UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
ESCOLA DE INFORMÁTICA APLICADA

# Uma Avaliação Experimental de Arquiteturas de Aprendizagem Profunda na Classificação de Relevância de Microtextos quanto ao Domínio de Trânsito

MANUELA FERNANDES BLANCO RODRIGUEZ

**Orientador**
PEDRO NUNO DE SOUZA MOURA

RIO DE JANEIRO, RJ – BRASIL
JULHO DE 2024

Catalogação informatizada pelo autor

FR696 Fernandes Blanco Rodriguez, Manuela
Uma Avaliação Experimental de Arquiteturas de Aprendizagem Profunda na Classificação de Relevância de Microtextos quanto ao Domínio de Trânsito / Manuela Fernandes Blanco Rodriguez. -- Rio de Janeiro, 2024.
100

Orientador: Pedro Nuno de Souza Moura.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal do Estado do Rio de Janeiro, Graduação em Sistemas de Informação, 2024.

1. Processamento de Linguagem Natural. 2. Aprendizagem Profunda. 3. Transformers. I. Nuno de Souza Moura, Pedro, orient. II. Título.

Uma Avaliação Experimental de Arquiteturas de Aprendizagem Profunda na Classificação de Relevância de Microtextos quanto ao Domínio de Trânsito

MANUELA FERNANDES BLANCO RODRIGUEZ

Projeto de Graduação apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) para obtenção do título de Bacharel em Sistemas de Informação.

Aprovado por:

_____

Prof. Pedro Nuno de Souza Moura, D.Sc. (UNIRIO)

_____

Profa. Geiza Maria Hamazaki da Silva, D.Sc. (UNIRIO)

_____

André Luiz Farias Novaes, M.Sc. (PUC-Rio)

RIO DE JANEIRO, RJ – BRASIL.

JULHO DE 2024

# Agradecimentos

Agradeço aos meus pais, Fabíola e Roberto, por terem me apoiado durante todo o meu percurso acadêmico e por, acima de tudo, sempre me incentivarem a perseguir os meus sonhos.

Agradeço à minha família, por me encorajarem a estudar desde pequena e por me ensinarem que o estudo e o conhecimento são coisas que seguirão conosco para sempre.

Agradeço ao meu namorado, João Pedro, por todo o amor e companheirismo durante a escrita deste trabalho.

Agradeço aos meus amigos da faculdade, com quem pude dividir as alegrias e os obstáculos da graduação e espero levar para o resto da vida. Em especial, gostaria de agradecer às minhas amigas que, em um curso predominantemente masculino, sempre se uniram para criar uma comunidade acolhedora para todos.

Agradeço ao meu orientador, Prof. Pedro Nuno de Souza Moura, que me apresentou o universo de Aprendizagem Profunda e me deu todo o suporte na realização deste trabalho.

Agradeço aos professores e funcionários do DIA, que compartilham todo o seu conhecimento conosco e nos ajudam a atravessar esse caminho por vezes tortuoso, mas recompensador que é a graduação.

## RESUMO

O acesso a informações atualizadas de trânsito é essencial para a mobilidade urbana no mundo contemporâneo, pois permite a escolha de trajetos mais eficientes pelos habitantes de uma determinada localidade. Com a promoção do acesso ubíquo à internet nos últimos anos, usuários de redes sociais passaram a compartilhar, de modo não estruturado, diversos eventos cotidianos como condições de clima e de tráfego. Desse modo, este trabalho se propõe a desenvolver e treinar modelos de Aprendizagem Profunda capazes de classificar microtextos em português brasileiro provenientes das redes sociais quanto à sua relevância e pertencimento ao domínio de trânsito. Assim, foram realizados experimentos com diferentes arquiteturas tradicionais de Aprendizagem Profunda e um modelo da arquitetura *Transformer*, de modo que os resultados obtidos se mostraram bastante promissores, com acurácia, precisão e *recall* acima de 98%. Além disso, é proposto um *dataset* composto por 64.237 microtextos etiquetados manualmente entre duas categorias: “evento relevante de trânsito”, com 29.632 registros, e “evento não relevante”, com 34.605 registros.



**Palavras-chave**: Processamento de Linguagem Natural, Aprendizagem Profunda, *Transformers*, Redes Sociais.

## ABSTRACT

In the contemporary world, having access to updated information about traffic situation is essential to urban mobility, as it allows residents of a determined location to choose more efficient routes. With the promotion of ubiquitous internet access in recent years, social network users have started to share, in a non-structured way, different events of their daily lives, such as climate and traffic conditions. Therefore, this work aims to develop and train deep learning models able to classify microtexts in Brazilian Portuguese extracted from social networks in terms of their relevance and belonging to the context of traffic events. To this end, experiments with traditional deep learning architectures and one Transformer model were carried out, showing promising results with accuracy, precision, and recall over 98%. Furthermore, it is proposed a dataset with 64,237 microtexts manually labeled into two categories: “relevant traffic event” with 29,634 records and “non-relevant event” with 34,605 records.



**Keywords**: Natural Language Processing, Deep Learning, Transformers, Social Networks.

## Índice

## Índice de Tabela

## Índice de Figuras

# 1 Introdução

## 1.1 Motivação

No contexto das grandes metrópoles, a promoção da mobilidade urbana é um dos maiores desafios da atualidade (Estadão, 2022). Esse conceito pode ser entendido como a maneira pela qual os habitantes de uma cidade transitam pelos espaços urbanos e as condições que viabilizam, ou não, essa circulação. O crescimento rápido da população, acompanhado pelo aumento do número de veículos, nem sempre é seguido por políticas públicas voltadas ao planejamento urbano, acarretando problemas de deslocamento devido à sobrecarga do trânsito.

Na conjuntura brasileira, é possível citar o caso da cidade de São Paulo como o exemplo de uma capital que teve um crescimento rápido desacompanhado de planejamento público de mobilidade. A cidade, em que são contabilizados oitocentos novos carros em circulação todos os dias, bateu, em março de 2023, seu recorde anual de trânsito com 1.206 quilômetros de congestionamento. O recorde anterior, em fevereiro do mesmo ano, chegou a 1.160 quilômetros de lentidão (G1, 2023).

Além da falta de planejamento, eventos pontuais como acidentes, obras, operações de fiscalização ou condições climáticas também podem contribuir para congestionamentos e lentidão. Assim, a frequência diária de engarrafamentos nas grandes cidades impõe um tempo de locomoção maior do que aquele contabilizado se considerada apenas a distância a ser percorrida. No Brasil, 36% da população passa mais de uma hora por dia se deslocando para atividades de rotina, como trabalho e estudo (Folha de S. Paulo, 2023).

Desse modo, o acesso a informações atualizadas sobre condições de tráfego se torna essencial para a escolha de caminhos mais rápidos, possibilitando uma locomoção mais eficiente para motoristas e passageiros.

Além disso, a disponibilização desses dados se alinha com a definição de “cidades inteligentes”, no que se refere ao Brasil, promovida pela Carta Brasileira para

Cidades Inteligentes (“Carta Brasileira para Cidades Inteligentes”, 2023). Esse documento é uma iniciativa do Ministério da Integração e do Desenvolvimento Nacional (MDR) e estabelece uma agenda pública visando à integração do desenvolvimento urbano sustentável com a transformação digital.

> “Cidades inteligentes” são cidades comprometidas com o desenvolvimento urbano e a transformação digital sustentáveis, em seus aspectos econômico, ambiental e sociocultural, que atuam de forma planejada, inovadora, inclusiva e em rede, promovem o letramento digital, a governança e a gestão colaborativas e utilizam tecnologias para solucionar problemas concretos, criar oportunidades, oferecer serviços com eficiência, reduzir desigualdades, aumentar a resiliência e melhorar a qualidade de vida de todas as pessoas, garantindo o uso seguro e responsável de dados e das tecnologias da informação e comunicação. (“Carta Brasileira para Cidades Inteligentes”, 2023, p. 28)

A Carta propõe como diretriz “Estimular o protagonismo comunitário” e com a popularização e uso intensivo das redes sociais, uma grande quantidade de dados não estruturados é produzida por cidadãos sobre diversos assuntos, incluindo eventos de trânsito. Por esse motivo, a coleta de informações reportadas por usuários através de suas publicações pode ser útil para o monitoramento de condições de tráfego em tempo real e se encaixa perfeitamente em tal proposta da Carta.

Em particular, a plataforma de *microblogging* Twitter foi bastante utilizada como fonte de dados por conceder acesso livre aos microtextos, chamados *tweets*, publicados na rede social[1]. Apesar de diversos trabalhos já terem sido realizados em tarefas de classificação a partir desse tipo de postagem, poucos exploraram microtextos em língua portuguesa no domínio de trânsito.

O uso de Aprendizagem Profunda (*Deep Learning*) para Processamento de Linguagem Natural (PLN), assim como em diversos outros domínios, ganhou bastante popularidade nos últimos anos (AGGARWAL, 2023). Em especial, sua habilidade em aprender características de um texto de forma automática é fundamental para a construção de modelos precisos de classificação. Em vista disso, técnicas de Aprendizagem Profunda podem ser exploradas com o objetivo de extrair conhecimento de dados provenientes de redes sociais. Essas técnicas podem ser aplicadas justamente na tarefa de classificação de textos quanto à pertinência ao domínio de trânsito, como

[1] Durante a escrita deste trabalho, a plataforma limitou o uso gratuito de sua API a operações de escrita, concedendo permissão de leitura apenas a planos de acesso pagos.

um passo inicial à detecção de eventos de trânsito e suas localizações, possibilitando a construção posterior de uma ferramenta capaz de fornecer informações sobre o tráfego urbano em tempo real a partir de observações provenientes de diferentes redes sociais.

## 1.2 Objetivos

O objetivo geral deste trabalho é realizar a tarefa de identificação de microtextos pertencentes ao domínio de trânsito na rede social Twitter, utilizando técnicas de Aprendizagem Profunda.

De maneira mais específica, este estudo busca desenvolver e treinar modelos de redes neurais orientadas a PLN na abordagem ao problema de classificação de relevância de microtextos quanto ao domínio de trânsito. Além disso, deseja-se comparar o desempenho de tais modelos, em relação a métricas de avaliação adotadas, a partir de um conjunto de textos coletados e manualmente etiquetados.

## 1.3 Organização do texto

O presente trabalho está estruturado em capítulos e, além desta introdução, será desenvolvido da seguinte forma:

- Capítulo II: Conceitos Preliminares – Explica conceitos necessários ao entendimento deste trabalho, tais como Aprendizagem Profunda e Processamento de Linguagem Natural.
- Capítulo III: Revisão Bibliográfica – Descreve os trabalhos relacionados levantados na literatura.
- Capítulo IV: Arquiteturas Adotadas – Apresenta as arquiteturas adotadas para os experimentos computacionais.
- Capítulo V: Experimentos Computacionais – Apresenta o ambiente computacional utilizado, a metodologia adotada e os resultados dos experimentos.
- Capítulo VI: Conclusões – Reúne as considerações finais, assinala as contribuições da pesquisa e sugere possibilidades de aprofundamento posterior.

# 2 Conceitos Preliminares

Este capítulo irá elucidar os conceitos necessários para o entendimento deste trabalho. Dessa forma, serão abordados os conceitos envolvendo Aprendizagem Profunda e Processamento de Linguagem Natural. Aqueles que já se sentirem confortáveis com tais conceitos podem seguir para o Capítulo 3 sem prejuízo.

## 2.1 Aprendizagem Profunda

Aprendizagem Profunda (*Deep Learning*) é uma subárea de Aprendizagem de Máquina (*Machine Learning*) que utiliza redes neurais artificiais para o processamento de informações de forma a simular o funcionamento do cérebro humano. A arquitetura de uma rede neural é composta por um grande número de camadas que se relacionam hierarquicamente, formadas por unidades de processamento chamadas de neurônios artificiais, capazes de enviar e receber informações entre si. Tradicionalmente, os dados de saída de uma camada correspondem aos dados de entrada da seguinte, permitindo que o processamento de informação ocorra de forma progressiva, apesar de existirem outras formas de organização das camadas.

O funcionamento básico de um neurônio artificial é constituído pela soma ponderada de suas entradas e a aplicação de uma função de ativação, conforme ilustrado pela Figura 1. A ponderação das entradas permite que esses valores sejam amplificados ou minimizados por pesos que são repetidamente ajustados durante o processo de aprendizagem e indicam a contribuição de cada entrada para o neurônio. Por fim, de forma a capturar relações não lineares entre os dados, é aplicada uma função de ativação não linear na entrada ponderada, atingindo o valor de saída (ativação). Nesse sentido, é fundamental que sejam adotadas funções de ativação não lineares, pois, caso contrário, a rede neural tem o mesmo poder computacional de uma regressão linear (AGGARWAL, 2023).

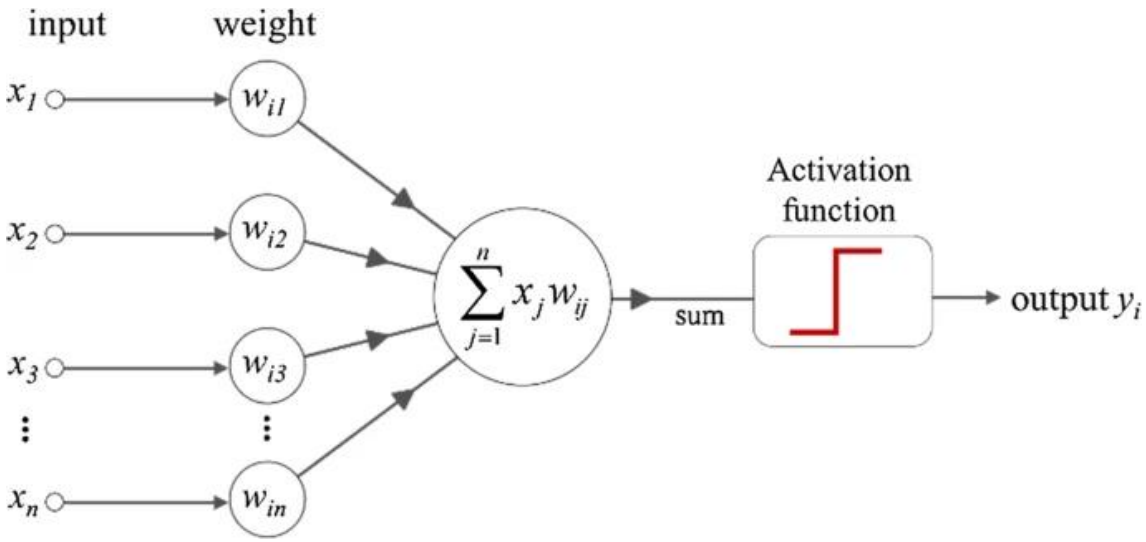


Figura 1: Diagrama esquemático de um neurônio artificial (extraído de RIBEIRO JUNIOR; DE ALMEIDA; GOMES, 2020).

Uma rede neural *feedforward* se decompõe em: camada de entrada, uma ou mais camadas ocultas e camada de saída. A camada de entrada de uma rede neural corresponde aos dados de entrada que são, por sua vez, repassados às camadas ocultas. Ao longo do treinamento da rede, as camadas ocultas são responsáveis por aprender representações incrementalmente complexas (mais abstratas) dos dados de entrada, de modo que, quanto mais camadas uma rede tiver, em geral, maior será a complexidade capturada (AGGARWAL, 2023). A última camada da rede é chamada de camada de saída e produz o resultado final a partir dos cálculos transmitidos pelas sucessivas camadas ocultas. Apesar de uma rede neural poder apresentar nenhuma ou somente uma camada oculta, em geral são consideradas redes neurais profundas aquelas que possuem pelo menos três camadas ocultas (KROHN; BEYLEVELD; BLASSENS, 2020).

Os pesos de uma rede neural costumam ser inicializados de forma aleatória a partir de uma determinada distribuição de probabilidade. Normalmente, as distribuições utilizadas são *Glorot Initialization* (GLOROT; BENGIO, 2010) e *He Initialization* (HE *et al*. 2015). Além disso, ao longo da etapa de treinamento, esses pesos podem divergir da distribuição definida, exigindo-se que sejam normalizados a fim de possibilitar a convergência da rede neural, por meio da técnica de *Batch Normalization* (ver AGGARWAL, 2023).

Por seu turno, o treinamento é realizado de forma iterativa e é composto pelas etapas de propagação para a frente (*feedforwarding*) e retropropagação (*backpropagation*). Na etapa de propagação para a frente, os dados de entrada são

inseridos na rede, processados e a respectiva predição é produzida como saída. As predições realizadas são então avaliadas por uma função de custo (*cost function*) que quantifica o quanto essas predições divergem do resultado final. Com base nessa quantificação, o algoritmo de Gradiente Descendente Estocástico (*Stochastic Gradient Descent* - SGD), associado ao algoritmo de retropropagação, determina um fator de ajuste aos parâmetros da rede, orientado pela minimização da função de custo. Esses passos são repetidos até que um determinado critério de parada seja atingido.

A pesquisa na área de Aprendizagem Profunda vem se popularizando na última década após o modelo AlexNet, proposto por Krizhevsky, Sutskever e Hinton (2012), vencer o desafio ILSVRC (*ImageNet Large Scale Visual Recognition Challenge*) ultrapassando os demais concorrentes da competição, ao atingir uma taxa de erro de apenas 15,3%. A arquitetura vencedora foi a única, dentre todos os competidores, a utilizar Aprendizagem Profunda em vez da abordagem tradicional de Aprendizagem de Máquina. O objetivo da competição foi avaliar o desempenho dos algoritmos participantes em tarefas de visão computacional utilizando um subconjunto da base de dados ImageNet (RUSSAKOVSKY *et al*., 2015), que contém 1,4 milhões de imagens divididas entre mil categorias.

Esse feito provocou, nos anos seguintes, uma verdadeira explosão de pesquisa e aplicações envolvendo técnicas de Aprendizagem Profunda, pela academia e pela indústria, voltadas para diversas áreas do conhecimento. A evolução das redes neurais para arquiteturas cada vez mais complexas foi possibilitada por uma grande disponibilidade de dados, avanço do poder computacional e o desenvolvimento de novas técnicas capazes de aperfeiçoar o processo de aprendizagem. Dentre essas aplicações, é possível destacar o uso de modelos para análise de imagens médicas, navegação de carros autônomos e recomendação de conteúdo em redes sociais e plataformas de *streaming*. Mais recentemente, os grandes modelos de linguagem (*Large Language Models* - LLM) despertaram bastante interesse da sociedade em geral após o lançamento do ChatGPT, um *chatbot* construído em cima do modelo GPT (*Generative Pre-trained Transformer*) (Radford *et al*., 2018) desenvolvido pela empresa OpenAI.

## 2.2 Funções de Ativação

No contexto das redes neurais, funções de ativação foram originalmente adotadas como uma forma de simular, em um neurônio artificial, o disparo de um potencial de ação por

um neurônio biológico. Atualmente, essas são utilizadas para introduzir não linearidade à saída de um neurônio, possibilitando o aprendizado da rede (AGGARWAL, 2023). Ao longo do treinamento, os pesos e vieses (*weights and biases*) dos neurônios são atualizados de forma a aproximar o valor da saída predita pelo modelo ao valor da saída real (*ground truth*) associada a uma determinada entrada. Desse modo, se pequenos ajustes nos parâmetros de um neurônio não forem refletidos na sua saída ou, inversamente, causarem drásticas mudanças na sua saída, o modelo como um todo não é capaz de aprender (KROHN; BEYLEVELD; BLASSENS, 2020). Assim sendo, tem-se que as funções de ativação desempenham um papel fundamental no treinamento de um modelo de Aprendizagem Profunda.

### 2.2.1 Sigmoid

A função de ativação Sigmoid recebe uma entrada e a transforma em um valor positivo contido no intervalo [0, 1], comprimindo, assim, a entrada recebida a um valor nesse intervalo. Por conta dessa característica, é qualificada como uma função de esmagamento (*squashing function*) (AGGARWAL, 2023). É útil para problemas de classificação binária, em que se deseja que a saída da rede neural corresponda a uma distribuição de probabilidade. Seu comportamento permite que pequenas alterações nos parâmetros de um neurônio produzam pequenas alterações em sua saída, de maneira suave. No entanto, para entradas com valores muito grandes, positivos ou negativos, as saídas se concentram nos extremos do intervalo, fazendo com que a maior parte dos ajustes nos parâmetros de um neurônio não alterem a saída da função. Esse fenômeno é chamado de saturação e tende a impossibilitar o aprendizado da rede neural (KROHN; BEYLEVELD; BLASSENS, 2020). A Figura 2 ilustra a representação gráfica da função, incluindo sua equação no canto superior esquerdo da imagem.

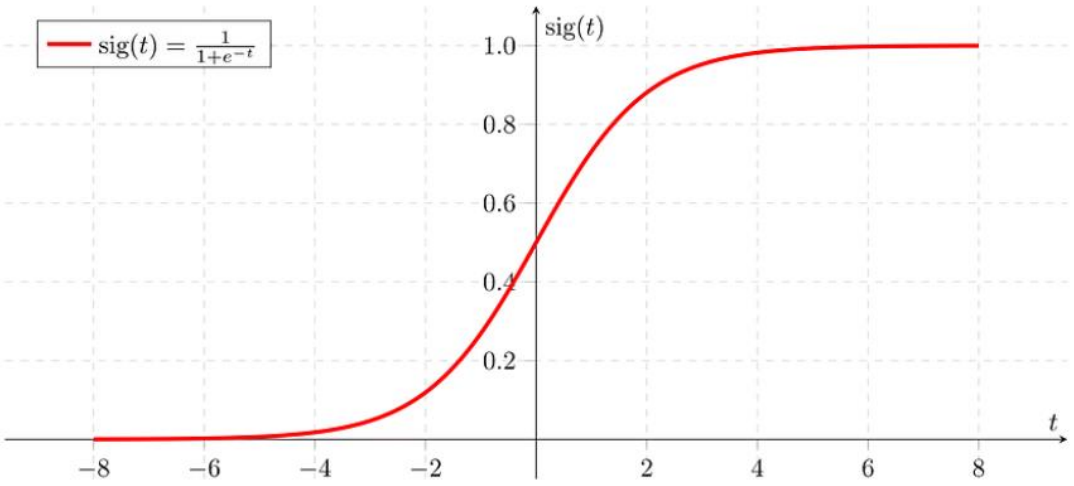


Figura 2: Gráfico da função de ativação Sigmoid (extraído de ARC, 2024).

### 2.2.2 Tanh

A função tangente hiperbólica (tanh) possui o mesmo formato de curva que a função Sigmoid, mas apresenta um intervalo de variação definido em [-1, 1], como representado na Figura 3. Sua fórmula, localizada no lado direito da imagem, descreve o comportamento da função, em que entradas negativas resultam em ativações negativas, entradas iguais a zero resultam em ativações iguais a zero e entradas positivas resultam em ativações positivas. Essa característica costuma produzir saídas centradas em zero, o que é considerado uma vantagem em relação ao uso da função Sigmoid por dificultar a saturação do neurônio e permitir uma melhor convergência da rede durante o treinamento (AGGARWAL, 2023).

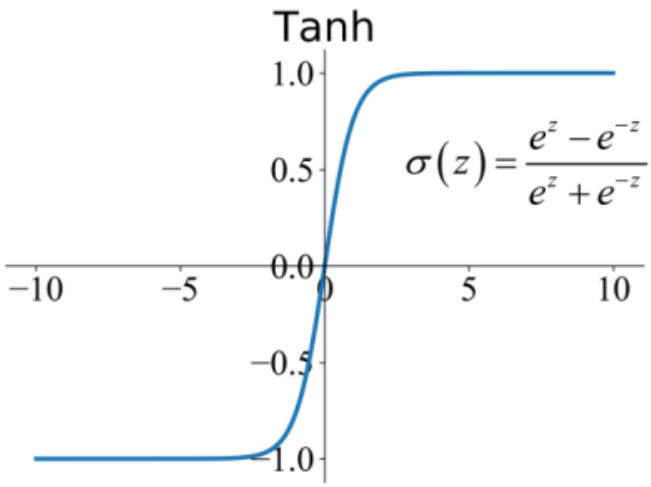


Figura 3: Curva da função tangente hiperbólica (extraído de FENG *et al*., 2019).

### 2.2.3 ReLU – *Rectified Linear Unit*

A função ReLU foi inspirada no comportamento de neurônios biológicos e apresenta um comportamento diferente da função Sigmoid. Apesar de ser fundamentalmente a combinação de duas funções lineares bem simples, isto é, a função constante e a função identidade, a ReLU é uma função não linear. Entradas positivas da função estão associadas a uma saída igual à própria entrada não modificada, já entradas não positivas estão associadas a uma saída igual a zero, como mostrado na Figura 3.

Sua principal vantagem se encontra em seu formato razoavelmente simples. Dois benefícios derivam desse fato: (i) o menor custo computacional para calculá-la, em oposição ao oneroso custo imposto pela Sigmoid; (ii) não leva à saturação do neurônio para valores de entrada positivos; e (iii) a sua derivada parcial, cujo cálculo é fundamental para o algoritmo SGD, é mais fácil de ser calculada. Além disso, o uso da função ReLU tende a levar a um menor tempo de convergência da rede neural no treinamento em relação à função Sigmoid (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

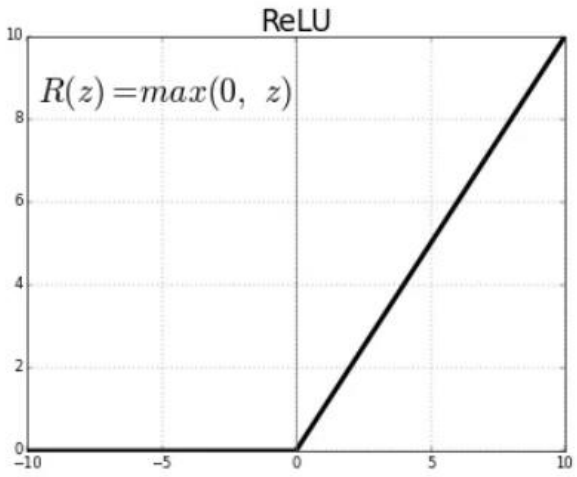


Figura 4: Gráfico da função de ativação ReLU (extraído de SHARMA, 2024).

### 2.2.4 *Softmax*

Caso a tarefa abordada corresponda a um problema de classificação multiclasse, isto é, envolvendo múltiplas classes, não é adequado empregar a função de ativação Sigmoid na camada de saída da rede neural. As redes construídas para processar problemas multiclasses demandam uma função de ativação cuja saída seja uma distribuição de probabilidade sobre as *n* classes contidas no domínio do problema, como é o caso da

função *Softmax*. Na verdade, é possível verificar que a função Sigmoid constitui um caso particular da função *Softmax* quando o número de classes *n* é igual a 2 (KROHN; BEYLEVELD; BLASSENS, 2020).

A *Softmax* é uma função de ativação que leva em consideração todos os neurônios de uma camada, recebendo como entrada um conjunto de valores Z = [z1, z2, ..., z*n*] e calcula uma proporção exponencial de cada valor em relação ao todo. Assim, a saída produzida será justamente uma distribuição de probabilidade considerando o resultado de cada classe como um número no intervalo [0, 1] e de forma que o somatório de todos os resultados seja igual a 1. A definição da fórmula da função *Softmax* segue abaixo, em que Z é o vetor de entrada contendo valores pertencentes ao conjunto dos números reais:

$$softmax(Z_i) = \frac{exp^{z_i}}{\sum_{j=1}^{k} exp^{z_j}}$$

## 2.3 Funções de Custo

As funções de custo (ou de perda) são funções utilizadas para quantificar o erro entre o resultado predito pela rede e o resultado verdadeiro esperado. Essa quantificação é importante para permitir o aprendizado pelo algoritmo SGD. Além disso, essa função permite validar a qualidade do modelo, pois verifica a sua capacidade de generalização, ou seja, sua capacidade de generalizar para dados que não foram vistos durante o treinamento.

Diz-se que, quando um modelo se ajusta bem aos dados de treinamento, mas não generaliza bem para dados não vistos, mas que pertencem à mesma distribuição original dos dados de treinamento, ele está sobreajustado (*overfitted*). Por outro lado, quando o modelo se ajusta muito pouco aos dados de treinamento, diz-se que ele subajustado (*underfitted*).

Existem diversas funções de custo propostas na literatura para Aprendizagem Profunda, mas de particular interesse para este trabalho é a função *Categorical Cross Entropy*, que será explicada na subseção a seguir. Essa é frequentemente utilizada para problemas de classificação multiclasse, ou seja, cujos exemplos pertençam a uma dentre três ou mais categorias.

### 2.3.1 *Categorical Cross Entropy*

A função *Categorical Cross Entropy* é uma função de custo criada na Teoria da Informação e baseada no conceito de entropia (LESKOVEC; RAJARAMAN; ULLMAN, [s.d.]) destinada à comparação de duas distribuições de probabilidade. Dessa forma, ela é utilizada para se comparar a distribuição de probabilidade gerada pela rede (por meio da função *Softmax*) com a distribuição da saída real (em formato *one-hot*, isto é, 0's e 1's), para uma determinada entrada da rede. Assim, quanto menor seu valor, mais próximas são as distribuições de probabilidade e, portanto, menor é o erro obtido pelo modelo. Sua fórmula é descrita a seguir:

$$CE = \sum_{i=1}^{N} y_i * \log(\hat{y}_i)$$

em que, para uma dada classe $i$, $\hat{y}_i$ corresponde à probabilidade predita pelo modelo para essa classe, enquanto $y_i$ corresponde à probabilidade real dessa classe.

