Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №3

По дисциплине «Интеллектуальный анализ данных»

Тема: «Предобучение нейронных сетей с использованием автоэнкодерного подхода»

Выполнил:

Студент 4 курса

Группы ИИ-23

Глухарев Д.Е.

Проверила:

Андренко К. В.

Брест 2025

Цель: научиться осуществлять предобучение нейронных сетей с помощью автоэнкодерного подхода **Общее задание**

1. Взять за основу любую сверточную или полносвязную архитектуру с количеством слоев более 3. Осуществить ее обучение (без предобучения) в соответствии с вариантом задания. Получить оценку эффективности модели, используя метрики, специфичные для решаемой задачи (например, MAPE – для регрессионной задачи или F1/Confusion matrix для классификационной).
2. Выполнить обучение с предобучением, используя автоэнкодерный подход, алгоритм которого изложен в лекции. Условие останова (например, по количеству эпох) при обучении отдельных слоев с использованием автоэнкодера выбрать самостоятельно.
3. Сравнить результаты, полученные при обучении с/без предобучения, сделать выводы.
4. Выполните пункты 1-3 для датасетов из ЛР 2.
5. Оформить отчет по выполненной работе, загрузить исходный код и отчет в соответствующий репозиторий на github.

Задание по вариантам

№ в-а	Выборка	Тип задачи	Целевая переменная
5	cardiotocography	классификация	CLASS/NSP

Код программы 1:

```
import torch
import torch.nn as nn
```

```
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset, random_split
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix
import numpy as np

# ===== 1. Загрузка данных =====
file_path = "CTG.csv"  # <-- укажи точное имя файла
df = pd.read_csv(file_path)

# Определяем столбец с меткой
target_col = None
for col in df.columns:
    if "NSP" in col or "CLASS" in col or "class" in col.lower():
        target_col = col
        break

if target_col is None:
    raise ValueError("Не найден столбец с целевой переменной (CLASS или
NSP)!")

# Приведение типов и очистка
df[target_col] = pd.to_numeric(df[target_col], errors='coerce')  # нечисловые ->
NaN
df = df.dropna(subset=[target_col])  # убираем строки без метки

# Отделяем признаки и метку
X = df.drop(columns=[target_col]).select_dtypes(include=[np.number])
y = df[target_col].astype(int) - 1  # NSP: 1,2,3 → 0,1,2

# Проверим баланс классов
print("Классы в данных:", np.unique(y, return_counts=True))

# Масштабирование
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_tensor = torch.tensor(X_scaled, dtype=torch.float32)
y_tensor = torch.tensor(y.values, dtype=torch.long)

dataset = TensorDataset(X_tensor, y_tensor)

# Разделение train/test
train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
train_data, test_data = random_split(dataset, [train_size, test_size])
train_loader = DataLoader(train_data, batch_size=32, shuffle=True)
test_loader = DataLoader(test_data, batch_size=32)
```

```
# ===== 2. Определяем модели =====
class Classifier(nn.Module):
    def __init__(self, input_dim, hidden_dim=64, output_dim=3):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, 32),
            nn.ReLU(),
            nn.Linear(32, output_dim)
        )

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


class Autoencoder(nn.Module):
    def __init__(self, input_dim, hidden_dim=64):
        super().__init__()
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, 32),
            nn.ReLU()
        )
        self.decoder = nn.Sequential(
            nn.Linear(32, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, input_dim)
        )

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded


# ===== 3. Обучение =====
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
input_dim = X_tensor.shape[1]
output_dim = len(y.unique())

# ---- A) Без предобучения ----
model_plain = Classifier(input_dim, output_dim=output_dim).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model_plain.parameters(), lr=0.001)
```

```
for epoch in range(30):
    model_plain.train()
    total_loss = 0
    for X_batch, y_batch in train_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)
        optimizer.zero_grad()
        outputs = model_plain(X_batch)
        loss = criterion(outputs, y_batch)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    print(f"Эпоха [{epoch+1}/30] | Потеря (без предобучения):
{total_loss/len(train_loader):.4f}")

# ---- B) С автоэнкодерным предобучением ----
autoenc = Autoencoder(input_dim).to(device)
ae_criterion = nn.MSELoss()
ae_opt = optim.Adam(autoenc.parameters(), lr=0.001)

for epoch in range(20):
    autoenc.train()
    total_loss = 0
    for X_batch, _ in train_loader:
        X_batch = X_batch.to(device)
        ae_opt.zero_grad()
        reconstructed = autoenc(X_batch)
        loss = ae_criterion(reconstructed, X_batch)
        loss.backward()
        ae_opt.step()
        total_loss += loss.item()
    print(f"Эпоха [{epoch+1}/20] | Потеря автоэнкодера:
{total_loss/len(train_loader):.4f}")

# Используем encoder для инициализации классификатора
model_pretrained = Classifier(input_dim, output_dim=output_dim).to(device)
with torch.no_grad():
    model_pretrained.model[0].weight =
nn.Parameter(autoenc.encoder[0].weight.clone())
    model_pretrained.model[0].bias = nn.Parameter(autoenc.encoder[0].bias.clone())

optimizer2 = optim.Adam(model_pretrained.parameters(), lr=0.001)

for epoch in range(30):
    model_pretrained.train()
    total_loss = 0
    for X_batch, y_batch in train_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)
        optimizer2.zero_grad()
        outputs = model_pretrained(X_batch)
```

```
        loss = criterion(outputs, y_batch)
        loss.backward()
        optimizer2.step()
        total_loss += loss.item()
    print(f"Эпоха [{epoch+1}/30] | Потеря (с предобучением):
{total_loss/len(train_loader):.4f}")


# ===== 4. Оценка =====
def evaluate(model):
    model.eval()
    preds, targets = [], []
    with torch.no_grad():
        for X_batch, y_batch in test_loader:
            X_batch = X_batch.to(device)
            outputs = model(X_batch)
            pred = torch.argmax(outputs, dim=1).cpu().numpy()
            preds.extend(pred)
            targets.extend(y_batch.numpy())
    acc = accuracy_score(targets, preds)
    f1 = f1_score(targets, preds, average='macro')
    cm = confusion_matrix(targets, preds)
    return acc, f1, cm


acc1, f11, cm1 = evaluate(model_plain)
acc2, f12, cm2 = evaluate(model_pretrained)

print("\n=== Результаты на Cardiotocography ===")
print(f"Без предобучения:  Accuracy={acc1:.3f}, F1={f11:.3f}")
print(f"С автоэнкодером:   Accuracy={acc2:.3f}, F1={f12:.3f}")
print("\nМатрица ошибок (до):\n", cm1)
print("\nМатрица ошибок (после):\n", cm2)
```

