

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

**Лабораторная работа №6**  
**По дисциплине:** «Основы машинного обучения»  
**Тема:** «Рекуррентные нейронные сети»

**Выполнила:**  
Студентка 3 курса  
Группы АС-66  
Прокурат В. Д.  
**Проверил:**  
Крощенко А. А.

**Брест 2025**

**Цель работы:** изучить применение нелинейной искусственной нейронной сети с одним скрытым слоем для решения задачи регрессии и прогнозирования, реализовать обучение сети на синтетических данных и оценить точность полученной модели.

### Вариант 9

#### Задание:

1. По вариантам предыдущей лабораторной работы реализовать предложенный вариант рекуррентной нейронной сети. Сравнить полученные результаты с ЛР 5.

Варианты заданий приведены в следующей таблице:

№	a	b	c	d	Кол-во входов ИНС	Кол-во НЭ в скрытом слое	Тип РНС
9	0.1	0.3	0.08	0.3	10	4	Мультирекуррентная

В качестве функций активации для скрытого слоя использовать сигмоидную функцию, для выходного - линейную.

Результаты для пунктов 3 и 4 приводятся для значения  $\alpha$ , при котором достигается минимальная ошибка. В выводах анализируются все полученные результаты.

#### Ход работы:

#### Код программы:

```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import pandas as pd

a, b, c, d = 0.1, 0.3, 0.08, 0.3

n_inputs = 10
n_hidden = 4
n_epochs = 2000

def func(x):
    return a * np.cos(b * x) + c * np.sin(d * x)

x = np.linspace(0, 30, 500)
y = func(x)

X, Y = [], []
```

```

for i in range(len(y) - n_inputs):
    X.append(y[i:i + n_inputs])
    Y.append(y[i + n_inputs])

X, Y = np.array(X), np.array(Y)

split = int(len(X) * 0.8)
X_train, X_test = X[:split], X[split:]
Y_train, Y_test = Y[:split], Y[split:]

torch.manual_seed(42)
X_train, Y_train = torch.tensor(X_train, dtype=torch.float32), torch.tensor(Y_train,
dtype=torch.float32).view(-1, 1)
X_test, Y_test = torch.tensor(X_test, dtype=torch.float32), torch.tensor(Y_test,
dtype=torch.float32).view(-1, 1)

class MultiRecurrentRNN(nn.Module):
    def __init__(self, n_inputs, n_hidden):
        super(MultiRecurrentRNN, self).__init__()
        self.hidden_layer = nn.Linear(n_inputs + n_hidden, n_hidden)
        self.output_layer = nn.Linear(n_hidden, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x, h_prev):
        x_combined = torch.cat((x, h_prev), dim=1)
        h = self.sigmoid(self.hidden_layer(x_combined))
        y = self.output_layer(h)
        return y, h

def evaluate_model(model, X):
    preds = []
    h_prev = torch.zeros((1, n_hidden))
    for i in range(len(X)):
        pred, h_prev = model(X[i:i+1], h_prev)
        preds.append(pred.item())
    return np.array(preds)

learning_rates = [0.001, 0.005, 0.01, 0.05, 0.1]
best_lr, best_loss = None, float('inf')

print("Подбор оптимального  $\alpha$ :")
for lr in learning_rates:
    model = MultiRecurrentRNN(n_inputs, n_hidden)
    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=lr)

    h_prev = torch.zeros((X_train.shape[0], n_hidden))

    for epoch in range(500):

```

```

        pred, h_prev = model(X_train, h_prev)
        loss = criterion(pred, Y_train)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        h_prev = h_prev.detach()

    test_pred = evaluate_model(model, X_test)
    mse = np.mean((test_pred - Y_test.flatten()).numpy()) ** 2)
    print(f"   α={lr:.3f} -> MSE={mse:.6f}")

    if mse < best_loss:
        best_loss = mse
        best_lr = lr

print(f"\nОптимальное значение α: {best_lr:.3f}, минимальная ошибка: {best_loss:.6f}\n")

model = MultiRecurrentRNN(n_inputs, n_hidden)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=best_lr)

losses = []
h_prev = torch.zeros((X_train.shape[0], n_hidden))

for epoch in range(n_epochs):
    pred, h_prev = model(X_train, h_prev)
    loss = criterion(pred, Y_train)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    h_prev = h_prev.detach()
    losses.append(loss.item())

# 4.    График прогнозируемой функции на участке обучения
train_pred = pred.detach().numpy()
plt.figure(figsize=(10,5))
plt.plot(Y_train.numpy(), label="Эталон")
plt.plot(train_pred, label="Прогноз")
plt.title("График прогнозируемой функции на участке обучения")
plt.legend()
plt.grid()
plt.show()

# 5.    Результаты обучения
# таблица
train_results = pd.DataFrame({
    "Эталонное значение": Y_train.numpy().flatten(),
    "Полученное значение": train_pred.flatten(),

