Redes Neurais Artificiais

Marcos Vinicius Silva - 202204192
Tamyris Cabral Gomes Rocha - 202204193

02 de junho de 2025

1 Introdução

Este estudo explora o uso de Redes Neurais Artificiais (RNA) para solucionar o problema "House Prices - Advanced Regression Techniques" disponibilizado na plataforma Kaggle.

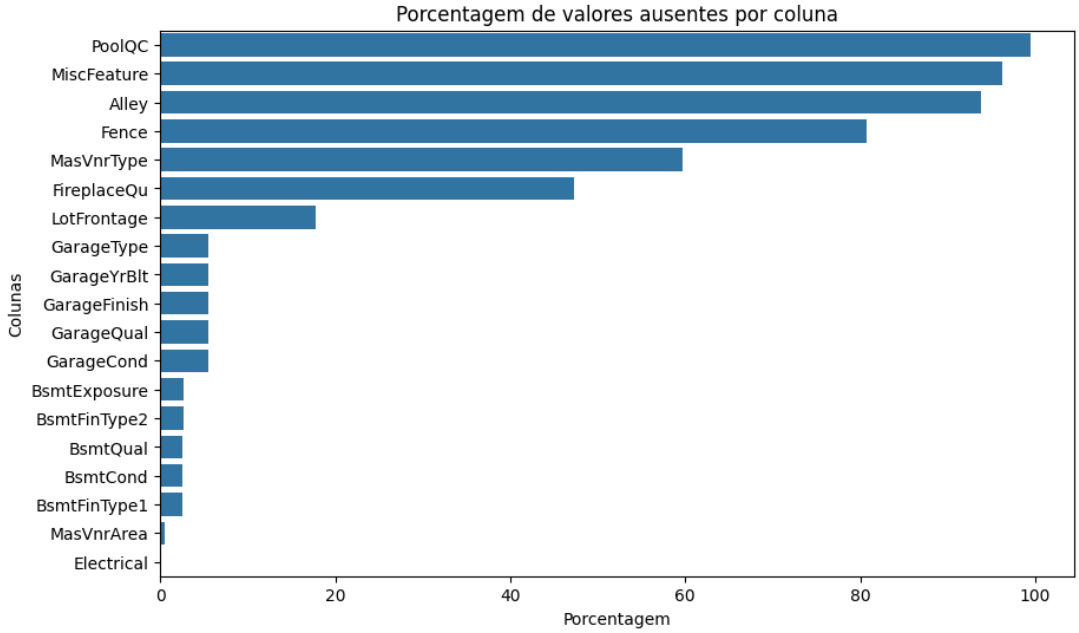
Para tanto, foi desenvolvido um modelo de rede neural para identificar características e padrões de imóveis com o objetivo de prever seu respectivo preço. Este relatório está dividido em quatro partes:

- 1. Dados
- 2. Modelo
- 3. Experimentos
- 4. Conclusões

2 Dados

Análise de nulos

Primeiramente foi necessário entender a natureza dos dados fornecidos, olhando para tipos e quantidade de nulos.

