24-787: Machine Learning and Artificial Intelligence for Engineers
Ryan Wu
ID: weihuanw
Homework 9
Due: Mar 30, 2024

Concept Questions:

Problem 1

Problem 1

Given: ground truth vector $y = [-4, 8, 7, -15, 12]$, $n = 5$

prediction vector $\hat{y} = [2, 9, -1, -16, 18]$

Find: MAE, MSE, MAPE

Equations: $MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$, $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$, $MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$

Solutions:

[MAE] $MAE = \frac{1}{5}[|2-(-4)| + |(9-8)| + |(-1-7)| + |(-16-(-15)| + |(18-12)|]$

$MAE = \frac{1}{5}(6+1+8+1+6) \rightarrow MAE = \frac{1}{5}(22) \rightarrow$ **MAE = 4.4 #**

[MSE] $MSE = \frac{1}{5}[(-4-2)^2 + (8-9)^2 + (7-(-1))^2 + (-15-(-16))^2 + (12-18)^2]$

$MSE = \frac{1}{5}(36+1+64+1+36) \rightarrow MSE = \frac{1}{5}(138) \rightarrow$ **MSE = 27.6 #**

[MAPE] $MAPE = \frac{1}{5}\left[\left|\left(\frac{-4-2}{-4}\right)\right| + \left|\left(\frac{8-9}{8}\right)\right| + \left|\left(\frac{7-(-1)}{7}\right)\right| + \left|\left(\frac{-15-(-16)}{-15}\right)\right| + \left|\left(\frac{12-18}{12}\right)\right|\right]$

$MAPE = \frac{1}{5}\left(\frac{3}{2} + \frac{1}{8} + \frac{8}{7} + \frac{1}{15} + \frac{1}{2}\right) \rightarrow$ **MAPE ≈ 0.6669 #**

Problem 2
Matrix 3.

Problem 3
4. (8, 10, 0, 2, 0.8, 1.0, 0.889)

M9-L1 Problem 1

Here, you will implement three loss functions from scratch in numpy: MAE, MSE, and MAPE.

```
import numpy as np

y_gt1 = np.array([1,2,3,4,5,6,7,8,9,10])
y_pred1 = np.array([1,1.3,3.1,4.6,5.9,5.9,6.4,9.2,8.1,10.5])

y_gt2 = np.array([-3.23594765, -3.74125693, -2.3040903 ,  0.        ,
0.30190142, -1.68434859,  1.10160357,  0.8587438 ,  1.76546802,
3.13787123, 3.72990216,  5.89871795,  6.06406803,  6.28329118,
7.46406525, 8.21246221, 10.23145281,  9.39080133, 10.76761316,
10.45903557, 9.61872736, 13.68392163, 14.75332509, 14.00530973,
17.87581523, 15.01028079, 17.36899084, 17.99463433, 20.57318325,
21.36834867, 20.91252318, 21.99432414, 21.58696173, 21.35253687,
23.84400704, 25.20685402, 27.13938159, 27.97005662, 27.23893581,
28.18254573, 28.29488138, 28.78200226, 29.35433587, 33.86996731,
32.2681256 , 33.19828933, 33.24215413, 36.13102571, 34.59822336,
36.85796679, 37.03382637, 39.17478129, 39.13565951, 39.32441832,
41.33545414, 42.65055409, 43.1473253 , 44.24186584, 44.1636577 ,
45.29382449, 45.84269107, 47.01418421, 47.41917695, 47.36462649,
50.12692109, 50.40629987, 50.03646832, 52.98803478, 52.47654002,
54.29436964, 55.83010066, 56.08857887, 57.9575825 , 56.44194186,
58.93769518, 58.7091293 , 59.3817281 , 60.53226145, 61.65814444,
62.88444817, 62.52171885, 65.44628103, 65.86970284, 64.72638258,
68.60946432, 69.87568716, 70.01716341, 69.51704486, 69.48480293,
72.46859314, 71.86955033, 74.3537582 , 74.19817397, 75.82512388,
76.0634371 , 77.27222973, 77.43474244, 80.06869878, 79.26832623,
80.40198936])
y_pred2 = np.array([-3.17886560e+00, -3.72628642e+00, -2.28154027e+00,
-2.42424242e-06, 2.96261368e-01, -1.70080838e+00,  1.09113641e+00,
8.60043722e-01, 1.76729042e+00,  3.12498677e+00,  3.72452933e+00,
5.81293300e+00, 6.01791742e+00,  6.27564586e+00,  7.43093457e+00,
8.18505900e+00, 1.00785853e+01,  9.41006754e+00,  1.07339029e+01,
1.05483666e+01, 9.86429504e+00,  1.35944803e+01,  1.46257911e+01,
1.41092530e+01, 1.74700758e+01,  1.52285866e+01,  1.73610430e+01,
1.80283176e+01, 2.02578402e+01,  2.10543695e+01,  2.08801196e+01,
2.19111495e+01, 2.17786086e+01,  2.17754891e+01,  2.39269636e+01,
2.51674432e+01, 2.68054871e+01,  2.76337491e+01,  2.73444399e+01,
2.82677426e+01, 2.85915692e+01,  2.91907133e+01,  2.98552019e+01,
3.32092384e+01, 3.24325813e+01,  3.33437229e+01,  3.36586115e+01,
3.58501097e+01, 3.51566050e+01,  3.69363787e+01,  3.73654528e+01,
3.90232127e+01, 3.93355670e+01,  3.97886962e+01,  4.13471034e+01,
4.24678677e+01, 4.31186248e+01,  4.41080463e+01,  4.44437982e+01,
4.54581242e+01, 4.61509657e+01,  4.71832256e+01,  4.78047650e+01,
4.81822755e+01, 5.00379827e+01,  5.06088232e+01,  5.08521636e+01,
```

```
5.27428151e+01, 5.29526597e+01,  5.42661662e+01,  5.54230479e+01,
5.60162341e+01, 5.72972123e+01,  5.71389028e+01,  5.87005639e+01,
5.91111760e+01, 5.98988234e+01,  6.08826528e+01,  6.18502423e+01,
6.28491288e+01, 6.32501917e+01,  6.48567227e+01,  6.55629719e+01,
6.57207391e+01, 6.75883810e+01,  6.85509197e+01,  6.91918142e+01,
6.96421235e+01, 7.02288144e+01,  7.17044458e+01,  7.21593122e+01,
7.34448231e+01, 7.40436375e+01,  7.50845851e+01,  7.57923722e+01,
7.67262442e+01, 7.74266118e+01,  7.86387737e+01,  7.91677250e+01,
8.00787815e+01])
```