```

```

}))
train_results["Отклонение"] = train_results["Полученное значение"] -
train_results["Эталонное значение"]
print("\nРезультаты обучения (первые 10 значений):")
print(train_results.head(10))

# график
plt.figure(figsize=(8,4))
plt.plot(losses)
plt.title("График изменения ошибки")
plt.xlabel("Эпоха")
plt.ylabel("MSE")
plt.grid()
plt.show()

# 6. Результаты прогнозирования
test_pred = evaluate_model(model, X_test)
test_results = pd.DataFrame({
    "Эталонное значение": Y_test.numpy().flatten(),
    "Полученное значение": test_pred.flatten(),
})
test_results["Отклонение"] = test_results["Полученное значение"] -
test_results["Эталонное значение"]

print("\nРезультаты прогнозирования (первые 10 значений):")
print(test_results.head(10))

```

Для выбора оптимального значения  $\alpha$  была проведена серия экспериментов с параметрами  $[0.001, 0.005, 0.01, 0.05, 0.1]$ . На каждом шаге сеть обучалась в течение 500 эпох, после чего вычислялась MSE на тестовой выборке.

```

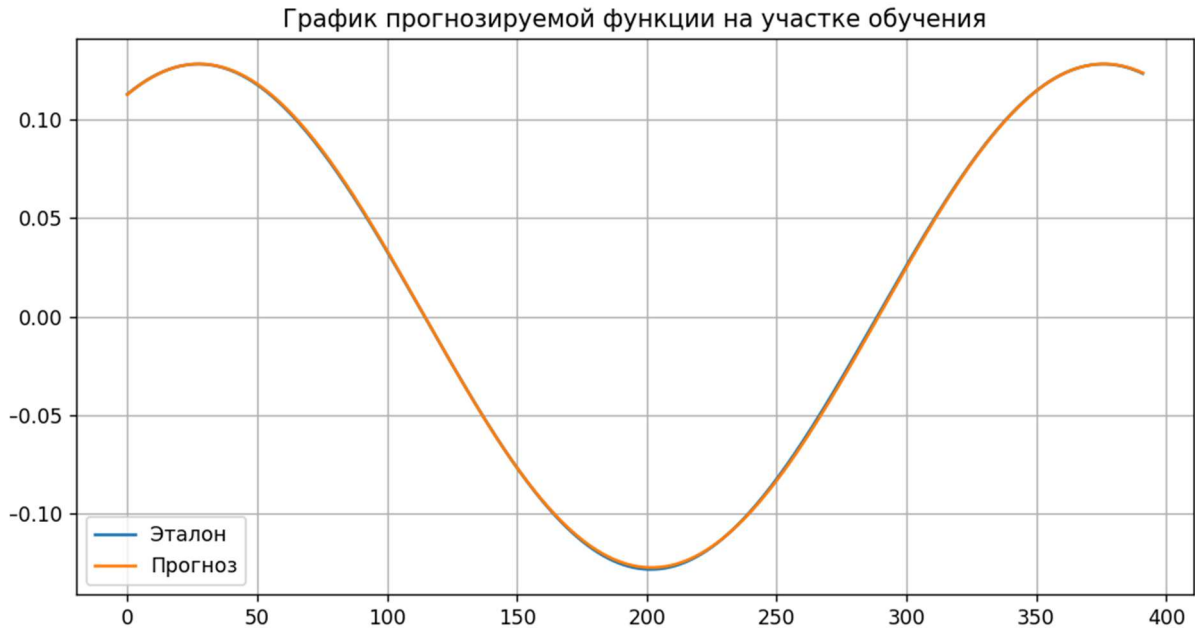
Подбор оптимального  $\alpha$ :
 $\alpha=0.001$  -> MSE=0.000174
 $\alpha=0.005$  -> MSE=0.000120
 $\alpha=0.010$  -> MSE=0.000074
 $\alpha=0.050$  -> MSE=0.000129
 $\alpha=0.100$  -> MSE=0.000403

Оптимальное значение  $\alpha$ : 0.010, минимальная ошибка: 0.000074

```

По результатам видно, что при  $\alpha = 0.01$  достигается минимальная ошибка. Значит, для дальнейшей работы будет использоваться именно это значение.

## График прогнозируемой функции на участке обучения:



На графике показано, насколько хорошо модель повторяет исходную функцию на тех данных, на которых обучалась.

## Результаты обучения:

Таблица со столбцами: эталонное значение, полученное значение, отклонение.