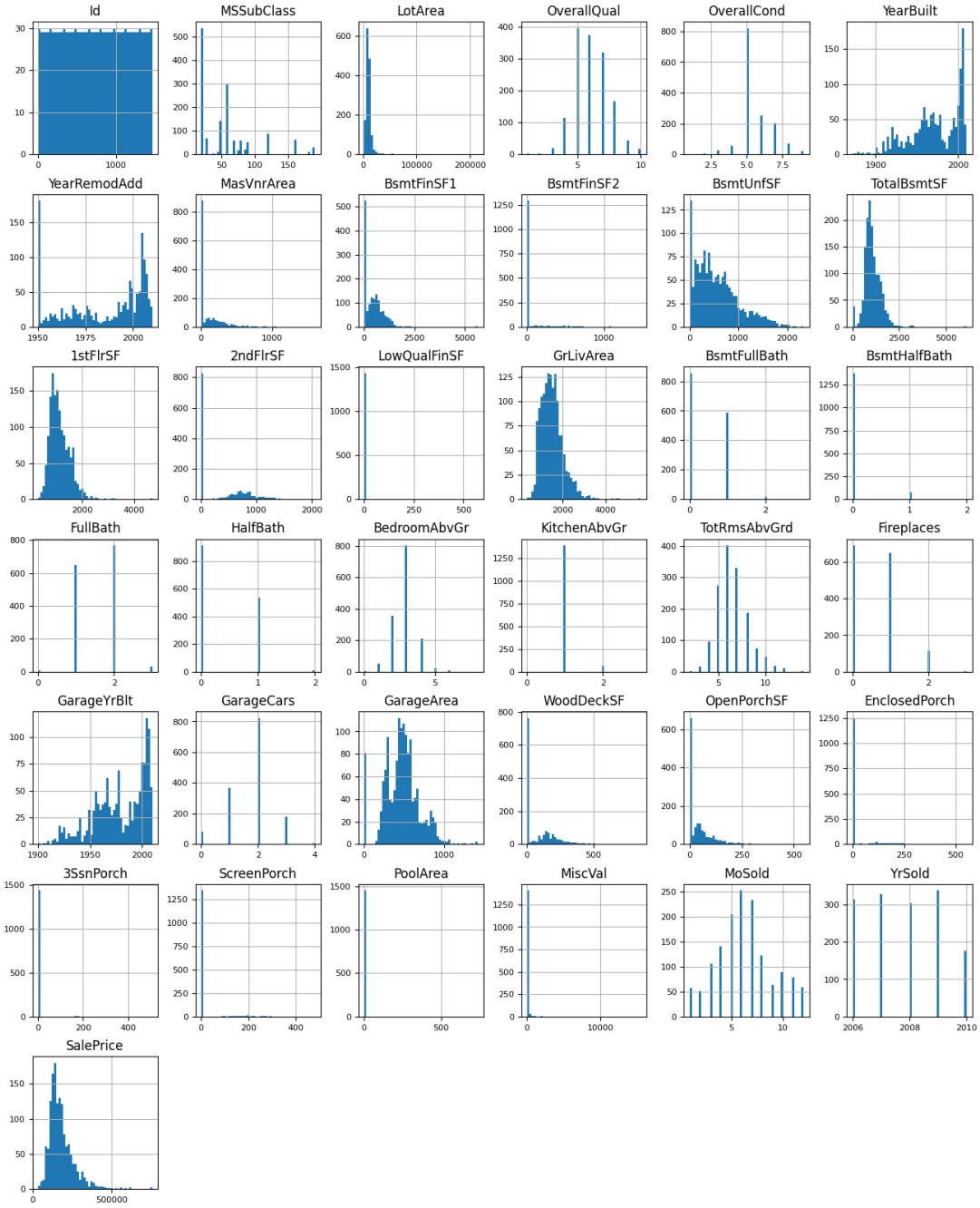


É possível notar a grande quantidade de nulos nas colunas PoolQC, MiscFeature, Alley, Fence, MasVnrType, FireplaceQu e LotFrontage. Para esse problema, colunas com 20% dos dados faltantes foram removidos para evitar viés nos dados.

Comportamento dos dados

Figura abaixo apresenta uma visão geral das distribuições de todas as variáveis numéricas do conjunto de dados. Cada histograma ilustra a frequência de valores observados em cada coluna, permitindo identificar padrões como assimetrias e concentrações em faixas específicas.

- SalePrice: A variável de preço de venda está concentrada em uma faixa de preço intermediária de (em sua maioria) até 500K USD.
- LotArea: Apresenta forte assimetria, com a maioria dos terresnos concetrados me áreas menores, o que explica a maioria dos imóveis considerados em uma faixa de preço não muito expressiva
- OverallQual e OverallCond: Ambas as variáveis, que medem a qualidade geral e a condição geral do imóvel, tendem a se agrupar em torno de valores médios (5–7), sugerindo que poucas casas estão em condição ou qualidade extremas.
- Banheiros e Quartos (BsmtFullBath, BsmtHalfBath, FullBath, HalfBath, BedroomAbvGr): A maior parte dos lares concentra-se em 2–3 banheiros totais e 3–4 quartos, com poucos casos extremos (por exemplo, casas sem banheiro no porão ou com mais de 5 quartos)
- **Escala dos dados**: O ponto mais importante de se notar nessa imagem é que as variáveis possuem escalas muito diferentes, tipo Qualidade (0 a 10) vs Preco (0 a 750K), isso pode causar gradientes desiguais, ocasionando em features de maior magnitude dominando o aprendizado e causando uma convergência lenta e estável, logo, é necessário normalizar os dados.



