Feedforward Neural Networks for Tabular Data: A Case Study on NYC Airbnb Price Classification

Abdul Hafeez [1]

Abstract

This report implements and analyzes a two-hidden-layer feedforward neural network for multi-class price classification of Airbnb listings in New York City. We develop the architecture from scratch using NumPy, demonstrating that ReLU activation achieves superior convergence compared to Sigmoid by mitigating the vanishing gradient problem. Using a derived gradient-based feature attribution method, we identify `amenity_score` and `minimum_nights` as dominant predictors, validating patterns observed during exploratory data analysis. However, evaluation on a clean test set reveals a severe generalization gap, which we trace to a distribution shift caused by the model's over-reliance on informative missingness flags during training. Finally, we derive the shared-bias gradient for a two-layer network and discuss its convergence implications. The complete implementation is available at: `https://github.com/hafeez381/Deep-Learning/tree/main/assignment_1`

1. Feedforward Neural Networks using Tabular Data

In this section, we explore the fundamentals of feedforward neural networks through a multi-class classification task on tabular data. Our objective is to predict the price category of Airbnb listings in New York City, where each listing is characterized by six features (independent variables), including spatial location, property characteristics, and host availability. The target variable is `price_class` $y \in \{0, 1, 2, 3\}$, corresponding to *Budget*, *Moderate*, *Premium*, and *Luxury* tiers, respectively.

1.1. Exploratory Data Analysis

1.1.1. Dataset Structure

The dataset is partitioned into a training set $\mathcal{D}_{\text{train}}$ ($N_{\text{train}} = 41,348$ samples) and a held-out test set $\mathcal{D}_{\text{test}}$ ($N_{\text{test}} = 7,297$ samples), representing an approximate 85%-15% split. The feature space $\mathcal{X}$ is characterized by a mixture of categorical and numerical variables. A detailed description of the independent variables can be found in Table 1, and a statistical summary of $\mathcal{D}_{\text{train}}$ is presented in Table 2.

Table 1. **Description of the Independent Variables.**

Feature	Description
`neighbourhood_group`	5 NYC boroughs: *Manhattan, Brooklyn, Bronx, Queens, Staten Island*
`room_type`	Accommodation style: *Private room, Entire home/apt, Shared room*
`minimum_nights`	Minimum nights required for booking
`number_of_reviews`	Total count of reviews received
`availability_365`	Number of available days in the next year (0–365)
`amenity_score`	Comparative listing quality/amenity score

Table 2. **Statistical profile of the full training set ($\mathcal{D}_{\text{train}}$).** Numerical features are characterized by central tendency (Mean, Median) and dispersion (Std, Range).

Feature	Type	Mean ± Std	Median	Range
`neighbourhood_group`	Categorical	—	—	5 Groups
`room_type`	Categorical	—	—	3 Types
`minimum_nights`	Numeric	7.0 ± 19.7	3.0	$[1, 1000]$
`number_of_reviews`	Numeric	23.6 ± 44.5	5.0	$[0, 607]$
`availability_365`	Numeric	111.9 ± 131.3	44.0	$[0, 365]$
`amenity_score`	Numeric	52.0 ± 19.5	51.4	$[10, 99]$

1. Univariate Analysis of Numeric Predictors

We assess the distributional properties of the continuous features we employ Quantile-Quantile (Q-Q) plots to visually compare the empirical feature quantiles against a theoretical standard normal distribution, a precondition for optimal and efficient gradient descent convergence.

As illustrated in Figure 1, the numeric features have distinct distributions:

- Both **`minimum_nights`** and **`number_of_reviews`** are strongly right-skewed. Their Q–Q plots deviate upward from the diagonal in

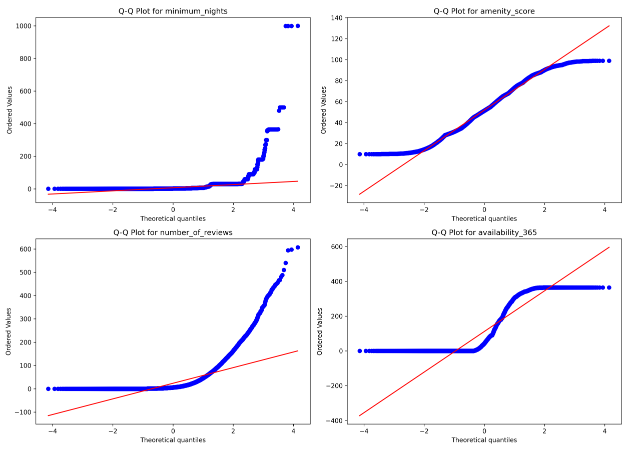


Figure 1. **Normal Q-Q Plots of Numeric Predictors.** The deviations from the 45-degree reference line indicate non-Gaussian behaviors.

the upper quantiles, indicating heavy tails. This aligns with the descriptive statistics in Table 2, where the means significantly exceed the medians (e.g., reviews: $\mu = 23.6$ vs. median $= 5.0$). This suggests that simple min-max scaling may compress the majority of data into a narrow range so a log-transformation scaling is required.

- The **`amenity_score`** closely follows the Q–Q reference line, and its mean and median are nearly equal ($\mu \approx$ Median ≈ 52). This indicates an approximately symmetric distribution, making standardization sufficient.

- The **`availability_365`** variable shows clear deviations at its bounds (0 and 365). This suggests clustering near these limits, consistent with listings that are either rarely available or available year-round.

2. *Univariate Analysis of Categorical Predictors*

The categorical feature space is low-dimensional, consisting of `neighbourhood_group` ($K = 5$) and `room_type` ($K = 3$). We analyze the frequency distribution to identify potential class imbalance, which can bias the network towards majority classes.

- **`neighbourhood_group`:** The spatial distribution is heavily skewed. *Manhattan* ($N = 17,905$; 44.2%) and *Brooklyn* ($N = 16,692$; 41.2%) cumulatively account for over 85% of the observed data. Conversely, *Staten Island* is a severe minority class ($N = 309$; 0.8%), posing a risk of underfitting for this specific region due to data sparsity.

- **`room_type`:** The data exhibits a bimodal dominance between *Entire home/apt* ($N = 21,099$; 51.8%) and *Private room* ($N = 18,663$; 45.8%). The *Shared room* category constitutes a marginal minority ($N = 975$; 2.4%).

Table 3. **Missingness Statistics and Mechanism Analysis.** Features with $p < 0.05$ show a dependency on the target variable (MNAR).

Feature	Count	%	χ^2 (dof=3)	p-value	Mech.
`amenity_score`	916	2.22	42.90	< 0.001	MNAR
`num_reviews`	1123	2.72	14.87	0.002	MNAR
`min_nights`	1322	3.20	11.74	0.008	MNAR
`availability`	595	1.44	6.62	0.085	MCAR
`neighbourhood`	839	2.03	1.10	0.777	MCAR
`room_type`	611	1.48	0.68	0.878	MCAR

1.1.2. Handling Missing Values

We identified missing values across all six predictors in the training set $\mathcal{D}_{\text{train}}$, with rates ranging from 1.44% to 3.20%. To determine the appropriate imputation strategy, we analyzed the missingness mechanism for each feature using statistical hypothesis testing (Little & Rubin, 2002).

