ML2R Coding Nuggets
Hopfield Nets for Bipartition Clustering

Christian Bauckhage*
Machine Learning Rhine-Ruhr
University of Bonn
Bonn, Germany

Fabrice Beaumont
Computer Science
University of Bonn
Bonn, Germany

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

ABSTRACT

We show that Hopfield networks can cluster numerical data into two salient clusters. Our derivation of a corresponding energy function is based on properties of the specific problem of 2-means clustering. Our corresponding *NumPy* code is short and simple.

1 INTRODUCTION

Recall that a Hopfield network is a recurrent neural network of n interconnected neurons $s_1, \ldots, s_n$ and that each of this neurons is a bipolar threshold unit. That is, if $\boldsymbol{w}_i \in \mathbb{R}^n$ and $\theta_i \in \mathbb{R}$ are the input weights and threshold value of neuron s_i and $\boldsymbol{s} = [s_1, \ldots, s_n]^\mathsf{T}$ denotes the overall activation pattern or global state of a Hopfield net, then neuron s_i computes its activation or local state as follows

$$s_i = \begin{cases} +1 & \text{if } \boldsymbol{w}_i^\mathsf{T}\boldsymbol{s} - \theta_i \geq 0 \\ -1 & \text{otherwise} \end{cases} \\ = \operatorname{sign}\left(\boldsymbol{w}_i^\mathsf{T}\boldsymbol{s} - \theta_i\right) \tag{1}$$

But this is to say that the global state of a Hopfield net of n neurons always is a bipolar vector $\boldsymbol{s} \in \{-1, +1\}^n$.

Also recall that we may gather all weight vectors and threshold values of a Hopfield net in an $n \times n$ matrix and an n vector, namely

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w}_1^\mathsf{T} \\ \vdots \\ \boldsymbol{w}_n^\mathsf{T} \end{bmatrix} \quad \text{and} \quad \boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \tag{2}$$

and that the energy of a Hopfield network in state $\boldsymbol{s}$ is defined as

$$E(\boldsymbol{s}) = -\tfrac{1}{2}\,\boldsymbol{s}^\mathsf{T}\boldsymbol{W}\boldsymbol{s} + \boldsymbol{\theta}^\mathsf{T}\boldsymbol{s} \tag{3}$$

Finally, recall that one can prove the following crucial result: *If* the weight matrix $\boldsymbol{W}$ of a Hopfield net is symmetric and hollow (i.e. if $\boldsymbol{W} = \boldsymbol{W}^\mathsf{T}$ and $\operatorname{diag}[\boldsymbol{W}] = \boldsymbol{0}$) and *if* its neurons update in an asynchronous manner (i.e. (re)evaluate their activation one at a time), *then* the energy of a Hopfield net can never increase. As there are "only" 2^n distinct states such a Hopfield net can be in, it will therefore settle in a (local) energy minimum after finitely many update steps [12].

All of this is thus to say that Hopfield nets can be used to (approximately) solve **quadratic unconstrained binary optimization (QUBO) problems** of the following form

$$\boldsymbol{s}^* = \underset{\boldsymbol{s}\in\{\pm 1\}^n}{\operatorname{argmin}} -\tfrac{1}{2}\,\boldsymbol{s}^\mathsf{T}\boldsymbol{W}\boldsymbol{s} + \boldsymbol{\theta}^\mathsf{T}\boldsymbol{s} \tag{4}$$

QUBOs like this often arise in the context of subset selection- or bipartition problems which, in turn, occur frequently in data mining, pattern recognition, machine learning, or artificial intelligence.

* 0000-0001-6615-2128

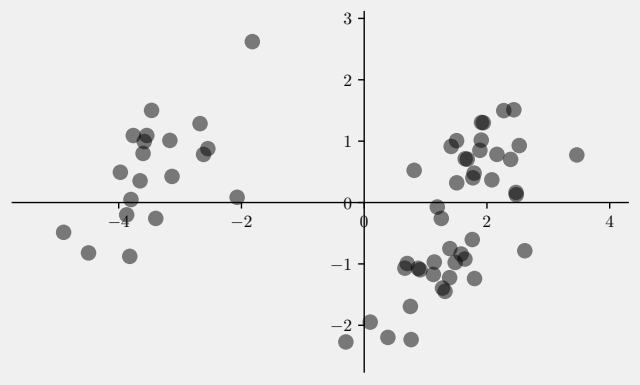

Figure 1: A set $\mathcal{X}$ of $n = 64$ data points $\boldsymbol{x}_i \in \mathbb{R}^2$. Apparently, the data form two distinct clusters $\mathcal{X}_1$ and $\mathcal{X}_2$. To identify these clusters automatically is a bipartition problem that can be solved by a Hopfield network.

