Assignment 2

Deep Learning KU, WS 2024/25

Team Members		
Last name	First name	Matriculation Number
Hinum-Wagner	Jakob	01430617
Nozic	Naida	12336462

Tasks

All tasks were done in a virtual conda environment according to the assignment sheet and with python 3.11.9

a)

Firstly, we have provided histograms that showcase the feature distributions. What we can see is that Median Income appears to be slightly right skewed, were the most values lie in the 2-6 range. The Average Rooms, Average Bedrooms, Population and Average Occupation are all heavily right skewed features, with a small percentage of outliers (the number of outliers is provided in the table below). The Latitude presents a multimodal distribution with most data points falling between 34 and 38. The Longitude also showcases a similar multimodal distribution, where the data is clustered in the range [-122, -118]. Lastly, the House Age does not seem to indicate any clear pattern.

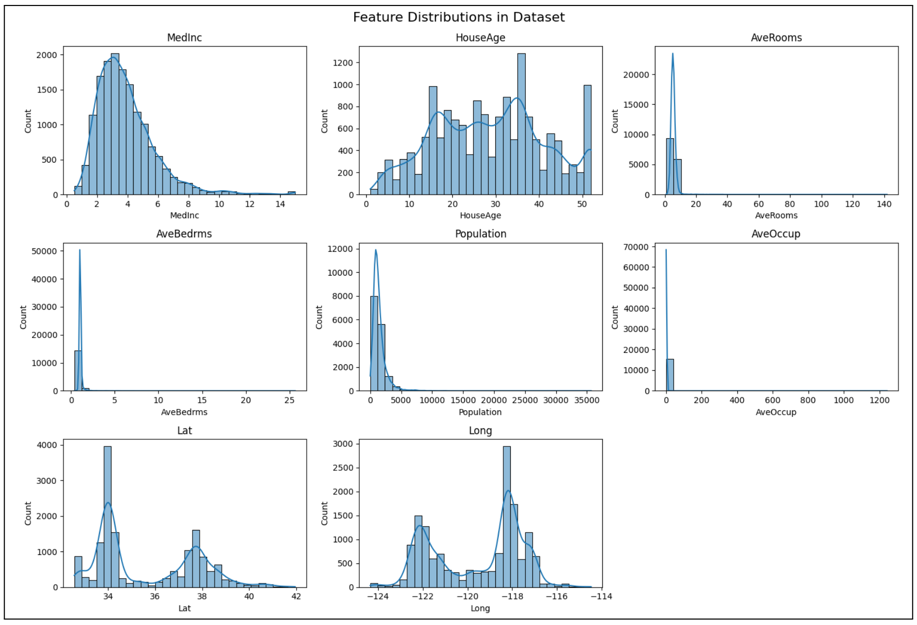

Figure 1: Feature distributions

Furthermore, the pair plot is generated to give us more information regarding feature relationships. We can conclude that there is a positive correlations between Median Income and Median House Value. Additionally, clustering is evident in Latitude and Longitude with Median House value, since specific regions on the plot are associated with higher or lower house values. We can also see a linear correlation between Average Rooms and Average Bedrooms. This is logical due to the fact that houses with more bedroom automatically have more rooms in total. A clean negative correlation is visible between Longitude and Latitude.

Feature	**Number of Outliers**
MedInc	513 (3.3%)
HouseAge	0 (0%)
AveRooms	384 (2.48%)
AveBedrms	1,060 (6.8%)
Population	908 (5.8%)
AveOccup	532 (3.4%)
Lat	0 (0%)
Long	0 (0%)
MedHouseVal	793 (5%)

