**UNIVERSIDADE DE SÃO PAULO**
**Instituto de Ciências Matemáticas e de Computação**
**Departamento de Estatística.**

**Luiz Gustavo de Sousa Bravo**

# Um estudo de Redes Neurais e Aprendizado Profundo com aplicações em processamento de linguagem natural

**São Carlos**

**2024**

**Luiz Gustavo de Sousa Bravo**

# Um estudo de Redes Neurais e Aprendizado Profundo com aplicações em processamento de linguagem natural

Trabalho de conclusão de curso entregue como parte dos requisitos de avaliação na disciplina de Projeto de Graduação em Estatística e Ciência de Dados do Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo - ICMC/USP.

Área de Concentração: Estatística, Ciência de Dados e Inteligência Artificial, Processamento de Linguagem Natural

Orientador: Francisco Aparecido Rodrigues

**Versão original**

**São Carlos**

**2024**

# RESUMO

BRAVO, L. G. S. **Um estudo de Redes Neurais e Aprendizado Profundo com aplicações em processamento de linguagem natural**. 2024. 27p. Trabalho de Conclusão de Curso em em Estatística e Ciência de Dados - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2024.

Este trabalho tem como objetivo explorar o campo do Processamento de Linguagem Natural (PLN) utilizando Redes Neurais e Aprendizado Profundo, áreas que não foram amplamente abordadas durante o bacharelado em Estatística e Ciência de Dados. A metodologia incluiu uma análise detalhada de Redes Neurais Superficiais e Profundas, com foco em modelos Transformers, especialmente BERT e RoBERTa. Para a avaliação, foram utilizados cálculos de autoatenção e uma comparação entre os dois modelos Transformers, destacando suas diferenças no pré-treinamento e suas respectivas vantagens e desvantagens. Os resultados foram obtidos a partir de um banco de dados específico para análise de sentimentos, onde foram aplicadas métricas de desempenho para avaliar a eficácia dos modelos. Conclui-se que tanto BERT quanto RoBERTa apresentam potencial significativo para aplicações em PLN, com variações em desempenho de acordo com a natureza do pré-treinamento.

**Palavras-chave**: Processamento de Linguagem Natural. Redes Neurais. Redes Neurais Profundas. Transformers. BERT. RoBERTa. Análise de Sentimentos.

## ABSTRACT

BRAVO, L. G. S. **A study of Neural Networks and Deep Learning with applications in natural language processing**. 2024. 27p. Course Completion Work Bachelor in Statistics and Data Science - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2024.

This work aims to explore the field of Natural Language Processing (NLP) using Neural Networks and Deep Learning, areas not extensively covered during the undergraduate program in Statistics and Data Science. The methodology included a detailed analysis of Shallow and Deep Neural Networks, focusing on Transformer models, specifically BERT and RoBERTa. The evaluation involved scalar product self-attention calculations and a comparison between the two Transformer models, highlighting their differences in pre-training as well as their respective advantages and disadvantages. The results were obtained from a specific dataset for sentiment analysis, where performance metrics were applied to assess model efficacy. It is concluded that both BERT and RoBERTa present significant potential for NLP applications, with performance variations according to pre-training characteristics.



**Keywords**: Natural Language Processing. Neural Networks. Transformers. BERT. RoBERTa. Sentiment Analysis.

# SUMÁRIO

# 1 INTRODUÇÃO

Nos últimos anos, a área de Processamento de Linguagem Natural (PLN) tem passado por uma revolução graças à evolução dos modelos de deep learning, especificamente os modelos conhecidos como transformers. Os transformers surgiram em 2017, introduzidos no artigo "Attention Is All You Need" (VASWANI *et al.*, 2017), e desde então tornaram-se o principal paradigma para uma variedade de tarefas em PLN, como tradução automática, geração de texto, análise de sentimentos e respostas a perguntas. Eles representam uma mudança significativa em relação aos métodos tradicionais, baseados em redes neurais recorrentes (RNNs) e Long Short-Term Memory (LSTM).

O conceito de *Recurrent Neural Networks* (RNNs) foi popularizado como modelo de linguagem por Mikolov et al. (MIKOLOV *et al.*, 2010). Mais tarde, as *Long Short-Term Memory* (LSTMs) foram introduzidas por Hochreiter e Schmidhuber (HOCHREITER; SCHMIDHUBER, 1997) para lidar com o problema de gradientes em redes recorrentes.

A ideia de "esquecimento"foi explorada por Gers et al. (GERS; SCHMIDHUBER; CUMMINS, 1999), e sua aplicação em modelagem de linguagem foi demonstrada em trabalhos como Sundermeyer et al. (SUNDERMEYER; SCHLÜTER; NEY, 2012).

Nos anos 2010, modelos como RNNs e LSTMs foram amplamente utilizados para lidar com dados sequenciais, especialmente texto. Embora eficientes em algumas tarefas, esses modelos enfrentavam desafios em termos de paralelização e captura de dependências de longo alcance em sequências extensas.

Os transformers superaram essas limitações ao introduzir o mecanismo de atenção, que permite que o modelo se concentre em diferentes partes de uma sequência de dados ao processar uma entrada. O mecanismo de atenção possibilita a análise contextual de palavras em um texto, o que melhora a compreensão e a geração de linguagem natural.

Com o desenvolvimento do modelo GPT (Generative Pre-trained Transformer) (RADFORD *et al.*, 2018) e o BERT (Bidirectional Encoder Representations from Transformers) (DEVLIN *et al.*, 2018), os transformers provaram seu potencial para produzir modelos pré-treinados em grandes corpora de texto, podendo posteriormente ser adaptados (*fine-tuned*) para tarefas específicas.

Essa abordagem de pré-treinamento e ajuste fino revolucionou a forma como lidamos com problemas em PLN e deu origem a modelos ainda mais avançados, como GPT-3 (BROWN *et al.*, 2020), T5 (RAFFEL *et al.*, 2020) e RoBERTa (LIU *et al.*, 2019), além dos modelos multimodais que incorporam dados de diferentes fontes, como texto e imagem.

## 1.1 Revisão Bibliográfica

Este trabalho baseou-se integralmente no livro, *"Understanding Deep Learning"*, Prince (2023) o mesmo apresenta em sua primeira parte os modelos de **deep learning**, discutindo como treiná-los, medir seu desempenho e aprimorá-los. Posteriormente, são introduzidas arquiteturas especializadas para diferentes tipos de dados, como imagens, textos e grafos, possibilitando uma compreensão mais abrangente das aplicações. Já as seções subsequentes do livro exploram modelos generativos e aprendizado por reforço.

A escolha deste livro como referência bibliográfica se deu devido à sua abordagem concisa e focada nos conceitos essenciais, permitindo uma compreensão eficiente do material apresentado. Os apêndices fornecem uma revisão dos pré-requisitos matemáticos, eliminando a necessidade de recorrer a fontes externas para entender os conceitos apresentados.

