Otimização de Hiperparâmetros e Métodos Avançados em Redes Neurais

Ismael Weslley Neves de Brito

Objetivo do Projeto

Desenvolver um sistema de apoio ao aprendizado de matemática básica utilizando redes neurais focando na otimização avançada de hiperparâmetros, esse modelo deve ser capaz de realizar operações simples como somar, subtrair, multiplicar e dividir.

Serão utilizados callbacks, como early stopping e monitoramento de métricas, a fim de otimizar o processo de treinamento e garantir melhor convergência do modelo.

Visão Geral do Projeto

Preparação dos dados: geração de dataset sintético com operações matemáticas, divisão em conjuntos de treino, validação e teste, e normalização dos dados.

Arquitetura da rede neural: definição de uma **MLP** com múltiplas camadas, aplicação de técnicas de regularização e comparação entre diferentes funções de ativação.

Otimização de hiperparâmetros: utilização do **Keras Tuner** e testes com diferentes otimizadores.

Treinamento do modelo: implementação de callbacks como **Early Stopping**, **Model Checkpoint** e **TensorBoard**, seguido pelo treinamento e avaliação do modelo.

Resultados: análise e visualização do desempenho obtido.

Bibliotecas Utilizadas e Ambiente

As seguintes bibliotecas foram utilizadas ao longo do desenvolvimento do projeto:

- **TensorFlow/Keras:** Framework para construção e treinamento de redes neurais
- **Keras Tuner:** Biblioteca para otimização de hiperparâmetros
- **NumPy/Pandas:** Manipulação e análise de dados
- **Matplotlib:** Visualização de dados e resultados
- **Scikit-learn:** Ferramentas para pré-processamento de dados
- **Plotly Express:** Visualização de gráficos interativos

O ambiente utilizado para o desenvolvimento do projeto foi o **Google Colab**

Parâmetros Globais do Projeto

```
FAIXA_NUMEROS = (-10, 10)                                  # Faixa de números inteiros usados como entrada (operandos)
TRIALS = 15                                                # Número de combinações de hiperparâmetros a serem testadas durante a busca
N_SAMPLES = 10000                                          # Quantidade total de amostras geradas para o dataset
TEST_SIZE = 0.2                                            # Proporção dos dados reservados para teste
VAL_SIZE = 0.2                                             # Proporção dos dados reservados para validação
SCALER_TYPE = 'standard'                                   # Tipo de normalização: 'standard' para StandardScaler
INPUT_SHAPE = 6                                            # Tamanho do vetor de entrada: 2 números + 4 operações codificadas one-hot
EPOCHS = 100                                               # Número máximo de épocas de treinamento
BATCH_SIZE = 32                                            # Tamanho do lote usado durante o treinamento
NUM_NEURONS_CHOICES = [128, 256]                           # Quantidade de neurônios possíveis por camada densa
OPTIMIZERS = ['rmsprop', 'adagrad', 'nadam']               # Otimizadores testados durante a busca
ACTIVATIONS = ['relu', 'leaky_relu', 'tanh']               # Funções de ativação permitidas
L2_REG_VALUES = [0.001, 0.01]                              # Valores possíveis de regularização L2 para evitar overfitting
DROPOUT_RATES = [0.0, 0.1, 0.2]                            # Taxas de dropout possíveis entre camadas
LEARNING_RATES = [0.05, 0.1]                               # Taxas de aprendizado testadas (ajustadas para valores mais baixos)
NUM_LAYERS_CHOICES = [3, 4, 5]                             # Quantidade de camadas densas no modelo
HYPERBAND_FACTOR = 3                                       # Fator de redução usado no Hyperband (impacta alocação de recursos por tentativa)
EARLY_STOPPING_PATIENCE = 5                                # Número de épocas sem melhoria para acionar early stopping
TUNING_DIRECTORY = 'tuning_dir'                            # Diretório onde os resultados da busca de hiperparâmetros serão salvos
PROJECT_NAME = 'math_mlp'                                  # Nome do projeto de tuning para identificação
TENSORBOARD_LOG_DIR = "./tuner_logs"                       # Diretório onde os logs para o TensorBoard serão armazenados
MODEL_FILENAME = 'melhor_modelo.keras'                     # Nome do arquivo onde o melhor modelo será salvo
RANDOM_STATE = 42                                          # Semente fixa para reprodutibilidade dos resultados
```

Geração do Dataset

Foi gerado para o treinamento um dataset de operações com um total de 10 mil operações.

Contendo uma distribuição equilibrada entre as 4 operações (adição, subtração, multiplicação, divisão)

Foi utilizada a codificação one-hot para representação das operações

	num1	num2	operacao_+	operacao_-	operacao_*	operacao_/	resultado
0	4.293851	2.506245	1	0	0	0	6.800095
1	7.371265	-0.546863	1	0	0	0	6.824403
2	-6.998244	-3.016155	1	0	0	0	-10.014399
3	-1.412271	-1.463740	1	0	0	0	-2.876011
4	-2.582232	2.893141	1	0	0	0	0.310910
...	...	...	...	...	...	...	...
9995	1.088798	-0.600000	0	0	0	1	-1.814663
9996	-6.215613	0.600000	0	0	0	1	-10.359355
9997	-0.819206	0.600000	0	0	0	1	-1.365343
9998	9.928068	-0.600000	0	0	0	1	-16.546781
9999	8.007519	0.571014	0	0	0	1	14.023345

10000 rows × 8 columns

Divisão e Normalização dos Dados

Divisão:

Treino (60%): usado para o aprendizado do modelo — é o conjunto de dados com o qual a rede realmente ajustar seus pesos.

