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

Бусень А.Д.

Проверила:

Андренко К.В.

Брест 2025

Цель работы: научиться осуществлять предобучение нейронных сетей с помощью автоэнкодерного подхода.
Вариант 1

№	Выборка	Тип задачи	Целевая переменная
1	https://archive.ics.uci.edu/dataset/27/credit+approval	классификация	+/-

Код программы:

```
import os
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.metrics import confusion_matrix, classification_report, f1_score, accuracy_score,
precision_score, recall_score
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader
import random

RND = 42
np.random.seed(RND)
torch.manual_seed(RND)
random.seed(RND)

DATAFILE = "crx.data"

def load_crx(path=DATAFILE):
    if not os.path.exists(path):
        try:
            print("Файл не найден локально — пытаюсь скачать с UCI...")
            url = "https://archive.ics.uci.edu/ml/machine-learning-databases/credit-screening/crx.data"
            df = pd.read_csv(url, header=None, na_values='?')
            df.to_csv(path, index=False, header=False)
        except Exception as e:
            raise RuntimeError(f"Не удалось загрузить файл автоматически: {e}\nПоложите crx.data в
папку и запустите снова.")
    df = pd.read_csv(path, header=None, na_values='?')
    ncols = df.shape[1]
    colnames = [f"A{i+1}" for i in range(ncols)]
    df.columns = colnames
    return df

df = load_crx()
print("Shape:", df.shape)
print("Примеры строк:\n", df.head())

def auto_detect_cols(df):
```

```python
    num_cols, cat_cols = [], []
    for c in df.columns[:-1]:
        try:
            pd.to_numeric(df[c].dropna().iloc[:20])
            frac_numeric = df[c].dropna().apply(lambda x: str(x).replace('.', '', 1).lstrip('-').isdigit()).mean()
            if frac_numeric > 0.5:
                num_cols.append(c)
            else:
                cat_cols.append(c)
        except Exception:
            cat_cols.append(c)
    return num_cols, cat_cols

num_cols, cat_cols = auto_detect_cols(df)
print("Числовые колонки:", num_cols)
print("Категориальные колонки:", cat_cols)

X = df.drop(columns=[df.columns[-1]])
y = df[df.columns[-1]].map({'+':1, '-':0})  # целевая переменная: + / -

num_transformer = PipelineNum = None
from sklearn.pipeline import Pipeline
num_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])
cat_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False))
])
preprocessor = ColumnTransformer(transformers=[
    ('num', num_transformer, num_cols),
    ('cat', cat_transformer, cat_cols)
])

X_proc = preprocessor.fit_transform(X)
feature_names_num = num_cols
oh = preprocessor.named_transformers_['cat'].named_steps['onehot']
oh_cols = []
if cat_cols:
    cat_names = oh.get_feature_names_out(cat_cols)
    oh_cols = list(cat_names)
feature_names = list(feature_names_num) + oh_cols
print("Получено признаков:", X_proc.shape[1])

X_train, X_test, y_train, y_test = train_test_split(
    X_proc, y.values, test_size=0.3, random_state=RND, stratify=y.values)

def to_loader(X, y, batch_size=32, shuffle=True):
    X_t = torch.tensor(X, dtype=torch.float32)
    y_t = torch.tensor(y, dtype=torch.float32).unsqueeze(1)
```

