ML2R Coding Nuggets
Hopfield Nets for Max-Sum Diversification

Christian Bauckhage*
Machine Learning Rhine-Ruhr
Fraunhofer IAIS
St. Augustin, Germany

Fabrice Beaumont
Computer Science
University of Bonn
Bonn, Germany

Sebastian Müller
Machine Learning Rhine-Ruhr
University of Bonn
Bonn, Germany

ABSTRACT

We demonstrate that Hopfield networks can tackle the max-sum diversification problem. To this end, we express max-sum diversification as a quadratic unconstrained binary optimization problem which can be cast as a Hopfield energy minimization problem. Since max-sum diversification is an NP-hard subset selection problem, we cannot guarantee that Hopfield nets will discover an optimal solution. Nevertheless, our simple *NumPy* implementation consistently produces good results.

1 INTRODUCTION

The problem of max-sum diversification has a venerable history in location- or portfolio optimization [8, 12, 16] but also occurs in the context of information retrieval, recommender systems, or data clustering [1, 9, 17, 19, 25]. In all these setting, we may be interested in identifying a small but diverse subset of a given set of objects. If we assume that two objects are diverse if they are far apart, this subset selection problem can be formalized as follows:

Given some set $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ and an appropriate distance measure $d(\cdot, \cdot)$, determine a subset $\mathcal{S} \subset \mathcal{X}$ of size $|\mathcal{S}| = k < n$ of maximum dispersion. In other words, solve

$$\begin{aligned} \mathcal{S}^* = \underset{\mathcal{S} \subset \mathcal{X}}{\operatorname{argmax}} & \sum_{\boldsymbol{x}_i \in \mathcal{S}} \sum_{\boldsymbol{x}_j \in \mathcal{S}} d(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ \text{s.t.} \quad & |\mathcal{S}| = k. \end{aligned} \tag{1}$$

If we gather all pairwise distances between the elements of $\mathcal{X}$ in an $n \times n$ distance matrix $\boldsymbol{D}$ and introduce a binary indicator vector $\boldsymbol{z} \in \{0, 1\}^n$ where

$$z_i = \begin{cases} 1 & \text{if } \boldsymbol{x}_i \in \mathcal{S} \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

we can write the problem in (1) much more succinctly, namely

$$\begin{aligned} \boldsymbol{z}^* = \underset{\boldsymbol{z} \in \{0,1\}^n}{\operatorname{argmax}} & \ \boldsymbol{z}^\mathsf{T} \boldsymbol{D} \boldsymbol{z} \\ \text{s.t.} \quad & \mathbf{1}^\mathsf{T} \boldsymbol{z} = k \end{aligned} \tag{3}$$

where $\mathbf{1}$ denotes the n-dimensional vector of all ones.

This formulation now immediately reveals that max-sum diversification is an NP-hard integer programming problem. Solution strategies therefore resort to greedy heuristics which achieve $O(nk)$ efficiency but lack optimality guarantees [19]. More sophisticated heuristics provide optimality guarantees of $O(1 - 1/k)$ and require $O(nk^2 \log k)$ runtime [7] or achieve $O(1 - 1/\epsilon)$ at $O(n/\epsilon \log k)$ runtime where the constant $\epsilon \ll 1$ [1].

* 0000-0001-6615-2128

However, the algebraic structure of the problem in (3) suggests that it could also be written as a **quadratic unconstrained binary optimization (QUBO) problem**. This, in turn, could allow us to cast max-sum diversification as an energy minimization problem that can be solved by Hopfield networks. Indeed, both these ideas are feasible and putting them into practice is what this note is all about.

In section 2, we first recall the basic theory behind Hopfield networks and then explain how to turn the max-sum diversification problem in (3) into an energy minimization problem a Hopfield net can solve.

As a use case for our *NumPy*/*SciPy* implementations of Hopfield nets for max-sum diversification in section 3, we will consider the $n = 400$ tiny face images in Fig. 1 and attempt to identify diverse subsets of $k \in \{9, 16, 25, 36\}$ faces.

