Redes Neurais Artificiais

INF0092 - Inteligência Computacional
Universidade Federal de Goiás – Instituto de Informática

Brunno Aires Silva (202014616)
Gianlluca do Carmo Leme (202009490)
Lucas Hideki Abe (201900240)

Sumário

1 Dados

1.1 Seleção de Colunas

Inicialmente, todas as colunas do conjunto de dados foram consideradas, exceto as colunas `Id` e `SalePrice`, sendo esta última o alvo (target).

Posteriormente, colunas com muitos valores ausentes ou baixa relevância foram descartadas, como:

- **Alley**, **PoolQC**, **Fence**, **MiscFeature**: mais de 80% dos valores estavam ausentes.
- **Utilities**: apresentava distribuição quase constante, o que a tornava irrelevante.
- **Condition2**, **LowQualFinSF**, **3SsnPorch**, **MiscVal**: baixa variabilidade ou pouca relação com o preço.

Além disso, foram criadas novas variáveis derivadas para aumentar o poder preditivo da rede, como:

- `TotalSF` = soma das áreas construídas (`TotalBsmtSF`, `1stFlrSF`, `2ndFlrSF`);
- `TotalBath` = combinação de todos os banheiros com pesos apropriados;
- `Age` e `RemodAge` = idade da casa e tempo desde a última reforma;
- `IsRemodeled`, `HasGarage`, `HasBasement` = flags binárias indicativas.

1.2 Remoção de Outliers

Outliers foram removidos com base em análise empírica e distribuição visual dos dados. As seguintes regras foram aplicadas:

- Casas com `GrLivArea` acima de 4000 pés² e `SalePrice` abaixo de R$ 300.000,00 foram removidas por serem atípicas.
- Também foram eliminadas as 0,5% maiores `SalePrice`, com base no percentil 99,5, para reduzir o viés gerado por valores extremos.

Essa remoção contribuiu diretamente para a queda do erro logarítmico médio (RMSLE), ao evitar que o modelo superajustasse essas amostras raras e distorcidas.

