UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

INSTITUTO DE COMPUTAÇÃO

CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

IVAN ALVARENGA DE SOUSA JÚNIOR

CLASSIFICAÇÃO DE INTENÇÃO DE CITAÇÃO USANDO TÉCNICAS DE APRENDIZADO *FEW-SHOT*

RIO DE JANEIRO

2024

IVAN ALVARENGA DE SOUSA JUNIOR

CLASSIFICAÇÃO DE INTENÇÃO DE CITAÇÃO USANDO TÉCNICAS DE APRENDIZADO *FEW-SHOT*

Trabalho de conclusão de curso de graduação apresentado ao Instituto de Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientadora: Profa. Giseli Rabello Lopes

RIO DE JANEIRO

2024

CIP - Catalogação na Publicação

S725c

Sousa Junior, Ivan Alvarenga de
Classificação de Intenção de Citação Usando Técnicas de Aprendizado Few-shot / Ivan Alvarenga de Sousa Junior. -- Rio de Janeiro, 2024.
72 f.

Orientadora: Giseli Rabello Lopes.
Trabalho de conclusão de curso (graduação) - Universidade Federal do Rio de Janeiro, Instituto de Computação, Bacharel em Ciência da Computação, 2024.

1. aprendizado de máquina. 2. redes neurais. 3. processamento de linguagem natural. I. Lopes, Giseli Rabello, orient. II. Título.

Elaborado pelo Sistema de Geração Automática da UFRJ com os dados fornecidos pelo(a) autor(a), sob a responsabilidade de Miguel Romeu Amorim Neto - CRB-7/6283.

IVAN ALVARENGA DE SOUSA JUNIOR

CLASSIFICAÇÃO DE INTENÇÃO DE CITAÇÃO USANDO TÉCNICAS DE APRENDIZADO *FEW-SHOT*

Trabalho de conclusão de curso de graduação apresentado ao Instituto de Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 01 de abril de 2024.

BANCA EXAMINADORA:

gov.br Documento assinado digitalmente
GISELI RABELLO LOPES
Data: 11/04/2024 23:44:07-0300
Verifique em https://validar.iti.gov.br

Profa. Giseli Rabello Lopes, D.Sc. (UFRJ)

gov.br Documento assinado digitalmente
JOAO CARLOS PEREIRA DA SILVA
Data: 12/04/2024 08:48:49-0300
Verifique em https://validar.iti.gov.br

Prof. João Carlos P. da Silva, D.Sc. (UFRJ)

gov.br Documento assinado digitalmente
MARIA LUIZA MACHADO CAMPOS
Data: 12/04/2024 09:19:39-0300
Verifique em https://validar.iti.gov.br

Profa. Maria Luiza Machado Campos, Ph.D. (UFRJ)

Dedico este trabalho à minha família.

AGRADECIMENTOS

Agradeço aos meus pais (Ivan e Iêda), ao meu irmão (Yuri) e a minha namorada (Bruna) por todo apoio e incentivo. Agradeço também a UFRJ e todos os seus funcionários por toda competência em manter a Universidade Pública funcionando da melhor forma. Agradeço também à professora Giseli Lopes por toda a atenção, dedicação e paciência que sempre teve ao me orientar em todo o período. Além disso, agradeço ao Hugo por disponibilizar tempo para acompanhar o desenvolvimento do trabalho.

Gostaria também de agradecer a professora Valéria Bastos que foi uma referência desde que a conheci e continua sendo mesmo após a conclusão de minha graduação.

Muito obrigado a todos!

RESUMO

Este estudo investiga a eficácia da metodologia de treinamento *few-shot SetFit* (*Sentence Transformer Fine-tuning*) na classificação das intenções de citação em pesquisas científicas. Com o aumento exponencial de publicações, torna-se crucial identificar trabalhos relevantes e entender as relações entre eles. Utilizando algoritmos de Processamento de Linguagem Natural (PLN), este trabalho compara a metodologia *few-shot* com outras abordagens tradicionais, observando o impacto do tamanho do conjunto de treinamento no desempenho dos modelos. Os resultados indicam que, mesmo com um número limitado de exemplos, a metodologia *SetFit* consegue refinar modelos de linguagem para classificar intenções de citação de forma eficiente, aproximando-se da precisão de modelos mais complexos. Portanto, conclui-se que a abordagem *few-shot* é uma alternativa promissora para aprimorar o mapeamento de intenções de citação, facilitando a construção do conhecimento e a identificação de artigos científicos relevantes.

Palavras-chave: citações científicas; processamento de linguagem natural; metodologia de treino *few-shot*; classificação de intenções de citação; mapeamento de intenções.

ABSTRACT

This study investigates the effectiveness of the few-shot SetFit (Sentence Transformer Fine-tuning) training methodology in classifying citation intentions in scientific research. With the exponential increase in publications, it becomes crucial to identify relevant works and understand the relationships between them. Using Natural Language Processing (NLP) algorithms, this work compares the few-shot methodology with other traditional approaches, observing the impact of the training set size on the performance of the models. The results indicate that, even with a limited number of examples, the SetFit methodology can refine language models to efficiently classify citation intentions, approaching the accuracy of more complex models. Therefore, it is concluded that the few-shot approach is a promising alternative for improving the mapping of citation intentions, facilitating the construction of knowledge and the identification of relevant scientific articles.

Keywords: scientific citations; natural language processing; few-shot training methodology; citation intention classification; intention mapping.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ACL	*Association for Computational Linguistics*
BERT	*Bidirectional Encoder Representations from Transformers*
BPTT	*BackPropagation Through Time*
CBOW	*Continuous Bag of Words*
EMQ	Erro Médio Quadrático
GD	Gradiente Descendente
GPT	*Generative Pre-trained Transformer*
KL	Kullback-Leibler
LLM	*Large Language Models*
LSTM	*Long Short-Term Memory*
MLP	*Multilayer Perceptron*
PLN	Processamento de Linguagem Natural
RNA	Rede Neural Artificial
RNN	*Recurrent Neural Network*
SetFit	*Sentence Transformer Fine-tuning*
ST	*Sentence Transformers*
TLU	*Threshold Logical Unit*
VarMAE	*Variational Masked AutoEncoder*

SUMÁRIO

1 INTRODUÇÃO

Citações são utilizadas para referenciar o estado da pesquisa científica em determinado tópico e identificar trabalhos relevantes para linhas de pesquisa (JURGENS et al., 2018). Assim, as citações desempenham uma função fundamental na evolução do conhecimento, servindo como base para a construção de uma linha de pensamento proposta pelo autor. Também segundo Jurgens et al. (2018), a forma como o autor estrutura as citações no artigo impacta diretamente como este será recebido e qual será o impacto da publicação no futuro.

Com o passar dos anos, o número de publicações científicas aumentou exponencialmente, tornando o acompanhamento de todo conhecimento gerado virtualmente impossível. Então, viu-se a necessidade de criar um mapeamento das relações entre os artigos, com o intuito de avaliar a importância dos mesmos baseando-se no número de referências que um determinado artigo recebeu (IHSAN; QADIR, 2019). Assim, a principal forma de representar a relação entre os artigos, normalmente, é através da criação de grafos de citações, nos quais, os artigos são representados por nós e a citação é representada por uma aresta direcionada que sai do artigo que cita e aponta para o artigo citado.

A representação e análise de grafos de citações frequentemente não consideram os motivos subjacentes que levam um autor a citar determinados artigos, ou seja, a intenção por trás da citação. Esse aspecto é crucial, pois, especialmente em contextos como a pandemia do COVID-19, muitos artigos publicados foram posteriormente retratados por variadas razões, como falta de rigor científico ou insuficiência de dados (TAROS et al., 2023). Durante a pandemia, houve casos em que trabalhos foram referenciados antes de serem retratados, gerando dúvidas se as citações foram feitas para apoiar ou discordar dos resultados apresentados. Esse cenário destaca uma falha significativa nas abordagens convencionais de análise de grafos de citações, que não capturam a complexidade e as nuances das intenções de citação dos autores.

Conforme Jurgens et al. (2018), houve diversos trabalhos que procuraram analisar as características das citações feitas nos artigos citantes e sua relação com o trabalho citado. A princípio, realizou-se análises manuais (MORAVCSIK; MURUGESAN, 1975; SWALES, 1990; HARWOOD, 2009), e recentemente, com os avanços dos algoritmos de Processamento de Linguagem Natural (PLN), foram realizadas abordagens automatizadas que utilizam essas técnicas, especialmente através do uso de Redes Neurais (JURGENS et al.,2018; COHAN et al., 2019; HU et al., 2022).

No entanto, as técnicas atuais para a criação de modelos de linguagens necessitam de uma grande quantidade de dados para o treinamento (DEVLIN et al., 2019), o que faz com que algumas áreas do conhecimento, nas quais a coleta de dados é custosa, demorada ou até inviável, não obtenham modelos com desempenho satisfatório. Assim, foram propostas algumas técnicas de aprendizado *few-shot* (LIU et al., 2022; TAM et al., 2021; MAHABADI et al., 2022) para que fosse possível realizar o refinamento (*fine-tuning*) de modelos pré-treinados mesmo com uma quantidade pequena de exemplos de treinamento.

Este trabalho tem como objetivo analisar o desempenho da metodologia de treinamento *few-shot* (WANG et al., 2020) em comparação com outras abordagens, a fim de validar sua eficiência no treinamento de modelos para classificação de intenções de citações. Ao investigar e comparar os resultados obtidos com outros trabalhos relacionados, buscamos demonstrar a eficiência desta abordagem inovadora, construir e disponibilizar um modelo refinado que seja capaz de classificar a intenção de uma determinada citação.

O restante deste trabalho está organizado da seguinte forma. No segundo capítulo, é descrita a fundamentação teórica sobre redes neurais, refinamento (*fine tuning)*, *few-shot learning*, *word embeddings*, *SetFit* e medidas de avaliação. No terceiro capítulo, são discutidos os trabalhos relacionados que utilizam técnicas distintas com o objetivo de treinar modelos para a classificação de intenção de citação. No quarto capítulo, são detalhados o conjunto de dados, o ambiente de execução, as configurações específicas dos experimentos e os resultados associados. Por fim, no quinto capítulo, estão as considerações finais e os possíveis trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta teórica e matematicamente as principais arquiteturas utilizadas na área de Processamento de Linguagem Natural (PLN), com o objetivo de deixar o leitor a par das principais técnicas que são utilizadas atualmente. Primeiro, será falado sobre Redes Neurais e as evoluções das arquiteturas para atender problemas mais complexos. Também será abordado sobre o refinamento de modelos de linguagem mais genéricos para tarefas mais específicas. Por fim, serão abordadas brevemente as métricas de avaliação utilizadas neste trabalho.

2.1 REDES NEURAIS

Redes Neurais Artificiais (RNA), ou apenas Redes Neurais, são estruturas computacionais que visam reproduzir o comportamento do cérebro humano. Essas estruturas são compostas por unidades de processamento (ou neurônios), que também são inspiradas na arquitetura e funcionamento dos neurônios biológicos. As Redes Neurais Artificiais ocupam uma posição central no campo do Aprendizado Profundo (*Deep Learning*) (RUSSELL; NORVIG, 2016) devido à sua capacidade versátil, escalabilidade e habilidade para identificar padrões nos dados.

Nos últimos anos, aplicações com Redes Neurais ganharam destaque devido aos excelentes resultados apresentados diante de outros algoritmos mais clássicos. Uma das principais vantagens de tais estruturas é a possibilidade de extrair padrões complexos a partir de dados brutos, permitindo que fossem aplicadas em problemas com difícil definição formal, como reconhecimento de faces, ou na conversão de áudio para texto.

2.1.1 Definição

As Redes Neurais Artificiais (RNAs) são modelos computacionais que mimetizam o sistema nervoso dos seres vivos, consistindo fundamentalmente de neurônios e camadas. Uma RNA típica engloba uma camada de entrada, várias camadas intermediárias ou ocultas, e uma camada de saída, estrutura conhecida como *perceptron* de multicamada (*Multilayer Perceptron*, MLP), conforme pode ser observado na Figura 1.

As MLPs são estruturadas com camadas que incluem unidades de lógica de limiar (*Threshold Logical Units*, TLUs) tanto nas intermediárias quanto na saída. Cada camada, com exceção da última, contém uma unidade de viés para ajustar o limiar de ativação dos

neurônios. A informação flui da camada de entrada, através das camadas intermediárias, onde ocorrem processamentos complexos, e finalmente chega à camada de saída, onde o resultado é produzido.

Esta arquitetura permite a modelagem de relações complexas entre os dados de entrada, por meio de ajustes nos pesos dos neurônios, que são modificados durante o processo de aprendizagem. Este ajuste é fundamental para que a rede possa extrair e aprender as relações subjacentes entre as variáveis de entrada e o resultado desejado.

O aprendizado das RNAs é alcançado através da exposição a dados, onde o algoritmo ajusta iterativamente os pesos dos neurônios com base no erro entre o resultado esperado e o obtido pela rede, refinando assim a sua capacidade de previsão ou classificação. Este processo de ajuste é o que permite que a RNA "aprenda" a partir dos dados, tornando-se mais precisa em suas previsões ou classificações à medida que mais dados são fornecidos e mais iterações de aprendizado são realizadas.

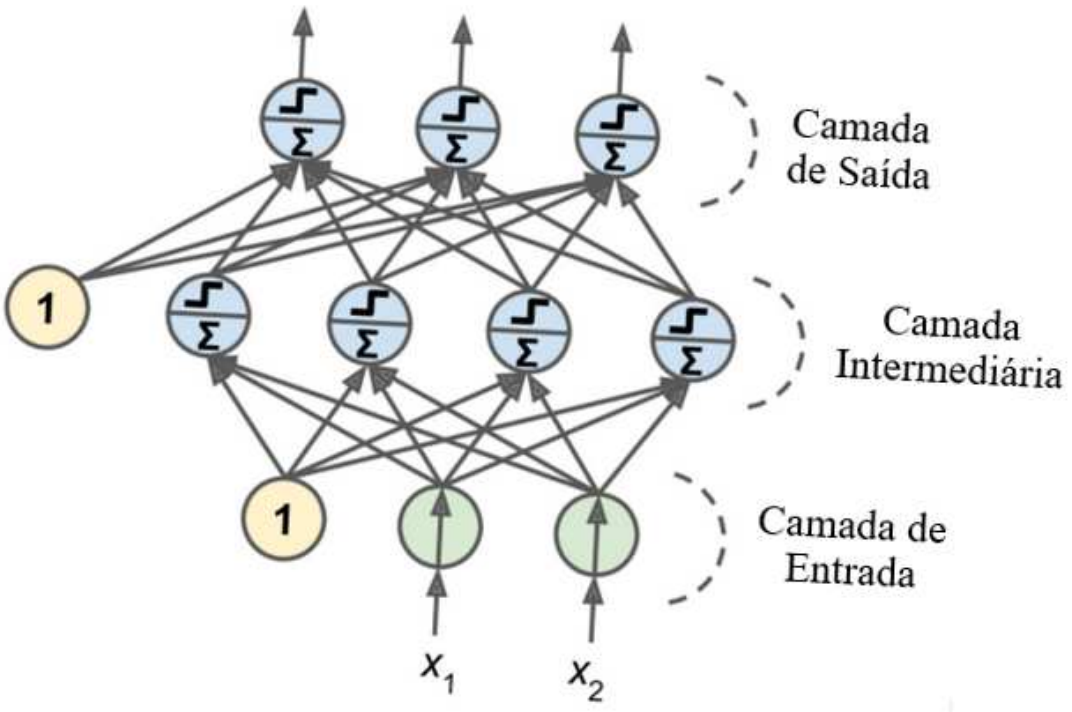


Figura 1: *Multilayer Perceptron* (adaptado de (GÉRON, 2019))

2.1.2 Aprendizado e Função de Custo

O *backpropagation*, utilizado em conjunto com o algoritmo de otimização Gradiente Descendente (GD), ajusta os pesos da rede neural conforme os gradientes derivados durante o

próprio processo de *backpropagation*. Os erros, medidos pela função de custo $J(W)$ — onde W simboliza os pesos da rede —, indicam o quanto as saídas da rede atual se desviam dos valores reais. Comumente, a função de custo emprega o Erro Médio Quadrático (EMQ), e é justamente na etapa de *backpropagatio*n que este EMQ é calculado para orientar a atualização dos pesos. Os passos da etapa de aprendizado são descritos a seguir:

- **Inicialização:** Os pesos da rede neural são inicializados de forma aleatória. É de extrema importância que os pesos sejam iniciados dessa forma, pois, caso sejam iniciados todos com 0, o algoritmo de *backpropagation* entenderá que todos os neurônios exercem a mesma influência, assim, é preciso que os pesos sejam inicializados de forma randômica para que possam quebrar a simetria.
- **Cálculo do gradiente:** Após a inicialização dos pesos, é iniciada a etapa de Cálculo dos Gradientes, na qual é calculado o gradiente da função de custo em relação aos parâmetros do modelo. Assim, são computadas as derivadas parciais da função de custo $J(W)$ em relação a cada um dos pesos da matriz W.
- **Atualização dos parâmetros:** Após a etapa de inicialização, segue-se o cálculo dos gradientes. Esta fase é essencial para o ajuste dos pesos, pois envolve o cálculo das derivadas parciais da função de custo em relação a cada peso da rede. O objetivo é determinar como a alteração de cada peso afeta o erro total, possibilitando ajustes precisos dos parâmetros para a minimização da função de custo.
- **Convergência:** O processo de *backpropagation* é iterativo, alternando entre previsões (propagação para frente) e atualizações de parâmetros (propagação para trás), buscando a otimização da rede neural. O critério de convergência pode ser definido por um limiar de mínimo erro desejado ou um número máximo de iterações. Este mecanismo garante que a rede se ajuste até alcançar uma solução ótima ou até que as melhorias se tornem insignificantes, indicando que o treinamento pode ser concluído.

Ao final, obtêm-se um novo modelo com pesos atualizados, que podem corresponder ao valor ótimo para a função de custo, ou seja, os parâmetros W da rede neural serão valores que farão com que a função custo possua o valor mínimo dado um vetor de dados X, o que permite que o modelo gerado seja mais fiel à realidade apresentada nos dados.

2.1.3 Arquiteturas de Redes Neurais

O processamento de texto por Redes Neurais Artificiais (RNAs) enfrenta desafios devido à natureza sequencial e ao tamanho variável das linguagens, dificultando a captura de relações entre elementos distantes na sequência e limitando a entrada a um tamanho fixo. Isso pode resultar na perda de contexto crucial, onde palavras e frases podem mudar de significado com base nas palavras ao redor. Para superar essas limitações, foram desenvolvidas arquiteturas como as redes neurais recorrentes (RNNs), *Long Short-Term Memory* (LSTM) e o *Transformer*. Esses modelos conseguem processar sequências de dados de comprimento variável, mantendo o contexto de partes anteriores do texto para uso em processamentos futuros, o que é essencial para tarefas como a tradução de texto. Essas inovações destacam a importância de preservar o contexto no processamento de linguagem natural, permitindo uma compreensão mais profunda e traduções de maior qualidade.

