In [4]:
```
import sympy as sp
import numpy as np

# Define the functions to generate the Taylor expansions for
x = sp.symbols('x')
f1 = sp.sin(x)
f2 = sp.exp(x)
f3 = sp.cos(x)
f4 = sp.log(x+1)
f5 = x**2
f6 = sp.tan(x)
f7 = sp.cosh(x)
f8 = sp.atan(x)
f9 = sp.sin(x)+sp.cos(x)
f10 = x*sp.exp(x)

# Generate the Taylor expansions up to the fourth order
funcs = [f1, f2, f3, f4, f5, f6, f7, f8, f9, f10]
func_evals = [sp.lambdify(x, f.series(x, 0, 5).removeO(), 'numpy') for f in funcs]

# Evaluate the functions at different values of x
x_values = np.linspace(-np.pi, np.pi, 1000)
y_values = np.zeros((len(x_values), len(funcs), 5))

for i, f in enumerate(func_evals):
    y = f(x_values)
    if len(y.shape) == 1:
        y = np.reshape(y, (len(y), 1))
    y_values[:, i, :] = y[:, :5]

# Combine the input and output values for all functions into a single dataset
dataset = []
for i in range(len(x_values)):
    data = [x_values[i]]
    data.extend(y_values[i, :, :].flatten().tolist())
    dataset.append(data)

# Tokenize the dataset
def tokenize_dataset(dataset):
    X = np.zeros((len(dataset), 1))
    Y = np.zeros((len(dataset), len(funcs) * 5))
    for i in range(len(dataset)):
        X[i, 0] = dataset[i][0]
        Y[i, :] = dataset[i][1:]
    return X, Y

X, Y = tokenize_dataset(dataset)
```

In [5]:
```
from sklearn.model_selection import train_test_split

# Split the dataset into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, Y, test_size=0.2, random_stat
```

In [3]:
```
#In the above code, I wrote 10 functions in the code itself.

#In the below code, I have modified the code to take an input file with 'functions
```

In [6]:
```
import sympy as sp
import numpy as np
import pandas as pd
```

```python
# Read the functions from a file
with open('functions.txt') as file:
    function_strings = file.read().splitlines()

# Convert the function strings to SymPy expressions
funcs = [sp.sympify(func) for func in function_strings]

# Generate the Taylor expansions up to the specified order
order = 4
func_evals = [sp.lambdify(x, f.series(x, 0, order+1).removeO(), 'numpy') for f in 

# Evaluate the functions at different values of x
x_values = np.linspace(-np.pi, np.pi, 1000)
x_values_shifted = x_values + np.pi  # shift the input values to be non-negative
y_values = np.zeros((len(x_values), len(funcs), order+1))

for i, f in enumerate(func_evals):
    y = f(x_values)
    if len(y.shape) == 1:
        y = np.reshape(y, (len(y), 1))
    y_values[:, i, :] = y[:, :order+1]

# Combine the input and output values for all functions into a single dataset
dataset = []
for i in range(len(x_values)):
    data = [x_values_shifted[i]]
    data.extend(y_values[i, :, :].flatten().tolist())
    dataset.append(data)

# Tokenize the dataset
def tokenize_dataset(dataset):
    X = np.zeros((len(dataset), 1))
    Y = np.zeros((len(dataset), len(funcs) * (order+1)))
    for i in range(len(dataset)):
        X[i, 0] = dataset[i][0]
        Y[i, :] = dataset[i][1:]
    return X, Y

X, Y = tokenize_dataset(dataset)

from sklearn.model_selection import train_test_split

# Split the dataset into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, Y, test_size=0.2, random_stat
```

In [ ]:

```python
#The following code is to check if I was able to create a dataset, tokenize the da
#then split the dataset into training and validation sets
```

In [7]:

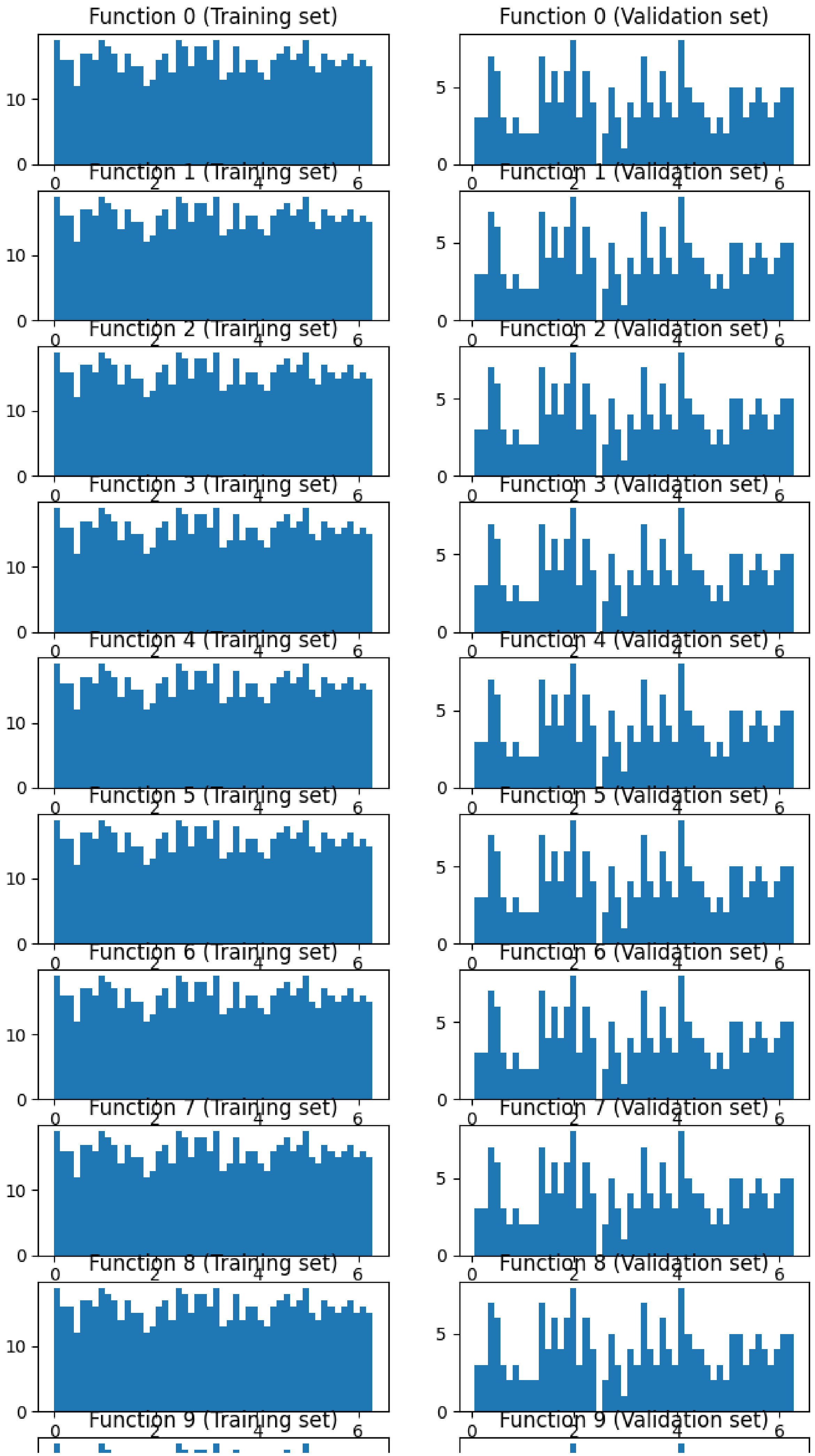
```python
print('X shape:', X.shape)
print('Y shape:', Y.shape)
print('X_train shape:', X_train.shape)
print('y_train shape:', y_train.shape)
print('X_val shape:', X_val.shape)
print('y_val shape:', y_val.shape)
```