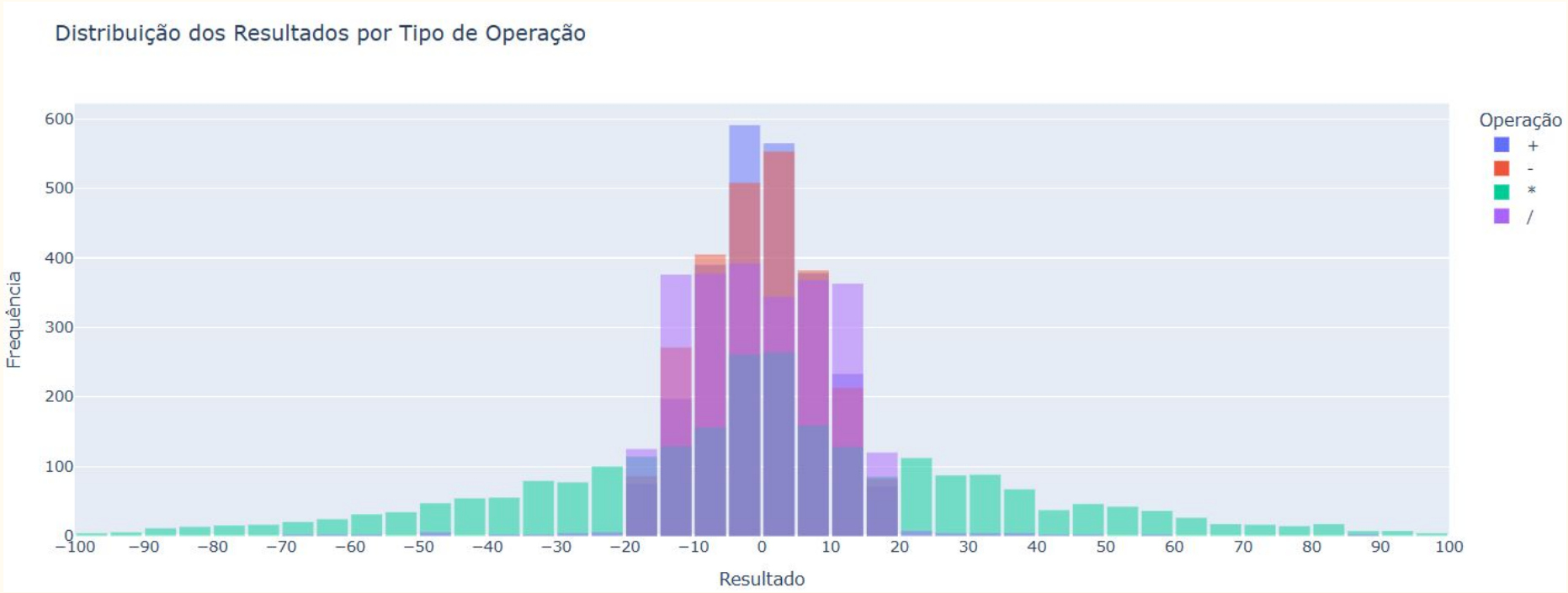
Validação (20%): usado no o treino para monitorar o desempenho do modelo e ajustar hiperparâmetros, ajudando a evitar overfitting.

Teste (20%): usado após o treino completo para avaliar o desempenho final e generalização do modelo em dados nunca vistos.

Normalização:

Foi utilizado o método **StandardScaler** (Média = 0 e desvio padrão = 1), aplicado apenas às colunas numéricas (num1, num2), ajustada no conjunto de treino e aplicada aos conjuntos de validação e teste para acelerar convergência e melhora estabilidade numérica

Visualização da Distribuição dos Dados



Observações da base

Algumas observações que podem ser feitas com relação a essa base é:

- **Multiplicação** e **divisão** geram valores com **maior amplitude**
- **Adição** e **subtração** produzem distribuições **mais concentradas**, ficando entre -20 e 20

Implicando para o modelo:

- Necessidade de **normalização** para lidar com **diferentes escalas**
- Possível desafio para o modelo em **valores extremos** da Multiplicação e divisão

Arquitetura da Rede Neural

Estrutura Geral

- **Modelo:** Perceptron Multicamadas (MLP)
- **Camada de entrada:** 6 neurônios (2 números + 4 operações one-hot)
- **Camadas ocultas:** Variável (3-5 camadas, definido por hiperparâmetros)
- **Camada de saída:** 1 neurônio (resultado da operação)
- **Função de perda:** Mean Squared Error (MSE)
- **Métrica de avaliação:** Mean Absolute Error (MAE)

Técnicas de Regularização

- **Dropout:** Taxas de 0.0, 0.1 ou 0.2 (definido por hiperparâmetros)
- **Regularização L2:** Valores de 0.001 ou 0.01 (definido por hiperparâmetros)
- **Early Stopping:** Parada antecipada se não houver melhoria em 5 épocas

Funções de Ativação

Funções Utilizadas:

- **ReLU (Rectified Linear Unit):** $f(x) = \max(0, x)$
- **LeakyReLU:** $f(x) = x$ se $x > 0$, αx caso contrário (α pequeno)
- **Tanh (Tangente Hiperbólica):** $f(x) = \tanh(x)$

Comparação:

- **ReLU:** Simples, eficiente, mas pode sofrer com "neurônios mortos"
- **LeakyReLU:** Soluciona o problema de neurônios mortos do ReLU
- **Tanh:** Saídas entre -1 e 1, pode sofrer com desvanecimento do gradiente

