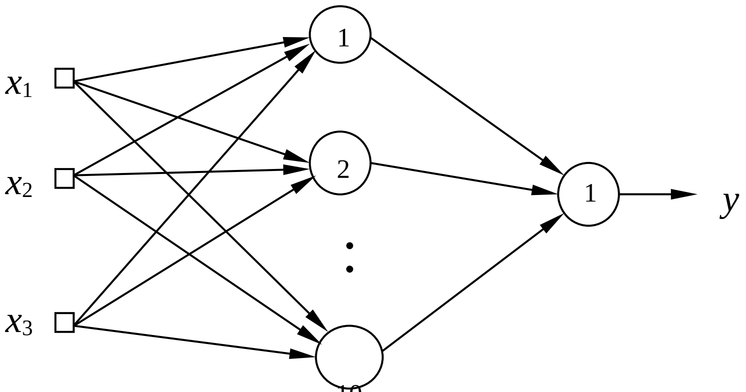
IMPLEMENTAR A SOLUÇÃO E RESPONDER AS QUESTÕES EM FORMA DE RELATÓRIO. DEVE-SE APRESENTAR O CÓDIGO EM ANEXO (FAZER NO GOOGLE COLAB:

Para a confecção de um sistema de ressonância magnética, observou-se que é de extrema importância para o bom desempenho do processador de imagens de que a variável $\{y\}$, que mede a energia absorvida do sistema, possa ser estimada a partir da medição de três outras grandezas $\{x_1 , x_2 , x_3\}$. Entretanto, em função da complexidade do sistema, sabe-se que este mapeamento é de difícil obtenção por técnicas convencionais, sendo que o modelo matemático disponível para representação do mesmo não fornece resultados satisfatórios.

Assim, a equipe de engenheiros e cientistas pretende utilizar uma rede perceptron multicamadas como um aproximador universal de funções, tendo-se como objetivo final de que, dado como entrada os valores de $\{x_1 , x_2 , x_3\}$, a mesma possa estimar (após o treinamento) o respectivo valor da variável $\{y\}$ que representa a energia absorvida. A topologia da rede perceptron constituída de duas camadas neurais está ilustrada na figura abaixo.