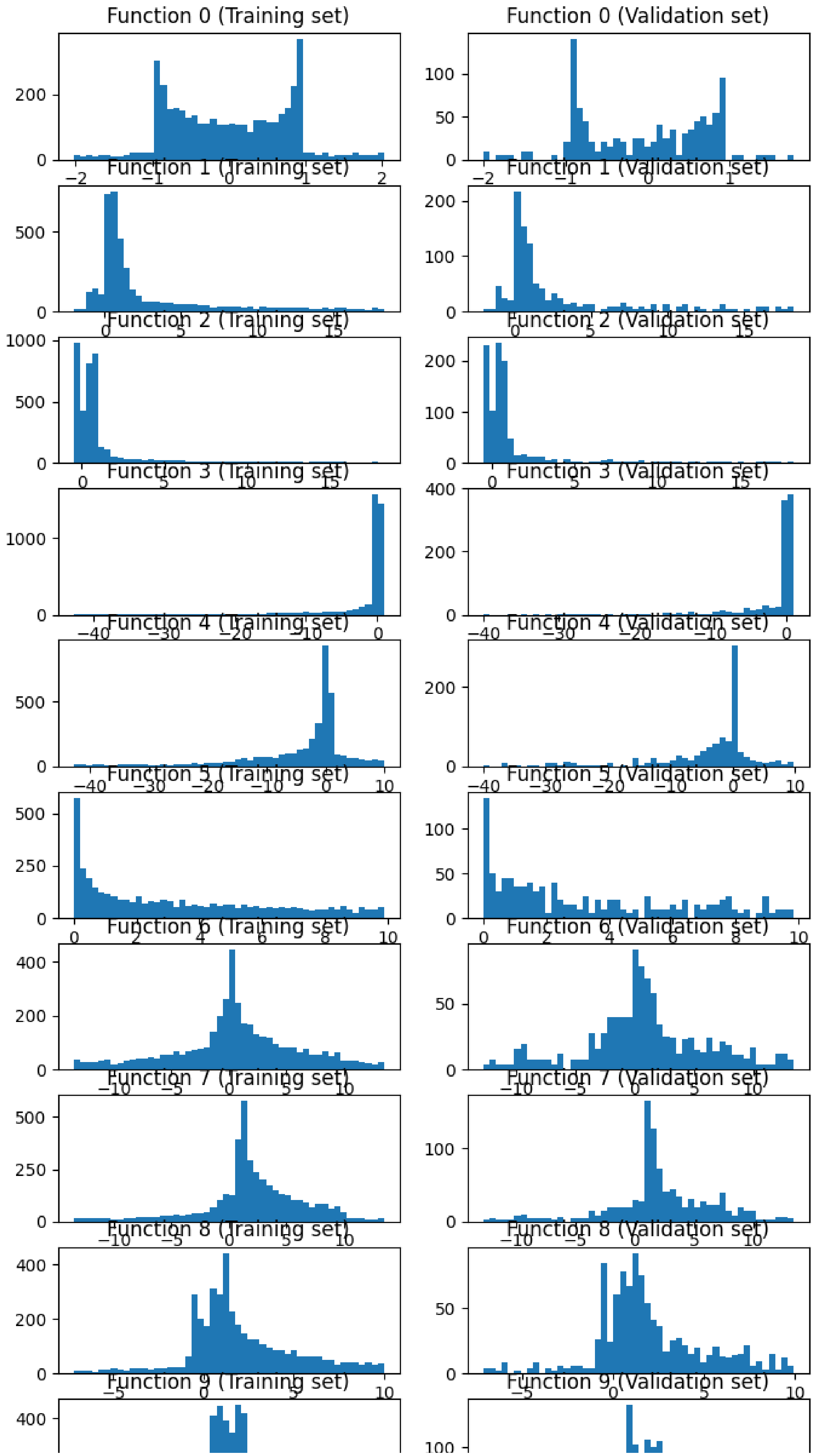
```
X shape: (1000, 1)
Y shape: (1000, 50)
X_train shape: (800, 1)
y_train shape: (800, 50)
X_val shape: (200, 1)
y_val shape: (200, 50)
```

