ANTÔNIO LUIZ MAIA NETO
PAULO FONSECA DALCIN

MODELAGEM DE PNEUS AUTOMOTIVOS: UMA ABORDAGEM EMPREGANDO-SE REDES NEURAIS ARTIFICIAIS

LAVRAS – MG
2022

ANTÔNIO LUIZ MAIA NETO
PAULO FONSECA DALCIN

MODELAGEM DE PNEUS AUTOMOTIVOS: UMA ABORDAGEM EMPREGANDO-SE REDES NEURAIS ARTIFICIAIS

Artigo Científico apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Engenharia Mecânica, para a obtenção do título de Bacharel.

Prof. Dr. Fábio Lúcio Santos
Orientador

LAVRAS – MG
2022

RESUMO

Para o estudo da dinâmica veicular, um aspecto determinante é a geração das forças advindas do pneu. Fatores como ângulo de deriva, carga vertical e câmber são variáveis intrinsecamente ligadas à geração de força lateral do pneu, além de outros aspectos dimensionais dos pneus que também contribuem com isso . Neste contexto, o uso de modelos de redes neurais artificiais (RNA) torna-se uma técnica viável para predizer a geração de forças laterais para valores desconhecidos das variáveis mencionadas. O objetivo deste trabalho foi desenvolver uma modelagem de pneu que afere a geração de forças laterais através do desenvolvimento de um algoritmo que utiliza-se de RNA. A modelagem desenvolvida apresentou um coeficiente de determinação de 99,15%, uma raiz quadrada do erro-médio de 0,0741 e uma raiz quadrada do erro médio normalizada de 1,61%.

Palavras-chave: Redes Neurais, Dinâmica Veicular, Força Lateral, Pneu.

ABSTRACT

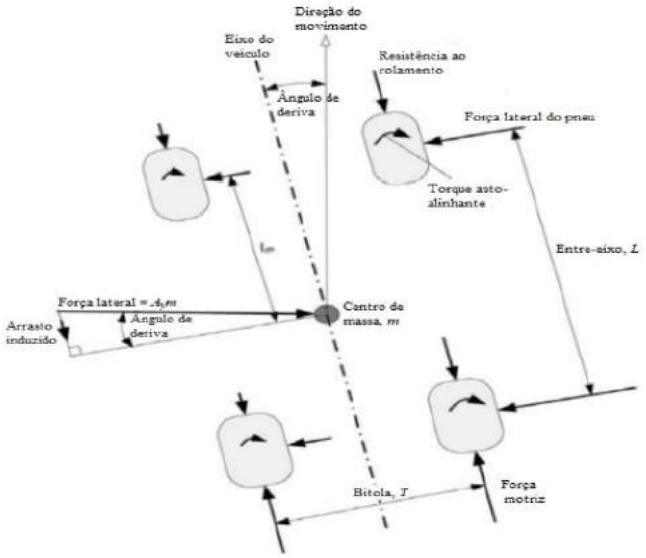
For the study of vehicle dynamics, a determining aspect is the generation of forces from the tire. Factors such as slip angle, vertical load and camber are variables intrinsically linked to the generation of lateral force of the tire, in addition to other dimensional aspects of the tires that also contribute to this. In this context, the use of artificial neural network (ANN) models becomes a viable technique to predict the generation of lateral forces for unknown values of the mentioned variables.The objective of this work was to develop a tire modeling that measures the generation of lateral forces through the development of an algorithm that uses ANN. The developed modeling presented a coefficient of determination of 99.15%, a root mean squared error of 0.0741 and a normalized root mean squared error of 1.61%.

Key-Words: Neural Networks, Vehicle Dynamics, Lateral Force, Tire.

1. INTRODUÇÃO

Tratando-se da dinâmica veicular de um automóvel, uma das principais vertentes é a dinâmica lateral. Quando um veículo é submetido a uma curva, há diversas forças que atuam diretamente em seu comportamento (SEWARD, 2014). A Figura 1 exemplifica as forças que atuam no automóvel.

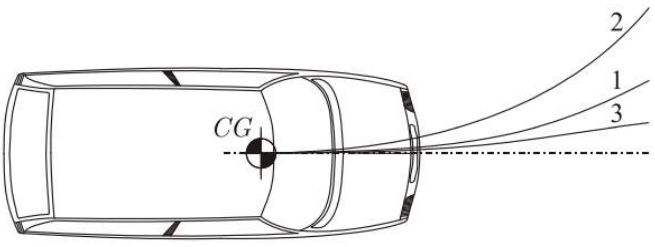
Figura 1 - Forças que afetam a dinâmica lateral de um automóvel



Fonte: Adaptado (SEWARD, 2014)

Dependendo das magnitudes das forças atuantes no pneu em uma situação de curva, sendo a força lateral a mais influente delas, o veículo pode comportar-se de maneira subesterçante (Perda de aderência no eixo dianteiro do veículo), sobreesterçante (Perda de aderência no eixo traseiro do veículo) ou neutra (NICOLAZZI, 2005), como demonstrado na Figura 2.

Figura 2 - Trajetórias reais de curva de um veículo neutro (1), subesterçante (3) e sobresterçante (2)



Fonte: (NICOLAZZI, 2005)

Na dinâmica de um veículo, busca-se atingir o comportamento previamente estabelecido no projeto. A dinâmica lateral é fundamental quando se trata de aspectos como a estabilidade, o desempenho e a confiabilidade de um veículo quando submetido a uma curva. Ao se projetar o comportamento do veículo, é necessário considerar as diversas variáveis que nele interferem, como a transferência lateral de carga e os pneus, esse primeiro, por si só, já está correlacionado com a geometria e a rigidez da suspensão (MILLIKEN, 1995).

Na modelagem de pneus, busca-se mensurar as forças advindas do pneu, como: Força lateral e força longitudinal. A força lateral, é dependente de diversas variáveis como: ângulo de deriva, carga vertical, pressão, câmber e largura do pneu, além de características físicas do próprio pneu e a superfície ao qual o pneu atua (BLUNDELL, 2014).

Uma dificuldade referente a modelagem de pneus, é encontrar um modelo analítico que possa predizer a geração de força lateral do componente para diversos tipos e modelos. Dessa forma, a utilização de modelos baseados em aprendizado de máquina é uma alternativa relevante para a modelagem de pneus.

Dentro dos modelos de aprendizado de máquina, as redes neurais artificiais (RNA's) são um dos algoritmos utilizados na resolução de diferentes problemas. No setor automotivo e aeroespacial encontram-se aplicações de redes neurais artificiais para auxiliar no mapeamento de processos que envolvem estimativas de variáveis de controle e parâmetros de projetos. (SILVA, 2010).

Por isso, a utilização de redes neurais artificiais torna-se interessante para construção de modelos de predição através de variáveis desconhecidas. Isso acontece devido à capacidade das RNA's de conseguir estimar as forças laterais de um padrão específico de pneu apenas baseando-se nos dados experimentais obtidos (OLAZAGOITIA, 2020).

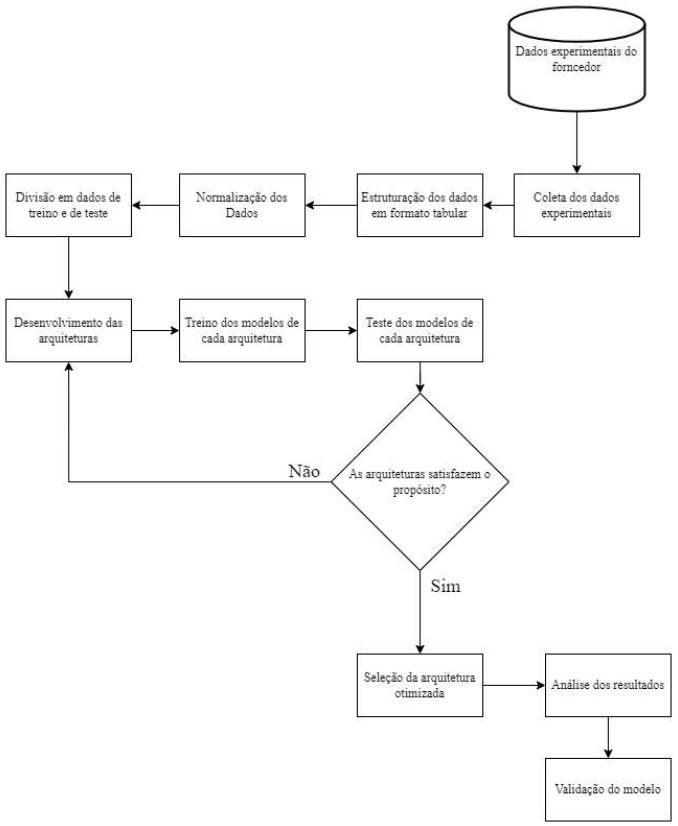
Este trabalho teve como finalidade, o desenvolvimento de um modelo de pneu através de RNA's, utilizando-se como insumo, dados experimentais fornecidos pela empresa *Avon Tyre* e obtidos por procedimentos experimentais. A partir de revisões bibliográficas, buscou-se averiguar a acuracidade do modelo e inferir sua capacidade de predição das forças laterais geradas pelo pneu.