For each feature, we constructed a binary indicator (1 if missing, 0 if present) and performed a Chi-Square (χ^2) Test of Independence against the target variable `price_class`. The hypothesis test is defined as:

$$H_0 : P(M_j|Y) = P(M_j) \quad \text{(Independence / MCAR)}$$
$$H_a : P(M_j|Y) \neq P(M_j) \quad \text{(Dependence / MNAR)}$$

The null hypothesis (H_0) states that the missingness of a feature is independent of the price (Missing Completely at Random). We rejected the null hypothesis at significance level $\alpha = 0.05$.

Analyzing the `amenity_score`, the test yielded a statistic of $\chi^2 \approx 42.90$ with 3 degrees of freedom. With a p-value of < 0.001, which is significantly less than our $\alpha = 0.05$ threshold, we reject the null hypothesis and conclude that the absence of an amenity score is statistically dependent on the price class. This indicates the data is Missing Not At Random (MNAR); for instance, cheaper listings may be less likely to report scores.

Similarly, we rejected H_0 for `minimum_nights` ($\chi^2 \approx 11.74, p \approx 0.008$) and `number_of_reviews` ($\chi^2 \approx 14.87, p \approx 0.002$). Conversely, for `availability_365`, we found a p-value of 0.085 ($\chi^2 \approx 6.62$). Since this exceeds 0.05, we fail to reject the null hypothesis, suggesting that missing availability data is random noise (MCAR). The categorical variables `neighbourhood_group` ($p \approx 0.777$) and `room_type` ($p \approx 0.878$) also exhibited random missingness.

1.1.3. Imputation Implementation

To prevent data leakage, we partitioned $\mathcal{D}_{\text{train}}$ into local training (80%) and validation (20%) sets using stratified sampling prior to imputation. All imputation statistics (medians) were derived strictly from the local training split. Based on the diagnostic results, we applied the following transformations:

- **Numerical Features (MNAR):** For `minimum_nights`, `amenity_score`, and `reviews`, we applied Median Imputation to handle the skewness. Additionally, we added binary flags (e.g., `amenity_is_missing`) to the input features. This allows the neural network to explicitly learn the relationship between the missingness and the price class identified by the Chi-Square test.
- **Numerical Features (MCAR):** For `availability_365`, we applied Median Imputation. Since the missingness is random and carries no information about the price, we did **not** add a binary flag, avoiding unnecessary complexity.
- **Categorical Features:** For `neighbourhood_group` and `room_type`, missing entries were encoded as a distinct `'Unknown'` category to preserve the distribution of known classes.

1.1.4. Class Distribution of Target Variable

We analyze the distribution of the target variable y (`price_class`) to establish the baseline difficulty of the classification task. As illustrated in Figure 2, the dataset exhibits significant class imbalance.

The class proportions are unevenly distributed:

- **Class 1 (Moderate):** This is the dominant majority class, accounting for 56.3% ($N = 23,287$) of all observations.
- **Class 2 (Premium):** Represents 23.8% ($N = 9,844$) of the data.
- **Class 0 (Budget):** Represents 13.5% ($N = 5,567$).
- **Class 3 (Luxury):** This is the severe minority class, comprising only 6.4% ($N = 2,650$) of the full training set.

1.1.5. Normalization

To facilitate stable gradient convergence and prevent features with large magnitude (e.g., `availability_365`) from dominating the loss landscape, we implemented a two-stage normalization pipeline.

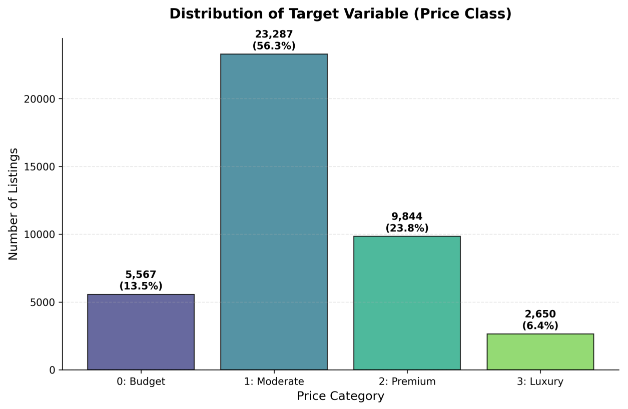


Figure 2. **Distribution of Price Categories.** The dataset is heavily skewed towards the 'Moderate' class (56.3%), while 'Luxury' listings are rare (6.4%).

1. Log-Transformation of Skewed Features:
As visualized in Figure 3, the EDA (Figure 1) revealed significant right-skewness in `minimum_nights` ($\gamma \approx 20.91$) and `number_of_reviews` ($\gamma \approx 3.61$). Such heavy-tailed distributions can cause gradients to explode or result in suboptimal local minima. To mitigate this, we applied a logarithmic transformation element-wise to these specific feature entries x:

$$x' = \ln(1 + x)$$

This operation successfully compressed the heavy tails (ranges $[1, 1000]$ and $[0, 607]$) and stabilized the distributions:

- `minimum_nights`: Skewness reduced from $20.91 \to 1.48$.
- `number_of_reviews`: Skewness reduced from $3.61 \to 0.34$.

2. Global Standardization:
Subsequently, we applied Standard Scaling (Z-score normalization) to the vector of numerical predictors $\mathbf{x} \in \mathbb{R}^4$ (including the log-transformed variables). The transformation is defined as:

$$\mathbf{z} = \mathbf{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})$$

where $\boldsymbol{\mu} \in \mathbb{R}^4$ is the mean vector and $\mathbf{\Sigma} = \text{diag}(\boldsymbol{\sigma})$ is the diagonal matrix of standard deviations, computed exclusively on $\mathcal{D}_{\text{train}}$. This ensures that the input features are centered at $\mathbf{0}$ with identity covariance $\mathbf{I}$.

1.1.6. Feature Encoding

To transform the nominal categorical variables (`neighbourhood_group` and `room_type`) into a

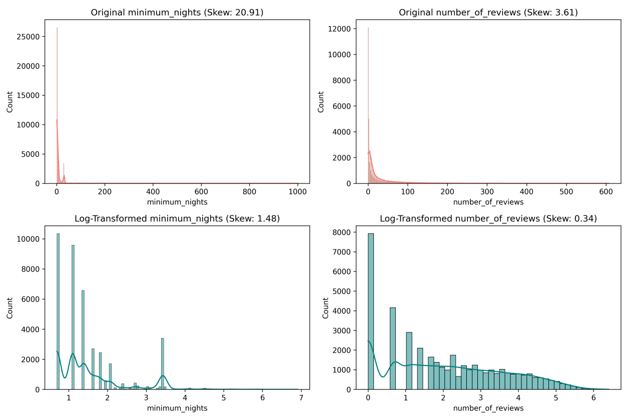


Figure 3. **Impact of Log-Transformation on Feature Distribution.** The top row shows the original heavy-tailed distributions. The bottom row demonstrates how the transformation $x' = \ln(1 + x)$ normalizes the spread, significantly reducing skewness (γ) and outlier leverage.

representation suitable for gradient-based optimization, we employed One-Hot Encoding (OHE).