## 1.2 Motivação-Justificativa do Tema

A crescente disponibilidade de dados textuais e a necessidade de interpretá-los impulsionam o estudo de Redes Neurais e Aprendizado Profundo em Processamento de Linguagem Natural (PLN). Essas técnicas, apoiadas por conceitos estatísticos, oferecem avanços significativos na compreensão e geração de linguagem, essenciais em áreas como tradução automática, análise de sentimentos e atendimento ao cliente. A escolha deste tema é motivada pela busca por soluções eficientes e precisas em PLN, fundamentadas em métodos estatísticos para análise e modelagem de dados. Justifica-se pela relevância dessas tecnologias, que aliam estatística e aprendizado profundo, transformando a interação humana com sistemas computacionais em diversos setores.

## 1.3 Objetivos

### Objetivo

O objetivo geral deste trabalho é aprofundar o estudo do Processamento de Linguagem Natural (PNL) com foco em Redes Neurais e Aprendizado Profundo, abordagens que não foram amplamente exploradas durante o bacharelado em Estatística e Ciência de Dados. Especificamente, o estudo concentra-se em modelos baseados em transformadores, como **BERT** e **RoBERTa**, com o intuito de compreender suas arquiteturas, capacidades e aplicações em tarefas de PNL.

Este trabalho busca demonstrar os benefícios do uso de modelos pré-treinados em tarefas como a classificação de sentimentos, destacando como essas técnicas revolucionaram o campo ao permitir a aplicação eficiente de modelos robustos em cenários diversos, mesmo com conjuntos de dados de tamanho moderado. Para isso, propõe-se um experimento prático que avalia e compara o desempenho dos modelos **BERT** e **RoBERTa** utilizando

a base de dados IMDb, fornecendo uma análise detalhada de suas métricas e orientando a escolha do modelo mais adequado para diferentes aplicações.

## 1.4 Organização

Nos capítulos iniciais, são apresentados os conceitos fundamentais de Redes Neurais, abordando tanto as Redes Neurais Superficiais quanto as Profundas, com o objetivo de construir uma base sólida para o entendimento das técnicas mais avançadas. Em sequência, o estudo se concentra nos modelos Transformers - BERT e RoBERTa, uma das abordagens mais sofisticadas em Processamento de Linguagem Natural (PLN). Posteriormente, são discutidas as principais métricas de desempenho empregadas na avaliação da eficácia de modelos em tarefas de PLN, com exemplos práticos que auxiliam na interpretação dessas métricas. Essa estrutura busca oferecer uma compreensão abrangente sobre o uso do Aprendizado Profundo em problemas relacionados à linguagem natural, evidenciando tanto os fundamentos teóricos quanto as aplicações práticas dessas técnicas.

# 2 METODOLOGIA

## 2.1 Redes Neurais Superficiais

Introduzida por (MCCULLOCH; PITTS, 1943), uma rede neural superficial, também conhecida como rede neural artificial simples, representa o tipo mais básico de rede neural. Essa arquitetura é caracterizada pela presença de uma única camada oculta entre a camada de entrada e a de saída. Compreender o funcionamento dessas redes é essencial para entender redes neurais profundas, que expandem essa estrutura ao incluir múltiplas camadas ocultas.

As redes neurais tem uma estrutura básica, composta por:

- **Camada ou Vetor de entrada**: recebe as variáveis de entrada, que podem ser univariadas $X$=$(X_1, ..., X_n)$ ou multivariadas $X$=$(X_{11}, ..., X_{nm})$ .
- **Camada oculta**: composta por um conjunto de neurônios que realizam operações matemáticas, combinando linearmente as entradas $z = XW + b$, onde $W$ é uma matriz de pesos e $b$ é vetor de *bias*, normalmente $b$ é inicializado com 0 ou um valor muito pequeno, aplicando uma função de ativação não linear $f(z)$, como a Sigmóide (Regressão Logística), a ReLU (Unidade Linear Retificada) etc. Essa camada permite que a rede modele relações complexas entre as variáveis de entrada, para obter o valor predito $\hat{y}$ na camada de saída.
- **Camada de saída**: gera a predição $\hat{y}$, que pode ser um vetor $(\hat{y}_1, ..., \hat{y}_n)$ ou um escalar $\hat{y}$ dependendo da natureza do problema.

Podemos visualizar sua estrutura na Figura 1 abaixo:

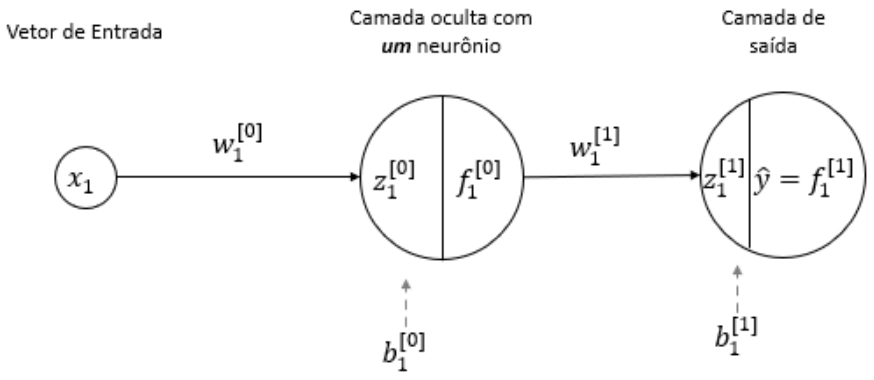


Figura 1 – Exemplo de uma rede neural superficial com uma camada oculta e um neurônio

Embora as redes neurais superficiais sejam relativamente simples, elas possuem a capacidade de se aproximar matematicamente de uma grande variedade de funções, sustentado pelo **Teorema da Aproximação Universal** com a seguinte definição:

> Dada uma função contínua $f : R^n \to R$ e qualquer valor de erro $\epsilon > 0$, existe uma rede neural com uma única camada oculta, com um número finito de neurônios e uma função de ativação adequada, tal que a saída da rede $g(x)$ aproxima $f(x)$ com precisão arbitrária, ou seja, para todos os $x$ no domínio de $f$,
>
> $$|f(x) - g(x)| < \epsilon.$$

Isso foi provado por (CYBENKO, 1989) para uma classe de ativações sigmoides e, posteriormente, demonstrado como verdadeiro para uma classe maior de funções de ativação não lineares (HORNIK, 1991). Esse teorema demonstra o poder das redes neurais, mostrando que elas podem, em teoria, resolver uma ampla gama de problemas se forem definidos um número de camadas e de neurônios adequados.

O conceito de redes neurais artificiais está intrinsecamente relacionado ao uso de funções compostas, representadas matematicamente como $y = g(f(z))$, onde $z = w_1x_1 + \cdots + w_nx_n + b$ é uma combinação linear dos pesos $(W)$ e entradas $(X)$, como ilustrado na Figura 1. Essas funções permitem modelar relações complexas entre os dados de entrada e saída.

