Основы

# Пространственные функции

Стандартное построение графика

x = torch.randn(1000)

y = torch.randn(1000)

f = torch.sin(2\*x) + torch.sin(2\*y)

fig = plt.figure(figsize=(9,9))

fig.scatter(x,y,c=f, s =5)

Построение графика пространства

x = torch.randn(1000)

y = torch.randn(1000)

space = torch.meshgrid(x,y, indexing='ij')

X,Y = space

f = torch.sin(2\*X) + torch.sin(2\*Y) - предсказания для каждой точки пространства

Такое пространство нельзя визуализировать как поле нужно стандартным образом:  
f = torch.sin(2\*x) + torch.sin(2\*y)

fig = plt.figure(figsize=(9,9))

fig.scatter(x,y,c=f, s =5)

Можно только линейное пространство визуализировать

x = torch.linspace(-5,5,1000)

y = torch.linspace(-5,5,1000)

space = torch.meshgrid(x,y, indexing='ij')

X,Y = space

f = torch.sin(2\*X) + torch.sin(2\*Y)

1)

# Создаем 3D график

fig = plt.figure(figsize=(10, 8))

ax = fig.add\_subplot(111, projection='3d')

# Преобразуем X, Y и f в массивы NumPy

X\_np = X.detach().numpy()

Y\_np = Y.detach().numpy()

f\_np = f.detach().numpy()

# Строим поверхность

ax.plot\_surface(X\_np, Y\_np, f\_np, cmap='viridis')

2)

# Визуализация как тепловая карта

plt.imshow(f.detach().numpy(), cmap='viridis', origin='lower', extent=[x.min().item(), x.max().item(), y.min().item(), y.max().item()])

plt.colorbar(label='f(x, y)') # Добавляем цветовую шкалу

# Градиент

[torch.gradient — PyTorch 2.6 documentation](https://pytorch.org/docs/stable/generated/torch.gradient.html#torch.gradient)

>>> # Estimates the gradient of f(x)=x^2 at points [-2, -1, 2, 4]

>>> coordinates = (torch.tensor([-2., -1., 1., 4.]),)

>>> values = torch.tensor([4., 1., 1., 16.], )

>>> torch.gradient(values, spacing = coordinates)

(tensor([-3., -2., 2., 5.]),)

Приёмы

# Создание линейного равномерного n-мерного пространства точек

В конечном итоге нужно получить датасет

x y z ….

x1 y1 z1 ….

x2 y2 z3 ….

1) Создаём n осей координат

n\_samples = 25

min = -100

max = 100

column1 = torch.linspace(min, max, steps = n\_samples)

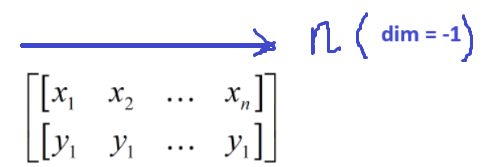
column2 = torch.linspace(min, max, steps = n\_samples)

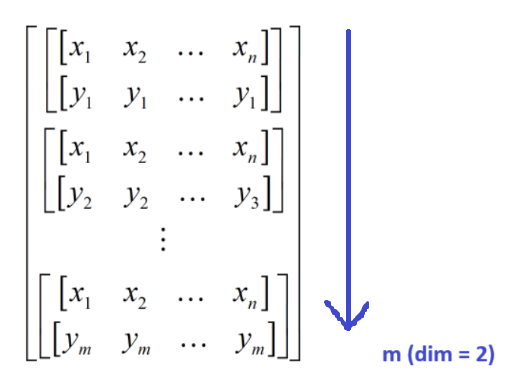
column3 = torch.linspace(min, max, steps = n\_samples)