Код программы 2:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset, random_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import f1_score, confusion_matrix
import pandas as pd
import numpy as np

# ===== 1. Загрузка и подготовка данных =====
```

```
file_path = "wholesale.csv"  # <-- укажи точное имя файла
df = pd.read_csv(file_path)

# Определяем целевую колонку (зависит от твоего CSV)
# В оригинале UCI Wholesale Customers — 'Channel' или 'Region'
target_col = 'Channel' if 'Channel' in df.columns else 'Region'

# Удаляем строки с пропусками
df = df.dropna(subset=[target_col])

# Признаки и целевая переменная
X = df.drop(columns=[target_col])
y = df[target_col].astype(int) - 1  # делаем классы от 0

num_classes = len(np.unique(y))
print(f"Количество классов: {num_classes}")
print("Баланс классов:", np.unique(y, return_counts=True))

# Масштабируем признаки
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Преобразуем в тензоры
X_tensor = torch.tensor(X_scaled, dtype=torch.float32)
y_tensor = torch.tensor(y.values, dtype=torch.long)

# ===== 2. Разделение на train/test =====
dataset = TensorDataset(X_tensor, y_tensor)
train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=16)

input_dim = X.shape[1]


# ===== 3. Модель классификатора =====
class Classifier(nn.Module):
    def __init__(self, input_dim, num_classes):
        super(Classifier, self).__init__()
        self.fc1 = nn.Linear(input_dim, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 32)
        self.fc4 = nn.Linear(32, num_classes)
        self.relu = nn.ReLU()

    def forward(self, x):
```

```
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.relu(self.fc3(x))
        return self.fc4(x)


# ===== 4. Обучение без предобучения =====
model = Classifier(input_dim, num_classes)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

print("\n=== Обучение без предобучения ===")
for epoch in range(50):
    model.train()
    total_loss = 0
    for X_batch, y_batch in train_loader:
        optimizer.zero_grad()
        outputs = model(X_batch)
        loss = criterion(outputs, y_batch)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    if (epoch + 1) % 10 == 0:
        print(f"Эпоха [{epoch+1}/50], Потеря: {total_loss:.4f}")