Tratamento dos dados

Preenchimento de espaços *NaN*

Após uma análise dos dados da planilha, percebeu-se que era necessário substituir espaços preenchidos com NaN(*"Not a Number"*) por informações que possam ser compreendidas pelo modelo.

Na categoria *"MasVnrType"*, que informa sobre o tipo de revestimento externo em alvenaria, os espaços que continham NaN foram subtituídos por *None*, informando que aquele imóvel não possui revestimento. Consequentemente, nesses imóveis, a categoria *"MasVnrArea"*, que informa a área coberta pelo revestimento, foi preenchida com 0.

Dentre os informativos sobre o porão, estão presentes as categorias *"BsmtQual"* , *"BsmtCond"*, *"BsmtExposure"*, *"BsmtFinType1"*, *"BsmtFinType2"*. Essas se tornam desnecessárias

quando a casa não possui porão. Portanto, nesses casos, o espaço foi preenchido com *"None"*. O mesmo procedimento foi utilizado para categorias relacionadas à garagem.

Para o sistema elétrico, aquelas casas que não possuíam informações sobre o tipo do sistema não poderiam ter uma classificação como nula, afinal, todas as casas tem um sistema. Pensando nisso, utilizou-se o tipo mais comum *"SBrkr"* para completar os espaços.

Características numéricas e categóricas

Para as categorias numéricas, em sua maioria, foi aplicada uma normalização utilizando a técnica min-max que consiste na seguinte formula:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

A exceção foi a categoria alvo: *SalePrice*.

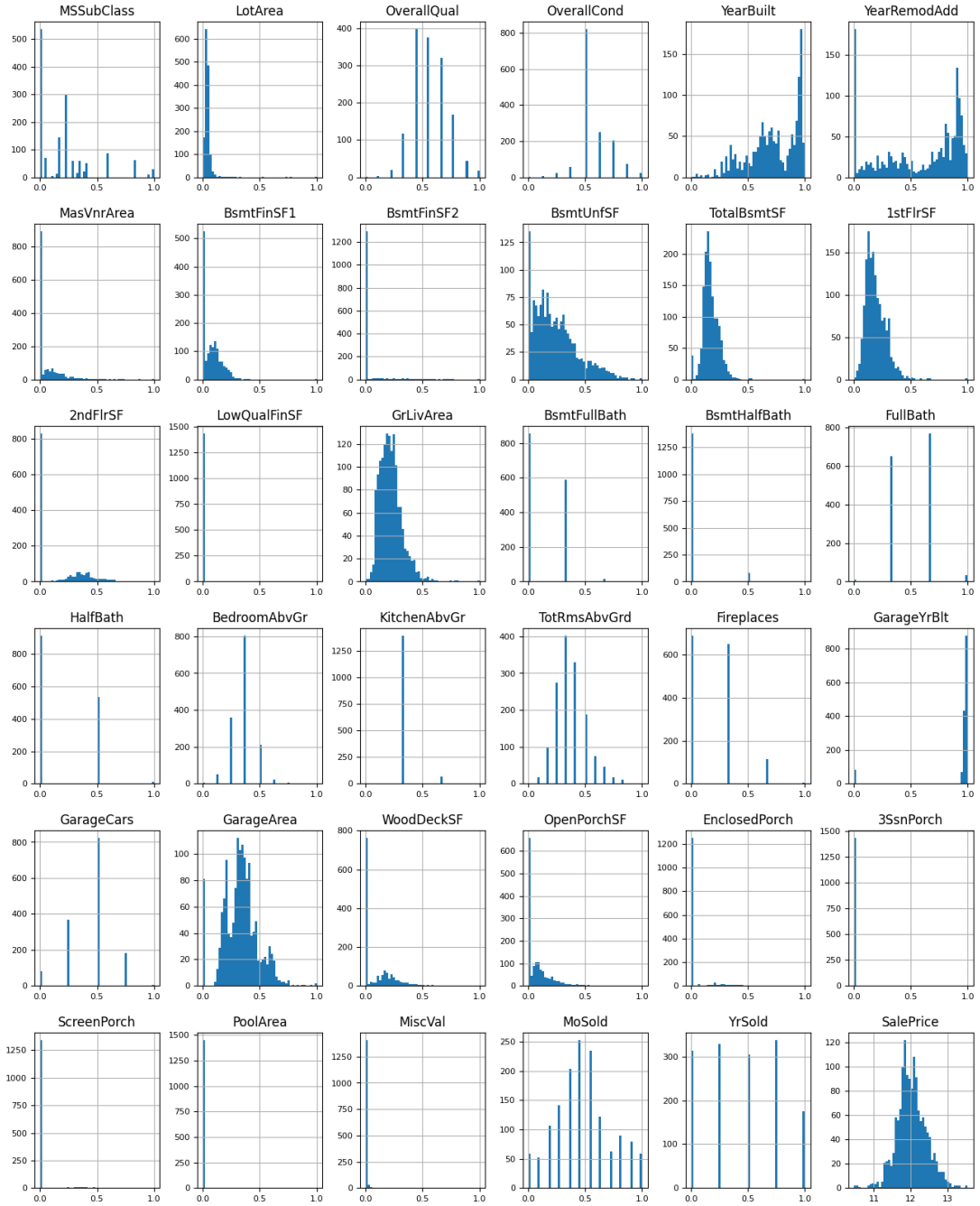
A variável *SalePrice* apresentava uma distribuição assimétrica, com valores concentrados em faixas médias e alguns outliers com preços muito altos. Para reduzir esse viés e facilitar o aprendizado do modelo, foi aplicada a transformação logarítmica $log1p$, que consiste em $log(SalePrice+1)$. Essa técnica reduz a influência de outliers, aproxima a distribuição de uma forma mais normal (gaussiana) e melhora a estabilidade e performance durante o treinamento da rede neural.

