

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»

Выполнил:
Студент 3 курса
Группы АС-65
Дацкевич М.Г.
Проверил:
Крощенко А.А.

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

Вариант 4

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 (ассураку, f1-score и т.д.), используя `sklearn.metrics`.

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?

```
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, f1_score import numpy as np

# 1. Импорт библиотек и подготовка данных
data = load_digits()
X, y = data.data, data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

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

# 2. Определение архитектуры нейронной сети class MLP(nn.Module):
def __init__(self, input_size=64, hidden_size=32, num_layers=1):
    super(MLP, self).__init__()
    self.num_layers = num_layers
    self.fc1 = nn.Linear(input_size,
hidden_size)
    self.relu = nn.ReLU()
    if
num_layers == 2:
        self.fc2 = nn.Linear(hidden_size, hidden_size)
    self.fc3 = nn.Linear(hidden_size, 10)
    self.fc2 = nn.Linear(hidden_size, 10)
    self.softmax =
nn.Softmax(dim=1)
    def forward(self,
x):
        x = self.relu(self.fc1(x))
        if
self.num_layers == 2:
            x = self.relu(self.fc2(x))
        x = self.fc3(x)
        else:
            x = self.fc2(x)
```

```

        x =
self.softmax(x)
return x

# 3. Инициализация модели, функции потерь и оптимизатора
model_one_layer = MLP(hidden_size=32, num_layers=1) criterion =
nn.CrossEntropyLoss() optimizer_one =
optim.Adam(model_one_layer.parameters(), lr=0.001)

# 4. Написание цикла обучения (Training Loop)
epochs = 1000 for epoch in range(epochs):
model_one_layer.train()
optimizer_one.zero_grad()      outputs =
model_one_layer(X_train)      loss =
criterion(outputs, y_train)
loss.backward()      optimizer_one.step()

# 5. Оценка модели (Evaluation) def
evaluate_model(model, X_test, y_test):
    model.eval()      with
torch.no_grad():
        outputs = model(X_test)
        _, predicted = torch.max(outputs.data, 1)
    accuracy = accuracy_score(y_test, predicted)      f1 =
f1_score(y_test, predicted, average='weighted')      return
accuracy, f1
    acc_one, f1_one = evaluate_model(model_one_layer, X_test,
y_test)
    print("Модель с одним скрытым слоем:") print(f"Accuracy:
{acc_one:.4f}, F1-score: {f1_one:.4f}")

# Эксперимент: модель с двумя скрытыми слоями model_two_layers
= MLP(hidden_size=32, num_layers=2)
optimizer_two = optim.Adam(model_two_layers.parameters(), lr=0.001)
for epoch in range(epochs):
model_two_layers.train()
optimizer_two.zero_grad()      outputs =
model_two_layers(X_train)      loss =
criterion(outputs, y_train)
loss.backward()
optimizer_two.step()
    acc_two, f1_two = evaluate_model(model_two_layers, X_test,
y_test)
    print("\nМодель с двумя скрытыми
слоями:")
print(f"Accuracy: {acc_two:.4f}, F1-score: {f1_two:.4f}")
    print("\nСравнение точности:") print(f"Accuracy: {acc_one:.4f} ->
{acc_two:.4f} (изменение: {acc_two - acc_one:+.4f})")

```

```

Модель с одним скрытым слоем:
Ассигасу: 0.9694, F1-score: 0.9695

Модель с двумя скрытыми слоями:
Ассигасу: 0.9750, F1-score: 0.9750

Сравнение точности:
Ассигасу: 0.9694 -> 0.9750 (изменение: +0.0056)

```

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