Universidade Federal do Rio de Janeiro
Programa de Pós-graduação em Engenharia Mecânica - COPPE

Análise do Ruído em Aerofólios com Redes Neurais

Bruno da Silva Machado

10 de junho de 2025

Resumo

Neste trabalho, foi desenvolvida uma rede neural do tipo Multi-Layer Perceptron (MLP) com o objetivo de modelar a pressão sonora gerada por aerofólios, utilizando um conjunto de dados experimental da NASA. A abordagem envolveu a análise exploratória dos dados, com destaque para a identificação de outliers, correlação entre variáveis e normalização com escore Z. Em seguida, o modelo MLP foi treinado com TensorFlow, utilizando uma camada oculta de 10 neurônios, função de ativação sigmoide e otimização com decaimento exponencial da taxa de aprendizado. Avaliou-se o desempenho por meio de métricas como MSE, MAE, RAE e R^2, além de gráficos de resíduos e comparação com regressão linear. Também foram realizados experimentos para analisar a influência da complexidade da rede e a capacidade de extrapolação. Os resultados indicaram que a MLP supera modelos lineares tradicionais em capacidade preditiva, e embora a extrapolação represente um desafio para redes neurais, o modelo demonstrou estabilidade e coerência nas predições mesmo fora da faixa de treino.

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Sumário

Capítulo 1

Multi-Layer Perceptron

1.1 O que é o Multi-Layer Perceptron

O Multi-Layer Perceptron (MLP) é uma das principais arquiteturas de redes neurais adaptativas, composta por várias camadas de neurônios, cada uma com pesos ajustáveis[1][2]. Os MLPs são amplamente utilizados em tarefas de regressão e classificação, principalmente por sua capacidade de modelar relações não lineares complexas.

Um Multi-Layer Perceptron é composto por várias camadas chamadas de perceptrons, unidade básica de processamento. Um perceptron recebe um conjunto de entradas, realiza uma combinação linear dessas entradas com pesos associados e aplica uma função de ativação para gerar uma saída. Em um MLP cada perceptron é conectado à camada seguinte por meio de pesos ajustáveis[1].

Na Figura 1.1 mostramos a arquitetura de uma rede neural Multi-Layer Perceptron. A primeira camada (em azul) é a camada de entrada, que recebe os dados brutos. As camadas intermediárias (em vermelho) são chamadas de ocultas, pois não é possível prever a saída desejada nessas camadas intermediárias e a última camada (em amarelo) é a saída, onde os erros de aproximação são calculados e a resposta final do modelo é produzida[3].

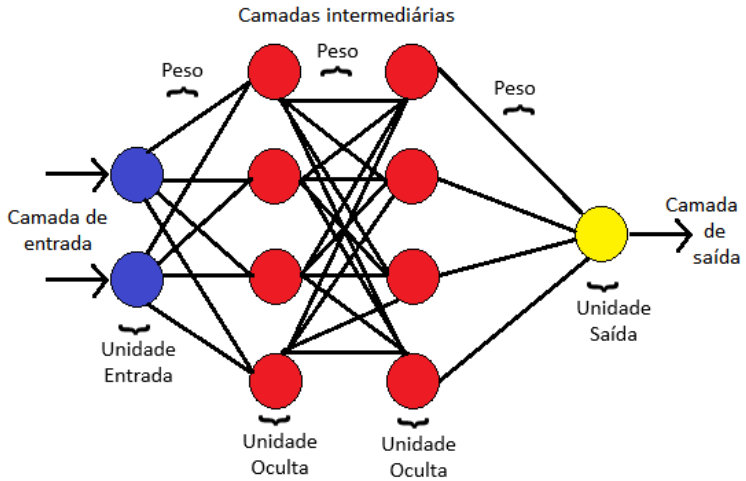


Figura 1.1: Diagrama de uma rede neural Multi-Layer Perceptron

1.2 Função de Ativação

Uma função de ativação é aplicada em cada perceptron para introduzir não linearidades no modelo. Isso permite que o MLP aprenda relações complexas nos dados. Alguns exemplos de funções de ativação comumente usadas são a função sigmoide, a função ReLU e a função tangente hiperbólica. Neste trabalho será utilizado a função sigmoide devido a sua simplicidade e versalidade.

1.2.1 Função Sigmoide

A função sigmoide ou logística é uma função de ativação dada por:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1.1}$$

Ela tem as seguintes propriedades:

- A função é infinitesimalmente suave e monótona.
- O intervalo da função é entre $(0, 1)$, ou seja, é uma função limitada. Esta função é utilizada na representação probabilística.
- Como a derivada decai rapidamente para zero a partir de $\epsilon = 0$, essa função de ativação pode levar a convergência lenta da rede durante o treinamento[4].

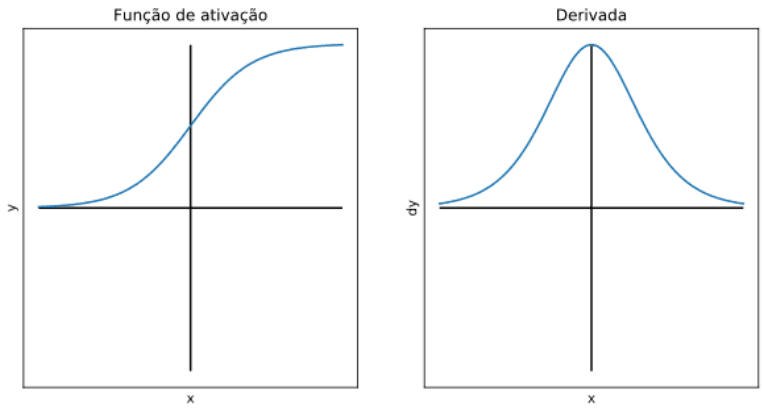


Figura 1.2: Função sigmoide e sua derivada.

Na Figura 1.2 a função Sigmoide mapeia todos os valores reais, x, para um intervalo limitado entre $(0, 1)$ de forma continua e suave. Além disso, a sua derivada é a função gaussiana que mapeia todos os pontos em uma distribuição e pode ser interpretado com uma distribuição de probabilidade, útil em problemas de classificação.

Entretanto, sua característica não-linear aumenta o custo computacional. Além disso, como observado na Figura 1.2, a função sigmoide não é centrada em zero. Não só isso, ela apresenta platôs para valores de x muito altos ou muito baixos, o que faz com que a derivada nessas regiões se aproxime de zero. A soma dessas características não faz da função sigmoide uma boa opção para ativação das camadas escondidas.

1.3 Regularização L2

A regularização é um conjunto de métodos para reduzir o overfitting em modelos de aprendizado de máquina. Normalmente, a regularização troca uma leve redução na precisão do treinamento por um aumento na capacidade de generalização[5].

A regularização L2 é uma técnica de regularização que penaliza coeficientes de alto valor ao introduzir um termo de penalização na função de custo[5].

A função de custo escolhida foi o Erro Quadrático Médio (MSE), pois penaliza erros maiores de forma mais significativa:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{1.2}$$

Para evitar overfitting, incluímos o termo de regularização L2:

$$J(\theta) = MSE + \lambda \sum \theta^2 \tag{1.3}$$

Portanto a função de custo é a soma do MSE com a regularização L2. A vantagem de construir a função custo dessa forma é corrigir o overfitting e a desvantagem é o risco de subajuste caso o modelo receba viés excessivo por meio da regularização.

1.4 Retropropagação de Erro (Backpropagation)

Comumente é utilizado o algoritmo de retropropagação do erro para treinar a rede neural MLP[3].

A ideia do algoritmo de retropropagação se baseia no cálculo do erro obtido na camada de saída da rede neural. Deve-se ajustar os valores dos pesos e limites conforme o erro e retro-propagado. O procedimento que algoritmo faz é calcular o erro entre o que foi obtido pela rede e o valor esperado, então ajustar os valores de todos os pesos, começando pelos pesos da camada de saída e retrocedendo até a entrada, sempre tentando minimizar este erro.

O algoritmo de retropropagação segue os seguintes etapas:

- Inicializa os pesos da rede com uma distribuição normal e uniforme de valores aleatórios;
- Fornece dados de entrada a rede e calcula o valor da função de erro obtida, ao comparar com o valor de saída esperado;
- Busca-se minimizar o valor da função de erro calculando os valores dos gradientes para cada peso da rede, uma vez que o gradiente fornece a direção de maior crescimento, bastando escolher o sentido contrário do gradiente;
- Com o vetor gradiente calculado, ajusta-se cada peso de maneira iterativa. Deve-se recalcular os gradientes a cada passo de iteração, até o erro atingir algum limiar, ou alcançar o número máximo de iterações.

1.4.1 Implementação do algoritmo de Retropropagação de Erro (Backpropagation)

Nessa seção, adaptamos a demonstração do algoritmo de backpropagation [6] para a arquitetura específica da rede utilizada neste projeto: uma rede neural com 5 neurônios de entrada, 10 neurônios na camada intermediária (camada oculta) e 1 neurônio de saída. A função de ativação utilizada é a sigmoide, e a função de erro é o erro quadrático médio (MSE).

