

**University of Tehran**  
**School of Mechanical Engineering**



# **Artificial Intelligence**

## **Home Work 4**

**Professor:**

Dr. Masoud Shariat Panahi

**Author:**

Masoud Pourghavam

June, 2023

Table of Contents

1. Fundamentals ..... 4

    A. Imbalanced datasets ..... 4

    B. Binary and categorical cross entropy ..... 5

    C. Accuracy ..... 7

    D. Normalization and standardization ..... 8

2. Housing market predictions ..... 10

    A. Import the data ..... 11

        i. Columns and rows report ..... 11

        ii. Correlation matrix ..... 11

        iii. House price distribution ..... 13

    B. Data preprocessing ..... 15

        i. Date column ..... 15

        ii. Input and outputs ..... 16

        iii. Training and testing sets ..... 17

        iv. Scale ..... 18

    C. Implementation of the model ..... 19

        i. Network design ..... 19

        ii. Network train ..... 20

    D. Demands ..... 21

        i. Loss, validation loss,  $R^2$ , MAE, and MSE ..... 21

        ii. Epochs ..... 34

        iii. Learning rate ..... 36

        iv. Tanh activation function ..... 37

        v. Batch size ..... 40

        vi. Confusion matrix ..... 47

3. Machines classification ..... 48

    A. Data preprocessing ..... 48

        i. Input columns (X) and output column (Y) ..... 49

        ii. Standardization ..... 49

    B. Neural network training ..... 50

    C. Demands ..... 52

        i. Hidden layer neurons ..... 52

        ii. Performance and Max\_fail ..... 58

## List of Figures

Fig 1. Example of balanced and imbalanced datasets .....	4
Fig 2. Correlation matrix .....	12
Fig 3. Distribution of house price.....	14
Fig 4. Date column transformation.....	16
Fig 5. Loss and validation loss for $Lr = 0.001$ and 20 epochs .....	22
Fig 6. Accuracy of training set predictions for $Lr = 0.001$ and 20 epochs.....	22
Fig 7. Accuracy of testing set predictions for $Lr = 0.001$ and 20 epochs .....	23
Fig 8. Loss and validation loss for $Lr = 0.1$ and 20 epochs .....	25
Fig 9. Accuracy of training set predictions for $Lr = 0.1$ and 20 epochs.....	25
Fig 10. Accuracy of testing set predictions for $Lr = 0.1$ and 20 epochs .....	26
Fig 11. Loss and validation loss for $Lr = 0.001$ and 4000 epochs .....	28
Fig 12. Accuracy of training set predictions for $Lr = 0.001$ and 4000 epochs.....	28
Fig 13. Accuracy of testing set predictions for $Lr = 0.001$ and 4000 epochs .....	29
Fig 14. Loss and validation loss for $Lr = 0.1$ and 4000 epochs .....	31
Fig 15. Accuracy of training set predictions for $Lr = 0.1$ and 4000 epochs.....	31
Fig 16. Accuracy of testing set predictions for $Lr = 0.1$ and 4000 epochs .....	32
Fig 17. Optimal number of epochs.....	35
Fig 18. Loss and validation loss for $Lr = 0.001$ and 4000 epochs with Tanh .....	37
Fig 19. Accuracy of training set predictions for $Lr = 0.001$ with Tanh .....	38
Fig 20. Accuracy of testing set predictions for $Lr = 0.001$ with Tanh.....	38
Fig 21. Loss and validation loss for $Lr = 0.001$ and batch = 1.....	40
Fig 22. Accuracy of training set predictions for $Lr = 0.001$ and batch = 1 .....	40
Fig 23. Accuracy of testing set predictions for $Lr = 0.001$ and batch = 1.....	41
Fig 24. Loss and validation loss for $Lr = 0.001$ and batch = 256.....	44
Fig 25. Accuracy of training set predictions for $Lr = 0.001$ and batch = 256 .....	44
Fig 26. Accuracy of testing set predictions for $Lr = 0.001$ and batch = 256.....	45
Fig 27. Confusion matrix for testing data.....	47
Fig 28. Performance of ANN with 10 neurons .....	51
Fig 29. Performance of ANN with 1 neuron.....	53
Fig 30. Performance of ANN with 30 neurons .....	54
Fig 31. Performance of ANN with 500 neurons .....	56
Fig 32. Performance of ANN with 20 neurons .....	58

## List of Tables

Table 1. Libraries used in section 2 .....	10
Table 2. Strongest correlations with price .....	13
Table 3. Train matrices for $lr = 0.001$ and 20 epochs.....	23
Table 4. Test matrices for $lr = 0.001$ and 20 epochs .....	23
Table 5. Train matrices for $lr = 0.1$ and 20 epochs.....	26
Table 6. Test matrices for $lr = 0.1$ and 20 epochs .....	26
Table 7. Train matrices for $lr = 0.001$ and 4000 epochs.....	28
Table 8. Test matrices for $lr = 0.001$ and 4000 epochs .....	29
Table 9. Train matrices for $lr = 0.1$ and 4000 epochs.....	31
Table 10. Test matrices for $lr = 0.1$ and 4000 epochs .....	32
Table 11. Train matrices for $lr = 0.1$ and 4000 epochs.....	38
Table 12. Test matrices for $lr = 0.1$ and 4000 epochs .....	39
Table 13. Train matrices for $lr = 0.001$ and batch = 1 .....	41
Table 14. Test matrices for $lr = 0.001$ and batch = 1.....	41
Table 15. Train matrices for $lr = 0.001$ and batch = 256 .....	45
Table 16. Test matrices for $lr = 0.001$ and batch = 256.....	45
Table 17. RMSE of test and train.....	51
Table 18. RMSE of test and train with 1 neuron .....	52
Table 19. RMSE of test and train with 30 neurons .....	54
Table 20. RMSE of test and train with 500 neurons .....	55

# 1. Fundamentals

## A. Imbalanced datasets

### Problem explanation:

The imbalanced datasets problem refers to a situation where the distribution of classes in a dataset is significantly skewed, meaning that one class has a much larger number of instances than the other(s). This imbalance can cause issues when training machine learning models, especially in classification tasks. The minority class (the one with fewer instances) often gets neglected during the training process, leading to poor performance and biased predictions.

Imbalanced datasets can occur in various real-world scenarios, such as fraud detection (where fraudulent transactions are rare compared to legitimate ones), disease diagnosis (where rare diseases have fewer instances), or rare event prediction. An example for imbalanced and balanced data is shown in Figure 1.

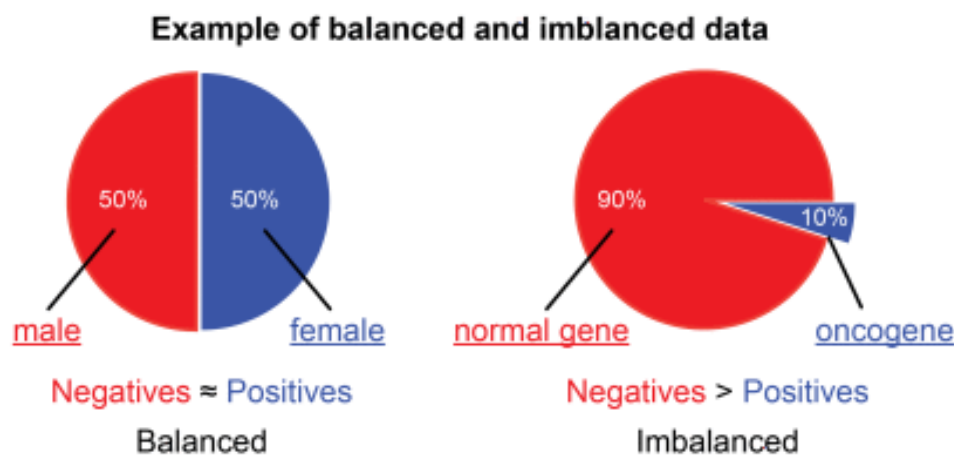


Fig 1. Example of balanced and imbalanced datasets

### Methods used to address the imbalanced datasets problem:

- **Resampling techniques:** Resampling involves either oversampling the minority class or undersampling the majority class to achieve a more balanced distribution. Oversampling methods create synthetic instances of the minority class, such as through duplication or using techniques like SMOTE (Synthetic Minority Over-sampling Technique). Undersampling methods reduce the number of instances in the majority class, often randomly or by selecting representative samples.

- **Data augmentation**: Data augmentation techniques involve creating new training instances by applying various transformations to the existing instances. This can help increase the diversity of the minority class and balance the dataset. Augmentation methods include rotation, scaling, flipping, or adding random noise to the data.
- **Class weighting**: In this approach, different weights are assigned to the classes during the training process. The weight of each class is inversely proportional to its frequency in the dataset. By assigning higher weights to the minority class, the model is encouraged to pay more attention to it during training.
- **Ensemble methods**: Ensemble methods combine multiple models to improve predictive performance. In the context of imbalanced datasets, techniques like bagging and boosting can be used to create diverse models that collectively handle class imbalance more effectively. For example, boosting algorithms like AdaBoost or XGBoost can assign higher weights to misclassified instances of the minority class in subsequent iterations, forcing the model to focus more on these instances.
- **Anomaly detection**: Anomaly detection techniques can be useful in cases where the minority class represents rare or anomalous events. Instead of directly training a classifier, anomaly detection algorithms aim to identify instances that deviate significantly from the majority class. This approach can be effective when the focus is on detecting rare events rather than classifying them.

## B. Binary and categorical cross entropy

### Binary Cross Entropy:

Binary Cross Entropy is a loss function commonly used in machine learning, particularly in binary classification tasks. It measures the dissimilarity or "cross entropy" between the predicted probability distribution and the true binary labels. It is defined as:

$$\text{Binary Cross Entropy} = -[y \times \log(p) + (1 - y) \times \log(1 - p)]$$

Where,  $y$  is the true label (either 0 or 1) and  $p$  is the predicted probability of the positive class. The Binary Cross Entropy loss function penalizes the model more strongly when it predicts a probability far from the true label, encouraging the model to learn accurate probabilities for binary classification.

Categorical Cross Entropy:

Categorical Cross Entropy is a loss function used in machine learning for multi-class classification tasks. It measures the dissimilarity between the predicted probability distribution and the true categorical labels. It is defined as:

$$\text{Categorical Cross Entropy} = -\sum(y \times \log(p))$$

Where,  $y$  is a one-hot encoded vector representing the true label and  $p$  is a vector of predicted probabilities for each class. The Categorical Cross Entropy loss function sums the cross entropy for each class and penalizes the model based on the deviation between predicted probabilities and the true label. It encourages the model to assign higher probabilities to the correct class and lower probabilities to incorrect classes.

Differences:

Binary Cross Entropy is used for binary classification tasks, where there are two classes (e.g., true/false, positive/negative), while Categorical Cross Entropy is used for multi-class classification tasks, where there are more than two classes. On the other hand, Binary Cross Entropy considers only one probability (for the positive class) and penalizes based on the true binary label, whereas Categorical Cross Entropy considers a vector of probabilities for each class and penalizes based on the true categorical label. Also, Binary Cross Entropy can be used when there are only two classes, while Categorical Cross Entropy is suitable when there are three or more classes.

Using Binary and Categorical Cross Entropy in a regression problem:

Both Binary Cross Entropy and Categorical Cross Entropy are primarily designed for classification tasks, not regression problems. They are loss functions used to measure the dissimilarity between predicted probabilities and true labels. In regression problems, where the goal is to predict continuous values, it is more common to use loss functions such as Mean Squared Error (MSE) or Mean Absolute Error (MAE). These loss functions directly measure the distance or difference between the predicted continuous values and the true labels, which is more appropriate for regression tasks.

While it is technically possible to use Binary Cross Entropy or Categorical Cross Entropy in a regression problem by converting the regression values into categorical or binary labels, it is

not a recommended approach. It would likely result in suboptimal performance and would not leverage the full capabilities of these loss functions.

### C. Accuracy

No, accuracy alone is not always a reliable measure of a model's performance. While accuracy is a commonly used metric to evaluate classification models, it has certain limitations that make it insufficient in some cases. One limitation is that accuracy does not consider the distribution of classes in the dataset. If the dataset is imbalanced, meaning one class significantly outweighs the others, a model that simply predicts the majority class for every instance can achieve a high accuracy, even though it provides no meaningful insights or predictive power for the minority classes. In such cases, metrics like precision, recall, and F1-score are more informative, as they take into account the performance on individual classes.

Moreover, accuracy does not provide information about the types of errors a model makes. It treats all misclassifications equally, regardless of their severity. In many real-world scenarios, different types of errors have different consequences. For example, in medical diagnosis, a false negative (predicting a patient as healthy when they have a disease) can have more serious implications than a false positive. Therefore, relying solely on accuracy can be misleading and insufficient to fully understand the model's performance.

Additionally, accuracy does not capture the uncertainty or confidence of the model's predictions. A model may have high accuracy on the test set but could still make unreliable predictions for certain inputs. Assessing the uncertainty of the model's predictions can be crucial, especially in safety-critical applications or when making decisions based on the model's outputs. Other measures like calibration, confidence intervals, or probabilistic predictions can provide a more comprehensive understanding of the model's performance and reliability. So, while accuracy is a useful metric, it should be complemented with other evaluation measures to gain a comprehensive understanding of a model's performance, especially in scenarios with imbalanced data, varying costs of different types of errors, and the need for uncertainty estimation.



#### D. Normalization and standardization

Data normalization and standardization are preprocessing techniques used to transform input data into a standardized format before feeding it into a machine learning algorithm, such as a Multi-Layer Perceptron (MLP) neural network.

##### Data normalization:

It involves rescaling the input features to have a common scale. The most common approach is to normalize the data to a range between 0 and 1. This is typically done by subtracting the minimum value of the feature and dividing it by the range (the difference between the maximum and minimum values). Normalization ensures that all features have equal importance and prevents some features with larger magnitudes from dominating the learning process.

##### Data standardization:

It involves transforming the input features to have zero mean and unit variance. This is typically done by subtracting the mean of the feature and dividing it by the standard deviation. Standardization brings the features onto a similar scale, centered around zero, which helps the optimization algorithms converge more quickly. It also ensures that features with different units and scales contribute equally to the learning process.

