МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«АДЫГЕЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра прикладной математики, информационных технологий и информационной безопасности

«ДОПУСКАЕТСЯ К ЗАЩИТЕ»
Заведующий кафедрой
_____ Алиев М.В.
« ___» _____20___г.

КУРСОВАЯ РАБОТА

Направление подготовки *01.03.02 Прикладная математика и информатика*
Направленность *«Системное программирование и компьютерные технологии»*

Тема
Поиск оптимальных упаковок кругов при помощи алгоритмов оптимизации пакета PyTorch

Научный руководитель _____ к.т.н., доцент Воронов В.А. _____

Обучающийся 2ПМ факультет математики и компьютерных наук Никитенко В.К.

Майкоп 2021

Содержание

Введение

Курсовая работа посвящена применению современных методов оптимизации, в частности стохастического градиентного спуска, к задаче поиска оптимальной упаковки кругов.

Тема была выбрана по причине актуальности нейронных сетей и машинного обучения, а также личного интереса к программированию и поставленной цели.

Цель работы – применить алгоритм стохастического градиентного спуска к задаче об оптимальной упаковке кругов.

Основные задачи работы:

- Научиться работать с фреймворком PyTorch на базовом уровне;
- Улучшить скорость работы алгоритма оптимизации за счет выбора параметров;
- Найти процент оптимальных конфигураций, вычисляемых алгоритмом оптимизации, предназначенным для обучения нейронных сетей, со случайным начальным приближением;
- Максимизировать долю оптимальных результатов;
- Собрать и проанализировать получаемые данные.

Фреймворк PyTorch

PyTorch — современная библиотека глубокого обучения, разработанная и развивающаяся компанией Facebook. Его разработка началась в 2012 году, но открытым и доступным широкой публике PyTorch стал лишь в 2017 году. С этого момента фреймворк очень быстро набирает популярность и привлекает внимание всё большего числа исследователей.

Под глубоким обучением, как правило, понимают обучение функции, представляющей собой композицию множества нелинейных преобразований. Такая сложная функция ещё называется потоком или графом вычислений. Фреймворк глубокого обучения должен уметь делать всего три вещи:

- Определять граф вычислений;
- Дифференцировать граф вычислений;
- Вычислять его.

Тензорные вычисления — основа PyTorch, каркас, вокруг которого наращивается вся остальная функциональность.

Тензор — это обобщение векторов и матриц на более высокие измерения. Например, тензор второго ранга — это матрица, тензор третьего ранга – это массив матриц.

Во многих вычислительных задачах удается многократно повысить быстродействие при помощи операций над тензорами, поскольку такие операции эффективно выполняются на графических ускорителях (GPU). В настоящей работе на данный момент вычисления на GPU не используются. Применение GPU является одним из возможных направлений дальнейшей работы.

Постановка задачи

Задачи упаковки — это класс задач оптимизации в математике, в которых пытаются упаковать геометрические объекты заданной формы (круги, треугольники, квадраты, шары) в некоторый контейнер. Цель поиска оптимальной упаковки — либо упаковать отдельный контейнер как можно плотнее, либо упаковать все объекты, использовав как можно меньше контейнеров, либо нахождение конфигурации, которая упаковывает один контейнер с максимальной плотностью. При этом объекты не должны пересекаться и объекты не должны пересекать стены контейнера.

Если задача поиска оптимальной упаковки кругов на плоскости может быть решена элементарными методами, то гипотеза Кеплера о плотнейшей упаковке шаров в трехмерном пространстве была доказана лишь в конце XX века путем весьма трудоемкого компьютерного перебора. Для задач упаковки объектов в контейнер, как правило, оптимальность доказана только при небольших значениях числа объектов.

Контейнер — обычно одна двумерная или трёхмерная выпуклая область или бесконечная область. Множество объектов может содержать различные объекты с заданными размерами, или один объект фиксированных размеров, который может быть использован несколько раз.

Многие из таких задач могут относиться к упаковке предметов в реальной жизни, вопросам складирования и транспортировки.