A fórmula geral de atualização dos pesos em uma iteração do algoritmo é:

$$w \leftarrow w - \eta \frac{\partial E}{\partial w}, \tag{1.4}$$

ou seja, o valor do peso w na iteração atual será o valor anterior, corrigido de valor proporcional ao gradiente. O sinal negativo significa que estamos indo na direção contrária à do gradiente. O parâmetro η representa a taxa de aprendizado da rede neural, controlando a largura do passo na correção do peso.

Na equação (1.4), o principal conceito é o cálculo das derivadas parciais da função de erro em relação ao vetor de pesos w. A figura 1.3 temos uma rede perceptron multicamada com um camada intermediária. Uma junção entre um neurônio j e um neurônio i da camada seguinte possui peso w_{ij}. Os números sobrescritos, entre parênteses, indicam o número da camada a qual a variável pertence.

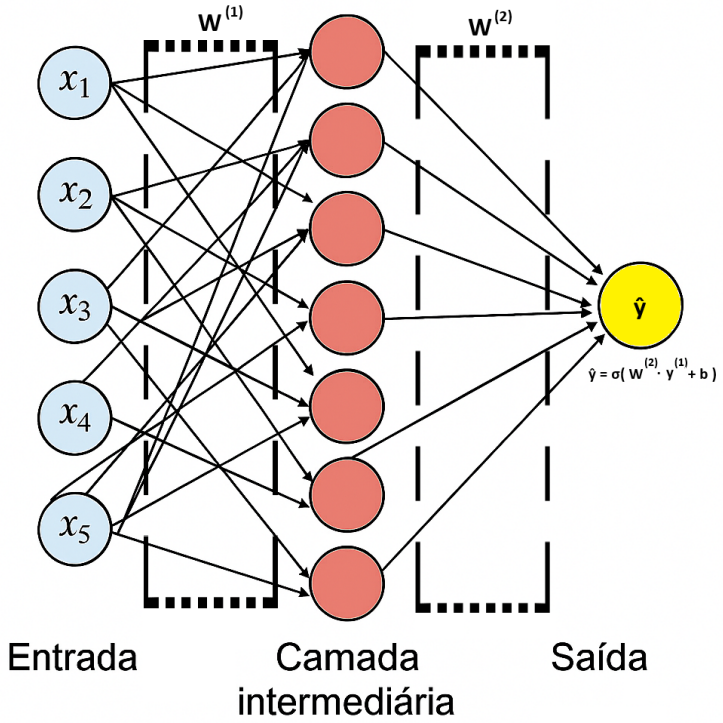


Figura 1.3: Implementação da rede neural MLP para a explicação da retropropagação

Na Figura 1.3 temos em cada circulo azul a representação das entadas do conjunto $\mathbf{x}$, os

círculos vermelhos representam as camadas intermediárias e em amarelo a saída $\hat{y} = \sigma(\mathbf{W}^{(2)} \cdot y^{(1)} + \mathbf{b}^{(2)})$

Considerando a arquitetura da rede: uma camada de entrada com 5 variáveis $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5)$, conectada a uma camada oculta com 10 neurônios. Os potenciais de ativação da camada oculta são:

$$z_j^{(1)} = \sum_{i=1}^{5} w_{i,j}^{(1)} x_i + b_j^{(1)}, \quad j = 1, ..., 10 \tag{1.5}$$

E suas saídas são:

$$y_j^{(1)} = \sigma(z_j^{(1)}) \tag{1.6}$$

A camada de saída é composta por um único neurônio, com entrada total:

$$z^{(2)} = \sum_{j=1}^{10} w_j^{(2)} y_j^{(1)} + b^{(2)} \tag{1.7}$$

E saída final:

$$\hat{y} = \sigma(z^{(2)}) \tag{1.8}$$

O erro quadrático médio é definido como $\sum_i^N (y_i - \hat{y}_i)^2/\mathrm{N}$. Entretanto temos apenas duas saídas então podemos simplificar o erro quadrático médio para uma amostra:

$$E = \frac{1}{2}(y - \hat{y})^2 \tag{1.9}$$

Aplicando a regra da cadeia, calculamos a derivada do erro em relação à saída predita:

$$\frac{\partial E}{\partial \hat{y}} = -(y - \hat{y}) \tag{1.10}$$

Como $\hat{y} = \sigma(z^{(2)})$, temos:

$$\frac{\partial E}{\partial z^{(2)}} = \frac{\partial E}{\partial \hat{y}} \cdot \sigma'(z^{(2)}) = -(y - \hat{y}) \cdot \hat{y}(1 - \hat{y}) \tag{1.11}$$

Chamamos esse termo de $\delta^{(2)}$:

$$\delta^{(2)} = -(y - \hat{y}) \cdot \hat{y}(1 - \hat{y}) \tag{1.12}$$

Calculamos as derivadas parciais do erro em relação aos pesos e viés da camada de saída:

$$\frac{\partial E}{\partial w_j^{(2)}} = \delta^{(2)} y_j^{(1)}, \quad \frac{\partial E}{\partial b^{(2)}} = \delta^{(2)} \tag{1.13}$$

O que gera as seguintes atualizações:

$$w_j^{(2)} \leftarrow w_j^{(2)} - \eta \delta^{(2)} y_j^{(1)}, \quad b^{(2)} \leftarrow b^{(2)} - \eta \delta^{(2)} \tag{1.14}$$

Agora, propagamos o erro para a camada oculta. Para cada neurônio oculto j, temos:

$$\delta_j^{(1)} = \delta^{(2)} w_j^{(2)} \cdot \sigma'(z_j^{(1)}) = \delta^{(2)} w_j^{(2)} y_j^{(1)} (1 - y_j^{(1)}) \tag{1.15}$$

Derivadas parciais do erro em relação aos pesos e viés da camada oculta:

$$\frac{\partial E}{\partial w_{i,j}^{(1)}} = \delta_j^{(1)} x_i, \quad \frac{\partial E}{\partial b_j^{(1)}} = \delta_j^{(1)} \tag{1.16}$$

Atualizamos os pesos da camada oculta:

$$w_{i,j}^{(1)} \leftarrow w_{i,j}^{(1)} - \eta \delta_j^{(1)} x_i, \quad b_j^{(1)} \leftarrow b_j^{(1)} - \eta \delta_j^{(1)} \tag{1.17}$$

Com essas equações, implementamos de forma sistemática o algoritmo de retropropagação adaptado à estrutura da rede neural deste projeto, com propagação do erro e atualização dos pesos desde a camada de saída até as entradas.

Capítulo 2

Introdução ao TensorFlow

Na seção 1.4.1 implementamos um algoritmo para a retropropagação do erro. Apesar disso, nesse trabalho foi utilizado a biblioteca TensorFlow para implementar e treinar a rede neural MLP.

2.1 O que é o TensorFlow

O TensorFlow é uma biblioteca de código aberto desenvolvida pela equipe do Google Brain, projetada para facilitar o desenvolvimento e a execução de algoritmos de aprendizado de máquina, especialmente redes neurais profundas. Ele permite a construção de modelos computacionais usando grafos de fluxo de dados, onde os nós representam operações matemáticas e as arestas representam tensores que são estruturas de dados multidimensionais [7].

A principal vantagem do TensorFlow está na flexibilidade. Ele pode ser utilizado em ambientes de produção com alto desempenho (via GPU e TPU), bem como em dispositivos móveis e navegadores, graças ao TensorFlow Lite e TensorFlow.js [8]. Além disso, é amplamente utilizado tanto na indústria quanto na academia para tarefas de classificação, regressão, processamento de linguagem natural, visão computacional, entre outras.

2.2 Multi-Layer Perceptron com TensorFlow

Um *Multi-Layer Perceptron* (MLP) é uma rede neural *feedforward* composta por camadas densamente conectadas (ou densas), normalmente contendo uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. Cada camada é composta por neurônios que aplicam uma transformação linear seguida de uma função de ativação não linear, como ReLU ou Sigmoid.

No TensorFlow, um MLP pode ser implementado usando a API `tf.keras.Sequential`, que permite empilhar camadas de maneira simples e direta. Por exemplo:

Código 2.1: implementação MLP no tensorflow

```
1    import tensorflow as tf
2    from tensorflow.keras.models import Sequential
3    from tensorflow.keras.layers import Dense
4
```

```
model = Sequential([
Dense(10, activation='sigmoid', input_shape=(X.shape[1],)),
Dense(1, activation='linear')  # saída contínua
])
```

No código 2.1 o modelo é criado pela classe `Sequential()` que define um conjunto de camadas lineares. A camada `Dense(10, activation='sigmoid')` possui 10 neurônios e utiliza a função de ativação sigmoide. O parâmetro `input_shape` define a dimensão da entrada. A camada de saída tem um único neurônio de saída, com ativação `linear`, adequada para regressão.

Após a definição da arquitetura, o modelo é compilado e treinado:

Código 2.2: Compilar e Treinar o Modelo

```
model.compile(optimizer='adam',loss='mse',metrics=['mae'])

model.fit(x_train, y_train, epochs=200, batch_size=32)
```

No método `modelo.compile()` o otimizador `Adam` é utilizado por ser eficiente e adaptativo. A função de perda `mse` é o erro quadrático médio definido na equação 1.2. O treinamento é realizado por 200 épocas com mini-lotes de 32 amostras.

O TensorFlow gerencia automaticamente os gradientes e a retropropagação (*backpropagation*), simplificando o treinamento do modelo [9, 10].