2. MATERIAIS E MÉTODOS

2.1 Metodologia

A metodologia da modelagem é constituída de etapas que passam desde a coleta de dados fornecidos pela empresa, o desenvolvimento do modelo do algoritmo de RNA até a sua validação. A Figura 3 ilustra detalhadamente as etapas do projeto.

Figura 3 – Etapas da metodologia



Fonte: (AUTOR, 2022)

2.2 Procedimento Experimental

O estudo foi conduzido através de dados obtidos de uma fabricante de pneus (*Avon Tyres*) utilizados em carros de competição da fórmula SAE. O procedimento experimental para obter-se os dados de geração de força lateral é realizado em uma bancada de teste própria para esse tipo de experimento, como ilustram as Figuras 4 e 5.

Figura 4 – Procedimento experimental para aquisição de dados

Fonte: (MTS, 2021)

Figura 5 - Procedimento experimental para aquisição de dados

Fonte: (MTS, 2021)

Esse tipo de teste, possibilita o controle das variáveis que atuam na geração de força lateral, portanto é possível avaliá-las separadamente ou sobrepostas entre elas. Na Figura 1, também pode-se observar que há um equipamento no centro da roda que é responsável por manter a pressão dentro dos pneus constante, pois, caso contrário, a variação da pressão pode comprometer na análise de cada variável separadamente (BLUNDELL, 2014).

No ensaio experimental em questão, os parâmetros que são utilizados para avaliar os diferentes resultados são: carga vertical, câmber e o ângulo de deriva, considerando a pressão constante de 21 psi para todos os casos. Há também, a avaliação de diferentes tipos de modelos de pneu, sendo que a variação dos três parâmetros citados anteriormente para cada

modelo é a mesma, o que muda é a largura do pneu. Porém, essas diferenças no formato do pneu em cada modelo, também acarreta em variações no resultado final. As especificações para cada pneu são: 14227S, 14254S e 14140S.

2.3 Base de Dados

Os dados utilizados advêm de testes experimentais realizados pela empresa *Avon* e retirados através do próprio site da empresa (AVON TYRES, 2020). Para que o modelo consiga interpretar os dados obtidos, é necessário que haja a transformação dos dados desestruturados para um formato estruturado tabular. De modo, que após a tabulação, os dados se encontrem conforme apresentados na Tabela1.

Tabela 1 - Representação da base de dados obtidas por procedimento experimental

ÂNGULO DE DERIVA (°)	CÂMBER(°)	CARGA VERTICAL (KG)	LARGURA DO PNEU (°)	FORÇA LATERAL (kN)
-9.0	0.0	75	6.2	-1.25
-8.0	0.0	75	6.2	-1.19
-7.0	0.0	75	6.2	-1.15
-6.0	0.0	75	6.2	-1.08
-5.0	0.0	75	6.2	-1.00
...	...	...	...	...
5.0	3.5	300	8.2	02.01
6.0	3.5	300	8.2	2.27
7.0	3.5	300	8.2	2.52
8.0	3.5	300	8.2	2.74
9.0	3.5	300	8.2	2.93

Fonte: Adaptado (AVON TYRES, 2020)

A base de dados utilizada possui 1824 linhas e 5 colunas, sendo que 4 destas colunas representam as variáveis que serão os dados de entrada do sistema, ângulo de deriva, câmber, carga vertical e largura do pneu, e a coluna restante o dado de saída, a força lateral resultante.

2.4 Modelagem das redes neurais artificiais

As redes neurais artificiais são redes interconectadas entre si e divididas em camadas. Em geral, são constituídas por uma camada de entrada, uma camada de saída e, entre elas, as camadas ocultas, que fazem o processamento e podem variar a quantidade (HAYKIN, 2008).

Primeiramente, o conjunto de dados utilizado foi dividido em dados de treino e dados de teste, sendo que 60% foram destinados para o treinamento do modelo, e os 40% restantes foram utilizados para a etapa de testes. Conforme Olazagoitia (2020), não há regras fixas para definir uma proporção exata para o conjunto de treino e teste, dependendo da complexidade e da não-linearidade do modelo.

Para melhorar a capacidade de generalização do modelo, todos os dados de entrada foram normalizados, de acordo com a seguinte equação:

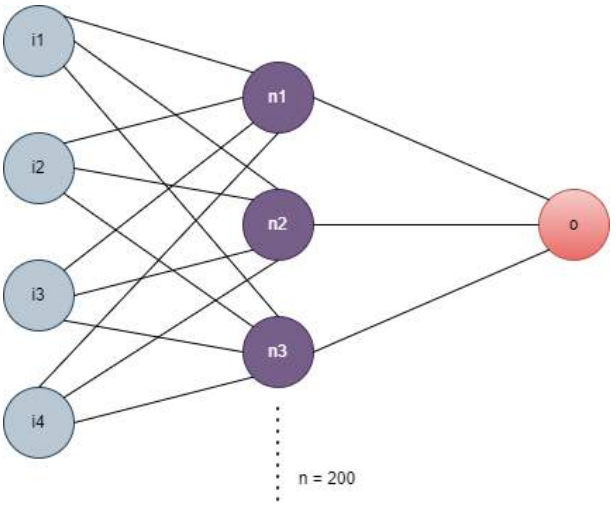
$$Z = \frac{x - u}{S} \tag{2.1}$$

Onde x são as amostras de treino, u é a média desse conjunto e S é o desvio padrão do mesmo. Após as etapas de treinamento e de testes do modelo, os dados de saída também se encontram de forma normalizada e posteriormente serão convertidos para sua forma desnormalizada.

Na arquitetura, haverá 4 dados de entrada (ângulo de deriva, câmber, carga vertical e largura do pneu), 1 de saída (força lateral) e as camadas ocultas serão escolhidas de modo que consiga encontrar a menor raiz quadrada do erro-médio com a mínima quantidade de neurônios possível, para que o modelo seja mais performático.

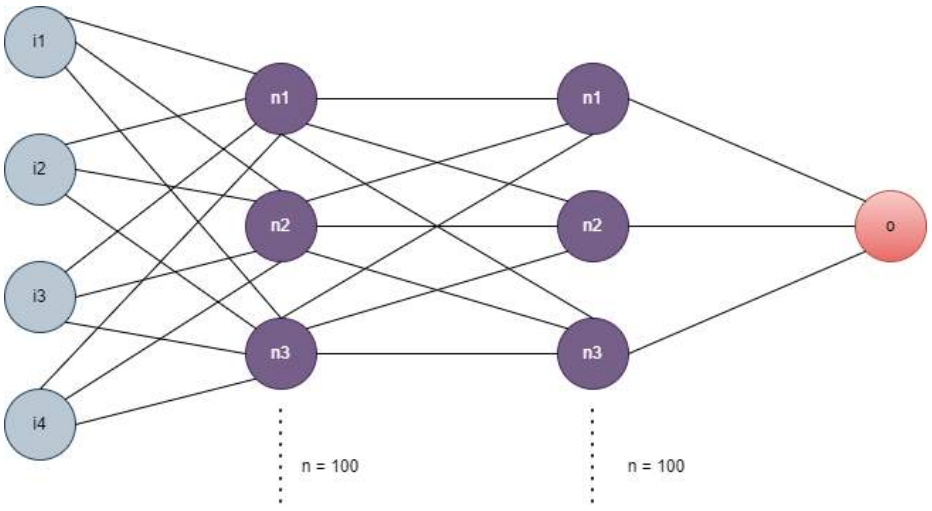
Para a definição da arquitetura a ser utilizada, será analisada uma camada oculta variando de 1 a 200 neurônios e duas camadas ocultas variando de 1 a 100 neurônios em cada uma delas. Com isso, na primeira análise serão testadas 200 arquiteturas, enquanto na segunda serão avaliadas 10.000 arquiteturas possíveis. Após o treinamento das arquiteturas, será avaliada quais delas conseguem atingir a menor raiz quadrada do erro-médio, e se mais de uma arquitetura obtiver o mesmo resultado, será utilizada a que tiver o menor número de neurônios envolvidos, para evitar um processamento desnecessário. As Figuras 6 e 7 representam as arquiteturas que serão testadas.

