UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROJETO DE GRADUAÇÃO

PEDRO BIASUTTI CUPERTINO DE CASTRO

PREDIÇÃO DE SÉRIES TEMPORAIS: UMA ANÁLISE COMPARATIVA ENTRE REDES NEURAIS LSTM E NAR

VITÓRIA – ES
JULHO/2018

PEDRO BIASUTTI CUPERTINO DE CASTRO

PREDIÇÃO DE SÉRIES TEMPORAIS: UMA ANÁLISE COMPARATIVA ENTRE REDES NEURAIS LSTM E NAR

Parte manuscrita do Projeto de Graduação do aluno **Pedro Biasutti Cupertino de Castro**, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Orientador: Dr. Klaus Fabian Coco

Co-orientador: Dr. Patrick Marques Ciarelli

VITÓRIA – ES
JULHO/2018

PEDRO BIASUTTI CUPERTINO DE CASTRO

PREDIÇÃO DE SÉRIES TEMPORAIS: UMA ANÁLISE COMPARATIVA ENTRE REDES NEURAIS LSTM E NAR

Parte manuscrita do Projeto de Graduação do aluno **Pedro Biasutti Cupertino de Castro**, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Aprovada em 12 de julho de 2018.

COMISSÃO EXAMINADORA:

_____

Dr. Klaus Fabian Coco
Universidade Federal do Espírito Santo
Orientador

_____

Dr. Patrick Marques Ciarelli
Universidade Federal do Espírito Santo
Co-orientador

_____

Dr. Jorge Aching Samatelo
Universidade Federal do Espírito Santo
Examinador

_____

MSc. Ana Paula Miranda Diniz
Examinadora

RESUMO

O objetivo deste trabalho é realizar a análise comparativa entre redes neurais baseadas em *deep learning*, em especial a *Long Short Term Memory* (LSTM), e redes neurais clássicas, em especial a rede não linear auto regressiva (*Nonlinear AutoRegressive* - NAR), com o intuito de prever séries temporais. Para o desenvolvimento e análise de desempenho das redes supracitadas, utilizou-se a ferramenta computacional *Matlab*. Adotou-se uma metodologia de desenvolvimento com base em comparação de parâmetros similares das duas redes estudadas. Para avaliação foram utilizadas quatro séries reais, além de uma série adicional resultante de equacionamento matemático que simula uma série levemente caótica. Como conclusão, demonstrou-se que as redes LSTM tendem a apresentar melhor desempenho para horizontes longínquos de predição. Em contrapartida, suas complexidades, relacionadas à série de parâmetros a serem determinados e aprendidos, faz com que seja necessária uma maior capacidade de processamento dos dados, demandando mais recursos computacionais.

Palavras-chave: *deep learning*, LSTM, redes neurais, NAR, séries temporais.

ABSTRACT

This paper presents a comparative analysis between deep leaning-based networks, specifically Long Short-Term Memory (LSTM), and classical neural networks, specifically Nonlinear AutoRegressive (NAR), aimed to time series prediction. For development and analysis of performances of the networks mentioned, a computational environment named Matlab was used. The methodology used in this paper was based on the analysis of similar parameters of both studied networks. Four real time series were considered additionally with one supplementary time series based on mathematical equation that tries to simulate a slightly chaotic series. As a conclusion, this work demonstrated that LSTM networks tend to perform better than NAR for long horizons of predictions. On the other side, its complexity, related to the number of parameters to be found and learned, demands a higher data processing capability that consequently requires additional computational resources.

Keywords: deep learning, LSTM, neural networks, NAR, time series.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ANN	*Artificial Neural Networks* – Redes Neurais Artificiais
MLP	*Multilayer Perceptron* – Rede Multicamadas Perceptron
DNN	*Deep Neural Network*– Redes Neurais Profundas
RMSE	*Root Mean Square Error*
MAPE	*Mean Average Percentual Error*
LSTM	*Long Short-Term Memory*
NAR	*Nonlinear Autoregressive*
GPU	*Graphical Processor Unit*
RAM	*Random Access Memory*
BPTT	*Backpropagation Through Time*
UCI	*University of California Irvine*
USP	Universidade de São Paulo
ICMC	Instituto de Ciências Matemáticas e de Computação

SUMÁRIO

1 APRESENTAÇÃO

1.1 Introdução

As Redes Neurais Artificiais (*Artificial Neural Network* - ANN) são modelos de máquinas de aprendizado inspirados no neurônio biológico. Desde sua origem em 1943, têm sido usadas para resolução de diversos problemas, entre eles reconhecimento de imagem, escrita e caligrafia até mesmo na tradução em tempo real da linguagem de sinais (GAMBOA, 2017).

Atualmente, as técnicas de inteligência artificial têm conseguido grandes avanços em aplicações de engenharia. O *deep learning*, por exemplo, conhecido também pelo termo “aprendizado profundo”, tem apresentado resultados importantes. Apesar de muitos pensarem se tratar de uma nova tecnologia, tem sua origem muito antes, em meados de 1940 (GOODFELLOW; BENGIO; COURVILLE, 2016), todavia, as limitações de *hardware* e de linguagens de programação mais adequadas impossibilitaram seu desenvolvimento até pouco tempo atrás com o desenvolvimento das GPU.

O recente sucesso do *deep learning* pode ser atribuído aos inúmeros avanços nas áreas de visão computacional, processamento de texto e reconhecimento de voz (LE, 2015). Além disso, a área de aprendizado profundo começou a ser mais pesquisada por grandes companhias como Facebook, Google e Microsoft (BATRES-ESTRADA, 2015) tornando ainda maior sua visibilidade. Suas aplicações abrangem uma grande área de conhecimento, incluindo a predição de séries temporais.

Cada vez mais é comum se ver registros de dados na forma de tempo, desde ações de mercado, sinais de áudio a até mesmo monitoramento do sono (GAMBOA, 2017). A este conjunto de dados armazenados em espaços de tempo se dá o nome de séries temporais. Em dados financeiros, como os valores das ações de uma empresa, as séries são muito complexas, visto que depende de uma variedade de parâmetros externos. Não é de se surpreender, portanto, que predição de ações de mercado é considerado um dos mais desafiadores problemas entre predições de séries temporais (BAO; YUE; RAO, 2017).

As redes neurais com termo de memória longa, *Long Short Therm Memory* (LSTM) emergiram como eficientes modelos para diversos problemas de aplicação relacionados a dados sequenciais. As redes LSTM não sofrem com as barreiras de otimização que penalizam as redes recursivas simples além de serem usadas para avançar o estado da arte para muitos problemas difíceis. Podem-se citar problemas de reconhecimento de caligrafia, modelagem de linguagem, tradução, modelagem acústica da fala, predição de estrutura secundária de proteína e análise de áudio e vídeo (GREFF *et al*., 2017).

Aplicações usando inteligência de máquina em séries temporais já vêm sendo desenvolvidas há algum tempo. O uso de predição de séries temporais usando as ANN existe desde 1980 (KUREMOTO *et al*., 2013), no entanto, foram identificados alguns tipos de problemas que as redes clássicas tinham dificuldades em resolver (TELGARSKY, 2015). Com o surgimento da LSTM foi possível encontrar soluções para alguns problemas complexos que as redes clássicas não conseguiam respostas adequadas (HOCHREITER; SCHMIDHUBER, 1997). Porém, uma pesquisa relevante é identificar o quanto que uma rede LSTM pode ser melhor do que uma rede clássica em problemas reais. Assim, este trabalho propõe um estudo, a partir de séries temporais, comparando os resultados de predição de redes neurais clássicas, que possuem arquitetura rasa, com os resultados de redes neurais LSTM. Para isso, baseou-se em metodologias e abordagens de publicações nesta mesma área (BAO; YUE; RAO, 2017; QIU *et al*., 2014; VENGERTSEV, 2014)

1.2 Escopo

O escopo do seguinte trabalho se delimita a implementação e estudo da rede LSTM, assim como da rede NAR (*Nonlinear Autoregressive*). Além disso, serão utilizadas ambas as abordagens para predição de séries temporais para que se possa arguir qual é a rede mais propensa a gerar melhores resultados.

1.3 Justificativa

A escolha do tema para o desenvolvimento do trabalho se baseia em duas principais justificativas. A primeira está relacionada ao aprendizado proporcionado pela elaboração do trabalho, em especial na área de *deep learning*. A segunda é realizar um estudo comparativo mais abrangente entre redes neurais clássicas e redes LSTM na tarefa de predição de valores, tendo em vista que alguns artigos focam somente em parâmetros relacionados aos erros de predição (LI *et al*., 2017; KUREMOTO *et al*., 2013; BAO; YUE; RAO, 2017; QIU *et al*., 2014).

1.4 Objetivos

1.4.1 Objetivo Geral

O objetivo geral deste trabalho é realizar uma análise comparativa entre as redes NAR e LSTM, efetuando a predição de amostras em séries temporais.

1.4.2 Objetivos Específicos

- Pesquisar sobre a rede LSTM para predição (*forecasting*) em séries temporais;
- Uso da rede NAR para predição de séries temporais.

1.5 Organização do Trabalho

Esse trabalho foi divido em 6 capítulos, sendo eles apresentação, embasamento teórico, metodologia e implementação, resultados e análises, conclusões e referências bibliográficas.

No primeiro capítulo são introduzidos os conceitos relacionados a séries temporais, aprendizado de máquina e *deep learnig*. Em adição, faz-se o estudo das redes utilizadas no trabalho, sendo essas a rede LSTM e a rede NAR.

No segundo capitulo são apresentadas as arquiteturas das redes, seus parâmetros de treinamento e suas implementações. Além disso, tem-se a metodologia de comparação utilizada no trabalho e as séries temporais utilizadas.

No terceiro capítulo, são introduzidos os resultados obtidos com as series temporais utilizadas, os parâmetros de desempenho adotados e as análises pertinentes.

O quinto capítulo aborda as conclusões do trabalho juntamente com propostas para trabalhos futuros.

O sexto capítulo é destinado a apresentação de todas referências bibliográficas utilizadas no trabalho.

2 EMBASAMENTO TEÓRICO

2.1 Séries Temporais

De maneira bem intuitiva, pode-se definir como séries temporais um conjunto de observações ordenadas no tempo (MORETTIM; TOLOI, 1981). Como exemplo, a Figura 1 exibe o preço de venda do dólar comercial, na qual apresenta o valor diário do preço de venda do dólar comercial entre o período de julho de 2017 a junho de 2018 (UOL, 2018).

Figura 1 – Preço de Venda Dólar Comercial

Fonte: UOL Economia (2018).

É muito comum representar a série temporal pela soma de três componentes, como apresentado pela Equação (1), onde, Z_t corresponde a série temporal, S_t a componente sazonal da série, T_t a componente de tendência e A_t um ruído.

$$Z_t = S_t + T_t + A_t \tag{1}$$

A componente sazonal é responsável por representar eventos cíclicos, situações que ocorrem sempre em um mesmo intervalo de tempo responsáveis por alterar o comportamento da série. A tendência, por sua vez, modela os valores para qual a série caminha. De maneira geral, ela é influenciada por fatores medidos durante períodos extensos de tempo. Por fim, a componente de ruído simboliza o efeito de eventos que podem alternar de valor, normalmente de maneira aleatória, sem se ter um comportamento preciso (MORETTIM; TOLOI, 1981).

2.2 Aprendizado de Máquina

As dificuldades enfrentadas pelos sistemas baseados em *hard-coded knowledge* (aprendizado por dados direto do código fonte) sugerem que sistemas de inteligência artificial necessitam da habilidade de aprender o próprio conhecimento por meio da extração de padrões nos dados originais. A essa habilidade é atribuído o nome de aprendizado de máquina. A introdução do aprendizado de máquina permitiu que modelos algorítmicos resolvessem problemas envolvendo análise de dados do mundo real e realizar tomadas de ações baseadas em subjetividade (GOODFELLOW *et al*., 2016).

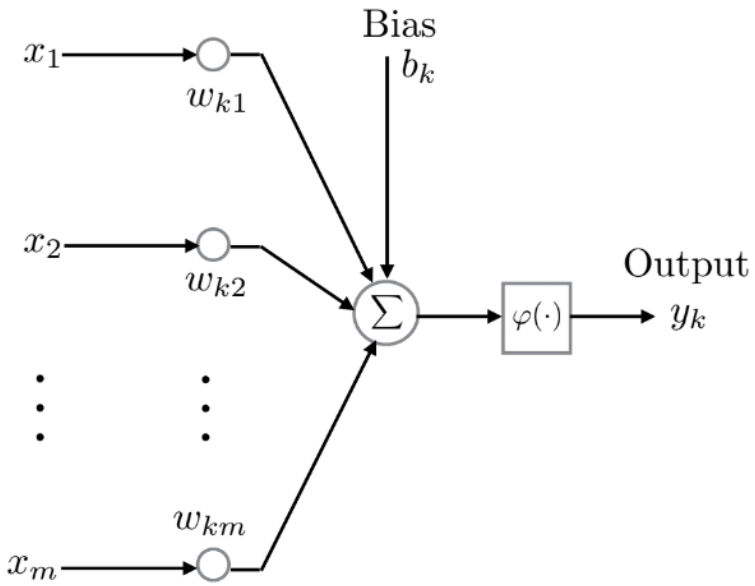
2.3 Redes Neurais Artificiais

A *ANN* é uma máquina desenvolvida para modelar o modo como o cérebro humano realiza certas atividades, ou até mesmo uma função de interesse. A rede usualmente é implementada utilizando componentes eletrônicos ou um *software* de um computador (HAYKIN, 2009).

A ANN é constituída basicamente de nós dispostos em diferentes camadas. A rede é composta por uma série de conexões entre os nós, onde se estipula um peso para cada conexão, seja entre neurônios ou dados de entrada. As entradas do neurônio são colocadas em um somador, onde se terá uma soma ponderada. Além disso, se tem uma função de ativação onde se delimita a amplitude do sinal de saída (SCHMIDHUBER, 2015).

Comumente se insere um *bias* responsável por reduzir/ampliar a entrada da função de ativação. A função de ativação é a responsável por definir o sinal de saída do neurônio dado os dados de entrada. Uma função tipicamente usada é a sigmoide (BATRES-ESTRADA, 2015). A Figura 2 exemplifica o modelo de um neurônio artificial, onde se pode ver as entradas x_m, os pesos w_k, o *bias* b_k, a função de ativação φ e a saída y_k. A letra k representa o número do neurônio enquanto m significa o numero de entradas no neurônio.

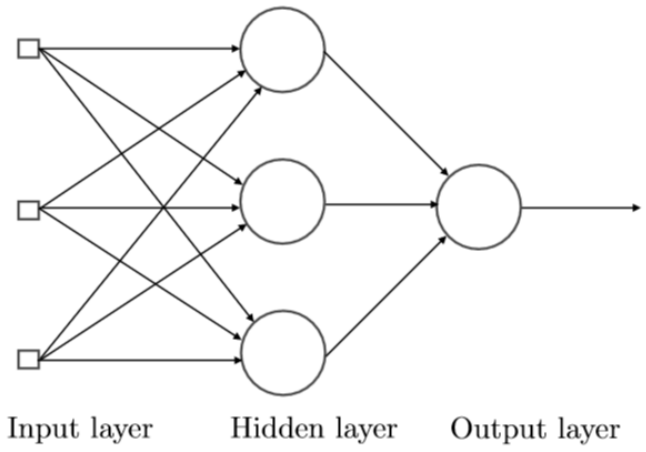
Figura 2 – Modelo de Neurônio



Fonte: Batres-Estrada, pag. 6 (2015).

Um modelo de rede muito usada é a Rede *Perceptron* Multicamadas, onde se tem a camada de entrada de dados na qual os dados são enviados a camada oculta. Nesta camada os dados passam por funções não lineares, e na camada de saída tem-se o resultado da rede, como mostrado na Figura 3 (BATRES-ESTRADA, 2015).

Figura 3 – Rede Perceptron Multicamadas com uma única camada oculta



Fonte: Batres-Estrada, pag. 16 (2015).

É importante destacar que a utilização de apenas uma camada oculta era uma implementação comum, até recentemente em redes neurais artificiais (DALTO, MATUSKO, VASAK, 2014). Contudo, é possível se ter diversas camadas escondidas originando as redes neurais profundas, *Deep Neural Networks* (DNN).

2.4 *Backpropagation Through Time* (BPTT)

O objetivo de se utilizar o método de *backpropagation* no aprendizado da rede é o de modificar os pesos da rede neural de forma a minimizar o erro da saída em relação ao dado esperado (WERBOS, 1990).

O algoritmo de BPTT, por sua vez, trata-se de uma variação do *backpropagation* para redes neurais recursivas aplicada a séries temporais. O BPTT funciona projetando todos as entradas para cada *timestep*. Cada *timestep* uma entrada, uma cópia da rede e uma saída. Dessa forma, calcula-se os erros e os acumula para cada *timestep*. Feito isso, atualiza-se os pesos da rede. Esse processo é feito repetidamente (BROWNLEE, 2017).

