

**DEEP LEARNING AND NEURAL NETWORK**

**PROJECT REPORT**

Submitted to: Saira Osama

Submitted by: Ahmad Shamayl f2020332021

Contents

[INTRODUCTION 1](#_Toc134318084)

[DATASET 2](#_Toc134318085)

[PREPROCESSING 3](#_Toc134318086)

[Data Cleaning 3](#_Toc134318087)

[Data Reduction 4](#_Toc134318088)

[Data transformation 4](#_Toc134318089)

[Methodology: 4](#_Toc134318090)

[Batch Gradient Descent 6](#_Toc134318091)

[Mini Batch Gradient descent 6](#_Toc134318092)

[Stochastic Gradient descent: 7](#_Toc134318093)

[Plotting learning curve 7](#_Toc134318094)

[Stochastic: 8](#_Toc134318095)

[Batch: 9](#_Toc134318096)

[Mini-Batch: 10](#_Toc134318097)

[Conclusion: 10](#_Toc134318098)

**INTRODUCTION**

Building a neural network from scratch might be challenging, but it is also an excellent method to learn how neural networks function and develop a greater understanding of machine learning. Using the Boston Housing dataset, we will create a neural network from scratch in this project. The Boston Housing dataset is a well-known dataset that provides information on various houses in Boston, such as the number of rooms, the house's age, and the median value of homes in the neighborhoods. The purpose of this project is to create a neural network that can estimate the median value of a home based on the other variables in the dataset.

Neural networks have gained importance in recent years due to their exceptional capacity to learn complicated patterns in data. The Boston House Price dataset, which consists of 506 samples, each with 13 variables and a target variable of housing price, is a common dataset for regression problems. Building a neural network from scratch to anticipate home prices is an interesting challenge in this type of environment.

The activation function in neural networks is critical in bringing nonlinearity to the model. For any input value less than or equal to 0, the ReLU activation function returns 0; for any input value larger than 0, it returns the input value. The ReLU activation function offers various benefits over the sigmoid and tanh activation functions. First and foremost, it is computationally efficient and simple to implement. Second, it avoids the vanishing gradient problem that may arise with sigmoid and tanh when input values are big. Finally, it has been demonstrated that it is effective and can increase the performance of deep neural networks.

Creating a neural network from scratch using the Boston Housing dataset and the ReLU activation function is an excellent method to learn more about neural networks and machine learning. With this project, we can investigate the many components of a neural network, such as the activation function, and observe how they affect the model's performance**.**

# **DATASET**

The United States Census Bureau gathered information about housing in the Boston area for this dataset. Despite the fact that some of these ratings were done outside of the Delve platform and may not be completely valid, they have been frequently utilized in the research community to evaluate alternative algorithms. The dataset is tiny, with 506 examples, as is customary.

The data came from D. Harrison and D. L. Rubinfeld's essay "Hedonic prices and the demand for clean air," which was published in the Journal of Environmental Economics and Management in 1978. There are 14 attributes in each case of the dataset. They are:

1. **CRIM** - per capita crime rate by town
2. **ZN** - proportion of residential land zoned for lots over 25,000 sq.ft.
3. **INDUS** - proportion of non-retail business acres per town.
4. **CHAS** - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
5. **NOX** - nitric oxides concentration (parts per 10 million)
6. **RM** - average number of rooms per dwelling
7. **AGE** - proportion of owner-occupied units built prior to 1940
8. **DIS** - weighted distances to five Boston employment centers
9. **RAD** - index of accessibility to radial highways
10. **TAX** - full-value property-tax rate per $10,000
11. **PTRATIO** - pupil-teacher ratio by town
12. **B** - 1000(Bk - 0.63) ^2 where Bk is the proportion of blacks by town
13. **LSTAT** - % lower status of the population
14. **MEDV** - Median value of owner-occupied homes in $1000's

# **PREPROCESSING**

Data preprocessing is the process of converting raw data into a format that can be analyzed by machine learning algorithms.

