Report (Borgalinov Timur)

Simulated Annealing and Sampling

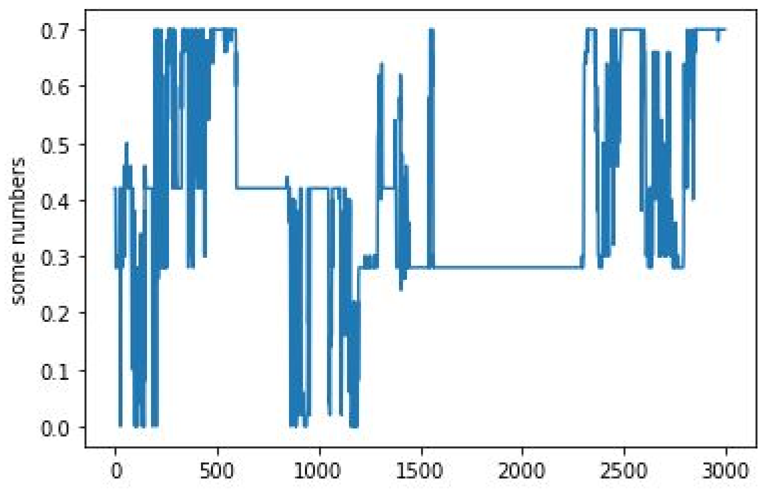
All sources can be found in github repository

1) In the first task i created fully connected NN with 3 layers, with 10,6 and 3 neurons respectively. After optimizing it with gradient i have these results

```
(100, 4) (50, 4) (100,) (50,)
Test Accuracy: 0.860
```

Where first two tuples are the shapes of train and test data, first two for X and second two for Y respectively
Sometimes after some fittings the result was 100% accuracy.

Then i tried to use Simulated annealing instead of gradient based optimizers
The function that made transition between two states were based on normal distribution, where past state were in role of Mu for new state.
Here we can see the accuracy graph, after 3000 iterations.



Also i tried to change different parameters such as T, change size of test and train sets, change state transition function, make more iterations and others. The result was different, but at the graph above you can see the latest one.
This is my Probability function, which is being used to calculate alpha coefficient

```
def p(energy_of_x, T):
  return np.exp(-1 * energy_of_x/T)
```

Here you can see the creation of SAmodel itself

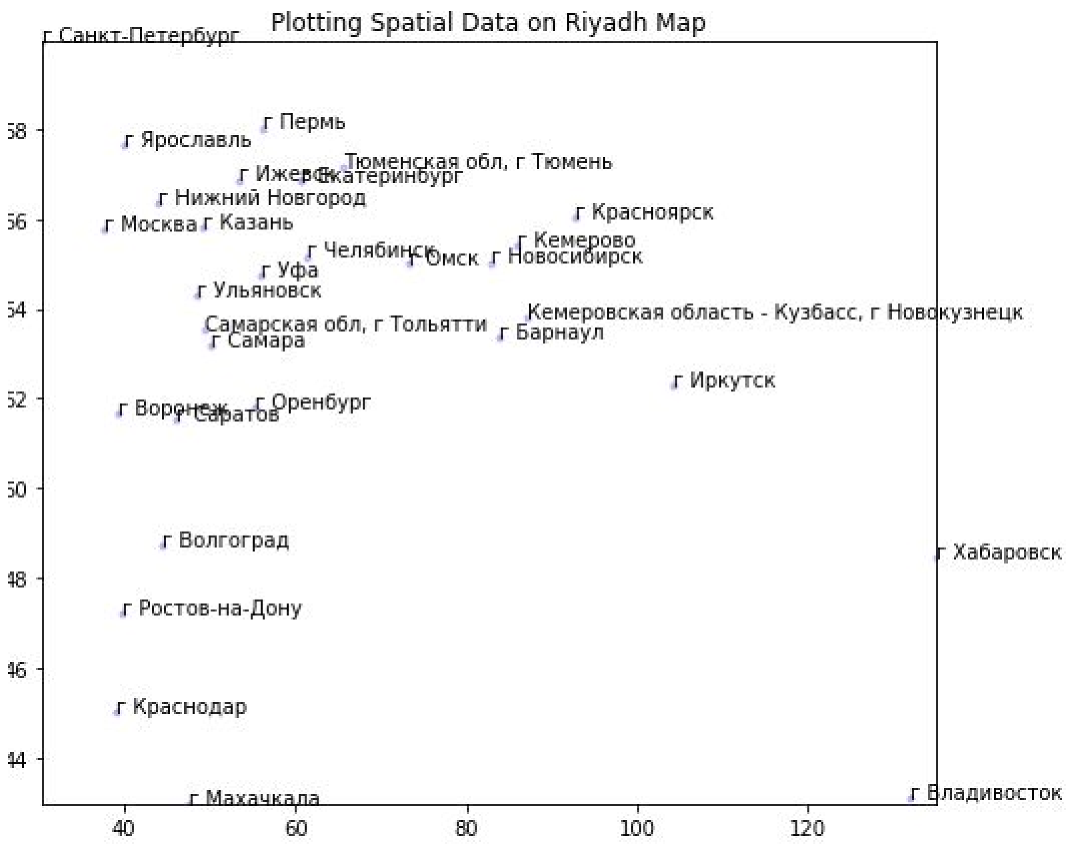
```
samodel = Sequential()
samodel.add(Dense(10, activation='relu', input_shape=(n_features,)))
samodel.add(Dense(6, activation='relu'))
samodel.add(Dense(3, activation='softmax'))
```