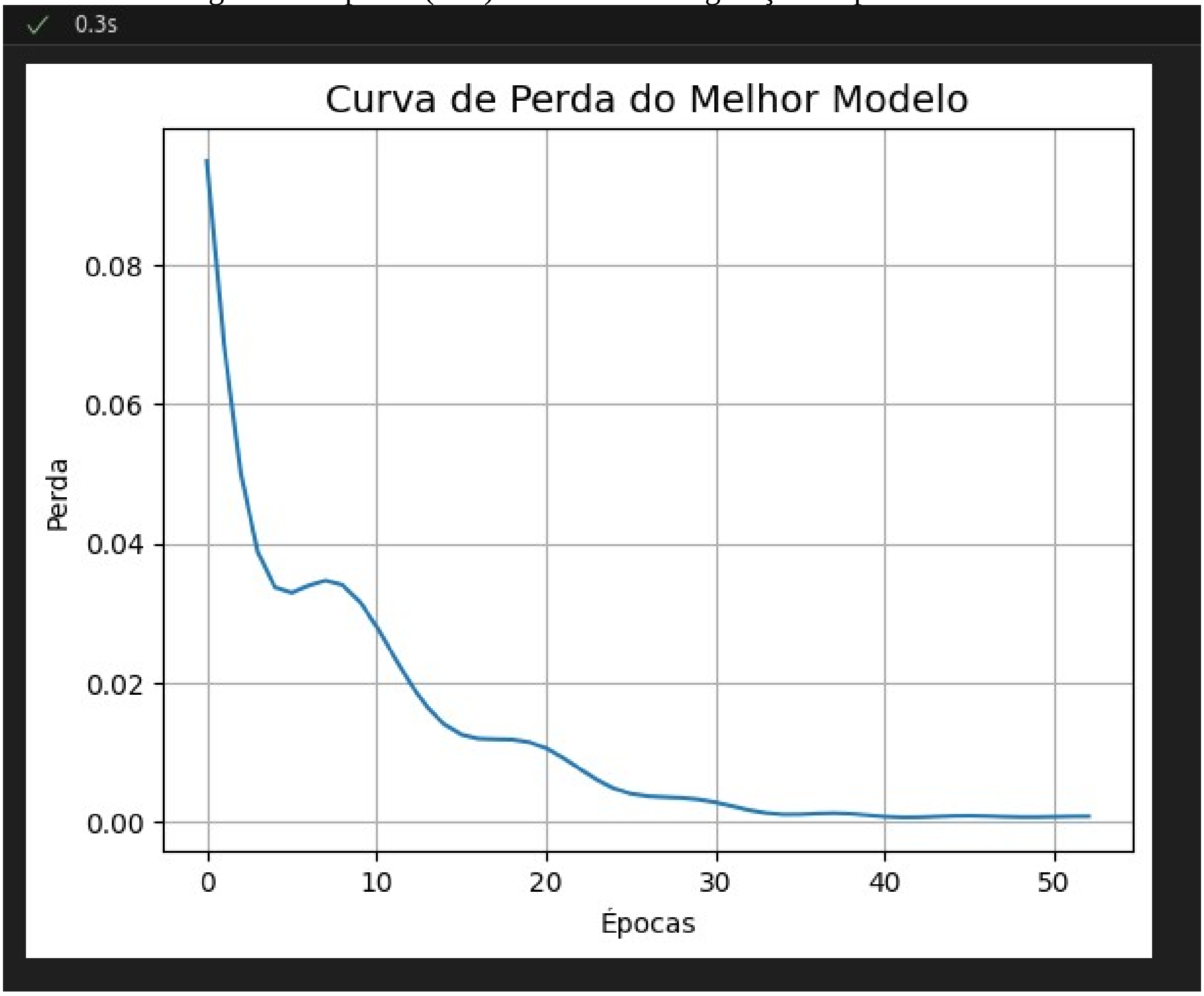


Utilizando a função MLPRegressor do Sklearn Python, treine a RNA com dados de apresentados no Anexo e disponíveis em csv no Moodle, considerando as variáveis de entrada $\{x_1 , x_2 , x_3\}$ e saída {d}. Note que o dataset já está normalizado. Realize as seguintes atividades:

1. Execute treinamentos da RNA gerando uma combinação das funções de ativação ('tanh', 'relu', 'logistic') e quantidade de neurônios na camada oculta (5, 10 e 15 neurônios). Considere uma taxa de aprendizagem de 0.01 e a quantidade máxima de épocas fixada em 1000. As demais configurações pode ser considerada o padrão. Portanto, considerando esta variação de possibilidades, você terá que efetuar 9 treinamentos para chegar na conclusão da melhor configuração proposta. Em resposta abaixo, informe qual é a melhor configuração obtida.

Treinamento	Neurônios	Ativação	Perda Final	Número de Épocas
1º (T1)	15	relu	0.0015	53
2º (T2)	10	relu	0.0015	65
3º (T3)	5	tanh	0.0017	127
4º (T4)	15	tanh	0.0031	28
5º (T5)	15	logistic	0.0168	18
6º (T6)	10	logistic	0.0218	20
7º (T7)	5	logistic	0.0251	35
8º (T8)	5	relu	0.0301	79
9º (T9)	10	tanh	0.0591	23

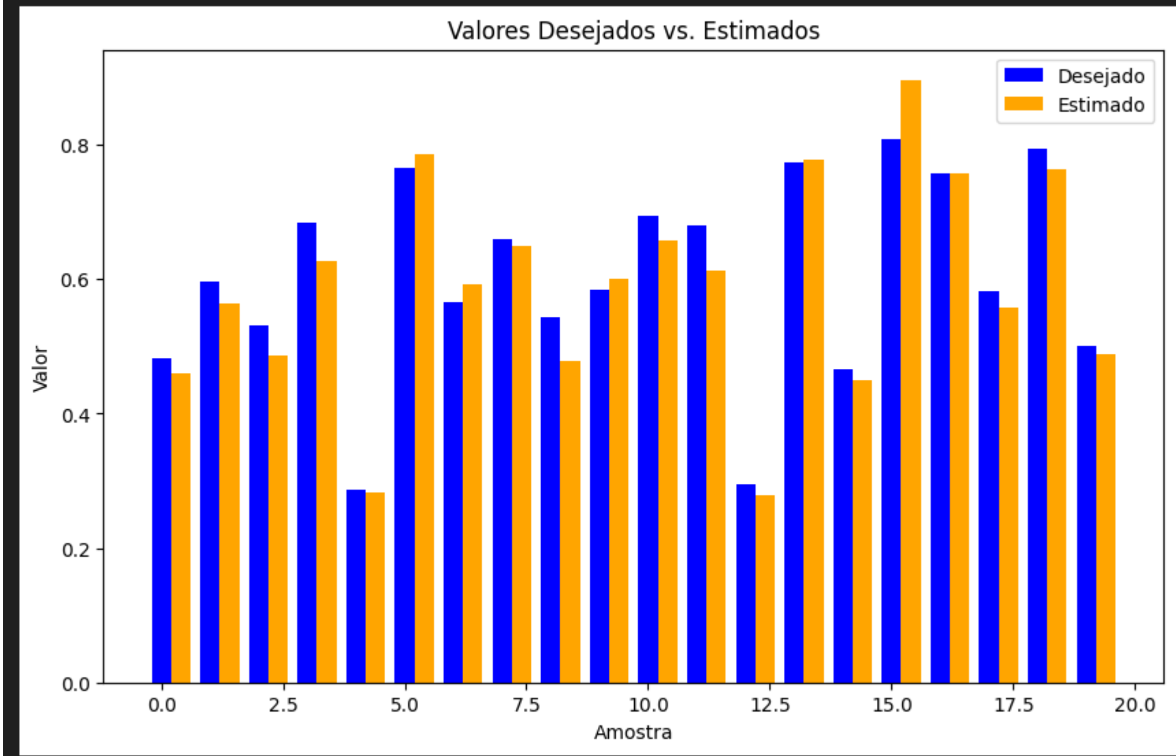
2. Mostrar o gráfico de perda (loss) da melhor configuração do passo anterior:



3. Para o melhor treinamento do Item 1, faça então a validação da rede aplicando o conjunto de teste fornecido na tabela abaixo. Apresente um gráfico de barras comparando os valores estimados e os desejados. Além disso, mostre as seguintes métricas na validação do lote de amostras: Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE) e Mean Absolute Percentage Error (MAPE).

