Universidade Federal de Ouro Preto
Escola de Minas
CECAU - Colegiado do Curso de
Engenharia de Controle e Automação

Luís Gustavo de Oliveira Miranda

Aplicação da Rede Neural Recorrente *Gated Recurrent Unit* na classificação multirrótulo de relações entre estrutura química e atividade biológica

Monografia de Graduação em Engenharia de Controle e Automação

Ouro Preto, 2024

Luís Gustavo de Oliveira Miranda

Aplicação da Rede Neural Recorrente *Gated Recurrent Unit* na classificação multirrótulo de relações entre estrutura química e atividade biológica

Trabalho apresentado ao Colegiado do Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como parte dos requisitos para a obtenção do Grau de Engenheiro de Controle e Automação.

Universidade Federal de Ouro Preto

Orientador: Prof. Dr. Jadson Castro Gertrudes

Coorientador: Profa. Dra. Adrielle de Carvalho Santana

Ouro Preto

2024

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
ESCOLA DE MINAS
DEPARTAMENTO DE ENGENHARIA CONTROLE E AUTOMACAO

FOLHA DE APROVAÇÃO

Luís Gustavo de Oliveira Miranda

Aplicação da Rede Neural Recorrente *Gated Recurrent Unit* na classificação multirrótulo de relações entre estrutura química e atividade biológica

Monografia apresentada ao Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Engenheiro de Controle e Automação

Aprovada em 18 de outubro de 2024

Membros da banca

Dr. Jadson Castro Gertrudes - Orientador - Universidade Federal de Ouro Preto
Dra. Adrielle de Carvalho Santana - Coorientadora - Universidade Federal de Ouro Preto
Erick Vinicius de Araújo - Convidado - Universidade Federal de Ouro Preto
Msc. Geovani Lopes Martins - Convidado - Universidade Federal de Ouro Preto

Jadson Castro Gertrudes, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em XX/10/2024

Documento assinado eletronicamente por **Jadson Castro Gertrudes**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 18/10/2024, às 18:35, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0795667** e o código CRC **F1CC362B**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: 3135591533 - www.ufop.br

Agradecimentos

Gostaria de agradecer primeiramente a Deus pela vida e pela oportunidade de presenciar essa obra magnifica que ele criou. A minha família que sempre me apoiou do jeito que só eles sabem, especialmente a minha mãe, que parecia estar lá a cada passo que eu dava e ao meu pai, que sempre tinha um norte para dar. Ao meu professor e orientador Jadson, que cedeu seu tempo e atenção para me auxiliar no projeto. Para a minha professora e coorientadora Adrielle, que me deu uma ajuda enorme mesmo quando eu cheguei sem aviso prévio. A minha amiga Anna, que presenciou essa jornada de perto. E agradecer a todos que me agraciaram com sua presença durante esses anos e que aqui não foram mencionados diretamente, pois não existe contribuição insignificante, apenas contribuição não apreciada.

"Quem ri de si mesmo nunca fica sem motivos para rir."

— Epictetus.

Resumo

O uso de técnicas computacionais tornou-se vital na descoberta de reagentes e medicamentos, especialmente quando alinhado a métodos modernos como o aprendizado de máquina. Este estudo explora a aplicação do aprendizado profundo, utilizando o modelo *Gated Recurrent Unit* (GRU), na previsão da atividade química de compostos biológicos de múltiplos alvos por meio da modelagem entre estrutura química e atividade biológica. A pesquisa envolve um extenso pré-processamento de dados, incluindo a limpeza e tokenização de sequências SMILES, seguido da transformação em formatos adequados para entrada no modelo GRU. Os resultados demonstram que o modelo GRU, apesar de sua arquitetura simplificada, pode prever efetivamente interações moleculares. Embora não atinja a mesma precisão de modelos mais complexos e com uso intensivo de dados, ele abre possibilidades para processos de descoberta mais rápida e econômica de medicamentos. Este estudo não só destaca o potencial das redes GRU em quimioinformática, mas também defende a exploração de modelos mais simples e eficientes em campos tradicionalmente dominados por abordagens mais complexas.

Palavras-chaves: Inteligência Artificial. Processamento de Linguagem Natural. Análise entre estrutura química e atividade biológica. Aprendizado de Máquina. Gated Recurrent Unit.

Abstract

The use of computational techniques has become vital in reagent and drug discovery, especially when combined with modern methods such as machine learning. This study explores the application of deep learning, using the Gated Recurrent Unit (GRU) model, to predict the chemical activity of multi-target biological compounds by modeling chemical structure and biological activity. The research involves extensive data preprocessing, including cleaning and tokenizing SMILES sequences, followed by transformation into formats suitable for input to the GRU model. The results demonstrate that the GRU model, despite its simplified architecture, can effectively predict molecular interactions. Although it does not achieve the same accuracy as more complex and data-intensive models, it opens up possibilities for faster and more cost-effective drug discovery processes. This study not only highlights the potential of GRU networks in chemoinformatics, but also advocates the exploration of simpler and more efficient models in fields traditionally dominated by more complex approaches.

Keywords: Artificial Intelligence. Natural Language Processing. Structure Activity Relationship. Machine Learning. Gated Recurrent Unit.

Lista de ilustrações

Lista de abreviaturas e siglas

UFOP	Universidade Federal de Ouro Preto
CADD	*Computer-Aided Drug Designer* (Designer de Medicamentos Auxiliado por Computador)
SMILES	*Simplified Molecular Input Line Entry System* (Sistema Simplificado de Entrada Linear de Moléculas)
RNN	Redes Neurais Recorrentes
GRU	*Gated Recurrent Unit*
LSTM	*Long Short-term Memory network*
QSAR	*Quantitative structure–activity relationship* (Modelagem de Relações Estrutura-Atividade)
NLP	*Natural Language Processing* (Processamento de linguagem natural)
GNN	*Graph Neural Networks* (Redes neurais de grafo)

Sumário

1 Introdução

1.1 Justificativa e Relevância

Atualmente, é indiscutível a importância e a presença da computação e das técnicas computacionais no campo da Química, especialmente na produção e identificação de fármacos. De acordo com Van Vlijmen, Desjarlais e Mirzadegan (2017), embora seja difícil quantificar com precisão, o impacto significativo da química computacional é evidenciado pelo crescente número de patentes que incluem pelo menos um componente desenvolvido por meio do *Computer-Aided Drug Design* (CADD, ou Desenvolvimento de Fármacos Auxiliado por Computador, em tradução livre).

O CADD é parte central das equipes de desenvolvimento e está envolvido em todas as etapas do processo de pesquisa e produção (VAN VLIJMEN; DESJARLAIS; MIRZADEGAN, 2017). Suas contribuições abrangem a seleção de bibliotecas de triagem, influenciando a síntese de moléculas e aprimorando a tomada de decisões por meio da integração e visualização de dados.

Com o avanço da tecnologia e das ferramentas associadas à inteligência artificial, surge a oportunidade de testar o desempenho dessas ferramentas para obter resultados de maneira mais rápida e precisa. Jarada, Rokne e Alhajj (2020) discutem que o reposicionamento computacional de medicamentos pode ser de enorme benefício para a humanidade ao descobrir novas indicações para medicamentos aprovados.

Além disso, o contexto que nos permite realizar essa interação e quantificar seu resultado é o *Quantitative Structure-Activity Relationship* (QSAR), um método para construir modelos químicos preditivos de atividades biológicas com base em suas estruturas moleculares (ISARANKURA-NA-AYUDHYA et al., 2009).

Conforme explicado por Patel e Shah (2022), o processo de descoberta e desenvolvimento de medicamentos utilizando aprendizado de máquina pode ser resumido nos seguintes passos: utilização de dados para treinar um modelo, teste e validação dos resultados obtidos pelo modelo com compostos reais e, com os novos dados, treinamento de um modelo mais complexo ou preciso, em um ciclo repetido até a obtenção do resultado desejado, como ilustrado na Figura 1.

O emprego de técnicas de aprendizado de máquina não é novidade na área, como apontado por Dara, Dhamercherla, Jadav et al. (2022): tarefas de regressão, como a predição de sucesso de testes clínicos, utilizam o algoritmo *Random Forest*, que combina uma grande quantidade de árvores de decisão para realizar previsões, e a regressão linear simples, que calcula a equação que melhor se ajusta aos dados de treino $\mathbf{x}$ para uma saída

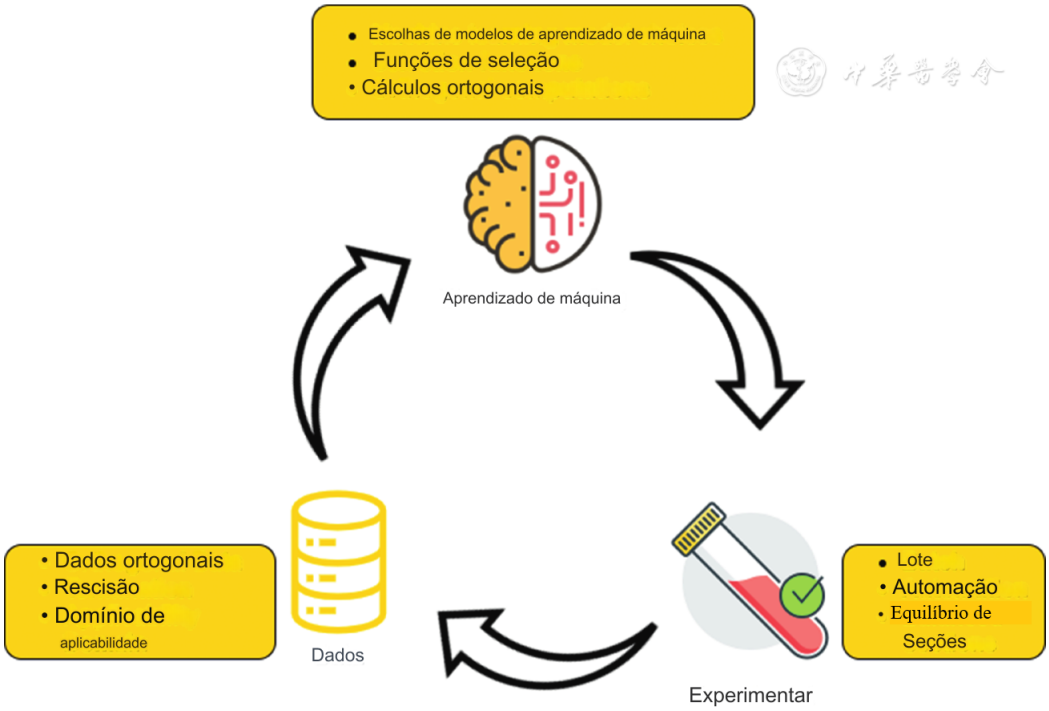

Figura 1 – Descrição do processo de identificação de uma nova substância utilizando aprendizado de máquina. Fonte: Patel e Shah (2022) (modificado).

y.

