Instituto Tecnológico de Aeronáutica

Divisão de Ciência da Computação (IEC)

CT-221: Redes Neurais com Aprendizagens Clássica e Profunda

Lista de Laboratório 1

Alunos

Daniel Bueno Borges
Davi Teixeira Lessa
Heyder Falheiro de Almeida
Igor Campos Ibiapina
Paulo Vinícius Ribeiro Silva
Rafael Lucas de Alburquerque Louvain

Professor

Paulo Marcelo Tasinaffo

02 de Outubro de 2024

1 Introdução

As redes neurais artificiais são modelos matemáticos inspirados no funcionamento do cérebro humano, capazes de realizar tarefas de classificação, predição e reconhecimento de padrões. Dentre as arquiteturas mais simples está o perceptron de camada única, introduzido por Rosenblatt em 1958. Este modelo consiste em uma camada de entrada que recebe o vetor de entrada $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, e uma camada de saída, onde cada neurônio aplica uma função de ativação a uma combinação linear das entradas ponderadas pelos pesos $\mathbf{w} = [w_1, w_2, \ldots, w_n]$, acrescidas de um termo de *bias* b. A saída de um perceptron é descrita pela equação:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

A função de ativação f mais comumente utilizada é a função degrau, que retorna 1 se o valor da soma for maior que zero, e 0 caso contrário. No entanto, o perceptron de camada única é limitado a problemas de classificação linearmente separáveis, como a função AND. Ele falha, por exemplo, ao tentar resolver o problema XOR, que é não-linearmente separável.

Para superar essas limitações, surgiu o perceptron de múltiplas camadas (MLP), que introduz camadas ocultas entre as camadas de entrada e saída. Essas camadas internas permitem a modelagem de relações não-lineares entre as entradas e saídas, ampliando a capacidade de classificação do modelo. Ao adicionar essas camadas, o MLP consegue encurvar o hiperplano de separação entre classes, possibilitando resolver problemas como o XOR. A estrutura geral de uma MLP é composta por:

- Um vetor de entrada $\mathbf{x} = [x_1, x_2, \ldots, x_n]$; - Um conjunto de pesos $\mathbf{W}$ que conecta cada camada; - Um *bias* b associado a cada neurônio; - Funções de ativação não-lineares, como a função sigmoidal ou a tangente hiperbólica.

A saída da rede em cada camada é calculada por:

$$\mathbf{y} = f(\mathbf{W} \cdot \mathbf{x} + \mathbf{b})$$

As camadas ocultas introduzem não-linearidades essenciais para a resolução de problemas complexos, e o processo de aprendizado dos pesos é feito por meio do algoritmo de retropropagação, que utiliza o gradiente do erro para ajustar os pesos e minimizar a função de custo.

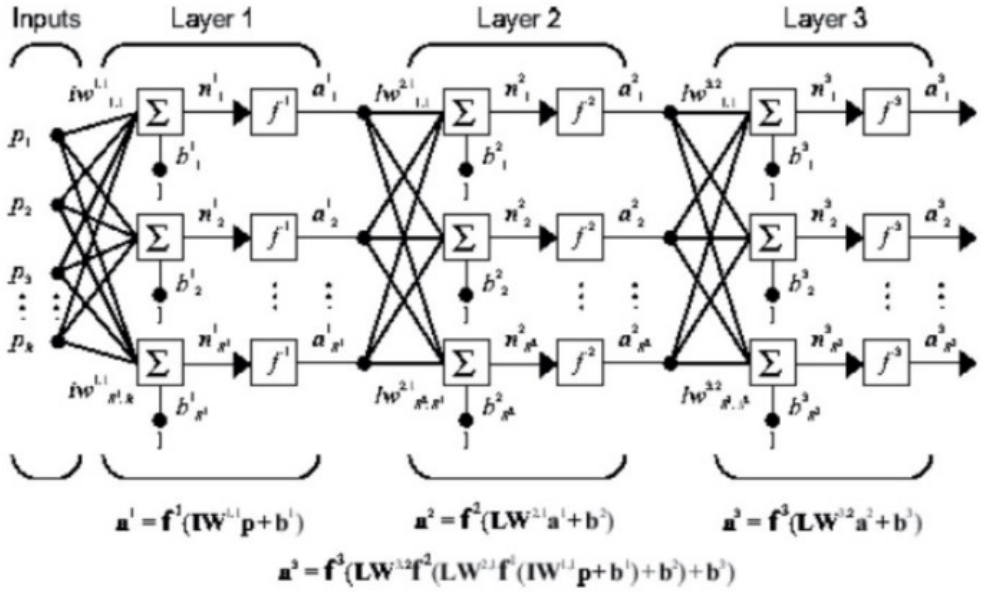

Figura 1.1: Arquitetura de um perceptron de múltiplas camadas.

A Toolbox de Redes Neurais do MATLAB simplifica a implementação e o treinamento de redes MLP. Essa ferramenta oferece diversas funções para a criação, configuração, treinamento e validação de redes neurais, permitindo a escolha de diferentes algoritmos de treinamento, como o Levenberg-Marquardt e o Resilient Backpropagation (RPROP). Além disso, a toolbox facilita o pré-processamento dos dados de entrada, divisão de dados em conjuntos de treinamento, validação e teste, e oferece funções para monitorar o desempenho durante o treinamento.

Com essas capacidades, o MATLAB permite a experimentação de diferentes configurações de redes neurais, ajustando hiperparâmetros como o número de camadas, neurônios por camada, taxa de aprendizado, e algoritmos de treinamento, o que possibilita otimizações no desempenho da rede em diversas aplicações.

Assim, pretende-se, nesse laboratório, analisar estruturas de redes MLP para o problema da previsão de vazão de dois rios ao longo do tempo.

2 Objetivo

O objetivo principal deste estudo é desenvolver e testar vários modelos de redes neurais baseados na arquitetura MLP (Perceptron Multicamadas), com o propósito de prever a vazão de dois rios a partir de dados históricos de vazões anteriores. Cada modelo implementa uma estrutura diferente de camadas ocultas e utiliza distintos algoritmos de treinamento, visando encontrar a melhor configuração para obter previsões precisas. A abordagem consiste em utilizar as vazões atrasadas de ambos os rios como entradas da rede e, a partir delas, determinar as vazões futuras, avaliando o desempenho de cada modelo em termos de convergência e precisão.

3 Metodologia

Para a realização deste estudo, foi implementada uma arquitetura de rede neural MLP (Perceptron Multicamadas) utilizando o software MATLAB. O foco foi prever a vazão de dois rios distintos (Camargos e Furnas) a partir de dados históricos de vazões. Os dados foram divididos em entradas (vazões atrasadas) e saídas (vazões futuras), com o intuito de treinar e testar diferentes configurações de redes neurais. Foi adotada a divisão dos dados da forma 0,7:0,15:0,15 em Treinamento:Validação:Teste.

Foram utilizadas duas séries temporais, uma para cada rio, cada qual contendo 82 meses de medições. Cada conjunto de dados foi organizado em uma matriz, onde cada linha representava as vazões mensais de um determinado ano. Para o treinamento dos modelos, as três primeiras colunas de cada linha representaram as vazões atrasadas, enquanto as três colunas seguintes foram utilizadas como as saídas desejadas, correspondendo às vazões futuras.

Além disso, o treinamento da rede neural foi realizado com base nos seguintes parâmetros, usando 100% dos dados para treinamento:

- Tempo máximo de treinamento: 10 minutos
- Número máximo de épocas (*epochs*): 1.000.000
- Taxa de aprendizado: 0.2
- Gradiente mínimo para convergência: 10^{-18}
- Número máximo de falhas de validação: 1000
- Critério de erro: $\mu = 10^{10}$

4 Resultados e Discussão

Neste estudo, foram implementados seis modelos distintos para a arquitetura da Rede Neural Multicamadas (MLP) conforme discutido anteriormente. Cada modelo foi desenvolvido com a finalidade de explorar diferentes configurações de hiperparâmetros e topologias da rede, visando otimizar e comparar a previsão das vazões dos rios Camargos e Furnas usando modelos diferentes. Além disso, vale ressaltar que em todos os modelos a estrutura da rede é formada por duas camadas adicionais: a camada de entrada, responsável por receber os dados de entrada, e a camada de saída, que fornece as previsões de vazão.

4.1 Modelo 1

Neste modelo, foi implementada uma arquitetura de rede neural com uma única camada oculta, composta por cinco neurônios, utilizando o algoritmo de treinamento Levenberg-Marquardt (trainlm). Esta abordagem possibilitou a obtenção de resultados altamente convergentes, demonstrando a eficácia do modelo na previsão das vazões.

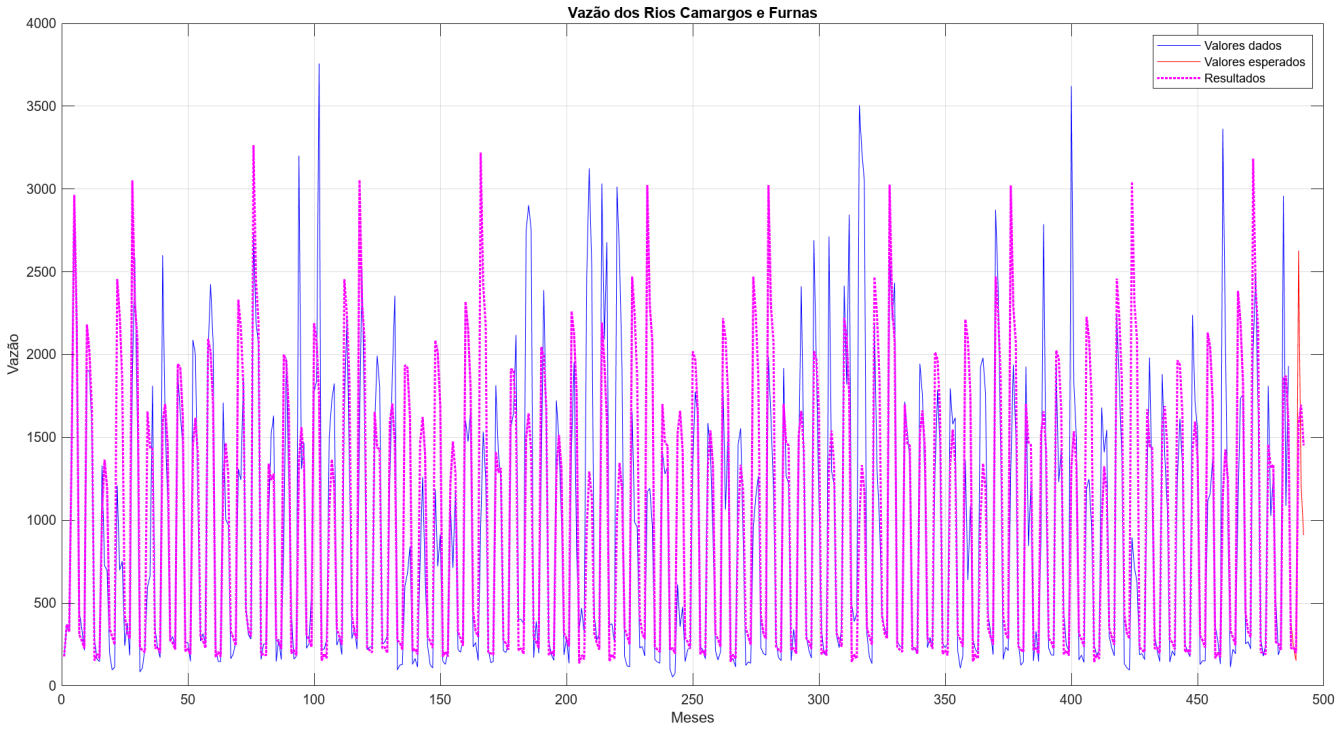

Figura 4.1: Resultado da implementação do Modelo 1.

