https://eventos.utfpr.edu.br//sicite/sicite2019

Aplicação de Redes Neurais Artificiais nos dados de temperatura da barragem de contraforte de uma hidroelétrica em funcionamento.

Artificial Neural Network application using temperature from a working hydroelectric buttress dam data

Isabella Basilio Rossato
isabellarossato@gmail.com
Universidade Tecnológica Federal do Paraná (UTFPR), Medianeira, Paraná, Brasil

Jairo Marlon Correa
jairocorrea@utfpr.edu.br
Universidade Tecnológica Federal do Paraná (UTFPR), Medianeira, Paraná, Brasil

Recebido: 19 ago. 2019.
Aprovado: 01 out. 2019.

RESUMO

A barragem de contraforte da hidrelétrica é dividida por blocos que possuem medidores de temperatura e deformações, armazenados e ordenados em séries temporais. O presente trabalho tem por objetivo modelar os dados de temperatura do bloco D-38 através de redes neurais artificiais voltadas para a previsão de séries temporais. Para que isso seja realizado utilizou-se a linguagem de programação *Python* para construir a rede, gerar as simulações e os resultados gráficos. Devido ao fato de serem coletados manualmente os dados - permitindo erro de paralaxe - e também por não possuir padronização quanto à periodicidade de coleta dos dados, foi necessário o tratamento dos mesmos para facilitar sua manipulação. A estrutura de rede utilizada para a modelagem foi o *Perceptron* de múltiplas camadas, que possui camadas de neurônios ocultas em seu modelo. Dessa forma foi possível gerar as previsões dentro dos dados e comprovar a eficácia da rede ao processar as informações.

PALAVRAS-CHAVE: Redes Neurais (Computação). Análise de séries temporais. Simulação (computadores). Python (Linguagem de programação de computador).

ABSTRACT

The buttress dam from the hydroelectric is divided in blocs, each one has temperature and deformation measurers that generate data, organizing it in temporal series format. The aim of this work was to model temperature data from D-38 bloc using Neural Network to predict time series values. To achieve this it was used python programing language for construct networks and generate simulations. Due to the fact that data have been manually collected – allowing parallax errors – also because the non-standardization of frequency to collect the data, was necessary to treat the measurements to make easy to process the information. To modelate data was used the multilayer perceptron network structure, whereby are neuronal hidden layers included in the architecture. Therefore was possible to develop forecasts inside real data, proving the effectiveness of the modeling network.

KEYWORDS: Artificial Neural Network. Analysis of time series. Computer modeling. Python (Computer programing language).

INTRODUÇÃO

No livro *Time Series Analysis: Forecasting and control* os autores Box e Jenkins (2016, p.1) afirmam que uma série é considerada temporal quando a distribuição de seus dados sé dá por meio de intervalos regulares de tempo (dia, mês, ano). Como as medidas são independentes entre si, é de suma importância a ordem dos dados, visto que se forem amostras coletadas em curto prazo de tempo, seus valores serão bem próximos, mas caso sejam coletadas em longos prazos seus valores serão bem distintos.

Através do estudo do processamento de informações no cérebro foi constatado que ele é capaz de reorganizar suas estruturas, que são os neurônios, de maneira a otimizar seus processos e as respostas a eles, fornecendo um controle eficiente de tudo que acontece no organismo que ele controla. O cérebro também é capaz de aprender padrões e grava-los na memória para que no futuro isso seja utilizado de alguma forma.

Dada tal complexidade de funcionamento do cérebro humano, uma rede neural artificial é comparada a ele, segundo Simon Haykin (2008, p.28) autor do livro "Rede Neurais Princípios e Práticas", uma rede neural é uma máquina projetada para modelar a maneira como o cérebro realiza uma tarefa particular ou função de interesse, onde a rede é normalmente implementada se utilizando componentes eletrônicos ou é simulada por programação em um computador. Assim é possível utilizar uma rede para modelar séries temporais e produzir previsões.