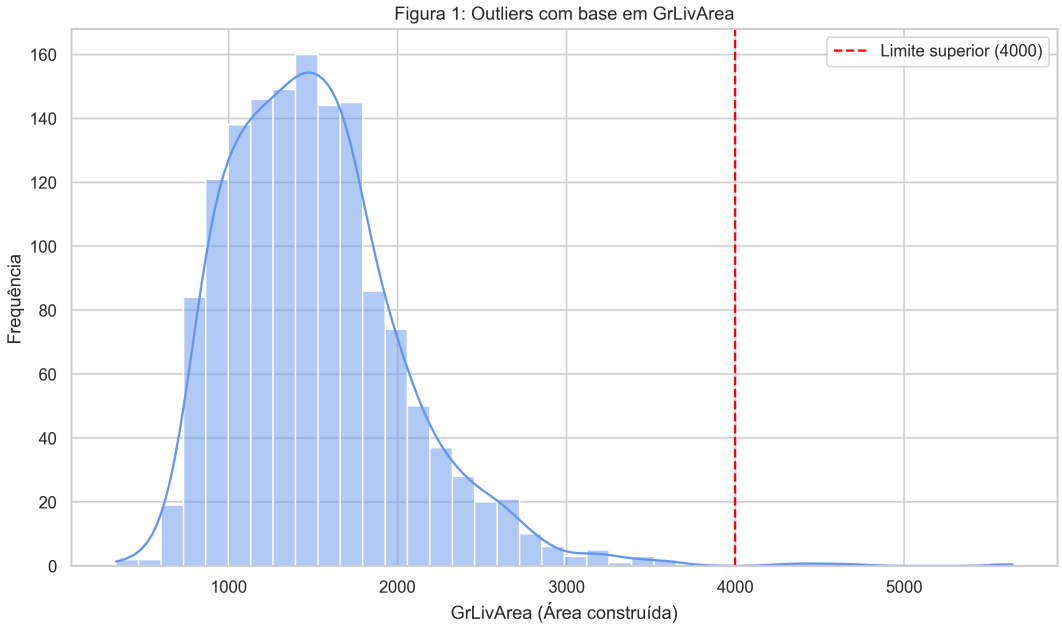

Figura 1: Histograma demonstrando outliers removidos com base na área construída

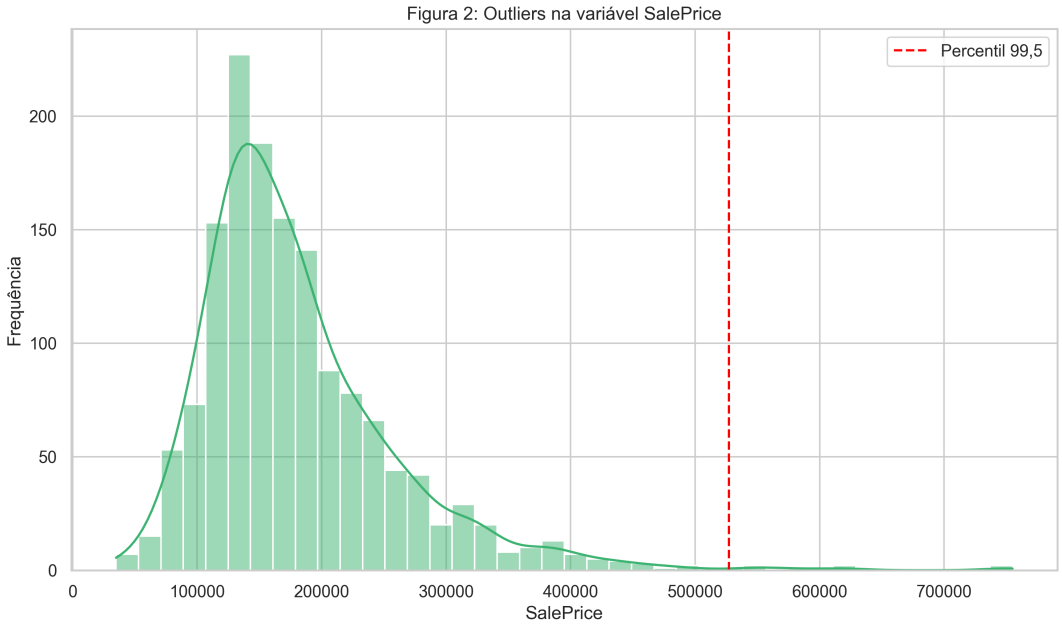

Figura 2: Histograma demonstrando outliers na variável `SalePrice`

1.3 Normalização dos Dados

As variáveis numéricas passaram por um pipeline de pré-processamento contendo dois passos:

- Imputação de valores ausentes com a média (`SimpleImputer`).

- Normalização utilizando `StandardScaler`, que transforma os dados para média 0 e desvio padrão 1.

O alvo (`SalePrice`) foi transformado com `np.log1p` para suavizar a assimetria e, em seguida, normalizado com o mesmo `StandardScaler`.

