Inteligência artificial aplicada em séries temporais

Daniel Walter de Paula Martins

Dissertação apresentada ao
Instituto de Matemática e Estatística
da Universidade de São Paulo
para obtenção do título de
Mestre em Estatística

Programa: Probabilidade e Estatística
Orientador: Prof. Dr. Anatoli Iambartsev

São Paulo
Janeiro de 2024

Inteligência artificial aplicada em séries temporais

Daniel Walter de Paula Martins

Esta é a versão original da dissertação elaborada pelo candidato Daniel Walter de Paula Martins, tal como submetida à Comissão Julgadora.

Á Deus e minha família, fonte inesgotável de amor e apoio,
e ao meu mentor, cuja orientação sábia moldou cada página
desta jornada acadêmica. Este trabalho é dedicado a vocês,
os alicerces que sustentaram meus sonhos e inspirações.

Agradecimentos

Gostaria de expressar minha profunda gratidão neste momento especial, marcado pela minha dissertação de mestrado. Em primeiro lugar, agradeço a Deus pela orientação, força e sabedoria concedidas ao longo dessa jornada acadêmica.

Á minha esposa, Evelyn Martins, meu alicerce e manancial inesgotável de apoio, meu mais sincero agradecimento. Sua compreensão, paciência e incentivo foram fundamentais para que eu pudesse dedicar-me plenamente a este desafio.

Aos meus queridos filhos, Davi e Eloá, que foram minha inspiração diária nesta reta final, agradeço por compreenderem os momentos em que estive dedicando-me aos estudos. O amor e presença constante de vocês foram meu combustível para superar os obstáculos.

A todos que contribuíram de alguma forma para este percurso, meu sincero obrigado. Este marco representa esforço, mas também o apoio inestimável daqueles que estiveram ao meu lado durante toda essa jornada de aprendizado.

Resumo

Deep Learning (redes neurais profundas, tradução literal), área também conhecida como 'Inteligência Artificial' é um campo que tem crescido consideravelmente nos últimos anos por intermédio de aplicações práticas, que podem substituir processos com diferentes complexidades. Com avanço da tecnologia tornou-se possível o processamento de grandes quantidades de informação (Big Data), fato que trouxe notoriedade para essa classe de algoritmos com maior ênfase em dados "não-tabulares" (imagem, vídeo e texto).

O problema de classificação de imagens foi um dos grandes propulsores dos modelos de Deep Learning, a técnica é um vasto campo de estudo e vêm sendo aprimorada constantemente. Muitas estruturas de Deep Learning têm sido adaptadas para problemas específicos e há uma variedade considerável de possibilidades. De modo geral, a estrutura desses algoritmos é extremamente flexível que permitem a modelagem de acordo com os interesses de aplicação e análise.

Embora as estruturas de Deep Learning tenham alcançado resultados consideráveis no campo de imagem, vídeo e texto o intuito desta dissertação é justamente aplicar essa metodologia para dados tabulares, a saber, séries temporais. Com destaque para uma classe específica de Deep Learning, que considera em sua estrutura mecanismos que permitem o fluxo de dependência temporal nos dados, o que possibilita a realização de previsões de valores futuros.

Para realizar esta aplicação é necessário compreender todo o contexto dos algoritmos de Deep Learning que têm seu próprio vocabulário, notações e definições. Para que os dados de séries temporais sejam compatíveis com as estruturas propostas é necessário preparação específica, que pode requerer diferente abordagem de acordo com a classe de algoritmo proposto. Diferentes estruturas são consideradas na aplicação para séries temporais, com destaque para os algoritmos "Long Short Term Memory" (LSTM), caso específico de Redes Neurais Recorrentes (RNR).

A estimação destes modelos pode ser interpretada como uma questão de otimização que envolve não somente parâmetros, mas também Hiperparâmetros, que controlam toda a estrutura dos algoritmos. Ao considerar o cenário de séries temporais a otimização ocorre ao longo do tempo, propagando características dos dados em diferentes momentos. Modelos da classe ARIMA são utilizados como base da aplicação para auxiliar na efetividade das diferentes estruturas de Deep Learning para os diversos cenários considerados.

A aplicação de Deep learning para séries temporais é uma área que têm ganhado destaque no cenário acadêmico dada a popularidade destes algoritmos em competições internacionais e os respectivos resultados obtidos Smyl e Shanmugam (2017). A abordagem destes algoritmos é consideravelmente diferente dos modelos clássicos de séries temporais, ao passo que as principais diferenças são propulsoras de pesquisas e exploração acadêmica.

Palavras-chave: Séries Temporais. Memória de curto e longo Prazo. Redes Neurais Recorrentes . Aprendizado Profundo de Redes Neurais. Inteligência Artificial .

Abstract

Deep Learning (also known as artificial Intelligence), is a field that has grown significantly in recent years through practical applications capable of replacing processes with varying complexities. With technological advancements, the processing of large amounts of information (Big Data) has become possible, bringing prominence to this class of algorithms, with a greater emphasis on "non-tabular" data such as images, videos, and text.

The problem of image classification has been a major driving force behind Deep Learning models, and the technique is a vast field of study continually undergoing refinement. Many Deep Learning structures have been adapted for specific problems, offering a considerable variety of possibilities. In general, the flexibility of these algorithms allows for modeling according to the application and analysis interests.

While Deep Learning structures have achieved notable results in the fields of image, video, and text, the purpose of this dissertation is to specifically apply this methodology to tabular data, namely, time series. This includes a focus on a specific class of Deep Learning that incorporates mechanisms in its structure enabling the flow of temporal dependencies in the data, facilitating predictions of future values.

To implement this application, a comprehensive understanding of the context of Deep Learning algorithms is necessary, each with its own vocabulary, notations, and definitions. To make time series data compatible with the proposed structures, specific preparation is required, potentially requiring different approaches depending on the proposed algorithm class. Various structures are considered in the application for time series, with emphasis on algorithms like "Long Short Term Memory" (LSTM), a specific case of Recurrent Neural Networks (RNN).

Estimating these models can be interpreted as an optimization issue involving not only parameters but also hyperparameters that control the entire structure of the algorithms. In the context of time series, optimization occurs over time, propagating data characteristics at different moments. ARIMA models serve as a basis for the application, aiding the effectiveness of different Deep Learning structures for various considered scenarios.

The application of Deep Learning to time series has gained prominence in the academic landscape due to the popularity of these algorithms in international competitions and the respective results achieved Smyl and Shanmugam (2017) .The approach of these algorithms is considerably different from classical time series models, and these key differences drive further research and academic exploration.

Keywords: Time Series. Long Short Term Memory. Recurent Neural Network . Deep learning . Artificial Intelligence .

Lista de Abreviaturas

LSTM	Memória de Curto e Longo Prazo (*Long Short Term Memory*)
RNN	Redes Neurais Recorrentes(*Recurrent Neural Networks*)
CNN	Rede Neural de Convolução (*Convolution Neural Network*)
ARIMA	Média Móvel Integrada Auto Regressiva
BPTT	Retropropagação ao Longo do Tempo (*Backpropagation through time*)
RNP	Redes Neurais Profundas
SARIMA	Média móvel Integrada Auto-Regressiva Sazonal
AR	Auto-Regressiva (*Auto Regressive*)
MA	Média Móvel (*Moving Average*)
RNA	Redes Neurais Artificiais
GD	Gradiente Descendente (*Gradient Descent*)
OCR	Reconhecimento Óptico de Caracteres (*Optical Character Recognition*)
ConvLSTM	Memória de Longo Prazo de Convolução (*Convolution Long Short Term Memory*)
GRU	Unidade Recorrente Fechada (*Gated Recurrent Unit*)
PLN	Processamento de Linguagem Natural
PNL	Programação Neurolinguística
BERT	Representações de Codificadores Bidirecionais de Transformadores
GPT	Transformador de Pré-Treinamento Generativo (*Generative Pre-Traing Transformer*)
RMSprop	Propagação Quadrática Média (*Root Mean Square Propagation*)
SGD	Descida Gradiente Estocática (*Stochatic Gradient Descent*)
RMSE	Erro Quadrático Médio (*Root Mean Square Error*)
MAE	Erro Absoluto Médio (*Mean Absolut Error*)
MAPE	Erro Médio de Porcentagem Absoluta (*Mean Absolut Percetage Error*)
VAR	Vetor Auto Regressivo (*Vector Auto Regressive*)
GPU	Unidade de Processamento Gráfico (*Graphics Processing Unit*)

Lista de Figuras

Lista de Tabelas

Lista de Programas

Sumário

Capítulo 1

Introdução

1.1 Motivação e Contexto

A motivação deste trabalho é explorar as estruturas de Deep Leaning adaptadas para o contexto de séries temporais. Estas estruturas têm obtido desempenhos consideráveis em diversas aplicações para dados "não-tabulares" (imagem, vídeo e texto). O contexto de séries temporais, normalmente, tem quantidade de dados menor quando comparado com dados de imagens Russakovsky (2014), ImageNet, por exemplo. É de interesse avaliar o comportamento das estruturas de Deep Learning no cenário em que a quantidade de dados é limitada e a grande quantidade de parâmetros pode resultar em problemas de estimação e overfiting (sobreajuste).

A proposta principal é trabalhar com a classe de Long Short Term Memory (LSTM) de algoritmos. Esta classe permite que informações sejam consideradas ao longo do tempo e considerando boa aderência em sequencias, sejam numéricas ou textuais. Variações dessa classe de algoritmos também são consideradas, assim como combinações com diferentes estruturas de Deep Learning, que se adequam a dependência temporal.

A flexibilidade dos algoritmos torna possível a construção de inúmeras estruturas de Deep Learning capazes de endereçar a mesma análise ou aplicação. Na literatura atual não há de forma concreta qual estrutura é mais recomendável dado cenário específico de aplicação Jozefowicz *et al.* (2015).Neste contexto será avaliado empiricamente diferentes estruturas e suas respectivas aderências no processo de otimização, dada uma função de perda. Os parâmetros e Hiperparâmetros serão considerados e avaliados por intermédio do processo de otimização e Grid Search, respectivamente.

Como baseline, algoritmos da classe ARIMA são considerados para avaliar o desempenho das diferentes estruturas de Deep Learning. O estímulo que será averiguado diz respeito ao comportamento dessas estruturas frente as classes clássicas de algoritmos de séries temporais com relação a diferentes cenários de aplicação para previsões pontuais.

Este texto, inclusive, tem também como objetivo explorar o caminho de aplicações de Deep Learning para séries temporais ao descrever conceitos, realizar comparações, aplicações e elencar campos de desdobramento e estudo. A abordagem de Redes Neurais

Profundas (RNP) aplicada em séries temporais frente aos modelos clássicos (ARIMA por exemplo) são distintas e suas especificidades representam tema de comparação, assim como mapeamento de áreas de exploração no contexto teórico de RNP associadas a dados tabulares e temas relacionados.

1.2 Organização do Trabalho

A dissertação está organizada em 5 capítulos, sendo o primeiro uma breve introdução e descrição dos principais objetivos elencados que serão abordados nos demais capítulos.

No segundo Capítulo é realizada uma descrição dos modelos ARIMA, denominados modelos clássicos de séries temporais, que serão considerados como baseline frente aos cenários de aplicação e comparação das abordagens de Deep Learning.

O terceiro capítulo detalha o tema de Deep Learning seus conceitos e abordagens. O funcionamento de algumas classes de Redes Neurais Profundas (RNP) são descritas no contexto de aplicação para cenários de sequências numéricas. A classe de Redes Neurais Recorrentes LSTM tem protagonismo por seus mecanismos serem de fundamental importância para compreensão da aplicação das Estruturas de RNPs para séries temporais.

O quarto capítulo tem enfoque na aplicação prática das estruturas de Deep Learning em séries temporais considerando diferentes cenários de aplicação. Para que seja possível a aplicação das RNPs é preciso adaptar os dados para um processo supervisionado, procedimento conhecido como Sliding Window. Com intuito de averiguar empiricamente certos comportamentos diferentes conjuntos de dados foram considerados no momento da aplicação, assim como algumas premissas relacionadas frente ao impacto na performance dos algoritmos propostos comparado aos algoritmos convencionais (classe ARIMA é utilizada como baseline).

O quarto capítulo também adota uma etapa de simulação que tem por objetivo analisar o comportamento dos parâmetros(pesos) dos algoritmos de LSTM aplicados a sequencias numéricas com e sem estrutura de dependência entre os respectivos elementos de sua composição.

O quinto Capítulo aborda as diferenças entre a metodologia de Deep Learnig aplicada para séries temporais frente a metodologia de Box e Jenkins (1970) descrita no segundo capítulo. As conclusões frente aos diferentes algoritmos e cenários de aplicação também são analisadas, assim como áreas de possível exploração e desenvolvimento acadêmico no campo de Deep Learning e séries temporais.

Capítulo 2

Séries Temporais

2.1 Conceitos Básicos de Séries Temporais

Uma série temporal pode ser caracterizada como uma coleção de observação de uma variável aleatória, dispostas de maneira sequencial e ordenada em uma determinada unidade de tempo (semana, mês, ano etc.). Tais observações, geralmente, estão em intervalos temporais de mesmo tamanho, isto é, equiespaçados. A Decomposição Clássica de uma série temporal Corrar e Théophilo (2004) consiste em quatro componentes principais, a partir dos quais, são feitas as previsões:

- **Tendência**: que se refere à direção geral segundo a qual o gráfico da série temporal se desenvolve em um intervalo de tempo.
- **Ciclo**: flutuações de longo prazo que não é necessariamente regular ou previsível. Ele representa variações que ocorrem em períodos mais longos do que os da sazonalidade e pode ser causado por fatores econômicos, demográficos ou outras influências de longo prazo.
- **Sazonalidade**:padrões regulares e previsíveis que se repetem em intervalos fixos e curtos de tempo, geralmente dentro de um ano. Esses padrões estão relacionados a fatores sazonais, como estações do ano, feriados, períodos de férias ou eventos recorrentes.
- **Termo Aleatório**: que aparece com flutuações de curto período, com deslocamento inexplicável (Makridakis & Wheelwright, Forecasting: Issues & Challenges for Marketing Management, 1989).

De modo teórico uma série temporal pode ser a representação de uma variável aleatória Y no instante de tempo t, a série temporal pode ser escrita por $Y_1, Y_2, ..., Y_N$, sendo N é o número de observações registradas pela série, o qual referido também como o tamanho da série Camargo e Souza (1996).As séries temporais podem ser classificadas como discretas, contínuas, determinísticas, estocásticas, multivariadas e multidimensionais.

Os modelos convencionais utilizados para descrever séries temporais são processos estocásticos, isto é, processos controlados por leis probabilísticas. Seja T um conjunto

arbitrário um processo estocástico é uma família $Z = \{Z(t), t \in T\}$, tal que, para cada $t \in T, Z(t)$ é uma variável aleatória. Nestas condições, um processo estocástico é uma família de variáveis aleatórias, que são consideradas definidas em um mesmo espaço de probabilidades Morettin e Toloi (2006).

Uma série temporal pode ser interpretada como uma realização de um processo estocástico sendo assim, uma amostra finita de observações em uma linha de tempo. O processo é estatisticamente determinado quando todas as suas funções de distribuição de probabilidade são conhecidas até a N-ésima ordem. Porém, o processo é desconhecido até a N-ésima ordem e, portanto, assume somente uma realização do processo estocástico, uma vez que o mecanismo gerador de dados do processo não é conhecido. Para contornar estes problemas, a teoria clássica de séries temporais assume duas condições: Ergodicidade e Estacionariedade.

No processo estocástico estacionário algumas características permanecem constantes ao longo do tempo. Podemos analisar a estacionariedade sendo do tipo forte (estrita) ou fraca (segunda ordem). O conceito de processo não estacionário homogêneo se dá, quando se consegue um processo estacionário através de diferenças sucessivas de um determinado processo não estacionário.

Um processo estocástico é dito ergódico se uma única realização do processo é o suficiente para caracterizá-lo. Na análise de séries temporais existe apenas uma realização do processo disponível e, portanto, é preciso supor que o processo subjacente é ergódico, pois será utilizada apenas uma de suas realizações para caracterizá-lo. Note que dada uma realidade (processo estocástico) retira-se uma amostra finita de observações equiespaçadas no tempo (série temporal) e através do estudo desta amostra (análise de séries temporais) identifica-se um modelo cujo objetivo é inferir sobre o comportamento da realidade, processo estocástico Camargo e Souza (1996).

As séries temporais descrevem o fenômeno gerador do processo ao longo do tempo, ou seja, as características do mecanismo gerador são descritas através da ordenação das observações no tempo. Deste modo, há dependência dos dados está diluída na sequência numérica ao longo do tempo em que o processo de modelagem visa justamente identificar os padrões no período de análise. Estes conceitos essenciais são considerados nos demais capítulos, principalmente na aplicação das estruturas de Deep Learning em Séries temporais (Capítulo 4).

A sessão seguinte descreve uma classe de modelos clássica e bastante conhecida na literatura de séries temporais, a classe de modelos ARIMA (Autoregressive Integrated and Moving Average), que são utilizados como baseline na comparação de desempenho frente as aplicações de Deep Learning no contexto de dados tabulares no quarto capítulo.

2.2 Modelos Arima

Metodologia consideravelmente utilizada em análise de modelos paramétricos, conhecida como abordagem de Box e Jenkins (1970). De forma concisa, esta metodologia ajusta modelos autorregressivos integrados de médias móveis, ARIMA (p, d, q), a um conjunto de dados. A construção dos modelos ARIMA (p, d, q) baseia-se em um ciclo interativo no

qual a escolha da estrutura do modelo é baseada nos próprios dados. Essa construção é dependente da experiência do responsável pela elaboração do algoritmo estatístico. Os estágios do ciclo interativo são:

- Especificação de uma classe geral de modelos.
- Identificação de um modelo, com base na análise dos gráficos de autocorrelação(ACF) e autocorrelação Parcial (PACF).
- Estimação do modelo, na qual os parâmetros do modelo identificado são estimados.
- Por último, diagnóstico do modelo ajustado, por meio da análise de resíduo; se o modelo não for adequado o processo deve retornar para a fase de identificação (veja, Figura 2.1).

$$Y_t = c + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + ... + \phi_p Y_{t-p} - \theta_1 \epsilon_{t-1} - \theta_2 \epsilon_{t-2} - ... - \theta_q \epsilon_{t-q} + \epsilon_t$$

Onde:

- Y_t é o valor da série temporal no tempo t.
- c é a constante do modelo.
- $\phi_1, \phi_2, ..., \phi_p$ são os parâmetros de autorregressão.
- ϵ_t é o termo de erro branco (erro de previsão) no tempo t.
- $\theta_1, \theta_2, ..., \theta_q$ são os parâmetros de média móvel.

Figura 2.1: *Modelo ARIMA(p,d,q)*

O modelo proposto por Box e Jenkins pode ser interpretado de forma particionada ao considerar os três elementos que representam a interpretação do nome da metodologia. A parte auto-regressiva (AR) do modelo ARIMA indica que a variável de interesse é regressada em seus próprios valores defasados, isto é, anteriores. A parte de média móvel (MA) indica que o erro de regressão é na verdade uma combinação linear dos termos de erro, cujos valores ocorreram contemporaneamente e em vários momentos no passado. A parte integrada (I) indica que os valores de dados foram substituídos com a diferença entre seus valores e os valores anteriores e este processo diferenciador pode ter sido realizado mais de uma vez. O propósito de cada uma destas características é fazer o modelo se ajustar aos dados da melhor forma possível.

Modelos ARIMA não sazonais são geralmente denotados como ARIMA(p,d,q), em que os parâmetros p, d e q são números inteiros não negativos, p é a ordem (número de defasagens) do modelo auto-regressivo, d é o grau de diferenciação (o número de vezes em que os dados tiveram valores passados subtraídos para se tornarem estacionários) e q é a ordem do modelo de média móvel. Modelos ARIMA sazonais (conhecidos como SARIMA) são geralmente denotados como ARIMA(p,d,q)(P,D,Q)s, em que s se refere ao número de períodos em cada temporada sazonal e P, D e Q se referem aos termos de auto-regressão, diferenciação e média móvel para a parte sazonal do modelo.

Os modelos da classe ARIMA são considerados, neste texto, como modelos base(baseline) na aplicação das estruturas de Deep Learning para séries temporais.

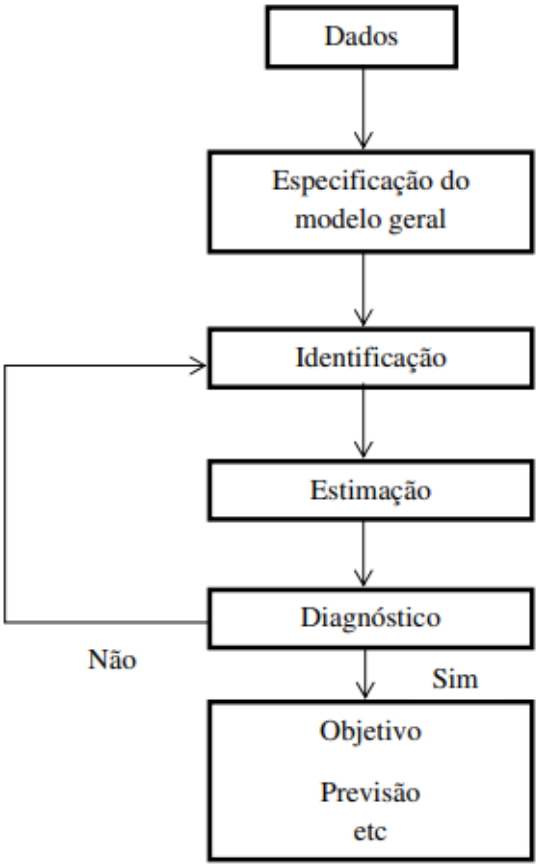


Figura 2.2: *Ciclo de desenvolvimento da metodologia Box e Jenkins*

Algumas das premissas adotadas para os modelos clássicos são averiguadas (Estacionariedade e distribuição dos resíduos por exemplo) ao decorrer da aplicação para que sua necessidade seja evidenciada ou desconsiderada, uma vez que não há premissas explicitas frente as estruturas de Deep Learning.

2.3 Modelos Vector Autoregressive(VAR)

Os modelos Vector Autoregressive (VAR) representam uma abordagem fundamental na análise de séries temporais multivariadas. Os modelos VAR são utilizados para capturar as relações lineares entre múltiplas séries temporais. Sua aplicação estende-se por diversos campos, como economia, finanças, meteorologia e qualquer domínio onde o comportamento dinâmico de sistemas inter-relacionados é de interesse.

A essência dos modelos VAR é descrever cada variável do sistema como uma combinação linear das suas próprias defasagens passadas (lags) e das defasagens das outras variáveis do sistema. Formalmente, um modelo $VAR(p)$ para um sistema de k variáveis, veja Figura 2.3.

Esses termos de erro são tipicamente assumidos como ruídos brancos multivariados, ou seja, são serialmente não correlacionados e cada um tem uma matriz de covariância

$$Y_t = A_1 Y_{t-1} + A_2 Y_{t-2} + \cdots + A_p Y_{t-p} + \mu + \varepsilon_t$$

onde:

- Y_t é um vetor $k \times 1$ de variáveis endógenas no tempo t,
- $A_1, A_2, \ldots, A_p$ são matrizes $k \times k$ de coeficientes a serem estimados,
- p é a ordem do modelo, isto é, o número de defasagens consideradas,
- μ é um vetor de constantes (interceptos) e
- ε_t é um vetor $k \times 1$ de termos de erro.

Figura 2.3: *Modelo VAR(p)*

constante.A estimação dos parâmetros de um modelo VAR é geralmente realizada pelo método dos mínimos quadrados ordinários (MQO) para cada equação. Uma etapa crucial na modelagem VAR é a seleção da ordem p, que pode ser determinada por critérios de informação, como o Critério de Informação de Akaike (AIC) ou o Critério de Informação Bayesiano (BIC).

Os modelos VAR permitem uma abordagem dinâmica para entender como as variáveis respondem ao longo do tempo a choques em outras variáveis. Isso é frequentemente explorado através de funções de impulso-resposta e decomposições da variância do erro de previsão, fornecendo insights sobre a transmissão e persistência dos efeitos ao longo do tempo.

Apesar de sua utilidade, os modelos VAR têm limitações, podem não ser adequados quando a série temporal é muito longa, devido à complexidade crescente e ao potencial de superajustamento. Além disso, os modelos VAR padrão são lineares e podem não capturar adequadamente a dinâmica não linear presente em muitos sistemas econômicos e financeiros.

Os modelos VAR são considerados como baseline frente as aplicações de Deep Learning para os conjuntos de dados multivariados do Capitulo 4. Essa classe de modelo é considerada a generalização dos modelos Auto-Regressivos da classe ARIMA e por este motivo são considerados como comparação para os casos de multiplas séries temporais.

2.4 Abordagem Tradiciona x Modelos de Deep Learning

Os modelos da classe ARIMA têm suas premissas estabelecidas e toda uma teoria comprovada que sustentam teoricamente seus resultados e aplicações. Neste contexto de aplicação de estruturas de Deep Learning também é de interesse avaliar a possibilidade de desconsiderar premissas convencionais requeridas nos modelos da classe ARIMA, com intuito de avaliar o respectivo desempenho e impacto dessas premissas nas aplicações propostas.

Importante destacar que os modelos de Deep Learning que permitem a modelagem de sequencias numéricas (LSTM, por exemplo), não impõem premissas explicitas aos dados. Desta forma, inicialmente, há maior flexibilidade nas aplicações, no entanto faz-se necessário comparar o quão efetivo estes modelos são em diferentes cenários e aplicação.

Devido às notáveis diferenças entre as metodologias na abordagem de séries temporais, especialmente no que diz respeito à obtenção de previsões pontuais, os modelos de Deep Learning apresentam maior flexibilidade em cenários de aplicação, desde que não seja necessário levar em consideração as premissas tradicionais da metodologia de Box e Jenkins (1970). Em outras palavras, as premissas clássicas podem ser desconsideradas, a menos que tenham um impacto significativo nos resultados em comparação com os modelos baseline da classe ARIMA.

De forma alternativa, caso haja impacto considerável no desempenho das Redes Neurais profundas quando as premissas não estão contempladas, estas também serão adaptadas no momento da aplicação para os algoritmos de Deep Learning. Estas avaliações empíricas serão realizadas frente aos cenários de aplicação propostos para as diferentes estruturas consideradas (Capítulo 4).

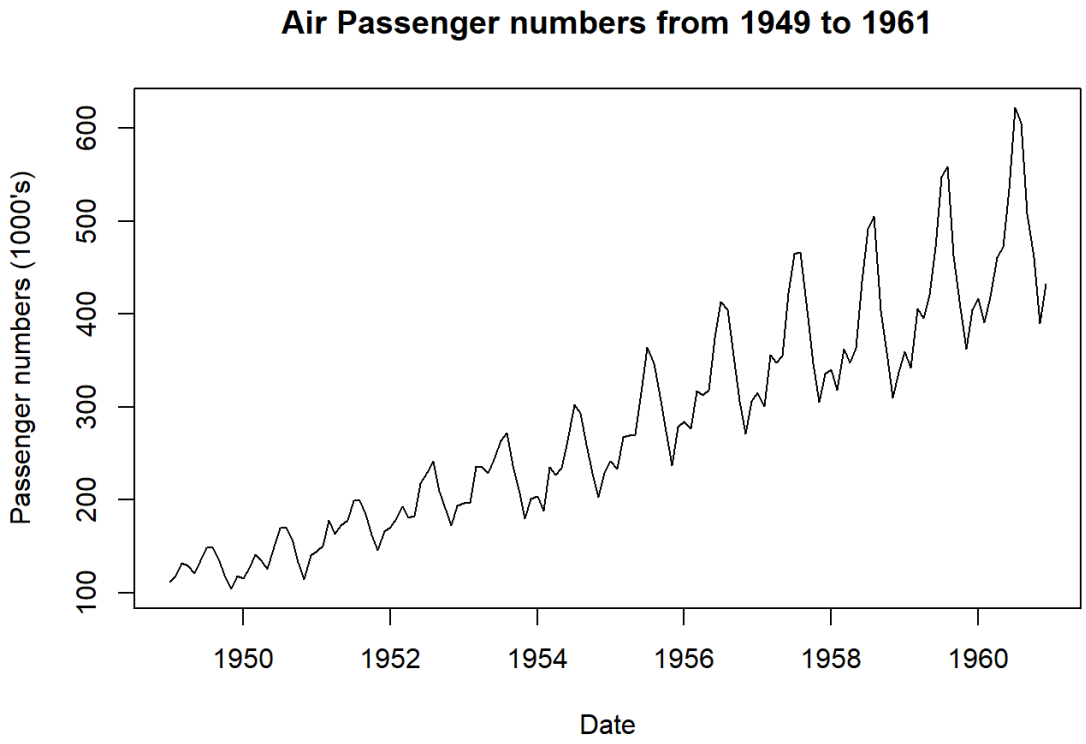


Figura 2.4: *Passagens aéreas nos Estados Unidos*

Capítulo 3

Deep Learning para Séries temporais

3.1 Deep Learning

Deep Learning é uma subárea de Machine Learning (Aprendizagem de Máquina) que abrange técnicas para simular o comportamento do cérebro humano em tarefas como por exemplo: reconhecimento visual, reconhecimento de fala, processamento de linguagem natural entre muitas outras aplicações. Este Capítulo apresenta uma introdução às técnicas, algoritmos e principais arquiteturas de Deep Learning com enfoque para estruturas que permitem sua aplicação em séries temporais para obtenção de previsões pontuais (também conhecido como Forecast do inglês).

Uma das motivações de Deep Learning originou-se na neurociência, mais especificamente no estudo da parte do cérebro denominada córtex visual. Sabe-se que, quando algum estímulo visual chega à retina, o sinal correspondente percorre uma sequência de regiões do cérebro, e que cada uma dessas regiões é responsável por identificar características específicas na imagem correspondente ao estímulo visual. Em particular, neurônios das regiões iniciais são responsáveis por detectar formas geométricas simples na imagem, tais como cantos e bordas, enquanto neurônios nas regiões finais têm a atribuição de detectar formas gráficas mais complexas, compostas das formas gráficas simples detectadas por regiões anteriores. Dessa forma, cada região de neurônios (que é análoga ao conceito de camada em redes neurais artificiais) combina padrões detectados pela região imediatamente anterior para formar características mais complexas. Esse processo se repete através das regiões até que os neurônios da região final têm a atribuição de detectar características de mais alto nível de abstração, tais como faces ou objetos específicos, por exemplo.

A ideia do deep learning tem como proposta que o aprendizado seja feito como o cérebro humano. Esta extração de características é feita em camadas, de forma similar ao funcionamento do cérebro para a visão. As camadas do algoritmo podem se especializar na identificação de formas especificas (similar aos neurônios), de modo que combinação das múltiplas camadas possibilitem a identificação de padrões complexos embutidos nos dados analisados.

Os algoritmos de Deep Learning estão, em sua maioria, conectados ao Big Data (grande volume de dados) até por conta da natureza das informações referentes as aplicações deste tipo (imagem, vídeo, texto etc.). As arquiteturas das redes, normalmente, são formadas por um número grande de parâmetros o que é ideal para um grande conjunto de dados. No entanto, esta dissertação visa justamente avaliar a aplicação dessas estruturas para dados tabulares que não necessariamente são considerados de grande volume (Big Data). Questões como o overfiting (sobreajuste) e a estimação de grande quantidade de parâmetros são avaliadas em termos da performance destes algoritmos aplicados no contexto de séries temporais.

3.1.1 Arquitetura Básica

A transmissão do sinal de um neurônio a outro no cérebro é um processo químico complexo, no qual substâncias específicas são liberadas pelo neurônio transmissor. O efeito é um aumento ou uma queda no potencial elétrico no corpo da célula receptora. Se este potencial alcançar o limite de ativação da célula, um pulso ou uma ação de potência e duração fixa é enviado para outros neurônios. Diz-se então que o neurônio está ativo. De forma semelhante à sua contrapartida biológica, uma RNA possui um sistema de neurônios artificiais (também conhecidas como unidades de processamento). Cada unidade realiza uma computação baseada nas demais unidades com as quais está conectada.

Os neurônios de uma RNA são organizados em camadas, com conexões entre neurônios de camadas consecutivas. As conexões entre unidades são ponderadas por valores reais denominados pesos(parâmetros). A RNA mais simples possível contém uma única camada, composta por um único neurônio. Redes dessa natureza são bastante limitadas, porque resolvem apenas processos de decisão binários (i.e., nos quais há duas saídas possíveis) e linearmente separáveis. Por outro lado, é possível construir redes mais complexas, capazes de modelar processos de decisão não linearmente separáveis, por meio de um procedimento de composição de blocos de computação organizados em camadas.

A Figura 3.1 ilustra esquematicamente os elementos da arquitetura de uma RNA simples, nessa arquitetura, a rede forma um grafo direcionado, no qual os vértices representam os neurônios, e as arestas representam os pesos das conexões. A camada que recebe os dados é chamada camada de entrada(input), e a que produz o resultado da computação é chamada camada de saída(output). Entre as camadas de entrada e saída, pode haver uma sequência de L camadas, onde ocorre o processamento interno da rede, cada elemento dessa sequência é chamado de camada oculta (Hiden Layer). Após o seu treinamento, a rede é capaz de realizar um mapeamento de vetores no espaço D-dimensional para vetores no espaço C-dimensional.

Cada neurônio de uma camada oculta recebe um vetor $x = (x_1, x_2, ..., x_N)$ como entrada e computa uma média ponderada pelos componentes de outro vetor de pesos associados $w = (w_1, w_2, ..., w_N)$, em que cada componente pode ser interpretado como a força da conexão correspondente. É comum também considerar uma parcela adicional nessa média ponderada, correspondente ao denominado viés (bias), cujo propósito é permitir que a rede neural melhor se adapte à função subjacente aos dados de entrada. Ao valor resultante dessa computação dá-se o nome de pré-ativação do neurônio (Equação 3.1).

