Homework 5 Code & Graphs
Name: Viprav Lipare
Student ID: 801288922
Homework Number: 5
Github Repository: https://github.com/vipravlipare/ECGR-5105-Intro-to-Machine-Learning

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)
```

```
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
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
Epoch 5000, Loss nan
Final loss: nan
Final params: tensor([nan, nan, nan], requires_grad=True)

Training with learning rate = 0.01
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
Epoch 5000, Loss nan
Final loss: nan
Final params: tensor([nan, nan, nan], requires_grad=True)

Training with learning rate = 0.001
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
Epoch 5000, Loss nan
Final loss: nan
Final params: tensor([nan, nan, nan], requires_grad=True)

Training with learning rate = 0.0001
Epoch 500, Loss 10.708596
Epoch 1000, Loss 8.642083
Epoch 1500, Loss 7.171005
Epoch 2000, Loss 6.123477
Epoch 2500, Loss 5.377227
Epoch 3000, Loss 4.845285
Epoch 3500, Loss 4.465788
Epoch 4000, Loss 4.194724
Epoch 4500, Loss 4.000802
Epoch 5000, Loss 3.861744
Final loss: 3.8615105152130127
Final params: tensor([-0.8881,  0.5570, -0.8753], requires_grad=True)

Best nonlinear model learning rate: 0.0001
Best nonlinear final loss: 3.8615105152130127
Best params: tensor([-0.8881,  0.5570, -0.8753], grad_fn=<CloneBackward0>)
```

Problem 1C

Changed Code:

```python
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)

loss_linear = loss_fn(pred_linear, t_c)
loss_nonlinear = loss_fn(pred_nonlinear, t_c)
print("\nLinear model final loss:", float(loss_linear))
print("Best nonlinear model final loss:", float(loss_nonlinear))
```

```
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='Best
Non-linear model')
plt.xlabel('t_u (normalized)')
plt.ylabel('t_c')
plt.title('Linear vs Non-linear Model Predictions')
plt.legend()
plt.show()
```

<u>Source Code Problem 1:</u>

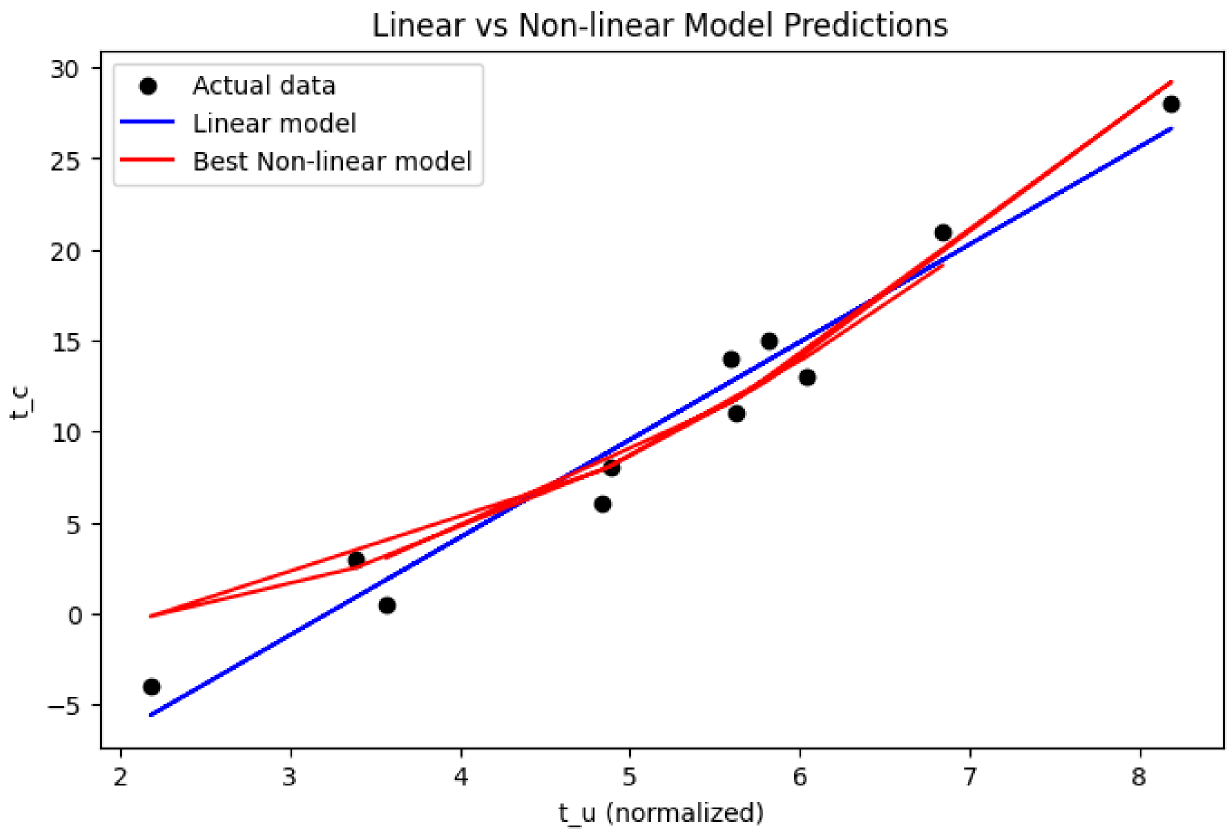
```
import torch
import torch.optim as optim
import matplotlib.pyplot as plt
t_c = [0.5, 14.0, 15.0, 28.0, 11.0, 8.0, 3.0, -4.0, 6.0, 13.0, 21.0]
t_u = [35.7, 55.9, 58.2, 81.9, 56.3, 48.9, 33.9, 21.8, 48.4, 60.4, 68.4]
t_c = torch.tensor(t_c)
t_u = torch.tensor(t_u)
```