Figura 6 - Representação da primeira arquitetura



FONTE: (AUTOR, 2022)

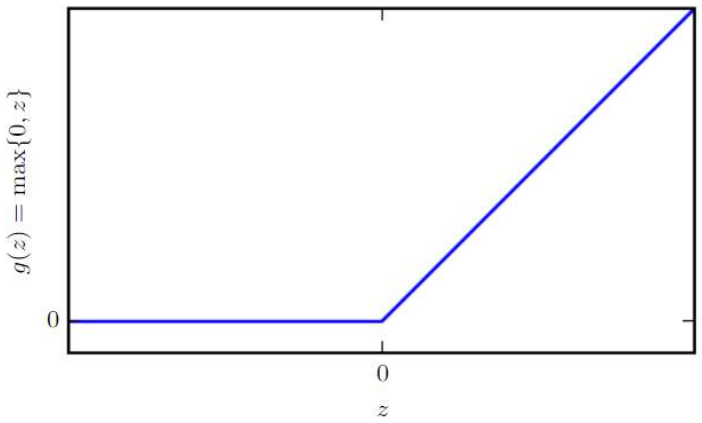
Figura 7 - Representação da segunda arquitetura



FONTE: (AUTOR, 2022)

De acordo com Goodfellow (2016), a função de ativação linear retificada (ReLU), é a função de ativação padrão recomendada para a maioria das redes neurais artificiais de retropropagação, por serem mais eficientes que as funções sigmoidais. Portanto, será a função de ativação definida para o desenvolvimento do algoritmo em todas as camadas ocultas utilizadas. A representação da função de ativação ReLU pode ser vista na Figura 8.

Figura 8 - Representação da função de ativação ReLU



Fonte: (GOODFELLOW, 2016)

No desenvolvimento das arquiteturas, os hiperparâmetros como taxa de aprendizagem, momentum e número de épocas serão utilizados os valores padrões do algoritmo de RNA, ou seja, os valores nativos caso não haja nenhuma modificação, como pode ser visto na Tabela 2.

Tabela 2 – Hiperparâmetros padrão

TAXA DE APRENDIZAGEM	MOMENTUM	NÚMERO DE ÉPOCAS	FUNÇÃO DE TRANSFERÊNCIA
0,001	0,9	200	ReLU

Fonte: (AUTOR, 2022)

Após isso, através de um algoritmo de otimização utilizando validação cruzada, esses hiperparâmetros serão redefinidos para valores otimizados que fazem com que o modelo melhore sua capacidade de generalização. O uso de validação cruzada é atraente principalmente quando temos que projetar uma grande rede neural com uma boa capacidade de generalização (HAYKIN, 2008).

Concluída essa etapa, será feito uma comparação entre os resultados obtidos pelo modelo e os resultados experimentais. Serão utilizados gráficos que demonstrem o comparativo entre todos os dados e também em cenários distintos. Posteriormente, será avaliado como o algoritmo comporta-se na predição de variáveis desconhecidas.

Por fim, após obtidos os valores da RQEM e do R^2, será avaliado a raiz quadrada do erro-médio normalizada através da equação 2.2.

$$NRQEM = \frac{RQEM}{max(F_{experimental}) - min(F_{experimental})} \quad (2.2)$$

Onde $F_{experimental}$ é a força real medida no procedimento experimental e RQEM é a raiz quadrada do erro médio já definido anteriormente. Uma vez calculada a NRQEM, será comparado o resultado com o que é encontrado na literatura, e através disso, será feito a validação da modelagem.

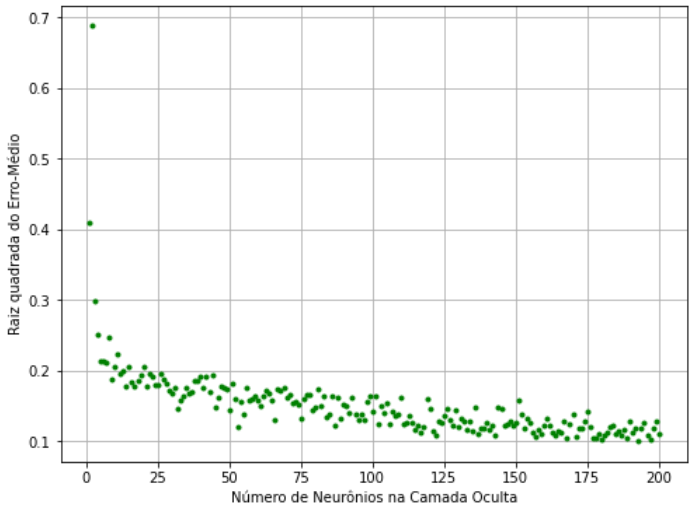
Toda a rotina de cálculos foi desenvolvida na linguagem *Python*, além de diversas bibliotecas que auxiliam no desenvolvimento da mesma.

3. RESULTADOS E DISCUSSÃO

3.1 Modelagem das redes neurais artificiais

Neste estudo, as redes neurais foram implementadas para predizer a geração de força lateral através das variáveis de entrada mencionadas anteriormente. Realizou-se duas etapas de otimização de arquitetura das redes neurais para obter a menor raiz quadrada do erro-médio com o mínimo de neurônios necessários. Para isso, foi avaliado primeiramente, o quanto uma arquitetura de somente uma camada oculta variando de 1 a 200 neurônios pode performar e, desse modo, qual seria o menor erro obtido por ela. A Figura 9 ilustra a raiz quadrada do erro-médio pelo número de neurônios em uma camada.

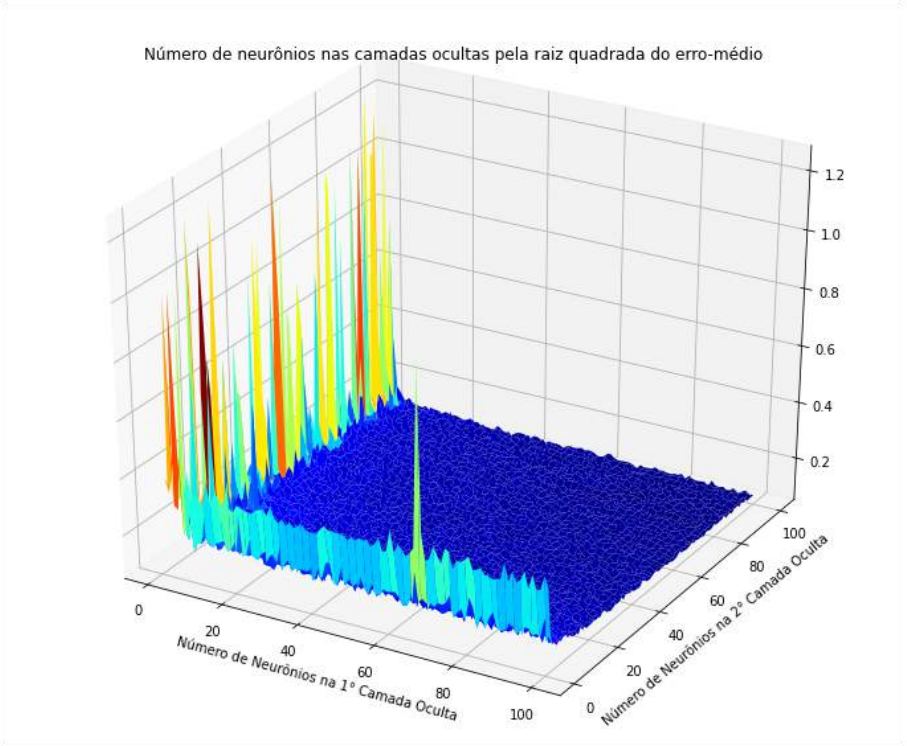
Figura 9 – Raiz quadrada do erro-médio por número de neurônios em uma camada



Fonte: (AUTOR, 2022)

Pelo gráfico, pode-se observar uma tendência de diminuição do erro de acordo com o incremento de neurônios. A menor raiz quadrada do erro-médio obtido foi de 0,1 e foram necessários 192 neurônios em uma camada para obtê-lo. Após isso, também foi avaliada uma arquitetura com duas camadas ocultas e ambas variando de 1 a 100 neurônios. A Figura 10 demonstra a variação do erro de acordo com o número da variação de neurônios em cada camada.