Table 1: Number of Outliers for Each Feature

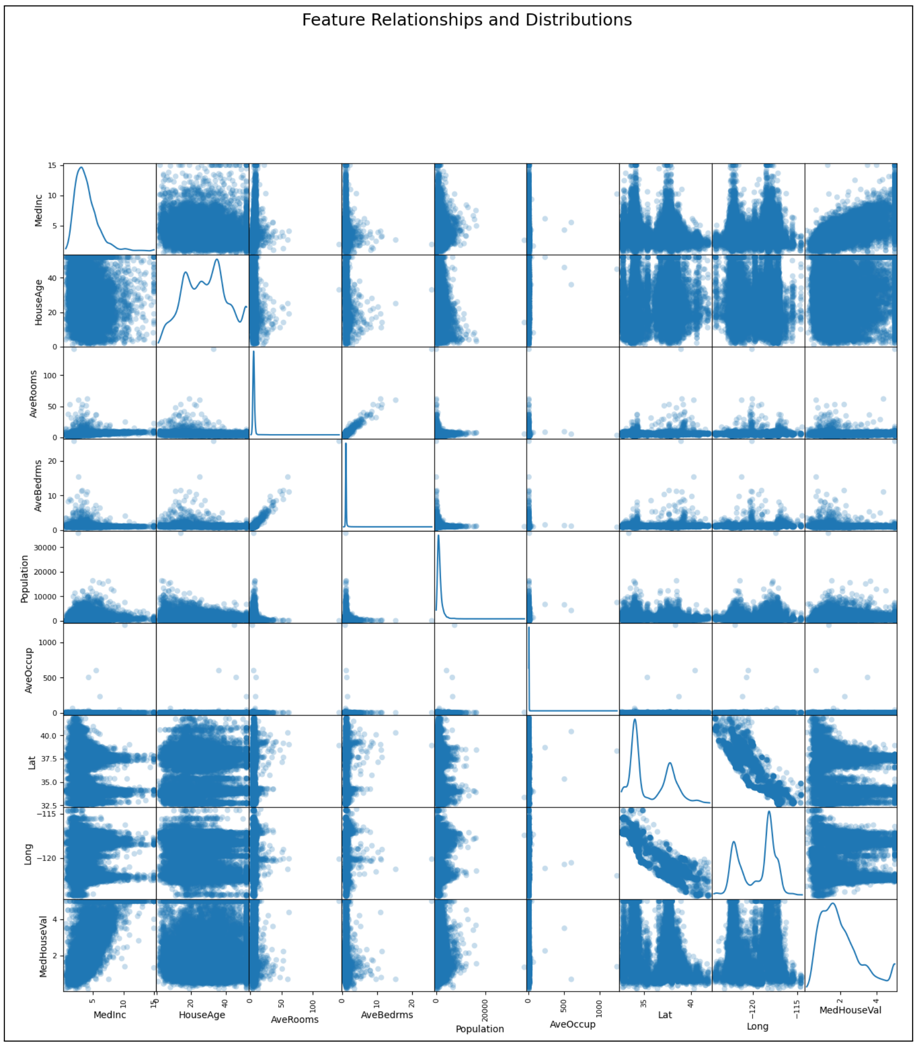

Figure 2: Feature relationships

To observe the relationship between features and the target value, we have provided a separate scatter plot with a fitted regression line. The most prominent relationship is present between Median Income and Median House Value. We see a positive correlation, indicating that higher values lead to higher values of the Median House Value.

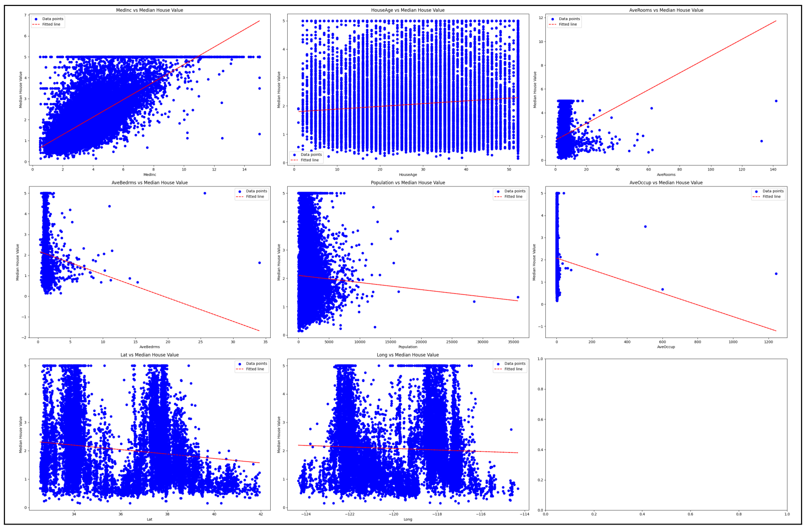

Figure 3: Feature vs Target relationships

The heat map gives us exact correlation values between all features and the target.

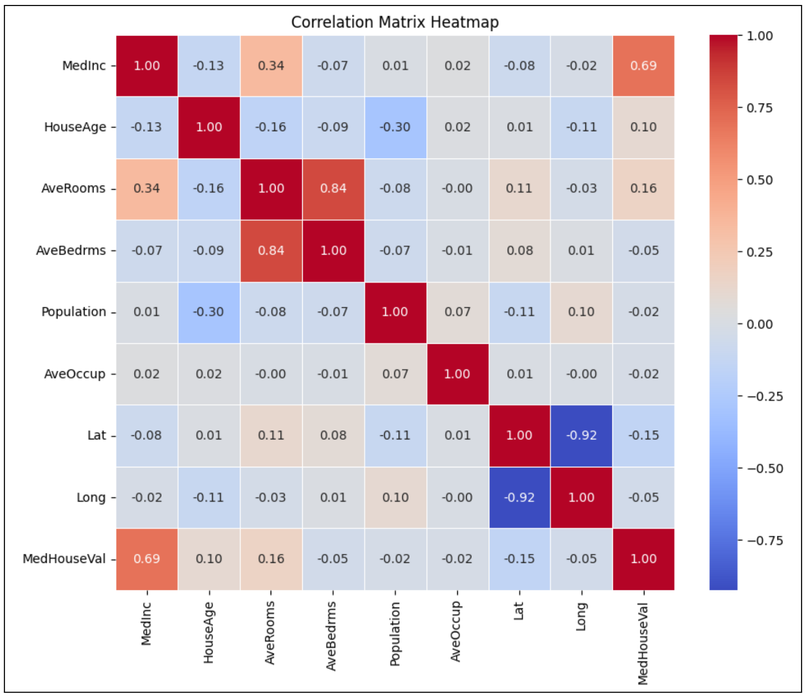

Figure 4: Feature vs Target relationships

It is important to note that the data were normalized before further manipulation in task b).

b)

The chosen architecture contains 3 hidden layers with 64, 32 and 16 neurons in each layer respectively. The input layer has 8 neurons, due to 8 features, and in the output layer we have placed one neuron since we are expecting one continuous value for the Median House Value. This architecture was selected as it gave the lowest validation loss of 0.277134, and train loss of 0.240074. As the loss function, we have used Mean Squared Error (MSE), because we find it suitable for the current regression task. It will minimize the squared differences between predicted and the true target values and also penalized large errors due to the squaring function. In the hidden layers we utilized the standard Rectified Linear Unit (ReLU), as the activation function and naturally no such function was added in the output layer since we are dealing with a regression task.