Alas, we will not discuss *Python* code for image I/O but only those code snippets required for corresponding Hopfield network computations. Readers who would like to experiment with those should be familiar with *NumPy*/*SciPy* [15] and only need to

```
import numpy as np
import numpy.random as rnd
import scipy.spatial as spt
```

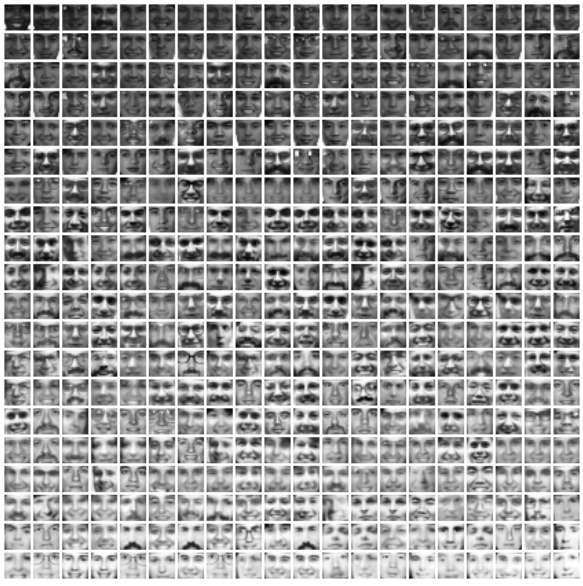

Figure 1: 400 tiny images sampled from the CBCL face database (and sorted from lowest to highest total brightness).

2 THEORY

A Hopfield network is a recurrent neural net of n interconnected neurons $s_1, s_2, \ldots, s_n$ each of which is a bipolar threshold unit

$$s_i = \begin{cases} +1 & \text{if } \boldsymbol{w}_i^\mathsf{T}\boldsymbol{s} - \theta_i \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

$$= \text{sign}\big(\boldsymbol{w}_i^\mathsf{T}\boldsymbol{s} - \theta_i\big) \tag{4}$$

where the bipolar vector $\boldsymbol{s} = [s_1, \ldots, s_n]^\mathsf{T} \in \{-1, +1\}^n$ denotes the overall state of the network.

If the weight matrix $\boldsymbol{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_n]$ of a Hopfield net is symmetric and hollow (i.e. $\boldsymbol{W} = \boldsymbol{W}^\mathsf{T}$ and $\text{diag}[\boldsymbol{W}] = \boldsymbol{0}$) and *if* its neurons s_i update asynchronously (i.e. compute (4) one at a time), *then* these updates will never increase the energy

$$E(\boldsymbol{s}) = -\tfrac{1}{2}\,\boldsymbol{s}^\mathsf{T}\boldsymbol{W}\boldsymbol{s} + \boldsymbol{\theta}^\mathsf{T}\boldsymbol{s} \tag{5}$$

of the network. As there are "only" 2^n states the network can be in, it will converge to a (local) energy minimum within finitely many steps [10].

This property of Hopfield nets can be used for problem solving if the problem at hand can be written as a QUBO. The idea simply is to run a Hopfield net whose weight- and bias parameters $\boldsymbol{W}$ and $\boldsymbol{\theta}$ are designed such that minimum energy state(s)

$$\boldsymbol{s}^* = \underset{\boldsymbol{s}\in\{\pm 1\}^n}{\text{argmin}} -\tfrac{1}{2}\,\boldsymbol{s}^\mathsf{T}\boldsymbol{W}\boldsymbol{s} + \boldsymbol{\theta}^\mathsf{T}\boldsymbol{s} \tag{6}$$

of the network encode the sought after solution(s).

2.1 Hopfield Nets for Max-Sum Diversification

The major challenge when using Hopfield nets for problem solving is to (re)write the given problem as a QUBO. Should this be possible, it is typically easy to implement a corresponding Hopfield net. We will see this in the following where we first turn the equality constrained integer programming problem in (3) into a QUBO for which we then derive an energy function that can be minimized by a Hopfield network.

In order to derive a QUBO representation of the max-sum diversification problem in (3) which, once again, reads

$$\begin{aligned} \underset{\boldsymbol{z}\in\{0,1\}^n}{\text{argmax}} \quad & \boldsymbol{z}^\mathsf{T}\boldsymbol{D}\boldsymbol{z} \\ \text{s.t.} \quad & \boldsymbol{z}^\mathsf{T}\boldsymbol{1} = k \end{aligned} \tag{7}$$

we first of all note that its objective function $\boldsymbol{z}^\mathsf{T}\boldsymbol{D}\boldsymbol{z}$ is a quadratic form in the decision variable $\boldsymbol{z}$. We can therefore turn the maximization problem in (7) into an equivalent minimization problem, namely

$$\begin{aligned} \underset{\boldsymbol{z}\in\{0,1\}^n}{\text{argmin}} \quad & -\boldsymbol{z}^\mathsf{T}\boldsymbol{D}\boldsymbol{z} \\ \text{s.t.} \quad & \boldsymbol{z}^\mathsf{T}\boldsymbol{1} = k \end{aligned} \tag{8}$$