Tarefas mais complexas de classificação, como o uso do processamento de linguagem natural para associar alvo-doença-medicação, por exemplo, já fazem uso de métodos como o Kernel, a classificação de Bayes ou o *K-nearest neighbors*. Pesquisas mais recentes já utilizam o perceptron multicamadas e até a aprendizagem profunda, apesar das dificuldades observadas ao utilizar esses modelos, como citado pelo mesmo autor. Apesar desses exemplos, percebe-se que o campo ainda oferece uma ampla gama de possibilidades que não foram completamente exploradas, seja pela falta de dados suficientes ou pelas limitações nos períodos de teste.

Um modelo pouco explorado, e que será o utilizado neste trabalho, é o GRU (*Gated Recurrent Units*), uma rede neural recorrente introduzida pela primeira vez por Cho et al. (2014), que busca solucionar o problema do gradiente explosivo/desvanecido e ser mais eficiente no tratamento de memórias de curto e longo prazo do que a LSTM (*Long Short-Term Memory*).

É válido ressaltar que a abordagem adotada neste trabalho, diferentemente das mencionadas anteriormente e mais comumente encontradas na literatura, envolverá a utilização de dois agentes biológicos como alvos de predição. Conforme mencionado indiretamente por Atas Guvenilir e Doğan (2023), modelos mais complexos tendem a alcançar resultados mais precisos na predição de múltiplos alvos, no entanto, esses modelos

apresentam desafios adicionais, como a necessidade de maior volume de dados, maior poder computacional e maior rigor nas etapas de pré-processamento. Mesmo assim, estão sujeitos a problemas de precisão quando a base de dados é muito dispersa, criando um cenário complexo para a modelagem.

Para a utilização dos dados químicos no aprendizado de máquina, e também para uso mais cotidiano, o ramo da química apresenta várias ferramentas que nos permitem categorizar de maneira desambigua e reproduzível, que pode ser então utilizado para a realização das mais diversas operações. Para o método QSAR, o mais comum é o conhecido como *Descriptors*, onde centenas ou até mesmo milhares de descritores são calculados com base nas qualidades físico-químicas da molécula, sendo então posteriormente alimentados a um modelo, como uma árvore de decisão (TODESCHINI; CONSONNI, 2000).

Há também o método de representação em grafos, como é observado pelo autor Gilmer et al. (2017), que utiliza grafos para realizar uma representação mais estrutural da cadeia, ganhando popularidade graças aos modelos do tipo GNN (*Graph Neural Networks*), assim como vários outros. É válido afirmar que cada tipo de modelo tenderá a uma representação mais específica, que facilita de alguma forma a preparação de dados e a manutenção de informações importantes (estrutura, conexões, relações, características físico-químicas).

Uma das técnicas que será utilizada e que receberá uma explicação mais aprofundada no próximo capítulo é o SMILES (*Simplified Molecular Input Line Entry System*, ou Sistema Simplificado de Entrada Linear de Moléculas, em tradução livre). Esta técnica, embora relativamente antiga, permite reescrever a estrutura molecular em uma sequência de caracteres, facilitando sua transformação e posterior alimentação em um modelo de aprendizado profundo, conforme demonstrado por Weininger (1988a).

A obtenção de um modelo funcional possibilitará ampliar as oportunidades para a descoberta de compostos reagentes em alvos biológicos, acelerando a produção e o teste de medicamentos, além de reduzir custos. Além disso, chamará a atenção para o uso de modelos mais simples e eficientes, como a GRU, em vez de modelos mais complexos e custosos, como a LSTM, e técnicas simplificadas.

1.2 Objetivos

Este trabalho tem como objetivo principal analisar o desempenho da arquitetura de rede neural recorrente GRU, em combinação com sequências SMILES, no processo de classificação de múltiplos alvos na análise QSAR. Para que o objetivo geral seja alcançado, os seguintes objetivos específicos serão definidos:

- Avaliar o desempenho da arquitetura GRU combinada com sequências SMILES na

mesma tarefa de classificação multi-alvo em QSAR.

- Analisar a capacidade da arquitetura em extrair características relevantes das sequências SMILES.
- Investigar as vantagens e limitações da arquitetura GRU no contexto da classificação multirrótulo em QSAR, considerando fatores como complexidade computacional e capacidade de generalização.

1.3 Materiais e Métodos

Neste trabalho, é proposta a criação de uma rede neural recorrente GRU (Gated Recurrent Unit), treinada de maneira supervisionada, com o objetivo de prever se um dado composto químico será reagente ou não em um determinado agente biológico. Esta seção detalha as principais ferramentas e técnicas utilizadas para atingir esse objetivo, bem como os métodos aplicados para a observação e validação dos resultados. Portanto, este trabalho abrange as seguintes ações:

- Criação da rede GRU, explicando sua arquitetura e parâmetros usados.
- Detalhamento dos dados utilizados e seu pré-processamento para poder alimentar o modelo;
- Explicações dos parâmetros de treinamento e das métricas para observar os resultados;
- Visualização e interpretação dos resultados;
- Técnicas de otimização e de interpretação mais profunda;
- Comentar possíveis melhorias que podem ser realizadas para projetos futuros.

Portanto, este trabalho buscará utilizar o modelo GRU, devido à sua simplicidade, em conjunto com técnicas QSAR, evitando, no entanto, o uso de descritores comumente encontrados na literatura. Em vez disso, será adotada a técnica SMILES, especialmente compatível com o modelo, o que simplificará a estrutura do modelo, bem como a tarefa, notoriamente desafiadora devido à necessidade de grande volume de dados e intenso processamento computacional. Além disso, este estudo explorará as possibilidades de combinar este método relativamente novo com um modelo ainda pouco explorado na área, contribuindo para avanços na predição de múltiplos rótulos.

1.4 Organização e estrutura

O Capítulo 2 define alguns conceitos necessários para uma melhor compreensão deste trabalho, como o modelo que será utilizado e alguns métodos de transcrição de moléculas.

O Capítulo 3 apresenta o preparo inicial, detalhando os dados que serão utilizados e os tratamentos necessários, tanto para transformar os dados de uma maneira compreensível para a máquina quanto para o modelo de aprendizagem profunda.

No Capítulo 4 , são inicialmente discutidos os resultados parciais com uma breve análise inicial. Em seguida, são feitas modificações pontuais em certos hiperparâmetros, visando melhorar a capacidade preditiva do modelo. Após isso, os resultados finais com as modificações aplicadas são analisados, acompanhados de uma breve discussão sobre as mudanças observadas. Ainda neste capítulo, é realizado um treinamento ponderado, levando em consideração a diferença na distribuição dos dados para cada rótulo, e é feita uma comparação entre o modelo treinado com esse ajuste e o melhor modelo obtido anteriormente.

Por fim, o Capítulo 5 apresenta uma síntese do projeto, analisando seu desempenho final e se o objetivo proposto foi alcançado, além de comentários pertinentes a projetos futuros.

2 Revisão de literatura

2.1 Redes Neurais Recorrentes

Dentro do escopo do aprendizado de máquina, e aprofundando nos modelos de aprendizagem profunda, tem-se as RNNs (Redes Neurais Recorrentes). De acordo com Abiodun et al. (2018), uma RNN pode ser descrita como uma rede neural que possui a capacidade de "memória"; ou seja, a saída (*output*) de uma célula não apenas produz um resultado, mas também alimenta a entrada da próxima célula.

Esse comportamento pode ser observado na Figura 2: para calcular uma determinada saída O, leva-se em consideração a entrada X multiplicada pelo peso U, juntamente com o estado anterior h, que funciona como uma espécie de memória. Esses dois fatores permitem calcular o estado atual h e, então, após multiplicar pelo peso V, obter a saída.

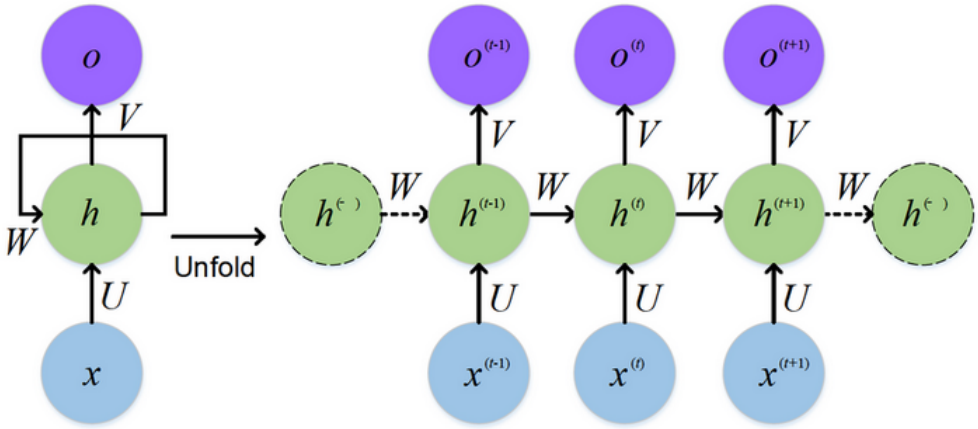

Figura 2 – Uma rede neural recorrente, pode ser observado que cada portão alimenta o próximo. Fonte: Feng et al. (2017).

Essa característica permite que as RNNs lidem com dados sequenciais, nos quais uma entrada influencia o comportamento da próxima, sejam sequências de palavras, números ou dados temporais. Essa capacidade é extremamente relevante para realizar predições em modelos químicos, pois a entrada será uma sequência de caracteres resultante da técnica SMILES.

No entanto, uma análise mais aprofundada do modelo RNN revela um problema crítico para tarefas mais complexas. Como explicado por Ribeiro et al. (2020), ao longo das iterações, o gradiente propagado tende a crescer indefinidamente ou a desaparecer completamente, resultando em instabilidade no treinamento: no primeiro caso, os valores

explodem; no segundo, o gradiente torna-se tão pequeno que o aprendizado praticamente cessa. Esse problema é conhecido como o *vanishing/exploding gradient* (problema do explodimento/desvanecimento do gradiente). Portanto, é necessário um modelo mais robusto que consiga lidar com sequências maiores de entradas.

2.1.1 Gated Recurrent Units - GRU

Um modelo que não só resolve o problema do *vanishing/exploding gradient*, mas também traz diversas vantagens, é a *Gated Recurrent Unit* (GRU), uma arquitetura de rede neural recorrente que utiliza estruturas chamadas de *gates* (portões) para realizar operações matemáticas mais complexas (REHMER; KROLL, 2020).

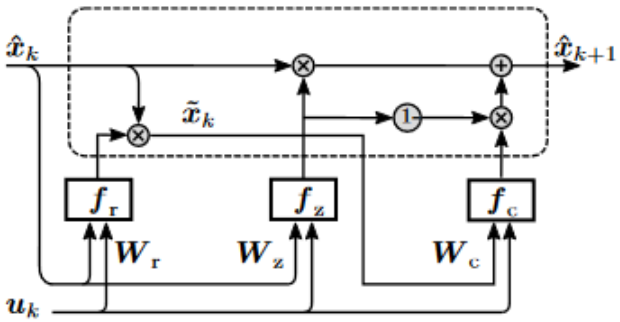

Figura 3 – O modelo GRU. Note que a complexidade é superior ao modelo RNN padrão. Fonte: Rehmer e Kroll (2020).

