UNIVERSIDADE FEDERAL DO PARANÁ

VICTOR UDO OBRIST BERTRAND

APLICAÇÃO DE REDES NEURAIS NOS RESÍDUOS DE DESLOCAMENTOS HORIZONTAIS DE BLOCOS DA BARRAGEM PRINCIPAL DA USINA HIDRELÉTRICA DE ITAIPU

CURITIBA

2018

VICTOR UDO OBRIST BERTRAND

APLICAÇÃO DE REDES NEURAIS NOS RESÍDUOS DE DESLOCAMENTOS HORIZONTAIS DE BLOCOS DA BARRAGEM PRINCIPAL DA USINA HIDRELÉTRICA DE ITAIPU

Monografia apresentada como requisito parcial à obtenção do título de Especialista, Curso de Especialização em Métodos Numéricos em Engenharia, Setor de Ciências Exatas e Setor de Tecnologia, Universidade Federal do Paraná.

Orientador: Prof. Dr. Anselmo Chaves Neto.
Coorientador: Prof. Dr. Jairo Marlon Corrêa.

CURITIBA
2018

FOLHA/TERMO DE APROVAÇÃO

VICTOR UDO OBRIST BERTRAND

APLICAÇÃO DE REDES NEURAIS NOS RESÍDUOS DE DESLOCAMENTOS HORIZONTAIS DE BLOCOS DA BARRAGEM PRINCIPAL DA USINA HIDRELÉTRICA DE ITAIPU

Monografia aprovada como requisito parcial à obtenção do título de Especialista, Curso de Especialização em Métodos Numéricos em Engenharia, Setor de Ciências Exatas e Setor de Tecnologia, Universidade Federal do Paraná, pela seguinte banca examinadora:

Dr. Anselmo Chaves Neto
Orientador - Programa de Pós-Graduação em Métodos Numéricos em Engenharia - UFPR

Dra. Isabella Andreczevski Chaves
Departamento de Construção Civil - UFPR

Dr. Emerson Lazzarotto
Centro de Engenharia e Ciências Exatas - UNIOESTE

CURITIBA
2018

Seja a resposta, não a pergunta

Yogi Bhajan

Dedicatória

Dedico este trabalho a minha esposa Lucía Rodríguez, pelo apoio incondicional durante a especialização e durante a minha vida.

Agradecimentos

A Deus pela vida.

Quero agradecer imensamente ao Dr. Anselmo Chaves Neto, meu Orientador, por todo o apoio durante o desenvolvimento do projeto, bem como as forças, o tempo dado no desenvolvimento do trabalho.

Ao Dr. Jairo Marlon Corrêa, por toda a ajuda brindada, a boa disposição e gentileza em todo momento.

Ao CEASB/PTI, pela oportunidade de fazer o programa e por brindar todo o apoio logístico e direção durante este trabalho.

Aos professores do PPGMNE UFPR, pelas aulas e ensinamentos.

A UFPR por ter estendido o seu alcance ao Oeste do Paraná e proporcionar uma excelente formação.

Resumo

Este trabalho propõe a aplicação da metodologia de Redes Neurais Recorrentes nos resíduos de quatro Séries Temporais de dados dos Deslocamentos Horizontais dos Blocos da Barragem Principal da Usina Hidrelétrica de Itaipu. Esses resíduos resultaram da modelagem da série por meio da metodologia Box & Jenkins. Dessa modelagem obteve-se uma parte linear e depois aplicada a Rede Neural Recorrente chamada Long Short-Term Memory (LSTM) aos resíduos da série (parte não linear). A partir de então, foram gerados valores não lineares que combinados com a parte linear, constituem os valores de previsões. Foi feito um comparativo da previsão da série com a metodologia Box & Jenkins com o modelo misto. Os resultados mostram que o modelo misto apresenta melhores resultados em três das quatro Séries Temporais estudadas, com redução do erro de até 40%.

Palavras chaves: Séries Temporais, Redes Neurais Recorrentes, Box & Jenkins, Long Short-Term Memory.

Abstract

This work proposes the application of the Recurrent Neural Networks methodology in the residuals of a Temporal Series of data of the Horizontal Displacements of the Blocks of the Main Dam of the Itaipu Hydroelectric Power Plant. These residues resulted from the modeling of the series by the Box & Jenkins methodology. From this model, a linear part was obtained and then a Recurrent Neural Network called Long Short-Term Memory (LSTM) was applied to the residuals of the series (non-linear part). Since then, nonlinear values have been generated that, combined with the linear part, are the prediction values. A comparison of the prediction of the series with the Box & Jenkins methodology with the mixed model was made. The results show that the mixed model presents better results in three of the four Time Series studied, with a reduction of the error up to 40%.

Key words: Temporal Series, Recurrent Neural Networks, Box & Jenkins, Long Short-Term Memory.

Lista de Figuras

Figura 3.1 Modelo de um Neurônio Artificial 10

Figura 3.2 Rede Neural Recorrente (RNN) 12

Figura 3.3 Redes Long Short-Term Memory 14

Figura 5.1 Detalhe dos Valores previsto da Série F05 31

Figura 5.2 Detalhe dos Valores previsto da Série F13 33

Figura 5.3 Detalhe dos Valores previsto da Série F19 35

Figura 5.4 Detalhe dos Valores previsto da Série F35 37

Lista de Gráficos

Gráfico 4.1 Série Temporal dos Deslocamentos Horizontais do Bloco F05 . 16

Gráfico 4.2 Série Temporal dos Deslocamentos Horizontais do Bloco F13 . 17

Gráfico 4.3 Série Temporal dos Deslocamentos Horizontais do Bloco F19 . 17

Gráfico 4.4 Série Temporal dos Deslocamentos Horizontais do Bloco F35 . 18

Lista de Tabelas

Tabela 4.1 Modelos ARIMA utilizado para cada Série Temporal. 26

Tabela 5.1 Valores durante o Treinamento Série F05. 30

Tabela 5.2 Valores durante o Teste Série F05. 30

Tabela 5.3 Comparação dos Erros da Série F05. 30

Tabela 5.4 Valores durante o Treinamento Série F13. 31

Tabela 5.5 Valores durante o Teste Série F13. 32

Tabela 5.6 Comparação dos Erros da Série F13. 32

Tabela 5.7 Valores durante o Treinamento Série F19. 33

Tabela 5.8 Valores durante o Teste Série F19. 34

Tabela 5.9 Comparação dos Erros da Série F19. 34

Tabela 5.10 Valores durante o Treinamento Série F35. 35

Tabela 5.11 Valores durante o Teste Série F35. 36

Tabela 5.12 Comparação dos Erros da Série F35. 36

Sumário

1 Introdução **1**

1.1 Objetivos 2

1.1.1 Objetivo Geral 2

1.1.2 Objetivos Específicos 2

1.2 Justificativa 2

1.3 Estrutura da Monografia 4

2 Revisão Bibliográfica **5**

3 Referencial Teórico **8**

3.1 Série Temporal 8

3.2 Redes Neurais Artificiais 9

3.3 Redes Neurais Recorrentes 11

3.4 Long Short-Term Memory 13

4 Material e Métodos 15

4.1 Materiais 15

4.1.1 Dados 15

4.1.2 ARIMA 18

4.1.3 Long Short-Term Memory 20

4.1.4 Linguagem de Programação e Bibliotecas 22

4.2 Metodologia Aplicada 25

4.2.1 Primeira Etapa 25

4.2.2 Segunda Etapa 27

5 Resultados e Discussões 29

6 Conclusão 38

Referências Bibliográficas 39

A Apêndice 42

A.1 Código Fonte 42

Capítulo 1

Introdução

Nesta era da informação, as empresas buscam cada vez mais maneiras de melhorar o seu desempenho buscando alcançar ótima qualidade para o seu produto e obter um menor custo. Para alcançar o objetivo armazenam dados e conhecimentos sobre os fatores (variáveis) que influenciam o seu serviço ou produto.

O histórico de dados armazenados nos computadores das empresas é a base para aplicação de distintas metodologias nas séries temporais. Essas metodologias de aplicação são: Box & Jenkins, Redes Neurais Artificiais, Algoritmos Genéticos entre outros e têm o objetivo de fazer previsões.

É cada vez mais comum utilizar uma ou mais destas metodologias para a previsão de séries temporais, pois possibilita a tomada de decisão com base nas informações obtidas dos dados e, consequentemente, previsões muito perto da realidade. Dentro desse contexto, a barragem da Usina Hidrelétrica de Itaipu, processa muitos dados do sistema de monitoramento da barragem por meio de distintos instrumentos de medição. Esses dados são de vital importância para a segurança estrutural e operacional da barragem.

1.1 Objetivos

1.1.1 Objetivo Geral

Aos resíduos de uma série temporal modelada pela Metodologia Box & Jenkins é aplicada Redes Neurais Recorrentes para a obtenção de previsões mais acertadas, obtendo-se um modelo composto por um modelo ARIMA e Redes Neurais Recorrentes.

1.1.2 Objetivos Específicos

- Aplicar a metodologia Box & Jenkins à série temporal de dados de monitoramento para modelar a parte linear da variação dos dados.
- Aplicar Redes Neurais Recorrentes, em especial, as redes LSTM nos resíduos da série temporal.
- Codificar na linguagem de programação Python usando as bibliotecas Theano e Scikit learn.
- Gerar previsões pontuais.
- Comparar os resultados obtidos usando Redes Neurais Recorrentes com os obtidos através da aplicação da Metodologia Box & Jenkins.

1.2 Justificativa

O projeto ITAIPU foi elaborado em conformidade com os melhores padrões de segurança vigentes na época, tendo-se elaborado critérios de projeto, especificações e diretrizes básicas, tanto para o projeto e construção, quanto para operação e manutenção, os quais continuam sendo obedecidos e atualizados quando necessário,

visando garantir o desempenho seguro tanto da barragens (estruturas e fundações), quanto das instalações e equipamentos (BETIOLI et al., 1999).