Construção do Modelo

```python
def build_model(hp):
    activation = hp.Choice('activation', values=ACTIVATIONS)
    l2_reg = hp.Float('l2_reg', min_value=min(L2_REG_VALUES), max_value=max(L2_REG_VALUES), sampling='log')
    dropout_rate = hp.Choice('dropout_rate', values=DROPOUT_RATES)
    optimizer_name = hp.Choice('optimizer', values=OPTIMIZERS)
    num_neurons = hp.Choice('num_neurons', values=NUM_NEURONS_CHOICES)
    num_layers = hp.Choice('num_layers', values=NUM_LAYERS_CHOICES)
    learning_rate = hp.Float('learning_rate', min_value=min(LEARNING_RATES), max_value=max(LEARNING_RATES), sampling='linear')
    model = keras.Sequential()
    model.add(layers.Input(shape=(INPUT_SHAPE,)))

    for _ in range(num_layers):
        model.add(layers.Dense(num_neurons, kernel_regularizer=keras.regularizers.l2(l2_reg)))
        if activation == 'leaky_relu':
            model.add(layers.LeakyReLU())
        else:
            model.add(layers.Activation(activation))
        model.add(layers.Dropout(dropout_rate))

    model.add(layers.Dense(1))

    if optimizer_name == 'rmsprop':
        optimizer = RMSprop(learning_rate=learning_rate)
    elif optimizer_name == 'adagrad':
        optimizer = Adagrad(learning_rate=learning_rate)
    elif optimizer_name == 'nadam':
        optimizer = Nadam(learning_rate=learning_rate)
    else:
        raise ValueError(f"[ERRO] Otimizador desconhecido: {optimizer_name}")

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

Otimização de Hiperparâmetros

Biblioteca: Keras Tuner

Algoritmo: RandomSearch (busca aleatória no espaço de hiperparâmetros)

Objetivo: Minimizar o **MAE** (Erro Absoluto Médio) no conjunto de validação

Número de tentativas: 15

Hiperparâmetros Otimizados:

- **Número de camadas:** 3, 4 ou 5
- **Função de ativação:** ReLU, LeakyReLU ou Tanh
- **Coeficiente de regularização L2:** 0.001 ou 0.01
- **Otimizador:** RMSprop, Adagrad ou Nadam
- **Neurônios por camada:** 128 ou 256
- **Taxa de dropout:** 0.0, 0.1 ou 0.2
- **Taxa de aprendizado:** 0.05 até 0.1

Otimizadores Testados

RMSprop: Adapta a taxa de aprendizado dividindo o gradiente pela média móvel dos quadrados dos gradientes. É eficiente em problemas não-estacionários e redes recorrentes (RNNs).

Adagrad: Ajusta a taxa de aprendizado individualmente para cada parâmetro, reduzindo-a mais para os que acumulam gradientes grandes. Pode acabar diminuindo demais a taxa com o tempo.

Nadam: Combina Adam com Nesterov Momentum, acelerando a convergência e mantendo boa estabilidade. É útil em muitas tarefas por equilibrar desempenho e velocidade.

Implementação de Callbacks

Early Stopping:

- Monitora: val_loss

- Paciência: 5 épocas

- Previne overfitting parando o treinamento quando não há melhoria

Model Checkpoint:

- Salva o melhor modelo durante o treinamento

- Critério: menor val_loss

- Arquivo: 'melhor_modelo.keras'

TensorBoard:

- Registra métricas para visualização

- Permite análise detalhada do treinamento

