

ГУАП

КАФЕДРА № 14

ОТЧЕТ  
ЗАЩИЩЕН С ОЦЕНКОЙ  
ПРЕПОДАВАТЕЛЬ

ассистент

\_\_\_\_\_  
должность, уч. степень, звание

\_\_\_\_\_  
подпись, дата

Н.Н. Эпаев

\_\_\_\_\_  
инициалы, фамилия

## ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №4

Влияние гиперпараметров на процесс обучения нейронной сети и её  
способность к извлечению признаков

по курсу: НЕЙРОННЫЕ СЕТИ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №

1045

\_\_\_\_\_  
подпись, дата

Бондарев К. А.

\_\_\_\_\_  
инициалы, фамилия

Санкт-Петербург 2024

## 1. Цель работы

- 1) Изучить теоретическую часть работы.
- 2) Реализовать персептрон с одним скрытым слоем.
- 3) Путем варьирования параметров модели определить влияние следующих факторов на скорость сходимости алгоритма распознавания: количества нейронов на скрытом слое персептрона, темп обучения.

## 2. Теоретический минимум

## 3. Ход работы

Массив  $X$  с входными данными для обучения нейронной сети:  $[[0, 0], [0, 1], [1, 0], [1, 1]]$ , и массив  $Y$  с ожидаемыми результатами:  $[[0], [1], [1], [0]]$ . Экземпляр класса  $nn$ , передавая массив  $X$ . В конструкторе инициализируются случайные веса между слоями и устанавливается скорость обучения.

Графики до изменения кода (рисунок 1,2):



Рисунок 1 - Функция потерь



Рисунок 2 – Точность классификации

Графики после исправления кода:

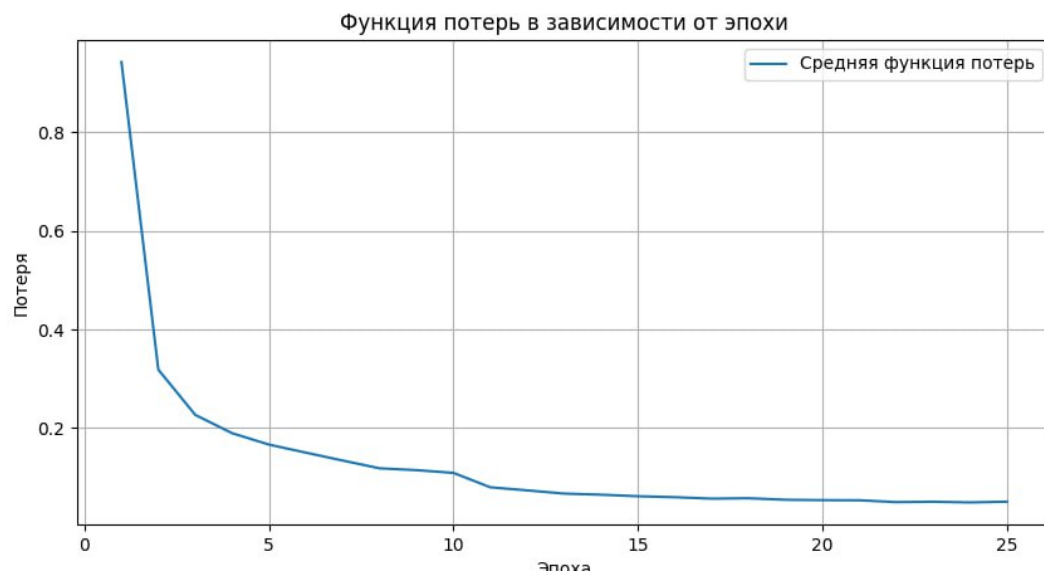


Рисунок 3 – Функция потерь

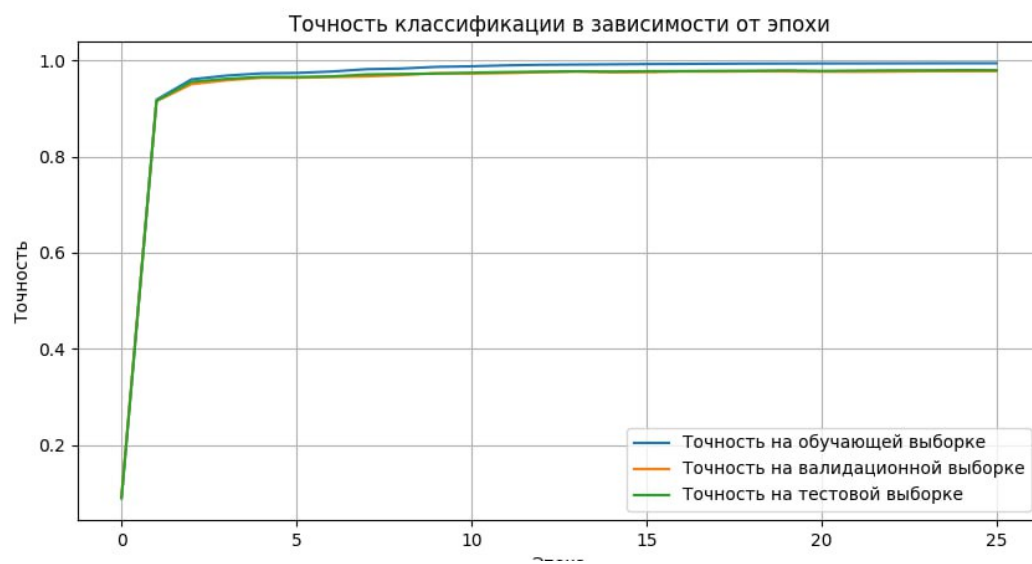


Рисунок 4 – Точность классификации

## Приложение 1.

### Листинг программы

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

class NeuralNetwork:
    def __init__(self, num_inputs, num_hidden_units, num_outputs, learning_rate):
        self.weights_input_hidden = np.random.randn(num_inputs, num_hidden_units)
        self.weights_hidden_output = np.random.randn(num_hidden_units,
num_outputs)
        self.learning_rate = learning_rate

    def sigmoid(self, x):
        return 1.0 / (1.0 + np.exp(-x))

    def sigmoid_derivative(self, x):
        return x * (1.0 - x)

    def forward_propagation(self, inputs):
        hidden = self.sigmoid(np.dot(inputs, self.weights_input_hidden))
        output = self.sigmoid(np.dot(hidden, self.weights_hidden_output))
        return output, hidden

    def loss_function(self, predicted, actual):
        return np.sum((predicted - actual) ** 2 * 0.5) / predicted.shape[0]

    def backward_propagation(self, inputs, hidden, outputs, actual):
        output_error = outputs - actual
        output_delta = output_error * self.sigmoid_derivative(outputs)

        hidden_error = np.dot(output_delta, self.weights_hidden_output.T)
        hidden_delta = hidden_error * self.sigmoid_derivative(hidden)

        self.weights_hidden_output -= np.dot(hidden.T, output_delta) *
self.learning_rate
        self.weights_input_hidden -= np.dot(inputs.T, hidden_delta) *
self.learning_rate

    def train(self, inputs, outputs, iterations, num_hidden_units=None,
learning_rate=None, vary_param=None):
        losses = []
        for i in range(iterations):
            predicted, hidden = self.forward_propagation(inputs)
            self.backward_propagation(inputs, hidden, predicted, outputs)
            loss = self.loss_function(predicted, outputs)
            losses.append(loss)

            if (i + 1) % 1000 == 0:
                print(f"Iteration: {i + 1} | Loss: {loss}")
```

```

    return losses

# Функция для графика решающих границ и функции потерь на одном холсте
def plot_loss_and_decision_boundary(nn, X, y, losses, num_hidden_units=None,
learning_rate=None, vary_param=None):
    fig, axes = plt.subplots(1, 2, figsize=(16, 6))

