PREVISÃO COM SÉRIES TEMPORAIS USANDO COMPUTAÇÃO POR RESERVATÓRIO - APLICAÇÃO EM ATIVO DO B3

MAYCON KAWLIN SARDÉVIST ALCÂNTARA E LIMA

TARCÍSIO MARCIANO DA ROCHA FILHO

DISSERTAÇÃO DE MESTRADO EM FÍSICA

DEPARTAMENTO DE PÓS-GRADUÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM FÍSICA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
PROGRAMA DE PÓS-GRADUAÇÃO EM FÍSICA
DEPARTAMENTO DE PÓS-GRADUÇÃO

TIME SERIES FORECASTING USING RESERVOIR COMPUTING - APPLICATION ON B3'S STOCK

PREVISÃO COM SÉRIES TEMPORAIS USANDO COMPUTAÇÃO POR RESERVATÓRIO - APLICAÇÃO EM ATIVO DO B3

MAYCON KAWLIN SARDÉVIST ALCÂNTARA E LIMA

TARCÍSIO MARCIANO DA ROCHA FILHO

ORIENTADOR: TARCÍSIO MARCIANO DA ROCHA FILHO, DR.

DISSERTAÇÃO DE MESTRADO EM FÍSICA

PUBLICAÇÃO: PPGEA.TD-001/11

BRASÍLIA/DF: JUNHO - 2021

UNIVERSIDADE DE BRASÍLIA
PROGRAMA DE PÓS-GRADUAÇÃO EM FÍSICA
DEPARTAMENTO DE PÓS-GRADUÇÃO

PREVISÃO COM SÉRIES TEMPORAIS USANDO COMPUTAÇÃO POR RESERVATÓRIO - APLICAÇÃO EM ATIVO DO B3

MAYCON KAWLIN SARDÉVIST ALCÂNTARA E LIMA

TARCÍSIO MARCIANO DA ROCHA FILHO

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO DEPARTAMENTO DE PÓS-GRADUÇÃO DO PROGRAMA DE PÓS-GRADUAÇÃO EM FÍSICA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE.

APROVADA POR:

Prof. Dr. Tarcísio Marciano da Rocha Filho – IF/Universidade de Brasília
Orientador

Prof. Dr. Jhon H. Watson – Dep./Universidade
Membro Interno

Dr. Evil – Dep./Universidade
Membro Externo

BRASÍLIA, 28 DE JUNHO DE 2021.

FICHA CATALOGRÁFICA

LIMA, MAYCON; ROCHA, TARCÍSIO
PREVISÃO COM SÉRIES TEMPORAIS USANDO COMPUTAÇÃO POR RESERVATÓRIO - APLICAÇÃO EM ATIVO DO B3 [Distrito Federal] 2021.
x, 67p., 210 x 297 mm (Mestrado/Instituto de Física/UnB, Mestre, Física, 2021).
Dissertação de Mestrado – Universidade de Brasília, Programa de Pós-Graduação em Física.
Departamento de Pós-gradução

1. Palavra 1	2. Palavra 2
3. Palavra 3	4. Palavra 4
I. Mestrado/Instituto de Física/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

LIMA, M. K. S. A. E, DA ROCHA, T. M. (2021). PREVISÃO COM SÉRIES TEMPORAIS USANDO COMPUTAÇÃO POR RESERVATÓRIO - APLICAÇÃO EM ATIVO DO B3 . Dissertação de Mestrado em Física, Publicação PPGEA.TD-001/11, Departamento de Pós-gradução, Universidade de Brasília, Brasília, DF, 67p.

CESSÃO DE DIREITOS

Departamento de Pós-gradução (Mestrado) - Instituto de Física
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

Dedicatória escrita pelo primeiro autor: Quero agradecer aos meus pais pelo apoio, ao CNPq pela ajuda financeira, ao meu orientador Tarcísio Marciano da Rocha Filho pelo aprendizado, ao colega de estudos Israel pelo desenvolvimento em conjunto desta dissertação, e a todos que contribuíram de alguma maneira para obtenção desta titulação.

Maycon Kawlin Sardévist Alcântara e Lima

ACKNOWLEDGMENTS

RESUMO

Título: Previsão Com Séries Temporais Usando Computação Por Reservatório - Aplicação em Ativo do B3
Autor: Maycon Kawlin Sardévist Alcântara e Lima
Autor: Tarcísio Marciano da Rocha Filho
Orientador: Tarcísio Marciano da Rocha Filho, Dr.
Programa de Pós-Graduação em Física
Brasília, 28 de junho de 2021

Esta dissertação trata da aplicação de ferramentas de Machine Learning, mais especificamente de redes neurais recorrentes (Echo State Network-ESN e Deep Echo State Network-DESN), na previsão do preço do ativo da Petróleo Brasileiro S.A. (PETR4), uma empresa brasileira de capital aberto cujo acionista majoritário é o Governo do Brasil. São utilizados diferentes métodos como variação diária do logaritmo do preço, uso de diferentes séries temporais e análise da componente principal (PCA). Os resultados mostraram que o uso desses três métodos em conjunto geram resultados satisfatórios, além de demonstrar uma superioridade da rede neural DESN.

Palavras-chave: Machine Learning, ESN, DESN, PETR4, logaritmo do preço, séries temporais, PCA.

ABSTRACT

Title: Time Series Forecasting Using Reservoir Computing - Application on B3's Stock
Author: Maycon Kawlin Sardévist Alcântara e Lima
Author: Tarcísio Marciano da Rocha Filho
Supervisor: Tarcísio Marciano da Rocha Filho, Dr.
Graduate Program in IF
Brasília, June 28th, 2021

This dissertation deals with the application of Machine Learning tools, more specifically of recurrent neural networks (Echo State Network-ESN and Deep Echo State Network-DESN), in the intent of forecast the asset price of Petróleo Brasileiro SA (PETR4), a Brazilian company of publicly traded company whose majority shareholder is the Government of Brazil. Different methods are used, such as daily variation of the price's logarithm, use of different time series and principal component analysis (PCA). The results showed that the use of these three methods together generate satisfactory results, besides that are demonstrating a superiority of the DESN neural network.

Keywords: Machine Learning, ESN, DESN, PETR4, price's logarithm, time series, PCA.

SUMÁRIO

LISTA DE FIGURAS

1 INTRODUÇÃO

Boa parte dos fenômenos da natureza são descritos por algum tipo de dinâmica temporal, como por exemplo o clima, fluxo de fluídos, movimento de partículas em um gás, infecção de doenças, disseminação de informação, tráfego de automóveis, preço de uma ação, entre outros fenômenos. O desafio se encontra na modelagem física e matemática desses fenômenos, seja por sua complexidade de dinâmica caótica ou pela quantidade de equações que devem ser solucionadas pela presença de muitos corpos. Muitos trabalhos lidam com essas complexidades como por exemplo em **Hethcote[6]**, onde descreve modelos epidêmicos e endêmicos como o SIR, MSEIR e SEIR para doenças como o sarampo, ou em **Kanazawa Takumi Sueshige e Takayasu[7]**, que descreve a dinâmica do livro de ordens de uma bolsa de valores utilizando teoria cinética, mais especificamente, utilizando as equações de Boltzmann e Langevin. Podemos citar ainda, **Packard J. P. Crutchfield e Shaw[8]** que se trata sobre atratores encontrados em sistemas caóticos de fluxo de fluídos e climas, bem como **Young[9]**, que aborda sobre arbitragem no câmbio de moedas utilizando teoria de calibre em rede.

Cálculos complexos e simulações demoradas limitam a velocidade de obtenção de resultados nessas áreas. Contudo nos últimos anos, há uma onda crescente de pesquisas nessas áreas utilizando uma ferramenta computacional que só se tornou possível recentemente, devido ao desenvolvimento tecnológico atrelado ao poder computacional, o aprendizado de máquinas (Machine Learning - ML). Essa ferramenta permite que com dados o suficiente, sejam feitas classificações, regressões e previsões acerca de um fenômeno com porcetagens de erros mínimas. No artigo **Li Nikola Kovachki e Anandkumar[10]**, é demonstrado que uma rede neural, Operador Neural de Fourier, consegue prever os fenômenos descritos pelas equações de Burgers, Darcy flow e de Navier-Stokes até 3 vezes mais rápido do que as solucionando computacionalmente. Em **Sheta e Faris[11]** é feito um comparitivo entre regressão linear multipla, rede neural artificial (Artificial Neural Network - ANN) e Support Vector Machines (SVM), onde este demonstra um melhor resultado na previsão do índice S&P 500. **Das e Ghosh[12]** aplicam técnicas de redes neurais em previsões metereológicas em regiões da Índia, e tem-se ainda o trabalho de **Filho e Rocha[13]** no qual realizam uma modelagem estatística sobre a previsibilidade do mercado brasileiro bem como previsões com uma rede neural Feed Foward (FFNN).

Adentrando nas necessidades da bolsa valores, vários trabalhos foram e estão sendo desenvolvidos na área de redes neurais, para facilitar na tomada de decisão quanto a compra ou venda de um ativo, índices, commodities, entre outros meios de se investir. Em **Chandra[14]**, o autor utiliza uma rede neural Bayesiana para prever ações antes e durante a pan-

demia causada pela COVID-19. Seus resultados mostraram que ao considerar os dados que contém a reação do mercado à pandemia durante o treinamento, as previsões obtiveram uma grande melhora comparada as previsões realizadas logo no inicio da pandemia. Em **Zheng e He[15]**, é utilizado a técnica de redução dimensional de dados (Principal Component Analisys - PCA), apresentando melhor acurácia e eficiência na previsão do preço da ação de duas companhias aéreas durante a pandemia de COVID-19, ou seja, considerando uma grande volatilidade. Além disso, o artigo ainda descreve que a entrada de dados pode ser maior na medida em que os dados são mais estáveis, sendo melhor considerar menos entradas quando se trata de ações de mercado com altas flutuações. Em **Ghashami, Kamyar e Riazi[16]**, foi realizado a previsão do índice NASDAQ utilizando uma FFNN em conjunto com um método de optmização de algoritmos, o Particle Swarm (Enxame de Partículas). Os resultados mostraram que a rede neural em conjunto com o método de optimização melhorou a acurácia da previsão apesar de requerer um tempo de execução maior.

Mas mesmo redes neurais podem requerer muito poder computacional para o treinamento e previsão de séries temporais caóticas, e quando se trata de mercado financeiro, tempo é dinheiro. Em **Wang et al.[17]**, é abordado uma nova técnica de ML na qual é utilizado um reservatório de neurônios conectados entre si aleatoriamente para o processamentos de dados, conhecido como computação de reservatório (Resevoir Computer - RC). Neste, artigo é apresentado que RC tem um melhor desempenho na previsão de índices do mercado financeiro quando comparado à modelos como LSTM(Long Short Term Memory) e redes neurais recorrentes (Recurrent Neural Networks - RNN). Além disso, devido a estrutura do RC, o tempo e esforço computacional de treinamento são drasticamente reduzidos. Em **Mallya[18]**, é feito um comparativo da Echo state Network, modelo de RC, com a LSTM, apresentando uma grande diferença na eficiência e acurácia durante o treinamento e previsão de diferentes ações como Microsoft Corporation (MSFT). Em **wyffels e Schrauwen[19]**, é utlizado técnicas de escalonamento, como decomposição sazonal mensal, para obter um melhor treinamento e melhorar a acurácia das previsões com RC. Em **Trierweiler Ribeiro et al.[20]**, é demonstrado que um modelo de RC, Echo State Network (ESN) em conjunto com um algoritmo de otimização, o Particle Swarm, e o modelo de autoregressão heterogênia (Heterogeneous AutoRegressive - HAR), apresenta melhores resultados que modelos como perceptron de múltiplas camadas (Multilayer Perceptron - MLP), na previsão de volatilidade de diferentes ativos para diferentes faixas de tempo de previsão (1,5 e 21 dias a frente). Em **Kim e King[21]**, é proposto uma evolução na estrutura da rede neural ESN, a Deep Echo State Network (DESN). Essa nova estrutura contém reservatórios interligados, que apresentam melhores resultado de previsão em diferentes séries temporais, como dados de temperatura, ações e dados caóticos simulados.

Neste trabalho aplicamos redes neurais recorrentes para a previsão de séries temporais. Talvez o maior desafio para essa abordagem seja aplica-la a séries temporais financeiras, dada

a grande quantidade de fatores que as afetam. O objetivo será demonstrar um comparativo entre ESN a DESN , bem como as possibilidades de lucro com o uso dessas ferramentas em conjunto com algoritmos de optimização e ideias de transformações de dados. No capítulo **2**, será realizado uma discussão sobre a existência de padrões e memórias no mercado financeiro que possibilitem a previsão no preço de uma ação. No capítulo **3**, será discutido sobre redes neurais em geral, começando pelo perceptron, depois evoluindo para as Feed Forwards Networks (FFNN) e por fim para Redes Neurais Recorrentes (RNN). Posteriormente, será descrito no capítulo **4**, as metodologias utilizadas para encontrar melhores previsões. No capítulo **5** teremos uma análise dos resultados encontrados de acordo com cada metodologia utilizada, e por fim, a conclusão no capítulo **6** sobre todo o trabalho aqui descrito.

2 PADRÃO E MEMÓRIA

É necessesário entender como funciona as distribuições estatísticas em um mercado financeiro. Como mostram em **Mantegna e Stanley[22]**, as distribuições que as mudanças de preços seguem, nem sempre são distribuições gaussianas ao levar em conta os diferentes tempos de registro de dados (segundos, dias, semanas, e assim por diante). Para isso, é necessário recorrer a outras distribuições como a distribuição de Lèvy, que por sua vez também não se ajusta totalmente a curva, como podemos ver na imagem a seguir

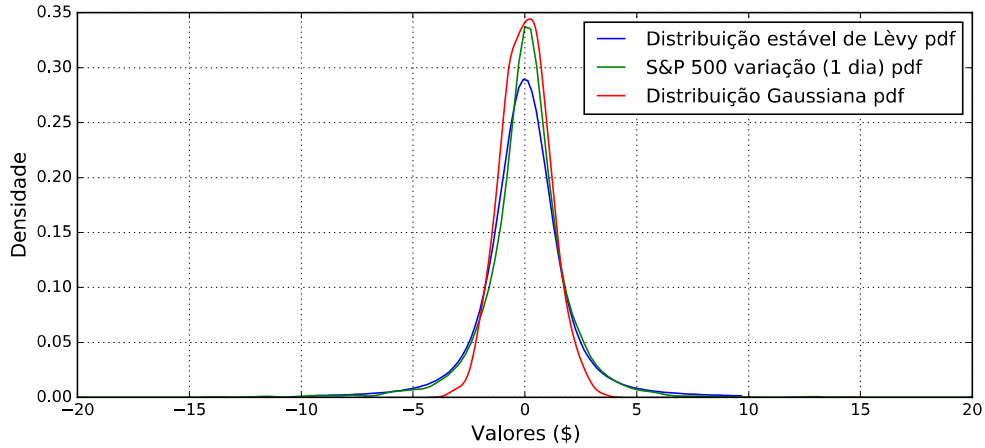

Figura 2.1 – Comparação da Função Distribuição de Propabilidade (PDF) da variação diária do fechamento da ETF S&P 500, com a distribuição estável de Lèvy ($\alpha = 1.4$, $\gamma = 0.00375$) e a distribuição Gaussiana ($\mu = 0$,$\sigma = 1.1$).

A distribuição de Lèvy é uma generalização do teorema do limite central que permite com que a variância da distribuição seja infinita, dando o aspecto de caudas mais longas quando comparada a distribuição gaussiana, se ajustando bem a distribuição das variações do S&P 500 na parte inferior. Dessa forma, **Mantegna e Stanley[22]** traz a ideia de reescalonar os dados como por exemplo, o uso de logaritmo nos dados e da seguinte equação que é uma definição da reescalonação da distribuição de Lévy.

$$\tilde{P}(\tilde{Z}) = \frac{P(Z)}{(\Delta t)^{-1/\alpha}} \tag{2.1}$$

onde $Z(t) = Y(t + \Delta t) - Y(t)$ ou variação do preço, $P(Z)$ é a PDF (Função Distribuição de Probabilidade) e o símbolo $\tilde{Z}$ significa a escalonação de Z. Podemos ver seus efeito no gráfico **2.2**, o que pode indicar a presença de padrões na mudança de preços se for feito as transformações apropriadas aos dados.

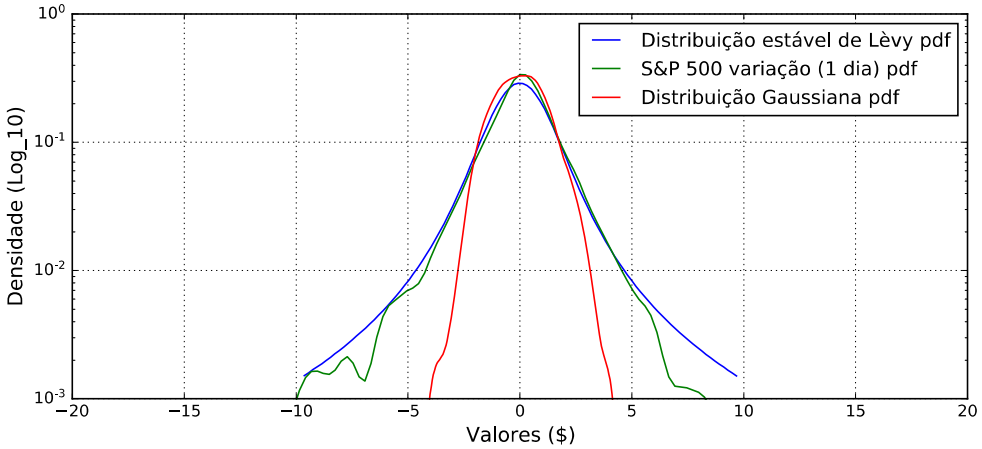

Figura 2.2 – Comparação da Função Distribuição de Propabilidade (PDF) da variação diária do fechamento da ETF S&P 500, com a distribuição estável de Lèvy ($\alpha = 1.4$, $\gamma = 0.00375$) e a distribuição Gaussiana ($\mu = 0$,$\sigma = 1.1$) em escala logaritmica.

Vemos aqui que ambas as distribuições, Lèvy e Gaussiana, se ajustam bem para valores pequenos, mas somente a distribuição de Lévy apresenta um bom ajuste nas laterais/caudas. Vamos entender um pouco melhor como funciona a reescalonação com a figura **2.3**. Podemos ver o plote do preço de fechamento da ação PETR4 (Petróleo Brasileiro S.A. ou Petrobras) em função do tempo, bem como a sua variação diária também ao longo do tempo. Essa é uma empresa de sociedade anônima cujo acionista majoritário é o governo do Brasil, sendo sua série temporal a ser estudada daqui em diante neste trabalho, devido a sua forte

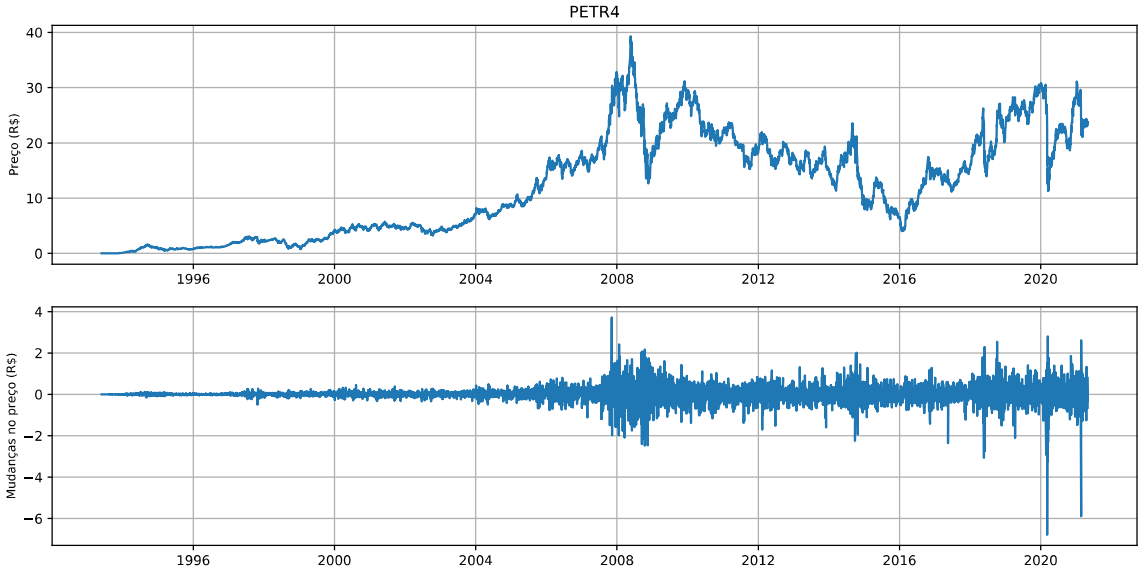

Figura 2.3 – O gráfico superior representa o preço em Reais (R$) do fechamento da ação PETR4 na bolsa de valores Ibovespa ao longo do tempo em anos. O gráfico inferior, representa a variação diária do gráfico superior ao longo do tempo em anos.

susceptibilidade a situações governamentais, o que dificulta a análise realizada por redes

neurais, e podendo assim ser observado o quão eficiente é o desempenho.

Os dados se estendem desde 26 de maio de 1993 até 30 de abril de 2021, e se nota que a maioria das característica proeminentes da ação se observam apenas a partir de 2006, uma vez que os valores referentes aumentaram com o passar dos anos. Agora, vamos plotar o gráfico com uma certa transformação, figura **2.4**.