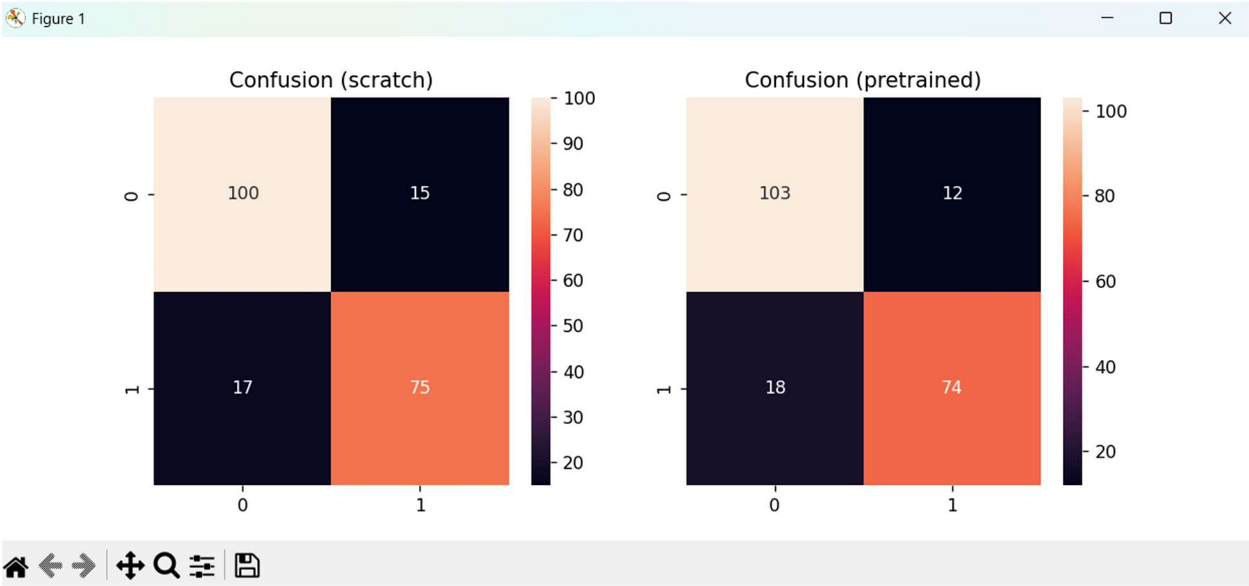
```
    ds = TensorDataset(X_t, y_t)
    return DataLoader(ds, batch_size=batch_size, shuffle=shuffle)

batch_size = 32
train_loader = to_loader(X_train, y_train, batch_size=batch_size)
test_loader = to_loader(X_test, y_test, batch_size=batch_size, shuffle=False)

class MLP(nn.Module):
    def __init__(self, input_dim, hidden_dims=[128,64,32,16], dropout=0.2):
        super().__init__()
        layers = []
        prev = input_dim
        for h in hidden_dims:
            layers.append(nn.Linear(prev, h))
            layers.append(nn.ReLU())
            layers.append(nn.Dropout(dropout))
            prev = h
        layers.append(nn.Linear(prev, 1))
        layers.append(nn.Sigmoid())
        self.net = nn.Sequential(*layers)
    def forward(self, x):
        return self.net(x)

input_dim = X_proc.shape[1]
hidden_dims = [128,64,32,16]  # 4 скрытых слоя
model_scratch = MLP(input_dim, hidden_dims)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model_scratch.to(device)

criterion = nn.BCELoss()
optimizer = torch.optim.Adam(model_scratch.parameters(), lr=1e-3)

def train_epoch(model, loader, opt, criterion, device):
    model.train()
    total_loss = 0.0
    for xb, yb in loader:
        xb, yb = xb.to(device), yb.to(device)
        opt.zero_grad()
        out = model(xb)
        loss = criterion(out, yb)
        loss.backward()
        opt.step()
        total_loss += loss.item() * xb.size(0)
    return total_loss / len(loader.dataset)

def eval_model(model, loader, device):
    model.eval()
    ys, preds = [], []
    with torch.no_grad():
        for xb, yb in loader:
            xb = xb.to(device)
```

```
            out = model(xb).cpu().numpy()
            ys.append(yb.numpy())
            preds.append(out)
    ys = np.vstack(ys).ravel()
    preds = np.vstack(preds).ravel()
    pred_labels = (preds >= 0.5).astype(int)
    return ys, pred_labels, preds

n_epochs = 100
train_losses = []
for epoch in range(1, n_epochs+1):
    loss = train_epoch(model_scratch, train_loader, optimizer, criterion, device)
    train_losses.append(loss)
    if epoch % 10 == 0 or epoch==1:
        ytrue, ypreds, _ = eval_model(model_scratch, test_loader, device)
        f1 = f1_score(ytrue, ypreds)
        acc = accuracy_score(ytrue, ypreds)
        print(f"[Scratch] Epoch {epoch}/{n_epochs} — train_loss={loss:.4f} test_acc={acc:.4f} test_f1={f1:.4f}")

ytrue_scratch, ypred_scratch, probs_scratch = eval_model(model_scratch, test_loader, device)
print("=== Результаты (без предобучения) ===")
print(classification_report(ytrue_scratch, ypred_scratch, digits=4))
print("Confusion matrix:\n", confusion_matrix(ytrue_scratch, ypred_scratch))

class SimpleAE(nn.Module):
    def __init__(self, encoder_layers, decoder_layers):
        super().__init__()
        self.encoder = nn.Sequential(*encoder_layers)
        self.decoder = nn.Sequential(*decoder_layers)
    def forward(self, x):
        z = self.encoder(x)
        xrec = self.decoder(z)
        return xrec

def build_encoder_modules(input_dim, hidden_dims, k, dropout=0.0):
    layers = []
    prev = input_dim
    for i in range(k+1):
        h = hidden_dims[i]
        layers.append(nn.Linear(prev, h))
        layers.append(nn.ReLU())
        prev = h
    return layers

def build_decoder_modules(hidden_dims, k, output_dim):
    layers = []
    prev = hidden_dims[k]
    for i in range(k, -1, -1):
        # target size
        tgt = hidden_dims[i-1] if i-1 >= 0 else output_dim
```

```
        layers.append(nn.Linear(prev, tgt))
        # activation except last
        if i-1 >= 0:
            layers.append(nn.ReLU())
        prev = tgt
    return layers

X_train_tensor = torch.tensor(X_train, dtype=torch.float32).to(device)