- Diretório de logs: './tuner_logs'

Métricas para avaliação

mae (Mean Absolute Error): É o erro absoluto médio entre as previsões do modelo e os valores reais nos dados de treinamento. Ele calcula a média da diferença absoluta entre os valores previstos e os valores reais.

loss: É a função de perda que o modelo está tentando minimizar durante o treinamento, também calculada sobre os dados de treinamento. Pode ser o próprio MAE, MSE (erro quadrático médio) ou outra função, dependendo da configuração.

val_mae (Validation Mean Absolute Error): É o MAE calculado sobre os dados de validação, que não são vistos pelo modelo durante o treinamento direto. Serve para avaliar o desempenho do modelo em dados "novos".

val_loss: É a função de perda (como o MAE ou MSE) aplicada sobre os dados de validação, semelhante ao val_mae, mas depende da função de perda usada.

Comparação dos modelos

modelo	camadas	neuronios por camada	otimizador	activation	learning_rate	l2_reg	dropout_rate	mae	loss	val_mae	val_loss
11	5	128	adagrad	tanh	0.095003	0.003273	0.0	11.317249	317.165009	[0.6001347899436951]	[12.788652420043945]
08	5	128	adagrad	leaky_relu	0.097998	0.002535	0.0	11.363183	318.123230	[0.696820080280304]	[9.325026512145996]
01	5	256	nadam	relu	0.054517	0.003144	0.0	11.361139	319.743195	[0.8422998189926147]	[15.404980659484863]
10	3	256	nadam	leaky_relu	0.085549	0.001073	0.0	11.360888	317.329376	[0.9448903203010559]	[13.792068481445312]
12	3	256	adagrad	leaky_relu	0.070621	0.001108	0.2	11.365726	317.611237	[1.0794795751571655]	[9.76041316986084]
13	3	128	adagrad	leaky_relu	0.050076	0.003049	0.2	11.364016	317.644958	[1.1289807558059692]	[9.774173736572266]
07	4	256	adagrad	relu	0.059727	0.002637	0.2	11.363213	318.650726	[1.2216414213180542]	[10.269089698791504]
06	3	128	rmsprop	tanh	0.060184	0.001583	0.2	11.384130	317.339966	[5.292263984680176]	[132.81378173828125]
02	4	128	rmsprop	tanh	0.096400	0.001103	0.1	11.332492	316.269836	[8.03134536743164]	[231.41094970703125]
04	4	256	rmsprop	tanh	0.068528	0.002107	0.0	11.358430	318.052307	[8.827977180480957]	[271.6308898925781]
05	3	256	rmsprop	tanh	0.074808	0.006523	0.0	11.352406	319.632935	[9.379322052001953]	[466.8419494628906]
09	5	128	rmsprop	tanh	0.091359	0.005137	0.0	11.340245	318.435577	[10.266425132751465]	[344.32476806640625]
14	5	128	nadam	tanh	0.056667	0.003291	0.1	11.417828	319.775360	[10.548226356506348]	[310.9977722167969]
03	5	128	rmsprop	leaky_relu	0.084212	0.002993	0.1	11.360936	317.957123	[232.54251098632812]	[87848.40625]
00	4	256	rmsprop	leaky_relu	0.077408	0.001093	0.1	11.359689	317.509552	[342.54632568359375]	[162545.6875]

Melhores Hiperparâmetros Encontrados

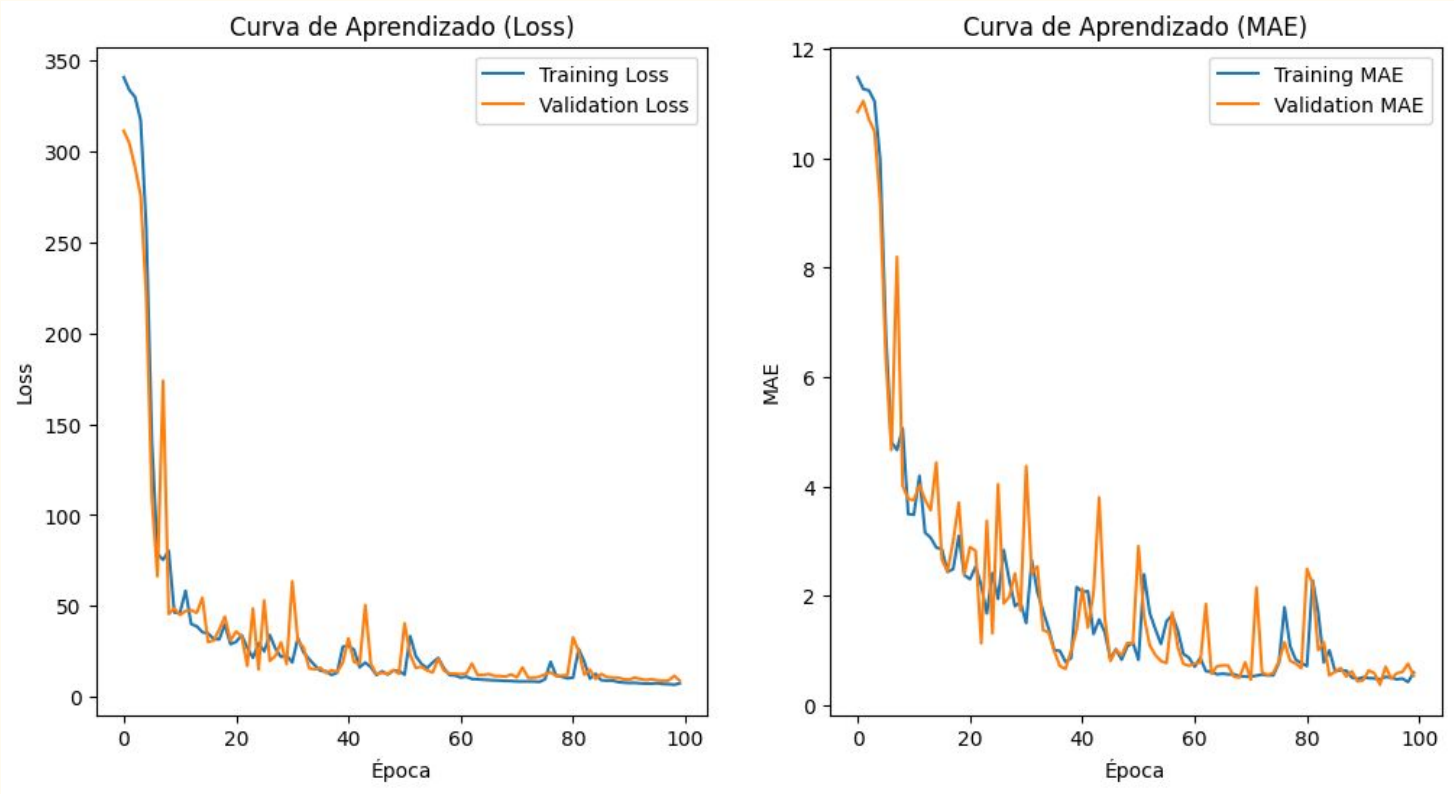
Resultados da Busca:

- Função de Ativação: tanh
- Taxa de Aprendizado: 0.095002910957782 72
- Coeficiente de Regularização L2: 0.0032729393230571 23
- Dropout: 0.0
- Otimizador: adagrad
- Número de Neurônios por camada: 128
- Número de Camadas: 5

Treinamento do melhor modelo