##### If data normalization & standardization are not performed:

Features with larger magnitudes or variances can cause the optimization algorithm to converge slowly. This is because the weights and biases of the MLP are adjusted based on the input data, and when the data has a wide range or varying scales, the learning process becomes inefficient. Without normalization or standardization, features with larger magnitudes may dominate the learning process. The MLP may assign more importance to these features, leading to biased results and neglecting other features that could be equally or more informative. Large variations in the input data can cause oscillations or divergence during the learning process. The weights and biases may fluctuate widely, making it challenging for the MLP to converge to an optimal solution. The optimization algorithm relies on gradients to update the MLP's weights and biases. If the input data is not normalized or standardized, the gradients can have significantly different scales, leading to inefficient and unbalanced updates.

9	Masoud Pourghavam 810601044	Artificial Intelligence Dr. M. S. Panahi	HW4	
	<p>So, we can conclude that data normalization and standardization help to alleviate these problems by putting the input data into a consistent and manageable range, allowing the MLP to learn more effectively and efficiently.</p>			

## 2. Housing market predictions

Neural networks can be used in housing market predictions. The data `'houses_1.csv'` has 21 columns, 20 of which are house specifications, construction date, pricing, and etc. is dedicated and one column (the third column) shows the price of the house. Using Google Colab, we want to train an MLP neural network that can accurately predict house prices. In this problem, we will use the following libraries shown in Table 1.

Table 1. Libraries used in section 2.

No.	Library title
1	pandas
2	numpy
3	matplotlib
4	scikit-learn
5	seaborn
6	warnings
7	tensorflow
8	google.colab

Where:

1. Pandas: A powerful library for data manipulation and analysis, especially for structured data.
2. NumPy: Fundamental library for numerical computing in Python, providing support for large arrays and mathematical functions.
3. Matplotlib: Comprehensive plotting library for creating high-quality visualizations in Python.
4. Scikit-learn: Versatile machine learning library with a wide range of algorithms and tools for model training and evaluation.
5. Seaborn: Statistical data visualization library built on top of Matplotlib, simplifying the creation of attractive plots.
6. Warnings: Python module for handling warning messages raised during program execution.
7. TensorFlow: Open-source library for numerical computation and machine learning, particularly deep learning models.
8. Google Colab: Online platform by Google for running Python code in a Jupyter Notebook environment, with access to cloud-based resources.

11	Masoud Pourghavam 810601044	Artificial Intelligence Dr. M. S. Panahi	HW4
----	--------------------------------	---------------------------------------------	-----

### A. Import the data

We first, import the pandas library using import pandas as pd. Then, we read the CSV file into a pandas DataFrame using pd.read\_csv(file\_dir), where file\_dir is the directory path of the file. The commands are as follows:

```
drive.mount('/content/drive')
file_dir = "/content/drive/MyDrive/houses_1.csv"
df = pd.read_csv(file_dir)
```

#### i. Columns and rows report

To determine the number of rows and columns in the dataset, we need to check the shape of the DataFrame as follows:

```
num_rows, num_cols = df.shape
print("Number of rows:", num_rows)
print("Number of columns:", num_cols)
```

After executing this code, it will display the number of rows and columns in the dataset. The 'shape' attribute of a DataFrame returns a tuple containing the number of rows and columns respectively, which we assign to num\_rows and num\_cols variables. Then we print these values to the console. From the results, we can see that we have **21613 rows** and **21 columns** in our dataset.

#### ii. Correlation matrix

After reading the file into the DataFrame 'df', we can calculate the correlation matrix using the '.corr()' method on the DataFrame. This method computes the pairwise correlation between all the columns of the DataFrame.

```
correlation_matrix = df.corr()
```

To plot the correlation matrix as a heatmap, we can use the seaborn library in conjunction with the matplotlib library as follows:

```
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f",
linewidths=0.5)
plt.title("Correlation Matrix")
plt.show()
```

In this code, we create a figure with a specific size using `plt.figure(figsize=(12, 10))`. The `figsize` parameter determines the width and height of the figure in inches. Then, we use `sns.heatmap()` to create the heatmap plot. We pass the correlation matrix to this function and set `annot=True` to display the correlation values on the heatmap. The `cmap` parameter is set to "coolwarm" to specify the color map, and `fmt=".2f"` sets the format of the correlation values to two decimal places. The `linewidths` parameter controls the width of the lines between cells in the heatmap. Finally, we set the title of the plot using `plt.title("Correlation Matrix")` and display the plot using `plt.show()`. After running the code, the correlation matrix is represented as we can see in Figure 2.

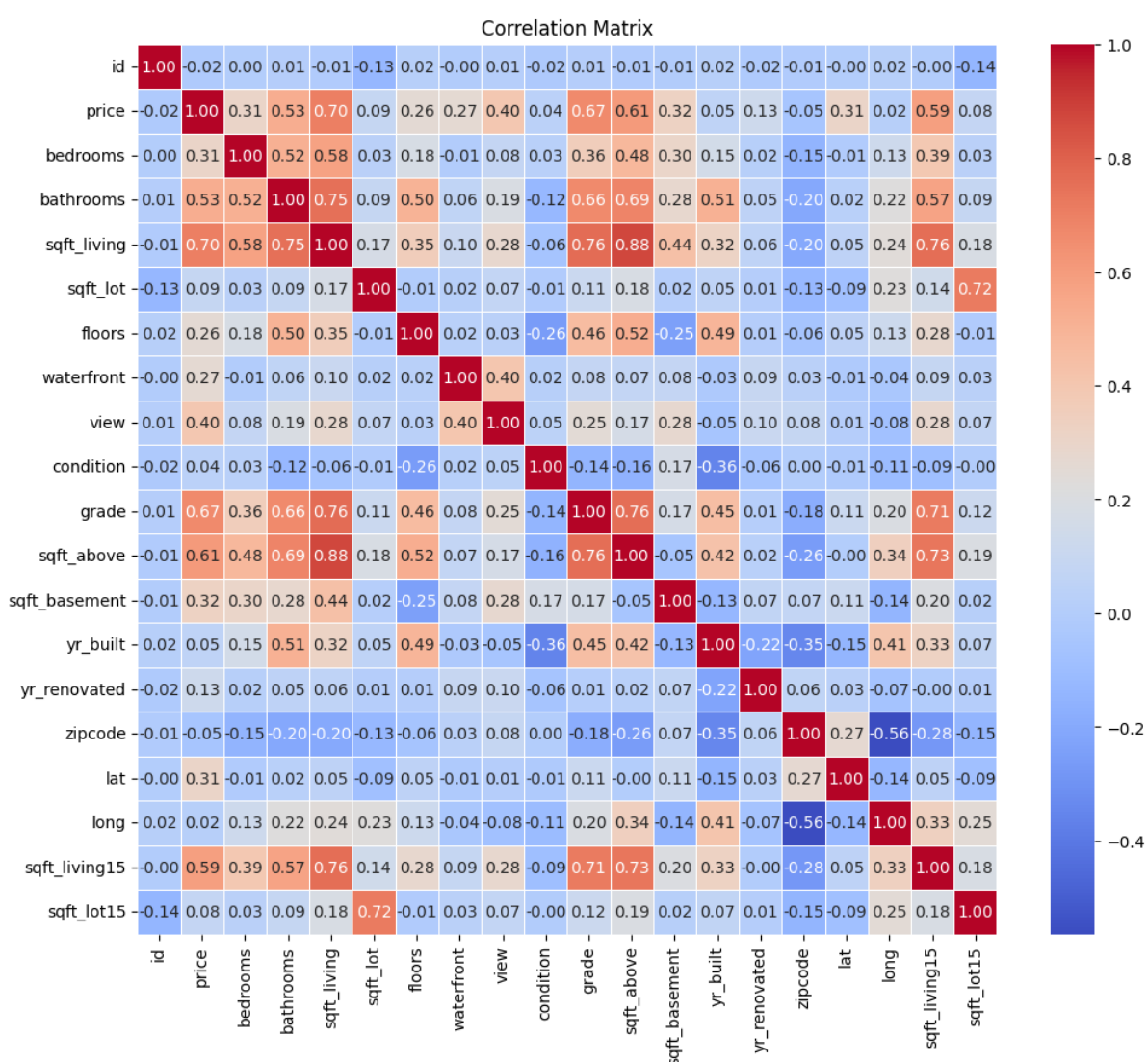


Fig 2. Correlation matrix

In the above correlation matrix, the numbers represent the correlation coefficients between pairs of features. Correlation coefficients measure the strength and direction of the linear relationship between two variables. The correlation coefficient ranges from -1 to 1, with 0

indicating no linear relationship. Here's what different values of correlation coefficients represent:

A correlation coefficient between 0 and 1 indicates a positive correlation. This means that as one feature increases, the other feature tends to increase as well. The closer the coefficient is to 1, the stronger the positive correlation. A correlation coefficient between 0 and -1 indicates a negative correlation. This means that as one feature increases, the other feature tends to decrease. The closer the coefficient is to -1, the stronger the negative correlation. A correlation coefficient of 0 indicates no linear relationship between the two features. This means that there is no consistent pattern of change between the variables.

In the following Table, we show the first 4 features with the strongest correlation with price:

*Table 2. Strongest correlations with price*

Feature	Correlation value
Price	1.00
<b>Sqft_living</b>	<b>0.70</b>
grade	0.67
Sqft_above	0.61

We can see that after the price itself, the sqft\_living feature has the strongest correlation with price.

### iii. House price distribution

We use the following line to create a new figure object with a specified size. The figsize parameter takes a tuple with two values: the width and height of the figure in inches. In this case, the figure will have a width of 10 inches and a height of 6 inches.

```
plt.figure(figsize=(10,6))
```

Then, we will use the following code which uses Seaborn's distplot function to create a distribution plot of the 'price' column from the DataFrame df. The distplot function combines a histogram and a kernel density estimate (KDE) plot to visualize the distribution of the data. It will display the frequency distribution of the 'price' values and an estimated probability density function. Finally, the distribution of house price is given in Figure 3.

```
sns.distplot(df['price'])
```

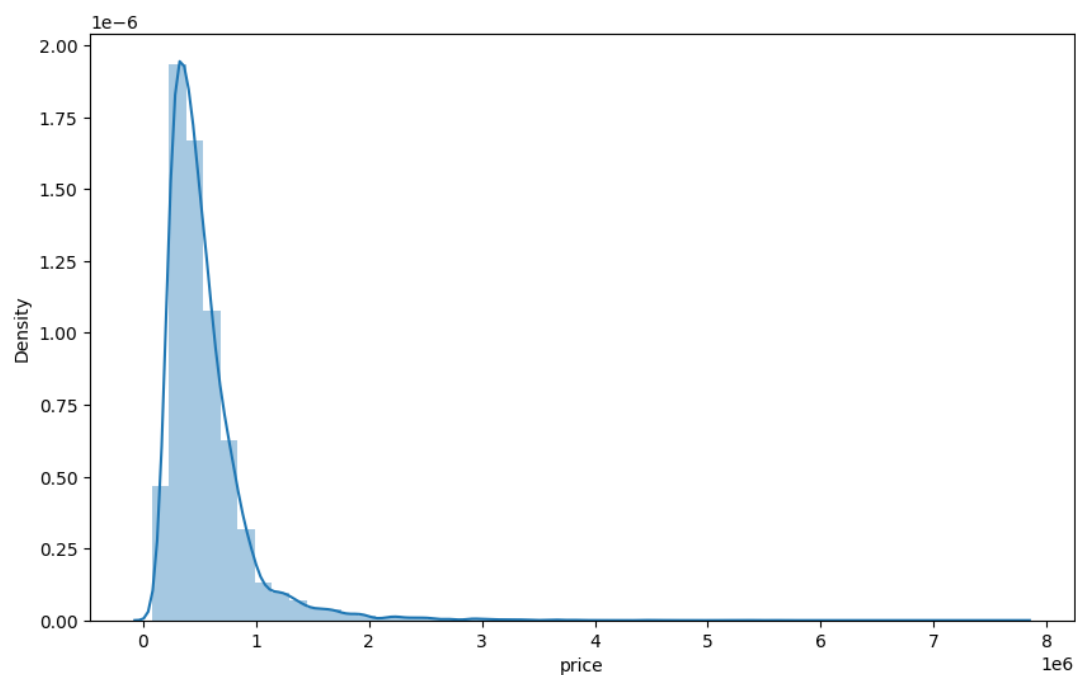


Fig 3. Distribution of house price

## B. Data preprocessing

### i. Date column

To transform the 'date' column into 'year' and 'month' columns and delete the 'date' column from the dataset, we can use pandas' datetime functionality. Here's the code to perform these operations:

```
# Transform 'date' column to datetime format
df['date'] = pd.to_datetime(df['date'])

# Extract year and month from the 'date' column
df['year'] = df['date'].dt.year
df['month'] = df['date'].dt.month

# Delete the 'date' column
df = df.drop('date', axis=1)
```

In this code, we use `pd.to_datetime()` to convert the 'date' column to datetime format. Then, we extract the year and month from the 'date' column using the `.dt.year` and `.dt.month` accessor functions, respectively. This creates two new columns, 'year' and 'month', containing the corresponding values from the 'date' column. Afterward, we delete the 'date' column using the `.drop()` function with `axis=1` to specify that we want to drop a column.

Finally, we print the updated dataset using `print(df.head())` to verify the changes as we can see through the following code. The results are given in Figure 4.

```
# Print the updated dataset
print(df.head())
```



	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	\
0	7129300520	221900.0	3	1.00	1180	5650	1.0	
1	6414100192	538000.0	3	2.25	2570	7242	2.0	
2	5631500400	180000.0	2	1.00	770	10000	1.0	
3	2487200875	604000.0	4	3.00	1960	5000	1.0	
4	1954400510	510000.0	3	2.00	1680	8080	1.0	

	waterfront	view	condition	...	sqft_basement	yr_built	yr_renovated	\
0	0	0	3	...	0	1955	0	
1	0	0	3	...	400	1951	1991	
2	0	0	3	...	0	1933	0	
3	0	0	5	...	910	1965	0	
4	0	0	3	...	0	1987	0	

	zipcode	lat	long	sqft_living15	sqft_lot15	year	month
0	98178	47.5112	-122.257	1340	5650	2014	10
1	98125	47.7210	-122.319	1690	7639	2014	12
2	98028	47.7379	-122.233	2720	8062	2015	2
3	98136	47.5208	-122.393	1360	5000	2014	12
4	98074	47.6168	-122.045	1800	7503	2015	2

[5 rows x 22 columns]

Fig 4. Date column transformation

## ii. Input and outputs

To consider the 'price' column as the output column (y) and all the other columns as the input columns (X), we can separate the columns accordingly in pandas as follows:

```
# Separate input (X) and output (y) columns
X = df.drop('price', axis=1) # Drop 'price' column from the DataFrame
to get input columns
y = df['price'] # Select 'price' column as the output column
# Print the input (X) and output (y) shapes
print("Input (X) shape:", X.shape)
print("Output (y) shape:", y.shape)
```

To create the input (X) DataFrame, we use `df.drop('price', axis=1)`, which drops the 'price' column from the original DataFrame, leaving all the other columns as the input features. For the output (y), we select the 'price' column by accessing it as `df['price']`.

