




# Pranav Nagaram

Btech. Computer Science and Engineering

IIT ISM Dhanbad


# PROJECT REPORT

# ML-BOOTCAMP

# Preface

In this project, we embark on the development of a Machine Learning Algorithm Library. Our goal is to implement key algorithms – **Linear and Polynomial Regression, Logistic Regression, KNN, K-Means Clustering, and n-layer Neural Networks** – from scratch, excluding external machine learning libraries.

Implementation is exclusively powered by fundamental libraries such as **NumPy** for numerical operations, **Matplotlib** for visualizations, and **Pandas** for data manipulation. Additionally, the **random** module for initializations.

The subsequent report outlines our journey, detailing conceptualization, implementation, and outcomes for each algorithm. Training logs, hyperparameters, and visualizations will be presented, providing a comprehensive account of our technical exploration.

Pranav

[22/01/2024]

Model 1

# Linear Regression Model

Linear Regression is a fundamental machine learning algorithm employed for predicting a continuous target variable, relying on one or more predictor variables. This implementation utilizes the gradient descent method to minimize the sum of squared errors.

We initiate our linear regression implementation with a class "LinReg," which takes the training dataset (x_train, y_train) as input. The class incorporates a "Train" method, employing gradient descent to update the coefficients of linear regression. Subsequently, the "Test" method utilizes these updated coefficients to predict the target variable on the test dataset (x_test).

To assess the model's effectiveness, we use the "R2Score" method, which compares the predictions with the actual target variable (y_test). The R2 score serves as a metric to check the model's performance.