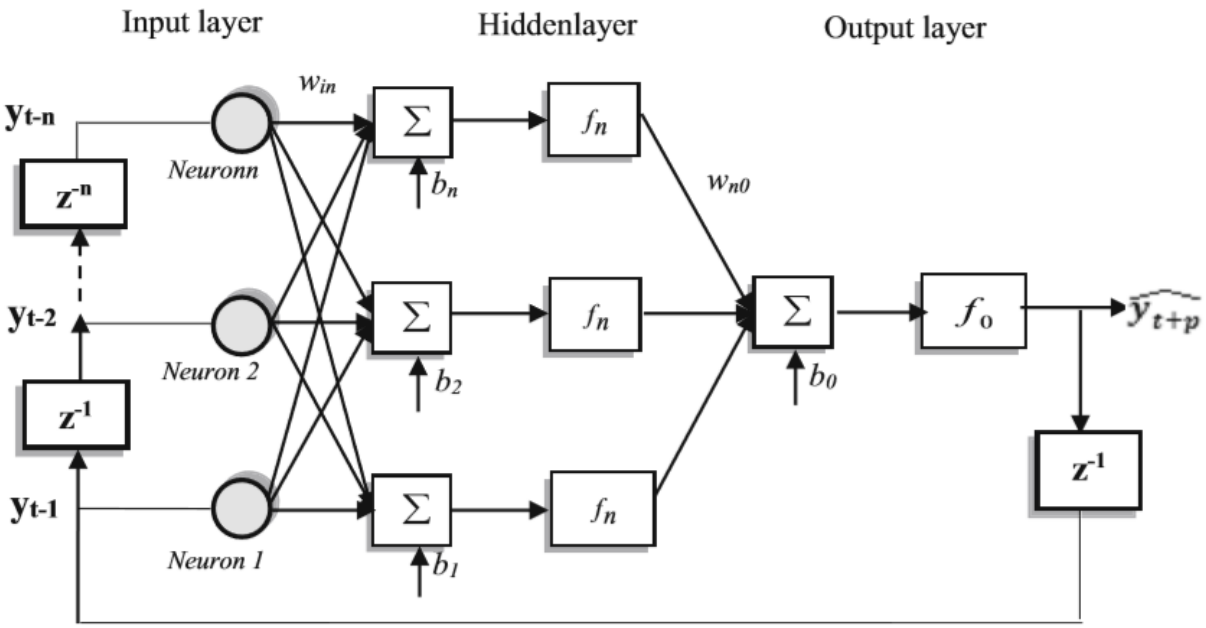
Pode-se destacar como vantagem do BPTT é a possibilidade de, usualmente, utilizar taxas de aprendizado mais baixas do que utilizada no *steepest descente* (WERBOS, 1990). Como contrapartida, o método pode ser custoso computacionalmente a medida que o número de *timesteps* aumenta (BROWNLEE, 2017).

2.5 Rede NAR (*Nonlinear Autoregressive Network*)

É uma rede recorrente do tipo *feedfoward*. As redes NAR são bem similares às redes Perceptron Multicamadas, apresentadas na Figura 3. Ambas possuem uma camada totalmente conectada com neurônios que utilizam uma função de ativação sigmoide nas camadas ocultas e uma função de ativação linear na camada de saída. A principal diferença consiste da utilização de realimentação (*feedback)* tanto entre a saída da rede e a sua entrada, como também entre a saída da rede para com os blocos de atraso (ERWINSKI *et al*., 2016).

A Figura 4 ilustra um exemplo de rede NAR com suas respectivas camadas. Pode-se observar os blocos de atraso, Z^{-1} a Z^{-n}, sendo *n* correspondente ao número de atrasos desejado, juntamente com os pesos, *w*, os *bias*, *b*, a função de ativação, f_n, e os neurônios dentro da camada oculta. Já na camada de saída, visualiza-se os respectivos pesos e os *bias* além da função linear de ativação,f_O.

Figura 4 – Esquemático NAR



Fonte: Benmouiza; Cheknane, pag. 6 (2015).

Ao se implementar uma rede do tipo NAR via *Matlab*, é importante notar que por padrão o programa gera uma rede *open loop feedback output* associado a um *feedback input*, ou seja, a saída da rede é passada para rede como uma entrada, de maneira não recursiva. Desta forma, para que se faça uma predição com um horizonte maior do que um necessita-se fechar a malha através do comando *closeloop*.

2.6 Deep Learning

A *Deep learning*, (aprendizado profundo), é uma área do aprendizado de máquina onde é utilizada a ideia de aprendizado via exemplos (BUDUMA, 2017). Esses métodos são utilizados para o treinamento de redes mais profundas com muitas camadas, onde se tem maior dificuldade para treiná-las utilizando os métodos clássicos.

Os avanços computacionais, tanto em relação ao desempenho e funcionalidades dos *softwares,* como também dos *hardwares*, permitiram o aprimoramento dos algoritmos de aprendizado profundo tornando-os mais eficientes e difundidos (GOODFELLOW; BENGIO; COURVILLE, 2016).

Atualmente, as redes neurais profundas se tornaram o ramo mais importante da inteligência artificial, atingindo sucesso em muitas aplicações (SHAN, GUAN, YANG, XU e ZHANG, 2017). Agregando, por conseguinte, a necessidade de se estudar métodos baseados em aprendizado profundo.

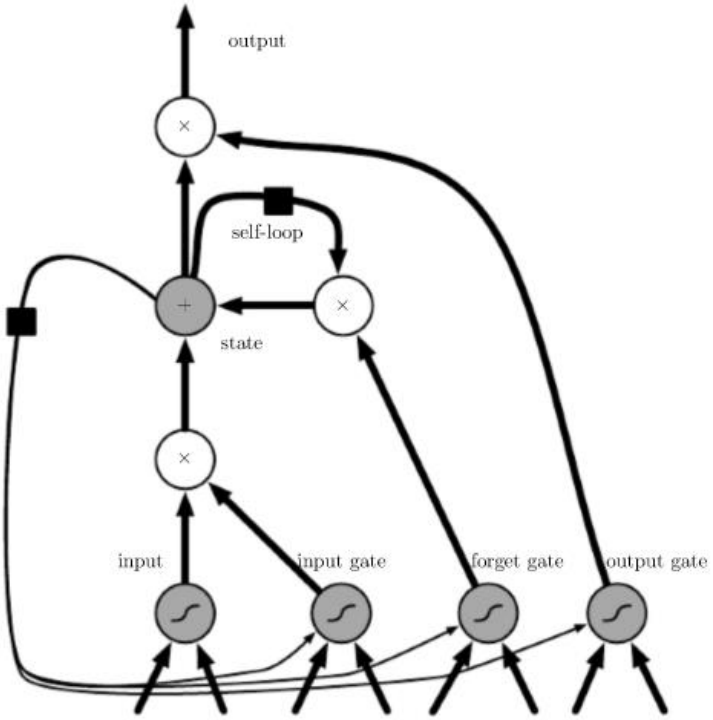
2.7 *Long Short-Term Memory* (LSTM)

A LSTM é uma rede neural recursiva capaz de aprender com intervalos de tempos bem elevados, mesmo em casos com ruído, sem perder a capacidade de aprender com os dados passados em curtos períodos de tempo (HOCHREITER; SCHMIDHUBER, 1997).

Uma vantagem das redes do tipo LSTM é não sofrer com o problema do *vanishing gradient* e o do *exploding gradient* inerentes às redes ANN (SAK; SENIOR; BEAUFAYS, 2014). O problema do *exploding gradient* se refere ao intenso aumento do valor da norma do gradiente durante o período de treinamento, isso ocorre devido aos valores bem elevados dos componentes de termo longo que podem crescer exponencialmente em relação aos de termo curto (PASCANU; MIKOLOV; BENGIO, 2013). O *vanishing gradient*, por sua vez, se refere ao fato dos termos longos diminuírem de maneira exponencial em direção a norma igual a zero, tornando impossível do modelo aprender a correlação temporal entre eventos longínquos (PASCANU; MIKOLOV; BENGIO, 2013).

O grande diferencial das redes LSTM está nas chamadas células de memória, contidas dentro da camada oculta. Cada célula de memória contem três *gates* ajustando e mantendo o estado de cada célula, sendo esses: *input gate*, *forget gate* e *output gate* (PASCANU; MIKOLOV; BENGIO, 2013), como pode-se visualizar na Figura 5.

Figura 5 – Diagrama de bloco de uma célula LSTM



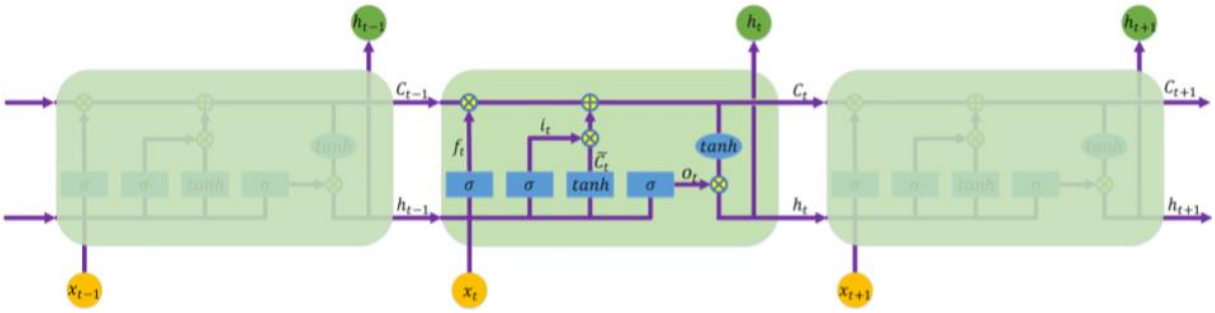
Fonte: Goodfellow; Bengio; Courville, pag. 409 (2016).

As células de memória são capazes de manter informações críticas aprendidas durante os passos da rede, sendo a rede construída para manter de maneira eficiente as informações relevantes dentro dessas células de memória. A cada iteração as células de memória são modificadas de acordo com novas informações (BUDAMA, 2017). A maneira como essa iteração da célula de memória se dá é exclusivamente pela atuação dos *gates*.

Os *gates* são responsáveis por controlar as interações entra as células de memória vizinhas e a memória propriamente dita. O *input gate* controla quando o sinal de entrada pode alterar o estado da célula. O *output gate*, por sua vez, controla quando pode alterar o estado de outra célula de memória. Em adição, o *forget gate* pode controlar se deve ou não esquecer o estado anterior (BAO; YUE; RAO, 2017).

Na Figura 6, tem-se um esquemático de como funcionam as células sequencialmente. O x_t e o h_t são os vetores de entrada e saída da célula de memória para cada instante de tempo t, respectivamente. O i_t representa o valor do *input gate*, já o f_t significa o valor do *forget gate* e o O_t é o valor do *output gate*, todos a cada instante de tempo, t. $\tilde{C}_t$, por sua vez, representa os valores candidatos ao estado da célula de memória.

Figura 6 – Esquemático de uma sequência de células LSTM.



Fonte: Bao; Yue; Rao, pag. 10 (2017).

2.8 Métodos de Otimização

Os princípios básicos para otimização foram descobertos desde o século XVII por cientistas e matemáticos como Kepler, Fermat, Newton e Leibnitz (SHAN *et al*., 2017). Em especial para ANN, os métodos de otimização são utilizados para minimizar a função de custo, de maneira a se chegar a uma resposta mais próxima do possível do real. Utiliza-se como função de custo, em geral, o erro quadrático médio (*Mean Square Error* - MSE).

A área de ANN conta com uma série de algoritmos destinados a otimização, sendo baseados, em sua grande maioria, no método *Stochastic Gradient Descent* (SGD) o método mais popular adotado pelas redes neurais profundas (SHAN et al., 2017).

Para as redes neurais profundas, podem-se citar como exemplos os seguintes métodos: SGDM (S*tochastic Gradient Descent with Mometum)*, Adam (*Adaptive Moment Estimation*), RMSProp (*Root Mean Square Propagation*), AdaGrad (*Adaptive Gradient*), (GOODFELLOW; BENGIO; COURVILLE, 2016). Para o software *Matlab*, encontram-se apenas os três primeiros métodos. O Adam será melhor abordado a seguir, tendo em vista que este foi o utilizado no trabalho.

As redes rasas, por sua vez, como é o caso da rede NAR, tem-se outras funções de otimização, por exemplo: Levenberg-Marquardt, *Bayesian regularization* e *Scaled conjugate*, (BENMOUIZA; CHEKNANE, 2015). O programa *Matlab*, contudo, além dos métodos citados acima, conta com algumas outras possibilidades. O Levenberg-Marquardt será melhor abordado a seguir, visto que também foi utilizado no trabalho.

2.8.1 Método de Otimização Adam

Adam, do inglês *Adaptive Moment Estimation*, é um método de otimização estocástico com taxa adaptativa de aprendizado. De forma mais específica, Adam é um algoritmo de otimização baseado em gradiente de primeira ordem de funções estocásticas utilizando estimativas adaptativas de momentos de baixa ordem (SHAN et al., 2017).

O algoritmo funciona calculando taxas de aprendizado de forma individual para diferentes parâmetros por meio da estimativa da primeira e segunda ordem do momento do gradiente via *Backpropagation*. A atualização dos parâmetros, por sua vez, é feita pela computação da divisão tanto do momento de primeira ordem quanto do de segunda ordem (SHAN et al., 2017).

O método foi desenvolvido com objetivo de combinar o bom desempenho do AdaGrad com gradientes espaçados junto com um bom desempenho em parâmetros não estacionários e *online* do RMSProp (KINGMA; BA, 2015), resultando em uma convergência mais rápida do que a obtida via *Stochastic Gradient Descent* (SGD) (SHAN et al., 2017).

É possível destacar o algoritmo por aspectos como, possuir implementação direta, baixo requisito em termos de memória e adequado a problemas com termos longínquos. Por essas razões, autores acreditam no potencial do método Adam na área de mercado de ações (KINGMA; BA, 2015). Na Figura 7, tem-se o algoritmo Adam, onde se explica passo a passo cada operação realizada neste método.

Figura 7 – Algoritmo Adam

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{0}$
  Initialize time step $t = 0$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1) \boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2) \boldsymbol{g} \odot \boldsymbol{g}$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
    Compute update: $\Delta \boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
  end while

Fonte: Goodfellow; Bengio; Courville, pag. 311 (2016).

2.8.2 Método de Otimização Levenberg-Marquardt

O algoritmo de Levenberg-Maquart (LM) é uma técnica iterativa capaz de localizar o mínimo de uma função multivariável expressa como a soma dos quadrados de uma função real não linear (LEVENBERG, 1944; MARQUARDT 1963). O LM é uma combinação do método de *steepest descent* com o método de Gauss-Newton que propicia uma variação na taxa de aprendizado (HONGWEI, 2010).

Desta forma, quando a solução está distante do valor correto, o algoritmo se comporta como o método de *steepest descent*, progredindo lentamente, mas garantindo a convergência. Quando a solução se aproxima do resultado correto, porém, o algoritmo se comporta como o método de Gauss-Newton (HONGWEI, 2010).

Sendo assim, LM é um dos métodos de otimização não linear mais adequado para desempenho baseado no critério de mínimos quadrados (THEVENAZ; RUTTIMANN; UNSER, 1998). O algoritmo é muito eficiente em treinamento de redes neurais com até centenas de pesos. Contudo, isto reflete em um esforço computacional mais elevado para cada iteração (HAGAN; MENHAJ, 1994). Na Figura 8, tem-se o algoritmo Levenberg-Marquardt, onde se explica passo a passo cada operação realizada neste método.

Figura 8 – Algoritmo Levenberg-Marquardt

<u>Given</u> A vector of measurements $\mathbf{X}$ with covariance matrix $\Sigma_{\mathbf{X}}$, an initial estimate of a set of parameters $\mathbf{P} = (\mathbf{a}^{\mathsf{T}}, \mathbf{b}^{\mathsf{T}})^{\mathsf{T}}$ and a function $f : \mathbf{P} \mapsto \widehat{\mathbf{X}}$ taking the parameter vector to an estimate of the measurement vector.

<u>Objective</u> Find the set of parameters $\mathbf{P}$ that minimizes $\epsilon^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\epsilon$ where $\epsilon = \mathbf{X} - \widehat{\mathbf{X}}$.

