```
# Using google colab - this first step is for loading in the data from my personal Dri

# Login with google credentials

from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

# Handle errors from too many requests

import logging
logging.getLogger('googleapiclient.discovery_cache').setLevel(logging.ERROR)

# The ID for my personal Drive folder is 1BVUuroPvozFxMjMIYrGOFtI4r6erSBCx
# I am now listing the ID numbers for the files in this folder to find the data files

#file_list = drive.ListFile({'q': "'1BVUuroPvozFxMjMIYrGOFtI4r6erSBCx' in parents and
#for file1 in file_list:
#  print('title: %s, id: %s' % (file1['title'], file1['id']))

# Data ID: 1F2KojI0d-ZnN8ssQFUWSyZA8I0mAgMEf

# Now that I have the ID files, load the files

data_downloaded = drive.CreateFile({'id': '1dwQLnIskShTXwSeMONhu__bYFf_f8-t6'})
data_downloaded.GetContentFile('sc_train.csv')

data_downloaded = drive.CreateFile({'id': '1IcNFIYUDKz1UxFL8W_JNjz9TzjAlAOVa'})
data_downloaded.GetContentFile('sc_unique_m.csv')



# Load the data into pandas

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import io

train = pd.read_csv('sc_train.csv',low_memory=False, lineterminator='\n')
unique = pd.read_csv('sc_unique_m.csv',low_memory=False, lineterminator='\n')


print(unique.shape)
print(train.shape)
```

```
(21263, 88)
(21263, 82)
```

```
print(unique.head(5))
print(train.head(5))
```

```
     H  He   Li   Be    B  ...  Po  At  Rn  critical_temp                material
0  0.0   0  0.0  0.0  0.0  ...   0   0   0           29.0          Ba0.2La1.8Cu1O4
1  0.0   0  0.0  0.0  0.0  ...   0   0   0           26.0   Ba0.1La1.9Ag0.1Cu0.9O4
2  0.0   0  0.0  0.0  0.0  ...   0   0   0           19.0          Ba0.1La1.9Cu1O4
3  0.0   0  0.0  0.0  0.0  ...   0   0   0           22.0        Ba0.15La1.85Cu1O4
4  0.0   0  0.0  0.0  0.0  ...   0   0   0           23.0          Ba0.3La1.7Cu1O4

[5 rows x 88 columns]
   number_of_elements  mean_atomic_mass  ...  wtd_std_Valence  critical_temp\r
0                   4         88.944468  ...         0.437059             29.0
1                   5         92.729214  ...         0.468606             26.0
2                   4         88.944468  ...         0.444697             19.0
3                   4         88.944468  ...         0.440952             22.0
4                   4         88.944468  ...         0.428809             23.0

[5 rows x 82 columns]
```

```
# merge the two dataframes, drop material string
merge_df = pd.concat([train, unique], axis=1, sort=False)
merge_df = merge_df.drop(['material\r'], axis=1)
# Create feature identifying high-temp superconductors
merge_df['is_highTc'] = merge_df['critical_temp'] > 73

high_Tc_df = merge_df[merge_df['is_highTc']]


# drop outlier
merge_df = merge_df[merge_df['critical_temp'] < 180]
#normalize
merge_df = (merge_df-merge_df.min())/(merge_df.max()-merge_df.min())
# fix any NA values created by division by zero
merge_df = merge_df.fillna(0)


#drop cols with one value
for col in merge_df.columns:
    if len(merge_df[col].unique()) == 1:
        merge_df.drop(col,inplace=True,axis=1)


# Create correlation matrix

features = list(merge_df.columns.values.tolist())
corrMat = merge_df[features].corr().abs()

# Select upper triangle of correlation matrix
upper = corrMat.where(np.triu(np.ones(corrMat.shape), k=1).astype(np.bool))
```

```
# Find index of feature columns with correlation greater than 0.95
to_drop = [column for column in upper.columns if any(upper[column] > 0.5)]

# make sure I don't drop my target variables
if 'critical_temp' in to_drop: to_drop.remove('critical_temp')
if 'is_highTc' in to_drop: to_drop.remove('is_highTc')


print(len(to_drop)) # 55
#to_drop

merge_df = merge_df.drop(merge_df[to_drop], axis=1)
```

```
 86
```

```
import seaborn as sn

features = list(merge_df.columns.values.tolist())
corrMat = merge_df[features].corr().abs()

plt.figure(figsize=(20,10))
#sn.heatmap(corrMat, annot=True)
```

```
<Figure size 1440x720 with 0 Axes>
<Figure size 1440x720 with 0 Axes>
```

```
import torch

class TwoLayerNet(torch.nn.Module):

  def __init__(self, D_in, H, D_out):
    super(TwoLayerNet, self).__init__()
    self.linear1 = torch.nn.Linear(D_in, H)
    #self.leaky1 = torch.nn.LeakyReLU(H, 30)
    self.drop1 = torch.nn.Dropout(p = 0.4)
    self.linear2 = torch.nn.Linear(H, D_out)
    '''
    self.linear3 = torch.nn.Linear(H, H)
    self.linear4 = torch.nn.Linear(H, H)
    self.linear5 = torch.nn.Linear(H, H)
    '''
  def forward(self, X):
    linear_1 = self.linear1(X)
    #leaky_1 = self.leaky1(linear_1)
    drop_1 = self.drop1(linear_1)
    linear_2 = self.linear2(drop_1)
    h_relu = linear_2.clamp(min=0)

    return h_relu

def MAPELoss(output, target):
```

```
    return 100*torch.mean(torch.abs((target - output) / (target + 0.001)))

def rmse(y, y_hat):

    #combined rmse value
    mse=torch.mean((y-y_hat)**2)
    rmse = torch.sqrt(mse)

    return rmse


## train test split

train_df = merge_df.sample(frac=0.8, random_state=np.random.seed())
test_df = merge_df.drop(train_df.index)

# set up train and test data
X_train = train_df.drop(['critical_temp', 'is_highTc'], axis=1).to_numpy()
X_test = test_df.drop(['critical_temp', 'is_highTc'], axis=1).to_numpy()

X_train_high = train_df[train_df['is_highTc'] == 1].drop(['critical_temp', 'is_highTc'
X_test_high = test_df[test_df['is_highTc'] == 1].drop(['critical_temp', 'is_highTc'],

# set up target variable
y_train = train_df['critical_temp'].to_numpy()
y_test = test_df['critical_temp'].to_numpy()

# Set up alternative target - is high_T SC or not
y_high_temp_train = train_df['is_highTc'].to_numpy()
y_high_temp_test = test_df['is_highTc'].to_numpy()
#convert to Torch

X_torch = torch.from_numpy(X_train).float()
X_torch_high = torch.from_numpy(X_train_high).float()
y_torch = torch.from_numpy(y_train).float()
y_torch_highTC = torch.from_numpy(y_high_temp_train).float()
type(X_torch)

print(sum(sum(torch.isnan(X_torch))))

## No nans
```

```
tensor(0)
```

```
print(y_torch)
```

```
tensor([0.2273, 0.3944, 0.4678,  ..., 0.0187, 0.6308, 0.0269])
```

```
D_in, H, D_out = X_train.shape[1], 5, 1
print(D_in, H, D_out)
```

```
73 5 1
```

```
epochs = 1000
model = TwoLayerNet(D_in, H, D_out)

use_cuda = torch.cuda.is_available()

if use_cuda:

    device = torch.device('cuda:0' if use_cuda else 'cpu')
    model.cuda()
    X_torch = X_torch.to(device)
    y_torch = y_torch.to(device)

criterion = torch.nn.MSELoss(reduction='mean')
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)

for epoch in range(epochs):
    # Forward pass: Compute predicted y by passing x to the model
    y_pred = model(X_torch)
    #print(X_torch)
    #print(y_pred)
    #print(y_torch)
    # Compute and print loss
    loss = criterion(y_pred, y_torch)
    if epoch % 100 == 0:
      print(epoch, loss.item(), rmse(y_pred, y_torch))

    # Zero gradients, perform a backward pass, and update the weights.
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