Incondicionalmente, as barragens são obras que requerem atenção às diversas condições de segurança estrutural e operacional, a fim de evitar algum dano em suas estruturas ou até mesmo ao meio ambiente ou perdas de vida humanas. A identificação de problemas e execução de reparos diminui os fatores de risco. Porém, esses estudos requerem muita atenção para a determinação de soluções adequadas. Tendo isso como base, a usina de Itaipu foi concebida com um plano de auscultação que conta com mais de 2400 instrumentos de monitoramento e um sistema de drenagem com mais de 5000 drenos, instalados em suas estruturas e fundações, formando com os registros do monitoramento um banco de dados de mais de 30 anos.

De acordo com Itaipu (2009 *apud* (CORRÊA et al., 2016)), foram instalados dois tipos de pêndulos em trechos da Usina Hidrelétrica de Itaipu: o pêndulo direto e o invertido. O pêndulo direto é responsável em monitorar os deslocamentos horizontais de pontos dos blocos instrumentados da barragem em determinadas cotas, em relação à fundação da estrutura. E, a função do pêndulo invertido é monitorar os deslocamentos da fundação da barragem em relação ao contato concreto/rocha ou a alguma cota determinada no bloco de concreto.

As informações pertinentes ao comportamento estrutural de uma barragem de uma usina hidrelétrica, por exemplo, são avaliadas por meio das leituras e análise dos dados da instrumentação (CORRÊA et al., 2016).

Nesse contexto, a previsão de valores das séries temporais dos deslocamentos horizontais pode beneficiar aos engenheiros e à equipe técnica para adotar medidas corretivas ou tomada de decisões, sendo aplicada a metodologia mista de ARIMA / Redes Neurais Recorrentes para realizar as previsões o mais perto possível dos valores reais da série.

1.3 Estrutura da Monografia

O presente trabalho está composto por seis capítulos: (1) Introdução; (2) Revisão Bibliográfica; (3) Referencial Teórico; (4) Materiais e Métodos (5) Resultados e Discussão; (6) Conclusão e Considerações Finais, finalizando com as Referências Bibliográficas.

No capítulo 1 apresenta-se um breve resumo do trabalho, a justificativa e os objetivos propostos, tanto o geral como os específicos. No capítulo 2 são apresentados resumos e conclusões de artigos relacionados ao trabalho. No capítulo 3 são explanados os conceitos e descrições de técnicas usadas no trabalho, tais como: Modelos de Box e Jenkins, Redes Neurais Artificiais, Redes Neurais Recorrentes, Long Short-Term Memory (LSTM). A seguir, no capítulo 4, são apresentados o material (séries temporais utilizadas) e a descrição da metodologia proposta. Já no capítulo 5 são apresentados e discutidos os resultados obtidos utilizando algumas das técnicas presentes no método descrito no capítulo 3. No capítulo 6 estão as Considerações Finais e a Conclusão. Finalmente, as Referências na última parte do trabalho.

Capítulo 2

Revisão Bibliográfica

A seguir alguns trabalhos relacionados com Séries Temporais e Redes Neurais Artificiais que serão discutidos no desenvolvimento do projeto. Cada um deles mostra um resumo do problema abordado e a maneira como foi resolvido.

No trabalho de Dantas et al. (2016) foi apresentado o objetivo de fazer a previsão da precipitação de chuva na área de Diamantina (MG), pois muitos produtores agrícolas utilizam a precipitação mensal como meio de irrigação e também como fator para o planejamento de seus projetos agrícolas. Nesse contexto, é necessário fazer o cálculo de previsão de uma maneira mais perto da realidade, para isso, foi utilizada a técnica de Redes Neurais Artificiais utilizando dados de precipitação diária desde 1977 até 2014. Foram feitos ajustes cronológicos de períodos de chuva e períodos de seca, existindo uma relação entre esses períodos. A Rede Neural Artificial foi treinada com uma parte dos dados, e logo após, gerou a previsão com os dados restantes. A utilização de Redes Neurais Artificiais apresentou resultados satisfatórios, apesar da complexidade do problema.

No trabalho de Figueredo (2008) o objetivo foi comparar a metodologia de Box & Jenkins e a técnica de Redes Neurais Artificiais, na proposta de identificar aquela que é a mais adequada para fazer a previsão de valores futuros de demanda de

produtos de uma empresa na região de Joinville. Com a globalização no mundo dos negócios, as empresas necessitam cada vez mais de uma política de produção que vise o menor custo e uma maior produtividade, por meio de processos contínuos de melhoria dos produtos e processos nas várias etapas de produção. Como resultado final, determinou-se que a técnica de computação conhecida como Redes Neurais é a mais adequada para a obtenção desses valores, por apresentar os menores valores da raiz do erro quadrático médio (RMSE), calculado sobre os valores previstos.

Segundo Souza (2008) foi apresentado o uso do Filtro de Kalman Estendido (EFK em inglês) para a minimização do erro de treinamento das Redes Neurais de Função de Base Radial na previsão de Séries Temporais financeiras de commodities do agronegócio, e a utilização da Evolução Diferencial para determinação dos parâmetros de sintonia do EFK. Desta maneira a aplicação combinada destas ferramentas para a modelagem e previsão de sistemas com comportamento não-linear possa produzir resultados satisfatórios para a previsão de séries temporais. Uma abordagem é proposta combinando a RN-RBF, uma adaptação do KF e a ED para a previsão de séries temporais. Os resultados desta abordagem são comparados, portanto, com os de outras abordagens baseadas em modelos lineares e não-lineares.

Os resultados desse algoritmo mostraram-se superiores em termos de MSE à otimização por AG com representação binária (algoritmo de computação evolucionária mais comumente utilizado), ao treinamento da RN-RBF por descida de encosta (algoritmo frequentemente utilizado para treinamento destas redes), e ao ARIMA (método linear mais popular na previsão de séries temporais) utilizando-se o mesmo critério de desempenho em termos do MSE.

No trabalho de Batista (2011) foi investigado a utilização de duas metodologias para realizar previsões de Séries Temporais de vazões fluviais. As metodologias utilizadas foram de Box & Jenkins e a de Redes Neurais Artificiais. A Série Temporal escolhida foi a das razões naturais do Rio Grande, coletadas na seção de controle no posto fluviométrico de Madre de Deus de Minas, MG. Posteriormente

foi feita uma análise comparativa entre ambas as técnicas e os resultados obtidos mostram que cada metodologia pode ser ajustada adequadamente ao conjunto de observações em estudo, entretanto cada técnica possui vantagens e desvantagens.

Capítulo 3

Referencial Teórico

3.1 Série Temporal

Uma definição de Série Temporal indica que é um conjunto de observações ordenadas no tempo (não necessariamente igualmente espaçadas), e que apresentam dependência serial (isto é, dependência entre os valores nos instantes de tempo). A notação usada para denotar uma Série Temporal é $Z_1, Z_2, Z_3, ..., Z_t$, que indica uma série de tamanho T. O instante T geralmente indica o último instante disponível.

De uma maneira um pouco mais formal, diz-se que uma Série Temporal é uma realização de um processo estocástico (PAULO, 2018).

Pode-se enumerar os seguintes exemplos de séries temporais: temperaturas máxima e mínima diárias em uma cidade, vendas mensais de uma empresa, valores mensais do IPC-A, valores de fechamento diários do IBOVESPA, resultado de um eletroencefalograma, gráfico de controle de um processo produtivo, entre outras.

A suposição básica que norteia a análise de séries temporais é que há um processo casual que pode ser estacionário, relacionado com o tempo, que exerce influência sobre os dados no passado e pode continuar a fazê-lo no futuro. Este

processo casual costuma atuar criando um padrão não aleatório e que pode ser observado em um gráfico da série temporal ou mediante algum outro processo estatístico (REIS, 2018).

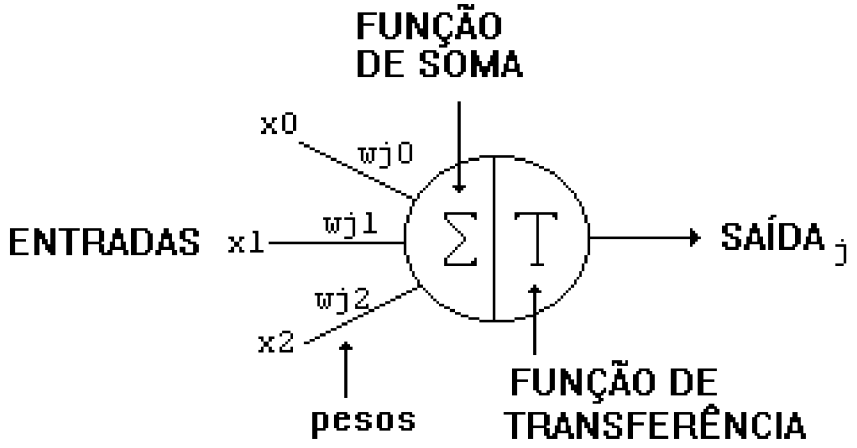
3.2 Redes Neurais Artificiais

A ideia do funcionamento das Redes Neurais Artificiais (RNA) consiste em capturar os princípios básicos de manipulação de informação do cérebro humano e aplicar esse conhecimento na resolução de problemas que exigem aprendizado a partir da experiência (BEALE apud MUELLER, 1996 *apud* (FIGUEREDO, 2008)).

As redes neurais possuem um funcionamento similar ao do cérebro humano, tentando reconhecer padrões e regularidades nos dados apresentados e, "são capazes de aprender com a experiência e fazer generalizações baseadas no seu conhecimento previamente acumulado" (FERNANDES, 1996 p.253, *apud* (FIGUEREDO, 2008)).

O modelo proposto para um neurônio artificial por McCulloch e Pitts se baseia no funcionamento de um neurônio natural. A descrição do modelo é ilustrada na Figura 3.1.

