# Model Tuning

June 2, 2022

[1]:
```
import numpy as np
import h5py
import torch
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn
from collections.abc import Iterable
import time
import math

batchSize = 32 #Batch size of training set
```

[2]:
```
def trainNetwork(model, loss_function, optimizer, numEpochs, dataloader,␣
 ↪numOutputs):

    #Set model to training mode
    model.train()

    for epoch in range(numEpochs):

        # Print epoch
        print(f'Starting epoch {epoch+1}')

        # Set current loss value
        current_loss = 0.0

        #Reset model parameters

        # Iterate over the DataLoader for training data
        for i, data in enumerate(dataloader, 0):

            # Get and prepare input

            preProcessedInputs = data[:, 0:4] #This line doesn't really do␣
 ↪anything, delete later?
            targets = data[:, 4:(4+numOutputs)]

            #Process intensity by putting it on a log scale
```

```
        intens = data[:, 0:1]
        intens = np.log(intens)
        inputs = torch.cat((intens, data[:,1:4]), axis = 1)

        #Process targets by putting them on a log scale
        targets = np.log(targets)

        #print(type(inputs))

        #Comment the next two lines out if not using GPU
        inputs = inputs.to('cuda')
        targets = targets.to('cuda')

        #Normalize inputs
        inputs, targets = inputs.float(), targets.float()
        targets = targets.reshape((targets.shape[0], numOutputs))

        # Zero the gradients
        optimizer.zero_grad()

        # Perform forward pass
        inputs = inputs
        outputs = model(inputs)

        #The following two lines are for debugging only
#         if i % 10 == 0:
#             print("Targets:", targets[0:2])
#             print("Outputs:", outputs[0:2])
#             print()
#             print()

        # Compute loss
        loss = loss_function(outputs, targets)

        # Perform backwards propagation
        loss.backward()

        # Perform optimization
        optimizer.step()

        # Print statistics
        current_loss += loss.item()
        if i % 10 == 0:
            print('Loss after mini-batch %5d: %.3f' %
                  (i + 1, current_loss / 500))
            current_loss = 0.0
```

```
    # Process is complete.
    print('Training process has finished.\n')
```

[3]:
```
def calc_MSE_Error(target, output, index):

    targetNP = np.exp(target[:, index].cpu().detach().numpy())
    outputNP = np.exp(output[:, index].cpu().detach().numpy())

#     print(targetNP)
#     print(outputNP)

    result = np.square(np.subtract(targetNP, outputNP)).mean()

#     print("Index: ", index)
#     print(target)
#     print(output)

    print("Result:", result)

    return result
```

[4]:
```
def calc_Avg_Percent_Error(target, output, index):

    targetNP = np.exp(target[:, index].cpu().detach().numpy())
    outputNP = np.exp(output[:, index].cpu().detach().numpy())

    difference = targetNP - outputNP
    difference = np.abs(difference)
    error = np.divide(difference, outputNP) * 100

    result = error.mean()

    return result
```

[5]:
```
def getModelError(model, epochList, loss_function, trainDataset, testDataset):
    mseErrorList = []
    avgErrorList = []
    mseTrainList = []
    avgTrainList = []
    timeList = []

    #print("Epochs to test:", epochList)


    for numEpochs in epochList:

        #Reset model parameters
```

```
        for layer in model.children():
            if hasattr(layer, 'reset_parameters'):
                layer.reset_parameters()

        print("Training with", numEpochs, "epochs.")

        #Define optimizer
        optimizer = torch.optim.Adam(model.parameters(), lr=1e-2)

        #Create dataloader for training set
        dataloader = DataLoader(trainDataset, batch_size=batchSize,␣
↪shuffle=True)

        #Start clock
        startTime = time.time()

        #First train the network
        trainNetwork(model, loss_function, optimizer, numEpochs, dataloader,␣
↪numOutputs = 3)

        #End clock
        endTime = time.time()
        timeSpent = endTime - startTime #In seconds

        #Next test the network
        model.eval()

        #Create dataloader for testing set
        testDataloader = DataLoader(testDataset, batch_size=math.floor(0.
↪1*numPoints), shuffle=True)
        iterDataLoader = iter(testDataloader)
        testData = next(iterDataLoader)

        #Process the intens value so it is in a log scale
        intens = testData[:, 0:1]
        logIntens = np.log(intens)

        #Create the final tensor of inputs we will feed into the model
        inputs = torch.cat((logIntens, testData[:,1:4]), axis = 1)

        #Create the tensor of our actual values
        target = testData[:, 4:7]
        target = np.log(target)

        #Push our tensors to the GPU
        inputs = inputs.to('cuda')
        target = target.to('cuda')
```

```
        inputs, target = inputs.float(), target.float()
        target = target.reshape((target.shape[0], 3))

        #Get the model predictions and apply a log-scale to our actual values
        #(Model predictions already have a log-scale applied to them)
        output = model(inputs)
        target = np.log(testData[:, 4:7])

#         print(output)
#         print(target)
#         print('\n')



        #Initialize error lists
        #Index mappings:
        #0 = Max KE
        #1 = Total Energy
        #2 = Average Energy
        error = [0., 0., 0.]
        percentError = [0., 0., 0.]


        print("Calculate error for test")
        for index in range(3):
            error[index] = calc_MSE_Error(target, output, index)
            percentError[index] = calc_Avg_Percent_Error(target, output, index)

        #Append error values into our list
        mseErrorList.append(error)
        avgErrorList.append(percentError)
        timeList.append(timeSpent)

        #Also retrieve the testing error

        dataloader = DataLoader(trainDataset, batch_size=math.floor(0.1 *␣
↪numPoints), shuffle=True)
        iterDataLoader = iter(dataloader)
        trainData = next(iterDataLoader)

        #Process the intens value so it is in a log scale
        intens = trainData[:, 0:1]
        logIntens = np.log(intens)

        #Create the final tensor of inputs we will feed into the model
        inputs = torch.cat((logIntens, trainData[:,1:4]), axis = 1)
```

```
        #Create the tensor of our actual values
        target = trainData[:, 4:7]
        target = np.log(target)

        #Push our tensors to the GPU
        inputs = inputs.to('cuda')
        target = target.to('cuda')

        inputs, target = inputs.float(), target.float()
        target = target.reshape((target.shape[0], 3))

        #Get the model predictions and apply a log-scale to our actual values
        #(Model predictions already have a log-scale applied to them)
        trainOutput = model(inputs)
        trainTarget = np.log(trainData[:, 4:7])

#          print(output)
#          print(target)


        print("Calculate error for train")

        trainError = [0., 0., 0.]
        trainPercentError = [0., 0., 0.]

        for index in range(3):
            trainError[index] = calc_MSE_Error(trainTarget, trainOutput, index)
            trainPercentError[index] = calc_Avg_Percent_Error(trainTarget,␣
↪trainOutput, index)

        #Append error values
        mseTrainList.append(trainError)
        avgTrainList.append(trainPercentError)

    return mseErrorList, avgErrorList, mseTrainList, avgTrainList, timeList
```