O gráfico encontrado de performance, com análise do erro quadrático, juntamente com a análise de regressão, histograma e dados do treinamento estão expostos abaixo.

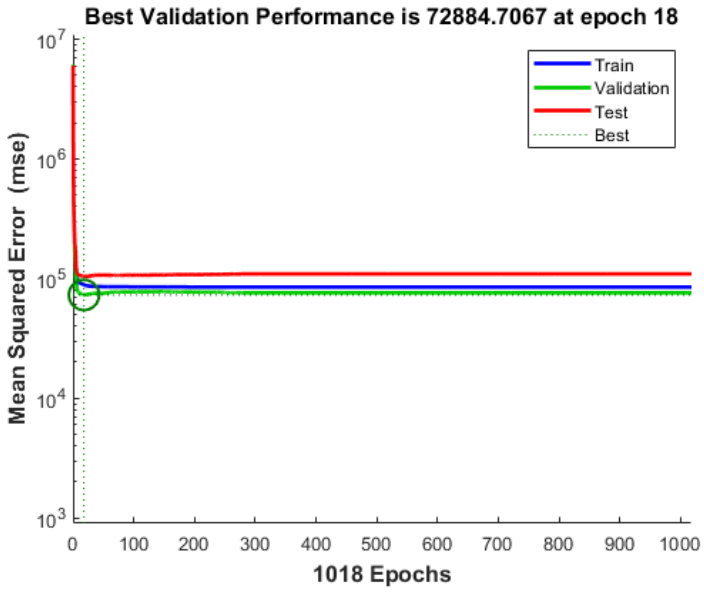

Figura 4.2: Performance do Modelo 1.

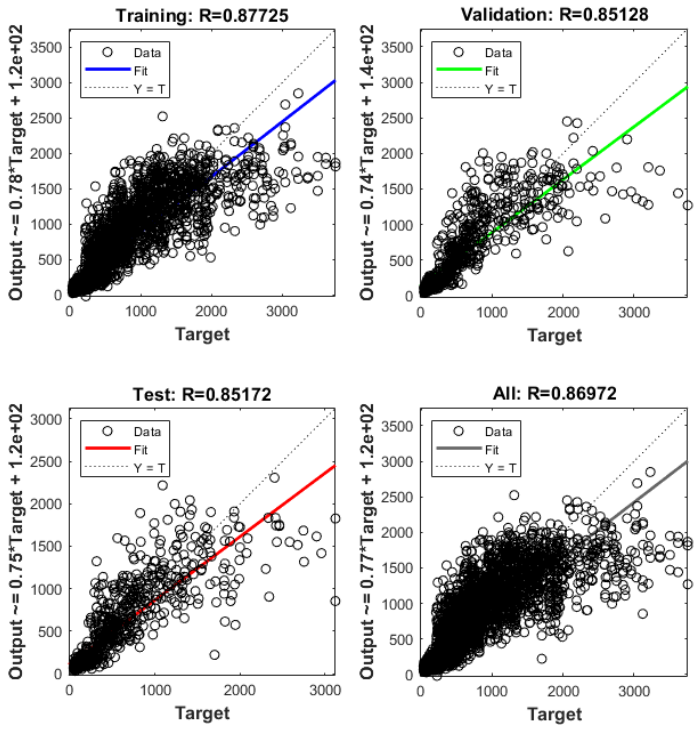

Figura 4.3: Análise de Regressão do Modelo 1.

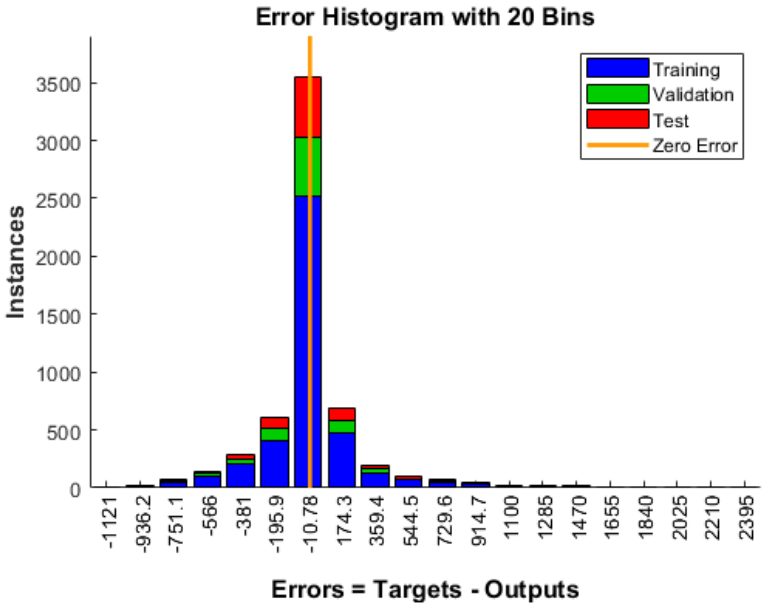

Figura 4.4: Histograma do Modelo 1.

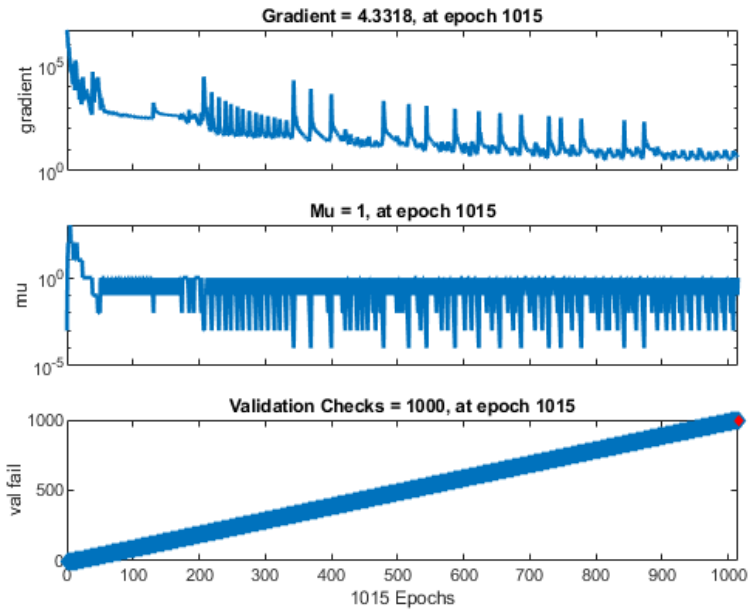

Figura 4.5: Dados do Treinamento do Modelo 1.

A análise dos resultados indica que a escolha do algoritmo Levenberg-Marquardt foi fundamental para maximizar a precisão da previsão, aproveitando a robustez do modelo com uma camada oculta. Este algoritmo não apenas facilitou a rápida convergência durante o treinamento, mas também garantiu que os resultados obtidos fossem consistentes e confiáveis, minimizando o risco de sobreajuste.

A configuração de cinco neurônios na camada oculta foi cuidadosamente selecionada para encontrar um equilíbrio ideal entre complexidade e desempenho. A eficiência deste modelo é evidenciada pelos resultados convergentes, que se aproximam significativamente das vazões reais, confirmando sua capacidade de generalização e efetividade na tarefa de previsão. Embora o valor de R esteja inferior a 0,9, nota-se que ele se aproxima bastante desse valor.

Um ponto que despertou quebra de expectativa foi o gráfico de performance, haja vista que esperaria-se a validação mais próxima do teste do que do treinamento, como se observa no gráfico.

4.2 Modelo 2

Neste modelo, também foi utilizado o treinamento Levenberg-Marquardt (trainlm). No entanto, a arquitetura consistiu em três camadas internas, compostas, respectivamente, por 15, 5 e 15 neurônios. Essa configuração mais complexa visava explorar a capacidade da rede neural em capturar padrões mais sutis nos dados.

