```
!pip install aeon
```

```
Requirement already satisfied: aeon in /usr/local/lib/python3.12/dist-packages (1.3.0)
Requirement already satisfied: deprecated>=1.2.13 in /usr/local/lib/python3.12/dist-packages (from aeon) (1.3.1)
Requirement already satisfied: numba<0.62.0,>=0.55 in /usr/local/lib/python3.12/dist-packages (from aeon) (0.60.0)
Requirement already satisfied: numpy<2.3.0,>=1.21.0 in /usr/local/lib/python3.12/dist-packages (from aeon) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from aeon) (25.0)
Requirement already satisfied: pandas<2.4.0,>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from aeon) (2.2.2)
Requirement already satisfied: scikit-learn<1.8.0,>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from aeon) (1.6.1)
Requirement already satisfied: scipy<1.16.0,>=1.9.0 in /usr/local/lib/python3.12/dist-packages (from aeon) (1.15.3)
Requirement already satisfied: typing-extensions>=4.6.0 in /usr/local/lib/python3.12/dist-packages (from aeon) (4.15.0)
Requirement already satisfied: wrapt<3,>=1.10 in /usr/local/lib/python3.12/dist-packages (from deprecated>=1.2.13->aeon) (2.0.1)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.12/dist-packages (from numba<0.62.0,>=0.55->
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas<2.4.0,>=2.0.0->aeo
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas<2.4.0,>=2.0.0->aeon) (2025.2
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas<2.4.0,>=2.0.0->aeon) (2025
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn<1.8.0,>=1.0.0->aeon)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn<1.8.0,>=1.0.0-
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas<2.4.0,>=
```

Gun Point Classification via MLP

In this assignment, you are asking to build a classification model via pytorch (discussed in the class), to classify a dataset called GunPoint.

This dataset involves one female actor and one male actor making a motion with their hand. The two classes are: Gun-Draw and Point: For Gun-Draw the actors have their hands by their sides. They draw a replicate gun from a hip-mounted holster, point it at a target for approximately one second, then return the gun to the holster, and their hands to their sides. For Point the actors have their gun by their sides. They point with their index fingers to a target for approximately one second, and then return their hands to their sides. For both classes, we tracked the centroid of the actor's right hands in both X- and Y-axes, which appear to be highly correlated. The data in the archive is just the X-axis. Class 1 is "gun" and class 2 is "no gun (pointing)"

Double-click (or enter) to edit

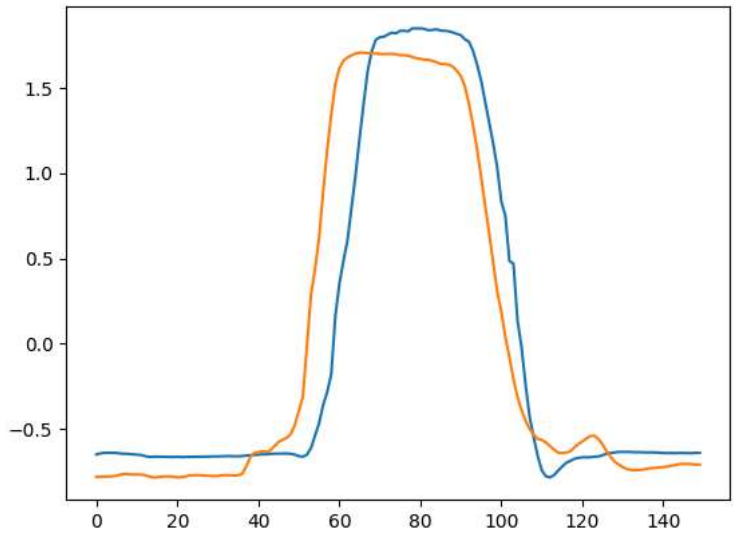
```
from aeon.datasets import load_classification

X, y = load_classification('GunPoint',split="TRAIN")
X = X.reshape(X.shape[0], -1)

y = y.astype('int')
```

```
y.astype('int')
```

```
array([2, 2, 1, 1, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 2, 1, 2, 2, 1, 2, 1, 1,
       1, 2, 1, 2, 1, 1, 2, 1, 1, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 1, 1, 1,
       2, 2, 1, 2, 1, 2])
```

```
import matplotlib.pyplot as plt
plt.plot(X[0,:]) #has gun
plt.plot(X[2,:]) #no gun
```