Figura 3.1 –Modelo de um Neurônio Artificial



FONTE: (FIGUEREDO, 2008)

O modelo possui várias entradas (x_i) com um peso associado a cada entrada (w_{ji}), alguns pesos com sinais excitatórios (+) e outros com sinais inibitórios (-). Os valores de entrada e ativação das células podem ser discretos, nos conjuntos 0, 1 ou -1, 0, 1 ou contínuos nos intervalos [0,1] ou [-1,1] (FIGUEREDO, 2008).

Uma Rede Neural Artificial é composta por várias unidades de processamento, cujo funcionamento é bastante simples. Essas unidades, geralmente são conectadas por canais de comunicação que estão associados a determinado peso. As unidades fazem operações apenas sobre seus dados locais, que são entradas recebidas pelas suas conexões. O comportamento inteligente de uma Rede Neural Artificial vem das interações entre as unidades de processamento da rede.

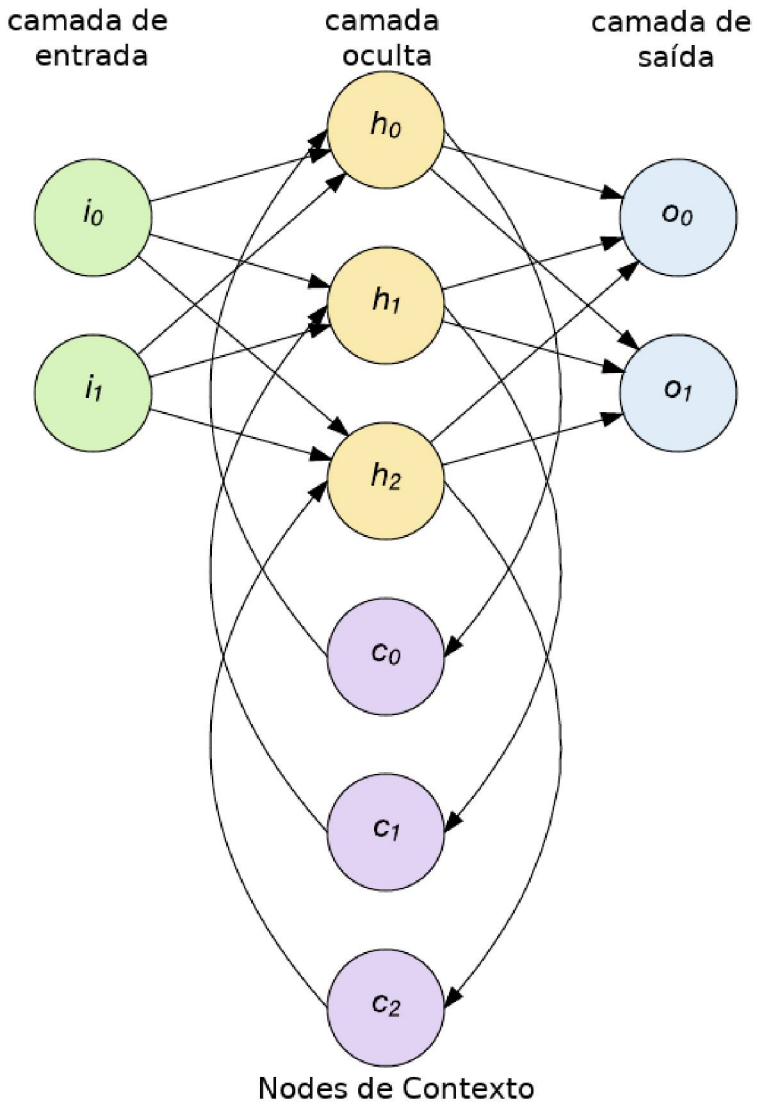
A propriedade mais importante das redes neurais é a habilidade de aprender de seu ambiente e com isso melhorar seu desempenho. Isso é feito através de um processo iterativo de ajustes aplicado a seus pesos, o treinamento. O aprendizado ocorre quando a rede neural atinge uma solução generalizada para uma classe de problemas (CARVALHO, 2009).

3.3 Redes Neurais Recorrentes

A Rede Neural Recorrente (RNN) é uma classe de redes neurais artificiais que inclui conexões ponderadas dentro de uma camada (em comparação com as redes de *feed-forward* tradicionais, onde conecta alimentação apenas para camadas subsequentes). Como as RNNs incluem *loops*, elas podem armazenar informações ao processar novas entradas. Essa memória os torna ideais para tarefas de processamento onde as entradas anteriores devem ser consideradas (como dados da série temporal). Por esta razão, as redes atuais são baseadas em RNNs.

As aplicações de dados de séries temporais exigem um novo tipo de topologia que possa considerar o histórico da entrada. Uma RNN inclui a capacidade de manter a memória interna com *feedback* e, portanto, suportar o comportamento temporal. Na Figura 3.2, a saída da camada oculta é aplicada de volta para a camada oculta. A rede permanece *feed-forward* (as entradas são aplicadas na camada oculta e, em seguida, na camada de saída), mas o RNN mantém o estado interno através dos nodes de contexto (que influenciam a camada oculta nas entradas subsequentes) (JONES, 2017).

Figura 3.2 –Rede Neural Recorrente (RNN)



FONTE: (JONES, 2017)

Um modelo de sequência de rede neural é comumente projetado para transformar sequências de entrada em sequências de saída, que vivem em um domínio diferente. Outro tipo comum de redes neurais chamadas RNN ou redes neurais

recorrentes são muito adequadas para esses propósitos, pois mostraram uma melhoria surpreendente em problemas como reconhecimento de fala, reconhecimento de manuscrito e tradução automática. Um modelo RNN nasce com uma incrível capacidade de processar dados sequenciais longos e lidar com tarefas muito complexas com o contexto distribuído ao longo de um período. O modelo recorrente, de fato, processa um único elemento na sequência neural da época. Após o cálculo inicial, esse estado de unidade recém-atualizado é facilmente transmitido para o próximo passo de tempo para facilitar o cálculo de cada próximo elemento (MILLSTEIN, 2018).

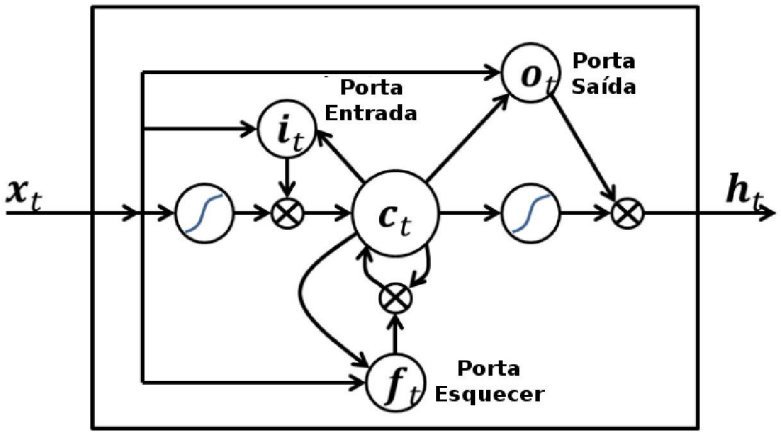
3.4 Long Short-Term Memory

Redes *Long Short-Term Memory* (LSTM) (Figura 3.3), sâo um modelo recorrente e profundo de redes neurais. LSTM foram introduzidas por Hochreiter & Schmidhuber (1997, *apud* (NELSON, 2017)) e sua motivação foi oferecer um desempenho melhor para resolver o problema de desaparecimento de gradiente que redes neurais recorrentes naturalmente sofrem quando lidam com grandes sequências de dados. As redes LSTM o fazem ao manter o fluxo de erro constante através de unidades especiais chamadas "portões" (*gates*), que permitem ajustes de pesos da mesma forma que o truncamento da sequência quando a informação não é necessária, simbolizando um esquecimento.

Esse tipo de rede tem sido amplamente usado e sido capaz de alcançar alguns dos melhores resultados quando colocados em comparação com outros métodos, Graves(2012, *apud* (NELSON, 2017)). Este fato é observado especialmente no campo de Processamento de Linguagem Natural, e em reconhecimento de caligrafia é considerada o estado-da-arte, Graves et al.(2009, *apud* (NELSON, 2017)). Desde a sua concepção, este método foi ramificado em diversas variações. Entretanto, quando avaliadas em relação à original por Greff et al. (2015, *apud* (NELSON, 2017)), nenhuma foi capaz de apresentar melhoria considerável em termos

de resultados (NELSON, 2017).

Figura 3.3 –Redes Long Short-Term Memory [Greff et. Al. 2015 apud(FIGUEREDO, 2008)]



FONTE: (NELSON, 2017)

Os modelos LTSM são inteligentes o suficiente para aprender o contexto de longo prazo. Esses modelos podem aprender por quanto tempo devem memorizar as informações antigas, quando esquecer informações, quando usar dados recém-atualizados e quando combinar a nova entrada com a memória antiga. Usando o poder das células LSTM e RNN, pode-se construir um modelo baseado em caracteres RNN que será capaz de aprender a relação específica entre os caracteres para formar palavras e sentenças sem qualquer conhecimento prévio do vocabulário em inglês. Esse modelo da RNN, na verdade, poderia alcançar um desempenho muito bom, mesmo sem um grande conjunto de dados de treinamento (MILLSTEIN, 2018).

