Part I:

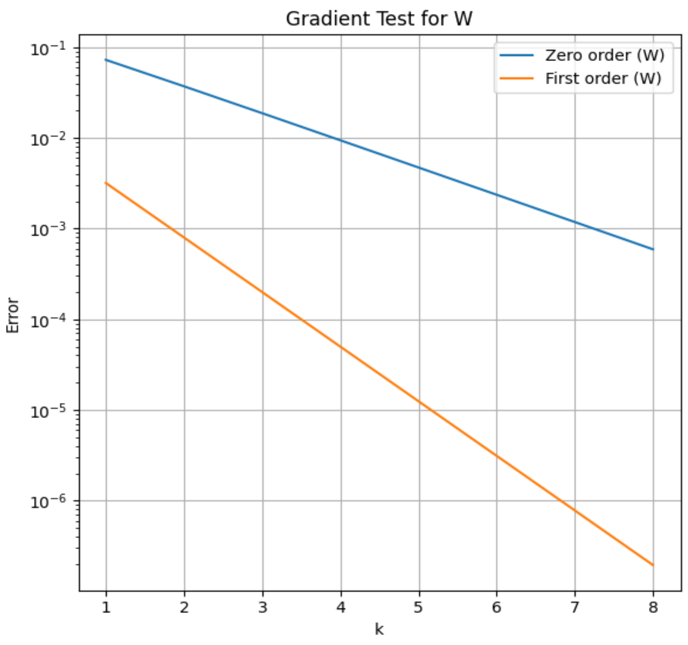
Introduction

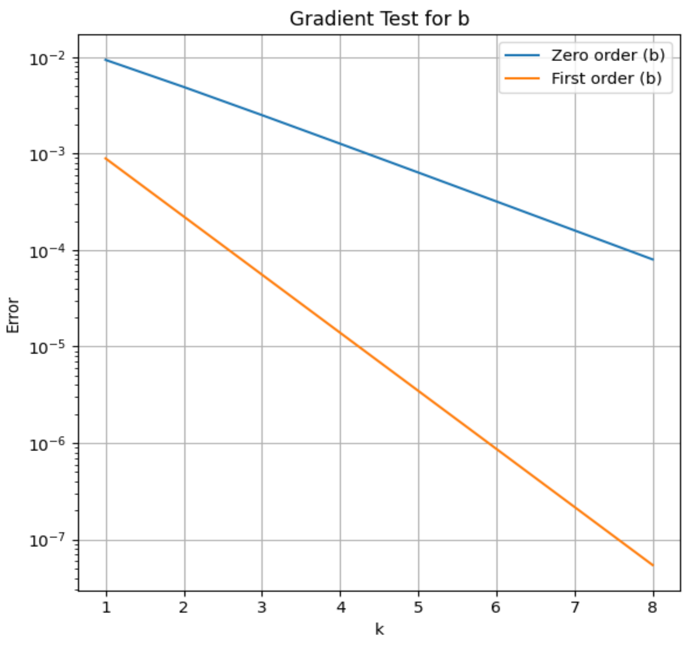
This section details the implementation of the softmax regression model, focusing on the computation of the loss function, gradient derivation, gradient validation through testing, and an evaluation of model performance. Key parameters such as the learning rate, number of epochs, and batch size are considered, with an emphasis on their impact on model convergence and stability. Graphical representations are utilized to demonstrate the behavior of gradients and the model's convergence during training.

1. Softmax Loss Function and Gradient Computation

The softmax loss function computes the cross-entropy loss, which quantifies the discrepancy between the predicted probabilities and the true labels, as well as its gradients with respect to the weights and biases.

Gradients were derived analytically to ensure efficient and mathematically accurate updates during optimization. To confirm their correctness, a gradient validation process was conducted using a gradient test.



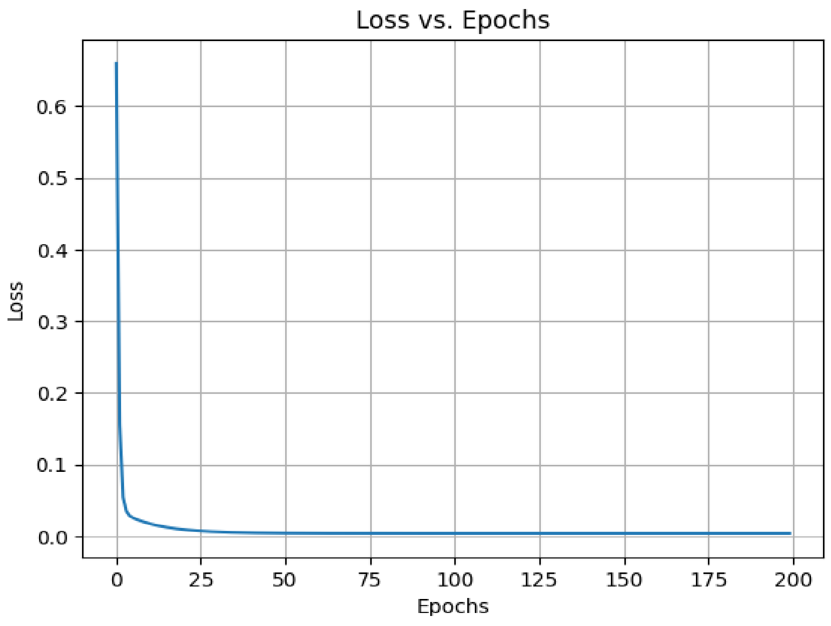


Both graphs show that as the perturbation magnitude decreases, the errors between the actual loss change and the linear approximation also decrease, confirming the accuracy of the analytical gradients.

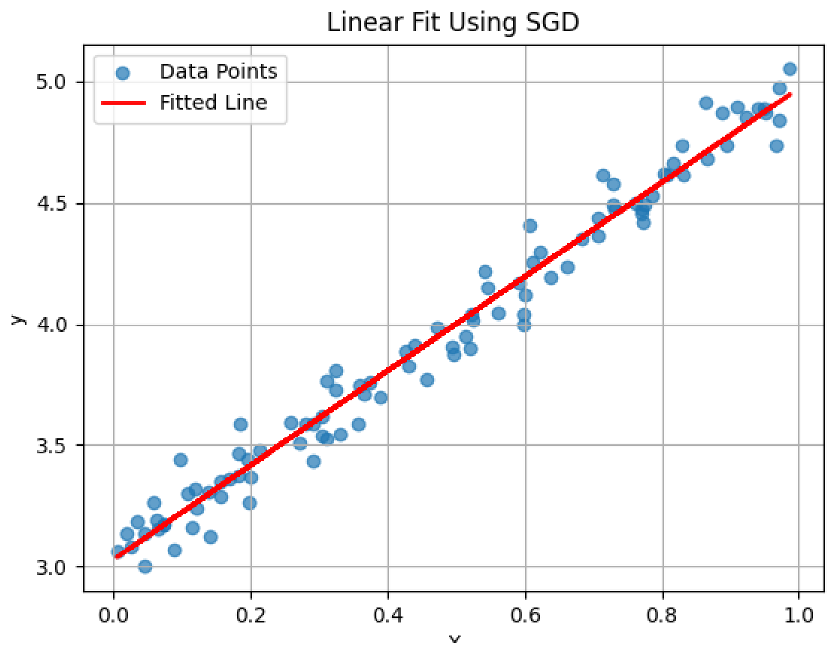
2. **Least Squares Loss and Performance Evaluation**

We evaluated the least squares loss during training using the Stochastic Gradient Descent optimizer. Our objective was to minimize the mean squared error between the model's predictions and the true target values.

To verify the effectiveness of the SGD optimizer, we created a dataset with known parameters. By experimenting with different learning rates and batch sizes, we identified hyperparameters that improved the performance. Specifically, a learning rate of 0.05 and a batch size of 8 achieved a loss of approximately 0.004 after 100 epochs. The optimizer was then applied to this dataset, and the results were evaluated using plots and performance metrics.