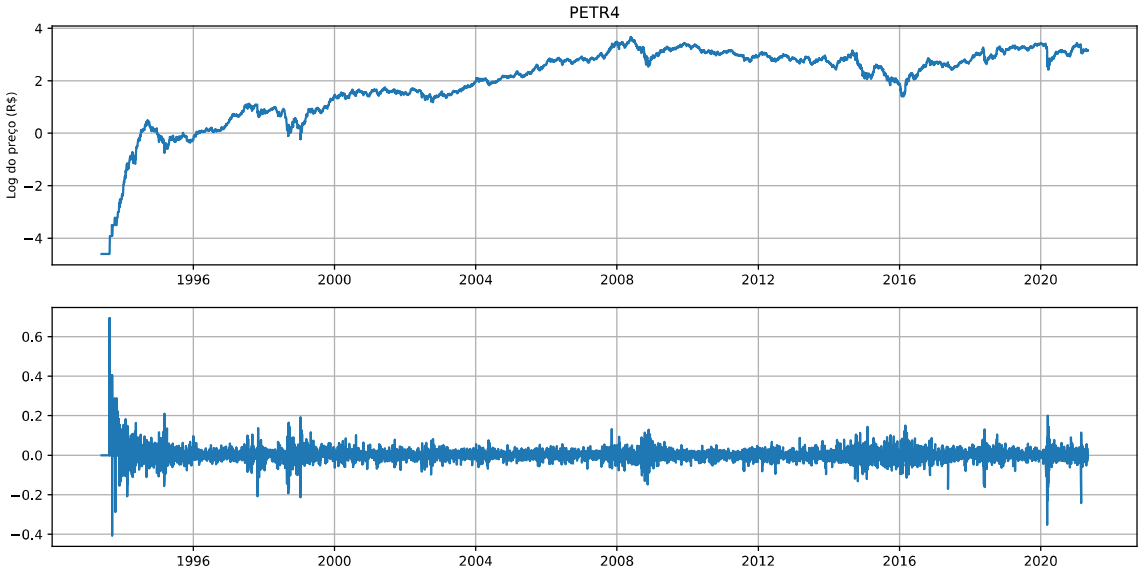

Figura 2.4 – Este é o mesmo gráfico da figura **2.3**, porém em escala logaritmica

Este é o mesmo gráfico de **2.3**, mas em escala logaritmica, ou seja, a mudança em 1% no preço em 1996 irá parecer o mesmo que a mudança no preço em 1% em 2021. A escala logaritmica reescalona os dados de tal forma que as mudanças nos valores são equalizadas independente dos valores. Podemos observar as grandes mudanças no preço em 1994/1995 devido ao plano real aplicado no governo do então presidente Fernando Henrique Cardoso, ainda se pode notar uma maior volatilidade em 2016 devido a crise financeira brasileira no governo de Dilma Rousseff. A mais, temos o impacto da pandemia no início de 2020, com grandes oscilações em poucos dias.

Contudo, muitos discutem a relevância do trabalho de Bachelier na Finança moderna, sendo a mesma a base para tal. Louis Bachelier (1870-1946), foi um matemático francês e o primeiro a modelar o mercado financeiro em sua tese de Doutorado (Teoria da Especulação) publicada em 1900, porém seus trabalhos não tiveram reconhecimento em vida por se tratar de uma aplicação matemática nunca realizada anteriormente. Seu trabalho sugere que a variação do preço de uma ação segue um comportamento bem semelhante ao movimento browniano. Um movimento browniano, é movimento aleatório de tempo contínuo sendo considerado um processo estocástico, ou melhor descrevendo, um processo estocástico é aquele que possui inúmeras soluções, ao contrário de processos determinísticos que podem ser regidos por equações diferenciais ordinárias possuindo soluções únicas.

Com isso se estabelece algumas hipóteses a respeito, a primeira hipótese é a de que a mudança no preço é independente da última, ou seja, o preço da semana passada ou do mês passado não influenciam nos dias atuais. Dessa forma, não há necessidade de estudar dados históricos para prever o preço de amanhã. A segunda hipótese estabelece a estacionariedade estatística (série temporal na qual o comportamento se limita ao redor de uma constante) da variação do preço, isso significa que o que gera as mudanças permanece o mesmo durante todo o tempo, assim como o movimento browniano. E a terceira hipótese, é a distribuição ao qual a mudança nos preços seguem, a distribuição normal. Apesar de serem boas, essas hipóteses são idealizações e se sabe que o mundo real é mais complicado, embora se veja um seguimento claro na continuação de publicações nesta vertente, como por exemplo as adaptações realizadas na equação de Black-Scholes, **Huang e Zhao[23]**.

O fato é, muitos estudos sugerem que se um índice cai em um determinado período de tempo as chances de caírem novamente são ligeiramente maiores. Caso subam, maiores serão as chances de continuarem a subir, o que quebra com a primeira hipótese de movimento browniano. A estacionariedade de um movimento browniano é diferente da estacionariedade observada na variação do preço de uma ação, assim como a distribuição real de preços não seguem uma curva normal como estabelecido na terceira hipótese, o que é apresentado nas figuras **2.1** e **2.5**, respectivamente.

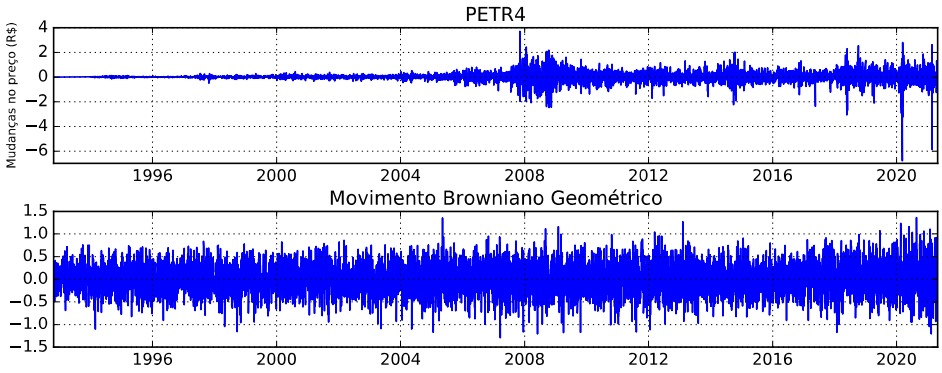

Figura 2.5 – Original vs. reprodução. Estes são plotes da variação diária da PETR4 e uma simulção do Movimento Browniano Geométrico

É possível visualizar a diferença entre as variações do preço de uma ação e de um movimento browniano, em uma ação as variações não são consistentes e parecem não ser aletórias, visto que para cada período de grande volatilidade se tem uma influência externa, seja financeira, política ou social. Podemos ver isso mais claramente na figura abaixo

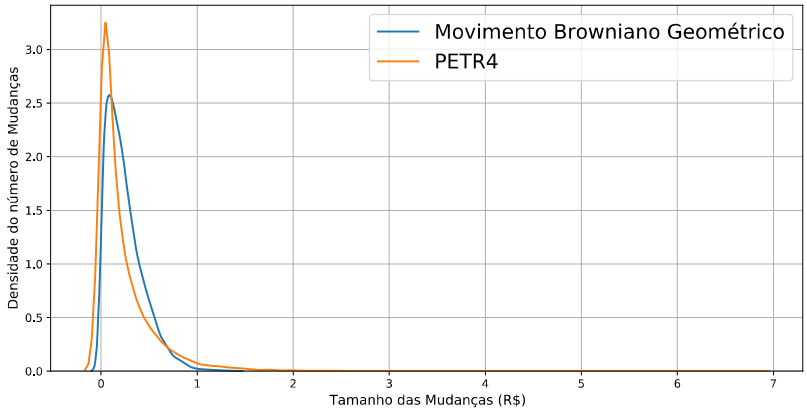

Figura 2.6 – Sobreposição distribuição da densidade do número de mudanças no preço em função da mudança no preço para a ação PETR4 e Movimento Browniano Geométrico (Processo estocástico no qual o logaritmo da variável segue um movimento browniano).

No figura **2.6**, temos a sobreposição da distribuição da densidade do número de mudanças no preço para a ação PETR4 e o Movimento Browniano Geométrico. Os valores estão ranqueados pelo tamanho do valor da variação diária, ao invés da ordem cronológica. Ou seja, as variações de mesmo valor são contabilizadas pelo eixo das ordenadas (Y), e as maiores variações, sejam postivias ou negativas, são apresentadas a direita do gráfico enquanto que as menores se concentram à esquerda. Vê-se que as variações no modelo browniano se limitam rapidamente em torno $R\$1,50$, que é mais ou menos 7σ (desvio padrão), enquanto que nas variações da PETR4 temos 21σ o tamanho do desvio padrão. Essa é a "Fat Tail"ao qual os estatísticos se referem, provando assim que o modelo padrão baseado no movimento browniano é errado. Mandelbrot destacou estes pontos em seu livro **Mandelbrot e Hudson[2]**.

É comumente encontrado na física, séries de pontência que governam fenômenos naturais, como por exemplo a escala *Rischter*, onde expressa que pequenos terremotos são comuns enquanto que os grandes são raros, com uma fórmula precisa que relaciona intensidade com frequência. Pareto realizou o mesmo quando estudou a distribuição de renda entre as sociedades em geral e em diferentes eras. Ele chegou em algo que sugere uma desigualdade de distribuição de renda, um "arco social"ao invés de uma distribuição piramidal como mostra a figura **2.7**.

Na época de Pareto, os eixos dos gráficos eram invertidos, mas independentemente, nota-se que a elite se concentra na parte mais fina da curva enquanto que massa populacional se concentra na parte mais larga. O ideal seria que a parte larga se deslocasse para cima, no caso de pareto ou para a direita no caso da distribuição de renda brasileira. Além disso, é visível que uma série de potencia está presente, Pareto calculou a inclinação da curva sendo como $\alpha = 3/2$. Isso significaria o que, se o menor ganho de uma pessoa seja por volta de $R\$4.60$

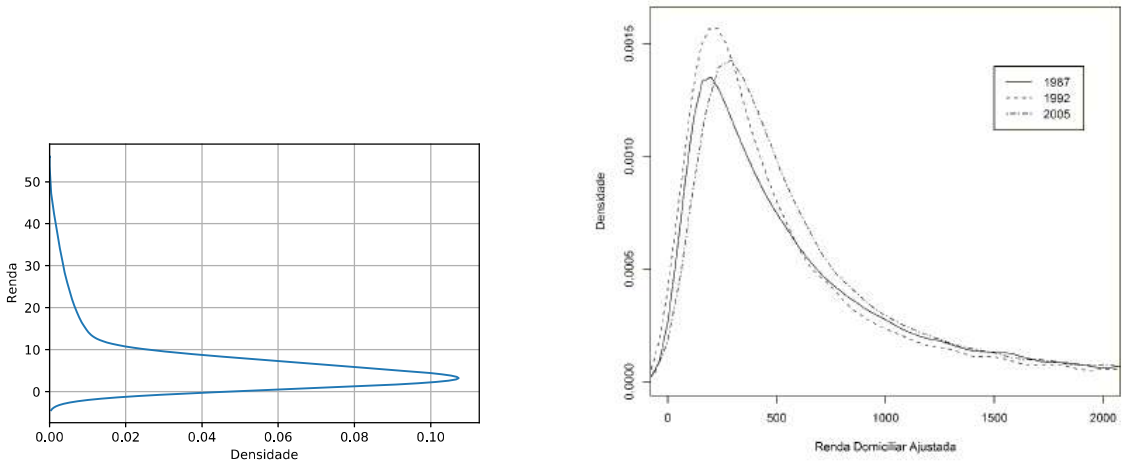

Figura 2.7 – À esqueda,a representação da curva de renda de Pareto para a distribuição de renda na sociedade. À direita, a distribuição de renda no Brasil. Fonte: **Figueiredo e Ziegelmann[1]**.

por hora ou $R\$9700.00$ por ano, qual a porcentagem de pessoas que ganham 10 vezes este valor? De acordo com Pareto seria 3.2%. E 100 vezes este valor? 0.1%.

Para se construir tal gráfico é necessário usar um gráfico log-log para que um padrão apareça nos dados. Pensando da mesma maneira, Mandelbrot reuniu dados de preço do algodão de um período de 100 anos para realizar o mesmo que Pareto, utilizando variações de preço do algodão de um dia para outro, de uma semana para outra, de um mês para outro e de um ano para outro. Surpreendentemente, obteve o gráfico a seguir

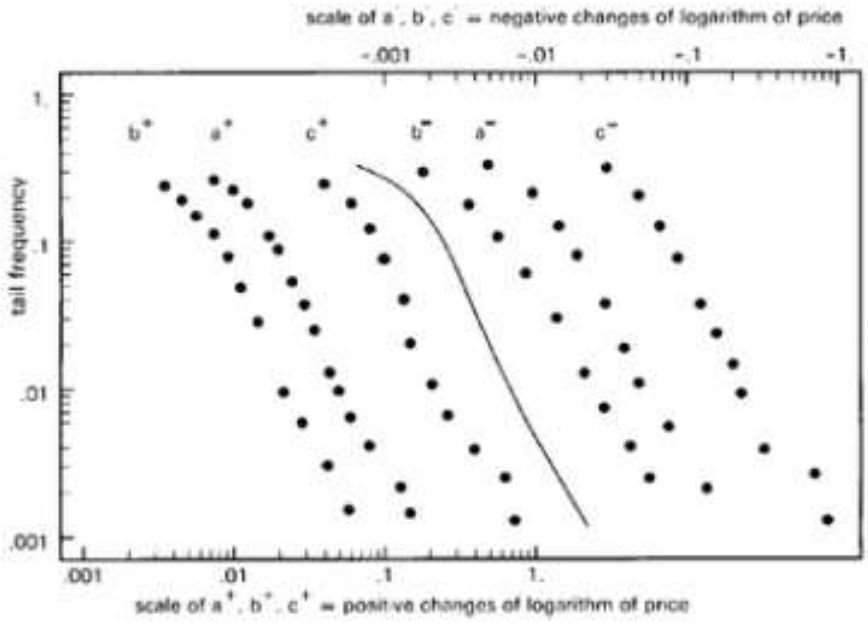

Figura 2.8 – Gráfico de Mandelbrot para a variação do logaritmo do preço do algodão em diferentes épocas. Fonte:**Mandelbrot e Hudson[2]**.

Temos neste gráfico a variação do logarítmo dos preços, onde a^+ e a^- indicam as varia-

ções diárias positivas e negativas de 1900 à 1945 e b^+, b^-, c^+, c^- indicam variações diárias (1944-1958) e mensais (1888-1940), respectivamente. Diferentemente de Pareto, tempo é um dimensão necessária a se considerar neste caso, na distribuição de renda o tempo é irrelevante, uma vez que Pareto utilizou de rendas anuais de diferentes indivíduos de forma não ordenada e em diferentes épocas. Em ações ou commodities o preço depende do tempo, mas o que se vê no gráfico é que em diferentes escalas de tempo o comportamento é semelhante, o que dá a ideia de que há um padrão envolvido. No gráfico **2.9**, temos a mesma tendência de padrão para a ação PETR4 no período de 26 de maio de 1993 até 30 de abril de 2021, com variações diárias, semanais e mensais, que equivalem a 1 dia, 5 dias e 22 dias, respectivamente.

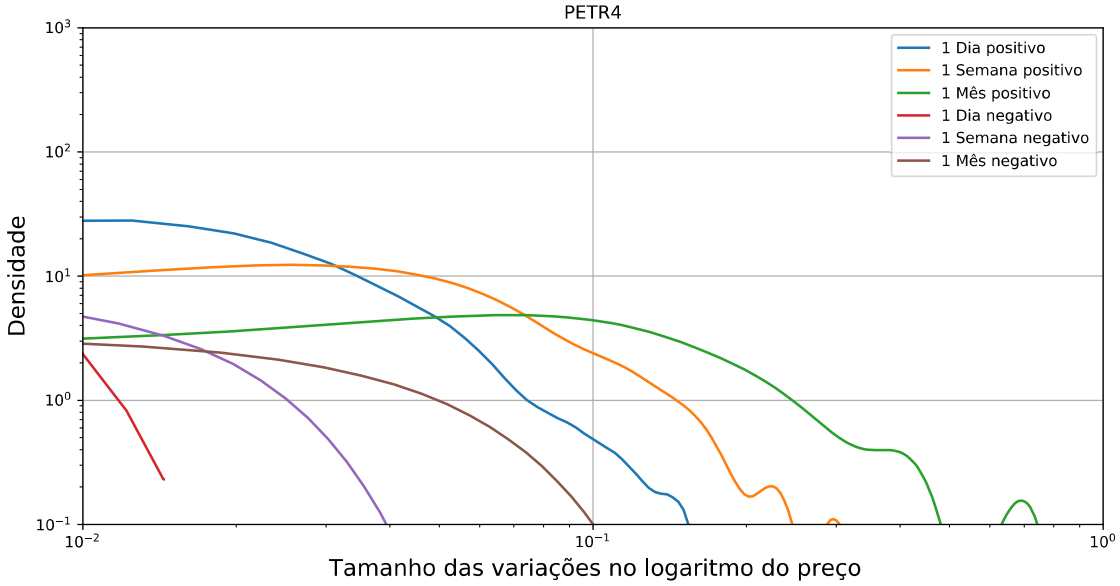

Figura 2.9 – Linhas de variação do logaritmo do preço da PETR4.

O padrão aqui encontrado é o de invariância temporal, ou de escalas de tempo, o que retorna na equação encontrada **2.1** por **Mantegna e Stanley[22]**. Outro padrão, é probabilidade de pequenas mudanças no preço serem maiores do que grande mudanças, assim como em Pareto, legiões de pessoas pobres são muito mais comuns do que pessoas ricas, ou o número de palavras raras no dicionários serem muito maiores do que o número de palavras comuns, e com isso, temos a angulação das distribuições independente das escalas de tempo. No caso das linhas de algodão, temos uma angulação de -1.7. Isso está relacionada com a distribuição de Levy, onde podemos utilizar a distribuição de Levy truncada com a angulação de -1.7 para definir os valores das variações de preço. Já a angulação da distribuição de renda de Pareto seria 1.5 e o jogo da moeda de cara ou coroa, 2. Tendo o padrão encontrado, é preciso discutir sobre a memória no mercado. Em [24], é encontrado forte autocorrelação para tempos pequenos (variações diárias), mas também é encontrado para tempos longos (variações mensais), mesmo que fracamente. A figura **2.10** nos mostra como decai a corre-

lação com o passar dos dias para a ação PETR4, onde é observável que a autocorrelação se estende a quase 600 dias, ou quase 2 anos e meio.

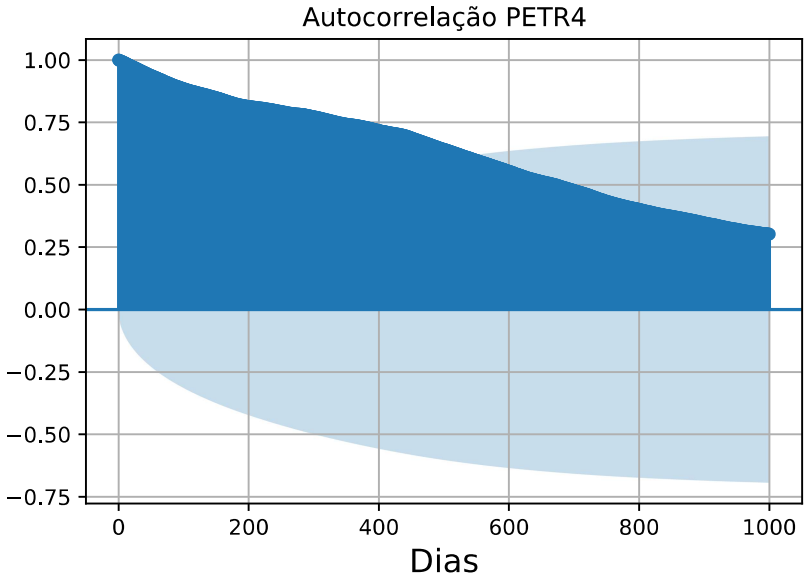

Figura 2.10 – Autocorrelação do preço da PETR4.

Portanto, é possível utilizar ferramentas que preveem o preço de uma ação, índice ou commodities, devido a presença de padrões e memórias. Neste trabalho será utilizado como ferramenta de previsão redes neurais recorrentes, devido a seu grande poder preditivo apresentado em outros trabalhos e seu intenso uso nos dias atuais, em ações da bolsa de valores Ibovespa B3.

3 INTRODUÇÃO ÀS REDES NEURAIS

Redes neurais artificias são algoritmos que imitam o comportamento biológico de redes de neurônios. A primeira relação que surgiu entre eletrônica e redes de neurônio foi na década de 40 com neurofisiologista Warren McCulloch e o matemático Walter Pitts em seu trabalho **[25]**. Já o primeiro modelo matemático bem conhecido que fosse capaz de aprender foi o *Perceptron* criado por Frank Rosenblatt em 1958 em seu trabalho **[3]**. O perceptron é um modelo simples que consegue classificar informações de forma linear como mostra figura a seguir

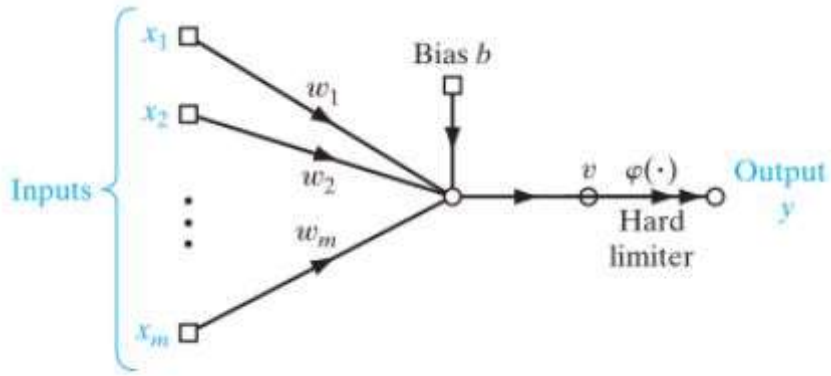

Figura 3.1 – Perceptron. Fonte: **[3]**

onde "Inputs"são as entradas de dados, w_i são os pesos de cada entrada, "Bias"é o viés somado as entradas multiplicadas pelos pesos o que gera o vetor v, que por fim, passa por uma função de ativação Φ, gerando a saída/output do perceptron. O perceptron é treinado de acordo com o erro gerado pela saída y, ou mais especificamente, atualiza os pesos w_i de acordo com o erro. Em **Haykin[26]** é descrito detalhadamente a matemática e a física por trás desta e de outras redes neurais. Contudo, em 1959 havia um problema, o poder computacional. Minsky e Seymour em **[27]**, demonstraram que o perceptron munido do poder computacional existente era difícil de ser implementado em um algoritmo, uma vez que Rosenblatt utilizou de técnicas matemáticas empíricas e difíceis, o que levou a estagnação da área.

