neural

June 19, 2025

[ ]:
```
#%%

import os
import sys

sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))

import pandas as pd
import torch
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from torch import nn
from torch.utils.data import DataLoader, Dataset

from neural_net.preprocessing import get_train_test_data
# from randomforest import export_model
from utils.graphs import compare, scatter_plot
from utils.neural_utils import LawDataset, NeuralNetwork, predict
```

[ ]:
```
# %%

X_train, X_test, y_train, y_test = get_train_test_data("../law_data.csv",␣
 ↪"first_pf")

training_data = LawDataset(X_train, y_train)
testing_data = LawDataset(X_test, y_test)

train_dataloader = DataLoader(training_data, batch_size=64, shuffle=True)
test_dataloader = DataLoader(testing_data, batch_size=64, shuffle=False)
```

[ ]:
```
# %%

train_features, train_labels = next(iter(train_dataloader))
print(f"Feature batch shape: {train_features.size()}")
print(f"Labels batch shape: {train_labels.size()}")
```

```
Feature batch shape: torch.Size([64, 25])
```

```
Labels batch shape: torch.Size([64])
```

[ ]:
```
# %%

device = torch.accelerator.current_accelerator().type if torch.accelerator.
 ↪is_available() else "cpu"
print(f"Using {device} device")
```

```
Using cuda device
```

[ ]:
```
# %%

model = NeuralNetwork(input_size=X_train.shape[1]).to(device)
model
```

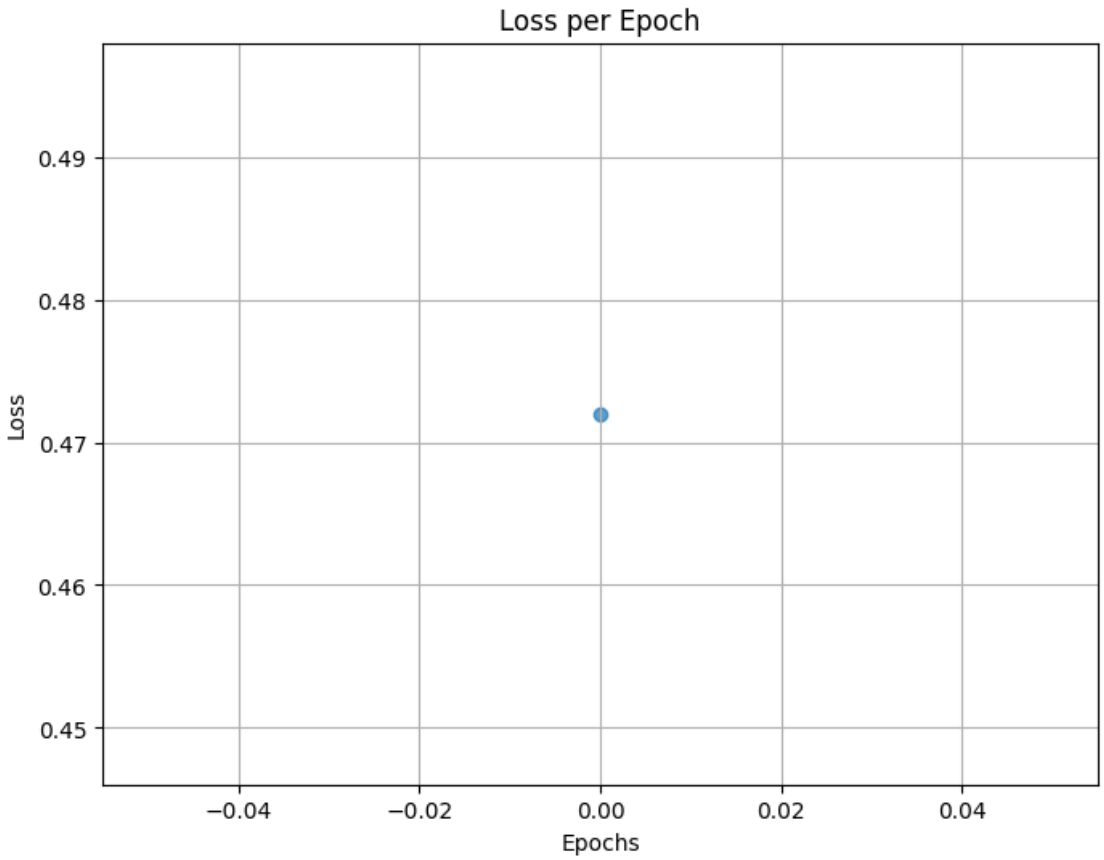
[ ]:
```
NeuralNetwork(
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=25, out_features=1, bias=True)
  )
)
```