Para as variáveis categóricas, foi utilizado o método *One-Hot Encoding*, que cria uma nova coluna binária para cada categoria possível. Essa abordagem permite que o modelo trate categorias como características independentes, evitando atribuir relações numéricas incorretas entre elas.

Por exemplo, a variável *Neighborhood* possui 4 classificações possíveis. Para separá-las, são criadas 4 colunas, uma para cada classificação. A casa pertencente a cada vizinhança recebe 1 em sua categoria e as outras recebem 0.

Neighborhood	CollgCr	OldTown	NAmes	Somerst
CollgCr	1	0	0	0
OldTown	0	1	0	0
NAmes	0	0	1	0
Somerst	0	0	0	1

A figura abaixo apresenta uma visão geral das distribuições de todas as variáveis numericas do conjunto de dados normalizadas utilizando o min-max, note que o formato dos histogramas é o mesmo, entretando é alterada a escala deles sendo os valores entre 0 e 1



3 Modelo

Para o problema de regressão, foi implementado um modelo do tipo Perceptron Multicamadas (MLP) com três camadas lineares e ativação não linear ReLU. Essa abordagem foi escolhida por ser versátil, eficiente e amplamente utilizada em situações de regressão com dados em tabelas. O MLP é ideal para aprendizado supervisionado com dados tabulares, pois consegue capturar interações não lineares entre as variáveis.

```
class MLP(nn.Module):
    def __init__(self, input_size):
        super(MLP, self).__init__()
        self.dropout = nn.Dropout(0.2)
        self.layer1 = nn.Linear(input_size, 32)
```

```
        self.layer2 = nn.Linear(32, 16)
        self.layer3 = nn.Linear(16, 1)
        self.activation = nn.ReLU()

        nn.init.kaiming_normal_(self.layer1.weight, mode='fan_in',
    nonlinearity='relu')
        nn.init.kaiming_normal_(self.layer2.weight, mode='fan_in',
    nonlinearity='relu')
        nn.init.kaiming_normal_(self.layer3.weight, mode='fan_in',
    nonlinearity='relu')
        nn.init.zeros_(self.layer1.bias)
        nn.init.zeros_(self.layer2.bias)
        nn.init.zeros_(self.layer3.bias)

    def forward(self, x):
        x = self.dropout(self.activation(self.layer1(x)))
        x = self.dropout(self.activation(self.layer2(x)))
        return self.layer3(x)
```

