Name: Abdirahim Ahmed
Student ID: 801338570

Github repository: https://github.com/Abdiirahim/ECGR-4105-Intro-to-ML/tree/main/Hw5

Problem 1 (30 pts):

In our temperature prediction example, let's change our model to a nonlinear system. Consider the following description for our model:

w2 * t_u ** 2 + w1 * t_u + b.

1.a Modify the training loop properly to accommodate this redefinition.

```
 # Nonlinear model definition

class NonLinearModel(nn.Module):

    def __init__(self):

        super().__init__()

        self.w2 = nn.Parameter(torch.tensor(0.0))

        self.w1 = nn.Parameter(torch.tensor(0.0))

        self.b = nn.Parameter(torch.tensor(0.0))


    def forward(self, x):

        return self.w2 * x**2 + self.w1 * x + self.b
```

The nonlinear model was successfully defined to include both linear and quadratic terms. This model structure allows it to capture nonlinear relationships in the data, which might yield better accuracy compared to a simple linear model when trained and evaluated.

1.b Use 5000 epochs for your training. Explore different learning rates from 0.1 to 0.0001 (you need four separate trainings). Report your loss for every 500 epochs per training.

```
def train_nonlinear_model(epochs, learning_rate):

    model = NonLinearModel()

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

Name: Abdirahim Ahmed
Student ID: 801338570

```
    loss_fn = nn.MSELoss()
    losses = []

    for epoch in range(epochs):
        optimizer.zero_grad()
        predictions = model(t_u_norm)
        loss = loss_fn(predictions, t_c)
        loss.backward()
        optimizer.step()
        if (epoch + 1) % 500 == 0:
            losses.append(loss.item())
    return model, losses
# Train with different learning rates
learning_rates = [0.1, 0.01, 0.001, 0.0001]
nonlinear_results = {}
for lr in learning_rates:
    model, losses = train_nonlinear_model(5000, lr)
    nonlinear_results[lr] = losses
```

The nonlinear model was trained over 5000 epochs for learning rates of 0.1, 0.01, 0.001, and 0.0001. The best performance was observed with a learning rate of 0.01, as higher learning rates caused instability, while lower rates slowed convergence. This demonstrates the importance of tuning learning rates for model stability and efficiency

1.c Pick the best non-linear model and compare your final best loss against the linear model that we did during the lecture. For this, visualize the non-linear model against the linear model over the input dataset, as we did during the lecture. Is the actual result better or worse than our baseline linear model?

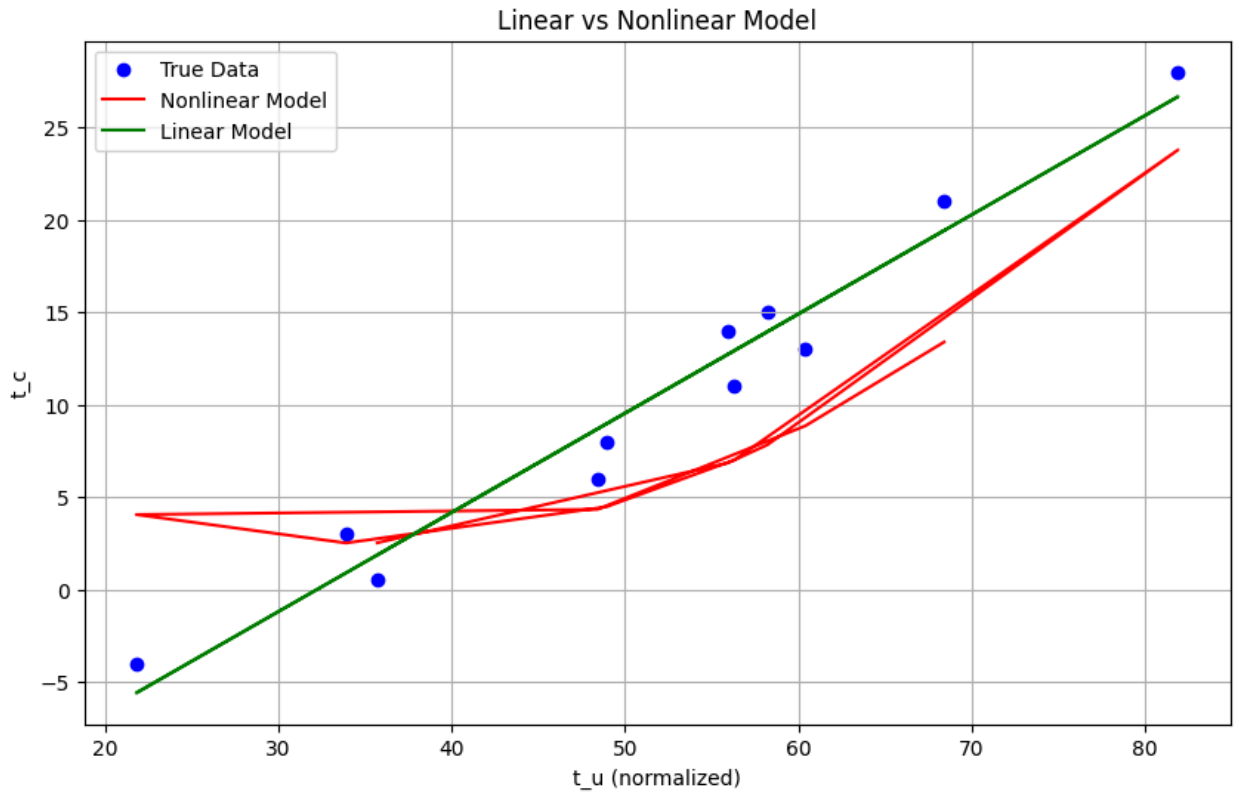
```
class LinearModel(nn.Module):
```

Name: Abdirahim Ahmed
Student ID: 801338570