# 1.1 Data Analysis

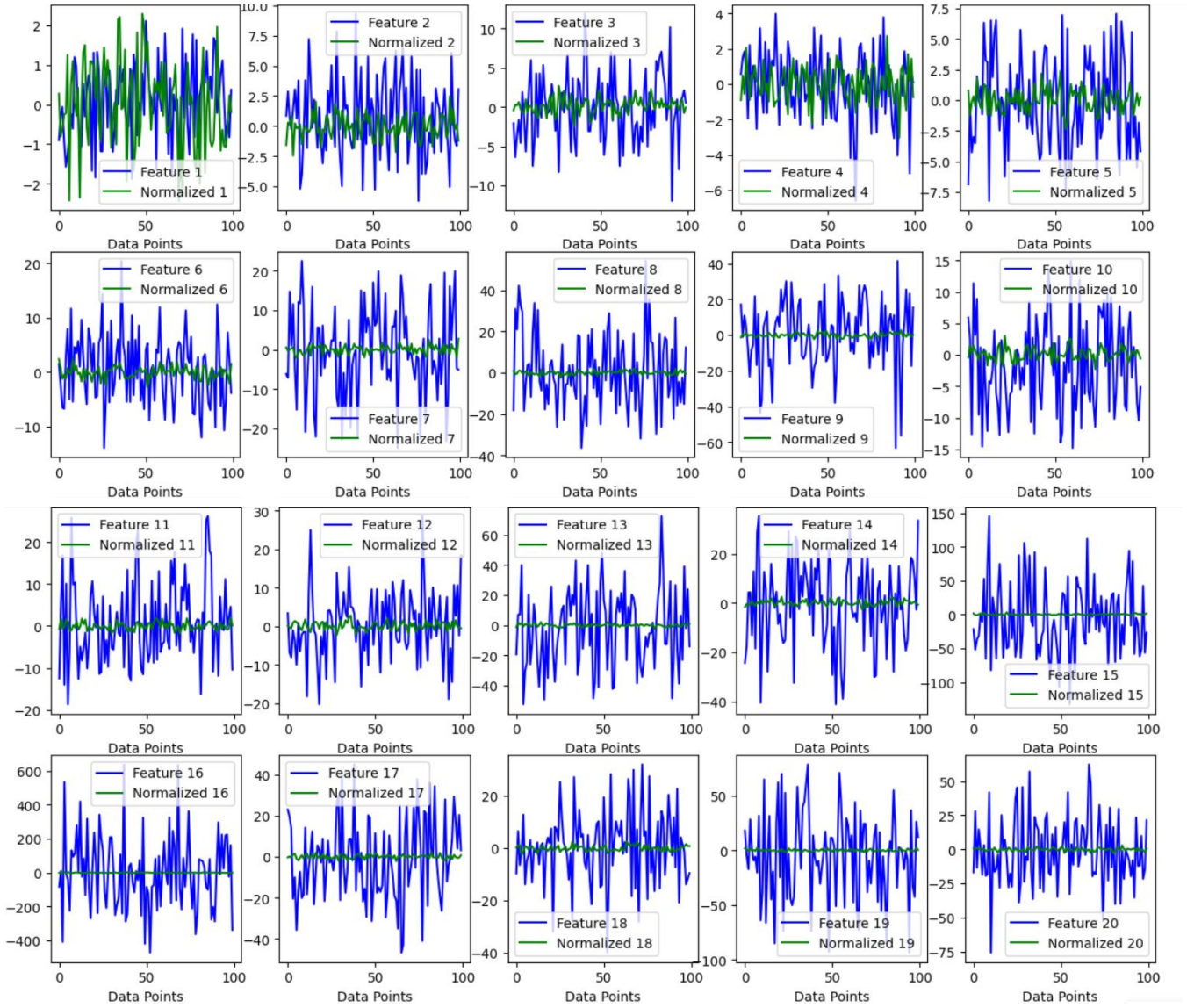
## 1.1.1 Standardization

As it is very clear, the features have a very different range of values and hence it is not advisable to train such a data using common hyper parameters such as learning rate and it would increase the complexity to adjust a different learning rate for each feature.

We apply **Z-Normalization**:

$$x_i = \frac{x_i - \mu}{\sigma}$$

The mean and variance of each feature is 1 and 0 respectively after normalization.



### 1.1.2 Data Splitting

The given dataset is split for training and testing in the ratio of 70:30 respectively.

## 1.2 Implementation

### 1.2.1 Algorithm

- Created a Class LinReg
- Defined learning rate(alpha) and iterations using __init__ method
- Defined a Normalize function using the below formulae
- Defined a Train function for training the model using the below listed formulae
- Defined Predict function to make prediction and calculate error
- Implemented R2score metrics to evaluate the performance

Cost Function:

$$\hat{y} = \mathbf{x} \cdot \mathbf{w} + b$$
$$J = \frac{1}{2m}\sum_{i=1}(\hat{y}_i - y_i)^2 \implies J = \frac{1}{2m}\sum_{i=1}^{m}(\mathbf{x}_i \cdot \mathbf{w}_i - b)^2$$

Gradient:

$$\frac{\partial J}{\partial w_i} = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{x}_i \cdot \mathbf{w}_i - b) \cdot \mathbf{x}_i$$
$$\frac{\partial J}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{x}_i \cdot \mathbf{w}_i - b)$$

Gradient Descent:

$$w_i = w_i - \alpha\frac{\partial J}{\partial w_i}$$
$$b = b - \alpha\frac{\partial J}{\partial b}$$

### 1.2.2 Training and Testing

After Training the dataset using the below hyperparameters the following parameters and cost were observed

Hyperparameters:

- Learning rate: 0.3
- Iterations: 1000

Weights:

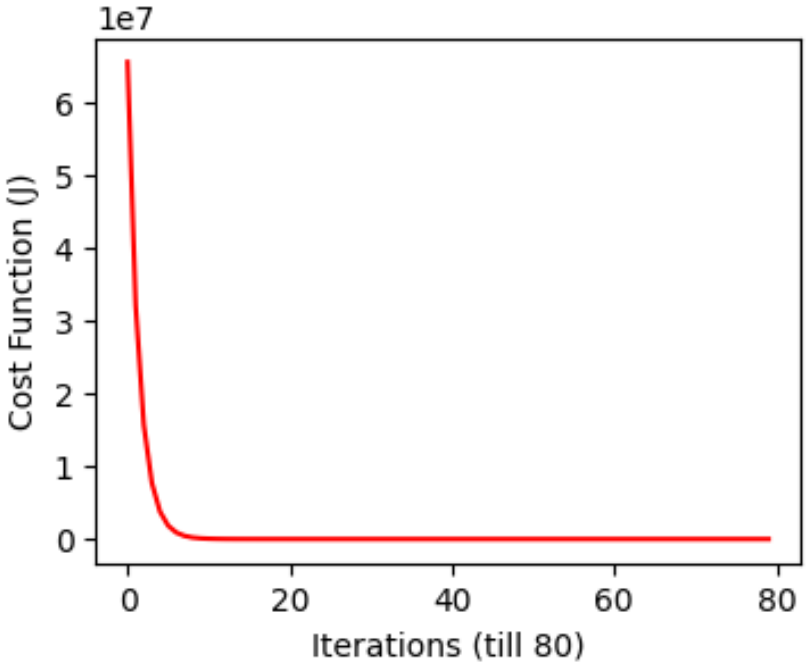
```
[[2.20207252e+01]
 [3.31890795e+01]
 [9.96997627e+00]
 [6.03812144e+00]
 [1.76332707e+02]
 [3.28868873e+02]
 [2.53732278e+02]
 [9.03338209e+02]
 [4.35707623e+02]
 [8.40716239e+01]
 [1.21127936e+02]
 [1.60055381e+03]
 [1.67554756e+03]
 [1.31759217e+03]
 [5.98759394e+02]
 [1.05639290e+04]
 [5.05086554e+02]
 [6.77476538e+02]
 [3.16400243e+03]
 [2.30206547e+02]]
```

Bias:

```
[[60.09220304]]
```

| Iteration | Cost |
|---|---|
| 0 | 65590577.84793029 |
| 200 | 0.005036888952156986 |
| 400 | 0.005036888952156986 |
| 600 | 0.0050368889521568445 |
| 800 | 0.005036888952156986 |

R2_score :0.9999775642782995



# Model 2

# Polynomial Regression Model

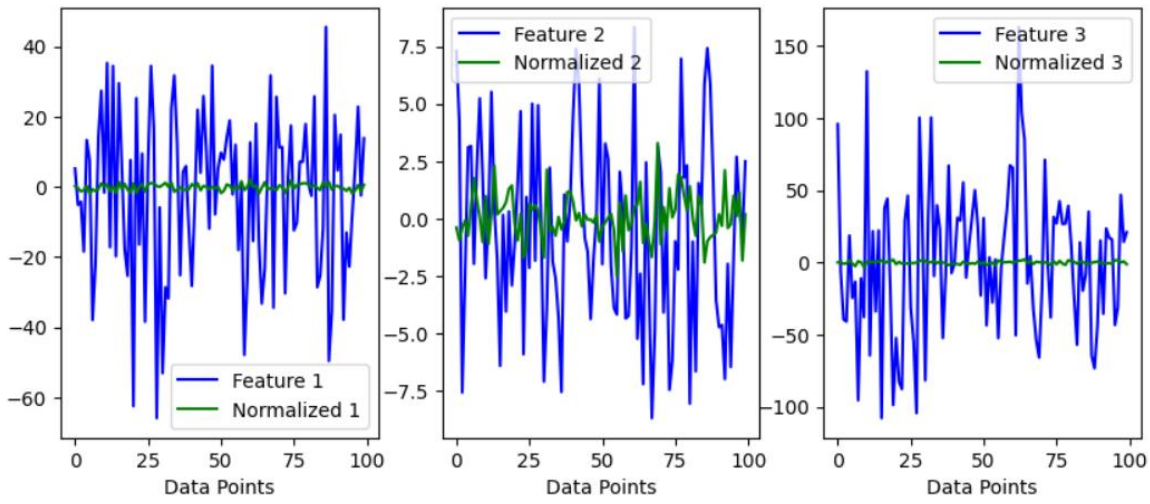
Similar to Linear Regression Model this Algorithm uses the gradient descent method to minimize the sum of squared errors and fit a polynomial function of suitable degree in the given dataset.