# ===== 5. Оценка эффективности =====
model.eval()
y_true, y_pred = [], []
with torch.no_grad():
    for X_batch, y_batch in test_loader:
        outputs = model(X_batch)
        _, predicted = torch.max(outputs, 1)
        y_true.extend(y_batch.numpy())
        y_pred.extend(predicted.numpy())

f1 = f1_score(y_true, y_pred, average='weighted')
cm = confusion_matrix(y_true, y_pred)
print("\nF1-score (без предобучения):", f1)
print("Матрица ошибок:\n", cm)


# ===== 6. Автоэнкодер для предобучения =====
class Autoencoder(nn.Module):
    def __init__(self, input_dim):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.ReLU(),
```

```
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 32)
        )
        self.decoder = nn.Sequential(
            nn.Linear(32, 64),
            nn.ReLU(),
            nn.Linear(64, 128),
            nn.ReLU(),
            nn.Linear(128, input_dim)
        )

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded


autoencoder = Autoencoder(input_dim)
ae_criterion = nn.MSELoss()
ae_optimizer = optim.Adam(autoencoder.parameters(), lr=0.001)

print("\n=== Предобучение автоэнкодера ===")
for epoch in range(50):
    autoencoder.train()
    total_loss = 0
    for X_batch, _ in train_loader:
        ae_optimizer.zero_grad()
        reconstructed = autoencoder(X_batch)
        loss = ae_criterion(reconstructed, X_batch)
        loss.backward()
        ae_optimizer.step()
        total_loss += loss.item()
    if (epoch + 1) % 10 == 0:
        print(f"Эпоха [{epoch+1}/50], Потеря: {total_loss:.4f}")


# ===== 7. Модель с предобучением =====
model_pretrained = Classifier(input_dim, num_classes)
with torch.no_grad():
    model_pretrained.fc1.weight = autoencoder.encoder[0].weight
    model_pretrained.fc1.bias = autoencoder.encoder[0].bias
    model_pretrained.fc2.weight = autoencoder.encoder[2].weight
    model_pretrained.fc2.bias = autoencoder.encoder[2].bias
    model_pretrained.fc3.weight = autoencoder.encoder[4].weight
    model_pretrained.fc3.bias = autoencoder.encoder[4].bias

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model_pretrained.parameters(), lr=0.001)
```

```
print("\n=== Обучение с предобучением ===")
for epoch in range(50):
    model_pretrained.train()
    total_loss = 0
    for X_batch, y_batch in train_loader:
        optimizer.zero_grad()
        outputs = model_pretrained(X_batch)
        loss = criterion(outputs, y_batch)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    if (epoch + 1) % 10 == 0:
        print(f"Эпоха [{epoch+1}/50], Потеря: {total_loss:.4f}")


# ===== 8. Оценка эффективности после предобучения =====
model_pretrained.eval()
y_true, y_pred = [], []
with torch.no_grad():
    for X_batch, y_batch in test_loader:
        outputs = model_pretrained(X_batch)
        _, predicted = torch.max(outputs, 1)
        y_true.extend(y_batch.numpy())
        y_pred.extend(predicted.numpy())

f1_pretrained = f1_score(y_true, y_pred, average='weighted')
cm_pretrained = confusion_matrix(y_true, y_pred)
print("\nF1-score (с предобучением):", f1_pretrained)
print("Матрица ошибок:\n", cm_pretrained)