<u>Algorithm</u>

(i) Initialize a constant $\lambda = 0.001$ (typical value).
(ii) Compute the derivative matrices $\mathtt{A} = [\partial\widehat{\mathbf{X}}/\partial\mathbf{a}]$ and $\mathtt{B} = [\partial\widehat{\mathbf{X}}/\partial\mathbf{b}]$ and the error vector ϵ.
(iii) Compute intermediate expressions

$$\mathtt{U} = \mathtt{A}^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\mathtt{A} \qquad \mathtt{V} = \mathtt{B}^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\mathtt{B} \qquad \mathtt{W} = \mathtt{A}^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\mathtt{B}$$

$$\epsilon_{\mathtt{A}} = \mathtt{A}^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\epsilon \qquad \epsilon_{\mathtt{B}} = \mathtt{B}^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\epsilon$$

(iv) Augment $\mathtt{U}$ and $\mathtt{V}$ by multiplying their diagonal elements by $1 + \lambda$.
(v) Compute the inverse $\mathtt{V}^{*-1}$, and define $\mathtt{Y} = \mathtt{W}\mathtt{V}^{*-1}$. The inverse may overwrite the value of $\mathtt{V}^{*}$ which will not be needed again.
(vi) Find $\boldsymbol{\delta}_{\mathbf{a}}$ by solving $(\mathtt{U}^{*} - \mathtt{Y}\mathtt{W}^{\mathsf{T}})\boldsymbol{\delta}_{\mathbf{a}} = \epsilon_{\mathtt{A}} - \mathtt{Y}\epsilon_{\mathtt{B}}$.
(vii) Find $\boldsymbol{\delta}_{\mathbf{b}}$ by back-substitution: $\boldsymbol{\delta}_{\mathbf{b}} = \mathtt{V}^{*-1}(\epsilon_{\mathtt{B}} - \mathtt{W}^{\mathsf{T}}\boldsymbol{\delta}_{\mathbf{a}})$.
(viii) Update the parameter vector by adding the incremental vector $(\boldsymbol{\delta}_{\mathbf{a}}^{\mathsf{T}}, \boldsymbol{\delta}_{\mathbf{b}}^{\mathsf{T}})^{\mathsf{T}}$ and compute the new error vector.
(ix) If the new error is less than the old error, then accept the new values of the parameters, diminish the value of λ by a factor of 10, and start again at step (ii), or else terminate.
(x) If the new error is greater than the old error, then revert to the old parameter values, increase the value of λ by a factor of 10, and try again from step (iv).

Fonte: Hartley; Zisserman, pag. 605 (2004).

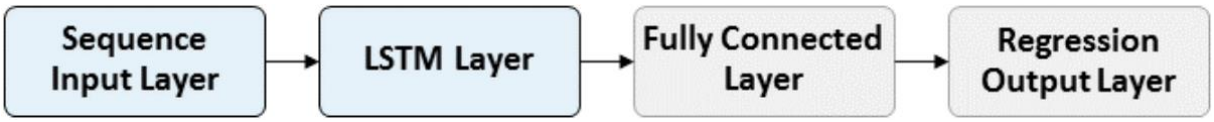
3 METODOLOGIA E IMPLEMENTAÇÃO

Nesta seção serão expostas as arquiteturas das redes abordadas: a rede LSTM e a rede NAR, respectivamente. Serão elucidadas às camadas de cada rede, assim como os parâmetros de treinamento. Serão expostas também as implementações das redes utilizadas, juntamente com os detalhes da variação de seus parâmetros. Além disso, serão introduzidas as séries temporais utilizadas, suas origens e características, assim como os resultados obtidos e as discussões pertinentes.

3.1 Desenvolvimento da Rede Utilizando LSTM

O desenvolvimento da rede utilizando LSTM segui a metodologia proposta pelo *Matlab* para área de regressão, um esquemático da arquitetura de rede LSTM para regressão é ilustrada na Figura 9.

Figura 9 – *Matlab Regression LSTM Network*



Fonte: Beale; Hagan; Demuth, pag. 116 (2018).

3.1.1 Arquitetura da Rede Utilizando LSTM

A rede utilizada para predição de séries temporais via LSTM e *deep learning* consiste na utilização de 4 camadas, sendo essas: camada de entrada, camada de LSTM, camada totalmente conectada e camada de regressão.

A primeira camada é a de entrada (*input layer*), onde se define o número de variáveis de entrada. Para os casos a serem apresentados, se definirá uma entrada apenas. Os dados serão passados pela camada de entrada de maneira sequencial.

A segunda é a de LSTM, onde os dados serão enviados as células de memória LSTM passando por toda uma matemática embutida dentro dos *gates* de entrada, saída e esquecimento, abordados na Seção 2.7, para ter então as saídas das células. Essas saídas são utilizadas como entrada para a camada totalmente conectada.

A terceira é a camada totalmente conectada, conhecida também como *dense layer*, em português camada densa. Tem como finalidade utilizar as saídas da camada de LSTM e tratá-las de forma a gerar uma saída em um formato conveniente para a previsão (COOK; HALL, 2017). Como desenvolveu-se a rede para realizar a predição de uma série temporal por vez, a saída da camada totalmente conectada deve ser um vetor unidimensional.

A quarta camada é a de regressão. Sua finalidade é gerar uma função, via regressão, de forma a predizer de maneira mais correta os próximos pontos da série. Para que se possa calcular os melhores parâmetros para essa função de predição, utiliza-se a função de custo, baseado no erro obtido. No caso em questão, será utilizado como função de custo para regressão o MSE.

3.1.2 Hiperparâmetros de Treinamento dos algoritmos de otimização do *Matlab*

A seguir serão abordados os principais parâmetros fornecidos pelo *Matlab* utilizados para o treinamento da rede. O programa proporciona a utilização de três métodos de otimização, anteriormente citados, sendo esses: SGM, Adam e RMSProp. Optou-se por utilizar o método Adam pelas vantagens citadas na Seção 2.8.1 juntamente com o fato de apresentar melhores resultados nos testes preliminares.

O parâmetro designado para delimitar o número de épocas usadas para o treinamento da rede é o *MaxEpochs*. Uma época pode ser definida como a passagem por completo do algoritmo de treinamento por todo dado de treino.

O *MiniBatchSize* é responsável por definir o tamanho do *mini batch* durante o treinamento. O *mini batch* é uma parcela dos dados de treinamento usada para avaliar o gradiente da função de custo e atualizar os pesos.

A taxa inicial de aprendizado é um parâmetro que estabelece uma relação numérica ao esforço da rede em aprender com os dados. Uma taxa alta pode resultar em um bom resultado, mas também pode rapidamente divergir e não encontrar uma resposta. Já uma taxa baixa tende a encontrar o ponto de operação ótimo, mesmo que necessite de um número significativamente maior de iterações.

É importante destacar que existem outros parâmetros relacionados ao treinamento da série, contudo, adotou-se os valores de configuração previamente estabelecidos pelo *Matlab*.

3.2 Desenvolvimento da Rede NAR

3.2.1 Arquitetura da Rede NAR

A arquitetura da rede NAR, abordada na Seção 2.6, utiliza três camadas: camada de entrada, camada oculta e camada de saída.

A camada de entrada, assim como explicado para rede utilizando LSTM, serve para realizar a entrada sequencial da série além de delimitar o número de dados de entrada a ser utilizado pela rede. Além disso, estão contidos os operadores de atraso referentes à quantidade de dados de atraso que serão utilizados pela rede.

Na camada oculta, como pode ser observado na Figura 4 estão contidos os nós, referentes aos neurônios ocultos da rede, assim como suas respectivas funções de ativação. É nesta camada que é feita a parte principal de processamento dos dados.

A camada de saída, por sua vez, é a responsável pela predição dos próximos dados por meio das saídas fornecidas pela camada escondida. Estas são calculadas em função dos pesos, *bias* e funções de ativação.

3.2.2 Hiperparâmetros de Treinamento dos algoritmos de otimização do *Matlab*

A seguir serão abordados os principais parâmetros utilizados para o treinamento da rede fornecidos pelo programa.

O parâmetro *Feedbackdelays* define a quantidade de atrasos *(delays)* da rede, ou seja, a quantidade de dados passados que serão utilizados pela rede para se realizar a predição. Cada série apresenta uma relação distinta com os dados passados, portanto, cada série tende a ter seu valor ótimo de atraso.

Já o parâmetro *Hiddensizes* tem como finalidade definir o número de neurônios a serem usados na camada oculta. Quanto maior o número de neurônios mais complexa a rede se torna e maior é o custo computacional para treiná-la.

Por fim, o parâmetro *TrainFcn*, define a função de treinamento utilizada pelo programa. O *Matlab* proporciona a utilização de quatro métodos, sendo esses: Levenberg-Marquardt, *Bayesian regularization*, *Scaled conjugate e Resilient backpropagation*. Optou-se por utilizar o método Levenberg-Marquardt pelas vantagens citadas na Seção 2.8.2.

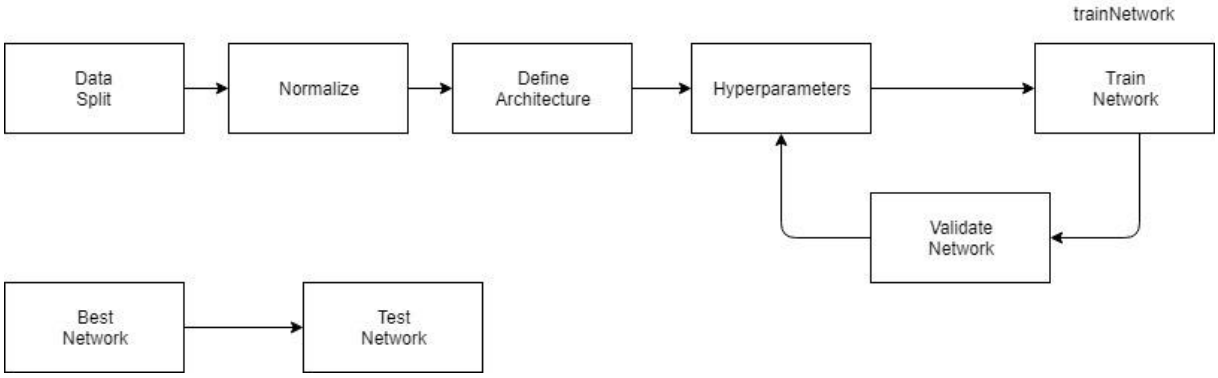
3.3 Implementação das Redes Utilizadas

Ambas as arquiteturas foram desenvolvidas utilizando o *Matlab R2018a*® que possui recursos tanto para implementação de redes clássicas, em especial redes NAR como para redes utilizando *deep learning*, com enfoque em predição de séries temporais.

3.3.1 Implementação da Rede Utilizando LSTM

A Figura 10 representa um esquemático de como foi realizada a implementação da rede com LSTM no *Matlab*, de forma que cada bloco será abordado a posteriori.

Figura 10 – Implementação da Rede Utilizando LSTM



Fonte: Autoria própria.

Após importados os dados das séries para o *Matlab*, deve-se separar os dados de acordo com sua finalidade: dados de treinamento, dado para validação e dados para teste. Adotou-se 70% dos dados para treinamento da rede, 15% para validação e os demais 15% destinados ao teste.

Em seguida, faz-se o tratamento dos dados com intuito de normaliza-los. Portanto, os dados são subtraídos de seu valor médio, para que o resultado desta operação seja dividido pelo valor do desvio padrão. Desta forma, os dados passam a ter média zero e desvio padrão unitário. A função *mean* extrai o valor médio dos pontos. A função *std*, standard *deviation*, significa desvio padrão. A Figura 11, evidencia o processo de normalização. É importante destacar, todavia, que a média e o desvio padrão são calculados com os dados de treinamento e que a normalização deve ser feita também para os dados de validação e de teste.

Figura 11 – Normalização dos Dados

```
mu = mean(XTrain);
sig = std(XTrain);

XTrain = (XTrain - mu) / sig;
YTrain = (YTrain - mu) / sig;
XValidation=(XValidation-mu)/sig;
XTest = (XTest - mu) / sig;
```

Fonte: Autoria própria.

Com os dados previamente normalizados, pode-se prosseguir para definição da estrutura da rede. Como dito anteriormente, a rede LSTM conta com quatro camadas: camada de entrada, camada de LSTM, camada totalmente conectada e camada de regressão.

Definido os parâmetros da rede, cria-se cada camada utilizando sua respectiva função e parâmetros. Agrupam-se as camadas de maneira sequencial para que possam ser treinadas, na parte destinada ao treinamento da rede, de forma correta.

O processo de definição da arquitetura é evidenciado na Figura 12, em que *inputsize* representa o número de dados de entrada na camada de entrada, *numHiddenUnits,* o número de células de memória LSTM e *numResponsesI,* o número de dados de saída da rede.

Figura 12 – Definindo Arquitetura da Rede

```
inputSize = 1;
numHiddenUnits = 175;
numResponses = 1;


layers = [ ...
    sequenceInputLayer(inputSize)
    lstmLayer(numHiddenUnits)
    fullyConnectedLayer(numResponses)
    regressionLayer];
```

Fonte: Autoria própria.

Outro passo importante é a definição dos hiperparâmetros. Nas opções de treinamento da rede tem-se o tipo de otimização a ser usado, os parâmetros referentes ao aprendizado da rede, a forma como os dados serão enviados, além do tipo de hardware a ser utilizado. Os parâmetros mais relevantes para execução do trabalho foram anteriormente detalhados. Esses hiperparâmetros são de extrema relevância para o desempenho da rede. Na Figura 13 pode-se observar os parâmetros de treinamento utilizados. Vale destacar que outras configurações foram utilizadas para o treinamento das redes, como pode-se verifica na Seção 3.4.

Figura 13 – Hiperparâmetros

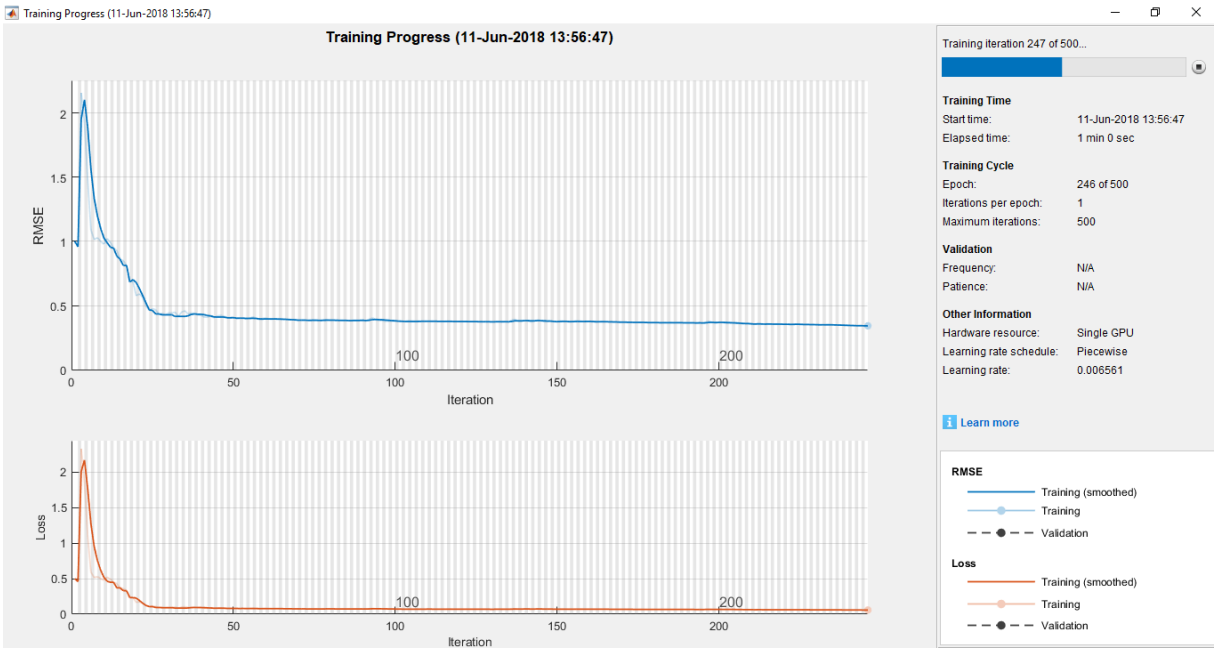
```
opts = trainingOptions('adam', ...
    'MaxEpochs',500, ...
    'GradientThreshold',1, ...
    'InitialLearnRate',0.01, ...
    'LearnRateSchedule','piecewise', ...
    'LearnRateDropPeriod',50, ...
    'LearnRateDropFactor',0.9, ...
    'Verbose',0, ...
    'MiniBatchSize', 64,...
    'Plots','training-progress');
```

Fonte: Autoria própria.

Inicializadas todas as variáveis requisitadas pelo sistema, são passadas como parâmetro para função de treinamento da rede, *TrainNetwork,* os dados de entrada de treinamento, os dados de saída de treinamento, as camadas e as opções de treinamento.

O *software* conta com uma interface de treinamento em que o desenvolvedor visualiza dinamicamente o aprendizado da rede à medida que as épocas são passadas. A interface apresenta um gráfico iterativo da função de custo, como pode ser observado na Figura 14.

Figura 14 – Interface de Treinamento da Rede com LSTM



Fonte: Autoria própria.

Treinada a rede, esta será validada para que se destine para a parte de predição dos valores da série. A parte de validação da rede será explicada na Seção 3.4.

Com 15% dos dados para teste será possível avaliar o desempenho da rede em aprender o comportamento dos dados avaliando para isso o resultado de suas predições.

