Report for Sep 29

Introduction

This report focuses on testing a dataset with $D = 1$ using the Stochastic Gradient Descent (SGD) method. In the current phase, we modified the forward function by introducing a new `nn.Module`, simulating `DivideByConstantLayer` instead of directly dividing by m after applying $< \vec{a}, \cdot >$.

```
import torch
import torch.nn as nn

class DivideByConstantLayer(nn.Module):
    def __init__(self, constant):
        super(DivideByConstantLayer, self).__init__()
        self.constant = constant

    def forward(self, x):
        return x / self.constant
```

The model is defined as follows, employing HE initialization:

```
class Model(nn.Module):
    def __init__(self, m, D):
        super(Model, self).__init__()
        self.linear1 = torch.nn.Linear(D + 1, m)
        self.tanh = torch.nn.Tanh()
        self.linear2 = torch.nn.Linear(m, 1)
        self.divide = DivideByConstantLayer(m)

        # HE initialization
        torch.nn.init.kaiming_normal_(self.linear1.weight)
        torch.nn.init.kaiming_normal_(self.linear2.weight)

    def forward(self, x):
        fc1 = self.linear1(x)
        fc2 = self.tanh(fc1)
        fc3 = self.linear2(fc2)
        fc4 = self.divide(fc3)
        return fc4
```

The parameters are set as follows: $D = 1, m = 50$, batch size = 32.

The relative error is computed as:

$$\text{error} = \frac{|y_{\text{pred}} - y_{\text{true}}|}{|y_{\text{true}}|}$$

Stochastic Gradient Descent (SGD) Method

We selected PyTorch's SGD optimizer for training.

