```
import torch
from torch import optim
import torchvision
import matplotlib.pyplot as plt
import pt_deep
import data
import numpy as np
from sklearn.svm import SVC

dataset_root = "./mnist"
mnist_train = torchvision.datasets.MNIST(dataset_root, train=True, download=False)
mnist_test = torchvision.datasets.MNIST(dataset_root, train=False, download=False)

x_train, y_train = mnist_train.data, mnist_train.targets
x_test, y_test = mnist_test.data, mnist_test.targets
x_train, x_test = x_train.double().div_(255.0), x_test.double().div_(255.0)

N = x_train.shape[0]
D = x_train.shape[1] * x_train.shape[2]
C = y_train.max().add_(1).item()

x_train_pt = torch.flatten(x_train, 1, 2)
y_train_pt_oh = torch.from_numpy(data.class_to_onehot(y_train))
x_test_pt = torch.flatten(x_test, 1, 2)
y_test_pt = y_test

x_train_np = x_train_pt.detach().numpy()
y_train_np = y_train.detach().numpy()
x_test_np = x_test_pt.detach().numpy()
y_test_np = y_test.detach().numpy()

def eval_and_print(model, x_train, y_train, x_test, y_test):
    """Doesn't save."""
    probs_train = model.eval(x_train)
    probs_test = model.eval(x_test)
    y_predicted_train = probs_train.argmax(axis=1)
    y_predicted_test = probs_test.argmax(axis=1)

    train_accuracy, train_pr, train_conf_m = (
        data.eval_perf_multi(y_predicted_train, y_train)
    )
    test_accuracy, test_pr, test_conf_m = (
        data.eval_perf_multi(y_predicted_test, y_test)
    )

    print("train")
    print("Accuracy: ", train_accuracy)
    print("Precision / Recall: ", train_pr)
    print("Confussion Matrix:\n", train_conf_m)
    print("test")
    print("Accuracy: ", test_accuracy)
    print("Precision / Recall: ", test_pr)
    print("Confussion Matrix:\n", test_conf_m)


def load_from_file(path):
    model_dict = torch.load(path)
    model = pt_deep.PTDeep([784, 10], activations=model_dict['activations'])
    model.weights = model_dict['weights']
    model.biases = model_dict['biases']
    model.loss_trace = model_dict['loss_trace']
    return model

def save_to_file(model, path):
    torch.save({
        'activations': model.activations,
```

```
        'weights': model.weights,
        'biases': model.biases,
        'loss_trace': model.loss_trace
    }, path)

def zad1_train():
    # [784, 10]
    model1 = pt_deep.PTDeep([784, 10], [pt_deep.my_softmax], param_lambda=0.05)
    model1.train(x_train_pt, y_train_pt_oh, param_niter=2_000, param_delta=1e-4, print_frequency=100, early_stopping=True)
    save_to_file(model1, './models/mnist_784_10_early_stopping_mb.pt')
    # [784, 100, 10]
    model2 = pt_deep.PTDeep([784, 100, 10], [torch.sigmoid, pt_deep.my_softmax])
    model2.train(x_train_pt, y_train_pt_oh, param_niter=10_000, param_delta=1e-4, print_frequency=100, early_stopping=True)
    save_to_file(model2, './models/mnist_784_100_10_early_stopping_mb.pt')


def show_worst_images(model):
    if isinstance(model, str):
        model = load_from_file(model)
    # dobiti one koji najvise pridonose gubitku
    y_test_np_oh = data.class_to_onehot(y_test_np)
    probs = model.eval(x_test_np)
    correct_class_probs = np.sum(probs * y_test_np_oh, axis=1)
    min_prob_indexes = np.argsort(correct_class_probs)
    fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2)
    axes = [ax1, ax2, ax3, ax4]
    for i in range(len(axes)):
        axes[i].imshow(x_test[min_prob_indexes[i]], cmap='gray')
        axes[i].set_title(y_test_np[min_prob_indexes[i]])
    plt.show()

def zad3():
    """Ovo je zapravo vec onaj s Adam optimizerom."""
    # [784, 10]
    model1 = pt_deep.PTDeep([784, 10], [pt_deep.my_softmax], param_lambda=0.05)
    optimizer = optim.Adam(model1.parameters(), lr=1e-4, weight_decay=0.05)
    model1.train_mb(x_train_pt, y_train_pt_oh, param_niter=2_000, print_frequency=100,
                    early_stopping=True, optimizer=optimizer)
    save_to_file(model1, './models/mnist_784_10_early_stopping_reg005_mb_Adam.pt')
    show_worst_images(model1)
    # [784, 100, 10]
    model2 = pt_deep.PTDeep([784, 100, 10], [torch.sigmoid, pt_deep.my_softmax])
    optimizer = optim.Adam(model2.parameters(), lr=1e-4, weight_decay=0.05)
    model2.train_mb(x_train_pt, y_train_pt_oh, param_niter=10_000, print_frequency=100,
                    early_stopping=True, optimizer=optimizer)
    save_to_file(model1, './models/mnist_784_100_10_early_stopping_reg005_mb_Adam.pt')
    show_worst_images(model2)

def zad4():
    """Onaj s varijabilnim learning rate-om"""
    model = pt_deep.PTDeep([784, 100, 10], [torch.sigmoid, pt_deep.my_softmax])
    optimizer = optim.Adam(model.parameters(), lr=1e-4)
    scheduler = optim.lr_scheduler.ExponentialLR(optimizer, gamma=1-1e-4)
    model.train_mb(x_train_pt, y_train_pt_oh, param_niter=2_000, print_frequency=50,
                   early_stopping=True, optimizer=optimizer, scheduler=scheduler)
    save_to_file(model, "./models/mnist_784_100_10_sve_ExponentialLR.pt")
    eval_and_print(model, x_train_np, y_train_np, x_test_np, y_test_np)
    show_worst_images(model)


def zad5():
    """Neistrenirani model treba probat."""
    model = pt_deep.PTDeep([784, 100, 10], [torch.sigmoid, pt_deep.my_softmax])
    eval_and_print(model, x_train_np, y_train_np, x_test_np, y_test_np)
    show_worst_images(model)
```

```python
def zad6(kernel='linear'):
    """SVM Linearni i jezgreni."""
    svm = SVC(kernel=kernel).fit(x_train_np, y_train_np)
    y_predicted_train = svm.predict(x_train_np)
    y_predicted_test = svm.predict(x_test_np)

    train_accuracy, train_pr, train_conf_m = (
        data.eval_perf_multi(y_predicted_train, y_train)
    )
    test_accuracy, test_pr, test_conf_m = (
        data.eval_perf_multi(y_predicted_test, y_test)
    )

    print("train")
    print("Accuracy: ", train_accuracy)
    print("Precision / Recall: ", train_pr)
    print("Confussion Matrix:\n", train_conf_m)
    print("test")
    print("Accuracy: ", test_accuracy)
    print("Precision / Recall: ", test_pr)
    print("Confussion Matrix:\n", test_conf_m)
```