UNIVERSIDADE FEDERAL DE SÃO CARLOS
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

CAUÊ BONFIM TREVISAN

ANÁLISE COMPARATIVA DE MÉTODOS DE APRENDIZADO PROFUNDO PARA PREDIÇÃO DE PREÇOS DE CONTRATOS FUTUROS DE MILHO

São Carlos, SP
21 de julho de 2025

CAUÊ BONFIM TREVISAN

ANÁLISE COMPARATIVA DE MÉTODOS DE APRENDIZADO PROFUNDO PARA PREDIÇÃO DE PREÇOS DE CONTRATOS FUTUROS DE MILHO

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Computação da Universidade Federal de São Carlos como parte dos requisitos para obtenção do grau de Engenheiro da Computação.

Orientador: Prof. Dr. Alexandre Luís Magalhães Levada

São Carlos, SP

21 de julho de 2025

Dedico este trabalho para minha família, meus amigos da República Paratudo e para o grupo de extensão Liga de Mercado Financeiro de São Carlos.

RESUMO

O aprendizado profundo é uma abordagem moderna de aprendizado de máquina, amplamente utilizada na resolução de problemas complexos e em constante evolução. Este trabalho propõe uma análise comparativa de diferentes modelos de aprendizado profundo aplicados à previsão da variação de preços de contratos futuros de milho. A metodologia consiste em testar múltiplas arquiteturas por modelo, selecionar as melhores configurações e, posteriormente, compará-las com um modelo base, construído a partir de uma heurística simples de previsão. A principal contribuição deste estudo é aprofundar o entendimento sobre a aplicabilidade de modelos de aprendizado profundo na predição de séries temporais financeiras, com potencial de expansão para outros domínios. Os resultados obtidos foram satisfatórios, com os modelos convolucional e recorrente superando o modelo base de forma consistente, enquanto o modelo LSTM (*Long-Short Term Memory Neural Network*) apresentou desempenho inferior ao esperado, destacando o impacto da complexidade arquitetural em cenários com bases de dados mais restritas.

Palavras-chave: regressão. aprendizado profundo. redes neurais. contratos futuros. mercado financeiro.

ABSTRACT

Deep learning is a modern machine learning approach widely used to solve complex and evolving problems. This work presents a comparative analysis of different deep learning models applied to the prediction of price variations in corn futures contracts. The methodology involves testing multiple architectures for each model, selecting the best configurations, and subsequently comparing them to a baseline model built on a simple forecasting heuristic. The main contribution of this study is to deepen the understanding of the applicability of deep learning models in financial time series forecasting, with potential extensions to other domains. The results were satisfactory, with convolutional and recurrent models consistently outperforming the baseline, while the LSTM (Long-Short Term Memory Neural Network) model delivered lower-than-expected performance, highlighting the impact of architectural complexity in scenarios with more limited datasets.

Keywords: regression. deep learning. neural networks. future contracts. financial market.

LISTA DE ILUSTRAÇÕES

LISTA DE CÓDIGOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ML	Machine Learning (Aprendizado de Máquina)
AI	Artificial Intelligence (Inteligência Artificial)
DL	Deep Learning (Aprendizado Profundo)
RNN	Recurrent Neural Network (Rede Neural Recorrente)
LSTM	Long Short-Term Memory (Memória de Longo e Curto Prazo)
MSE	Mean Squared Error (Erro Quadrático Médio)
MAE	Mean Absolute Error (Erro Absoluto Médio)
DNN	Deep Neural Network (Rede Neural Profunda)
CNN	Convolutional Neural Network (Rede Neural Convolucional)
CME	Chicago Mercantile Exchange (Bolsa de Mercadorias de Chicago)
CBOT	Chicago Board of Trade (Bolsa de Comércio de Chicago)
BPTT	Backpropagation Through Time (Retropropagação Através do Tempo)
MLP	Multi-Layer Perceptron (Perceptron de Múltiplas Camadas)
GPU	Graphics Processing Unit (Unidade de Processamento Gráfico)
TensorFlow	Biblioteca de código aberto para aprendizado de máquina e redes neurais
Keras	Interface de alto nível para construção e treinamento de redes neurais

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo, serão explicadas as motivações do presente trabalho, junto com seus objetivos gerais e específicos e, por fim, a forma com que este está organizado.

1.1 MOTIVAÇÃO

Técnicas de aprendizado de máquina têm sido usadas exaustivamente para resolução de diversos problemas modernos da humanidade. Esses avanços são recentes, uma vez que, para serem possíveis, dependeram de toda uma infraestrutura de conectividade, hardware e desenvolvimento de software para acontecerem, e assim o mundo tem hoje sido capaz de aproveitar tais recursos.

O aprendizado de máquina surge em 1950, e como subárea de estudo, mais tarde surge o aprendizado profundo. O aprendizado profundo, como explicado por Goodfellow et al. (2016) (GOODFELLOW; BENGIO; COURVILLE, 2016), se refere ao uso de redes neurais artificiais com camadas intermediárias. Assim, os algoritmos e modelos que decorrem desse conceito são muitos, desde um simples agrupamento de neurônios densos (LECUN; BENGIO; HINTON, 2015) até o uso de camadas que fazem convoluções sobre os dados que recebem, como nas CNNs (*Convolutional Neural Network*) (KRIZHEVSKY; SUTSKEVER; HINTON, 2012), e até mesmo redes com camadas que se utilizam de memória temporária de dados anteriores para prever informações atuais, como as RNNs (*Recurrent Neural Network*) (HOCHREITER; SCHMIDHUBER, 1997).

O surgimento de diversas arquiteturas diferentes para as redes ocorreu de acordo com a necessidade. Através de arquitetos que dominavam o entendimento das redes, problemas sem resolução recebiam interpretações e representações cada vez melhores, resultando em arquiteturas mais adequadas para a resolução dos problemas. O recente avanço de chats com linguagem natural, por exemplo, se deu devido a esse processo, através do desenvolvimento de um tipo de estrutura chamada *Transformer* (VASWANI et al., 2017) que, alicerçada por uma rede neural, interpretava textos entendendo dependências de longo prazo com atenção contextualizada.

Dessa maneira, aplicações do aprendizado profundo já se deram em muitas áreas, como análise de imagens (LITJENS et al., 2017), prevenção de fraudes (NGAI et al., 2011), identificação de fenômenos naturais como terremotos e tsunamis (PEROL; GHARBI; DENOLLE, 2018), entre outros.

Entre esses vários modelos diferentes, encontra-se um tipo específico de rede, a LSTM (*Long Short-Term Memory Neural Newtork*) (HOCHREITER; SCHMIDHUBER, 1997), que é uma variação das RNNs com capacidade de se adaptar melhor para predição de séries temporais.

Dessa forma, espera-se com o presente estudo verificar os resultados da aplicação dessas redes na série temporal de preços de fechamento de contratos futuros de milho, commodity extremamente relevante e presente para a humanidade.

1.2 OBJETIVOS

O objetivo geral deste trabalho é investigar a aplicabilidade e precisão de diferentes modelos de aprendizado profundo - redes densas MLP (*Multi-Layer Perceptron*), CNN, RNN e LSTM; quando aplicados à predição de uma série temporal financeira específica: os contratos futuros de milho negociados na bolsa de Chicago.

Essa série foi escolhida por representar um ativo com características particularmente favoráveis à aplicação de modelos de aprendizado profundo. Os contratos futuros de milho, negociados na bolsa de Chicago, apresentam alta liquidez e volume de negociação, o que contribui para a confiabilidade dos dados históricos e a estabilidade estatística das séries temporais.

Além disso, o milho é uma commodity agrícola de importância global, com presença histórica consolidada nos mercados financeiros e cuja demanda se mantém com crescimento estável no longo prazo, por ser um insumo essencial na cadeia alimentar e energética. Isso o torna um ativo menos sujeito a descontinuidades ou mudanças estruturais abruptas — diferentemente de ações de empresas específicas, que podem ser fortemente afetadas por eventos corporativos, decisões políticas ou crises institucionais.

A escolha também se justifica pela menor sensibilidade do milho a fatores macroeconômicos e políticos de curto prazo, quando comparado a ativos como ações ou moedas. Essa característica sugere, a priori, um comportamento mais regular e potencialmente mais previsível, o que favorece a investigação da eficácia de diferentes arquiteturas de redes neurais na tarefa de predição.

Para fins de comparação, será também desenvolvido um modelo base, cuja heurística de predição consiste em assumir que o preço de fechamento do dia atual é igual ao preço do dia anterior. Este modelo simples serve como linha de base para avaliar a eficácia das arquiteturas mais complexas.

A fim de garantir que cada modelo opere sob sua melhor configuração, serão realizados testes exaustivos com diferentes arquiteturas para cada tipo de rede, ajustando hiperparâmetros e estruturas. Assim, será possível comparar os modelos em termos de desempenho e precisão, permitindo uma análise justa e aprofundada sobre a efetividade do aprendizado profundo na tarefa proposta.

Por fim, o estudo apresenta um aspecto desafiador que é tentar conseguir boas predições - que ganhem do modelo base - utilizando apenas os dados mais simples possíveis adquiridos da Bolsa de Valores de Chicago, ou seja, os valores de abertura, fechamento, volume de operações e valores mais alto e mais baixo atingidos ao longo da janela de um

dia.

Espera-se que seja possível conseguir uma precisão boa, mas que não necessariamente supere a heurística utilizada para desenvolvimento do modelo base.

1.3 ORGANIZAÇÃO

O restante deste trabalho está organizado da seguinte forma: o Capítulo 2 apresenta a fundamentação teórica, onde são discutidos inicialmente trabalhos relacionados que exploram o uso de redes neurais na previsão de séries temporais financeiras. Em seguida, é feita uma introdução aos contratos futuros, com foco nos contratos de milho, destacando suas características como liquidez, estabilidade e relevância no contexto de commodities. Na sequência, abordam-se os principais conceitos de aprendizado de máquina e aprendizado profundo, preparando o leitor para o entendimento das redes neurais recorrentes do tipo LSTM, as mais complexas redes em termos de parâmetros implementados no presente trabalho.

O Capítulo 3 detalha a metodologia empregada, desde a coleta e pré-processamento dos dados até a definição e treinamento dos modelos, incluindo tanto o modelo base quanto as redes neurais com diferentes arquiteturas. O Capítulo 4 apresenta os resultados obtidos, acompanhados de análises quantitativas e comparações de desempenho entre os modelos. Por fim, o Capítulo 5 reúne as considerações finais, destacando as conclusões do estudo, suas limitações e sugestões para trabalhos futuros. As referências bibliográficas utilizadas encontram-se listadas ao final do documento.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, serão explicados os trabalhos relacionados que serviram como base e inspiração para desenvolvimento deste estudo, seguindo para explicação conceitos teóricos fundamentais utilizados.

2.1 TRABALHOS RELACIONADOS

Diversos estudos recentes têm demonstrado a eficácia de técnicas de aprendizado profundo na previsão de preços de ativos financeiros e índices de mercado. Modelos densos, convolucionais, recorrentes e arquiteturas híbridas destacam-se por sua capacidade de capturar padrões temporais complexos e não lineares presentes nas séries financeiras.

Dentro desse contexto, Mehtab e Sen conduziram uma série de pesquisas entre 2020 e 2022, avaliando diferentes arquiteturas baseadas em redes neurais convolucionais (CNN) e redes recorrentes (LSTM) aplicadas à previsão de séries temporais financeiras de alta frequência. Seus resultados indicaram que, em termos de erro quadrático médio (RMSE), as CNNs superaram as LSTMs. Além disso, observaram que modelos univariados apresentaram desempenho superior aos multivariados, sugerindo que o aumento na complexidade da entrada nem sempre se traduz em ganhos de acurácia (MEHTAB; SEN, 2020; MEHTAB; SEN, 2021d; MEHTAB; SEN, 2022; MEHTAB; SEN, 2021c; MEHTAB; SEN, 2021a; MEHTAB; SEN, 2021b).

Enquanto esses estudos focam predominantemente em dados estruturados, outra linha de pesquisa tem se dedicado à incorporação de dados não estruturados — especialmente textos de mídias sociais e notícias financeiras — como fonte complementar para previsão de mercados. Um trabalho pioneiro nesse campo é o de Bollen et al., que utilizaram sentimentos públicos extraídos do Twitter, por meio das ferramentas *OpinionFinder* e *GPOMS*, para prever o índice Dow Jones (DJIA). Seus achados revelaram correlações significativas entre variáveis emocionais e os movimentos do mercado (BOLLEN; MAO; ZENG, 2011).

A evolução desse tipo de abordagem pode ser observada em estudos como o de Checkley et al., que analisaram os sentimentos *bullish* e *bearish* presentes em microblogs, demonstrando como essas informações impactam a volatilidade e a direção dos preços no curto prazo (CHECKLEY; HIGóN; ALLES, 2017). Na mesma linha, Chen et al. exploraram o uso de técnicas de *Word Embedding*, mais especificamente o *Word2Vec*, para transformar textos de notícias financeiras em vetores numéricos. Esses vetores foram então processados por redes LSTM para previsão de preços no mercado de Taiwan, evidenciando o potencial do Processamento de Linguagem Natural quando integrado a arquiteturas profundas (CHEN; LIAO; HSIEH, 2019).

Buscando aumentar ainda mais a robustez dos modelos, Hu et al. propuseram arquiteturas híbridas que incorporam mecanismos de atenção hierárquica (*Hierarchical Attention Network* - HAN), capazes de mitigar o ruído e as inconsistências inerentes aos textos financeiros (HU; ZHAO; KHUSHI, 2018). De forma complementar, Jeong et al. desenvolveram um sistema abrangente, que combina detecção de *fake news*, análise de risco de crédito e extração de sinais relevantes para previsão de preços, contribuindo assim para a geração de informações mais precisas e confiáveis no suporte à tomada de decisão (JEONG; LEE; CHO, 2018).