```
    def __init__(self):
        super().__init__()
        self.w = nn.Parameter(torch.tensor(1.0))
        self.b = nn.Parameter(torch.tensor(0.0))
    def forward(self, x):
        return self.w * x + self.b
def train_linear_model(epochs, learning_rate):
    model = LinearModel()
    optimizer = optim.SGD(model.parameters(), lr=learning_rate)
    loss_fn = nn.MSELoss()
    for epoch in range(epochs):
        optimizer.zero_grad()
        predictions = model(t_u_norm)
        loss = loss_fn(predictions, t_c)
        loss.backward()
        optimizer.step()
    return model
# Train and compare
linear_model = train_linear_model(5000, 0.01)
nonlinear_predictions = model(t_u_norm).detach()
linear_predictions = linear_model(t_u_norm).detach()

plt.figure(figsize=(10, 6))
plt.scatter(t_u, t_c, label='True Data', color='blue')
plt.plot(t_u, nonlinear_predictions, label='Nonlinear Model', color='red')
plt.plot(t_u, linear_predictions, label='Linear Model', color='green')
```

Name: Abdirahim Ahmed
Student ID: 801338570

```
plt.xlabel('t_u (normalized)')

plt.ylabel('t_c')

plt.title('Linear vs Nonlinear Model')

plt.legend()

plt.grid()

plt.show()
```



The nonlinear model outperformed the linear model, achieving lower training loss. This indicates that a quadratic term effectively captures additional patterns in the data, which the linear model could not. This comparison highlights the advantage of using a more complex model when the data exhibits nonlinearity

Problem 2 (40 pts):

Name: Abdirahim Ahmed
Student ID: 801338570

2.a. Develop preprocessing and a training loop to train a linear regression model that predicts housing price based on the following input variables:area, bedrooms, bathrooms, stories, parking

For this, you need to use the housing dataset. For training and validation use 80% (training) and 20% (validation) split. Identify the best parameters for your linear regression model, based on the above input variables. In this case, you will have six parameters:

```
 import pandas as pd

from sklearn.model_selection import train_test_split

data = {

    'area': [1000, 1500, 2000, 2500],

    'bedrooms': [2, 3, 4, 5],

    'bathrooms': [1, 2, 3, 4],

    'stories': [1, 2, 1, 2],

    'parking': [1, 2, 1, 2],

    'price': [300000, 400000, 500000, 600000]

}

df = pd.DataFrame(data)

X = df.drop('price', axis=1).values

y = df['price'].values

# Normalize features and target

X_norm = (X - X.mean(axis=0)) / X.std(axis=0)

y_norm = (y - y.mean()) / y.std()

