Model Tuning

June 1, 2022

[1]:
```python
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]:
```python
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

[8]:
```
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+')
```

[ ]:
```
#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]

print(npFile.shape)
```

3 Prepare our dataset

[10]:
```
training_dataset = h5File.create_dataset(name=None, data=npTrain)
test_dataset = h5File.create_dataset(name=None, data=npTest)
```

[11]:
```
#Choose our loss function

loss_function = nn.MSELoss()
```

[12]:
```
#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]
```

[ ]:
```
#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)
```

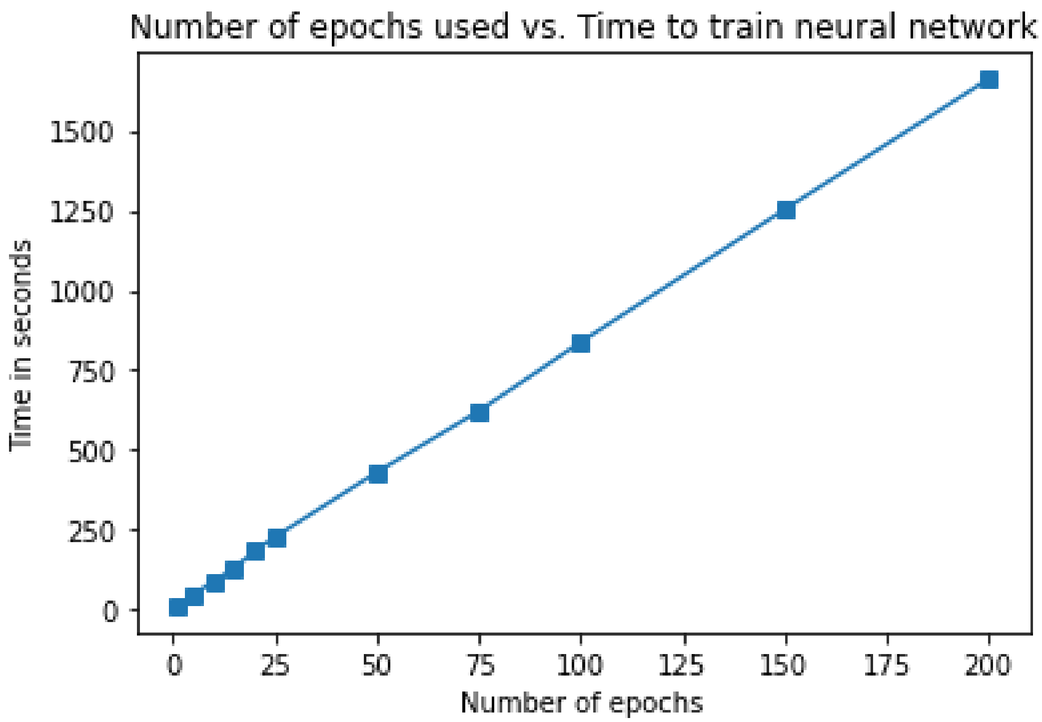
[14]:
```
def splitErrorList(errorList):
    maxEnergyError = []
    totalEnergyError = []
    avgEnergyError = []

    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

[48]:
```
%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()
```



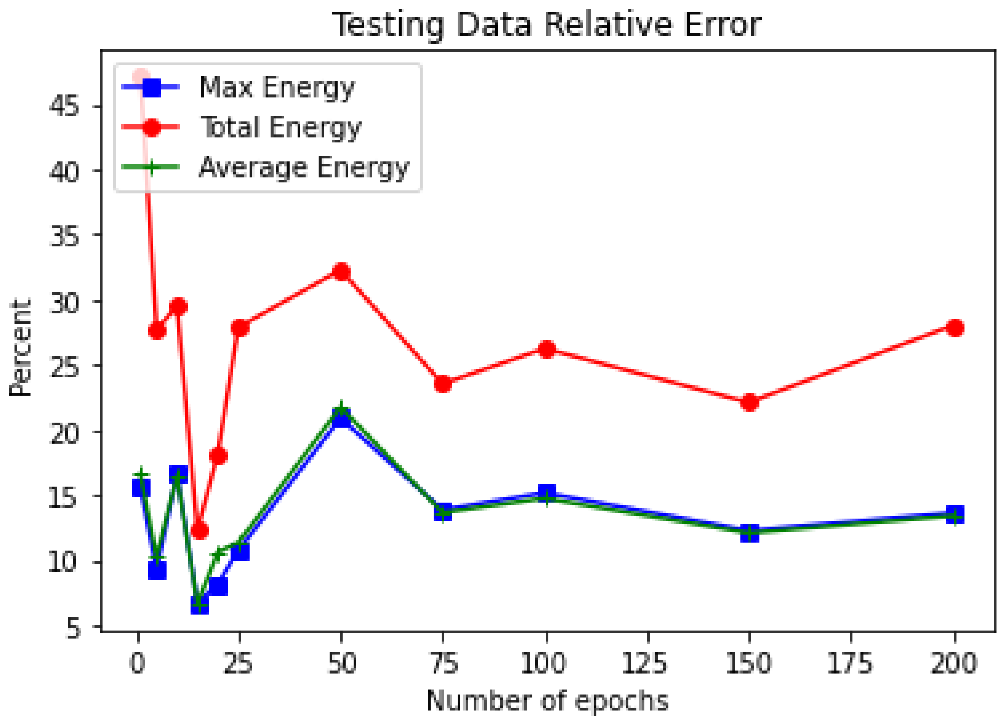
[50]:
```
for epochElement, timeElement in zip(epochList, timeList):