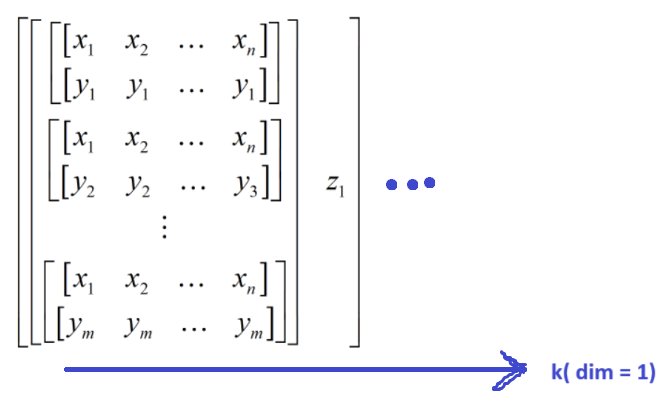
column4 = torch.linspace(min, max, steps = n\_samples)

2) Создаем сетку из всех комбинаций значений

x,y,z,n = torch.meshgrid(column1, column2, column3, column4)







и так далее для всех z

Таким образом получается тензор (k,m,n)

3) Создаём датасет

Итого чтобы получить тензор вида:

x y z ….

x1 y1 z1 ….

x2 y2 z3 ….

x\_flat = x.reshape(-1, 1)

y\_flat = y.reshape(-1, 1)

z\_flat = z.reshape(-1, 1)

n\_flat = n.reshape(-1, 1)

# 2. Объединяем столбцы в один тензор

space = torch.cat((x\_flat, y\_flat, z\_flat, n\_flat), dim=1)

print(space.shape)

print(space)

## Код

n\_samples = 25

min = -100

max = 100

column1 = torch.linspace(min, max, steps = n\_samples)

column2 = torch.linspace(min, max, steps = n\_samples)

column3 = torch.linspace(min, max, steps = n\_samples)

column4 = torch.linspace(min, max, steps = n\_samples)

x,y,z,n = torch.meshgrid(column1, column2, column3, column4)

x\_flat = x.reshape(-1, 1)

y\_flat = y.reshape(-1, 1)

z\_flat = z.reshape(-1, 1)

n\_flat = n.reshape(-1, 1)

space = torch.cat((x\_flat, y\_flat, z\_flat, n\_flat), dim=1)

print(space.shape)

print(space)

# Пятна разности на равномерной поверхности

#@title Пятна разницы на сетке

import numpy as np

from scipy.spatial import KDTree

def diff(x\_meta\_embed, predictions\_meta, x\_embed, predictions):

x0 = x\_meta\_embed

x\_meta\_embed = x0[:,0]

y\_meta\_embed = x0[:,1]

x1 = x\_embed

x\_embed = x1[:,0]

y\_embed = x1[:,1]

# Создаем KDTree для второго пространства

points\_meta\_embed = np.column\_stack((x\_meta\_embed, y\_meta\_embed))

tree = KDTree(points\_meta\_embed)

# Определяем радиус окрестности

radius = 0.1 # Выберите подходящий радиус в зависимости от ваших данных

# Создаем массив для хранения разниц, инициализируем нулями или NaN

difference\_array = np.zeros\_like(predictions\_meta) # или np.full\_like(predictions\_meta2, np.nan)

# Для каждой точки из первого пространства находим ближайшие точки во втором пространстве

for i in range(len(x\_embed)):

point = np.array([x\_embed[i], y\_embed[i]])

# Находим все точки в окрестности

indices = tree.query\_ball\_point(point, radius)

if indices:

# Вычисляем среднее значение z для окрестности

mean\_z = np.mean(predictions\_meta[indices])

# Вычисляем разницу между z точки из первого пространства и средним z окрестности

diff = predictions[i] - mean\_z

# Записываем разницу в массив difference\_array для всех точек окрестности

difference\_array[indices] = diff

# Теперь differences содержит разницы для каждой точки из первого пространства

return difference\_array

# Создание поля (сетки), соответствующей данным датасета

import torch

import matplotlib.pyplot as plt

cnt = 100

x0 = torch.randn(cnt, requires\_grad=True)

y0 = torch.randn(cnt, requires\_grad=True)

vals0 = torch.randn(cnt, requires\_grad=True)