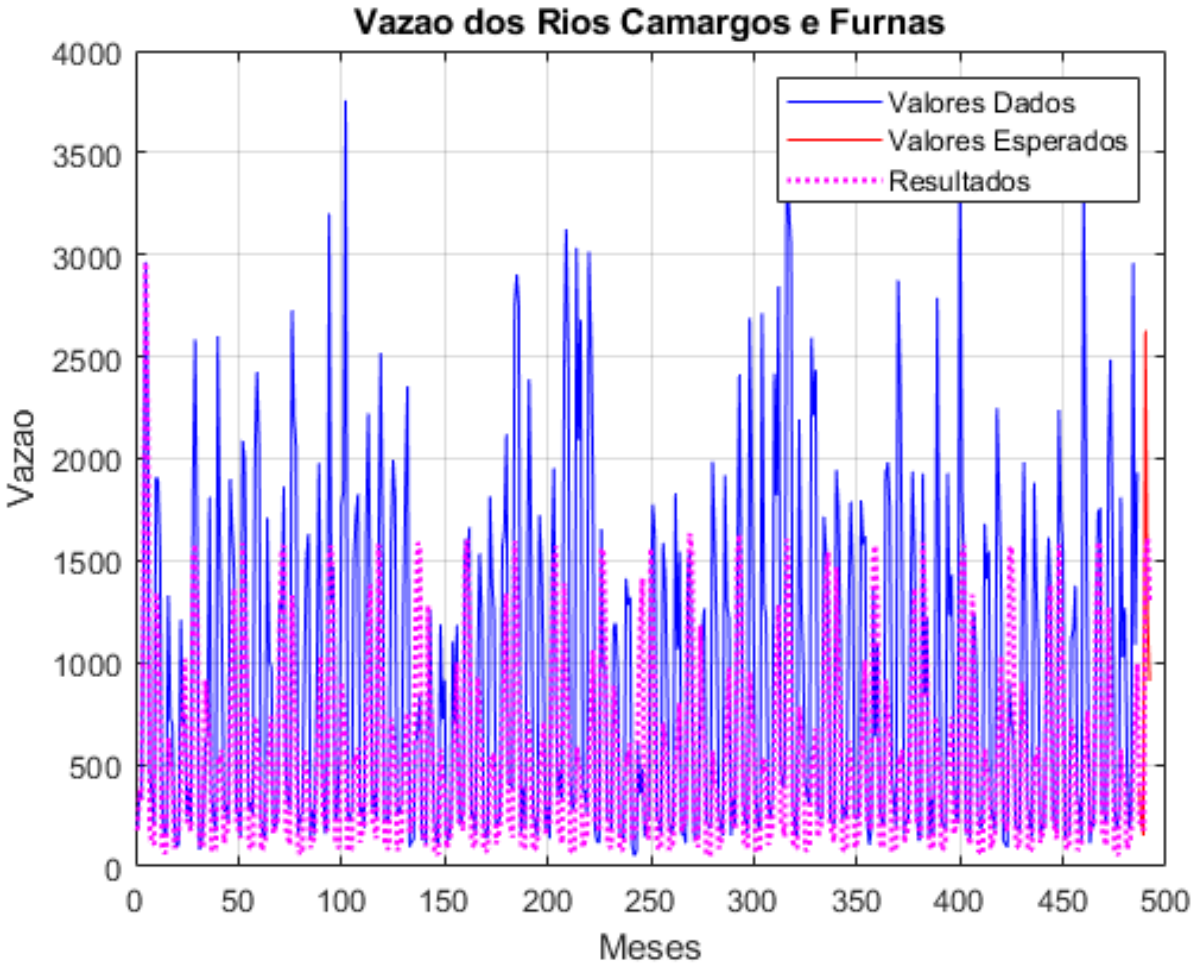

Figura 4.6: Resultado da implementação do Modelo 2.

O gráfico encontrado de performance, com análise do erro quadrático, juntamente com a análise de regressão, histograma e dados do treinamento estão expostos abaixo.

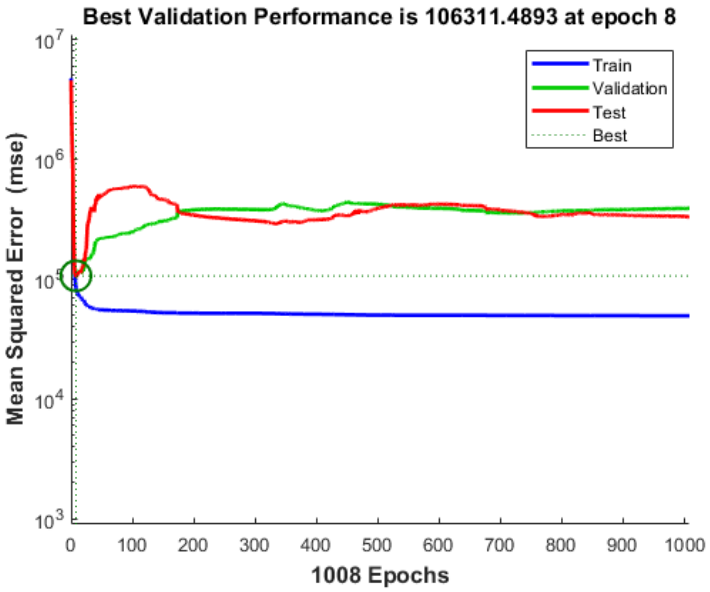

Figura 4.7: Performance do Modelo 2.

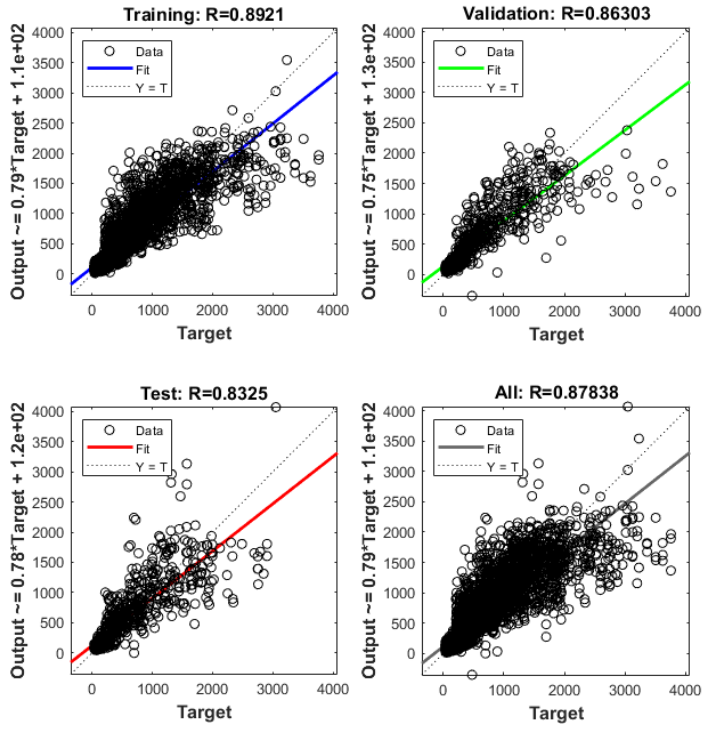

Figura 4.8: Análise de Regressão do Modelo 2.

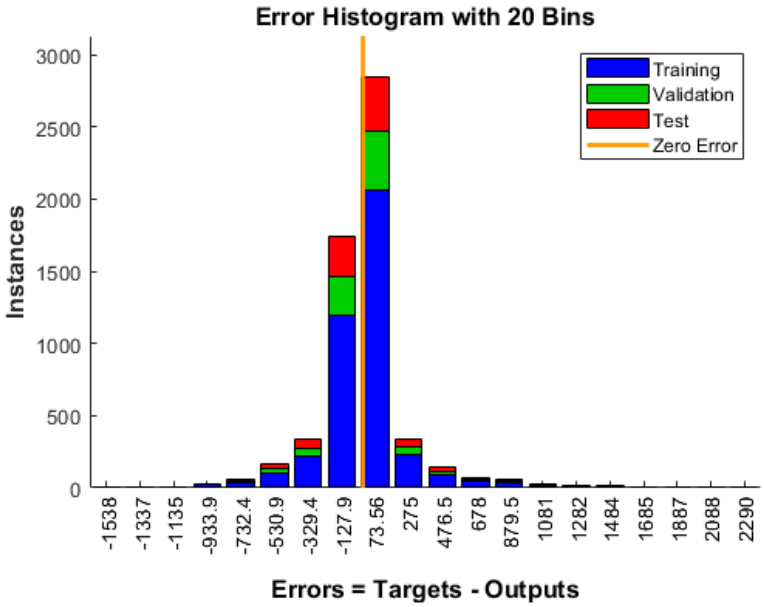

Figura 4.9: Histograma do Modelo 2.

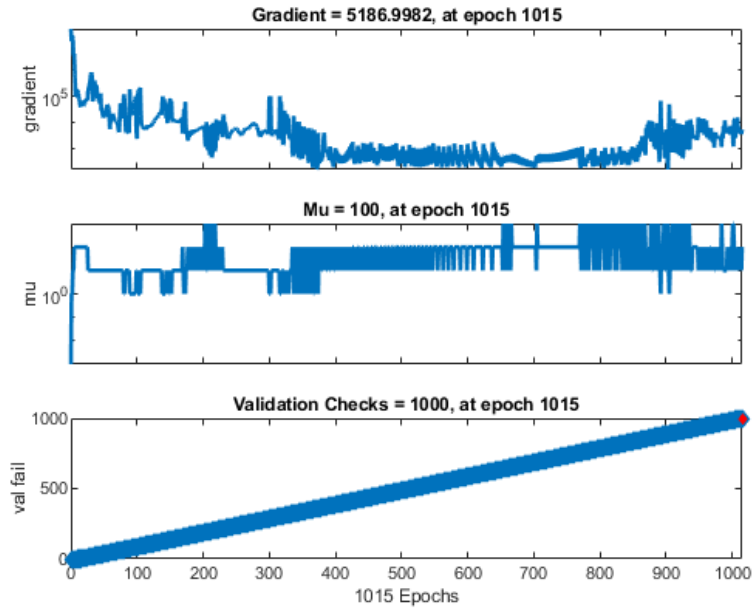

Figura 4.10: Dados do Treinamento do Modelo 2.

Entretanto, a adição de camadas e neurônios leva a um tempo de treinamento significativamente maior para a convergência dos resultados, conforme pode ser facilmente visto no gráfico de performance. Apesar do resultado encontrando com esse modelo obviamente não ser o ideal em vista dos resultados, como por exemplo vê-se o parâmetro R na faixa de 0,83 para o teste, vê-se

que o modelo convergiu. Conforme comentado, possivelmente esse modelo se adequada melhor com problemas mais sutis na distribuição de dados.

4.3 Modelo 3

Neste modelo, foi implementada uma arquitetura de rede neural com uma única camada interna composta por 15 neurônios, utilizando o algoritmo de treinamento Resilient Backpropagation (trainrp). Esta escolha de algoritmo é notável por sua capacidade de lidar eficientemente com problemas de otimização, especialmente em cenários onde as funções de erro podem ser difíceis de modelar.