In [8]:
```python
df = pd.DataFrame(dataset, columns=['x'] + [f'f{i}_{j}' for i in range(len(funcs))
print(df.head())
```

```
          x      f0_0      f0_1      f0_2      f0_3      f0_4      f1_0  \
0  0.000000  2.026120  2.026120  2.026120  2.026120  2.026120  1.684209
1  0.006289  2.001434  2.001434  2.001434  2.001434  2.001434  1.669330
2  0.012579  1.976873  1.976873  1.976873  1.976873  1.976873  1.654560
3  0.018868  1.952435  1.952435  1.952435  1.952435  1.952435  1.639901
4  0.025158  1.928120  1.928120  1.928120  1.928120  1.928120  1.625350

       f1_1      f1_2      f1_3  ...      f8_0      f8_1      f8_2      f8_3  \
0  1.684209  1.684209  1.684209  ...  2.150030  2.150030  2.150030  2.150030
1  1.669330  1.669330  1.669330  ...  2.112679  2.112679  2.112679  2.112679
2  1.654560  1.654560  1.654560  ...  2.075606  2.075606  2.075606  2.075606
3  1.639901  1.639901  1.639901  ...  2.038812  2.038812  2.038812  2.038812
4  1.625350  1.625350  1.625350  ...  2.002294  2.002294  2.002294  2.002294

       f8_4      f9_0      f9_1      f9_2      f9_3      f9_4
0  2.150030  7.459722  7.459722  7.459722  7.459722  7.459722
1  2.112679  7.389840  7.389840  7.389840  7.389840  7.389840
2  2.075606  7.320442  7.320442  7.320442  7.320442  7.320442
3  2.038812  7.251527  7.251527  7.251527  7.251527  7.251527
4  2.002294  7.183092  7.183092  7.183092  7.183092  7.183092

[5 rows x 51 columns]
```

In [9]:
```python
import matplotlib.pyplot as plt

# Plot histograms of the input values
fig, axs = plt.subplots(len(funcs), 2, figsize=(8, 16))
for i in range(len(funcs)):
    axs[i, 0].hist(X_train[y_train[:, i*order].nonzero()[0], 0], bins=50)
    axs[i, 0].set_title(f'Function {i} (Training set)')
    axs[i, 1].hist(X_val[y_val[:, i*order].nonzero()[0], 0], bins=50)
    axs[i, 1].set_title(f'Function {i} (Validation set)')
plt.show()

# Plot histograms of the output values
fig, axs = plt.subplots(len(funcs), 2, figsize=(8, 16))
for i in range(len(funcs)):
    axs[i, 0].hist(y_train[:, i*order:i*order+order+1].flatten(), bins=50)
    axs[i, 0].set_title(f'Function {i} (Training set)')
    axs[i, 1].hist(y_val[:, i*order:i*order+order+1].flatten(), bins=50)
    axs[i, 1].set_title(f'Function {i} (Validation set)')
plt.show()
```


Function 0 (Training set)
Function 0 (Validation set)
Function 1 (Training set)
Function 1 (Validation set)
Function 2 (Training set)
Function 2 (Validation set)
Function 3 (Training set)
Function 3 (Validation set)
Function 4 (Training set)
Function 4 (Validation set)
Function 5 (Training set)
Function 5 (Validation set)
Function 6 (Training set)
Function 6 (Validation set)
Function 7 (Training set)
Function 7 (Validation set)
Function 8 (Training set)
Function 8 (Validation set)
Function 9 (Training set)
Function 9 (Validation set)
0
10
5
0
2
4
6


Function 0 (Training set)
Function 0 (Validation set)
Function 1 (Training set)
Function 1 (Validation set)
Function 2 (Training set)
Function 2 (Validation set)
Function 3 (Training set)
Function 3 (Validation set)
Function 4 (Training set)
Function 4 (Validation set)
Function 5 (Training set)
Function 5 (Validation set)
Function 6 (Training set)
Function 6 (Validation set)
Function 7 (Training set)
Function 7 (Validation set)
Function 8 (Training set)
Function 8 (Validation set)
Function 9 (Training set)
Function 9 (Validation set)

