```
#https://github.com/jhelto12/4105_Intro_to_ML/upload/main/Homework%205
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load dataset
housing_url = "https://raw.githubusercontent.com/HamedTabkhi/Intro-to-ML/refs/heads/main/Dataset/Housing.csv"
housing_df = pd.read_csv(housing_url)


# DATASET PROCESSING (Part 1)
#converts yes/no columns to binary
binary_columns = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'prefarea']
housing_df[binary_columns] = housing_df[binary_columns].replace({'yes': 1, 'no': 0})
housing_df.pop('furnishingstatus')

# Split features and target
housing_X = housing_df.drop('price', axis=1)
housing_Y = housing_df['price']
```

```
<ipython-input-36-e3d3e6d68aca>:4: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future versic
  housing_df[binary_columns] = housing_df[binary_columns].replace({'yes': 1, 'no': 0})
```

```
# SCALING THE DATASET
scaler = StandardScaler()
housing_X = scaler.fit_transform(housing_X)

print(housing_X[:5])
```

```
[[ 1.04672629  1.40341936  1.42181174  1.37821692  0.40562287 -0.46531479
  -0.73453933 -0.2192645   1.4726183   1.51769249  1.80494113]
 [ 1.75700953  1.40341936  5.40580863  2.53202371  0.40562287 -0.46531479
  -0.73453933 -0.2192645   1.4726183   2.67940935 -0.55403469]
 [ 2.21823241  0.04727831  1.42181174  0.22441013  0.40562287 -0.46531479
    1.3613975  -0.2192645  -0.67906259  1.51769249  1.80494113]
 [ 1.08362412  1.40341936  1.42181174  0.22441013  0.40562287 -0.46531479
    1.3613975  -0.2192645   1.4726183   2.67940935  1.80494113]
 [ 1.04672629  1.40341936 -0.57018671  0.22441013  0.40562287  2.14908276
    1.3613975  -0.2192645   1.4726183   1.51769249 -0.55403469]]
```

```
# DATASET PROCESSING (Part 2)
housing_X = torch.tensor(housing_X, dtype=torch.float32)
housing_Y = torch.tensor(housing_Y.values, dtype=torch.float32).view(-1, 1)  # Ensure housing_Y is shaped correctly (N, 1)

#splits the dataset
housing_train_X, housing_test_X, housing_train_Y, housing_test_Y = train_test_split(housing_X, housing_Y, test_size=0.2, random_state=42)

#gets the indices of the desired columns for problem 2
desired_columns = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking']
column_indices = [housing_df.columns.get_loc(col) for col in desired_columns]

# Select the desired columns from the NumPy arrays
small_housing_train_X = pd.DataFrame(housing_train_X[:, column_indices], columns=desired_columns)
small_housing_test_X = pd.DataFrame(housing_test_X[:, column_indices], columns=desired_columns)

#all columns for problem 3 - Assuming you want all 11 columns here
full_housing_train_X = pd.DataFrame(housing_train_X, columns=housing_df.columns[:-1]) # Exclude 'price' column
full_housing_test_X = pd.DataFrame(housing_test_X, columns=housing_df.columns[:-1]) # Exclude 'price' column


# Define the linear regression model using nn.Linear
small_input_dim = small_housing_train_X.shape[1]
small_model = nn.Linear(small_input_dim, 1)

# Define the loss function and optimizer
loss_function = nn.MSELoss()

epochs = 5000
```

```
# TRAINING LOOP WITH LEARNING RATE OF 0.1
learning_rate = 0.1
small_optimizer = optim.Adam(small_model.parameters(), lr=learning_rate)

# TRAINING LOOP
epochs = 5000

for epoch in range(epochs + 1):
    # Forward pass
    small_housing_Y_pred = small_model(torch.tensor(small_housing_train_X.values, dtype=torch.float32))

    # Compute loss
    small_loss = loss_function(small_housing_Y_pred, housing_train_Y)

    # Zero gradients, perform a backward pass, and update the weights