[ ]:
```
# %%

learning_rate = 1e-2
batch_size = 64
epochs = 10

losses = {}

def train_loop(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    model.train()
    for batch, (X, y) in enumerate(dataloader):
        X = X.to(device)
        y = y.to(device).float().view(-1, 1)
        pred = model(X)
        # print(X, y)
        loss = loss_fn(pred, y)

        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        if batch % 100 == 0:
            loss_value = loss.item()
            current = batch * batch_size + len(X)
            print(f"loss: {loss_value:>7f}  [{current:>5d}/{size:>5d}]")
```

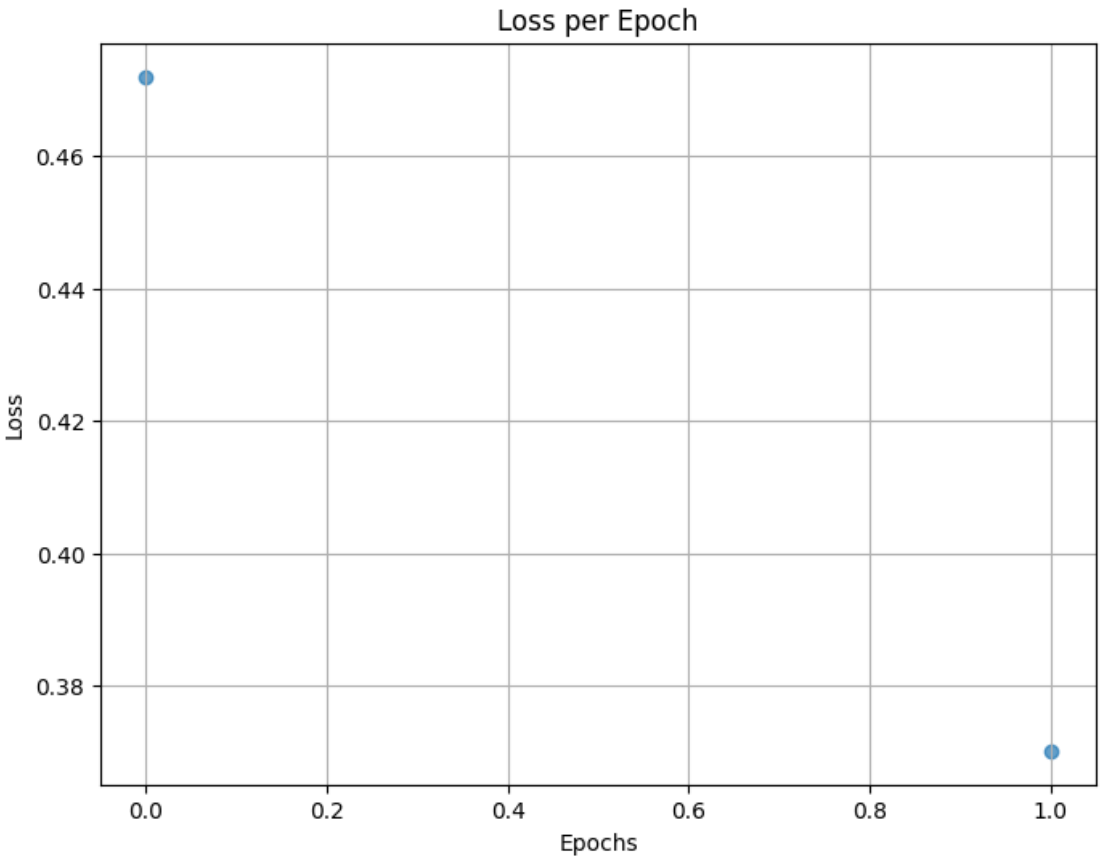
```
def test_loop(dataloader, model, loss_fn, epoch):
    model.eval()
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            X = X.to(device)
            y = y.to(device).float().view(-1, 1)
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            # Pour l'accuracy :
            pred_label = (torch.sigmoid(pred) > 0.5).float()
            correct += (pred_label == y).sum().item()
    test_loss /= num_batches
    correct /= size
    losses[epoch] = test_loss

    scatter_plot(losses, xlabel="Epochs", ylabel="Loss", title="Loss per Epoch")
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss:␣
 ↪{test_loss:>8f} \n")
```

[ ]:
```
# %%

loss_fn = nn.BCEWithLogitsLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

epochs = 10
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train_loop(train_dataloader, model, loss_fn, optimizer)
    test_loop(test_dataloader, model, loss_fn, t)
print("Done!")
```

