Равномерное распределение точек на сфере

Часть 1. Постановка задачи. Расчёт функционала.

- Аудитория: начальный уровень deep learning
- В результате прочтения: опыт применения векторизированных матричных вычислений для решения реальных задач; более полное понимание разных методов минимизации функции многих переменных, их параметров; опыт визуализации данных с помощью PovRay и vapory.

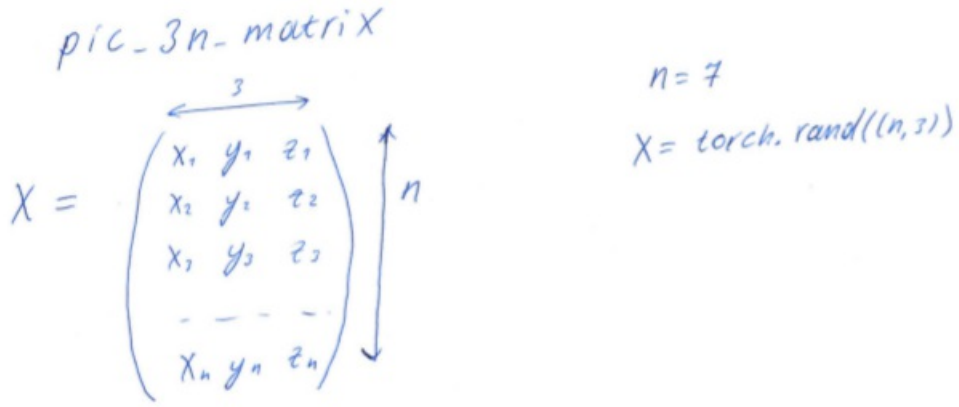
Откуда задача

> Если в вашей компании есть человек, целый день смотрящий видосики и чатящийся на форумах - это датасайнтист. И у него ОБУЧАЕТСЯ.

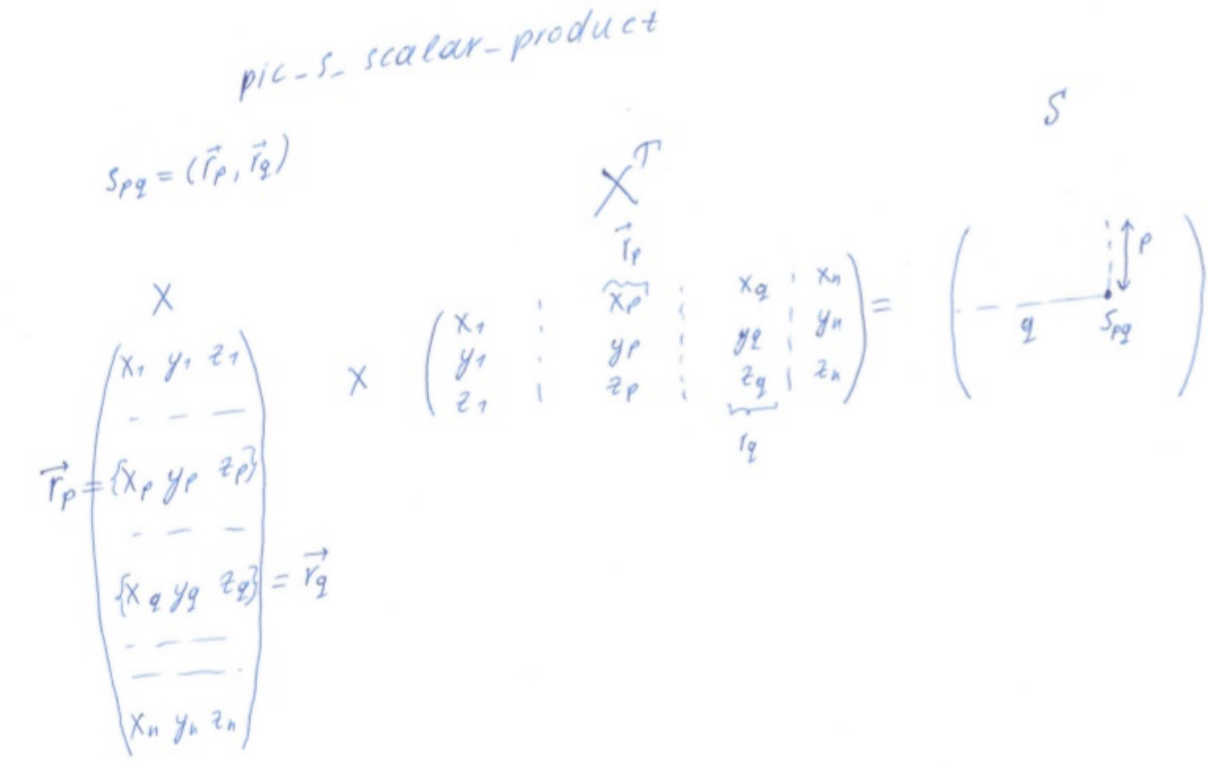
Чаще всего процесс обучения моделей machine learning (ML) - это подбор параметров, минимизирующих некоторый функционал. При этом контролировать этот процесс можно, глядя на одно единственное число. Давайте рассмотрим задачу, в которой мы будем минимизировать довольно сложный функционал, но при этом в любой момент времени сможем видеть, что у нас получилось. Будем искать равномерное распределения по сфере заданного количества n точек. Такое распределение бывает нужно акустику для того, чтобы понять, в каком направлении запустить волну в кристалле. Связисту - чтобы узнать как расположить на орбите спутники для достижения наилучшего качества связи. Метеорологу - как разместить станции слежения за погодой.
Для некоторых n задача решается легко. Например, если $n = 8$, то мы можем взять куб и его вершины будут являться ответом к задаче. Так же нам повезёт, если n будет равно количеству вершин икосаэдра, додекаэдра или другого платонова тела. В противном случае задача не столь проста.
Для достаточно большого количества точек есть формула с эмпирически подобранными коэффициентами. Но есть и более универсальное, хотя и более сложное решение, которому посвящена данная статья. Разбросаем n точек случайно, а потом заставим их притягиваться к какой-то поверхности, например, к сфере и отталкиваться друг от друга. Притяжение и отталкивание определяются функцией - потенциалом. Значение функции мало - точки легли как нужно.