    small_optimizer.zero_grad()  # Zero the gradients
    small_loss.backward()        # Backpropagate the loss
    small_optimizer.step()       # Update the parameters

    # Optionally, print loss for monitoring
    if epoch % 500 == 0:
        print(f"Epoch {epoch} \tLoss: {small_loss.item()}")
```

```
Epoch 0         Loss: 25234788843520.0
Epoch 500       Loss: 25234025480192.0
Epoch 1000      Loss: 25233260019712.0
Epoch 1500      Loss: 25232498753536.0
Epoch 2000      Loss: 25231737487360.0
Epoch 2500      Loss: 25230972026880.0
Epoch 3000      Loss: 25230208663552.0
Epoch 3500      Loss: 25229445300224.0
Epoch 4000      Loss: 25228679839744.0
Epoch 4500      Loss: 25227916476416.0
Epoch 5000      Loss: 25227153113088.0
```

```
# TRAINING LOOP WITH LEARNING RATE OF 0.01
learning_rate = 0.01
small_optimizer = optim.Adam(small_model.parameters(), lr=learning_rate)

# TRAINING LOOP
epochs = 5000

for epoch in range(epochs + 1):
    # Forward pass
    small_housing_Y_pred = small_model(torch.tensor(small_housing_train_X.values, dtype=torch.float32))

    # Compute loss
    small_loss = loss_function(small_housing_Y_pred, housing_train_Y)

    # Zero gradients, perform a backward pass, and update the weights
    small_optimizer.zero_grad()  # Zero the gradients
    small_loss.backward()        # Backpropagate the loss
    small_optimizer.step()       # Update the parameters

    # Optionally, print loss for monitoring
    if epoch % 500 == 0:
        print(f"Epoch {epoch} \tLoss: {small_loss.item()}")
```

```
Epoch 0         Loss: 25227151015936.0
Epoch 500       Loss: 25227075518464.0
Epoch 1000      Loss: 25226997923840.0
Epoch 1500      Loss: 25226922426368.0
Epoch 2000      Loss: 25226844831744.0
Epoch 2500      Loss: 25226771431424.0
Epoch 3000      Loss: 25226691739648.0
Epoch 3500      Loss: 25226618339328.0
Epoch 4000      Loss: 25226542841856.0
Epoch 4500      Loss: 25226463150080.0
Epoch 5000      Loss: 25226387652608.0
```

```
# TRAINING LOOP WITH LEARNING RATE OF 0.001
learning_rate = 0.001
small_optimizer = optim.Adam(small_model.parameters(), lr=learning_rate)

# TRAINING LOOP
epochs = 5000
```

```
for epoch in range(epochs + 1):
    # Forward pass
    small_housing_Y_pred = small_model(torch.tensor(small_housing_train_X.values, dtype=torch.float32))

    # Compute loss
    small_loss = loss_function(small_housing_Y_pred, housing_train_Y)

    # Zero gradients, perform a backward pass, and update the weights
    small_optimizer.zero_grad()  # Zero the gradients
    small_loss.backward()        # Backpropagate the loss
    small_optimizer.step()       # Update the parameters

    # Optionally, print loss for monitoring
    if epoch % 500 == 0:
        print(f"Epoch {epoch} \tLoss: {small_loss.item()}")
```

```
Epoch 0         Loss: 25226387652608.0
Epoch 500       Loss: 25226377166848.0
Epoch 1000      Loss: 25226375069696.0
Epoch 1500      Loss: 25226364583936.0
Epoch 2000      Loss: 25226358292480.0
Epoch 2500      Loss: 25226349903872.0
Epoch 3000      Loss: 25226343612416.0
Epoch 3500      Loss: 25226333126656.0
Epoch 4000      Loss: 25226326835200.0
Epoch 4500      Loss: 25226320543744.0
Epoch 5000      Loss: 25226312155136.0
```

```
# TRAINING LOOP WITH LEARNING RATE OF 0.0001
learning_rate = 0.0001
small_optimizer = optim.Adam(small_model.parameters(), lr=learning_rate)

# TRAINING LOOP
epochs = 5000

for epoch in range(epochs + 1):
    # Forward pass
    small_housing_Y_pred = small_model(torch.tensor(small_housing_train_X.values, dtype=torch.float32))

    # Compute loss
