Министерство образования Республики Беларусь
Учреждение образования
«Брестский Государственный технический университет»
Кафедра ИИТ

Лабораторная работа №4
По дисциплине «Основы машинного обучения»
Тема: **«Введение в нейронные сети:
построение многослойного перцептрона»**

Выполнил:
Студент 3 курса
Группы АС-65
Ярмак К. А.
Проверил:
Крощенко А. А.

Брест 2025

Цель: построить, обучить и оценить многослойный перцептрон (MLP) для решения задачи классификации.

Вариант 12

1.Импорт библиотек и подготовка данных

- импортируйте torch, torch.nn, torch.optim, а также sklearn для загрузки данных и их предобработки;
- загрузите датасет, выполните стандартизацию (StandardScaler) и кодирование признаков;
- разделите данные на обучающую и тестовую выборки;
- преобразуйте данные (признаки и метки) в тензоры PyTorch: torch.tensor(X_train, dtype=torch.float32).

2. Определение архитектуры нейронной сети

- создайте класс, наследуемый от torch.nn.Module;
- в методе __init__ определите все слои, которые будете использовать (например, nn.Linear, nn.ReLU, nn.Dropout);
- в методе forward опишите последовательность применения слоев к входным данным.

3. Инициализация модели, функции потерь и оптимизатора

- создайте экземпляр вашей модели: model = MLP();
- определите функцию потерь. Для бинарной классификации используйте nn.BCEWithLogitsLoss, для многоклассовой – nn.CrossEntropyLoss;
- определите оптимизатор: optimizer = torch.optim.Adam(model.parameters(), lr=0.001).

4. Написание цикла обучения (Training Loop)

- запустите цикл на определенное количество эпох;
- внутри цикла:
    1. переведите модель в режим обучения: model.train();
    2. сделайте предсказание (forward pass): y_pred = model(X_train);
    3. рассчитайте потери (loss): loss = criterion(y_pred, y_train);
    4. обнулите градиенты: optimizer.zero_grad();
    5. выполните обратное распространение loss.backward();
    6. сделайте шаг оптимизации: optimizer.step().

5. Оценка модели (Evaluation)

- переведите модель в режим оценки: model.eval();
- используйте with torch.no_grad():, чтобы отключить расчет градиентов;
- сделайте предсказания на тестовых данных;
- преобразуйте выходные данные (логиты) в предсказанные классы (например, с помощью torch.argmax или проверки порога > 0);
- рассчитайте метрики (accuracy, f1-score и т.д.), используя sklearn.metrics.

Детекция аномалий в сети
KDD Cup 1999
Задача: классифицировать подключения на "нормальные" и "атаки" (бинарная классификация).
Архитектура:

- входной слой;
- один скрытый слой с 64 нейронами (ReLU);
- выходной слой с 1 нейроном (Sigmoid).

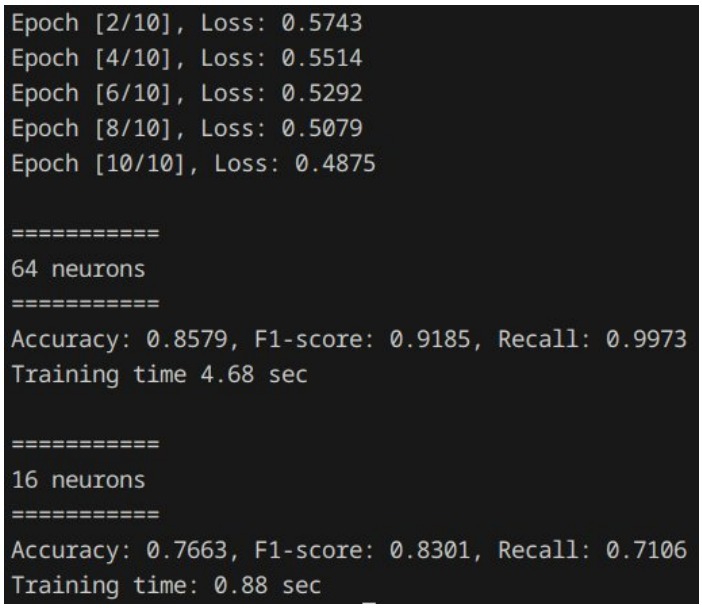
Эксперимент: уменьшите количество нейронов до 16. Насколько сильно упала точность и как изменилось время обучения?

