```
import sklearn
import numpy as np
from sklearn.datasets import load_diabetes


#Check out dataset
diabetes = load_diabetes()
print(diabetes.data.shape)
print(diabetes.target.shape)
```

```
(442, 10)
(442,)
```

Problem 1B

```
# Save the data
X = diabetes.data
X_transpose = X.T
y = diabetes.target



# Closed form solution
X_transpose_X = np.dot(X_transpose, X) # https://numpy.org/doc/stable/reference/generated/numpy.dot.html
X_transpose_X_inv = np.linalg.inv(X_transpose_X) # https://numpy.org/doc/stable/reference/routines.linalg.html
X_transpose_y = np.dot(X_transpose, y)
w_star = np.dot(X_transpose_X_inv, X_transpose_y)
```

Problem 1C

```
# Linear solver
from sklearn.linear_model import LinearRegression



# Load data and fit to model, extract w_star: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target
regressionModel = LinearRegression()
regressionModel.fit(X, y)
w_star_solver = regressionModel.coef_
```

Sanity Check

```
# https://stackoverflow.com/questions/10580676/comparing-two-numpy-arrays-for-equality-element-wise
assert np.allclose(w_star, w_star_solver), "Numpy arrays are not close elementwise to one another"


# Closed form
print("My solution: ")
print(w_star)

# Sklearn solver
print("\nSklearn: ")
print(w_star_solver)
```

```
My solution:
[ -10.0098663  -239.81564367  519.84592005  324.3846455  -792.17563855
  476.73902101  101.04326794  177.06323767  751.27369956   67.62669218]

Sklearn:
[ -10.0098663  -239.81564367  519.84592005  324.3846455  -792.17563855
  476.73902101  101.04326794  177.06323767  751.27369956   67.62669218]
```

Problem 2C

```
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_wine
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load dataset
X, y = load_wine(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Fit our model
model = LogisticRegression()
model.fit(X_train, y_train)
```

```
c:\Users\liemj\anaconda3\envs\ASLenv\Lib\site-packages\sklearn\linear_model\_logistic.py:460: ConvergenceWarning: lbfgs failed to conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

LogisticRegression

LogisticRegression()

```
y_true = y_test
y_pred = model.predict(X_test)

print("Model coef: ")
print(model.coef_)

print("\nGround truth: ")
print(y_true)

print("\nPredicted")
print(y_pred)
```

```
Model coef:
[[-1.71720584e-02  1.49617205e-01  1.25232870e-01 -2.25825380e-01
  -3.19189734e-02  1.80023530e-01  4.02454060e-01 -1.80556348e-02
   7.23346435e-02 -2.19815151e-02  1.60663612e-02  3.03674641e-01
   7.74886272e-03]
 [ 3.97803158e-01 -5.70976516e-01 -1.05541699e-01  1.42490944e-01
   1.56126453e-02  2.92111209e-01  4.43766844e-01  3.16179559e-02
   3.16850041e-01 -1.10306732e+00  2.17536302e-01  4.11674292e-01
  -8.07889407e-03]
 [-3.80631099e-01  4.21359311e-01 -1.96911709e-02  8.33344361e-02
   1.63063281e-02 -4.72134739e-01 -8.46220904e-01 -1.35623212e-02
  -3.89184685e-01  1.12504884e+00 -2.33602663e-01 -7.15348933e-01
   3.30031346e-04]]

Ground truth:
[0 1 2 2 2 1 1 1 1 2 2 1 1 1 0 1 1 1 0 0 1 2 1 1 1 1 2 1 0 2 0 2 2 2 0 2]