Considere uma rede neural com apenas uma camada oculta e um único neurônio, onde desejamos estimar $y$. Neste caso, temos uma única entrada $x_1$, associada a dois pesos: $w_1^{[0]}$, aplicado antes da ativação da primeira camada oculta, e $w_1^{[1]}$, aplicado após a ativação na camada de saída. O índice entre colchetes $l = \{0, 1\}$ indica o número da camada, com $l = 0$ correspondendo à camada de entrada (oculta) e $l = 1$ à camada de saída, além disso vamos introduzir um valor $b^{[l]}$ chamado bias que é ajustado durante o treinamento, assim como os pesos, para minimizar a função de custo e, consequentemente, melhorar a previsibilidade do modelo, sendo atualizado por meio do backpropagation. A função de ativação escolhida para exemplificar é a distribuição logística:

$$f(x) = \frac{1}{1 + e^{-z}} \tag{2.1}$$

A distribuição logística mapeia qualquer valor real para o intervalo $(0, 1)$, o que a torna útil para modelar probabilidades.

Agora, calculamos o valor intermediário $z_1^{[0]}$ para a camada de entrada:

$$z_1^{[0]} = x_1 w_1^{[0]} + b_1^{[0]} \tag{2.2}$$

Em seguida, aplicamos a função de ativação:

$$a_1^{[0]} = f^{[0]}(z_1^{[0]}) \tag{2.3}$$

Esse resultado representa a saída da camada oculta. Prosseguimos para calcular $\hat{y}$ da camada de saída:

$$z_1^{[1]} = a_1^{[0]} w_1^{[1]} + b_1^{[1]} \tag{2.4}$$

E, novamente, aplicamos a função de ativação:

$$\hat{y} = a_1^{[1]} = f^{[1]}(z_1^{[1]}) \tag{2.5}$$

Portanto, a saída final do modelo pode ser representada como uma função composta:

$$\hat{y} = f^{[1]}(f^{[0]}(z_1^{[0]})) \tag{2.6}$$

Agora vamos generalizar uma rede neural superficial com um vetor de entrada $X = (x_1, \cdots, x_m)$ e uma camada oculta com $n$ neurônios.

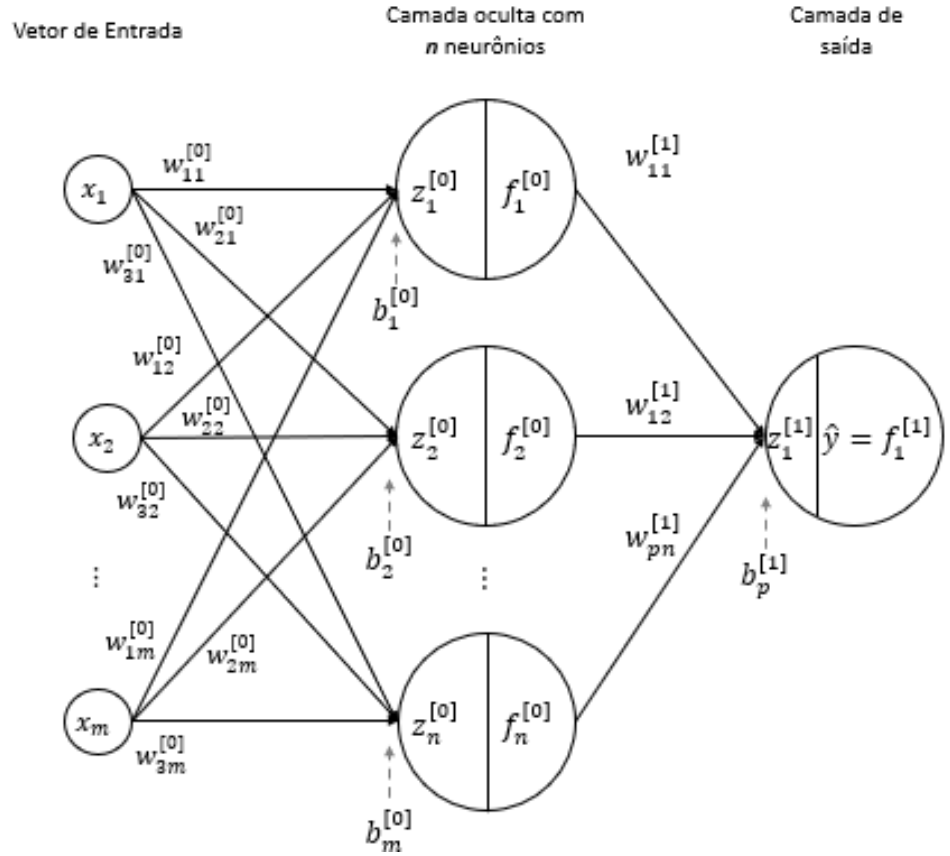


Figura 2 – Exemplo de uma rede neural superficial com uma camada oculta e $n$ neurônios

Com essa compreensão, a partir da Figura 2 podemos representar uma rede neural superficial na forma matricial, como:

$$\hat{\mathbf{y}} = f^{[1]}\left(\mathbf{W}^{[1])} \cdot f^{[0]}\left(\mathbf{W}^{[0]} \cdot \mathbf{x} + \mathbf{b}^{[0]}\right) + \mathbf{b}^{[1]}\right) \tag{2.7}$$

onde:

- $\mathbf{x} \in R^m$: vetor de entrada com $m$ dimensões.
- $\mathbf{W}^{[0]} \in R^{n \times m}$: matriz de pesos entre a entrada e a camada oculta.
- $\mathbf{b}^{[0]} \in R^n$: vetor de bias da camada oculta.
- $f^{[0]}$: função de ativação aplicada na camada oculta.
- $\mathbf{W}^{[1]} \in R^{p \times n}$: matriz de pesos entre a camada oculta e a camada de saída.
- $\mathbf{b}^{[1]} \in R^p$: vetor de bias da camada de saída.
- $f^{[1]}$: função de ativação aplicada na camada de saída.
- $\mathbf{y} \in R^p$: vetor de saída com $p$ dimensões.

Para melhor visualização, as matrizes e vetores podem ser representados como:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, \quad \mathbf{W}^{[0]} = \begin{bmatrix} w_{11}^{[0]} & w_{12}^{[0]} & \cdots & w_{1m}^{[0]} \\ w_{21}^{[0]} & w_{22}^{[0]} & \cdots & w_{2m}^{[0]} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1}^{[0]} & w_{n2}^{[0]} & \cdots & w_{nm}^{[0]} \end{bmatrix}, \quad \mathbf{b}^{[0]} = \begin{bmatrix} b_1^{[0]} \\ b_2^{[0]} \\ \vdots \\ b_n^{[0]} \end{bmatrix}$$

$$\mathbf{W}^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} & \cdots & w_{1m}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} & \cdots & w_{2m}^{[1]} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1}^{[1]} & w_{n2}^{[1]} & \cdots & w_{nm}^{[1]} \end{bmatrix}, \quad \mathbf{b}^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_p^{[1]} \end{bmatrix}$$

Essa representação cobre o fluxo de informações da entrada até a saída, considerando todos os pesos, bias e funções de ativação.

