ECGR_4105_HW5_Problem_1_Source_Code

November 16, 2024

[ ]:
```
# Link to Google Colab: https://colab.research.google.com/drive/
↪1uuVb2mNPxP1AjOUYkQ3Zse1hhXMfgxj1?usp=sharing
```

[ ]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
```

[ ]:
```
t_u = torch.linspace(0, 10, 100)  # input values
y_actual = 3.5 * t_u ** 2 + 2 * t_u + 0.5
```

[ ]:
```
# linear model to non-linear model
class NonLinearModel(nn.Module):
    def __init__(self):
        super(NonLinearModel, self).__init__()
        self.w2 = nn.Parameter(torch.randn(()))
        self.w1 = nn.Parameter(torch.randn(()))
        self.b = nn.Parameter(torch.randn(()))

    def forward(self, t_u):
        return self.w2 * t_u ** 2 + self.w1 * t_u + self.b
```

[ ]:
```
# function to train non-linear
def train_model(learning_rate, epochs=5000, log_interval=500):
    model = NonLinearModel()
    criterion = nn.MSELoss()
    optimizer = optim.SGD(model.parameters(), lr=learning_rate)
    losses = []

    for epoch in range(epochs):
        optimizer.zero_grad()
        output = model(t_u)
        loss = criterion(output, y_actual)
        loss.backward()
        optimizer.step()

        if epoch % log_interval == 0:
```

```python
            losses.append(loss.item())
            print(f"Epoch {epoch}, Loss: {loss.item()}")

    return model, losses
```

[ ]:
```python
class LinearModel(nn.Module):
    def __init__(self):
        super(LinearModel, self).__init__()
        self.w = nn.Parameter(torch.randn(()))
        self.b = nn.Parameter(torch.randn(()))

    def forward(self, t_u):
        return self.w * t_u + self.b
```

[ ]:
```python
# function to train linear for comparison
def train_linear_model(learning_rate=0.01, epochs=5000):
    model = LinearModel()
    criterion = nn.MSELoss()
    optimizer = optim.SGD(model.parameters(), lr=learning_rate)

    for epoch in range(epochs):
        optimizer.zero_grad()
        output = model(t_u)
        loss = criterion(output, y_actual)
        loss.backward()
        optimizer.step()