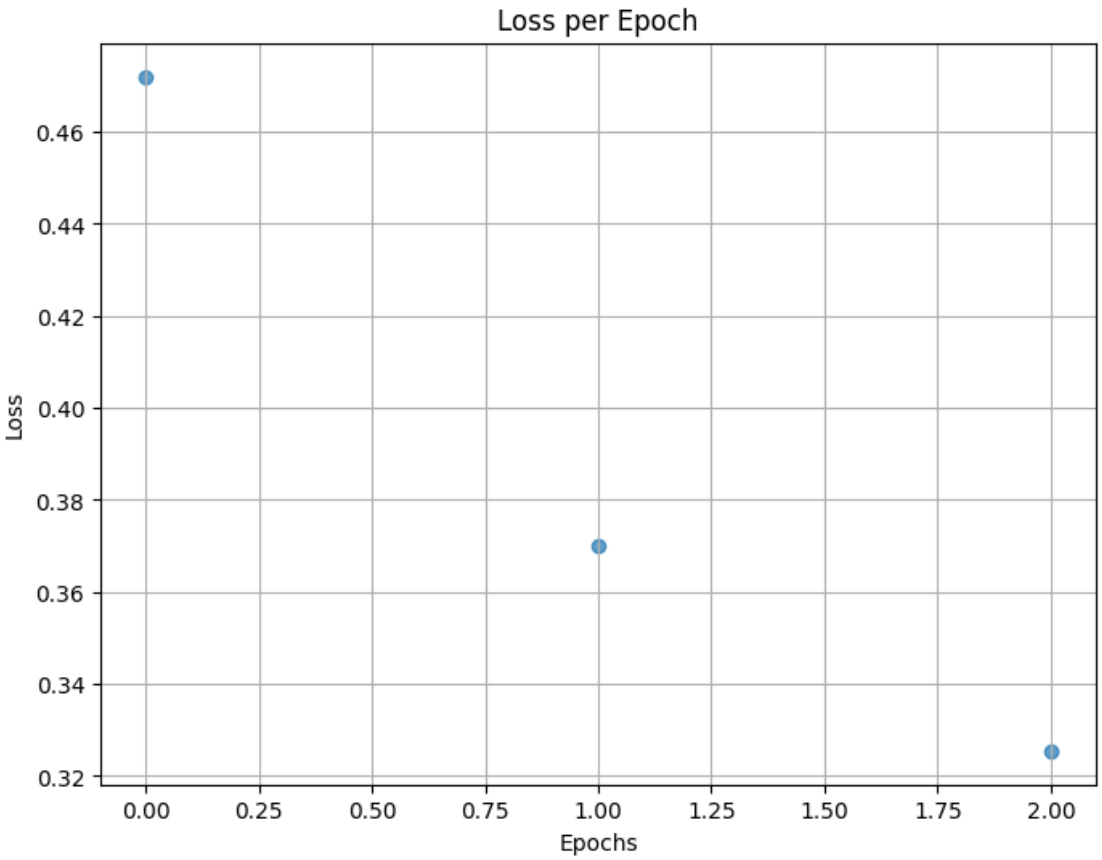
```
Epoch 1
-------------------------------
loss: 0.728525  [   64/15253]
loss: 0.554184  [ 6464/15253]
loss: 0.451272  [12864/15253]
```