To look into the problem of overfitting or underfitting **lasso** and **ridge** regression have been implemented.

## 2.1 Data Analysis

### 2.1.1 Standardization:

Using Z-Normalization:



### 2.1.2 Data Splitting:

The data is split in the ratio 70:30 for Training and Testing respectively

## 2.2 Implementation

### 2.2.1 Algorithm:

- Created a Class PolReg
- Defined degree, learning rate(α), iterations, λ1 and λ2 using __init__ method
- Function to Polymerize the dataset according to highest degree
- Defined a Train function for training the model using the below listed formulae
- Defined Predict function to make prediction and calculate error

Cost Function:

$$J = \frac{1}{2m}(\sum_{i=1}^{m}(x \cdot w - y)^2 + \alpha_{reg}\lambda_1 \sum_{j=1}^{n}|w_j| + (1 - \alpha_{reg})\lambda_2 \sum_{j=1}^{n} w_j^2)$$

With **Lasso** and **Ridge** Regression respectively.

Gradient:

$$\frac{\partial J}{\partial w} = \frac{1}{m}(x_i \cdot (x_i \cdot w - y) + \alpha_{\text{reg}}\lambda_1 \cdot \text{sign}(w) + (1 - \alpha_{\text{reg}})\lambda_2 \cdot w)$$

$$\frac{\partial J}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(x \cdot w - y)$$

Gradient Descent:

$$w_i = w_i - \alpha\frac{\partial J}{\partial w_i}$$

$$b = b - \alpha\frac{\partial J}{\partial b}$$

### Regularization:

L1 regularization or Lasso regularization adds a penalty to the cost function which is proportional to the abs value of weights. This reduces the values and selects only the most important features and is commonly used for datasets with many features.

L2 regularization of Ridge regularization adds a penalty to the cost function which is proportional to the sq. of weights. It helps in reducing the variance and also prevents overfitting.

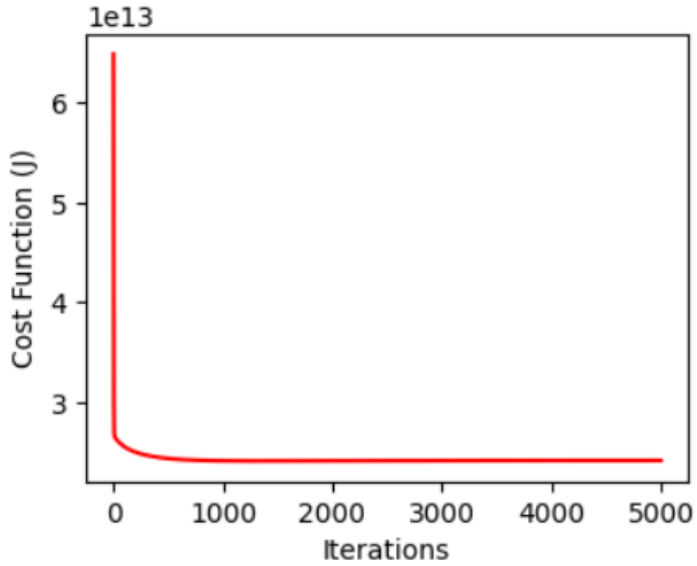
### 2.2.2 Training and Testing

After Training the dataset using the below hyperparameters the following parameters and cost were observed

Hyperparameters:

- Learning rate: 0.003
- Iterations: 1000
- $\lambda 1=0.6$
- $\lambda 2=0.6$
- $\alpha_{reg}=0.5$
- Degree =4

```
Cost after 0th iteration is: 64766983955491.96
Cost after 500th iteration is: 24433182242887.12
Cost after 1000th iteration is: 24200789417445.387
Cost after 1500th iteration is: 24187132590102.98
Cost after 2000th iteration is: 24202136706493.64
Cost after 2500th iteration is: 24214728286307.273
Cost after 3000th iteration is: 24222650499001.24
Cost after 3500th iteration is: 24227471244428.965
Cost after 4000th iteration is: 24230500159408.074
Cost after 4500th iteration is: 24232496299881.65
cost after 5000th iterations is: 24233867535713.26
R2_score :0.6145585318897964
```



# Model 3

# Logistic Regression Model

Multiclass Logistic Regression is a classification algorithm that predicts the probability that an instance belongs to a particular class.