Seguindo nessa mesma direção, Li et al. integraram um módulo de mineração de sentimentos a uma LSTM, complementado por um classificador *Naïve Bayes*, utilizado para filtrar opiniões consideradas irracionais. Aplicado ao índice CSI300, esse modelo apresentou desempenho superior quando comparado a abordagens tradicionais (LI et al., 2017).

Por fim, Schumaker e Chen exploraram o impacto de notícias no comportamento das ações que compõem o S&P 500. Através da extração de entidades e da vetorização textual, seu modelo foi capaz de prever a direção dos preços com uma taxa de acerto de 57,1% em janelas de 20 minutos após a publicação das notícias. O método, baseado em SVM (*Support Vector Machine*), obteve um MSE de 0,04261 e um retorno simulado de 2,06% (SCHUMAKER; CHEN, 2009).

De forma geral, esses trabalhos reforçam o crescente interesse acadêmico no uso de aprendizado profundo para a modelagem de séries financeiras, seja a partir de dados estruturados, como séries temporais, seja por meio da análise de dados não estruturados, como textos. As evidências apontam para o elevado potencial dessas abordagens na captura de padrões complexos, oferecendo ferramentas cada vez mais robustas para a compreensão e previsão dos movimentos de mercado.

2.2 CONTRATOS FUTUROS

Contratos futuros são instrumentos derivativos amplamente utilizados nos mercados financeiros, permitindo que produtores, investidores e empresas se protejam contra oscilações de preços, além de serem utilizados para especulação e formação de preços de mercado. Eles representam um acordo padronizado para comprar ou vender um ativo em uma data futura, por um preço previamente definido, sendo negociados em bolsas como a CME Group, nos Estados Unidos, e a B3, no Brasil.

A padronização desses contratos abrange características como tipo e quantidade do ativo, data de vencimento e, em alguns casos, o local de entrega. Essa estrutura garante maior liquidez, transparência e segurança nas negociações. Um aspecto fundamental dos contratos futuros é o mecanismo de ajuste diário, no qual os ganhos e perdas são calculados diariamente com base na variação do preço do ativo. Esse processo, realizado via contas

de margem, reduz o risco de inadimplência e aumenta a confiança entre os participantes do mercado.

No caso das commodities agrícolas, como milho, soja e café, os contratos futuros são essenciais para proteção contra a volatilidade de preços, que pode ser causada por fatores como clima, oferta e demanda, condições logísticas e aspectos macroeconômicos. Este trabalho concentra-se na previsão dos preços dos contratos futuros de milho negociados na CBOT (Chicago Board of Trade), pertencente ao grupo CME, referência global nesse mercado.

A precificação dos contratos futuros reflete a expectativa dos agentes de mercado sobre o preço futuro do ativo, sendo influenciada por variáveis como oferta, demanda, taxas de juros, custos de armazenagem e risco.

2.3 APRENDIZADO DE MÁQUINA

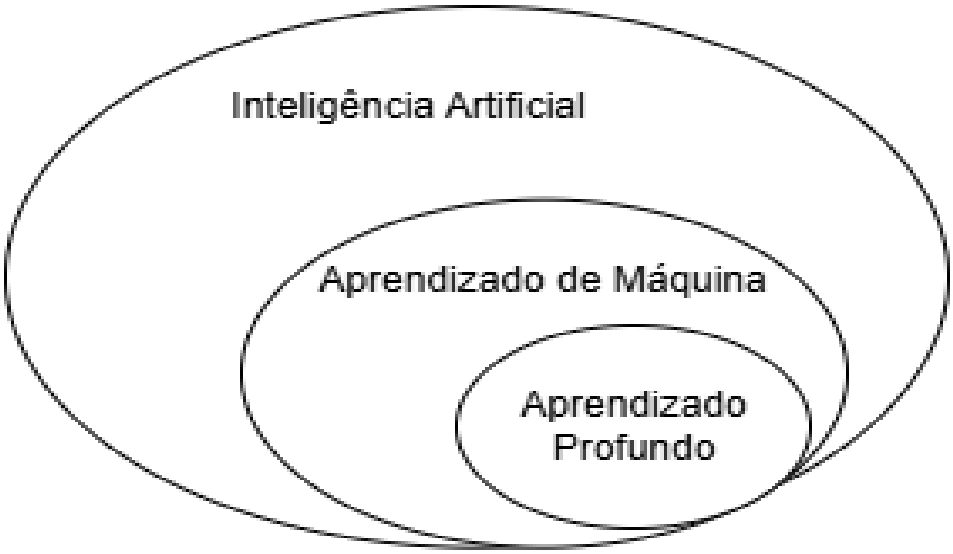
O Aprendizado de Máquina, ou *Machine Learning* (ML), é uma área da Inteligência Artificial (IA) voltada ao desenvolvimento de algoritmos capazes de identificar padrões e aprender a partir de dados, sem a necessidade de instruções explícitas programadas. Trata-se de um campo central para aplicações modernas que envolvem reconhecimento de padrões, análise preditiva e tomada de decisão baseada em dados históricos.

Conceitualmente, o Aprendizado de Máquina está inserido em uma hierarquia de áreas inter-relacionadas. No nível mais abrangente está a Inteligência Artificial, cujo objetivo é simular a inteligência humana por meio de sistemas computacionais, contando muitas vezes com sistemas cujas regras são explicitamente inseridas para o programa se adaptar a "todos"os casos de resolução. Dentro desse campo, encontra-se o Aprendizado de Máquina, responsável por dotar sistemas da capacidade de aprender com dados. Por fim, como subconjunto de ML, está o Aprendizado Profundo, caracterizado pelo uso de redes neurais profundas para modelar relações altamente complexas. A Figura 1 ilustra essa hierarquia conceitual.

A lógica de funcionamento de sistemas de aprendizado de máquina difere do paradigma tradicional de programação. No modelo clássico, um programador define explicitamente as regras e insere os dados, gerando um programa que produz os resultados esperados. Já no Aprendizado de Máquina, o processo é invertido: alimentam-se os algoritmos com dados e os resultados desejados (rótulos ou saídas esperadas), permitindo que o sistema aprenda automaticamente as regras subjacentes. Essa inversão de fluxo é representada na Figura 2.

O principal objetivo de um modelo de ML é a capacidade de generalização: o sistema deve aprender uma função que não apenas acerte os dados de treinamento, mas também seja eficaz quando aplicada a dados novos e desconhecidos. Essa função pode ser representada genericamente pela equação 2.1:

Figura 1 – Inteligência Artificial, Aprendizado de Máquina e Aprendizado Profundo



Elaboração: autor

Figura 2 – Paradigma do Aprendizado de Máquina

Elaboração: autor.

$$f(x) = y \tag{2.1}$$

em que x corresponde aos dados de entrada (por exemplo, sequências temporais de preços passados) e y à saída desejada (como o preço futuro a ser previsto).

Existem três paradigmas principais de aprendizado: supervisionado, não supervisionado e semi-supervisionado. No aprendizado supervisionado, o algoritmo é treinado com pares de entrada e saída conhecidos, permitindo que aprenda uma função mapeadora entre eles. Já no aprendizado não supervisionado, os dados não possuem rótulos e o objetivo

é descobrir estruturas ocultas, como agrupamentos ou padrões recorrentes. O aprendizado semi-supervisionado, por sua vez, combina os dois paradigmas anteriores, utilizando um pequeno conjunto de dados rotulados junto a uma grande quantidade de dados não rotulados, explorando ambas as fontes de informação para treinar o modelo.

Este trabalho está centrado no uso do aprendizado supervisionado, mais especificamente na tarefa de regressão, uma vez que se busca prever um valor numérico contínuo a partir de dados históricos. Nesse contexto, o treinamento de modelos supervisionados envolve um processo iterativo composto por três etapas principais: a propagação direta (em que são geradas predições com base nos pesos atuais do algoritmo), o cálculo do erro entre as predições e os valores reais (por meio de uma função de perda) e, finalmente, a retropropagação do erro para ajuste dos pesos do algoritmo.

O pseudocódigo 1 resume o funcionamento de um modelo supervisionado:

Algoritmo 1: Treinamento de modelo com lotes

Entrada: Dados de entrada divididos em lotes $(x_{\text{lote}}, y_{\text{lote}})$

Saída: Modelo treinado com pesos atualizados

```
modelo ← inicializar_modelo()
para cada época faça
    para cada lote (x_lote, y_lote) faça
        previsoes ← modelo(x_lote)
        erro ← calcular_erro(previsoes, y_lote)
        gradientes ← retropropagar(erro)
        atualizar_pesos(modelo, gradientes)
```

Outras tarefas comuns dentro do Aprendizado de Máquina incluem a classificação, que busca atribuir rótulos a dados (por exemplo, decidir entre "comprar"ou "vender"um ativo), o agrupamento clustering, voltado à identificação de padrões em dados não rotulados.

Dessa forma, o Aprendizado de Máquina fornece a base teórica e prática sobre a qual se desenvolvem os modelos preditivos do presente trabalho, sendo uma etapa fundamental para o entendimento das arquiteturas de aprendizado profundo discutidas na próxima seção.

2.4 APRENDIZADO PROFUNDO

O aprendizado profundo é uma subárea do aprendizado de máquina baseada em redes neurais artificiais compostas por múltiplas camadas de processamento. Ele se destaca por sua capacidade de aprender representações hierárquicas de dados diretamente a partir dos exemplos, sem necessidade de engenharia manual de atributos. Essa abordagem tem se mostrado eficaz em áreas como visão computacional, processamento de linguagem natural e previsão de séries temporais.

As redes neurais são os principais modelos utilizados no aprendizado profundo. Essas redes são organizadas em camadas: uma de entrada, uma ou mais ocultas e uma de saída. Cada camada é composta por uma ou mais unidades de processamento chamadas de neurônios artificiais.

O neurônio de uma rede MLP (*Multi-Layer Perceptron*) realiza três operações principais: calcula um somatório ponderado das entradas, adiciona um viés e aplica uma função de ativação não-linear. Matematicamente, essa operação pode ser expressa pela Equação 2.2:

$$a = \phi\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.2}$$

em que:

a) x_i são as entradas;

b) w_i são os pesos associados a cada entrada;

c) b é o termo de viés;

d) ϕ é a função de ativação;

e) a é a saída do neurônio;

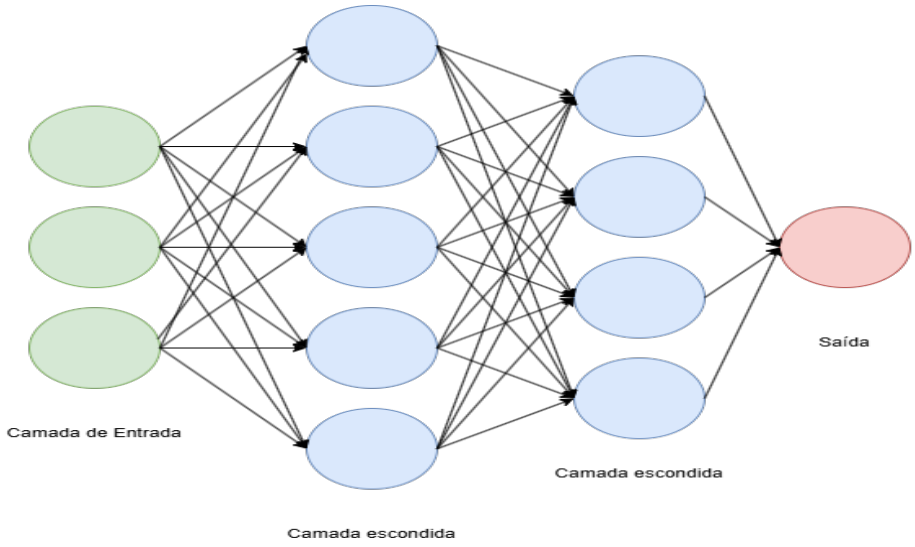
f) n é a quantidade de entradas no neurônio em questão.

De forma conceitual, os pesos (w_i) representam a intensidade ou a importância de cada entrada na formação da saída do neurônio. Cada peso atua como um coeficiente que amplifica ou atenua o impacto de sua respectiva entrada. O viés (b), por sua vez, permite que o neurônio desloque a função de ativação para cima ou para baixo, proporcionando maior flexibilidade no ajuste dos dados e garantindo que a rede consiga representar funções que não passam necessariamente pela origem. Já a função de ativação (ϕ) introduz não linearidade ao modelo, o que é fundamental para que a rede seja capaz de aprender relações complexas nos dados. Sem ela, a rede seria equivalente a um modelo linear, independentemente da quantidade de camadas.

Uma rede MLP é composta por diversas camadas de neurônios organizadas sequencialmente. Cada neurônio de uma camada recebe como entrada as saídas da camada anterior e transmite sua própria saída à camada seguinte. Um esquema pode ser visto na Figura 3.

Durante o treinamento, os pesos e vieses da rede são ajustados por meio do algoritmo de retropropagação do erro, em conjunto com métodos de otimização, como o gradiente descendente e seus variantes. O objetivo desse processo é minimizar a função de perda, uma função matemática que quantifica o quão distante estão as previsões do modelo em relação aos valores reais. Essa função de perda é construída a partir da combinação de todos os neurônios da rede, considerando os valores atuais de seus pesos, vieses e ativações. Dessa forma, ela reflete diretamente a configuração atual dos parâmetros do modelo, sendo

Figura 3 – Esquema exemplo de rede neural com duas camadas intermediárias



Elaboração: autor.

impactada pela interação e influência conjunta de todos os neurônios em um determinado instante do treinamento.

O algoritmo do gradiente descendente cumpre papel de otimizador dessa função de perda, uma vez que indica a direção que os pesos devem ser atualizados a fim de minimizar essa função de perda. Esse ajuste é feito calculando-se o gradiente, ou seja, a derivada da função de perda em relação aos parâmetros. A atualização dos pesos ocorre segundo a Equação 2.3:

$$\theta = \theta - \eta \frac{\partial L}{\partial \theta} \tag{2.3}$$

em que:

a) θ representa um parâmetro da rede (peso ou viés);

b) η é a taxa de aprendizado, que controla o tamanho do passo na atualização;

c) L é a função de perda;

d) $\frac{\partial L}{\partial \theta}$ é o gradiente da função de perda em relação ao parâmetro θ.