The loss decreases rapidly in the initial epochs, indicating effective optimization, and then stabilizes around 0.004



The fitted line closely follows the true data points, demonstrating the model's strong performance in capturing the underlying relationship

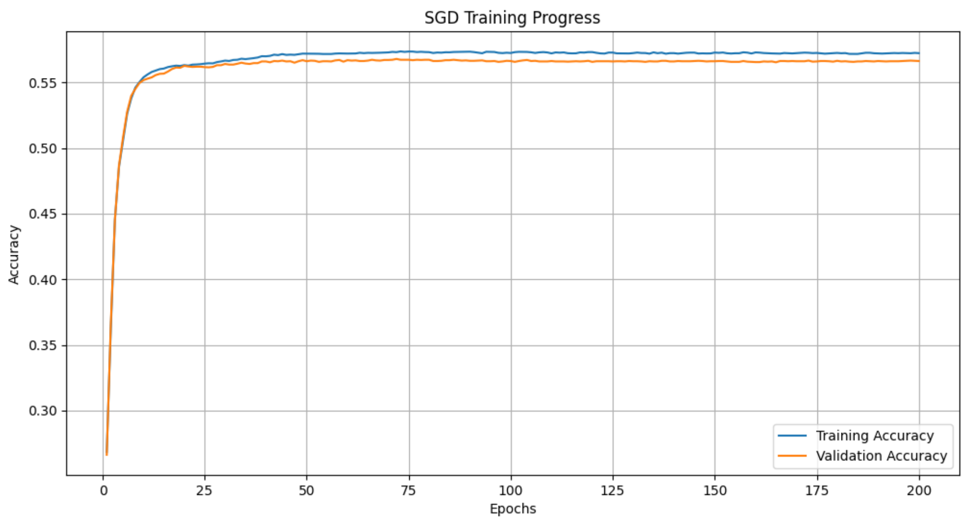
Results

- Loss Reduction:
  - The loss decreased rapidly during the initial epochs, indicating effective optimization.
  - After around 50 epochs, the loss stabilized near 0.004, demonstrating convergence.
- Model Fit:
  - The fitted regression line closely followed the true data points.
  - This alignment confirmed the model's ability to accurately capture the underlying linear relationship between the feature and target variable.

The SGD optimizer effectively minimized the least squares loss on the dataset. The rapid decrease and stabilization of the loss, coupled with the accurate fit of the regression line, validate the optimizer's performance in parameter estimation for linear regression tasks.

3. Implementation of stochastic gradient descent for optimizing a softmax: Demonstrate the minimization of the softmax function:

The following graph illustrates the minimization process of the softmax function on the Peaks dataset, achieved with a learning rate of 0.1 and a batch size of 200. The plot highlights the convergence of training and validation accuracy over 200 epochs:

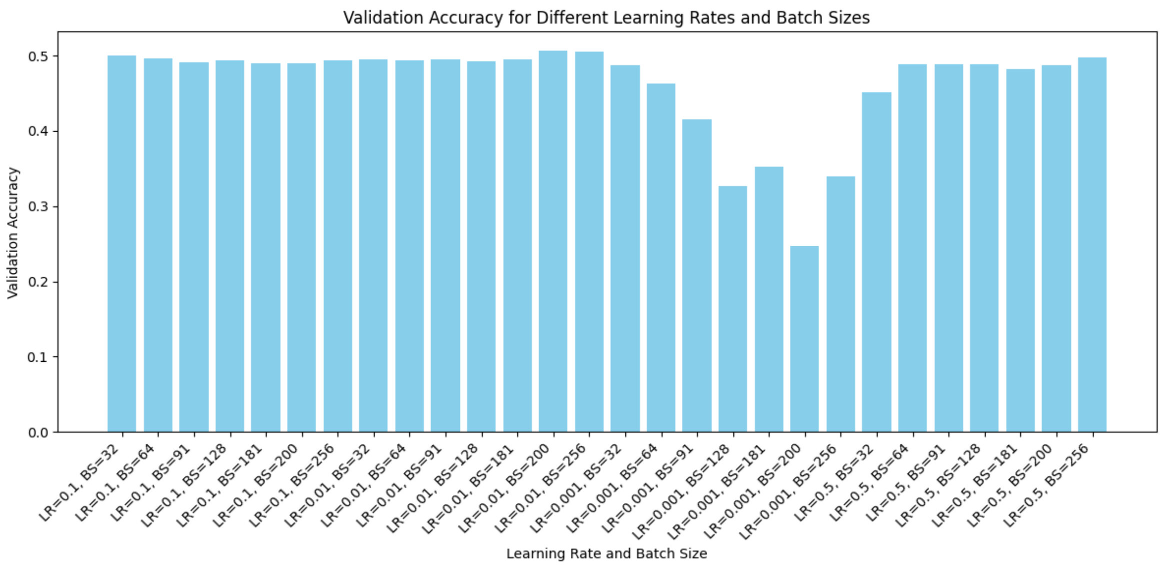


The training accuracy and validation accuracy stabilize at around 57%.

How learning rates and min-batch sizes influence on the performance:

The stochastic gradient descent algorithm was implemented to iteratively update weights and biases using small random batches of data. Random subsampling was applied to estimate accuracy after each epoch, reducing computational costs compared to evaluating the entire dataset.

- **Effect of Hyperparameters**
  To examine the impact of batch size and learning rate on model performance, the function $find\_best\_learning\_rates\_and\_batch\_sizes$ tested various configurations:
  Learning Rates**:** 0.001, 0.01, 0.1, 0.5.
  Batch Sizes**:** 32, 64, 128, 200, 256.
- **Findings**
  Results varied slightly due to randomness, but most tests showed that for a small learning rate, the performance decreased as batch size increased.
  Also, the best configuration was learning rate of 0.01 with any batch size between 32 and 256.



a running example where the optimal learning rate at the end of 50 iterations of training is 0.01 and the batch size is 200

The graphs obtained after 200 epochs demonstrate the effects of different learning rates and batch sizes, further supporting the findings above.

Observations Across Tests:

- Smaller learning rates (0.001) led to stable convergence but required more epochs.
- Larger learning rates (0.5) fast convergence but caused initial instability, performed best with more iterations.
- Smaller batch sizes (etc. 32) Improved convergence speed but required more iterations due to higher gradient noise.
- Larger batch sizes (etc. 256-300) Smoothed gradients, leading to slower convergence and lower accuracy due to reduced exploration of the loss surface.

- Therefore, at 50 iterations, learning rate 0.1 was preferred for its smooth convergence, while learning rate 0.5 was unstable. However, with more iterations, 0.5 consistently outperformed 0.1, achieving faster convergence and better final accuracy. This underscores the importance of tailoring learning rate and batch size to the training duration.

similar observations can be seen in the other data sets.

Part II:

Introduction

This report presents the results of training and evaluating neural networks on three datasets: The Gaussian Mixture Model (GMM), the SwissRoll dataset, and the Peaks dataset. We'll also include details on the Jacobian test, gradient checking, and performance metrics such as loss and accuracy over multiple training epochs. Each dataset employs a different architecture and hyperparameters (learning rate, batch size, etc.).

1. Standard Neural Network Implementation

1.1. Neural Network Architecture

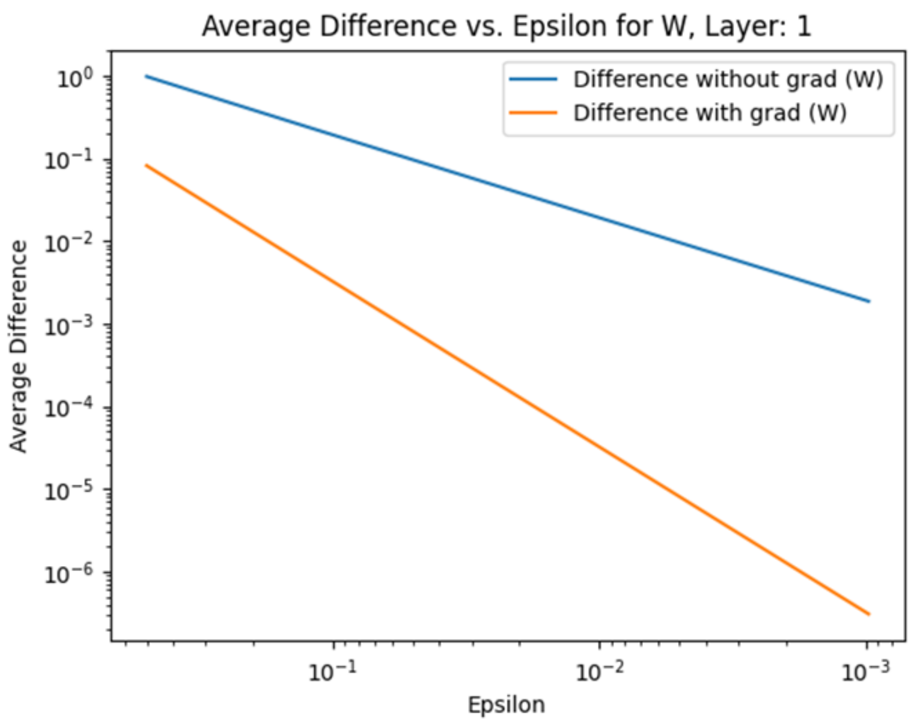
The implemented standard neural network comprises multiple layers, each containing weights and biases that are randomly initialized and subsequently learned during the training process. The architecture is defined by a list of layer sizes, where each size corresponds to the number of neurons in that layer. For example, a $layer\_sizes$ list of $[5, 14, 14, 5]$ signifies:

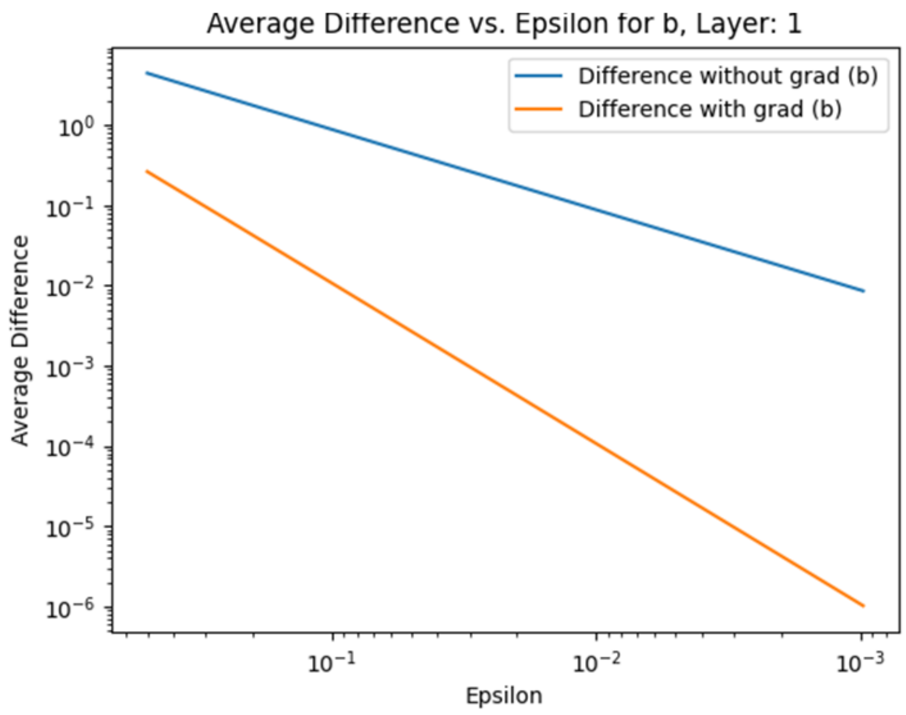
- Input Layer: 5 neurons
- Hidden Layers: Two hidden layers with 14 neurons each
- Output Layer: 5 neurons

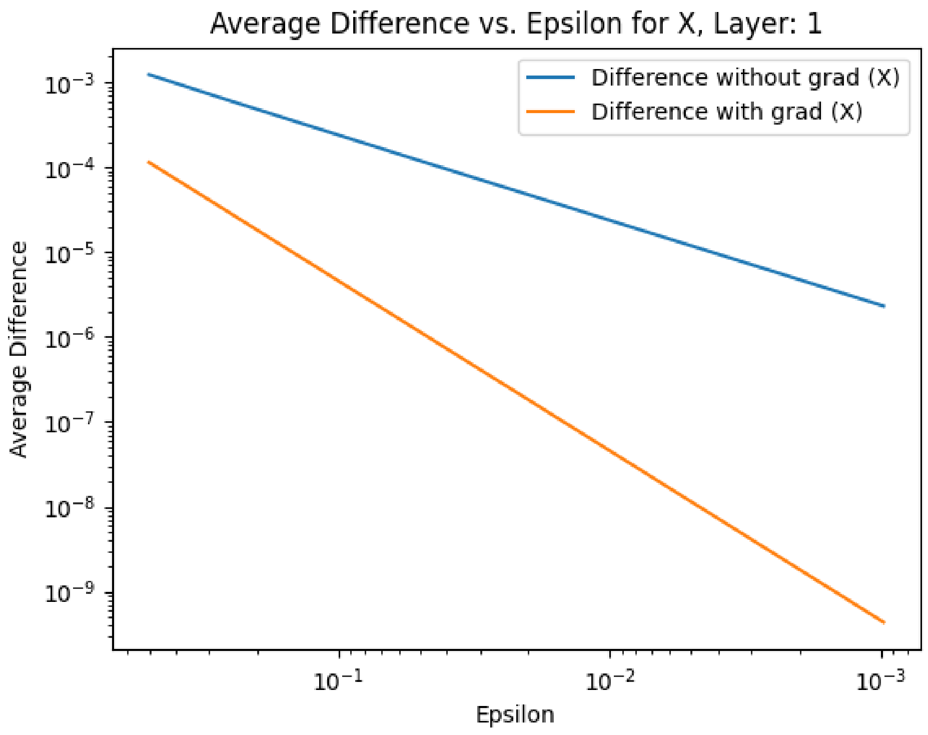
The implemented network supports both Tanh and ReLU activation functions for hidden layers and softmax function for the output layer.

1.2. Jacobian Test

The Jacobian Test is a verification of the consistency between computed gradients and numerical derivatives at each layer of the model. For each of the given datasets, we performed the Jacobian test at different layers and verified that the computed gradients match the numerical approximations.







The above plots demonstrate consistency between the computed Jacobian and its numerical approximations, showing quadratic convergence of the first order error.