In this project we have used data cleaning, dimensionality reduction, and data transformation preprocessing techniques

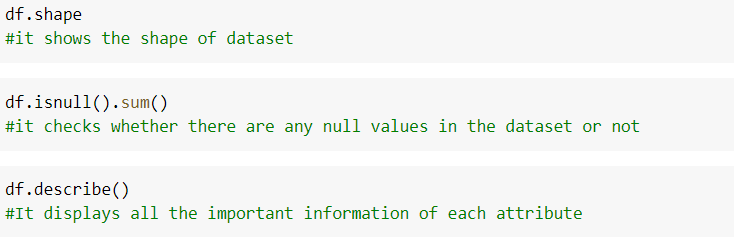
Data cleaning

Data reduction

Data transformation (Standardization)

# **Data Cleaning**

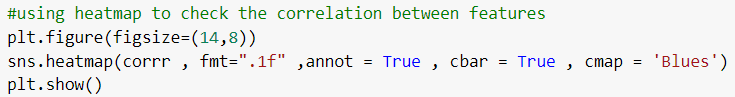
In this case, we must remove missing values, smooth the noisy data, and make the data consistent.



# **Data Reduction**

In data reduction, we may erase features that are not necessary for our model's prediction; this process is known as Dimensionality Reduction.

There is also the option of reducing the number of data possibilities to make it more correlated.

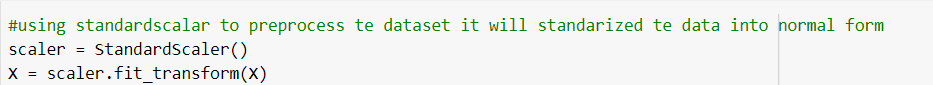




# **Data transformation**

Standardization is the process of subtracting the mean and then dividing it by the standard deviation.

We have also plotted pair plot of all the features which shows the relation of each attribute with other



# **Methodology:**

Our neural network architecture is divided in 70 30 ratio and it will consist of three layers: an input layer, a hidden layer, and an output layer. The input layer will have 12 neurons basically this dataset has 13 features to train but after plotting the correlation of all features we came to know that ‘RAD’ and ‘TAX’ are highly correlated so we have then dropped ‘RAD’ attribute in order to reduce its dimensions so that our model can train well. The hidden layer will have 30 neurons, and the output layer will have only 1 neuron as we have to predict the prices

In the hidden layer of this neural network, we use the ReLU activation function.

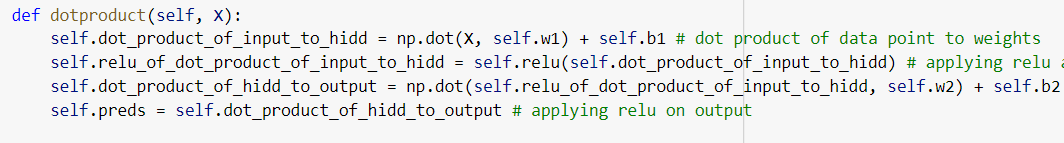
Because of its efficacy and simplicity, the ReLU activation function is utilized in neural networks. It uses a simple max operation, which makes it more computationally efficient than other activation functions.

ReLU has been employed in neural networks to introduce non-linearity and improve the network's ability to represent input-output correlations.



We will train the model across 250 epochs using stochastic, batch, and small batch gradient descent with a batch size of 32 and a learning rate of 0.01. During each training iteration, we will randomly choose 32 samples from the training dataset, feed them forward through the network,

**Forward** **propagation** is a method used in neural networks to determine the network's output for a given input. It involves passing the input data through the network layers one by one until the output layer is reached. The inputs are multiplied by weights and added to bias terms, and the result is then sent via an activation function. The output is then compared to the desired output to determine the network error.



And then calculating the loss using mean squared error,

**Mean Squared Error (MSE)** is a regularly used loss function in regression projects to assess the difference between predicted and actual data. It is determined as the average of the squared discrepancies between the target variable's expected and actual values.