Este ramo teve seu "boom"somente em 1986, principalmente devido a evolução computacional, mas também do desenvolvimento de técnicas de treinamento dos neurônios, como o Backpropagation. Desta época em diante, houve novamente uma grande evolução do poder computacional e de novas arquiteturas de redes neurais para diferentes tipos de problemas. Mas o que causou realmente uma grande diferença foi a capacidade de armazenar grandes quantidades de informação, ou seja, dados.

Atualmente, a arquitetura de rede neural Feed Foward Neural Network (FFNN) é uma das mais utilizadas, ela consiste em utilizar camadas de neurônios de entrada (input layer), escondido (hidden layer) e de saída (output layer), na sua forma mais simples. Geralmente se usa o método backpropagation para a correção dos pesos, sendo eficiente quando se tem poucas camadas escondidas como mostra figura **3.2**.

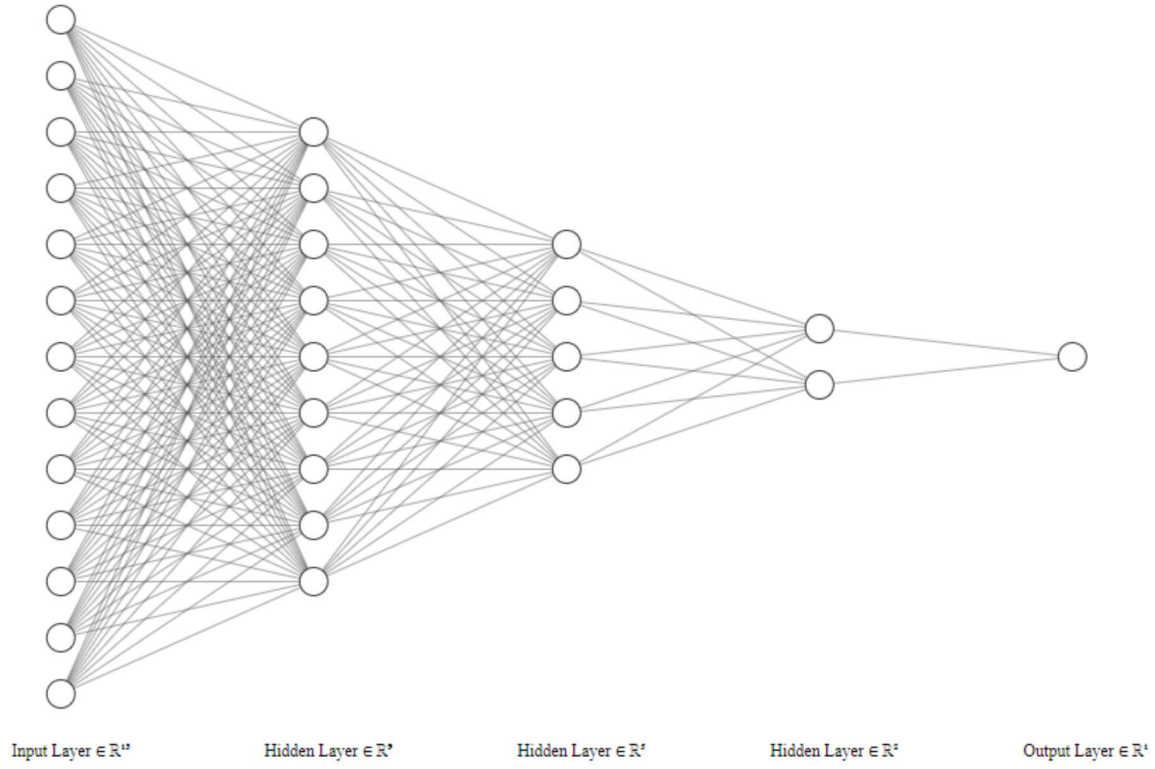

Figura 3.2 – Feed Foward Neural Network (FFNN)

Segundo **Hornik Maxwell Stinchcombe[28]**, FFNN conseguem aproximar uma função diferenciável de dimensão espacial finita e única de outra dimensão com um grau de acurácia desejável, desde que sejam fornecidos suficiente camadas de neurônios escondidas. Ou seja, FFNN são uma classe de aproximadores universais. A pergunta que pode surgir é como uma rede neural pode aprender e reproduzir um sistema que possui uma dinâmica caótica e estocástica como o mercado financeiro. A resposta é que o mercado possui momentos em que a aleatoriedade governa a dinâmica, mas em outros momentos existe uma função diferenciável que se destaca, possibilitando assim, o aprendizado das redes neurais. O processamento de dados se dá ao inserir um conjunto de dados na camada de entrada, que são multiplicados por pesos estatísticos e aleatórios. Após isso, os valores são somados nos diferentes neurônios da camada de entrada e passam por uma função de ativação (função logística, tangente hiperbólico, etc. Vai depender se o problema é de regressão ou classificação). A saída dos neurônios da camada de entrada são conectados aos neurônios da camada escondida, tendo também pesos regulando a conexão. Da mesma maneira, as saídas da camada de entrada são

somados nos neurônios da camada escondida e passam pela função de ativação. Por fim, o mesmo processo ocorre na camada de saída. A depender do valor no(s) neurônio(s) de saída, deve ser feito a correção dos pesos nas diferentes camadas, o problema se encontra quando a arquitetura possui muitas camadas escondidas (Deep Neural Network). Se utilizado back-propagation, apenas as camadas próximas a camada de saída serão corrigidas de maneira eficiente, pois o processo perde a informação sobre a correção na medida que avança para camadas iniciais. Pode ser aplicada em diferentes problemas de classificação e regressão.

Existem outras arquiteturas como a Rede Neural Convolucional (CNN) (**Albawi, Mohammed e Al-Zawi[29]**), uma rede neural invariante a transformações espaciais, muito utilizada para reconhecimento ou classificação de imagens e vídeos (**Chauhan, Ghanshala e Joshi[30]**), bem como processamento de linguagens (NLP) (**Li, Li e Wang[31]**). Mas existe uma outra classe de arquiteturas de redes neurais chamada Recurrent Neural Networks (RNN), que consiste em recorrência de dados dentro da própria estrutura, muito utilizado quando se trata de dados que possuem dependência temporal. Este assunto será tratado na secção seguinte.

3.1 REDES NEURAIS RECORRENTES

Redes neurais recorrentes tem a característica de poderem guardar memória ao se auto alimentar com os dados de saída de seus neurônios. Podemos citar algumas arquiteturas que utilizam desta técnica, sendo muito utilizado para séries temporais. Isso é devido ao fato de que séries temporais são dados que se auto correlacionam, podendo ser bem comportados ou caóticos, como é caso do sistema do Lorenz ou o preço de fechamento de ações (o objetivo deste trabalho).

A Long-Short-Term-Memory (LSTM) (**Graves[32]**, **Hochreiter e Schmidhuber[33]**), é uma *Recurrent Neural Network* (RNN) que é capaz de aprender dependências de longo e curto alcance, mantendo um erro constante de correção evitando o problema de dissipação do gradiente (**Informatik et al.[34]**) , o que permite que continue aprendendo na medida que são feitas as iterações. A LSTM é constituída de células nas quais informações podem ser armazenadas ou lidas. Elas tomam decisões do que guardar ou descartar, de acordo com portões (gates), que serão explicados na secção **3.2.1**.

A Gated Recurrent Unit (GRU) (**Loye[35]**), é também uma RNN que utiliza portões (gates) como mecanismo de controle de fluxo de dados em suas células. Sendo uma versão simplificada da LSTM, a GRU capta dependências de séries temporais sem eliminar informações do início da série temporal, ou seja, considera a influência de dados iniciais nos dados finais. Além disso, assim como a LSTM, resolve o problema de correção do back-propagation para RNN (problema de desaparecimento/explosão do gradiente - **Informatik**

et al.[34]). Será discretizado sobre na secção **3.2.2**.

A Echo State Network (ESN) (**Lukosevicius[4]**), é uma rede neural recorrente que se baseia no teorema de Takens (**Hart, Hook e Dawes[36]**) para a construção de sua arquitetura. A rede possui uma camada de entrada seguida de um reservatório, que simula uma rede neural profunda (Deep Neural Network), porém com uma necessidade menor de poder computacional para executá-la. Logo após o reservatório há a camada de saída, está será a única camada a ser corrigida, o que torna a execução rápida e eficiente, uma vez que as camada de entrada e o reservatório são apenas gerados e não corrigidos. Um detalhe importante no reservatório, é o fato de que os neurônios que o constituem são conectados entres si de forma aleatória, podendo ser controlado o quão são conectados entre si. Vale ressaltar que, **Grigoryeva e Ortega[37]** estende o teorema de aproximador universal, dado a FFNN, para a ESN.

A Deep Echo State Network (Deep ESN) funciona com os mesmos princípios da ESN (**Gallicchio, Micheli e Pedrelli[5]**, **Kim e King[21]**), possuindo uma camada de entrada e saída, no entanto com múltiplos reservatórios entres estes e conectados entre si. Computacionalmente, dependendo do número de camadas ela é um pouco mais lenta que a ESN, mas ainda mais rápida do que a LSTM e a GRU. A Deep ESN realiza uma combinação linear dos estados de cada reservatório e dos pesos de saída, permitindo que os diferentes pesos contribuam para um desenvolvimento múltiplo da dinâmica do estado geral da rede, como apresentado em **3.2.4**.

3.2 CONSTRUÇÃO DAS REDES NEURAIS

3.2.1 Long Short Term Memory - LSTM

Sua arquitetura é composta por células e é dada pela figura **3.3**. Em resumo, cada célula possui três portões dos quais regulam o fluxo de dados, onde é escolhido os dados que serão adicionados, descartados e que serão a saída.

A célula permite três entradas, os dados (x_t) no tempo t , o estado da última célula (S_{t-1}) no tempo ($t-1$) e a saída da última célula (y_{t-1}) no tempo ($t-1$). O Termo de longo alcance é representado por s enquanto que o termo de curto alcance é representado por y. Os portões são definidos da seguinte forma

- $f_{(gt)}$ é o portão de descarte (forget gate)
- $i_{(gt)}$ é o portão de entrada (input gate)
- $o_{(gt)}$ é o portão de saída (output gate)

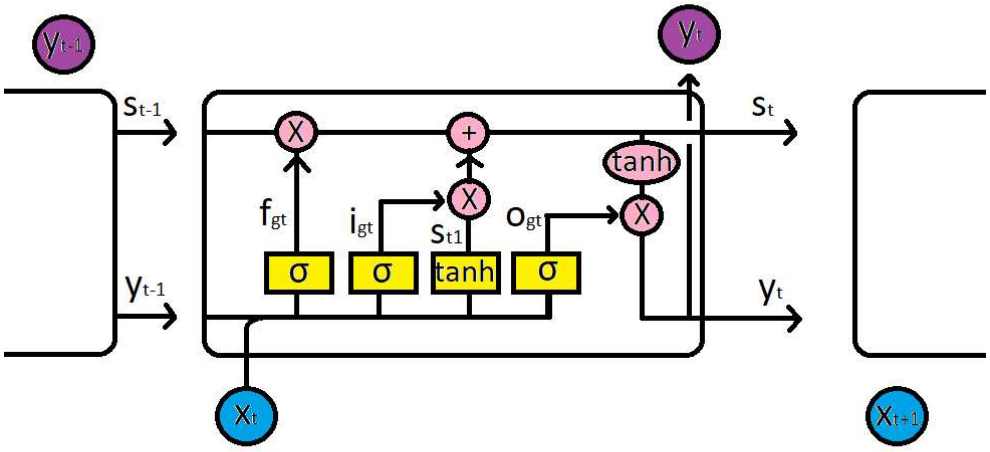

Figura 3.3 – Arquitetura LSTM

Primeiro passo: A LSTM irá decidir qual informação será descartada, sendo feito pelo forget gate que utiliza uma função de ativação sigmoide. O valor de 0 representa descarte enquanto que o valor 1 significa guardar esta informação. As entradas y_{t-1} e x_t serão concatenadas, multiplicadas pelo peso $W_{(fg)}$ e somadas com o o viés (bias) $b_{(fg)}$. Após isso, passarão pela função de ativação sigmoide. Matematicamente temos que

$$f_{(gt)} = Sigmoid(W_{fg}[y_{t-1}, x_t] + b_{fg}) \tag{3.1}$$

Segundo passo: Agora a LSTM irá decidir qual informação será adicionada à célula através do input gate. É utilizado a função sigmoide como função de ativação e também a função tangente hiperbólica. Podemos separar este passo em dois.

- O input gate utiliza a função sigmoide, sendo 0 para não ser adicionado e 1 para ser adicionado. As entradas y_{t-1} e x_t serão concatenadas, multiplicadas pelo peso $W_{(ig)}$ e somadas com o o viés (bias) $b_{(ig)}$, passando pela função de ativação. Matematicamente temos

$$i_{(gt)} = Sigmoid(W_{ig}[y_{t-1}, x_t] + b_{ig}) \tag{3.2}$$

- A tanh serve para regular o fluxo da célula, pois seu range é de $[-1, 1]$, o que evita overfitting e underfitting. Possui seus próprios para serem multiplicados pelas entradas

e somados com um bias. Matematicamente se tem

$$S_{(t1)} = tanh(W_c[y_{t-1}, x_t] + b_c) \tag{3.3}$$

Terceiro Passo: Agora será criado o novo estado da célula S_t a partir do primeiro e segundo passos. O estado da última célula $S_{(t-1)}$ será multiplicado pelo que saiu do output gate e somado com a multiplicação do que saiu do input gate com $S_{(t_1)}$. Visualmente

$$S_{(t)} = S_{(t2)} = (f_{gt} * S_{(t-1)}) + (i_{gt} * S_{(t1)}) \tag{3.4}$$

Quarto Passo: Será decido a saída da célula em dois passos.

- O output gate decide que parte das entradas y_{t-1} e x_t irá influenciar na saída. Possuindo seus próprios pesos, tem-se que

$$O_{gt} = Sigmoid(W_{og}[y_{t-1}, x_t] + b_{og}) \tag{3.5}$$

- O resultado é multiplicado por $tanh(S_t)$, dando em

$$y_t = O_{gt} * tanh(S_t) \tag{3.6}$$

3.2.2 Gated Recurrent Unit - GRU

Como dito anteriormente, a GRU é uma versão simplificadas da LSTM, possuindo apenas 2 portões, **Update gate** (portão de atualização) e o **Reset gate** (portão de resetar). Esses portões, assim como na LSTM, são treinadas para regular o fluxo de dados pela célula, retirando os dados que são irrelevantes. A célula de GRU possui apenas 2 entradas, a entrada de dados x_t e a última saída y_{t-1}, ou estado escondido (hidden state). Se os vetores gerados pelos portões forem zero, significa que a informação é inútil, enquanto que se forem 1 significa que a informação é muito relevante. A estrutura da GRU é dada a seguir

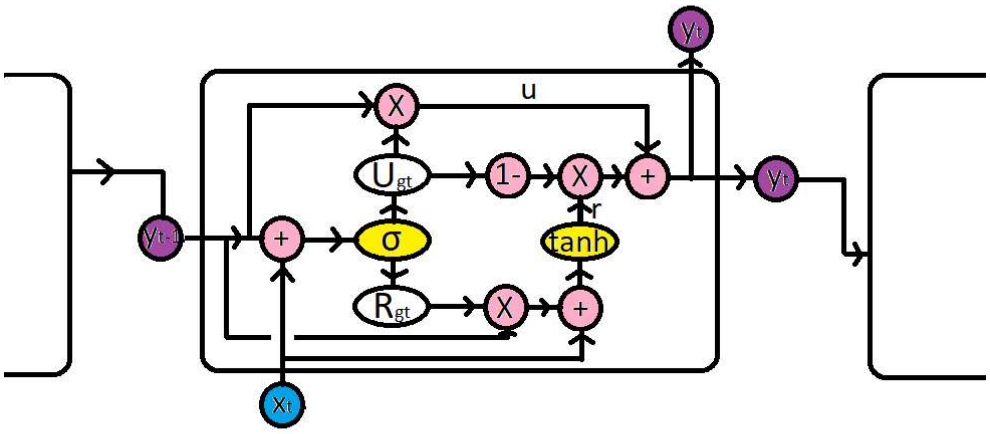

Figura 3.4 – Arquitetura da célula GRU

Primeiro passo: O primeiro passo é criar o portão de resetar, para isso a última saída y_{t-1} e a entrada no tempo atual x_t são multiplicados por pesos e somados, passando pela função de ativação sigmoide. Temos então que

$$R_{gt} = sigmoid(W_{R_{input}}.x_t + W_{R_{hidden}}.y_{t-1}) \tag{3.7}$$

Ao treinar toda a rede neural por back-propagation, os pesos serão atualizados e os vetores dos portões irão guardar somente a informação que seja útil. Continuando, a última saída será multiplicada por um outro peso (W_{y_1}), e por conseguinte pelo resultado da última expressão (R_{gt}), mas este será um produto Hadamard (o produto de matrizes de dimensão mxn irá gerar uma matriz mxn). Então, será somado à multiplicação dos dados de entrada por um peso (W_{x_1}), e passarão pela função de ativação tangente hiperbólica

$$r = tanh(R_{gt} \odot (W_{y_1}.y_{t-1}) + W_{x_1}.x_t) \tag{3.8}$$

Segundo passo: Será criado agora o portão de atualização. Matematicamente, as expressões para ambos os portões são iguais, o que os diferencia são os pesos, dessa forma

$$U_{gt} = sigmoid(W_{U_{input}}.x_t + W_{U_{hidden}}.y_{t-1}) \tag{3.9}$$

O resultado desta expressão será submetido ao produto Hadamard com a última saída

para obter u

$$u = U_{gt} \odot y_{t-1} \tag{3.10}$$

O papel do portão de atualização é determinar o quanto de informação presente na última saída será necessário para o futuro.

Terceito Passo: Será utilizado agora r e u para obter a saída ou output. Será feita a versão inversa do vetor de atualização $(1 - U_{gt})$ e produto Hadamard com r. Por fim, será somado à u, como mostra a seguir

$$y_t = r \cdot (1 - U_{gt}) + u. \tag{3.11}$$

Este será a saída no tempo t e o estado escondido (última saída) na próxima interação.

3.2.3 Echo State Network - ESN

Foi utilizado, principalmente para a criação da ESN, a biblioteca "Numpy"(**Oliphant Alex Griffing[38]**) com suas funções matemáticas que permitem cálculos com matrizes de maneira fácil e intuitiva. O algoritmo foi baseado em **Lukosevicius[4]**, onde é descrito matematicamente como funciona uma ESN. Basicamente, esta rede neural possui entradas, que são multiplicados por pesos esteatíticos aleatórios, mas que não se alteram a cada interação. Além disso, há um reservatório com pesos aleatórios, que também não se modificam, mas que cumpre o papel das camadas escondidas de redes neurais profundas, e por fim uma saída com pesos que se atualizam/corrigem a cada interação, figura **3.5**.

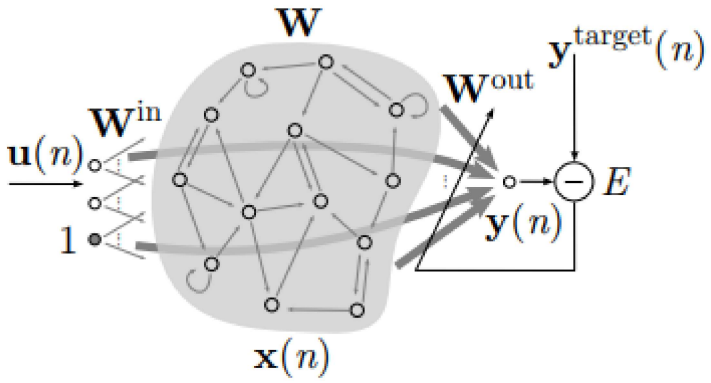

Figura 3.5 – Arquitetura da ESN. Fonte: **Lukosevicius[4]**

Essa rede neural é a aplicação na prática do teorema de Takens **Hart, Hook e Dawes[36]**. O teorema de Takens diz que dada uma dinâmica temporal, ou um atrator, que dependa de n grandezas, é possível que com apenas uma dessas grandezas seja reproduzido um novo

atrator, não com a mesma forma geométrica, mas com as mesmas características como o expoente de Lyapunov e a topologia. Portanto, é factível a reprodução de outras grandezas do sistema que sejam desconhecidas. O reservatório da ESN procura reproduzir essa dinâmica da série temporal para a previsão da mesma. A figura **3.6** demonstra essa reprodução utilizando o sistema proposto por Lorenz para modelar a dinâmica do clima.

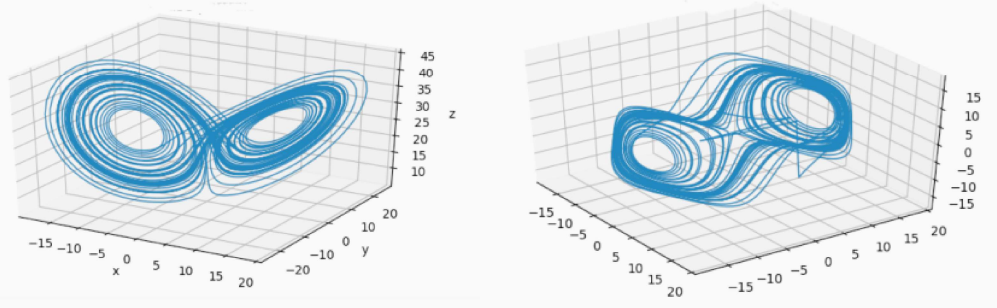

Figura 3.6 – Teorema de incorporação de Takens (Takens' Embedding Theorem). Sistema de Lorenz a esquerda e sua reprodução pelo reservatório à direita. Fonte: **https://github.com/carlosgmartin/takens/blob/master/README.md**

Vamos entender melhor como se dá o processo de entrada e saída dos dados. Matematicamente, temos que

$$\tilde{\mathbf{x}}(n) = tanh(\mathbf{W}^{in}[1; \mathbf{u}(n)] + \mathbf{W}\mathbf{x}(n-1)), \tag{3.12}$$

$$\mathbf{x}(n) = (1-\alpha)\mathbf{x}(n-1) + \alpha\tilde{\mathbf{x}}(n), \tag{3.13}$$

onde $\mathbf{x}(n) \in \mathbb{R}^{N_x}$ é um vetor de ativação do reservatório de neurônios e $\tilde{\mathbf{x}}(n) \in \mathbb{R}^{N_x}$ é sua atualização na iteração n. $\mathbf{W}^{in} \in \mathbb{R}^{N_x\, x\, (1+N_u)}$ e $\mathbf{W} \in \mathbb{R}^{N_x\, x\, N_x}$ são os inputs e as matrizes de pesos respectivamente, sendo $\alpha \in (0, 1]$ a taxa de vazamento (leaking rate). A camada de saída é definida como

$$\mathbf{y}(n) = \mathbf{W}^{out}[1; \mathbf{u}(n); \mathbf{x}(n)], \tag{3.14}$$

onde $\mathbf{y}(n) \in \mathbb{R}^{N_y}$ é a saída da rede neural, $\mathbf{W}^{out} \in \mathbb{R}^{N_y\, x\, (1+N_u+N_x)}$ é a matriz de peso da saída. Pode ser feito ainda conexões "feedback"ou "teacher forcing"que utiliza de recorrência ao introduzir $\mathbf{W}^{fd}\mathbf{y}^{target}(n-1)$ em $\tilde{\mathbf{x}}(n)$. Os passos a serem seguidos são

- Gerar um grande reservatório aleatório ($\mathbf{W}^{in}, \mathbf{W}, \alpha$);
- Rodar usando o input de treinamento $\mathbf{u}(n)$ e coletar os estados do reservatório $\mathbf{x}(n)$;

- Computar os pesos de saída $\mathbf{W}^{out}$ do reservatório de forma a minimizar a métrica desejada;
- Usar a rede neural treinada com novos inputs $\mathbf{u}(n)$ computando $\mathbf{y}(n)$ empregando os pesos de saída treinados $\mathbf{W}^{out}$.