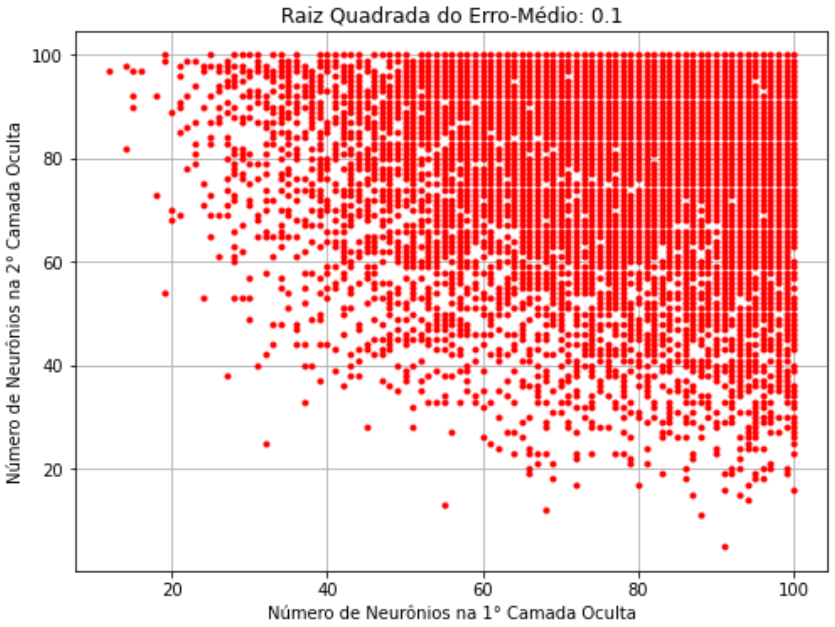
Figura 10 - Raiz quadrada do erro-médio por número de neurônios em duas camadas



Fonte: (AUTOR, 2022)

Da mesma forma, observa-se também uma tendência de diminuição dos erros com o incremento de neurônios em ambas camadas. Para efeito de comparação, ao utilizar-se de duas camadas ocultas são apresentadas diversas possibilidades de arquitetura para uma raiz quadrada do erro-médio menor do que de 0,1, que é o erro mínimo atingido ao utilizar-se somente uma camada. A Figura 11 ilustra as combinações possíveis:

Figura 11 - Número mínimo de neurônios para o erro especificado



Fonte: (AUTOR, 2022)

O mínimo de neurônios em cada camada para que seja possível atingir uma raiz quadrada do erro-médio menor que a arquitetura de uma camada, é de 32 neurônios na primeira e 25 na segunda, totalizando 57.

Já com a arquitetura de duas camadas, a menor RQEM obtida foi de 0,0741, sendo 90 e 98 neurônios na primeira e na segunda camada, respectivamente. De acordo com Olazagoitia (2020), a quantidade de camadas ocultas utilizadas pode acarretar em diferentes performances em relação a raiz quadrada do erro-médio do modelo, sendo que o incremento de mais camadas ocultas não implica, necessariamente, na diminuição da RQEM.

A Tabela 3 resume e faz uma comparação entre as duas arquiteturas.

Tabela 3 – Avaliação das Arquiteturas

	ARQUITETURA 1	ARQUITETURA 2
N° DE CAMADAS OCULTAS	1	2
N° DE NEURÔNIOS NA CAMADA OCULTA 1	192	90
N° DE NEURÔNIOS NA CAMADA OCULTA 2	-	98
TOTAL DE NEURÔNIOS	192	188
RQEM MÍNIMO	0,1	0,0741
R^2	98,6%	99,15%

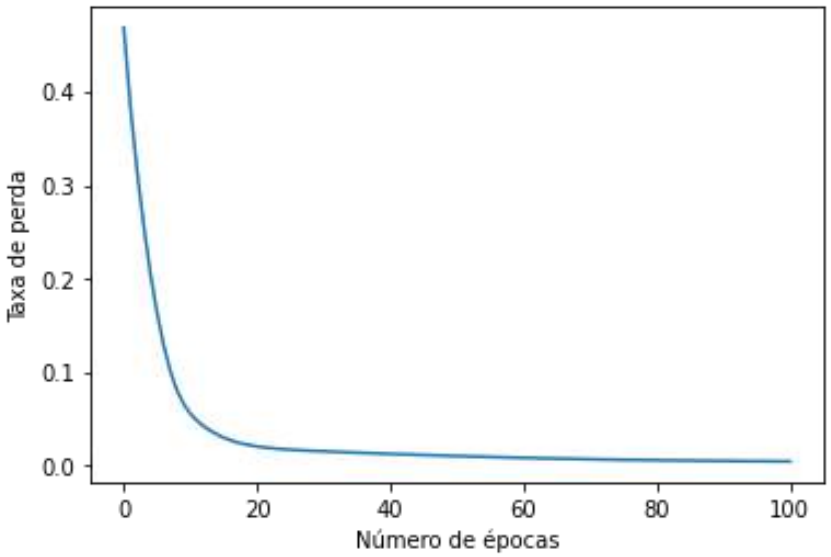
Fonte: (AUTOR, 2022)

Constata-se que a segunda arquitetura de duas camadas ocultas possui uma performance geral melhor que a primeira e, portanto, será a utilizada. Os hiperparâmetros, como a taxa de aprendizagem, momentum e função de transferência, foram definidos com os valores padrões para a definição das arquiteturas.

Após definidas as arquiteturas, foi utilizado um algoritmo de otimização por validação cruzada para encontrar os valores para os hiperparâmetros que melhor satisfazem o modelo.

O número máximo de épocas efetivo para diminuir a perda, pode ser averiguado no gráfico representado pela Figura 12 e será o mesmo valor definido no número máximo de épocas para a otimização da modelagem.

Figura 12 – Taxa de perda por número de épocas



Fonte: (AUTOR, 2022)

Os valores encontrados para a otimização da modelagem estão descritos na Tabela 4.

Tabela 4 – Hiperparâmetros otimizados

TAXA DE APRENDIZAGEM	MOMENTUM	NÚMERO DE ÉPOCAS	FUNÇÃO DE TRANSFERÊNCIA
0,0189	0.778	101	ReLU

Fonte: (AUTOR, 2022)

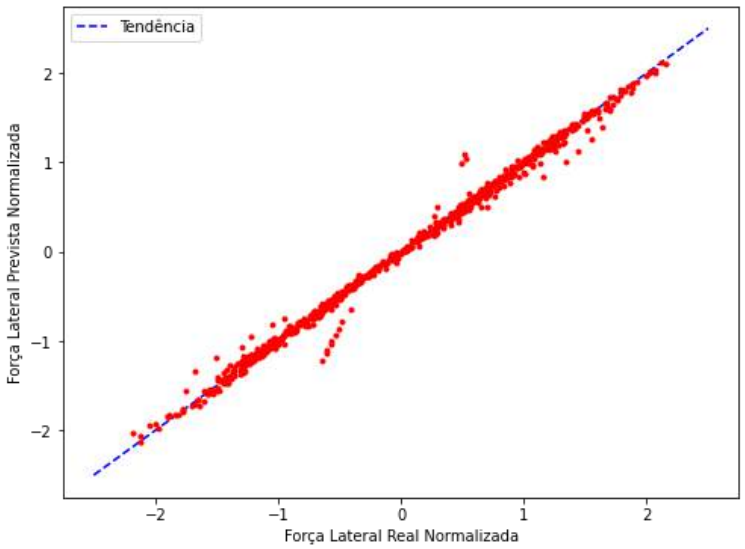
Os valores do coeficiente de determinação (R^2) e raiz quadrada do erro-médio não sofreram alterações, demonstrando que o algoritmo de otimização por validação cruzada não ocasiona em uma melhora da generalização do modelo. De acordo com Haykin (2008), o uso do método de validação cruzada torna-se interessante para modelos grandes de redes neurais. Como no modelo não há necessidade de uma modelagem muito complexa devido a baixa quantidade de variáveis de entrada e do volume de dados, a validação cruzada, de fato, não demonstra surtir melhoras de performance no modelo em questão.

3.2 Modelagem das redes neurais artificiais para a predição da geração das forças laterais do pneu

Com a arquitetura da modelagem definida, calculou-se a raiz quadrada do erro médio normalizado e obteve-se um resultado igual a 1,61%. De acordo com Nan Xu (2022) um valor de NRQEM abaixo de 5% é considerado um erro satisfatório, pois comparado a trabalhos da área que envolvem a predição das forças do pneu com foco em força lateral, o valor encontrado demonstra uma alta acurácia do modelo.