    return model
```

[ ]:
```python
# Training with different learning rates
learning_rates = [0.1, 0.01, 0.001, 0.0001]
all_losses = {}
```

[ ]:
```python
for lr in learning_rates:
    print(f"\nTraining with learning rate: {lr}")
    model, losses = train_model(lr)
    all_losses[lr] = losses
```

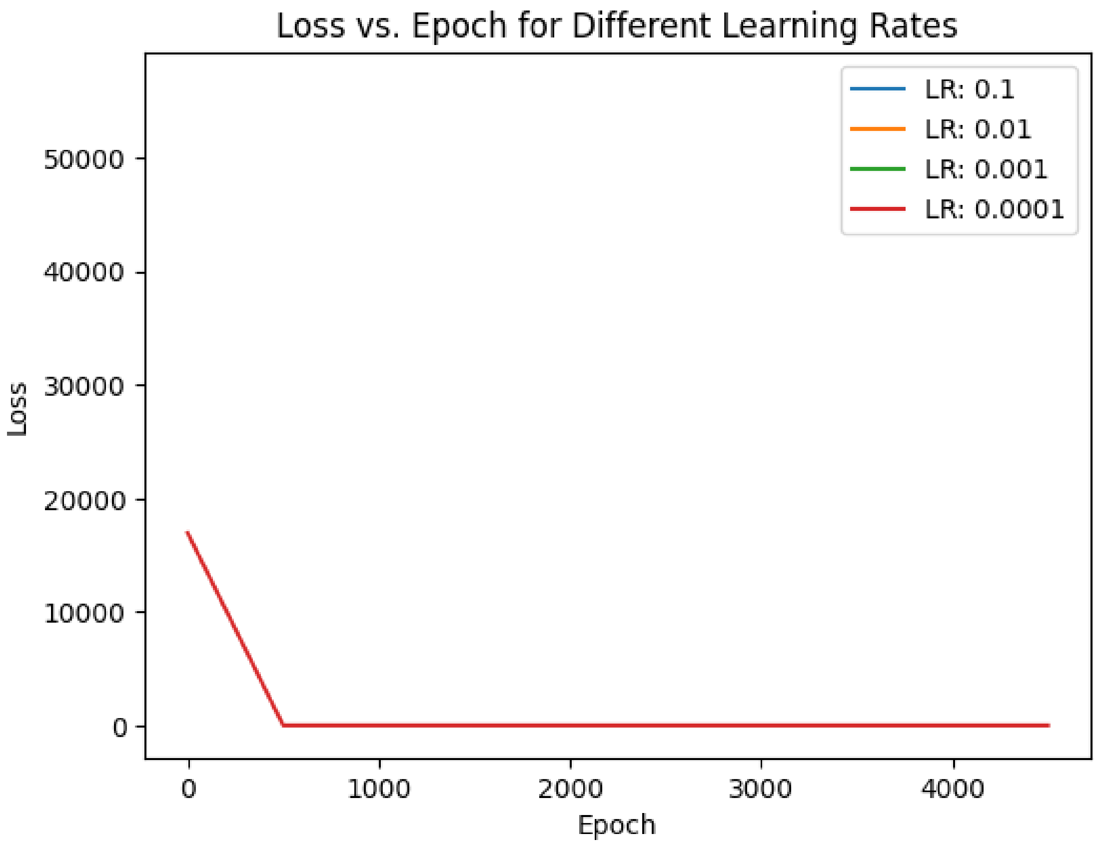
```
Training with learning rate: 0.1
Epoch 0, Loss: 52230.57421875
Epoch 500, Loss: nan
Epoch 1000, Loss: nan
Epoch 1500, Loss: nan
Epoch 2000, Loss: nan
Epoch 2500, Loss: nan
Epoch 3000, Loss: nan
```

```
Epoch 3500, Loss: nan
Epoch 4000, Loss: nan
Epoch 4500, Loss: nan

Training with learning rate: 0.01
Epoch 0, Loss: 56430.42578125
Epoch 500, Loss: nan
Epoch 1000, Loss: nan
Epoch 1500, Loss: nan
Epoch 2000, Loss: nan
Epoch 2500, Loss: nan
Epoch 3000, Loss: nan
Epoch 3500, Loss: nan
Epoch 4000, Loss: nan
Epoch 4500, Loss: nan

Training with learning rate: 0.001
Epoch 0, Loss: 35192.46875
Epoch 500, Loss: nan
Epoch 1000, Loss: nan
Epoch 1500, Loss: nan
Epoch 2000, Loss: nan
Epoch 2500, Loss: nan
Epoch 3000, Loss: nan
Epoch 3500, Loss: nan
Epoch 4000, Loss: nan
Epoch 4500, Loss: nan