Next, we consider the single equality constraint of this problem and note the implication $\boldsymbol{z}^\mathsf{T}\boldsymbol{1} = k \Leftrightarrow \big(\boldsymbol{z}^\mathsf{T}\boldsymbol{1} - k\big)^2 = 0$. This allows us to rewrite (8) as

$$\begin{aligned} \underset{\boldsymbol{z}\in\{0,1\}^n}{\text{argmin}} \quad & -\boldsymbol{z}^\mathsf{T}\boldsymbol{D}\boldsymbol{z} \\ \text{s.t.} \quad & \big(\boldsymbol{z}^\mathsf{T}\boldsymbol{1} - k\big)^2 = 0 \end{aligned} \tag{9}$$

If we then introduce a Lagrange multiplier λ which we will henceforth treat as a parameter that has to be chosen manually, we can eliminate the equality constraint altogether and also express our problem as

$$\underset{\boldsymbol{z}\in\{0,1\}^n}{\text{argmin}} \quad -\boldsymbol{z}^\mathsf{T}\boldsymbol{D}\boldsymbol{z} + \lambda \cdot \big(\boldsymbol{z}^\mathsf{T}\boldsymbol{1} - k\big)^2 \tag{10}$$

Next, we expand the squared term in (10) and find the following

$$\underset{\boldsymbol{z}\in\{0,1\}^n}{\text{argmin}} \quad -\boldsymbol{z}^\mathsf{T}\boldsymbol{D}\boldsymbol{z} + \lambda\,\boldsymbol{z}^\mathsf{T}\boldsymbol{1}\boldsymbol{1}^\mathsf{T}\boldsymbol{z} - 2\,\lambda\,k\,\boldsymbol{1}^\mathsf{T}\boldsymbol{z} + const \tag{11}$$

where *const* indicates terms independent of $\boldsymbol{z}$. Since these do not impact the solution of our optimization problem, we may drop them. This, and further clean-up, results in

$$\underset{\boldsymbol{z}\in\{0,1\}^n}{\text{argmin}} \quad \boldsymbol{z}^\mathsf{T}\big[\lambda\,\boldsymbol{1}\boldsymbol{1}^\mathsf{T} - \boldsymbol{D}\big]\,\boldsymbol{z} - 2\,\lambda\,k\,\boldsymbol{1}^\mathsf{T}\boldsymbol{z} \tag{12}$$

Now, we introduce two short-hands, namely a matrix and a vector

$$\boldsymbol{P} = \lambda\,\boldsymbol{1}\boldsymbol{1}^\mathsf{T} - \boldsymbol{D} \tag{13}$$

$$\boldsymbol{p} = -2\,\lambda\,k\,\boldsymbol{1} \tag{14}$$

which finally allows us to express the max-sum diversification problem as

$$\underset{\boldsymbol{z}\in\{0,1\}^n}{\text{argmin}} \quad \boldsymbol{z}^\mathsf{T}\boldsymbol{P}\,\boldsymbol{z} + \boldsymbol{p}^\mathsf{T}\boldsymbol{z} \tag{15}$$

At this point, we have achieved our first major goal and cast max-sum diversification as a QUBO. However, (15) minimizes over *binary* vectors $\boldsymbol{z}$ with entries $z_i \in \{0, 1\}$ while the Hopfield energy minimization problem in (5) minimizes over *bipolar* vectors $\boldsymbol{s}$ with entries $s_i \in \{-1, +1\}$.