Através da modelagem das RNA's e da arquitetura otimizada, fez-se um comparativo entre os resultados obtidos pelo modelo e pelos resultados experimentais. Na Figura 13 é demonstrada a comparação realizada, sendo que quanto mais próximos os resultados forem entre si, mais próximos se encontram em uma linha de tendência, cujo coeficiente angular é igual a 45°. Ao contrário disso, quanto mais os pontos estão distantes da linha de tendência, maior é a divergência entre os resultados.

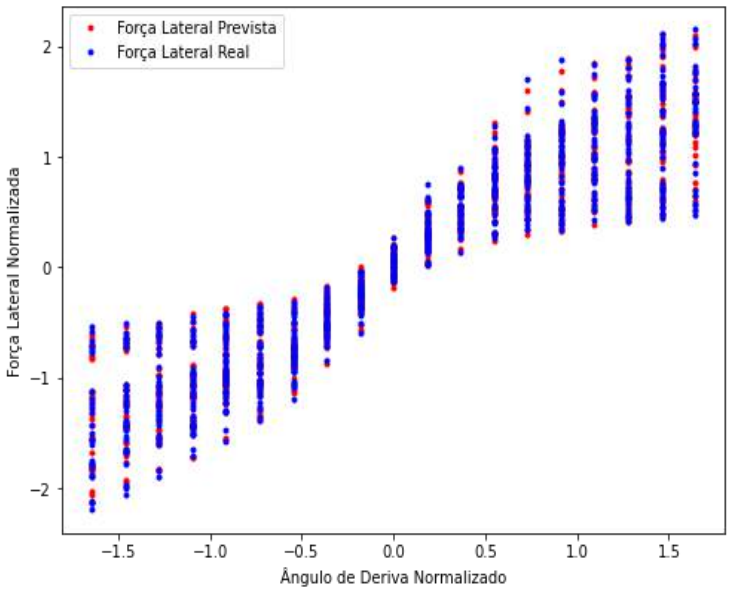
Figura 13 - Comparação entre força lateral prevista e real



Fonte: (AUTOR, 2022)

Constata-se uma convergência entre os valores do modelo e os experimentais, sendo que a maior parte encontra-se na linha de tendência ou, senão, muito próxima a ela, e a minoria dos demais pontos divergem, mas não de maneira que esteja muito distante da linha de tendência. Na Figura 14, encontra-se os mesmos resultados, porém, agora de maneira sobreposta entre eles e em função da variação do ângulo de deriva normalizado.

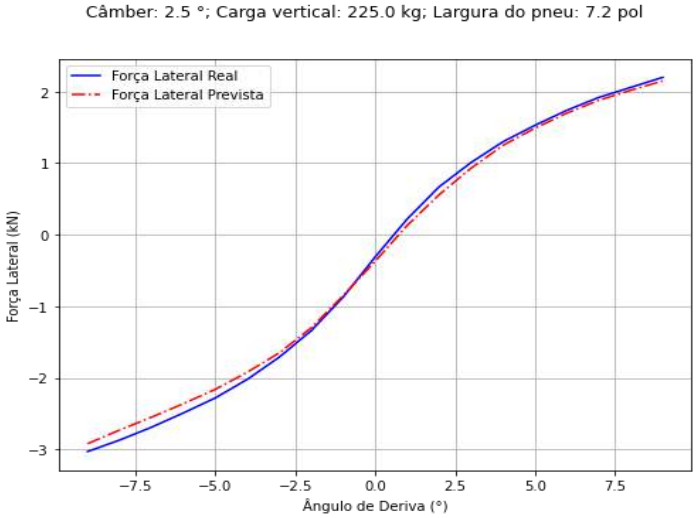
Figura 14 - Força lateral prevista e real por ângulo de deriva



Fonte: (AUTOR, 2022)

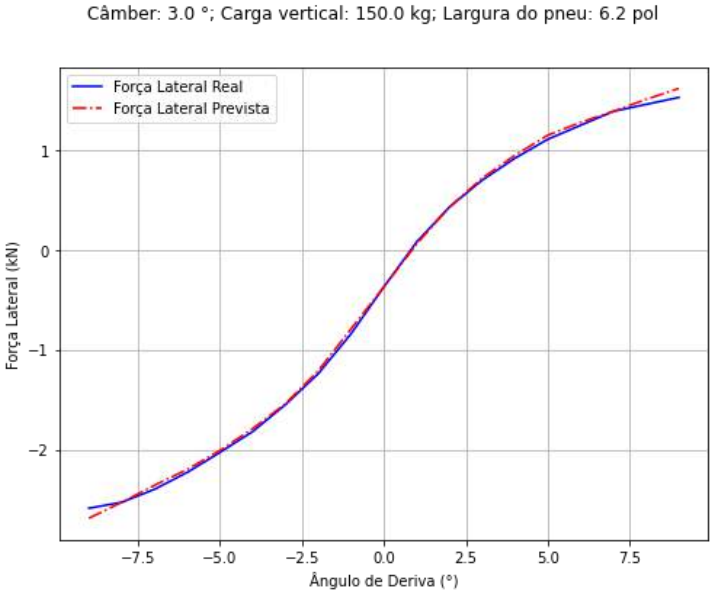
Do mesmo modo, a proximidade e a sobreposição entre os pontos com a variação do ângulo de deriva também demonstram a convergência dos valores. A seguir, estão três cenários, representados pelas Figuras 15, 16 e 17, que servirão de comparativos entre os valores simulados e experimentais com todas as variáveis de câmber, carga vertical e a largura do pneu, além da própria variação do ângulo de deriva. Vale ressaltar também que agora os valores não estão mais normalizados.

Figura 15 - Comparação entre força lateral previstas e real por ângulo de deriva (1)



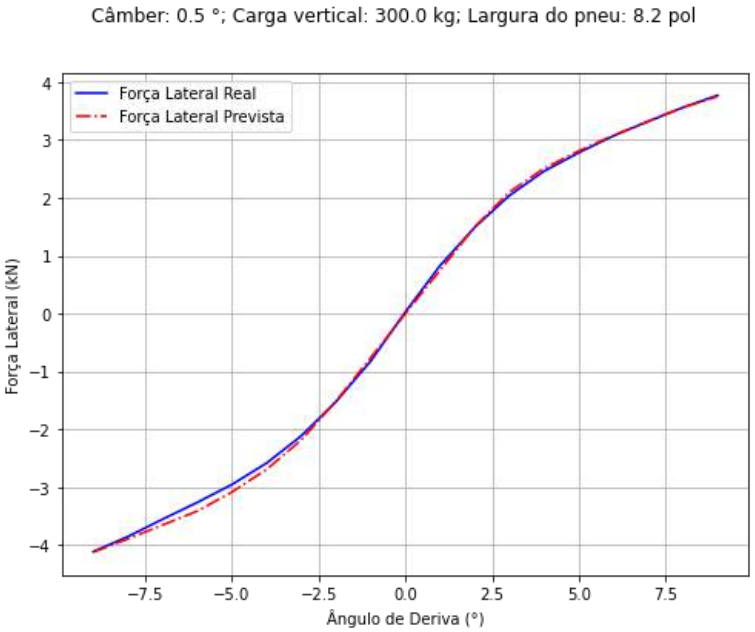
Fonte: (AUTOR, 2022)

Figura 16 - Comparação entre força lateral previstas e real por ângulo de deriva (2)



Fonte: (AUTOR, 2022)

Figura 17 - Comparação entre força lateral previstas e real por ângulo de deriva (3)



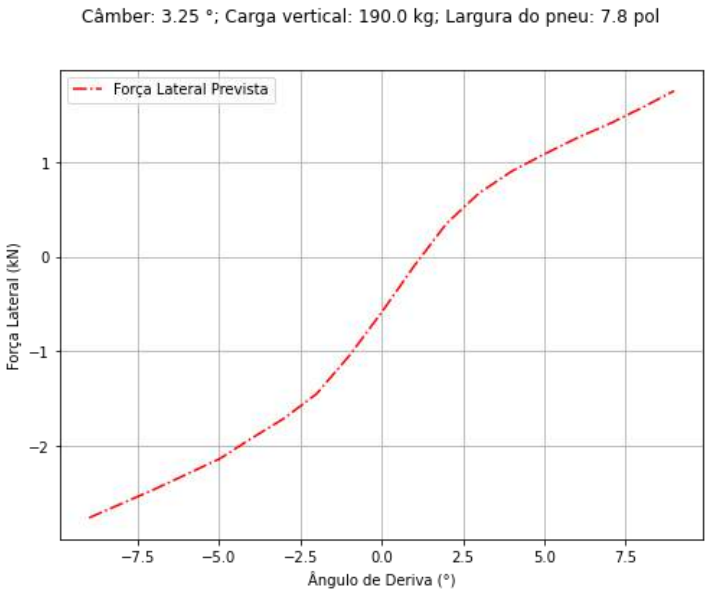
Fonte: (AUTOR, 2022)

Nas três situações, verifica-se a proximidade entre os valores da força simulada e a força real obtida experimentalmente.

