

Assignment Report

Scalable Data Mining

⋮ Assignment 1: Optimizers

Hardik Soni

20CS30023

27th August, 2023

Introduction


1. **Task::** The aim of this assignment is to train and test your own custom-made model for a regression task on Boston Housing Dataset using PyTorch nn.Module.
2. **Data::** The Boston Housing Dataset is derived from information collected by the U.S. Census Service concerning housing in the area of Boston. The following describes the dataset columns: There are
 - CRIM - per capita crime rate by town
 - ZN - the proportion of residential land zoned for lots over 25,000 sq.ft. INDUS - proportion of non-retail business acres per town.
 - CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise) NOX - nitric oxides concentration (parts per 10 million)
 - RM - the average number of rooms per dwelling
 - AGE - the proportion of owner-occupied units built prior to 1940
 - DIS - weighted distances to five Boston employment centers
 - RAD - index of accessibility to radial highways
 - TAX - full-value property-tax rate per \$10,000
 - PTRATIO - pupil-teacher ratio by town
 - B - $1000(B_k - 0.63)^2$ where B_k is the proportion of blacks by town
 - LSTAT - % lower status of the population
 - MEDV - Median value of owner-occupied homes in \$1000's

506 data points in the dataset. MEDV is the target variable.

3. Basic Ideology::


1. Model:-

- 1.1. A **2-layer neural network for performing regression** is a relatively simple yet effective model used to predict continuous numerical values. In this context, a "layer" refers to a group of neurons with shared properties, and a 2-layer network includes an input layer, one hidden layer, and an output layer. While the term "2-layer" might seem misleading due to the presence of only one hidden layer, it's commonly referred to as such because the input layer is not considered when counting the layers.

- 
- 1.2. Here's a breakdown of the key components and concepts of a 2-layer neural network for regression:
- 1.2.1. **Input Layer:** The input layer of the network consists of neurons corresponding to the input features of your dataset. Each neuron in this layer represents a specific feature. For regression tasks, the input layer doesn't have an activation function; it simply passes the input data to the hidden layer.
 - 1.2.2. **Hidden Layer:** The hidden layer is where the network learns complex patterns and representations from the input data. It consists of a set of neurons, each connected to all neurons from the input layer. This layer applies a linear transformation to the inputs followed by an activation function like ReLU (Rectified Linear Unit) or Sigmoid.
 - 1.2.2.1. - *Linear Transformation:* Each neuron in the hidden layer performs a weighted sum of its inputs (from the input layer), which is similar to a dot product between the input features and their associated weights. These weighted sums represent the importance of each input feature for predicting the target.
 - 1.2.2.2. - *Activation Function:* After the linear transformation, an activation function is applied element-wise to the weighted sum. Activation functions introduce non-linearity to the network, allowing it to model complex relationships between inputs and outputs. Common activation functions for regression tasks are ReLU and Sigmoid.
 - 1.2.3. **Output Layer:** The output layer consists of a single neuron (since this is a regression task), which produces the predicted continuous numerical value. There's no activation function applied in the output layer, as the regression task aims to predict a continuous value directly.

2. Optimizers:-

- 2.1. **Stochastic Gradient Descent (SGD):-** is an optimization algorithm commonly used to train machine learning models, including neural networks. It's an iterative optimization technique that aims to find the optimal parameters (weights and biases) of a model to minimize



a given loss function. SGD is particularly well-suited for large datasets because it processes individual data points (or small batches) at a time, which reduces memory requirements and speeds up convergence.

2.2. In traditional Gradient Descent, each iteration of the algorithm computes the gradient of the loss over the entire dataset. This can be computationally expensive, especially for large datasets. Stochastic Gradient Descent (SGD) improves upon this by randomly selecting a single data point (or a small batch of data points) for each iteration.

2.3. Let's break down the explanation of **Stochastic Gradient Descent (SGD) optimizer with the Nesterov momentum parameter** set to True and an initial momentum value.

2.3.1. Stochastic Gradient Descent (SGD) Overview: Stochastic Gradient Descent is an optimization algorithm used to train machine learning models, especially in the context of neural networks. It's an iterative method that updates the model's parameters in the opposite direction of the gradient of the loss function, with respect to the parameters. This process aims to minimize the loss and improve the model's performance.

2.3.2. **Momentum in Optimization**: Momentum is a technique used to accelerate the convergence of optimization algorithms. It introduces a memory of the past gradients to make updates smoother and less likely to get stuck in shallow local minima. Momentum enhances the gradient descent process by adding a fraction of the previous update to the current update.

2.3.3. **Nesterov Accelerated Gradient (NAG)**: Nesterov Accelerated Gradient (NAG), also known as Nesterov Momentum or Nesterov Accelerated Descent (NAD), is an enhancement of the momentum technique in optimization. It involves taking a "look-ahead" step by using the momentum to estimate the future position of the parameters before calculating the gradient. This can lead to more accurate updates and faster convergence, especially when the optimizer is near a minimum.

- 2.3.4. Explanation of Nesterov Momentum with SGD: When using Nesterov Momentum with SGD, the optimizer updates the model parameters as follows:
 - 2.3.4.1. Compute the gradient of the loss with respect to the current parameters. Update the velocity (momentum) by applying a fraction of the previous velocity and subtracting the gradient multiplied by the learning rate.
 - 2.3.4.2. Update the parameters by adding the current velocity to the parameters.
- 2.3.5. Initial Momentum Value: The initial momentum value is the momentum term applied during the first iteration of optimization. It helps to initialize the velocity in a specific direction to aid in convergence. It's common to start with a small value (e.g., 0.5 or 0.9) to avoid overshooting.
- 2.3.6. In summary, using the Stochastic Gradient Descent (SGD) optimizer with the Nesterov parameter set to True and an initial momentum value enhances the convergence speed and stability of the optimization process. Nesterov Momentum allows the optimizer to look ahead before calculating gradients, leading to improved convergence near minima. The initial momentum value sets the initial velocity of the updates, helping to avoid overshooting and achieving smoother convergence.
- 2.4. Adadelta is an optimization algorithm commonly used for training machine learning models, especially deep neural networks. It's an adaptive learning rate optimization algorithm that addresses some limitations of traditional optimization algorithms like Stochastic Gradient Descent (SGD) and its variants. Adadelta adjusts the learning rate for each parameter individually based on the historical gradient information, allowing it to handle different scales of gradients and parameters effectively. Here's an explanation of the Adadelta optimizer and its key concepts:
 - 2.4.1. Adaptive Learning Rate: Adadelta's main advantage lies in its ability to adaptively adjust the learning rate for each parameter during training. Unlike traditional methods where a global learning rate is used for all parameters, Adadelta

automatically scales the learning rate based on the recent history of gradients.

- 2.4.2. Exponential Moving Averages: Adadelata uses exponential moving averages of squared gradients to calculate the root mean squared (RMS) gradient for each parameter. These averages help in normalizing the gradients and addressing issues of varying gradient scales.
- 2.4.3. Accumulated Gradient and Update History: At each iteration, Adadelata accumulates the squared gradients over time and maintains an update history of past gradients. This history smooths out the noise in the gradient updates.
- 2.4.4. RMS Propagation: For each parameter, Adadelata divides the current gradient by the root mean squared gradient from the update history. This division essentially scales the learning rate by the gradient's magnitude, enabling a larger step size for smaller gradients and a smaller step size for larger gradients.
- 2.4.5. Delta Update Rule: Adadelata introduces a new concept called the "delta" (Δ). This delta represents an accumulated change in parameter values. It's used to update parameters instead of directly using the gradients.
- 2.4.6. Decay Factor and Epsilon: Adadelata uses a decay factor (usually denoted as ρ or "rho") to control how much past gradient history influences the current update. A small value like 0.9 is often used. Additionally, Adadelata includes a small constant ϵ (epsilon) to prevent division by zero in the update rule.
- 2.4.7. Advantages of Adadelata: - Adapts the learning rate on a per-parameter basis.

- Automatically adjusts the learning rate as training progresses, reducing the need for manual tuning.

- Handles varying gradient scales well, making it suitable for complex models.

- Reduces the need for setting an initial learning rate.

- Tends to converge more robustly than standard SGD or even variants like RMSProp.


- 2.4.8. Usage in Deep Learning: Adadelta is popularly used in training deep neural networks, where the choice of optimization algorithm plays a crucial role in achieving efficient convergence and generalization. In summary, the Adadelta optimizer is an adaptive learning rate optimization algorithm that uses the history of gradients and accumulated update history to adjust the learning rate for each parameter. This adaptability makes it particularly well-suited for training complex models with different scales of gradients and parameters.

Do you observe any differences between the different training configurations ?

Based on the configurations you've described, you are planning to experiment with different optimization algorithms and hyperparameters for training a 2-layer neural network on the Boston Housing Dataset for regression tasks. You'll be using both unnormalized and normalized data for your experiments. Let's go through each part and discuss potential differences you might observe:

Part A: With Unnormalized Data

- 1) **SGD Optimizer with Different Learning Rates:** You will be training the model using the Stochastic Gradient Descent (SGD) optimizer with different learning rates (0.1, 0.01, 0.001) and MSE loss as the loss function. Generally, higher learning rates might result in faster initial convergence, but they can also cause the loss to oscillate or diverge. Smaller learning rates might lead to slower convergence but potentially more stable learning. You might observe that the choice of learning rate affects the training time and the quality of convergence.
- 2) **SGD Optimizer with Nesterov Momentum:** By using the Nesterov momentum with SGD, you're adding a look-ahead mechanism to the updates. This could lead to faster convergence compared to regular SGD. The initial momentum value also affects the convergence speed and



stability. You might observe improved convergence behavior compared to plain SGD

- 3) **Adadelata Optimizer:** Adadelata adapts the learning rates individually based on accumulated historical gradients. It aims to handle varying gradient scales more effectively. With Adadelata, you might observe smoother convergence due to its adaptability to the gradient landscape.

Part B: With Normalized Data

Normalization Impact: Using mean-variance normalization can help in improving the convergence behavior of optimization algorithms. It can also aid in handling features with different scales. Normalization ensures that features have similar magnitudes, which can lead to better convergence rates and more stable updates. You might observe faster and more consistent convergence compared to unnormalized data.

Observations and Differences:

The differences you observe will depend on a combination of factors, including the optimization algorithm, learning rates, use of Nesterov momentum, data normalization, and the specific characteristics of the Boston Housing Dataset. Some potential observations could include:

- Faster convergence with normalized data due to reduced scale differences.
- Different learning rates leading to varying convergence speed and quality.
- Nesterov momentum potentially improving convergence behavior.
- Adadelata showing smoother convergence compared to SGD.
- Unnormalized data might require careful tuning of learning rates to avoid divergence.

It's important to experiment with different configurations and observe the learning curves, loss values, and model performance on a validation or test set. Keep in mind that the best configuration might vary depending on the dataset and problem at hand. It's

recommended to perform a grid search or use techniques like learning rate schedulers to find optimal hyperparameters.