# 1 Define our neural networks

[6]:
```
#Neural network with 2 hidden layers

class MultiRegressor2Layers(nn.Module):
  '''
    Multilayer Perceptron for regression.
  '''
  def __init__(self):
```

```python
    super().__init__()
    self.norm0 = nn.BatchNorm1d(4)
    self.linear1 = nn.Linear(in_features=4, out_features=64)
    self.norm1 = nn.BatchNorm1d(64)
    self.act1 = nn.LeakyReLU()
    self.dropout = nn.Dropout()
    self.linear2 = nn.Linear(in_features=64, out_features=16)
    self.norm2 = nn.BatchNorm1d(16)
    #self.dropout = nn.Dropout()
    self.act2 = nn.LeakyReLU()
    self.linear3 = nn.Linear(in_features=16, out_features=8)
    self.act3 = nn.LeakyReLU()
    #self.dropout = nn.Dropout()
    self.output = nn.Linear(in_features=8, out_features = 3)



  def forward(self, x):
    '''
      Forward pass
    '''
    x = self.norm0(x)
    x = self.linear1(x)
    x = self.norm1(x)
    x = self.act1(x)
    #x = self.dropout(x)
    x = self.linear2(x)
    x = self.norm2(x)
    #x = self.dropout(x)
    x = self.act2(x)
    x = self.linear3(x)
    x = self.act3(x)
    #x = self.dropout(x)
    x = self.output(x)


    return x
```

[7]:
```python
#Neural network with 1 hidden layer

class MultiRegressor1Layer(nn.Module):
  '''
    Multilayer Perceptron for regression.
  '''
  def __init__(self):
    super().__init__()
    self.norm0 = nn.BatchNorm1d(4)
```

```
        self.linear1 = nn.Linear(in_features=4, out_features=64)
        self.norm1 = nn.BatchNorm1d(64)
        self.act1 = nn.LeakyReLU()
        self.dropout = nn.Dropout()
        self.linear2 = nn.Linear(in_features=64, out_features=16)
        self.norm2 = nn.BatchNorm1d(16)
        #self.dropout = nn.Dropout()
        self.act2 = nn.LeakyReLU()
        self.output = nn.Linear(in_features=16, out_features = 3)



    def forward(self, x):
        '''
          Forward pass
        '''
        x = self.norm0(x)
        x = self.linear1(x)
        x = self.norm1(x)
        x = self.act1(x)
        #x = self.dropout(x)
        x = self.linear2(x)
        x = self.norm2(x)
        #x = self.dropout(x)
        x = self.act2(x)
        x = self.output(x)


        return x
```

## 2 Read in the data

[29]:
```
numPoints = 20000
#numPoints = 500

#filename = 'Data_Fuchs_v_2.2_lambda_um_0.8_points_' + str(numPoints) +␣
 ↪'_seed_0.h5'
filename = 'Data_Fuchs_v_2.2_Wright_Pat_Narrow_Range_lambda_um_0.8_points_' +␣
 ↪str(numPoints) + '_seed_0.h5'

h5File = h5py.File(filename, 'r+')
```

[30]:
```
#Read columns

intens = h5File['Intensity_(W_cm2)']
duration = h5File['Pulse_Duration_(fs)']
```

```
thickness = h5File['Target_Thickness (um)']
spotSize = h5File['Spot_Size_(FWHM um)']
maxEnergy = h5File['Max_Proton_Energy_(MeV)']
totalEnergy = h5File['Total_Proton_Energy_(MeV)']
avgEnergy = h5File['Avg_Proton_Energy_(MeV)']


#Convert columns into numpy arrays
npIntens = np.fromiter(intens, float)
npDuration = np.fromiter(duration, float)
npThickness = np.fromiter(thickness, float)
npSpot = np.fromiter(spotSize, float)
npMaxEnergy = np.fromiter(maxEnergy, float)
npTotalEnergy = np.fromiter(totalEnergy, float)
npAvgEnergy = np.fromiter(avgEnergy, float)


#Join all of those arrays into one big numpy array
#npFile = np.dstack((npIntens, npDuration, npThickness, npSpot, npMaxEnergy,␣
 ↪npTotalEnergy, npAvgEnergy))

npFile = npFile.reshape(numPoints, 7)

#npTrain = npFile[:math.floor(.9*numPoints), 0:7]
#npTest = npFile[math.floor(.9*numPoints):, 0:7]

npTrain = npFile[:, 0:7]

#print(npFile.shape)
```