```
Epoch 90/100
188/188 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 6.8832 - mae: 0.4482 - val_loss: 9.6002 - val_mae: 0.4325
Epoch 91/100
188/188 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 8.3144 - mae: 0.5607 - val_loss: 9.4004 - val_mae: 0.4513
Epoch 92/100
188/188 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 8.6919 - mae: 0.5060 - val_loss: 10.5377 - val_mae: 0.6318
Epoch 93/100
188/188 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - loss: 7.7296 - mae: 0.5151 - val_loss: 9.7047 - val_mae: 0.5824
Epoch 94/100
188/188 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - loss: 6.8072 - mae: 0.4442 - val_loss: 9.1858 - val_mae: 0.3714
Epoch 95/100
188/188 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 7.6655 - mae: 0.5751 - val_loss: 9.5946 - val_mae: 0.7020
Epoch 96/100
188/188 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 6.8889 - mae: 0.4828 - val_loss: 9.0438 - val_mae: 0.4817
Epoch 97/100
188/188 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 6.6802 - mae: 0.4364 - val_loss: 8.7839 - val_mae: 0.5819
Epoch 98/100
188/188 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 6.6694 - mae: 0.5761 - val_loss: 8.9199 - val_mae: 0.6045
Epoch 99/100
188/188 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 5.5942 - mae: 0.3680 - val_loss: 11.4390 - val_mae: 0.7552
Epoch 100/100
188/188 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 7.1831 - mae: 0.5685 - val_loss: 8.7952 - val_mae: 0.5318
Loss no conjunto de teste: 7.3892
MAE no conjunto de teste: 0.5576
```

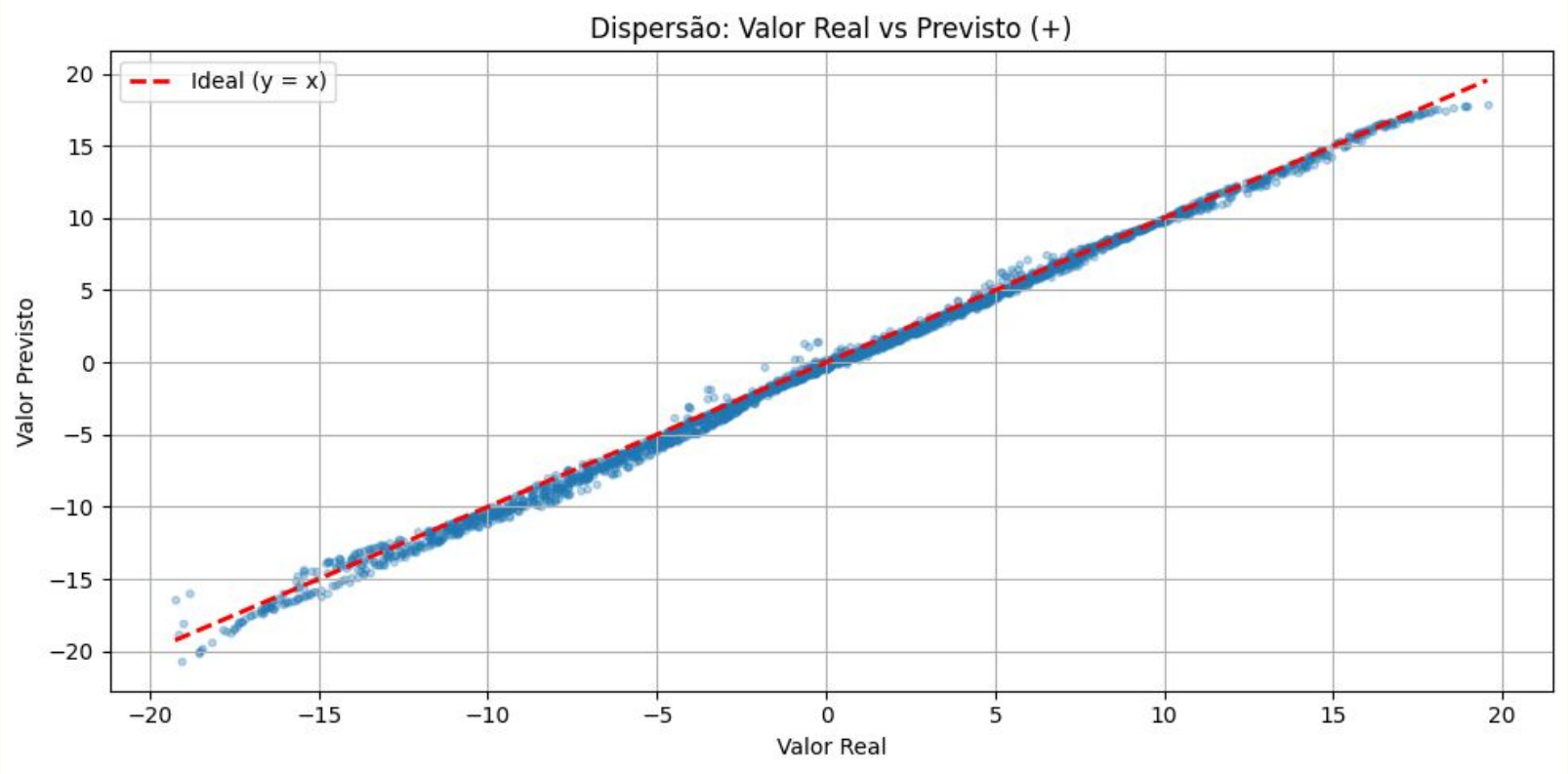
Curva de aprendizado



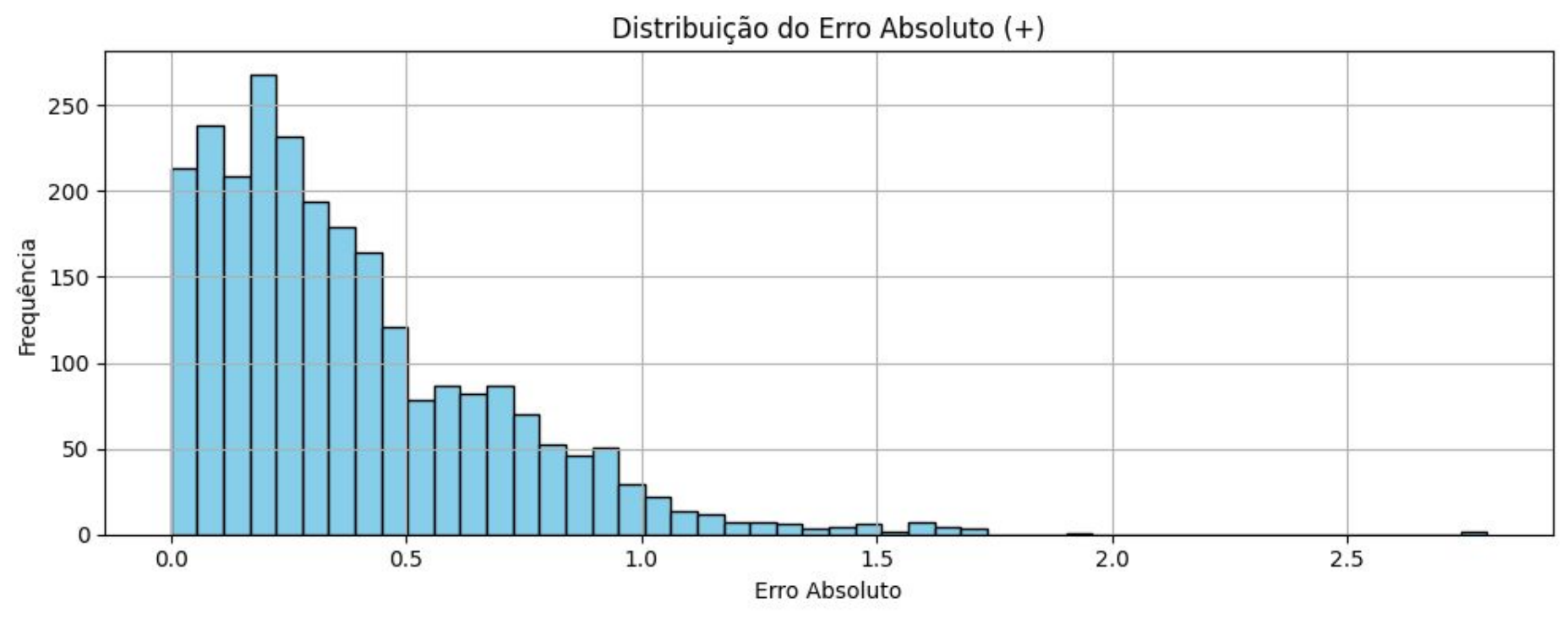
Teste com Novas Equações