Generally speaking, given a set $\mathcal{X}$, a *subset selection problem* asks for a subset $\mathcal{X}' \subset \mathcal{X}$ where the elements of $\mathcal{X}'$ have to meet certain criteria. A *bipartition problem*, on the other hand, asks for a partition $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2$ where the elements of each of the two disjoint subsets $\mathcal{X}_1$ and $\mathcal{X}_2$ have to meet certain criteria.[1] Their connection to QUBOs is as follows: *If* we can formulate a subset selection- or bipartition problem as a QUBO over bipolar vectors $\boldsymbol{s}$, we can think of the entries of the solution $\boldsymbol{s}^*$ as indicator variables. For a subset selection problem, they would define the sought after subset as

$$\mathcal{X}' = \left\{x_i \in \mathcal{X} \;\middle|\; s_i^* = +1\right\} \tag{5}$$

and, for a bipartition problem, we would have

$$\mathcal{X}_1 = \left\{x_i \in \mathcal{X} \;\middle|\; s_i^* = +1\right\} \\ \mathcal{X}_2 = \left\{x_i \in \mathcal{X} \;\middle|\; s_i^* = -1\right\} \tag{6}$$

The max-sum diversification problem we considered in [3] is easily recognized as a subset selection problem. Other obvious examples include the computation of medoids in k-medoids clustering or the identification of support vectors in SVM training. But subset selection problems may also come in disguise. An example is the sorting problem in [8] where we were searching for an optimal permutation matrix, i.e., we were implicitly considering the set of all permutation matrices and had to select a singleton subset.

[1] Note that subset selection is a special case of set bipartition because $\mathcal{X} = \mathcal{X}' \cup (\mathcal{X} \setminus \mathcal{X}')$.

In this note, we now look at a bipartition problem, namely the problem of clustering a set $\mathcal{X}$ into two salient, non-overlapping clusters $\mathcal{X}_1$ and $\mathcal{X}_2$.

As a practical example, we consider data such as the $n = 64$ data points $\boldsymbol{x}_i \in \mathbb{R}^2$ in Fig. 1. Looking at this figure, it is obvious that our exemplary data form two distinct clusters and our goal is to determine these by means of running a Hopfield net.

The main challenge regarding this idea is to formulate bipartition clustering as a QUBO. However, our discussion in section 2 will show that this can be done. In fact, the resulting QUBO is simple and can easily be turned into a Hopfield energy minimization problem. Even better, the resulting energy minimization problem can be solved by means of Hopfield nets which update in an informed and thus efficient manner [20]. Our *NumPy* implementation of the corresponding steepest energy descent procedure in section 3 will therefore be simple, too.

Readers who would like to experiment with our code should be familiar with *NumPy* [19] and need to

```
import numpy as np
import numpy.random as rnd
```

2 THEORY

In this section, we devise a Hopfield energy function for the problem of clustering a sample $\mathcal{X}$ of n data points $\boldsymbol{x} \in \mathbb{R}^m$ into two disjoint clusters $\mathcal{X}_1$ and $\mathcal{X}_2$ where $|\mathcal{X}_1| = n_1$, $|\mathcal{X}_2| = n_2$, and $n_1 + n_2 = n$.

2.1 A Hopfield Energy for Clustering

Without loss of generality but importantly, we assume that the given data points have been normalized to zero mean so that

$$\boldsymbol{\mu} = \frac{1}{n} \sum_{\boldsymbol{x} \in \mathcal{X}} \boldsymbol{x} = \frac{1}{n} \sum_{j=1}^{2} \sum_{\boldsymbol{x} \in \mathcal{X}_j} \boldsymbol{x} = \frac{1}{n} \left[n_1 \, \boldsymbol{\mu}_1 + n_2 \, \boldsymbol{\mu}_2 \right] = \mathbf{0} \quad (7)$$

This crucial assumption implies $n_1 \, \boldsymbol{\mu}_1 = -n_2 \, \boldsymbol{\mu}_2$ which is to say that the two means $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ of any two non-empty, disjoints subsets $\mathcal{X}_1$ and $\mathcal{X}_2$ of $\mathcal{X}$ will be of opposite sign.