In order to determine the best model architecture, we have provided 6 tables with training and validation losses for comparison. Below, in the tables 2-6, the architecture is described with the first number presenting the number of neurons in the first layer, the last number representing the number of neurons in the last layer and everything in between is referring to the hidden layers. Regarding the batch sizes, we have ran one model on multiple sizes to see which one provides the best results. The size 50 has generated the lowest validation and train loss.
On the table 7, we have also provided the best epoch with the lowest validation loss and its training loss. This is provided for a better overview of which model gives the lowest loss.

Epoch	Train Loss	Validation Loss
10	0.402	0.442
20	0.379	0.411
30	0.367	0.405
40	0.360	0.393
50	0.357	0.391
60	0.355	0.394
70	0.352	0.388
80	0.352	0.384
90	0.350	0.381
100	0.345	0.377
110	0.341	0.374
120	0.338	0.371
130	0.335	0.369
140	0.334	0.364
150	0.333	0.366
160	0.331	0.363
170	0.331	0.363
180	0.329	0.362
190	0.329	0.359
200	0.327	0.360

Table 2: Model 1: [8, 50, 1]

Epoch	Train Loss	Validation Loss
10	0.390	0.417
20	0.377	0.408
30	0.354	0.383
40	0.327	0.355
50	0.310	0.330
60	0.298	0.320
70	0.289	0.320
80	0.283	0.313
90	0.278	0.324
100	0.271	0.313
110	0.267	0.307
120	0.265	0.300
130	0.259	0.298
140	0.258	0.300
150	0.255	0.293
160	0.254	0.291
170	0.252	0.291
180	0.249	0.289
190	0.246	0.293
200	0.246	0.304

Table 3: Model 2 : [8, 100, 50, 1]

c)

The objective of this study was to investigate and compare the performance of different optimizers—Stochastic Gradient Descent (SGD), Momentum SGD, and Adam—when applied to a regression task using a feedforward neural network. The evaluation was performed under varying learning rates and learning rate scheduling techniques. The task also included implementing early stopping to prevent overfitting and ensure efficient training.

The results were analyzed to identify the optimal combination of hyperparameters, including optimizer, learning rate, and scheduler. The lowest training and validation losses were tracked, and the best-performing model for each configuration was saved for further evaluation.

Theoretical Background

Optimizers Gradient-based optimization algorithms aim to minimize a loss function by iteratively updating model parameters in the direction of the negative gradient. The optimizers used in this study are described as follows:

- **SGD:** Stochastic Gradient Descent is a simple optimization algorithm that updates parameters based on the gradient of the loss function computed on a single batch. While effective, it can suffer from slow convergence and instability.

- **Momentum SGD:** This variant introduces a momentum term to the parameter updates, which helps accelerate convergence in the direction of consistent gradients and reduces oscillations in directions with noisy gradients.

Epoch	Train Loss	Validation Loss
10	0.393	0.433
20	0.381	0.410
30	0.374	0.404
40	0.345	0.373
50	0.330	0.364
60	0.320	0.362
70	0.305	0.342
80	0.296	0.330
90	0.290	0.324
100	0.282	0.318
110	0.276	0.312
120	0.273	0.307
130	0.270	0.301
140	0.261	0.301
150	0.255	0.299
160	0.252	0.293
170	0.255	0.295
180	0.243	0.284
190	0.242	0.281
200	0.241	0.287

Table 4: Model 3 : [8, 64, 32, 16, 1]

Epoch	Train Loss	Validation Loss
10	0.395	0.424
20	0.377	0.410
30	0.364	0.406
40	0.338	0.366
50	0.312	0.356
60	0.295	0.329
70	0.287	0.311
80	0.280	0.310
90	0.273	0.314
100	0.270	0.305
110	0.264	0.304
120	0.259	0.298
130	0.254	0.292
140	0.253	0.301
150	0.250	0.283
160	0.246	0.290
170	0.245	0.285
180	0.241	0.283
190	0.236	0.283
200	0.235	0.315

Table 5: Model 4 : [8, 100, 50, 10, 1]

- **Adam:** Adaptive Moment Estimation combines the benefits of RMSProp and Momentum. It uses moving averages of both the gradients and their squared values to adapt the learning rate for each parameter. This optimizer is particularly effective for tasks with sparse gradients or high-dimensional parameter spaces.

Learning Rate Schedulers Learning rate schedulers dynamically adjust the learning rate during training to enhance convergence:

- **StepLR:** Reduces the learning rate by a fixed factor after a predefined number of epochs, encouraging convergence as training progresses.
- **ReduceLROnPlateau:** Monitors validation loss and reduces the learning rate when improvements stagnate, allowing the optimizer to focus on fine-tuning the parameters.
- **None:** No scheduler is applied; the learning rate remains constant throughout training.