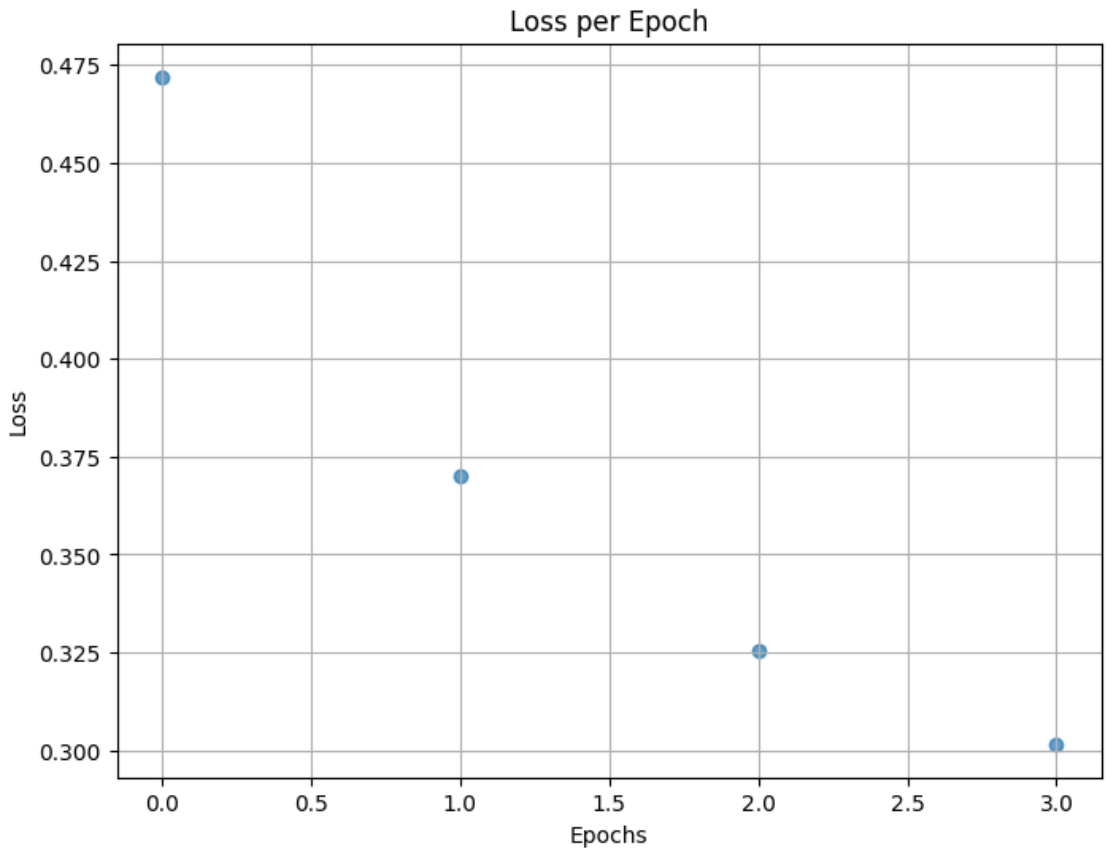

```
Test Error:
 Accuracy: 85.9%, Avg loss: 0.471998

Epoch 2
-------------------------------
loss: 0.479992  [   64/15253]
loss: 0.413908  [ 6464/15253]
loss: 0.366082  [12864/15253]
```



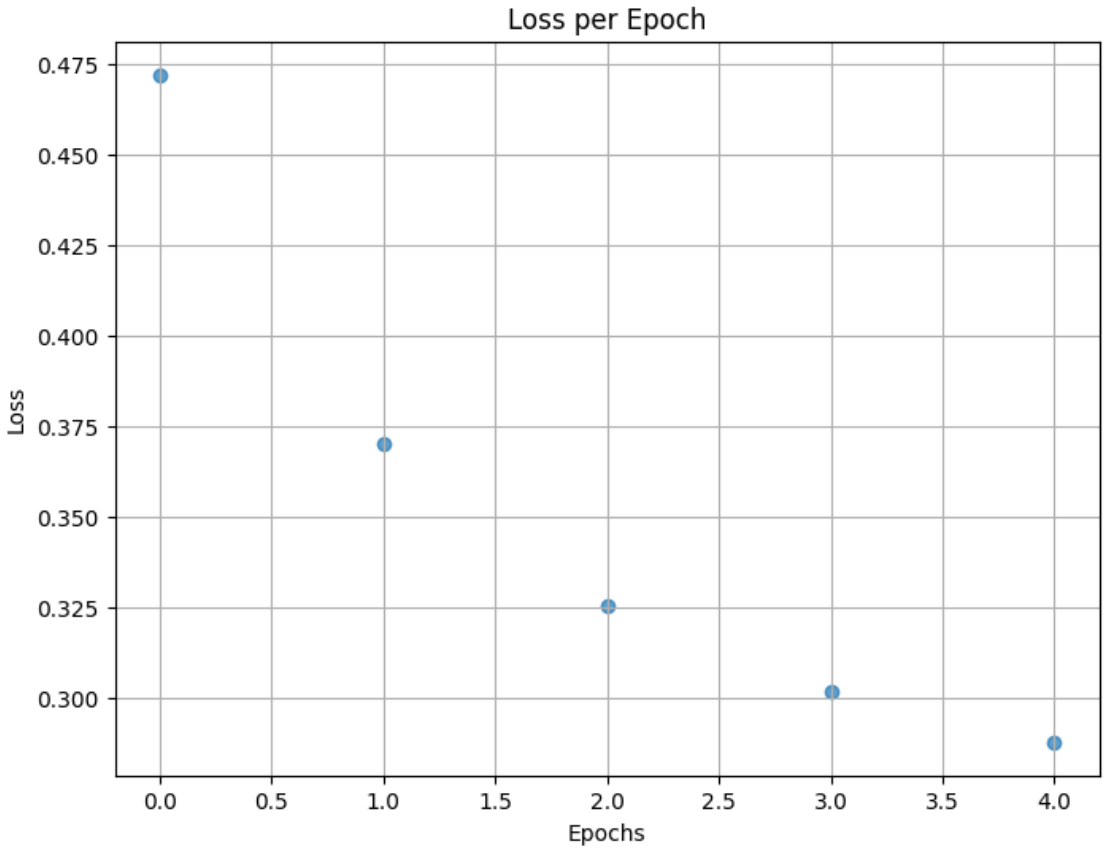

```
Test Error:
 Accuracy: 89.1%, Avg loss: 0.370166

Epoch 3
-------------------------------
loss: 0.306903  [   64/15253]
loss: 0.352540  [ 6464/15253]
loss: 0.273468  [12864/15253]
```