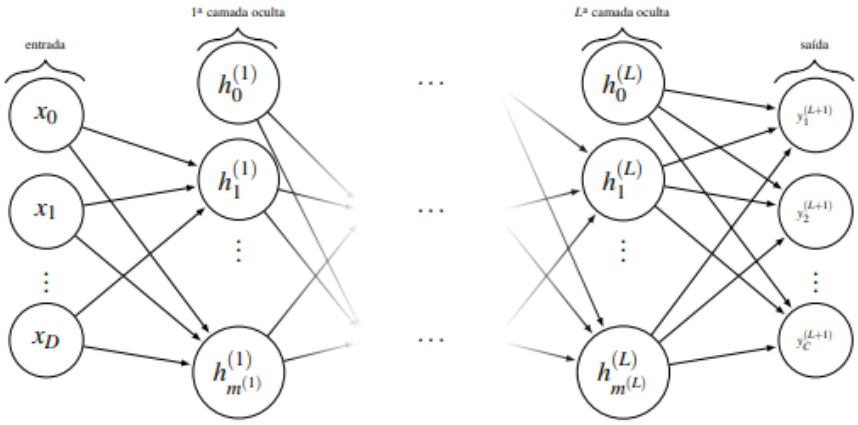


Figura 3.1: *Esquema de uma RNA com (L+1) camadas*

$$a(x) = b + \sum_i w_i x_i = \mathbf{b} + \mathbf{w}^{\mathrm{T}}\mathbf{x} \tag{3.1}$$

Após computar $a(x)$, o neurônio aplica uma transformação não linear sobre esse valor por meio de uma função de ativação, $g(\cdot)$. A função composta $h = g \circ a$ produz o que se chama de ativação do neurônio (ver Equação 3.2).

$$y = h(x) = g\left(a(x)\right) = g\left(b + \sum_i w_i x_i\right) \tag{3.2}$$

Há diferentes alternativas para a função de ativação g, dependendo da tarefa em questão. As três funções mais convencionais de ativação são a sigmoide logística (Equação 3.3), a sigmoide hiperbólica tangente (Equação 3.4) e a retificada linear (ReLu, Equação 3.5).

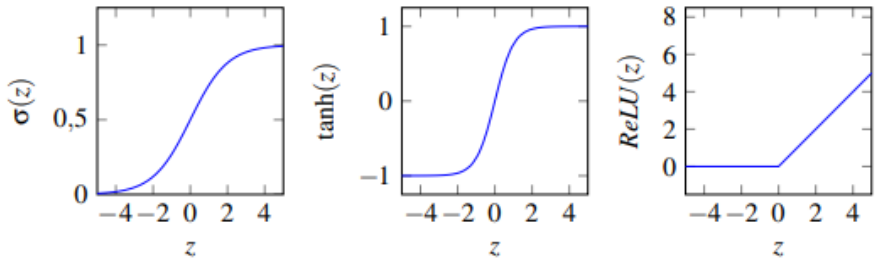
$$g(z) = \frac{1}{1 + e^{-z}} \tag{3.3}$$

$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{3.4}$$

$$g(z) = max(0, z) \tag{3.5}$$

A Figura 3.2 apresenta os gráficos dessas funções. Repare que, com exceção da ReLU em $z = 0$, todas são diferenciáveis e estritamente monotônicas. Repare também que as funções sigmoides possuem assíntotas que as limitam inferior e superiormente.

As equações previamente apresentadas para a pré-ativação e para a ativação de uma única unidade podem ser estendidas para o caso em que há L camadas ocultas, com vários neurônios em cada camada. Em primeiro lugar, em vez de pensar na pré-ativação de um único neurônio, podemos considerar a pré-ativação de toda uma camada oculta, conforme



(a) Logística. (b) Hiperbólica tangente. (c) Retificada linear.

Figura 3.2: *Funções de ativação*

a Equação 3.6. Nessa equação, que vale para todo $k > 0$, o sobrescrito k denota a k-ésima camada intermediária, a Equação 1 é aplicada a cada neurônio dessa camada. Explicação análoga pode ser apresentada para a Equação 3.7. Além disso, $W^{(k)}$ é a matriz de pesos das conexões entre os neurônios das camadas $k-1$ e k. Finalmente, considere que, se $k = 0$, então $h^{(k)}(x) = x$.

$$a^{(k)}(x) = b^{(k)} + W^{(k)} h^{(k-1)}(x) \tag{3.6}$$

$$h^k(x) = g\left(a^k(x)\right) \tag{3.7}$$

Como um todo, podemos considerar que, dado um vetor x de entrada, a RNA computa uma função $f(x)$ tal que $f(x) : R^D \longrightarrow R^C$, conforme a Equação 8.

$$f(x) = h^{(L+1)}(x) = o\left(a^{L+1}(x)\right) \tag{3.8}$$

Na Equação 3.8, $o(\cdot)$ representa a função de ativação usada para as unidades da camada de saída(output). A função de ativação da camada de Output (saída) está intrinsicamente relacionada com o tipo de resposta esperada da estrutura do algoritmo, diferentes objetivos diligenciam diferentes funções.

3.1.2 Conceitos de Deep Learning

No aprendizado supervisionado de uma RNA, considera-se a disponibilidade de um conjunto de treinamento da forma $\{(x(t), y(t)) : 1 \leq t \leq T\}$ em que $x(t)$ é o vetor de características associado ao componente $y(t)$. No contexto de redes neurais artificiais, esse aprendizado (ou treinamento) supervisionado corresponde a utilizar o conjunto de treinamento para ajustar os valores dos parâmetros da rede, de tal forma que o erro de treinamento (training error) seja minimizado. Dessa forma, o treinamento de uma rede neural é convertido em um problema de otimização, cujo objetivo é minimizar o erro cometido pela rede, quando considerados todos os exemplos de treinamento.

Com o propósito de medir o quanto a função inferida pela rede (Equação 3.8) se ajusta aos exemplos de treinamento, o problema de otimização é definido para minimizar uma

função de custo (cost function), $J(f(x^{(t)};\theta);y^{(t)})$ que mede a diferença (ou erro) que a rede comete relativamente ao exemplo $x^{(t)}$. O uso de θ na definição dessa função denota a dependência de J com relação ao vetor de parâmetros θ da RNA. A principal convenção para a função J é a soma dos erros quadrados (mean squared sum), no entanto a escolha da função de custo adequada depende de cada análise e suas respectivas especificidades. A Equação 3.9 (função de custo) corresponde ao custo relativo ao conjunto de treinamento como um todo.

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J\left(f\left(x^{(t)};\theta\right), y^{(t)}\right) + \frac{\lambda}{2T}\sum_{k}\sum_{l} w_{k,l}^{2} \tag{3.9}$$

A segunda parcela da Equação 3.9 contém a soma dos quadrados de todos os pesos da RNA. Nessa parcela, o parâmetro λ, outro hiperparâmetro (Mais detalhes na sessão 3.4) da RNA, controla a importância relativa entre as duas parcelas da função $J(\theta)$. Visto que o propósito da otimização é minimizar $J(\theta)$, conclui-se que combinações de valores grandes para os pesos não são desejadas, ao menos que façam com que o valor da primeira parcela seja muito pequeno. A intuição neste cenário condiciona o fato de que se o vetor de pesos ficar muito grande, um método de otimização que aplica pequenas alterações gradativas a esse vetor não terá condições de alterá-lo significativamente, resultando na convergência para uma solução inadequada. Além disso, incentivar o processo de otimização a manter os pesos pequenos resulta em uma RNA menos suscetível a peculiaridades nos dados de entrada.

Quando θ é um vetor de muitos componentes (quando existem muitas funções candidatas) ou o conjunto de treinamento não é suficientemente grande, o algoritmo de treinamento é capaz de memorizar todos os exemplos do conjunto, sem produzir um modelo com boa capacidade de generalização. Esse fenômeno é conhecido como overfiting (sobreajuste). A segunda parcela da Equação 3.9 é apenas uma forma, de diversas possíveis que podem ser usadas com o propósito de evitar que o processo de otimização fique restrito a uma solução sub-ótima. Existem outras técnicas com o mesmo propósito, que são genericamente conhecidas como técnicas de regularização Yoshua Bengio *et al.* (2015) .

Não há restrições de algoritmos de otimização frente ao treinamento de uma RNA. Entretanto, por razões práticas, um método de otimização comumente utilizado é o gradiente descendente (gradient descent, GD), ou uma de suas diversas variações (stochastic gradient descent, batch gradient descent etc.). O GD considera a superfície multidimensional representada pela função de custo J. A ideia básica desse procedimento é realizar (de forma iterativa) pequenas modificações no vetor de parâmetros θ que levem esse vetor na direção da maior descida nessa superfície, o que corresponde a seguir na direção do gradiente negativo da função, $-\nabla J$.

Considere que $w(t)$ corresponde a matriz de pesos de uma RNA na t-ésima iteração do GD, é possível provar que a atualização nos parâmetros realizada pelo GD é dada conforme a Equação 3.10. Nessa equação, γ é a taxa de aprendizagem (learning rate), um hiperparâmetro da RNA cujo valor adequado precisa ser encontrado. De acordo com o discutido em Bishop (2011), esta abordagem tem várias dificuldades. Uma delas é a escolha

do valor da taxa de aprendizagem que propicie o aprendizado rápido, mas ao mesmo tempo, evite o fenômeno da oscilação.

$$\Delta w(t) = -\gamma \frac{\partial J}{\partial w(t)} = -\gamma \nabla J\left(w(t)\right) \tag{3.10}$$

Um aspecto importante na Equação 3.10 diz respeito ao modo de calcular as derivadas parciais $\frac{\partial J}{\partial w(t)}$. Nesse contexto, um componente fundamental dos algoritmos para treinamento de redes neurais, incluindo as arquiteturas profundas modernas, é a retropropagação de erros através das camadas ocultas, a fim de atualizar os parâmetros (pesos e viés) utilizados pelos neurônios Rumelhart *et al.* (1986). Esse algoritmo, denominado retropropagação de erros (backward propagation of errors ou backpropagation) é combinado com algum método de otimização (e.g., gradiente descendente), que atua para minimizar a função de custo da RNA.

O conceito do algoritmo de retropropagação é como segue: se a saída produzida pela rede para uma determinada observação $x(t)$ é diferente da esperada, então é necessário determinar o grau de responsabilidade de cada parâmetro da rede nesse erro para, em seguida, alterar esses parâmetros com o propósito de reduzir o erro produzido. Para isso, dois passos são realizados, conforme descrição a seguir.

O primeiro passo está relacionado a determinar o erro associado à observação $x(t)$, inicialmente o algoritmo de retropropagação apresenta $x(t)$ como entrada para a rede e propaga esse sinal através de todas as camadas ocultas até a camada de saída (passo conhecido como forward step). Ao realizar esse passo, o algoritmo mantém o registro das pré-ativações $a(k)$ e ativações $h(k)$ para todas as camadas intermediárias, uma vez que estes valores serão utilizados para atualização dos parâmetros. O segundo passo (denominado passo de retropropagação) consiste em atualizar cada um dos pesos na rede de modo a fazer com que a saída produzida pela rede se aproxime da saída correta, minimizando assim o erro para cada neurônio de saída e para a rede como um todo.

O algoritmo de retropropagação é uma aplicação da regra da cadeia para computar os gradientes de cada camada de uma RNA de forma iterativa. De modo que os parâmetros (pesos) são atualizados, para as camadas da rede, até que o critério de convergência seja atingido dentro dos cenários propostos. Na sessão 3.6.1 este conceito será recapitulado com ênfase na propagação dos pesos ao longo do tempo (Backpropagation through time, BPTT), abordagem essencial para aplicação das RNAs no contexto onde há dependência temporal nos dados em análise.

Os modelos baseados em redes neurais artificiais são promissores devido possibilitarem a resolução de problemas os quais se obtêm pouco avanço com outras técnicas. Nesta sessão do terceiro Capítulo foi abordado os principais conceitos dentro do contexto de Deep Learning, que permite ampla flexibilidade na estrutura dos algoritmos. Estes conceitos são essenciais na aplicação e entendimento das estruturas de RNA para de Séries temporais que são objetos de exploração do Capítulo 4.

3.2 Diferentes Estruturas de Deep Learning para Séries Temporais

3.2.1 Convolutional Neural Networks (CNNs)

As redes convolucionais (Convolutional Neural Network, CNN) têm desempenhado um papel importante na história do Deep Learning. São exemplo importante de aplicação bem-sucedida de analogias obtidas pelo estudo do cérebro em aplicações de Machine Learning. Estas estruturas foram alguns dos primeiros modelos de Deep Learning a obter bom desempenho, muito antes de modelos de RNA arbitrários serem considerados viáveis. As CNNs também foram uma das primeiras estruturas de redes neurais a resolver aplicações comerciais importantes e permanecem na vanguarda das aplicações comerciais atualmente. Por exemplo, na década de 1990, o grupo de pesquisa de redes neurais da AT&T desenvolveu uma rede convolucional para ler verificações LeCun (1995), esse sistema foi implantado pela NEC e lia mais de 10% de todos os cheques nos Estados Unidos. Posteriormente, vários Optical Character Recognitio(OCR) e sistemas de reconhecimento de escrita foram influenciados e embutiram em sua estrutura redes convolucionais, muitos foram implantados pela Microsoft Simard (2003) .

As redes convolucionais também foram aplicadas para ganhar muitos concursos e competições. A intensidade atual do interesse comercial no aprendizado profundo começou quando Krizhevsky *et al.* (2012) venceu o desafio de reconhecimento de objeto ImageNet Russakovsky (2014), mas as redes convolucionais foram usadas para vencer outros concursos de aprendizado de máquina e visão computacional com menos impacto anos antes. As redes convolucionais foram algumas das primeiras redes profundas em funcionamento, treinadas com retropropagação. De muitas maneiras, estas estruturas carregaram expectativas para o avanço do Deep Learning e pavimentaram o caminho para a aceitação das redes neurais de um modo geral.

As redes convolucionais fornecem uma maneira de especializar as redes neurais de forma bem-sucedida em uma topologia de imagem bidimensional. Para processar dados sequenciais unidimensionais, a recomendação é outra especialização poderosa da estrutura das redes neurais: as redes neurais recorrentes. Importante destacar que as CNNs também comportam a aplicação em dados sequenciais com dependência temporal, este texto se restringe apenas ao cenário de combinação de estruturas que envolvem camadas das redes CNNs (sessão 3.3.4) e aos conceitos dessa classe de Deep learning, que são essenciais para compreensão das demais classes de RNN consideradas nos cenários de aplicação do Capítulo 4.

Redes neurais convolucionais são inspiradas no funcionamento do córtex visual (Goodfellow *et al.* (2015)). Nesta Seção, são apresentados os principais conceitos acerca dessa classe de redes, assim como características particulares, tais como o compartilhamento de pesos(parâmetros), que são auxílio para a classe de RNAs Recorrentes (sessão 3.3.2). A estrutura de convolução será importante para compreensão de uma classe específica das Redes Neurais Recorentes as LSTMs (Long Short Term Memory), que têm papel importante na aplicação de Deep Learning para séries temporais, assim como suas respectivas variações e combinações de estrutura (ConvLSTM, combinação de CNN e LSTM, por exemplo; sessão

3.3.4).

Redes de convolução se baseiam em algumas ideias principais, a saber, compartilhamento de pesos (shared weights), convolução (convolution) e subamostragem (subsampling ou pooling).

- **Compartilhamento de Pesos/Parâmetros(Shared Weights)** : em uma CNN as unidades de uma determinada camada são organizadas em conjuntos disjuntos, cada um dos quais é denominado um mapa de característica (feature map), também conhecido como filtro. As unidades contidas em um mapa de características são únicas na medida em que cada uma delas está ligada a um conjunto de unidades (i.e., ao seu campo receptivo) diferente na camada anterior. Além disso, todas as unidades de um mapa compartilham os mesmos parâmetros (i.e., a mesma matriz de pesos e viés). O resultado disso é que essas unidades dentro de um mapa servem como detectores de uma mesma característica, mas cada uma delas está conectada a uma região diferente da imagem ou matriz/conjunto de dados. Portanto, em uma CNN, uma camada oculta é segmentada em diversos mapas de características (os planos ilustrados na Figura 3.3), em que cada unidade de um mapa tem o objetivo realizar a mesma operação sobre a imagem de entrada, com cada unidade aplicando essa operação em uma região específica do conjunto de dados com os parâmetros compartilhados.

- **Convolução** : Para entender essa operação, considere que os dados de entrada da CNN é uma matriz de pixels $28x28$. Por simplicidade, considere ainda que essas imagens de entrada são em escala de cinza. Em processamento de imagens, uma convolução sobre uma imagem L corresponde a aplicar o produto de Hadamard Horn e Johnson (2012) entre a matriz de pixels de L e outra matriz, denominada núcleo da convolução (convolution kernel). Por exemplo, a Figura 3.4 apresenta o resultado da aplicação de um tipo particular de convolução sobre uma matriz de dados.

- **Pooling**: Este tipo de camada em uma CNN é a denominada camada de subamostragem (subsampling layer ou pooling layer). O objetivo desta camada, que também é composta por um conjunto de mapas de características, é realizar a agregação das saídas (ativações) de um conjunto de unidades da camada anterior. É possível mostrar que camadas de subamostragem resultam em uma rede mais robusta a transformações espaciais (e.g., mais invariante a eventuais translações dos objetos em uma imagem, por exemplo).

Em geral, uma CNN possui diversos tipos de camadas: camadas de convolução, camadas de subamostragem, camadas de normalização de contraste e camadas completamente conectadas (full conected layers). Na forma mais comum de arquitetar uma CNN, a rede é organizada em estágios. Cada estágio é composto por uma ou mais camadas de convolução em sequência, seguidas por uma camada de Pooling, que por sua vez é seguida (opcionalmente) por uma camada de normalização. Uma CNN pode conter vários estágios empilhados após a camada de entrada. Após o estágio final da rede, são adicionadas uma ou mais camadas completamente conectadas.

Por meio das redes neurais convolucionais, é possível realizar a detecção de padrões

de forma mais adequada, no sentido de que características dos dados dessa natureza podem ser exploradas para obter melhores resultados. Desde sua origem, esse tipo de rede tem se mostrado adequado em tarefas que envolvem reconhecimento visual, tais como reconhecimento de caracteres manuscritos e detecção de faces LeCun (1995). Com o sucesso resultante do uso de arquiteturas de Deep Learning para o processamento visual e o surgimento de bases de dados de imagem com milhões de casos rotulados (por exemplo, ImageNet10, Places11), o estado da arte em visão computacional por meio de CNNs tem avançado rapidamente.

De fato, desde 2012, a área de Visão Computacional tem sido explorada por abordagens que usam CNNs para tarefas de classificação e detecção de objetos em imagens, em substituição às abordagens anteriores que envolviam a extração manual das características das imagens para posteriormente aplicar algum método de classificação. Aplicações mais recentes de CNNs envolvem a detecção de pedestres e de placas de sinais de trânsito, carros autônomos e diversas outras aplicações. Para maiores detalhes sobre este tipo de estrutura de deep Learning a referência, Sermanet (2013), é indicada.

Apesar dos algoritmos da classe CNN possibilitarem a aplicação de dados com dependência temporal o principal objetivo desta sessão é introduzir os conceitos populares desta estrutura pois são utilizados de forma abrangente por outras classes de RNAs. As estruturas de Deep Learning são tão flexíveis que permitem a combinação de diferentes camadas/estruturas em um único algoritmo, este tema será explorado na sessão 3.3.4 na descrição de abordagens hibridas que permitem dependência temporal.

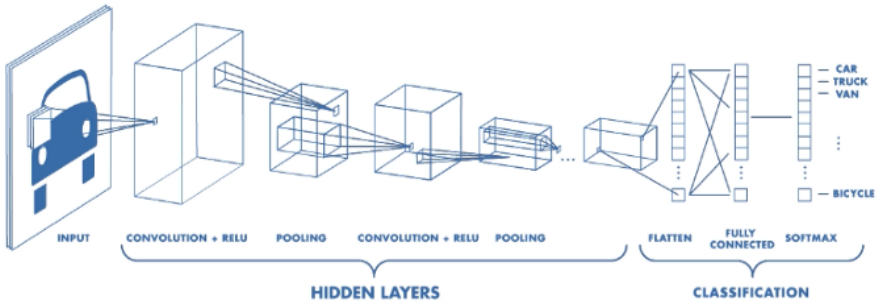


Figura 3.3: *Convolutional Neural Network*

3.2.2 Recurrent Neural Network(RNNs)

Redes Neurais Recorrentes (Recurrent Neural Networks, RNN) constituem uma ampla classe de redes cuja evolução do estado depende tanto da entrada corrente quanto do estado atual interno da rede. Essa propriedade (ter estado dinâmico) proporciona a possibilidade de realizar processamento dependente de contexto e aprender dependências de longo prazo: um sinal que é fornecido a uma rede recorrente em um instante de tempo t pode alterar o comportamento dessa rede em um momento $t + k, k > 0$.

A arquitetura usada em RNNs é adequada para permitir o processamento de informação sequencial (e.g., textos, sequências numéricas, áudio e vídeo). Nesse tipo de tarefa, é notável

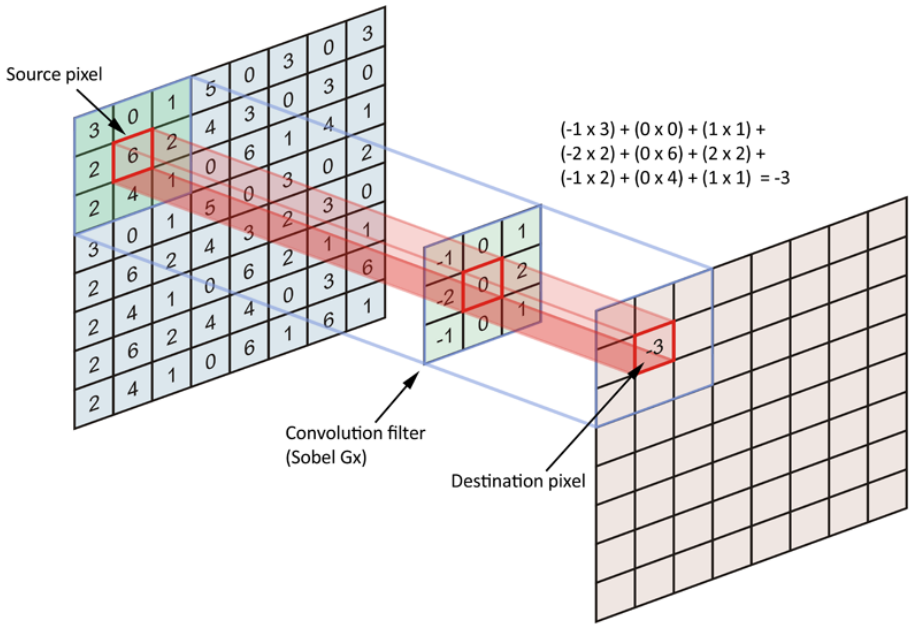


Figura 3.4: *Convolução Matricial*

perceber que o conhecimento da sequência de elementos anteriores é importante para a predição. Uma particularidade das RNNs é que esse tipo de rede possui uma memória, que permite registrar quais informações foram processadas até o momento atual, característica interessante no contexto de séries temporais em que a dependência no tempo é fundamental para realizar projeções futuras.

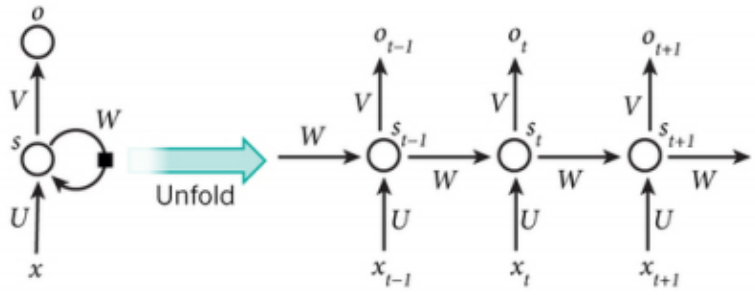


Figura 3.5: *Recurrent Neural Network (Unfold)*

Há muitas maneiras pelas quais as unidades de uma RNN podem estar conectadas entre si. A ilustração, parte esquerda da Figura 3.5, apresenta a situação mais simples possível, uma RNN que possui apenas uma camada oculta, composta de uma única unidade, denotada por s. Apresentada dessa forma, a rede é aparentemente simples, entretanto, repare que a unidade oculta contém uma conexão que a liga consigo mesma. Dessa forma, s recebe sinais que foram gerados por ela própria um passo de tempo no passado.

O desdobramento no tempo (time unfolding) de uma RNN corresponde à situação em que essa rede recebe uma sequência de sinais da entrada, um em cada passo de tempo. Na ilustração da Figura 3.5 (lado direito), é apresentada a mesma rede da esquerda desdobrada (unfolded) no tempo. Note que, vista dessa forma, uma RNN que recebe uma sequência de

n vetores de entrada podem ser vista como uma rede alimentada adiante(feedfoward) de n camadas.

Note também que, embora na Figura 3.5 são apresentadas entradas e saídas em cada passo de tempo, dependendo da tarefa específica para qual a RNN deve ser projetada, alguns dos passos de tempo podem não conter nem entradas nem saídas. Isso permite que RNNs aprendam mapeamentos entre sequências de entrada e de saída com diferentes tamanhos: um-para-n (e.g., produzir a legenda correspondente a uma imagem), n-para-um (e.g., a previsão de uma série temporal), n-para-m (e.g., traduzir uma frase do inglês para o português). Considerando ainda o cenário da Figura 3.5, os vários componentes de uma RNN representados nessa ilustração são descritos a seguir:

- x_t é o vetor que representa a entrada fornecida à RNN no passo de tempo t. Por exemplo, se a RNN processa sequências de números, então x_0 é a representação do primeiro elemento, x_1 representa o segundo elemento, e assim por diante. Note que x_0 e x_1 podem neste cenário ser um vetor, não há restrição de que sejam escalares.
- s_t é o estado (i.e., o valor de ativação) da unidade oculta no passo de tempo t. Essa unidade representa a memória da rede, porque sua finalidade é registrar qual informação foi processada desde alguns passos de tempo no passado até o presente momento. A ativação dessa unidade é computada como uma função recursiva, que depende do estado anterior e da entrada no passo atual: $s_t = \sigma(Ux_t + W_s(t-1) + b_s)$. σ é a função de ativação escolhida.
- o_t é a saída no passo de tempo t. Por exemplo, se a tarefa corresponde a predizer a próximo elemento de uma sequência numérica, então o_t é uma distribuição de probabilidades sobre todos os valores possíveis, i.e., $o_t = softmax(Vs_t + b_0)$.
- As matrizes U (conexões entre estados ocultos), V (conexões dos estados ocultos para a saída) e W (conexões da entrada para os estados ocultas) são os parâmetros que a rede deve aprender durante o treinamento. De forma similar ao que acontece com as redes convolucionais, esses parâmetros são compartilhados por todos os passos de tempo. Esse compartilhamento permite que a rede processe entradas de tamanho variável.

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

Figura 3.6: *Função Softmax para um vetor de entrada* $z = (z_1, z_2, ..., z_n)$

Uma vez definida a arquitetura da RNN, é possível iniciar o treinamento por meio da minimização de uma função de custo J. Nesse caso, uma variante do algoritmo de retropropagação do erro é utilizada, a denominada retropropagação através do tempo (Backpropagation Through Time, BPTT), veja Seção 3.2.2 . O BPTT pode ser entendido se considerarmos que uma RNN corresponde a uma rede alimentada adiante(feedfoward)

com uma camada para cada passo de tempo e na qual a matriz de pesos de cada camada é a mesma, isto é, ao considerar o desdobramento da rede através do tempo. Dessa forma, após o fornecimento da sequência de entrada, os gradientes são calculados e posteriormente alterados para manter a restrição de igualdade da matriz de pesos em cada camada oculta. Por exemplo, se $w1$ e $w2$ são os valores de conexões correspondentes em duas camadas ocultas distintas (em dois passos de tempo distintos), então $w1 = w2$. Para fazer com que esses pesos continuem sendo iguais no próximo passo de tempo, o BPTT calcula $\partial J/\partial w_1$ e $\partial J/\partial w_2$ e usa a média simples (ou a soma) dessas quantidades para atualizar tanto w_1 quanto w_2 . O procedimento BPTT é detalhado na sessão 3.3.6.

Devido a sua flexibilidade no que concerne ao processamento de entradas de tamanho variável, RNNs têm sido aplicadas recentemente em uma variedade de problemas. De fato, a maioria das aplicações recentes de RNNs envolve o uso de um tipo mais complexo dessa classe de redes proposto em 1997, as denominadas redes LSTM (Long Short-Term Memory), tema da próxima Sessão 3.3.3. De maneira concisa, uma rede LSTM é formada de blocos (ou células) que, por possuírem uma estrutura interna, podem armazenar um valor por uma quantidade arbitrária de passos de tempo. As redes LSTM e sua contrapartida mais moderna, as redes GRU (Gated Recurrent Unit), são também apropriadas para evitar o problema da dissipação dos gradientes Hochreiter e Schmidhuber (1997) .

O conceito da classe RNN é um dos grandes motivadores da aplicação de estruturas de Deep Learning para series temporais, pois permitem a identificação de padrões ao longo do tempo. As variações dessa classe de algoritmos tem sido objeto de estudo e pesquisa e novas abordagens têm sido consideradas, ao herdar características importantes das RNN convencionais em suas estruturas.

3.2.3 Long Short-Term Memory Networks (LSTMs)

Os humanos não começam a pensar do princípio a cada segundo. Ao ler este texto, o entendimento de cada palavra tem fundamento na compreensão de palavras anteriores. O aprendizado é aproveitado, pois os pensamentos têm persistência. Redes neurais tradicionais não são capazes de absorver informações de forma similar, redes neurais recorrentes resolvem esse problema (Sessão 3.3.2). São redes com loops, permitindo que as informações sejam persistidas ao longo do tempo. A Long-short term memory ou LSTM, classe dos algoritmos RNN, foi criada em 1997 e introduziu a ideia das portas(Gates) de controle da memória, para resolver obstáculo existente nas redes recorrentes tradicionais, que é a memória de longo prazo. Com o passar dos anos, recebeu algumas modificações e ultimamente tem tomado destaque entre as redes recorrentes, inclusive se tornando o estado da arte em alguns problemas. A Figura 3.6 descreve a representação de uma célula LSTM.

A ideia inteligente de introduzir Self-loops(interações internas) para produzir caminhos onde o gradiente pode fluir por longos períodos é uma contribuição fundamental das LSTM Hochreiter e Schmidhuber (1997). Uma adição crucial tem sido fazer com que o peso desse self-loop seja condicionado ao contexto, ao invés de fixo Gers *et al.* (2000). A LSTM foi considerada extremamente bem-sucedida em muitas aplicações, como o reconhecimento de caligrafia irrestrito Graves (2009), reconhecimento de fala Graves e Jaitly (2014), geração de caligrafia Graves (2013), tradução automática Sutskever *et al.* (2014), legendagem de

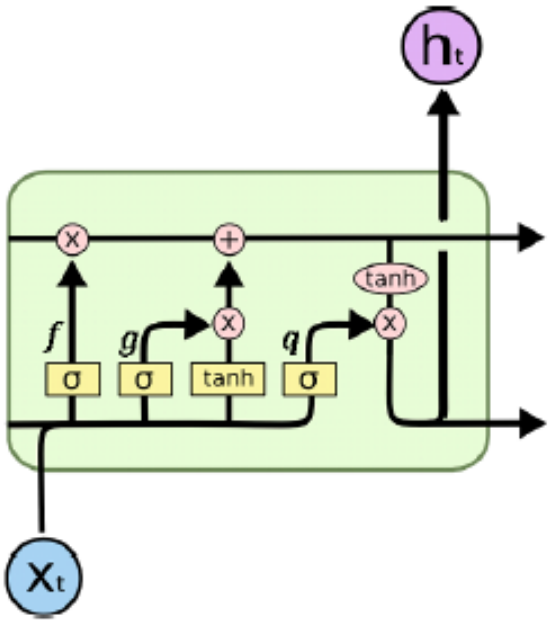


Figura 3.7: *Arquitetura de uma célula LSTM*

imagem Kiros *et al.* (2014) Cho *et al.* (2014) Xu e Zhu (2015) e análise Vinyals *et al.* (2014) .

Para esta estrutura de RNA, em vez de uma unidade que simplesmente aplica não linearidade a cada elemento para a transformação, LSTM têm "células LSTM"que possuem recorrência interna (um loop automático), além da recorrência externa da RNN. Cada célula tem as mesmas entradas e saídas como uma rede recorrente comum, mas tem mais parâmetros e um sistema de portas que controlam o fluxo de informações.

A estrutura da rede neural recorrente (analogamente ao descrito na Figura 3.5 da sessão 3.3.2), pode parecer complexa e misteriosa por conta dos Self-loops propostos em sua respectiva composição. No entanto, a análise detalhada da estrutura reflete que não são tão diferentes de uma rede neural convencional. Uma rede neural recorrente pode ser imaginada como múltiplas cópias da mesma rede, cada rede transmitindo uma mensagem a um sucessor. Considere o que acontece quando a RNN é desenrolada(unfolded) do looping interno (Figura 3.7).

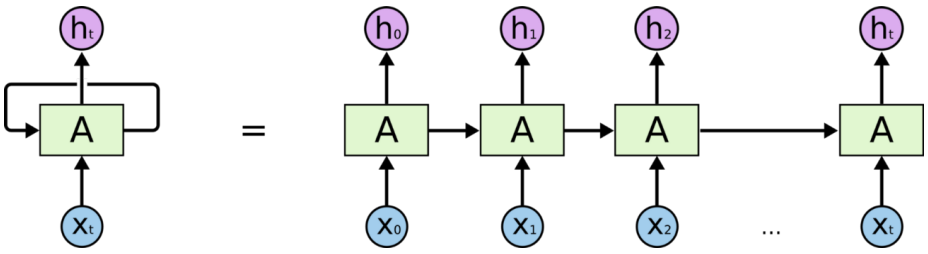


Figura 3.8: *RNN Unfolded*

Essa natureza de cadeia revela que RNNs estão intimamente relacionadas a sequências e listas, o que abrange séries temporais (sequência de números ordenados no tempo). São

o tipo de arquitetura natural de rede neural a ser usada para dados de séries temporais. Em teoria, as RNNs são absolutamente capazes de lidar com essas "dependências de longo prazo". Infelizmente, na prática, as RNNs não parecem ser capazes de aprendê-las. O problema foi explorado em profundidade por Hochreiter (1991) e Y. Bengio (1994) que encontraram algumas razões bastante fundamentais do porque isso pode ser difícil.

De modo abrangente o problema das RNNs é que os gradientes propagados ao longo de muitos estágios tendem a desaparecer (na maioria das vezes) ou explodir (raramente, mas com muitos danos à otimização). Mesmo ao assumir que os parâmetros são tais que a rede recorrente é estável (pode armazenar memórias, com gradientes não explodindo), surge a dificuldade com dependências de longo prazo dos pesos exponencialmente menores dados às interações de longo prazo (envolvendo a multiplicação de muitos jacobianos) em comparação com os de curto prazo.

Redes de Memória de Curto e Longo Prazo (LSTMs), são um tipo especial de RNN, capaz de aprender dependências de longo prazo. Os LSTMs são explicitamente projetados para evitar o problema de dependência de longo prazo (Vanish Gradient). Todas as redes neurais recorrentes têm a forma de uma cadeia de módulos repetitivos da rede neural. Em RNNs padrão, este módulo de repetição terá uma estrutura muito simples, como uma única camada (Figura 3.8).

