproximaríamos da verdade factual com incrível precisão em relação às preferências humanas na montagem dessas peças. Pois bem, duas grandes tradições se aproximariam dessa nova caixa de peças, a das palavras: de um lado, Chomsky e os linguistas formais, interessados nas estruturas internas da linguagem; de outro, Shannon, Markov e a tradição estatística, interessados na contagem, na codificação, na incerteza e na probabilidade das sequências. Por caminhos diferentes, ambos ajudaram a transformar a linguagem em objeto técnico.

Primeiramente levando em consideração regras sintáticas e semânticas, e depois deixando de lado essas estruturas e partindo somente para a análise combinatória e estatística da língua, esse batalhão de pessoas se dedicou a inferir probabilidades para nossa próxima palavra, escrita ou falada. Um verdadeiro jogo de adivinhar o próximo número nos dados.

E fizeram isso não a partir de 300 pessoas montando peças, mas a partir de volumes cada vez maiores de linguagem registrada. Em escala crescente, sistemas computacionais passaram a processar enciclopédias, livros, arquivos de bibliotecas, publicações científicas, textos jornalísticos, documentos universitários, poemas, letras de música, páginas da web e, mais tarde, também linguagem matemática.

Em certo sentido, começaram a "ler o mundo": contar palavras, mapear combinações, identificar recorrências e estimar o que poderia vir a seguir.

Com verdadeiros observadores de padrões, como tantos outros citados neste capítulo, esses sistemas transformaram linguagem em contagem, combinação e probabilidade.

Importante notar que, por muito tempo, essa leitura do mundo ocorreu majoritariamente em língua inglesa, tema que trataremos adiante. Ainda assim, a lógica estava dada: quanto maior o volume de textos observados, maior a capacidade de estimar continuidades prováveis. A partir disso, os modelos passaram a imaginar o que você dirá a seguir, muitas vezes com impressionante precisão, apoiados na Lei dos Grandes Números.

Referências bibliográficas

ALBERTI, Leon Battista. *De componendis cifris*. c. 1466-1467.

AL-KADĪ, Ibrahim A. Origins of Cryptology: The Arab Contributions. *Cryptologia*, v. 16, n. 2, p. 97-126, 1992.

AL-KINDĪ, Ya´qūb ibn Ishāq. Risāla fī Istikhrāj al-Mu´ammā *[A Manuscript on Deciphering Cryptographic Messages]*. c. século IX.

CHOMSKY, Noam. *Syntactic Structures*. The Hague: Mouton, 1957.

CHOMSKY, Noam; HALLE, Morris. *The Sound Pattern of English*. New York: Harper & Row, 1968.

DARWIN, Charles. *On the Origin of Species by Means of Natural Selection*. London: John Murray, 1859.
DEMING, W. Edwards. *Out of the Crisis*. Cambridge, MA: MIT Press, 1982.

FISHER, Irving. *The Making of Index Numbers.* Boston: Houghton Mifflin, 1922.

FODOR, Jerry A. *The Modularity of Mind: An Essay on Faculty Psychology.* Cambridge, MA: MIT Press, 1983.

FORD, Henry; CROWTHER, Samuel. *My Life and Work.* Garden City, NY: Doubleday, Page & Company, 1922.

GALILEI, Galileo. *Dialogo sopra i due massimi sistemi del mondo.* Firenze: Giovanni Battista Landini, 1632.

GALTON, Francis. *Natural Inheritance.* London: Macmillan, 1889.

HARRIS, Zellig S. *Methods in Structural Linguistics.* Chicago: University of Chicago Press, 1951.

INTERNATIONAL HUMAN GENOME SEQUENCING CONSORTIUM. *Initial Sequencing and Analysis of the Human Genome.* Nature, v. 409, p. 860-921, 2001.

JAKOBSON, Roman. *Selected Writings.* The Hague: Mouton, 1962-1985.

KEPLER, Johannes. *Astronomia Nova.* Heidelberg, 1609.

KERMACK, W. O.; MCKENDRICK, A. G. A Contribution to the Mathematical Theory of Epidemics. *Proceedings of the Royal Society A*, v. 115, n. 772, p. 700-721, 1927.

KUZNETS, Simon; EPSTEIN, Lillian; JENKS, Elizabeth. *National Income and Its Composition*, 1919-1938. New York: National Bureau of Economic Research, 1941.

LINNAEUS, Carolus. *Systema Naturae.* Leiden: Theodorum Haak, 1735.

MARKOV, Andrey A. An Example of Statistical Investigation of the Text *Eugene Onegin* Concerning the Connection of Samples in Chains. *Science in Context*, v. 19, n. 4, p. 591-600, 2006. Original publicado em russo em 1913.

MAXWELL, James Clerk. *A Treatise on Electricity and Magnetism.* Oxford: Clarendon Press, 1873.

MENDELEEV, Dmitri. On the Relationship of the Properties of the Elements to their Atomic Weights. *Zeitschrift für Chemie,* v. 12, p. 405-406, 1869.

MENDEL, Gregor. Versuche über Pflanzen-Hybriden. *Verhandlungen des naturforschenden Vereines in Brünn*, v. 4, p. 3-47, 1866.

MILLER, George A. *Language and Communication*. New York: McGraw-Hill, 1951.

PĀNINI. Astādhyāyī. c. séculos V-IV a.C.

PASCAL, Blaise; FERMAT, Pierre de. *Correspondance sur la théorie des probabilités.* 1654. Publicada em diversas edições modernas.

PASTEUR, Louis. *Études sur la bière: ses maladies, causes qui les provoquent, procédé pour la rendre inaltérable; avec une théorie nouvelle de la fermentation.* Paris: Gauthier-Villars, 1876.

SAUSSURE, Ferdinand de. *Cours de linguistique générale.* Lausanne/Paris: Payot, 1916.

SHANNON, Claude E. A Mathematical Theory of Communication. *The Bell System Technical Journal,* v. 27, p. 379-423; 623-656, 1948.

SHEWHART, Walter A. *Economic Control of Quality of Manufactured Product.* New York: D. Van Nostrand Company, 1931.

SNOW, John. *On the Mode of Communication of Cholera.* 2. ed. London: John Churchill, 1855.

TAYLOR, Frederick Winslow. *The Principles of Scientific Management.* New York: Harper & Brothers, 1911.

TURING, Alan M. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, ser. 2, v. 42, n. 1, p. 230-265, 1937.

TURING, Alan M. *Computing Machinery and Intelligence. Mind*, v. 59, n. 236, p. 433-460, 1950.

VERNAM, Gilbert S. Cipher Printing Telegraph Systems for Secret Wire and Radio Telegraphic Communications. *Journal of the American Institute of Electrical Engineers*, v. 45, p. 109-115, 1926.

WATSON, James D.; CRICK, Francis H. C. Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid. *Nature*, v. 171, p. 737-738, 1953.

CAPÍTULO 2

OS ACONTECIMENTOS NO TEMPO, A MATEMÁTICA REAL APLICADA, AS ESTRUTURAS DE LINGUAGEM E OS EFEITOS PRÁTICOS QUE SENTIMOS HOJE.

A evolução na linha do tempo.

Se o leitor já compreendeu que tudo se trata de um esforço em compreender a próxima palavra, ou conjunto delas, e que isso ocorre puramente por aplicação matemática (o que será mais comentado adiante), vamos dar uma olhada nos acontecimentos mais importantes na evolução dessa tecnologia no tempo.

Antes de falarmos das aplicações práticas que foram tomando corpo nos últimos 70 anos de cálculo lógico da linguagem, precisamos trazer para o contexto duas figuras importantes, Warren McCulloch e Walter Pitts.

McCulloch nasceu em Nova Jersey e estudou filosofia e psicologia na Universidade Yale e mais tarde Medicina na Universidade Columbia, enquanto Pitts, embora brilhante pensador com profundo conhecimento de lógica, matemática e filosofia, jamais obtivera graduação educacional formal.

Os dois se conheceram na Universidade de Chicago e juntos publicaram, em 1943, o artigo "Um Cálculo Lógico das Ideias Imanentes à Atividade Nervosa", atualmente considerado o artigo fundador das redes neurais artificiais, das quais falaremos adiante.

O que McCulloch e Pitts propuseram é que é possível usar modelos matemáticos e computacionais para simular o funcionamento dos neurônios do cérebro, baseando-se na ideia de que esses neurônios se comunicam por meio de sinais simples, como "sim" ou "não", "ligado" ou "desligado".

Embora o cérebro real seja muito mais complexo, esse modelo binário ajudou a compreender, de forma lógica, como redes de neurônios podem representar pensamentos. Além disso, inspirou

o desenvolvimento das redes neurais artificiais, utilizadas nos dias de hoje em sistemas de inteligência artificial para reconhecer padrões, aprender representações e resolver problemas complexos.

McCulloch e Pitts mostraram que redes neuronais idealizadas, compostas por unidades binárias simples, podiam implementar operações lógicas. Essa aproximação entre neurônios, lógica formal e computação ajudou a aproximar o estudo do cérebro das ideias de máquina universal e computabilidade formuladas por Turing.

Logo adiante, Frank Rosenblatt, nascido em 1928 em New Rochelle, Nova York, formado em psicologia pela Cornell University e mais tarde pesquisador no Cornell Aeronautical Laboratory (hoje Calspan), propôs o Perceptron, baseado nas ideias de McCulloch e Pitts.

Embora fosse psicólogo de formação, Rosenblatt tinha um forte interesse interdisciplinar, e unindo neurociência, matemática e computação, tentou entender como o cérebro processa informações. A proposta era de uma máquina inspirada no funcionamento do neurônio biológico.

Ele acreditava que a mente humana aprendia ajustando a "força" das conexões entre neurônios, e projetou um sistema que fazia o mesmo: ajustava pesos numéricos com base em erros de previsão.

O Perceptron era uma máquina com entradas (inputs), pesos (weights), um somador e uma função de ativação. Ele aprendia a classificar padrões (por exemplo, distinguir letras, figuras geométricas ou rostos), ajustando seus pesos após cada erro. Peço ao leitor que reserve esse racional e lá na frente você se encontrará com ele novamente.

O Perceptron foi anunciado publicamente pela Marinha dos EUA (que financiava o projeto) em 1958, como um marco na construção de máquinas capazes de aprender.

O evento teve ampla cobertura na imprensa. O New York Times chegou a escrever em um artigo denominado *New Navy Device Learns By Doing*, em 8 de Julho de 1958, que o Perceptron seria capaz de "andar, falar e se conscientizar de sua própria existência" no futuro. O entusiasmo era enorme, e a era pública da IA parecia ter começado ali, com promessas que excediam em muito a capacidade técnica real daquele momento.

Anos 1950 aos anos 1970: primeiros modelos estatísticos de linguagem

Utilizando uma proposição matemática desenvolvida por Andrey Markov no início do século XX e aplicada por ele, em 1913, à análise estatística de letras em *Eugene Onegin*, os primeiros modelos estatísticos de linguagem passariam a explorar a ideia de dependência entre eventos sucessivos. Esses modelos trabalhavam com matrizes que atribuíam probabilidades pré-definidas a cada transição possível. Assim, quando o sistema era provocado por uma palavra digitada, ele indicava qual seria a palavra mais provável em seguida.

Em um exemplo prático e simplório, se você digitasse a palavra "eu", o modelo, previamente alimentado com pesos estatísticos, atribuía 37% (0,37) de chance para a palavra "quero" e 22% (0,22) para a palavra "vou". Nesse cenário, escolhia a opção mais provável. É aqui que aparece o conceito matemático de probabilidade condicional: a chance de ocorrer algo (uma palavra) dado que algo já aconteceu (a palavra anterior).

Vemos aí o surgimento da ideia do que hoje chamamos de conjunto de treinamento: quando os pesquisadores analisavam manualmente grandes volumes de texto (corpus), contavam ocorrências e calculavam probabilidades de combinações observadas, inicialmente com muito mais intervenção manual e, depois, com crescente apoio computacional.

Esses modelos eram naturalmente limitados, tanto pela tecnologia da época quanto pela complexidade dos cálculos, que dependiam de recursos computacionais limitados e de procedimentos muito mais artesanais do que os atuais. Por isso, conseguiam analisar apenas duas ou três palavras anteriores por vez, formando o que chamamos de modelos de bi-grama ou tri-grama e tendo dificuldades com frases mais longas. Essa dificuldade iria acompanhar os pesquisadores em inteligência artificial por décadas.

Seu uso comercial direto ainda não havia se consolidado naquele período, mas há registros de aplicações experimentais importantes conduzidas por centros como a IBM, Bell Labs e o MIT, especialmente no campo do reconhecimento de fala. Esses modelos ajudavam a interpretar mensagens com sons ambíguos, aprimorando a comunicação por voz.

O Bell Labs, por exemplo, apresentou nos anos 1950 o sistema AUDREY, voltado ao reconhecimento de dígitos falados, enquanto a IBM demonstrou nos anos 1960 o Shoebox, capaz de reconhecer números e comandos simples. Esses sistemas ainda eram restritos, mas ajudaram a consolidar a ligação entre linguagem, estatística, acústica e computação. Também foram empregados por governos, como os dos Estados Unidos e da URSS, para analisar padrões de comunicação interceptada. Em universidades como MIT e Stanford, começaram a surgir sistemas de correção automática de palavras com base nessas técnicas.

Essas pesquisas formaram a base para o que viria a ser, anos depois, o autocompletar de palavras e frases, presente em diversos produtos, como o Microsoft Word, que hoje já utiliza modelos mais sofisticados, baseados em combinações de diferentes modelos matemáticos e computacionais.

Para o leitor ter uma ideia da profundidade histórica, os fundamentos matemáticos desses modelos de linguagem surgem em 1654, nas correspondências entre Blaise Pascal e Pierre Fermat, ao discutirem jogos de azar. O que Andrey Markov faria mais de dois séculos depois fora um avanço conceitual: tratar a dependência entre eventos sucessivos de forma sistemática, criando o modelo de cadeia de estados, em que o futuro depende apenas do presente.

A matemática por trás dos primeiros modelos

Os primeiros modelos de linguagem se baseavam em uma ideia simples: a de que o futuro dependia apenas do presente. Essa era a essência das cadeias de Markov.

Formalmente, a probabilidade de uma palavra (w_t) ocorrer em uma sequência dependia apenas de um número limitado de palavras anteriores.

No caso mais simples, o bi-grama:

$$P(w_t \mid w_{t-1})$$

Em versões mais longas, o tri-grama ou n-grama:

$$P\big(w_t \mid w_{\{t-1\}},\, w_{\{t-2\}},\, \ldots,\, w_{\{t-n+1\}}\big)$$

A ideia central era estimar essas probabilidades a partir da contagem das ocorrências no corpus, o conjunto de textos usados no treinamento. Por exemplo, se em um corpus a palavra "eu" aparecesse 10.000 vezes como contexto, e em 370 dessas ocorrências fosse seguida por "quero", o modelo calcularia:

$$P(\text{quero} \mid \text{eu}) = \frac{370}{10.000} = 0{,}037$$

Esse cálculo simples ilustrava como as máquinas "aprendiam" a linguagem: não por significado, mas por frequência. Quanto mais vezes duas palavras apareciam juntas, mais forte era o vínculo estatístico entre elas. Essa relação entre probabilidade e contexto formava o primeiro elo entre a matemática pura e a linguagem humana: o nascimento da linguística computacional moderna.

Referências bibliográficas

HEBB, Donald O. The Organization of Behavior: A Neuropsychological Theory. New York: Wiley, 1949.

FERMAT, Pierre de; PASCAL, Blaise. Correspondance sur la théorie des probabilités. 1654. Publicada em diversas edições modernas.

MARKOV, Andrey A. An Example of Statistical Investigation of the Text *Eugene Onegin* Concerning the Connection of Samples in Chains. *Science in Context*, v. 19, n. 4, p. 591-600, 2006. Original publicado em russo em 1913.

MCCULLOCH, Warren S.; PITTS, Walter. A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, v. 5, p. 115-133, 1943.

DAVIS, K. H.; BIDDULPH, R.; BALASHEK, S. Automatic Recognition of Spoken Digits. *The Journal of the Acoustical Society of America*, v. 24, n. 6, p. 637-642, 1952.

NEW YORK TIMES. New Navy Device Learns by Doing: Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser. *The New York Times*, New York, 8 jul. 1958.

ROSENBLATT, Frank. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, v. 65, n. 6, p. 386-408, 1958.

TURING, Alan M. On Computable Numbers, with an Application to the Entscheidungsproblem. P*roceedings of the London Mathematical Society*, ser. 2, v. 42, n. 1, p. 230-265, 1937.

IBM. Speech Recognition: The World's First Speech-Recognition System Was the Size of a Shoebox. IBM History, s.d.

CAPÍTULO 3

1980s-2010s: HIDDEN MARKOV MODELS (HMMS).

Agora veremos os modelos de linguagem escrita sofrerem uma paralisação que duraria aproximadamente 20 anos.

Em outras palavras, nós, a humanidade, ficamos 20 anos no modelo de Bi-Gram ou Tri-Gram de Markov, que era capaz de prever a próxima palavra que viria depois de gato, mas não conseguia observar relações entre gato, rato, miau, tigre, cão, ou outras palavras diretamente relacionadas ao contexto dos gatos.

Mesmo que se criassem modelos para relacionar palavras distantes, como por exemplo, ligar a primeira e a última palavra de uma frase com 20 palavras, o problema era outro.

Em um corpus, conjunto de textos limpos para análise, ou *dataset training*, como preferir chamar, com um vocabulário de 50 mil palavras únicas, seria relativamente simples calcular o peso estatístico de cada palavra isoladamente.

Usando linguagens como C, que era a principal linguagem usada em projetos de NLP (Processamento de Linguagem Natural), bastaria contar quantas vezes a palavra "você" aparece (por exemplo, 300 vezes) e dividir pelo total de palavras do corpus, obtendo a probabilidade de ocorrência de "você" nesse conjunto (P = 0,006 ou 0,6%).

O problema surgia quando se tentava combinar essas palavras em sequência. Para calcular a probabilidade de pares de palavras, ou seja, "você" combinada com mais uma palavra, "você" precisaria ser combinado com cada uma das 50 mil palavras do vocabulário. Para todos os pares, isso significava 50 mil ao quadrado. Se fossem trios de palavras (trigramas), ou seja, "você" combinada com mais duas palavras, o cálculo seria de 50 mil ao cubo. Aqui já estamos falando de 125 trilhões de combinações, um volume que extrapo-

lava a capacidade computacional da época.

Levaria duas décadas para transpormos esse obstáculo, período chamado de inverno da IA, marcado por muito esforço e pequenos avanços.

Em virtude de interesses governamentais e comerciais, a humanidade seguiria para outro sentido por um certo tempo. Com o aumento no uso de voz (telefones, assistentes de voz, call centers), além dos interesses estratégicos dos países desenvolvidos, os modelos passaram a se concentrar em converter sons em palavras.

O que era preciso estava disponível: a fala é bem definida em termos acústicos e é sequencial, o modelo matemático era suficiente para reconhecer os fonemas (não para compreendê-los), e nos anos 90 os processadores da geração Pentium podiam processar o reconhecimento de pequenos vocabulários. Além disso, era possível gerar receitas a partir do que já estava disponível.

Softwares que convertiam ditados em texto para usos médicos, legais e negócios, automação de call centers para bancos, empresas aéreas e de telecomunicações. A tecnologia cortou custos relevantes para essas indústrias, e sistemas de atendimento com a função "com quem você quer falar" substituíram milhares de pessoas.

Acreditava-se que o reconhecimento de fala era o santo Graal em defesa, negócios e telecomunicações, e todo mundo queria uma cadeira nessa mesa. Os Estados Unidos tinham uma agência concentrada nisso, a DARPA, que financiou pesquisas no Carnegie Mellon University, SRI International, MIT Lincoln Labs, IBM Research, AT&T Bell Labs, Stanford, entre outros.

A União Européia e o Japão não ficaram de fora, com a primeira

financiando várias iniciativas descentralizadas, mas que viriam a ser importantes futuramente como formação de corpora, e o último financiando iniciativas da NTT, Nec, Fujitsu e Hitachi, entre outras.

Mas se o modelo matemático disponível era dito suficiente, assim como a capacidade de processamento e seu uso real, como isso funcionava na prática?

Modelos baseados em HMM (Hidden Markov Models) "ouviam" um som. A partir das características desse som (frequência, amplitude, duração), que era um input definido (embora muitas vezes com ruídos), e basicamente formado por uma sequência de fonemas, o modelo buscava, entre possibilidades ocultas (palavras que não estavam diretamente observadas nos fonemas ouvidos), qual delas tinha maior probabilidade de, ao ser pronunciada, gerar sons semelhantes aos fonemas captados. Na maior probabilidade de similaridade, o modelo definia que aquele som correspondia a determinada palavra.

Como o modelo operava relacionando um som com um fonema, e, na contramão, um fonema com o som esperado ao ser pronunciado, tudo a partir de um conjunto de dados organizados (training dataset), que era basicamente um grande volume de vozes gravadas associadas a fonemas e palavras, a questão, colocada de forma simples, era de relacionamento matemático direto, um problema solucionável com a matemática e a capacidade computacional disponíveis na época.

Uma vez que o fonema era transformado em palavra escrita, novamente podiam ser aplicados os modelos bi-gram e tri-gram de Markov, utilizando corpus de palavras bem definidos, para prever a próxima palavra. Mas esse era o limite para os modelos de linguagem daquela época.

O caso mais emblemático de aplicação comercial de reconhecimento de fala à época foi o Dragon NaturallySpeaking, lançado comercialmente em 1997 pela Dragon Systems. Capaz de processar um vasto vocabulário, entre 30 e 100 mil palavras pré-treinadas (os modelos anteriores se limitavam a 1000 palavras), de reconhecer a fala natural e continuada, e de calibragem rápida da voz do usuário (os modelos anteriores demandavam horas de treino da voz do usuário), o Dragon Naturally Speaking foi a primeira demonstração comercial de massa que indicava que o reconhecimento de voz não era somente uma curiosidade dos pesquisadores.

Com o uso eficiente do algoritmo que Andrew Viterbi, italiano, professor da UCLA e fundador da Qualcomm, criou em 1967, e que basicamente se resume a um grafo onde, a cada instante, o algoritmo acumula o melhor score de cada caminho possível (forward), e ao final faz um backtracking para reconstruir o melhor caminho global (backward), o reconhecimento de voz ganhou força, principalmente a partir do Dragon, para se tornar uma peça fundamental nos modelos de comunicação e computação atuais. Por caminhos tortuosos, o Dragon acabou nas mãos da Microsoft em 2022 para integrar serviços de reconhecimento de voz em soluções Cloud e Enterprise.

James Baker, com sua tese de doutorado "Modelagem Estocástica como um Meio para o Reconhecimento Automático de Fala", e Janet Baker, sua esposa, ambos PhD pelo Carnegie Mellon e criadores do Dragon Naturally Speaking, trabalharam por mais de 15 anos no desenvolvimento dessas soluções, e estão entre os nomes centrais da transição do reconhecimento de fala de laboratório para aplicações comerciais de maior escala. Acabaram por nunca receber um único centavo por sua obra, em meio às disputas e perdas empresariais ocorridas após a venda da Dragon Systems

para a belga Lernout & Hauspie, cujos ativos depois passaram pela ScanSoft, empresa que se tornaria Nuance, adquirida pela Microsoft em 2022.

Outro movimento ocorre nesse mesmo período histórico, que conto agora.

O Motor silencioso: scanner e OCR na formação dos grandes bancos de texto

O surgimento de duas tecnologias constituiria as bases fundamentais para os modelos modernos de linguagem que conhecemos hoje.

O scanner, em sentido próximo ao que conhecemos hoje, ganhou uma base experimental importante nos trabalhos de Russell A. Kirsch, no National Bureau of Standards, atual NIST, a partir de 1956.

O OCR, por sua vez, começava a resolver o problema seguinte: não apenas capturar a imagem do texto, mas convertê-la em caracteres interpretáveis pela máquina. Esse avanço aparece de forma pioneira nos trabalhos de David H. Shepard, cuja patente "Apparatus for Reading", depositada em 1951 e concedida em 1953, descrevia uma máquina voltada ao reconhecimento de caracteres impressos.

Os primeiros usos comerciais vieram a partir de empresas como a de Shepard (IMR Corp), a IBM, a NCR e a Recognition Equipment Inc, principalmente utilizados para a leitura de cheques bancários, documentos oficiais e formulários. Eram caros e capazes de ler uma ou duas fontes específicas (OCR-A e OCR-B).

O modelo trabalhava com um conjunto de imagens de referência contendo diversas representações de cada letra do alfabeto, assim como de cada número.

Deixando de lado pontuação e outros símbolos da escrita, pense o leitor em quase 40 caracteres e, digamos, 100 diferentes formas de escrever cada um deles, o que resultaria em 4.000 imagens de referência.

O sistema analisava a matriz de pixels dessas imagens, comparando a intensidade dos tons. Regiões mais claras formavam o fundo e regiões mais escuras formavam os caracteres, a mesma base de tecnologias de reconhecimento facial ou de veículos autônomos que vemos atualmente.

A partir daí, o modelo relacionava a imagem capturada às imagens de referência, estimando de qual caractere se tratava, e convertia a imagem em texto.

Nesse contexto, o visionário Ray Kurzweil, engenheiro elétrico formado no MIT, que criou o primeiro OCR capaz de lidar com múltiplas fontes tipográficas, foi um dos primeiros a perceber que bibliotecas inteiras poderiam ser digitalizadas.

Sua empresa foi comprada pela Xerox em 1980 e se tornou a Xerox Imaging Systems. Foi essa tecnologia que abriu caminho para as digitalizações de larga escala. Kurzweil trabalha desde 2012 no Google.

Este livro não está tratando do processamento de imagens, o que faremos em momento oportuno, mas é importante delinearmos o funcionamento dessa dupla, scanner e OCR, e atestar que foram a base conceitual para os modelos modernos de IA Multimodal, que fazem em larga escala, com imagens complexas e coloridas, o

que o OCR fazia com um único tipo de imagem: o texto visual.

O movimento de digitalizar documentos, obtendo as imagens dos textos a partir de scanners e convertendo-as em textos digitais a partir do OCR, começou discreto e foi ganhando força. Jornais como o New York Times iniciaram a digitalização de seus próprios arquivos, o que mais tarde se tornaria um dos importantes corpora para diversos projetos, o NYTimes Historical Archive. Parte dessa digitalização foi realizada em parceria com empresas de arquivamento.

Jornais, coleções jurídicas e acervos acadêmicos foram digitalizados por bibliotecas nacionais, órgãos governamentais, universidades, projetos de pesquisa e arquivos históricos com financiamento governamental.

Quando o movimento ganhou força, já nos anos 90, grandes projetos de digitalização começaram a ocorrer, como o JSTOR (artigos científicos), LexisNexis (jurisprudência e documentos legais), a Biblioteca Britânica e a Biblioteca do Congresso Americano, entre outros.

Algumas empresas ganharam relevância nesse contexto, como ABBYY na Rússia, Readiris na Bélgica, e novamente a empresa que viria a se tornar a Nuance, antes ScanSoft, da qual já falamos no caso do Dragon e de James e Janet Baker.

Um dos projetos mais importantes na construção de corpora linguísticos naquele momento foi o LDC (Linguistic Data Consortium), liderado pela Universidade da Pensilvânia, com apoio direto do governo americano através da DARPA.

O LDC produziu, por exemplo, o Gigaword, um dos maiores cor-

pus de texto de notícias disponível, e o Penn Tree Bank, corpus anotado com estrutura sintática. Quase todos os papers de NLP entre 1992 e 2015 usaram corpora do LDC, e esses conjuntos estão presentes direta ou indiretamente em todos os modelos modernos de linguagem.

Mais tarde, em 2004, um projeto que muitas vezes passa despercebido na perspectiva histórica, consolidaria a transformação de parte dos textos da humanidade em corpus digital: o Google Books, inicialmente anunciado como parte do Google Print/Library Project. Em parceria com algumas das maiores bibliotecas de pesquisa do mundo, como Harvard, Stanford, Oxford, University of Michigan e New York Public Library, o Google passou a digitalizar acervos em escala inédita, convertendo livros físicos em imagens pesquisáveis e, quando possível, em texto processável por OCR.

O objetivo declarado dialogava diretamente com a ambição mais ampla da empresa: organizar a informação do mundo e torná-la acessível. No caso dos livros, isso significava transformar séculos de produção textual, antes presos a prateleiras, catálogos e arquivos físicos, em material indexável, buscável e parcialmente acessível por sistemas digitais.

O projeto colocou em circulação digital uma parcela imensa da produção textual acumulada ao longo de séculos, incluindo literatura, ciência, jornalismo, religião, manuais técnicos e poesia. Ainda que nem todo esse material tenha se tornado integralmente acessível, e ainda que o OCR produzisse imperfeições, o Google Books ampliou radicalmente a quantidade de texto pesquisável em escala global.

Para a história dos modelos de linguagem, sua importância estava justamente nessa combinação entre volume, diversidade e curadoria

editorial. Diferente da web aberta, marcada por ruído, duplicação e baixa padronização, os livros digitalizados ofereciam textos mais longos, estruturados e linguisticamente ricos. O Google Books, portanto, não apenas transformou bibliotecas físicas em acervos pesquisáveis, mas também ajudou a consolidar a ideia de que grandes massas textuais digitalizadas poderiam servir como infraestrutura para busca, indexação, tradução automática, correção textual e, posteriormente, modelos estatísticos e neurais de linguagem.

O projeto também revelou a tensão central dessa nova etapa: a mesma infraestrutura que tornava o conhecimento pesquisável em escala inédita levantava questões difíceis sobre direitos autorais, reprodução de obras, acesso público e controle privado sobre acervos culturais. A disputa culminaria em processos relevantes, como Authors Guild v. Google, no qual a Justiça norte-americana analisou a digitalização de milhões de livros e o uso de trechos pesquisáveis no Google Books.

Enquanto os avanços em modelos matemáticos e computacionais, e na capacidade de processamento per se, dependiam de deixar para trás os modelos n-gram de Markov, nossa produção intelectual de séculos era digitalizada para se tornar a base da revolução que se anunciava.