Como pode ser observado na Figura 3, o modelo GRU possui funções matemáticas que permitem calcular os valores de seus dois principais *gates*: o *Reset Gate* e o *Update Gate* (REHMER; KROLL, 2020).

O *Reset Gate* calcula quanto da memória anterior será utilizado na computação do novo estado. Sua expressão matemática é:

$$r_k = \sigma(W_r \cdot [x_{k-1}, u_k])\,. \tag{2.1}$$

onde:

- σ é a função de ativação sigmoide.
- W_r é a matriz peso para o *Reset Gate*.
- x_{k-1} é o *Hidden State* anterior.
- u_k é a entrada atual.

Há também o *Update Gate*, que determina quanto das informações internas serão passadas para o próximo estado, ou seja, é ele que controla o quanto da memória antiga (*Hidden State*) irá ser mantida para próxima interação e quanto da para a memória nova será utilizada para calcular a saída (*Candidate Hidden State*). Ele possui a seguinte fórmula:

$$z_k = \sigma(W_z \cdot [x_{k-1}, u_k])\,. \tag{2.2}$$

onde:

- W_z é a matriz peso para o *Update Gate*.
- σ é a função de ativação sigmoide.
- x_{k-1} é o *Hidden State* anterior.
- u_k é a entrada atual.

É por meio desses dois *gates* que o modelo GRU não só resolve o problema do gradiente que as RNNs tradicionais apresentam, mas também permite um melhor controle sobre a memória de curto e longo prazo. Outras vantagens desse modelo incluem maior eficiência quando comparado a outras RNNs, como a LSTM, e, graças à sua arquitetura, um controle mais robusto sobre a memória a longo prazo, permitindo detectar e aprender dependências mais distantes (RAVANELLI et al., 2018).

Além desses *gates*, o modelo GRU possui o *Candidate Hidden State* e o *Hidden State*, que enquanto não necessariamente *gates*, são usados para calcular a saída da célula.

O *Candidate Hidden State* é a nova memória candidata, calculada a partir do estado anterior e da entrada atual, ajustados pelo *Reset Gate*, em outras palavras, é uma proposta de nova memória a ser utilizada. Sua expressão matemática é:

$$\tilde{x}_k = \tanh(W_h \cdot [r_k \odot x_{k-1}, u_k])\,. \tag{2.3}$$

onde:

- tanh é a função de ativação tangente hiperbólica.
- W_h é a matriz peso para o *Candidate Hidden State*.
- r_k é o valor do *Reset Gate*.
- $\odot$ é o produto de Hadamard (ou multiplicação elemento a elemento).
- x_{k-1} é o *Hidden State* anterior.
- u_k é a entrada atual.

Por fim, o novo *Hidden State* é uma combinação linear entre o estado anterior e o estado candidato, ponderada pelo *Update Gate*, sendo o seu valor final a saída da célula. Sua equação é:

$$x_k = z_k \odot x_{k-1} + (1 - z_k) \odot \tilde{x}_k \,. \tag{2.4}$$

onde:

- z_k é o valor do *Update Gate*.
- x_{k-1} é o *Hidden State* anterior.
- $\tilde{x}_k$ é o *Candidate Hidden State*.
- $\odot$ é o produto de Hadamard (multiplicação elemento a elemento).

Os quatro *gates* funcionam em sincronia para calcular a saída da célula: primeiramente, é calculado o valor do *Reset Gate*, controlando a influência da memória passada; passando em seguida para computar o *Candidate Hidden State*, levando em consideração o valor do *Reset Gate* e da entrada atual; em seguida, o *Update Gate* decidirá quanto da memória antiga (*Hidden State* anterior) será mantida e quanto da memória nova (*Candidate Hidden State*) será utilizada, calculando então o valor do *Hidden State* atual, que será o valor de saída ao mesmo tempo em que será carregado para futuras iterações (CHO et al., 2014).

2.2 Modelos de aprendizado *multilabel*

Os problemas mais clássicos da literatura para o aprendizado de máquinas são aqueles conhecidos como *single label*, isto é, quando um exemplo dos dados pertence a um único rótulo dentro de um número finito de rótulos. Quando um exemplo pode pertencer a dois ou mais rótulos, o problema passa a ser da categoria *multilabel* (multirrótulo, em tradução livre) (TSOUMAKAS; KATAKIS, 2009).

No contexto químico, uma aplicação prática do aprendizado multirrótulo ocorre quando um composto é eficaz contra múltiplos alvos biológicos. Por exemplo, um medicamento (determinada estrutura química) pode exibir propriedades antivirais (atividade biológica) eficazes contra o vírus influenza e o vírus da hepatite (alvo biológico).

Para lidar com problemas dessa categoria, foram desenvolvidas algumas metodologias, que podem ser divididas em duas principais categorias: métodos de transformação de problema, que consistem na manipulação e reorganização dos dados para que adquiram as características de um problema *single label*, e métodos de adaptação de algoritmos, nos

quais o modelo é adaptado para lidar com o formato de uma saída multirrótulo (MADJAROV et al., 2012).

A abordagem utilizada neste estudo, similarmente aplicada por Kumar, Kumar, Dev et al. (2023), é conhecida como *binary relevance with label powerset* e pertence à categoria de reorganização de dados. Esse método considera cada rótulo independentemente como um problema de classificação binária separado, criando novas classes com base em combinações exclusivas dos rótulos originais.

Por exemplo, se dois rótulos, A e B, representam se um medicamento é eficaz contra o Vírus 1 e o Vírus 2, respectivamente, o espaço de saída seria transformado em quatro classes: A/B(0,0) (ineficaz contra ambos os vírus), A/B(1,0) (eficaz apenas contra o Vírus 1), A/B(0,1) (eficaz apenas contra o Vírus 2) e A/B(1,1) (eficaz contra ambos os vírus). Essa transformação permite que o modelo aprenda as interações entre vários rótulos enquanto ainda aproveita as técnicas de classificação binária.

2.3 Representação de Moléculas com Cadeias SMILES

Com o vasto número de moléculas e combinações, faz-se necessário uma estrutura que permita representar fielmente uma cadeia molecular, que seja única e que mantenha as características da molécula, com uma perda mínima de informação. Uma das maneiras mais comuns para tal procedimento é a fórmula estrutural, que de maneira simplificada pode ser explicada como um "desenho" da molécula no espaço e de suas ligações (GOODWIN, 2007).

Apesar de ser o método de representação mais conhecido e usado, ele é insuficiente para o propósito de aprendizado de máquina quando analisado o modelo que será usado, visto que não é trivial transcrevê-lo diretamente em uma forma que possa ser alimentada a uma RNN, sendo a fórmula SMILES (*Simplified Molecular Input Line Entry System*) mais interessante nesse sentido. O SMILES consiste em representar a cadeia estrutural química em uma sequência de caracteres usando ASCII, e por se tratar de linguagem natural, o procedimento para a alimentação no modelo de aprendizado de máquinas acaba tornando-se mais simplificado (WIGH; GOODMAN; LAPKIN, 2022).

Nesse método, como pode ser visto na Figura 4, átomos são representados respectivamente pelos seus símbolos atômicos, ganhos ou perdas de elétrons são indicados com o símbolo + ou - junto ao símbolo atômico, todos dentro de um colchete, e ligações simples, duplas, triplas e aromáticas são representadas pelos símbolos —, =, #. Há outras regras para cadeias mais complexas, mas essa simplificação é suficiente para demonstrar como a sequência de caracteres é criada.

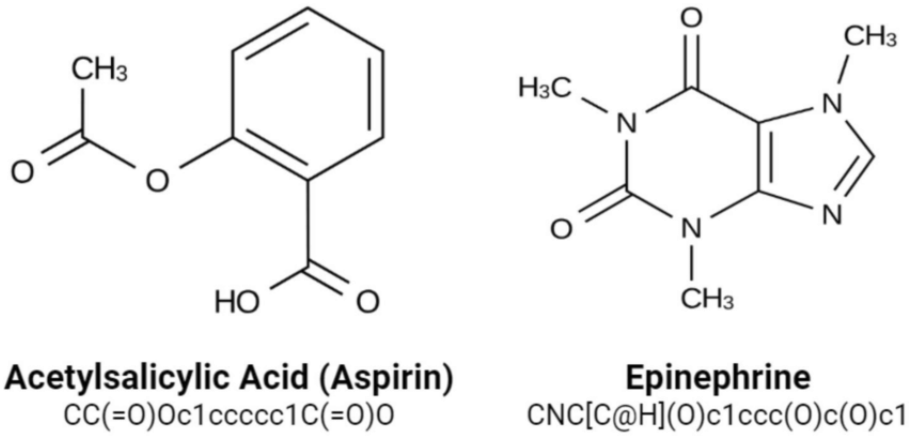

Figura 4 – Uma demonstração da transcrição do modelo estrutural para o modelo SMILES. Fonte: Yasonik (2020).

Como informado por Weininger (1988b), o método SMILES é considerado o melhor método para a interação humano-máquina: é de fácil interpretação humana, criação e manutenção, e pode ser rapidamente interpretado pelos computadores graças à sua compactação e por possuir uma sintaxe que permita a compreensão da cadeia em apenas uma interação. A versão utilizada em um primeiro momento para o treinamento e testes será o *Standard* SMILES, a versão padrão e mais conhecida.

2.4 QSAR - Modelagem de Relações Estrutura-Atividade

A análise *QSAR* (Relação Quantitativa Estrutura-Atividade, em tradução livre) permite predizer a reação de compostos com base em sua estrutura molecular (ROY; KAR; DAS, 2015). Essa abordagem na química computacional e na biologia começou a ser utilizada formalmente por volta de 1960 e teve seu crescimento intensificado nos anos 1970 e 1980. Graças à sua longa história e desenvolvimento, esse método passou por décadas de aperfeiçoamento (DEARDEN, 2017).

No início, eram utilizados métodos computacionais convencionais para realizar os cálculos e análises sobre a relação. Com o desenvolvimento de métodos de aprendizado profundo e técnicas de transcrição de moléculas, é possível utilizá-los para obter resultados mais precisos utilizando menos dados de entrada (DEARDEN, 2017).

Como pode ser observado na Figura 5, o desenvolvimento começa com os dados que serão utilizados na predição, obtidos por meio de experimentos e descritores moleculares (números que descrevem as características da molécula). De posse desses dados, um modelo preditivo é criado. Com o modelo treinado, ele é então utilizado para predizer a saída quando se utilizam novos descritores cuja atividade é desconhecida.