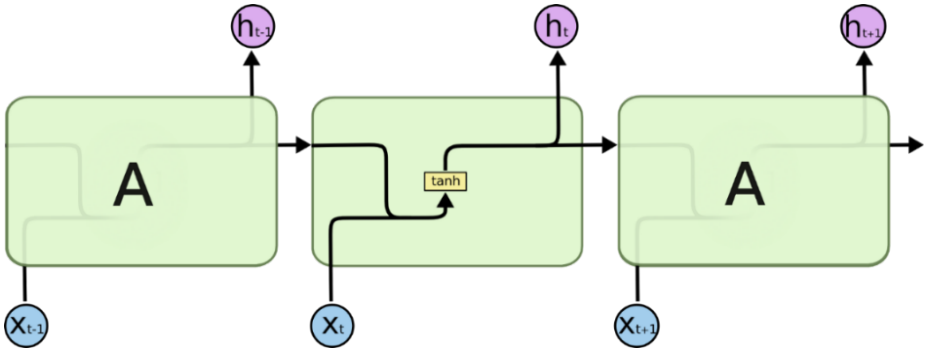


Figura 3.9: *RNN Padrão*

As LSTMs também possuem esta estrutura de cadeia, mas o módulo de repetição possui uma estrutura diferente. Em vez de ter uma única camada de rede neural, existem quatro, interagindo de uma maneira muito especial.

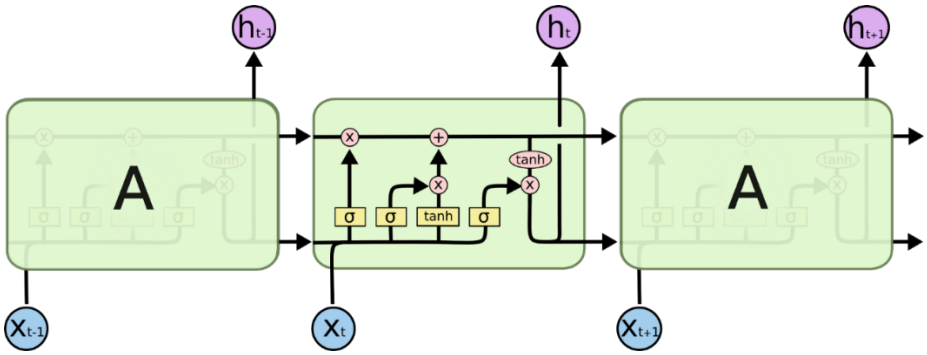


Figura 3.10: *módulo de repetição LSTM com as quatro camadas de interação*

No diagrama acima (Figura 3.10), cada linha carrega um vetor, desde a saída de um nó até as entradas de outro. Os círculos cor-de-rosa representam operações pontuais, como a adição de vetores, enquanto as caixas amarelas são camadas de redes neurais aprendidas. As linhas que se fundem denotam a concatenação, enquanto uma linha bifurcada denota que seu conteúdo está sendo copiado e as cópias vão para locais diferentes.

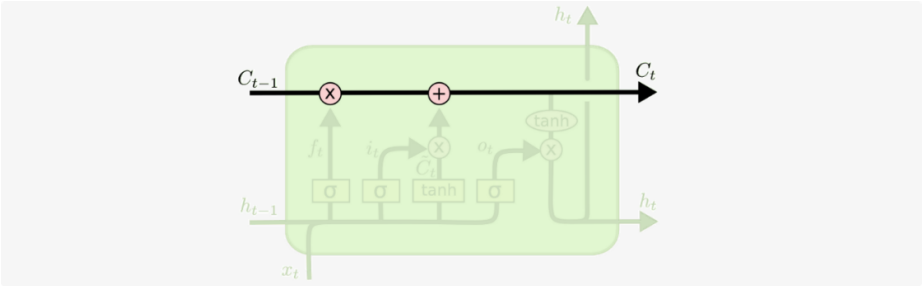


Figura 3.11: *Estado da Célula LSTM*

O diferencial das LSTMs é o estado da célula, a linha horizontal que passa pela parte superior do diagrama. O estado da célula é como uma correia transportadora. Ele percorre toda a cadeia, com apenas algumas interações lineares menores. Note que as informações podem fluir em um determinado instante sem sofrer alterações.

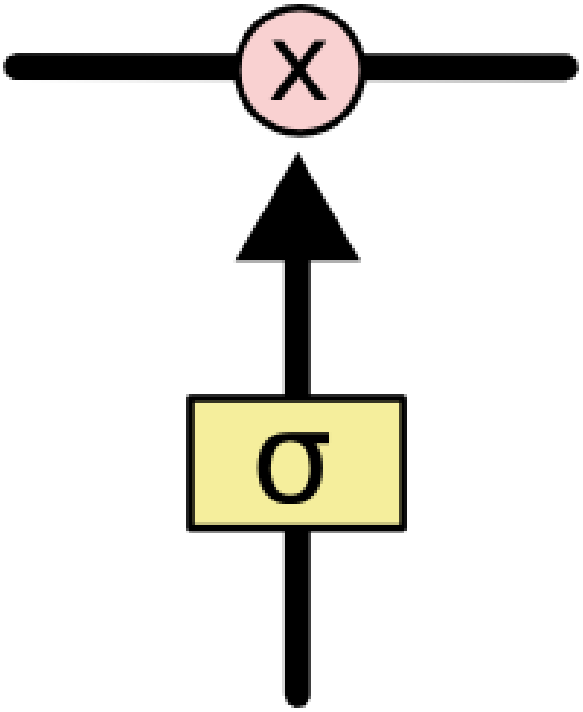


Figura 3.12: *Camada sigmóide e multiplicação pontual*

A LSTM tem a capacidade de remover ou adicionar informações ao estado da célula, cuidadosamente reguladas por estruturas chamadas portas (Gates, Portões). Os portões são uma forma de, opcionalmente, controlar o fluxo de informações ao longo do tempo. Eles são compostos de uma camada de rede neural sigmóide (Figura 3.2 - função sigmóide) e uma operação de multiplicação pontual (Figura 3.12).

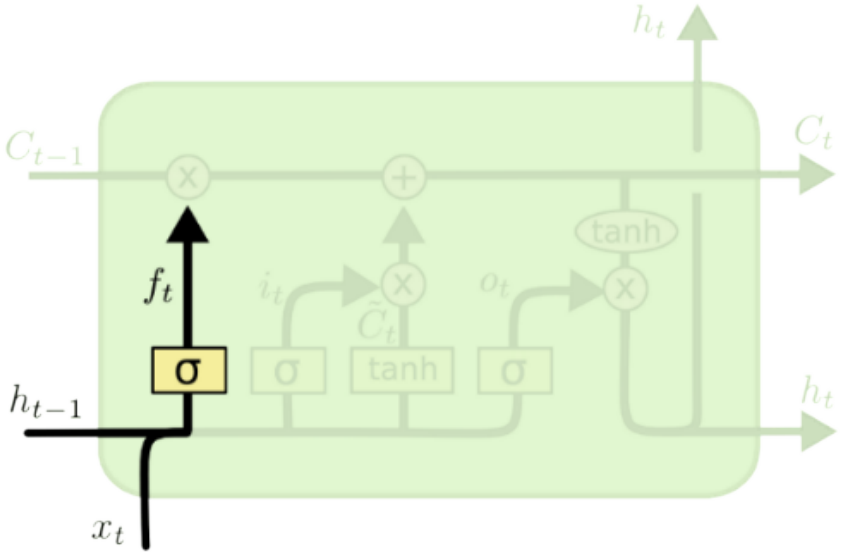


Figura 3.13: *Forget Gate*

O procedimento das LSTMs será descrito através de etapas para facilitar o entendimento do fluxo interno da célula. A primeira etapa na LSTM é decidir quais informações serão descartadas do estado da célula. Esta decisão é tomada por uma camada sigmóide chamado de "forget gate" (portão de esquecimento). Este portão observa h_{t-1} (estado anterior da célula) e x_t(dados de entrada no instante t) e gera um número entre 0 e 1 para cada número no estado da célula C_{t-1}. Um 1 representa "manter completamente isso", ao passo que 0 representa "completamente se livrar disso" (Figura 3.13).

A segunda etapa é decidir quais novas informações serão armazenadas no estado da célula. Este procedimento tem duas partes; primeiro, uma camada sigmóide decide quais valores serão atualizados. Em seguida, uma camada de tanh(função de ativação) cria um vetor de novos valores candidatos, $\widetilde{C}_t$ isso poderia ser adicionado ao estado. Na sequência, ambos serão combinados para criar uma atualização para o estado da célula (Figura 3.14).

A terceira etapa diz respeito a atualização do estado da célula antiga, C_{t-1}, no novo estado da célula C_t. O antigo estado da célula é multiplicado por f_t, (forget gate), esquecendo as informações consideradas não relevantes anteriormente. Então os dados de entrada(input) são adicionados a $[i_t \circ C]_{t-1}$ (Multiplicação de elemento por elemento conhecido como Hadamard). Esses são os novos valores candidatos, dimensionados na definição da atualização de cada valor de estado.

Finalmente, será gerado o output da célula, essa saída é baseada no estado da célula, mas será uma versão filtrada. Há uma camada sigmóide que decide quais partes do estado da célula serão produzidos. Então, o estado da célula é gerado através da tanh (para limitar os valores entre -1 e 1) e multiplicado pela saída da sigmóide, de modo que apenas as partes de interesse sejam produzidas.

Este é o detalhamento do procedimento que os dados percorrerem em um cenário de arquitetura de uma célula LSTM. Os portões realizam o controle de todo percurso dos dados, onde é realizada a seleção de quais informações devem ser mantidas, quais devem

ser desconsideradas, quais informações devem ser propagadas no tempo para os próximos períodos. Essa abordagem possibilita que as características sequenciais sejam armazenadas na memória (estado) da célula LSTM, o que possibilita aplicações de projeções de valores futuros.

$$f_t = \sigma\left(W_f\left[h_{t-1}, x_t\right] + b_f\right) \tag{3.11}$$

$$i_t = \sigma\left(W_i \cdot \left[h_{t-1}, x_t\right] + b_i\right) \tag{3.12}$$

$$\widetilde{C}_t = tanh\left(W_C \cdot \left[h_{t-1}, x_t\right] + b_c\right) \tag{3.13}$$

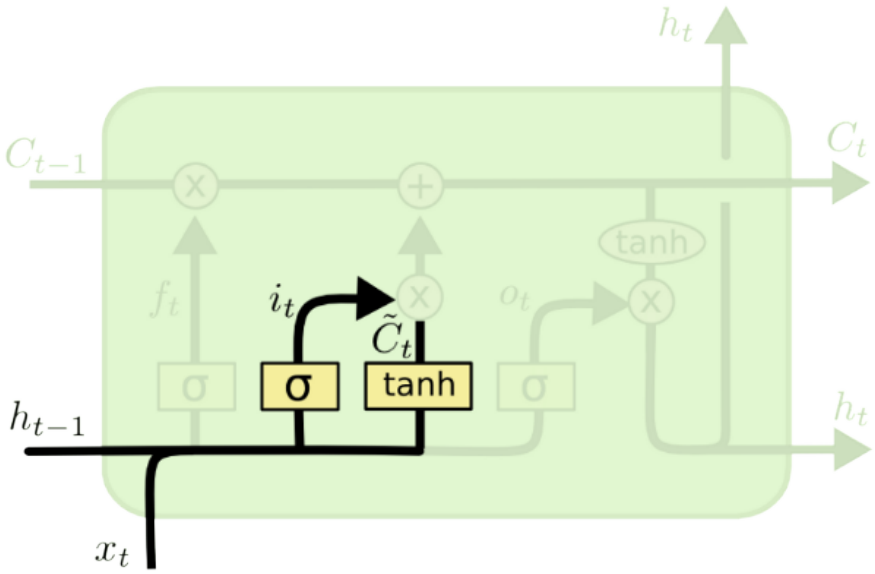


Figura 3.14: *Segunda Etapa (camada sigmóide e função de ativação tanh)*

Este é o procedimento de funcionamento da estrutura LSTM, este fluxo ocorre de forma recorrente em cada instante de tempo t, em que as informações da célula são atualizadas de acordo com o procedimento descrito nesta sessão. Desta forma, as LSTMs resolvem a questão do problema do Gradiente e permitem que dependências temporais sejam consideradas para compreensão de fenômenos geradores assim como previsão de valores futuros.

Há múltiplas variações das LSTMs, está estrutura tem sido alvo de muito desenvolvimento acadêmico em termos de pesquisa e aplicação. Uma variação notável referente as LSTM é a Unidade Recorrente Gated, ou GRU, introduzida por Cho *et al.* (2014) . Esta variação da LSTM(GRU) combina as portas de esquecimento e de entrada em uma única "porta de atualização". Além de mesclar o estado da célula e o estado oculto e algumas outras.

$$C_t = f_t \circ C_{t-1} + i_t \circ \widetilde{C}_t \tag{3.14}$$

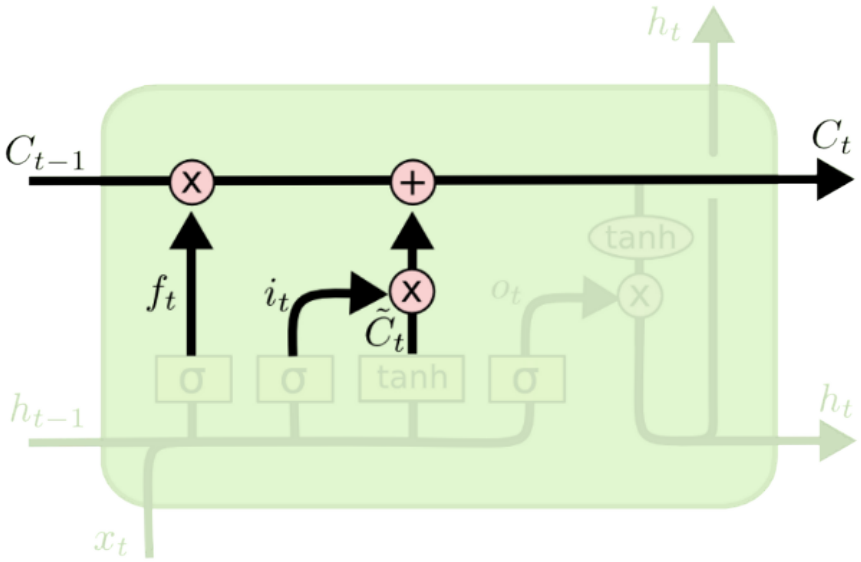


Figura 3.15: *Terceira Etapa (Estado de célula)*

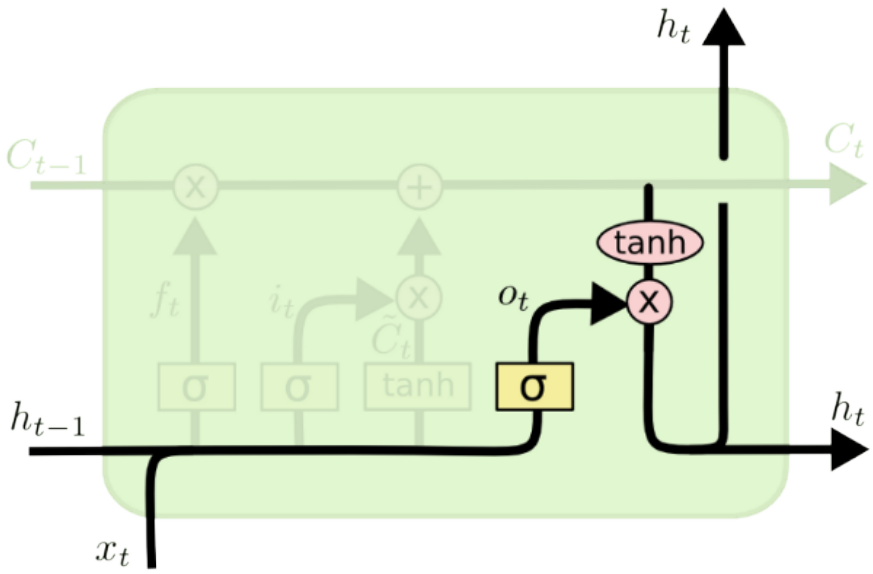


Figura 3.16: *Output e Hiden State (LSTM)*

$$O_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right) \tag{3.15}$$

$$h_t = o_t \circ tanh\left(C_t\right) \tag{3.16}$$

$$z_t = \sigma\left(Wz \cdot \left[h_{t-1}, x_t\right]\right) \tag{3.17}$$

$$r_t = \sigma\left(W_r \cdot \left[h_{t-1}, x_t\right]\right) \tag{3.18}$$

$$\widetilde{h_t} = tanh\left(W \cdot \left[r_t \circ h_{t-1}, x_t\right]\right) \tag{3.19}$$

$$h_t = \left(1 - z_t\right) \circ h_{t-1} + z_t \circ \widetilde{h_t} \tag{3.20}$$

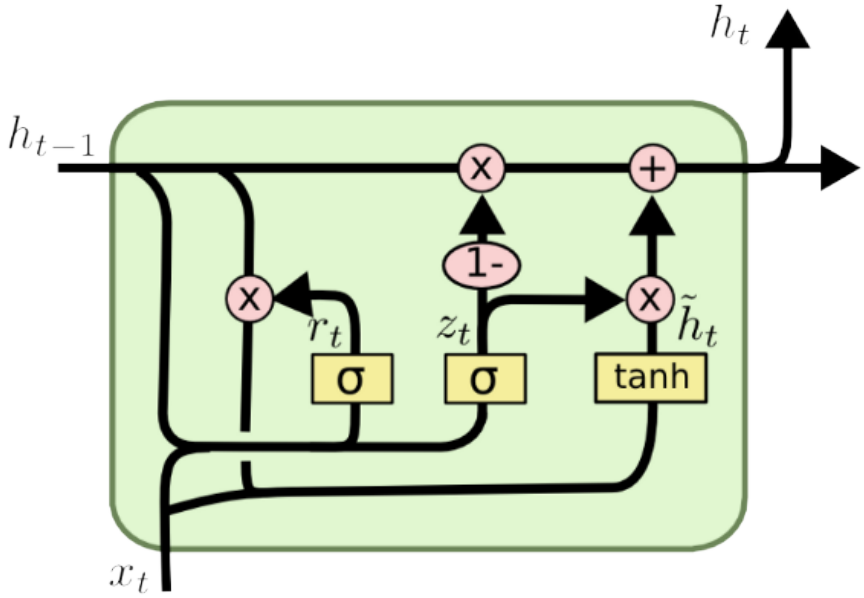


Figura 3.17: *Célula GRU*

GRU é apenas uma das variantes mais notáveis da estrutura LSTM, há muitas outras arquiteturas, como RNNs Profundas. Há também uma abordagem completamente diferente para lidar com dependências de longo prazo, como RNNs de Koutník *et al.* (2014) . Greff *et al.* (2015) fazem uma boa comparação de variantes populares, achando que são todas similares. Jozefowicz *et al.* (2015) testaram mais de dez mil arquiteturas RNN, encontrando algumas adaptações com boas performance em tarefas especificas.

De forma mais recente o algoritmo proposto pelo Google denominado Transformers com mecanismo Attention (atenção) tem se destacado no que diz respeito a dependência de sequências. O Mecanismo de Atenção consiste em mapear o relacionamento relevante

dos diferentes elementos da sequência entre si possibilitando a propagação de informações relevantes ao longo do processamento da sequência. Essa abordagem tem sido considerada o estado da arte em diversas aplicações no campo textual, mas é flexível o suficiente para ser generalizada para outras áreas do conhecimento. Na sessão 3.3.3 é descrito em detalhes a proposta do algoritmo transformers e seu respectivo mecanismo de atenção.

3.2.4 Parâmetros da Estrutura LSTM

Esta sessão tem por objetivo descrever o processo interno da estrutura LSTM para compreensão da quantidade de parâmetros da rede proposta. Para compreender como é realizado o cálculo dos parâmetros de uma LSTM é fundamental a compreensão do cálculo de parâmetros de uma camada densa (dense layer), pois a LSTM tem 4 camadas densas em sua respectiva estrutura interna. A Figura 3.18 representa uma camada densa em que: i = tamanho dos dados de entrada (input size), h = tamanho da camada oculta, o = tamanho dos dados de saída (output size).

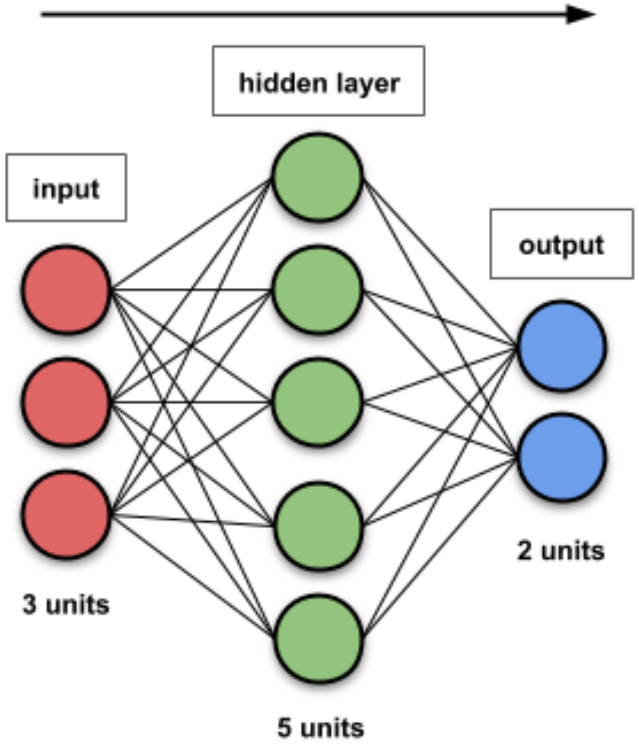


Figura 3.18: *Camada Densa (Dense layer)*

Ao considerar o cenário da Figura acima, com apenas uma única camada oculta, o número de parâmetros da estrutura pode ser expresso pela seguinte equação:

$$(i \cdot h + h \cdot o) + (h + o) \tag{3.21}$$

No exemplo da Figura 3.18 acima temos que i = 3, h =5 e o =2. Considerando a Equação

3.21 temos que a quantidade de parâmetros (rede Densa e totalmente conectada) neste exemplo é 32 parâmetros. A Figura 3.19 destaca a estrutura interna de uma célula LSTM, existem três entradas para a célula LSTM: h_{t-1} valor do estado oculto, c_{t-1} valor do estado da célula, x_t valor de entrada no momento atual. Há quatro portões: Forget Gate(portão do esquecimento),Input Gate(Entrada + Candidato), Output Gate (portão de Saída).

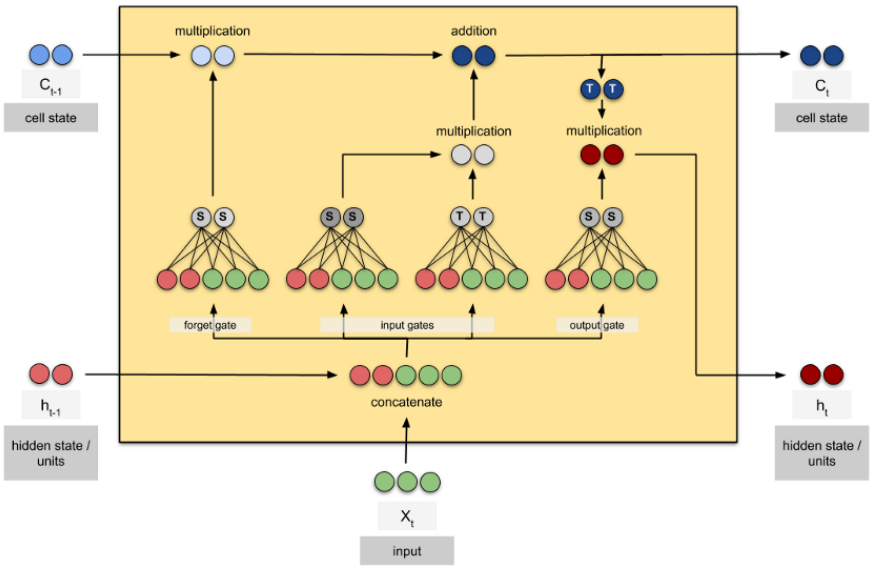


Figura 3.19: *Fluxo Célula LSTM*

As dimensões do vetor de entrada e saída são simbolizadas por círculos. Na figura 3.18, o estado da Célula (C_{t-1}) e a camada oculta (h_{t-1}) são vetores que têm uma dimensão igual a 2. Este número é definido no momento de configuração da rede LSTM (LSTMoutputDimension). A entrada é um vetor que tem uma dimensão igual a 3. Este número também é definido na configuração ao decidir quantas dimensões irão representar uma entrada.

Observe que, por definição:

- As dimensões do vetor de estado oculto (hidden layer : h_{t-1}) e vetor de input (x_t) do algoritmo devem ser as mesmas.
- As dimensões do vetor de estado oculto (hidden layer : h_{t-1}) vetor de input (x_t) no tempo $t-1$ e t devem ser as mesmas.
- Cada entrada na sequência deve ter as mesmas dimensões vetoriais.

Conforme visto na figura acima, existem valores de estado Oculto (2 círculos vermelhos) e valores de entrada (3 círculos verdes). Total de 5 números são introduzidos em uma camada densa. A camada de saída tem 2 valores que devem ser iguais à dimensão de h_{t-1} (vetor de estado oculto na célula LSTM). Deste modo, é possível calcular o número de parâmetros nesta camada densa conforme realizado no início desta sessão:

$$Parâmetros\ por\ Gate = (h_{t-1} + x_t) \cdot h_{t-1} \ + \ h_{t-1} \tag{3.22}$$

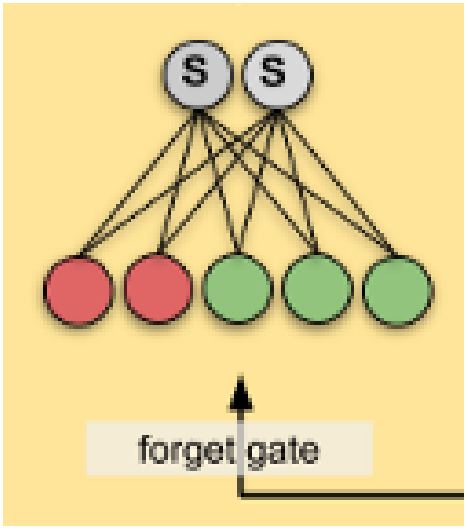


Figura 3.20: *Forget Gate*

No exemplo em destaque o Forget Gate tem 12 parâmetros (pesos + tendências). Uma vez que existem 4 Gates na unidade LSTM que têm exatamente a mesma arquitetura de camada densa, haverá 48 parâmetros (4 × 12) na estrutura completa do exemplo ilustrado pela Figura 3.19. É possível formular os números dos parâmetros em uma camada LSTM dado que x é a dimensão de entrada, h é o número de unidades (células ou dimensões de saída) da LSTM:

$$Número\ total\ de\ Parâmetros = (\ 4 \cdot ((x\ +\ h) \cdot h\ +\ h)) \tag{3.23}$$

Alternativamente, os parâmetros podem ser obtidos por intermédio das equações matemáticas que descrevem cada Gate (portão) da estrutura LSTM. As saídas das 4 portas na Figura 3.19 podem ser expressas da seguinte forma:

$$f_t = \sigma_g \left(W_f x_t + U_f h_{t-1} + b_f\right) \tag{3.24}$$

$$i_t = \sigma_g \left(W_i x_t + U_i h_{t-1} + b_i\right) \tag{3.25}$$

$$o_t = \sigma_g \left(W_o x_t + U_o h_{t-1} + b_o\right) \tag{3.26}$$

$$\widetilde{c_t} = \sigma_h \left(W_c x_t + U_c h_{t-1} + b_c\right) \tag{3.27}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \widetilde{c_t} \tag{3.28}$$

$$h_t = o_t \circ \sigma_h \left(c_t\right) \tag{3.29}$$

Variáveis

$x_t \in \mathbb{R}^d$: vetor de entrada da unidade LSTM

$f_t \in \mathbb{R}^h$: vetor de esquecimento (forget)

$i_t \in \mathbb{R}^h$: entrada do vetor do ativação

$o_t \in \mathbb{R}^h$: vetor de output(saída)

$h_t \in \mathbb{R}^h$: estado escondido conhecido como vetor de saída da unidade LSTM

$\widetilde{c_t} \in \mathbb{R}^h$: vetor de ativação de entrada da célula

$c_t \in \mathbb{R}^h$: vetor de estado da célula

$W \in \mathbb{R}^{hxd}$, $U \in \mathbb{R}^{hxh}$ $e\, b \in \mathbb{R}^h$: matrizes de peso e parâmetros de viés

Funções de ativação

σ_g: função sigmóide

σ_c: função tangente hiperbólica

σ_h: função tangente hiperbólica, considerando que $\sigma_h(x) = x$

Como visto, entre essas funções, existem 4 funções que têm 2 matrizes de peso: W e U para valores h_{t-1} e x_t, 1 vetor de polarização b. Ao considerar o portão de esquecimento (forget gate), note que os quatro portões são similares no quesito quantidade de parâmetros, veja Figura 3.19.

Existem 4 funções que são definidas exatamente da mesma forma, na camada LSTM, portanto o Número do parâmetro LSTM é dado por: $4((x + h)h + h)$, resultando na mesma quantidade de parâmetros da Equação 3.23. A compreensão da estrutura de parâmetros da rede é de extrema importância no momento da aplicação uma vez que há flexibilidade de escolha da estrutura interna e empiricamente são testadas as estruturas que resultam em melhores ajustes aos cenários dos dados em análise. Adicionalmente, a quantidade de parâmetros da rede é fundamental frente a quantidade de dados disponível para o treinamento afim de evitar o sobreajuste (Overfitting) nos dados.

3.2.5 Transformers (Mecanismo Attention)

A arquitetura das LSTMs processa os dados sequencialmente (Figura 3.10), mantendo o estado oculto atualizado a cada nova entrada. Teoricamente, informações muito importantes podem se propagar em sequências longas, no entanto, por ser sequencial e restringir a quantidade de informação durante a propagação no tempo, pode haver limitações de performance neste sentido. Em 2017 foi introduzido o modelo Transformer que tem revolucionado a área de Processamento de linguagem Natural (PLN ou NLP do inglês), devido a sua arquitetura e procedimento para adoção de problemas complexos. A grande diferença dos Transformers em relação as LSTMs é a permissão da captura de informações de forma paralela frente aos dados e não apenas sequencial, o que aumenta consideravelmente o poder de processamento para grandes sequencias (principalmente as textuais).

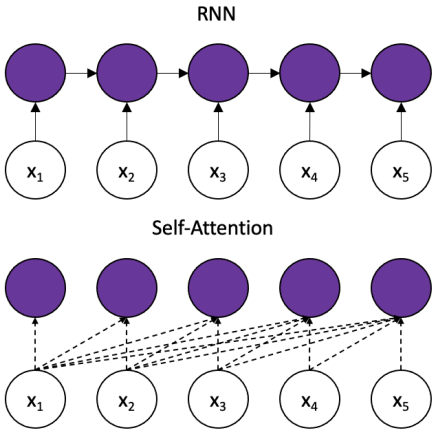


Figura 3.21: *RNN e Self-Attention*

A Figura 3.22 representa a arquitetura completa do modelo original proposto como Transformers. Uma das grandes inovações é o mecanismo de Atenttion(Atenção em tradução literal) que tem diferentes variações sendo a mais popular o "Self-Attention", grande diferencial da estrutura Transformers.

Como as RNNs os Transformers são projetados para lidar com dados de entrada sequenciais, como linguagem natural, para tarefas como tradução e resumo de texto, além de outros tipos de sequências. No entanto, ao contrário das RNNs, os Transformers não processam necessariamente os dados de forma ordenada. Em vez disso, o mecanismo de Atenção fornece contexto para qualquer posição na sequência de entrada. Por exemplo, se os dados de entrada são uma sequência de números, o Transformer não precisa processar o início da sequência antes do final. Em vez disso, identifica o contexto que confere significado a cada parte da sequência. Esse recurso permite mais paralelização do que RNNs e, portanto, reduz o tempo de treinamento.

Os Transformers, atualmente, é a principal classe de modelos de Deep Learning para problemas de Programação Neuro Linguística (PNL ou NLP do inglês), substituindo os modelos RNNs, como a memória de curto prazo longa (LSTM). A paralelização de treinamento adicional permite o treinamento em conjuntos de dados maiores do que antes. Isso levou ao desenvolvimento de sistemas pré-treinados, como BERT (Bidirectional Encoder Representations from transformers) Devlin e Chang (2018) e GPT (Generative Pre-Training Transformer), que foram treinados com grandes conjuntos de dados de linguagem, como Wikipedia Corpus e Common Crawl, e podem ser ajustados para tarefas específicas.

O mecanismo Attention(Atenção em tradução literal) tem recebido bastante notoriedade ultimamente, principalmente após produzir resultados de ponta em vários campos de pesquisa. De legendas de imagens e tradução de idiomas a respostas interativas de perguntas, a Atenção rapidamente se tornou uma ferramenta fundamental de compreensão dos pesquisadores. Este mecanismo permite que o modelo se concentre em partes importantes da sequência processada. A atenção se manifesta de maneira diferente em diferentes contextos. Em alguns problemas de PNL, a Atenção permite que o modelo coloque mais ênfase em palavras diferentes (i.e, partes especificas da sequência) durante a previsão, por exemplo, de Yang (2018) .

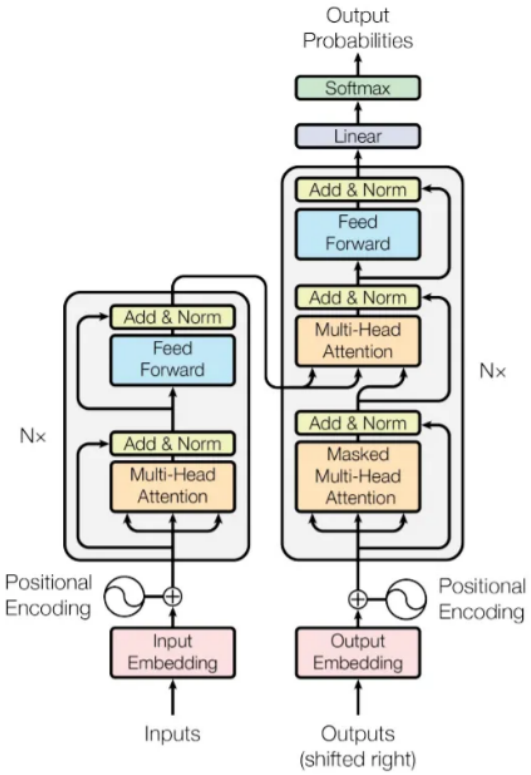


Figura 3.22: *Estrutura Transformers*

Em psicologia, atenção é a concentração da consciência em um estímulo enquanto exclui outros. Da mesma forma, os mecanismos de atenção são projetados para enfocar apenas os subconjuntos mais importantes de sequências arbitrariamente longas, que são relevantes para realizar uma determinada tarefa. Concretamente, o modelo deve decidir quais detalhes das informações anteriores são relevantes para a codificação atual. O bloco de Self-Attention (auto-atenção em tradução literal) codifica cada nova entrada em relação a todas as outras entradas anteriores, focalizando com base em um cálculo de relevância em relação a informação atual. O artigo "llustrated Transformer" de Alammar (2018) descreve todo o mecanismo Transformers com riqueza de detalhes.