Capítulo 4

Material e Métodos

4.1 Materiais

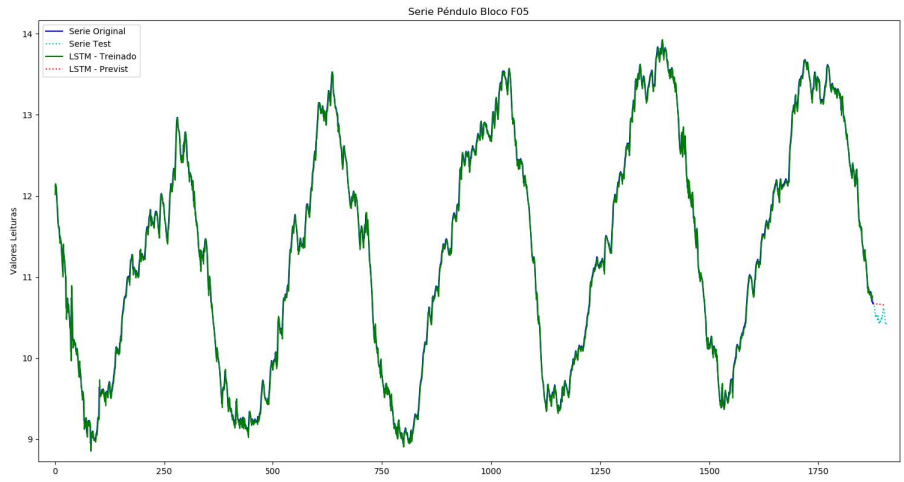
4.1.1 Dados

Os dados que foram utilizados são de Corrêa (2015), que faz referência aos deslocamentos horizontais de blocos da barragem principal da Usina Hidrelétrica de Itaipu, aferidas pelos pêndulos diretos automatizados. A série temporal mostra que os deslocamentos descrevem um movimento que gera uma curva cujo comportamento é cíclico, influenciados pela temperatura ambiente.

Os dados foram obtidos pelo sistema ADAS (Automatic Data Acquisition System) que Itaipu dispõe desde 2005. Para a presente análise utilizam-se as leituras automatizadas dos sensores do pêndulo direto que aferem os deslocamentos horizontais relativos aos blocos F05, F13, F19 e F35, na direção do fluxo (x). O período de dados disponível para a análise foi do dia 28 de outubro de 2005 a 21 de fevereiro de 2011, totalizando 1909 observações para cada bloco, com periodicidade de uma leitura por dia as 10:00am. Nos Gráficos 4.1, 4.2, 4.3 e 4.4 apresenta-se o comportamento das séries temporais dos blocos F05, F13, F19 e F35, respectivamente.

Os deslocamentos tem como unidade de medida o milímetro.

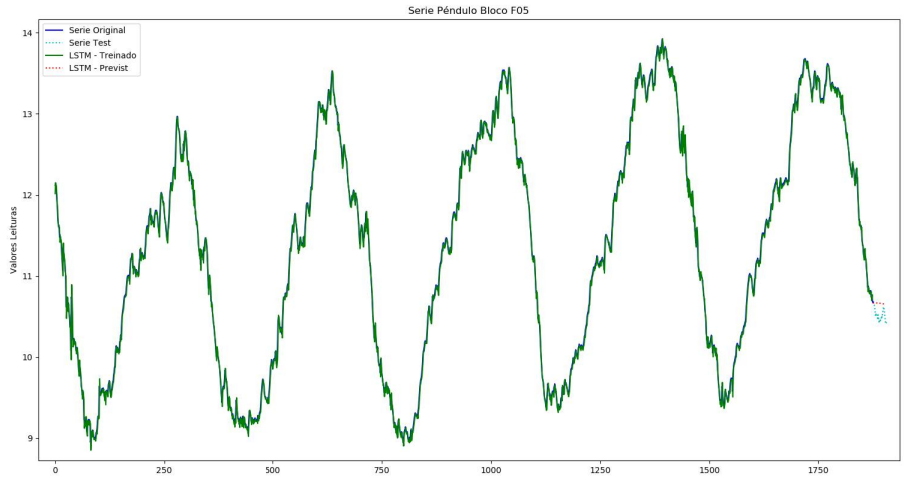
Gráfico 4.1 –Série Temporal dos Deslocamentos Horizontais do Bloco F05



Fonte: O autor (2018)

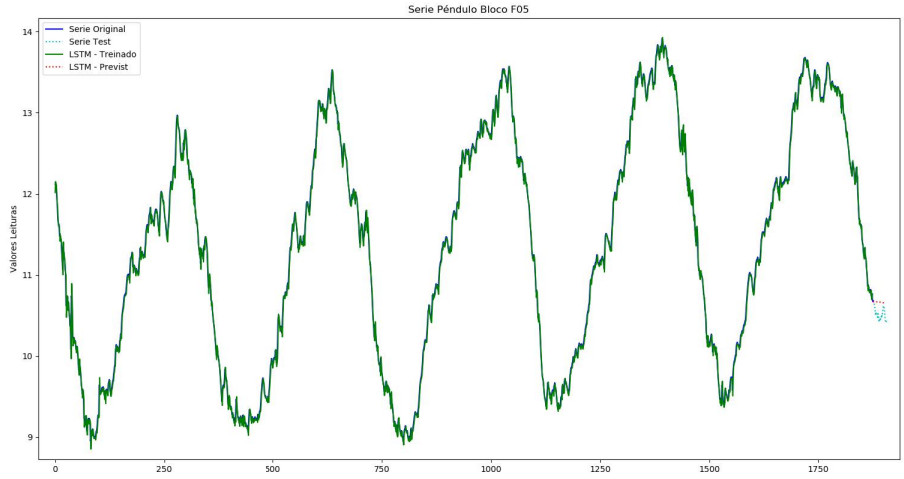
Pode-se observar que no Gráfico 4.1 apresenta uma sazonalidade, devido a natureza da mesma, ja que os deslocamentos se mantêm estáveis ao longo dos anos pela ação de manutenção e controle dos instrumentos e da barragem.

Gráfico 4.2 –Série Temporal dos Deslocamentos Horizontais do Bloco F13



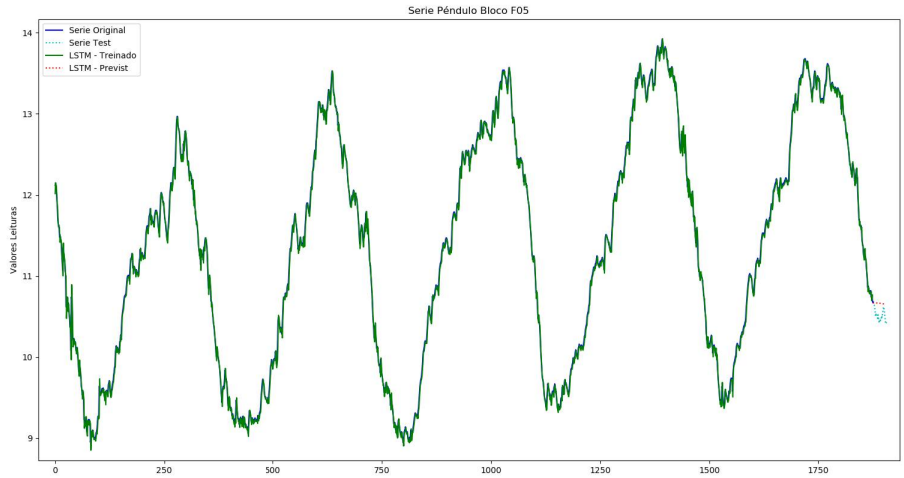
Fonte: O autor (2018)

Gráfico 4.3 –Série Temporal dos Deslocamentos Horizontais do Bloco F19



Fonte: O autor (2018)

Gráfico 4.4 –Série Temporal dos Deslocamentos Horizontais do Bloco F35



Fonte: O autor (2018)

Assim como o Gráfico 4.1, nos Gráficos 4.2, 4.3 e 4.4, pode-se observar a presença de sazonalidade. Também se pode observar que as Séries Temporais são não estacionarias.

4.1.2 ARIMA

Segundo Box e Jenkins (1970 apud (TEIXEIRA et al., 2016)) um modelo ARMA plausível para a série temporal $\{y_t\}_{t=1}^{T}$, com cardinalidade T, de ordem p (autorregressivo - AR) e q (médias móveis - MA) é descrito pela equação 4.1.

$$y_t = \delta + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} - \theta_1 a_{t-1} - \theta_2 a_{t-2} - ... - \theta_q a_{t-q} + a_t \quad (4.1)$$

O modelo da equação 4.1 combina valores passados das entradas y_t e choques aleatórios (a_t) descorrelacionados, de média zero e variância constante. Neste modelo $\phi_i, \theta_j \in \mathrm{R}$, com $i = 1, ..., p$ e $j = 1, ..., q$, denotam os parâmetros do modelo e δ uma constante. Na hipótese da série temporal ser não estacionária, esta deve ser diferenciada e o modelo ARMA(p,q) substituído pelo ARIMA(p,d,q), sendo d a ordem de diferenciação da série. Para a identificação do modelo, as ordens p e q podem ser determinadas por meio da análise do perfil dos gráficos das funções de autocorrelação (FAC) e autocorrelação parcial (FACP) (HAMILTON 1994 apud (TEIXEIRA et al., 2016)).

Identificado o modelo, passa-se ao estágio seguinte que é a estimação dos parâmetros. Para tanto, é necessário utilizar métodos iterativos não lineares de mínimos quadrados. Maiores detalhes destas aplicações podem ser encontrados em Box e Jenkins (1970 apud (TEIXEIRA et al., 2016)) e Morettin e Toloi (2006 apud (TEIXEIRA et al., 2016)). Para a validação do modelo já com os parâmetros estimados pode-se usar testes estatísticos, tais como: teste de Box-Pierce, teste do periodograma acumulado, teste da autocorrelação residual, entre outros. No caso da série temporal $\{y_t\}_{t=1}^{T}$ apresentar componente sazonal, o modelo de Box & Jenkins plausível é o ARIMA multiplicativo, dado genericamente na equação 4.2.

$$\phi(B)(1-\Phi_1 B-...-\Phi_p B^{PS})\bigtriangledown^d(1-B^S)^D y_t = \theta(B)(1-\Theta_1 B-...-\Theta_Q B^{QS})a_t \quad (4.2)$$

onde: $\phi(B) = (1 - \phi_1 B - ... - \phi_p B^p)$,$\theta(B) = (1 - \theta_1 B - ... - \theta_q B^q)$,$d$ é a ordem das diferenças simples, D é a ordem das diferenças sazonais, S é o período sazonal, $\phi_k \in \mathrm{R}$ e $\theta_j \in \mathrm{R}$ são os coeficientes dos polinômios não sazonais e $\Phi_m \in \mathrm{R}$ e $\Theta_n \in \mathrm{R}$ são os coeficientes dos polinômios sazonais.