This is my transition function

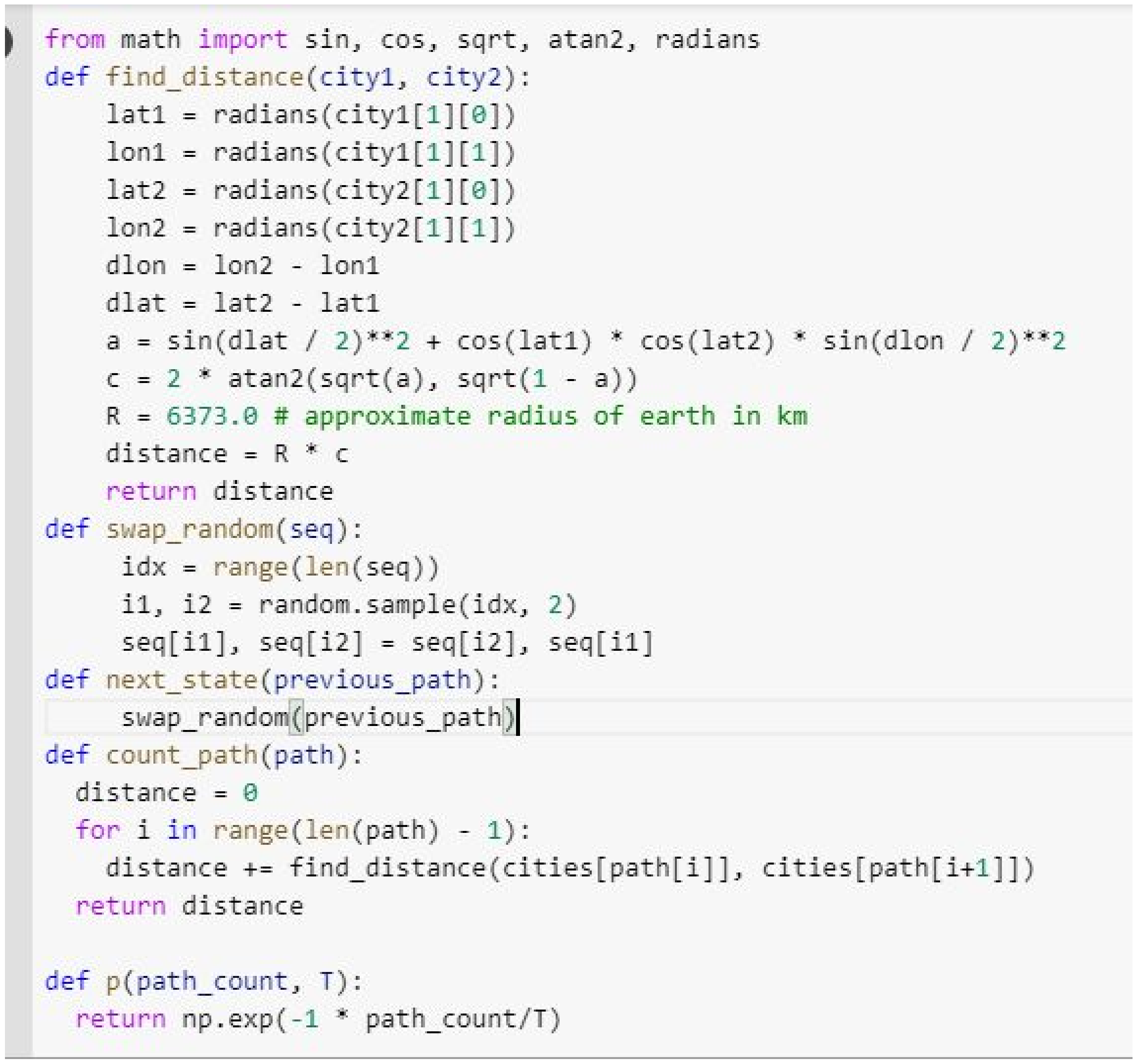
```
def next_state(previous_state):
  return np.random.normal(previous_state, std)
```