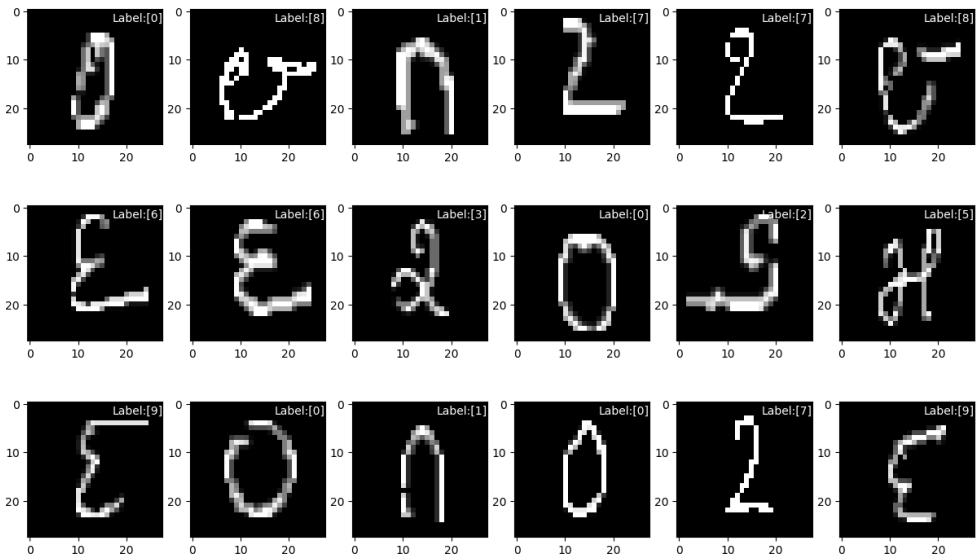
In this model gradient descent is used to reduce the minimize the cross-entropy function.

## 3.1 Data Analysis

The given data set consists of multiple grayscale images of 784 pixels(28x28) with labelled with 10 different classes with value of a single pixel ranging from 0 to 256 which represent the brightness of the respective pixel.

Standardizing the dataset by dividing it by 256. The values now range from 0 to 1

Few images:



## 3.2 Implementation

### 3.2.1 Algorithm

- Created a class LogReg
- Defining two methods Train and Predict which perform the training and predictions respectively
- Defining functions like one_hotcode, softmax, prop which help to reduce complexity of the Train function
- Accuracy function to gauge how efficient the code is

SoftMax Function:

$$\sigma(x)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{for } j = 1, \ldots, K.$$

Cross Entropy Loss:

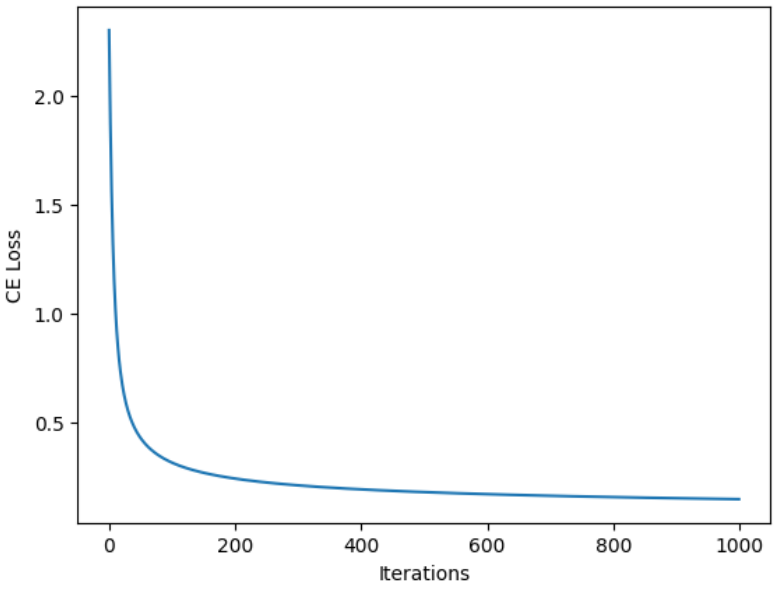
$$CELoss = -\sum_{c=1}^{N} y_c \log P_c$$