Код программы:

```
import pandas as pd
import time
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score, f1_score, recall_score
import torch
import torch.nn as nn
import torch.optim as optim

from columns import file_path, columns


# 1 data prepare
df = pd.read_csv(file_path, names=columns)

cat_cols = ['protocol_type', 'service', 'flag']
encoders = {col: LabelEncoder() for col in cat_cols}
for col in cat_cols:
    df[col] = encoders[col].fit_transform(df[col])

df['target'] = df['label'].apply(lambda x: 0 if x == 'normal.' else 1)
df = df.drop('label', axis=1)

X = df.drop('target', axis=1)
y = df['target']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

torch.manual_seed(42)
X_train = torch.tensor(X_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train.values, dtype=torch.float32).view(-1, 1)
y_test = torch.tensor(y_test.values, dtype=torch.float32).view(-1, 1)


# 2 class(nn.Modle)
class MLP(nn.Module):
    def __init__(self, input_dim, hidden_dim):
```

```
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_dim, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.sigmoid(x)
        return x


# 3 model init 64
input_dim = X_train.shape[1]
hidden_dim = 64

model = MLP(input_dim, hidden_dim)
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)


# 4 model training
epochs = 10
start_time = time.time()

for epoch in range(epochs):
    model.train()
    y_pred = model(X_train)
    loss = criterion(y_pred, y_train)

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

    if (epoch + 1) % 2 == 0:
        print(f"Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}")

train_time = time.time() - start_time


# 5 model eval
model.eval()
with torch.no_grad():
    y_pred_test = model(X_test)
    y_pred_classes = (y_pred_test > 0.5).float()

acc = accuracy_score(y_test, y_pred_classes)
f1 = f1_score(y_test, y_pred_classes)
recall = recall_score(y_test, y_pred_classes)

print("\n==========\n64 neurons\n==========")
print(f"Accuracy: {acc:.4f}, F1-score: {f1:.4f}, Recall: {recall:.4f}")
print(f"Training time {train_time:.2f} sec")


# 6 experiment - 16 neur
hidden_dim_small = 16
model_small = MLP(input_dim, hidden_dim_small)
criterion_small = nn.BCELoss()
optimizer_small = optim.Adam(model_small.parameters(), lr=0.001)
```

```
start_time = time.time()
for epoch in range(epochs):
    model_small.train()
    y_pred = model_small(X_train)
    loss = criterion_small(y_pred, y_train)

    optimizer_small.zero_grad()
    loss.backward()
    optimizer_small.step()

train_time_small = time.time() - start_time

model_small.eval()
with torch.no_grad():
    y_pred_test = model_small(X_test)
    y_pred_classes = (y_pred_test > 0.5).float()

acc_small = accuracy_score(y_test, y_pred_classes)
f1_small = f1_score(y_test, y_pred_classes)
recall_small = recall_score(y_test, y_pred_classes)

print("\n==========\n16 neurons\n==========")
print(f"Accuracy: {acc_small:.4f}, F1-score: {f1_small:.4f}, Recall:
{recall_small:.4f}")
print(f"Training time: {train_time_small:.2f} sec")
```



После уменьшения нейронов до 16 accuracy упала приблизительно на 11%, F1-score упал приблизительно на 10%, а время обучения сократилось примерно в 5 раз.

Вывод: построили, обучили и оценили многослойный перцептрон (MLP) для решения задачи классификации.