O mecanismo de atenção tem como base vetores de consulta, chaves e valores, além do processo de codificação e decodificação operado por Tokens. Cada token gera uma consulta, chave e valor associados. A consulta do token atual é comparada com as chaves de todos os outros tokens, a fim de decidir sua relevância em relação ao token atual. O vetor de valor é a verdadeira representação de um determinado token, que é usado na criação da nova codificação. Durante o treinamento, o modelo aprende gradualmente essas três matrizes de peso pelas quais cada token é multiplicado para gerar sua chave, consulta e valor correspondentes.

A pontuação de Autoatenção é a medida de relevância entre o token atual e qualquer outro token que tenha sido visto anteriormente na sequência. É calculado computando o produto escalar entre o vetor de consulta do token atual com o vetor chave do token sendo pontuado. Pontuações altas indicam alta relevância, enquanto pontuações baixas indicam o oposto. As pontuações são então passadas por uma função softmax, de modo

que sejam todas positivas e somadas a 1. Nesta etapa, a pontuação de Autoatenção pode ser vista como uma porcentagem do foco total que é dado a um token na sequência, na codificação do token atual.

Partes de sequência (palavras, em um exemplo de sequência textual) que tiveram um alto valor de softmax na etapa anterior devem contribuir mais fortemente para a codificação do token atual, enquanto partes sequenciais com uma pontuação baixa devem contribuir muito pouco. Para fazer isso, cada vetor de valor é multiplicado por sua pontuação softmax. Isso mantém o valor original intacto, mas dimensiona o vetor geral de acordo com sua importância relativa para o token atual. Finalmente, todos os valores em escala são somados para produzir a codificação do token atual.

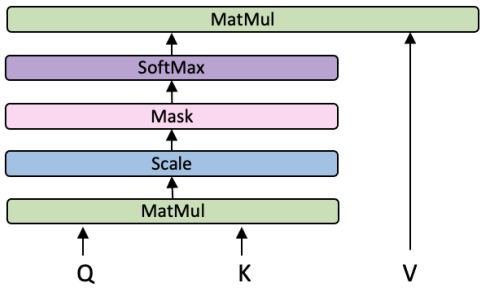


Figura 3.23: *Representação gráfica dos cálculos do mecanismo Self-Attention*

Considerando a aplicação em Séries Temporais há também outro desafio que precisa ser enfrentado. A série temporal não é processada sequencialmente; assim, o Transformer não aprenderá inerentemente dependências temporais. Para combater isso, as informações posicionais para cada token devem ser lidas. Ao fazer isso, o bloco de autoatenção terá contexto de distância relativa entre um determinado registro de tempo e o atual, como um importante indicador de relevância na autoatenção. A entrada(input) para o Transformer é uma determinada série temporal (univariada ou multivariada). O alvo é então a sequência deslocada uma vez para a direita, descrita em azul na Figura 3.23. Dado que as três etapas consistem em operações de matrizes, podem ser representadas da seguinte forma:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.30}$$

Para representar isso em uma sequência de comprimento cinco, para a primeira entrada x_1, o modelo produzirá sua previsão para o próximo token: $x_2^{'}$. Em seguida, ele recebe os verdadeiros x_1 e x_2 e prevê $x_3^{'}$, e assim por diante. A cada nova etapa, ele recebe todas as entradas verdadeiras antes da sequência, para prever a próxima etapa. Assim, o vetor de saída do modelo serão os tokens previstos $x_2^{'}, x_3^{'}, x_4^{'}, x_5^{'}, x_6^{'}$. Isso é então comparado aos valores reais $x_2, x_3, x_4, x_5 x_6$ para treinar o modelo, em que cada token de saída contribui igualmente para a perda.

De forma resumida, o mecanismo de Auto-atenção é semelhante ao mecanismo de Atenção, de modo geral eles compartilham o mesmo conceito e muitas operações matemáticas em comum. O método de auto-atenção recebe um conjunto de n entradas e retorna n

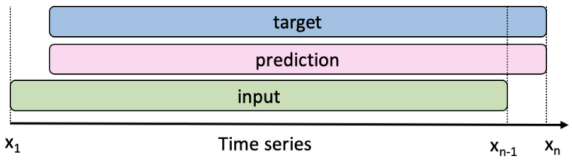


Figura 3.24: *Estrutura de entrada(input) e predição Transformer*

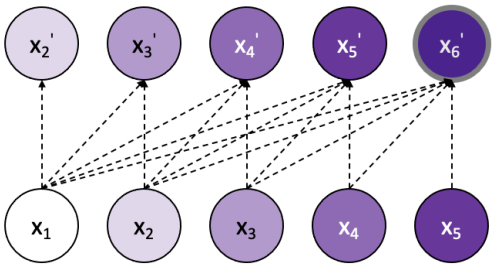


Figura 3.25: *Representação gráfica da utilização dos inputs na geração das previsões(outputs)*

saídas, o mecanismo permite interação entra as entradas com intuito de descobrir dada uma saída em qual entrada deve prestar mais Atenção (Attention). Note que apesar de permitir mapeamento temporal a arquitetura Transformer com mecanismo Atention ou Self-Attention não são considerados redes neurais recorrentes, portanto não ficam limitado ao processamento sequencial das RNNs, além de obter boa performance frente ao tempo de processamento/treinamento do algoritmo.

3.2.6 Modelos Híbridos de Deep Learning para séries temporais

Os algoritmos de Deep Learning são estruturas bastante flexíveis e permitem a combinação de diferentes camadas em um mesmo algoritmo. Nesta sessão são abordadas algumas das diversas combinações dos modelos híbridos existentes, a saber, ConvLSTM (combinação da estrutura de convolução e LSTM), modelo LSTM Bidirecional (bidirecional), Encoder-Decoder LSTM, Stacked LSTM e Dense LSTM. Grande maioria das combinações descritas nesta sessão são consideradas na etapa de aplicação das estruturas no capítulo 4.

ConvLSTM

Uma rede neural convolucional, ou CNN, é um tipo de rede neural desenvolvida para trabalhar majoritariamente com dados de imagens bidimensionais. A CNN pode ser muito eficaz em extrair e aprender recursos automaticamente de dados de sequência unidimensional, como dados de série temporal univariada. Um modelo CNN pode ser usado em um modelo híbrido com um backend LSTM, onde o CNN é usado para interpretar subsequências de entrada que, juntas, são fornecidas como uma sequência para um modelo LSTM interpretar. Este modelo híbrido é denominado CNN-LSTM.

A teoria por trás da ConvLSTM

Dados coletados sequencialmente ao longo do tempo são caracterizados como uma série temporal. Nestes casos, uma boa abordagem é usar um modelo baseado em LSTM, pois desta forma, o modelo consegue levar em conta dados dos passados para tomar decisões no presente. Conforme descrito na sessão 3.3.3 (veja figura 3.26 que representa a estrutura LSTM)

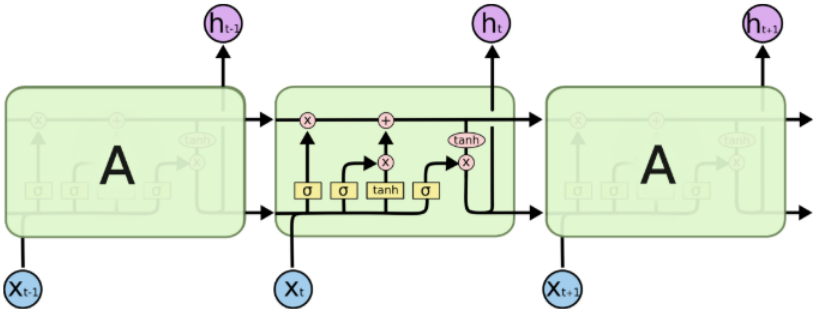


Figura 3.26: *Um neurônio de uma rede LSTM*

Quando se trata de imagens, comumente é utilizado a arquitetura CNN (Convolutional Neural Network). Nela, a imagem (matriz de pixels) passa por camadas de convolução, onde filtros extraem informações importantes da imagem, que por fim se conectam a uma rede Densa totalmente conectada de várias camadas (descrito na sessão 3.3.1).

No caso de imagens sequenciais ou matrizes sequenciais, podemos utilizar camadas ConvLSTM. É uma camada recorrente, assim como a LSTM, mas todas as operações de multiplicação interna de matrizes que aconteceria em uma LSTM são substituídas por convoluções. Isto faz com que os dados que fluem pela célula sejam da forma 3D, e não apenas um vetor 1D, como em uma simples LSTM. No caso de séries temporais é necessário adaptar os dados para o formato adequado para que este tipo de estrutura seja aplicado. É importante ressaltar que a sequência de informações ordenadas no tempo será mantida e o ajuste se restringe apenas a questão estrutural dos dados.

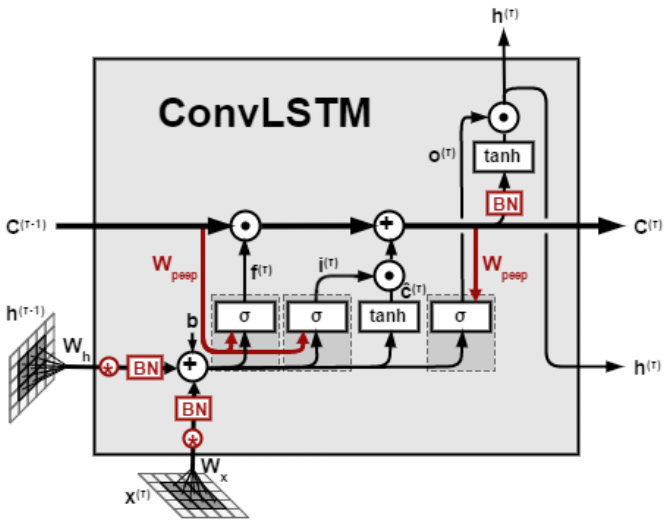


Figura 3.27: *Estrutura ConvLSTM*

Diferente de uma ConvLSTM, é um modelo Convolução-LSTM, isto é, acoplar na estrutura do algoritmo uma camada de convolução e na sequência uma camada LSTM, a entrada de dados primeiro passaria por camadas de convolução. O resultado desta convolução seria transformado para 1D com as características obtidas. Ao fazer isto para os dados no tempo, o resultado é um conjunto de características ao longo do tempo e esta seria a entrada da camada LSTM. A ConvLSTM é muito similar a estrutura LSTM, permite também dados com dependência temporal além de adicionar etapas de convolução dentro do mecanismo interno da célula recorrente. Esta variante de estrutura será aplicada e avaliada no Capítulo 4 em alguns dos cenários propostos.

Bidirectional LSTM

Para situações em que todos os passos de tempo da sequência de entrada estão disponíveis, os LSTMs bidirecionais treinam duas redes em vez de apenas uma, nos dados de input. A primeira rede é uma LSTM convencional já a segunda rede é uma LSTM com dados de input invertidos (do final para o começo). Isso pode fornecer contexto adicional para a rede e resultar em um aprendizado mais rápido e mais completo do problema em análise.

LSTM em seu núcleo, preserva informações de entradas que já passaram usando o estado oculto. O LSTM unidirecional apenas preserva as informações do passado porque as únicas entradas conhecidas são do passado. O uso de bidirecional executará as entradas de duas maneiras, uma do passado para o futuro e outra do futuro para o passado. O que difere essa abordagem da unidirecional é que no LSTM que funciona de modo reverso preserva as informações do futuro, usando os dois estados ocultos combinados são capazes de, a qualquer momento, preservar informações do passado e do futuro.

Estas estruturas são adequadas para cenários complexos onde é necessário adequar o entendimento do contexto da análise em questão. Importante enfatizar que além da estrutura proposta é necessário ajustar os parâmetros e os Hiperparâmetros (abordados na sessão 3.4) para que a rede Bidirecional obtenha bom ajuste. O fato de trabalhar com duas redes simultaneamente não garante a superioridade de performance das redes bidirecionais (BiLSTMs) frente as demais variações de estruturas existentes, cada cenário de análise irá requerer testes empíricos para esta averiguação.

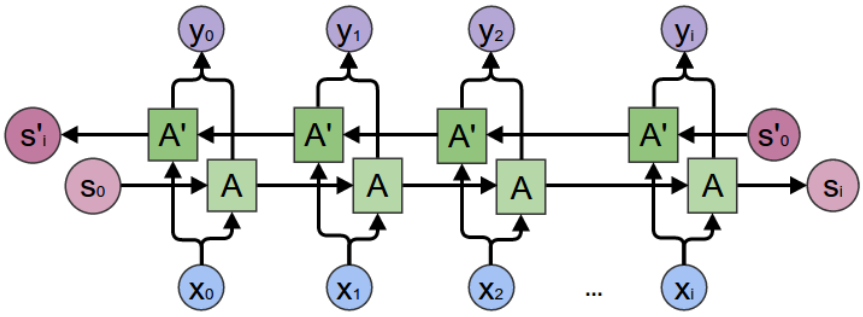


Figura 3.28: *LSTM Bidirecional*

Encoder-Decoder LSTM

Um modelo desenvolvido especificamente para prever sequências de saída de compri-

mento variável é chamado de Codificador-Decodificador LSTM. O modelo foi projetado para problemas de previsão em que existem sequências de entrada e saída, os chamados problemas sequência a sequência ou seq2seq, como a tradução de texto de um idioma para outro. Este modelo pode ser usado para previsão de séries temporais em várias etapas. Como o próprio nome sugere, o modelo é composto por dois submodelos: o codificador e o decodificador.

O codificador é um modelo responsável por ler e interpretar a sequência de entrada. A saída do codificador é um vetor de comprimento x que representa a interpretação do modelo da sequência. O codificador é tradicionalmente um modelo LSTM convencional, embora outros modelos de codificador possam ser usados, como os modelos Stacked, Bidirecional e CNN.

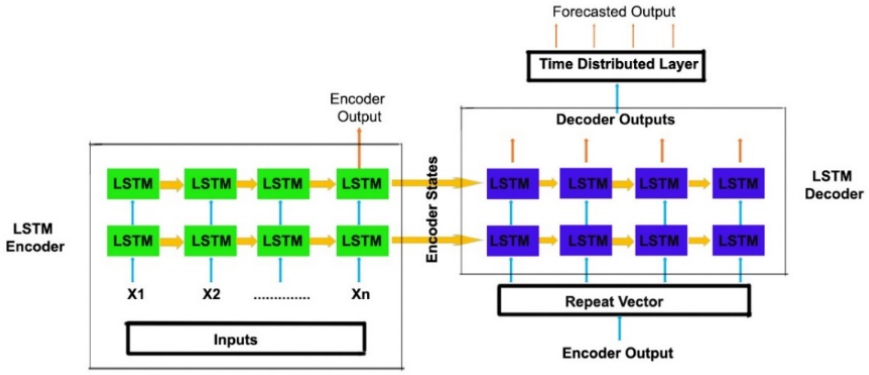


Figura 3.29: *LSTM Codificada*

Stacked LSTM

Múltiplas camadas LSTM ocultas podem ser empilhadas uma em cima da outra no que é conhecido como modelo LSTM Empilhado. Uma camada LSTM requer uma entrada tridimensional e LSTMs por padrão produzirão uma saída bidimensional como uma interpretação do final da sequência. Podemos resolver isso fazendo com que o LSTM produza um valor para cada intervalo de tempo nos dados de entrada, definindo que as sequências de retorno tenham argumento True na camada. Isso nos permite ter uma saída 3D da camada LSTM oculta como entrada para a próxima. Podemos, portanto, definir um LSTM Empilhado com número arbitrário de camadas.

Dense LSTM

Em qualquer rede neural, uma camada densa é aquela que está profundamente conectada à camada anterior, o que significa que os neurônios da camada estão conectados a todos os neurônios da camada anterior. É possível combinar camadas Densas com camadas resultando em um modelo Dense LSTM (Figura 3.31). A camada Densa é muito comum em diversos tipos de arquiteturas de Redes neurais e são flexíveis a ponto de permitirem serem combinadas em diferentes cenários. A flexibilidade está justamente na estrutura dos dados que precisam estar de acordo com a quantidade de dados de entrada, parâmetros da camada Densa e da camada LSTM.

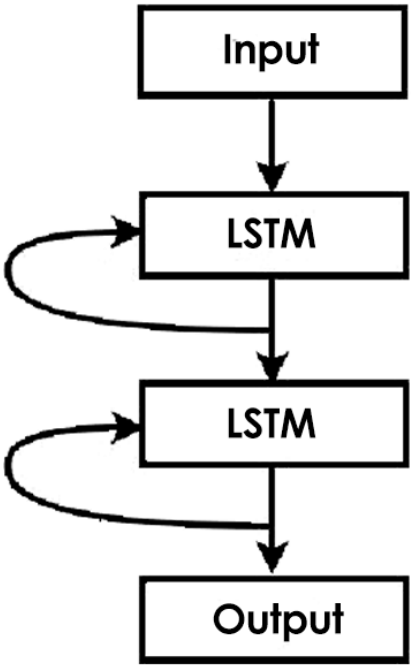


Figura 3.30: *LSTM Empilhada(Stacked)*

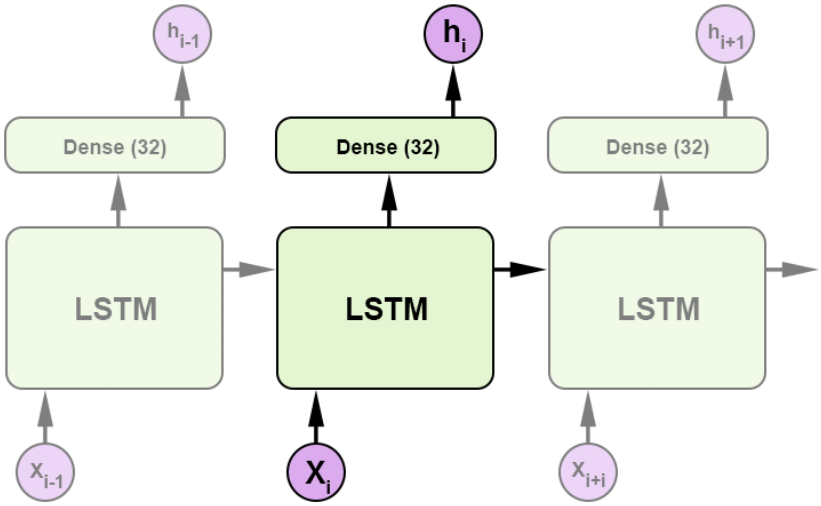


Figura 3.31: *Dense LSTM*

3.3 Parâmetros e Hiperpâmetros

As estruturas de Deep Learning têm em sua composição os convencionais parâmetros (bastante conhecidos dos algoritmos supervisionados clássicos), assim como os Hiperparâmetros que exercem controle sobre o processo de treinamento. Alguns Hiperparâmetros foram descritos nas sessões anteriores (como Learning Rate, por exemplo) outros serão descritos nesta sessão. O entendimento destes componentes é importante nos resultados que serão obtidos na aplicação dessas estruturas para séries temporais.

Para compreender os Hiperparâmetros é importante a descrição também dos parâmetros dos algoritmos de Deep Learning, segue breve descrição de ambos:

- Os **parâmetros** do modelo são as variáveis que a técnica de machine learning escolhida usa para ajustar os dados. Por exemplo, uma rede neural profunda (RNP) é composta por nós de processamento (neurônios), cada um com uma operação realizada nos dados enquanto trafegam pela rede. Quando a RNA é treinada, cada nó tem um valor de peso que informa ao modelo o impacto que tem na predição final. Esses pesos são um exemplo dos parâmetros do modelo. De muitas formas, esses parâmetros são o modelo. Ou seja, são eles que diferenciam um modelo específico de outros modelos do mesmo tipo que trabalham com dados semelhantes.

- Os **hiperparâmetros** são variáveis que controlam o próprio processo de treinamento do algoritmo. Por exemplo, faz parte da configuração de uma rede neural profunda decidir quantas camadas ocultas de nós precisam ser usadas entre a camada de entrada e a camada de saída, bem como quantos nós cada camada utilizará. Essas variáveis não estão diretamente relacionadas aos dados de treinamento, são variáveis de configuração. Os parâmetros mudam durante o processo de treinamento, enquanto os hiperparâmetros geralmente permanecem constantes durante o processo ou são otimizados através de Grid Search (busca exaustiva em um conjunto de opções previamente descrito).

Os parâmetros do modelo são otimizados (ou seja, "ajustados") pelo processo de treinamento. Os dados são executados por meio das operações do modelo, a predição resultante é comparada com o valor real de cada instância de dados, a precisão é avaliada e o ajuste ocorre até encontrar os melhores valores. Os hiperparâmetros são ajustados por meio da execução de todo o processo de treinamento, a observação da precisão aprimora o ajuste. Nos dois casos, o algoritmo está sendo modificado em sua composição com intuito de encontrar a melhor combinação para lidar com os dados em análise.

O ajuste de hiperparâmetros realiza vários testes em um único processo de treinamento. Cada teste é uma execução completa de treinamento com valores para a seleção de hiperparâmetros, definidos dentro dos limites especificados. Quando o processo de treinamento é concluído, analisa-se um resumo de todos os testes realizados com os hiperparâmetros, junto com a configuração de valores mais eficazes de acordo com os critérios especificados.

Obter bom ajuste dos Hiperparâmetros pode ser uma tarefa complexa pois está relacionada com os parâmetros do algoritmo e intrinsecamente conectada aos dados em análise. Cada estrutura de Deep Learning pode ter variações na quantidade de parâmetros e

Hiperparâmetros, dada a flexibilidade das estruturas. Abaixo são listadas boas práticas que serão adotadas para obtenção dos Hiperparâmetros no Capítulo 4 (aplicação das estruturas de RNAs em séries temporais):

- Análise do Overfiting (sobreajuste), que acontece quando uma rede neural essencialmente "memoriza"os dados de treinamento. Overfitting significa obtenção de ótimo desempenho nos dados de treinamento, mas o modelo torna-se inútil para previsões futuras.
- Regularização, descrita na sessão 3.3.1, auxilia evitar o sobreajuste e otimizar os hiperparâmetros da RNA. Ocultar partes do algoritmo durante o treinamento força que a rede diminua dependência de certos neurónios específicos tornando ainda mais propício a generalização para novos casos, previsões pontuais.
- Considerar conjunto de dados de teste apartado dos dados de treinamento, com intuito de simular um cenário real de performance dos algoritmos. Um bom desempenho na base de teste é um forte indício de que o modelo é capaz de generalizar, no caso de séries temporais de realizar previsões futuras de modo assertivo;
- Parcimônia frente a quantidade de parâmetros da rede. Note que essas estruturas facilmente atingem milhares de parâmetros, neste cenário, é importante balizar com o tamanho da série em análise. Um número muito grande de parâmetros pode também acarretar problemas de estimação/otimização do algoritmo;
- Quanto maior a quantidade de informações torna-se mais eficiente o combate ao Overfiting, por este motivo alguns cenários utilizarão dados simulados para que haja abundância de dados disponíveis para os algoritmos, além do controle de características dos dados em análise.
- O treinamento será realizado em várias épocas (passagens completas do conjunto de dados pela estrutura do algoritmo). Esta abordagem contribui para o ajuste dos parâmetros e Hiperparâmetros no processo de otimização;
- Será considerada a normalização dos dados com intuito de avaliar o impacto no processo de otimização, assim como diferentes funções de perda serão consideradas para cada cenário.

Os principais Hiperparâmetros que serão considerados na aplicação das diferentes estruturas de Deep Learning em séries temporais são:

- **Nodes LSTM**: Os Nodes (nós em tradução literal) da arquitetura LSTM definem toda a dimensão da rede e consequentemente a quantidade de parâmetros. Os Nodes são definidos no momento de configuração da arquitetura da rede e representam o "Estado de Célula"C_{t-1}, assim como a dimensão da “Estado Oculto” H_{t-1};
- **Slide Window**: Este hiperparâmetro, também conhecido como "Janela Deslizante"representa o período no tempo que será utilizado para o aprendizado do algoritmo (sessão 4.1.1); Epochs: Na terminologia de Deep Learning
- **Epoch**(época em tradução literal): é a computação dos valores de saída (forward pass) assim como atualização dos parâmetros (backward pass) por todo o conjunto de treinamento;

- **Batch Size**: Tamanho do lote (Batch Size) é um termo usado que se refere ao número de exemplos de treinamento usados em uma iteração. O Batch Size pode ser uma das três opções: **batch mode**: onde o tamanho do lote é igual ao conjunto de dados total, tornando os valores de iteração e épocas equivalentes. **mini-batch mode**: onde o tamanho do lote é maior que um, mas menor que o tamanho total do conjunto de dados. Geralmente, um número que pode ser dividido no tamanho total do conjunto de dados. **stochastic mode**: onde o tamanho do lote é igual a um. Portanto, o gradiente e os parâmetros da rede neural são atualizados após cada amostra;

Alguns Hiperparâmetros são controlados diretamente através do algoritmo de otimização (Adam, sessão 3.6.2), como o "Learning Rate"(Taxa de aprendizado). Esta sessão destaca apenas os principais Hiperparâmetros do processo de ajuste dos modelos de Deep Learning, dada a flexibilidade das estruturas é possível propor novos Hiperparâmetros que contribuam para o controle do processo de treinamento dos algoritmos de Redes Neurais Artificiais.

3.4 Algoritmo de Otimização

3.4.1 Backpropagação no Tempo (BPTT)

Conforme descrito nas sessões anteriores, um dos objetivos das redes neurais recorrentes é classificar com precisão uma entrada sequencial (por exemplo, dada uma sequência numérica, prever o próximo elemento). Para isto, a retropropagação (backpropagation) do erro e o gradiente descendente são considerados para realizar a projeção de interesse.

A retropropagação em redes feedforward retrocede do erro final através das saídas, pesos e entradas de cada camada oculta, atribuindo a esses pesos a responsabilidade por uma parte do erro calculando suas derivadas parciais, ou a relação entre suas taxas de mudança. Essas derivações são então usadas por nossa regra de aprendizado, gradiente descendente, para ajustar os pesos para cima ou para baixo, qualquer que seja a direção que diminua o erro.

As redes recorrentes dependem de uma extensão da retropropagação, chamada Backpropagation Through Time, ou BPTT (retropropagação no tempo em tradução literal). O tempo, neste caso, é simplesmente expresso por uma série ordenada e bem definida de cálculos, ligando um passo de tempo ao seguinte, o que significa que toda a retropropagação precisa funcionar. Redes neurais, sejam elas recorrentes ou não, são exclusivamente funções compostas aninhadas da forma: $f(g(h(x)))$. A adição de um elemento de tempo apenas estende a série de funções para as quais calculamos derivadas com a regra da cadeia. Considere a equação básica de uma RNN descrita pelas Equações 3.31 e 3.32:

$$s_t = \tanh\left(Ux_t + Ws_{t-1}\right) \tag{3.31}$$

$$\widehat{y}_t = softmax\left(Vs_t\right) \tag{3.32}$$

Considere, neste contexto, que a função de perda é dada pela entropia cruzada, repre-

sentada pela Equação 3.33:

$$E_t\left(y_t,\ \widehat{y}_t\right) = -y_t \log \widehat{y}_t \ ; E = \ \sum_t E_t\ \left(y_t,\ \widehat{y}_t\ \right) = \ -\ \sum_t y_t\ \ \log \widehat{y}_t \tag{3.33}$$

Em que, y_t é o elemento correto no momento do passo t, e $\hat{y}_t$ é a previsão. Normalmente, a sequência completa é tratada como um exemplo de treinamento, portanto, o erro total é apenas a soma dos erros em cada etapa de tempo.

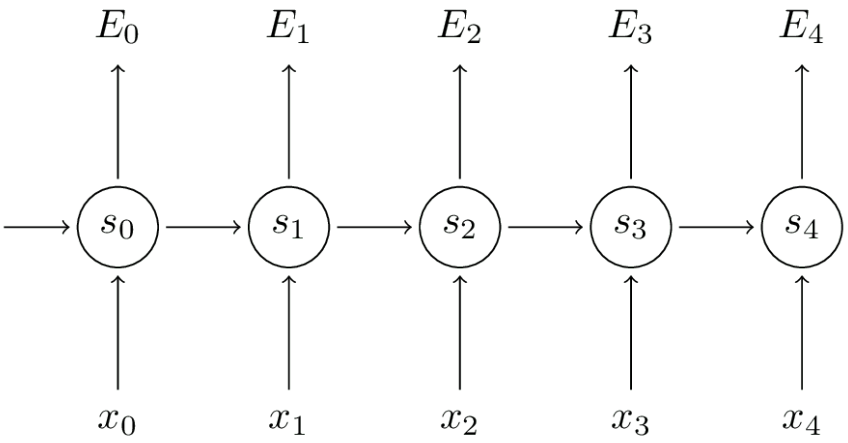


Figura 3.32: *Estrutura temporal de propagação*

Note que o intuito é calcular os gradientes do erro em relação aos parâmetros U, V e W e, em seguida, aprender bons parâmetros usando o Gradiente Descendente Estocástico. Assim como os erros serão resumidos, os gradientes também são somados em cada etapa de tempo para um exemplo de treinamento:

$$\frac{\partial E}{\partial W} = \ \sum_t \frac{\partial E_t}{\partial W} \tag{3.34}$$

Para calcular esses gradientes, a regra de diferenciação da cadeia é utilizada. Esse é o algoritmo de retropropagação quando aplicado para trás a partir do erro. O caso E_3 será usado como exemplo, apenas para trabalhar com números concretos. Em que $\otimes$ representa o produto tensorial.

$$\frac{\partial E_3}{\partial V} = \frac{\partial E_3}{\partial \widehat{y}_3}\ \frac{\partial \widehat{y}_3}{\partial V} = \ \frac{\partial E_3}{\partial \widehat{y}_3}\ \frac{\partial \widehat{y}_3}{\partial z_3}\ \frac{\partial z_3}{\partial V} = \left(\widehat{y}_3 - y_3\right)\ \otimes\ s_3 \tag{3.35}$$

Note a dependência apenas dos valores do momento atual,$\hat{y}_3$, y_3, s_3. Ao obter estes valores o cálculo do gradiente para V é uma multiplicação de matriz simples. No entanto o cenário é diferente para W (e para U):

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \widehat{y}_3} \frac{\partial \widehat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial W} \tag{3.36}$$

Agora, note que

$$s_3 = \tanh U \cdot x_t + W \cdot s_2$$

depende de s_2, que depende de W e s_1 e assim por diante. Então, ao considerar a derivada em relação à W, não é possível simplesmente tratar s_2 como uma constante, é preciso aplicar a regra da cadeia novamente para obter o seguinte:

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \widehat{y}_3} \frac{\partial \widehat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W} \tag{3.37}$$

A contribuição de cada passo no tempo é somada para o gradiente. Isto é, como W é usado em todas as etapas até a saída de interesse, faz-se necessário retroceder gradientes na rede de $t = 3$ até $t = 0$ (Figura 3.33):

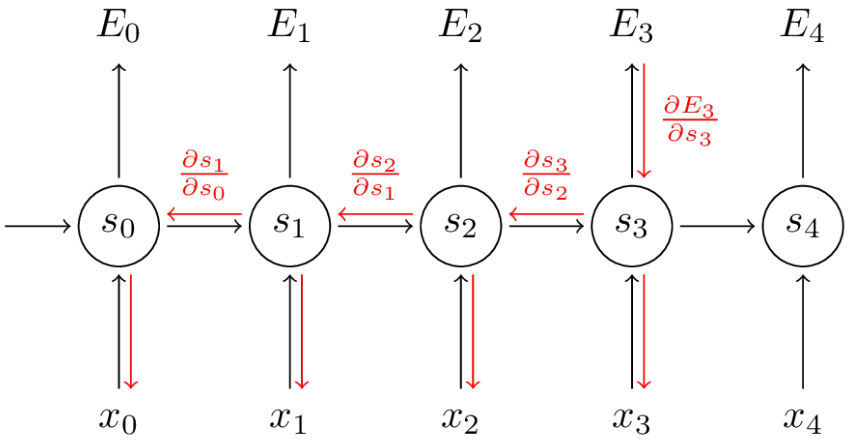


Figura 3.33: *Estrutura temporal de retropropagação (exemplo E_3)*

Observe que o contexto é exatamente o mesmo que o algoritmo de retropropagação padrão utilizado nas Redes Neurais Profundas Feedforward. A principal diferença é que os gradientes para W são resumidos em cada etapa de tempo. Em uma rede neural tradicional, não há o compartilhamento de parâmetros entre camadas, portanto, é preciso realizar a soma. O BPTT pode ser interpretado como retropropagação padrão em uma RNN "desenrolada" (unfoulded).

O conceito de Retropropagação no tempo está embutido nas Redes Neurais recorrentes e suas respectivas variações e derivações, LSTM e GRU por exemplo. Desta forma, as informações são propagadas através do tempo frente uma sequência (seja numérica ou textual), ao passo que o processamento se torna sequencial em cada unidade de tempo. Outras estruturas de Deep Learning que permitem certa dependência temporal, mas não são consideradas recorrentes utilizam outros mecanismos para mapear a dependência

temporal dos dados, como o mecanismo Attention adotado pela estrutura Transformers (Sessão 3.3.3), por exemplo.

3.4.2 Adam

Adam Goodfellow *et al.* (2015) é um algoritmo de otimização de taxa de aprendizagem (learning Rate) adaptável que foi projetado especificamente para treinar redes neurais profundas. Publicado pela primeira vez em 2014, Adam foi apresentado em uma conferência de grande prestígio para praticantes de aprendizagem profunda - ICLR 2015. O artigo continha alguns diagramas muito promissores, mostrando enormes ganhos de desempenho em termos de velocidade de treinamento. Muitas pesquisas foram realizadas para evoluir o algoritmo de otimização Adam em situações específicas.

Os algoritmos aproveitam o poder dos métodos de taxas de aprendizagem adaptativas para encontrar taxas de aprendizagem individuais para cada parâmetro. Também tem vantagens do Adagrad Duchi *et al.* (2011), que funciona muito bem em configurações com gradientes esparsos, mas tem dificuldades na otimização não convexa de redes neurais, e RMSprop, que trata de resolver alguns dos problemas do Adagrad e funciona muito bem em configurações on-line. A popularidade de Adam tem crescido exponencialmente, de acordo com o artigo "A Peek at Trends in Machine Learning"de Karpathy (2017) .

Adam pode ser visto como uma combinação de RMSprop e Stochastic Gradient Descent (SGD) com momento. Este algoritmo de otimização utiliza gradientes quadráticos para escalar a taxa de aprendizado como RMSprop para tirar vantagem do momento ao usar a média móvel do gradiente em vez do próprio gradiente como SGD. Adam usufrui da taxa de aprendizagem adaptativa, o que significa que calcula taxas de aprendizagem individuais para diferentes parâmetros. Seu nome é derivado de estimativa de momento adaptável, e a razão pela qual é chamado deste modo é porque usa estimativas de primeiro e segundo momentos do gradiente para adaptar a taxa de aprendizado para cada peso da rede neural.