### 3.2.2 Testing and Training

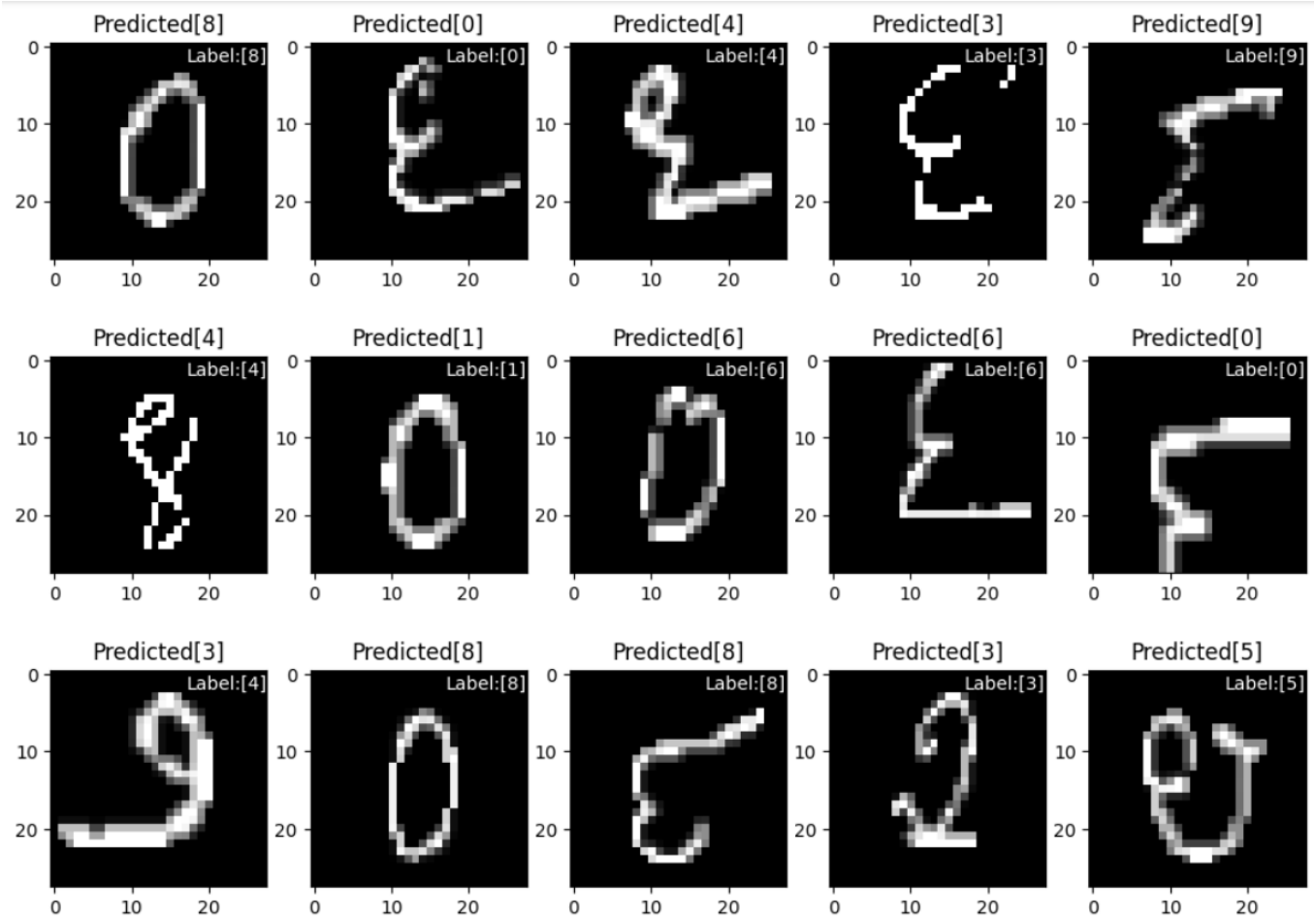
After Training the dataset using the below hyperparameters the following parameters and cost were observed