Intuitivamente, o gradiente indica a inclinação da função de perda. O algoritmo ajusta os parâmetros na direção oposta ao gradiente, buscando reduzir o erro. Esse processo se repete iterativamente até que a função de perda atinja um valor satisfatoriamente baixo, indicando que a rede aprendeu os padrões presentes nos dados.

Embora as redes MLP sejam capazes de capturar padrões não lineares em dados tabulares, elas apresentam limitações quando aplicadas a dados que possuem estrutura espacial ou sequencial, como imagens e séries temporais. Para superar essas limitações, surgiram arquiteturas especializadas dentro do campo do aprendizado profundo.

Dentre essas arquiteturas, destacam-se:

- **Redes Neurais Convolucionais (CNN)**: projetadas para explorar padrões espaciais ou temporais locais, inicialmente desenvolvidas para visão computacional, mas também aplicáveis na análise de séries temporais.
- **Redes Neurais Recorrentes (RNN)**: desenvolvidas para lidar com dados sequenciais, capazes de manter uma memória de estados anteriores, sendo adequadas para modelagem de séries temporais, texto e áudio.
- **Long Short-Term Memory (LSTM)**: uma variante das RNNs, criada para mitigar problemas como o gradiente desvanecido, permitindo capturar dependências de longo prazo em sequências.

No contexto deste trabalho, essas arquiteturas são exploradas com o objetivo de modelar e prever séries temporais financeiras. As redes CNN são utilizadas pela sua capacidade de extrair padrões locais em janelas temporais, enquanto as redes LSTM são aplicadas devido à sua aptidão para capturar dependências temporais de longo prazo.

2.5 REDES NEURAIS CONVOLUCIONAIS (CNN)

As Redes Neurais Convolucionais (*Convolutional Neural Networks* — CNN) são uma classe de modelos de aprendizado profundo projetados para processar dados que possuem estrutura local, como imagens, sinais e séries temporais. Inicialmente desenvolvidas para tarefas de visão computacional, essas redes mostraram-se altamente eficazes também na modelagem de dados sequenciais, devido à sua capacidade de extrair padrões locais e recorrentes.

Ao contrário das redes MLP, que conectam cada neurônio de uma camada a todos os neurônios da camada seguinte, as CNNs utilizam o conceito de camadas convolucionais, nas quais pequenos filtros (*kernels*) percorrem a entrada, realizando operações de convolução que permitem detectar padrões locais, como bordas, formas ou sequências recorrentes. Cada filtro atua como um detector de padrões, extraindo características específicas por meio da aplicação repetida de operações matemáticas locais, compartilhando os mesmos pesos ao longo de toda a entrada (GOODFELLOW; BENGIO; COURVILLE, 2016).

O funcionamento de uma CNN pode ser dividido nas seguintes etapas principais:

a) Camada Convolucional: aplica filtros que percorrem os dados, extraindo características locais. Cada filtro aprende a detectar um padrão específico nos dados.

b) Camada de Pooling (subamostragem): reduz a dimensionalidade dos mapas de ativação gerados na convolução, mantendo as informações mais relevantes e ajudando a prevenir o sobreajuste.

c) Camada Flatten: transforma os mapas de ativação em um vetor unidimensional, para que possa ser processado por camadas densas.

d) Camadas Densas: realizam a etapa final de processamento, combinando as características extraídas para produzir a predição.

Matematicamente, a operação de convolução em uma dimensão, frequentemente utilizada na análise de séries temporais, é expressa pela Equação 2.4:

$$s(t) = (x * w)(t) = \sum_{i=0}^{k-1} x(t+i) \cdot w(i) \tag{2.4}$$

em que:

a) x representa os dados de entrada;

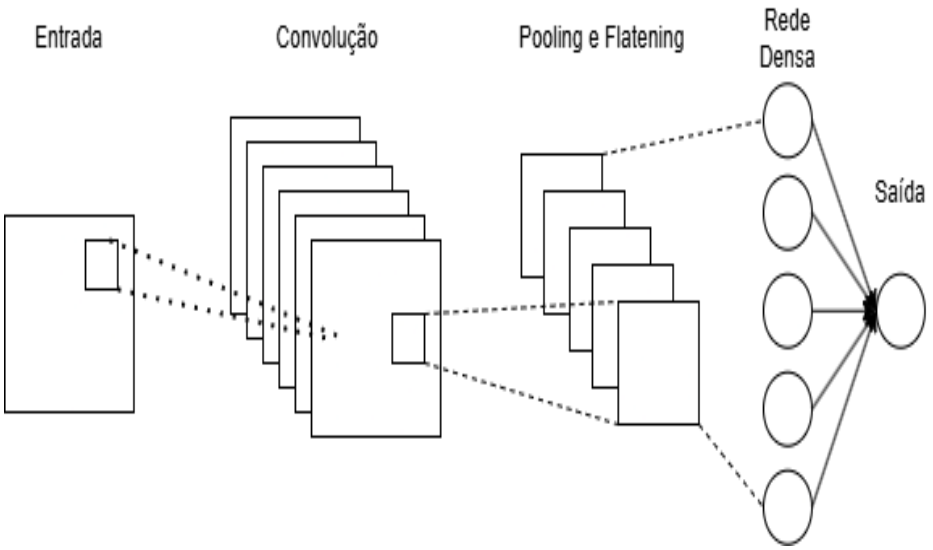
b) w é o filtro (ou kernel) de tamanho k;

c) $s(t)$ é o valor da saída da convolução na posição t.

Cada filtro é capaz de capturar um padrão específico presente na sequência de entrada. Durante o treinamento, os filtros aprendem automaticamente quais padrões são mais relevantes para a tarefa, seja ela de classificação ou regressão.

O fluxo de dados em uma CNN pode ser visualizado na Figura 4, que ilustra a passagem dos dados pela camada convolucional, seguida pela camada de pooling, flatten e, por fim, pelas camadas densas responsáveis pela saída.

Figura 4 – Esquema de CNN



Elaboração: autor.

O processo de treinamento de uma CNN segue os mesmos princípios fundamentais das redes MLP, envolvendo propagação direta, cálculo da função de perda e retropropagação para ajuste dos filtros e dos pesos das camadas densas. O pseudocódigo 2 resume esse funcionamento.

Algoritmo 2: Treinamento de modelo CNN com lotes

Entrada: Dados de entrada $(x_{\text{lote}}, y_{\text{lote}})$

Saída: Modelo treinado com filtros e pesos atualizados

```
modelo ← inicializar_cnn()
para cada época faça
    para cada lote (x_lote, y_lote) faça
        mapas_ativacao ← convoluir(x_lote)
        mapas_reduzidos ← pooling(mapas_ativacao)
        vetor ← flatten(mapas_reduzidos)
        saida ← camada_densa(vetor)
        erro ← calcular_erro(saida, y_lote)
        gradientes ← retropropagar(erro)
        atualizar_filtros_e_pesos(modelo, gradientes)
```

Portanto, as redes neurais convolucionais representam uma poderosa ferramenta dentro do campo do aprendizado profundo, especialmente quando se deseja capturar padrões locais presentes nos dados. No contexto deste trabalho, sua aplicação visa explorar dependências locais em séries temporais financeiras, para depois comparar com o desempenho das outras redes e principalmente as recorrentes, uma vez que em trabalhos relacionados observa-se que ambas competem pelos melhores desempenhos - basicamente um embate entre melhores resultados vindos da observação de padrões curtos (CNN) e padrões longos (RNN).

2.6 REDES NEURAIS RECORRENTES (RNN) E LSTM

As Redes Neurais Recorrentes (RNN) são uma classe de redes neurais projetadas especificamente para lidar com dados sequenciais, como séries temporais, texto ou sinais. Ao contrário das redes MLP e CNN, que assumem que todas as entradas são independentes, as RNN incorporam uma memória interna que permite considerar informações de passos anteriores da sequência durante o processamento atual (GOODFELLOW; BENGIO; COURVILLE, 2016).

Matematicamente, uma RNN simples realiza, a cada passo temporal t, os seguintes cálculos:

$$h_t = \phi(U_o h_{t-1} + W_o x_t + b) \tag{2.5}$$

em que:

a) x_t representa a entrada no tempo t;

b) h_t é o estado oculto no tempo t, que funciona como uma memória da rede e saída desse neurônio específico;

c) W_o representa o vetor de pesos de entrada da rede;

d) U_o representa o vetor de pesos recorrentes da rede;

e) b é o termo de viés;

f) ϕ é a função de ativação utilizada no estado oculto.

O estado oculto h_t permite que a RNN armazene informações sobre entradas anteriores, possibilitando que o modelo capture dependências temporais na série de dados.

No entanto, as RNN simples enfrentam sérias limitações quando tentam capturar dependências de longo prazo devido ao problema conhecido como gradiente desvanecido ou gradiente explosivo. Esse fenômeno ocorre durante a retropropagação através do tempo (*Backpropagation Through Time* — BPTT), levando os gradientes a se tornarem extremamente pequenos (dificultando o aprendizado) ou extremamente grandes (instabilidade) (BENGIO; SIMARD; FRASCONI, 1994).

Para superar essas limitações, foram desenvolvidas as redes *Long Short-Term Memory* (LSTM) (HOCHREITER; SCHMIDHUBER, 1997), que introduzem uma estrutura interna mais complexa composta por portas que controlam o fluxo de informações.

A célula LSTM possui três portas principais:

- Porta de Esquecimento (f_t): decide quais informações devem ser descartadas do estado da célula;
- Porta de Entrada (i_t): decide quais novas informações serão armazenadas;
- Porta de Saída (o_t): controla quais informações serão passadas para a saída e para o próximo estado oculto.

O funcionamento da célula LSTM é descrito pelas Equações 2.6:

$$\begin{aligned} f_t &= \sigma(U_f h_{t-1} + W_f x_t + b_f) \\ i_t &= \sigma(U_i h_{t-1} + W_i x_t + b_i) \\ k_t &= \sigma(U_k h_{t-1} + W_k x_t + b_k) \\ c_t &= f_t \odot c_{t-1} + i_t \odot k_t \\ o_t &= \sigma(U_o h_{t-1} + W_o x_t + V_o c_t + b_o) \\ h_t &= o_t \odot \sigma(c_t) \end{aligned} \tag{2.6}$$

em que:

a) x_t representa a entrada no tempo t;

b) h_t é o estado oculto no tempo t, ou seja, a saída da célula LSTM;

c) c_t é o estado de célula no tempo t (também chamado de sinal de memória ou carry);

d) f_t, i_t, o_t são, respectivamente, as ativações das portas de esquecimento, entrada e saída;

e) k_t é a informação candidata a ser adicionada à célula;

f) W_* representa os pesos associados à entrada x_t para cada porta;

g) U_* representa os pesos associados ao estado oculto anterior h_{t-1} para cada porta;

h) V_o representa os pesos aplicados ao sinal de memória;

i) b_* são os termos de viés;

j) σ é a função de ativação;

k) $\odot$ denota o produto elemento a elemento.

A grande inovação da LSTM é a manutenção do estado da célula (C_t), que permite preservar informações ao longo de sequências longas, mitigando significativamente o problema do gradiente desvanecido.

Ademais, vale notar que por conta dessa maior quantidade de parâmetros da LSTM e consequente capacidade superior de modelar dependências temporais de longo prazo, essas redes apresentam uma tendência mais acentuada ao sobreajuste em comparação com arquiteturas mais simples. Essa característica torna as LSTMs particularmente sensíveis a cenários com dados escassos ou de alta dimensionalidade, o que é comum em aplicações de séries temporais e processamento de linguagem natural. O elevado número de parâmetros das LSTMs pode levar a um ajuste excessivo aos dados de treino, comprometendo sua capacidade de generalização (CHOLLET, 2021; GAL; GHAHRAMANI, 2016). Por essa razão, estratégias de regularização, como o uso de *dropout* recorrente, penalização por norma L2 e divisão rigorosa dos dados em conjuntos de treino, validação e teste, tornam-se indispensáveis no desenvolvimento de modelos baseados nessa arquitetura.

O processo de treinamento das RNN e LSTM segue a lógica comum das redes neurais, mas com a retropropagação estendida no tempo, o que permite ajustar os pesos levando em consideração toda a sequência.