3.3 Predição da geração de força lateral do pneu

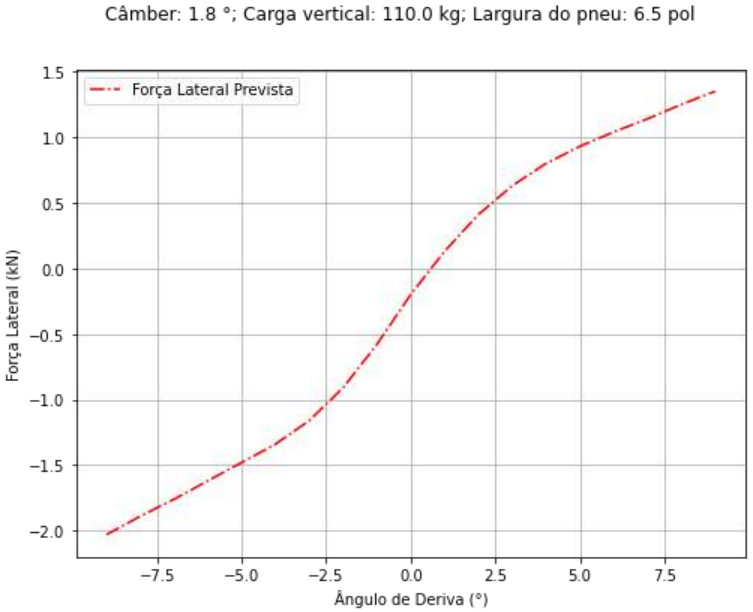
Nas Figuras 18, 19 e 20 serão apresentados os resultados da predição de geração da força lateral do pneu para variáveis desconhecidas através de gráficos.

Figura 18 - Força lateral previstas por ângulo de deriva (1)



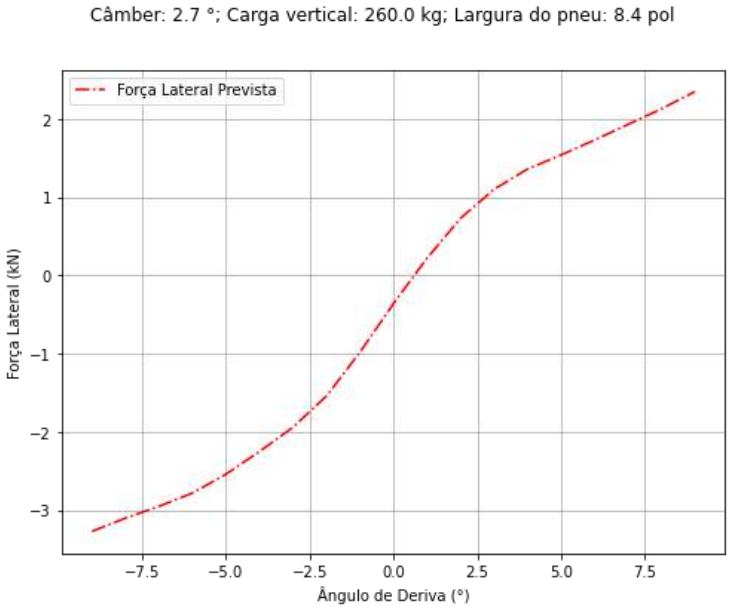
Fonte: (AUTOR, 2022)

Figura 19 - Força lateral previstas por ângulo de deriva (2)



Fonte: (AUTOR, 2022)

Figura 20 - Força lateral previstas por ângulo de deriva (3)



Fonte: (AUTOR, 2022)

Pode-se avaliar que a predição apresenta resultado congruente ao que foi visto anteriormente com os valores experimentais e simulados. Sendo que da mesma forma, os valores da predição, apresentam não-linearidade, além de estarem contidos entre a faixa de valores do conjunto de dados, o que faz sentido, já que não espera-se que esses resultados da predição excedam essa faixa de valores.

4. CONCLUSÃO

Nas condições em que o trabalho foi executado, pode-se concluir que:

- O estudo demonstra a correlação entre as variáveis do conjunto de dados fornecidos pelo fabricante com a geração de força lateral do pneu, quando alguma dessas variáveis é alterada ocasiona em diferentes resultados.

- A arquitetura de RNA com duas camadas ocultas mostrou-se mais eficiente por conseguir uma raiz quadrada do erro-médio menor e um coeficiente de determinação maior.

- Observa-se uma acurácia satisfatória do modelo de RNA, sendo ela mensurada através de um coeficiente de determinação (R^2) igual a 99,15%, e também pela raiz quadrada do erro-médio normalizado (NRQEM) que foi abaixo de 5%.

- Constata-se coerência entre os resultados simulados e observados em literatura para predição da força lateral dos pneus.

REFERÊNCIAS

BLUNDELL, Mike; HARTY, Damian. **The Multibody Systems Approach to Vehicle Dynamics.** 2. ed. [*S. l.*: *s. n.*], 2014.

LEAL, L. d. C. M.; ROSA, E. d.; NICOLAZZI, L. C. Uma introdução à modelagem quase estática de automóveis. **Publicação interna do GRANTE–Departamento de Engenharia Mecânica da UFSC, Florianópolis, Brazil,** 2012.

MILLIKEN, W. F.; MILLIKEN, D. L. et al. **Race car vehicle dynamics.** [S.l.]: Society of Automotive Engineers Warrendale, PA, 1995. v. 400

GILLESPIE, T. D. **Fundamentals of vehicle dynamics.** [S.l.], 1992.

SEWARD, Derek. **Race Car Design.** [S. l.: s. n.], 2014.

HAYKIN, Simon. **Neural Networks and Learning Machines**. 3. ed. [*S. l.*: *s. n.*], 2008.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLED, Aaron. **Deep Learning: An MIT Press Book.** [S. l.: s. n.], 2016.

OLAZAGOITIA, José Luis; PEREZ, Jesus Angel; BADEA, Francisco. **Identification of Tire Model Parameters with Artificial Neural Networks**. **MDPI**, [*s. l.*], 20 dez. 2020.

N. Xu, H. Askari, Y. Huang, J. Zhou and A. Khajepour, **"Tire Force Estimation in Intelligent Tires Using Machine Learning,"** in IEEE Transactions on Intelligent Transportation Systems, vol. 23, no. 4, pp. 3565-3574, April 2022, doi: 10.1109/TITS.2020.3038155.

SILVA, Ivan Nunes da e SPATTI, Danilo Hernane e FLAUZINO, Rogério Andrade. **Redes neurais artificiais para engenharia e ciências aplicadas.** São Paulo: Artliber Editora. Acesso em: 31 ago. 2022. , 2010

Scikit-learn: **Machine Learning in Python**, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

FLAT-TRAC® **Tire Force & Moment Measurement Systems**. [*S. l.*], 2021. Disponível em: https://www.mts.com/en/products/automotive/tire-test-systems/flat-trac-tire-system#technical. Acesso em: 22 ago. 2022.

FORMULA STUDENT - A92 STAB RIG DATA. [*S. l.*], 2020. Disponível em: https://www.avontyres.com/en-gb/tyre-care/motorsport-technical-data/technical-data-resources/formula-student/. Acesso em: 14 fev. 2022.