    small_loss = loss_function(small_housing_Y_pred, housing_train_Y)

    # Zero gradients, perform a backward pass, and update the weights
    small_optimizer.zero_grad()  # Zero the gradients
    small_loss.backward()        # Backpropagate the loss
    small_optimizer.step()       # Update the parameters

    # Optionally, print loss for monitoring
    if epoch % 500 == 0:
        print(f"Epoch {epoch} \tLoss: {small_loss.item()}")
```

```
Epoch 0         Loss: 25226312155136.0
Epoch 500       Loss: 25226312155136.0
Epoch 1000      Loss: 25226307960832.0
Epoch 1500      Loss: 25226307960832.0
Epoch 2000      Loss: 25226307960832.0
Epoch 2500      Loss: 25226307960832.0
Epoch 3000      Loss: 25226305863680.0
Epoch 3500      Loss: 25226305863680.0
Epoch 4000      Loss: 25226305863680.0
Epoch 4500      Loss: 25226305863680.0
Epoch 5000      Loss: 25226303766528.0
```

I'll be completely honest, I have no idea why these losses are so high. At least these are consistently high, so the issue might be something else than the training loop. If I would have to guess, it has something to do with the data processing.

```
full_input_dim = full_housing_train_X.shape[1]
full_model = nn.Linear(full_input_dim, 1)


# TRAINING LOOP WITH LEARNING RATE OF 0.1
learning_rate = 0.1
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# TRAINING LOOP
epochs = 5000
```

```
for epoch in range(epochs + 1):
    # Forward pass: compute predicted y by passing x to the model
    housing_Y_pred = full_model(housing_train_X)

    # Compute loss
    loss = loss_function(housing_Y_pred, housing_train_Y)

    # Zero gradients, perform a backward pass, and update the weights
    optimizer.zero_grad()  # Zero the gradients
    loss.backward()        # Backpropagate the loss
    optimizer.step()       # Update the parameters

    # Optionally, print loss for monitoring
    if epoch % 500 == 0:
        print(f"Epoch {epoch} \tLoss: {loss.item()}")

# TESTING PHASE (Optional)
with torch.no_grad():  # Disable gradient calculation for testing
    housing_Y_test_pred = model(housing_test_X)
    test_loss = loss_function(housing_Y_test_pred, housing_test_Y)
    print(f"Test\t\tLoss: {test_loss.item()}")
```

```
Epoch 0         Loss: 25234788843520.0
Epoch 500       Loss: 25234788843520.0
Epoch 1000      Loss: 25234788843520.0
Epoch 1500      Loss: 25234788843520.0
Epoch 2000      Loss: 25234788843520.0
Epoch 2500      Loss: 25234788843520.0
Epoch 3000      Loss: 25234788843520.0
Epoch 3500      Loss: 25234788843520.0
Epoch 4000      Loss: 25234788843520.0
Epoch 4500      Loss: 25234788843520.0
Epoch 5000      Loss: 25234788843520.0
Test            Loss: 30067266682880.0
```

```
# TRAINING LOOP WITH LEARNING RATE OF 0.01
learning_rate = 0.01
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# TRAINING LOOP
epochs = 5000

for epoch in range(epochs + 1):
    # Forward pass: compute predicted y by passing x to the model
    housing_Y_pred = full_model(housing_train_X)

    # Compute loss
    loss = loss_function(housing_Y_pred, housing_train_Y)

    # Zero gradients, perform a backward pass, and update the weights
    optimizer.zero_grad()  # Zero the gradients
    loss.backward()        # Backpropagate the loss
    optimizer.step()       # Update the parameters

    # Optionally, print loss for monitoring
    if epoch % 500 == 0:
        print(f"Epoch {epoch} \tLoss: {loss.item()}")

# TESTING PHASE (Optional)
with torch.no_grad():  # Disable gradient calculation for testing
    housing_Y_test_pred = model(housing_test_X)