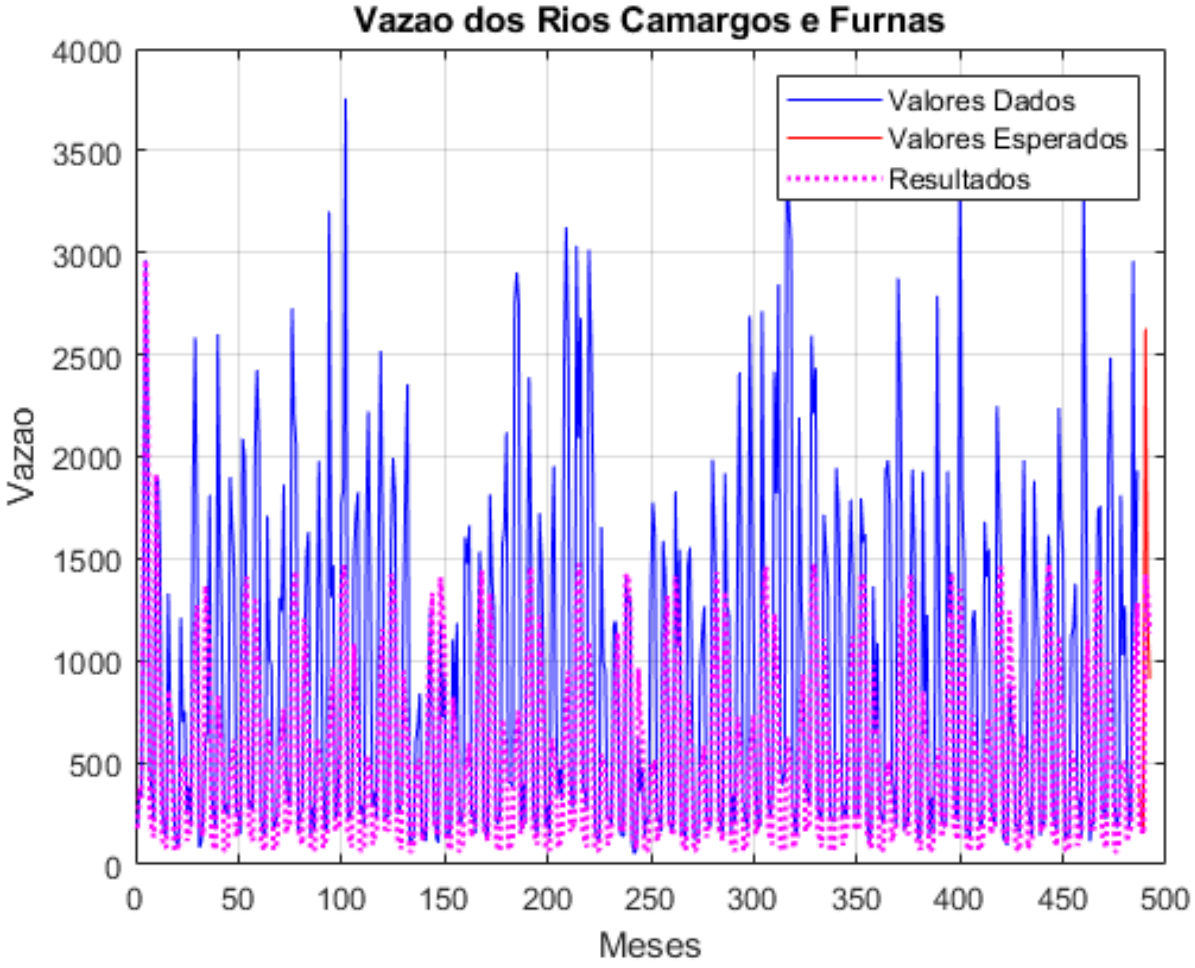

Figura 4.11: Resultado da implementação do Modelo 3.

O gráfico encontrado de performance, com análise do erro quadrático, juntamente com a análise de regressão, histograma e dados do treinamento estão expostos abaixo.

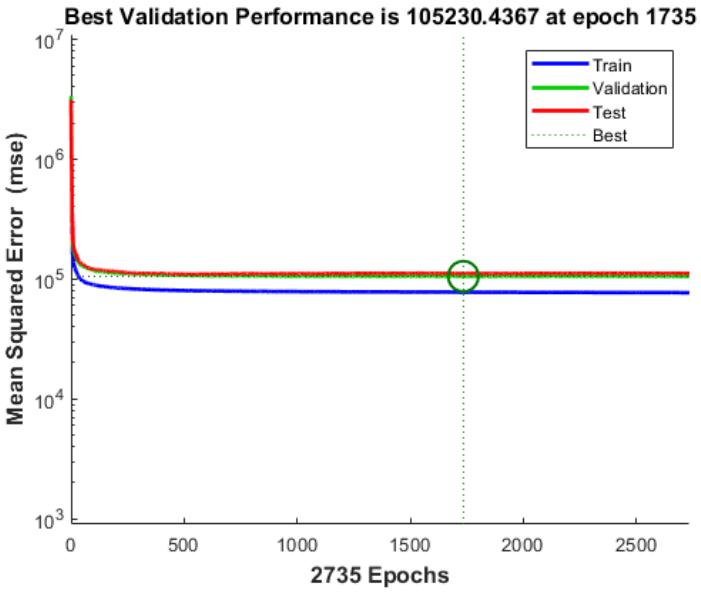

Figura 4.12: Performance do Modelo 3.

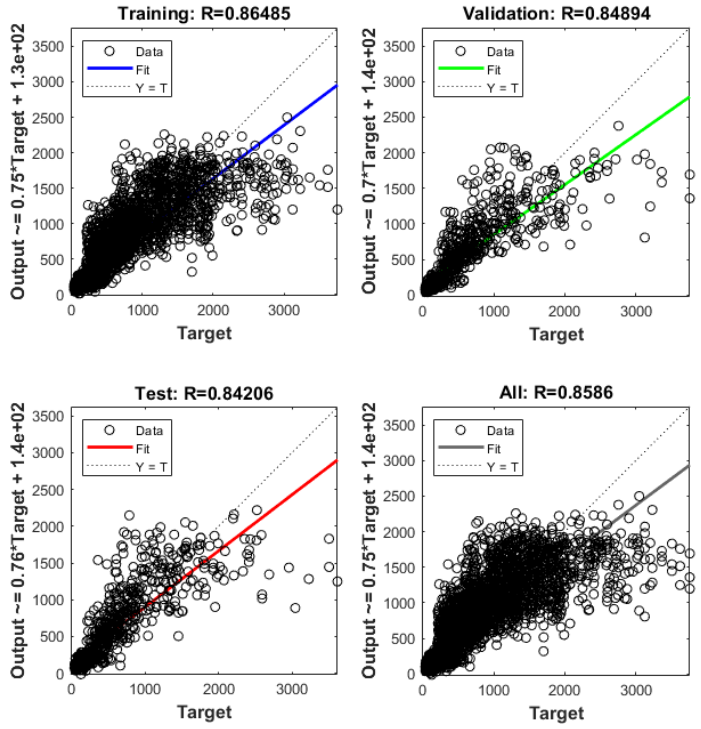

Figura 4.13: Análise de Regressão do Modelo 3.

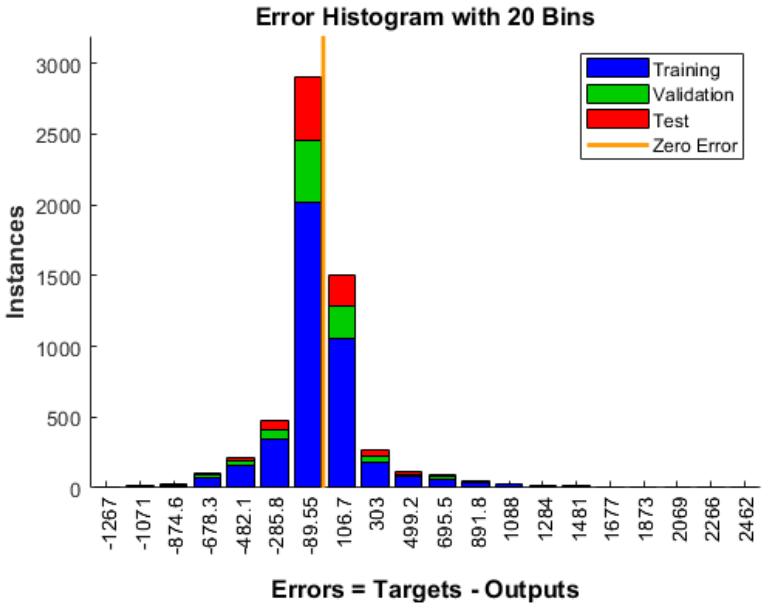

Figura 4.14: Histograma do Modelo 3.

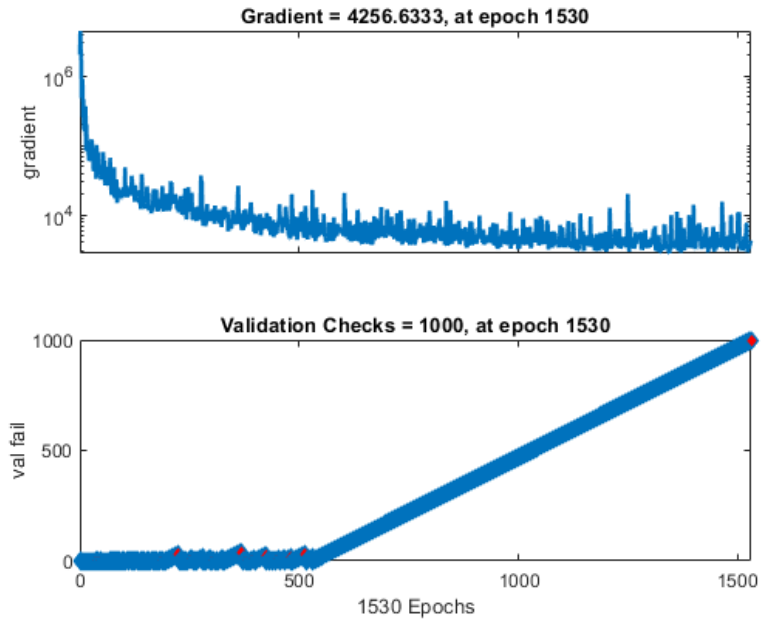

Figura 4.15: Dados do Treinamento do Modelo 3.