We prioritized OHE over Ordinal (Label) Encoding because the spatial and property attributes lack an intrinsic hierarchical order (e.g., *Manhattan* is not numerically "greater" than *Bronx*). In a neural network, an integer mapping could erroneously imply that Brooklyn $>$ Manhattan, biasing the gradient updates. By mapping each category to an orthogonal binary vector, we ensure that the distance between any two distinct categories remains equidistant in the input space. Given the low total cardinality ($5 + 3 = 8$ unique categories), the increase in input dimensionality is negligible and does not introduce the curse of dimensionality or sparsity issues often associated with OHE on high-cardinality data.

The encoding process treats the imputed `'Unknown'` category as a valid distinct class. Consequently, the feature space expansion is as follows:

- `neighbourhood_group`: Mapped to 6 binary columns (5 boroughs + 1 Unknown).
- `room_type`: Mapped to 4 binary columns (3 types + 1 Unknown).

Combined with the 4 continuous numerical features and the 3 informative missingness flags, the final input dimensionality for the neural network is $D_{in} = 17$. Furthermore, we configured the encoder to ignore unseen categories during inference (`handle_unknown='ignore'`) by assigning an all-zero vector. This ensures the pipeline remains stable if the test set contains categories not observed during training, preventing runtime failures.

1.1.7. Bivariate Analysis of Features and Target Variable

We examined the conditional distributions of predictors across the target classes ($y \in \{0, 1, 2, 3\}$) to identify distinguishing patterns.

1. Numerical Features: As shown in Figure 4, the `amenity_score` exhibits a strong positive correlation with price; higher tiers (*Premium*, *Luxury*) display significantly higher median scores and lower variance. Conversely, `number_of_reviews` shows an inverse trend, where *Moderate* listings accumulate higher review volumes, likely due to increased turnover and affordability. `minimum_nights` shows significant overlap across classes. `availability_365` displays a higher median for the Premium and Luxury classes, suggesting that professional hosts (who maintain year-round availability) are more prevalent in higher price brackets.

2. Categorical Features: `room_type` acts as a strong discriminator: the *Luxury* class is composed almost exclusively of 'Entire home/apt' listings, while 'Shared rooms' are confined to the *Budget* tier. `neighbourhood_group` reinforces this hierarchy, with *Manhattan* hosting the highest proportion of Premium (Class 2) and Luxury (Class 3) listings, whereas *Bronx* and *Queens* are dominated by Budget and Moderate options.

1.1.8. Pairwise Correlation between Numerical Predictors

To identify potential multicollinearity, which could inflate the variance of the model's weight estimates, we computed the Pearson correlation coefficient matrix for the transformed numerical features in $\mathcal{D}_{\text{train}}$. As illustrated in Figure 6, the features exhibit a high degree of orthogonality.

The strongest observed relationship is a weak inverse correlation between `minimum_nights` and `number_of_reviews` ($\rho \approx -0.22$). This aligns with well intuition: listings requiring long-term stays (higher value for `minimum_nights`) generate fewer bookings per year, thereby accumulating reviews at a slower rate than short-term rentals. Consequently, we retain all numerical features, as they appear to contribute unique information to the classification task without introducing redundant signals.

1.1.9. Hypothesized Predictor Influence Based on EDA

Based on the visual and statistical evidence presented in Figure 4 and Figure 5, we hypothesize the following hierarchy of feature importance:

1. **Primary Discriminators:** `room_type` and

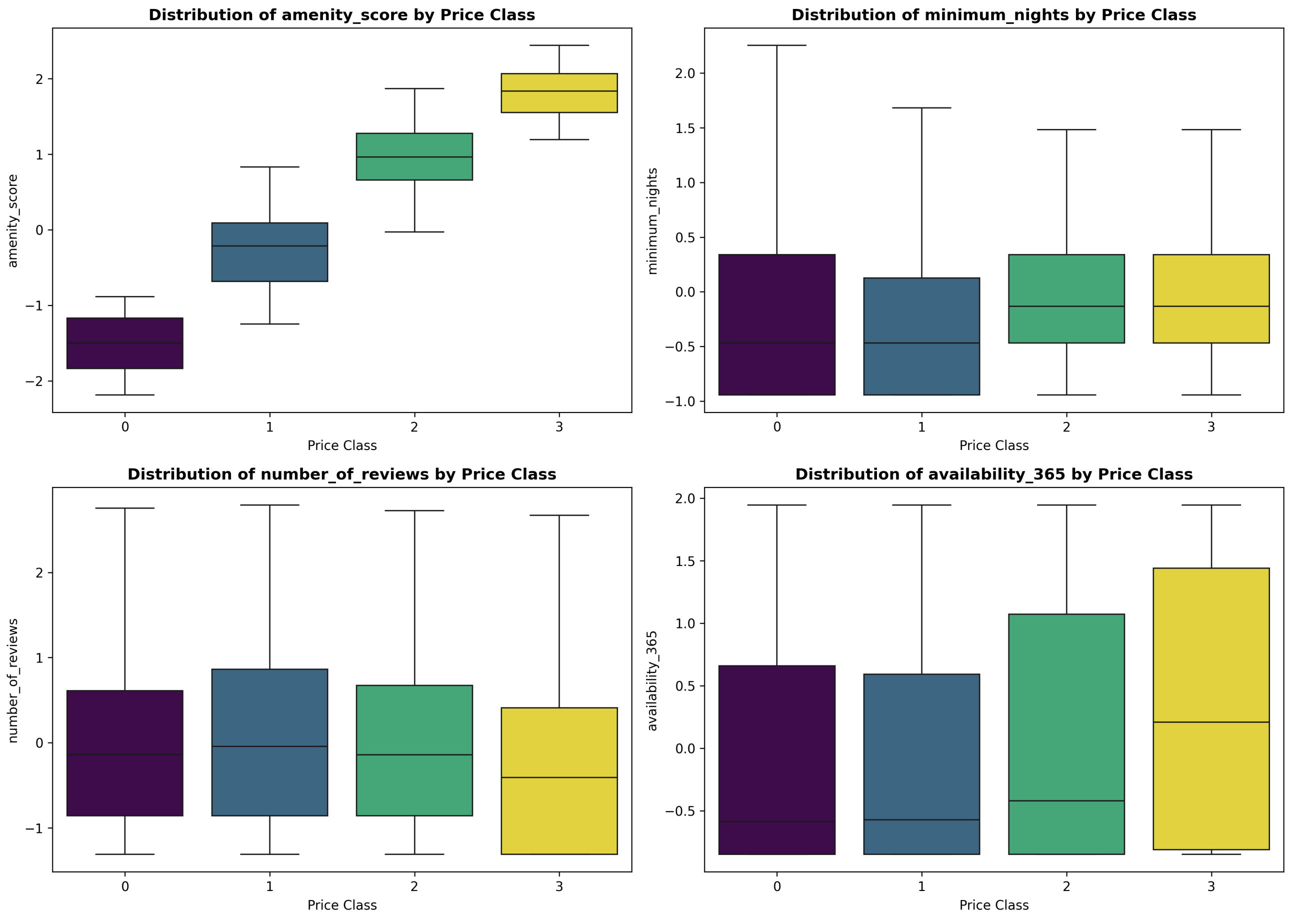


Figure 4. **Numeric Bivariate Analysis.** Boxplots reveal that `amenity_score` effectively separates price tiers, while reviews are inversely correlated. Outliers in `minimum_nights` are prevalent across all classes.

