Exploring 3D Sinusoidal Data | Artificial Neural Networks

Applied Machine Learning

Name: Kevin Veeder

Overview

The purpose of this project was to experiment with artificial neural networks for solving a basic regression problem using noisy 3D sinusoidal data. The goal was to explore how different hyperparameters affect a model's predictions, and to ultimately build a neural network that captures the underlying trend in the data smoothly and accurately.

I used a lightweight Sequential model that trains quickly on the provided dataset, which allowed me to manually adjust hyperparameters and observe their influence on performance. This was an exploratory, hands-on process—adjusting values, training the model, reviewing the prediction curve, and repeating until I identified a reasonable configuration. In short: a good old-fashioned manual hyperparameter sweep.

The objective was to build and fine-tune a model that could predict a curve that closely fits the underlying signal, with minimal noise and erratic behavior. In the end, the model should produce a smooth and coherent prediction line when plotted alongside the training data.

Note on Visualization

Smoother prediction curves can result not just from improved models, but also from thoughtful data sorting before plotting. For example, sorting by input value to minimize the arc length of the prediction curve can improve visualization. While that optimization wasn't a primary focus here, I made sure to generate a clean figure that shows the model's prediction line superimposed on the original data.

Here are a few of the hyperparameters I experimented with during the process:

- number of nodes per layer
- number of layers
- activation functions
- normalization method (should be negligible)
- number of epochs
- learning rate
- loss function

You'll know the model is performing well when the prediction curve aligns closely with the trend of the data. Initially, the line plot of the model's predictions may look erratic. If that happens, switching to a scatter plot of the predictions can help diagnose the issue—usually related to the input data not being sorted. Once resolved, I finalized the project with a connected line plot that clearly visualizes the learned function.

Evaluation

To evaluate the model's performance, I also calculated the generalization error on a held-out test set.

Preliminaries

Let's import some common packages:

In [1]:
```
# Common imports
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib import cm
import numpy as np
import pandas as pd
%matplotlib inline
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
import os

# Where to save the figures
PROJECT_ROOT_DIR = "."
FOLDER = "figures"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, FOLDER)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)

def plot3Ddata(data_df):
    from mpl_toolkits.mplot3d import Axes3D
    # split dataframe into X, Y, Z
    x = data_df['x']
    y = data_df['y']
    z = data_df['z']
    fig = plt.figure(figsize=[18, 10])
    font = {'family': 'serif',
        'color':  'darkred',
        'weight': 'normal',
        'size': 16,
        }
    # first subplot
    ax = fig.add_subplot(2, 2, 1, projection='3d')
```

```
    ax.view_init(0, 90)
    ax.scatter3D(x, y, z, c=z, cmap='Blues');
    ax.set_xlabel('x',fontdict=font)
    ax.set_ylabel('y',fontdict=font)
    ax.set_zlabel('z',fontdict=font)
    # second subplot
    ax = fig.add_subplot(2, 2, 2, projection='3d')
    ax.view_init(90, 0)
    ax.scatter3D(x, y, z, c=z, cmap='Blues');
    ax.set_xlabel('x',fontdict=font)
    ax.set_ylabel('y',fontdict=font)
    ax.set_zlabel('z',fontdict=font)
    # third subplot
    ax = fig.add_subplot(2, 2, 3, projection='3d')
    ax.view_init(45, 45)
    ax.scatter3D(x, y, z, c=z, cmap='Blues');
    ax.set_xlabel('x',fontdict=font)
    ax.set_ylabel('y',fontdict=font)
    ax.set_zlabel('z',fontdict=font)
    # 4th subplot
    ax = fig.add_subplot(2, 2, 4, projection='3d')
    ax.view_init(25, 75)
    ax.scatter3D(x, y, z, c=z, cmap='Blues');
    ax.set_xlabel('x',fontdict=font)
    ax.set_ylabel('y',fontdict=font)
    ax.set_zlabel('z',fontdict=font)

    plt.tight_layout(pad=0.0, w_pad=0.0, h_pad=0.0)
    plt.show()


def plotscatter3Ddata(fit_x, fit_y, fit_z, scat_x, scat_y, scat_z):
    fig = plt.figure(figsize=[18, 10])
    font = {'family': 'serif',
        'color':  'darkred',
        'weight': 'normal',
        'size': 16,
        }
    # first subplot
    ax = fig.add_subplot(2, 2, 1, projection='3d')
    ax.view_init(0, 90)
    ax.scatter3D(scat_x, scat_y, scat_z, c=scat_z, cmap='Blues');
    ax.plot3D(fit_x, fit_y, fit_z, '-k');
    ax.set_xlabel('x',fontdict=font)
    ax.set_ylabel('y',fontdict=font)
    ax.set_zlabel('z',fontdict=font)
    # second subplot
    ax = fig.add_subplot(2, 2, 2, projection='3d')
    ax.view_init(90, 0)
    ax.scatter3D(scat_x, scat_y, scat_z, c=scat_z, cmap='Blues');
    ax.plot3D(fit_x, fit_y, fit_z, '-k');
    ax.set_xlabel('x',fontdict=font)
    ax.set_ylabel('y',fontdict=font)
    ax.set_zlabel('z',fontdict=font)
    # third subplot
    ax = fig.add_subplot(2, 2, 3, projection='3d')
    ax.view_init(45, 45)
    ax.scatter3D(scat_x, scat_y, scat_z, c=scat_z, cmap='Blues');
    ax.plot3D(fit_x, fit_y, fit_z, '-k');
    ax.set_xlabel('x',fontdict=font)
```

```
    ax.set_ylabel('y',fontdict=font)
    ax.set_zlabel('z',fontdict=font)
    # 4th subplot
    ax = fig.add_subplot(2, 2, 4, projection='3d')
    ax.view_init(25, 75)
    ax.scatter3D(scat_x, scat_y, scat_z, c=scat_z, cmap='Blues');
    ax.plot3D(fit_x, fit_y, fit_z, '-k');
    ax.set_xlabel('x',fontdict=font)
    ax.set_ylabel('y',fontdict=font)
    ax.set_zlabel('z',fontdict=font)
    plt.tight_layout(pad=0.0, w_pad=0.0, h_pad=0.0)
    plt.show()
```