Até este ponto, discutimos o funcionamento do *feedforward* em redes neurais, detalhando como as informações percorrem as camadas até a obtenção da saída final. No entanto, não abordaremos neste trabalho os tópicos relacionados ao *backpropagation* e às funções de custo, que são fundamentais para o ajuste dos pesos e o treinamento de redes neurais. Essas técnicas, amplamente exploradas na literatura (RUMELHART; HINTON; WILLIAMS, 1986), estão fora do escopo desta análise, que se concentra exclusivamente nos aspectos estruturais e de fluxo da rede.

O *backpropagation* é um método que utiliza o cálculo do gradiente das funções de custo em relação aos pesos da rede neural, aplicando a regra da cadeia para propagar erros da camada de saída para as camadas internas. Ele é amplamente usado para ajustar os pesos durante o treinamento e minimizar uma **função de custo**.

## 2.2 Redes Neurais Profundas

Diferentemente das redes neurais superficiais, que possuem apenas uma camada oculta as redes neurais profundas, se caracterizam por possuírem múltiplas camadas ocultas.

À medida que o número de neurônios aumenta, as redes neurais superficiais aprimoram sua capacidade de modelagem. De fato, com um número suficiente de neurônios, elas podem aproximar funções arbitrariamente complexas em dimensões elevadas. No entanto, para certas funções, o número necessário de neurônios em uma rede superficial pode tornar-se inviável. A partir disso, redes profundas oferecem uma alternativa mais eficiente, pois ao invés de aumentar o número de neurônios numa mesma camada, criamos camadas adicionais, ampliando a gama de funções que pode representar.

A partir da composição de duas redes superficiais Figura 3, onde a saída da primeira se torna a entrada da segunda, ou seja, $\hat{y}' = f_t^{[2]}(\hat{y})$, onde $t = (1, 2, \cdots, k)$. A primeira rede recebe uma entrada $x$ e produz uma saída $\hat{y}$,neste caso, temos um caso especial de um rede neural profunda com três camadas ocultas.

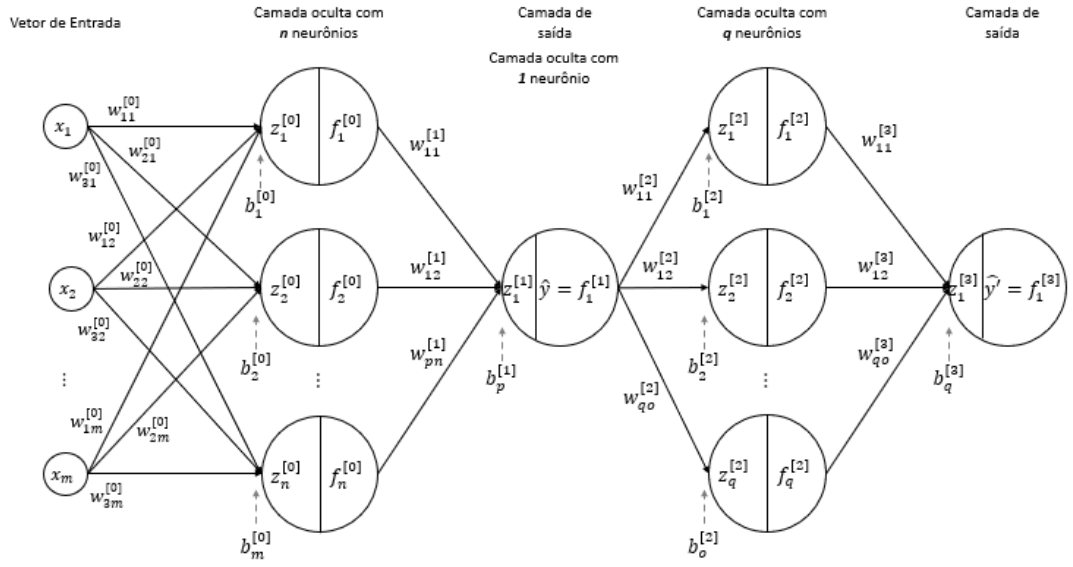


Figura 3 – Exemplo de uma rede neural profunda com três camadas ocultas e $n, 1$ e $q$ neurônios respectivamente.

A partir das equação 2.7, podemos estendê-la para a rede neural da Figura 3, com três camadas ocultas, como:

$$\hat{\mathbf{y}} = f^{[3]}\left(\mathbf{W}^{[3]} \cdot f^{[2]}\left(\mathbf{W}^{[2]} \cdot f^{[1]}\left(\mathbf{W}^{[1]} \cdot f^{[0]}\left(\mathbf{W}^{[0]} \cdot \mathbf{x} + \mathbf{b}^{[0]}\right) + \mathbf{b}^{[1]}\right) + \mathbf{b}^{[2]}\right) + \mathbf{b}^{[3]}\right) \quad (2.8)$$

Com isso, podemos generalizar uma rede neural profunda com $L$ camadas ocultas por:

$$\hat{\mathbf{y}} = f^{[L+1]}\left(\mathbf{W}^{[L+1]} \cdot f^{[L]}\left(\mathbf{W}^{[L]} \cdot \ldots f^{[1]}\left(\mathbf{W}^{[1]} \cdot f^{[0]}\left(\mathbf{W}^{[0]} \cdot \mathbf{x} + \mathbf{b}^{[0]}\right) + \mathbf{b}^{[1]}\right) \cdots + \mathbf{b}^{[L]}\right) + \mathbf{b}^{[L+1]}\right) \quad (2.9)$$

Em Redes Neurais Profundas o número de neurônios em cada camada (denominado **largura**) e o número total de camadas ocultas (denominado **profundidade**) são hiperparâmetros que afetam a capacidade da rede. Redes profundas modernas podem ter centenas de camadas com milhares de neurônios em cada camada, tornando-as altamente flexíveis e capazes de modelar funções complexas.

Assim como em redes neurais superficiais, discutimos o funcionamento do *feedforward* em redes neurais profundas, detalhando como as informações percorrem as camadas até a obtenção da saída final e também, não abordaremos os tópicos relacionados ao *backpropagation* e às funções de custo para redes neurais profundas.

O livro *Deep Learning* (GOODFELLOW; BENGIO; COURVILLE, 2016) é uma referência padrão em aprendizado profundo, incluindo uma explicação abrangente e detalhada, no Capítulo 6, sobre backpropagation e funções de custo no contexto de redes neurais profundas.

## 2.3 Transformers

### 2.3.1 Arquitetura dos Transformers

Considere a seguinte passagem presente em (PRINCE, 2023):

> *"O restaurante se recusou a me servir um sanduíche de presunto porque só cozinha comida vegetariana. No final, deram-me apenas duas fatias de pão. O ambiente deles era tão bom quanto a comida e o serviço."*

O objetivo é projetar uma rede neural capaz de processar este texto e representá-lo de maneira adequada para tarefas subsequentes, como classificar a avaliação como positiva ou negativa, ou responder a perguntas como “O restaurante serve bife?”.