Now, assume we were to use $k = 2$ means clustering to solve our problem. Then, we would typically try to determine two cluster means $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ that minimize the *within cluster scatter*

$$S_W = \sum_{j=1}^{k} \sum_{\boldsymbol{x} \in \mathcal{X}_j} \left\| \boldsymbol{x} - \boldsymbol{\mu}_j \right\|^2 \quad (8)$$

Indeed, most of the well known k-means algorithms such as those of Lloyd [14], Hartigan [11], or MacQueen [16] consider this objective.

However, in this note, we follow a different route and observe that minimizing the above within cluster scatter is equivalent to maximizing the *between cluster scatter*

$$S_B = \sum_{i=1}^{k} \sum_{j=1}^{k} n_i \, n_j \left\| \boldsymbol{\mu}_i - \boldsymbol{\mu}_j \right\|^2 \quad (9)$$

This claim holds for any $k > 1$ and follows from Fisher's analysis of variance [1, 10]. It establishes that the *total scatter* of the given data can be written as

$$S_T = \sum_{\boldsymbol{x} \in \mathcal{X}} \left\| \boldsymbol{x} - \boldsymbol{\mu} \right\|^2 = S_W + \frac{1}{2\,n} S_B \quad (10)$$

which, since S_T and n are positive constants, implies that any decrease of S_W in (8) entails an increase of S_B in (9).

For our specific k-means clustering problem where $k = 2$, the general expression of the maximization objective in (9) is overly complicated. Indeed, if $k = 2$, it is an easy exercise to show that it can be simplified to

$$S_B = 2 \, n_1 \, n_2 \left\| \boldsymbol{\mu}_1 - \boldsymbol{\mu}_2 \right\|^2 \quad (11)$$

Interestingly, this simplification provides us with an intuition as to why k-means clustering is agnostic of cluster shapes and distances and often produces clusters of about equal size [15]: In order for S_B in (11) to be large, both the distance $\|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\|$ between the two cluster centers and the product $n_1 n_2$ of the two cluster sizes have to be large. However, since the sum $n_1 + n_2 = n$ is fixed, the product of the cluster sizes will be maximal if $n_1 = n_2 = \frac{n}{2}$ [2].

This observation now hands us a heuristic argument for how to rewrite the objective in (11) in a way that will lead to a QUBO.

Since k-means clustering often tends to produce clusters of about equal sizes, we next boldly assume that the two clusters that result from maximizing (11) are of about the same size $n_1 \approx n_2 \approx \frac{n}{2}$. This assumption is not generally justified but allows us to approximate

$$2 \, n_1 \, n_2 \left\| \boldsymbol{\mu}_1 - \boldsymbol{\mu}_2 \right\|^2 \approx 2 \, \tfrac{n^2}{4} \left\| \boldsymbol{\mu}_1 - \boldsymbol{\mu}_2 \right\|^2 \quad (12)$$

Again, this approximation will not hold true in general. But it is interesting nevertheless, because we can rewrite the right hand side of (12) as

$$2 \, \tfrac{n^2}{4} \left\| \boldsymbol{\mu}_1 - \boldsymbol{\mu}_2 \right\|^2 = 2 \left\| \tfrac{n}{2} \left[\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2 \right] \right\|^2 \quad (13)$$

$$= 2 \left\| n_1 \, \boldsymbol{\mu}_1 - n_2 \, \boldsymbol{\mu}_2 \right\|^2 \quad (14)$$

which, in turn, turns the problem of $k = 2$ means clustering into the problem of having to solve

$$\boldsymbol{\mu}_1^*, \, \boldsymbol{\mu}_2^* = \underset{\boldsymbol{\mu}_1, \, \boldsymbol{\mu}_2}{\operatorname{argmax}} \left\| n_1 \, \boldsymbol{\mu}_1 - n_2 \, \boldsymbol{\mu}_2 \right\|^2 \quad (15)$$

To appreciate what this approximation of our clustering problem buys us, we next observe that the squared norm in the objective of (15) can be expressed in a form that does not explicitly depend on the two decision variables $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$. In order to see this, we first gather the given data point in a data matrix

$$\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n] \in \mathbb{R}^{m \times n} \quad (16)$$

and second consider two binary indicator vectors $\boldsymbol{z}_1, \boldsymbol{z}_2 \in \{0, 1\}^n$. These are supposed to indicate cluster memberships in the sense that entry i of $\boldsymbol{z}_j$ equals 1 if $\boldsymbol{x}_i \in \mathcal{X}_j$ and 0 otherwise. This way, we can write

$$n_1 \, \boldsymbol{\mu}_1 = \boldsymbol{X} \boldsymbol{z}_1 \quad (17)$$

$$n_2 \, \boldsymbol{\mu}_2 = \boldsymbol{X} \boldsymbol{z}_2 \quad (18)$$

and consequently find that

$$\left\| n_1 \, \boldsymbol{\mu}_1 - n_2 \, \boldsymbol{\mu}_2 \right\|^2 = \left\| \boldsymbol{X} \left[\boldsymbol{z}_1 - \boldsymbol{z}_2 \right] \right\|^2 = \left\| \boldsymbol{X} \boldsymbol{s} \right\|^2 \quad (19)$$