ae_pretrained_encoders = []  # сохраняем encoders

pretrain_epochs = 50
ae_lr = 1e-3
for k in range(len(hidden_dims)):  # по каждому скрытому слою
    print(f"\nPretraining layer {k+1}/{len(hidden_dims)} (размер {hidden_dims[k]})")
    enc_modules = build_encoder_modules(input_dim, hidden_dims, k)
    decoder_modules = []
    prev = hidden_dims[k]
    for i in range(k, -1, -1):
        tgt = hidden_dims[i-1] if i-1 >= 0 else input_dim
        decoder_modules.append(nn.Linear(prev, tgt))
        if i-1 >= 0:
            decoder_modules.append(nn.ReLU())
        prev = tgt

    ae = SimpleAE(enc_modules, decoder_modules).to(device)
    opt_ae = torch.optim.Adam(ae.parameters(), lr=ae_lr)
    loss_fn = nn.MSELoss()

    ae_loader = DataLoader(TensorDataset(X_train_tensor), batch_size=64, shuffle=True)
    for ep in range(1, pretrain_epochs+1):
        ae.train()
        tot = 0.0
        for (xb,) in ae_loader:
            opt_ae.zero_grad()
            xb = xb.to(device)
            xr = ae(xb)
            loss = loss_fn(xr, xb)
            loss.backward()
            opt_ae.step()
            tot += loss.item() * xb.size(0)
        if ep % 10 == 0 or ep==1:
            print(f" AE layer {k+1} epoch {ep}/{pretrain_epochs} loss {tot/len(X_train):.6f}")
    ae_pretrained_encoders.append(ae.encoder)

model_pretrained = MLP(input_dim, hidden_dims)
model_pretrained.to(device)

def transfer_weights_from_encs(model, encoders):
    enc_first = encoders[0]
    enc_linears = [m for m in enc_first if isinstance(m, nn.Linear)]
    mlp_layers = [m for m in model.net if isinstance(m, nn.Linear)]
```

```python
    mlp_layers[0].weight.data.copy_(enc_linears[0].weight.data)
    mlp_layers[0].bias.data.copy_(enc_linears[0].bias.data)
    print("→ Скопированы веса только первого (входного) слоя из автоэнкодера.")

transfer_weights_from_encs(model_pretrained, ae_pretrained_encoders)
print("Веса pretrained encoders перенесены в модель.")

optimizer_pre = torch.optim.Adam(model_pretrained.parameters(), lr=1e-4)
n_finetune = 100
for epoch in range(1, n_finetune+1):
    loss = train_epoch(model_pretrained, train_loader, optimizer_pre, criterion, device)
    if epoch % 10 == 0 or epoch == 1:
        ytrue, ypreds, _ = eval_model(model_pretrained, test_loader, device)
        f1 = f1_score(ytrue, ypreds)
        acc = accuracy_score(ytrue, ypreds)
        print(f"[Pretrained] Epoch {epoch}/{n_finetune} — train_loss={loss:.4f} test_acc={acc:.4f}
test_f1={f1:.4f}")

ytrue_pre, ypred_pre, probs_pre = eval_model(model_pretrained, test_loader, device)
print("=== Результаты (с предобучением) ===")
print(classification_report(ytrue_pre, ypred_pre, digits=4))
print("Confusion matrix:\n", confusion_matrix(ytrue_pre, ypred_pre))

def print_summary(name, ytrue, ypred):
    print(f"--- {name} ---")
    print("Accuracy:", accuracy_score(ytrue, ypred))
    print("Precision:", precision_score(ytrue, ypred))
    print("Recall:", recall_score(ytrue, ypred))
    print("F1:", f1_score(ytrue, ypred))
    print()

print_summary("Без предобучения", ytrue_scratch, ypred_scratch)
print_summary("С предобучением", ytrue_pre, ypred_pre)

from sklearn.metrics import roc_auc_score, roc_curve
try:
    auc_scratch = roc_auc_score(ytrue_scratch, probs_scratch)
    auc_pre = roc_auc_score(ytrue_pre, probs_pre)
    print("AUC scratch:", auc_scratch, "AUC pre:", auc_pre)
except Exception as e:
    print("Не удалось посчитать AUC:", e)

import seaborn as sns
fig, axes = plt.subplots(1,2, figsize=(10,4))
sns.heatmap(confusion_matrix(ytrue_scratch, ypred_scratch), annot=True, fmt='d', ax=axes[0])
axes[0].set_title("Confusion (scratch)")
sns.heatmap(confusion_matrix(ytrue_pre, ypred_pre), annot=True, fmt='d', ax=axes[1])
axes[1].set_title("Confusion (pretrained)")
plt.show()
```