2. Residual Neural Network Implementation

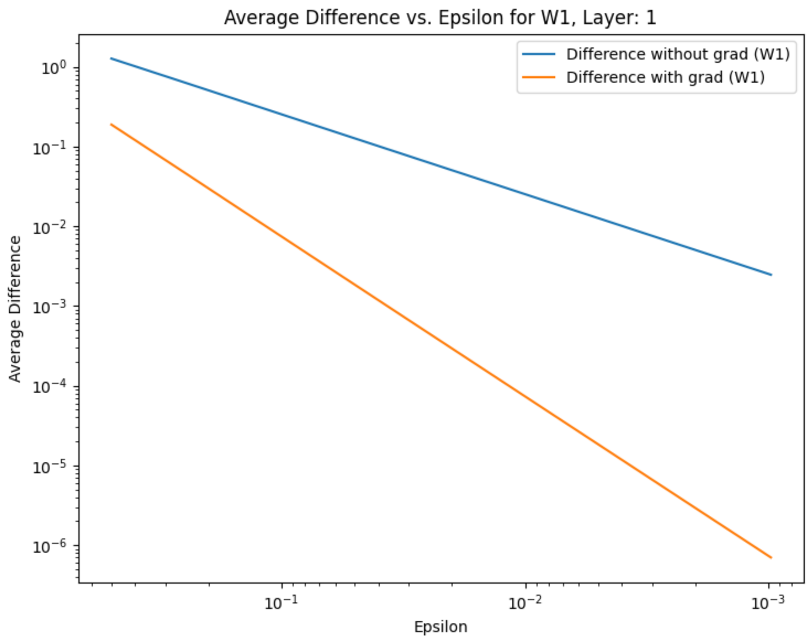
2.1. Neural Network Architecture

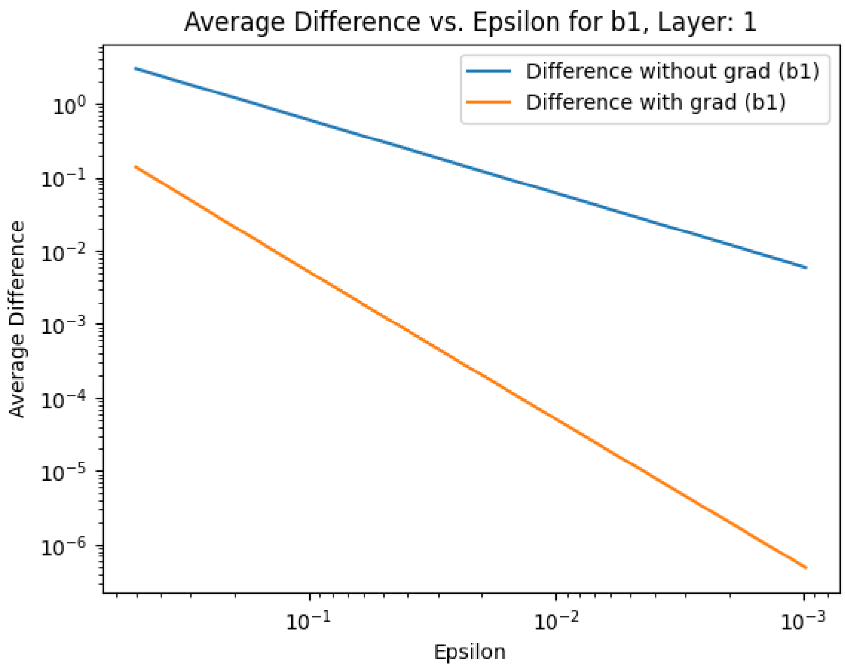
The Residual Neural Network implemented in this project extends the standard neural network architecture by incorporating residual connections. he ResNet architecture is parameterized by the number of input neurons, hidden layers, and output neurons. For example, a ResNet with $input\_size = 5$, $hidden\_sizes = [5, 5]$, and $output\_size = 5$ comprises:

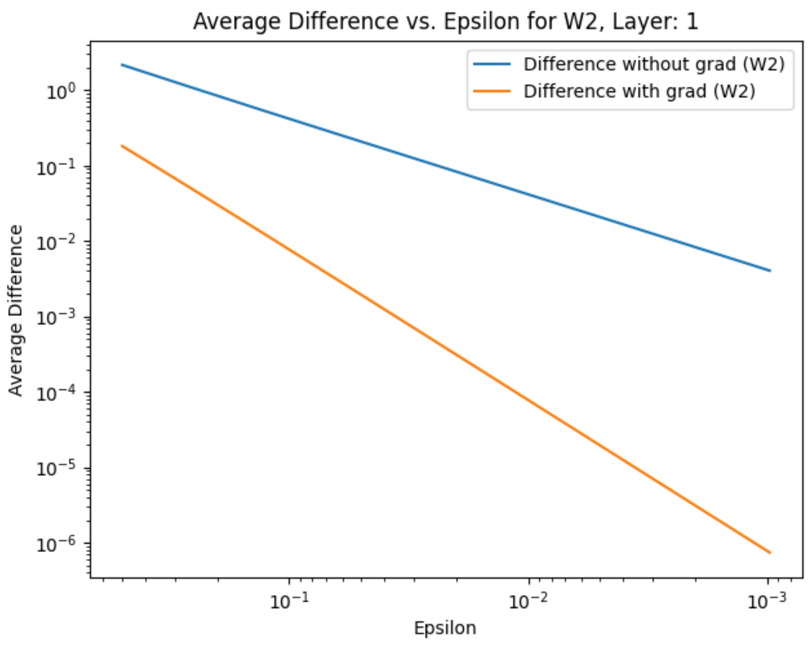
- Input Layer: 5 neurons
- Hidden Layers: Two hidden layers with 5 neurons each
- Output Layer: 5 neurons

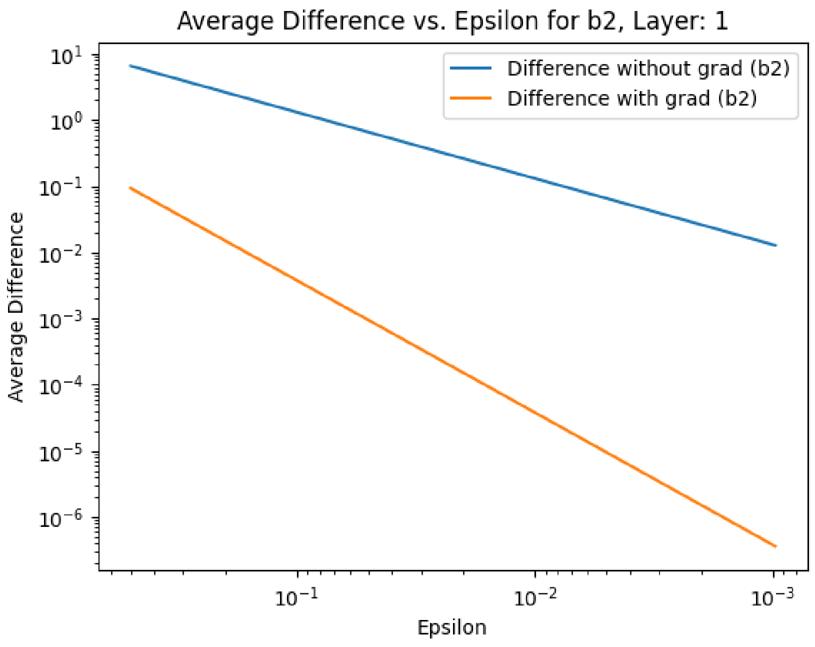
The implemented network supports both Tanh and ReLU activation functions for hidden layers and softmax function for the output layer.

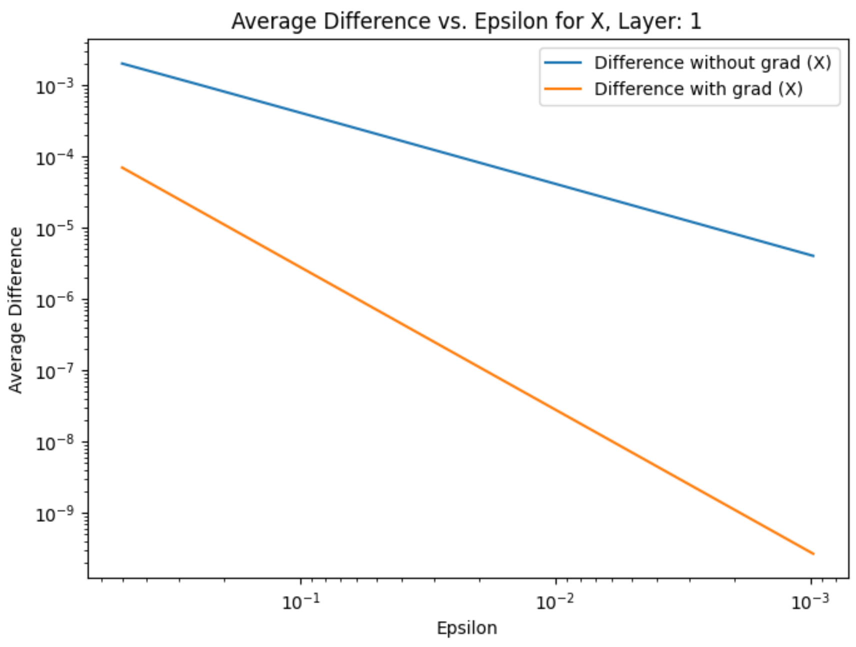
2.2. Jacobian Test






Average Difference vs. Epsilon for W2, Layer: 1
Difference without grad (W2)
Difference with grad (W2)
Average Difference
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-1}
10^{-2}
10^{-3}
Epsilon


Average Difference vs. Epsilon for b2, Layer: 1
Difference without grad (b2)
Difference with grad (b2)
Average Difference
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-1}
10^{-2}
10^{-3}
Epsilon


Average Difference vs. Epsilon for X, Layer: 1
Difference without grad (X)
Difference with grad (X)
Average Difference
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
10^{-9}
10^{-1}
10^{-2}
10^{-3}
Epsilon