2.3 Taxa de Aprendizado com Decaimento Exponencial

A taxa de aprendizado é um dos hiperparâmetros mais importantes no treinamento de redes neurais. Ela define o tamanho dos passos que o otimizador dá em direção ao mínimo da função de perda. Uma taxa muito alta pode fazer o modelo divergir, enquanto uma taxa muito baixa pode torná-lo extremamente lento para convergir.

Para lidar com essa sensibilidade, o TensorFlow oferece mecanismos para alterar dinamicamente a taxa de aprendizado durante o treinamento. Um método comum é o *decaimento exponencial* (*exponential decay*), onde a taxa de aprendizado diminui exponencialmente ao longo do tempo. Isso permite passos maiores no início do treinamento e passos mais precisos na fase final de ajuste fino.

No TensorFlow, isso pode ser implementado com:

Código 2.3: Decaimento exponencial

```
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
initial_learning_rate=0.01,
decay_steps=100000,
decay_rate=0.96,
staircase=True)

optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule)
```

Onde

- `initial_learning_rate=0.01` define a taxa inicial de aprendizado.
- `decay_steps=100000` indica que a taxa será atualizada a cada 100.000 passos de treinamento.

- `decay_rate=0.96` significa que a taxa de aprendizado será multiplicada por 0.96 a cada `decay_steps`.
- `staircase=True` faz com que o decaimento ocorra em degraus (em vez de contínuo), criando uma curva em escada.
- O otimizador Adam é então criado utilizando esse agendador de taxa de aprendizado.

Esse tipo de abordagem ajuda a obter um treinamento mais estável e eficaz, especialmente em redes neurais profundas [11].

Capítulo 3

Resultados

Neste projeto, foi construído uma rede neural para analisar e modelar um conjunto de dados experimentais produzido pelo NASA no final dos anos 80. Esses dados foram produzidos em um túnel de vento e visavam compreender a produção de ruído no entorno de uma asa. Na mecânica dos fluídos o problema do ruído em aerofólios está relacionado à geração de ruído por meio de fenômenos aerodinâmicos, como o escoamento turbulento e a separação da camada limite.

Essa seção foi dividida em duas parte: A primeira parte foram realizadas análises na descrição estatística dos dados, nos gráficos de dispersão e histogramas, na correlação dos dados e por fim na localização de outliers e normalização dos dados. A segunda parte foi realizado o treinamento e validação da rede neural MLP, além de avaliar três cenários possíveis que são a variação do número de neurônios, a comparação da rede neural com outros modelos e a extrapolação dos dados.

3.1 Análise dos Dados

O conjunto de dados da NASA compreende aerofólios NACA 0012 [12] de tamanhos diferentes em várias velocidades de túnel de vento e ângulos de ataque. A envergadura do aerofólio e a posição do observador foram as mesmas em todos os experimentos. Não há valores ausentes nesse conjunto de dados.

Informações sobre as variáveis.

Esse problema tem as seguintes entradas:

- Frequência, em Hertzs.
- Ângulo de ataque, em graus.
- Comprimento da corda, em metros.
- Velocidade de fluxo livre, em metros por segundo.
- Espessura do deslocamento do lado da sucção, nos medidores.

A única saída é: Nível de pressão sonora escalonado, em decibéis.

Na Tabela 3.1 temos a análise descritiva do conjunto de dados nela podemos verificar algumas informações uteis dos dados antes de realizar o treinamento da rede neural.

Tabela 3.1: Estatísticas descritivas das variáveis

	Frequência	**Angulo de Ataque**	**Comprimento da Corda**	**Velocidade do Fluxo**	**Espessura Sucção**	**Pressão Sonora**
média	2886,380	6,782	0,136	50,860	0,011	124,835
std	3152,573	5,918	0,094	15,572	0,013150	6,899
min	200,000	0,000	0,0254	31,700	0,000401	103,380
25%	800,000	2,000	0,051	39,600	0,002535	120,191
50%	1600,000	5,400	0,102	39,600	0,004957	125,721
75%	4000,000	9,900	0,229	71,300	0,0156	129,995
máx	20000,000	22,200	0,305	71,300	0,0584	140,987

Na Figura 3.1 complementamos a analise com gráficos de boxplots. Um boxplot é um diagrama de caixa que mostra o resumo das propriedades de um conjunto de valores numéricos como mínimo, primeiro quartil, mediana, terceiro quartil e máximo.

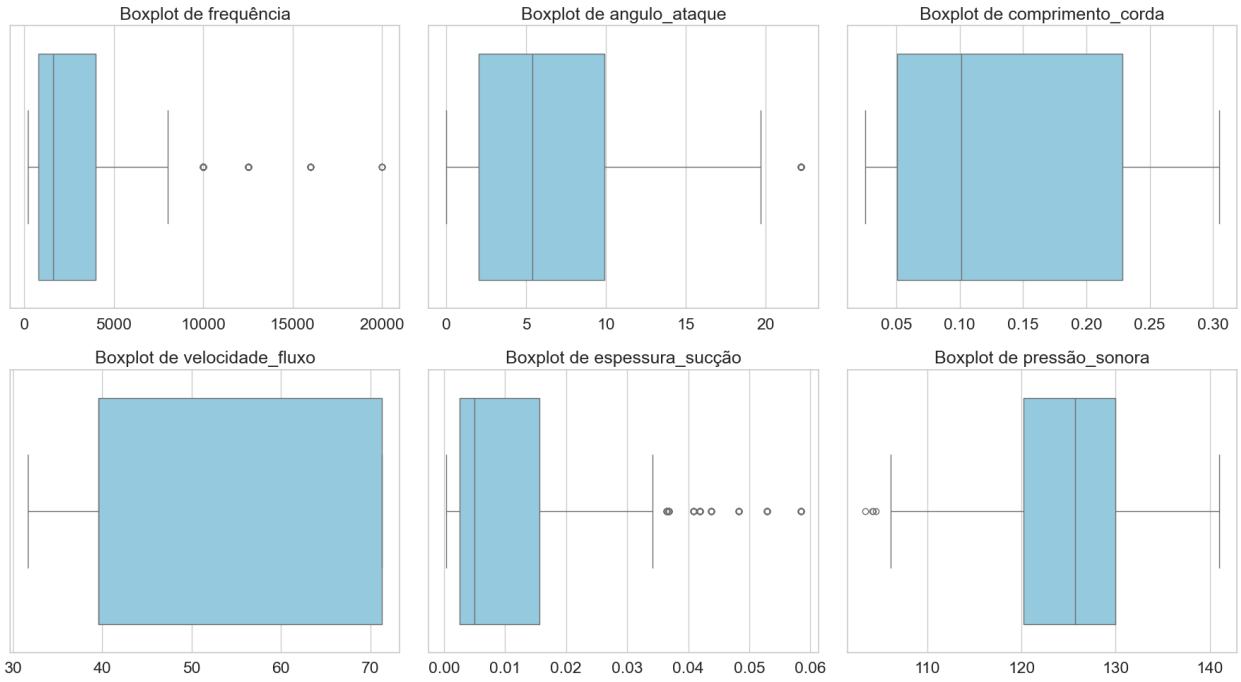


Figura 3.1: Boxplot do conjunto de dados

Analisando os valores da coluna `Frequência` temos uma média de $2886,38$ Hz e um desvio padrão (`std`) de aproximadamente $3152,57$ Hz isso indica uma alta dispersão e um desvio muito elevado, além disso o intervalo é de $200,00$ Hz a $20.000,00$ Hz, ou seja, há uma grande amplitude. Na Figura 3.1a podemos verificar que há vários outliers no extremo superior.

Na coluna `Ângulo de ataque` temos uma média de $6,782°$ e um desvio padrão de $5,918°$ o que é uma dispersão moderada. A mediana de $5,4°$ é um pouco menor que a média, indicando

possível leve assimetria à direita. Na Figura 3.1b observamos uma distribuição compacta, sem outliers visíveis e dados bem comportados.

Na coluna `Comprimento da corda` a mediana de $0,102$ m é menor que a média ($0,136$ m), sugerindo presença de valores maiores extremos. Na Figura 3.1c não consta outliers relevantes e variação pequena.

Na coluna `Velocidade do fluxo` temos um comportamento curioso: A velocidade mínima é de $31,7$ m/s, mas o valor de 75% é igual ao máximo ($71,3$ m/s), indicando uma possível concentração no valor máximo. Na Figura 3.1d constamos uma distribuição bem centrada, sem outliers aparentes.

Na coluna `Espessura de sucção` a média é muito maior que a mediana ($0,004957$ m), o que indica uma forte assimetria à direita, ou seja, valores extremos altos. Alguns outliers são visíveis no lado superior.

Na coluna `Pressão sonora` a mediana ($125,721$ dB) é um pouco maior que a média ($124,835$ dB), sugerindo uma leve assimetria à esquerda. Alguns poucos outliers inferiores, mas a maioria dos dados está bem distribuída.

3.1.1 Gráficos de dispersão com LOWESS

Utilizamos gráficos de dispersão para mostra os valores de duas variáveis, permitindo visualizar padrões, tendências e dispersões. No contexto de regressão, queremos ver como cada variável de entrada se relaciona com a variável de saída (pressão).