In [ ]:
```
#Following code is for the LSTM Model
```

In [10]:
```
import torch
import torch.nn as nn
```

In [11]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
```

In [12]:
```
# Define the LSTM model
class LSTM(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(LSTM, self).__init__()
        self.hidden_size = hidden_size
        self.lstm = nn.LSTM(input_size, hidden_size)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        lstm_out, _ = self.lstm(x.view(len(x), 1, -1))
        output = self.fc(lstm_out[-1])
        return output
```

In [13]:
```
# Define the dataset class
class TaylorDataset(Dataset):
    def __init__(self, X, y):
        self.X = X
        self.y = y

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        return torch.Tensor(self.X[idx]), torch.Tensor(self.y[idx])
```

In [14]:
```
# Initialize the model and the optimizer
input_size = 1
hidden_size = 64
output_size = len(funcs) * (order+1)
model = LSTM(input_size, hidden_size, output_size)
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

In [15]:
```
# Define the loss function
criterion = nn.MSELoss()
```

In [16]:
```
# Define the dataset and data loader
train_dataset = TaylorDataset(X_train, y_train)
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
val_dataset = TaylorDataset(X_val, y_val)
val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)
```

In [ ]:
```
# Train the model
num_epochs = 100
for epoch in range(num_epochs):
    train_loss = 0
```

```
    model.train()
    for X_batch, y_batch in train_loader:
        optimizer.zero_grad()
        y_pred = model(X_batch)
        loss = criterion(y_pred, y_batch)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()
    train_loss /= len(train_loader)

    val_loss = 0
    model.eval()
    with torch.no_grad():
        for X_batch, y_batch in val_loader:
            y_pred = model(X_batch)
            loss = criterion(y_pred, y_batch)
            val_loss += loss.item()
        val_loss /= len(val_loader)

    print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Val Loss: 
```

In [18]:

```
# Convert the validation set inputs to PyTorch tensors
X_val_tensor = torch.Tensor(X_val)

# Make predictions on the validation set
with torch.no_grad():
    model.eval()
    y_pred = model(X_val_tensor).numpy()

# Compute the mean squared error
mse = ((y_pred - y_val)**2).mean()
print("Validation set MSE: ", mse)
```

```
Validation set MSE:  27.73634761848905
```

In [ ]:

```
#Following is the code for LSTM Model too, I am trying to fine tune it.
```

In [19]:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader

# Define the LSTM model
class LSTM(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(LSTM, self).__init__()
        self.hidden_size = hidden_size
        self.lstm = nn.LSTM(input_size, hidden_size)
        self.dropout = nn.Dropout(p=0.2)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        lstm_out, _ = self.lstm(x.view(len(x), 1, -1))
        lstm_out = self.dropout(lstm_out)
        output = self.fc(lstm_out[-1])
        return output

# Define the dataset class
class TaylorDataset(Dataset):
    def __init__(self, X, y):
        self.X = X
        self.y = y
```

```
    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        return torch.Tensor(self.X[idx]), torch.Tensor(self.y[idx])

# Initialize the model and the optimizer
input_size = 1
hidden_size = 64
output_size = len(funcs) * (order+1)
model = LSTM(input_size, hidden_size, output_size)
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Define the loss function
criterion = nn.MSELoss()

# Define the dataset and data loader
train_dataset = TaylorDataset(X_train, y_train)
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
val_dataset = TaylorDataset(X_val, y_val)
val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)

# Train the model
num_epochs = 100
for epoch in range(num_epochs):
    train_loss = 0
    model.train()
    for i, (X_batch, y_batch) in enumerate(train_loader):
        optimizer.zero_grad()
        y_pred = model(X_batch)
        loss = criterion(y_pred, y_batch)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1)
        optimizer.step()
        train_loss += loss.item()
    train_loss /= len(train_loader)

    val_loss = 0
    model.eval()
    with torch.no_grad():
        for X_batch, y_batch in val_loader:
            y_pred = model(X_batch)
            loss = criterion(y_pred, y_batch)
            val_loss += loss.item()
        val_loss /= len(val_loader)