```
t_un = t_u * 0.1

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

def linear_model(t_u, w, b):
    return w * t_u + b

def loss_fn(t_p, t_c):
    squared_diffs = (t_p - t_c)**2
    return squared_diffs.mean()

params = torch.tensor([1.0, 1.0, 0.0], requires_grad=True)
w = torch.ones(())
b = torch.zeros(())
t_p = model(t_u, *params)
print(t_p)

loss = loss_fn(t_p, t_c)
print(loss)

params = torch.tensor([1.0, 1.0, 0.0], requires_grad=True)
loss = loss_fn(model(t_u, *params), t_c)
loss.backward()
params.grad
params = torch.tensor([1.0, 1.0, 0.0], requires_grad=True)
learning_rate = 1e-2
optimizer = optim.SGD([params], lr=learning_rate)
tp = model(t_un, *params)
loss = loss_fn(tp, t_c)
optimizer.zero_grad()
loss.backward()
optimizer.step()
params
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()
```

```python
        loss.backward()
        optimizer.step()

        if epoch % 500 == 0:
            print('Epoch %d, Loss %f' % (epoch, float(loss)))

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

best_loss = float('inf')
best_params = None
best_lr = None

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,
        model_fn=model  # nonlinear model
    )

    final_loss = loss_fn(model(t_un, *trained_params), t_c)
    print("Final loss:", float(final_loss))

    if final_loss < best_loss:
        best_loss = final_loss
        best_params = trained_params.clone()
        best_lr = lr

    print("Final params:", trained_params)
    print("")
print("Best nonlinear model learning rate:", best_lr)
print("Best nonlinear final loss:", float(best_loss))
print("Best params:", best_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)

loss_linear = loss_fn(pred_linear, t_c)
loss_nonlinear = loss_fn(pred_nonlinear, t_c)
print("\nLinear model final loss:", float(loss_linear))
print("Best nonlinear model final loss:", float(loss_nonlinear))
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='Best
Non-linear model')
plt.xlabel('t_u (normalized)')
plt.ylabel('t_c')
plt.title('Linear vs Non-linear Model Predictions')
plt.legend()
plt.show()
```

Results:

```
Epoch 500, Loss 7.860115
Epoch 1000, Loss 3.828538
Epoch 1500, Loss 3.092191
Epoch 2000, Loss 2.957698
Epoch 2500, Loss 2.933134
Epoch 3000, Loss 2.928648
Epoch 3500, Loss 2.927830
Epoch 4000, Loss 2.927679
Epoch 4500, Loss 2.927652
Epoch 5000, Loss 2.927647

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



Problem 2A

Source Code (2A):

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

file_path = "Housing.csv"
```

```
df = pd.read_csv(file_path)

inputs = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking']
X = df[inputs].values
Y = df['price'].values
m = len(Y)

X_0 = np.ones((m, 1))
X = np.hstack((X_0, X))
theta = np.zeros(X.shape[1])

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=42)
scaler = MinMaxScaler()
X_train[:, 1:] = scaler.fit_transform(X_train[:, 1:])
X_test[:, 1:] = scaler.transform(X_test[:, 1:])
# Training data
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
Y_train_tensor = torch.tensor(Y_train, dtype=torch.float32).view(-1,1)  #
shape (N,1)