Default Learning Rate

Initially, we tested the default learning rate with 100,000 epochs:

```
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

After 10,000 epochs, the results were:

- Train Loss: 0.005463460925966501
- Test Loss: 0.002377670258283615
- Relative Error: 0.15652328729629517

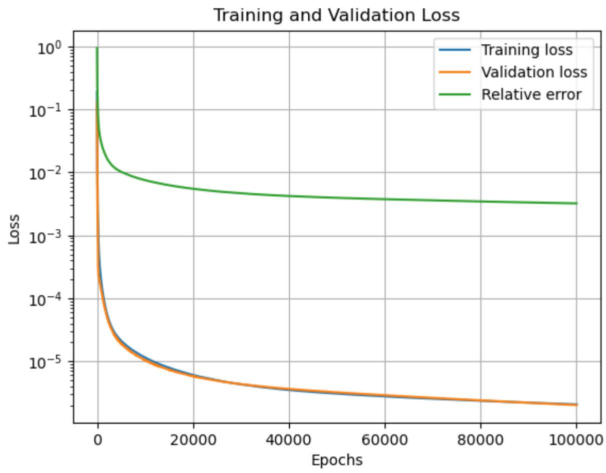
The default SGD method failed to adequately fit the curve. We then adjusted the learning rate to find an optimal result.

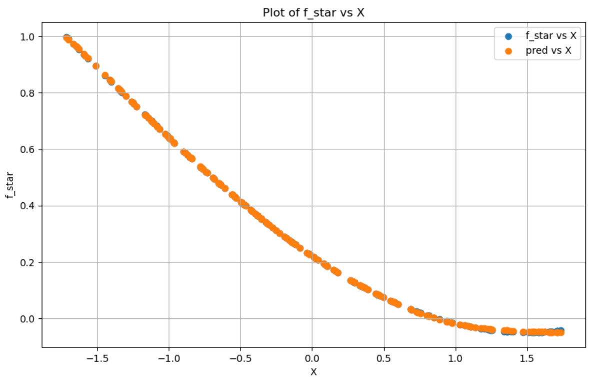
Learning Rate = 1

After 100,000 epochs:

- Train Loss: 2.066×10^{-6}
- Test Loss: 2.018×10^{-6}
- Relative Error: 0.003224582178518176

While increasing the learning rate improved the results, we observed significant errors in the interval $x \in [1, 1.5]$.

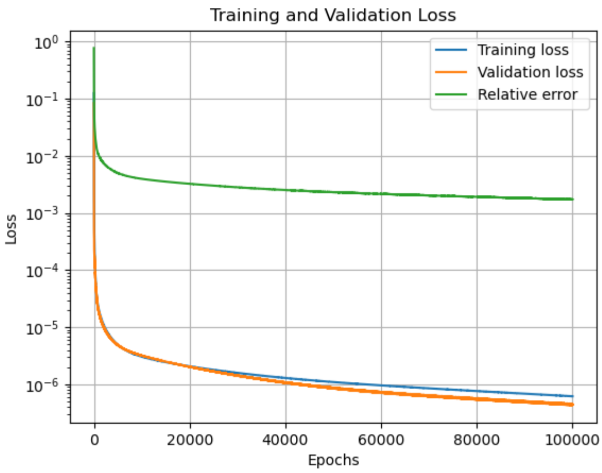


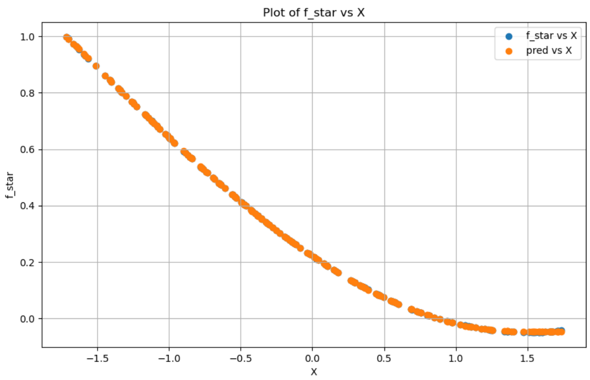


Learning Rate = 5

After 100,000 epochs:

- Train Loss: 6.208×10^{-7}
- Test Loss: 4.469×10^{-7}