A fim de validar o modelo ajustado foram analisados os resíduos com a construção dos gráficos das funções FAC e FACP, buscando mostrar que os mesmos são não autocorrelacionados.

Para testar a hipótese que todos os coeficientes de autocorrelação p_k sejam iguais a zero, foi utilizada a estatística $Q^* = n\sum_{k=1}^{m} \hat{p}_k^2$, desenvolvida por Box e Pierce, em que n é o tamanho da amostra e m a duração da defasagem. A estatística Q^* tem distribuição qui-quadrado com m graus de liberdade. Quando Q^* excede o valor crítico, rejeita-se a hipótese nula de que todos os p_k são iguais a zero (GUJARATI, 2000 apud (TEIXEIRA et al., 2016)).

4.1.3 Long Short-Term Memory

Em uma LSTM, a camada que se repete ao longo do tempo possui três módulos que trabalham em conjunto para gerar o novo estado oculto, em vez de gerá-la em uma única etapa. Cada um desses módulos tem uma função especial no processo de decidir quais informações devem ser mantidas, quais devem ser transmitidas para as seguintes etapas e quais informações devem ser descartadas, utilizando, para tal, informações usuais provenientes da entrada, da saída das camadas ocultas anteriores e da chamada Célula de Memória. Esses módulos são chamados de portas (*gates*). Um gate é composto por uma função de ativação sigmóide e uma operação de multiplicação elemento a elemento (operação também conhecida de forma geral como produto de Hadamard). Além dos gates, as LSTMs também empregam uma camada oculta que recebe a concatenação da entrada atual com o estado oculto anterior, assim como as redes neurais recorrentes originais.

O primeiro *gate*, o *forget gate*, tem a responsabilidade de decidir quais elementos da célula de memória serão jogadas fora e quais poderão continuar. Ele realiza essa tarefa produzindo um vetor de valores entre zero e um e multiplicando a memória por esses valores, de forma que aqueles valores multiplicados por números próximos de zero são esquecidos, e aqueles multiplicados por valores próximos a um são mantidos. Essa tarefa é realizada, a cada passo, a partir do estado oculto anterior da rede e da entrada atual.

A equação do *forget gate* é dada, portanto, por 4.3:

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \tag{4.3}$$

Em que $[h_{t-1}, x_t]$ é a concatenação do estado oculto do passo anterior e a entrada atual, e substitui a entrada x na transformação afim usada para a *Multilayer Perceptron* (MLP). W_f e b_f são os parâmetros a serem aprendidos para o *forget gate*, e σ a função de ativação logística que "achata"os valores para $[0, 1]$.

Depois de esquecer dados julgados irrelevantes, a rede então gera um novo estado para a memória, processo conhecido como criação do candidato $\hat{C}_t$ da memória, usando uma função afim e uma função de ativação da mesma forma que a RNN tradicional conforme a equação 4.4, que utiliza tipicamente a tangente hiperbólica (*tanh*) como ativação:

$$\hat{C}_t = tanh(W_C.[h_{t-1}, x_t] + b_C) \tag{4.4}$$

Em que W_C e b_C são os pesos e o viés utilizados para gerar o novo candidato a memória $\hat{C}_t$. Tendo, então, um novo candidato a memória, entra em funcionamento o segundo *gate*, o *input gate*, equação 4.5:

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i) \tag{4.5}$$

Este, por sua vez, também possui seu conjunto de parâmetros W_i e b_i, e também gera um vetor de valores entre [0,1], só que o mesmo será multiplicado ao candidato da nova memória $\hat{C}_t$, de forma a selecionar quais campos vão ser guardados na memória e quais serão imediatamente esquecidos. Esse novo candidato a memória é então somado à célula de memória após ter tido campos apagados pelo *forget gate*, segundo a equação 4.6:

$$C_t = f_t * C_{t-1} + i_t * \hat{C}_t \tag{4.6}$$

Em que o primeiro operando $f_t * C_{t-1}$ representa o processo de esquecimento de dados não mais pertinentes da memória e o segundo operando $i_t * \hat{C}_t$ a introdução de novos valores na memória. Com isso gera-se a memória do passo t, C_t.

Finalmente, constrói-se o novo estado oculto e a nova saída, que são o mesmo vetor para a LSTM, e para tanto usamos o terceiro *gate*, o *output gate*, dado pela equação 4.7

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o) \tag{4.7}$$

Em que W_o e b_o são mais um conjunto independente de pesos e viés para o modelo. O vetor o_t é então multiplicado pela memória atual para gerar a saída e estado oculto h_t, selecionando quais informações na memória devem ser usados para gerar a saída desse passo, conforme a equação 4.8

$$h_t = o_t * tanh(C_t) \tag{4.8}$$

O resultado de todo esse processo é um novo valor para a célula de memória e um novo estado oculto, ambos podendo ser usados pelo próximo passo da rede recorrente juntamente com a nova entrada, e uma saída que pode ser usada diretamente para o erro ou predição, ou ligada a uma nova camada para realizar mais processamentos (CARREGOSA, 2018).

4.1.4 Linguagem de Programação e Bibliotecas

A seguir apresentam-se a linguagem de programação e as bibliotecas utilizadas neste trabalho.

Python

Python é uma linguagem de programação criada por Guido van Rossum em 1991. Os objetivos do projeto da linguagem eram: produtividade e legibilidade. Em outras palavras, Python é uma linguagem que foi criada para produzir código bom e fácil de manter de maneira rápida. Entre as características da linguagem que ressaltam esses objetivos estão:

- Baixo uso de caracteres especiais, o que torna a linguagem muito parecida com pseudo-código executável;
- O uso de identação para marcar blocos;
- Quase nenhum uso de palavras-chave voltadas para a compilação;
- Coletor de lixo para gerenciar automaticamente o uso da memória;

Além disso, Python suporta múltiplos paradigmas de programação. A programação procedimental pode ser usada para programas simples e rápidos, mas estruturas de dados complexas, como tuplas, listas e dicionários, estão disponíveis para facilitar o desenvolvimento de algoritmos complexos. Grandes projetos podem ser feitos usando técnicas de orientação a objetos, que é completamente suportada em Python (inclusive sobrecarga de operadores e herança múltipla). Um suporte modesto para programação funcional existe, o que torna a linguagem extremamente expressiva: é fácil fazer muita coisa com poucas linhas de comando. Também possui inúmeras capacidades de meta-programação: técnicas simples para alterar o comportamento da linguagem, permitindo a criação de linguagens de domínio específico (PYSCIENCE, 2018).

Theano

O Theano é uma biblioteca Python que permite definir, otimizar e avaliar expressões matemáticas, especialmente aquelas com matrizes multidimensionais (numpy. ndarray). Usando Theano é possível obter velocidades que rivalizam com

implementações C feitas à mão para problemas envolvendo grandes quantidades de dados. Ele também pode superar C em uma CPU em muitas ordens de magnitude, aproveitando as GPUs recentes.

Theano combina aspectos de um sistema de álgebra computacional (CAS) com aspectos de um compilador otimizador. Também pode gerar código C customizado para muitas operações matemáticas. Essa combinação de CAS com otimização de compilação é particularmente útil para tarefas em que expressões matemáticas complicadas são avaliadas repetidamente e a velocidade de avaliação é crítica. Para situações em que muitas expressões diferentes são avaliadas, uma vez que Theano pode minimizar a quantidade de sobrecarga de compilação / análise, mas ainda fornece recursos simbólicos, como a diferenciação automática (THEANO, 2018).

Keras

Keras é uma Interface de Programação de Aplicação (API em inglês) de alto nível redes neurais, escrito em Python e capaz de correr em cima de TensorFlow , CNTK , ou Theano. Foi desenvolvido com foco em permitir a experimentação rápida. Ser capaz de ir da ideia ao resultado com o menor atraso possível é a chave para fazer uma boa pesquisa.

Princípios orientadores:

- Facilidade de utilização. Keras é uma API projetada para seres humanos, não para máquinas. Coloca a experiência do usuário na frente e no centro. A Keras segue as práticas recomendadas para reduzir a carga cognitiva: ela oferece APIs consistentes e simples, minimiza o número de ações do usuário necessárias para casos de uso comuns e fornece um *feedback* claro e acionável sobre o erro do usuário.
- Modularidade. Um modelo é entendido como uma sequência ou um gráfico de módulos autônomos configuráveis que podem ser conectados com o menor

número de restrições possível. Em particular, camadas neurais, funções de custo, otimizadores, esquemas de inicialização, funções de ativação, esquemas de regularização são todos módulos independentes que podem ser combinados para criar novos modelos.

- Extensibilidade fácil. Novos módulos são simples de adicionar (como novas classes e funções), e os módulos existentes fornecem exemplos amplos. Poder criar facilmente novos módulos permite total expressividade, tornando Keras adequado para pesquisa avançada.

- Compatível com a linguagem Python. Não há arquivos de configuração de modelos separados em um formato declarativo. Os modelos são descritos em código Python, que é compacto, mais fácil de depurar e permite a facilidade de extensibilidade (KERAS, 2018).

4.2 Metodologia Aplicada

O trabalho foi dividido em duas etapas:

4.2.1 Primeira Etapa

Na Análise de Séries Temporais, a metodologia que foi utilizada é a Box & Jenkins, escolhendo-se o modelo auto regressivo integrado de média móveis (ARIMA) que melhor se ajustou a cada uma das séries temporais. Na Tabela 4.1 pode-se observar qual modelo ARIMA foi utilizado em cada série.

Tabela 4.1 –Modelos ARIMA utilizado para cada Série Temporal.