Результаты обучения (первые 10 значений):			
	Эталонное значение	Полученное значение	Отклонение
0	0.112729	0.112592	-0.000136
1	0.113806	0.113676	-0.000130
2	0.114847	0.114722	-0.000125
3	0.115850	0.115731	-0.000119
4	0.116815	0.116702	-0.000113
5	0.117743	0.117635	-0.000108
6	0.118632	0.118530	-0.000102
7	0.119483	0.119386	-0.000097
8	0.120294	0.120203	-0.000091
9	0.121067	0.120982	-0.000085

Отклонения маленькие ( $< 0.0002$ ), что подтверждает успешное обучение сети.

## График изменения ошибки в зависимости от итерации:

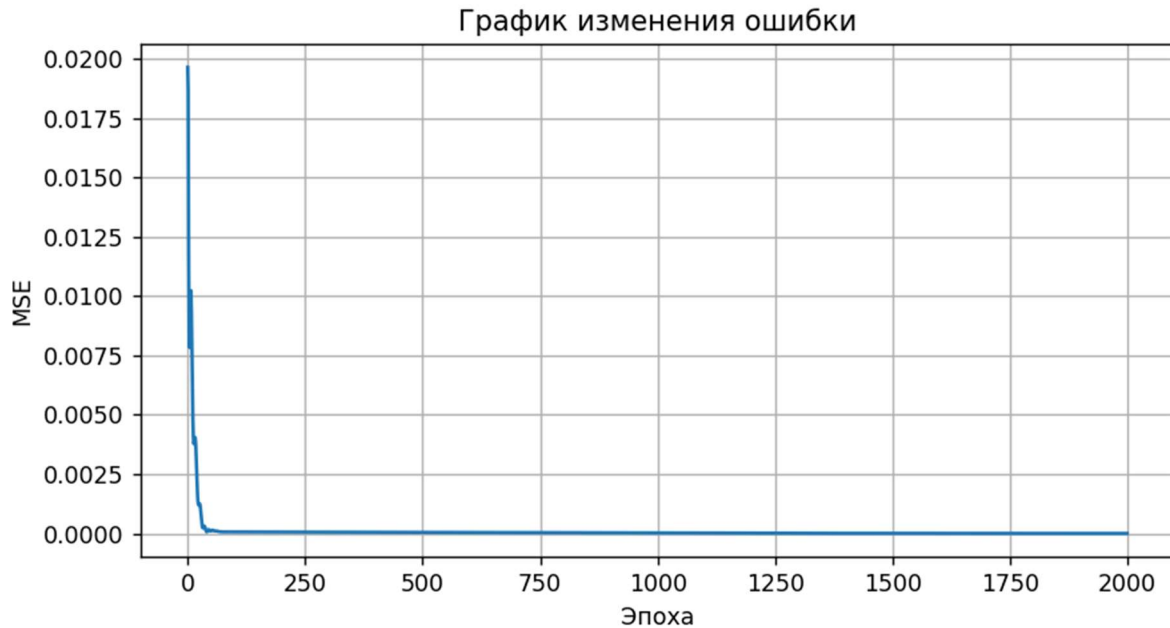


График показывает, что в процессе обучения среднеквадратичная ошибка (MSE) быстро падает и почти сразу стабилизируется. То есть сеть уверенно сходится.

## Результаты прогнозирования:

Результаты прогнозирования (первые 10 значений):			
Эталонное значение	Полученное значение	Отклонение	
0	0.122620	0.134022	0.011402
1	0.121934	0.122562	0.000628
2	0.121209	0.121631	0.000423
3	0.120443	0.120842	0.000399
4	0.119639	0.120054	0.000415
5	0.118796	0.119230	0.000434
6	0.117914	0.118367	0.000453
7	0.116994	0.117466	0.000472
8	0.116035	0.116526	0.000491
9	0.115039	0.115549	0.000509

Разница между эталоном и предсказанием тоже небольшая, что говорит о хорошем качестве прогноза.

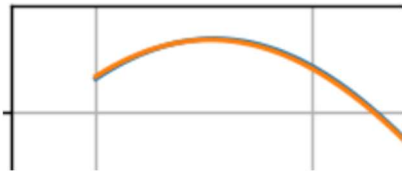
## Сравнение полученных результатов с ЛР5:

В ЛР6 рекуррентная нейронная сеть показала более плохую точность обучения по сравнению с моделью ЛР5 (минимальная ошибка увеличилась с 0.000001 до 0.000074).

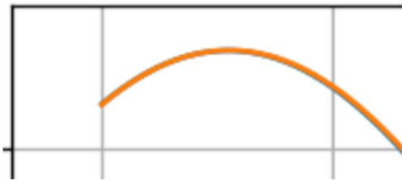
Отклонения при обучении и прогнозировании незначительно снизились (в общем примерно на 0.0003).

Сильных отклонений на графиках замечено не было.

ЛР5:



ЛР6:



**Вывод:** изучила применение нелинейной искусственной нейронной сети с одним скрытым слоем для решения задачи регрессии и прогнозирования, реализовала обучение сети на синтетических данных и оценила точность полученной модели.