SPHERICAL PROJECTION OF GAUSSIAN DATA: A NEURAL NETWORK APPROACH

Swati Swati

Victoria University of Wellington, NZ

1. INTRODUCTION

In the realms of machine learning and data visualization, projecting high-dimensional data onto lower-dimensional geometric surfaces has garnered significant attention. One such problem involves mapping data from a three-dimensional space onto a unit sphere, which serves as a powerful tool for visualizing and understanding complex data structures.

With the advent of neural networks and deep learning, there is an opportunity to leverage these advanced models to learn complex mappings from data to geometric surfaces, including the unit sphere.

Neural networks, particularly those involving multiple layers and nonlinear activation functions, offer a robust framework for learning intricate mappings from high-dimensional spaces to lower-dimensional representations. By training a neural network to perform this projection, we can exploit its ability to capture non-linear relationships and intricate patterns in the data that traditional methods might miss.

2. THEORY

In this section, we focus on the theoretical background relevant to the implemented system.

2.1. Neural Networks

Neural networks are a class of machine learning models that consist of interconnected layers of nodes, or neurons, where each connection represents a weight that can be adjusted during training [1]. It typically includes:

Fully Connected Layers that connects every neuron in one layer to every neuron in the next layer, allowing the model to learn complex patterns.

Activation Functions that introduces non-linearity into the network, enabling it to learn complex mappings. Common activation functions include ReLU (Rectified Linear Unit), sigmoid, etc.

2.2. 3D Gaussian Distribution

A 3D Gaussian distribution, also known as a multivariate normal distribution, is a generalization of the normal distribution to three dimensions. It is characterized by a mean vector and a covariance matrix. We utilize the normal distribution (Gaussian distribution) to generate 3D data points with zero mean and unit variance [4]. If we have a covariance matrix $\mathbf{\Sigma}$, then this distribution is given by the pdf function:

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{n/2}|\mathbf{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)\right) \tag{1}$$

2.3. Loss Function

The loss function quantifies the difference between the predicted outputs of the model and the actual target values. The goal during training is to minimize this loss [2].

2.4. Hyperparameter Tuning

Hyperparameter tuning involves selecting the best set of hyperparameters for a model, which are set before training begins. These include the number of epochs, batch size, and learning rate. Proper tuning is crucial for optimizing model performance and achieving the best results.

2.5. Gradient Descent

Gradient descent is an optimization algorithm used to find a model's parameters that minimize the loss function in machine learning models. It works by iteratively adjusting the model's parameters (weights) in the direction of the negative gradient of the loss function with respect to those parameters [2]. For a loss function $\mathbf{L}(\theta)$ where θ represents the parameters of the model, the gradient descent update rule is:

$$\theta_{\text{new}} = \theta_{\text{old}} - \alpha\nabla L(\theta) \tag{2}$$

3. EXPERIMENTS

In this section, we evaluate the performance of a neural network model designed to project 3D Gaussian-distributed data onto the surface of a unit-radius sphere. The experiment involves generating synthetic data from a 3D normal distribution and training a neural network to map this data onto the sphere.

3.1. Experimental Setup

The experiments were conducted using Google Colab, with PyTorch as the deep learning framework and Plotly for visualizing the results.

In the data generation part, we generated *10,000* samples from a 3D normal distribution and normalized them to lie on the unit sphere. The dataset was split into training (**60%**), validation (**20%**), and test (**20%**) sets. Finally, the training data output was visualized on the sphere as shown in Fig. 1.

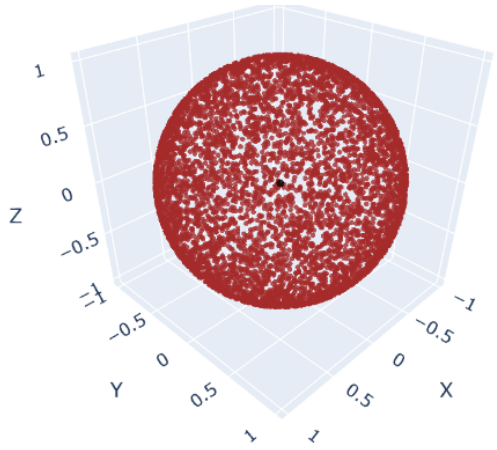

Fig. 1. Training Data Output on Unit Sphere

In the model architecture part, the model used is a feed-forward neural network with four fully-connected layers, an *Input layer with 3 neurons* for 3D input data, *two hidden layers with 20 neurons each*, and an *Output layer with 3 neurons*. The *ReLU* activation function is applied to the input and hidden layers.

3.2. Training Process

The training process involves tuning various hyperparameters. A series of hyperparameters were tested along with the different values and combinations as mentioned in Fig. 2.

Hyperparameters	Tested Values	Best Set
Epochs	25, 50, 100, 200	200
Batch Size	16, 25, 32, 64	64
Learning Rate	0.1, 0.01, 0.001	0.001

Fig. 2. Hyperparameters along with the tested values and best set.