Training Losses

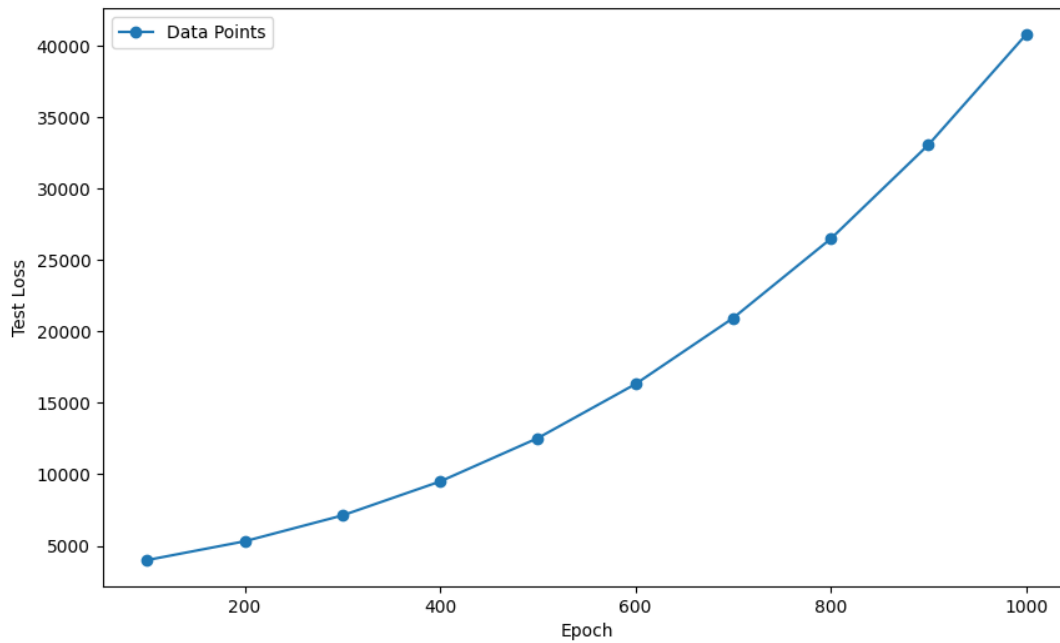
Learning Rates	SGD		SGD with Nesterov and Momentum Value - 0.9		Adadelta	
	Normalised	Unnormalized	Normalised	Unnormalized	Normalised	Unnormalized
0.1	NaN	50.4324	NaN	50.4324	8.1799	88.7176
0.01	7.2000	57.2119	2.6475	50.4324	18.2261	28.4567
0.01	13.0754	20917678080.0000	6.3808	50.4324	172.5328	27.4248

Testing Losses

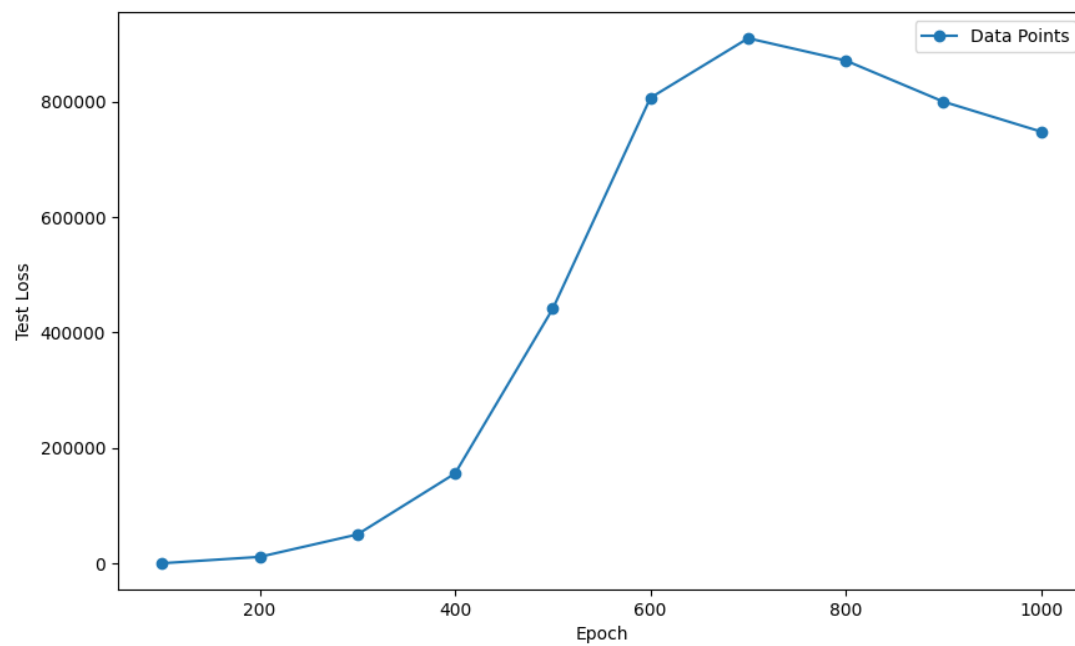
Learning Rates	SGD		SGD with Nesterov and Momentum Value - 0.9		Adadelta	
	Normalised	Unnormalized	Normalised	Unnormalized	Normalised	Unnormalized
0.1	NaN	52.9123	NaN	52.9123	21.3035	95.8364
0.01	21.6527	54.4687	26.8175	52.9123	19.2268	22.9567
0.01	16.9829	20833808384.0000	19.0011	52.9123	195.5520	21.5525

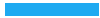
Plots

Testing Loss VS. Epoch with Adaptive Delta Loss Function and Normalized Data and Learning Rate: 0.0010

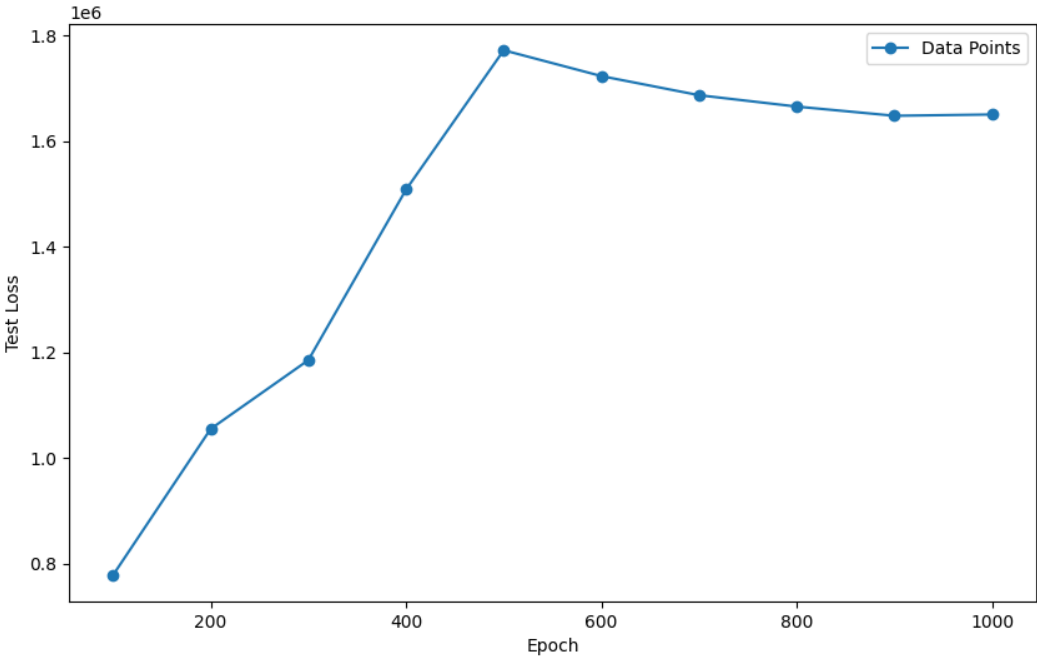


Testing Loss VS. Epoch with Adaptive Delta Loss Function and Normalized Data and Learning Rate: 0.0100