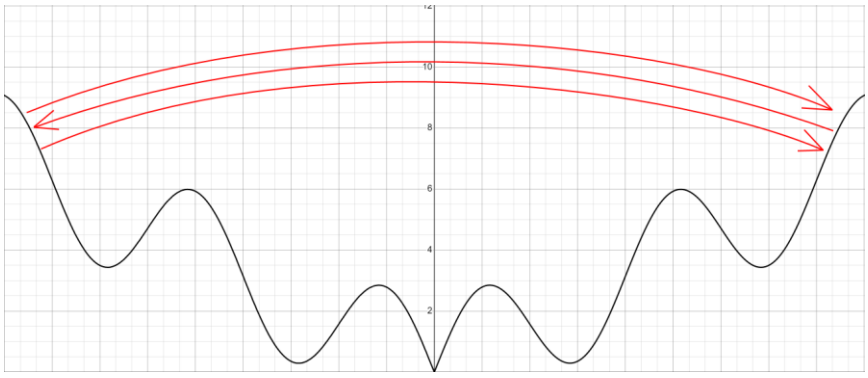
Мы будем наиболее подробно рассматривать упаковку одинаковых кругов в квадрат.

Градиентный спуск

Градиентный спуск — это эвристический алгоритм, который выбирает случайную точку, рассчитывает направление скорейшего убывания функции (пользуясь градиентом функции в данной точке), а затем пошагово рассчитывает новые значения функции, двигаясь в выбранную сторону. Если убывание значения функции становится слишком медленным, алгоритм останавливается и говорит, что нашел минимум.

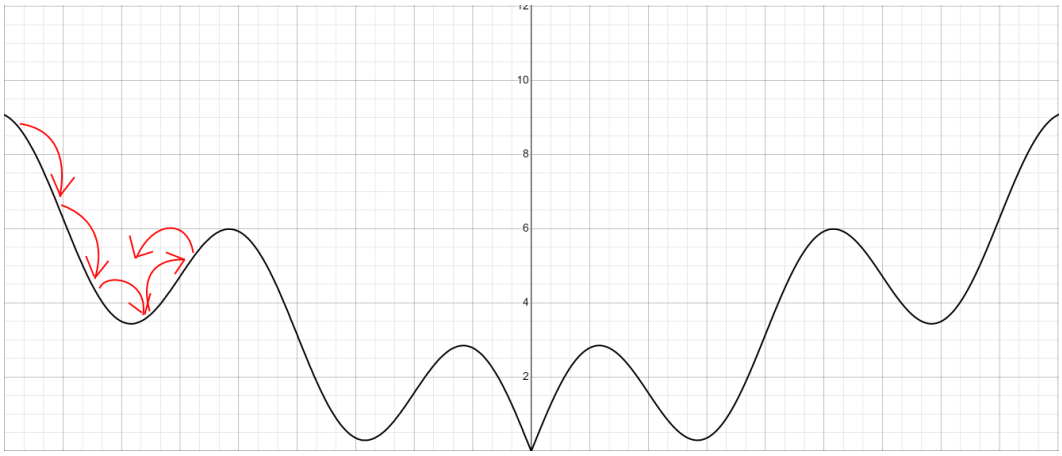
Градиентный спуск выбирает случайную точку, находит направление самого быстрого убывания функции и двигается до ближайшего минимума вдоль этого направления. Размер одного шага можно настроить. Если на каком-то этапе разность между старой точкой (до шага) и новой снижается ниже предела, считается, что минимум найден, алгоритм завершен. Таким образом алгоритм будет работать до тех пор, пока разность не станет пренебрежимо мала.

Размер шага алгоритма определяет, насколько мы собираемся двигать точку на графике функции потерь, и этот параметр называется «скоростью обучения». Но не всегда высокая скорость обучения гарантирует хороший результат. Скорость обучения стоит воспринимать как ширину шагов. В некоторых случаях бывает так, что слишком широкие шаги вообще не позволяют достичь минимума, и машина бесконечно перешагивает через него, затем градиент «разворачивает» ее обратно, и алгоритм снова перескакивает через минимум.