Para treinar os pesos de saída, vamos reescrever a equação **3.14**

$$\mathbf{Y} = \mathbf{W}^{out}\mathbf{X}, \tag{3.15}$$

onde $\mathbf{Y} \in \mathbb{R}^{N_y\, x\, T}$ são todos os $\mathbf{y}(n)$ e $\mathbf{X} \in \mathbb{R}^{(1+N_u+N_x)\, x\, T}$ são todos os $[1; \mathbf{u}(n); \mathbf{x}(n)$ produzido pelo reservatório com $\mathbf{u}(n)$, ambos coletados durante o período de treinamento $n = 1, ..., T$. Mas queremos uma relação de $\mathbf{W}^{out}$ em função de $\mathbf{Y}^{target}$ para poder corrigí-lo de maneira apropriada aos dados. Com isso

$$\mathbf{Y}^{target} = \mathbf{W}^{out}\mathbf{X}, \tag{3.16}$$

onde $\mathbf{Y}^{target} \in \mathbb{R}^{N_y\, x\, T}$ são todos as saídas $\mathbf{y}(n)$ e $\mathbf{X}$ pode ser chamada de "matriz design". Uma das soluções que resolvem a equação acima é o "ridge regression"dado por

$$\mathbf{W}^{out} = \mathbf{Y}^{target}\mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \beta\mathbf{I})^{-}1, \tag{3.17}$$

onde β é o coeficiente de regularização e $\mathbf{I}$ é a matriz identidade. Outra solução possível é "Direct Pseudoinverse"dado por

$$\mathbf{W}^{out} = \mathbf{Y}^{target}\mathbf{X}^{\dagger}, \tag{3.18}$$

onde $\mathbf{X}^{\dagger}$ é a matriz pseudoinversa Moore-Penrose de $\mathbf{X}$.

3.2.4 Deep Echo State Network - DESN

Da mesma maneira que foi criado algoritmo para a ESN, foi criado o algoritmo da Deep ESN, utilizando a biblioteca "Numpy". Neste caso, o algoritmo foi baseado em **Gallicchio, Micheli e Pedrelli[5]**, onde é descrito matematicamente o seu funcionamento bem como sua aplicação na diagnosticação da doença de parkinson. Assim como a ESN, a Deep ESN possui entradas com seus respectivos pesos, no entanto, possui reservatório interligados por pesos de conexão. Por fim, sua saída é semelhante à saída da ESN, onde apenas estes pesos são atualizados a cada iteração, como é apresentado na figura 3.7. Matematicamente, temos que a entrada é denotada por $\mathbf{u}(t) \in \Re^{N_U}$ e o estado dos reservatórios são dados por $\mathbf{x}^{(l)}(t) \in \Re^{N_R}$ onde N_L é quantidade de reservatórios ou camadas. O estado do primeiro reservatório é dado

por

$$\mathbf{x}^{(1)}(t) = (1 - a^{(1)})\mathbf{x}^{(1)}(t-1) + a^{(1)}\mathbf{tanh}(W_{in}\mathbf{u}(t) + \hat{\mathbf{W}}^{(1)}\mathbf{x}^{(1)}(t-1)). \tag{3.19}$$

enquanto que para os reservatórios seguintes ($l > 1$), tem-se a seguinte expressão

$$\mathbf{x}^{(l)}(t) = (1 - a^{(l)})\mathbf{x}^{(l)}(t-1) + a^{(l)}\mathbf{tanh}(W^{(l)}\mathbf{x}^{(l)}(t) + \hat{\mathbf{W}}^{(l)}\mathbf{x}^{(l)}(t-1)) \tag{3.20}$$

A matriz de pesos de entrada é representada por $\mathbf{W}_{in} \in \Re^{N_R\, x\, N_U}$, $\hat{\mathbf{W}}^{(l)} \in \Re^{N_R\, x\, N_R}$ é a matriz de pesos do reservatório l, $\mathbf{W}^{(l)} \in \Re^{N_R\, x\, N_R}$ é a matriz de pesos de conexão entre os reservatórios, ou entre as camadas $l-1$ e l. O símbolo $a^{(l)}$ é a taxa de vazamento (leaking rate) da camada l e **tanh** é a função de ativação utilizada. O estado geral da Deep ESN é dado pela concatenação do estado de seus reservatórios, como é mostrado na equação **3.21**.

$$\mathbf{x}(t) = (\mathbf{x}^{(1)}(t), ..., \mathbf{x}^{(N_L)}(t)) \in \Re^{NLN_R}. \tag{3.21}$$

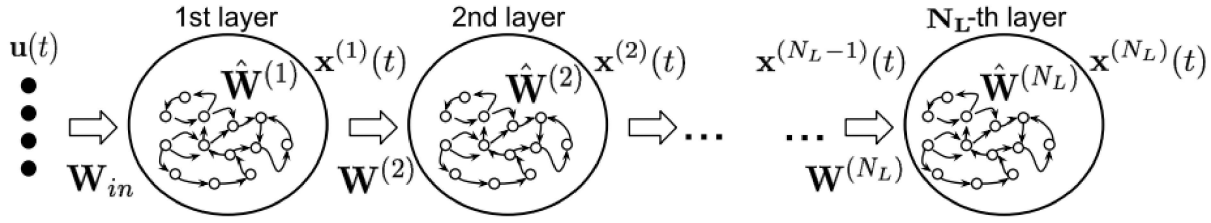

Figura 3.7 – Arquitetura da Deep ESN. Fonte: **Gallicchio, Micheli e Pedrelli[5]**

Deve ser realizado ainda, o escalonamento dos pesos de entrada e de conexão entres os reservatórios dados por σ e $\hat{\sigma}$, respectivamente. A matriz de pesos $\mathbf{W}_{in}$ e $\{\mathbf{W}_{(l)}\}_{l=2}^{N_L}$ são iniciadas randomicamente com distribuição uniforme e escalonadas com a norma $p = 2$, tal que $\|\mathbf{W}_{in}\|_2 = \sigma$ e $\|\mathbf{W}_{(l)}\|_2 = \hat{\sigma}$. Essa norma p é definida sendo como $\|\mathbf{A}\|_2 = \sqrt{\lambda_{max}}$, onde $\mathbf{A}$ é uma matriz e λ_{max} é o maior autovalor do produto $(\mathbf{A}^{\dagger}\mathbf{A})$. $\mathbf{A}^{\dagger}$ é definido como o conjugado transposto da matriz $\mathbf{A}$. Já a matriz de pesos $\{\hat{\mathbf{W}}_{(l)}\}_{l=1}^{N_L}$ são também inciados randomicamente com distribuição uniforme, porém, sua escalonação é dada por

$$\frac{max}{1 \leq l \leq N_L} \rho\left((1 - a^{(l)})\mathbf{I} + a^{(l)}\hat{\mathbf{W}}^{(l)}\right) < 1 \tag{3.22}$$

onde ρ é o raio espectral da matriz, ou maior auto valor absoluto. A matriz design, ou estado médio, é computada por

$$\chi(\mathbf{S}) = \frac{1}{n}\sum_{t=1}^{n}\mathbf{x}(t), \quad \mathbf{S} = [\mathbf{u}(1), \cdots, \mathbf{u}(n)] \tag{3.23}$$

onde n é o tamanho da entrada. Com isso, a saída da rede pode ser expressa como

$$\mathbf{y}(\mathbf{S}) = \mathbf{W}^{out}\chi(\mathbf{S}) \tag{3.24}$$

sendo $\mathbf{W}^{out}$ a matriz de pesos de saída. Assim como na ESN, a saída é treinada pelo método "ridge regression"ou "Direct Pseudoinverse", descritos em **3.17** e **3.18** respectivamente.

4 MÉTODOLOGIA

4.1 MÉTRICAS

A aprendizagem de rede neural que consideramos aqui consiste em minimizar um função de erro, que chamaremos de métrica, que essencialmente mede quão bom é a saída da rede quando comparadas com dados reais. Uma das mais comuns é o Erro Quadrático Médio (MSE) que é calculado pela seguinte expressão

$$MSE = \frac{\sum_i^N (\hat{y}_i - y_i)^2}{N} \tag{4.1}$$

onde $\hat{y}_i$ é o valor previsto e y_i é o valor real. Além disso, quanto mais próximo de zero ($MSE \approx 0$) melhor é a previsão da rede neural. Entretanto, há outras como o Coeficiente de Determinação R_2 onde é medido a variância da variável dependente que é prevista, expresso logo abaixo

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)}{\sum_i (y_i - \bar{y}_i)} = 1 - \frac{SS_{res}}{SS_{tot}} \tag{4.2}$$

onde temos que SS_{res} é a Soma Residual dos Quadrados, SS_{tot} é a Soma Total dos Quadrados e o valor de R_2 vai de 0 à 1, sendo 1 o valor máximo e de menor variância da previsão. No entanto, pode haver valores negativos que indicam que o modelo utilizado não é bom.

Contudo, o que é necessário prever é se o o preço de fechamento de um determinado ativo irá se elevar ou decair em relação ao dia anterior. Note, que a figura 4.2, apresenta retas seguidas uma após a outra com diferentes coeficientes angulares. Com isso, ao realizar a previsão com dados de teste basta comparar o sinal do coeficiente angular dos valores reais e previstos, assim poderá ser avaliado o quanto a rede neural acertou a elevação ou queda do preço. Matematicamente, podemos expressar o coeficiente angular como sendo

$$\alpha = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} \tag{4.3}$$

onde x e y são os pontos vizinhos no plano cartesiano representados pelo circulo na figura **4.2**. Mais do que isso, podemos fazer a quantidade de acertos percentual, nos dando o percentual de acerto independente da quantidade de previsões

$$\alpha(\%) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} . \frac{100\%}{n} \tag{4.4}$$

onde n é a quantidade de previsões realizadas. Isso foi utilizado em **Tsang et al.[39]**, onde também tinham o intuito de prever e investir em ações.

Além disso, como se trata de ações, é interessante o conhecimento do retorno adquirido em função da utilização do método aqui proposto. Portanto, foi criado a métrica "Profit"(lucro), que de acordo com a estratégia de investimento, irá retornar o lucro médio mensal. A estratégia de investimento aqui proposta tem como base as estratégias de investimento do trabalho **Tsang et al.[39]**, mas com algumas alterações. O investimento inicial será de $R\$10.000,00$, sendo que para comprar uma ação será analisada antes a previsão de subida ou descida ditada pela rede neural. Ao fim de cada dia, a rede neural será treinada e realizará a previsão do fechamento do dia seguinte, onde a compra ou a venda do ativo será realizado com o valor de abertura do mercado. O ideal seria a compra ou venda durante o leilão para que o valor seja o máximo parecido com o valor de fechamento do dia anterior, mas como não há acesso a esses dados, as operações serão realizadas com o valor de abertura. Além disso, será analisado também se a rede neural cometeu erros ao operar, caso tenha cometido será criado situações ao qual ela não poderá operar por um certo período de tempo escolhido, para que possa treinar e recuperar as informações perdidas e posteriormente operar novamente. São sois parâmetros envolvidos, a quantidade de erros seguidos permitido e o tempo de espera para operar devido a esses erros. A figura **4.1** demonstra esquematicamente a estratégia de investimento enquanto que a figura **4.2** apresenta a previsão da ESN e dados reais.

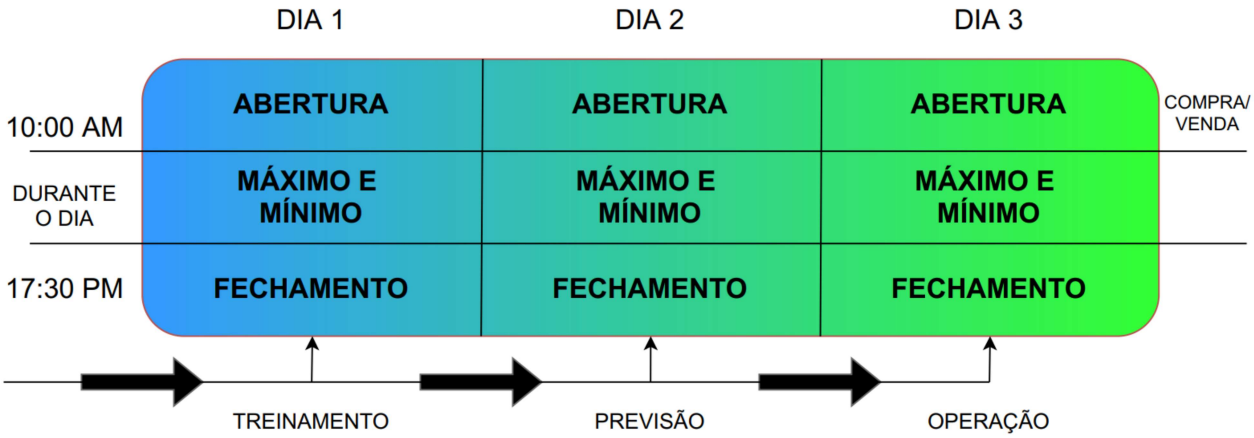

Figura 4.1 – Representação da estratégia de investimento.

A penalidade se dá quando a ESN erra a subida ou descida, como podemos ver do ponto 10 ao ponto 11, onde os dados reais subiram e a previsão foi de queda. Dessa maneira, caso

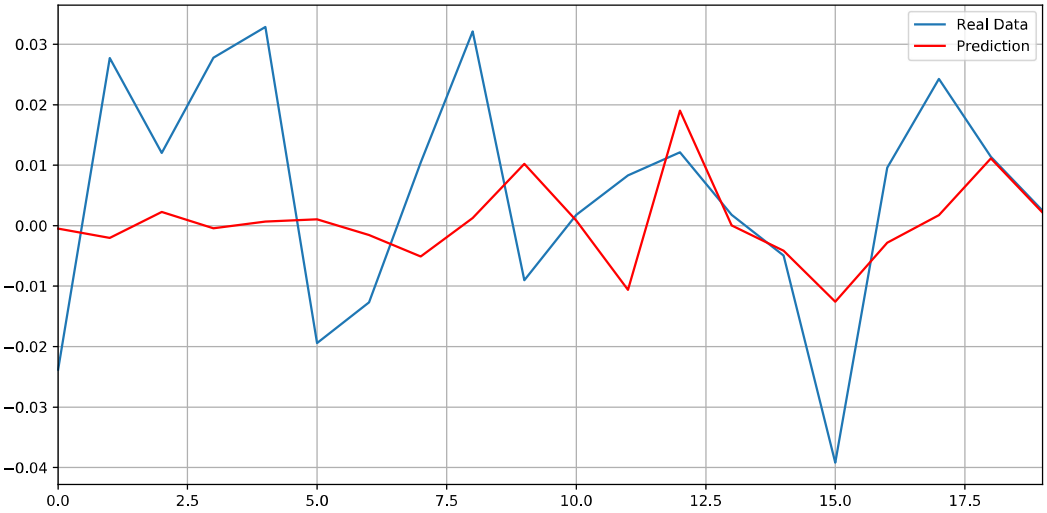

Figura 4.2 – Previsão de 20 dias para PETR4 com variação de 1 dia.

o limite de erros seja um único erro, mantém-se a ESN treinando e prevendo sem que opere, pela quantidade de tempo de espera pré-determinada.

4.2 OTIMIZAÇÃO DE PARÂMETROS

A Echo State Network (ESN), possui alguns parâmetros que podem ser escolhidos para deixar mais rápido a execução, ou limitar os valores dos pesos, ou fazer com que a aprendizagem seja mais lenta ou rápida, entre outros parâmetros, a fim de obter um melhor "fitting"de acordo com a métrica desejada. Mas como não se pode encontrar máximos e mínimos em um função caótica, como a série temporal de uma ativo, é necessário técnicas de otimização de parâmetros. Existem inúmeras técnicas que podem ser utilizadas, e elas se dividem em dois ramos os que usam derivadas (Gradient Based Algorithms) e os que não usam derivadas (Gradient Free Algorithms). Considerando que nossos dados são caóticos, descontínuos e rugosos, é razoável e conveniente utilizar algo que não dependa de derivadas, embora seja mais lento para para executar um algoritmo deste tipo. Dentro deste ramo, pode-se encontrar algumas técnicas de otimização como Busca Exaustiva ou (Exhaustive Search), Algoritmo Genético (Genetic Algorithm), Enxame de Partículas (Particle Swarm), Simulação de aquecimento (Simulated Annealing) e Nelder-Mead.

Busca Exaustiva pode ser considerado o mais preciso, o mais simples e o mais lento. Ele testa todos os resultados possíveis, e quando se tem muitos parâmetros se torna impossível utiliza-lo. Por exemplo, caso se queira executar o algoritmo 10 vezes para cada parâmetro existente, será executado $10^{(\text{números de parâmetros})}$ vezes, ou seja, caso se tenha 5 parâmetros, será

algo da ordem de 10^5 execuções. Uma maneira de contornar tal situação, é utilizando o Random Grid Search, ou Busca Exaustiva Randômica. Ao invés de testar todos os valores possíveis, se escolhe randomicamente os valores dos parâmetros, e assim é possível observar/analisar a tendência de valores dos parâmetros para maximizar ou minimizar a métrica.

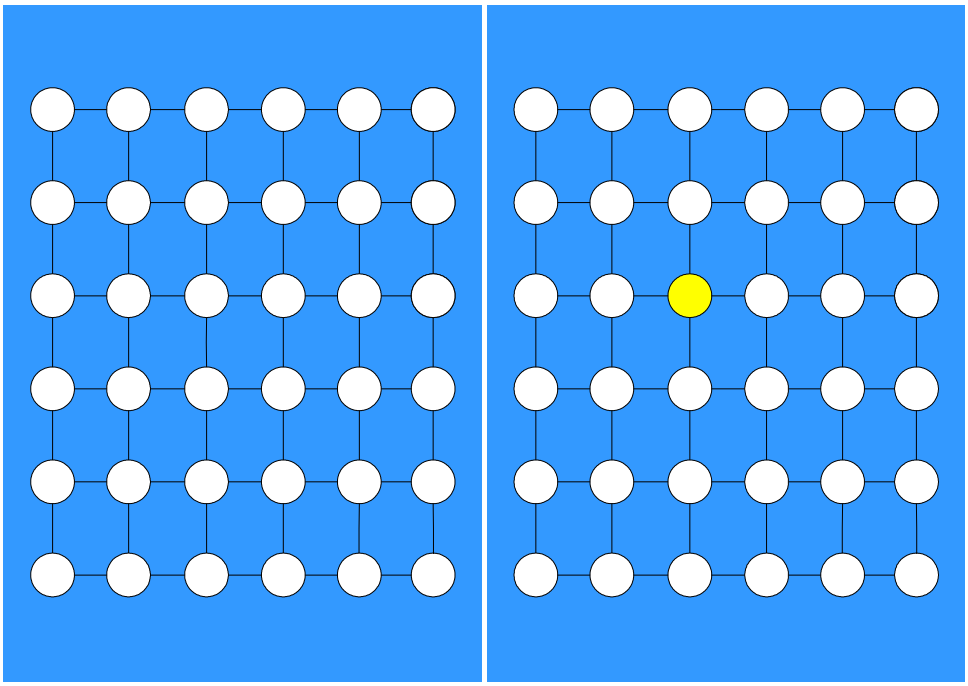

Figura 4.3 – Representação do método Busca Exaustiva

Algorítimo Genético é uma otimização que foi baseada na ideia de evolução natural, primeiramente se propõe algumas amostras/soluções, depois se analisa/classifica qual delas foi a melhor considerando a métrica utilizada. Logo após, se gera novas amostras com variações em torno da ultima escolhida, realizando novamente a análise e assim por diante. Com algumas interações, se chega no máximo ou mínimo desejado. Note que este método não é tão preciso quanto a busca exaustiva, no entanto é mais rápido e dependendo do problema abordado, mais conveniente. Podemos ver uma representação na figura 4.4.