Считаем потенциал

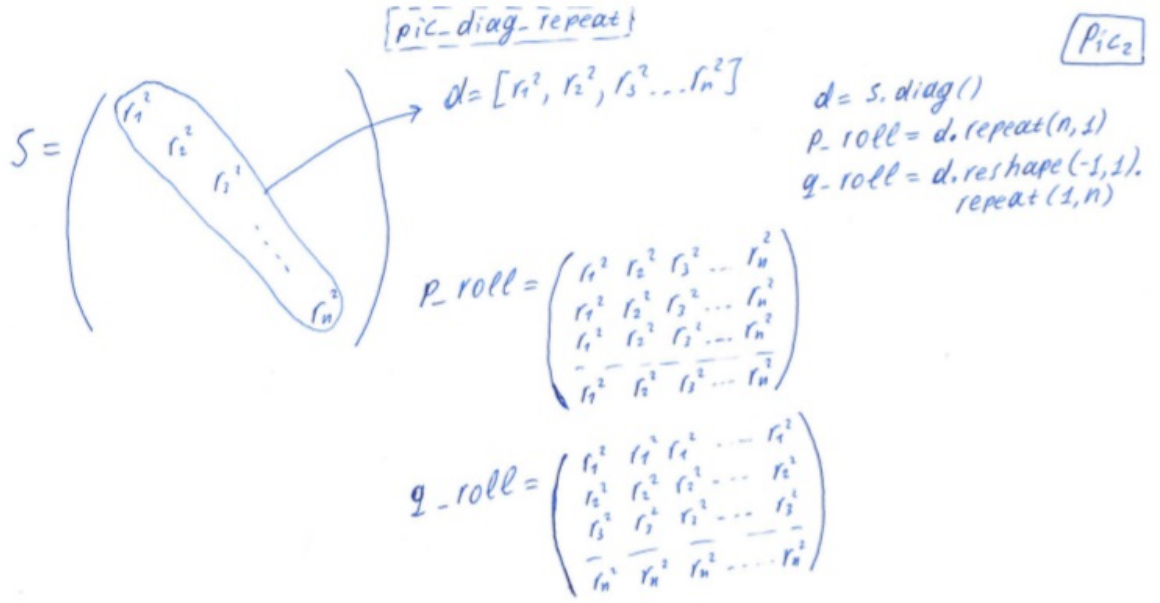
В школе нам рассказали две формулы. $F = -k_1 x$ - сила упругой деформации и $F = k_2 \cdot q_1 \cdot q_2 / r^2$ - сила электростатического взаимодействия. Потенциал упругого взаимодействия двух точек $u_1 = k_1 r^2 / 2$, электростатического - $u_2 = k_2 / r$.
Чтобы посчитать потенциал попарного взаимодействия n точек, запишем их координаты в виде матрицы $3 \times n$.