Early Stopping Early stopping halts training when the validation loss ceases to improve for a specified number of epochs, preventing overfitting and saving computational resources. The following components were used:

- **Patience:** The number of epochs to wait after the last improvement in validation loss.
- **Delta:** The minimum decrease in validation loss required to qualify as an improvement.
- **Best Model Tracking:** The model achieving the lowest validation loss was saved, ensuring that the final evaluation used the most optimal configuration.

Epoch	**Train Loss**	**Validation Loss**
10	0.394	0.431
20	0.373	0.410
30	0.359	0.390
40	0.347	0.378
50	0.331	0.361
60	0.314	0.343
70	0.305	0.337
80	0.298	0.330
90	0.295	0.322
100	0.293	0.321
110	0.288	0.317
120	0.288	0.329
130	0.284	0.316
140	0.282	0.323
150	0.280	0.319
160	0.278	0.307
170	0.272	0.304
180	0.271	0.306
190	0.267	0.316
200	0.264	0.297

Table 6: Model 5 : [8, 128, 64, 32, 1]

Model	**Best Train Loss**	**Best Validation Loss**	**Best Epoch**
Model_1	0.327237	0.358227	194
Model_2	0.247931	0.280148	190
Model_3	0.240074	0.277134	194
Model_4	0.240514	0.278426	180
Model_5	0.264134	0.292847	198

Table 7: Best Loss and Epoch Summary

Implementation Details

The feedforward neural network consisted of:

- Input Layer: 8 features
- Hidden Layers: 64, 32, and 16 neurons with ReLU activation
- Output Layer: A single neuron for regression

The experiments were conducted using:

- **Batch Size:** 50
- **Maximum Epochs:** 200
- **Early Stopping:** Patience of 40 epochs with a delta of 0.001
- **Gradient Clipping:** A maximum gradient norm to ensure numerical stability

Experimental Procedure

For each optimizer, learning rate, and scheduler combination:

1. The network was trained using the specified optimizer and learning rate.
2. Validation loss was monitored, and the model achieving the lowest validation loss was saved.
3. Early stopping was applied, and training was halted if no improvement in validation loss occurred for 40 consecutive epochs.
4. The training and validation losses were logged and analyzed and the best model is then taken with the lowest validation loss during training even if without early stopping the loss increases afterwards

Results

The results are summarized in Table 8, showing the best training and validation losses, along with the epoch at which the lowest validation loss was achieved.

Table 8: Comparison of Optimizers and Schedulers

Optimizer	**Learning Rate**	**Scheduler**	**Best Train Loss**	**Best Val Loss**	**Best Epoch**
SGD	0.001	None	1.3304	1.3723	19
SGD	0.001	StepLR	1.3301	1.3728	35
SGD	0.001	ReduceLROnPlateau	1.3311	1.3722	21
SGD	0.010	None	1.3297	1.3714	20
SGD	0.010	StepLR	1.3307	1.3714	8
SGD	0.010	ReduceLROnPlateau	1.3292	1.3714	14
SGD	0.100	None	0.2201	0.2723	167
SGD	0.100	StepLR	0.3535	0.3831	59
SGD	0.100	ReduceLROnPlateau	0.2255	0.2734	119
SGD_Momentum	0.001	None	1.3293	1.3714	14
SGD_Momentum	0.001	StepLR	1.3300	1.3716	2
SGD_Momentum	0.001	ReduceLROnPlateau	1.3297	1.3714	15
SGD_Momentum	0.010	None	0.2356	0.2712	108
SGD_Momentum	0.010	StepLR	1.3324	1.3714	7
SGD_Momentum	0.010	ReduceLROnPlateau	1.3307	1.3705	22
SGD_Momentum	0.100	None	1.3625	1.3716	12
SGD_Momentum	0.100	StepLR	1.3303	1.3716	23
SGD_Momentum	0.100	ReduceLROnPlateau	1.3400	1.3714	20
Adam	0.001	None	0.2492	0.2830	127
Adam	0.001	StepLR	0.3778	0.4125	47
Adam	0.001	ReduceLROnPlateau	0.2596	0.3015	175
Adam	0.010	None	0.2565	0.2873	57
Adam	0.010	StepLR	0.2852	0.3198	29
Adam	0.010	ReduceLROnPlateau	0.2120	0.2744	82
Adam	0.100	None	0.3833	0.3733	28
Adam	0.100	StepLR	0.3209	0.3558	45

Optimizer	Learning Rate	Scheduler	Best Train Loss	Best Val Loss	Best Epoch
Adam	0.100	ReduceLROnPlateau	0.2804	0.3144	75

Analysis and Discussion

- **Optimizer Performance:** SGD_Momentum outperformed Adam and standard SGD, achieving the lowest validation loss. This demonstrates the advantage of momentum-based optimization in effectively navigating the loss landscape.
- **Learning Rates:** A learning rate of 0.01 proved optimal for SGD_Momentum, offering a good balance between convergence speed and stability, enabling the model to achieve the best results.
- **Schedulers:** The configuration with no scheduler highlighted that a static learning rate can yield exceptional performance when paired with an effective optimization method like SGD_Momentum.
- **Early Stopping:** Played a crucial role in identifying the optimal point at epoch 108, ensuring computational efficiency and preventing overfitting.