[31]:
```
filename_test = 'Data_Fuchs_v_2.2_Wright_Pat_Narrow_Range_lambda_um_0.
 ↪8_points_' + str(100000) + '_seed_1.h5'

h5FileTest = h5py.File(filename_test, 'r+')

#Read columns

intens = h5FileTest['Intensity_(W_cm2)']
duration = h5FileTest['Pulse_Duration_(fs)']
thickness = h5FileTest['Target_Thickness (um)']
spotSize = h5FileTest['Spot_Size_(FWHM um)']
maxEnergy = h5FileTest['Max_Proton_Energy_(MeV)']
totalEnergy = h5FileTest['Total_Proton_Energy_(MeV)']
avgEnergy = h5FileTest['Avg_Proton_Energy_(MeV)']


#Convert columns into numpy arrays
```

```
npIntens = np.fromiter(intens, float)
npDuration = np.fromiter(duration, float)
npThickness = np.fromiter(thickness, float)
npSpot = np.fromiter(spotSize, float)
npMaxEnergy = np.fromiter(maxEnergy, float)
npTotalEnergy = np.fromiter(totalEnergy, float)
npAvgEnergy = np.fromiter(avgEnergy, float)


#Join all of those arrays into one big numpy array
npFile = np.dstack((npIntens, npDuration, npThickness, npSpot, npMaxEnergy,␣
 ↪npTotalEnergy, npAvgEnergy))

npFile = npFile.reshape(100000, 7)

#npTrain = npFile[:math.floor(.9*numPoints), 0:7]
#npTest = npFile[math.floor(.9*numPoints):, 0:7]

npTest = npFile[:, 0:7]

print(npFile.shape)
```

```
(100000, 7)
```

# 3 Prepare our dataset

[32]:
```
training_dataset = h5File.create_dataset(name=None, data=npTrain)
test_dataset = h5File.create_dataset(name=None, data=npTest)
```

[33]:
```
#Choose our loss function

loss_function = nn.MSELoss()
```

[34]:
```
#List which epochs we should test

#epochList = [1]
#epochList = [1, 2, 3]
epochList = [1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80,␣
 ↪85, 90, 100, 150, 200, 250]
#epochList = [1, 5, 10, 15, 20, 25, 50, 75, 100, 150, 200]
#epochList = [1, 5, 10, 15, 20, 50]
#epochList = [1, 5, 10, 15, 20, 25]
```

[ ]:
```
#Initialize neural network and dataloader
model1Layer = MultiRegressor1Layer().to('cuda')
```

```
model1LayerMSE, model1LayerPercentError, trainMSE, trainPercent, timeList =␣
↪getModelError(model1Layer, epochList, loss_function, training_dataset,␣
↪test_dataset)

# print(model1LayerMSE)
# print(trainMSE)
```

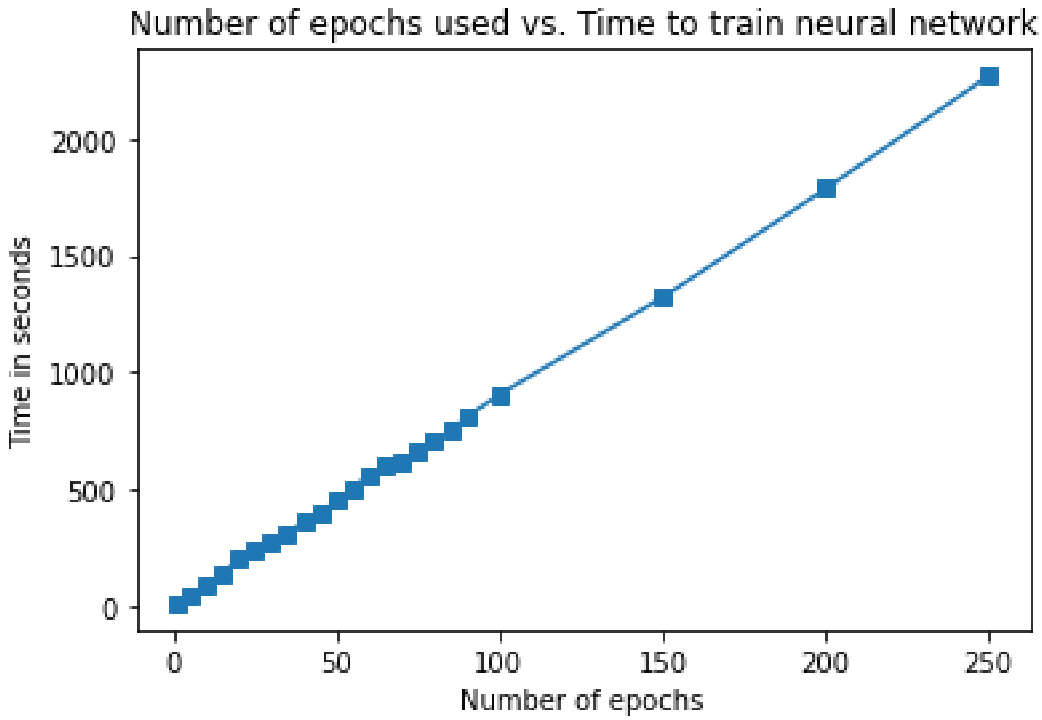
[36]:
```
def splitErrorList(errorList):
    maxEnergyError = []
    totalEnergyError = []
    avgEnergyError = []

    for element in errorList:
        maxEnergyError.append(element[0])
        totalEnergyError.append(element[1])
        avgEnergyError.append(element[2])

    return maxEnergyError, totalEnergyError, avgEnergyError
```

[ ]:
```
# print(model1LayerMSE)
# print(model1LayerPercentError)
# print('\n')
# print(trainMSE)
# print(trainPercent)

maxEnergyMSE, totalEnergyMSE, avgEnergyMSE = splitErrorList(model1LayerMSE)
maxEnergyPercent, totalEnergyPercent, avgEnergyPercent =␣
↪splitErrorList(model1LayerPercentError)

trainMaxMSE, trainTotalMSE, trainAvgMSE = splitErrorList(trainMSE)
trainMaxPercent, trainTotalPercent, trainAvgPercent =␣
↪splitErrorList(trainPercent)

#print(trainMaxMSE)
#print(maxEnergyMSE)
```