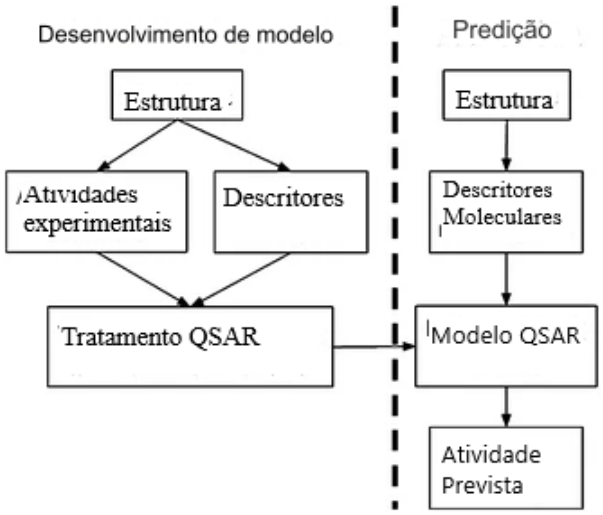

Figura 5 – A ordem dos passos realizados para a utilização do QSAR, desde o desenvolvimento do modelo até sua utilização. Fonte: Patel, Noolvi, Sharma et al. (2014) (modificado).

2.5 Aplicações de Aprendizado Profundo na Quimioinformática

Como mencionado anteriormente, a utilização de aprendizado profundo na química não é algo inédito, tampouco sua aplicação na descoberta de medicamentos e cadeias moleculares. Portanto, é útil obter inspirações em trabalhos anteriores e comparar resultados.

Inicialmente, tem-se a obra de Chakravarti e Alla (2019), que parte de uma ideia similar. A proposta nesse trabalho utiliza uma cadeia SMILES tokenizada com um padrão próprio, evitando a divisão padrão, e a entrada é apenas a cadeia molecular, sem descritores ou outras ferramentas comumente usadas na modelagem QSAR. Essa obra é, portanto, a principal inspiração para a comparação dos resultados, pois realizou tarefa semelhante ao criar um modelo neural recorrente para prever reações de agentes biológicos utilizando SMILES, mas com um modelo mais complexo, uma base de dados mais ampla e predizendo apenas um agente.

Além disso, tem-se o artigo de Shiri et al. (2023), que aponta as vantagens e desvantagens das arquiteturas LSTM e GRU em um contexto mais geral, ressaltando que o modelo GRU possui uma arquitetura simplificada, demandando menos tempo e, dependendo do problema em questão, é capaz de ter um desempenho superior ao da LSTM.

O autor Shtar et al. (2023) utiliza o GRU para realizar predições sobre a interação de duas cadeias SMILES diferentes, tentando determinar se haverá uma reação indesejada ou não, obtendo resultados extremamente precisos. Isso sugere otimismo quanto à utilização desse modelo em questões químicas. Essa publicação permite perceber algumas nuances na

utilização da cadeia SMILES como entrada para o modelo GRU e na aplicação do mesmo para problemas de classificação, que é o caso do presente projeto, apesar de seu objetivo geral ser diferente.

Por fim, cita-se uma obra recente de Silva (2024) que apresenta um estudo ligado diretamente à utilização de múltiplos alvos no contexto da predição de alvos biológicos, destacando algumas das dificuldades em trabalhar com predições mais complexas, como a necessidade de uma quantidade maior de dados e maior poder de processamento. Apesar de utilizar modelos especializados para lidar com o problema, em vez de abordagens mais simples, a obra possui uma relevância considerável devido à sua discussão aprofundada sobre a complexidade e metodologia na predição de múltiplos alvos.

3 Desenvolvimento

Para o desenvolvimento deste projeto, segue-se o passo a passo ilustrado no fluxograma da Figura 6, inicialmente com um foco maior na escolha e no tratamento de dados, seguido pela configuração do modelo e dos hiperparâmetros.

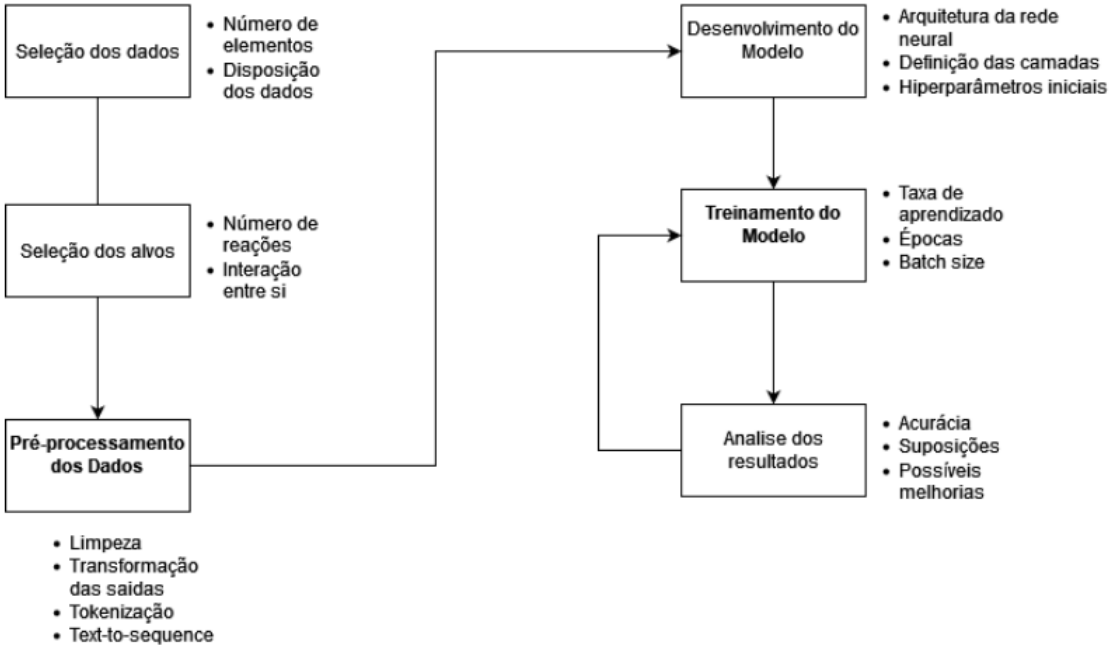

Figura 6 – Fluxograma do experimento, desda seleção de dados até a analise dos resultados.

Primeiramente, a seleção dos dados focou em escolher os agentes biológicos que possuíam o maior número de reações dentro do *dataset* e que apresentavam alguma similaridade química, analisada por meio do prefixo. Em seguida, os dados passaram por uma etapa de pré-processamento, que incluiu a limpeza e a transformação das sequências de entrada. As etapas seguintes envolveram o desenvolvimento do modelo, em que a arquitetura da rede neural foi definida inicialmente seguindo os padrões das técnicas de processamento de linguagem natural, e o treinamento do modelo, utilizando diferentes configurações de hiperparâmetros. Por fim, a análise dos resultados avaliou a precisão e as possíveis melhorias no desempenho.

Todas as etapas serão detalhadas nos subcapítulos a seguir.

3.1 Visualização dos Dados

Os dados que serão utilizados para o projeto são provenientes da base Toxic21, que contém medições qualitativas de toxicidade em 12 alvos biológicos, incluindo receptores nucleares e vias de resposta ao estresse.

Como pode ser observado na Figura 7, a base de dados contém os alvos biológicos, a identificação pelo mol (Mol_id) e a cadeia SMILES, sendo possível visualizar 8 dos 12 alvos biológicos na imagem. Ela possui, ao todo, 7.831 elementos distintos, sendo que o conteúdo apresenta 3 possibilidades: 0 para inativo, 1 para ativo e em branco para inconclusivo. Cada alvo biológico possui uma dessas 3 opções para uma dada cadeia SMILES.

NR-ER	NR-ER-LBD	NR-PPAR-gamma	SR-ARE	SR-ATAD5	SR-HSE	SR-MMP	SR-p53	mol_id	smiles
	0	0	1	0	0	0	0	TOX3021	CCOc1ccc2nc(S(N)(=O)=O)sc2c1
0	0	0		0		0	0	TOX3020	CCN1C(=O)NC(c2ccccc2)C1=O
			0		0			TOX3024	CC[C@]1(O)CC[C@H]2[C@@H]3CCC4=CCCC[C@@H]
0	0	0		0		0	0	TOX3027	CCCN(CC)C(CC)C(=O)Nc1c(C)cccc1C
0	0	0	0	0	0	0	0	TOX20800	CC(O)(P(=O)(O)O)P(=O)(O)O
0	0	0		0	0	0	0	TOX5110	CC(C)(C)OOC(C)(C)CCC(C)(C)OOC(C)(C)C
0	0	0	0	0	0	0	0	TOX6619	O=S(=O)(Cl)c1ccccc1
1			1	0	1	0	1	TOX25232	O=C(O)Cc1cc(I)c(Oc2ccc(O)c(I)c2)c(I)c1
0	0	0	0	0	0		0	TOX22514	OC[C@H](O)[C@@H](O)[C@H](O)CO
			0		0			TOX22517	CCCCCCCC(=O)[O-].CCCCCCCC(=O)[O-].[Zn+2]

Figura 7 – Disposição dos dados no *dataframe* antes de qualquer modificação.

Como o conjunto possui uma grande quantidade de alvos biológicos, seria extremamente difícil criar um modelo que possua a capacidade de computar todos simultaneamente em termos de capacidade de processamento. Portanto, serão selecionados apenas dois agentes, com base no número de dados disponíveis. Maiores detalhes serão apresentados nas próximas seções.

3.2 Limpeza dos dados

Como já se sabe a distribuição dos valores, a verificação de nulls e outros NaN pode ser ignorado. Os espaços vazios são considerados como “Inconclusivo”, portando serão substituídos por -1, facilitando a visualização. Também serão removidos nesse primeiro momento as duas ultimas colunas, já que não é conveniente utiliza-las para visualização. Isso é feito no trecho de código a seguir.

```
# Preenchendo valores nulos com -1
df1 = df1.fillna(-1)

# Obter as colunas excluindo as duas últimas
smiles = df1[['smiles']]
colunas_a_analisar = df1.columns[:-2]
```

Em seguida, itera-se para poder melhor observar a distribuição:

```
# Criar um dicionário para armazenar os resultados
resultados = {}

# Iterar sobre as colunas
for coluna in colunas_a_analisar:
    # Contar as ocorrências de cada classe na coluna
    contagem_classes = df1[coluna].value_counts()

    # Armazenar os resultados no dicionário
    resultados[coluna] = contagem_classes

# Exibir os resultados
print("\nContagem de Classes por Coluna:")
for coluna, contagem_classes in resultados.items():
    print(f"\nColuna: {coluna}")
    print(contagem_classes)
```

Feito isso, pode ser facilmente constatado que a maioria das reações é de "não reagente", possuindo uma disparidade extremamente forte de até 15 para 1 em alguns casos, o que torna a etapa de tratamento de dados essencial.

3.3 Transformação dos dados

Primeiramente, precisa-se realizar uma limpeza nos dados, deixando apenas as informações úteis, ao mesmo tempo que busca-se selecionar quais agentes biológicos possuem uma quantidade de dados adequada para poder realizar o treino do modelo.