2.1.3.1 Redes Neurais Recorrentes

Redes Neurais Recorrentes (*Recurrent Neural Networks*, RNNs) (JURAFSKY; MARTIN, 2020) são modelos de redes neurais que se caracterizam por possuírem conexões cíclicas entre os neurônios de uma mesma camada. Essa estrutura cria um *loop*, permitindo que a rede mantenha uma espécie de "memória" sobre as informações anteriores, como ilustrado na Figura 2. Nesta representação, cada variável tem um papel específico:

- X_t: Representa a entrada da rede no tempo t, que pode ser, por exemplo, um elemento de uma sequência de palavras em uma tarefa de processamento de linguagem natural;
- h_t: É o estado oculto no tempo t, que funciona como uma "memória" da rede. Ele é calculado com base no estado oculto anterior h_{t-1} e a entrada atual X_t, permitindo que a RNN processe informações sequenciais de maneira eficaz;
- Y_t: Denota a saída da rede no tempo t, que pode ser a classificação de uma palavra ou previsão do próximo elemento em uma sequência.

A seta cíclica em torno de h_t destaca a natureza recorrente da rede, enfatizando que o estado oculto no tempo t é influenciado pelo estado no tempo t-1. Este mecanismo possibilita que o contexto seja propagado através do tempo e influencie os cálculos subsequentes, permitindo à rede capturar dependências temporais em dados sequenciais.

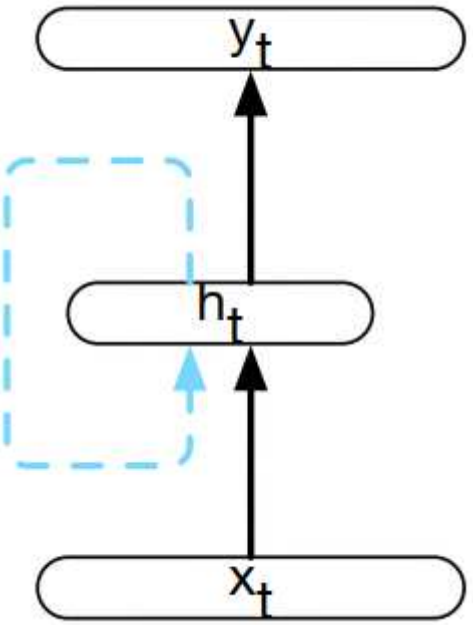


Figura 2: Representação de uma rede recorrente simples (adaptado de (JURAFSKY; MARTIN, 2020))

Logo, com essa nova proposta de arquitetura, têm-se dois tipos de pesos para a rede, uma matriz *W* referente aos pesos que já observamos nas RNAs padrão, e uma matriz *U* que é uma matriz que conecta passos temporais anteriores à camada intermediária atual. Assim, o processo de aprendizado utilizando o gradiente descendente em conjunto com o *backpropagation* envolve o uso das duas matrizes para otimizar a função de custo.

O processo de treinamento difere um pouco do treinamento das redes neurais *feedforward*. Como as RNNs possuem conexões de passos anteriores que afetam previsões futuras, é necessário que o algoritmo de otimização leve em consideração a influência desses valores no erro. Então, o algoritmo *backpropagation* através do tempo (*BackPropagation Through Time*, BPTT) foi proposto por Werbos (1990) com o intuito de endereçar este problema.

A solução com o BPTT consiste em realizar o passo de inferência calculando h_t e Y_t, salvando os valores das camadas intermediárias que serão utilizados nos próximos passos de tempo. Em seguida, faz o processo de forma reversa, calculando o gradiente e o erro nas camadas intermediárias para que sejam utilizados em etapas temporais anteriores. No entanto, além de ser um processo de treinamento demorado (pois precisa calcular o erro em cada uma das etapas temporais), as RNNs também têm um problema conhecido como *desaparecimento/explosão dos gradientes.*

No desaparecimento de gradientes, o processo de treinamento das RNNs faz com que os gradientes sejam reduzidos a zero ou fiquem muito próximos, que elimina a influência de etapas anteriores na previsão, transformando-se em uma rede neural *feedforward*. Já na explosão de gradientes, o mesmo processo faz com que os valores dos gradientes assumam valores muito grandes, o que dilui a influência dos pesos da rede neural. Então, para solucionar esse tipo de problema uma nova arquitetura foi proposta: *Long Short Term Memory* (LSTM).

2.1.3.2 Long Short Term Memory

Para resolver o problema mencionado na seção anterior, as redes neurais *Long Short Term Memory* (LSTM) (HOCHREITER; SCHMIDHUBER, 1997) adotam uma estratégia dividida em duas partes: (i) remover informações que não serão utilizadas mais pelo contexto; e (ii) adicionar informações que podem ser utilizadas posteriormente. O processo de aprendizado das LSTMs envolve aprender justamente como realizar essa estratégia de forma eficiente. Assim, foram propostos dois novos componentes: uma nova camada contextual, e também a adição de portões (ou *gates*) que serão responsáveis por gerenciar o controle de informações na rede neural.

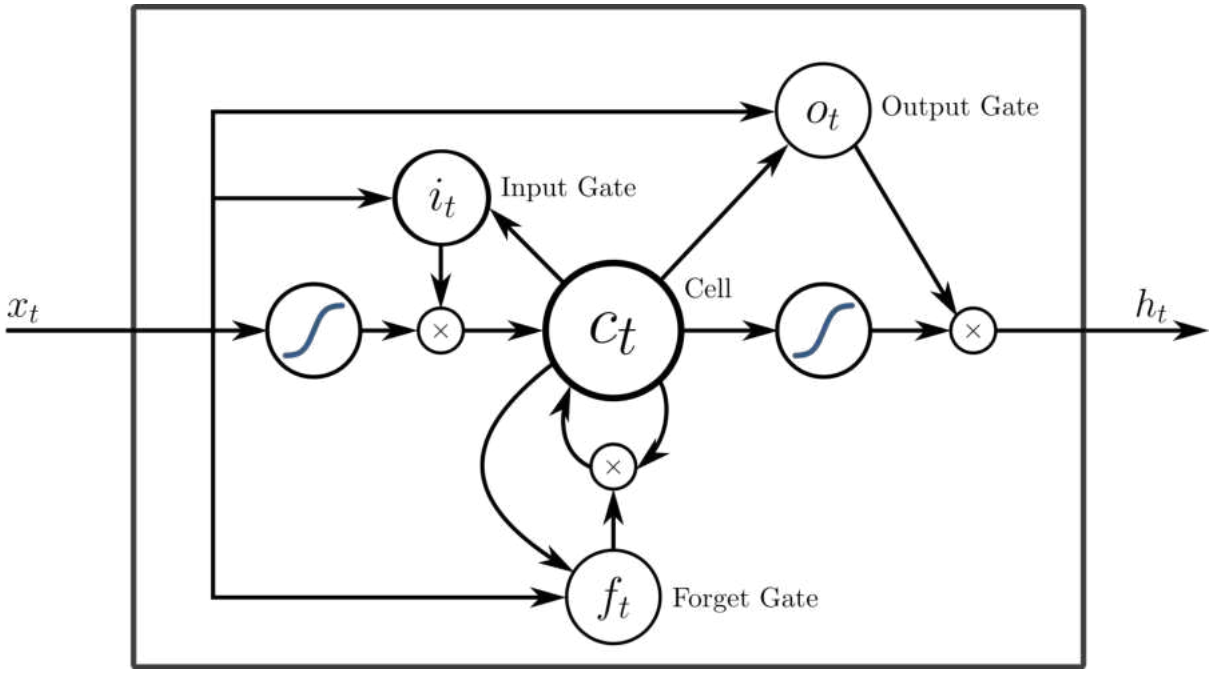


Figura 3: Diagrama de representação simplificado das LSTMs (adaptado de (JURAFSKY; MARTIN, 2020))

Conforme pode ser observado na Figura 3, a arquitetura das redes LSTMs é mais complexa que as RNNs. As LSTMs têm três tipos de portas: Porta de entrada (*Input Gate*), a porta de esquecimento (*Forget Gate)*, e a porta de saída (*Output Gate*). A porta de entrada decide quais informações novas serão adicionadas à memória da célula; a porta de esquecimento determina quais informações serão descartadas da memória, permitindo à LSTM esquecer informações não relevantes ou obsoletas; e a porta de saída define quais informações da memória da célula serão utilizadas para gerar a saída atual da rede.

Além disso, a arquitetura das LSTMs é projetada para combater o problema dos gradientes que desaparecem, comum em redes neurais recorrentes tradicionais (RNNs). Através da regulação cuidadosa do fluxo de informação permitida por suas portas e estado da célula, as LSTMs mantêm a estabilidade do gradiente ao longo do tempo, permitindo o treinamento eficaz em sequências de dados extensas. Esta característica é essencial para a aplicação de LSTMs em campos como processamento de linguagem natural e análise de séries temporais, onde a capacidade de aprender dependências de longo alcance é crucial.

Os *Transformers*, introduzidos por Vaswani et al. (2017), representam uma evolução do modelo de processamento sequencial, eliminando a necessidade de operações sequenciais e permitindo o processamento paralelo de dados. Isso é alcançado através do mecanismo de atenção, que pondera a importância relativa de diferentes partes da entrada para a produção da saída. Enquanto as LSTMs abordam as limitações das RNNs no que diz respeito ao aprendizado de dependências de longo prazo de maneira eficaz, os *Transformers* oferecem uma solução escalável para o processamento de sequências extremamente longas, marcando outro avanço significativo na modelagem de sequências e no campo da Inteligência Artificial.

2.1.3.3 Self-Attention: Transformers

Nesta seção, apresentaremos os *Transformers* e o mecanismo de *Atenção*, que são essenciais para o desenvolvimento de modelos de linguagem de grande escala (*Large Language Models*, LLM). A ideia principal dessa arquitetura é a utilização de camadas de Auto-Atenção, que serão responsáveis por extrair a relação entre as palavras de uma sentença. Porém, primeiro precisa-se introduzir como funciona o mecanismo de Atenção.

A função de Atenção pode ser entendida como o mapeamento de uma *busca* (*query*) e um conjunto de pares *chave-valor* (*key-value*) à uma *saída* (*output*), onde a busca, as chaves, os valores e saída são vetores (VASWANI et al., 2017). A busca é o vetor que representa o que o modelo está procurando. Já as chaves representam as diferentes partes da entrada. Os

valores são representações vetoriais da importância das partes da entrada. E a saída é um vetor que representa a relação entre a saída e as chaves.

Por exemplo, se a sentença “O rato roeu a roupa do rei de Roma” estivesse sendo analisada por um modelo de processamento de linguagem natural com mecanismo de atenção, a “consulta” poderia ser um vetor que representa o contexto atual no qual queremos focar, como a palavra “roeu”. As “chaves” seriam as representações vetoriais de todas as palavras da sentença, e os “valores” seriam representações vetoriais que codificam as informações a serem recuperadas, aqui também derivadas das palavras da sentença. A saída do mecanismo de atenção seria um vetor que representa o contexto agregado de todo o texto ao redor da palavra “roeu”, ponderado pela relevância calculada entre a “consulta” e as “chaves”.

O avanço introduzido pelos *Transformers* reside na sua capacidade de aplicar o mecanismo de Auto-Atenção em múltiplas camadas. Isso permite ao modelo não apenas identificar a relevância de diferentes palavras em relação à consulta, mas também compreender as interações complexas entre as palavras ao longo de uma sentença ou documento. Essa habilidade de capturar nuances contextuais transforma significativamente a eficácia dos modelos de PLN, permitindo avanços significativos em tarefas como tradução automática, geração de texto e classificação de texto, que é o foco deste trabalho. Mais informações sobre a arquitetura de *Self-Attention: Transformer*s estão apresentadas no Apêndice A.

2.2 REFINAMENTO DE MODELOS (FINE-TUNING)

O refinamento de modelos, ajuste fino, ou *fine-tuning*, em aprendizado de máquina é um processo que envolve ajustar um modelo pré-treinado para que ele funcione bem em uma nova tarefa semelhante (HOWARD; RUDDER, 2018). Ao se iniciar um projeto de aprendizado de máquina, muitas vezes não é necessário criar um novo modelo do zero, principalmente quando existem modelos pré-treinados disponíveis que foram treinados em conjuntos de dados grandes e abrangentes. O uso desses modelos e ajustá-los para uma tarefa específica tem a vantagem de economizar tempo e recursos de computação, pois evita treinar um modelo complexo inteiro do zero.

O processo de *fine-tuning* geralmente envolve alguns passos. Primeiramente, as camadas superiores do modelo são “congeladas” de forma que seu aprendizado prévio não seja perdido durante o treinamento nas novas tarefas. Então, as últimas camadas do modelo são treinadas no novo conjunto de dados. Desta forma, o modelo pode “aprender”

características que são específicas para a nova tarefa, enquanto ainda retém o conhecimento valioso que foi aprendido anteriormente. A taxa de aprendizado (*learning rate*) também é geralmente reduzida durante o *fine-tuning* para garantir que as mudanças feitas no modelo sejam sutis e não "destruam" o aprendizado prévio.

O *fine-tuning* é uma técnica amplamente utilizada no aprendizado profundo e, em particular, em tarefas de visão computacional. Ele permite que os praticantes de aprendizado de máquina aproveitem modelos pré-treinados que foram treinados em conjuntos de dados massivos, como o ImageNet[1], e os ajustes para funcionar bem em uma tarefa de visão computacional específica. Além disso, essa técnica também tem sido eficaz em outros domínios, como o processamento de linguagem natural, onde modelos como BERT (DEVLIN et al., 2019) e GPT-3 (BROWN et al., 2020) podem ser refinados para tarefas específicas como tradução automática, compreensão de texto e geração de texto. O processo de refinamento de modelos de linguagem em grande escala pode ser feito de três formas: semi-supervisionado, supervisionado e aprendizado por reforço.

No aprendizado semi-supervisionado, são passadas sentenças para o modelo de linguagem e ele tenta prever a próxima palavra (ou *token*). Por exemplo, ao passar a frase "Escute o seu" o modelo pode completar com "coração". Porém, se completar com uma palavra diferente, a função de erro deve refletir essa diferença e assim, o modelo poderá ajustar os pesos para realizar a tarefa de geração da palavra com melhor eficiência.

No aprendizado supervisionado, são utilizados *prompts* para realizar o refinamento do modelo. Por exemplo, para a tarefa de resposta da pergunta "Quem foi o primeiro presidente do Brasil?", que ainda é um problema de geração dos próximos *tokens*, o modelo de linguagem poderia simplesmente repetir outras perguntas como "Quem foi o segundo presidente do Brasil?", então são gerados *prompts template* no formato de perguntas e respostas da forma: "Responda a seguinte pergunta `Q: {Question} A:{Answer}`", na qual `{Question}` e `{Answer}` são *placeholders* para os valores presentes no conjunto de dados.

Por fim, o aprendizado por reforço consiste em treinar um outro modelo que irá criar pontuações para as gerações de texto do modelo dado uma determinada entrada. Por exemplo, para a pergunta "Quem foi o primeiro presidente do Brasil?", o modelo pode gerar três respostas: "Marechal Deodoro da Fonseca"; "Juscelino Kubitschek" e "Getúlio Vargas".

[1] https://www.image-net.org

Assim, o modelo que gera as pontuações pode criar pontuações para cada um dos resultados e os anotadores irão avaliar se os resultados realmente satisfazem o que foi pedido.

No entanto, em alguns domínios do conhecimento, o acesso a dados é escasso e muitas vezes a sua aquisição é muito cara, assim, surgiram outros tipos de técnicas que tentam mitigar esse tipo de problema conforme abordado na seção seguinte (seção 2.3).

2.3 FEW-SHOT LEARNING

O notável sucesso dos modelos modernos de aprendizado de máquina tem sido em grande parte impulsionado por sua capacidade de aprender a partir de vastas quantidades de dados rotulados. No entanto, essa necessidade por grande quantidade de dados apresenta um desafio significativo, particularmente em domínios onde a coleta de dados é cara, demorada ou simplesmente não viável. O aprendizado *few-shot* emergiu como uma abordagem promissora para enfrentar essa limitação, possibilitando que os modelos aprendam efetivamente a partir de apenas um punhado de exemplos (FEI-FEI; FERGUS; PERONA, 2006).

A chave para o aprendizado *few-shot* reside no conceito de meta-aprendizado, na qual o modelo aprende a aprender (FINN; ABBEEL; LEVINE, 2017). Em vez de treinar em uma única tarefa, modelos de meta-aprendizado são expostos a um conjunto diversificado de tarefas relacionadas durante o processo de treinamento. Essa exposição permite que o modelo desenvolva um entendimento geral de como aprender de forma eficiente, que pode então ser aplicado a novas tarefas com dados limitados. Abordagens proeminentes de aprendizado *few-shot*, tais como redes prototípicas (SNELL; SWERSKY; ZEMEL, 2017) e meta-aprendizado agnóstico de modelo (*Model Agnostic Meta Learning*, MAML) (FINN; ABBEEL; LEVINE, 2017), demonstraram resultados impressionantes em várias aplicações, incluindo reconhecimento de imagem, processamento de linguagem natural e até mesmo descoberta de drogas (ALTAE-TRAN et al., 2017).

Recentemente, avanços no aprendizado *few-shot* têm incorporado técnicas de aprendizado por transferência, onde conhecimento adquirido em tarefas anteriores é aplicado a novas tarefas com poucos exemplos (PAN; YANG, 2010). Isso tem sido particularmente útil em domínios específicos, como a robótica, onde dados de treinamento podem ser extremamente caros ou difíceis de obter (YIN et al., 2020).

A capacidade de aprender a partir de dados limitados tem implicações de grande alcance para o futuro do aprendizado de máquina e da inteligência artificial. Em domínios

onde a coleta de dados é um gargalo significativo, o aprendizado *few-shot* oferece uma solução promissora para preencher a lacuna entre a crescente demanda por sistemas inteligentes e as restrições práticas da disponibilidade de dados. Ao capacitar modelos para aprender de forma eficiente a partir de informações mínimas, o aprendizado *few-shot* tem o potencial de revolucionar como abordamos problemas em campos que vão desde a saúde até sistemas autônomos.

Uma inovação recente na implementação do aprendizado *few-shot* envolve o uso de *prompts* em LLMs. *Prompts* são instruções ou estímulos textuais fornecidos a modelos de linguagem de grande escala (LLMs) para induzir uma resposta específica ou para direcionar o modelo a executar uma tarefa particular. Esta técnica, conhecida como aprendizado com exemplos no contexto (*in-context learning*), permite que LLMs, como o GPT-3, realizem tarefas de aprendizado *few-shot* sem a necessidade de ajustes finos específicos do modelo. Por exemplo, para uma tarefa de Question-Answer o modelo pode receber um *prompt*: "Responda a seguinte pergunta: `Q: <question> R: <response>`". Assim o modelo será treinado para que a probabilidade das palavras sejam as mais próximas possíveis do valor real.