# Сортируем x0 и y0

x0\_sorted, x0\_indices = torch.sort(x0)

y0\_sorted, y0\_indices = torch.sort(y0)

# Создаем сетку для отсортированных значений

X, Y = torch.meshgrid(x0\_sorted, y0\_sorted, indexing='xy')

# Инициализируем vals средним значением vals0

vals\_mean = vals0.mean().item()

vals = torch.full\_like(X, vals\_mean)

# Заполняем значения в vals на основе индексов

for i in range(len(x0)):

x1 = torch.where(x0\_indices == i)[0]

y1 = torch.where(y0\_indices == i)[0]

vals[y1, x1] = vals0[i]

# Первый график

fig00 = plt.figure()

plt.scatter(x0.detach().numpy(), y0.detach().numpy(), c=vals0.detach().numpy(), cmap='viridis')

plt.colorbar()

fig00.show()

# # Второй график

fig00 = plt.figure()

plt.scatter(X.detach().flatten().numpy(), Y.detach().flatten().numpy(), c=vals.detach().flatten().numpy(), cmap='viridis')

plt.colorbar()

fig00.show()

# Второй график

# fig10 = plt.figure()

# plt.imshow(vals.detach().numpy(), extent=[x0.min().item(), x0.max().item(),

# y0.min().item(), y0.max().item()],

# origin='lower', cmap='viridis')

# plt.colorbar()

# fig10.show()

# Разбиение данных на n-мерные кубы

Разбиение датасета любой размерности на n-мерные кубы, каждый элемент которых имеет индекс куба. Каждый куб имеет уникальный индекс. количество кубов можно настроить

import torch

import matplotlib.pyplot as plt

import numpy as np

# Создаем данные

#torch.manual\_seed(42) # Для воспроизводимости

x = torch.rand(10000, 3) \* 10 # 100 точек в 2D-пространстве (2 столбца)

\_,ids = torch.sort(x[:,0])

DATA1 = x[ids]

print(x)

num\_bins = 3 # Количество корзин для каждого измерения

# Создаем bins для каждого измерения

bins\_list = []

for i in range(x.shape[1]):

bin\_axis = torch.linspace(DATA1[:, i].min(), DATA1[:, i].max(), num\_bins+1) # Границы корзин

bins\_list.append(bin\_axis)

print(bins\_list)

# Применяем bucketize для каждого измерения

bin\_inds = torch.zeros\_like(DATA1, dtype=torch.int64) # Для хранения индексов корзин

for i in range(DATA1.shape[1]):

bin\_inds[:, i] = torch.bucketize(DATA1[:, i], bins\_list[i], right=True)

print(bin\_inds)

# Вычисляем линейные индексы для 2D-кубов

linear\_inds = torch.zeros(x.shape[0], dtype=torch.int64) # Линейные индексы

for i in range(DATA1.shape[1]):

linear\_inds += bin\_inds[:, i] \* (num\_bins \*\* i) # Обобщенная формула

print(linear\_inds)

# Визуализация

# plt.figure(figsize=(10, 8))

# scatter = plt.scatter(DATA1[:, 0].numpy(), DATA1[:, 1].numpy(), c=linear\_inds.numpy(), cmap='viridis', s=50, edgecolor='black')

# plt.colorbar(scatter, label='Linear Index of Bin')

# plt.show()

import plotly.graph\_objects as go

fig = go.Figure()

fig.add\_trace(go.Scatter3d(x = DATA1[:,0].numpy().flatten(), y = DATA1[:,1].numpy().flatten(), z = DATA1[:,2].numpy().flatten(), mode = 'markers', marker = dict( color = linear\_inds.numpy().flatten()) ))

fig.show()

Линейная алгебра

# SVD decomposition

[How to Calculate the SVD from Scratch with Python -](https://machinelearningmastery.com/singular-value-decomposition-for-machine-learning/) [MachineLearningMastery.com](http://machinelearningmastery.com)