Podemos fazer três observações imediatas sobre os desafios envolvidos neste processo:

1. **Dimensionalidade da entrada**: Neste exemplo, cada uma das 37 palavras pode ser representada por um vetor de incorporação com 1024 dimensões. Isso resulta em uma entrada com $37 \times 1024 = 37.888$ elementos. Para textos mais longos, redes totalmente conectadas tornam-se impraticáveis devido ao número de parâmetros.

2. **Comprimento variável da entrada**: Em problemas de Processamento de Linguagem Natural (PLN), as entradas (textos) possuem tamanhos(comprimentos) variados, não sendo trivial aplicar redes totalmente conectadas. Esse desafio sugere que a rede precisa compartilhar parâmetros entre palavras em diferentes posições, de forma semelhante ao que ocorre em redes convolucionais.

3. **Ambiguidade da linguagem**: A linguagem é inerentemente ambígua. Por exemplo, na frase dada, o substantivo *ambiente* se refere ao restaurante e não ao sanduíche de presunto. Para compreender corretamente o texto, a palavra *ambiente* precisa estar de alguma forma conectada ao *restaurante*. No contexto dos transformers, essa conexão é realizada por meio do mecanismo de atenção, que estabelece dependências entre palavras ao longo do texto.

Os transformers têm duas características importantes, a primeira é a capacidade de lidar com passagens longas e com diferentes comprimentos de texto e a segunda é a habilidade de capturar as relações entre palavras com base no próprio contexto delas.

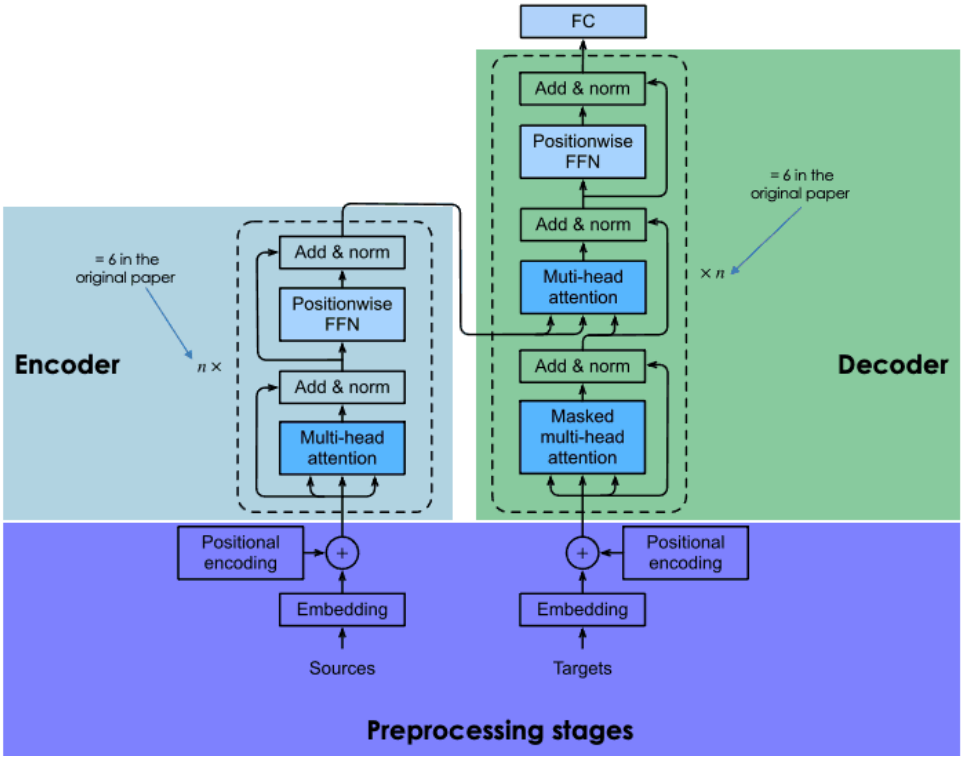


Figura 4 – Exemplo de uma arquitetura de um transformer, retirada do site: https://www.datacamp.com/pt/blog/what-is-bert-an-intro-to-bert-models

.

A base dos LLMs é tem a arquitetura apresentada na Figura 4, introduzida por (VASWANI *et al.*, 2017) em 2017. O Transformer utiliza mecanismos de atenção, camadas

de normalização e redes feedforward para processar dados sequenciais sem as limitações de modelos recorrentes. Os principais componentes são:

- **Embeddings:** O primeiro estágio de um LLM transforma tokens textuais em representações vetoriais densas usando *embeddings*. Esses vetores são somados a embeddings posicionais para introduzir informações de ordem na sequência.
- **Mecanismo de Atenção:** O mecanismo de atenção é o núcleo do Transformer, permitindo que o modelo atribua pesos diferentes a partes distintas da entrada.
- **Camadas Feedforward e Normalização:** Cada bloco Transformer contém uma rede feedforward aplicada a cada posição de forma independente, seguida por uma camada de normalização. Essas operações garantem a estabilidade do treinamento e aumentam a capacidade de aprendizado.
- **Cabeças de Atenção Múltiplas:** A atenção multi-cabeças permite que o modelo foque em diferentes partes da sequência em paralelo.

Embora poderosos, os LLMs enfrentam desafios como o alto custo computacional para treinamento e inferência, propagação de vieses presentes nos dados de treinamento e interpretação limitada das decisões do modelo.

### 2.3.2 Mecanismo de Autoatenção

Em uma camada de rede neural tradicional, chamada $f[x]$, recebemos uma entrada no formato $D \times 1$ (indicada por $x$) e aplicamos uma transformação linear seguida de uma função de ativação, como a ReLU. Formalmente, isso é representado assim:

$$f[x] = \text{ReLU}[\beta + \Omega x], \tag{2.10}$$

onde $\beta$ contém os vieses e $\Omega$ contém os pesos.

Por outro lado, um bloco de autoatenção, denotado por $sa[\cdot]$, recebe $N$ entradas $x_1, \ldots, x_N$, cada uma de dimensão $D \times 1$, e retorna $N$ vetores de saída do mesmo tamanho. No contexto de PLN (Processamento de Linguagem Natural), cada entrada representa uma palavra ou fragmento de palavra. O primeiro passo é calcular um conjunto de valores para cada entrada:

$$v_m = \beta_v + \Omega_v x_m, \tag{2.11}$$

onde $\beta_v \in R^{D\times 1}$ e $\Omega_v \in R^{D\times D}$ representam os vieses e os pesos, respectivamente.

A n-ésima saída, $sa_N[x_1, \ldots, x_N]$, é então uma soma ponderada de todos os valores $v_1, \ldots, v_N$:

$$sa_N[x_1, \ldots, x_N] = \sum_{m=1}^{N} a[x_m, x_n] v_m. \tag{2.12}$$

O peso escalar $a[x_m, x_n]$ é a atenção que a enésima saída dá à entrada $x_m$. Os $N$ pesos $a[\cdot, x_n]$ são não negativos e somam 1. Assim, a autoatenção pode ser entendida como o encaminhamento dos valores em diferentes proporções para criar cada resultado (Figura 5).