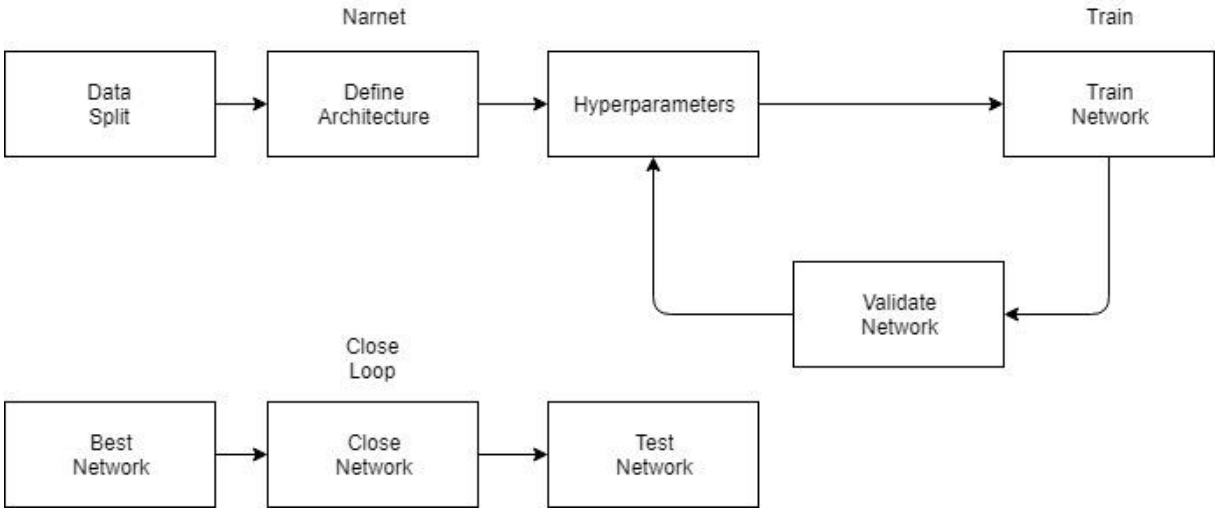
A função utilizada para predição foi a *predictAndUpdateState*, essa função foi especialmente desenvolvida para predição usando redes neurais recursivas. A importância de se atualizar a rede está no fato de redes recursivas utilizarem a saída da célula anterior para executar suas ações, para então levar sua resposta para a próxima célula. No caso do LSTM a entrada passará por diversos *gates* em que operações matemáticas são efetuadas para ao final a célula resultar em uma saída.

Após a realização da predição do correspondente a um horizonte de previsão igual a 15% das amostras correspondente a parte de teste, os dados armazenados em um vetor, *Ypred*, deverão ser novamente tratados. Como inicialmente foram normalizados, necessitam, portanto, passar pelo processo inverso para obter-se o valor genuíno.

3.3.2 Implementação da Rede NAR

A Figura 15 representa um esquemático de como foi realizada a implementação da rede NAR no *Matlab*, de forma que cada bloco será abordado a posteriori.

Figura 15 – Implementação da Rede NAR



Fonte: Autoria própria.

Para os devidos fins de comparação buscou-se a implementação da rede clássica NAR de maneira similar à implementação da rede LSTM, para tornar a comparação fidedigna. Para tanto, repartiu-se os dados em dados de treinamento, dados de validação e dados de teste, de modo análogo ao feito para rede LSTM, 70% dos dados para treino, 15% para validação e 15% para teste.

Primeiro, iniciou-se pelo processo de variação de parâmetros conforme será detalhado na Seção 3.4. A Figura 16 ilustra os parâmetros de treinamento utilizados na rede NAR.

Figura 16 – Definindo rede NAR

```
% ajusta parâmetros da NET
net.TrainParam.epochs = Epocas;
net.divideFcn = 'divideblock';
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio =15/100;
net.trainParam.max_fail = 0.30*Epocas;
net.trainParam.goal = 0;
net.trainParam.min_grad = 0;
net.trainParam.lr = LearningRate;
net.divideMode = 'time';  % Divide up every sample
net.performFcn = 'mse';  % Mean Squared Error
% treina a NET em malha aberta
```

Fonte: Autoria própria.

Uma vez definidas as configurações da rede, prosseguiu-se através da definição de uma rede NAR através do comando *narnet*. Uma vez definida a rede e seus parâmetros, iniciou-se o processo de treino onde foi utilizada a função *train*, função destinada principalmente para as redes clássicas.

Treinada a rede, está deve ser validada para que se prossiga para a predição dos valores da série. A parte de validação da rede será explicada na Seção 3.4. A implementação da predição é realizada conforme segue na Figura 17 onde temos a utilização do *Closed Loop*, explicado na Seção 2.5.

Figura 17 – Closed Loop NAR

```
[y1,xfo,afo] = net(x,xi,ai);
[netc,xic,aic] = closeloop(net,xfo,afo);
[y2,~,~] = netc(cell(0,HP),xic,aic);
MP = [M(:,1:net.numInputDelays) cell2mat(y1) cell2mat(y2)];
```

Fonte: Autoria própria.

3.4 Variação de Parâmetros

Feito o treinamento da rede com a parte dos dados de treinamento, os dados de validação foram usados para avaliar o modelo. Na validação, estabeleceu-se uma metodologia de variação dos parâmetros de treinamento da rede, com maior impacto no desempenho da rede, de forma a encontrar a rede com menor *Root Mean Square Error* (RMSE), dentre as possibilidades geradas. A configuração da rede com o menor RMSE sobre os dados de validação foi usada nos dados de teste.

Para a rede utilizando LSTM, escolheu-se variar os seguintes parâmetros: número de unidades ocultas, número de épocas, taxa inicial de aprendizado e tamanho de *mini-batch*. Devido ao custo computacional, foram escolhidos 4 valores de unidades ocultas, 4 valores de épocas, 4 valores de taxas iniciais de aprendizado e 4 valores de *mini-batches*, gerando um total de 256 combinações.

Idealmente, quanto maior a faixa de valores, mais perto se chegaria do resultado ótimo. Contudo, devido ao custo computacional gerado pelo treinamento, adotou-se as faixas de valores descritas na Tabela 1.

Tabela 1 – Parâmetros LSTM

Parâmetros	**Faixa de valores**
Unidades Ocultas	[100, 125, 150, 175]
Épocas	[50, 100, 150, 200]
Taxa inicial de aprendizado	[0,01, 0,05, 0,001, 0,005]
Mini-batch	[40, 60, 80, 100]

Fonte: Autoria própria.

Na rede NAR, procedeu-se de forma semelhante, sendo que os parâmetros a serem variados foram os seguintes: unidades ocultas, atrasos (*delay*), épocas e taxas de aprendizado. Foram escolhidos 4 valores de unidades ocultas, 4 valores de épocas, 4 valores de taxa de aprendizado e 4 valores de *delays*, gerando um total de 256 combinações. Esses valores são mostrados na Tabela 2.

Tabela 2 – Parâmetros NAR

Parâmetros	**Faixa de valores**
Unidades Ocultas	[10, 20, 30, 40]
Épocas	[50, 100, 150, 200]
Taxa de aprendizado	[0,01, 0,05, 0,001, 0,005]
Delays	[10, 20, 30, 40]

Fonte: Autoria própria.

3.5 Análise de Desempenho

3.5.1 Métricas de Avaliação de Desempenho

Com intuito de realizar comparações técnicas para se observar o desempenho das redes neurais utilizadas, foram adotadas as 4 medidas de desempenho a seguir: RMSE, *Mean Avarage Percentage Error* (MAPE), Tempo de Processamento (TP) e número de unidades ocultas.

Matematicamente, o RMSE é definido de acordo com a fórmula apresentada na Equação (2), onde $Previsto$ corresponde ao vetor com os valores dos dados previstos, $Real$ ao vetor de dados reais, para faixa de valores previstos, e $NumeroTotaldeDados$ corresponde ao total de dados analisados (LI *et al*., 2017).

$$\mathrm{RMSE} = \sqrt{\frac{\sum(Previsto - Real)^2}{NúmeroTotaldeDados}} \tag{2}$$

A popularidade do RMSE se deve ao fato de ser, de maneira geral, o melhor método de aproximação do desvio padrão com dados finitos, sendo este o método mais popular em termos de erro. Uma desvantagem do RMSE é não ser invariante para com as unidades usadas pelas séries. Por exemplo, o uso de metros por segundos e metros por minutos para a velocidade resultaram em diferentes valores de RMSE dado os mesmos erros de posição e velocidade (LI; ZHAO, 2001).

O erro absoluto médio percentual (MAPE) segue uma linha similar à implementação do RMSE. A diferença reside em dois fatores. O primeiro relacionado com o modo como a magnitude do erro é obtida. Enquanto o MAPE faz uso da função valor absoluto, que retira o módulo do número, o RMSE o faz elevando o erro ao quadrado. O segundo consiste em avaliar o erro percentualmente.

Matematicamente, o MAPE é definido de acordo com a fórmula apresentada na Equação (3). Novamente $Previsto$ corresponde ao vetor com os valores dos dados previstos, $Real$ ao vetor de dados reais, para faixa de valores previstos, e $NumeroTotaldeDados$ corresponde ao total de dados analisados (BAO; YUE; RAO, 2017).

$$\mathrm{MAPE} = \frac{100}{NúmeroTotaldeDados} \sum \left| \frac{Previsto - Real}{Real} \right| \tag{3}$$

O MAPE tem como vantagem ser o método mais natural de aproximação de desvio padrão, dado um número finito de amostras (LI; ZHAO, 2001). Contudo, percebe-se através de uma análise matemática que para valores da série perto de zero o valor de MAPE tende a valores elevados. Dessa forma, quanto maior o erro de predição para valores perto de zero, maior será a penalização do MAPE.

O tempo de processamento foi avaliado utilizando-se a ferramenta do *Matlab*, *Run and Time*. O comando executa o *script* cronometrando seu período de duração. Além de indicar o tempo de execução, em segundos, esta ferramenta também fornece o tempo gasto em cada função do código, possibilitando um melhor entendimento do código processado.

Como as comparações foram realizadas no computador pessoal, faz-se necessário informar as configurações do dispositivo. Trata-se de um *notebook* com processador *Intel® Core™* i7-4710HQ, 8 *Gigabytes* de RAM e GPU *NVIDIA® GeForce® GTX™* 850M.

O número de unidades ocultas é um parâmetro importante para avaliar a complexidade da rede. Quanto maior o número de unidades ocultas, mais parâmetros necessitam ser calculados. É importante ressaltar a diferença deste parâmetro para as distintas redes abordadas neste trabalho. Enquanto para rede NAR o número de unidades ocultas está relacionado a quantidade de neurônios nas camadas ocultas, para a rede utilizando LSTM esse número significa o número de células de memória da camada de LSTM.

3.6 Séries Temporais

Como forma de avaliar o desempenho das redes, utilizou-se um total de 5 séries temporais, sendo essas: Mackey-Glass *Chaotic Series*, *SunSpots*, *Mean Daily Temperature in Fisher River*, Aluguel de bicicletas em Porto, *Chaotic Red Laser*.

A Tabela 3 apresenta uma síntese das séries utilizadas, onde são evidenciados o número de amostras da série assim como seu repositório de origem.

Tabela 3 – Resumo das Séries

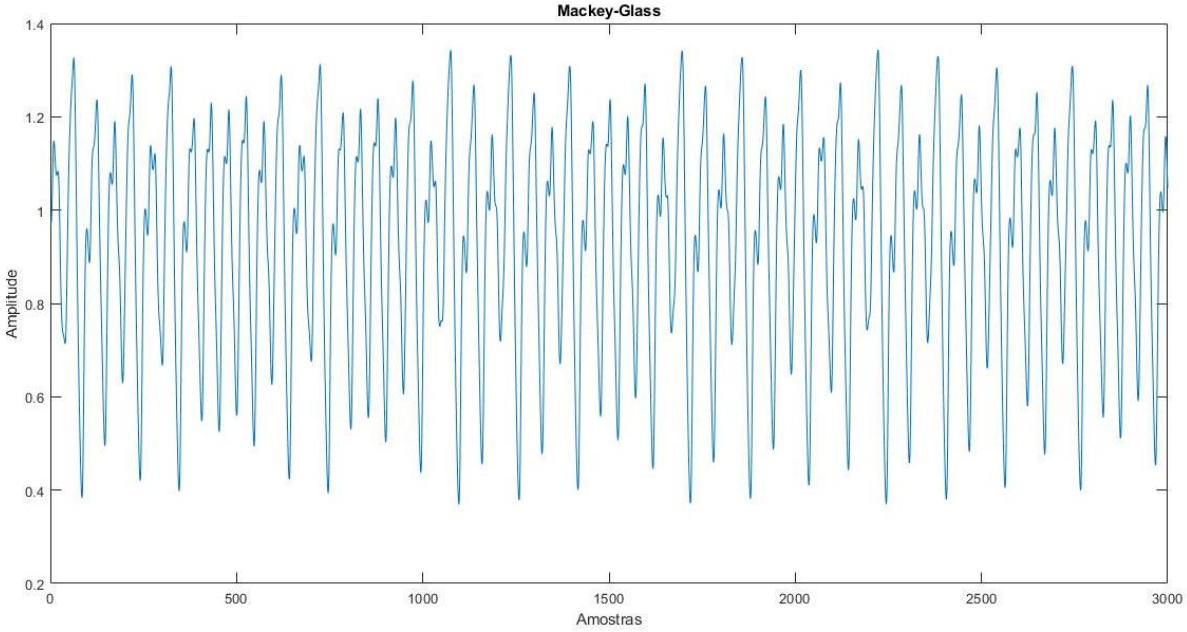
Série	Número de Amostras	Repositório
Mackey-Glass *Chaotic Series*	3000	ICMC-USP
Sunspots	2899	*Matlab*
Mean Daily Temperature in Fisher River	1461	*Datamarket*
Aluguel de Bicicletas na Cidade do Porto	7000	UCI
Chaotic Red Laser	10093	*Matlab*

Fonte: Autoria própria.

3.6.1 Mackey-Glass *Chaotic Series*

A série Mackey-Glass utiliza um equacionamento matemático de forma a gerar uma série levemente caótica. Esta série possui 3000 pontos e foi obtida via ICMC-USP *Time Series Prediction Repository*, repositório do Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC, 2014). A Figura 18 ilustra a série.

Figura 18 – Mackey-Glass *Chaotic Serie*

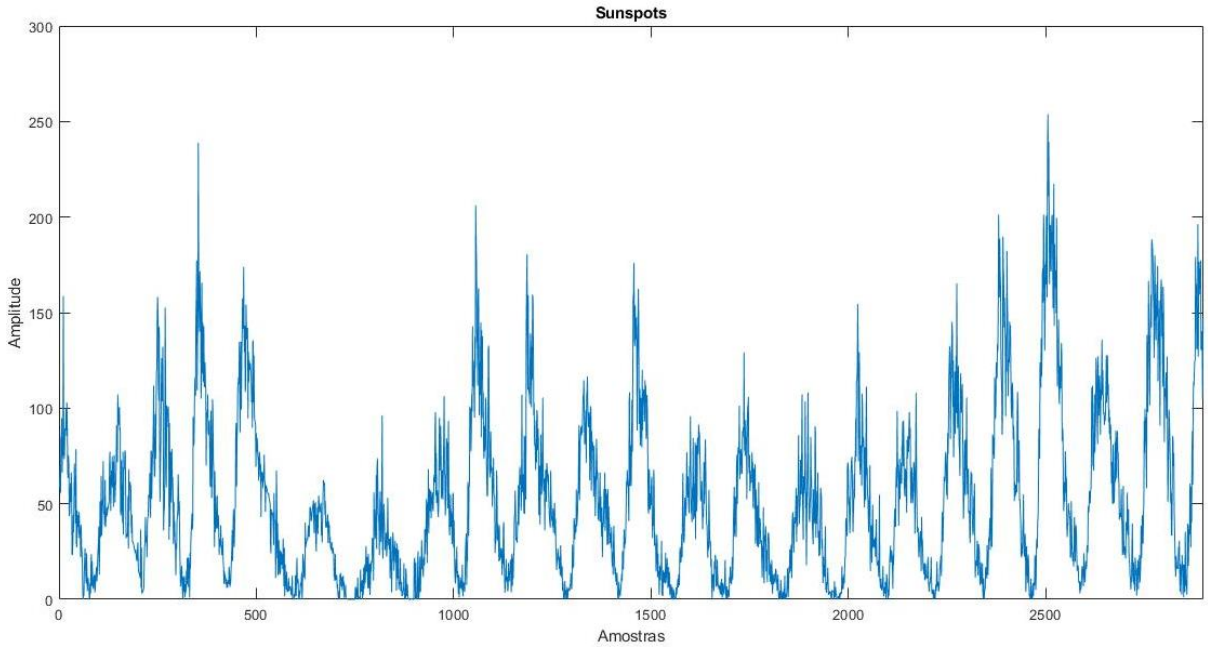


Fonte: Adaptado pelo autor.

3.6.2 Sunspots

A série *Sunspots* são as manchas solares que são ocasionadas por um intenso campo magnético criado no interior do Sol (Chattopadhyay, G; Chattopadhyay, S., 2012). A série foi obtida por meio do *Matlab* e contém a média mensal de *sunspots* em um período de 240 anos, de janeiro de 1749 a julho de 1990, gerando um total de 2899 dados, como pode-se observar na Figura 19.