Um pouco da matemática dos primeiros modelos

Os Modelos Ocultos de Markov (HMMs) representavam a fala como uma sequência de estados invisíveis (os fonemas e palavras) que geravam observações audíveis, os sons propriamente ditos.
A estrutura era definida por três elementos principais:

Um conjunto de estados ocultos:

$$S = \{1, 2, \ldots, N\}$$

Uma sequência de observações:

$$O = (o_1, o_2, \ldots, o_T)$$

Três grupos de probabilidades:

Probabilidade inicial de cada estado, que indica a probabilidade de o sistema iniciar no estado oculto i.

$$\pi_i = P(q_1 = i)$$

Probabilidades de transição entre estados, que indicam a probabilidade de o sistema passar do estado oculto i, no tempo $(t-1)$, para o estado oculto j, no tempo t.

$$a_{ij} = P(q_t = j \mid q_{t-1} = i)$$

Probabilidades de emissão, que indicam a probabilidade de observar $o_{t,}$, dado que o sistema está no estado oculto j no tempo t.

$$b_j(o_t) = P(o_t \mid q_t = j)$$

Com esses elementos, o modelo podia calcular a probabilidade conjunta de uma sequência observada O e de uma sequência específica de estados ocultos Q:

$$P(O, Q) = P(q_1) \prod_{t=2}^{T} P(q_t \mid q_{t-1}) \prod_{t=1}^{T} P(o_t \mid q_t)$$

Essa formulação parecia simples, mas trazia uma elegância poderosa: descrevia o som da fala como o produto de eventos dependentes no tempo.

Quando se desejar descobrir qual sequência de estados ocultos mas, provavelmente gerara os sons observados, entrava em cena o algoritmo de Viterbi, introduzido em 1967.

Ele percorria todos os caminhos possíveis, acumulando as probabilidades mais altas a cada passo (*forward*) e, ao final, refazia o caminho de trás para frente (*backtracking*) para encontrar a sequência mais provável de fonemas ou palavras. Formalmente, o cálculo seguia a recursão:

$$\delta_t(j) = \max_i \left[\delta_{t-1}(i) \cdot a_{ij}\right] b_j(o_t)$$

onde ($\delta_t(j)$) representava a probabilidade do caminho mais provável que terminava no estado (j) no instante (t).

Essa combinação de cadeias de Markov, teoria da probabilidade e dinâmica sequencial fez dos HMMs o primeiro modelo estatístico de larga aplicação prática em IA.

Embora suas equações fossem lineares e discretas, elas já capturavam a essência de um fenômeno temporal e probabilístico: a ideia de que o passado influencia o futuro.

Referências bibliográficas

AUTHORS GUILD v. GOOGLE, INC. 804 F.3d 202. United States Court of Appeals for the Second Circuit, 2015.

BAKER, James K. Stochastic Modeling as a Means of Automatic Speech Recognition. 1975. Tese de doutorado, Carnegie Mellon University.

BAKER, James K. The DRAGON System: An Overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, v. 23, n. 1, p. 24-29, 1975.

BAUM, Leonard E.; PETRIE, Ted; SOULES, George; WEISS, Norman. A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains. *Annals of Mathematical Statistics*, v. 41, n. 1, p. 164-171, 1970.

GRAFF, David; CIERI, Christopher. English Gigaword. Philadelphia: Linguistic Data Consortium, 2003. Catálogo LDC2003T05.

HARVARD GAZETTE. Google to Digitize Some Harvard Library Holdings. Cambridge, MA, 16 dez. 2004.

JELINEK, Frederick. Statistical Methods for Speech Recognition. Cambridge, MA: MIT Press, 1997.

KIRSCH, Russell A. SEAC and the Start of Image Processing at the National Bureau of Standards. *IEEE Annals of the History of Computing*, v. 20, n. 2, p. 7-13, 1998.

KLEINER, A.; KURZWEIL, R. C. A Description of the Kurzweil Reading Machine and a Status Report on Its Testing and Dissemination. *Bulletin of Prosthetics Research*, v. 10, n. 27, p. 72-81, 1977.

MARCUS, Mitchell P.; SANTORINI, Beatrice; MARCINKIEWICZ, Mary Ann. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, v. 19, n. 2, p. 313-330, 1993.

MARKOV, Andrey A. An Example of Statistical Investigation of the Text *Eugene Onegin* Concerning the Connection of Samples in Chains. *Science in Context*, v. 19, n. 4, p. 591-600, 2006. Original publicado em russo em 1913.

MICROSOFT. Microsoft Completes Acquisition of Nuance, Ushering in New Era of Outcomes-Based AI. Redmond, 4 mar. 2022.

RABINER, Lawrence R. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, v. 77, n. 2, p. 257-286, 1989.

RABINER, Lawrence R.; JUANG, Biing-Hwang. Fundamentals of Speech Recognition. Englewood Cliffs, NJ: Prentice Hall, 1993.

SHEPARD, David H. Apparatus for Reading. U.S. Patent n. 2,663,758. Pedido em 1 mar. 1951; concedido em 22 dez. 1953.

VITERBI, Andrew J. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *IEEE Transactions on Information Theory*, v. 13, n. 2, p. 260-269, 1967.

WIRED. There's No Need to Speak Slowly and Clearly. *Wired*, 2 abr. 1997.

WIRED. Finally, a Computer That Understands You. *Wired*, jun. 1997.

WIRED. The Library of Google. *Wired*, 14 dez. 2004.

CAPÍTULO 4

1990s-2000s: SMOOTHED N-GRAM MODELS.

A batalha para aprimorar os modelos linguísticos prosseguia, e o desafio central era calibrá-los para que conseguissem processar combinações de palavras jamais observadas.

Ou seja, além dos bigramas ou trigramas com palavras que estavam sempre juntas nos textos, era desejo dos pesquisadores considerar combinações menos frequentes, como "gato" e "sofá".

No processo chamado suavização, buscava-se evitar que o sistema considerasse como probabilisticamente impossível a ocorrência de sequências como "o gato no sofá", apenas por não a ter registrado nos conjuntos de treinamento, aqueles grandes volumes de jornais e livros digitalizados por OCR, conhecidos como corpus (ou corpora no conjunto), em número suficiente. Isso ocorria nos modelos n-gram de Markov, causando o que é chamado de *Sparsity*, que basicamente decorre da diferença entre o número de combinações possíveis e o número efetivamente registrado no corpus.

Com um vocabulário de V palavras e um modelo N-gram de ordem N, o total de sequências distintas que podem ser formadas é de V elevado a N. Por exemplo, com um vocabulário de apenas 5 palavras (V=5) e considerando frases de 3 palavras (trigramas, $N = 3$), já teríamos 5 elevados ao cubo, ou 125 combinações possíveis.

Não é porque o modelo só conhece 10 combinações que as outras 115 não sejam possíveis gramaticalmente. À medida que o vocabulário cresce, esse número cresce exponencialmente.

Com a ferramenta mais básica conhecida, a suavização de Pierre-

-Simon Laplace (1749-1827), e na ausência de qualquer registro dessa combinação, o modelo assumia a existência de pelo menos um caso, impedindo que o cálculo probabilístico resultasse em probabilidades nulas (ou, em situações mais extremas, divisões por zero quando o próprio contexto não fora observado).

Assim, em vez de falhar na sugestão de uma palavra após "o gato no sofá", o modelo ainda conseguiria oferecer algumas opções, ainda que, previsivelmente, não fossem as mais sensatas. Afinal, o que poderia surgir após combinações jamais vistas? Normalmente, a continuidade de combinações insensatas do ponto de vista estatístico e gramatical. Aqui aparece uma analogia embrionária com aquilo que, em modelos generativos modernos, passaria a ser chamado de alucinação: a produção de uma continuidade formalmente possível, mas estatisticamente, gramaticalmente ou semanticamente inadequada.

O modelo de Laplace achatava a distribuição de probabilidades em razão da adição do vocabulário no denominador da fórmula (o total de palavras do conjunto de treinamento), distribuindo cargas probabilísticas relativamente semelhantes entre diferentes "próximas" palavras, o que limitava a capacidade de gerar predições úteis em ferramentas de autocompletar ou de correção de texto, já que atribuía, por regra (rule based), um "pouco de probabilidade" para cada token, inclusive àqueles semanticamente improváveis.

Se um modelo não sabe estimar adequadamente a próxima palavra após uma sequência linguisticamente possível, mas ausente do corpus de treinamento, qual seria sua utilidade prática?

A suavização de Laplace, ou *add-one smoothing*, tornou-se uma das formulações clássicas para enfrentar o problema das probabilidades nulas em modelos estatísticos de linguagem. Embora

simples demais para os grandes modelos n-gram que se consolidariam depois, ela introduziu a intuição central da suavização: reservar alguma massa probabilística para eventos não observados.

Essa preocupação atravessaria os trabalhos posteriores de modelagem estatística da linguagem em centros como IBM, Bell Labs/AT&T Bell Labs e universidades norte-americanas, nos quais métodos mais sofisticados, como Jelinek-Mercer, Good-Turing, Katz e Kneser-Ney, ganhariam espaço.

O maior problema da suavização de Laplace era tratar de forma excessivamente parecida palavras já observadas e palavras nunca vistas, diluindo as diferenças reais de frequência entre as combinações. No caso de "o gato no sofá", por exemplo, o modelo de Laplace conseguiria atribuir alguma probabilidade à sequência, mas ainda não conseguiria perceber que a palavra "gato" já havia aparecido em diversos outros contextos mais comuns, como "o gato era branco" ou "o gato dormia". Essa limitação mostrava que o simples ajuste estatístico ainda estava longe de capturar a estrutura contextual da linguagem.

Diante desse obstáculo, a modelagem estatística da linguagem avançaria para métodos mais sofisticados, capazes de considerar não apenas a ocorrência bruta das sequências, mas também a forma como as palavras se distribuíam em diferentes contextos.

Nesse contexto, Reinhard Kneser, pesquisador da Philips Research Laboratories em Aachen, e Hermann Ney, formado em Física pela Universidade de Göttingen, doutor em Engenharia Elétrica pela Universidade de Braunschweig e professor da RWTH Aachen, pertenciam à tradição alemã de pesquisa em reconhecimento de fala e modelagem estatística da linguagem, em diálogo com a linhagem estatística consolidada por centros industriais como

IBM e Philips. Em 1995, escreveram o artigo "Improved Backing-Off for M-Gram Language Modeling", apresentado na Conferência Internacional da IEEE sobre Acústica, Fala e Processamento de Sinais, a ICASSP 1995.

A partir de princípios semelhantes aos já utilizados por Slava M. Katz, do IBM T. J. Watson Research Center em Nova Iorque, Kneser e Ney precisavam atribuir alguma probabilidade às ocorrências da palavra gato observadas em outros contextos.

Se "gato" tivesse sido visto junto a "branco" ou "dormindo", o modelo deveria levar em conta essas ocorrências para ponderar possibilidades alternativas, indicando que gato era uma palavra frequente em diferentes combinações, mesmo que distintas da combinação buscada. Assim, a palavra gato e suas diversas combinações ganhavam peso estatístico, mesmo que inferior à combinação mais comum, como "o gato miou".

Katz já havia proposto aplicar um "desconto" sobre as contagens das ocorrências mais relevantes, a fim de reservar uma parte da "massa total de probabilidade" para combinações não observadas diretamente. Como a soma total das probabilidades deve ser igual a 1, Katz aplicava desconto nas ocorrências observadas para redistribuí-la entre os eventos de ordem inferior.

O que Kneser e Ney fizeram foi redistribuir de maneira mais inteligente a probabilidade restante originada pelo desconto, baseando-se não apenas na frequência, mas na quantidade de contextos distintos em que as palavras apareciam. Além disso, refinaram o próprio desconto aplicado, substituindo o ajuste Good-Turing, formalizado por Irving J. Good em 1953 com base nas ideias de Alan Turing, por um desconto absoluto mais robusto e empiricamente eficiente.

Se Laplace achatava a distribuição atribuindo probabilidades artificiais uniformes aos *n-gramas* ausentes ou raros, Kneser e Ney procuravam redistribuir as probabilidades com base na frequência de continuidade dos *n-gramas*, levando em conta em quantos contextos distintos uma palavra ocorria. Nesse caso, ajustando a curva não para um perfil "normalizado", e sim para um perfil empírico mais aderente ao padrão de distribuição real de frequência da linguagem, tornando o modelo mais sensível às regularidades naturais do texto.

Era o início de uma estrutura de cálculo estatístico e probabilístico que viria a moldar os modelos de linguagem no futuro: eu vi a palavra gato nesse contexto ("o gato miou") diversas vezes, mas também vi "gato" em combinação com outras palavras em vários outros contextos. Basicamente, o modelo começava a responder com base no contexto direto, mas também a incorporar informações sobre outras ocorrências, ampliando sua capacidade de generalização. Tratava-se de uma ideia sofisticada: a de valor estatístico contextual.

Mais do que uma inovação matemática, a abordagem de Kneser e Ney às combinações léxicas representou um avanço na tentativa de lidar com a generalização, esforço que, embora distante formalmente, preparou o terreno conceitual para os modelos de linguagem modernos baseados em *backpropagation*, como veremos adiante.

Reinhard Kneser faleceu em 2012, aos 54 anos de idade.

Hermann Ney, por sua vez, aposentou-se de sua posição de professor na RWTH Aachen, mas permaneceu ligado à pesquisa aplicada em tecnologias de linguagem. Sua trajetória posterior inclui tanto o reconhecimento acadêmico de sua carreira, com ce-

lebração pública de aposentadoria na RWTH Aachen em 2022, quanto a continuidade de atuação científica em iniciativas privadas como a AppTek.

A Matemática por trás dos Modelos N-Gram e da suavização

Os modelos N-gram se baseavam em uma ideia simples e poderosa: a probabilidade de uma palavra dependeria apenas de um número limitado de palavras anteriores. Formalmente, expressava-se como:

$$P(w_t \mid w_1^{t-1}) \approx P(w_t \mid w_{t-N+1}^{t-1})$$

onde w_t representava a palavra atual, e o termo à direita indicava que apenas as $N-1$ palavras anteriores eram consideradas.
Assim, em um modelo trigrama ($N = 3$), a probabilidade de "sofá" após "gato no" era estimada por:

$$P(\text{sofá} \mid \text{gato no})$$

Se quiséssemos estimar a probabilidade de "sofá" após "o gato no", estaríamos usando um modelo de ordem superior, isto é, um 4-grama:

$$P(\text{sofá} \mid \text{gato no})$$

A estimativa mais direta dessas probabilidades vinha da contagem das ocorrências no corpus:

$$P(w_t \mid w_{t-N+1}^{t-1}) = \frac{C(w_{t-N+1}^{t})}{C(w_{t-N+1}^{t-1})}$$

em que $C(w_{t-N+1}^{t})$ representa o número de vezes em que a sequência completa apareceu no corpus, incluindo a palavra w_t, enquanto $C(w_{t-N+1}^{t-1})$ representa o número de vezes em que o contexto anterior apareceu.

Essa era a forma "bruta" do modelo de Markov, e o problema surgia quando uma combinação nunca fora observada: o numerador se tornava zero, e a probabilidade da sequência inteira colapsava.

Pierre-Simon Laplace propusera um método para evitar probabilidades nulas, que seria reaproveitado na modelagem de linguagem. O princípio consistia em adicionar uma contagem fictícia, um pequeno "+1", a todas as possíveis ocorrências:

$$P_{\text{Laplace}}(w_t \mid w_{t-N+1}^{t-1}) = \frac{C(w_{t-N+1}^{t}) + 1}{C(w_{t-N+1}^{t-1}) + V}$$

em que V correspondia ao tamanho do vocabulário.

A soma de 1 no numerador garantia que nenhuma combinação tivesse probabilidade zero, e a soma de V no denominador mantinha a normalização. Na prática, o modelo "achatava" a distribuição de probabilidades, atribuindo um pequeno valor até mesmo a sequências jamais vistas. Essa operação impedia falhas matemáticas, mas também reduzia o contraste entre eventos frequentes e raros.

Slava M. Katz, da IBM, propôs em 1987 uma solução mais sofisticada: aplicar um desconto sobre as contagens observadas para reservar uma fração da "massa de probabilidade" às combinações ausentes.

O modelo de Katz partia da ideia de que as probabilidades deveriam obedecer à soma total igual a 1. Assim, aplicava-se um des-

conto às probabilidades estimadas para eventos observados, reservando parte da massa probabilística para eventos não observados.

O cálculo envolvia duas etapas. Para N-gramas observados, reduzia-se a probabilidade estimada diretamente pelas frequências, aplicando um fator de desconto. A massa probabilística retirada desses casos era então reservada para sequências não observadas. Para os N-gramas observados, a forma era:

$$P_{\text{Katz}}(w_t \mid h) = d_{C(h,w_t)} \cdot \frac{C(h, w_t)}{C(h)}$$

em que h indica o histórico anterior:

$$h = w_{t-N+1}^{t-1}$$

e $C(h, w_t)$ representa a contagem do N-grama completo.
Para os N-gramas ausentes, a probabilidade era "emprestada" de um modelo de ordem inferior, no mecanismo chamado back-off:

$$P_{\text{Katz}}(w_t \mid h) = \alpha(h) \cdot P_{\text{Katz}}(w_t \mid h')$$

em que h' representa um histórico mais curto, isto é, o contexto depois de remover a palavra mais antiga de h.

A forma completa pode ser escrita como:

$$P_{\text{Katz}}(w_t \mid h) = \begin{cases} d_{C(h,w_t)} \dfrac{C(h, w_t)}{C(h)}, & C(h, w_t) > 0 \\ \alpha(h)\, P_{\text{Katz}}(w_t \mid h'), & C(h, w_t) = 0 \end{cases}$$

em que h' representava um histórico mais curto, como um bigrama em vez de um trigrama. Dessa forma, o modelo preservava as relações empíricas mais fortes e ainda oferecia uma probabilidade razoável para sequências novas.

Em 1995, Reinhard Kneser e Hermann Ney propuseram uma forma mais robusta de suavização, baseada em desconto absoluto e redistribuição da massa probabilística. A inovação central estava em considerar não apenas a frequência bruta de uma palavra, mas também a diversidade de contextos nos quais ela aparecia.

A forma simplificada do modelo Kneser-Ney pode ser escrita como:

$$P_{\mathrm{KN}}(w_t \mid h) = \frac{\max\{C(h, w_t) - D,\ 0\}}{C(h)} + \lambda(h)\, P_{\mathrm{cont}}(w_t)$$

em que D representava o desconto fixo e $P_{\mathrm{cont}}(w_t)$ media a probabilidade de continuação da palavra w_t, isto é, levava em conta o número de diferentes contextos em que w_t aparecia, e não apenas sua contagem total. Isso tornava o modelo mais sensível à variedade contextual das palavras.

Além disso, nessa família de métodos, é importante distinguir duas estratégias: o back-off, no qual o modelo retrocedia para ordens inferiores apenas quando a sequência de ordem superior não existia, e a interpolação (*interpolation*), em que o modelo combinava, de forma ponderada, as probabilidades de diferentes ordens, como unigramas, bigramas e trigramas, em todas as situações.

Na prática, a interpolação suavizava continuamente a transição entre ordens e produzia distribuições mais estáveis, enquanto o

back-off preservava uma lógica mais parcimoniosa, recorrendo a modelos de ordem inferior apenas quando necessário. Essa formulação consolidou-se como uma das bases estatísticas mais sólidas da era pré-neural da modelagem de linguagem.

Os métodos de Laplace, Katz e Kneser-Ney representaram etapas fundamentais na tentativa de dar às máquinas uma noção estatística de continuidade contextual: a probabilidade de uma palavra não depende apenas de sua frequência isolada, mas também do ambiente linguístico em que ela aparece. Décadas mais tarde, essa intuição seria reinterpretada pelos modelos neurais e pelos mecanismos de atenção.

Referências bibliográficas

CHEN, Stanley F.; GOODMAN, Joshua. An Empirical Study of Smoothing Techniques for Language Modeling. *Computer Speech & Language*, v. 13, n. 4, p. 359-394, 1999.

GALE, William A.; CHURCH, Kenneth W. What's Wrong with Adding One? In: OOSTDIJK, Nelleke; DE HAAN, Pieter. *Corpus-Based Research into Language: In Honour of Jan Aarts*. Amsterdam: Rodopi, 1994. p. 189-200.
Essa é a melhor referência para sustentar a crítica ao *add-one smoothing*; o próprio texto define e discute o método *add-one* em estimativas de bigramas.

GOOD, Irving J. The Population Frequencies of Species and the Estimation of Population Parameters. *Biometrika*, v. 40, n. 3/4, p. 237-264, 1953.

GOOD, Irving J. Studies in the History of Probability and Sta-

tistics. XXXVII. A. M. Turing's Statistical Work in World War II. *Biometrika*, v. 66, n. 2, p. 393-396, 1979.

JELINEK, Frederick; MERCER, Robert L. Interpolated Estimation of Markov Source Parameters from Sparse Data. In: *Proceedings of the Workshop on Pattern Recognition in Practice*. Amsterdam: North-Holland, 1980. p. 381-397.

JURAFSKY, Daniel; MARTIN, James H. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models. 3. ed. Online manuscript released January 6, 2026. Stanford, 2026.

KATZ, Slava M. Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, v. 35, n. 3, p. 400-401, 1987.

KNESER, Reinhard; NEY, Hermann. Improved Backing-Off for M-Gram Language Modeling. In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*. Detroit: IEEE, 1995. v. 1, p. 181-184.

LAPLACE, Pierre-Simon. Théorie analytique des probabilités. Paris: Courcier, 1812.

CAPÍTULO 5

2003-2013: NEURAL NETWORK LANGUAGE MODELS (NNLM) - YOSHUA BENGIO ET AL.

Após um inverno nos esforços em Inteligência Artificial causado pelas limitações na capacidade computacional, afinal certos modelos matemáticos eram complicados demais e tomavam muito tempo das CPU's para processamento, além do ceticismo que tomou conta do espírito dos envolvidos à época em razão dos modelos matemáticos utilizados nos n-gramas terem limitações que ficaram explícitas nas aplicações, outras possibilidades passaram a ser aventadas.

Enquanto os n-gram eram modelos puramente estatísticos, que se limitavam a medir a relação de uma palavra com uma, duas ou três anteriores em suas aplicações, ou contar as relações de uma palavra com outra em um conjunto de textos (algo puramente probabilístico), e se o desejo fosse medir a relação de uma palavra com todas as outras de certo léxico (vocabulário), relembrando de certa forma o funcionamento de nosso cérebro e o que havia sido teorizado por Chomsky, modelos matemáticos mais complexos precisavam entrar em campo.

Álgebra linear, cálculo, otimização e geometria seriam os caminhos para um modelo que conseguisse aprender, diferenciar e se auto otimizar, em processo contínuo. Abandonava-se a matemática discreta pura aplicada nos modelos n-gram em função do abandono da estrita contagem de eventos discretos para, então, operar representações contínuas e apreendíveis. Vamos tentar quebrar esses modelos em partes que possibilitem um entendimento intuitivo.

Se as relações que uma palavra tinha com outras palavras nos textos do corpora era simplesmente por proximidade com outras palavras nos modelos n-gram (contagem pura), é óbvio que o modelo jamais saberia, por exemplo, que “gato” e “sofá”, “gato” e “tigre”,

"gato" e "branco", "gato" e "árvore", mantinham relações indiretas no mundo da gramática e da cognição.

Como seria possível atribuir tantas relações que cada palavra tinha com tantos contextos e com tantas outras palavras? A resposta seria encontrada nas matrizes, as mesmas que aprendemos na educação formal, no segundo grau, no Brasil.

Imagine vários textos, muitos de fato, que tivessem algo em comum: um vocabulário com 1000 palavras distintas. Agora pense em atribuir características para cada palavra desse vocabulário para compreender suas relações contextuais.

Por ora, sugiro que o leitor registre que a beleza dos modelos atuais de linguagem é que eles interpretam e se organizam a partir das características de cada palavra, e voltam para corrigir tais características a todo instante.

No meio do caminho houve um fato incidental

Operações com matrizes, e considere que uma matriz de bom tamanho para modelos clássicos de linguagem que antecederam o "Transformer", como é o caso do NNLM tratado nesse capítulo, era de 100.000 palavras distintas com 300 dimensões (100.000x300), são custosas do ponto de vista computacional, ou seja, consomem tempo. As CPU's tradicionais eram desenhadas para propósitos gerais e atividades sequenciais. Tinham poucos "núcleos", ou seja, unidades de processamento das informações, e eram boas em mudar de atividade repentinamente.

Do outro lado da sala estavam os gamers. Suas GPU's tinham hardware especializado que contavam com milhares de pequenos

núcleos (cores), usava computação paralela eficientemente, processava *arrays* enormes, e operava matrizes também eficientemente.

Um *array* nada mais é do que uma estrutura organizada de dados numéricos, semelhante a uma planilha ou matriz, na qual cada posição é processada simultaneamente por um núcleo distinto. Tudo isso usado para renderizar quadros de imagens em jogos, onde cada pixel é independente.

Essas GPU's desenhavam milhões de pixels por segundo, enchendo a tela com cores e formatos, aplicavam transformações em 3D em tempo real, rotacionando, aumentando ou diminuindo, e movendo personagens (para o que usavam muitas operações de matrizes), aplicavam simulações e equações físicas quando o personagem caía ou batia contra algo, e faziam matemática em iluminação, refletindo luz e criando sombra, o que requer trigonometria, cálculo e vetores.

Em algum momento, nos anos 2000, os pesquisadores em IA pensaram: "espera aí, uma rede neural é somente... matemática de matrizes!"

O que redes neurais precisam para funcionar é multiplicar matrizes enormes com vetores dimensionais, voltar com ajustes gradientes e fazer isso bilhões de vezes na fase de treinamento. Os pesquisadores olharam para as GPU's de videogames e disseram: "vamos treinar os modelos de IA nisso aí!"

Geoffrey Hinton, inglês-canadense que obteve seu PhD em Inteligência Artificial na Universidade de Edimburgo em 1978, mudou-se para os EUA em 1980 para trabalhar no Carnegie Mellon University, e logo mudou-se para o Canadá em 1987 para dar aulas na Universidade de Toronto e colaborar com o CIFAR (Canadian Institute for Advanced Research), escreveu em 2006, junto

com Ruslan Salakhutdinov, um artigo denominado "*Reducing the Dimensionality of Data with Neural Networks*".

Nele, Hinton descreve um modo de treinar redes neurais profundas quando todo mundo pensava que era impossível otimizar esses modelos. Seu trabalho é considerado o ponto de reinício da revolução do aprendizado profundo de máquina.

Ainda em 2006, mas sem ligação direta, a fabricante de processadores Nvidia anunciava o CUDA (Compute Unified Device Architecture), uma plataforma de computação paralela acompanhada de um modelo de programação, que permitiria que desenvolvedores utilizassem as GPU's da Nvidia para outras aplicações que não jogos. O lançamento oficial do CUDA Toolkit 1.0 ocorreu em 2007, mas seus primeiros drivers beta e suporte público começaram a ser divulgados no final de 2006.

Antes do CUDA, as GPU's eram usadas quase que integralmente para o desenvolvimento gráfico, e a partir desse momento passariam a ser usadas para processar álgebra linear, operações com matrizes, simulações, equações físicas e químicas, biociência, astronomia e outras importantes aplicações.

Logo em 2006 e 2007, o projeto Folding@Home, da Universidade Stanford simulou o desdobramento de proteínas, que envolve o cálculo massivo de dinâmicas moleculares, e foi uma das primeiras aplicações não gráficas feitas com as GPU's da Nvidia, que colaborou com a universidade e ajudou na adaptação do software com o CUDA. O resultado foi um aumento de 20 a 30 vezes na velocidade de processamento quando comparado com simulações em CPU's. O CUDA foi uma revolução silenciosa nesse período, e muitas outras áreas passaram a utilizar as GPU's da Nvidia e sua plataforma antes da revolução de AI que se aproximava.

Nos anos a seguir, 2007 e 2008, graduandos da Universidade de Toronto que colaboravam com o grupo de Hinton na universidade, e que se tornaria mundialmente renomado por sua pesquisa pioneira em aprendizado profundo, iniciaram os testes com GPU's de jogos da Nvidia, inspirados nos modelos propostos no artigo de Hinton.

Já em 2009, Rajat Raina, Anand Madhavan e Andrew Ng publicaram, a partir de Stanford, o artigo "Large-scale Deep Unsupervised Learning Using Graphics Processors", no qual demonstraram a possibilidade de treinar grandes redes neurais de forma mais rápida e barata, sem depender exclusivamente de laboratórios cheios de servidores, utilizando GPUs comuns, originalmente associadas aos jogos. Ng teve como forte inspiração os trabalhos de Hinton.

Ainda em 2009, e já em 2010, os pesquisadores em IA e o CUDA se aproximaram definitivamente. Rajat Raina e seus coautores demonstraram kernels, isto é, funções paralelas de GPU, para a multiplicação de matrizes, o processamento de funções de ativação e de otimização, como *loss*, além de outras aplicações importantes.

Alguns anos depois, Adam Coates, Andrew Ng e coautores publicariam o artigo "Deep Learning with COTS HPC Systems", ampliando essa linha de trabalho ao demonstrar o uso de sistemas comerciais de alto desempenho para o treinamento de redes neurais profundas.

Em 2010, Yoshua Bengio, nascido na França e vivendo no Canadá desde jovem, doutor em ciência da computação pela McGill University, também conhecido como um dos pais do aprendizado profundo de máquina, liderou em seu laboratório o desenvolvimento do Theano, uma das primeiras bibliotecas open-source em Python voltadas à definição, otimização e execução eficiente de

expressões matemáticas com arrays multidimensionais. Com suporte a GPU por meio do CUDA, o Theano permitiu que pesquisadores treinassem redes neurais profundas com maior eficiência computacional, tornando-se uma ferramenta importante na fase inicial da retomada do deep learning, antes da chegada e consolidação de bibliotecas posteriores.

Bengio tinha uma relação direta com Hinton, com quem trabalhou em conjunto com Yann LeCun (os três são considerados os pais fundadores do deep learning), foi fundador do laboratório MILA (Montreal Institute for Learning Algorithms), e membro do grupo de elite financiado pelo CIFAR (Canadian Institute for Advanced Research).

Os dois grupos, em Stanford e no Canadá, são parte importante da "renascença global do deep learning", que ocorreu entre 2006 e 2012, após o já citado longo inverno. O que vamos explorar agora se baseia nos trabalhos de Yoshua Bengio e sua equipe.

A matemática nos modelos de NNLM

Se antes os modelos n-gram tinham severas limitações em razão de só contar as ocorrências das palavras que ocorriam antes de "certa palavra", agora os pesquisadores começariam a tentar atribuir mais "características" a cada palavra.

Se o objetivo era trabalhar como o cérebro, que consegue atribuir uma série de usos e contextos para cada palavra, o esforço continuava nessa mesma direção. A direção imaginada por McCulloch e Pitts.

Essas características atribuídas a cada palavra foram denominadas dimensões. Imagine um conjunto de 100 textos (textos científicos,

livros, publicações online) com uma média de 1000 palavras cada, somando 100.000 palavras. Agora imagine que nessas 100.000 palavras, apenas 1000 são distintas. Intuímos que cada uma delas se repete uma média de 100 vezes, correto?