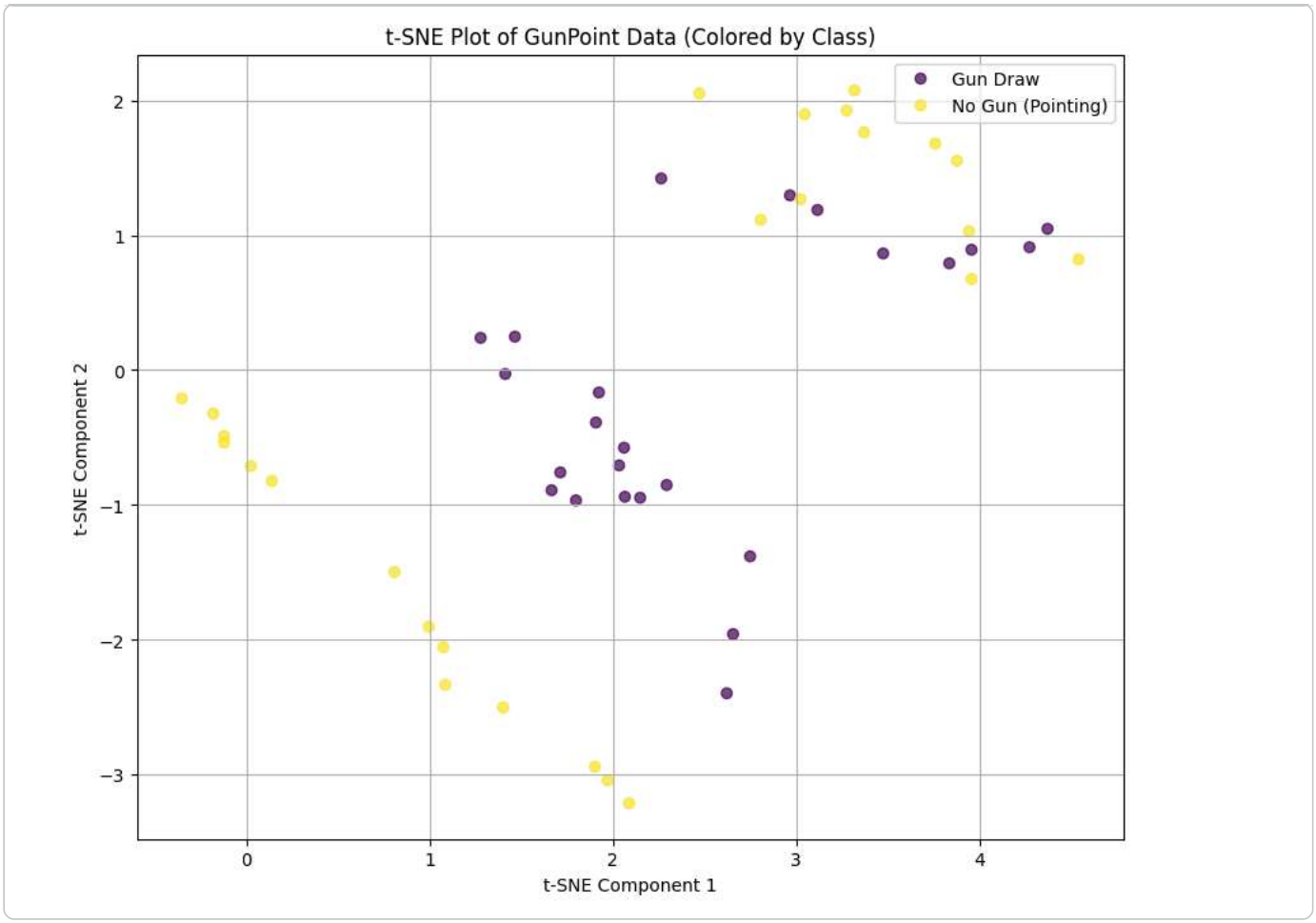
```
[<matplotlib.lines.Line2D at 0x79399e380e00>]
```



Data Visualization

The first step is understanding our data. We will use a method named t-SNE to visualize the data by maping all the data into "points in 2-D plane".

```
import numpy as np
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

# Ensure X_flattened and y_mapped are available from previous steps
# If not, re-run the data preprocessing step or assume they are in the kernel state.

# Apply t-SNE to the flattened data
tsne = TSNE(n_components=2, random_state=42, perplexity=30)
X_tsne = tsne.fit_transform(X)

# Create the t-SNE plot
plt.figure(figsize=(10, 8))
scatter = plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y, cmap='viridis', alpha=0.7)

# Add a legend to distinguish the classes
legend_labels = {1: 'Gun Draw', 2: 'No Gun (Pointing)'}
handles, _ = scatter.legend_elements()

plt.title('t-SNE Plot of GunPoint Data (Colored by Class)')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.grid(True)
plt.legend(handles, [legend_labels[i] for i in np.unique(y)])
plt.show()
```