O enésimo momento de uma variável aleatória é definido como o valor esperado dessa variável à potência de n (veja equação 3.38).

$$m_n = \mathbb{P}\left[X^n\right] \tag{3.38}$$

Observe que o gradiente da função de custo da rede neural pode ser considerado uma variável aleatória, uma vez que geralmente é avaliada em algum pequeno lote aleatório de dados. O primeiro momento é a média e o segundo momento é a variância não centrada (o que significa que não é subtraída a média durante o cálculo da variância). Para estimar os momentos, Adam utiliza médias móveis exponencialmente, calculadas no gradiente avaliado em um minilote atual:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\, g_t \tag{3.39}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\, g_t^2 \tag{3.40}$$

Onde m e v são médias móveis, g é o gradiente no minilote atual e betas - novos hiperparâmetros introduzidos do algoritmo. Os vetores de médias móveis são inicializados com zeros na primeira iteração. Para compreensão do relacionamento dos valores e momento, definido como na primeira equação, observa-se os valores esperados de nossas médias móveis. Uma vez que m e v são estimativas do primeiro e do segundo momentos, nota-se a seguinte propriedade:

$$\mathbb{P}\left[m_t\right] = \mathbb{P}\left[g_t\right] \tag{3.41}$$

$$\mathbb{P}\left[v_t\right] = \mathbb{P}\left[g_t^2\right] \tag{3.42}$$

Os valores esperados dos estimadores devem ser iguais ao parâmetro que está sendo estimado, de fato, o parâmetro neste caso também é o valor esperado. Se essas propriedades forem verdadeiras, isso significa estimadores imparciais. (Para saber mais sobre as propriedades estatísticas de diferentes estimadores, consulte o livro Deep Learning de Yoshua Bengio *et al.* (2015) , Capítulo 5). No entanto, não se aplica às médias móveis descritas anteriormente, pois foram inicializadas com zeros, os estimadores são tendenciosos para zero, será provado para m (a prova para v é análoga). É necessário formular para m até o primeiro gradiente e, então, desdobrar alguns valores de m para ver o padrão a utilizar:

$$m_0 = 0 \tag{3.43}$$

$$m_1 = \beta_1 m_0 + (1 - \beta_1)\, g_1 = (1 - \beta_1)\, g_1 \tag{3.44}$$

$$m_2 = \beta_1 m_1 + (1 - \beta_1)\, g_2 = \beta_1 (1 - \beta_1)\, g_1 + (1 - \beta_1)\, g_2 \tag{3.45}$$

$$m_3 = \beta_1 m_2 + (1 - \beta_1)\, g_3 = \beta_1^2 (1 - \beta_1)\, g_1 + \beta_1 (1 - \beta_1)\, g_2 + (1 - \beta_1)\, g_3 \tag{3.46}$$

Note que ao expandir o valor de m, menor a contribuição dos primeiros valores dos gradientes para o valor geral, à medida que são multiplicados por betas cada vez menor. Com este padrão é possível reescrever a média móvel da seguinte forma:

$$m_t = (1 - \beta_1) \sum_{i=0}^{t} \beta_1^{t-i} g_i \tag{3.47}$$

Observe o valor esperado de m com o primeiro momento verdadeiro, para que seja proposta uma correção na discrepância entre ambos:

$$\mathbb{P}\left[m_t\right] = \mathbb{P}\left[(1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}g_i\right] = \mathbb{P}\left[g_i\right](1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i} + \xi \tag{3.48}$$

$$\mathbb{P}\left[m_t\right] = \mathbb{P}\left[g_i\right]\left(1-\beta_1^t\right) + \xi \tag{3.49}$$

Na Equação 3.48, a fórmula de expansão de m através da média móvel é reescrita. Em seguida, aproximamos g_i com g_t. Como a aproximação está ocorrendo, o erro C surge na fórmula. Na Equação 3.49, é utilizado apenas a fórmula para a soma de uma série geométrica finita. Há duas características importantes a serem destacadas:

- O estimador é não enviesado. Isso não é verdade apenas para Adam, o mesmo vale para algoritmos usando médias móveis (SGD com momento, RMSprop, etc.).
- Dado que o valor beta revelado a potência t está indo rapidamente para zero, não haverá muito efeito, a menos que seja o início do treinamento.

É preciso realizar uma correção no estimador para que o valor esperado seja conforme desejado. Esta etapa é denominada de "Correção de viés":

$$\widehat{m}_t = \left(\frac{m_t}{1-\beta_1^t}\right) \tag{3.50}$$

$$\widehat{v_t} = \left(\frac{v_t}{1-\beta_2^t}\right) \tag{3.51}$$

Por fim, as medias móveis serão utilizadas para dimensionar a taxa de aprendizagem individualmente para cada parâmetro. Adam realiza a atualização da seguinte forma:

$$w_t = w_{t-1} - \eta\frac{\widehat{m}_t}{\sqrt{\widehat{v_t}+\varepsilon}} \tag{3.52}$$

Algumas propriedades do Adam são muito importantes e contribuem significativamente para seu desempenho:

- tamanho real do passo obtido pelo Adam em cada iteração é aproximadamente limitado pelo hiperparâmetro do tamanho do passo. Esta propriedade adiciona compreensão intuitiva ao hiperparâmetro anterior da taxa de aprendizagem não intuitiva.
- tamanho do passo da regra de atualização do Adam é invariante para a magnitude do gradiente, o que ajuda muito ao passar por áreas com gradientes minúsculos. Nessas áreas, a SGD se esforça para navegar rapidamente por elas.
- Adam foi projetado para combinar as vantagens do Adagrad, que funciona bem com gradientes esparsos, e RMSprop, que funciona bem em configurações on-line. Ter ambos permite usar Adam para uma gama mais ampla de tarefas. Adam também pode ser visto como a combinação de RMSprop e SGD com momento.

Destaca-se os parâmetros padrão que são considerados no otimizador Adam para todas as aplicações das estruturas do Capítulo 4:

- **learning rate**: taxa de aprendizado, valor padrão utilizado neste parâmetro é 0,001.
- **beta1**: Um valor flutuante. A taxa de decaimento exponencial para as estimativas do primeiro momento. O padrão é 0,9.
- **beta2**: Um valor flutuante. A taxa de decaimento exponencial para as estimativas do segundo momento. O padrão é 0,999.
- ϵ : Uma pequena constante para estabilidade numérica. O padrão é 1e-7.
- **amsgrad**: Booleano. Determina se deve ocorrer a aplicar da variante AMSGrad deste algoritmo. O padrão é False.
- **kwargs**: Tem permissão para ser "clipnorm"ou "clipvalue". "clipnorm"(float) recorta gradientes por norma; "clipvalue"(float) recorta gradientes por valor.

Para maiores detalhes sobre as propriedades veja o artigo Adam Bushaev (2018). Adam é definitivamente um dos melhores algoritmos de otimização para aprendizado profundo e sua popularidade é crescente. Embora existam limitações com o uso de Adam em certas áreas especificas, pesquisas continuam sedo desenvolvidas para encontrar soluções e adaptações para que os resultados de Adam sejam cada vez mais aprimorados. No Capítulo 4 Adam será o algoritmo de otimização utilizado na aplicação dos modelos de Deep Learning para series temporais frente a diferentes arquiteturas, conjuntos de dados e cenários de aplicação.

Capítulo 4

Aplicação de Deep Learning para Séries Temporais

O crescente destaque das redes neurais profundas, em especial as arquiteturas como Long Short-Term Memory (LSTM) e Recurrent Neural Networks (RNN), reforça a relevância dessa aplicação. Embora o Deep Learning tenha conquistado resultados notáveis em imagens, vídeo e texto, sua aplicação específica em séries temporais demanda uma análise mais aprofundada. Este capítulo se propõe a mergulhar nesse cenário, explorando como as estruturas flexíveis dos algoritmos de Deep Learning podem ser adaptadas para lidar com a dinâmica temporal inerente a conjuntos de dados sequenciais.

A compreensão profunda do vocabulário, das notações e das definições dos algoritmos de Deep Learning é essencial para garantir que a aplicação em séries temporais seja efetiva. Além disso, a preparação específica dos dados temporais para se adequarem a essas estruturas é um ponto crucial, destacando a necessidade de abordagens específicas para diferentes classes de algoritmos. Dessa forma, este capítulo se propõe não apenas a aplicar o Deep Learning a séries temporais, mas também a estabelecer uma base sólida de entendimento para as etapas subsequentes da análise.

Este capítulo explora diferentes cenários para que seja observado o comportamento dos algoritmos em contextos distintos entre si. Casos Multivariados também serão explorados pois as arquiteturas são flexíveis e permitem essa aplicação. Para uma mesma série algumas variações de modelo serão aplicadas, tendo como Baseline (base de comparação) os modelos ARIMA, descrito no Capítulo 2.

4.1 Preparação dos Dados

No âmbito da aplicação de modelos de Deep Learning, em particular Long Short-Term Memory (LSTM), à análise de séries temporais, a preparação dos dados emerge como um elemento crítico para o sucesso da modelagem. Esta Sessão se dedica a uma exploração detalhada da fase de "Preparação dos Dados", reconhecendo a singularidade desse processo em comparação com abordagens convencionais de séries temporais. Ao contrário de métodos tradicionais, onde a sequencialidade dos dados é muitas vezes tratada

de forma linear, a natureza recorrente dos modelos LSTM demanda uma atenção especial às características temporais intrínsecas dos conjuntos de dados.

A preparação específica dos dados para modelos LSTM abrange diversas nuances, desde a formatação adequada dos conjuntos temporais até a consideração de janelas temporais e a adequação dos dados para o fluxo de dependência temporal. A sessão delineará as etapas necessárias para alinhar os dados com as exigências estruturais dos modelos LSTM, assegurando que a informação temporal seja adequadamente incorporada no processo de treinamento. Ademais, serão abordadas estratégias para lidar com diferentes granularidades temporais e características específicas dos dados, reconhecendo a flexibilidade inerente aos modelos LSTM, que permite adaptações às complexidades dos cenários em estudo.

A compreensão dessas etapas de preparação é fundamental para explorar plenamente o potencial dos modelos LSTM em séries temporais. Este entendimento permite estabelecer um alicerce sólido para os estágios subsequentes da aplicação, proporcionando insights cruciais sobre a manipulação eficaz de dados temporais e a maximização do desempenho dos modelos propostos.

4.1.1 Sliding Window (Janela Deslizante)

Grande parte das aplicações de Machine Learning têm como fundamento o aprendizado supervisionado. A aprendizagem supervisionada no caso univariado utiliza variáveis de entrada x para identificar os padrões da variável de saída y em que um algoritmo é utilizado para aprender a função de mapeamento da entrada para a saída. O objetivo é aproximar o mapeamento real subjacente tão bem que quando houver novos dados de entrada x, será possível prever a variável de saída(target) y para esses dados.

$$y = f(x) \tag{4.1}$$

É denominado aprendizagem supervisionada porque o processo de aprendizagem do algoritmo tem como base um treinamento, em que o conjunto de dados pode ser pensado como um processo de aprendizagem. Neste contexto; o algoritmo faz previsões iterativamente sobre os dados de treinamento e é corrigido fazendo atualizações e comparações com os valores reais de interesse. O aprendizado é interrompido quando o algoritmo atinge um nível aceitável de desempenho. Problemas de aprendizagem supervisionada podem ser agrupados em duas grandes vertentes de regressão e classificação.

- **Classificação**: problema de classificação é quando a variável de saída y é uma categoria, tal como vermelho e azul ou possui característica de interesse e não possui característica de interesse.
- **Regressão**: problema de regressão é quando a variável de saída y é um valor real, faturamento de uma empresa, por exemplo.

A aplicação de algoritmos de Deep Learning em séries temporais requer preparação específica na estrutura dos dados, isto é, o problema de Séries temporais precisa ser adaptado em um problema de Regressão/Classificação no que diz respeito a estrutura das informações de análise. Note que este é o mesmo procedimento adotado por modelos da Classe ARIMA,

por exemplo, mas esta preparação na estrutura dos dados é feita automaticamente pelo algoritmo no momento da estimação, caso similar ocorre para a grande maioria das classes de algoritmos convencionais.

Os dados de séries temporais podem ser expressos como aprendizado supervisionado. Dada uma sequência de números ordenados no tempo, conjunto de dados da série, é possível reestruturar os dados para se tornarem um problema de aprendizado supervisionado. Este procedimento é realizado através da utilização de etapas de tempo anteriores como variáveis de entrada, ao passo que a próxima etapa de tempo é considerada como a variável de saída. O Exemplo abaixo ilustra de forma didática a transformação(adaptação) de uma série temporal em um problema de dados supervisionado.

Tempo	Medida
1	100
2	110
3	108
4	115
5	120

Figura 4.1: *Série Temporal*

O conjunto de dados da Figura 4.1 pode ser reestruturado como um problema de aprendizado supervisionado usando o valor na etapa de tempo anterior para prever o valor na etapa de tempo seguinte. Reorganizando o conjunto de dados de séries temporais dessa forma, os dados são dispostos da seguinte forma (Figura 4.2):

Entrada(X)	Saída(Y)
?	100
100	110
110	108
108	115
115	120
120	?

Figura 4.2: *Série temporal com Slide Window de um período (r=1)*

A Figura 4.3 descreve o procedimento de Sliding Window(Janela deslizante, tradução literal) com uma janela de tamanho w escolhida (e fixa), que neste exemplo é 4. Isso significa que, o modelo vai mapear as informações contidas nessas janelas, com a previsão no ponto que está em t+1. Há um r no tamanho da resposta, porque é possível prever várias etapas de tempo no passado. Esse seria um relacionamento de muitos para muitos. Para simplificar e facilitar a visualização, no exemplo da Figura 4.2 é considerado r=1.

Ao ajustar o conjunto de dados de séries temporais dessa forma, nota-se as seguintes características:

- A etapa anterior é a entrada x e a próxima etapa é o output y já considerando o problema como uma análise de aprendizado supervisionado.
- A ordem entre as observações é preservada, e deve continuar a ser preservada ao usar este conjunto de dados para treinar um modelo supervisionado.
- Não há nenhum valor anterior que possa ser usado para prever o primeiro valor em sequência. Geralmente está linha é desconsiderada pois não há como utilizá-la neste formato.
- Não há um próximo valor conhecido para prever o último valor na sequência. Este valor também será desconsiderado do treinamento supervisionado do modelo, análogo ao caso anterior.

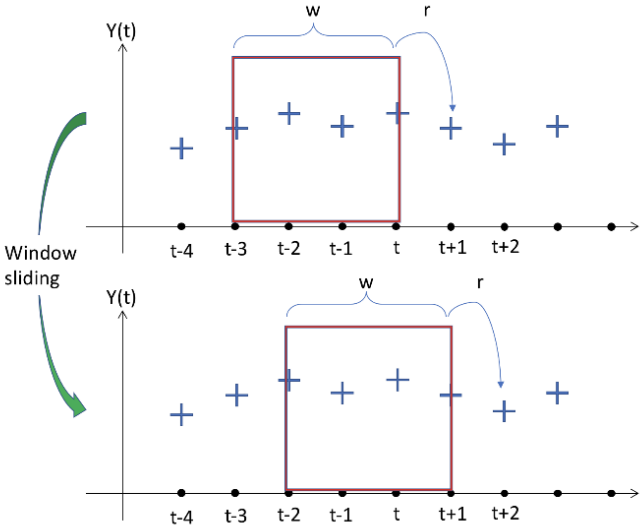


Figura 4.3: *Sliding Window*

Este método pode ser chamado de "método da janela" em algumas literaturas. Em estatística e em análise de séries temporais, é também denominado de método de latência ou latência. O número de etapas anteriores é chamado de largura da janela ou tamanho do atraso. Esta janela deslizante é a base de como adaptar conjunto de dados de séries temporais em um problema de aprendizado supervisionado. A partir deste exemplo simples, nota-se que:

- Sliding Window pode ser aplicado para transformar uma série temporal em uma regressão ou uma classificação;
- Uma vez que um conjunto de dados de séries temporais é preparado desta forma qualquer algoritmo de aprendizado de máquina linear e não linear pode ser aplicado, desde que a ordem das linhas seja preservada.
- A largura da janela deslizante pode ser aumentada para incluir histórico maior de informações no momento de aprendizado.

- A abordagem da janela deslizante pode ser usada também nas séries temporais multivariadas, o procedimento é análogo.

Devido a existência de dependência temporal entre as observações de uma série, instantes passados influenciam instantes futuros. Quando é desejado prever o valor em um instante t de uma série temporal, utiliza-se somente as informações contidas nos instantes anteriores a t. É possível imaginar a utilização de todos os valores da série anteriores ao instante t, porém é mais sensato utilizar uma janela de valores imediatamente anteriores(lags) a t pois estes normalmente apresentam uma maior autocorrelação com o instante que queremos prever além de reduzir o tempo de treino do modelo.

A Figura 4.4 esquematiza a utilização de 4 lags para a previsão(forecast) do próximo instante (similar ao caso apresentado na Figura 4.2):

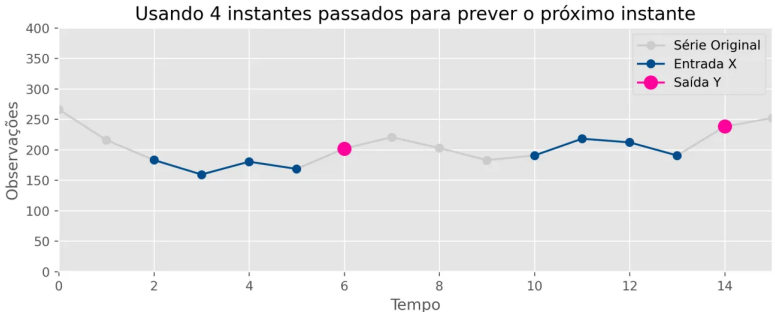


Figura 4.4: *Slide Window de janela de 4 períodos*

Neste exemplo, há uma série temporal S com duas instâncias de treinamento possíveis indicadas, sendo elas pares (X, Y) de vetores X_i de tamanho 4 e Y_i de um único escalar (Equações 55 e 56)

$$X_0 = [S_2,\ S_3,\ S_4, S_5]\ e\ Y_0 = [S_6] \tag{4.2}$$

$$X_1 = [S_{10}, S_{11}, S_{12}, S_{13}]\ \ e\ \ Y_1 = [S_{14}] \tag{4.3}$$

A técnica de Janela deslizante consiste em percorrer os valores da série criando pares (X, Y) onde X consiste em uma janela de r lags a um instante t e Y consiste no instante t em questão (Figura 4.5):

Desta forma é possível reestruturar uma sequência de valores em pares (X, Y) de forma a utilizar os valores passados de um instante t como entrada X para o modelo e o instante t que queremos prever como saída desejada Y. O Método Sliding Window será utilizado para todas as aplicações das diferentes estruturas de deep Learning para séries temporais desta dissertação. Esta etapa de preparação dos dados é importante para que seja possível ajustar e realizar previsões com as estruturas de RNAs que permitem dependência ao longo do tempo.

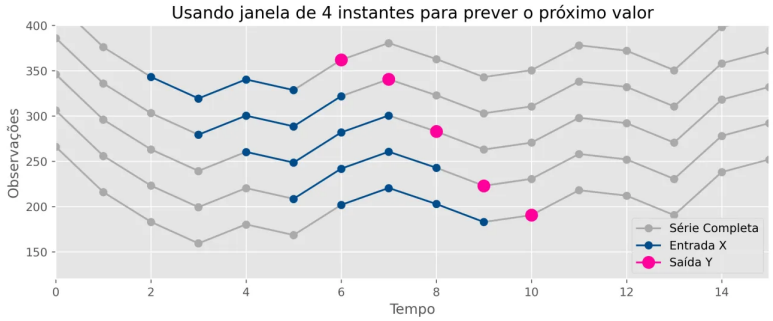


Figura 4.5: *Janela deslizante com 4 instantes*

4.1.2 Reorganização (Reshape) dos Dados

Os dados da série temporal devem ser transformados antes que possam ser usados em um modelo de Machine Learning. Desta forma, os dados podem ser usados imediatamente por um algoritmo de aprendizado de máquina supervisionado e até mesmo Deep Learning (Rede Neural). Uma outra transformação adicional é necessária para preparar os dados para aplicação de uma Rede Neural Long Short-Term(LSTM). Especificamente, a estrutura bidimensional (Tabela 1) dos dados supervisionados de aprendizagem deve ser transformada em uma estrutura tridimensional para ser compatível com a estrutura dos algoritmos de aprendizagem profunda.

A camada de entrada para modelos CNN e LSTM é especificada pelo argumento de forma de entrada (input shape argument) na primeira camada oculta da rede. A entrada para cada camada CNN e LSTM deve ser tridimensional, as três dimensões desta entrada são:

- **Samples(Amostras)**: Uma amostra pode referir-se a exemplos de treinamento individual. Uma variável "batch size" é, portanto, a contagem de amostras enviadas para a rede neural. Ou seja, quantos exemplos diferentes são fornecidos de uma única vez para a rede neural.
- **Time Steps(Passos de tempo)**: são intervalos de tempo (ticks/lags no tempo). É a duração de cada uma de suas amostras. Para séries temporais um intervalo de tempo é considerado um lag (descolamento no tempo de uma unidade).
- **Features(Séries)** : são simplesmente o número de dimensões que alimentam o algoritmo em cada etapa de tempo. No caso univariado feature será sempre igual a 1, ao passo que nos casos multivariados dependerá da quantidade de séries em análise.

De forma sintetizada, cada coluna representa uma feature do modelo e corresponde a uma variável. De modo que cada linha representa uma sample(amostra) e corresponde a um novo exemplo com input e output. Esta estrutura tridimensional esperada de dados de entrada é muitas vezes resumida usando a notação matricial da seguinte forma: samples, Time Steps e Features. Note que a dimensão dos conjuntos de dados de séries temporais é bidimensional da forma de Samples e Features (Figura 4.6). Isso significa que é necessário adicionar uma nova dimensão de Time Steps. Então, realmente, há a inclusão da dimensão de Time Steps, para que os dados se tornem tridimensionais atingindo a estrutura requerida

pelas arquiteturas dos algoritmos de Deep Learning).

Considere a seguinte sequência numérica (10, 20, 30, 40, 50, 60, 70, 80, 90, 100) que pode ser interpretada como uma série temporal ao longo do tempo, conforme Figura 4.6:

Tempo	Série
1	10
2	20
3	30
4	40
5	50
6	60
7	70
8	80
9	90
10	100

Figura 4.6: *Série temporal ilustrativa*

Ao transformar está série em um problema supervisionado considerando a adição da dimensão de Time Steps o resultado desta transformação segue descrito na Figura 4.7.Neste Exemplo há 7 samples (amostras completas), organizadas por 3 Time Steps(que representa o tamanho da Janela Deslizante - Sliding Window) cada (3 lags no tempo serão utilizados para o aprendizado) e 1 feature pois trata-se de uma série univariada. Note que as 3 primeiras observações de saída da série foram desconsideradas por não ser possível obter input para os primeiros casos (cenário similar ocorre na estimação dos modelos ARIMA). Neste cenário, essas amostras(samples) teriam dados faltantes(missing), portanto foram removidas da tabela de aplicação. Nas próximas sessões este processo de preparação dos dados será realizado para todas as séries que são contempladas nos diferentes cenários de aplicação.

4.2 Aplicação de Deep Learning para Séries Univariadas

Esta sessão visa explorar a aplicação dos algoritmos de Deep Learning no contexto de séries temporais para dados Univariados. Diferentes conjuntos de dados serão considerados com intuito de averiguar a performance dos modelos em diferentes contextos frente a diversas características dos dados. Alguns conjuntos de dados são bastante conhecidos e

Entrada(X)	Saída(Y)
[10 20 30]	40
[20 30 40]	50
[30 40 50]	60
[40 50 60]	70
[50 60 70]	80
[60 70 80]	90
[70 80 90]	100

Figura 4.7: *Série temporal com slide window com 3 períodos (r=3)*

experimentados dentro de cenários de aplicação de séries temporais. As análises visam a compreensão da adaptabilidade das estruturas diante das perspectivas consideradas. Segue a macro descrição dos respectivos conjuntos de dados:

- **Airline Passangers**: Conjunto de dados que descreve a quantidade de passagens aéreas nos Estados Unidos entre 1949 e 1960, com periodicidade mensal.
- **Montly beer Production**: Produção de Cervejas na Australia entre janeiro de 1956 a agosto de 1995, com periodicidade mensal.
- **GE Open**: Valor das ações da empresa GE no momento da Abertura da bolsa de valores entre o período de 04/11/2019 a 27/08/2021, com periodicidade diária;

Para cada conjunto de dados, diferentes arquiteturas serão consideradas. Uma arquitetura é um conjunto de camadas que compõem um algoritmo de Deep Learning. Os algoritmos mais simples possuem uma única estrutura de camadas, ao passo que os algoritmos mais complexos combinam diferentes estruturas de camadas. A descrição das camadas utilizadas na aplicação encontra-se descritas no Capítulo 3. Abaixo são listadas as camadas que formam os algoritmos que serão considerados na aplicação:

- **LSTM (Original)**
- **LSTM Bidirecional**
- **LSTM + Dense Layer**
- **ConvLSTM**
- **Stacked LSTM**

Importante destacar que para cada conjunto de dados univariados será ajustado um modelo de Classe ARIMA que seram o Baseline de comparação para os modelos de Deep Learning. Cada estrutura tem um conjunto de Hiperparâmetros que foram otimizados

através de Grid Search (busca através de valores pré-estipulados). A grande maioria dos Hiperparâmetros são comuns para as arquiteturas propostas (veja sessão 3.6.2 - Hiperpâmetros).

Para cada série temporal todas as arquiteturas propostas serão aplicadas e para cada arquitetura será otimizado os Hiperparâmetros do Grid de acordo com a descrição realizada. Testar uma grande quantidade de modelos (centenas para cada estrutura) permite compreender uma série de detalhes frente as diferentes abordagens e suas respectivas características, dado que os modelos de Deep Learning praticamente não têm premissas sobre os dados, apenas sobre sua estrutura/dimensão conforme descrito na sessão inicial do capítulo quatro.

Para cada etapa de otimização dos Hiperparâmetros será repetida 10 vezes e a média da métrica de comparação será considerada. Diferentes métricas serão consideradas (RSME, MAE e MAPE) frente as respetivas visões que permitem para os cenários de aplicação. MAPE será a métrica adotada para descrição dos resultados frente a performance dos diferentes modelos aplicados a base de teste para cada cenário.

4.2.1 Airline Passangers Dataset

O conjunto de dados de passagens aéreas nos Estados Unidos entre as décadas de 1950 e 1960 é uma das séries temporais mais conhecidas do universo acadêmico por ser utilizado em diversas aplicações e experimentos. Trata-se de um Dataset relativamente pequeno que tem em torno de 144 registros mensais ao longo do tempo (Figura 4.8).

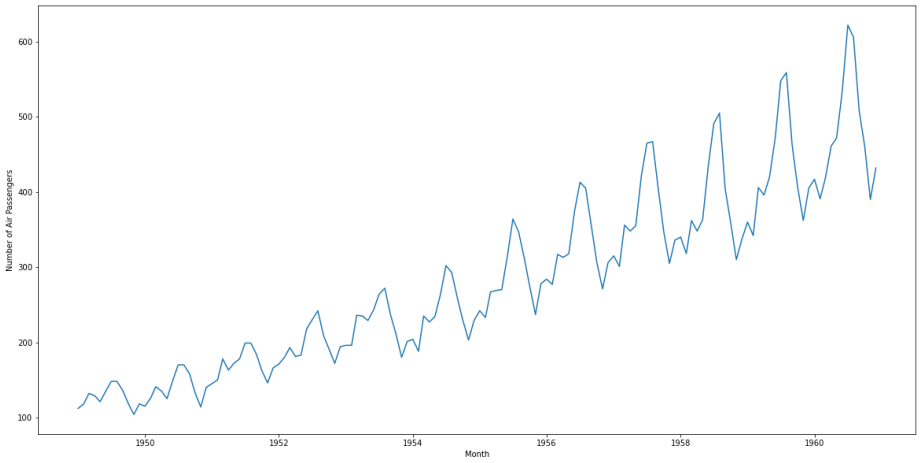


Figura 4.8: *Série Airline Passangers*

Breve análise descritiva foi realizada neste conjunto de dados para destacar algumas características especificas. Visualmente é possível identificar os 3 elementos que compõe esta série temporal de forma muito nítida nos dados de passagens áreas (característica que contribui para a popularidade desta base de dados): Tendência, Sazonalidade e Heterocedasticidade. A tendência crescente ao longo do tempo indica que há um aumento gradativo no número de passagens áreas no período de tempo analisado, a Sazonalidade tem característica anual (12 meses) repetindo certas aspectos de forma sazonal espaçadas

nos meses, já a Heterocedasticidade(também conhecida como variabilidade) aumenta (oscilação) cada vez mais conforme a evolução do tempo (Figura 4.9 e 4.10).

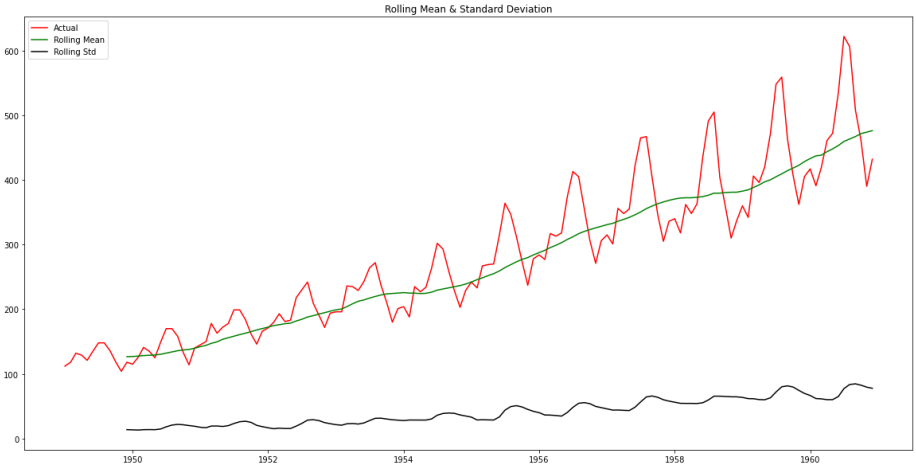


Figura 4.9: *Média e Variância Flutuante da série Airline*

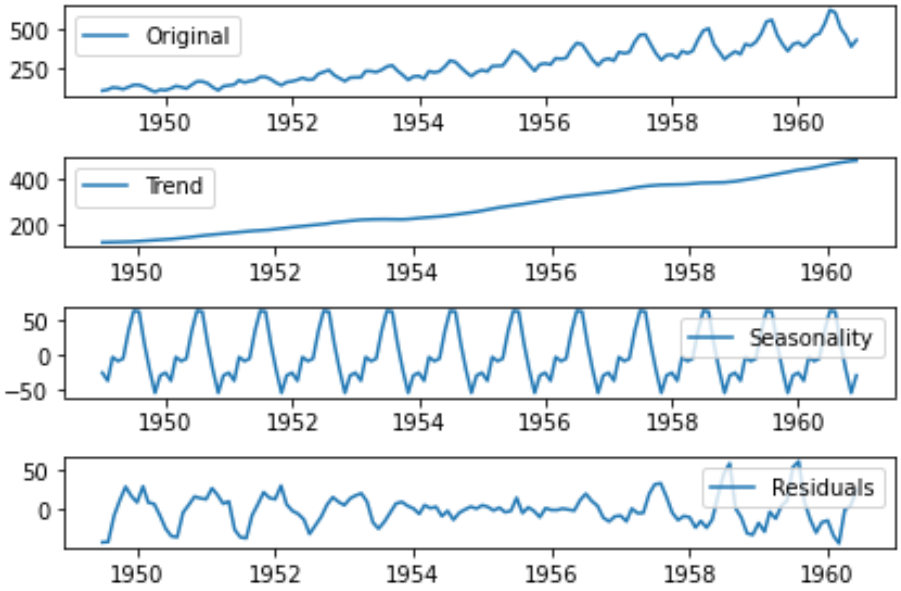


Figura 4.10: *Decomposição da série Airline*

Conforme descrito no Capítulo 2, o algoritmo AARIMA foi considerado para a base de passagens áreas como modelo Baseline O modelo que obteve melhor desempenho entre os 43 modelos testados no procedimento do algoritmo tem a seguinte configuração ARIMA(2,0,0)(1,0,1)[12]. Destaque para os parâmetros Auto-regressivos que aparecem de forma convencional e sazonal no modelo, a sazonalidade identificada representa ciclos de um ano (12 meses), conforme destacado nos Figura 4.8 e 4.9.

Cinco diferentes estruturas de Algoritmos de Deep Learning foram consideradas para o conjunto de dados Airline Passangers: LSTM, Stacked LSTM, Bidirection LSTM, Bidirection Dense LSTM e ConvLSTM. Para cada algoritmo foram selecionados um Grid para os Hiperparâmetros(listados na sessão 3.3)) resultando no total de 54 configurações para

cada algoritmo. Adicionalmente, por se tratar de um processo de otimização para cada configuração foram feitas entre 10 repetições e considerou-se a média da medida de qualidade (MAPE) para elencar a melhor performance para cada estrutura. Cerca de 2700 modelos foram ajustados considerando todas as estruturas propostas, a Tabela 4.1 destaca os resultados de melhor performance para cada algoritmo.

Algoritmo	Configuração	Parâmetros	MAPE
LSTM	[12, 32, 50, 1, 0]	17.920	5.73
Stacked LSTM	[12, 64, 50, 1, 0]	19.456	4.94
Bidirectional LSTM	[6, 64, 50, 150, 12]	35.840	3.01
Bidirectional Dense LSTM	[6, 32, 50, 150, 12]	10.752	2.93
ConvLSTM	[10, 64, 100 ,1, 0]	18.994	18.21
ARIMA(Baseline)	ARIMA(2,0,0)(1,0,1)[12]	5	3.07

Tabela 4.1: *Performance dos modelos na base Airline*

A tabela 4.1 consolida os resultados dos experimentos realizados. O modelo Baseline (ARIMA) obteve um ótimo desempenho (MAPE de 3.07) na base de teste, que significa a obtenção do erro percentual médio de 3.07%. Dado a performance do modelo Baseline todos os algoritmos de Deep Learning não obtiveram melhor performance, com exceção do "Bidirectional LSTM" que obteve MAPE similar, porém melhor que o modelo Baseline (MAPE 3.07). A coluna de configurações da Tabela 5 representa os seguintes hiper parâmetros para os modelos de Deep Learning: **Sliding Window, Nodes, epochs, batch size e diferenciação**, respectivamente.

Os algoritmos de Deep Learning mesmo tendo uma quantidade expressiva de parâmetros, comparado ao modelo Baseline, obtiveram performance semelhante, sendo que a característica sazonal foi incorporada por todas as estruturas consideradas nesta aplicação, seja através do Sliding Window ou pela diferenciação. ConvLSTM é a única estrutura que não destaca componentes sazonais e resulta na pior performance dos algoritmos ajustados. De acordo com a performance na base de teste não é observado problemas de Overfitting nos dados nem problemas de estimação frente a expressiva quantidade de parâmetros comparado ao modelo ARIMA.