Mean Absolute Error

Complete the definition for `MAE(y_gt, y_pred)` below.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \hat{y}_i\right| = \frac{1}{n}\sum_{i=1}^{n}\left|e_i\right|$$

`MAE(y_gt1, y_pred1)` should return 0.560.

```
def MAE(y_gt, y_pred):
    # YOUR CODE GOES HERE
    return np.mean(np.abs(y_gt - y_pred))

print(f"MAE(y_gt1, y_pred1) = {MAE(y_gt1, y_pred1):.3f}")
print(f"MAE(y_gt2, y_pred2) = {MAE(y_gt2, y_pred2):.3f}")
```

```
MAE(y_gt1, y_pred1) = 0.560
MAE(y_gt2, y_pred2) = 0.290
```

Mean Squared Error

Complete the definition for `MSE(y_gt, y_pred)` below.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2 = \frac{1}{n}\sum_{i=1}^{n}\left(e_i\right)^2 = \frac{1}{n}e^T e$$

`MSE(y_gt1, y_pred1)` should return 0.454.

```
def MSE(y_gt, y_pred):
    # YOUR CODE GOES HERE
    return np.mean((y_gt - y_pred)**2)

print(f"MSE(y_gt1, y_pred1) = {MSE(y_gt1, y_pred1):.3f}")
print(f"MSE(y_gt2, y_pred2) = {MSE(y_gt2, y_pred2):.3f}")
```

```
MSE(y_gt1, y_pred1) = 0.454
MSE(y_gt2, y_pred2) = 0.174
```

Mean Absolute Percentage Error

Complete the definition for `MAPE(y_gt, y_pred, epsilon)` below.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\frac{\left|y_i - \hat{y}_i\right)}{\left|y_i\right)+\varepsilon} = \frac{1}{n}\sum_{i=1}^{n}\frac{\left|e_i\right)}{\left|y_i\right)+\varepsilon}$$

`MAPE(y_gt1, y_pred1, 1e-6)` should return 0.112.

```
def MAPE(y_gt, y_pred, epsilon=1e-6):
    # YOUR CODE GOES HERE
    return np.mean(np.abs((y_gt - y_pred) / (y_gt + epsilon)))

print(f"MAPE(y_gt1, y_pred1) = {MAPE(y_gt1, y_pred1):.3f}")
print(f"MAPE(y_gt2, y_pred2) = {MAPE(y_gt2, y_pred2):.3f}")
```

```
MAPE(y_gt1, y_pred1) = 0.112
MAPE(y_gt2, y_pred2) = 0.032
```

M9-L1 Problem 2

Recall the von Mises stress prediction problem from the module 6 homework. In this problem, you will compute the R^2 score for a few model predictions for a single shape in this dataset. You will also plot the predicted-vs-actual stress for each model.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score

float32 = np.float32

xs= np.load("data/L1P2/xs.npy")
ys= np.load("data/L1P2/ys.npy")
gt = np.load("data/L1P2/gt.npy")
model1= np.load("data/L1P2/model1.npy")
model2= np.load("data/L1P2/model2.npy")
model3= np.load("data/L1P2/model3.npy")
```

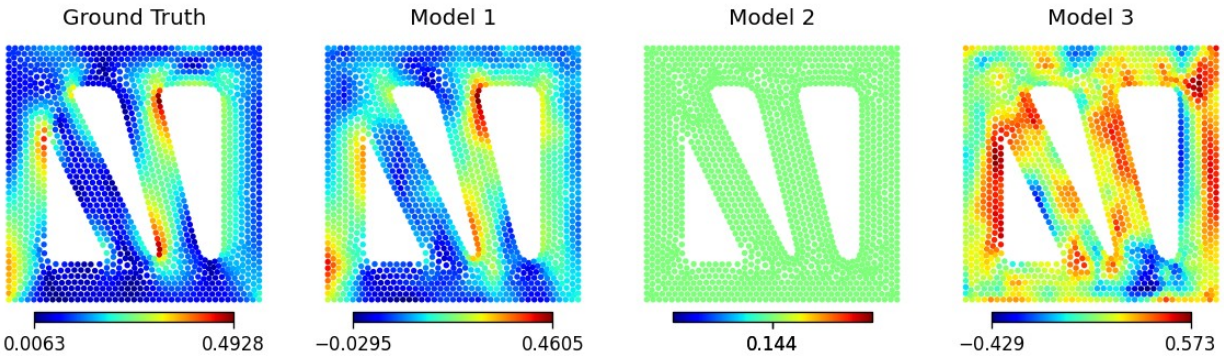
Visualizing data

Run the following cell to load the data and visualize the 3 model predictions.

- `gt` is the ground truth von Mises stress vector
- `model1` is the vector of stress predictions for model 1
- `model2` is the vector of stress predictions for model 2
- `model3` is the vector of stress predictions for model 3