Import, Split and Standardize Data

To get started, I'll begin by importing the dataset from a file named `3DSinusoidalANN.csv` and loading it into a DataFrame called `data`.

From there, I'll split the dataset into training and test sets using `train_test_split()`, with 20% of the data reserved for testing. Based on the structure of the dataset, it makes the most sense to treat the x and z columns as input features, and the y column as the target variable. I'll assign the resulting splits to X_train, X_test, y_train, and y_test, and set random_state=42 to ensure the results are reproducible. I'll continue using this random state value consistently throughout the notebook.

After the split, I'll standardize the feature data using `StandardScaler()` to ensure the model trains more effectively. This will help normalize the input values, which is especially important when working with neural networks.

In [2]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

data = pd.read_csv("3DSinusoidalANN.csv")
# data = data.sort_values(by=['z'])

x = data['x']
y = data['y']
z = data['z']

X = pd.DataFrame({
    'x': x,
    'z': z
})

X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, test_size=0.20, ra
X_train, X_valid, y_train, y_valid = train_test_split(X_train_full, y_train_full, test

scaler = StandardScaler().fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

```
X_valid = scaler.transform(X_valid)

scaler_y = MinMaxScaler(feature_range = (-1.5, 1.5)).fit(pd.DataFrame(y_train))

y_train = scaler_y.transform(pd.DataFrame(y_train))
y_test = scaler_y.transform(pd.DataFrame(y_test))
y_valid = scaler_y.transform(pd.DataFrame(y_valid))
```

Plot Data

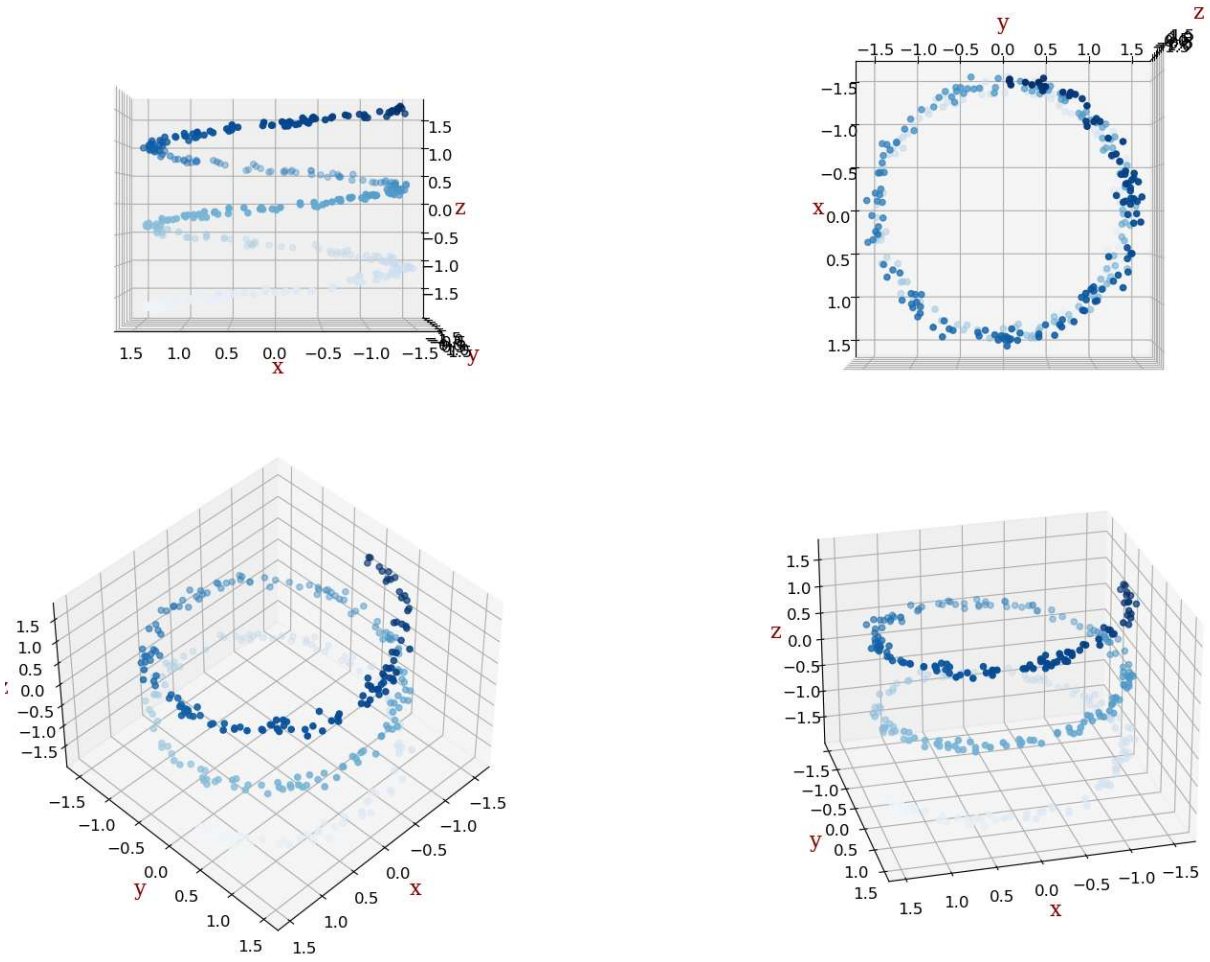
Before moving on to model building, I'll start by visualizing the training data to better understand its structure. To do this, I'll define a custom function called `plot3Ddata`. This function will take a Pandas DataFrame containing three spatial coordinates and generate a 3D scatter plot using `scatter3D()` from Matplotlib.