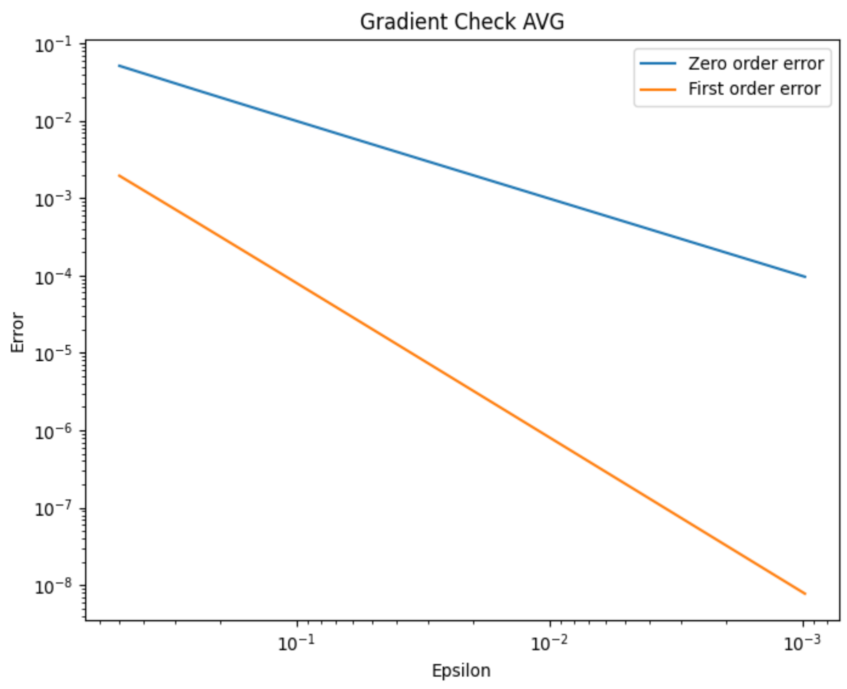
2.3. gradient test for the whole network

2.3.1. Gradient testing was performed on both the Standard Neural Network and the Residual Neural Network to verify the accuracy of gradients computed.

Error Calculation:

- Zero-Order Error- $\left|F_{plus} - F_0\right|$
  Measures the absolute difference in loss due to parameter perturbation.
- First-Order Error- $\left|F_{plus} - F_0 - \varepsilon(grad \cdot perturbation)\right|$
  Assesses the deviation from the linear approximation provided by the analytical gradient.

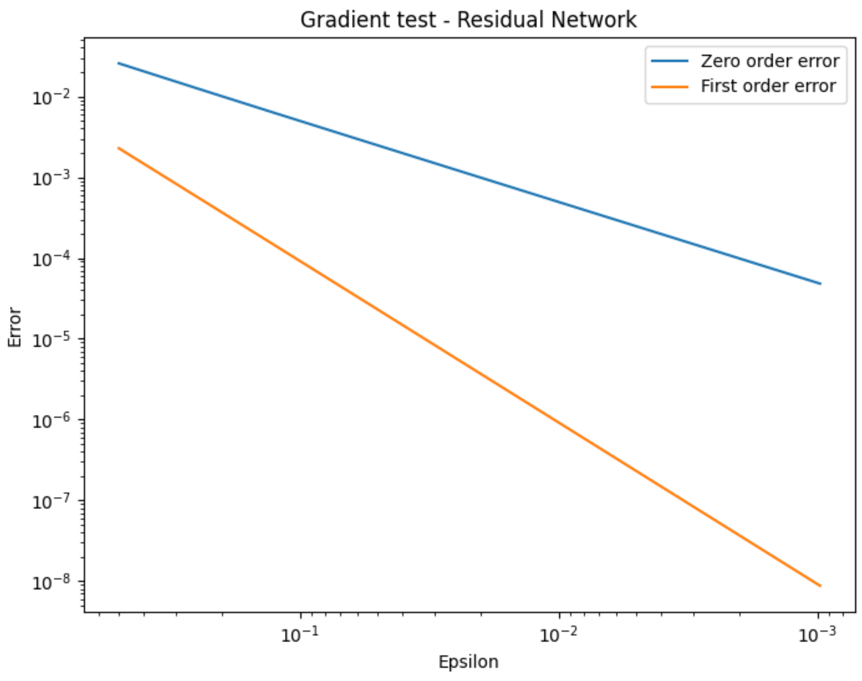
Sampling: The process was repeated for a specified number of samples (num_samples) to average the errors, leveraging the randomness to enhance the robustness and reliability of the gradient check.



2.3.2. Results and Analysis

The gradient tests exhibited strong alignment between analytical and numerical gradients for both network architectures.

- Quadratic Convergence: As ε decreased, the first-order error diminished proportionally to ε^2, indicating minimal higher-order approximation errors.
- Linear Error Behavior: The zero-order error decreased linearly with ε, consistent with theoretical expectations for first-order approximations.
- These results confirm the precision and reliability of the backpropagation implementation, validating the network's readiness for effective training and optimization.



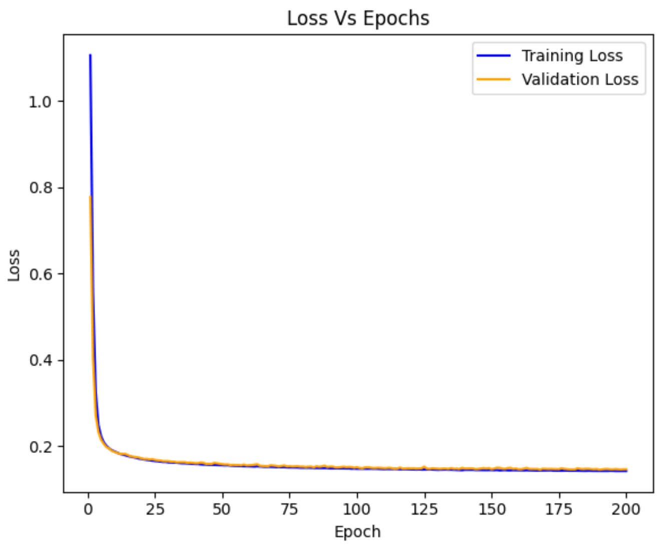
2.4. Training Results

This section presents the performance of the models for each dataset over their respective training epochs. The evolution of the loss and accuracy metrics for both training and validation sets is analyzed to evaluate the model's performance and generalization capabilities.

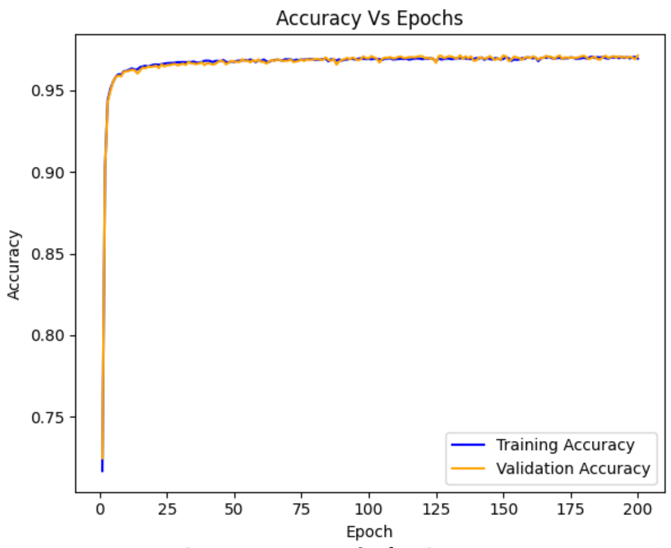
Standard Neural Networks:

The model was trained on the GMM, SwissRoll, and Peaks datasets. The activation function used for the hidden layers was tanh, and the training and validation performance metrics—loss and accuracy—were tracked at each epoch.