Ao fornecer exemplos de tarefas juntamente com *prompts* de texto, os LLMs podem gerar respostas precisas para novas instâncias de tarefas com uma quantidade mínima de exemplos de treinamento. Essa abordagem tem mostrado grande promessa, especialmente em tarefas de processamento de linguagem natural, onde os *prompts* cuidadosamente projetados podem orientar o modelo a entender e executar a tarefa desejada com eficácia (BROWN et al., 2020).

2.4 WORD EMBEDDINGS

Word Embedding é um técnica essencial no campo de Processamento de Linguagem Natural, desempenhando um papel fundamental na representação numérica de palavras e documentos, em um espaço de dimensionalidade reduzida. Ao converter informações linguísticas em vetores numéricos, *word embeddings* possibilitam que modelos de aprendizado de máquina compreendam e processem linguagem humana de maneira eficaz. Uma das principais razões para sua utilização é superar as limitações das representações tradicionais, como *one-hot encoding*, capturando relações semânticas e contextuais de forma mais precisa.

Uma das abordagens de representação vetorial de termos é o *One-hot Encoding*, que atribui um vetor de dimensionalidade igual ao tamanho do vocabulário, no qual, o valor 1 é

associado à dimensão correspondente à palavra e o valor 0 nas demais dimensões (correspondente às demais palavras do vocabulário). Nesta representação, a vetorização de um texto, composto por várias palavras, pode ser feita, por exemplo, atribuindo um vetor de zeros e uns com o mesmo tamanho do vocabulário, no qual, a presença da palavra é indicada pelo valor 1 na dimensão correspondente à mesma e a ausência da palavra é indicada pelo valor 0. Por exemplo, conforme a Tabela 1, é possível observar o vetor que representa as sentenças “O cachorro está latindo” e “O gato está miando”.

Tabela 1: Representação das sentenças sem redução de dimensionalidade

	o	cachorro	está	latindo	gato	miando
“o cachorro está latindo”	1	1	1	1	0	0
“o gato está miando”	1	0	1	0	1	1

Em contraste, a representação vetorial de *One-hot Encoding* que utiliza vetores esparsos para representação das palavras, temos abordagens, como o Word2Vec (MIKOLOV et al., 2013) e o GloVe (PENNINGTON; SOCHER; MANNING, 2014) que utilizam vetores densos, com redução de dimensionalidade. Esses vetores densos conseguem encapsular informações semânticas complexas e relações contextuais em um espaço vetorial compacto.

O Word2Vec utiliza redes neurais para aprender representações vetoriais a partir de grandes conjuntos de dados textuais. Funciona através de dois modelos arquiteturais principais: o CBOW (*Continuous Bag of Words*) e o *Skip-gram*. O modelo CBOW prediz uma palavra atual com base no contexto de palavras ao redor, enquanto o *Skip-gram* faz o inverso, predizendo o contexto a partir de uma palavra atual. Ambos os modelos resultam em vetores de termos que capturam relações semânticas e sintáticas: palavras com significados semelhantes ou que frequentemente aparecem em contextos similares são posicionadas próximas umas das outras no espaço vetorial (HENDERSON; POPA, 2016).

A transição de vetores esparsos para densos, possibilitada por técnicas como Word2Vec e GloVe, representou um salto qualitativo na capacidade de modelos de PLN de compreender e manipular a linguagem natural. Os vetores densos contêm uma riqueza de informação semântica e sintática em um espaço vetorial com dimensões muito menores do que seria possível com representações esparsas. Esta eficiência e profundidade de informação facilitam uma variedade de tarefas de PLN, desde a classificação de textos até a tradução automática e a geração de texto (CHITTY-VENKATA et al., 2022; THOYYIBAH et al., 2023).

Na classificação de textos e outras tarefas de PLN, os *embeddings* são utilizados como camada de entrada dos modelos, especialmente os baseados em *transformers* (VASWANI et al., 2017), que são projetados para capturar dependências de longo alcance entre palavras em um texto, conforme visto na seção 2.1. Os *transformers* utilizam os *embeddings* para entender a estrutura e o significado do texto, aplicando o mecanismo de *Atenção* para ponderar diferentes partes do *input* de forma a otimizar a tarefa de interesse, como a classificação, tradução ou geração de texto.

A introdução do BERT (*Bidirectional Encoder Representations from Transformers*) por Devlin et al. (2019) marcou um ponto de virada significativo no uso de *embeddings* em Processamento de Linguagem Natural (PLN). Diferentemente dos modelos anteriores que tratavam os textos de maneira sequencial, o BERT é capaz de entender o contexto de cada palavra em relação a todas as outras palavras na sentença, de forma bidirecional. Isso é alcançado através do treinamento prévio em grandes *corpora* de texto, utilizando tarefas como previsão de palavras ocultas e entendimento de relação entre sentenças.

Como resultado, os *embeddings* gerados pelo BERT capturam nuances semânticas complexas e relações contextuais, proporcionando uma base rica para uma ampla gama de tarefas de PLN, desde classificação de texto até perguntas e respostas, com um desempenho notavelmente aprimorado em comparação com as abordagens anteriores. Este avanço permitiu uma compreensão mais profunda e natural da linguagem humana, pavimentando o caminho para sistemas de IA mais intuitivos e eficazes em entender e processar texto.

Seguindo a evolução trazida pelo BERT, os *Sentence Transformers* (REIMERS; GUREVYCH, 2019) surgiram como uma adaptação para superar um dos desafios associados ao BERT: a geração eficiente de *embeddings* para sentenças inteiras. Os *Sentence Transformers* modificam a arquitetura padrão do BERT para produzir *embeddings* que representam não apenas palavras individuais, mas sim o significado completo de sentenças ou parágrafos. Treinando o modelo em tarefas específicas que exigem a compreensão do significado completo das sentenças, como similaridade semântica ou classificação de parágrafos, os *Sentence Transformers* geram *embeddings* de alta qualidade. Estes podem ser utilizados de forma eficaz em tarefas de PLN que requerem a compreensão do conteúdo semântico completo das sentenças, como agrupamento de texto, busca semântica, e sistemas de recomendação, tornando-os uma ferramenta valiosa para avançar no estado da arte em PLN.

2.5 SETFIT

Nesta seção, vamos abordar a metodologia de treinamento *SetFit* (*Sentence Transformer Fine-tuning*), que foi utilizada para a realização dos experimentos de refinamento dos modelos neste trabalho. A principal diferença entre o *SetFit* e outras abordagens de refinamento é que não é necessário utilizar *prompts* para o treinamento como outras técnicas de *fine-tuning* descritas na seção 2.2. Essa abordagem realiza o *fine-tuning* (ou refinamento) de modelos de *Sentence Transformers* (ST) utilizando um pequeno número de exemplos para criar pares de textos a partir da técnica de aprendizado contrastivo (KOCH et al., 2015).

A abordagem do *SetFit* divide-se em duas fases, como pode ser observado na Figura 4: (i) Refinamento do modelo de codificação dos vetores (*ST Fine tuning*) e (ii) o Treinamento da camada de classificação (*Classification head training*).

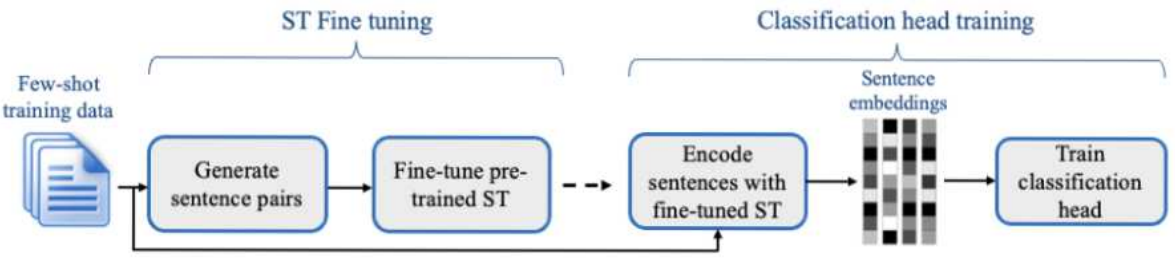


Figura 4: Diagrama representando os passos do SeFit (TUNSTALL et al., 2022)

Devido à baixa quantidade de dados para o treinamento em um cenário de *few-shot*, os autores decidiram utilizar uma abordagem de treinamento contrastiva, geralmente utilizada para similaridade de imagens (KOCH et al., 2015), na qual são gerados pares de textos. Assim, a descrição formal do algoritmo baseia-se na utilização de um pequeno conjunto de dados com K exemplos rotulados, tal que $D = \{(x_i, y_i)\}$, onde x_i representa uma sentença com índice i e y_i representa o rótulo associado à sentença.

O algoritmo de geração de dados para aprendizado contrastivo pode ser formalizado como a operação que, para cada $x_i \in S$, gera duas triplas:

- uma tripla positiva (x_i, x_j, 1), onde x_j é escolhido de forma aleatória tal que $x_i \neq x_j$ e $y_i = y_j$;
- uma tripla negativa (x_i, x_j, 0), onde x_j é escolhido de forma aleatória tal que $x_i \neq x_j$ e $y_i \neq y_j$.

Além disso, temos um outro hiperparâmetro que controla a geração de triplas definido por *R*. Este hiperparâmetro irá definir quantas vezes o processo de geração de triplas ocorrerá. Por exemplo, se *R* = 20, então para cada sentença serão gerados 20 triplas positivas e 20 triplas negativas. Por conseguinte, se tivermos um conjunto de dados com 16 exemplos, 8 exemplos da classe *A* e 8 exemplos da classe *B*, ao final do processo de geração de dados teremos 2**R***K* = 2*20*16 = 640 triplas que serão utilizadas no processo de refinamento do modelo de linguagem.

Após a geração dos pares de sentenças, ocorre o *fine-tuning* do ST, que tem como objetivo aproximar sentenças que são semanticamente semelhantes, e afastar sentenças que são semanticamente distintas no espaço vetorial. Em seguida, ocorre o treinamento da camada de classificação com o *dataset* original, assim temos uma sentença, por exemplo, que serve de entrada para o modelo de *embedding* que foi refinado, e em seguida passada para a camada de classificação.

A vantagem dessa técnica é que não são necessários *prompts* para realizar o refinamento do modelo de linguagem, que, neste caso, foram os *Sentence Transformers*. Além disso, apresentam uma versatilidade inerente à abordagem, pois é possível apenas "trocar" o corpo do modelo de *embedding* para realizar classificações em outros idiomas ou domínios específicos. Os autores também destacam a velocidade de treinamento e inferência em relação a outras técnicas de *few-shot* como T-Few (LIU et al., 2022), ADAPET (TAM et al., 2021) e PERFECT (MAHABADI et al., 2022).

2.6 AVALIAÇÃO

No estudo e desenvolvimento de modelos de linguagem para processamento de linguagem natural, a métrica F1 assume um papel fundamental na avaliação de desempenho desses modelos. Ela baseia-se nos princípios de precisão e revocação, fornecendo um meio equilibrado de mensurar a capacidade do modelo de classificar as intenções ou classes com precisão e revocação (LIPTON; ELKAN; NARAYANASWAMY, 2014). Este equilíbrio é crucial, dada a natureza frequentemente conflitante dessas duas métricas.

Além disso, o suporte para cada classe se torna um aspecto relevante na interpretação das métricas, indicando o número de exemplos por classe que o modelo tenta prever. Esta informação é valiosa para entender a confiabilidade das métricas de precisão e revocação, especialmente, em conjuntos de dados desequilibrados onde algumas classes podem ter muito mais ou menos exemplos do que outras.

A Figura 5 mostra um diagrama de Venn utilizado para avaliar modelos de classificação binária. “Verdadeiros positivos” indicam acertos do modelo para a classe desejada. “Falsos positivos” e “Falsos negativos” representam os erros do modelo, onde ele, respectivamente, classifica incorretamente exemplos como positivos ou falha em identificar exemplos positivos. “Verdadeiros negativos” são os acertos do modelo para não pertencimento à classe desejada. Precisão e revocação são métricas derivadas desses valores, e o F1-Score é a média harmônica entre elas, fornecendo um balanço entre precisão e revocação.

A precisão, conforme ilustrada na Figura 6, é calculada pela divisão do número de verdadeiros positivos pelo total de exemplos classificados como positivos, refletindo a exatidão das classificações positivas do modelo. Já a revocação, também representada na Figura 6, é determinada pela proporção de verdadeiros positivos em relação ao total de exemplos que são de fato positivos, evidenciando a habilidade do modelo em identificar todos os exemplos relevantes dentro de uma classe específica (HASSANPOUR; BAY; LANGLOTZ, 2017).

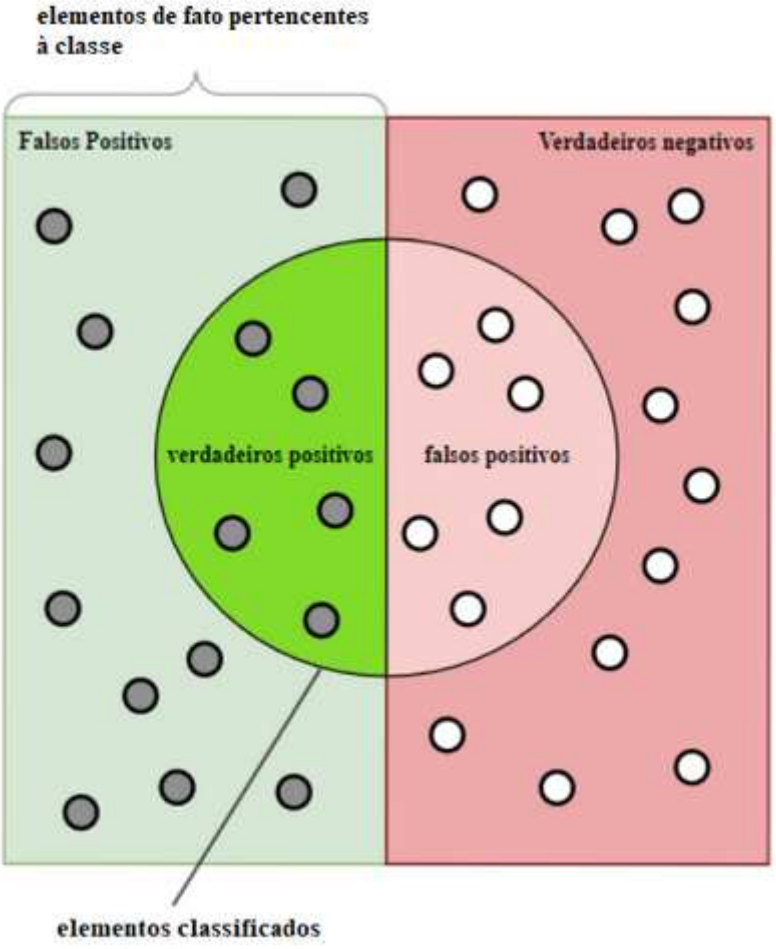


Figura 5: Diagrama de representação de Verdadeiros Positivos, Verdadeiros Negativos, Falsos Positivos e Falsos Negativos

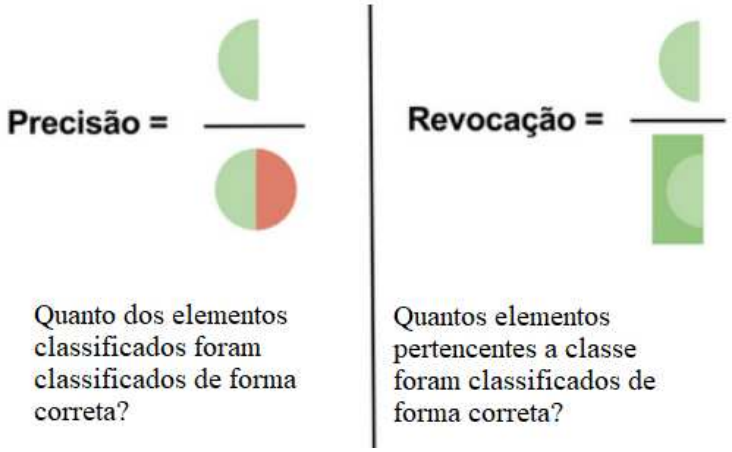


Figura 6: Diagrama de representação das métricas precisão e revocação (adaptado de (Wikimedia Commons[2]))

A precisão e a revocação frequentemente se movem em direções opostas; aumentar uma pode diminuir a outra. Esse fenômeno reflete o desafio inerente de melhorar ambas as métricas ao mesmo tempo. A métrica F1, expressa na equação 1, é a média harmônica de precisão e revocação. Ao contrário da média aritmética, a média harmônica penaliza valores extremos, promovendo um compromisso mais equilibrado entre as duas métricas. Essa propriedade é crucial em situações onde os dados não são desbalanceados. Em tais casos, um modelo poderia se inclinar para prever principalmente a classe majoritária, inflando artificialmente uma das métricas enquanto negligencia a outra. O F1-Score, então, serve como um indicativo mais robusto da performance geral do modelo, assegurando que tanto a precisão quanto a revocação são razoavelmente altas e, portanto, que o modelo é verdadeiramente eficaz.

$$F1\ score\ =\ 2\ *\ \frac{Precisão * Revocação}{Precisão + Revocação} \quad (1)$$

A Figura 5 serve como uma representação visual dos conceitos de verdadeiros positivos, verdadeiros negativos, falsos positivos e falsos negativos, e, na Figura 6, como estes se relacionam com as métricas de precisão e revocação. O uso conjunto dessas visualizações e da equação 1 facilita a compreensão da importância de uma abordagem equilibrada na avaliação e otimização de modelos de linguagem (TRIVEDI et al., 2017).

[2] https://upload.wikimedia.org/wikipedia/commons/2/26/Precisionrecall.svg

Assim, a métrica F1 não apenas atua como um critério de avaliação, mas também orienta o ajuste fino dos modelos de linguagem, incentivando melhorias que beneficiem tanto a precisão quanto a revocação. Este enfoque na métrica F1 sublinha sua relevância na busca por sistemas de processamento de linguagem natural que sejam não apenas eficientes, mas também adaptáveis e sensíveis às nuances das tarefas de classificação que enfrentam.