```
def plot_shape(x, y, stress, lims=None):

    if lims is None:
        lims = [min(stress),max(stress)]

    plt.scatter(x,y,s=5,c=stress,cmap="jet",vmin=lims[0],vmax=lims[1])
    plt.colorbar(orientation="horizontal", shrink=.75,
pad=0,ticks=lims)
    plt.axis("off")
    plt.axis("equal")

def plot_all(x, y, gt, model1, model2, model3):
    plt.figure(figsize=[12,3.2], dpi=120)
    plt.subplot(141)
    plot_shape(x, y, gt)
    plt.title("Ground Truth")

    plt.subplot(142)
    plot_shape(x, y, model1)
    plt.title("Model 1")
```

```
    plt.subplot(143)
    plot_shape(x, y, model2)
    plt.title("Model 2")

    plt.subplot(144)
    plot_shape(x, y, model3)
    plt.title("Model 3")

    plt.show()

plot_all(xs, ys, gt, model1, model2, model3)
```



Computing R^2

Calculate the R^2 value for each model and print the results.

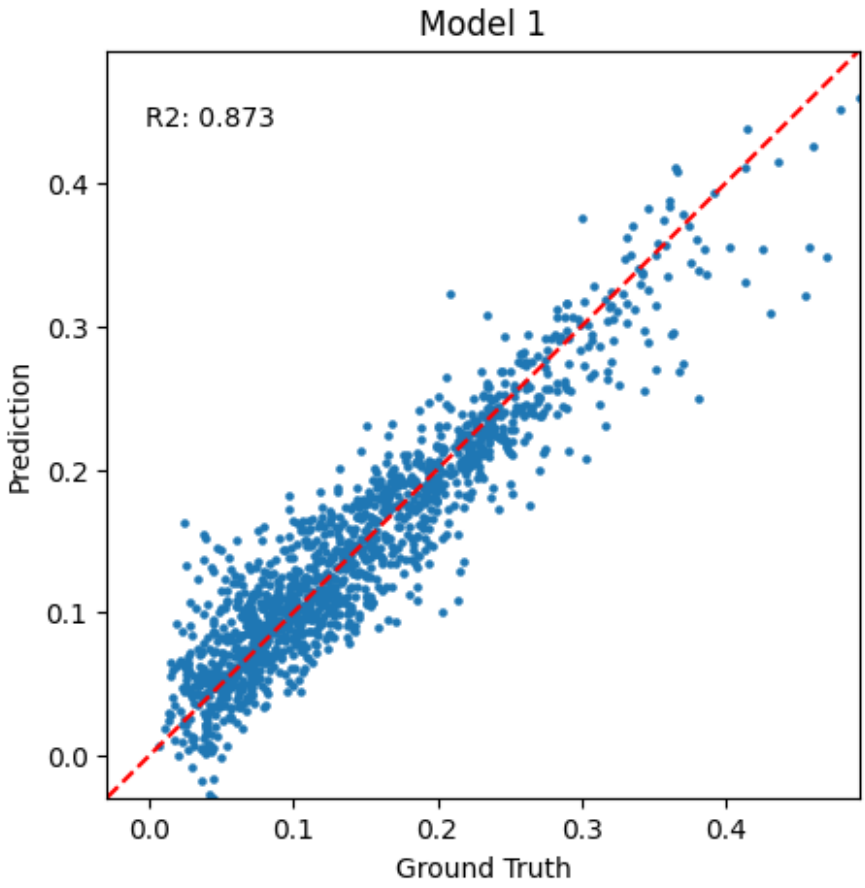
```
# YOUR CODE GOES HERE

models = [model1, model2, model3]

for model in models:
    r2_gt_model = r2_score(gt, model)
    print(f"R2 score: {r2_gt_model}")