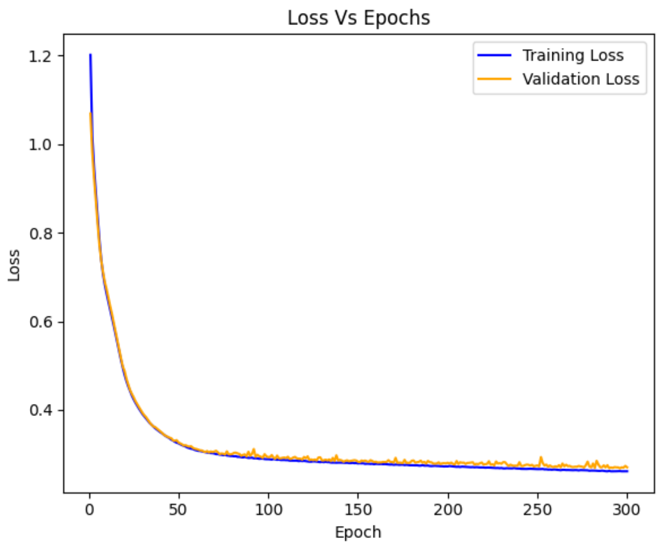
Standard	Performance Metrics				Model Architectures and Hyperparameters				
Dataset	**Final train loss**	**Final train accuracy**	**Final validation loss**	**Final validation accuracy**	**Number of Layers**	**Hidden layers**	**Learning rate**	**Batch size**	**Epochs**
GMM	0.1432	97.14%	0.1482	97.22%	Input Layer → 1 Hidden Layer → Output Layer	22 neurons each, tanh activation function	0.2	200	300
SwissRoll	0.1223	97.36%	0.1310	97.24%	Input Layer → 2 Hidden Layers → Output Layer	14 neurons each, tanh activation function	0.3		
Peaks	0.2409	93.43%	0.2501	93.09%	3 Input Layer → 2 Hidden Layers → Output Layer	14 neurons each, tanh activation function	0.05		



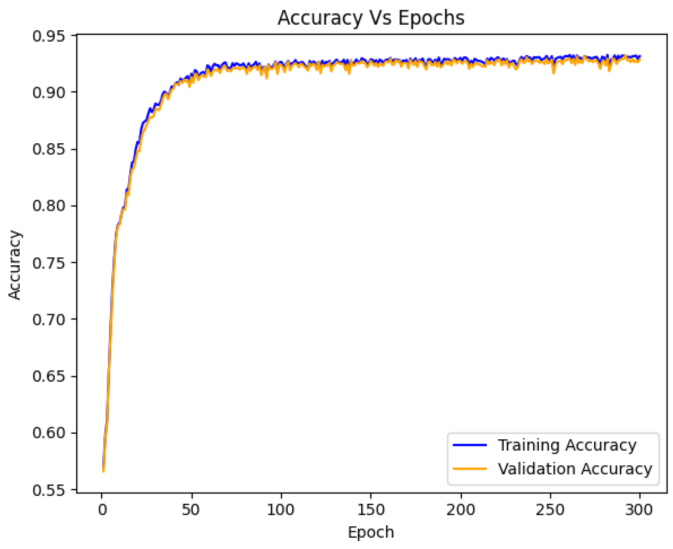
Loss vs. Epochs for GMM Dataset



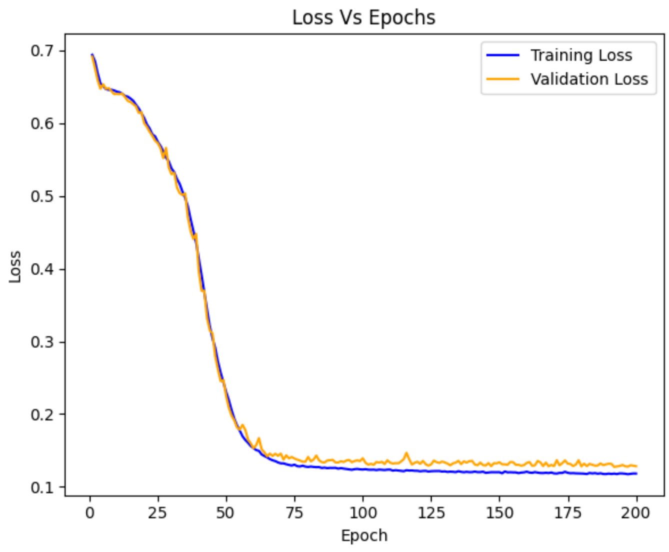
Accuracy vs. Epochs for GMM Dataset



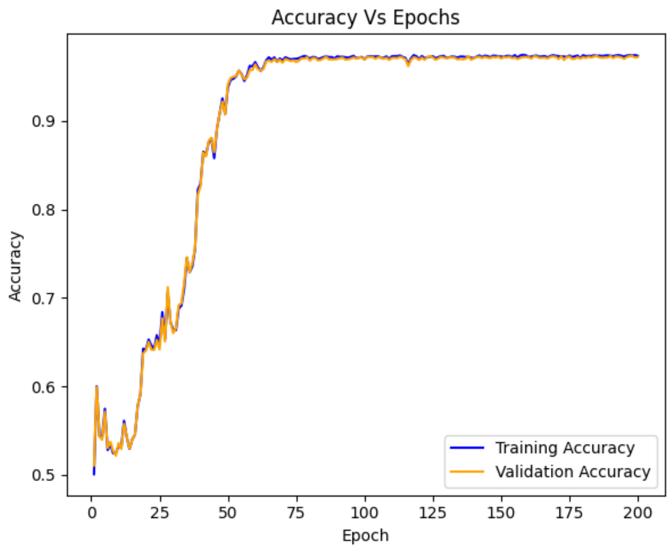
Loss vs. Epochs for Peaks Dataset



Accuracy vs. Epochs for Peaks Dataset



Loss vs. Epochs for SwissRoll Dataset



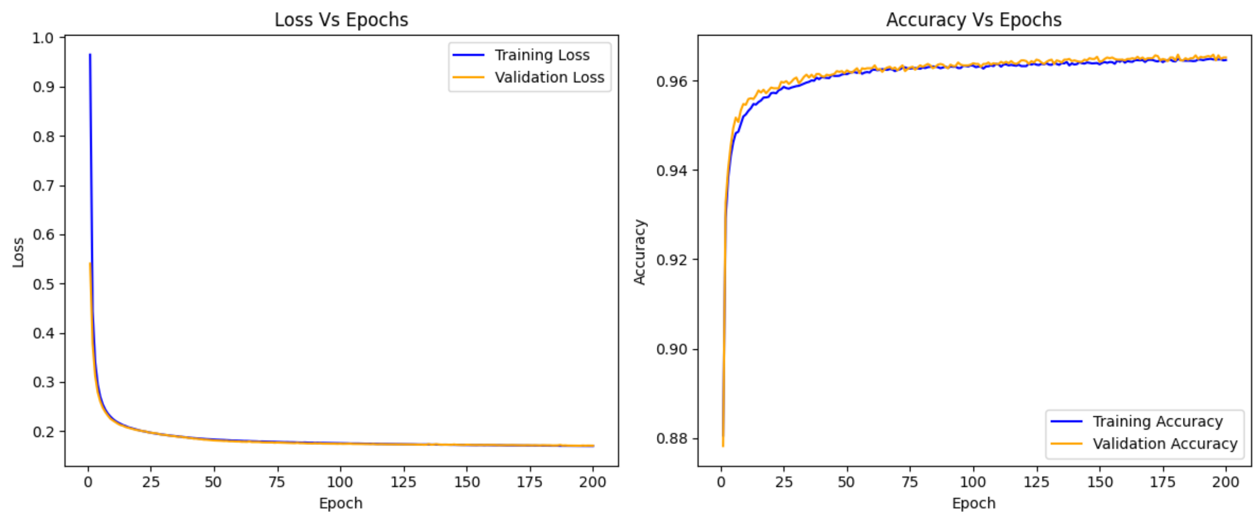
Accuracy vs. Epochs for SwissRoll Dataset

The accuracy graphs illustrating the improvement in the model's performance over time. The loss graphs show the reduction in training and validation loss during training, reflecting the optimization of the model. Observations:

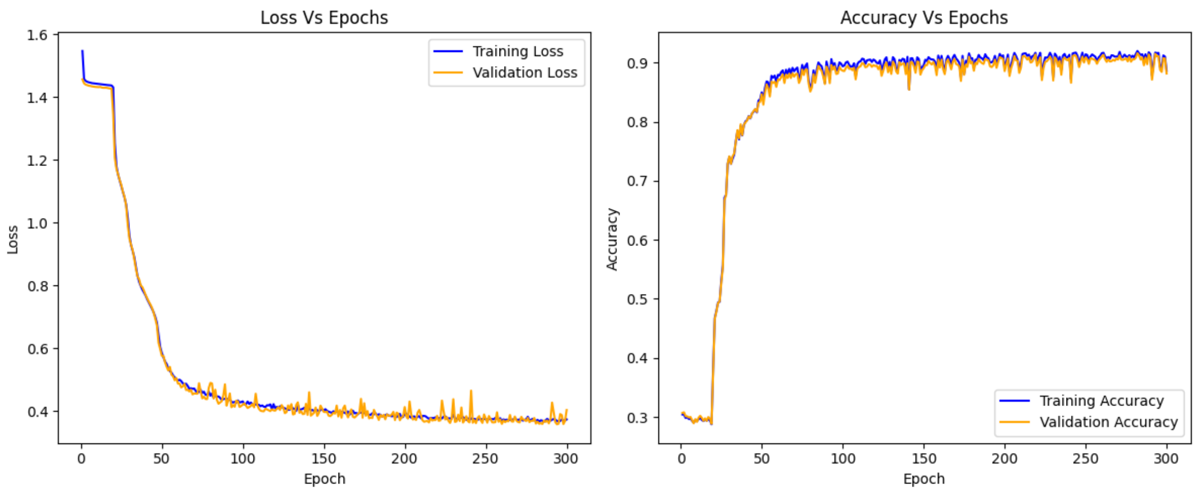
- The results show the model effectively generalizes on GMM, Swissroll, and Peaks datasets, attaining high accuracy and low loss within few epochs.
- The SwissRoll and GMM dataset outperformed the Peaks slightly.