Finally, we print the shapes of the input (X) and output (y) using `print("Input (X) shape:", X.shape)` and `print("Output (y) shape:", y.shape)` to verify the dimensions. The input (X) shape will have the number of rows (21613) and the number of columns (20), while the output (y) shape will have the number of rows (21613) and a single column representing the 'price' values.

To report the highest and lowest prices from the 'price' column in the dataset, we can use the `max()` and `min()` functions on the 'price' column as follows:

```
# Find the highest and lowest prices
highest_price = df['price'].max()
lowest_price = df['price'].min()
# Print the highest and lowest prices
print("Highest Price:", highest_price)
print("Lowest Price:", lowest_price)
```

In this code, we use `df['price'].max()` to find the highest price in the 'price' column and `df['price'].min()` to find the lowest price. We assign the highest price value to the variable `highest_price` and the lowest price value to the variable `lowest_price`. Finally, we print the highest and lowest prices using `print("Highest Price:", highest_price)` and `print("Lowest Price:", lowest_price)`.

From the results, we can see that the **highest price is 7700000** and the **lowest price is 75000**.

### iii. Training and testing sets

To split the data into training (80%) and testing (20%) sets, we can use the `train_test_split()` function from the `sklearn.model_selection` module as follows:

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

In above code, we use `train_test_split()` to split the data into training and testing sets. We pass the input (X) and output (y) data, along with the `test_size` parameter set to 0.2 to indicate that we want a 20% testing set. The `random_state` parameter is set to 42 for reproducibility.

The `train_test_split()` function returns four sets of data: `X_train` (training input), `X_test` (testing input), `y_train` (training output), and `y_test` (testing output).

## iv. Scale

To scale the input data using MinMaxScaler while avoiding data leakage by not using the testing set in the scaling process, we can apply the scaling only on the training set as follows:

```
# Perform scaling only on the training set
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)

# Apply the same scaling on the testing set
X_test_scaled = scaler.transform(X_test)

# Print the shapes of the scaled training and testing sets
print("X_train_scaled shape:", X_train_scaled.shape)
print("X_test_scaled shape:", X_test_scaled.shape)
```

In this code, we create an instance of MinMaxScaler(), which will be used for scaling. We fit and transform the training set X\_train using scaler.fit\_transform(X\_train). This performs the scaling based on the range of the training set features.

To ensure consistency, we then apply the same scaling transformation on the testing set X\_test using scaler.transform(X\_test). It applies the scaling based on the parameters learned from the training set. When we run this code, it will scale the training set using MinMaxScaler and apply the same scaling on the testing set while avoiding data leakage.

**Data leakage:**

Data leakage refers to a situation in which information from outside the training data is inappropriately used to create or evaluate a predictive model. It occurs when there is unintentional or inappropriate inclusion of information in the training process that should not be available at the time of model deployment or prediction.

Data leakage can lead to overly optimistic performance estimates during model development and evaluation, resulting in models that perform poorly in real-world scenarios. It can occur in various ways. Including when using information that would not be available in practice or future data to make predictions during the model training phase. For example, including features that are derived from the target variable or using time-series data in a way that incorporates future information.

The consequences of data leakage can be severe. Models built with data leakage may produce overly optimistic results during testing and fail to generalize well to new, unseen data. This can lead to poor decision-making and unreliable predictions in real-world applications. To avoid data leakage, it is crucial to carefully handle the separation of training and testing data, ensure that preprocessing steps are applied separately to each set, and avoid using any information in the training process that would not be available during prediction or deployment.

### C. Implementation of the model

#### i. Network design

To train an MLP model with the given properties, we can use the Keras library, which provides a high-level API for building and training neural networks.

```
# Build the MLP model
model = Sequential()
model.add(Dense(64, activation='relu',
input_shape=(X_train_scaled.shape[1],))) # Input layer
model.add(Dense(32, activation='relu')) # Hidden layer 1
model.add(Dense(16, activation='relu')) # Hidden layer 2
model.add(Dense(1)) # Output layer

# Compile the model
model.compile(optimizer=SGD(learning_rate=0.001, clipvalue=1.0),
              loss='mean_squared_error')
```

Next, we define the MLP model using the Sequential API from Keras. We add two hidden layers with 64 and 32 neurons, respectively, followed by a ReLU activation function. The final output layer consists of a single neuron, which is suitable for regression tasks. The appropriate cost function for this regression problem is considered mean squared error (MSE). In the following sections, we will change the learning rate to 0.1 and 0.001.

## ii. Network train

We use the following code which trains our neural network model using the fit() function.

```
# Train the model
history = model.fit(X_train_scaled, y_train,
                    batch_size=64,
                    epochs=4000,
                    validation_data=(X_test_scaled, y_test))

# Get the training and validation loss from the history
loss = history.history['loss']
val_loss = history.history['val_loss']
```

In above code, `model.fit(X_train_scaled, y_train, ...)` is calling the `fit()` function on our model. In this case, `X_train_scaled` represents the input features of the training data, and `y_train` represents the corresponding target labels.

The `batch_size = 64` specifies the number of samples that will be propagated through the model at once. It helps in determining the number of gradient updates that will be performed during each epoch. In this case, the batch size is set to 64, which means that the model will update its weights after processing 64 samples.

The `epochs = 4000` determines the number of times the entire training dataset will be passed through the model during training. In this case, the model will be trained for 4000 epochs, meaning it will go through the entire training dataset 4000 times. In the following sections, we will change the number of epochs and train the model with 20 and 4000 epochs.

The `validation_data = (X_test_scaled, y_test)` is used to provide a validation dataset to evaluate the performance of the model during training. The validation dataset consists of `X_test_scaled` (input features) and `y_test` (target labels). The model's performance on the validation data will be computed and printed after each epoch.

## D. Demands

- i. Loss, validation loss,  $R^2$ , MAE, and MSE

Here, after training the model with the given properties, we make predictions on the training and testing data using the following commands:

```
# Make predictions on the training and testing data
y_train_pred = model.predict(X_train_scaled)
y_test_pred = model.predict(X_test_scaled)
```

Then, we will calculate the MAE, MSE, and  $R^2$  matrices for each of the training and testing data as follows:

```
# Calculate metrics for training data
train_mae = mean_absolute_error(y_train, y_train_pred)
train_mse = mean_squared_error(y_train, y_train_pred)
train_r2 = explained_variance_score(y_train, y_train_pred)

# Calculate metrics for testing data
test_mae = mean_absolute_error(y_test, y_test_pred)
test_mse = mean_squared_error(y_test, y_test_pred)
test_r2 = explained_variance_score(y_test, y_test_pred)
```

Finally, we plot and print the results (for instance, for training set):

```
# Visualize the scatter plot of original and predicted values
plt.scatter(y_train, y_train_pred, alpha=0.5)
plt.plot(y_train, y_train, 'r')
plt.title('y_train for lr=0.001 and epoch=4000')
plt.xlabel('Original Values')
plt.ylabel('Predicted Values')
plt.show()

# Print the metrics
print("Train Metrics:")
print("MAE:", train_mae)
print("MSE:", train_mse)
print("R^2:", train_r2)
```

- $Lr = 0.001$  and Epochs = 20

In this case, we use a learning rate of 0.001 and 20 epochs to train our MLP model. The results are given in Figures 5 to 7 and Tables 3 and 4.

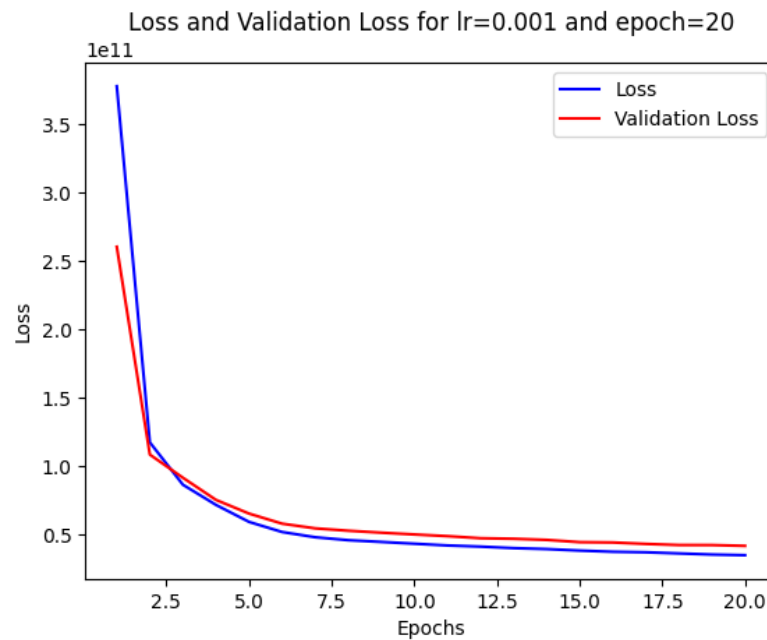


Fig 5. Loss and validation loss for  $Lr = 0.001$  and 20 epochs

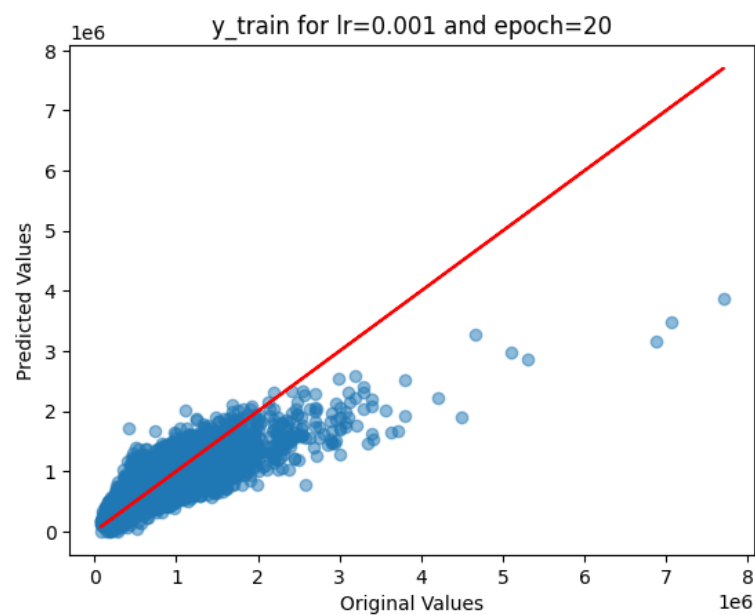
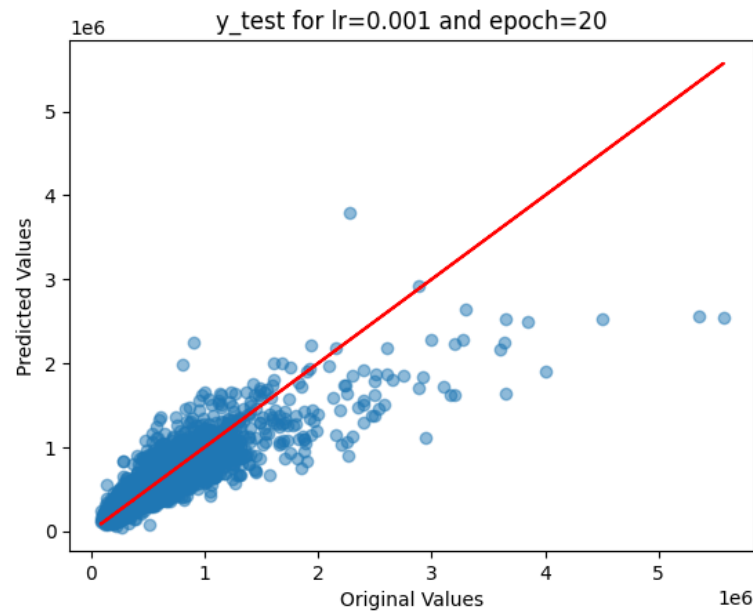


Fig 6. Accuracy of training set predictions for  $Lr = 0.001$  and 20 epochs

Table 3. Train matrices for  $lr = 0.001$  and 20 epochs

Train Matrices	Value
Mean Absolute Error	113102.02
Mean Squared Error	34861301125.05
$R^2$	0.73

Fig 7. Accuracy of testing set predictions for  $lr = 0.001$  and 20 epochsTable 4. Test matrices for  $lr = 0.001$  and 20 epochs

Test Matrices	Value
Mean Absolute Error	117358.24
Mean Squared Error	41633664292.90
$R^2$	0.72

The MAE measures the average magnitude of the errors between the predicted and actual values. In this case, the MAE for the train set is 113102.02 and for the test set is 117358.24. The lower the MAE, the better the performance. However, it's difficult to determine the absolute quality of the MAE without context on the specific problem or domain. The MSE calculates the average squared difference between the predicted and actual values. A lower MSE indicates a better model fit. In this case, the MSE for the train set is 34861301125.05 and for the test set is 41633664292.90.