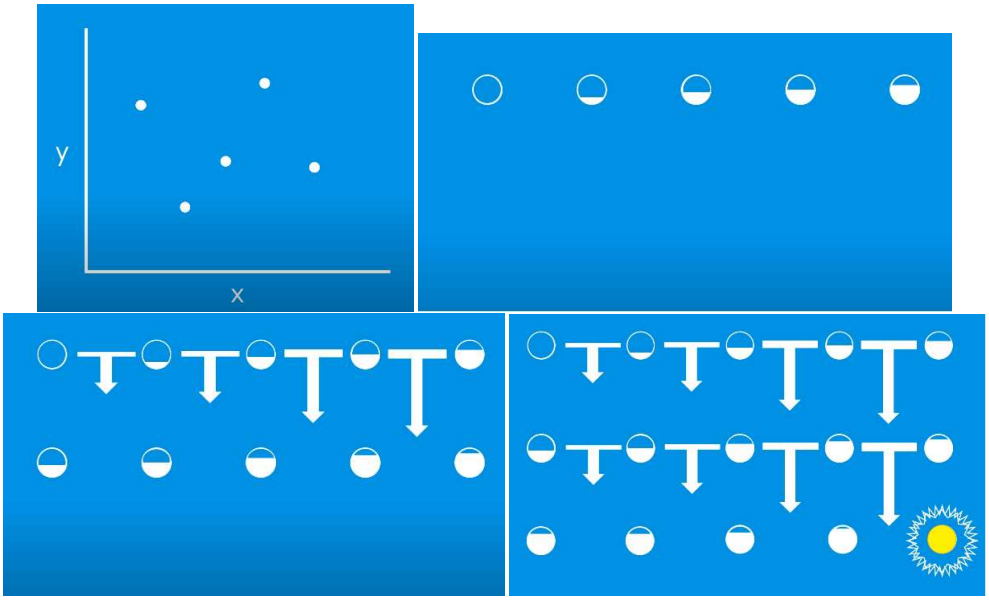

Figura 4.4 – Representação do método Algorítmo Genético

Enxame de Partículas consiste em propor soluções como partículas que possuem direção e velocidade. A cada interação se análisa a melhor solução e o movimento de todas as outras soluções é determinado por uma mistura de direções que está se atualizando em direção da melhor solução. Após "n"interações, todas as partículas convergem para o(s) ponto(s) de melhor solução como podemos ver em 4.5.

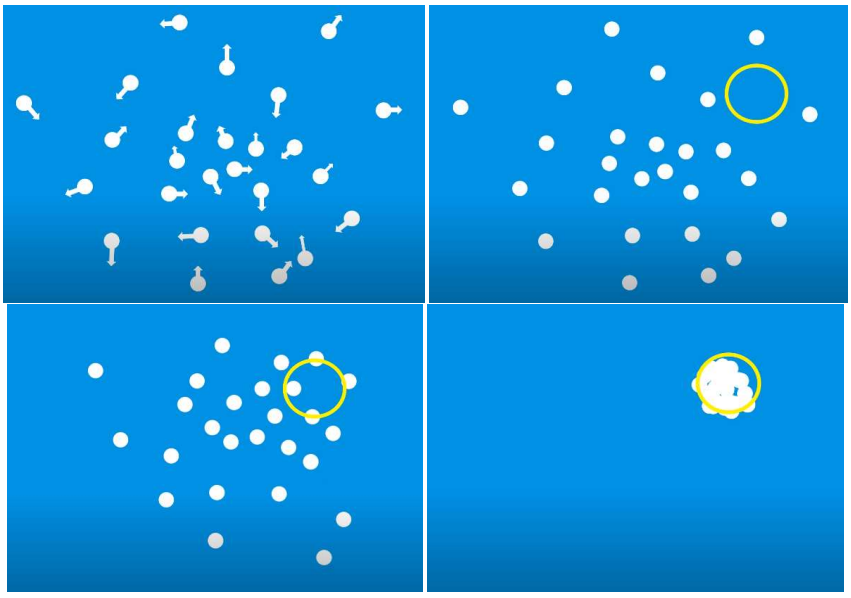

Figura 4.5 – Representação do método Particle Swarm

Simulated Annealing é método que se baseia no aquecimento de metais, ou seja, no fato de que o calor se propaga de partícula para partícula. Para isso, é proposto uma solução. A partir disso, varia-se os parâmetros próximo a solução inicial e é analisado em relação ao anterior, sendo escolhido de acordo com a métrica. É permitido , dentro de um certo limite,

que se tenha soluções piores para que o algorítimo possa explorar toda a região de soluções para que se encontre máximos ou mínimos globais ao invés de locais. A figura 4.6 demonstra gráficamente o método.

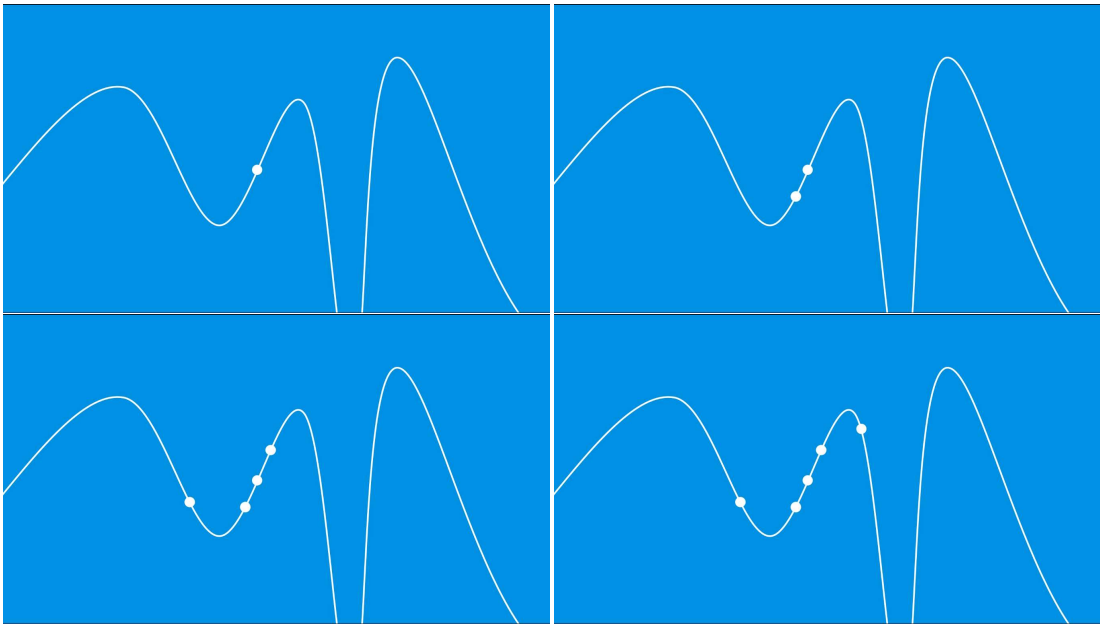

Figura 4.6 – Representação do método Simulated Annealing

Nelder-Mead é um método heurístico que se baseia em movimentos de um simplex (figura geométrica que possui n arestas em um espaço dimensional $(n-1)$, por exemplo um triângulo no espaço 2D). Cada aresta no simplex representa uma solução, onde se organiza em ordem crescente ou decrescente dependendo se o objetivo é maximizar ou minimizar. A cada interação se procura uma aresta melhor do que a pior, assim o simplex pode refletir, expandir, contrair e contrair na direção da melhor solução. Primeiramente se deve propor $(n+1)$ soluções, onde n é o espaço dimensional, avaliar a métrica $f(x)$ nestes pontos

$$f(x_i)\,,\ \ i = 1, 2, 3, ..., (n+1) \tag{4.5}$$

e ordená-los de acordo com o objetivo, crescente para maximizar e decrescente para minimizar segundo JOGLEKAR[40] e Gao e Han[41]

$$\begin{array}{ccc} f(x_h) & , & f(x_s) \quad , ..., f(x_l). \\ (\text{Pior} & , & \text{Segundo pior}, ..., \text{Melhor}) \end{array} \tag{4.6}$$

Após isso, se calcula o centróide ou o centro do simplex, sem considerar x_h

$$C = \frac{1}{n} \sum_{i \neq h} x_i. \tag{4.7}$$

Agora, se tem a transformação do simplex em uma das seguintes formas. A primeira é a *Reflexão*, onde se calcula o ponto refletido

$$x_r = C + \alpha(C - x_h) \tag{4.8}$$

onde α é o parâmetro de reflexão, sendo utilizado geralmente igual a 1. Note, como exemplo de um problema 2D, na figura 4.7 que o ponto x_r serve para mover o simplex para uma região oposta a de x_h. Se $f(x_s) < f(x_r) \leq f(x_l)$, então x_h pode ser substituído por x_r no simplex e ir para a próxima interação.

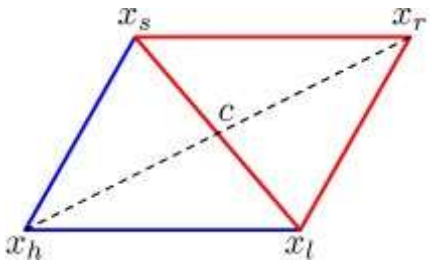

Figura 4.7 – Representação da reflexão em 2-D

A segunda transformação possível é a *Expansão* ocorrendo quando $f(x_r) > f(x_l)$. Dessa forma, tenta-se explorar uma possibilidade além de $f(x_r)$ para talvez encontrar uma solução melhor ainda, calculando

$$x_e = C + \gamma(x_r - C) \tag{4.9}$$

onde γ é o parâmetro de expansão e seu valor geralmente é 2. Com isso, se substitui x_h pelo ponto que é melhor, x_e ou x_r. Isto é chamado de "greddy optmization" (otimização gananciosa), mas para uma melhor exploração de soluções existe "greddy expansion", no qual se substitui x_h por x_e sempre que $f(x_e) > f(x_l)$, independentemente se $x_e > x_r$. Veja um exemplo dessa transformação em 4.8.

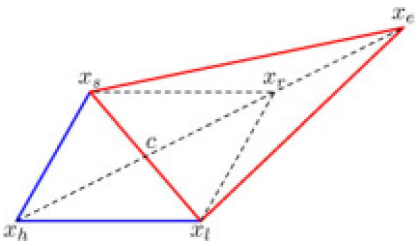

Figura 4.8 – Representação da expansão em 2-D

A terceira transformação é a *Contração* do simplex caso $f(x_r) < f(x_s)$. Isso significa que a direção de x_r não é a ideal, e portanto se faz a avaliação do ponto

$$x_c = C + \beta(x_h - C) \tag{4.10}$$

onde β é o parâmetro de contração e seu valor geralmente é 0.5. Se $f(x_c) > f(x_h)$, se substitui x_h por x_c como mostra em 4.9. No entanto, se este não for o caso existe o ultimo caso, *Srink contraction* .

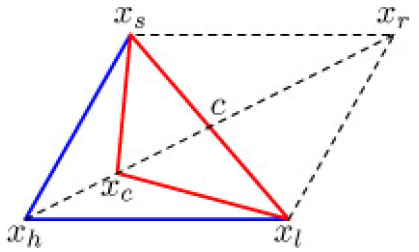

Figura 4.9 – Representação da contração em 2-D

Shrink Contraction também é uma contração do simplex, entretanto em direção do melhor ponto x_l. O que se faz é reescrever todos os pontos considerando x_l e mantendo o mesmo.

$$x_j = x_l + \delta(x_j - x_l) \ , \ x_j = x_{i \neq l} \tag{4.11}$$

onde δ é o parâmetro de encolhimento (shrinkage) e seu valor é usualmente 0.5. Podemos ver tal transformação em 4.10

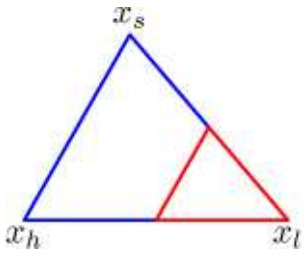

Figura 4.10 – Representação de Srink contraction em 2-D

O algoritmo termina a execução quando se chega ao número de interações desejadas ou alcance um determinado valor de solução.

4.3 VARIAÇÃO DO LOGARITMO DO PREÇO DE FECHAMENTO

Na seção 2, descrevemos sobre as transformações causadas pela aplicação de logaritmo nos dados, como por exemplo, a linearização. Foi comentado também, sobre a variação desses dados, como a variação diária, mensal e anual. Usaremos aqui variações de tempo de 0 à 10 dias, por conta da quantidade de dados disponíveis. O método de variação aqui utilizado é demonstrado na figura 4.11, onde temos variações de 1,2 e 3 dias.

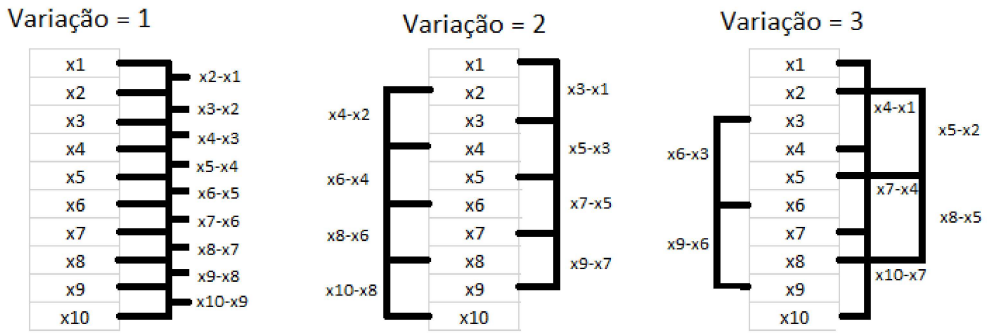

Figura 4.11 – Método de variação entre diferentes dias

Observa-se, que ao aumentar as quantidades de dias de diferença, surgem novas séries temporais. Se há a variação entre dois dias, haverá duas séries temporais, três dias está para três séries temporais, e assim por diante. Dessa maneira, teremos valores pequenos com média zero, desvio padrão de valor baixo e uma série temporal com menos ruído presente. Mesmo havendo valores fora da do padrão, ou comumente chamados de "outliers", esses

valores não podem ser eliminados, uma vez que em uma série temporal há autocorrelação e essas mudanças podem influenciar em valores futuros. Na figura abaixo, temos exemplos de dados variados em 1,5 e 10 dias, onde para 5 e 10 dias foram escolhidos uma serie temporal arbitrariamente.

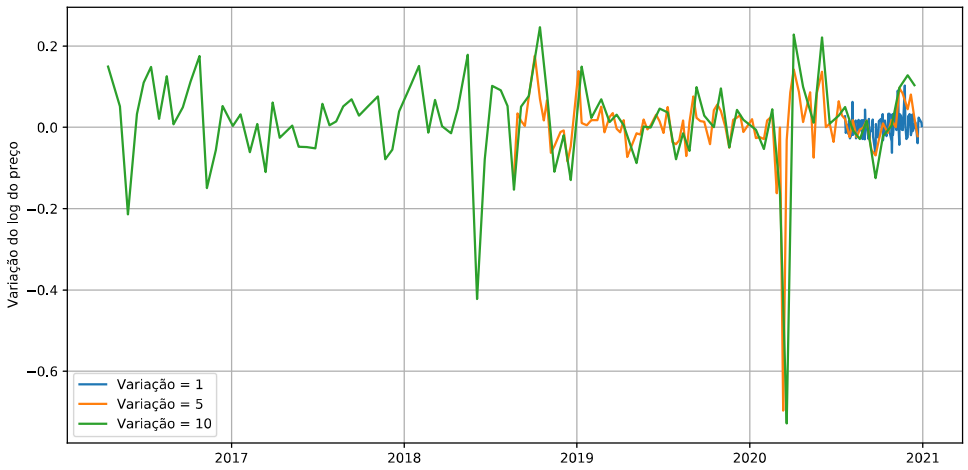

Figura 4.12 – Séries temporais para variações do log do preço de 1,5 e 10 dias.

Nota-se também, que com o aumento das variações a quantidade de tempo envolvida também será maior. Portanto, quando se trata de variação de 1 dia, estamos considerando todos os dias de funcionamento da bolsa, enquanto que para variação de 10 dias, as operações serão realizadas no máximo 2 vezes ao mês. Outra informação que podemos perceber, são os valores maiores presente na variação de 10 dias quando comparado aos valores da variação de 1 dia. Fisicamente falando, se observamos entre 2020 e 2021, há uma redução do ruído ao aumentar o tamanho das variações, porém, pode acontecer de haver muita perda de informação. No capítulo **5** será visto os resultados para cada uma das variações.

4.4 CONJUNTO DE SÉRIES TEMPORAIS

Sabe-se que além da existência da autocorrelação em um série temporal de um ativo, há também correlações entre ativos. Muitos mercados são influenciados por outros mercados, como é caso da bolsa brasileira Ibovespa em relação a bolsa americana $S\&P500$. Não somente isso, mas o dólar influência em todos os mercados como uma das moedas mais estáveis. Portanto, será utilizado diferentes ações, índices, moedas e commodities na previsão de um da PETR4, na tentativa de que as redes neurais possam encontrar correlações e informações que ajudem na previsão. Os dados a serem utilizados estão descritos na secção **4.9**, sendo ao todo 92 séries temporais considerando a abertura, máximo, mínimo e fechamento

de cada item.

4.5 ANÁLISE DA COMPONENTE PRINCIPAL (PCA)

Sabe-se que os ativos da bolsa são influenciados por outros, assim como pelo índice da bolsa de valores ao qual está presente. Mas há, também, influência entre as bolsa de valores do mundo, uma vez que boa parte dos investidores no Brasil são estrangeiros. Outro fato, é que a economia do Brasil é dependente de commodities e seu principal comprador, é o mercado Chinês. Com isso, caso o índice *Xangai Composite* sofra uma queda em seus pontos, irá refletir no índice *Ibovespa*. Contudo, o mercado brasileiro sofre grande influência nas tendências dos ativos, e portanto podemos utilizar as informações de outras séries temporais para aprimorar o aprendizado da rede neural. Isso pode ser realizado utilizando o método de Análise da Componente Principal, ou PCA. Esta ferramenta permite reduzir a dimensão dos dados sem muita perda de informação, sendo uma maneira de destacar as similaridades e diferenças quando se tem muitas dimensões ou séries temporais (o que não é possível ver graficamente). A secção seguinte é baseada no livro de **Smith[42]**.

4.5.1 Aplicação

Imagine que se tenha um conjunto de dados composto por 2 dimensões dado na tabela abaixo (**4.1**), para que se possa observar graficamente o que o método pode afetar.

X	Y
$2,5$	$2,4$
$0,5$	$0,7$
$2,2$	$2,9$
$1,9$	$2,2$
$3,1$	$3,0$
$2,3$	$2,7$
$2,0$	$1,6$
$1,0$	$1,1$
$1,5$	$1,6$
$1,1$	$0,9$

Tabela 4.1 – Dados X e Y.

Agora, subtraia a média nos dados para cada dimensão, onde a média é respectiva a cada dimensão, ou seja, $(x - \bar{x})$ e $(y - \bar{y})$. Feito isso, calcula-se a matriz de covariância que é dada por

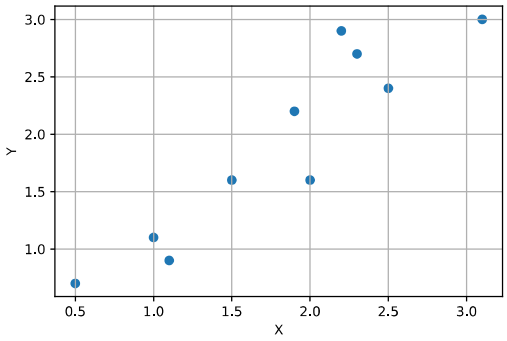

Figura 4.13 – Plote do conjunto de dados

$$\mathbf{c^{n\,x\,n}} = \begin{pmatrix} cov(x,x) & cov(x,y) \\ cov(y,x) & cov(y,y) \end{pmatrix}$$

onde n é a dimensão dos dados e

$$cov(x,y) = \frac{\sum_i^n (x_i - \bar{x})(y_i - \bar{y})}{(n-1)} \tag{4.12}$$

, sendo que $cov(x,y) = cov(y,x)$. Se calcularmos a matriz covariância de **??**, teremos a seguinte matriz

$$\mathbf{c^{2\,x\,2}} = \begin{pmatrix} 0.616555556 & 0.615444444 \\ 0.615444444 & 0.716555556 \end{pmatrix} \quad , n = 2$$

como, os elementos fora da diagonal são positivos, a variável x cresce na medida que y aumenta. Feito tudo, calcula-se os autovalores e os autovetores da matriz de correlação como feito em **Smith[42]**, lembrando que só se é possível calculá-los quando a matriz é quadrada. Neste caso, teremos que

$$\mathbf{autovalores} = \begin{pmatrix} 0.0490833989 \\ 1.28402771 \end{pmatrix}$$

$$\mathbf{autovetores} = \begin{pmatrix} -0.735178656 & -0.677873399 \\ 0.677873399 & -0.735178656 \end{pmatrix}.$$

O gráfico apresentado abaixo, mostra o plote dos dois autovetores como linhas na diagonal e perpendiculares entre si, onde um deles (o de maior autovalor) consegue fitar bem os dados normalizados mostrando a relação entre as duas variáveis, ao contrário do outro autovetor(de menor autovalor). Se expressarmos os dados em função deste autovetor que os relaciona ou componente principal, teremos dados que podem facilitar no treinamento e previsão da rede neural.

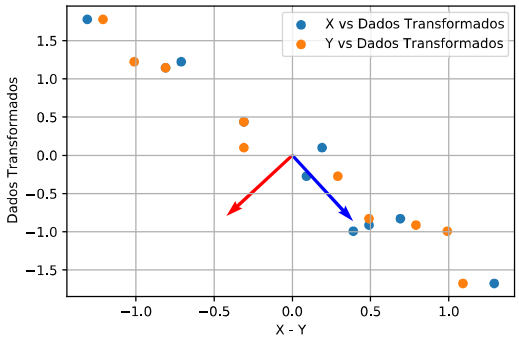

Figura 4.14 – Plote dos dados transformados e dos autovetores da matriz de correlação

Ordenando os autovetores pelos autovalores de forma crescente, há a opção de ignorar os pequenos autovalores. Haverá perda de informação, mas se o autovalor é pequeno não será muita. Retirando esses pequenos n autovalores, automaticamente estará reduzindo n dimensões dos dados. Com isso, deve-se formar um vetor com os autovetores nas colunas

$$\textit{vetor de características} = \begin{pmatrix} autovetor_1 & autovetor_2 & autovetor_3 & \dots & autovetor_n \end{pmatrix}.$$

No exemplo dado, como há dois autovetores se pode escolher formar o vetor de características com os dois ou só com o de maior autovalor, que no caso é

$$\textit{vetor de características} = \begin{pmatrix} -0.677873399 \\ -0.735178656 \end{pmatrix}.$$

Para se obter os dados originais novamente, basta fazer o que está descrito na equação a seguir. Contudo, deve ser ressaltado que há perda de informação, dependendo da quantidade de autovetores que forem escolhidos.

$$\textit{Dados Originais} = \left(\textit{vetor de características}^T \times \textit{Dados transformados}\right) + \textit{Média} \quad (4.13)$$

Na prática podemos ver a aplicação do PCA no ativo *PETR4* na figura a seguir, podendo-se ver que a uma perda de informação na transformação inversa, como podemos ver uma pequena diferença entre os dados originais e transformação inversa.