Figura 19 – *Sunspots*

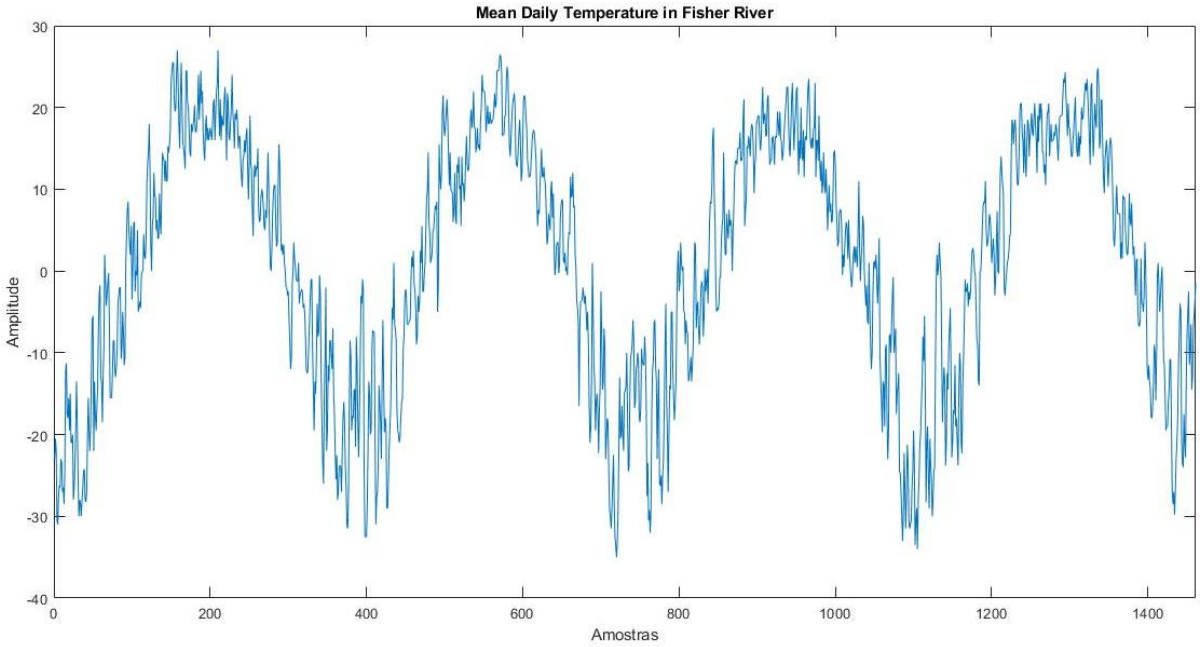


Fonte: Adaptado pelo autor.

3.6.3 Mean Daily Temperature in Fisher River

Como o nome sugere, a série fornece a temperatura média diária no condado de Fisher River, Texas, Estados Unidos. Os dados fornecidos pelo DataMarket (TSDL, 1994) foram colhidos desde o dia primeiro de janeiro de 1988 até 31 de dezembro de 1991 totalizando de 1461 pontos, como pode ser visualizado na Figura 20.

Figura 20 – Mean Daily Temperature in Fisher River



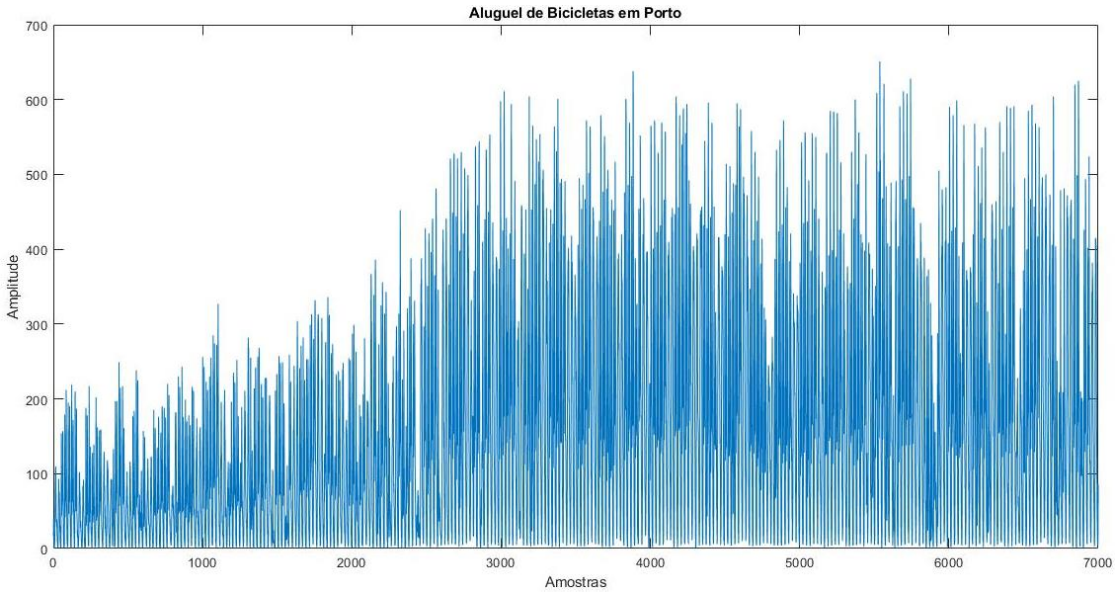
Fonte: Adaptado pelo autor.

3.6.4 Aluguel de Bicicletas na Cidade do Porto

O processo de compartilhamento de bicicletas é a nova geração do sistema tradicional de aluguel de bicicletas, onde todo o processo, que inclui as fases de inscrição, aluguel e retorno, é feito de modo automático.

Os dados foram obtidos pelo UCI *Machine Learning Repository*, um site que hospeda mais de 400 séries temporais com intuito de ajudar a comunidade de aprendizado de máquina. Os dados foram obtidos a cada hora, desde o primeiro dia de janeiro de 2011 até 31 de dezembro de 2012 (UCI, 2013), totalizando mais de 17 mil dados. Como foi observado empiricamente que uma série dessa magnitude requisitaria um tempo muito elevado de processamento, optou-se por reduzi-la para apenas 7 mil amostras, conforme ilustrado na Figura 21.

Figura 21 – Aluguel de Bicicletas em Porto

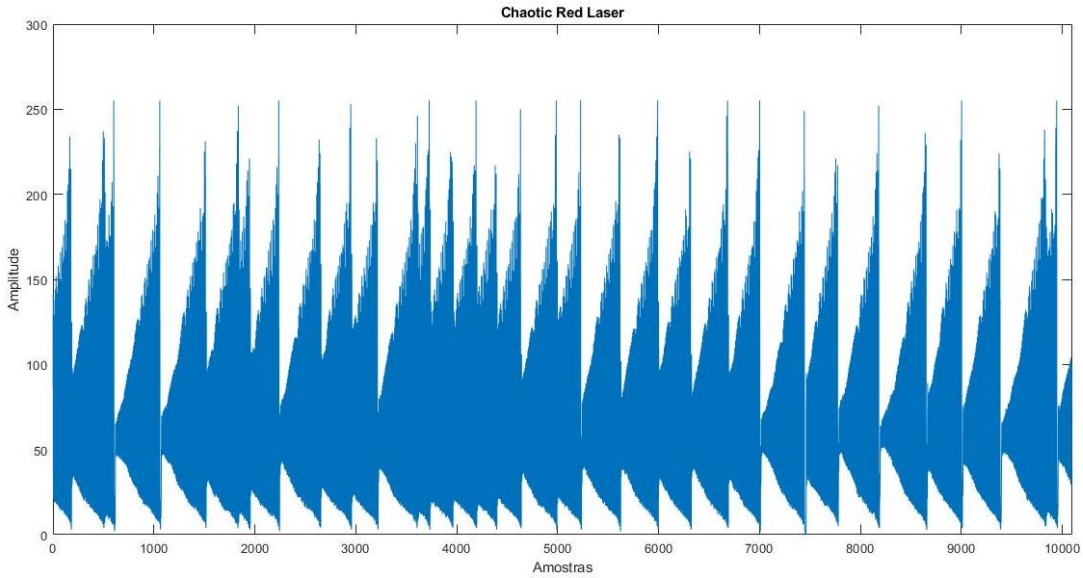


Fonte: Adaptado pelo autor.

3.6.5 Chaotic Red Laser

Os dados consistem em uma medição feita em um laser do tipo 81.5-micron 14NH3 cw, representando a intensidade do lazer em um período caótico de atividade. A série também foi obtida por meio do *Matlab*, contendo cerca de 10 mil amostras, como se pode observar na Figura 22.

Figura 22 – *Chaotic Red Laser*



Fonte: Adaptado pelo autor.

4 RESULTADOS E ANÁLISES

4.1 Metodologia utilizada para obtenção dos resultados

A metodologia de análise dos resultados obtidos compreende a observação conjunta de 4 recursos, para cada série analisada. Sendo estes, uma tabela de análise de desempenho, duas figuras, ambas exibindo os dados previstos, e um gráfico de análise de horizonte de previsão.

4.1.1 Tabela de Análise de Desempenho

As tabelas de análise de desempenho contemplam os itens citados na Seção 3.5 e mencionados na sequência: RMSE, MAPE, tempo de processamento e unidades ocultas. Nestas tabelas, tanto o RMSE quanto o MAPE foram calculados a partir da predição da série original, considerando-se os horizontes de teste, compreendidos pelos 15% finais das mesmas.

Para a obtenção das predições utilizadas para o cálculo do RMSE e do MAPE, a rede previa o próximo valor da série, sendo este realimentado, de forma a prever o valor seguinte, e assim sucessivamente, até atingir um número de predições equivalentes ao total de dados da série de teste.

O tempo de processamento considera todo o tempo destinado ao treinamento e validação da rede. Já o número de unidades ocultas corresponde ao da melhor rede encontrada obtida pela variação de parâmetros, como mencionado na Seção 3.5.

4.1.2 Figuras

A primeira figura exibida para cada série mostra os dados de predição desta série, considerando um horizonte de tamanho igual aos 15% de dados da série original, como descrito anteriormente. Desta forma, pode-se visualizar os dados da série de teste juntamente com as predições realizadas pela rede utilizando LSTM e pela rede NAR.

A segunda figura foi introduzida neste contexto pelo fato de que, como em certas séries o número de dados destinado ao teste era elevado, em algumas das figuras geradas originalmente, mal se podia observar o comportamento dos dados. Sendo assim, optou-se por dar um *zoom* na imagem anterior de forma a mostrar apenas os 300 primeiros pontos de predição.

4.1.3 Gráfico

Para cada série analisada foi apresentado um gráfico de barras que permite a comparação entre a rede baseada em LSTM e a NAR, no quesito RMSE por horizonte de predição.

Para obtenção deste gráfico a rede previu um certo número de passos à frente (horizonte de previsão), de tal forma que, para um horizonte de previsão de 5 observações, por exemplo, a rede previa 5 valores futuros, a partir do valor presente, sendo o processo feito de forma sequencial. Desta forma, pode-se comparar os últimos valores previstos com os seus respectivos valores reais, propiciando assim o cálculo de RMSE para determinado horizonte de previsão.

4.2 Resultados

A seguir são apresentados os resultados obtidos para cada base de dados utilizando a metodologia exposta na Seção 4.1.

4.2.1 Mackey-Glass *Chaotic Series*

A Tabela 4 fornece os parâmetros comparativos de desempenho para ambas as redes. Tanto RMSE e MAPE foram calculados para um horizonte de previsão de 449, equivalente ao total de dados destinados ao teste. O tempo de processamento (TP) foi considerado para todo o processo de treinamento e validação das redes, segundo metodologia abordada na Seção 3.4. O número de camadas ocultas corresponde ao da rede com menor RMSE, obtido pela metodologia abordada na Seção 3.4. Para rede NAR foi empregado um *delay* igual a 30.

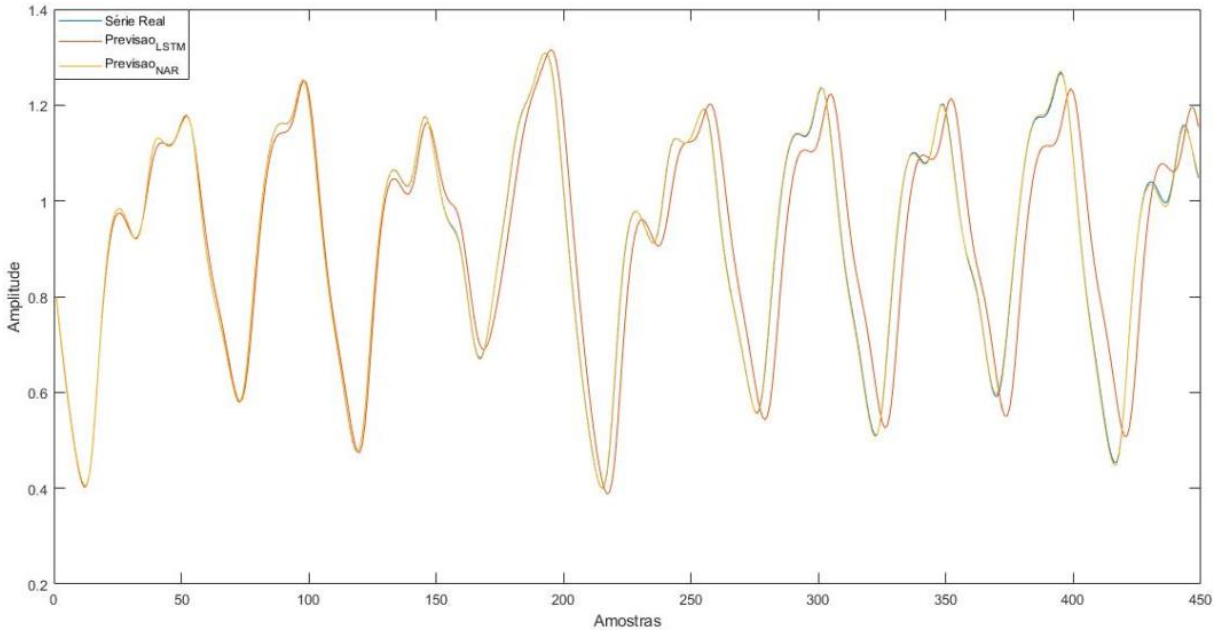
Tabela 4 – Análise de Desempenho Mackey-Glass

Parâmetros	LSTM	NAR
RMSE	0,092	0,003
MAPE (%)	0,29	0,02
TP (s)	6256	3420
Unidades ocultas	125	10

Fonte: Autoria própria.

A Figura 23 mostra a previsão para um horizonte de previsão de 449 pontos, equivalente ao total de dados destinados para testes, tanto para rede com LSTM quanto para rede NAR. Além disso, são expostos os dados de testes da série.

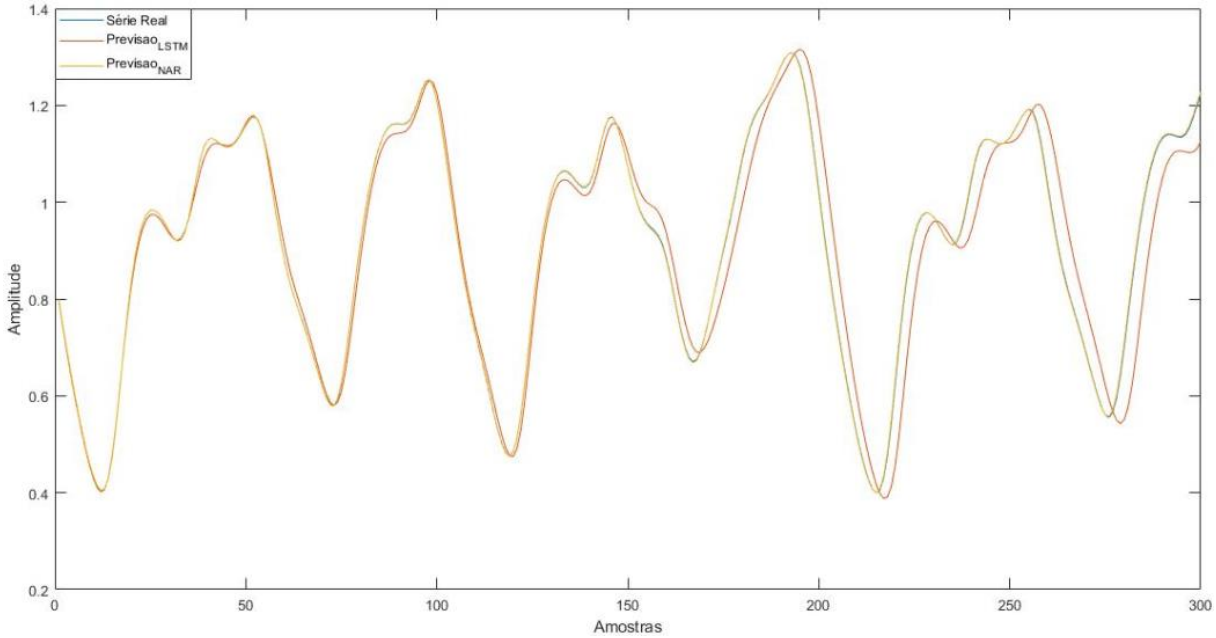
Figura 23 – Predição da série Mackey-Glass



Fonte: Autoria própria.

A Figura 24 representa uma aproximação da Figura 23 para que se possa observar melhor o comportamento das series para os 300 primeiros pontos de previsão.