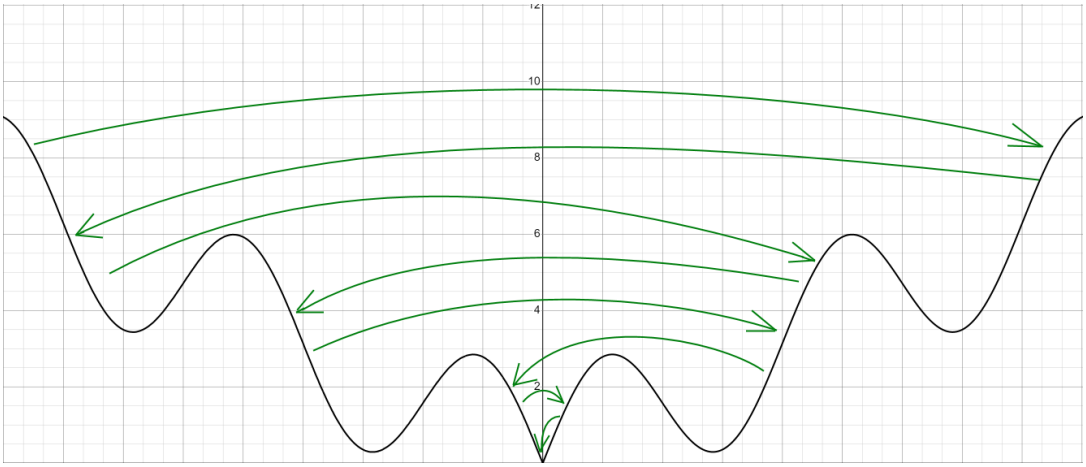
Скорость обучение слишком велика

Слишком маленькая скорость обучения тоже может не дать нужного результата, так как можно попасть в локальный минимум, из которого уже не получится выйти. Также чем ниже скорость обучения, тем больше времени требуется на вычисления.



Скорость обучения слишком мала

Поэтому будем использовать затухающую скорость обучения. Программа начинает работать с большой скоростью обучения, когда разность между старой точкой и новой перестает меняться или меняется слишком незначительно указанное количество шагов (значение patience), скорость обучения понижается. Таким образом, постепенно доходим до минимума функции.



Затухающая скорость обучения

Значение Patience

При затухающей скорости обучения важно знать, в какой момент нужно уменьшить скорость. Для этого используется значение patience. Чтобы определить, какое значение даст наилучший результат, было решено провести большое количество запусков программы при разных значениях, при этом подсчитывая среднее арифметическое радиусов кругов и записывая всю полученную информацию в документ. Таким образом получилось выяснить примерные лучшие варианты.

```
 20_10: (100, 14.419323831796646, 0.14419323831796646)
 30_10: (100, 14.557886108756065, 0.14557886108756066)
 40_10: (100, 14.594097211956978, 0.14594097211956977)
 50_10: (100, 14.652108877897263, 0.14652108877897263)
  60_10: (100, 14.65576159954071, 0.1465576159954071)
 70_10: (100, 14.666099473834038, 0.14666099473834038)
 80_10: (100, 14.678517773747444, 0.14678517773747443)
 90_10: (100, 14.695189774036407, 0.14695189774036407)
100_10: (100, 14.747388735413551, 0.14747388735413552)
110_10: (100, 14.727175563573837, 0.14727175563573838)
120_10: (100, 14.719042524695396, 0.14719042524695397)
130_10: (100, 14.752179980278015, 0.14752179980278016)
140_10: (100, 14.742900416254997, 0.14742900416254998)
150_10: (100, 14.759741485118866, 0.14759741485118866)
160_10: (100, 14.738642171025276, 0.14738642171025276)
170_10: (100, 14.739636912941933, 0.14739636912941934)
180_10: (100, 14.766204684972763, 0.14766204684972764)
190_10: (100, 14.741886541247368, 0.14741886541247368)
 200_10: (100, 14.74767418205738, 0.1474767418205738)
210_10: (100, 14.759666308760643, 0.14759666308760644)
220_10: (100, 14.758980199694633, 0.14758980199694632)
 230_10: (100, 14.75654511153698, 0.14756545111536978)
240_10: (100, 14.767590835690498, 0.14767590835690497)
250_10: (100, 14.766855612397194, 0.14766855612397195)
260_10: (100, 14.757829666137695, 0.14757829666137695)
270_10: (100, 14.743579804897308, 0.14743579804897308)
280_10: (100, 14.758969902992249, 0.14758969902992247)
290_10: (100, 14.758339539170265, 0.14758339539170265)
300_10: (100, 14.746790930628777, 0.14746790930628775)
 300_10: (200, 29.56049780547619, 0.14780248902738094) ПИК
310_10: (200, 29.511401757597923, 0.1475570087879896)
320_10: (200, 29.537805780768394, 0.14768902890384197)
330_10: (200, 29.536685064435005, 0.14768342532217502)
340_10: (200, 29.518293246626854, 0.14759146623313427)
350_10: (200, 29.521195843815804, 0.14760597921907903)
360_10: (200, 29.534051343798637, 0.1476702567189932)
370_10: (200, 29.527116358280182, 0.1476355817914009)
380_10: (200, 29.525327995419502, 0.14762663997709752)
390_10: (200, 29.555556058883667, 0.14777778029441835)
   400_10: (200, 29.514304921031, 0.147571524605155)
```

Пример полученных данных

Разберем на примере первой строки смысл полученных результатов:

- 20 – значение patience;
- 10 – количество шагов;
- 100 – количество итераций при данной конфигурации;
- 14.419... – сумма радиусов;
- 0.144... – среднее значение радиуса.