Para problemas de classificação *multiclasse*, como é o caso específico avaliado neste trabalho, as métricas de avaliação utilizadas podem ser agregadas para fornecer uma avaliação abrangente do desempenho do modelo, tais como: média micro (*micro average*), média macro (*macro average*) e média ponderada (*weighted average*). A agregação *micro* calcula métricas globais baseadas na soma total de verdadeiros positivos, falsos negativos e falsos positivos, sem levar em conta a *performance* em relação a cada classe individualmente. Por exemplo, no caso do F1 score, significa que todos os acertos, erros e omissões são acumulados para calcular a precisão e a revocação, antes de calcular o F1 score. A média *macro* é calculada a partir da média não ponderada dos valores da métrica específica para cada classe, oferece uma visão da uniformidade de desempenho entre as classes e destaca o impacto negativo das classes menos representadas no desempenho geral. Já, a média ponderada (*weighted*) leva em consideração o suporte de cada classe, revelando que as classes com maior número de exemplos têm um impacto mais significativo na avaliação geral do desempenho do modelo.

3 TRABALHOS RELACIONADOS

Este capítulo contemplará, inicialmente, trabalhos relacionados com a classificação de intenção de citação, mostrando a evolução das abordagens e avanços no campo. Serão abordados os trabalhos utilizando o *feature-rich random forest* (JURGENS et al., 2018), posteriormente falaremos sobre a utilização de *structural scaffolds* com o intuito de melhorar a previsão do modelo, em seguida falaremos sobre o atual estado da arte para a classificação de intenção de citações utilizando o conjunto de dados ACL-ARC que utiliza um modelo de linguagem pré-treinado. Todos os trabalhos relacionados apresentaram resultados satisfatórios na tarefa de classificação de intenção de citação, e foram utilizados como comparação com os resultados obtidos neste trabalho.

No entanto, é necessário primeiro entender o que é uma intenção de citação. Citações são utilizadas pelos autores para destacar as suas contribuições e conectar uma linha de pensamento (LATOUR, 1987). Assim, existe uma função (ou intenção) para que um autor faça referência a um trabalho predecessor, seja para construir uma linha de pensamento com base no que foi proposto no artigo citado, para discordar do que foi proposto, ou simplesmente para comparação.

Além disso, existem dois tipos de citação: as diretas e as indiretas. As citações diretas envolvem a transcrição exata de uma passagem do texto original, mantendo a mesma redação e pontuação. Devem ser usadas com moderação e apenas quando a maneira como o autor original expressa uma ideia é crucial para o argumento. Por exemplo, em um estudo sobre a influência da mídia na sociedade, uma citação direta poderia ser: *De acordo com Smith (2020), "a mídia não apenas reflete a realidade social, mas também a molda ativamente"*.

Por outro lado, as citações indiretas (ou paráfrases) reescrevem as ideias do autor original com palavras próprias do autor do novo texto. As citações indiretas são mais flexíveis e podem ser adaptadas para se encaixar naturalmente no fluxo do argumento, mantendo a fidelidade ao conceito original. Continuando com o exemplo anterior, uma citação indireta seria: *This is often referred to as incorporating deterministic closure (Dörre, 1993)*. Essa abordagem permite integrar conceitos de outros trabalhos de forma mais fluida e adaptável ao contexto do novo estudo.

Para este trabalho, iremos estudar apenas as citações indiretas, especificamente, a sentença que compõe a citação, não considerando a referência entre aspas em si. A intenção refere-se a uma citação específica, ou seja, a construção que precede/sucede a referência entre parênteses; dessa forma, um mesmo artigo pode ser citado com intenções diferentes. Por

exemplo, a citação: *Em nossa análise das práticas de sustentabilidade em cadeias de suprimentos globais, recorremos ao estudo de Lee e Kim (2020)...* é utilizada como base de informações relevantes. Já a citação: *Contrastando com as descobertas de (Lee & Kim, 2020) sobre a adoção...* está sendo usada para comparar o trabalho atual com o trabalho citado. Nas próximas seções, iremos destacar estratégias que foram utilizadas para construir modelos que realizassem a classificação das intenções de citação dado o texto encontrado. Todos os trabalhos destacados neste capítulo utilizam o conjunto de dados ACL-ARC.

3.1 ACL-ARC E MODELO DE CLASSIFICAÇÃO

No artigo (JURGENS et al., 2018), a investigação concentra-se nas motivações subjacentes às citações em textos acadêmicos, explorando a relação entre a estrutura das citações e fatores como a seção do artigo, o veículo de publicação e sua evolução ao longo do tempo. Além disso, teve-se como objetivo desenvolver um classificador baseado em uma variedade de variáveis derivadas das características textuais, as quais serão detalhadas posteriormente.

A primeira ação foi criar um conjunto de dados a partir de artigos no tópico de Processamento de Linguagem Natural (PLN) publicados na *Association for Computational Linguistics* (ACL). O conjunto de dados foi construído, nomeado ACL-ARC, a partir de textos de 134.127 citações distribuídas em 20.000 artigos (JURGENS et al., 2018). Em seguida, os autores utilizaram uma metodologia de anotação de parte dos dados, a partir de uma proposta realizada em um estudo pioneiro (BIRD et al., 2008), na qual chegaram a seis classes (que são grupos que indicam a intenção que um autor teve ao citar um determinado artigo), que seriam utilizadas para agrupar as citações em grupos de intenções de citação semelhantes, distribuídas conforme a Tabela 2, totalizando 1.969 exemplos anotados. De acordo com a Tabela 2, pode-se observar que a distribuição das classes é desigual, resultando em um conjunto de dados desbalanceados.

Em seguida, para alimentar o modelo de classificação, os autores criaram variáveis que se distribuíram em tipos diferentes: 1) variáveis estruturais que iriam informar onde a citação se encontra no artigo; 2) variáveis lexicais e gramaticais que indicam como a citação foi descrita; 3) variáveis de campo que apontam o veículo de publicação ou outra informação externa; 4) variáveis de usabilidade (uso) que informam como a referência é utilizada no trabalho científico.

Tabela 2: Distribuição das classes de intenção de citação no *dataset* anotado

Classe	**Número de Citações**
BACKGROUND	1.021
USES	365
COMPARES OR CONTRASTS	344
MOTIVATION	98
CONTINUATION	73
FUTURE	68

Fonte: adaptado de (JURGENS et al., 2018)

Segundo Jurgens et al. (2018) todos os modelos de classificação utilizados foram Florestas Aleatórias (FERNÁNDEZ-DELGADO et al., 2014) pois são menos propensas a sofrer com *overfit* mesmo com um número grande de variáveis. Além disso, foi aplicada uma técnica de *grid search* para descobrir os melhores hiperparâmetros. A técnica de *grid search* consiste em definir diversas listas de valores possíveis para os hiperparâmetros do modelo e combiná-los em diversas execuções com o objetivo de encontrar a melhor configuração para os dados de treinamento.

Além disso, conforme citado anteriormente, o conjunto de dados está desbalanceado, assim, foi utilizada uma técnica de geração de exemplos sintéticos para tentar equilibrar a distribuição das classes. Os autores utilizaram a técnica SMOTE (CHAWLA et al., 2002). O modelo foi implementado utilizando a biblioteca SciKit-learn[3] e a vetorização das citações foi feita utilizando o modelo pré-treinado GloVe (PENNINGTON; SOCHER; MANNING, 2014) de dimensão 300.

3.2 MULTICLASSIFICADORES

O objetivo do trabalho (COHAN et al., 2019) é utilizar uma arquitetura de modelagem diferente das abordagens utilizadas anteriormente (JURGENS et al., 2018; PETERS et al., 2018) para criar um modelo de classificação de intenção de citação conhecido como *Structural Scaffolds* (SWAYAMDIPTA et al., 2018). Segundo os autores, todas as informações necessárias para classificação podem ser extraídas diretamente dos dados, o que tornaria desnecessária a utilização de variáveis com informações externas como o número da

[3] https://scikit-learn.org

seção que a citação foi encontrada ou diferença em anos das datas de publicação do artigo citante e o citado, conforme observamos nos modelos projetados por Jurgens et al. (2018).

Paralelamente à proposta de arquitetura, também foi criado o conjunto de dados SciCite (COHAN et al., 2019) para superar as limitações dos conjuntos de dados existentes que, muitas vezes, são muito específicos de um domínio ou contém classes de intenção de citação muito detalhadas e raras. Os autores do artigo extraíram um amplo e variado conjunto de citações de artigos científicos do *corpus* da *Semantic Scholar* (FRICKE, 2018), que abrange áreas gerais de ciência da computação e medicina. Com a ajuda da plataforma de *crowdsourcing Figure Eight*[4], anotadores classificaram as intenções das citações em classes concisas e mais generalizáveis, resultando em um *dataset* significativamente maior que os anteriores, com mais de cinco vezes o tamanho do conjunto de dados ACL-ARC e abrangendo múltiplos domínios científicos. A intenção foi criar um recurso que seja não só grande em volume, mas também mais representativo da literatura científica em geral, para facilitar a análise automatizada de publicações científicas e a navegação por tópicos de pesquisa.

A ideia do *framework* de *Structural Scaffolds* é que são utilizados diversos classificadores, sendo um principal e classificadores auxiliares. No artigo de Cohan et al. (2019), os autores utilizam a intenção de citação para a classificação principal, e dois outros classificadores secundários: 1) previsão do título da seção em que a citação foi retirada; 2) previsão se a sentença precisa de citação. Isso facilita a coleta dos dados devido à natureza das duas tarefas auxiliares, na qual o título da seção sempre está presente no artigo utilizado, não necessitando de anotação dos dados e podendo ser apenas extraídos a partir do texto dos documentos.

Conforme pode ser observado na Figura 7, o núcleo da arquitetura do modelo é a rede neural *Long Short-Term Memory* (LSTM), que processa os contextos de citação. Na primeira parte do fluxo da rede neural, a entrada passa por um modelo de *embedding* para converter o texto em um vetor representativo. Em seguida, o vetor de entrada *X* passa por uma camada de LSTM bidirecional. Essa LSTM é complementada por um mecanismo de atenção, permitindo que o modelo foque nas partes mais relevantes do texto ao fazer predições. Além da tarefa principal de prever a intenção de citação, dois *scaffolds* (estruturas auxiliares) são incluídos: a previsão do título da seção onde a citação aparece e a avaliação da *citation worthiness* (se uma frase requer citação). Estas tarefas auxiliares ajudam o modelo a capturar nuances estruturais e contextuais que são importantes para a classificação das intenções.

[4] https://www.figure-eight.com

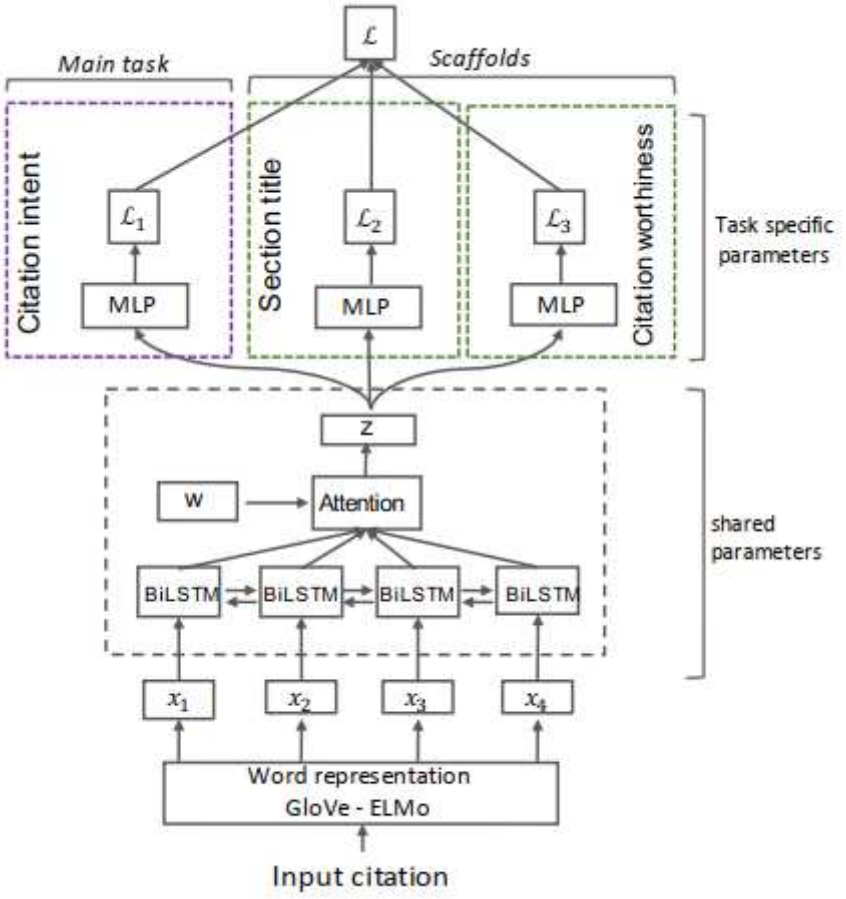


Figura 7: Representação da arquitetura de multi-tarefas (COHAN et al., 2019)

A principal vantagem dessa abordagem é que, ao contrário de trabalhos anteriores que dependiam de características linguísticas manuais ou recursos externos (JURGENS et al, 2018), o modelo proposto é capaz de aprender representações ricas e contextualizadas diretamente dos dados. As tarefas auxiliares funcionam como um mecanismo de regularização, direcionando o modelo para aprender padrões relevantes que são úteis para a tarefa principal, sem a necessidade de intervenção manual. Isso resulta em um sistema que não só atinge um novo estado da arte no momento de publicação, mas também demonstra maior generalização, capaz de lidar com uma variedade de domínios científicos e tipos de citação.

3.3 AUTO CODIFICADORES MASCARADOS VARIANTES

O objetivo desta seção é examinar a arquitetura do modelo que tem os melhores resultados entre os modelos apresentados neste capítulo, com foco na aplicação ao conjunto

de dados ACL-ARC. Será abordada a proposta de um *Autoencoder* Variacional adaptado, denominado *Variational Masked AutoEncoder* (VarMAE) (HU et al., 2022), desenvolvido para superar desafios em contextos de escassez de dados. A inovação principal do VarMAE reside na modelagem das representações dos *tokens* como distribuições latentes, o que facilita a quantificação das incertezas decorrentes das ambiguidades linguísticas.

Primeiro, é necessário entender como funciona a técnica de *Masked Autoencoders*. *Autoencoder* é um tipo de arquitetura de redes neurais que possui uma parte que codifica a entrada, reduzindo a dimensionalidade da mesma, a uma parte decodificação, que é responsável por reconstruir a entrada a partir do vetor gerado a partir da codificação. Já a técnica de *Masked Autoencoders* procura "esconder" uma parte dos *tokens* de entrada para que a rede aprenda a criar representações vetoriais ricas baseadas no contexto. Assim, nesta técnica, um percentual dos *tokens* da entrada que será passada para a rede é substituído por um *token* especial de mascaramento.

Por exemplo, na frase "O gato está dormindo", a palavra "dormindo" pode ser mascarada e substituída por "`[M]`". A sequência é passada por um codificador que irá gerar uma representação baseada no contexto das palavras que não foram mascaradas. Posteriormente, o decodificador tenta prever os *tokens* originais que foram mascarados, baseando-se nas representações geradas pelo codificador. O treinamento é guiado por uma função de perda que compara as previsões do decodificador com os *tokens* originais.

Na arquitetura do VarMAE, descrita na Figura 8, observa-se uma evolução em relação aos *Masked Autoencoders* (MAE) tradicionais, incorporando o módulo de "Aprendizado de Incerteza Contextual". Este módulo é fundamental, pois permite a codificação de incertezas nas representações latentes dos *tokens*, que são inerentes à linguagem pois os *tokens* podem aparecer em diversos contextos, gerando saídas contextualmente enriquecidas e adaptáveis.

O processo de pré-treinamento do modelo tem como objetivo a minimização da divergência de Kullback-Leibler (KL) (KULLBACK; LEIBLER, 1951). A divergência de KL atua como uma régua que mede quão bem as suposições iniciais sobre os dados se mantêm após o modelo ser exposto a dados reais. Esse processo não apenas regulariza a previsão dos *tokens*, mas também garante que as representações latentes mantenham consistência probabilística com a distribuição *a priori*, sendo crucial para a adaptação a novos domínios.

Ao utilizar essa técnica de treinamento de modelos de linguagem para o treinamento de modelos em áreas com pouca quantidade de dados, foi possível obter resultados satisfatórios, inclusive na utilização na identificação de intenção de citação, na qual obteve-se resultados melhores dos que foram obtidos anteriormente. Portanto, o VarMAE representa um

avanço significativo no desenvolvimento de modelos de linguagem adaptáveis a contextos de dados limitados.

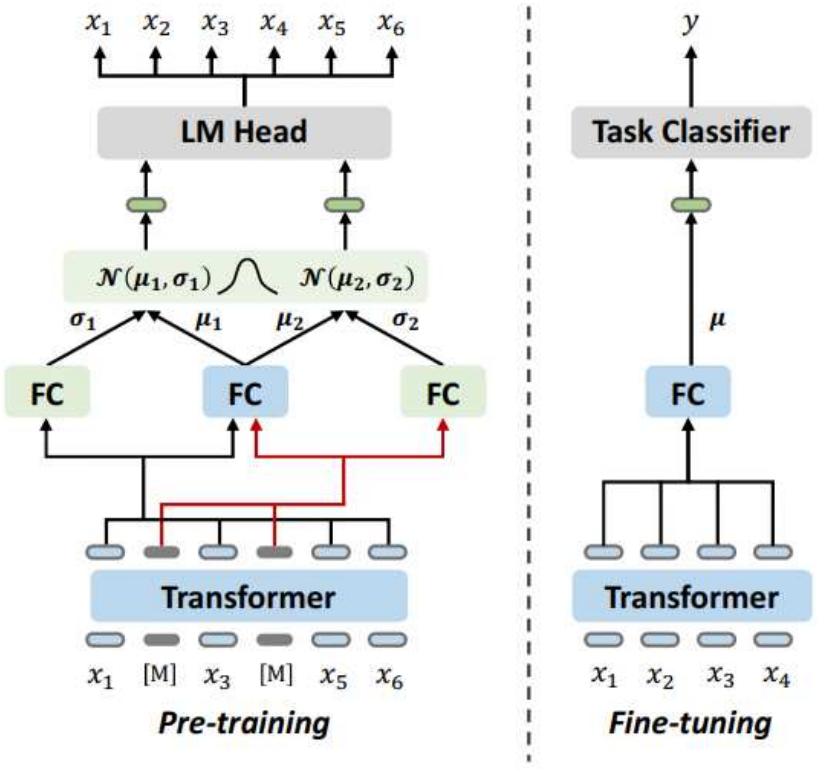


Figura 8: Arquitetura da VarMAE (HU et al., 2022)

3.4 COMPARATIVO DAS ABORDAGENS

As três abordagens apresentam resultados significativos na classificação de intenção de citação utilizando abordagens distintas, variando em complexidade de implementação, conforme pode ser observado na Tabela 3. No entanto, o trabalho (JURGENS et al., 2018) (seção 3.1), foca em agregar variáveis derivadas de informações que podem ser extraídas do contexto dos artigos analisados (como diferença entre os anos de publicação), já os artigos (COHAN et al., 2019) (seção 3.2) e (HU et al., 2022) (seção 3.3) utilizam abordagens que focam apenas nas relações semânticas e sintáticas presentes no texto utilizado para o treinamento.