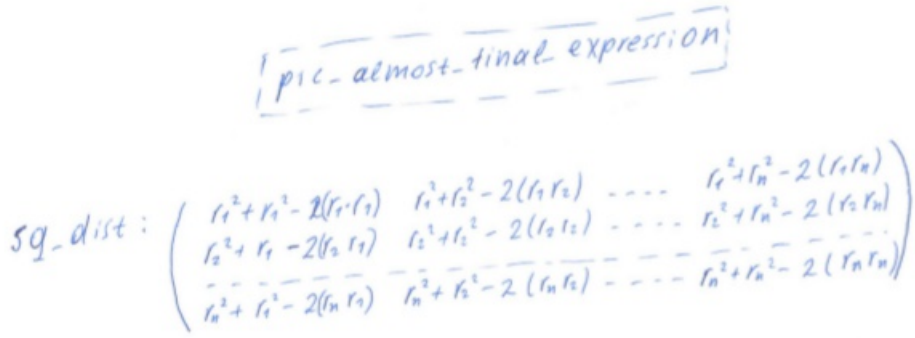
Матрица $S = x \cdot x^T$ обладает многими интересными свойствами и часто встречается в выкладках, связанных с теорией линейных классификаторов в ML. Так, если мы посмотрим на строчки матрицы X с индексами p и q как на вектора трёхмерного пространства r^p, r^q то обнаружим что матрица S состоит из скалярных произведений этих векторов.



На диагонали матрицы S стоят квадраты длин векторов r^p : $s_{pp} = r_p^2$. Зная это, давайте считать полный потенциал взаимодействия. Начнём с расчёта двух вспомогательных матриц. В одной диагональ матрицы S будет повторяться в строках, в другой - в столбцах.



Посмотрим теперь на значение выражения `p_roll + q_roll - 2 * d.diag`



Элемент с индексами (p, q) матрицы `sq_dist` равен $r_p^2 + r_q^2 - 2 \cdot (r^p, r^q) = (r^p - r^q)^2$. То есть, у нас получилась матрица квадратов расстояний между точками. Выражение sq_dist.sum() даст одно-единственное число - полный потенциал попарного упругого взаимодействия между всеми точками. Попытки минимизировать этот потенциал не дают ничего хорошего. При данном потенциале взаимодействия точка сильнее чувствует точки, которые находятся от неё далеко. А чтобы получить равномерное распределение, нам нужно чтобы она в первую очередь обращала внимание на своих ближайших соседей.

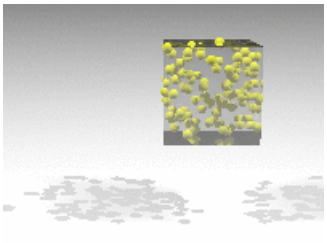
Электростатическое отталкивание на сфере

`dist = torch.sqrt(sq_dist)` - матрица расстояний между точками. Нам нужно посчитать потенциал, учитывающий отталкивание точек между собой и притяжение к сфере. Поставим на диагональ единицы и заменим каждый элемент на обратный ему (только не подумайте, что мы при этом обратили матрицу!): `rec_dist_one = 1 / (dist + torch.eye(n))`. Получилась матрица, на диагонали которой стоят единицы, другие элементы - потенциалы электростатического взаимодействия между точками.

Добавим теперь квадратичный потенциал притяжения к поверхности единичной сферы. Расстояние от поверхности сферы $(1 - r)$. Возводим его в квадрат и умножаем на k, который задаёт соотношение между ролью электростатического отталкивания частиц и притяжения сферы. Итого `k = 1000`, `all_interactions = rec_dist_one - torch.eye(n) + (d.sqrt() - torch.ones(n))**2`. Долгожданный таргет, который мы будем минимизировать: `t = all_interactions.sum()`

Визуализация

В наше время данные можно визуализировать средствами огромного количества пакетов, таких как Matlab, Wolfram Mathematics, Mapple, Matplolib и т.д. и т.п. В этих пакетах очень много сложных функций, делающих сложные вещи. К сожалению, если перед тобой стоит простая, но нестандартная задача, ты оказываешься безоружен. Моё любимое решение в такой ситуации - povray. Это очень мощная программа, которую обычно применяют для создания фотореалистичных изображений, но её можно использовать как “ассемблер визуализации”. Обычно, сколь бы сложной не была поверхность, которую хочется отобразить, достаточно попросить povray нарисовать сферы с центрами, лежащими на этой поверхности.
С помощью библиотеки vapory можно создать povray сцену прямо в python, отрендерить её и посмотреть на результат. Сейчас он выглядит так:



Далее будет рассказано о том, как запустить минимизацию функционала, чтобы точки вышли из куба и равномерно расползлись по сфере.

> Written with StackEdit.