- Relative Error: 0.0017211942467838526

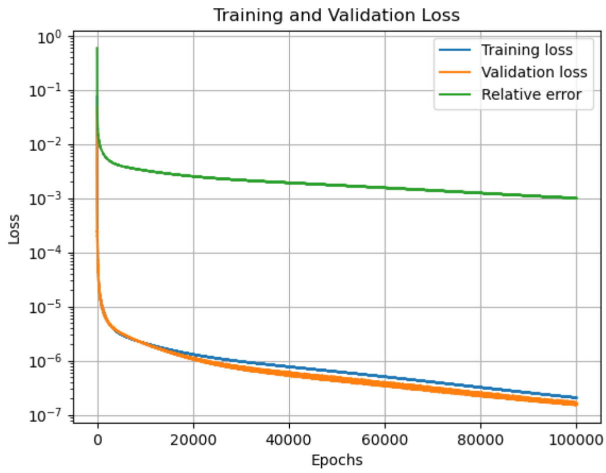


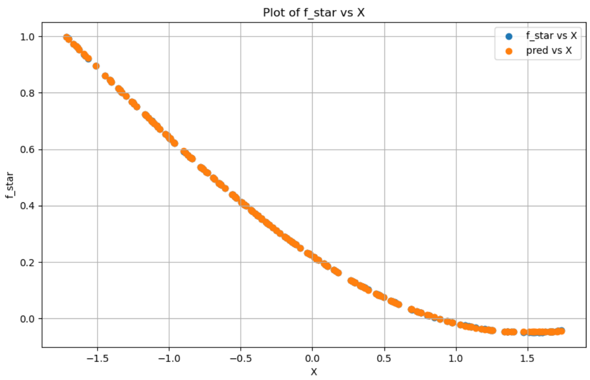


Learning Rate = 10

After 100,000 epochs:

- Train Loss: 2.060×10^{-7}
- Test Loss: 1.590×10^{-7}
- Relative Error: 0.0009969599777832627

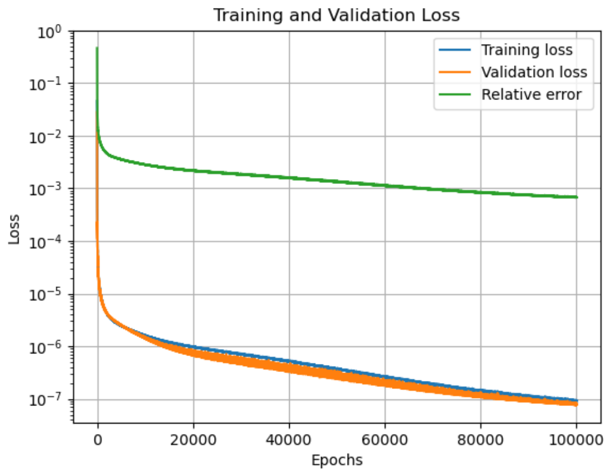


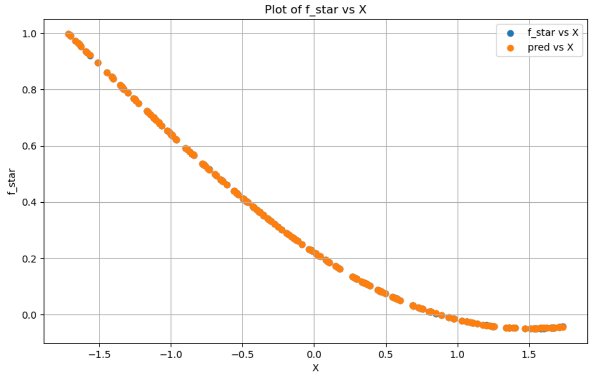


Learning Rate = 15

After 100,000 epochs:

- Train Loss: 9.079×10^{-8}
- Test Loss: 7.902×10^{-8}
- Relative Error: 0.0006686306442134082



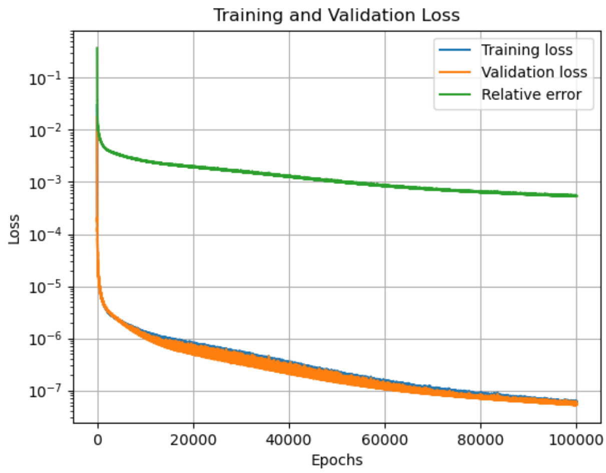


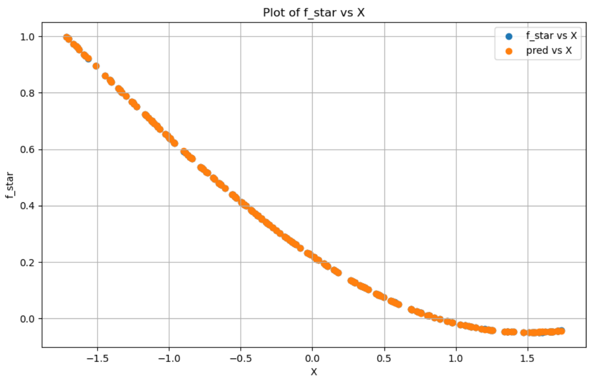
Learning Rate = 20

After 100,000 epochs:

- Train Loss: 5.602×10^{-8}
- Test Loss: 5.357×10^{-8}
- Relative Error: 0.0005298249307088554

Significant oscillations were observed in both training and testing losses.



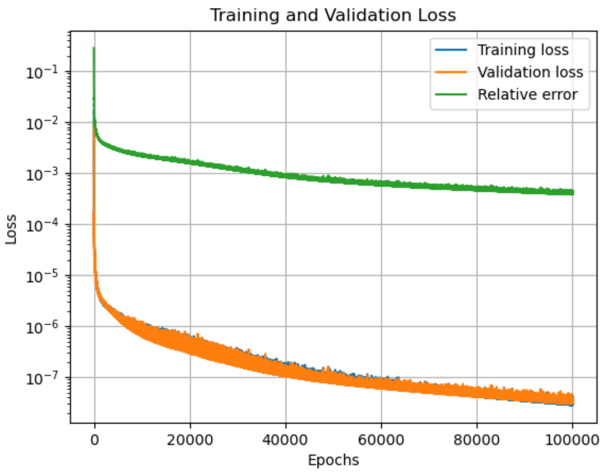


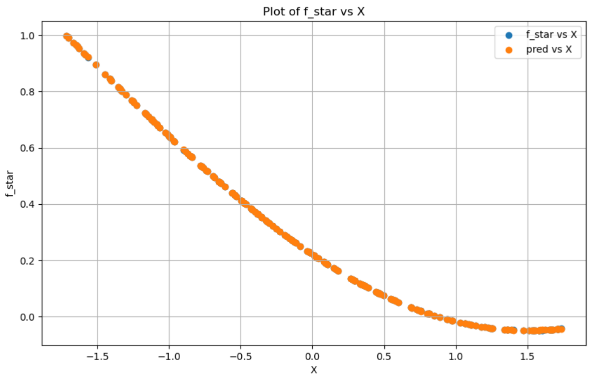
Learning Rate = 30

After 100,000 epochs:

- Train Loss: 2.965×10^{-8}