The hyperparameter tuning is performed using *grid search* [5] to identify the combination that yields the best validation performance, and the results are mentioned in the last column of Fig. 2.

The model is then trained using the *Adam optimizer* with *Mean Squared Error* (MSE) as the loss function, as it resulted in lower validation loss compared to Stochastic Gradient Descent (SGD) [3]. Fig. 4 (in appendix) illustrates the Validation Loss vs. epochs for both optimizers.

Performance is assessed by tracking both training and validation losses over epochs using the best-found hyperparameter set. Fig. 3 clearly demonstrates the rapid convergence of both training and validation losses, stabilizing at a low value within the first 20 epochs (early stopping at 106^{th} epoch).

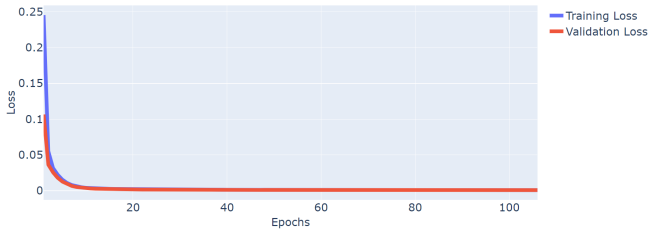

Fig. 3. Training and Validation Loss over Epochs.

Finally, the model's performance in projecting data onto the unit sphere was evaluated using the optimal hyperparameters on the unseen test data.

3.3. Results

The best-performing model, selected based on the lowest validation loss, was evaluated on the test set and showed a high degree of accuracy in projecting the 3D data onto the unit sphere. The test loss was approximately 0.000871, indicating excellent performance in minimizing the mean squared error between the predicted and actual sphere coordinates. Fig. 5 displays the predicted test data output on the unit sphere.

4. CONCLUSION

The model successfully projected 3D Gaussian-distributed unseen data onto a unit sphere with minimal error or loss. Hyperparameter tuning was crucial in achieving optimal performance. The chosen model, with its finely tuned hyperparameters and the use of the Adam optimizer, demonstrated effective mapping capabilities, as evidenced by the significantly reduced validation loss.

5. STATEMENT OF TOOLS USED

The tools used in this study include PyTorch for neural network implementation, Plotly for graph visualization, and Google Colab for executing the code. The code was written by me, and I verify that it is my original work. The code can be accessed at the following link: *Final-Code* with the tuned optimal hyperparameters. For the code with grid search to tune the hyperparameters refer *Hyperparameter-Tuning*.

6. REFERENCES

[1] Prof Bastiaan Kleijn. Feedforward neural network. Victoria University Lecture in DLIntro.pdf, 2024.

[2] Prof Bastiaan Kleijn. Gradient descent and loss function. Victoria University Lecture in DLIntro.pdf, 2024.

[3] Prof Bastiaan Kleijn. Learning algorithms: Adam. Victoria University Lecture in SGD.pdf, 2024.

[4] Prof Bastiaan Kleijn. Multivariate gaussian distribution. Victoria University Lecture in probability.pdf, 2024.

[5] User Guide Scikit-learn. Tuning the hyper-parameters of an estimator. https://scikit-learn.org/stable/modules/grid_search.html.

7. APPENDIX

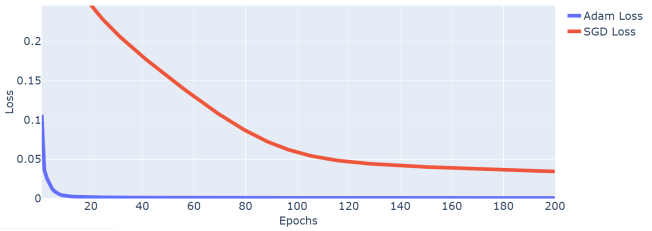

Fig. 4. Plot for SGD vs Adam Validation Loss over number of epochs.

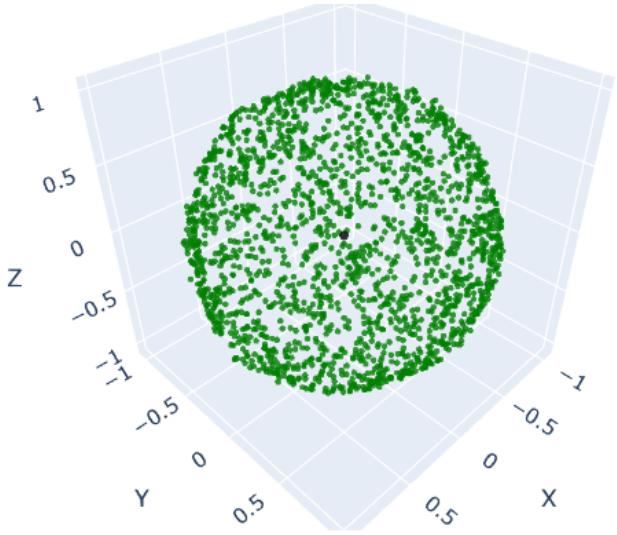

Fig. 5. Predicted output of testing dataset.