Série Temporal	Modelo ARIMA
F05	ARIMA (2, 1, 2)
F13	ARIMA (0, 2, 2)
F19	ARIMA (2, 2, 2)
F35	ARIMA (2, 2, 1)

FONTE: O autor (2018)

Uma vez ajustado o modelo linear do Box & Jenkins para cada série temporal, absorve-se a variação linear e os resíduos do ajuste podem conter parte não linear.

Para a esta etapa do trabalho, foi utilizado o *software Statgraphics Centurion XVI*, na sua versão de avaliação, aplicando o *Forecasting* as séries temporais representando os dados dos deslocamentos dos blocos F05, F13, F19 e F35.

Observou-se que ao realizar o tratamento das séries, dos 1909 dados em cada série, foi possível utilizar 1907 valores, sendo os valores faltantes utilizados pelo algoritmo do *software*, para a previsão da parte linear, tendo por tanto 1907 valores para a análise e para a seguinte etapa do trabalho.

Foram exportados para cada série os seguintes dados:

- Dados originais;
- Previsão do modelo linear;
- Resíduos do modelo não linear.

Os dados foram exportados em um arquivo no formato Excel (xlsx), uma folha para cada série temporal.

4.2.2 Segunda Etapa

Na segunda etapa foi aplicado um algoritmo de Redes Neurais Recorrentes utilizando a rede do tipo *Long Short-Term Memory* (LSTM) aos resíduos para modelar a parte não linear e assim obter valores de previsão dos resíduos.

Para isso, foi desenvolvido um algoritmo na linguagem de programação Python (ver Apêndice A.1), implementando as bibliotecas Theano, Keras, citadas anteriormente na seção III Materiais e Métodos.

O algoritmo realiza a importação dos dados do arquivo Excel, trabalhando com os resíduos da série como dados de entrada do algoritmo, para isso, foi realizada uma normalização dos mesmos, já que o algoritmo LSTM trabalha com função sigmoide e com a *tanh*, explicados na seção III Materiais e Métodos, sendo necessário normalizar os valores dos dados entre 0 e 1.

Após a normalização, os dados são separados em dois grupos: um grupo de treinamento e um grupo de teste, sendo que dos 1907 valores, 1877 são de treinamento e 30 são para testes. As quantidades foram escolhidas de tal maneira que a parte de teste represente a previsão de 30 valores, o que equivale a 30 dias de previsão.

A estrutura da rede LSTM utilizada foi definida com um valor de entrada, quatro blocos LSTM ou células de memória e um valor de saída, e o processo de previsão foi realizado gerando um valor por vez, ou seja, um passo à frente por vez, sendo que para prever o segundo valor do teste, é necessário ter o primeiro valor de teste, e assim por diante, até chegar em trinta valores. O treinamento foi realizado durante 100 épocas.

Durante o treinamento e o teste, o algoritmo utiliza como métrica de avaliação interna a estatística RMSE (Root Mean Square Error), Raiz do Erro Quadrático Médio, e como algoritmo de otimização é utilizado o algoritmo Adam.

Depois de executados os processos de treinamento e teste, os dados processados são revertidos ao formato original, pois ainda estão normalizados entre 0 e 1. A seguir é somado os valores do modelo linear extraídos na primeira etapa e os resultados da rede LSTM. Como critério de desempenho foi utilizado à estatística de RMSE (Root Mean Square Error), ou Raiz do Erro Quadrático Médio, entre os valores reais e os valores gerados pelo algoritmo.

Por último, são gerados os gráficos contendo a série original em duas partes, a de treinamento e a de teste; também os resultados do treinamento do algoritmo do Modelo LSTM e a previsão dos valores do Modelo LSTM.

Para a comparação das previsões feitas pelo Modelo ARIMA e as previsões do Modelo LSTM, aos resultados da segunda etapa, e feita a soma dos resultados da parte linear da aplicação da metodologia Box & Jenkins.

Capítulo 5

Resultados e Discussões

Na fase de treinamento, os dados tiveram que ser adaptados a necessidade do algoritmo, pelo qual dos 1877 dados para treinamento, 1875 forma utilizados, sendo 2 deles excluídos, da mesma maneira ao realizar o teste, dos 30 valores, foram previstos 28. Este fator e muito importante no momento de utilizar uma rede LSTM, pois dependendo da configuração de quantos passos à frente querem se prever, uma parte dos dados serão excluídos.

A modelagem da Rede Neural Recorrente LSTM consistiu em treinar 1875 valores e gerar previsões um passo à frente, mas utilizando os valores anteriores como entrada da rede, isto é, gerando um valor previsto por vez, até os 28 valores gerados, o que permitiu prever valores muito pertos dos reais.

A seguir se observam os valores de treinamento, teste, e comparativo de erro entre modelos para cada uma das séries estudadas.

Para a série do Bloco F05, pode-se observar na Tabela 5.1 que na Fase de Treinamento, os valores do Modelo LSTM ficaram perto dos valores reais, mais não tanto como os do Modelo ARIMA, sendo que o mesmo comportamento se da em todas a séries estudadas, ou seja, na Fase de Treinamento, o modelo LSTM ajusta a sua rede de tal maneira a ter valores similares ao valores reais, mais ainda

não tao perto como os do Modelo ARIMA.

Tabela 5.1 –Valores durante o Treinamento Série F05.

Posição	Valor Real	Modelo ARIMA	Modelo LSTM
3	12,13	12,1695	12,17464
8	11,71	11,8326	11,81903
500	9,93	9,9853	10,01984
1250	11,14	11,124	11,11844
1877	10,67	10,6687	10,63768

FONTE: O autor (2018)

Na Tabela 5.2 pode se observar o distanciamento dos valores do Modelo LSTM com respeito aos valores reais, sendo que o erro, apresentado na Tabela 5.3, indica que o Modelo ARIMA apresenta um desempenho melhor do que o Modelo LSTM.

Tabela 5.2 –Valores durante o Teste Série F05.

Posição	Valor Real	Modelo ARIMA	Modelo LSTM
1880	10,64	10,6868	10,68574
1885	10,52	10,6851	10,72125
1895	10,45	10,6794	10,83738
1907	10,47	10,6726	10,92545

FONTE: O autor (2018)

Tabela 5.3 –Comparação dos Erros da Série F05.

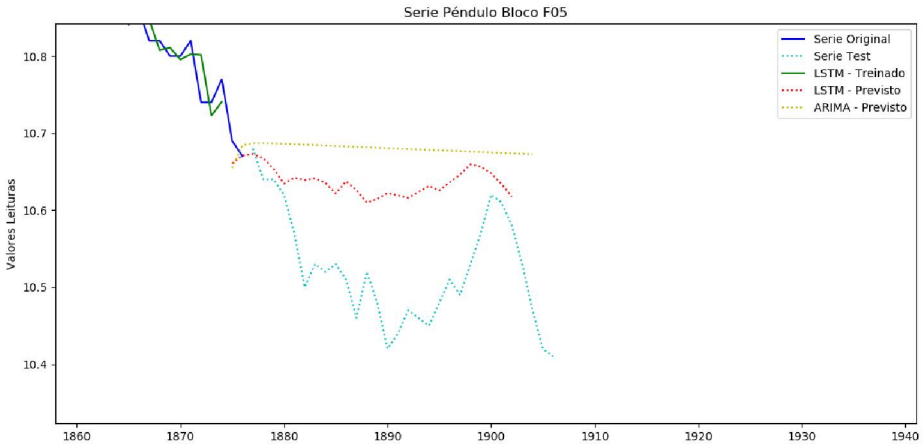
Grupo	RMSE ARIMA	RMSE LSTM
Treinamento	0,063924	0,065703
Teste	0,173239	0,295356

FONTE: O autor (2018)

A pesar desta diferença, na parte gráfica, o Modelo LSTM se aproxima melhor que o Modelo ARIMA, como pode-se observar na Figura 5.1, a série prevista pelo Modelo LSTM, na cor vermelha, apresenta semelhança com a série testada, na cor

azul clara, que a série prevista pelo Modelo ARIMA, na cor amarela, o que sugere um estudo de se isto ocorre por uma peculiaridade da série ou algum outro motivo associado a mesma série temporal.

Figura 5.1 –Detalhe dos Valores previsto da Série F05



FONTE: O autor (2018)

Para a série do Bloco F13, os valores da Fase de Treinamento do Modelo LSTM são próximos aos do Modelo ARIMA, como pode-se observar na Tabela 5.4.

Tabela 5.4 –Valores durante o Treinamento Série F13.

Posição	Valor Real	Modelo ARIMA	Modelo LSTM
3	15,91	15,902	15,90714
8	15,37	15,3722	15,35863
500	13,22	13,3087	13,34319
1250	14,72	14,7179	14,71234
1877	14,43	14,3944	14,36338

FONTE: O autor (2018)

Na Tabela 5.5 observa-se que os valores do Modelo LSTM estão mais apro-

ximados dos valores reais do que os valores do Modelo ARIMA, sendo o mesmo resultado refletido no erro, na Tabela 5.6 os erros do Modelo LSTM foram reduzidos em aproximadamente 40%.

Tabela 5.5 –Valores durante o Teste Série F13.

Posição	Valor Real	Modelo ARIMA	Modelo LSTM
1880	14,34	14,3484	14,4007
1885	14,24	14,2012	14,2534
1895	14,13	13,9068	13,9569
1907	14,06	13,5535	13,5995

FONTE: O autor (2018)

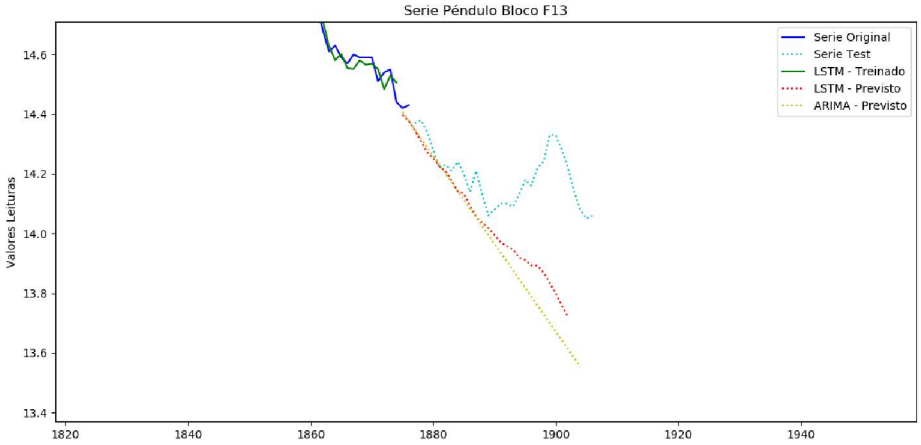
Tabela 5.6 –Comparação dos Erros da Série F13.