Para aprofundar a análise foi utilizado o LOWESS (ou LOESS) que significa *Locally Weighted Scatterplot Smoothing*. É um método não paramétrico que ajusta uma curva suave sobre os dados no gráfico de dispersão ao realizar regressões locais ponderadas, revelando relações não lineares entre as variáveis. o LOWESS não assume uma forma específica da relação (linear, polinomial, etc.) e a curva é ajustada localmente em torno de cada ponto. Assim devemos interpretar os gráficos da seguinte forma:

- Se a curva não for reta, isso indica uma relação não linear.
- Uma curva descendente indica que, à medida que a variável aumenta, a pressão sonora tende a diminuir (e vice-versa).
- Ondulações ou curvaturas mais complexas sugerem relações mais complicadas, que redes neurais conseguem modelar melhor do que modelos lineares.

Na Figura 3.2 a curva mostra uma clara tendência decrescente. Isso confirma a correlação negativa moderada: conforme a frequência aumenta, a pressão sonora tende a diminuir. Relação não linear suave, com declínio mais acentuado em frequências mais baixas.

Na Figura 3.3a a curva tem uma leve queda, mas quase plana. Confirma que há pouca influência direta, talvez com pequenas oscilações não lineares. Pode indicar que esta variável tem influência condicional em interação com outras variáveis.

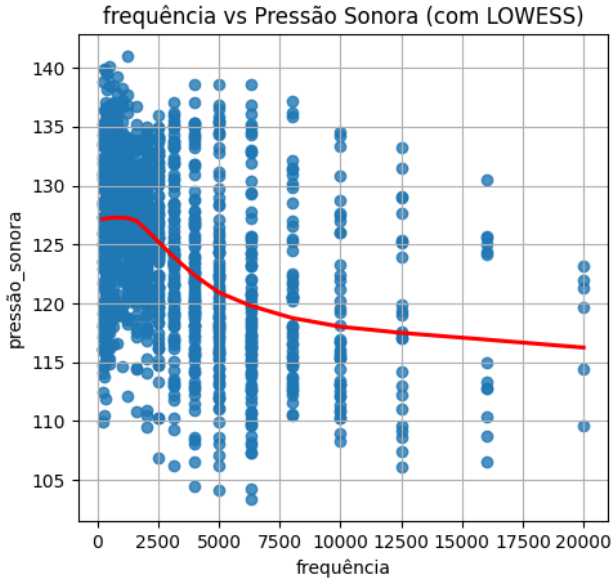
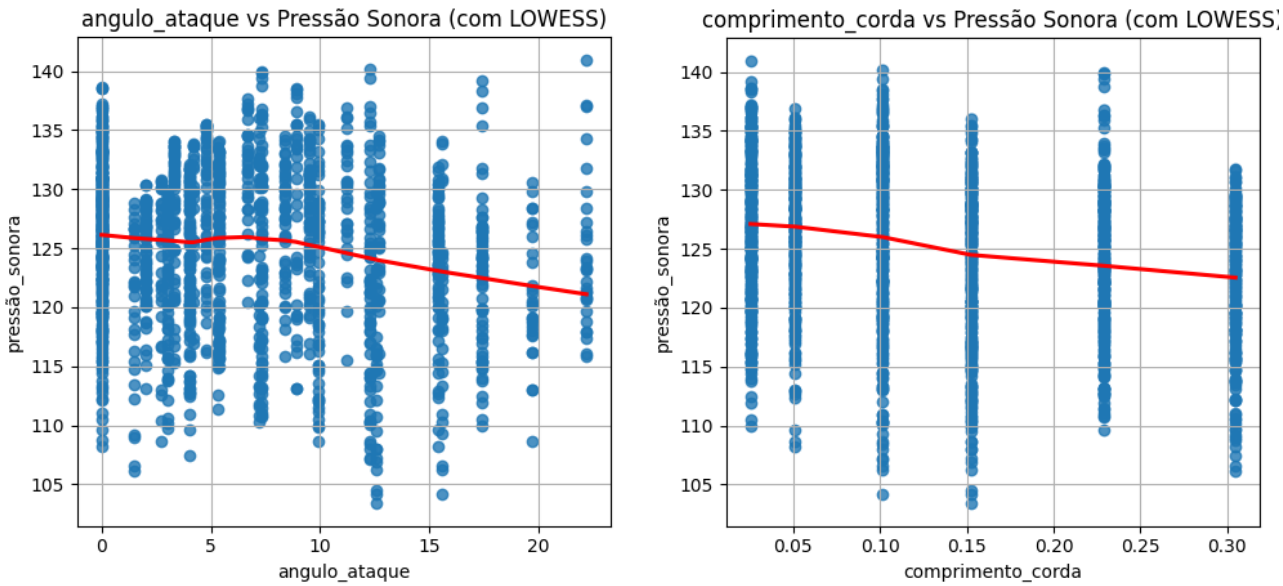


Figura 3.2: Frequência (Hz) vs Pressão Sonora (dB)

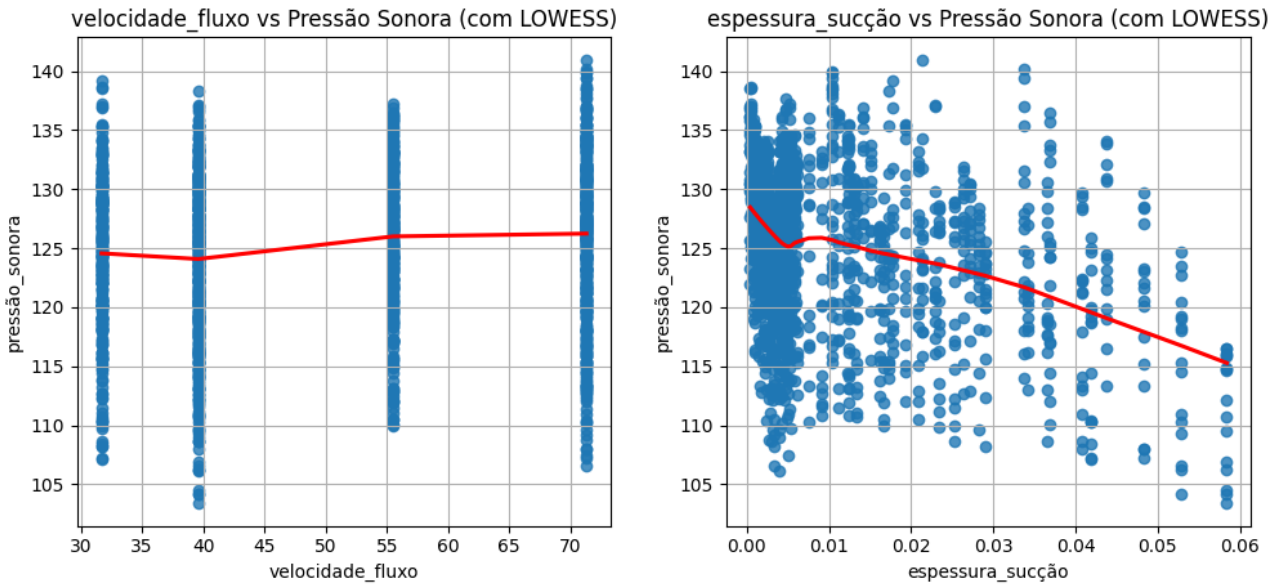
Na Figura 3.3b a relação é ligeiramente não linear, com uma tendência decrescente. Sugere que aerofólios com corda maior tendem a produzir um pouco menos de ruído. A não linearidade é suave, o que pode ser modelado por redes neurais.



(a) Ângulo (graus) vs Pressão Sonora (dB) (b) Comprimento (m) vs Pressão Sonora (dB)

Figura 3.3: Ângulo de Ataque, Comprimento da Corda vs Pressão Sonora

Na Figura 3.4a a curva é ligeiramente ascendente, mas bem suave. Mostra uma influência fraca, mas crescente da velocidade no aumento da pressão sonora. Reforça a correlação positiva fraca vista anteriormente.



(a) Velocidade (m/s) vs Pressão Sonora (dB) (b) Espessura (m) vs Pressão Sonora (dB)

Figura 3.4: Velocidade de Fluxo e Espessura de Sucção vs Pressão Sonora

Na Figura 3.4b a curva é descendente, com alguma variação suave. Mostra uma relação inversa não linear: maior espessura tende a reduzir o ruído. Pode ser uma variável significativa no modelo.

Ao analisar as Figuras 3.3a até 3.4b inferimos que as variáveis `frequência` e `espessura de sucção` apresentam relações não lineares moderadamente fortes com a saída e devem ser valorizadas no modelo. As outras variáveis têm relações fracas, mas ainda assim podem ser relevantes em interações não lineares, que a rede neural pode capturar bem.

3.1.2 Correlação dos Dados

Em estatística, a correlação é qualquer relação estatística, causal ou não, entre duas variáveis aleatórias. Segundo Silva[13], a correlação entre duas variáveis denota o grau em que as variáveis estão linearmente relacionadas. Ou seja, é uma medida de quanto as variáveis se relacionam em termos de força e direção.

Matriz de Correlação de Pearson

A matriz de correlação de Pearson é uma tabela que mostra o coeficiente de correlação de Pearson entre pares de variáveis numéricas. Cada valor na matriz mede a intensidade e direção da relação linear entre duas variáveis.