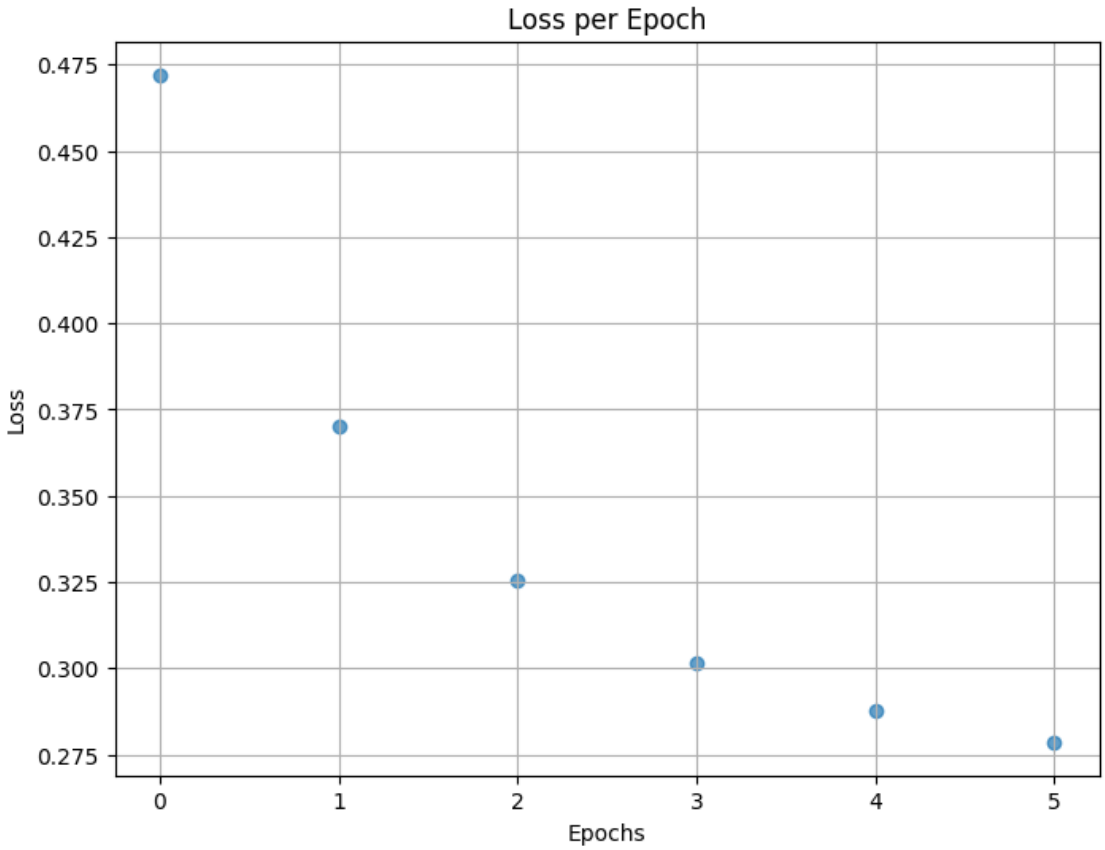

```
Test Error:
 Accuracy: 89.7%, Avg loss: 0.325475

Epoch 4
-------------------------------
loss: 0.329936  [   64/15253]
loss: 0.372335  [ 6464/15253]
loss: 0.356452  [12864/15253]
```