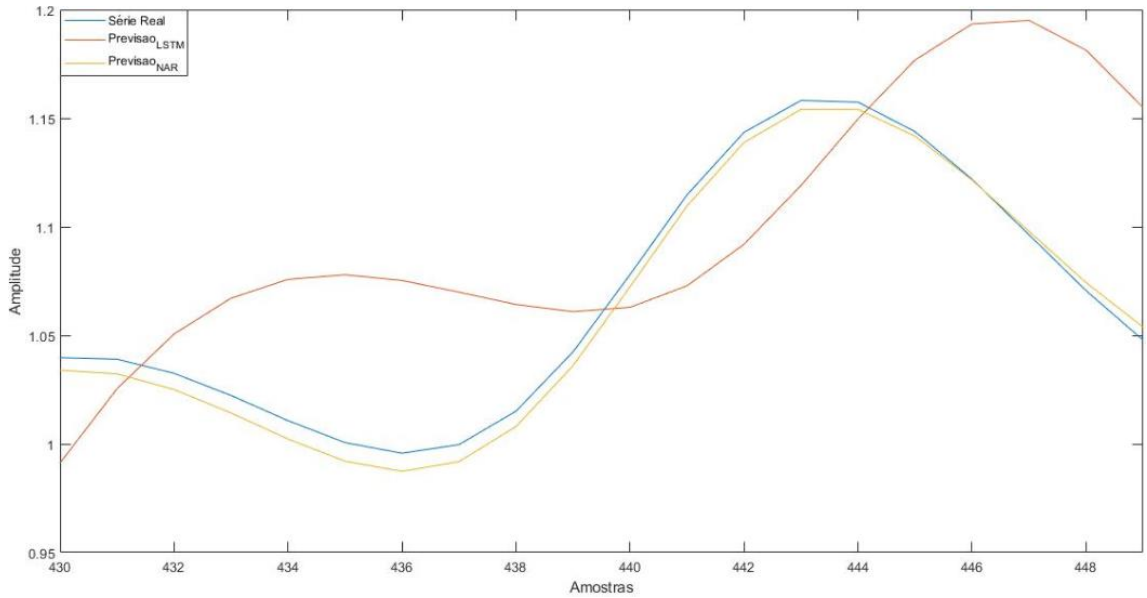
Figura 24 – Predição da série Mackey-Glass com 300 Pontos



Fonte: Autoria própria.

A Figura 25 foi inserida com intuito de se evidenciar que a predição da rede NAR está tão próxima da série real que foi preciso dar um *zoom* nos últimos pontos de predição para que se pudesse distingui-los.

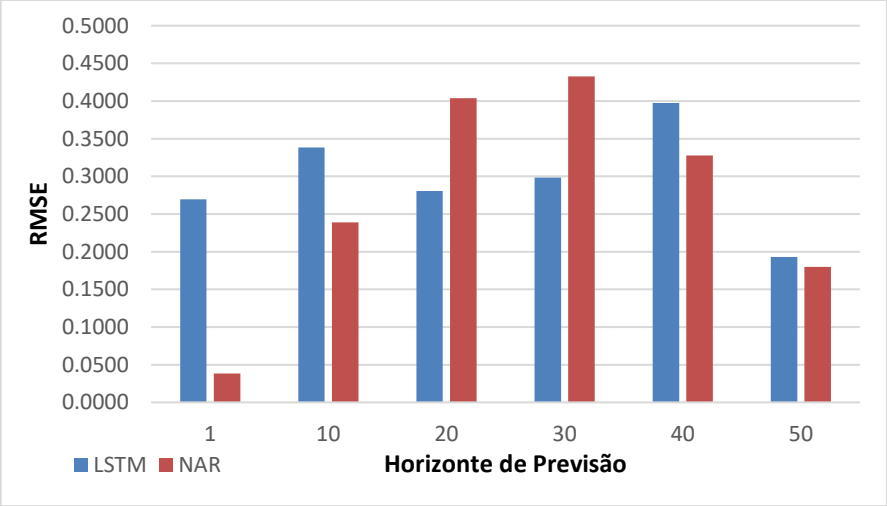
Figura 25– Predição da série Mackey-Glass ultimos pontos



Fonte: Autoria própria.

O Gráfico 1 representa os valores de RMSE obtidos pelas duas redes para um horizonte de previsão igual 1, 10, 20, 30, 40 e 50, respectivamente.

Gráfico 1 – RMSE x Horizonte de Previsão (Mackey-Glass)



Fonte: Autoria própria.

Analisando as Figuras 23, 24 e 25, percebe-se uma predição mais fidedigna da rede NAR do que para rede LSTM, de forma a ser difícil, visualmente, notar a diferença da série real para a série prevista pela rede NAR. Esta conclusão é consolidada nos parâmetros de comparação de erro RMSE e MAPE, ambos significativamente menores.

Isso se deve ao fato das predições realizadas pela rede com LSTM apresentar um atraso gradativo em relação a série original, claramente percebido ao final dos dados. O mesmo pode-se notar com a análise realizada por Yeo (2017), em seu trabalho sobre o comportamento da LSTM para predição de valores da série Mackey-Glass. Neste estudo, Yeo (2017) encontra um bom desempenho para os primeiros 100 pontos previstos, mas a partir daí os resultados começam a divergir.

Ao analisar o quesito horizonte de predição (Gráfico 1) percebe-se, em geral, um melhor desempenho da rede NAR. Contudo, para determinados valores, a rede utilizando LSTM chega a obter resultados bem próximos, chegando, inclusive, a obter melhor desempenho para certos horizontes de previsão.

Observa-se também um menor tempo requerido de processamento da rede NAR, isso se deve ao fato da diferença acentuada de células ocultas, o que implica diretamente na quantidade de parâmetros a serem determinados.

4.2.2 Sunspots

A Tabela 5 fornece os parâmetros comparativos de desempenho para ambas as redes. Tanto RMSE e MAPE foram calculados para um horizonte de previsão de 434, equivalente ao total de dados destinados ao teste. Para rede NAR foi empregado um *delay* igual a 40.

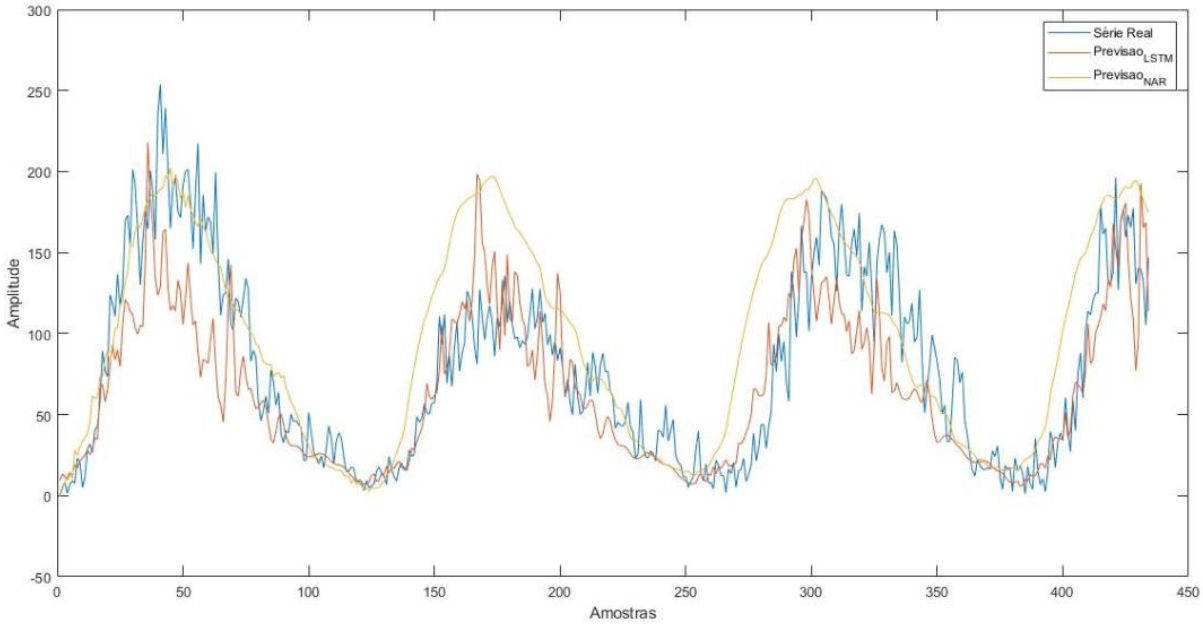
Tabela 5 – Análise de Desempenho *Sunspots*

Parâmetros	**LSTM**	**NAR**
RMSE	35,75	41,31
MAPE	22,70	11,37
TP (s)	6199	2040
Unidades ocultas	175	20

Fonte: Autoria própria.

A Figura 26 mostra a previsão para um horizonte de previsão de 434 pontos, equivalente ao total de dados destinados para testes, tanto para rede com LSTM quanto para rede NAR. Além disso, são expostos os dados de testes da série.

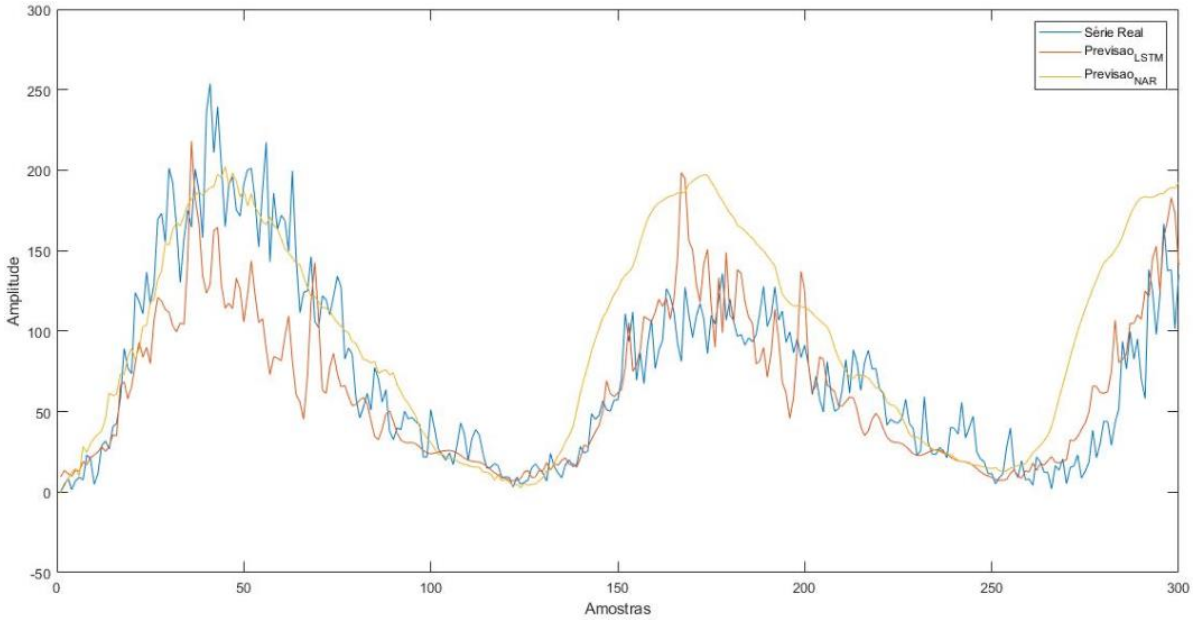
Figura 26 – Predição da série *Sunspots*



Fonte: Autoria própria.

A Figura 27 representa uma aproximação da Figura 26 para que se possa observar melhor o comportamento das series para os 300 primeiros pontos de previsão.

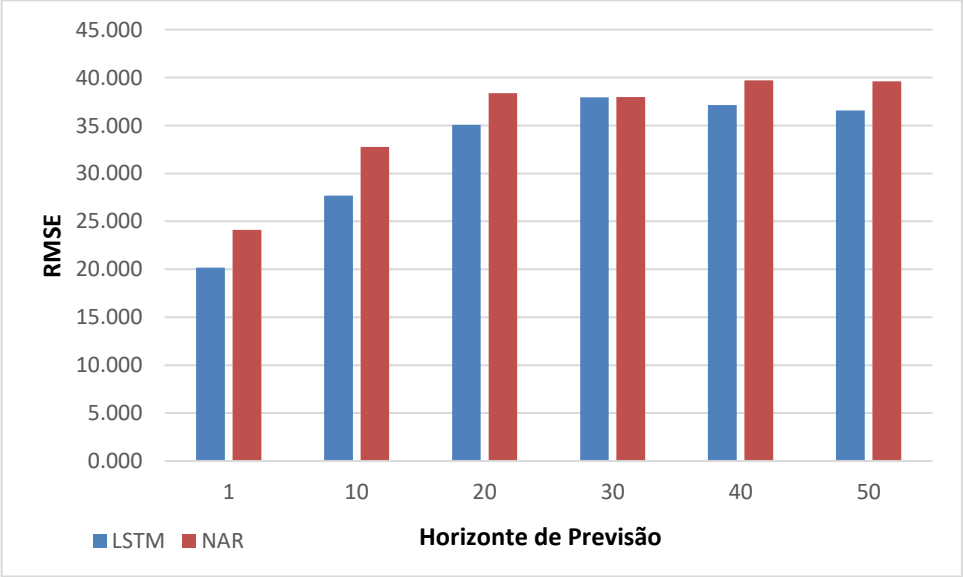
Figura 27 – Predição da série *Sunspots* com 300 Pontos



Fonte: Autoria própria.

O Gráfico 2 representa os valores de RMSE obtidos pelas duas redes para um horizonte de previsão igual 1, 10, 20, 30, 40 e 50, respectivamente.

Gráfico 2 – RMSE x Horizonte de Previsão (*Sunspots*)



Fonte: Autoria própria.

Ao observar a Figura 27, nota-se que para as primeiras predições, cerca de 150 dados, a rede NAR aparenta melhor desempenho, de forma que suas predições se aproximam tanto em amplitude quanto em fase. A partir desta faixa, contudo, observa-se tanto na Tabela 5 quanto nas Figuras 26 e 27, que a predição da rede utilizando LSTM passa a ter maior precisão, refletindo diretamente em um menor RMSE. Esta melhora pode ser explicada pelo fato das predições da rede com LSTM entrarem em fase com os dados reais. A rede NAR, entretanto, repete de certa forma o mesmo comportamento dos 150 dados, ficando assim distantes dos dados reais (Figura 27). Em relação ao quesito horizonte de predição, o resultado de ambas as séries é similar, como pode ser observado pelo Gráfico 2. Pode-se observar, novamente, que o tempo da rede NAR foi significativamente menor do que o da rede baseada em LSTM, aproximadamente em 3 vezes.

4.2.3 Mean Daily Temperature in Fisher River

A Tabela 6 fornece os parâmetros comparativos de desempenho para ambas as redes. Tanto RMSE e MAPE foram calculados para um horizonte de previsão de 219, equivalente ao total de dados destinados ao teste. Para rede NAR foi empregado um *delay* igual a 30.

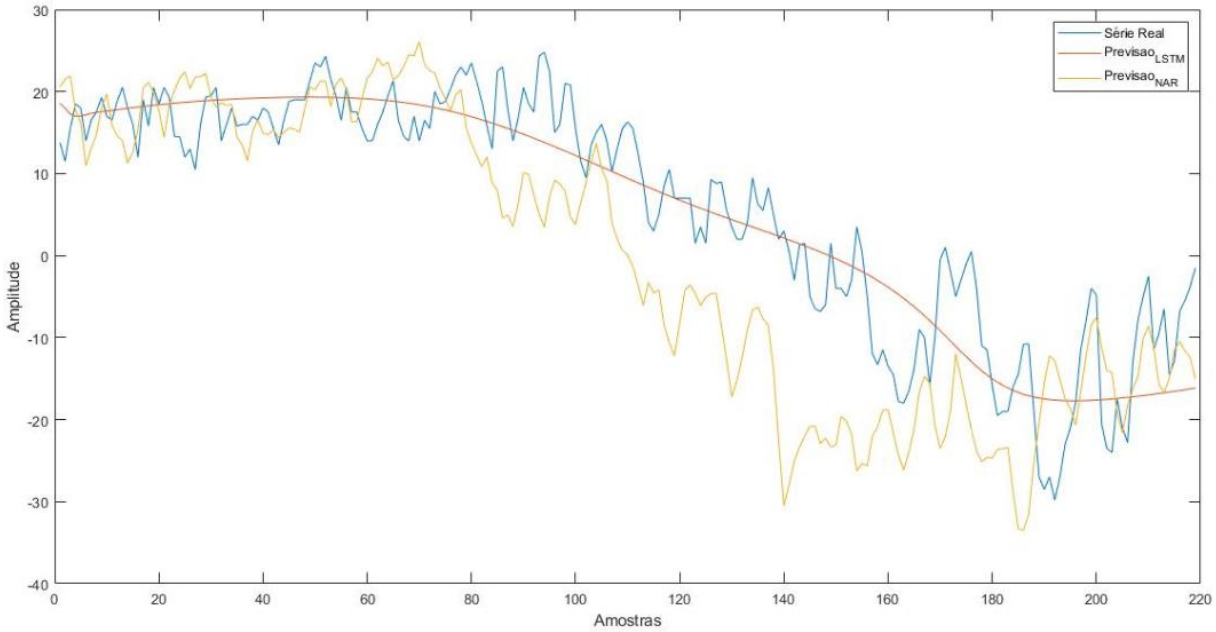
Tabela 6 – Análise de Desempenho *Mean Daily Temperature in Fisher River*

Parâmetros	**LSTM**	**NAR**
RMSE	5,29	11,32
MAPE	9,75	12,15
TP (s)	3796	1687
Unidades ocultas	125	40

Fonte: Autoria própria.