Hyperparameters:

- Learning rate: 0.3
- Iterations: 1000




Accuracy = 95.98%



# Model 4

# KNN Model

KNN or K-Nearest Neighbours is a non-parametric, lazy learning algorithm. It's easy to understand and implement but may be computationally expensive, especially with large datasets, as it requires calculating distances to all training points.

For each datapoint, the K-Nearest Neighbours are observed and the most frequent class will be assigned to that datapoint.

## 4.1 Implementation

### 4.1.1 Algorithm

- Created a class KNN
- Defining two methods Train and Predict which perform the training and predictions respectively
- Creating help methods such as Mode, Euclidean Distance, Absolute distance

### 4.1.2 Testing

Using the Euclidean distance since it is vectorized and computational time is less.

Optimal value of k =6

Accuracy= 98.37%

# Model 5

# K-Means Clustering Model

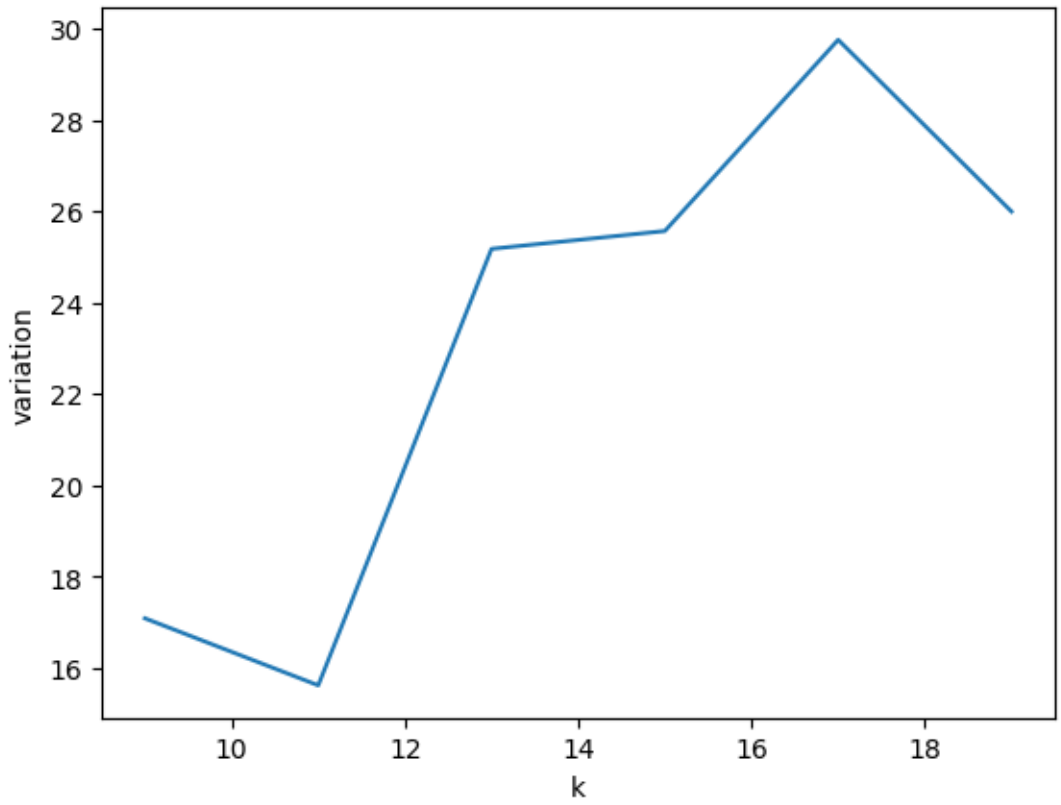
K-Means is a popular unsupervised machine learning algorithm used for clustering. The main goal of the K-Means algorithm is to partition a dataset into K clusters, where each data point belongs to the cluster with the nearest mean. The algorithm iteratively assigns data points to clusters based on their similarity and updates the cluster centroids until convergence.