Formatação da Rede

Para a camada de entrada, utilizou-se o número de atributos extraídos após o pré-processamento com a variável *input_size*. Depois, são utilizadas duas camadas ocultas, com 32 e 16 neurônios respectivamente. Por fim, há uma camada de saída que gera a estimativa de preço.

Função de ativação

A função escolhida foi a ReLU (Rectified Linear Unit) devido a sua capacidade de obter soluções não lineares, essenciais para o padrão complexo do problema. Além disso, ela também evita o problema do gradiente evanescente e é eficiente computacionalmente.

Inicialização dos Pesos

Para inicializar os pesos, foi adotado o método *Kaiming Normal(He)*. Essa técnica é especialmente adequada para redes que utilizam a função de ativação ReLU, pois leva em consideração a variância dos neurônios da camada anterior ao calcular os pesos iniciais. O objetivo principal é evitar que os valores dos gradientes se tornem muito pequenos ou muito grandes durante a propagação do erro, o que poderia dificultar o treinamento. Ao manter a escala das ativações estável entre as camadas, a inicialização Kaiming contribui para uma convergência mais rápida e estável do modelo, especialmente em arquiteturas profundas ou com muitas conexões.

Regularização

Durante testes iniciais, foi constatado overfitting nos treinamentos. Para mitigar esse problema, foi aplicado um dropout com taxa de 20% nas camadas ocultas como forma de regularização, forçando a rede a não depender de combinações específicas de neurônios.

Função de treino

A função *"train_model_house"* realiza o ciclo completo de treinamento e avaliação da rede neural por um número definido de épocas. Parâmetros principais:

- **model:** o modelo MLP (rede neural) definido anteriormente;
- **X_train, y_train:** dados de entrada e saída (preço) para treinamento;
- **X_test, y_test:** dados de entrada e saída para avaliação durante o treinamento;
- **EPOCHS:** número de vezes que o modelo verá todos os dados de treino;
- **LEARNING_RATE:** taxa de aprendizado usada pelo otimizador (SGD);
- **BATCH_SIZE:** número de amostras usadas por atualização de gradiente;
- **DEVICE:** define que o modelo será treinado em 'cpu'.

Para atingir seu objetivo, a função executa algumas etapas. Primeiramente, ela configura o modelo e os dados no dispositivo(CPU), inicializa o otimizador com a taxa de aprendizado fornecida e cria listas para armazenar o erro de treino e teste a cada época. Após isso, ela entra no loop de treinamente, onde, para cada época, embaralha os dados e devide em batches conforme o tamanho definido. Para cada mini-batch, zera os gradientes, faz a predição, calcula a loss, realiza a backpropagation e atualiza os pesos. Após cada época, avalia o modelo no conjunto de teste, armazena as pernas de treino e teste e atualiza a barra de progresso. Por fim, ela retorna a lista com o histórico de perdas(*train_losses e test_losses*.

Pós-treinamento

Após o treinamento da rede neural com a função *train_model_house*, foram obtidas as listas contendo os valores da função de perda (loss) para cada época, tanto no conjunto de treinamento quanto no conjunto de validação. Esses valores foram utilizados para gerar gráficos que mostram a evolução do erro ao longo do tempo. A análise dessas curvas permite verificar se o modelo está aprendendo adequadamente e ajuda a identificar possíveis sinais de overfitting. Essa visualização é fundamental para avaliar o desempenho da rede e guiar ajustes nos hiperparâmetros e na arquitetura.