O pseudocódigo 3 resume esse funcionamento:

Algoritmo 3: Treinamento de modelo RNN/LSTM com lotes

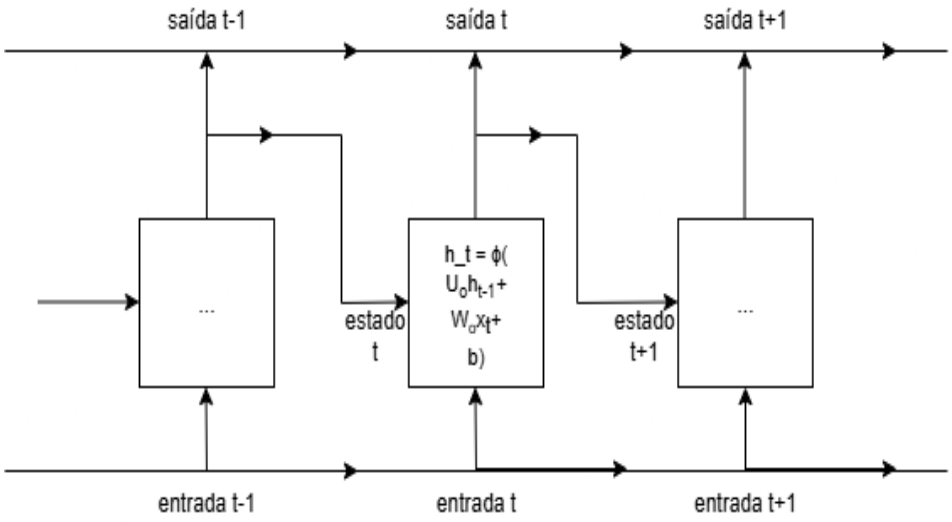
Entrada: Sequências de entrada $(x_{\text{lote}}, y_{\text{lote}})$

Saída: Modelo treinado com pesos atualizados

```
modelo ← inicializar_rnn()
para cada época faça
    para cada sequência (x_seq, y_seq) faça
        estado ← inicializar_estado()
        para cada passo t na sequência faça
            saida, estado ← modelo(x_t, estado)
        erro ← calcular_erro(saida, y_seq)
        gradientes ← retropropagar_atraves_do_tempo(erro)
        atualizar_pesos(modelo, gradientes)
```

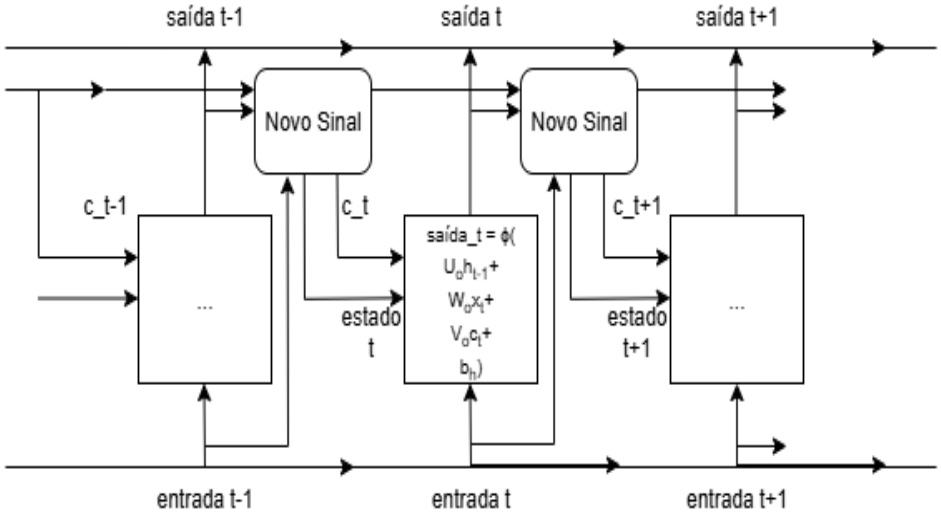
A Figura 5 ilustra o funcionamento de uma rede recorrente, destacando o fluxo de informações ao longo do tempo. Já a Figura 6 ilustra o funcionamento de uma LSTM. Vale notar como principal diferença entre elas a presença do "Novo Sinal"na LSTM, que internamente opera sobre as três portas já citadas.

Figura 5 – Anatomia de uma RNN simples



Elaboração: autor.

Figura 6 – Anatomia de uma LSTM



Elaboração: autor.

Portanto, as redes recorrentes, especialmente as variantes baseadas em LSTM, desempenham um papel central na modelagem de séries temporais e por muitos são consideradas o estado da arte para modelagem deste tipo de problema.

3 METOLODIA

3.1 ESCOLHA DOS DADOS

A seleção criteriosa da base de dados é uma etapa fundamental para garantir a robustez e a confiabilidade dos modelos de previsão baseados em aprendizado profundo. Conforme discutido no capítulo 2, existem múltiplos tipos de contratos futuros relacionados ao milho, negociados em diversas bolsas ao redor do mundo, cada qual com suas particularidades em termos de liquidez, regras de negociação e sazonalidades regionais.

Neste trabalho, optou-se por utilizar os dados históricos dos contratos futuros de milho negociados na Bolsa de Chicago (Chicago Board of Trade — CBOT), parte do grupo CME (Chicago Mercantile Exchange). A CME é reconhecida globalmente como a maior bolsa de derivativos do mundo, apresentando alta liquidez e volume expressivo de operações, o que favorece a construção de modelos preditivos mais precisos e generalizáveis.

Importante destacar que os dados utilizados não correspondem a um único contrato futuro com vencimento fixo, mas sim a uma série contínua ajustada, construída a partir da cotação do contrato mais líquido disponível em cada momento. Essa abordagem é amplamente utilizada em análises históricas, pois permite a construção de uma série temporal mais longa e contínua, livre das descontinuidades naturais que ocorrem com o vencimento periódico dos contratos futuros. A plataforma `Investing.com`, utilizada como fonte dos dados, fornece essa série sintética ajustada automaticamente, refletindo os preços diários do contrato de milho mais representativo em termos de volume negociado.

O intervalo de tempo analisado compreende o período entre janeiro de 2010 e dezembro de 2018. Esse recorte foi escolhido estrategicamente para garantir uma quantidade suficiente de observações e, ao mesmo tempo, evitar distorções oriundas de eventos atípicos, como a elevada volatilidade causada pela pandemia de COVID-19 a partir de 2020.

Para cada dia útil de negociação dentro do período analisado, foram coletadas as seguintes variáveis: preço de abertura (*open*), preço de fechamento (*close*), maior preço do dia (*high*), menor preço do dia (*low*) e volume negociado. O volume é apresentado em milhares de contratos e indica o nível de atividade do mercado em cada sessão.

Além disso, optou-se por trabalhar com valores diários, dada a maior frequência de informações e o maior volume de observações para o treinamento dos modelos. A escolha por granularidade diária também permite capturar melhor a dinâmica dos preços e o comportamento sazonal típico de commodities agrícolas como o milho.

3.2 ANÁLISE EXPLORATÓRIA E PRÉ-PROCESSAMENTO

Com a base de dados dos contratos futuros de milho devidamente coletada, realizou-se uma etapa fundamental de análise exploratória e pré-processamento dos dados, com o objetivo de extrair padrões relevantes, lidar com possíveis inconsistências e preparar as variáveis para posterior utilização nos modelos de aprendizado profundo.

Inicialmente, os dados brutos foram carregados com o auxílio da biblioteca `pandas` e tratados para padronização das colunas. A coluna de preços de fechamento foi renomeada para `Close`, e a de volume de negociações para `Volume`. Esta última apresentava os valores no formato de milhares (indicado por "K"), sendo necessário convertê-los para tipo numérico. Além disso, a coluna `Change %`, que apresentava variações percentuais já agregadas, foi descartada por não ser necessária neste estudo, dado que tais variações foram calculadas diretamente para cada variável de forma personalizada.

Código 1 – Trecho de código de carreagmento e formatação inicial dos dados

```
import pandas as pd
corn_futures = pd.read_csv('US Corn Futures Historical Data.csv',
    index_col='Date', parse_dates=True)
corn_futures.rename(columns={'Price':'Close','Vol.':'Volume'}, inplace=
    True)
corn_futures['Volume'] = corn_futures['Volume'].str.replace('K','').
    astype(float)
corn_futures.drop(columns=['Change %'], inplace=True)
```

Na sequência, foram criadas colunas de variação percentual diária para cada uma das variáveis disponíveis, como fechamento, volume, preços máximos e mínimos. Essa transformação permitiu observar oscilações relativas entre os dias, o que é especialmente relevante para a modelagem de séries temporais financeiras. A primeira linha resultante dessas transformações foi descartada por conter valores nulos decorrentes do cálculo da variação percentual.

Código 2 – Cálculo das variações percentuais diárias

```
import pandas as pd
for col in corn_futures.columns:
    corn_futures[col + '_%Chg'] = corn_futures[col].pct_change()
corn_futures.drop(corn_futures.index[0], inplace=True)
```

Em relação à integridade da base, foi realizada uma verificação completa de valores nulos. Constatou-se que apenas a coluna `Volume` apresentava ausências, as quais indicam dias sem negociação ativa. Optou-se por preencher esses valores com zero, mantendo a coerência da série temporal sem realizar imputações que poderiam enviesar os dados.

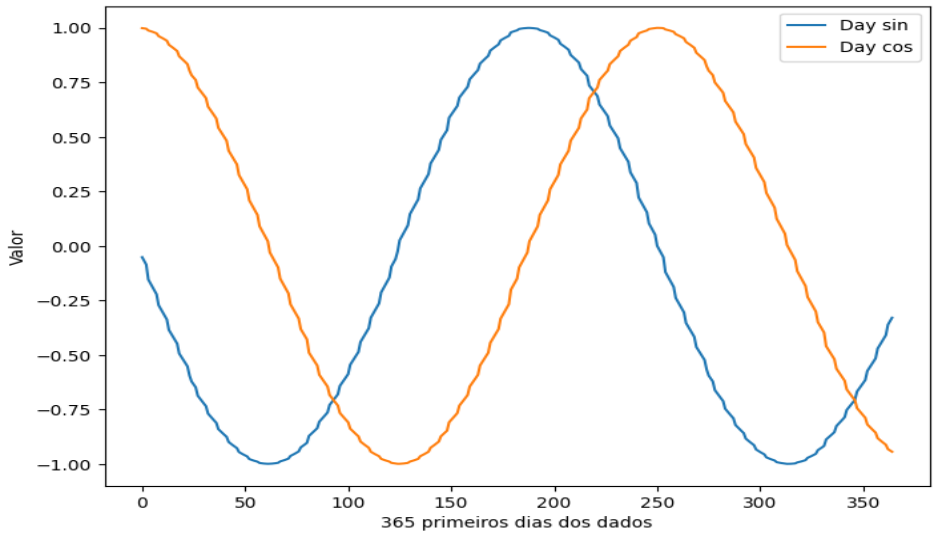
Para incorporar possíveis padrões sazonais ligados ao ciclo agrícola do milho, foram adicionados dois novos atributos temporais baseados na data: as funções seno e cosseno do dia do ano. Essa técnica, comumente utilizada em problemas sazonais, transforma a

informação de tempo em representações periódicas contínuas que auxiliam os modelos na detecção de recorrências anuais, como épocas de colheita, plantio ou variações climáticas. Representações cíclicas como essas são recomendadas especialmente quando se trabalha com variáveis temporais periódicas, pois evitam descontinuidades artificiais — por exemplo, entre os dias 31 de dezembro e 1º de janeiro — que poderiam confundir os modelos baseados em redes neurais recorrentes ou convolucionais (BROWNLEE, 2018). O valor que as variáveis assumem podem ser verificados na Figura 7 abaixo.

Código 3 – Adição de componentes sazonais com funções trigonométricas

```
corn_futures.insert(0, 'Day sin', np.sin(corn_futures.index.dayofyear *
    (2 * np.pi / 365)))
corn_futures.insert(1, 'Day cos', np.cos(corn_futures.index.dayofyear *
    (2 * np.pi / 365)))
```

Figura 7 – Componentes sazonais com funções seno e cosseno para os dias do ano

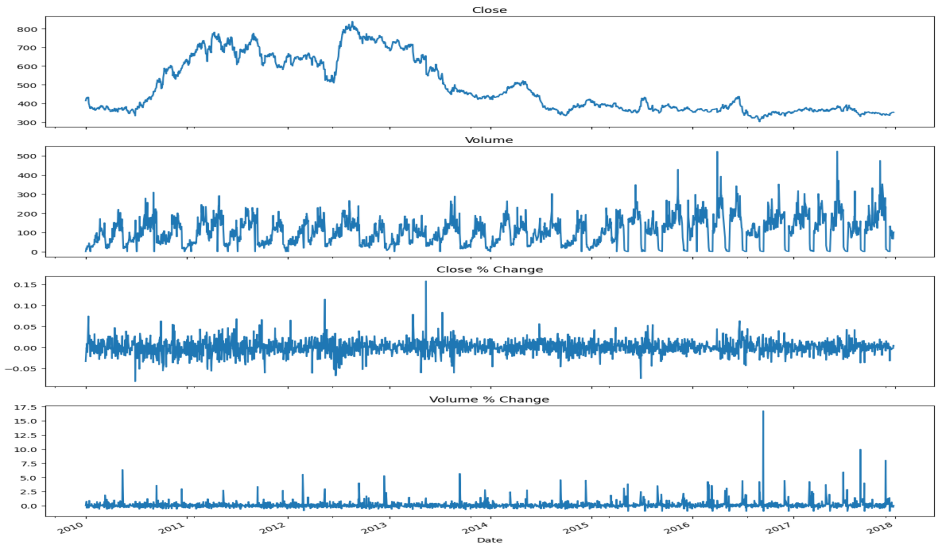


Fonte: CME. Elaboração: autor.

Além disso, foram gerados gráficos exploratórios das principais variáveis e suas respectivas variações percentuais. Esses gráficos (Figura 8) permitiram observar a presença de picos ou padrões anômalos, como outliers. Embora existam algumas oscilações bruscas, estas foram interpretadas como legítimas, representando momentos de elevada volatilidade provavelmente causados por choques de oferta, eventos climáticos, decisões regulatórias ou variações no comércio internacional.