O primeiro passo é identificar quais cadeias SMILES não possuem reação com algum agente biológico para logo em seguida serem removidas. Se a cadeia não possui alguma reação, ela então não conterá informação que poderá ser utilizada para o treinamento.

Para isso, será criado um outro dataframe temporário que irá realizar a contagem de reações ativas (rótulos com o valor 1) que cada SMILES obteve. Isso pode ser feito transformando todas as outras (não reativas e inconclusivas) em 0, somando a linha inteira e adicionando esse resultado no final do dataframe original, em uma nova coluna. Após isso, será checada linha por linha e então removida a cadeia do dataframe original.

```
# Exclui as duas últimas linhas e a coluna 'SMILES' para operações numéricas
df2 = df1.iloc[:-2]

# Certifique-se de que todas as colunas, exceto 'SMILES', sejam numéricas, substituindo valores não numéricos por NaN
df2_numeric = df2.drop(columns=['smiles']).apply(pd.to_numeric, errors='coerce').fillna(0)

# Cria uma nova coluna 'TotalReac' que é a soma de todas as outras colunas
df2['TotalReac'] = df2_numeric.sum(axis=1)

# Remover Nones
def remover(row):
    if row['TotalReac'] == 0:
        row['smiles'] = None
    return row

# Aplique a função a cada linha usando apply
df_modificado = df2.apply(remover, axis=1)

# Remova as linhas onde o valor da coluna 'SMILES' é None. Isso deixa apenas as moléculas que tiveram alguma reação
df_modificado = df_modificado.dropna(subset=['smiles'])
```

Agora, restando apenas as sequências que possuem informação, são selecionados os dois agentes que serão utilizados para a aprendizagem e predição. Após analisar os agentes disponíveis, foram selecionados 'SR-ARE' e 'SR-MMP' como objetos de análise, já que possuíam uma maior quantidade de reações ativas, facilitando o treinamento e a limpeza de dados.

Portanto, são extraídas as colunas de interesse ('SR-ARE', 'SR-MMP' e 'smiles'), são removidas as reações inconclusivas, já que não serão úteis para os testes e poderão gerar forte ruído devido ao baixo número de dados, e, por fim, são transformadas as colunas para *Multi-Hot Encoding*.

```
# Extraindo as colunas 'SR-ARE' e 'SR-MMP' e a coluna 'smiles'
smiles_1 = df_modificado[['smiles']]
agentes = df_modificado[['SR-ARE', 'SR-MMP']]

# Remove linhas onde qualquer coluna contém -1 do df_modificado
df_modificado = df_modificado[(df_modificado[['SR-ARE', 'SR-MMP']] != -1).all(axis=1)]

# Extraia a coluna 'SMILES' novamente após filtrar
smiles_1 = df_modificado[['smiles']]
```

```
agentes = df_modificado[['SR-ARE', 'SR-MMP']]

# Juntando a smiles_1 com agentes
df_intermediario = smiles_1.join(agentes)

# Converte colunas 'SR-ARE' e 'SR-MMP' em tipos categóricos
df_intermediario['SR-ARE'] = df_intermediario['SR-ARE'].astype('category')
df_intermediario['SR-MMP'] = df_intermediario['SR-MMP'].astype('category')

# Reseta o index do df_intermediario
df_intermediario = df_intermediario.reset_index(drop=True)
```

Por último, a saída em *Multi-Hot Encoding* é recodificada utilizando álgebra booleana, sendo então convertida em números inteiros logo em seguida, resultando em uma saída *One-Hot Encoding*, o que facilita a implementação do modelo e o ajuste de hiperparâmetros.

```
# Gerar todas as combinações possíveis entre as classes A e B
agentes['SR-ARE_SR-MMP(0,0)'] = (agentes['SR-ARE'] == 0) & (agentes['SR-MMP'] == 0)
agentes['SR-ARE_SR-MMP(1,0)'] = (agentes['SR-ARE'] == 1) & (agentes['SR-MMP'] == 0)
agentes['SR-ARE_SR-MMP(0,1)'] = (agentes['SR-ARE'] == 0) & (agentes['SR-MMP'] == 1)
agentes['SR-ARE_SR-MMP(1,1)'] = (agentes['SR-ARE'] == 1) & (agentes['SR-MMP'] == 1)

# Converter os valores booleanos para inteiros (0 ou 1)
agentes = agentes.astype(int)
agentes_1 = agentes.iloc[:, -4:]

df_intermediario_2 = smiles_1.join(agentes_1)
```

No final da etapa de limpeza de dados, tem-se então um *dataframe* com 5 colunas, como pode ser observado na Figura 8. Na ordem: a codificação em SMILES, caso não tenha havido reação em nenhum dos alvos biológicos, caso tenha havido reação em somente um agente 'SR-ARE' ou 'SR-MMP', e caso tenha havido reação em ambos.

	smiles	SR-ARE_SR-MMP(0,0)	SR-ARE_SR-MMP(1,0)	SR-ARE_SR-MMP(0,1)	SR-ARE_SR-MMP(1,1)
11	O=c1[nH]c(=O)n([C@H]2C[C@H](O)[C@@H](CO)O2)cc1I	1	0	0	0
14	Cl/C=C\C[N+]12CN3CN(CN(C3)C1)C2	0	1	0	0
15	O=C([O-])Cc1cccc2ccccc12	1	0	0	0
21	O=C(O)[C@H](O)c1ccccc1	1	0	0	0
32	Nc1nc2ccccc2[nH]1	1	0	0	0
...	...	...	...	...	...
7809	CCC1CO1	1	0	0	0
7812	O=S1OCC2C(CO1)C1(Cl)C(Cl)=C(Cl)C2(Cl)C1(Cl)Cl	0	0	0	1
7817	CCCn1cnc2c1c(=O)n(CCCCC(C)=O)c(=O)n2C	1	0	0	0
7819	Cc1ccc(=O)n(-c2ccccc2)c1	1	0	0	0
7828	C[C@]12CC[C@H]3[C@@H](CCC4=CC(=O)CC[C@@]43C)[C...	0	1	0	0

1790 rows × 5 columns

Figura 8 – Dataframe final após limpeza de dados, mostrando as colunas SMILES e de reação.

3.4 Preparação das entradas

Com o *dataframe* limpo, pode-se então começar os preparativos para a alimentação dele no modelo, isto é, realizar as devidas modificações e transformações que tornem a entrada compreensível para os modelos de aprendizagem profunda.

A cadeia SMILES, apesar de não parecer em um momento inicial, possui todas as características de um Processamento de Linguagem Natural (NLP, do inglês Natural Language Processing): sentenças são organizadas em palavras ou tokens distintos, essa sequência é uma sequência distinta entre si que cria algum sentido lógico e cada token tem sentido apenas quando visto em um contexto (o que precede e sucede o mesmo). Portanto, o tratamento da cadeia será o mesmo que qualquer NLP padrão (FERREIRA; ANDRICOPULO, 2018).

O primeiro passo, então, é dividir cada grande unidade em unidades menores, chamadas de tokens, o que permite criar uma diferenciação entre as mesmas e, no futuro, permitir a realização de diferentes cálculos com cada uma. Geralmente, essa divisão é feita de uma maneira mecânica: se linguagem natural escrita, separar de palavra em palavra; se outra coisa, separar em unidade de caracteres (MIELKE et al., 2021).

Baseado no trabalho de Chakravarti e Alla (2019), é criada uma função de tokenização específica para trabalhar com dados SMILES. Os principais pontos da função são não separar elementos que possuem sentido apenas juntos, como por exemplo:

- Elementos químicos que possuem dois caracteres, como bromo (Br) ou cloro (Cl);
- Conteúdos em colchetes são mantidos como uma unidade, já que representam variação de carga elétrica;
- Números que precedem o caractere "%", que indica o número de anéis de carbono;

- Conteúdos que precedem os caracteres "@@", "-", ".", "+", "=" ou "#", que também indicam uma característica específica da cadeia.

Portanto, a função de tokenização será esta:

```
def custom_tokenize_series(input_series):
    def tokenize_string(input_string):
        tokens = []
        i = 0
        while i < len(input_string):
            # Para casos de 4 caracters 1 token
            if input_string[i:i+4] in ['(=O)', '[nH]']:
                tokens.append(input_string[i:i+4])
                i += 4
            # Para casos de 2 caracters 1 token (@@ é considerado uma
               unidade de anotação, apesar de possuis 2 caracteres)
            elif input_string[i:i+2] in ['He', 'Li', 'Be', 'Ne', 'Na', 'Mg'
                , 'Al', 'Si', 'Cl', 'Ar', 'Ca', 'Sc','Ti', 'Cr', 'Mn', 'Fe',
                 'Co', 'Ni', 'Cu', 'Zn', 'Ga', 'Ge', 'As', 'Se', 'Br', 'Kr',
                 'Rb', 'Sr', 'Zr', 'Nb', 'Mo', 'Tc', 'Ru', 'Rh', 'Pd', 'Ag',
                 'Cd', 'In', 'Sn', 'Sb',  'Te', 'Xe','Cs', 'Ba', 'La', 'Hf',
                 'Ta', 'Re', 'Os', 'Ir', 'Pt', 'Au', 'Hg', 'Tl', 'Pb', 'Bi',
                 'Po','At', 'Rn', 'Fr', 'Ra', 'Ac', 'Rf', 'Db', 'Sg', 'Bh',
                'Hs', 'Mt', 'Ds', 'Rg', 'Cn', 'Nh', 'Fl', 'Mc', 'Lv', 'Ts',
                'Og', 'Cl', 'Br', '@@', '-', '.', '+', '=', '#']:

                tokens.append(input_string[i:i+2])
                i += 2
            # Para casos de colchetes
            elif input_string[i] == '[':
                closing_bracket_index = input_string.find(']', i)
                if closing_bracket_index != -1:
                    tokens.append(input_string[i:closing_bracket_index +
                        1])
                    i = closing_bracket_index + 1
                else:
                    tokens.append(input_string[i])
                    i += 1
            # Para casos de %
            elif input_string[i] == '%':
                match = re.match(r'%\d+', input_string[i:])
                if match:
                    tokens.append(match.group())
                    i += len(match.group())
                else:
                    tokens.append(input_string[i])
                    i += 1
            # Para todo resto
```

```
            else:
                tokens.append(input_string[i])
                i += 1
        return tokens

    tokenized_series = input_series.apply(tokenize_string)
    return tokenized_series
```

Por fim, tem-se então uma sequência de tokens como os dados de entrada.Como o modelo é, em todos os sentidos, um modelo estatístico matemático, é necessário transformar cada token em um número, sendo então o momento de realizar a transformação *text-to-sequence*, que consistem em transformar cada token em um número inteiro com base em um dicionário.

Para preservar o máximo de elementos possíveis e evitar que algum token que apareça pouco, mas possua grande importância, seja "ocultado", o dicionário terá um tamanho ilimitado, ou seja, cada token possuirá um valor inteiro próprio. Para facilitar a alimentação ao modelo, as entradas, após a transformação em inteiros, receberão *post-padding* (adição de zeros até que se alcance o tamanho desejado).