Расчеты были проведены для 10, 15, 20, 25 и 30 кругов:

Количество кругов	**Лучшее значение patience**
10	300
15	320
20	370
25	380
30	390

По полученным результатам можно сказать, что с ростом количества кругов лучшие результаты получаются на больших значениях patience, но разница между ними уменьшается. Также стоит учитывать, что большую роль играет фактор случайности, т.к. центры кругов генерируются случайным образом.

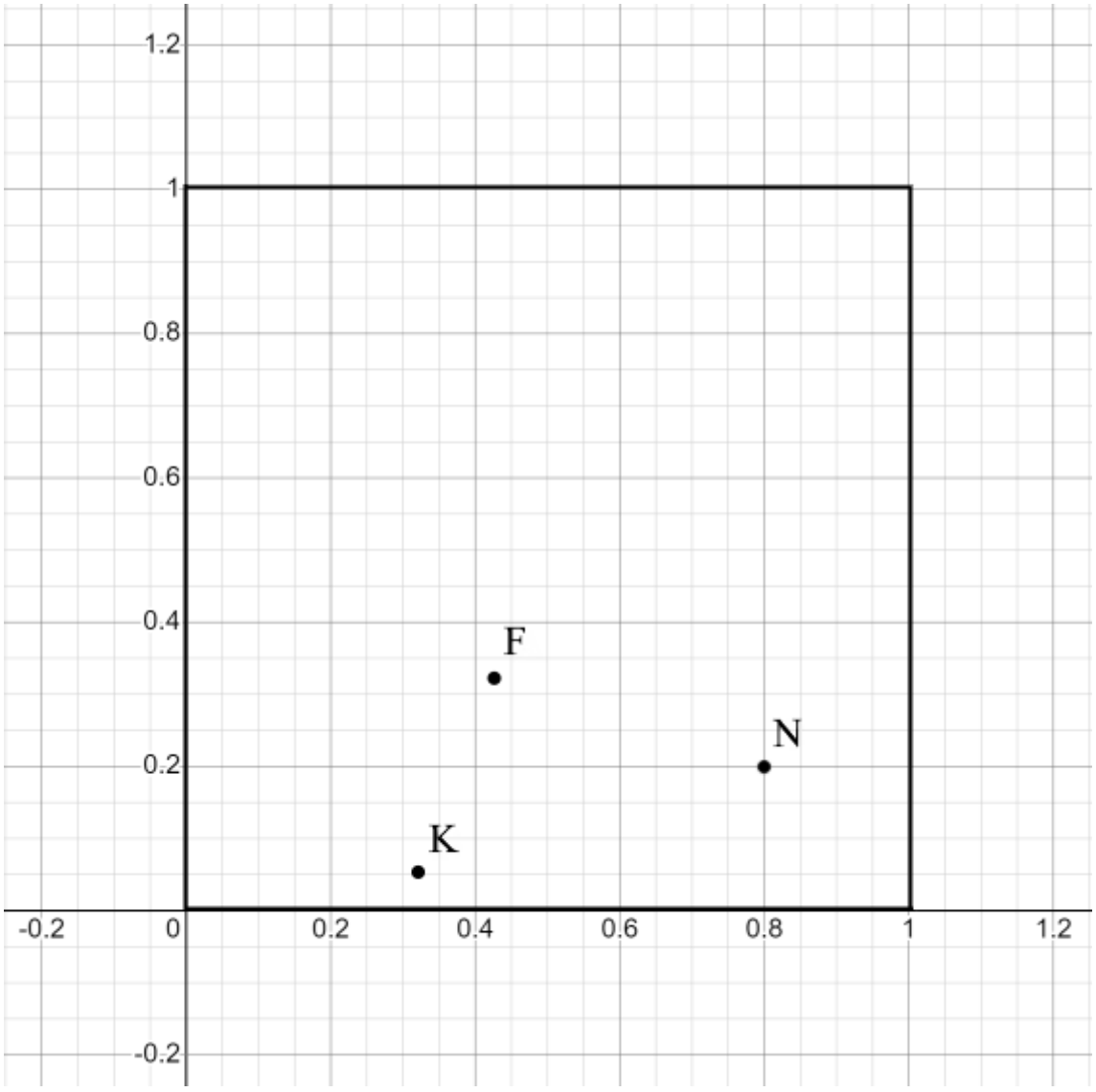
Поэтому для дальнейших вычислений было выбрано среднее из лучших значений – 320.

