

Министерство образования Республики Беларусь
Учреждение образования
«Брестский Государственный технический университет»
Кафедра ИИТ

Лабораторная работа №5
По дисциплине «Основы машинного обучения»
Тема: «Нелинейные ИНС в задачах регрессии»

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

Брест 2025

Цель: оформить моделирование прогнозирующей нелинейной ИНС, построить график, вывести результаты обучения и прогнозирования.
Вариант 4

1. Выполнить моделирование прогнозирующей нелинейной ИНС. Для генерации обучающих и тестовых данных использовать функцию

$$y = a \cos(bx) + c \sin(dx)$$

Варианты заданий приведены в следующей таблице:

№ варианта	a	b	c	d	Кол-во входов ИНС	Кол-во НЭ в скрытом слое
1	0.1	0.1	0.05	0.1	6	2
2	0.2	0.2	0.06	0.2	8	3
3	0.3	0.3	0.07	0.3	10	4
4	0.4	0.4	0.08	0.4	6	2

Для прогнозирования использовать многослойную ИНС с одним скрытым слоем. В качестве функций активации для скрытого слоя использовать сигмоидную функцию, для выходного - линейную.

```
import numpy as np
import
matplotlib.pyplot as plt

# Параметры варианта 4 a, b, c,
d = 0.4, 0.4, 0.08, 0.4

# Генерация данных x =
np.arange(0, 30.1, 0.1)
y = a * np.cos(b * x) + c * np.sin(d * x)

# Разделение на обучающую и тестовую выборки
train_size = 200 # [0, 20] с шагом 0.1 x_train,
x_test = x[:train_size], x[train_size:] y_train,
y_test = y[:train_size], y[train_size:]

# Создание dataset с окном = 6 def
create_dataset(data, window_size=6):      X,
Y = [], []      for i in range(window_size,
len(data)):
    X.append(data[i-window_size:i])
Y.append(data[i])      return
np.array(X), np.array(Y)
window_size =6
```

```

X_train, Y_train = create_dataset(y_train, window_size)
X_test, Y_test = create_dataset(y_test, window_size)

# Нормализация
mean, std = X_train.mean(), X_train.std()
X_train = (X_train - mean) / std
Y_train = (Y_train - mean) / std
X_test = (X_test - mean) / std
Y_test = (Y_test - mean) / std

# ИНС с одним скрытым слоем
class NeuralNetwork:
    def __init__(self, input_size, hidden_size,
                 output_size):
        self.W1 =
            np.random.randn(input_size, hidden_size) * 0.1
        self.b1 = np.zeros((1, hidden_size))           self.W2 =
            np.random.randn(hidden_size, output_size) * 0.1
        self.b2 = np.zeros((1, output_size))

        def
        sigmoid(self,
        x):
            return 1 / (1 + np.exp(-x))
        def
        forward(self,
        X):
            self.hidden = self.sigmoid(np.dot(X, self.W1) +
            self.b1)           self.output = np.dot(self.hidden,
            self.W2) + self.b2      return self.output
        def backward(self, X, y,
                    output, lr=0.01):
            m = X.shape[0]          dZ2 = output - y          dw2
            = (1/m) * np.dot(self.hidden.T, dZ2)          db2 = (1/m) *
            np.sum(dZ2, axis=0, keepdims=True)           dz1 =
            np.dot(dZ2, self.W2.T) * self.hidden * (1 - self.hidden)
            dw1 = (1/m) * np.dot(X.T, dz1)           db1 = (1/m) *
            np.sum(dz1, axis=0, keepdims=True)           self.W2
            -= lr * dw2          self.b2 -= lr * db2          self.W1 -=
            lr * dw1          self.b1 -= lr * db1
        def train(self, X, y,
                  epochs=1000, lr=0.01):
            errors = []          for
            epoch in range(epochs):
                output = self.forward(X)
                error = np.mean((output - y)**2)
                errors.append(error)
            self.backward(X, y, output, lr)
            return errors