Por fim, antes de alimentar os modelos preditivos, realizou-se uma normalização dos dados. Todas as colunas foram padronizadas por meio da subtração da média e divisão pelo desvio padrão de cada variável, conforme Equação 3.1. Esse procedimento é essencial para garantir que nenhuma variável com magnitude mais alta domine o processo de aprendizado da rede neural. A padronização z-score, também conhecida como normalização estatística, é amplamente adotada em aplicações de aprendizado profundo por sua

Figura 8 – Séries temporais das variáveis: fechamento, volume e suas respectivas variações percentuais



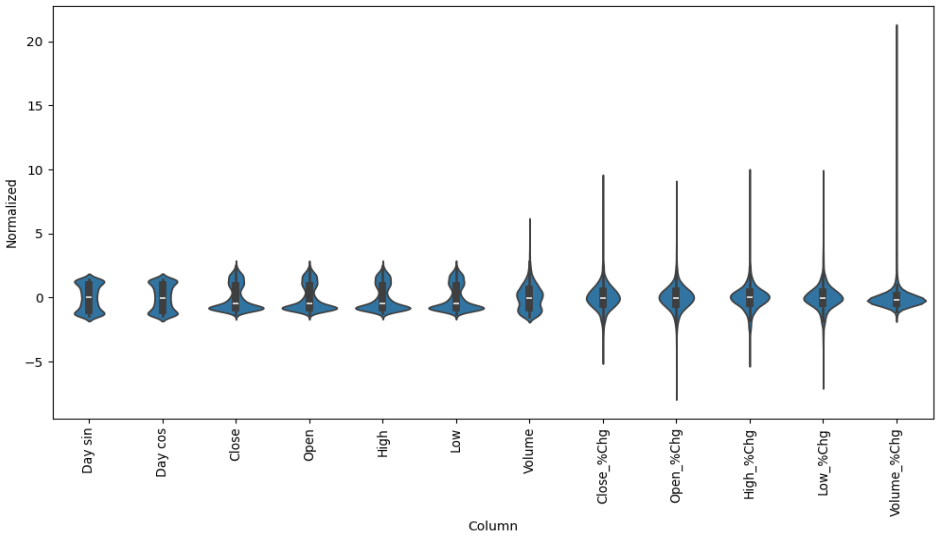
Fonte: CME. Elaboração: autor.

capacidade de acelerar a convergência durante o treinamento e melhorar a estabilidade numérica dos algoritmos (GOODFELLOW; BENGIO; COURVILLE, 2016).

$$x_{norm} = \frac{x - \mu}{\sigma} \tag{3.1}$$

Essa etapa resultou em um conjunto de dados final mais limpo, coerente e numericamente estável, como pode ser visto na Figura 9. Ainda assim notam-se alguns ruídos nas colunas de variações mas consideradas normais pela volatilidade de mercados.

Figura 9 – Distribuição dos valores das colunas pós normalização



Fonte: CME. Elaboração: autor.

A etapa final do pré-processamento consistiu na divisão do conjunto de dados normalizado em três subconjuntos: treino (60%), validação (20%) e teste (20%). Essa divisão respeitou a ordem temporal das observações, o que é fundamental em problemas de séries temporais para evitar vazamento de informação do futuro para o passado.

Código 4 – Função de divisão do conjunto de dados

```
def split_df(df):
    n = len(df)
    train_df = df[0:int(n*0.6)]
    val_df = df[int(n*0.6):int(n*0.8)]
    test_df = df[int(n*0.8):]
    train_mean = train_df.mean()
    train_std = train_df.std()
    train_df = (train_df - train_mean) / train_std
    val_df = (val_df - train_mean) / train_std
    test_df = (test_df - train_mean) / train_std
    return train_df, val_df, test_df
```

O conjunto de validação desempenha um papel essencial no desenvolvimento de modelos robustos de aprendizado profundo. Sua utilização permite acompanhar o desempenho do modelo durante o treinamento, auxiliando na detecção de sobreajuste (*overfitting*) e na realização de ajustes de hiperparâmetros de forma segura. A separação explícita desses dados garante que as decisões sobre arquitetura e treinamento sejam baseadas em informações não vistas, o que favorece uma melhor generalização do modelo (CHOLLET, 2021).

3.3 JANELAS TEMPORAIS

Um dos principais desafios ao se aplicar redes neurais artificiais em séries temporais é adaptar os dados sequenciais para um formato compatível com o treinamento supervisionado. Para isso, foi empregada a técnica de janelas deslizantes (*sliding windows*), que transforma a série original em pares de entrada-saída. Cada janela define um intervalo fixo de tempo anterior (entrada) e um ou mais passos futuros (rótulo), viabilizando a previsão.

Conforme discutido por Chollet (2021), esse tipo de segmentação permite capturar padrões temporais relevantes, além de facilitar o treinamento ao tornar explícitas as dependências sequenciais no tempo. Também é um método amplamente utilizado na literatura para tarefas de previsão multivariada e univariada em séries temporais (BROWNLEE, 2018).

Neste trabalho, utilizou-se uma classe chamada `WindowGenerator`, inspirada do tutorial oficial de séries temporais do TensorFlow[1], a qual automatiza a criação das janelas

[1] https://www.tensorflow.org/tutorials/structured_data/time_series

de entrada e seus respectivos rótulos. A classe recebe como parâmetros principais a largura da entrada (`input_width`), o horizonte de previsão (`label_width`) e o deslocamento temporal (`shift`), além dos dataframes de treino, validação e teste.

Foi escolhido neste trabalho a utilização de uma janela de entrada de 20 dados, a fim de representar aproximadamente um mês de pregão regular — considerando quatro semanas úteis com cinco dias de negociação (segunda a sexta-feira), totalizando 20 dias úteis. Assim, espera-se uma janela longa o suficiente para capturar tendências de curto prazo, mas não excessivamente longa a ponto de diluir padrões de mais alta frequência. Abaixo está um exemplo da instância utilizada nos modelos deste trabalho:

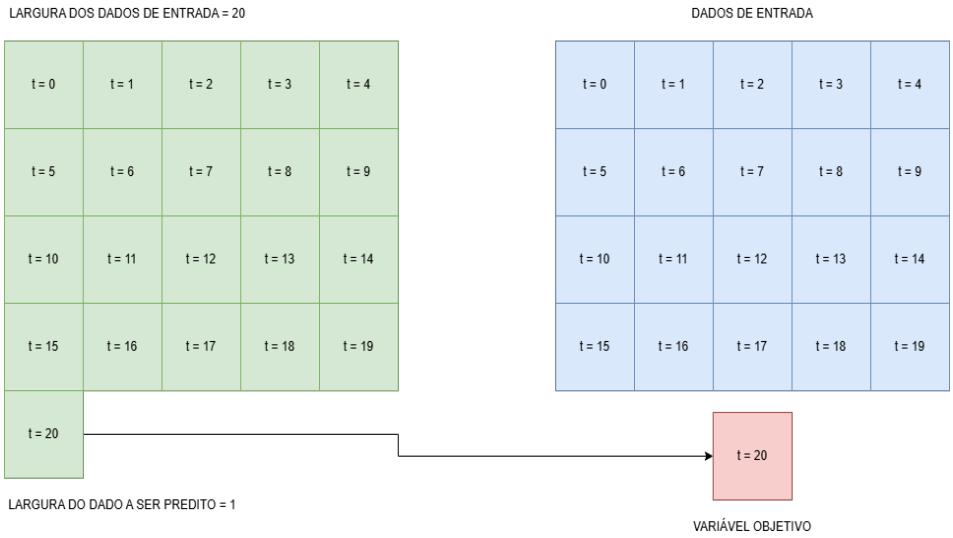
Código 5 – Instanciando a janela de tempo utilizada nos experimentos

```
w_month = WindowGenerator(input_width=20, label_width=1, shift=1,
                          train_df = train_df, val_df = val_df, test_df
                             = test_df,
                          label_columns=['Close'])
```

O código acima indica que, para cada previsão, são utilizados os 20 dias anteriores da série como entrada, com o objetivo de prever o valor de fechamento do dia seguinte.

A Figura 10 ilustra visualmente o funcionamento da janela deslizante utilizada nos modelos do presentre estudo.

Figura 10 – Representação da janela temporal utilizada



Elaboração: autor.

Essa abordagem foi fundamental para estruturar os dados de forma compatível com os modelos de aprendizado profundo utilizados.

3.4 MÉTRICAS DE AVALIAÇÃO

A avaliação de desempenho dos modelos implementados neste trabalho é realizada com base na métrica *Mean Absolute Error* (MAE), ou erro absoluto médio. Esta métrica foi escolhida por sua simplicidade interpretativa e por ser amplamente empregada em tarefas de regressão envolvendo séries temporais financeiras, uma vez que fornece uma estimativa direta do erro médio entre os valores previstos e os observados, na mesma unidade da variável-alvo.

Matematicamente, o MAE é definido abaixo, na Equação 3.2:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{3.2}$$

onde y_i representa o valor real da série no instante i, $\hat{y}_i$ é o valor previsto pelo modelo para o mesmo instante, e n é o número total de observações no conjunto avaliado.

Durante o processo de treinamento dos modelos, foi utilizada uma função de compilação unificada, que define a função de perda (*loss*) como o *Mean Squared Error* (MSE) e a métrica de avaliação como o MAE. A escolha do MSE como função de perda se justifica por sua capacidade de penalizar erros maiores de forma mais acentuada, favorecendo a estabilidade do processo de otimização por gradiente durante o treinamento.

Matematicamente, o MSE é definido na Equação 3.3:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{3.3}$$

Já o MAE foi mantido como métrica de avaliação por ser mais robusto à presença de *outliers* e por fornecer uma estimativa mais interpretável do erro médio.

É importante destacar que não há contradição técnica entre o uso de MSE para a função de perda e MAE como métrica de desempenho, e está presente na literatura. De fato, essa combinação é bastante comum em problemas de regressão: enquanto o MSE guia o processo de aprendizado com maior sensibilidade a erros grandes, o MAE é utilizado para avaliar com maior realismo a performance do modelo final (CHOLLET, 2021; BROWNLEE, 2018; GOODFELLOW; BENGIO; COURVILLE, 2016).

Para escolha da melhor arquitetura para cada modelo, com exceção do modelo base, foram realizados treinamentos exaustivos, variando-se profundidade da rede, número de neurônios por camada, funções de ativação, regularizações e outros hiperparâmetros relevantes. A cada treinamento, foi gerado um gráfico da função de perda ao longo das épocas para os conjuntos de treino e validação. Esses gráficos serviram de suporte para análise do comportamento do modelo quanto ao aprendizado e à ocorrência de *overfitting*. Essa abordagem está de acordo com as diretrizes apresentadas por Chollet (2021), que recomenda observar as curvas de perda de treino e validação ao longo das épocas para identificar tendências de generalização ou sobreajuste, em vez de tomar decisões com base

apenas no menor valor final da métrica de validação (CHOLLET, 2021). Essa estratégia de verificação do desempenho dos modelos também valida a escolha de possuir um conjunto de dados de validação, diferente do de treino e diferente do de teste.

Além disso, ao final de cada experimento, os valores finais da função de perda e do MAE foram registrados em arquivos `.json`, permitindo a comparação sistemática entre diferentes configurações. O processo de escolha da arquitetura final se baseou, portanto, no menor erro observado em conjunto com análise da curva de validação, priorizando modelos cuja descida da função de perda se mostrou mais estável e expressiva até o ponto de início do sobreajuste.

Com isso, uma vez escolhida a arquitetura ideal para representar cada tipo de modelo, caso ao longo das 500 épocas tenha-se observado sobreajuste, uma nova rodada de treinamento foi conduzida com menos épocas, de forma a interromper o aprendizado no ponto anterior ao início do sobreajuste.

Por fim, a avaliação definitiva de cada modelo é realizada sobre o conjunto de teste, utilizando como métrica principal o MAE. Esse valor final serve como referência para aferir a capacidade preditiva dos modelos quanto ao comportamento futuro dos preços dos contratos futuros de milho, e é também a métrica que orienta as comparações entre as abordagens desenvolvidas.

3.5 MODELOS

3.5.1 Modelo Base

O modelo base foi desenvolvido como uma heurística simples para servir como referência de desempenho para os modelos mais complexos. Sua importância reside tanto na avaliação relativa da eficácia dos modelos mais elaborados quanto na constatação de que, em problemas reais, até mesmo predições aparentemente triviais podem ser difíceis de superar, como explicado por Chollet (2021), e sua utilização é muito recomendada e orientada em trabalhos similares justamente para maior poder comparativo da precisão dos modelos (CHOLLET, 2021).

Neste trabalho, o modelo base foi definido da seguinte forma: assume-se que o preço de fechamento do contrato futuro de milho no dia seguinte $(D + 1)$ será igual ao preço de fechamento do dia atual (D). Essa abordagem, apesar de ingênua, já incorpora uma premissa válida sobre a continuidade e autocorrelação dos preços em séries temporais financeiras.

$$\text{Previsão}(D + 1) = \text{Preco}(D) \tag{3.4}$$

A implementação do modelo é realizada utilizando TensorFlow. A classe criada `Baseline`, conforme código abaixo, herda de `tf.keras.Model` e retorna, como saída,

o próprio valor de entrada correspondente ao índice da variável `Close` (preço de fechamento). Dessa forma, não há aprendizado de pesos ou ajustes de parâmetros – o modelo simplesmente propaga o valor da entrada como previsão.

Código 6 – Definição da classe do modelo base

```
from keras.saving import register_keras_serializable

class Baseline(tf.keras.Model):
    def __init__(self, label_index=None):
        super().__init__()
        self.label_index = label_index

    def call(self, inputs):
        if self.label_index is None:
            return inputs
        result = inputs[:, :, self.label_index]
        return result[:, :, tf.newaxis]
```

O desempenho do modelo base foi avaliado sobre os conjuntos de validação e teste, e seus resultados salvos para análise posterior.

3.5.2 Modelo Denso

A primeira abordagem avaliada neste trabalho é um modelo denso simples, cuja escolha se fundamenta na intenção de compreender o desempenho básico de aprendizado profundo ao tentar captar padrões nos preços dos contratos futuros de milho. Essa estratégia serve como um ponto de partida para avaliar até que ponto uma estrutura puramente densa, sem mecanismos específicos para lidar com sequências temporais, pode capturar dinâmicas relevantes nos dados. Como o objetivo é também analisar o comportamento do processo iterativo de aprendizado, este modelo permite uma visualização clara dos efeitos de técnicas fundamentais de regularização e controle de complexidade: *dropout* e penalização L2.

Diversas arquiteturas foram testadas, variando-se a quantidade de camadas ocultas e o número de neurônios em cada uma delas. As arquiteturas incluem combinações com camadas com 16, 32 e 64 neurônios. Para cada estrutura, foram aplicadas variações nas taxas de *dropout* (0.3 e 0.5) e na força da regularização L2 (1×10^{-3} e 1×10^{-4}), totalizando várias de combinações diferentes. O objetivo desses testes foi avaliar a robustez do modelo frente ao risco de *overfitting* e identificar combinações que promovem melhor generalização.