O **Coeficiente de Pearson (r)** é um valor entre -1 e $+1$, onde :

- $1 \rightarrow$ correlação linear positiva perfeita: quando uma variável aumenta, a outra também aumenta.
- $-1 \rightarrow$ correlação linear negativa perfeita: quando uma variável aumenta, a outra diminui.

- $0 \rightarrow$ sem correlação linear: não há tendência linear entre as variáveis.

Existe mais de um método diferente de medir a relação entre duas variáveis, mas o mais conhecido e utilizado é o coeficiente de correlação de Pearson que avalia a relação linear entre variáveis da seguinte forma[13]:

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}} \tag{3.1}$$

onde $\bar{x}$, $\bar{y}$: médias das variáveis.

O numerador é a covariância entre x e y. O denominador normaliza pela variabilidade (desvio padrão) de cada variável.

Quando temos várias variáveis $x_1, x_2, \ldots, x_n$, a matriz de correlação é uma matriz $n \times n$ onde a posição (i, j) representa a correlação entre x_i e x_j e a diagonal é sempre 1 (uma variável é perfeitamente correlacionada com ela mesma).

A matriz de correlação de Person é confiável para entender como os dados se comportam e é extremamente útil para identificar relações fortes entre entradas e saída. Além disso, Pode revelar colinearidade entre entradas, o que pode causar problemas em alguns modelos.

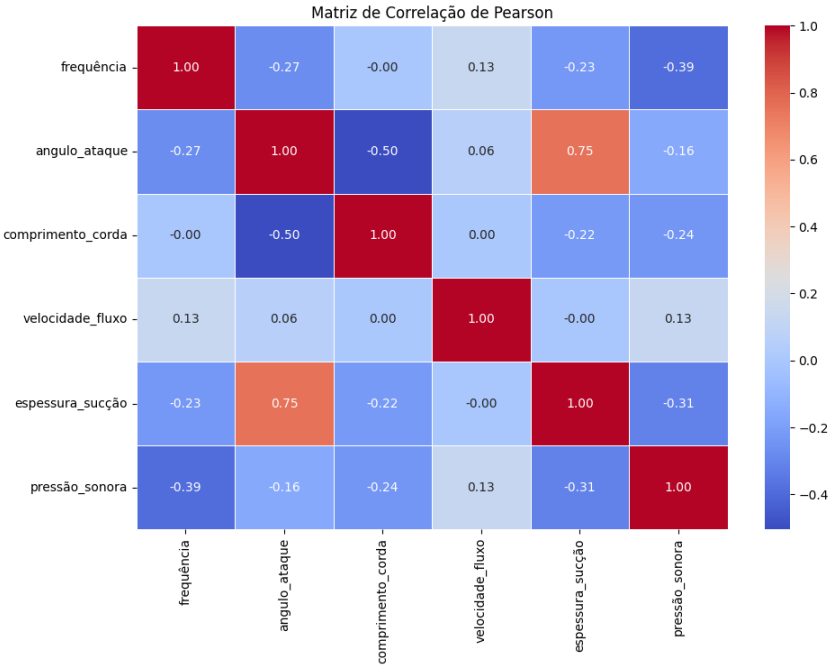


Figura 3.5: Matriz de correlação de Person.

Na Figura 3.5 matriz de correlação revelou que as variáveis frequência e espessura de sucção têm maior correlação com a pressão sonora. Além disso, inferimos as seguintes correlações com a variável alvo pressão sonora. `Frequência`: correlação de $-0,39$, portanto avaliada como moderada e negativa. Isso significa que quanto maior a frequência, menor tende a ser a pressão sonora. Isso indica que frequência é relevante, com tendência inversa.

`Ângulo de ataque`: correlação de $-0,16$, ou seja, fraca e negativa. A influência do ângulo de ataque na pressão sonora é fraca, mas há uma leve tendência inversa.

`Comprimento da corda`: correlação de $-0,24$, isto é, entre fraca a moderada negativa. Exite alguma influência da corda na redução da pressão sonora.

`Velocidade de fluxo`: correlação $0,13$, logo avaliada como fraca e positiva. Têm pouca influência direta, mas há uma leve tendência de aumento da pressão sonora com velocidade.

`Espessura de sucção`: correlação de $-0,31$, logo moderada e negativa. Mostra uma influência inversa: deslocamentos maiores tendem a reduzir a pressão sonora.

Portanto as variáveis mais relevantes, com base na correlação linear, são: `Frequência` (correlação negativa moderada), a `Espessura de sucção` (correlação negativa moderada) e `Comprimento da corda` (correlação fraca a moderada negativa).

3.1.3 Outliers e Normalização

Outliers são valores extremos que podem distorcer o treinamento de modelos. Detectá-los ajuda a entender a confiabilidade dos dados.

Verificá-los é importante porque podem distorcer o aprendizado da rede neural, fazendo com que o modelo foque demais nesses pontos raros. Além disso podem influenciar estatísticas como média e variância, prejudicando normalizações. Algumas redes neurais (especialmente com sigmoide ou tanh) são sensíveis a variações extremas, pois suas ativações saturam.

Normalização

Normalizar os dados (trazer para uma mesma escala) é essencial para evitar que variáveis com grande escala (ex: frequência em Hz) dominem as de escala pequena (ex: comprimento em m). Além disso, acelera o treinamento, melhora a convergência do otimizador e deixa o espaço de entrada mais bem condicionado para modelos como redes neurais.

Durante a análise dos dados foi constatado que há regiões pouco cobertas nos dados, principalmente em valores extremos de frequência e espessura, o que pode comprometer a generalização do modelo. A normalização contribui para uma cobertura mais abrangente e confiável dos dados ao eliminar anomalias e otimizar a forma como os dados são tratados.

Na etapa de pré-processamento, um dos objetivos foi identificar possíveis outliers que pudessem afetar o desempenho do modelo. Para isso, utilizamos o escore Z (Z-score), que mede a distância de cada valor em relação à média da variável, em unidades de desvio padrão:

$$z = \frac{x - \mu}{\sigma} \tag{3.2}$$

Onde x é o valor observado, μ é a média e σ é o desvio padrão da variável. Valores de $|z| > 3$ são comumente considerados outliers. Essa escolha baseia-se em dois fundamentos:

- **Distribuição Normal:** em uma distribuição aproximadamente normal, cerca de 99,7% dos dados estão dentro de três desvios padrão da média. Logo, valores fora desse intervalo são estatisticamente raros.

- **Teorema de Chebyshev[14]:** independentemente da distribuição, ao menos 88,9% dos dados devem estar dentro de $\pm 3\sigma$. Assim, $|z| > 3$ representa uma abordagem conservadora e robusta para identificação de valores extremos.

Portanto, o critério $|z| > 3$ é uma prática estatisticamente fundamentada para diagnóstico de outliers, sendo particularmente útil em bases de dados experimentais como esta. Na tabela 3.2 realizamos a contagem de outliers por variáveis com base no Z-score $|z| > 3$.

Tabela 3.2: Contagem de outliers por variável com base no Z-score

Variável	**Outliers**
Frequência	44
Ângulo de ataque	0
Comprimento da corda	0
Velocidade do fluxo livre	0
Espessura de sucção	32
Pressão sonora	2

3.2 Treinamento do Multi-Layer Perceptron

Na Seção 1, apresentamos os fundamentos teóricos do perceptron multicamada (MLP), e na Seção 1.4.1, desenvolvemos a dedução formal do algoritmo de retropropagação de erro aplicado à arquitetura da nossa rede, composta por 5 entradas, 10 neurônios na camada intermediária e 1 saída. Para a implementação e treinamento do modelo, utilizamos a biblioteca TensorFlow, amplamente consolidada no meio acadêmico e em aplicações de produção. A construção prática da MLP com TensorFlow foi detalhada na Seção 2.2, e o código completo encontra-se disponível no Apêndice A.

O modelo desenvolvido nesse trabalho possui as seguintes configurações:

- 1 camada intermediária com 10 neurônios;
- Função de ativação sigmoide;
- Otimizador Adam com taxa de aprendizado com decaimento exponencial;
- Função de perda Erro Quadrático Médio (MSE);
- Mini-batch com 32 amostras;
- Treinamento realizado em 1000 épocas.

Após o treinamento as seguintes métricas foram utilizadas para avaliar o modelo:

Erro Quadrático Médio (MSE) : Definida na Equação (1.2), o MSE calcula a média dos quadrados das diferenças entre os valores previstos e reais. Quanto menor o valor do MSE, melhor a qualidade do modelo, indicando menor erro entre as previsões e os valores observados.

Erro Absoluto Médio (MAE) : é uma métrica usada para medir a precisão de um modelo de regressão.

Erro Absoluto Médio Relativo (RAE) : é uma métrica que mede a média dos erros absolutos relativos entre os valores previstos e reais. Também é utilizado para avaliar a precisão de modelos preditivos, especialmente em situações onde é importante considerar a proporção do erro em relação ao valor real.

Coeficiente de Determinação (R^2) : é uma medida estatística que representa a qualidade do ajuste de um modelo de regressão. O valor do R^2 está entre 0 e 1. Valor próximo de 1 indica um bom ajuste, enquanto próximo de 0 indica ajuste ruim.[15]

Além disso, foram gerados gráficos de resíduos e de valores reais vs preditos. Esses resultados mostram o desempenho do modelo e seu comportamento.