Pode ser que as palavras "eu", "você", "casa", "acordou", "falou" se repitam com mais frequência que "lastimou", "determinou", ou mesmo "orvalho" e "alpendre". Era esse o desafio que passava a ser cumprido com álgebra linear e cálculo, deixando de lado aquela contagem pura e simples dos modelos n-gram. Em que situações as palavras "você", ou "casa", eram usadas? Com que frequência e acompanhadas de que outras palavras?

Voltando ao nosso exemplo, vamos ao passo a passo matemático. Se temos um conjunto de textos para treinamento (corpus) com 100.000 palavras e 1000 delas distintas, nosso vocabulário é igual a 1000 (V = 1000).

Já nas dimensões, o modelo passava a atribuir valores a cada uma das dimensões, criando relações de proximidade. Por exemplo: "casa", "cozinha", "cama" poderiam receber valores parecidos em alguma dimensão, a fim de que o modelo posicionasse essas palavras próximas umas das outras em um certo espaço, tudo com números. Algo chamado espaço vetorial.

Vamos determinar que nosso vocabulário terá 20 características para cada palavra, ou seja, 20 dimensões. Agora temos uma matriz 1000x20 chamada *embedding matrix*, sendo cada linha, de um a mil, atribuída a uma palavra, e cada coluna atribuída a uma dimensão. Essas características são registradas como números, que podem ser negativos ou positivos a cada passo do treinamento do nosso modelo.

Para começar, o modelo colocará valores randômicos, perto de zero, em cada uma das dimensões. Imagine 0,01, -0,01, 0,001, -0,001, e assim por diante. Dessa forma a dimensão não tem valor efetivo, mas tem um começo para corrigir e alterar na evolução do processo. Começar com valor nulo dificultaria o funcionamento do processo porque o modelo não teria nenhum "chute inicial" para aprender.

Se todas as palavras começarem exatamente iguais (com zero em todas as dimensões), os neurônios do modelo vão receber sempre as mesmas combinações no começo, e não terão como saber qual peso ajustar para cada palavra.

Antes de montar nosso modelo neural, estruturaremos o corpus em sequências de treinamento, de modo semelhante ao que era feito nos modelos N-gram, mas agora com a finalidade de alimentar uma rede neural. Vamos determinar que faremos uma janela de 5 palavras consecutivas, ou seja, para cada trecho, as quatro primeiras palavras formarão o contexto anterior, e a quinta palavra será aquela que estamos tentando prever, isto é, o alvo (*target*). Com isso, temos um corpo de treino estruturado.

Mesmo que você queira treinar sobre "gato", o modelo só verá "gato" como alvo nas janelas em que essa palavra aparecer como a próxima palavra a ser prevista depois de um contexto anterior.

Para colocar o modelo para rodar, temos que definir uma janela de contexto (*context window*). Se queremos prever a próxima palavra, vamos observar as palavras anteriores. Nesse caso, acabamos de determinar uma janela de contexto igual a 4.

Vamos também representar a palavra-alvo por meio de um vetor *one-hot*, isto é, uma representação em que apenas a posição cor-

respondente à palavra correta recebe valor 1, enquanto as demais recebem valor 0.

Cada vez que dispararmos o modelo, ele vai olhar para quatro palavras anteriores. Assim funciona o modelo na perspectiva do NNLM: prever a próxima palavra a partir de um contexto anterior fixo.

Se temos um corpus com 100.000 palavras e definimos uma janela de contexto igual a 4, cada posição do corpus, a partir da quinta palavra, pode gerar um exemplo de treino: quatro palavras anteriores como contexto e uma próxima palavra como alvo. Assim, o conjunto total de treino pode chegar a cerca de 99.996 exemplos formados por contexto e alvo. As repetições são mantidas de propósito, pois permitem que o modelo aprenda as frequências reais de ocorrência e reproduza padrões estatísticos naturais da linguagem.

Agora vamos olhar para alguns números do treinamento. Queremos uma aplicação que saiba qual a próxima palavra provável, dado um contexto. Claro, nosso modelo só conhecerá o vocabulário e os textos que treinou, mas dentro desse universo, ele será mais rápido e eficiente do que qualquer grupo de humanos.

O treinamento do modelo de linguagem NNLM consiste em processar esses exemplos em partes, chamadas de lotes (batches). A cada lote, ele compara o que previu com o que deveria ter previsto, calcula os erros e ajusta seus parâmetros (os famosos weights e bias). Esses novos parâmetros já entram em ação no processamento do próximo lote, e o processo de autocorreção acontece assim, passo a passo. Quando termina todos os lotes, ele completou uma época (epoch). Ele repete as épocas várias vezes até aprender bem.

Vamos supor que usamos batches de 128 exemplos. Teremos

aproximadamente 782 lotes para nossos 99.996 exemplos de treino. Quando terminarmos de rodar todos, teremos completado uma época. Faremos 10 épocas para garantir que o modelo compreendeu os diferentes contextos possíveis.

Voltando à nossa matriz do vocabulário (1000x20), digamos que a palavra "gato" esteja na linha 185. O modelo, ao buscar a palavra "gato", traz as 20 dimensões associadas a ela. Temos uma matriz 1x20. Mas lembre-se: a entrada do modelo, nesse caso, são as 4 palavras anteriores do contexto, cada uma com suas 20 dimensões. Isso gera uma matriz 4x20, que podemos organizar como uma linha de 80 números, matriz 1x80. Essa linha é ordenada segundo a posição das palavras anteriores no contexto.

Como treinamos 128 exemplos por vez, teremos uma matriz de entrada com 128 linhas, cada uma com 80 números — ou seja, 128x80. Essa matriz entra no nosso modelo.

Agora vamos montar o modelo. Temos uma camada de entrada (input layer) com 80 valores por exemplo, ou seja, 128 exemplos com 80 valores cada. Vamos construir uma rede neural com uma camada oculta com 128 neurônios, e cada um desses neurônios vai aplicar seus próprios pesos sobre os mesmos 80 valores de todos os exemplos do lote. Em outras palavras, cada neurônio aplica sua própria interpretação dos 80 valores de cada exemplo. Ao final do processamento dos 128 exemplos, completamos um lote.

O que esses neurônios fazem? Cada um aplica uma função chamada de ativação. Primeiro, eles multiplicam os 80 valores por um conjunto de números (os W, de weights), um para cada valor. Cada neurônio tem sua interpretação dos 80 valores, e os pesos W aplicados a cada um deles são o seu jeito de interpretar.

Esses pesos ajudam o neurônio a focar em certos padrões, como palavras relacionadas a animais, sentimentos, transportes, ainda que os significados não estejam claramente definidos nem mesmo para os pesquisadores. O que sabemos com precisão é que esses valores posicionam as palavras em locais em um espaço multidimensional.

Para se ter uma ideia dos padrões entrópicos dos modelos, se resetarmos todos os pesos e reinicializarmos o mesmo modelo com o mesmo corpus de treino, os pesos finais (W e b) após 10 épocas serão diferentes do modelo anterior, e as posições das palavras nesse espaço multidimensional também serão diferentes. Ainda assim, palavras com significados ou contextos próximos estarão novamente agrupadas em regiões similares.

Depois de multiplicar as dimensões com os pesos W, somava-se tudo e adicionam um valor extra, chamado bias. O bias permite mais flexibilidade no funcionamento dos neurônios, ajustando melhor os resultados. Se os valores ainda estiverem muito próximos de zero depois da aplicação dos pesos W, o bias entra em cena para "acordar" o neurônio, adicionando um empurrãozinho final.

Ao longo do processo, lote após lote, o bias vai ganhando mais importância: se os pesos W aprendem a relação entre os valores de entrada, o bias aprende a calibrar a relevância do resultado final dessa combinação. Existem casos em que o bias permanece próximo de zero. E isso também é aprendizado. Significa que, para aquele neurônio, o conjunto de entradas não é relevante o suficiente para justificar uma ativação frequente. O bias, nesse caso, está dizendo: "essa combinação não é comigo, não sou especializado nesse conjunto de palavras".

Após isso, aplicava-se a função de ativação ReLU (Rectified Linear

Unit), que zera valores negativos e mantém os positivos como estão. Essa função é essencial porque ela evita que o modelo fique preso em respostas lineares demais. Ao manter os valores positivos passando direto, a ReLU permite que os neurônios "acesos" continuem contribuindo com força, e ao zerar os negativos, impede ruído desnecessário.

Mais do que normalizar, ela ajuda a rede a produzir respostas não lineares, ou seja, que a rede tenha variedade de respostas, e reduz, em muitos casos, o problema do desaparecimento dos gradientes, permitindo que o modelo continue aprendendo com mais eficiência.

Outros modelos, como a sigmoide, comprimem os valores negativos e positivos para que caibam entre 0 e 1, o que limita o poder de aprendizado por achatar diferenças e matar o gradiente.

Cada neurônio da camada oculta gera então um número, que é resultado da aplicação da função de ativação, e esse número é enviado para os neurônios de saída, um para cada palavra do nosso vocabulário. São 1000 neurônios de saída.

Apresento-lhes a função de ativação:

$$z = \sum x \cdot w + b \qquad \text{e} \qquad a = \text{ReLU}(z)$$
$$\text{ReLU}(x) = \max(0, x)$$

Ou seja, retorna 0 se x < 0 e retorna x se x ≥ 0. Depois que os 128 neurônios da camada oculta enviam seus valores a, esses números chegam aos 1000 neurônios de saída, cada um representando uma palavra do nosso vocabulário.

Nenhum deles sabe, de cara, se deve levantar a mão. Cada neurô-

nio pega os 128 valores da camada anterior e faz seu próprio cálculo: aplica um conjunto exclusivo de pesos (um W próprio para cada valor de a) e soma um pequeno ajuste, o bias. Esse cálculo gera um número bruto, sem filtro, chamado *logit*, um placar interno. Um neurônio pode dizer: "para mim, esse contexto vale 2,3"; outro, "-1,5"; outro, "0". Esse conjunto de mil notas, positivas ou negativas, são os *logits*.

Ainda não são probabilidades, pois não existe probabilidade negativa, mas é a partir deles que o modelo decide quais palavras fazem sentido.

Perceba um detalhe importante: esses pesos e bias não vieram prontos de fábrica. O próprio modelo os cria na largada, com valores aleatórios bem próximos de zero. A cada lote de treino, ele compara o que previu com a palavra que deveria prever (a resposta está no corpus). Se errou, ajusta os números. Lote após lote, época após época, esses ajustes vão afinando o ouvido de cada neurônio de saída: ele aprende a reconhecer os sinais que combinam com "a palavra dele", e a ignorar o resto.

No fim das contas, cada neurônio de saída vira especialista em contextos específicos, ou seja, aprende a responder com mais força a certos padrões de contexto, geralmente aqueles que envolvem, de algum modo, as probabilidades da palavra que representa. Quando o contexto bate com o que ele aprendeu, o logit sobe. Se não bate, o logit despenca.

Depois dessa etapa, uma função chamada Softmax transforma esse painel de mil logits em probabilidades que somam 1, e temos a lista das palavras mais prováveis para continuar a frase.

A função Softmax nasceu fora do vocabulário contemporâneo da

IA, mas se liga a tradições anteriores da estatística, da termodinâmica estatística e dos modelos de escolha discreta. Na econometria, a família dos modelos logit multinomiais foi formalizada por Daniel McFadden nos anos 1970, contribuição que lhe renderia, em 2000, o Prêmio Nobel de Ciências Econômicas. Na aprendizagem de máquina, a Softmax passou a ser usada como forma de transformar saídas brutas de uma rede neural em uma distribuição de probabilidades sobre múltiplas classes.

A transposição para a aprendizagem de máquina veio nos anos 1980, quando David Rumelhart, Geoffrey Hinton e Ronald Williams precisavam de uma função que produzisse valores normalizados entre 0 e 1, representasse probabilidades relativas entre múltiplas saídas e fosse diferenciável, requisito essencial para o backpropagation.

Agora vamos para uma beleza matemática. Os neurônios de saída receberam logits (valores brutos) positivos e negativos. Precisamos transformá-los em probabilidades. Para isso, vamos normalizar. E como esse processo será usado dentro do treinamento logo adiante (com gradiente e correção de erro), precisaremos aplicar derivadas.

Aqui entram dois truques maravilhosos. Primeiro: não existe exponenciação cujo resultado seja negativo. Mesmo que o expoente seja negativo, o que ocorre é uma inversão, ele passa para o denominador. O valor de Euler elevado a um número negativo vira um número positivo muito pequeno. Em outras palavras: valores baixos continuam existindo, mas com menos força. Segundo, o número de Euler tem uma beleza própria, sua derivada é igual à própria função. Isso facilita muito o trabalho quando o modelo precisar ajustar seus pesos com base no erro.

E o que o Softmax faz, finalmente? Ele exponencia Euler ao valor de cada logit individual, e depois divide esse valor pela soma de todos os exponenciais calculados a partir de Euler elevado aos logits.

É como se dissesse:

"Vamos transformar esses palpites brutos em probabilidades justas, onde ninguém fica com valor negativo, todo mundo é positivo, e a chance de cada um é proporcional ao quanto gritou no placar."

Formalmente:

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

O resultado final é uma lista com todas as palavras do vocabulário e suas respectivas chances de aparecerem após aquele conjunto de palavras que usamos como entrada. Aplicações como o autocompletar de e-mails usam exatamente esse tipo de estrutura para sugerir palavras conforme você digita.

Mas como sabemos se o modelo acertou? Lá atrás, quando aplicamos o n-gram, estruturamos as combinações reais do nosso corpus, e é a palavra que realmente apareceu naquele contexto que serve como gabarito, o nosso target. Se o modelo não acertar o target, ele calcula a diferença entre o que previu e o que deveria prever. Esse cálculo se chama loss.

É a partir daqui que começa o processo de autocorreção que fará efeito no próximo lote. Logo em seguida, é usada uma técnica chamada backpropagation para ajustar os valores de W (peso) e b (bias), refazendo o caminho ao contrário, ou seja, voltando ca-

mada por camada na rede neural, corrigindo os erros com base no erro final. Como funciona o cálculo de loss? Ele compara duas coisas: o que o modelo previu, ou seja, a lista de probabilidades que saiu do Softmax, e o que realmente deveria acontecer, a palavra certa que estava no nosso corpus.

Esses dois elementos têm o mesmo formato: listas com mil posições (supondo um vocabulário de mil palavras). A saída do modelo tem vários números entre 0 e 1, todos somando 1, como uma distribuição de probabilidades.

Já a resposta correta vinda do corpus de treino é uma lista cheia de zeros, com um único valor 1 na posição da palavra certa. É como se dissesse: "é essa, e nenhuma outra".

A técnica matemática usada para fazer essa comparação se chama Cross-Entropy. Ela não olha apenas se o modelo acertou ou errou. Ela mede o quanto o modelo estava confiante no chute errado. Se o modelo errou com força, ou seja, atribuiu probabilidade muito baixa à palavra correta, por exemplo 0,01, a punição é alta. Se atribuiu probabilidade alta à palavra correta, ou seja, errou pouco, como 0,89 ou 0,99, a punição é baixa. A Cross-Entropy, portanto, não mede apenas se o modelo acertou ou errou, ela mede o quanto a distribuição prevista se afastou da resposta correta.

Errar com convicção, dando baixa probabilidade, pesa mais do que errar com hesitação. A Cross-Entropy está ali para isso: ensinar o modelo não só a acertar, mas também a errar com menos teimosia. A função Cross-Entropy entra em cena justamente para se aproveitar dos logaritmos, que são a inversa da exponencial.

E por que isso faz sentido agora? Porque todos os nossos números, tanto os que saem do Softmax quanto os 0 ou 1 que vêm do

corpus, estão sempre entre 0 e 1. Nessa faixa, o log natural (base de Euler) tem um comportamento precioso: quanto menor for o número, mais negativo (e em módulo, "maior") fica o log. Quanto mais próximo de 1, mais perto de zero ele se mantém. Em outras palavras, se o Softmax deu uma probabilidade baixinha para a palavra correta, digamos, 0,01, o log (0,01) resulta em algo como –4,6. Um puxão de orelha gigantesco. Se o modelo estava quase certo e pôs 0,99 na palavra certa, o log (0,99) resulta em –0,01, uma punição leve. E se cravou 1, perfeito: log (1) é igual a zero. Nenhuma punição.

A Cross-Entropy então pega esse log, multiplica pelo 1 que marca a posição da palavra certa (e por 0 nas posições erradas), aplica o sinal negativo para transformar o valor em positivo, soma, e entrega um número que mostra exatamente o quão "arrogante" foi o erro, ou, no caso do acerto, o quão confiante e certeiro foi o palpite.

Agora é hora de corrigir os valores de W e bias dos 128 neurônios da hidden layer. A Cross-Entropy já nos entregou um número: o tamanho do erro. Mas a mágica não é só saber o que errou. É saber onde errou, quanto errou, e como voltar atrás. Entra em cena o processo conhecido como backpropagation. Backpropagation é exatamente o que o nome diz: uma propagação para trás. A rede começa do final, dos neurônios de saída, e vai recuando camada por camada, revisando os passos que a levaram ao erro.

Lembra do placar que os neurônios de saída deram, os tais logits? Pois bem: agora a rede olha para eles com outros olhos e pergunta: "quais desses valores mais contribuíram para a burrada que acabei de cometer?". A resposta vem via cálculo de derivada. Mais precisamente, da derivada da função de perda com respeito aos pesos W e aos bias b de cada camada.

Lembra da nossa função de ativação ReLU? Aquela que deixa os positivos passarem e zera os negativos? Pois é. Agora ela mostra outro lado: sua derivada é simples, limpa, binária. Ela diz: "se passou por mim positivo, posso te mostrar o quanto você ajudou no erro; se fui zero, não me responsabilize".

Com esse diagnóstico na mão, o modelo então volta à camada oculta, aquela dos 128 neurônios, e começa a corrigir peso por peso, bias por bias. Os valores de W e b são ajustados usando um algoritmo chamado Gradiente Descendente. Ele basicamente empurra os parâmetros na direção oposta ao erro, tentando descer a montanha do prejuízo. Descer a montanha em razão de seu ponto ótimo ser no chão, levando em conta que ele realmente percorre uma topografia multidimensional com vales e picos. Valores errados estão nos picos, valores corretos mais próximos dos vales.

O conjunto de valores que se transformou em um escalar colocou o resultado final longe da palavra que deveríamos ter previsto, o valor target. Esse "longe" é como um ponto alto, descolado do lugar ideal que o modelo queria atingir. Imagine um tecido multidimensional, com altos e baixos, vincos e dobras, uma superfície de erro. O ponto ideal está em alguma parte plana, no "piso" desse relevo.

O que o gradiente faz é justamente isso: calcula a primeira derivada do loss com relação aos pesos, ou seja, como o erro muda se você mexer levemente em cada peso do modelo. Mesmo que o loss pareça um número só, ele é, na verdade, uma função oculta de todos os W e b.

A derivada mostra a direção de maior subida, e o gradiente descendente faz o oposto: aponta pra baixo, na direção onde o erro vai diminuindo. É como se a rede tivesse uma bússola interna dizendo

"desça por aqui, por essa dobra do tecido, que o chão está logo ali". Quando a rede erra, queremos saber não só quanto errou, mas para onde seguir para errar menos. É aí que entra a primeira derivada da função de perda, que é, na prática, uma maneira de descobrir a inclinação local do erro em relação a cada peso que usamos no cálculo.

Em outras palavras, é como se dissesse: "se você mexer esse peso um pouquinho pra cá, o erro sobe"; "pra lá, o erro desce". É isso mesmo, usamos derivada, uma estrutura matemática presente nos primeiros passos em cálculo analítico.

Se estivermos num gráfico 2D, essa inclinação seria um ângulo. Mas aqui estamos num espaço com centenas de dimensões, cada uma correspondente a um peso ou bias. Ainda assim, a lógica é a mesma, e calcularemos a derivada da perda (loss) em relação a cada um dos pesos: a derivada mostra a direção de maior crescimento do erro. E como não queremos subir, queremos descer, seguimos na direção oposta. Pronto: temos o gradiente descendente.

O gradiente nada mais é do que um vetor com todas essas derivadas. Um para cada peso W e para cada bias b.

$$\nabla L(W, b) = \left[\frac{\partial L}{\partial W_1}, \ \frac{\partial L}{\partial W_2}, \ \ldots, \ \frac{\partial L}{\partial b_n} \right]$$

Com isso, sabemos para onde o erro aponta em cada dimensão. Mas falta saber quanto andar. A resposta vem de um valor que escolhemos antes: a taxa de aprendizado, representada pela letra grega η (eta).

Ela regula o tamanho do passo. Se for muito pequena, o modelo

aprende devagar. Se for muito grande, ele pode passar direto do ponto ideal, como alguém descendo uma ladeira e tropeçando na própria pressa.

A atualização dos pesos e bias é feita com:

$$W_{\text{novo}} = W_{\text{atual}} - \eta \cdot \frac{\partial L}{\partial W}$$

e

$$b_{\text{novo}} = b_{\text{atual}} - \eta \cdot \frac{\partial L}{\partial b}$$

A taxa de aprendizado pode, e deve, ser calibrada. Se ela for muito pequena, o modelo ajusta os pesos com passos minúsculos. Isso até pode garantir precisão, mas exige muito mais épocas para aprender, o que aumenta o custo computacional.

Por outro lado, se exageramos e colocamos uma taxa muito alta, o modelo altera os pesos W e b com agressividade demais, fazendo com que eles "passem direto" do ponto ideal. Resultado: o modelo até chega rápido, mas não para no lugar certo. Corre, mas aprende mal.

Assim que o modelo atualiza os pesos W e b ao final do primeiro lote, inicia-se o segundo, repetindo todas essas etapas, do cálculo dos logits à correção pelo gradiente, para um novo conjunto de combinações. Esse processo se repete ao longo dos aproximadamente 782 lotes do corpus, encerrando o que chamamos de uma época.

Realizando 10 épocas (neste exemplo), teremos um modelo trei-

nado. Se no início ele poderia sugerir "roda" depois de "gato", ao final das 10 épocas esse tipo de erro já não passa despercebido: os pesos foram ajustados, o modelo afinou sua escuta e está agora muito mais preciso.

O raciocínio e a modelagem matemática fundamentais de função de perda (loss) e gradiente descendente permanecem os mesmos até os modelos atuais no momento que este texto foi escrito (Novembro de 2025), sob o paradigma "Transformers", inclusive nos modelos de ponta como GPT-4, GPT-5 ou PaLM-2.

Para se ter uma ideia, com uma GPU moderna, cada lote no nosso modelo NNLM de Bengio levaria aproximadamente 5 milissegundos para ser processado. Como temos aproximadamente 782 lotes, uma época levaria cerca de 3,9 segundos e as 10 épocas levariam aproximadamente 39 segundos. Em uma CPU comum, dependendo da implementação, o mesmo processo poderia levar dezenas de minutos

Aplicações comerciais do modelo de Yoshua Bengio

As aplicações comerciais à época foram simples, mas impactantes: corretores ortográficos inteligentes, sugestões de palavras em motores de busca, autocompletar de e-mails e sistemas de recomendação básica.

Embora os modelos NNLM a partir de Bengio ainda carregassem algumas limitações dos n-grams, como janelas de contexto fixas e treinamento computacionalmente caro, suas diferenças operacionais, especialmente a capacidade de aprendizado distribuído por embeddings e o uso de gradientes, permitiram avanços práticos antes impensáveis. Esses modelos foram adotados com sucesso por grandes empresas

de tecnologia.

A Microsoft, por exemplo, integrou variantes de NNLM em seus sistemas de correção automática no Word e Outlook, melhorando sugestões de escrita e gramática.

A Google aplicou modelos neurais de linguagem em produtos como Smart Reply, que sugeria respostas curtas no Gmail, e Smart Compose, que completava frases em tempo real com base no contexto. Esses produtos pertencem à linhagem técnica aberta pelos modelos neurais de linguagem, embora já empregassem arquiteturas, otimizações e infraestruturas posteriores às formulações originais de Bengio.

Esses casos não apenas demonstraram que era possível usar modelos neurais de linguagem em produção real, mas também provaram que eles ofereciam desempenho superior. Mais preciso, mais adaptável, mais "natural". Foi o suficiente para destravar o interesse comercial e acadêmico pela linguagem neural.

Estava aberta, enfim, a primavera da Inteligência Artificial.

Referências bibliográficas

BENGIO, Yoshua; DUCHARME, Réjean; VINCENT, Pascal; JANVIN, Christian. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, v. 3, p. 1137-1155, 2003.

BRIDLE, John S. Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition. In: SOULIÉ, Françoise Fogelman; HÉRAULT, Jeanny. Neurocomputing: Algorithms, Architectures

and Applications. Berlin: Springer, 1990. p. 227-236.

CHEN, Mia Xu; LEE, Benjamin N.; BANSAL, Gagan; CAO, Yuan; ZHANG, Shuyuan; LU, Justin; TSAY, Jackie; WANG, Yinan; DAI, Andrew M.; CHEN, Zhifeng; SOHN, Timothy; WU, Yonghui. Gmail Smart Compose: Real-Time Assisted Writing. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Anchorage: ACM, 2019. p. 2287-2295.

COATES, Adam; HUVAL, Brody; WANG, Tao; WU, David; NG, Andrew Y.; CATANZARO, Bryan. Deep Learning with COTS HPC Systems. In: *Proceedings of the 30th International Conference on Machine Learning*. Atlanta: PMLR, 2013. p. 1337-1345.

HENDERSON, Matthew; AL-RFOU, Rami; STROPE, Brian; SUNG, Yun-Hsuan; LUKÁCS, László; GUO, Ruiqi; KUMAR, Sanjiv; MIKLOS, Balint; KURZWEIL, Ray. Efficient Natural Language Response Suggestion for Smart Reply. *arXiv preprint*, arXiv:1705.00652, 2017.

HINTON, Geoffrey E.; RUMELHART, David E.; WILLIAMS, Ronald J. Learning Representations by Back-Propagating Errors. *Nature*, v. 323, p. 533-536, 1986.

HINTON, Geoffrey E.; SALAKHUTDINOV, Ruslan R. Reducing the Dimensionality of Data with Neural Networks. *Science*, v. 313, n. 5786, p. 504-507, 2006.

KANNAN, Anjuli; KURACH, Karol; RAVI, Sujith; KAUFMANN, Tobias; TOMKINS, Andrew; MIKLOS, Balint; CORRADO, Greg; LUKÁCS, László; GANEA, Marina; YOU-

NG, Peter; RAMAVAJJALA, Vivek. Smart Reply: Automated Response Suggestion for Email. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Francisco: ACM, 2016. p. 955-964.

LUCE, R. Duncan. Individual Choice Behavior: A Theoretical Analysis. New York: Wiley, 1959.

McFADDEN, Daniel. Conditional Logit Analysis of Qualitative Choice Behavior. In: ZAREMBKA, Paul. Frontiers in Econometrics. New York: Academic Press, 1974. p. 105-142.

NVIDIA CORPORATION. NVIDIA CUDA Compute Unified Device Architecture: Programming Guide, Version 1.0. Santa Clara: NVIDIA Corporation, 2007.

RAINA, Rajat; MADHAVAN, Anand; NG, Andrew Y. Large--Scale Deep Unsupervised Learning Using Graphics Processors. In: *Proceedings of the 26th International Conference on Machine Learning*. Montreal: ACM, 2009. p. 873-880.

RUMELHART, David E.; HINTON, Geoffrey E.; WILLIAMS, Ronald J. Learning Internal Representations by Error Propagation. In: RUMELHART, David E.; McCLELLAND, James L. Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations. Cambridge, MA: MIT Press, 1986. p. 318-362.

THE THEANO DEVELOPMENT TEAM. Theano: A Python Framework for Fast Computation of Mathematical Expressions. *arXiv preprint*, arXiv:1605.02688, 2016.

VINCENT, Pascal; LAROCHELLE, Hugo; BENGIO, Yoshua; MANZAGOL, Pierre-Antoine. Extracting and Composing Robust Features with Denoising Autoencoders. In: *Proceedings of the 25th International Conference on Machine Learning*. Helsinki: ACM, 2008. p. 1096-1103.

CAPÍTULO 6

2013-2014: A ERA DOS EMBEDDING MODELS - Word2Vec e GloVe.

O modelo de Bengio, mesmo aplicado comercialmente, ainda tinha um desafio a ser superado: seu alto custo computacional.

Modelos aplicados a grandes corpus, com vocabulários com milhares de palavras com, por exemplo, 300 dimensões, criavam matrizes *embedding* gigantes, e os loops de atualização dos erros (loss e backpropagation) tomavam centenas de horas.

Recorda-se da iniciativa de digitalizar a produção humana de textos? Ela não parava, e os modelos de IA para linguagem precisavam ser escaláveis.

Um treinamento com 1 milhão de pares de contexto-palavra, cada um representado por quatro vetores de 300 dimensões, equivaleria, em uma arquitetura baseada na concatenação desses vetores, a entradas de 1.200 dimensões. Em escala, isso significaria processar uma matriz de entrada aproximada de 1.000.000 × 1.200, antes mesmo das multiplicações pelas matrizes de pesos, do cálculo da saída e da atualização dos parâmetros por *backpropagation*.

A concatenação dessas informações em vetores, a atualização dos pesos (W e b) e o cálculo do Softmax na camada de saída, ao final de cada lote, tornavam-se algo em escala proibitiva para o amplo uso dos modelos.

Nesse contexto emerge Tomas Mikolov, tcheco nascido em 1982 com PhD em Ciência da Computação pela Universidade de Brno, completado em 2010. Sua tese de PhD versava sobre modelos estatísticos de linguagem para redes neurais. Tomas foi trabalhar na Microsoft em 2011 como pesquisador no time especializado em fala e, em 2012, migrou para o Google, onde foi o autor principal de dois artigos fundamentais: *Efficient Estimation of Word Repre-*

sentations in Vector Space e *Distributed Representations of Words and Phrases and their Compositionality.*

O primeiro apresentou as arquiteturas centrais do Word2Vec, especialmente o CBOW e o Skip-Gram. O segundo aprofundou essas soluções e consolidou técnicas de treinamento mais eficientes, entre elas o *negative sampling*, que veremos adiante. Juntos, esses trabalhos formaram a base técnica do Word2Vec no Google e marcaram mais um grande passo na evolução dos modelos de linguagem.

O que Tomas e seus colegas fizeram foi aplicar simplificações matemáticas elegantes e eficientes no modelo de Bengio. Mikolov foi buscar inspiração em princípios matemáticos aplicados à linguística pelo menos 60 anos antes. A ideia parece simples, mas é carregada de sofisticação: palavras viajam em grupos.

Em 1957, John Rupert Firth escreveu um artigo denominado *"You shall know a word by the company it keeps."* Afirmava que o significado de uma palavra é determinado pelo contexto linguístico em que aparece, ou seja, que palavras não carregam significado isoladas, mas em relação a outras palavras.