Математическое решение задачи

Рассмотрим пример решения при следующих данных: $M_1(0,1)$, $M_2(1,1)$, $M_3(1,0)$, $M_4(0,0)$ — вершины квадрата, $F(0.4263,\ 0.3223)$, $K(0.3210,\ 0.0534)$, $N(0.8, 0.2)$ - центры кругов.

Сначала построим область по заданным координатам вершин. Найдем общие уравнения прямых $(Ax_0 + By_0 + C = 0)$ по формуле: $(y_1 - y_2)x + (x_2 - x_1)y + (x_1y_2 - x_2y_1) = 0$, получим: M_1M_2: $y - 1 = 0$; M_2M_3: $x - 1 = 0$; M_3M_4: $-y = 0$; M_4M_1: $x = 0$.

Получаем область:



Далее находим попарно расстояние между центрами кругов по формуле: $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$, делим на 2 и выбираем наименьшее из них. Находим расстояние от центров кругов до сторон квадрата по формуле: $d = \frac{|Ax_0 + By_0 + C|}{\sqrt{A^2 + B^2}}$, берем наименьшее из них. Выбираем наименьшее значение из двух полученных, оно и будет является новым радиусом кругов. Далее сдвигаем центры кругов и повторяем тот же алгоритм, таким образом на n-ном шаге находим лучшее значение радиуса.

Реализация алгоритма на Python

Для нахождения уравнений прямых квадрата был создан класс выпуклых многоугольников:

```
def __init__(self, center=[0.0, 0.0], pts=[]):
    self.lines = []
    self.center = center
    self.points = pts.copy()
    for i in range(len(pts) - 1):
        self.addline(pts[i], pts[i + 1])
    self.addline(pts[0], pts[len(pts) - 1])

def addline(self, a, b):
    d = np.array(b) - np.array(a)
    n = np.array([-d[1], d[0]])
    n = n / np.linalg.norm(n)
    c = n.dot(a)
    if n.dot(self.center) - c < 0:
        n = -n
        c = -c
    self.lines.append([n, c])

def prepare(self):
    n = len(self.lines)
    self.U = np.zeros([n, 2])
    self.v = np.zeros(n)
    k = 0
    for l in self.lines:
        self.U[k, :] = l[0]
        self.v[k] = l[1]
        k += 1
```

При создании переменной этого класса, отправляем в конструктор координаты вершин и центра многоугольника. Вызывая метод .prepare() получаем всю информацию о уравнениях прямых: значения A, B и C ($Ax_0 + By_0 + C = 0$). Как уже говорилось выше, важной особенностью фреймворка PyTorch является работа с тензорами и отсутствие циклов, поэтому всю информацию храним в виде тензоров.

```
poly = CPoly(center=[0.5, 0.5], pts=[[1, 1], [1, 0], [0, 0], [0, 1]])
poly.prepare()
U = torch.Tensor(poly.U)  # Значения A и B
v = torch.Tensor(poly.v).reshape((len(poly.lines), 1))  # Значение C
```

Переходим к градиентному спуску. Генерируем случайным образом центры кругов и настраиваем градиентный спуск, его скорость обучения.

```
x = torch.rand((n, 2), requires_grad=True)
optimizer = torch.optim.Adam([x], lr=lr)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1, patience=patience)
```

Преобразуем формулу расстояния между центрами кругов: $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} = \sqrt{({x_1}^2 + {x_2}^2 + {y_1}^2 + {y_2}^2) - 2(x_1x_2 + y_1y_2)}$.

Посчитаем отдельно обе скобки подкоренного выражения:

1) Пусть $X = \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \end{pmatrix}$ – тензор координат центров двух кругов.