Predicted
[0 1 2 2 2 1 1 1 1 2 2 1 1 1 0 1 1 1 0 0 1 2 1 1 1 1 2 1 0 2 0 2 2 2 0 2]
```

Problem 3A

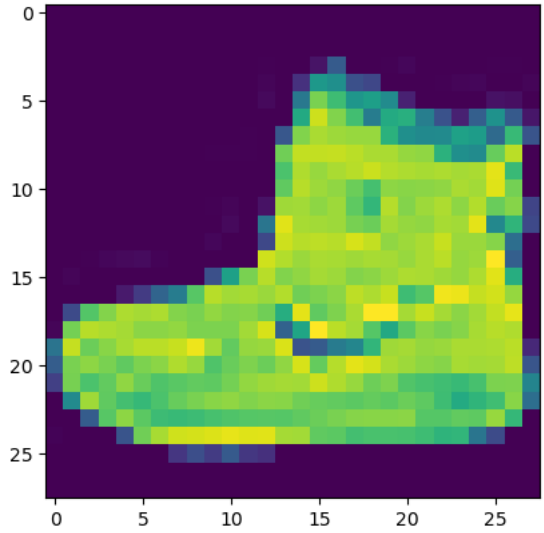
```
import torchvision
import torch
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from torch.utils.data import DataLoader
from torch.utils.data import TensorDataset

# Download and load the data
data = torchvision.datasets.FashionMNIST('./FashionMNIST', train=True, download=True, transform=torchvision.transforms.ToTensor())
```

```
sample_img, sample_lbl = data[0]
print(sample_img.shape)
print(sample_lbl)
```

```
torch.Size([1, 28, 28])
9
```

```
plt.imshow(sample_img.squeeze())
```

```
<matplotlib.image.AxesImage at 0x1695e09ad50>
```




```
train_length = int(0.8 * len(data))
test_length = int(0.2 * len(data))

# Prepare into dataloader
train_dataset, test_dataset = torch.utils.data.random_split(data, (train_length, test_length))
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32)
```

Problem 3B

```
from torch.nn import Module
from torch.nn import CrossEntropyLoss
from torch.optim import SGD
from torch.nn import Linear
from torch.nn import ReLU
from torch.nn import Sequential
from tqdm import tqdm


class JosefNet(Module):
  def __init__(self, inshape=(28, 28), num_classes=10):
    super(JosefNet, self).__init__()
    self.inshape = inshape

    input_size = self.inshape[0] * self.inshape[1]

    self.layer1 = self.JosefLayer(input_size, input_size + 10)
    self.layer2 = self.JosefLayer(input_size + 10, input_size + 20)
    self.layer3 = self.JosefLayer(input_size + 20, input_size + 10)
    self.layer4 = Linear(input_size + 10, num_classes)

  def JosefLayer(self, in_channels, out_channels):
    layer = Sequential(
        Linear(in_channels, out_channels),
        ReLU(),
    )
    return layer

  def forward(self, x):
    x = x.view(-1, self.inshape[0] * self.inshape[1])
```

```
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        return x

    # 1. Model 2. Loss 3. Optimization
    model = JosefNet(inshape=(28, 28), num_classes=10)
    criterion = CrossEntropyLoss()
    optimizer = SGD(model.parameters(), lr=0.01)


    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    print(f'Using {device}...')

