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
Савинец М.Д.
Проверил:
Крощенко А.А.

Брест 2025

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

Вариант 11

Анализ маркетинговой кампании банка
• Bank Marketing UCI
• Задача: предсказать, согласится ли клиент на вклад (бинарная классификация).
• Архитектура:
о входной слой;
о один скрытый слой с 20 нейронами (ReLU);
о выходной слой с 1 нейроном (Sigmoid).
• Эксперимент: попробуйте использовать оптимизатор RMSprop вместо Adam. Сравните итоговый F1-score.

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, classification_report
import warnings
import matplotlib.pyplot as plt

warnings.filterwarnings('ignore')

print("1. Загрузка и подготовка данных...")

df = pd.read_csv('bank.csv', delimiter=';')

categorical_columns = ['job', 'marital', 'education', 'default', 'housing',
'loan', 'contact', 'month', 'poutcome', 'y']
label_encoders = {}

for col in categorical_columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col].astype(str))
    label_encoders[col] = le

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

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

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

X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train.values, dtype=torch.float32).reshape(-
1, 1)
y_test_tensor = torch.tensor(y_test.values, dtype=torch.float32).reshape(-1,
1)
```

```
print(f"Размерность данных: X_train: {X_train_tensor.shape}, y_train:
{y_train_tensor.shape}")
print(f"Размерность данных: X_test: {X_test_tensor.shape}, y_test:
{y_test_tensor.shape}")

class MLP(nn.Module):
    def __init__(self, input_size, hidden_size=20, output_size=1):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, output_size)
        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

input_size = X_train.shape[1]
hidden_size = 20
output_size = 1

model_adam = MLP(input_size, hidden_size, output_size)
criterion = nn.BCELoss()
optimizer_adam = optim.Adam(model_adam.parameters(), lr=0.001)

model_rmsprop = MLP(input_size, hidden_size, output_size)
optimizer_rmsprop = optim.RMSprop(model_rmsprop.parameters(), lr=0.001)

print(f"\n2. Архитектура сети:")
print(f"   - Входной слой: {input_size} нейронов")
print(f"   - Скрытый слой: {hidden_size} нейронов (ReLU)")
print(f"   - Выходной слой: {output_size} нейрон (Sigmoid)")

print("\n3. Обучение модели с оптимизатором Adam...")
epochs = 100
train_losses_adam = []

for epoch in range(epochs):
    model_adam.train()

    y_pred = model_adam(X_train_tensor)
    loss = criterion(y_pred, y_train_tensor)

    optimizer_adam.zero_grad()
    loss.backward()
    optimizer_adam.step()

    train_losses_adam.append(loss.item())

    if (epoch + 1) % 20 == 0:
        print(f'   Epoch [{epoch + 1}/{epochs}], Loss: {loss.item():.4f}')

print("\n4. Обучение модели с оптимизатором RMSprop...")
train_losses_rmsprop = []

for epoch in range(epochs):
    model_rmsprop.train()

    y_pred = model_rmsprop(X_train_tensor)
    loss = criterion(y_pred, y_train_tensor)
```

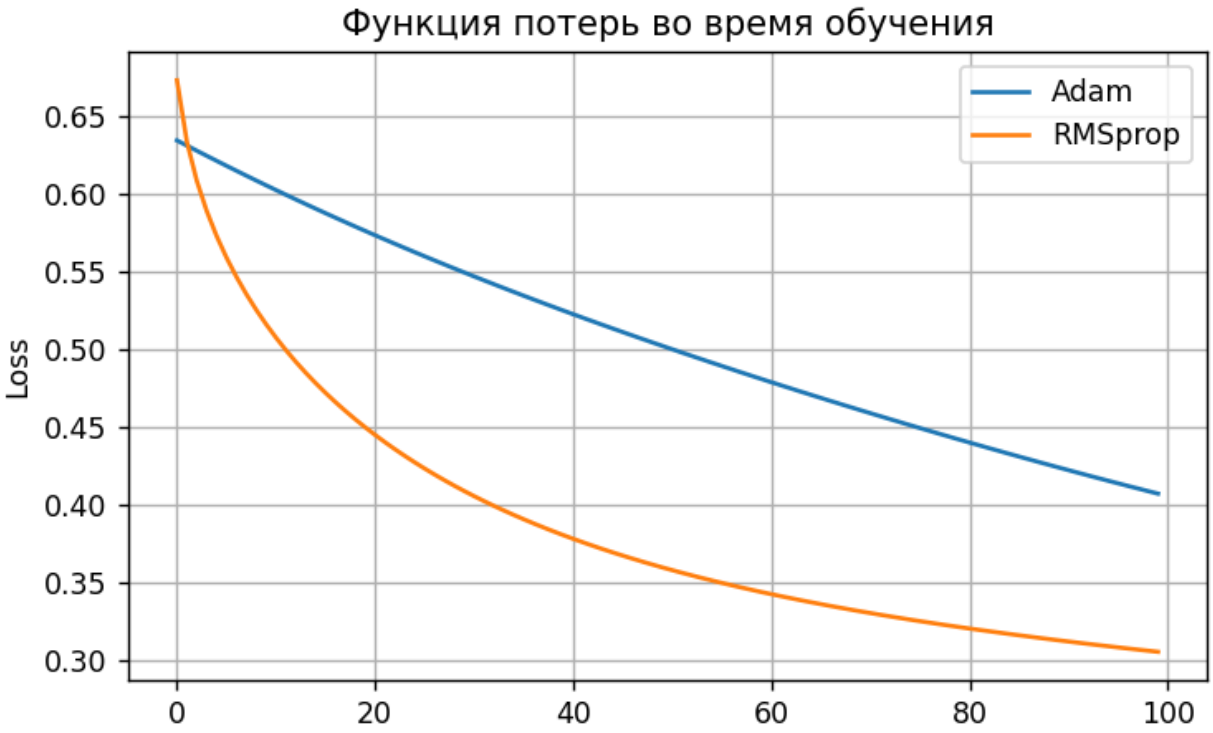
```
    optimizer_rmsprop.zero_grad()
    loss.backward()
    optimizer_rmsprop.step()