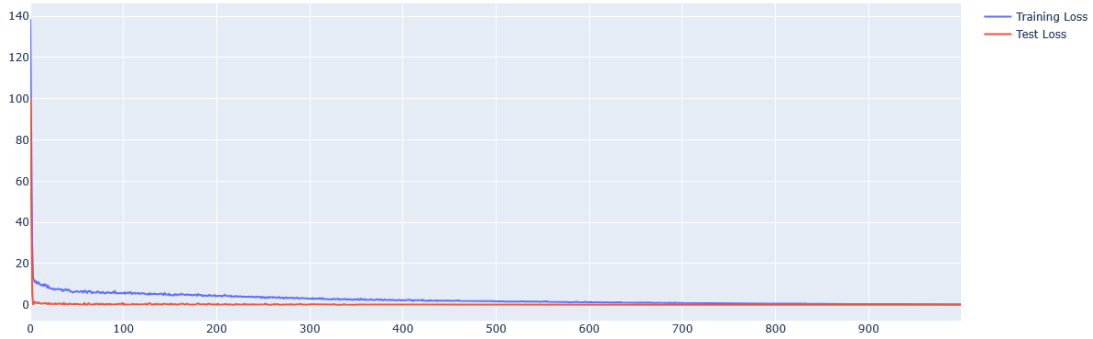
4 Experimentos

Para a escolha do otimizador foram testados 3 modelos: Adam, SGD e RMSprop.

Adam

Quando ananlisamos o gráfico gerado pelo aprendizado do modelo, é possível notar que as curvas começam com valores altos, especialmente a de treino. Há uma queda rápida nas primeiras épocas, o que indica que o modelo está aprendendo e se ajustando rapidamente aos dados.

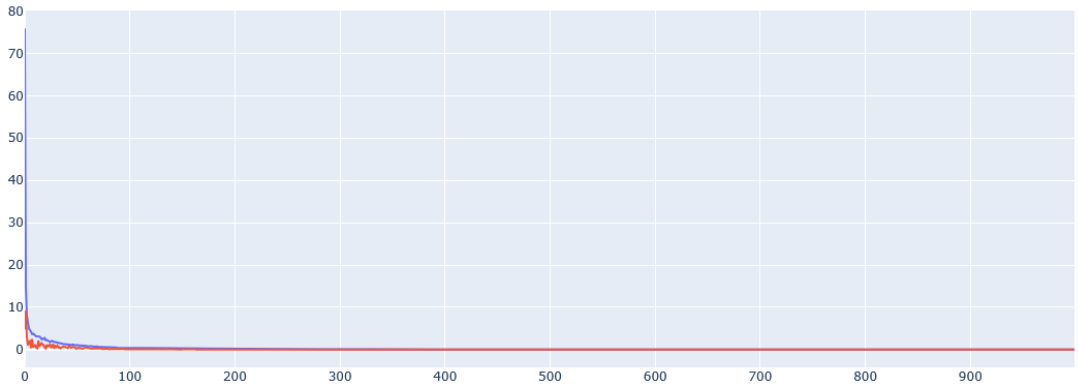
Até aproximadamente 700 épocas, a curva de treino continua diminuindo de forma gradual. A curva de teste também decresce suavemente e permanece abaixo da curva de treino — o que é um excelente sinal de boa generalização. A partir daí, as curvas se estabilizam com valores baixos e próximos de zero. Isso mostra que o modelo atingiu convergência.



RMSprop

No gráfico do modelo RMSprop, ambas as curvas, de treino (azul) e de teste (vermelha), caem drasticamente logo no início do treinamento (aproximadamente nas primeiras 50-100 épocas) e depois se estabilizam. Isso mostra que o otimizador RMSprop foi eficiente em encontrar uma boa região de mínimos rapidamente.

O ”problema” é que essa eficiência pode levar a uma convergência prematura. O otimizador ajusta os parâmetros do modelo de forma tão rápida que o processo se estabiliza na primeira solução de mínimo local que encontra. Com isso, ele deixa de explorar outras configurações de parâmetros que poderiam resultar em um erro final inferior, que é a solução que o SGD, por ser mais lento e exploratório, conseguiu alcançar.