Conclusion

The SGD_Momentum optimizer with a learning rate of 0.01 and no scheduler achieved the best validation loss of 0.2712, with a training loss of 0.2356, at epoch 108. This highlights the effectiveness of momentum-based optimization combined with a moderate learning rate and a straightforward training approach for optimal performance.

d)

After investigating five types of architecture and experimenting with a variety of optimizers, learning rates and schedulers, we have determined the optimal model, with the following characteristics:

Summary of the chosen architecture:
Input layer: 8 neurons
Three hidden layers:

- 1. hidden layer: 64 neurons
- 2. hidden layer: 32 neurons
- 3. hidden layers: 16 neurons

Output layer: 1 neuron

The chosen batch size is 50. Activation function applied between the layers is ReLU (the output layer does not have the activation function, since its a regression task). We have utilized the Adam optimizer with the learning rate of 0.001 and ReduceLROnPlateau scheduler. The lowest validation loss with this architecture occurred in the epoch 82 with the value of 0.274383. In the given epoch, we have also detected the lowest training loss of 0.211952. After training the model on the whole training set (combined training and validation set), we obtained the final test loss of 0.2670.

With determining the best architecture, we have conducted the training process once more and plotted the graph on Figure 5, that showcases the evolution of the train and validation loss throughout iterations. Both the losses have been gradually decreasing with minor rises. At the end we can see the lowest loss value between 0.2-0.3 and with a steady curve, indicating stability in the model.

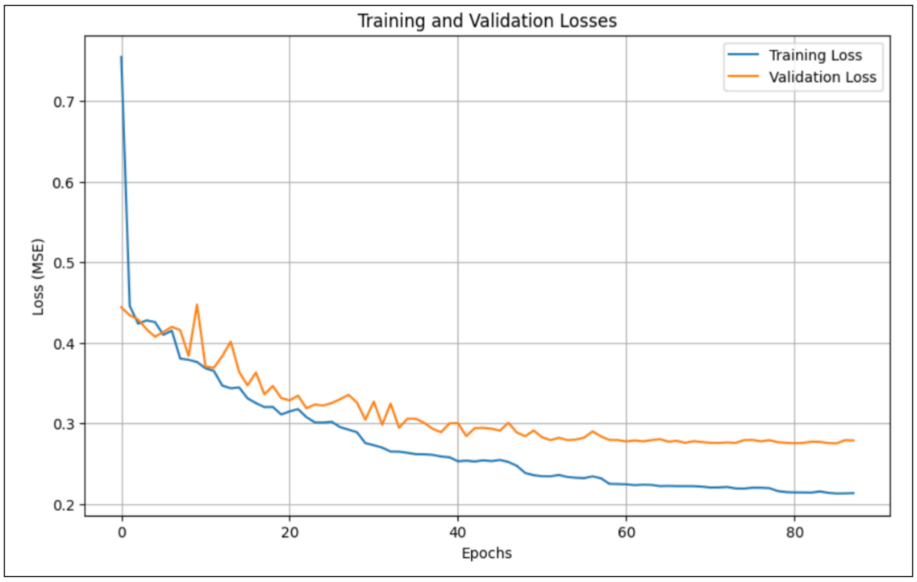

Figure 5: Evolution of train and validation loss

Furthermore, Figure 6 contains a scatter plot of model predictions (x-axis) with their ground truth values (y-axis) on the test set. We have also provided a red regression line. The closer the points are to the line, the closer are the predictions to the ground truth. We observe a steady model with relatively low deviations from the true y values.

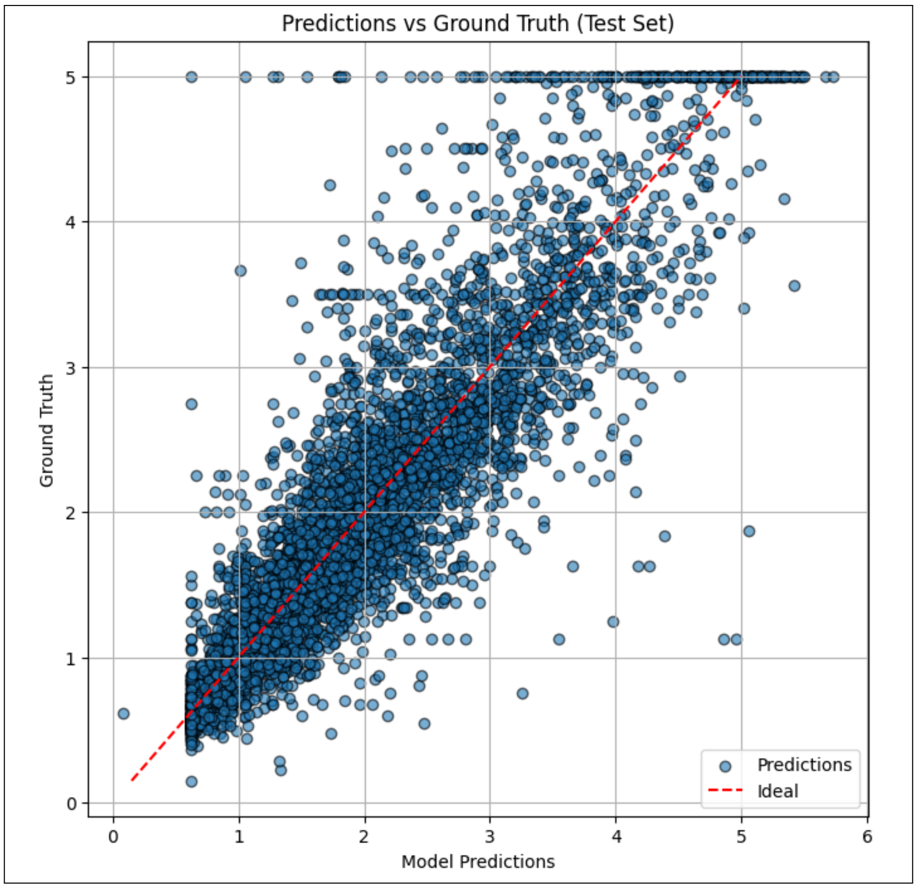

Figure 6: Scatter Plot: Predictions vs. Ground Truth (Test Set)

e)