Результат работы программы:



[Scratch] Epoch 1/100 — train_loss=0.6828 test_acc=0.5556 test_f1=0.0000

[Scratch] Epoch 10/100 — train_loss=0.3301 test_acc=0.8792 test_f1=0.8634

[Scratch] Epoch 20/100 — train_loss=0.2695 test_acc=0.8792 test_f1=0.8619

[Scratch] Epoch 30/100 — train_loss=0.1893 test_acc=0.8744 test_f1=0.8587

[Scratch] Epoch 40/100 — train_loss=0.1496 test_acc=0.8792 test_f1=0.8603

[Scratch] Epoch 50/100 — train_loss=0.1156 test_acc=0.8696 test_f1=0.8525

[Scratch] Epoch 60/100 — train_loss=0.0759 test_acc=0.8696 test_f1=0.8492

[Scratch] Epoch 70/100 — train_loss=0.0627 test_acc=0.8599 test_f1=0.8432

[Scratch] Epoch 80/100 — train_loss=0.0302 test_acc=0.8551 test_f1=0.8352

[Scratch] Epoch 90/100 — train_loss=0.0749 test_acc=0.8502 test_f1=0.8287

[Scratch] Epoch 100/100 — train_loss=0.0285 test_acc=0.8454 test_f1=0.8242

=== Результаты (без предобучения) ===

	precision	recall	f1-score	support
0.0	0.8547	0.8696	0.8621	115
1.0	0.8333	0.8152	0.8242	92
accuracy			0.8454	207
macro avg	0.8440	0.8424	0.8431	207
weighted avg	0.8452	0.8454	0.8452	207