# r2_gt_model1 = r2_score(gt, model1)
# r2_gt_model2 = r2_score(gt, model2)
# r2_gt_model3 = r2_score(gt, model3)

R2 score: 0.8727994044645364
R2 score: 0.0
R2 score: -3.0451391287323784
```

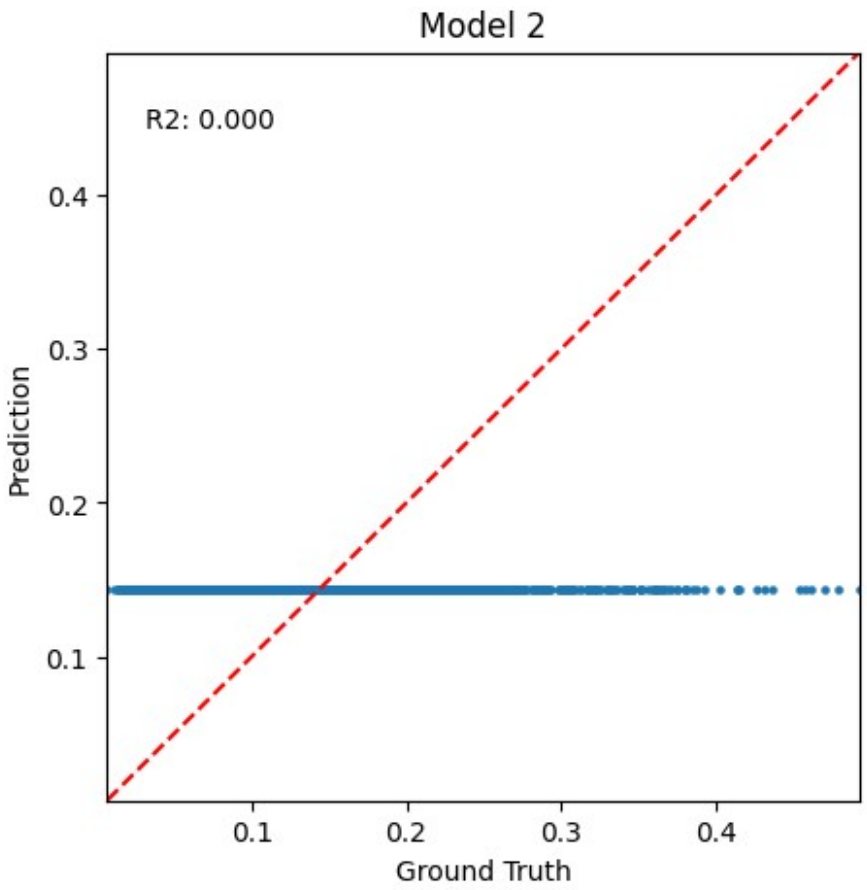
Plotting predictions vs ground truth

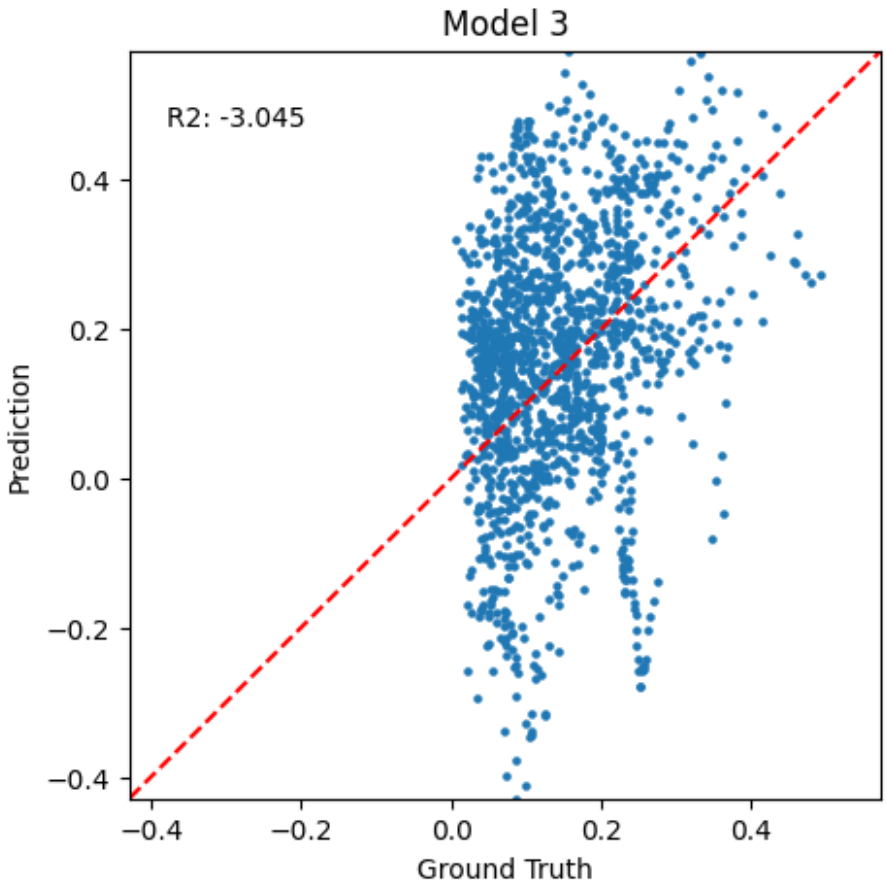
Complete the function definition below for `plot_r2(gt, pred, title)`

Then create plots for all 3 models.

```
def plot_r2(gt, pred, title):
    plt.figure(figsize=[5,5])

    # YOUR CODE GOES HERE
    plt.scatter(gt, pred, s=5)
    plt.text(0.05, 0.9, f"R2: {r2_score(gt, pred):.3f}",
transform=plt.gca().transAxes)


    plt.plot([-1000,1000], [-1000,1000],"r--")

    all = np.concatenate([gt, pred])
    plt.xlim(np.min(all), np.max(all))
    plt.ylim(np.min(all), np.max(all))
    plt.xlabel("Ground Truth")
    plt.ylabel("Prediction")
    plt.title(title)
    plt.show()

plot_r2(gt, model1,"Model 1")
plot_r2(gt, model2,"Model 2")
plot_r2(gt, model3,"Model 3")
```




Model 2
R2: 0.000
Prediction
Ground Truth
0.1
0.2
0.3
0.4



Questions

1. Model 2 has an R^2 of exactly 0. Why?

   The total sum of squares is equal to the sum of squared residuals. The model's predicted value is equal to the ground truth value.

2. Model 3 has an R^2 less than 0. What does this mean?

   It implies that the predicted machine learning model fits the data worse than the mean ground truth value. The model is incorrectly defined and thus an unreliable model.

Problem 1:

Once again consider the plane-strain compression problem shown in "data/plane-strain.png". In this problem you are given node features for 100 parts. These node features have been extracted by processing each part shape using a neural network. You will train a neural network to von Mises stress at each node given its 60 features. Then you will analyze R^2 for the training and testing data, both for the full dataset and for individual shapes within each dataset.

Summary of deliverables

- Neural network model definition
- Training function
- Training loss curve
- Overall R^2 on training and testing data
- Predicted-vs-actual plots for training and testing data
- Histograms of R^2 distributions on training and testing shapes
- Median R^2 values across training and testing shapes

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score

import torch
from torch import nn, optim

def plot_shape(dataset, index, model=None, lims=None):
    x = dataset["coordinates"][index][:,0]
    y = dataset["coordinates"][index][:,1]

    if model is None:
        c = dataset["stress"][index]
    else:
        c = model(torch.tensor(dataset["features"]
[index])).detach().numpy().flatten()

    if lims is None:
        lims = [min(c),max(c)]

    plt.scatter(x,y,s=5,c=c,cmap="jet",vmin=lims[0],vmax=lims[1])
    plt.colorbar(orientation="horizontal", shrink=.75,
pad=0,ticks=lims)
    plt.axis("off")
    plt.axis("equal")

def plot_shape_comparison(dataset, index, model, title=""):
    plt.figure(figsize=[6,3.2], dpi=120)
    plt.subplot(1,2,1)
```

```
    plot_shape(dataset,index)
    plt.title("Ground Truth",fontsize=9,y=.96)
    plt.subplot(1,2,2)
    c = dataset["stress"][index]
    plot_shape(dataset, index, model, lims = [min(c), max(c)])
    plt.title("Prediction",fontsize=9,y=.96)
    plt.suptitle(title)
    plt.show()

def load_dataset(path):
    dataset = np.load(path)
    coordinates = []
    features = []
    stress = []
    N = np.max(dataset[:,0].astype(int)) + 1
    split = int(N*.8)
    for i in range(N):
        idx = dataset[:,0].astype(int) == i
        data = dataset[idx,:]
        coordinates.append(data[:,1:3])
        features.append(data[:,3:-1])
        stress.append(data[:,-1])
    dataset_train = dict(coordinates=coordinates[:split],
features=features[:split], stress=stress[:split])
    dataset_test = dict(coordinates=coordinates[split:],
features=features[split:], stress=stress[split:])
    X_train, X_test = np.concatenate(features[:split], axis=0),
np.concatenate(features[split:], axis=0)
    y_train, y_test = np.concatenate(stress[:split], axis=0),
np.concatenate(stress[split:], axis=0)
    return dataset_train, dataset_test, X_train, X_test, y_train,
y_test

def get_shape(dataset,index):
    X = torch.tensor(dataset["features"][index])
    Y = torch.tensor(dataset["stress"][index].reshape(-1,1))
    return X, Y

def plot_r2_distribution(r2s, title=""):
    plt.figure(dpi=120,figsize=(6,4))
    plt.hist(r2s, bins=10)
    plt.xlabel("$R^2$")
    plt.ylabel("Number of shapes")
    plt.title(title)
    plt.show()
```