`amenity_score` appear to be the most influential predictors. The stacked bar charts reveal that the *Luxury* class is almost structurally defined by the 'Entire home/apt' category, offering near-perfect separability from 'Shared rooms'. Similarly, the monotonic increase in median `amenity_score` across price tiers (from $\approx$ 40 for Budget to $\approx$ 70 for Luxury) observed in Figure 4 indicates a strong linear relationship with the target. Its minimal correlation with other numerical features ($\rho < 0.2$) further suggests it provides unique, non-redundant information about listing quality.

2. **Secondary Discriminators:** `neighbourhood_group` provides strong spatial priors (e.g., $P(\text{Luxury}|\text{Manhattan}) \gg P(\text{Luxury}|\text{Bronx})$). `number_of_reviews` acts as a meaningful inverse signal for the *Budget* class, likely capturing the higher turnover rate of affordable listings.

3. **Weak Predictors:** `minimum_nights` exhibits high variance and significant overlap across all target classes, suggesting it contributes minimal marginal information for linear separability on its own.

1.1.10. Assessment of Potential Data Leakage or Dominant Predictors

We detected no evidence of data leakage, as all features are known a priori to the listing price. While `room_type` is a strong discriminator, this represents a valid structural prior reflecting real-world price difference rather than an error in the data. Furthermore, the explicit modeling of informative missingness (MNAR) in `amenity_score` and the low correlation between features ($|\rho| < 0.22$) confirm that the data is statistically valid for predictive modeling.

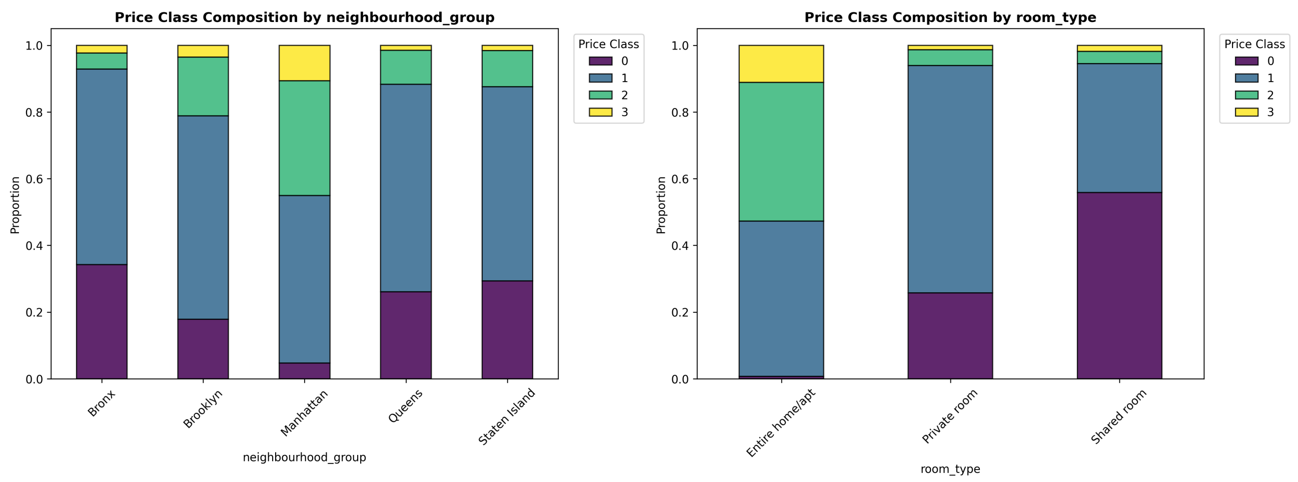
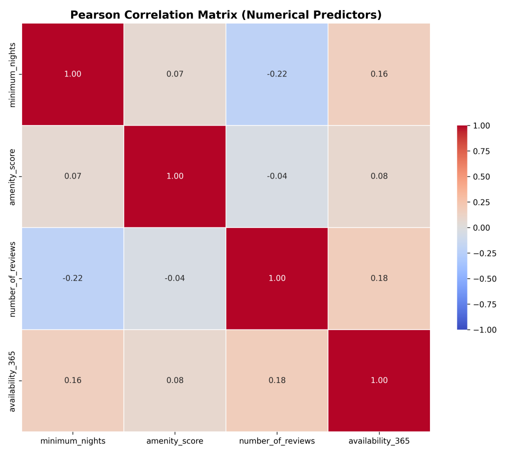


Figure 5. **Categorical Bivariate Analysis.** Stacked bar charts demonstrate that `room_type` and `neighbourhood` strongly determine the class priors. Manhattan and Entire Homes skew towards higher price points.



	minimum_nights	amenity_score	number_of_reviews	availability_365
minimum_nights	1.00	0.07	-0.22	0.16
amenity_score	0.07	1.00	-0.04	0.08
number_of_reviews	-0.22	-0.04	1.00	0.18
availability_365	0.16	0.08	0.18	1.00

Figure 6. **Pearson Correlation Matrix.** The low magnitude of all coefficients ($|\rho| < 0.22$) indicates that the numerical predictors are largely independent, minimizing the risk of redundancy.

1.2. Two-Layer Perceptron Implemented from Scratch

1.2.1. Implementation Details

We implemented a fully connected feedforward neural network using only NumPy to ensure transparency in the computational graph. The architecture consists of an input layer ($M^{(0)} = 17$), two hidden layers ($M^{(1)} = 64, M^{(2)} = 32$), and an output layer ($M^{(3)} = 4$).

1. Forward Propagation: The network processes the input batch $\mathbf{X}^{(0)} \in \mathbb{R}^{17\times N}$ through a sequence of affine transformations and non-linear activations. For each layer $l \in \{1, 2\}$, the pre-activation $\mathbf{A}^{(l)}$ and activation $\mathbf{X}^{(l)}$ are computed as:

$$\mathbf{A}^{(l)} = \mathbf{W}^{(l)}\mathbf{X}^{(l-1)} + \mathbf{b}^{(l)}$$
$$\mathbf{X}^{(l)} = g(\mathbf{A}^{(l)})$$

where $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weight matrix and bias vector, and $g(\cdot)$ is the activation function. The output layer applies the Softmax function to generate a probability distribution $\hat{\mathbf{Y}}$ over the $K = 4$ classes.

2. Loss Computation: We minimize the Categorical Cross-Entropy loss, defined for the batch as:

$$L(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_k^{(i)} \log(\hat{y}_k^{(i)})$$

3. Backward Propagation: Gradients are computed via the chain rule using the "Delta" notation. The error term $\delta^{(l)}$ for the hidden layers is propagated recursively:

$$\delta^{(l)} = ((\mathbf{W}^{(l+1)})^T\delta^{(l+1)}) \odot g'(\mathbf{A}^{(l)})$$

The gradients for weights and biases are then derived as $\nabla_{\mathbf{W}^{(l)}} L = \frac{1}{N}\delta^{(l)}(\mathbf{X}^{(l-1)})^T$ and $\nabla_{\mathbf{b}^{(l)}} L = \frac{1}{N}\sum \delta^{(l)}$.