In order to reach our overall goal of expressing max-sum diversification as a Hopfield energy minimization problem we therefore need to keep going. Hence, we recall that binary and bipolar vectors are isomorphic in the sense that

$$\boldsymbol{s} = 2\,\boldsymbol{z} - \boldsymbol{1} \;\Leftrightarrow\; \boldsymbol{z} = \tfrac{1}{2}\,[\boldsymbol{s} + \boldsymbol{1}] \tag{16}$$

If we thus plug $\boldsymbol{z} = \frac{1}{2}\,[\boldsymbol{s} + \boldsymbol{1}]$ into (15), we find that the max-sum diversification problem can also be expressed as

$$\underset{\boldsymbol{s}\in\{\pm 1\}^n}{\text{argmin}} \quad \tfrac{1}{4}\,[\boldsymbol{s} + \boldsymbol{1}]^\mathsf{T}\boldsymbol{P}\,[\boldsymbol{s} + \boldsymbol{1}] + \tfrac{1}{2}\,\boldsymbol{p}^\mathsf{T}[\boldsymbol{s} + \boldsymbol{1}] \tag{17}$$

Expanding all the matrix-vector products and inner products in this expression yields

$$\underset{\boldsymbol{s}\in\{\pm 1\}^n}{\text{argmin}} \quad \tfrac{1}{4}\,\boldsymbol{s}^\mathsf{T}\boldsymbol{P}\,\boldsymbol{s} + \tfrac{1}{2}\,\boldsymbol{s}^\mathsf{T}\boldsymbol{P}\,\boldsymbol{1} + \tfrac{1}{2}\,\boldsymbol{s}^\mathsf{T}\boldsymbol{p} + const \tag{18}$$

where *const* now denotes terms independent of $\boldsymbol{s}$. Since these, too, do not impact the sought after solution, we may again drop them. This, and further clean-up, results in

$$\underset{\boldsymbol{s}\in\{\pm 1\}^n}{\text{argmin}} \quad \tfrac{1}{4}\,\boldsymbol{s}^\mathsf{T}\boldsymbol{P}\,\boldsymbol{s} + \tfrac{1}{2}\,\boldsymbol{s}^\mathsf{T}\big[\boldsymbol{P}\,\boldsymbol{1} + \boldsymbol{p}\big] \tag{19}$$

At this point, we observe that the diagonal elements of matrix $\boldsymbol{P} = \lambda\,\boldsymbol{1}\boldsymbol{1}^\mathsf{T} - \boldsymbol{D}$ are all given by $P_{ii} = \lambda$ because, for any distance matrix $\boldsymbol{D}$, we have $D_{ii} = 0$. This allows us to rewrite (19) as

$$\underset{\boldsymbol{s}\in\{\pm 1\}^n}{\text{argmin}} \quad \tfrac{1}{4}\,\boldsymbol{s}^\mathsf{T}\big[\boldsymbol{P} - \lambda\,\boldsymbol{I}\big]\,\boldsymbol{s} + \tfrac{\lambda}{4}\,\boldsymbol{s}^\mathsf{T}\boldsymbol{I}\,\boldsymbol{s} + \tfrac{1}{2}\,\boldsymbol{s}^\mathsf{T}\big[\boldsymbol{P}\,\boldsymbol{1} + \boldsymbol{p}\big] \tag{20}$$

Algorithm 1 Random asynchronous updates of a Hopfield net

initialize $\boldsymbol{s}_0 \in \{\pm 1\}^n$

for $t = 0, \ldots, t_{\max}$ **do**

$u \sim Uniform(\{1, \ldots, n\})$

$(\boldsymbol{s}_{t+1})_u = \text{sign}(\boldsymbol{w}_u^\mathsf{T} \boldsymbol{s}_t - \theta_u)$

where $\boldsymbol{I}$ denotes the $n \times n$ identity matrix. If we next recall a crucial general property of bipolar vectors $\boldsymbol{s}$ with entries $s_i \in \{-1, +1\}$, namely that

$$\boldsymbol{s}^\mathsf{T} \boldsymbol{I}\, \boldsymbol{s} = \sum_{i=1}^{n} s_i^2 = n \tag{21}$$

we realize that the term $\frac{\lambda}{4}\, \boldsymbol{s}^\mathsf{T} \boldsymbol{I}\, \boldsymbol{s} = \frac{\lambda n}{4}$ is yet another constant independent of $\boldsymbol{s}$. But this means that we may cast the max-sum diversification problem as the problem of solving

$$\underset{\boldsymbol{s} \in \{\pm 1\}^n}{\text{argmin}} \quad \boldsymbol{s}^\mathsf{T} \boldsymbol{Q}\, \boldsymbol{s} + \boldsymbol{q}^\mathsf{T} \boldsymbol{s} \tag{22}$$

where we have introduced the short-hands

$$\boldsymbol{Q} = \tfrac{1}{4} \left[\boldsymbol{P} - \lambda\, \boldsymbol{I} \right] \tag{23}$$

$$\boldsymbol{q} = \tfrac{1}{2} \left[\boldsymbol{P}\, \boldsymbol{1} + \boldsymbol{p} \right] \tag{24}$$