    model = model.to(device)
```

```
Using cpu...
```

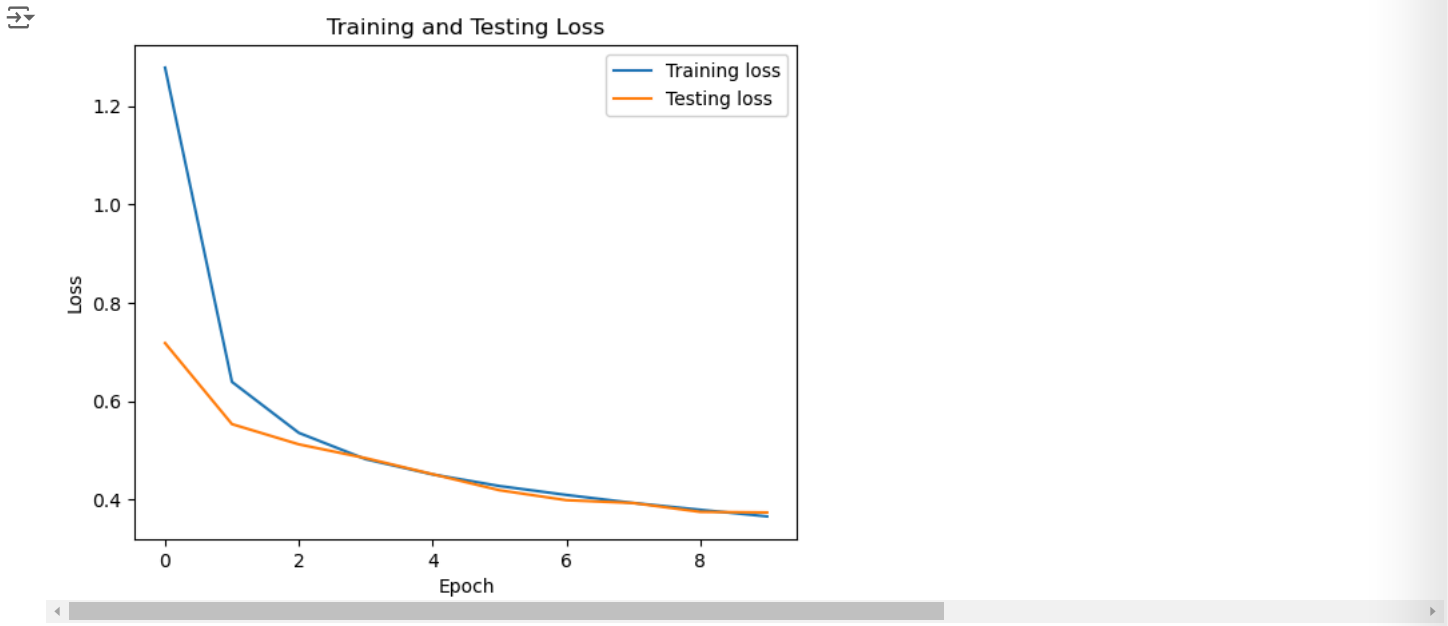
```
    n_epochs = 10
    train_losses = []
    test_losses = []

    for epoch in range(n_epochs):
      train_loss = 0
      test_loss = 0

      model.train()
      for imgs, lbls in tqdm(train_loader, desc=f"Epoch {epoch+1}/{n_epochs} - Training"):
        imgs = imgs.to(device)
        lbls = lbls.to(device)

        # get prediction and calculate training loss on batch
        preds = model(imgs)
        loss = criterion(preds, lbls)
        loss.backward() # Backward pass
        optimizer.step() # Gradient update
        optimizer.zero_grad() # Zero gradient for next run

        train_loss += loss.item()

      # calculate the loss
      train_loss /= len(train_loader)
      train_losses.append(train_loss)

      model.eval()
      with torch.no_grad():
        # do the same on test data each epoch
        for imgs, lbls in tqdm(test_loader, desc=f"Epoch {epoch+1}/{n_epochs} - Testing"):
          imgs = imgs.to(device)
          lbls = lbls.to(device)

          # get preds and calculate batchwise loss
          preds = model(imgs)
          loss = criterion(preds, lbls)

          test_loss += loss.item()

        # Calculate test loss
        test_loss /= len(test_loader)
        test_losses.append(test_loss)

      print(f'Training Loss: {train_loss} \t Test Loss: {test_loss}')
```

```
Epoch 1/10 - Training: 100%|██████████| 1500/1500 [00:12<00:00, 115.80it/s]
Epoch 1/10 - Testing: 100%|██████████| 375/375 [00:01<00:00, 199.58it/s]
Training Loss: 1.2783720241983731         Test Loss: 0.7180028606255849
Epoch 2/10 - Training: 100%|██████████| 1500/1500 [00:12<00:00, 120.83it/s]
Epoch 2/10 - Testing: 100%|██████████| 375/375 [00:01<00:00, 202.13it/s]
Training Loss: 0.6391100450754166         Test Loss: 0.5531621694564819
Epoch 3/10 - Training: 100%|██████████| 1500/1500 [00:12<00:00, 120.18it/s]
Epoch 3/10 - Testing: 100%|██████████| 375/375 [00:01<00:00, 202.75it/s]
Training Loss: 0.535347327152888          Test Loss: 0.5120111543337504
Epoch 4/10 - Training: 100%|██████████| 1500/1500 [00:12<00:00, 119.75it/s]
Epoch 4/10 - Testing: 100%|██████████| 375/375 [00:01<00:00, 206.03it/s]
Training Loss: 0.4817542056838671         Test Loss: 0.48389638829231263
Epoch 5/10 - Training: 100%|██████████| 1500/1500 [00:12<00:00, 120.13it/s]
Epoch 5/10 - Testing: 100%|██████████| 375/375 [00:02<00:00, 185.58it/s]
```