This derivation aims to demonstrate that minimizing the Mean Squared Error (MSE) loss function is equivalent to maximizing the likelihood function under specific conditions. Given a training dataset:

$$D = \{(x_1, y_1),\ (x_2, y_2),\ \ldots,\ (x_n, y_n)\},$$

where each input $x_i \in \mathbb{R}^d$ and corresponding target $y_i \in \mathbb{R}$. Assume a neural network model $f_\theta : \mathbb{R}^d \to \mathbb{R}$ parameterized by θ. We aim to prove:

$$\arg\min_\theta \frac{1}{n} \sum_{i=1}^{n} \left(f_\theta(x_i) - y_i\right)^2 = \arg\max_\theta p_\theta(y_1, y_2, \ldots, y_n \,|\, x_1, x_2, \ldots, x_n),$$

under the following assumptions:

1. **Independence Assumption:** The outputs y_i are independent given the inputs x_i, i.e., $y_i \sim p^*(y_i \,|\, x_i)$.

2. **Gaussian Noise Assumption:** The conditional probability $p_\theta(y_i \mid x_i)$ follows a Gaussian distribution with mean $f_\theta(x_i)$ and fixed variance σ^2:

$$p_\theta(y_i \mid x_i) = \mathcal{N}(y_i; f_\theta(x_i), \sigma^2).$$

Step 1: Defining the MSE Loss Function The MSE loss function quantifies the average squared difference between predictions and actual outputs:

$$\text{MSELoss}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \left(f_\theta(x_i) - y_i\right)^2.$$

Minimizing this loss yields the parameter set:

$$\theta_{\text{MSE}} = \arg\min_\theta \text{MSELoss}(\theta).$$

Step 2: Formulating the Likelihood Function Given the Gaussian assumption, the conditional probability density for y_i is:

$$p_\theta(y_i \mid x_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\left(y_i - f_\theta(x_i)\right)^2}{2\sigma^2}\right).$$

Since y_i are independent, the joint likelihood for all outputs is:

$$p_\theta(y_1, y_2, \ldots, y_n \mid x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} p_\theta(y_i \mid x_i).$$

Maximizing this likelihood identifies the parameters:

$$\theta_{\text{MLE}} = \arg\max_\theta p_\theta(y_1, y_2, \ldots, y_n \mid x_1, x_2, \ldots, x_n).$$

Step 3: Simplifying the Likelihood via Logarithm Taking the natural logarithm of the likelihood simplifies the expression:

$$\mathcal{L}(\theta) = \ln p_\theta(y_1, y_2, \ldots, y_n \mid x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} \ln p_\theta(y_i \mid x_i).$$

Substitute $p_\theta(y_i \mid x_i)$ into the log function:

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} \left[\ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{\left(y_i - f_\theta(x_i)\right)^2}{2\sigma^2}\right].$$

The constant term simplifies as:

$$\ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) = -\frac{1}{2}\ln(2\pi\sigma^2),$$

so the log-likelihood becomes:

$$\mathcal{L}(\theta) = -\frac{n}{2}\ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n} \left(y_i - f_\theta(x_i)\right)^2.$$

Step 4: Reducing the Optimization Problem Since $-\frac{n}{2}\ln(2\pi\sigma^2)$ does not depend on θ, it can be ignored. Maximizing the log-likelihood simplifies to:

$$\theta_{\text{MLE}} = \arg\max_{\theta} \left(-\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2 \right).$$

Step 5: Establishing Equivalence The term $-\frac{1}{2\sigma^2}$ is a negative constant, so maximizing $\mathcal{L}(\theta)$ is equivalent to minimizing:

$$\theta_{\text{MLE}} = \arg\min_{\theta} \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2 .$$

Multiplying the MSE loss by n does not affect the minimizer, so:

$$\theta_{\text{MSE}} = \arg\min_{\theta} \sum_{i=1}^{n} (f_\theta(x_i) - y_i)^2 .$$

Thus, $\theta_{\text{MLE}} = \theta_{\text{MSE}}$.

Assumption and Methodological Insights

- **Independence**: This ensures the likelihood decomposes into a product of individual terms.
- **Gaussian Noise**: The assumption of Gaussian residuals enables the equivalence between minimizing squared errors and maximizing the likelihood.
- **Logarithm Transformation**: Simplifies optimization by converting a product into a sum.
- **Constant Terms**: Removing constants independent of θ does not alter the optimization result.

This derivation confirms that minimizing the MSE loss function is mathematically equivalent to maximizing the likelihood under Gaussian noise, bridging statistical estimation with regression model training.