Amostra	x_1	x_2	x_3	d
1	0.0611	0.2860	0.7464	0.4831
2	0.5102	0.7464	0.0860	0.5965
3	0.0004	0.6916	0.5006	0.5318
4	0.9430	0.4476	0.2648	0.6843
5	0.1399	0.1610	0.2477	0.2872
6	0.6423	0.3229	0.8567	0.7663
7	0.6492	0.0007	0.6422	0.5666
8	0.1818	0.5078	0.9046	0.6601
9	0.7382	0.2647	0.1916	0.5427
10	0.3879	0.1307	0.8656	0.5836
11	0.1903	0.6523	0.7820	0.6950
12	0.8401	0.4490	0.2719	0.6790
13	0.0029	0.3264	0.2476	0.2956
14	0.7088	0.9342	0.2763	0.7742
15	0.1283	0.1882	0.7253	0.4662
16	0.8882	0.3077	0.8931	0.8093
17	0.2225	0.9182	0.7820	0.7581
18	0.1957	0.8423	0.3085	0.5826
19	0.9991	0.5914	0.3933	0.7938
20	0.2299	0.1524	0.7353	0.5012

✓ 0.5s




```
Mean Absolute Error (MAE): 0.02990738521857437
Mean Squared Error (MSE): 0.001412530598836247
Root Mean Squared Error (RMSE): 0.03758364802458972
Mean Absolute Percentage Error (MAPE): 4.948898357013116
```