Note that the vector $\boldsymbol{s}$ we introduced in (19) is guaranteed to be a bipolar vector because, in hard k-means clustering, every data point is assigned to one and only one cluster. The difference of the binary cluster indicator vectors $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ therefore amounts to

$$\boldsymbol{z}_1 - \boldsymbol{z}_2 = \boldsymbol{z}_1 - \left[\mathbf{1} - \boldsymbol{z}_1 \right] = 2 \, \boldsymbol{z}_1 - \mathbf{1} = \boldsymbol{s} \in \{-1, +1\}^n \quad (20)$$

This, however, establishes that there exists a Hopfield energy minimization formulation for the problem of $k = 2$ means clustering of zero mean data. First of all, since

$$\left\| \boldsymbol{X}\boldsymbol{s} \right\|^2 = \boldsymbol{s}^\mathsf{T}\boldsymbol{X}^\mathsf{T}\boldsymbol{X}\boldsymbol{s} \tag{21}$$

is convex in $\boldsymbol{s}$, the maximization problem in (15) is equivalent to the following minimization problem

$$\boldsymbol{s}^* = \underset{\boldsymbol{s}\in\{\pm 1\}^n}{\operatorname{argmin}} -\boldsymbol{s}^\mathsf{T}\boldsymbol{X}^\mathsf{T}\boldsymbol{X}\,\boldsymbol{s} \tag{22}$$

$$= \underset{\boldsymbol{s}\in\{\pm 1\}^n}{\operatorname{argmin}} -\boldsymbol{s}^\mathsf{T}\boldsymbol{Q}\,\boldsymbol{s} \tag{23}$$

Because of (7), this will necessarily yield a solution vector $\boldsymbol{s}^*$ whose entries are not all equal and thus induce a clustering of the data gathered in matrix $\boldsymbol{X}$.

Second of all, we note that we can decompose the coupling matrix in (23) into $\boldsymbol{Q} = \boldsymbol{H} + \boldsymbol{D}$ where

$$H_{ij} = \begin{cases} 0 & \text{if } i = j \\ \boldsymbol{x}_i^\mathsf{T}\boldsymbol{x}_j & \text{otherwise} \end{cases} \tag{24}$$

$$D_{ij} = \begin{cases} \boldsymbol{x}_i^\mathsf{T}\boldsymbol{x}_j & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

so that our problem becomes

$$\boldsymbol{s}^* = \underset{\boldsymbol{s}\in\{\pm 1\}^n}{\operatorname{argmin}} -\boldsymbol{s}^\mathsf{T}\boldsymbol{H}\,\boldsymbol{s} - \boldsymbol{s}^\mathsf{T}\boldsymbol{D}\,\boldsymbol{s} \tag{26}$$

However, $\boldsymbol{H}$ is a hollow matrix (i.e. has a diagonal of all zeros) and $\boldsymbol{D}$ is a diagonal matrix for which we have $\boldsymbol{s}^\mathsf{T}\boldsymbol{D}\,\boldsymbol{s} = const$. Hence, if we finally introduce a weight matrix

$$\boldsymbol{W} = 2\,\boldsymbol{H} \tag{27}$$

our problem can be written as a Hopfield energy minimization problem without threshold parameters, namely

$$\boldsymbol{s}^* = \underset{\boldsymbol{s}\in\{\pm 1\}^n}{\operatorname{argmin}} -\tfrac{1}{2}\,\boldsymbol{s}^\mathsf{T}\boldsymbol{W}\,\boldsymbol{s} \tag{28}$$

2.2 Running Hopfield Nets for Clustering

As QUBOs go, the energy minimization problem for bipartition clustering in (28) is a rather simple one. There therefore is a simple steepest descent algorithm that causes a corresponding Hopfield net to quickly converge to a stable state of minimum energy.

