Link: www.github.com/HidanCloud/pmldl_assignment1_SA

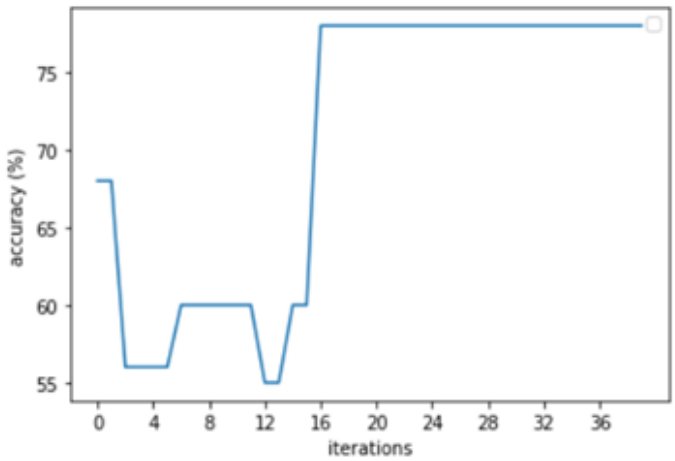
Simulated Annealing: Task 1.

The first task is implemented with use of *Keras* library. The neural network has one hidden layer and in total the following architecture was used:

```
model.add(Dense(units=first_layer_units, input_dim=input_size, activation='relu', name="first"))
model.add(Dense(units=second_layer_units, input_dim=first_layer_units, activation='sigmoid', name="second"))
model.add(Dense(units=3, activation='softmax', name="last"))
```

I have tried different number of nodes in the hidden layer, but the best results got with 8 of them:

```
final temperature: 0.0016080010389340948, final loss: 0.3711, final accuracy: 77.78%
```



In fact, the distribution of the results has big std, in average SA provides by weights with losses from 65% to 80%. And the model is much better with use of one-hot-encoding part for classes of Iris dataset. That is why there are 3 nodes at the output, one for each class.

The general procedure is like it is described in the task: firstly random weights were generated (with mean = 0 and std 1) and then on each iteration random vector (each of it's component i.i.d. and comes from Normal distribution with mean 0 and std 1) is added to the weights which are also represented as a vector.

```
for i in range(len(model.layers)):
  rand_step = list()
  weights = np.array(model.layers[i].get_weights()[0])
  bias = np.array(model.layers[i].get_weights()[1])
  rand_step = [np.random.randn(*weights.shape), np.random.randn(*bias.shape)]
  model_copy.layers[i].set_weights([weights + rand_step[0], bias + rand_step[1]])
```

After that according to the SA description from the task I used p*(x) as an energy estimation, but a bit optimized (instead of division I am subtracting the powers of exponent):

```
def P_star(new_loss, old_loss, T):
  ### energy function that is p*(x) in the task
  return math.exp((old_loss - new_loss)/T)
```

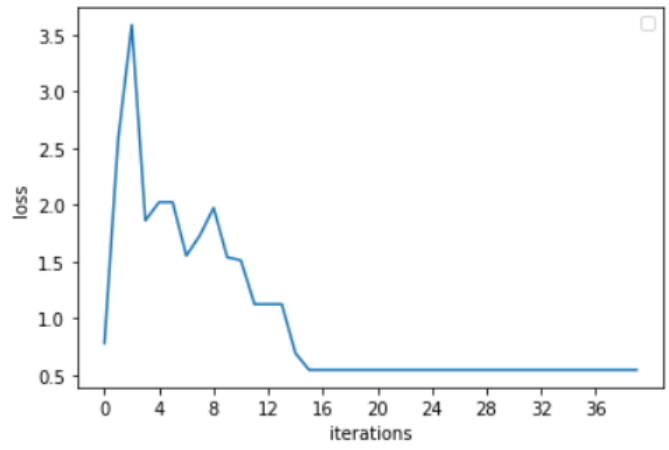
Then generated from uniform distribution [0;1) u determines, if new weights will be accepted and step will be done or not: if u < P_star then proceed with new weights, else return to the weights before the step. At the end of each iteration temperature is multiplies by the coefficient which corresponds to it's range (I used two different numbers there, if T > 0.08, then const = 0.8, else const = 0.85).

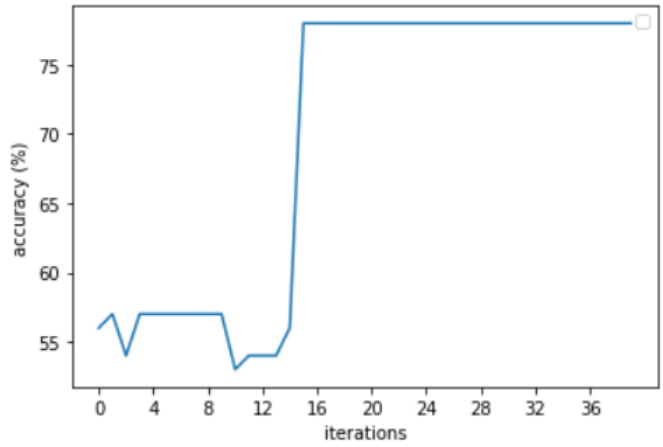
I'm using both iterations and T_cold stopping conditions: either temperature is cold or maximum number of iterations is reached implies stop of the SA algo.