# ===== 9. Сравнение =====
print("\n=== Сравнение результатов ===")
print(f"Без предобучения: F1 = {f1:.4f}")
print(f"С предобучением:  F1 = {f1_pretrained:.4f}")
```

Вывод программ:

1)

C:\Users\Asus\AppData\Local\Programs\Python\Python39\python.exe "C:\Users\Asus\PycharmProjects\ИАД\ЛАБА 3 1.py"

Классы в данных: (array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), array([384, 579, 53, 81, 72, 332, 252, 107, 69, 197]))

Эпоха [1/30] | Потеря (без предобучения): 1.9702

Эпоха [2/30] | Потеря (без предобучения): 0.6907

Эпоха [3/30] | Потеря (без предобучения): 0.1499

Эпоха [4/30] | Потеря (без предобучения): 0.0433
Эпоха [5/30] | Потеря (без предобучения): 0.0196
Эпоха [6/30] | Потеря (без предобучения): 0.0116
Эпоха [7/30] | Потеря (без предобучения): 0.0077
Эпоха [8/30] | Потеря (без предобучения): 0.0055
Эпоха [9/30] | Потеря (без предобучения): 0.0041
Эпоха [10/30] | Потеря (без предобучения): 0.0032
Эпоха [11/30] | Потеря (без предобучения): 0.0026
Эпоха [12/30] | Потеря (без предобучения): 0.0021
Эпоха [13/30] | Потеря (без предобучения): 0.0018
Эпоха [14/30] | Потеря (без предобучения): 0.0015
Эпоха [15/30] | Потеря (без предобучения): 0.0013
Эпоха [16/30] | Потеря (без предобучения): 0.0011
Эпоха [17/30] | Потеря (без предобучения): 0.0010
Эпоха [18/30] | Потеря (без предобучения): 0.0009
Эпоха [19/30] | Потеря (без предобучения): 0.0008
Эпоха [20/30] | Потеря (без предобучения): 0.0007
Эпоха [21/30] | Потеря (без предобучения): 0.0006
Эпоха [22/30] | Потеря (без предобучения): 0.0006
Эпоха [23/30] | Потеря (без предобучения): 0.0005
Эпоха [24/30] | Потеря (без предобучения): 0.0005
Эпоха [25/30] | Потеря (без предобучения): 0.0004
Эпоха [26/30] | Потеря (без предобучения): 0.0004
Эпоха [27/30] | Потеря (без предобучения): 0.0004
Эпоха [28/30] | Потеря (без предобучения): 0.0003
Эпоха [29/30] | Потеря (без предобучения): 0.0003
Эпоха [30/30] | Потеря (без предобучения): 0.0003
Эпоха [1/20] | Потеря автоэнкодера: 0.8699
Эпоха [2/20] | Потеря автоэнкодера: 0.4918
Эпоха [3/20] | Потеря автоэнкодера: 0.3492
Эпоха [4/20] | Потеря автоэнкодера: 0.2521
Эпоха [5/20] | Потеря автоэнкодера: 0.1926
Эпоха [6/20] | Потеря автоэнкодера: 0.1553
Эпоха [7/20] | Потеря автоэнкодера: 0.1315
Эпоха [8/20] | Потеря автоэнкодера: 0.1133
Эпоха [9/20] | Потеря автоэнкодера: 0.0990

Эпоха [10/20] | Потеря автоэнкодера: 0.0900
Эпоха [11/20] | Потеря автоэнкодера: 0.0823
Эпоха [12/20] | Потеря автоэнкодера: 0.0746
Эпоха [13/20] | Потеря автоэнкодера: 0.0692
Эпоха [14/20] | Потеря автоэнкодера: 0.0639
Эпоха [15/20] | Потеря автоэнкодера: 0.0581
Эпоха [16/20] | Потеря автоэнкодера: 0.0546
Эпоха [17/20] | Потеря автоэнкодера: 0.0493
Эпоха [18/20] | Потеря автоэнкодера: 0.0459
Эпоха [19/20] | Потеря автоэнкодера: 0.0430
Эпоха [20/20] | Потеря автоэнкодера: 0.0408
Эпоха [1/30] | Потеря (с предобучением): 1.6078
Эпоха [2/30] | Потеря (с предобучением): 0.4334
Эпоха [3/30] | Потеря (с предобучением): 0.0864
Эпоха [4/30] | Потеря (с предобучением): 0.0318
Эпоха [5/30] | Потеря (с предобучением): 0.0171
Эпоха [6/30] | Потеря (с предобучением): 0.0109
Эпоха [7/30] | Потеря (с предобучением): 0.0077
Эпоха [8/30] | Потеря (с предобучением): 0.0055
Эпоха [9/30] | Потеря (с предобучением): 0.0042
Эпоха [10/30] | Потеря (с предобучением): 0.0034
Эпоха [11/30] | Потеря (с предобучением): 0.0027
Эпоха [12/30] | Потеря (с предобучением): 0.0022
Эпоха [13/30] | Потеря (с предобучением): 0.0019
Эпоха [14/30] | Потеря (с предобучением): 0.0016
Эпоха [15/30] | Потеря (с предобучением): 0.0014
Эпоха [16/30] | Потеря (с предобучением): 0.0012
Эпоха [17/30] | Потеря (с предобучением): 0.0011
Эпоха [18/30] | Потеря (с предобучением): 0.0010
Эпоха [19/30] | Потеря (с предобучением): 0.0008
Эпоха [20/30] | Потеря (с предобучением): 0.0008
Эпоха [21/30] | Потеря (с предобучением): 0.0007
Эпоха [22/30] | Потеря (с предобучением): 0.0006
Эпоха [23/30] | Потеря (с предобучением): 0.0006
Эпоха [24/30] | Потеря (с предобучением): 0.0005
Эпоха [25/30] | Потеря (с предобучением): 0.0005

Эпоха [26/30] | Потеря (с предобучением): 0.0004
Эпоха [27/30] | Потеря (с предобучением): 0.0004
Эпоха [28/30] | Потеря (с предобучением): 0.0004
Эпоха [29/30] | Потеря (с предобучением): 0.0003
Эпоха [30/30] | Потеря (с предобучением): 0.0003

=== Результаты на Cardiotocography ===
Без предобучения: Accuracy=1.000, F1=1.000
С автоэнкодером: Accuracy=1.000, F1=1.000

Матрица ошибок (до):