ANEXO – Dataset Completo para Treinamento

Amostra	x_1	x_2	x_3	d	Amostra	x_1	x_2	x_3	d	Amostra	x_1	x_2	x_3	d
1	0.8799	0.7998	0.3972	0.8399	**71**	0.3644	0.2948	0.3937	0.5240	**141**	0.2858	0.9688	0.2262	0.5988
2	0.5700	0.5111	0.2418	0.6258	**72**	0.2014	0.6326	0.9782	0.7143	**142**	0.7931	0.8993	0.9028	0.9728
3	0.6796	0.4117	0.3370	0.6622	**73**	0.4039	0.0645	0.4629	0.4547	**143**	0.7841	0.0778	0.9012	0.6832
4	0.3567	0.2967	0.6037	0.5969	**74**	0.7137	0.0670	0.2359	0.4602	**144**	0.1380	0.5881	0.2367	0.4622
5	0.3866	0.8390	0.0232	0.5316	**75**	0.4277	0.9555	0.0000	0.5477	**145**	0.6345	0.5165	0.7139	0.8191
6	0.0271	0.7788	0.7445	0.6335	**76**	0.0259	0.7634	0.2889	0.4738	**146**	0.2453	0.5888	0.1559	0.4765
7	0.8174	0.8422	0.3229	0.8068	**77**	0.1871	0.7682	0.9697	0.7397	**147**	0.1174	0.5436	0.3657	0.4953
8	0.6027	0.1468	0.3759	0.5342	**78**	0.3216	0.5420	0.0677	0.4526	**148**	0.3667	0.3228	0.6952	0.6376
9	0.1203	0.3260	0.5419	0.4768	**79**	0.2524	0.7688	0.9523	0.7711	**149**	0.9532	0.6949	0.4451	0.8426
10	0.1325	0.2082	0.4934	0.4105	**80**	0.3621	0.5295	0.2521	0.5571	**150**	0.7954	0.8346	0.0449	0.6676
11	0.6950	1.0000	0.4321	0.8404	**81**	0.2942	0.1625	0.2745	0.3759	**151**	0.1427	0.0480	0.6267	0.3780
12	0.0036	0.1940	0.3274	0.2697	**82**	0.8180	0.0023	0.1439	0.4018	**152**	0.1516	0.9824	0.0827	0.4627
13	0.2650	0.0161	0.5947	0.4125	**83**	0.8429	0.1704	0.5251	0.6563	**153**	0.4868	0.6223	0.7462	0.8116
14	0.5849	0.6019	0.4376	0.7464	**84**	0.9612	0.6898	0.6630	0.9128	**154**	0.3408	0.5115	0.0783	0.4559
15	0.0108	0.3538	0.1810	0.2800	**85**	0.1009	0.4190	0.0826	0.3055	**155**	0.8146	0.6378	0.5837	0.8628
16	0.9008	0.7264	0.9184	0.9602	**86**	0.7071	0.7704	0.8328	0.9298	**156**	0.2820	0.5409	0.7256	0.6939
17	0.0023	0.9659	0.3182	0.4986	**87**	0.3371	0.7819	0.0959	0.5377	**157**	0.5716	0.2958	0.5477	0.6619
18	0.1366	0.6357	0.6967	0.6459	**88**	0.1555	0.5599	0.9221	0.6663	**158**	0.9323	0.0229	0.4797	0.5731
19	0.8621	0.7353	0.2742	0.7718	**89**	0.7318	0.1877	0.3311	0.5689	**159**	0.2907	0.7245	0.5165	0.6911
20	0.0682	0.9624	0.4211	0.5764	**90**	0.1665	0.7449	0.0997	0.4508	**160**	0.0068	0.0545	0.0861	0.0851
21	0.6112	0.6014	0.5254	0.7868	**91**	0.8762	0.2498	0.9167	0.7829	**161**	0.2636	0.9885	0.2175	0.5847
22	0.0030	0.7585	0.8928	0.6388	**92**	0.9885	0.6229	0.2085	0.7200	**162**	0.0350	0.3653	0.7801	0.5117
23	0.7644	0.5964	0.0407	0.6055	**93**	0.0461	0.7745	0.5632	0.5949	**163**	0.9670	0.3031	0.7127	0.7836
24	0.6441	0.2097	0.5847	0.6545	**94**	0.3209	0.6229	0.5233	0.6810	**164**	0.0000	0.7763	0.8735	0.6388
25	0.0803	0.3799	0.6020	0.4991	**95**	0.9189	0.5930	0.7288	0.8989	**165**	0.4395	0.0501	0.9761	0.5712
26	0.1908	0.8046	0.5402	0.6665	**96**	0.0382	0.5515	0.8818	0.5999	**166**	0.9359	0.0366	0.9514	0.6826
27	0.6937	0.3967	0.6055	0.7595	**97**	0.3726	0.9988	0.3814	0.7086	**167**	0.0173	0.9548	0.4289	0.5527
28	0.2591	0.0582	0.3978	0.3604	**98**	0.4211	0.2668	0.3307	0.5080	**168**	0.6112	0.9070	0.6286	0.8803
29	0.4241	0.1850	0.9066	0.6298	**99**	0.2378	0.0817	0.3574	0.3452	**169**	0.2010	0.9573	0.6791	0.7283
30	0.3332	0.9303	0.2475	0.6287	**100**	0.9893	0.7637	0.2526	0.7755	**170**	0.8914	0.9144	0.2641	0.7966
31	0.3625	0.1592	0.9981	0.5948	**101**	0.8203	0.0682	0.4260	0.5643	**171**	0.0061	0.0802	0.8621	0.3711
32	0.9259	0.0960	0.1645	0.4716	**102**	0.6226	0.2146	0.1021	0.4452	**172**	0.2212	0.4664	0.3821	0.5260
33	0.8606	0.6779	0.0033	0.6242	**103**	0.4589	0.3147	0.2236	0.4962	**173**	0.2401	0.6964	0.0751	0.4637
34	0.0838	0.5472	0.3758	0.4835	**104**	0.3471	0.8889	0.1564	0.5875	**174**	0.7881	0.9833	0.3038	0.8049
35	0.0303	0.9191	0.7233	0.6491	**105**	0.5762	0.8292	0.4116	0.7853	**175**	0.2435	0.0794	0.5551	0.4223
36	0.9293	0.8319	0.9664	0.9840	**106**	0.9053	0.6245	0.5264	0.8506	**176**	0.2752	0.8414	0.2797	0.6079