É bastante comum utilizá-la em simultâneo com a função *Softmax* como camada de saída, visto que, pelo comportamento da *Softmax*, a saída da rede será uma distribuição de probabilidade a ser comparada com a distribuição de probabilidade da saída real. Além disso, uma vantagem importante é que a derivada da função de custo resultante dessa combinação assume o formato y' - y, o que é muito vantajoso por ser uma derivada simples e fácil de ser calculada (LESKOVEC; RAJARAMAN; ULLMAN, [s.d.]).

## 2.4 Métricas de Avaliação

Para garantir a qualidade de um modelo de redes neurais, é necessário escolher as métricas que serão empregadas para avaliar e testar seu desempenho. Algumas das métricas mais comumente adotadas são: acurácia, precisão, *recall* (ou revocação) e F1-Score. A partir do conceito de Matriz de Confusão, que corresponde a uma tabela que exibe os erros e acertos do modelo caracterizando-os entre VP (verdadeiro positivo), FP (falso positivo), FN (falso negativo) e VN (verdadeiro negativo), é possível obter as fórmulas que descrevem as métricas de avaliação. A Figura 5 exibe o esquema geral da Matriz de Confusão para um problema de classificação binária.

A acurácia corresponde à proporção de dados classificados corretamente pelo modelo em relação ao total de predições feitas. Apesar de ser uma métrica relevante, ela

pode não ser suficiente para avaliar modelos que tenham sido treinados em um conjunto de dados desbalanceados, em que a quantidade de exemplos de uma ou mais classes excede a quantidade de exemplos de outras classes. A acurácia é então definida pela seguinte fórmula:

$$acurácia = \frac{VP + VN}{VP + VN + FP + FN}$$

Por sua vez, a precisão mede a proporção de exemplos positivos que são corretamente classificados como positivos em relação ao total de exemplos classificados como positivos pelo modelo. Assim, a precisão é bastante útil quando se deseja analisar o modelo sob a perspectiva de falsos positivos obtidos. Ela é calculada através da seguinte fórmula:

$$precisão = \frac{VP}{VP + FP}$$

Já o *recall* é a proporção de acertos verdadeiros, ou seja, corresponde à proporção de exemplos positivos corretamente classificados como positivos sobre o total de exemplos realmente positivos. Dessa forma, o *recall* é uma métrica útil quando se deseja analisar o modelo sob a perspectiva de número de falsos negativos. É possível calculá-lo através da fórmula abaixo:

$$recall = \frac{VP}{VP + FN}$$

Nos casos em que se faz necessário reduzir o número de falsos positivos e o número de falsos negativos, é possível utilizar o F1-Score, que combina a precisão e o *recall* em uma única métrica. Seu valor é calculado a partir da média harmônica das duas métricas, possibilitando a penalização de valores extremamente baixos em qualquer uma delas. Dessa forma, o F1-Score só será alto se o valor de ambas as métricas também forem e, quanto maior seu valor, melhor tende a ser a qualidade do modelo avaliado. Tal métrica é determinada pela seguinte fórmula:

$$F1 - Score = \frac{2 * precisão * recall}{precisão + recall}$$

Assim como acontece com a acurácia, se utilizadas sozinhas, essas métricas também podem fornecer uma visão incompleta dos resultados obtidos, como, por exemplo, nos casos em que a distribuição de classes é desigual. Por isso, é importante utilizá-las em conjunto para obter uma avaliação abrangente do desempenho do modelo.

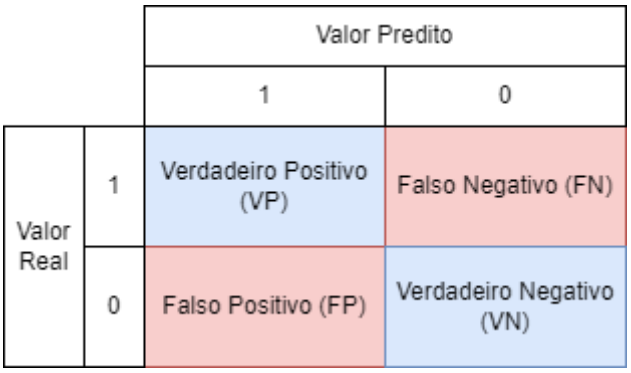


Figura 5: Esquema geral da Matriz de Confusão para um problema de classificação binária.

## 2.5 Redes Convolucionais

As Redes Neurais Convolucionais são redes constituídas por uma ou mais camadas convolucionais, como ilustrado na Figura 6. No entanto, elas também podem apresentar outros tipos de camadas, como camadas de *pooling* ou camadas densas, dado que apresentam uma arquitetura composta por uma parte voltada à extração de características (*features*) e uma parte voltada à classificação (KROHN; BEYLEVELD; BLASSENS, 2020). Esse tipo de arquitetura é comumente utilizado para a classificação de imagens, pois apresenta um processamento eficiente de padrões espaciais, mas também pode ser aplicada a dados unidimensionais em tarefas de PLN, por exemplo.

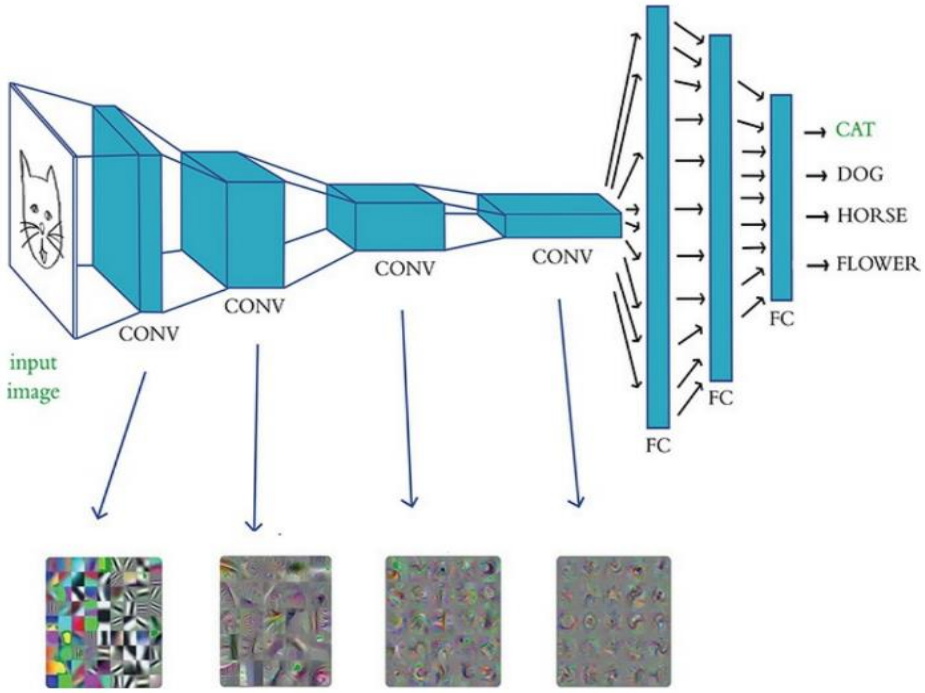


Figura 6: Representação gráfica de uma rede convolucional (extraído de KROHN; BEYLEVELD; BLASSENS, 2020).

As camadas convolucionais são responsáveis por efetuar a operação de convolução nos dados processados pela rede. Uma convolução corresponde à aplicação de uma matriz de pesos, chamada de filtro ou *kernel*, que será atualizada ao longo do treinamento e que percorre toda a matriz de entrada realizando um produto interno entre seus pesos e a região da imagem sobre a qual o filtro está aplicado. O resultado é um único escalar que passa por uma função de ativação cujo resultado será armazenado em uma matriz resultante. Desse modo, por conter o resultado da convolução da entrada com tal filtro, cada valor da matriz resultante representa características da entrada que se deseja destacar, possibilitando uma melhor análise do classificador. Uma camada convolucional costuma ter diversos filtros, criando assim um mapa de ativação composto pelas diversas matrizes resultantes das operações de convolução de tais filtros com a entrada.

As camadas de *pooling* se encontram em pares com as camadas convolucionais. Após a aplicação dos filtros convolucionais, a operação de *pooling* é realizada de forma a reduzir a dimensionalidade espacial do mapa de ativação. Essa técnica foi introduzida pelo modelo AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) e sua aplicação consiste na aplicação de um filtro, ou *kernel*, que percorre todo o mapa de ativação

efetuando uma operação predefinida, que pode ser o cálculo da média dos valores das ativações da região (*average pooling*), ou a seleção do valor correspondente à maior ativação (*max pooling*) da região analisada. Outras operações também podem ser empregadas, mas a *max pooling* costuma ser a mais frequente por sintetizar as ativações mais fortes das regiões percorridas pelo filtro. O objetivo da *max pooling* é adicionar invariância translacional (*translational invariance*) ao modelo, ou seja, mesmo se pequenas mudanças forem aplicadas à localização de uma característica na entrada, o mapa de ativação resultante da operação de *pooling* conterá a característica localizada no mesmo lugar.

Por fim, a parte da rede responsável pela classificação dos dados é composta por uma camada de achatamento (*flatten*) e uma camada densa, também chamada de "totalmente conectada" (*fully connected layer*), em que cada neurônio está conectado a todos os neurônios da camada anterior. A camada de *flatten* é utilizada para converter a matriz multidimensional dos mapas de ativação resultantes em uma matriz com uma única dimensão, isto é, um vetor, possibilitando que ele seja utilizado como entrada para a camada densa. Esta, por sua vez, é utilizada para agregar as características aprendidas pelas camadas convolucionais e realizar a combinação não linear de tais características, de forma a mapear diferentes combinações de características a categorias específicas dos dados. Em uma tarefa de classificação, o resultado da camada densa é tipicamente utilizado como entrada de uma função *Softmax*, descrita na Seção 2.2.4, que irá retornar uma distribuição de probabilidade considerando as diferentes categorias possíveis para a tarefa em questão.

## 2.6 Redes Recorrentes

As Redes Neurais Recorrentes (*Recurrent Neural Networks* - RNNs) correspondem a uma categoria de modelos de Aprendizagem Profunda capazes de lidar com longas sequências de dados de formato unidimensional (KROHN; BEYLEVELD; BLASSENS, 2020), tais como séries temporais ou textos em linguagem natural. As redes *Long Short-Term Memory* (LSTMs) e *Gated Recurrent Units* (GRUs) fazem parte dessa categoria e a principal característica desse tipo de arquitetura é a presença de *loops* que possibilitam que informações do início da sequência sejam passadas adiante para os módulos de recorrência seguintes. Isso é possível porque o resultado de um módulo $n$ é utilizado como uma entrada adicional para o módulo $n$+1. Dessa maneira, a

partir de uma sequência de dados, a rede é capaz de aprender relações entre dados que foram processados anteriormente na sequência e dados que estão sendo processados no contexto atual.

O conceito de LSTM foi introduzido por Hochreiter e Schmidhuber (1997) e trata-se de um tipo de rede neural recorrente adequada para casos em que a sequência de dados apresenta grandes distâncias entre as informações de interesse e o contexto em que elas são utilizadas na sequência. A arquitetura de uma LSTM é mais complexa do que uma RNN simples, como detalha a Figura 7. No caso de uma RNN simples, as células de recorrência passam adiante sua saída, também chamada de *hidden state*, e possuem apenas uma função de ativação. No caso das LSTMs, além do *hidden state*, que funciona como memória de curto prazo, é passado adiante o *cell state*, uma informação que persiste ao longo da sequência e funciona como uma memória de longo prazo. Ademais, as células de uma LSTM possuem "portões" (*gates*) na forma de funções de ativação, decidindo se a informação deve ser incorporada ou não ao *hidden state* e ao *cell state*. Os dois primeiros portões Sigmoid são responsáveis por decidir se a informação recebida é relevante e, se for, de que forma ela será incorporada (linear ou não linearmente) ao *cell state*. O último portão Sigmoid, por sua vez, é responsável por decidir se a informação recebida deve ser "esquecida" no contexto local ou se deve ser incorporada ao *hidden state* e consequentemente adicionada ao *output* da célula.

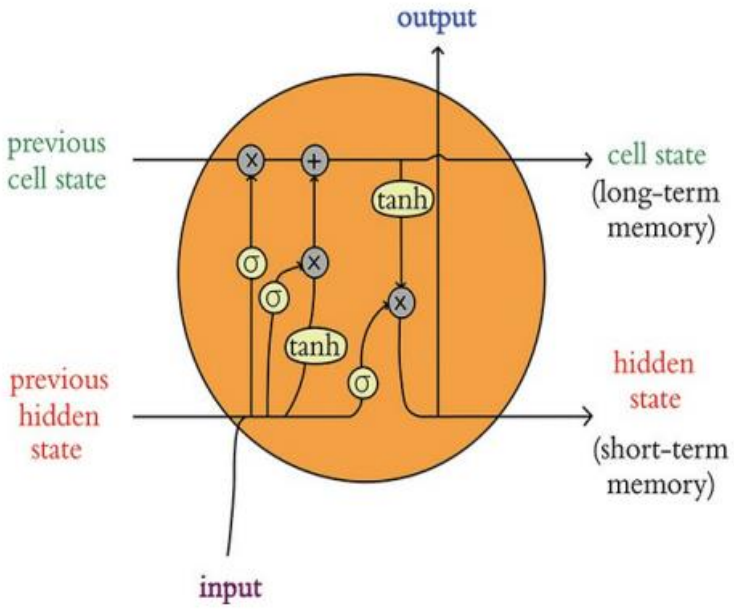


Figura 7: Diagrama esquemático de uma rede LSTM (extraído de KROHN; BEYLEVELD; BLASSENS, 2020).

## *2.7 Transformers*

As arquiteturas *Transformer* são modelos de Aprendizagem Profunda introduzidos em (VASWANI *et al*., 2017) e desenvolvidos, inicialmente, para o processamento de dados sequenciais. Diferentemente das Redes Recorrentes, os *Transformers* conseguem processar as sequências de entrada em paralelo, tornando os processos de treinamento e inferência do modelo mais eficientes. Essa arquitetura ganhou bastante popularidade no treinamento de *Large Language Models* (LLMs), modelos generativos treinados em grandes quantidades de texto capazes de desempenhar de maneira eficiente diversas tarefas de PLN.

As principais inovações dessa arquitetura correspondem ao mecanismo de *Self Attention*, explicado mais à frente na Seção 2.7.1, e à técnica de *Positional Encoding* ou *Positional Embedding*, que mapeia o índice de cada elemento da sequência a um vetor que representa sua posição relativa na entrada. O vetor resultante é então somado à representação vetorial do próprio elemento para que o modelo saiba a ordem específica da entrada e seja capaz de processá-la paralelamente.

A arquitetura de um modelo *Transformer* foi proposta originalmente contendo um *encoder* e um *decoder*, mas pode variar de acordo com a tarefa a ser realizada e conter somente um dos componentes. Os *encoders* costumam ser utilizados para tarefas de PLN como classificação e reconhecimento de entidades nomeadas (*Named Entity Recognition* - NER), pois produzem uma representação vetorial da entrada. Os *decoders*, no entanto, costumam ser empregados em tarefas generativas de PLN, pois produzem uma sequência de saída em que cada elemento é gerado a partir de sua probabilidade de ocorrência após o elemento gerado anteriormente. Ambos são compostos por diversas camadas contendo mecanismos de Autoatenção (*Self Attention*) e Redes Neurais Diretas (*Feedforward Neural Networks* - FNNs), esta última consistindo no tipo mais simples de rede neural em que os dados se movem apenas para frente, não havendo ciclos ou *loops*. A saída da camada de Autoatenção passa pela camada de FNN possibilitando que esta adicione não linearidade às representações dos elementos da entrada e o modelo seja capaz de aprender relacionamentos e padrões complexos contidos nos dados. Além disso, o fato de as camadas que constituem os *encoders* e *decoders* serem empilhadas permite que o modelo seja capaz de extrair características hierárquicas dos dados, ou seja, o nível de abstração das características aprendidas aumenta progressivamente ao longo das camadas do modelo.

### 2.7.1 Modelos de Atenção

Os modelos de atenção foram introduzidos no artigo *Attention is All You Need* (VASWANI *et al*., 2017) e foram desenvolvidos com o objetivo de aumentar a performance de modelos de Aprendizagem Profunda em tarefas de PLN, podendo ser integrados a diversos tipos de redes neurais, como redes convolucionais, redes recorrentes e *Transformers*.

Essa técnica é responsável por atribuir um nível de importância para cada elemento da entrada, possibilitando que a rede foque nas partes mais relevantes de um determinado texto de acordo com a tarefa a ser realizada. Assim, em uma operação de *Self Attention*, *n* vetores de *embeddings* são processados e *n* vetores contextualizados com o nível de importância de cada palavra contida na entrada são retornados. Nessa operação são utilizados três mecanismos de atenção derivados dos dados de entrada: os vetores *Query*, *Key* e *Value*, de forma que cada um deles possui uma matriz de pesos associada que será atualizada ao longo do treinamento.

Ao processar cada elemento da entrada para calcular seu valor de importância em relação a si mesmo e em relação aos outros, o vetor *Query* do elemento analisado é comparado, através do cálculo do produto interno, às diversas possibilidades de vetores *Keys*, retornando um conjunto de pontuações de atenção. Em seguida, as pontuações resultantes são normalizadas pela função *Softmax*, que resultará em uma distribuição de probabilidade mensurando a importância de cada elemento da entrada para o elemento analisado. Por fim, é calculada a soma ponderada dos vetores *Value* multiplicados pelas saídas correspondentes da função *Softmax* para cada elemento, produzindo a representação final de atenção do elemento analisado, como ilustrado na Figura 8.

No entanto, apenas uma camada de *Self Attention* não é suficiente para capturar a complexidade dos relacionamentos contidos em dados de entrada como um texto, por exemplo. Assim, a ideia de *Multi-head Attention* também é introduzida no mesmo artigo, possibilitando que uma rede aprenda diferentes padrões de atenção paralelamente. Cada conjunto de *Key*, *Query* e *Value* utilizado irá corresponder a uma *Attention Head* e será capaz de detectar diferentes padrões nos dados de entrada, dado que suas matrizes de pesos associadas são inicializadas aleatoriamente no início do treinamento e irão produzir diferentes vetores *Query*.

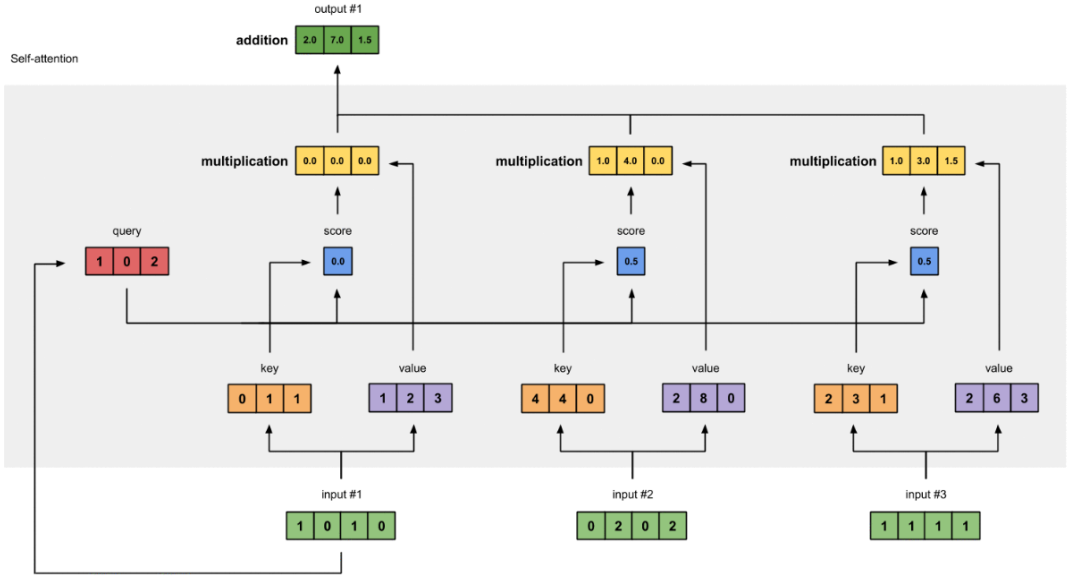


Figura 8: Representação gráfica do mecanismo de *Self Attention* (extraído de KARIM, 2023).

## 2.8 Processamento de Linguagem Natural

A área de Processamento de Linguagem Natural busca estudar como a linguagem produzida por nós, humanos, pode ser processada por máquinas de forma a realizar tarefas automáticas. Essas tarefas compreendem, por exemplo, a tradução simultânea de idiomas, o desenvolvimento de mecanismos de busca e também os mecanismos de reconhecimento de voz utilizados por assistentes virtuais. Algumas delas, por envolverem um maior nível de complexidade, são candidatas adequadas para o uso de Aprendizagem Profunda (KROHN; BEYLEVELD; BLASSENS, 2020).

No cenário de PLN, um modelo de linguagem é um modelo de Aprendizagem Profunda capaz de estimar a probabilidade da próxima palavra ou frase ao analisar um determinado texto em um documento. No entanto, para que um texto seja utilizado como entrada de uma rede neural, é necessário que sejam construídas representações numéricas das palavras que o compõem e os relacionamentos estabelecidos entre essas. Desse modo, ao longo do tempo, diversas técnicas de representação de texto foram propostas e se tornaram progressivamente mais complexas, com o objetivo de produzir representações vetoriais mais sofisticadas a partir de algoritmos cada vez mais eficientes na captura de sentido e contexto entre as palavras. Nas subseções a seguir, serão descritas algumas dessas abordagens.

### 2.8.1 *Bag of Words* e TF-IDF

As técnicas de *Bag of Words* e TF-IDF são as mais simples e clássicas, mas apresentam algumas limitações. Para obter a representação de cada palavra no escopo dos *corpora* (conjunto de textos), esses métodos levam em consideração sua ocorrência ou frequência. Assim, na técnica de *Bag of Words*, cada texto é representado por um determinado vetor, em que cada componente do vetor se refere a uma palavra e o valor representa a contagem de quantas vezes essa palavra aparece no texto.

A técnica de TF-IDF, por sua vez, busca medir o quão importante é uma palavra através da multiplicação de duas métricas: frequência do termo (TF - *term frequency*) e inverso da frequência nos documentos (IDF - *inverse document frequency*). A primeira métrica mede a frequência com a qual um termo aparece em um dado texto e a segunda mede o quão importante um termo é no conjunto de todos os textos que estão sendo analisados. Desse modo, palavras que se repetem com mais frequência em textos individuais são consideradas mais importantes do que aquelas que se repetem com frequência de maneira geral.

No entanto, esse tipo de representação se torna um problema nos casos em que o vocabulário do texto é muito grande, o que é bastante comum em tarefas de PLN, levando a uma representação em um espaço altamente dimensional e, pois, suscetível ao problema da "maldição da dimensionalidade" (LESKOVEC; RAJARAMAN; ULLMAN, 2020). Além disso, a representação construída não é capaz de captar o significado das palavras processadas e o contexto em que aparecem.

### 2.8.2 *Word Embeddings*

Os *Word Embeddings*, entretanto, são estruturas multidimensionais capazes de representar palavras utilizando sua localização no texto e também seu significado. Isso significa que palavras que costumam aparecer juntas são armazenadas em localizações parecidas no espaço multidimensional, preservando a relação semântica entre elas. O processo de vetorização é realizado através de métodos de Aprendizagem Profunda e diferentes implementações de *Word Embeddings* foram propostas ao longo dos anos, das quais o *Word2Vec* foi uma das primeiras.

A técnica de *Word2Vec* (MIKOLOV *et al*., 2013) utiliza redes neurais e apresenta duas abordagens diferentes de implementação: CBOW (*Continuous Bag-of-Words*) e *Skip-Gram*. Nos dois tipos de implementação, o processo de treinamento irá

gerar as representações vetoriais das palavras processadas, mas se a abordagem CBOW for utilizada, a rede neural é treinada para aprender uma palavra dado o seu contexto, ou seja, prever uma palavra alvo a partir das *t* palavras anteriores e *t* palavras posteriores contidas no texto analisado. No caso da abordagem *Skip-Gram*, a rede é treinada para a tarefa oposta e irá aprender um contexto (*t* palavras anteriores e *t* posteriores) a partir de uma dada palavra.

A técnica FastText (BOJANOWSKI *et al*., 2017) foi desenvolvida com um funcionamento bastante parecido ao vetorizador *Word2Vec*, utilizando as mesmas abordagens de implementação. No entanto, seu principal diferencial é o processamento de uma palavra como a soma de suas subpalavras, ou seja, através de subconjuntos de letras que compõem a palavra em vez da palavra como um todo. Consequentemente, o modelo é capaz de aprender representações úteis para palavras raras ou que não tenham sido vistas durante o treinamento da rede, o que pode ser uma vantagem em relação a outros métodos em tarefas cujos *corpora* contenham um vocabulário muito vasto.

Finalmente, a técnica GloVe (*Global Vectors for Word Representation*) (PENNINGTON; SOCHER; MANNING, 2014) é capaz de construir *Word Embeddings* que armazenam não somente a representação vetorial das palavras e suas relações semânticas, mas também suas relações sintáticas. No entanto, diferentemente dos vetorizadores *Word2Vec* e FastText, suas representações vetoriais são construídas a partir da fatoração de uma matriz de co-ocorrência, que avalia a probabilidade de duas palavras aparecerem em um mesmo texto.

# 3 **Revisão Bibliográfica**

Este capítulo apresenta os trabalhos relacionados ao tema escolhido, ou seja, que tenham aplicado modelos de Aprendizagem de Máquina na tarefa de classificação de microtextos extraídos de redes sociais.

O trabalho de Teteo *et al*. (2019) apresenta um *framework* para a extração, tratamento e identificação de eventos de trânsito e suas localidades a partir de *tweets* escritos em língua portuguesa. A técnica de *Conditional Random Fields* (CRF) foi aplicada para realizar a tarefa de Reconhecimento de Entidades Nomeadas, isto é, identificar nomes próprios que se referem a uma localização ou a um evento nos dados recuperados. CRFs são modelos estatísticos, comumente representados por grafos não-direcionados, capazes de relacionar, a partir de uma entrada sequencial, a probabilidade condicional das etiquetas de saída considerando o contexto compartilhado por etiquetas vizinhas. Assim, o modelo de CRF foi utilizado para mapear a relação das palavras contidas nos *tweets* extraídos e as etiquetas atribuídas a elas. As entidades reconhecidas são etiquetadas de acordo com sua categoria, de forma que palavras como "congestionamento", "engarrafamento" e "acidente", por exemplo, sejam rotuladas como eventos; e nomes de ruas, avenidas e bairros como localidades. A partir das entidades identificadas, uma aplicação *web* foi construída para exibir a localização dos eventos extraídos em um mapa de fácil visualização.

Os experimentos do trabalho foram realizados dispondo de três conjuntos de dados. A primeira amostra se referia a *tweets* provenientes do público geral, a segunda a *tweets* extraídos de perfis oficiais voltados para a notificação de eventos de trânsito e a terceira a *tweets* oriundos de ambos os tipos de perfis. O modelo apresentou excelente performance na tarefa de identificação de localidades, obtendo precisão acima de 95% em todas as amostras. No entanto, para a tarefa de identificação de eventos, o modelo apresentou precisão de 0% ao trabalhar com a primeira amostra, o que pode ser explicado pelo fato de que *tweets* do público geral, em sua grande maioria, não contêm

menções a eventos de trânsito, formando um conjunto de dados esparso. Em relação à segunda amostra, foi obtida precisão de 100%, evidenciando a melhora de performance do modelo ao fazer uso de *tweets* extraídos de perfis oficiais na fase de treinamento. Os resultados alcançados com a terceira amostra também foram animadores, com precisão de 98%.

Em (DABIRI; HEASLIP, 2019), é demonstrada a utilização de diferentes arquiteturas de Aprendizagem Profunda para classificar um *tweet* em duas ou três categorias, sendo sua categorização responsável por determinar se um texto é relacionado a eventos de trânsito ou não. A classificação em três classes contempla as seguintes categorias: (i) *tweet* não relacionado ao domínio de trânsito; (ii) *tweet* relacionado ao domínio de trânsito e que informa a ocorrência de um incidente de trânsito, tal como uma colisão entre dois veículos em uma avenida; (iii) *tweet* relacionado ao domínio de trânsito, mas que não reporta um incidente de fato, informando apenas uma condição ou nova regra de trânsito. Os microtextos utilizados no estudo são provenientes dos Estados Unidos e escritos em língua inglesa.

Assim, três *datasets* e um dicionário de palavras mais frequentes no domínio de trânsito foram construídos com o objetivo de acelerar o processo de classificação manual (etiquetamento) dos dados. O primeiro *dataset* foi formado por *tweets* extraídos do público geral que mencionassem pelo menos uma das palavras-chave estabelecidas como filtro nas requisições realizadas à API (*Application Programming Interface*) do Twitter. Tal filtro utilizava de duas a quatro palavras reunidas no dicionário construído, de maneira a aumentar a probabilidade de um evento de trânsito ser mencionado no conjunto de *tweets* retornados pela API.

Os experimentos foram realizados utilizando Redes Neurais Convolucionais (*Convolutional Neural Networks* - CNNs), Rede de Memória de Curto e Longo Prazo (*Long Short-Term Memory* - LSTM) e um modelo composto pela combinação de ambas as redes, de modo que as camadas CNNs foram sucedidas por camadas LSTMs. As três arquiteturas foram avaliadas através das métricas de acurácia e F1-Score e apresentaram performances superiores a modelos propostos por trabalhos anteriores. A combinação de CNN pura com o vetorizador *Word2Vec* (MIKOLOV *et al*., 2013) obteve o melhor resultado, atingindo acurácia de 0,986.