## 5.1 Implementation

### 5.1.1 Algorithm

- Created a class KM_cluster
- Initialising centroids in a way such that it is not randomized every time it is trained for a different value of k so that they can be compared
- Using Euclidean distance to calculate the distance
- Finding the optimum value of k

### 5.1.2 Testing




The least value of loss is for k=11.

# Model 6

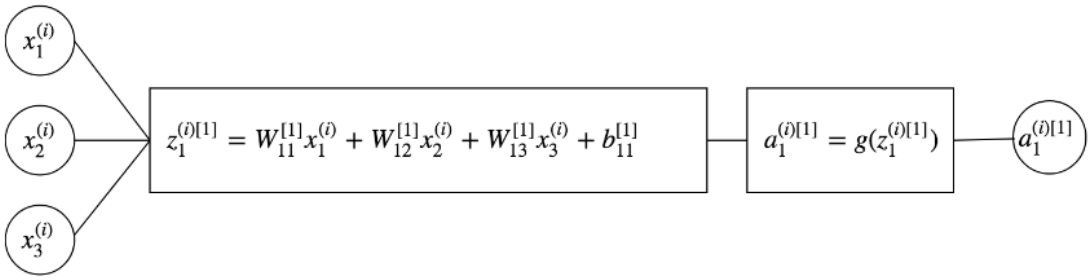
# L-layer Neural Network

An L-layer neural network refers to a neural network architecture with L layers, where L includes both the input and output layers, as well as any hidden layers in between. Each layer consists of a certain number of neurons, also known as units or nodes, and these neurons are interconnected through weights. The architecture typically follows a feedforward approach, where information flows from the input layer through the hidden layers to the output layer.

## 5.1 Implementation

- Created a class network which takes in layers and the number of neurons in them as a single array of the format [784,….,10].
- I have implemented mini batch gradient descent method to increase the efficiency and reduce the computational time.
- I have created train method which performs the training of the model which uses two other methods forward propagation and backward propagation, also a predict method which makes the predictions.
- While training, forward propagation computes neuron activations using weights and biases. Backward propagation calculates gradients for parameter updates based on the final layer's output. This iterative process occurs across batches and epochs, optimizing the neural network.

**Forward Propagation:**




- Where g(x) is the respective Activation function
- I have implemented 3 different activation functions namely Sigmoid, Tanh, Leaky ReLU
- Using Categorical cross entropy as loss function and reducing it in backward propagation as follows

**Backward Propagation:**

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^{[l-1]}} = \left[\boldsymbol{W}^{[l]}\right]^{\mathrm{T}} \cdot \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^{[l]}} * g^{[l-1]\prime}\left(\boldsymbol{z}^{[l-1]}\right)$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^{[l]}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^{[l]}} \cdot \left[\boldsymbol{a}^{[l-1]}\right]^{\mathrm{T}}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^{[l]}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^{[l]}}$$

Where g’ is the derivative of the respective activation function and a[l-1] is the activations of previous layer

## 5.2 Training and Testing

Hyperparameters:

- Learning rate: 0.3
- Iterations: 30
- Layers = [784,426,68,10]

```
Accuracy for tanh activation is 94.17%
Accuracy for sigmoid activation is 95.39%
Accuracy for leaky_relu activation is 96.7%
leaky_relu has the highest accuracy of 96.7%
```