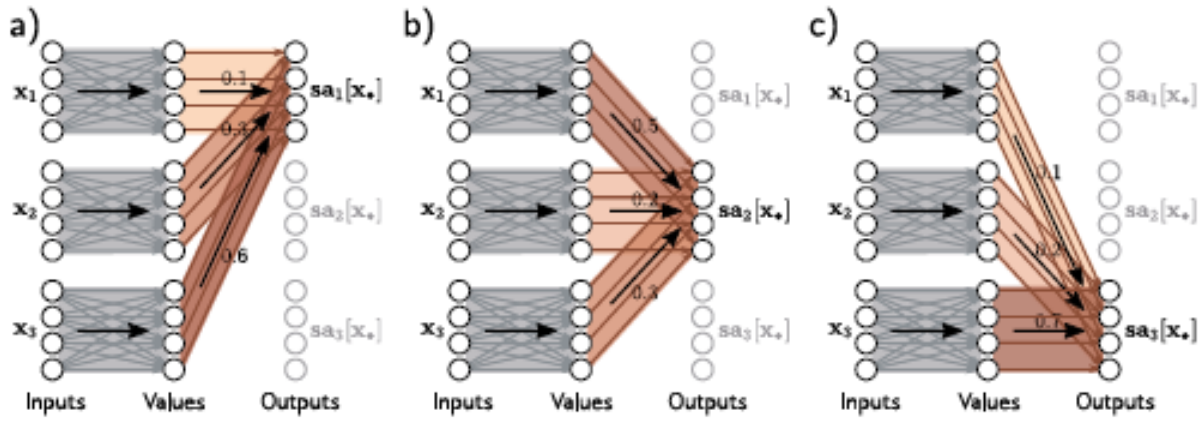


Figura 5 – O mecanismo de autoatenção recebe $N$ entradas $x_1, \ldots, x_N \in R^D$ (aqui $N = 3$ e $D = 4$) e processa cada uma separadamente para calcular $N$ vetores de valor. A $n$-ésima saída $\text{sa}_n[x_1, \ldots, x_N]$ (escrita como $\text{sa}_n[x_\bullet]$ para simplificar) é então calculada como uma soma ponderada dos $N$ vetores de valor, onde os pesos são positivos e somam um. **(a)** A saída $\text{sa}_1[x_\bullet]$ é calculada com $a[x_1, x_1] = 0.1$ vezes o primeiro vetor de valor, $a[x_2, x_1] = 0.3$ vezes o segundo vetor de valor e $a[x_3, x_1] = 0.6$ vezes o terceiro vetor de valor. **(b)** A saída $\text{sa}_2[x_\bullet]$ é calculada da mesma forma, mas desta vez com pesos de 0.5, 0.2 e 0.3. **(c)** A ponderação para a saída $\text{sa}_3[x_\bullet]$ é diferente novamente. Cada saída pode, portanto, ser vista como um roteamento diferente dos $N$ valores (PRINCE, 2023).

A Equação 2.17 mostra que os mesmos pesos $\Omega_v \in R^{D \times D}$ e vieses $\beta_v \in R^D$ são aplicados a cada entrada $x_\bullet \in R^D$. Esse cálculo é dimensionado linearmente com o comprimento da sequência $N$, o que requer menos parâmetros do que uma rede totalmente conectada que relaciona todas as entradas $DN$ a todas as saídas $DN$. O cálculo dos valores pode ser visto como uma operação matricial esparsa com parâmetros compartilhados.

Os pesos de atenção $a[x_m, x_n]$ combinam os valores de diferentes entradas. Eles também são esparsos, pois existe apenas um peso para cada par ordenado de entradas $(x_m, x_n)$, independentemente do tamanho dessas entradas.

Segue-se que o número de pesos de atenção tem uma dependência quadrática do comprimento da sequência $N$, mas é independente do comprimento $D$ de cada entrada.

Sabemos agora que os resultados surgem de duas transformações lineares encadeadas; os vetores de valor $\beta_v + \Omega_v x_m$ são calculados independentemente para cada entrada $x_m$, e esses vetores são combinados linearmente pelos pesos de atenção $a[x_m, x_n]$. No entanto, o cálculo geral da autoatenção não é linear e os próprios pesos de atenção são funções não lineares de entrada. Este é um exemplo de hiper-rede, onde uma ramificação da rede calcula os pesos de outra.

Para calcular a atenção, aplicamos mais duas transformações lineares às entradas:

$$q_n = \beta_q + \Omega_q x_n \tag{2.13}$$

$$k_m = \beta_k + \Omega_k x_m, \tag{2.14}$$

onde $\{q_n\}$ e $\{k_m\}$ são chamados de consultas e chaves, respectivamente. Então calculamos o produto escalar entre as consultas e chaves e passamos os resultados por meio de uma função softmax:

$$a[x_m, x_n] = \text{softmax}_m[k_m^T q_n] = \frac{\exp[k_m^T q_n]}{\sum_{m'=1}^{N} \exp[k_{m'}^T q_n]}, \tag{2.15}$$

onde, para cada $x_n$, os valores são positivos e somam um, isso é conhecido como autoatenção de produto escalar.

Os nomes “consultas” e “chaves” foram herdados do campo de recuperação de informação e têm a seguinte interpretação: a operação de produto escalar retorna uma medida de similaridade entre as entradas, de modo que os pesos $a[x_\bullet, x_n]$ dependem da similaridade relativa entre a enésima consulta e todas as chaves. A função softmax significa que as chaves “competem” entre si para contribuir para o resultado final. As consultas e chaves devem ter as mesmas dimensões. No entanto, estas podem diferir da dimensão dos valores, que geralmente são do mesmo tamanho que a entrada. No artigo *"Attention Is All You Need"* (VASWANI *et al.*, 2017) apresenta-se detalhadamente os mecanismos de atenção.

### 2.3.3 Formato Matricial

O cálculo acima pode ser escrito de forma compacta se as $N$ entradas $x_n$ formarem as colunas da matriz $D \times N$ denotada por $X$. Os valores, consultas e chaves podem ser calculados como:

$$V[X] = \beta_v \mathbf{1}^T + \Omega_v X \tag{2.16}$$

$$Q[X] = \beta_q \mathbf{1}^T + \Omega_q X \tag{2.17}$$

$$K[X] = \beta_k \mathbf{1}^T + \Omega_k X, \tag{2.18}$$

onde $\mathbf{1}$ é um vetor $N \times 1$ contendo unidades. O cálculo da autoatenção é então:

$$\text{Sa}[X] = V[X] \cdot \text{Softmax}\left[K[X]^T Q[X]\right], \tag{2.19}$$

onde a função Softmax[·] aplica a operação softmax independentemente a cada uma das colunas da matriz.