Firth era professor de linguística geral na Universidade de Londres, e concentrava seus estudos em colocação (palavras que aparecem frequentemente juntas) e relações sintagmáticas (sequências lineares na fala ou no texto), para o que importa neste texto. De forma simplificada, Firth afirmava que certas palavras andam sempre próximas de outras certas palavras, e que essa companhia recorrente é observável nos usos concretos da linguagem. Esses padrões podem se manter relativamente estáveis por longos períodos, às vezes por décadas, ainda que o léxico seja adaptado pelas gerações e seu uso sofra alterações criativas ao longo do tempo.

Ainda assim, é possível afirmar que "gato" estará frequentemente perto de "miau" e "leite", enquanto dificilmente será relacionado com "furadeira". Pode ocorrer, mas a raridade dessa ocorrência reduz sua probabilidade no uso cotidiano. O que Firth estava afirmando em seu artigo era uma ideia em que vinha trabalhando por toda sua vida, porém em termos semânticos, com menos formalidade.

Do outro lado do oceano, nos Estados Unidos, estava Zellig Harris, que já foi mencionado neste conjunto de texto como mentor de Noam Chomsky, e que pensava da mesma forma. Harris se concentrou em uma estrutura analítica mais formal, matemática, observando regularidades sintáticas e distribucionais da linguagem. Seu livro *Methods in Structural Linguistics*, publicado em 1951, foi uma referência importante dessa abordagem.

Poucos anos depois, no artigo "Distributional Structure", de 1954, Harris formularia de modo ainda mais direto a ideia de que elementos linguísticos podem ser compreendidos pelas posições e contextos em que aparecem.

O que ambos propuseram, e que provavelmente iluminou Tomas Mikolov para que simplificasse os modelos linguísticos de Inteligência Artificial, é que palavras basicamente estão sempre nas redondezas de outras palavras na grande maioria do tempo. Por consequência, os esforços matemáticos de precisão nos diversos parâmetros de cada palavra, como ocorria no modelo de Bengio, poderiam ser parcialmente substituídos por uma aposta estatística mais simples: observar ocorrências em grande escala e deixar que a repetição dos contextos revelasse as relações mais prováveis.

Embora Zellig Harris tenha publicado seu artigo antes, era contemporâneo de Firth, que veio a falecer três anos depois de sua

obra seminal, em 1960.

A matemática por trás do modelo Word2Vec de Mikolov

Mikolov propôs dois modelos, ambos com simplificações de cunho estatístico.

O primeiro, CBOW (Continuous Bag of Words), tratava a palavra central como alvo (target) e usava as palavras ao redor (duas antes, duas depois) como contexto. A ordem dessas palavras agora deixava de importar: o que interessava era o grupo.

Em vez de tratar os vetores de cada palavra de contexto separadamente, ou concatená-los em uma entrada longa, o modelo somava as dimensões correspondentes (coluna 1 de todas, depois coluna 2, e assim por diante) e fazia uma média simples. A busca por padrões estatísticos substituía a tentativa de parametrizar tudo, uma escolha que refletia um aprendizado histórico da humanidade.

O CBOW também inovava ao simplificar a camada escondida (Hidden Layer), tornando-a linear e sem função de ativação. A média vetorial, obtida a partir da matriz de embeddings W, era então multiplicada pela matriz de pesos W', gerando scores (*logits*) para cada palavra do vocabulário.

Daí em diante, o restante do processamento, ativação, cálculo de erro e ajustes, se tornava semelhante ao seu segundo modelo, o Skip-Gram, que acabou tendo maior impacto prático em razão de melhorias. Se o CBOW priorizava a máxima simplificação e, por isso, tinha dificuldade em capturar combinações raras de palavras, aproximando-se do problema de sparsity, o Skip-Gram enfrentaria esse ponto com outra estratégia.

A primeira alteração do modelo Skip-Gram, e provavelmente a mais sensível, foi deixar para trás a ideia da palavra central ser o target, o que precisava ser aprendido em razão das palavras do entorno, que era predominante nos modelos contemporâneos de NNLM e no CBOW. Agora a palavra central era o input, e a janela de contexto dava o tom de quais palavras "costumavam" acompanhar a palavra central, e quantas vezes (probabilístico novamente).

Se determinássemos *context window* igual a 2, o modelo varreria o corpus, palavra por palavra, lote a lote, época a época, determinando quais eram as palavras que costumavam acompanhar a palavra central. E como isso ocorria?

Para cada palavra do corpus, o modelo fazia pares de combinação. A frase "o gato pulou no telhado", encontrada casualmente pelo modelo no corpus enquanto realizava os lotes e épocas, era combinada em pares assim:

("O", "gato"); ("O", "pulou"); ("gato", "O"); ("gato", "pulou"); ("gato", "no"); ("pulou", "O"); ("pulou", "gato"); ("pulou", "no"); ("pulou", "telhado"); ("no", "gato"); ("no", "pulou"); ("no", "telhado"); ("telhado", "pulou"); ("telhado", "no").

O modelo continha duas matrizes de *embeddings*, uma para os vetores de input (W) e outra para os de output (W'), o que, em outras palavras, quer dizer: cada palavra no vocabulário possuía dois vetores distintos, representando como se comportava quando era central (input) e quando era uma das palavras de contexto (output) de qualquer outra.

Para o par ("gato", "O"), com "gato" como palavra central, o modelo fazia o produto escalar entre o vetor de "gato" na matriz W e o vetor

de "O" na matriz W'. Desse produto escalar surgia um valor, o logit, que representava o grau de afinidade entre essas duas palavras. Esse logit era então passado por uma função sigmoide, que o transformava em uma probabilidade entre 0 e 1. Quanto mais próxima de 1, maior a expectativa de que aquele par ocorresse com frequência no contexto real.

Para cada um desses pares reais (positivos), o modelo também criava alguns pares falsos, ou negativos, escolhendo palavras aleatórias do vocabulário. Nesses casos, o par recebia rótulo 0, e o modelo aprendia a empurrar sua probabilidade para baixo.

Perceba que, se nos modelos anteriores, como o NNLM, o Softmax aplicava uma probabilidade na camada de saída sobre todas as palavras do vocabulário, agora o Skip-Gram, quando treinado com negative sampling, se contentava, usando sigmoide em vez do Softmax completo, em comparar cada par real com meia dúzia de pares inventados.

O Softmax em sua condição integral é um monstro probabilístico, preciso, detalhista e caro computacionalmente, mas Mikolov enxergou outras possibilidades a partir da teoria distribucional e da estatística de coocorrência. Era a mágica do *negative sampling*: o modelo aprendia com poucos pares por vez, mas em muitos ciclos, entendendo aos poucos quem costumava andar com quem na linguagem humana.

Com essa probabilidade calculada do par provável e dos pares negativos, o modelo então media seu próprio erro. Se o par era verdadeiro, o erro (ou perda, loss) era calculado como –log(p). Se o par era negativo, o erro era –log(1 – p). Essa estratégia, chamada de *binary cross-entropy*, dava ao modelo uma bússola matemática: aproximar os vetores das palavras que de fato aparecem juntas, e

afastar os vetores das que não se relacionam.

Esse erro servia de guia para o processo de retropropagação (*backpropagation*), que ajustava os vetores envolvidos. A cada iteração, os vetores de entrada (quando a palavra era central) e de saída (quando a palavra era de contexto) eram levemente ajustados para que os pares reais se tornassem mais prováveis e os pares falsos menos prováveis.

Esse "ajuste leve", aliás, era controlado por uma variável chamada *learning rate* (taxa de aprendizado) que determinava o tamanho do passo dado em direção à correção. Se esse valor fosse muito alto, o modelo poderia oscilar e não convergir, se fosse muito baixo, o aprendizado seria lento ou até ineficaz. O segredo estava em encontrar um equilíbrio para que os vetores evoluíssem de forma estável, sem dar saltos bruscos.

Com o tempo, esse processo organizava o vocabulário inteiro em um espaço vetorial onde as relações semânticas e sintáticas entre palavras emergiam naturalmente. Tudo isso a partir de simples estatísticas de coocorrência, multiplicações vetoriais e algumas boas ideias.

O segundo artigo de Mikolov também explorava técnicas complementares, como a redução do peso de palavras excessivamente frequentes, como *the*, *of*, e *and*, que aparecem demais, mas carregam pouca informação semântica isolada. Também tratava certas expressões compostas, como *New York* ou *machine learning*, como unidades mais estáveis. Esses pontos eram importantes para melhorar a qualidade prática dos vetores, mas o salto central já estava dado: transformar palavras em representações matemáticas reutilizáveis.

Funcionamento do Skip-Gram com Negative Sampling – Passo a Passo

a) Corpus de texto:
Um grande conjunto de textos é usado como base de aprendizado.

b) Janela de contexto (context window):
Define quantas palavras antes e depois da palavra central serão consideradas. Ex: janela = 2, o modelo pega até 2 palavras para cada lado.

c) Criação dos pares reais (positivos):
A cada palavra central (input), são gerados pares com suas vizinhas na janela.

Ex: Na frase "o gato pulou no telhado", com janela = 2, a palavra "pulou" gera: ("pulou", "gato"), ("pulou", "o"), ("pulou", "no"), ("pulou", "telhado").

d) Criação de pares falsos (negativos):
Para cada par verdadeiro, o modelo sorteia algumas palavras aleatórias do vocabulário e forma pares negativos, tratados naquele passo de treino como exemplos de rótulo 0.
Ex: ("pulou", "ônibus"), ("pulou", "chuva"), etc.

e) Vetores de representação (embeddings):
Cada palavra tem dois vetores:
Um na matriz W (input): representa a palavra como central.
Um na matriz W' (output): representa a palavra como contexto.

A partir daqui, o modelo realizava cálculos para cada par, positivo ou negativo.

1. Logit, ou pontuação bruta: produto escalar entre os vetores da palavra central e da palavra de contexto.

$$\text{logit} = v_{\text{input}} \cdot v_{\text{output}}$$

2. Sigmoide, ou probabilidade: transforma o *logit* em uma probabilidade entre 0 e 1.

$$p = \frac{1}{1 + \exp(-\text{logit})}$$

3. Função de perda (Loss)
Se o par é real $(y = 1)$:

$$\text{Loss} = -\log(p)$$

Se o par é negativo $(y = 0)$:

$$\text{Loss} = -\log(1 - p)$$

Na forma completa da função de perda, apresentada abaixo, ocorre um efeito matematicamente elegante: quando o valor de y alterna binariamente entre 0 e 1, uma das metades da equação é anulada automaticamente em cada caso. Essa forma unificada permite calcular a perda para pares positivos e negativos ao mesmo tempo.

$$\text{Loss} = -\left[y \log(p) + (1 - y) \log(1 - p)\right]$$

4. Gradiente em relação ao vetor da palavra central

$$\frac{\partial \text{Loss}}{\partial v_w} = (p - y)\, v_c'$$

5. Gradiente em relação ao vetor da palavra de contexto

$$\frac{\partial \text{Loss}}{\partial v'_c} = (p - y)\, v_w$$

6. Atualizações com taxa de aprendizado η (learning rate)

$$v_w \leftarrow v_w - \eta(p - y)v'_c$$
$$v'_c \leftarrow v'_c - \eta(p - y)v_w$$

Em que v_w é o vetor da palavra central, v'_c é o vetor da palavra de contexto, p é a probabilidade estimada pela sigmoide, e y é o rótulo do par: 1 para par real e 0 para par negativo.

Na prática, essas duas atualizações usam os valores dos vetores antes da correção daquele passo, para que uma atualização não contamine a outra.

Vale lembrar que essa estrutura de cálculo da perda, amplamente utilizada em modelos de linguagem, deriva da função de verossimilhança da estatística clássica, especialmente quando aplicamos o logaritmo e passamos a trabalhar com a log-verossimilhança negativa. A aplicação do logaritmo a essa função foi sistematizada por Ronald Fisher, em 1922, em seu artigo *"On the Mathematical Foundations of Theoretical Statistics"*.

Décadas mais tarde, Claude Shannon, já citado neste editorial, aplicaria uma base formal semelhante em sua Teoria da Informação, ao desenvolver a fórmula da entropia, que mede a incerteza média de uma distribuição. Essa mesma linguagem matemática, baseada em probabilidades e logaritmos, ecoaria depois na *cross--entropy*, usada atualmente no aprendizado de máquina para me-

dir o afastamento entre a previsão do modelo e a resposta esperada no treinamento, isto é, a distribuição observada no corpus.

O modelo GloVe de Pennington, Socher e Manning

Logo depois que Mikolov e sua equipe no Google introduziram o Word2Vec, Pennington, Socher e Manning, em Stanford em 2014, trouxeram o seu modelo, denominado GloVe.

O GloVe oferecia uma abordagem estatística, voltando às origens da contagem de coocorrências, e que diferia do Word2Vec no modelo de treinamento.

Seu modelo, introduzido no artigo "GloVe: Global Vectors for Word Representation", simplesmente contava as ocorrências dos pares de palavras por um modelo simplificado de leitura do corpus chamado janela deslizante (sliding window), e usava esses dados para otimizar vetores em duas matrizes distintas do vocabulário: uma em que as palavras eram consideradas inputs, e outra em que eram consideradas outputs.

Aplicava o log da quantidade de ocorrências do par como referência estatística, usava um procedimento de ponderação para que pares raros fossem percebidos sem dominar o cálculo e para que pares muito frequentes não pesassem demais, realizava um *dot product* das duas matrizes por cada par, somava um *bias* de input e um de output, e comparava esse valor final com o log da contagem real de coocorrência.

Esse erro, por sua vez, era penalizado por uma função de ponderação que limitava a influência de pares extremamente raros ou muito frequentes, gerando uma perda (loss) suave e controlada. O

modelo, então, ajustava os vetores por *backpropagation*, buscando alinhar o produto escalar ao log da coocorrência, sem deixar que outliers dominassem o aprendizado.

A grande diferença entre o Word2Vec e o GloVe era o modo como aprendiam as representações das palavras.

Enquanto o Word2Vec apostava em um modelo preditivo, tentando adivinhar palavras do contexto a partir de uma palavra central (ou o contrário), o GloVe preferia uma abordagem mais estatística e direta: contar, pesar e ajustar.

No Word2Vec, os vetores vão sendo ajustados com base em acertos e erros nas previsões durante o treinamento. Já no GloVe, a ideia era partir daquilo que já se sabe, a frequência com que os pares de palavras aparecem juntos (estatística) e usar isso como base para definir a relação entre os vetores. Um "conta, ajusta e calibra", o outro "tenta prever e erra até acertar" (probabilidade). Ambos os modelos geravam vetores úteis, mas o caminho até eles era bem diferente.

A matemática usada no modelo GloVe

Equação de predição: produto escalar + vieses

$$w_i \cdot w_j + b_i + b_j \approx \log(X_{ij})$$

Aqui, w_i é o vetor da palavra principal, $\tilde{w}_j$ é o vetor da palavra de contexto, b_i e $\tilde{b}_j$ são os vieses, e X_{ij} representa a quantidade de vezes que a palavra j apareceu no contexto da palavra i.

Função de ponderação: peso aplicado ao erro

$$f(X_{ij}) = \begin{cases} \left(\frac{X_{ij}}{x_{\max}}\right)^{\alpha}, & \text{se} X_{ij} < x_{\max} \\ 1, & \text{se} X_{ij} \geq x_{\max} \end{cases}$$

Essa função controla o peso de cada par de palavras no treinamento: pares muito raros entram com peso menor, e pares muito frequentes deixam de crescer indefinidamente em importância.

Função de custo: perda a ser minimizada

$$J = \sum_{i,j} f(X_{ij}) \left(w_i \cdot w_j + b_i + b_j - \log(X_{ij})\right)^2$$

Essa função mede o quanto o produto escalar entre os vetores, somado aos vieses, se afasta do logaritmo da coocorrência observada. O treinamento ajusta os vetores para reduzir esse erro.

O papel dos modelos de embedding vetoriais no mundo prático

O que o nome Word2Vec dizia era simples e direto: vamos transformar tudo em vetores. Sem gramática, sem regras sintáticas, sem categorias semânticas explícitas. Apenas coocorrências estatísticas, retiradas de corpus gigantescos.

O modelo de Stanford, o GloVe, propunha a mesma lógica, embora com um viés mais estatístico e estrutural. Ambos representaram uma ruptura com a geração anterior de modelos, como o NNLM, que tentava modelar a distribuição da linguagem como um todo.

O NNLM olhava para todas as combinações possíveis de palavras no vocabulário, atribuía pesos e probabilidades para cada inferência e ajustava esses parâmetros continuamente. Isso tornava o modelo pesado demais para aplicações em escala industrial: treinamentos que levavam semanas e exigiam enorme capacidade computacional.

Foi nesse cenário que o Word2Vec trouxe uma revolução. Com a introdução dos modelos Skip-Gram e CBOW, além de técnicas como *negative sampling* e *hierarchical softmax*, tornou-se possível treinar *embeddings* de forma leve e rápida, em horas ou até minutos. A ideia não era mais prever sentenças completas, mas extrair representações vetoriais eficientes das palavras. Pela primeira vez, os *embeddings* deixaram de estar acoplados ao modelo de linguagem e passaram a ser reutilizáveis: representações pré-treinadas e independentes.

Isso abriu caminho para o transfer learning na NLP, um dos maiores saltos conceituais que a área veria nos anos seguintes.
E os vetores funcionavam. Word2Vec mostrou que capturava regularidades semânticas e sintáticas com incrível elegância: subtrair "homem" de "rei" e somar "mulher" levava, de forma vetorial, à ideia de "rainha".

As palavras passaram a ser posicionadas em um espaço contínuo com base na similaridade de contexto, uma representação estatística da linguagem que era, ao mesmo tempo, simples e poderosa.

O Word2Vec acabou por democratizar a NLP: era leve, fácil de implementar e seus vetores podiam ser aplicados diretamente a inúmeros produtos, pesquisas e aplicações reais. Foi adotado em larga escala quase da noite para o dia, e foi amplamente utilizado, muito em razão de estar internalizado no Google, o que facilitou sua

integração em diversos produtos e pipelines da empresa. Sua eficiência computacional e simplicidade também contribuíram para sua ampla adoção na indústria.

Já o GloVe teve uso mais concentrado em contextos acadêmicos e projetos de pesquisa, sendo valorizado por sua base estatística sólida e pela qualidade semântica dos vetores gerados.

Embora ambos os modelos tenham sido amplamente reconhecidos, o Word2Vec acabou tendo maior projeção prática e disseminação por estar associado a uma infraestrutura poderosa e amplamente acessível desde cedo.

Aplicações comerciais do Word2Vec

Motores de busca (Google, Bing, etc.)
O Word2Vec ajudou motores de busca a incorporar relações semânticas entre palavras, e não apenas sua presença literal. Assim, se alguém buscava por "médico de olhos", o sistema podia aproximar essa expressão de termos relacionados, como "oftalmologista". Além disso, embeddings desse tipo podiam melhorar funções de autocompletar, correção e sugestões do tipo "você quis dizer...".

Recomendação de produtos (e-commerce, varejo, mídia)
Em recomendação de produtos, varejo, mídia e entretenimento, empresas como Amazon, Netflix e Spotify passaram a usar modelos vetoriais e embeddings para representar produtos, filmes, músicas, playlists ou comportamentos de usuários em espaços matemáticos comparáveis. Em alguns casos, a lógica era diretamente inspirada no Word2Vec: se palavras que aparecem nos mesmos contextos podem ser aproximadas vetorialmente, produtos comprados juntos, músicas ouvidas em sequência ou filmes consumidos por perfis

semelhantes também poderiam ser aproximados em espaços vetoriais. O Spotify, por exemplo, chegou a treinar modelos do tipo Word2Vec sobre playlists geradas por usuários; em outros casos, como Amazon e Netflix, a lógica evoluiu para arquiteturas próprias de embeddings e recomendação em larga escala.

Atendimento ao cliente e chatbots
Em atendimento ao cliente e em chatbots, embeddings ajudavam a aproximar expressões diferentes com sentido parecido, como "fatura atrasada", "boleto vencido" ou "não paguei ainda", permitindo que sistemas tratassem variações linguísticas como manifestações próximas de uma mesma intenção.

Referências bibliográficas

BARKAN, Oren; KOENIGSTEIN, Noam. Item2Vec: Neural Item Embedding for Collaborative Filtering. *arXiv preprint*, arXiv:1603.04259, 2016.

BENGIO, Yoshua; DUCHARME, Réjean; VINCENT, Pascal; JANVIN, Christian. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, v. 3, p. 1137-1155, 2003.

FIRTH, John Rupert. A Synopsis of Linguistic Theory, 1930-1955. In: FIRTH, J. R. et al. Studies in Linguistic Analysis. Oxford: Philological Society, 1957. p. 1-32.

FISHER, Ronald A. On the Mathematical Foundations of Theoretical Statistics. *Philosophical Transactions of the Royal Society of London. Series A*, v. 222, n. 594-604, p. 309-368, 1922.

HARRIS, Zellig S. Methods in Structural Linguistics. Chicago: University of Chicago Press, 1951.

HARRIS, Zellig S. Distributional Structure. Word, v. 10, n. 2-3, p. 146-162, 1954.

LEVY, Omer; GOLDBERG, Yoav. Neural Word Embedding as Implicit Matrix Factorization. In: *Advances in Neural Information Processing Systems*, v. 27, 2014.

MIKOLOV, Tomas; CHEN, Kai; CORRADO, Greg; DEAN, Jeffrey. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint*, arXiv:1301.3781, 2013.

MIKOLOV, Tomas; SUTSKEVER, Ilya; CHEN, Kai; CORRADO, Greg; DEAN, Jeffrey. Distributed Representations of Words and Phrases and their Compositionality. In: *Advances in Neural Information Processing Systems*, v. 26, 2013.

MIKOLOV, Tomas; YIH, Wen-tau; ZWEIG, Geoffrey. Linguistic Regularities in Continuous Space Word Representations. In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Atlanta: Association for Computational Linguistics, 2013. p. 746-751.

PENNINGTON, Jeffrey; SOCHER, Richard; MANNING, Christopher D. GloVe: Global Vectors for Word Representation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. Doha: Association for Computational Linguistics, 2014. p. 1532-1543.

SHANNON, Claude E. A Mathematical Theory of Communication. *The Bell System Technical Journal*, v. 27, p. 379-423; 623-656, 1948.

SPOTIFY RESEARCH. Contextual and Sequential User Embeddings for Music Recommendation. Spotify Research, 2021.

AMAZON SCIENCE. MERLIN: Multimodal & Multilingual Embedding for Recommendations at Large-Scale via Item Associations. Amazon Science, 2024.

NETFLIX RESEARCH. Recommendations. Netflix Research, s.d.

CAPÍTULO 7

2014: Seq2Seq E OS ESFORÇOS EM TRADUÇÃO.

Os esforços em IA iam muito bem nas aplicações computacionais que previam a próxima palavra. O modelo de Bengio havia trazido de volta a IA para o centro dos acontecimentos e o Word2Vec, com suas simplificações matemáticas, arquitetura mais leve e métodos que evitavam *backpropagations* custosos sobre todo o vocabulário a cada atualização, havia acelerado esse movimento.

No campo do processamento, as GPUs começavam a ganhar escala. Em 2014 e nos anos seguintes, passavam a ser aplicadas com crescente importância em modelos neurais de linguagem e tradução.

As variantes de GPU da fabricante NVIDIA – Tesla, Kepler, Maxwell – eram utilizadas nos modelos linguístico-matemáticos com bastante eficácia, e foram elementos centrais para o crescimento da capacidade computacional e consequente sofisticação dos modelos de IA.

Somado a isso, entre os anos de 2012 e 2014, a chegada e a popularização de frameworks de *deep learning* – algoritmos voltados à construção e treinamento de modelos, por assim dizer – como Theano e Caffe, possibilitaram que mais pessoas treinassem redes neurais, o que seria de grande importância para o desenvolvimento dessas tecnologias em um ambiente aberto e de compartilhamento de conhecimento.

Logo depois veríamos a chegada de novos frameworks de deep learning, como TensorFlow, lançado pelo Google em 2015, e PyTorch, desenvolvido a partir do ambiente Facebook AI e lançado no ano seguinte.

A computação paralela, que, em termos simples, consiste em dividir uma tarefa entre múltiplos processadores para execução simultânea, começava a se consolidar como abordagem promissora.

Embora não fosse aplicada diretamente aos modelos de tradução discutidos neste capítulo, vale mencioná-la para ilustrar como o ambiente tecnológico evoluía em ritmo sem precedentes, chegando a ultrapassar, em certos aspectos, a própria lógica da Lei de Moore.

Se as aplicações para prever a próxima palavra eram normalmente utilizadas para um idioma somente, os esforços em tradução idiomática nunca deixaram de existir. Eram urgentes e já contavam com bons benchmarks.

Como já foi demonstrado em capítulos anteriores, o uso governamental e militar vinha impulsionando as aplicações de tradução desde os anos 1950, pelo menos. Ou seja, a turma da tradução nunca parou. Seja em razão da guerra, seja pela paixão dos linguistas pelas estruturas de linguagem, ou mesmo pelo desafio probabilístico dos matemáticos, estático ou dinâmico, a sala dos tradutores esteve cheia todo esse tempo.

As aplicações de tradução eram, até agora, baseadas em regras gramaticais e de sintaxe bem definidas, previamente determinadas para os modelos, ou, com evoluções importantes, baseadas em sistemas estatísticos.

No primeiro caso, o modelo dependia de regras escritas por especialistas: Dicionários, estruturas sintáticas, equivalências gramaticais e decisões linguísticas eram programadas previamente.

Exemplos dessas aplicações são o IBM Georgetown, de 1954, e o Systran, lançado em 1968. Este último foi financiado pelas forças

armadas americanas, usado na Guerra Fria e mais tarde pela NASA.

Em 1997, a AltaVista integrou essa aplicação ao Babel Fish, baseado no Systran, registrando uma das primeiras experiências comerciais de tradução automática para o grande público. A aplicação foi parar nas mãos do Yahoo pela aquisição da AltaVista e ficou amplamente conhecida como Yahoo Babel Fish. Para o usuário comum, a tradução automática aparecia ali como uma ferramenta de internet. Ainda rígida, muitas vezes literal e limitada, mas já visível fora dos ambientes militares, acadêmicos e institucionais.

No segundo caso, o avanço veio pelos sistemas estatísticos. Em vez de depender apenas de regras programadas previamente, esses modelos aprendiam padrões a partir de grandes conjuntos de textos já traduzidos. Se uma frase aparecia em inglês em um documento oficial canadense e sua versão correspondente aparecia em francês, o modelo podia comparar as duas versões e estimar, por probabilidade, quais palavras ou grupos de palavras em um idioma costumavam corresponder a palavras ou grupos de palavras no outro.

Era uma forma diferente de resolver o problema. O computador não precisava saber gramática como um linguista. Precisava observar recorrências. Ao encontrar milhares de vezes expressões como "the European Parliament" e "le Parlement européen", ou "the motion was adopted" e "la motion a été adoptée", o modelo começava a inferir correspondências estatísticas. Primeiro entre palavras, depois entre pequenos grupos de palavras, e finalmente entre frases mais prováveis.

O público sentiria essa fase principalmente com o Google Translate, lançado em 2006, já apoiado em tradução estatística. A experiência era diferente daquela do Babel Fish. A tradução deixava de

parecer apenas uma substituição mecânica de palavras por regras fixas e passava a parecer uma escolha probabilística entre formulações possíveis.

Portanto, antes da tradução neural, havia duas grandes famílias em operação: os modelos baseados em regras e os modelos estatísticos. As duas haviam produzido avanços importantes, tanto institucionais quanto comerciais, mas ainda esbarravam em uma dificuldade central: as duas linhagens de tradução demandavam uma habilidade que até então era um desafio matemático, já mencionado nesse conjunto de textos: a capacidade do modelo de preservar informações de contexto ao longo de frases longas.

Lembra-se da palavra central e as vizinhas, duas a menos, duas a mais? Esses modelos eram incapazes de trabalhar com frases longas, com, por exemplo, 10 palavras ou mais. Muito menos "compreender" e "aprender" a aplicação em contextos.

Colocado de outra forma, os modelos discutidos até aqui ainda não resolviam bem o problema das sequências longas. Os modelos de Bengio e Word2Vec tratavam de palavras, proximidade e representações vetoriais, mas não eram desenhados para preservar contexto ao longo de uma frase inteira, muito menos para transformar uma sequência em outro idioma, palavra por palavra.

Redes recorrentes já existiam e já operavam com dados sequenciais. O problema é que, ao lidar com frases longas, sofriam para manter a informação relevante ao longo de muitos passos temporais. A sequência era processada, mas o contexto se degradava. Anos antes, em 1997, Sepp Hochreiter introduzira o conceito *Long Short-Term Memory (LSTM)*, que era capaz de "memorizar" longas sentenças e suas aplicações. Era uma tentativa de solucionar os problemas que redes recorrentes (RNN), tais como Elman

e Jordan, continuavam a sofrer ao lidar com frases longas.

Quando chegava a hora de ajustar os números internos, os famosos pesos, os RNNs precisavam "voltar no tempo" pela frase inteira, num processo chamado *backpropagation through time*, corrigindo os pesos de cada palavra da sequência (frase).

Nesse processo, os valores que viajavam para corrigir os pesos iam se multiplicando várias vezes, palavra após palavra. Em frases curtas o efeito exponencial não era rampante, mas em frases longas esses valores sumiam (ficavam tão pequenos que era como se fossem zero) ou explodiam (ficavam tão grandes que bagunçavam o cálculo).

O LSTM de Sepp Hochreiter mudou esse jogo. Ele colocou portões de controle, que funcionavam como filtros inteligentes, capazes de guardar o que importava e descartar o que era irrelevante em cada passo da frase. Com isso, mesmo em sentenças longas, o modelo conseguia lembrar o contexto e fazer os ajustes matemáticos de forma estável, sem o efeito de sumiço ou explosão dos números.

Do outro lado, na tradução (*decoder*), ele fazia um trabalho computacionalmente caro como o de Bengio, atribuindo probabilidades para todo o idioma de output (2º idioma) a fim de reconstruir a frase que vinha do idioma original.

Em vez de esquecer o que estava no início, o LSTM atualizava não só o vetor da palavra que estava sendo processada, mas também um vetor de estado, que ia acumulando o sentido da sequência inteira.

Isso era possível nesse momento em virtude de uma escolha na modelagem. Os ajustes não tratavam de atualizar os pesos (*em-*

bedding dimensions) das palavras da frase em relação à totalidade do mesmo vocabulário, como em Bengio, nem em compreender estatisticamente onde as palavras mais aparecem, lote a lote, até que houvesse uma razão estatisticamente inabalável do comportamento dessa palavra. O modelo tratava especificamente daquela frase no idioma de input.