Inspirados no trabalho de (DABIRI; HEASLIP, 2019), Teixeira *et al*. (2020) aplicaram uma metodologia similar a *tweets* provenientes do Brasil e escritos em língua portuguesa. No pré-processamento, dado que costuma ser necessário escolher um

tamanho fixo de entrada para as redes, foi feito um truncamento dos *tweets* a partir da 30ª palavra, de forma a contemplar 98,5% dos exemplos da base coletada. No trabalho de (DABIRI; HEASLIP, 2019), por sua vez, o truncamento foi realizado a partir da 13ª palavra, abrangendo 90% dos *tweets*. Essa diferença se justifica na medida em que a língua inglesa costuma ser mais sintética do que a língua portuguesa, de maneira que os *tweets* em inglês têm, em geral, menos palavras do que *tweets* em português.

A etapa de coleta de dados também foi realizada de forma a construir um *dataset* que possua diversidade na sua composição, isto é, que haja *tweets* que sejam provenientes do público geral e *tweets* que tenham sido coletados de perfis de órgãos públicos voltados para o monitoramento do trânsito, como o perfil da Linha Amarela, por exemplo. As arquiteturas empregadas foram as mesmas usadas no trabalho de (DABIRI; HEASLIP, 2019), mas houve uma comparação mais extensa das combinações entre os modelos pré-treinados e os *word embeddings* adotados. De maneira geral, o classificador composto pela combinação de CNN com LSTM demonstrou performance superior às outras redes, o que já era esperado visto que sua arquitetura se beneficia das qualidades de ambas os modelos. Ademais, pôde ser constatado que os vetorizadores utilizados, *Word2vec*, GloVe e FastText, obtiveram performances equivalentes em todas as métricas de avaliação utilizadas, com ligeiro desempenho superior apresentado pelo FastText.

Chamby-Diaz e Bazzan (2019) realizaram a classificação de eventos de trânsito a partir de *tweets* em língua portuguesa provenientes da conta @EPTC_POA. Essa conta é mantida por um órgão público da cidade de Porto Alegre e tem como propósito compartilhar informações sobre o tráfego local. O trabalho leva em consideração diferentes tipos de eventos, a saber: *breakdowns*, *heavy traffic*, *light traffic*, *incidents*, *traffic lights* e *weather*. Um aspecto interessante, deste trabalho, foi adotar uma abordagem de multiclassificação, em que se admite que um *tweet* pode mencionar mais do que um único tipo de evento. Dessa forma, um *tweet* pode conter, por exemplo, eventos dos tipos *incidents* e *traffic lights* ao mesmo tempo.

Os experimentos foram conduzidos a partir da abordagem de *Ensemble Learning*, em que as predições de diferentes modelos são combinadas para melhorar a performance do sistema. Em vista disso, foram utilizados seis algoritmos classificadores treinados por aprendizado supervisionado combinados a um classificador base, de modo que cada tipo de evento correspondeu a um problema de classificação binária. Os algoritmos *Naive Bayes* (NB) e J48 *Decision Tree* (DT) foram empregados como

classificadores base e as doze combinações possíveis entre modelos foram avaliadas para cada tarefa de classificação. As métricas de avaliação dos resultados demonstraram que o classificador *Random k-Labelsets* (RAkEL) combinado ao classificador base *Decision Tree* apresentou, em geral, resultados melhores do que os outros modelos testados. No entanto, seu desempenho não conseguiu ultrapassar a performance do classificador *Binary Relevance Method* (BR).

Em relação ao desempenho por tipo de evento, os eventos dos tipos *weather* e *traffic lights* foram classificados com acurácia por todos os classificadores, à medida que aqueles dos tipos *breakdowns* e *light traffic* se mostraram os mais difíceis de serem classificados corretamente. Os tópicos *incidents* e *heavy traffic* foram classificados com alta acurácia pelos métodos RAkEL (95,1%) e BR (95,3 %), respectivamente.

Mais recentemente, Bedi e Toshniwal (2022) empregaram modelos de *word embeddings* e arquiteturas de Aprendizagem Profunda nas tarefas de análise de sentimento e classificação de reclamações. Para a realização da pesquisa, foram extraídos *tweets* em inglês que mencionassem palavras-chave relacionadas à eletricidade e cujas localizações correspondessem às cidades de Delhi e Bangalore na Índia. A primeira etapa do estudo consistiu em identificar os sentimentos dos usuários nos microtextos analisados. Em seguida, os *tweets* associados a sentimentos negativos foram processados e utilizados na criação de um modelo de classificação de reclamações, que rotula os *tweets* em duas possíveis classes. A classe H (*High-Priority Complaint*) inclui os *tweets* que foram reportados por um grupo de indivíduos e que, portanto, devem receber mais atenção. Essa classe também engloba os microtextos que fazem referência a problemas mencionados frequentemente pelos usuários. Já a classe L (*Low-Priority Complaints*) contempla os *tweets* que foram reportados por apenas um indivíduo ou por um grupo pequeno de usuários, muitas vezes relacionados a problemas pontuais de energia em residências.

As arquiteturas de redes neurais utilizadas para construir os modelos de classificação de sentimento e de reclamações foram LSTM, Bi-LSTM e CNN. O modelo estado da arte *Bi-Directional Encoded Representation for Transformer* (BERT) também foi empregado para melhorar a performance de predição, apresentando a maior acurácia de classificação. A base de dados foi dividida agrupando os *tweets* extraídos por localidade, de maneira que o *dataset* Delhi atingiu dois mil *tweets* e o *dataset* Bangalore alcançou mil e setecentos.

No que concerne aos modelos de vetorização por *word embeddings*, os algoritmos *Word2Vec*, GloVe e FastText foram integrados aos modelos de classificação com o objetivo de aperfeiçoar seus resultados. Em todos os modelos em que foi aplicado, FastText apresentou a melhor performance, o que pode ser explicado através de sua capacidade de gerar representações vetoriais (*embeddings*) para palavras menos frequentes, ou até mesmo raras, presentes no *corpus*. Dentre as arquiteturas utilizadas na construção do modelo de reclamações, sua combinação com a rede CNN obteve a maior taxa de acurácia (94,06%) no *dataset* Bangalore. No entanto, a combinação não superou a acurácia atingida pelo modelo BERT, que obteve 95,78% no mesmo cenário.

Em (FERREIRA; DUARTE; UGULINO, 2022), é proposta uma metodologia para a construção de um Sistema de Informação capaz de coletar estatísticas de eventos de segurança pública (tiroteios, operações policiais, assaltos, entre outros) a partir de notificações feitas por usuários em suas redes sociais. Além disso, um modelo de reconhecimento de eventos de segurança pública também foi construído, atingindo uma acurácia de 95%. No contexto do trabalho, a área de ocorrência dos eventos corresponde à cidade do Rio de Janeiro e o Twitter foi a rede social escolhida como fonte dos dados.

A metodologia proposta é representada por um *pipeline* de mineração de dados, composto por diversas etapas sequenciais, divididas em três blocos, que vão desde a parametrização e a definição dos atributos até a forma como os dados processados serão apresentados ao usuário. O primeiro bloco da metodologia, chamado "*Parameterization and Initiation*", é constituído por tarefas responsáveis por realizar todas as configurações iniciais para que a metodologia funcione corretamente. Além disso, o bloco também contém tarefas voltadas para a coleta dos dados das redes sociais e para a separação dos dados coletados entre microtextos originais e microtextos repostados, de forma que seja possível processar os microtextos originais em primeiro lugar.

Duas tarefas do *pipeline* não fazem parte de nenhum bloco: "*Get Initial Publication*" e "*Label Publications*". A primeira é responsável por garantir que todos os microtextos repostados estejam associados a um microtexto original, enquanto a segunda é responsável pelas atividades de aprendizagem, treinamento e avaliação dos algoritmos, tornando o sistema de reconhecimento de entidades mais eficiente. Desse modo, o número de execuções da tarefa "*Label Publications*" irá depender da performance, e a necessidade de ajuste, dos modelos utilizados.

O segundo bloco, chamado "*Preparation and Mining*", é responsável por realizar as tarefas de tratamento e processamento dos dados recebidos do bloco anterior e também

pela extração das entidades contidas nesses dados. Assim, são conduzidas diversas etapas de limpeza, consolidação, integração e separação de dados para que em seguida eles sejam submetidos aos modelos de reconhecimento de entidades. Dois modelos foram utilizados: um modelo nativo da biblioteca Spacy e outro criado especificamente para a identificação de eventos de segurança pública. A arquitetura do modelo proposto é composta por uma camada de *embedding* que combina o uso de subpalavras e *Bloom embeddings* como estratégia, a fim de comprimir a representação da entrada; e uma rede neural profunda com conexões residuais (ver AGGARWAL, 2023).

Os microtextos possuem quatro entidades que precisam de identificação: o evento de segurança pública em si, sua localização, data e hora. O modelo da biblioteca Spacy é responsável por identificar as últimas três entidades, mas o evento de segurança pública é identificado pelo modelo construído especificamente para a metodologia. Por fim, é executada uma tarefa de recuperação das coordenadas geográficas dos eventos para que elas sejam salvas na base de dados.

O terceiro bloco, nomeado "*Marking and Presentation*" tem como objetivo permitir que os usuários vejam os resultados na forma de indicadores, possibilitando o processo de análise e tomada de decisão. Dessa forma, são executadas tarefas que preparam e filtram os dados para que em seguida sejam contabilizadas métricas e estatísticas.

A implementação da metodologia foi realizada utilizando um *bot* na linguagem Python que persistiu os dados obtidos através da API do Twitter em um banco de dados PostgreSQL. No total, 12.000 *tweets* foram coletados durante o ano de 2020. Os resultados obtidos utilizaram o conjunto de testes referente às últimas sete semanas de coleta de dados, em que as acurácias do modelo na tarefa de extração de entidades atingiram seu melhor resultado (99,30%) na primeira semana e seu pior resultado (86,25%) na última. Em relação à tarefa de reconhecimento de localidades, o modelo atingiu seu melhor desempenho (97,93%) na terceira semana e seu pior desempenho (91,25%) na última semana. Para a visualização dos resultados, uma aplicação *web* foi construída utilizando as bibliotecas Django, um *framework* de desenvolvimento *web* da linguagem Python, e Folium, que permite a criação de mapas e gráficos dinâmicos para a exibição dos eventos de segurança pública identificados.

O trabalho de (QUDAR; MAGO, 2020) propõe dois modelos de representação de linguagem criados para lidar especificamente com dados extraídos da rede social Twitter. Chamados de TweetBERTs, os modelos foram pré-treinados em milhões de *tweets* em inglês e superaram as performances de modelos pré-existentes como

ALBERT, BioBERT e SciBERT na tarefa de análise dos microtextos extraídos da plataforma.

Para a etapa de pré-treino, os dados utilizados foram coletados a partir da ferramenta *big data analytics platform* (MENDHE *et al*., 2020). Dois conjuntos de dados foram adquiridos: Corpus140, com 140 milhões de *tweets*, e Corpus540, com 540 milhões de *tweets*. Os microtextos foram extraídos a partir das 100 contas com mais seguidores, de forma a assegurar que os *tweets* fossem extraídos de perfis autênticos, e das 100 *hashtags* mais mencionadas na plataforma.

O modelo TweetBERTv1 foi inicializado usando os pesos do modelo BERT e em seguida pré-treinado na base de *tweets* Corpus140. No caso do modelo TweetBERTv2, a inicialização foi feita a partir dos pesos do modelo ALBERT e em seguida o modelo foi pré-treinado na base de *tweets* Corpus540. Para demonstrar a efetividade da solução proposta para análise de textos provenientes do Twitter, os modelos foram ajustados finamente (*fine-tuned*) em duas tarefas de PLN: análise de sentimento e classificação.

Os modelos foram avaliados em trinta e um *datasets* de diferentes domínios, de forma que seus desempenhos foram testados não somente em *tweets*, mas também em textos científicos, biomédicos e de contexto geral. Ademais, sete outros modelos baseados na arquitetura BERT foram avaliados, como ALBERT e RoBERTa, possibilitando a comparação entre os resultados obtidos pelos modelos TweetBERT e os resultados obtidos pelos modelos pré-existentes na literatura.

Os resultados obtidos mostraram que os modelos TweetBERT obtiveram o melhor desempenho nos experimentos conduzidos com *datasets* do Twitter, conforme esperado. O modelo TweetBERTv2 obteve uma acurácia de 95,18% quando avaliado utilizando o *dataset* Sentiment140, um *dataset* para análise de sentimento composto por 1.600.000 *tweets* anotados dentre as categorias neutro, positivo ou negativo; enquanto os modelos ALBERT e RoBERTa obtiveram acurácias de 90,59% e 86,71%, respectivamente.

Além disso, ALBERT e RoBERTa apresentaram as melhores acurácias nos *datasets* de domínio geral, atingindo respectivamente 96,99% e 96,4% no *dataset* GLUE *The Stanford Sentiment Treebank* (SST). Nos *datasets* de domínio biomédico, o modelo RoBERTa obteve o melhor desempenho de maneira geral e os modelos TweetBERT obtiveram valores de precisão maiores do que o modelo BioBERT em quase todos os *datasets*, excetuando-se apenas o *dataset* EU-ADR. No domínio científico, os modelos TweetBERT obtiveram a maior acurácia em três dos quatro *datasets* avaliados, de forma que o TweetBERTv1 obteve a maior acurácia (68,85%) no

*dataset* Sci-RE e o modelo TweetBERTv2 obteve as maiores acurácias nos *datasets* Paper Field (66,49%) e SciCite (88,56%).

Por seu turno, apesar de não utilizar dados extraídos de redes sociais, o modelo de linguagem FinBERT-PT-BR proposto em (SANTOS; BIANCHI; COSTA, 2023) foi treinado de modo semi-supervisionado a partir do ajuste fino do modelo BERTimbau (SOUZA; NOGUEIRA; LOTUFO, 2020), com notícias coletadas das plataformas Valor Econômico, Exame e InfoMoney, que são destinadas a notícias e informações sobre o mercado financeiro.

Os modelos BERTimbau são modelos de linguagem BERT pré-treinados para o português brasileiro. Tais modelos foram treinados utilizando uma grande quantidade de dados textuais a partir do *dataset* BrWaC (*Brazilian Web as Corpus*) que contém 2,68 bilhões de elementos textuais de 3,53 milhões de documentos. No entanto, como os modelos BERTimbau são de escopo geral, tornou-se necessária a etapa de *fine-tuning* para a construção de um modelo no contexto de mercado financeiro.

A extração dos dados ocorreu através da ferramenta Scrapy[2] e 2,7 milhões de sentenças foram coletadas a partir de notícias do mercado financeiro publicadas entre 2006 e 2022. Após a etapa de limpeza dos dados, 1,6 milhão de sentenças foram alcançadas. O modelo FinBERT-PT-BR foi treinado utilizando PyTorch e Hugging Face, tendo como pesos iniciais os pesos do modelo BERTimbau. Para avaliar seu desempenho, foi aplicada a métrica de perplexidade, que indica o quão bom o modelo é em prever uma palavra dado o seu contexto. Quanto menor o valor de perplexidade, melhor o resultado. O modelo atingiu uma perplexidade de 1,24, que é considerado um resultado bastante satisfatório dado que a perplexidade alcançada pelo modelo BERTimbau foi de 1,51.

Além do modelo FinBERT-PT-BR, o trabalho também propõe a construção do modelo SentFinBERT-PT-BR, um classificador de sentimentos de notícias da área financeira, a partir do treinamento do modelo FinBERT-PT-BR em uma base de dados anotada de notícias do mercado financeiro. Assim, ao receber um texto como entrada, o modelo deve ser capaz de classificá-lo como positivo, negativo ou neutro de acordo com o evento mencionado na notícia. A base de dados utilizada para o pré-treinamento foi composta por 503 textos anotados manualmente, sendo 160 positivos, 203 negativos e 140 neutros. Ademais, foi necessário adicionar uma camada de classificação (com

[2] https://scrapy.org/

ativação *Softmax*) na primeira dimensão de saída do BERT, a fim de contemplar a tarefa de análise de sentimento.

Para evitar o esquecimento de informações durante o treinamento do modelo SentFinBERT-PT-BR, tendo em vista que se utilizou o FinBERT-PT-BR como modelo pré-treinado com seus pesos congelados, a técnica de *Gradual Unfreezing*, que consiste em desbloquear a atualização dos pesos da rede neural de forma gradativa, foi aplicada. Além disso, foi empregada a técnica de validação cruzada com a base de treino dividida em cinco partes, sendo quatro partes para treinamento e uma parte para validação. O processo de validação cruzada foi realizado cinco vezes e o modelo de sentimento com a melhor convergência em relação à validação foi escolhido para realizar a tarefa de classificação de sentimentos na base de teste.

Com o objetivo de comparar o desempenho do modelo construído, foi treinado um segundo modelo, chamado Sent-BERTimbau, que também empregou a técnica de *transfer learning* do BERTimbau original para a tarefa de classificação de sentimentos. Os resultados dos testes mostraram que o modelo SentFinBERT-PT-BR obteve uma acurácia de 0,76 enquanto o modelo Sent-BERTimbau atingiu uma acurácia de 0,67.

Como exemplo de aplicabilidade do modelo proposto, foi desenvolvido um índice de sentimentos de mercado. Um índice de sentimentos consiste em uma série temporal de sentimentos construída a partir da classificação de notícias ao longo do tempo. Dessa forma, foi possível avaliar a relação entre as classificações feitas pelo modelo e fatos ocorridos no Brasil e no mundo no período de tempo analisado.

Em (NIRBHAYA; SUADAA, 2023), é realizado um estudo comparativo entre o desempenho de diferentes modelos na tarefa de detecção de incidentes de trânsito na cidade de Jacarta a partir de dados coletados do Twitter. O estudo utiliza uma abordagem de classificação *multi-label*, de forma que um *tweet* possa ser classificado em mais de uma categoria de evento de trânsito, sendo elas: *smooth traffic*, *heavy traffic/jammed*, *weather conditions* e *traffic accident*. Os *tweets* foram extraídos da conta @TMCPoldaMetro entre agosto e outubro de 2022 através da biblioteca Twint[3], contabilizando um *dataset* com 1066 registros.

O processo de etiquetamento foi realizado de forma manual e em duas etapas. Inicialmente, os dados coletados foram etiquetados entre as categorias “relevante para incidentes de trânsito” e “não relevante”. Apenas os dados relevantes, 881 tweets,

[3] https://github.com/twintproject/twint

seguiram para a segunda etapa de etiquetamento, em que cada *tweet* foi etiquetado de acordo com os eventos de trânsito específicos presentes em seu texto. As categorias correspondentes aos eventos detectados em um dado *tweet* receberam o valor "*available*" e as categorias não detectadas receberam o valor "*not available*". Desse modo, em relação aos *tweets* com apenas uma categoria, 621 apresentaram a categoria *smooth*, 105 apresentaram a categoria *jammed*, 11 apresentaram a categoria *weather* e 49 apresentaram a categoria *accident*. Dentre os *tweets* com duas categorias, 36 apresentaram as categorias *smooth* e *jammed*, cinco apresentaram as categorias *smooth* e *weather*, 18 apresentaram as categorias *smooth* e *accident*, cinco apresentaram as categorias *jammed* e *weather*, 27 apresentaram as categorias *jammed* e *accident* e apenas um *tweet* apresentou as categorias *weather* e *accident*. Finalmente, quanto aos *tweets* com três categorias, dois apresentaram as categorias *smooth, jammed* e *accident* e somente um *tweet* apresentou as categorias *smooth, jammed* e *weather*.

Foram desenvolvidos modelos de Aprendizagem de Máquina tradicional com *Support Vector Machine* (SVM), *Complement Naïve Bayes* e Regressão Logística combinados à técnica TF-IDF, que calcula o quão importante uma palavra é em um texto pela sua frequência de utilização, sem considerar seu significado. Um modelo de Aprendizagem Profunda com LSTM também foi desenvolvido e, por fim, foi avaliado o desempenho de um modelo obtido a partir do *fine-tuning* do modelo IndoBERT (KOTO *et al*., 2020), uma variante do BERT treinada em um grande *corpus* de língua indonésia.

Os resultados dos experimentos, realizados utilizando a técnica de validação cruzada com a base de treino dividida em cinco partes, mostraram que a melhor performance foi alcançada pelo modelo construído a partir do *fine-tuning* do IndoBERT, que atingiu uma acurácia de 99,26% e um F1-Score de 99,10%. Por seu turno, o modelo de LSTM atingiu uma acurácia de 96% e F1-Score de 95%. Com desempenho similar ao do modelo de SVM, o modelo de Regressão Logística obteve uma acurácia de 98,69% e um F1-Score de 98,37%, enquanto o modelo de SVM obteve uma acurácia de 98,81% e um F1-Score de 98,54%. Por fim, o modelo de *Complement Naïve Bayes* alcançou 93% de acurácia e um F1-Score de 92%.

# 4 Arquiteturas Adotadas

O presente capítulo introduzirá os modelos utilizados na realização dos experimentos, especificamente na tarefa de identificação de eventos relevantes de trânsito em microtextos publicados na rede social Twitter. Foram adotadas as três arquiteturas apresentadas em Teixeira *et al*. (2020) (CNN, LSTM e um modelo combinado), além de um modelo de linguagem obtido a partir do *fine-tuning* do BERTimbau (SOUZA; NOGUEIRA; LOTUFO, 2020).

## 4.1 CNN

O modelo de CNN foi implementado utilizando a arquitetura descrita na Figura 9. O primeiro passo do modelo, não considerando as etapas de transformação da entrada, consiste na aplicação da camada de convolução, que utiliza 200 filtros para a extração de características e emprega a função ReLu como função de ativação. Além disso, a janela de convolução (tamanho dos filtros) é igual a 2 x *n*, ou seja, a função de convolução é aplicada em janelas compostas por duas palavras com *n* dimensões. No contexto deste trabalho, o número de dimensões não é fixo já que vetorizadores com diferentes dimensões foram aplicados para fins de comparação.

Em seguida, são aplicadas as camadas de *max pooling* e *dropout*. A primeira é responsável por reduzir cada janela a que é aplicada a um valor único, diminuindo a espacialidade do mapa de ativação. Por sua vez, a camada de *dropout* irá descartar aleatoriamente 50% das ativações obtidas após a camada de *max pooling*, de modo a evitar o *overfitting* e contribuir para a generalização do modelo. No entanto, a camada de *dropout* é utilizada somente durante o treinamento, de forma que o descarte seja ignorado durante a etapa de inferência.

Na sequência, a camada de achatamento (*flatten*) é utilizada para converter o mapa de ativação resultante da camada anterior em um *array* unidimensional,

permitindo que ele seja fornecido como entrada para a última camada da rede. O último passo do modelo corresponde a uma camada de saída densa que utiliza a função *Softmax* para associar, de maneira ponderada, as características extraídas às classes possíveis. Desse modo, a saída produzida pela rede corresponde a uma distribuição de probabilidade, contendo uma componente para cada classe possível. Ressalta-se que, no problema abordado neste trabalho, há duas classes possíveis: a relevância ou não de um *tweet* em relação ao domínio de trânsito.

A função de custo e o otimizador adotados foram *Categorical Cross Entropy* e Adam, respectivamente. O otimizador é o algoritmo responsável pela minimização da função de custo e que, com base nesse objetivo, rege as atualizações dos parâmetros da rede ao longo do treinamento. O Adam (*Adaptive Moment Estimation*) corresponde a uma evolução do algoritmo de Gradiente Descendente Estocástico, buscando suplantar algumas limitações deste. Proposto por Kingma *et al*. (2014), o algoritmo utiliza uma taxa de aprendizado (*learning rate*) única para cada parâmetro, que será atualizada ao longo do treinamento. Ademais, no processo de atualização dos parâmetros da rede, em vez de se basear somente no primeiro momento do gradiente, o algoritmo também considera a média dos segundos momentos do gradiente (AGGARWAL, 2023).

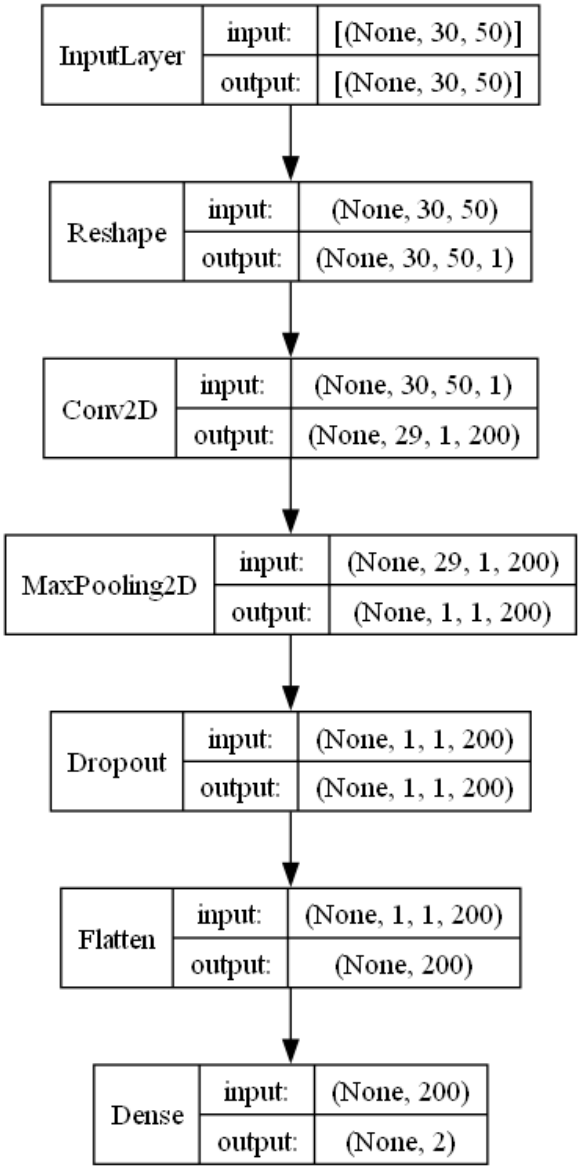


Figura 9: Representação gráfica da arquitetura do modelo baseado em CNN.

### 4.2 LSTM

A arquitetura do modelo de LSTM adotado é descrita pela Figura 10. A primeira etapa do modelo corresponde a uma camada LSTM configurada com 50 unidades dimensionais, permitindo que 50 recorrências sejam executadas em paralelo. A função de ativação utilizada é a tangente hiperbólica (tanh), definida como padrão para camadas LSTM na literatura.

Em seguida, aplica-se a camada de *dropout*, que descarta aleatoriamente 50% das ativações obtidas na camada anterior, assim como para o modelo CNN. Dado que a saída da camada LSTM é um vetor linear, não há necessidade de se empregar uma camada de achatamento, permitindo que a camada seguinte seja a própria camada saída,

que é densa. Assim como na implementação da CNN, a função *Softmax* é adotada na camada de saída, produzindo uma saída que consiste em uma distribuição de probabilidade com duas componentes, uma para cada classe existente.

Por fim, a regularização L2 foi aplicada aos parâmetros da entrada como função restritiva, com o objetivo de regularizá-los. A função restritiva penaliza a inclusão de novos parâmetros ao adicioná-los no cálculo da função de custo. Desse modo, parâmetros não são retidos pelo modelo a não ser que contribuam para o seu efetivo aprendizado, controlando-se assim o número total de parâmetros. No caso da função L2, a penalização corresponde a adição de $\lambda w^2$ ao parâmetro utilizado na camada densa, sendo *w* o valor original do parâmetro de entrada e $\lambda$ uma constante de valor 0,01.

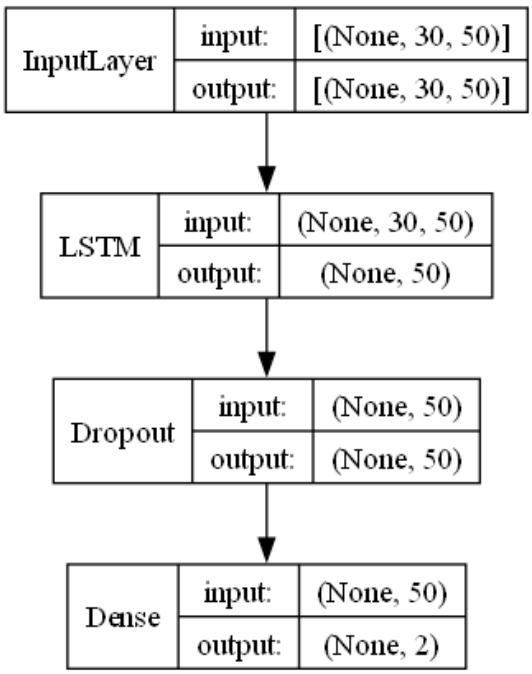


Figura 10: Representação gráfica da arquitetura do modelo baseado em LSTM.

### 4.3 CNN + LSTM

O modelo combinado é composto pelas redes CNN e LSTM, de modo que a entrada é submetida primeiro a uma rede CNN e, em seguida, a uma rede LSTM com 100 unidades, conforme exibido na Figura 11. O algoritmo inicia com uma camada convolucional que apresenta as mesmas configurações da rede descrita na Seção 4.1. Na sequência, uma camada de *reshape* é utilizada para transformar a saída da camada de CNN em uma entrada apropriada para a camada de LSTM. Desse modo, a saída da camada de *reshape* será um vetor de três dimensões contendo o tamanho da amostra (*batch size*), o número de iterações (equivalente ao tamanho da sequência), e o número

de características, que é igual ao número de filtros utilizados pela CNN. Em seguida, é aplicada uma camada de *dropout* de 25% cujo resultado será fornecido como entrada à camada de LSTM. Após isso, uma nova camada de *dropout* de 50% é aplicada, e seu resultado é então submetido à camada de saída que adota a função *Softmax* como função de ativação.