```
Training Loss: 0.4507949015696843        Test Loss: 0.4513076793750127
Epoch 6/10 - Training: 100%|██████████| 1500/1500 [00:12<00:00, 119.91it/s]
Epoch 6/10 - Testing: 100%|██████████| 375/375 [00:01<00:00, 199.35it/s]
Training Loss: 0.4271812918086847        Test Loss: 0.4186215761701266
Epoch 7/10 - Training: 100%|██████████| 1500/1500 [00:12<00:00, 121.31it/s]
Epoch 7/10 - Testing: 100%|██████████| 375/375 [00:01<00:00, 201.48it/s]
Training Loss: 0.40905945078035194       Test Loss: 0.398398036022981
Epoch 8/10 - Training: 100%|██████████| 1500/1500 [00:12<00:00, 120.33it/s]
Epoch 8/10 - Testing: 100%|██████████| 375/375 [00:01<00:00, 200.26it/s]
Training Loss: 0.39271588702499866       Test Loss: 0.3925811664263407
Epoch 9/10 - Training: 100%|██████████| 1500/1500 [00:12<00:00, 120.47it/s]
Epoch 9/10 - Testing: 100%|██████████| 375/375 [00:01<00:00, 202.14it/s]
Training Loss: 0.3788658427745104        Test Loss: 0.37457411682605746
Epoch 10/10 - Training: 100%|██████████| 1500/1500 [00:12<00:00, 119.54it/s]
Epoch 10/10 - Testing: 100%|██████████| 375/375 [00:01<00:00, 204.16it/s]Training Loss: 0.3653795603265365       Test Loss: 0.3734455082
```

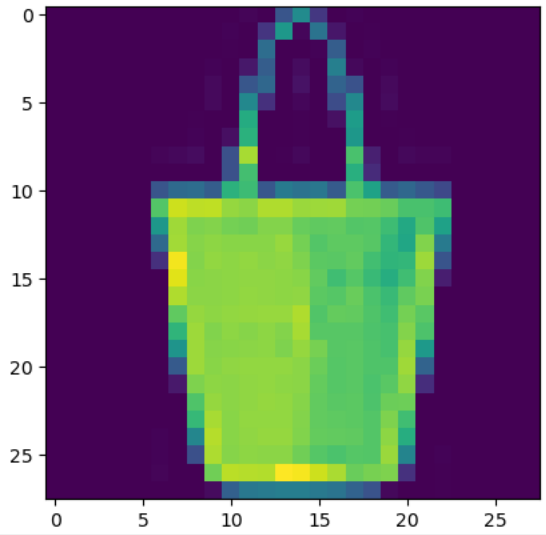
Problem 3C

```
#plot the losses
plt.plot(train_losses, label='Training loss')
plt.plot(test_losses, label='Testing loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Testing Loss')
plt.legend()
plt.show()
```




```
# sample prediction
model.eval()
sample_img, sample_lbl = test_dataset[1]
sample_img = sample_img.to(device)

with torch.no_grad():
    sample_pred = model(sample_img.unsqueeze(0))
    sample_pred = torch.argmax(sample_pred, dim=1)

print(f'Predicted: {sample_pred.item()} \t Ground Truth: {sample_lbl}')
plt.imshow(sample_img.squeeze().cpu().numpy())
```

```
Predicted: 8     Ground Truth: 8
<matplotlib.image.AxesImage at 0x16959d62e10>
```




```
# accuracy
correct = 0
total = 0
model.eval()

with torch.no_grad():
    for imgs, lbls in test_loader:
        imgs = imgs.to(device)
        lbls = lbls.to(device)

        preds = model(imgs)
        preds = torch.argmax(preds, dim=1)

        total += lbls.size(0)
        correct += (preds == lbls).sum().item()

print(f'Accuracy: {correct/total * 100}%')
```

```
Accuracy: 86.90833333333333%
```

Training loss and testing losss are relatively comparable in my training process, thus it seems like it isn't really over or underfitting. That said, an accuracy of 87% can be improved some more. Tweaking with the number of channels and observing consequent performance indicates it is likely a problem of too large a number of parameters for a rather simple dataset.

Problem 4A