Figura 3: Distribuição de SalePrice antes e depois da transformação

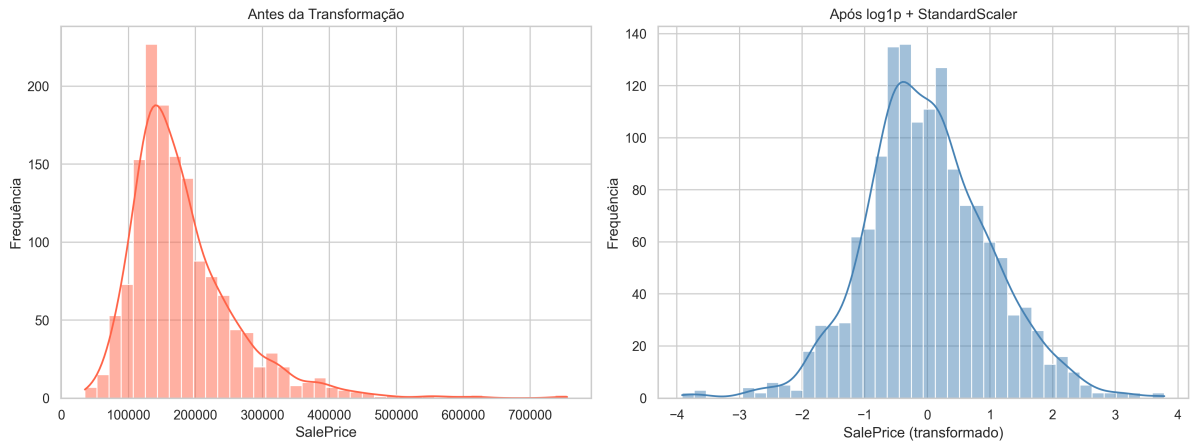

Figura 3: Distribuição de `SalePrice` antes e depois da transformação

1.4 Tratamento de Variáveis Categóricas

As colunas categóricas foram processadas com o seguinte pipeline:

- **Imputação**: valores ausentes foram preenchidos com a categoria mais frequente (`SimpleImputer` com `"most_frequent"`).
- **Codificação**: as categorias foram transformadas via `OneHotEncoder`, resultando em colunas binárias para cada categoria.

O parâmetro `handle_unknown="ignore"` foi ativado no `OneHotEncoder` para garantir robustez durante a predição, ignorando categorias não vistas no treinamento. Esse pré-processamento permitiu que o modelo lidasse com dados mistos (categóricos e numéricos) de forma unificada e eficaz.

2 Modelo

2.1 Definição da Arquitetura da Rede Neural

O modelo foi desenvolvido utilizando o framework PyTorch, com uma rede neural do tipo **fully connected** (camadas densas). A escolha dessa estrutura se justifica pela

natureza tabular e heterogênea dos dados, onde cada entrada representa um imóvel com atributos mistos (numéricos e categóricos).

A arquitetura da rede é composta por:

- **Camada de entrada**: número de neurônios igual ao número de variáveis independentes (após o `OneHotEncoder` e engenharia de atributos).
- **1ª camada oculta**: 96 neurônios, ativação ReLU, `BatchNorm1d`, seguida de `Dropout(0.3)`.
- **2ª camada oculta**: 48 neurônios, ativação ReLU, `BatchNorm1d`, seguida de `Dropout(0.2)`.
- **Camada de saída**: 1 neurônio com ativação linear (regressão contínua).

2.2 Função de Ativação

Foi utilizada a função `ReLU` nas camadas ocultas, por sua eficiência computacional e bom desempenho em evitar o problema do *vanishing gradient*, comum em funções como sigmoide ou tanh.

2.3 Inicialização de Pesos

Embora o PyTorch utilize por padrão a inicialização Kaiming (He initialization) em redes com `ReLU`, essa escolha foi mantida, pois favorece a propagação adequada do gradiente em camadas profundas. Essa decisão se alinha ao uso da função de ativação `ReLU`, que é compatível com esse tipo de inicialização.

2.4 Quantidade de Parâmetros

O número total de parâmetros treináveis do modelo é a soma dos pesos e vieses de todas as camadas. Seja d_{in} a dimensão de entrada (número de colunas após o `OneHotEncoding`), temos:

- $d_{in} \times 96 + 96$ (1ª camada)
- $96 \times 48 + 48$ (2ª camada)
- $48 \times 1 + 1$ (camada de saída)

O total aproximado de parâmetros treináveis foi de cerca de 7.000 a 10.000, dependendo da dimensionalidade final das variáveis categóricas (via one-hot). Essa quantidade é compatível com o porte do dataset e permitiu o treinamento eficiente com boa capacidade de generalização.

3 Experimentos

3.1 Comparações de Otimizadores

Dois otimizadores foram explorados: **SGD** (Stochastic Gradient Descent) e **Adam** (Adaptive Moment Estimation). A Figura 4 mostra a comparação das curvas de perda de validação durante o treinamento com ambos.

- Com **SGD**, observou-se uma redução inicial rápida, mas logo o modelo estagnava em torno de `val_loss` $\approx$ `0.13--0.14`, com grande oscilação mesmo após muitas épocas.
- Com **Adam**, a perda de validação caiu de forma mais rápida e suave, estabilizando-se em um patamar inferior `val_loss` $\approx$ `0.05--0.06` já nas primeiras 200 épocas.