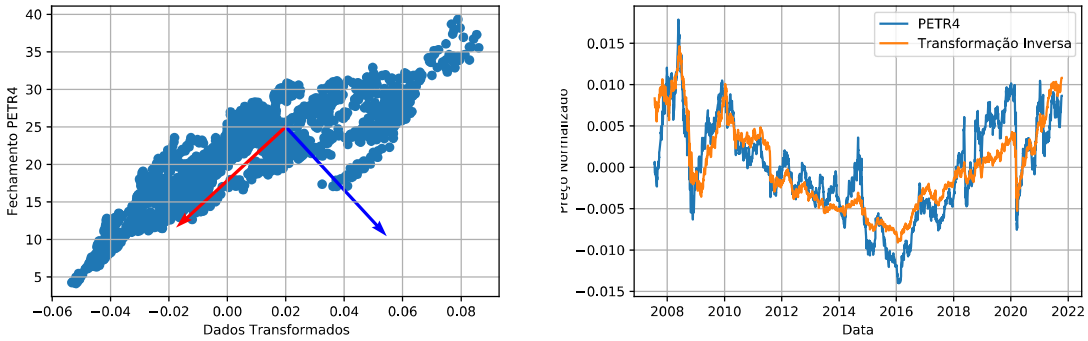

Figura 4.15 – A esquerda os dados transformados por PCA do fechamento do ativo PETR4 e a direita a transformação inversa.

4.6 CRIANDO O CONJUNTO DE DADOS DE TREINAMENTO E PREVISÃO

É importante descrever como serão realizados as entradas e saídas de dados das redes neurais. Considerando que tenhamos uma única série temporal, deve-se escolher o tamanho da entrada para a rede neural. Foi visto em 2 que há autocorrelação na PETR4 em até 500 dias, no entanto, considerando as métricas "Hits" e "Profit", os melhores valores se encontram com entradas de 200 à 600 dados, como vemos nas figuras logo abaixo

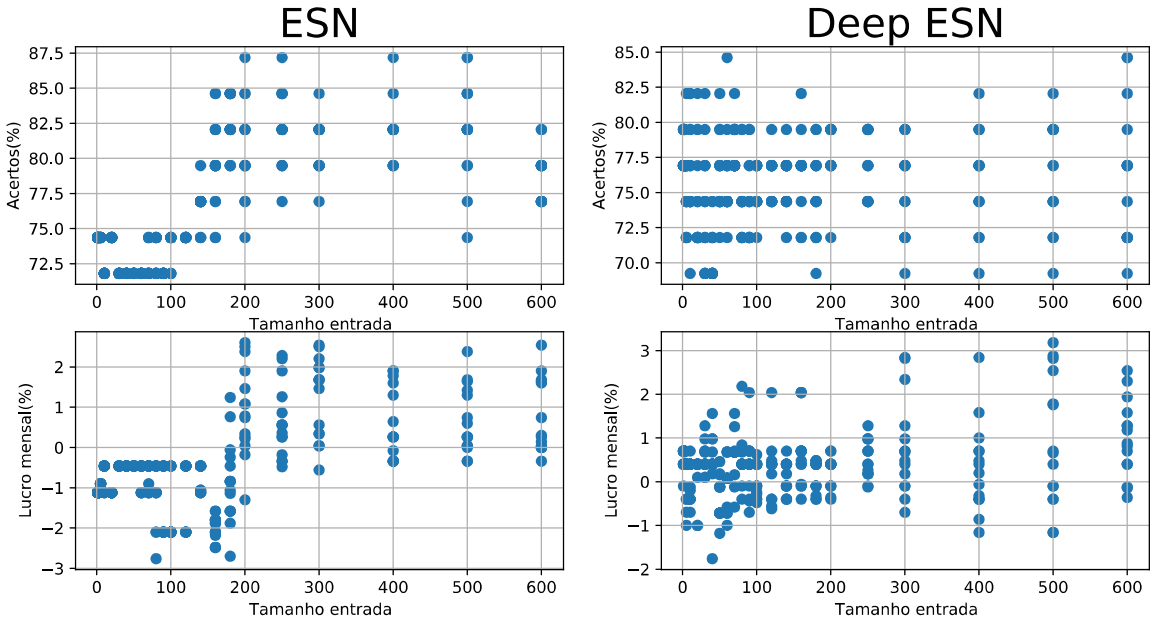

Figura 4.16 – Entrada vs Hits e Entrada vs Lucro

Além disso, será realizado o esquema de entrada e saída descrito na imagem 4.17. Após

cada época de treinamento será realizado uma previsão, sendo que a cada iteração a janela de treinamento aumenta em um dado a frente.

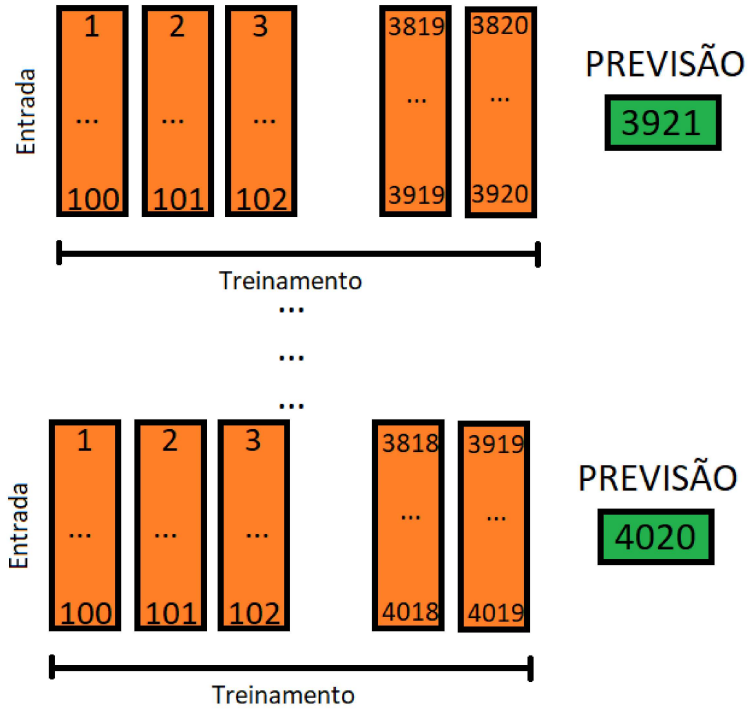

Figura 4.17 – Treinamento e previsão com uma série temporal

Já quando se trata de múltiplas séries temporais o esquema é um pouco diferente. As entradas são um dado de cada série temporal, como é mostrado na figura 4.18, onde S_i denomina cada série temporal e T_i são os dados da série temporal alvo, ou que se deseja prever.

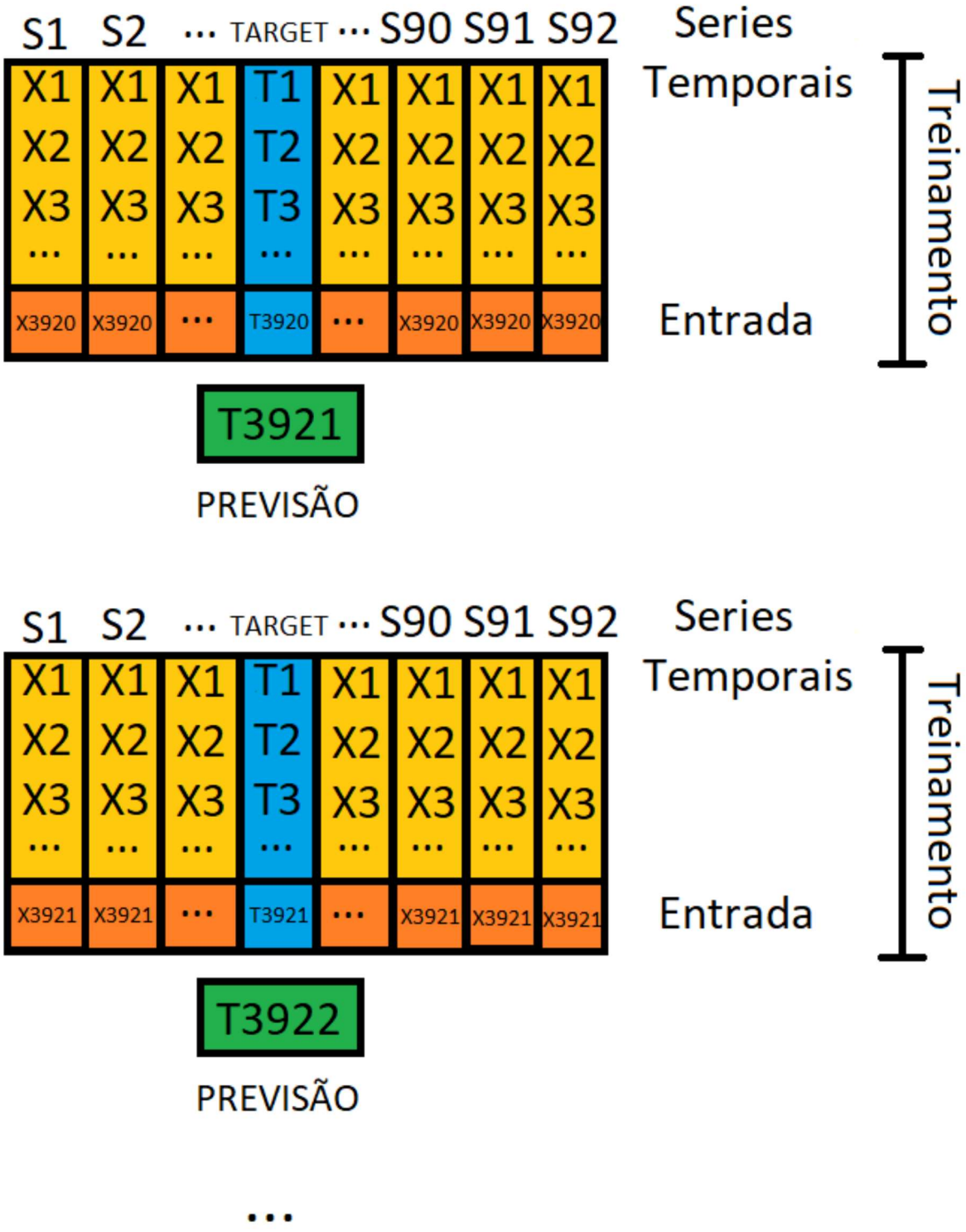

Figura 4.18 – Treinamento e previsão com múltiplas séries temporais

4.7 HARDWARE UTILIZADO

O computador utilizado para executar os algoritmos é um do tipo desktop, contendo os seguintes componentes:

- Memória RAM kllisre de 16 Gb e 1866 MHz
- Placa mãe MSI Z77A-G43
- Processador Intel Core I7-3770
- Placa de vídeo nvidea GTX 660 TI
- Fonte Corsair 500W reais
- SSD kunup 1Tb
- Windowns 10 versão 21H1

Apesar de se ter uma placa de vídeo, ela não foi utilizada pra a execução dos algoritmos, devido a sua versão ser antiga e não possuir compatibilidade com bibliotecas como Pytorch ou TensorFlow.

4.8 LINGUAGEM DE PROGRAMAÇÃO

É possível criar um código de uma rede neural em diferentes linguagens, tanto alto nível quanto baixo nível. No entanto, uma será mais fácil de se criar, embora com uma execução mais lenta e a outra será mais difícil de se criar, entretanto será mais rápida a execução. Dentre as linguagens de programação, Python foi escolhido pela facilidade de se encontrar inúmeras bibliotecas que lidam com processamento de dados, cálculos, funções e até mesmo arquiteturas de redes neurais. Além disso, há também a facilidade de se escrever os códigos devido a linguagem ser de alto nível (Mais próxima da linguagem falada). No entanto, surge o problema da menor velocidade de execução do código, como mencionado antes, ao comparar com linguagens de baixo nível como C++ (Mais próxima da linguagem das máquinas). Mas como os códigos não são extensos exigindo poder computacional apenas nas operações matriciais de inversão, é possível lidar com o tempo de execução do Python.

4.9 ORIGEM DOS DADOS

Para obter os dados foi usado uma biblioteca/API (Application Programming Interface) denominada "Alpha Vantage", onde se pode obter séries temporais de ativos tanto intraday

(durante o dia) como daily (diariamente), bem como de índices, moedas, commodities, entre outras. É possível obter tais dados de diferentes bolsas de valores, existentes no mundo. No caso, será usado os dados diários, onde é constituído por 5 séries temporais: valor de abertura, máximo do dia, mínimo do dia, valor de fechamento e volume negociado. O objetivo principal é prever a queda ou a elevação do valor de fechamento do ativo. Foi feito um pré-processamento desses dados, a fim de encontrar valores incoerentes com o real como valores negativos, valores muito grandes/ pequenos, letras, dentre outras coisas. Os dados escolhidos para compor os métodos são os que seguem na lista abaixo.

- Ambev SA (ABEV3 - Stock BM&FBovespa)
- Banco Bradesco SA (BBDC3 - Stock BM&FBovespa)
- Centrais Elétricas Brasileiras SA (EBR - Stock NYSE)
- Embraer SA (EMBR3 - Stock BM&FBovespa)
- Itausa (ITSA4 - Stock BM&FBovespa)
- Petróleo Brasileiro SA PN (PETR4 - Stock BM&FBovespa)
- Vale SA (VALE3- Stock BM&FBovespa)
- Bank of America Corp (BAC - Stock NYSE)
- JPM (JPMorgan Chase & Co - Stock NYSE)
- US Dollar Brazil Real (USD/BRL - FX)
- Bovespa (BVSP - Index BM&FBovespa)
- Ibovespa Futuros (IBov - Index Futuro BM&FBovespa)
- ASX Small Ordinaries (AXSO - Index Sydney)
- Hang Seng (HK50 - Index Hong Kong)
- DAX (GDAXI - Index Xetra)
- Euronext 100 (N100 - Index Paris)
- JASDAQ (JSD - Index Tokyo)
- Nikkei 225 (JP225 - Index Tokyo)
- NASDAQ Composite (IXIC - Index NASDAQ)
- Brent Oil Futures (B - Commodities ICE)

- Crude Oil WTI Futures (T - Commodities ICE)
- Heating Oil Futures (NYF - Commodities ICE)
- Natural Gas Futures (NG - Commodities NYMEX)

Foram escolhidos ações, índices e comodities de diferentes continentes, com o intuito de assegurar informação suficiente para que a rede neural consiga captar o passo seguinte da previsão. Caso contrário, a rede neural irá estabelecer que o valor mais provável para a próxima previsão seja o último valor, o que não é interessante que aconteça.

5 RESULTADOS E DISCUSSÕES

5.1 PARÂMETROS DA ECHO STATE NETWORK

Como método de otimização, decidiu-se utilizar o Grid Search, Random Grid Search e Nelder Mead, por conta da suas facéis implementações. Os diferentes métodos apresentados, tiveram diferentes desempenhos, o método Nelder Mead frequentemente se mostrou encontrar mínimos/máximos locais ao invés de mínimos/máximos globais das métricas empregadas, o que impossibilita a averiguação de diferentes regiões/hiperparâmetros, uma solução seria utilizar o método Nelder Mead Estocástico onde é trabalhado diferentes "ensembles" de hiperparâmetros, mas por ser um método derivativo sua implementação se mostrou custosa. Já o método RGS se mostrou um pouco melhor que o método Nelder Mead no quesito diferentes regiões do domínio, no entanto, as diferentes mudanças nos diferentes parâmetros não apresentou uma tendência clara em relação as métricas apresentadas. Por fim, decidiu-se fixar os parâmetros e variar apenas um dentre eles, o que configura um Grid Search, resultando nos gráficos da figura 5.1. Esta otimização de parâmetros foi realizada considerando uma única série temporal e os parâmetros escolhidos estão contidos na tabela **5.1**.

Nota-se aqui que os parâmetros para as duas redes neurais, ESN e Deep ESN, não se comportam da mesma maneira. Para a ESN, o aumento do número de reservatórios apresenta uma melhora no lucro mensal enquanto que para a Deep ESN, a tendência é o contrário. Outro caso similar é os valores de β, onde quanto maior, melhores são as métricas para o caso ESN, enquanto que para a Deep ESN os menores valores de β resultam em métricas melhores. Quanto a quantidade de épocas de treinamento e as esparsidades do reservatório, não houve uma relação clara com as métricas. Por fim, podemos notar que quanto maior valor de α para a ESN, melhor são os resultados, o que vai contra os argumentos de que os valores anteriores de fechamento, ou melhor, de anos anteriores influenciam na previsão, uma vez que, pela equação 3.12 valores maiores de α indicam menor influência do último estado do reservatório. Já as camadas da Deep ESN não apresentou nenhuma tendência em relação as métricas "Acertos" e "Lucro mensal".

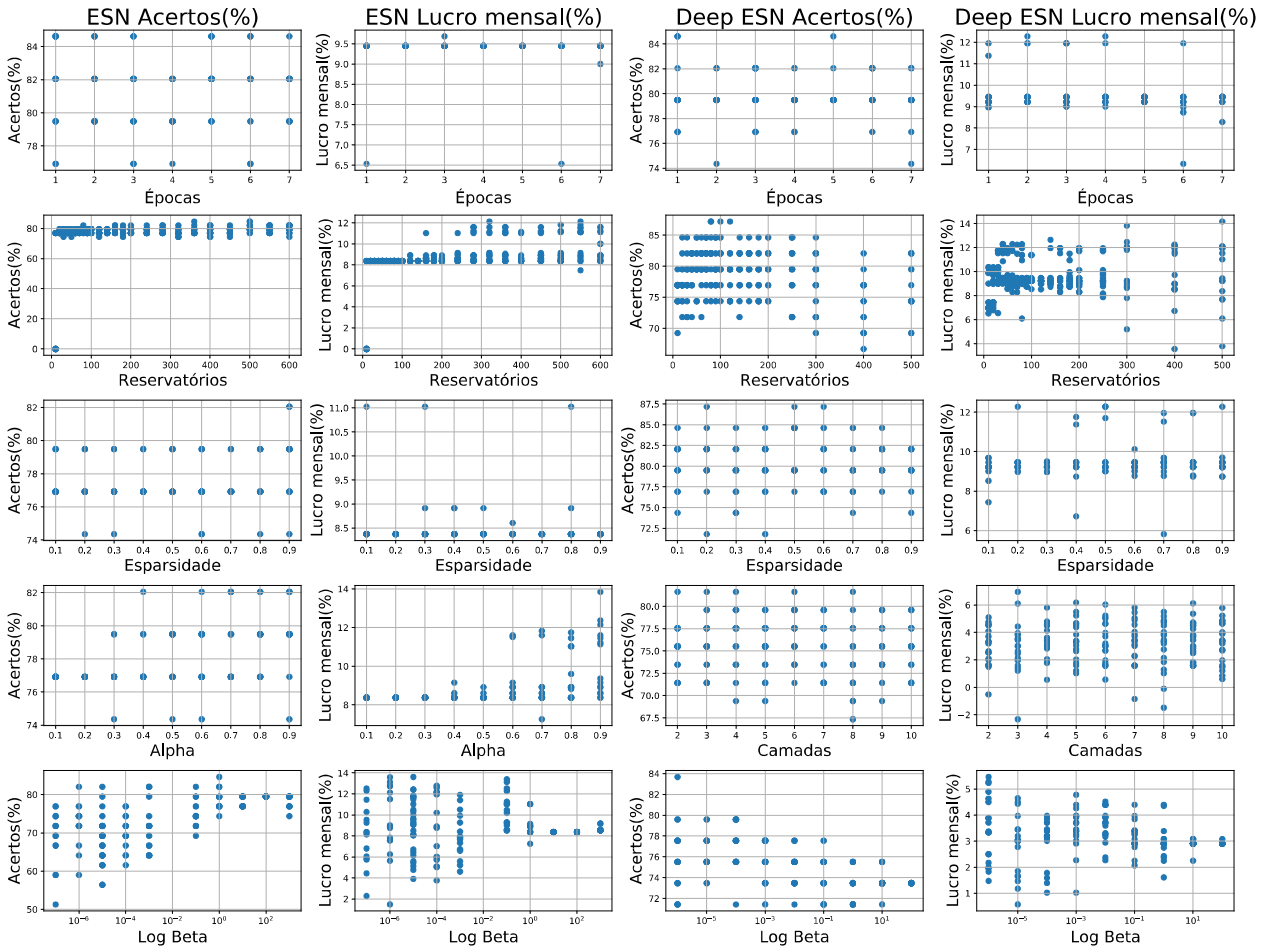

Figura 5.1 – Parâmetros encontrados com o método de Grid Search, com dados de treinamento

Parâmetros	**ESN**	**Deep ESN**
Épocas	3	1
Tamanho Reservatório	150	100
Esparsidade	0.8	0.9
Alpha/Camadas	0.9	4
Log Beta	10^{-1}	10^{-5}

Tabela 5.1 – Parâmetros escolhidos.

5.2 PREVISÕES COM UMA ÚNICA SÉRIE TEMPORAL

Utilizar dados brutos não se apresentou muito satisfatório na previsão ao considerar a métrica Hits, apesar de que graficamente a curva esteja bem ajustada. Ao contrário de muitos trabalhos, ao olhar de perto o ajuste da curva real na figura 5.2, nota-se um atraso dos dados de previsão em relação aos dados reais. Isso se deve, principalmente, pela falta de informação para a rede neural, dessa forma, suas previsões são os valores mais prováveis, ou os valores anteriores, considerando a ESN. Já a Deep ESN se mostrou com uma previsão pobre, que pouco se ajusta a curva real.

