Learning Problem Description

The goal of this project is to understand the relationship between housing features and housing prices in a California district through the training and novel application of a neural network.

The function of the neural network involves taking data pertaining to the houses in a district (coordinates, population, income, etc), and using that data to predict the median house value of the area.

The resulting model has possible applications in the real estate industry, being used to predict median house prices in an area.

In [ ]:
```python
import numpy as np
import pandas as pd
import torch
from torch import (nn, optim)
from torch.utils.data import (TensorDataset, DataLoader, random_split)

import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```python
df = pd.read_csv('sample_data/california_housing_test.csv')
```

In [ ]:
```python
len(df)
```

Out[ ]:
```
3000
```

In [ ]:
```python
df = pd.read_csv('sample_data/california_housing_train.csv')

means, maxs, mins = dict(), dict(), dict()
for col in df:
  vals = df[col].values #gets values for current column
  #saving for later reversal
  means[col] = vals.mean()
  maxs[col] = vals.max()
  mins[col] = vals.min()

  norms = (vals - vals.mean()) / (vals.max() - vals.min())
  df[col] = norms

X = df.drop('median_house_value', axis=1).values
y = df['median_house_value'].values

X = torch.tensor(X, dtype=torch.float32)
y = torch.tensor(y.reshape(-1, 1), dtype=torch.float32)


dataset = TensorDataset(X, y)
```

In [ ]:
```python
df
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	0.523118	-0.152521	-0.266458	0.078248	0.115392	-0.011620	-0.004805	-0.164824	-0.289485
1	0.507182	-0.130205	-0.188027	0.131971	0.211296	-0.008424	-0.006285	-0.142314	-0.262269
2	0.498218	-0.205656	-0.227242	-0.050709	-0.056706	-0.030734	-0.063184	-0.153976	-0.250722
3	0.497222	-0.210970	-0.286066	-0.030122	-0.031411	-0.025633	-0.045259	-0.047715	-0.276083
4	0.497222	-0.218409	-0.168419	-0.031361	-0.033118	-0.022578	-0.039339	-0.135072	-0.292372
...	...	...	...	...	...	...	...	...	...
16995	-0.467918	0.526544	0.459032	-0.011247	-0.022565	-0.014647	-0.021743	-0.105273	-0.197733
16996	-0.468914	0.538233	0.145307	-0.007768	-0.001771	-0.006603	-0.005957	-0.094183	-0.264537
16997	-0.471902	0.660444	-0.227242	0.000879	-0.001305	-0.005201	-0.007437	-0.058777	-0.213815
16998	-0.471902	0.656193	-0.188027	0.000747	0.001954	-0.003688	-0.003819	-0.131300	-0.250516
16999	-0.476882	0.522293	0.459032	-0.021713	-0.037153	-0.017477	-0.038024	-0.059922	-0.232372

17000 rows × 9 columns

In [ ]:
```python
df = pd.read_csv('sample_data/california_housing_train.csv')
df
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936	66900.0
1	-114.47	34.40	19.0	7650.0	1901.0	1129.0	463.0	1.8200	80100.0
2	-114.56	33.69	17.0	720.0	174.0	333.0	117.0	1.6509	85700.0
3	-114.57	33.64	14.0	1501.0	337.0	515.0	226.0	3.1917	73400.0
4	-114.57	33.57	20.0	1454.0	326.0	624.0	262.0	1.9250	65500.0
...	...	...	...	...	...	...	...	...	...
16995	-124.26	40.58	52.0	2217.0	394.0	907.0	369.0	2.3571	111400.0
16996	-124.27	40.69	36.0	2349.0	528.0	1194.0	465.0	2.5179	79000.0
16997	-124.30	41.84	17.0	2677.0	531.0	1244.0	456.0	3.0313	103600.0
16998	-124.30	41.80	19.0	2672.0	552.0	1298.0	478.0	1.9797	85800.0
16999	-124.35	40.54	52.0	1820.0	300.0	806.0	270.0	3.0147	94600.0