Возводим тензор X в квадрат, записываем полученные значения в тензор X2. Создаём тензор X2S, записываем в него сумму строк тензора X2:

X2S = $({x_1}^2 + {y_1}^2 \quad {x_2}^2 + {y_2}^2)$, складываем его с таким же тензором, записанным в другой форме X2S.reshape = $\begin{pmatrix} {x_1}^2 + {y_1}^2 \\ {x_2}^2 + {y_2}^2 \end{pmatrix}$, получаем: $\begin{pmatrix} 2({x_1}^2 + {y_1}^2) & {x_1}^2 + {y_1}^2 + {x_2}^2 + {y_2}^2 \\ {x_1}^2 + {y_1}^2 + {x_2}^2 + {y_2}^2 & 2({x_2}^2 + {y_2}^2) \end{pmatrix}$.

2) Для нахождения второй скобки умножаем тензор X на X^T и домножаем на -2, получим: $\begin{pmatrix} -2{x_1}^2 - 2{y_1}^2 & -2x_1x_2 - 2y_2y_2 \\ -2x_1x_2 - 2y_2y_2 & -2{x_2}^2 - 2{y_2}^2 \end{pmatrix}$.

Применяя булеву маску к сумме найденных скобок получим искомое подкоренное выражение. Вычисляя корень из этого числа и деля полученный результат на 2 найдем тензор радиусов кругов (1).

```
x2 = torch.square(x)
x2s = torch.sum(x2, 1)
distm = - 2 * x.mm(x.t()) + x2s + x2s.reshape((n, 1))
dist_points = 0.5 * torch.sqrt(torch.masked_select(distm, mask))
```

Найдем тензор расстояний от центров кругов до сторон квадрата. Для этого умножим тензор U, состоящий из значений A и B уравнений прямых, на

тензор X^T и отнимем V – тензор значений C (2). На $\sqrt{A^2 + B^2}$ не делим, т.к. при данных координатах квадрата это значение всегда равняется 1.

```
dist_lines = U.mm(x.t()) - v
```

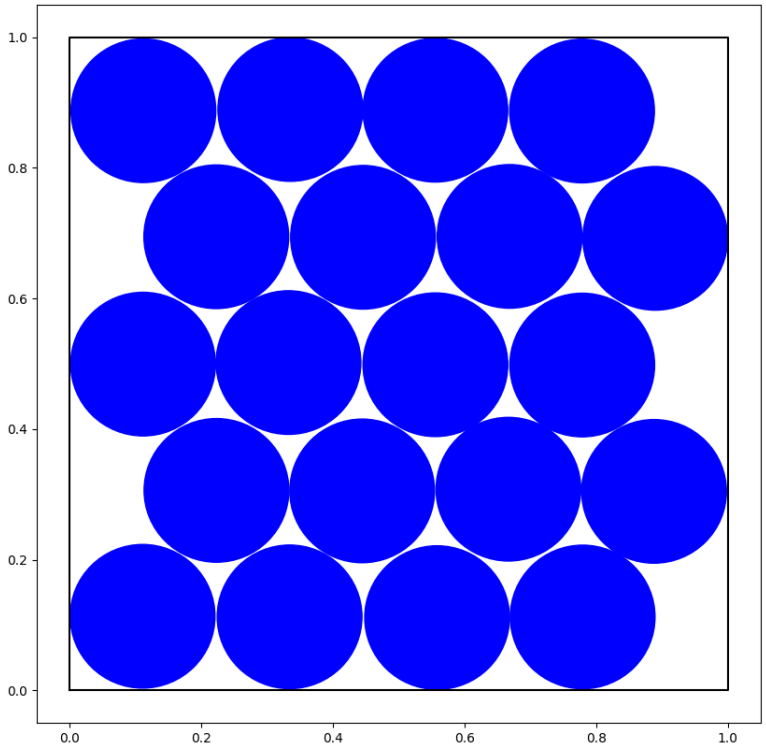
Отбираем наименьшее из значений (1) и (2), оно и будет новым радиусом кругов. Далее производятся расчеты градиентов и алгоритм повторяется снова с учетом полученных данных, при этом меняя начальное положение центров кругов.

```
y = -torch.min(torch.min(dist_points), torch.min(dist_lines))
y.backward(retain_graph=True)
optimizer.step()
scheduler.step(y)
```

Остается лишь вывести на экран полученный результат:

```
fig, ax = plt.subplots(figsize=(10, 10))
ax.plot([0, 0, 1, 1, 0], [0, 1, 1, 0, 0], 'k')
for i in range(n):
  c = plt.Circle(x0[i,:], y, color='blue')
  ax.add_patch(c)
plt.show()
```

Получаем следующее изображение:



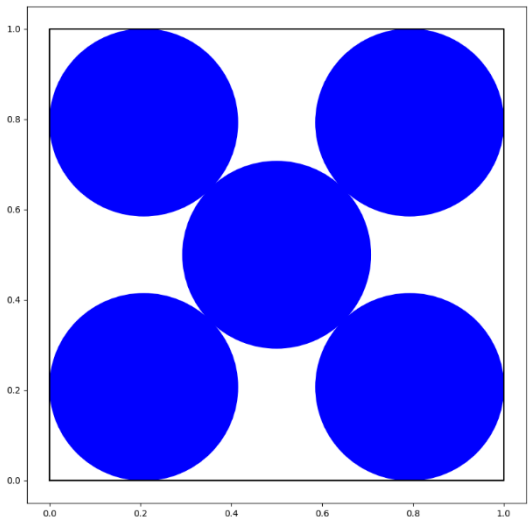
Пример выполнения программы для 20 кругов

Полученные результаты

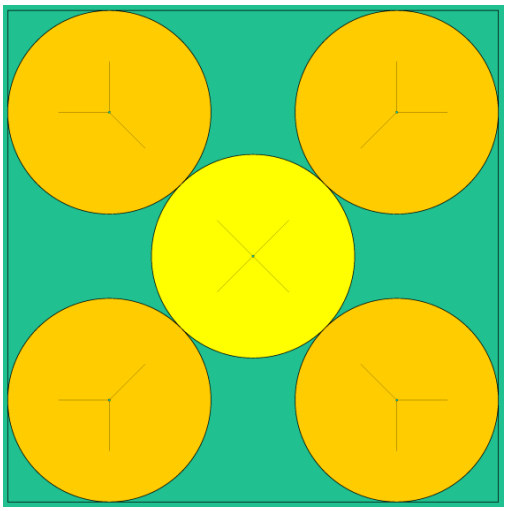
В следующей таблице приведены результаты вычислений. Все значения округлены до 10^{-7}. Полученное значение считалось рекордным, если разница мирового рекорда и этого значения была меньше либо равна 10^{-5}.

Количество кругов	Количество запусков	Количество рекордных значений	Процент рекордных значений	Лучшее значение	Мировой рекорд
5	5000	0	0%	0.2070484	0. 2071067
10	5000	2	0.04%	0.1481970	0.1482043
15	5000	304	6.08%	0.1271605	0.1271665
20	5000	292	5.84%	0. 1113750	0.1113823
25	5000	178	3.56%	0.0999949	0.1
30	5000	2	0.04%	0. 0916613	0.0916710

Как можно заметить, при малом количестве кругов рекордных значений практически нет. Скорее всего, это связано с тем, что при такой размерности критерии остановки алгоритма оказываются слишком грубыми. По получаемому изображению, например, для 5 кругов, видно, что структура составляется верно, но алгоритм не может найти более точно локальный минимум.