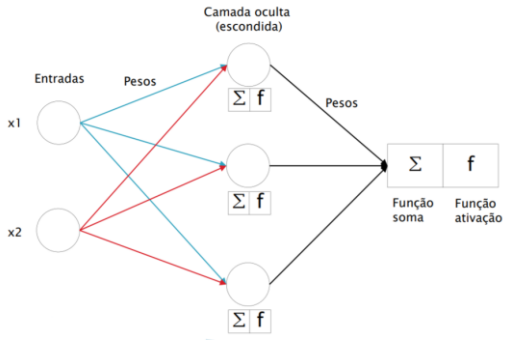
MATERIAL E MÉTODOS

Os dados fornecidos pela hidrelétrica se tratam de uma série temporal contínua, no entanto a periodicidade de coleta dos dados não é regular, fazendo com que hajam partes da série com dados diários, dados mensais e dados faltantes em determinados períodos. Observou-se que inicialmente eram feitas medições diárias pois a barragem havia sido recém construída, necessitando de um acompanhamento mais rígido, enquanto a partir do ano 2000 foram mensais.

Dessa forma trabalhou-se com a série contendo o período dos anos 2000 a 2018 em que as informações estavam organizadas mensalmente e em alguns períodos específicos estavam em trimestres. Para que essa série pudesse ser manipulada pela rede foi necessário um processo de tratamento nos dados brutos, que consistiu em fazer a média entre os valores anteriores ao faltante com os valores posteriores, seguindo a ordem de tempo. Por serem grandes quantidades de informação optou-se por fazer a limpeza dos dados através da programação na linguagem *python*, que com um código simples automatizou o tratamento dos dados.

A rede neural artificial possui características de funcionamento particulares para cada uma de suas variadas estruturas, dentre elas foi construída uma rede do tipo *Multilayer Perceptron* (MLP – *perceptron* de camadas múltiplas). A MLP é indicada para solução de sistemas não lineares, possuindo uma complexidade maior que as estruturas básicas.

Figura 1 – Modelo de Rede MLP com estrutura de camada simples



Fonte: GRANATYR, Jones. Redes Neurais Artificiais em Python. Udemy.

A estrutura da Figura 1 possui todos os seus neurônios interligados entre si, de forma que não importa a quantidade de camadas ocultas que possui, pois, todos os neurônios estão conectados. Isso permite um bom fluxo de sinal através das camadas, que enviado sempre para a camada da frente (*feed foward*) até chegar na camada de saída, carregando junto ao sinal o valor do erro de cada etapa de processamento. A rede desenvolvida segue exatamente o fluxograma da Figura 1, com uma camada de entrada e uma de saída, bem como uma única camada oculta.

O erro auxilia no ajuste dos pesos da rede, que são os valores que amplificam ou reduzem o sinal de entrada, fazendo a correlação entre os sinais. As entradas formam a primeira camada da rede, em seguida os sinais são transmitidos para as próximas camadas, sejam ocultas ou não, até a saída.

No modelo pode-se aplicar diferentes funções de ativação (*activation function*) que são aplicadas nas entradas dependendo de qual for a finalidade da rede, como por exemplo a função sigmoide e a função tangente hiperbólica.

Para fazer os cálculos de entrada utilizamos os sinais de entrada (x) multiplicados pelos pesos sinápticos (w) seguindo a equação (1):

$$soma = \sum x_i \, . \, w_i \qquad (1)$$

O resultado de cada soma é aplicado na função de ativação, gerando o valor de cada neurônio da primeira camada oculta. Para a próxima, os valores de cada neurônio associados na camada anterior são novamente multiplicados pelos pesos e novamente o resultado de cada soma é aplicado na função. O processo é repetido até que chegue na camada de saída.

No processo de aprendizagem supervisionada da rede que foi implementado, o erro é calculado da seguinte forma (equação 2):

$$erro = resposta\ correta - resposta\ calculada \qquad (2)$$

Lembrando que nesse processo de aprendizagem a resposta correta é o resultado que se espera ter e a resposta calculada é o resultado final gerado pela rede. Pode-se fazer uma média utilizando o erro absoluto, obtendo assim o erro total.

