In [1]:
```
from matplotlib import pyplot as plt
import pandas as pd
import numpy as np
import torch
import time
import torch.optim as optim
from sklearn.preprocessing import MinMaxScaler, StandardScaler
```

In [2]:
```
housing = pd.read_csv('Housing.csv')
housing
```

Out[2]:

| | price | area | bedrooms | bathrooms | stories | mainroad | guestroom | basement | hotwaterheat |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 13300000 | 7420 | 4 | 2 | 3 | yes | no | no | |
| **1** | 12250000 | 8960 | 4 | 4 | 4 | yes | no | no | |
| **2** | 12250000 | 9960 | 3 | 2 | 2 | yes | no | yes | |
| **3** | 12215000 | 7500 | 4 | 2 | 2 | yes | no | yes | |
| **4** | 11410000 | 7420 | 4 | 1 | 2 | yes | yes | yes | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **540** | 1820000 | 3000 | 2 | 1 | 1 | yes | no | yes | |
| **541** | 1767150 | 2400 | 3 | 1 | 1 | no | no | no | |
| **542** | 1750000 | 3620 | 2 | 1 | 1 | yes | no | no | |
| **543** | 1750000 | 2910 | 3 | 1 | 1 | no | no | no | |
| **544** | 1750000 | 3850 | 3 | 1 | 2 | yes | no | no | |

545 rows × 13 columns

In [3]:
```
num_vars = ['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'parking']
df_new = housing[num_vars]
df_new
```

Out[3]:

| | price | area | bedrooms | bathrooms | stories | parking |
|---|---|---|---|---|---|---|
| **0** | 13300000 | 7420 | 4 | 2 | 3 | 2 |
| **1** | 12250000 | 8960 | 4 | 4 | 4 | 3 |
| **2** | 12250000 | 9960 | 3 | 2 | 2 | 2 |
| **3** | 12215000 | 7500 | 4 | 2 | 2 | 3 |
| **4** | 11410000 | 7420 | 4 | 1 | 2 | 2 |
| **...** | ... | ... | ... | ... | ... | ... |
| **540** | 1820000 | 3000 | 2 | 1 | 1 | 2 |
| **541** | 1767150 | 2400 | 3 | 1 | 1 | 0 |
| **542** | 1750000 | 3620 | 2 | 1 | 1 | 0 |
| **543** | 1750000 | 2910 | 3 | 1 | 1 | 0 |
| **544** | 1750000 | 3850 | 3 | 1 | 2 | 0 |

545 rows × 6 columns

In [4]:
```
df_new.shape
```

Out[4]:
```
(545, 6)
```

In [5]:
```
scaler = MinMaxScaler()
df_new[num_vars] = scaler.fit_transform(df_new[num_vars])
df_new.head()
```

```
C:\Users\shenc\AppData\Local\Temp\ipykernel_46180\2943118898.py:2: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user
_guide/indexing.html#returning-a-view-versus-a-copy
  df_new[num_vars] = scaler.fit_transform(df_new[num_vars])
```

Out[5]:

| | price | area | bedrooms | bathrooms | stories | parking |
|---|---|---|---|---|---|---|
| **0** | 1.000000 | 0.396564 | 0.6 | 0.333333 | 0.666667 | 0.666667 |
| **1** | 0.909091 | 0.502405 | 0.6 | 1.000000 | 1.000000 | 1.000000 |
| **2** | 0.909091 | 0.571134 | 0.4 | 0.333333 | 0.333333 | 0.666667 |
| **3** | 0.906061 | 0.402062 | 0.6 | 0.333333 | 0.333333 | 1.000000 |
| **4** | 0.836364 | 0.396564 | 0.6 | 0.000000 | 0.333333 | 0.666667 |

In [6]:
```
X = df_new.iloc[:, 1:6].values
Y = df_new.iloc[:, 0].values
```

In [7]:
```
X = torch.tensor(X)
Y = torch.tensor(Y)
```

In [8]:
```
n_samples = X.shape[0]
n_val = int(0.2 * n_samples)
shuffled_indices = torch.randperm(n_samples)

train_indices = shuffled_indices[:-n_val]
val_indices = shuffled_indices[-n_val:]
```

In [9]:
```
train_X = X[train_indices]
train_Y = Y[train_indices]
train_Y = train_Y.unsqueeze(1)

val_X = X[val_indices]
val_Y = Y[val_indices]
val_Y = val_Y.unsqueeze(1)
```

In [10]:
```
import torch
import torch.nn as nn
import torch.optim as optim

model = nn.Sequential(
            nn.Linear(5,8),
            nn.ReLU(),
            nn.Linear(8,1),
            nn.LogSoftmax(dim=1))
model = model.float()
learning_rate = 0.1

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

out = model(train_X.float())


loss_fn = nn.MSELoss()

n_epochs = 200
correct = 0
total = 0

for epoch in range(n_epochs+1):
        training_start_time = time.time()
        out = model(train_X.float())
        loss = loss_fn(out,train_Y.float())

        out_v= model(val_X.float()) # <1>
        val_loss = loss_fn(out_v, val_Y.float())

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if epoch==0 or epoch % 20 ==0:
            print("Epoch: %d, Training Loss: %f, Validation Loss: %f" % (epoch, float(l
        _, predicted = torch.max(out, dim=1)
        total += val_Y.shape[0]
        correct += int((predicted == train_Y).sum())

print("Accuracy: %f" % (correct / total))
print('Time {:.2f}s'.format(time.time() - training_start_time))
```

```
Epoch: 0, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 20, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 40, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 60, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 80, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 100, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 120, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 140, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 160, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 180, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 200, Training Loss: 0.094215, Validation Loss: 0.095123
Accuracy: 8.000000
Time 0.00s
```

In [11]:

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

model2 = nn.Sequential(
            nn.Linear(5,16),
            nn.ReLU(),
            nn.Linear(16,8),
            nn.ReLU(),
            nn.Linear(8,1),
            nn.ReLU(),
            nn.LogSoftmax(dim=1))
model = model.float()
learning_rate = 0.1

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

out = model2(train_X.float())


loss_fn = nn.MSELoss()

n_epochs = 200
correct = 0
total = 0

for epoch in range(n_epochs+1):
        training_start_time = time.time()
        out = model2(train_X.float())
        loss = loss_fn(out,train_Y.float())

        out_v= model2(val_X.float()) # <1>
        val_loss = loss_fn(out_v, val_Y.float())

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if epoch==0 or epoch % 20 ==0:
            print("Epoch: %d, Training Loss: %f, Validation Loss: %f" % (epoch, float(l
        _, predicted = torch.max(out, dim=1)
        total += val_Y.shape[0]
        correct += int((predicted == train_Y).sum())

print("Accuracy: %f" % (correct / total))
print('Time {:.2f}s'.format(time.time() - training_start_time))
```

```
Epoch: 0, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 20, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 40, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 60, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 80, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 100, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 120, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 140, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 160, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 180, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 200, Training Loss: 0.094215, Validation Loss: 0.095123
Accuracy: 8.000000
Time 0.00s
```