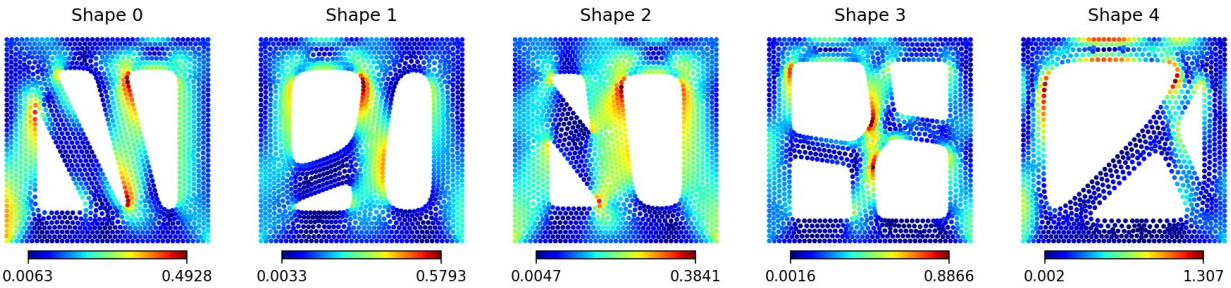
Loading the data

First, complete the code below to load the data and plot the von Mises stress fields for a few shapes.
You'll need to input the path of the data file, the rest is done for you.

All training node features and outputs are in `X_train` and `y_train`, respectively. Testing nodes are in `X_test`, `y_test`.

`dataset_train` and `dataset_test` contain more detailed information such as node coordinates, and they are separated by shape.
Get features and outputs for a shape by calling `get_shape(dataset,index)`. `N_train` and `N_test` are the number of training and testing shapes in each of these datasets.

```
data_path = "../HW9/data/stress_nodal_features.npy"  # updated for
correct file path
dataset_train, dataset_test, X_train, X_test, y_train, y_test =
load_dataset(data_path)
N_train = len(dataset_train["stress"])
N_test = len(dataset_test["stress"])

plt.figure(figsize=[15,3.2], dpi=150)
for i in range(5):
    plt.subplot(1,5,i+1)
    plot_shape(dataset_train,i)
    plt.title(f"Shape {i}")
plt.show()
```



Neural network to predict stress

Create a PyTorch neural network class `StressPredictor` below. This should be an MLP with 60 inputs (the given features) and 1 output (stress). The hidden layer sizes and activations are up to you.

```
import torch.nn.functional as F

class StressPredictor(nn.Module):
    # YOUR CODE GOES HERE
    input_size = 60
```

```
    hidden_size = 64
    output_size = 1

    def __init__(self, input_size, hidden_size, output_size):
        super().__init__()
        self.lin1 = nn.Linear(input_size, hidden_size)
        self.lin2 = nn.Linear(hidden_size, hidden_size)
        self.lin3 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = self.lin1(x)
        x = F.relu(x)
        x = self.lin2(x)
        x = F.relu(x)
        x = self.lin3(x)
        return x
```

Training function

Below, you should define a function `train(model, dataset, lr, epochs)` that will train `model` on the data in `dataset` with the Adam optimizer for `epochs` epochs with a learning rate of `lr`.

Because there are so many total nodes, you should treat each shape as a batch of nodes -- each epoch of training will require you to loop through each shape in the dataset in a random order, performing a step of gradient descent for each shape encountered. Your function should automatically generate a plot of the loss curve on training data.