We thus have reached another major milestone and expressed max-sum diversification as a QUBO over bipolar vectors $\boldsymbol{s}$. We also emphasize an important characteristic of the result in (22), namely that the coupling matrix $\boldsymbol{Q}$ has a diagonal of all zeros. We have therefore found a formulation of our problem that meets all the requirements for the use of Hopfield networks.

Yet, the sign pattern and scaling of the problem in (22) still deviate from the general form of a Hopfield energy minimization problem in (6). However, we may simply define

$$\boldsymbol{W} = -2\, \boldsymbol{Q} \tag{25}$$

$$\boldsymbol{\theta} = \boldsymbol{q} \tag{26}$$

to finally turn max-sum diversification into an energy minimization problem that reads

$$\underset{\boldsymbol{s} \in \{\pm 1\}^n}{\text{argmin}} \; -\tfrac{1}{2}\, \boldsymbol{s}^\mathsf{T} \boldsymbol{W}\, \boldsymbol{s} + \boldsymbol{\theta}^\mathsf{T} \boldsymbol{s} \tag{27}$$

and can thus be solved by a Hopfield net.

2.2 Asynchronous Updates of Hopfield Nets

To conclude our theoretical discussion, we briefly look at a simple idea for how to run a Hopfield net asynchronously.

This idea is summarized in Alg. 1 and indeed rather trivial. We simply assume that a Hopfield net evolves over a sequence of time steps $t = 0, 1, \ldots, t_{\max}$. At time t, it is in state $\boldsymbol{s}_t$. If neuron u is then randomly chosen to update its activation, its activation at time $t+1$ amounts to

$$(\boldsymbol{s}_{t+1})_u = \text{sign}(\boldsymbol{w}_u^\mathsf{T} \boldsymbol{s}_t - \theta_u) \tag{28}$$

for all other neurons $i \neq u$, we have

$$(\boldsymbol{s}_{t+1})_i = (\boldsymbol{s}_t)_i \tag{29}$$

This way the neurons of the network do indeed update one at a time. Depending on the current global state $\boldsymbol{s}_t$, the update of neuron u in (28) may either cause its activation to switch from ± 1 to ∓ 1 or to remain the same. In case of the latter, we obviously have $\Delta E = E(\boldsymbol{s}_{t+1}) - E(\boldsymbol{s}_t) = 0$. In case of the former, it is a tedious yet straightforward exercise to prove that, if $\boldsymbol{W}$ is symmetric and hollow, $\Delta E = -2 \cdot (\boldsymbol{s}_{t+1})_u \cdot (\boldsymbol{w}_u^\mathsf{T} \boldsymbol{s}_t - \theta_u)$. And, since $(\boldsymbol{s}_{t+1})_u$ has, by definition, the same sign as $(\boldsymbol{w}_u^\mathsf{T} \boldsymbol{s}_t - \theta_u)$ it follows that $\Delta E < 0$.

Hence, if we run Alg. 1 to evolve a Hopfield network over time, we are guaranteed that its energy at the end of this evolution can never be greater than its initial energy. On the contrary, it will almost surely be (much) smaller. With respect to the max-sum diversification problem, this means that the +1-entries of $\boldsymbol{s}_{t_{\max}}$ will index mutually far apart elements of the data set under consideration.

3 PRACTICE

In this section, we discuss how to practically implement Hopfield nets for the max-sum diversification problem.

As an application example, we consider the set $\mathcal{X}$ of $n = 400$ face images in Fig. 1. These were sampled from the CBCL face database [22] which contains intensity images of 19×19 pixels. We may therefore represent the tiny images in Fig. 1 in terms of vectors $\boldsymbol{x}_i \in \mathbb{R}^{361}$ and gather them in a data matrix

$$\boldsymbol{X} = \left[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \right] \tag{30}$$

of size $361 \times n$. This way, our objects of interest are now real valued vectors for which we can consider distance measures such as the squared Euclidean distance.[1] Hence, it is easy to compute an $n \times n$ distance matrix

$$\boldsymbol{D} \quad \text{where} \quad D_{ij} = \left\| \boldsymbol{x}_i - \boldsymbol{x}_j \right\|^2 \tag{31}$$

which we need to define the weights of our Hopfield net.