2) At the second task firstly i changed type of population column in dataset, it was object to float. It was necessary for me to sort values by population to know 30 most popular cities. I couldn’t create beautiful animation of algorithm working
3) They are [('г Москва', (55.7540471, 37.620405)), ('г Санкт-Петербург', (59.9391313, 30.315900399999997)), ('г Новосибирск', (55.02819100000001, 82.9211489)), ('г Екатеринбург', (56.8385216, 60.605491099999995)), ('г Нижний Новгород', (56.3240627, 44.00539129999999)), ('г Казань', (55.7943584, 49.1114975)), ('г Самара', (53.195030599999996, 50.1069518)), ('г Омск', (54.9848566, 73.3674517)), ('г Челябинск', (55.16026239999999, 61.400807799999995)), ('г Ростов-на-Дону', (47.2224566, 39.718803)), ('г Уфа', (54.734943999999999, 55.9578468)), ('г Волгоград', (48.7070042, 44.5170339)), ('г Пермь', (58.0102583, 56.2342034)), ('г Красноярск', (56.00938789999999, 92.85248059999999)), ('г Воронеж', (51.659333200000006, 39.1969229)), ('г Саратов', (51.5303047, 45.9529348)), ('г Краснодар', (45.0401604, 38.975964700000006)), ('Самарская обл, г Тольятти', (53.5205348, 49.3894028)), ('г Барнаул', (53.3479968, 83.7798064)), ('г Ижевск', (56.852738, 53.21148960000001)), ('г Ульяновск', (54.307941500000005, 48.3748487)), ('г Владивосток', (43.116490399999996, 131.8823937)), ('г Ярославль', (57.62154770000001, 39.897741100000005)), ('г Иркутск', (52.2864036, 104.28074659999999)), ('Тюменская обл, г Тюмень', (57.1529744, 65.5344099)), ('г Махачкала', (42.9849159, 47.504718100000005)), ('г Хабаровск', (48.4647258, 135.05989419999997)), ('г Оренбург', (51.7875092, 55.1018828)), ('Кемеровская область - Кузбасс, г Новокузнецк', (53.794315000000005, 87.2142745)), ('г Кемерово', (55.391065100000006, 86.0467781))]

   With their coordinates respectively, then i plotted out them



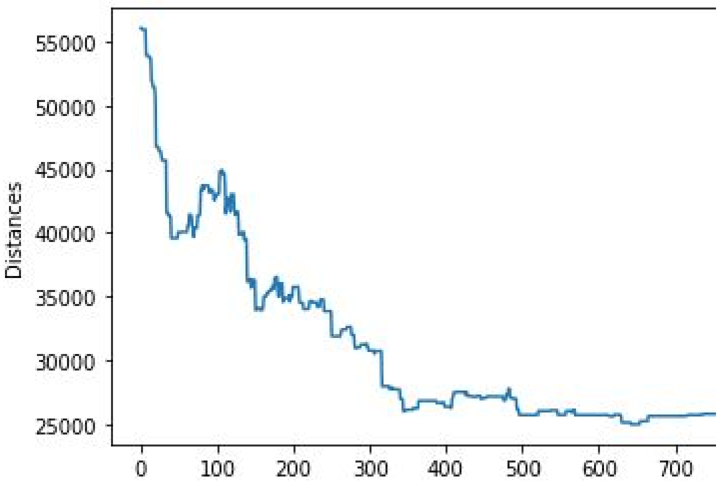
Here you can see my transition function :



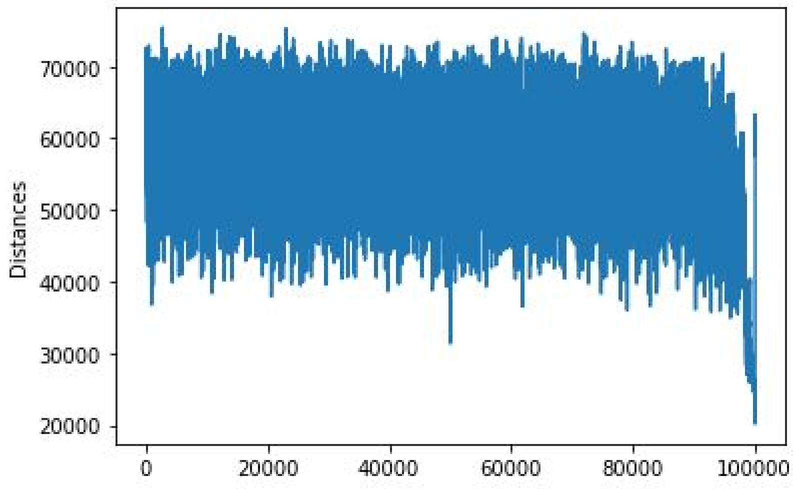
```python
from math import sin, cos, sqrt, atan2, radians
def find_distance(city1, city2):
    lat1 = radians(city1[1][0])
    lon1 = radians(city1[1][1])
    lat2 = radians(city2[1][0])
    lon2 = radians(city2[1][1])
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
    R = 6373.0 # approximate radius of earth in km
    distance = R * c
    return distance
def swap_random(seq):
     idx = range(len(seq))
     i1, i2 = random.sample(idx, 2)
     seq[i1], seq[i2] = seq[i2], seq[i1]
def next_state(previous_path):
     swap_random(previous_path)
def count_path(path):
  distance = 0
  for i in range(len(path) - 1):
    distance += find_distance(cities[path[i]], cities[path[i+1]])
  return distance

def p(path_count, T):
  return np.exp(-1 * path_count/T)
```

Where
 find_distance is being used to find approximate distance between two cities using their longitude and latitude
swap_random - swaps 2 cities in a path
count_path - counts distance for the path

Here you can see the result after 700 epochs



Here you can see the results after 100000 epochs


Distances