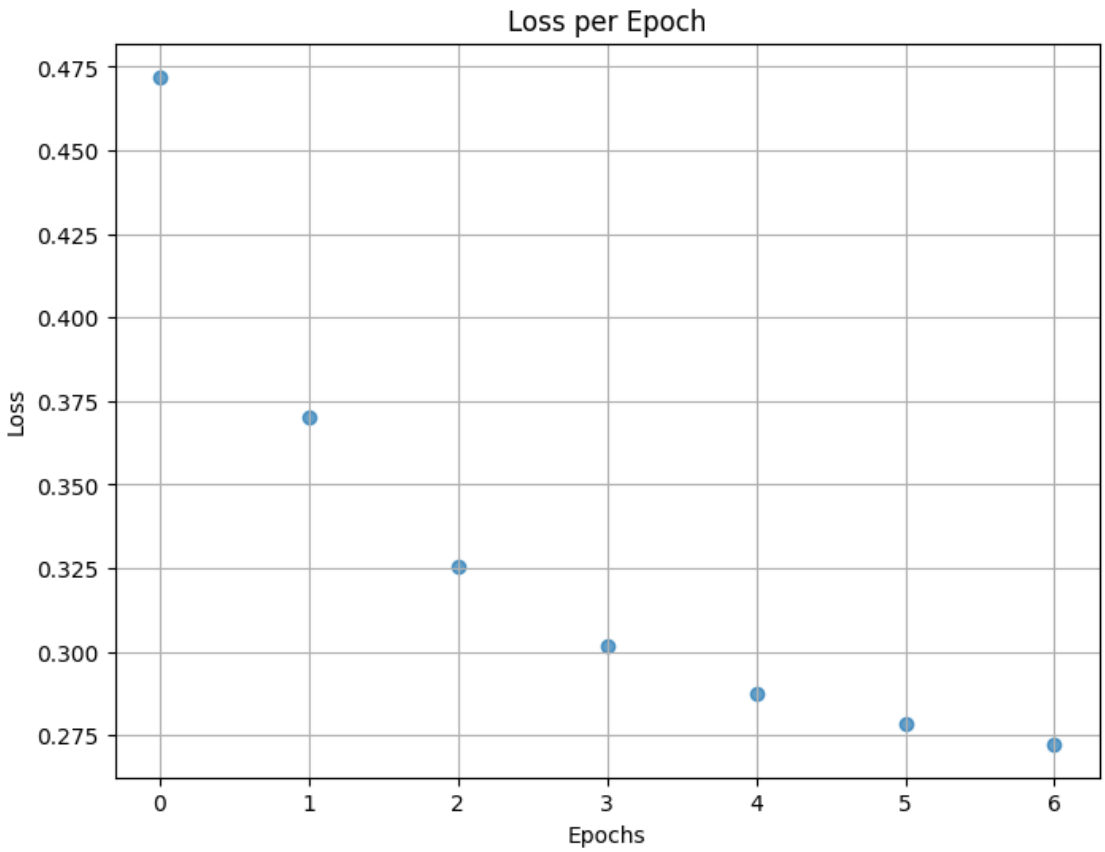

```
Test Error:
 Accuracy: 90.0%, Avg loss: 0.301626

Epoch 5
-------------------------------
loss: 0.308612  [   64/15253]
loss: 0.365436  [ 6464/15253]
loss: 0.306604  [12864/15253]
```



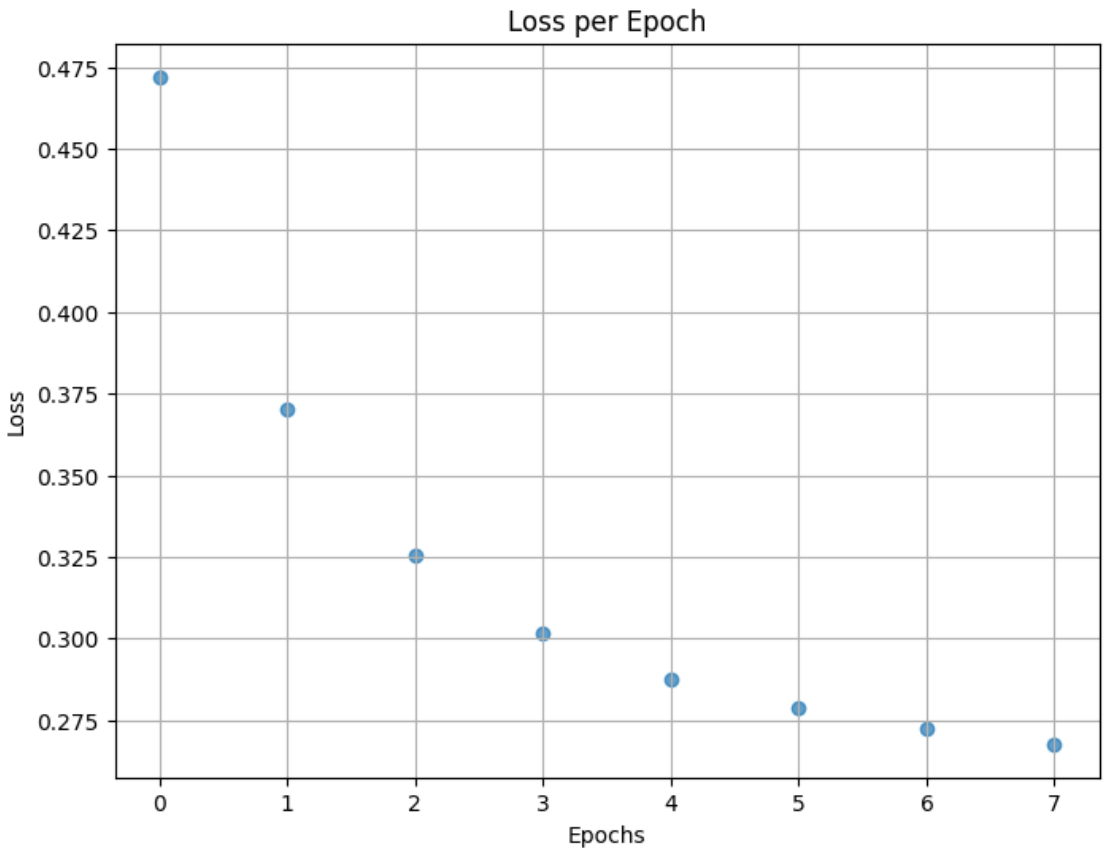

```
Test Error:
 Accuracy: 90.1%, Avg loss: 0.287642

Epoch 6
-------------------------------
loss: 0.250827  [   64/15253]
loss: 0.303448  [ 6464/15253]
loss: 0.250208  [12864/15253]
```