```
/usr/local/lib/python3.6/dist-packages/torch/nn/modules/loss.py:431: UserWarning:
  return F.mse_loss(input, target, reduction=self.reduction)
0 0.11578492075204849 tensor(0.3403, device='cuda:0', grad_fn=<SqrtBackward>)
100 0.11579353362321854 tensor(0.3403, device='cuda:0', grad_fn=<SqrtBackward>)
200 0.11579088866710663 tensor(0.3403, device='cuda:0', grad_fn=<SqrtBackward>)
300 0.1157890111207962 tensor(0.3403, device='cuda:0', grad_fn=<SqrtBackward>)
400 0.11579207330942154 tensor(0.3403, device='cuda:0', grad_fn=<SqrtBackward>)
500 0.1157846599817276 tensor(0.3403, device='cuda:0', grad_fn=<SqrtBackward>)
600 0.11579275131225586 tensor(0.3403, device='cuda:0', grad_fn=<SqrtBackward>)
700 0.115787073969841 tensor(0.3403, device='cuda:0', grad_fn=<SqrtBackward>)
800 0.11578858643770218 tensor(0.3403, device='cuda:0', grad_fn=<SqrtBackward>)
900 0.11579195410013199 tensor(0.3403, device='cuda:0', grad_fn=<SqrtBackward>)
```