Task 1: Design a MLP model which input is GunPoint data and its output a classification result. You should follow the classifier design idea we discussed in the class. Choose the activation function wisely from all potential activation functions listed here:

https://docs.pytorch.org/docs/stable/nn.functional.html#non-linear-activation-functions

Hint: Try to use generative model wisely, they may make mistake

```
import torch
print("X Shape:",X.shape)
print("Y Shape:",y.shape)
print("Unique Labels:", np.unique(y))
y = y-1
y_test = y_test - 1
print("unique labels after shift:", np.unique(y))
X_torch = torch.tensor(X).float()
y_torch = torch.tensor(y).long()
print("X_Torch:",X_torch)
print("y_Torch:",y_torch)
```

```
X Shape: (50, 150)
Y Shape: (50,)
Unique Labels: [1 2]
unique labels after shift: [0 1]
X_Torch: tensor([[-0.6479, -0.6420, -0.6382,  ..., -0.6404, -0.6387, -0.6387],
        [-0.6444, -0.6454, -0.6471,  ..., -0.6349, -0.6345, -0.6316],
        [-0.7784, -0.7783, -0.7772,  ..., -0.7042, -0.7076, -0.7071],
        ...,
        [-0.7791, -0.7784, -0.7757,  ..., -0.5050, -0.5037, -0.5044],
        [-0.7030, -0.7026, -0.7025,  ..., -0.6411, -0.6414, -0.6421],
```

```
        [-1.4357, -1.4323, -1.4329,  ..., -1.4355, -1.4353, -1.4309]])
y_Torch: tensor([1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1,
        0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1,
        0, 1])
```

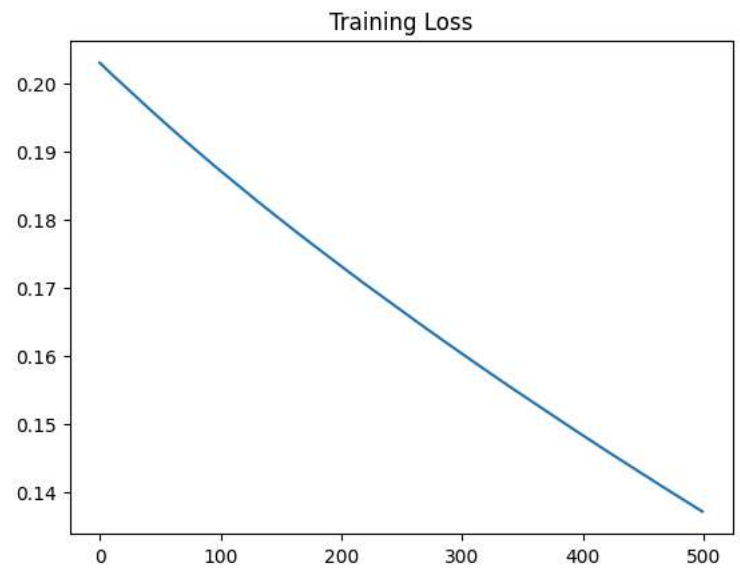
```
import torch.nn as nn

class HandorGun(nn.Module):
  def __init__(self):
    super().__init__()
    self.input_layer = nn.Linear(150, 80)
    self.activation_function = nn.ReLU()
    self.output_layer = nn.Linear(80,2)

  def forward(self, x):
    x = self.input_layer(x)
    x = self.activation_function(x)
    x = self.output_layer(x)
    return x
model = HandorGun()
print(model)
```

```
HandorGun(
  (input_layer): Linear(in_features=150, out_features=80, bias=True)
  (activation_function): ReLU()
  (output_layer): Linear(in_features=80, out_features=2, bias=True)
)
```

```
import torch.optim as optim
loss_func = nn.CrossEntropyLoss()
learning = torch.optim.SGD(model.parameters(), lr=0.01)
```

Task 2: Train the designed MLP model through loaded data X and y

```
loss_history = []
for i in range(500):
  learning.zero_grad()
  outputs = model(X_torch)
  loss = loss_func(outputs, y_torch)
  loss.backward()
  learning.step()
  loss_history.append(loss.item())
import matplotlib.pyplot as plt
plt.plot(loss_history)
plt.title("Training Loss")
plt.show()
```



Task 3: Test the designed MLP model in X_test. Report accuracy based on difference between y_test and your prediction

```
X_test, y_test = load_classification('GunPoint', split='TEST')
X_test = X_test.reshape(X_test.shape[0], -1)

y_test = y_test.astype('int') - 1

X_test_torch = torch.tensor(X_test).float()
y_test_torch = torch.tensor(y_test).long()

print("X_test shape:", X_test_torch.shape)
```