0.1 f)

In this task, we adapt the existing neural network architecture, originally designed for a regression task, to address a binary classification problem. Specifically, we aim to classify whether the median house value is below or above $200,000. This adaptation requires modifications to the network architecture, the training pipeline, and the evaluation metrics. We discuss these changes in detail, providing theoretical justifications and practical considerations.

0.1.1 Modifications to the Architecture and Training Pipeline

Transforming a regression model into a binary classifier involves several key changes:

Output Layer Activation Function

- **Original:** The regression model uses a linear activation function (or no activation function) in the output layer to predict continuous values.
- **Modification:** Replace the linear activation function with a sigmoid activation function in the output layer.
- **Reason:** The sigmoid function, defined as $\sigma(z) = \frac{1}{1+e^{-z}}$, maps any real-valued input z to a value between 0 and 1. This property makes it suitable for producing probability estimates in binary classification tasks, where the output represents the probability of belonging to a particular class.

Loss Function

- **Original:** The regression model uses Mean Squared Error (MSE) loss, which is appropriate for continuous target variables.
- **Modification:** Replace MSE loss with Binary Cross Entropy (BCE) loss.
- **Reason:** BCE loss is specifically designed for binary classification problems. It quantifies the difference between the predicted probabilities and the actual binary labels. The BCE loss function is given by:

  $$\text{BCE Loss} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right], \tag{1}$$

  where N is the number of samples, y_i is the true binary label (0 or 1), and $\hat{y}_i$ is the predicted probability.

Evaluation Metrics

- **Original:** Regression models are evaluated using metrics like Mean Squared Error (MSE) or Mean Absolute Error (MAE), which measure the average discrepancy between predicted and actual continuous values.
- **Modification:** Use classification metrics such as accuracy, precision, recall, F1-score, and Receiver Operating Characteristic Area Under the Curve (ROC-AUC).
- **Reason:** These metrics are appropriate for classification tasks and provide insights into the model's ability to correctly predict class labels, balance false positives and false negatives, and overall classification performance. For example:
  - **Accuracy** measures the proportion of correct predictions.
  - **Precision** evaluates the correctness of positive predictions.
  - **Recall** assesses the model's ability to identify all positive instances.
  - **F1-score** is the harmonic mean of precision and recall.
  - **ROC-AUC** quantifies the model's ability to distinguish between the two classes across all thresholds.

Architecture Adjustments While the hidden layers of the network can remain largely unchanged, it's crucial to adjust the output layer to produce a single probability value suitable for binary classification. The architecture can be summarized as follows:

- **Input Layer:** Accepts the feature vector of size 8.
- **Hidden Layers:** Three hidden layers with sizes 64, 32, and 16 neurons, respectively, each followed by a ReLU activation function to introduce non-linearity.
- **Output Layer:** A single neuron with a sigmoid activation function to output a probability between 0 and 1.

0.1.2 Implementation of Changes

Data Preparation To adapt the dataset for binary classification, the following steps are applied:

- **Target Transformation:** The target variable is redefined as binary labels:
  - Assign 0 to samples where the median house value is below $200,000.
  - Assign 1 to samples where the median house value is at least $200,000.
- **Feature Normalization:** All features are standardized using `StandardScaler` to ensure they have a mean of zero and a standard deviation of one.
- **Data Splitting:** The dataset is divided into training and testing sets, with 75% allocated for training and 25% for testing.
- **Tensor Conversion:** The preprocessed data is converted to PyTorch tensors to facilitate model training.

Model Architecture The binary classification model is structured as follows:

- **Layer Design:** The network comprises one input layer, multiple hidden layers activated by ReLU, and a single output layer.
- **Output Configuration:** The output layer contains one neuron with a sigmoid activation function, which outputs the probability of the positive class.

Training Pipeline The training pipeline is configured with these components:

- **Loss Function:** `BCEWithLogitsLoss` is employed to combine sigmoid activation with binary cross-entropy loss in a numerically stable way.
- **Optimizer:** The optimization process uses Stochastic Gradient Descent (`SGD`) with momentum to accelerate convergence.
- **Batch Size:** Training data is processed in batches of 50 samples.
- **Early Stopping:** Training stops if validation loss fails to improve for 8 consecutive epochs.

Training Steps The training loop executes the following:

1. **Forward Pass:** Input data flows through the network to produce predictions.
2. **Loss Calculation:** Binary cross-entropy loss is computed using the predictions and true labels.
3. **Backward Pass:** Gradients of the loss with respect to model parameters are calculated.
4. **Weight Update:** The optimizer updates weights using the gradients.
5. **Validation Monitoring:** Model performance is evaluated on the validation set, guiding early stopping.

Model Assessment After completing training, the model's test performance is analyzed using the following metrics:

- **Accuracy:** Measures the fraction of correctly classified instances.
- **Precision:** Evaluates the ratio of true positives to all predicted positives.
- **Recall:** Assesses the ratio of true positives to all actual positives.
- **F1 Score:** Provides a balanced measure of precision and recall.
- **ROC-AUC:** Captures the model's ability to distinguish between classes across thresholds.

Visualizations and Outcomes

- **Confusion Matrix:** Displays the distribution of true and predicted classes.
- **Loss Trend:** Training and validation loss curves are plotted to visualize convergence over epochs.
- **ROC Curve:** The ROC curve is plotted to demonstrate the trade-off between sensitivity and specificity.