```
import torchvision
import torch
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from torch.utils.data import DataLoader
from torch.utils.data import TensorDataset
from torch.nn import Module
from torch.nn import CrossEntropyLoss
from torch.optim import SGD
from torch.nn import Linear
from torch.nn import ReLU
from torch.nn import Sequential
from tqdm import tqdm
import numpy as np


class JosefNet(Module):
  def __init__(self, inshape=(28, 28), num_classes=10):
    super(JosefNet, self).__init__()
    self.inshape = inshape
```

```
    input_size = self.inshape[0] * self.inshape[1]

    self.layer1 = self.JosefLayer(input_size, input_size + 24)
    self.layer2 = self.JosefLayer(input_size + 24, input_size + 16)
    self.layer3 = Linear(input_size + 16, num_classes)

  def JosefLayer(self, in_channels, out_channels):
    layer = Sequential(
        Linear(in_channels, out_channels),
        ReLU(),
    )
    return layer

  def forward(self, x):
    x = x.view(-1, self.inshape[0] * self.inshape[1])
    x = self.layer1(x)
    x = self.layer2(x)
    x = self.layer3(x)
    return x


# Download and load the data
data = torchvision.datasets.FashionMNIST('./FashionMNIST', train=True, download=True, transform=torchvision.transforms.ToTensor())


train_length = int(0.8 * len(data))
test_length = int(0.2 * len(data))

# Prepare a subsample into dataloader
train_dataset, test_dataset = torch.utils.data.random_split(data, (train_length, test_length))

subsampled_train_dataset = torch.utils.data.Subset(train_dataset, range(25000))
sample_train_loader = DataLoader(subsampled_train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32)


# What is the size of our subsample?
print(f"Shape of subsample training data: {len(sample_train_loader.dataset)}")
```

```
Shape of subsample training data: 25000
```

Problem 4B

```
def train_benchmarker(optimizer, model, n_epochs = 10):
  criterion = CrossEntropyLoss()

  # Move model to device
  device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
  model = model.to(device)

  # Track losses over epochs
  train_losses = []
  test_losses = []

  for epoch in range(n_epochs):
    train_loss = 0
    test_loss = 0

    model.train()
    for imgs, lbls in tqdm(sample_train_loader, desc=f"Epoch {epoch+1}/{n_epochs} - Training"):
      imgs = imgs.to(device)
      lbls = lbls.to(device)

      # get prediction and calculate training loss on batch
      preds = model(imgs)
      loss = criterion(preds, lbls)
      loss.backward() # Backward pass
      optimizer.step() # Gradient update
      optimizer.zero_grad() # Zero gradient for next run

      train_loss += loss.item()

    # calculate the loss
    train_loss /= len(train_loader)
    train_losses.append(train_loss)
```

```
    model.eval()
    with torch.no_grad():
      # do the same on test data each epoch
      for imgs, lbls in tqdm(test_loader, desc=f"Epoch {epoch+1}/{n_epochs} - Testing"):
        imgs = imgs.to(device)
        lbls = lbls.to(device)

        # get preds and calculate batchwise loss
        preds = model(imgs)
        loss = criterion(preds, lbls)

        test_loss += loss.item()

      # Calculate test loss
      test_loss /= len(test_loader)
      test_losses.append(test_loss)

    print(f'Training Loss: {train_loss} \t Test Loss: {test_loss}')
  return (train_losses, test_losses)
```

Problem 4C

```
# SGD optim
n_epochs = 10
SGD_test_losses = []

for i in range(5):
  model = JosefNet(inshape=(28, 28), num_classes=10)
  optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
  _, SGD_test_loss = train_benchmarker(optimizer, model, n_epochs=n_epochs)
  SGD_test_losses.append(SGD_test_loss)
```