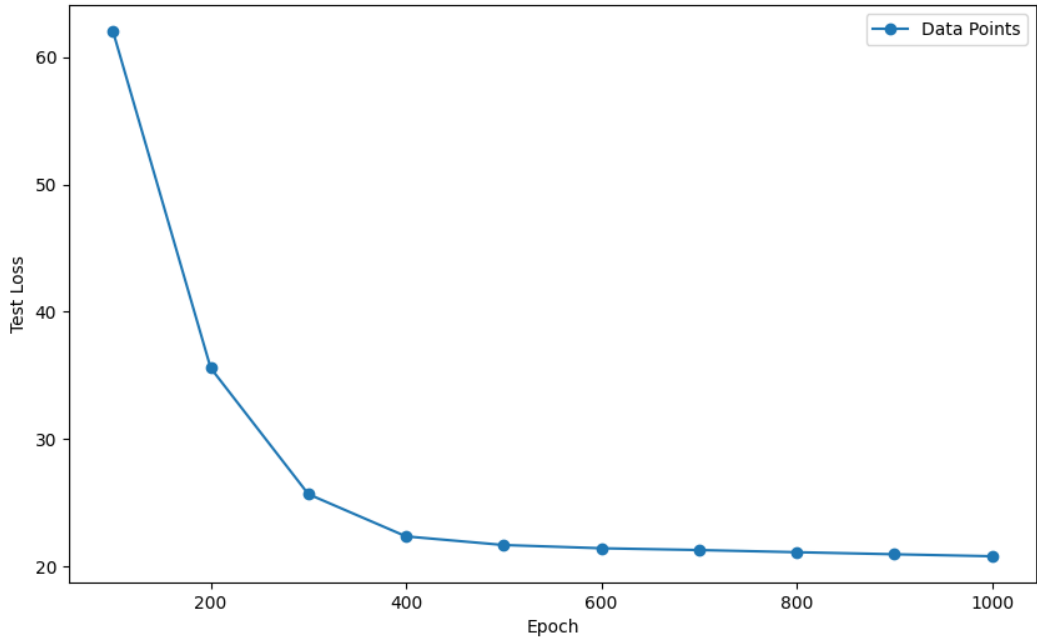




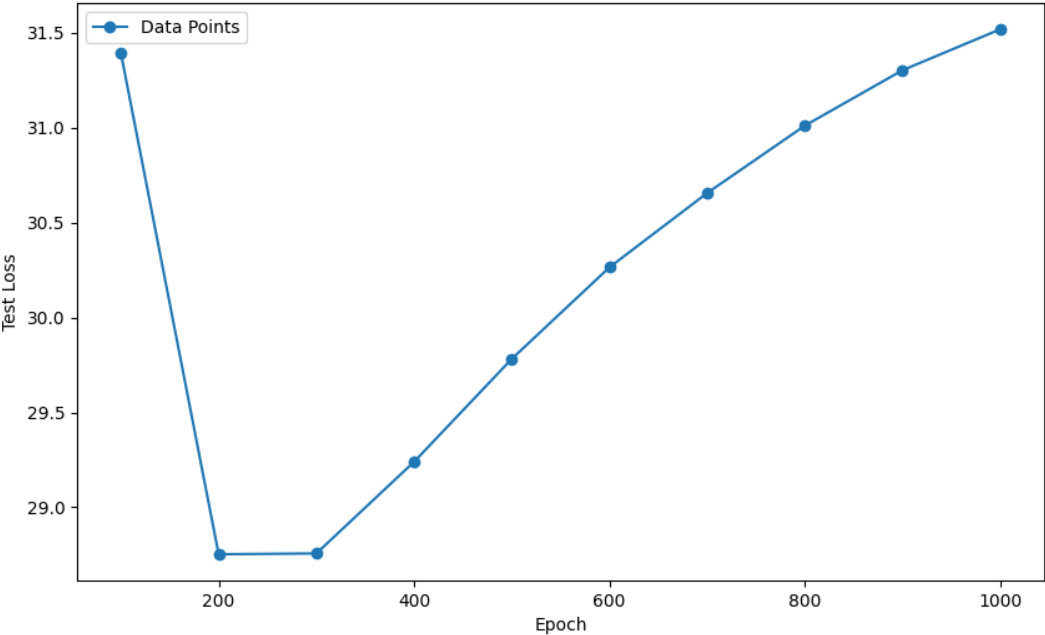
Testing Loss VS. Epoch with Adaptive Delta Loss Function and Normalized Data and Learning Rate: 0.1000



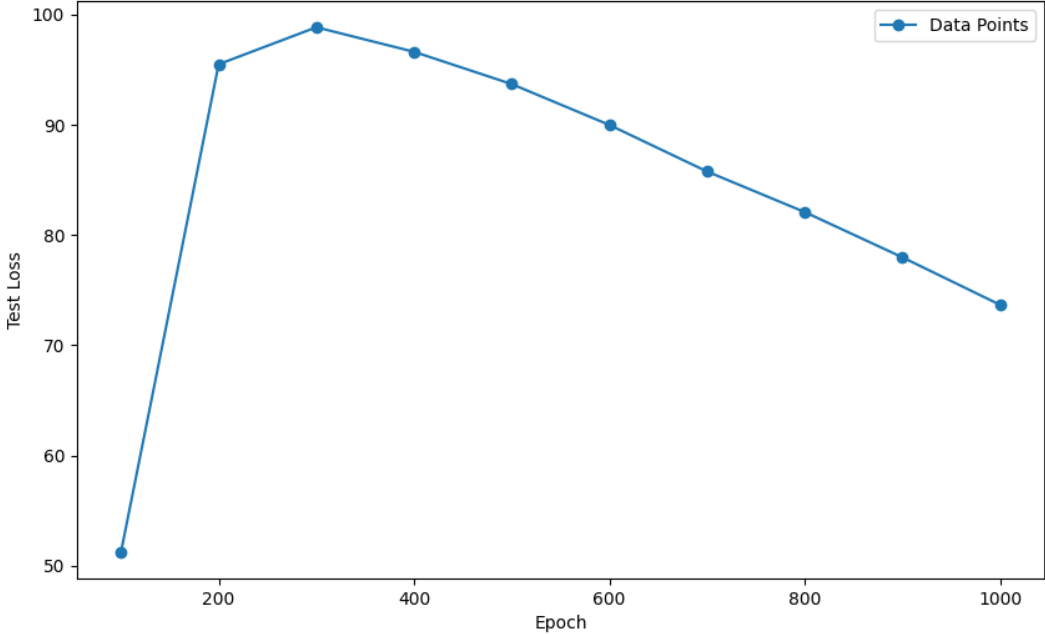
Testing Loss VS. Epoch with Adaptive Delta Loss Function and Unnormalized Data and Learning Rate: 0.0010



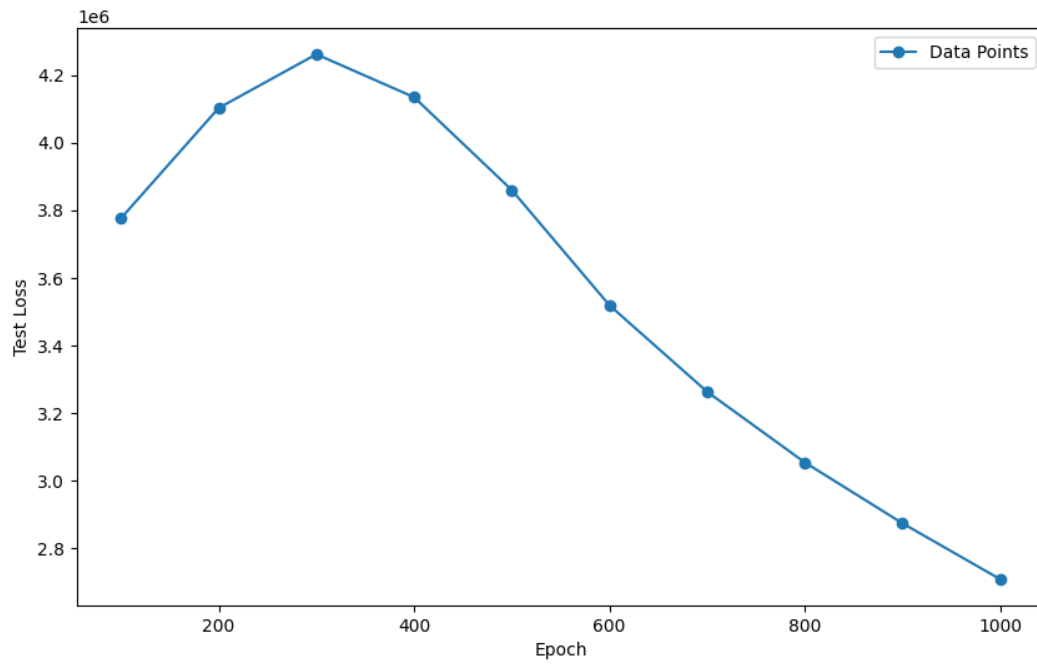
Testing Loss VS. Epoch with Adaptive Delta Loss Function and Unnormalized Data and Learning Rate: 0.0100



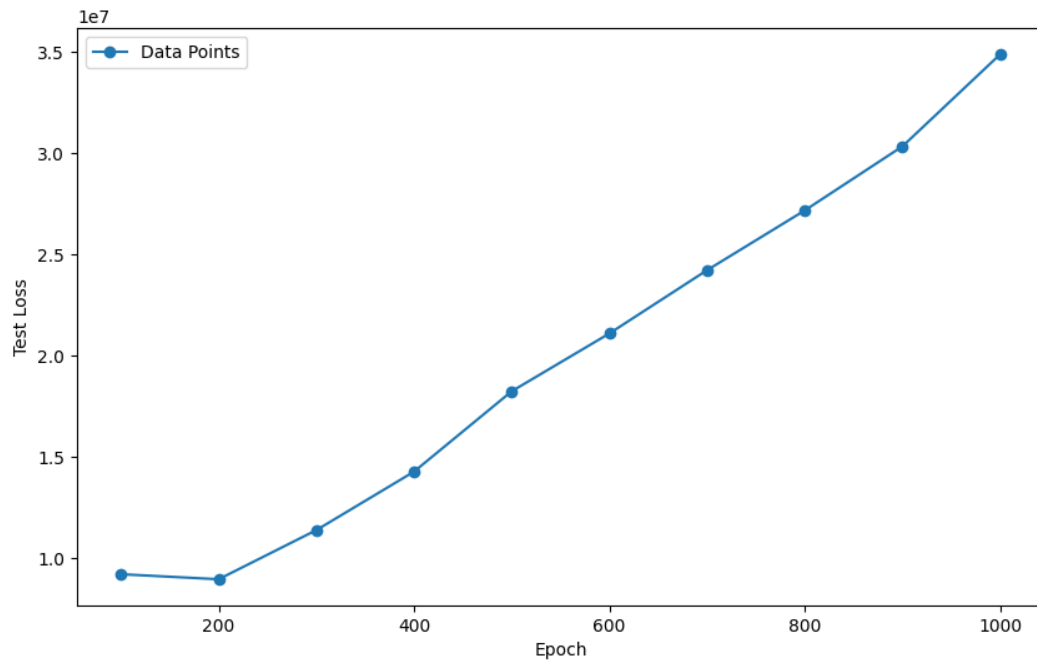
Testing Loss VS. Epoch with Adaptive Delta Loss Function and Unnormalized Data and Learning Rate: 0.1000



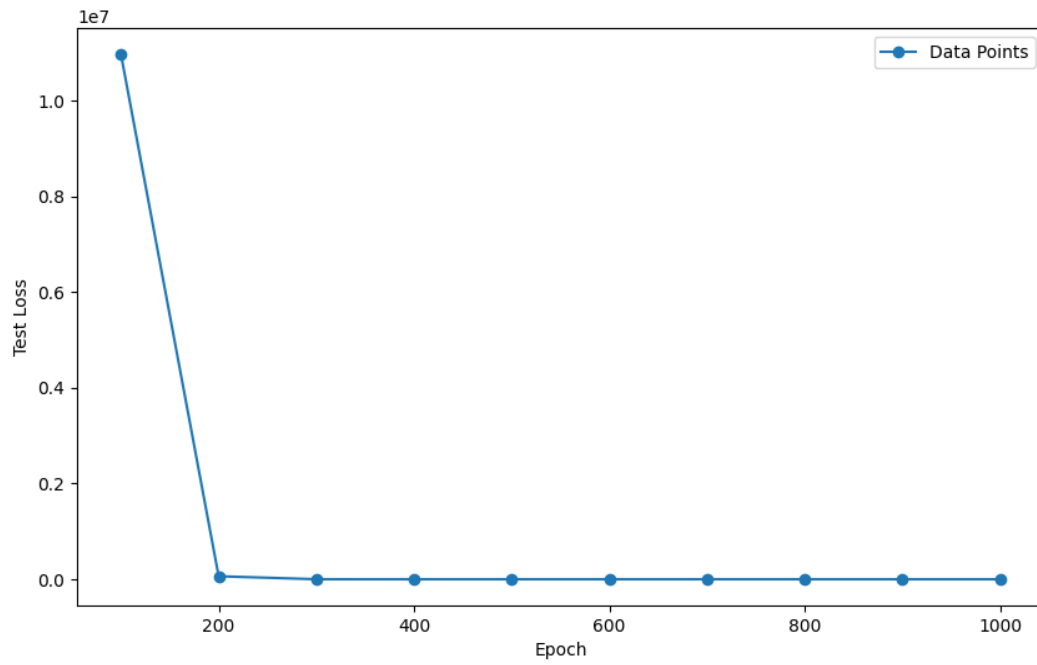
s VS. Epoch with Stochastic Gradient Descent Loss Function with Nesterov parameter and Normalized Data and Learning