Os hiperparâmetros gerais utilizados em todos os experimentos com este modelo incluem o uso do otimizador *Adam*, considerado eficiente e robusto para tarefas de regressão com redes neurais, com um tamanho de lote fixado em 32 amostras e um número máximo de 500 épocas de treinamento. A função de ativação escolhida para as camadas ocultas

foi a a *Rectified Linear Unit* (ReLU). Essa função adiciona um aspecto de não linearidade à previsão, e funciona escolhendo apenas valores positivos ou iguais a zero. Foi escolhida devido à sua simplicidade e referência de bons resultados na literatura.

A implementação do modelo foi realizada por meio da função `make_dense_model`, reproduzida abaixo, a qual permite a criação programática das diferentes arquiteturas testadas:

Código 7 – Definição função de criação dos modelos densos

```
import tensorflow as tf
from tensorflow.keras import layers, regularizers
def make_dense_model(layer_units, dropout_rate=0.2, l2_strength=1e-4):

    model = tf.keras.Sequential()

    for units in layer_units:
        model.add(layers.Dense(
            units=units,
            activation='relu',
            kernel_regularizer=regularizers.l2(l2_strength)
        ))
        model.add(layers.Dropout(rate=dropout_rate))

    model.add(layers.Dense(units=1))  # camada de saída
    return model
```

A função recebe como argumentos uma lista `layer_units`, que define a quantidade de camadas ocultas e o número de neurônios em cada uma, a taxa de *dropout* aplicada entre essas camadas, e a força da regularização L2. Dessa forma, foi possível automatizar a avaliação de diversas configurações, permitindo uma análise sistemática do impacto de cada hiperparâmetro no desempenho do modelo.

Essa abordagem serviu como primeiro experimento para avaliar a capacidade de generalização de modelos densos em séries temporais financeiras. É um modelo simples de redes neurais, e verificar seu desempenho é crucial para comparação com os posteriores modelos mais complexos e entendimento de como as funções de perda se comportam. Espera-se que esse modelo performe pior do que os posteriores.

3.5.3 Modelo Convolucional

A escolha por testar uma rede neural convolucional (CNN) se deu pelo fato de esse tipo de arquitetura, embora majoritariamente aplicada em tarefas de visão computacional, também aparecer em diversas literaturas como alternativa viável e eficaz para a modelagem de séries temporais. Como discutido na Seção 2.1, há aplicações bem-sucedidas desse modelo em problemas similares de previsão, o que justifica sua inclusão neste trabalho, tanto pelo potencial preditivo quanto pelo valor comparativo. Dessa forma, a CNN foi uti-

lizada com o intuito de enriquecer a análise com uma arquitetura consagrada em outras áreas, observando sua performance neste contexto específico da previsão dos contratos futuros de milho.

Foram testadas diversas arquiteturas convolucionais com diferentes quantidades de camadas e filtros. As combinações incluíram variações de profundidade como uma, duas ou três camadas convolucionais com 16, 32, 64, 128 ou 256 neurônios por camada. Além disso, também foram aplicadas diferentes taxas de *dropout* (0.3 e 0.5) e forças de regularização L2 (10^{-3} e 10^{-4}), buscando maior robustez e controle sobre o *overfitting*. O tamanho do *kernel* foi fixado em 5 para todas as configurações, representando aproximadamente uma janela de cinco dias úteis, equivalente a uma semana de pregão regular.

Os hiperparâmetros gerais utilizados nesse conjunto de experimentos foram mantidos similares aos demais modelos: tamanho de lote igual a 32, número de épocas de treinamento igual a 500, função de ativação $ReLU$ nas camadas convolucionais e otimizador *Adam*. O parâmetro de *padding* foi definido como *same*, o que garante que a saída das convoluções mantenha o mesmo comprimento da entrada. Isso é importante para preservar a estrutura temporal da série ao longo das camadas, evitando que o sinal perca resolução conforme é processado.

A seguir, apresenta-se a função `make_cnn`, que foi utilizada para construir os diferentes modelos convolucionais com base nos hiperparâmetros escolhidos:

Código 8 – Definição função de criação dos modelos convolucionais

```
import tensorflow as tf
from tensorflow.keras import layers, regularizers
def make_cnn(filters_list, kernel_size=5, dropout_rate=0.2, l2_strength
    =1e-4):

    model = tf.keras.Sequential()

    for i, filters in enumerate(filters_list):
        return_sequences = i < len(filters_list) - 1  # todas exceto a
            ltima  mant m sequ ncia
        model.add(layers.Conv1D(
            filters=filters,
            kernel_size=kernel_size,
            activation='relu',
            padding='same',
            kernel_regularizer=regularizers.l2(l2_strength)
        ))
        model.add(layers.Dropout(rate=dropout_rate))

    model.add(layers.Flatten())
    model.add(layers.Dense(units=1))
    return model
```

A estrutura da função permite a fácil variação dos parâmetros e a realização de testes com diferentes arquiteturas. A aplicação do *dropout* após cada camada convolucional atua como mecanismo de regularização, e a regularização L2 adicionada ao kernel busca penalizar pesos excessivos, promovendo maior generalização do modelo. É esperado que o desempenho desse modelo supere o modelo denso e tenha resultados bons, podendo até ser o melhor entre todos os modelos, como apontado por alguns trabalhos anteriores na Seção 2.1.

3.5.4 Modelo Recorrente

A Rede Neural Recorrente (RNN) simples foi escolhida devido à sua relevância histórica no tratamento de dados sequenciais. Representando uma das primeiras arquiteturas desenvolvidas com a finalidade de capturar relações temporais, as RNNs são amplamente utilizadas em tarefas de predição de séries temporais e oferecem bons resultados, apesar de limitações conhecidas como o desvanecimento do gradiente. Neste trabalho, a proposta de utilizá-las visa enriquecer a análise comparativa ao se aplicar uma abordagem recorrente mais simples, antes da introdução de modelos mais sofisticados como as LSTM, consideradas atualmente o estado da arte nesse domínio.

Durante o desenvolvimento do modelo RNN, diversas arquiteturas foram testadas

variando-se a quantidade de camadas, o número de unidades por camada e os mecanismos de regularização. Foram avaliadas estruturas com uma, duas ou três camadas compostas por 16, 32, 64, 128 ou 256 unidades, com e sem aplicação de *dropout* e regularização L2. A ideia foi verificar a capacidade de generalização e o comportamento do modelo frente a diferentes profundidades e larguras, bem como diferentes combinações de regularização.

Os hiperparâmetros gerais escolhidos para o treinamento da RNN seguiram o padrão adotado nos outros modelos do trabalho, com tamanho de lote igual a 32, 500 épocas de treinamento, função de ativação `tanh` (tangente hiperbólica) e o otimizador `Adam`. A mudança da função ReLU para a tangente hiperbólica segue a recomendação clássica para redes recorrentes simples, pois ajuda no controle do intervalo dos valores ativados. Além disso, o argumento `return_sequences` foi configurado como verdadeiro para todas as camadas, exceto a última, garantindo que as informações temporais intermediárias fossem passadas adiante nas camadas recorrentes, de modo a preservar a característica sequencial do dado ao longo do fluxo da rede.

A seguir, apresenta-se a função `make_rnn`, que foi utilizada para construir os diferentes modelos recorrentes com base nos hiperparâmetros escolhidos:

Código 9 – Definição função de criação dos modelos recorrentes

```
import tensorflow as tf
from tensorflow.keras import layers, regularizers

def make_rnn(units_list, dropout_rate=0.2, l2_strength=1e-4):

    model = tf.keras.Sequential()

    for i, units in enumerate(units_list):
        return_sequences = i < len(units_list) - 1  # True para todas
            menos a  ltima
        model.add(layers.SimpleRNN(
            units=units,
            return_sequences=return_sequences,
            activation='tanh',
            kernel_regularizer=regularizers.l2(l2_strength)
        ))
        model.add(layers.Dropout(rate=dropout_rate))

    model.add(layers.Dense(units=1))
    return model
```

Essa abordagem permitiu explorar as capacidades de modelagem temporal de uma arquitetura recorrente básica. Espera-se que seu desempenho seja superior ao da rede densa também, e desempenho inferior a rede convolucional.

3.5.5 Modelo LSTM

Entre os modelos implementados, a rede LSTM (Long Short-Term Memory) foi escolhida por representar o estado da arte em tarefas de previsão de séries temporais. Esse tipo de rede é uma evolução das RNNs tradicionais, superando limitações como o problema do gradiente desvanecido por meio de uma arquitetura especializada em preservar informações de longo prazo com o uso de portas de entrada, esquecimento e saída.

Diversas arquiteturas foram testadas com o intuito de avaliar o impacto da profundidade e da quantidade de unidades por camada. As redes variaram desde estruturas simples, como uma única camada com 16 unidades, até modelos mais robustos com três camadas contendo 256 unidades cada. Foram também combinadas diferentes taxas de *dropout* (0,3, 0,4 e 0,5) e regularizações L2 (com taxas de 1×10^{-3} e 1×10^{-4}), com o objetivo de mitigar o *overfitting* sem comprometer a capacidade preditiva do modelo. A diversidade de configurações permitiu uma análise mais abrangente do desempenho do LSTM em cenários de complexidade variada, possibilitando observar com mais clareza os ganhos marginais de desempenho ao se incrementar a estrutura da rede.

Os hiperparâmetros gerais escolhidos para os treinamentos foram similares aos demais modelos utilizados, com tamanho de lote de 32 e um total de 500 épocas de treinamento. A função de ativação selecionada foi a tangente hiperbólica (`tanh`), que é apropriada para LSTMs por manter os valores centrados em torno de zero, facilitando o aprendizado ao longo do tempo. O otimizador adotado foi o Adam, uma escolha crítica para redes recorrentes por sua capacidade de lidar com gradientes instáveis, ajudando a contornar tanto o problema do gradiente desvanecido quanto o do gradiente explosivo. Outro argumento comum das redes recorrentes é o argumento `return_sequences`, o qual foi definido como verdadeiro em todas as camadas exceto na última, como nas RNN. Esse parâmetro controla se cada célula LSTM deve retornar a sequência completa de saídas (útil quando há múltiplas camadas) ou apenas a última saída (caso da última camada), sendo essencial para garantir que a informação temporal acumulada seja propagada corretamente ao longo da rede.

A função de construção do modelo, `make_lstm`, foi desenvolvida de maneira flexível para permitir a parametrização do número de camadas e unidades por camada, bem como das taxas de *dropout* recorrente, como indicado por (GAL; GHAHRAMANI, 2016) em vez do *dropout* comum, e regularização L2. A função percorre uma lista de unidades, adicionando uma camada LSTM para cada entrada da lista, com a devida configuração de `input_shape` na primeira camada e `return_sequences` ajustado de forma automática. Abaixo está o código da função utilizada:

Código 10 – Definição função de criação dos modelos de LSTM

```
import tensorflow as tf
from tensorflow.keras import layers, regularizers, Sequential

def make_lstm_model(units, recurrent_dropout_rate, l2_rate, input_shape)
    :

    model = Sequential()

    # Se units    int, transforma para lista
    if isinstance(units, int):
        units = [units]

    for i, n_units in enumerate(units):
        is_last_layer = (i == len(units) - 1)

        model.add(layers.LSTM(
            units=n_units,
            input_shape=input_shape if i == 0 else None,  # S   na
                primeira camada
            return_sequences=not is_last_layer,  # True se n o for a
                 ltima  camada
            kernel_regularizer=regularizers.l2(l2_rate) if l2_rate > 0
                else None,
            recurrent_dropout=recurrent_dropout_rate
        ))

    # Camada final densa (previs o)
    model.add(layers.Dense(units=1))  # Previs o de um  nico  valor (
        Close)

    return model
```

Com esse último modelo, considerado estado da arte para previsão de séries temporais, espera-se um resultado superior ao da RNN e modelo denso, e comparável com o das CNNs, como indica a literatura. Espera-se também que as redes apresentem maior sobreajuste conforme o número de épocas de treinamento aumenta, por conta de sua alta carga de parametrização, como já explicado na Seção 2.5.

4 DISCUSSÃO E RESULTADOS

Este capítulo apresenta o processo de seleção das arquiteturas dos modelos, bem como os respectivos gráficos de desempenho. O objetivo é viabilizar uma análise comparativa dos melhores modelos desenvolvidos e seus resultados na tarefa de previsão da série de contratos futuros de milho.

4.1 AMBIENTE COMPUTACIONAL

Os experimentos foram executados em um ambiente computacional composto por:

a) **Processador:**
   - CPU Intel Core i5;
   - 4 núcleos e 8 threads;
   - Frequência base de 2,4 GHz (máxima de 4,1 GHz);
   - Cache de 8 MB.

b) **Memória RAM:**
   - 16 GB (2x8 GB) em dual channel;
   - Frequência de 2667 MT/s, padrão SODIMM.

c) **Placa de vídeo:**
   - GPU NVIDIA GeForce GTX 1650;
   - 4 GB de memória dedicada.

d) **Software:**
   - Os experimentos foram desenvolvidos utilizando a linguagem de alto nível `Python`;
   - As bibliotecas `TensorFlow` e `Keras` foram utilizadas para construção e treinamento dos modelos de aprendizado profundo;
   - Também foram empregadas bibliotecas auxiliares como `NumPy`, `Pandas` e `Matplotlib` para manipulação de dados e visualização dos resultados.

Apesar da presença de GPU, os treinamentos foram realizados exclusivamente na CPU devido a incompatibilidades entre versões das bibliotecas utilizadas. A utilização da GPU, se viável, teria reduzido significativamente os tempos de execução. Cada modelo demandou, aproximadamente, oito horas de processamento.

4.2 RESULTADOS CONSOLIDADOS

A Tabela 1 apresenta os resultados consolidados dos modelos treinados, com os valores de *Mean Absolute Error* (MAE) médio e mediano obtidos sobre os dados de teste, considerando os preços de fechamento normalizados.