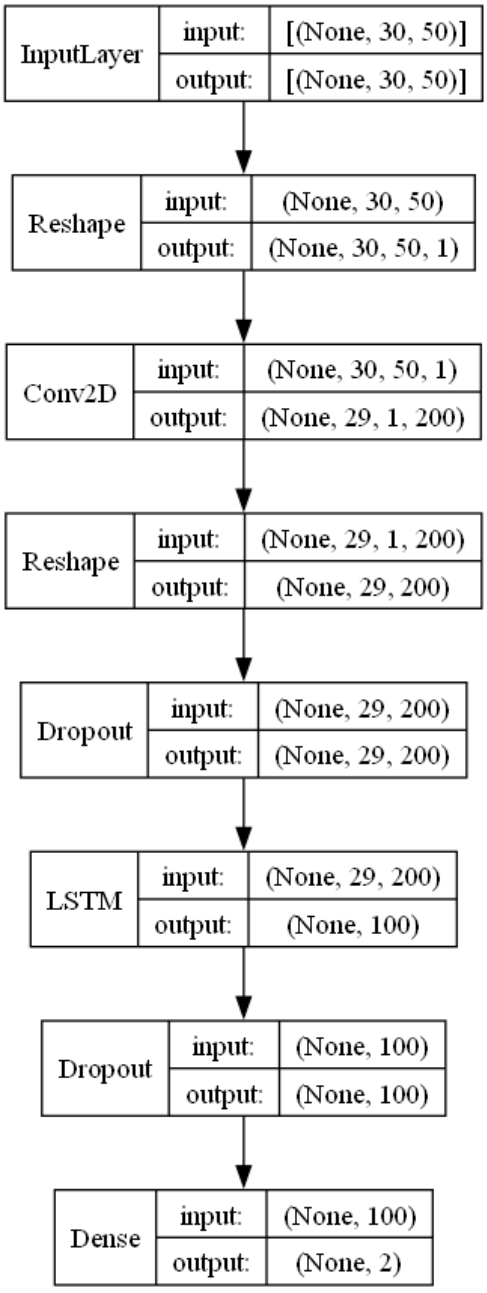


Figura 11: Representação gráfica da arquitetura do modelo construído a partir da combinação de CNN e LSTM.

### 4.4 BERTimbau

Também se adotou neste trabalho um modelo de classificação baseado na arquitetura *Transformer Encoder*, ilustrado pela Figura 12. Tal modelo foi obtido a partir do *fine-tuning* do modelo pré-treinado BERTimbau no *dataset* de microtextos extraídos do Twitter contendo eventos relevantes de trânsito e *tweets* de domínio geral, a ser descrito na Seção 5.2. As camadas do modelo obtido são inicializadas com as mesmas configurações e pesos do BERTimbau, de forma que a diferença entre a arquitetura original e a arquitetura resultante corresponde à adição de uma camada de classificação na primeira dimensão de saída do BERTimbau.

A camada de classificação é composta por uma camada de *dropout* de 10% e uma camada densa com ativação linear que produz duas probabilidades como saídas, uma para cada classe do problema abordado. Sua inclusão é necessária porque o BERTimbau, assim como o BERT, é um modelo de linguagem orientado para tarefas *sequence-to-sequence*, ou seja, para transformar uma sequência recebida como entrada em outra que é gerada como saída. Além disso, dado que o BERTimbau é originado a partir do ajuste fino do BERT, para o português, em um escopo generalista, o processo de *fine-tuning* possibilitará que o modelo se especialize na tarefa de interesse deste trabalho, ou seja, na classificação de relevância de microtextos quanto ao domínio de trânsito.

Finalmente, os pesos das camadas oriundas do BERTimbau são congelados ao longo do processo de treinamento, a fim de que não sejam modificados, à medida que os pesos da camada de classificação são inicializados aleatoriamente e ajustados no treinamento.

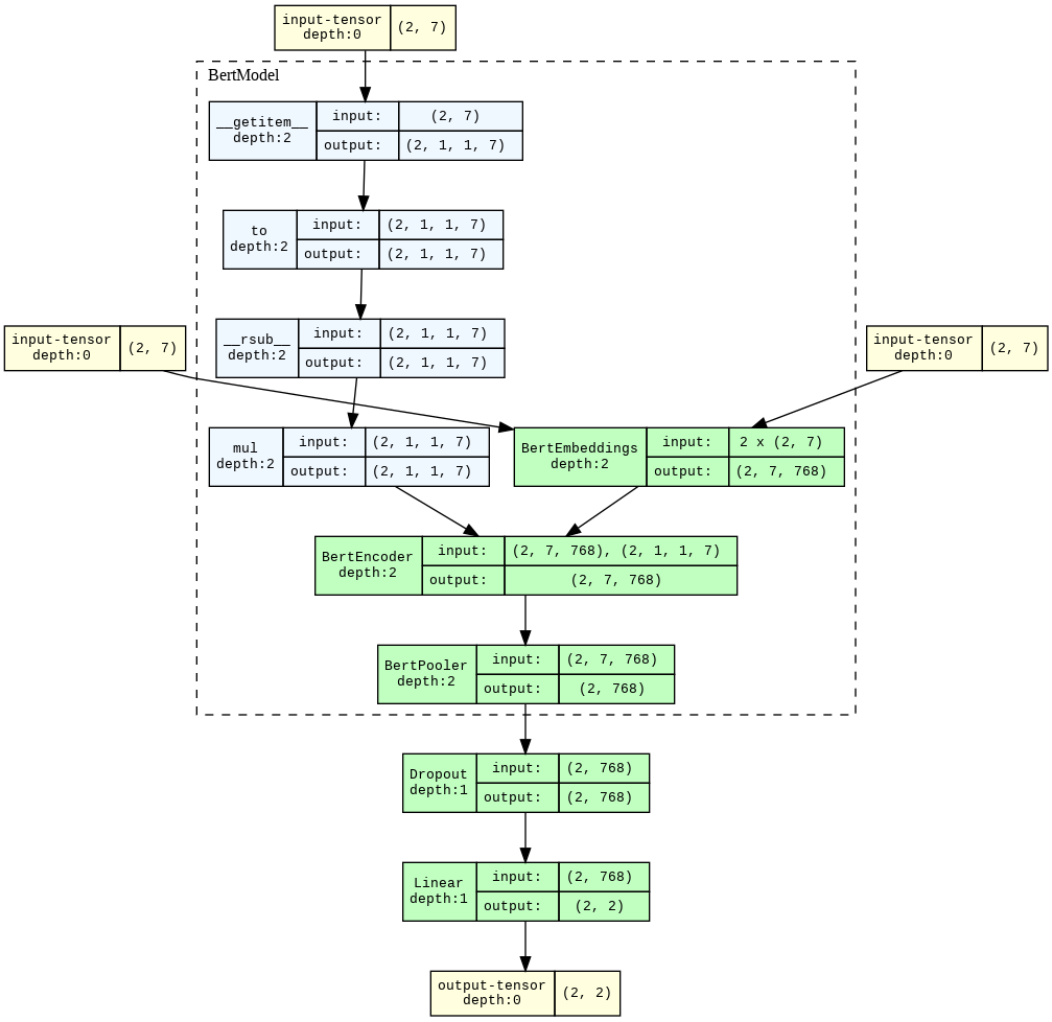


Figura 12: Representação gráfica da arquitetura do modelo construído a partir do *fine-tuning* do BERTimbau.

# 5 Experimentos Computacionais

Neste capítulo, são apresentados os experimentos computacionais conduzidos na tarefa de identificação de eventos relevantes de trânsito, assim como seus resultados obtidos. A Figura 13 mostra como o processo de experimentação foi realizado, desde a coleta dos *tweets* até a fase de inferência, em que os modelos obtidos são capazes de classificar os dados extraídos.

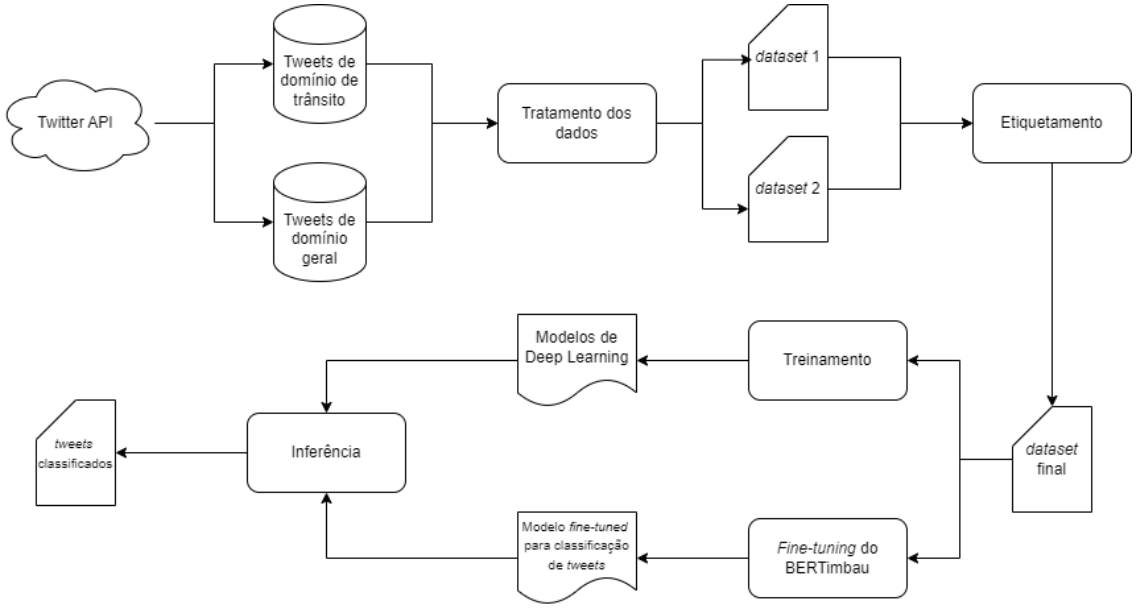


Figura 13: Etapas do processo de treinamento.

## 5.1 Ambiente Computacional

O ambiente computacional utilizado nos experimentos foi composto pelo processador 12th Gen Intel(R) Core(TM) i7-12700H, 16GB de memória RAM DDR5 e sistema operacional Windows 11 Home. A implementação dos modelos foi feita na linguagem de programação Python na versão 3.9, usando-se a biblioteca PyTorch na versão 2.1.2[4] para os experimentos com o modelo BERTimbau e a biblioteca Keras na versão 2.10.0[5] para os experimentos com os modelos de Aprendizagem Profunda tradicionais. Além disso, os experimentos com o BERTimbau foram realizados através da plataforma Hugging Face, que disponibiliza grandes bases de dados e modelos pré-treinados que podem ser acessados através de sua biblioteca Python. Todos os experimentos, incluindo os experimentos com o BERTimbau, foram executados sem o uso de GPU devido a uma limitação de memória ao rodar os modelos de Aprendizagem Profunda tradicionais combinados a vetorizadores com muitas dimensões. Os códigos desenvolvidos nesta pesquisa, assim como o resultado dos experimentos, estão disponíveis publicamente em um repositório criado no GitHub[6].

## 5.2 *Dataset* adotado

O *dataset* utilizado neste estudo foi construído a partir de microtextos extraídos da rede social Twitter, através da Twitter API[7]. Na data de extração, o modo de acesso básico e gratuito, chamado *Essential*, não fornecia entrada a todo o histórico de postagens. Por esse motivo, foi utilizado o modo *Academic Research*[8], voltado para projetos de pesquisa e acessado somente após o preenchimento de um formulário de solicitação e a sua respectiva aprovação pela organização Twitter. Essa solicitação foi realizada antes da mudança de política da plataforma, que fez parte de uma série de alterações que incluiu até mesmo a mudança do nome da rede social para X[9]. Atualmente, segundo a documentação atualizada[10], os dados compartilhados na rede só podem ser extraídos através dos planos *Basic*, *Pro* e *Enterprise*, que são acessos pagos. O plano *Free*, que

[4] https://pytorch.org/get-started/previous-versions/
[5] https://keras.io/2.16/api/
[6] https://github.com/manuela-blanco/traffic_events_detection_dl_llm
[7] Disponível em: <https://developer.twitter.com/en/products/twitter-api>. Acesso em: 24 Set. 2023.
[8] Ao final da escrita deste estudo, o modo de acesso Academic Research havia sido descontinuado.
[9] https://x.com
[10] Disponível em: <https://developer.x.com/en/docs/twitter-api/getting-started/about-twitter-api#item0>. Acesso em: 02 Jul. 2024.

pode ser acessado sem custo, permite que a API seja utilizada apenas para a criação de postagens.

Inicialmente, foram extraídos dois *datasets*, sendo o primeiro composto por *tweets* provenientes de perfis voltados ao domínio de trânsito e o segundo composto por *tweets* oriundos de perfis do público geral. Em ambos os casos, os *tweets* foram filtrados de forma a corresponder aos dados de interesse para este estudo, ou seja, que tivessem sido escritos em português e publicados no Brasil entre os dias 1 de janeiro e 31 de março de 2023, que foi o período de tempo adotado para coleta. Do mesmo modo, *tweets* decorrentes do compartilhamento de outro *tweet*, chamados de *retweets*, não foram considerados no processo de extração. Os critérios mencionados foram aplicados às requisições feitas à Twitter API, que permite que filtros, chamados de *search operators*, sejam enviados no corpo da requisição de forma a construir uma *query* para a consulta e extração dos dados.

Para a construção do primeiro *dataset*, isto é, aquele composto por *tweets* do domínio de trânsito, foram utilizados os perfis: @OperacoesRio, @Arteris_AFL, @_ecoponte, @LinhaAmarelaRJ, @CETSP_ e @_dersp. Em seguida, seguindo a abordagem proposta por (DABIRI; HEASLIP, 2019), foram contabilizadas as cinco palavras mais frequentes dentre os *tweets* coletados, sendo elas: "sentido", "minutos", "normal", "fluxo" e "rio". A fim de que elas pudessem ser utilizadas como filtro no processo de coleta do segundo *dataset*, a palavra "rio" foi desconsiderada e substituída por "km", a sexta palavra mais frequente. Dado que quatro dos seis perfis utilizados reportam eventos localizados na cidade do Rio de Janeiro, o grande número de ocorrências da palavra "rio" provavelmente indica um viés.

Na construção do segundo *dataset*, foram coletados apenas os microtextos que mencionassem pelo menos uma das palavras obtidas na etapa anterior e cujos perfis de origem não tivessem sido utilizados na construção do primeiro conjunto de dados. Assim, foram extraídos 40.090 *tweets* pertencentes ao domínio de trânsito e 34.992 *tweets* do público geral, somando 75.082 *tweets* ao final da etapa de extração.

### 5.3 Tratamento dos dados

Os microtextos passaram por diversas fases de pré-processamento de maneira a remover dados irrelevantes e melhorar a acurácia dos modelos utilizados posteriormente. A manipulação dos *datasets* foi realizada de modo automatizado com o emprego da

biblioteca Pandas[11], que disponibiliza múltiplas funções para o tratamento de dados tabulares através de uma estrutura bidimensional chamada *DataFrame*. A primeira etapa consistiu na transformação de todos os *tweets* em caixa baixa e na remoção de URLs, caracteres especiais, *emojis*, sinais de pontuação e uma lista de *stop words* da língua portuguesa obtida através da biblioteca NLTK[12]. Palavras que compõem a lista de *stop words* de uma língua costumam ser utilizadas muito frequentemente, mas carregam pouca relevância semântica. No caso da língua portuguesa, as palavras "não", "ou", "para", por exemplo, são consideradas *stop words*.

Adicionalmente, os símbolos "@" e "#" foram removidos por não adicionar nenhum valor semântico aos modelos de aprendizagem. O símbolo "@" é utilizado para marcar e referenciar outros usuários em uma publicação e o símbolo "#" é utilizado antes de uma palavra ou frase para classificar ou facilitar a busca por uma publicação.

Em seguida, também seguindo a abordagem proposta por (DABIRI; HEASLIP, 2019), foram removidos *tweets* com menos do que cinco palavras por possuírem baixa carga semântica. Finalmente, como última etapa, foi utilizada a estratégia de *undersampling*, de maneira que foram removidos, de forma aleatória, sete mil *tweets* provenientes do primeiro *dataset*. Essa estratégia foi adotada com o objetivo de melhorar a proporção entre *tweets* que fazem menção a eventos de trânsito (primeiro *dataset*) e *tweets* que não fazem menção a eventos de trânsito (segundo *dataset*) na composição do *dataset* final.

Ao final do pré-processamento, foram totalizados 64.237 *tweets*, sendo 32.136 *tweets* do primeiro *dataset*, formado por *tweets* do domínio de trânsito, e 32.101 *tweets* do segundo *dataset*, formado por *tweets* do público geral.

### 5.4 Etiquetamento

A tarefa a ser abordada, sob o ponto de vista da Aprendizagem de Máquina, corresponde a um problema de classificação binária, em que se deseja classificar os exemplos (*tweets*) em relação a duas classes: a relevância ou não ao domínio de trânsito. Tem-se, assim, um problema de aprendizado supervisionado, o que exige necessariamente a provisão dos rótulos, isto é, as classes, de cada um dos exemplos.

[11] https://pandas.pydata.org/
[12] https://www.nltk.org/

Para tal, foi efetuado um processo de etiquetamento manual por três pessoas, em que cada *dataset* foi processado separadamente e cada etiquetador revisou uma parte de cada *dataset*. Devido às diferenças entre os métodos de extração, era esperado que o primeiro *dataset* contivesse predominantemente *tweets* do domínio de trânsito, enquanto uma prevalência de *tweets* fora do domínio de trânsito era esperada para o segundo. Assim, a utilização de dois *datasets* permitiu que o processo de etiquetamento fosse realizado mais rapidamente, visto que o tipo de *tweet* dominante em cada conjunto já era conhecido. Ao final do etiquetamento, um *dataset* único foi construído a partir da concatenação dos dois conjuntos iniciais, contabilizando 29.632 registros rotulados como "Evento relevante de trânsito" e 34.605 registros rotulados como "Evento não relevante".

Em relação aos critérios utilizados para estabelecer se um evento de trânsito mencionado em um *tweet* é relevante ou não, foi levado em consideração se a mensagem contida no *tweet* faz referência a uma localização. Isso significa que *tweets* que citam ocorrências relacionadas ao trânsito sem indicar sua localização não foram classificados como relevantes para o domínio de trânsito. Esse cenário pode ser exemplificado pelo *tweet* da Figura 14: "eu tô a 50 minutos tentando chegar em casa, que trânsito horrível", que relata um congestionamento, mas não menciona textualmente o local do evento.

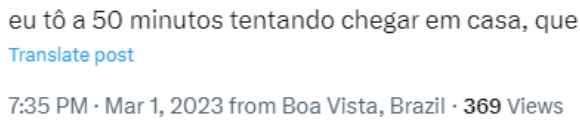


Figura 14: Exemplo de *tweet* com evento de trânsito não relevante.

O mesmo foi aplicado a *tweets* que mencionam eventos climáticos, mas não fazem referência textualmente a uma localização, como o *tweet* da Figura 15, que relata: "Fique atento ao dirigir na chuva! A aquaplanagem é um fenômeno em que um veículo, ao passar sobre uma camada de água, perde o atrito com o asfalto e desliza. Garanta a segurança de todos, reduza a velocidade e mantenha uma distância segura do veículo à frente!".

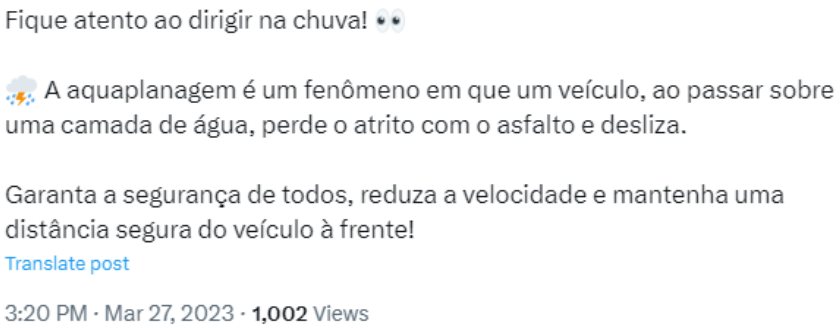


Figura 15: Exemplo de *tweet* com evento climático não relevante para o escopo de trânsito.

Por sua vez, um evento de trânsito relevante pode ser exemplificado pelo *tweet* da Figura 16 que relata o evento "Atualização: Faixa liberada no km 321, em Niterói, sentido Rio de Janeiro." O evento mencionado é relevante porque faz menção a uma localidade específica, possibilitando que, em um contexto em que haja uma dada aplicação que consuma a classificação feita pelo modelo de aprendizagem, outros usuários possam utilizar a informação no processo de tomada de decisão na escolha de um trajeto no trânsito.

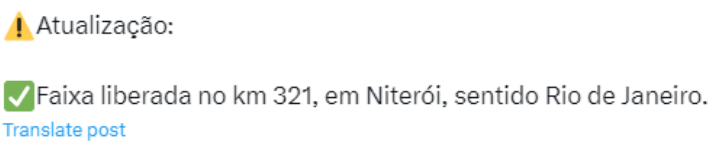


Figura 16: Exemplo de *tweet* com evento de trânsito relevante.

### 5.5 Vetorização

No escopo deste trabalho, um *token* corresponde a uma palavra contida em um *tweet*, ou seja, um elemento individual do texto. Dessa forma, para que os *tokens* obtidos na fase de pré-processamento possam ser utilizados como entrada das redes neurais, esses devem ser transformados em *word embeddings*, tal como explicado no Capítulo 2.

Para gerar as representações vetoriais, foi utilizado um sistema de pré-vetorização, em que um dicionário palavra-vetor é construído a partir da análise de uma vetorização pré-existente, preferivelmente obtida a partir de um *corpus* similar. Assim, no algoritmo de vetorização utilizado, cada *token* é traduzido para o vetor correspondente pré-existente ou para um vetor de zeros, caso não seja encontrado.

Assim como em Teixeira *et al*. (2020), os vetores foram truncados no 30° *token* para garantir uma entrada de tamanho fixo, um requerimento da biblioteca Keras

utilizada para implementar os modelos. O tamanho de 30 foi escolhido a partir da contagem do número de palavras contidas nos *tweets* de todo o *dataset*, em que foi possível observar que esse valor proporciona uma cobertura de cerca de 98% dos microtextos, como evidenciado nas Figuras 17 e 18, que apresentam o número de ocorrências de quantidade de *tokens* por *tweet* e o acumulado dessas ocorrências, respectivamente. Desse modo, o modelo recebe como entrada uma matriz cujo tamanho corresponde a 30 vezes o número de dimensões do vetorizador utilizado.

No caso do BERTimbau, conforme definido pelo modelo, o tamanho máximo aceito de uma sequência de entrada é de 512 *tokens*. No entanto, a fim de se ter uma comparação justa nos experimentos realizados, as suas entradas também foram truncadas no 30° *token*.

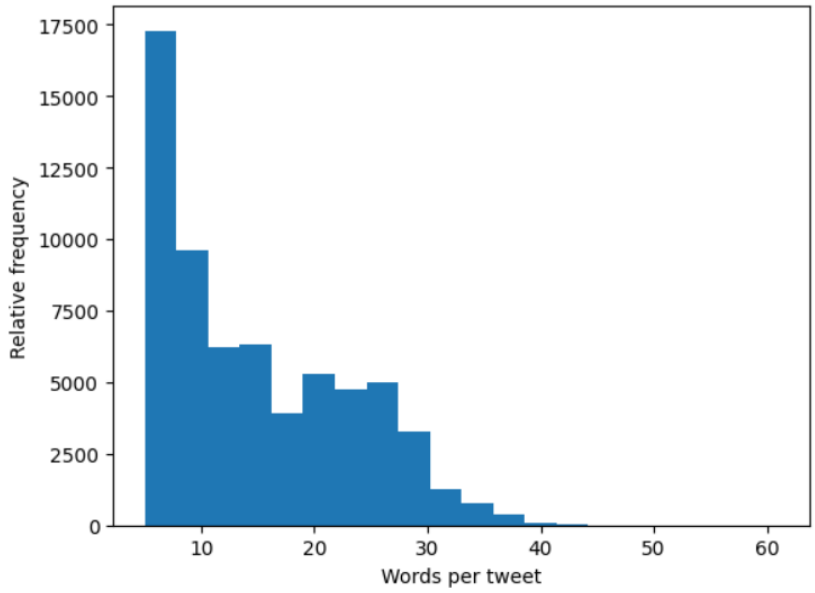


Figura 17: Histograma da frequência relativa de quantidade de palavras por *tweet*.

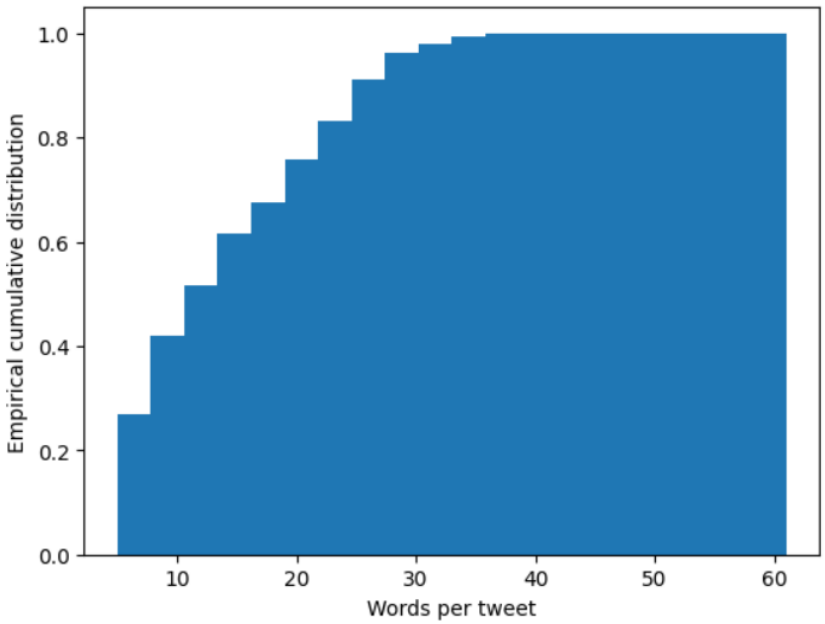


Figura 18: Distribuição cumulativa das ocorrências de quantidade de palavras por *tweet*.

No caso deste trabalho, foram utilizados os vetorizadores pré-treinados *Word2Vec*, GloVe e FastText adaptados à língua portuguesa produzidos por Hartmann *et al*. (2017). Os *word embeddings* foram geradas a partir de 17 *corpus* com fontes variadas do português brasileiro e do português europeu e foram disponibilizadas pelo Núcleo Interinstitucional de Linguística Computacional da Universidade de São Paulo (NILC-USP)[13] e apresentam vetorizações em 50, 100, 300, 600 e 1000 dimensões, de modo que todas as variações foram empregadas com o propósito de comparação dos resultados produzidos.

### 5.6 Experimentos

No caso das arquiteturas tradicionais de Aprendizagem Profunda, os experimentos foram executados dez vezes para cada combinação de vetorizador (*Word2Vec*, GloVe e FastText), dimensão (50, 100, 300, 600 e 1000) e modelo (CNN, LSTM e CNN+LSTM), e a média dos resultados foi calculada para cada uma das 45 combinações possíveis.

Os experimentos realizados com o BERTimbau também foram executados dez vezes, mas não houve variação de vetorizadores (e, portanto, tampouco de dimensões), dado que os modelos *Transformers* constroem do zero as próprias representações

[13] http://nilc.icmc.usp.br/embeddings

vetoriais dos *tokens* de entrada. No entanto, é necessário utilizar um tokenizador para transformar as sequências de entrada em um formato que o modelo seja capaz de processar. Diferentemente das arquiteturas tradicionais, os *Transformers* esperam *tokens* específicos que possuem o papel de definir a estrutura de uma sequência e facilitar seu entendimento. Devido ao BERTimbau ser um modelo pré-treinado, o tokenizador escolhido deve ser o mesmo que foi utilizado durante o pré-treinamento, de forma a garantir a consistência entre o resultado da tokenização e a tokenização esperada pelo modelo.

Na avaliação de desempenho das redes propostas, foram adotadas as métricas descritas na Seção 2.4: acurácia, precisão, *recall* e F1-Score. Além disso, foram analisados também os tempos de vetorização (exceto para o BERTimbau), de treinamento e de inferência, que serão detalhados, assim como os resultados das métricas tradicionais, nas subseções a seguir.

Os resultados obtidos foram unificados nas Tabelas 1 a 13 a seguir, que apresentam os valores mínimo, máximo e médio de cada métrica adotada. Esses valores também são exibidos nos gráficos deste capítulo, que evidenciam os resultados dos modelos tradicionais de Aprendizagem Profunda apresentando a variação da métrica analisada em função do número de dimensões do vetorizador. As curvas dos gráficos representam a média dos resultados e correspondem ao vetorizador ou ao classificador utilizado, enquanto a variância é exibida através de barras de erro.

### 5.6.1 Tempo de Vetorização

No escopo deste trabalho, o tempo de vetorização é definido como o tempo decorrido entre a inicialização da classe de vetorização e a finalização da função vetorizadora responsável por gerar os *word embeddings* de cada *token* da entrada. A Tabela 1 apresenta os tempos de vetorização em segundos para cada combinação de vetorizador, dimensão e classificador. Por sua vez, os gráficos introduzidos nas Figuras 19 a 21 mostram os comparativos entre esses tempos, em que é possível observar um crescimento linear do tempo de vetorização em relação ao número de dimensões. Nessas figuras, é possível observar que o *embedding* GloVe é aquele que possui a maior variação no tempo de vetorização tomado, conforme evidenciado pela barra de erro exibida.