A Figura 28 mostra a previsão para um horizonte de previsão de 219 pontos, equivalente ao total de dados destinados para testes, tanto para rede com LSTM quanto para rede NAR. Além disso, são expostos os dados de testes da série.

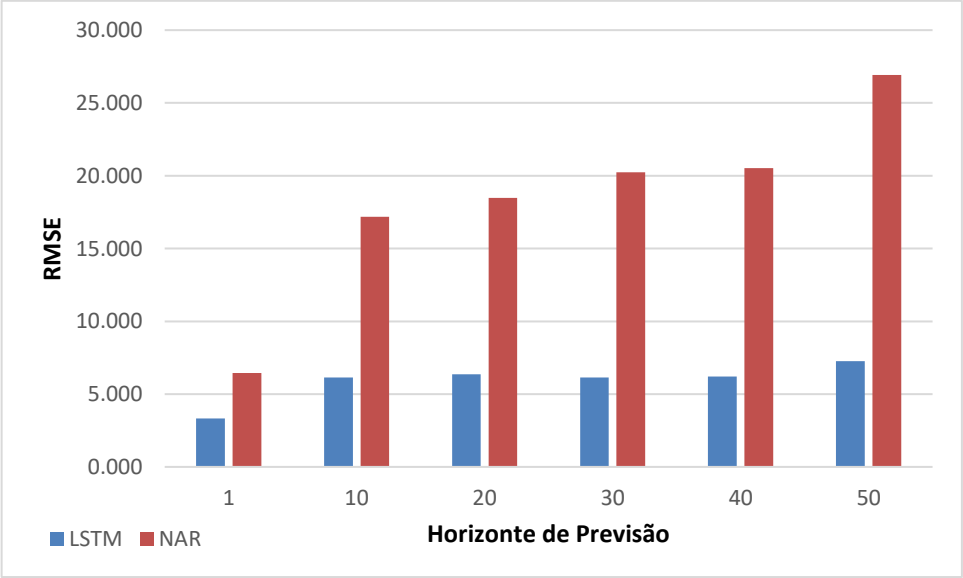
Figura 28 – Predição da série *Mean Daily Temperature in Fisher River*



Fonte: Autoria própria.

O Gráfico 3 representa os valores de RMSE obtidos pelas duas redes para um horizonte de previsão igual 1, 10, 20, 30, 40 e 50, respectivamente.

Gráfico 3 – RMSE x Horizonte de Previsão (*Mean Daily Temperature*)



Fonte: Autoria própria.

Nota-se pela Figura 28 que, inicialmente, até cerca de 70 dados, a rede NAR obtém resultados não muito distantes da série original. Todavia, para os demais pontos a rede NAR tende a se distanciar significativamente.

A rede utilizando LSTM, no entanto, apresenta um comportamento similar por toda predição, como pode ser verificado na Figura 28, onde a rede aproximou-se da média da série.

É importante destacar que o comportamento dos dados previstos pela rede LSTM, nesta série, se assemelha a uma filtragem dos dados da série original. Este comportamento reflete, como observado na Tabela 6, em valores de RMSE e MAPE consideravelmente menores.

Nota-se por meio do Gráfico 3, valores de RMSE significativamente maiores obtidos pela rede NAR em comparação a rede utilizando LSTM para toda faixa de horizonte de predição estabelecida, nesta série.

Novamente, percebe-se pela Tabela 6 um tempo de processamento circunstancialmente maior da rede com LSTM do que para rede NAR, cerca de 2,5 vezes maior.

4.2.4 Aluguel de Bicicletas na Cidade do Porto

A Tabela 7 fornece os parâmetros comparativos de desempenho para ambas as redes. Tanto RMSE e MAPE foram calculados para um horizonte de previsão de 1049, equivalente ao total de dados destinados ao teste. Para rede NAR foi empregado um *delay* igual a 20.

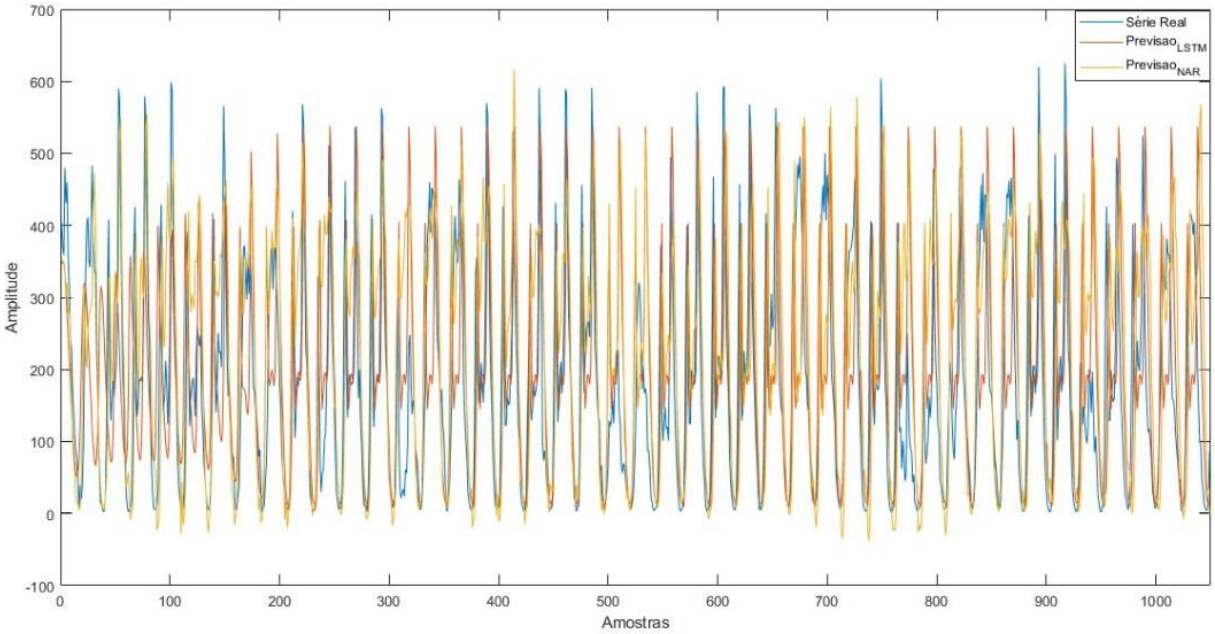
Tabela 7 – Análise de Desempenho Aluguel de Bicicletas em Porto

Parâmetros	**LSTM**	**NAR**
RMSE	133,76	145,70
MAPE	15,30	1,05
TP (s)	12680	3780
Unidades ocultas	175	40

Fonte: Autoria própria.

A Figura 29 mostra a previsão para um horizonte de previsão de 1049 pontos, equivalente ao total de dados destinados para testes, tanto para rede com LSTM quanto para rede NAR. Além disso, são expostos os dados de testes da série.

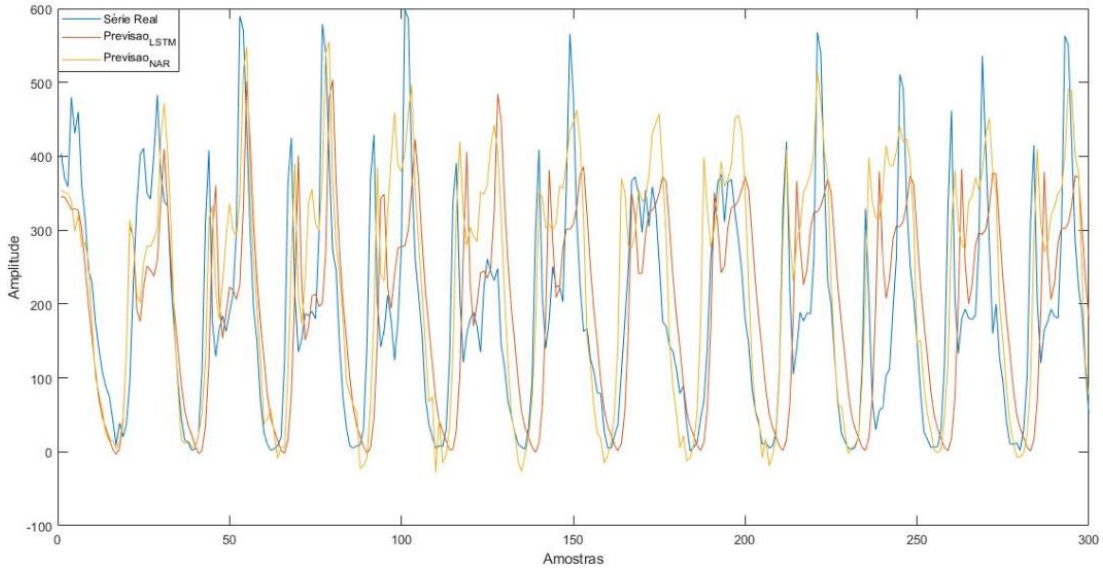
Figura 29 – Predição da série Aluguel de Bicicletas



Fonte: Autoria própria.

A Figura 30 representa uma aproximação da Figura 29 para que se possa observar melhor o comportamento das series para os 300 primeiros pontos de previsão.

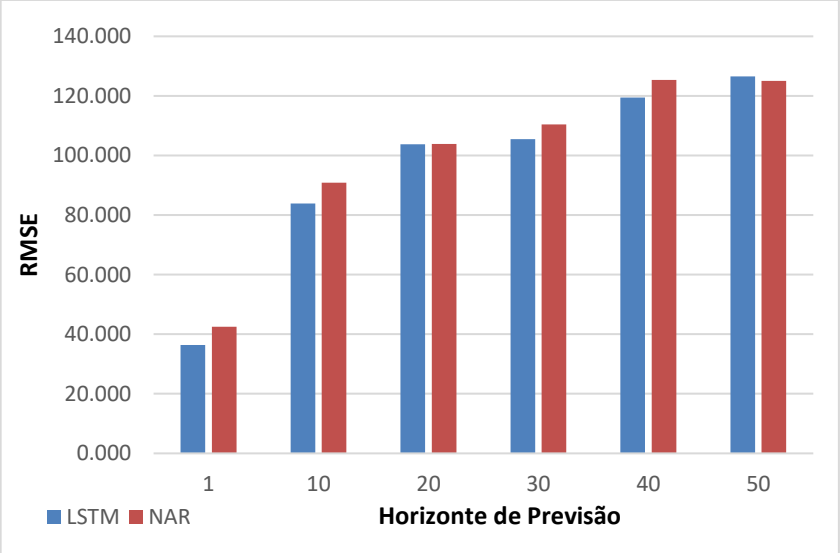
Figura 30 – Predição da série Aluguel de Bicicletas com 300 Pontos



Fonte: Autoria própria.

O Gráfico 4 representa os valores de RMSE obtidos pelas duas redes para um horizonte de previsão igual 1, 10, 20, 30, 40 e 50, respectivamente.

Gráfico 4 – RMSE x Horizonte de Previsão (Aluguel de Bicicletas)



Fonte: Autoria própria.

Ao se atentar para Figura 30 visualiza-se um comportamento de predição bem similar de ambas as redes. Contudo, como indicam o resultado de RMSE, obtido pela Tabela 7, a predição da rede baseada em LSTM, acaba por reproduzir menores índices de erro, na maior parte dos horizontes de previsão exibidos, o que sinaliza um melhor comportamento para dados longínquos.

Um fato interessante, contudo, é o valor de MAPE, proporcionado pela rede NAR, ser aproximadamente 1, enquanto que o obtido pela rede com LSTM é cerca de 15, apesar do RMSE da rede LSTM ser inferior.

A justificativa para este fato está na forma como se é calculado o MAPE, vide Equação 3, onde ocorre uma divisão pelo dado da série real. Em casos cujos valores reais da série se aproximam de zero, o erro na predição tende a ser altamente penalizado, sendo que quando mais perto de zero maior será esta penalização.

Pelo Gráfico 4, percebe-se um valor de RMSE bem similar para ambas as séries para toda faixa de horizonte de predição. Como esperado, o tempo de processamento da rede LSTM é maior do que o da rede NAR, aproximadamente 3,5 vezes maior, como pode-se constatar pela Tabela 7.

4.2.5 Chaotic Red Laser

A Tabela 8 fornece os parâmetros comparativos de desempenho para ambas as redes. Tanto RMSE e MAPE foram calculados para um horizonte de previsão de 1513, equivalente ao total de dados destinados ao teste. Para rede NAR foi empregado um *delay* igual a 20.

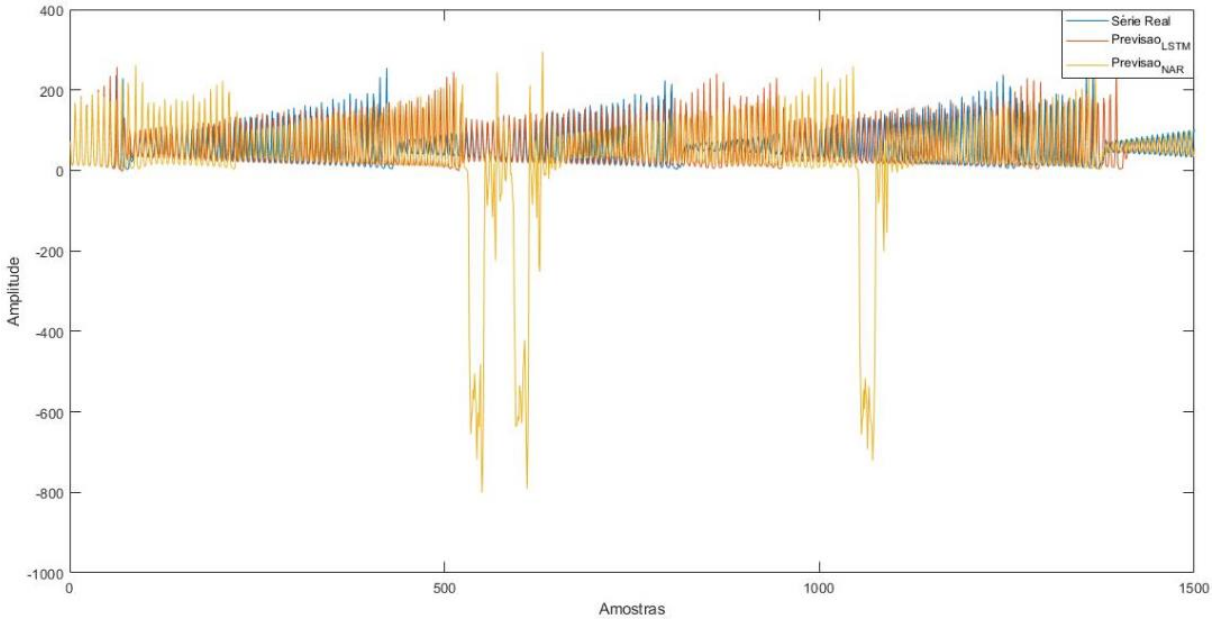
Tabela 8 – Análise de Desempenho *Chaotic Red Laser*

Parâmetros	**LSTM**	**NAR**
RMSE	67,93	148,82
MAPE	32,01	68,35
TP (s)	18427	6600
Unidades ocultas	150	40

Fonte: Autoria própria.

A Figura 31 mostra a previsão para um horizonte de previsão de 1513 pontos, equivalente ao total de dados destinados para testes, tanto para rede com LSTM quanto para rede NAR. Além disso, são expostos os dados de testes da série.

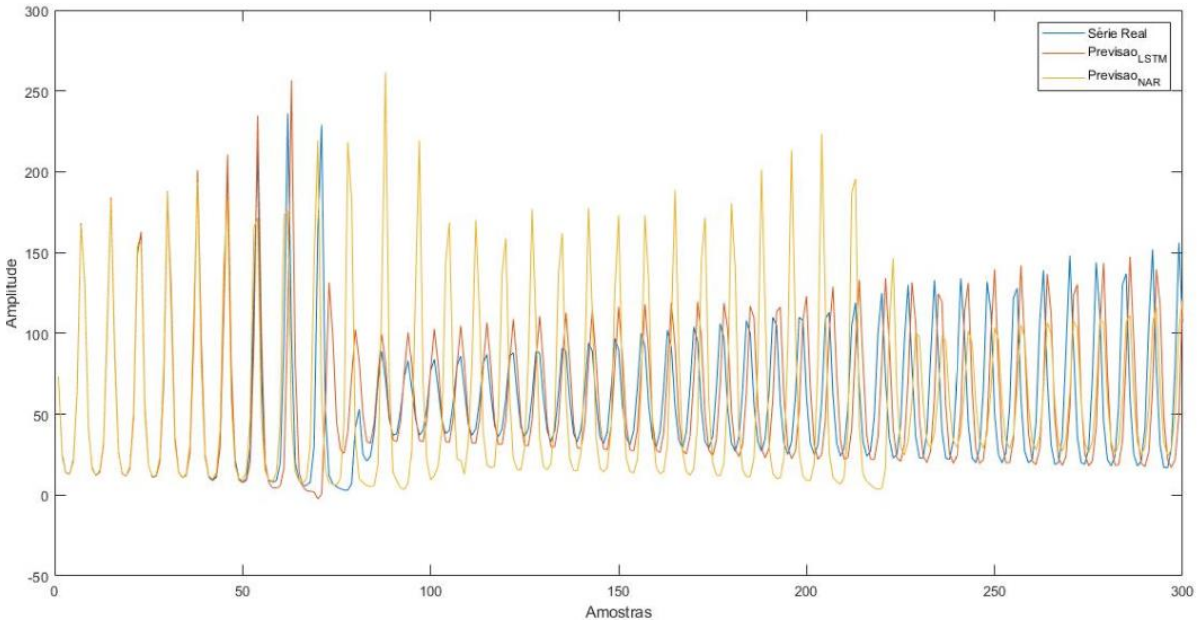
Figura 31 – Predição da série *Chaotic Red Laser*



Fonte: Autoria própria.