1.2.2. Activation Functions & Gradient Properties

We evaluated two distinct activation functions:

- **Sigmoid ($\sigma(z) = \frac{1}{1+e^{-z}}$):** This function maps inputs to $(0, 1)$. However, it suffers from the *vanishing gradient problem*. When inputs $|z|$ are large, the derivative $\sigma'(z) = \sigma(z)(1 - \sigma(z))$ approaches zero, causing the error signal $\delta^{(l)}$ to decay exponentially during backpropagation.

- **ReLU ($g(z) = \max(0, z)$):** The Rectified Linear Unit is non-saturating for positive inputs, maintaining a constant gradient of 1. This facilitates efficient gradient flow deep into the network and induces sparsity in the hidden representations.

1.2.3. Training Dynamics

We trained both models using Batch Gradient Descent for 300 iterations. The ReLU network was trained with a learning rate $\eta = 0.01$, while the Sigmoid network required a higher rate $\eta = 0.1$ to overcome gradient suppression.

The ReLU model demonstrated faster convergence, achieving a final training accuracy of **76.18%** and validation accuracy of **76.14%**. In contrast, the Sigmoid model exhibited prolonged stagnation, plateauing at exactly **56.3%** accuracy for the first 250 iterations. This value corresponds precisely to the prevalence of the majority 'Moderate' class observed in the EDA (Figure 2). This indicates that the Sigmoid network initially collapsed into a trivial local minimum, predicting the mode for all inputs (the "lazy" solution), before finally learning non-linear discriminators to reach a final accuracy of **63.08%** (Train) and **62.77%** (Val).

1.2.4. Performance Visualization

The distinct learning trajectories are visualized in Figure 7.

1.3. Gradient Magnitude Comparison Across Layers

1.3.1. Gradient Magnitude Analysis

We monitored the mean absolute gradients $|\nabla_{\mathbf{W}^{(l)}} L|$ for the first ($l = 1$) and second ($l = 2$) hidden layers to assess the gradient propagation through the network.

As shown in Figure 8:

- **Sigmoid Model:** The network exhibits a severe discrepancy between layers. While the second layer gradients (Green) maintain a magnitude of $\approx 10^{-3}$, the first layer gradients (Blue) are nearly zero. This confirms that the gradient "vanishes" as it backpropagates from the output to the input.
- **ReLU Model:** Both layers receive strong gradient magnitudes ($\approx 10^{-2}$ at initialization). Notably, the gradients for Layer 1 and Layer 2 decay in sync, meaning the entire network is learning at the same pace. This is indicative of healthy convergence towards a local minimum.

1.3.2. Observations on Gradient Flow

The experimental results isolate the *vanishing gradient problem* as the primary cause of the Sigmoid network's initial stagnation at the majority-class baseline (56.3%).

By the chain rule, the gradient at layer l depends on the product of downstream derivatives: $\delta^{(l)} \propto \delta^{(l+1)} \cdot g'(\mathbf{z}^{(l)})$. Since the derivative of the Sigmoid function is upper-bounded by $\max(\sigma'(z)) = 0.25$, each layer multiplies the error signal by at most $\frac{1}{4}$. This means that every time the signal passes backward through a layer, it gets cut by at least 75%. Consequently, the weights in the first layer ($\mathbf{W}^{(1)}$) received insufficient updates, preventing the network from learning the low-level representations of features like `amenity_score` and `room_type` required for classification.

In contrast, the ReLU activation, with $g'(z) = 1$ for $z > 0$, allows gradients to pass through unchanged. This preservation of magnitude allowed the gradients to propagate effectively to $\mathbf{W}^{(1)}$, enabling the model to learn non-linear decision boundaries immediately (Iter 20 accuracy: 61.5%) rather than defaulting to the mode.

1.4. Gradient-Based Feature Attribution

1.4.1. Symbolic Derivation of Input Gradients

To quantify the sensitivity of the loss function L with respect to the input features $\vec{x}$, we extend the standard backpropagation procedure.

1. Network Definitions and Recurrence:
Consider the two-layer network where the pre-activation at layer l is given by $\vec{a}^{(l)} = W^{(l)}\vec{x}^{(l-1)} + \vec{b}^{(l)}$ and the activation is $\vec{x}^{(l)} = g(\vec{a}^{(l)})$. We define the *error vector* (or delta) at layer l as the gradient of the loss with respect to the pre-activation as

$$\vec{\delta}^{(l)} \triangleq \frac{\partial L}{\partial \vec{a}^{(l)}}.$$

Standard backpropagation computes these deltas recursively from the output layer ($l = 3$) backwards as

$$\vec{\delta}^{(l-1)} = \left(W^{(l)^T}\vec{\delta}^{(l)}\right) \odot g'(\vec{a}^{(l-1)}).$$

2. Loss Formulation and Output Error Derivation:
We define the objective as the Categorical Cross-Entropy loss L over K classes. The network output $\hat{y}$ (or $\vec{x}^{(3)}$) is obtained by applying the Softmax function to the output pre-activation $\vec{a}^{(3)}$,

$$\hat{y}_k = \text{softmax}(\vec{a}^{(3)})_k = \frac{e^{a_k^{(3)}}}{\sum_{j=1}^{K} e^{a_j^{(3)}}}, \quad L = -\sum_{k=1}^{K} y_k \log(\hat{y}_k),$$

where $\vec{y}$ is the one-hot encoded ground truth vector. To initiate backpropagation, we compute the gradient of the loss with respect to the pre-activation $\vec{a}^{(3)}$, denoted as $\vec{\delta}^{(3)}$. Applying the chain rule, we get

$$\delta_k^{(3)} = \frac{\partial L}{\partial a_k^{(3)}} = \sum_{j=1}^{K} \frac{\partial L}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial a_k^{(3)}}.$$

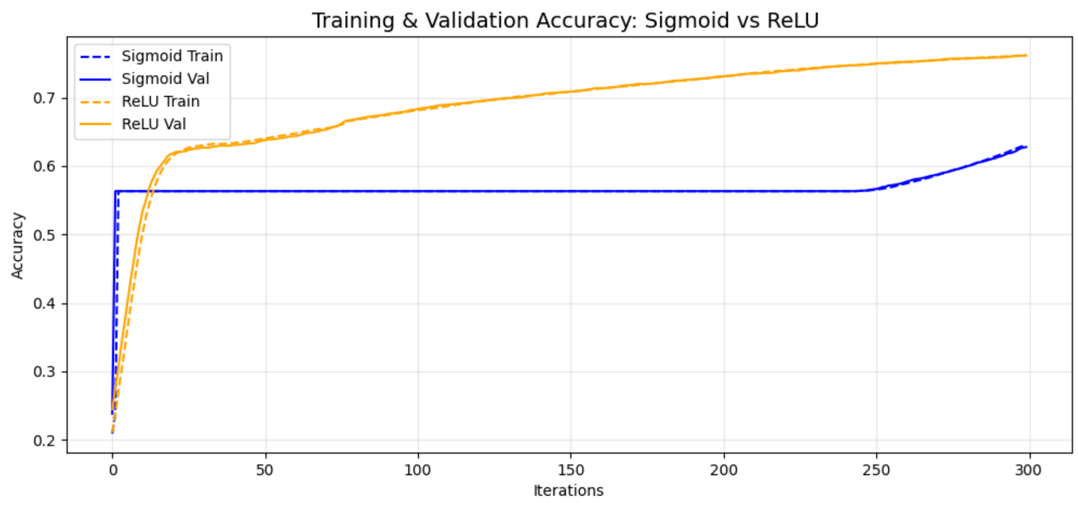


Figure 7. **Training Dynamics: Sigmoid vs. ReLU.** The Sigmoid model (blue) exhibits a distinct plateau at $\approx 56.3\%$ accuracy (the majority class baseline) for the first 250 iterations, indicating gradient stagnation. Conversely, the ReLU model (orange) escapes the null baseline of 56.3% immediately with monotonic convergence.