O modelo baseline não realiza diferenciação nos dados ao passo que o modelo de melhor performance (Bidirectional LSTM) realiza a diferenciação de ordem 12 (sazonal). Dada a flexibilidade dos modelos de Deep Learning estas características podem ser testadas por intermédio dos Hiperparâmetros mapeados. Importante destaque para a quantidade de registros (144 observações) que não evidenciou possíveis impedimentos de estimação nas estruturas de aprendizado profundo neste contexto.

As Figura 4.11 e 4.12 ilustram o ajuste do modelo Baseline ARIMA e LSTM Bidirectional a base de teste da série de passagens aéreas. Visualmente ambos os modelos são capazes de captar e projetar, em cenários futuros, as características dos dados de passagens aéreas. Todos os componentes (Tendência, Sazonalidade e Heterocedasticidade) são contemplados de modo que de forma visual as curvas são próximas e o erro percentual médio em torno de 3%.

Apesar do Algoritmo de Deep Learning obter melhor performance sobre o Baseline, para este cenário o resultado de performance é bem próximo. Não há um destaque relevante para os modelos de RNA. Em contrapartida a interpretação do modelo Bidirectional LSTM se resume aos Hiperparâmetros, ao passo que o modelo ARIMA permite avaliar todos os parâmetros do modelo Baseline.

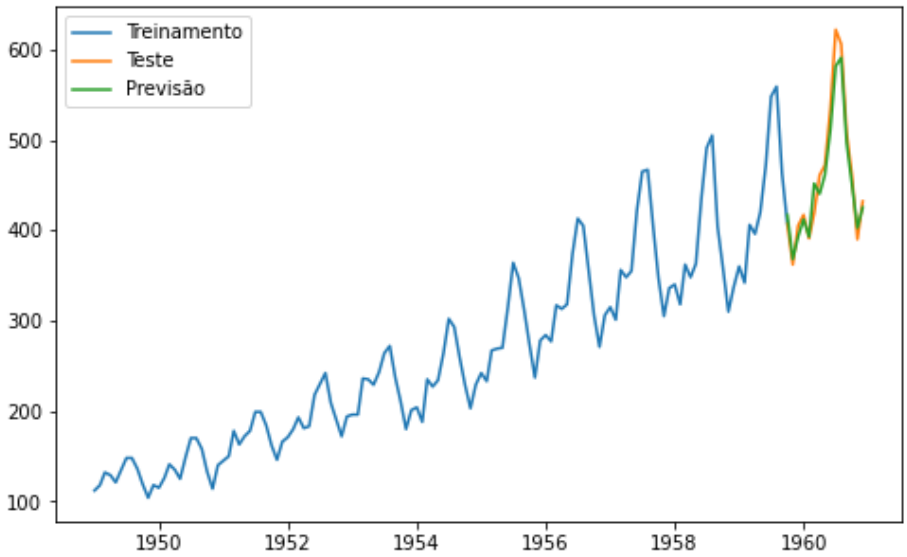


Figura 4.11: *Performance do Modelo ARIMA na base de Teste*

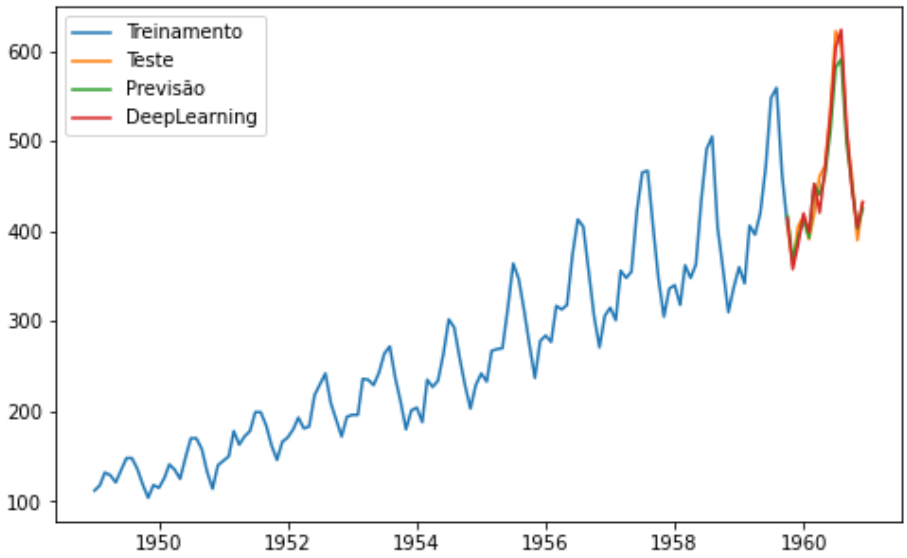


Figura 4.12: *Performance do modelo ARIMA e LSTM Bidirectional na base de Teste*

4.2.2 Montly Beer Australian Production

A indústria de cerveja contribui significativamente para a economia australiana. A grande maioria da população (82%) declarou consumir bebidas alcoólicas, além de que o país é o maior produtor de cervejas de toda a Oceania. Este conjunto de dados tem a

produção de cervejas de janeiro de 1956 até agosto de 1995, totalizando 477 registros ao longo do tempo (cerca de 4 vezes mais registros que a base de passagens aéreas da sessão 4.2.1). O Figura 4.13 descreve visualmente a série histórica.

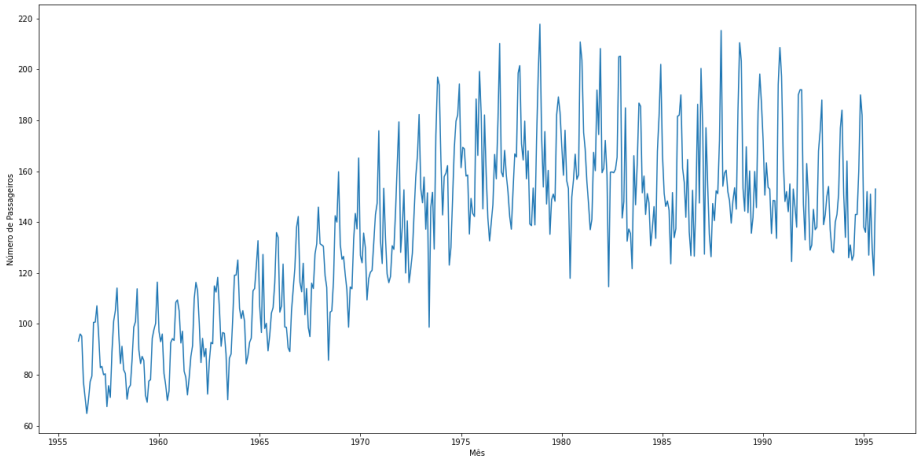


Figura 4.13: *Série de produção de cerveja (Beer Production) na Austrália*

A série demonstra ser volátil e sazonal além de ter uma tendência crescente (ao considerar o início e o término da série), as Figuras 4.14 e 4.15 corroboram estas características em maiores detalhes. Apesar de ter quatro vezes mais observações do que a "Air Passangers" da sessão anterior o cenário também pode ser considerado de "small data" e algumas validações serão realizadas neste sentido. Objetivo consiste em ajustar um modelo ARIMA (Baseline Model) e comparar com as diferentes estruturas de Deep Learning propostas.

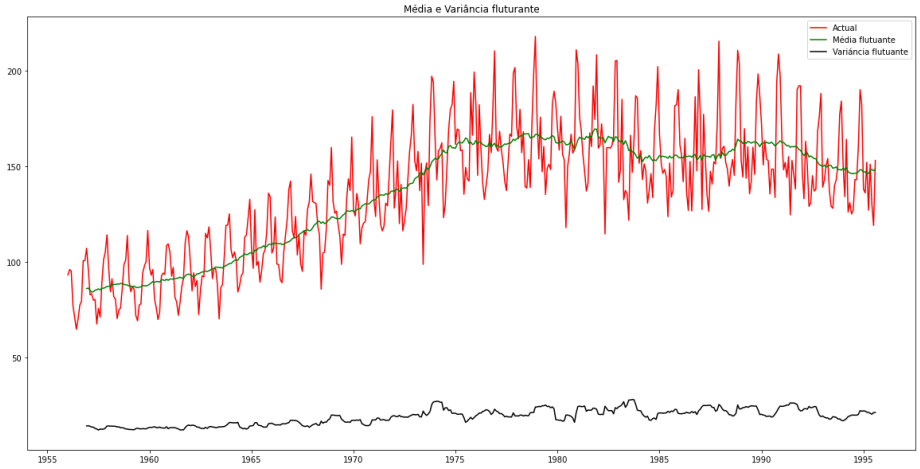


Figura 4.14: *Média e variância Fluturante Beer Production*

De forma similar a sessão 4.2.1 o algoritmo AUTO ARIMA foi aplicado na base de produção de cervejas Australianas possibilitando a estimação de um conjunto de 52 modelos da classe ARIMA. O Procedimento equivale a selecionar o modelo que obteve melhor performance de acordo com o critério de seleção escolhido, neste cenário é considerado o critério AIC.

Dos mais de cinquenta modelos ARIMA ajustados, o modelo ARIMA (3,0,0)(1,0,1)[12]

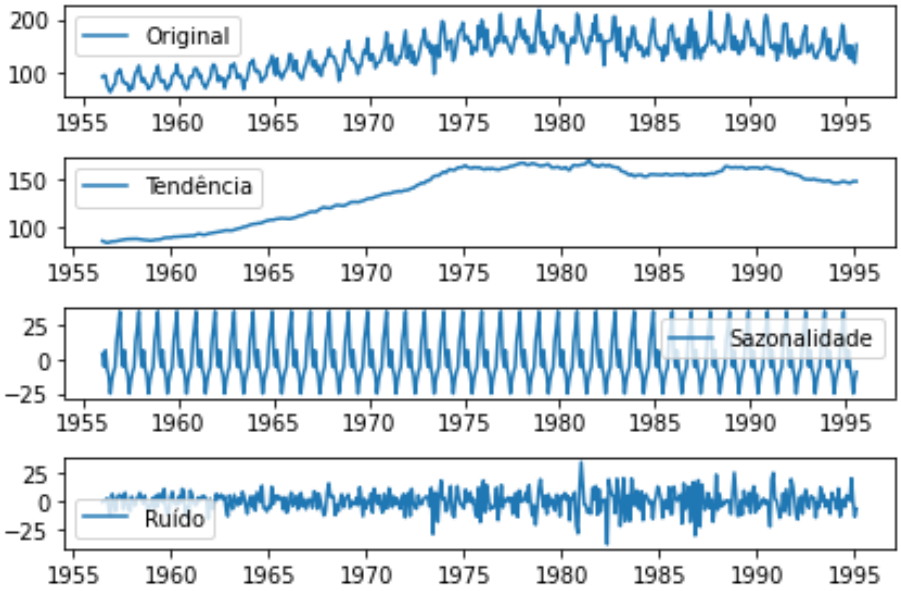


Figura 4.15: *Decomposição da série de Produção de cervejas na Austrália*

com intercepto obteve melhor performance segundo AIC e será utilizado como Baseline para a série de produção de cervejas na Austrália. Importante destacar que o modelo ARIMA selecionado é sazonal de ordem 12 além de possuir parâmetro autorregressivo (sazonal e não sazonal) e média móvel sazonal. Este modelo demonstra que o modelo ARIMA foi capaz de modelar a característica comportamental sazonal anual identificada descritivamente nos dados.

Algoritmo	Configuração	Parâmetros	MAPE
LSTM	[12, 16, 50, 1, 12]	1.792	5.70
Stacked LSTM	[10, 16, 50, 150, 12]	3.328	5.69
Bidirectional LSTM	[10, 64, 50, 150, 12]	37.952	5.68
Bid. Dense LSTM	[12, 16, 50, 150, 12]	3.840	6.13
ConvLSTM	[10, 64, 100, 1, 0]	18.944	15.51
ARIMA	ARIMA(3,0,0)(1,0,1)[12]	6	6.25

Tabela 4.2: *Performance dos modelos na base Beer*

Cinco diferentes estruturas de Deep Learning foram propostas para ajustar a série de produção de cerveja sendo que para cada estrutura um processo exaustivo de Grid Search foi realizado para que os modelos fossem otimizados através dos Hiperparâmetros. Para cada estrutura cerca de 50 configurações foram propostas de acordo com os as possibilidades dos Hiperparâmetros. A Tabela 4.2 consolida todo o processo de aplicação e performance dos algoritmos propostos:

As estruturas LSTM, Stacked LSTM e Bidirectional LSTM obtiveram MAPE muito próximo entre si, sendo que a estrutura Bidirectional apresenta melhor performance ao considerar a segunda casa decimal. Desta forma, sendo o melhor modelo ajustado a base de produção de cervejas australianas, superando o modelo Baseline (ARIMA), que obteve

uma performance interessante na série em análise, mas não o suficiente para a grande maioria das estruturas propostas de RNA.

A estrutura ConvLSTM não obteve um bom ajuste quando comparada com as demais estruturas RNA e até mesmo com o modelo Baseline, está foi a única arquitetura que obteve performance abaixo do modelo ARIMA proposto. A arquitetura Bidirectional Dense LSTM tem performance parecida com modelo ARIMA mas superior considerando a primeira casa decimal do MAPE. O modelo de melhor performance (Bidirectional LSTM) foi capaz de captar comportamentos importantes da estrutura da série em estudo. A configuração, inclusive das três melhores estruturas de Deep Learning, permitiram captar através dos Hiperparâmetros o comportamento sazonal da série por meio da diferença sazonal de ordem 12 ou até mesmo pelo tamanho da Janela de Sliding Window(janela de 12 ou 10 lags para os algoritmos).

Apesar da quantidade de parâmetros considerada nos modelos de Deep Learning o método de otimização através do algoritmo Adam (veja sessão 3.6.2) se mostrou eficiente, o que pode ser constatado através dos resultados obtidos. Overfitting também não foi identificado pois os modelos ajustados foram capazes de generalizar (realizar previsões) em uma base de teste, obtendo bons resultados de Forecast (MAPE em torno de 5%).

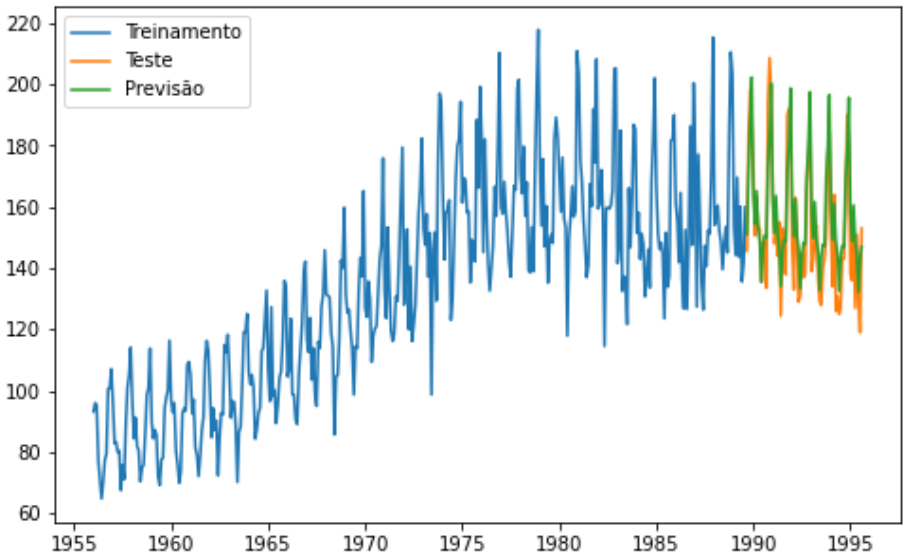


Figura 4.16: *Performance Modelo ARIMA na base de Teste (Beer Production)*

Figura 4.16 demonstra visualmente a performance do modelo ARIMA na base de teste. Com MAPE de 5 é realmente esperado que o modelo ajustado tenha comportamento similar aos dados reais da base de teste captando os comportamentos sazonais, de tendência e volatilidade identificados ao longo do tempo.

O modelo de Deep Learning (Birirectional LSTM) também obteve bom ajuste nos dados (MAPE em torno de 5%) e visualmente o ajuste pode ser corroborado (veja Figura 4.17). A série deste cenário tem comportamento volátil que consegue ser incorporado por ambos os modelos em destaque no gráfico acima (Baselise e Bidirectional LSTM). A sazonalidade dos modelos de Deep Learning é descrita por intermédio dos Hiperparâmetros (Sliding Window ou Diferenciação).

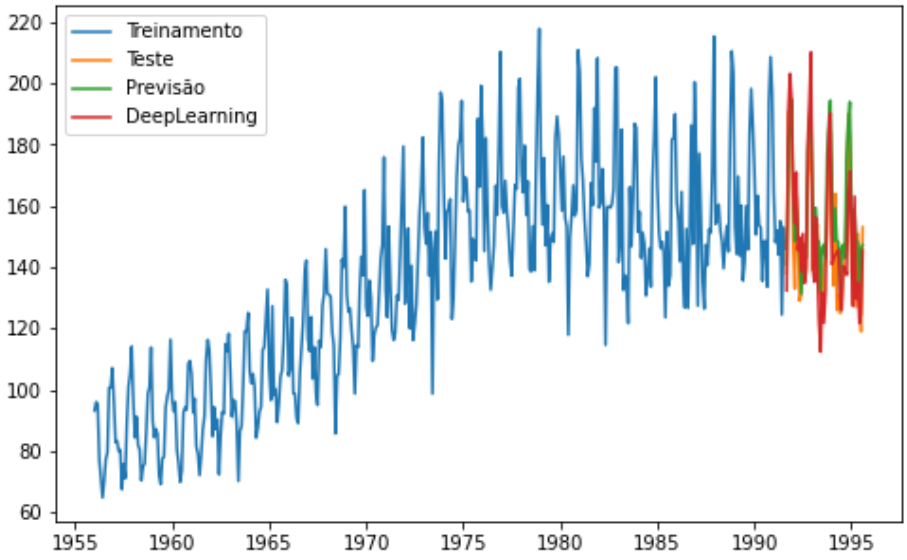


Figura 4.17: *Performance modelo ARIMA e Bidirectional LSTM na base de Teste (Beer Production)*

Neste cenário de aplicação é possível identificar a flexibilidade dos modelos de Deep Learning na obtenção de performance superior ao modelo Baseline. Apesar de performance semelhante, todos os modelos, com exceção do ConvLSTM, obtiveram melhor desempenho ao comparado com Baseline, mesmo com número significativamente maior de parâmetros. A estrutura Bidirectional merece destaque pois também se sobressaiu para a série de passagens aéreas ameridacas (sessão 4.2.1). Esta estrutura pode ser indicada como Baseline para modelo RNA frente a séries temporais para cenários semelhantes de dados de contagem.

4.2.3 GE Open

General Electric (GE) é um conglomerado multinacional de Nova York e sediado em Boston, Massachusetts, Estados Unidos. Em 2016, a empresa atuava nos seguintes segmentos: aviação, software, conexões de energia, pesquisa global, assistência médica, iluminação, petróleo e gás, energia renovável, transportes e capital, serviços financeiros, dispositivos médicos, ciências da vida, produtos farmacêuticos, indústria automotiva e indústrias de engenharia. Em 2017, a GE foi classificada na Fortune 500 como a terceira maior empresa dos Estados Unidos por receita bruta. Em 2011, a GE foi classificada como a 14ª mais lucrativa. Em 2012, a empresa foi listada como a quarta maior do mundo entre o Forbes Global 2000. O Prêmio Nobel foi premiado duas vezes aos funcionários da General Electric: Irving Langmuir em 1932 e Ivar Giaever em 1973.

Para este cenário de aplicação as ações da GE na bolsa americana no período de 04/11/2019 até 27/08/2021, totalizando 458 registros (observações). Dada a característica estocástica da série (Veja Figura 4.18) o objetivo é justamente analisar o comportamento e desempenho dos modelos de Deep Learning neste contexto. As séries das sessões 4.2.1 e 4.2.2 apresentavam comportamento de tendência determinística ao passo que as ações da GE, visualmente, apresentam tendência estocástica que evolui de forma diferente ao longo do tempo (ora crescente, ora decrescente e ora estacionária, dependendo do período

considerado).

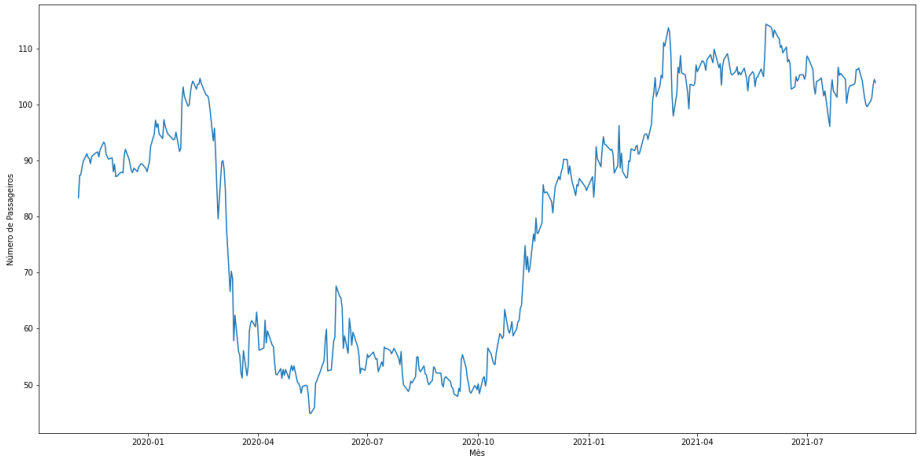


Figura 4.18: *Preço de Abertura(Open) de ações da GE*

O comportamento da tendência das ações da GE é corroborado na análise descritiva da série por intermédio do Figura 4.19 de decomposição dos componentes de forma flutuante, a saber, média e variância. O Comportamento estocástico de ações é comum devido ao mecanismo gerador da série que é influenciado por acontecimentos externos (cenários políticos e movimentos setoriais, por exemplo) que podem impactar o preço mudando a tendência e a volatilidade em determinados períodos.

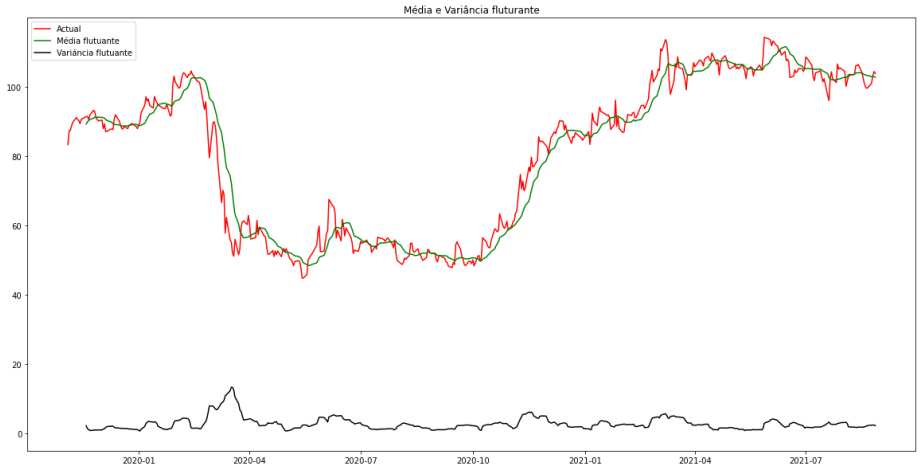


Figura 4.19: *Média e variância flutuante (GE Open)*

Diferentes classes de modelos de Deep Learning foram consideradas na modelagem do preço de abertura das ações da GE. Das seis classes consideradas (Tabela 4.3) a arquitetura que demonstrou melhor performance na modelagem do preço de abertura das ações da GE foi a estrutura LSTM simples que obteve erro médio percentual de 3.65. O modelo baseline ARIMA teve MAPE em torno de 4.62 ficando melhor apenas do modelo ConvLSTM que obteve a pior performance das estruturas de modelos aplicados.

Algoritmo	Configuração	Parâmetros	MAPE
LSTM	[21, 16, 50, 150, 14]	2.368	3.66
Stacked LSTM	[21, 16, 50, 150, 14]	4.736	3.70
Bidirectional LSTM	[21, 32, 50, 150, 7]	13.568	3.68
Bid. Dense LSTM	[7, 32, 50, 150, 0]	11.008	3.67
ConvLSTM	[7, 32, 50, 150, 7]	4.992	7.80
ARIMA	ARIMA(1,0,1)(0,0,0)[12]	3	4.62

Tabela 4.3: *Performance dos modelos na base de Preços de abertura da ação GE*

Os modelos de Deep Learning demonstram versatilidade ao ajustar e obter performance melhor que o modelo Baseline (ARIMA) frente aos dados de ações, pois têm característica estocástica quando comparado aos dois primeiros cenários (4.2.1 e 4.2.2) que representam dados de contagem. As Figuras 4.20 e 4.21 descrevem a performance dos modelos na base de teste, visualmente o modelo LSTM é capaz de captar a oscilação dos dados e projetar em períodos futuros, ao passo que o modelo ARIMA realiza uma projeção "linear decrescente" não acompanhando o movimento de volatilidade dos dados em períodos de previsão.

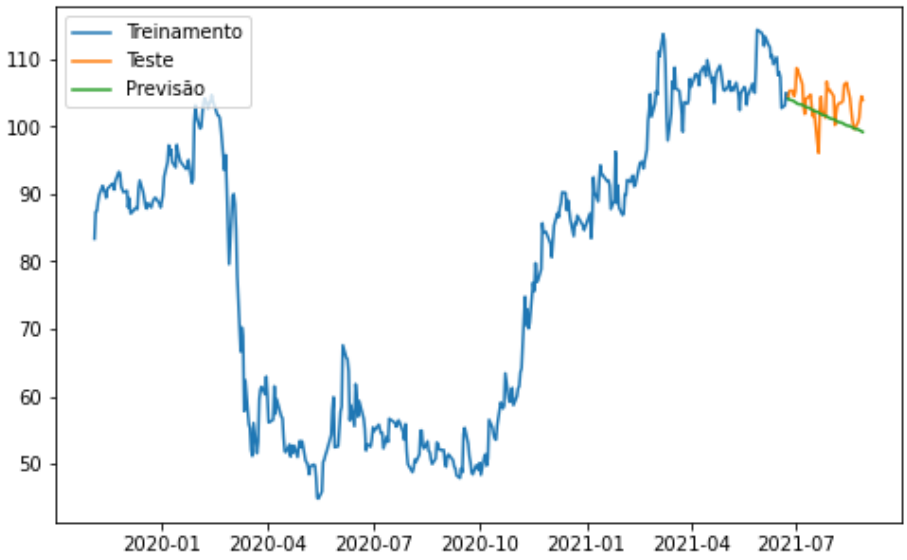


Figura 4.20: *Performance do modelo ARIMA na base de teste GE Open*

Com o cenário do preço de abertura das ações da GE as aplicações univariadas são finalizadas, destaque que os modelos de Deep learning obtiveram performance superior ao Baseline em todos os cenários propostos. Mesmo sem considerar as premissas convencionais dos modelos tradicionais as arquiteturas de RNA são capazes de captar certas características importantes na performance dos modelos por intermédio dos hiperparâmetros considerados.

Os modelos foram flexíveis modelando dados de contagem (dois primeiros cenários), assim como dados de ações, com características estocásticas mais expressivas (cenário das ações da GE). A quantidade de parâmetros não se mostrou um problema de estimação para séries "small data", no entanto a interpretação dos modelos de RNA não é intuitiva como

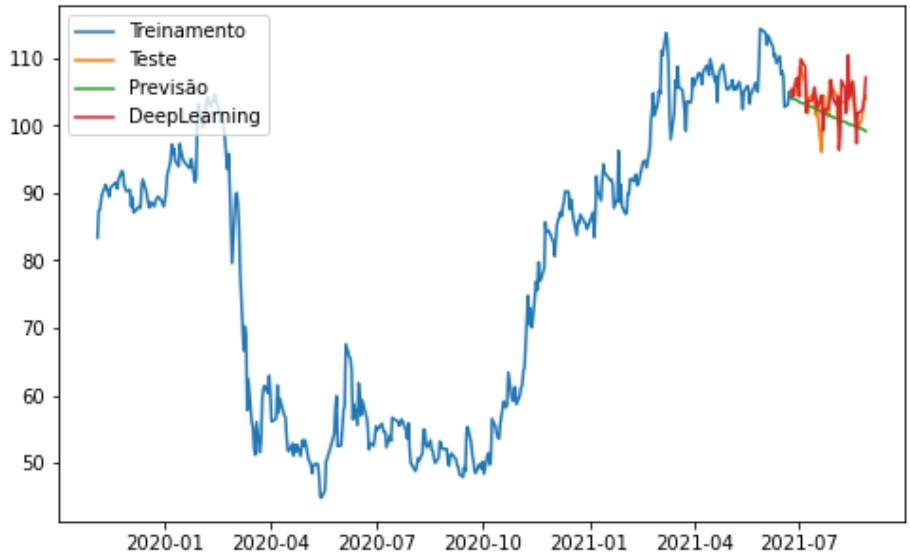


Figura 4.21: *Performance modelo ARIMA e LSTM na base de teste GE Open*

nos modelos ARIMA. Ao considerar séries com maior número de registros a performance geral dos modelos de Deep Learning evoluiu, indício de que essas arquiteturas têm melhor performance conforme aumento de dados/registros disponíveis.

4.3 Séries Multivariadas

No cenário dinâmico e complexo dos dados temporais, a análise de séries temporais multivariadas emerge como uma vertente fundamental para compreender as interações entre diversas variáveis ao longo do tempo. Enquanto as abordagens tradicionais têm desempenhado um papel crucial, a ascensão da inteligência artificial, em particular do Deep Learning, oferece uma promissora fronteira para aprimorar a modelagem e previsão nesse contexto.

Esta dissertação explora o emprego de técnicas avançadas de Deep Learning na análise de séries temporais multivariadas, buscando transcender as limitações das metodologias convencionais. O desafio reside na capacidade de capturar não apenas as dependências temporais intrínsecas em cada variável, mas também as relações complexas e interdependências entre diferentes séries temporais.

Ao delinear o caminho entre os fundamentos das séries temporais multivariadas e as nuances do Deep Learning, esta pesquisa visa não apenas expandir o arsenal de ferramentas analíticas disponíveis, mas também proporcionar uma compreensão aprofundada das implicações e oportunidades que surgem ao adotar abordagens inovadoras. Através dessa jornada, busca-se responder questões específicas e catalisar o avanço contínuo no emprego de Deep Learning para séries temporais multivariadas.

4.3.1 GE Open(Usando as 5 séries de preço para prever o Volume)

As Estruturas de Deep Learning são flexíveis e permitem abordagem para o caso Multivariado, o que possibilita a modelagem de um conjunto de séries de forma conjunta. As mesmas estruturas consideradas no caso Univariado serão adaptadas para o caso Multivariado. O tratamento nos dados será realizado de forma similar por intermédio dos mesmos conceitos: Sliding Window e Reshape dos dados (Sessão 4.1). Não há premissas ou restrições para os dados, basta que a estrutura das séries seja compatível com a arquitetura dos algoritmos multivariados propostos.

Os modelos da Classe ARIMA são limitados e não são aplicados no caso Multivariado a classe adotada como baseline neste contexto é a classe VAR(Vector autoregression) que permite a contribuição de diversas séries temporais para previsão de uma série específica. Neste contexto já é possível observar que a complexidade para os modelos tradicionais (VAR no caso Multivariado) aumenta assim como as restrições frente aos dados (há premissas sobre a distribuição conjunta dos dados).

O caso Multivariado tem como contexto o fornecimento de diversas séries temporais sejam utilizadas com intuito de ajuste ou previsão de uma série específica. O conjunto de dados de ações da empresa GE são considerados na aplicação Multivariada das estruturas de RNA. O conjunto de dados é composto por 5 séries: GE (Open – Abertura, High – maior valor diário, Low – menor valor diário, Close – valor de fechamento, AdjClose – valor de fechamento ajustado), Figura 4.22.

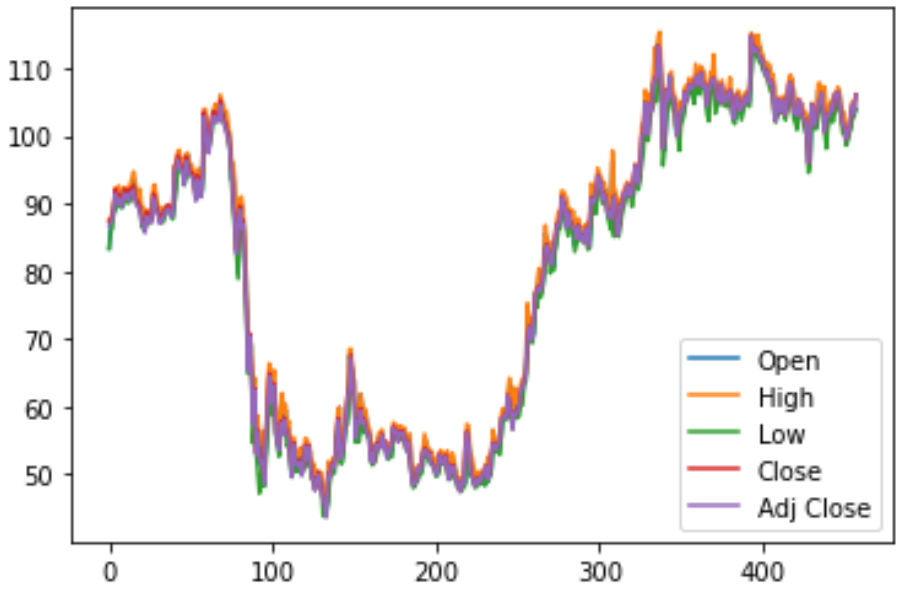


Figura 4.22: *GE Open e os valores de Abertura, fechamento, máximo, mínimo e ajustado*

As séries dizem respeito aos valores da ação da empresa ao longo do tempo (de 2019 a 2021). O Gráfico acima descreve o comportamento similar das séries consideradas pois tratam do mesmo ativo (a série GE Open foi modelada de forma univariada na sessão anterior). No entanto, a análise Multivariada desta sessão terá como objetivo utilizar as 5

séries descritas para prever o volume de ações ao longo do tempo. Isto é, será avaliado o quanto o preço da ação pode ajudar a prever o volume de ações que serão negociadas em um período.