O LSTM visava dotar certa sequência, uma frase retirada de um corpus, de uma identidade numérica multidimensional própria, sem precisar recalcular sua relação com todas as palavras do vocabulário a cada processamento. Essa identidade não eliminava os vetores das palavras isoladas. Era construída a partir deles, passo a passo, enquanto o modelo percorria a frase e atualizava seus estados internos. Ou seja, ao final do processamento, o vetor passava a representar a sequência inteira, e não apenas palavras consideradas isoladamente.

Esse vetor ganhava sentido no lado oposto do modelo, quando o *decoder* tentava reconstruir, no idioma B, a frase correspondente à sequência de entrada no idioma A. O treinamento ocorria sobre corpus paralelo, com sentenças equivalentes em idiomas diferentes.

A cada tentativa de reconstrução, o modelo comparava a frase gerada no idioma B com a tradução correta, calculava o erro e ajustava seus pesos. A correção não ficava apenas no *decoder*. Voltava também ao *encoder,* fazendo com que a representação numérica da frase original se tornasse cada vez mais útil para produzir a frase equivalente no outro idioma.
O modelo matemático no idioma B realizava parte do mesmo processo do idioma A, pois também trabalhava sequencialmente e atualizava seus estados internos a cada palavra. Mas fazia algo a mais em termos de sofisticação: em vez de apenas condensar a frase recebida, construía uma nova frase, palavra por palavra, usando o vetor recebido do *encoder*, o estado anterior do próprio *decoder* e

as palavras já produzidas.

No caso da tradução, isso era fundamental. O *encoder* lia a frase no idioma A e condensava seu sentido em um vetor. O *decoder* então usava esse vetor para gerar a frase correspondente no idioma B, palavra por palavra.

O modelo de Sepp tinha um custo computacional alto à época. Multidimensional, com ajustes complexos e com necessidade de memória, o que o manteve restrito a usos conceituais e acadêmicos por mais de uma década, com raras tentativas de uso comercial.

Isso mudaria no ano de 2014, quando a capacidade computacional havia superado certas fronteiras e a curva de aprendizado da indústria havia se inclinado positivamente de forma íngreme. Este ano traria acontecimentos que o tornariam mais um ponto de inflexão importante na história dos modelos de IA.

Nesse ano, Ilya Sutskever, Oriol Vinyals e Quoc V. Le, todos trabalhando no Google Brain, publicaram o artigo *"Sequence to Sequence Learning with Neural Networks"*, introduzindo uma poderosa arquitetura para tradução de máquina que se utilizava do modelo LSTM de Sepp.

Com o objetivo de contextualização histórica desse livro, vamos olhar novamente para as pessoas envolvidas nesses acontecimentos. Ilya nasceu em 1986 na antiga União Soviética. Mudou-se com a família para Israel, onde completou seus estudos médios e fez alguns cursos livres a distância na Universidade de Israel. Em 2002 sua família mudou-se para o Canadá, e Ilya ingressou na Universidade de Toronto. Ali conquistou seu bacharelado em Matemática, depois em Ciência da Computação e, finalmente, seu doutorado em Ciência da Computação. Importante citar que lá

Ilya cooperou com Geoffrey Hinton.

Ilya, Oriol e Quoc, de dentro do Google Brain, unidade do Google para estudos em inteligência artificial, lançaram nesse ano o Seq2Seq. Como o nome já diz, tratava-se de relacionamentos entre sequências.

Ainda em 2014, outro artigo trataria da mesma questão. No artigo "*Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation*", Cho et al. introduziram o GRU (Gated Recurrent Unit), uma versão simplificada do LSTM de Sepp. Simplificado em relação ao LSTM, o GRU apresentava resultados muito próximos, porém com mais aplicabilidade em razão de utilizar-se de menos parâmetros e ter mais rapidez.

Kyunghyun Cho nasceu na Coreia do Sul, obteve seu bacharelado em Ciência da Computação no KAIST (Instituto de Ciência e Tecnologia Avançada da Coreia), e mestrado e doutorado na Universidade Aalto, na Finlândia. Sua tese de doutorado foi nomeada "Fundamentos e avanços em Deep Learning".

Cho juntou-se à Universidade de Montreal como pesquisador em fase de pós-doutorado, onde conheceu ninguém menos que Yoshua Bengio (aquele que foi responsável pelo fim do inverno em IA). Já como professor de Cho, Bengio foi coautor do artigo sobre GRU, que viria a compor esse momento em que os modelos de IA ganhavam muita velocidade.

O modelo Seq2Seq do Google utilizava-se da aplicação de Ilya Sutskever, Oriol Vinyals e Quoc V. Le, baseada em LSTM. Seus pesquisadores perceberam também as vantagens do GRU de Cho, testaram e assimilaram o modelo, mas logo surgiria outro mecanismo matemático computacional que iniciaria uma verdadeira

revolução na capacidade de apreensão de contexto dos modelos de IA: os modelos de atenção.

A velocidade de desenvolvimento dos cálculos computacionais aplicados à linguagem já havia tomado, nesse momento, uma dimensão incomparável com qualquer outra fase da humanidade. Essa revolução não ocorreu isoladamente, e é importante trazer à luz outros movimentos importantes que ocorreram simultaneamente e que possibilitaram esse processo.

Esforços em tradução

O que se buscava em tradução era que o modelo conseguisse ler uma frase em um idioma e expressasse o mesmo sentido em outro. No Seq2Seq, o primeiro módulo (*encoder*) percorria a frase em inglês e condensava seu sentido em um vetor, uma espécie de "resumo matemático" daquela sentença.

Em seguida, o segundo módulo (*decoder*) começava a desempacotar esse vetor, gerando palavra por palavra em francês, por exemplo, até formar uma frase que carregasse o mesmo significado.

Nesse contexto ocorreram movimentos significativos, governamentais e privados: a criação de corpora paralelos de idiomas, métodos de medição e até campeonatos de performance de modelos. Um dos mais importantes foi a criação de corpora paralelos, ou seja, conjuntos em um idioma que estivessem emparelhados a outro idioma. Esses corpora vinham se desenvolvendo desde os anos 1980 e era usado no treinamento desses modelos.

Alguns tiveram papel central, como o Europarl, consolidado em 2005 pelo pesquisador Philipp Koehn, um dos nomes importan-

tes da tradução automática estatística. O Europarl era um corpus composto por transcrições do Parlamento Europeu, inicialmente organizado em 11 idiomas e depois expandido para 21, permitindo que pesquisadores comparassem frases equivalentes entre diferentes línguas europeias.

Assim foi também o Canadian Parliament, nos idiomas inglês e francês e um dos primeiros corpora paralelos modernos, o United Nations Parallel Corpus, com documentos oficiais traduzidos para 6 idiomas, o Opus Project, que agregava diversos corpora paralelos, menos formais, e que incluía filmes, softwares e sites multilíngues, e o WMT (workshop on machine translation), que não é um corpus único, mas um conjunto de dados liberados anualmente para os campeonatos WMT. Os campeonatos WMT faziam parte de um ecossistema de esforços em tradução.

Além do WMT, ocorriam o IWSLT, com foco em tradução de fala e tendo as Ted Talks traduzidas em diversos idiomas como um dos *datasets*, e o NIST MT.

Esses campeonatos de tradução automática foram cruciais para acelerar a evolução dos modelos entre 2005 e 2015. Eles funcionavam como pontos de encontro entre academia e indústria, onde equipes do mundo todo competiam para ver quem traduzia melhor com base em corpora paralelos padronizados.

A fim de padronizar as pontuações que mediam a performance desses modelos de tradução, pois antes de 2000 cada grupo de pesquisa media a qualidade da tradução de um jeito diferente, por avaliação humana, contagem de palavras corretas ou critérios próprios, a IBM propôs o BLEU, Bilingual Evaluation Understudy, em 2002. O BLEU comparava a tradução produzida pela máquina com traduções humanas de referência, medindo a sobre-

posição de palavras e pequenos grupos de palavras, os chamados n-gramas, além de penalizar traduções excessivamente curtas.

O BLEU foi rapidamente adotado como padrão de avaliação pelo NIST Machine Translation Evaluation, NIST OpenMT, e, pouco depois, tornou-se também a métrica de referência do WMT. Depois do BLEU vieram outros padrões de avaliação, como NIST, Meteor, TER, chrF e mais tarde Comet e Bleurt.

Essa era a base invisível, mas imprescindível, dos esforços em modelos probabilísticos simples e mais tarde neurais de tradução, e formou a estrutura para o compartilhamento de conhecimento além de fronteiras e idiomas, que estamos presenciando atualmente.

A matemática por trás do Google seq2seq

Foram dois os modelos RNN utilizados nesse período: LSTM e GRU. Aqui vamos explorar matematicamente o mais sofisticado, o LSTM utilizado pelo Google Brain no lançamento do Seq2Seq. Trata-se de uma arquitetura em que a sequência (frase) no primeiro idioma é processada e comprimida em um vetor por um módulo chamado *encoder.* Outro módulo, o *decoder*, descomprime esse vetor e começa a reconstruir a frase no segundo idioma, palavra por palavra, a partir de *logits* e aplicando *softmax* sobre todo o vocabulário do idioma 2.

A partir dos *logits* resultantes antes do *softmax* no idioma 2, o erro é calculado via *cross-entropy* e, pelo método de *gradient descent*, os ajustes necessários são aplicados no idioma 2 e enviados de volta ao *encoder*, que, por sua vez, altera seus pesos e as representações vetoriais das palavras na sequência de entrada (*input sentence*).

A grande inovação do LSTM está nos *gates* do *encoder.* São chamados portões por realizarem um processo na passagem do vetor de entrada, e controlam duas funções cruciais: a memória de curto prazo e a de longo prazo.

O modelo descrito aqui é treinado com um corpus paralelo, por exemplo, inglês e francês, contendo 20.000 pares de frases equivalentes nos dois idiomas. Cada par contém uma frase no idioma 1 e sua tradução correspondente no idioma 2. Seu objetivo não era identificar, dentro de uma lista, qual frase no idioma 2 correspondia a uma frase no idioma 1, mas aprender a gerar, palavra por palavra, uma frase no idioma 2 que preservasse o sentido da frase recebida no idioma 1.

Começamos com uma representação por *embeddings*, em que cada palavra ou token é representado por um vetor de 512 dimensões. Supondo que todas as frases no idioma 1 estejam representadas com comprimento fixo de 5 posições, T = 5, com uso de *padding* para frases menores, a estrutura inicial é uma matriz de formato (20.000 × 5).

No entanto, cada célula contém, na verdade, um vetor (1 × 512) representando o *embedding* do token. O conjunto completo forma um tensor 3D com formato (20.000, 5, 512): 20.000 frases, cada uma com 5 posições, e cada posição com um vetor de 512 dimensões.

O treinamento ocorre em lotes (*batches*). Por exemplo, usando um batch de 100 frases, o formato será (100, 5, 512). O processamento ocorre simultaneamente para todas as frases e percorre os tokens na ordem temporal: primeiro token de todas as frases (t = 0), depois o segundo (t = 1), até (t = 4).

Cada token (1 × 512) passa por multiplicações de matrizes com

três parâmetros fundamentais:

W – pesos ligados ao *input* atual, com formato $(m, n) = (512, 512)$ multiplicando o vetor de entrada do token (x_t) e transformando as dimensões do embedding nas dimensões da camada oculta;
U – pesos ligados ao hidden state anterior, carregando informações do token anterior, com formato $(n, n) = (512, 512)$, multiplicando h_{t-1} e permitindo que a rede carregue informação do passo anterior; b – *bias* com formato $(n,) = (512,)$, somado ao resultado das multiplicações para ajustar o deslocamento antes da função de ativação.

Na formulação moderna do LSTM, já com um portão específico de esquecimento, chamado *forget gate*, responsável por controlar quanto do estado de célula anterior deve ser preservado ou descartado, essas operações ocorrem separadamente para cada gate:

$i_t = \sigma(W_i \cdot x_t + U_i \cdot h_t^{-1} + b_i)$ - input gate
$f_t = \sigma(Wf \cdot x_t + Uf \cdot h_t^{-1} + bf)$ - forget gate
$\tilde{c}_t = tanh(Wc \cdot x_t + Uc \cdot h_{t-1} + bc)$ - célula candidata
$o_t = \sigma(Wo \cdot x_t + Uo \cdot h_t^{-1} + bo)$ - output gate

A partir desses valores, os estados de longo e curto prazo são atualizados por:

$$c_t = f_t \odot c_t^{-1} + i_t \odot \tilde{c}_t$$

$$h_t = o_t \odot tanh(c_t)$$

onde σ é a função sigmoide, $\tanh$ é a tangente hiperbólica e $\odot$ representa multiplicação elemento a elemento (Hadamard product).

Note que o hidden state ht é calculado a partir do hidden state

anterior $ht-1$, carregando a memória de curto prazo para o passo seguinte.

Já o cell state c_t é atualizado a partir do valor anterior $ct-1$, preservando a memória de longo prazo ao longo da sequência.

No início do decoder, $h0$ e $c0$ são inicializados, respectivamente, com hT e cT produzidos pelo encoder, o que garante que a tradução comece com todo o contexto processado da frase de entrada.

Na prática, como cada gate tem seu próprio conjunto de W, U e b, o LSTM mantém quatro grupos de parâmetros, que podem ser empacotados em matrizes únicas (4n representando a concatenação dos gates input, forget, candidata e output):

W com formato $(m, 4n) = (512, 2048)$
U com formato $(n, 4n) = (512, 2048)$
b com formato $(4n,) = (2048,)$

O modelo descrito é unidirecional (processa apenas de $t = 0$ a $t = T - 1$), tem tamanho de hidden state $n = 512$ igual ao *embedding size* $m = 512$, apenas uma camada LSTM e usa padding para sentenças menores que 5 tokens, com o símbolo <PAD>.

Depois que o *encoder* processa toda a frase, ele entrega ao *decoder* o par final (h_T, c_T), que se torna (h_0, c_0) do *decoder*. Este executa o mesmo mecanismo interno (mesmas equações de *gates*), mas na primeira entrada recebe um símbolo especial <SOS> (*start of sequence*).

Se o modelo usar *teacher forcing*, durante o treinamento a entrada do próximo passo no *decoder* não é a palavra prevista, mas a palavra correta do conjunto de treinamento, acelerando e estabilizando o aprendizado.

No modo livre (*free running*), o *decoder* usa como entrada a própria palavra gerada anteriormente, o que é mais próximo da inferência real, mas mais suscetível a erros acumulativos.

No *decoder*, a cada passo:

1. Recebe x_t, h_{t-1} e c_{t-1};
2. Passa pelas multiplicações W,U e $bias$ nos quatro *gates*;
3. Atualiza c_t e ht;
4. Aplica uma camada linear $(\text{projection layer})$ para gerar *logits* do tamanho do vocabulário de saída;
5. Aplica softmax para obter as probabilidades;
6. Escolhe a próxima palavra por uma estratégia de decodificação, como escolha da palavra mais provável, amostragem probabilística ou o beam search, muito usado em tradução automática por comparar várias hipóteses de frase.

Quando a frase de saída é concluída (ou um <EOS> é gerado), calcula-se o erro entre a frase prevista e a real por *cross-entropy*. Esse erro é propagado para trás (*Backpropagation Through Time* — BPTT), ajustando todos os pesos W, U e b do *encoder* e do *decoder*.

Esse ciclo se repete para cada *batch*, por várias épocas, até que o modelo aprenda a mapear frases do idioma 1 para o idioma 2 com precisão satisfatória.

O que podemos perceber aqui é a complexidade do modelo, com cálculos matriciais de grande dimensão no encoder, a complexidade do *softmax* e da cross entropy no decoder e o retorno com correções, agora sequenciado pelos passos temporais de processamento, por isso chamado BPTT.

O amadurecimento dos modelos matemáticos e a evolução da capacidade de processamento (seja pelos processadores ou por frameworks), além dos esforços paralelos em tradução, foram os responsáveis pelo salto qualitativo observado nos modelos de IA entre 2014 e 2016.

Usos comerciais do Seq2Seq

Em tradução automática (NMT – Neural Machine Translation), o Google Translate foi o caso mais emblemático.

Em 2016, o Google substituiu seu sistema estatístico baseado em frases, SMT, por uma arquitetura neural de tradução, o GNMT, Google Neural Machine Translation, baseada em Seq2Seq. A mudança é importante porque levou a lógica Seq2Seq para uma aplicação comercial de grande escala. O sistema continuava baseado na ideia central de transformar uma sequência de entrada em uma sequência de saída, mas já incorporava attention, um mecanismo que reduzia o gargalo do vetor único ao permitir que o decoder consultasse partes diferentes da frase de entrada durante a tradução, e que veremos em capítulos adiante.

O Baidu, a Microsoft (Bing Translator), e o Facebook AI também adotaram variantes do Seq2Seq com LSTM/GRU para tradução multilíngue. Para chatbots e assistentes virtuais, os modelos Seq2Seq foram usados para geração de respostas em sistemas de atendimento automatizado e assistentes como o Microsoft Xiaoice na China e versões iniciais de Google Assistant e Amazon Alexa para diálogos curtos. O Facebook Messenger usou frameworks baseados em Seq2Seq para o bot de FAQ e suporte ao cliente.

No caso de legendas automáticas e legendagem em tempo real,

um bom exemplo foi o YouTube Auto-Captions e os sistemas de legendas para videoconferências, que usaram o Seq2Seq para converter áudio em texto e, em seguida, fazer tradução ou ajuste gramatical.

O Seq2Seq foi usado também em plataformas internas de empresas para responder automaticamente a perguntas frequentes, com base em bases de conhecimento preexistentes, e em sistemas de geração de código e preenchimento automático anteriores à consolidação dos Transformers. Ferramentas como Kite ajudaram a popularizar a ideia de sugestão automática de código, embora a geração moderna em larga escala tenha se consolidado depois com modelos baseados em Transformers, como o Codex usado no GitHub Copilot. O modelo virou uma tecnologia pivô para qualquer tarefa que envolvesse mapear uma sequência de entrada para uma sequência de saída, e sua aplicação comercial foi ampla: tradução, atendimento automático, legendagem, resumo e até geração de código.

Referências bibliográficas

ABADI, Martín et al. TensorFlow: A System for Large Scale Machine Learning. In: OSDI 2016, 12th USENIX Symposium on Operating Systems Design and Implementation. Savannah: USENIX Association, 2016. p. 265–283.

BAHDANAU, Dzmitry; CHO, Kyunghyun; BENGIO, Yoshua. Neural Machine Translation by Jointly Learning to Align and Translate. In: International Conference on Learning Representations, ICLR 2015. San Diego: ICLR, 2015.

BANERJEE, Satanjeev; LAVIE, Alon. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation

with Human Judgments. In: ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization. Ann Arbor: Association for Computational Linguistics, 2005. p. 65–72.

BARRAULT, Loïc et al. Findings of the 2020 Conference on Machine Translation, WMT20. In: Proceedings of the Fifth Conference on Machine Translation. Online: Association for Computational Linguistics, 2020.

BASTIEN, Frédéric et al. Theano: New Features and Speed Improvements. In: NIPS 2012 Deep Learning Workshop. Lake Tahoe: Neural Information Processing Systems Foundation, 2012.
BENGIO, Yoshua et al. A Neural Probabilistic Language Model. Journal of Machine Learning Research, v. 3, p. 1137–1155, 2003.

BERGSTRA, James et al. Theano: A CPU and GPU Math Compiler in Python. In: Proceedings of the 9th Python in Science Conference, SciPy 2010. Austin: SciPy, 2010. p. 1–7.

BROWN, Peter F. et al. The Mathematics of Statistical Machine Translation: Parameter Estimation. Computational Linguistics, v. 19, n. 2, p. 263–311, 1993.

CHO, Kyunghyun et al. Learning Phrase Representations using RNN Encoder Decoder for Statistical Machine Translation. In: EMNLP 2014, Conference on Empirical Methods in Natural Language Processing. Doha: Association for Computational Linguistics, 2014. p. 1724–1734.

DODDINGTON, George. Automatic Evaluation of Machine Translation Quality Using N-gram Co occurrence Statistics. In: Human Language Technology Conference, HLT 2002. San Die-

go: Morgan Kaufmann, 2002.

FEDERICO, Marcello et al. Overview of the IWSLT 2012 Evaluation Campaign. In: Proceedings of the 9th International Workshop on Spoken Language Translation, IWSLT 2012. Hong Kong: IWSLT, 2012. p. 12–33.

GERS, Felix A.; SCHMIDHUBER, Jürgen; CUMMINS, Fred. Learning to Forget: Continual Prediction with LSTM. Neural Computation, v. 12, n. 10, p. 2451–2471, 2000.

HOCHREITER, Sepp; SCHMIDHUBER, Jürgen. Long Short Term Memory. Neural Computation, v. 9, n. 8, p. 1735–1780, 1997.
HUTCHINS, W. John. The Georgetown IBM Experiment Demonstrated in January 1954. In: AMTA 2004, Conference of the Association for Machine Translation in the Americas. Washington: AMTA, 2004.

JIA, Yangqing et al. Caffe: Convolutional Architecture for Fast Feature Embedding. In: ACM Multimedia 2014, Open Source Software Competition. Orlando: Association for Computing Machinery, 2014.

KOEHN, Philipp. Europarl: A Parallel Corpus for Statistical Machine Translation. In: MT Summit 2005. Phuket: Asia Pacific Association for Machine Translation, 2005. p. 79–86.

KOEHN, Philipp; OCH, Franz Josef; MARCU, Daniel. Statistical Phrase Based Translation. In: HLT NAACL 2003, Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics. Edmonton: Association for Computational Linguistics, 2003. p. 127–133.

KOEHN, Philipp et al. Moses: Open Source Toolkit for Statis-

tical Machine Translation. In: ACL 2007, 45th Annual Meeting of the Association for Computational Linguistics, Companion Volume, Proceedings of the Demo and Poster Sessions. Prague: Association for Computational Linguistics, 2007. p. 177–180.

LINGUISTIC DATA CONSORTIUM. Hansard French English. LDC95T20. Philadelphia: Linguistic Data Consortium, 1995.

MIKOLOV, Tomas et al. Efficient Estimation of Word Representations in Vector Space. arXiv preprint arXiv:1301.3781, 2013.

MIKOLOV, Tomas et al. Distributed Representations of Words and Phrases and their Compositionality. In: Advances in Neural Information Processing Systems 26, NeurIPS 2013. Lake Tahoe: Neural Information Processing Systems Foundation, 2013. p. 3111–3119.

OCH, Franz Josef. Statistical Machine Translation Live. Google Research Blog, 28 abr. 2006.

OCH, Franz Josef; NEY, Hermann. The Alignment Template Approach to Statistical Machine Translation. Computational Linguistics, v. 30, n. 4, p. 417–449, 2004.

PAPINENI, Kishore et al. BLEU: A Method for Automatic Evaluation of Machine Translation. In: ACL 2002, 40th Annual Meeting of the Association for Computational Linguistics. Philadelphia: Association for Computational Linguistics, 2002. p. 311–318.

PASZKE, Adam et al. Automatic Differentiation in PyTorch. In: NIPS 2017 Autodiff Workshop. Long Beach: Neural Information Processing Systems Foundation, 2017.

POPOVIĆ, Maja. chrF: Character n-gram F score for Automatic MT Evaluation. In: WMT 2015, Tenth Workshop on Statistical Machine Translation. Lisbon: Association for Computational Linguistics, 2015. p. 392–395.

REI, Ricardo et al. COMET: A Neural Framework for MT Evaluation. In: EMNLP 2020, Conference on Empirical Methods in Natural Language Processing. Online: Association for Computational Linguistics, 2020.

SELLAM, Thibault; DAS, Dipanjan; PARIKH, Ankur. BLEURT: Learning Robust Metrics for Text Generation. In: ACL 2020, 58th Annual Meeting of the Association for Computational Linguistics. Online: Association for Computational Linguistics, 2020. p. 7881–7892.

SLOCUM, Jonathan. Machine Translation: Its History, Current Status, and Future Prospects. Computational Linguistics, v. 11, n. 1, p. 1–17, 1985.

SNOVER, Matthew et al. A Study of Translation Edit Rate with Targeted Human Annotation. In: AMTA 2006, 7th Conference of the Association for Machine Translation in the Americas. Cambridge: AMTA, 2006. p. 223–231.

SUTSKEVER, Ilya; VINYALS, Oriol; LE, Quoc V. Sequence to Sequence Learning with Neural Networks. In: Advances in Neural Information Processing Systems 27, NeurIPS 2014. Montreal: Neural Information Processing Systems Foundation, 2014.

TIEDEMANN, Jörg. Parallel Data, Tools and Interfaces in OPUS. In: LREC 2012, Eighth International Conference on

Language Resources and Evaluation. Istanbul: European Language Resources Association, 2012.

WU, Yonghui et al. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. arXiv preprint arXiv:1609.08144, 2016.

YANG, Jin. SYSTRAN on AltaVista. In: AMTA 1998, Conference of the Association for Machine Translation in the Americas. Langhorne: AMTA, 1998.

ZIEMSKI, Michał; JUNCZYS DOWMUNT, Marcin; POULIQUEN, Bruno. The United Nations Parallel Corpus v1.0. In: LREC 2016, Tenth International Conference on Language Resources and Evaluation. Portorož: European Language Resources Association, 2016.

CAPÍTULO 8

2014 A 2015 - BAHDANAU, LUONG E O NASCIMENTO DO ATTENTION MODERNO EM TRADUÇÃO NEURAL.

Relembrando um ponto de inflexão importante – a tentativa de melhorar a capacidade dos mecanismos de tradução pelo Seq2Seq, quando as frases recebiam um conjunto de características encapsuladas em um vetor que poderia ser desempacotado em outro idioma – podemos denominar o próximo movimento do ecossistema de computação como o início da era da cognição de máquina, expressão cunhada por este autor.

Esse movimento foi marcado pela passagem do paradigma do processamento sequencial limitado das RNNs para uma forma de rede capaz de consultar seletivamente a informação conforme a necessidade.

O Seq2Seq tinha um gargalo. Seu vetor fixo, representado pelo estado final do encoder h_T, e no caso do LSTM também o estado de célula c_T, era entregue ao decoder como resumo da frase de entrada. Esse "pacote" de dimensões fixas (por exemplo, 512 dimensões numéricas) obrigava o modelo a comprimir toda a informação semântica e sintática em um único ponto de partida para o *decoder*.

Para frases longas e complexas, o vetor ficava sobrecarregado. Detalhes importantes, como advérbios, negações e relações de dependência sintática, se perdiam. Tentando reconstruir a frase de saída só a partir desse resumo comprimido, o resultado eram traduções truncadas, sem coerência ou com erros de concordância.

Esse obstáculo técnico, mais uma vez, foi a motivação para a próxima inovação nos modelos de IA.

Nesse contexto surge Dzmitry Bahdanau. Bielorusso, estudou matemática aplicada na Belarusian State University e obteve seu

mestrado em Ciência da Computação na Jacobs University em Bremen, na Alemanha, sob supervisão de Herbert Jaeger, conhecido por suas Echo State Networks (ESN). Mais tarde obteve seu PhD no MILA (Université de Montréal), sob orientação de ninguém menos que Yoshua Bengio, um dos pais do Deep Learning. Estando no MILA, Bahdanau se posicionou no núcleo que catalisou o renascimento das redes neurais.

Para fins de ilustração, o jovem Dzmitry foi medalha de ouro na IOI (International Olympiad in Informatics) em 2009, ocorrida na Suécia naquele ano, com o 13° lugar nos resultados individuais.

Bahdanau pensou: "E se, em vez de forçar toda a informação da frase em um único vetor, o *decoder* pudesse consultar as informações (*hidden states*) de cada palavra da frase do *encoder* quando precisar?" Foi assim que o *attention mechanism* se tornou uma peça central da tradução neural moderna. Em vez de depender só do último *hidden state* do *encoder* h_T o decoder olha para trás em todos os h_j e decide dinamicamente quais palavras são mais relevantes a cada passo.

O gargalo de compressão em um único vetor foi o problema que levou Dzmitry Bahdanau, Kyunghyun Cho e Yoshua Bengio a proporem o *attention* no artigo *Neural Machine Translation by Jointly Learning to Align and Translate*, submetido em 2014 e apresentado no ICLR em 2015. Cho era o mesmo pesquisador que já havia contribuído para o GRU e para o modelo RNN Encoder Decoder, ainda em 2014. Bahdanau e Cho trabalhavam no ambiente acadêmico de Bengio, a quem se credita a sugestão do nome *attention*. Tudo ocorria muito rápido.

A ideia central era atribuir a cada posição da frase no input/idioma 1 um peso de contexto, ou seja, uma medida de relevância

daquela palavra, já transformada em representação interna pelo *encoder,* dentro daquela sequência específica. Cada vez que o *decoder* trabalhasse para desempacotar os dados e prever a próxima palavra da sequência no output/idioma 2, ele poderia olhar para a relevância de cada posição da sequência que veio do *encoder*, isto é, para cada palavra já convertida em *hidden state*.

Era uma forma de entender a presença de palavras em diferentes combinações em que ela aparecia e quão importante essa palavra era em uma cada uma dessas sequências específicas.

Em outras palavras, se além de eu saber que você aparece, eu compreender a importância que você tem naquela frase, cercado por aquelas palavras, naquele momento da tradução, eu acabo sabendo quando devo prestar atenção em você e ao que está em sua volta.

Em seu modelo de atenção, chamado *additive attention*, Bahdanau sempre olhava globalmente, para todos os *hidden states* do *encoder*, acrescido de outros detalhes, como em que momento o contexto era observado. Bahdanau tinha um peso matemático e computacional grande, mas a proposição era revolucionária.

Olhando para as complexidades do modelo e seu custo computacional, outro grupo propôs melhorias no mecanismo de atenção. A partir do departamento de ciência da computação da Universidade de Stanford, os jovens Minh-Thang Luong e Hieu Pham, estudantes de PhD em Ciência da Computação e orientados pelo professor Christopher D. Manning, à época chefe do Stanford NLP Group, escreveram o artigo *Effective Approaches to Attention-based Neural Machine Translation* (2015).

Buscando simplificar o modelo de atenção, Luong deslocou o momento em que o cálculo do contexto entrava em ação. Se Bah-

danau gerava os scores (ou pesos de importância) de cada palavra antes de atualizar o estado do *decoder*, ou seja, o contexto já influenciava a atualização do *decoder*, Luong primeiro atualizava o estado do *decoder* e só depois gerava os *scores*, comparando esse estado com os *hidden states* do *encoder*.