A configuração da rede, com uma camada oculta de 15 neurônios, possibilitou uma representação satisfatória dos dados durante a fase inicial da geração dos resultados. Com o avanço, no entato, resultou em uma baixa correlação entre previsões e valores reais. O algoritmo RPROP,

embora eficiente em muitos casos, pode não ter sido o mais adequado para este problema específico devido à oscilação observada nos erros.

Nota-se, no entanto, elevada proximidade no gráfico de performance entre a validação e o teste, conforme esperado para resultados satisfatórios.

4.4 Modelo 4

Neste modelo, foi implementada uma arquitetura de rede neural com uma única camada interna composta por 100 neurônios, utilizando o algoritmo de treinamento Polak-Ribiére Conjugate Gradient (traincgp). Este algoritmo é conhecido por sua eficiência em otimização, permitindo convergências mais rápidas em comparação com métodos tradicionais.

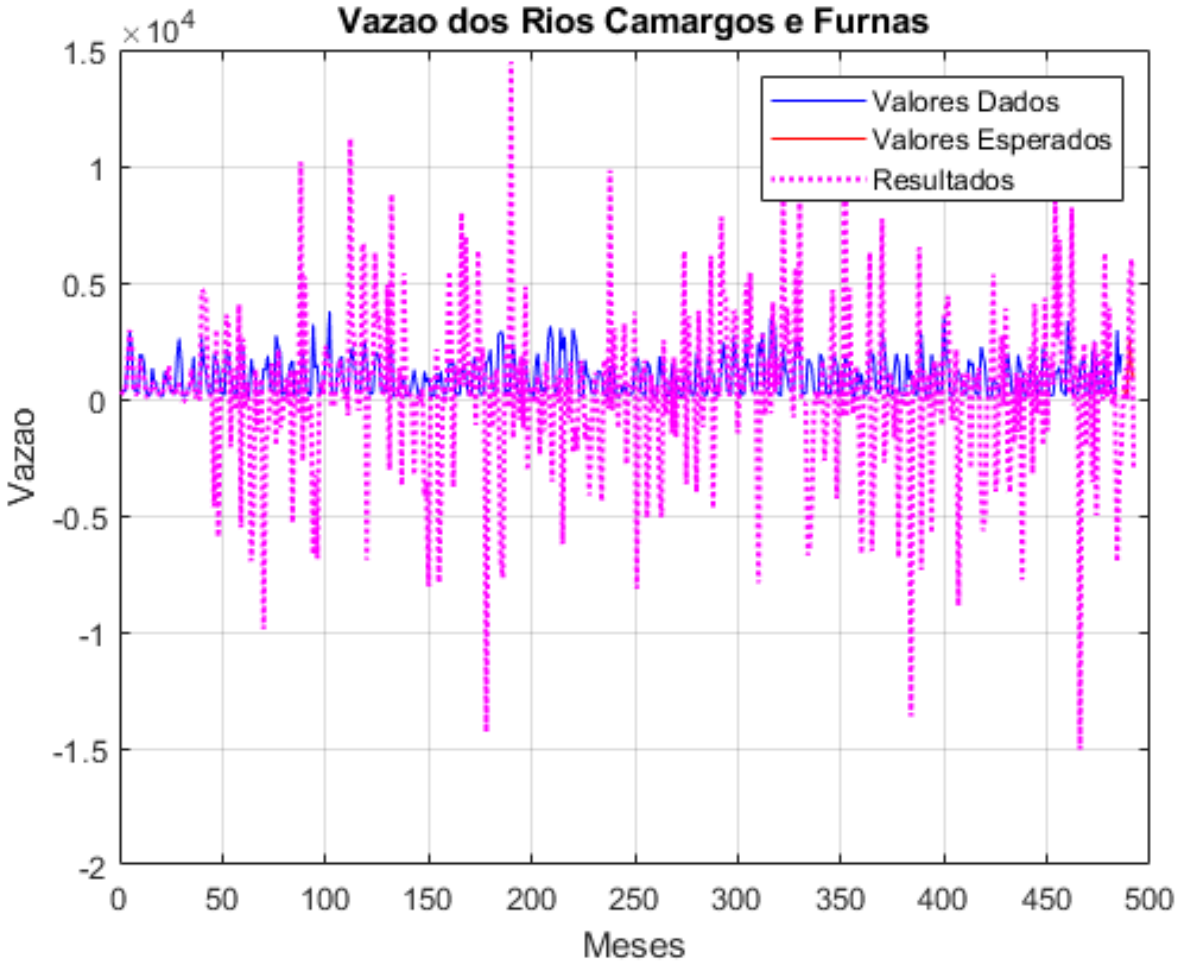

Figura 4.16: Resultado da implementação do Modelo 4.

O gráfico encontrado de performance, com análise do erro quadrático, juntamente com a análise de regressão, histograma e dados do treinamento estão expostos abaixo.

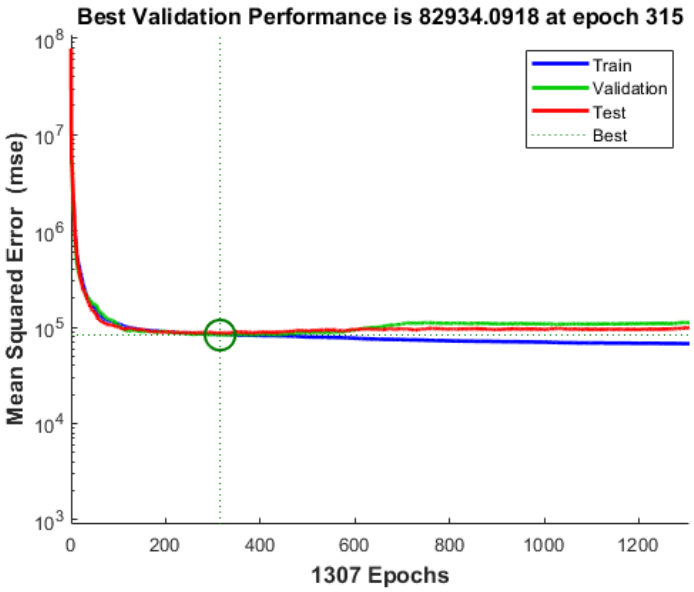

Figura 4.17: Performance do Modelo 4.

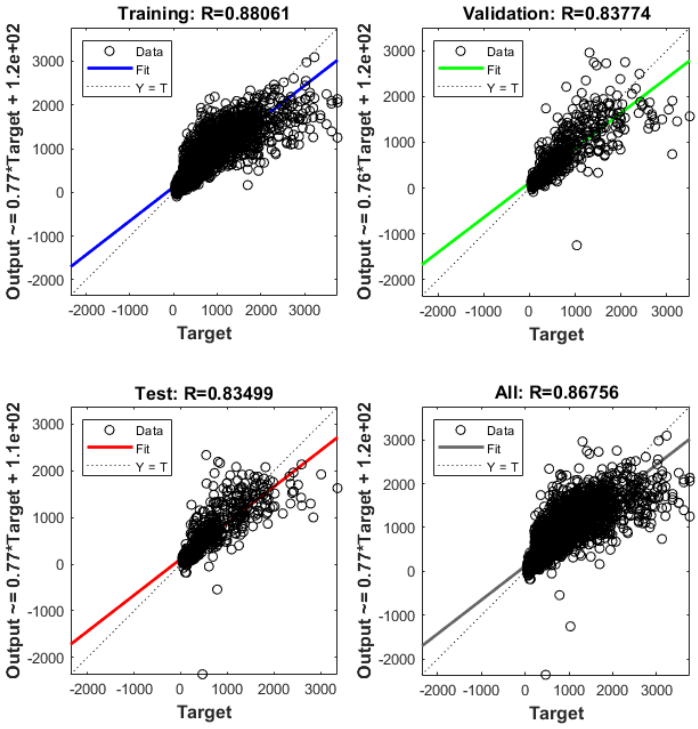

Figura 4.18: Análise de Regressão do Modelo 4.

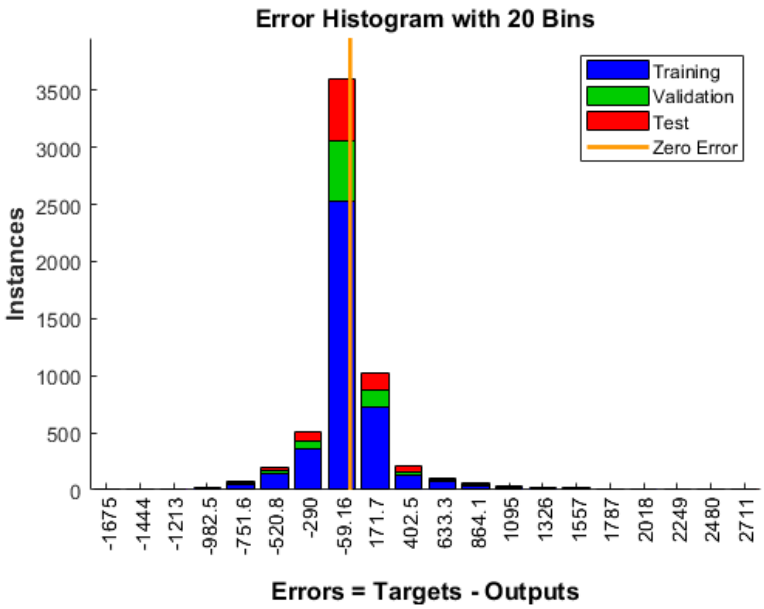

Figura 4.19: Histograma do Modelo 4.

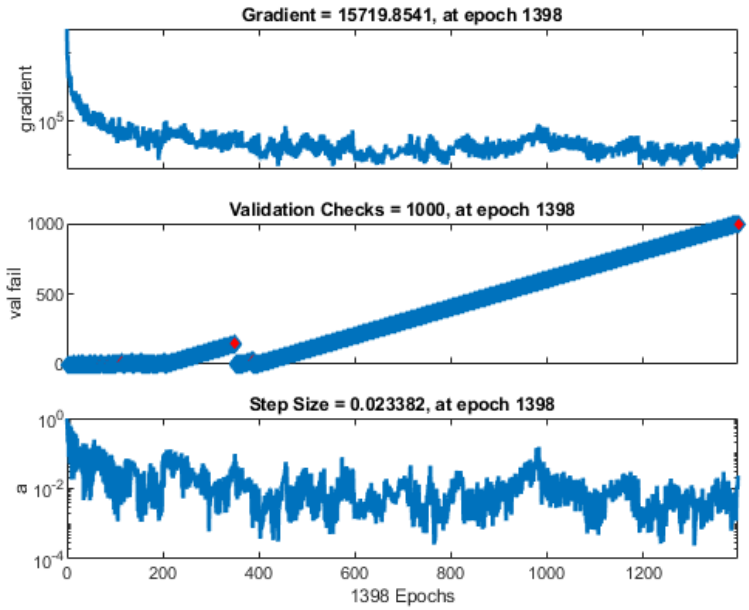

Figura 4.20: Dados do Treinamento do Modelo 4.