17000 rows × 9 columns

In [ ]:
```python
#test the dataloader
dataloader = DataLoader(dataset, batch_size=128, shuffle=True)
xs, targets = next(iter(dataloader))
print(xs.shape)
print(targets.shape)
```

```
torch.Size([128, 8])
torch.Size([128, 1])
```

In [ ]:
```python
len(df.columns)
```

Out[ ]:
```
9
```

In [ ]:
```python
print(torch.cuda.is_available())
# print(torch.cuda.get_device_name(0))
device = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
device
```

```
True
```

Out[ ]:
```
device(type='cuda')
```

Creating the Linear regression model

In [ ]:
```python
class LinearRegression(nn.Module):
    def __init__(self,num_features):
        super().__init__()
        self.layer1 = torch.nn.Linear(num_features,64).to(device)
        self.layer2 = torch.nn.Linear(64,64).to(device)
        self.layer3 = torch.nn.Linear(64,1).to(device)

        self.relu = nn.ReLU().to(device)
        self.sigmoid = nn.Sigmoid().to(device)

    def forward(self,x):
        y_pred = self.layer1(x)
        y_pred = self.relu(y_pred)
        y_pred = self.layer2(y_pred)
        y_pred = self.relu(y_pred)
        y_pred = self.sigmoid(self.layer3(y_pred))
        return y_pred
```

In [ ]:
```python
!pip install torchmetrics
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting torchmetrics
  Downloading torchmetrics-0.11.4-py3-none-any.whl (519 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 519.2/519.2 kB 28.2 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17.2 in /usr/local/lib/python3.9/dist-packages (from torchmetrics) (1.22.4)
Requirement already satisfied: torch>=1.8.1 in /usr/local/lib/python3.9/dist-packages (from torchmetrics) (2.0.0+cu11
8)
Requirement already satisfied: packaging in /usr/local/lib/python3.9/dist-packages (from torchmetrics) (23.0)
Requirement already satisfied: triton==2.0.0 in /usr/local/lib/python3.9/dist-packages (from torch>=1.8.1->torchmetri
cs) (2.0.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.9/dist-packages (from torch>=1.8.1->torchmetrics) (3.
1.2)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.9/dist-packages (from torch>=1.8.1->torchm
etrics) (4.5.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.9/dist-packages (from torch>=1.8.1->torchmetrics)
(3.11.0)
Requirement already satisfied: networkx in /usr/local/lib/python3.9/dist-packages (from torch>=1.8.1->torchmetrics)
(3.1)
Requirement already satisfied: sympy in /usr/local/lib/python3.9/dist-packages (from torch>=1.8.1->torchmetrics) (1.1
1.1)
Requirement already satisfied: lit in /usr/local/lib/python3.9/dist-packages (from triton==2.0.0->torch>=1.8.1->torch
metrics) (16.0.1)
Requirement already satisfied: cmake in /usr/local/lib/python3.9/dist-packages (from triton==2.0.0->torch>=1.8.1->tor
chmetrics) (3.25.2)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.9/dist-packages (from jinja2->torch>=1.8.1->
torchmetrics) (2.1.2)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.9/dist-packages (from sympy->torch>=1.8.1->torc
hmetrics) (1.3.0)
Installing collected packages: torchmetrics
Successfully installed torchmetrics-0.11.4
```

In [ ]:
```python
from torchmetrics import Accuracy
import itertools
import time
```

In [ ]:
```python
num_features = len(df.columns)-1
```

Train the model

In [ ]:
```python
def train(model,
          train_dataset: TensorDataset,
          epochs: int,
          max_batches=None):

    loss_fn = torch.nn.MSELoss(reduction = 'sum')
    optimizer = torch.optim.Adam(model.parameters())

    dataloader = DataLoader(train_dataset, batch_size=128, shuffle=True, pin_memory=True)

    start0 = time.time()
    start = time.time()

    for epoch in range(epochs):
        #train logic
        l_list = []
        acc_list = []
        count = 0
        while (count < max_batches):
            count += 1
            x, target = next(iter(dataloader))
            x, target = x.to(device), target.to(device)

            y_out = model(x).to(device)
            # print(y_out)
            loss = loss_fn(y_out, target)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            with torch.no_grad():
                l_list.append(loss.item())

        train_loss = np.mean(l_list)


        if epoch % 100 == 0:
          duration = time.time() - start
          start = time.time()
          print("[%d (%.2fs)]: train_loss=%.2f" % (
              epoch, duration, train_loss))

    duration0 = time.time() - start0
    print("== Total training time %.2f seconds ==" % duration0)
```