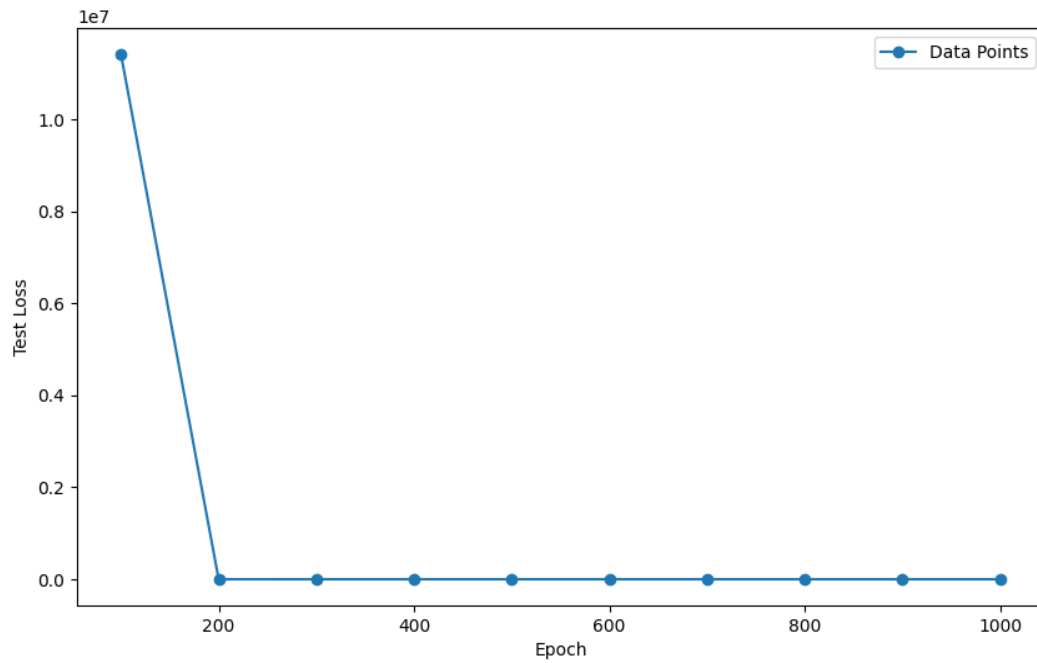
s VS. Epoch with Stochastic Gradient Descent Loss Function with Nesterov parameter and Normalized Data and Learning



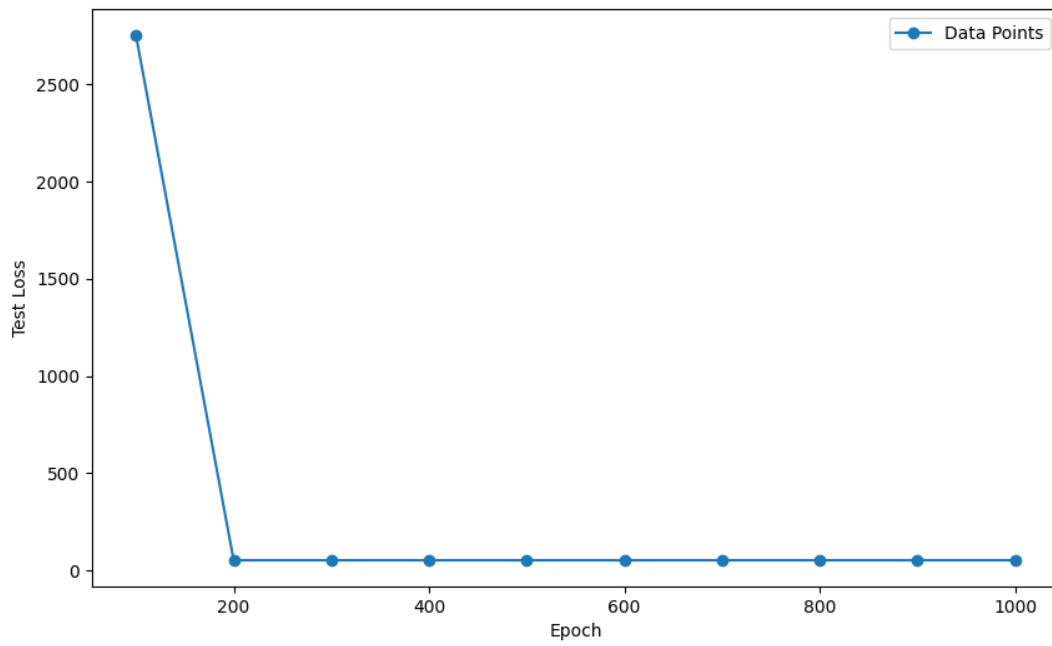
; VS. Epoch with Stochastic Gradient Descent Loss Function with Nesterov parameter and Unnormalized Data and Learnin



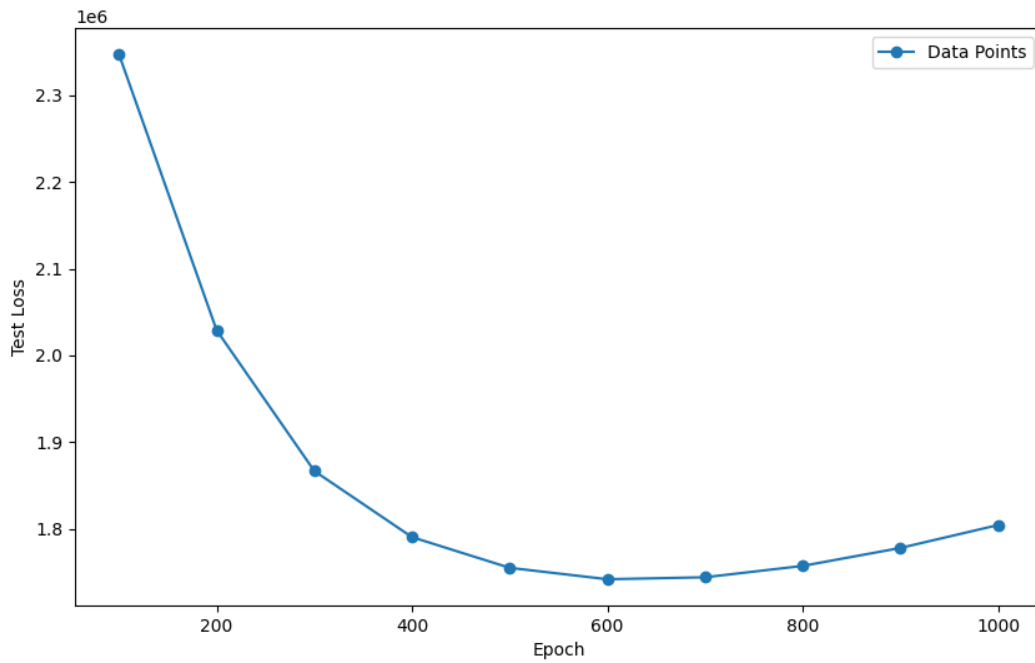
; VS. Epoch with Stochastic Gradient Descent Loss Function with Nesterov parameter and Unnormalized Data and Learnin



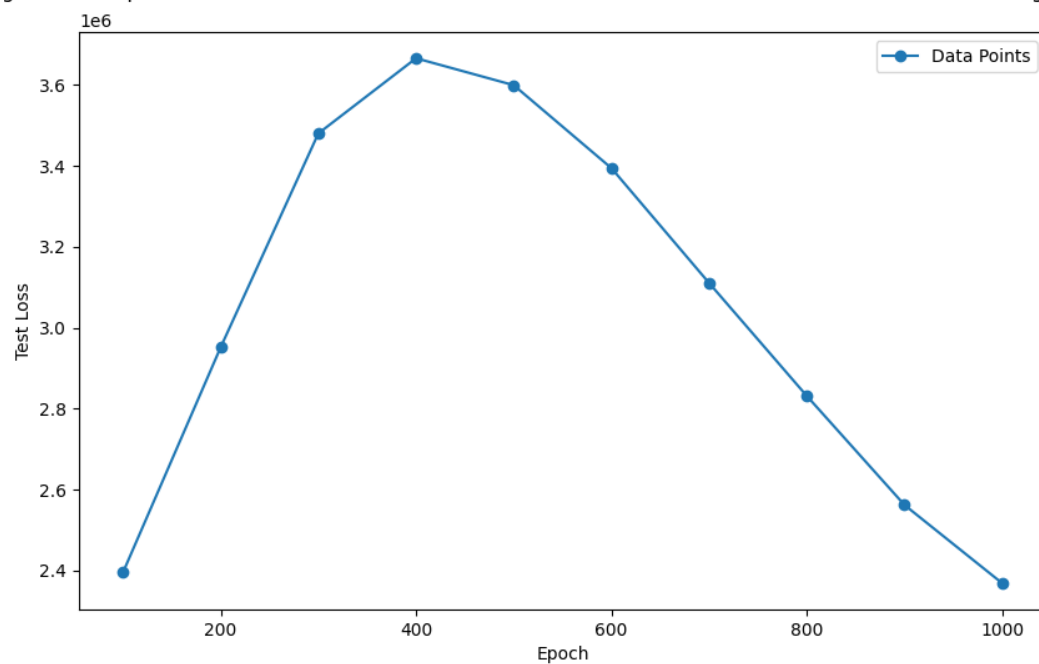
; VS. Epoch with Stochastic Gradient Descent Loss Function with Nesterov parameter and Unnormalized Data and Learnin



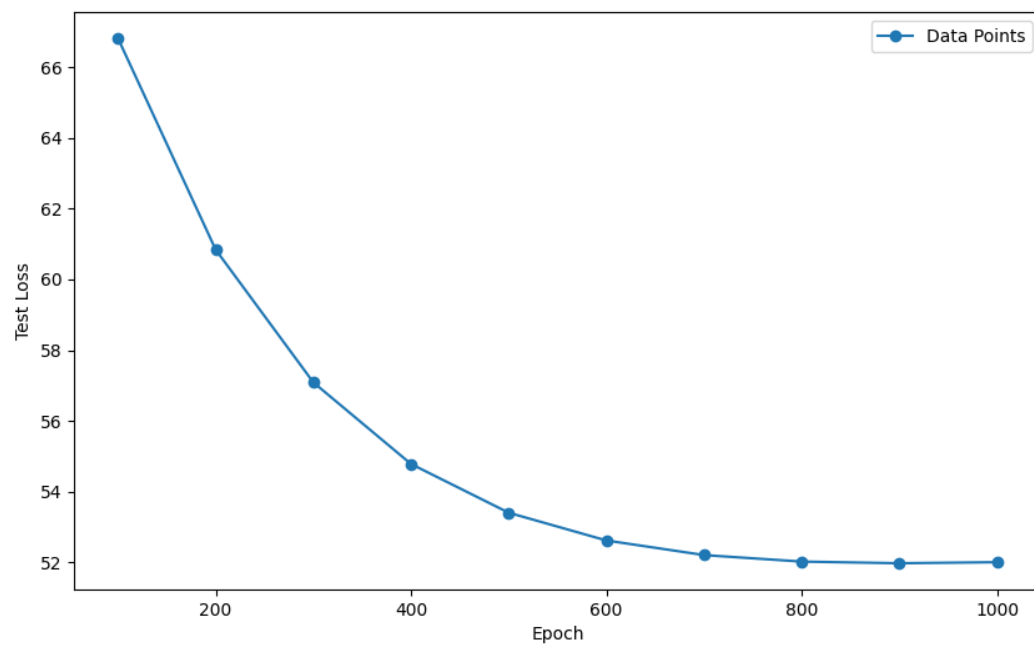
Testing Loss VS. Epoch with Stochastic Gradient Descent Loss Function and Normalized Data and Learning Rate: 0.001