# 4 Now plot errors and running time

[38]:
```
%matplotlib inline
import matplotlib.pyplot as plt

#Time spent plot
fig = plt.figure()
ax1 = fig.add_subplot(1,1,1)

plt.plot(epochList, timeList, marker='s')
```

```
plt.title("Number of epochs used vs. Time to train neural network")
plt.xlabel("Number of epochs")
plt.ylabel("Time in seconds")


#plt.legend(loc='upper left');
plt.show()
```



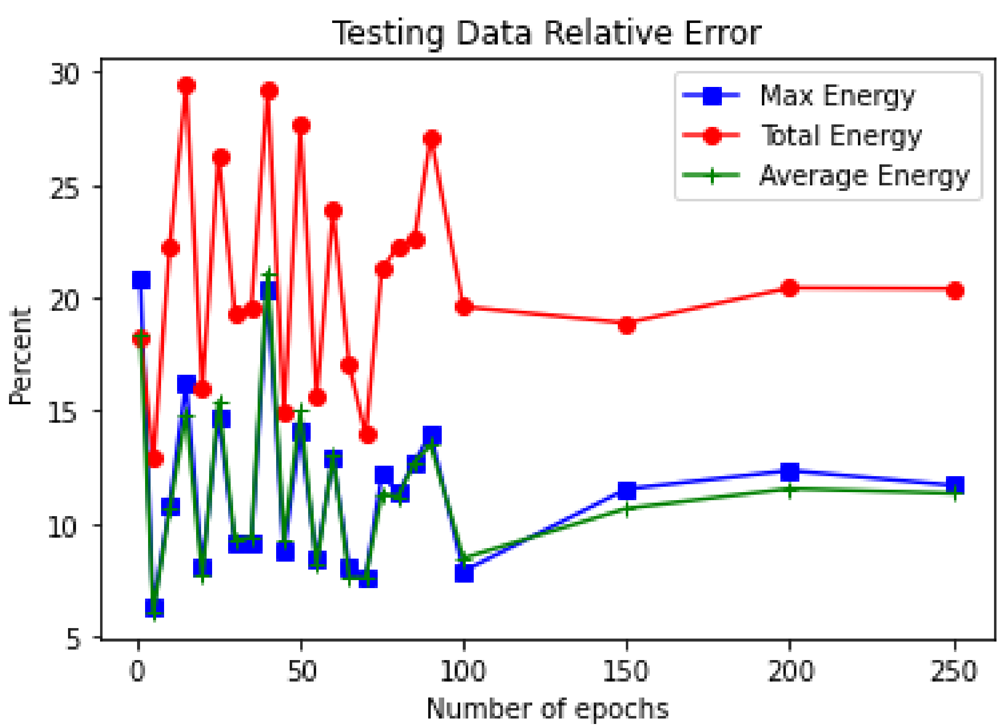

[39]:
```
for epochElement, timeElement in zip(epochList, timeList):

    minuteValue = timeElement / 60

    print("Number of epochs:", epochElement)
    print("Time spent:", minuteValue, "minutes", '\n')
```

```
Number of epochs: 1
Time spent: 0.14703834056854248 minutes

Number of epochs: 5
Time spent: 0.7457958658536276 minutes

Number of epochs: 10
Time spent: 1.4910252849260965 minutes
```

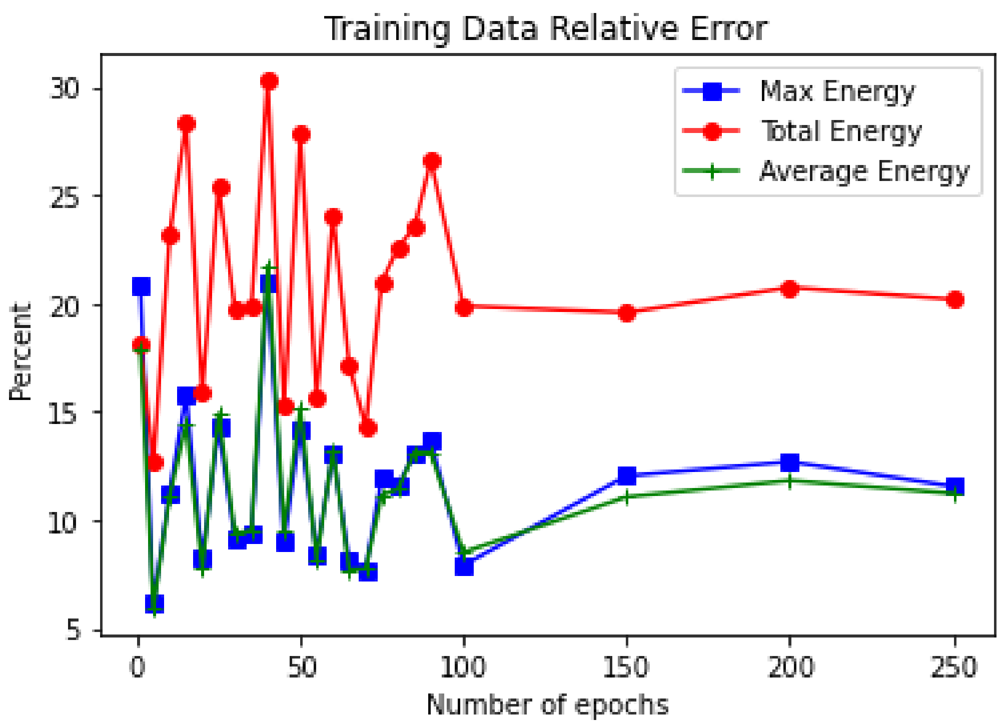
```
Number of epochs: 15
Time spent: 2.292820946375529 minutes

Number of epochs: 20
Time spent: 3.344715170065562 minutes

Number of epochs: 25
Time spent: 3.9810705820719403 minutes

Number of epochs: 30
Time spent: 4.580164515972138 minutes

Number of epochs: 35
Time spent: 5.102071925004323 minutes

Number of epochs: 40
Time spent: 6.038470617930094 minutes

Number of epochs: 45
Time spent: 6.526627858479817 minutes

Number of epochs: 50
Time spent: 7.543764340877533 minutes

Number of epochs: 55
Time spent: 8.418479128678639 minutes

Number of epochs: 60
Time spent: 9.331233354409536 minutes

Number of epochs: 65
Time spent: 10.031408512592316 minutes

Number of epochs: 70
Time spent: 10.151241747538249 minutes

Number of epochs: 75
Time spent: 11.008545513947805 minutes

Number of epochs: 80
Time spent: 11.8485617518425 minutes

Number of epochs: 85
Time spent: 12.469416495164236 minutes

Number of epochs: 90
Time spent: 13.462661596139272 minutes
```

```
Number of epochs: 100
Time spent: 15.062532075246175 minutes

Number of epochs: 150
Time spent: 22.001067531108855 minutes

Number of epochs: 200
Time spent: 29.7786949634552 minutes

Number of epochs: 250
Time spent: 37.83274068832397 minutes
```