I'll include the function definition in the earlier "Preliminaries" section of the notebook, keeping the cell below it reserved only for calling the function with the training dataset. At this point, I'll avoid looking at the test set entirely—this helps ensure that my evaluation later on remains unbiased and focused on generalization.

To mirror the structure of the visualizations shown later in the notebook, I'll create a figure with four subplots, each from a different viewing angle using `view_init()`. Each axis will be clearly labeled to match the expected format. This will give a well-rounded perspective of the training data in three dimensions.

In [3]:
```
train_df = pd.DataFrame({
    'x': X_train[:, 0],
    'y': y_train[:,0],
    'z': X_train[:, 1]
})
```

In [4]:
```
plot3Ddata(train_df)
```



A Quick Note

Next, I'll prepare for visualizing the model's predictions alongside the training data. To do this, I'll complete the definition of a function called `plotscatter3Ddata(fit_x, fit_y, fit_z, scat_x, scat_y, scat_z)`, which is already scaffolded in the Preliminaries section of the notebook.

This function will take six NumPy arrays as input: three representing the x, y, and z coordinates of the model's predicted values (`fit_x` , `fit_y` , `fit_z`), and three representing the coordinates of the actual training data (`scat_x` , `scat_y` , `scat_z`). When called, it will generate a 3D scatter plot of the training data with the model's predicted curve superimposed on top.

The completed function will be placed in the designated code cell in the Preliminaries section, and I'll use it later in the notebook to visualize how well the model has learned the underlying pattern in the data. This will be key for assessing model fit and spotting any overfitting or erratic behavior before evaluating on the test set.

Explore 3D Sinusoidal Data with Artifical Neural Networks

With the training data prepared and visualized, the next step will be to fit a `Sequential` model to the dataset. I'll manually set the values for key hyperparameters, including the number of neurons per layer and the number of hidden layers in the network.

The plan is to experiment with different network configurations—adjusting these hyperparameters and observing how they impact the model's performance. This hands-on tuning process will help identify a structure that models the data effectively.

Once I've found a configuration that produces a smooth and accurate fit to the training data, I'll finalize that model and use it to calculate the generalization error in the next phase of the project.

In [5]:
```
import tensorflow as tf
from tensorflow import keras
np.random.seed(42)
tf.random.set_seed(42)
```