```
y_pred*185
```

```
tensor([[0.],
        [0.],
        [0.],
        ...,
        [0.],
        [0.],
        [0.]], device='cuda:0', grad_fn=<MulBackward0>)
```

```
epochs = 1000
model = TwoLayerNet(D_in, H, D_out)

use_cuda = torch.cuda.is_available()

if use_cuda:
    print("Using GPU!")
    device = torch.device('cuda:0' if use_cuda else 'cpu')
    model.cuda()
    X_torch_high = X_torch_high.to(device)
    y_torch_highTC = y_torch_highTC.to(device)
else:
  print("Using CPU!")

criterion = torch.nn.MSELoss(reduction='mean')
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)

for epoch in range(epochs):
    # Forward pass: Compute predicted y by passing x to the model
    y_pred = model(X_torch_high)
    #print(X_torch)
    #print(y_pred)
    #print(y_torch)
    # Compute and print loss
    loss = criterion(y_pred, y_torch_highTC)
    if epoch % 100 == 0: print(epoch, loss.item())

    # Zero gradients, perform a backward pass, and update the weights.
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

X_test_torch = torch.from_numpy(X_test_high).float()
y_test_torch_highTC = torch.from_numpy(y_high_temp_test).float()

print("Test set!")

if use_cuda:
    print("Using GPU!")
    device = torch.device('cuda:0' if use_cuda else 'cpu')
    model.cuda()
    X_test_torch = X_test_torch.to(device)
    y_test_torch_highTC = y_test_torch_highTC.to(device)
```

```
else:
  print("Using CPU!")

test_preds = model(X_test_torch)
testLoss = criterion(test_preds, y_test_torch_highTC)
print(testLoss.item())
```

```
Using GPU!
0 0.16562584042549133
/usr/local/lib/python3.6/dist-packages/torch/nn/modules/loss.py:431: UserWarning:
  return F.mse_loss(input, target, reduction=self.reduction)
100 0.16534408926963806
200 0.16521555185317993
300 0.16513268649578094
400 0.16508185863494873
500 0.16504457592964172
600 0.1650126576423645
700 0.16499842703342438
800 0.16498292982578278
900 0.16499193012714386
Test set!
Using GPU!
0.15887029469013214
/usr/local/lib/python3.6/dist-packages/torch/nn/modules/loss.py:431: UserWarning:
  return F.mse_loss(input, target, reduction=self.reduction)
```

```
y_test_torch_highTC
```

```
tensor([0., 0., 0.,  ..., 0., 0., 1.], device='cuda:0')
```