[51]:
```
#Percent Error plot
fig = plt.figure()
ax1 = fig.add_subplot(1,1,1)



plt.plot(epochList, maxEnergyPercent, c='b', marker="s", label='Max Energy')
plt.plot(epochList, totalEnergyPercent, c='r', marker="o", label='Total Energy')
plt.plot(epochList, avgEnergyPercent, c='g', marker='+', label='Average Energy')
plt.title("Testing Data Relative Error")
plt.xlabel("Number of epochs")
plt.ylabel("Percent")
plt.legend(loc='upper right')
plt.show()
```



[52]:
```python
#Percent Error plot for training data
fig = plt.figure()
ax1 = fig.add_subplot(1,1,1)

plt.plot(epochList, trainMaxPercent, c='b', marker="s", label='Max Energy')
plt.plot(epochList, trainTotalPercent, c='r', marker="o", label='Total Energy')
plt.plot(epochList, trainAvgPercent, c='g', marker='+', label='Average Energy')
plt.title("Training Data Relative Error")
plt.xlabel("Number of epochs")
plt.ylabel("Percent")
plt.legend(loc='upper right')
plt.show()
```

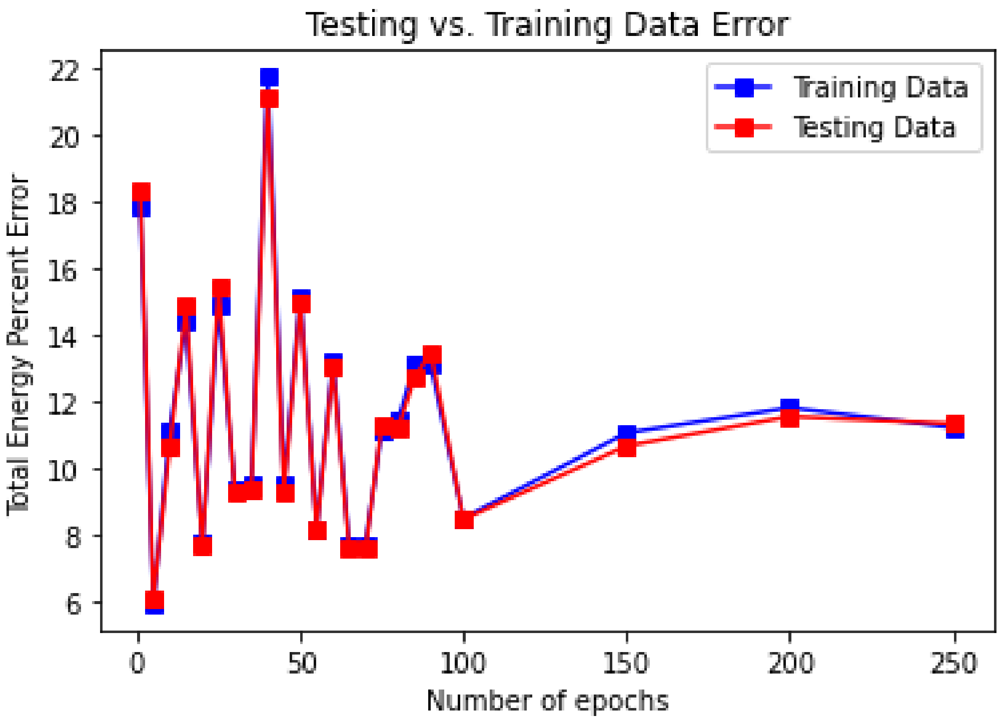



[53]:
```python
#Compare errors of train and test using just the max energy % error

fig = plt.figure()
#ax1 = fig.add_subplot(1,1,1)

plt.plot(epochList, trainMaxPercent, c='b', marker="s", label='Training Data')
plt.plot(epochList, maxEnergyPercent, c='r', marker="s", label='Testing Data')
```

```
plt.title("Testing vs. Training Data Error")
plt.xlabel("Number of epochs")
plt.ylabel("Max Energy Percent Error")
plt.legend(loc='upper right')
plt.show()
```


[54]:
```
#Compare errors of train and test using just the total energy % error

fig = plt.figure()
#ax1 = fig.add_subplot(1,1,1)

plt.plot(epochList, trainTotalPercent, c='b', marker="s", label='Training Data')
plt.plot(epochList, totalEnergyPercent, c='r', marker="s", label='Testing Data')

plt.title("Testing vs. Training Data Error")
plt.xlabel("Number of epochs")
plt.ylabel("Total Energy Percent Error")
plt.legend(loc='upper right')
plt.show()
```