3.5 Modelo

Com as entradas devidamente tratadas e preparadas para o treinamento, agora será realizada a definição mais concreta do modelo, trabalhando diretamente nos hiperparâmetros de cada camada e algumas ferramentas mais específicas.

A primeira camada, por se tratar de um modelo que utilizará linguagem natural (ou uma aproximação a ela), será a camada de *Embedding*. Como descrito por Li e Yang (2018), essa camada é naturalmente presente quando as entradas são do tipo NLP, criando uma série de vetores que armazenam a relação das palavras inseridas, como pode ser observado na Figura 9. As vantagens de utilizar essa camada incluem o uso reduzido de espaço computacional (quando comparado com outros métodos, como OneHotEncoding) e o fato de que seus pesos serão treinados juntamente com o modelo, simplificando o processo.

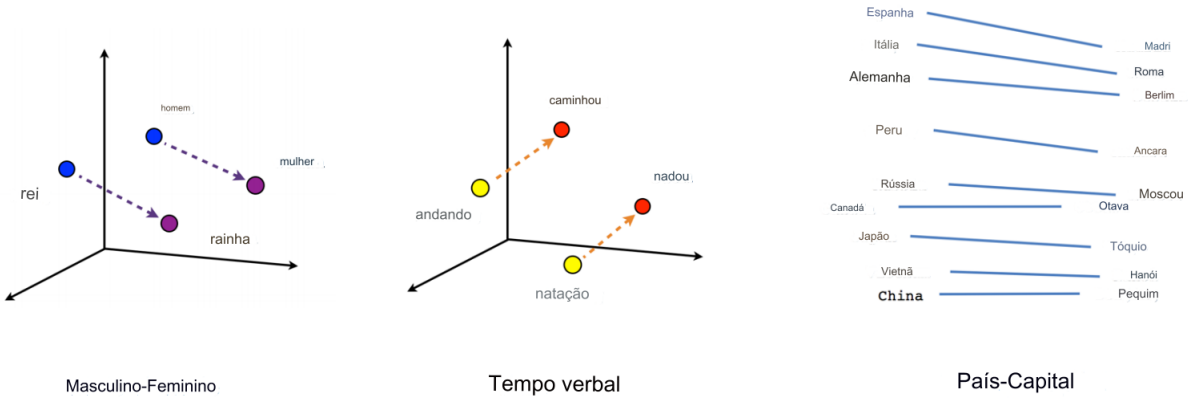

Figura 9 – Exemplificação da criação dos vetores da camada Embedding. Fonte: Ruizendaal (2017).

A dimensão de entrada dessa camada será definida pelo número de palavras distintas no dicionário de palavras criado ao mapear os tokens distintos, mais o token para palavras fora de contexto. A dimensão de saída será o tamanho máximo que uma entrada pode possuir, ou seja, o número de caracteres em uma sequência SMILES após a tokenização.

A próxima camada será uma camada GRU bidirecional: a parte bidirecional, como explicado por Schuster e Paliwal (1997), proporciona uma interpretabilidade bidirecional. Ou seja, em vez de utilizar apenas o elemento anterior para predizer o futuro, é possível também levar em consideração o elemento seguinte para realizar a predição, utilizando ambos os contextos (da primeira à última e da última à primeira) para prever o próximo estado do ponto alvo. Sua configuração inicial de neurônios será variada no primeiro experimento, portanto será denominado como n como referencia.

Em seguida, haverá uma camada GRU simples, funcionando como intermediária entre a bidirecional e a camada de saída. Essa camada sempre possuirá metade dos neurônios da anterior, portando seu valor será $n/2$, uma prática comum que traz certos benefícios, como: redução da dimensão, permitindo focar em características mais relevantes; redução do *overfitting* geral, graças à redução de camadas ocultas; eficiência computacional, pelo motivo anterior; e uma melhor representação hierárquica (GOODFELLOW; BENGIO; COURVILLE, 2016).

Por fim, tem-se a camada de saída: uma camada de neurônios simples (também conhecida como *Dense*), possuindo quatro neurônios, que é o número de possíveis etiquetas, e a função de ativação sigmoide, que permitirá calcular a probabilidade de cada etiqueta ser a correta com base na entrada, selecionando então a mais provável.

É válido ressaltar que, nesse primeiro momento, o modelo será simplificado, possuindo apenas um *dropout* de 30%, sem grandes outras modificações, como regularizadores

ou outras camadas intermediárias mais complexas. Modificações serão feitas no próximo capítulo com o propósito de melhorar a predição com base no resultado inicial.

Uma visão simplificada do modelo final pode ser observado na Figura 10: é possível observar desda entrada dos dados, a ordem que cada camada esta disposta até a saída, sendo a predição.

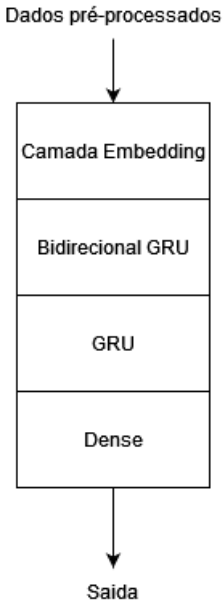

Figura 10 – Descrição visual da disposição das camadas do modelo.

Em seguida, o modelo é compilado utilizando o otimizador Adam por ser o mais usado, com uma ampla margem de diferença dos outros, além de possuir várias vantagens, com a função de perda sendo a *Binary Crossentropy*, que permite calcular valores entre 0 e 1 (portanto, a probabilidade) para problemas de multi-classificação, que é o caso deste trabalho.

```
1  vocab_size = len(tokenizer.word_index) + 1
2  first_layer = 200
3  dropout = 0.3
4
5  model = Sequential()
6  model.add(Embedding(vocab_size, max_sequence_length, mask_zero=True))
7  model.add(Bidirectional(GRU(int(first_layer), dropout=dropout,
       return_sequences=True)))
8  model.add(GRU(int(first_layer/2), dropout=dropout))
9  model.add(Dense(4, activation='sigmoid'))
10
11 model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['
       accuracy'])
```

4 Resultados

4.1 Primeiro experimento

Com tudo devidamente preparado, foi realizado o treinamento do modelo e a validação de seu desempenho levando em consideração a acurácia e a função de perda. Vinte por cento dos dados foram separados previamente para a validação, e o treinamento foi realizado com cinco configurações de neurônios diferentes para a primeira camada GRU: 32, 64, 128, 256, 512. O *batch size* foi de 64 elementos e o treinamento ocorreu por 1000 épocas, mas o programa forçou a terminação caso não houvesse uma melhoria na acurácia em até cinco épocas.

Como pode ser observado na Tabela 1, o resultado inicial consegue informar alguns detalhes: para o treinamento, houve um desempenho aparente melhor para o modelo com mais neurônios, tanto para a acurácia quanto para a função de perda, mas que não se traduziu em um melhor desempenho real na validação, levando à suposição de que houve *overfitting*.

Somado à análise anterior, é possível então notar que os modelos mais simples possuíram um melhor desempenho na validação dos dados, com acurácias mais altas, o que nos permite inferir uma melhor generalização do aprendizado. A Tabela 1 mostra as métricas de desempenho para diferentes números de neurônios. Para concisão, "Train" se refere ao treinamento, "Val" à validação e "Perda" à função de perda, essa nomenclatura se repetirá para todas as demais tabelas.

Neurônios	Perda Train	Perda Val	Precisão Train	Precisão Val
32	0.3874	0.4251	0.6571	**0.6201**
64	0.3846	0.4278	0.6557	**0.6201**
128	0.3834	0.4242	0.6571	0.6117
256	0.3904	0.4294	0.6501	0.6089
512	**0.3739**	0.4289	**0.6634**	0.6117

Tabela 1 – Métricas de desempenho para diferentes números de neurônios.

Outra maneira de visualizar os dados é com a Figura 11, que permite uma melhor comparação visual entre neurônios x acurácia e neurônios x função de perda, tanto para a amostra de treino (em azul) quanto de validação (em vermelho).

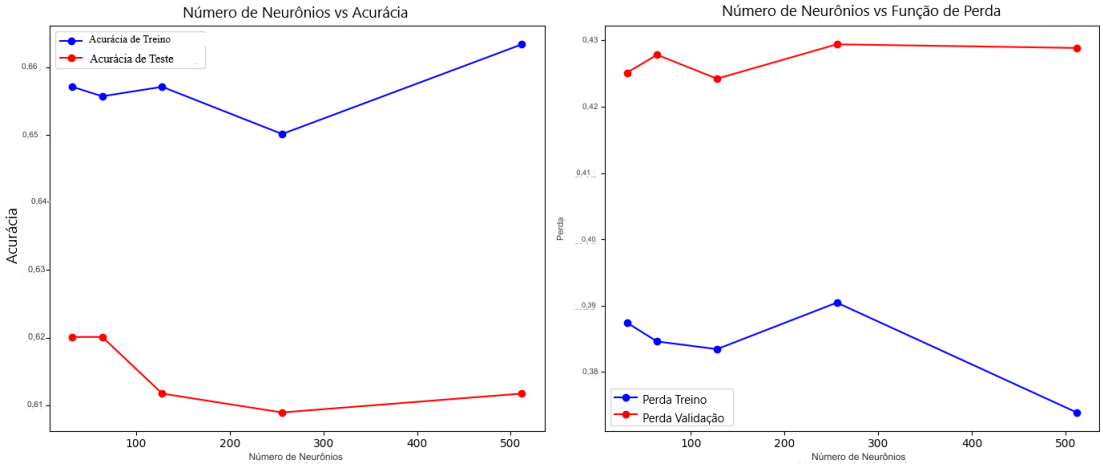

Figura 11 – Gráfico mostrando a acurácia e função de perda final em relação ao número de neurônios. Fonte: Autor.

É valido ressaltar que o treinamento dos modelos foram interrompidos entre 15 a 25 épocas por falta de melhora na acurácia da validação durante o treinamento.

Analisando os resultados do primeiro experimento, chega-se à conclusão de que existem ferramentas que possam auxiliar no treinamento, almejando alcançar um melhor desempenho.

Primeiramente, tem-se a alteração do *batch size*, o que permitirá mais atualizações dos pesos do modelo, além de introduzir mais ruído, o que auxilia em evitar mínimos locais. Como o modelo não sofre de *overfitting*, ele tende a generalizar bem, mas talvez não esteja aprendendo todas as características que a base de dados permite, o que nos leva a essa modificação.

Outra possibilidade seria alterar a taxa de aprendizado. Uma taxa menor é capaz de dar saltos menores e evitar que o modelo fique preso nos em valores repetitivos. Pode-se notar que o número de épocas utilizadas no treinamento é muito pequeno, levando em consideração a complexidade do problema. Portanto, reduzir a *learning rate* poderá trazer resultados mais precisos, mesmo que a custo computacional.