Para reduzir o erro, dentro das camadas ocultas serão feitos cálculos de vetor gradiente, que nada mais é do que as derivadas parciais dos pesos, com o gradiente

encontra-se a combinação dos pesos com o menor erro possível e assim a rede saberá em quanto terá que ajustar os pesos para chegar ao resultado esperado.

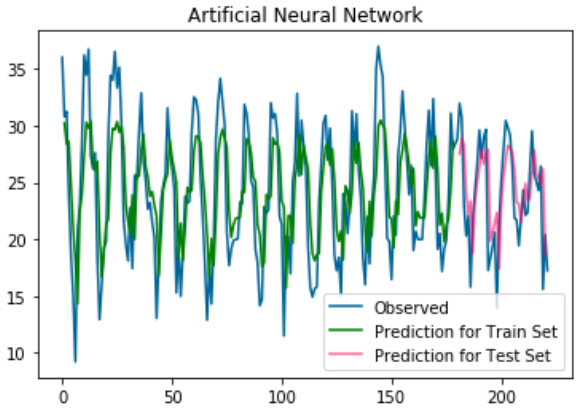
Há também um algoritmo de retropropagação (*backpropagation*) que é composto pelas duas fases, a *feed foward* e a *backward,* a segunda caminha em direção contrária a primeira, ou seja, vai fazer o processo de volta, indo da camada de saída para todas as outras camadas, fazendo o reajuste dos pesos de acordo com o erro.

RESULTADOS E DISCUSSÃO

Comprovou-se a eficácia do código desenvolvido na estrutura de múltiplas camadas do *perceptron* através de simulações efetuadas com os dados do bloco D-38, de forma que a rede possibilitou a modelagem dos dados, a previsão de treino, a previsão para validação do modelo. As funções de ativação foram avaliadas aquelas que melhor modelaram os dados, visando produzir o mínimo de erro.

Abaixo encontram-se os gráficos das simulações efetuadas com os dados do Termômetro TS 003 tratados. Nesse caso a função que melhor se adequou foi a de unidade linear retificada (relu) exibida na Figura 3, note que por serem dados mais simples, várias funções conseguiram se adaptar.

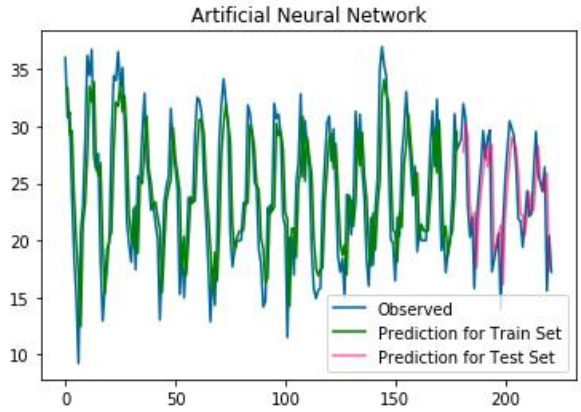
Figura 2 – Resultado com a função Sigmoide



Fonte: Produção do próprio autor.

Nos gráficos os dados reais estão representados em azul, a previsão de treino em verde e a previsão de validação em rosa. É fundamental observar o comportamento das previsões para que seja definida qual a função fornece a melhor modelagem e processamento dos dados, veja que na Figura 2 a parte rosa está razoável em relação a parte azul do gráfico, ou seja, a validação está coerente. No entanto a linha verde do gráfico contém bastante erro e não se aproxima tanto do real.

Figura 3 – Resultado com a função uniddade linear retificada (relu)

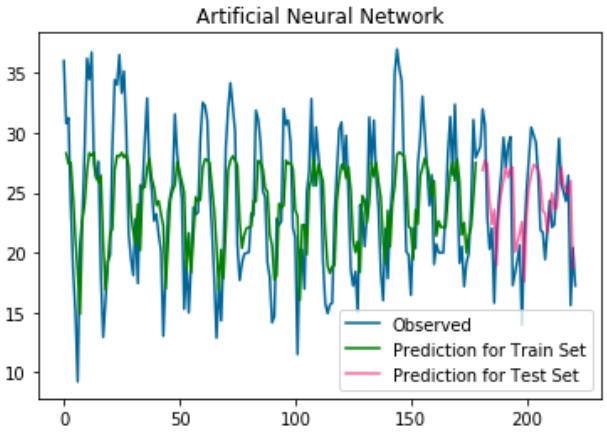


Fonte: Produção do próprio autor.