In [6]:
```
early_stopping_cb = keras.callbacks.EarlyStopping(patience=20,
                                                  restore_best_weights=True)

model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[2,]),
    keras.layers.Dense(100, activation='tanh'),
    keras.layers.Dense(100, activation='tanh'),
    keras.layers.Dense(80, activation='tanh'),
    keras.layers.Dense(80, activation='tanh'),
    keras.layers.Dense(1)
])

model.compile(loss = 'Huber', optimizer=keras.optimizers.Adam(learning_rate=0.008))

history = model.fit(X_train, y_train,
                    epochs=300,
                    batch_size=200,
                    validation_data=(X_valid, y_valid),
                    callbacks=[early_stopping_cb])
```

```
Epoch 1/300
C:\Users\kevve\anaconda3\Lib\site-packages\keras\src\layers\reshaping\flatten.py:37:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in the mo
del instead.
  super().__init__(**kwargs)
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 97ms/step - loss: 0.7596 - val_loss: 0.4474
Epoch 2/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.5093 - val_loss: 0.5361
Epoch 3/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.4888 - val_loss: 0.4609
Epoch 4/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.4919 - val_loss: 0.4858
Epoch 5/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.4258 - val_loss: 0.5305
Epoch 6/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.4594 - val_loss: 0.5017
Epoch 7/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 0.4515 - val_loss: 0.4493
Epoch 8/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 0.4222 - val_loss: 0.4433
Epoch 9/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.4222 - val_loss: 0.4715
Epoch 10/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.4257 - val_loss: 0.4873
Epoch 11/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.4164 - val_loss: 0.4809
Epoch 12/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.4070 - val_loss: 0.4562
Epoch 13/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.4048 - val_loss: 0.4386
Epoch 14/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.4091 - val_loss: 0.4310
Epoch 15/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.4008 - val_loss: 0.4376
Epoch 16/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.3893 - val_loss: 0.4381
Epoch 17/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.3833 - val_loss: 0.4208
Epoch 18/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.3778 - val_loss: 0.4133
Epoch 19/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.3730 - val_loss: 0.4206
Epoch 20/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.3736 - val_loss: 0.4119
Epoch 21/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.3746 - val_loss: 0.3952
Epoch 22/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.3721 - val_loss: 0.4003
Epoch 23/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.3692 - val_loss: 0.4004
Epoch 24/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.3654 - val_loss: 0.3971
Epoch 25/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.3630 - val_loss: 0.4018
Epoch 26/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.3604 - val_loss: 0.3850
Epoch 27/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 0.3569 - val_loss: 0.3823
Epoch 28/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.3542 - val_loss: 0.3790
Epoch 29/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.3524 - val_loss: 0.3690
Epoch 30/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.3485 - val_loss: 0.3572
Epoch 31/300
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.3469 - val_loss: 0.3456
Epoch 32/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.3434 - val_loss: 0.3474
Epoch 33/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.3376 - val_loss: 0.3382
Epoch 34/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 0.3335 - val_loss: 0.3288
Epoch 35/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.3268 - val_loss: 0.3210
Epoch 36/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.3240 - val_loss: 0.3166
Epoch 37/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.3169 - val_loss: 0.3106
Epoch 38/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.3162 - val_loss: 0.2932
Epoch 39/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.3088 - val_loss: 0.3027
Epoch 40/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.3094 - val_loss: 0.2813
Epoch 41/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.2979 - val_loss: 0.2878
Epoch 42/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 0.2961 - val_loss: 0.2554
Epoch 43/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.2844 - val_loss: 0.2746
Epoch 44/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.2765 - val_loss: 0.2325
Epoch 45/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.2733 - val_loss: 0.2532
Epoch 46/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.2531 - val_loss: 0.2243
Epoch 47/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.2502 - val_loss: 0.2140
Epoch 48/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.2331 - val_loss: 0.2084
Epoch 49/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.2339 - val_loss: 0.2138
Epoch 50/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.2431 - val_loss: 0.2303
Epoch 51/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.2542 - val_loss: 0.1972
Epoch 52/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.2282 - val_loss: 0.2137
Epoch 53/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.2321 - val_loss: 0.2014
Epoch 54/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.2146 - val_loss: 0.1881
Epoch 55/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.2173 - val_loss: 0.1680
Epoch 56/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.2007 - val_loss: 0.1820
Epoch 57/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.1967 - val_loss: 0.1517
Epoch 58/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.1747 - val_loss: 0.1461
Epoch 59/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.1624 - val_loss: 0.1113
Epoch 60/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.1328 - val_loss: 0.1291
Epoch 61/300
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.1157 - val_loss: 0.0580
Epoch 62/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.0574 - val_loss: 0.0465
Epoch 63/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0358 - val_loss: 0.0222
Epoch 64/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0233 - val_loss: 0.0269
Epoch 65/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.0319 - val_loss: 0.0815
Epoch 66/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0816 - val_loss: 0.0280
Epoch 67/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0280 - val_loss: 0.0491
Epoch 68/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0397 - val_loss: 0.0494
Epoch 69/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0519 - val_loss: 0.0741
Epoch 70/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0472 - val_loss: 0.0534
Epoch 71/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0450 - val_loss: 0.0569
Epoch 72/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0380 - val_loss: 0.0378
Epoch 73/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0306 - val_loss: 0.0231
Epoch 74/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0224 - val_loss: 0.0175
Epoch 75/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0183 - val_loss: 0.0212
Epoch 76/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0197 - val_loss: 0.0175
Epoch 77/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0155 - val_loss: 0.0140
Epoch 78/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0157 - val_loss: 0.0255
Epoch 79/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0156 - val_loss: 0.0144