Ao analisar a imagem apresentada, observamos que o desempenho do modelo não correspondeu às expectativas. Apesar do resultado ter sido gerado de forma não tão lenta , a precisão das previsões não foi satisfatória. A alta complexidade da rede, com 100 neurônios na camada oculta,

pode ter contribuído para um ajuste inadequado aos dados, resultando em uma generalização insuficiente.

A arquitetura com 100 neurônios e o algoritmo Polak-Ribiére Conjugate Gradient não conseguiu generalizar bem, resultando em previsões imprecisas. A complexidade da rede, combinada com uma falha em reduzir o erro, indica que o modelo sofreu com o sobreajuste. Um ajuste na quantidade de neurônios e a implementação de regularização poderiam melhorar a generalização e a precisão do modelo.

4.5 Modelo 5

Neste modelo, foi implementada uma arquitetura de rede neural com uma única camada oculta composta por 50 neurônios, utilizando o algoritmo de treinamento Bayesian Regularization (trainbr). O algoritmo trainbr é conhecido por ser eficaz na redução do sobreajuste, pois incorpora regularização automática ao treinamento, ajustando a complexidade do modelo com base nos dados.

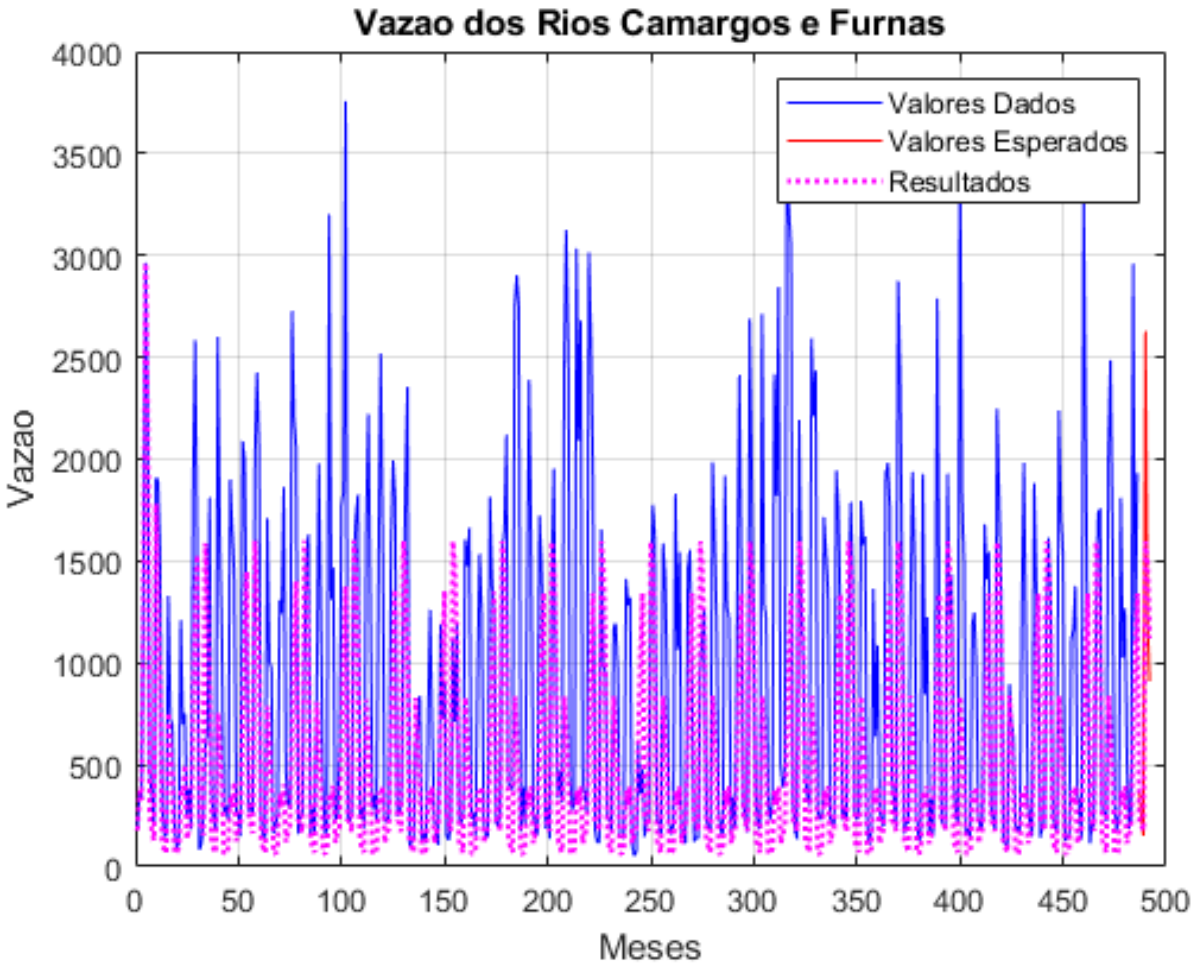

Figura 4.21: Resultado da implementação do Modelo 5.

O gráfico encontrado de performance, com análise do erro quadrático, juntamente com a análise de regressão, histograma e dados do treinamento estão expostos abaixo.

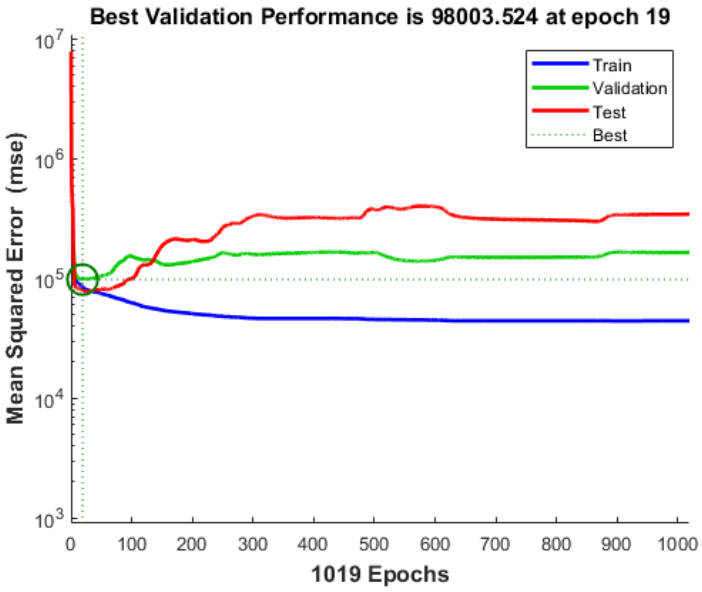

Figura 4.22: Performance do Modelo 5.

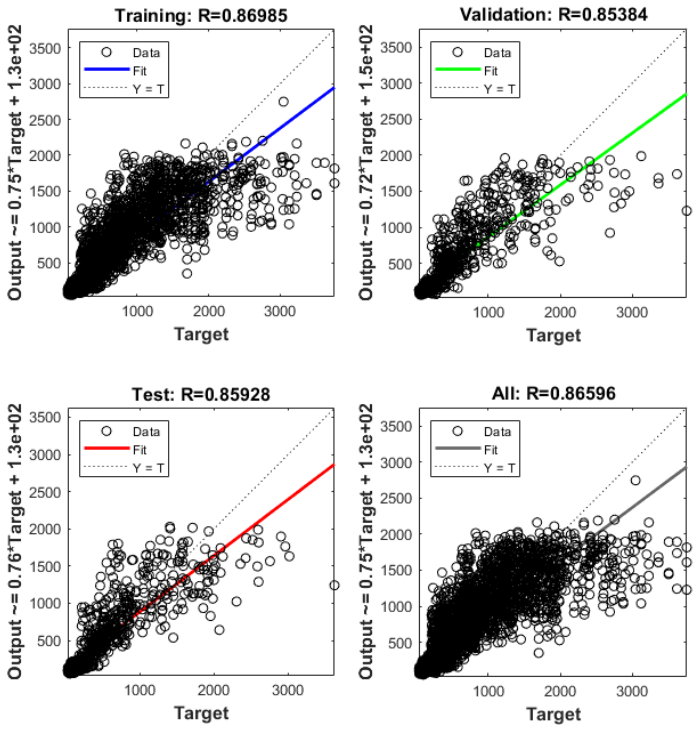

Figura 4.23: Análise de Regressão do Modelo 5.

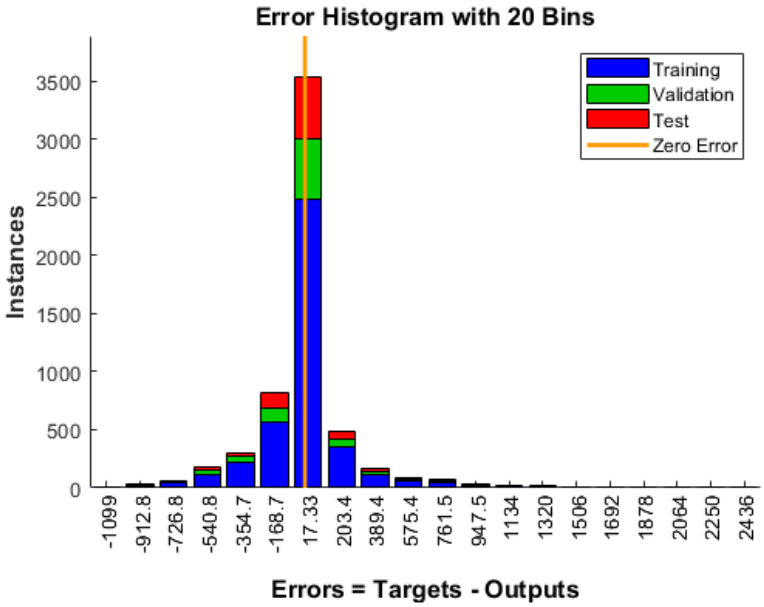

Figura 4.24: Histograma do Modelo 5.