Na Tabela 3.3 temos o resultado do treinamento para as seguintes métricas: MSE, MAE, R^2 e RAE. Nela podemos observar que o MSE é baixo o que significa que temos um bom modelo. Isso é confirmado pelo MAE e RAE que estão próximos de zero. No caso do RAE valores baixos significa que a previsão do modelo é melhor que calcular a média. Para R^2 próximo de 1 significa que modelo explica bem a variabilidade.

Tabela 3.3: Resultado do treinamento

Métrica	**Valor**
MSE	0,1595
MAE	0,2929
RAE	0,3481
R^2	0,8485

Na Figura 3.6 temos dois gráficos o primeiro faz comparação entre a função de perda (MSE) e a validação cruzada. O segundo tem o mesmo objetivo, mas compara o MAE com a validação cruzada. A validação cruzada é útil para avaliar o desempenho do modelo e detecta a presença de *overfitting*.

Na Figura 3.6 ambos os gráficos apresentaram uma curva decrescente ao longo do treinamento e não só isso, nas primeiras iterações o valor do erro no treinamento é maior, mas ao longo das iterações esse valor cai abaixo da validação cruzada. Esse comportamento já era esperado, pois a validação mede o desempenho em dados não vistos. Como a diferença entre as duas curvas é pequena o modelo está se ajustando melhor ao treino do que à validação.

Na Figura 3.7 temos o gráfico de resíduo que mostra a diferença entre os valores reais e os preditos pelo modelo. Ele é definido por resíduo = $(y_{real} - y_{predito})$. No eixo horizontal são representados os valores preditos e no eixo vertical são representados os resíduos.

Os valores estão centrados em zero sem desvios aparente, isso significa que o modelo não apresenta vieses, ou seja, não tende a errar mais para cima ou para baixo. A maioria dos pontos estão dentro de uma amplitude entre -1 e 1, isso significa que a maioria dos erros está dentro de 1 unidade da saída real, o que sugere boa precisão nas previsões.

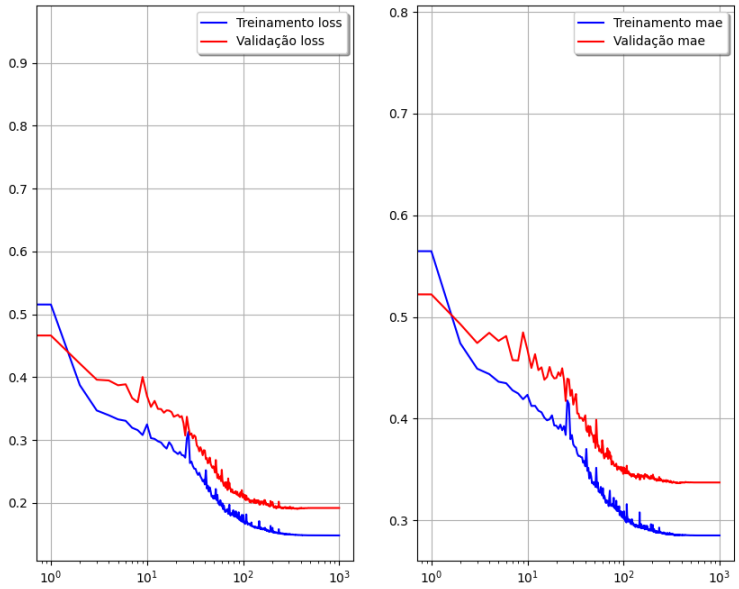


Figura 3.6: Treino e validação.

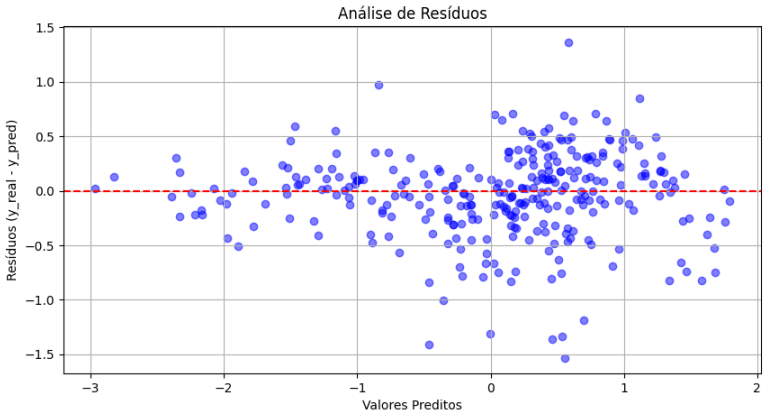


Figura 3.7: Resíduos ($y_{real} - y_{predito}$)

Na Figura 3.8 apresentamos um gráfico comparando os valores reais com os preditos. A linha tracejada é o caso ideal, ou seja, todos os valores coincidem. O objetivo é avaliar a dispersão dos valores ao redor do valor ideal. Como podemos observar a maioria dos pontos estão próximos da linha diagonal,ou seja, o modelo está fazendo boas previsões. Desvios grandes da linha indicam erros sistemáticos ou outliers. Os pontos não estão bem próximos dessa linha, isso implica que o modelo não tem boa precisão global.

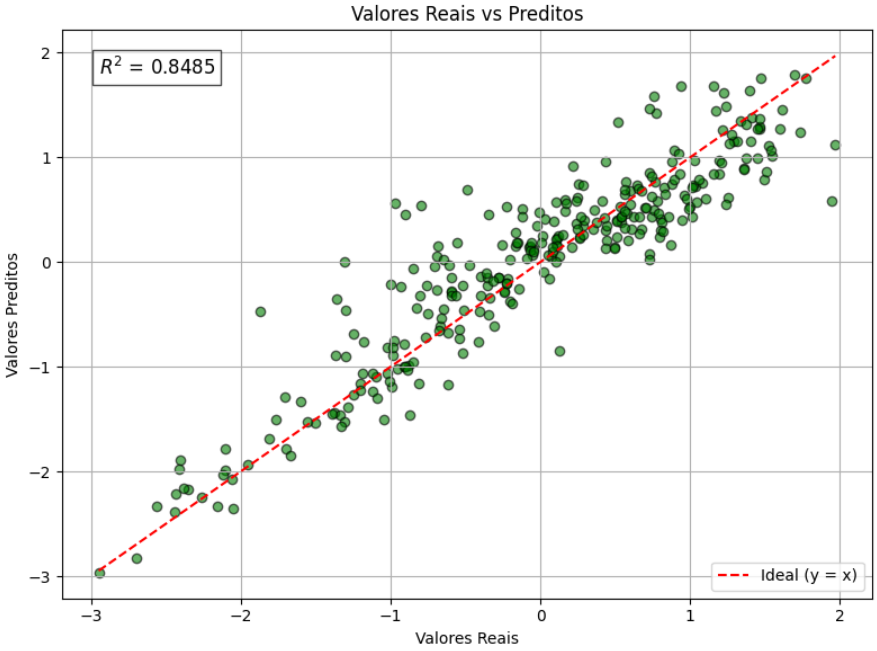


Figura 3.8: Valores Reais vs Preditos.

3.2.1 Experimentos

Experimento 1: Variação do número de neurônios O objetivo é verificar em que momento adicionar mais neurônios na camada intermediária deixa de melhorar o desempenho. Na Figura 3.9 observou-se que após certo limite entre 10^2 e 10^3, o R^2 se manteve estável.Após 10^3 os valores de R^2 começaram a oscilar, indicando risco de overfitting. Para evitar subajuste nos casos onde o número de neurônios é grande decidimos fixar um limiar (*threshold*) ao invés do número de épocas. O limiar escolhido foi o erro absoluto médio menor que $0,1$, essa escolha seguiu o seguinte critério: Como o nível de pressão sonora varia entre aproximadamente 103 dB e 141 dB, com desvio padrão de $6,9$ dB, um erro médio absoluto de $0,1$ dB representa apenas $1,4\%$ da variabilidade típica. Isso garante que o modelo teve números de épocas o suficientes para convergir.

A partir da Figura 3.9 constatamos que a oscila no gráfico, quer dizer que aumentar a capacidade não melhora mais o modelo. Isso pode indicar que o problema tem baixa complexidade e poucos neurônios são suficientes. Nesse caso excesso de neurônios pode gerar *overfitting* ou ineficiência computacional.

Experimento 2: Comparar MLP e Regressão Linear Nesse experimento foi avaliado se usar uma rede neural realmente oferece vantagens em relação a um modelo clássico mais simples, como a regressão linear.

A partir do mesmo conjunto de teste foi avaliado duas métricas que são MSE e o R^2. Se o R^2 da MLP for significativamente maior, então vale a pena usar a rede neural. Caso contrário se a diferença for pequena, a simplicidade da regressão linear pode ser preferível.