Além disso, Bahdanau tinha um sistema mais pesado para calcular os scores das palavras da sequência, usando projeções lineares seguidas da função tangente hiperbólica. Por isso, seu modelo ficou conhecido como *additive attention*. Já Luong propôs formas mais simples para calcular os scores, como *dot product*, *general* e *concat*. As duas primeiras eram as formas mais claramente multiplicativas, pois usavam produto escalar, de modo direto ou com uma matriz treinável intermediária. Já *concat*, embora apresentada por Luong, aproximava-se mais da lógica aditiva de Bahdanau, pois juntava os vetores antes da projeção final. Ele também apresentou a ideia de o *decoder* olhar apenas para uma janela em torno de uma posição da sequência (*local attention*), diminuindo ainda mais o custo e forçando foco no modelo.

A comunidade de NLP passou a usar com frequência formas mais simples de *attention* em frameworks de código aberto porque eram mais rápidas. Embora o mérito conceitual permanecesse com Bahdanau, as formulações de Luong tiveram maior aderência prática.

Em resumo, se o Seq2Seq construía estruturas informacionais que carregavam a estrutura contextual de uma frase em um bloco (o vetor fixo do *encoder*), que seria desmontado e interpretado por um mecanismo "do outro lado", mas que trazia dificuldades para o lado interpretador em razão da compressão do bloco, Bahdanau, e depois Luong, criaram um modelo que permitia que o "lado interpretador" pudesse olhar para dentro do processo (consultando dinamicamen-

te os *hidden states* do *encoder*) e pegasse as informações que achasse mais importantes para facilitar a interpretação do bloco.

Os resultados foram traduções mais precisas, com maior fluidez e bem menos distorções, consolidando a atenção como um divisor de águas na evolução dos modelos de linguagem.

O modelo de *attention* foi criado especificamente para o uso em traduções, mas logo passou a ser aplicado em resumo automático, reconhecimento de fala, resposta a perguntas, e até legenda de imagens (neste último caso, em uma estrutura um pouco diferente). No exemplo do resumo automático, o modelo continuava operando com pares de treino. A diferença é que, em vez de ter uma sequência em um idioma e outra sequência em outro idioma, trabalhava-se no mesmo idioma: de um lado, textos ou frases longas, e de outro, suas versões resumidas. Cada par de treino associava uma sequência longa à sua sequência curta correspondente.

O *encoder* gerava os *hidden states* da entrada (frase) longa e, ao gerar o resumo, o *decoder* usava o *attention* para atribuir mais peso às posições da sequência, isto é, palavras já representadas pelo *encoder*, que ajudavam a produzir a versão resumida, deixando em segundo plano aquelas que eram menos úteis para transmitir o conceito. Dessa forma, frases ou textos longos eram convertidos em versões mais curtas.

O modelo não "sabia" que estava resumindo ou traduzindo. Ele aprendia a mapear uma sequência de entrada em uma sequência de saída, com o *attention* ajudando a selecionar, a cada passo, quais partes da entrada eram mais úteis para gerar a saída.

A matemática por trás do attention

Cada token da frase de entrada era inicialmente convertido em um vetor denso (um *embedding* de 512 dimensões em nosso exemplo), e processado sequencialmente por uma rede recorrente (LSTM ou GRU).

A cada passo temporal (t), essa rede recebia o vetor de entrada atual (x_t), combinava-o com o estado oculto anterior (h_{t-1}) e produzia um novo estado (ht), que encapsulava as informações acumuladas até aquele ponto da sequência. Assim, ao final do processamento de uma frase de comprimento (T), o *encoder* gerava uma série de vetores ($h_1, h_2, \ldots, h_T$), cada um representando um resumo contextual de sua posição.

No modelo Seq2Seq original, apenas o último estado, (hT), era transmitido ao *decoder*. Isso criava o chamado "gargalo do vetor fixo", pois todo o conteúdo semântico da sentença de entrada precisava ser comprimido em um único vetor.

O mecanismo de atenção, proposto por Bahdanau et al. (2014), surgiu justamente para contornar essa limitação, permitindo que o *decoder*, a cada passo, consultasse todos os *hidden states* do *encoder* e atribuísse diferentes graus de relevância a cada um deles.

No passo do decoder, o estado oculto anterior do *decoder* (s_{t-1}) oriundo de uma GRU no modelo original, era comparado com cada *hidden state* (h_j) produzido pelo *encoder*. O objetivo era calcular, para cada posição j, um escore de alinhamento ($e_{t,j}$), que indicava o quanto o token daquela posição da frase de entrada era relevante para a geração do próximo token de saída. Esse escore era definido por:

$$e_{t,j} = v_a^\top \tanh(W_s s_{t-1} + W_h h_j)$$

Nessa expressão, (W_s) e (W_h) eram matrizes de projeção que mapeavam o espaço do *decoder* e do *encoder* para uma dimensão comum (d_a), e (v) era um vetor que colapsava essa dimensão intermediária em um escalar.

A função $(\tanh)$ atuava como não linearidade suavizadora, comprimindo os valores no intervalo (-1,1). Isso limitava as ativações, ou seja, impedia que os valores intermediários crescessem indefinidamente dentro daquela etapa do cálculo, embora pudesse introduzir certa saturação de gradientes.

Uma vez calculados todos os escores $(e_{t,1}, e_{t,2}, \ldots, e_{t,T})$, eles eram normalizados por meio da função Softmax, transformando os escores em pesos de atenção $(\alpha_{t,j})$:

$$\alpha_{t,j} = \frac{\exp(e_{t,j})}{\sum_{k=1}^{T} \exp(e_{t,k})}$$

Esses pesos funcionavam como uma distribuição de probabilidade normalizada sobre os *hidden states* do *encoder*, indicando o quanto o *decoder* deveria "olhar" para cada posição da sentença de entrada ao gerar o próximo token. Assim, um valor alto de $(\alpha_{t,j})$ significava que a posição j contribuía fortemente para a tradução, ou predição, corrente, enquanto valores baixos indicavam menor influência.

Com os pesos definidos, o próximo passo era calcular o vetor de contexto (c_t), que representava uma síntese ponderada das informações do *encoder* mais relevantes para o instante atual:

$$c_t = \sum_{j=1}^{T} \alpha_{t,j}\, h_j$$

Esse vetor (c_t) era, portanto, uma média ponderada dos *hidden states* do *encoder*, guiada pela distribuição de atenção aprendida pela rede. Ele carregava, de forma dinâmica, as partes da sentença de entrada que o modelo julgava mais importantes para o passo de geração em curso.

Por fim, o *decoder* utilizava o par (s_t, c_t) para prever o próximo token de saída $\hat{y}_t$ e atualizar seu estado interno, de acordo com a função de recorrência adotada:

$$s_{t+1} = f(s_t, y_{t-1}, c_t)$$

onde (f) representava a célula recorrente (LSTM ou GRU), e (y_{t-1}) era o token produzido anteriormente.

Dessa forma, o mecanismo de attention transformava o processo de tradução, resumo ou predição de sequência em algo mais flexível e mais inspecionável: em vez de depender de uma única representação comprimida, o modelo aprendia a recuperar, a cada passo, os fragmentos mais relevantes da sequência de entrada, como se "consultasse" dinamicamente sua própria memória.

O *multiplicative attention* de Luong

Pouco depois da proposta de Bahdanau, Luong et al. (2015) apresentaram uma variante mais leve do mecanismo de atenção, conhecida como *multiplicative attention*.

A estrutura conceitual permanecia a mesma: calcular escores de alinhamento, normalizá-los via *softmax* e então produzir um vetor de contexto. Mas o momento em que o contexto entrava no cálculo foi modificado.

Enquanto o modelo de Bahdanau integrava o vetor de contexto diretamente na atualização recorrente do estado do decoder, Luong primeiro atualizava o estado do decoder (s_t), usando o token anterior e o estado oculto anterior, e só depois combinava esse estado com o vetor de contexto na etapa de saída. Somente após a obtenção de (s_t) é que o *decoder* "olhava" para os *hidden states* do *encoder* para formar o vetor de contexto (c_t). Essa pequena mudança estrutural reduzia o custo computacional e tornava o mecanismo mais simples de treinar, sem perda significativa de desempenho.

O cálculo dos escores seguia três variantes principais, que diferiam pelo modo como o estado do *decoder* e os vetores do *encoder* eram combinados:

$$\begin{aligned}
&\text{dot } e_{(t,j)} = s_t^\top h_j \\
&\text{general } e_{(t,j)} = s_t^\top W h_j \\
&\text{concat } e_{(t,j)} = v^\top \tanh(W[s_t; h_j])
\end{aligned}$$

Nas duas primeiras variantes, *dot* e *general*, o cálculo era essencialmente um produto interno entre os vetores de ativação, com ou sem uma matriz intermediária W para ajuste dimensional. Já a versão *concat* aproximava-se mais da forma de Bahdanau, pois concatenava s_t e h_j, aplicava uma transformação linear e uma não linearidade antes de projetar o resultado em um escalar via $v^\top$.

Essa diferença revelava o que se chamava de dupla projeção no modelo de Bahdanau: nele, tanto (s_t) quanto (h_j) eram projetados separadamente para um espaço latente comum por meio das matrizes (W_s) e (W_h), antes de serem combinados e comprimidos.

Luong simplificava essa arquitetura, eliminando uma das projeções. No caso do *dot*, inclusive, não havia projeção alguma além do produto interno. Essa economia tornava o cálculo mais direto, ideal para aplicações em tempo real e sistemas de tradução mais longos.

Após o cálculo dos escores $(e_{t,j})$, a etapa seguinte permanecia idêntica à de Bahdanau: aplicava-se o Softmax para converter esses valores em pesos de atenção $(\alpha_{t,j})$,

$$\alpha_{t,j} = \frac{\exp(e_{t,j})}{\sum_{k=1}^{T} \exp(e_{t,k})}$$

e o vetor de contexto era formado pela média ponderada dos *hidden states* do *encoder*:

$$c_t = \sum_{j=1}^{T} \alpha_{t,j} h_j$$

A diferença fundamental estava no que acontecia depois: o vetor de contexto (c_t) não interferia diretamente na geração do novo estado oculto. Em vez disso, ele era fundido tardiamente com o estado do decoder já atualizado, formando um vetor intermediário (s_t):

$$s_t = \tanh(W_c[c_t; s_t])$$

no qual (W_c) era uma matriz de combinação que integrava o contexto e o estado corrente em uma única representação enriquecida. Esse vetor (s_t) era então projetado sobre o espaço de saída, convertido em *logits* e normalizado via *softmax* para gerar a probabilidade da próxima palavra.

O resultado foi um mecanismo de atenção mais enxuto, com menor custo de treinamento e inferência, que se mostrava particularmente eficaz em tarefas de tradução neural.

Ao deslocar o papel do contexto para um momento posterior no ciclo de geração, Luong reduziu a sobrecarga sem sacrificar a capacidade do modelo de capturar relações de longo alcance, abrindo caminho, alguns anos depois, para o surgimento do *self-attention* do Transformer, que radicalizaria essa ideia ao abandonar completamente a recorrência, e que veremos no próximo capítulo.

Diferenças estruturais e variações de atenção

A principal distinção entre os mecanismos de Bahdanau e Luong era estrutural. No modelo de Bahdanau, o vetor de contexto (c_t) já influenciava diretamente a atualização do estado oculto do decoder. Ou seja, o decoder levava em conta tanto o token anterior quanto as partes relevantes da sequência de entrada ao calcular o novo estado (s_t).

No modelo de Luong, em contrapartida, o cálculo do estado (s_t) ocorria de forma independente, exatamente como em uma rede recorrente convencional. O contexto era incorporado apenas em um segundo momento, na projeção da saída. Essa diferença sutil

tinha impacto prático: Bahdanau oferecia maior flexibilidade de aprendizado contextual, ao custo de mais parâmetros e operações, já Luong simplificava o processo e reduzia o consumo computacional, mantendo o desempenho em tarefas de tradução de médio porte.

Atenção global e local

Outro avanço trazido por Luong foi a introdução das variantes global e local de atenção.

O modelo de Bahdanau utilizava uma forma global, isto é, a cada passo do *decoder* todos os *hidden states* do *encoder* eram consultados para calcular os pesos de atenção. Essa abordagem oferecia a máxima capacidade de alinhamento, pois o modelo podia, teoricamente, conectar qualquer palavra de entrada a qualquer palavra de saída, mas também impunha um custo alto em termos de tempo e memória, especialmente para sequências longas.

Para contornar esse problema, Luong propôs a atenção local, em que o decoder se concentrava apenas em uma janela de tokens da entrada, delimitada em torno de uma posição-alvo (p_t). A ideia era reduzir o campo de busca, permitindo que o modelo focasse em segmentos relevantes sem precisar varrer toda a sequência a cada passo.

Havia duas formas principais de definir essa posição alvo:

Local-m (monotônica): nessa variante, o centro da janela era definido de forma direta:

$$p_t = t$$

Assumia-se que a tradução era aproximadamente monotônica, isto é, a palavra gerada no passo t do decoder tendia a se alinhar à palavra de mesma posição na entrada. Essa suposição era razoável em pares de idiomas com estrutura sintática semelhante.

Local-p (paramétrica): nessa formulação, o modelo aprendia a prever dinamicamente a posição mais provável na entrada para cada passo da saída:

$$p_t = S \cdot \sigma(v_p^\top \tanh(W_p s_t))$$

S representava o comprimento da sequência de entrada, s_t era o estado atual do *decoder*, e W_p e v_p eram parâmetros treináveis.
Esse mecanismo permitia que o modelo aprendesse a "deslizar" seu foco pela sequência, ajustando-se automaticamente à estrutura da linguagem, sem depender de hipóteses rígidas sobre monotonicidade.

Uma vez definido (p_t), delimitava-se uma janela de tamanho (D) em torno dessa posição (por exemplo, ±2 ou ±5 tokens). Os escores e pesos de atenção $(\alpha_{t,j})$ eram então calculados apenas dentro dessa janela, o que reduzia drasticamente a complexidade do processo sem comprometer a qualidade da tradução.

Treinamento, inferência e retropropagação

Durante o treinamento, esses modelos utilizavam uma técnica chamada *teacher forcing*. Em vez de alimentar o *decoder* com a palavra que ele próprio havia previsto no passo anterior, o que poderia levar à propagação de erros acumulados, injetava-se a palavra correta da sequência de referência como entrada para o próximo

passo. Essa abordagem acelerava a convergência e estabilizava o aprendizado.

Ainda durante o treinamento, o erro entre as palavras previstas e as palavras esperadas era medido por uma função de perda baseada em entropia cruzada (*cross-entropy loss*). Esse erro era então propagado para trás por meio do *backpropagation through time* (BPTT), ajustando todos os parâmetros treináveis: W_s, W_h, v, W_c, W_p e v_p, além dos pesos internos do *encoder* e do *decoder*.

Já na fase de inferência, o modelo não tinha mais acesso às palavras verdadeiras e precisava usar suas próprias predições. Como a tradução era gerada palavra por palavra, a saída era construída por *beam search*, técnica que mantinha várias hipóteses de tradução ao mesmo tempo antes de escolher a sequência final.

Usos comerciais do attention de Bahdanau e Luong

O mecanismo de *attention*, introduzido por Bahdanau et al. em 2014 e aprimorado por Luong et al. em 2015, chegou ao uso comercial principalmente pela tradução automática.

O caso mais emblemático foi o Google Neural Machine Translation (GNMT), anunciado em 2016. Sua importância não estava apenas no ganho de qualidade, mas no fato de levar a tradução neural, com LSTMs e attention, para o centro de um produto global de uso cotidiano.

Entre 2016 e 2017, a mesma virada apareceu em outros grandes sistemas comerciais. A Microsoft passou a incorporar tradução neural ao Microsoft Translator e a seus produtos associados. O

Facebook também migrou seus sistemas de tradução para modelos neurais, usados para traduzir publicações e comentários dentro da própria plataforma.

Com isso, o *attention* deixou de ser apenas uma solução acadêmica para o gargalo do Seq2Seq e passou a integrar sistemas comerciais de tradução em escala global.

Referências bibliográficas

BAHDANAU, Dzmitry; CHO, Kyunghyun; BENGIO, Yoshua. Neural Machine Translation by Jointly Learning to Align and Translate. In: ICLR 2015, International Conference on Learning Representations, 2015.

LUONG, Minh Thang; PHAM, Hieu; MANNING, Christopher D. Effective Approaches to Attention Based Neural Machine Translation. In: EMNLP 2015, Conference on Empirical Methods in Natural Language Processing. ACL, 2015. p. 1412–1421.

TUROVSKY, Barak. Found in Translation: More Accurate, Fluent Sentences in Google Translate. Google Blog, 15 nov. 2016.
WU, Yonghui et al. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. arXiv:1609.08144, 2016.

MICROSOFT TRANSLATOR. Microsoft Translator Launching Neural Network Based Translations for All Its Speech Languages. Microsoft Translator Blog, 15 nov. 2016.

MICROSOFT TRANSLATOR. Microsoft Translator Accelerates Use of Neural Networks Across Its Offerings. Microsoft Translator Blog, 15 nov. 2017.

FACEBOOK ENGINEERING. Transitioning Entirely to Neural Machine Translation. Engineering at Meta, 3 ago. 2017.

CAPÍTULO 9

2017: VASWANI E O TRANSFORMER. TUDO QUE VOCÊ PRECISA É ATENÇÃO.

O mecanismo de atenção de Bahdanau, melhorado por Luong, que permitia que o decoder consultasse diretamente os estados do encoder em cada passo (t), olhando para tokens que têm mais importância na sequência, havia ajudado as redes recorrentes (RNN - *recurrent neural networks*) como o Seq2Seq, a reduzir a perda de informações em sequências longas.

Ainda assim, o processo permanecia limitado: em frases com duas palavras específicas com certa distância entre si, ou relações menos frequentes entre essas palavras, a atenção calculada sobre todos os estados podia se dispersar, tornando o aprendizado menos eficiente. Nesses casos, o modelo acabava se perdendo ao tentar relacionar muitas palavras ao mesmo tempo dentro da frase. Quando duas palavras tinham uma ligação mais rara ou indireta, a atenção se espalhava demais e o sistema ficava sem clareza sobre quais conexões realmente importar.

Em 2017 surge o artigo "Attention Is All You Need", assinado por Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Gomez, Lukasz Kaiser e Illia Polosukhin, todos ligados ao ecossistema de pesquisa do Google.

Vaswani é indiano, estudou Engenharia Elétrica na Índia e fez mestrado na University of Southern California (USC), nos EUA. Sua trajetória é menos midiática se comparada com Bengio, Hinton ou LeCun mas seu nome entrou para a história por estar associado ao artigo que mudaria o paradigma das redes de aprendizado profundo e dos modelos de linguagem.

Entre os demais autores estavam: Noam Shazeer, americano com PhD em Matemática pela Stanford University e um dos pes-

quisadores mais experientes do grupo; Jakob Uszkoreit, alemão formado em Matemática e Ciência da Computação na Universidade Técnica de Berlim, que já trabalhava com linguagem e mecanismos de atenção no Google; Lukasz Kaiser, polonês formado em Matemática e Ciência da Computação pela Universidade de Wroclaw e PhD pela RWTH Aachen University, na Alemanha; Niki Parmar, pesquisadora que depois atuaria em empresas como Adept, Essential AI e Anthropic; Llion Jones, galês formado em Ciência da Computação e Inteligência Artificial pela Universidade de Birmingham, depois cofundador da Sakana AI; Aidan Gomez, então estagiário no Google Brain e mais tarde cofundador da Cohere; e Illia Polosukhin, pesquisador associado ao Google Research e posteriormente cofundador da NEAR.

É importante citar que o artigo registrava contribuição equivalente entre todos os autores, com a ordem de assinatura definida aleatoriamente, isto é, sem hierarquia autoral entre primeiro, segundo ou último nome.

Se nas RNNs (LSTM/GRU), o processamento era sequencial, ou seja, cada passo (t) dependia do estado do passo anterior ($t-1$), isso significava que não se poderia calcular todos os estados em paralelo. Era preciso calcular h_1, depois h_2, depois h_3 … até h_T. Em GPUs/TPUs, que são otimizadas para operações matriciais paralelas (lembre dos games e da descoberta das GPUs para processamento de modelos de linguagem no capítulo 5), isso é um gargalo. O treino ficava lento e custoso.

Se a sequência tinha 50 tokens (ou palavras se preferir), eram 50 passos encadeados, um após o outro, e cada um dependia do anterior. Uma dependência temporal. Além disso, as RNNs, como mencionado acima, sofriam com o problema de desvanecimento/explosão de gradiente: quanto mais longas eram as sequências,

mais difícil era para o modelo propagar a informação dos primeiros tokens até os últimos. Mesmo com LSTM/GRU, que introduziram gates para atenuar isso, capturar relações entre tokens distantes era difícil.

Vaswani propôs que a recorrência fosse eliminada e que somente a atenção carregasse a função estrutural de relacionar os tokens entre si. Até 2016 era consenso que a linguagem vem em sequência, logo parecia natural processá-la passo a passo, e o insight de Vaswani foi que a linguagem não é só ordem temporal, mas uma rede de relações entre palavras. Por que forçar o modelo a andar da esquerda para a direita se podemos calcular todas as relações ao mesmo tempo?

Em outras palavras, a linguagem não precisa ser processada como fluxo linear, mas como um grafo de dependências. É como sair da visão "linha do tempo" e adotar a visão "campo de forças": cada palavra exerce influência sobre as outras, e o modelo aprende quais interações são mais fortes.

A reflexão por trás disso não era nova, e tratamos disto no capítulo 01. Já se sabia, desde Chomsky e os estruturalistas, que a linguagem não é apenas sequência temporal, mas uma rede de dependências. A tradição chomskiana (anos 1950 a 70) via a linguagem como uma estrutura hierárquica (árvores sintáticas), não apenas como uma sequência linear de palavras. Na gramática generativa, cada frase tem um "esqueleto" de dependências (quem se liga a quem).

Ademais, a ordem poderia ser recuperada mais tarde com um truque simples, mas a essência era deixar que a máquina aprendesse relações em paralelo. E foi o que aconteceu.

De forma simples, Vaswani "jogou fora" toda a estrutura recorrente das RNNs, como o Seq2Seq, e partiu da matriz input com os *embedding values* de cada token, aplicando operações lineares e transformações de forma paralela.

Compreendendo o modelo – o encoder

No primeiro passo, o modelo de Vaswani trabalhava com uma matriz de dimensões $(n \times d) : (n)$ representava o número de tokens da frase, e (d) o tamanho do vetor que descrevia cada token. Essa era a matriz de entrada, como vimos em capítulos anteriores, formada pelos *embeddings*.

Para que o modelo também soubesse a ordem em que cada token aparecia, Vaswani somou a essa matriz uma segunda matriz, de mesma dimensão (n x d): a matriz de posição. Enquanto a primeira dizia *"o que"* cada palavra era, a segunda marcava *"onde"* ela estava na sequência.

Essa matriz de posição era construída a partir de funções de seno e cosseno aplicadas de forma alternada nas dimensões do embedding. Se cada token tivesse, por exemplo, $d = 512$ dimensões, a matriz de posição também teria 512 dimensões, organizadas em pares numerados de $k = 0$ até $k = 255$.

Em cada par numerado, a dimensão par $(2k)$ recebia um valor de seno e a dimensão ímpar $(2k + 1)$ recebia um valor de cosseno. O resultado é que cada posição na frase ganhava uma "assinatura ondulatória" única, combinando senos e cossenos, como se cada palavra fosse marcada por um carimbo que indicava não só seu conteúdo, mas também o lugar exato onde apareceu.

À medida que (k) aumentava, a frequência das ondas ia ficando mais lenta. Dessa forma, os diferentes pares cobriam várias escalas: os (k) mais baixos marcavam variações rápidas, úteis para distinguir vizinhos próximos, enquanto os (k) mais altos capturavam padrões mais amplos, ajudando a reconhecer estruturas de longo alcance na frase. Dessa forma o modelo obtinha uma escala logarítmica de frequências, cobrindo do curto ao longo alcance em cada token da sequência.

Isso significava que, se a palavra gato aparecesse em diferentes posições ao longo do corpus, seus *embeddings* básicos seriam os mesmos. O que mudava eram os vetores finais após a soma com a matriz de posição, ou seja, gato mantinha o mesmo "conteúdo", mas ganhava uma assinatura distinta conforme a posição em que surgia em cada frase.

Com a matriz de características e posição pronta, o modelo de Vaswani seguia, trabalhando agora paralelamente com três matrizes de pesos aplicáveis aos *embeddings* de entrada (e esse é outro ponto de inflexão na história dos modelos). Essa operação gerava três novas matrizes de representação:

Q, representando consulta,
K, representando chave,
V, representando valor.

Cada uma dessas três matrizes cumpria um papel conceitualmente distinto, embora todas derivassem da mesma matriz de entrada.

Q (query) traduzia o que cada token "procurava" na sequência, a pergunta que ele fazia ao contexto.

K (key) representava o que cada token "oferecia", sua assinatura

informacional, usada para medir compatibilidade com as consultas.

V (value) carregava o conteúdo efetivo, o vetor de informação que seria recuperado e combinado conforme a relevância calculada entre Q e K.

Em termos geométricos, Q e K definiam a superfície de atenção (o quanto cada vetor se projeta sobre os demais), enquanto V fornecia o material que seria ponderado e redistribuído. É dessa interação entre busca (Q), referência (K) e conteúdo (V) que emergia a capacidade contextual do Transformer.

De forma simplificada, se entrássemos com uma matriz de entrada $n \times 512$, onde n é o número de tokens e 512 é a dimensão do modelo (colunas de parâmetros), e o bloco de atenção fosse dividido em oito cabeças, cada uma delas produziria um resultado $n \times 64$.

Esse é um momento importante na evolução dos modelos de linguagem e a possibilidade de computação paralela surge para completar o *insight* do uso das GPUs que já citamos nesse livro. Como exemplo, e para racionalização do modelo, uma cabeça poderia se especializar em relações gramaticais (quem faz o quê), outra poderia focar em proximidade semântica ("gato" e "sofá"), e outra poderia capturar ritmo e ordem (tokens sequenciais).

Esse resultado viria de três projeções lineares, com os pesos Q, K e V, cada uma também de forma $n \times 64$., obtidas pela multiplicação da matriz de entrada completa por cada uma das matrizes de pesos.

Em seguida, cada cabeça realizaria os cálculos de atenção: um produto interno $QK^\top$, seguido pela normalização pela raiz qua-

drada do número de colunas das matrizes (nesse caso, $\sqrt{64}$), pela aplicação da função Softmax (gerando os pesos de atenção $n \times n$), e finalmente pela multiplicação por V, resultando em uma matriz $n \times 64$ já ponderada. As oito cabeças eram então concatenadas, formando uma matriz $n \times 512$.

Por fim, essa matriz era multiplicada por uma matriz de pesos W^O, de forma 512×512, resultando novamente em uma saída $n \times 512$, que voltava a ter a mesma dimensão da entrada, já representando a sequência com a atenção aplicada. Vale a pena falarmos sobre esse último passo da aplicação do Attention, que era a multiplicação por uma matriz chamada W^O, de dimensão 512×512 no exemplo que estamos usando.

W^O era um parâmetro global da camada de atenção. Não mudava de sequência para sequência ou de batch para batch. Embora se ajustasse a cada *batch* no *backpropagation* do erro, refinando-se aos poucos, ainda assim seguia sendo o mesmo conjunto de parâmetros para todo o conjunto de treinamento.

A razão disso era o processamento paralelo, dividido por um certo número de cabeças operando simultaneamente, cada uma devolvendo uma matriz resultante, formada a partir de sua própria operação matricial com os pesos W_Q, W_K e W_V.

O próximo passo era concatenar esses vetores, lado a lado, para que a matriz voltasse a ter, por exemplo, 512 colunas. Essa concatenação gerava um conjunto de representações desalinhadas, visto que cada uma das cabeças tinha sua própria leitura semântica da frase, ou seu próprio aprendizado.

Geometricamente, essa matriz concatenada ainda precisava ser reorganizada no espaço vetorial, e a matriz W^O misturava as di-

mensões entre as cabeças e reorientava o vetor resultante para o espaço dimensional do modelo. Tecnicamente, em álgebra linear, era uma transformação linear aprendida, quase como uma reorganização do espaço vetorial, aplicada à matriz concatenada vinda das cabeças de processamento.

O que isso afirmava indiretamente era ainda mais interessante: que Vaswani aceitava essa generalização porque, indiretamente, e essa é a afirmação estrutural mais sofisticada do paradigma Transformer, a "generalidade cognitiva" do modelo é derivada e dependente da regularidade estatística dos corpora. Ou seja, em outras palavras, a linguagem humana, em sua prática ordinária, repete-se o suficiente para ser generalizável.

Uma vez que a atenção multicabeça foi consolidada, o modelo tinha mais um truque elegante, chamado conexão residual (*skip connection*).

A matriz de *embeddings* original era somada à matriz de *embeddings* com atenção aplicada. Ou seja, x era somado a $MHA(x)$.

$$f(x) = MHA(x) + x$$

Assim, se o sinal aprendido pela atenção passasse a contribuir com valores numericamente muito pequenos ao atravessar as camadas, f(x) preservava um caminho direto para a informação original e reduzia o risco de vanishing gradient. Essa soma criava aprendizado residual, embedding original + aprendizado, uma diferença, não uma substituição, e reduzia o risco de apagamento de informações.

E tinha mais elegância matemática, agora usando estatística clássica. Lembre-se de que cada token ficava com uma matriz 1×512

que era formada por $MHA(x) + x$.

Esses números tinham dimensões com variações muito grandes, ainda resultantes das diferenças entre as cabeças de processamento, mesmo com a aplicação de $W^{\wedge}O$ (que reorganizava, mas não normalizava). Era nesse ponto que o modelo aplicava algo chamado $LayerNorm$. Usando os conceitos de média e desvio padrão, o $LayerNorm$ centralizava o vetor em torno de zero e trazia o desvio padrão para perto de 1.

Aplicava nesse cálculo dois pesos apreendíveis, γ e β, reescalando e recentrando o vetor de acordo com o que o próprio modelo aprendia ao longo do treino. Em outras palavras, primeiro ele retirava a escala natural, usando média e desvio padrão, e logo em seguida devolvia uma nova escala, agora ajustada pela experiência do modelo.

O resultado era um vetor equilibrado, com valores distribuídos em torno de zero, estáveis para o fluxo de informação entre as camadas. Essa etapa de normalização estatística simples, mas elegante, era o que ajudava a manter o Transformer matematicamente estável ao longo das camadas empilhadas.