Tabela 1: Tempos de vetorização em segundos para cada combinação de vetorizador, dimensão e classificador, considerando somente as arquiteturas tradicionais.

| Vetorizador | Dimensões | Classificador | Mínimo (s) | Média (s) | Máximo (s) |
|---|---|---|---|---|---|
| Word2Vec | 50 | cnn | 41.43 | 50.01 | 75.76 |
| Word2Vec | 100 | cnn | 79.47 | 90.83 | 141.81 |
| Word2Vec | 300 | cnn | 211.67 | 226.57 | 269.22 |
| Word2Vec | 600 | cnn | 401.73 | 410.15 | 442.06 |
| Word2Vec | 1000 | cnn | 669.23 | 691.1 | 730.89 |
| Word2Vec | 50 | lstm | 43.42 | 44.61 | 45.87 |
| Word2Vec | 100 | lstm | 79.38 | 81.46 | 83.98 |
| Word2Vec | 300 | lstm | 211.34 | 219.74 | 222.94 |
| Word2Vec | 600 | lstm | 400.52 | 557.25 | 725.52 |
| Word2Vec | 1000 | lstm | 1113.56 | 1133.69 | 1144.39 |
| Word2Vec | 50 | mixed | 63.93 | 66.29 | 69.48 |
| Word2Vec | 100 | mixed | 123.48 | 127.22 | 133.67 |
| Word2Vec | 300 | mixed | 214.94 | 294.42 | 355.79 |
| Word2Vec | 600 | mixed | 396.15 | 415.15 | 453.55 |
| Word2Vec | 1000 | mixed | 687.89 | 706.64 | 821.5 |
| GloVe | 50 | cnn | 49.39 | 81.65 | 357.55 |
| GloVe | 100 | cnn | 78.62 | 161.38 | 781.06 |
| GloVe | 300 | cnn | 266.88 | 274.58 | 288.98 |
| GloVe | 600 | cnn | 398.98 | 515.45 | 745.15 |
| GloVe | 1000 | cnn | 667.5 | 684.23 | 715.37 |
| GloVe | 50 | lstm | 40.7 | 44.02 | 45.15 |
| GloVe | 100 | lstm | 80.4 | 90.15 | 98.46 |
| GloVe | 300 | lstm | 214.26 | 222.73 | 227.79 |
| GloVe | 600 | lstm | 407.29 | 416.47 | 425.85 |
| GloVe | 1000 | lstm | 689.89 | 713.39 | 816.2 |
| GloVe | 50 | mixed | 50.89 | 60.26 | 74.48 |
| GloVe | 100 | mixed | 98.18 | 104.71 | 109.55 |
| GloVe | 300 | mixed | 210.77 | 274.62 | 324.71 |
| GloVe | 600 | mixed | 395.29 | 562.44 | 694.62 |
| GloVe | 1000 | mixed | 685.65 | 930.34 | 1170.38 |
| FastText | 50 | cnn | 54.67 | 56.85 | 59.31 |
| FastText | 100 | cnn | 68.33 | 86.51 | 104.01 |
| FastText | 300 | cnn | 192.02 | 204.52 | 256.85 |
| FastText | 600 | cnn | 377.06 | 380.73 | 383.97 |
| FastText | 1000 | cnn | 626.24 | 633.12 | 639.18 |
| FastText | 50 | lstm | 35.25 | 39.06 | 40.5 |
| FastText | 100 | lstm | 65.56 | 67.62 | 69.89 |
| FastText | 300 | lstm | 180.69 | 194.11 | 303.08 |
| FastText | 600 | lstm | 355.36 | 387.84 | 484.25 |
| FastText | 1000 | lstm | 596.53 | 626.82 | 648.03 |
| FastText | 50 | mixed | 37.44 | 39.19 | 43.51 |
| FastText | 100 | mixed | 68.39 | 69.51 | 70.95 |
| FastText | 300 | mixed | 191.85 | 201.41 | 208.91 |
| FastText | 600 | mixed | 389.3 | 435.85 | 484.71 |
| FastText | 1000 | mixed | 577.23 | 641.0 | 815.95 |

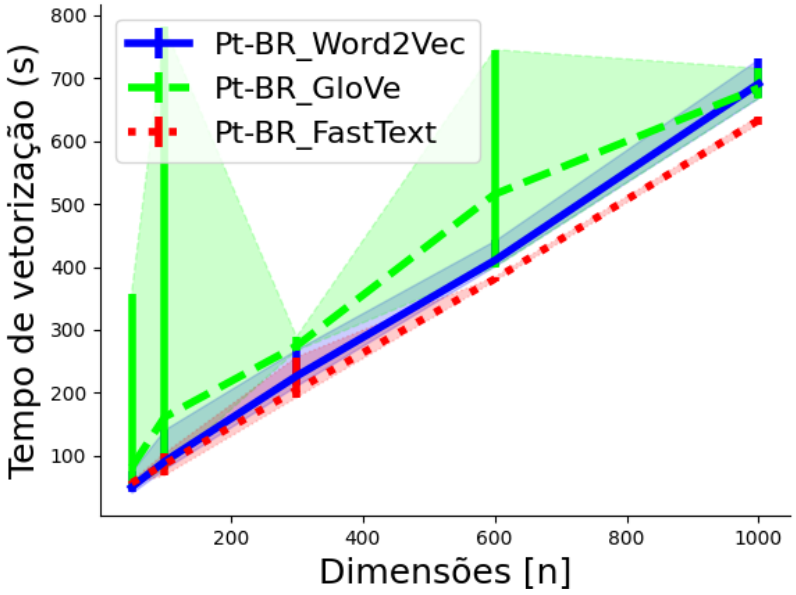


Figura 19: Comparação de tempo de vetorização dos *embeddings Word2Vec*, GloVe e FastText utilizando o modelo CNN.

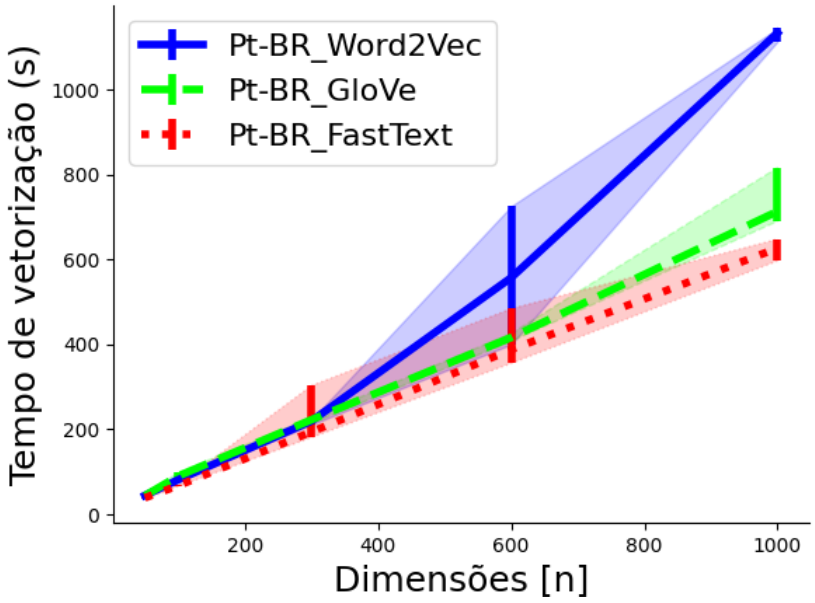


Figura 20: Comparação de tempo de vetorização dos *embeddings Word2Vec*, GloVe e FastText utilizando o modelo LSTM.

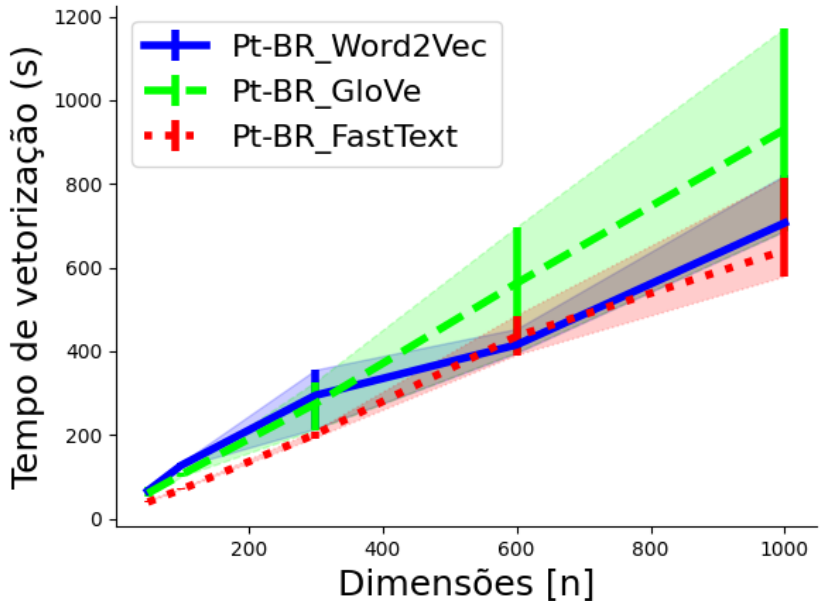


Figura 21: Comparação de tempo de vetorização dos *embeddings Word2Vec*, GloVe e FastText utilizando o modelo CNN+LSTM.

### 5.6.2 Tempo de Treinamento

O tempo de treinamento corresponde ao tempo passado entre a entrada e a saída da função de treinamento de cada modelo. A Tabela 2 apresenta os resultados obtidos nos experimentos com arquiteturas tradicionais, considerando cada combinação de vetorizador, dimensão e classificador variada. Analogamente, a Tabela 3 apresenta o tempo de treinamento obtido para os experimentos com o modelo BERTimbau. Os gráficos exibidos pelas Figuras 22 a 24 exibem o tempo de treinamento de cada *embedding* em relação ao seu número de dimensões, considerando cada arquitetura tradicional. Já as Figuras de 25 a 27 mostram o tempo de treinamento obtido por cada arquitetura clássica em relação ao número de dimensões do *embedding*, considerando cada estratégia de vetorização.

É possível observar que os tempos de treinamento do modelo CNN são mais estáveis, apresentando poucas variações em comparação com os outros modelos. Além disso, é possível notar que o modelo CNN é treinado mais rapidamente, exibindo tempos muito menores do que as outras arquiteturas propostas. Essa diferença é especialmente perceptível nos gráficos de comparação dos modelos por vetorizador (Figuras de 25 a 27). A rede CNN + LSTM apresenta tempos de treinamento maiores, o

que já era esperado considerando a maior complexidade da sua arquitetura, que é composta pela combinação de duas redes diferentes.

Em relação ao BERTimbau, os tempos de treinamento obtidos foram muito maiores do que os tempos apresentados pelas arquiteturas tradicionais. Enquanto o maior tempo de treinamento do BERTimbau foi de 15.387,73 segundos, o maior tempo apresentado dentre as arquiteturas tradicionais foi de 2.139,03 segundos no experimento com 1000 dimensões, vetorizador GloVe e classificador LSTM. Isso se explica pelo fato da arquitetura *Transformer* ser bem mais custosa computacionalmente, apresentar um número muito maior de parâmetros e pelo caráter quadrático do mecanismo de atenção (AGGARWAL, 2023). No caso da versão *Base* do BERTimbau, utilizada neste trabalho, sua arquitetura é composta por 12 camadas de *Encoder*, 12 *Attention Heads* e 110 milhões de parâmetros.

Tabela 2: Tempos de treinamento em segundos para cada combinação de vetorizador, dimensão e classificador, considerando somente as arquiteturas tradicionais.

| Vetorizador | Dimensões | Classificador | Mínimo (s) | Média (s) | Máximo (s) |
|---|---|---|---|---|---|
| Word2Vec | 50 | cnn | 38.98 | 57.24 | 166.88 |
| Word2Vec | 100 | cnn | 42.56 | 72.04 | 179.41 |
| Word2Vec | 300 | cnn | 127.35 | 133.97 | 141.17 |
| Word2Vec | 600 | cnn | 212.36 | 217.89 | 223.31 |
| Word2Vec | 1000 | cnn | 329.27 | 331.69 | 333.64 |
| Word2Vec | 50 | lstm | 139.5 | 144.36 | 147.55 |
| Word2Vec | 100 | lstm | 183.48 | 204.8 | 224.24 |
| Word2Vec | 300 | lstm | 400.79 | 425.76 | 445.2 |
| Word2Vec | 600 | lstm | 744.85 | 1124.9 | 1743.44 |
| Word2Vec | 1000 | lstm | 779.27 | 938.42 | 1769.77 |
| Word2Vec | 50 | mixed | 936.63 | 1275.69 | 1756.5 |
| Word2Vec | 100 | mixed | 1241.59 | 1655.51 | 1980.89 |
| Word2Vec | 300 | mixed | 1145.64 | 1342.44 | 1585.46 |
| Word2Vec | 600 | mixed | 1241.12 | 1409.6 | 1673.99 |
| Word2Vec | 1000 | mixed | 1466.67 | 1713.39 | 1919.2 |
| GloVe | 50 | cnn | 157.79 | 174.39 | 185.03 |
| GloVe | 100 | cnn | 134.94 | 195.88 | 222.71 |
| GloVe | 300 | cnn | 192.11 | 217.52 | 234.72 |
| GloVe | 600 | cnn | 305.45 | 348.09 | 654.38 |
| GloVe | 1000 | cnn | 411.72 | 427.48 | 440.7 |
| GloVe | 50 | lstm | 306.52 | 368.8 | 395.71 |
| GloVe | 100 | lstm | 165.39 | 185.49 | 205.68 |
| GloVe | 300 | lstm | 267.93 | 417.41 | 650.27 |
| GloVe | 600 | lstm | 667.51 | 1353.96 | 1663.28 |
| GloVe | 1000 | lstm | 888.32 | 1392.86 | 2139.03 |
| GloVe | 50 | mixed | 1118.38 | 1235.45 | 1561.26 |
| GloVe | 100 | mixed | 1127.91 | 1438.1 | 1815.36 |
| GloVe | 300 | mixed | 1357.19 | 1617.09 | 2099.25 |
| GloVe | 600 | mixed | 1051.98 | 1410.43 | 2018.6 |
| GloVe | 1000 | mixed | 1157.0 | 1462.98 | 2078.58 |
| FastText | 50 | cnn | 198.76 | 245.91 | 277.4 |
| FastText | 100 | cnn | 101.87 | 122.05 | 253.47 |
| FastText | 300 | cnn | 178.66 | 185.5 | 199.61 |
| FastText | 600 | cnn | 280.63 | 287.49 | 296.25 |
| FastText | 1000 | cnn | 448.31 | 466.59 | 516.4 |
| FastText | 50 | lstm | 192.34 | 204.28 | 226.7 |
| FastText | 100 | lstm | 225.27 | 251.36 | 297.63 |
| FastText | 300 | lstm | 366.96 | 459.06 | 543.86 |
| FastText | 600 | lstm | 505.25 | 618.0 | 960.02 |
| FastText | 1000 | lstm | 728.61 | 1138.97 | 1455.53 |
| FastText | 50 | mixed | 870.89 | 949.38 | 1035.26 |
| FastText | 100 | mixed | 760.17 | 890.66 | 1063.21 |
| FastText | 300 | mixed | 901.23 | 983.78 | 1126.47 |
| FastText | 600 | mixed | 1051.36 | 1184.27 | 1399.49 |
| FastText | 1000 | mixed | 1084.67 | 1288.5 | 2055.82 |

Tabela 3: Tempo de treinamento do BERTimbau.

| Classificador | Mínimo (s) | Média (s) | Máximo (s) |
|---|---|---|---|
| BERTimbau | 11605.8 | 13176.79 | 15387.73 |

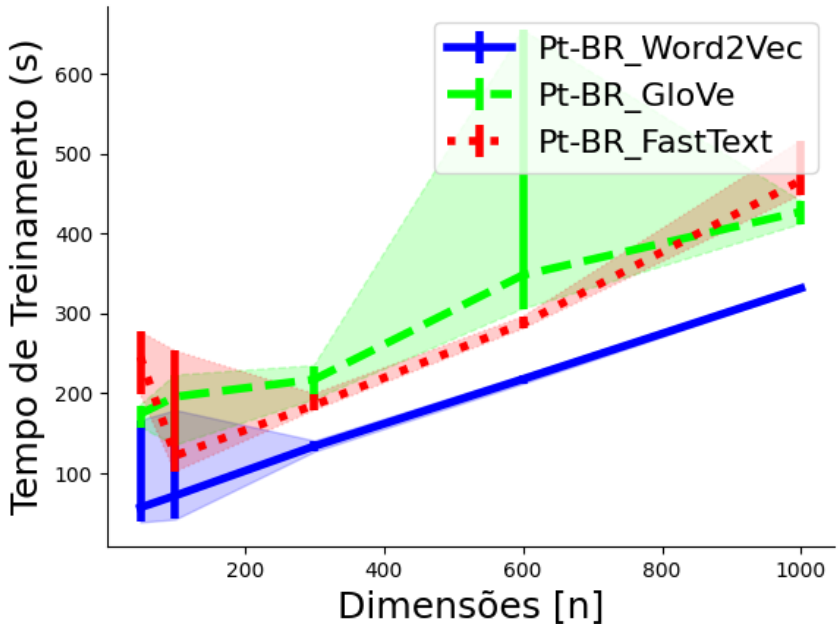


Figura 22: Comparação do tempo de treinamento dos *embeddings Word2Vec*, GloVe e FastText utilizando o modelo CNN.

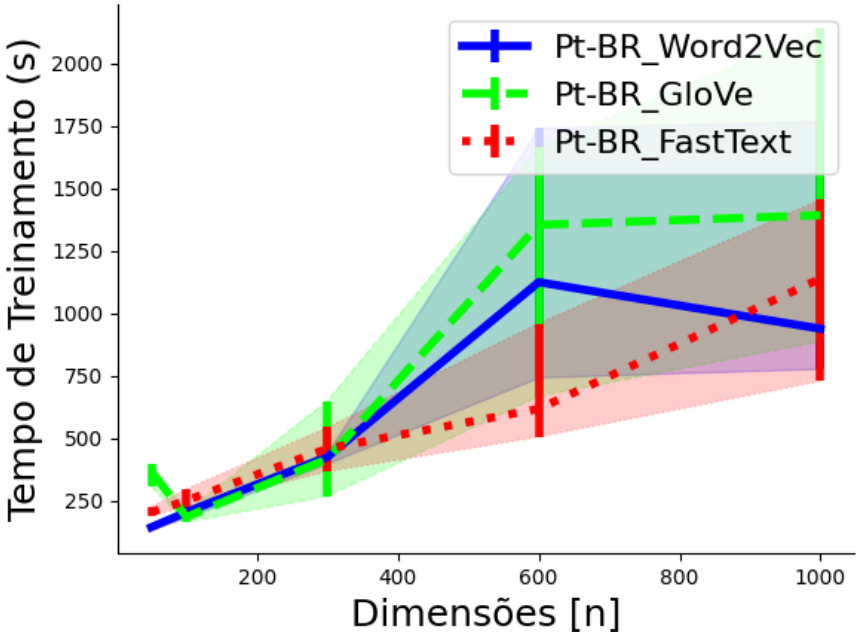


Figura 23: Comparação do tempo de treinamento dos *embeddings Word2Vec*, GloVe e FastText utilizando o modelo LSTM.

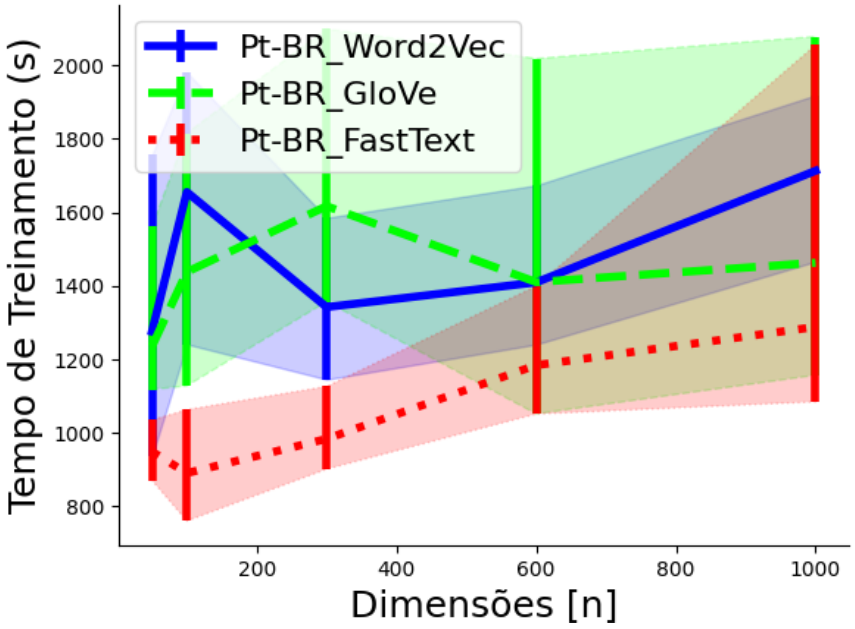


Figura 24: Comparação do tempo de treinamento dos *embeddings Word2Vec*, GloVe e FastText utilizando o modelo CNN+LSTM.

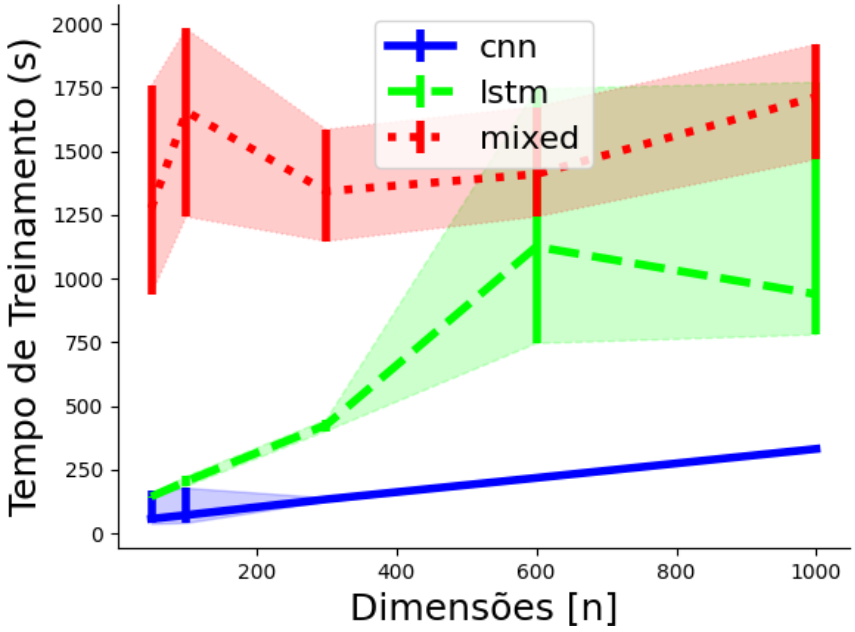


Figura 25: Comparação do tempo de treinamento dos modelos CNN, LSTM e CNN+LSTM utilizando o *embedding Word2Vec*.

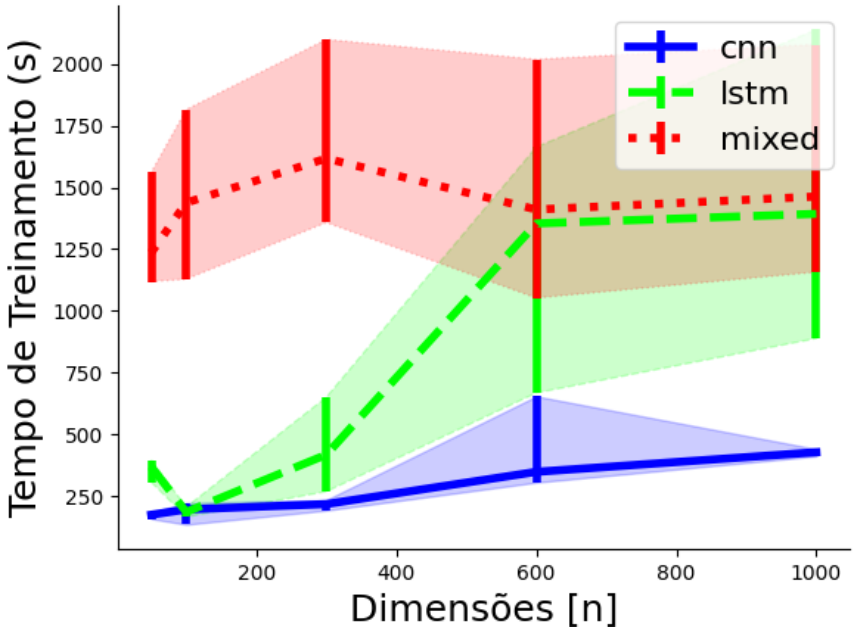


Figura 26: Comparação do tempo de treinamento dos modelos CNN, LSTM e CNN+LSTM utilizando o *embedding* GloVe.

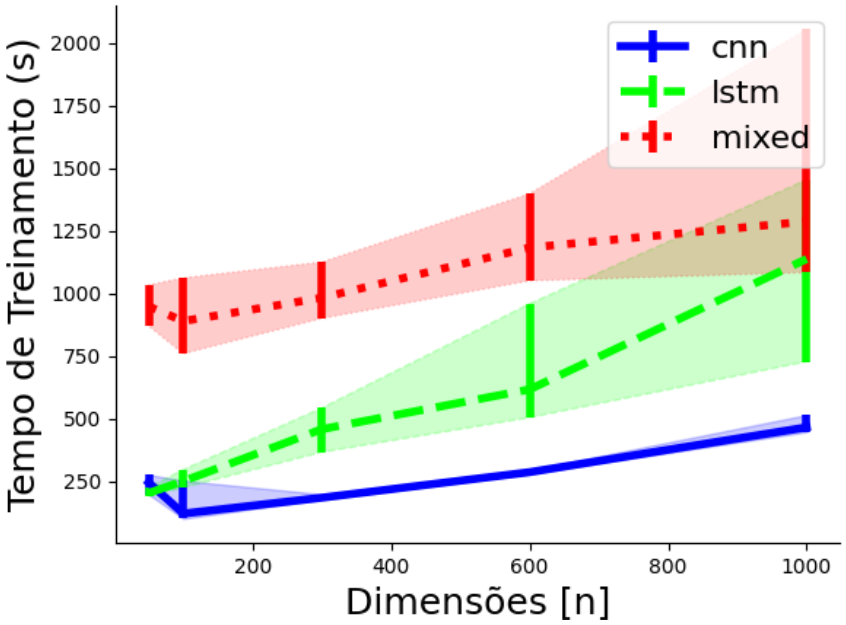


Figura 27: Comparação do tempo de treinamento dos modelos CNN, LSTM e CNN+LSTM utilizando o *embedding* FastText.

### 5.6.3 Tempo de Inferência

O tempo de inferência contabiliza o tempo necessário para que o modelo faça uma predição ao receber como entrada um exemplo que não tinha sido visto durante o

treinamento. Desse modo, o cálculo do tempo de inferência é relevante porque, enquanto o tempo de treinamento costuma ser bem maior, o treinamento é executado poucas vezes. As inferências, no entanto, são executadas diversas vezes uma vez que o modelo tenha sido treinado e disponibilizado para uso.

Desse modo, para calcular o tempo médio de inferência dos modelos, foi criado um conjunto de dados contendo 100 *tweets* não vistos durante a fase de treinamento. Em seguida, foram realizadas 100 rodadas de inferência para cada combinação de arquitetura e contabilizado o tempo associado a cada rodada, de modo que cada *tweet* foi utilizado como entrada uma única vez. Por fim, foi calculada a média aritmética dos tempos de inferência obtidos para cada combinação.

A Tabela 4 apresenta o tempo médio de inferência obtido nos experimentos com arquiteturas tradicionais, considerando cada combinação de vetorizador, dimensão e classificador. Já a Tabela 5 mostra o tempo médio de inferência para o modelo BERTimbau. Tais resultados mostram que o número de dimensões dos vetorizadores afeta o tempo de inferência, o que já era esperado devido à necessidade de se converter as entradas textuais em vetores numéricos a cada predição, isto é, realizar a vetorização. O maior tempo de inferência observado foi de 0,0555 segundos obtido pela combinação do classificador CNN+LSTM com o vetorizador FastText de 1000 dimensões, enquanto o menor tempo foi obtido pela combinação do classificador LSTM com o vetorizador *Word2Vec* de 50 dimensões contabilizando 0,0357 segundos.

O BERTimbau, por sua vez, apresentou um tempo de inferência menor, de 0,0339 segundos, não sendo uma diferença considerável em comparação aos tempos apresentados pelas arquiteturas tradicionais.

