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-66
Прокурат В. Д.
Проверил:
Крощенко А. А.

Брест 2025

Цель работы: 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 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 9

Определение вида стекла
Glass Identification
Задача: классифицировать тип стекла (многоклассовая задача).
Архитектура:

- входной слой;
- два скрытых слоя по 10 нейронов в каждом (ReLU);
- выходной слой с необходимым количеством нейронов (Softmax).

Эксперимент: попробуйте обучить модель с одним скрытым слоем на 20 нейронов. Какая архитектура оказалась лучше?

Код программы:

```
import pandas as pd
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, LabelEncoder
from sklearn.metrics import accuracy_score, f1_score, classification_report


df = pd.read_csv("glass.csv")

X = df.drop("Type", axis=1).values
y = df["Type"].values

le = LabelEncoder()
y_encoded = le.fit_transform(y)

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

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

X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.long)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.long)

input_size = X_train.shape[1]
num_classes = len(le.classes_)


# 10x2
class MLP_2Hidden(nn.Module):
    def __init__(self, input_size, num_classes):
        super().__init__()
        self.layers = nn.Sequential(
            nn.Linear(input_size, 10),
            nn.ReLU(),
            nn.Linear(10, 10),
            nn.ReLU(),
```

```
            nn.Linear(10, num_classes)
        )

    def forward(self, x):
        return self.layers(x)

# 20x1
class MLP_1Hidden(nn.Module):
    def __init__(self, input_size, num_classes):
        super().__init__()
        self.layers = nn.Sequential(
            nn.Linear(input_size, 20),
            nn.ReLU(),
            nn.Linear(20, num_classes)
        )

    def forward(self, x):
        return self.layers(x)


def train_model(model, X_train, y_train, epochs=1000, lr=0.001):
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=lr)

    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) % 200 == 0:
            print(f"Epoch [{epoch+1}/{epochs}] Loss: {loss.item():.4f}")


def evaluate_model(model, X_test, y_test):
    model.eval()
    with torch.no_grad():
        outputs = model(X_test)
        _, predicted = torch.max(outputs, 1)

    acc = accuracy_score(y_test, predicted)
    f1 = f1_score(y_test, predicted, average='weighted')
    print("Accuracy:", round(acc, 3))
    print("F1-score:", round(f1, 3), "\n")

    y_true_labels = le.inverse_transform(y_test)
    y_pred_labels = le.inverse_transform(predicted)
    print(classification_report(y_true_labels, y_pred_labels, digits=3,
zero_division=0))


print("Модель с двумя скрытыми слоями по 10 нейронов:")
model_2hidden = MLP_2Hidden(input_size, num_classes)
train_model(model_2hidden, X_train, y_train)
evaluate_model(model_2hidden, X_test, y_test)

print("\nМодель с одним скрытым слоем на 20 нейронов:")
model_1hidden = MLP_1Hidden(input_size, num_classes)
```

```
train_model(model_1hidden, X_train, y_train)
evaluate_model(model_1hidden, X_test, y_test)
```

Результат работы программы:

```
Модель с двумя скрытыми слоями по 10 нейронов:
Epoch [200/1000] Loss: 0.9521
Epoch [400/1000] Loss: 0.6674
Epoch [600/1000] Loss: 0.4999
Epoch [800/1000] Loss: 0.3561
Epoch [1000/1000] Loss: 0.2706
Accuracy: 0.674
F1-score: 0.677

              precision    recall  f1-score   support

           1      0.750     0.643     0.692        14
           2      0.611     0.733     0.667        15
           3      0.333     0.333     0.333         3
           5      1.000     0.667     0.800         3
           6      0.500     0.500     0.500         2
           7      0.833     0.833     0.833         6

    accuracy                          0.674        43
   macro avg      0.671     0.618     0.638        43
weighted avg      0.690     0.674     0.677        43
```

```
Модель с одним скрытым слоем на 20 нейронов:
Epoch [200/1000] Loss: 0.9734
Epoch [400/1000] Loss: 0.6733
Epoch [600/1000] Loss: 0.5425
Epoch [800/1000] Loss: 0.4386
Epoch [1000/1000] Loss: 0.3574
Accuracy: 0.698
F1-score: 0.701

              precision    recall  f1-score   support

           1      0.692     0.643     0.667        14
           2      0.588     0.667     0.625        15
           3      0.667     0.667     0.667         3
           5      1.000     0.667     0.800         3
           6      0.667     1.000     0.800         2
           7      1.000     0.833     0.909         6

    accuracy                          0.698        43
   macro avg      0.769     0.746     0.745        43
weighted avg      0.717     0.698     0.701        43
```

Модель с одним скрытым слоем на 20 нейронов показала немного лучшие результаты: accuracy = 0.698 против 0.674, F1-score = 0.701 против 0.677.

Следовательно, архитектура с одним скрытым слоем (20 нейронов) оказалась чуть более эффективной для данного набора данных.

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