No entanto, os trabalhos (COHAN et al., 2019) e (HU et al., 2022) possuem arquiteturas bem mais complexas do que a utilizada no trabalho (JURGENS et al., 2018). O trabalho (JURGENS et al., 2018) procura utilizar um modelo de aprendizado de máquina mais tradicional, como a Floresta Aleatória (*Random Forest*), já os dois outros trabalhos

utilizam arquiteturas de redes neurais complexas como LSTMs para tentar capturar a relação sequencial e conseguir utilizar o contexto para auxiliar nas tarefas de classificação.

Apesar de mais robustos, os trabalhos dos artigos (COHAN et al., 2019) e (HU et al., 2022) são difíceis de implementar e treinar devido a sua complexidade, além de esforços extra como a criação de conjunto de dados auxiliares para as subtarefas do *Structural Scaffolds*, ou criar *prompts* que serão utilizados no treinamento com o método de *Masked AutoEncoders*. Os resultados de F1-Score dos três trabalhos utilizando o conjunto de dados ACL-ARC podem ser observados na Tabela 3. Conforme os resultados, é possível observar que o melhor experimento relatado é o de (HU et al., 2022) (seção 3.3) que utiliza a estratégia de *Variational Encoders*.

Tabela 3: Resultados de F1-score para os trabalhos relacionados usando o *dataset* ACL-ARC

Trabalho Relacionado	**F1**
(JURGENS et al., 2018)	0,54 (macro)
(COHAN et al., 2019)	0,68 (macro)
(HU et al., 2022)	0,76 (micro)

4 IMPLEMENTAÇÃO E EXPERIMENTOS

O objetivo deste trabalho é utilizar a técnica de otimização de modelos de linguagem *SetFit* para treinar um modelo de classificação de intenção de citação e avaliar o seu desempenho em relação a outras abordagens, utilizadas previamente na classificação, destacadas no capítulo 3. Além disso, iremos realizar experimentos comparando modelos de linguagem pré-treinados e avaliar qual tem o melhor desempenho. Por fim, iremos usar três tamanhos diferentes de conjunto de treino para avaliar quantos exemplos são necessários para se obter resultados satisfatórios. Neste capítulo, são destacadas as principais etapas, conforme ilustra a Figura 9, envolvidas na implementação e experimentação deste trabalho, além de discutir os resultados obtidos.

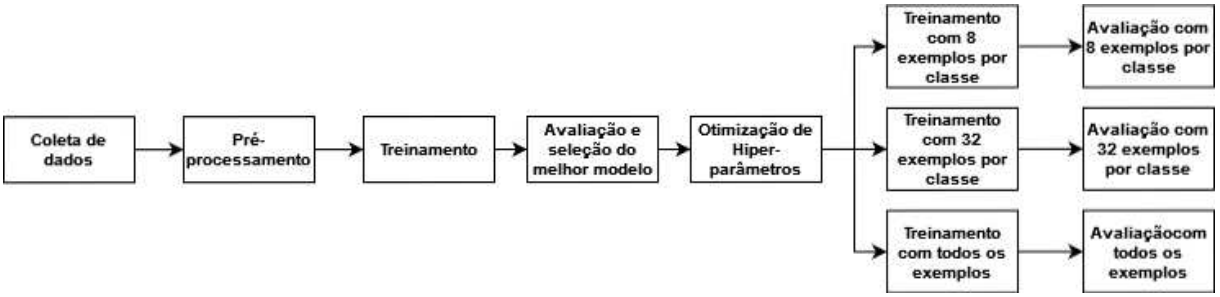


Figura 9: Principais etapas do desenvolvimento do trabalho

Antes da coleta de dados, foi necessário escolher o conjunto de dados apropriado para a tarefa de classificação que seria desenvolvida. Logo, foram analisadas algumas opções dentre a lista de diferentes conjuntos de dados relacionados a citações, que pode ser encontrada no trabalho de (ROMAN et al., 2021).

Para que fosse possível realizar a otimização do modelo de linguagem, precisava-se de um conjunto de dados com *corpus* textual associado à citação propriamente dita e à classificação de intenção de citação anotada para cada exemplo, o que não era disponibilizado pela maioria dos conjuntos listados no artigo (ROMAN et al., 2021). Então, restaram duas alternativas: ACL-ARC e o Sci-Cite (Sci-CI). O ACL-ARC (previamente apresentado na seção 3.1) é um conjunto de dados que contém 1.969 citações que foram anotadas em 6 classes distintas. Já, o Sci-CI (previamente apresentado na seção 3.2) é uma extensão do ACL-ARC com mais instâncias e uma variabilidade de áreas de pesquisa maior, no entanto, possui apenas 3 classes das 6 presentes no ACL-ARC, por essa razão, não foi utilizado neste trabalho.

Após a coleta e preparo dos dados do *dataset* escolhido, foram desenvolvidos os algoritmos para treinamento da classificação de intenção de citação e realizados experimentos com a biblioteca *SetFit*[5]. O primeiro experimento foi realizado para identificar qual o modelo de linguagem apresentava o melhor desempenho para a tarefa de classificação. Após a escolha do melhor modelo, foi realizada a otimização dos hiperparâmetros, a fim de identificar a melhor configuração de treinamento e melhorar a eficácia do modelo. Por último, foram realizados experimentos com 3 tamanhos de conjunto de dados diferentes. O primeiro experimento foi realizado com apenas 8 exemplos de cada classe, já o segundo experimento foi realizado com 32 exemplos por classe e o terceiro foram utilizados todos os exemplos de cada uma das classes. Por fim, foi realizada a otimização do hiperparâmetros, para avaliar se haveria alguma melhoria em relação ao modelo já treinado. Todos os treinamentos foram salvos na plataforma *Hugging Face*[6] com o objetivo de facilitar a inferência em novos exemplos. A seguir, serão abordados os detalhes das etapas mencionadas, além da apresentação dos resultados.

4.1 AMBIENTE DE EXECUÇÃO

O ambiente de execução envolveu o uso do *Google Colab*[7], que oferece unidades computacionais equipadas com placas de vídeo avançadas, facilitando assim a redução dos tempos de treinamento. O ambiente selecionado para rodar os códigos foi configurado com 12,7 GB de memória RAM e 15,0 GB dedicados à placa de vídeo. Detalhes específicos sobre o processador não foram identificados, mas o desempenho foi garantido primordialmente pelo uso eficiente da placa de vídeo e da memória RAM. A versão do *Python* utilizada neste contexto foi a 3.10.12.

Todos os códigos também foram salvos e versionados na plataforma *GitHub*[8] com o intuito de disponibilizar todo o trabalho desenvolvido, com os códigos de treinamento e também de inferência. Os modelos treinados também foram disponibilizados por meio da plataforma *Hugging Face*, o que facilita também a sua utilização para realizar previsões a partir de novos exemplos sem a necessidade de realizar o treinamento com o conjunto de dados.

[5] https://github.com/huggingface/setfit
[6] https://huggingface.co/
[7] https://colab.research.google.com/
[8] https://github.com/

4.2 CONJUNTO DE DADOS

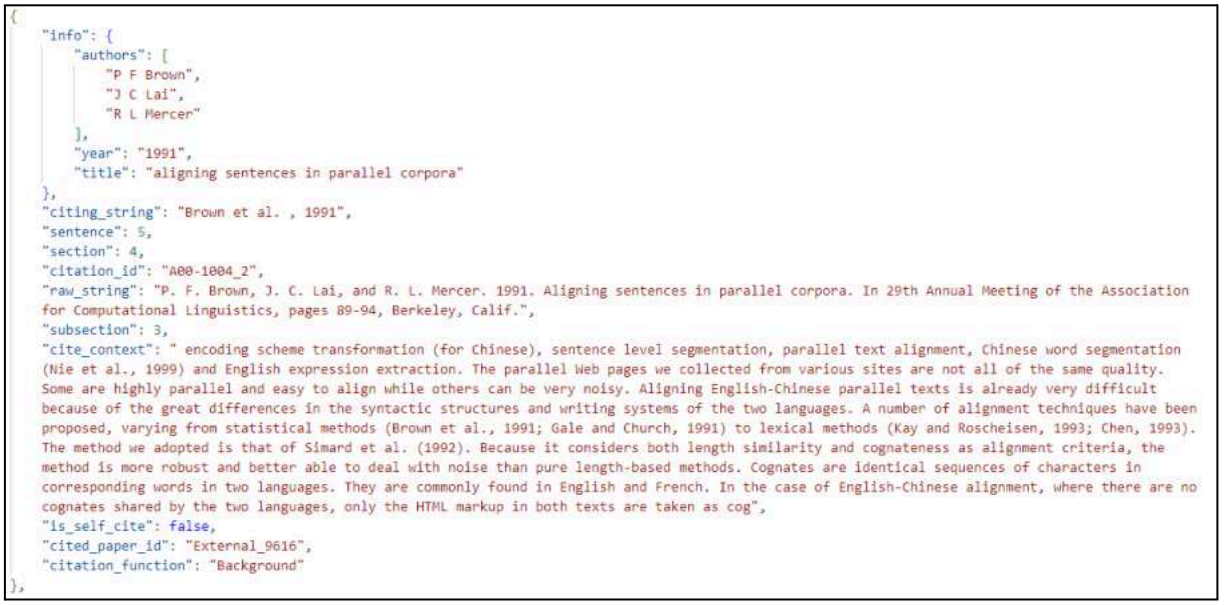
Após a seleção do conjunto de dados ACL-ARC para o treinamento de modelo, foi necessário realizar o *download* do mesmo a partir do portal disponibilizado pelos autores do artigo (JURGENS et al., 2018)[9]. O *dataset* é composto por diversos arquivos no formato JSON (*JavaScript Object Notation*) que possuem diversos itens no formato descrito na Figura 10. As chaves presentes em cada item do *dataset* no formato JSON são descritas a seguir:

- `info`: informações como autores, data de publicação, e título do artigo;
- `citing_string`: informação da referência da citação em si, que geralmente vem entre parênteses durante a citação;
- `sentence`: número da sentença que se encontra a citação em evidência;
- `section`: número da seção em que a citação se encontra. As seções são numeradas de 1 a *n*, sendo *n* o número de seções no artigo;
- `citation_id`: um identificador único gerado para cada uma das citações;
- `raw_string`: referência do artigo citado.
- `subsection`: subseção em que a citação foi encontrada (se presente);
- `cite_context`: bloco de texto que a citação foi extraída, não foi possível encontrar na documentação quantas palavras antes e depois foram usadas;
- `if_self_cite`: caso o autor tenha citado a si mesmo;
- `cited_paper_id`: um identificador único representando o artigo citado;
- `citation_function`: este item representa a classe da citação. Nem todas as citações possuem esta chave-valor, pois nem todas foram anotadas.

O passo inicial foi o desenvolvimento de um algoritmo para ler a lista de arquivos JSON e, em seguida, iterar em cada um dos itens e, caso encontrasse a chave `citation_function`, os conteúdos de `citation_function` e `cite_context` eram salvos em uma lista. Posteriormente, o conteúdo de `cite_context` passou por um algoritmo que retornava apenas a sentença que contém a citação referenciada. Por exemplo, o resultado final para a sentença do exemplo da Figura 10 seria "*A number of alignment techniques have been proposed, varying from statistical methods (Brown et al., 1991; Gale and Church, 1991) to lexical methods (Kay and Roscheisen, 1993; Chen, 1993)*". Em seguida,

[9] https://jurgens.people.si.umich.edu/citation-function/

todos os itens selecionados foram salvos em um objeto da classe chamada `Dataset`, que é utilizada na biblioteca do *Hugging Face* para facilitar o carregamento dos dados.



```
{
    "info": {
        "authors": [
            "P F Brown",
            "J C Lai",
            "R L Mercer"
        ],
        "year": "1991",
        "title": "aligning sentences in parallel corpora"
    },
    "citing_string": "Brown et al. , 1991",
    "sentence": 5,
    "section": 4,
    "citation_id": "A00-1004_2",
    "raw_string": "P. F. Brown, J. C. Lai, and R. L. Mercer. 1991. Aligning sentences in parallel corpora. In 29th Annual Meeting of the Association for Computational Linguistics, pages 89-94, Berkeley, Calif.",
    "subsection": 3,
    "cite_context": " encoding scheme transformation (for Chinese), sentence level segmentation, parallel text alignment, Chinese word segmentation (Nie et al., 1999) and English expression extraction. The parallel Web pages we collected from various sites are not all of the same quality. Some are highly parallel and easy to align while others can be very noisy. Aligning English-Chinese parallel texts is already very difficult because of the great differences in the syntactic structures and writing systems of the two languages. A number of alignment techniques have been proposed, varying from statistical methods (Brown et al., 1991; Gale and Church, 1991) to lexical methods (Kay and Roscheisen, 1993; Chen, 1993). The method we adopted is that of Simard et al. (1992). Because it considers both length similarity and cognateness as alignment criteria, the method is more robust and better able to deal with noise than pure length-based methods. Cognates are identical sequences of characters in corresponding words in two languages. They are commonly found in English and French. In the case of English-Chinese alignment, where there are no cognates shared by the two languages, only the HTML markup in both texts are taken as cog",
    "is_self_cite": false,
    "cited_paper_id": "External_9616",
    "citation_function": "Background"
},
```


Figura 10: Descrição da estrutura de cada item do conjunto de dados através de um exemplo

O conjunto ACL-ARC (previamente descrito na seção 3.1) é um conjunto de dados extraído a partir do *ACL Anthology Corpus* (ACL-ARC) e inicialmente possuía 1.969 exemplos de citações em inglês com a anotação da intenção de citação. No entanto, após algumas revisões pelos autores do trabalho que deu origem ao *dataset* (JURGENS et al., 2018), o conjunto de dados foi reduzido a 1.916 exemplos. Todas as versões dos dados podem ser encontradas no site oficial dos do artigo original[10]. Os dados foram anotados em 6 classes distintas: “BACKGROUND”, “USES”, “COMPARES OR CONTRASTS”, “MOTIVATION”, “CONTINUATION” e “FUTURE”. Cada uma dessas classes possui uma intenção descrita de acordo com o Quadro 1.

Outra característica do conjunto de dados é que ele é desbalanceado. As classes “BACKGROUND”, “USES” e “COMPARES OR CONTRASTS” possuem bem mais exemplos que as outras (com maior destaque para a classe “BACKGROUND”), conforme observado na Tabela 4.

[10] http://jurgens.people.si.umich.edu/citation-function/

Quadro 1: Descrição e exemplos das classes

Classe	Descrição	Exemplo
BACKGROUND	*P* possui informação relevante para este domínio	*This is often referred to as incorporating deterministic closure (Dörre, 1993).*
MOTIVATION	*P* ilustra a necessidade de dados, objetivos, métodos, etc.	*As shown in Meurers (1994), this is a well-motivated convention* [...]
USES	Usa dados, métodos, etc., de *P*.	*The head words can be automatically extracted* [...] *in the manner described by Magerman (1994).*
EXTENSION	Estende os dados, métodos, etc., de *P*.	[...] *we improve a two-dimensional multimodal version of LDA (Andrews et al., 2009)* [...]
COMPARES OR CONTRASTS	Expressa similaridade/diferenças com *P*.	*Other approaches use less deep linguistic resources (e.g., POS-tags Styne (2008)* [...]
FUTURE	*P* é um caminho potencial para trabalhos futuros.	[...] *but we plan to do so in the near future using the algorithm of Littlestone and Warmuth (1992).*

Fonte: Adaptado de (JURGENS et al., 2019)

Tabela 4: Distribuição das classes no conjunto de dados

Classe	Número de Citações
BACKGROUND	985
USES	359
COMPARES OR CONTRASTS	347
MOTIVATION	86
CONTINUATION	71
FUTURE	68

Cada exemplo do conjunto de dados é composto por duas colunas:

- `intent`: classificação da intenção de citação em forma numérica, com valores inteiros de 0 a 5, representando as classes: “BACKGROUND”, “USES”, “COMPARES OR CONTRASTS”, “MOTIVATION”, “CONTINUATION” e “FUTURE”, respectivamente.
- `text`: contém o texto da citação, englobando a sentença que foi utilizada para referenciar o artigo.

A divisão do conjunto de dados foi feita da seguinte forma: 1.532 exemplos no conjunto de treino, 192 para o conjunto de validação e 192 para o conjunto de teste, respeitando a proporção 80%-10%-10%. A distribuição das classes também foi mantida de forma aproximada ao conjunto de dados completo, sendo os valores de acordo com a Tabela 5. Para os experimentos com 8 e 32 exemplos por classe, o tamanho dos conjuntos de validação e testes foram mantidos e apenas o conjunto de treino sofreu alterações, com 48 e 192 exemplos no conjunto de treino respectivamente.

Tabela 5: Divisão do *dataset* por cada conjunto de separação

Classe	**Treino**	**Validação**	**Teste**
BACKGROUND	788	99	98
USES	296	36	36
COMPARES OR CONTRASTS	275	35	35
MOTIVATION	72	8	9
CONTINUATION	52	7	7
FUTURE	49	7	7

4.3 EXPERIMENTOS PROPOSTOS

Para a execução do estudo, foram propostas divisões do experimento em três tamanhos diferentes de conjunto de dados, e para cada um desses conjuntos foram utilizados 2 modelos de linguagem diferentes. Assim, com a variação do número de exemplos por classe, é possível observar a variação das métricas de avaliação em cada um dos experimentos e compará-los.

O primeiro experimento foi realizado com 8 exemplos de cada classe do conjunto de dados, totalizando 48 exemplos no conjunto de treinamento. Para o segundo experimento,

foram utilizados 32 exemplos de cada classe, totalizando 192 exemplos para treino. A princípio, seriam utilizadas 64 instâncias para um outro tipo de experimento, no entanto, devido à falta de exemplos em algumas classes, como a "FUTURE", utilizou-se todo o conjunto de treinamento para o terceiro experimento. Os conjuntos de validação e teste mantiveram-se inalterados em todos os três experimentos, respeitando a proporção da Tabela 5.

Para os modelos de linguagem, foram utilizadas quatro alternativas para a avaliação: `all-mpnet-base-v2`, `all-distilroberta-v1`, `all-MiniLM-L6-v2` e `all-MiniLM-L12-v2`. Os quatro são modelos com arquitetura de *Sentence Transformers* (REIMERS; GUREVYCH, 2019), que foi descrita nas seções 2.4 e 2.5, onde mapeiam sentenças ou parágrafos em vetores densos. Os dois primeiros criam vetores com dimensões 768 e os dois últimos criam vetores com 384 dimensões. Ambos foram utilizados a partir da plataforma *Hugging Face* que disponibiliza diversos modelos pré-treinados. Esses quatro modelos foram escolhidos com base nos resultados de *benchmarks* disponibilizados pelos autores[11], onde os dois apresentaram melhor *performance* na vetorização de sentenças.