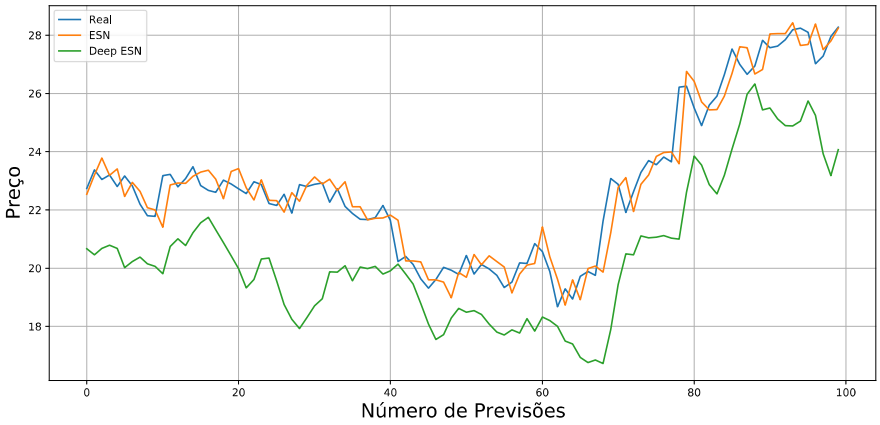

Figura 5.2 – Resultados de previsão PETR4.

Utilizando o método de Mandelbrot, variação do logarítmo, temos o resultado para a ação PETR4 apresentado na figura 5.3. Esta figura apresenta 5 gráficos, todos em função do número de dias de variação. O primeiro é a quantidade de acertos em porcentagem, seguido do MSE, R_2, porcentagem de lucro mensal e quantidade de meses ao qual foram feitas previsões. Nota-se que a rede neural Deep ESN tem um desempenho melhor em todas as métricas, exceto em porcentagem de lucro mensal na variação de até 3 dias. Tem uma leve piora também em variações de 10 dias, porém nada expressivo em comparação com a ESN. A quantidade de meses varia de 6 meses, quando não se tem variação (apenas o logaritmo), até 56 meses com variação de 10 dias.

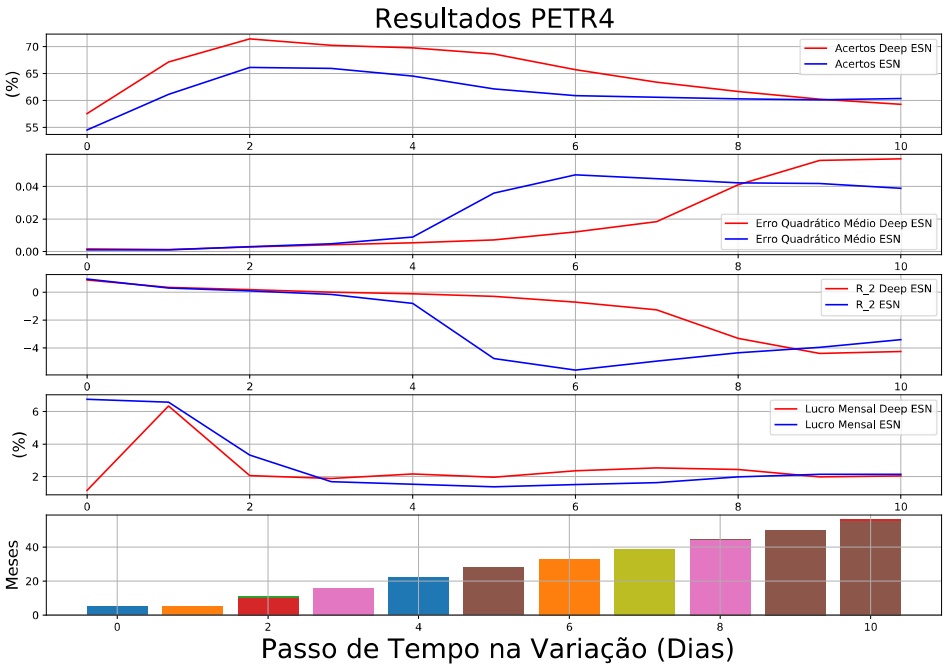

Figura 5.3 – Resultados de previsão PETR4 considerando a variação do logarítmo para diferentes dias de variação.

5.3 PREVISÕES COM MÚLTIPLAS SÉRIES TEMPORAIS

Considerando o que foi dito em **4.4**, utilizar informações de outras séries temporais proporcionou o seguinte resultado apresentado na figura **5.4**. Vemos aqui que em relação a uma única série temporal, houve uma melhora nos acertos, no MSE e no R_2, porém, houve uma queda na porcentagem de lucro mensal para ambas rede neurais. Como no caso anterior, a medida que se aumenta o número de variações pior é o desempenho nas métricas, em excessão da métrica Hits, ao qual os acertos são maiores para variações de 2 à 4 dias. Nota-se ainda, que apesar dos resultados melhores, a Deep ESN se saiu melhor apenas na métrica de porcentagem de lucro mensal em comparação com a ESN, que atingiu melhores resultados em Hits, MSE e R_2. Vale ressaltar que a quantidade de meses para cada variação não foi alterada em relação aos resultados anteriores.

Se adicionarmos o método PCA ao método anterior, como uma série temporal a mais, temos o resultado a seguir em 5.5. Podemos observar que, Neste caso, a Deep ESN levou vantagem na métrica Lucro mensal e para varições pequenas de dias para o restante das métricas, o que indica que variações mais altas possuem poucas informações para previsão. A ESN teve um desempenho inferior, porém, com resultados mais consistentes.

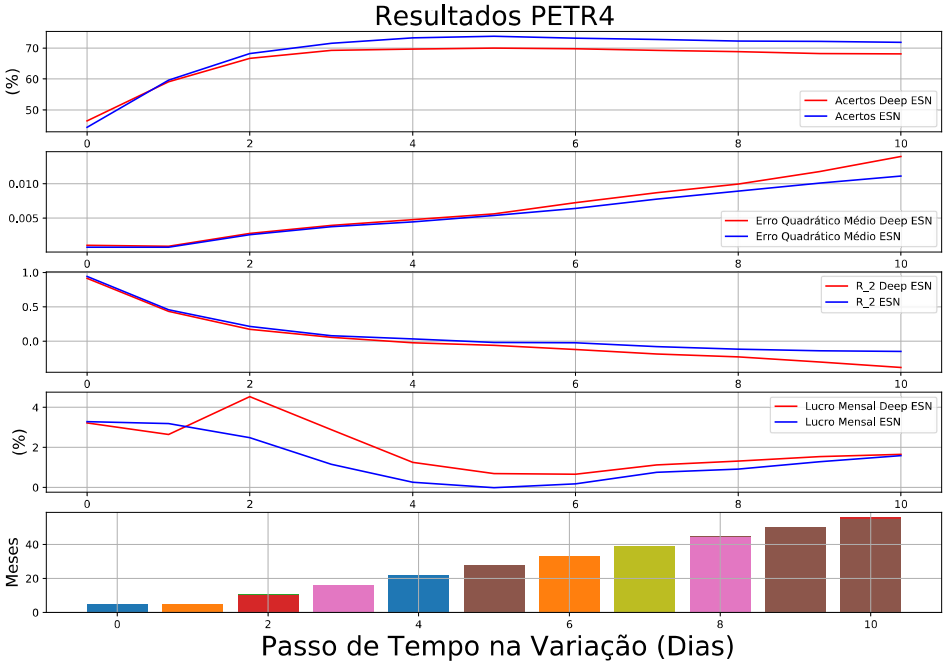

Figura 5.4 – Resultados de previsão PETR4 para diferentes dias de variação e com diferentes séries temporais.

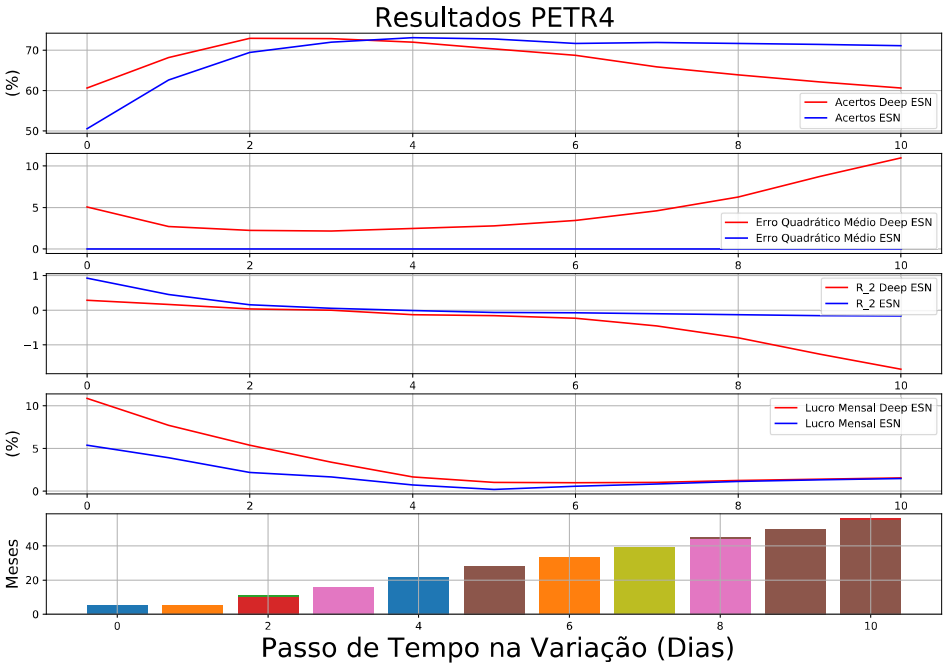

Figura 5.5 – Resultados de previsão PETR4 para diferentes dias de variação e com diferentes séries temporais e PCA.

6 CONCLUSÕES

Aplicar técnicas de machine learning para prever movimentos de preço de séries temporais tem importantes aplicações na indústria financeira (**Bose e Mahapatra[43]**), mas não é algo novo como mostram em **Zuckerman[44]**. Ultimamente, está surgindo novas classes de redes neurais recorrentes como a computação por reservatório (RC) (**Lukosevicius[4]**, **wyffels e Schrauwen[19]**,**Gallicchio, Micheli e Pedrelli[5]**) aqui descrita, redes neurais que contém um reservatório de neurônios conectados entre si, de maneira aleatória ou não, que de acordo com o teorema de Hornik (**Hornik Maxwell Stinchcombe[28]**, **Grigoryeva e Ortega[37]**,**Gonon e Ortega[45]**), tem a capacidade de reproduzir qualquer função matemática devido a sua estrutura não linear. Além disso, RC requer apenas que os pesos de saída da rede neural sejam treinados, mantendo os pesos de entrada e do reservatório fixos, consequentemente, aumentado a velocidade de treinamento. Mas, ainda pouco se vê de suas aplicações em séries temporais de cunho financeiro, ao contrário das famosas estruturas de Machine Learning (ML) e Deep Learning (DL), como os modelos LSTM e FFNN.

Neste trabalho, desenvolvemos dois modelos de RC utilizando topologias aleatórias, para investigar a capacidade de previsão de um dia/dado a frente durante 100 dias/dados e por sua vez, o lucro mensal neste mesmo período. Os modelos desenvolvidos foram a Echo State Network e a Deep Echo State Network. A série temporal prevista foi o preço de fechamento da ação da Petrobras (PETR4) com cerca de 4000 dias, com início em 2002 e fim em 2020. Utilizou-se diferentes séries temporais, bem como diferentes escalas de tempo (variações de 1 à 10 dias) e transformações (Logarítmo e PCA) para a previsão da mesma. As métricas consideradas para este trabalho foram *MSE*, R^2, *Acertos* e *Lucro Mensal*, onde as duas ultimas foram criadas para fins de estratégia de investimento e quantificação de retorno. Comparando-se as duas redes neurais, a DESN demonstrou resultados melhores em relação a ESN considerando as métricas criadas, *Acertos* e *Lucro Mensal*. Já levando em conta as métricas *MSE* e R^2, a ESN demonstrou melhores resultados para as diferentes escalas de tempos propostas. Em termos de velocidade de treinamento, a ESN fica a frente da DESN, uma vez que esta possui 4 reservatórios interligados, e portanto mais interações matemáticas a serem computadas.

No geral, o método proposto por Mandelbrot melhora as previsões, principalmente quando se trata de utilizar dados brutos, sem nenhuma transformação. No entanto, percebe-se que quantidade de acertos mais baixas produzem mais lucro, sendo que o esperado seria uma proporcionalidade entre essas duas métricas. Pode-se dizer que em casos de porcentagem de acertos mais altos ocorra grandes investimentos no momento em que se tem previsões equivocadas resultando em um menor lucro. Consequentemente, ocorre o inverso, quando

se acerta poucas previsões, mas uma previsão correta pode retornar um grande retorno. Podemos ver isso no apêndice A, onde estão guardadas operações para a previsão da ESN no gráfico 5.2, onde no dia 19/11/2020 comprou-se 4 cotas (ou 400 ações) à $R\$23,82$ cada ação e vendeu tudo, erroneamente ("score : 0"), em 25/11/2020 à $R\$26,25$. O capital total ("Value" + "capital") nessa situação estava em $R\$10.396,00$, sendo que ao vender este valor subiu para $R\$11.368,00$, ou quase mil reais a mais. Vê-se ainda, que para variações de dias maiores o desempenho de ambas redes neurais se reduzem, o que indica que há pouca informação para ser utilizada nas previsões. Outra observação, é que os resultados para variações menores se mostraram satisfatórios quando se analisado a porcentagem de lucro mensal obtida. Contudo, ambas redes neurais obtiveram retornos positivos para as diferentes escalas de tempo.

Para trabalhos futuros, poderiam ser realizados testes com outros métodos que ampliam séries temporais, como a transformada de Fourier, facilitando o treinamento e previsão devido a fácil regressão de funções senoides e cossenoides, mais ainda, testar outras estratégias de operação no mercado, bem como previsão de outros tipos de séries temporais como índices, commodities, moedas, futuros, ou mesmo as séries temporais de abertura, máximo e mínimo. Pode-se considerar ainda, a aplicação de escalas de tempo diferentes das diárias, como segundos, minutos e horas. A mais, o método de previsão de um dia a frente pode ser modificado para prever a tendência de semanas a frente, bastando realimentar a rede com suas próprias previsões.

Por fim, vê-se que a computação por reservatório pode ser aplicada não somente a áreas da ciência, como propagação de calor, clima, epidemiologia, reconhecimentos de textos, entre outros, mas também para finanças.

REFERÊNCIAS BIBLIOGRÁFICAS

1 FIGUEIREDO, E. A. de; ZIEGELMANN, F. A. MudanÇa na distribuiÇÃo de renda brasileira: SignificÂncia estatÍstica e bem-estar econÔmico. *Economia Aplicada*, v. 13, p. 257–277, 2009.

2 MANDELBROT, B. B.; HUDSON, R. L. *The Misbehavior of the Markets*. [S.l.]: Basic Books, 2006. (-). ISBN 978-0465043576.

3 ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, v. 65, p. 386–408, - 1958.

4 LUKOSEVICIUS, M. A practical guide to applying echo state networks. *Springer*, v. 7700, p. 386–408, 2012.

5 GALLICCHIO, C.; MICHELI, A.; PEDRELLI, L. Deep echo state networks for diagnosis of parkinson's disease. *ArXiv*, abs/1802.06708, 2018.

6 HETHCOTE, H. W. The mathematics of infectious diseases. *SIAM Review*, v. 42, p. 599–653, Dec 2000.

7 KANAZAWA TAKUMI SUESHIGE, H. T. K.; TAKAYASU, M. Kinetic theory for finance brownian motion from microscopic dynamics. *arXiv*, v1, p. 1802.05993v1, Feb 2018.

8 PACKARD J. P. CRUTCHFIELD, J. D. F. N. H.; SHAW, R. S. Geometry from a time series. *PHYSICAL RERIEW LETTERS*, v. 45, p. 712–716, Nov 1979.

9 YOUNG, K. Foreign exchange market as a lattice gauge theory. *American Journal of Physics*, v. 67, p. 862–868, Apr 1999.

10 LI NIKOLA KOVACHKI, K. A. B. L. K. B. A. S. Z.; ANANDKUMAR, A. Fourier neural operator for parametric partial differential equations. *arXiv*, v. 1, p. 2010.08895v1, Oct 2020.

11 SHETA, S. E. M. A. A. F.; FARIS, H. A comparison between regression, artificial neural networks and support vector machines for predicting stock market index. *International Journal of Advanced Research in Artificial Intelligence*, v. 4, p. 55–63, - 2015.

12 DAS, M.; GHOSH, S. K. Data-driven approaches for meteorological time series prediction: A comparative study of the state-of-the-art computational intelligence techniques. *Elsevier*, -, p. 1–10, Aug 2017.

13 FILHO, T. M. da R.; ROCHA, P. Inefficiency of the brazilian stock market: the ibovespa future contracts. *arXiv*, v. 1904.09214v1, Apr 2019.

14 CHANDRA, Y. H. R. Bayesian neural networks for stock price forecasting before and during covid-19 pandemic. *PLoS ONE*, v. 16(7), p. e0253217, March 2021.

15 ZHENG, L.; HE, H. Share price prediction of aerospace relevant companies with recurrent neural networks based on pca. *Expert Systems with Applications*, v. 183, p. 115384, 06 2021.

16 GHASHAMI, F.; KAMYAR, K.; RIAZI, A. Prediction of stock market index using a hybrid technique of artificial neural networks and particle swarm optimization. *Applied Economics and Finance*, v. 8, p. 1, 04 2021.

17 WANG, W.-J. et al. Stock market index prediction based on reservoir computing models. *Expert Systems with Applications*, v. 178, p. 115022, 04 2021.

18 MALLYA, A. Echo state networks and existing paradigms for stock market prediction. In: *2021 International Conference on Emerging Smart Computing and Informatics (ESCI)*. [S.l.: s.n.], 2021. p. 13–18.

19 WYFFELS, F.; SCHRAUWEN, B. A comparative study of reservoir computing strategies for monthly time series prediction. *Neurocomputing*, v. 73, p. 1958–1964, 06 2010.

20 Trierweiler Ribeiro, G. et al. Novel hybrid model based on echo state neural network applied to the prediction of stock price return volatility. *Expert Systems with Applications*, Elsevier Limited, v. 184, jun. 2021. ISSN 0957-4174.

21 KIM, T.; KING, B. Time series prediction using deep echo state networks. *Neural Computing and Applications*, v. 32, 12 2020.

22 MANTEGNA, R. N.; STANLEY, H. E. *An Introduction To Econophysics: Correlations and Complexity in Finance*. [S.l.]: Cambridge University Press, 2000. ISBN 0521620082.

23 HUANG, Z. C. J.; ZHAO, J. An adaptive moving mesh method for a time-fractional black–scholes equation. *SpringOpen Journal*, v. 0, p. 0–0, - 2019.

24 LO, A. W. Long-term memory in stock market prices. *The Econometric Society*, v. 59, p. 1279–1313, Sep 1991.

25 MCCULLOCH, W.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, v. 5, p. 115–133, 1943.

26 HAYKIN, S. *Neural Networks and Learning Machines*. [S.l.]: Pearson, 2008. (-). ISBN 978-0-13-147139-9.

27 MINSKY, M.; PAPERT, S. A. *Perceptrons: An Introduction to Computational Geometry*. [S.l.]: The MIT Press, 1969. (-). ISBN 978-0262130431.

28 HORNIK MAXWELL STINCHCOMBE, H. W. K. Multilayer feedforward networks are universal approximators. *Neural Networks*, v. 2, p. 359–366, March 1989.

29 ALBAWI, S.; MOHAMMED, T. A.; AL-ZAWI, S. Understanding of a convolutional neural network. In: *2017 International Conference on Engineering and Technology (ICET)*. [S.l.: s.n.], 2017. p. 1–6.

30 CHAUHAN, R.; GHANSHALA, K. K.; JOSHI, R. Convolutional neural network (cnn) for image detection and recognition. In: *2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)*. [S.l.: s.n.], 2018. p. 278–282.

31 LI, P.; LI, J.; WANG, G. Application of convolutional neural network in natural language processing. In: *2018 15th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*. [S.l.: s.n.], 2018. p. 120–122.

32 GRAVES, A. *Supervised Sequence Labelling with Recurrent Neural Networks*. [S.l.: s.n.], 2012. v. 385. ISBN 978-3-642-24796-5.

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

34 INFORMATIK, F. et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. *A Field Guide to Dynamical Recurrent Neural Networks*, 03 2003.

35 LOYE, G. *Gated Recurrent Unit (GRU) With PyTorch*. Disponível em: <https://blog.floydhub.com/gru-with-pytorch/>.

36 HART, A.; HOOK, J.; DAWES, J. Embedding and approximation theorems for echo state networks. 08 2019.

37 GRIGORYEVA, L.; ORTEGA, J.-P. Echo state networks are universal. *Neural Networks*, Forthcoming, 08 2018.

38 OLIPHANT ALEX GRIFFING, M. v. K. A. H. N. S. J. T. P. V. J. F. d. R. T. *Numpy*. Disponível em: <https://numpy.org/doc/stable/index.html>.

39 TSANG, P. et al. Design and implementation of nn5 for hong kong stock price forecasting. *Engineering Applications of Artificial Intelligence*, v. 20, p. 453–461, 06 2007.

40 JOGLEKAR, S. *Nelder-Mead Optimization*. Disponível em: <https://codesachin.wordpress.com/2016/01/16/nelder-mead-optimization/>.

41 GAO, F.; HAN, L. Implementing the nelder-mead simplex algorithm with adaptive parameters. *Comput Optim Appl*, Jan 2010.

42 SMITH, L. I. *A tutorial on Principal Components Analysis*. [S.l.]: Department of Computer Science, University of Otago, 2002.

43 BOSE, I.; MAHAPATRA, R. Business data mining - a machine learning perspective. *Information Management*, v. 39, p. 211–225, 12 2001.

44 ZUCKERMAN, G. *The Man Who Solved the Market: How Jim Simons Launched the Quant Revolution*. [S.l.]: Penguin UK, 2019. (9780241981344). ISBN 0241981344.