    train_losses_rmsprop.append(loss.item())

    if (epoch + 1) % 20 == 0:
        print(f'   Epoch [{epoch + 1}/{epochs}], Loss: {loss.item():.4f}')

print("\n5. Оценка моделей на тестовых данных...")

def evaluate_model(model, X_test, y_test):
    model.eval()
    with torch.no_grad():
        y_pred_proba = model(X_test)
        y_pred = (y_pred_proba > 0.5).float()

        accuracy = accuracy_score(y_test.numpy(), y_pred.numpy())
        f1 = f1_score(y_test.numpy(), y_pred.numpy())

    return accuracy, f1, y_pred

accuracy_adam, f1_adam, y_pred_adam = evaluate_model(model_adam,
X_test_tensor, y_test_tensor)

accuracy_rmsprop, f1_rmsprop, y_pred_rmsprop = evaluate_model(model_rmsprop,
X_test_tensor, y_test_tensor)

print("\n6. Результаты сравнения оптимизаторов:")
print("=" * 50)
print(f"{'Метрика':<15} {'Adam':<10} {'RMSprop':<10}")
print("-" * 50)
print(f"{'Accuracy':<15} {accuracy_adam:.4f}    {accuracy_rmsprop:.4f}")
print(f"{'F1-Score':<15} {f1_adam:.4f}    {f1_rmsprop:.4f}")
print("=" * 50)

print("\n7. Детальный отчет по классификации:")
print("\nAdam Optimizer:")
print(classification_report(y_test_tensor.numpy(), y_pred_adam.numpy(),
target_names=['No Deposit', 'Deposit']))

print("\nRMSprop Optimizer:")
print(classification_report(y_test_tensor.numpy(), y_pred_rmsprop.numpy(),
target_names=['No Deposit', 'Deposit']))

print("\n8. Анализ результатов:")
if f1_adam > f1_rmsprop:
    print(f"Оптимизатор Adam показал лучший F1-Score: {f1_adam:.4f} против
{f1_rmsprop:.4f} у RMSprop")
    print("Разница: {:.4f}".format(f1_adam - f1_rmsprop))
elif f1_rmsprop > f1_adam:
    print(f"Оптимизатор RMSprop показал лучший F1-Score: {f1_rmsprop:.4f}
против {f1_adam:.4f} у Adam")
    print("Разница: {:.4f}".format(f1_rmsprop - f1_adam))
else:
    print("Оба оптимизатора показали одинаковый F1-Score")

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(train_losses_adam, label='Adam')
plt.plot(train_losses_rmsprop, label='RMSprop')
plt.title('Функция потерь во время обучения')
```

```
plt.xlabel('Эпоха')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

plt.subplot(1, 2, 2)
optimizers = ['Adam', 'RMSprop']
f1_scores = [f1_adam, f1_rmsprop]
plt.bar(optimizers, f1_scores, color=['blue', 'orange'])
plt.title('Сравнение F1-Score')
plt.ylabel('F1-Score')
plt.ylim(0, 1)

for i, v in enumerate(f1_scores):
    plt.text(i, v + 0.01, f'{v:.4f}', ha='center', va='bottom')

plt.tight_layout()
plt.show()
```

Результат:

```
2. Архитектура сети:
    - Входной слой: 16 нейронов
    - Скрытый слой: 20 нейронов (ReLU)
    - Выходной слой: 1 нейрон (Sigmoid)
```

График изменения ошибок:



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