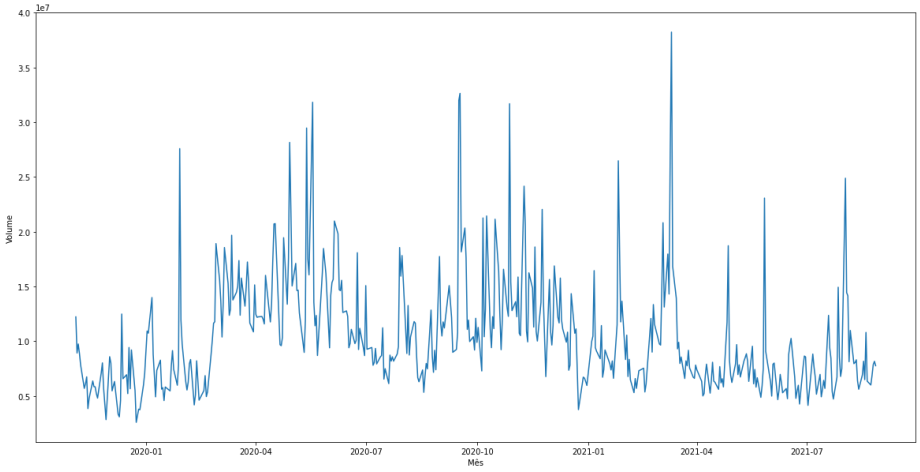


Figura 4.23: *GE Open Volume*

A Figura 4.23 descreve o volume de ações da GE no mesmo período que os preços foram considerados. É nítida a diferença de escala: a escala do volume é em milhares ao passo que a escala do preço está limitada a dezenas. A volatilidade do volume tem característica mais agressiva do que a do preço na ótica visual das séries. Abaixo o A Figura 4.24 descreve todas as séries em uma mesma visualização. Como esperado a escala do volume acaba distorcendo as séries de preço de forma que a visualização conjunta acaba de certo modo comprometida.

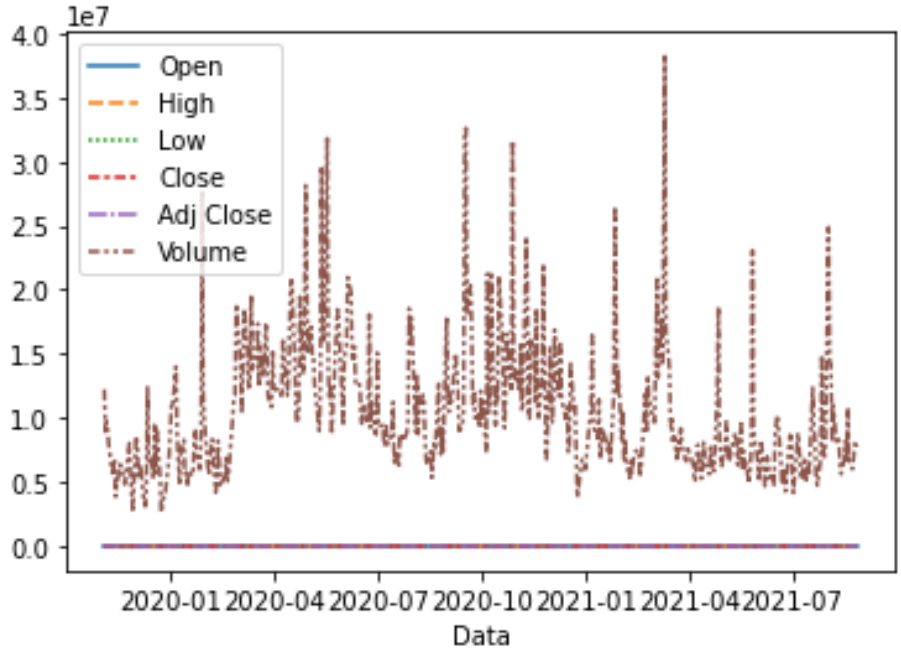


Figura 4.24: - *GE Open Abertura, fechamento, máximo, mínimo, ajustado e Volume*

Os modelos de Deep Learning têm a missão de ajustar a série de Volume de ações da GE com base nas séries históricas do preço da ação por intermédio de uma análise de série temporal multivariada. As séries foram observadas no mesmo período totalizando 458 observações para cada série. O conjunto de dados dos Preços (Open,High, Low,Close e Adj Close) têm visualmente comportamento similar pois tratam-se dos valores do mesmo fenômeno de comportamento em diferentes momentos do dia (Abertura da Bolsa, Maior valor diário, menor valor diário, valor de fechamento e valor de fechamento ajustado, respectivamente). No geral, a série tem uma tendência que evolui junto ao tempo com heterocedasticidade expressiva em momentos/anos específicos além de uma sazonalidade não identificada de forma visual.

A Figura 4.25 permite análise mais detalhada da série de Volume de ações da GE por intermédio da média e variância flutuante. Para o volume de ações a média e a variância não são constantes ao longo do tempo, ambas têm características estocásticas neste sentido. O Volume, apesar da Heterocedasticidade, permite identificar de forma visual qual tipo de tendência a série mais se assemelha (crescente, decrescente, estacionária ou estocástica), indicando uma evolução ao longo do tempo, característica de uma tendência estocástica.

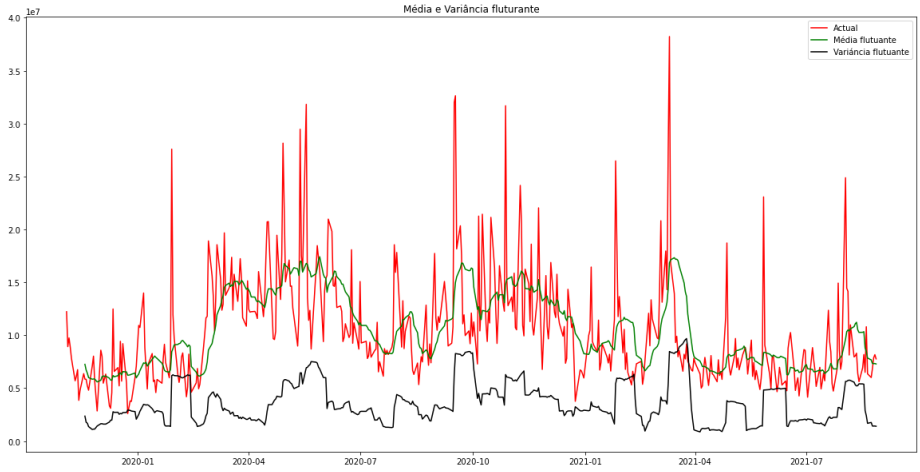


Figura 4.25: *Decomposição GE Open Volume*

Algoritmo	Parâmetros	MAPE
LSTM	22,401	23.92
Stacked LSTM	53,313	29.89
Bidirectional LSTM	35,969	26.05
Dense LSTM	30,625	20.28
VAR	375	25.06

Tabela 4.4: *Performance dos modelos na base de Volume da ação GE*

Similar ao caso Univáriado, diferentes arquiteturas de RNA foram consideradas para o ajuste Multivariado da série de Volume. Todos os modelos consideraram o mesmo conjunto de dados tendo como target o Volume de ações da GE, como baseline o modelo VAR

foi ajustado. A Tabela 4.4 descreve o resultado do ajuste dos modelos de Deep Learning Multiraviados.

Diferentes configurações foram consideradas para cada arquitetura do caso Multivariado, porém de forma mais restrita ao caso Univariado por conta do tempo de processamento de cada modelo. Os Hiperparâmetros foram os mesmos das sessões Univariadas e não houve uma característica específica que obteve destaque para as arquiteturas consideradas na modelagem do volume de ações da GE por intermédio dos preços da ação.

O modelo LSTM com camada Densa obteve o melhor resultado entre todos os modelos testados. O modelo baseline obteve MAPE 25 ao passo que o modelo Dense LSTM obteve MAPE 20. Importante destacar que o modelo baseline teve desempenho melhor do que as arquiteturas Bidirectional e Stacked LSTM que obtiveram 26 e 29 de MAPE, respectivamente. No entanto a classe de Deep Learning obtive melhor performance ao modelo Baseline proposto.

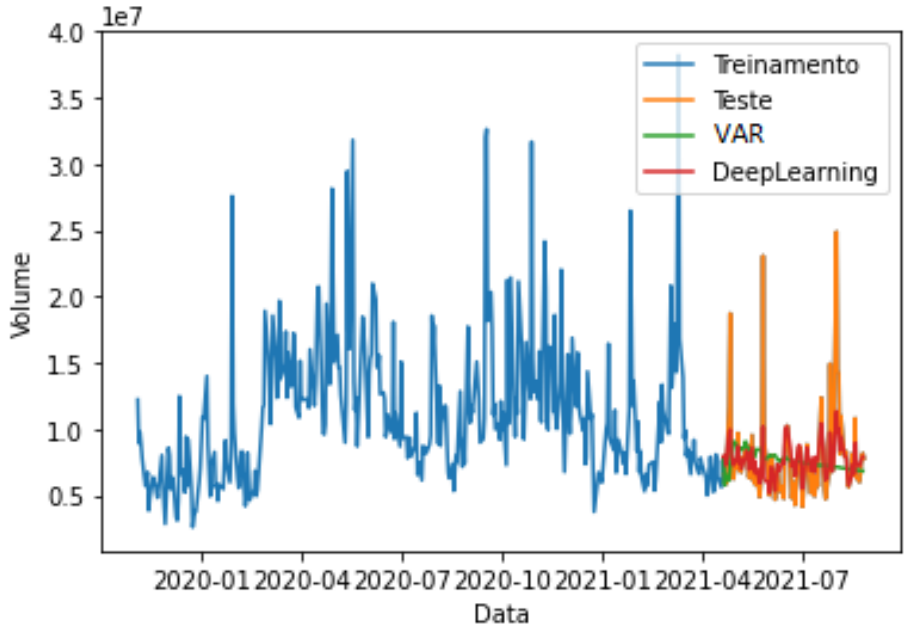


Figura 4.26: *Performance modelo ARIMA e LSTM na base de teste Volume da GE Open*

A Figura 4.26 descreve de forma visual o desempenho de cada modelo, apesar do MAPE de 25 o modelo VAR tem característica estacionária no momento da previsão (tende a estabilizar em torno de uma média constante) ao passo que o modelo Denso LSTM consegue captar as respectivas oscilações da série (note que em proporções menores aos picos não sazonais).

Mesmo em um cenário multivariado e complexo os modelos de RNA demonstram flexibilidade e variedade de opções frente a este tipo de aplicação. Obtiveram desempenho melhor do que o modelo Baseline, com benefício de não ter premissas sobre a distribuição conjunta dos dados (na prática essas suposições são difíceis de serem reais e ajustes/transformações precisam serem aplicadas). O Contraponto para os modelos de Deep Learning é a interpretação, mas no caso Multivariado até mesmo a interpretação do VAR é complexa por envolver múltiplas séries e consequentemente muitos parâmetros.

4.3.2 Prevendo todas as séries simultaneamente

A sessão anterior aborda o caso Multivariado no qual múltiplas séries são utilizadas para prever uma única série específica, no caso o Volume da GE Open. Nesta sessão trata-se o caso mais abrangente, pois é considerado como o objetivo prever diversas séries de forma simultânea através das estruturas de Deep Learning considerando como Baseline o modelo VAR que também possibilita previsão simultânea de diversas séries, mas fica restrito ao procedimento autorregressivo sendo que internamente uma equação individual para cada série é elaborada.

Três cenários distintos foram considerados para os modelos Baseline:

1. Todas as séries do Dataset GE Open foram consideradas simultaneamente;
2. A variável Volume é retirada da análise por ter uma escala muito maior do que as séries de preços (os preços estão na casa das centenas ao passo que o volume está em milhares – veja Figura 4.24 da sessão 4.3.1)
3. Todas as séries do Dataset GE Open padronizadas na escala [0,1], com intuito de averiguar o impacto da diferença de escala nos resultados obtidos;

Os cenários descritos são considerados também para os modelos de Deeplearning no caso Multivariado para previsão simultânea de séries temporais. No primeiro cenário os modelos de Deeplearning não obtiveram resultado satisfatório. Os resultados obtidos das projeções foram valores negativos, cenários pouco prováveis para série de preço e inadmissível frente ao volume (que por definição não pode ser negativo). Identificou-se que a diferença de escala entre as variáveis ocasionou tal fenômeno distorcendo os resultados de performance e resultando também em problemas de estimação.

Dada a limitação dos modelos de Deep learning no cenário de diferenças significativas de escala (no caso preço e volume) os cenários 2 e 3 foram considerados para análise. Ao remover a variável Volume da análise conjunto das séries GE Open os modelos de Deep Learning obtiveram performance adequada na base de teste e casos negativos não ocorreram nas previsões das séries de preço.

Cenário 1

Modelo	Série	Mape
VAR	Open	20.54
	High	20.85
	Low	20.28
	Close	20.72
	Adj Close	20.81
	Volume	25.06

Tabela 4.5: *Performance dos VAR Cenário 1.*

Para o Cenário 1, previsão simultânea de todas as 6 séries temporais (Open, High, Low, Close, Adj Close e Volume), o modelo VAR mostrou-se a melhor opção devido a questão da diferença de escala entre as séries em análise. Averiguou-se que o MAPE da

série de Volume (25) não é tão discrepante das demais séries (em torno de 20). Os modelos de Deep Learning apresentaram problemas de estimação devido a diferença considerável de escala nas séries da GE Open e o Volume. Neste contexto o modelo VAR mostrou-se mais flexível.

Cenário 2

Com intuito de eliminar o problema de diferença de escala, a série de Volume não foi considerada nesta análise do Cenário 2. Isto é, apenas as séries: Open, High, Low, Close e Adj Close foram modeladas simultaneamente e a performance dos respectivos modelos considerada. A Tabela 4.6 descreve o resultado de cada modelo para o conjunto multivariado de séries da GE Open.

Os modelos de Deep Learning obtiveram performance superior ao modelo baseline VAR, com exceção do algoritmo LSTM Dense que obteve melhor performance apenas para algumas séries (Low e Adj Close) e pior performance para as demais. O Erro percentual médio do modelo VAR é em torno de 23 para todas as séries consideradas ao passo que o modelo de Deep Learning que obteve melhor resultado, LSTM Bidirectional Dense, tem MAPE em torno de 2 para todas as séries. Neste cenário é nítido a superioridade de performance dos modelos de Deep Learning, considerando que as séries não têm uma escala similar.

Modelo	Série	MAPE
VAR	Open	23.38
	High	23.66
	Low	23.25
	Close	23.62
	Adj Close	23.69
LSTM	Open	2.75
	High	2.42
	Low	2.32
	Close	2.91
	Adj Close	2.70
LSTM Bidirectional	Open	2.33
	High	2.24
	Low	2.50
	Close	2.57
	Adj Close	2.59
LSTM Dense	Open	24.41
	High	31.34
	Low	20.77
	Close	25.01
	Adj Close	18.69
LSTM Bidirectional Dense	Open	2.12
	High	2.19
	Low	2.09
	Close	2.30

Modelo	Série	MAPE
	Adj Close	2.29

Tabela 4.6: *Performance Modelos Multivariados Cenário 2.*

Ao remover séries com grandes diferenças de escala observa-se performance bem superior dos modelos de Deep Learning no cenário Multivariado. Este é o primeiro cenário que os modelos de LSTM se destacam consideravelmente da abordagem tradicional (ARIMA/VAR). Considerando escala semelhantes para o conjunto de séries em análise os algoritmos de Deep Learning mostram-se como uma interessante alternativa. Esses modelos têm muito mais parâmetros do que os modelos convencionais, conforme sessão 4.2, destacando que conforme o aumento da quantidade dos dados (com a premissa de escala semelhantes) melhor a performance desses modelos.

Cenário 3

Com objetivo de minimizar o impacto da diferença de escala entre as séries este cenário considera a padronização das séries temporais na escala [0,1], desta forma todas as variáveis do Dataset GE Open (inclusive a Série de Volume) são consideradas assim como no Cenário 1. Há outras formas de padronização disponíveis, mas não foram adotadas neste cenário. A Série de volume, mesmo padronizada, traz maior complexidade devido a volatilidade e característica estocástica de tendência (Tabela 4.7 descreve as respectivas performances para cada Série do conjunto Multivariado padronizado).

O Modelo Baseline tem uma queda de performance frente a padronização das variáveis, o MAPE é em torno de 35 ao passo que no Cenário 1 é de 20. Ou seja, a padronização de variáveis minimiza o impacto da diferença de escala para os modelos de Deep learning mas ocasiona diminuição de performance para a classe de modelos multivariados VAR. Para os modelos de Deep Learning, nota-se que não há problemas de estimação nem valores negativos como ocorrido no cenário 1. Os modelos têm performance superior ao Baseline para as variáveis de preço (Open, High, Low, Close e Adj Close). Para a variável Volume os modelos LSTM e LSTM Bidirectional obtivem melhor performance ao passo que os modelos LSTM Dense Layer e LSTM Bidirectional Dense Layer tiveram performance inferior apenas para a série de Volume na comparação com modelo Baseline.

Destaca-se neste contexto a importância da escala das séries no caso multivariado para modelos de Deep Learning. O Modelo VAR obtém performance similar para todas as séries consideradas demonstrando que a diferença de escala não impacta na performance geral do conjunto de dados adotado. Uma vez que a performance dos modelos de Deep Learning é superior no caso de escalas semelhantes é recomendável avaliar descritivamente qual dos cenários adotar em cada aplicação.

Considerando os três Cenários analisados destaca-se que a escala do conjunto de séries multivariadas pode impactar consideravelmente na performance dos modelos aplicados. A padronização pode minimizar o impacto da diferença de escala entre as variáveis de interesse, diferentes métodos de padronização podem ser considerados. Para Séries com escala similar os modelos de Deep Learning se destacam quando comparados ao modelo

Baseline VAR, o que posiciona os modelso de LSTM como boa alternativa em cenários de aplicação e modelagem com essas características.

Série	Modelo	MAPE
Open	VAR	36.16
	LSTM	3.69
	LSTM Bidirectional	3.83
	LSTM Dense	4.23
	LSTM Bidirectional Dense	3.36
High	VAR	36.58
	LSTM	3.34
	LSTM Bidirectional	3.88
	LSTM Dense	4.03
	LSTM Bidirectional Dense	3.41
Low	VAR	35.42
	LSTM	3.47
	LSTM Bidirectional	3.68
	LSTM Dense	3.77
	LSTM Bidirectional Dense	3.39
Close	VAR	35.88
	LSTM	3.81
	LSTM Bidirectional	3.91
	LSTM Dense	4.13
	LSTM Bidirectional Dense	3.67
Adj Close	VAR	35.88
	LSTM	3.79
	LSTM Bidirectional	3.95
	LSTM Dense	4.16
	LSTM Bidirectional Dense	3.67
Volume	VAR	43.32
	LSTM	39.97
	LSTM Bidirectional	26.73
	LSTM Dense	47.64
	LSTM Bidirectional Dense	72.26

Tabela 4.7: *Performance Modelos Multivariados por cada Série do Cenário 3.*

4.4 Simulação de Parâmetro das Matrizes de Pesos

Os algoritmos de Deep Learning têm dezenas de milhares de parâmetros, conforme descrito nas sessões anteriores, de modo que parte deste estudo é justamente avaliar a performance(convergência) dos modelos em um caso em que há um número restrito de informações disponíveis. No contexto de LSTM é possível utilizar o conceito de épocas que permitem que os dados sejam "varridos" pelo modelo algumas vezes, contribuindo para estimação e convergência dos parâmetros/pesos.

A estimação de um algoritmo de Deep learning para arquiteturas LSTMs trata-se de um problema de otimização no tempo (BBTT conforme sessão 3.4.1) em que há eventualmente compartilhamento de parâmetros entre as etapas de estimação por intermédio das matrizes de parâmetros, a saber: W, U e b (conforme Capítulo 3). Esta sessão tem por objetivo estudar o comportamento dos parâmetros de pesos e suas respectivas convergências.

Dois cenários de simulação são propostos: o primeiro cenário considera um passeio aleatório no intervalo [0,10] em uma sequência numérica de 1000 registros (veja Figura 4.27); o segundo cenário consiste em uma sequência aleatória no intervalo [0,10] também com 1000 registros (veja Figura 4.28). A principal diferença entre os cenários é justamente a estrutura de dependência existente, uma vez que no primeiro cenário o passei aleatório decide o próximo número com base no elemento anterior e uma probabilidade alpha de ser menor (X-1, próximo elemento) e probabilidade Beta de ser maior (X+1 ser o próximo elemento da sequência). O segundo cenário considera que o próximo elemento pode ser qualquer inteiro no intervalo da sequência com igual probabilidade.

A diferença entre os cenários é facilmente identificada através das imagens Figura 4.27 e 4.28. No primeiro cenário, passeio aleatório, há dependência probabilística do elemento com seu antecessor e com seu sucessor (evidência de forma matemática tal dependência). O cenário de sequência numérica, não há estrutura de dependência, qualquer número dentro do intervalo pode ser o próximo elemento da sequência, fato que deixa o gráfico visualmente mais denso por conta dessa característica. Duzentas sequências foram geradas para cada cenário e os respectivos modelos LSTMs ajustados para cada sequências em ambos os cenários propostos. Cada sequência tem mil elementos de acordo com critério gerador (a saber: passeio aleatório no cenário 1 e Sequência numérica no cenário 2).

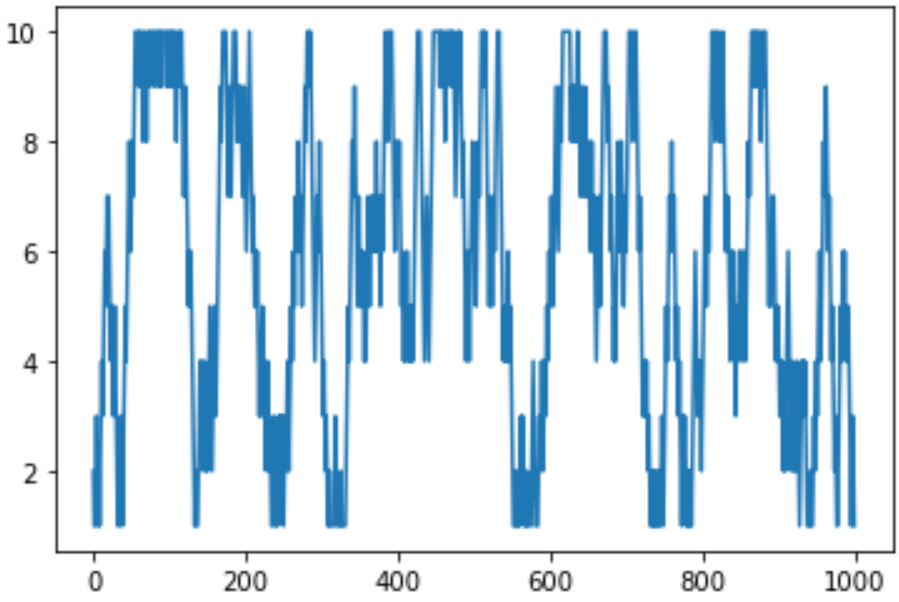


Figura 4.27: *Passeio Aleatório (Cenário 1)*

Para cada cenário duzentos modelos LSTM foram gerados e as respectivas matrizes de parâmetro (W, U e b), de cada modelo, foram armazenadas. Por se tratar de um processo de otimização e convergência, objetivo deste experimento de simulação é averiguar a

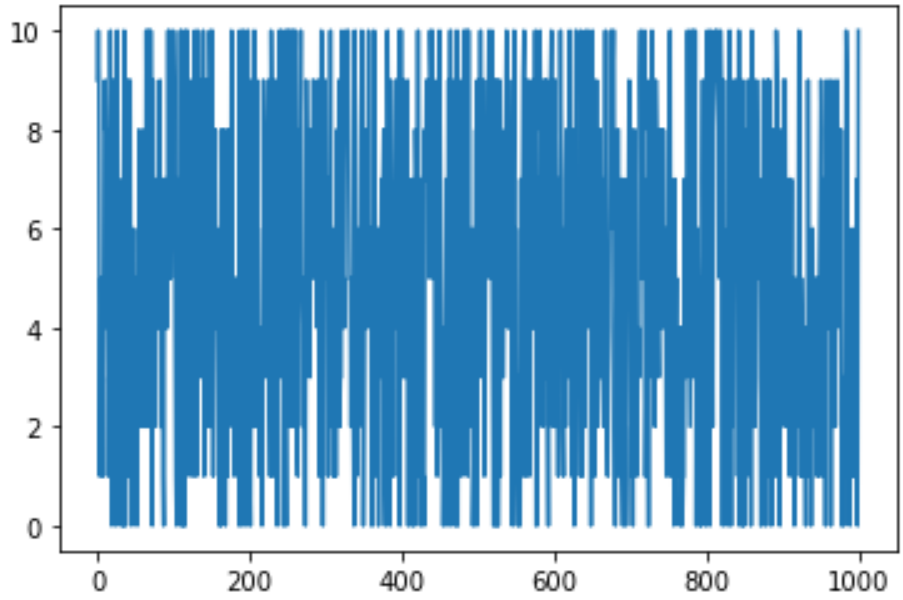


Figura 4.28: *Sequência Aleatória (Cenário 2)*

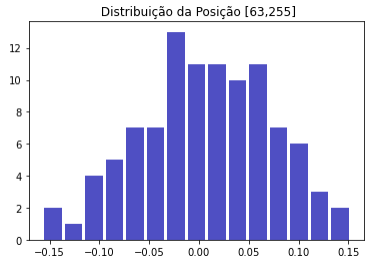
convergência de elementos específicos das matrizes de peso e até mesmo a convergência de toda a matriz, comparando a respectiva norma dos duzentos modelos entre si. Levando também em consideração o cenário do passeio aleatório e da sequência numérica.

Ao comparar a matriz final de parâmetros (W, U e b) de cada modelo evidenciou-se que de fato são distintas, em ambos os cenários. Ou seja, apesar de usar as mesmas sequências para ambos os cenários as matrizes finais de parâmetros de cada modelo não são idênticas, cada modelo tem uma matriz de pesos especifica ao término da estimação/convergência do algoritmo.

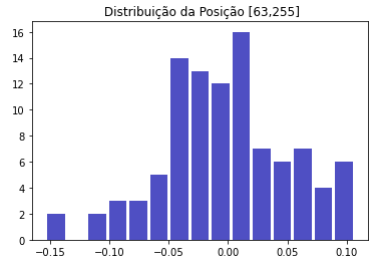
Evoluindo na comparação, elenca-se os pesos de mesma dimensão $(W_{11,1}, W_{21,1}, ...W_{200,1})$, em que W_1 é a matriz de peso do modelo1 cenário 1).Onde os 200 pesos da posição $W_{1,1}$ são comparados entre si. O mesmo se aplica também para as matrizes $U(U_{11,1}, U_{21,1}, ..., U_{2001,1})$ e $b(b_{1,1}, b_{2,1}, ..., b_{200,1})$. Os pesos individualmente das matrizes para cada referência de posicionamento também não são idênticos entre si, dentro de cada cenário. Nem mesmo há alguns casos de igualdade por conta da convergência dos parâmetros.

Nesse contexto avaliou-se a distribuição dos 200 pesos para cada posição em análise (veja Figuras 4.29, 4.30 e 4.31). A distribuição dos pesos demonstra que não há convergência individual dos parâmetros para ambos os cenários. Mesmo com mecanismos geradores distintos (Cenário 1 e 2) cada parâmetro tem uma distribuição de probabilidade, não há uma convergência de parâmetros ainda que, dentro de cada cenário, seja o mecanismo gerador e a arquitetura idêntica para os 200 modelos analisados.

Após constatar que as matrizes são distintas e que cada peso tem sua própria distribuição de probabilidade, foi calculada a norma de cada matriz de peso (cada modelo tem 3 matrizes: W, U e b, respectivamente 3 normas para cada um dos 200 modelos). Plotando as normas em um espaço Tridimensional temos o resultado no Figura 4.32. Visualmente não há um padrão específico nas respectivas normas, assim como cada peso as normas dos modelos

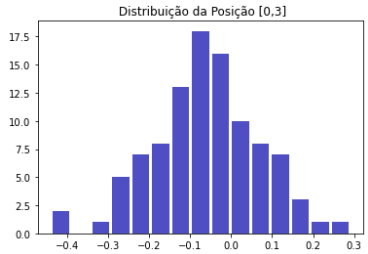


(a) *Pesos* $U_[63, 255]$ *Passeio Aleatório*

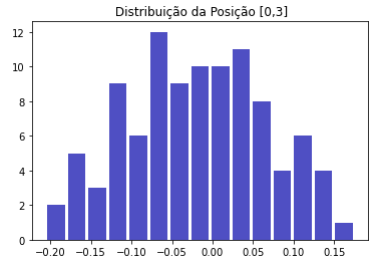


(b) *Pesos* $U_[63, 255]$ *Sequência Aleatória*

Figura 4.29: *Distribuição dos Pesos da Matriz U posição [63,255] para os 200 modelos gerados*

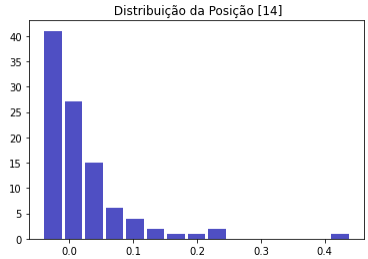


(a) *Pesos* $W[0, 3]$ *Passeio Aleatório*

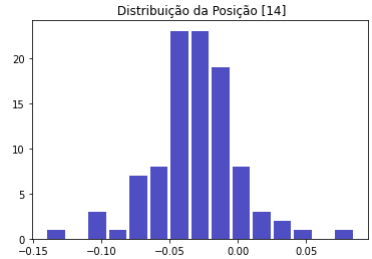


(b) *Pesos* $W[0, 3]$ *Sequencia Aleatório*

Figura 4.30: *Distribuição dos Pesos da Matriz W posição [0,3] para os 200 modelos gerados*



(a) *Pesos b[14] Passeio Aleatório*



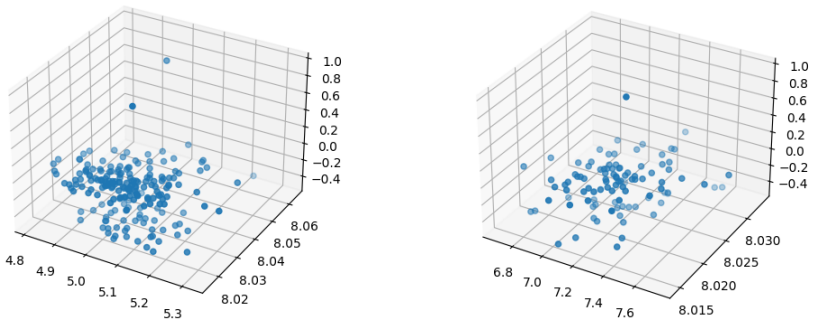
(b) *Pesos b[14] Sequência Aleatório*

Figura 4.31: *Distribuição dos Pesos do vetor b posição [14] para os 200 modelos gerados*

também têm suas respectivas distribuições de probabilidade (Veja Figura 4.33).

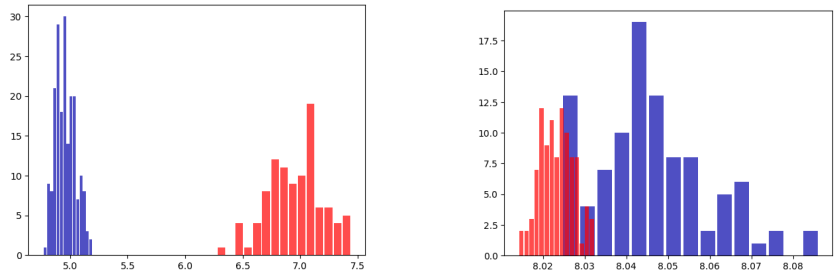
O mecanismo gerador de dependência tem influência no comportamento das matrizes de pesos dos modelos testados. Isto é, o tipo de dependência temporal contribui para a distribuição das matrizes de pesos(parâmetros) apesar das matrizes não convergirem para uma matriz única. O fato de cada modelo envolver a otimização de milhares de parâmetros ressalta a complexidade intrínseca dessas arquiteturas. Nesse contexto, a análise das distribuições das matrizes de pesos proporciona uma visão mais profunda sobre como diferentes padrões de dependência temporal moldam a estrutura dos modelos LSTM. Essas nuances são importantes para compreender as adaptações específicas de cada modelo diante dos desafios impostos pelos distintos cenários de Passeio Aleatório e Sequência Aleatória, oferecendo interpretações para aprimorar o entendimento e interpretação dessas arquiteturas em aplicações práticas.

A presente sessão teve foco minucioso no mecanismo de parâmetros, especificamente



(a) *Representação 3D Passeio Aleatório* **(b)** *Representação 3D Sequência Aleatória*

Figura 4.32: *Distribuição 3D das respectivas normas das matrizes U,W e b de Pesos dos 200 modelos gerados*



(a) *Representação das normas das matrizes W* **(b)** *Representação das Normas das matrizes b*

Figura 4.33: *Distribuição 2D das respectivas normas das matrizes W e b de Pesos dos 200 modelos gerados para ambos os cenários*

as matrizes de peso (W), matrizes de célula (U), e vetores de bias (b), em modelos LSTM, considerando-se os Cenários 1 e 2. A motivação para tal abordagem reside na compreensão das dinâmicas internas desses modelos em diferentes contextos. Cada cenário foi submetido a uma extensa avaliação por meio da construção de uma amostra composta por 200 modelos LSTM únicos. Para cada modelo, as matrizes W, U e o vetor b foram extraídos e submetidos a uma análise detalhada. Notavelmente, observou-se que os pesos não convergiam entre si dentro de cada cenário, indicando uma diversidade intrínseca nos padrões de aprendizado.

De maneira mais intrigante, constatou-se que os parâmetros divergiam significativamente ao serem comparados entre os dois cenários propostos. A variação nas configurações dos modelos LSTM gerou matrizes de peso distintas, evidenciando que a natureza dos dados influencia diretamente na estrutura dos parâmetros. Como parte do processo de análise, calculou-se a norma euclidiana das matrizes W, U e b para cada modelo. Esses resultados foram então visualizados por meio de gráfico em 3D (Figura 4.34, proporcionando uma representação espacial das diferenças e similaridades entre os parâmetros nos cenários de Passeio Aleatório e Sequência Aleatória.

Em síntese, esta investigação oferece insights sobre a variabilidade e adaptabilidade dos parâmetros em modelos LSTM, destacando a importância de considerar o contexto

específico do conjunto de dados ao configurar e interpretar tais arquiteturas. Essa compreensão refinada contribui para a melhoria do desempenho e interpretabilidade desses modelos em diferentes domínios de aplicação.