Tabela 1 – Resultados consolidados dos modelos treinados

Modelo	**MAE Médio**	**MAE Mediano**
Modelo Base	0,111	0,111
Rede Densa	0,541	0,487
CNN	0,318	0,257
RNN	0,135	0,112
LSTM	0,230	0,235

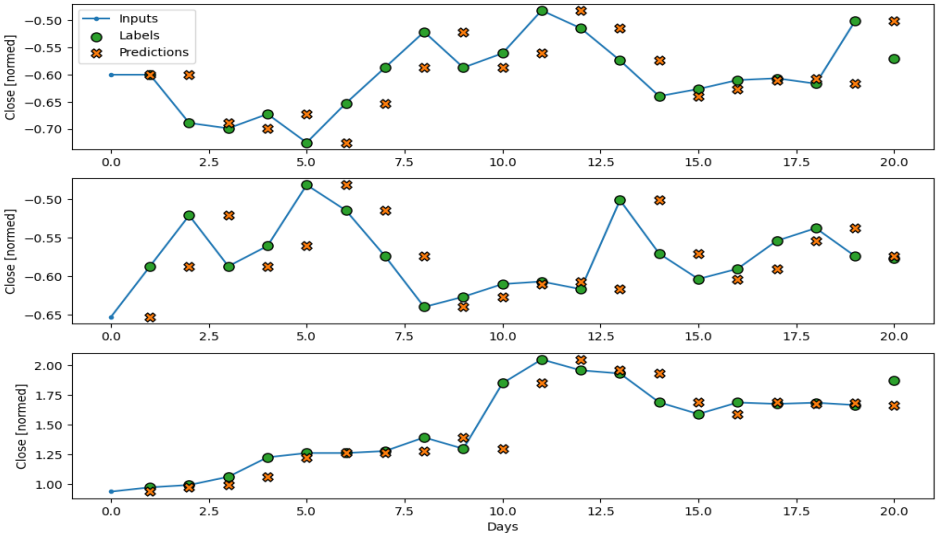
Fonte: Elaboração própria.

Os próximos tópicos apresentam a análise detalhada de cada modelo testado.

4.3 MODELO BASE

O modelo base segue uma heurística simples, assumindo que o preço de fechamento do contrato futuro no dia seguinte será igual ao preço do dia atual. A Figura 11 ilustra o comportamento desse modelo em três diferentes janelas de vinte dias.

Figura 11 – Representação dos resultados do modelo base.



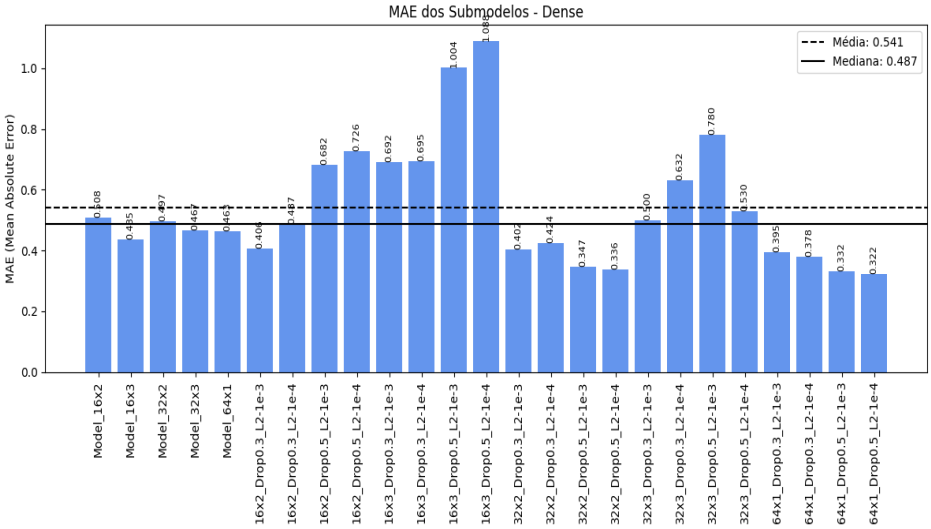
Fonte: Elaboração própria.

O desempenho do modelo base foi consideravelmente bom, com um MAE de 0,111, servindo como referência mínima para avaliar os modelos de aprendizado profundo propostos.

4.4 REDE DENSA

A Figura 12 apresenta o desempenho das arquiteturas densas testadas.

Figura 12 – Desempenho geral das arquiteturas densas treinadas até 500 épocas.

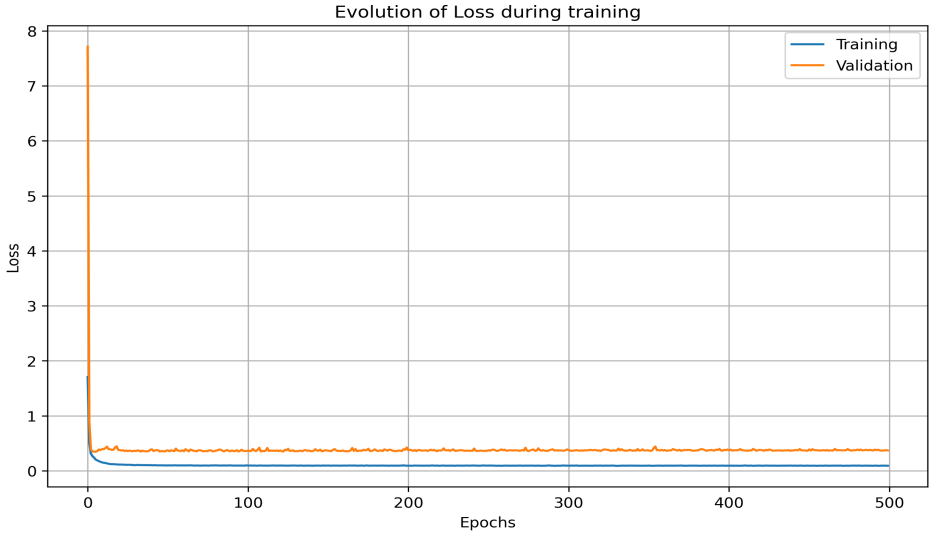


Fonte: Elaboração própria.

De modo geral, as variações na arquitetura da rede densa não proporcionaram melhorias significativas. As curvas de desempenho revelam uma rápida convergência nas primeiras épocas, seguida de estagnação e sobreajuste. Mesmo com o uso de técnicas de regularização, como *dropout* e penalização L2, os ganhos foram pouco expressivos, e em grande parte das redes que usaram dessas técnicas, o resultado da curva de validação ao longo das 500 épocas se provou muito mais volátil e menos preciso, por mais que o sobreajuste tenha sido, em partes, evitado.

A arquitetura escolhida foi composta por uma única camada densa com 64 neurônios, *dropout* de 0,5 e regularização L2 de 0,0001, treinada até 500 épocas. Essa configuração apresentou o melhor equilíbrio entre desempenho e estabilidade.

Figura 13 – Curva de validação da arquitetura escolhida para a rede densa.

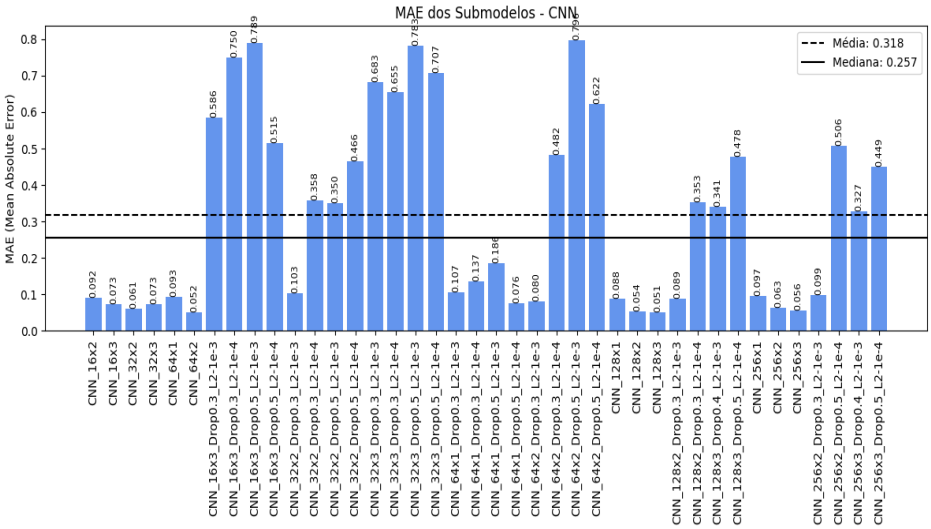


Fonte: Elaboração própria.

4.5 REDE CONVOLUCIONAL (CNN)

A Figura 14 apresenta os resultados das arquiteturas CNN testadas.

Figura 14 – Desempenho geral das arquiteturas CNN treinadas até 500 épocas.

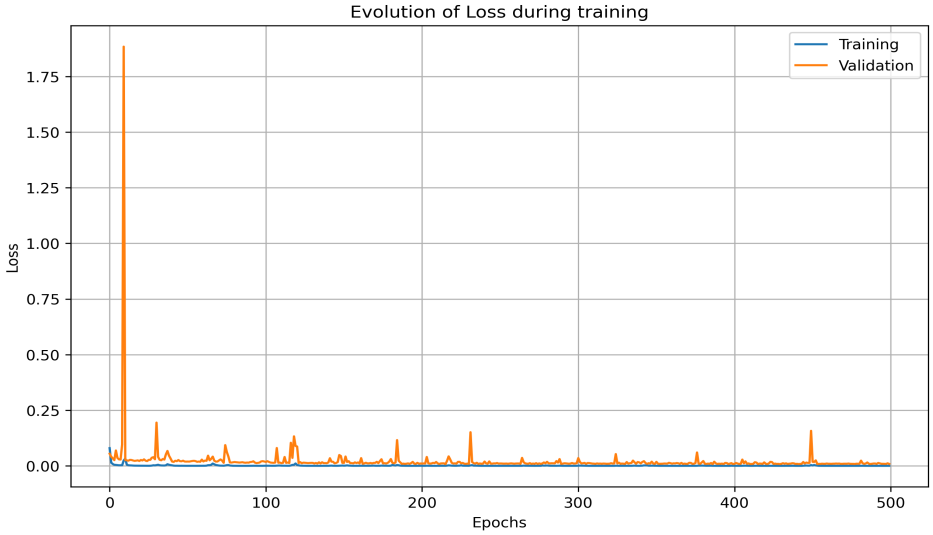


Fonte: Elaboração própria.

O modelo CNN obteve melhorias significativas em relação à rede densa. Curiosamente, os melhores desempenhos foram obtidos nas arquiteturas sem utilização de técnicas de *dropout* ou regularização L2, o que sugere menor sensibilidade ao sobreajuste neste contexto.

A arquitetura selecionada possui duas camadas convolucionais de 128 filtros cada, treinada até 500 épocas. Observou-se que, mesmo nesse limite, não houve sinais de sobreajuste, conforme a curva de validação apresentada na Figura 15.

Figura 15 – Curva de validação da arquitetura escolhida para a CNN.

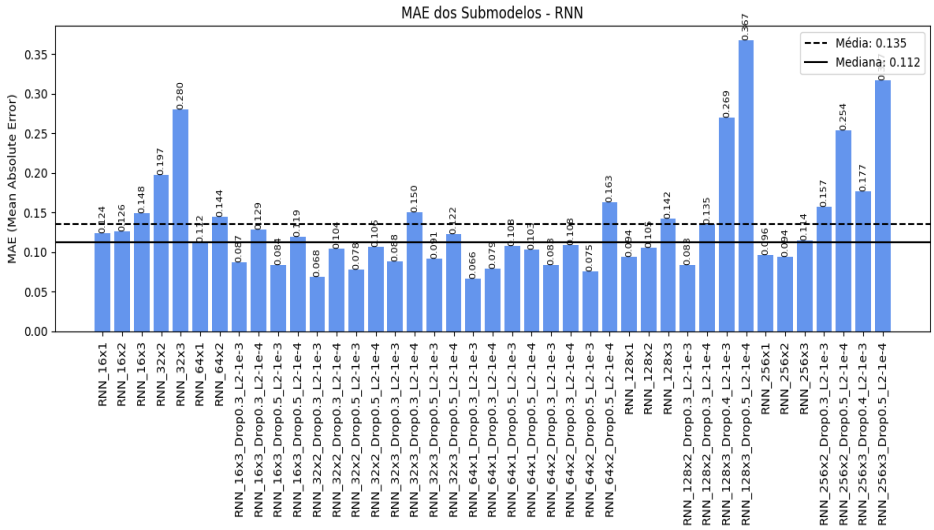


Fonte: Elaboração própria.

4.6 REDE RECORRENTE (RNN)

A Figura 16 apresenta os resultados da rede recorrente simples (RNN).

Figura 16 – Desempenho geral das arquiteturas RNN treinadas até 500 épocas.

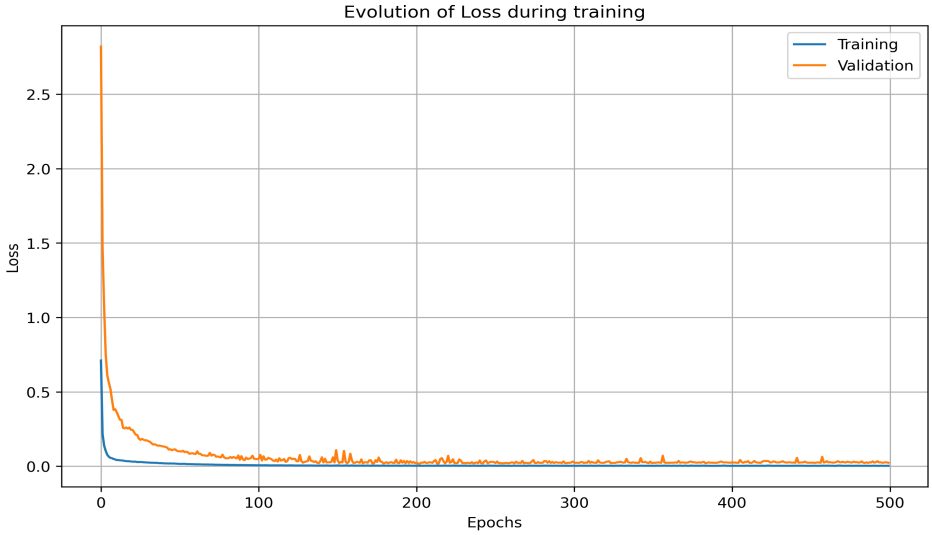


Fonte: Elaboração própria.