Expressão	Previsto	Real	Erro Absoluto
4.93 + 4.02	8.9561	8.9494	0.0067
-0.36 * 4.02	-1.3772	-1.4449	0.0678
-3.65 - -2.07	-1.6622	-1.5872	0.0750
-9.39 + -1.78	-11.0855	-11.1721	0.0866
-7.48 - -2.36	-5.2212	-5.1159	0.1053
-1.73 * 2.93	-4.9612	-5.0674	0.1062
-2.11 * 7.56	-16.0634	-15.9502	0.1132
-5.72 - 4.82	-10.6653	-10.5479	0.1174
-5.39 * -7.31	39.5141	39.3623	0.1518
-4.76 - 5.10	-10.0234	-9.8616	0.1618
5.05 / -0.60	-8.5909	-8.4200	0.1709
-6.30 / -0.60	10.7374	10.4959	0.2415
-4.11 + -3.85	-7.7178	-7.9621	0.2443
-0.02 - 6.50	-6.8118	-6.5153	0.2965
2.38 / 0.60	3.6532	3.9747	0.3215
6.86 * -9.21	-63.4798	-63.1445	0.3353
7.08 / -0.60	-12.1417	-11.7997	0.3420
-4.72 + 7.68	2.5463	2.9601	0.4139
-2.45 + 4.87	1.7786	2.4217	0.6431
4.84 / -0.23	-25.7482	-21.3752	4.3730

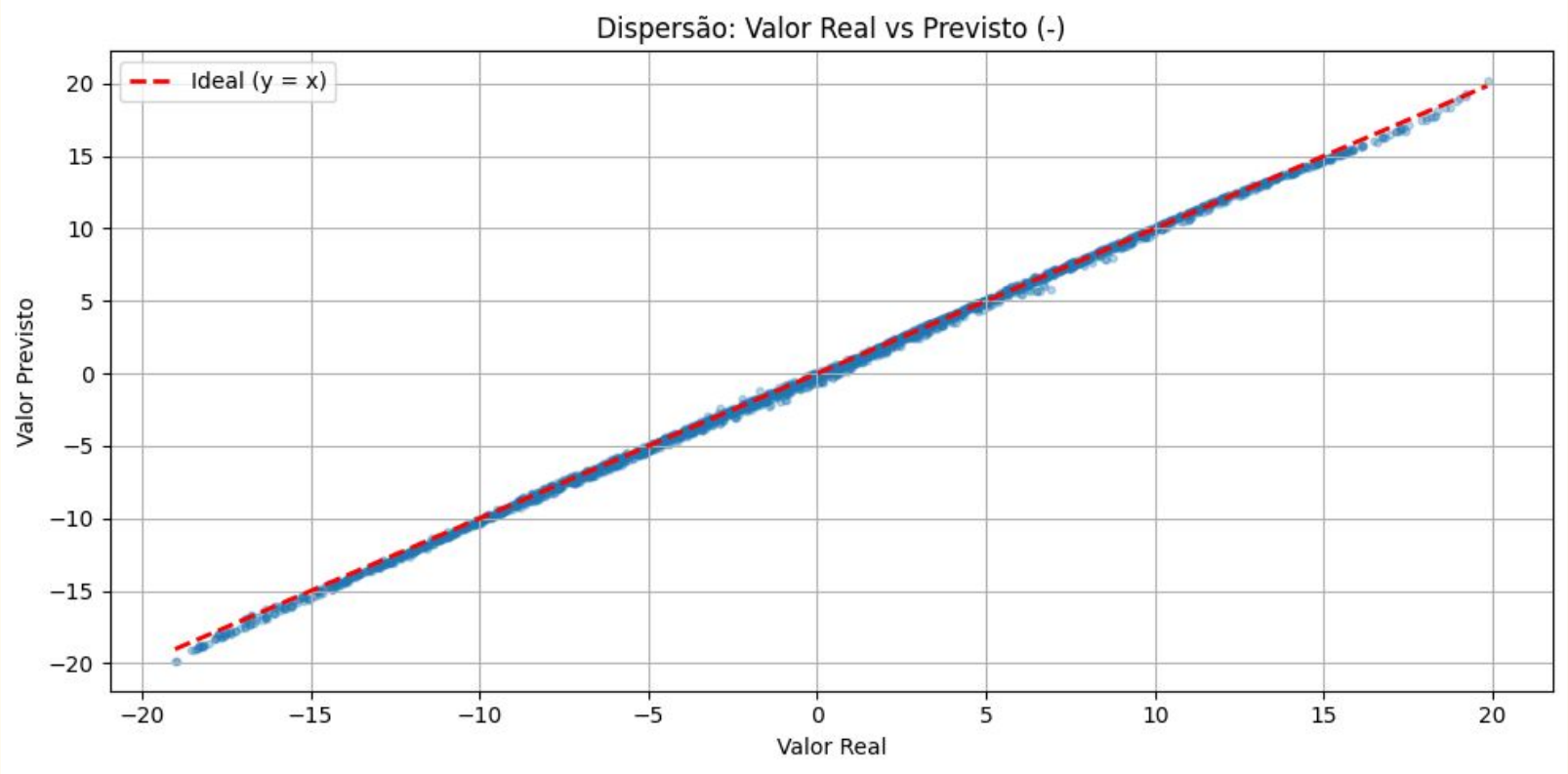
Teste com 10 mil somas