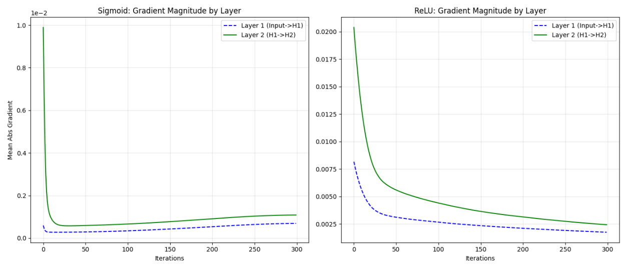


Figure 8. **Gradient Flow Analysis.** (Left) The Sigmoid network suffers from vanishing gradients; the signal reaching Layer 1 (Blue) is negligible compared to Layer 2, causing the 200-iteration plateau. (Right) The ReLU network maintains strong gradient flow across all layers, enabling rapid feature learning.

The partial derivative of the loss is $\frac{\partial L}{\partial \hat{y}_j} = -\frac{y_j}{\hat{y}_j}$. The Jacobian of the Softmax function is $\frac{\partial \hat{y}_j}{\partial a_k^{(3)}} = \hat{y}_j(\mathbb{I}_{jk} - \hat{y}_k)$, where $\mathbb{I}_{jk}$ is the Kronecker delta. Substituting these terms yields

$$\delta_k^{(3)} = \sum_{j=1}^{K} \left(-\frac{y_j}{\hat{y}_j}\right) \hat{y}_j(\mathbb{I}_{jk} - \hat{y}_k)$$

$$= -y_k(1-\hat{y}_k) - \sum_{j \neq k} y_j(-\hat{y}_k) = \hat{y}_k \sum_{j=1}^{K} y_j - y_k$$

Since $\vec{y}$ is one-hot ($\sum y_j = 1$), this simplifies to the vector form

$$\vec{\delta}^{(3)} = \hat{y} - \vec{y}.$$

3. Extending to the Input Layer:
The standard algorithm terminates after computing $\vec{\delta}^{(1)}$ to update $W^{(1)}$. To find $\nabla_{\vec{x}} L$, we treat the input $\vec{x} = \vec{x}^{(0)}$ as the output of a layer 0. The input feeds into the first hidden layer via the affine transformation

$$\vec{a}^{(1)} = W^{(1)}\vec{x} + \vec{b}^{(1)}.$$

Applying the chain rule, the gradient with respect to $\vec{x}$ is

$$\frac{\partial L}{\partial \vec{x}} = \left(\frac{\partial \vec{a}^{(1)}}{\partial \vec{x}}\right)^T \frac{\partial L}{\partial \vec{a}^{(1)}}.$$

The term $\frac{\partial L}{\partial \vec{a}^{(1)}}$ is simply $\vec{\delta}^{(1)}$. The Jacobian of the linear transformation $\frac{\partial \vec{a}^{(1)}}{\partial \vec{x}}$ is the weight matrix $W^{(1)}$. Substituting these terms yields the result

$$\boxed{\frac{\partial L}{\partial \vec{x}} = W^{(1)^T} \vec{\delta}^{(1)}.}$$

4. Full Closed-Form Expression:
By unrolling the recurrence starting from the output error $\vec{\delta}^{(3)} = \hat{y} - \vec{y}$, we obtain the full analytical formula

$$\nabla_{\vec{x}} L = W^{(1)^T} \left[\left(W^{(2)^T} \left(W^{(3)^T} (\hat{y} - \vec{y}) \right) \odot g'(\vec{a}^{(2)}) \right) \odot g'(\vec{a}^{(1)}) \right].$$

The i-th component gives the gradient with respect to the i-th input feature

$$\left(\frac{\partial L}{\partial \vec{x}}\right)_i = \frac{\partial L}{\partial x_i}, \quad i = 1, 2, \ldots, d.$$

1.4.2. Algorithmic Pseudocode

The following algorithm outlines the procedure to analytically compute input gradients for each training sample and aggregate their magnitudes to derive a global feature importance ranking.

Algorithm 1 Feature Importance via Average Gradient Magnitude

```
1:  Input: Dataset $\mathcal{D} = \{(\vec{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$, Network Parameters $\theta = \{W^{(l)}, \vec{b}^{(l)}\}_{l=1}^{3}$
2:  Output: Ranked List of Features based on Importance Score $\vec{I}$
3:  Initialize importance vector $\vec{I} \leftarrow \vec{0} \in \mathbb{R}^M$
4:  for $n = 1$ to $N$ do
5:      // Step 1: Forward Pass
6:      $\vec{a}^{(1)} \leftarrow W^{(1)}\vec{x}^{(n)} + \vec{b}^{(1)}$;   $\vec{x}^{(1)} \leftarrow g(\vec{a}^{(1)})$
7:      $\vec{a}^{(2)} \leftarrow W^{(2)}\vec{x}^{(1)} + \vec{b}^{(2)}$;   $\vec{x}^{(2)} \leftarrow g(\vec{a}^{(2)})$
8:      $\vec{a}^{(3)} \leftarrow W^{(3)}\vec{x}^{(2)} + \vec{b}^{(3)}$;   $\hat{y} \leftarrow \text{softmax}(\vec{a}^{(3)})$
9:      // Step 2: Backward Pass (Standard Backprop)
10:     $\vec{\delta}^{(3)} \leftarrow \hat{y} - \text{one\_hot}(y^{(n)})$
11:     $\vec{\delta}^{(2)} \leftarrow (W^{(3)^T}\vec{\delta}^{(3)}) \odot g'(\vec{a}^{(2)})$
12:     $\vec{\delta}^{(1)} \leftarrow (W^{(2)^T}\vec{\delta}^{(2)}) \odot g'(\vec{a}^{(1)})$
13:     // Step 3: Compute Input Gradient
14:     $\vec{g}^{(n)} \leftarrow W^{(1)^T}\vec{\delta}^{(1)}$   // Compute $\nabla_{\vec{x}} L$
15:     // Step 4: Accumulate Absolute Magnitude
16:     for $j = 1$ to $d$ do
17:         $I_j \leftarrow I_j + |g_j^{(n)}|$   // Sum absolute gradients for feature $j$
18:     end for
19: end for
20: // Step 5: Average and Rank
21: $\vec{I} \leftarrow \frac{1}{N}\vec{I}$   // Compute mean sensitivity per feature
22: Sort feature indices $j \in \{1, \ldots, M\}$ in descending order of $I_j$
23: return Sorted Feature List
```