Tabela 4: Tempo de inferência médio em segundos para cada combinação de vetorizador, dimensão e classificador, considerando somente as arquiteturas tradicionais.

| Vetorizador | Dimensões | Classificador | Tempo de Inferência (s) |
|---|---|---|---|
| Word2Vec | 50 | cnn | 0.0433 |
| Word2Vec | 100 | cnn | 0.0412 |
| Word2Vec | 300 | cnn | 0.0418 |
| Word2Vec | 600 | cnn | 0.0358 |
| Word2Vec | 1000 | cnn | 0.0361 |
| Word2Vec | 50 | lstm | 0.0357 |
| Word2Vec | 100 | lstm | 0.0359 |
| Word2Vec | 300 | lstm | 0.0363 |
| Word2Vec | 600 | lstm | 0.0368 |
| Word2Vec | 1000 | lstm | 0.04 |
| Word2Vec | 50 | mixed | 0.039 |
| Word2Vec | 100 | mixed | 0.0366 |
| Word2Vec | 300 | mixed | 0.038 |
| Word2Vec | 600 | mixed | 0.037 |
| Word2Vec | 1000 | mixed | 0.0445 |
| GloVe | 50 | cnn | 0.0369 |
| GloVe | 100 | cnn | 0.0478 |
| GloVe | 300 | cnn | 0.0367 |
| GloVe | 600 | cnn | 0.0386 |
| GloVe | 1000 | cnn | 0.0405 |
| GloVe | 50 | lstm | 0.0375 |
| GloVe | 100 | lstm | 0.0371 |
| GloVe | 300 | lstm | 0.0369 |
| GloVe | 600 | lstm | 0.0396 |
| GloVe | 1000 | lstm | 0.0431 |
| GloVe | 50 | mixed | 0.0454 |
| GloVe | 100 | mixed | 0.0372 |
| GloVe | 300 | mixed | 0.0373 |
| GloVe | 600 | mixed | 0.038 |
| GloVe | 1000 | mixed | 0.0434 |
| FastText | 50 | cnn | 0.0408 |
| FastText | 100 | cnn | 0.0396 |
| FastText | 300 | cnn | 0.0472 |
| FastText | 600 | cnn | 0.0451 |
| FastText | 1000 | cnn | 0.049 |
| FastText | 50 | lstm | 0.0467 |
| FastText | 100 | lstm | 0.0483 |
| FastText | 300 | lstm | 0.0452 |
| FastText | 600 | lstm | 0.0399 |
| FastText | 1000 | lstm | 0.0409 |
| FastText | 50 | mixed | 0.0429 |
| FastText | 100 | mixed | 0.0402 |
| FastText | 300 | mixed | 0.0396 |
| FastText | 600 | mixed | 0.0423 |
| FastText | 1000 | mixed | 0.0555 |

Tabela 5: Tempo médio de inferência do BERTimbau.

| Classificador | Tempo de Inferência (s) |
|---|---|
| BERTimbau | 0.0339 |

### 5.6.4 Acurácia

A métrica acurácia, descrita na Seção 2.4, corresponde à taxa de acerto do classificador, isto é, a quantidade de exemplos positivos e negativos que foram classificados corretamente pelo modelo.

A Tabela 6 apresenta os valores de acurácia dos experimentos com arquiteturas tradicionais, tomando-se cada combinação de vetorizador, dimensão e classificador variada. De maneira análoga, a Tabela 7 exibe os valores de acurácia para o modelo BERTimbau. Os gráficos representados nas Figuras 28 a 33 evidenciam uma baixa variação da métrica, que se concentra entre os valores 98,92% e 99,37%. Por esse motivo, é possível observar uma estabilidade nos resultados obtidos, de modo que a variação dimensional e as diferentes combinações entre vetorizadores e classificadores parecem não ter causado grande impacto nas porcentagens de acurácia resultantes. De modo geral, o uso do classificador CNN com o vetorizador FastText parece ter produzido os resultados mais baixos, enquanto o vetorizador GloVe apresentou boas porcentagens com todos os classificadores, em especial com as redes CNN e LSTM.

Quanto ao BERTimbau, a Tabela 7 mostra que os resultados obtidos se encontram em um intervalo de variação ainda menor do que o observado nos experimentos com as arquiteturas tradicionais. O valor mínimo de acurácia foi de 99,24% e o valor máximo alcançado de 99,34% é superado apenas pelas combinações de classificador CNN com vetorizador GloVe de 300 dimensões e classificador LSTM com vetorizador GloVe com 600 dimensões, que apresentaram valores máximos de 99,37% e 99,35%, respectivamente. Além disso, a combinação de classificador LSTM com vetorizador GloVe com 1000 dimensões também apresentou um valor máximo de 99,34%.

Tabela 6: Valores de acurácia para cada combinação de vetorizador, dimensão e classificador, considerando somente as arquiteturas tradicionais.

| Vetorizador | Dimensões | Classificador | Mínimo (%) | Média (%) | Máximo (%) |
|---|---|---|---|---|---|
| Word2Vec | 50 | cnn | 99.05 | 99.09 | 99.14 |
| Word2Vec | 100 | cnn | 99.13 | 99.16 | 99.18 |
| Word2Vec | 300 | cnn | 99.22 | 99.25 | 99.27 |
| Word2Vec | 600 | cnn | 99.23 | 99.27 | 99.29 |
| Word2Vec | 1000 | cnn | 99.24 | 99.26 | 99.28 |
| Word2Vec | 50 | lstm | 98.99 | 99.05 | 99.1 |
| Word2Vec | 100 | lstm | 99.12 | 99.16 | 99.21 |
| Word2Vec | 300 | lstm | 99.19 | 99.23 | 99.27 |
| Word2Vec | 600 | lstm | 99.18 | 99.22 | 99.26 |
| Word2Vec | 1000 | lstm | 99.2 | 99.26 | 99.32 |
| Word2Vec | 50 | mixed | 99.03 | 99.07 | 99.11 |
| Word2Vec | 100 | mixed | 99.1 | 99.2 | 99.25 |
| Word2Vec | 300 | mixed | 99.17 | 99.22 | 99.26 |
| Word2Vec | 600 | mixed | 99.21 | 99.25 | 99.32 |
| Word2Vec | 1000 | mixed | 99.23 | 99.26 | 99.3 |
| GloVe | 50 | cnn | 99.1 | 99.16 | 99.2 |
| GloVe | 100 | cnn | 99.23 | 99.26 | 99.28 |
| GloVe | 300 | cnn | 99.28 | 99.32 | 99.37 |
| GloVe | 600 | cnn | 99.27 | 99.29 | 99.32 |
| GloVe | 1000 | cnn | 99.27 | 99.3 | 99.33 |
| GloVe | 50 | lstm | 99.07 | 99.12 | 99.18 |
| GloVe | 100 | lstm | 99.21 | 99.23 | 99.28 |
| GloVe | 300 | lstm | 99.3 | 99.32 | 99.33 |
| GloVe | 600 | lstm | 99.28 | 99.31 | 99.35 |
| GloVe | 1000 | lstm | 99.27 | 99.3 | 99.34 |
| GloVe | 50 | mixed | 99.11 | 99.17 | 99.23 |
| GloVe | 100 | mixed | 99.18 | 99.24 | 99.28 |
| GloVe | 300 | mixed | 99.23 | 99.28 | 99.32 |
| GloVe | 600 | mixed | 99.22 | 99.26 | 99.32 |
| GloVe | 1000 | mixed | 99.22 | 99.25 | 99.29 |
| FastText | 50 | cnn | 98.92 | 98.97 | 99.03 |
| FastText | 100 | cnn | 98.99 | 99.04 | 99.08 |
| FastText | 300 | cnn | 99.06 | 99.11 | 99.16 |
| FastText | 600 | cnn | 99.06 | 99.12 | 99.15 |
| FastText | 1000 | cnn | 99.03 | 99.09 | 99.14 |
| FastText | 50 | lstm | 99.05 | 99.15 | 99.22 |
| FastText | 100 | lstm | 99.15 | 99.19 | 99.22 |
| FastText | 300 | lstm | 99.17 | 99.2 | 99.27 |
| FastText | 600 | lstm | 99.21 | 99.25 | 99.27 |
| FastText | 1000 | lstm | 99.17 | 99.23 | 99.27 |
| FastText | 50 | mixed | 99.1 | 99.16 | 99.24 |
| FastText | 100 | mixed | 99.09 | 99.17 | 99.22 |
| FastText | 300 | mixed | 99.17 | 99.21 | 99.24 |
| FastText | 600 | mixed | 99.2 | 99.25 | 99.29 |
| FastText | 1000 | mixed | 99.16 | 99.23 | 99.28 |

Tabela 7: Valores de acurácia para o modelo BERTimbau.

| Vetorizador | Mínimo (%) | Média (%) | Máximo (%) |
|---|---|---|---|
| BERTimbau | 99.24 | 99.28 | 99.34 |

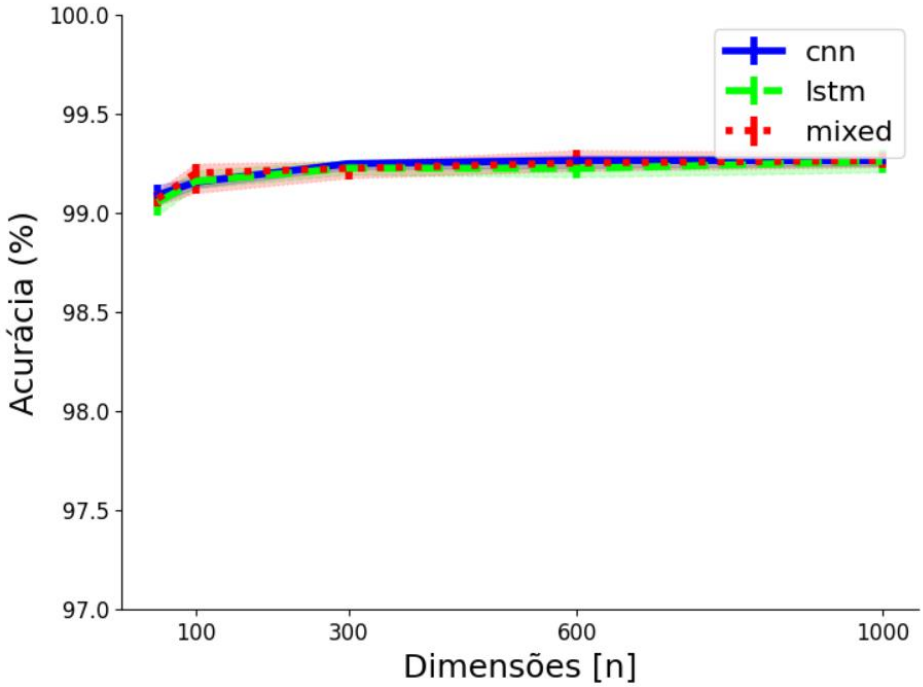


Figura 28: Porcentagem de acurácia dos modelos CNN, LSTM e CNN+LSTM utilizando o *embedding Word2Vec*.

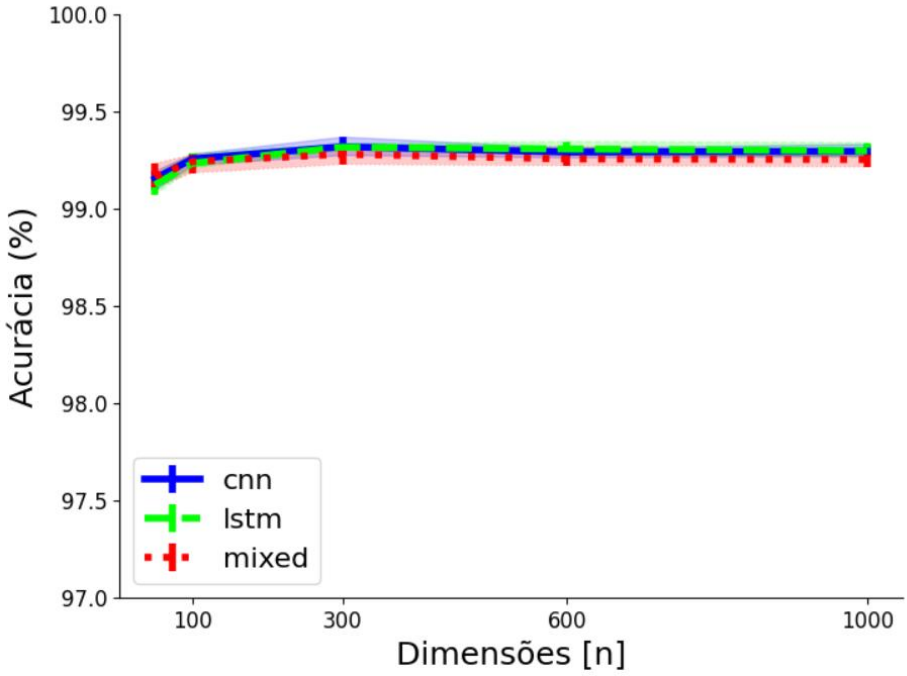


Figura 29: Porcentagem de acurácia dos modelos CNN, LSTM e CNN+LSTM utilizando o *embedding* GloVe.

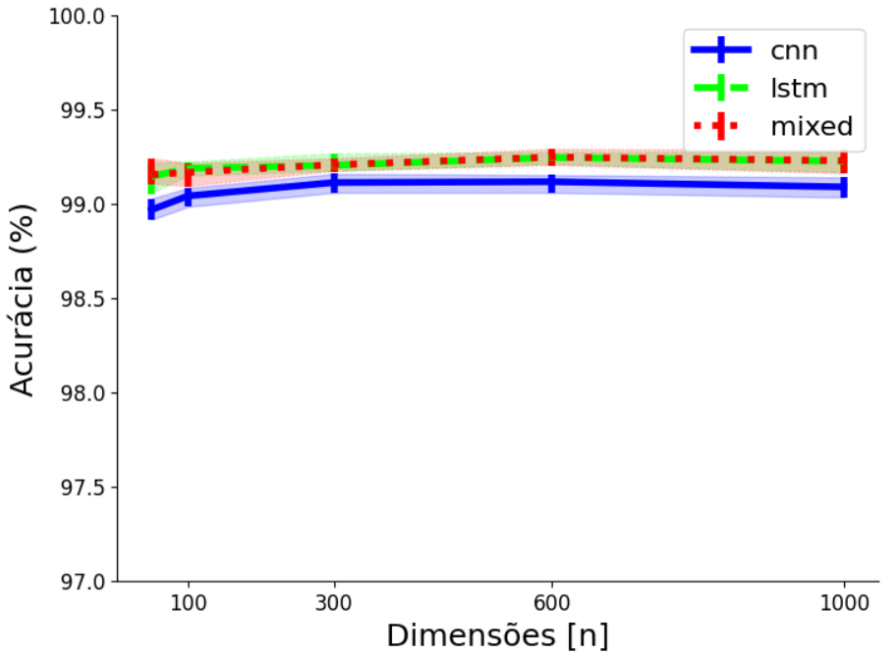


Figura 30: Porcentagem de acurácia dos modelos CNN, LSTM e CNN+LSTM utilizando o *embedding* FastText.

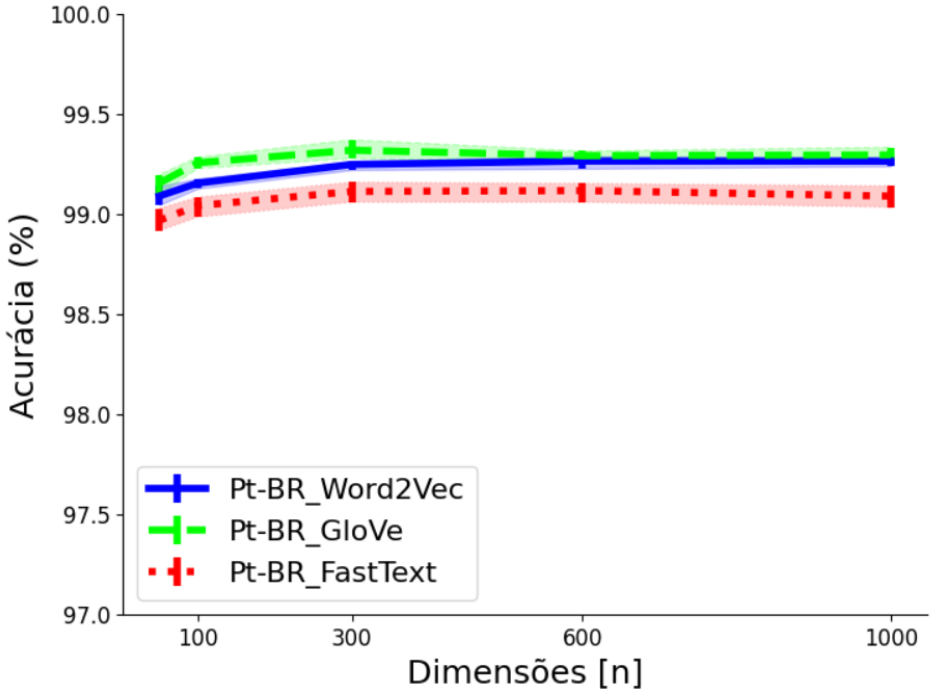


Figura 31: Porcentagem de acurácia dos *embeddings Word2Vec*, GloVe e FastText utilizando o modelo CNN.

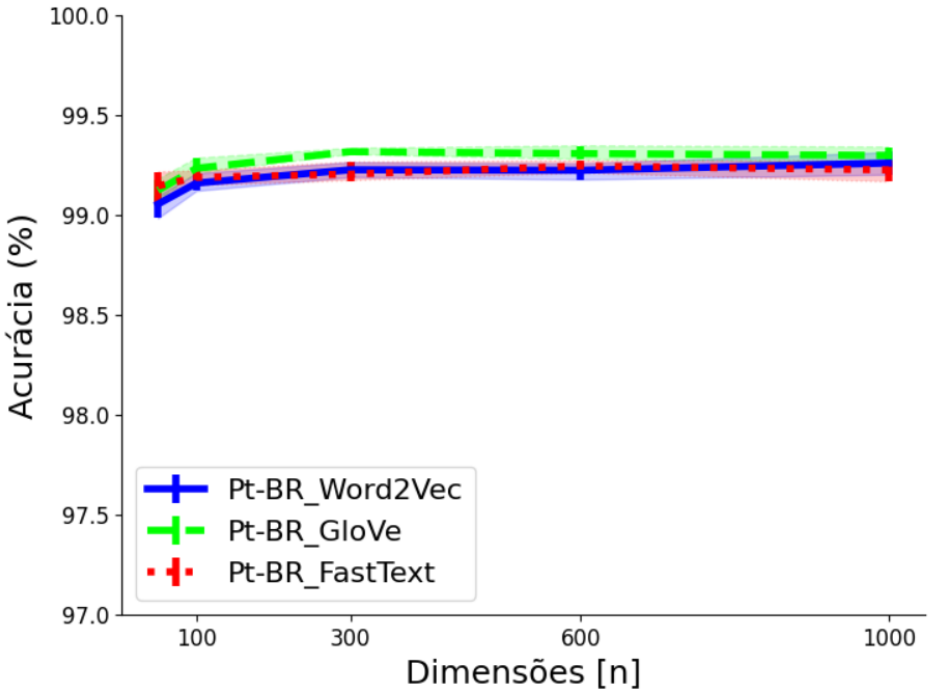


Figura 32: Porcentagem de acurácia dos *embeddings Word2Vec*, GloVe e FastText utilizando o modelo LSTM.

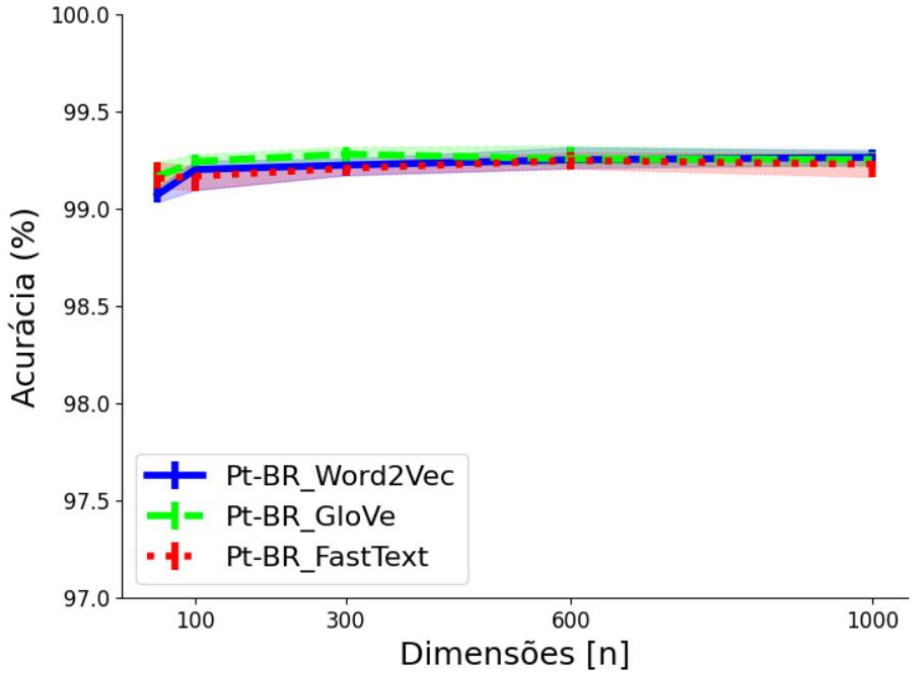


Figura 33: Porcentagem de acurácia dos *embeddings Word2Vec*, GloVe e FastText utilizando o modelo CNN+LSTM.

### 5.6.5 Precisão

A precisão corresponde à proporção do número de exemplos classificados como positivos, e para os quais o modelo acertou, em relação a todos os resultados

classificados como positivos, ou seja, considerando tanto os verdadeiros positivos quanto os falsos positivos. Assim como a acurácia, sua fórmula é descrita na Seção 2.4.

A Tabela 8 exibe os valores de precisão dos experimentos com arquiteturas tradicionais, tomando-se cada combinação de vetorizador, dimensão e classificador variada. Os gráficos descritos nas Figuras 34 a 39 mostram uma variação pequena entre o valor mínimo de precisão de 98,69% e o valor máximo de 99,61%, notando-se novamente uma estabilidade nos resultados. No entanto, é possível verificar a presença de valores maiores do que os valores obtidos para a métrica de acurácia.

Assim como visto nos resultados de acurácia, a variação de dimensionalidade e as variadas combinações entre classificadores e vetorizadores parecem não interferir de forma significativa nos valores de precisão. O vetorizador GloVe apresenta resultados um pouco maiores se comparado aos outros vetorizadores, especialmente em relação ao *Word2Vec*. Além disso, novamente o vetorizador FastText apresenta resultados mais baixos quando combinado ao modelo CNN.

No que tange aos resultados do BERTimbau, a Tabela 9 também demonstra uma variação pequena entre o valor mínimo de 99,3% e o valor máximo de 99,44%, evidenciando um valor menor do que o obtido para a acurácia. Além disso, apesar dos modelos tradicionais apresentarem resultados com valores mínimos inferiores ao valor apresentado pelo BERTimbau, o valor máximo alcançado é superado por diversas das combinações de arquiteturas tradicionais. Tal fato vai de encontro à intuição que se tinha de que o desempenho do modelo BERTimbau seria superior ao dos modelos baseados nas arquiteturas tradicionais.

Tabela 8: Valores de precisão para cada combinação de vetorizador, dimensão e classificador, considerando somente as arquiteturas tradicionais.

| Vetorizador | Dimensões | Classificador | Mínimo (%) | Média (%) | Máximo (%) |
|---|---|---|---|---|---|
| Word2Vec | 50 | cnn | 99.06 | 99.17 | 99.34 |
| Word2Vec | 100 | cnn | 98.83 | 99.1 | 99.25 |
| Word2Vec | 300 | cnn | 99.17 | 99.25 | 99.33 |
| Word2Vec | 600 | cnn | 99.12 | 99.28 | 99.42 |
| Word2Vec | 1000 | cnn | 99.16 | 99.26 | 99.45 |
| Word2Vec | 50 | lstm | 98.69 | 98.92 | 99.17 |
| Word2Vec | 100 | lstm | 98.92 | 99.17 | 99.38 |
| Word2Vec | 300 | lstm | 99.04 | 99.24 | 99.3 |
| Word2Vec | 600 | lstm | 99.01 | 99.27 | 99.41 |
| Word2Vec | 1000 | lstm | 99.12 | 99.26 | 99.4 |
| Word2Vec | 50 | mixed | 98.83 | 98.97 | 99.26 |
| Word2Vec | 100 | mixed | 99.03 | 99.26 | 99.41 |
| Word2Vec | 300 | mixed | 99.12 | 99.26 | 99.49 |
| Word2Vec | 600 | mixed | 99.16 | 99.29 | 99.49 |
| Word2Vec | 1000 | mixed | 98.99 | 99.27 | 99.41 |
| GloVe | 50 | cnn | 99.05 | 99.15 | 99.28 |
| GloVe | 100 | cnn | 99.24 | 99.32 | 99.45 |
| GloVe | 300 | cnn | 99.19 | 99.32 | 99.42 |
| GloVe | 600 | cnn | 99.2 | 99.39 | 99.58 |
| GloVe | 1000 | cnn | 99.27 | 99.34 | 99.52 |
| GloVe | 50 | lstm | 98.92 | 99.13 | 99.41 |
| GloVe | 100 | lstm | 99.12 | 99.23 | 99.4 |
| GloVe | 300 | lstm | 99.21 | 99.35 | 99.41 |
| GloVe | 600 | lstm | 99.2 | 99.31 | 99.52 |
| GloVe | 1000 | lstm | 99.19 | 99.31 | 99.48 |
| GloVe | 50 | mixed | 98.92 | 99.12 | 99.24 |
| GloVe | 100 | mixed | 99.03 | 99.23 | 99.42 |
| GloVe | 300 | mixed | 99.05 | 99.25 | 99.41 |
| GloVe | 600 | mixed | 99.15 | 99.32 | 99.61 |
| GloVe | 1000 | mixed | 99.08 | 99.25 | 99.41 |
| FastText | 50 | cnn | 98.8 | 99.03 | 99.23 |
| FastText | 100 | cnn | 98.95 | 99.18 | 99.38 |
| FastText | 300 | cnn | 98.93 | 99.17 | 99.45 |
| FastText | 600 | cnn | 99.09 | 99.19 | 99.38 |
| FastText | 1000 | cnn | 98.95 | 99.12 | 99.38 |
| FastText | 50 | lstm | 98.84 | 99.15 | 99.32 |
| FastText | 100 | lstm | 99.0 | 99.19 | 99.41 |
| FastText | 300 | lstm | 99.01 | 99.27 | 99.44 |
| FastText | 600 | lstm | 99.11 | 99.31 | 99.42 |
| FastText | 1000 | lstm | 99.23 | 99.36 | 99.54 |
| FastText | 50 | mixed | 98.96 | 99.12 | 99.24 |
| FastText | 100 | mixed | 98.87 | 99.1 | 99.37 |
| FastText | 300 | mixed | 99.11 | 99.28 | 99.38 |
| FastText | 600 | mixed | 99.19 | 99.33 | 99.56 |
| FastText | 1000 | mixed | 99.07 | 99.24 | 99.46 |

Tabela 9: Valores de precisão para o modelo BERTimbau.

| Vetorizador | Mínimo (%) | Média (%) | Máximo (%) |
|---|---|---|---|
| BERTimbau | 99.3 | 99.36 | 99.44 |

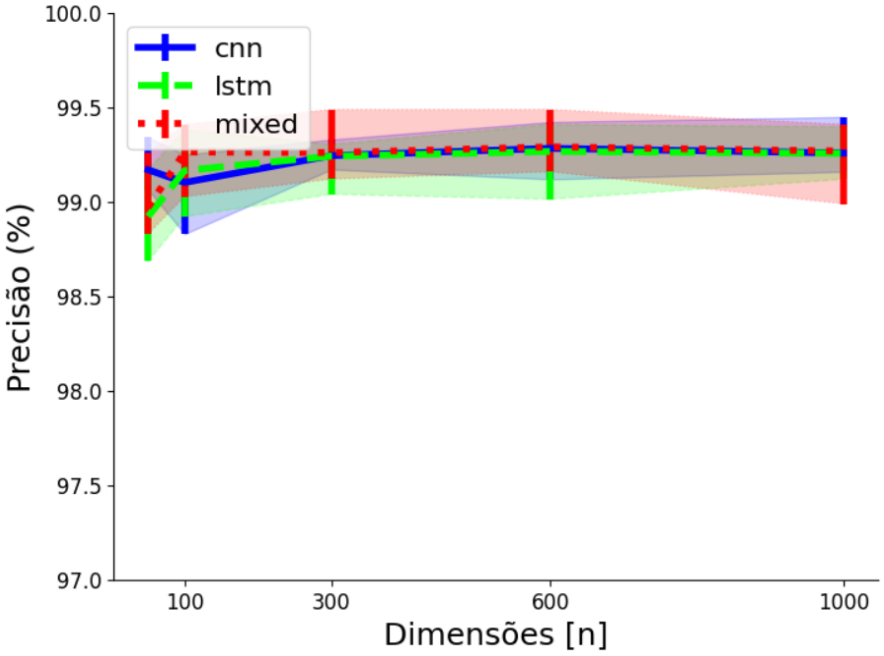


Figura 34: Porcentagem de precisão dos modelos CNN, LSTM e CNN+LSTM utilizando o *embedding Word2Vec*.

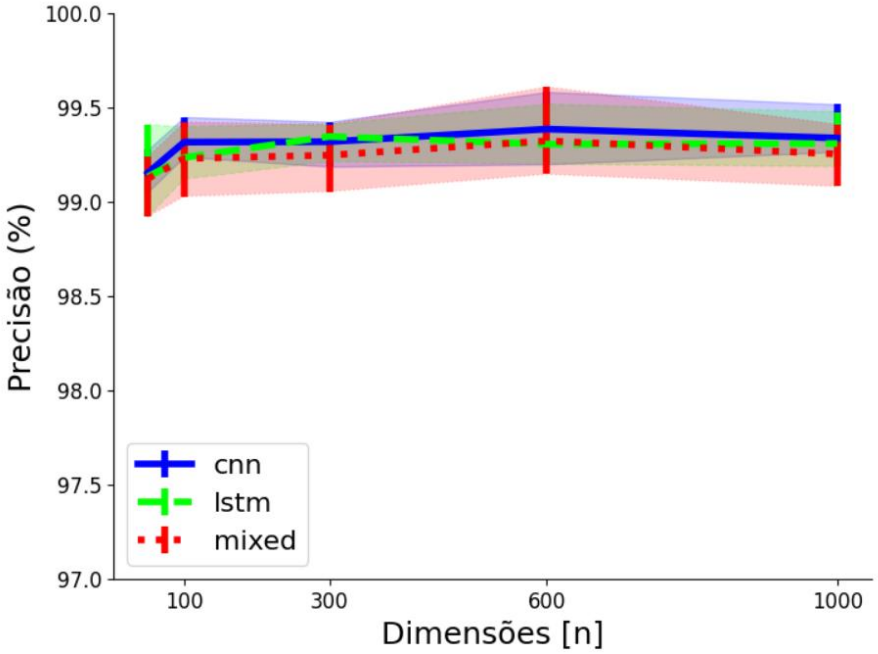


Figura 35: Porcentagem de precisão dos modelos CNN, LSTM e CNN+LSTM utilizando o *embedding* GloVe.