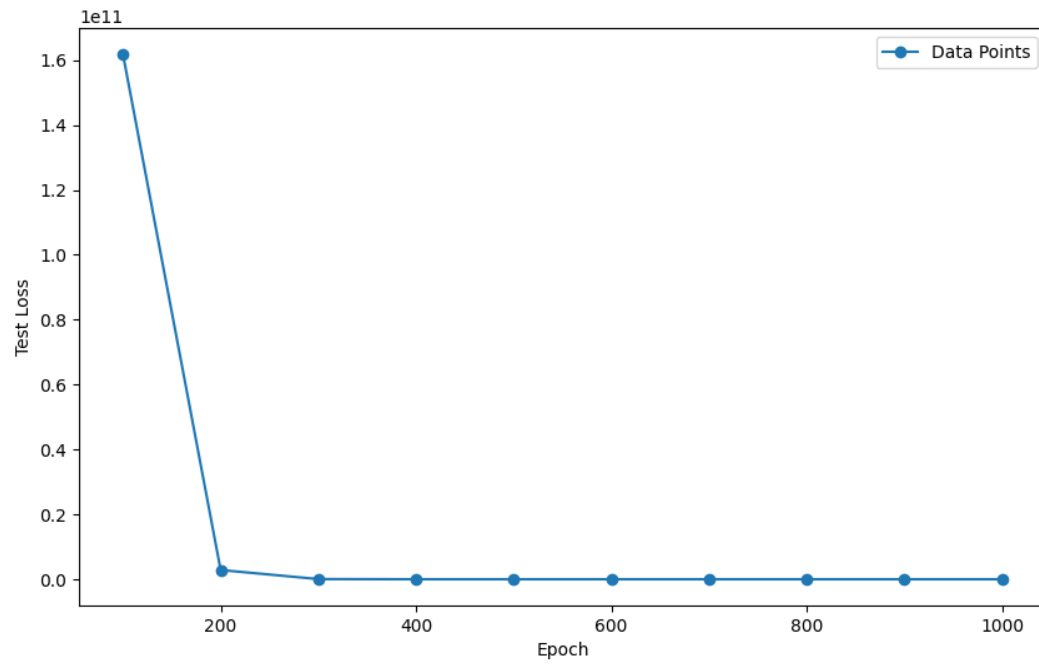
Testing Loss VS. Epoch with Stochastic Gradient Descent Loss Function and Normalized Data and Learning Rate: 0.010



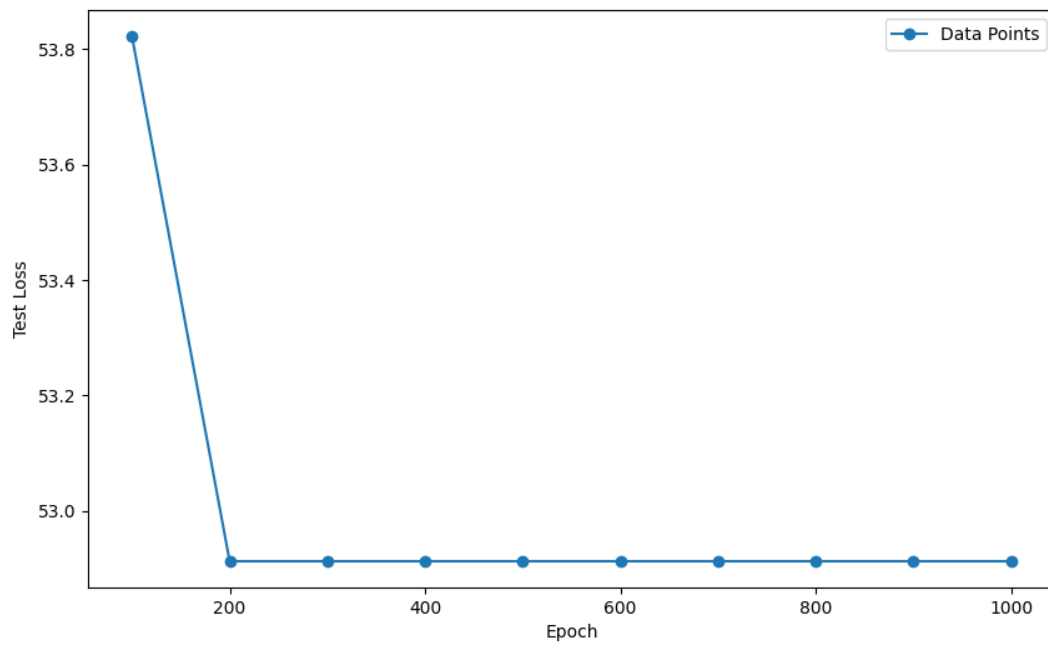
Testing Loss VS. Epoch with Stochastic Gradient Descent Loss Function and Unnormalized Data and Learning Rate: 0.00



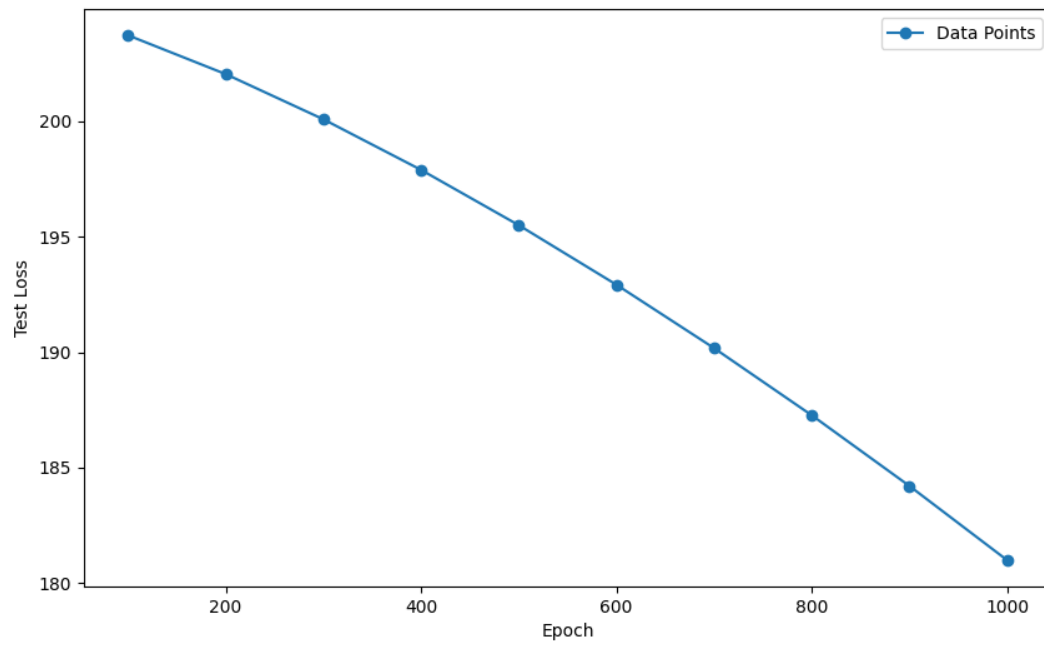
Testing Loss VS. Epoch with Stochastic Gradient Descent Loss Function and Unnormalized Data and Learning Rate: 0.01



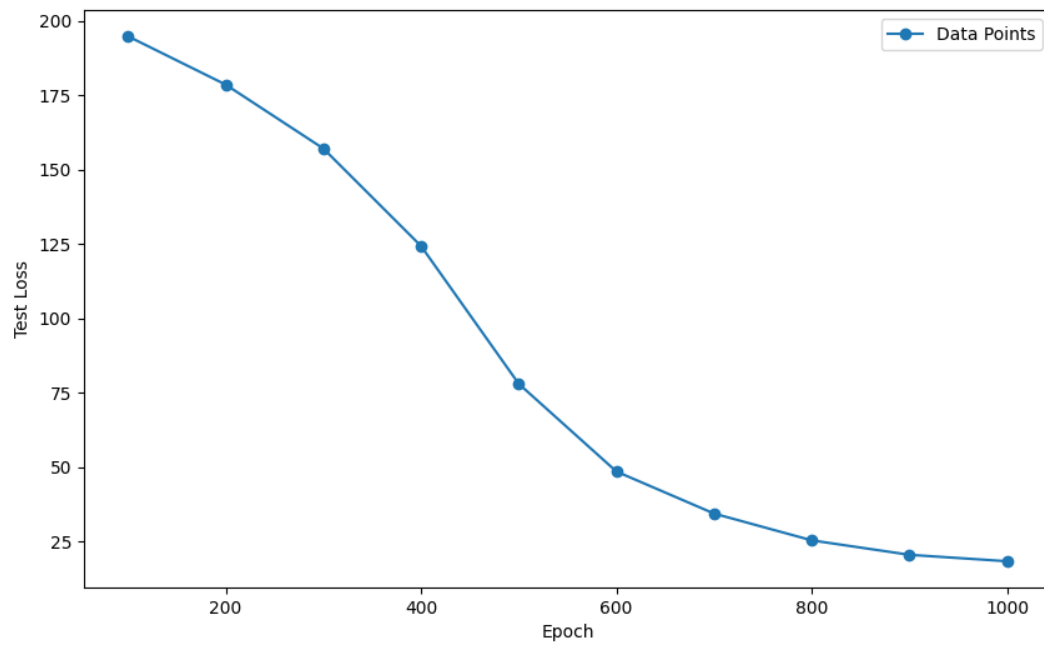
Testing Loss VS. Epoch with Stochastic Gradient Descent Loss Function and Unnormalized Data and Learning Rate: 0.10