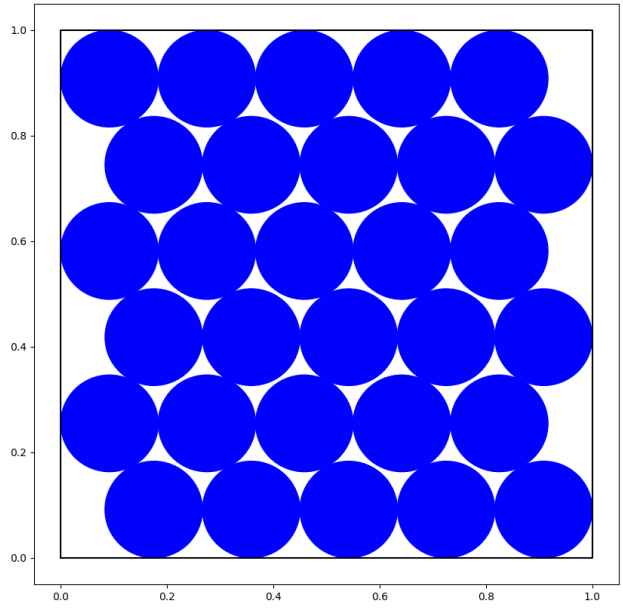
Результат работы программы



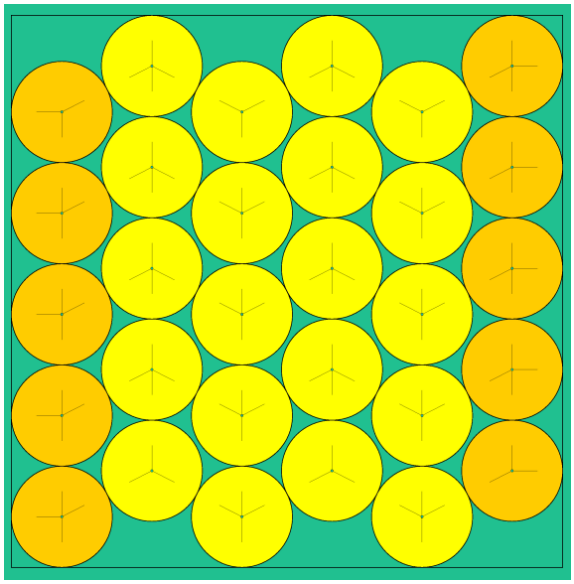
Мировой рекорд

С ростом количества кругов сильно растет влияние фактора случайности генерации центров, поэтому процент рекордных значений уменьшается. При этом программа не редко находит нужную структуру размещения, но ей не удается получить лучшее значение локального минимума.

Рассмотрим пример при 30 кругах:

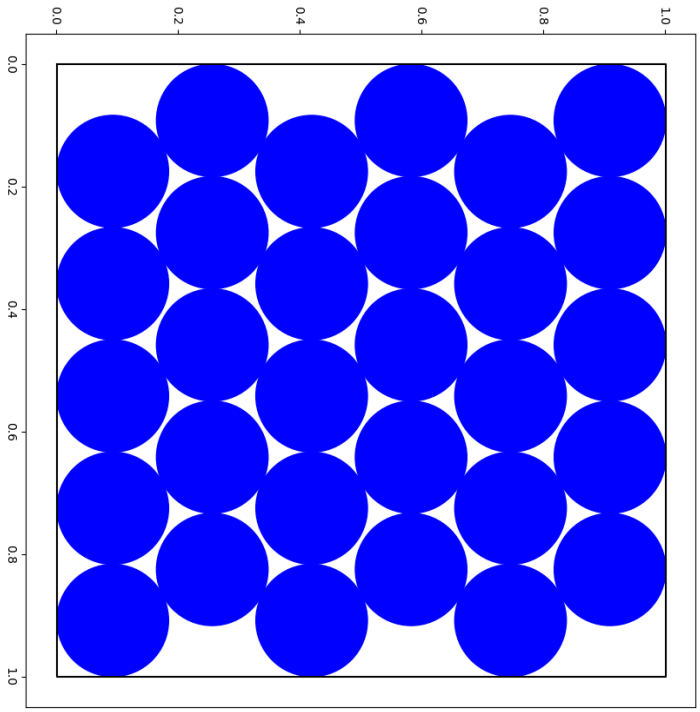


Результат работы программы



Мировой рекорд

Если полученный рисунок повернуть по часовой стрелке, то увидим, что структура полностью совпадает с рекордной:



Заключение

Как показали исследования, при выбранных параметрах алгоритма и критериях сравнения с рекордом количество рекордных значений быстро уменьшается с ростом количества кругов. Чтобы улучшить результаты нужно более детально подобрать значение patience для каждого набора кругов, а также изменять множитель learning rate для более тонкой настройки.

Из приведенных рисунков видно, что во многих случаях программа правильно находит структуру оптимальной упаковки, но не может вычислить значение с высокой точностью.

Развитие программы может быть продолжено добавлением возможности изменения контейнера на другую фигуру, а также смены помещаемых объектов (например, на правильные многоугольники).

Список литературы

1. Задачи упаковки. [Электронный русрс]. Режим доступа: https://mathworld.wolfram.com/CirclePacking.html
2. Обучение нейронной сети. [Электронный русрс]. Режим доступа: https://yandex.ru/turbo/pythonist.ru/s/glubokoe-obuchenie-i-nejronnye-seti-s-python-i-pytorch-chast-iv-obuchenie-nejronnoj-seti/
3. Информация о мировых рекордах упаковки кругов в квадрат. [Электронный русрс]. Режим доступа: http://hydra.nat.uni-magdeburg.de/packing/csq/csq.html
4. Фреймворк PyTorch. [Электронный русрс]. Режим доступа: https://habr.com/ru/post/334380/
5. Градиентный спуск. [Электронный русрс]. Режим доступа: https://sysblok.ru/knowhow/razbiraem-nejroseti-po-chastjam-kak-rabotaet-gradientnyj-spusk/