# Лабораторная работа № 4

## Сети с радиальными базисными элементами

Цель работы: исследование свойств некоторых видов сетей с радиальными базисными элементами, алгоритмов обучения, а также применение сетей в задачах классификации и аппроксимации функции.

Студент Мариничев И.А.

Группа М8О-408Б-19

Вариант 5

Определим три группы точек (эллипсы с поворотом), соответствующие трем классам

def ellipse(t, a, b, x0, y0):  
 x = x0 + a \* np.cos(t)  
 y = y0 + b \* np.sin(t)  
 return x, y

def rotate(x, y, alpha):  
 xr = x \* np.cos(alpha) - y \* np.sin(alpha)  
 yr = x \* np.sin(alpha) + y \* np.cos(alpha)  
 return xr, yr

t = np.linspace(0, 2 \* np.pi, 200)

# Эллипс: a = 0.4, b = 0.15, α = π/6, x0 = −0.1, y0 = 0.15  
x1, y1 = ellipse(t, a=0.4, b=0.15, x0=-0.1, y0=0.15)  
x1, y1 = rotate(x1, y1, np.pi / 6.)

# Эллипс: a = 0.7, b = 0.5, α = −π/3, x0 = 0, y0 = 0  
x2, y2 = ellipse(t, a=0.7, b=0.5, x0=0., y0=0.)  
x2, y2 = rotate(x2, y2, -np.pi / 3.)

# Эллипс: a = 1, b = 1, α = 0, x0 = 0, y0 = 0  
x3, y3 = ellipse(t, a=1., b=1., x0=0., y0=0.)  
x3, y3 = rotate(x3, y3, 0.)

points1 = [[x, y] for x, y in zip(x1, y1)]  
points2 = [[x, y] for x, y in zip(x2, y2)]  
points3 = [[x, y] for x, y in zip(x3, y3)]  
  
classes1 = [[1., 0., 0.] for \_ in range(len(points1))]  
classes2 = [[0., 1., 0.] for \_ in range(len(points2))]  
classes3 = [[0., 0., 1.] for \_ in range(len(points3))]  
  
X = points1 + points2 + points3  
y = classes1 + classes2 + classes3

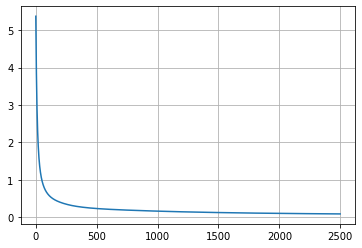
Будем обучать нашу сеть батчами размера batch\_size, так как данных уже довольно много.

Создадим класс слоя радиально-базисной функции (РБФ)

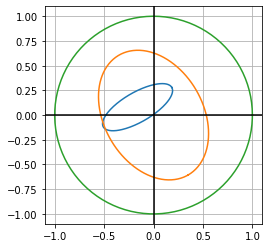
# норма  
def l\_norm(x, p=2):  
 return torch.norm(x, p=p, dim=-1)  
  
# радиально-базисная функция (мультиквадратичная)  
def rbf\_multiquadric(x):  
 return (1 + x.pow(2)).sqrt()

class RBFLayer(nn.Module):  
 def \_\_init\_\_(self, in\_features: int, num\_kernels: int, out\_features: int):  
 super(RBFLayer, self).\_\_init\_\_()  
 self.in\_features = in\_features  
 self.num\_kernels = num\_kernels   
 self.out\_features = out\_features  
 self.\_make\_parameters()  
  
 def \_make\_parameters(self):  
 self.weights = nn.Parameter(torch.zeros(self.out\_features, self.num\_kernels, dtype=torch.float32))  
 self.kernels\_centers = nn.Parameter(torch.zeros(self.num\_kernels, self.in\_features, dtype=torch.float32))  
 self.log\_shapes = nn.Parameter(torch.zeros(self.num\_kernels, dtype=torch.float32))  
 self.reset()  
  
 def reset(self, upper\_bound\_kernels: float = 1.0, std\_shapes: float = 0.1, gain\_weights: float = 1.0):  
 nn.init.uniform\_(self.kernels\_centers, a=-upper\_bound\_kernels, b=upper\_bound\_kernels)  
 nn.init.normal\_(self.log\_shapes, mean=0.0, std=std\_shapes)  
 nn.init.xavier\_uniform\_(self.weights, gain=gain\_weights)  
  
 def forward(self, input: torch.Tensor):  
 batch\_size = input.size(0)  
  