    minuteValue = timeElement / 60

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

```
Number of epochs: 1
Time spent: 0.20063339471817015 minutes

Number of epochs: 5
Time spent: 0.7759333491325379 minutes
```

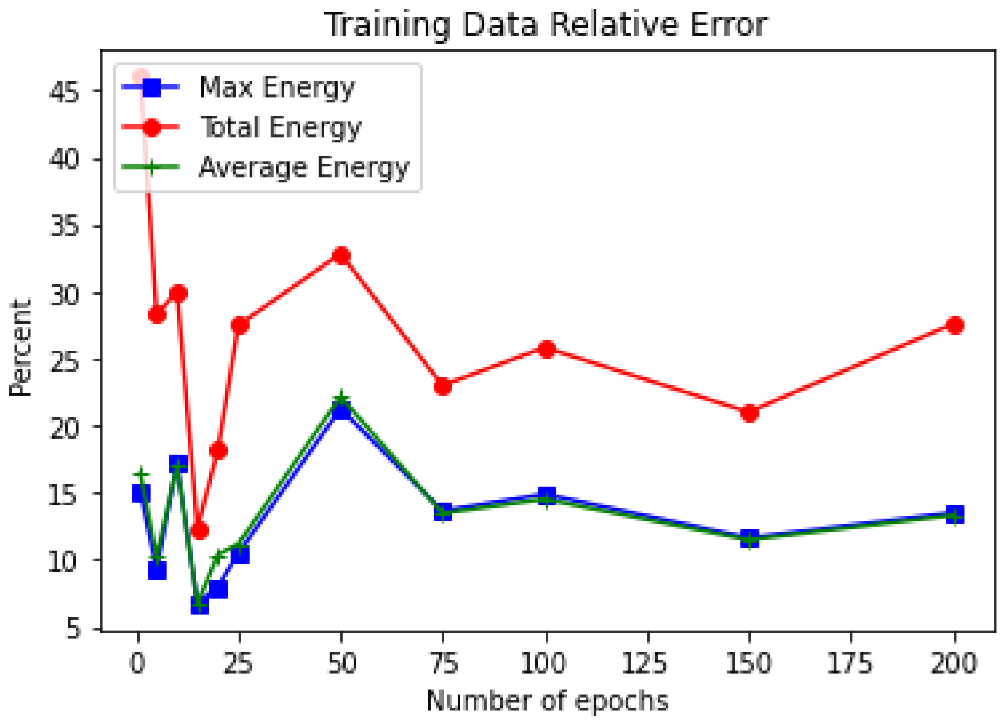
```
Number of epochs: 10
Time spent: 1.4926666935284933 minutes

Number of epochs: 15
Time spent: 2.164199980099996 minutes

Number of epochs: 20
Time spent: 3.0516333977381387 minutes

Number of epochs: 25
Time spent: 3.78549298842748 minutes

Number of epochs: 50
Time spent: 7.170249978701274 minutes

Number of epochs: 75
Time spent: 10.36313331524531 minutes

Number of epochs: 100
Time spent: 13.951545910040538 minutes

Number of epochs: 150
Time spent: 20.878193310896556 minutes

Number of epochs: 200
Time spent: 27.690767661730447 minutes
```

[46]:

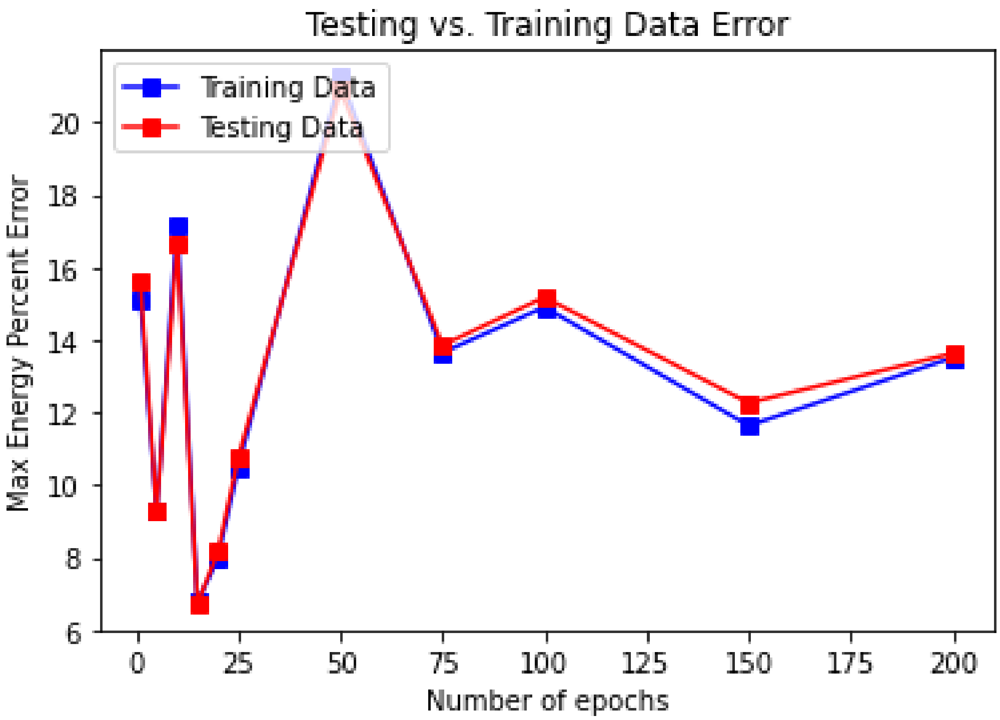
```
#Percent Error plot
fig = plt.figure()
ax1 = fig.add_subplot(1,1,1)


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 left');
plt.show()
```



[45]:
```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 left');
plt.show()
```



[33]:
```
#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 left')
plt.show()
```



[21]:
```
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: 15.598560588172345
Total energy percent error: 47.124909053003016
Average energy percent error: 16.59115917077054

Number of epochs: 5
Max energy percent error: 9.26801726744719
Total energy percent error: 27.692389026009653
Average energy percent error: 10.253590384507486

Number of epochs: 10
Max energy percent error: 16.614030091514753
Total energy percent error: 29.522811426012172
Average energy percent error: 16.454574957292795

Number of epochs: 15
Max energy percent error: 6.721553119876682
```

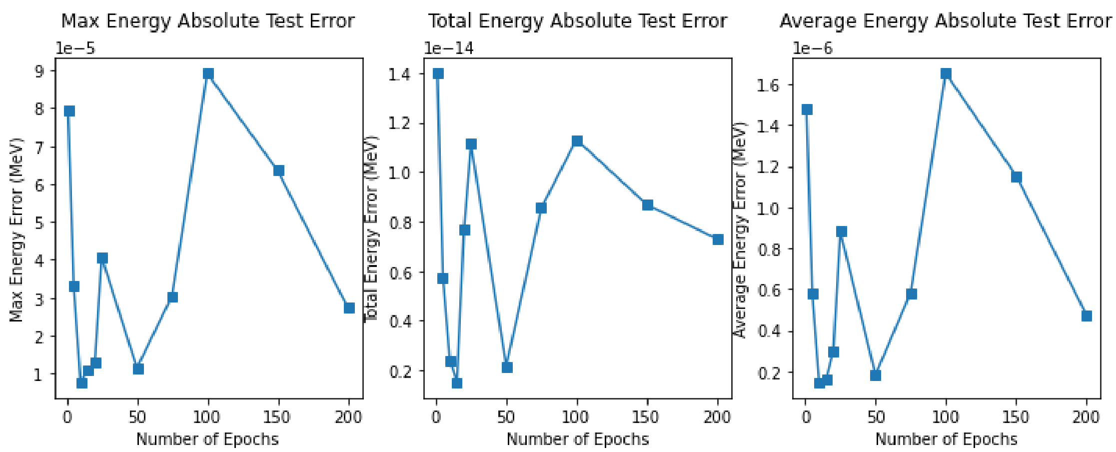
```
Total energy percent error: 12.283105957943928
Average energy percent error: 6.644414430812231

Number of epochs: 20
Max energy percent error: 8.153420619284182
Total energy percent error: 18.1708747179025
Average energy percent error: 10.619825973180802

Number of epochs: 25
Max energy percent error: 10.749034568917663
Total energy percent error: 27.877116218303698
Average energy percent error: 11.340451394236554

Number of epochs: 50
Max energy percent error: 20.89389093393007
Total energy percent error: 32.28915718973411
Average energy percent error: 21.74327586777992

Number of epochs: 75
Max energy percent error: 13.8519135505585
Total energy percent error: 23.53603350619211
Average energy percent error: 13.651349608178377

Number of epochs: 100
Max energy percent error: 15.170820979970525
Total energy percent error: 26.253288932755865
Average energy percent error: 14.740454818279206

Number of epochs: 150
Max energy percent error: 12.25809513808918
Total energy percent error: 22.097661064607696
Average energy percent error: 12.083517918488942

Number of epochs: 200
Max energy percent error: 13.623188369637374
Total energy percent error: 28.02565265702911
Average energy percent error: 13.392833297569197
```