[55]:
```
#Compare errors of train and test using just the avg energy % error

fig = plt.figure()
#ax1 = fig.add_subplot(1,1,1)

plt.plot(epochList, trainAvgPercent, c='b', marker="s", label='Training Data')
plt.plot(epochList, avgEnergyPercent, c='r', marker="s", label='Testing Data')

plt.title("Testing vs. Training Data Error")
plt.xlabel("Number of epochs")
plt.ylabel("Total Energy Percent Error")
plt.legend(loc='upper right')
plt.show()
```




```
[43]: for epoch, maxError, totalError, avgError in zip(epochList, maxEnergyPercent,␣
       ↪totalEnergyPercent, avgEnergyPercent):
          print("Number of epochs:", epoch)
          print("Max energy percent error:", maxError)
          print("Total energy percent error:", totalError)
          print("Average energy percent error:", avgError, '\n')
```

```
Number of epochs: 1
Max energy percent error: 20.87278003795522
Total energy percent error: 18.278400748824065
Average energy percent error: 18.300609469347663

Number of epochs: 5
Max energy percent error: 6.292433321827366
Total energy percent error: 12.894555185327413
Average energy percent error: 6.099359933356997

Number of epochs: 10
Max energy percent error: 10.824749565167979
Total energy percent error: 22.20004334964132
Average energy percent error: 10.645141615041783

Number of epochs: 15
Max energy percent error: 16.23049546093938
```

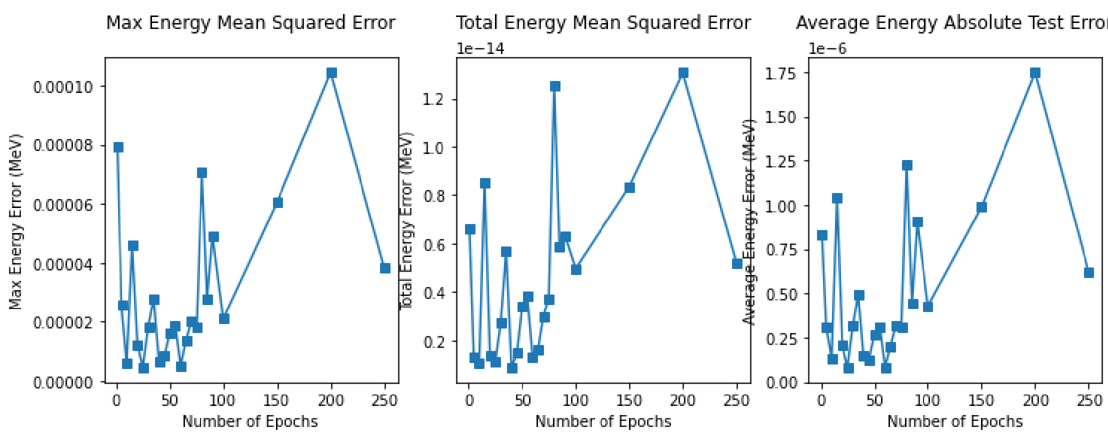
```
Total energy percent error: 29.41500321625747
Average energy percent error: 14.87635943268122

Number of epochs: 20
Max energy percent error: 8.109079144438734
Total energy percent error: 16.021951408149125
Average energy percent error: 7.682810163347712

Number of epochs: 25
Max energy percent error: 14.75316779293649
Total energy percent error: 26.293908007551746
Average energy percent error: 15.416446130359935

Number of epochs: 30
Max energy percent error: 9.130424157808424
Total energy percent error: 19.295348564525987
Average energy percent error: 9.275129060466746

Number of epochs: 35
Max energy percent error: 9.141269329040213
Total energy percent error: 19.579482206878268
Average energy percent error: 9.397601465528368

Number of epochs: 40
Max energy percent error: 20.307473421023865
Total energy percent error: 29.198628130719502
Average energy percent error: 21.102139122343043

Number of epochs: 45
Max energy percent error: 8.8341279915337
Total energy percent error: 14.919483201244534
Average energy percent error: 9.289932889614821

Number of epochs: 50
Max energy percent error: 14.074466067224789
Total energy percent error: 27.63038883666942
Average energy percent error: 14.996706746114512

Number of epochs: 55
Max energy percent error: 8.416797538776743
Total energy percent error: 15.586736914469085
Average energy percent error: 8.160750172149728

Number of epochs: 60
Max energy percent error: 12.914375112271223
Total energy percent error: 23.88993033294988
Average energy percent error: 13.027638933574712
```

```
Number of epochs: 65
Max energy percent error: 8.057336357181724
Total energy percent error: 17.01902153882975
Average energy percent error: 7.6058993829144566

Number of epochs: 70
Max energy percent error: 7.565505299744417
Total energy percent error: 13.952628163527562
Average energy percent error: 7.586146700376765

Number of epochs: 75
Max energy percent error: 12.193658759300002
Total energy percent error: 21.248828568475126
Average energy percent error: 11.330218968109268

Number of epochs: 80
Max energy percent error: 11.346021844652919
Total energy percent error: 22.27386942219833
Average energy percent error: 11.212229891098604

Number of epochs: 85
Max energy percent error: 12.743847457485478
Total energy percent error: 22.59085550378961
Average energy percent error: 12.71617667457452

Number of epochs: 90
Max energy percent error: 13.988382036383832
Total energy percent error: 27.12014269309939
Average energy percent error: 13.495641069467952

Number of epochs: 100
Max energy percent error: 7.897922616144397
Total energy percent error: 19.59624919618485
Average energy percent error: 8.48123049936957

Number of epochs: 150
Max energy percent error: 11.536754725405073
Total energy percent error: 18.87969310548973
Average energy percent error: 10.676295011760246

Number of epochs: 200
Max energy percent error: 12.356540707101708
Total energy percent error: 20.43384763316827
Average energy percent error: 11.558902502372193

Number of epochs: 250
Max energy percent error: 11.703276505441663
Total energy percent error: 20.399311833860175
```