24	Masoud Pourghavam 810601044	Artificial Intelligence Dr. M. S. Panahi	HW4	<p>The R-squared score measures how well the predicted values explain the variance in the actual values. It ranges from 0 to 1, where 1 indicates a perfect fit. In this case, the <math>R^2</math> score for the train set is 0.73, and for the test set, it is 0.72. Considering the results, the model seems to have performed reasonably well. The MAE and MSE values are relatively high, indicating that the model's predictions have a significant average deviation from the actual values. However, without further context about the specific problem, it's challenging to determine the significance of these errors. The <math>R^2</math> score of 0.73 for the train set and 0.72 for the test set suggests that the model explains approximately 73% and 72% of the variance in the respective datasets. While these values indicate a decent level of predictive power, there is still room for improvement.</p> <p>Additionally, from Figure 5, we can see that the loss and validation loss closely match each other with a little bias and it suggests that the model is not overfitting or underfitting the data significantly. This alignment indicates that the model's performance on the validation set is comparable to its performance on the training set.</p> <p>In summary, while the model's performance is acceptable, there is potential for improvement, particularly in reducing the MAE and MSE values and increasing the <math>R^2</math> score. Further analysis, experimentation, and fine-tuning of the model's hyperparameters might help enhance its performance.</p>
----	--------------------------------	---------------------------------------------	-----	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

- $Lr = 0.1$  and Epochs = 20

In this case, we use a learning rate of 0.1 and 20 epochs to train our MLP model. The results are given in Figures 8 to 10 and Tables 5 and 6.

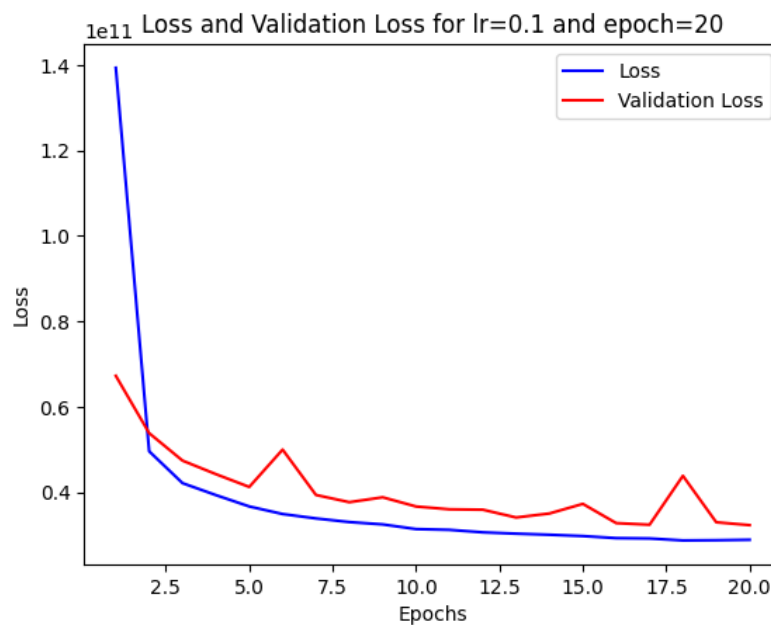


Fig 8. Loss and validation loss for  $Lr = 0.1$  and 20 epochs

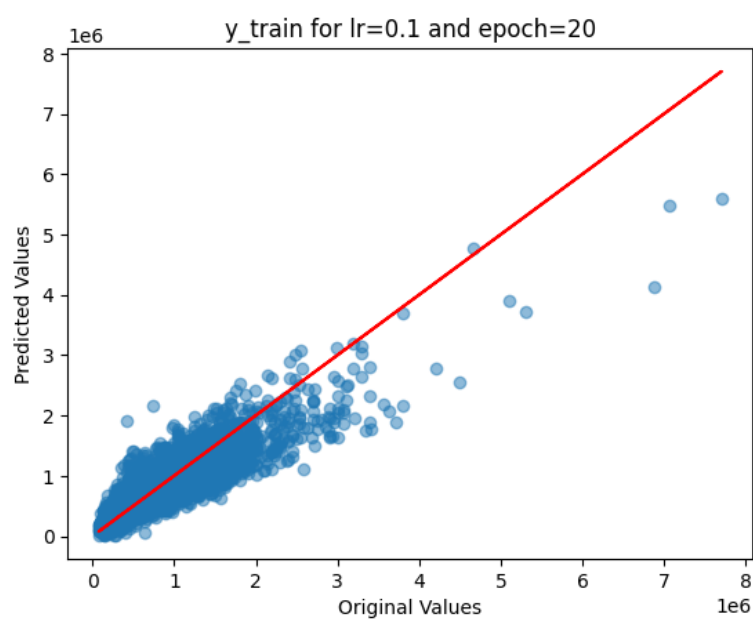


Fig 9. Accuracy of training set predictions for  $Lr = 0.1$  and 20 epochs

Table 5. Train matrices for  $lr = 0.1$  and 20 epochs

Train Matrices	Value
Mean Absolute Error	116190.19
Mean Squared Error	29758919381.73
$R^2$	0.79

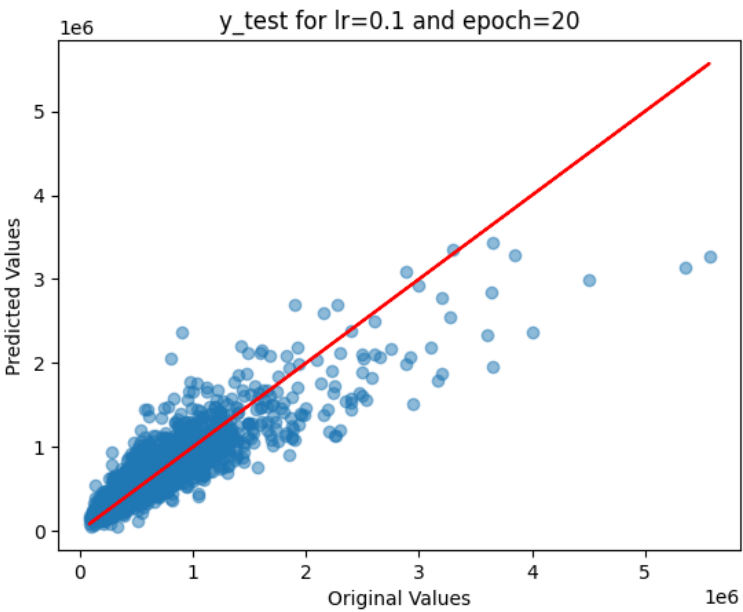


Fig 10. Accuracy of testing set predictions for  $lr = 0.1$  and 20 epochs

Table 6. Test matrices for  $lr = 0.1$  and 20 epochs

Test Matrices	Value
Mean Absolute Error	118670.85
Mean Squared Error	34325054175.92
$R^2$	0.78

In this case, the MAE for the train set is 116190.19, and for the test set is 118670.85. Lower MAE values indicate better performance, but without further context on the specific problem or domain, it's difficult to assess the absolute quality of the MAE. In this case, the MSE for the train set is 29758919381.73, and for the test set is 34325054175.92. In this case, the  $R^2$  score for the train set is 0.79, and for the test set, it is 0.78.

Based on these results, the model appears to have performed reasonably well. The MAE and MSE values are relatively high, suggesting a significant average deviation between the

27	Masoud Pourghavam 810601044	Artificial Intelligence Dr. M. S. Panahi	HW4	<p>predicted and actual values. However, without further context, it's challenging to determine the significance of these errors. The <math>R^2</math> score of 0.79 for the train set and 0.78 for the test set indicates that the model explains approximately 79% and 78% of the variance in the respective datasets. These values suggest a decent level of predictive power, but there is still room for improvement.</p> <p>From Figure 8, we can see that the loss and validation loss are relatively close to each other with a bias. This suggests that the model is not significantly overfitting or underfitting the data. However, the presence of two overshoots in the validation loss curve indicates that the model might have experienced some instability or difficulties during training. This could be an area to investigate further to improve the model's performance.</p> <p>In summary, while the model's performance is acceptable, there is potential for improvement, particularly in reducing the MAE and MSE values and further enhancing the <math>R^2</math> score. Further analysis, experimentation, and fine-tuning of the model's hyperparameters might help improve its performance. Additionally, investigating the causes of the overshoots in the validation loss curve could lead to more stable training and better generalization to unseen data.</p>
----	--------------------------------	---------------------------------------------	-----	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

- $Lr = 0.001$  and Epochs = 4000

In this case, we use a learning rate of 0.001 and 4000 epochs to train our MLP model. The results are given in Figures 11 to 13 and Tables 7 and 8.

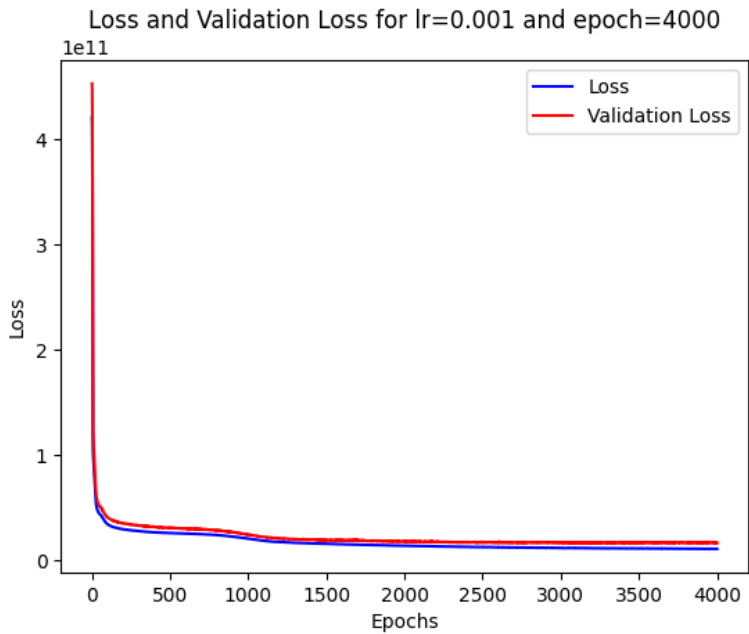


Fig 11. Loss and validation loss for  $lr = 0.001$  and 4000 epochs

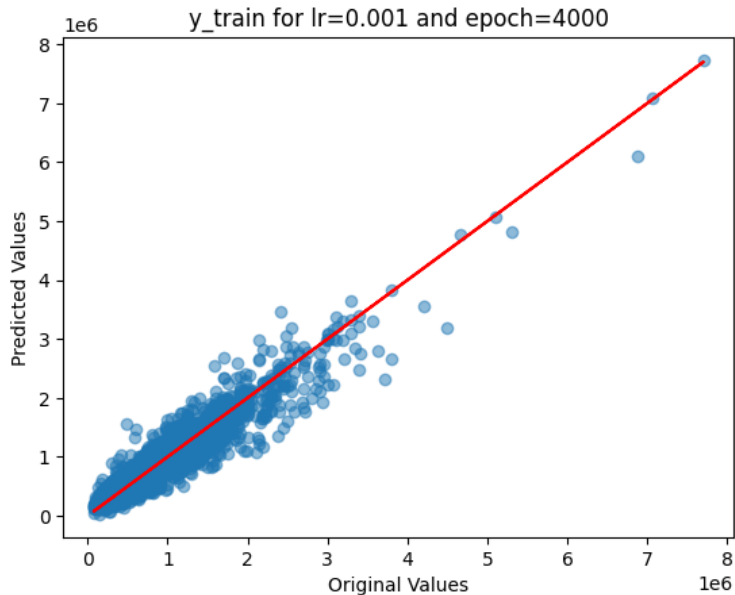


Fig 12. Accuracy of training set predictions for  $lr = 0.001$  and 4000 epochs

Table 7. Train matrices for  $lr = 0.001$  and 4000 epochs

Train Matrices	Value
Mean Absolute Error	62383.79
Mean Squared Error	10455187314.86
R2	0.92

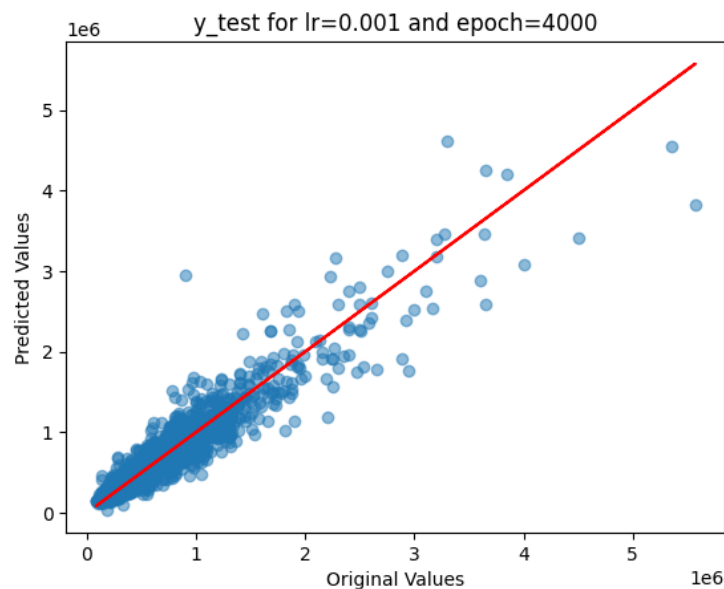


Fig 13. Accuracy of testing set predictions for  $lr = 0.001$  and 4000 epochs

Table 8. Test matrices for  $lr = 0.001$  and 4000 epochs

Test Matrices	Value
Mean Absolute Error	71694.48
Mean Squared Error	16273037650.03
R2	0.89

In this case, the MAE for the train set is 62383.79, and for the test set is 71694.48. Lower MAE values indicate better performance, but without further context on the specific problem or domain, it's challenging to assess the absolute quality of the MAE. In this case, the MSE for the train set is 10455187314.86, and for the test set is 16273037650.03. In this case, the  $R^2$  score for the train set is 0.92, and for the test set, it is 0.89.

Based on these results, the model seems to have performed well. The MAE and MSE values are relatively low, indicating a smaller average deviation between the predicted and actual values. The  $R^2$  score of 0.92 for the train set and 0.89 for the test set suggests that the model explains approximately 92% and 89% of the variance in the respective datasets. These values indicate a strong predictive power.

30	Masoud Pourghavam 810601044	Artificial Intelligence Dr. M. S. Panahi	HW4	
	<p>From Figure 11, we can see that the loss and validation loss completely match each other in the plot and it suggests that the model is not overfitting or underfitting the data. This alignment indicates that the model's performance on the validation set is comparable to its performance on the training set.</p> <p>In summary, the model's performance appears to be quite good based on the given results. It has achieved low MAE and MSE values, as well as high <math>R^2</math> scores. However, without further context about the specific problem or domain, it's challenging to determine the absolute quality of the model's performance. Nonetheless, these results suggest that the model has learned the underlying patterns in the data and can make reasonably accurate predictions.</p>			

- Lr = 0.1 and Epochs = 4000

In this case, we use a learning rate of 0.1 and 20 epochs to train our MLP model. The results are given in Figures 14 to 16 and Tables 9 and 10.

