

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 курса
Группы АС-66
Лысюк Р. А.
Проверил:
Крощенко А. А.

Брест 2025

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

Ход работы

Задачи:

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()`;

Вариант 4 Распознавание рукописных цифр

- Digits

- Задача: распознать цифру от 0 до 9 (10 классов).

- Архитектура:

- о входной слой;

- о один скрытый слой с 32 нейронами (ReLU);

- о выходной слой с 10 нейронами (Softmax).

- Эксперимент: сравните результаты с архитектурой, где два скрытых слоя по 32 нейрона. Улучшилась ли точность?

Код:

```
import torch

import torch.nn as nn

import torch.optim as optim

from sklearn.datasets import load_digits

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy_score, classification_report

import numpy as np
```

```
digits = load_digits()
```

```
X, y = digits.data, digits.target
```

```
scaler = StandardScaler()
```

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

```
X_train, X_test, y_train, y_test = train_test_split(
```

```
    X_scaled, y, test_size=0.3, random_state=42)
```

```
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, dtype=torch.long)
```

```
y_test_tensor = torch.tensor(y_test, dtype=torch.long)
```

```
class MLP1(nn.Module):  
    def __init__(self):  
        super().__init__()  
        self.model = nn.Sequential(  
            nn.Linear(64, 32),  
            nn.ReLU(),  
            nn.Linear(32, 10)  
        )
```

```
    def forward(self, x):  
        return self.model(x)
```

```
class MLP2(nn.Module):  
    def __init__(self):  
        super().__init__()  
        self.model = nn.Sequential(  
            nn.Linear(64, 32),  
            nn.ReLU(),  
            nn.Linear(32, 32),  
            nn.ReLU(),  
            nn.Linear(32, 10)  
        )
```

```
    def forward(self, x):  
        return self.model(x)
```

```
def train_model(model, X_train, y_train, epochs=50):  
    criterion = nn.CrossEntropyLoss()  
    optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
    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()
```

```

def evaluate_model(model, X_test, y_test):
    model.eval()
    with torch.no_grad():
        logits = model(X_test)
        preds = torch.argmax(logits, dim=1)
        acc = accuracy_score(y_test.numpy(), preds.numpy())
        report = classification_report(y_test.numpy(), preds.numpy())
        return acc, report

model1 = MLP1()
train_model(model1, X_train_tensor, y_train_tensor)
acc1, report1 = evaluate_model(model1, X_test_tensor, y_test_tensor)

model2 = MLP2()
train_model(model2, X_train_tensor, y_train_tensor)
acc2, report2 = evaluate_model(model2, X_test_tensor, y_test_tensor)

print("=== MLP с 1 скрытым слоем ===")
print(f"Точность: {acc1:.4f}")
print(report1)

print("\n=== MLP с 2 скрытыми слоями ===")
print(f"Точность: {acc2:.4f}")
print(report2)

```

Вывод консоли:

```
=== MLP с 1 скрытым слоем ===
```

```
Точность: 0.7907
```

	precision	recall	f1-score	support
0	0.94	0.89	0.91	53
1	0.81	0.60	0.69	50
2	0.69	0.85	0.76	47
3	0.57	0.93	0.71	54
4	0.82	0.98	0.89	60
5	0.93	0.85	0.89	66
6	0.88	0.98	0.93	53
7	0.85	0.95	0.90	55
8	0.68	0.44	0.54	43
9	0.79	0.37	0.51	59
accuracy			0.79	540
macro avg	0.80	0.78	0.77	540
weighted avg	0.80	0.79	0.78	540

```
=== MLP с 2 скрытыми слоями ===
```

```
Точность: 0.6370
```

	precision	recall	f1-score	support
0	0.61	0.96	0.75	53
1	0.87	0.26	0.40	50
2	0.45	0.94	0.61	47
3	0.53	0.93	0.67	54
4	0.67	1.00	0.81	60
5	1.00	0.09	0.17	66
6	0.98	0.77	0.86	53
7	0.64	0.89	0.75	55
8	0.94	0.37	0.53	43
9	0.70	0.24	0.35	59
accuracy			0.64	540
macro avg	0.74	0.64	0.59	540
weighted avg	0.74	0.64	0.58	540

Вывод: построил, обучил и оценил многослойный перцептрон (MLP) для решения задачи классификации.