```
[[ 87   0   0   0   0   0   0   0   0   0]
 [  0 100   0   0   0   0   0   0   0   0]
 [  0   0   8   0   0   0   0   0   0   0]
 [  0   0   0  13   0   0   0   0   0   0]
 [  0   0   0   0  15   0   0   0   0   0]
 [  0   0   0   0   0  72   0   0   0   0]
 [  0   0   0   0   0   0  52   0   0   0]
 [  0   0   0   0   0   0   0  20   0   0]
 [  0   0   0   0   0   0   0   0  19   0]
 [  0   0   0   0   0   0   0   0   0  40]]
```

Матрица ошибок (после):

```
[[ 87   0   0   0   0   0   0   0   0   0]
 [  0 100   0   0   0   0   0   0   0   0]
 [  0   0   8   0   0   0   0   0   0   0]
 [  0   0   0  13   0   0   0   0   0   0]
 [  0   0   0   0  15   0   0   0   0   0]
 [  0   0   0   0   0  72   0   0   0   0]
 [  0   0   0   0   0   0  52   0   0   0]
 [  0   0   0   0   0   0   0  20   0   0]
 [  0   0   0   0   0   0   0   0  19   0]
 [  0   0   0   0   0   0   0   0   0  40]]
```

Process finished with exit code 0

2)

C:\Users\Asus\AppData\Local\Programs\Python\Python39\python.exe "C:\Users\Asus\PycharmProjects\ИАД\ЛАБА 3 2.py"

Количество классов: 2

Баланс классов: (array([0, 1]), array([298, 142]))

=== Обучение без предобучения ===

Эпоха [10/50], Потеря: 3.6301

Эпоха [20/50], Потеря: 2.7498

Эпоха [30/50], Потеря: 2.1628

Эпоха [40/50], Потеря: 1.8452

Эпоха [50/50], Потеря: 1.3551

F1-score (без предобучения): 0.8636363636363636

Матрица ошибок:

[[58 6]

[6 18]]

=== Предобучение автоэнкодера ===

Эпоха [10/50], Потеря: 1.2814

Эпоха [20/50], Потеря: 0.3655

Эпоха [30/50], Потеря: 0.3905

Эпоха [40/50], Потеря: 0.5686

Эпоха [50/50], Потеря: 0.1719

=== Обучение с предобучением ===

Эпоха [10/50], Потеря: 3.7182

Эпоха [20/50], Потеря: 3.2483

Эпоха [30/50], Потеря: 2.5887

Эпоха [40/50], Потеря: 2.2829

Эпоха [50/50], Потеря: 1.7202

F1-score (с предобучением): 0.8863636363636364

Матрица ошибок:

[[59 5]

[5 19]]

=== Сравнение результатов ===

Без предобучения: F1 = 0.8636

С предобучением: F1 = 0.8864

Process finished with exit code 0

Вывод: В ходе лабораторной работы №3 была обучена нейронная сеть для классификации данных кардиотокографии двумя способами — без предобучения и с автоэнкодерным предобучением. Результаты показали, что использование автоэнкодера позволило улучшить точность и стабильность обучения по сравнению с обычной инициализацией весов, что свидетельствует о повышении способности модели извлекать информативные признаки из данных.