### 2.3.4 Camadas do Transformador

A autoatenção é apenas uma parte de uma camada transformadora completa. Consiste em uma unidade de autoatenção com várias cabeças (que permite que as representações de palavras interajam entre si) seguida por uma rede totalmente conectada MLP[$x$], que opera separadamente em cada palavra. Ambas as unidades são redes residuais, ou seja, sua saída é somada à entrada original. Além disso, é comum adicionar uma operação `LayerNorm` após as redes de autoatenção e totalmente conectadas. Isso é semelhante ao `BatchNorm`, mas utiliza estatísticas entre os tokens em uma única sequência de entrada para realizar a normalização. A camada completa pode ser descrita pela seguinte série de operações:

$$X \leftarrow X + \text{MhSa}[X]$$

### 2.3.5 Tokenização

Um pipeline de processamento de texto começa com um *tokenizer*, que divide o texto em unidades constituintes menores (tokens) a partir de um vocabulário de possíveis tokens. Embora esses tokens sejam frequentemente palavras, diversas dificuldades surgem:

- Algumas palavras (por exemplo, nomes) podem não estar no vocabulário.
- A pontuação é importante e deve ser tratada. Por exemplo, se uma frase terminar em ponto de interrogação, é desejável codificar essa informação.
- O vocabulário precisa incluir diferentes formas de uma palavra com sufixos diferentes (por exemplo, *caminhar*, *caminhou*, *caminhando*), mas não há garantias de que essas variações estejam diretamente relacionadas.

Uma abordagem seria usar letras e sinais de pontuação como vocabulário, mas isso resultaria em tokens muito pequenos, exigindo que a rede reaprendesse as relações entre eles. Na prática, utiliza-se um compromisso entre letras e palavras completas.

### 2.3.6 Incorporações

Cada token no vocabulário $V$ é mapeado para uma incorporação de palavra única, e as incorporações para todo o vocabulário são armazenadas em uma matriz $\Omega_e \in R^{D\times|V|}$. Os $N$ tokens de entrada são codificados na matriz $T \in R^{|V|\times N}$, onde a enésima coluna corresponde ao enésimo token como um vetor *one-hot*. Os embeddings de entrada são então calculados como $X = \Omega_e T$, onde $\Omega_e$ é aprendido como qualquer outro parâmetro da rede . Tipicamente, o tamanho da incorporação $D$ é 1024, e o tamanho do vocabulário total $|V|$ é de 30.000, resultando em muitos parâmetros em $\Omega_e$ a serem aprendidos antes da rede principal.

$$X \leftarrow \text{LayerNorm}[X]$$

$$x_n \leftarrow x_n + \text{MLP}[x_n] \quad \forall n \in \{1, \ldots, N\}$$

$$X \leftarrow \text{LayerNorm}[X]$$

onde os vetores de coluna $x_n$ são retirados separadamente da matriz de dados completa $X$. Em uma rede real, os dados passam por uma série dessas camadas de transformador.

### 2.3.7 Exemplo de Modelo de Codificador: BERT

O BERT (DEVLIN *et al.*, 2018) é um modelo de codificação com um vocabulário de 30.000 tokens e 340 milhões de parâmetros. Seus tokens de entrada são convertidos em *embeddings* de 1.024 dimensões e processados por 24 camadas de transformador, cada uma com autoatenção de 16 cabeças. Esse modelo ajusta seus parâmetros por meio de auto-supervisão em grandes corpora de texto e, posteriormente, é refinado para tarefas específicas com dados supervisionados.

#### 2.3.7.1 Pré-Treinamento

No pré-treinamento, o BERT é treinado para prever palavras ausentes em sequências de 256 tokens, por um milhão de etapas, totalizando aproximadamente 50 épocas de um corpus de 3,3 bilhões de palavras. Esse processo permite que o modelo aprenda estatísticas da linguagem e compreenda dependências contextuais.

#### 2.3.7.2 Ajuste Fino

Durante o ajuste fino, uma camada adicional é incluída ao transformador para especializar a rede em tarefas específicas, como:

- **Classificação de Texto**: O token `[CLS]` é usado para classificar a sequência, como em análise de sentimentos.
- **Classificação de Palavras**: Cada palavra é rotulada como uma entidade (pessoa, lugar etc.) usando uma função *softmax*.
- **Previsão de Extensão de Texto**: Para tarefas de resposta a perguntas, como SQuAD, o BERT prevê o início e o fim do trecho que contém a resposta.

### 2.3.8 Comparação entre Modelos de Codificador: BERT e RoBERTa

O *RoBERTa* (LIU *et al.*, 2019) conforme explorado por (IVANOV, 2023) é uma versão otimizada do BERT com várias melhorias, incluindo:

- **Remoção da Tarefa de Próxima Sentença**: Eliminada para melhorar o desempenho geral.
- **Corpus Maior e Mais Épocas**: Treinado com 160 GB de dados, o *RoBERTa* explora representações mais profundas da linguagem.
- **Mascaramento Dinâmico**: As palavras mascaradas mudam a cada iteração, aumentando a diversidade de predições.

Essas modificações resultam em um modelo mais robusto, com desempenho superior em benchmarks de compreensão de linguagem, mas a um custo computacional maior.

## 2.4 Acurácia como Métrica para Modelos Transformers

A acurácia é uma métrica muito utilizada na avaliação de modelos de *Processamento de Linguagem Natural* (PLN), especialmente em tarefas de classificação, como análise de sentimentos etc. No artigo *"Scaling Laws for Neural Language Models"* (KAPLAN *et al.*, 2020) explora-se como a acurácia de modelos Transformers é influenciada por fatores como o tamanho do modelo, a quantidade de dados e a capacidade computacional, além disso demonstra que aumentar consistentemente o tamanho do modelo e os dados de treinamento leva a melhorias na acurácia e introduz o conceito de leis de escala para prever o desempenho de modelos Transformers baseados em variáveis como parâmetros, dados e FLOPs.

A acurácia é definida como a proporção de previsões corretas em relação ao total de previsões feitas. Formalmente, podemos representá-la como:

$$\text{Acurácia} = \frac{\text{Número de previsões corretas}}{\text{Total de previsões}} = \frac{TP + TN}{TP + TN + FP + FN}, \tag{2.20}$$

onde $TP$, $TN$, $FP$ e $FN$ representam, respectivamente, os verdadeiros positivos, verdadeiros negativos, falsos positivos e falsos negativos. A acurácia é intuitiva e fácil de interpretar, tornando-se uma métrica inicial útil para avaliar a performance geral do modelo. Em tarefas de PLN onde as classes estão balanceadas (isto é, possuem proporções semelhantes no conjunto de dados), a acurácia fornece uma boa medida de quão bem o modelo está performando na tarefa.

Entretanto, em tarefas de PLN, muitas vezes os conjuntos de dados apresentam desequilíbrios entre as classes (por exemplo, em tarefas de classificação de sentimentos, pode haver muito mais exemplos de sentimentos neutros em comparação a positivos ou negativos). Em casos como esse, a acurácia pode ser enganosa, pois um modelo que tende a prever apenas a classe majoritária ainda pode obter uma alta acurácia, apesar de não capturar a diversidade das classes.

Para mitigar este problema, outras métricas, como precisão, revocação e a pontuação $F_1$, são frequentemente utilizadas em conjunto com a acurácia. Estas métricas são especialmente úteis em tarefas onde é importante distinguir corretamente entre classes menores.