[52]:

```
fig = plt.figure(figsize = (12, 4))

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

```python
plt.subplot(1, 3, 2)
plt.plot(epochList, totalEnergyMSE, marker = 's')
plt.title("Total Energy Absolute Test 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)')
```

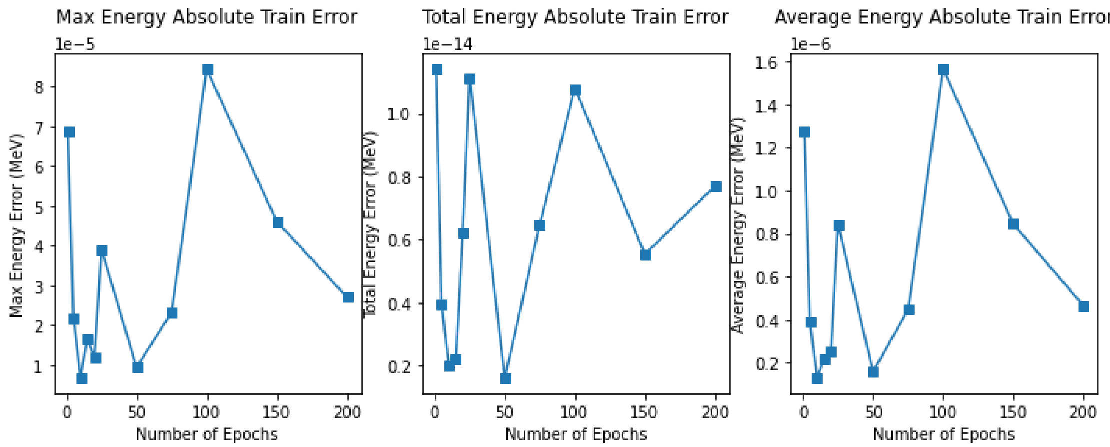
[52]: Text(0, 0.5, 'Average Energy Error (MeV)')



[51]:
```python
fig = plt.figure(figsize = (12, 4))

plt.subplot(1, 3, 1)
plt.plot(epochList, trainMaxMSE, marker='s')
plt.title("Max Energy Absolute Train 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 Absolute Train 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)')
```

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



[24]:
```
# Results with 20,000 points

# Number of epochs: 1
# Time spent: 9.709996938705444

# Number of epochs: 5
# Time spent: 42.58600425720215

# Number of epochs: 10
# Time spent: 86.28900504112244

# Number of epochs: 15
# Time spent: 129.9060001373291

# Number of epochs: 20
# Time spent: 172.23799514770508

# Number of epochs: 50
# Time spent: 436.66264843940735

# Number of epochs: 1
# Max energy percent error: 14.266001682042852
# Total energy percent error: 36.2113883322816
# Average energy percent error: 14.382154573345156

# Number of epochs: 5
```

```
# Max energy percent error: 13.186282608741399
# Total energy percent error: 20.86685418910202
# Average energy percent error: 11.821664805706583

# Number of epochs: 10
# Max energy percent error: 17.19713317728656
# Total energy percent error: 23.92806554969021
# Average energy percent error: 12.881195411127187

# Number of epochs: 15
# Max energy percent error: 19.317774507765645
# Total energy percent error: 28.18222565512904
# Average energy percent error: 16.43670669838012

# Number of epochs: 20
# Max energy percent error: 25.952888453939533
# Total energy percent error: 39.46280229124166
# Average energy percent error: 25.0593774302352

# Number of epochs: 50
# Max energy percent error: 24.551931096353997
# Total energy percent error: 34.28238226862895
# Average energy percent error: 20.571137202898512
```

[ ]:

```

```