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

Лабораторная работа №4
По дисциплине: «ОМО»
Тема : “Введение в нейронные сети:
построение многослойного перцептрона”

Выполнил:
Студент 3-го курса
Группы АС-66
Цеван К.А.
Проверил:
Крощенко А.А.

Брест 2025

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

Вариант 12

Вариант 12 Детекция аномалий в сети
• KDD Cup 1999

Основы машинного обучения, 2025, Крощенко А.А.
• Задача: классифицировать подключения на "нормальные" и "атаки" (бинарная классификация).
• Архитектура:
o входной слой;
o один скрытый слой с 64 нейронами (ReLU);
o выходной слой с 1 нейроном (Sigmoid).
• Эксперимент: уменьшите количество нейронов до 16. Насколько сильно упала точность и как изменилось время обучения?

```
# -*- coding: utf-8 -*-

import time
import random
import warnings
from typing import Dict, List

warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, f1_score, recall_score, precision_score

import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader

DATA_PATH = "kddcup.data_10_percent_corrected"
SAMPLE_SIZE = 5000
TEST_SIZE = 0.30
BATCH_SIZE = 128
EPOCHS = 50
LR = 1e-3
RANDOM_STATE = 42

DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")

def set_seed(seed: int = 42) -> None:
```

```
    np.random.seed(seed)
    random.seed(seed)
    torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed)

COLUMNS = [
    "duration","protocol_type","service","flag","src_bytes","dst_bytes","land","wrong_fragment","urgent",
    "hot","num_failed_logins","logged_in","num_compromised","root_shell","su_attempted","num_root",
    "num_file_creations","num_shells","num_access_files","num_outbound_cmds","is_host_login","is_guest_login",
    "count","srv_count","serror_rate","srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate",

"diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count","dst_host_same_srv_rate",

"dst_host_diff_srv_rate","dst_host_same_src_port_rate","dst_host_srv_diff_host_rate","dst_host_serror_rate",
    "dst_host_srv_serror_rate","dst_host_rerror_rate","dst_host_srv_rerror_rate","label"
]

CAT_COLS = ["protocol_type", "service", "flag"]

def load_and_prepare(df_path: str) -> pd.DataFrame:
    df = pd.read_csv(df_path, names=COLUMNS)
    df["target"] = (df["label"] != "normal.").astype(int)
    df_num = pd.get_dummies(df.drop(columns=["label"]), columns=CAT_COLS)
    return df_num

def stratified_sample(df: pd.DataFrame, n_samples: int, seed: int) -> pd.DataFrame:
    if n_samples < len(df):
        sample, _ = train_test_split(df, train_size=n_samples, stratify=df["target"], random_state=seed)
        return sample
    return df.copy()

def train_test_tensors(X: np.ndarray, y: np.ndarray, test_size: float, seed: int):
    X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=test_size, stratify=y, random_state=seed)
    return (
        torch.tensor(X_tr, dtype=torch.float32),
        torch.tensor(X_te, dtype=torch.float32),
        torch.tensor(y_tr, dtype=torch.float32).unsqueeze(1),
        torch.tensor(y_te, dtype=torch.float32).unsqueeze(1)
```

```
    )

class MLP(nn.Module):
    def __init__(self, input_dim: int, hidden_dim: int, p_dropout: float = 0.2):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.ReLU(),
            nn.Dropout(p_dropout),
            nn.Linear(hidden_dim, 1)
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        return self.net(x)

@torch.no_grad()
def evaluate(model: nn.Module, loader: DataLoader) -> Dict[str, float]:
    model.eval()
    preds, targs = [], []
    for xb, yb in loader:
        xb, yb = xb.to(DEVICE), yb.to(DEVICE)
        probs = torch.sigmoid(model(xb))
        preds.append((probs > 0.5).float().cpu().numpy().reshape(-1))
        targs.append(yb.cpu().numpy().reshape(-1))
    y_pred = np.concatenate(preds)
    y_true = np.concatenate(targs)
    return {
        "accuracy": accuracy_score(y_true, y_pred),
        "f1": f1_score(y_true, y_pred),
        "recall": recall_score(y_true, y_pred),
        "precision": precision_score(y_true, y_pred),
    }

def fit_one_hidden(hidden_size: int, train_loader: DataLoader, test_loader: DataLoader, in-
put_dim: int):
    model = MLP(input_dim=input_dim, hidden_dim=hidden_size).to(DEVICE)
    criterion = nn.BCEWithLogitsLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=LR)
    for _ in range(EPOCHS):
        model.train()
        for xb, yb in train_loader:
            xb, yb = xb.to(DEVICE), yb.to(DEVICE)
            loss = criterion(model(xb), yb)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
    metrics = evaluate(model, test_loader)
```

```
    return {"hidden_size": hidden_size, **metrics}

def main():
    set_seed(RANDOM_STATE)
    df_num = load_and_prepare(DATA_PATH)
    df_sample = stratified_sample(df_num, SAMPLE_SIZE, RANDOM_STATE)
    X = df_sample.drop(columns=["target"]).values
    y = df_sample["target"].values
    X = StandardScaler().fit_transform(X)
    X_train_t, X_test_t, y_train_t, y_test_t = train_test_tensors(X, y, TEST_SIZE, RAN-
DOM_STATE)
    train_loader = DataLoader(TensorDataset(X_train_t, y_train_t),
batch_size=BATCH_SIZE, shuffle=True)
    test_loader = DataLoader(TensorDataset(X_test_t, y_test_t), batch_size=BATCH_SIZE,
shuffle=False)
    input_dim = X_train_t.shape[1]
    h1 = fit_one_hidden(64, train_loader, test_loader, input_dim)
    h2 = fit_one_hidden(16, train_loader, test_loader, input_dim)
    print(h1)
    print(h2)

if __name__ == "__main__":
    main()
```

Вывод: построили, обучили и оценили многослойный перцептрон (MLP) для решения задачи классификациина практике сравнили работу несколько алгоритмов классификации.