37	0.7268	0.1440	0.9753	0.7096	**107**	0.2860	0.0793	0.0549	0.2224	**177**	0.7616	0.4698	0.5337	0.7809
38	0.2888	0.6593	0.4078	0.6328	**108**	0.9567	0.3034	0.4425	0.6993	**178**	0.3395	0.0022	0.0087	0.1836
39	0.5515	0.1364	0.2894	0.4745	**109**	0.5170	0.9266	0.1565	0.6594	**179**	0.7849	0.9981	0.4449	0.8641
40	0.7683	0.0067	0.5546	0.5708	**110**	0.8149	0.0396	0.6227	0.6165	**180**	0.8312	0.0961	0.2129	0.4857
41	0.6462	0.6761	0.8340	0.8933	**111**	0.3710	0.3554	0.5633	0.6171	**181**	0.9763	0.1102	0.6227	0.6667
42	0.3694	0.2212	0.1233	0.3658	**112**	0.8702	0.3185	0.2762	0.6287	**182**	0.8597	0.3284	0.6932	0.7829
43	0.2706	0.3222	0.9996	0.6310	**113**	0.1016	0.6382	0.3173	0.4957	**183**	0.9295	0.3275	0.7536	0.8016
44	0.6282	0.1404	0.8474	0.6733	**114**	0.3890	0.2369	0.0083	0.3235	**184**	0.2435	0.2163	0.7625	0.5449
45	0.5861	0.6693	0.3818	0.7433	**115**	0.2702	0.8617	0.1218	0.5319	**185**	0.9281	0.8356	0.5285	0.8991
46	0.6057	0.9901	0.5141	0.8466	**116**	0.7473	0.6507	0.5582	0.8464	**186**	0.8313	0.7566	0.6192	0.9047
47	0.5915	0.5588	0.3055	0.6787	**117**	0.9108	0.2139	0.4641	0.6625	**187**	0.1712	0.0545	0.5033	0.3561
48	0.8359	0.4145	0.5016	0.7597	**118**	0.4343	0.6028	0.1344	0.5546	**188**	0.0609	0.1702	0.4306	0.3310
49	0.5497	0.6319	0.8382	0.8521	**119**	0.6847	0.4062	0.9318	0.8204	**189**	0.5899	0.9408	0.0369	0.6245
50	0.7072	0.1721	0.3812	0.5772	**120**	0.8657	0.9448	0.9900	0.9904	**190**	0.7858	0.5115	0.0916	0.6066
51	0.1185	0.5084	0.8376	0.6211	**121**	0.4011	0.4138	0.8715	0.7222	**191**	1.0000	0.1653	0.7103	0.7172
52	0.6365	0.5562	0.4965	0.7693	**122**	0.5949	0.2600	0.0810	0.4480	**192**	0.2007	0.1163	0.3431	0.3385
53	0.4145	0.5797	0.8599	0.7878	**123**	0.1845	0.7906	0.9725	0.7425	**193**	0.2306	0.0330	0.0293	0.1590
54	0.2575	0.5358	0.4028	0.5777	**124**	0.3438	0.6725	0.9821	0.7926	**194**	0.8477	0.6378	0.4623	0.8254
55	0.2026	0.3300	0.3054	0.4261	**125**	0.8398	0.1360	0.9119	0.7222	**195**	0.9677	0.7895	0.9467	0.9782
56	0.3385	0.0476	0.5941	0.4625	**126**	0.2245	0.0971	0.6136	0.4402	**196**	0.0339	0.4669	0.1526	0.3250
57	0.4094	0.1726	0.7803	0.6015	**127**	0.3742	0.9668	0.8194	0.8371	**197**	0.0080	0.8988	0.4201	0.5404
58	0.1261	0.6181	0.4927	0.5739	**128**	0.9572	0.9836	0.3793	0.8556	**198**	0.9955	0.8897	0.6175	0.9360
59	0.1224	0.4662	0.2146	0.4007	**129**	0.7496	0.0410	0.1360	0.4059	**199**	0.7408	0.5351	0.2732	0.6949
60	0.6793	0.6774	1.0000	0.9141	**130**	0.9123	0.3510	0.0682	0.5455	**200**	0.6843	0.3737	0.1562	0.5625
61	0.8176	0.0358	0.2506	0.4707	**131**	0.6954	0.5500	0.6801	0.8388					
62	0.6937	0.6685	0.5075	0.8220	**132**	0.5252	0.6529	0.5729	0.7893					
63	0.2404	0.5411	0.8754	0.6980	**133**	0.3156	0.3851	0.5983	0.6161					
64	0.6553	0.2609	0.1188	0.4851	**134**	0.1460	0.1637	0.0249	0.1813					
65	0.8886	0.0288	0.2604	0.4802	**135**	0.7780	0.4491	0.4614	0.7498					
66	0.3974	0.5275	0.6457	0.7215	**136**	0.5959	0.8647	0.8601	0.9176					
67	0.2108	0.4910	0.5432	0.5913	**137**	0.2204	0.1785	0.4607	0.4276					
68	0.8675	0.5571	0.1849	0.6805	**138**	0.7355	0.8264	0.7015	0.9214					
69	0.5693	0.0242	0.9293	0.6033	**139**	0.9931	0.6727	0.3139	0.7829					
70	0.8439	0.4631	0.6345	0.8226	**140**	0.9123	0.0000	0.1106	0.3944					

```python
import pandas as pd

train = pd.read_csv("./dataset/ressonanciaMLP.csv")
test = pd.read_csv("./dataset/ressonanciaMLPTest.csv")

base = pd.concat([train, test])

base
```

```
        x1      x2      x3       d
0   0.8799  0.7998  0.3972  0.8399
1   0.5700  0.5111  0.2418  0.6258
2   0.6796  0.4117  0.3370  0.6622
3   0.3567  0.2967  0.6037  0.5969
4   0.3866  0.8390  0.0232  0.5316
..     ...     ...     ...     ...
15  0.8882  0.3077  0.8931  0.8093
16  0.2225  0.9182  0.7820  0.7581
17  0.1957  0.8423  0.3085  0.5826
18  0.9991  0.5914  0.3933  0.7938
19  0.2299  0.1524  0.7353  0.5012

[220 rows x 4 columns]
```