Após o $LayerNorm$, o modelo aplicava outro conjunto de funções, mais complexo em termos matemáticos, chamado Feed-Forward Network (FFN). Em poucas palavras, essa etapa expandia a matriz de cada token já processado pela atenção no caso do modelo de Vaswani, a matriz tomava quatro vezes o tamanho do *input*, ou seja, de 1×512 para 1×2048 no nosso exemplo, usando dois conjuntos de pesos aprendíveis e atualizáveis (W_1 e b_1).

Em seguida, aplicava uma função que zerava os valores negativos ($ReLU$) e, logo depois, comprimia novamente o vetor para

1×512, agora utilizando os pesos W_2 e b_2, atualizando mais uma vez a matriz de *embeddings*.

Perceba que as camadas repetiam um mesmo padrão de comportamento: refinar os *embeddings* que haviam recebido o mecanismo de atenção, buscando uma representação cada vez mais precisa de cada token já contextualizado dentro da sequência. Ou, dito de forma mais intuitiva, de cada palavra dentro de uma frase.

Nessa etapa, o processo ocorria em duas transformações sucessivas, cada uma com seu próprio conjunto de W e b : a primeira para expandir e a segunda para recomprimir o vetor. Entre essas duas transformações, a $ReLU$ atuava como um filtro matemático simples e eficiente, mantendo apenas as ativações positivas e eliminando as negativas.

Não satisfeito, Vaswani buscava um refinamento quase perfeito de seu modelo. Ele somava a matriz resultante do primeiro $LayerNorm$ à matriz resultante da FFN, gerando outro aprendizado residual. Sobre essa soma, aplicava um novo $LayerNorm$, com o mesmo processo de centralização e reescala explicado anteriormente.

O modelo de Vaswani repetia esse procedimento seis vezes em sequência, fazendo com que a saída de cada camada se tornasse a entrada da próxima.

Em termos simplificados, a primeira camada captava relações locais (quem está perto de quem), a segunda começava a perceber padrões sintáticos, as intermediárias identificavam dependências de médio alcance, e as últimas passavam a representar significados globais, contextuais e abstratos. Cada camada, portanto, aprendia um nível diferente de abstração linguística.

Após o processamento pelas seis camadas empilhadas, a matriz de *embeddings* resultante estava finalmente pronta para ser utilizada pelo Decoder, já carregando múltiplos níveis de compreensão da sequência.

Decodificando o decoder

O exemplo clássico no modelo de Vaswani era a tradução automática. O *encoder* lia a frase em uma língua e o *decoder* produzia, palavra por palavra, a tradução correspondente. Mas havia uma diferença estrutural essencial entre o *encoder* e o *decoder*: enquanto o *encoder* tinha acesso a toda a sequência de entrada, o *decoder* só podia ver o que já havia produzido até o momento.

Ele precisava gerar cada token de forma sequencial, usando o que já havia produzido até aquele ponto. É aqui que surgiu o conceito do *Masked Self-Attention.* Cada camada do *decoder* era mais complexa do que uma camada do *encoder.* Se uma camada do *encoder* era formada, estruturalmente, pelo *multihead self-attention* e pelo *feed-forward*, no *decoder* a camada era formada pelo *masked multi-head self-attention*, o *encoder-decoder attention* e o *feed-forward*.

O *multi-head self-attention* do *encoder* permitia que cada token observasse todos os outros da sequência. Não havia restrições de tempo ou ordem. Já no *decoder*, isso seria um problema. Ao gerar a palavra 4 ("token$_4$"), o modelo não podia "espiar" as palavras 5, 6, etc., porque elas ainda não existiam.

Por essa razão o *self-attention* do decoder era mascarado: ele só podia olhar para os tokens anteriores e para si mesmo. A forma de mascarar os tokens que o modelo não deveria enxergar era

notavelmente elegante. Antes de aplicar o softmax, que utilizava a função exponencial de Euler para converter os scores em probabilidades, o Transformer introduzia uma matriz de máscara sobre a matriz de atenção $QK^\top/\sqrt{d_k}$.

Essa máscara definia uma regra simples: para cada posição i da sequência do *decoder*, apenas os tokens até i podem ser vistos. Assim, posições futuras (j > i) recebiam o valor $-\infty$, enquanto as posições permitidas (j ≤ i) mantinham zero. Como $e^{-\infty} = 0$, o *Softmax* anulava completamente a influência dos tokens futuros, garantindo que o *decoder* gerasse a sequência de forma autoregressiva.

Após o *masked multi-head self-attention*, o segundo bloco, *encoder-decoder attention*, fazia o papel de elo entre as duas metades do Transformer (*encoder* e *decoder*). Ali, as Queries (Q) vinham do decoder, enquanto as Keys (K) e Values (V) vinham da saída final do encoder.

Era nesse momento que o decoder "consultava" o que o encoder havia compreendido da frase original. Esse mecanismo permitia, por exemplo, que o modelo relacionasse a palavra gerada com a parte da frase de entrada que mais se conectava a ela.

Peguemos como exemplo a frase "*The cat is sitting on the couch*" introduzida no *encoder,* e que o *decoder* devia traduzir como "O gato está sentado no sofá". Na tradução, era ali que o modelo descobria que "gato" correspondia a "*cat*", e "sentado no sofá" se ligava a "*sitting on the couch*".

Depois que o *encoder* processava a frase de entrada, ele produzia um conjunto de vetores de *embedding* (um por palavra), que carregavam informações semânticas e posicionais, estas últimas

introduzidas no modelo de Vaswani.

Esses vetores alimentavam as matrizes K e V. Quando o decoder começava a gerar a tradução ("O gato está…"), ele criava Q com base no que já havia produzido.

O produto $Q \times K^{\top}$ media quão relevante era cada palavra de entrada para o token que estava sendo gerado, e o Softmax transformava essas relevâncias em pesos de atenção. Esses pesos ponderavam V, formando o vetor de contexto, uma espécie de "resumo ponderado" da entrada, focado no que importava para aquele momento da tradução.

Como simplificação, quando o decoder precisava gerar a palavra "sentado", após o cálculo das pontuações de atenção e sua normalização pelo Softmax, os maiores pesos tendiam a se concentrar em "sitting", mas também podiam distribuir parte da atenção pelo entorno da expressão "on the couch". Em seguida, ao gerar "sofá", os maiores pesos tenderiam a se deslocar para "couch". A expressão inteira, "sentado no sofá", era reconstruída pela sequência dessas aproximações, não por uma associação direta e fixa entre blocos de palavras.

Isso significava que o token "sentado" estava "olhando" principalmente para "sitting", mas também para o entorno da expressão "on the couch" na frase original em inglês e, a partir dos Values (V) correspondentes, formava um vetor interno de representação, como soma ponderada desses vetores.

Esse vetor de contexto era então combinado com a representação já produzida pelo decoder para aquela posição e processado pelas camadas seguintes, incluindo a feed-forward, que refinava a informação e produzia uma nova representação oculta (ht) para

aquele instante da tradução.

Em seguida, o modelo comparava (ht) com todos os vetores do seu vocabulário, a *embedding matrix*, que continha uma representação vetorial para cada palavra conhecida. O produto entre (ht) e essa matriz gerava uma lista de escores (*logits*) que expressavam o quanto cada palavra do vocabulário "se encaixava" naquele contexto.

Aplicando novamente o Softmax, o Transformer transformava esses escores em probabilidades: quanto maior o valor, maior a chance de aquela palavra ser a próxima da sequência. Assim, o vetor (ht), obtido ao olhar para "*sitting on the couch*", aproximava-se no espaço semântico do vetor correspondente à palavra "sentado", ou, se assim preferir, o espaço matricial aprendido.

Essa proximidade era interpretada pelo modelo como a melhor correspondência possível para o contexto atual, fazendo com que "sentado" recebesse a maior probabilidade e fosse escolhida como o próximo token gerado. Desse modo, o decoder traduzia passo a passo, reconstruindo a frase de destino a partir de relações numéricas de similaridade entre vetores, e não de associações diretas entre palavras.

O terceiro bloco do *decoder*, *Feed-Forward Network* (FFN), era idêntico ao do encoder: uma expansão linear, aplicação da função ReLU, e recompressão do vetor, com pesos aprendíveis (W^1, b^1, W^2, b^2). Essa etapa atuava refinando o vetor do token recém-gerado, consolidando a informação combinada entre o contexto já produzido e o conteúdo da entrada original.

Cada um desses três blocos era seguido por uma conexão residual e uma LayerNorm, garantindo estabilidade numérica e preser-

vando o fluxo de gradiente, exatamente como no Encoder.

Após passar por todas as camadas do *decoder,* que, assim como no encoder, são empilhadas em seis blocos, o modelo aplicava uma projeção linear final e uma função Softmax. Essa função convertia o vetor de saída de cada token em uma distribuição de probabilidades sobre todo o vocabulário. O token com maior probabilidade era então escolhido, dependendo da estratégia, formando a palavra final que o modelo "escreve".

Assim, o decoder encerrava o ciclo completo do Transformer: do entendimento profundo da entrada (*encoder*) à produção ordenada e contextualizada da saída (*decoder*).

A lógica sequencial do Decoder

O processo de geração funcionava em loop autoregressivo.

O *decoder* começava recebendo um token especial de início de sequência (*Start of Sequence – SOS*, como mencionado anteriormente) e, a partir dele, previa o próximo token provável.

Esse token gerado era então realimentado no modelo, que repetia o ciclo:

atenção mascarada > atenção cruzada > feed-forward > projeção > softmax.

A cada passo, o *decoder* utilizava o que já havia produzido e o que aprendera do Encoder para decidir qual era a próxima palavra mais provável. Esse processo se repetia até que o modelo gerasse o token de fim de sequência (*End of Sequence – EOS*), sinalizando

que a frase estava completa.

Aqui havia um ponto interessante que, embora muitas tentativas tenham sido feitas para contornar, seguia sendo um obstáculo nos modelos de linguagem até hoje. Durante a geração autoregressiva, o decoder produzia os tokens de forma estritamente sequencial, isto é, cada passo dependia do anterior.

Matematicamente, o processo ocorria em instantes discretos $(t, t+1, t+2, \ldots)$, em que o modelo gerava o token (y_t) com base em todos os tokens anteriores $(y_1, y_2, \ldots, y_{t-1})$. Somente após concluir a previsão de (y_t), ele podia utilizar esse resultado como entrada para calcular (y_{t+1}).

Durante o treinamento, essa limitação não era um problema, pois todas as sequências corretas estavam disponíveis de antemão, permitindo processar todos os tokens em paralelo, com o *masking* garantindo que cada posição só "visse" o passado.

Porém, na inferência, isto é, quando o modelo estava de fato gerando texto, o processo se tornava inevitavelmente sequencial. Cada novo token exigia que as operações de atenção, projeção e passagem pelas camadas fossem novamente realizadas com base nos resultados anteriores, o que impunha um gargalo natural ao paralelismo.

Esse comportamento tinha consequências práticas diretas: como a inferência não podia ser completamente paralelizada, o tempo e o custo energético de geração aumentavam proporcionalmente ao comprimento da sequência. Cada token adicional significava mais operações de atenção, mais multiplicações matriciais e, portanto, mais consumo elétrico. Em escala de *data centers*, onde milhões de consultas ocorrem simultaneamente, essa dependência temporal

se traduz em altíssimo gasto energético e em limites físicos para a velocidade de resposta.

Mesmo com otimizações modernas, como *key/value caching*, *speculative decoding* ou predições múltiplas, a essência do processo permanece a mesma no momento que escrevo esse livro: a geração segue sendo sequencial no tempo. Cada novo token é, ainda, o produto inevitável de todos os anteriores.

Em outras palavras, os modelos atuais, em Novembro de 2025, ainda não conseguem aplicar paralelismo pleno na etapa de geração (inferência). Essa limitação permanece como um dos maiores obstáculos computacionais da arquitetura Transformer, que segue sendo o paradigma matemático dominante nos modelos de inteligência artificial contemporâneos, impondo um custo direto em tempo de inferência e em consumo energético.

A dependência sequencial entre os tokens faz com que cada nova palavra exija o processamento cumulativo de todas as anteriores, tornando a geração textual um processo intensivo em recursos computacionais e energia elétrica, mesmo em data centers de última geração.

A Matemática por trás do Transformer de Vaswani

Cada elemento de PE era calculado de forma determinística, usando funções senoidais alternadas que variavam com a posição (pos) e com o índice da dimensão (i):

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

Com a matriz de entrada pronta, Vaswani projetava (X') em três subespaços lineares distintos (consultas, chaves e valores) multiplicando-a por três matrizes de pesos aprendíveis $(W_Q), (W_K)$ e $(W_V) \in \mathbb{R}^{d_{model} \times d_k}$, resultando em:

$$Q = X'W_Q$$
$$K = X'W_K$$
$$V = X'W_V$$

O resultado, normalizado pela raiz quadrada de d_k era transformado em probabilidades pelo Softmax, formando a matriz de pesos de atenção A:

$$A = Softmax\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}}{}_k\right)$$

A partir de A, o modelo calculava a nova representação contextual de cada token como uma combinação ponderada dos valores V:

$$Attention(Q, K, V) = Softmax\left(\frac{QK^{\mathrm{T}}}{\sqrt{d}}{}_k\right)V$$

Cada cabeça calculava atenção com matrizes de pesos próprias $W_{Q(i)}, W_{K(i)}, W_{V(i)}$, e cada uma delas se especializava em tipos diferentes de relações linguísticas: sintáticas, semânticas ou posicionais. Suas saídas eram então concatenadas e reorientadas por uma projeção linear $W_O \in \mathbb{R}^{hd_k \times d_{\mathrm{model}}}$:

$$MultiHead(Q, K, V) = Concat(head^1, ..., head_{\mathrm{h}})W_O$$

Essa multiplicação final misturava as direções aprendidas por cada cabeça, restabelecendo o vetor de cada token ao espaço original de 512 dimensões. O resultado passava por uma conexão residual, somando a entrada original X' ao vetor de atenção:

$$Z^1 = X' + MultiHead(Q, K, V)$$

Em seguida, aplicava-se a normalização de camada ($LayerNorm$), que recentrava e reescalava cada vetor, garantindo estabilidade numérica:

$$\text{LayerNorm}(x_i) = \gamma \times \frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

O resultado dessa normalização alimentava uma pequena rede neural densa, aplicada individualmente a cada token, chamada de Feed-Forward Network (FFN). Ela expandia o vetor para quatro vezes sua dimensão original, aplicava uma função de ativação ReLU e, em seguida, projetava de volta ao tamanho original:

$$FFN(x) = \max(0, xW^1 + b^1)W^2 + b^2$$

O vetor resultante passava novamente por soma residual e normalização, completando um bloco Transformer:

$$H = LayerNorm(X + MultiHead(X))$$
$$Block(X) = LayerNorm(H + FFN(H))$$

Esse bloco era repetido seis vezes no *encoder* e seis vezes no *decoder*.

No decoder, a máscara M era uma matriz triangular inferior definida por:

$$M_{ij} - 0,\ se\ j \leq i\ -\infty,\ se\ j > i$$

e aplicada antes do Softmax, anulando as posições proibidas, já que $e^{(-\infty)} = 0$. O decoder também introduzia um segundo mecanismo de atenção, o encoder-decoder attention, em que as queries Q provinham do decoder e as keys e values (K,V) do encoder - permitindo que o modelo "consultasse" as informações da frase original enquanto gerava a tradução.

A camada final projetava o vetor oculto (ht) de cada token sobre o espaço do vocabulário, com a matriz de embeddings (W_E), obtendo os logits:

$$logits = h_t W_E^\top$$

e convertendo-os em probabilidades com o Softmax:

$$P(y_t = w_i \mid y_{<t}, X) = \frac{e^{\text{logit}_i}}{\sum_j e^{\text{logit}_j}}$$

O token de maior probabilidade era então escolhido e realimentado ao modelo, fechando o ciclo autorregressivo.

Usos comerciais do Transformer de Vaswani et al.

A arquitetura Transformer, apresentada por Vaswani et al. em 2017 no artigo "*Attention Is All You Need*", marcou uma ruptura definitiva com os modelos recorrentes.

Ao eliminar as RNNs e LSTMs, substituindo-as por mecanismos de atenção totalmente paralelizáveis, tornou-se possível treinar modelos de linguagem em larga escala com eficiência e desempe-

nho inéditos. Essa mudança redefiniu a indústria de NLP a partir de 2018 e segue sendo o paradigma matemático dos modelos LLM contemporâneos.

Um dos primeiros usos industriais de grande visibilidade foi o BERT, apresentado pelo Google em 2018 como modelo de representação de linguagem e aplicado depois ao Google Search, em 2019, para melhorar a compreensão contextual das buscas.

Poucos meses depois, a Microsoft também integrou variantes baseadas em Transformer a seus sistemas de tradução, busca e assistentes digitais, aprimorando a contextualização das respostas e reforçando que a arquitetura deixava rapidamente o campo experimental para entrar em produtos comerciais.
No mesmo período, o OpenAI GPT mostrou o potencial generativo dos Transformers ao produzir texto coerente em escala, inaugurando uma nova etapa dos modelos de linguagem de propósito geral. Se o BERT reforçava o lado do entendimento de linguagem, o GPT apontava para a geração de linguagem.

Entre 2019 e 2021, a arquitetura Transformer tornou-se a espinha dorsal de praticamente todos os grandes modelos de NLP, como T5, do Google, GPT 2 e GPT 3, da OpenAI, RoBERTa, da Meta, XLNet, do Google e da Carnegie Mellon University, e Megatron, da NVIDIA. Plataformas comerciais começaram a oferecer APIs e produtos baseados nesses modelos para correção gramatical, redação automatizada, sumarização, busca semântica, classificação de texto e automação de atendimento.

O GPT 2, lançado pela OpenAI em fevereiro de 2019 com 1,5 bilhão de parâmetros, ampliou esse debate ao demonstrar maior capacidade de geração textual e ao ser inicialmente restrito por razões de segurança, em um processo de liberação gradual. Foi mais

um marco técnico e público do que um marco comercial direto.

O salto comercial mais claro veio com o GPT 3, em 2020. Em junho daquele ano, a OpenAI abriu sua API para empresas e desenvolvedores, permitindo que o modelo fosse integrado a produtos de redação, busca, classificação, resumo e automação textual. Poucos meses depois, a Microsoft obteve licença para incorporar GPT 3 aos seus próprios produtos e serviços, sem impedir o acesso de outros usuários pela API da OpenAI.

Ainda antes de 2022, começaram a surgir aplicações comerciais mais visíveis. AI Dungeon usou GPT 3 para geração narrativa em jogo interativo. Algolia explorou o modelo em busca semântica em linguagem natural. Jasper, então Conversion.ai, avançou no uso de GPT 3 para textos de marketing e publicidade. Quizlet passou a explorar o modelo em ferramentas de estudo adaptativo. Outras empresas testaram usos em atendimento, moderação, análise de feedback, escrita assistida e classificação de conteúdo.

A partir de 2022, com a chegada de modelos como GPT 3.5, ChatGPT e Claude, os Transformers se consolidaram como infraestrutura de base da economia da IA, impulsionando assistentes corporativos, copilotos de programação, ferramentas criativas de texto, imagem e vídeo, sistemas de busca aumentada e automações baseadas em linguagem natural.

A arquitetura de Vaswani não apenas aprimorou o desempenho técnico. Ela redefiniu o conceito de produto de IA comercial.

Usos comerciais iniciais de GPT 3 (antes de 2022)

Jun 2020 - lançamento da API GPT-3 para empresas e devs.

A OpenAI abriu uma API "text in, text out", permitindo integrar GPT-3 em produtos (NLP geral, prototipagem, automação).

Set 2020 - licença exclusiva da Microsoft.
A Microsoft obteve licença para incorporar GPT-3 em seus próprios produtos/serviços, em paralelo ao acesso público via API.

Jul 2020 - AI Dungeon (Latitude) lança "Dragon" em GPT-3.
Jogo narrativo com geração de histórias dinâmicas para usuários premium, rodando em modelo baseado em GPT-3.

2020 - Algolia integra GPT-3 no "Answers".
Busca semântica em linguagem natural para publishers e help desks, co-anunciada pela OpenAI como case de lançamento.
2020 - Koko e Replika exploram GPT-3 em saúde mental/companions.
Koko (triagem de crise) e Replika (companheiro conversacional) aparecem entre os primeiros usuários da API.

2020 - Reddit testa GPT-3 para moderação.
Exploração de automação de moderação e suporte interno com a API em fase inicial.

Set 2020 a 2023 - Quizlet adota GPT-3 para estudo adaptativo.
A empresa relata colaboração com a OpenAI desde 2020 (vocabulário, practice tests) e, mais tarde, amplia para tutorias.

Jan 2021 - Jasper (ex-Conversion.ai) estreia com GPT-3 para marketing.
Ferramenta de copy para anúncios e conteúdo, um dos primeiros "app layers" comerciais sobre GPT-3.

Mar 2021 - onda de "apps de GPT-3".
A OpenAI destaca dezenas de apps comerciais (incluindo Algolia

Answers) na primeira vitrine pública de parceiros.

2021 - estudos e cases setoriais (ex.: análise de feedback com Viable). Uso de GPT-3 para resumir e classificar feedback de clientes em escala.

Linha do tempo do Transformer

2014: Sutskever, Vinyals e Le publicam o modelo Seq2Seq com redes recorrentes.

2015: Bahdanau, Cho e Bengio apresentam o mecanismo de atenção aplicado à tradução neural.

2015: TensorFlow é lançado pelo Google.

2016: PyTorch começa a ganhar espaço como framework de deep learning.

2017: Vaswani et al. publicam "Attention Is All You Need".

2017: O Transformer elimina a recorrência como eixo estrutural do modelo.

2017: O modelo passa a usar embeddings somados a positional encoding.

2017: A atenção passa a ser calculada por Query, Key e Value.

2017: O modelo introduz o uso sistemático de multi head attention.

2017: O Transformer original usa seis camadas no encoder e seis

camadas no decoder.

2017: O decoder usa masked self attention para impedir acesso a tokens futuros.

2017: O decoder usa encoder decoder attention para consultar a saída do encoder.

2017: O código do Transformer aparece no ecossistema Tensor-2Tensor, do Google.

2018: O GPT, da OpenAI, mostra o potencial generativo dos Transformers.

2018: O BERT, do Google, mostra a força dos Transformers no entendimento de linguagem.

2019: O Google anuncia o uso de BERT no Google Search.

2019: A OpenAI lança o GPT 2, com 1,5 bilhão de parâmetros.

2019: RoBERTa, XLNet e outros modelos ampliam o uso de Transformers em NLP.

2020: A OpenAI lança a API do GPT 3 para empresas e desenvolvedores.

2020: A Microsoft obtém licença para incorporar GPT 3 em seus produtos e serviços.

2020: AI Dungeon usa GPT 3 em geração narrativa interativa.

2020: Algolia explora GPT 3 em busca semântica.

2020: Koko e Replika exploram GPT 3 em aplicações conversacionais.

2020: Reddit testa GPT 3 em moderação.

2020: Quizlet inicia colaboração com a OpenAI em ferramentas de estudo.

2021: Jasper, então Conversion.ai, usa GPT 3 para textos de marketing.

2021: OpenAI divulga uma primeira onda de aplicações comerciais baseadas em GPT 3.
2021: ViT, Vision Transformer, amplia o uso da arquitetura Transformer para visão computacional.

2022: ChatGPT populariza os modelos conversacionais baseados em Transformer.

O que possibilitou a sofisticação matemática do Transformer de Vaswani

À primeira vista, o Transformer parece um salto repentino, um avanço matemático de magnitude inatingível, como se Vaswani e seus coautores tivessem inventado um novo universo de cálculo.

Mas a verdade é mais interessante. O Transformer não nasceu do nada, e tampouco foi fruto apenas de genialidade individual. Ele foi, sobretudo, uma escolha dentro de um ecossistema já pronto, um conjunto de peças que vinham sendo construídas por décadas.

Quando o artigo *Attention is All You Need* foi publicado, em 2017,

o terreno técnico já estava fértil. As GPUs haviam se tornado acessíveis, os frameworks de *deep learning* já encapsulavam toda a álgebra linear necessária, e o Python havia se consolidado como a língua franca da ciência de dados. O papel de Vaswani e sua equipe foi reorganizar essas peças, não as reinventar.

O modelo original foi implementado pelo time do Google Research, e seu primeiro código público apareceu pouco depois, na biblioteca Tensor2Tensor (T2T), liderada por Łukasz Kaiser, coautor do artigo e engenheiro sênior do Google Brain.

O código-fonte completo do modelo, incluindo *encoder, decoder, feed-forward* e camadas de atenção, somava milhares de linhas. Mas o núcleo matemático, a *Scaled Dot-Product Attention*, cabia em menos de 50 linhas de Python.

O que fazia essa simplicidade possível era o que estava por trás:

- TensorFlow (lançado em 2015) e o emergente PyTorch (2016) já traziam operações vetorizadas otimizadas (multiplicações de matrizes, *softmax, dropout,* convoluções), tudo implementado em C++ e CUDA, mas acessível via Python.

- O mecanismo de *autograd* realizava automaticamente o *backpropagation*: bastava descrever o *forward*, e o *framework* cuidava dos gradientes.

- As bibliotecas já dispunham de infraestrutura de *batching, datasets*, inicialização de pesos e otimizadores (como Adam e RMSProp).

Assim, Vaswani e seus colegas não precisaram reescrever a matemática de base. Eles apenas rearranjaram essas funções, multipli-

cações de matrizes, normalizações, funções não lineares, em uma estrutura nova e coerente. O que antes exigiria milhares de linhas em C++ ou CUDA pôde ser expresso, em 2017, com poucas dezenas em Python.

Em certo sentido, o Transformer foi inevitável. A infraestrutura estava madura o suficiente para que alguém, mais cedo ou mais tarde, conectasse as ideias certas. A atenção escalonada era apenas a peça que faltava para encaixar na caixa de ferramentas que já existia: GPUs rápidas, *backpropagation* eficiente, bibliotecas estáveis e *datasets* massivos.

Por isso, compreender o Transformer é também compreender o tempo em que ele surgiu. Sua sofisticação matemática não foi um acidente, mas o resultado de um longo processo de descobertas, erros e ferramentas. Quando as condições convergiram (hardware, software e teoria), a arquitetura simplesmente aconteceu. Como tudo, inovação incremental da humanidade.

Referências bibliográficas

ABADI, Martín et al. TensorFlow: Large Scale Machine Learning on Heterogeneous Distributed Systems. In: arXiv preprint arXiv:1603.04467, 2016.

BAHDANAU, Dzmitry; CHO, Kyunghyun; BENGIO, Yoshua. Neural Machine Translation by Jointly Learning to Align and Translate. In: ICLR 2015, International Conference on Learning Representations, 2015.

BROWN, Tom B. et al. Language Models are Few Shot Learners. In: NeurIPS 2020, Advances in Neural Information Proces-

sing Systems, v. 33, 2020.

CHILD, Rewon; GRAY, Scott; RADFORD, Alec; SALIMANS, Tim. Generating Long Sequences with Sparse Transformers. In: arXiv preprint arXiv:1904.10509, 2019.

CHO, Kyunghyun et al. Learning Phrase Representations using RNN Encoder Decoder for Statistical Machine Translation. In: EMNLP 2014, Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, 2014.

DEVLIN, Jacob; CHANG, Ming Wei; LEE, Kenton; TOUTANOVA, Kristina. BERT: Pre training of Deep Bidirectional Transformers for Language Understanding. In: NAACL 2019, Conference of the North American Chapter of the Association for Computational Linguistics. Association for Computational Linguistics, 2019.

DOSOVITSKIY, Alexey et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In: ICLR 2021, International Conference on Learning Representations, 2021.
GOOGLE. Understanding searches better than ever before. The Keyword, 25 out. 2019.

HOCHREITER, Sepp; SCHMIDHUBER, Jürgen. Long Short Term Memory. Neural Computation, v. 9, n. 8, p. 1735–1780, 1997.

KAISER, Łukasz et al. One Model to Learn Them All. In: ICML 2017, Proceedings of the 34th International Conference on Machine Learning, 2017.

LATITUDE. AI Dungeon: Dragon Model Upgrade. Medium, 14 jul. 2020.

LEWIS, Mike et al. BART: Denoising Sequence to Sequence Pre training for Natural Language Generation, Translation, and Comprehension. In: ACL 2020, Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics, 2020.

LIU, Yinhan et al. RoBERTa: A Robustly Optimized BERT Pretraining Approach. In: arXiv preprint arXiv:1907.11692, 2019.

MICROSOFT. Microsoft teams up with OpenAI to exclusively license GPT 3 language model. Microsoft Official Blog, 22 set. 2020.
OPENAI. OpenAI API. OpenAI, 11 jun. 2020.

OPENAI. GPT 3 powers the next generation of apps. OpenAI, 25 mar. 2021.

OPENAI. Introducing ChatGPT. OpenAI, 30 nov. 2022.

PASZKE, Adam et al. PyTorch: An Imperative Style, High Performance Deep Learning Library. In: NeurIPS 2019, Advances in Neural Information Processing Systems, 2019.

RADFORD, Alec; NARASIMHAN, Karthik; SALIMANS, Tim; SUTSKEVER, Ilya. Improving Language Understanding by Generative Pre Training. OpenAI Technical Report, 2018.

RADFORD, Alec et al. Language Models are Unsupervised Multitask Learners. OpenAI Technical Report, 2019.

SUTSKEVER, Ilya; VINYALS, Oriol; LE, Quoc V. Sequence to Sequence Learning with Neural Networks. In: NeurIPS 2014, Advances in Neural Information Processing Systems, v. 27, 2014.

VASWANI, Ashish et al. Attention Is All You Need. In: NeurIPS 2017, Advances in Neural Information Processing Systems, v. 30, 2017.

VASWANI, Ashish et al. Tensor2Tensor for Neural Machine Translation. In: arXiv preprint arXiv:1803.07416, 2018.

WOLF, Thomas et al. Transformers: State of the Art Natural Language Processing. In: EMNLP 2020, Conference on Empirical Methods in Natural Language Processing: System Demonstrations. Association for Computational Linguistics, 2020.

ZHANG, Xiang; SUN, Xu; WEI, Wei; ZHOU, Xiaodong. Memory Augmented Self Attention for Sequence Modeling. In: AAAI 2018, Proceedings of the Thirty Second AAAI Conference on Artificial Intelligence, 2018.

CAPÍTULO 10

2018: ULMFIT E ELMO - A CAUDA DA LSTM E OS MOMENTOS ANTERIORES AOS GRANDES MODELOS DE LINGUAGEM

Embora o Transformer de Vaswani tivesse sido divulgado em junho de 2017 e sua arquitetura tivesse dispensado a parte mais complexa dos modelos de recorrência, em função dos mecanismos de atenção, uma parcela relevante da pesquisa ainda se concentrava nos modelos recorrentes, principalmente nas LSTMs examinadas nos capítulos 5, 6 e 7.

