Homework 5 – Code & Graphs

Gilberto Feliu
Student ID: 801257813
https://github.com/gfeliu-rgb/Intro-to-ML-ECGR-4105

Problem 1A

Linear Model Code:

```
def model(t_u, w, b):
    return w * t_u + b
```

Updated Code for Problem 1A:

```
def model(t_u, w2, w1, b):
    return w2 * t_u**2 + w1 * t_u + b
```

```
params = torch.tensor([1.0, 1.0, 0.0], requires_grad=True)
learning_rate = 1e-2
optimizer = optim.SGD([params], lr=learning_rate)

training_loop(
    n_epochs = 5000,
    optimizer = optimizer,
    params = params,
    t_u = t_un,
    t_c = t_c
)
```

Problem 1B

Changed Code:

```
params_init = [1.0, 1.0, 0.0]
learning_rates = [0.1, 0.01, 0.001, 0.0001]

for lr in learning_rates:
    print("Training with learning rate =", lr)
    params = torch.tensor(params_init, requires_grad=True)
    optimizer = optim.SGD([params], lr=lr)
    trained_params = training_loop(
```

```
        n_epochs = 5000,
        optimizer = optimizer,
        params = params,
        t_u = t_un,
        t_c = t_c
    )
    print("Final params:", trained_params)
    print("")
```

Results:

```
Training with learning rate = 0.1 ... Final loss: nan
Training with learning rate = 0.01 ... Final loss: nan
Training with learning rate = 0.001 ... Final loss: nan
Training with learning rate = 0.0001 ... Final loss: 3.8615105152130127
Final params: tensor([-0.8881, 0.5570, -0.8753])
```

Best nonlinear model learning rate: 0.0001
Best nonlinear final loss: 3.8615

Problem 1C

Comparison Code:

```
def training_loop(n_epochs, optimizer, params, t_u, t_c, model_fn):
    for epoch in range(1, n_epochs + 1):
        t_p = model_fn(t_u, *params)
        loss = loss_fn(t_p, t_c)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if epoch % 500 == 0:
            print('Epoch %d, Loss %f' % (epoch, float(loss)))
    return params
```

```
params_linear = torch.tensor([1.0, 0.0], requires_grad=True)
optimizer_linear = optim.SGD([params_linear], lr=0.01)
trained_linear_params = training_loop(
    n_epochs=5000,
    optimizer=optimizer_linear,
    params=params_linear,
    t_u=t_un,
    t_c=t_c,
    model_fn=linear_model
)
pred_linear = linear_model(t_un, *trained_linear_params)
pred_nonlinear = model(t_un, *best_params)
plt.figure(figsize=(8,5))
```

```
plt.scatter(t_un, t_c, color='black', label='Actual data')
plt.plot(t_un, pred_linear.detach(), color='blue', label='Linear model')
plt.plot(t_un, pred_nonlinear.detach(), color='red', label='Non-linear
    model')
plt.legend(); plt.show()
```

Results:

```
Linear model final loss: 2.9276480674743652
Best nonlinear model final loss: 3.8615105152130127
```

Problem 2A

Source Code:

```
import numpy as np, pandas as pd, torch, torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

file_path = "Housing.csv"
df = pd.read_csv(file_path)
inputs = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking']
X = df[inputs].values
Y = df['price'].values
X_0 = np.ones((len(Y), 1))
X = np.hstack((X_0, X))
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)
scaler = MinMaxScaler()
X_train[:,1:] = scaler.fit_transform(X_train[:,1:])
X_test[:,1:] = scaler.transform(X_test[:,1:])
```

Results (2A):

```
Epoch 500 Loss= 1.70e12
Epoch 2000 Loss= 1.44e12
Validation loss: 2.44e12
Trained parameters [B, W1..W5]:
[2508056.5 2933135.8 1395469.2 2351054.8 1603754.9 1423087.1]
```

Problem 2B

Results:

```
Best learning rate = 0.1
Lowest validation loss = 2.29e12 at Epoch 2000
Higher learning rates diverged, lower rates trained slower.
```

Problem 3A

NN Model:

```
class NN(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = torch.nn.Linear(6, 8)
        self.fc2 = torch.nn.Linear(8, 1)
    def forward(self, x):
        x = torch.relu(self.fc1(x))
        return self.fc2(x)
```

Results (3A):

```
Training time = 0.333 s
Final Training Loss = 3.98e12
Evaluation Accuracy (R^2) = -0.3001
```

Problem 3B

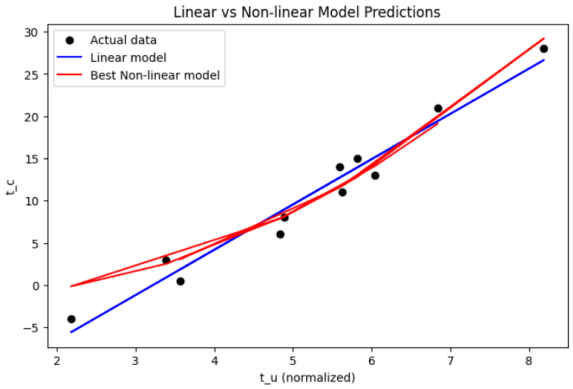
Extended NN Model:

```
class NN(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = torch.nn.Linear(6, 8)
        self.fc2 = torch.nn.Linear(8, 8)
        self.fc3 = torch.nn.Linear(8, 8)
        self.fc4 = torch.nn.Linear(8, 1)
    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        return self.fc4(x)
```

Results (3B):

```
Learning rate 0.01 -> Loss = NaN
Learning rate 1e-9 -> Loss = 2.52e13, R^2 = -4.96
Training time = 0.277 s
```

Graphs


Linear vs Non-linear Model Predictions
Actual data
Linear model
Best Non-linear model
t_c
t_u (normalized)
30
25
20
15
10
5
0
−5
2
3
4
5
6
7
8