```
Training Loss: 0.25132671433464687       Test Loss: 0.49179240469137825
Epoch 7/10 - Training: 100%|██████████| 782/782 [00:09<00:00, 84.93it/s]
Epoch 7/10 - Testing: 100%|██████████| 375/375 [00:02<00:00, 140.96it/s]
Training Loss: 0.2427383978466193        Test Loss: 0.49242932720979055
Epoch 8/10 - Training: 100%|██████████| 782/782 [00:09<00:00, 82.12it/s]
Epoch 8/10 - Testing: 100%|██████████| 375/375 [00:02<00:00, 148.17it/s]
Training Loss: 0.23422922939062119       Test Loss: 0.46829562246799467
Epoch 9/10 - Training: 100%|██████████| 782/782 [00:09<00:00, 83.02it/s]
Epoch 9/10 - Testing: 100%|██████████| 375/375 [00:02<00:00, 144.48it/s]
Training Loss: 0.22837601993481318       Test Loss: 0.4565233874320984
Epoch 10/10 - Training: 100%|██████████| 782/782 [00:10<00:00, 77.86it/s]
Epoch 10/10 - Testing: 100%|██████████| 375/375 [00:02<00:00, 145.13it/s]Training Loss: 0.2224973182529211       Test Loss: 0.4492346
```

```
# Calculate average test losses
average_SGD_test_losses = np.mean(SGD_test_losses, axis=0)


# Adam optim
n_epochs = 10
Adam_test_losses = []

for i in range(5):
  model = JosefNet(inshape=(28, 28), num_classes=10)
  optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
  _, Adam_test_loss = train_benchmarker(optimizer, model, n_epochs=n_epochs)
  Adam_test_losses.append(Adam_test_loss)
```

```
Epoch 10/10 - Training: 100%|██████████| 782/782 [00:31<00:00, 25.19it/s]
Epoch 10/10 - Testing: 100%|██████████| 375/375 [00:02<00:00, 134.70it/s]Training Loss: 0.2116421319829921       Test Loss: 0.4604064
```

```
# Calculate average test losses
average_Adam_test_losses = np.mean(Adam_test_losses, axis=0)
```

```
# Adam optim
n_epochs = 10
RMS_test_losses = []

for i in range(5):
  model = JosefNet(inshape=(28, 28), num_classes=10)
  optimizer = torch.optim.RMSprop(model.parameters(), lr=0.01)
  _, RMS_test_loss = train_benchmarker(optimizer, model, n_epochs=n_epochs)
  RMS_test_losses.append(RMS_test_loss)
```