    print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Val Loss: 
```

```
Epoch 1/100, Train Loss: 39.7316, Val Loss: 33.4478
Epoch 2/100, Train Loss: 38.4026, Val Loss: 31.3571
Epoch 3/100, Train Loss: 34.9429, Val Loss: 28.4132
Epoch 4/100, Train Loss: 32.5051, Val Loss: 26.2515
Epoch 5/100, Train Loss: 30.1893, Val Loss: 25.1162
Epoch 6/100, Train Loss: 28.7906, Val Loss: 24.6094
Epoch 7/100, Train Loss: 28.0885, Val Loss: 24.4214
Epoch 8/100, Train Loss: 28.3416, Val Loss: 24.4270
Epoch 9/100, Train Loss: 28.0188, Val Loss: 24.4341
Epoch 10/100, Train Loss: 28.4203, Val Loss: 24.5256
Epoch 11/100, Train Loss: 28.0228, Val Loss: 24.5315
Epoch 12/100, Train Loss: 28.3193, Val Loss: 24.5240
Epoch 13/100, Train Loss: 27.8346, Val Loss: 24.4574
Epoch 14/100, Train Loss: 28.2425, Val Loss: 24.5278
Epoch 15/100, Train Loss: 28.0294, Val Loss: 24.5457
Epoch 16/100, Train Loss: 28.1351, Val Loss: 24.4827
Epoch 17/100, Train Loss: 28.3394, Val Loss: 24.4650
Epoch 18/100, Train Loss: 27.7901, Val Loss: 24.4853
Epoch 19/100, Train Loss: 28.0348, Val Loss: 24.4645
Epoch 20/100, Train Loss: 28.0695, Val Loss: 24.4615
Epoch 21/100, Train Loss: 27.9859, Val Loss: 24.4907
Epoch 22/100, Train Loss: 28.1154, Val Loss: 24.4987
Epoch 23/100, Train Loss: 27.7688, Val Loss: 24.4658
Epoch 24/100, Train Loss: 27.8035, Val Loss: 24.4716
Epoch 25/100, Train Loss: 27.3361, Val Loss: 24.4766
Epoch 26/100, Train Loss: 27.6307, Val Loss: 24.4786
Epoch 27/100, Train Loss: 28.2208, Val Loss: 24.4987
Epoch 28/100, Train Loss: 28.1948, Val Loss: 24.5096
Epoch 29/100, Train Loss: 28.3129, Val Loss: 24.4982
Epoch 30/100, Train Loss: 28.6108, Val Loss: 24.5083
Epoch 31/100, Train Loss: 27.7763, Val Loss: 24.5583
Epoch 32/100, Train Loss: 28.6778, Val Loss: 24.5128
Epoch 33/100, Train Loss: 28.0204, Val Loss: 24.4722
Epoch 34/100, Train Loss: 27.6334, Val Loss: 24.5150
Epoch 35/100, Train Loss: 28.1559, Val Loss: 24.4781
Epoch 36/100, Train Loss: 28.2511, Val Loss: 24.4976
Epoch 37/100, Train Loss: 27.9849, Val Loss: 24.4773
Epoch 38/100, Train Loss: 28.5367, Val Loss: 24.4315
Epoch 39/100, Train Loss: 28.2898, Val Loss: 24.4370
Epoch 40/100, Train Loss: 27.9151, Val Loss: 24.4424
Epoch 41/100, Train Loss: 28.0955, Val Loss: 24.4533
Epoch 42/100, Train Loss: 27.5555, Val Loss: 24.4656
Epoch 43/100, Train Loss: 27.7959, Val Loss: 24.4594
Epoch 44/100, Train Loss: 27.9989, Val Loss: 24.4290
Epoch 45/100, Train Loss: 28.1333, Val Loss: 24.4298
Epoch 46/100, Train Loss: 28.2860, Val Loss: 24.4728
Epoch 47/100, Train Loss: 28.1631, Val Loss: 24.5628
Epoch 48/100, Train Loss: 28.1747, Val Loss: 24.5520
Epoch 49/100, Train Loss: 27.7942, Val Loss: 24.4463
Epoch 50/100, Train Loss: 28.0117, Val Loss: 24.3930
Epoch 51/100, Train Loss: 27.9459, Val Loss: 24.4105
Epoch 52/100, Train Loss: 28.4847, Val Loss: 24.4030
Epoch 53/100, Train Loss: 28.2651, Val Loss: 24.4715
Epoch 54/100, Train Loss: 27.5438, Val Loss: 24.5836
Epoch 55/100, Train Loss: 27.9445, Val Loss: 24.5732
Epoch 56/100, Train Loss: 28.2483, Val Loss: 24.5260
Epoch 57/100, Train Loss: 27.9972, Val Loss: 24.5208
Epoch 58/100, Train Loss: 28.2638, Val Loss: 24.5210
Epoch 59/100, Train Loss: 27.8823, Val Loss: 24.4571
Epoch 60/100, Train Loss: 28.3261, Val Loss: 24.4796
Epoch 61/100, Train Loss: 28.0424, Val Loss: 24.4361
Epoch 62/100, Train Loss: 28.3519, Val Loss: 24.3817
Epoch 63/100, Train Loss: 27.8220, Val Loss: 24.3822
Epoch 64/100, Train Loss: 27.9652, Val Loss: 24.3891
```