 # рассчитывам расстояния до центров  
 mu = self.kernels\_centers.expand(batch\_size, self.num\_kernels, self.in\_features)  
 diff = input.view(batch\_size, 1, self.in\_features) - mu  
  
 # применяем нормировочную функцию  
 r = l\_norm(diff)  
  
 # применяем параметр формы  
 eps\_r = self.log\_shapes.exp().expand(batch\_size, self.num\_kernels) \* r  
  
 # применяем радиально-базисную функцию  
 rbfs = rbf\_multiquadric(eps\_r)  
  
 # в качестве ответа даем линейную комбинацию  
 out = self.weights.expand(batch\_size, self.out\_features, self.num\_kernels) \* rbfs.view(batch\_size, 1, self.num\_kernels)  
  
 return out.sum(dim=-1)  
  
 @property  
 def get\_kernels\_centers(self):  
 return self.kernels\_centers.detach()  
  
 @property  
 def get\_weights(self):  
 return self.weights.detach()  
  
 @property  
 def get\_shapes(self):  
 return self.log\_shapes.detach().exp()

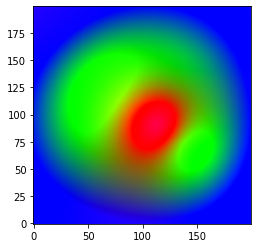
Наша сеть будет принимать на вход два признака, координаты (x, y), а на выходе будет выдавать три значения в диапазоне [0, 1], чтобы их можно было интерпретировать как цветовую компоненту RGB. В качестве функции потерь берем nn.MSELoss(). Обучим модель. Посмотрим на график функции потерь, вычисляющей MSE между исходными и полученными данными.



Теперь соберем предсказания модели для каждой точки области [-1, 1] x [-1, 1], построим наши три класса



И посмотрим на цветовое представление того, как наша сеть справилась с разделением области на три класса, которые линейно неразделимы



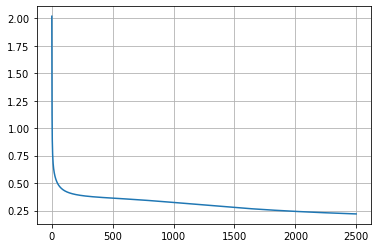
Теперь перейдем к задаче аппроксимации функции. Создадим разреженную дискретную версию нашей исходной функции и будем использовать ее для обучения, чтобы потом получить при увеличенни шага приближение исходной функции

def function(t):  
 return np.cos(-3 \* t\*\*2 + 5 \* t + 10)

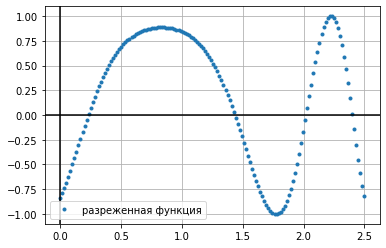
t1 = np.linspace(0, 2.5, 150)  
f1 = function(t1)  
  
t2 = np.linspace(0, 2.5, 2000)  
f2 = function(t2)

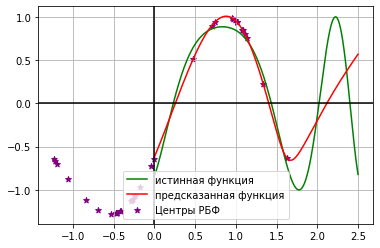
Наша сеть будет принимать на вход один признак, координату x, а на выходе будет выдавать значение функции в этой точке. В качестве функции потерь берем nn.MSELoss().Будем обучать нашу сеть батчами размера batch\_size, так как данных уже довольно много. Обучим модель.

Посмотрим на график функции потерь, вычисляющей MSE между исходными и полученными данными



Cоберем предсказания модели и визуализируем полученные результаты, сравним приближение, разреженный вариант и истинную функцию





Выводы: в ходе данной работы была построена сеть основанная на радиально-базисной функции, которая была использована для решения двух типов задач:

* классификация (линейно неразделимые данные)
* аппроксимация

После обучения (2500 эпох) были получены довольно неплохие результаты.