# 3 RESULTADOS

## 3.1 Banco de Dados para o Estudo

A base de dados IMDB, amplamente utilizada em tarefas de classificação de sentimentos, está disponível no pacote `datasets`. Ela é composta por três divisões principais:

- **train**: utilizado para treinamento de modelos (25.000 exemplos).
- **test**: usado para avaliação de modelos (25.000 exemplos).
- **unsupervised**: dados não rotulados (50.000 exemplos).

As colunas relevantes para os conjuntos **train** e **test** são:

- **text**: contém a análise (positiva ou negativa) de um filme.
- **label**: classe associada ao sentimento (`0`: negativo, `1`: positivo).

| Index | Text | Label |
|---|---|---|
| 0 | I rented I AM CURIOUS-YELLOW from my video store... | 0 |
| 1 | "I Am Curious: Yellow"is a risible and pretentious movie... | 0 |
| 2 | If only to avoid making this type of film in the future... | 0 |
| 3 | This film was probably inspired by Godard's Masculin Féminin... | 0 |
| 4 | Oh, brother...after hearing about this ridiculous film... | 0 |

Tabela 1 – Amostras do dataset IMDB

## 3.2 Configuração do Experimento

Nos experimentos realizados, utilizamos o comprimento máximo de sequência de 512 tokens, compatível com as especificações do BERT e do RoBERTa. Cada entrada é representada por um tensor de dimensão $[B, 512, 768]$, onde:

- $B$: tamanho do lote (é a quantidade de exemplos processados simultaneamente);
- 512: comprimento máximo da sequência;
- 768: dimensão do embedding por token.

Os modelos geram **embeddings**, que são representações vetoriais de **tokens**, as menores unidades de texto utilizadas no processamento. Essas representações mantêm a dimensão $[B, 512, 768]$ e são posteriormente transformadas para uma saída final de dimensão $[B, 2]$, adequada para tarefas de classificação binária.

## 3.3 Análise dos Resultados

Os resultados obtidos para os modelos treinados com a base IMDB são apresentados na Tabela 2.

| **Modelo** | **Acurácia** | **Perda (Loss)** | **Tempo (s)** |
|---|---|---|---|
| BERT | 84% | 0.389 | 434.95 |
| RoBERTa | 85.5% | 0.460 | 461.25 |

Tabela 2 – Resultados dos experimentos com os modelos BERT e RoBERTa.

Embora o RoBERTa tenha alcançado uma acurácia ligeiramente superior, o BERT apresentou menor perda, indicando maior confiança em suas previsões. Em termos de tempo de execução, o BERT foi marginalmente mais rápido.

# 4 CONCLUSÃO

A análise comparativa entre os modelos **BERT** e **RoBERTa** evidenciou os avanços significativos proporcionados pelo uso de modelos pré-treinados em tarefas de *Processamento de Linguagem Natural* (PNL), como a classificação de sentimentos. Ambos os modelos, fundamentados em redes neurais profundas, utilizam arquiteturas baseadas em transformadores que permitem capturar relações semânticas complexas em textos. Essa capacidade se traduz em representações mais ricas e abstratas, facilitando o aprendizado de padrões intrínsecos à linguagem humana.

Os experimentos realizados neste trabalho, utilizando a base de dados IMDb, demonstraram a eficácia de ambos os modelos em alcançar altos níveis de acurácia, mesmo em cenários com conjuntos de dados de tamanho moderado. Além disso, a análise comparativa por meio de diferentes métricas ressaltou a importância de se considerar múltiplos aspectos de desempenho ao avaliar modelos de *deep learning* para aplicações práticas.

A principal contribuição deste estudo está em reforçar os benefícios do uso de modelos pré-treinados, que oferecem uma base robusta para tarefas específicas sem a necessidade de treinamento do zero. Essa abordagem não apenas reduz o tempo e os recursos computacionais necessários, mas também garante que os modelos possam ser aplicados com eficiência a uma ampla gama de contextos. Por fim, a escolha entre **BERT** e **RoBERTa** deve levar em conta as demandas específicas de cada aplicação, como a necessidade de maior precisão ou menor custo computacional, destacando o papel fundamental da experimentação para orientar essa decisão.

# REFERÊNCIAS


BROWN, T. B. *et al.* Language models are few-shot learners. **Advances in neural information processing systems**, v. 33, p. 1877–1901, 2020.

CYBENKO, G. Approximation by superpositions of a sigmoidal function. **Mathematics of Control, Signals and Systems**, v. 2, n. 4, p. 303–314, 1989.

DEVLIN, J. *et al.* Bert: Pre-training of deep bidirectional transformers for language understanding. **arXiv preprint arXiv:1810.04805**, 2018.

GERS, F. A.; SCHMIDHUBER, J.; CUMMINS, F. Learning to forget: Continual prediction with lstm. **Neural Computation**, v. 12, p. 2451–2471, 1999.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. MIT Press, 2016. Chapter 6: Backpropagation and Other Differentiation Algorithms. Disponível em: https://www.deeplearningbook.org.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural computation**, MIT Press, v. 9, n. 8, p. 1735–1780, 1997.

HORNIK, K. Approximation capabilities of multilayer feedforward networks. **Neural Networks**, v. 4, n. 2, p. 251–257, 1991.

IVANOV, I. **A Review of Pre-trained Language Models: From BERT, RoBERTa, to ELECTRA, DeBERTa, BigBird, and More**. 2023. Tung M Phung's Blog. Disponível em: https://tungmphung.com/a-review-of-pre-trained-language-models-from-bert-roberta-to-electra-deberta-bigbird-and-more/.

KAPLAN, J. *et al.* Scaling laws for neural language models. **arXiv preprint arXiv:2001.08361**, 2020. Disponível em: https://arxiv.org/abs/2001.08361.

LIU, Y. *et al.* Roberta: A robustly optimized BERT pretraining approach. **arXiv preprint arXiv:1907.11692**, 2019.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **Bulletin of Mathematical Biophysics**, Springer, v. 5, n. 4, p. 115–133, 1943.

MIKOLOV, T. *et al.* Recurrent neural network based language model. **Interspeech**, v. 2, p. 1045–1048, 2010.

PRINCE, S. J. **Understanding Deep Learning**. The MIT Press, 2023. Disponível em: http://udlbook.com.

RADFORD, A. *et al.* Improving language understanding by generative pre-training. **OpenAI**, v. 12, p. 1–12, 2018.

RAFFEL, C. *et al.* Exploring the limits of transfer learning with a unified text-to-text transformer. **Journal of Machine Learning Research**, v. 21, n. 140, p. 1–67, 2020.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. **Nature**, Springer Nature, v. 323, n. 6088, p. 533–536, 1986.

SUNDERMEYER, M.; SCHLÜTER, R.; NEY, H. Lstm neural networks for language modeling. **Interspeech**, v. 13, p. 601–604, 2012.

VASWANI, A. *et al.* Attention is all you need. **Advances in neural information processing systems**, v. 30, p. 5998–6008, 2017.