In what follows, we assume that the images in Fig. 1 have already been read into memory, vectorized, and gathered in a data matrix which is implemented as a 2D *NumPy* array `matX` whose size we determine using

```
m, n = matX.shape
```

Next, we initialize the weight matrix $\boldsymbol{W}$ and bias vector $\boldsymbol{\theta}$ of our Hopfield net. To this end, we use function `hnetInitParameters` in Listing 1 and call, say

```
matW, vecT = hnetInitParameters(vecX, 9)
```

to obtain corresponding *NumPy* arrays `matW` and `vecT`.

The three parameters of `hnetInitParameters` in Listing 1 are the data array `vecX` we assume to be given, an integer `k` which indicates the number of data points we wish to determine, and a number `l` which represent the multiplier λ that first occurred in equation (10). The latter can be set by the user, however, we choose its default value to be `None` and shortly explain why.

In lines 4 and 5 of `hnetInitParameters`, we apply methods provided in *SciPy*'s `spatial` module in order to compute a 2D array `matD` which represents the distance matrix $\boldsymbol{D}$ in (31). For a detailed explanation of the inner working of these *SciPy* methods, we refer to our earlier discussion in [3].

[1] We may, of course, also work with other distances. We consider squared Euclidean distances really just because they are easy to compute.

Listing 1: initializing parameters W and θ of a Hopfield net

```
1  def hnetInitParameters(matX, k, l=None):
2      m, n = matX.shape
3
4      matD = spt.distance.pdist(matX.T, 'sqeuclidean')
5      matD = spt.distance.squareform(matD)
6      matD = matD / np.max(matD)
7
8      mat1 = np.ones((n,n))
9      vec1 = np.ones(n)
10     matI = np.eye(n)
11
12     if l is None:
13         l = 10*n
14
15     matP =  l * mat1 - matD
16     vecP = -l * 2*k * vec1
17
18     matQ = 0.25 * (matP - l * matI)
19     vecQ = 0.50 * (matP @ vec1 + vecP)
20
21     matW = -2 * matQ
22     vecT = vecQ
23
24     return matW, vecT
```

Listing 2: running a Hopfield net

```
1  def signum(x):
2      return np.where(x >= 0, +1, -1)
3
4
5  def hnetRunRnd(vecS, matW, vecT, tmax=100):
6      for u in rnd.randint(0, len(vecS), tmax):
7          vecS[u] = signum(matW[u] @ vecS - vecT[u])
8      return vecS
```

In line 6, we normalize array `matD` such that it largest entry equals 1. From the point of view of the (symmetric) distances matrix $\boldsymbol{D}$ represented by `matD`, this has the effect that its row- and column sums are bounded by

$$\sum_i D_{ij} = \sum_j D_{ij} < n \tag{32}$$

and our motivation behind this normalization will become clear shortly.

The arrays `vec1`, `mat1`, and `matI` that are initialized in lines 8–10 represent the vector $\mathbf{1}$ and the matrices $\mathbf{1}\mathbf{1}^\mathsf{T}$ and $\boldsymbol{I}$, respectively.

Lines 12 and 13 address the crucial open question of how to set the Lagrange multiplier λ which is a parameter of the weights and bias values of our network. To make a long story short, its choice generally has to trade off the two objectives of identifying far apart data points but only k of those. If λ was too small, the the former objective would outweigh the latter and our Hopfield net would likely determine more than k diverse objects which is undesirable. However, the implicit weight of the first objective depends on the sizes of the entries of distance matrix $\boldsymbol{D}$ which are data dependent and thus cannot be known in general. To circumvent this issue, we applied the normalization in line 6 which bounds the implicit weight of the first objective by n, the number of given data. Working with this normalization, $\lambda = 10 \cdot n$ is then generally large enough to enforce the second objective and this is the value we set in line 13.

The arrays computed in lines 15–22 are straightforward *NumPy* implementations of the matrices and vectors we defined in (13), (14), (23), (24), (25), and (26) and thus do not need elaboration[2].