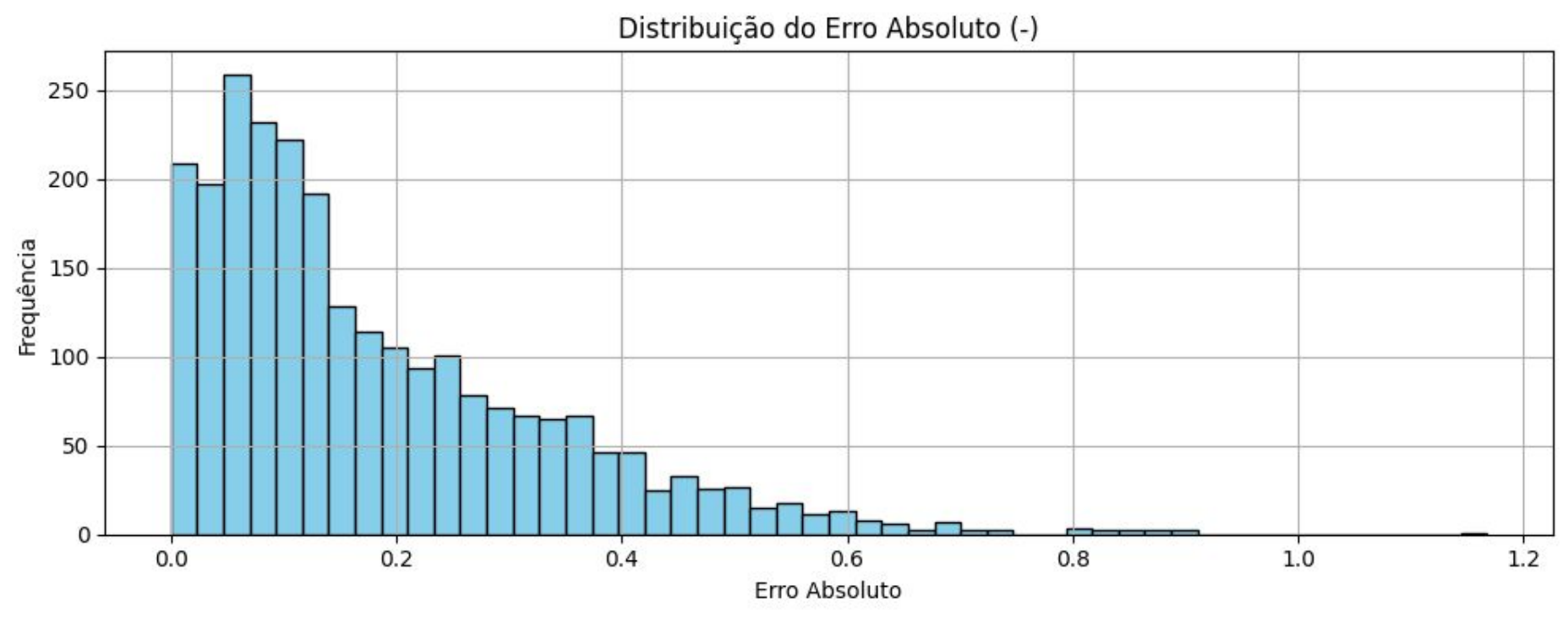
Teste com 10 mil somas



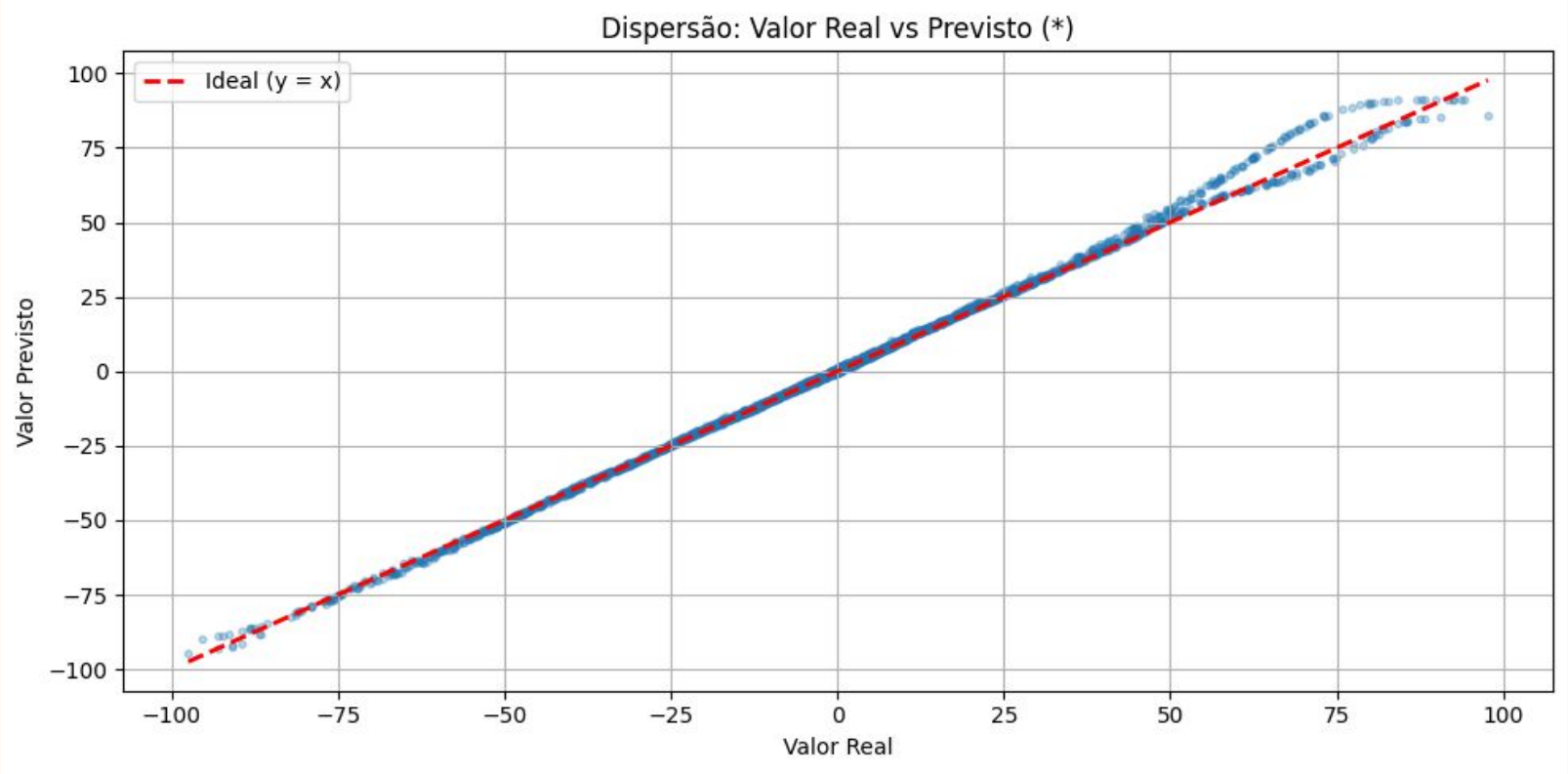
Teste com 10 mil subtrações



Teste com 10 mil subtrações

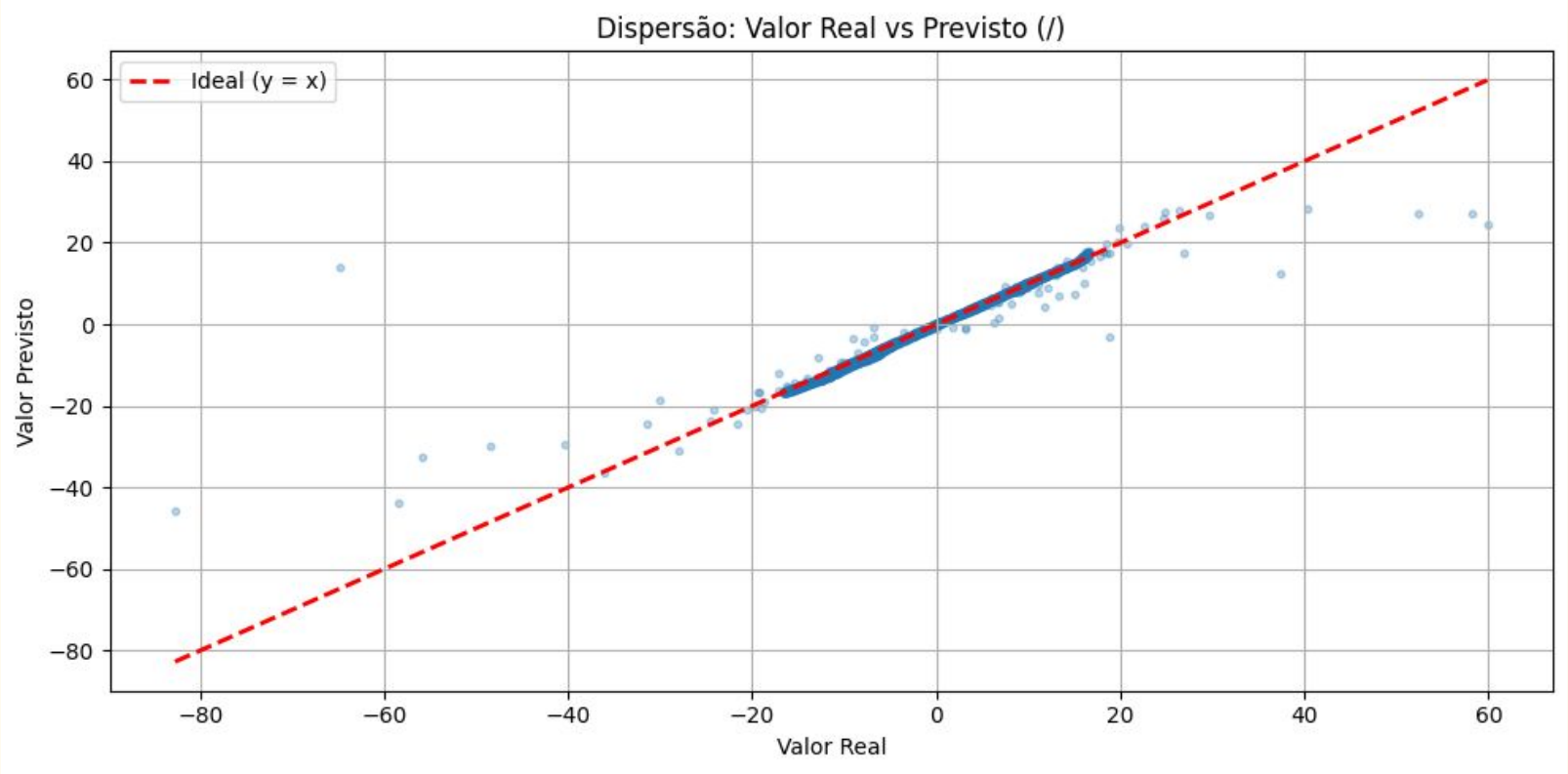


Teste com 10 mil multiplicações

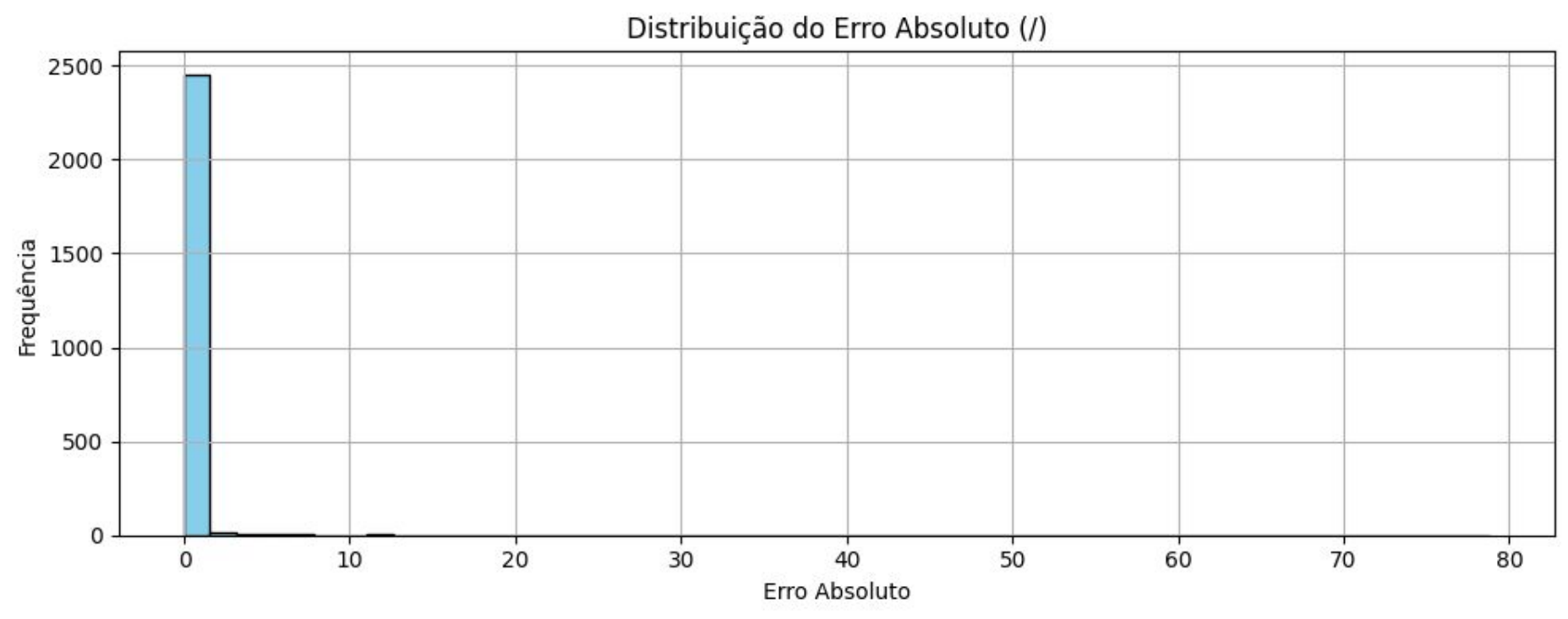


Teste com 10 mil multiplicações

Teste com 10 mil divisões



Teste com 10 mil divisões



Conclusões

O modelo conseguiu aprender com sucesso as quatro operações matemáticas básicas no universo observado, embora como esperado, com as operações de multiplicação e divisão ele teve uma maior dificuldade de acompanhar os dados

Possíveis melhorias futuras

Melhorias futuras incluem interface interativa para aprendizado, aumentar a faixa de valores do treino, treinar um modelo com operações mais avançadas como potenciação, raízes e funções trigonométricas.

OBRIGADO!