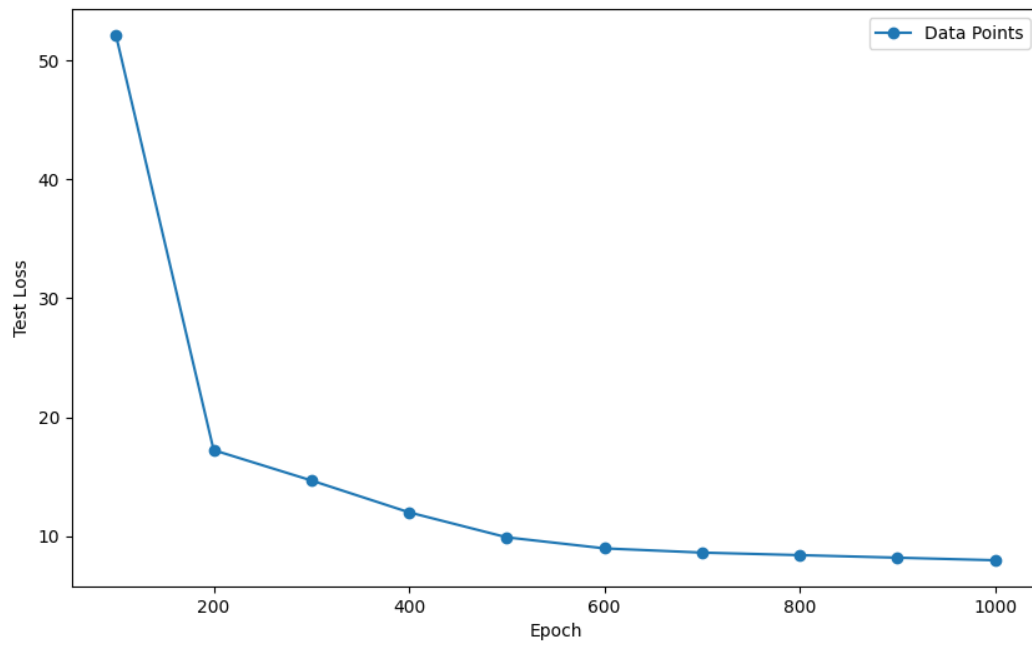
Training Loss VS. Epoch with Adaptive Delta Loss Function and Normalized Data and Learning Rate: 0.0010



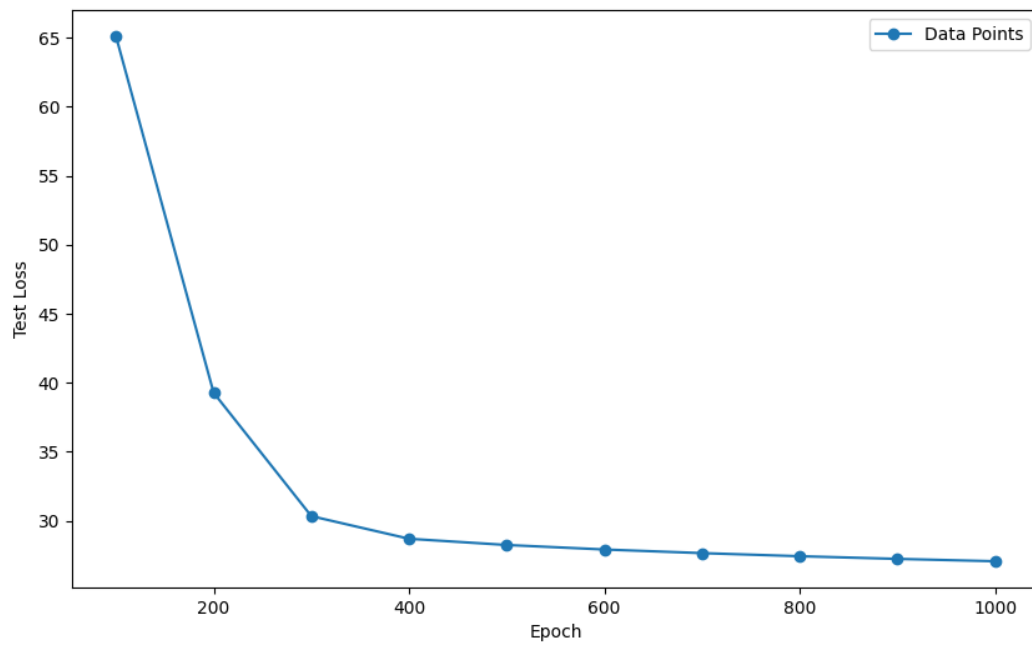
Training Loss VS. Epoch with Adaptive Delta Loss Function and Normalized Data and Learning Rate: 0.0100



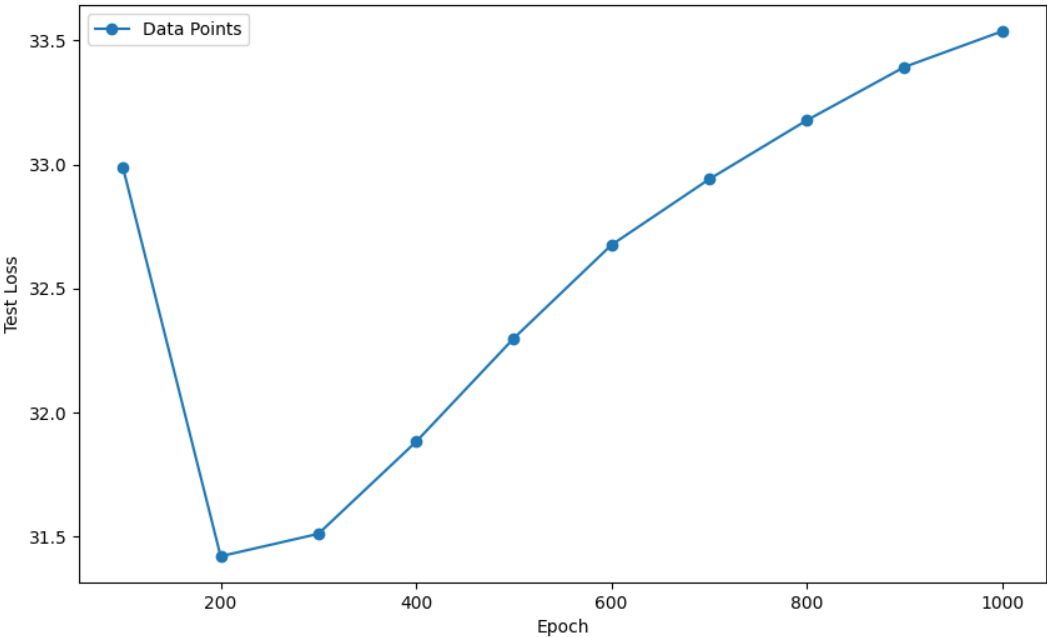
Training Loss VS. Epoch with Adaptive Delta Loss Function and Normalized Data and Learning Rate: 0.1000



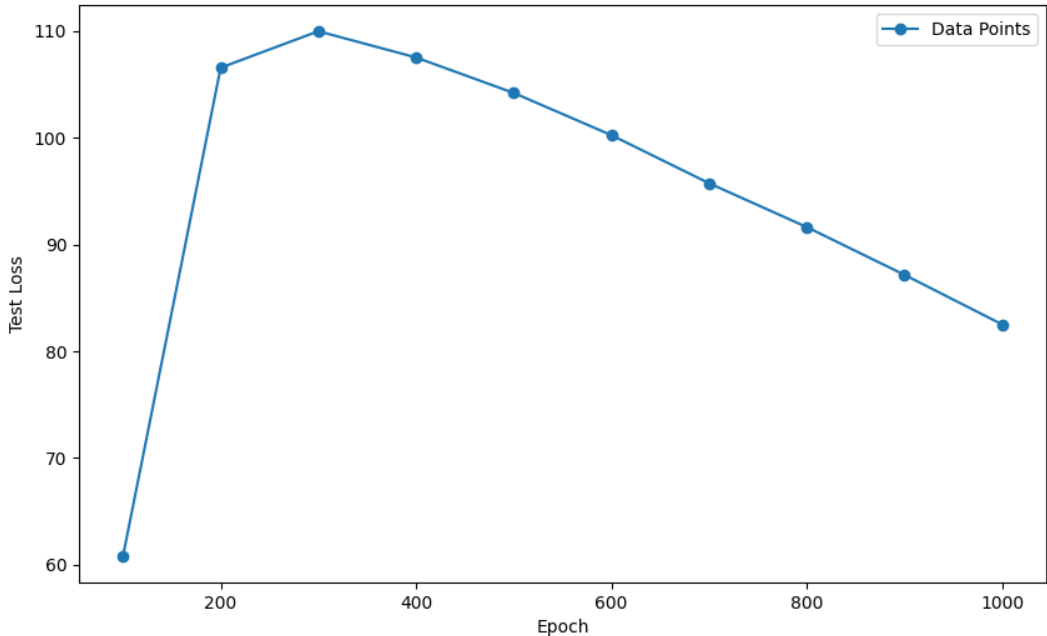
Training Loss VS. Epoch with Adaptive Delta Loss Function and Unnormalized Data and Learning Rate: 0.0010



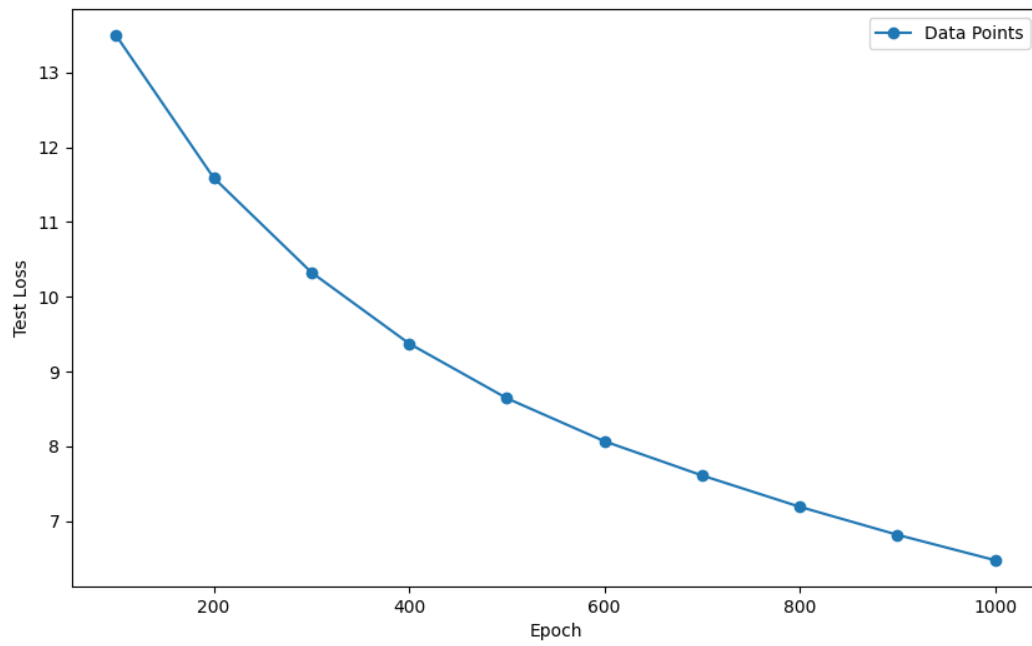
Training Loss VS. Epoch with Adaptive Delta Loss Function and Unnormalized Data and Learning Rate: 0.0100



