In [2]:

```
import numpy as np
from torch import nn
import torch
from torchvision import datasets, transforms
from sklearn.model_selection import train_test_split
import pandas as pd
import math
from sklearn.linear_model import LinearRegression
from torch.utils.data import TensorDataset
import torch.nn.functional as F

if torch.cuda.is_available():
    device = torch.device('cuda')
else:
    device = torch.device('cpu')

print('Using PyTorch version:', torch.__version__, ' Device:', device)
```

```
Using PyTorch version: 1.4.0  Device: cpu
```

In [3]:

```
df = pd.read_csv('Dataset-4/household_power_consumption.txt', sep=';', low_memory=Fa
```

In [4]:

```
df = df[['Global_active_power']]
```

In [5]:

```
df.iloc[:,0] = df.iloc[:,0].fillna(df.iloc[:,0].mean())
```

In [6]:

```
df.head()
```

Out[6]:

| | Global_active_power |
|---|---|
| **0** | 4.216 |
| **1** | 5.360 |
| **2** | 5.374 |
| **3** | 5.388 |
| **4** | 3.666 |

In [7]:

```
a = df.shape
```

In [8]:

```
data = []
```

# Linear Regression

In [9]:

```
for i in range(60, a[0]+1):
    data.append(df.iloc[i-60:i,0])
```

In [10]:

```
x_train = np.array(data)
```

In [11]:

```
y_train = np.array(df.iloc[60:a[0]+1,0])
```

In [12]:

```
x_train = x_train[:-1]
```

In [13]:

```
reg = LinearRegression().fit(x_train, y_train)
```

In [14]:

```
reg.score(x_train, y_train)
```

Out[14]:

```
0.9389125749029633
```

In [15]:

```
reg.coef_
```

Out[15]:

```
array([ 7.57310589e-03, -3.21065458e-03, -1.10667268e-04, -1.27282133e
-03,
       -3.57005603e-04,  2.33058369e-03, -6.58377457e-04, -1.20640020e
-03,
        1.40212268e-03,  2.57095035e-03, -3.95237886e-03,  1.52086393e
-03,
       -8.12909035e-04, -2.18253951e-03, -3.84794908e-04,  3.48074869e
-03,
        3.37032894e-03, -6.69151765e-04,  3.65954637e-03,  2.49591470e
-03,
       -3.06530523e-03, -1.82434959e-06, -9.84578896e-04, -4.18009936e
-04,
        7.21904608e-04,  1.85299210e-03,  2.69574516e-03, -3.53533885e
-03,
       -1.24381624e-03,  2.23041410e-03,  9.03947956e-04, -2.53621766e
-03,
        2.61597689e-04,  1.18694579e-03,  1.13953247e-04, -1.99670003e
-03,
        4.29409968e-03, -4.12373094e-03,  4.55037796e-03,  4.28222283e
-03,
       -1.22802420e-02,  3.55948556e-03,  1.37135887e-02,  8.25015453e
-03,
        4.85110578e-03,  4.00208750e-03,  6.27591046e-04, -3.68888603e
-03,
        3.45902088e-03, -5.12212250e-03,  4.22058598e-03, -4.52799571e
-03,
       -1.51837332e-02, -3.81683958e-02,  5.80598689e-02,  2.05420278e
-02,
        4.17159164e-02,  1.24598338e-02, -1.36314084e-01,  1.00365636e
+00])
```

In [16]:

```
y_pred = reg.predict(x_train)
```

In [23]:

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_pred, y_train)
```

In [ ]:

```
# print(y_pred)
```

In [24]:

```
mse
```

Out[24]:

```
0.06741620938643988
```

## Multi Layer Perceptron

In [15]:

```
input_size = 60
hidden_sizes = [40, 20]
output_size = 1

model = nn.Sequential(nn.Linear(input_size, hidden_sizes[0]),
                      nn.ReLU(),
                      nn.Linear(hidden_sizes[0], hidden_sizes[1]),
                      nn.ReLU(),
                      nn.Linear(hidden_sizes[1], output_size))

optimizer = torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.5)

print(model)
```

```
Sequential(
  (0): Linear(in_features=60, out_features=40, bias=True)
  (1): ReLU()
  (2): Linear(in_features=40, out_features=20, bias=True)
  (3): ReLU()
  (4): Linear(in_features=20, out_features=1, bias=True)
)
```

In [16]:

```
y_train = y_train.reshape(y_train.shape[0],1)
```

In [17]:

```
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2)
```

In [18]:

```
x_train, x_val, y_train, y_val = map(torch.tensor, (x_train, x_val, y_train, y_val))
x_train = x_train.type(torch.FloatTensor)
y_train = y_train.type(torch.FloatTensor)
x_val = x_val.type(torch.FloatTensor)
y_val = y_val.type(torch.FloatTensor)

train_dataset = TensorDataset(x_train, y_train)
val_dataset = TensorDataset(x_val, y_val)
```

In [19]:

```
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=32, shuffle=Tru
val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=32, shuffle=True)
```

In [21]:

```
def train(epoch, log_interval=20000):
    # Set model to training mode
    model.train()

    # Loop over each batch from the training set
    for batch_idx, (data, target) in enumerate(train_loader):

        data = data.view(data.shape[0], -1)
        # Copy data to GPU if needed
        data = data.to(device)
        target = target.to(device)

        # Zero gradient buffers
        optimizer.zero_grad()

        # Pass data through the network
        output = model(data)

        # Calculate loss
        loss = F.mse_loss(output, target)

        # Backpropagate
        loss.backward()

        # Update weights
        optimizer.step()

        if batch_idx % log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.data.item()))
```

In [55]:

```
def validate(loss_vector, accuracy_vector):
    pred_arr = []
    model.eval()
    val_loss, correct = 0, 0
    for data, target in val_loader:
        data = data.view(data.shape[0], -1)
        data = data.to(device)
        target = target.to(device)
        output = model(data)
        val_loss += F.mse_loss(output, target).data.item()
        pred = output.data.max(1)[0] # get the index of the max log-probability
        for i in np.array(pred):
            pred_arr.append(i)

    val_loss /= len(val_loader)
    loss_vector.append(val_loss)

    print('\nValidation set: Average loss: {:.4f}\n'.format(
        val_loss))

    return pred_arr
```

In [56]:

```
%%time
epochs = 10

lossv, accv = [], []
for epoch in range(1, epochs + 1):
    train(epoch)
    output = validate(lossv, accv)
```

```
Train Epoch: 1 [0/1660159 (0%)] Loss: 0.130404
Train Epoch: 1 [640000/1660159 (39%)]   Loss: 0.027199
Train Epoch: 1 [1280000/1660159 (77%)]  Loss: 0.014153

Validation set: Average loss: 0.0626

Train Epoch: 2 [0/1660159 (0%)] Loss: 0.038056
Train Epoch: 2 [640000/1660159 (39%)]   Loss: 0.013188
Train Epoch: 2 [1280000/1660159 (77%)]  Loss: 0.011561

Validation set: Average loss: 0.0621

Train Epoch: 3 [0/1660159 (0%)] Loss: 0.036577
Train Epoch: 3 [640000/1660159 (39%)]   Loss: 0.143138
Train Epoch: 3 [1280000/1660159 (77%)]  Loss: 0.011874

Validation set: Average loss: 0.0614

Train Epoch: 4 [0/1660159 (0%)] Loss: 0.122412
Train Epoch: 4 [640000/1660159 (39%)]   Loss: 0.014628
Train Epoch: 4 [1280000/1660159 (77%)]  Loss: 0.037135

Validation set: Average loss: 0.0630

Train Epoch: 5 [0/1660159 (0%)] Loss: 0.103690
Train Epoch: 5 [640000/1660159 (39%)]   Loss: 0.045446
Train Epoch: 5 [1280000/1660159 (77%)]  Loss: 0.012782

Validation set: Average loss: 0.0600

Train Epoch: 6 [0/1660159 (0%)] Loss: 0.040755
Train Epoch: 6 [640000/1660159 (39%)]   Loss: 0.034850
Train Epoch: 6 [1280000/1660159 (77%)]  Loss: 0.019612

Validation set: Average loss: 0.0612

Train Epoch: 7 [0/1660159 (0%)] Loss: 0.119588
Train Epoch: 7 [640000/1660159 (39%)]   Loss: 0.003428
Train Epoch: 7 [1280000/1660159 (77%)]  Loss: 0.141765

Validation set: Average loss: 0.0644

Train Epoch: 8 [0/1660159 (0%)] Loss: 0.211412
Train Epoch: 8 [640000/1660159 (39%)]   Loss: 0.113606
Train Epoch: 8 [1280000/1660159 (77%)]  Loss: 0.037486

Validation set: Average loss: 0.0602

Train Epoch: 9 [0/1660159 (0%)] Loss: 0.073469
Train Epoch: 9 [640000/1660159 (39%)]   Loss: 0.015832
Train Epoch: 9 [1280000/1660159 (77%)]  Loss: 0.054559
```

```
Validation set: Average loss: 0.0618

Train Epoch: 10 [0/1660159 (0%)]        Loss: 0.221374
Train Epoch: 10 [640000/1660159 (39%)]  Loss: 0.074724
Train Epoch: 10 [1280000/1660159 (77%)] Loss: 0.043197

Validation set: Average loss: 0.0650

CPU times: user 7min 31s, sys: 6.58 s, total: 7min 37s
Wall time: 7min 37s
```

In [57]:

```
output
```

Out[57]:

```
[0.70004773,
 3.38912,
 0.30652094,
 0.15509534,
 0.46095562,
 2.283328,
 1.3422738,
 0.22774863,
 0.27269435,
 0.18676448,
 1.2484523,
 0.25082445,
 2.352014,
 0.37089396,
 1.5387925,
 1.1215504,
 3.6695535,
 0.22235775,
```

In [ ]:

```

```