Grupo	RMSE ARIMA	RMSE LSTM
Treinamento	0,077260	0,077713
Teste	0,313363	0,187782

FONTE: O autor (2018)

A seguir, a Figura 5.2, pode-se observar a melhora do Modelo LSTM com respeito ao Modelo ARIMA, sendo que a curva apresentada pelo Modelo LSTM (em vermelho) se aproxima melhor da série original (em azul claro), do que a curva do Modelo ARIMA (em amarelo).

Figura 5.2 –Detalhe dos Valores previsto da Série F13



FONTE: O autor (2018)

Para a série do Bloco F19, na Fase de Treinamento do Modelo LSTM também são próximos aos do Modelo ARIMA (Tabela 5.7), apresentado pequenas diferenças entre os dois Modelos e os valores reais.

Tabela 5.7 –Valores durante o Treinamento Série F19.

Posição	Valor Real	Modelo ARIMA	Modelo LSTM
3	16,72	16,7307	16,73584
8	16,1	16,1025	16,08893
500	14,12	14,1574	14,19189
1250	15,31	15,3136	15,30804
1877	14,84	14,7937	14,76268

FONTE: O autor (2018)

Na Tabela 5.8 observa-se que os valores do Modelo LSTM são mais aproximados dos valores reais do que os valores do Modelo ARIMA, especialmente nas últimas posições, sendo o mesmo resultado também refletido no erro, na Tabela 5.9 os erros do Modelo LSTM foram reduzidos em aproximadamente 40% novamente. Desta

maneira sendo a terceira série analisada e a segunda com resultados melhores por parte do Modelo LSTM com respeito ao Modelo ARIMA.

Tabela 5.8 –Valores durante o Teste Série F19.

Posição	Valor Real	Modelo ARIMA	Modelo LSTM
1880	14,75	14,7652	14,76414
1885	14,64	14,6187	14,65485
1895	14,66	14,3182	14,47178
1907	14,49	14,028	14,28065

FONTE: O autor (2018)

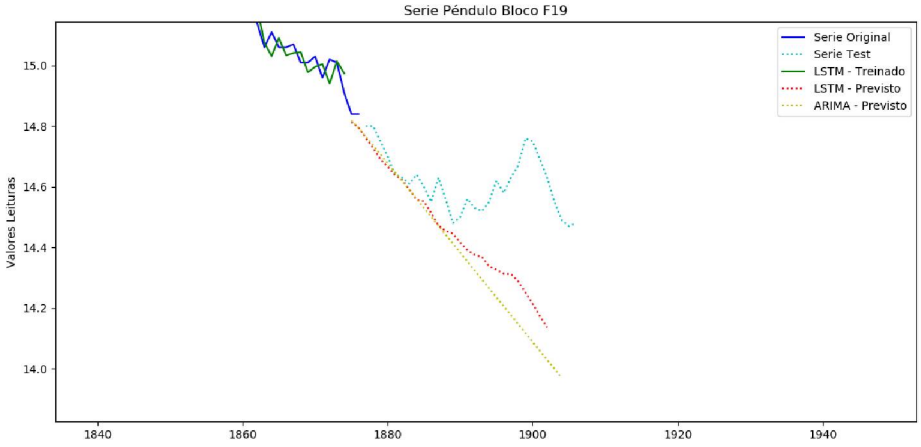
Tabela 5.9 –Comparação dos Erros da Série F19.

Grupo	RMSE ARIMA	RMSE LSTM
Treinamento	0,064132	0,067133
Teste	0,315611	0,189572

FONTE: O autor (2018)

Na Figura 5.3, pode-se observar a melhora do Modelo LSTM com respeito ao Modelo ARIMA, sendo que o Modelo LSTM (em vermelho) se aproxima melhor da série original (em azul claro), do que o Modelo ARIMA (em amarelo).

Figura 5.3 –Detalhe dos Valores previsto da Série F19



FONTE: O autor (2018)

Finalmente, para a série do Bloco F35, os valores da Fase Treinamento do Modelo LSTM novamente são próximos aos do Modelo ARIMA, como pode-se observar na Tabela 5.10, sendo sempre que nesta Fase, o Modelo LSTM ainda aprende com cada nova série, e por tanto, não consegue melhorar os valores do Modelo ARIMA.

Tabela 5.10 –Valores durante o Treinamento Série F35.

Posição	Valor Real	Modelo ARIMA	Modelo LSTM
3	14,54	14,5377	14,54284
8	13,82	13,7804	13,76683
500	12,33	12,3873	12,42179
1250	13,62	13,6038	13,59824
1877	11,19	11,1565	11,12548

FONTE: O autor (2018)

Na Tabela 5.11 observa-se que os valores da Fase de Teste, sendo que o Modelo LSTM apresenta valores mais aproximados dos valores reais do que os valores do

Modelo ARIMA, sendo o mesmo resultado refletido no erro, na Tabela 5.12 os erros do Modelo LSTM foram reduzidos em aproximadamente 19,4%, sendo a quarta série analisada e a terceira com resultados a favor do Modelo LSTM.

Tabela 5.11 –Valores durante o Teste Série F35.

Posição	Valor Real	Modelo ARIMA	Modelo LSTM
1880	11,09	11,0902	11,08914
1885	11,03	10,8749	10,91105
1895	11,1	10,4425	10,61608
1907	11,05	10,01	10,26125

FONTE: O autor (2018)

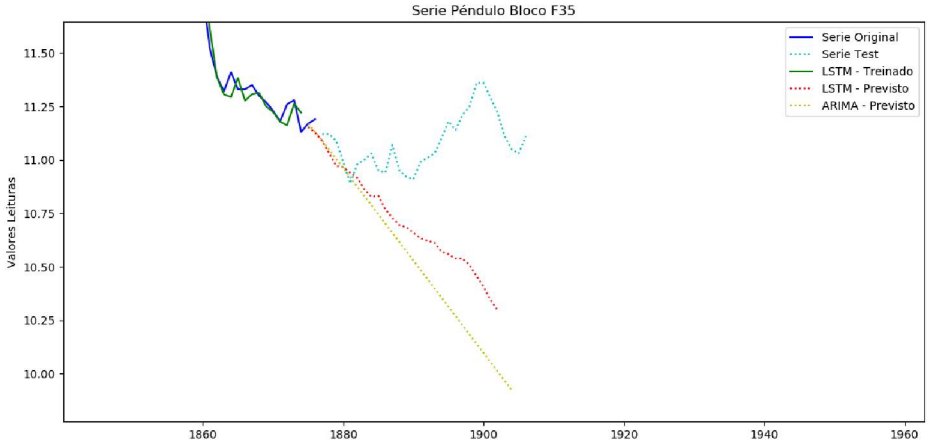
Tabela 5.12 –Comparação dos Erros da Série F35.

Grupo	RMSE ARIMA	RMSE LSTM
Treinamento	0,090288	0,092538
Teste	0,695870	0,560800

FONTE: O autor (2018)

A seguir, a Figura 5.4, pode-se observar a melhora do Modelo LSTM com respeito ao Modelo ARIMA, sendo que a curva apresentada pelo Modelo LSTM (em vermelho) se aproxima melhor da série original (em azul claro), do que a curva do Modelo ARIMA (em amarelo).

Figura 5.4 –Detalhe dos Valores previsto da Série F35



FONTE: O autor (2018)

Como se pode observar nas tabelas, durante a Fase de Treinamento os valores previstos eram próximos aos valores reais mais não quanto os valores previstos pelo modelo ARIMA, mais no caso dos valores da Fase de Teste, a rede LSTM teve a maioria dos valores melhores que o modelo ARIMA, sendo a exceção a primeira série, do Bloco F05, em que por mais perto gráficamente que a série prevista pelo Modelo LSTM estivesse da série original, o Modelo ARIMA teve melhor desempenho e menor erro.

Os resultados foram satisfatórios, é possível notar que os erros com o Modelo LSTM ficam mais estáveis quando iniciada a Fase de Teste, Já no Modelo ARIMA, os erros aumentam consideravelmente na Fase de Teste. Isso representa melhor o poder de previsão do Modelo LSTM, apesar do melhor ajuste do Modelo ARIMA.

Capítulo 6

Conclusão

Neste trabalho, foi aplicada a metodologia Box & Jenkins em quatro Séries Temporais, criando um Modelo ARIMA, logo foi aplicada uma Rede Neural Recorrente (RNN) do tipo LSTM nos resíduos gerados pela metodologia Box & Jenkins, com a linguagem de programação Python e as bibliotecas Theano e Scikit-learn, foram geradas previsões de 28 valores no total, equivalente a 28 días, finalizando com a comparação destes valores gerados com os valores da metodologia Box & Jenkins utilizada.

Como critério de desempenho foi utilizada a estatística de RMSE (Root Mean Square Error), ou Raiz do Erro Quadrático Medio, dos valores gerados pela metodologia Box & Jenkins e pela rede LSTM, sendo um critério muito utilizado no ambiente científico.

A partir da aplicação da metodologia de Redes Neurais Recorrentes LSTM, conclui-se que as mesmas se mostram eficientes na tarefa de previsão devido a que os erros foram menores em três dos quatro casos estudados, mostrando uma redução de até 40% do erro previsto em duas delas, e quase 20% na terceira, ou seja, valores mais perto dos reais.