- Test Loss: 3.113×10^{-8}
- Relative Error: 0.00038910453440621495

Further increasing the learning rate showed diminishing returns, with the relative error converging to approximately 3×10^{-4}. At this stage, we proceeded to experiment with the Adam optimizer.





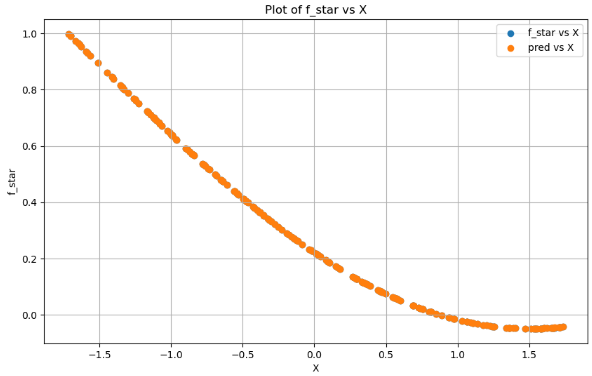
Adam Method

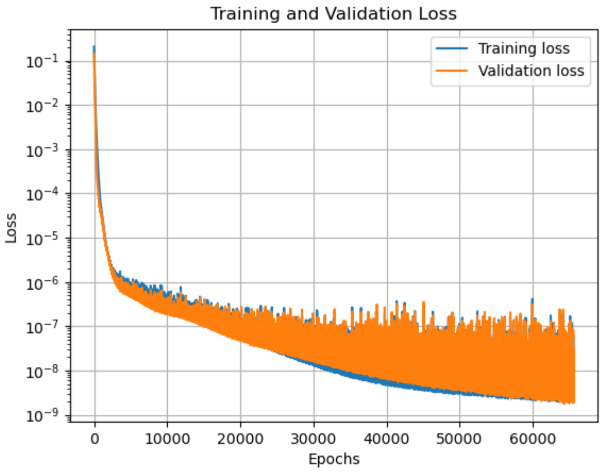
```
optimizer = torch.optim.Adam(model.parameters())
```

Using the default Adam optimizer, we observed the following at Epoch 65,574:

- Train Loss: 1.883×10^{-9}
- Test Loss: 1.899×10^{-9}
- Relative Error: 9.765×10^{-5}

Given that the relative error dropped below 1×10^{-4}, we terminated the process.


Plot of f_star vs X
f_star vs X
pred vs X
f_star
X
1.0
0.8
0.6
0.4
0.2
0.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Training and Validation Loss
Training loss
Validation loss
Loss
Epochs
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
10^{-9}
0
10000
20000
30000
40000
50000
60000