É possível baixar os dois modelos apenas fazendo o *download* da biblioteca `sentence-transformers` a partir da biblioteca de gerenciamento de pacotes *Python Package Index* (PIP)[12]. Posteriormente, basta referenciar o identificador do modelo, que é uma *string* que representa unicamente cada modelo presente na plataforma, um exemplo pode ser observado na Figura 11.

```
from sentence_transformers import SentenceTransformer
sentences = ["This is an example sentence", "Each sentence is converted"]

model = SentenceTransformer('sentence-transformers/all-MiniLM-L6-v2')
embeddings = model.encode(sentences)
print(embeddings)
```

Figura 11: Exemplo de importação e utilização da biblioteca *sentence-transformers*

4.4 TREINAMENTO

Para os experimentos, foi utilizada a implementação do algoritmo *SetFit* (TUNSTALL et al., 2022) apresentado na seção 2.5. Todo o código da biblioteca, assim como exemplos de

[11] https://www.sbert.net/docs/pretrained_models.html
[12] https://pypi.org

como utilizá-la, podem ser encontrados no repositório original dos autores no *GitHub*[13]. Existem duas formas de utilizar a biblioteca: a primeira é clonar o repositório oficial do projeto; e a segunda é utilizar a biblioteca a partir do gerenciador de pacotes *Python Package Index* (PIP).

Os experimentos também foram divididos em dois *notebooks* também disponibilizados no *GitHub*[14]. Além disso, os experimentos foram divididos em 3 tamanhos diferentes para o conjunto de dados. Para isso, foi criada uma função para extrair um número pré-definido de exemplos de uma classe, conforme mostra a Figura 12.

```python
def get_subset(dataset, instances_per_class = 8):

    # Create a dictionary to store examples by class
    class_examples = {}

    # Iterate through the dataset and group examples by class
    for example in dataset:
        label = example["intent"]
        if label not in class_examples:
            class_examples[label] = []
        class_examples[label].append(example)

    # Create a new subset with 8 examples of each class
    subset = []

    for label, examples in class_examples.items():
        if len(examples) >= instances_per_class:
            subset.extend(random.sample(examples, instances_per_class))
        else:
            # If there are fewer than 8 examples, you can choose to include them all or skip the class.
            # Here, we include all available examples for this class.
            subset.extend(examples)

    # Assuming 'subset' is your list of dictionaries containing text and labels
    subset_data = {
        "text": [example["text"] for example in subset],
        "intent": [example["intent"] for example in subset],
    }

    # Create a Dataset instance
    return Dataset.from_dict(subset_data)
```

Figura 12: Função que cria um subconjunto com um número exato de exemplos por classe

Para o carregamento dos dados, foi utilizada a biblioteca `datasets` da *Hugging Face*. O conjunto de dados já havia sido previamente dividido em treino, validação e teste, conforme mencionado na seção 4.2. Em seguida, foi necessário realizar a codificação das classes de números inteiros para vetores de dimensão 6, que seriam compostos por zeros e um dígito 1 que representa a classe do exemplo, conforme mostra a Tabela 6.

[13] https://github.com/huggingface/setfit/tree/main
[14] https://github.com/ialvarenga/setfit-experiments

Tabela 6: Representação das classes em forma de vetores

Classe	Representação Inteiro	Label Vetor
BACKGROUND	0	[1,0,0,0,0,0]
USES	1	[0,1,0,0,0,0]
COMPARES OR CONTRASTS	2	[0,0,1,0,0,0]
MOTIVATION	3	[0,0,0,1,0,0]
CONTINUATION	4	[0,0,0,0,1,0]
FUTURE	5	[0,0,0,0,0,1]

Essa conversão foi um passo necessário devido à forma como a função de treinamento do pacote *SetFit* aceita dados. Para essa tarefa, foi utilizada a função detalhada na Figura 13, a qual utiliza a codificação *One Hot Encoding*.

```
def encode_labels(record):
    onehot_vec = [0 for x in range(6)]
    onehot_vec[record['intent']] = 1
    record['label'] = onehot_vec
    return record
```

Figura 13: Função de codificação das variáveis

Para os experimentos propostos, foram utilizadas duas abordagens de execução. A primeira corresponde ao *fine tuning* da rede neural com os hiperparâmetros padrão de inicialização da classe. Já a segunda, foi utilizada a biblioteca *optuna*[15], para buscar as configurações ótimas para a rede neural, por meio da estratégia *Grid Search* (discutida previamente na seção 3.1).

4.5 CONFIGURAÇÃO PADRÃO DE PARÂMETROS

Na inicialização com as configurações padrão, precisamos passar um modelo pré-treinado que pode ser baixado utilizando a função `from_pretrained` da biblioteca

[15] https://optuna.org/

SetFit, conforme mostra a Figura 14. Todos os modelos pré-treinados estão hospedados no site oficial do *Hugging Face*[16] que disponibiliza dezenas de modelos para diferentes tarefas.

```
[ ]  model_id = "sentence-transformers/paraphrase-mpnet-base-v2"
     model_fullset = SetFitModel.from_pretrained(model_id,  multi_target_strategy="one-vs-rest")
```

Figura 14: *Download* do modelo pré-treinado

De acordo com a documentação da função `from_pretrained`, dois parâmetros são obrigatórios para a execução correta: `model_id` e `multi_target_strategy`. O primeiro parâmetro é o caminho do modelo na plataforma *Hugging Face*, que será utilizado para o *download* do modelo. O segundo parâmetro é a estratégia de treinamento da camada de classificação durante o ajuste fino. Neste trabalho, foi escolhido `one-vs-rest` onde são treinados um classificador para cada classe e, durante a previsão, a classe escolhida é a que possui maior probabilidade. Em seguida, a classe de treinamento foi inicializada com os parâmetros conforme mostra a Figura 15.

```
# Create trainer
trainer = SetFitTrainer(
    model=model_fullset,
    metric='f1',
    metric_kwargs={'average': 'micro'},
    train_dataset=train_ds,
    eval_dataset=eval_ds,
    loss_class=CosineSimilarityLoss,
    batch_size=16,
    learning_rate=2e-5,
    num_iterations=20,
    num_epochs=5,
    seed=42
)
```

Figura 15: Inicialização da classe de treinamento

Para a tarefa de classificação, são necessários alguns parâmetros que serão explicados a seguir:

[16] https://huggingface.co/models

- `model`: é o objeto instanciado do modelo baixado, conforme exposto na Figura 14.
- `metric:` foi escolhida a métrica `f1`, pois a mesma permite que haja um equilíbrio entre o treinamento para cada uma das classes, sem que uma classe específica fique com desempenho menor que as outras. Vale ressaltar que essa métrica é utilizada na parte de treinamento da camada de classificação.
- `metric_kwargs:` nesse parâmetro, são passadas configurações específicas da métrica escolhida no parâmetro `metric`. Neste exemplo, passamos a opção `weighted`, que conta o total de verdadeiros positivos, falsos positivos e falsos negativos.
- `train_dataset:` referencia o conjunto de treinamento.
- `eval_dataset:` referencia o conjunto de validação.
- `loss_class:` neste parâmetro, é passada a função objetivo para o ajuste fino da rede neural em relação à codificação do texto, isto é, a função objetivo será utilizada para fazer com que sentenças semelhantes tenham vetores mais próximos e sentenças com classificações distintas tenham representações vetoriais mais distantes. Assim, é utilizada a função similaridade de cosseno (RAHUTOMO; KITASUKA; ARITSUGI, 2012) para minimizar ou maximizar a distância entre os vetores representativos das sentenças.
- `batch_size:` ou tamanho do lote, é o número de exemplos que serão processados simultaneamente. Escolher o tamanho do lote adequado é crucial para o desempenho do treinamento, pois um lote pequeno pode levar a uma convergência instável e demorada, enquanto um lote muito grande pode exigir mais memória do que disponível e potencialmente levar a uma convergência prematura com mínimos locais.
- `learning_rate:` ou taxa de aprendizado, é o tamanho do "passo" dado pelo algoritmo de otimização. Encontrar o valor ideal para a taxa de aprendizado é essencial para o bom desempenho da rede neural, pois valores muitos altos podem fazer com que os pesos sejam ajustados de forma excessiva podendo "passar" do ponto ótimo. Já taxas de aprendizado muito pequenas ocasionam ajustes muito pequenos, o que torna o processo de aprendizado lento e o processo de otimização pode ficar preso em mínimos locais.
- `num_interations:` o número iterações para geração de pares positivos e negativos. Este valor é utilizado como parâmetro no algoritmo do *contrastive learning* representado por *R* (abordado na seção 2.5). O valor associado será utilizado para

calcular o número de pares, sendo *R* pares positivos e *R* pares negativos para cada uma das sentenças no conjunto de dados.

- `num_epochs:` o número de épocas que o corpo da rede será treinado. O número de épocas no treinamento de uma rede neural refere-se à quantidade de vezes que o conjunto de dados completo é usado para atualizar os pesos da rede durante o processo de treinamento. Isso é feito para otimizar os pesos da rede de maneira que o erro na saída seja minimizado, e a rede aprenda de forma eficaz os padrões presentes nos dados.
- `seed:` valor utilizado para reprodutibilidade dos resultados.

Para o processo de treinamento, foram gerados 61.280 (2 * 20 * 1.532) pares de exemplos únicos de treinamentos, conforme pode ser visto na Figura 16, e com um total de 3.830 passos de otimização, pois os exemplos foram divididos em *batchs* de tamanho 16 e para a ilustração usamos apenas 1 época. Para o experimento com os parâmetros padrão, foram realizadas 30 execuções e a média de tempo foi de ~17 minutos com a utilização de uma GPU A100 disponibilizada no ambiente de execução do *Google Google Colab*, como pode ser observado na Figura 16. Além disso, é possível observar o cálculo da métrica de avaliação F1 micro, passada como parâmetro para o `trainer`, no conjunto de dados de avaliação passado nos parâmetros conforme observado na Figura 15.

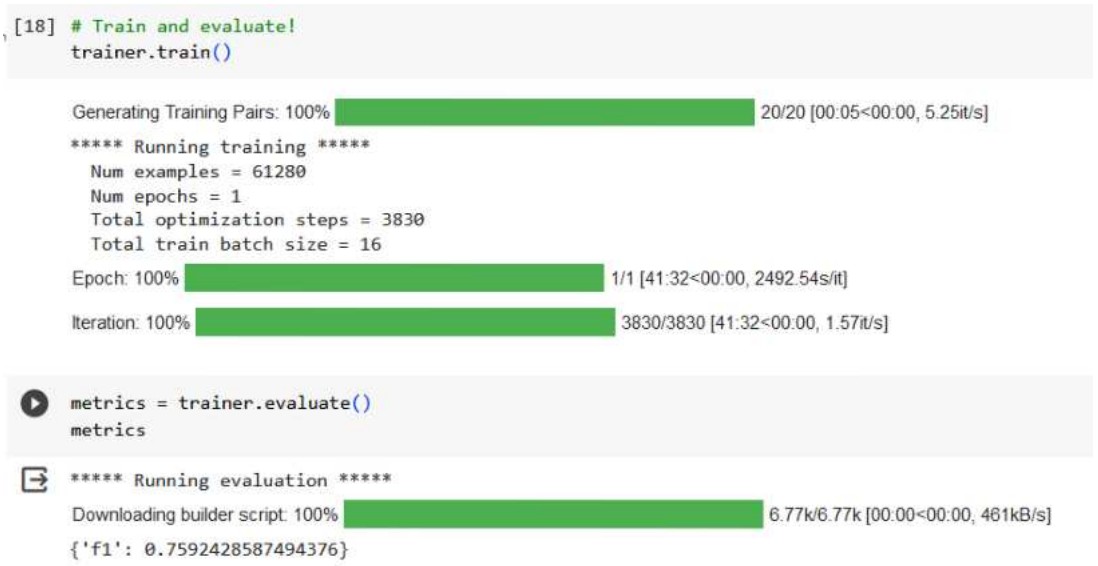


Figura 16: Células de treinamento e métricas de treino

Em seguida, o modelo foi avaliado no conjunto de dados de teste. A função `evaluate_model` recebe o conjunto de treinamento e o modelo. A partir desses parâmetros, o modelo realiza a previsão das classes a partir da coluna `text` presente nos dados. A partir dos resultados, são calculadas as métricas de precisão (*precision*), revocação (*recall*), F1-score e suporte (*support*). A Figura 17 ilustra um exemplo de saída do da função, e para análises futuras iremos considerar a agregação macro para as métricas expostas nos resultados, para que possamos comparar com os resultados dos trabalhos relacionados.

	precision	recall	f1-score	support
0	0.80	0.79	0.80	99
1	0.80	0.89	0.84	36
2	0.70	0.54	0.61	35
3	0.50	0.71	0.59	7
4	0.57	0.50	0.53	8
5	0.83	0.71	0.77	7
micro avg	0.76	0.74	0.75	192
macro avg	0.70	0.69	0.69	192
weighted avg	0.77	0.74	0.75	192
samples avg	0.74	0.74	0.74	192

Figura 17: Métricas de avaliação do experimento com todo o *dataset*

As métricas agregadas na parte inferior da Figura 17 — média micro (*micro average*), média macro (*macro average*), média ponderada (*weighted average*) e média das amostras (*samples average*), conforme descritas previamente na seção 2.6 — fornecem uma avaliação abrangente do desempenho do modelo. Para as avaliações as médias macro (ou micro) foram consideradas para se comparar de forma fiel aos trabalhos relacionados abordados no capítulo 3. Observando todas essas métricas, fica evidente que existe uma variação notável no desempenho entre as classes, e que aprimorar o balanceamento e a representatividade dentro do conjunto de dados pode levar a uma melhoria na precisão global do modelo.

4.6 CLASSIFICAÇÃO COM OTIMIZAÇÃO DOS HIPERPARÂMETROS

Após a análise inicial das métricas apresentadas pela rede neural após o treinamento com os hiperparâmetros padrão, foi realizado um experimento no qual foi utilizada a biblioteca *optuna*[17] para realizar a busca dos hiperparâmetros otimizados para a tarefa de

[17] https://optuna.org/

classificação das intenções. Para essa tarefa, foi adicionado o módulo *optuna* a biblioteca *SetFit* no momento da instalação, conforme mostra a Figura 18.

```
!pip install setfit[optuna]
!pip install datasets
```

Figura 18: Instalação das bibliotecas datasets e SetFit + optuna

O código é muito semelhante ao descrito na seção 4.5, no entanto, ao invés de passar uma instância do modelo carregado em memória RAM, conforme a Figura 14, são necessárias duas ações. A primeira é definir uma função que retorna um dicionário, no qual as chaves são os títulos dos hiperparâmetros e as chaves são listadas com os valores que serão utilizados na busca, conforme mostra a Figura 19.

```
def make_model(params=None):
    multi_target_strategy = params["multi_target_strategy"] if params else "one-vs-rest"
    return SetFitModel.from_pretrained(
        model_id, multi_target_strategy=multi_target_strategy
    )
```

Figura 19: Declaração da função que é utilizada na otimização dos hiperparâmetros

A segunda ação é passar a função declarada para um outro parâmetro de inicialização da classe `SetFitTrainer`, que neste caso é o `model_init`. O parâmetro `model_init` recebe uma função com a definição semelhante ao que foi apresentado anteriormente na Figura 15. Assim, a instância `trainer` poderá inicializar diversos modelos e avaliar qual conjunto de hiperparâmetros possui as melhores métricas. Após a finalização da busca, um dicionário com os melhores parâmetros é retornado e, posteriormente, um novo modelo é treinado a partir dessa configuração, conforme mostrado nas Figuras 20 e 21.

```
# Create trainer
trainer = SetFitTrainer(
    model_init=make_model,
    metric='f1',
    metric_kwargs={'average': 'micro'},
    train_dataset=train_ds,
    eval_dataset=eval_ds,
    loss_class=CosineSimilarityLoss,
    num_iterations=20,
    num_epochs=1
)
```

Figura 20: Inicialização do *trainer* passando a função de possíveis hiperparâmetros

```
best = trainer.hyperparameter_search(hyperparameter_search_function, n_trials=10)
best
```

Figura 21: Inicialização da busca por hiperparâmetros

Além disso, é possível utilizar a biblioteca *optuna* para plotar a importância dos hiperparâmetros, ou seja, os hiperparâmetros que mais contribuíram para a melhoria das métricas de avaliação do modelo. O resultado da importância das hiperparâmetros pode ser visualizado na Figura 22, e podemos observar que os hiperparâmetros `batch_size` e `learning_rate` possuem maior importância na otimização do modelo ao utilizar a biblioteca *optuna*.

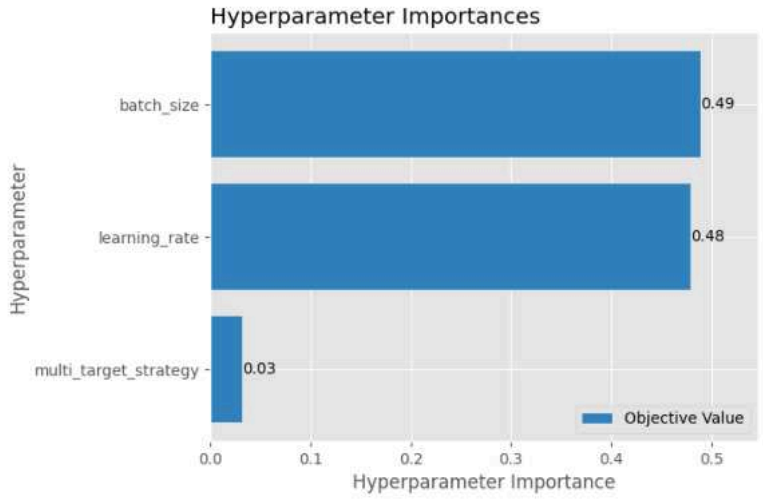


Figura 22: Resultado do experimento utilizando um otimizador de hiperparâmetros

4.7 RESULTADOS

Nesta seção, serão apresentados os resultados dos diversos experimentos, e iremos comparar cada um deles. Primeiro, quatro modelos de linguagem serão comparados em relação ao desempenho na tarefa de classificação de intenção de citação utilizando todos os dados disponíveis. Posteriormente, foi realizada a busca dos melhores hiperparâmetros para o modelo escolhido na seção 4.7.1. Por fim, foi analisado o desempenho da técnica de *fine-tuning* em três tamanhos do conjunto de dados diferentes, com o intuito de verificar qual é o tamanho mínimo satisfatório para o conjunto de treinamento para se obter um desempenho razoável de previsão.