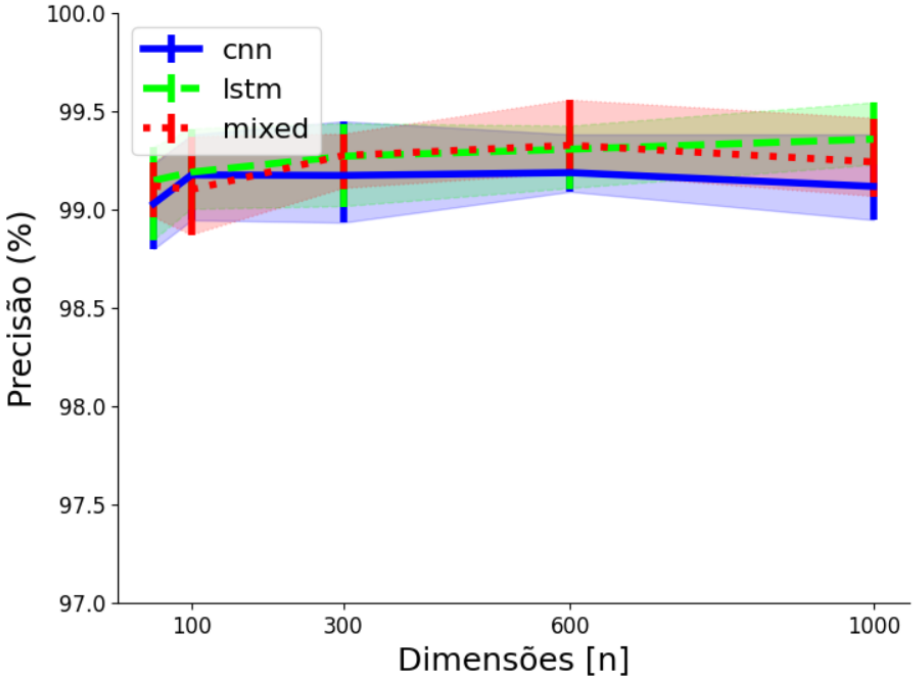


Figura 36: Porcentagem de precisão dos modelos CNN, LSTM e CNN+LSTM utilizando o *embedding* FastText.

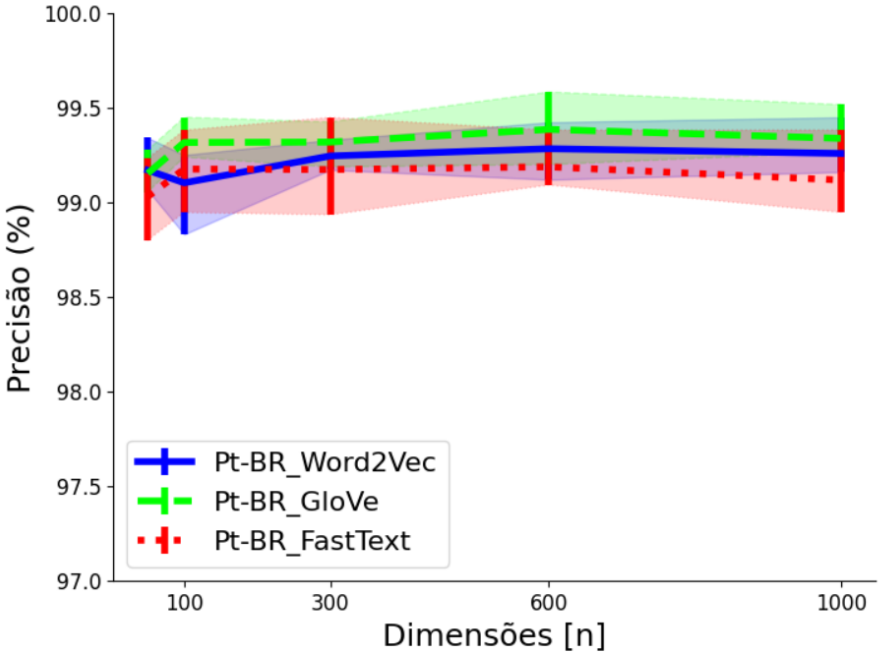


Figura 37: Porcentagem de precisão dos *embeddings Word2Vec*, GloVe e FastText utilizando o modelo CNN.

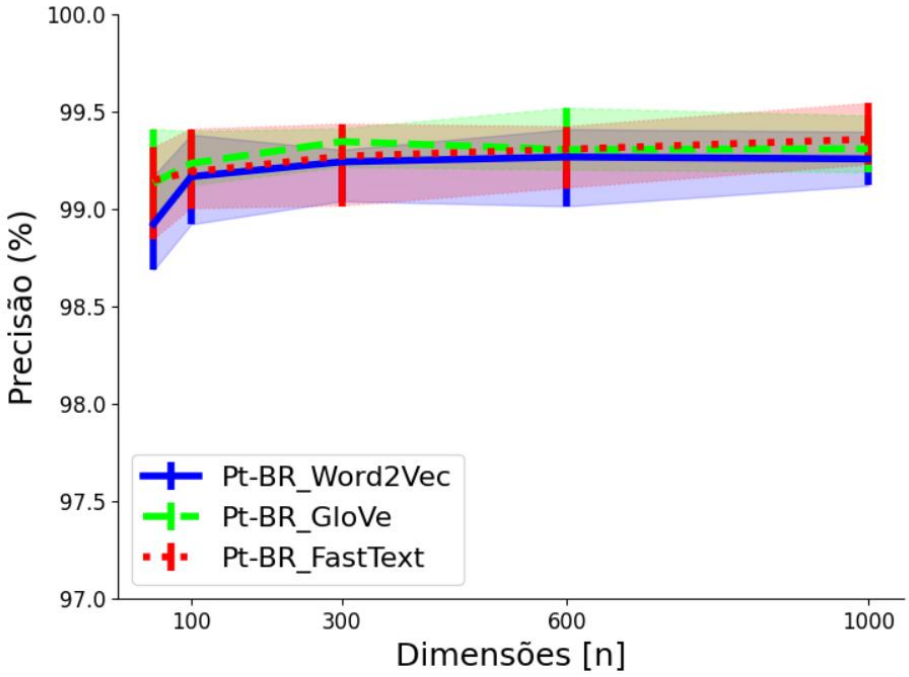


Figura 38: Porcentagem de precisão dos *embeddings Word2Vec*, GloVe e FastText utilizando o modelo LSTM.

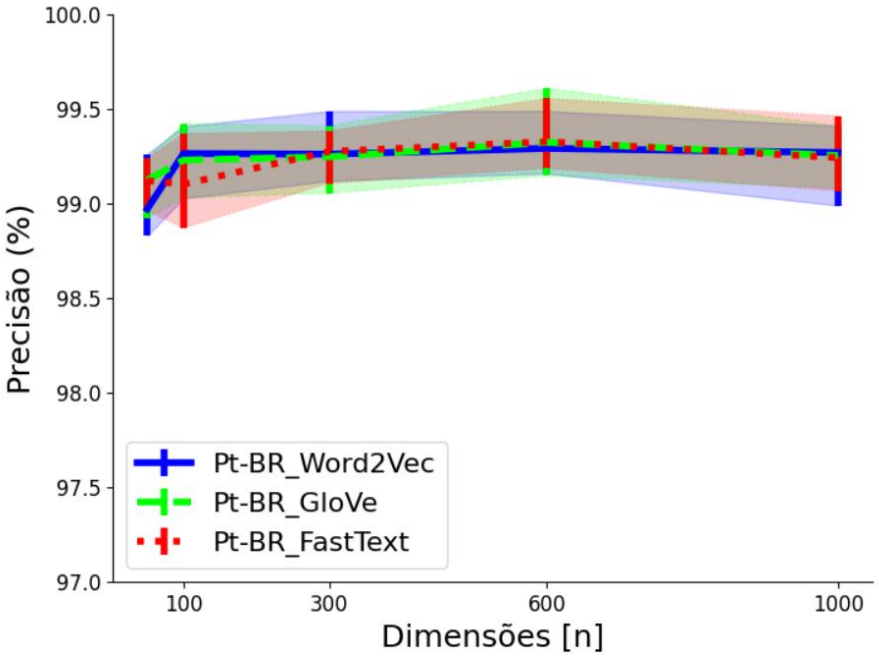


Figura 39: Porcentagem de precisão dos *embeddings Word2Vec*, GloVe e FastText utilizando o modelo CNN+LSTM.

### 5.6.6 *Recall*

A métrica *recall* calcula a proporção de verdadeiros positivos identificada corretamente. Isso significa que é calculado a quantidade de verdadeiros positivos em relação à

quantidade de verdadeiros positivos somados aos falsos negativos. Assim como as outras métricas mencionadas previamente, sua fórmula é apresentada na Seção 2.4.

A Tabela 10 apresenta os valores de *recall* dos experimentos com arquiteturas tradicionais, considerando-se a combinação de vetorizador, dimensão e classificador variada. Por seu turno, os gráficos representados nas Figuras 40 a 45 indicam uma variação da porcentagem de *recall* entre 98,48% e 99,41%, exibindo resultados menores do que aqueles obtidos para as outras métricas. Além disso, do mesmo modo que ocorreu com as métricas analisadas anteriormente, a variação de dimensionalidade não parece impactar os valores alcançados pelos modelos. Por fim, foi possível constatar que o modelo CNN combinado ao vetorizador FastText apresentou resultados menores, assim como na análise das métricas anteriores.

Em relação ao BERTimbau, os resultados também foram mais baixos se comparados com as outras métricas adotadas, apresentando um valor mínimo de 98,96% e um valor máximo de 99,23%, como exibido pela Tabela 11.

Tabela 10: Valores de *recall* para cada combinação de vetorizador, dimensão e classificador, considerando somente as arquiteturas tradicionais.

| Vetorizador | Dimensões | Classificador | Mínimo (%) | Média (%) | Máximo (%) |
|---|---|---|---|---|---|
| Word2Vec | 50 | cnn | 98.68 | 98.88 | 98.99 |
| Word2Vec | 100 | cnn | 98.99 | 99.09 | 99.31 |
| Word2Vec | 300 | cnn | 99.03 | 99.14 | 99.2 |
| Word2Vec | 600 | cnn | 99.01 | 99.14 | 99.27 |
| Word2Vec | 1000 | cnn | 98.92 | 99.16 | 99.31 |
| Word2Vec | 50 | lstm | 98.88 | 99.05 | 99.24 |
| Word2Vec | 100 | lstm | 98.76 | 99.03 | 99.21 |
| Word2Vec | 300 | lstm | 98.96 | 99.1 | 99.23 |
| Word2Vec | 600 | lstm | 98.93 | 99.07 | 99.24 |
| Word2Vec | 1000 | lstm | 99.03 | 99.16 | 99.27 |
| Word2Vec | 50 | mixed | 98.68 | 99.04 | 99.21 |
| Word2Vec | 100 | mixed | 98.83 | 99.02 | 99.24 |
| Word2Vec | 300 | mixed | 98.81 | 99.08 | 99.29 |
| Word2Vec | 600 | mixed | 98.83 | 99.1 | 99.28 |
| Word2Vec | 1000 | mixed | 99.05 | 99.15 | 99.39 |
| GloVe | 50 | cnn | 98.93 | 99.04 | 99.23 |
| GloVe | 100 | cnn | 98.89 | 99.09 | 99.19 |
| GloVe | 300 | cnn | 99.09 | 99.22 | 99.41 |
| GloVe | 600 | cnn | 98.88 | 99.1 | 99.23 |
| GloVe | 1000 | cnn | 99.01 | 99.15 | 99.29 |
| GloVe | 50 | lstm | 98.69 | 98.99 | 99.23 |
| GloVe | 100 | lstm | 99.01 | 99.12 | 99.25 |
| GloVe | 300 | lstm | 99.13 | 99.19 | 99.33 |
| GloVe | 600 | lstm | 98.99 | 99.21 | 99.31 |
| GloVe | 1000 | lstm | 99.01 | 99.19 | 99.32 |
| GloVe | 50 | mixed | 98.96 | 99.09 | 99.23 |
| GloVe | 100 | mixed | 98.91 | 99.15 | 99.36 |
| GloVe | 300 | mixed | 99.07 | 99.21 | 99.33 |
| GloVe | 600 | mixed | 98.79 | 99.08 | 99.25 |
| GloVe | 1000 | mixed | 99.03 | 99.15 | 99.36 |
| FastText | 50 | cnn | 98.48 | 98.76 | 98.89 |
| FastText | 100 | cnn | 98.48 | 98.77 | 99.01 |
| FastText | 300 | cnn | 98.72 | 98.92 | 99.11 |
| FastText | 600 | cnn | 98.75 | 98.92 | 99.08 |
| FastText | 1000 | cnn | 98.68 | 98.93 | 99.11 |
| FastText | 50 | lstm | 98.8 | 99.02 | 99.2 |
| FastText | 100 | lstm | 98.83 | 99.07 | 99.17 |
| FastText | 300 | lstm | 98.81 | 99.02 | 99.21 |
| FastText | 600 | lstm | 99.0 | 99.08 | 99.2 |
| FastText | 1000 | lstm | 98.81 | 98.98 | 99.17 |
| FastText | 50 | mixed | 98.95 | 99.07 | 99.16 |
| FastText | 100 | mixed | 98.95 | 99.11 | 99.2 |
| FastText | 300 | mixed | 98.89 | 99.03 | 99.2 |
| FastText | 600 | mixed | 98.91 | 99.06 | 99.19 |
| FastText | 1000 | mixed | 98.96 | 99.11 | 99.2 |

Tabela 11: Valores de *recall* para o modelo BERTimbau.

| Vetorizador | Mínimo (%) | Média (%) | Máximo (%) |
|---|---|---|---|
| BERTimbau | 98.96 | 99.1 | 99.23 |

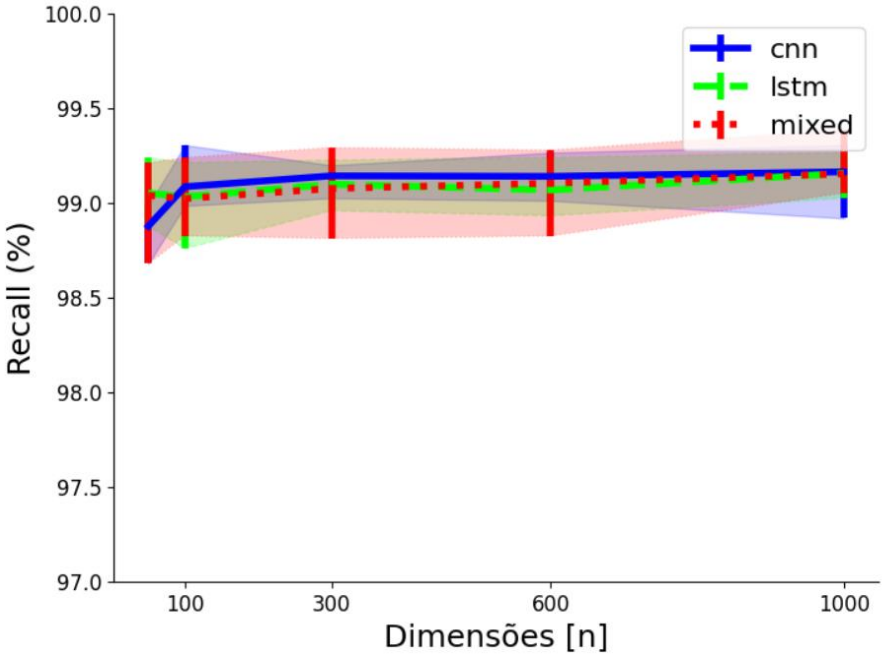


Figura 40: Porcentagem de *recall* dos modelos CNN, LSTM e CNN+LSTM utilizando o *embedding Word2Vec*.

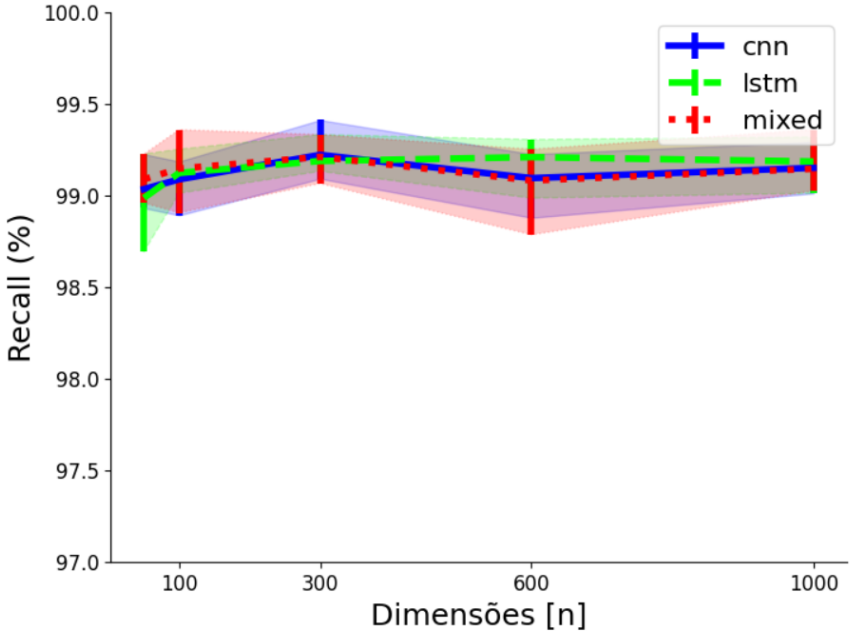


Figura 41: Porcentagem de *recall* dos modelos CNN, LSTM e CNN+LSTM utilizando o *embedding* GloVe.

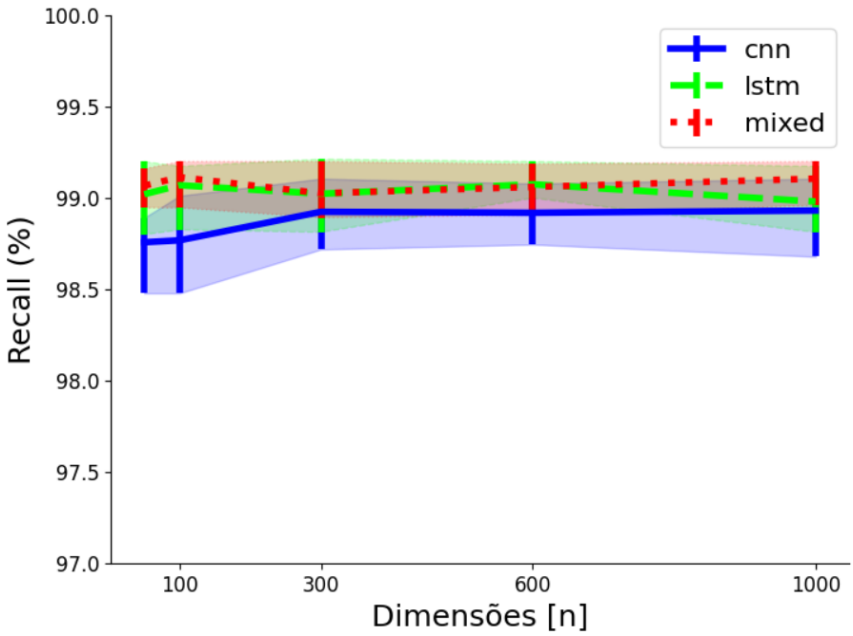


Figura 42: Porcentagem de *recall* dos modelos CNN, LSTM e CNN+LSTM utilizando o *embedding* FastText.

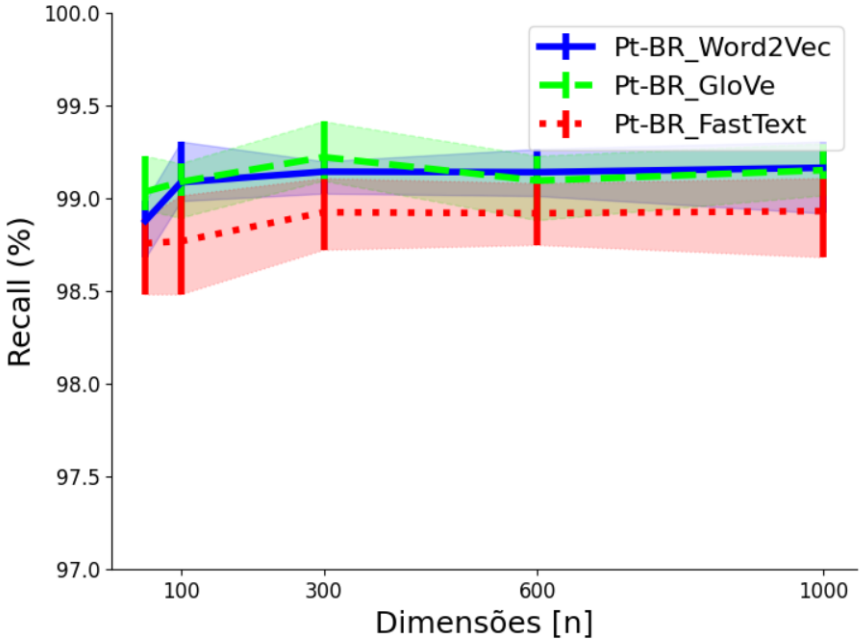


Figura 43: Porcentagem de *recall* dos *embeddings Word2Vec*, GloVe e FastText utilizando o modelo CNN.

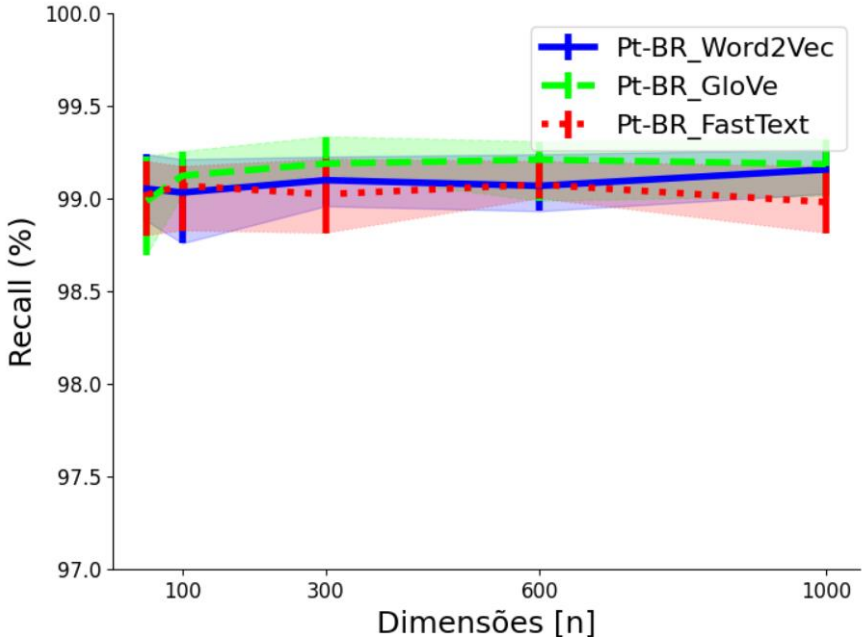


Figura 44: Porcentagem de *recall* dos *embeddings Word2Vec*, GloVe e FastText utilizando o modelo LSTM.

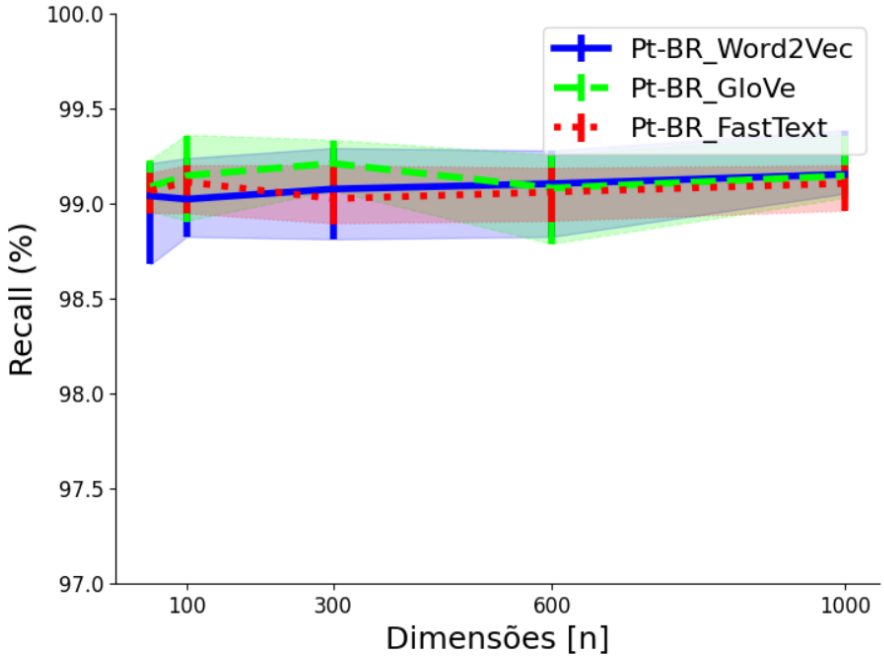


Figura 45: Porcentagem de *recall* dos *embeddings Word2Vec*, GloVe e FastText utilizando o modelo CNN+LSTM.

### 5.6.7 F1-Score

Com o objetivo de realizar uma comparação equilibrada entre as métricas de precisão e *recall*, foi computada a métrica F1-Score, que calcula a média harmônica dessas duas métricas. Sua fórmula pode ser encontrada na Seção 2.4.

A Tabela 12 expõe os valores de F1-Score dos experimentos com arquiteturas tradicionais, variando-se vetorizador, dimensão e classificador. Os gráficos representados nas Figuras 46 a 51 demonstram uma variação entre 98,69% e 99,61%, assim como o intervalo de variação demonstrado pela métrica de precisão. É possível notar uma estabilidade nos resultados obtidos e, assim como na análise de outras métricas, a variação do número de dimensões e as diferentes combinações entre vetorizadores e classificadores parecem não ter influenciado os valores de F1-Score resultantes. Ademais, analogamente ao constatado na análise das outras métricas avaliadas, o classificador CNN combinado ao vetorizador FastText também apresentou resultados menores.

A respeito do BERTimbau, é possível verificar, a partir da Tabela 13, que os resultados obtidos são maiores que os de *recall* e menores que os de precisão, dado que o valor máximo alcançado foi de 99,29% e o valor mínimo de 99,18%.

Tabela 12: Valores de F1-Score para cada combinação de vetorizador, dimensão e classificador, considerando somente as arquiteturas tradicionais.

| Vetorizador | Dimensões | Classificador | Mínimo (%) | Média (%) | Máximo (%) |
|---|---|---|---|---|---|
| Word2Vec | 50 | cnn | 99.06 | 99.17 | 99.34 |
| Word2Vec | 100 | cnn | 98.83 | 99.1 | 99.25 |
| Word2Vec | 300 | cnn | 99.17 | 99.25 | 99.33 |
| Word2Vec | 600 | cnn | 99.12 | 99.28 | 99.42 |
| Word2Vec | 1000 | cnn | 99.16 | 99.26 | 99.45 |
| Word2Vec | 50 | lstm | 98.69 | 98.92 | 99.17 |
| Word2Vec | 100 | lstm | 98.92 | 99.17 | 99.38 |
| Word2Vec | 300 | lstm | 99.04 | 99.24 | 99.3 |
| Word2Vec | 600 | lstm | 99.01 | 99.27 | 99.41 |
| Word2Vec | 1000 | lstm | 99.12 | 99.26 | 99.4 |
| Word2Vec | 50 | mixed | 98.83 | 98.97 | 99.26 |
| Word2Vec | 100 | mixed | 99.03 | 99.26 | 99.41 |
| Word2Vec | 300 | mixed | 99.12 | 99.26 | 99.49 |
| Word2Vec | 600 | mixed | 99.16 | 99.29 | 99.49 |
| Word2Vec | 1000 | mixed | 98.99 | 99.27 | 99.41 |
| GloVe | 50 | cnn | 99.05 | 99.15 | 99.28 |
| GloVe | 100 | cnn | 99.24 | 99.32 | 99.45 |
| GloVe | 300 | cnn | 99.19 | 99.32 | 99.42 |
| GloVe | 600 | cnn | 99.2 | 99.39 | 99.58 |
| GloVe | 1000 | cnn | 99.27 | 99.34 | 99.52 |
| GloVe | 50 | lstm | 98.92 | 99.13 | 99.41 |
| GloVe | 100 | lstm | 99.12 | 99.23 | 99.4 |
| GloVe | 300 | lstm | 99.21 | 99.35 | 99.41 |
| GloVe | 600 | lstm | 99.2 | 99.31 | 99.52 |
| GloVe | 1000 | lstm | 99.19 | 99.31 | 99.48 |
| GloVe | 50 | mixed | 98.92 | 99.12 | 99.24 |
| GloVe | 100 | mixed | 99.03 | 99.23 | 99.42 |
| GloVe | 300 | mixed | 99.05 | 99.25 | 99.41 |
| GloVe | 600 | mixed | 99.15 | 99.32 | 99.61 |
| GloVe | 1000 | mixed | 99.08 | 99.25 | 99.41 |
| FastText | 50 | cnn | 98.8 | 99.03 | 99.23 |
| FastText | 100 | cnn | 98.95 | 99.18 | 99.38 |
| FastText | 300 | cnn | 98.93 | 99.17 | 99.45 |
| FastText | 600 | cnn | 99.09 | 99.19 | 99.38 |
| FastText | 1000 | cnn | 98.95 | 99.12 | 99.38 |
| FastText | 50 | lstm | 98.84 | 99.15 | 99.32 |
| FastText | 100 | lstm | 99.0 | 99.19 | 99.41 |
| FastText | 300 | lstm | 99.01 | 99.27 | 99.44 |
| FastText | 600 | lstm | 99.11 | 99.31 | 99.42 |
| FastText | 1000 | lstm | 99.23 | 99.36 | 99.54 |
| FastText | 50 | mixed | 98.96 | 99.12 | 99.24 |
| FastText | 100 | mixed | 98.87 | 99.1 | 99.37 |
| FastText | 300 | mixed | 99.11 | 99.28 | 99.38 |
| FastText | 600 | mixed | 99.19 | 99.33 | 99.56 |
| FastText | 1000 | mixed | 99.07 | 99.24 | 99.46 |

Tabela 13: Valores de F1-Score para o modelo BERTimbau.

| Vetorizador | Mínimo (%) | Média (%) | Máximo (%) |
|---|---|---|---|
| BERTimbau | 99.18 | 99.23 | 99.29 |

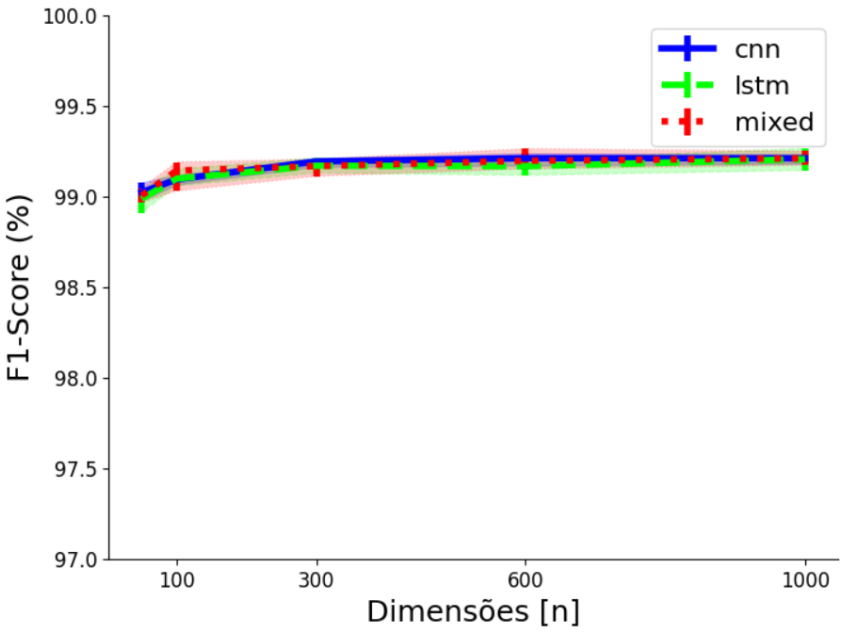


Figura 46: Porcentagem de F1-Score dos modelos CNN, LSTM e CNN+LSTM utilizando o *embedding Word2Vec*.

