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

Лабораторная работа №4
По дисциплине «ОМО»

Выполнил:
Студент 3-го курса
Группы АС-65
Грущинский Д.Д.
Проверил:
Крощенко А.А

Брест 2025

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

Общий план для всех вариантов:

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 ошибки:
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.

Вариант 2

Диагностика рака груди

• Breast Cancer Wisconsin

• **Задача:** определить, является ли опухоль злокачественной (бинарная классификация).

• **Архитектура:**

о входной слой;

о один скрытый слой с 16 нейронами (ReLU);

о выходной слой с 1 нейроном (Sigmoid).

• Эксперимент: обучите модель с 32 нейронами в скрытом слое. Как изменились метрики precision и recall?

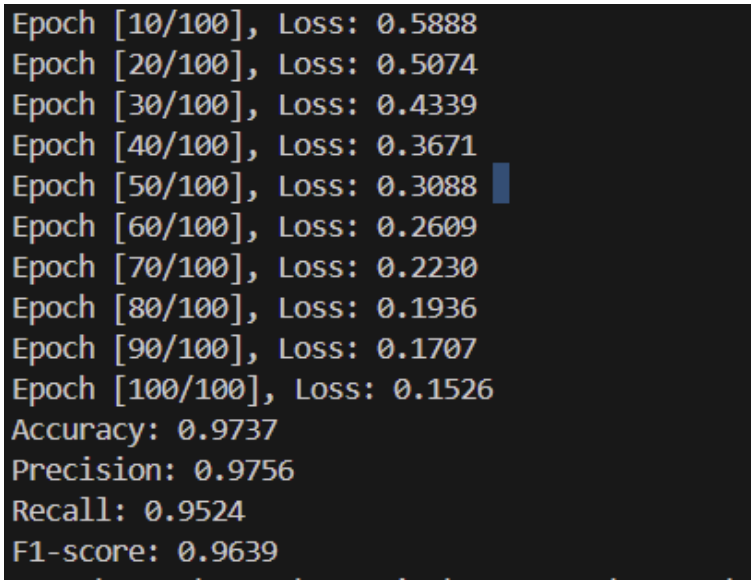
Ход работы

Код программы данной лабораторной работы:

```
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
import pandas as pd
import numpy as np
data_df = pd.read_csv(r'C:\Users\GRUDA\OneDrive\Документы\3
курс\ОМО\lab1_correct\ml_as65\reports\Грущинский\4\src\breast_cancer.csv').dr
opna()
data_df['diagnosis'] = data_df['diagnosis'].map({'M': 1, 'B': 0})
x = data_df.drop(['id', 'diagnosis'], axis=1)
y = data_df['diagnosis'].values
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
y_arr = np.array(y)
x_train, x_test, y_train, y_test = train_test_split(
    x_scaled, y, test_size=0.2, random_state=42, stratify=y_arr
)
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.reshape(-1, 1), dtype=torch.float32)
y_test_tensor = torch.tensor(y_test.reshape(-1, 1), dtype=torch.float32)
class MLP(nn.Module):
    def __init__(self, input_dim):
        super(MLP, self).__init__()
        self.hidden = nn.Linear(input_dim, 32)
        self.relu = nn.ReLU()
        self.output = nn.Linear(32, 1)
    def forward(self, x):
        x = self.hidden(x)
```

```
        x = self.relu(x)
        x = self.output(x)
        return x
input_dim = x.shape[1]
model = MLP(input_dim)
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
num_epochs = 100
for epoch in range(num_epochs):
    model.train()
    optimizer.zero_grad()
    outputs = model(x_train_tensor)
    loss = criterion(outputs, y_train_tensor)
    if torch.isnan(loss):
        print(f"NaN loss at epoch {epoch}")
        break
    loss.backward()
    optimizer.step()
    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {loss.item():.4f}')
model.eval()
with torch.no_grad():
    logits = model(x_test_tensor)
    y_pred_probs = torch.sigmoid(logits).numpy().flatten()
    y_pred = (y_pred_probs >= 0.5).astype(int)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1-score: {f1:.4f}')
```

Вывод программы:



При добавлении 32 нейронов в скрытый слой модель становится гибче и precison с recall растут приблизительно на 14%.

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