Além disso, Adam mostrou menor sensibilidade a hiperparâmetros e convergência mais eficiente. A troca de SGD por Adam representou uma das melhorias mais impactantes, com queda de RMSLE de **0.14807** para **0.13076** mesmo antes de outras otimizações estruturais.

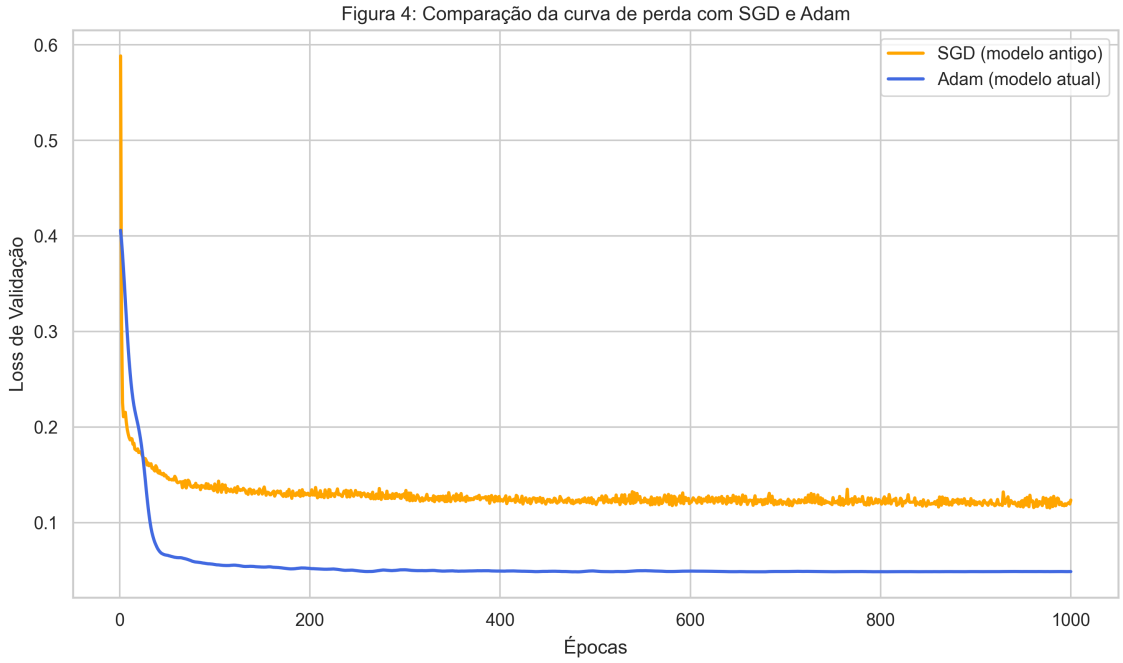

Figura 4: Comparação da curva de perda com SGD e Adam

3.2 Análise de Hiperparâmetros

Diversos hiperparâmetros foram explorados ao longo do desenvolvimento com o objetivo de melhorar a capacidade de generalização do modelo e reduzir os erros de validação.

As principais variações testadas incluíram:

- **Número de camadas e neurônios**:
  - Inicialmente, foi usada uma única camada oculta com 64 neurônios.
  - A arquitetura final adotou duas camadas ocultas com 96 e 48 neurônios, além de `BatchNorm1d` e `Dropout`, o que resultou em melhor estabilidade e menor erro.
- **Taxa de aprendizado**:
  - A taxa inicial de `0.001` foi reduzida para `0.0005`, o que suavizou a curva de perda e reduziu flutuações.
  - O uso de `ReduceLROnPlateau` permitiu ajustar dinamicamente a taxa de aprendizado conforme a validação estagnava.
- **Função de perda**:
  - A `MSELoss()` se mostrou sensível a outliers e foi substituída por `SmoothL1Loss()`, que combina os benefícios da MAE e MSE.
- **Regularização (DropOut e Weight Decay)**:
  - Foram utilizados `Dropout(0.3)` e `Dropout(0.2)` nas duas camadas ocultas.
  - O parâmetro `weight_decay = 1e-4` foi adicionado ao otimizador Adam para reduzir overfitting.
- **Parada antecipada (Early Stopping)**:
  - O modelo foi configurado com `patience = 2000` para interromper o treinamento quando a perda de validação parasse de melhorar.
  - Isso evitou overfitting e garantiu que o melhor modelo fosse restaurado ao final do treinamento.