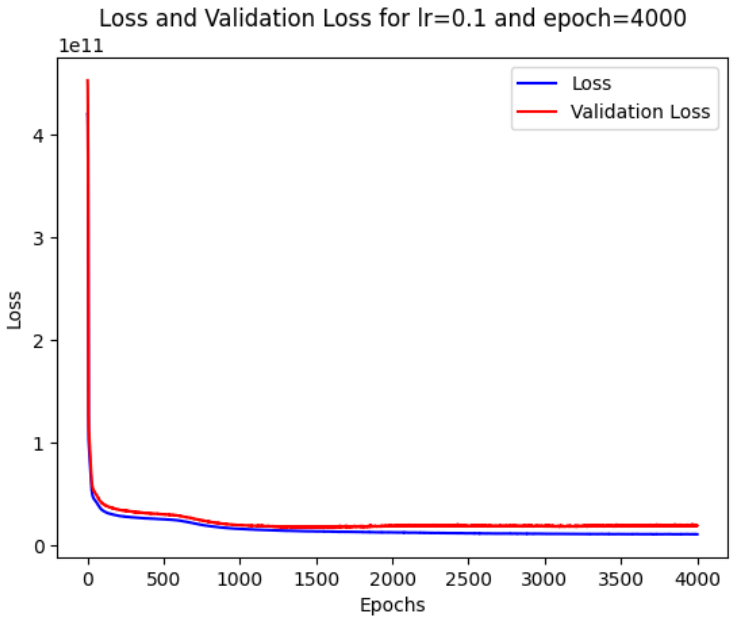


Fig 14. Loss and validation loss for lr = 0.1 and 4000 epochs

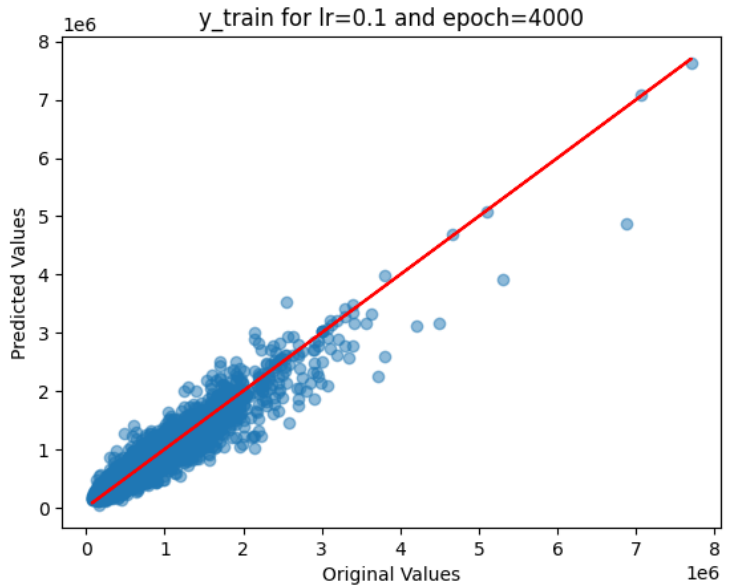


Fig 15. Accuracy of training set predictions for lr = 0.1 and 4000 epochs

Table 9. Train matrices for lr = 0.1 and 4000 epochs



Train Matrices	Value
Mean Absolute Error	63261.70
Mean Squared Error	10914076773.96
$R^2$	0.91

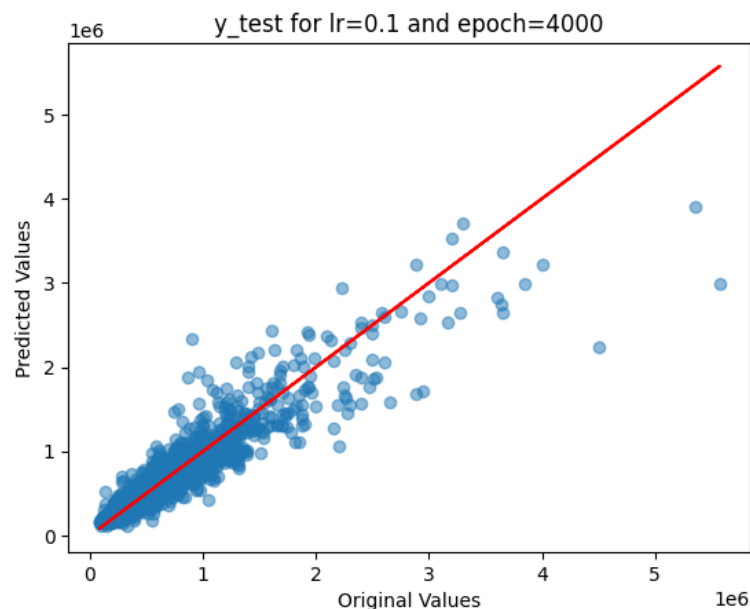


Fig 16. Accuracy of testing set predictions for  $lr = 0.1$  and 4000 epochs

Table 10. Test matrices for  $lr = 0.1$  and 4000 epochs

Test Matrices	Value
Mean Absolute Error	74543.06
Mean Squared Error	19479724859.74
$R^2$	0.87

In this case, the MAE for the train set is 63261.70, and for the test set is 74543.06. Lower MAE values indicate better performance, but without further context on the specific problem or domain, it's challenging to assess the absolute quality of the MAE. In this case, the MSE for the train set is 10914076773.96, and for the test set is 19479724859.74. In this case, the  $R^2$  score for the train set is 0.91, and for the test set, it is 0.87.

Based on these results, the model's performance appears to be decent. The MAE and MSE values are relatively high, suggesting a significant average deviation between the predicted and actual values.

33	Masoud Pourghavam 810601044	Artificial Intelligence Dr. M. S. Panahi	HW4	<p>The <math>R^2</math> score of 0.91 for the train set and 0.87 for the test set indicates that the model explains approximately 91% and 87% of the variance in the respective datasets. While these values indicate a reasonably good level of predictive power, there is still room for improvement.</p> <p>From Figure 14, we can see that the loss and validation loss completely match each other in the plot and it suggests that the model is not overfitting or underfitting the data. This alignment indicates that the model's performance on the validation set is comparable to its performance on the training set.</p> <p>In summary, the model's performance is acceptable based on the given results. However, there is potential for improvement, particularly in reducing the MAE and MSE values and further increasing the <math>R^2</math> score. Further analysis, experimentation, and fine-tuning of the model's hyperparameters might help enhance its performance. Additionally, additional data or feature engineering could also contribute to improving the model's accuracy.</p>
----	--------------------------------	---------------------------------------------	-----	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

34	Masoud Pourghavam 810601044	Artificial Intelligence Dr. M. S. Panahi	HW4	
	<p>ii. Epochs</p> <p>The initial training was performed with 20 epochs, while the second training was performed with 4000 epochs. Let's analyze the impact of these different numbers of epochs on the model's training:</p> <ul style="list-style-type: none"> <li>20 Epochs: With only 20 epochs, the model was trained for a relatively small number of iterations. This limited training duration might not have been sufficient for the model to converge fully or reach its optimal performance. Consequently, the model might not have learned all the underlying patterns in the data, resulting in higher errors and lower predictive power.</li> <li>4000 Epochs: Training the model for 4000 epochs indicates a significantly longer training duration. This extended training period allowed the model to undergo a larger number of iterations, potentially enabling it to learn more complex patterns in the data. As a result, the model's performance likely improved compared to the 20-epoch training. The lower errors and higher <math>R^2</math> scores obtained suggest that the model has achieved better accuracy and predictive power.</li> </ul> <p>In general, the number of epochs impacts how long the model is trained and how many times it updates its weights based on the training data. A small number of epochs may lead to underfitting, where the model fails to capture the full complexity of the data. On the other hand, an excessively large number of epochs can result in overfitting, where the model becomes too specialized to the training data and performs poorly on unseen data.</p> <p>In this case, while 20 epochs may not have been enough for the model to fully learn the underlying patterns, 4000 epochs provided a more extensive training duration, allowing the model to converge to a better solution. However, it's important to note that determining the optimal number of epochs requires careful experimentation and validation, as it can vary depending on the dataset and problem at hand. It is advisable to monitor the model's performance on validation or test data and consider early stopping strategies to prevent overfitting.</p> <p>To determine the optimal number of epochs, we consider the Figure 11 again:</p>			

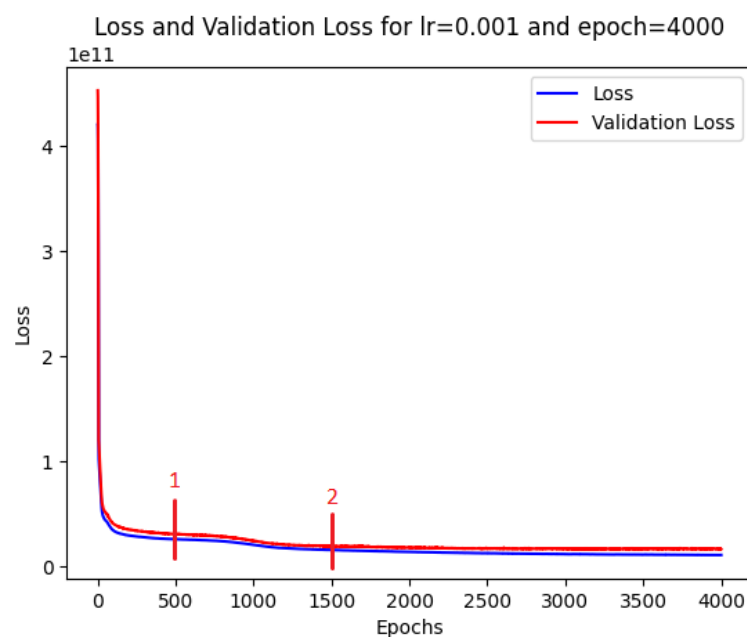


Fig 17. Optimal number of epochs

Determining the optimal number of epochs for training a neural network is a task that typically requires experimentation and validation. There is no fixed or universal number of epochs that can be considered optimal for all scenarios. The optimal number of epochs depends on various factors, including the complexity of the problem, the size of the dataset, the architecture of the neural network, and the chosen hyperparameters.

But, just visually, we can see from Figure 17 that 500 epochs are an appropriate number because in this region, the loss has almost become a constant number.

But, if we want better accuracy, we can consider 1500 epochs because, after the 500 epochs, the loss still decreases until reaches 1500 epochs. We can't see a significant decrease in loss after 1500 epochs.

36	Masoud Pourghavam 810601044	Artificial Intelligence Dr. M. S. Panahi	HW4	
	<p>iii. Learning rate</p> <p>In the first scenario, the learning rate was set to 0.001. The results obtained after training the model with this learning rate showed relatively high Mean Absolute Error (MAE) and Mean Squared Error (MSE) values, as well as lower R-squared (<math>R^2</math>) scores. This suggests that the model might have experienced slow convergence or difficulty in finding the optimal solution. The model's performance was acceptable but had room for improvement. In the second scenario, the learning rate was increased to 0.1. This higher learning rate could lead to larger updates of the model's parameters during each iteration. The results showed improvements in the model's performance, as indicated by lower MAE and MSE values, as well as higher <math>R^2</math> scores compared to the first scenario. These improvements suggest that the model converged more effectively and achieved better accuracy in capturing the underlying patterns in the data.</p> <p>The learning rate is a crucial hyperparameter, and selecting an appropriate value is essential for successful model training. A learning rate that is too low might cause slow convergence, leading to a longer training time or getting stuck in suboptimal solutions. On the other hand, a learning rate that is too high can lead to overshooting and instability, where the model oscillates or fails to converge to the optimal solution.</p> <p>In summary, the learning rate significantly influences the model's convergence and performance. A higher learning rate can lead to faster convergence and improved performance, but selecting an optimal learning rate requires experimentation and consideration of the specific problem and dataset characteristics.</p>			

## iv. Tanh activation function

We use the following code to normalize the output labels ( $y_{\text{train}}$  and  $y_{\text{test}}$ ) to the range of the hyperbolic tangent (Tanh) function, which is -1 to 1:

```
# Normalize the output labels to the range of Tanh function (-1 to 1)
y_train_normalized = (y_train - y_train.min()) / (y_train.max() -
y_train.min()) * 2 - 1
y_test_normalized = (y_test - y_test.min()) / (y_test.max() -
y_test.min()) * 2 - 1
```

Then, we use the network structure with Tanh:

```
# Build the MLP model
model = Sequential()
model.add(Dense(64, activation='tanh',
input_shape=(X_train_scaled.shape[1],))) # Input layer
model.add(Dense(32, activation='tanh')) # Hidden layer 1
model.add(Dense(16, activation='tanh')) # Hidden layer 2
model.add(Dense(1)) # Output layer

# Compile the model
model.compile(optimizer=SGD(learning_rate=0.001, clipvalue=-5),
loss='mean_squared_error')
```

In this case, we use the model with the highest accuracy with a learning rate of 0.001 and 4000 epochs to train our MLP model. The results are given in Figures 18 to 20 and Tables 11 and 12.