Epoch 80/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0155 - val_loss: 0.0160
Epoch 81/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0134 - val_loss: 0.0161
Epoch 82/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0129 - val_loss: 0.0125
Epoch 83/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0126 - val_loss: 0.0146
Epoch 84/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0112 - val_loss: 0.0146
Epoch 85/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0125 - val_loss: 0.0144
Epoch 86/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0114 - val_loss: 0.0118
Epoch 87/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0110 - val_loss: 0.0130
Epoch 88/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0109 - val_loss: 0.0122
Epoch 89/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0106 - val_loss: 0.0123
Epoch 90/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0107 - val_loss: 0.0123
Epoch 91/300
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.0102 - val_loss: 0.0117
Epoch 92/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0104 - val_loss: 0.0118
Epoch 93/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0103 - val_loss: 0.0121
Epoch 94/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0099 - val_loss: 0.0111
Epoch 95/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0103 - val_loss: 0.0122
Epoch 96/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0097 - val_loss: 0.0112
Epoch 97/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0101 - val_loss: 0.0117
Epoch 98/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0097 - val_loss: 0.0113
Epoch 99/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0099 - val_loss: 0.0114
Epoch 100/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0096 - val_loss: 0.0112
Epoch 101/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0096 - val_loss: 0.0114
Epoch 102/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0096 - val_loss: 0.0112
Epoch 103/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0094 - val_loss: 0.0109
Epoch 104/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0095 - val_loss: 0.0113
Epoch 105/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.0094 - val_loss: 0.0108
Epoch 106/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.0094 - val_loss: 0.0110
Epoch 107/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.0093 - val_loss: 0.0110
Epoch 108/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.0093 - val_loss: 0.0108
Epoch 109/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0093 - val_loss: 0.0108
Epoch 110/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0092 - val_loss: 0.0108
Epoch 111/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0092 - val_loss: 0.0107
Epoch 112/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0091 - val_loss: 0.0107
Epoch 113/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.0091 - val_loss: 0.0106
Epoch 114/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 0.0091 - val_loss: 0.0106
Epoch 115/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0090 - val_loss: 0.0106
Epoch 116/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.0090 - val_loss: 0.0106
Epoch 117/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0090 - val_loss: 0.0105
Epoch 118/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0090 - val_loss: 0.0106
Epoch 119/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.0089 - val_loss: 0.0104
Epoch 120/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0089 - val_loss: 0.0105
Epoch 121/300
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0088 - val_loss: 0.0104
Epoch 122/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.0088 - val_loss: 0.0104
Epoch 123/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0088 - val_loss: 0.0104
Epoch 124/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0087 - val_loss: 0.0103
Epoch 125/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0087 - val_loss: 0.0103
Epoch 126/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 0.0087 - val_loss: 0.0103
Epoch 127/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.0087 - val_loss: 0.0103
Epoch 128/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0086 - val_loss: 0.0103
Epoch 129/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0086 - val_loss: 0.0103
Epoch 130/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.0085 - val_loss: 0.0102
Epoch 131/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.0085 - val_loss: 0.0102
Epoch 132/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0085 - val_loss: 0.0102
Epoch 133/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0085 - val_loss: 0.0101
Epoch 134/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0084 - val_loss: 0.0102
Epoch 135/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0084 - val_loss: 0.0101
Epoch 136/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0084 - val_loss: 0.0102
Epoch 137/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0083 - val_loss: 0.0101
Epoch 138/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0084 - val_loss: 0.0101
Epoch 139/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0083 - val_loss: 0.0101
Epoch 140/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0083 - val_loss: 0.0101
Epoch 141/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0082 - val_loss: 0.0101
Epoch 142/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.0083 - val_loss: 0.0100
Epoch 143/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0082 - val_loss: 0.0101
Epoch 144/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0082 - val_loss: 0.0099
Epoch 145/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0082 - val_loss: 0.0102
Epoch 146/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0081 - val_loss: 0.0098
Epoch 147/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0082 - val_loss: 0.0103
Epoch 148/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0081 - val_loss: 0.0097
Epoch 149/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0082 - val_loss: 0.0103
Epoch 150/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0080 - val_loss: 0.0098
Epoch 151/300
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0083 - val_loss: 0.0102
Epoch 152/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0080 - val_loss: 0.0101
Epoch 153/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0085 - val_loss: 0.0100
Epoch 154/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0081 - val_loss: 0.0109
Epoch 155/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0089 - val_loss: 0.0101
Epoch 156/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0088 - val_loss: 0.0126
Epoch 157/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0098 - val_loss: 0.0108
Epoch 158/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.0099 - val_loss: 0.0152
Epoch 159/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0109 - val_loss: 0.0118
Epoch 160/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0114 - val_loss: 0.0176
Epoch 161/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0118 - val_loss: 0.0127
Epoch 162/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0128 - val_loss: 0.0187
Epoch 163/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0129 - val_loss: 0.0144
Epoch 164/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0158 - val_loss: 0.0214
Epoch 165/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0175 - val_loss: 0.0208
Epoch 166/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0222 - val_loss: 0.0257
Epoch 167/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0204 - val_loss: 0.0194
Epoch 168/300
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0175 - val_loss: 0.0199
```

