SHAPE nucleotide reactivity value prediction using Feedforward Neural Network

Requirements:

- Python 3
- Dictionaries & Lists
- Numpy
- Pandas
- Pytorch

Software Used:

- Google Colab or Jupyter Notebook

Importing Relevant Libraries

Note:

- If You do no have them installed, Run **!pip install "Library Name here"**. For Example, if I want to install **numpy**, I will write !pip install numpy

In [1]:

```
import numpy as np
import pandas as pd
import random
import json
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
import os
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

Creating Training Data:

We will transform the data into a format that our PyTorch Model can Easily Understand

20 x : 1 y

In [2]:

```
# Sliding window with window size 21, the middle value as target value y, the other 20 values as x
# xxxxxxxxxxyxxxxxxxxxx

X = []
y = []
window_size = 21

def sliding_window_extraction(df):
    length = df.shape[0]
    for index in range(length-20):
        x_i = []
        for k in range(index, index+window_size):
            x_i.append(df['reactivity'][k])
        # remove the 11th element y from 21 elements
        x_i.pop(10)
        X.append(x_i)
        y.append(df['reactivity'][index+10])
```

In [3]:

```
# Read the data file from the folder

entries = os.scandir(r'./shape')
i = 0

for entry in entries:
    df = pd.read_csv(entry, delimiter = "\t", header=None, names=["index", "reactivity"])
    sliding_window_extraction(df)
    i = i+1

print('the number of files: ', i)
```

```
the number of files:  274
```

In [4]:

```
# Dimensions of the data
print(len(X))
print(len(y))
print("x: ", X[1])
print("X size:", len(X[1]))
print("y: ", (y[1]))
```

```
24951
24951
x:  [-999.0, -0.044254, 0.174412, 0.028481, 0.26327, 0.238318, 1.543775, 0.334246, 0.081736, 0.267372, 1.124189, 0.151453,
2.946678, 0.856317, 0.340136, 0.089146, 0.083207, 1.431363, 0.303295, 0.454896]
X size: 20
y:  2.423364
```

In [5]:

```
# create training data
X_train = []
y_train = []

# Let's say we want to split the data in 80:10:10 for train:valid:test dataset
train_size=0.8

# In the first step we will split the data in training and remaining dataset
X_train, X_rem, y_train, y_rem = train_test_split(X,y, train_size=train_size)

# Now since we want the valid and test size to be equal (10% each of overall data).
# we have to define valid_size=0.5 (that is 50% of remaining data)
test_size = 0.5
X_valid, X_test, y_valid, y_test = train_test_split(X_rem,y_rem, test_size=test_size)

X_train = np.array(X_train)
y_train = np.array(y_train)
X_valid = np.array(X_valid)
y_valid = np.array(y_valid)
X_test = np.array(X_test)
y_test = np.array(y_test)

print(X_train.shape), print(y_train.shape)
print(X_valid.shape), print(y_valid.shape)
print(X_test.shape), print(y_test.shape)
```

```
(19960, 20)
(19960,)
(2495, 20)
(2495,)
(2496, 20)
(2496,)
```

Out[5]:

```
(None, None)
```

PyTorch Model

Here we will be making a class to implement our custom neural network. It will be a feed forward neural Network which will have 3 Linear Layers and we will be using activation function “ReLU” .

Feed Forward Neural Network

`A feedforward neural network is an artificial neural network wherein connections between the nodes do not form a cycle. As such, it is different from its descendant: recurrent neural networks.`

The feedforward neural network was the first and simplest type of artificial neural network devised In this network, the information moves in only one direction—forward—from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network [1]

ReLU Function:

There are a number of widely used activation functions in deep learning today. One of the simplest is the rectified linear unit, or ReLU function, which is a piece wise linear function that outputs zero if its input is negative, and directly outputs the input otherwise:

Creating our Model

In [6]:

```
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.l1 = nn.Linear(input_size, hidden_size)
        self.l2 = nn.Linear(hidden_size, hidden_size)
        self.l3 = nn.Linear(hidden_size, num_classes)
        self.relu = nn.ReLU()

    def forward(self, x):
        out = self.l1(x)
        out = self.relu(out)
        out = self.l2(out)
        out = self.relu(out)
        out = self.l3(out)
        # no activation and no softmax at the end
        return out
```

Here we have inherited a class from NN.Module because we will be customizing the model & its layers

Assigning the Dataset to the Model:

In [7]:

```
class TrainDataset(Dataset):

    def __init__(self):
        self.n_samples = len(X_train)
        self.x_data = X_train
        self.y_data = y_train

    # support indexing such that dataset[i] can be used to get i-th sample
    def __getitem__(self, index):
        return self.x_data[index], self.y_data[index]

    # we can call len(dataset) to return the size
    def __len__(self):
        return self.n_samples
```

In [8]:

```
class ValidDataset(Dataset):

    def __init__(self):
        self.n_samples = len(X_valid)
        self.x_data = X_valid
        self.y_data = y_valid

    # support indexing such that dataset[i] can be used to get i-th sample
    def __getitem__(self, index):
        return self.x_data[index], self.y_data[index]

    # we can call len(dataset) to return the size
    def __len__(self):
        return self.n_samples
```

Hyper Parameters:

Every Neural network has a set of hyper parameters that need to be set before use.

Before Instantiating our Neural Net Class or Model that we wrote earlier, we will first define some hyper parameters which can be changed accordingly.

In [9]:

```
# Hyper-parameters
num_epochs = 100
batch_size = 16
learning_rate = 0.0001
input_size = len(X_train[0])
hidden_size = 8
output_size = 1
print(input_size, output_size)
```

```
20 1
```

Loss and optimizer:

We will now Instantiate the model, loss and optimizer functions.

```
Loss Function: Mean Squared Error = MSELoss
Optimizer: Stochastic Gradient Descent = SGD Optimizer
```

In [10]:

```
train_dataset = TrainDataset()
valid_dataset = ValidDataset()

train_loader = DataLoader(dataset=train_dataset,
                          batch_size=batch_size,
                          shuffle=True,
                          num_workers=0)

valid_loader = DataLoader(dataset=valid_dataset,
                          batch_size=batch_size,
                          shuffle=True,
                          num_workers=0)

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

model = NeuralNet(input_size, hidden_size, output_size).to(device)

# Loss and optimizer
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

Test Loss before training

In [11]:

```
model.eval()
```

Out[11]:

```
NeuralNet(
  (l1): Linear(in_features=20, out_features=8, bias=True)
  (l2): Linear(in_features=8, out_features=8, bias=True)
  (l3): Linear(in_features=8, out_features=1, bias=True)
  (relu): ReLU()
)
```

In [12]:

```
X_test = torch.FloatTensor(X_test)
y_test = torch.FloatTensor(y_test)
y_pred = model(X_test)
before_train = criterion(y_pred.squeeze(), y_test)
print('Test Loss before training: ', before_train.item())
```

```
Test Loss before training:  194454.171875
```

Training the Model

In [13]:

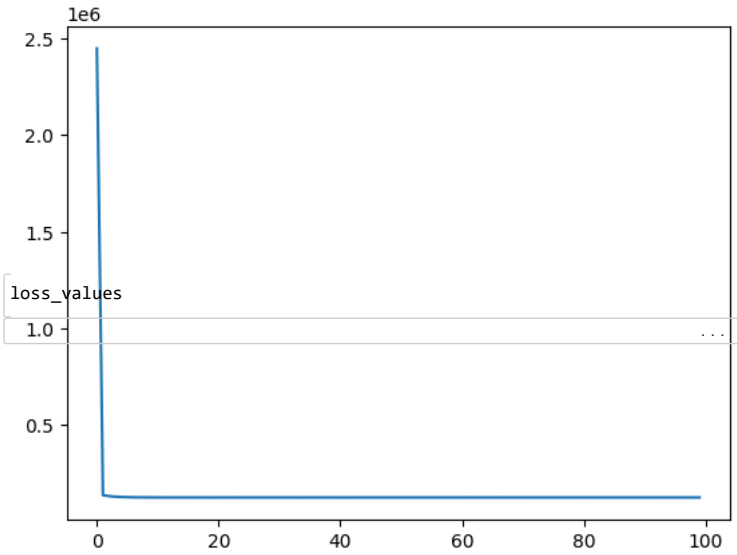
```
# Training the model with Validation
min_valid_loss = np.inf
loss_values = []

for epoch in range(num_epochs):
    train_loss = 0.0
    for (x, y) in train_loader:
        x = x.to(dtype=torch.float32).to(device)
        y = y.to(dtype=torch.float32).to(device)

        # Forward pass
        outputs = model(x)
        # Find the loss
        loss = criterion(outputs, y)

        # Backward and optimize
        # Clear the gradients
        optimizer.zero_grad()
        # Calculate the gradients
        loss.backward()
        # Update weights
        optimizer.step()
        # Calculate Loss
        train_loss += loss.item()

    # Record the loss
    loss_values.append(train_loss / len(train_loader))

    #validation step
    valid_loss = 0.0
    for (x, y) in valid_loader:
        x = x.to(dtype=torch.float32).to(device)
        y = y.to(dtype=torch.float32).to(device)

        # Forward pass
        outputs = model(x)
        # Find the loss
        loss = criterion(outputs, y)
        # Calculate Loss
        valid_loss += loss.item()

    if (epoch+1) % 5 == 0:
        print (f'Epoch [{epoch+1}/{num_epochs}], Training Loss: {train_loss/ len(train_loader):.4f}, Validation Loss: {valid_loss/ len(va

    if min_valid_loss > valid_loss:
        print(f'Validation Loss Decreased({min_valid_loss/ len(valid_loader):.4f} ---> {valid_loss/ len(valid_loader):.4f}) \t Saving The
        min_valid_loss = valid_loss
        # Saving State Dict
        data = {
        "model_state": model.state_dict(),
        "input_size": input_size,
        "hidden_size": hidden_size,
        "output_size": output_size,
        }
        FILE = "data.pth"
        torch.save(data, FILE)

print(f'final loss: {loss.item():.4f}')

plt.plot(loss_values)
```

```
/home/liuxr/miniconda3/lib/python3.10/site-packages/torch/nn/modules/loss.py:536: UserWarning: Using a target size (torch.S
ize([16])) that is different to the input size (torch.Size([16, 1])). This will likely lead to incorrect results due to bro
adcasting. Please ensure they have the same size.
  return F.mse_loss(input, target, reduction=self.reduction)
/home/liuxr/miniconda3/lib/python3.10/site-packages/torch/nn/modules/loss.py:536: UserWarning: Using a target size (torch.S
ize([8])) that is different to the input size (torch.Size([8, 1])). This will likely lead to incorrect results due to broad
casting. Please ensure they have the same size.
  return F.mse_loss(input, target, reduction=self.reduction)
/home/liuxr/miniconda3/lib/python3.10/site-packages/torch/nn/modules/loss.py:536: UserWarning: Using a target size (torch.S
ize([15])) that is different to the input size (torch.Size([15, 1])). This will likely lead to incorrect results due to bro
adcasting. Please ensure they have the same size.
  return F.mse_loss(input, target, reduction=self.reduction)
```

```
Validation Loss Decreased(inf ---> 149377.8360)          Saving The Model
Validation Loss Decreased(149377.8360 ---> 143160.7988)          Saving The Model
Validation Loss Decreased(143160.7988 ---> 139246.0104)          Saving The Model
Validation Loss Decreased(139246.0104 ---> 136817.1752)          Saving The Model
Epoch [5/100], Training Loss: 129591.6507, Validation Loss: 135322.1813
Validation Loss Decreased(136817.1752 ---> 135322.1813)          Saving The Model
Validation Loss Decreased(135322.1813 ---> 134438.4531)          Saving The Model
Validation Loss Decreased(134438.4531 ---> 133797.8908)          Saving The Model
Validation Loss Decreased(133797.8908 ---> 133428.4060)          Saving The Model
Validation Loss Decreased(133428.4060 ---> 133138.7576)          Saving The Model
Epoch [10/100], Training Loss: 127172.8961, Validation Loss: 132974.5124
Validation Loss Decreased(133138.7576 ---> 132974.5124)          Saving The Model
Validation Loss Decreased(132974.5124 ---> 132882.6905)          Saving The Model
Validation Loss Decreased(132882.6905 ---> 132841.7784)          Saving The Model
Validation Loss Decreased(132841.7784 ---> 132768.1793)          Saving The Model
Validation Loss Decreased(132768.1793 ---> 132711.2046)          Saving The Model
Epoch [15/100], Training Loss: 127043.3741, Validation Loss: 132743.0618
Validation Loss Decreased(132711.2046 ---> 132685.6206)          Saving The Model
Validation Loss Decreased(132685.6206 ---> 132643.2162)          Saving The Model
Epoch [20/100], Training Loss: 126956.3612, Validation Loss: 132667.3794
Validation Loss Decreased(132643.2162 ---> 132627.0297)          Saving The Model
Validation Loss Decreased(132627.0297 ---> 132620.6018)          Saving The Model
Epoch [25/100], Training Loss: 127025.4413, Validation Loss: 132599.4659
Validation Loss Decreased(132620.6018 ---> 132599.4659)          Saving The Model
Epoch [30/100], Training Loss: 126990.2743, Validation Loss: 132613.4265
Epoch [35/100], Training Loss: 126955.2279, Validation Loss: 132632.7906
Epoch [40/100], Training Loss: 126990.2700, Validation Loss: 132611.7306
Epoch [45/100], Training Loss: 126955.2209, Validation Loss: 132630.8972
Epoch [50/100], Training Loss: 127025.5268, Validation Loss: 132650.6443
Epoch [55/100], Training Loss: 127025.4395, Validation Loss: 132686.9836
Epoch [60/100], Training Loss: 126955.0680, Validation Loss: 132594.6854
Validation Loss Decreased(132599.4659 ---> 132594.6854)          Saving The Model
Epoch [65/100], Training Loss: 127060.2940, Validation Loss: 132630.8043
Epoch [70/100], Training Loss: 126990.2497, Validation Loss: 132629.8481
Epoch [75/100], Training Loss: 126990.2600, Validation Loss: 132668.4062
Epoch [80/100], Training Loss: 126990.3131, Validation Loss: 132592.8847
Validation Loss Decreased(132594.6854 ---> 132592.8847)          Saving The Model
Epoch [85/100], Training Loss: 126990.3248, Validation Loss: 132630.7732
Epoch [90/100], Training Loss: 126990.2103, Validation Loss: 132667.8920
Validation Loss Decreased(132592.8847 ---> 132592.5595)          Saving The Model
Epoch [95/100], Training Loss: 127060.5713, Validation Loss: 132667.7372
Epoch [100/100], Training Loss: 126990.2944, Validation Loss: 132612.5353
final loss: 69052.1875
```