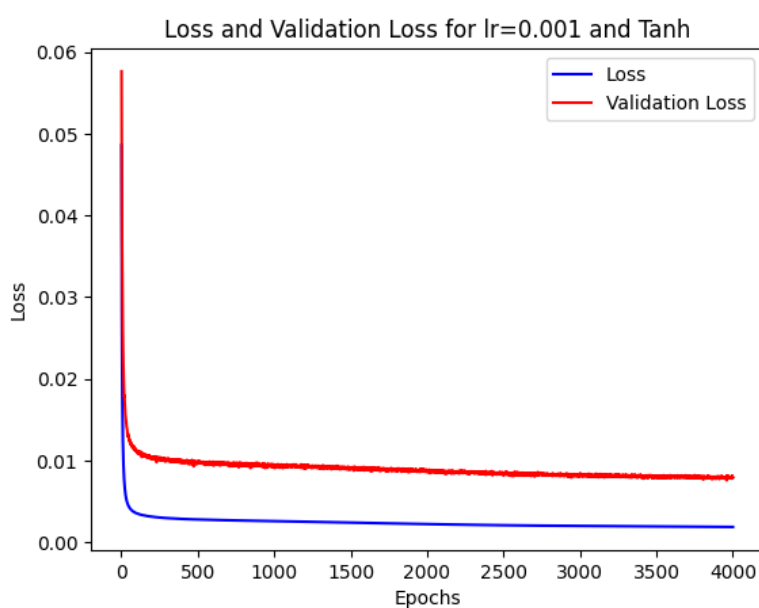


Fig 18. Loss and validation loss for  $lr = 0.001$  and 4000 epochs with Tanh

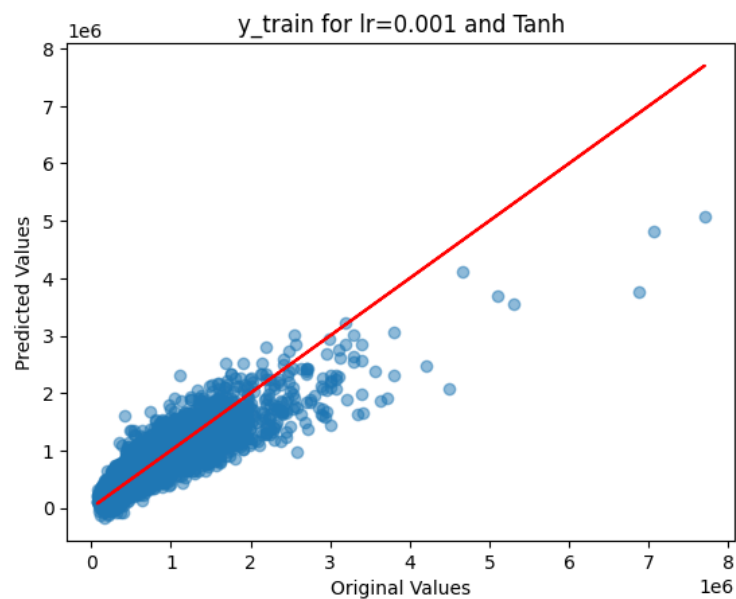


Fig 19. Accuracy of training set predictions for  $lr = 0.001$  with Tanh

Table 11. Train matrices for  $lr = 0.1$  and 4000 epochs

Train Matrices	Value
Mean Absolute Error	104894.16
Mean Squared Error	27242452758.56
$R^2$	0.79

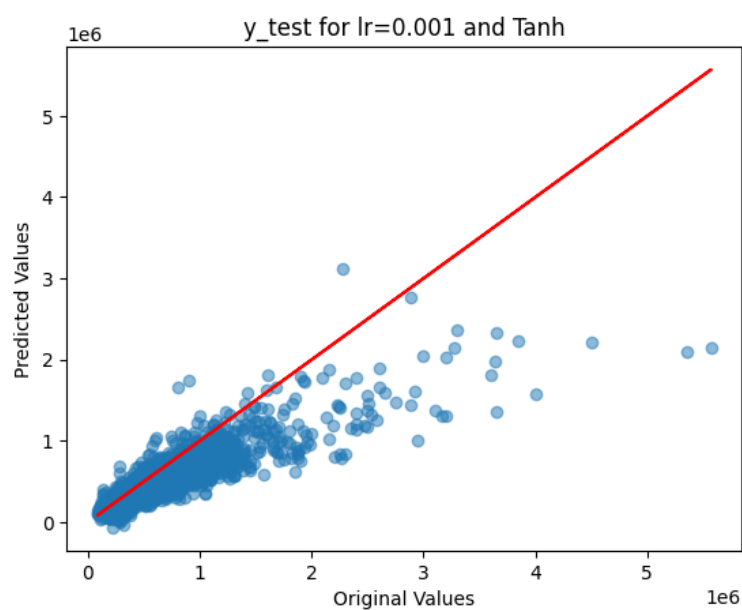


Fig 20. Accuracy of testing set predictions for  $lr = 0.001$  with Tanh

Table 12. Test matrices for  $lr = 0.1$  and 4000 epochs

Test Matrices	Value
Mean Absolute Error	146222.39
Mean Squared Error	59591437382.92
$R^2$	0.71

The model's performance on the training set is as follows:

The MAE of 104,894.16 indicates that, on average, the model's predictions deviate from the actual values by approximately \$104,894.16. The MSE of 27,242,452,758.56 represents the average squared difference between the predicted and actual values. The R2 score of 0.79 suggests that the model explains around 79% of the variance in the dependent variable, indicating a reasonably good fit.

The model's performance on the test set is slightly worse:

The MAE of 146,222.39 indicates that, on average, the model's predictions deviate from the actual values by approximately \$146,222.39, which is higher than the MAE on the training set. The MSE of 59,591,437,382.92 represents a higher average squared difference between the predicted and actual values compared to the training set. The R2 score of 0.71 suggests that the model explains around 71% of the variance in the dependent variable for the test set, indicating a slightly weaker fit compared to the training set.

We can conclude that the model performs reasonably well on the training set, with a relatively low MAE and MSE and a decent R2 score. However, its performance on the test set is slightly worse, indicating a higher level of error and lower explanatory power. It may be worth considering further optimization or exploring alternative activation functions to improve the model's performance on unseen data.



v. Batch size

Batch size = 1:

In this case, we consider the model with the highest accuracy and change the batch size to 1. The results of training with a learning rate of 0.001 and 20 epochs are given in Figures 21 to 23 and Tables 13 and 14.

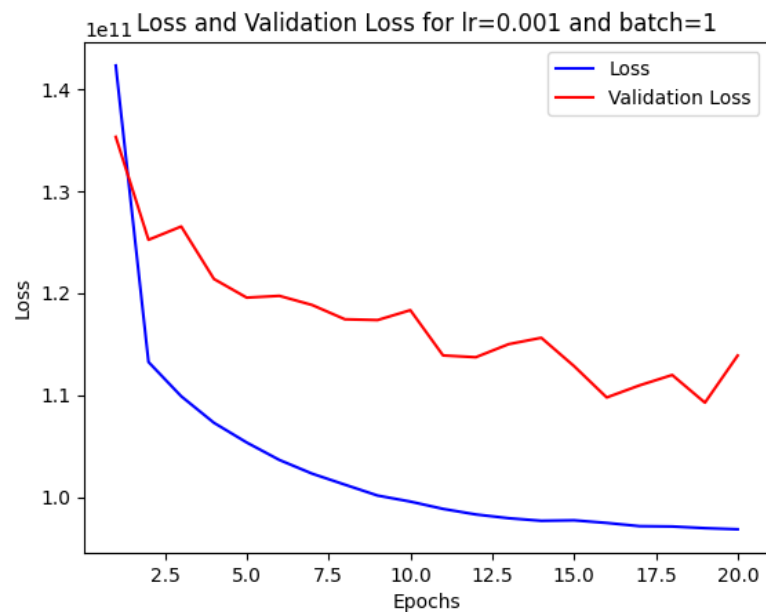


Fig 21. Loss and validation loss for  $lr = 0.001$  and  $batch = 1$

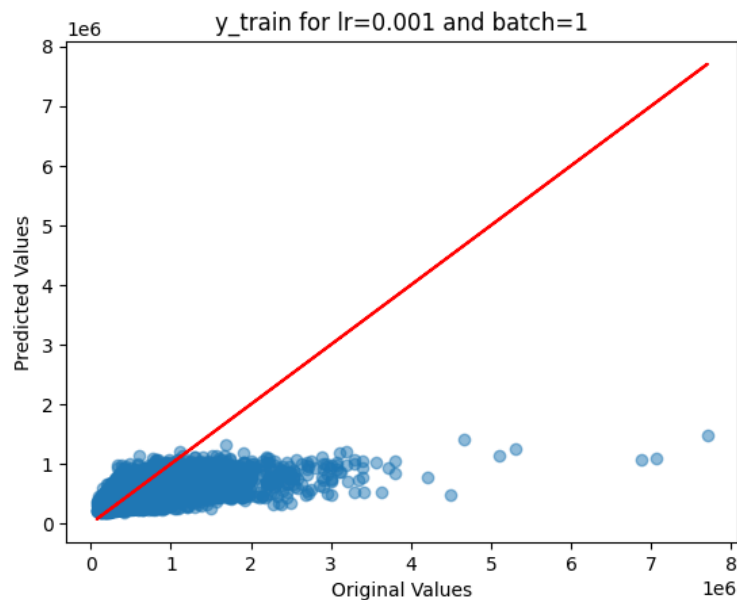


Fig 22. Accuracy of training set predictions for  $lr = 0.001$  and  $batch = 1$

Table 13. Train matrices for lr = 0.001 and batch = 1

Train Matrices	Value
Mean Absolute Error	184074.80
Mean Squared Error	98603905104.81
R <sup>2</sup>	0.28

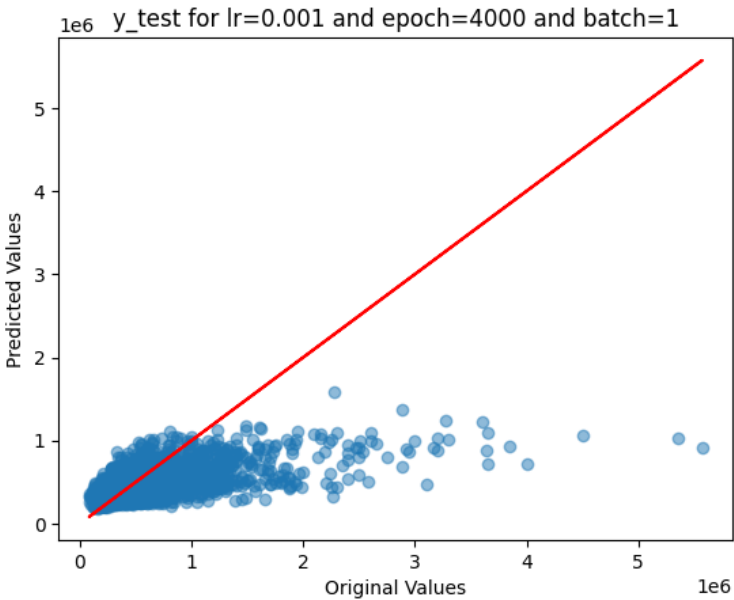


Fig 23. Accuracy of testing set predictions for lr = 0.001 and batch = 1

Table 14. Test matrices for lr = 0.001 and batch = 1

Test Matrices	Value
Mean Absolute Error	190582.08
Mean Squared Error	113867221460.49
R <sup>2</sup>	0.29

First, let's analyze the training metrics:

In our case, the MAE on the training data is 184,074.80. This value indicates that, on average, the predictions of our model deviate by approximately 184,074.80 units from the true values. Our training MSE is 98,603,905,104.81. MSE squares the errors, so it tends to penalize larger errors more than MAE. The MSE value indicates that, on average, the predictions of our model deviate by approximately 98,603,905,104.81 squared units from the true values. An R2 score of 0.28 on the training data suggests that our model explains 28% of the variance in the target variable.

42	Masoud Pourghavam 810601044	Artificial Intelligence Dr. M. S. Panahi	HW4	<p>Now, let's evaluate the performance of the test data:</p> <p>The MAE on the test data is 190,582.08. It indicates that, on average, the predictions of our model deviate by approximately 190,582.08 units from the true values in the test set. The MSE on the test data is 113,867,221,460.49. Similar to the training set, it measures the average squared difference between the predicted and actual values. The MSE value indicates that, on average, the predictions of our model deviate by approximately 113,867,221,460.49 squared units from the true values in the test set. The R2 score on the test data is 0.29. It suggests that our model explains around 29% of the variance in the target variable in the test set.</p> <p>Based on these metrics, we can conclude that both the MAE and MSE values are quite high, indicating that the model's predictions have relatively large errors. This suggests that the model may not be capturing the underlying patterns in the data effectively. The reason is that we used a batch size of 1 and to reduce the processing time, we reduced the number of epochs to 20. This caused the model to underfit and has a limited performance. When we consider a batch size of 1, it means that during training, our model updates its parameters after processing each individual data point. This is known as online or stochastic gradient descent, where the model learns from one data point at a time.</p> <p>Using a batch size of 1 can have both advantages and disadvantages:</p> <p>Advantages:</p> <ul style="list-style-type: none"> <li>• <b>Memory Efficiency:</b> With a batch size of 1, we only need to load and store one data point in memory at a time. This can be beneficial when working with large datasets that do not fit entirely in memory.</li> <li>• <b>Fast Training Updates:</b> Since the model updates its parameters after processing each data point, the updates are more frequent compared to larger batch sizes. This can lead to faster convergence and quicker adjustments to the training data.</li> </ul> <p>Disadvantages:</p> <ul style="list-style-type: none"> <li>• <b>Noisy Gradients:</b> Training with a batch size of 1 can result in noisy gradients because each update is based on a single data point. Noisy gradients can make the training process more unstable and can lead to slower convergence or suboptimal solutions.</li> <li>• <b>Slower Training:</b> While the updates are faster with a batch size of 1, the overall training process can be slower because we need to process and update parameters for each data</li> </ul>
----	--------------------------------	---------------------------------------------	-----	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

43	Masoud Pourghavam 810601044	Artificial Intelligence Dr. M. S. Panahi	HW4	
				<p>point individually. This can result in slower training compared to larger batch sizes, where multiple data points are processed simultaneously.</p> <ul style="list-style-type: none"> <li>• <b>Less Generalization:</b> With a batch size of 1, the model learns from each data point individually without considering the patterns that may exist across multiple data points. This can lead to overfitting, where the model becomes too specific to the training data and fails to generalize well to unseen data.</li> </ul> <p>In our case, using a batch size of 1 might have contributed to the high errors and limited performance of our MLP model. It's worth experimenting with different batch sizes to find the optimal balance between stability, convergence speed, and generalization. Larger batch sizes, such as mini-batch or full-batch training, are often used to provide more stable gradients and better generalization, but they come with increased memory requirements and slower updates.</p>

Batch size = 256:

In this case, we consider the model with the highest accuracy and change the batch size to 256. The results of training with a learning rate of 0.001 and 400 epochs are given in Figures 24 to 26 and Tables 15 and 16.