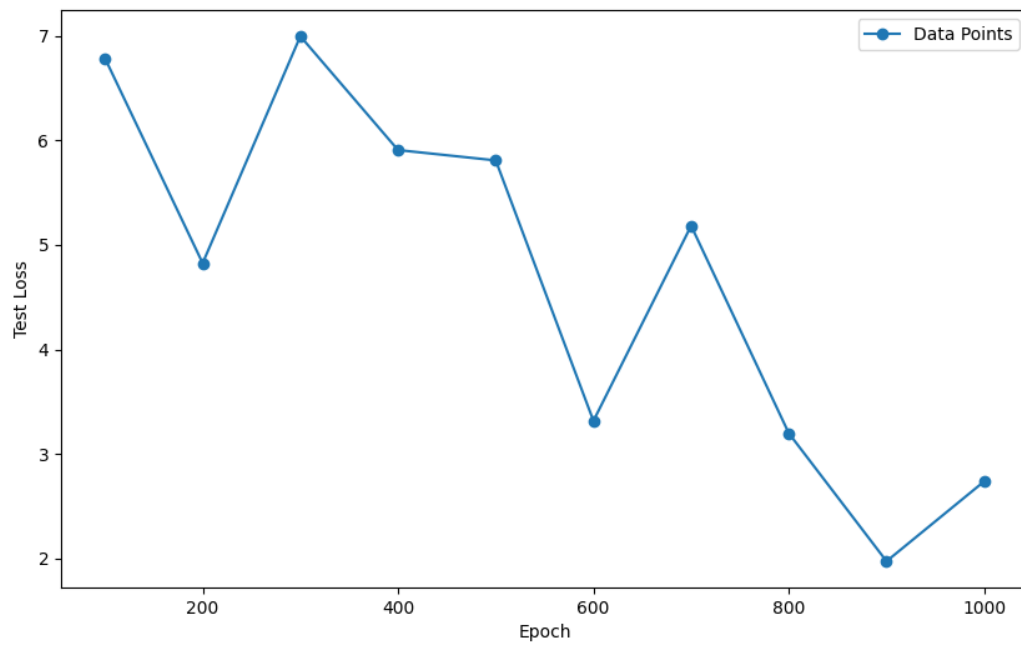
Training Loss VS. Epoch with Adaptive Delta Loss Function and Unnormalized Data and Learning Rate: 0.1000



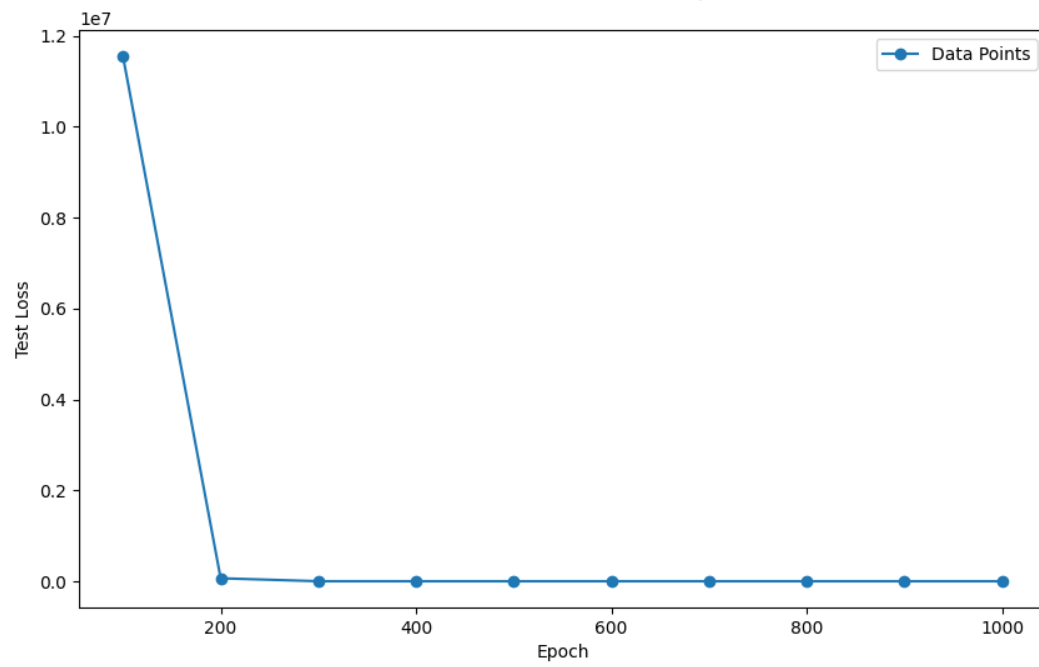
is VS. Epoch with Stochastic Gradient Descent Loss Function with Nesterov parameter and Normalized Data and Learning



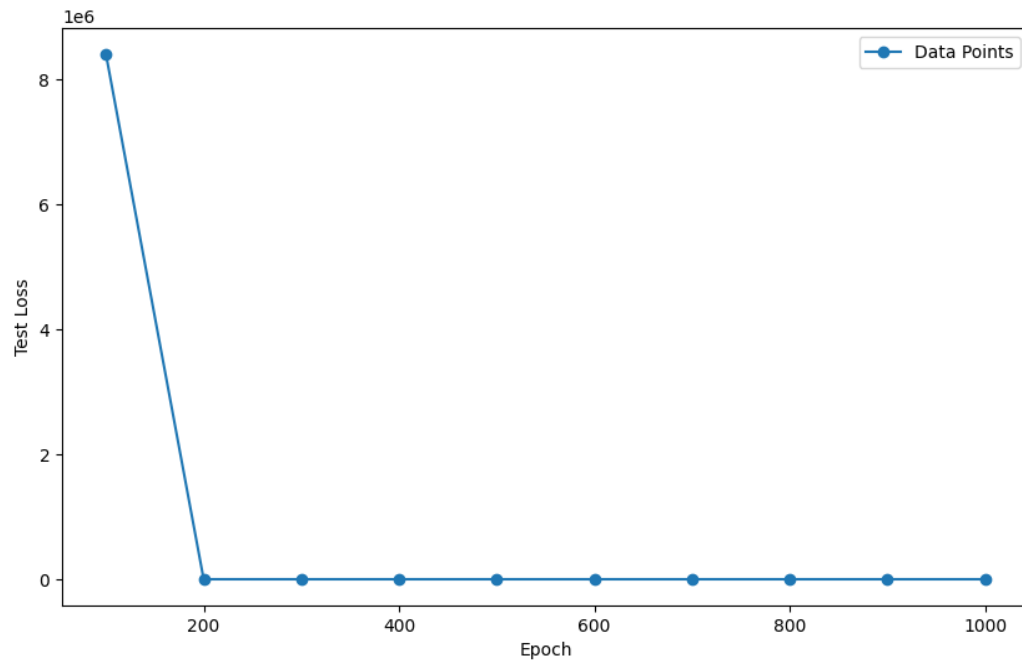
is VS. Epoch with Stochastic Gradient Descent Loss Function with Nesterov parameter and Normalized Data and Learning



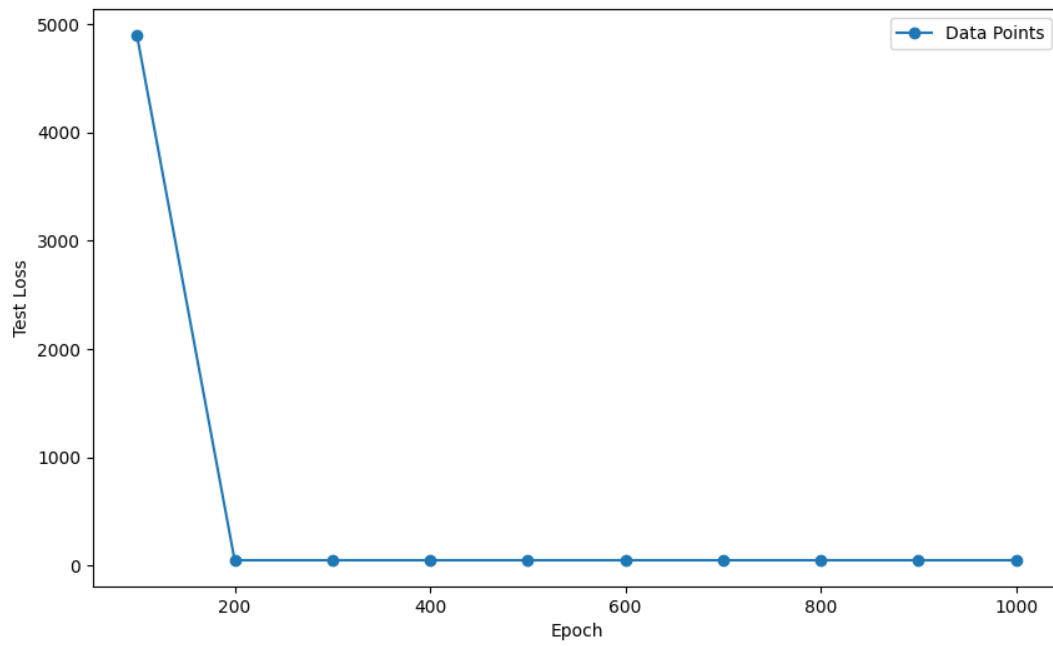
s VS. Epoch with Stochastic Gradient Descent Loss Function with Nesterov parameter and Unnormalized Data and Learning