Having initialized the parameters `matW` and `vecT` of our Hopfield net for max-sum diversification, we next need to set its initial state $\boldsymbol{s}$. Here, a random initialization may be a good idea in practice, however, when in doubt, we may simply choose $\boldsymbol{s} = -\mathbf{1}$. To accomplish this in *NumPy*, we use

```
vecS = -np.ones(n)
```

After all these preparatory steps, we are finally good to go and may call

```
vecS = hnetRunRnd(vecS, matW, vecT)
```

to solve our max-sum diversification sorting problem. This applies unction `hnetRunRnd` in Listing 2 which implements the ideas in Alg. 1.

The first three parameters of `hnetRunRnd` are self-explanatory. Parameter `tmax` indicates the number of update iterations our Hopfield should perform.

The `for` loop in line 6 is over an array of size `tmax` whose entries are random integers `u` in the interval $[0, n-1]$ and indicate which neurons should should update their activation. The respective update is computed in line 7 and another straightforward implementation of the underlying mathematics. However, it is worth mentioning that line 7 involves a custom made sign function `signum` which is defined in lines 1 and 2 of Listing 2. We use `signum` instead of the *NumPy* function `np.sign` because the latter implements

$$\text{sign}(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ +1 & \text{if } x > 0 \end{cases}$$

However, for Hopfield nets to work properly, we must insists on

$$\text{sign}(x) = \begin{cases} -1 & \text{if } x < 0 \\ +1 & \text{if } x \geq 0 \end{cases}$$

as computed by `signum`. Finally, `hnetRunRnd` returns a state vector $\boldsymbol{s}$ which represents the activation pattern of the neurons of the network after `tmax` updates.

In order to turn this activation pattern into a selection of (hopefully) diverse images, we have to carry out one more step which we realize as follows

```
mask = np.where(vecS>0, True, False)
matS = matX[:,mask]
```

This will create a 2D array `matS` whose columns represent the images our Hopfield net deemed to be diverse.

The prototypical examples in Fig. 2 indicate the our implementation of max-sum diversification Hopfield nets works appropriately. The shown examples resulted from choosing $k \in \{9, 16, 25, 36\}$ and were found within $t_{\max} = 100$ update iterations of our networks. While there is no guarantee that the energies $E(\boldsymbol{s}_{t_{\max}})$ of the resulting final states are the lowest possible, the selected images appear reasonably diverse to human observers.

[2]Note, however, that our implementation sacrifices efficiency for readability. There are more (memory) efficient ways of implementing matrix $\boldsymbol{W}$ and vector $\boldsymbol{\theta}$ which would make use of *NumPy* features that would render the underlying math less recognizable.

(a) $k = 9$ **(b)** $k = 16$ **(c)** $k = 25$ **(d)** $k = 36$

Figure 2: Diverse subsets of sizes $k \in \{9, 16, 25, 26\}$ of the $n = 400$ face images in Fig. 1 (all sorted from darkest to brightest).

4 CONCLUSION

This note demonstrated that Hopfield networks can tackle the NP-hard max-sum diversification problem. To achieve this, we first expressed max-sum diversification as a QUBO which we then rewrote as a Hopfield energy minimization problem. Our *NumPy* implementations of corresponding Hopfield nets were straightforward translations from this mathematics into code.

To some, it may seem miraculous that such a simple neurocomputing paradigm can tackle such a difficult problem. Others may now wonder if Hopfield nets are a strong contender for max-sum diversification. Since there are highly specialized heuristics for this problem [1, 7], we will not argue that the latter is the case.

However, the fact that Hopfield nets can tackle max-sum diversification is interesting for another reason: If a problem can be written such that it can be solved using Hopfield nets, it can also be solved using adiabatic quantum computing [4, 11] or highly efficient digital annealers [6, 13, 14]. For the max-sum diversification problem, we already demonstrated this in eralier work [5]. This, in turn, is interesting w.r.t. other anlytics methods, too. For instance, there are variants of archetypal analysis or non-negative matrix factorization which require the computation of diverse sets of data points [2, 18, 20, 21, 23, 24]. Running this preparatory step on quantum computers could, in the future, significantly speed up such procedures.

ACKNOWLEDGMENTS

This material was produced within the Competence Center for Machine Learning Rhine-Ruhr (**ML2R**) which is funded by the Federal Ministry of Education and Research of Germany (grant no. 01IS18038C). The authors gratefully acknowledge this support.