Out[13]:

```
[<matplotlib.lines.Line2D at 0x7f82c3e72530>]
```




```
loss_values
```

In [16]:

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

FILE = "data.pth"
data = torch.load(FILE)

input_size = data["input_size"]
hidden_size = data["hidden_size"]
output_size = data["output_size"]
model_state = data["model_state"]

model = NeuralNet(input_size, hidden_size, output_size).to(device)
model.load_state_dict(model_state)
model.eval()
```

Out[16]:

```
NeuralNet(
  (l1): Linear(in_features=20, out_features=8, bias=True)
  (l2): Linear(in_features=8, out_features=8, bias=True)
  (l3): Linear(in_features=8, out_features=1, bias=True)
  (relu): ReLU()
)
```

Prediction with test dataset:

In [17]:

```
X_test = torch.FloatTensor(X_test)
y_test = torch.FloatTensor(y_test)
y_pred = model(X_test)
after_train = criterion(y_pred.squeeze(), y_test)
print('Test Loss after training: ', after_train.item())
print('Test Loss before training: ', before_train.item())
```

```
Test Loss after training:  128947.15625
Test Loss before training:  194454.171875
```

Bibliography

[1] Wikipedia, "FeedForward Neural Network," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Feedforward_neural_network (https://en.wikipedia.org/wiki/Feedforward_neural_network).

[2] T. Wood, "Activation Function," DeepAI, [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/activation-function (https://deepai.org/machine-learning-glossary-and-terms/activation-function).