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error

# Carregar e pré-processar os dados
class DataLoader:
    @staticmethod
    def load_data(path):
        data = pd.read_csv(path)
        return data

class DataPreprocessor:
    def __init__(self, df):
        self.df = df

    def split_features_and_target(self, target_column='d'):
        X = self.df[['x1', 'x2', 'x3']]
        y = self.df[target_column]
        return X, y

    def normalize_features(self, X):
        scaler = StandardScaler().fit(X)
        X = scaler.transform(X)
        return X, scaler
```

```python
class MLPTrainer:
    def __init__(self, activation, hidden_layer_sizes,
learning_rate_init=0.01, max_iter=1000, random_state=42):
        self.model = MLPRegressor(
            activation=activation,
            hidden_layer_sizes=hidden_layer_sizes,
            learning_rate_init=learning_rate_init,
            max_iter=max_iter,
            random_state=random_state
        )

    def train(self, X_train, y_train):
        self.model.fit(X_train, y_train)
        return self.model

    def evaluate(self, X_test, y_test):
        y_pred = self.model.predict(X_test)
        loss = mean_squared_error(y_test, y_pred)
        return loss, self.model.n_iter_




# Carregar os dados
data = base.copy()


data
```

	x1	x2	x3	d
0	0.8799	0.7998	0.3972	0.8399
1	0.5700	0.5111	0.2418	0.6258
2	0.6796	0.4117	0.3370	0.6622
3	0.3567	0.2967	0.6037	0.5969
4	0.3866	0.8390	0.0232	0.5316
..	...	...	...	...
15	0.8882	0.3077	0.8931	0.8093
16	0.2225	0.9182	0.7820	0.7581
17	0.1957	0.8423	0.3085	0.5826
18	0.9991	0.5914	0.3933	0.7938
19	0.2299	0.1524	0.7353	0.5012

[220 rows x 4 columns]

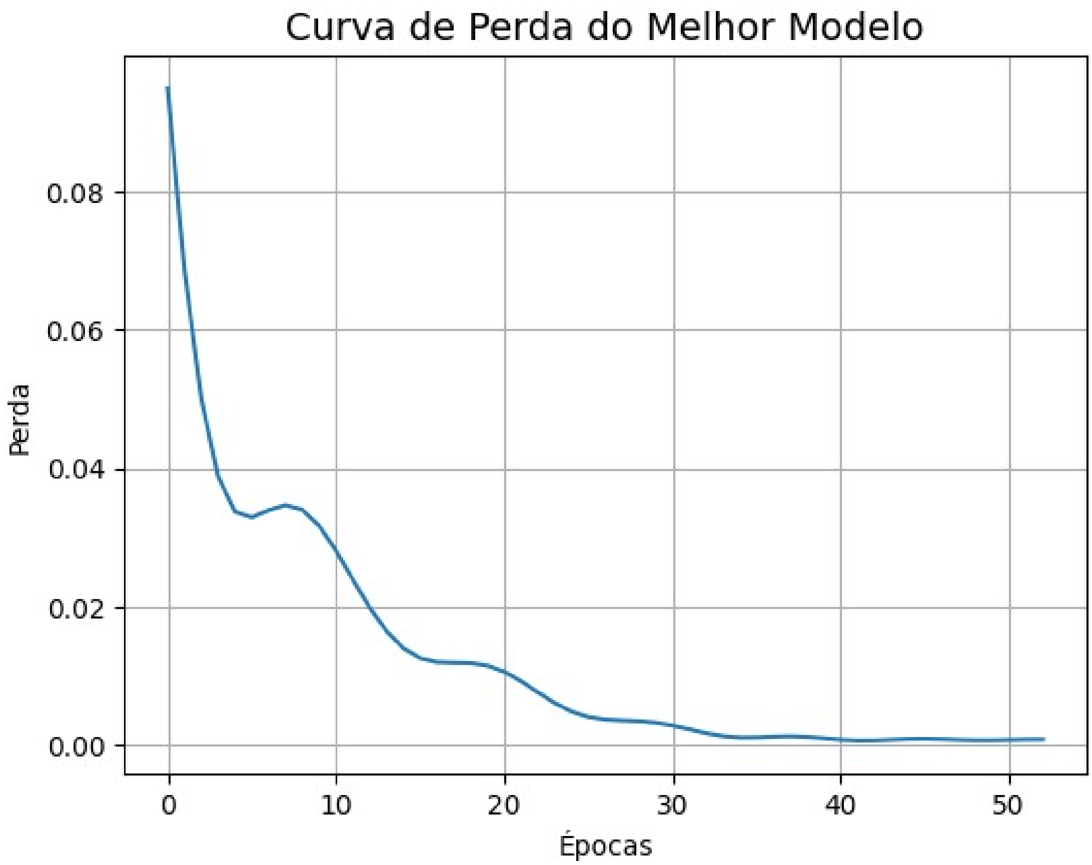
```python
# Pré-processar os dados
preprocessor = DataPreprocessor(data)
X, y = preprocessor.split_features_and_target()
#X, scaler = preprocessor.normalize_features(X)

from prettytable import PrettyTable
```

```python
# Dividir os dados em conjunto de treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
# Configurações para os treinamentos
activations = ['tanh', 'relu', 'logistic']
neurons = [5, 10, 15]
results = []
# Executar os treinamentos
for activation in activations:
    for neuron in neurons:
        trainer = MLPTrainer(activation=activation,
hidden_layer_sizes=(neuron,))
        model = trainer.train(X_train, y_train)
        loss, epochs = trainer.evaluate(X_test, y_test)
        results.append((activation, neuron, loss, epochs, model))