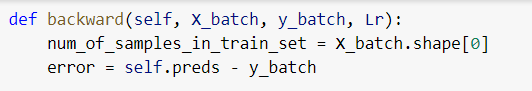
Mathematically, MSE can be represented as:

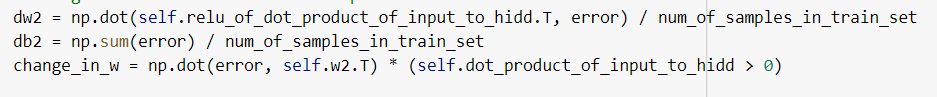
MSE = (1/n) \* Σ(yi - ŷi)²

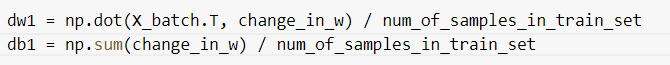


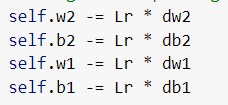
and use backpropagation to update the network's weights and biases.

**Backpropagation** is a technique used to train models in machine learning and artificial neural networks. It computes the gradients of the loss function with respect to the network weights, which can be used to adjust the weights using an optimization approach.









## **Batch Gradient Descent**

A prominent optimization approach known as batch gradient descent minimizes the training phase cost function of a machine learning model.

This approach divides the training dataset into smaller groups, or "batches," and trains the model on each batch repeatedly until convergence. For each iteration, the method calculates the gradient of the cost function relative to the model parameters for each sample in the batch. The model parameters are then updated by multiplying the batch average gradient by a hyperparameter known as the learning rate. The learning rate determines the magnitude of the step toward the minimum of the cost function. This method is continued for all batches until the model reaches a minimum of the cost function.

Batch Gradient Descent is an excellent alternative for small datasets since the batch size may be adjusted to fit in memory. The model parameters may be modified continuously for each batch, making the process computationally efficient. Other optimization techniques, such as Stochastic Gradient Descent or Mini-Batch Gradient Descent, may be better appropriate for huge datasets since they are faster and more effective.

## **Mini Batch Gradient descent**

The training cost function of a machine learning model can be decreased using the optimization approach known as mini-batch gradient descent.

Mini-Batch Gradient Descent splits training data into batches in the same way as Batch Gradient Descent does. Unlike Batch Gradient Descent, the gradient of the cost function is only calculated for each iteration using a subset or mini-batch of data. It is conventional to choose a mini-batch size that is both small enough to fit in memory and large enough to represent the complete dataset accurately.

For each mini-batch and sample inside it, the method computes the gradients of the cost function with respect to the model parameters. The model parameters are then updated using the average gradient across the mini-batch and a learning rate hyperparameter that controls the size of the step made in the direction of the cost function's minimum.

All mini-batches go through this process of updating model parameters until the model converges to a cost function minimum.

Batch Gradient Descent is less efficient than Mini-Batch Gradient Descent in terms of processing efficiency, but it can incorporate noise into gradient estimations.

## **Stochastic Gradient descent:**

The training phase cost function of a machine learning model can be reduced using a popular optimization approach known as stochastic gradient descent.

In contrast to Batch Gradient Descent and Mini-Batch Gradient Descent, Stochastic Gradient Descent adjusts the model parameters after each unique training sample. The approach calculates the gradient of the cost function with respect to the model parameters for each training sample before updating the parameters using a learning rate hyperparameter.

This method is repeated for all training samples until the model reaches the minimum of the cost function.