# Validation data
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
Y_test_tensor = torch.tensor(Y_test, dtype=torch.float32).view(-1,1)
params = torch.zeros((6,1), requires_grad=True)

def linear_model(X, params):
    B = params[0]
    W1 = params[1]
    W2 = params[2]
    W3 = params[3]
    W4 = params[4]
    W5 = params[5]

    U = B + W1*X[:,1:2] + W2*X[:,2:3] + W3*X[:,3:4] + W4*X[:,4:5] +
W5*X[:,5:6]
    return U

def loss_fn(y_pred, y_true):
    return ((y_pred - y_true)**2).mean()
```

```python
def training_loop(n_epochs, optimizer, params, X, y):
    for epoch in range(1, n_epochs+1):
        y_pred = linear_model(X, params)
        loss = loss_fn(y_pred, y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if epoch % 500 == 0:
            print("Epoch", epoch, "Loss=", loss.item())


    return params
learning_rate = 0.01
optimizer = optim.SGD([params], lr=learning_rate)
trained_params = training_loop(
    n_epochs=2000,
    optimizer=optimizer,
    params=params,
    X=X_train_tensor,
    y=Y_train_tensor
)

# Evaluate on validation set
with torch.no_grad():
    y_val_pred = linear_model(X_test_tensor, trained_params)
    val_loss = loss_fn(y_val_pred, Y_test_tensor)
    print("Validation loss:", val_loss.item())

# Show trained parameters
print("Trained parameters [B, W1..W5]:")
print(trained_params.detach().view(-1).numpy())
```

<u>Results(2A):</u>

```
Epoch 500 Loss= 1699869229056.0
Epoch 1000 Loss= 1556255735808.0
Epoch 1500 Loss= 1481826238464.0
Epoch 2000 Loss= 1436944433152.0
Validation loss: 2440182366208.0
```

Trained parameters [B, W1..W5]:
[2508056.5 2933135.8 1395469.2 2351054.8 1603754.9 1423087.1]

Problem 2B

Source Code (2B):

```
learning_rates = [0.1, 0.01, 0.001, 0.0001]

for lr in learning_rates:
    print("Training with learning rate =", lr)
    params = torch.zeros((6,1), requires_grad=True)
    optimizer = optim.SGD([params], lr=lr)

    for epoch in range(1, 5001):
        y_pred = linear_model(X_train_tensor, params)
        loss = loss_fn(y_pred, Y_train_tensor)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if epoch % 500 == 0:
            with torch.no_grad():
                y_val = linear_model(X_test_tensor, params)
                val_loss = loss_fn(y_val, Y_test_tensor).item()

            print("Epoch", epoch, "Training Loss=", loss.item(),
"Validation Loss=", val_loss)
```

Results(2B):