- You can use the provided `get_shape` to access feature and output tensors for each shape.
- Use MSE as a your loss function.
- Look into `np.random.permutation()` for generating a random index order

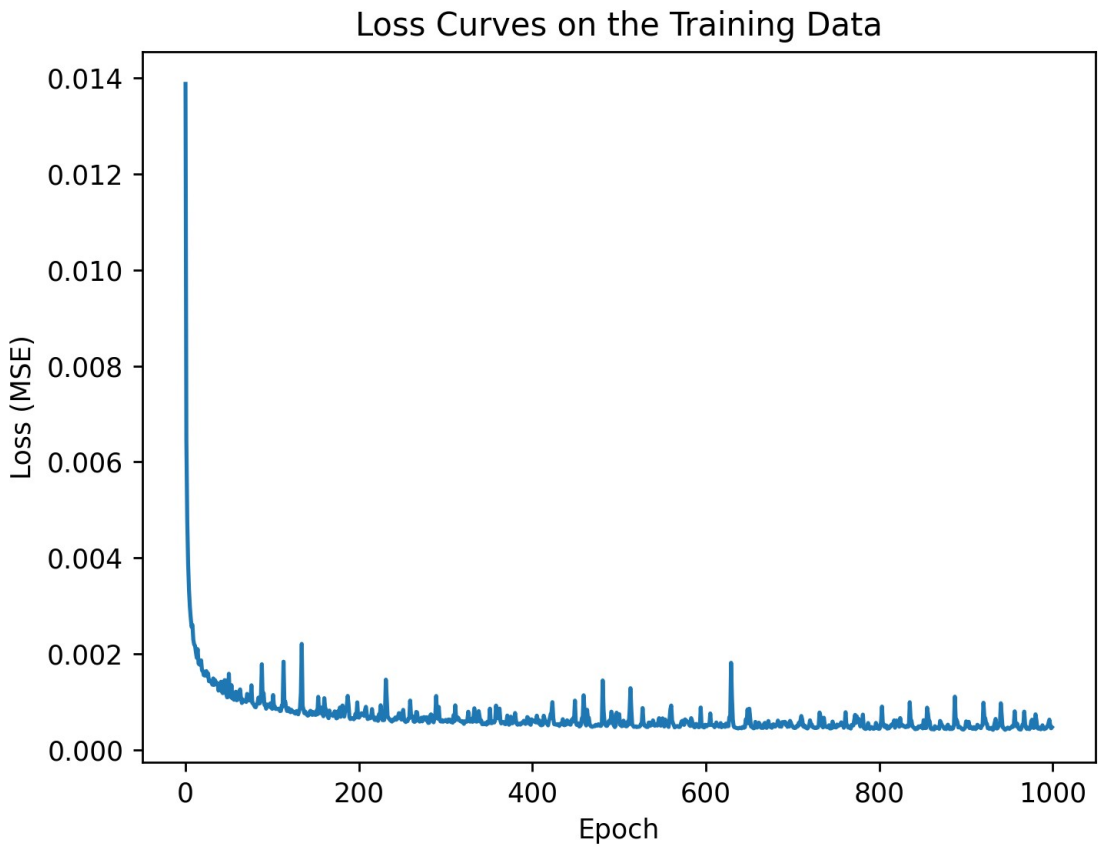
```
def train(model, dataset, lr, epochs):
    # YOUR CODE GOES HERE
    # use np.random.permutation(N_train) to shuffle the training data
    lossfun = nn.MSELoss()
    opt = optim.Adam(model.parameters(), lr=lr)
    train_hist = []
    val_hist = []

    for epoch in range(epochs):
        model.train()
        train_loss = 0
        for i in np.random.permutation(len(dataset["features"])):
            x, y = get_shape(dataset, i)
            opt.zero_grad()
            y_pred = model(x)
            loss = lossfun(y_pred, y)
```

```
            loss.backward()
            opt.step()
            train_loss += loss.item()

        train_hist.append(train_loss / len(dataset["features"]))

        model.eval()
        val_loss = 0
        for i in range(N_test):
            x, y = get_shape(dataset, i)
            y_pred = model(x)
            loss = lossfun(y_pred, y)
            val_loss += loss.item()
        val_hist.append(val_loss / len(dataset["features"]))

    # plot the loss curves
    plt.figure(dpi=250)
    plt.plot(train_hist, label="Train")
    # plt.plot(val_hist, label="Validation")
    plt.title("Loss Curves on the Training Data")
    plt.xlabel("Epoch")
    plt.ylabel("Loss (MSE)")
    # plt.legend()
    plt.show()
```

Training your Neural Network

Now, create your neural network model and run your train function on the training dataset `dataset_train`.
Determining the right number of epochs and learning rate are up to you. The training loss curve should be shown.

```
# YOUR CODE GOES HERE
lr = 1e-3
epochs = 1000
model = StressPredictor(60, 64, 1)
train(model, dataset_train, lr, epochs)
```



R^2 Score

Compute the R^2 Score on the training dataset. You will have to convert between tensors and arrays versions to use sklearn functions, or you can write your own function.

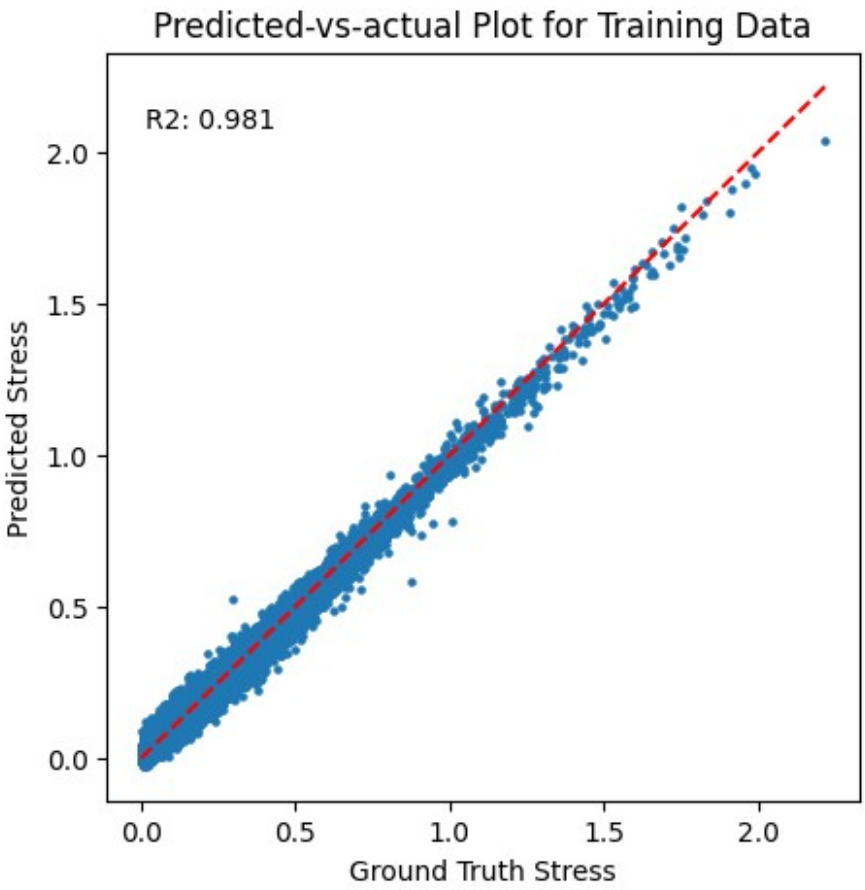
```
# YOUR CODE GOES HERE
# covert to tensor
X_train_tensor = torch.tensor(X_train).float()
X_test_tensor = torch.tensor(X_test).float()

# predict the stress
y_train_pred = model(X_train_tensor).detach().numpy()
y_test_pred = model(X_test_tensor).detach().numpy()

# calculate the R^2 score
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)

print(f"R^2 score on the training dataset: {r2_train:.4f}")
print(f"R^2 score on the test dataset: {r2_test:.4f}")
```