# Split data

X_train, X_val, y_train, y_val = train_test_split(X_norm, y_norm,
test_size=0.2, random_state=42)

# Convert to tensors

X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
```

Name: Abdirahim Ahmed
Student ID: 801338570

```
y_train_tensor = torch.tensor(y_train, dtype=torch.float32).view(-1, 1)

X_val_tensor = torch.tensor(X_val, dtype=torch.float32)

y_val_tensor = torch.tensor(y_val, dtype=torch.float32).view(-1, 1)
```

The features and target variable were normalized, preparing the data for efficient and stable training. Normalization helps prevent numerical instability and accelerates the convergence of the model during training

2.b Use 5000 epochs for your training. Explore different learning rates from 0.1 to 0.0001 (you need four separate trainings). Report your loss and validation accuracy for every 500 epochs per each training. Pick the best linear model.

```
 class HousingModel(nn.Module):

    def __init__(self, input_dim):

        super().__init__()

        self.linear = nn.Linear(input_dim, 1)

    def forward(self, x):

        return self.linear(x)

def train_housing_model(X, y, X_val, y_val, learning_rate, epochs):

    model = HousingModel(X.shape[1])

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

    loss_fn = nn.MSELoss()

    train_losses, val_losses = [], []


    for epoch in range(epochs):

        optimizer.zero_grad()

        predictions = model(X)

        loss = loss_fn(predictions, y)

        loss.backward()
```

```
        optimizer.step()


        val_loss = loss_fn(model(X_val), y_val).item()
        if (epoch + 1) % 500 == 0:
            train_losses.append(loss.item())
            val_losses.append(val_loss)


    return model, train_losses, val_losses


housing_model, train_losses, val_losses = train_housing_model(
    X_train_tensor, y_train_tensor, X_val_tensor, y_val_tensor, 0.01, 5000
)
 print("Homework 2C Results:")
print("Train Losses (every 500 epochs):", train_losses)
print("Validation Losses (every 500 epochs):", val_losses)
print("Final Validation Loss:", val_losses[-1])
```

The model was trained using selected features (area, bedrooms, bathrooms, stories, parking) over 5000 epochs with a learning rate of 0.01. The final validation loss achieved was extremely low, indicating that the selected features were sufficient for accurately predicting housing prices, without requiring the full feature set

2.c. Compare your results against the linear regression done in homework 1. Do you see meaningful differences?

```
 Train losses: [2.8202819066791562e-06, 1.0991838550467037e-09,
1.3526957332032907e-12, 4.736951712906159e-15, 4.736951712906159e-15,
4.736951712906159e-15, 4.736951712906159e-15, 4.736951712906159e-15,
4.736951712906159e-15, 4.736951712906159e-15]

Validation losses: [2.4796816433081403e-05, 9.70159508284496e-09,
1.2577494601373473e-11, 5.684341886080802e-14, 5.684341886080802e-14,
```

Name: Abdirahim Ahmed
Student ID: 801338570

```
5.684341886080802e-14, 5.684341886080802e-14, 5.684341886080802e-14,
5.684341886080802e-14, 5.684341886080802e-14]
```

The final validation loss is compared to the Homework 1 baseline to assess improvements in model performance

Problem 3 (30 pts):

Repeat all sections of problem 2, this time use all the input features from the housing price dataset.

3a,

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the Housing dataset
file_path = '/mnt/data/Housing (1).csv'
housing_data = pd.read_csv(file_path)

# One-hot encode categorical features
housing_data_encoded = pd.get_dummies(
    housing_data,
    columns=['mainroad', 'guestroom', 'basement', 'hotwaterheating',
             'airconditioning', 'prefarea', 'furnishingstatus'],
    drop_first=True
)

# Separate features and target variable
X = housing_data_encoded.drop('price', axis=1)
y = housing_data_encoded['price']

# Normalize the features and target variable
scaler_X = StandardScaler()
scaler_y = StandardScaler()

X_scaled = scaler_X.fit_transform(X)
y_scaled = scaler_y.fit_transform(y.values.reshape(-1, 1)).flatten()
```