```

Создание и обучение сети

```

nn = NeuralNetwork(input_size=6, hidden_size=2, output_size=1)
errors = nn.train(X_train, Y_train.reshape(-1, 1), epochs=5000,
lr=0.1)
# Прогноз train_predict =
nn.forward(X_train)
test_predict =
nn.forward(X_test)

# Обратная нормализация
train_predict = train_predict *
std + mean test_predict =
test_predict * std + mean
Y_train_orig = Y_train * std + mean
Y_test_orig = Y_test * std + mean

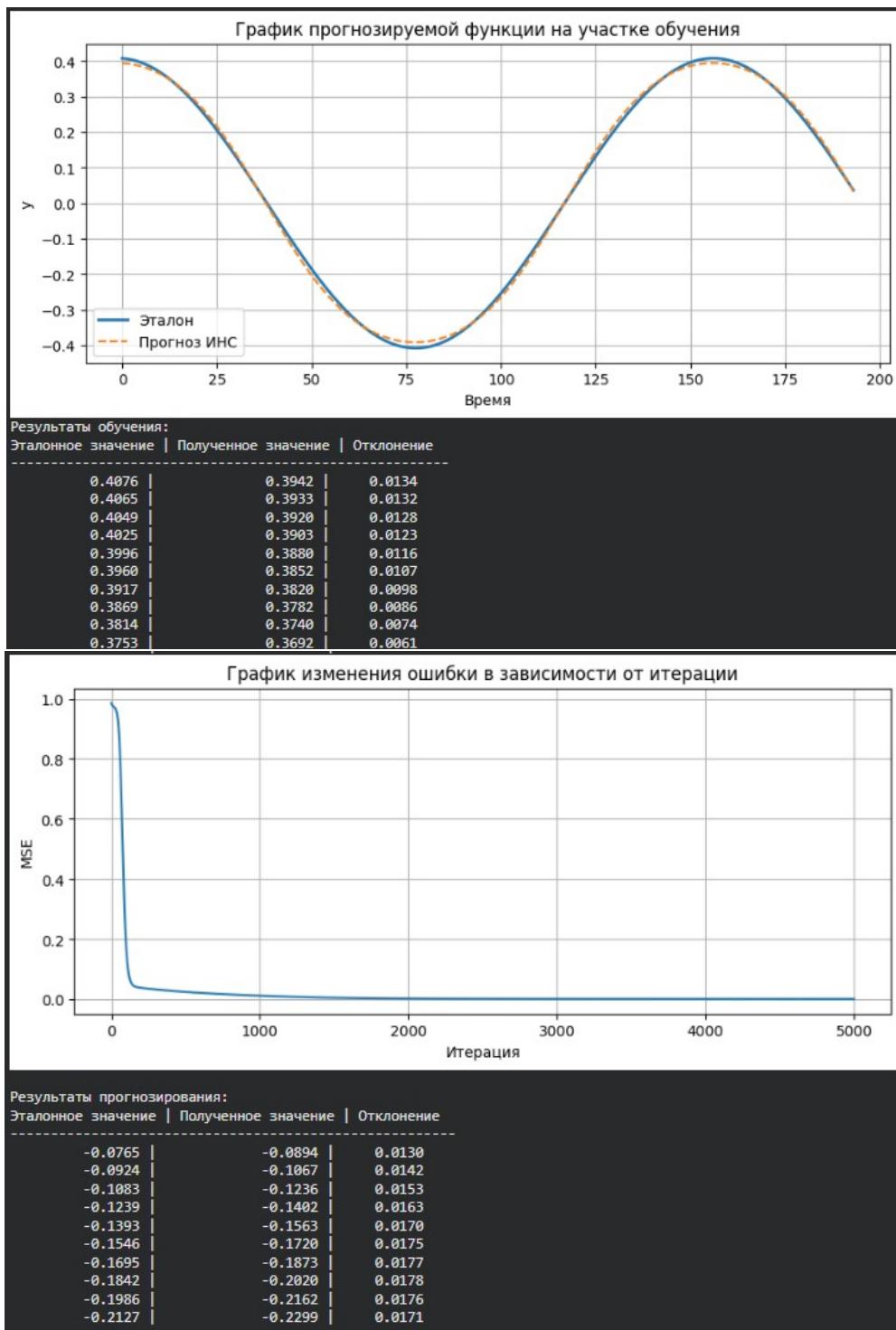
# 4. График прогнозируемой функции на участке обучения
plt.figure(figsize=(10, 4))
plt.plot(y_train>window_size:, label='Эталон',
linewidth=2) plt.plot(train_predict, label='Прогноз
ИНС', linestyle='--') plt.title('График прогнозируемой
функции на участке обучения') plt.xlabel('Время')
plt.ylabel('y') plt.legend() plt.grid(True) plt.show()

# 5. Результаты обучения print("Результаты обучения:")
print("Эталонное значение | Полученное значение |
Отклонение") print("-" * 55) for i in range(10):
    etalon = Y_train_orig[i] predicted =
train_predict[i, 0] deviation = abs(etalon - predicted)
print(f"{etalon:16.4f} | {predicted:19.4f} |
{deviation:10.4f}")
# График изменения ошибки
plt.figure(figsize=(10, 4))
plt.plot(errors)
plt.title('График изменения ошибки в зависимости от
итерации') plt.xlabel('Итерация') plt.ylabel('MSE')
plt.grid(True) plt.show()

# 6. Результаты прогнозирования print("\nРезультаты
прогнозирования:") print("Эталонное значение |
Полученное значение | Отклонение") print("-" * 55) for
i in range(10):
    etalon = Y_test_orig[i]
predicted = test_predict[i, 0]
deviation = abs(etalon -
predicted)

print(f"{etalon:16.4f} | {predicted:19.4f} | {deviation:10.4f}")

```



Вывод: оформил моделирование прогнозирующей нелинейной ИНС, построил график, вывел результаты обучения и прогнозирования.