Training with learning rate: 0.0001
Epoch 0, Loss: 16955.552734375
Epoch 500, Loss: 2.676705837249756
Epoch 1000, Loss: 1.656842827796936
Epoch 1500, Loss: 1.0265941619873047
Epoch 2000, Loss: 0.6370910406112671
Epoch 2500, Loss: 0.3963528871536255
Epoch 3000, Loss: 0.2475438266992569
Epoch 3500, Loss: 0.15553589165210724
Epoch 4000, Loss: 0.09863147884607315
Epoch 4500, Loss: 0.06342044472694397
```

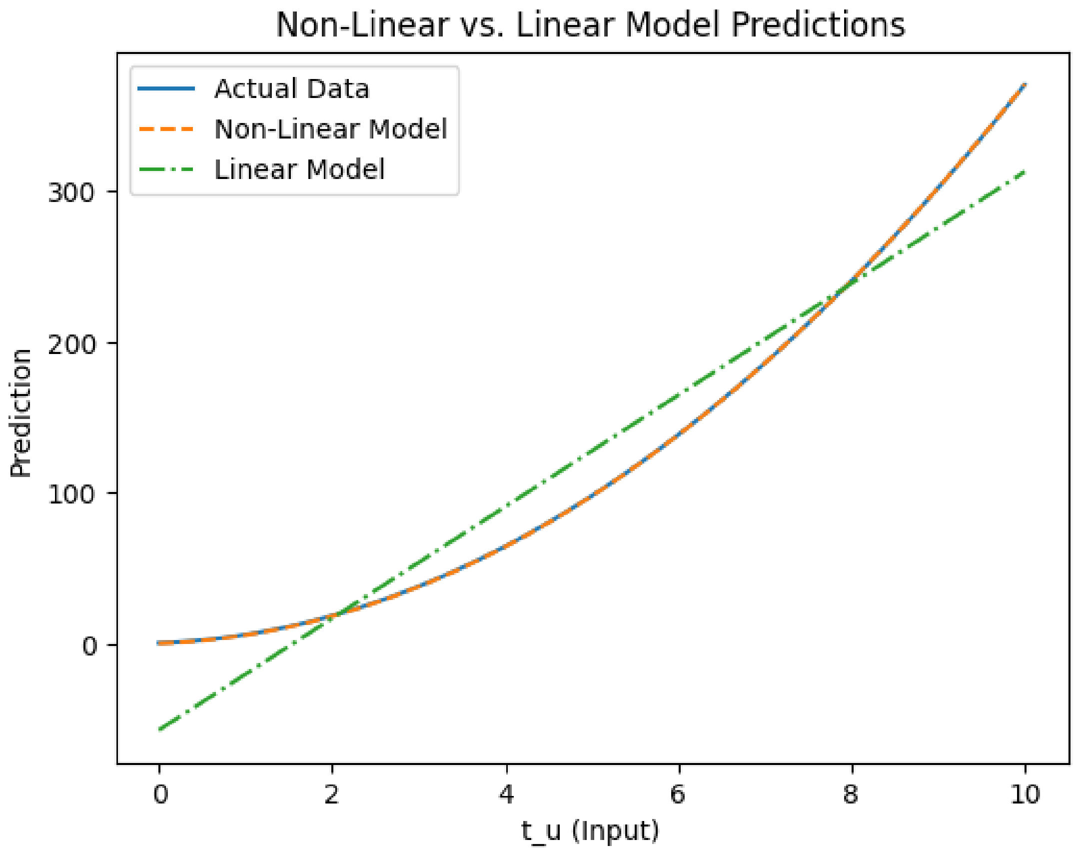
[ ]:

```
# Plotting the losses for each learning rate
for lr, losses in all_losses.items():
    plt.plot(range(0, 5000, 500), losses, label=f"LR: {lr}")

plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
```

```
plt.title("Loss vs. Epoch for Different Learning Rates")
plt.show()
```



[ ]:
```
linear_model = train_linear_model()

# Plotting the models for comparison
plt.plot(t_u, y_actual, label="Actual Data")
plt.plot(t_u, model(t_u).detach(), label="Non-Linear Model", linestyle="--")
plt.plot(t_u, linear_model(t_u).detach(), label="Linear Model", linestyle="-.")
plt.xlabel("t_u (Input)")
plt.ylabel("Prediction")
plt.legend()
plt.title("Non-Linear vs. Linear Model Predictions")
plt.show()
```


Non-Linear vs. Linear Model Predictions
Actual Data
Non-Linear Model
Linear Model
300
200
100
0
Prediction
0
2
4
6
8
10
t_u (Input)