A alteração do *batch size* é facilmente realizada modificando o número na função *model.fit*. Para a alteração da *learning rate* (taxa de aprendizado), será inicializada uma *learning rate* no otimizador e, posteriormente, adicionada uma função de *callback* (função de retorno) que permitirá a redução gradual de seu valor, sempre que a variável de análise parar de sofrer melhorias.

```
initial_learning_rate = 0.001
optimizer = Adam(learning_rate=initial_learning_rate)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5,
    min_lr=0.0005)

model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['
    accuracy'])

history = model.fit(x_train, y_train, verbose=2, epochs=epochs,
    validation_data=(x_val, y_val), batch_size=batch_size, callbacks=[
    checkpoint, early_stopping, reduce_lr])
```

4.2 Segundo experimento

Foi então realizado novamente o treinamento do modelo, agora utilizando os *batch sizes* que obtiveram mais sucesso anteriormente (32 e 64), variando o coeficiente de aprendizado inicial ao longo do treino, e testando o desempenho de um *batch size* menor, com tamanho 16.

Batch Size 32, Taxa de Aprendizado Inicial 0.001				
Neurônios	Perda Train	Perda Val	Precisão Train	Precisão Val
32	0.3653	0.4200	0.6690	**0.6201**
64	0.3524	0.4255	0.6837	0.6117
Batch Size 16, Taxa de Aprendizado Inicial 0.001				
Neurônios	Perda Train	Perda Val	Precisão Train	Precisão Val
32	0.3295	0.4283	0.7297	0.6173
64	0.2807	0.4262	0.7758	**0.6201**
Batch Size 32, Taxa de Aprendizado Inicial 0.0001				
Neurônios	Perda Train	Perda Val	Precisão Train	Precisão Val
32	0.4532	0.4541	0.5922	0.5922
64	0.4443	0.4483	0.5929	0.5922
Batch Size 64, Taxa de Aprendizado Inicial 0.0001				
Neurônios	Perda Train	Perda Val	Precisão Train	Precisão Val
32	0.4626	0.4632	0.5929	0.5922
64	0.4510	0.4526	0.5929	0.5922

Tabela 2 – Métricas de desempenho para diferentes *batch size* e taxas de aprendizado inicial, com o ajuste fino

Como pode ser observado na Tabela 2, o resultado final não apresentou grandes

diferenças em relação aos obtidos anteriormente, embora ainda seja possível extrair a informação de que há um melhor desempenho do modelo quando inicializado com uma taxa de aprendizado menor.

4.3 Terceiro experimento

Contexto e motivação

Como mencionado no subcapítulo 3.2, os dados iniciais apresentam uma notável desproporção entre as classes. Apesar das escolhas dos dois principais reagentes e da junção das saídas para *one-hot encoded* terem mitigado parcialmente o problema, ainda foi possível perceber uma diferença significativa nos dados. Portanto, foi realizado um teste *Post Hoc*, aplicando um ajuste de pesos ponderados ao treinamento e à validação.

É válido mencionar que a estratégia para tentar mitigar o desbalanceamento foi o ajuste ponderado por uma série de fatores: Utilizar subamostragem (*under-sampling*) da classe dominante seria problemático por conta do baixo número total de elementos que nossa base de dados possui, e como mencionado anteriormente, o modelo necessita de uma grande quantidade de elementos, tornando cada dado crucial. Sobreamostragem (*oversampling*) seria ainda mais problemática, pois não só poderia levar a um problema de *overfitting*, como também à criação de dados equivocados e irreais, devido à natureza das sequências químicas.

Portanto, a utilização de um ajuste ponderado torna-se a melhor solução para o problema. Como mencionado por Barandela et al. (2003), a aplicação de um peso ponderado diretamente aos rótulos apresenta um ótimo desempenho quando utilizada em problemas de classificação com classes desbalanceadas, que é o caso presente.

Implementação

Primeiramente, é necessário obter a proporção entre as possíveis classes da saída que será usada para o treinamento e teste, rodando o seguinte código:

```
1 unique_combinations, counts = np.unique(y_train, axis=0, return_counts=True
      )
2
3 print("Contagens para cada combinação de rótulos em y_train:")
4 for combination, count in zip(unique_combinations, counts):
5     print(f"{combination}: {count}")
```

Com os números de aparecimento de cada classe obtidos, basta fazer uma ponderação de frequência inversa, dando prioridade aos pesos dos elementos que menos aparecem e repassar a informação para a função *.fit*.

```
class_weight = {0: 1.0,        #  [0 0 0 1]
                1: 848/217,    #  [0 0 0 1]
                2: 848/153,    #  [0 0 1 0]
                3: 848/214}    #  [0 1 0 0]


model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size,
    validation_data=(x_val, y_val), class_weight=class_weight)
```

Resultados Ponderados

Com os resultados em mãos, é possível notar uma diferença significativa entre os resultados dos treinos e validações com os dados ponderados. Para o modelo com 32 neurônios na primeira camada, houve uma queda de um 13% na acurácia, um valor extremamente significativo, sendo ainda mais extrema para o modelo de 64 neurônios, que apresentou uma queda aproximada de 20% quando comparado à validação com a amostra não ponderada.

Os resultados completos podem ser observados na Tabela 3 abaixo:

Batch Size 32, Taxa de Aprendizado Inicial 0.001				
Neurônios	Perda Train	Perda Val	Precisão Train	Precisão Val
32	0.9925	0.4894	0.6117	**0.5391**
64	0.8040	0.4814	0.6746	**0.5028**

Tabela 3 – Métricas de desempenho para determinado *batch size* e taxa de aprendizado inicial, com ajuste ponderado

Os valores obtidos ao decorrer do treinamento podem ser observados na Figura 12.

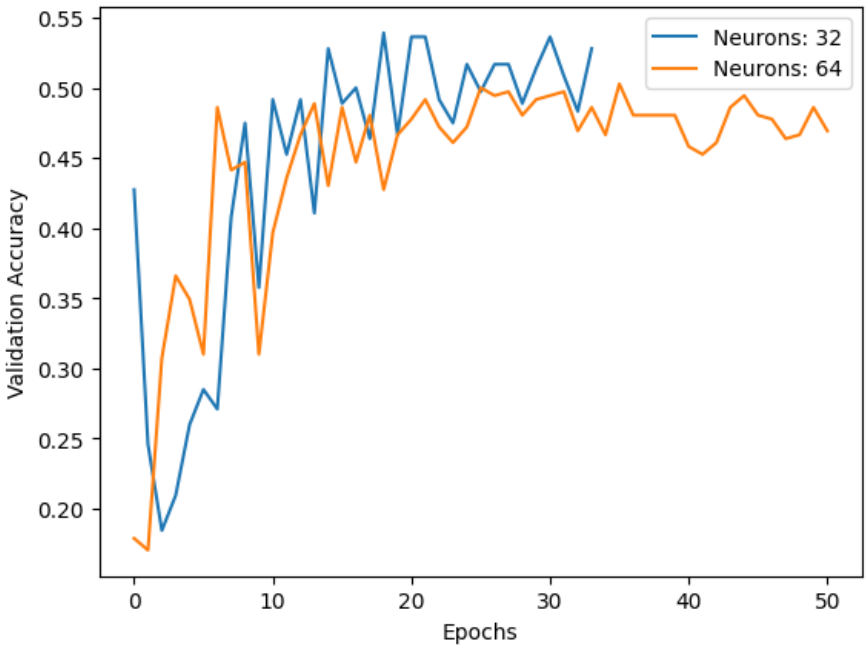

Figura 12 – Gráfico mostrando a acurácia e função de perda final em relação ao número de neurônios quando aplicado ponderação de frequência inversa. Fonte: Autor.

Limitações do teste

Apesar de criar um resultado mais "real" quando analisado os dados relacionados à acurácia, esse método adiciona mais de complexidade ao modelo, podendo até gerar *overfitting* em classes pouco representadas graças ao baixo número de amostras e a tendencia colocar um peso maior a elas.

4.4 Discussão do resultados

Agora, com os resultados finais em mãos, pode-se realizar a análise final do projeto e avaliar o desempenho geral dos modelos.

O melhor resultado obtido apresentou uma acurácia de 62% para os quatro rótulos possíveis quando não se considerou a ponderação de frequência inversa. Mais de uma configuração de modelo alcançou esse resultado ótimo, sendo a configuração inicial de 32 neurônios a versão mais simplificada e menos custosa para o *hardware*.

Ao analisar o modelo em que se considerou a distribuição de cada rótulo e aplicou-se a ponderação de frequência inversa, nota-se uma acurácia inferior. Isso demonstra que o modelo anterior desempenhava bem para o rótulo mais frequente, mas tendia a apresentar um desempenho inferior nos resultados menos frequentes. Os resultados sugerem que, embora o modelo original fosse adequado para previsões gerais, o ajuste ponderado aumentou a eficácia do modelo em casos específicos.

Quando comparado a outros projetos similares, como o de Chakravarti e Alla (2019), o modelo ficou aquém, mas com algumas ressalvas: o modelo de Chakravarti possui uma arquitetura bem mais complexa, com uma base de dados de até 30.000 elementos, e seus resultados variaram de 98% (agente Ames Mutagenicity) até 70% (agente inibidor do P. falciparum Dd2).

Já o modelo final apresentado aqui contou com uma versão simplificada, almejando diminuir gastos computacionais e possuía uma base de dados não superior a 6.500 elementos, unificando 4 rótulos totais (sendo dois agentes biológicos), aumentando consideravelmente a dificuldade para o aprendizado do modelo.

Um dos fatores que poderiam futuramente levar a um melhor resultado seria a tentativa de retreinamento com uma base de dados maior ou a utilização de algumas técnicas um pouco mais custosas para tentar elevar a acurácia, como, por exemplo, algumas utilizadas no trabalho de Silva (2024) para problemas de multirrótulo.

Outra possibilidade para lidar com o problema de desbalanceamento de dados seria adotar uma metodologia diferente: em vez de utilizar apenas pesos ponderados, a validação cruzada com estratificação pode ser uma alternativa promissora. Como mencionado por Kohavi (1995), essa técnica oferece uma boa relação entre viés e variância, fatores que, apesar de não terem sido métricas analisadas diretamente no trabalho, impactam indiretamente o desempenho do modelo e a acurácia. A implementação dessa abordagem poderia complementar as estratégias testadas, aumentando a robustez do modelo e potencialmente melhorando a precisão para os rótulos menos frequentes.

5 Conclusão

Neste trabalho, foi realizado uma análise da rede neural recorrente GRU quando aplicada na predição de multirrótulo de alvos biológicos, utilizando o modelo QSAR sem descritores, por meio da representação molecular SMILES. Com isso, pode-se perceber a eficiência de modelos mais simples na análise da relação entre estrutura química e atividade biológica.

Para atingir esse objetivo, foram realizados vários experimentos, testando diferentes arquiteturas do modelo GRU, com foco na otimização de hiperparâmetros e no desbalanceamento de classes, utilizando pesos ponderados.

O modelo que obteve a melhor desempenho atingiu 62% de acurácia, que apesar de promissor, aponta a possibilidade de melhores otimizações. Outro ponto notado nos experimentos é que o desbalanceamento de classes afetou significativamente o modelo. O ajuste ponderado ajudou a melhorar as previsões para essas classes, mas levou a uma diminuição na precisão geral.