1.4.3. Justification of Gradient Magnitude as Feature Importance

From an optimization perspective, the gradient quantifies the local sensitivity of the loss function to input perturbations. We use a first-order Taylor expansion around the input $\vec{x}$ as

$$L(\vec{x} + \epsilon\vec{e}_i) \approx L(\vec{x}) + \epsilon \cdot \frac{\partial L}{\partial x_i},$$

where $\vec{e}_i$ is the standard basis vector for feature i. This relationship demonstrates that $\frac{\partial L}{\partial x_i}$ measures the rate of change in the loss given a unit perturbation in x_i, holding all other features constant. Consequently, a large magnitude $\left|\frac{\partial L}{\partial x_i}\right|$ indicates that the model's prediction is highly sensitive to variations in this feature, identifying it as an influential predictor.

1.5. Implementation and Results of Feature Attribution

1.5.1. Implementation Strategy

We implemented a Gradient-Based Feature Attribution method (Sensitivity Analysis) following the formulation in Section 2.8 of (Aggarwal, 2018). Instead of backpropagating the loss (which depends on the ground truth), we compute the gradient of the **winning class score** o_c with respect to the input features $\vec{x}$. This quantity, $|\nabla_{\vec{x}} o_c|$, measures how much the model's confidence in its chosen prediction changes given a perturbation in feature x_i.

We reused the pre-trained **Two-Layer ReLU Perceptron** from Section 1.2 without retraining. To ensure the attribution reflects valid learned patterns rather than noise, we computed gradients exclusively on the subset of the validation set that was correctly classified ($N = 6,297$ samples).

1.5.2. Ranked Feature Importance

Aggregating the mean absolute gradient magnitude across the validation set yielded the following hierarchy of feature influence. The top 5 predictors are presented below:

Table 4. **Top 5 Most Influential Features.** Ranked by Mean Absolute Gradient Magnitude ($|\nabla_{\vec{x}} o_c|$).

Feature	Importance Score
1. `amenity_score`	0.591
2. `minimum_nights`	0.547
3. `room_type_Private room`	0.410
4. `minimum_nights_is_missing`	0.409
5. `neighbourhood_group_Queens`	0.379

1.5.3. Interpretation and Correlation with EDA

The attribution results provide a distinct contrast to the linear assumptions made during the Exploratory Data Analysis (Section 1.1):

- **Validation of Primary Discriminators:** As hypothesized, `amenity_score` and `room_type` are dominant predictors. The high sensitivity of `amenity_score` (0.591) confirms the boxplot separation observed in Figure 4, proving the model relies heavily on listing quality to stratify price tiers.
- **Re-evaluation of 'minimum_nights':** In the EDA, we classified `minimum_nights` as a "Weak Predictor" due to significant distributional overlap across classes. However, the gradient analysis ranks it as the **second most important feature** (0.547). The log-transformation applied in the pipeline likely facilitated this by normalizing the feature's dynamic range.
- **Confirmation of Informative Missingness:** The feature `minimum_nights_is_missing` ranks 4th

(0.409). This empirically validates our Chi-Square test result from the EDA (Table Table 2), which identified the missingness mechanism as MNAR (Missing Not At Random). The model has learned that the *absence* of this data is a strong predictor in itself, justifying the creation of the binary flag.

1.5.4. Algorithm Pseudocode

The procedure for computing the sensitivity of the winning class score is detailed below.

Algorithm 2 Aggarwal's Sensitivity Analysis

```
1: Input: Trained Model $\mathcal{M}$, Validation Data $X_{\text{val}}$
2: Output: Feature Importance Scores $\vec{S} \in \mathbb{R}^d$
3: Initialize $\vec{S} \leftarrow \vec{0}$
4: Filter $X_{\text{correct}} \leftarrow \{\vec{x} \in X_{\text{val}} \mid \text{argmax}(\mathcal{M}(\vec{x})) = y_{\text{true}}\}$
5: for each sample $\vec{x}^{(n)}$ in $X_{\text{correct}}$ do
6:     // 1. Forward Pass
7:     $\vec{o} \leftarrow \mathcal{M}.\text{forward}(\vec{x}^{(n)})$   // Get output logits
8:     $c \leftarrow \text{argmax}(\vec{o})$   // Identify winning class index
9:     // 2. Define Gradient at Output
10:    $\vec{\delta}^{(3)} \leftarrow \vec{0}$
11:    $\delta_c^{(3)} \leftarrow 1.0$   // Set gradient of winning node $o_c$ to 1
12:    // 3. Backpropagate (Sensitivity)
13:    $\vec{\delta}^{(2)} \leftarrow (W^{(3)^T}\vec{\delta}^{(3)}) \odot g'(\vec{a}^{(2)})$
14:    $\vec{\delta}^{(1)} \leftarrow (W^{(2)^T}\vec{\delta}^{(2)}) \odot g'(\vec{a}^{(1)})$
15:    $\vec{g}^{(n)} \leftarrow W^{(1)^T}\vec{\delta}^{(1)}$   // Result: $\nabla_{\vec{x}} o_c$
16:    // 4. Accumulate Magnitude
17:    for $j = 1$ to $d$ do
18:        $S_j \leftarrow S_j + |g_j^{(n)}|$   // Aggregate absolute sensitivity
19:    end for
20: end for
21: $\vec{S} \leftarrow \frac{1}{|X_{\text{correct}}|}\vec{S}$   // Average over samples
22: Return Rank features by sorting $\vec{S}$ descending
```

1.6. Test Evaluation and Generalization Analysis

1.6.1. Test Performance Metrics

The final ReLU network achieved a Test Accuracy of **45.44%**. This is significantly lower than the Training and Validation accuracy of $\approx 76.1\%$, and notably worse than the majority-class baseline of 56.3%.

1.6.2. Performance Gap Analysis

The massive performance drop ($\approx 30\%$) indicates a severe failure to generalize. Since we applied normalization and log-transformations identically to the test set, we suspect that the issue is a fundamental **Distribution Shift** between the training and test environments.

1.6.3. Root Cause Identification

The generalization failure is driven by the model's over-reliance on "Missingness Flags," which acted as shortcuts during training:

1. **Evidence from Feature Attribution Section 1.4:** The Gradient-based Feature Attribution identified `minimum_nights_is_missing` as the **4th most important feature**. The model heavily relied on this binary flag to make decisions.

2. **Evidence from EDA Section 1.1:** As noted in the implementation, the Test set contained **zero missing values**. Consequently, the `_is_missing` features were all-zero vectors in the test set.

1.6.4. Explanation of Training vs. Test Discrepancy

During training, the network learned that "Missing Data = Specific Price Class". It assigned large weights to these flags. In the test set, this gradient vanished entirely ($x = 0$ for all samples). Missing its top predictors, the model was forced to rely on the raw `minimum_nights` values, which we identified in the EDA as having poor separability, leading to confident but wrong predictions.