```
Training Loss: 3.167836314479 51          Test Loss: 0.6961359899044037
Epoch 2/10 - Training: 100%|██████████| 782/782 [00:15<00:00, 51.20it/s]
Epoch 2/10 - Testing: 100%|██████████| 375/375 [00:02<00:00, 142.39it/s]
Training Loss: 0.32685648160551983        Test Loss: 0.6004725130796432
Epoch 3/10 - Training: 100%|██████████| 782/782 [00:15<00:00, 51.12it/s]
Epoch 3/10 - Testing: 100%|██████████| 375/375 [00:02<00:00, 143.30it/s]
Training Loss: 0.3055389421880245         Test Loss: 1.0797079923152924
Epoch 4/10 - Training: 100%|██████████| 782/782 [00:15<00:00, 51.57it/s]
Epoch 4/10 - Testing: 100%|██████████| 375/375 [00:04<00:00, 93.30it/s]
Training Loss: 0.32157494181394575        Test Loss: 0.8298569944699605
Epoch 5/10 - Training: 100%|██████████| 782/782 [00:15<00:00, 50.27it/s]
Epoch 5/10 - Testing: 100%|██████████| 375/375 [00:02<00:00, 142.55it/s]
Training Loss: 0.2914831153303385         Test Loss: 0.5993122043609619
Epoch 6/10 - Training: 100%|██████████| 782/782 [00:19<00:00, 40.77it/s]
Epoch 6/10 - Testing: 100%|██████████| 375/375 [00:03<00:00, 98.49it/s]
Training Loss: 0.27585549017290273        Test Loss: 0.5378888872861862
Epoch 7/10 - Training: 100%|██████████| 782/782 [00:25<00:00, 30.60it/s]
Epoch 7/10 - Testing: 100%|██████████| 375/375 [00:02<00:00, 128.60it/s]
Training Loss: 0.2719623350203037         Test Loss: 0.5339740462700526
Epoch 8/10 - Training: 100%|██████████| 782/782 [00:30<00:00, 25.49it/s]
Epoch 8/10 - Testing: 100%|██████████| 375/375 [00:02<00:00, 137.03it/s]
Training Loss: 0.2664204407334328         Test Loss: 0.5773544978300731
Epoch 9/10 - Training: 100%|██████████| 782/782 [00:41<00:00, 18.88it/s]
Epoch 9/10 - Testing: 100%|██████████| 375/375 [00:03<00:00, 104.37it/s]
Training Loss: 0.2661888409157594         Test Loss: 0.5212955634991328
Epoch 10/10 - Training: 100%|██████████| 782/782 [00:44<00:00, 17.73it/s]
Epoch 10/10 - Testing: 100%|██████████| 375/375 [00:02<00:00, 137.36it/s]
Training Loss: 0.26102608639001845        Test Loss: 0.7628260831038157
Epoch 1/10 - Training: 100%|██████████| 782/782 [00:15<00:00, 51.67it/s]
Epoch 1/10 - Testing: 100%|██████████| 375/375 [00:02<00:00, 145.47it/s]
Training Loss: 5.822545525372028          Test Loss: 0.5917075935999553
Epoch 2/10 - Training: 100%|██████████| 782/782 [00:15<00:00, 49.36it/s]
Epoch 2/10 - Testing: 100%|██████████| 375/375 [00:03<00:00, 119.03it/s]
Training Loss: 0.29796630357205867        Test Loss: 0.5649822023510933
Epoch 3/10 - Training: 100%|██████████| 782/782 [00:15<00:00, 50.65it/s]
Epoch 3/10 - Testing: 100%|██████████| 375/375 [00:02<00:00, 143.43it/s]
Training Loss: 0.27856097034116584        Test Loss: 0.4943006470998128
Epoch 4/10 - Training: 100%|██████████| 782/782 [00:15<00:00, 49.97it/s]
Epoch 4/10 - Testing: 100%|██████████| 375/375 [00:03<00:00, 121.11it/s]
Training Loss: 0.27168866800765196        Test Loss: 0.512252598643303
Epoch 5/10 - Training: 100%|██████████| 782/782 [00:16<00:00, 46.86it/s]
Epoch 5/10 - Testing: 100%|██████████| 375/375 [00:02<00:00, 139.84it/s]
Training Loss: 0.2668871119270722         Test Loss: 0.5534438116550445
Epoch 6/10 - Training: 100%|██████████| 782/782 [00:19<00:00, 40.25it/s]
Epoch 6/10 - Testing: 100%|██████████| 375/375 [00:03<00:00, 94.86it/s]
Training Loss: 0.24255538199841975        Test Loss: 0.5044092157681783
Epoch 7/10 - Training: 100%|██████████| 782/782 [00:25<00:00, 30.78it/s]
Epoch 7/10 - Testing: 100%|██████████| 375/375 [00:03<00:00, 120.91it/s]
Training Loss: 0.23655242166419824        Test Loss: 0.6205205432573955
Epoch 8/10 - Training: 100%|██████████| 782/782 [00:31<00:00, 25.13it/s]
Epoch 8/10 - Testing: 100%|██████████| 375/375 [00:02<00:00, 138.64it/s]
Training Loss: 0.23216824392974378        Test Loss: 0.4737894734342893
Epoch 9/10 - Training: 100%|██████████| 782/782 [00:40<00:00, 19.51it/s]
Epoch 9/10 - Testing: 100%|██████████| 375/375 [00:03<00:00, 107.57it/s]
Training Loss: 0.23211938532193502        Test Loss: 0.5611137472391129
Epoch 10/10 - Training: 100%|██████████| 782/782 [00:43<00:00, 17.90it/s]
Epoch 10/10 - Testing: 100%|██████████| 375/375 [00:02<00:00, 140.28it/s]Training Loss: 0.22849595892926058      Test Loss: 0.7176744
```

```
# Calculate average test losses
average_RMS_test_losses = np.mean(RMS_test_losses, axis=0)


plt.plot(average_SGD_test_losses, label='SGD')
plt.plot(average_Adam_test_losses, label='Adam')
plt.plot(average_RMS_test_losses, label='RMSprop')
plt.title('Average Test Loss per Epoch')
plt.xlabel('Epoch')
plt.ylabel('Average Test Loss')
plt.legend()
plt.show()
```