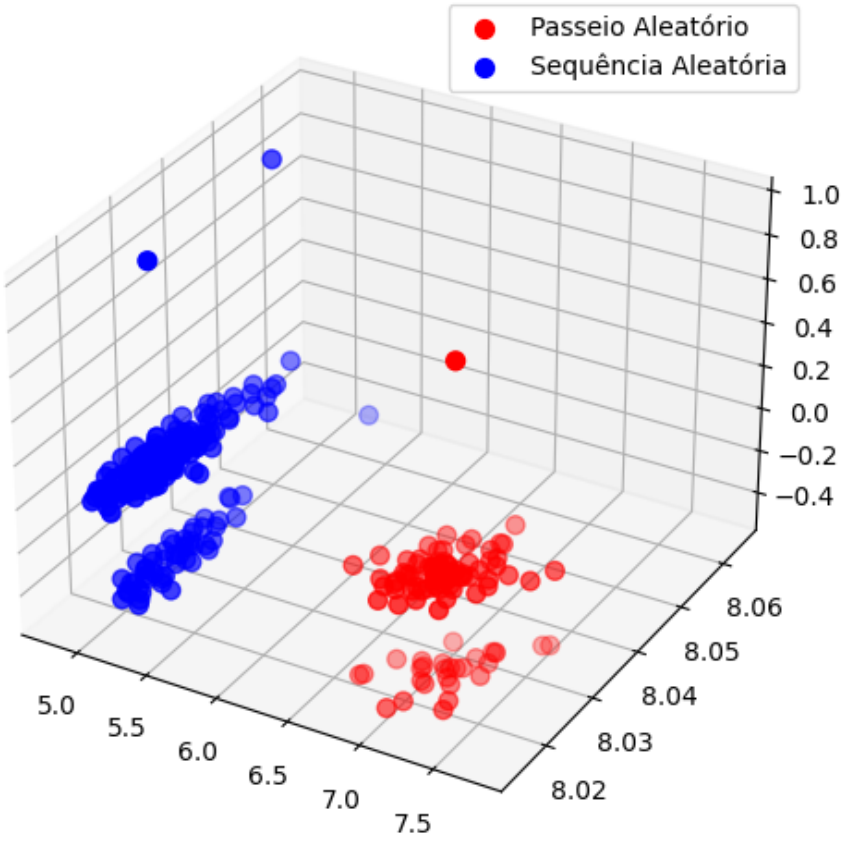


Figura 4.34: *Normas do cenários Passeio Aleatório e Sequência Aleatória em 3D*

Capítulo 5

Conclusão

5.1 Comparação entre as abordagens

O campo de Deep Learning tem ganhado cada vez mais áreas distintas de aplicações, dada e versatilidade das estruturas dos algoritmos propostos, é esperado aumento contínuo da aplicação destes modelos frente diferentes esferas de estudo. Em séries temporais há um universo considerável a ser explorado por estes algoritmos que vêm surpreendendo positivamente diversas áreas com suas aplicações e nova metodologia de captar padrões nos dados em análise. A notoriedade destes algoritmos em diferentes aplicações reforça a importância e o valor que podem agregar frente múltiplas áreas do conhecimento, e em especial séries temporais.

Conforme mencionado anteriormente a competição mais famosa do mundo de previsões e séries temporais "Makridakis Competitions, 2022"(Competição é uma homenagem a Makridakis, referência na área de Séries Temporais) obteve um algoritmo híbrido (abordagem convencional de Séries temporais combinado com algoritmos de Deep Learning) como campeão da sua mais recente competição. Algoritmos híbridos consideram estruturas de Deep Learning combinadas com modelos clássicos de séries temporais. O resultado da aplicação pode variar de acordo com cenário considerado, mas a tendência é de cada vez haja maior cooperação de Redes Neurais profundas no ramo de séries temporais. A abordagem proposta por Deep Learning é completamente diferente da abordagem clássica (modelos ARIMA, por exemplo). Esta sessão destaca as principais diferenças entre essas abordagens:

- **Estimação dos parâmetros**: a grande maioria dos modelos clássicos são estimados através de máxima verossimilhança ou por meio do método dos mínimos quadrados, uma vez que os modelos de Deep Learning são estimados por intermédio de retropropagação (Backpropagation), sendo as Redes Neurais Recorrentes utilizam a retropropagação no tempo (BBTT) para estimação dos parâmetros do modelo. Adicionalmente, os modelos de Deep Learning têm uma estrutura não linear em sua arquitetura (funções de ativação) que difere também dos algoritmos convencionais de séries temporais.
- **Hiperparâmetros**: Na abordagem convencional não há o conceito de hiperparâ-

metros, que controlam todo o processo de aprendizagem do algoritmo proposto. Os modelos de redes neurais além de obterem parâmetros também são compostos por hiperparâmetros que podem mudar de acordo com a estrutura proposta para o cenário de aplicação;

- **Quantidade de parâmetros**: a quantidade de parâmetros de uma rede neural profunda pode facilmente chegar no patamar de milhares, fato raro na abordagem tradicional de séries temporais. O grande número de parâmetros pode contribuir para o overfiting dos dados o que faz com que o processo de aprendizado atente ainda mais para a generalização do modelo proposto;
- **Flexibilidade dos Algoritmos**: ambas abordagens têm sua flexibilidade, no entanto os algoritmos de Deep Learning têm se mostrado extremamente flexíveis pois permitem adaptações completas em toda sua estrutura proposta. Os algoritmos convencionais permitem certas flexibilizações, mas são restritos a premissas e pressuposições sobre os dados impostas para a aplicação.
- **Suposições**: os algoritmos de Deep learning a princípio não impõem nenhuma suposição aos dados, diferente da abordagem clássica que, normalmente, há restrições sobre a distribuição dos resíduos e estacionariedade da série em análise. Modelos de Redes neurais estão sempre muito relacionados com a performance obtida e o respectivo poder de generalização nos dados de teste com base em métricas de performance.
- **Univariado e Multivariado**: um único modelo de Deep Learning pode ser aplicado para um conjunto de séries temporais, o que na abordagem clássica aumenta consideravelmente as restrições e suposições assim como a complexidade de processamento para casos multivariados. As Redes neurais profundas são mais flexíveis no cenário Multivariado, seja no contexto de prever uma única série de interesse com base em um conjunto de dados (Cenário 2 da sessão 4.3.2) ou a previsão de múltiplas séries de forma simultânea (Cenário 3 da sessão 4.3.2). O ponto de atenção é a escala das séries do conjunto Multivariado, sendo muito discrepantes pode ocorrer problemas de estimação para os modelos de Deep Learning.
- **Facilidade de Uso**: os modelos clássicos são muito populares e utilizados de forma bem abrangente devido, entre outros fatores, a facilidade de uso e a interpretabilidade dos modelos propostos. Modelos de Deep Learning, normalmente, não são interpretáveis, dado a complexidade da arquitetura, e o uso requer certo conhecimento aprofundado. Novas propostas, como Transformers por exemplo, têm proposto abordagens "interpretáveis" para algoritmos de Redes Profundas e vêm ganhando destaque neste sentido. De fato, quanto a facilidade de uso dos modelos de Deep Learning, há bastante espaço para evolução neste sentido frente aos algoritmos convencionais;
- **Dados de Treinamento**: Os algoritmos de Deep Learning têm preferência por maior quantidade de dados, no geral, quanto maior a quantidade de dados mais facilidade o modelo tem para captar os padrões de interesse. Em alguns casos, simulações de processos são utilizadas para contribuir frente a quantidade de informações disponíveis;

- **Custo de Treinamento**: O ajuste de um modelo de RNA dependendo da configuração dos Hiperparâmetros e Parâmetros pode levar horas de processamento em uma máquina comum, ao passo que os modelos tradicionais levam apenas minutos. Essa característica está diretamente relacionada com quantidade de Parâmetros e Hiperparâmetros do modelo RNN proposto;
- **Slide Window**: A janela de observação dos dados do passado é determinada no momento do ajuste dos modelos clássicos, já em Deep Learning o tamanho da janela deslizante pode ser um Hiperparâmetro, o que permite testar diferentes distâncias (Grid Search) no tempo para os dados e modelos ajustados.
- **Reshape dos Dados**: Os modelos de RNA, conforme sessão 4.1.2, não têm suposições ou restrições frente aos dados, porém é necessário adaptar os dados no formato adequado para que as respectivas arquiteturas sejam alimentadas com as informações respeitando a ordenação dos dados.

A metodologia de Deep Learning aplicado para séries temporais difere da abordagem convencional em muitos aspectos e especificidades. Utilizar um modelo clássico, conforme proposto no Capítulo 4, como baseline para aplicação de modelos de Redes Profundas, é sempre recomendável para que a avaliação de performance seja contemplada. Importante destacar que a combinação de modelos clássicos com modelos de Deep learning são possíveis e têm se destacado em competições conforme mencionado no início desta sessão. A aplicação de Deep Learning no campo de séries temporais tem sido motivo de estudo e desenvolvimento de literatura pois há muitas possibilidades de aplicação e adaptações possíveis. A sessão 5.3 destaca as "Áreas de Exploração e desenvolvimento" para modelos de Deep Learning no campo de séries temporais.

5.2 Aplicações de Deep Learning

Os modelos de deep Learning foram considerados em diferentes cenários de aplicação. Distintos conjuntos de dados também foram examinados nas aplicações: conjuntos univariados, assim como conjuntos Multivariados. Mesmo neste contexto, este texto não exauriu os diferentes cenários possíveis de aplicação, pois tem como proposta de análise os cenários mais convencionais, a saber: dados de contagem (Airline e Beer Production, dados de ações (GE) e o caso Multivariado (séries de Preços da GE e Volume de ações).

Em todos os contextos pelo menos uma das arquiteturas de Deep Learning obteve performance melhor quando comparado com os modelos Baselines (ARIMA para o caso univariado e VAR para o caso Multivariado). Apesar de melhor performance há uma variação no que diz respeito às arquiteturas dos modelos. A Classe Bidirectional obteve performance superior no contexto dos dados de contagem (Airline Passangers e Beer Production). Já o modelo ConvLSTM não obteve performance superior aos modelos Baseline em nenhum dos cenários propostos.

Para o caso das ações da GE, comportamento estocástico, o modelo LSTM simples se destacou pela performance, superando o modelo Bidirectional LSTM e o Baseline. No contexto Multivariado o modelo LSTM com uma camada Densa obteve melhor performance no exercício de prever o volume de Ações da GE com base as séries históricas do preço

(Cenário 3 da sessão 4.3.2).

Quanto maior a quantidade de dados melhor performance os modelos de Deep Learning, corroborando a origem e foco inicial destes algoritmos que foram planejados para situações com abundância de informação (BigData). Os casos de aplicações para séries temporais explorados, são casos em que não há grande número de dados (small Data), mas os modelos obtiveram performance melhor do que os modelos convencionais (Baseline) e a performance evoluiu conforme o número de registros considerados nos cenários de aplicação.

O processamento dos modelos pode levar muito tempo quando comparado a modelos baselines. Como diferentes Hiperparâmetros são considerados e trata-se de um processo de otimização, além da grande quantidade de parâmetros de cada arquitetura, são fatores que contribuem de forma negativa para o tempo de estimação e seleção de modelos. Nestes casos é indicado computadores com infraestrutura específica de GPU (Graphics Processing Unit) para dar celeridade no processamento e estimação dos modelos de Deep Learning.

No geral os modelos de Deep Learning se mostraram superior em performance, no entanto não há uma definição específica sobre qual arquitetura deva ser considerada frente ao cenário de aplicação. Há pesquisas acadêmicas no sentido de averiguar qual estrutura tem preferência a certos cenários, mas os resultados não são diretivos até o momento, este inclusive é um campo de estudo acadêmico que será destacado na próxima sessão sobre áreas a serem exploradas dentro do universo de Deep Learning aplicado a séries temporais.

Os cenários de aplicação demostraram que é possível aplicar modelos de Deep Learning no contexto de séries temporais e obter performance superior aos modelos clássicos, amplamente utilizados e conhecidos no ramo acadêmico e nos negócios. No geral, não há restrições ou suposições frente aos dados, mas em contrapartida é possível otimizar diferentes cenários para que se obtenha a performance desejada. Os modelos d RNA não têm interpretação intuitiva a não ser por intermédio de alguns Hiperparâmetros do modelo que permitem a compreensão de algumas características consideradas.

Houve problemas de estimação por parte dos algoritmos de Deep Learning no caso Multivariado onde a escala do conjunto de séries temporais era bem discrepante (dezenas versus milhares), Cenário 1 da sessão 4.3.2. Já a abordagem convencional (VAR) não obteve problemas similares de estimação e se mostrou mais eficiente neste tipo de contexto. A alternativa foi padronizar as Séries para um mesmo range de valores e reajustar os modelos (Cenário 3), neste caso os modelos de RNN obtiveram performance superior.

Os modelos de Deep learning obtiveram melhor performance no cenário Univariado, mas similar, de certo modo, aos modelos Baseline (ARIMA) ajustados (Sessão 4.2). O destaque ocorreu, de fato, no caso Multivariado com escalas semelhantes (Cenário 2 sessão 4.3.2). No caso multivariado em que há diferença de escala é necessário padronizar as variáveis e avaliar ambas as classes de modelos (Cenário 3 sessão 4.3.2).

5.3 Áreas de Exploração e desenvolvimento

O momento atual dos algoritmos de Deep Learning tem proporcionado considerável variedade de temas a serem explorados através de pesquisa e desenvolvimento no que diz respeito a aplicação dessas estruturas na modelagem de séries temporais. Esta sessão faz uma pequena descrição de alguns campos de pesquisa no que tange a aplicação de Redes Neurais Profundas para sequencias (os campos podem ser tanto para o caso numérico, séries temporais, ou textual):

- **Arquiteturas de Redes Neurais**: Dada a flexibilidade, há infinitas arquiteturas de Deep Learning que podem ser aplicadas para séries temporais, além de que frequentemente novas estruturas, completamente revolucionárias, são propostas (Transformers com mecanismo de Atenção é o exemplo mais recente). É possível explorar novas arquiteturas ou até mesmo a combinação de arquiteturas já existentes.
- **Algoritmo de Otimização**: No Capítulo 4 Adam foi o algoritmo utilizado para otimização dos processos de estimação dos parâmetros das diferentes estruturas propostas. Conforme mencionado na sessão 3.6.2 Adam por si só é a combinação de outros dois métodos de otimização. É possível propor melhorias no método já existente ou até mesmo propor novas abordagens de otimização e estimação dos parâmetros. Note que este cenário pode ser interpretado como um cenário de otimização e, desta forma, expandindo consideravelmente o campo de estudo.
- **Interpretação da estrutura**: A flexibilidade é uma característica super relevante de Deep Learning, mas em contrapartida a interpretação torna-se mais desafiadora dada a complexidade interna do processo como um todo. Novas abordagens (Transformers por exemplo) têm características que facilitam a questão de interpretação do processo e de suas diferentes etapas, mas esta é uma área que pode ser muito explorada pois na grande maioria dos casos baseia-se apenas nos resultados sem que seja possível a interpretação do processo/modelo;
- **Recomendação de Aplicação de acordo com cenário proposto**: dado a grande quantidade de possibilidades de arquiteturas não há de forma teórica uma prova definitiva de qual arquitetura tem melhor desempenho para um determinado cenário. Alguns artigos científicos trazem essa discussão para as LSTMs mas como descrito no Capítulo 3 há diversas variações e novas possibilidades, de modo que o caminho empírico por enquanto é recomendado. Neste contexto há campo de estudo para observar o comportamento de diferentes arquiteturas para diferentes cenários além de provas teóricas que corroborem as recomendações propostas;
- **Modelos Híbridos (Modelos Clássicos + Deep Learning)**: Esta classe de modelos tem ganhado importância frente a reputação que têm obtido em competições de Previsão Madrakaris (M5, campeão da competição de 2022). Com a flexibilidade dos modelos de Deep Learning os modelos híbridos acabam sendo também muito flexíveis e podem contemplar os benefícios de ambas as abordagens.
- **Small Data**: Os algoritmos de Deep Learning têm preferência pelo universo de "Big Data" (Grande quantidade de Dados), no entanto quando não há uma grande quantidade de Dados ou até mesmo uma pequena quantidade (Small Data) é um

cenário de estudo e análise de como estes algoritmos tendem a se comportar nesses cenários;

- **Estimativa Intervala**r: O foco desta dissertação é estimativa pontual das séries em questão. Há oportunidade de explorar estimativas intervalares para estas abordagens baseadas em Deep Learning. Uma vertente de pesquisa a ser explorada conforme a popularização desse tipo de algoritmo para os cenários de Séries temporais.
- **Split dos Dados**r:Na prática de modelagem estatística, especialmente no que diz respeito a modelos de regressão e classificação, é um padrão estabelecido e recomendável dividir o conjunto de dados em subconjuntos distintos para treinamento, validação e teste. Essa metodologia é empregada com o objetivo principal de evitar o overfitting, uma condição na qual um modelo estatístico se adapta excessivamente aos dados de treinamento, prejudicando sua capacidade de generalização para novos dados. O overfitting é particularmente problemático, pois pode levar a previsões enganosas ou imprecisas quando o modelo é aplicado em condições reais. A complexidade da separação dos dados assume uma dimensão adicional quando se lida com séries temporais. Em tais casos, a seleção aleatória de pontos de dados para formar os conjuntos de treinamento, validação e teste não é viável devido à natureza sequencial e dependente do tempo das observações. A ordem cronológica dos dados é crucial e deve ser preservada para que o fenômeno em estudo seja compreendido e modelado corretamente. Assim, a segmentação do conjunto de dados deve garantir que a parte de teste consista nas observações mais recentes, refletindo a progressão natural do tempo. No que se refere aos dados de validação, quando incorporados, é igualmente importante que a sequência temporal seja mantida. A decisão estratégica entre utilizar duas divisões (treinamento e teste) ou três (treinamento, validação e teste) é passível de exploração em conjuntos de dados menores. Isso porque pode surgir um dilema onde um modelo exibe um desempenho robusto no conjunto de validação, mas falha em manter essa performance no conjunto de teste. Para mitigar essa questão, uma abordagem comum pode envolver uma separação inicial em três conjuntos, selecionar o modelo ou algoritmo mais promissor e, posteriormente, unificar os conjuntos de treinamento e validação para a estimação final, essencialmente revertendo para uma estrutura de duas divisões. No entanto, esse procedimento tem similaridades com a adoção inicial de apenas dois conjuntos, uma vez que os dados de validação eventualmente alimentam a fase de treinamento. É importante sublinhar que a prioridade constante é prevenir o overfitting, assegurando que os modelos ajustados sejam capazes de generalizar além dos dados apresentados inicialmente. Para as aplicações discutidas no Capítulo 4, optou-se por trabalhar com duas divisões, alocando 80% dos dados para treinamento e 20% para testes. Através da incorporação de diversos mecanismos de regulação e validação cruzada, foi possível desenvolver modelos que não somente evitam o overfitting, mas também demonstram uma habilidade notável de generalização. Estes modelos, portanto, oferecem uma expectativa confiável de desempenho em situações futuras e dados não observados, refletindo a eficácia das técnicas de modelagem aplicadas e a abordagem de validação adotada.

As áreas de pesquisa mencionadas nesta sessão são as mais evidentes, pois há muitas outras áreas de estudo frente a aplicação de Redes Neurais para Séries temporais que podem ser exploradas. O tema tem atraído o mundo acadêmico por novas descobertas teóricas,

assim como o mundo dos negócios por possibilidades de aplicação frente a resultados de performance alcançados.

Apêndice A

Código Python Aplicação Univariada

Esta sessão visa exemplificar o código python de aplicação de um modelo LSTM para uma série univariada. Por praticidade os dados são simulados pra que a reporodução/aplicação do código seja imediata. A preparação dos dados também é realizada e o procedimento descrito nas etapas do próprio código.

O código importa as bibliotecas necessárias, incluindo TensorFlow, NumPy e ferramentas de avaliação como "train test split"e mean average perceentage error (MAPE). Em seguida, dados de exemplo são gerados como uma função senoidal com ruído. Os dados são preparados em sequências de entrada (X) e saídas correspondentes (y). Posteriormente, esses dados são divididos em conjuntos de treino e teste. O modelo LSTM é construído utilizando a API Sequential do TensorFlow, composto por uma camada LSTM com 64 unidades seguida por uma camada densa para a saída.

O modelo é treinado usando o conjunto de treino, e a avaliação é realizada no conjunto de teste, calculando o MAPE. O código também inclui uma visualização gráfica que compara os dados reais com as previsões do modelo. Este código oferece uma implementação básica de um modelo LSTM para séries temporais univariadas utilizando TensorFlow em Python. É importante adaptar a arquitetura e os parâmetros conforme necessário para atender aos requisitos específicos da base de aplicação.

Programa A.1 Código Python Aplicação Univariada

```
1  # Importante as Bibliotecas e pacotes que serão utilizados na aplicação
2      import numpy as np
3      import tensorflow as tf
4      from tensorflow.keras.models import Sequential
5      from tensorflow.keras.layers import LSTM, Dense
6      from sklearn.model_selection import train_test_split
7      from sklearn.metrics import mean_average_percentage_error
8      import matplotlib.pyplot as plt
```

cont ⟶

⟶ *cont*

```
9
10    Geração de dados de exemplo
11      n_samples = 1000
12      time_steps = 10
13      data = np.sin(np.linspace(0, 2 * np.pi, n_samples)) + np.random.normal(0, 0.1, n_samples)
14
15    Preparação dos dados
16      X, y = [], []
17      for i in range(len(data) - time_steps):
18            X.append(data[i:(i + time_steps)])
19            y.append(data[i + time_steps])
20
21      X, y = np.array(X), np.array(y)
22      X = np.reshape(X, (X.shape[0], X.shape[1], 1))
23
24    Divisão dos dados em conjuntos de treino e teste
25      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
26
27    Construção do modelo LSTM
28      model = Sequential()
29      model.add(LSTM(64, input_shape=(time_steps, 1)))
30      model.add(Dense(1))
31      model.compile(optimizer=adam, loss=mse)
32
33    Treinamento do modelo
34      model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
35
36    Avaliação do modelo no conjunto de teste
37      y_pred = model.predict(X_test)
38      mape = mean_average_percentage_error(y_test, y_pred)
39      print(fMean Average Percentage Error on Test Set: {mse})
40
41    Visualização da previsão em comparação com os dados reais
42      plt.plot(y_test, label=Real Data)
43      plt.plot(y_pred, label=Predicted Data)
44      plt.legend()
45      plt.show()
```

Apêndice B

Código Python Aplicação Multivariada

Analogamente a sessão anterior mas para o caso Multivariado. É descrito código Python que modela Multiplas séries temporais para um algoritmo LSTM (conforma aplicação do capitulo 4). As sessões de Apêndice são versões simplificadas da aplicação deste texto e abordam apenas a camada LSTM. Outras camadas podem ser facilmente adicionadas ao algoritmo proposto (conforme descrito nos cenários de aplicação do capítulo 4).

O código cria três séries temporais correlacionadas, representando dados multivariados. Essas séries são combinadas em uma matriz data. Em seguida, os dados são preparados em sequências de entrada (X) e saídas correspondentes (y). Os dados são divididos em conjuntos de treino e teste, e o modelo LSTM é construído utilizando a API Sequential do TensorFlow. A camada LSTM é configurada para aceitar entradas tridimensionais, pois temos múltiplas séries temporais. O treinamento do modelo é realizado usando o conjunto de treino.

A avaliação do modelo é feita no conjunto de teste, calculando o erro médio quadrático (MSE) para avaliar o desempenho. O código também inclui uma visualização gráfica que compara as séries temporais reais com as previsões do modelo. Este código oferece uma implementação básica de um modelo LSTM para séries temporais multivariadas usando TensorFlow em Python.

Programa B.1 Código Python Aplicação Multivariada

```
1  # Instalando e carregando as bibliotecas necessárias
2     import numpy as np
3     import tensorflow as tf
4     from tensorflow.keras.models import Sequential
5     from tensorflow.keras.layers import LSTM, Dense
6     from sklearn.model_selection import train_test_split
7     from sklearn.metrics import mean_squared_error
8     import matplotlib.pyplot as plt
```

cont ⟶

⟶ *cont*

```
9
10      # Geração de dados de exemplo (3 séries temporais)
11      n_samples = 1000
12      time_steps = 10
13
14      # Criando três séries temporais correlacionadas
15      series1 = np.sin(np.linspace(0, 2 * np.pi, n_samples)) + np.random.normal(0, 0.1,
            n_samples)
16      series2 = np.cos(np.linspace(0, 2 * np.pi, n_samples)) + np.random.normal(0, 0.1,
            n_samples)
17      series3 = np.random.normal(0, 0.1, n_samples)
18
19      # Combinando as séries em uma matriz
20      data = np.column_stack((series1, series2, series3))
21
22      # Preparação dos dados
23      X, y = [], []
24      for i in range(len(data) - time_steps):
25          X.append(data[i:(i + time_steps)])
26          y.append(data[i + time_steps])
27
28      X, y = np.array(X), np.array(y)
29
30      # Divisão dos dados em conjuntos de treino e teste
31      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
32
33      # Construção do modelo LSTM para Séries Temporais Multivariadas
34      model = Sequential()
35      model.add(LSTM(64, input_shape=(time_steps, data.shape[1])))
36      model.add(Dense(data.shape[1]))
37      model.compile(optimizer=adam, loss=mse)
38
39      # Treinamento do modelo
40      model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
41
42      # Avaliação do modelo no conjunto de teste
43      y_pred = model.predict(X_test)
44      mse = mean_squared_error(y_test, y_pred)
45      print(fMean Squared Error on Test Set: {mse})
46
47      # Visualização da previsão em comparação com os dados reais
48      plt.figure(figsize=(12, 6))
49      for i in range(data.shape[1]):
50          plt.subplot(data.shape[1], 1, i+1)
51          plt.plot(y_test[:, i], label=fReal Data {i+1})
52          plt.plot(y_pred[:, i], label=fPredicted Data {i+1})
```

cont ⟶

```
    ⟶ cont
53          plt.legend()
54
55      plt.show()
```

Apêndice C

Código Python Aplicação Transformers

A Arquitetura Transformers tem se destacado e ganhado muita popularidade por intermédio do ChatGPT (Generative Pre-Trained Transformers). Um cenário a ser explorado é justamente aplicar esse tipo de arquitetura para prever sequências numéricas (conforme sugerido no Capitulo 5 como área de desdobramento).

O código apresenta uma aplicação básica de um modelo Transformer para séries temporais usando a biblioteca Transformers da Hugging Face.Este código serve como uma base simples para a aplicação de modelos Transformers em problemas de séries temporais, e a arquitetura do Transformer pode ser ajustada conforme necessário para problemas mais complexos. As 3 etapas de aplicação do modelo transformes para séries temporais são descritas abaixo:

1. Preparação dos Dados e Modelo: A classe TimeSeriesDataset é criada para manipular os dados temporais, enquanto TimeSeriesTransformer define a arquitetura do modelo Transformer adaptado para séries temporais. 2. Treinamento do Modelo: O código inclui a inicialização do modelo, otimização com AdamW e treinamento por um número especificado de épocas usando a perda de erro quadrático médio (MSELoss). 3. Previsão: Após o treinamento, uma nova sequência é gerada e passada pelo modelo para realizar a previsão. O resultado é exibido no final do código. Este é um exemplo básico e a arquitetura do Transformer pode ser ajustada de acordo com a complexidade do problema.

Programa C.1 Código Python Aplicação Transformers

```
# Gera dados de exemplo (substitua isso pelos seus próprios dados)
class TimeSeriesDataset(Dataset):
    def __init__(self, data, time_steps):
        self.data = data
        self.time_steps = time_steps

    def __len__(self):
```

cont ⟶

```
⟶ cont
            return len(self.data) - self.time_steps

        def __getitem__(self, idx):
            x = self.data[idx:(idx + self.time_steps), :]
            y = self.data[idx + self.time_steps, :]
            return torch.Tensor(x), torch.Tensor(y)

    # Modelo Transformer personalizado para séries temporais
    lass TimeSeriesTransformer(nn.Module):
        def __init__(self, input_size, output_size, num_layers=2, hidden_size=128, nhead=8):
            super(TimeSeriesTransformer, self).__init__()
            self.transformer = Transformer(
                d_model=input_size,
                nhead=nhead,
                num_encoder_layers=num_layers,
                num_decoder_layers=num_layers,
                dim_feedforward=hidden_size,
                output_dim=output_size
            )

        def forward(self, x):
            return self.transformer(x)

    # Parâmetros
     n_samples = 1000
     n_features = 3
     time_steps = 10
     batch_size = 32

    # Gera dados de exemplo e prepara para o modelo Transformer
        data = np.random.randn(n_samples, n_features)
        dataset = TimeSeriesDataset(data, time_steps)
        dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)

        # Inicializa o modelo
        model = TimeSeriesTransformer(input_size=n_features, output_size=n_features)
            optimizer = AdamW(model.parameters(), lr=1e-4)
            scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=0,
                num_training_steps=len(dataloader) * 10)

    # Treina o modelo
            num_epochs = 10
            for epoch in range(num_epochs):
                for batch_x, batch_y in dataloader:
                    optimizer.zero_grad()
                    output = model(batch_x)
```

cont ⟶

```
    ⟶ cont
53                      loss = nn.MSELoss()(output, batch_y)
54                      loss.backward()
55                      optimizer.step()
56                      scheduler.step()
57
58  # Gera uma nova sequência para prever (substitua isso pelos seus próprios dados)
59              new_sequence = np.random.randn(1, time_steps, n_features)
60              new_sequence = torch.Tensor(new_sequence)
61
62  # Faz a previsão
63              with torch.no_grad():
64                  model.eval()
65                  prediction = model(new_sequence)
66
67          print(Previsão:, prediction)
```

Referências

[Alammar 2018] Jay Alammar. *The Illustrated Transformer*. 2018 (citado na pg. 33).

[Y. Bengio 1994] Y. Bengio. *Learning long-term dependencies with gradient descent is difficult*. 1994 (citado na pg. 22).

[Yoshua Bengio *et al.* 2015] Yoshua Bengio, Aaron Courville e Ian Goodfellow. *Deep Learning*. 2015 (citado nas pgs. 13, 46).

[Bishop 2011] Christopher M. Bishop. *Solutions for Pattern Recognition and Machine Learning 1st*. 2011 (citado na pg. 13).

[Box e Jenkins 1970] George Box e Gwilym Jenkins. *Time Series Analysis: Forecasting and Control*. 1970 (citado nas pgs. 2, 4, 8).

[Bushaev 2018] Vitaly J. Bushaev. *latest trends in deep learning optimization*. 2018 (citado na pg. 48).

[Camargo e Souza 1996] Maria Emilia Camargo e Reinaldo Castro Souza. *Análise e Previsão de Séries Temporais: os Modelos Arima*. 1996 (citado nas pgs. 3, 4).

[Cho *et al.* 2014] Kyunghyun Cho *et al. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014 (citado nas pgs. 21, 25).

[Corrar e Theóphilo 2004] Luiz Corrar e Carlos Renato Theóphilo. *Pesquisa operacional para decisão em contabilidade e administração: contabilometria*. 2004 (citado na pg. 3).

[Devlin e Chang 2018] Jacob Devlin e Ming-Wei Chang. *Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing*. 2018 (citado na pg. 32).

[Duchi *et al.* 2011] John Duchi, Elad Hazan e Yoram Singer. *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization*. 2011 (citado na pg. 45).

[Gers *et al.* 2000] Felix A. Gers, Jürgen Schmidhuber e Fred Cummins. *Learning to Forget: Continual Prediction with LSTM*. 2000 (citado na pg. 20).

[Goodfellow *et al.* 2015] Ian Goodfellow, Aaron Courville e Yoshua Bengio. *Deep Learning*. 2015 (citado nas pgs. 15, 45).

[Graves 2009] Alex Graves. *Supervised Sequence Labelling with Recurrent*. 2009 (citado na pg. 20).

[Graves 2013] Alex Graves. *Generating Sequences With Recurrent Neural Networks*. 2013 (citado na pg. 20).

[Graves e Jaitly 2014] Alex Graves e Navdeep Jaitly. *Towards End-To-End Speech Recognition with Recurrent Neural Networks*. 2014 (citado na pg. 20).

[Greff *et al.* 2015] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink e Jürgen Schmidhuber. *LSTM: A Search Space Odyssey*. 2015 (citado na pg. 27).

[Hochreiter 1991] Sepp Hochreiter. *Fundamental Deep Learning Problem*. 1991 (citado na pg. 22).

[Hochreiter e Schmidhuber 1997] Sepp Hochreiter e Jürgen Schmidhuber. *Long Short-Term Memory*. 1997 (citado na pg. 20).

[Horn e Johnson 2012] Roger A. Horn e Charles R. Johnson. *Matrix Analysis*. 2012 (citado na pg. 16).

[Jozefowicz *et al.* 2015] Rafal Jozefowicz, Wojciech Zaremba e Ilya Sutskever. *An Empirical Exploration of Recurrent Network Architectures*. 2015 (citado nas pgs. 1, 27).

[Karpathy 2017] Andrej Karpathy. *A Peek at Trends in Machine Learning*. 2017 (citado na pg. 45).

[Kiros *et al.* 2014] Ryan Kiros, Ruslan Salakhutdinov e Richard S. Zemel. *Unifying Visual-Semantic Embeddings with Multimodal Neural Language Models*. 2014 (citado na pg. 21).

[Koutník *et al.* 2014] Jan Koutník, Klaus Greff, Faustino Gomez e Jürgen Schmidhuber. *A Clockwork RNN*. 2014 (citado na pg. 27).

[Krizhevsky *et al.* 2012] Alex Krizhevsky, Ilya Sutskever e Geoffrey E. Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. 2012 (citado na pg. 15).

[LeCun 1995] Yann LeCun. *Convolutional networks for images, speech, and time series*. 1995 (citado nas pgs. 15, 17).

[Morettin e Toloi 2006] Pedro A. Morettin e Clélia M. C. Toloi. *Análise de séries temporais*. 2006 (citado na pg. 4).

[Rumelhart *et al.* 1986] David E. Rumelhart, Geoffrey E. Hinton e Ronald J. Williams. *Learning internal representations by error propagation*. 1986 (citado na pg. 14).

[Russakovsky 2014] Olga Russakovsky. *ImageNet Large Scale Visual Recognition Challenge*. 2014 (citado nas pgs. 1, 15).

[Sermanet 2013] Pierre Sermanet. *Integrated Recognition, Localization and Detection using Convolutional Networks*. 2013 (citado na pg. 17).

[Simard 2003] Steinkraus P.Y. Simard. *Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. Proceedings of the 7th International Conference on Document Analysis and Recognition*. 2003 (citado na pg. 15).

[Smyl e Shanmugam 2017] Slawek Smyl e Santhosh Shanmugam. *Engineering Extreme Event Forecasting at Uber with Recurrent Neural Networks*. 2017 (citado nas pgs. ii, iii).

[Sutskever *et al.* 2014] Ilya Sutskever, Oriol Vinyals e Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*. 2014 (citado na pg. 20).

[Vinyals *et al.* 2014] Oriol Vinyals, Alexander Toshev, Samy Bengio e Dumitru Erhan. *Show and Tell: A Neural Image Caption Generator*. 2014 (citado na pg. 21).

[Xu e Zhu 2015] J Xu e L Zhu. *Temporal-Spatial Resolution Fate Mapping Reveals Distinct Origins for Embryonic and Adult Microglia in Zebrafish*. 2015 (citado na pg. 21).

[Yang 2018] Zhang X. Yang. *Temporal variation of SO2 emissions embodied in Chinese supply*. 2018 (citado na pg. 32).