Residual networks: The performance of the ResNet model for each dataset across their respective training epochs. The evolution of the loss and accuracy metrics for both training and validation sets is analyzed to evaluate the model's learning progress and generalization capabilities.

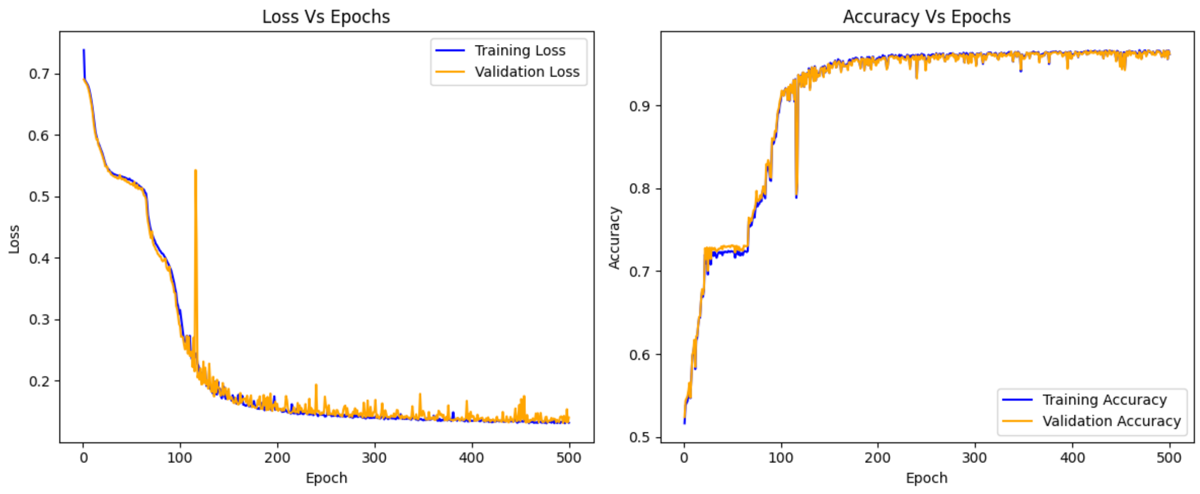
Residual	Performance Metrics				Model Architectures and Hyperparameters				
Dataset	**Final train loss**	**Final train accuracy**	**Final validation loss**	**Final validation accuracy**	**Number of Layers**	**Hidden layers**	**Learning rate**	**Batch size**	**Epochs**
GMM	0.1402	97.25%	0.1453	97.42%	Input Layer → 2 Hidden Layer → Output Layer	20 neurons each, tanh activation function	0.2		
SwissRoll	0.1302	96.57%	0.1403	96.32%	Input Layer → 2 Hidden Layers → Output Layer	20 neurons each, tanh activation function	0.03	200	300
Peaks	0.3196	91.63%	0.3221	91.65%	Input Layer → 2 Hidden Layers → Output Layer	20 neurons each, tanh activation function	0.03		



The performance of the residual netwrok descirbed above on the GMM dataset



The performance of the residual netwrok described above on the Peaks dataset



The performance of the residual netwrok described above on the SwissRoll dataset

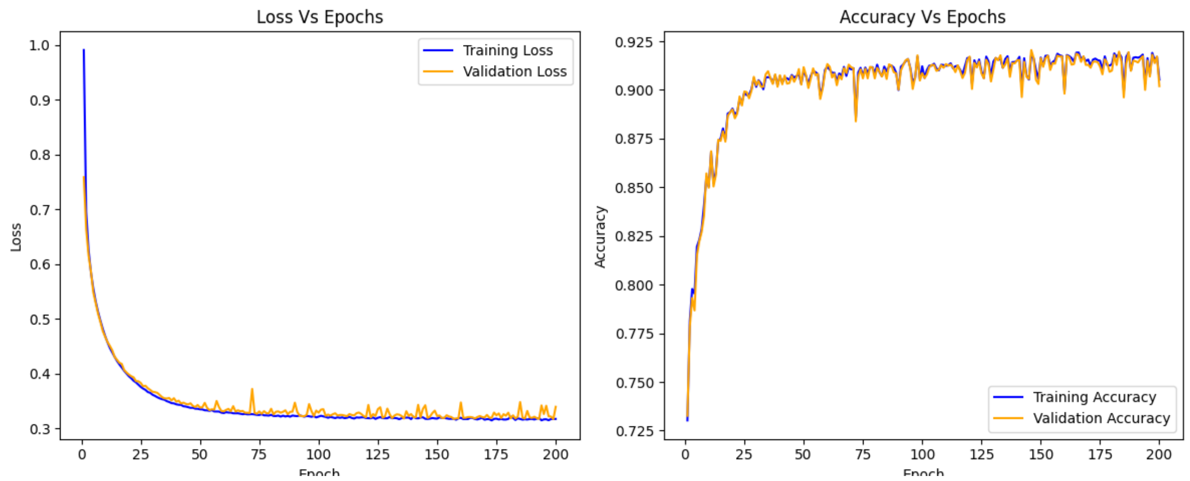
Observations:

- The ResNet model achieves high training and validation accuracy with low loss values at the final epoch in relatively few epochs. Similarly, the SwissRoll and GMM datasets slightly outperformed the Peaks dataset.
- The GMM dataset exhibited high stability, allowing for the use of a high learning rate and achieving strong performance in fewer iterations.
- The complexity of the residual network prompts us to adjust the number of layers and neuron sizes to achieve better results and align with the dataset requirements.

2.4 The effect of network depth

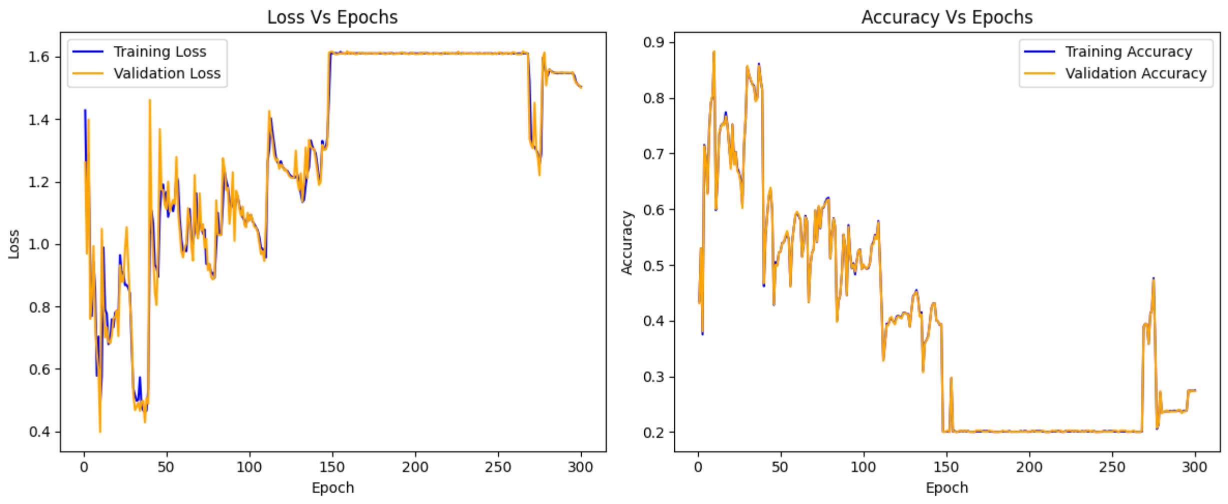
To examine the effect of network depth on performance, we trained models with varying numbers of hidden layers. For very deep networks, we observed significant fluctuations in both training and validation loss, with convergence taking much longer to stabilize, as illustrated in the graph below. This aligns with the known challenges of training deeper networks with multiple hidden layers, which often require more epochs due to their complexity and capacity to model intricate patterns. However, in our case, the datasets did not demand such a highly complex network structure.