A Figura 32 representa uma aproximação da Figura 31 para que se possa observar melhor o comportamento das series para os 300 primeiros pontos de previsão.

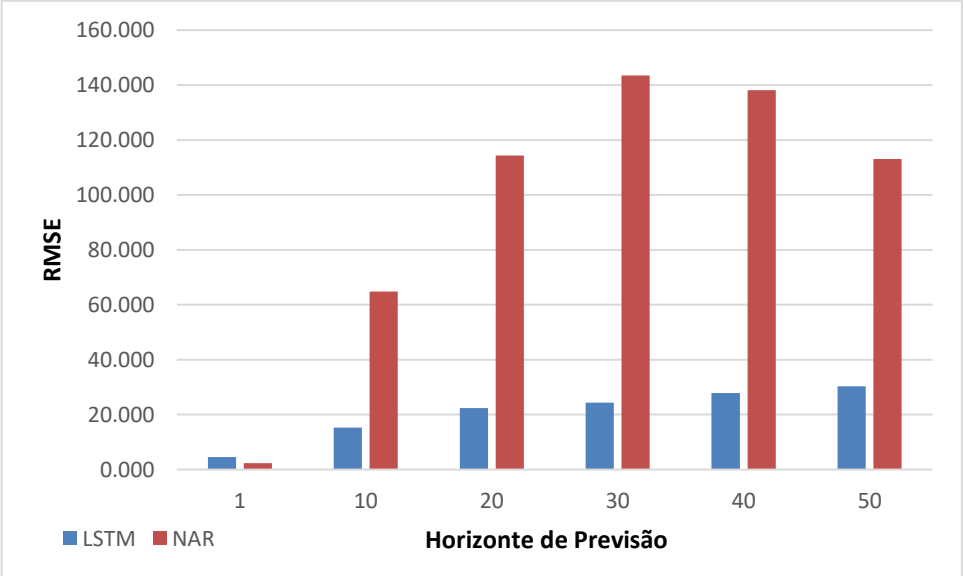
Figura 32 – Predição da série *Chaotic Red Laser* com 300 Pontos



Fonte: Autoria própria.

O Gráfico 5 representa os valores de RMSE obtidos pelas duas redes para um horizonte de previsão igual 1, 10, 20, 30, 40 e 50, respectivamente.

Gráfico 5 – RMSE x Horizonte de Previsão (*Chaotic Red Laser*)



Fonte: Autoria própria.

Verifica-se, por meio da Figura 32, que para as predições iniciais, até cerca de 150 pontos, tanto a rede NAR quanto a baseada em LSTM, acabam prevendo os pontos de maneira similar. Contudo, à medida que se avança nos dados previstos, percebe-se uma diferença de desempenho. Esta diferença se propaga para os demais dados de predição, sendo percebida, inclusive, por intermédio da Figura 31, onde destaca-se elevada discrepância da rede NAR em certos trechos.

Como resultado, a Tabela 8, mostra tanto um valor de RMSE quanto de MAPE significativamente inferiores para rede baseada em LSTM. A Tabela 8 também indica um tempo de processamento maior da rede com LSTM, neste caso de cerca de 3 vezes o valor obtido pela rede NAR

Analisando-se o Gráfico 5, percebe-se para um horizonte unitário um desempenho similar entre as duas redes, contudo, para os demais valores, a rede NAR acaba obtendo valores consideravelmente mais altos de RMSE. Pode-se concluir, portanto, que a rede NAR não foi capaz de prevê corretamente o valor da série.

4.3 Análises

É importante destacar a complexidade de cada uma das séries utilizadas. A rede NAR acaba por ser uma rede mais simples, isso porque possui um menor número de camadas, além de, em geral, utilizar poucas unidades ocultas para obter um desempenho relativamente satisfatório. Sendo assim, a rede possui um número menor de variáveis a serem encontradas, levando, desta forma, a um menor tempo de processamento da rede. Esta diferença entre o treinamento da rede NAR e a rede utilizando LSTM é comprovada através das análises das Tabelas de 4 a 8, onde encontra-se uma diferença de tempo total de processamento da ordem de 3 vezes.

Nota-se também que, como observado na Seção 4.1, para séries reais o modelo utilizando LSTM tende a manter uma predição com menores índices de RMSE se comparado à rede NAR. Isso se deve a alta capacidade das células de memória da LSTM em aprender com dados passados (HOCHREITER; SCHMIDHUBER, 1997).

Pode-se perceber que as redes com LSTM tendem a apresentar melhor desempenho para horizontes longínquos de predição. Em contrapartida, suas complexidades, relacionadas à série de parâmetros a serem determinados e aprendidos, faz com que seja necessária maior capacidade de processamento dos dados, para que se obtenha o desempenho adequado.

Os resultados demonstrados para rede utilizando LSTM poderiam ter sido melhores se o número de valores variados nos hiperparâmetros de treinamento fosse maior. Sabe-se que quanto maior a faixa de variação dos parâmetros, maior a chance de se encontrar a faixa de operação ótima da rede. Além disso, ao contrário da rede NAR onde se tem apenas 4 parâmetros a se variar, a rede LSTM conta com alguns outros, em especial *LearnRateDropPeriod*, período de decaimento da taxa de aprendizado, e *LearnRateDropFactor*, fator de decaimento da taxa de aprendizado, que foram testados durante os testes preliminares e, causaram impacto nos resultados.

Contudo, destaca-se que as medidas adotadas na Seção 3.4 levaram em conta o custo computacional envolvido para execução dos testes. Um aumento, tanto na faixa de variação dos parâmetros quanto na quantidade destes parâmetros, implicaria no crescimento do número de combinações, o que resultaria em um tempo de processamento mais elevado.

Ao contabilizar o tempo total para as 5 redes com as 256 combinações, se chegará a um total de cerca de 20 horas de processamento. Um aumento no número de combinações implicaria em um tempo de processamento de dias ou até semanas, o que tornaria a ferramenta inviável.

Pode-se concluir que apesar do número total de pontos das séries testadas ser baixo, em média 5 mil pontos, devida a variação dos parâmetros utilizada como forma de se chegar no ponto de operação ótimo da rede, o poder de processamento do dispositivo utilizado (um computador pessoal) não foi adequado, mesmo contendo uma *Graphics Processing Unit* (GPU).

5 CONCLUSÕES

Este trabalho teve como objetivo realizar uma análise comparativa acerca das predições de séries temporais entre redes neurais baseadas em *deep learning*, em especial a LSTM, e redes neurais clássicas, em especial a rede NAR. Para avaliação foram utilizadas quatro séries reais, além de uma série adicional resultante de equacionamento matemático que simula uma série levemente caótica. Tendo em vista o apresentado pelos capítulos 3 e 4, nos quais se abordou a metodologia de implementação e comparação das redes juntamente com os respectivos resultados das predições das redes, o objetivo geral do trabalho foi alcançado.

Em relação aos objetivos específicos, ambos foram atendidos. Após pesquisas sobre as áreas de atuação do *deep learning* e suas redes neurais associadas, encontrou-se uma arquitetura de rede neural baseada em *deep learning* capaz de realização predições para séries temporais. Além disso, implementou-se tanto a rede baseada em *deep learning* utilizando LSTM, quanto a rede clássica, especificamente a rede NAR.

Em relação as predições obtidas realizando as cinco séries temporais adotadas, a rede utilizando LSTM apresentou melhor desempenho para horizontes de previsões longínquos. Contudo, o custo computacional envolvido para sua implementação foi circunstancialmente maior do que para rede NAR. É importante destacar que existem outras arquiteturas de redes neurais utilizando LSTM, em que seus resultados demostraram ser promissores (BAO; YUE; RAO, 2017).

5.1 Trabalhos Futuros

Uma alternativa é a utilização da ferramenta, implementada pela *Google*, *Cloud AI*, especialmente desenvolvida para trabalhar com aprendizado de máquina. Este recurso conta com processamento via sistema de computadores da *Google,* cuja capacidade computacional supera de maneira significativa a de computadores pessoais e comerciais disponíveis, tornando possível executar treinamentos complexos em períodos de tempo factíveis.

Para um trabalho futuro, poder-se utilizar a metodologia referenciada como WSAEs-LSTM proposta por Bao; Yue; Rao (2017). Em seu artigo, eles propõem uma junção de metodologias com intuito de se criar um *framework* com alta eficácia em predição de séries temporais, em especial para séries financeiras. O modelo é constituído de 3 partes: Transformada *Wavelet* (TW), *Stacked AutoEncoder* (SAE) e LSTM. A série primeiramente é decomposta via TW para que se possa filtrar o ruído. Em seguida, SAE é aplicada para gerar *features* altamente profundas. Por fim, essas *features* são utilizadas pela rede LSTM para predição da série. O treinamento da SAE é feito de maneira que cada *AutoEncoder* (AE) seja treinado de cada vez por minimização do erro de entrada e saída, permitindo que a SAE seja capaz de aprender *features* invariantes e abstratas.

Ao analisar os resultados da metodologia proposta por Bao; Yue; Rao (2017), percebe-se que este modelo obteve melhores resultados, se comparado às metodologias LSTM e WLSTM (LSTM com TW), para a utilização de predição de séries temporais, com foco na avaliação a rentabilidade das ações analisadas no artigo. Indicando, desta forma, ser uma arquitetura de rede promissora para área de predições de séries temporais.

6 REFERÊNCIAS

BAO, W.; YUE, J.; RAO, Y. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. **PLOS ONE**, 2017.

BATRES-ESTRADA, G. **Deep Learning for Multivariate Financial Time Series.** 2015. Tese (Doutorado em Engenharia Elétrica) – Programa de Pós-Graduação em Engenharia Elétrica, KTH Royal Institute of Technology, Stockholm, 2015.

BENMOUIZA, K; CHEKNANE, A. Small-Scale Solar Radiation Forecasting Using ARMA and Nonlinear Autoregressive Neural Network Models. **Theoretical and Applied Climatology**, 2015.

BEALE, M;HAGAN, M; DEMUTH, H . **Matlab Neural Network Toolbox User's Guide**. Massachusetts,2018.

BROWNLEE, J. Long Short-Term Memory Networks With Python: Develop sequence prediction models with deep learning. [S.l.]: Jason Brownlee, 2017.

BUDUMA, N. **Fundamentals of Deep Learning.** 1ª ed. California: O'Reilly Media, 2017.

CHATTOPADHYAY, G; CHATTOPADHYAY, S. Monthly sunspot number time series analysis and its modeling through autoregressive artificial neural network. **EUR. PHYS. J. PLUS**, 2012.

COOK, T; HALL, ARON. Macroeconomic Indicator Forecasting with Deep Neural Networks. **Federal Reserve Bank of Kansas City**, Research Working Paper 17-11, setembro, 2017.

DALTO, M.; MATUSKO, J.; VASAK, M. Deep neural networks for ultra-short-term wind forecasting. **IEEE International Conference on Industrial Technology (ICIT)**, p.1657-1663, mar. 2014.

DUA, D;TANISKIDOU, K. UCI Machine Learning Repository. **Bike Sharing Dataset Data Set.** Disponível em: < https://archive.ics.uci.edu/ml/datasets>. Acesso em: 06 de junho de 2018.

ECONOMIA UOL CÂMBIO DÓLAR COMERCIAL. Disponível em: <https://economia.uol.com.br/cotacoes/cambio/dolar-comercial-estados-unidos/>. Acesso em: 08 de junho de 2018.

ERWINSKI, K; PAPROCKI, M; WAWRZAK, A; GRZESIAK, L. Neural Network Contour Error Predictor in CNC Control Systems. **21st International Conference on Methods and Models in Automation and Robotics (MMAR)**, Miedzyzdroje, Polônia, 2016.

GAMBOA, J. Deep Learning for Time-Series Analysis. **Seminar on Collaborative Intelligence**, 2017.

GREFF, K; SRIVASTAVA, R; KOUTNÍK, J; STEUNEBRINK, B; SCHMIDHUBER, J. LSTM: A Search Space Odyssey. **IEEE Transactions on Neural Networks and Learning Systems**, vol.28, p.2222-2232, 2017.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning.** Massachusetts: MIT Press, 2017.

HAGAN, T; MENHAJ, M. Training Feedforward Networks with the Marquardt Algorithm. **IEEE Transactions on Neural Networks**, vol.5, 1991.

HARTLEY, R; ZISSERMAN, A. **Multiple View Geometryin Computer Vision**, 2ª ed. Nova York: Cambridge University Press, 2004.

HAYKIN, S. **Neural networks and learning machines**, 3ª ed. Nova Jérsia: Pearson Education, 2009.

HOCHREITER, S; SCHMIDHUBER, J. Long Short-Term Memory. **Neural Computation**, vol.9, p.1735-1780,1997.

HONGWEI,L. On the Levenberg-Marquardt training method for feed-forward neural networks. **Sixth International Conference on Natural Computation**, Yantai, China, 2010.

HYNDMAN, R. DataMarket. **Mean daily temperature, Fisher River near Dallas, Jan 01, 1988 to Dec 31, 1991**. Disponível em: < https://datamarket.com/data/list/?q=provider:tsdl >. Acesso em: 30 de maio de 2018.

KINGMA, D; BA, J. Adam: A Method for Stochastic Optimization. **3rd International Conference for Learning Representations**, San Diego, 2015.

KUREMOTO, T.; KIMURA, S.; KOBAYASHI, K.; OBAYASHI, M. Time series forecasting using a deep belief network with restricted Boltzmann machines. **Neurocomputing**, v.137, p.47-56, mar. 2013.

LEVENBERG, K. A Method for the Solution of Certain Non-linear Problems in Least Squares. **Quarterly of Applied Mathematics**, p. 164-168, 1944.

LE, Q. A Tutorial on Deep Learning Part 1: Nonlinear Classifiers and The Backpropagation Algorith. **Google Brain**, Mountain View, California, 2015.

LI, C.; DING, Z.; ZHAO, D.; YI, J.; ZHANG, G. Building Energy Consumption Prediction: An Extreme Deep Learning Approach. **Energies**, v.10, p.1525-1524**,** out. 2017.

LI, X; ZHAO, Z. Measures of Performance for Evaluation of Estimators and Filters. SPIE. **Conference on Signal and Data Processing of Small Targets**, San Diego, California, 2001.

MARQUARDT, D. An Algorithm for the Least-Squares Estimation of Nonlinear Parameters. **SIAM Journal of Applied Mathematics**, p. 431-441, 1963.

MORETTIN, P. A.; TOLOI, C. M. C. **Análise de Séries Temporais.** 1ª. ed. Rio de Janeiro: Editora Atual, 1981.

PASCANU, R.; MIKOLOV, T.; BENGIO, Y. On the difficulty of training recurrent neural networks. **JMLR W&CP**, v. 28, p.1310-1318, 2013.

QIU, X; ZHANG, L; REN, Y; AMARATUNGA, G. Ensemble Deep Learning for Regression and Time Series Forecasting. **IEEE SSCI 2014-2014 IEEE Symposium Series on Computational Intelligence**, 2014.

SAK, H., SENIOR, A., BEAUFAYS, F. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. **ARXIV**, 1402.1128, 2014.

SHAN, L; GUAN, N; YANG, C; XU, W; ZHANG, M. Delay Compensated Asynchronous Adam Algorithm for Deep Neural Networks. **2017 IEEE International Symposium on Parallel and Distributed Processing with Applications**, 2017.

SCHMIDHUBER J. Deep Learning in Neural Networks: An Overview. **Neural Networks**, v.61, p.85-117, 2014.

Time Series Prediction Repository. Disponível em: <http://sites.labic.icmc.usp.br/icmc_tspr/index.php/datasets >. Acesso em: 06 de junho de 2018.

TELGARSKY, M. Representation Benefits of Deep Feedforward Networks. **ARXIV**, 1509.08101, 2015

THEVENAZ, P; RUTTIMANN, U; UNSER, M. A Pyramid Approach to Subpixel Registration Based on Intensity. **IEEE Transactions on Image Processing**, vol.7, número 1, 1998.

VENGERTSE, D. Deep Learning Architecture for Univariate Time Series Forecasting. **CS229**, 2014.

WERBOS, P .Backpropagation through time: what it does and how to do it.

Proceedings of the IEEE, **v.78, 1990.**

YEO, K. Model-free prediction of noisy chaotic time series by Deep Learning. **IBM T.J. Watson Research Center**, Yorktown Heights, Nova York, 2017.