Os resultados comparando o desempenho entre MLP e a regressão linear está na Tabela 3.4:

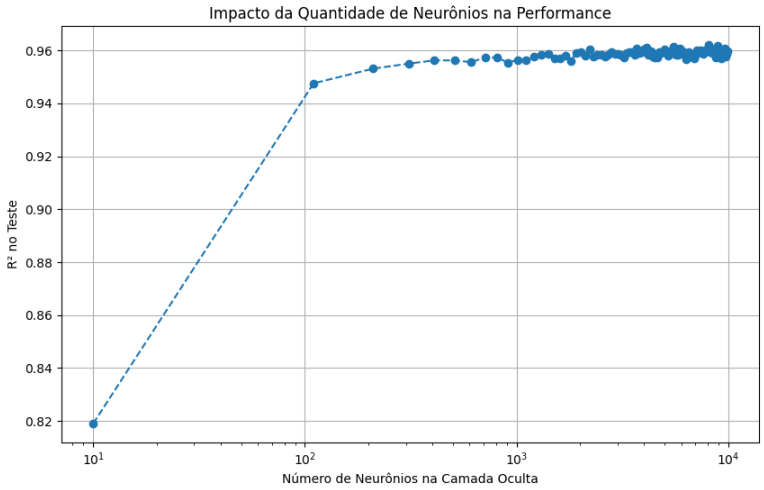


Figura 3.9: Treino e validação.

Tabela 3.4: Comparação MLP vs Regressão Linear

	MLP	**Regressão Linear**
MSE	0,1562	0,4650
R^2	0,8516	0,5583

Os resultados mostraram que a MLP superou a regressão linear em termos de R^2, validando seu uso para esse problema. A regressão linear é indicada para casos em que a relação linear é forte. Mas ao longo da nossa análise foi constatado que os dados possuem uma relação não linear e padrões complexos, caso ideal para utilizar uma rede neural.

Experimento 3: Extrapolação com Dados Fora do Domínio Para avaliar a capacidade de generalização do modelo em situações que não foram observadas durante o treinamento, realizamos um experimento de extrapolação. Foram geradas 10 novas amostras de entrada, com valores de atributos propositalmente situados fora do intervalo dos dados originais. Nesse trabalho, foram utilizadas frequências entre 22.000 e 30.000 Hz, ângulos de ataque entre 15° e 25°, e velocidades de fluxo superiores a 70 m/s.

Essas amostras foram normalizadas utilizando a média e o desvio padrão do conjunto de treino original, e então submetidas à rede neural previamente treinada. As predições resultantes foram:

Amostra	1	2	3	4	5	6	7	8	9	10
Predição (normalizada)	0,135	0,357	0,579	0,801	1,038	1,280	1,522	1,765	2,007	2,250

Aplicando a transformação inversa do Z-score com $\mu = 124{,}84$ e $\sigma = 6{,}74$, as predições foram convertidas para a escala original de decibéis, resultando em:

Amostra	1	2	3	4	5	6	7	8	9	10
Predição (dB)	125,76	127,25	128,74	130,23	131,84	133,46	135,07	136,69	138,30	139,92

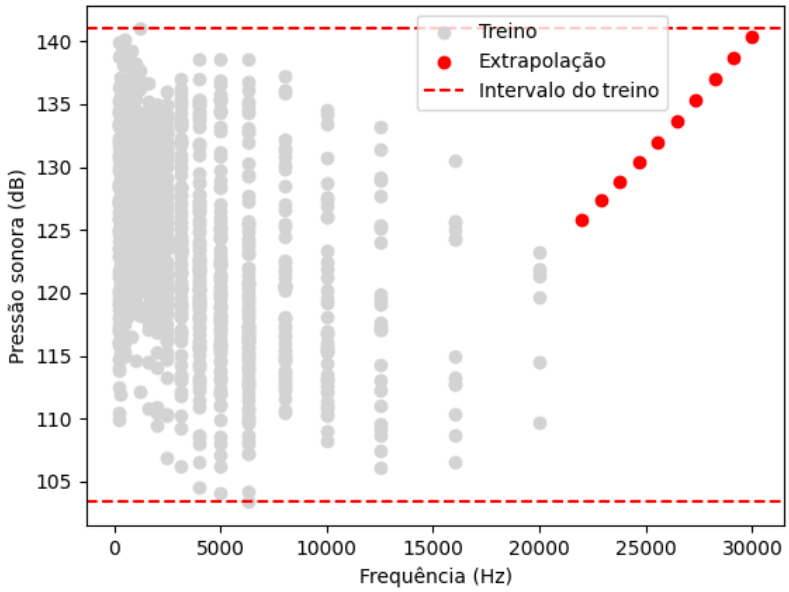


Figura 3.10: Treino e validação.

Na Figura 3.10 foi comparado o intervalo de saída dos dados de treino, entre 103,38 dB e 140,99 dB. Observamos que todas as predições extrapoladas permanecem dentro dos limites observados. Além disso, os valores aumentam de forma contínua e suave, sugerindo que a rede neural captou uma tendência coerente mesmo fora do intervalo de treino.

Essa estabilidade na extrapolação indica que o modelo, apesar de treinado em um domínio limitado, é capaz de manter consistência nas predições para entradas razoavelmente extrapoladas. No entanto, é importante destacar que, quanto mais distante dos dados de treino forem os novos exemplos, maior será a incerteza das predições, especialmente em modelos com funções de ativação como a sigmoide, que podem saturar nas extremidades.

Capítulo 4

Conclusão

Este trabalho investigou a aplicação de redes neurais artificiais, em especial o modelo Multi-Layer Perceptron (MLP), para previsão da pressão sonora em aerofólios. O estudo envolveu desde a análise exploratória dos dados, até a implementação e avaliação de um modelo treinado com TensorFlow.

A análise estatística e gráfica revelou a importância da frequência e da espessura de sucção como variáveis de maior influência sobre a pressão sonora. O processo de normalização com Z-score e a remoção de outliers foram essenciais para melhorar a estabilidade e a capacidade de generalização do modelo.

A implementação do MLP com uma camada oculta e função sigmoide foi eficaz, atingindo bom desempenho em métricas como $R^2 = 0,8485$ e MSE baixo. A comparação com regressão linear confirmou a superioridade da rede neural para modelar padrões não lineares presentes nos dados.

Três experimentos complementares foram realizados: o primeiro demonstrou que há um ponto ótimo no número de neurônios, além do qual o desempenho se estabiliza ou piora; o segundo confirmou que a MLP supera modelos lineares em desempenho preditivo; e o terceiro evidenciou que, apesar das limitações naturais de extrapolação, a rede neural produziu predições coerentes e dentro da faixa física plausível para dados fora do domínio de treino. As predições aumentaram suavemente com as entradas e mantiveram-se dentro do intervalo real observado no conjunto de treino, reforçando a estabilidade do modelo.

Concluímos que a MLP é adequada para tarefas de regressão dentro de domínios bem representados no conjunto de treino. No entanto, sua capacidade de extrapolação é limitada, sendo essencial garantir a boa cobertura do espaço de entrada. A escolha cuidadosa da arquitetura, regularização e pré-processamento foram determinantes para o sucesso da modelagem.

Referências Bibliográficas

1 TRACKER. *O que é : Multi-Layer Perceptron*. 2024. Acesso em: 31 de mai. 2025. Disponível em: <https://iatracker.com.br/glossario/o-que-e-multi-layer-perceptron/>.

2 MIX, C. *Perceptron Multicamadas*. 2009. Acesso em: 31 de mai. 2025. Disponível em: <https://www.culturamix.com/tecnologia/inteligencia-artificial/perceptron-multicamadas/>.

3 MOREIRA, S. Rede neural perceptron multicamadas. *Ensina.Ai*, 2018. Disponível em: <https://medium.com/ensina-ai/rede-neural-perceptron-multicamadas-f9de8471f1a9>.

4 RAY, D.; PINTI, O.; OBERAI, A. A. *Deep Learning and Computational Physics (Lecture Notes)*. 2023. Disponível em: <https://arxiv.org/abs/2301.00942>.

5 PH.D., E. K. J. M. *O que é regularização?* Databricks, 2023. Acesso em: 06 de jun. 2025. Disponível em: <https://www.ibm.com/br-pt/think/topics/regularization>.

6 LEITE, T. M. *Redes Neurais, Perceptron Multicamadas e o Algoritmo Backpropagation*. 2018. Acesso em: 06 de jun. 2025. Disponível em: <https://medium.com/ensina-ai/redes-neurais-perceptron-multicamadas-e-o-algoritmo-backpropagation-eaf89778f5b8>.

7 WIKIPEDIA. *TensorFlow – Wikipédia, a enciclopédia livre*. 2022. Acesso em: 06 de jul. 2025. Disponível em: <https://pt.wikipedia.org/wiki/TensorFlow>.

8 TECH, D. *O que é TensorFlow? Para que serve na prática?* Didática Tech, 2024. Acesso em: 31 de mai. 2025. Disponível em: <https://didatica.tech/o-que-e-tensorflow-para-que-serve>.

9 GEEKS, G. for. *Multi-Layer Perceptron Learning in TensorFlow*. Geeks for Geeks, 2025. Acesso em: 31 de mai. 2025. Disponível em: <https://www.geeksforgeeks.org/multi-layer-perceptron-learning-in-tensorflow/>.

10 LEITE, T. M. *Redes Neurais com TensorFlow: primeiros passos*. Medium, 2018. Acesso em: 31 de mai. 2025. Disponível em: <https://medium.com/ensina-ai/redes-neurais-com-tensorflow-primeiros-passos-20847dd5d27f>.

11 DATABRICKS. *Tudo o que você queria saber sobre o TensorFlow*. Databricks, 2024. Acesso em: 31 de mai. 2025. Disponível em: <https://www.databricks.com/br/glossary/tensorflow-guide>.