Our experiments showed that a network with two hidden layers consistently struck the optimal balance between training and validation accuracy across all datasets. While networks with only one hidden layer converged faster, their final accuracy was slightly lower, reflecting their limited capacity to capture the complexity of the datasets.



A network with 1 hidden layer of size 14 with the tanh activation function and Peaks, performed 90% accuracy , while the two hidden layers performed 93% accuracy.

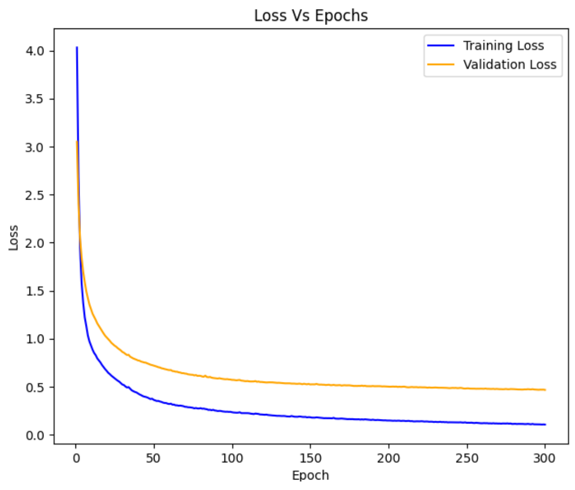
On the other hand, networks with three or four hidden layers performed good as well but required more processing time to achieve similar results. Deeper architectures exceeding five layers, however, introduced significant instability and led to a decline in performance. These findings confirm that a two-hidden-layer architecture is both efficient and effective for modeling the given datasets, striking a balance between computational cost and predictive accuracy.

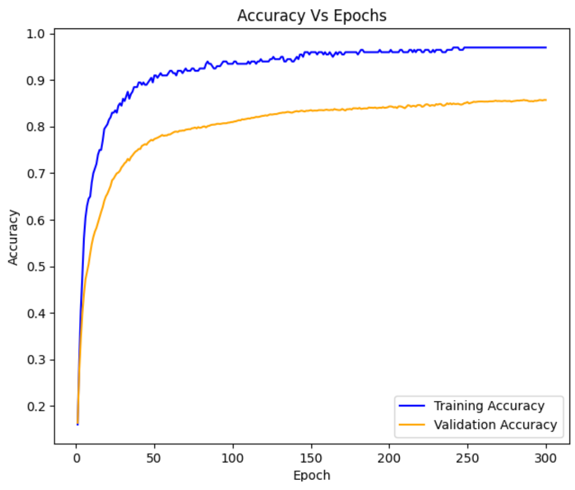


A too complex network with 10 hidden layers of size 20 with the tanh activation function and GMM

2.5. Training with 200 Data Points:

Using 200 Randomly Sampled Data Points from the GMM Dataset:

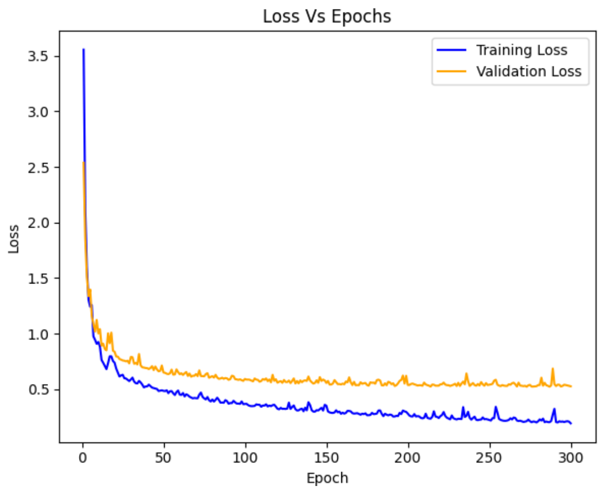


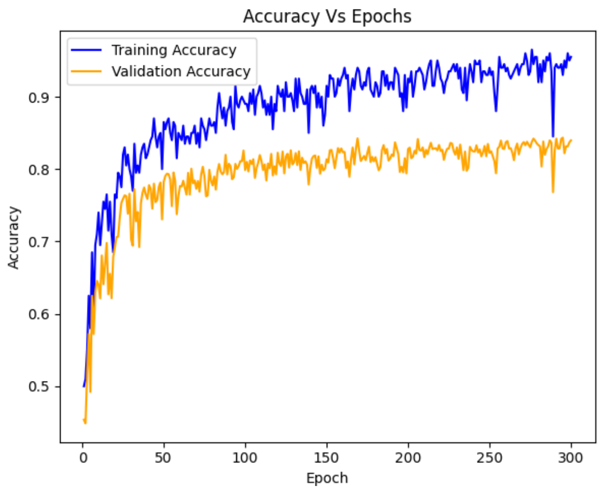


Train loss around 0.1, Train Accuracy around 95%

Validation Loss around 0.46, Validation accuracy around 83%

Using 200 Randomly Sampled Data Points from the Peaks Dataset:

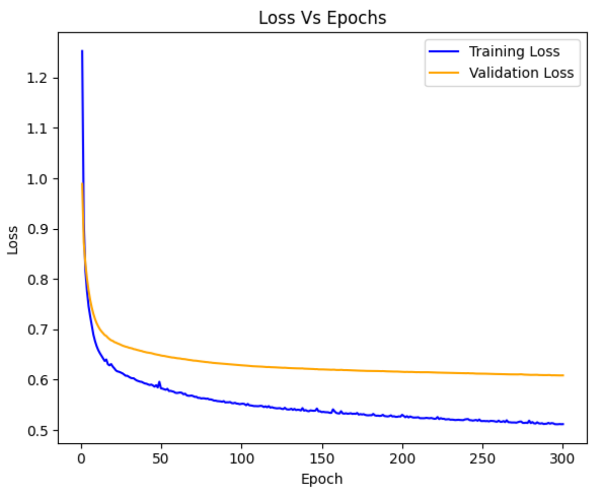


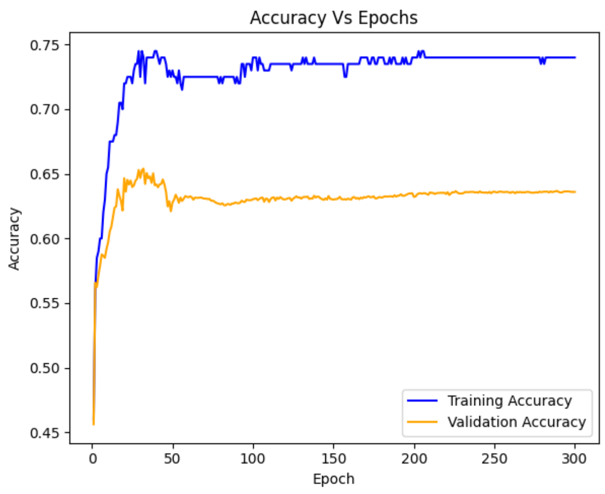


Train loss around 0.19, Train Accuracy around 97%

Validation Loss around 0.52, Validation accuracy around 83%

Using 200 Randomly Sampled Data Points from the SwissRoll Dataset:





Train loss around 0.51, Train Accuracy around 74%

Validation Loss around 0.6, Validation accuracy around 63%

Observations:

- The training loss decreases steadily, and the training accuracy approaches 100% by the end of training.
- The validation loss converges but remains consistently higher than the training loss. Validation accuracy is lower than training accuracy and plateaus before reaching optimal performance.
- The larger gap between training and validation performance indicates overfitting, as the model struggles to generalize to unseen data.

In conclusions, reducing the training dataset to 200 data points results in faster training times but causes lower validation accuracy and increased overfitting due to the limited representation of the data distribution.