O modelo RNN demonstrou desempenho bastante consistente. Além de obter bons resultados quantitativos, observou-se uma redução significativa no ruído das curvas de validação quando comparado aos modelos anteriores.

A arquitetura selecionada possui uma única camada recorrente com 64 unidades, *dropout* recorrente de 0,3 e regularização L2 de 0,001, treinada até 500 épocas. A Figura 17 ilustra a suavidade e estabilidade das curvas de validação.

Figura 17 – Curva de validação da arquitetura escolhida para a RNN.

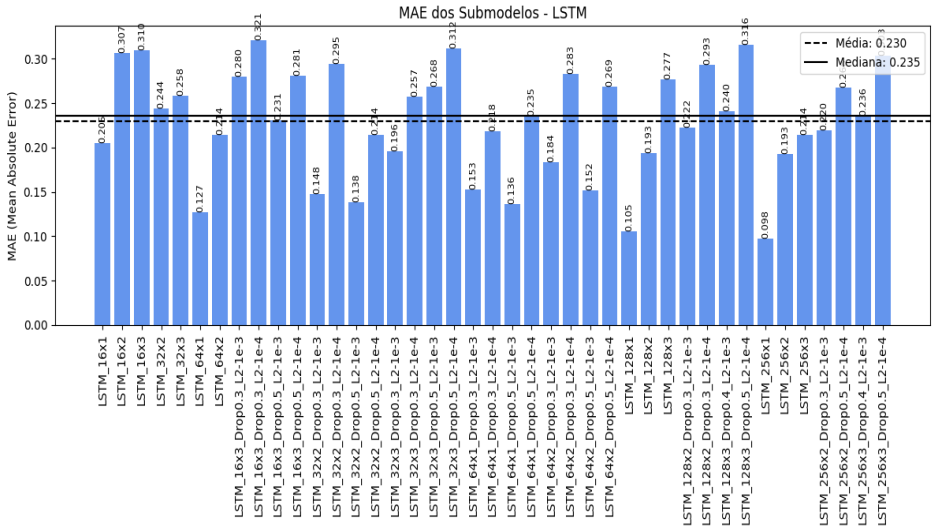


Fonte: Elaboração própria.

4.7 REDE LSTM

Os resultados da rede LSTM estão apresentados na Figura 18.

Figura 18 – Desempenho geral das arquiteturas LSTM treinadas até 500 épocas.



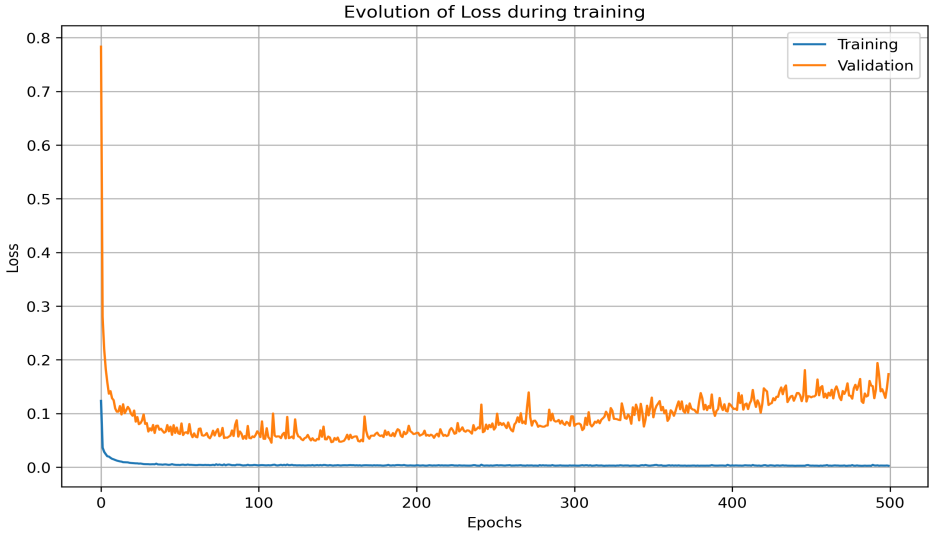
Fonte: Elaboração própria.

Apesar de sua maior capacidade teórica para capturar padrões de longo prazo, a LSTM apresentou desempenho inferior ao da RNN simples e da CNN. As curvas de validação evidenciaram forte sobreajuste após determinado ponto do treinamento, com aumento consistente da função de perda no conjunto de validação.

Esse comportamento era esperado, conforme discutido na Seção 2.6, dado que a LSTM possui um número significativamente maior de parâmetros, exigindo um volume de dados maior para seu pleno aproveitamento.

A arquitetura selecionada foi composta por uma única camada LSTM com 64 unidades, *dropout* de 0,5 e regularização L2 de 0,001. O treinamento foi interrompido na centésima época, momento em que se observou o menor valor da função de perda no conjunto de validação antes da manifestação evidente do sobreajuste, como pode ser visto na Figura 19.

Figura 19 – Curva de validação da arquitetura escolhida para a LSTM.



Fonte: Elaboração própria.

4.8 COMPARAÇÃO FINAL DOS MODELOS

A Tabela 2 resume os resultados finais de cada modelo, considerando o erro médio absoluto (MAE) e a função de perda (*Mean Squared Error* — MSE), além do ranking de desempenho com base no MAE.

Tabela 2 – Comparação dos resultados dos modelos finais

Modelo	**MAE Final**	**MSE Final**	**Ranking (MAE)**
Modelo Base	0,1141	0,0242	4º
Rede Densa	0,3224	0,1976	5º
CNN	**0,0541**	0,0078	**1º**
RNN	0,0664	0,0105	2º
LSTM	0,1106	0,0293	3º

Fonte: Elaboração própria.

Observa-se que a rede convolucional (CNN) obteve o melhor desempenho, seguida pela rede recorrente simples (RNN). O modelo base, apesar de sua simplicidade, superou tanto a rede densa quanto se igualou à LSTM nesse contexto. Esse resultado reforça a importância da escolha da arquitetura adequada para problemas de séries temporais, especialmente considerando limitações no volume de dados disponível.

5 CONCLUSÃO

Por fim, os resultados obtidos neste trabalho se mostraram, em grande parte, alinhados com as expectativas teóricas, embora algumas surpresas tenham surgido ao longo do processo:

- O modelo base apresentou resultados bastante satisfatórios, com um MAE de aproximadamente 0,1141, o que estabelece um referencial competitivo para os demais modelos.
- O modelo denso, devido à sua simplicidade arquitetural, apresentou o pior desempenho entre todos os modelos testados, com um MAE de 0,3257, desempenho inferior até mesmo ao modelo base.
- O modelo convolucional apresentou o melhor desempenho geral, alcançando um MAE de 0,0541. Esse resultado já era, em certa medida, esperado, com base em trabalhos anteriores, e demonstra uma superioridade substancial em relação ao modelo base.
- O modelo recorrente simples (RNN) obteve o segundo melhor desempenho, com MAE de 0,0664, também superando de forma consistente o modelo base. Esse resultado era esperado, dado que modelos recorrentes são adequados para capturar dependências temporais nas séries.
- Por outro lado, o modelo LSTM apresentou resultados abaixo do esperado. Embora se supusesse que seu desempenho superasse tanto a RNN simples quanto se aproximasse da CNN, isso não se confirmou. Com um MAE de 0,1106, a LSTM apenas marginalmente supera o modelo base e apresenta desempenho consideravelmente inferior às arquiteturas CNN e RNN. Esse resultado é interpretado como consequência da maior complexidade paramétrica da LSTM, que exige um volume de dados substancialmente maior para aproveitar todo seu potencial na modelagem de padrões temporais longos e curtos. No presente cenário, a maior complexidade da LSTM se traduziu em pior capacidade de generalização.

Diante desses resultados, considera-se que o presente trabalho foi bem-sucedido na comparação de diferentes arquiteturas de modelos preditivos aplicados à série temporal dos contratos futuros de milho. Utilizando atributos simples, obtidos diretamente da CBOT, combinados com uma engenharia de atributos básica, foi possível demonstrar que modelos modernos de aprendizado profundo, especialmente as redes convolucionais e recorrentes simples, são capazes de superar um modelo base robusto, estabelecendo uma margem clara de superioridade em termos de acurácia.

Por outro lado, o desempenho inferior tanto da rede densa quanto da LSTM reforça a percepção de que a predição de séries temporais financeiras, particularmente com dados limitados e de alta variabilidade, permanece como um desafio significativo no campo de aprendizado de máquina. A escolha adequada da arquitetura se mostra, portanto, um fator determinante para o sucesso na modelagem desse tipo de problema.

Como continuidade deste trabalho, sugere-se a exploração de uma busca mais ampla de hiperparâmetros, incluindo ajustes no tamanho dos lotes de treinamento, taxa de aprendizado, número de épocas, uso de diferentes funções de ativação e otimizadores.

Além disso, o enriquecimento dos dados com variáveis macroeconômicas e informações de safra de milho pode contribuir significativamente para uma representação mais robusta da série, potencializando a capacidade preditiva dos modelos, especialmente das arquiteturas mais complexas, como as redes LSTM.

Ademais, o espaço amostral de séries temporais financeiras é amplo e diferentes ativos e séries financeiras poderiam ter seus números previstos e comparados por métodos de aprendizado profundo também.

REFERÊNCIAS

BENGIO, Y.; SIMARD, P.; FRASCONI, P. Learning long-term dependencies with gradient descent is difficult. **IEEE transactions on neural networks**, IEEE, v. 5, n. 2, p. 157–166, 1994.

BOLLEN, J.; MAO, H.; ZENG, X. Twitter mood predicts the stock market. **Journal of Computational Science**, v. 2, n. 1, p. 1–8, 2011.

BROWNLEE, J. **Deep Learning for Time Series Forecasting: Predict the Future with MLPs, CNNs and LSTMs in Python**. Machine Learning Mastery, 2018. Disponível em: https://machinelearningmastery.com/deep-learning-for-time-series-forecasting/.

CHECKLEY, M.; HIGóN, D. A.; ALLES, M. The predictive power of 'bullish' and 'bearish' sentiment for market volatility. **International Review of Financial Analysis**, v. 50, p. 39–49, 2017.

CHEN, M.-Y.; LIAO, C. H.; HSIEH, R.-P. Modeling public mood and emotion: Stock market trend prediction with anticipatory computing approach. **Computers in Human Behavior**, v. 101, 03 2019.

CHOLLET, F. **Deep Learning with Python**. Second edition. Manning Publications, 2021. Disponível em: https://www.manning.com/books/deep-learning-with-python-second-edition.

GAL, Y.; GHAHRAMANI, Z. A theoretically grounded application of dropout in recurrent neural networks. In: **Advances in neural information processing systems**. [s.n.], 2016. v. 29, p. 1019–1027. Disponível em: https://arxiv.org/abs/1512.05287.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016.

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

HU, Z.; ZHAO, Y.; KHUSHI, M. Listening to chaotic whispers: A deep learning framework for news-oriented stock trend prediction. **arXiv preprint arXiv:1803.00111**, 2018.

JEONG, H.-C.; LEE, K.-A.; CHO, S.-B. News-driven stock price prediction using attention-based multi-task learning. **arXiv preprint arXiv:1811.10107**, 2018.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2012. v. 25, p. 1097–1105.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **Nature**, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015.

LI, X. et al. News impact on stock price return via sentiment analysis. In: **Proceedings of the 2017 International Conference on Big Data Technologies**. [S.l.: s.n.], 2017. p. 1–6.

LITJENS, G. et al. A survey on deep learning in medical image analysis. **Medical image analysis**, Elsevier, v. 42, p. 60–88, 2017.

MEHTAB, M.; SEN, J. Price trend prediction of bitcoin using deep learning models. **Procedia Computer Science**, v. 167, p. 1228–1238, 2020.

MEHTAB, M.; SEN, J. A comparative analysis of lstm and cnn for financial time series prediction. **Journal of Emerging Technologies and Innovative Research**, v. 8, n. 8, p. 182–193, 2021.

MEHTAB, M.; SEN, J. Deep learning-based financial market prediction: A comparative study of lstm and cnn. **Journal of Investment and Management**, v. 10, n. 1, p. 1–12, 2021.

MEHTAB, M.; SEN, J. Predicting stock price movement using lstm with technical indicators. **International Journal of Business Forecasting and Marketing Intelligence**, v. 6, n. 3, p. 263–281, 2021.

MEHTAB, M.; SEN, J. A robust predictive model for stock price prediction using deep learning and technical analysis. **IEEE Access**, v. 9, p. 31786–31802, 2021.

MEHTAB, M.; SEN, J. A novel hybrid deep learning model for stock market prediction based on technical analysis and sentiment analysis. **International Journal of Business Forecasting and Marketing Intelligence**, v. 7, n. 1, p. 1–22, 2022.

NGAI, E. W. et al. The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature. **Decision support systems**, Elsevier, v. 50, n. 3, p. 559–569, 2011.

PEROL, T.; GHARBI, M.; DENOLLE, M. Convolutional neural network for earthquake detection and location. **Science Advances**, American Association for the Advancement of Science, v. 4, n. 2, p. e1700578, 2018.

SCHUMAKER, R. P.; CHEN, H. Textual analysis of stock market prediction using breaking financial news: The azfin text system. **ACM Transactions on Information Systems (TOIS)**, v. 27, n. 2, p. 12, 2009.

VASWANI, A. et al. Attention is all you need. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2017. v. 30, p. 5998–6008.