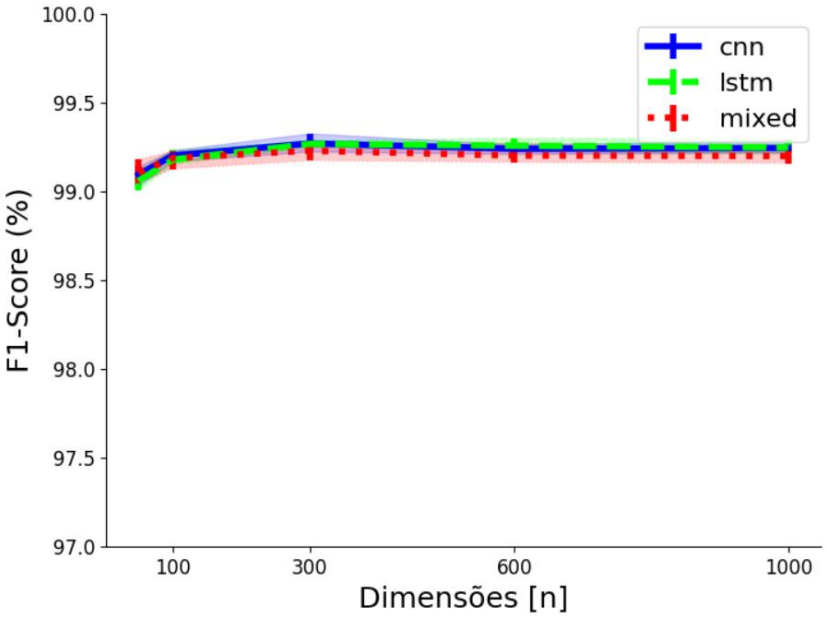


Figura 47: Porcentagem de F1-Score dos modelos CNN, LSTM e CNN+LSTM utilizando o *embedding* GloVe.

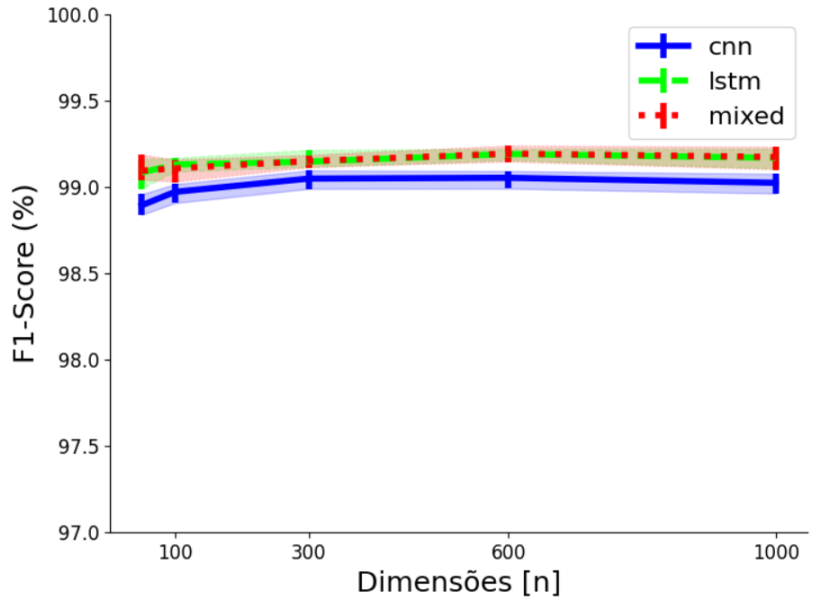


Figura 48: Porcentagem de F1-Score dos modelos CNN, LSTM e CNN+LSTM utilizando o *embedding* FastText.

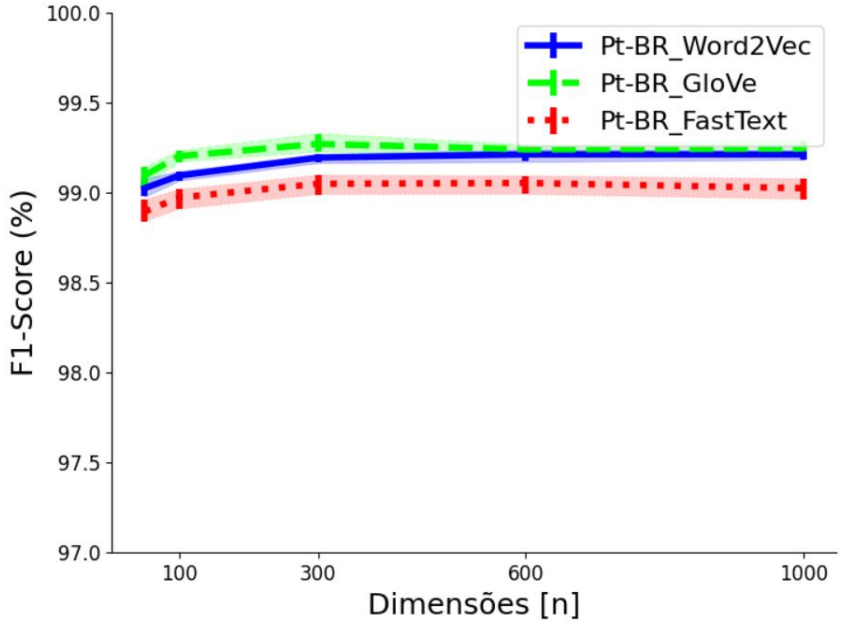


Figura 49: Porcentagem de F1-Score dos *embeddings Word2Vec*, GloVe e FastText utilizando o modelo CNN.

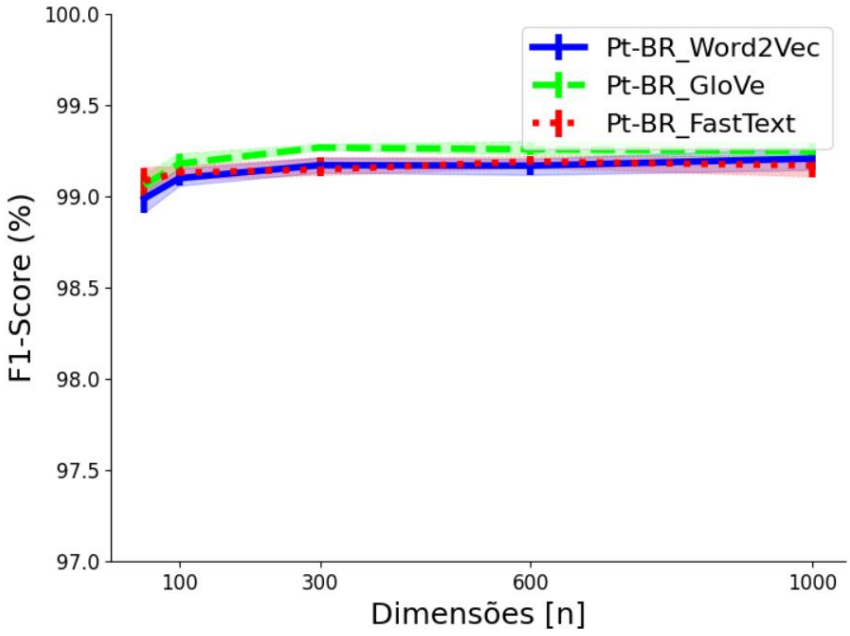


Figura 50: Porcentagem de F1-Score dos *embeddings Word2Vec*, GloVe e FastText utilizando o modelo LSTM.

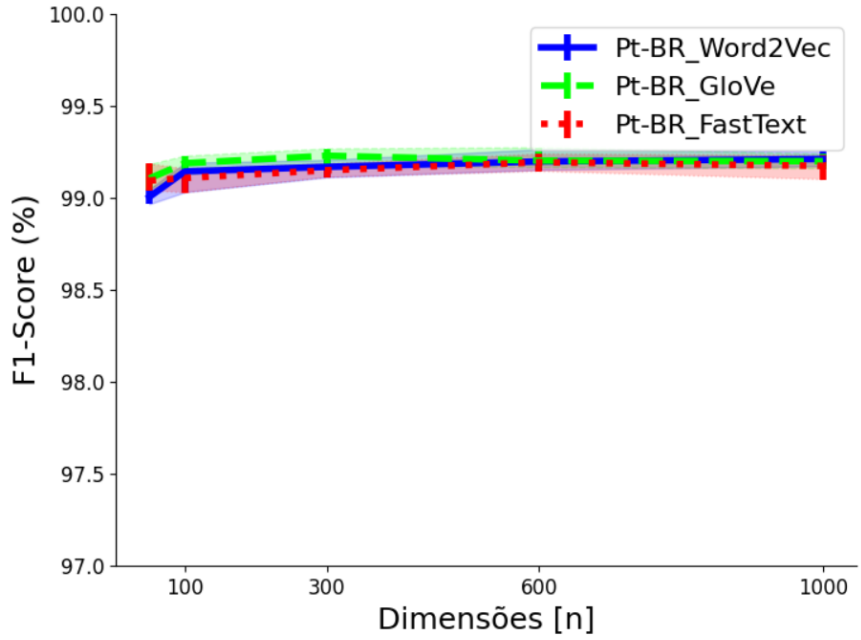


Figura 51: Porcentagem de F1-Score dos *embeddings Word2Vec*, GloVe e FastText utilizando o modelo CNN + LSTM.

## 5.7 Discussão dos Resultados

As arquiteturas tradicionais utilizadas neste trabalho foram retiradas de Teixeira *et al*. (2020), em que a tarefa de classificação de microtextos de trânsito foi abordada através de experimentos com os modelos CNN, LSTM e o modelo combinado de CNN e

LSTM. Por esse motivo, é interessante comparar os resultados obtidos neste estudo com os resultados alcançados previamente. No entanto, destaca-se que o *dataset* utilizado por Teixeira *et al*. (2020) continha microtextos distintos e contabilizou um total de 43.152 *tweets*, enquanto o *dataset* utilizado nesta pesquisa totalizou 64.237 registros. Ademais, com o objetivo de enriquecer a análise de desempenho de diferentes classificadores no problema apresentado, foram conduzidos, neste trabalho, experimentos com uma arquitetura *Transformer*. O modelo adotado foi obtido a partir do ajuste fino do BERTimbau no *dataset* proposto e, dado que o estudo original não realizou experimentos com esse tipo de arquitetura, esta pesquisa contribui com a análise de resultados de um modelo *Transformer* para uma comparação mais abrangente entre classificadores comumente utilizados em tarefas de PLN.

Desse modo, foi possível constatar que os resultados alcançados foram razoavelmente maiores do que os resultados obtidos por Teixeira *et al*. (2020), com acurácia, precisão e *recall* em torno de 98%, enquanto no estudo anterior tais métricas se concentraram em torno de 96%. Em relação às combinações de classificadores e vetorizadores, os modelos CNN e LSTM associados ao vetorizador GloVe se destacaram, enquanto Teixeira *et al*. (2020) observou que as combinações entre o vetorizador FastText e os modelos CNN e CNN + LSTM exibiram as melhores performances. Além disso, o vetorizador GloVe apresentou melhores resultados de modo geral, em contraste com o que foi observado pelo trabalho antecedente, em que o vetorizador FastText demonstrou o melhor desempenho. Por fim, em relação à variação dimensional dos *word embeddings*, não foi constatada nenhuma influência significativa nos valores resultantes das métricas analisadas. No entanto, assim como visto em Teixeira *et al*. (2020), notou-se que, conforme o número de dimensões cresce, os tempos de vetorização, treinamento e inferência também se estendem.

# 6 Conclusão

## 6.1 Considerações Finais

Devido ao cenário de trânsito nas cidades brasileiras, o acesso a informações atualizadas sobre condições de tráfego se torna essencial para uma locomoção mais eficiente e segura. Além da falta de planejamento urbano, ocorrências isoladas como acidentes, obras e condições climáticas também podem promover congestionamentos e lentidão. Na última década, observou-se um crescimento expressivo do uso das redes sociais a partir da popularização dos *smartphones*. Com fácil acesso à internet, usuários impactados por incidentes de trânsito passaram a relatar em diversas plataformas a insatisfação com os trajetos percorridos.

Com o objetivo de extrair conhecimento dos diversos dados não estruturados compartilhados nas redes, é possível utilizar técnicas de Aprendizagem Profunda na tarefa de detecção de eventos de trânsito em dados textuais. Dessa maneira, este trabalho propôs o uso de modelos de Aprendizagem Profunda na abordagem a tal tarefa e realizou diversos experimentos computacionais com o intuito de comparar o desempenho apresentado por cada modelo. Os dados utilizados são provenientes da rede social Twitter e constituem um *dataset* com 64.237 registros, sendo 32.136 *tweets* provenientes do domínio de trânsito e 32.101 *tweets* provenientes do público geral. Assim, após serem rotulados manualmente, os dados coletados se dividem entre as categorias “Evento relevante de trânsito” e “Evento não relevante”.

As arquiteturas utilizadas nos experimentos foram CNN, LSTM, a combinação CNN + LSTM e BERTimbau. Eles foram escolhidos por sua relevância na área de Aprendizagem Profunda e também pelo bom desempenho observado na resolução de tarefas de PLN, sendo apropriados para o problema apresentado. Em especial, o uso do BERTimbau neste trabalho é significativo por possibilitar a comparação de desempenho entre um modelo baseado na arquitetura *Transformer* e modelos de arquiteturas

tradicionais de Aprendizagem Profunda. Além disso, pelo fato do BERTimbau se tratar de um modelo de linguagem *sequence-to-sequence*, sua aplicação em uma tarefa de classificação se caracteriza como *transfer learning* através do *fine-tuning* do modelo no *dataset* coletado.

Segundo as métricas tradicionais de avaliação, os modelos apresentaram, de modo geral, um ótimo desempenho, com acurácia, precisão, *recall* e F1-Score em torno de 98%. Em relação aos modelos de arquitetura tradicional, foi constatado um desempenho maior das redes CNN e LSTM combinadas ao vetor GloVe. Além disso, assim como observado por Teixeira *et al*. (2020), a rede CNN, combinada a qualquer um dos vetorizadores, apresenta os melhores tempos de treinamento por sua maior simplicidade.

No que se refere à variação dimensional, não foi constatada nenhuma influência significativa nos valores resultantes das métricas analisadas. No entanto, é importante ressaltar seu impacto na eficiência dos modelos ao afetarem os tempos de vetorização, treinamento e inferência. Observou-se que, à medida que o número de dimensões aumenta, maiores são as durações observadas para os tempos mencionados anteriormente. Por fim, diferentemente do observado por Teixeira *et al*. (2020), que apresentou melhores resultados com o vetorizador FastText, o vetorizador GloVe apresentou resultados razoavelmente maiores do que os outros *word embeddings* utilizados.

Com relação ao uso do BERTimbau, os resultados alcançados foram bem próximos aos resultados obtidos pelas arquiteturas tradicionais. Se comparado a outras tarefas de PLN, como por exemplo a tarefa de Reconhecimento de Entidades Nomeadas, o problema de classificação binário, como é o caso deste trabalho, pode ser considerado simples. Desse modo, o uso de modelos menos custosos e mais rápidos como CNNs pode ter um melhor custo-benefício, considerando que o tempo de treinamento do BERTimbau é muito maior do que os tempos observados para os outros classificadores. No entanto, é importante mencionar que os experimentos não foram executados com GPU, e seu uso poderia melhorar os tempos de vetorização, treinamento e inferência de todos os modelos.

## 6.2 Trabalhos Futuros

Como trabalhos futuros, vislumbra-se a realização de experimentos utilizando *Large Language Models* (LLMs). Apesar do BERT ser um modelo de linguagem, ele é considerado um *small language model* devido à sua quantidade de parâmetros. Enquanto o modelo GPT-4, considerado um LLM, possui 1,76 trilhões de parâmetros, o BERT apresenta 110 milhões. De maneira geral, quanto maior o número de parâmetros do modelo, maior a sua capacidade de resolução de problemas computacionais.

Ao trabalhar com dados textuais em português brasileiro, é recomendável utilizar LLMs que tenham sido pré-treinados em um *corpora* de mesmo idioma. Além disso, é aconselhado o uso de modelos monolíngues, visto que, em geral, o desempenho desses costuma ser maior do que o desempenho de modelos multilíngues, como mostrado por Rust *et al*. (2021), especialmente no caso de línguas com poucos dados disponíveis como o português. Desse modo, modelos como Sabiá-2 (ALMEIDA *et al*., 2024), Canarim-7B (MAICON DOMINGUES, 2023), Bode (GARCIA *et al*., 2024), Cabrita (LARCHER *et al*., 2023) e Albertina-PTBR (RODRIGUES *et al*., 2023) podem ser considerados bons candidatos para a realização de tais experimentos, dado que foram treinados especificamente para a língua portuguesa.

Além disso, para aumentar o nível de complexidade da tarefa, os experimentos poderiam incluir a detecção da localização do evento de trânsito uma vez que um texto for classificado como “Evento relevante de trânsito”. Desse modo, é possível propor adicionalmente a construção de um *framework* para visualização do estado do trânsito em tempo real, permitindo que informações consideradas como "Evento Relevante de Trânsito" sejam extraídas e exibidas em um mapa, auxiliando no processo de tomada de decisão no que diz respeito à escolha de trajetos e horário de locomoção.

# Referências Bibliográficas


1. ALMEIDA, T. S. *et al*. Sabi\'a-2: A New Generation of Portuguese Large Language Models. arXiv, , 26 mar. 2024. Disponível em: <http://arxiv.org/abs/2403.09887>. Acesso em: 30 jun. 2024

2. AGGARWAL, Charu C. Neural Networks and Deep Learning: A Textbook. 2. ed. Springer, 2023. 529 p.

3. ARC. Derivative of the Sigmoid function. Disponível em: <https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e>. Acesso em: 30 mar. 2024.

4. BEDI, J.; TOSHNIWAL, D. CitEnergy : A BERT based model to analyse Citizens' Energy-Tweets. Sustainable Cities and Society, v. 80, p. 103706, maio 2022.

5. BOJANOWSKI, P. *et al*. Enriching Word Vectors with Subword Information. arXiv, , 19 jun. 2017. Disponível em: <http://arxiv.org/abs/1607.04606>. Acesso em: 20 jun. 2024

6. Carta Brasileira para Cidades Inteligentes. Disponível em: <https://cartacidadesinteligentes.org.br/>. Acesso em: 22 set. 2023.

7. CHAMBY-DIAZ, J.C; BAZZAN, A. Identifying traffic event types from twitter by multi-label classification. In: 2019 8th Brazilian conference on intelligent systems (bracis). IEEE, 2019. p. 806-811.

8. DABIRI, S.; HEASLIP, K. Developing a Twitter-based traffic event detection model using deep learning architectures. Expert Systems with Applications, v. 118, p. 425–439, mar. 2019.

9. MAICON DOMINGUES. canarim-7b (Revision 08fdd2b). Hugging Face, , 2023. Disponível em: <https://huggingface.co/dominguesm/canarim-7b>

10. Estadão. Agosto de 2022. Mobilidade urbana: um grande desafio para as cidades no século 21. https://mobilidade.estadao.com.br/mobilidade-para-que/mobilidade-urbana-um-grande-desafio-para-as-cidades-no-seculo-xxi/.

11. FENG, J. *et al*. Reconstruction of porous media from extremely limited information using conditional generative adversarial networks. Physical Review E, v. 100, 16 set. 2019.

12. FERREIRA, F.; DUARTE, J.; UGULINO, W. Automated Statistics Extraction of Public Security Events Reported Through Microtexts on Social Networks. XVIII Brazilian Symposium on Information Systems. Anais... Em: SBSI: XVIII BRAZILIAN SYMPOSIUM ON INFORMATION SYSTEMS. Curitiba Brazil: ACM, 16 maio 2022. Disponível em: <https://dl.acm.org/doi/10.1145/3535511.3535513>. Acesso em: 23 nov. 2023

13. Folha de S. Paulo. Agosto de 2023. 36% dos brasileiros passam mais de 1 hora por dia no transporte. https://www1.folha.uol.com.br/cotidiano/2023/08/36-dos-brasileiros-passam-mais-de-1h-por-dia-no-transporte.shtml

14. GARCIA, G. L. *et al*. Introducing Bode: A Fine-Tuned Large Language Model for Portuguese Prompt-Based Task. arXiv, , 5 jan. 2024. Disponível em: <http://arxiv.org/abs/2401.02909>. Acesso em: 30 jun. 2024

15. G1. Março de 2023. Com 1.206 km, cidade de SP bate recorde anual de trânsito nesta quarta, segundo novo método de medição da CET. https://g1.globo.com/sp/sao-paulo/noticia/2023/03/08/com-1206-km-cidade-de-sp-bate-recorde-anual-de-transito-nesta-quarta-segundo-novo-metodo-de-medicao-da-cet.ghtml.

16. GLOROT, Xavier; BENGIO, Yoshua. Understanding the difficulty of training deep feedforward neural networks. The 13th International Conference on Artificial Intelligence and Statistics (AISTATS 2010), JMLR.org. p. 249 - 256.

17. HARTMANN, N. *et al*. Portuguese Word Embeddings: Evaluating on Word Analogies and Natural Language Tasks. arXiv, , 20 ago. 2017. Disponível em: <http://arxiv.org/abs/1708.06025>. Acesso em: 29 jun. 2024

18. He, K.; Zhang, X.; Ren, S. & Sun, J.. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. 2015 IEEE International Conference on Computer Vision (ICCV 2015), IEEE Computer Society. p. 1026 - 1034.

19. HOCHREITER, Sepp & SCHMIDHUBER, Jürgen. (1997). Long Short-term Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.

20. KARIM, R. Illustrated: Self-Attention. Disponível em: <https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a>. Acesso em: 6 abr. 2024.

21. KOTO, F. *et al*. IndoLEM and IndoBERT: A Benchmark Dataset and Pre-trained Language Model for Indonesian NLP. arXiv, , 1 nov. 2020. Disponível em: <http://arxiv.org/abs/2011.00677>. Acesso em: 27 fev. 2024

22. KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. ImageNet Classification with Deep Convolutional Neural Networks. Advances in neural information processing systems, v. 25, p. 1097-1105, 2012.

23. KROHN, JON; BEYLEVELD, GRANT; BLASSENS, AGLAÉ. Deep learning illustrated, 2020.

24. LESKOVEC, J.; RAJARAMAN, A.; ULLMAN, J. D. Mining of Massive Datasets. [s.d.].

25. LARCHER, C. et al. Cabrita: closing the gap for foreign languages. arXiv, , 22 ago. 2023. Disponível em: <http://arxiv.org/abs/2308.11878>. Acesso em: 30 jun. 2024

26. MENDHE, C. et al. A Scalable Platform to Collect, Store, Visualize, and Analyze Big Data in Real Time. IEEE Transactions on Computational Social Systems, v. PP, p. 1–10, 3 jun. 2020.

27. MIKOLOV, T. et al. Efficient Estimation of Word Representations in Vector Space. arXiv, , 6 set. 2013. Disponível em: <http://arxiv.org/abs/1301.3781>. Acesso em: 7 nov. 2023

28. NIRBHAYA, M. A. W.; SUADAA, L. H. Traffic Incident Detection in Jakarta on Twitter Texts Using a Multi-Label Classification Approach. 2023 International Conference on Computer, Control, Informatics and its Applications (IC3INA). Anais... Em: 2023 INTERNATIONAL CONFERENCE ON COMPUTER, CONTROL, INFORMATICS AND ITS APPLICATIONS (IC3INA). out. 2023. Disponível em: <https://ieeexplore.ieee.org/document/10285731>. Acesso em: 23 fev. 2024

29. KINGMA, D. P.; BA, J. Adam: A Method for Stochastic Optimization. arXiv, , 29 jan. 2017. Disponível em: <http://arxiv.org/abs/1412.6980>. Acesso em: 7 jun. 2024

30. PENNINGTON, J.; SOCHER, R.; MANNING, C. GloVe: Global Vectors for Word Representation. (A. Moschitti, B. Pang, W. Daelemans, Eds.)Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). Anais... Em: EMNLP 2014. Doha, Qatar: Association for Computational Linguistics, out. 2014. Disponível em: <https://aclanthology.org/D14-1162>. Acesso em: 20 jun. 2024

31. QUDAR, M. M. A.; MAGO, V. TweetBERT: A Pretrained Language Representation Model for Twitter Text Analysis. arXiv, , 16 out. 2020. Disponível em: <http://arxiv.org/abs/2010.11091>. Acesso em: 23 nov. 2023

32. RADFORD, A. et al. Improving Language Understanding by Generative Pre-Training. [s.d.].

33. RIBEIRO JUNIOR, R. F.; DE ALMEIDA, F. A.; GOMES, G. F. Fault classification in three-phase motors based on vibration signal analysis and artificial neural networks. Neural Computing and Applications, v. 32, n. 18, p. 15171–15189, 1 set. 2020.

34. RODRIGUES, J. et al. Advancing Neural Encoding of Portuguese with Transformer Albertina PT-*. Em: [s.l: s.n.]. v. 14115p. 441–453.

35. RUSSAKOVSKY, O. et al. ImageNet Large Scale Visual Recognition Challenge. arXiv, , 29 jan. 2015. Disponível em: <http://arxiv.org/abs/1409.0575>. Acesso em: 7 fev. 2024

36. RUST, P. et al. How Good is Your Tokenizer? On the Monolingual Performance of Multilingual Language Models. arXiv, , 1 jun. 2021. Disponível em: <http://arxiv.org/abs/2012.15613>. Acesso em: 3 jul. 2024

37. SANTOS, L. L.; BIANCHI, R. A. C.; COSTA, A. H. R. FinBERT-PT-BR: Análise de Sentimentos de Textos em Português do Mercado Financeiro. Anais do Brazilian Workshop on Artificial Intelligence in Finance (BWAIF). Anais... Em: ANAIS DO II BRAZILIAN WORKSHOP ON ARTIFICIAL INTELLIGENCE IN FINANCE. SBC, 6 ago. 2023. Disponível em: <https://sol.sbc.org.br/index.php/bwaif/article/view/24960>. Acesso em: 23 nov. 2023

38. SHARMA, S. Activation Functions in Neural Networks. Disponível em: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>. Acesso em: 30 mar. 2024.

39. SOUZA, F.; NOGUEIRA, R.; LOTUFO, R. BERTimbau: Pretrained BERT Models for Brazilian Portuguese. Em: [s.l: s.n.]. p. 403–417.

40. TEIXEIRA, E.; MOURA, P.; CAMPOS, C.A. Classificação de Tweets sobre Trânsito Utilizando Diferentes Técnicas de Deep Learning. In: Anais Estendidos do X Simpósio Brasileiro de Engenharia de Sistemas Computacionais. SBC, 2020. p. 89-96.

41. VASWANI, A. et al. Attention Is All You Need. arXiv, , 12 jun. 2017. Disponível em: <http://arxiv.org/abs/1706.03762>. Acesso em: 30 mar. 2024