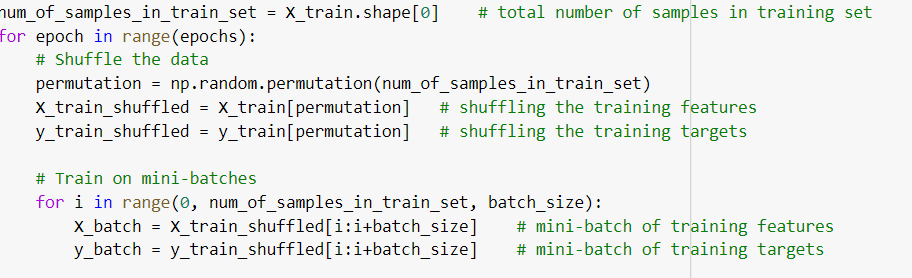
Stochastic gradient descent can be quicker than Batch Gradient Descent and Mini-Batch Gradient Descent since it changes the model parameters more often. However, because the gradient estimates are based on only one training sample at a time, the cost function may experience greater variations during training, making it more vulnerable to noise. Furthermore, because the method is constantly updating, fine-tuning the learning rate hyperparameter in stochastic gradient descent can be challenging.

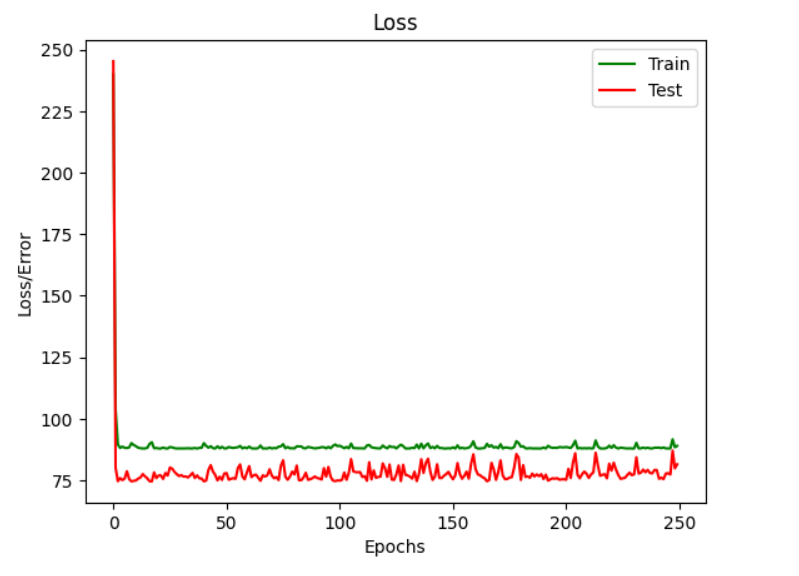
# **Plotting learning curve**

We have now plotted a machine learning model's learning curve using the computed losses from the training and testing stages. A learning curve may be used to determine if a model is overfitting or underfitting the data.

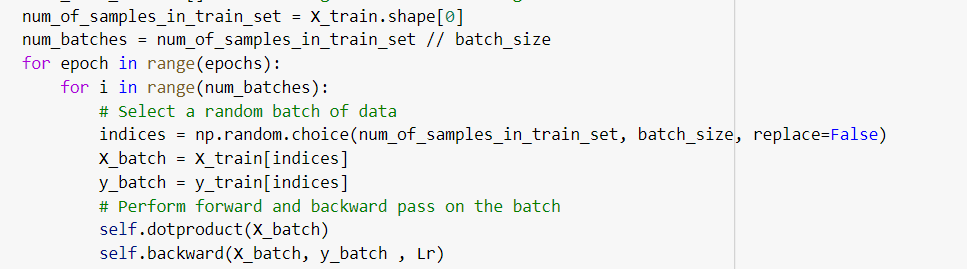
We can determine how successfully our model gets data by displaying the learning curve. If the training and testing losses are identical, our model is appropriately calibrated and capable of generalizing to new data. If the testing loss plateaus or climbs while the training loss continues to decrease, our model is overfitting and not generalizing adequately. If both the training and testing losses are substantial, our model may not be capturing the underlying patterns in the data effectively.