REFERENCES

[1] Z. Abassi, V. Mirrokni, and M. Thakur. 2013. Diversity Maximization under Matroid Constraints. In *Proc. KDD*.
[2] C. Bauckhage. 2014. A Purely Geometric Approach to Non-Negative Matrix Factorization. In *Proc. KDML-LWA*.
[3] C. Bauckhage. 2014. NumPy / SciPy Recipes for Data Science: Squared Euclidean Distance Matrices. researchgate.net. https://dx.doi.org/10.13140/2.1.4426.1127.
[4] C. Bauckhage, R. Sanchez, and R. Sifa. 2020. Problem Solving with Hopfield Networks and Adiabatic Quantum Computing. In *Proc. IJCNN*. IEEE.
[5] C. Bauckhage, R. Sifa, and S. Wrobel. 2020. Adiabatic Quantum Computing for Max-Sum Diversication. In *Proc. SDM*. SIAM.
[6] J. Boyd. 2018. Silicon Chip Delivers Quantum Speeds. *IEEE Spectrum* 55, 7 (2018).
[7] A. Cevallos, F. Eisenbrand, and R. Zenklusen. 2016. Max-Sum Diversity Via Convex Programming. In *Int. Symp. on Computational Geometry*, S. Fekete and A. Lubiw (Eds.). Dagstuhl Publishing.
[8] R. Farahani and M. Hekmatfar (Eds.). 2009. *Facility Location*. Springer.
[9] S. Gollapudi and A. Sharma. 2009. An Axiomatic Approach for Result Diversification. In *Proc. WWW*.
[10] J. Hopfield. 1982. Neural Networks and Physical Systems with Collective Computational Abilities. *PNAS* 79, 8 (1982).
[11] M. Johnson and et al. 2011. Quantum Annealing with Manufactured Spins. *Nature* 473, 7346 (2011).
[12] H. Markowitz. 2002. Efficient Portfolios, Sparse Matrices, and Entities: A Retrospective. *Operations Research* 50, 1 (2002).
[13] S. Mücke, N. Piatkowski, and K. Morik. 2019. Hardware Acceleration of Machine Learning Beyond Linear Algebra. In *Proc. ECML/PKDD*.
[14] S. Mücke, N. Piatkowski, and K. Morik. 2019. Learning Bit by Bit: Extracting the Essence of Machine Learning. In *Proc. LWDA*.
[15] T.E. Oliphant. 2007. Python for Scientific Computing. *Computing in Science & Engineering* 9, 3 (2007).
[16] S. Ravi, D. Rosenkrantz, and G. Tayi. 1994. Heuristic and Special Case Algorithms for Dispersion Problems. *Operations Research* 42, 2 (1994).
[17] L. Santos Rodrygo, C. Macdonald, and I. Ounis. 2011. Intent-aware Result Diversification. In *Proc. SIGIR*.
[18] R. Sifa and C. Bauckhage. 2013. Archetypical Motion: Supervised Game Behavior Learning with Archetypal Analysis. In *Proc. CIG*. IEEE.
[19] R. Sifa and C. Bauckhage. 2017. Online k-Maxoids Clustering. In *Proc. DSAA*. IEEE.
[20] R. Sifa, C. Bauckhage, and A. Drachen. 2014. Archetypal Game Recommender Systems. In *Proc. KDML-LWA*.
[21] R. Sifa, R. Yawar, R. Ramamurthy, C. Bauckhage, and K. Kersting. 2020. Matrix- and Tensor Factorization for Game Content Recommendation. *KI – Künstliche Intelligenz* 34, 1 (2020).
[22] K.-K. Sung. 1996. *Learning and Example Selection for Object and Pattern Recognition*. Ph.D. Dissertation. MIT, Center for Biological and Computational Learning.
[23] C. Thurau, K. Kersting, and C. Bauckhage. 2009. Convex Non-Negative Matrix Factorization in the Wild. In *Proc. ICDM*. IEEE.
[24] C. Thurau, K. Kersting, and C. Bauckhage. 2012. Deterministic CUR for Improved Large-Scale Data Analysis: An Empirical Study. In *Proc. SDM*. SIAM.
[25] L. Wu, Y. Wang, J. Shepherd, and Z. Zhao. 2013. Max-sum Diversification on Image Ranking with Non-uniform Matroid Constraints. *Neurocomputing* 118, Oct. (2013).