Referências Bibliográficas

BATISTA, A. L. F. Análises e previsões de vazões do rio grande utilizando modelos de box & jenkins e redes neurais artificiais. *Computer on the Beach 2011*, p. 227–283, 2011. Disponível em: <https://siaiap32.univali.br/seer/index.php/acotb/article/download/6405/3634>. Acesso em: 10 fev. 2018.

BETIOLI, I. et al. Projeto itaipu – aspectos relacionados com a segurança das estruturas e equipamentos. *XXIII Seminario Nacional de Grandes Barragens*, Belo Horizonte, 1999. Disponível em: <http://www.cbdb.org.br/documentos/23sngb/PROJETO%20ITAIPU.pdf>. Acesso em: 27 set. 2018.

CARREGOSA, F. B. *Implementando uma Máquina Virtual Diferenciável Mínima em Redes Neurais Recorrentes*. Dissertação (Mestrado) — Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2018. Disponível em: <https://www.cos.ufrj.br/uploadfile/publicacao/2838.pdf>. Acesso em: 29 set. 2018.

CARVALHO, A. C. P. de Leon Ferreira de. *Redes Neurais Artificiais*. 2009. Disponível em: <http://conteudo.icmc.usp.br/pessoas/andre/research/neural/>. Acesso em: 23 fev. 2018.

CORRÊA, J. M. *Método Warimax-Garch Neurais para Previsão de Séries Temporais*. Tese (Doutorado) — Universidade Federal do Paraná, Curitiba, 2015. Disponível em: <https://dspace.c3sl.ufpr.br/bitstream/handle/1884/41955/R%20-%20T%20-%20JAIRO%20MARLON%20CORREA.pdf?sequence=1&isAllowed=y>. Acesso em: 25 jan. 2018.

CORRÊA, J. M. et al. Método warimax-garch neural para previsão de séries temporais de deslocamentos de blocos da barragem principal de itaipu. In: SILVA, A. da et al. (Ed.). *Métodos Numéricos Aplicados a Análise de Segurança de Barragens*. 1. ed. Foz do Iguaçu: Parque Itaipu, 2016. p. 77–105.

DANTAS, D. et al. Uso de redes neurais artificiais na previsão da precipitação de períodos chuvosos. *Revista Espinhaço*, v. 5, n. 1, p. 11–18, 2016. Disponível em:

<http://www.revistaespinhaco.com/index.php/journal/article/view/96/90>. Acesso em: 23 fev. 2018.

FIGUEREDO, J. C. *Previsão de Séries Temporais Utilizando a Metodologia Box & Jenkins e Redes Neurais para Inicialização de Planejamento e Controle de Produção*. Dissertação (Mestrado) — Universidade Federal do Paraná, Curitiba, 2008. Disponível em: <http://www.acervodigital.ufpr.br/bitstream/handle/1884/16675/Previs?sequence=1>. Acesso em: 22 fev. 2018.

JONES, M. T. *Um mergulho profundo nas redes neurais recorrentes*. 2017. Disponível em: <https://imasters.com.br/data/um-mergulho-profundo-nas-redes-neurais-recorrentes>. Acesso em: 10 ago. 2018.

KERAS. *Keras: The Python Deep Learning library*. 2018. Disponível em: <https://keras.io/>. Acesso em: 26 set. 2018.

MILLSTEIN, F. *Deep Learning: 2 Manuscripts - Deep Learning With Keras And Convolutional Neural Networks In Python*. [S.l.]: CreateSpace Independent Publishing Platform, 2018. ISBN 1986718271.

NELSON, D. M. Q. *Uso de Redes Neurais Recorrentes para Previsão de Séries Temporais Financeiras*. Dissertação (Mestrado) — Universidade Federal de Minas Gerais, Belo Horizonte, 2017. Disponível em: <https://www.dcc.ufmg.br/pos/cursos/defesas/1999M.PDF>. Acesso em: 12 ago. 2018.

PAULO, U. de S. *Capítulo 5: Introdução as Séries Temporais e aos Modelos ARIMA*. 2018. Disponível em: <https://edisciplinas.usp.br/pluginfile.php/176834/mod_resource/content/1/Resumo%20Wooldridge%20-%20Introdu%C3%A7%C3%A3o%20%C3%A0s%20S%C3%A9ries%20Temporais.pdf>. Acesso em: 20 fev. 2018.

PYSCIENCE, B. *Python: O que é? Por que usar?* 2018. Disponível em: <http://pyscience-brasil.wikidot.com/python:python-oq-e-pq>. Acesso em: 26 set. 2018.

REIS, M. M. *Notas de aula da disciplina INE 7001 – Análise de Séries Temporais*. 2018. Disponível em: <http://www.inf.ufsc.br/~marcelo.menezes.reis/Cap4.pdf>. Acesso em: 20 fev. 2018.

SOUZA, R. C. T. de. *Previsão de Séries Temporais Utilizando Rede Neural Treinada Por Filtro de Kalman e Evolução Diferencial*. Dissertação (Mestrado) — Pontifícia Universidade Católica do Paraná, Curitiba, 2008. Disponível em: <http:

//www.biblioteca.pucpr.br/tede/tde_busca/arquivo.php?codArquivo=1300>. Acesso em: 23 fev. 2018.

TEIXEIRA, L. L. et al. Previsão de séries temporais por meio de um método híbrido wavelet-neural integrado com bootstrap. In: SILVA, A. da et al. (Ed.). *Métodos Numéricos Aplicados a Análise de Segurança de Barragens*. 1. ed. Foz do Iguaçu: Parque Itaipu, 2016. p. 159–189.

THEANO. *Theano at a Glance*. 2018. Disponível em: <http://deeplearning.net/software/theano/introduction.html>. Acesso em: 26 set. 2018.

Apêndice A

Apêndice

A.1 Código Fonte

```
import numpy
import matplotlib.pyplot as plt
import pandas
import itertools
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from openpyxl import load_workbook

# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset) - look_back -1):
```

```
        a = dataset[i:(i + look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return numpy.array(dataX), numpy.array(dataY)

def create_list(dataset):
    data = []
    for i in range(0,len(dataset)):
        # print(dataset[i])
        data.append([dataset[i]])
    return numpy.array(data)

## ***********   PARTE EXCEL   **************
xlsx_file = 'datos_completo.xlsx'
xlsx = pandas.ExcelFile(xlsx_file)
# folhas = xlsx.sheet_names

folhas = ['F05', 'F13']
CANT_DADOS = 1907
CANT_PREV = 30
CANT_TREINO = CANT_DADOS - CANT_PREV

for f in folhas:
    dataframe = pandas.read_excel(xlsx_file, sheet_name=f,
        header = 0, usecols=[0,1,2], nrows=CANT_DADOS)

    # fix random seed for reproducibility
    numpy.random.seed(7)

    # serie original
    # serie = dataframe[0][:CANT_TREINO].values.tolist()
```

```
serie = dataframe['Real'][:CANT_TREINO].values.tolist()
serie = pandas.to_numeric(serie).astype('float32')

## serie completa
# serie_completa = dataframe['Real'][:CANT_DADOS].
    values.tolist()
# serie_completa = pandas.to_numeric(serie_completa).
    astype('float32')

serie_test = dataframe['Real'][CANT_TREINO:].values.
    tolist()
serie_test = pandas.to_numeric(serie_test).astype('
    float32')

arima = dataframe['ARIMA'][:CANT_TREINO].values.tolist
    ()
arima = pandas.to_numeric(arima).astype('float32')

arima_previsto = dataframe['ARIMA'][CANT_TREINO:].
    values.tolist()
arima_previsto = pandas.to_numeric(arima_previsto).
    astype('float32')

residuos_arima = dataframe['Residuo'][:CANT_TREINO].
    values.tolist()
residuos_arima = pandas.to_numeric(residuos_arima).
    astype('float32')

dataset = dataframe['Residuo'][:CANT_DADOS]

dataset = dataframe['Residuo'][:CANT_DADOS].values
```

```
dataset = create_list(dataset)

# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset2 = scaler.fit_transform(dataset)

# split into train and test sets
train_size = CANT_TREINO
test_size = len(dataset2) - train_size
train, test = dataset2[0:train_size, :], dataset2[
    train_size:len(dataset2), :]

# reshape into X=t and Y=t+1
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)

# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1,
    trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.
    shape[1]))

# create and fit the LSTM network
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='
    adam')
```

```
model.fit(trainX, trainY, epochs=10, batch_size=1,
    verbose=2, shuffle=False)

# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)

# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])

# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0],
    trainPredict[:, 0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0],
    testPredict[:, 0]))
print('Test Score: %.2f RMSE' % (testScore))

treinado = list(itertools.chain.from_iterable(
    trainPredict))
testado = list(itertools.chain.from_iterable(
    testPredict))

train_lstm = numpy.add(arima[:len(treinado)],treinado)
test_lstm = numpy.add(arima_previsto[:len(testado)],
    testado)

#graficar a serie
```

```
plt.title("Serie Pendulo Bloco "+f)
plt.ylabel('Valores Leituras')
plt.plot(serie, 'b-', label='Serie Original')

vacio = numpy.empty(len(serie))
vacio.fill(numpy.nan)

serie_test_plot = numpy.insert(serie_test,0,vacio)
plt.plot(serie_test_plot, 'c:', label='Serie Test')

plt.plot(train_lstm, 'g-', label='LSTM - Treinado')

vacio = numpy.empty(len(train_lstm))
vacio.fill(numpy.nan)

previsto_plot = numpy.insert(test_lstm, 0, vacio)
plt.plot(previsto_plot, 'r:', label='LSTM - Previst')

plt.legend()

# manager = plt.get_current_fig_manager()
# manager.window.showMaximized()
#
plt.show()
#plt.savefig('FIG'+f+'_final.png', dpi=500)

plt.close()
print('Proceso Finalizado para la Serie '+f)
```