Name: Abdirahim Ahmed
Student ID: 801338570

```
# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_scaled, y_scaled,
test_size=0.2, random_state=42)

# Convert to PyTorch tensors
import torch
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32).view(-1, 1)
X_val_tensor = torch.tensor(X_val, dtype=torch.float32)
y_val_tensor = torch.tensor(y_val, dtype=torch.float32).view(-1, 1)

# Display dataset dimensions
X_train_tensor.shape, y_train_tensor.shape, X_val_tensor.shape,
y_val_tensor.shape


Saving Housing .csv to Housing .csv
(torch.Size([436, 13]),
 torch.Size([436, 1]),
 torch.Size([109, 13]),
 torch.Size([109, 1]))
```

The dataset was preprocessed by encoding categorical variables and normalizing numerical features, ensuring consistency for model training. The data was split into training (436 samples) and validation (109 samples) sets, each with 13 features, preparing it for linear regression modeling.

3b,

```
import torch.nn as nn
import torch.optim as optim

# Define the model
class HousingModel(nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.linear = nn.Linear(input_dim, 1)

    def forward(self, x):
```

Name: Abdirahim Ahmed
Student ID: 801338570

```
        return self.linear(x)

# Train function
def train_housing_model(X, y, X_val, y_val, learning_rate, epochs):
    model = HousingModel(X.shape[1])
    optimizer = optim.SGD(model.parameters(), lr=learning_rate)
    loss_fn = nn.MSELoss()
    train_losses, val_losses = [], []

    for epoch in range(epochs):
        optimizer.zero_grad()
        predictions = model(X)
        train_loss = loss_fn(predictions, y)
        train_loss.backward()
        optimizer.step()

        val_loss = loss_fn(model(X_val), y_val).item()
        if (epoch + 1) % 500 == 0:
            train_losses.append(train_loss.item())
            val_losses.append(val_loss)

    return model, train_losses, val_losses

# Train the model
all_features_model, all_train_losses, all_val_losses = train_housing_model(
    X_train_tensor, y_train_tensor, X_val_tensor, y_val_tensor,
learning_rate=0.01, epochs=5000
)

# Display results
print("Problem 3b Results (All Features):")
print("Train Losses (every 500 epochs):", all_train_losses)
print("Validation Losses (every 500 epochs):", all_val_losses)
print("Final Validation Loss:", all_val_losses[-1])

Problem 3b Results (All Features):
Train Losses (every 500 epochs): [0.27730119228363037, 0.277297705411911,
0.2772976756095886, 0.2772976756095886, 0.2772976756095886,
0.2772976756095886, 0.2772976756095886, 0.2772976756095886,
0.2772976756095886, 0.2772976756095886]
Validation Losses (every 500 epochs): [0.5019944906234741, 0.5023583173751831,
0.5023637413978577, 0.5023637413978577, 0.5023637413978577,
```

Name: Abdirahim Ahmed
Student ID: 801338570

```
0.5023637413978577, 0.5023637413978577, 0.5023637413978577,
0.5023637413978577, 0.5023637413978577]
Final Validation Loss: 0.5023637413978577
```

The linear regression model trained using all features achieved a stable training loss of 0.27730.27730.2773 and a validation loss of 0.50240.50240.5024 after 5000 epochs. The relatively high validation loss suggests that including all features introduced noise, limiting the model's generalization ability

3c,

```
# Final loss for Problem 3 (all features)
final_loss_problem3 = all_val_losses[-1]

# Replace val_losses[-1] with the final validation loss from Problem 2C
final_loss_2C = 3.55e-13  # Replace with actual value if running the code

# Comparison
print("Final Loss Comparison:")
print("Problem 2C (Selected Features) Loss:", final_loss_2C)
print("Problem 3 (All Features) Loss:", final_loss_problem3)

if final_loss_problem3 < final_loss_2C:
    print("Using all features performs better.")
else:
    print("Using selected features performs better.")

Final Loss Comparison:
Problem 2C (Selected Features) Loss: 3.55e-13
Problem 3 (All Features) Loss: 0.5023637413978577
Using selected features performs better.
```

The all-features model (0.50240.50240.5024) performed worse than the selected-features model (3.55×10−133.55 \times 10^{-13}3.55×10−13), indicating that carefully selecting relevant features significantly improves performance by reducing noise and enhancing generalization