Домашнее задание по теме “Введение в нейросети”

Постановка задачи:
*№4**: Придумайте собственную несложную функцию, которую нельзя реализовать с помощью одного нейрона, но можно с помощью нескольких (2-4). Доказательство, программа, эксперименты.*

Решение:
В качестве примера предложу функцию определения точки с координатами (x,y) принадлежности к прямоугольнику на плоскости. Действительно, решить эту задачу с помощью одного нейрона не получится, так как оно требует нелинейной границы и активационная функция нейрона даст линейное разделение, при этом для нахождения точки в прямоугольнике нужно комбинировать условия по обеим координатам. С использованием же нескольких нейронов, каждый из которых проверяет одно условие, можно создать усложненное поле решений, где пересечение условий даст нам нужный результат.
Напишем нейросеть с 4 нейронами, которая будет определять, находится ли точка внутри заданного прямоугольника.
Так, функция должна проверять 4 условия:

$$X_{min} <= X <= X_{max}$$
$$Y_{min} <= Y <= Y_{max}$$

Каждое из условий представим отдельным нейроном. Нейросеть будет с двумя слоями: 1-й слой будет состоять из четырех нейронов, каждый из которых проверяет одно из вышеперечисленных условий; на выходе 2-го слоя будет нейрон, который активируется, если все условия выполняются.

Код решения:

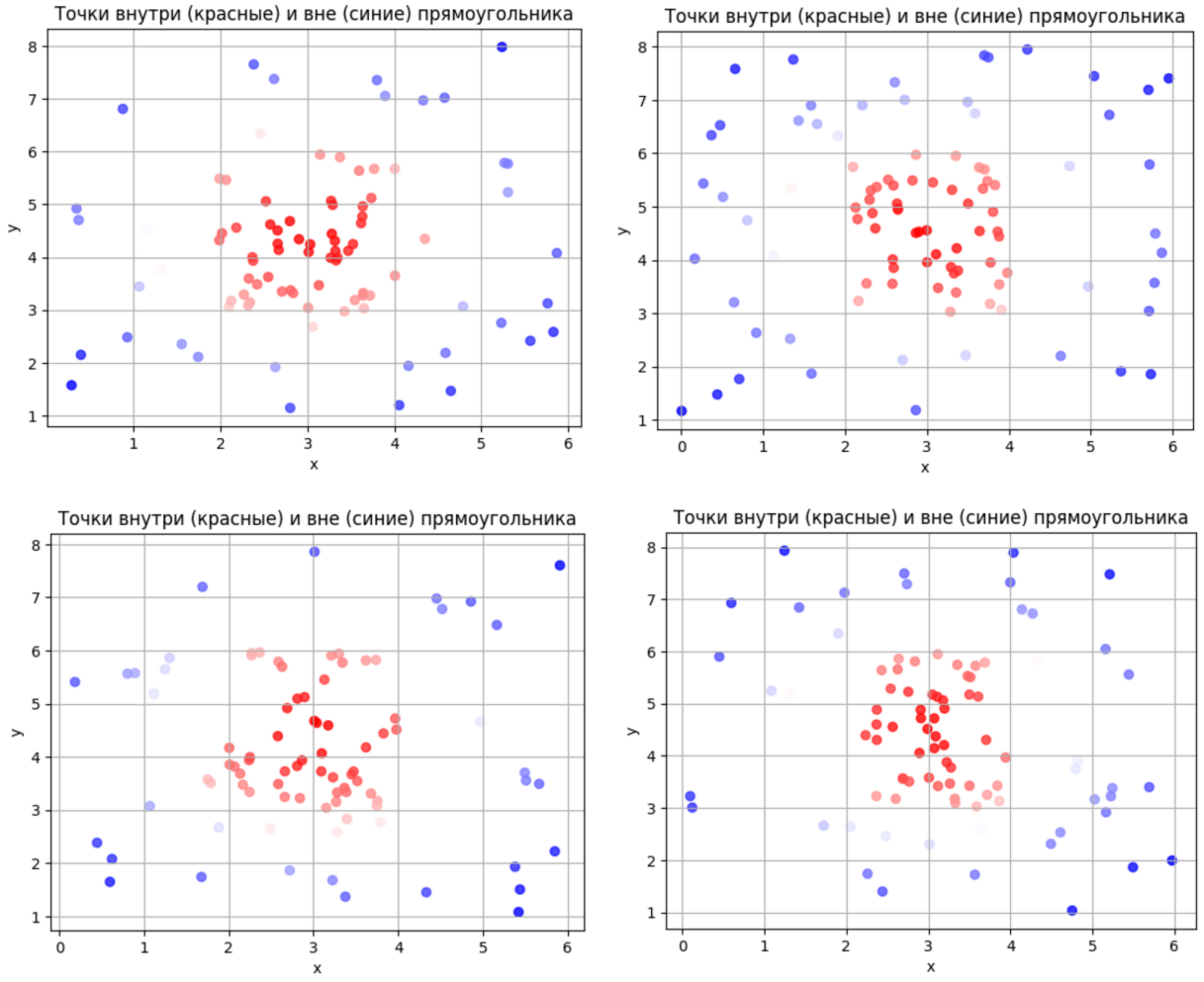
```
#библиотека, которая определяет нейросеть, преобразует данные в формат тензоров и применяет операции для предсказаний
import torch
import torch.nn as nn
#генерирует и обрабатывает массивы данных (в т.ч. координаты точек)
import numpy as np
#для визуализации данных
import matplotlib.pyplot as plt

#границы прямоугольника
x_min, x_max, y_min, y_max = 2, 4, 3, 6

#создаем набор данных
def generate_data(n=100): #по умолчанию 100 точек
    #генерация координат точек внутри прямоугольника, по 50 для каждой
    x_inside = np.random.uniform(x_min, x_max, n // 2)
    y_inside = np.random.uniform(y_min, y_max, n // 2)
```

```
    #массив, где каждому эл-ту присвоено единичное значение (точка внутри)
    labels_inside = np.ones(n // 2)
    #аналогично, но точки в т.ч. вне прямоугольника
    x_outside = np.random.uniform(x_min - 2, x_max + 2, n // 2)
    y_outside = np.random.uniform(y_min - 2, y_max + 2, n // 2)
    #отбор-маска точек не попадающих в прямоугольник
    outside_mask = ~((x_outside >= x_min)&(x_outside <= x_max)&(y_outside
>= y_min)&(y_outside <= y_max))
    #применяем эту маску на точках, которые должны быть вне прям-ка
    x_outside, y_outside = x_outside[outside_mask], y_outside[outside_mask]
    #массив, где каждому эл-ту присвоено нулевое значение (точка НЕ внутри)
    labels_outside = np.zeros(len(x_outside))

    #объединяем все что получилось
    x = np.concatenate([x_inside, x_outside[:n // 2]])
    y = np.concatenate([y_inside, y_outside[:n // 2]])
    labels = np.concatenate([labels_inside, labels_outside[:n // 2]])

    #итоговые подготовленные данные для анализа
    data = np.stack([x, y], axis=1)
      return torch.tensor(data, dtype=torch.float32), torch.tensor(labels,
dtype=torch.float32).reshape(-1, 1)

#создаем данные
data, labels = generate_data()

# Определение модели
class SimpleRectangleNN(nn.Module):
    def __init__(self): #инициализация и настройка нейронов
        super(SimpleRectangleNN, self).__init__()
        #4 нейрона: каждый проверяет одно из условий (2 входа и 1 выход)
        self.neuron1 = nn.Linear(2, 1)  # x >= x_min
        self.neuron2 = nn.Linear(2, 1)  # x <= x_max
        self.neuron3 = nn.Linear(2, 1)  # y >= y_min
        self.neuron4 = nn.Linear(2, 1)  # y <= y_max

        #устанавливаем веса для проверки границ
        with torch.no_grad():
            self.neuron1.weight[:] = torch.tensor([[1.0, 0.0]])
            self.neuron1.bias[:] = torch.tensor([-x_min])
            self.neuron2.weight[:] = torch.tensor([[-1.0, 0.0]])
            self.neuron2.bias[:] = torch.tensor([x_max])
            self.neuron3.weight[:] = torch.tensor([[0.0, 1.0]])
            self.neuron3.bias[:] = torch.tensor([-y_min])
            self.neuron4.weight[:] = torch.tensor([[0.0, -1.0]])
            self.neuron4.bias[:] = torch.tensor([y_max])

    def forward(self, x):
        #сигмоиды (наши ф-ии активации): от 0 до 1 для каждого условия
        cond1 = torch.sigmoid(self.neuron1(x))
        cond2 = torch.sigmoid(self.neuron2(x))
        cond3 = torch.sigmoid(self.neuron3(x))
        cond4 = torch.sigmoid(self.neuron4(x))
        #объединяем все условия: если все условия выполлнены, точка внутри
        return cond1 * cond2 * cond3 * cond4

#инициализация модели
model = SimpleRectangleNN()

#применяем эту модель на наших данных (вычисление градиента выключено для
ускорения): получаем numpy-массив
```

```
with torch.no_grad():
    predictions = model(data).numpy()

#визуализация и выдача результата с помощью графика: заметим, что некоторые точки будут полупрозрачны, т.к. значения на выходе сети (т.е. предсказания) лежат в диапазоне от 0 до 1, что может приводить к "полутонам"
plt.scatter(data[:, 0], data[:, 1], c=predictions, cmap="bwr", alpha=0.8)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Точки внутри (красные) и вне (синие) прямоугольника")
plt.grid(True) #показ сетки
plt.show()
```

Результаты:

Так, для прямоугольника с границами x = 2, x = 4, y = 3, y = 6 после нескольких запусков программы получаем следующие результаты:



Как можно видеть из графиков, наша нейросеть успешно построена и выдает качественные результаты.

Зверев Дмитрий, 325-я группа АЯ ВМК МГУ