Average energy percent error: 11.346132617351072

[56]:
```
fig = plt.figure(figsize = (12, 4))

plt.subplot(1, 3, 1)
plt.plot(epochList, maxEnergyMSE, marker = 's')
plt.title("Max Energy Mean Squared Error", pad = 20)
plt.xlabel('Number of Epochs')
plt.ylabel('Max Energy Error (MeV)')

plt.subplot(1, 3, 2)
plt.plot(epochList, totalEnergyMSE, marker = 's')
plt.title("Total Energy Mean Squared Error", pad = 20)
plt.xlabel('Number of Epochs')
plt.ylabel('Total Energy Error (MeV)')

plt.subplot(1, 3, 3)
plt.plot(epochList, avgEnergyMSE, marker = 's')
plt.title("Average Energy Absolute Test Error", pad = 20)
plt.xlabel('Number of Epochs')
plt.ylabel('Average Energy Error (MeV)')
```

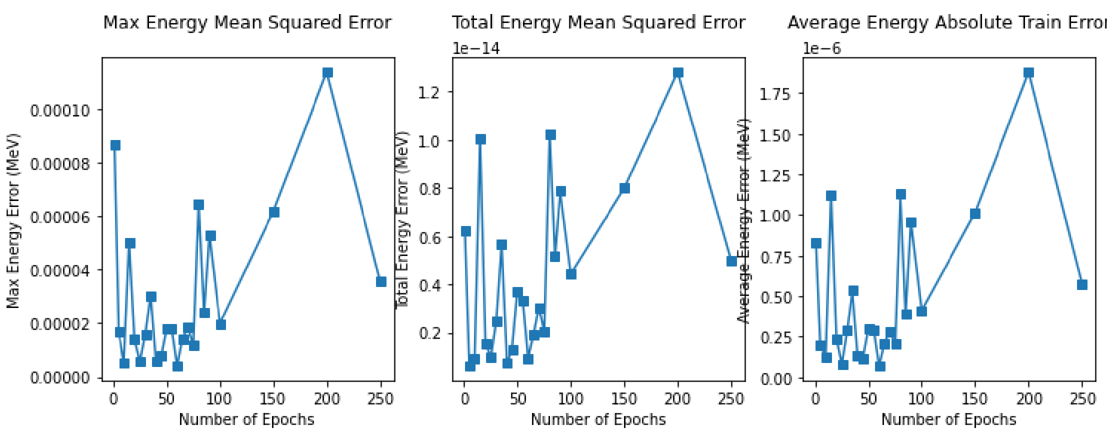
[56]: Text(0, 0.5, 'Average Energy Error (MeV)')




[57]:
```
fig = plt.figure(figsize = (12, 4))

plt.subplot(1, 3, 1)
plt.plot(epochList, trainMaxMSE, marker='s')
plt.title("Max Energy Mean Squared Error", pad = 20)
plt.xlabel('Number of Epochs')
plt.ylabel('Max Energy Error (MeV)')
```

```
plt.subplot(1, 3, 2)
plt.plot(epochList, trainTotalMSE, marker='s')
plt.title("Total Energy Mean Squared Error", pad = 20)
plt.xlabel('Number of Epochs')
plt.ylabel('Total Energy Error (MeV)')

plt.subplot(1, 3, 3)
plt.plot(epochList, trainAvgMSE, marker='s')
plt.title("Average Energy Absolute Train Error", pad = 20)
plt.xlabel('Number of Epochs')
plt.ylabel('Average Energy Error (MeV)')
```

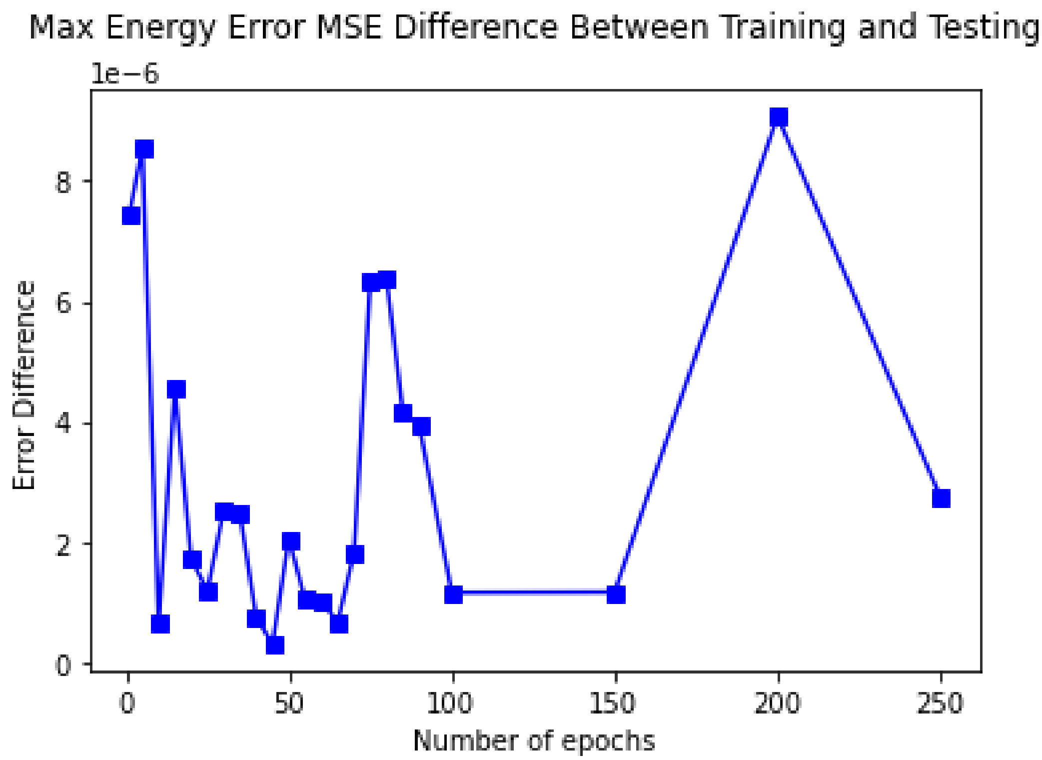
[57]: Text(0, 0.5, 'Average Energy Error (MeV)')