```
Epoch 65/100, Train Loss: 28.0402, Val Loss: 24.4070
Epoch 66/100, Train Loss: 27.7999, Val Loss: 24.3988
Epoch 67/100, Train Loss: 28.0156, Val Loss: 24.3772
Epoch 68/100, Train Loss: 28.2887, Val Loss: 24.3971
Epoch 69/100, Train Loss: 28.1081, Val Loss: 24.4860
Epoch 70/100, Train Loss: 28.3403, Val Loss: 24.4793
Epoch 71/100, Train Loss: 27.4178, Val Loss: 24.5027
Epoch 72/100, Train Loss: 27.6193, Val Loss: 24.4596
Epoch 73/100, Train Loss: 28.2566, Val Loss: 24.3999
Epoch 74/100, Train Loss: 28.1281, Val Loss: 24.3797
Epoch 75/100, Train Loss: 28.6260, Val Loss: 24.4840
Epoch 76/100, Train Loss: 28.2933, Val Loss: 24.5126
Epoch 77/100, Train Loss: 27.9165, Val Loss: 24.4640
Epoch 78/100, Train Loss: 27.7485, Val Loss: 24.4766
Epoch 79/100, Train Loss: 27.7113, Val Loss: 24.4827
Epoch 80/100, Train Loss: 28.1124, Val Loss: 24.4152
Epoch 81/100, Train Loss: 27.7421, Val Loss: 24.4344
Epoch 82/100, Train Loss: 27.9792, Val Loss: 24.4247
Epoch 83/100, Train Loss: 28.4738, Val Loss: 24.4320
Epoch 84/100, Train Loss: 27.9350, Val Loss: 24.4672
Epoch 85/100, Train Loss: 27.8935, Val Loss: 24.4080
Epoch 86/100, Train Loss: 28.3107, Val Loss: 24.3805
Epoch 87/100, Train Loss: 28.5225, Val Loss: 24.4359
Epoch 88/100, Train Loss: 27.9515, Val Loss: 24.4686
Epoch 89/100, Train Loss: 28.2026, Val Loss: 24.4031
Epoch 90/100, Train Loss: 28.1320, Val Loss: 24.4723
Epoch 91/100, Train Loss: 28.5269, Val Loss: 24.4352
Epoch 92/100, Train Loss: 28.1788, Val Loss: 24.4932
Epoch 93/100, Train Loss: 27.9640, Val Loss: 24.4772
Epoch 94/100, Train Loss: 27.7069, Val Loss: 24.4512
Epoch 95/100, Train Loss: 27.7885, Val Loss: 24.4022
Epoch 96/100, Train Loss: 28.1033, Val Loss: 24.4287
Epoch 97/100, Train Loss: 28.5088, Val Loss: 24.4644
Epoch 98/100, Train Loss: 28.0082, Val Loss: 24.4428
Epoch 99/100, Train Loss: 27.5276, Val Loss: 24.5096
Epoch 100/100, Train Loss: 27.6717, Val Loss: 24.4774
```

In [20]:

```
# Convert the validation set inputs to PyTorch tensors
X_val_tensor = torch.Tensor(X_val)

# Make predictions on the validation set
with torch.no_grad():
    model.eval()
    y_pred = model(X_val_tensor).numpy()

# Compute the mean squared error
mse = ((y_pred - y_val)**2).mean()
print("Validation set MSE: ", mse)
```

```
Validation set MSE:  27.712765151500697
```

In [ ]:

```
#Following code is for Transformer Model:-
```

In [21]:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
```

In [22]:

```
# Set the device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

In [23]:

```
import torch
import torch.nn as nn
```

```
import torch.optim as optim

class TransformerModel(nn.Module):
    def __init__(self, ntoken, ninp, nhead, nhid, nlayers, dropout=0.5):
        super(TransformerModel, self).__init__()
        from torch.nn import TransformerEncoder, TransformerEncoderLayer
        self.model_type = 'Transformer'
        self.src_mask = None
        self.pos_encoder = nn.Sequential(
            nn.Linear(1, ninp),
            nn.Tanh(),
            nn.Linear(ninp, ninp),
            nn.Tanh(),
            nn.Linear(ninp, ninp),
            nn.Tanh()
        )
        encoder_layers = TransformerEncoderLayer(ninp, nhead, nhid, dropout)
        self.transformer_encoder = TransformerEncoder(encoder_layers, nlayers)
        self.decoder = nn.Linear(ninp, ntoken)

        self.init_weights()

    def init_weights(self):
        initrange = 0.1
        self.pos_encoder[0].weight.data.uniform_(-initrange, initrange)
        self.pos_encoder[0].bias.data.zero_()
        self.pos_encoder[2].weight.data.uniform_(-initrange, initrange)
        self.pos_encoder[2].bias.data.zero_()
        self.pos_encoder[4].weight.data.uniform_(-initrange, initrange)
        self.pos_encoder[4].bias.data.zero_()
        self.decoder.weight.data.uniform_(-initrange, initrange)
        self.decoder.bias.data.zero_()

    def forward(self, src):
        if self.src_mask is None or self.src_mask.size(0) != len(src):
            device = src.device
            mask = self._generate_square_subsequent_mask(len(src)).to(device)
            self.src_mask = mask
        src = self.pos_encoder(src)
        output = self.transformer_encoder(src, self.src_mask)
        output = self.decoder(output)
        return output.transpose(0,1)

    def _generate_square_subsequent_mask(self, sz):
        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        mask = (torch.triu(torch.ones(sz, sz)) == 1).transpose(0, 1)
        mask = mask.float().masked_fill(mask == 0, float('-inf')).masked_fill(mask
        return mask

# Set the hyperparameters
ntoken = len(funcs) * (order+1)  # number of tokens (outputs)
ninp = 256  # dimension of the embeddings
nhead = 8  # number of attention heads
nhid = 512  # dimension of the feedforward network
nlayers = 6  # number of layers
dropout = 0.2  # dropout probability

# Initialize the model
model = TransformerModel(ntoken, ninp, nhead, nhid, nlayers, dropout)

# Set the loss function and optimizer
criterion = nn.MSELoss()
lr = 0.001
optimizer = optim.Adam(model.parameters(), lr=lr)
```

```
# Set the number of epochs and batch size
epochs = 100
batch_size = 64

# Convert the data to PyTorch tensors
X_train_tensor = torch.from_numpy(X_train).float()
y_train_tensor = torch.from_numpy(y_train).float()
X_val_tensor = torch.from_numpy(X_val).float()
y_val_tensor = torch.from_numpy(y_val).float()

# Train the model
for epoch in range(1, epochs + 1):
    model.train()

torch.save(model.state_dict(), 'best_model.pt')
```