    # Classificar os resultados
results.sort(key=lambda x: x[2])

table = PrettyTable()
table.field_names = ["Treinamento", "Neurônios", "Ativação", "Perda
Final", "Número de Épocas", "Model"]

for i, (activation, neuron, loss, epochs, model) in enumerate(results,
start=1):
        table.add_row([f"{i}º (T{i})", neuron, activation,
f"{loss:.4f}", epochs, model])

print(table)
```

```
+-------------+-----------+----------+-------------+------------------
+------------------------------------------------------------------
-----------+
| Treinamento | Neurônios | Ativação | Perda Final | Número de Épocas
|                                        Model
|
+-------------+-----------+----------+-------------+------------------
+------------------------------------------------------------------
-----------+
|   1º (T1)   |     15    |   relu   |    0.0015   |        53
| MLPRegressor(hidden_layer_sizes=(15,), learning_rate_init=0.01,
max_iter=1000, |
|             |           |          |             |
|                                       random_state=42)
|
|   2º (T2)   |     10    |   relu   |    0.0015   |        65
| MLPRegressor(hidden_layer_sizes=(10,), learning_rate_init=0.01,
max_iter=1000, |
|             |           |          |             |
```

```
|                                      random_state=42)
|
|   3º (T3)   |     5     |   tanh   |    0.0017   |       127
|            MLPRegressor(activation='tanh', hidden_layer_sizes=(5,),
|
|             |           |          |             |
|                  learning_rate_init=0.01, max_iter=1000,
random_state=42)      |
|   4º (T4)   |     15    |   tanh   |    0.0031   |        28
|           MLPRegressor(activation='tanh', hidden_layer_sizes=(15,),
|
|             |           |          |             |
|                  learning_rate_init=0.01, max_iter=1000,
random_state=42)      |
|   5º (T5)   |     15    | logistic |    0.0168   |        18
|         MLPRegressor(activation='logistic',
hidden_layer_sizes=(15,),          |
|             |           |          |             |
|                  learning_rate_init=0.01, max_iter=1000,
random_state=42)      |
|   6º (T6)   |     10    | logistic |    0.0218   |        20
|         MLPRegressor(activation='logistic',
hidden_layer_sizes=(10,),          |
|             |           |          |             |
|                  learning_rate_init=0.01, max_iter=1000,
random_state=42)      |
|   7º (T7)   |     5     | logistic |    0.0251   |        35
|          MLPRegressor(activation='logistic',
hidden_layer_sizes=(5,),          |
|             |           |          |             |
|                  learning_rate_init=0.01, max_iter=1000,
random_state=42)      |
|   8º (T8)   |     5     |   relu   |    0.0301   |        79
| MLPRegressor(hidden_layer_sizes=(5,), learning_rate_init=0.01,
max_iter=1000,  |
|             |           |          |             |
|                                      random_state=42)
|
|   9º (T9)   |     10    |   tanh   |    0.0591   |        23
|           MLPRegressor(activation='tanh', hidden_layer_sizes=(10,),
|
|             |           |          |             |
|                  learning_rate_init=0.01, max_iter=1000,
random_state=42)      |
+-------------+-----------+----------+-------------+-----------------
+-------------------------------------------------------------------
-----------+

import matplotlib.pyplot as plt
```

```
best_model = results[0][4]
plt.plot(best_model.loss_curve_)
plt.title("Curva de Perda do Melhor Modelo", fontsize=14)
plt.xlabel('Épocas')
plt.ylabel('Perda')
plt.grid(True)
plt.show()
```



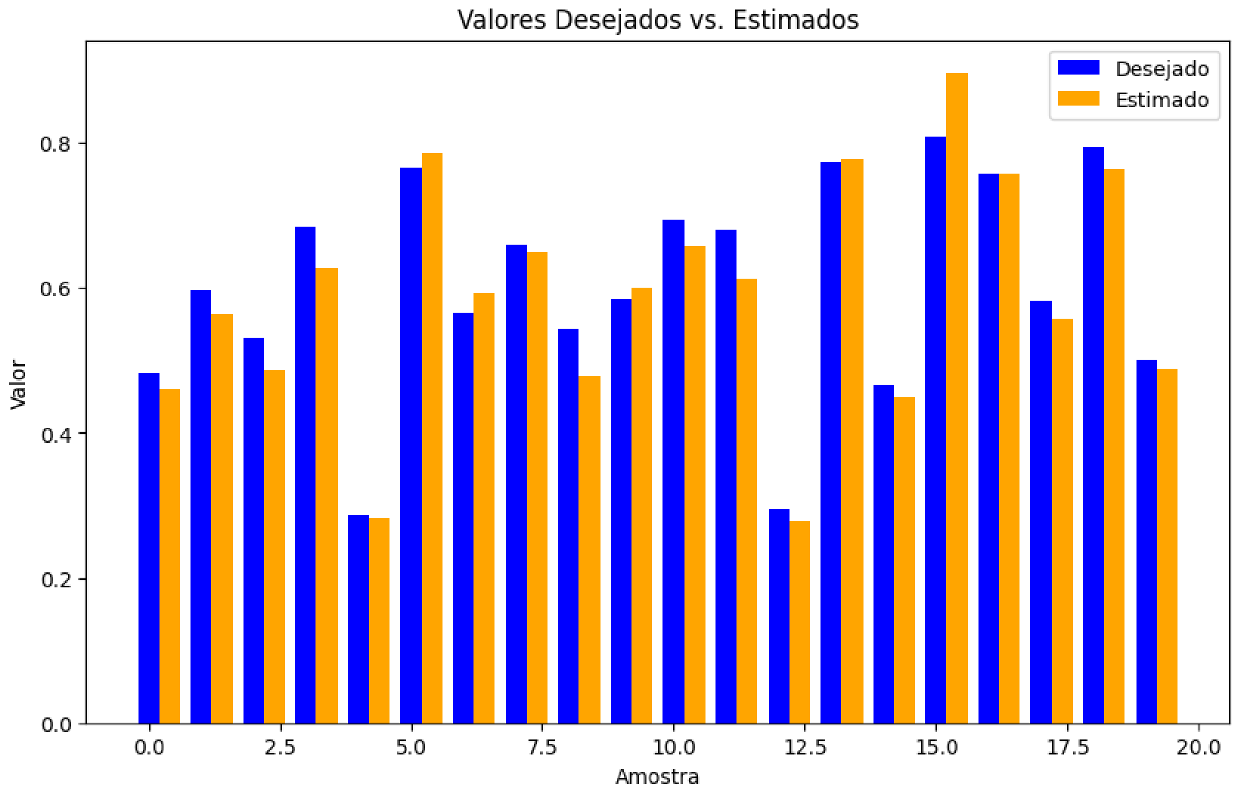

