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»

Выполнила:
Студентка 3 курса
Группы АС-65
Рапин Е. Ю.
Проверил:
Крощенко А. А.

Брест 2025

Цель работы: 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:**
  - входной слой;
  - один скрытый слой с 20 нейронами (ReLU);
  - выходной слой с 1 нейроном (Sigmoid).
- **Эксперимент:** попробуйте использовать оптимизатор RMSprop вместо Adam. Сравните итоговый F1-score.

1. Импорт библиотек и подготовка данных

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

data = pd.read_csv('bank.csv', comment='#', header=None)

# Категориальные признаки
categorical_columns = data.select_dtypes(include=['object']).columns
print(f"\nКатегориальные признаки: {list(categorical_columns)}")

label_encoders = {}
for column in categorical_columns:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column].astype(str))
    label_encoders[column] = le
    print(f"{column}: {len(le.classes_)} классов")

# Обновляем X и y после кодирования
X = data.iloc[:, :-1].copy()
y = data.iloc[:, -1].copy()

# Стандартизация числовых признаков
numeric_columns = X.select_dtypes(include=[np.number]).columns
scaler = StandardScaler()
if len(numeric_columns) > 0:
    X[numeric_columns] = scaler.fit_transform(X[numeric_columns])

# Кодирование целевой переменной
label_encoder_y = LabelEncoder()
y_encoded = label_encoder_y.fit_transform(y)
print(f"\nУникальные значения целевой переменной: {np.unique(y_encoded)}")
print(f"Количество классов: {len(np.unique(y_encoded))}")
```

```
print("Распределение классов:", np.bincount(y_encoded))

# Разделение на train/test
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded,
test_size=0.2, random_state=42)

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

2. Определение архитектуры нейронной сети

```
class BankMarketingNN(nn.Module):
    def __init__(self, input_size, hidden_size=20):
        super(BankMarketingNN, self).__init__()
        self.layer1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.output = nn.Linear(hidden_size, 1)

    def forward(self, x):
        x = self.layer1(x)
        x = self.relu(x)
        x = self.output(x)
        return x
```

3. Инициализация модели, функции потерь и оптимизатора

```
input_size = X_train.shape[1]
model_adam = BankMarketingNN(input_size=input_size)
model_rmsprop = BankMarketingNN(input_size=input_size)

criterion = nn.BCEWithLogitsLoss()
optimizer_adam = optim.Adam(model_adam.parameters(), lr=0.001)
optimizer_rmsprop = optim.RMSprop(model_rmsprop.parameters(), lr=0.001)
```

4. Написание цикла обучения (Training Loop)

```
def train_and_evaluate(model, optimizer, criterion, X_train, y_train, X_test,
y_test, epochs=200):
    model.train()
    for epoch in range(epochs):
        outputs = model(X_train)
        loss = criterion(outputs, y_train)

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

        if epoch % 50 == 0:
            print(f'Epoch [{epoch}/{epochs}], Loss: {loss.item():.4f}')
```

5. Оценка модели (Evaluation)

```
model.eval()
with torch.no_grad():
    test_outputs = model(X_test)
    probabilities = torch.sigmoid(test_outputs)
```

```
    predicted = (probabilities > 0.75).float()

    y_test_np = y_test.numpy().flatten()
    predicted_np = predicted.numpy().flatten()

    if len(np.unique(y_test_np)) > 2:
        recall = recall_score(y_test_np, predicted_np, average='macro',
zero_division=0)
        precision = precision_score(y_test_np, predicted_np, average='macro',
zero_division=0)
        f1 = f1_score(y_test_np, predicted_np, average='macro',
zero_division=0)
    else:
        recall = recall_score(y_test_np, predicted_np, zero_division=0)
        precision = precision_score(y_test_np, predicted_np, zero_division=0)
        f1 = f1_score(y_test_np, predicted_np, zero_division=0)

    accuracy = accuracy_score(y_test_np, predicted_np)

return accuracy, recall, precision, f1
```

Эксперимент

```
print("\nОбучение модели с оптимизатором Adam:")
accuracy_adam, recall_adam, precision_adam, f1_adam = train_and_evaluate(
    model_adam, optimizer_adam, criterion, X_train_tensor, y_train_tensor,
X_test_tensor, y_test_tensor
)

print("\nОбучение модели с оптимизатором RMSprop:")
accuracy_rmsprop, recall_rmsprop, precision_rmsprop, f1_rmsprop =
train_and_evaluate(
    model_rmsprop, optimizer_rmsprop, criterion, X_train_tensor,
y_train_tensor, X_test_tensor, y_test_tensor
)

# Результаты
print(f"\nСравнение оптимизаторов:")
print(f"Adam - Accuracy: {accuracy_adam:.4f}, Recall: {recall_adam:.4f},
Precision: {precision_adam:.4f}, F1-score: {f1_adam:.4f}")
print(f"RMSprop - Accuracy: {accuracy_rmsprop:.4f}, Recall:
{recall_rmsprop:.4f}, Precision: {precision_rmsprop:.4f}, F1-score:
{f1_rmsprop:.4f}")
```

```
Сравнение оптимизаторов:
Adam - Accuracy: 0.5087, Recall: 0.3322, Precision: 0.1700, F1-score: 0.2249
RMSprop - Accuracy: 0.5101, Recall: 0.3330, Precision: 0.1701, F1-score: 0.2252
```

Вывод: построила, обучила и оценила **многослойный перцептрон (MLP)** для решения задачи классификации.