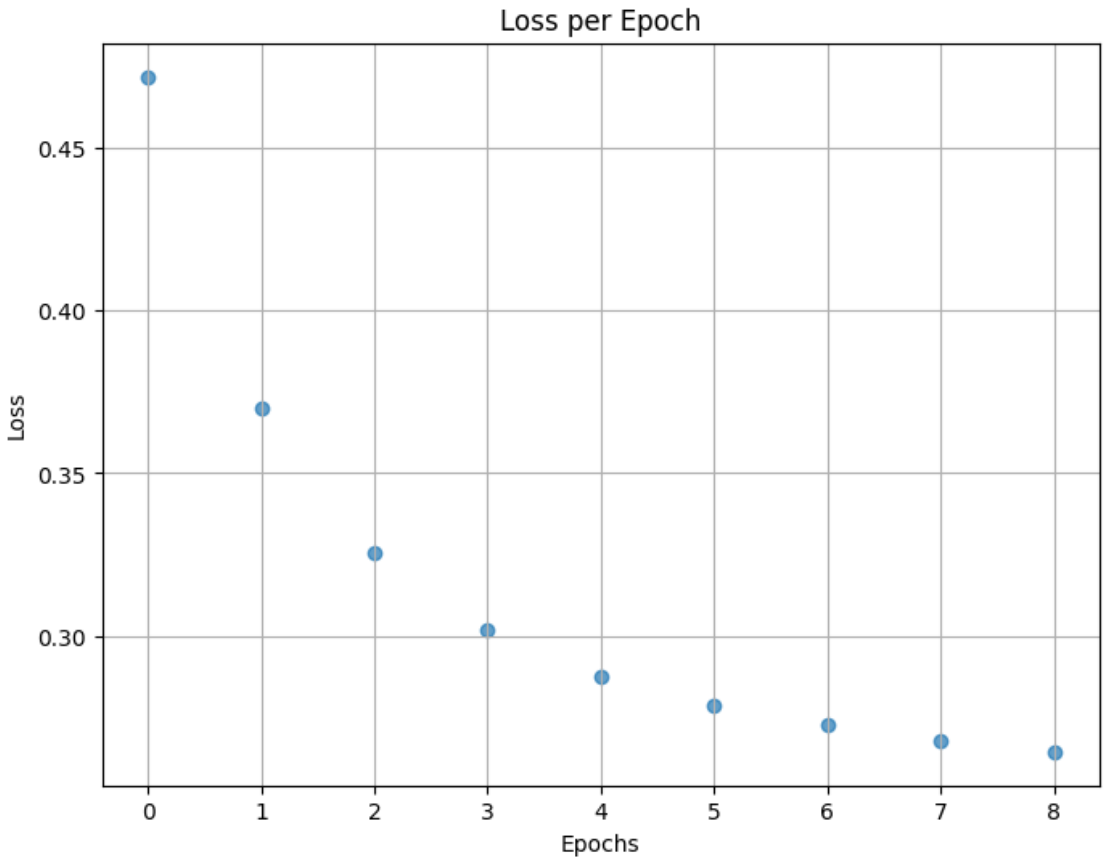

```
Test Error:
 Accuracy: 90.1%, Avg loss: 0.278643

Epoch 7
-------------------------------
loss: 0.181801  [   64/15253]
loss: 0.258912  [ 6464/15253]
loss: 0.238733  [12864/15253]
```