45 GONON, L.; ORTEGA, J.-P. Fading memory echo state networks are universal. *Neural Networks*, Forthcoming, 10 2020.

APPENDIX

A OPERAÇÕES NO MERCADO

Data	Quantidade	Valor da ação	Valor Total	Natureza	Capital	Acertos
22.07.2020	400	23.05	9220.0	buy	780.0	1
27.07.2020	400	23.2	9280.0	sem operação	780.0	0
28.07.2020	400	22.8	9120.0	sem operação	780.0	1
29.07.2020	400	23.17	9268.0	sem operação	780.0	0
30.07.2020	400	22.82	9128.0	sell	9908.0	0
31.07.2020	0	22.2	0	sem operação	9908.0	1
03.08.2020	0	21.8	0	sem operação	9908.0	1
04.08.2020	0	21.78	0	sem operação	9908.0	1
05.08.2020	400	23.18	9272.0	buy	636.0	0
06.08.2020	400	23.22	9288.0	sell	9924.0	0
07.08.2020	0	22.79	0	sem operação	9924.0	0
11.08.2020	0	23.08	0	sem operação	9924.0	1
12.08.2020	400	23.48	9392.0	buy	532.0	1
13.08.2020	400	22.84	9136.0	sem operação	532.0	1
14.08.2020	400	22.67	9068.0	sell	9600.0	1
17.08.2020	400	22.6	9040.0	buy	560.0	0
18.08.2020	400	23.02	9208.0	sem operação	560.0	0
19.08.2020	400	22.89	9156.0	sell	9716.0	0
20.08.2020	400	22.73	9092.0	buy	624.0	0
21.08.2020	400	22.56	9024.0	sem operação	624.0	0
24.08.2020	400	22.97	9188.0	sem operação	624.0	0
25.08.2020	400	22.87	9148.0	sell	9772.0	0
26.08.2020	400	22.22	8888.0	buy	884.0	0
27.08.2020	400	22.15	8860.0	sell	9744.0	1
28.08.2020	400	22.54	9016.0	buy	728.0	0
31.08.2020	400	21.89	8756.0	sell	9484.0	0
01.09.2020	400	22.87	9148.0	buy	336.0	0
02.09.2020	400	22.8	9120.0	sem operação	336.0	1
03.09.2020	400	22.88	9152.0	sell	9488.0	1
04.09.2020	400	22.92	9168.0	buy	320.0	1
08.09.2020	400	22.26	8904.0	sem operação	320.0	1
09.09.2020	400	22.73	9092.0	sem operação	320.0	0

Data	Quantidade	Valor da ação	Valor Total	Natureza	Capital	Acertos
10.09.2020	400	22.12	8848.0	sem operação	320.0	1
11.09.2020	400	21.88	8752.0	sem operação	320.0	0
14.09.2020	400	21.68	8672.0	sell	8992.0	1
15.09.2020	400	21.67	8668.0	buy	324.0	0
16.09.2020	400	21.73	8692.0	sell	9016.0	1
17.09.2020	400	22.15	8860.0	buy	156.0	1
18.09.2020	400	21.65	8660.0	sell	8816.0	0
23.09.2020	0	20.23	0	sem operação	8816.0	1
24.09.2020	400	20.4	8160.0	buy	656.0	0
25.09.2020	400	20.13	8052.0	sem operação	656.0	1
28.09.2020	400	19.63	7852.0	sell	8508.0	1
29.09.2020	400	19.31	7724.0	buy	784.0	0
30.09.2020	400	19.61	7844.0	sell	8628.0	1
05.10.2020	400	20.03	8012.0	buy	616.0	1
06.10.2020	400	19.93	7972.0	sell	8588.0	0
07.10.2020	0	19.79	0	sem operação	8588.0	1
08.10.2020	0	20.44	0	sem operação	8588.0	1
09.10.2020	0	19.8	0	sem operação	8588.0	0
13.10.2020	400	20.13	8052.0	buy	536.0	0
14.10.2020	400	19.97	7988.0	sem operação	536.0	1
15.10.2020	400	19.75	7900.0	sem operação	536.0	0
16.10.2020	400	19.33	7732.0	sem operação	536.0	0
19.10.2020	400	19.52	7808.0	sem operação	536.0	0
20.10.2020	400	20.18	8072.0	sell	8608.0	0
21.10.2020	400	20.16	8064.0	buy	544.0	1
22.10.2020	400	20.84	8336.0	sell	8880.0	1
23.10.2020	0	20.57	0	sem operação	8880.0	0
27.10.2020	400	19.88	7952.0	buy	928.0	0
28.10.2020	400	18.67	7468.0	sell	8396.0	1
29.10.2020	0	19.29	0	sem operação	8396.0	1
30.10.2020	400	18.94	7576.0	buy	820.0	1
04.11.2020	400	19.72	7888.0	sem operação	820.0	0
05.11.2020	400	19.89	7956.0	sell	8776.0	0
06.11.2020	400	19.75	7900.0	buy	876.0	1
09.11.2020	400	21.61	8644.0	sem operação	876.0	0
10.11.2020	400	23.08	9232.0	sem operação	876.0	1

Data	Quantidade	Valor da ação	Valor Total	Natureza	Capital	Acertos
11.11.2020	400	22.88	9152.0	sell	10028.0	0
12.11.2020	0	21.91	0	sem operação	10028.0	1
13.11.2020	400	22.63	9052.0	buy	976.0	0
16.11.2020	400	23.29	9316.0	sem operação	976.0	1
17.11.2020	400	23.69	9476.0	sem operação	976.0	1
18.11.2020	400	23.55	9420.0	sell	10396.0	0
19.11.2020	400	23.82	9528.0	buy	868.0	0
20.11.2020	400	23.65	9460.0	sem operação	868.0	1
24.11.2020	400	26.22	10488.0	sem operação	868.0	0
25.11.2020	400	26.25	10500.0	sell	11368.0	0
27.11.2020	400	25.5	10200.0	buy	1168.0	1
30.11.2020	400	24.9	9960.0	sem operação	1168.0	0
01.12.2020	400	25.6	10240.0	sem operação	1168.0	0
02.12.2020	400	25.91	10364.0	sem operação	1168.0	1
03.12.2020	400	26.64	10656.0	sell	11824.0	0
04.12.2020	0	27.53	0	sem operação	11824.0	0
07.12.2020	400	27.0	10800.0	buy	1024.0	1
08.12.2020	400	26.66	10664.0	sem operação	1024.0	0
09.12.2020	400	26.94	10776.0	sem operação	1024.0	0
10.12.2020	400	27.82	11128.0	sem operação	1024.0	1
11.12.2020	400	27.57	11028.0	sell	12052.0	0
14.12.2020	400	27.62	11048.0	buy	1004.0	0
15.12.2020	400	27.85	11140.0	sem operação	1004.0	1
16.12.2020	400	28.19	11276.0	sem operação	1004.0	1
17.12.2020	400	28.24	11296.0	sem operação	1004.0	1
18.12.2020	400	28.1	11240.0	sell	12244.0	0
21.12.2020	0	27.02	0	sem operação	12244.0	1
22.12.2020	400	27.28	10912.0	buy	1332.0	0
23.12.2020	400	27.95	11180.0	sem operação	1332.0	1
29.12.2020	400	28.27	11308.0	sem operação	1332.0	1
30.12.2020	400	28.34	11336.0	sem operação	1332.0	1

B ALGORÍTMOS CRIADOS E UTILIZADOS

Algoritmo para utilizar a rede neural ESN:

```
import numpy as np

class ESN:

    def __init__(self,input_scaling,epochs,N_u,N_y,N_r,sparsity = 0.2,
    alpha = 0.1, beta = 3, verbose = True, method = 'ridge_regression',TF
    = True):
        # Parameters

        self.input_scaling = input_scaling
        self.epochs = epochs # epochs
        self.N_u = N_u # input length
        self.N_y = N_y # output length
        self.N_r = N_r # reservoir size
        self.sparsity = sparsity # sparsity to resevoir to fast computing
        self.beta = beta  #coefficient for the regularization in the
    ridge regression
        self.x = np.zeros((N_r,1)) # the state of the echo state
        self.alpha = alpha # the leaky parameter for each iteration in
    the echo state
        self.verbose = verbose # show logs during the running of the deep
     echo
        self.method = method # method used for training the readout
        self.TF = TF # Teacher Forcing
        self.y_tf = 0

        # Initializing weights

        ## Initiate the W_in
        self.W_in = np.random.uniform(low=-self.input_scaling,high=self.
    input_scaling, size=(self.N_r,self.N_u))

        ## Initiate the reservoir weights
        self.W_reservoir = np.random.uniform(size=(self.N_r,self.N_r))
        self.W_reservoir[np.random.uniform(size=(self.N_r,self.N_r))<self
    .sparsity]=0

        ### Computing the spectral radius and applying
        spectral_radius = np.max(np.abs(np.linalg.eigvals(self.
    W_reservoir)))
        self.W_reservoir = self.W_reservoir/spectral_radius
```

```
34
35          ## Iniciate the W_feedb weights
36          self.W_feedb = np.random.uniform(size = (self.N_r, self.N_y))
37
38          ## Initiate the W_out weights
39          self.W_out = np.random.uniform(size = (self.N_y, 1 + self.N_u +
    self.N_r))
40
41          if self.verbose:
42              print('W_reservoir.shape: ', self.W_reservoir.shape)
43              print('W_in.shape: ', self.W_in.shape)
44              print('W_feedb.shape',self.W_feedb.shape)
45              print('W_out.shape: ', self.W_out.shape)
46          else:
47              pass
48
49
50      # Function to update the state of the reservoir
51      def update(self,reservoir_state,input,output):
52          ## Using teacher forcing
53          if self.TF:
54              reservoir_state = (1-self.alpha)*reservoir_state + self.alpha
    *np.tanh(self.W_in @ input + self.W_reservoir @ reservoir_state + self
    .W_feedb @ np.reshape(output,(1,1)))
55
56          ## Without teacher forcing
57          else:
58              reservoir_state = (1-self.alpha)*reservoir_state + self.alpha
    *np.tanh(self.W_in @ input + self.W_reservoir @ reservoir_state)
59
60          return reservoir_state
61
62
63      # Function to train the echo state
64      def train(self,train_input,train_label):
65
66          # Initiate the design matrix
67          design_matrix = np.zeros((1 + self.N_u + self.N_r, train_input.
    shape[0]))
68
69          ## iteration over epochs
70          for epoch in range(self.epochs):
71
72              self.y_tf = 0
73
74              ## Iteration over the train data
75              for i in range(train_input.shape[0]):
```

```
76
77                  self.x = self.update(self.x,train_input[i][:, np.newaxis
    ],self.y_tf)
78                  self.y_tf = train_label[i]
79
80                  ## Update the design matrix
81                  design_matrix[:, i] = np.squeeze(np.vstack([1,train_input
    [i][:, np.newaxis],self.x]))
82
83              ## Compute the output
84              output = np.squeeze(self.W_out @ design_matrix)
85
86              ## Update W_out
87              if self.method == 'ridge_regression':
88                  self.W_out = train_label.T @ design_matrix.T @ np.linalg.
    inv(design_matrix @ design_matrix.T + self.beta*np.identity(1 + self.
    N_u + self.N_r))
89
90              if self.method == 'pseudo_inverse':
91                  self.W_out = train_label.T @ np.linalg.pinv(design_matrix
    )
92
93              if self.verbose:
94                      print('Epoch', epoch + 1, 'of', self.epochs, end = '\
    r')
95          return output[-1]
96
97      # Function to make predictions
98      def predict(self,test_input, test_label):
99
100         self.x = self.update(self.x,test_input[:, np.newaxis],self.y_tf)
101
102         self.y_tf = test_label
103
104         design_matrix = np.squeeze(np.vstack([1,test_input[:, np.newaxis
    ],self.x]))
105
106         output = self.W_out @ design_matrix
107
108         return output
```

Algoritmo para utilizar a rede neural Deep ESN:

```
import numpy as np
#
    ------------------------------------------------------------------------------

# deep esn in numpy
#
    ------------------------------------------------------------------------------

def get_p2_norm(x):
    x = x.conj().T @ x
    w, v = np.linalg.eig(x)
    return w, v, x

def get_matrix_normalized(x, sigma):
    w,_,_ = get_p2_norm(x)
    x = x * sigma/ np.sqrt(np.amax(np.abs(w)))
    return x

class deep_echo_state_np:
    """
    comments here!
    """

    def __init__(self, input_scaling, inter_layer_scaling, epochs, N_u,
    N_y, N_r, N_l, spectral_radius = 0.5, sparsity = 0.2, beta = 0,
    verbose = False, method = 'ridge_regression'):
        # parameters

        self.input_scaling = input_scaling
        self.inter_layer_scaling = inter_layer_scaling
        self.epochs = epochs # epochs
        self.N_u = N_u # input length
        self.N_y = N_y # output length
        self.N_r = N_r # reservoir size
        self.N_l = N_l # number of layers in the deep echo state
        self.spectral_radius = spectral_radius
        self.sparsity = sparsity # make the resevoir matrix sparse
        self.beta = beta # coefficient for the regularization in the
    ridge regression
        self.x = np.zeros((self.N_l, self.N_r))   # the state of the deep
     echo state
        self.alpha = np.random.uniform(size=(self.N_l,1)) # tensor
    containing the leaky parameter for each iteration in the deep echo
    state
        self.verbose = verbose  # show logs during the running of the
    deep echo
```

```
36          self.method = method # method used for training the readout
37
38
39
40
41          ## initiate the reservoir tensor
42
43          self.W_reservoir = np.random.uniform(size=(self.N_l,self.N_r,self
    .N_r))
44          self.W_reservoir[np.random.uniform(size=(self.N_l,self.N_r,self.
    N_r))<self.sparsity]=0
45
46          # Compute the spectral radius and put the resevoirs in accordance
     with spectral radius
47
48          #radius = []
49          #for layer in range(self.N_l):
50          #    temp_reservoir = (1 - self.alpha[layer]) * np.identity(self.
    N_r) + self.alpha[layer] * self.W_reservoir[layer]
51          #    w, _ = np.linalg.eig(temp_reservoir)
52          #    w = np.abs(w)
53          #    radius.append(np.amax(w))
54
55          radius = []
56          for i in range(self.N_l):
57              radius.append(np.max(np.real(np.linalg.eigvals((1-self.alpha[
    i])*np.identity(self.N_r)+self.alpha[i]*self.W_reservoir[i]))))
58
59          #radius = np.amax(radius)
60          self.W_reservoir = self.W_reservoir*self.spectral_radius/np.amax(
    radius)
61
62          ## initiate the W_in tensor
63
64          self.W_in = np.random.uniform(low=-1, high=1, size=(self.N_r,self
    .N_u))
65          self.W_in = get_matrix_normalized(self.W_in,self.input_scaling)
66
67          ## initiate the W_connection tensor
68
69          self.W_connection = np.random.uniform(low=-1, high=1,size=(self.
    N_l - 1,self.N_r,self.N_r))
70          for layer in range(self.N_l-1):
71              self.W_connection[layer] = get_matrix_normalized(self.
    W_connection[layer],self.inter_layer_scaling)
72
73          ## initiate the W_out tensor
```

```
74
75          self.W_out = np.random.uniform(size=(self.N_y, self.N_l * self.
    N_r))
76
77
78
79
80          if self.verbose:
81              print('W_reservoir.shape: ', self.W_reservoir.shape)
82              print('W_in.shape: ', self.W_in.shape)
83              print('W_connection.shape: ', self.W_connection.shape)
84              print('W_out.shape: ', self.W_out.shape)
85
86          pass
87
88
89      # function to train the deep echo state
90
91      def train(self, train_input, train_label):
92
93          """
94          The train_input tensor must have the shape (N_samples, N_u),
    where N_u is the input dimension and N_samples is the number of
    samples
95          The train_label tensor must have the shape (N_samples, N_y),
    where N_y is the output dimension and N_samples is the number of
    samples
96          """
97
98          # initiate the design matrix
99
100         design_matrix = np.zeros((self.x.shape[0] * self.x.shape[1],
    train_input.shape[0]))
101
102         # training process
103
104         ## iteration over epochs
105
106         for epoch in range(self.epochs):
107
108             ## iteration over the train data
109
110             for i in range(train_input.shape[0]):
111
112                 ## iteration over the number of layers
113
114                 for layer in range(self.N_l):
```

```
115
116                 if layer == 0:
117                     self.x[layer] = (1 - self.alpha[layer]) * self.x[
    layer] + self.alpha[layer] * np.tanh(np.dot(self.W_in, train_input[i])
     + np.dot(self.W_reservoir[layer], self.x[layer]))
118                 else:
119                     self.x[layer] = (1 - self.alpha[layer]) * self.x[
    layer] + self.alpha[layer] * np.tanh(np.dot(self.W_connection[layer -
    1], self.x[layer - 1]) + np.dot(self.W_reservoir[layer], self.x[layer
    ]))
120
121             ## get the state of the deep echo state
122
123             X = np.reshape(self.x,[self.x.shape[0] * self.x.shape
    [1]])
124
125             if self.verbose:
126                 pass
127                 #print('Full depp echo state shape:', X.shape)
128
129             ## update the design matrix
130
131             design_matrix[:, i] = X
132
133             if self.verbose:
134                 pass
135                 #print('Design matrix shape:', design_matrix.shape)
136
137         ## compute the output
138
139         output = np.squeeze(self.W_out @ np.mean(design_matrix, axis
    =1))
140
141         if self.method == 'ridge_regression':
142
143             ## update W_out
144             self.W_out = np.transpose(train_label) @ np.transpose(
    design_matrix) @ np.linalg.inv(design_matrix @ np.transpose(
    design_matrix) + self.beta * np.identity(self.x.shape[0] * self.x.
    shape[1]))
145
146         if self.method == 'pseudo_inverse':
147
148             ## update W_out
149             self.W_out = train_label.T @ np.linalg.pinv(design_matrix
    )
150
```

```
151
152             if self.verbose:
153                 print('Epoch', epoch + 1, 'of', self.epochs, end = '\r')
154         return output
155
156
157
158     # function to make predictions
159
160     def predict(self, input_signal):
161
162         ## iteration over the number of layers
163
164         for layer in range(self.N_l):
165             if layer == 0:
166                 self.x[layer] = (1 - self.alpha[layer]) * self.x[layer] +
     self.alpha[layer] * np.tanh(np.dot(self.W_in, input_signal) + np.dot(
    self.W_reservoir[layer], self.x[layer]))
167             else:
168                 self.x[layer] = (1 - self.alpha[layer]) * self.x[layer] +
     self.alpha[layer] * np.tanh(np.dot(self.W_connection[layer - 1], self
    .x[layer - 1]) + np.dot(self.W_reservoir[layer], self.x[layer]))
169
170         ## compute the output signal
171
172         X = np.reshape(self.x,[self.x.shape[0] * self.x.shape[1]])
173
174         output_signal = self.W_out @ X
175
176         ## return output signal
177
178         return output_signal
```

Biblioteca para utilizar a função PCA:

```
import numpy as np
from math import floor

class PCA():
    def __init__(self,n_data=1,n_features=1):
        self.n_data = n_data
        self.n_features = n_features

    def varia_o_simples(self,data,dias_diferena):
        new_data = np.zeros((data.shape[0]-dias_diferena,data.shape[1])
    )
        for j in range(data.shape[1]):
            for i in range(data.shape[0]-dias_diferena):
                new_data[i,j] = data[i+dias_diferena,j]-data[i,j]
        return new_data

    def varia_o(self,data,dias_diferena):
        if (data.shape[0]%dias_diferena)==0:
            new_data = np.zeros((dias_diferena,int((data.shape[0])/
    dias_diferena),data.shape[1]))
            position_before = np.zeros((dias_diferena,int((data.shape
    [0])/dias_diferena),1))
            position_after = np.zeros((dias_diferena,int((data.shape
    [0])/dias_diferena),1))
        else:
            new_data = np.zeros((dias_diferena,floor(data.shape[0]/
    dias_diferena),data.shape[1]))
            position_before = np.zeros((dias_diferena,floor(data.shape
    [0]/dias_diferena),1))
            position_after = np.zeros((dias_diferena,floor(data.shape
    [0]/dias_diferena),1))

        for n in range(new_data.shape[0]):
            #print('n=',n)
            for j in range(data.shape[1]):
                #print('j=',j)
                for i in range(dias_diferena+n,data.shape[0],
    dias_diferena):
                    #print('i=',i)
                    #print('i escrito=',(i-dias_diferena-n)/
    dias_diferena)
                    new_data[n,int((i-dias_diferena-n)/dias_diferena)
    ,j] = data[i,j]-data[i-dias_diferena,j]
                    if j==0:
                        position_before[n,int((i-dias_diferena-n)/
    dias_diferena),0] = i-dias_diferena
```

```
                        position_after[n,int((i-dias_diferena-n)/dias_diferena),0] = i
            #print(new_data[n,int((i-dias_diferena-n)/dias_diferena),3])
            #print(new_data)
        return new_data, position_before, position_after

    def medium(self,datap):
        medias = np.zeros((self.n_features,1))
        adjusted_soma = np.zeros((self.n_features,1))
        media = 0
        soma = 0
        for k in range(self.n_features):
            for n in range(datap.shape[0]):
                media += datap[n,k]
            media = media/datap.shape[0]
            medias[k] = media
            for i in range(datap.shape[0]):
                datap[i,k] -= medias[k]
                soma += datap[i,k]
            adjusted_soma[k] = soma
        return datap,adjusted_soma


    def PCA(self,data, dimensions = 1):
        data_adjusted = np.array(data,copy=True)
        medias = np.zeros((self.n_features,1))
        sum_data = np.zeros(self.n_features)
        for k in range(self.n_features):
            medias[k] = np.mean(data[:,k])
            data_adjusted[:,k] -= medias[k]

        covariancia = np.zeros((self.n_features,self.n_features), dtype=float)
        for i in range(covariancia.shape[0]):
            for j in range(covariancia.shape[1]):
                covariancia[i,j] = np.dot(data_adjusted[:,i], data_adjusted[:,j].T)/(data.shape[0]-1)
        a,b = np.linalg.eig(covariancia)
        eing = np.vstack([a,b])
        eing_sorted = np.argsort(eing,axis=1)
        eing_sorted[1:]=eing_sorted[0]
        a = np.take_along_axis(eing, eing_sorted, axis=1)[0,:]
        b = np.take_along_axis(eing, eing_sorted, axis=1)[1:,:]
        a,b
        b = b[:,-dimensions:] #row feature vector
        #return a,b
```

```
79          #the eigenvectors are now in the rows, with the most significant
    eigenvector at the top
80          #the data items are in each column, with each row holding a
    separate dimension
81          final_data = np.dot(b.T,data_adjusted.T).T
82          #final_data is the final data set, with data items in columns,
    and dimensions along rows.
83          #It will give us the original data solely in terms of the vectors
     we chose
84          return final_data,b
```