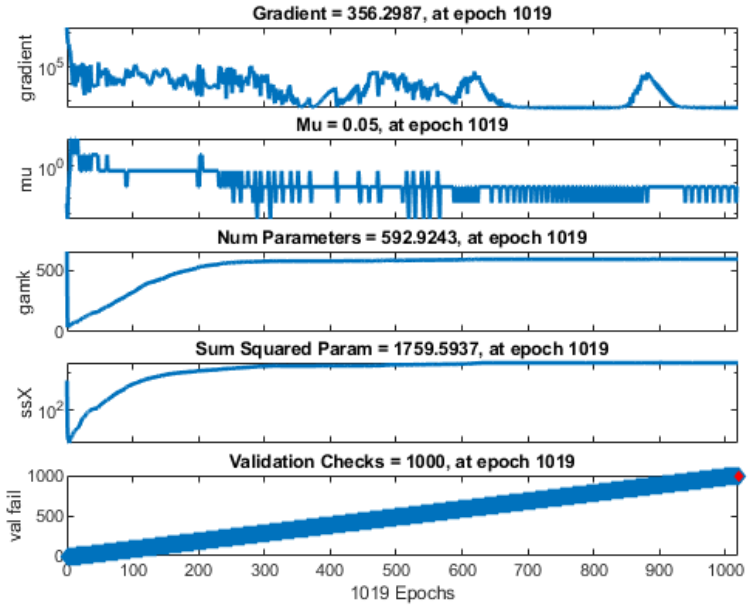

Figura 4.25: Dados do Treinamento do Modelo 5.

Embora o treinamento não tenha atingido o limite de tempo estipulado, o modelo não convergiu adequadamente. Isso indica que, mesmo com a regularização bayesiana, a configuração atual de 50 neurônios na camada oculta e os dados disponíveis não foram suficientes para alcançar

uma solução otimizada dentro dos parâmetros de treino. Mais ajustes nos hiperparâmetros, ou mesmo a adição de mais camadas ocultas, poderiam ser necessários para melhorar a convergência.

Nota-se, para esse modelo, um significativo descompasso entre a curva de validação e teste, o que não representa um resultado buscado para modelos que se adaptam bem ao problema proposto.

4.6 Modelo 6

Neste modelo, foi implementada uma arquitetura de rede neural com duas camadas ocultas, cada uma composta por 4 neurônios, utilizando o algoritmo de treinamento Levenberg-Marquardt (trainlm).

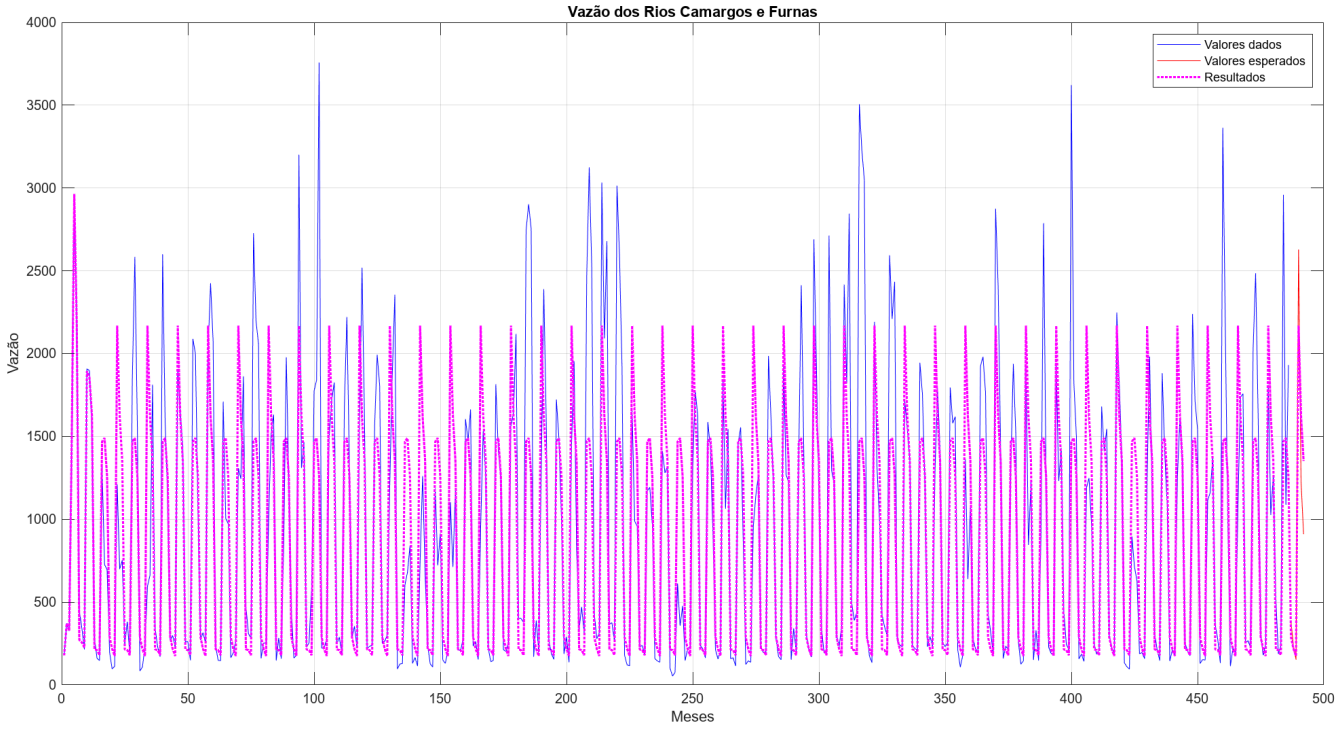

Figura 4.26: Resultado da implementação do Modelo 6.

O gráfico encontrado de performance, com análise do erro quadrático, juntamente com a análise de regressão, histograma e dados do treinamento estão expostos abaixo.

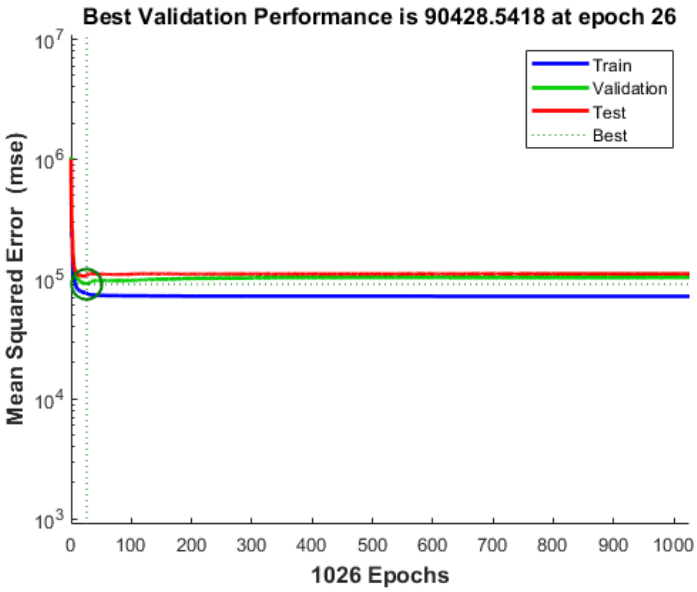

Figura 4.27: Performance do Modelo 6.

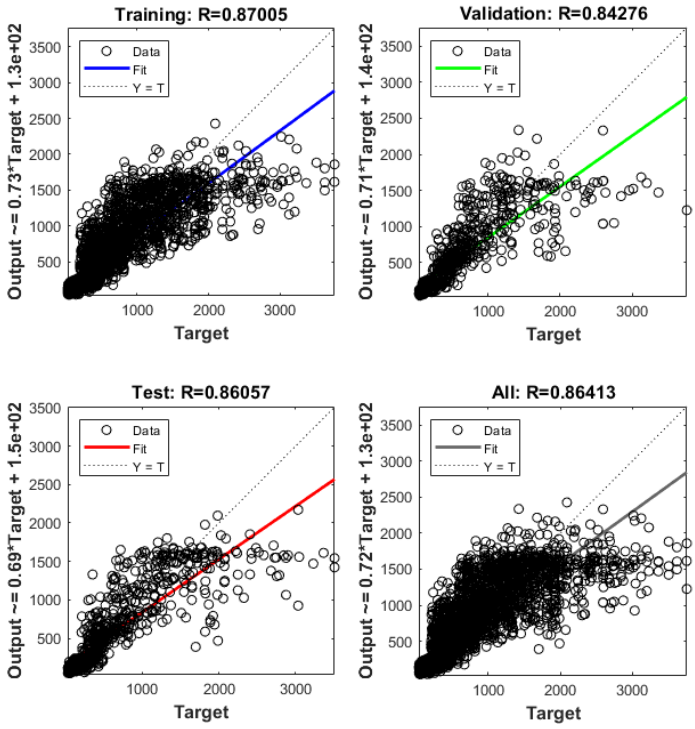

Figura 4.28: Análise de Regressão do Modelo 6.

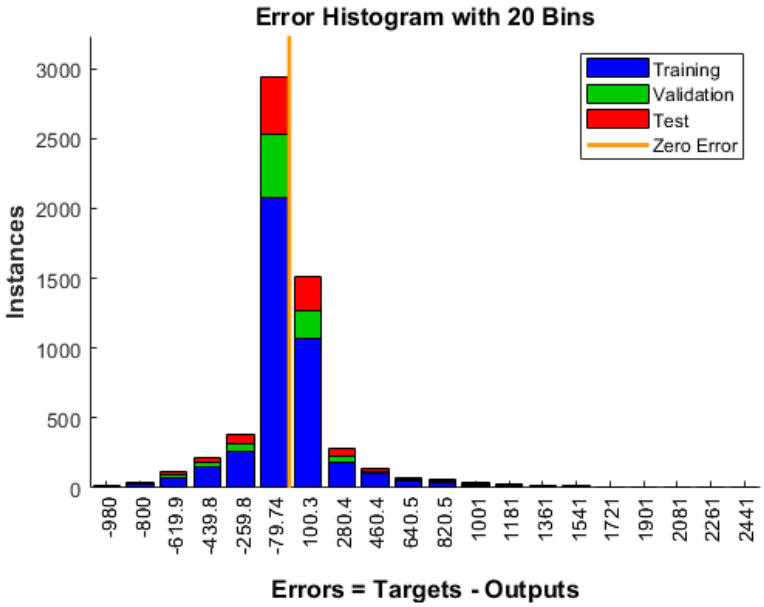

Figura 4.29: Histograma do Modelo 6.