Plot Model Predictions for Training Set

After finalizing the model, I'll use its `predict()` method to generate predictions for the y values based on the x and z training inputs. Once I have the predicted outputs, I'll visualize them alongside the original training data.

To do this, I'll use the previously defined `plotscatter3Ddata(fit_x, fit_y, fit_z, scat_x, scat_y, scat_z)` function. This will allow me to clearly compare the model's prediction curve against the actual data in 3D space, helping to assess how well the network has learned the underlying pattern.

In [7]:
```
y_pred = model.predict(X_train)
y_pred
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```

Out[7]:
```
array([[ 1.382307  ],
       [ 1.3478147 ],
       [ 0.9305923 ],
       [-1.3390226 ],
       [ 1.0855839 ],
       [ 0.6030176 ],
       [-1.2703333 ],
       [-1.1448557 ],
       [ 0.28109175],
       [ 0.31182885],
       [ 1.3752881 ],
       [ 1.0840849 ],
       [ 0.1678003 ],
       [ 1.2858893 ],
       [ 1.0467572 ],
       [-0.22525552],
       [ 0.8687113 ],
       [ 0.47328094],
       [ 0.5689226 ],
       [-1.0686786 ],
       [ 0.24083576],
       [ 0.5280413 ],
       [ 0.55867875],
       [-1.3769689 ],
       [ 1.3122096 ],
       [ 1.3713176 ],
       [ 1.3872917 ],
       [-0.41572902],
       [ 0.17713115],
       [ 1.0363495 ],
       [ 0.63306373],
       [-1.3714254 ],
       [-0.5873329 ],
       [ 0.3905378 ],
       [ 0.9975099 ],
       [ 0.31160653],
       [ 0.7196976 ],
       [ 0.97404766],
       [ 0.3877966 ],
       [-1.2759585 ],
       [ 0.48995322],
       [-0.37823617],
       [-0.23983482],
       [ 0.3726124 ],
       [ 0.43043688],
       [-0.43407366],
       [ 1.3647714 ],
       [ 1.284941  ],
       [ 0.4697726 ],
       [-0.69420934],
       [ 0.35532507],
       [ 0.30228725],
       [-0.8241782 ],
       [-1.0156122 ],
       [ 1.1089361 ],
       [-0.26387215],
       [ 0.9985581 ],
       [ 0.5855705 ],
       [ 1.36689   ],
       [ 1.3697265 ],
```

```
[ 0.42052153],
[ 1.3009626 ],
[ 0.23244014],
[-1.3483865 ],
[ 1.2055008 ],
[-1.4316962 ],
[ 0.9708659 ],
[-0.5016344 ],
[-1.4570074 ],
[-1.1071134 ],
[ 1.3744358 ],
[-1.0990266 ],
[-0.6792837 ],
[ 0.52160716],
[-0.09143943],
[ 1.2886018 ],
[-1.4193895 ],
[ 1.3470836 ],
[ 0.80272585],
[-1.2939222 ],
[ 0.57985544],
[-0.01109201],
[-0.1935513 ],
[ 1.3205111 ],
[-0.19187671],
[-0.6851479 ],
[ 0.48736468],
[ 1.2820714 ],
[ 0.40044835],
[-1.4195573 ],
[ 1.1518011 ],
[ 0.286044  ],
[ 0.10318085],
[ 1.1853079 ],
[ 0.5548158 ],
[-1.4168061 ],
[ 1.2381177 ],
[-1.4205089 ],
[ 0.38259485],
[ 1.3715887 ],
[-0.9587797 ],
[ 1.3482809 ],
[ 1.0887641 ],
[-0.5913803 ],
[-1.4375913 ],
[ 0.9138134 ],
[ 1.3191888 ],
[ 1.1581287 ],
[ 0.1988793 ],
[ 0.952504  ],
[ 1.3079331 ],
[-0.3342885 ],
[ 0.38025555],
[-1.3345435 ],
[-0.48070917],
[-1.1148174 ],
[ 1.3720345 ],
[ 1.3229103 ],
[ 1.3138547 ],
[-0.8626305 ],
```