```
Training with learning rate = 0.1
Epoch 500 Training Loss= 1358686322688.0 Validation Loss= 2313595125760.0
Epoch 1000 Training Loss= 1350310166528.0 Validation Loss= 2293964472320.0
Epoch 1500 Training Loss= 1350024036352.0 Validation Loss= 2292591099904.0
Epoch 2000 Training Loss= 1350009225216.0 Validation Loss= 2292614692864.0
Epoch 2500 Training Loss= 1350008176640.0 Validation Loss= 2292681539584.0
Epoch 3000 Training Loss= 1350008176640.0 Validation Loss= 2292710899712.0
Epoch 3500 Training Loss= 1350008045568.0 Validation Loss= 2292719812608.0
Epoch 4000 Training Loss= 1350008045568.0 Validation Loss= 2292721909760.0
Epoch 4500 Training Loss= 1350008176640.0 Validation Loss= 2292722434048.0
Epoch 5000 Training Loss= 1350008176640.0 Validation Loss= 2292722434048.0
Training with learning rate = 0.01
Epoch 500 Training Loss= 1699869229056.0 Validation Loss= 2915957211136.0
Epoch 1000 Training Loss= 1556255735808.0 Validation Loss= 2649620742144.0
Epoch 1500 Training Loss= 1481826238464.0 Validation Loss= 2515905806336.0
Epoch 2000 Training Loss= 1436944433152.0 Validation Loss= 2440182366208.0
Epoch 2500 Training Loss= 1408152895488.0 Validation Loss= 2393896648704.0
Epoch 3000 Training Loss= 1389229244416.0 Validation Loss= 2364096643072.0
Epoch 3500 Training Loss= 1376647643136.0 Validation Loss= 2344159281152.0
Epoch 4000 Training Loss= 1368219844608.0 Validation Loss= 2330412449792.0
Epoch 4500 Training Loss= 1362538266624.0 Validation Loss= 2320701849600.0
Epoch 5000 Training Loss= 1358683963392.0 Validation Loss= 2313703129088.0
Training with learning rate = 0.001
Epoch 500 Training Loss= 3540332052480.0 Validation Loss= 5707507695616.0
Epoch 1000 Training Loss= 2057299951616.0 Validation Loss= 3644702851072.0
Epoch 1500 Training Loss= 1917634740224.0 Validation Loss= 3342630125568.0
Epoch 2000 Training Loss= 1870883323904.0 Validation Loss= 3233899085824.0
Epoch 2500 Training Loss= 1834175954944.0 Validation Loss= 3161707511808.0
Epoch 3000 Training Loss= 1801548726272.0 Validation Loss= 3101782704128.0
Epoch 3500 Training Loss= 1772170379264.0 Validation Loss= 3048592113664.0
Epoch 4000 Training Loss= 1745609162752.0 Validation Loss= 3000403755008.0
Epoch 4500 Training Loss= 1721512230912.0 Validation Loss= 2956429361152.0
Epoch 5000 Training Loss= 1699577331712.0 Validation Loss= 2916155129856.0
Training with learning rate = 0.0001
Epoch 500 Training Loss= 19724295471104.0 Validation Loss= 24090727415808.0
Epoch 1000 Training Loss= 15514222985216.0 Validation Loss= 19448236867584.0
Epoch 1500 Training Loss= 12304728981504.0 Validation Loss= 15875069444096.0
Epoch 2000 Training Loss= 9857700724736.0 Validation Loss= 13121063223296.0
Epoch 2500 Training Loss= 7991701536768.0 Validation Loss= 10995033440256.0
Epoch 3000 Training Loss= 6568476672000.0 Validation Loss= 9350820134912.0
Epoch 3500 Training Loss= 5482659971072.0 Validation Loss= 8076610502656.0
Epoch 4000 Training Loss= 4653971406848.0 Validation Loss= 7086844936192.0
Epoch 4500 Training Loss= 4021239414784.0 Validation Loss= 6316007358464.0
Epoch 5000 Training Loss= 3537835130880.0 Validation Loss= 5713884610560.0
```

Problem 3A

Source Code(3A):

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

```python
import torch
import torch.optim as optim
import time

file_path = "Housing.csv"
df = pd.read_csv(file_path)

inputs = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking']
X = df[inputs].values
Y = df['price'].values
m = len(Y)

X_0 = np.ones((m, 1))
X = np.hstack((X_0, X))
theta = np.zeros(X.shape[1])

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=42)
scaler = MinMaxScaler()
X_train[:, 1:] = scaler.fit_transform(X_train[:, 1:])
X_test[:, 1:] = scaler.transform(X_test[:, 1:])
# Training data
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
Y_train_tensor = torch.tensor(Y_train, dtype=torch.float32).view(-1,1)  #
shape (N,1)

# Validation data
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
Y_test_tensor = torch.tensor(Y_test, dtype=torch.float32).view(-1,1)
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))
        x = self.fc2(x)
        return x
model = NN()
loss_fn = torch.nn.MSELoss()
```

```
optimizer = optim.SGD(model.parameters(), lr=0.01)

# training 200 epochs
start = time.time()
for epoch in range(1, 201):
    y_pred = model(X_train_tensor)
    loss = loss_fn(y_pred, Y_train_tensor)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
end = time.time()

print("Training time =", end - start, "seconds")
print("Final Training Loss =", loss.item())

with torch.no_grad():
    y_val_pred = model(X_test_tensor)

SS_res = torch.sum((Y_test_tensor - y_val_pred)**2)
SS_tot = torch.sum((Y_test_tensor - torch.mean(Y_test_tensor))**2)
R2 = 1 - SS_res/SS_tot

print("Evaluation Accuracy (R^2) =", R2.item())
```

Results(3A):

Training time = 0.3331565856933594 seconds
Final Training Loss = 3985036279808.0
Evaluation Accuracy (R^2) = -0.30007660388946533

Problem 3B

Changed Code(3B):

```
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))
        x = self.fc4(x)
        return x
```

```
model = NN()
loss_fn = torch.nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=1e-9)
```

<u>Results(3B):</u>

Training time = 0.2768397331237793 seconds
Final Training Loss = 252347741634560.0
Evaluation Accuracy (R^2) = -4.960936069488525