Apesar dos obstáculos encontrados, o modelo GRU demonstra capacidade para servir como uma ferramenta auxiliar na análise QSAR, especialmente se aliado a uma base de dados mais numerosa e robusta. Embora não seja suficiente como uma ferramenta única, ele proporciona uma eficiência computacional que pode ser ainda mais aprimorada em trabalhos futuros.

Referências

ABIODUN, OI et al. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, Elsevier, v. 4, n. 11, e00938, 2018. DOI: 10.1016/j.heliyon.2018.e00938. Citado 1 vez na página 15.

ATAS GUVENILIR, H.; DOĞAN, T. How to approach machine learning-based prediction of drug/compound–target interactions. *Journal of Cheminformatics*, v. 15, n. 1, p. 16, 2023. DOI: 10.1186/s13321-023-00689-w. Citado 1 vez na página 11.

BARANDELA, R. et al. Strategies for learning in class imbalance problems. *Pattern Recognition*, v. 36, n. 3, p. 849–851, 2003. DOI: 10.1016/s0031-3203(02)00257-1. Citado 1 vez na página 36.

CHAKRAVARTI, S. K.; ALLA, S. R. M. Descriptor Free QSAR Modeling Using Deep Learning With Long Short-Term Memory Neural Networks. *Frontiers in Artificial Intelligence*, v. 2, 2019. DOI: 10.3389/frai.2019.00017. Citado 3 vezes nas páginas 21, 28, 39.

CHO, Kyunghyun et al. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. arXiv: 1406.1078 [cs.CL]. Disponível em: https://arxiv.org/abs/1406.1078. Citado 2 vezes nas páginas 11, 18.

DARA, S.; DHAMERCHERLA, S.; JADAV, S. S. et al. Machine Learning in Drug Discovery: A Review. *Artificial Intelligence Review*, v. 55, p. 1947–1999, 2022. DOI: 10.1007/s10462-021-10058-4. Citado 1 vez na página 10.

DEARDEN, John C. The History and Development of Quantitative Structure-Activity Relationships (QSARs). In: ONCOLOGY: Breakthroughs in Research and Practice. IGI Global, 2017. P. 51. DOI: 10.4018/978-1-5225-0549-5.ch003. Citado 2 vezes na página 20.

FENG, Weijiang et al. Audio visual speech recognition with multimodal recurrent neural networks. In: p. 681–688. DOI: 10.1109/IJCNN.2017.7965918. Citado 0 vez na página 15.

FERREIRA, Leonardo L. G.; ANDRICOPULO, Adriano D. Editorial: Chemoinformatics Approaches to Structure- and Ligand-Based Drug Design. *Frontiers in Pharmacology*, v. 9, p. 1416, dez. 2018. DOI: 10.3389/fphar.2018.01416. Citado 1 vez na página 28.

GILMER, Justin et al. *Neural Message Passing for Quantum Chemistry*. 2017. arXiv: 1704.01212 [cs.LG]. Disponível em: https://arxiv.org/abs/1704.01212. Citado 1 vez na página 12.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning*. MIT Press, 2016. Citado 1 vez na página 31.

GOODWIN, William M. Structural formulas and explanation in organic chemistry. *Foundations of Chemistry*, v. 10, n. 2, p. 117–127, 2007. DOI: 10.1007/s10698-007-9033-2. Citado 1 vez na página 19.

ISARANKURA-NA-AYUDHYA, Chartchalerm et al. A practical overview of quantitative structure-activity relationship. en. *Excli - Experimental and Clinical Sciences*, v. 8, 2009. Citado 1 vez na página 10.

JARADA, T. N.; ROKNE, J. G.; ALHAJJ, R. A review of computational drug repositioning: strategies, approaches, opportunities, challenges, and directions. *Journal of Cheminformatics*, v. 12, n. 1, p. 46, 2020. DOI: 10.1186/s13321-020-00450-7. Citado 1 vez na página 10.

KOHAVI, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. *Morgan Kaufman Publishing*, 1995. Citado 1 vez na página 39.

KUMAR, Sandeep; KUMAR, Naresh; DEV, Archana et al. Movie genre classification using binary relevance, label powerset, and machine learning classifiers. *Multimedia Tools and Applications*, Springer, v. 82, p. 945–968, 2023. DOI: 10.1007/s11042-022-13211-5. Citado 1 vez na página 19.

LI, Y.; YANG, T. Word Embedding for Understanding Natural Language: A Survey. In: SRINIVASAN, S. (Ed.). *Guide to Big Data Applications*. Cham: Springer, 2018. v. 26. (Studies in Big Data). P. 83–104. DOI: 10.1007/978-3-319-53817-4_4. Citado 1 vez na página 30.

MADJAROV, Gjorgji et al. An extensive experimental comparison of methods for multi-label learning. *Pattern Recognition*, Elsevier, v. 45, n. 9, p. 3084–3104, 2012. DOI: 10.1016/j.patcog.2012.03.004. Citado 1 vez na página 19.

MIELKE, Sabrina J. et al. Between words and characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP. *CoRR*, abs/2112.10508, 2021. arXiv: 2112.10508. Disponível em: https://arxiv.org/abs/2112.10508. Citado 1 vez na página 28.

PATEL, H. M.; NOOLVI, M. N.; SHARMA, P. et al. Quantitative structure–activity relationship (QSAR) studies as strategic approach in drug discovery. *Medicinal Chemistry Research*, v. 23, p. 4991–5007, 2014. DOI: 10.1007/s00044-014-1072-3. Citado 0 vez na página 21.

PATEL, Veer; SHAH, Manan. Artificial intelligence and machine learning in drug discovery and development. *Intelligent Medicine*, v. 02, n. 03, p. 134–140, 2022. DOI: 10.1016/j.imed.2021.10.001. eprint: https://mednexus.org/doi/pdf/10.1016/j.imed.2021.10.001. Disponível em: https://mednexus.org/doi/abs/10.1016/j.imed.2021.10.001. Citado 1 vez nas páginas 10, 11.

RAVANELLI, M. et al. Light Gated Recurrent Units for Speech Recognition. *IEEE Transactions on Emerging Topics in Computational Intelligence*, v. 2, n. 2, p. 92–102, 2018. DOI: 10.1109/tetci.2017.2762739. Citado 1 vez na página 17.

REHMER, A.; KROLL, A. On the vanishing and exploding gradient problem in Gated Recurrent Units. *IFAC-PapersOnLine*, Elsevier, v. 53, n. 2, p. 1243–1248, 2020. DOI: 10.1016/j.ifacol.2020.12.1342. Citado 2 vezes na página 16.

RIBEIRO, António H. et al. Beyond exploding and vanishing gradients: analysing RNN training using attractors and smoothness. In: CHIAPPA, Silvia; CALANDRA, Roberto (Ed.). *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. PMLR, 26–28 Aug 2020. v. 108. (Proceedings of Machine Learning Research), p. 2370–2380. Disponível em: https://proceedings.mlr.press/v108/ribeiro20a.html. Citado 1 vez na página 15.

ROY, Kunal; KAR, Supratik; DAS, Rudra Narayan. What is QSAR? Definitions and Formulism. In: A primer on QSAR/QSPR modeling: Fundamental Concepts. New York: Springer-Verlag Inc., 2015. cap. 1.2, p. 2–6. ISBN 978-3-319-17281-1. Citado 1 vez na página 20.

RUIZENDAAL, Rutger. *Deep Learning #4: Why You Need to Start Using Embedding Layers And how there's more to it than word embeddings*. Jul. 2017. *Towards Data Science*. 7 min read. Disponível em: https://towardsdatascience.com/deep-learning-4-embedding-layers-f9a02d55ac12. Citado 0 vez na página 31.

SCHUSTER, M.; PALIWAL, K.K. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, v. 45, n. 11, p. 2673–2681, 1997. DOI: 10.1109/78.650093. Citado 1 vez na página 31.

SHIRI, Farhad Mortezapour et al. *A Comprehensive Overview and Comparative Analysis on Deep Learning Models: CNN, RNN, LSTM, GRU*. 2023. arXiv: 2305.17473 [cs.LG]. Citado 1 vez na página 21.

SHTAR, Guy et al. A simplified similarity-based approach for drug-drug interaction prediction. *PLOS ONE*, Public Library of Science, v. 18, n. 11, p. 1–19, nov. 2023. DOI: 10.1371/journal.pone.0293629. Disponível em: https://doi.org/10.1371/journal.pone.0293629. Citado 1 vez na página 21.

SILVA, Getúlio Rodrigues. *Redução de Dimensionalidade em Problemas de Múltiplos Alvos: uma Comparação entre PCA e Autoencoder*. Ouro Preto, Brazil, 2024. Citado 2 vezes nas páginas 22, 39.

TODESCHINI, Roberto; CONSONNI, Viviana. *Handbook of Molecular Descriptors*. WILEY-VCH Verlag GmbH, 2000. (Methods and Principles in Medicinal Chemistry). ISBN 9783527299133. DOI: 10.1002/9783527613106. Disponível em: https://doi.org/10.1002/9783527613106. Citado 1 vez na página 12.

TSOUMAKAS, Grigorios; KATAKIS, Ioannis. Multi-Label Classification: An Overview. *International Journal of Data Warehousing and Mining*, v. 3, p. 1–13, set. 2009. DOI: 10.4018/jdwm.2007070101. Citado 1 vez na página 18.

VAN VLIJMEN, H.; DESJARLAIS, R. L.; MIRZADEGAN, T. Computational chemistry at Janssen. *Journal of Computer-Aided Molecular Design*, v. 31, n. 3, p. 267–273, mar. 2017. Epub 2016 Dec 19. DOI: 10.1007/s10822-016-9998-9. Citado 2 vezes na página 10.

WEININGER, David. SMILES, a Chemical Language and Information System. *Journal of Chemical Information and Computer Sciences*, American Chemical Society, v. 28, n. 1, p. 31–36, 1988. Published February 1, 1988. DOI: 10.1021/ci00057a005. Citado 1 vez na página 12.

WEININGER, David. SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, ACS Publications, v. 28, n. 1, p. 31–36, 1988. Citado 1 vez na página 20.

WIGH, Daniel S.; GOODMAN, Jonathan M.; LAPKIN, Alexei A. A review of molecular representation in the age of machine learning. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, v. 12, n. 1, e1603, 2022. Citations: 37. Edited by Raghavan Sunoj. Funding: Engineering and Physical Sciences Research Council, Grant/Award Number: EP/S024220/1; UCB. DOI: 10.1002/wcms.1603. Disponível em: https://doi.org/10.1002/wcms.1603. Citado 1 vez na página 19.

YASONIK, Jacob. Multiobjective de novo drug design with recurrent neural networks and nondominated sorting. *Journal of Cheminformatics*, v. 12, fev. 2020. DOI: 10.1186/s13321-020-00419-6. Citado 0 vez na página 20.