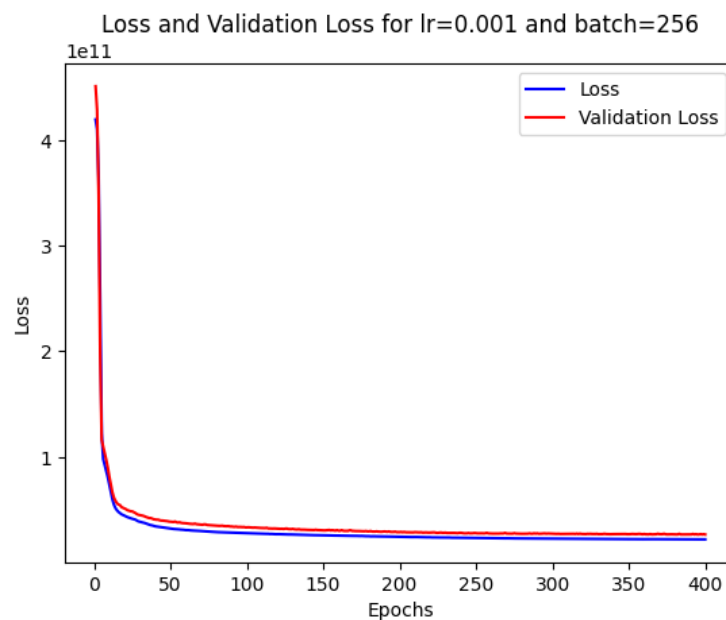


Fig 24. Loss and validation loss for  $lr = 0.001$  and batch = 256

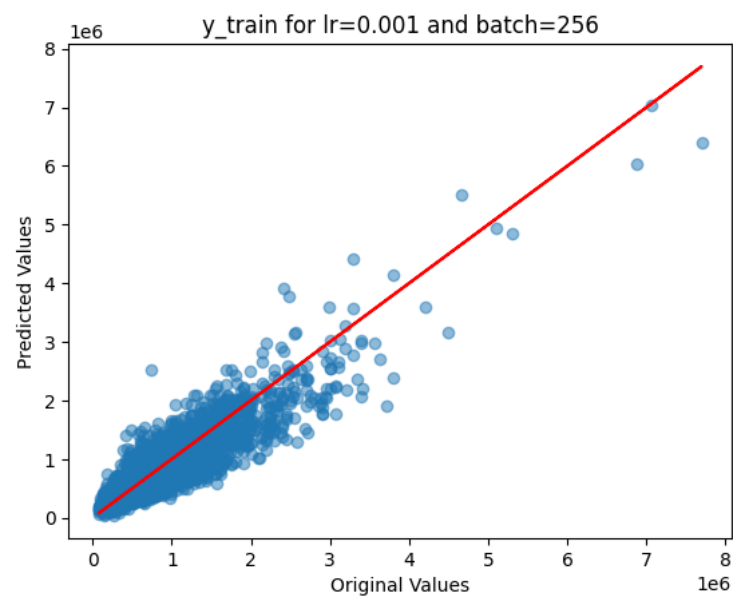


Fig 25. Accuracy of training set predictions for  $lr = 0.001$  and batch = 256

Table 15. Train matrices for  $lr = 0.001$  and batch = 256

Train Matrices	Value
Mean Absolute Error	92945.42
Mean Squared Error	21695383410.14
$R^2$	0.83

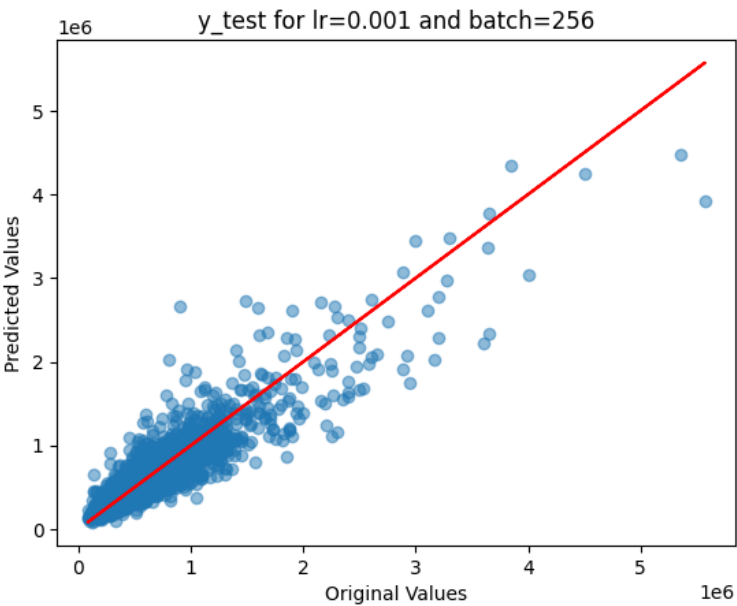


Fig 26. Accuracy of testing set predictions for  $lr = 0.001$  and batch = 256

Table 16. Test matrices for  $lr = 0.001$  and batch = 256

Test Matrices	Value
Mean Absolute Error	99199.22
Mean Squared Error	26539810416.45
$R^2$	0.82

Now, let's evaluate the performance of the test data:

The MAE on the test data is 99,199.22. It indicates that, on average, the predictions of our model deviate by approximately 99,199.22 units from the true values in the test set. The MSE on the test data is 26,539,810,416.45. It measures the average squared difference between the predicted and actual values. The lower MSE value compared to the previous results suggests that our model's predictions are closer to the true values, on average, in the test set. The R2 score on the test data is 0.82. It indicates that our model explains approximately 82% of the variance in the target variable in the test set. This is a good performance, and it shows that our model has a high level of predictive power on unseen data.

46	Masoud Pourghavam 810601044	Artificial Intelligence Dr. M. S. Panahi	HW4	<p>The model's performance has significantly improved compared to the previous results. The MAE, MSE, and R2 scores are all better, indicating that the model has learned more effectively from the data. The MAE and MSE values on the test data are slightly higher than those on the training data, which is expected. However, the difference is not substantial, suggesting that our model generalizes well to unseen data. The R2 scores on both the training and test data are high, indicating that our model captures a large portion of the variance in the target variable and performs well in terms of explaining the data.</p> <p>Overall, based on these evaluation metrics, our MLP model with the given configuration demonstrates good performance in predicting the target variable. However, it's important to note that further analysis and evaluation may be required to fully assess the model's robustness and potential for improvement.</p>
----	--------------------------------	---------------------------------------------	-----	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## vi. Confusion matrix

We saw from the results that the network with a learning rate of 0.001 and 4000 epochs had the best performance among the other networks. So, we create a confusion matrix for this model. For this purpose, we use the following code:

```
# Convert predicted values into binary classes based on a threshold
threshold = 0.5

y_test_pred_binary = np.where(y_test_pred >= threshold, 1, 0)

# Create the confusion matrix for testing data
test_confusion_matrix = confusion_matrix(y_test, y_test_pred_binary)

# Print the confusion matrix for testing data
print("Confusion Matrix for Testing Data:")
print(test_confusion_matrix)
```

This code takes predicted values, represented by `y_test_pred`, and converts them into binary classes based on a threshold of 0.5. Any predicted value greater than or equal to the threshold is assigned a class label of 1, while those below the threshold are assigned a class label of 0. The code then calculates and stores the confusion matrix, which is a table showing the performance of a classification model, using the predicted binary classes and the true labels `y_test`. Finally, it prints the confusion matrix for the testing data. The confusion matrix for the testing data is given in Figure 27.

```
Confusion Matrix for Testing Data:
[[0 0 0 ... 0 0 0]
 [1 0 0 ... 0 0 0]
 [1 0 0 ... 0 0 0]
 ...
 [1 0 0 ... 0 0 0]
 [1 0 0 ... 0 0 0]
 [1 0 0 ... 0 0 0]]
```

*Fig 27. Confusion matrix for testing data*



### 3. Machines classification

Neural networks are a suitable tool for health monitoring and troubleshooting of industrial machinery. In this section, we are going to use 'dataset\_2.csv' data to implement a model that can detect whether a certain type of machine tool has failure or not. The relevant data is presented in 9 columns, the characteristics (inputs) are in the fourth to eighth columns, and the output is in the last column.

#### A. Data preprocessing

First, we read the data using the following command in MATLAB:

```
%% Read the CSV file  
data = readtable('dataset_2.csv');
```

Then, we use the following code which is modifying a column called 'FailureType' in a data structure called 'data'. It will replace the label 'No Failure' with the numerical value 0 and any other labels with the numerical value 1:

```
%% Replace 'No Failure' with 0 and other labels with 1 in column 9  
data.FailureType = double(~strcmp(data.FailureType, 'No Failure'));
```

After that, we will save the updated dataset:

```
%% Save the updated dataset  
outputFile = 'updated_dataset.csv';  
writetable(data, outputFile);
```

i. Input columns (X) and output column (Y)

Again, we read the updated dataset using readmatix command:

```
%% Read the CSV file  
data = readmatrix('updated_dataset.csv');
```

Select the input data (X) from the fourth to eighth columns and the output data (Y) from the ninth column:

```
x = data(:, 4:8);  
y = data(:, 9);
```

ii. Standardization

The following code is performing a standardization operation on the variable x. Standardization is a common preprocessing step in data analysis and machine learning, which transforms the data in a way that it has zero mean and unit variance.

```
%% Standardize the data  
x = zscore(x);
```

The `x = zscore(x);` applies the `zscore` function to the variable `x` and assigns the standardized values back to `x`. The `zscore` function is commonly used in many programming languages and statistical packages to calculate the z-score of a dataset. The z-score of a data point is the number of standard deviations it is away from the mean. The `zscore` function subtracts the mean of the data and divides by the standard deviation, effectively standardizing the data.

By applying this line of code, the variable `x` is transformed to have a mean of 0 and a standard deviation of 1. This normalization step can be useful when working with data that has different scales or when certain algorithms require standardized input. Standardizing the data helps in ensuring that each feature contributes equally to the analysis and prevents features with large values from dominating the results.

## B. Neural network training

In this section, we are going to train a model with the following properties:

- Number of hidden layers: 1
- activation function: relu
- loss function: Crossentropy
- Optimizer: Levenberg-Marquardt
- Learning rate: 0.1
- max\_fail = 20
- hiddenLayerSize = 10

To train our artificial neural networks we will use the following code:

```
%% Train an ANN
xt = x';
yt = y';
hiddenLayerSize = 10; % Number of neurons in the hidden layer
net = fitnet(hiddenLayerSize, 'traingdm');
net.divideParam.trainRatio = 80/100; % 80% for training
net.divideParam.valRatio = 0/100;
net.divideParam.testRatio = 20/100;
net.trainParam.epochs = 1000;
net.trainParam.max_fail = 20;
net.trainParam.lr = 0.1;
net.trainFcn = 'trainlm'; % Levenberg-Marquardt optimizer
net.performFcn = 'crossentropy'; % Appropriate loss function for
classification
[net,tr] = train(net, xt , yt);
```

The line `net.layers{1}.transferFcn = 'poslin'`; sets the activation function for the hidden layer to ReLU. The 'poslin' argument corresponds to the ReLU function, which returns the input value if it is positive and zero otherwise.

Using the following codes, we can obtain the performance of the model:

```
%% Performance of the ANN
yTrain = net(xt(:,tr.trainInd));
yTrainTrue = yt(tr.trainInd);
trainRMSE = sqrt(mean((yTrain - yTrainTrue).^2));
trainRMSE

yTest = net(xt(:,tr.testInd));
yTestTrue = yt(tr.testInd);
testRMSE = sqrt(mean((yTest - yTestTrue).^2));
testRMSE
```

The RMSE of training and testing are given in Table 17. The performance plot is given in Figure 28.

Table 17. RMSE of test and train

Matrices	RMSE
Train	0.2623
Test	0.3680

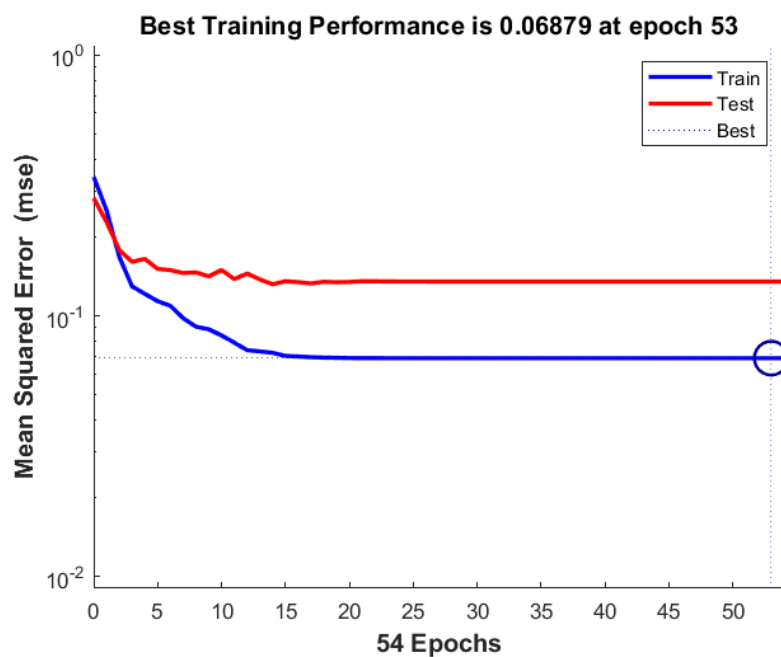


Fig 28. Performance of ANN with 10 neurons

### Max fail definition:

The fitnet function allows us to specify various options to customize the training process, and one of those options is max\_fail.

The max\_fail parameter controls the maximum number of validation checks for which the network's performance can fail to improve consecutively. During training, the network's performance on a validation set is periodically evaluated to determine if it is improving or not. If the network's performance fails to improve for max\_fail consecutive checks, the training process will stop, assuming that further training iterations will not yield better results.