    # График функции потерь
    axes[0].plot(losses, color='midnightblue', linewidth=2.5)
    axes[0].grid(True, linestyle='--', color='gray', alpha=0.6)
    axes[0].set_ylabel('Loss', fontsize=14, fontweight='bold')
    axes[0].set_xlabel('Iterations', fontsize=14, fontweight='bold')
    title_text = 'Loss Function Over Iterations'
    if vary_param == "hidden_units" and num_hidden_units is not None:
        title_text += f" | Hidden Units: {num_hidden_units}"
    elif vary_param == "learning_rate" and learning_rate is not None:
        title_text += f" | Learning Rate: {learning_rate}"
    axes[0].set_title(title_text, fontsize=16, fontweight='bold',
color='darkblue')

    # График решающих границ
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                        np.arange(y_min, y_max, 0.01))
    grid_points = np.c_[xx.ravel(), yy.ravel()]
    Z, _ = nn.forward_propagation(grid_points)
    Z = Z.reshape(xx.shape)

    axes[1].contourf(xx, yy, Z, levels=[0, 0.5, 1], cmap='coolwarm', alpha=0.75)
    axes[1].contour(xx, yy, Z, levels=[0.5], colors='black', linewidths=1.2)
    scatter = axes[1].scatter(X[:, 0], X[:, 1], c=y.flatten(), s=100,
edgecolors='k', marker='o', cmap='viridis', alpha=0.9)
    for i, txt in enumerate(y.flatten()):
        axes[1].annotate(f'Class {int(txt)}', (X[i, 0] + 0.1, X[i, 1] + 0.1),
fontsize=10, color='black')

    legend1 = axes[1].legend(*scatter.legend_elements(), loc="upper right",
title="Classes", title_fontsize='13', fontsize='11')
    axes[1].add_artist(legend1)

    axes[1].set_xlabel('Feature 1', fontsize=14, fontweight='bold')
    axes[1].set_ylabel('Feature 2', fontsize=14, fontweight='bold')
    axes[1].set_xlim(x_min, x_max)
    axes[1].set_ylim(y_min, y_max)
    axes[1].grid(True, linestyle=':', color='black', alpha=0.3)
    title_text = 'Enhanced Decision Boundary'
    if vary_param == "hidden_units" and num_hidden_units is not None:
        title_text += f" | Hidden Units: {num_hidden_units}"

```

```

        elif vary_param == "learning_rate" and learning_rate is not None:
            title_text += f" | Learning Rate: {learning_rate}"
            axes[1].set_title(title_text, fontsize=18, fontweight='bold',
                             color='darkred')

    plt.tight_layout()
    plt.show()

# Параметры нейронной сети и обучение
num_inputs = 2
num_hidden_units = 4
num_outputs = 1
learning_rate = 1
nn = NeuralNetwork(num_inputs, num_hidden_units, num_outputs, learning_rate)
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])
iterations = 25000
losses = nn.train(X, y, iterations, num_hidden_units=num_hidden_units,
                  learning_rate=learning_rate)

plot_loss_and_decision_boundary(nn, X, y, losses,
                               num_hidden_units=num_hidden_units, learning_rate=learning_rate)

# Варьирование количества скрытых нейронов
for num_hidden in [2, 4, 8]:
    nn = NeuralNetwork(num_inputs, num_hidden, num_outputs, learning_rate)
    losses = nn.train(X, y, iterations, num_hidden_units=num_hidden,
                      vary_param="hidden_units")
    plot_loss_and_decision_boundary(nn, X, y, losses,
                                   num_hidden_units=num_hidden, vary_param="hidden_units")

# Варьирование темпа обучения
for lr in [0.1, 0.5, 1]:
    nn = NeuralNetwork(num_inputs, num_hidden_units, num_outputs, lr)
    losses = nn.train(X, y, iterations, learning_rate=lr,
                      vary_param="learning_rate")
    plot_loss_and_decision_boundary(nn, X, y, losses, learning_rate=lr,
                                   vary_param="learning_rate")

```