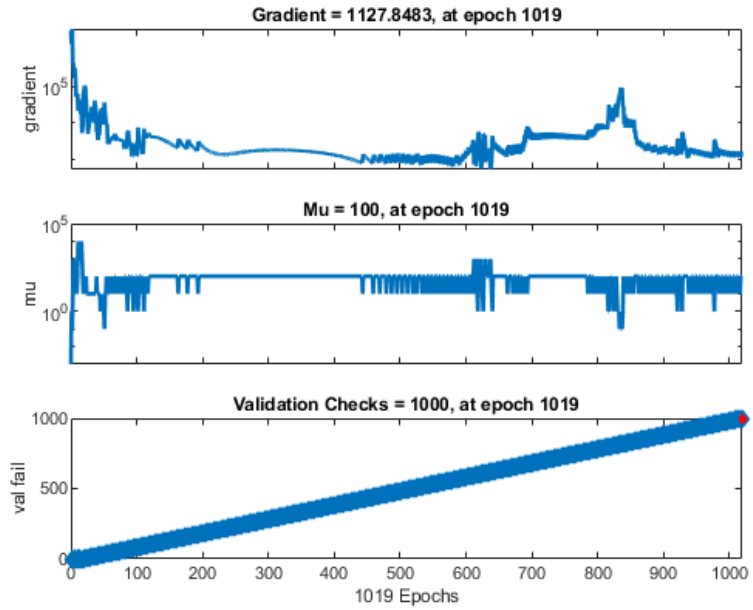

Figura 4.30: Dados do Treinamento do Modelo 6.

Apesar de a rede contar com duas camadas ocultas, o número reduzido de neurônios (apenas 4 em cada camada) limitou a capacidade do modelo de capturar padrões mais complexos presentes nos dados. Como resultado, a rede apresentou uma evolução promissora durante o treinamento, porém não atingiu a convergência esperada. Esse comportamento pode ser atribuído à falta de

flexibilidade na arquitetura para modelar adequadamente as relações temporais entre as vazões dos rios.

Além disso, outro fator que pode ter contribuído para a falta de convergência foi o tempo máximo de treinamento estipulado (10 minutos), que pode não ter sido suficiente para o algoritmo Levenberg-Marquardt refinar os parâmetros da rede de forma ideal, especialmente em uma tarefa tão desafiadora como a previsão de séries temporais de múltiplas entradas e saídas.

5 Conclusão

Neste estudo, foram explorados seis modelos de redes neurais MLP com diferentes configurações de camadas ocultas e algoritmos de treinamento, com o objetivo de prever as vazões dos rios Camargos e Furnas. A análise comparativa dos modelos revelou importantes insights sobre o impacto da arquitetura e dos parâmetros de treinamento no desempenho das previsões.

O **Modelo 1**, com uma única camada oculta de 5 neurônios e utilizando o algoritmo Levenberg-Marquardt, demonstrou a melhor performance em termos de convergência e precisão, com um valor de R próximo de 0,9 e uma distribuição de erros concentrada. Este resultado confirma que, em certos cenários, arquiteturas simples podem generalizar bem e oferecer um equilíbrio ideal entre complexidade e desempenho.

O **Modelo 2**, que utilizou três camadas ocultas (15, 5 e 15 neurônios), apresentou uma maior complexidade, mas não conseguiu convergir de forma adequada no tempo disponível, resultando em um R de 0,83. Isso sugere que, apesar de camadas adicionais potencialmente capturarem padrões mais sutis, a complexidade exige mais tempo de treinamento para ajustar os parâmetros de forma eficaz.

O **Modelo 3**, com 15 neurônios em uma única camada e utilizando o algoritmo Resilient Backpropagation, teve dificuldades de convergência, com flutuações significativas no erro e um desempenho inferior aos dois primeiros modelos. Isso evidencia que o algoritmo RPROP pode não ter sido o mais adequado para este problema específico.

No **Modelo 4**, com 100 neurônios em uma única camada e o algoritmo Polak-Ribiére Conjugate Gradient, o desempenho foi insatisfatório, com uma baixa correlação entre previsões e valores reais. A alta complexidade do modelo, combinada com a falta de regularização, provavelmente causou sobreajuste, limitando sua capacidade de generalização.

O **Modelo 5**, utilizando 50 neurônios e regularização bayesiana, também não convergiu adequadamente no tempo estipulado. Embora a regularização tenha ajudado a evitar o sobreajuste, o tempo insuficiente de treinamento e a complexidade do modelo prejudicaram a precisão das previsões.

Por fim, o **Modelo 6**, com duas camadas ocultas de 4 neurônios cada, não conseguiu capturar de forma eficaz as relações temporais presentes nos dados. O número reduzido de neurônios e o tempo limitado de treinamento resultaram em uma baixa correlação e uma dispersão significativa nos erros.

De modo geral, os resultados indicam que modelos mais simples, como o Modelo 1, tendem a obter melhores resultados quando há limitação de tempo de treinamento. Modelos mais complexos, embora promissores em termos de captura de padrões mais sofisticados, exigem mais tempo para ajustar seus parâmetros e alcançar uma convergência satisfatória. Para futuras implementações,

recomenda-se o ajuste cuidadoso de hiperparâmetros, aumento no tempo de treinamento e o uso de técnicas de regularização mais robustas para evitar sobreajuste em modelos complexos.

6 Apêndice A: Descrição da Linguagem Utilizada

O código utilizado para a implementação das redes neurais no relatório foi desenvolvido em **MATLAB**, uma linguagem de programação amplamente utilizada em cálculos numéricos, análise de dados e desenvolvimento de algoritmos. O MATLAB oferece uma *Toolbox* específica para Redes Neurais, a qual simplifica a criação, treinamento e validação de modelos de redes neurais artificiais, como o Perceptron de Múltiplas Camadas (MLP).

A *Toolbox* de Redes Neurais do MATLAB permite a configuração de diversos hiperparâmetros, como o número de camadas ocultas, neurônios por camada e algoritmos de treinamento (como Levenberg-Marquardt, Resilient Backpropagation e Bayesian Regularization). Além disso, facilita o pré-processamento dos dados e a divisão dos conjuntos de treinamento, validação e teste, bem como a visualização de métricas de desempenho, como gráficos de erro quadrático médio, regressão e histogramas.

A linguagem MATLAB é ideal para experimentação rápida em redes neurais, uma vez que oferece diversas funções integradas para otimização e análise de dados, além de ser amplamente utilizada no meio acadêmico e na indústria para a modelagem de sistemas e algoritmos complexos.

Apêndice B: Código fonte

Listing 1: Todo código utilizado

```
clear all
clc

% Criar Padr es de E/S
camargos = load('Camargos.txt');
furnas = load('Furnas.txt');

camargos_linha = reshape(camargos', 1, []);
furnas_linha = reshape(furnas', 1, []);

P = [];
T = [];

for i = 1:1:(82*12-5)
    % Entradas
    entrada_camargos = camargos_linha(i:(i+2))';
    entrada_furnas = furnas_linha(i:(i+2))';
    entrada = [entrada_camargos; entrada_furnas];

    % Sa das
    saida_camargos = camargos_linha((i+3):(i+5))';
    saida_furnas = furnas_linha((i+3):(i+5))';
    saida = [saida_camargos; saida_furnas];

    % Adicionar em P e T
    P = [P entrada];
```

```
    T = [T saida];
end

% Estruturas a serem testadas
net = feedforwardnet(5);
%net = feedforwardnet([15 5 15]);
%net = feedforwardnet(15);
%net = feedforwardnet(100);
%net = feedforwardnet(50);
%net = feedforwardnet([4,4]);
net = configure(net, P, T);

% Divis o de Padr es
net.divideFcn = 'dividerand';
net.divideParam.trainRatio = 0.7;
net.divideParam.valRatio = 0.15;
net.divideParam.testRatio = 0.15;

% Inicializando os pesos
net = init(net);

% Treinamento
net.trainParam.showWindow = true;

net.layers{1}.transferFcn = 'tansig';
net.layers{2}.transferFcn = 'purelin';
%net.layers{3}.transferFcn = 'tansig';
%net.layers{4}.transferFcn = 'purelin';

net.performFcn = 'mse';

net.trainFcn = 'trainlm';
%net.trainFcn = 'trainrp';
%net.trainFcn = 'traincgp';
%net.trainFcn = 'trainbr';

net.trainParam.epochs = 1000000;
net.trainParam.time = 600;
net.trainParam.lr = 0.2;
net.trainParam.min_grad = 10^-18;
net.trainParam.max_fail = 1000;
[net, tr] = train(net, P, T);

% Padr es de Treinamento
xP = 1:1:(81*6);       % Define o intervalo para as primeiras 81 linhas (6 ...
   valores por linha)
xF = (81*6)+1:1:82*6;  % Define o intervalo para a  ltima  linha (82   linha)

XriosP = [];

for i = 1:1:81
    tres_camargos = camargos(i, 1:3);
    tres_furnas = furnas(i, 1:3);
```

```
    ent = [tres_camargos, tres_furnas];
    XriosP = [XriosP ent];
end

xriosF = [camargos(82, 1:3), furnas(82, 1:3)];  % Valores da  ltima  linha ...
    (82  )

% Plotar os dados reais
figure;
plot(xP, XriosP, 'b', 'DisplayName', 'Valores Dados');
hold on;
plot(xF, xriosF, 'r', 'DisplayName', 'Valores Esperados');
xlabel('Meses');
ylabel('Vazao');
title('Vazao dos Rios Camargos e Furnas');
grid on;

% Plotar  simula  o
xS = 1:1:(82*6);
PsA = [camargos(1, 1:3)'; furnas(1, 1:3)'];
Ms = PsA;

% Loop para prever as vaz es
for i = 1:1:81
    PsD = sim(net, PsA);
    Ms = [Ms, PsD];
    PsA = PsD;
end

% Ajustar os dados simulados para o gr fico
yS = [];

for i = 1:size(Ms, 2)
    yS = [yS Ms(:, i)'];
end

% Plotar os resultados simulados
plot(xS, yS, ':m', 'LineWidth', 1.5, 'DisplayName', 'Resultados');
legend();
hold off;
```