project 1:

Hopfield networks for problem solving

solution(s) due:

December 19, 2019 at **12:00** via email to **bauckhag@bit.uni-bonn.de**

problem specification:

task 2.1: the k-rooks problem
In the lecture, we introduced the k-rooks problem and saw how to solve it using a Hopfield network. In particular, we considered the special case where $k = 4$

a) use what we discussed in the lecture to implement a Hopfield net (with asynchronous updates) that solves the $k = 4$ rooks problem

b) now implement a Hopfield net that solves the $k = 8$ rooks problem; to this end, you need to redefine or recompute the network parameters $\boldsymbol{W}_r$, $\boldsymbol{W}_c$, θ_r, and θ_c; the team that comes up with the "most elegant" numpy code to set these parameters will receive a honorary mention

task 2.2: faster convergence to a stable state
Recall that the state of a Hopfield network of n neurons is given by a vector $\boldsymbol{s} \in \{-1, +1\}^n$. Hence, there are a total of 2^n possible states the network can be in; if n is large it is practically impossible to test all these states to figure out which one(s) have the lowest energy value $H(\boldsymbol{s})$.

Also recall that asynchronous updates of a Hopfield network happen in a random manner. That is, in each iteration t, a neuron s_u is chosen at random and only this neuron updates its state. While this guarantees that the net converges to a (local) energy minimum, the randomness of this process typically causes very slow convergence rates.

Can you think of a non-random (i.e. deterministic) algorithm for choosing the neuron that “aught” to be updated in iteration t?

Hint: a good update should lower the networks energy as much as possible and it is OK if your search for such an update introduces additional computations . . .

Implement you ideas and test them on the k-rooks problem considered in the previous task.

Also, think about why deterministic (energy minimizing) updates are dangerous w.r.t. overall performance even though the lead to faster convergence.

task 2.3: binary clustering of images

A general rule of thumb in data science is the following: if we do not know how to handle a given data set, we can always resort to modeling the data in terms of a graph and analyze / work with that graph. In this task, we put this idea into practice to group a set of images into two clusters.

Consider the first 100 tiny face images from the last project, i.e. run this piece of code

```
X = np.load('faceMatrix.npy').astype('float')
X = X[:,:100]
```

Given this data matrix $\boldsymbol{X}$, compute a Euclidean distance matrix $\boldsymbol{D}^{n\times n}$ where, in our case $n = 100$ and

$$d_{ij} = \left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|^2.$$

If you do not know how to do this efficiently, you might want to read this

> C. Bauckhage, “NumPy / SciPy Recipes for Data Science: Squared Euclidean Distance Matrices”, technical report, 2014
> (available at researchgate.net)

Given $\boldsymbol{D}$, determine the mean (or maybe the median) distance $\hat{d}$ between the images and then compute an adjacency matrix $\boldsymbol{A}^{n\times n}$ where

$$a_{ij} = \begin{cases} 1 & \text{if } d_{ij} \neq 0 \wedge d_{ij} \leq \hat{d} \\ 0 & \text{otherwise} \end{cases}$$

Given $\boldsymbol{A}$, compute the corresponding $n \times n$ Laplacian matrix $\boldsymbol{L}$ and the degree vector $\boldsymbol{d} = \operatorname{diag}\left[\mathrm{L}\right]$.

Note that a so called *balanced graph cut* (that chops a connected graph into two disconnected components of about equal size) can be computed by solving the following bipolar optimization problem

$$\boldsymbol{s}^* = \underset{\boldsymbol{s}\in\{-1,+1\}^n}{\operatorname{argmin}} \; \boldsymbol{s}^\intercal \left(\boldsymbol{L} + \frac{2}{\boldsymbol{1}^\intercal \boldsymbol{d}} \boldsymbol{d}\boldsymbol{d}^\intercal\right) \boldsymbol{s} \qquad (\star)$$

This looks like a problem, a Hopfield network can handle. So, go ahead and implement a Hopfield network that minimizes $(\star)$.

Note: the matrix that occurs in $(\star)$ is not hollow. Therefore, you first need to rewrite $(\star)$ in terms of a hollow weight matrix and then use your result to implement a corresponding Hopfield net.

Once a (local) minimizer $\boldsymbol{s}^*$ has been found, assign data points $\boldsymbol{x}_i$ for which $s_i = +1$ to the first cluster and those for which $s_i = -1$ to the second cluster. Visualize some of the images in both cluster to see if the clustering makes sense.

task 1.5: finding maximally different images
Once again consider the matrix $\boldsymbol{X}$ of 100 tiny images from the previous task and recall that, in the last project, we were concerned with the problem of finding k data points that are as far apart as possible. Back then, we formalized this problem as follows

$$\begin{aligned} S^* = \underset{S \subset X}{\operatorname{argmax}} & \sum_{\boldsymbol{x}_i \in S} \sum_{\boldsymbol{x}_j \in S} \left\| \boldsymbol{x}_i - \boldsymbol{x}_j \right\|^2 \\ \text{s.t.} \quad & |S| = k \end{aligned}$$

Working with the distance matrix $\boldsymbol{D}$ from the previous task and introducing a binary indicator vector $\boldsymbol{z} \in \{0,1\}^n$, this problem can also be cast as

$$\begin{aligned} \boldsymbol{z}^* = \underset{\boldsymbol{z} \in \{0,1\}^n}{\operatorname{argmax}} & \ \boldsymbol{z}^\intercal \boldsymbol{D} \boldsymbol{z} \\ \text{s.t.} \quad & \boldsymbol{1}^\intercal \boldsymbol{z} = k \end{aligned} \tag{$\star\star$}$$

This, too, looks like a problem a Hopfield network could handle. So, go ahead and implement a Hopfield network that solves $(\star\star)$.

Note: $(\star\star)$ is written in terms of binary vectors $\boldsymbol{z}$. However, for the Hopfield networks we are interested in, we require problem formulations in terms of bipolar vectors $\boldsymbol{s}$. That is, you'll need to rewrite the problem in $(\star\star)$ as a QUBO that *minimizes* over bipolar vectors (and make sure that the resulting weight matrix is hollow) before you implement the Hopfield net. (Indeed, in our last exercise session, some of you have already shown that this possible . . . all you need to do is to be be very careful and diligent with your algebraic manipulations).

Run your Hopfield net and visualize the extreme images it determines for several choices of k, say $k \in \{5, 10, 25\}$.

bonus task: finding k nearest neighbors
Since some of you have already solved the previous task, here is an idea for what to do if you are bored:

Run this piece of code

```
X = np.load('faceMatrix.npy').astype('float')
x = X[:,-1]
X = X[:,:100]
```

to obtain a data vector $\boldsymbol{x}$ and a data matrix $\boldsymbol{X}$. Express the problem of finding the $k > 1$ nearest neighbors of $\boldsymbol{x}$ in $\boldsymbol{X}$ as a bipolar QUBO and run a Hopfield network to solve it.

Hint: the problem of finding the single nearest neighbor of $\boldsymbol{x}$ in $\boldsymbol{X}$ can be expressed as binary optimization problem involving standard basis vectors:

$$\boldsymbol{e}_j = \underset{\boldsymbol{e}_i \in \mathbb{R}^n}{\operatorname{argmin}} \left\| \boldsymbol{x} - \boldsymbol{X}\boldsymbol{e}_i \right\|^2$$

Once this is solve, the nearest neighbor is $\boldsymbol{x}_j = \boldsymbol{X}\boldsymbol{e}_j$.