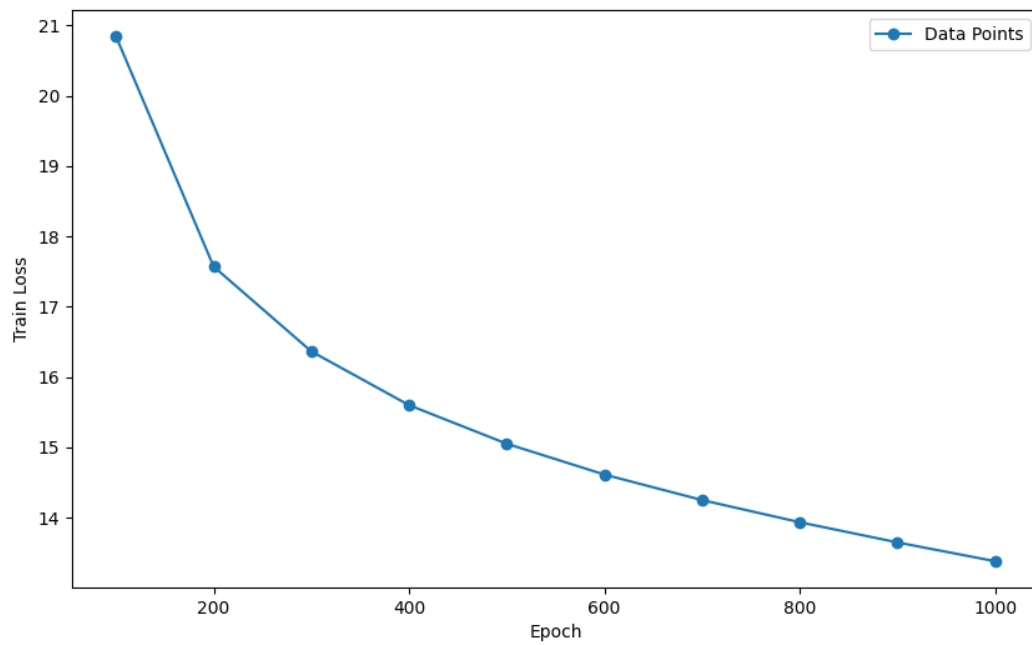
s VS. Epoch with Stochastic Gradient Descent Loss Function with Nesterov parameter and Unnormalized Data and Learning



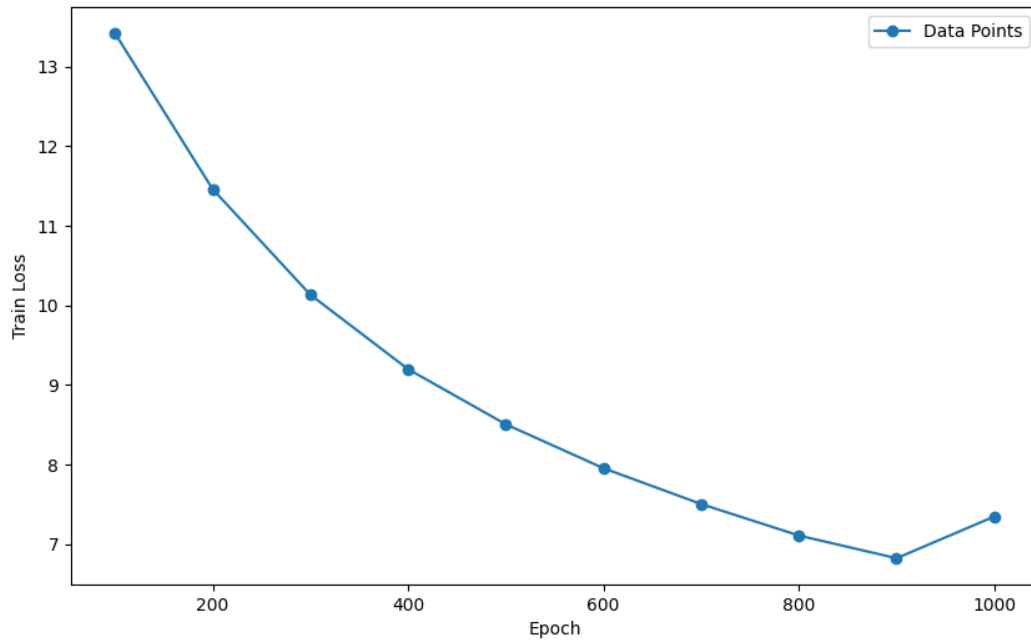
s VS. Epoch with Stochastic Gradient Descent Loss Function with Nesterov parameter and Unnormalized Data and Learning



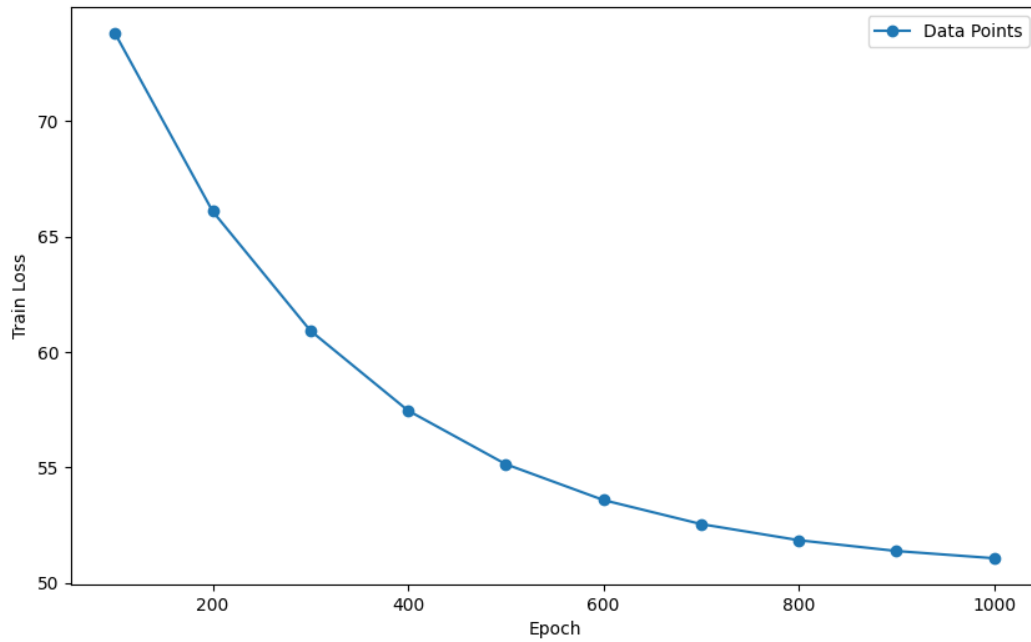
Training Loss VS. Epoch with Stochastic Gradient Descent Loss Function and Normalized Data and Learning Rate: 0.001



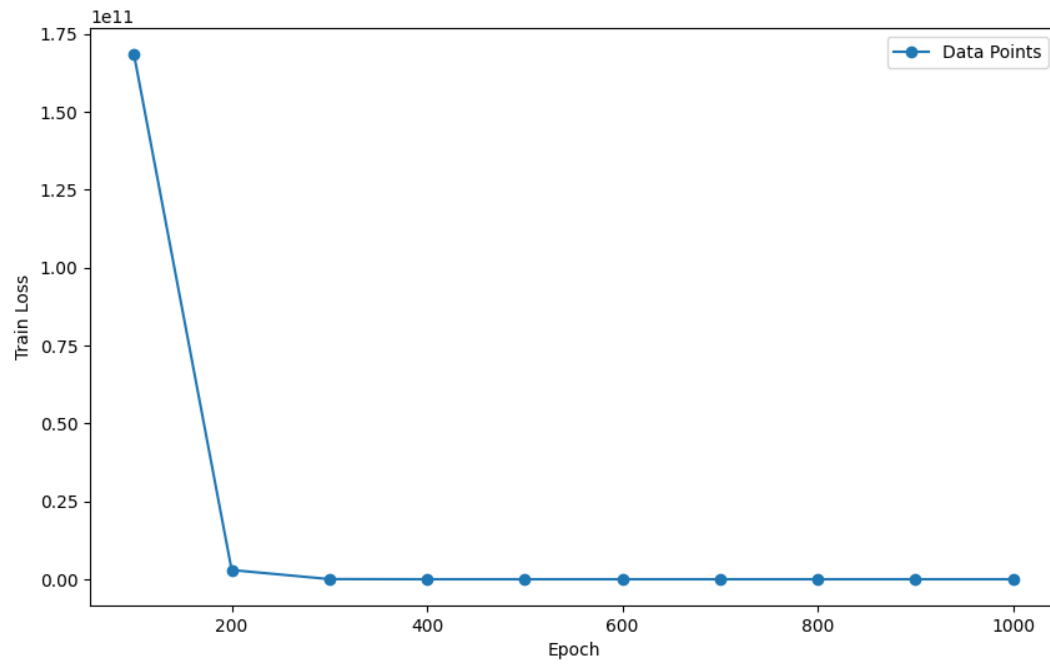
Training Loss VS. Epoch with Stochastic Gradient Descent Loss Function and Normalized Data and Learning Rate: 0.01



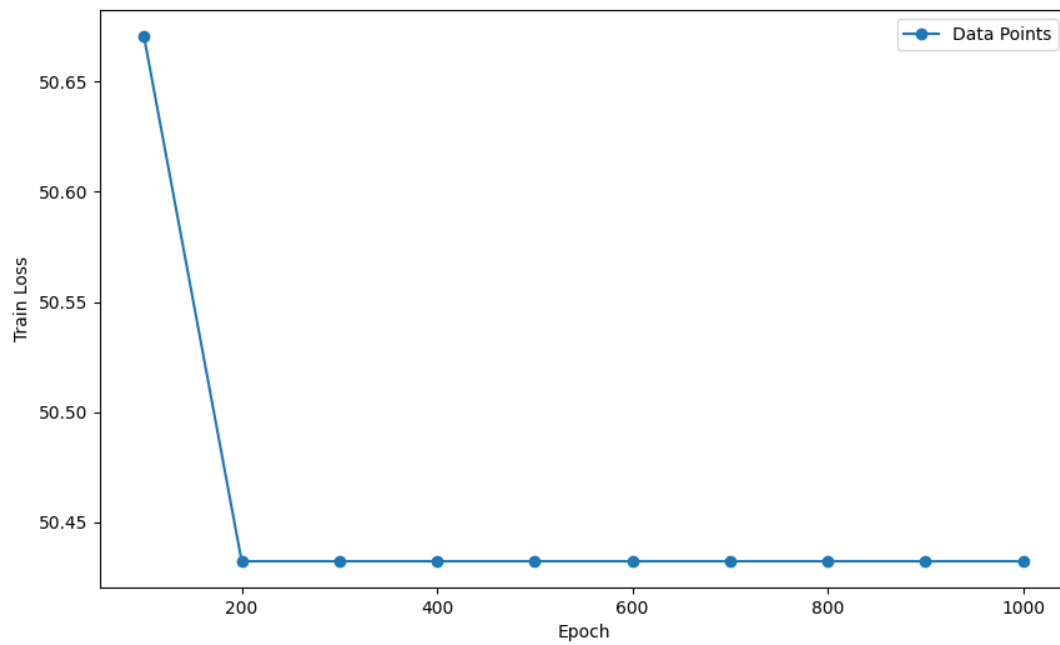
Training Loss VS. Epoch with Stochastic Gradient Descent Loss Function and Unnormalized Data and Learning Rate: 0.00



Training Loss VS. Epoch with Stochastic Gradient Descent Loss Function and Unnormalized Data and Learning Rate: 0.01



Training Loss VS. Epoch with Stochastic Gradient Descent Loss Function and Unnormalized Data and Learning Rate: 0.10



Link To NoteBook

Link:- [🔗 Assignment_1_SDM.ipynb](#)