```
import random
import numpy as np
import pandas as pd
import torch
import torch.nn as nn

from sklearn.model_selection import train_test_split


random.seed(0)
np.random.seed(0)
torch.manual_seed(0)
torch.use_deterministic_algorithms(True)

data = pd.read_csv('/datasets/Electrical_Grid_Stability.csv', sep=';')

X_train, X_test, y_train, y_test = train_test_split(
    data.drop(columns=['stability']),
    data.stability,
    test_size=0.3,
    shuffle=True)

X_train = torch.FloatTensor(X_train.values)
X_test = torch.FloatTensor(X_test.values)
y_train = torch.FloatTensor(y_train.values)
y_test = torch.FloatTensor(y_test.values)

n_neurons = [12, 9, 6, 3, 1]
net_layers = []

for i in range(1, len(n_neurons) - 1):
           net_layers.append(...)
           if (i+1) % 2 == 0:
                      net_layers.append(...)
           else:
                      net_layers.append(...)

net_layers.append(nn.Linear(n_neurons[-2], n_neurons[-1]))
net_layers.append(...)

net = nn.Sequential(*net_layers) # такая запись позволяет передавать
элементы списка как параметры для инициализации

def init_weights(layer):
    if type(layer) == ...: # Проверка, что слой – полносвязный
        nn.init....(layer.weight, ...)
        nn.init....(layer.bias, ...)

net.apply(init_weights)

optimizer = torch.optim.Adam(net.parameters(), lr=1e-3)

loss = nn.BCELoss()
```

```
num_epochs = 100

for epoch in range(num_epochs):
    optimizer.zero_grad()

    preds = net.forward(X_train).flatten()

    loss_value = loss(preds, y_train)
    # print(loss_value)

    loss_value.backward()

    optimizer.step()

    if epoch % 10 == 0 or epoch == num_epochs - 1:
        net.eval()
        test_preds = net.forward(X_test)
        accuracy = (torch.round(test_preds) ==
y_test).float().mean().data
        print(accuracy)
```