Results:

As it is already mentioned results are about 70% in average, almost every third try gives about 80% performance by 40 iterations. That is good enough for this small neural network, but also nice for SA to find such optimal weights in a space of big dimension (67-d).

```
final temperature: 0.0016080010389340948, final loss: 0.6251, final accuracy: 75.56%
```





Task 2

In the second task I used database of Russian cities from the task, it contains cities, population, geographic coordinates and some odd information. I extracted top30 cities with the biggest population with their coordinates from this file, and then used *geopy* library for feature that computes distance between cities given their coordinates, because simple l2 metrics were not accurate enough.

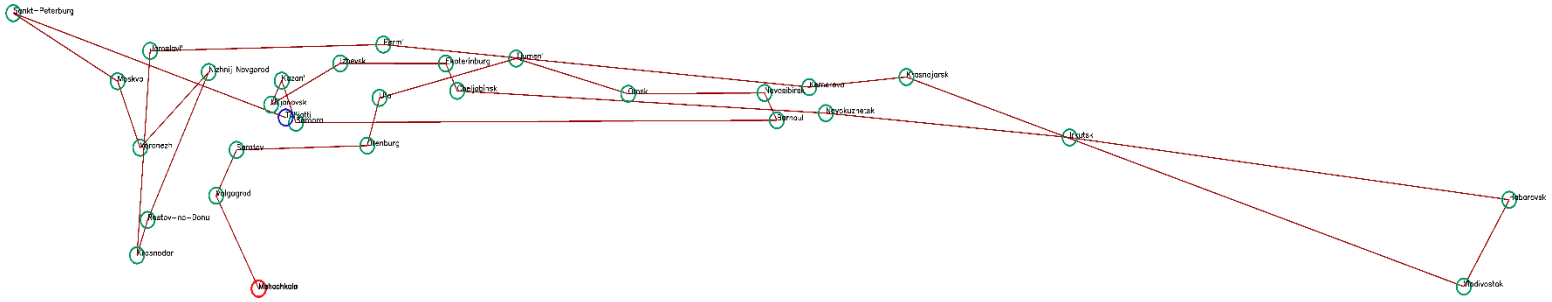
So, the task is to minimize length of the path through all of the cities with use of SA. For that I did everything described in the task, in particular:

Initialized random path through these 30 cities

For each iteration tried to swap two random cities in a route, and then ***u*** from uniform distribution [0; 1] determines if new route will be excepted with use of the same *P_star* from the previous task (but it works now with the length of the route in thousands of km instead of just loss).

Results

At the end of minimizing length of the route by SA the typical output is the following:



And the length is reduced from the initial random ~ 60 000 km to ~ 25 000 km:

```
final temperature: 2.909796624575626e-09
final path length: 25539 km
The first city is Тольятти
next city number 2 is Санкт-Петербург, it is 2217.8 km away
next city number 3 is Москва, it is 798.4 km away
next city number 4 is Воронеж, it is 1075.0 km away
next city number 5 is Нижний Новгород, it is 402.9 km away
next city number 6 is Ростов-на-Дону, it is 494.9 km away
next city number 7 is Краснодар, it is 1303.7 km away
next city number 8 is Ярославль, it is 1157.2 km away
next city number 9 is Пермь, it is 1864.1 km away
next city number 10 is Кемерово, it is 2797.1 km away
next city number 11 is Красноярск, it is 2208.3 km away
next city number 12 is Владивосток, it is 3526.9 km away
next city number 13 is Хабаровск, it is 2951.3 km away
next city number 14 is Иркутск, it is 2289.8 km away
next city number 15 is Новокузнецк, it is 3335.3 km away
next city number 16 is Челябинск, it is 2803.3 km away
next city number 17 is Екатеринбург, it is 1711.5 km away
next city number 18 is Ижевск, it is 544.1 km away
next city number 19 is Ульяновск, it is 819.9 km away
next city number 20 is Казань, it is 279.6 km away
next city number 21 is Самара, it is 168.5 km away
next city number 22 is Барнаул, it is 2234.7 km away
next city number 23 is Новосибирск, it is 2135.8 km away
next city number 24 is Омск, it is 703.3 km away
next city number 25 is Тюмень, it is 1104.6 km away
next city number 26 is Уфа, it is 1115.5 km away
next city number 27 is Оренбург, it is 900.6 km away
next city number 28 is Саратов, it is 757.7 km away
next city number 29 is Волгоград, it is 828.1 km away
next city number 30 is Махачкала, it is 957.2 km away
```