ANEXOS

Imports

```
import pandas as pd
import numpy as np
import joblib
import matplotlib.pyplot as plt

from pathlib import Path
from matplotlib import cm
from matplotlib.ticker import LinearLocator
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

"""DataFrame Train/Test"""

dataset = pd.read_csv("/content/drive/MyDrive/Colab Datasets/Tyre_TNN/3TyresTyreData.csv")

"""Pré-Processamento"""

dataset

scaler = StandardScaler()
dataset_normalized = scaler.fit_transform(dataset) #Normalização

x = dataset_normalized[:, 0:4]
y = dataset_normalized[:, 4]
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.6) #Divisão entre os dataframes de treino e de teste

"""## Arquitetura Otimizada

Arquitetura otimizada para uma uma camada oculta
"""

# Define o número de neurônios em cada camada, matriz que comporta o mse para cada etapa e o r2 score para esse modelo

n1h = np.arange(1, 201)

a1h = len(n1h)

mse_1h = np.zeros(a1h)
r2_1h = np.zeros(a1h)

# Gera os dados de erro para os números de neurônios em cada camada

def arquitetura_1h():

  for i in range(a1h):

      regression_1h = MLPRegressor(verbose = False, hidden_layer_sizes= (n1h[i])) #Modelo de redes meurais
      history_1h = regression_1h.fit(x_train, y_train.ravel()) #Treino do modelo com os dados de pneu
      y_pred_1h = regression_1h.predict(x_test) #resultados obtido com os dados de treino

      mse_1h[i] = mean_squared_error(y_test, y_pred_1h, squared = False).round(4)
      r2_1h[i] = 100*r2_score(y_test, y_pred_1h).round(4)
      ###
```

```
        filename = '/content/drive/MyDrive/Colab Models/models_1h_rsme/regression_model_'
+ str(i+1) + '.sav'
        joblib.dump(regression_1h, filename)

    def save_results_1h():

      results_mse_1h = pd.DataFrame(mse_1h)
      results_mse_1h.to_csv('results_mse_1h.csv')

      results_r2_1h = pd.DataFrame(r2_1h)
      results_r2_1h.to_csv('results_r2_1h.csv')

    #arquitetura_1h()

    #save_results_1h() #ativa salva o resultado das arquiteturas

    def grafico_arquitetura_1h(mse):

      plt.figure(figsize = [8,6])
      plt.plot(n1h, mse, 'g.')
      #plt.legend([ 'Number of neurons in each layer'])
        plt.xlabel('Número de Neurônios na Camada Oculta'); plt.ylabel('Raiz quadrada do
Erro-Médio')
      #plt.title('Root Mean Squared Error: '+ str(error))
      plt.grid(True)

    pd_mse_1h_results = pd.read_csv('/content/drive/MyDrive/Colab
Datasets/Tyre_TNN_RSME/results_mse_1h.csv')
    pd_mse_1h_results.drop('Unnamed: 0', axis = 1, inplace = True)

    mse_1h_results = pd_mse_1h_results.to_numpy()

    grafico_arquitetura_1h(mse_1h_results)
```

```
mse_1h_min = mse_1h_results.min()

min_neuron_1h = np.where(mse_1h_results == mse_1h_min)[0][0]

print("O menor valor de erro quadrático médio em uma camada camada oculta é de " +
str(mse_1h_min) +
      " e o menor número de neurônios para obter-se esse valor é de " + str(min_neuron_1h)
+ " neurônios.")

"""R2 Arquitetura 1"""

pd_r2_1h_results = pd.read_csv('/content/drive/MyDrive/Colab
Datasets/Tyre_TNN_RSME/results_r2_1h.csv')
pd_r2_1h_results.drop('Unnamed: 0', axis = 1, inplace = True)

r2_1h_results = pd_r2_1h_results.to_numpy()

r2_1h_max = r2_1h_results[min_neuron_1h - 1][0]

print("o R² para a primeira arquitetura é de " + str(r2_1h_max)+ " %")

"""##Arquitetura otimizada para duas camadas ocultas"""

# Define o número de neurônios em cada camada e a matriz que comporta o mse para cada
etapa

n1 = np.arange(1, 101)
n2 = np.arange(1, 101)

a = len(n1)
b = len(n2)
```

```
mse_2h = np.zeros((a, b))

# Gera os dados de erro para os números de neurônios em cada camada

def arquitetura():

  for i in range(a):

    for j in range(b):

        regression_2h = MLPRegressor(verbose = False, hidden_layer_sizes= (n1[i], n2[j])) #Modelo de redes meurais
        history_2h = regression_2h.fit(x_train, y_train.ravel()) #Treino do modelo com os dados de pneu
        y_pred_2h = regression_2h.predict(x_test) #resultados obtido com os dados de treino

        mse_2h[i, j] = mean_squared_error(y_test, y_pred_2h, squared = False).round(4)
        ####
                                        filename = '/content/drive/MyDrive/Colab Models/models_2h_rsme/regression_model_' + str(i+1) + '_' + str(j+1) + '.sav'
        joblib.dump(regression_2h, filename)

def save_results():

  results_mse_2h = pd.DataFrame(mse_2h)
  results_mse_2h.to_csv('results_mse_2h.csv')

                    #filepath = Path('/content/drive/MyDrive/Colab Models/models_2h_rsme/results_mse_2h_' + str(i) + '_' + str(j) +'.csv')
  #results_mse_2h = pd.DataFrame(mse_2h)
  #results_mse_2h.to_csv(filepath)

#arquitetura()
```

```
#save_results()

pd_mse_2h_results = pd.read_csv('/content/drive/MyDrive/Colab Datasets/Tyre_TNN_RSME/results_mse_2h.csv')
pd_mse_2h_results.drop('Unnamed: 0', axis = 1, inplace = True)

mse_2h_results = pd_mse_2h_results.to_numpy()

def grafico_arquitetura(mse):

# Make data.
  N1, N2 = np.meshgrid(n1, n2)

  fig = plt.figure(figsize = [12,8])
  ax = fig.gca(projection = '3d')
  surf = ax.plot_surface(N1, N2, mse, cmap = cm.jet, linewidth=0)

# Add a color bar which maps values to colors.
  cbar = fig.colorbar(surf, shrink=0.4, aspect=10)

  plt.title('Número de neurônios nas camadas ocultas pela raiz quadrada do erro-médio')
    plt.xlabel('Número de Neurônios na 1° Camada Oculta'); plt.ylabel('Número de Neurônios na 2° Camada Oculta')
  plt.tight_layout()
  cbar.set_label('Raiz Quadrada do Erro-Médio ')
  plt.show()

grafico_arquitetura(mse_2h_results)

mse_2h_results.min() #erro mínimo do modelo da segunda arquitetura

"""Mínimo de neurônios para encontrar o erro máximo desejado"""
```

```
    error = mse_1h_results.min()

    error_filtered = pd_mse_2h_results[pd_mse_2h_results <= error].to_numpy()

    neurons_decision = []

    for i in range (a):
      for j in range (b):

        if error_filtered[i, j] > 0:
          neurons_decision.append([i + 1, j + 1])

    neurons_decision_sum = []

    for i in range(len(neurons_decision)):
      neurons_decision_sum.append(sum(neurons_decision[i]))

    neuron_layer_1 = []
    neuron_layer_2 = []

    for i in range(len(neurons_decision)):
      neuron_layer_1.append(neurons_decision[i][0])
      neuron_layer_2.append(neurons_decision[i][1])

    plt.figure(figsize = [8,6])
    plt.plot(neuron_layer_1, neuron_layer_2, 'r.')
    #plt.legend([ 'Number of neurons in each layer'])
    plt.xlabel('Número de Neurônios na 1° Camada Oculta'); plt.ylabel('Número de Neurônios
na 2° Camada Oculta')
    plt.title('Raiz Quadrada do Erro-Médio: '+ str(error))
    plt.grid(True)
```

```
coord_num_min = neurons_decision_sum.index(min(neurons_decision_sum))

num_min_neurons = neurons_decision[coord_num_min]

num_min_neurons

print('Erro quadrático médio requerido: ' + str(error) + ', Número mínimo de neurônios na camada oculta 1: ' +
      str(num_min_neurons[0]) + ', Número mínimo de neurônios na camada oculta 2: ' + str(num_min_neurons[1]))

"""##Otimização dos Hiperparâmetros/ Validação Cruzada"""

regression = joblib.load('/content/drive/MyDrive/Colab Models/models_2h_rsme/regression_model_' + str(num_min_neurons[0]) + '_'
                         + str(num_min_neurons[1]) + '.sav')

y_pred_regression = regression.predict(x_test)

r2_2h = 100*r2_score(y_test, y_pred_regression).round(4)

print ("R2 Score da 2° Arquitetura: " + str(r2_2h) + " %")

rmse_regression = mean_squared_error(y_test, y_pred_regression, squared = False)

error_regression = (rmse_regression/(y.max() - y.min()))*100
print(error_regression)

score_train_regression = 100*regression.score(x_train, y_train).round(4) #R2 Score
print(score_train_regression)

score_test_regression = 100*regression.score(x_test, y_test).round(4) #R2 Score