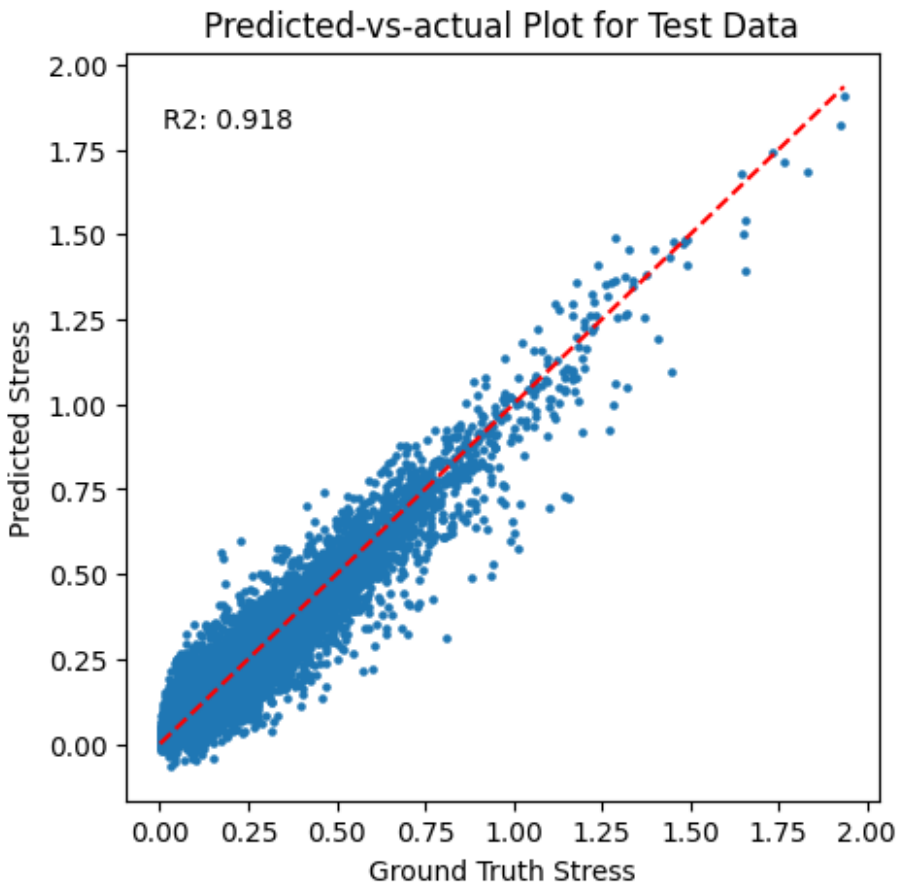
```
R^2 score on the training dataset: 0.9806
R^2 score on the test dataset: 0.9181
```

R^2 Plots

Now, generate predicted-vs-actual plots that display both data and a theoretical best fit line. Make 2 such plots - one for training data and one for testing.

```
# YOUR CODE GOES HERE
def plot_r2(y_true, y_pred, title):
    plt.figure(figsize=[5,5])
    plt.scatter(y_true, y_pred, s=5)
    plt.text(0.05, 0.9, f"R2: {r2_score(y_true, y_pred):.3f}",
transform=plt.gca().transAxes)
    plt.plot([min(y_true), max(y_true)], [min(y_true), max(y_true)],
"r--")

    plt.xlabel("Ground Truth Stress")
    plt.ylabel("Predicted Stress")
    plt.title(title)
    plt.show()

plot_r2(y_train, y_train_pred,"Predicted-vs-actual Plot for Training
Data")
plot_r2(y_test, y_test_pred, "Predicted-vs-actual Plot for Test Data")
```


Predicted-vs-actual Plot for Training Data
R2: 0.981
Predicted Stress
Ground Truth Stress
0.0
0.5
1.0
1.5
2.0