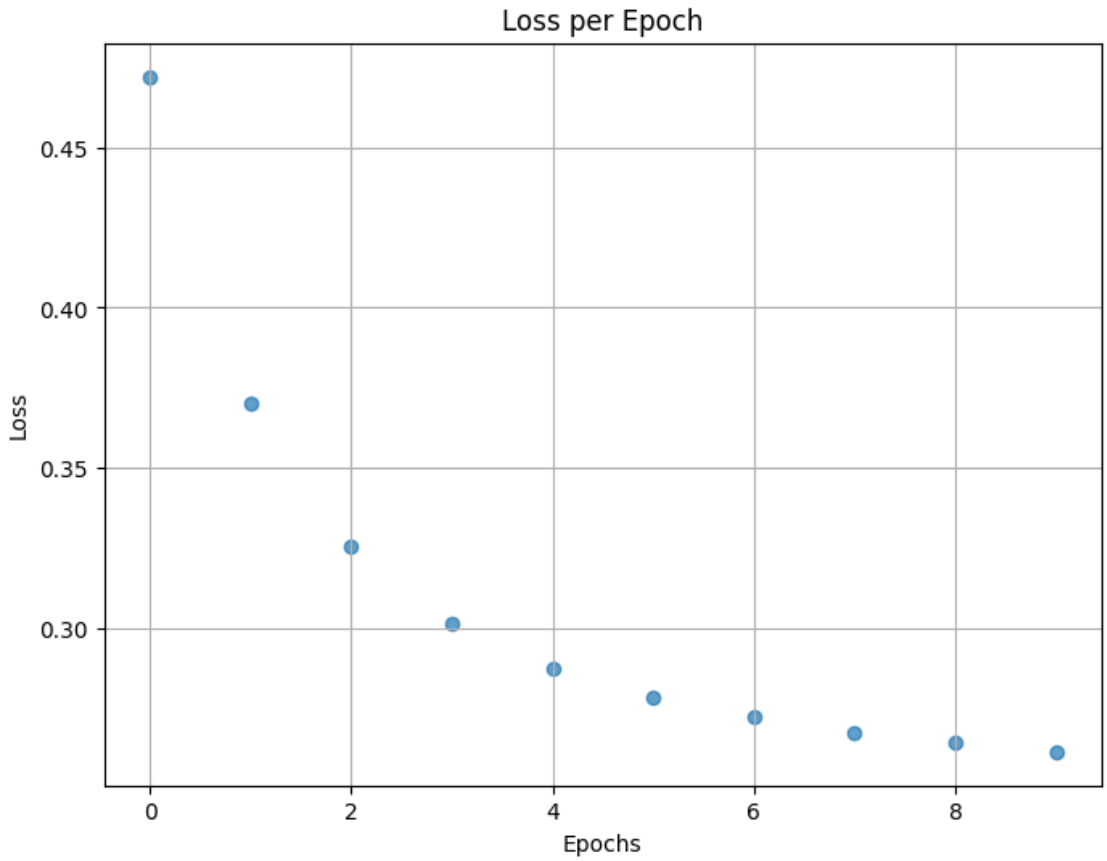

```
Test Error:
 Accuracy: 90.2%, Avg loss: 0.272408

Epoch 8
-------------------------------
loss: 0.247495  [   64/15253]
loss: 0.307326  [ 6464/15253]
loss: 0.289869  [12864/15253]
```




```
Test Error:
 Accuracy: 90.3%, Avg loss: 0.267715

Epoch 9
-------------------------------
loss: 0.295637  [   64/15253]
loss: 0.232339  [ 6464/15253]
loss: 0.266307  [12864/15253]
```




```
Test Error:
 Accuracy: 90.3%, Avg loss: 0.264333

Epoch 10
-------------------------------
loss: 0.267383  [   64/15253]
loss: 0.266995  [ 6464/15253]
loss: 0.284033  [12864/15253]
```




```
Test Error:
 Accuracy: 90.3%, Avg loss: 0.261657

Done!
```

[ ]:
```
# %%

# rf, reg = export_model()

# pred_rf = rf.predict(X_test)
# pred_reg = reg.predict(X_test)

# # Generate predictions for the neural network on the test set
# pred_neural = predict(model, X_test)

# compare(
#     [
#         classification_report(y_test, pred_rf, output_dict=True),
#         classification_report(y_test, pred_reg, output_dict=True),
```

```
#         classification_report(y_test, (pred_neural > 0.5).astype(float),␣
 ↪output_dict=True)
#     ],
#     model_names=["Random Forest", "Logistic Regression", "Neural Network"],
#     print_output=True,
# )
```

[ ]:
```
# %%

torch.save(model.state_dict(), "model.pth")
```