**Linear Regression on Cancer Data**

* **Introduction**
* In this project, we investigate the relationship between X variables and Y variables in a dataset called Cancer Data. The goal is to use linear regression to predict the Y values based on the X values.
* This code implements linear regression using three different optimization algorithms: gradient descent, mini-batch gradient descent, and Adam optimization.
* **Data Exploration**
* First, we load the data into a pandas data frame and explore its structure and contents.
* The first few lines of code import necessary libraries (NumPy and Matplotlib) and load the dataset ('cancer\_data.csv').
* **Data Preprocessing**
* The code extracts the input features (X) and target variable (y) from the loaded dataset. The input features are all columns except the last one, and the target variable is the last column.
* The code then normalizes the input features using the standardization technique, which involves subtracting the mean of each feature and dividing by its standard deviation. The code also prints the mean and standard deviation of the normalized features to check if they are centered on 0 and have a standard deviation of 1.
* The code then adds a column of ones to the input features matrix X to account for the bias term in linear regression.
* The code defines three functions for linear regression: predict\_linear\_regression, which predicts the target variable given the input features and model parameters; cost\_linear\_regression, which calculates the cost (mean squared error) of the model given the input features, target variable, and model parameters; and gradient\_linear\_regression, which calculates the gradient of the cost function with respect to the model parameters.
* The code defines a function for gradient descent optimization, which updates the model parameters (theta) iteratively to minimize the cost function using the calculated gradient. The function takes in the input features, target variable, learning rate (alpha), and number of iterations as inputs.
* The code then runs gradient descent optimization for different values of learning rate (alpha) and plots the cost (mean squared error) over iterations for each value of alpha.
* The code defines a function for mini-batch gradient descent optimization, which updates the model parameters (theta) iteratively using a random subset (batch) of the input features and target variable to minimize the cost function using the calculated gradient. The function takes in the input features, target variable, learning rate (alpha), number of iterations, and batch size as inputs.
* The code then runs mini-batch gradient descent optimization for different values of batch size and plots the cost (mean squared error) over iterations for each value of batch size.
* The code defines a function for Adam optimization, which updates the model parameters (theta) iteratively using adaptive estimates of the first and second moments of the gradient to minimize the cost function. The function takes in the initial model parameters (theta), input features, target variable, learning rate (alpha), number of iterations, and hyper parameters (epsilon, beta1, and beta2) as inputs.
* The code then runs Adam optimization and plots the cost (mean squared error) over iterations.
* **Model Evaluation**
* We evaluate the performance of the model by computing the mean squared error and coefficient of determination (R^2) on the testing data.
* **Results**
* We present the results of the linear regression and evaluate its performance using the mean squared error and coefficient of determination. We also plot the predicted values against the true values to visualize the performance of the model.
* **Conclusion**
* The cost function decreases faster with smaller batch sizes. Smaller batch sizes introduce more noise in the gradient estimation, which can help escape from local minima and converge to a better solution. However, smaller batch sizes also increase the variance of the gradient estimation, which can make the algorithm converge to a suboptimal solution if the learning rate is not adjusted accordingly.
* Based on the graph, we can see that Adam converges faster than Mini-Batch Gradient Descent for all batch sizes. This is because Adam adapts the learning rate for each parameter based on the magnitude of its gradient and its historical variance, which allows it to converge faster and more reliably.