1.6.5. Mitigation Strategy

To fix this, we should remove the missingness flags from the feature set. While they improve training accuracy, they are not strong features by themselves. Relying only on the imputed values (median) would force the model to learn the actual underlying distribution of the listing rather than the "missing" label, improving generalization to the clean test dataset.

2. Bias Gradients, Parameter Sharing, and Activation Effects

2.0.1. Derivation of Gradient for Shared Bias Parameter

Consider the feedforward network defined by the following forward propagation equations:

$$\begin{aligned}
\vec{a}^{(1)} &= W\vec{x} + \vec{b} && \text{(Hidden Pre-activation)}\\
\vec{h}_1 &= g(\vec{a}^{(1)}) && \text{(Hidden Activation)}\\
\vec{a}^{(2)} &= U\vec{h}_1 + \vec{b} && \text{(Output Pre-activation)}\\
\hat{y} &= g_1(\vec{a}^{(2)}) && \text{(Output Prediction)}\\
L &= \ell(\hat{y}, \vec{y}) && \text{(Loss Function)}
\end{aligned}$$

Since the bias vector $\vec{b}$ appears in both the hidden and output layers, the total gradient $\frac{\partial L}{\partial \vec{b}}$ is the sum of the gradients flowing through two distinct computational paths:

- Path 1: $\vec{b} \to \vec{a}^{(2)} \to \hat{y} \to L$
- Path 2: $\vec{b} \to \vec{a}^{(1)} \to \vec{h}_1 \to \vec{a}^{(2)} \to \hat{y} \to L$

Applying the multivariate chain rule, we express the total derivative as:

$$\frac{\partial L}{\partial \vec{b}} = \underbrace{\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \vec{a}^{(2)}} \frac{\partial \vec{a}^{(2)}}{\partial \vec{b}}}_{\text{Path 1}} + \underbrace{\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \vec{a}^{(2)}} \frac{\partial \vec{a}^{(2)}}{\partial \vec{h}_1} \frac{\partial \vec{h}_1}{\partial \vec{a}^{(1)}} \frac{\partial \vec{a}^{(1)}}{\partial \vec{b}}}_{\text{Path 2}}$$

We first evaluate the gradients common to both paths $\frac{\partial L}{\partial \vec{a}^{(2)}}$. The gradient of the loss with respect to the output pre-activation $\vec{a}^{(2)}$, denoted as $\vec{\delta}^{(2)}$, is derived from the composition of the loss function L and the output activation $\hat{y} = g_1(\vec{a}^{(2)})$. Applying the chain rule, we get

$$\vec{\delta}^{(2)} \triangleq \frac{\partial L}{\partial \vec{a}^{(2)}} = \frac{\partial L}{\partial \hat{y}} \odot g_1'(\vec{a}^{(2)}),$$

where $\odot$ denotes the Hadamard product and g_1' is the element-wise derivative of the output activation function.

From the affine transformations $\vec{a}^{(2)} = U\vec{h}_1 + \vec{b}$ and $\vec{a}^{(1)} = W\vec{x} + \vec{b}$, the partial derivatives are identity matrices of size $m \times m$:

$$\frac{\partial \vec{a}^{(2)}}{\partial \vec{b}} = I_m, \quad \frac{\partial \vec{a}^{(1)}}{\partial \vec{b}} = I_m.$$

Substituting these into the first term of the total gradient equation (Path 1) yields exactly $\vec{\delta}^{(2)}$.

For the second term (Path 2), we evaluate the intermediate derivatives. The gradient flows through the weight matrix U (from $\vec{a}^{(2)} = U\vec{h}_1 + \vec{b}$) and the hidden activation function g (from $\vec{h}_1 = g(\vec{a}^{(1)})$):

$$\frac{\partial \vec{a}^{(2)}}{\partial \vec{h}_1} = U, \quad \frac{\partial \vec{h}_1}{\partial \vec{a}^{(1)}} = \text{diag}(g'(\vec{a}^{(1)}))$$

Substituting these into the chain rule expansion and multiplying by $\frac{\partial L}{\partial \vec{a}^{(2)}}$, the output error $\vec{\delta}^{(2)}$, allows us to define the hidden layer error $\vec{\delta}^{(1)}$:

$$\vec{\delta}^{(1)} \triangleq \left(U^T \vec{\delta}^{(2)}\right) \odot g'(\vec{a}^{(1)}).$$

Thus, the gradient for the second term (Path 2) simplifies yields exactly $\vec{\delta}^{(1)}$. Combining both terms, the total gradient with respect to the shared bias is the sum of the error vectors from all layers where $\vec{b}$ is applied

$$\frac{\partial L}{\partial \vec{b}} = \vec{\delta}^{(2)} + \vec{\delta}^{(1)}.$$

The full analytical expression becomes

$$\boxed{\frac{\partial L}{\partial \vec{b}} = \left(\frac{\partial L}{\partial \hat{y}} \odot g_1'(\vec{a}^{(2)})\right) + \left[U^T \left(\frac{\partial L}{\partial \hat{y}} \odot g_1'(\vec{a}^{(2)})\right)\right] \odot g'(\vec{a}^{(1)}).}$$

2.0.2. Optimization and Convergence Effect

Yes, bias sharing is expected to affect convergence speed and stability.

In an independent-bias model, the parameters update according to their local layer errors:

$$\frac{\partial L}{\partial \vec{b}_1} = \vec{\delta}^{(1)}, \qquad \frac{\partial L}{\partial \vec{b}_2} = \vec{\delta}^{(2)}$$

In the shared-bias model, the single parameter $\vec{b}$ receives the vector sum of these gradients:

$$\frac{\partial L}{\partial \vec{b}} = \vec{\delta}^{(1)} + \vec{\delta}^{(2)}$$

This summation generally results in a gradient vector with a larger magnitude than either component alone. Consequently, for a fixed global learning rate α, the effective step size taken by $\vec{b}$ is larger relative to the weights (which only receive gradients from a single layer). This disproportionate update size can cause the bias parameter to overshoot its optimum or oscillate, destabilizing the training.

The error vectors $\vec{\delta}^{(2)}$ (direct output error) and $\vec{\delta}^{(1)}$ (back-propagated error scaled by U^T and g') often possess significantly different scales and directions. The shared bias is forced to find a compromise value that minimizes the error for both affine transformations simultaneously. If $\vec{\delta}^{(1)}$ and $\vec{\delta}^{(2)}$ point in opposing directions, the summed gradient may be small or erratic, leading to slow convergence. Furthermore, the mismatch in scale between the layers worsens the conditioning of the optimization problem, potentially requiring a carefully tuned, smaller learning rate to prevent divergence.

References

Aggarwal, C. C. *Neural Networks and Deep Learning*. Springer, Cham, 2018. ISBN 978-3-319-94462-3. doi: 10.1007/978-3-319-94463-0.

Little, R. and Rubin, D. *Statistical analysis with missing data*. Wiley series in probability and mathematical statistics. Probability and mathematical statistics. Wiley, 2002. ISBN 9780471183860. URL `http://books.google.com/books?id=aYPwAAAAMAAJ`.