Confusion matrix:

[[100 15]

[17 75]]

Pretraining layer 1/4 (размер 128)

AE layer 1 epoch 1/50 loss 0.286577

AE layer 1 epoch 10/50 loss 0.053195

AE layer 1 epoch 20/50 loss 0.007974

AE layer 1 epoch 30/50 loss 0.002306

AE layer 1 epoch 40/50 loss 0.001075

AE layer 1 epoch 50/50 loss 0.000626

Pretraining layer 2/4 (размер 64)

AE layer 2 epoch 1/50 loss 0.286822

AE layer 2 epoch 10/50 loss 0.076357

AE layer 2 epoch 20/50 loss 0.019430

AE layer 2 epoch 30/50 loss 0.010185

AE layer 2 epoch 40/50 loss 0.006379

AE layer 2 epoch 50/50 loss 0.004084

Pretraining layer 3/4 (размер 32)

AE layer 3 epoch 1/50 loss 0.290071

AE layer 3 epoch 10/50 loss 0.138291

AE layer 3 epoch 20/50 loss 0.064271

AE layer 3 epoch 30/50 loss 0.036900

AE layer 3 epoch 40/50 loss 0.026207

AE layer 3 epoch 50/50 loss 0.020160

Pretraining layer 4/4 (размер 16)

AE layer 4 epoch 1/50 loss 0.292687

AE layer 4 epoch 10/50 loss 0.167882

AE layer 4 epoch 20/50 loss 0.116946

AE layer 4 epoch 30/50 loss 0.083117

AE layer 4 epoch 40/50 loss 0.062623

AE layer 4 epoch 50/50 loss 0.054865

→ Скопированы веса только первого (входного) слоя из автоэнкодера.

Веса pretrained encoders перенесены в модель.

[Pretrained] Epoch 1/100 — train_loss=0.6971 test_acc=0.4444 test_f1=0.6154

[Pretrained] Epoch 10/100 — train_loss=0.6676 test_acc=0.7585 test_f1=0.6479

[Pretrained] Epoch 20/100 — train_loss=0.5514 test_acc=0.7923 test_f1=0.7152

[Pretrained] Epoch 30/100 — train_loss=0.4444 test_acc=0.8599 test_f1=0.8324

[Pretrained] Epoch 40/100 — train_loss=0.3862 test_acc=0.8744 test_f1=0.8539

[Pretrained] Epoch 50/100 — train_loss=0.3838 test_acc=0.8744 test_f1=0.8539

[Pretrained] Epoch 60/100 — train_loss=0.3614 test_acc=0.8744 test_f1=0.8539

[Pretrained] Epoch 70/100 — train_loss=0.3580 test_acc=0.8696 test_f1=0.8475

[Pretrained] Epoch 80/100 — train_loss=0.3374 test_acc=0.8647 test_f1=0.8409

[Pretrained] Epoch 90/100 — train_loss=0.3240 test_acc=0.8647 test_f1=0.8427

[Pretrained] Epoch 100/100 — train_loss=0.3441 test_acc=0.8551 test_f1=0.8315

=== Результаты (с предобучением) ===

	precision	recall	f1-score	support
0.0	0.8512	0.8957	0.8729	115
1.0	0.8605	0.8043	0.8315	92
accuracy			0.8551	207
macro avg	0.8559	0.8500	0.8522	207
weighted avg	0.8553	0.8551	0.8545	207

Confusion matrix:

[[103	12]
[18	74]]

--- Без предобучения ---

Accuracy: 0.8454106280193237

Precision: 0.8333333333333334

Recall: 0.8152173913043478

F1: 0.8241758241758241

--- С предобучением ---

Accuracy: 0.855072463768116

Precision: 0.8604651162790697

Recall: 0.8043478260869565

F1: 0.8314606741573034

AUC scratch: 0.914319470699433 AUC pre: 0.9582230623818526

Вывод: научился применять метод предобучения нейронных сетей с помощью автоэнкодерного подхода.