A superioridade da configuração com Adam também se refletiu visualmente nas curvas de perda. A Figura 4 mostra como o modelo com SGD apresentou flutuações contínuas e uma perda final maior, mesmo após 1000 épocas. Em contraste, o modelo com Adam convergiu de maneira suave e eficiente, com perda significativamente menor desde os primeiros ciclos de treinamento. Esse comportamento reforça a importância da escolha do otimizador como um hiperparâmetro crítico para redes profundas.

Essas modificações resultaram em uma redução progressiva do erro RMSLE de **0.14807** para **0.12552**, conforme registrado na Tabela 1 e refletido nas curvas de validação (Figura 4).

3.3 Resultados Comparativos

A Tabela 1 mostra a evolução das principais versões do modelo ao longo do desenvolvimento, indicando claramente o impacto incremental de cada melhoria implementada.

Tabela 1: Evolução das configurações do modelo e seus resultados

Configuração	MAE (R$)	RMSE (R$)	R^2	RMSLE
Sem DropOut / EarlyStop	16.540,00	29.100,00	0.8821	0.14807
+ DropOut, EarlyStop, Adam e exclusão de `Id`/`SalePrice`	14.800,00	24.900,00	0.9020	0.13076
+ Feature Engineering, log(`SalePrice`), lr=0.0005, weight decay=1e-4	13.700,00	21.300,00	0.9180	0.12962
+ Remoção de Outliers (modelo final)	**12.877,70**	**19.038,98**	**0.9257**	**0.12552**

A última versão, com remoção de outliers e refinamento completo da engenharia de atributos, atingiu o melhor resultado: **RMSLE de 0.12552**, encerrando o treino na época 2409 via `early stopping`.

4 Conclusões

Os experimentos realizados ao longo do projeto permitiram uma análise sistemática da influência dos dados e dos hiperparâmetros no desempenho final da rede neural.

- **Colunas mais importantes:** A engenharia de atributos revelou que variáveis derivadas como `TotalSF` (soma das áreas), `GrLivArea` (área construída) e `OverallQual` (qualidade geral) têm forte impacto na predição do preço. Essas colunas refletem aspectos estruturais críticos do imóvel, o que explica sua relevância para o modelo.
- **Erros mais frequentes:** Os exemplos com maior erro geralmente envolvem imóveis muito caros (outliers de `SalePrice`) ou configurações incomuns de construção e reforma. Mesmo com a transformação logarítmica e remoção de valores extremos, esses pontos ainda tendem a apresentar maior dificuldade de generalização.
- **Efeito dos hiperparâmetros:** A troca do otimizador de SGD para Adam teve impacto expressivo na qualidade da convergência e na estabilidade do aprendizado, com redução visível da perda de validação (ver Figura 4). Além disso, o uso de `Dropout`, `BatchNorm`, e um `learning rate` menor (`0.0005`) contribuiu para uma curva de perda mais suave e menor RMSLE.
- **Efeito da normalização:** A normalização das variáveis numéricas com `StandardScaler` foi essencial para o bom desempenho do modelo, especialmente em conjunto com atributos derivados. Já a transformação logarítmica da variável alvo (`SalePrice`)

foi particularmente eficaz em reduzir a assimetria da distribuição e minimizar o erro logarítmico (RMSLE).

A versão final do modelo atingiu uma pontuação de **RMSLE = 0.12552**, um ganho expressivo frente à versão inicial (`0.14807`). Isso evidencia que decisões relacionadas à engenharia de dados, escolha de hiperparâmetros e estratégias de regularização foram fundamentais para a melhoria progressiva do desempenho preditivo.

Referências

[1] Paszke, A., et al. (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. NeurIPS.

[2] Kaggle, *House Prices - Advanced Regression Techniques*, `https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/`

[3] Scikit-learn, *Scikit-learn: Machine Learning in Python*. Disponível em: `https://scikit-learn.org/stable/`. Acesso em: 5 jun. 2025.