```
[ 1.3542428 ],
[ 0.31295472],
[ 0.38249093],
[ 1.3109795 ],
[ 0.52948856],
[ 1.3085486 ],
[ 0.13649914],
[ 1.0532779 ],
[ 1.3670785 ],
[ 0.75769305],
[ 1.3407031 ],
[ 1.3947564 ],
[-1.1780219 ],
[ 1.2161927 ],
[ 1.3997841 ],
[-0.17339778],
[-0.1765174 ],
[-0.94521904],
[ 0.7845865 ],
[-1.4213537 ],
[-0.07765477],
[-0.96980417],
[-0.65457886],
[ 1.3811632 ],
[ 1.3132498 ],
[-0.2552658 ],
[-0.77884907],
[ 1.3726251 ],
[ 1.3853086 ],
[-0.08568165],
[-0.4287515 ],
[-1.0211997 ],
[ 0.3681536 ],
[-0.6944635 ],
[-0.10384161],
[ 1.3815703 ],
[-1.396445  ],
[-0.17846143],
[-0.55185527],
[ 0.9599253 ],
[-1.4201628 ],
[-1.3530264 ],
[ 1.1778133 ],
[ 0.59456384],
[ 1.3734329 ],
[ 0.6096027 ],
[ 1.219848  ],
[ 0.6963408 ],
[-0.8735753 ],
[ 1.0762824 ],
[ 1.0986449 ],
[-0.53392017],
[-0.3177285 ],
[-1.3089769 ],
[ 0.53947616],
[-0.3718674 ],
[-1.091765  ],
[ 1.0687994 ],
[ 1.1091877 ],
[-1.2367034 ],
```

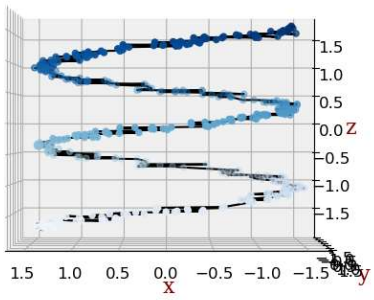
```
       [ 0.33975136],
       [-1.1991514 ],
       [ 0.30695325],
       [-0.410363  ],
       [ 0.05496251],
       [-0.5122165 ],
       [ 0.04144373],
       [ 0.67049783],
       [ 1.3938649 ],
       [ 0.5270274 ],
       [ 0.31485942],
       [ 0.01295444],
       [ 1.3350999 ],
       [ 1.3800566 ],
       [ 0.4132091 ],
       [ 1.3596721 ],
       [-0.17309502],
       [ 1.3434768 ],
       [ 1.2954768 ],
       [ 0.5805954 ],
       [ 0.19947866],
       [ 0.18345577],
       [ 0.9483884 ],
       [-0.6691869 ],
       [-1.2195617 ],
       [ 1.2844843 ],
       [ 0.16135937],
       [ 0.9538239 ],
       [-1.2719028 ],
       [ 1.2937955 ],
       [-1.2762938 ],
       [ 1.3505322 ],
       [ 1.3760703 ],
       [-0.38072777],
       [-1.0659655 ],
       [ 1.1246014 ],
       [ 1.391934  ],
       [ 1.2618484 ],
       [ 0.69860363],
       [ 1.2868952 ],
       [ 0.7478031 ],
       [ 0.6711977 ],
       [ 0.77793205],
       [ 0.8918717 ],
       [ 1.2076976 ],
       [-0.43622687],
       [ 1.1056879 ],
       [ 0.9123709 ],
       [ 1.3749589 ],
       [ 1.0906504 ],
       [ 1.022261  ],
       [-1.2140551 ],
       [ 1.3853121 ],
       [ 1.066803  ],
       [-0.3538115 ],
       [ 1.3572309 ],
       [ 1.1508077 ],
       [-1.3194611 ],
       [ 0.16549578],
       [-0.21236491],
```

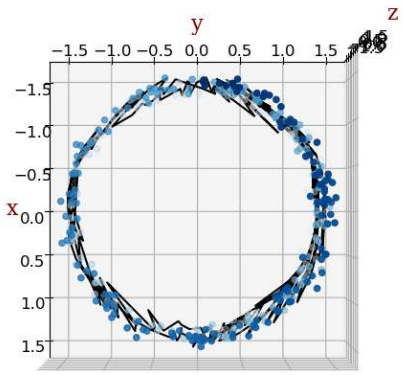
```
[-0.02262969],
[ 0.27079624],
[ 1.3792727 ],
[ 1.3019965 ],
[ 0.39482704],
[-1.1642963 ],
[ 0.22701985],
[-1.415121  ],
[-1.220114  ],
[ 1.2455    ],
[ 1.3646903 ],
[ 0.39885306],
[-0.00429473],
[ 0.3772849 ],
[ 0.27485186],
[ 1.0053562 ],
[-1.3778403 ],
[ 1.1917957 ],
[ 0.50808394],
[-1.2047403 ],
[ 1.015983  ],
[ 0.3705335 ],
[-0.48115155],
[ 1.0041397 ],
[ 0.6139743 ],
[ 0.3904193 ],
[-1.3650577 ],
[-1.0180839 ],
[ 1.1510204 ],
[ 0.2663314 ],
[ 0.3416127 ],
[ 1.3232993 ],
[-0.9953064 ],
[ 0.5170033 ],
[-1.4195373 ],
[ 1.2691535 ],
[-1.3127989 ],
[ 1.374857  ],
[ 0.6553744 ],
[ 0.6099262 ],
[-1.166825  ],
[-0.72549677],
[ 0.93659765],
[ 1.2570319 ],
[ 1.2299291 ],
[ 1.1043495 ],
[-0.11848772],
[ 0.83135515],
[ 0.35513955],
[ 1.0768728 ],
[-0.6455106 ],
[ 0.847519  ],
[-1.3728449 ],
[ 0.43549433],
[ 1.2070663 ],
[ 0.57697237],
[-0.7399721 ],
[ 0.6115694 ],
[ 1.3494003 ],
[ 0.31420976],
```

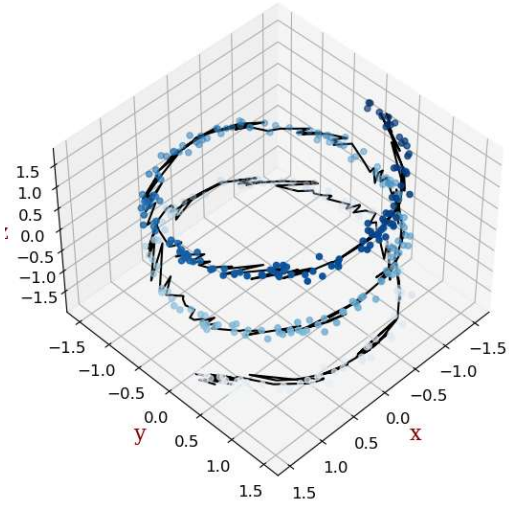
```
[ 1.2343457 ],
[-1.0595554 ],
[-0.9457983 ],
[ 1.3838232 ],
[ 1.2108624 ],
[-1.4143829 ],
[-1.4012556 ],
[-0.42184582],
[-0.86595565],
[ 0.99003506],
[ 1.1103173 ],
[ 1.253675  ],
[ 1.3796457 ],
[ 0.10539409],
[-1.2677134 ],
[-0.11027882],
[ 0.09985127],
[-1.4086173 ],
[ 0.15449941],
[ 1.357533  ],
[ 0.54277146],
[-1.1600773 ],
[-0.87185055],
[ 0.5419601 ],
[-0.04483221],
[ 0.2926559 ],
[-1.0764775 ],
[-0.9892423 ],
[-1.3789682 ],
[ 1.3614501 ],
[ 1.14229   ],
[ 0.92449445],
[-1.3777984 ],
[ 0.40764233],
[ 0.2870461 ],
[ 0.21014178],
[ 0.4982895 ],
[-1.0262177 ],
[ 0.25834572],
[-0.22118366],
[ 0.5888022 ],
[ 1.3288987 ],
[ 1.3426468 ],
[ 0.5802575 ],
[-0.71762276],
[-1.4060552 ],
[ 0.8850281 ],
[-1.1940234 ],
[ 0.22708547],
[ 0.7189786 ],
[-1.43632   ],
[ 1.3629924 ],
[ 0.8772292 ],
[ 0.35174903],
[-0.6257854 ],
[ 0.8633455 ],
[-1.373888  ],
[ 0.8707722 ]], dtype=float32)
```

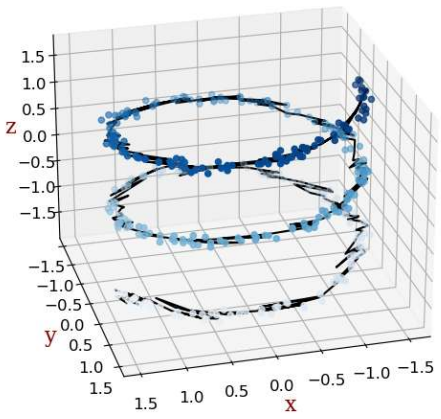
In [8]:
```
fits_df = pd.DataFrame({
    'x': list(X_train[:, 0]),
    'y': list(y_pred[:, 0]),
    'z': list(X_train[:, 1])
})