0.1.3 Results and Discussion

Evaluation Metrics The model's performance on the test set is assessed using the following metrics:

- **Accuracy (A):**

$$A = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}, \tag{2}$$

  where TP is true positives, TN is true negatives, FP is false positives, and FN is false negatives.

- **Precision (P):**

$$P = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{3}$$

  representing the proportion of positive identifications that are correct.

- **Recall (R):**
$$R = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{4}$$
indicating the proportion of actual positives correctly identified.

- **F1-score ($F1$):**
$$F1 = 2 \times \frac{P \times R}{P + R}, \tag{5}$$
balancing precision and recall.

- **ROC-AUC Score:** The area under the ROC curve (AUC) quantifies the model's ability to discriminate between classes at various thresholds.

- **Confusion Matrix:** A table of TP, TN, FP, and FN providing insight into the types of classification errors.

Performance Analysis The trained model achieves the following performance metrics on the test set:

- **Accuracy:** 87.33%
- **Precision:** 86.96%
- **Recall:** 82.79%
- **F1-score:** 84.83%
- **ROC-AUC:** 95.21%

These results indicate that the model effectively distinguishes between houses below and above $200,000 in median value. The high ROC-AUC score demonstrates excellent discriminative ability, and the balanced precision and recall suggest that the model performs well across both classes.

Confusion Matrix Figure 7 shows the confusion matrix, summarizing the model's predictions:

- True negatives: $2,678$
- False positives: 274
- False negatives: 380
- True positives: $1,828$

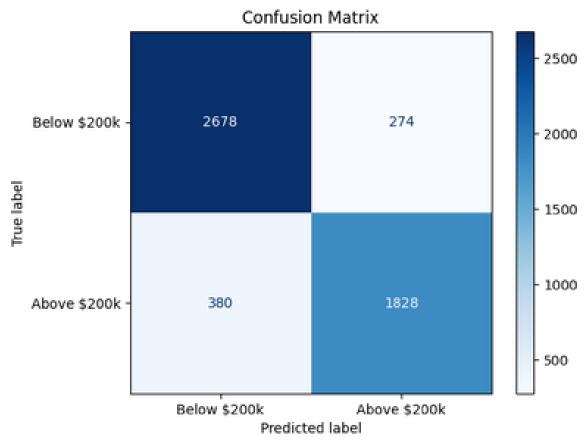

Figure 7: Confusion Matrix for Binary Classification

ROC Curve The Receiver Operating Characteristic (ROC) curve, depicted in Figure 8, demonstrates the trade-off between true positive rate and false positive rate at different thresholds. The AUC of 0.9521 reflects strong overall performance.

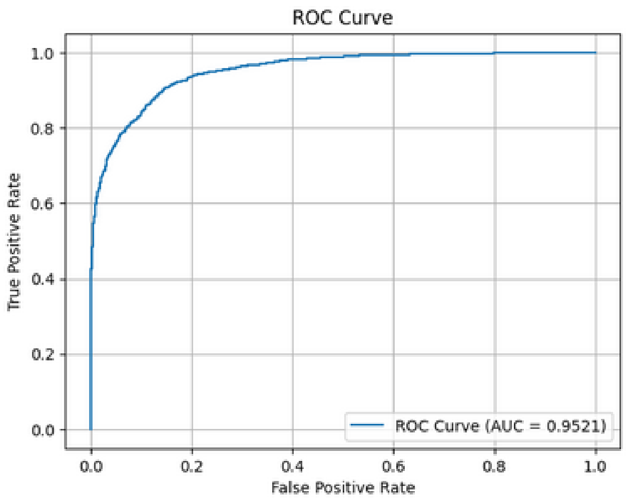

Figure 8: ROC Curve with AUC = 0.9521

Training and Validation Loss Figure 9 illustrates the training and validation loss across epochs. The early stopping criterion triggered at epoch 67 when the validation loss plateaued, preventing overfitting.

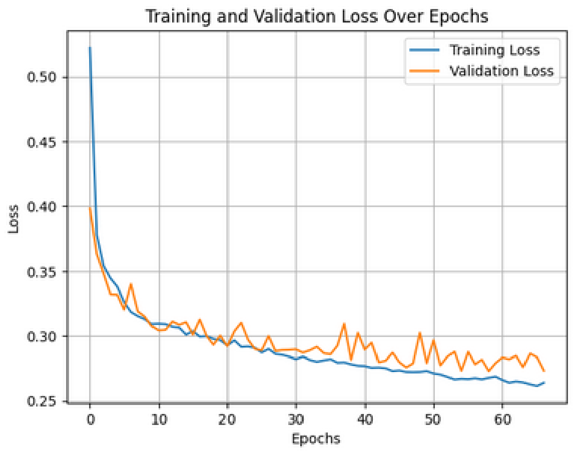

Figure 9: Training and Validation Loss Over Epochs

Interpretation and Conclusion The results confirm that the modifications to the neural network architecture and training pipeline successfully adapted the regression model for binary classification. Key changes included:

- Adding a sigmoid activation to the output layer for probability estimation.
- Using Binary Cross Entropy loss to suit the binary classification task.
- Incorporating appropriate evaluation metrics such as precision, recall, F1-score, and ROC-AUC.

The model achieves high accuracy and demonstrates robustness in identifying houses below and above the $200,000 threshold, evidenced by the balanced evaluation metrics and high ROC-AUC score.