4.7.1 Comparação entre modelos de linguagem

Para este experimento foram escolhidos 4 modelos de linguagem disponibilizados na plataforma *Hugging Face* pela equipe de autores do artigo (REIMERS; GUREVYCH, 2019). A escolha dos modelos foi determinada pelo desempenho dos modelos em um *benchmark*, também realizado pelos autores do artigo (REIMERS; GUREVYCH, 2019), de *embedding* de sentença em diversas tarefas, a lista pode ser encontrada no portal do artigo[18]. A Tabela 7 apresenta o resultado de diversas execuções com hiperparâmetros ótimos encontrados para cada um dos modelos. Além disso, esses experimentos foram realizados com os dados de treinamento disponíveis (1.532 exemplos) com o intuito de comparar modelos de linguagens distintos.

Tabela 7: Resultado da experimentação com diferentes Sentence Transformers

Modelo	F1-Score (MACRO)
`all-mpnet-base-v2`	0,57
`all-distilroberta-v1`	0,41
`all-MiniLM-L6-v2`	0,67
`all-MiniLM-L12-v2`	0,69

Conforme é possível observar na Tabela 7, o modelo que apresentou o melhor desempenho foi o `all-MiniLM-L12-v2`, este modelo possui menos dimensões que os

[18] https://sbert.net/docs/pretrained_models.html

`all-mpnet-base-v2` (768) e o `all-distilroberta-v1` (768), mas tem a mesma dimensão que o modelo `all-MiniLM-L6-v2` (384). Assim, pode-se notar que para este conjunto de dados, modelos que têm menos dimensões apresentam melhor desempenho que os modelos com maior dimensão.

O próximo passo foi comparar a execução do modelo com melhores resultados otimizado com o modelo ao ser treinado com os hiperparâmetros otimizados, com o intuito de comparar como o desempenho das seis classes evoluiu com a melhoria dos hiperparâmetros. A Tabela 8 mostra os resultados do modelo `all-MiniLM-L12-v2` sem a otimização dos hiperparâmetros.

Tabela 8: Resultados do modelo com parâmetros padrão

Classe	Precisão	Revocação	F1-score	Suporte
BACKGROUND	0,8	0,79	0,8	99
USES	0,8	0,89	0,84	36
COMPARES OR CONTRASTS	0,7	0,54	0,61	35
MOTIVATION	0,5	0,71	0,59	7
CONTINUATION	0,57	0,5	0,53	8
FUTURE	0,83	0,71	0,77	7
Média Macro	0,70	0,69	0,69	192

Como é possível observar, os valores padrão de configuração de treinamento retornam métricas de avaliação boas se comparados a outros experimentos realizados no conjunto de dados ACL-ARC na literatura (JURGENS et al., 2018; COHAN et al., 2019). Também é possível verificar que, apesar de algumas classes possuírem mais exemplos de treinamento que outras, não ocorreu um melhor desempenho nesses casos. Por exemplo, a classe “COMPARES OR CONTRASTS” não teve um desempenho tão bom quanto a classe “FUTURE”. Uma hipótese é que a classe “FUTURE” pode ser facilmente indicada pelos verbos de ação ou expressões como “em trabalhos futuros”, que caracterizam unicamente essas sentenças, em contraste com a classe “COMPARES OR CONTRASTS” que pode utilizar verbos que podem ser compartilhados com outras classes.

Além disso, os resultados do experimento com todos os exemplos do conjunto de dados foram satisfatórios em relação aos resultados mostrados na Tabela 4, apresentada

previamente na seção 3.4. No entanto, o modelo apresenta um desempenho variado entre as classes devido à própria natureza desbalanceada da distribuição dos exemplos. Também é possível observar que as classes “BACKGROUND” e “USES” têm os melhores resultados e possuem o maior suporte entre as classes.

Com relação à precisão, as classes “BACKGROUND”, “USES” e “FUTURE” apresentaram resultados satisfatórios. As classes “BACKGROUND” e “USES” possuem ampla representatividade no conjunto de dados, o que pode ter contribuído para o desempenho robusto nesses casos, sugerindo que a quantidade de exemplos disponíveis para o treinamento é um fator crítico para a precisão do modelo. A classe “FUTURE”, apesar de seu menor número de exemplos, também exibiu uma precisão comparativamente elevada. Esta observação pode ser atribuída às propriedades distintas das expressões de citação futuras, as quais, frequentemente, incluem locuções específicas, como “no futuro próximo”, que podem ser reconhecidas pelo modelo, indicando uma maior facilidade de identificação dessas expressões pelo algoritmo de aprendizado de máquina utilizado.

O erro de algumas sentenças pode até ser justificado pela falta de contexto com relação ao pertencimento de uma citação a um classe específica. Por exemplo, a citação “*One approach to this problem is that taken by the ASCOT project (Akkerman et al., 1985; Akkerman, 1986)*” que foi anotada pertencendo à classe “BACKGROUND”, foi classificada pelo modelo como “COMPARES OR CONTRASTS”, e pela leitura do texto da citação, ambas as classes fazem sentido como intenção, o que seria necessário passar mais contexto para o modelo.

Por outro lado, algumas citações são bem claras quanto à intenção em relação ao trabalho citado. Por exemplo, a citação “*Better results would be expected by combining the PCFG-LA parser with discriminative reranking approaches (Charniak and Johnson, 2005 ;Huang, 2008) for self training.*” é uma referência que indica que experimentos futuros utilizando um *parser* com re-ranqueamento discriminativo podem levar a resultados melhores do que os obtidos previamente.

Ainda sobre a precisão, é possível observar que a classe “COMPARES OR CONTRASTS” tem desempenho moderado em relação às outras duas classes que possuem praticamente a mesma quantidade de exemplos de treinamento. Já as classes “MOTIVATION” e “CONTINUATION” apresentaram desempenhos ruins, possivelmente pela falta de exemplos suficientes para o treinamento e a falta de capacidade do modelo de capturar relações que identificam unicamente textos com essas classes como foi observado para “FUTURE”.

Já com relação à revocação, é possível observar um mesmo padrão em relação à precisão, com a exceção de “MOTIVATION” que possui um desempenho razoável em comparação a “COMPARES OR CONTRASTS” e “CONTINUATION”. A classe “FUTURE” (5), apesar de não ter desempenhado tão bem quanto na métrica de precisão, ainda consegue se destacar em relação a classes que possuíam bem mais exemplos.

Como a métrica F1-score é uma média harmônica entre as duas métricas abordadas nos parágrafos anteriores, a mesma irá refletir o comportamento observado previamente: “BACKGROUND e “USES se destacam numericamente em relação às outras classes, e “FUTURE também possui um desempenho razoável. No entanto, é importante notar que as classes “COMPARES OR CONTRASTS”, “MOTIVATION” e “CONTINUATION” apresentam F1-scores mais baixos, o que sugere um equilíbrio menos eficiente entre precisão e revocação, provavelmente, devido ao menor número de exemplos disponíveis para essas classes no treinamento do modelo ou devido à complexidade intrínseca em identificar corretamente tais intenções de citação.

4.7.2 Busca de hiperparâmetros ótimos

Para o segundo experimento, foram utilizados os hiperparâmetros ótimos encontrados a partir da utilização da biblioteca *optuna*, para isso, foi realizada uma busca em grade (*grid search*) a partir dos valores mostrados na Tabela 9. Para a taxa de aprendizado (`learning_rate`), foi utilizado o parâmetro `log` como verdadeiro, assim, os valores intermediários no intervalo da Tabela 10 são sugeridos utilizando a função matemática *log*. Para o tamanho do lote de treinamento (`batch_size`), foram escolhidos 4 tamanhos diferentes. Para a estratégia de classificação (`multi_target_strategy`), foram testadas duas opções: `one-vs-rest` e `multi-output`. Por fim, para o número de épocas (`num_epochs`) foram usados quatro valores conforme a Tabela 10. Após a busca por hiperparâmetros ótimos, os valores, apresentados na Tabela 10, foram selecionados a partir da Tabela 9.

Tabela 9: Intervalo de hiperparâmetros utilizados no treinamento

Hiperparâmetro	Intervalo
`learning_rate`	`[1e-5, 1e-3]`
`batch_size`	`[4,8,16]`[19]
`mult_target_strategy`	`["one-vs-rest", "multi-output"]`
`num_epochs`	`[1,3,5,7]`

Tabela 10: Quadro com hiperparâmetros ótimos selecionados

Hiperparâmetro	Valor
`learning_rate`	`2,14e-5`
`batch_size`	`16`
`multi_target_strategy`	`multi-output`
`num_epochs`	`5`

Após a finalização da busca, um novo modelo foi treinado a partir da ST `all-MiniLM-L12-v2` com os valores da Tabela 10 e os resultados podem ser observados na Tabela 11.

Tabela 11: Resultados do modelo com parâmetros após a otimização

Classe	Precisão	Revocação	F1-score	Suporte
BACKGROUND	0,82	0,8	0,81	99
USES	0,83	0,81	0,82	36
COMPARES OR CONTRASTS	0,61	0,66	0,63	35
MOTIVATION	1,0	0,57	0,73	7
CONTINUATION	1,0	0,62	0,77	8
FUTURE	0,75	0,86	0,8	7
Média Macro	0,78	0,76	0,76	192

[19] Não foi possível utilizar a configuração `batch_size` de 32 devido à limitação de recursos computacionais

Como pode observado na Tabela 11, todas as métricas agregadas do F1-score tiveram melhoras, isto é, a média macro (macro avg) foi de 0,69 para 0,76. Esse resultado a média macro é equiparável ao trabalho de (HU et al., 2022). Em nível mais detalhado, todas as classes tiveram um aumento no desempenho com exceção da classe “USES” que foi de um F1-score de 0,84 para 0,82, o que não é uma perda significativa.

Além disso, é possível observar que houve um *overfit* em relação às classes “MOTIVATION” e “CONTINUATION”, onde ambas apresentaram valor para precisão igual a 1. O que pode ter ocorrido devido à pouca quantidade de exemplos de teste, fazendo com que todos os exemplos fossem classificados de forma correta, porém a revocação se mantém baixa.

4.7.3 Experimento com diferentes tamanhos para o conjunto de treinamento

Como foi abordado na introdução do capítulo 4, também houve experimentações com diferentes tamanhos de conjunto de dados, nas quais, o número de exemplos por classe seria especificado de duas formas. Assim, foi possível analisar a influência do número de exemplos por classe e o desempenho geral do modelo treinado.

Tabela 12: Métricas em relação ao número de exemplos no treino

Experimento	Exemplos por classe	F1-Score (Macro)
1	8	0,28
2	32	0,47
3	Todos os exemplos de cada classe	0,76

Conforme pode ser observado na Tabela 12, mesmo com a geração de exemplos a partir da técnica de aprendizado contrastivo, a *performance* de 8 exemplos por classe (totalizando 48 exemplos no conjunto de treinamento antes do *fine tuning*) tem um desempenho baixo, comparada com os outros trabalhos abordados na Tabela 14, no entanto, o tempo de treinamento foi apenas 30 segundos. Já, o experimento com 32 exemplos por classe (totalizando 192 exemplos no conjunto de treinamento antes do *fine tuning*), obteve resultados semelhantes aos retratados no artigo (JURGENS et al., 2018), onde foram utilizados todos os exemplos, com tempo de treinamento de 3 minutos. Isso demonstra um resultado promissor para o treinamento de modelos para domínios com dados escassos, pois com apenas 192

exemplos, a técnica de aprendizado contrastivo pode otimizar o modelo, equiparando o desempenho com modelos treinados com todos os dados (JURGENS et al., 2018).

Os melhores resultados vieram da utilização de todos os exemplos disponíveis em cada uma das classes e, mesmo com o desequilíbrio na quantidade de exemplos de cada uma das classes, foi possível obter métricas satisfatórias. Além disso, o artigo que apresenta o *SetFit* realiza experimentos em um *dataset* balanceado, e com a realização dos experimentos deste trabalho podemos observar que o aprendizado contrastivo apresenta resultados satisfatórios mesmo com classes que possuem apenas algumas dezenas de exemplos.

Por fim, os modelos com um número pequeno de dimensões treinados com a metodologia do *SetFit* apresentaram desempenho melhor que os trabalhos (JURGENS et al., 2018) e (COHAN et al., 2019) (discutidos nas seções 3.1 e 3.2), além de apresentar métricas equivalentes ao modelo treinado utilizando *Masked AutoEncoders* (HU et al., 2022) (discutido na seção 3.3), conforme observado na Tabela 13. Estes resultados mostram indícios de que áreas com poucos dados e/ou poucos recursos computacionais podem se utilizar desta técnica para criar modelos com desempenho satisfatório.

Tabela 13: Comparação entre o trabalho atual e os trabalhos relacionados

Trabalho	**F1**
(JURGENS et al., 2018)	0,54 (macro)
(COHAN et al., 2019)	0,68 (macro)
(HU et al., 2022)	0,76 (micro)
Este Trabalho	0,76 (macro) 0,78 (micro)

5 CONCLUSÃO

Foi realizada uma pesquisa com diversos experimentos, cujo resultado foi um modelo de linguagem otimizado para classificação de intenção de citação em artigos científicos. Para isso, foram propostos experimentos variando o tamanho do conjunto de treino em 8 exemplos por classe, 32 exemplos por classe e todos os exemplos disponíveis, além de otimizar dois modelos de linguagem para a tarefa de classificação. Então, todos os experimentos foram avaliados e comparados com trabalhos prévios de classificação de intenção de citação.

A partir dos experimentos, foi possível observar que o melhor modelo de linguagem utilizado nos experimentos foi o `all-MiniLM-L6-v2` (384 dimensões) que possui menor número de dimensões que o outro modelo de linguagem `all-mpnet-base-v2` (768 dimensões), o que é um ponto positivo para realizar treinamentos em computadores com poucos recursos computacionais. Outro fator observado é que aumentar o número de dimensões do vetor que representa o texto não necessariamente irá melhorar os resultados das métricas de avaliação.

Além disso, conforme mostrado nos experimentos, foi possível refinar um modelo de linguagem (LLM) para realizar a tarefa de classificação de intenção de citação com apenas alguns exemplos e obter resultados bem próximos do estado da arte em que foram utilizados modelos mais robustos e de implementação complexa. Isso permite que áreas do conhecimento, com dados relativamente escassos, possam ser utilizadas para criar modelos para tarefas como classificação com resultados satisfatórios. Ademais, a metodologia de *contrastive learning* permite que diversos novos exemplos sejam gerados, possibilitando uma maior eficiência mesmo em conjuntos de dados com poucos exemplos.

Outro ponto importante é que a utilização da metodologia de aprendizado *SetFit* é uma técnica poderosa que consome poucos recursos computacionais, e também possui tempo de execução mais rápido que métodos tradicionais de aprendizado *few-shot*.

Por fim, a partir dos experimentos, foi criado um modelo de classificação de intenção de citação com fácil acesso a partir da plataforma *Hugging Face,* ou mesmo a partir dos *notebooks* disponibilizados no *GitHub*. O modelo apresenta resultados equivalentes ao atual estado da arte, o que permitirá a criação de aplicações de mapeamentos de intenções e evitar que publicações com retratações possam ser identificadas pela intenção que foi usada em artigos, e evitando a perpetuação de informações que podem ser inválidas.

No entanto, houve algumas limitações, principalmente pela disponibilidade de recursos computacionais que comportassem modelos mais robustos. Além disso, devido à

natureza desbalanceada do conjunto de dados, algumas classes tiveram suas métricas prejudicadas devido aos poucos exemplos disponíveis para o treinamento. Por fim, o conjunto de dados utilizado se limitava apenas à área de computação linguística, assim, não foi possível analisar a generalização para citações em outras áreas de estudo. Por conseguinte, existem oportunidades de trabalhos futuros a partir desta pesquisa, em contextos diversos, tais como:

- Com relação aos modelos de linguagem utilizados, é possível utilizar modelos mais robustos com o intuito de obter melhores resultados. Além disso, é possível integrar com modelos de análise de sentimentos para que possam servir como uma classificação complementar para identificar se o autor está citando com sentimento *positivo*, *neutro* ou *negativo*. Além disso, é possível utilizar o modelo para a análise de intenção de citação no contexto de citação de artigos retratados, com o intuito de analisar a intenção dos autores ao referenciar um artigo retratado.
- Quanto ao conjunto de dados, é possível expandir o conjunto de dados buscando novos exemplos de intenção de citação, e ainda utilizar a classificação do modelo com um pré-anotador para incrementar o número de exemplos. Também é possível realizar experimentos com dados sintéticos, a partir da descrição das classes, utilizando modelos de linguagem mais robustos como GPT-4 (ACHIAM et al., 2023) ou o LLAMA 2 (TOUVRON et al., 2023), com o objetivo de analisar o desempenho desses modelos em dados que não foram retirados de um artigo real.
- Por fim, é possível integrar o modelo de classificação em aplicações Web que auxiliem na pesquisa e possibilitem a identificação da intenção de um determinado autor ao citar um trabalho prévio, focando principalmente em artigos que foram retratados. Isso permitirá que a pesquisa por referências possa ser mais segura, e que as informações nas quais o autor está se baseando sejam confiáveis.

REFERÊNCIAS

ACHIAM, J., ADLER, S., AGARWAL, S., AHMAD, L., AKKAYA, I., ALEMAN, F. L., MCGREW, B. 2023. Gpt-4 technical report. **arXiv** preprint arXiv:2303.08774.

ALTAE-TRAN, H., RAMSUNDAR, B., PAPPU, A. S., PANDE, V. 2017. Low Data Drug Discovery with One-shot Learning. **ACS Central Science**, 3(4), 283-293.

BIRD, S., DALE, R., DORR, B. J., GIBSON, B. R., JOSEPH, M. KAN,M.-Y., LEE, D. POWLEY, B., RADEV, D. R., TAN, Y. F. 2008. The ACL Anthology Reference Corpus: A Reference Dataset for Bibliographic Research in Computational Linguistics. In **Proceedings of International Conference on Language Resources and Evaluation (LREC)**.

BROWN, T. B., MANN, B., RYDER, N., SUBBIAH, M., KAPLAN, J. D., DHARIWAL, P., AMODEI, D. 2020. Language models are few-shot learners. **Nature**, 577(7792), 590-597.

CHAWLA, N. V., BOWYER, K. W., HALL, L. O., KEGELMEYER, W. P. 2002. SMOTE: synthetic minority over-sampling technique. **Journal of artificial intelligence research**, 16, 321-357.

CHITTY-VENKATA, K. T., EMANI, M., VISHWANATH, V., SOMANI, A. K. 2022. Neural architecture search for transformers: A survey. **IEEE Access**, 10, 108374-108412.

COHAN, A. et al. 2019. Structural scaffolds for citation intent classification in Scientific Publications, **Proceedings of the 2019 Conference of the North** [Preprint]. doi:10.18653/v1/n19-1361.