12 BROOKS, T.; POPE, D.; MARCOLINI, M. *Airfoil Self-Noise*. 1989. UCI Machine Learning Repository. doi: https://doi.org/10.24432/C5VW2C.

13 SILVA, F. da. *Análise de Correlação em Estatística*. Análise Macro, 2023. Acesso em: 07 de jun. 2025. Disponível em: <https://analisemacro.com.br/econometria-e-machine-learning/analise-de-correlacao-em-estatistica/>.

14 LIVRE, W. a enciclopédia. *Chebyshev's inequality*. Wikipédia a enciclopédia livre, 2025. Acesso em: 07 de jun. 2025. Disponível em: <https://en.wikipedia.org/wiki/Chebyshev%27s_inequality#>.

15 GEEKS, G. for. *R-squared in Regression Analysis in Machine Learning*. Geeks for Geeks, 2024. Acesso em: 08 de jun. 2025. Disponível em: <https://www.geeksforgeeks.org/ml-r-squared-in-regression-analysis/>.

Apêndice A

Códigos Python para Regressão Logística

Código A.1: Analise dos dados

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from ucimlrepo import fetch_ucirepo
import seaborn as sns
from scipy import stats

airfoil_self_noise = pd.read_table('./airfoil+self+noise/airfoil_self_noise.dat',
    names=["frequência","angulo_ataque","comprimento_corda","velocidade_fluxo",
    "espessura_sucção","pressão_sonora"])
X = airfoil_self_noise.iloc[:,0:5]
y = airfoil_self_noise.iloc[:,5:6]

# Gerar scatter plot com curva LOWESS para cada entrada
for coluna in airfoil_self_noise.columns[:-1]:  # Exclui a variável alvo
g=sns.lmplot(x=coluna,y='pressão_sonora',
data=airfoil_self_noise,lowess=True,line_kws={'color': 'red'})

g.ax.grid(True, axis='both')
sns.despine(fig=None, ax=None, top=False, right=False, left=False, bottom=False,
    offset=None, trim=False)
plt.title(f'{coluna} vs Pressão Sonora (com LOWESS)')
plt.xlabel(coluna)
plt.ylabel('pressão_sonora')
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.savefig(f'img/t1_disperssao_lowess_{coluna}.png',format='png')
plt.show()

# Calcular matriz de correlação
corr = airfoil_self_noise.corr()

# Visualizar com heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Matriz de Correlação de Pearson')
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.savefig('img/t1_matriz_correlacao.png',format='png')
```

```
plt.show()

# 1. Detectar Outliers usando Z-score
z_scores = np.abs(stats.zscore(airfoil_self_noise))           # calcula z-score absoluto
outliers = z_scores > 3                          # define outliers como z > 3
outliers_por_variavel = pd.Series(np.sum(outliers, axis=0), index=airfoil_self_noise.columns)

#print("Quantidade de outliers por variável:")
outliers_por_variavel

# Configurar estilo dos gráficos
sns.set(style="whitegrid", font_scale=1.5)

# Criar boxplots para cada variável
fig, axes = plt.subplots(2, 3, figsize=(18, 10))  # cria um grid 2x3
axes = axes.flatten()  # achata para iterar facilmente

for idx, coluna in enumerate(airfoil_self_noise.columns):
sns.boxplot(data=airfoil_self_noise, x=coluna, ax=axes[idx], color='skyblue')
axes[idx].set_title(f'Boxplot de {coluna}')
axes[idx].set_xlabel("")  # remove o nome do eixo X para estética

plt.tight_layout()
plt.savefig('img/t1_boxplot.png',format='png')
plt.show()
```

Código A.2: Multi-Layer Perceptron

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers.schedules import ExponentialDecay
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
import pandas as pd
from sklearn.metrics import r2_score

# 1. Carregar os dados normalizados
airfoil_self_noise = pd.read_table('./airfoil+self+noise/airfoil_self_noise.dat',
names=["frequência","angulo_ataque","comprimento_corda","velocidade_fluxo",
"espessura_sucção","pressão_sonora"])
df_normalizado = (airfoil_self_noise - airfoil_self_noise.mean()) / airfoil_self_noise.std()

X = df_normalizado.drop(columns=['pressão_sonora']).values
y = df_normalizado['pressão_sonora'].values

# 2. Dividir em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 3. Definir taxa de aprendizado com decaimento exponencial
initial_lr = 0.01
lr_schedule = ExponentialDecay(initial_learning_rate=initial_lr,
decay_steps=100,decay_rate=0.96,staircase=True
)
```

```
27
28 # 4. Construir o modelo
29 model = Sequential([
30 Dense(100, activation='sigmoid', kernel_regularizer=regularizers.l2(0.001)
31  input_shape=(X.shape[1],)), Dense(1, activation='linear')  # saída contínua
32 ])
33
34 # 5. Compilar o modelo
35 model.compile(optimizer=Adam(learning_rate=lr_schedule),
36 loss='mse',metrics=['mae'])
37
38 # 6. Treinar o modelo
39 history = model.fit(X_train, y_train,epochs=2000,
40 batch_size=32,validation_split=0.2,verbose=1
41 )
42
43 # 7. Avaliar no conjunto de teste
44 test_loss, test_mae = model.evaluate(X_test, y_test)
45 print(f"Erro quadrático médio no teste: {test_loss:.4f}")
46 print(f"Erro absoluto médio no teste: {test_mae:.4f}")
47
48 #Vamos ver como foi o treino?
49 fig, ax = plt.subplots(1,2, figsize=(10,8))
50 ax[0].semilogx(history.history['loss'], color='b', label="Treinamento loss")
51 ax[0].semilogx(history.history['val_loss'], color='r', label="Validação loss")
52 legend = ax[0].legend(loc='best', shadow=True)
53
54 ax[1].semilogx(history.history['mae'], color='b', label="Treinamento mae")
55 ax[1].semilogx(history.history['val_mae'], color='r',label="Validação mae")
56 legend = ax[1].legend(loc='best', shadow=True)
57 plt.grid(True)
58 plt.savefig('img/t1_training_validation.png',format='png')
59
60 # 1. Predições do modelo
61 y_pred = model.predict(X_test).flatten()
62
63 # 2. Coeficiente de determinação (R^2)
64 r2 = r2_score(y_test, y_pred)
65 print(f"Coeficiente de determinação R^2: {r2:.4f}")
66
67 # 3. Erro Absoluto Médio Relativo (RAE)
68 rae = np.sum(np.abs(y_test - y_pred)) / np.sum(np.abs(y_test - np.mean(y_test)))
69 print(f"Erro Absoluto Médio Relativo (RAE): {rae:.4f}")
70
71 # 4. Análise gráfica de resíduos
72 residuos = y_test - y_pred
73
74 plt.figure(figsize=(10, 5))
75 plt.scatter(y_pred, residuos, alpha=0.5, color='blue')
76 plt.axhline(0, color='red', linestyle='--')
77 plt.xlabel('Valores Preditos')
78 plt.ylabel('Resíduos (y_real - y_pred)')
79 plt.title('Análise de Resíduos')
80 plt.grid(True)
81 plt.savefig('img/t1_residuos.png',format='png')
82 plt.show()
83
84 y_pred = model.predict(X_test).flatten()
```

```
r2 = r2_score(y_test, y_pred)

# Gráfico: valores reais vs preditos
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.6, color='green', edgecolors='k')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', label='
    Ideal (y = x)')

# Adiciona R^2 no gráfico
plt.text(x=min(y_test), y=max(y_pred), s=f"$R^2$ = {r2:.4f}", fontsize=12, color='
    black', bbox=dict(facecolor='white', alpha=0.7))

plt.xlabel('Valores Reais')
plt.ylabel('Valores Preditos')
plt.title('Valores Reais vs Preditos')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig('img/t1_valores_reais_preditos_leg.png',format='png')
plt.show()

# Experimentos

# Lista para armazenar resultados
resultados = []
neuronios_testados = []

# Loop para testar diferentes quantidades de neurÃ´nios
for neuronios in range(10, 10000, 100):
    # Aprendizado com decaimento exponencial
    lr_schedule = ExponentialDecay(initial_learning_rate=0.01,
    decay_steps=100,decay_rate=0.96,staircase=True
    )

    # Modelo MLP
    model = Sequential([
    Dense(neuronios, activation='sigmoid', input_shape=(X.shape[1],)),
    Dense(1, activation='linear')
    ])
    model.compile(optimizer=Adam(learning_rate=lr_schedule), loss='mse')
    model.fit(X_train, y_train, epochs=200, batch_size=32, verbose=0)

    # Avaliação
    y_pred = model.predict(X_test).flatten()
    r2 = r2_score(y_test, y_pred)
    resultados.append(r2)
    neuronios_testados.append(neuronios)

# Plotar os resultados
plt.figure(figsize=(10, 6))
plt.semilogx(neuronios_testados, resultados, marker='o', linestyle='--')
plt.xlabel("Número de NeurÃ´nios na Camada Oculta")
plt.ylabel(f"$R^2$ no Teste")
plt.title("Impacto da Quantidade de NeurÃ´nios na Performance")
plt.grid(True)
plt.savefig('img/t1_impacto_neuronio_performace.png',format='png')
plt.show()
```