Individual Shape R^2

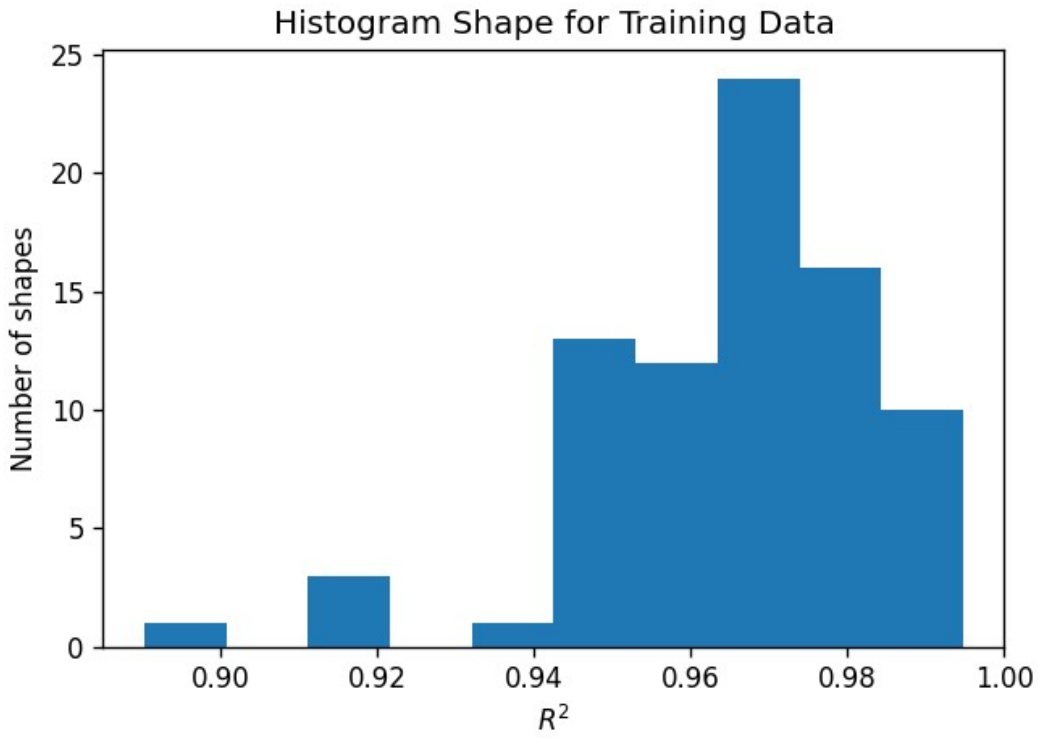
Because we have a unique problem where groups of nodes in a dataset form a single shape, we can compute an R^2 score for an individual shape. For each shape in the training set, compute an R^2 score. Then create a histogram of the values with the function `plot_r2_hist(r2s)`. Repeat for the testing set.

Report the median R^2 score across all training shapes, and the median across all testing shapes.

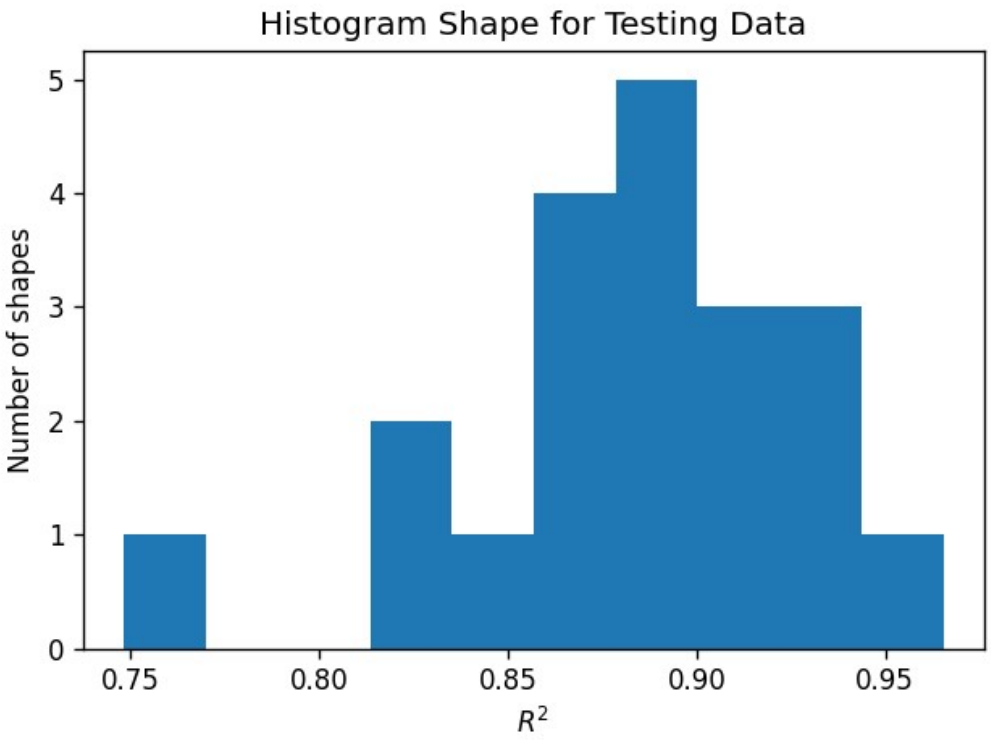
If your test median is below 0.85, try and tune your network size/training hyperparameters until it reaches this threshold.

```
# YOUR CODE GOES HERE
def r2_shape(dataset, model):
    r2_scores = []
    for i in range(len(dataset["features"])):
        x, y = get_shape(dataset, i)
        y_pred = model(x).detach().numpy()
        r2_scores.append(r2_score(y, y_pred))
    return r2_scores

r2_scores_train = r2_shape(dataset_train, model)
```

```
r2_scores_test = r2_shape(dataset_test, model)

# plot and median for training data
plot_r2_distribution(r2_scores_train, "Histogram Shape for Training
Data")
r2_train_median = np.median(r2_scores_train)
print(f"Median R^2 on the training dataset: {r2_train_median:.4f}")

# plot and median for test data
plot_r2_distribution(r2_scores_test, "Histogram Shape for Testing
Data")
r2_test_median = np.median(r2_scores_test)
print(f"Median R^2 on the testing dataset: {r2_test_median:.4f}")
```




```
Median R^2 on the training dataset: 0.9683
```




```
Median R^2 on the testing dataset: 0.8890
```