Na Figura 3 (acima) está o gráfico da função que obteve os melhores resultados, ao fazer a análise entre as linhas de previsão (verde e rosa) e a linha dos dados reais (azul), é fácil perceber que estão muito próximas umas das outras, a função relu foi a que gerou o menor erro, tornando a modelagem e a previsão bem aproximada dos dados de entrada.

Na Figura 4 está o resultado quando aplicada a função Softmax, note que ela faz com que as previsões sejam semelhantes à uma média entre os dados, fugindo do esperado que é o mais próximo possível dos dados fornecidos.

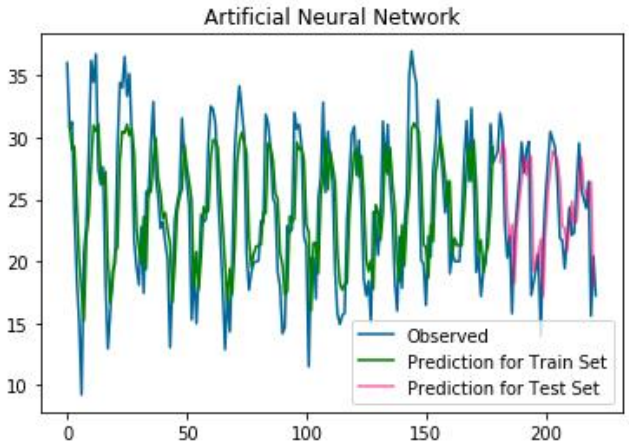
Figura 4 – Resultado com a função softmax



Fonte: Produção do próprio autor.

A função hiperbólica também se mostra bastante eficiente na modelagem, sua utilização gerou o gráfico apresentado na Figura 5, onde a previsão de teste e também a de validação estão razoavelmente próximas dos dados reais, é valido observar que no início do treino ela demorou a se ajustar, conforme o gráfico em verde mostra, mas depois de ajustada gerou uma boa previsão de validação (em rosa).

Figura 5 – Resultado com a função tangente hiperbólica



Fonte: Produção do próprio autor.

Para gerar as previsões o algoritmo utiliza os dados de entrada para gerar os dados de saída, fazendo com que a rede necessite ser constantemente alimentada com dados reais para poder obter novos valores de previsão. Os gráficos apresentados seguem esse algoritmo, todos eles são produzidos pelo autor a partir o processamento dos dados efetuados pela MLP.

CONCLUSÃO

Através do trabalho desenvolvido foi possível modelar a série temporal dos dados do termômetro de uma hidrelétrica em funcionamento aplicando uma rede neural artificial de estrutura *perceptron* de múltiplas camadas, e também efetuar as previsões de treino e de validação da rede dentro das informações de entrada.

A rede desenvolvida mostrou-se bastante eficaz, mesmo com funções de ativação não tão boas para a modelagem, ela foi capaz de gerar resultados razoáveis. Suas simulações permitiram definir a função que promove o melhor desempenho e também fixar parâmetros no código que foi desenvolvido.

REFERÊNCIAS

HAYKIN, Simon. **Redes Neurais: princípios e prática.** 2 ed. Porto Alegre: Bookman, 2001.

BOX, George E. P. et al. **Time Series Analysis: Forecasting and Control.** 5 ed. New Jersey: Wiley, 2016.

Artificial Intelligence #5: MLP Networks with scikit & keras. Udemy. Disponível em: <https://tinyurl.com/y3ndl4kh>. Acesso em: Julho, 2019.

Arquiteturas de Aprendizado Profundo. IBM Developer. Disponível em: <https://tinyurl.com/y2k6lrn2>. Acesso em: Julho, 2019.

GRANATYR, Jones. Redes Neurais Artificiais em Python. Udemy. Disponivel em: <https://tinyurl.com/y3rakgn2>. Acesso em: Março, 2019.