[62]:
```
def listSubtract(list1, list2):
    result = []

    for x, y in zip(list1, list2):
        difference = x - y
        difference = abs(difference)
        result.append(difference)


    return result
```

[69]:
```
#Compare train and test MSE errors on the max energy

fig = plt.figure()
#ax1 = fig.add_subplot(1,1,1)

maxMSEDiff = listSubtract(trainMaxMSE, maxEnergyMSE)
```

```
plt.plot(epochList, maxMSEDiff, c='b', marker="s")

plt.title("Max Energy Error MSE Difference Between Training and Testing", pad =␣
↪20)
plt.xlabel("Number of epochs")
plt.ylabel("Error Difference")
#plt.legend(loc='upper left')
plt.show()
```




[67]:
```
#Compare train and test percent errors on the max energy

fig = plt.figure()
#ax1 = fig.add_subplot(1,1,1)

maxPercentDiff = listSubtract(trainMaxPercent, maxEnergyPercent)

plt.plot(epochList, maxPercentDiff, c='b', marker="s")

plt.title("Max Energy Relative Error Difference Between Training and Testing")
plt.xlabel("Number of epochs")
plt.ylabel("Error Difference")
```

```
#plt.legend(loc='upper left')
plt.show()
```


[48]:
```
for epoch, maxError, totalError, avgError in zip(epochList, maxEnergyMSE,
↪totalEnergyMSE, avgEnergyMSE):
    print("Number of epochs:", epoch)
    print("Max energy MSE:", maxError)
    print("Total energy MSE:", totalError)
    print("Average energy MSE:", avgError, '\n')
```

```
Number of epochs: 1
Max energy MSE: 7.90678981275435e-05
Total energy MSE: 6.628874470347595e-15
Average energy MSE: 8.270581280748943e-07

Number of epochs: 5
Max energy MSE: 2.56097509304956e-05
Total energy MSE: 1.3159836973675904e-15
Average energy MSE: 3.052881171627591e-07

Number of epochs: 10
Max energy MSE: 5.843151638475434e-06
Total energy MSE: 1.0487468404820815e-15
Average energy MSE: 1.3450930758344612e-07
```

```
Number of epochs: 15
Max energy MSE: 4.573764136075536e-05
Total energy MSE: 8.539439887026031e-15
Average energy MSE: 1.043652028339493e-06

Number of epochs: 20
Max energy MSE: 1.2225174624579374e-05
Total energy MSE: 1.4019149672236502e-15
Average energy MSE: 2.0449766341159457e-07

Number of epochs: 25
Max energy MSE: 4.617832026437266e-06
Total energy MSE: 1.1386756715137832e-15
Average energy MSE: 8.176574557675277e-08

Number of epochs: 30
Max energy MSE: 1.8398282073741432e-05
Total energy MSE: 2.738247805600269e-15
Average energy MSE: 3.1320008596527395e-07

Number of epochs: 35
Max energy MSE: 2.7830366415829203e-05
Total energy MSE: 5.691832001998356e-15
Average energy MSE: 4.916228436796306e-07

Number of epochs: 40
Max energy MSE: 6.417194722972398e-06
Total energy MSE: 8.916700181568558e-16
Average energy MSE: 1.455500193902616e-07

Number of epochs: 45
Max energy MSE: 8.540746332391318e-06
Total energy MSE: 1.47539524937632e-15
Average energy MSE: 1.2596657171892955e-07

Number of epochs: 50
Max energy MSE: 1.6168281817421286e-05
Total energy MSE: 3.403557766137718e-15
Average energy MSE: 2.642834794067944e-07

Number of epochs: 55
Max energy MSE: 1.8911618411471807e-05
Total energy MSE: 3.842817341325633e-15
Average energy MSE: 3.124143072395634e-07

Number of epochs: 60
Max energy MSE: 5.27086171034873e-06
```

```
Total energy MSE: 1.3366690771814578e-15
Average energy MSE: 8.476458235106203e-08

Number of epochs: 65
Max energy MSE: 1.3720378880948688e-05
Total energy MSE: 1.6251378711588333e-15
Average energy MSE: 2.0060483950655062e-07

Number of epochs: 70
Max energy MSE: 2.025310256572672e-05
Total energy MSE: 3.009400704078984e-15
Average energy MSE: 3.2010991889383347e-07

Number of epochs: 75
Max energy MSE: 1.8158031675834693e-05
Total energy MSE: 3.744467968609542e-15
Average energy MSE: 3.081494290984527e-07

Number of epochs: 80
Max energy MSE: 7.07717333755127e-05
Total energy MSE: 1.2551130044070479e-14
Average energy MSE: 1.223065524918084e-06

Number of epochs: 85
Max energy MSE: 2.804888512250297e-05
Total energy MSE: 5.851463281155415e-15
Average energy MSE: 4.4407015455334704e-07

Number of epochs: 90
Max energy MSE: 4.901962701589263e-05
Total energy MSE: 6.330414435933989e-15
Average energy MSE: 9.035084792530954e-07

Number of epochs: 100
Max energy MSE: 2.107425527150117e-05
Total energy MSE: 4.9675397124597325e-15
Average energy MSE: 4.2803590030569e-07

Number of epochs: 150
Max energy MSE: 6.050833718744156e-05
Total energy MSE: 8.351028222649266e-15
Average energy MSE: 9.87727346500104e-07

Number of epochs: 200
Max energy MSE: 0.00010443767923438354
Total energy MSE: 1.3065264444251528e-14
Average energy MSE: 1.7464690807542189e-06
```

```
Number of epochs: 250
Max energy MSE: 3.848314989036754e-05
Total energy MSE: 5.206516255650622e-15
Average energy MSE: 6.201112748931364e-07
```

[25]:

```

```