fits_df_sorted = fits_df.sort_values(by='z', ascending=True)

fit_x = fits_df_sorted['x']
fit_y = fits_df_sorted['y']
fit_z = fits_df_sorted['z']

scat_df = pd.DataFrame({
    'x': list(train_df['x']),
    'y': list(train_df['y']),
    'z': list(train_df['z'])
})

scat_df_sorted = scat_df.sort_values(by='z', ascending=True)

scat_x = scat_df_sorted['x']
scat_y = scat_df_sorted['y']
scat_z = scat_df_sorted['z']
```

In [9]:
```
plotscatter3Ddata(fit_x, fit_y, fit_z, scat_x, scat_y, scat_z)
```









Compute Generalization Error

To evaluate how well the model generalizes to unseen data, I'll compute the generalization error using Mean Squared Error (MSE) as the evaluation metric. This will be calculated by comparing the model's predictions on the test set with the actual test y values.

The resulting MSE score will be rounded to four decimal places for clarity and then printed. This final step will give a quantitative measure of the model's performance on data it hasn't seen before, providing insight into its real-world predictive capability.

In [10]:
```
model.evaluate(X_test, y_test)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0077
```

Out[10]:
```
0.008189760148525238
```

In [11]:
```
from sklearn.metrics import mean_squared_error

predictions = model.predict(X_test)

mse_test = mean_squared_error(y_test, predictions).round(4)
print("Generalization Error: ", mse_test)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Generalization Error:  0.0164
```

In [ ]:
```

```