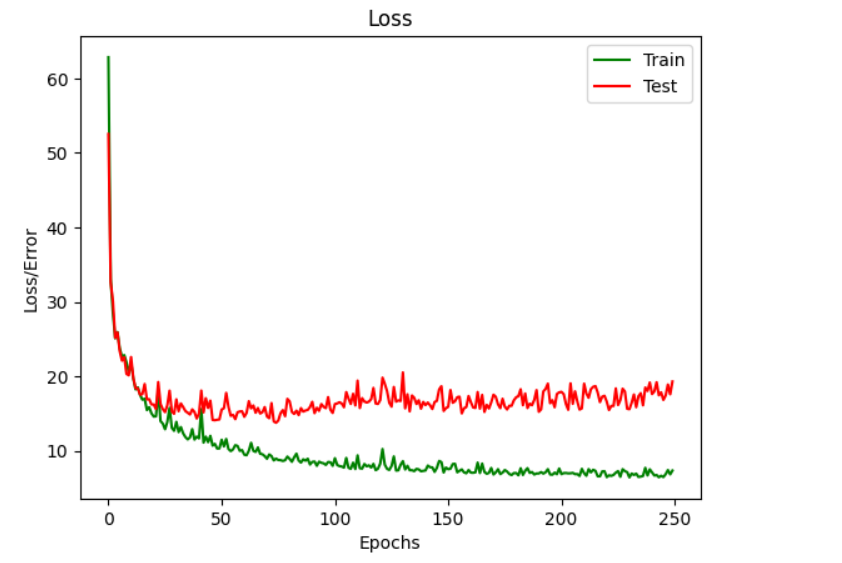
## **Stochastic:**



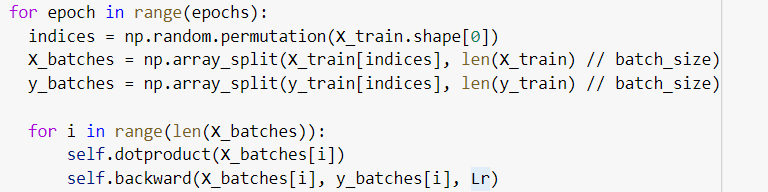


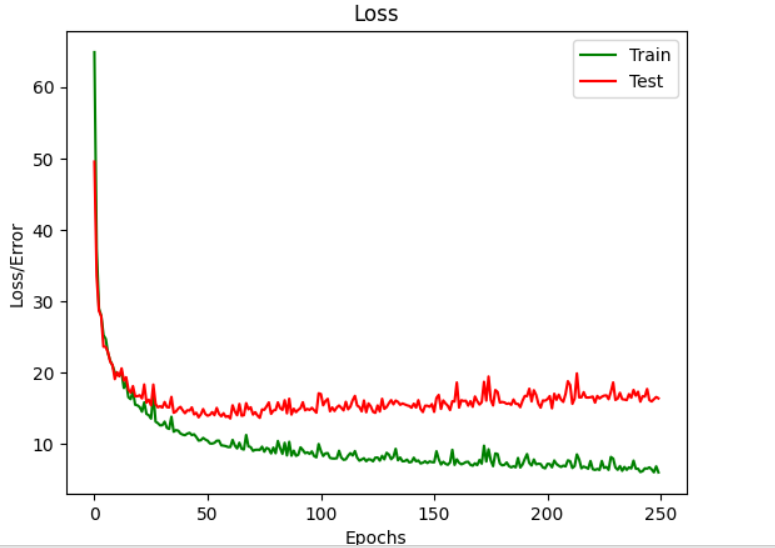
## **Batch:**

****



## **Mini-Batch:**





# **Conclusion:**

The United States Census Bureau collected data on homes in the Boston area, including 14 variables to train. The neural network architecture is divided into three layers, with the input layer including 12 neurons and the output layer containing 30 neurons. The ReLU activation function is used to introduce nonlinearity. Batch Gradient Descent is a prominent optimization strategy that minimizes the training phase cost function of a machine learning model by iteratively training the model on each batch until convergence. Mini-Batch Gradient Descent and Stochastic Gradient Descent both help to reduce the training cost function of a machine learning model.

Stochastic gradient descent is quicker, but it is more prone to noise and more difficult to adjust. A learning curve can be used to detect whether or not a model is overfit