```
results
```

```
[('relu', 15, 0.0015170114658609683, 53),
 ('relu', 10, 0.0015295566920569925, 65),
 ('tanh', 5, 0.001735316666974427, 127),
 ('tanh', 15, 0.003109737315709396, 28),
 ('logistic', 15, 0.01680438358616946, 18),
 ('logistic', 10, 0.021775989215754365, 20),
 ('logistic', 5, 0.025114968531775538, 35),
 ('relu', 5, 0.03007950069396447, 79),
 ('tanh', 10, 0.05913046287692675, 23)]
```

```
import pandas as pd
import numpy as np
```

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
import matplotlib.pyplot as plt

# Dados de teste fornecidos
test_data = {
    'x1': [0.0611, 0.5102, 0.0004, 0.9430, 0.1399, 0.6423, 0.6492,
0.1818, 0.7382, 0.3879,
           0.1903, 0.8401, 0.0029, 0.7088, 0.1283, 0.8882, 0.2225,
0.1957, 0.9991, 0.2299],
    'x2': [0.2860, 0.7464, 0.6916, 0.4476, 0.1610, 0.3229, 0.0007,
0.5078, 0.2647, 0.1307,
           0.6523, 0.4490, 0.3264, 0.9342, 0.1882, 0.3077, 0.9182,
0.8423, 0.5914, 0.1524],
    'x3': [0.7464, 0.0860, 0.5006, 0.2648, 0.2477, 0.8567, 0.6422,
0.9046, 0.1916, 0.8656,
           0.7820, 0.2719, 0.2476, 0.2763, 0.7253, 0.8931, 0.7820,
0.3085, 0.3933, 0.7353],
    'd': [0.4831, 0.5965, 0.5318, 0.6843, 0.2872, 0.7663, 0.5666,
0.6601, 0.5427, 0.5836,
          0.6950, 0.6790, 0.2956, 0.7742, 0.4662, 0.8093, 0.7581,
0.5826, 0.7938, 0.5012]
}

test = pd.DataFrame(test_data)
# Realizar a predição com o modelo treinado
y_pred = best_model.predict(test[['x1', 'x2', 'x3']])

# Calcular as métricas de avaliação
mae = mean_absolute_error(test_data['d'], y_pred)
mse = mean_squared_error(test_data['d'], y_pred)
rmse = np.sqrt(mse)
mape = np.mean(np.abs((test_data['d'] - y_pred) / test_data['d'])) *
100

# Plotar um gráfico de barras comparando os valores estimados e
desejados
plt.figure(figsize=(10, 6))
plt.bar(np.arange(len(test_data['d'])), test_data['d'], width=0.4,
label='Desejado', color='blue', align='center')
plt.bar(np.arange(len(y_pred)) + 0.4, y_pred, width=0.4,
label='Estimado', color='orange', align='center')
plt.xlabel('Amostra')
plt.ylabel('Valor')
plt.title('Valores Desejados vs. Estimados')
plt.legend()
plt.show()

# Exibir as métricas de avaliação
print('Mean Absolute Error (MAE):', mae)
print('Mean Squared Error (MSE):', mse)
```

```
print('Root Mean Squared Error (RMSE):', rmse)
print('Mean Absolute Percentage Error (MAPE):', mape)
```




```
Mean Absolute Error (MAE): 0.02990738521857437
Mean Squared Error (MSE): 0.001412530598836247
Root Mean Squared Error (RMSE): 0.03758364802458972
Mean Absolute Percentage Error (MAPE): 4.948898357013116
```