DEVLIN, J., CHANG, M.-W., LEE, K., TOUTANOVA, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. **Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)**, 4171-4186.

FEI-FEI, L., FERGUS, R., PERONA, P. 2006. One-shot learning of object categories. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, 28(4), 594-611.

FERNÁNDEZ-DELGADO, M., CERNADAS, E., BARRO, S., AMORIM, D. 2014. Do we need hundreds of classifiers to solve real world classification problems? **J. Mach. Learn. Res.** 15, 1 (January 2014), 3133–3181.

FINN, C., ABBEEL, P., LEVINE, S. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. **Proceedings of the 34th International Conference on Machine Learning**, 70, 1126-1135.

FRICKE, S. 2018. Semantic scholar. **Journal of the Medical Library Association: JMLA**, 106(1), 145.

GÉRON, A. 2019. **Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems**. 2nd ed. CA 95472: O'Reilly.

HARWOOD, N. 2009. An interview-based study of the functions of citations in academic writing across two disciplines. **Journal of Pragmatics**, 41(3):497–518.

HASSANPOUR, S., BAY, G., LANGLOTZ, C. P. 2017. Characterization of Change and Significance for Clinical Findings in Radiology Reports Through Natural Language Processing. **Journal of Digital Imaging**, 30(3), 314-322.

HENDERSON, J., POPA, D. 2016. A Vector Space for Distributional Semantics for Entailment. **ArXiv**, abs/1607.03780.

HOCHREITER, S., SCHMIDHUBER, J. 1997. Long Short-Term Memory. **Neural Computation**, 9(8), 1735-1780.

HOWARD, J., RUDER, S. 2018. Universal Language Model Fine-tuning for Text Classification. **ArXiv** preprint arXiv:1801.06146.

HU, D., HOU, X., DU, X., ZHOU, M., JIANG, L., MO, Y., SHI, X. 2022. VarMAE: Pre-training of Variational Masked Autoencoder for Domain-adaptive Language Understanding. **arXiv** preprint arXiv:2211.00430.

IHSAN, I., QADIR, M.A. 2019. CCRO: Citation's Context & Reasons Ontology, **IEEE Access**, 7, pp. 30423–30436. doi:10.1109/access.2019.2903450.

JURAFSKY, D., MARTIN, J. 2020. **Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition**. 3rd Edition draft. (https://web.stanford.edu/~jurafsky/slp3/)

JURGENS, D., KUMAR, S., HOOVER, R., MCFARLAND, D., JURAFSKY, D. 2018. Measuring the Evolution of a Scientific Field through Citation Frames. **Transactions of the Association for Computational Linguistics**, 6:391–406.

KOCH, G., ZEMEL, R., SALAKHUTDINOV, R. et al. 2015. Siamese neural networks for one-shot image recognition. In **ICML deep learning workshop**, volume 2, page 0. Lille.

KULLBACK, S., LEIBLER, R.A., 1951. On information and sufficiency. **The annals of mathematical statistics**, 22(1), pp.79-86.

LATOUR, B. 1987. **Science in action: How to follow scientists and engineers through society**. Harvard University Press.

LIPTON, Z. C., ELKAN, C., NARAYANASWAMY, B. 2014. Thresholding classifiers to maximize F1 score. **arXiv**: Machine Learning.

LIU, H., TAM, D., MUQEETH, M., MOHTA, J., HUANG, T., BANSAL, M., RAFFEL, C. 2022. Few-shot parameter-efficient finetuning is better and cheaper than in-context learning. **Advances in Neural Information Processing Systems**.

MAHABADI, R. K., ZETTLEMOYER, L., HENDERSON, J., MATHIAS, L., SAEIDI, M., STOYANOV, V., YAZDANI, M. 2022. Prompt-free and efficient few-shot learning with language models. In **Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics** (Volume 1: Long Papers), pages 3638–3652, Dublin, Ireland. Association for Computational Linguistics.

MIKOLOV, T., CHEN, K., CORRADO, G., DEAN, J. 2013. Efficient estimation of word representations in vector space. **arXiv** preprint arXiv:1301.3781.

MORAVCSIK, M. J., MURUGESAN, M. 1975. Some results on the function and quality of citations. **Social Studies of Science**, 5(1):86–92.

PAN, S. J., YANG, Q. 2010. A Survey on Transfer Learning. **IEEE Transactions on Knowledge and Data Engineering**, 22, 1345-1359.

PETERS, M. E., NEUMANN, M., IYYER, M., GARDNER, M., CLARK, C., LEE, K., ZETTLEMOYER, L. (1802). Deep contextualized word representations. CoRR abs/1802.05365 (2018). **arXiv** preprint arXiv:1802.05365, 42.

PENNINGTON, J., SOCHER, R., MANNING, C. D. 2014, October. Glove: Global vectors for word representation. In **Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)**, pp. 1532-1543.

RAHUTOMO, F., KITASUKA, T., ARITSUGI, M. 2012, October. Semantic cosine similarity. In **The 7th international student conference on advanced science and technology ICAST** (Vol. 4, No. 1, p. 1). South Korea: University of Seoul.

REIMERS, N., GUREVYCH, I. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. **arXiv** preprint arXiv:1908.10084.

ROMAN, M. et al. 2021. Citation intent classification using word embedding, **IEEE Access**, 9, pp. 9982–9995. doi:10.1109/access.2021.3050547.

RUSSELL, S.J., NORVIG, P. 2016. **Artificial Intelligence: A Modern Approach**. Pearson Education Limited, Malaysia.

SNELL, J., SWERSKY, K., ZEMEL, R. 2017. Prototypical Networks for Few-shot Learning. **Advances in Neural Information Processing Systems**, 30.

SWALES, J. 1990. **Genre Analysis: English in Academic and Research Settings**. Cambridge, UK: Cambridge University Press.

SWAYAMDIPTA, S., THOMSON, S., LEE, K., ZETTLEMOYER, L., DYER, C., SMITH, N. A. 2018. Syntactic Scaffolds for Semantic Structures. In **Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing**, pages 3772–3782, Brussels, Belgium. Association for Computational Linguistics.

TAM, D., MENON, R. R., BANSAL, M., SRIVASTAVA, S., RAFFEL, C. 2021. Improving and simplifying pattern exploiting training. In **Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing**, pages 4980–4991, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

TAROS, T., ZOPPO, C., YEE, N., HANNA J., MACGINNIS, C. 2023. Retracted Covid-19 articles: significantly more cited than other articles within their journal of origin. **Scientometrics**. 2023;128(5):2935-2943. doi: 10.1007/s11192-023-04707-4. Epub 2023 Apr 12. PMID: 37101974; PMCID: PMC10089824.

THOYYIBAH, T., HARYONO, W., ZAILANI, A. U., DJAKSANA, Y. M., ROSMAWARNI, N., ARIANTI, N. D. 2023. Transformers in Machine Learning: Literature Review. **Jurnal Penelitian Pendidikan IPA**, 9(9), 604-610.

TOUVRON, H., MARTIN, L., STONE, K., ALBERT, P., ALMAHAIRI, A., BABAEI, Y., ... SCIALOM, T. 2023. Llama 2: Open foundation and fine-tuned chat models. **arXiv** preprint arXiv:2307.09288.

TRIVEDI, G., PHAM, P., CHAPMAN, W., HWA, R., WIEBE, J., HOCHHEISER, H. 2017. An Interactive Tool for Natural Language Processing on Clinical Text. **arXiv** preprint arXiv:1707.01890.

VASWANI, A., SHAZEER, N., PARMAR, N., USZKOREIT, J., JONES, L., GOMEZ, A. N., ... POLOSUKHIN, I. 2017. Attention is all you need. Advances in neural information processing systems, **Advances in Neural Information Processing Systems** 30.

TUNSTALL, L., REIMERS, N., JO, U. E. S., BATES, L., KORAT, D., WASSERBLAT, M., PEREG, O. 2022. Efficient few-shot learning without prompts. **arXiv** preprint arXiv:2209.11055.

WANG, Y., YAO, Q., KWOK, J.T., NI, L.M. 2020. Generalizing from a Few Examples: A Survey on Few-shot Learning. **ACM Comput. Surv.** 53, 3, Article 63 (May 2021), 34 pages. https://doi.org/10.1145/3386252.

WERBOS, P. 1990. Backpropagation through time: what does it do and how to do it. **Proceedings of the IEEE**, vol. 78, no. 10, pp. 1550-1560, Oct. 1990, doi: 10.1109/5.58337.

YIN, H., PAN, S., RUAN, Y., LIM, A. 2020. Learning to Use the Past: A Framework for Time-Aware Few-Shot Learning. **IEEE Transactions on Neural Networks and Learning Systems**.

APÊNDICE A – *SELF-ATTENTION: TRANSFORMERS*

Neste apêndice, apresentaremos os *Transformers* e o mecanismo de *Atenção*, que são essenciais para o desenvolvimento de modelos de linguagem de grande escala (*Large Language Models*, LLM). A ideia principal dessa arquitetura é a utilização de camadas de Auto-Atenção, que serão responsáveis por extrair a relação entre as palavras de uma sentença. Porém, primeiro precisa-se introduzir como funciona o mecanismo de Atenção.

A função de Atenção pode ser entendida como o mapeamento de uma *busca* (*query*) e um conjunto de pares *chave-valor* (*key-value*) à uma *saída* (*output*), onde a busca, as chaves, os valores e saída são vetores (VASWANI et al., 2017). A busca é o vetor que representa o que o modelo está procurando. Já as chaves representam as diferentes partes da entrada. Os valores são representações vetoriais da importância das partes da entrada. E a saída é um vetor que representa a relação entre a saída e as chaves.

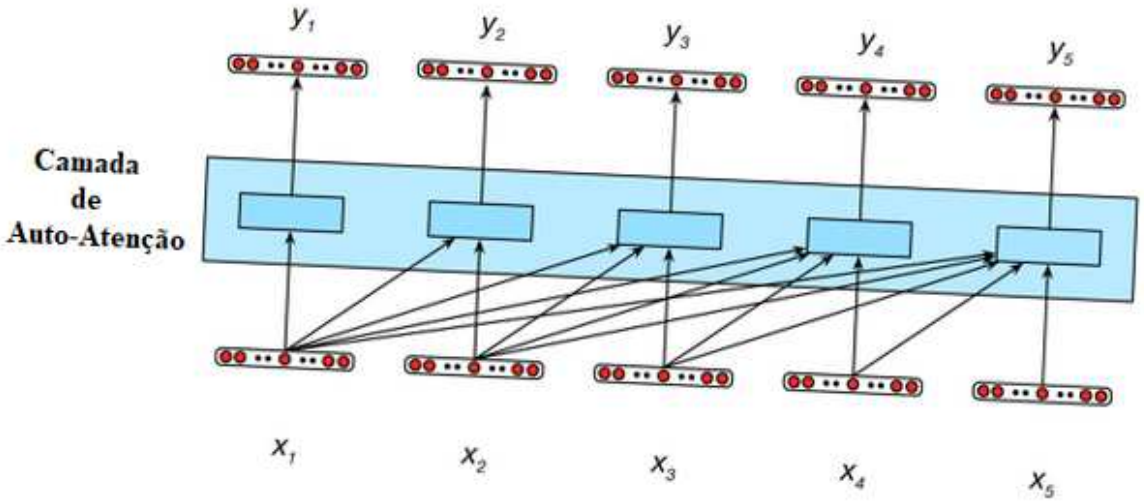


Figura 23: Diagrama de representação das conexões das camadas de *Self-Attention* (adaptado de (JURAFSKY; MARTIN, 2020))

Como é possível observar na Figura 23, uma entrada x_i só tem acesso a entradas x_j, tal que $j < i$, isso permite que o modelo aprenda a realizar o aprendizado sem que tenha conhecimento das palavras posteriores (o que é essencial para a criação de modelos de linguagem). O mecanismo de atenção baseia-se no produto entre dois itens de entrada, e para que o mecanismo de aprendizado possa ser utilizado, foram introduzidas três matrizes de pesos: W^Q, W^K e W^V. Logo, esses pesos serão utilizados nos cálculos das transformações

lineares de *X*, para serem utilizadas em etapas posteriores, conforme apresentado na equação 2.

$$q_i = W^Q x_i; k_i = W^K x_i; v_i = W^V x_i \quad (2)$$

Na equação 2, temos:

- $q_i = W^Q x_i$: Vetor de "*query*" calculado multiplicando a entrada x_i pela matriz de pesos W^Q;
- $k_i = W^K x_i$: Vetor "*key*" calculado, de forma semelhante, multiplicando a entrada x_i pela matriz de pesos W^K;
- $v_i = W^V x_i$: Vetor de "*value*" calculado multiplicando a entrada x_i pela matriz de pesos W^V.

Conforme explicado anteriormente, a pontuação de atenção é definida pelo produto vetorial entre dois vetores. Neste caso, o produto vetorial entre o foco de atenção atual q_i e as chaves k_j de elementos anteriores, conforme apresentado na equação 3.

$$score(x_i, x_j) = q_i . k_j \quad (3)$$

No entanto, esse produto pode resultar em valores muito altos, então é necessário realizar a normalização, de acordo com as equações 4 e 5.

$$\alpha_{ij} = softmax(score(x_i, x_j)) \; \forall \, j \leq i \quad (4)$$

$$= \frac{exp(score(x_i, x_j))}{\sum_{k=1}^{i} exp(score(x_i, x_j))} \; \forall \, j \leq i \quad (5)$$

Nas equações 4 e 5, temos:

- α_{ij}: Relação de normalização dos valores objetivos pelo *score* de acordo com equação 4. O *score* de atenção para a entrada *i* em relação a *j* é exponencializado e dividido pela soma de todos os *scores* exponencializados para a entrada *i*, o que normaliza os *scores* para que eles somem 1.

Por fim, a saída y_i é calculada somando as entradas até o ponto *i* multiplicadas por suas respectivas pontuações α, conforme a equação 6.

$$y_i = \sum_{j \leq i} \alpha_{ij} v_i \quad (6)$$

Para exemplificar, o processo de cálculo da terceira entrada de um modelo pode ser observado na Figura 24.

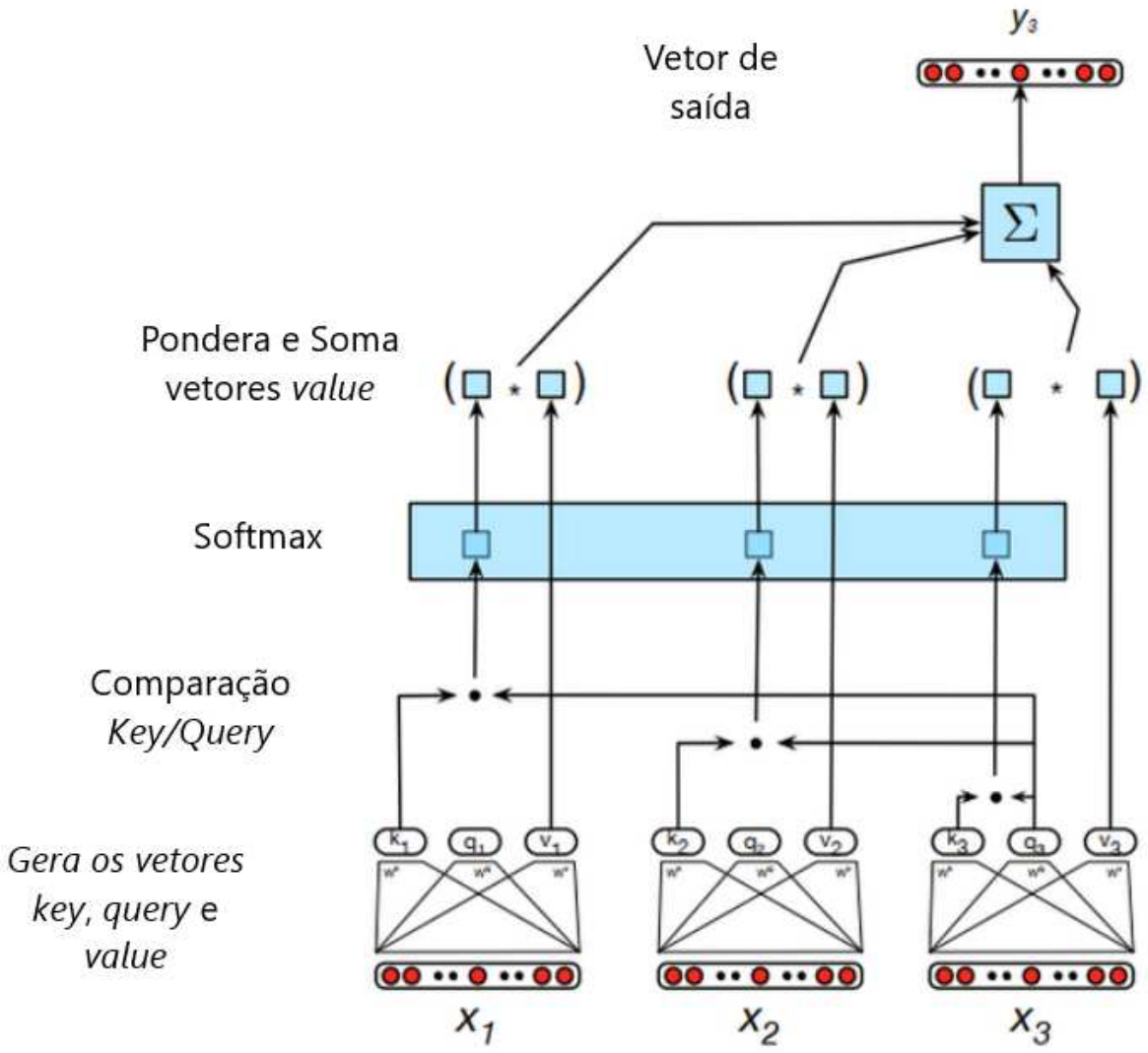


Figura 24: Representação dos cálculos realizados na camada de *Self-Attention* (adaptado de (JURAFSKY; MARTIN, 2020))

Uma segunda grande inovação da arquitetura das redes *Transformers* é que as conexões não estão ligadas de forma sequencial. Assim, é possível realizar o processamento em paralelo, diminuindo de forma considerável o tempo de treinamento. Logo, todo o processo de Auto-Atenção poderia ser resumido da forma apresentada nas equações 7 e 8.

$$Q = W^Q X; K = W^K X; V = W^V X \tag{7}$$

$$SelfAttention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (8)$$

Nas quais, X é a matriz de vetores de entrada. d_q, d_k, d_v são as dimensões de W^Q, W^K e W^V, respectivamente. No entanto, o mecanismo de *Atenção* é um dos componentes que compõem o bloco de transformação (*transformer block*). Assim, esse bloco também é composto por redes *feedforward*, conexões residuais e camadas de normalização como é possível observar na Figura 24.