    test_loss = loss_function(housing_Y_test_pred, housing_test_Y)
    print(f"Test\t\tLoss: {test_loss.item()}")
```

```
Epoch 0         Loss: 25234788843520.0
Epoch 500       Loss: 25234788843520.0
Epoch 1000      Loss: 25234788843520.0
Epoch 1500      Loss: 25234788843520.0
Epoch 2000      Loss: 25234788843520.0
Epoch 2500      Loss: 25234788843520.0
Epoch 3000      Loss: 25234788843520.0
Epoch 3500      Loss: 25234788843520.0
Epoch 4000      Loss: 25234788843520.0
Epoch 4500      Loss: 25234788843520.0
Epoch 5000      Loss: 25234788843520.0
Test            Loss: 30067266682880.0
```

```
# TRAINING LOOP WITH LEARNING RATE OF 0.001
learning_rate = 0.001
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# TRAINING LOOP
epochs = 5000

for epoch in range(epochs + 1):
    # Forward pass: compute predicted y by passing x to the model
    housing_Y_pred = full_model(housing_train_X)

    # Compute loss
    loss = loss_function(housing_Y_pred, housing_train_Y)

    # Zero gradients, perform a backward pass, and update the weights
    optimizer.zero_grad()  # Zero the gradients
    loss.backward()        # Backpropagate the loss
    optimizer.step()       # Update the parameters

    # Optionally, print loss for monitoring
    if epoch % 500 == 0:
        print(f"Epoch {epoch} \tLoss: {loss.item()}")

# TESTING PHASE (Optional)
with torch.no_grad():  # Disable gradient calculation for testing
    housing_Y_test_pred = model(housing_test_X)
    test_loss = loss_function(housing_Y_test_pred, housing_test_Y)
    print(f"Test\t\tLoss: {test_loss.item()}")
```

```
Epoch 0         Loss: 25234788843520.0
Epoch 500       Loss: 25234788843520.0
Epoch 1000      Loss: 25234788843520.0
Epoch 1500      Loss: 25234788843520.0
Epoch 2000      Loss: 25234788843520.0
Epoch 2500      Loss: 25234788843520.0
Epoch 3000      Loss: 25234788843520.0
Epoch 3500      Loss: 25234788843520.0
Epoch 4000      Loss: 25234788843520.0
Epoch 4500      Loss: 25234788843520.0
Epoch 5000      Loss: 25234788843520.0
Test            Loss: 30067266682880.0
```

```
# TRAINING LOOP WITH LEARNING RATE OF 0.0001
learning_rate = 0.0001
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# TRAINING LOOP
epochs = 5000

for epoch in range(epochs + 1):
    # Forward pass: compute predicted y by passing x to the model
    housing_Y_pred = full_model(housing_train_X)

    # Compute loss
    loss = loss_function(housing_Y_pred, housing_train_Y)

    # Zero gradients, perform a backward pass, and update the weights
    optimizer.zero_grad()  # Zero the gradients
    loss.backward()        # Backpropagate the loss
    optimizer.step()       # Update the parameters

    # Optionally, print loss for monitoring
    if epoch % 500 == 0:
        print(f"Epoch {epoch} \tLoss: {loss.item()}")

# TESTING PHASE (Optional)
with torch.no_grad():  # Disable gradient calculation for testing
    housing_Y_test_pred = model(housing_test_X)
    test_loss = loss_function(housing_Y_test_pred, housing_test_Y)
    print(f"Test\t\tLoss: {test_loss.item()}")
```

```
Epoch 0         Loss: 25234788843520.0
Epoch 500       Loss: 25234788843520.0
Epoch 1000      Loss: 25234788843520.0
Epoch 1500      Loss: 25234788843520.0
Epoch 2000      Loss: 25234788843520.0
Epoch 2500      Loss: 25234788843520.0
Epoch 3000      Loss: 25234788843520.0
```

```
Epoch 3500      Loss: 25234788843520.0
Epoch 4000      Loss: 25234788843520.0
Epoch 4500      Loss: 25234788843520.0
Epoch 5000      Loss: 25234788843520.0
Test            Loss: 30067266682880.0
```