In [ ]:
```python
model = LinearRegression(8).to(device)
train(model, dataset, epochs = 2500, max_batches = 500)
```

```
[0 (3.06s)]: train_loss=7.24
[100 (223.62s)]: train_loss=3.38
[200 (224.66s)]: train_loss=3.36
[300 (225.91s)]: train_loss=3.24
[400 (224.52s)]: train_loss=3.21
[500 (221.51s)]: train_loss=3.17
[600 (216.88s)]: train_loss=3.16
[700 (221.46s)]: train_loss=3.12
[800 (221.90s)]: train_loss=3.09
[900 (228.17s)]: train_loss=3.07
[1000 (245.56s)]: train_loss=3.07
[1100 (234.84s)]: train_loss=3.07
[1200 (246.96s)]: train_loss=3.04
[1300 (236.76s)]: train_loss=3.04
[1400 (226.16s)]: train_loss=3.04
[1500 (227.76s)]: train_loss=3.04
[1600 (236.58s)]: train_loss=3.01
[1700 (231.38s)]: train_loss=3.05
[1800 (231.92s)]: train_loss=3.05
[1900 (231.74s)]: train_loss=3.05
[2000 (232.15s)]: train_loss=3.04
[2100 (232.64s)]: train_loss=3.03
[2200 (227.88s)]: train_loss=3.03
[2300 (229.87s)]: train_loss=2.99
[2400 (234.53s)]: train_loss=3.01
== Total training time 5749.94 seconds ==
```

In [ ]:
```python
#save the model
torch.save(model, '/content/mymodel.pt')
```

Test the model

In [ ]:
```python
def test_saved_model(model = None):
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    # test_dataset = torch.load('./sample_data/california_housing_test.csv')
    df = pd.read_csv('sample_data/california_housing_test.csv')
    std = np.std(df['median_house_value'])
    mean = np.mean(df['median_house_value'])

    means, maxs, mins = dict(), dict(), dict()
    for col in df:
        vals = df[col].values #gets values for current column
        #saving for later reversal
        means[col] = vals.mean()
        maxs[col] = vals.max()
        mins[col] = vals.min()

        norms = (vals - vals.mean()) / (vals.max() - vals.min())
        df[col] = norms

    X = df.drop('median_house_value', axis=1).values
    y = df['median_house_value'].values

    X = torch.tensor(X, dtype=torch.float32)
    y = torch.tensor(y.reshape(-1, 1), dtype=torch.float32)

    test_dataset = TensorDataset(X, y)
    dataloader = DataLoader(test_dataset, batch_size=128, shuffle=True)

    prediction = pd.DataFrame(columns=['Actual Median House Price',
                                       'Predicted Median House Price',
                                       'Difference',
                                       'Percentage Error'])

    i = 0
    with torch.no_grad():
        for xs, targets in dataloader:
            xs, targets = xs.to(device), targets.to(device)
            ys = model(xs)

            #Convert normalized values back to original
            actual = round((targets[i].item()*std) + mean, 1)
            predicted = round((ys[i].item()*std) + mean , 1)

            prediction = prediction.append({'Actual Median House Price' : actual,
                                            'Predicted Median House Price' : predicted,
                                            'Difference' : actual - predicted,
                                            'Percentage Error': np.abs((actual - predicted)/actual)},
                                            ignore_index=True)
            i += 1
        print('mean error accuracy:',np.mean(prediction['Percentage Error']))
    return prediction
```

In [ ]:
```python
model = LinearRegression(8)
model = torch.load('/content/mymodel.pt')
# model.eval()
test_saved_model(model.to(device))
```

```
mean error accuracy: 0.20079277147555016
```

Out[ ]:

	Actual Median House Price	Predicted Median House Price	Difference	Percentage Error
0	184210.0	205846.3	-21636.3	0.117455
1	180917.7	317644.6	-136726.9	0.755741
2	222226.0	229423.1	-7197.1	0.032386
3	198729.5	313148.2	-114418.7	0.575751
4	178146.4	205846.3	-27699.9	0.155490
5	185323.2	205846.3	-20523.1	0.110742
6	207185.4	205846.8	1338.6	0.006461
7	220970.7	318219.9	-97249.2	0.440100
8	211354.1	205866.5	5487.6	0.025964
9	210714.6	205846.3	4868.3	0.023104
10	212349.0	308062.3	-95713.3	0.450736
11	211756.8	317003.9	-105247.1	0.497019
12	230350.3	223459.0	6891.3	0.029917
13	189397.2	205846.3	-16449.1	0.086850
14	236935.0	227945.0	8990.0	0.037943
15	212041.0	269430.9	-57389.9	0.270655
16	238143.0	251201.3	-13058.3	0.054834
17	219431.1	286862.1	-67431.0	0.307299
18	214670.2	318537.1	-103866.9	0.483844
19	210264.6	205846.3	4418.3	0.021013
20	228763.4	239233.6	-10470.2	0.045769
21	194537.1	205846.3	-11309.2	0.058134
22	180609.7	205846.3	-25236.6	0.139730
23	226726.4	205846.7	20879.7	0.092092

Demonstrate the model

In [ ]:
```python
def demonstrate_model(model, data):
  #For demonstration purposes, training data will be
  #used to preprocess new data

    #Preprocess
    df = pd.read_csv('sample_data/california_housing_train.csv')
    std = np.std(df['median_house_value'])
    mean = np.mean(df['median_house_value'])

    means, maxs, mins = dict(), dict(), dict()
    for col in df.drop('median_house_value', axis=1):
        vals = df[col].values #gets values for current column

        means[col] = vals.mean()
        maxs[col] = vals.max()
        mins[col] = vals.min()

        norms = (data[col] - vals.mean()) / (vals.max() - vals.min())
        data[col] = norms

    data = torch.tensor(data.values, dtype=torch.float32)

    output = 0.0
    with torch.no_grad():
      ys = model(data)
      output = round((ys.item()*std) + mean , 1)

    return output
```

In a practical application, such as a real estate website, the pre-trained model could be loaded, and the user would enter data on the features of their district. The data would then be preprocessed, and given to the model, which would then output a prediction of the median house price of the area.

For the purposes of this demonstration, data will be predefined in the cell below.

In [ ]:
```python
data = pd.DataFrame(
    {'longitude' : -120,
    'latitude' : 30,
    'housing_median_age' : 30.0,
    'total_rooms' : 2000.0,
    'total_bedrooms' : 400.0,
    'population' : 1000.0,
    'households' : 400.0,
    'median_income' : 2.0000}, index=[0])

model = LinearRegression(8)
model = torch.load('/content/mymodel.pt')

output = demonstrate_model(model, data)
print('Predicted median house price: ', output)
```

```
Predicted median house price:  236335.5
```

In [ ]:
```python
df = pd.read_csv('sample_data/california_housing_train.csv')

data = pd.DataFrame(
    {'longitude' : np.random.uniform(df['longitude'].min(), df['longitude'].max()),
    'latitude' : np.random.uniform(df['latitude'].min(), df['latitude'].max()),
    'housing_median_age' : np.random.uniform(df['housing_median_age'].min(), df['housing_median_age'].max()),
    'total_rooms' : np.random.uniform(df['total_rooms'].min(), df['total_rooms'].max()),
    'total_bedrooms' : np.random.uniform(df['total_bedrooms'].min(), df['total_bedrooms'].max()),
    'population' : np.random.uniform(df['population'].min(), df['population'].max()),
    'households' : np.random.uniform(df['households'].min(), df['households'].max()),
    'median_income' : np.random.uniform(df['median_income'].min(), df['median_income'].max())}, index=[0])

model = LinearRegression(8)
model = torch.load('/content/mymodel.pt')

output = demonstrate_model(model, data)
print('Predicted median house price: ', output)
```

```
Predicted median house price:  322062.9
```