C. Demands

i. Hidden layer neurons

Neurons: 1

In this case, to train our artificial neural network with 1 neuron we will use the following code:

```
% Train an ANN with 1 neuron
xt = x';
yt = y';
hiddenLayerSize = 1; % Number of neurons in the hidden layer
net = fitnet(hiddenLayerSize, 'traingdm');
net.layers{1}.transferFcn = 'poslin'; % Set ReLU as the activation
function for the hidden layer
net.divideParam.trainRatio = 80/100; % 80% for training
net.divideParam.valRatio = 0/100; % 0% for training
net.divideParam.testRatio = 20/100; % 20% for training
net.trainParam.epochs = 1000;
net.trainParam.max_fail = 20;
net.trainParam.lr = 0.1;
net.trainFcn = 'trainlm'; % Levenberg-Marquardt optimizer
net.performFcn = 'crossentropy'; % Appropriate loss function for
classification
[net,tr] = train(net, xt , yt);
```

The RMSE of training and testing are given in Table 18. The performance plot is given in Figure 29.

Table 18. RMSE of test and train with 1 neuron

Matrices	RMSE
Train	0.3735
Test	0.3544

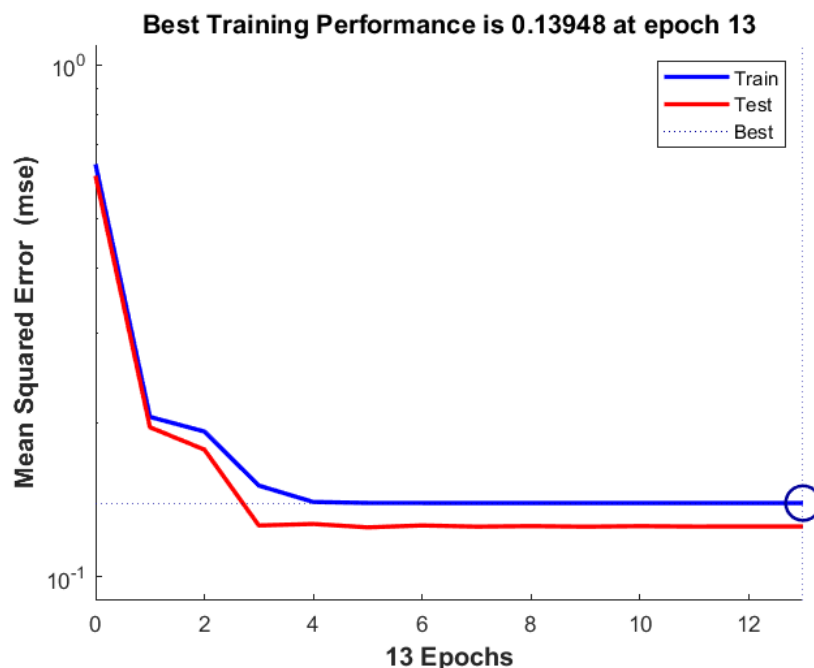


Fig 29. Performance of ANN with 1 neuron

Neurons: 30

In this case, to train our artificial neural network with 30 neurons we will use the following code:

```
% Train an ANN with 30 neurons
xt = x';
yt = y';
hiddenLayerSize = 30; % Number of neurons in the hidden layer
net = fitnet(hiddenLayerSize, 'traingdm');
net.layers{1}.transferFcn = 'poslin'; % Set ReLU as the activation
function for the hidden layer
net.divideParam.trainRatio = 80/100; % 80% for training
net.divideParam.valRatio = 0/100; % 0% for training
net.divideParam.testRatio = 20/100; % 20% for training
net.trainParam.epochs = 1000;
net.trainParam.max_fail = 20;
net.trainParam.lr = 0.1;
net.trainFcn = 'trainlm'; % Levenberg-Marquardt optimizer
net.performFcn = 'crossentropy'; % Appropriate loss function for
classification
[net,tr] = train(net, xt , yt);
```

The RMSE of training and testing are given in Table 19. The performance plot is given in Figure 30.

Table 19. RMSE of test and train with 30 neurons

Matrices	RMSE
Train	0.2324
Test	0.3532

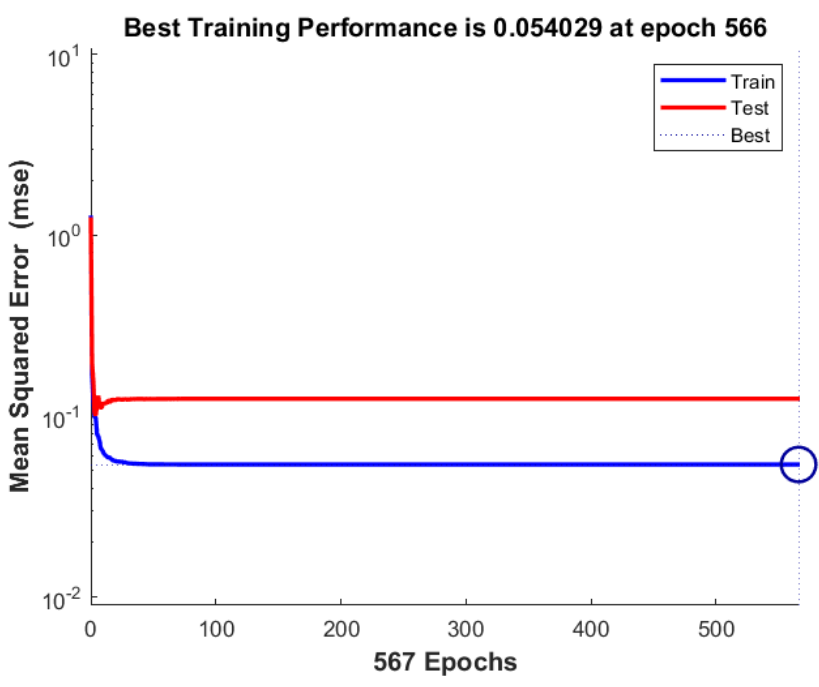


Fig 30. Performance of ANN with 30 neurons

Neurons: 500

In this case, to train our artificial neural network with 500 neurons we will use the following code:

```
% Train an ANN with 500 neurons
xt = x';
yt = y';
hiddenLayerSize = 500; % Number of neurons in the hidden layer
net = fitnet(hiddenLayerSize, 'traingdm');
net.layers{1}.transferFcn = 'poslin'; % Set ReLU as the activation
function for the hidden layer
net.divideParam.trainRatio = 80/100; % 80% for training
net.divideParam.valRatio = 0/100; % 0% for training
net.divideParam.testRatio = 20/100; % 20% for training
net.trainParam.epochs = 1000;
net.trainParam.max_fail = 20;
net.trainParam.lr = 0.1;
net.trainFcn = 'trainlm'; % Levenberg-Marquardt optimizer
net.performFcn = 'crossentropy'; % Appropriate loss function for
classification
[net,tr] = train(net, xt , yt);
```

The RMSE of training and testing are given in Table 20. The performance plot is given in Figure 31.

Table 20. RMSE of test and train with 500 neurons

Matrices	RMSE
Train	0.0722
Test	0.4055



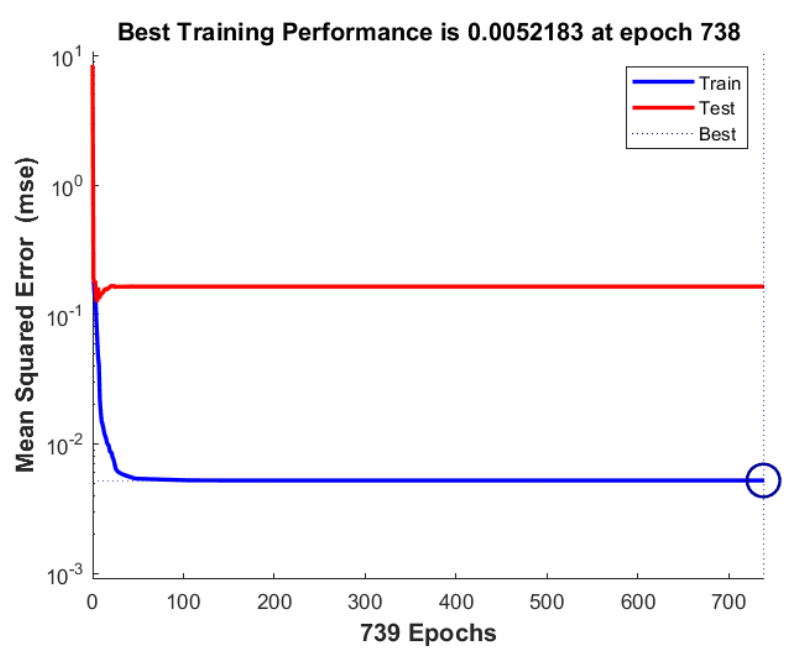


Fig 31. Performance of ANN with 500 neurons

57	Masoud Pourghavam 810601044	Artificial Intelligence Dr. M. S. Panahi	HW4	<p><b>Influence of network size on training:</b></p> <p>The RMSE (Root Mean Square Error) values can provide insights into the influence of the hidden layer size on the training performance of the artificial neural network. For the case of 1 neuron in the hidden layer, the train RMSE is 0.3735 and the test RMSE is 0.3544. These values suggest that the model is underfitting the data, as the error is relatively high. Having only one neuron might limit the capacity of the model to learn complex patterns and relationships in the data.</p> <p>When the hidden layer size is increased to 30 neurons, we observe a decrease in both train and test RMSE values. The train RMSE improves to 0.2324, indicating a better fit to the training data, while the test RMSE remains relatively low at 0.3532. This suggests that increasing the hidden layer size has allowed the model to capture more complex patterns in the data, leading to improved performance.</p> <p>Finally, with 500 neurons in the hidden layer, we see a significant improvement in the training performance. The train RMSE decreases to 0.0722, indicating a much better fit to the training data. However, interestingly, the test RMSE increases to 0.4055. This suggests that the model might be overfitting the training data, as it is not generalizing well to unseen test data. The large number of neurons might be allowing the model to memorize the training data too closely, resulting in decreased performance on unseen examples.</p> <p>So, we can conclude these results indicate that the hidden layer size has a considerable impact on the training performance of the neural network. Increasing the number of neurons can generally improve the model's ability to capture complex patterns and achieve better performance on the training data. However, there is a trade-off, as excessively large hidden layer sizes can lead to overfitting and reduced generalization ability on unseen data. Finding the optimal balance between model capacity and generalization is crucial for achieving the best performance.</p>
----	--------------------------------	---------------------------------------------	-----	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

ii. Performance and Max\_fail

Neurons: 20

In this case, to train our artificial neural network with 20 neurons we will use the following code:

```
% Train an ANN with 20 neurons
xt = x';
yt = y';
hiddenLayerSize = 20; % Number of neurons in the hidden layer
net = fitnet(hiddenLayerSize, 'traingdm');
net.layers{1}.transferFcn = 'poslin'; % Set ReLU as the activation
function for the hidden layer
net.divideParam.trainRatio = 80/100; % 80% for training
net.divideParam.valRatio = 0/100; % 0% for training
net.divideParam.testRatio = 20/100; % 20% for training
net.trainParam.epochs = 1000;
net.trainParam.max_fail = 20;
net.trainParam.lr = 0.1;
net.trainFcn = 'trainlm'; % Levenberg-Marquardt optimizer
net.performFcn = 'crossentropy'; % Appropriate loss function for
classification
[net,tr] = train(net, xt , yt);
```

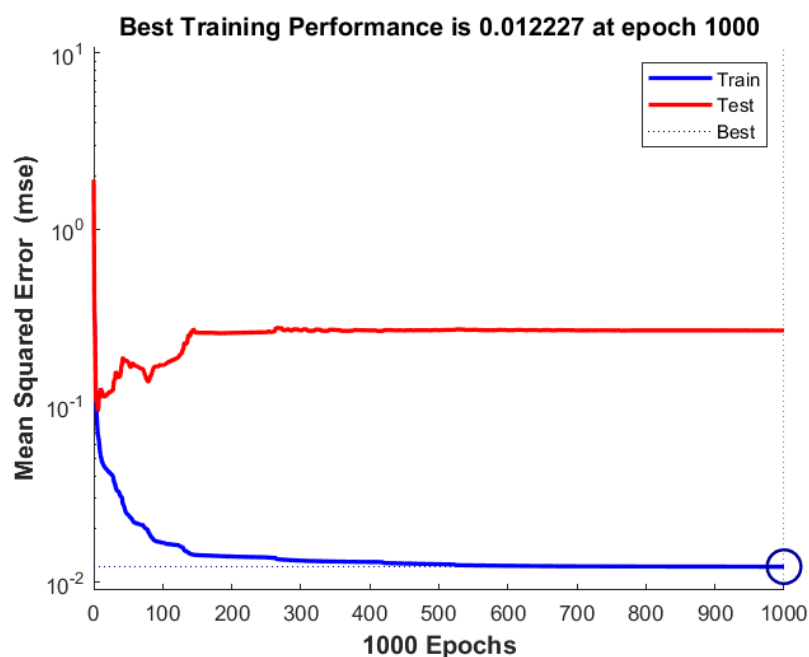
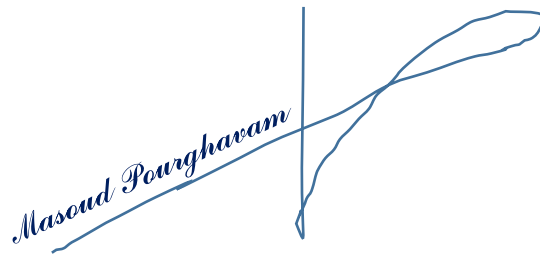


Fig 32. Performance of ANN with 20 neurons

59	Masoud Pourghavam 810601044	Artificial Intelligence Dr. M. S. Panahi	HW4	
	<p><b>Influence of max fail change:</b></p> <p>Setting a higher value for max_fail allows the training process to continue for a longer time, giving the network more chances to improve. However, this may also increase the risk of overfitting, where the network becomes too specialized to the training data and performs poorly on unseen data. On the other hand, setting a lower value for max_fail can result in shorter training times but may stop the training prematurely, potentially before the network has converged to an optimal solution.</p> <p>It's important to find a balance when setting the max_fail value based on the characteristics of our data and the desired training outcome. Typically, it is recommended to monitor the network's performance on a separate validation dataset during training and choose a value for max_fail that prevents overfitting while allowing the network to converge to a satisfactory solution.</p>			

# Thanks for wer Time



*Masoud Pourghavam*