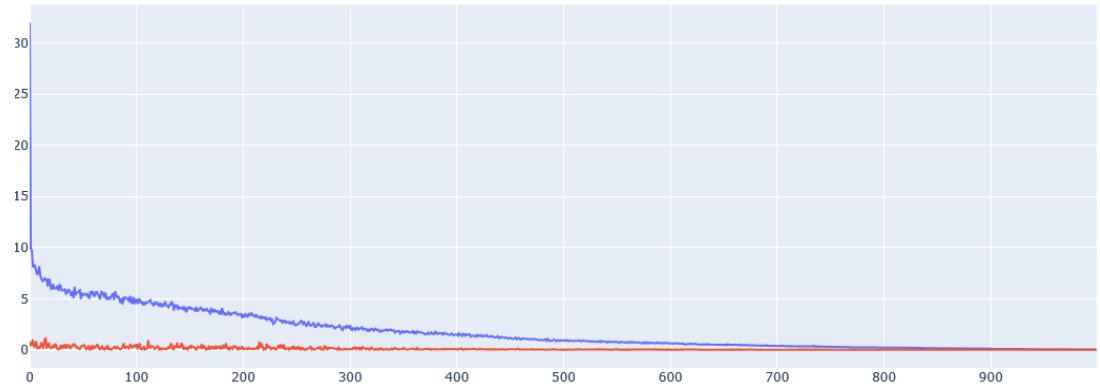


SGD

O modelo que utilizou o otimizador SGD, embora apresente uma curva de aprendizado visivelmente mais ruidosa, demonstrou um desempenho final superior. A perda de treinamento (linha azul) diminui de forma consistente, provando que o modelo aprendeu efetivamente com os dados. De forma crucial, a perda no conjunto de teste (linha vermelha) se manteve extremamente baixa e estável, um sinal claro de excelente generalização e da ausência de overfitting. A aparente instabilidade na curva de treino não é um defeito, mas sim uma característica intrínseca do SGD, que atualiza os parâmetros do modelo com base em amostras aleatórias de dados, causando essas oscilações.

A vantagem competitiva do SGD, que o levou ao melhor resultado, reside justamente em sua natureza estocástica. Essa característica ”ruidosa” funciona como um mecanismo de exploração, impedindo que o otimizador convirja de forma prematura para o primeiro mínimo local que encontra. Ao ”saltar” pela superfície de perda em vez de seguir o caminho mais direto, o SGD teve a capacidade de investigar uma área mais ampla de soluções potenciais. Como resultado,

ele foi capaz de encontrar e se estabilizar em uma região de mínimo mais ótima, alcançando um erro final mais baixo do que a abordagem mais rápida e direta do RMSprop.



5 Conclusão

5.1 Normalizações dos dados

No desenvolvimento do projeto foram utilizadas 3 tipos de normalização: Min-Max, One hot encoding e logaritmica. Em termos de eficiência, a combinação dessas técnicas de normalização e transformação é altamente eficaz para este tipo de problema. A normalização Min-Max é adequada para redes neurais, pois garante que todas as features numéricas contribuam de forma equitativa para o cálculo do gradiente. A transformação logarítmica da variável alvo é fundamental para lidar com a assimetria dos preços dos imóveis, melhorando a capacidade do modelo de aprender e prever com precisão. O One-Hot Encoding é a escolha padrão e eficiente para variáveis categóricas

5.2 Importância das variáveis

Para verificação da importância de variáveis, vamos utilizar a técnica de Importância por Permutação. A técnica consiste em avaliar o impacto de cada variável na performance do modelo embaralhando (permutando) os seus valores, uma de cada vez, e observando quanto a métrica de erro se deteriora. Se a permutação de uma variável causar uma queda significativa na performance, isso indica que a variável é relevante para o modelo. variáveis que mais afetam o modelo:

- **2ndFlrSF:** Indica a área do primeiro andar em pés quadrados;
- **MSZoningC:** Identifica a classificação geral de zoneamento da venda;
- **LandContourBnk:** Descreve a planicidade da propriedade;
- **GrLivArea:** epresenta a área de estar acima do nível do solo em pés quadrados;
- **1stFlrSF:** Indica a área do primeiro andar em pés quadrados;
- **HalfBath:** efere-se ao número de banheiros;
- **OverallQual:** Avalia a qualidade geral do material e acabamento da casa. .

Analisando o significado das variáveis, é possível perceber forte relevância em características de imóveis que representam localidade e tamanho de casa.