Nos meses de janeiro e fevereiro de 2018, foram divulgados dois trabalhos baseados na arquitetura de portões, memória e atualização sequencial das LSTMs: ULMFiT e ELMo.

De forma simplificada, os pesquisadores dessas iniciativas partiram de uma arquitetura recorrente já consolidada, mas procuraram modificar a maneira como o conhecimento adquirido durante o treinamento poderia ser preservado e reutilizado. Eles investiram na hipótese de que um modelo treinado sobre um corpus amplo poderia aprender regularidades linguísticas transferíveis para outros conjuntos de dados e outras tarefas, evitando treinar novamente toda a estrutura linguística desde o início para cada tarefa e reduzindo o custo de treinamento e o volume de exemplos classificados por humanos. ULMFiT e ELMo ajudaram, assim, a consolidar o *transfer learning* em NLP (*Natural Language Processing*), embora por caminhos diferentes.

O ULMFiT pegava um modelo que já havia aprendido padrões gerais da linguagem e o adaptava a uma nova tarefa. O ELMo aproveitava o que o modelo havia aprendido sobre o significado das palavras em cada contexto e transferia esse conjunto de representações contextuais para outros modelos. Vamos olhar para as melhorias dos dois modelos, o que nos ajudará a compreender como o aprendizado adquirido podia ser preservado e reutilizado.

Para compreender as contribuições do ULMFiT e do ELMo, precisamos voltar um pouco no tempo e olhar para a LSTM. Quando uma LSTM recebia uma palavra, representada pelo vetor x_t, ela também consultava duas informações trazidas do passo anterior:

$ht-1$: o estado de saída anterior;
$ct-1$: a memória acumulada.

Três portões controlavam o fluxo dessas informações:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

Ao mesmo tempo, a rede produzia uma nova informação candidata a entrar na memória:

$$\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c)$$

O vetor $\tilde{c}_t$ não era ainda a nova memória. Ele continha as informações candidatas, construídas a partir da palavra recebida e do estado de saída anterior.

O portão f_t determinava quanto da memória anterior deveria ser preservado. O portão i_t controlava quanto da nova informação candidata seria acrescentado à memória. Finalmente, o portão o_t determinava quanto da memória atualizada seria apresentado como saída.

Como a função sigmoide (σ) produzia valores entre zero e um, cada portão funcionava como uma torneira: zero fechava a passagem e um permitia a passagem completa. Valores intermediários deixavam passar apenas parte da informação.

A memória era então atualizada:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

A primeira parcela preservava uma fração da memória anterior. A segunda acrescentava uma fração da nova informação candidata. Finalmente, a LSTM produzia uma nova saída:

$$h_t = o_t \odot \tanh(c_t)$$

Esse processo se repetia palavra por palavra. A LSTM não guardava a frase literalmente; mantinha uma representação numérica continuamente atualizada daquilo que havia julgado relevante. A cada atualização, parte da informação anterior podia ser atenuada ou descartada. Com isso, um elemento que parecesse pouco importante em determinado momento poderia deixar de estar disponível quando se tornasse necessário mais adiante, o que para os pesquisadores era o problema a ser resolvido nas LSTMs.

E essa era justamente uma das vantagens do mecanismo de atenção: em vez de depender apenas de uma memória comprimida e continuamente sobrescrita, o modelo podia voltar diretamente às representações das diferentes palavras da sequência.

Na tentativa de ampliar a utilidade da arquitetura LSTM, nenhum dos dois métodos alterou radicalmente sua lógica matemática. Suas inovações, entretanto, atacavam o problema de formas diferentes. O ULMFiT procurava adaptar os parâmetros do mo-

delo sem apagar o conhecimento adquirido no pré-treinamento. O ELMo mantinha a rede pré-treinada essencialmente congelada e reutilizava suas representações internas. O ULMFiT também reduzia, por meio do concat pooling, a dependência exclusiva do último estado da sequência.

ULMFiT: controlar matematicamente o que deve mudar

O ULMFiT nasceu da aproximação entre experimentação prática e formalização acadêmica. Jeremy Howard e Sebastian Ruder encontraram uma solução para adaptar um modelo de linguagem a uma nova tarefa sem apagar abruptamente o conhecimento adquirido durante o pré-treinamento.

Jeremy Howard nasceu em Londres, em 1973, mas mudou-se ainda criança para Melbourne. Estudou filosofia na Universidade de Melbourne, com interesse particular por ética, cognição e inteligência. Portanto, não veio de uma graduação convencional em ciência da computação, matemática ou engenharia.

Trabalhou durante aproximadamente oito anos em consultoria, passando por McKinsey e A.T. Kearney, e depois fundou diversas empresas australianas. Também foi presidente e cientista chefe do Kaggle, depois de alcançar as primeiras posições em suas competições internacionais de ciência de dados. Em 2016, fundou a fast.ai com Rachel Thomas, onde continua como cofundador e pesquisador. É também cofundador do Answer.AI, laboratório de pesquisa aplicada criado com Eric Ries em 2023, e é professor honorário da Universidade de Queensland. Integra ainda o Stanford Digital Economy Lab como *Digital Fellow*.

Sebastian Ruder é cidadão alemão e estudou linguística compu-

tacional na Universidade de Heidelberg, na Alemanha. Depois, iniciou um doutorado no Insight Centre for Data Analytics, ligado à National University of Ireland Galway. Sua tese, concluída em 2019, recebeu o título *Neural Transfer Learning for Natural Language Processing*. É atualmente pesquisador da Meta, em Berlim. Seu trabalho concentra-se nas capacidades multilíngues dos LLMs e em métodos de avaliação.

Howard havia desenvolvido e apresentado em seu curso da fast.ai o método que daria origem ao ULMFiT. Segundo o próprio Howard, Ruder tomou conhecimento do trabalho por meio do curso e entrou em contato, argumentando que os resultados deveriam ser publicados. Ruder ofereceu-se para escrever o artigo, que também poderia integrar sua trajetória de doutorado, e Howard aceitou a colaboração. Dessa aproximação nasceu o artigo assinado pelos dois e divulgado em janeiro de 2018.

A principal inovação não estava nos portões da LSTM, mas no controle matemático de como o modelo deveria aprender uma nova tarefa sem destruir o conhecimento adquirido anteriormente.

A matemática por trás do ULMFiT

A contribuição do método ULMFiT estava em organizar a transferência e desenvolver mecanismos para modificar os parâmetros sem apagar abruptamente o conhecimento adquirido anteriormente. O processo possuía três etapas claramente separadas. Como a matemática da primeira etapa, o pré-treinamento geral do modelo de linguagem, já foi apresentada no início do capítulo, interessa agora observar o que o método acrescentou ao processo de transferência.

Um modelo treinado sobre textos gerais encontraria distribuições diferentes ao ser levado para um novo domínio. Avaliações de filmes, documentos jurídicos e mensagens de clientes, por exemplo, utilizam vocabulários e construções próprias.

Por isso, na segunda etapa, o ULMFiT continuava treinando o modelo de linguagem sobre os textos do novo domínio. Ainda não era necessário utilizar os rótulos da tarefa. O modelo continuava tentando prever a próxima palavra, mas agora aprendia as particularidades do conjunto de textos em que seria empregado.

Uma atualização comum dos parâmetros seria:

$$\theta_t = \theta_{t-1} - \eta \nabla_\theta J(\theta)$$

Nessa expressão, $\nabla_\theta J(\theta)$ é o gradiente da função de erro. Ele aponta para a direção em que o erro cresce mais rapidamente. O sinal negativo faz os parâmetros caminharem na direção oposta, na qual o erro tende a diminuir. O valor η, chamado taxa de aprendizagem, determina o tamanho de cada mudança, como vimos em capítulos anteriores.

O problema era que aplicar a mesma velocidade a toda a rede poderia modificar excessivamente as camadas que continham padrões linguísticos mais gerais. Para evitar isso, o ULMFiT introduziu o discriminative fine-tuning: cada camada recebia sua própria taxa de aprendizagem.

Para a camada l, a atualização passava a ser:

$$\theta_t^l = \theta_{t-1}^l - \eta^l \nabla_{\theta^l} J(\theta)$$

O artigo escolhia primeiro a taxa da camada superior, represen-

tada por η^L, e reduzia progressivamente a velocidade das camadas inferiores:

$$\eta^{l-1} = \frac{\eta^l}{2{,}6}$$

A lógica era simples. As camadas inferiores haviam aprendido padrões mais gerais da linguagem e deveriam mudar lentamente. As superiores, mais próximas da saída e da tarefa específica, poderiam mudar mais rapidamente.

O ULMFiT também modificava a taxa de aprendizagem ao longo do tempo. Em vez de manter uma velocidade constante, utilizava o slanted triangular learning rate: a taxa subia rapidamente no começo do treinamento e depois diminuía durante um período mais longo.

Se T representa o número total de atualizações e q a fração reservada ao crescimento inicial, podemos representar o ponto em que a taxa deixava de subir por:

$$c = \lfloor Tq \rfloor$$

Como c precisa identificar uma atualização inteira, o resultado era arredondado para o inteiro inferior.

A posição relativa dentro das fases de crescimento e redução era calculada por:

$$p_t = \begin{cases} \dfrac{t}{c}, & t < c \\ 1 - \dfrac{t - c}{c\left(\dfrac{1}{q} - 1\right)}, & t \geq c \end{cases}$$

A taxa de aprendizagem no momento t era então:

$$\eta_t = \eta_{\max} \frac{1 + p_t(r-1)}{r}$$

O valor r indicava quantas vezes a taxa mínima era menor que a máxima:

$$\eta_{\min} = \frac{\eta_{\max}}{r}$$

No artigo, os autores geralmente utilizaram q = 0,1 e r = 32. Isso significava que a taxa crescia durante os primeiros 10% das atualizações e diminuía durante os 90% restantes. O crescimento inicial permitia chegar rapidamente a uma região útil dos parâmetros. A redução posterior permitia fazer ajustes menores, diminuindo o risco de ultrapassar uma boa solução.

As duas ideias podiam operar simultaneamente. A camada superior seguia o ciclo de crescimento e redução, enquanto as inferiores acompanhavam o mesmo movimento com taxas progressivamente menores. Podemos representar essa relação por:

$$\eta_t^l = \frac{\eta_t^L}{(2{,}6)^{L-l}}$$

Ao final dessa etapa, o modelo ainda era um modelo de linguagem. A diferença era que agora conhecia tanto os padrões gerais aprendidos no corpus amplo quanto as particularidades linguísticas do novo domínio.

Na terceira etapa, o ULMFiT acrescentava ao modelo de lingua-

gem duas estruturas lineares destinadas à classificação. Somente essas novas estruturas começavam com parâmetros aleatórios. As camadas recorrentes traziam os parâmetros acumulados nas duas etapas anteriores.

Se a tarefa fosse classificar uma avaliação como positiva ou negativa, por exemplo, o sistema precisava transformar todos os estados produzidos durante a leitura do texto em uma representação única.

Considere o conjunto de estados da última camada da LSTM:

$$H = \{h_1, h_2, \ldots, h_T\}$$

Usar apenas h_T, o último estado, criava uma fragilidade. Uma palavra decisiva poderia aparecer no começo ou no meio do documento e perder influência durante as atualizações seguintes.

O ULMFiT enfrentava esse problema por meio do concat pooling. O método reunia três informações:

$$h_c = [h_T \,;\, \text{maxpool}(H) \,;\, \text{meanpool}(H)]$$

Nessa expressão:

- h_T era o último estado produzido pela LSTM;
- $\text{maxpool}(H)$ conservava, em cada dimensão, o maior valor encontrado ao longo do texto;
- $\text{meanpool}(H)$ calculava, em cada dimensão, a média dos estados produzidos;
- os pontos e vírgulas indicavam que os três vetores eram concatenados, isto é, colocados lado a lado

Assim, o classificador não dependia exclusivamente daquilo que restasse no final da sequência. Uma característica importante poderia permanecer no valor máximo ou influenciar a média, mesmo que tivesse perdido força no último estado.

De forma simplificada, as duas estruturas lineares acrescentadas ao modelo podiam ser representadas por:

$$z = \mathrm{ReLU}(W_1 h_c + b_1)$$

$$\hat{y} = \mathrm{softmax}(W_2 z + b_2)$$

O vetor z era uma transformação intermediária da representação do documento. A função *Softmax* convertia a saída final em probabilidades para as classes possíveis. Na implementação, essas estruturas também utilizavam normalização e dropout, omitidos aqui para manter visível a lógica principal.

Descongelar sem apagar

O momento mais delicado surgia quando o classificador e a LSTM começavam a ser ajustados conjuntamente com os exemplos rotulados. Se todas as camadas fossem liberadas ao mesmo tempo, uma adaptação agressiva poderia alterar os parâmetros gerais e eliminar parte do benefício do pré-treinamento. Esse fenômeno é chamado de esquecimento catastrófico.

Para reduzir esse risco, o ULMFiT utilizava o descongelamento gradual. Primeiro, ajustava apenas a camada superior. Depois de uma época de treinamento, liberava a camada imediatamente inferior. O processo continuava, uma camada por vez, até que toda a rede estivesse disponível para ajustes.

As camadas já liberadas continuavam sendo treinadas quando uma nova camada era acrescentada. Ao mesmo tempo, o *discriminative fine-tuning* mantinha velocidades menores nas regiões inferiores da rede. O descongelamento gradual controlava quando cada camada podia mudar, as taxas discriminativas controlavam quanto ela podia mudar, e o ciclo triangular controlava como essa velocidade variava ao longo do treinamento.

Documentos longos e duas direções

Para tornar viável a classificação de documentos longos, que não podiam ser processados de uma só vez, o ULMFiT utilizava uma adaptação da retropropagação através do tempo chamada *BPTT for Text Classification* – BPT3C. O documento era dividido em blocos. O estado final de um bloco iniciava o bloco seguinte, enquanto o sistema preservava os estados necessários para calcular os valores máximo e médio usados pelo *concat pooling*.

O artigo também treinava um modelo da esquerda para a direita e outro da direita para a esquerda. Entretanto, eles não eram combinados internamente como veremos no ELMo. Se $\hat{y}_F$ representa a previsão do classificador que lia para a frente e $\hat{y}_B$ a previsão daquele que lia para trás, o resultado era a média das duas:

$$\hat{y} = \frac{\hat{y}_F + \hat{y}_B}{2}$$

Portanto, o ULMFiT não inventou uma nova memória para a LSTM e não modificou seus portões. Sua contribuição foi organizar matematicamente a transferência do aprendizado. O modelo aprendia primeiro a linguagem geral, adaptava-se depois à

linguagem do novo domínio e, somente então, aprendia a tarefa supervisionada. Durante essa passagem, diferentes mecanismos controlavam a velocidade das alterações, o momento em que cada camada podia mudar e a maneira de reunir informações distribuídas ao longo do texto.

ELMo: abrir a LSTM e aproveitar suas camadas internas

O ELMo nasceu de uma colaboração concentrada em Seattle, nos Estados Unidos, entre o Instituto Allen para IA e a Universidade de Washington. O artigo reuniu sete pesquisadores: Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee e Luke Zettlemoyer.

Matthew Peters, matemático de formação e principal nome associado à criação do ELMo, trabalhava no Allen Institute e atualmente é cofundador da Envive, onde desenvolve modelos de linguagem para agentes comerciais. Mark Neumann, também ligado ao núcleo técnico do AI2 e do AllenNLP, transferiu posteriormente seu trabalho com representações neurais para modelos de materiais e proteínas, atuando hoje na EvolutionaryScale, incorporada ao Biohub.

Mohit Iyyer, então Young Investigator do AI2, seguiu a carreira acadêmica e tornou-se professor associado na Universidade de Maryland. Matt Gardner, doutor pela Carnegie Mellon e criador da plataforma AllenNLP, passou pela Scaled Cognition e atualmente trabalha no Microsoft AI.

Do lado da Universidade de Washington estavam os doutorandos Christopher Clark e Kenton Lee, orientados por Luke Zettlemoyer. Clark retornou posteriormente ao Allen Institute, onde

hoje pesquisa modelos que combinam linguagem e visão. Lee ingressou no Google e, poucos meses depois do ELMo, tornou-se um dos autores do BERT, constituindo a ligação mais direta entre a linhagem das LSTMs e a nova geração baseada em Transformers que veremos nos capítulos adiante.

Zettlemoyer, doutor pelo MIT e professor da Universidade de Washington, tornou-se também diretor de pesquisa na Meta. A equipe do ELMo acabou, assim, espalhando sua influência por universidades, grandes empresas de tecnologia, startups e aplicações científicas da inteligência artificial.

O grupo do ELMo não representava simplesmente uma resistência ao Transformer. Ele estava tentando resolver, com as melhores redes recorrentes disponíveis, o mesmo problema que modelos posteriores, baseados no mecanismo de atenção, resolveriam em outra escala: como treinar uma representação geral da linguagem e fazê-la circular entre diferentes tarefas.

O ELMo seguiu um caminho diferente do ULMFiT. Em vez de adaptar toda a rede para cada nova tarefa, utilizava os estados internos de uma LSTM já treinada como fonte de representações linguísticas. O artigo foi submetido em 15 de fevereiro de 2018 e recebeu o prêmio de melhor artigo longo da NAACL daquele ano.

O sistema possuía duas pilhas de camadas LSTM. A primeira percorria a frase da esquerda para a direita e tentava prever a palavra seguinte. A segunda percorria a frase no sentido contrário e tentava prever a palavra anterior. Desse modo, em uma frase como “ele foi ao banco para depositar dinheiro”, a representação de “banco” carregava informações vindas tanto das palavras anteriores quanto das posteriores. Em “ele sentou no banco da praça”,

a mesma palavra produzia outra representação.

Para cada palavra, o ELMo colocava lado a lado os estados produzidos pelas duas direções, e o resultado era uma representação que reunia o que a rede havia encontrado antes e depois daquela palavra. Esses resultados não eram guardados em um arquivo de frases prontas para consulta. O que permanecia armazenado eram os parâmetros aprendidos pelas LSTMs durante o treinamento. Diante de uma nova frase, o ELMo utilizava esses parâmetros para produzir novas representações contextuais.

O ELMo também mostrou que as diferentes camadas da LSTM aprendiam tipos distintos de informação. A representação inicial, construída a partir dos caracteres, capturava aspectos da forma das palavras. Os estados inferiores da LSTM modelavam mais aspectos da construção gramatical, enquanto os superiores reuniam informações mais dependentes do significado contextual. Para cada tarefa, o sistema aprendia quanto deveria aproveitar de cada camada e combinava essas informações em uma representação final.

Essa representação era então entregue ao modelo encarregado da nova tarefa. Um classificador de sentimentos, por exemplo, conservava sua própria arquitetura, mas passava a receber, junto com cada palavra, a informação contextual produzida pelo ELMo. O conhecimento era transferido, portanto, por meio desses vetores contextuais, sem a necessidade de treinar novamente toda a LSTM para cada aplicação.

A matemática por trás do ELMo

A matemática da LSTM permanecia a mesma: seus portões continuavam decidindo quais informações deveriam entrar, perma-

necer ou sair do estado de memória. A inovação do ELMo estava na maneira de treinar duas redes em sentidos opostos e combinar os resultados produzidos por suas diferentes camadas.

Considere uma frase formada pelas palavras:

$$w_1, w_2, \ldots, w_N$$

A LSTM que percorria o texto da esquerda para a direita aprendia a calcular a probabilidade de cada palavra a partir das anteriores:

$$P(w_k \mid w_1, \ldots, w_{k-1})$$

Na direção contrária, a segunda LSTM aprendia a calcular a probabilidade da mesma palavra utilizando as palavras posteriores:

$$P(w_k \mid w_{k+1}, \ldots, w_N)$$

O treinamento procurava reduzir simultaneamente os erros das duas direções:

$$J_{\text{biLM}} = -\sum_{k=1}^{N} [\log P(w_k \mid w_1, ..., w_{k-1}) + \log P(w_k \mid w_{k+1}, ..., w_N)]$$

Esse modelo de linguagem bidirecional era representado pela sigla *biLM* , que significa *bidirectional language model*, ou. Quanto menor fosse o valor dessa função de erro, melhor a primeira direção estaria prevendo as palavras com base no contexto anterior e melhor a segunda estaria fazendo a previsão correspondente com base no contexto posterior.

Os estados recorrentes das duas direções não eram misturados

enquanto a frase era percorrida. As LSTMs possuíam parâmetros recorrentes separados, embora as duas direções compartilhassem a representação inicial das palavras e os parâmetros da camada de saída. Cada direção produzia sua própria sequência de estados, e as duas leituras eram combinadas posteriormente.

$$\overrightarrow{h_{k,j}}$$

$$\overleftarrow{h_{k,j}}$$

O símbolo k indica a posição da palavra na frase, enquanto j identifica a camada da rede. A seta mostra o sentido em que a frase foi percorrida.

Para cada palavra e em cada camada contextual $j \geq 1$, o ELMo concatenava os dois estados:

$$h_{k,j} = [\overrightarrow{h_{k,j}} \; ; \; \overleftarrow{h_{k,j}}]$$

O ponto e vírgula representa a concatenação: os valores dos dois vetores eram colocados lado a lado, sem que um apagasse ou substituísse o outro. A representação de uma palavra passava, assim, a carregar informações extraídas dos dois lados da frase.

No caso da palavra "banco", por exemplo, a primeira direção poderia carregar informações provenientes de "ele foi ao", enquanto a direção contrária encontraria "para depositar dinheiro". A concatenação reunia as duas leituras em um único vetor contextual.

O ELMo não utilizava apenas o resultado da última camada. Para a palavra localizada na posição k, conservava o conjunto de repre-

sentações produzidas ao longo da rede:

$$R_k = \{h_{k,0}, h_{k,1}, \ldots, h_{k,L}\}$$

A representação da camada zero correspondia à forma inicial da palavra, construída a partir de seus caracteres. As demais provinham das sucessivas camadas bidirecionais da LSTM. Cada camada observava a frase a partir daquilo que havia sido produzido pela camada anterior.

Como essas camadas capturavam informações diferentes, o ELMo permitia que cada aplicação aprendesse sua própria combinação:

$$\text{ELMo}_k^{(t)} = \gamma^{(t)} \sum_{j=0}^{L} s_j^{(t)} h_{k,j}$$

O índice t representava a tarefa. Assim, uma tarefa de classificação de sentimentos e uma tarefa de identificação de pessoas e lugares poderiam construir combinações diferentes para a mesma palavra.

Os coeficientes atribuídos às camadas determinavam a participação de cada uma delas. Eles eram normalizados para que sua soma fosse igual a 1:

$$\sum_{j=0}^{L} s_j^{(t)} = 1$$

Na implementação, essa normalização era obtida por uma função Softmax. Podemos explicitar essa normalização por meio da seguinte expressão:

$$s_j^{(t)} = \frac{e^{a_j^{(t)}}}{\sum_{m=0}^{L} e^{a_m^{(t)}}}$$

Os valores livres associados a cada camada eram ajustados durante o treinamento da nova tarefa. Depois de convertidos pela Softmax, transformavam-se nos pesos atribuídos às diferentes camadas.

Se determinada tarefa aprendesse, por exemplo, os pesos 0,15, 0,55 e 0,30, sua representação final utilizaria 15% da camada inicial, 55% da primeira camada contextual e 30% da camada superior. Esses números são apenas ilustrativos: cada aplicação aprendia seus próprios valores.

O coeficiente γ controlava a escala do vetor final. Ele permitia aumentar ou diminuir a intensidade da contribuição do ELMo em relação às demais informações utilizadas pelo modelo da tarefa.

Uma das formas de integração consistia em concatenar o vetor contextual à representação que o novo modelo já atribuía à palavra:

$$\tilde{x}_k = \left[x_k;\ \mathrm{ELMo}_k^{(t)} \right]$$

A representação original empregada pelo modelo da tarefa era preservada. O vetor produzido pelo ELMo entrava como informação adicional. O classificador ou sistema de respostas continuava sendo treinado normalmente, mas agora recebia cada palavra acompanhada do contexto calculado pela rede bidirecional.

Durante esse processo, os parâmetros do modelo bidirecional pré-treinado permaneciam congelados. A nova tarefa aprendia seus próprios parâmetros, os pesos atribuídos às camadas e o fator de escala. Portanto, o ELMo transferia conhecimento sem copiar frases nem treinar novamente o modelo bidirecional.

A transferência ocorria por meio dos vetores produzidos. A LSTM pré-treinada funcionava como uma fábrica matemática de representações contextuais, enquanto cada aplicação aprendia quais partes dessa produção eram mais úteis para o seu próprio problema.

Aplicações práticas e legado comercial de ULMFiT e ELMo

Os métodos ULMFiT e ELMo não foram comercializados como produtos independentes, mas foram incorporados aos bastidores de sistemas de análise textual. O ULMFiT permitia adaptar um modelo de linguagem inteiro a um domínio com poucos dados e menor custo computacional do que seria necessário para treinar toda a estrutura desde o início, enquanto o ELMo fornecia representações contextuais que podiam ser acrescentadas a classificadores, mecanismos de busca e sistemas de extração já existentes.

O legado comercial, porém, foi demonstrar que o conhecimento adquirido durante o pré-treinamento poderia ser reaproveitado, reduzindo a quantidade de dados classificados e o custo necessário para a criação de novas aplicações de linguagem.

O ULMFiT, com seu formato de transferência dos parâmetros e possibilidade de ajustar o modelo inteiro para uma nova tarefa, ajudou a consolidar o caminho que se tornaria dominante para os modelos que viriam a seguir, GPT e BERT: pré-treinamento so-

bre textos não rotulados, seguido de fine-tuning para uma tarefa específica.

O GPT-1, por exemplo, substituiu a AWD-LSTM pelo Transformer, mas preservou os princípios centrais do ULMFiT: treinar um modelo de linguagem sobre grande quantidade de texto não rotulado, acrescentar uma pequena estrutura para a tarefa desejada e ajustar os parâmetros do modelo com dados rotulados. O próprio artigo do GPT identifica Howard e Ruder como parte da linha de trabalhos em modelos de linguagem pré-treinados e posteriormente ajustados para classificação. A diferença declarada é que o GPT troca a LSTM pelo Transformer para alcançar relações linguísticas mais longas. O GPT adotou o princípio mais importante do ULMFiT, que o conhecimento podia ser transferido ajustando os próprios parâmetros de um modelo de linguagem pré-treinado.

O ELMo, que mantinha o modelo congelado e fornecia seus estados internos como vetores para outra arquitetura, reapareceu no uso de Transformers como extratores de representações contextuais.

O BERT, modelo posterior que será explorado nos capítulos seguintes, aproximou características dessas duas linhas. Assim como o ULMFiT, permitia ajustar os parâmetros do modelo pré-treinado para uma nova tarefa. Ao mesmo tempo, assim como o ELMo, produzia uma representação diferente para cada ocorrência de uma palavra, de acordo com o contexto. A diferença era que o BERT construía essa contextualização simultaneamente nos dois sentidos, dentro de todas as camadas do Transformer, e não por meio de duas redes recorrentes independentes.

Não se trata apenas de uma interpretação histórica. O próprio

artigo do BERT separa explicitamente as duas estratégias então existentes: a abordagem *feature-based*, representada pelo ELMo, e a abordagem de *fine-tuning*, representada diretamente pelo GPT por herança do ULMFiT. Na revisão dessa segunda linha, o artigo também inclui o trabalho de Howard e Ruder.

Em outras palavras, ULMFiT e ELMo foram a última importante ponte entre a primavera da Inteligência Artificial e a expansão dos grandes modelos de linguagem disponíveis ao usuário leigo. Na periodização adotada neste livro, essa primavera começa no intervalo entre o surgimento da LSTM, em 1997, e o modelo neural de linguagem de Bengio, em 2003. ULMFiT e ELMo não serviram apenas como ponte: transportaram um conjunto importante de técnicas para preservar, adaptar e transferir o aprendizado. Esse percurso técnico ajudaria a formar a linhagem que, depois do lançamento do ChatGPT, alcançaria uma das maiores e mais rápidas curvas de adoção da história da tecnologia moderna.

Referências bibliográficas

BICK, Alexander; BLANDIN, Adam; DEMING, David J. The Rapid Adoption of Generative AI. NBER Working Paper, n. 32966. National Bureau of Economic Research, 2024.

DEVLIN, Jacob; CHANG, Ming-Wei; LEE, Kenton; TOUTANOVA, Kristina. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: NAACL-HLT 2019, Conference of the North American Chapter of the Association for Computational Linguistics. Association for Computational Linguistics, 2019. p. 4171–4186.

GERS, Felix A.; SCHMIDHUBER, Jürgen; CUMMINS, Fred.

Learning to Forget: Continual Prediction with LSTM. Neural Computation, v. 12, n. 10, p. 2451–2471, 2000.

HOCHREITER, Sepp; SCHMIDHUBER, Jürgen. Long Short-Term Memory. Neural Computation, v. 9, n. 8, p. 1735–1780, 1997.

HOWARD, Jeremy; RUDER, Sebastian. Universal Language Model Fine-tuning for Text Classification. In: ACL 2018, Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics, 2018. p. 328–339. Preprint arXiv:1801.06146, submetido em 18 jan. 2018.

HU, Krystal. ChatGPT Sets Record for Fastest-Growing User Base: Analyst Note. Reuters, 2 fev. 2023.

MERITY, Stephen; KESKAR, Nitish Shirish; SOCHER, Richard. Regularizing and Optimizing LSTM Language Models. In: ICLR 2018, International Conference on Learning Representations, 2018.

PETERS, Matthew E.; NEUMANN, Mark; IYYER, Mohit; GARDNER, Matt; CLARK, Christopher; LEE, Kenton; ZETTLEMOYER, Luke. Deep Contextualized Word Representations. In: NAACL-HLT 2018, Conference of the North American Chapter of the Association for Computational Linguistics. Association for Computational Linguistics, 2018. p. 2227–2237. Preprint arXiv:1802.05365, submetido em 15 fev. 2018.

RADFORD, Alec; NARASIMHAN, Karthik; SALIMANS, Tim; SUTSKEVER, Ilya. Improving Language Understanding by Generative Pre-Training. OpenAI Technical Report, 2018.

RUDER, Sebastian; PETERS, Matthew E.; SWAYAMDIPTA, Swabha; WOLF, Thomas. Transfer Learning in Natural Language Processing. In: NAACL-HLT 2019: Tutorials. Association for Computational Linguistics, 2019. p. 15–18.

THE LATENT SPACE. The End of Finetuning: With Jeremy Howard of fast.ai. Entrevista com Jeremy Howard, 19 out. 2023.

VASWANI, Ashish et al. Attention Is All You Need. In: NeurIPS 2017, Advances in Neural Information Processing Systems, v. 30, 2017. p. 5998–6008.


ISBN: 978-65-02-21237-0
CBL
9 786502 212370