print(score_test_regression)
```

```
    """Definição Hiperparâmetros"""

    hiperparameters = {'learning_rate_init' : np.linspace(0.01, 0.02, 10),
                       'momentum': np.linspace(0, 1, 10)}

    gridsearch = GridSearchCV(estimator = regression, param_grid = hiperparameters)

    def optimizer():

      gridsearch.fit(x_train, y_train)

      print(gridsearch.best_params_)
      print(gridsearch.best_score_)

    #optimizer()

    #learning_rate_optimized = gridsearch.best_params_['learning_rate_init']
    #momentum_optimized = gridsearch.best_params_['momentum']

    #print(learning_rate_optimized, momentum_optimized)

    """Função de perda - n° de épocas"""

    #plt.rcParams.update({'font.size':12})
    plt.figure(figsize = [8,6])
    pd.DataFrame(regression.loss_curve_).plot(legend = False) #Função de perda decaindo

    plt.ylabel('Taxa de perda')
    plt.xlabel('Número de épocas')

    #final_regression = regression.set_params(learning_rate_init = learning_rate_optimized,
momentum = momentum_optimized, max_iter = regression.n_iter_)
```

```
    #final_filename = '/content/drive/MyDrive/Colab Models/models_2h_rsme/final_regression_model.sav'
    #joblib.dump(final_regression, final_filename)

    final_regression = joblib.load('/content/drive/MyDrive/Colab Models/models_2h_rsme/final_regression_model.sav')
    final_regression = final_regression.set_params(max_iter = regression.n_iter_)

    y_pred = final_regression.predict(x_test)

    rmse_final_regression = mean_squared_error(y_test, y_pred, squared = False)
    print(rmse_final_regression)

    error_final_regression = (rmse_final_regression/(y.max() - y.min()))*100
    print(error_final_regression)

    score_train_final_regression = 100*final_regression.score(x_train, y_train).round(4) #R2 Score
    print(score_train_final_regression)

    score_test_final_regression = 100*final_regression.score(x_test, y_test).round(4) #R2 Score
    print(score_test_final_regression)

    regression.momentum

    final_regression.momentum

    """##Validação do Modelo

    Comparação entre todas as forças laterais reais e previstas
    """
```

```
plt.figure(figsize = [8,6])
plt.plot(x_test[:, 0], y_pred, 'r.')
plt.plot(x_test[:, 0], y_test, 'b.')
plt.legend(['Força Lateral Prevista', 'Força Lateral Real'])
plt.xlabel(' ngulo de Deriva Normalizado'); plt.ylabel('Força Lateral Normalizada')

"""Avaliação da acurácia do modelo"""

def tendencia(var):
  yt = var
  return yt

tend_x = np.linspace(-2.5,2.5, 10)
tend_y = tendencia(tend_x)

df_x_train_normalized = pd.DataFrame(x_train, columns=['slip_angle', 'camber', 'load', 'size'])
df_y_train_normalized = pd.DataFrame(y_train, columns=['lateral_force'])

df_x_test_normalized = pd.DataFrame(x_test, columns=['slip_angle', 'camber', 'load', 'size'])
df_y_test_normalized = pd.DataFrame(y_test, columns=['lateral_force'])

df_train_normalized = df_x_train_normalized.join(df_y_train_normalized)
df_test_normalized = df_x_test_normalized.join(df_y_test_normalized)

df_y_pred_normalized = pd.DataFrame(y_pred, columns=['lateral_force'])

df_pred_normalized = df_x_test_normalized.join(df_y_pred_normalized)

df_train = pd.DataFrame(scaler.inverse_transform(df_train_normalized), columns = ['slip_angle', 'camber', 'load', 'size', 'lateral_force'])
```

```
df_test = pd.DataFrame(scaler.inverse_transform(df_test_normalized), columns = ['slip_angle', 'camber', 'load', 'size', 'lateral_force'])

df_pred = pd.DataFrame(scaler.inverse_transform(df_pred_normalized), columns = ['slip_angle', 'camber', 'load', 'size', 'lateral_force'])

df_train = df_train.round(2)
df_test = df_test.round(2)
df_pred = df_pred.round(2)

"""Avaliação dos resultados obtidos para camber e carga específica"""

camber = 2.5#2.5.
load = 225#225
size = 7.2#7.2

#Filtros para encontrar os parâmetros a serem analisados
df_train_filtered = df_train[df_train['load'].isin([load])]
df_train_filtered_2 = df_train_filtered[df_train_filtered['camber'].isin([camber])]
df_train_filtered_3 = df_train_filtered_2[df_train_filtered['size'].isin([size])]

df_test_filtered = df_test[df_test['load'].isin([load])]
df_test_filtered_2 = df_test_filtered[df_test_filtered['camber'].isin([camber])]
df_test_filtered_3 = df_test_filtered_2[df_test_filtered['size'].isin([size])]

dataset_filtered = dataset[dataset['load'].isin([load])]
dataset_filtered_2 = dataset_filtered[dataset_filtered['camber'].isin([camber])]
dataset_filtered_3 = dataset_filtered_2[dataset_filtered['size'].isin([size])]

df_pred_filtered = df_pred[df_pred['load'].isin([load])]
df_pred_filtered_2 = df_pred_filtered[df_pred_filtered['camber'].isin([camber])]
df_pred_filtered_3 = df_pred_filtered_2[df_pred_filtered['size'].isin([size])]
```

```
    plt.figure(figsize = [8,6])
    plt.plot(dataset_filtered_3['slip_angle'], dataset_filtered_3['lateral_force'], 'b-')
    plt.plot(df_pred_filtered_3['slip_angle'], df_pred_filtered_3['lateral_force'], 'r.')
    plt.legend(['Força Lateral Real', 'Força Lateral Prevista'])
    plt.xlabel(' ngulo de Deriva (°)'); plt.ylabel('Força Lateral (kN)')
    plt.grid(True)
    plt.suptitle('Câmber: '+ str(camber) + ' °; Carga vertical: ' + str(load) + ' kg;' + ' Largura do pneu: ' + str(size) + ' pol')

    """Predição"""

    df_predict = pd.read_csv('/content/drive/MyDrive/Colab Datasets/Tyre_TNN/3TyresPredict.csv')

    #Camber e carga para predição
    camber_predict = camber #1.8
    load_predict = load #110
    size_predict = size #6.5

    for i in range(len(df_predict)):
      df_predict['camber'][i] = camber_predict
      df_predict['load'][i] = load_predict
      df_predict['size'][i] = size_predict

    predict_normalized = scaler.transform(df_predict) #Normalização dos dados a serem predizidos

    x_predict_normalized = predict_normalized[:, 0:4]

    results_pred = regression.predict(x_predict_normalized) #Predição dos resultados do modelo

    df_y_predict_normalized = pd.DataFrame(results_pred, columns = ['lateral_force'])
```

```
    df_x_predict_normalized = pd.DataFrame(x_predict_normalized, columns = ['slip_angle', 'camber', 'load', 'size'])

    df_predict_normalized = df_x_predict_normalized.join(df_y_predict_normalized)

    df_predict = pd.DataFrame(scaler.inverse_transform(df_predict_normalized), columns = ['slip_angle', 'camber', 'load', 'size', 'lateral_force'])
    df_predict = df_predict.round(2)

    """Resultados obtidos pela predição"""

    plt.figure(figsize = [8,6])
    plt.plot(df_predict['slip_angle'], df_predict['lateral_force'], 'r-.')
    plt.legend([ 'Força Lateral Prevista'])
    plt.xlabel(' ngulo de Deriva (°)'); plt.ylabel('Força Lateral (kN)')
    plt.grid(True)
    plt.suptitle('Câmber: '+ str(df_predict['camber'][0]) + ' °; Carga vertical: ' + str(df_predict['load'][0]) + ' kg; Largura do pneu: '+ str(df_predict['size'][0]) + ' pol')

    df_predict

    """Comparação entre força real e prevista"""

    plt.figure(figsize = [8,6])
    plt.plot(dataset_filtered_3['slip_angle'], dataset_filtered_3['lateral_force'], 'b-')
    plt.plot(df_predict['slip_angle'], df_predict['lateral_force'], 'r-.')
    plt.legend([ 'Força Lateral Real','Força Lateral Prevista'])
    plt.xlabel(' ngulo de Deriva (°)'); plt.ylabel('Força Lateral (kN)')
    plt.grid(True)
    plt.suptitle('Câmber: '+ str(df_predict['camber'][0]) + ' °; Carga vertical: ' + str(df_predict['load'][0]) + ' kg; Largura do pneu: '+ str(df_predict['size'][0]) + ' pol')
```

```
    """Comparação entre todos os dados"""

    plt.figure(figsize = [8,6])
    plt.plot(tend_y, tend_x, 'b--')
    plt.plot(y_test, y_pred, 'r.')
    plt.legend(['Tendência'])
    plt.xlabel('Força Lateral Real Normalizada'); plt.ylabel('Força Lateral Prevista Normalizada')
```