Let $\boldsymbol{s}_t$ and $\boldsymbol{s}_{t+1}$ be two consecutive states during the temporal evolution of a Hopfield net and assume that neuron s_i updated at time t. It is well known that the energy difference in this situation amounts to $\Delta E = E(\boldsymbol{s}_{t+1}) - E(\boldsymbol{s}_t) = 2 \cdot \left(\boldsymbol{s}_t\right)_i \cdot \left(\boldsymbol{w}_i^\mathsf{T}\boldsymbol{s}_t - \theta_i\right) \leq 0$. Yet, sine the problem in (28) does not even involve threshold parameters, we actually only need to consider

$$\Delta E = 2 \cdot \left(\boldsymbol{s}_t\right)_i \cdot \boldsymbol{w}_i^\mathsf{T}\boldsymbol{s}_t \tag{29}$$

to determine which neuron should update its activation in iteration t so as to maximally decrease the network's current energy $E(\boldsymbol{s}_t)$. Letting $\odot$ denote the Hadamard product, we could simply compute a vector

$$\Delta\boldsymbol{e} = 2 \cdot \boldsymbol{s}_t \odot \boldsymbol{W}\boldsymbol{s}_t \tag{30}$$

of expected energy decrements, determine the index u of its most negative entry

$$u = \underset{i}{\operatorname{argmin}}\ \Delta e_i \tag{31}$$

and then update neuron s_u to obtain a new global state where

$$\left(\boldsymbol{s}_{t+1}\right)_i = \begin{cases} \left(\boldsymbol{s}_t\right)_u \cdot \operatorname{sign}\left(\Delta e_u\right) & \text{if } i = u \\ \left(\boldsymbol{s}_t\right)_i & \text{otherwise} \end{cases} \tag{32}$$

Recall that we already discussed this idea of greedy steepest energy descent when we looked at Hopfield nets for sorting [8]. All that has changed here is that the computation of energy deltas in (30) has simplified because our current problem in (28) is simpler.

3 PRACTICE

Given the above theory, we now discuss *NumPy* implementations of Hopfield nets for bipartition clustering of data $\boldsymbol{x}_i \in \mathbb{R}^m$.

Throughout, we assume that we are given a data matrix $\boldsymbol{X}$ such as in (16) which we represent as a 2D *NumPy* array `matX` whose numbers of rows and columns can be determined using

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

Recall that, for our above ideas to make sense, we have to insist on the given data to have zero mean. Hence, we normalize our data matrix as follows

```
matX = matX - np.mean(matX, axis=1).reshape(m,1)
```

Next, we initialize the hollow weight matrix $\boldsymbol{W}$ of a Hopfield net for bipartition clustering. To this end, we simply use

```
matW = 2 * matX.T @ matX
np.fill_diagonal(matW, 0)
```

and point out the the *NumPy* convenience function `fill_diagonal` operates in place, i.e. modifies `matW` directly and does not have a return value.

Having initialized the weights of our Hopfield net, we still need to set its initial global state $\boldsymbol{s}$. Here, we opt for a random initialization which we realize as follows: First, we randomly sample a binary vector $\boldsymbol{z} \in \{0,1\}^n$ whose elements are drawn from a Bernoulli distribution with success probability $p = 0.5$

```
vecZ = rnd.binomial(n=1, p=0.5, size=n)
```

Note that function `binomial` in *NumPy*'s `random` module draws from a binomial distribution and that a binomial distribution over only one trial (`n=1`) is a Bernoulli distribution. Having sampled a binary vector $\boldsymbol{z}$, we then make use of the fact that $2\,\boldsymbol{z} - 1 = \boldsymbol{s}$ is bipolar and compute a corresponding 1D array

```
vecS = 2 * vecZ - 1
```

After these comparatively simple preparations, we are good to go and may call

```
vecS = hnetRunGreedy(vecS, matW, tmax=100)
```

to solve our clustering problem. This uses function `hnetRunGreedy` in Listing 1 which implements the greedy update mechanism we discussed above.

The first two parameters of `hnetRunGreedy` are self-explanatory. Parameter `tmax` indicates the number of update iterations our Hopfield should perform and `verbose` is a flag variable which, when

Listing 1: greedily running a sorting Hopfield net

```
1  def signum(x):
2      return np.where(x >= 0, +1, -1)
3
4
5  def hnetRunGreedy(vecS, matW, tmax=100, verbose=False):
6      for t in range(tmax):
7          if verbose:
8              s = ' '.join(['+' if x >= 0 else '-' for x in vecS])
9              E = -0.5 * vecS @ matW @ vecS
10             print ('{:4d}  {}  {:+.1f}'.format(t, s, E))
11
12         dltE = vecS * (matW @ vecS)
13         updt = np.argmin(dltE)
14
15         vecS[updt] = vecS[updt] * signum(dltE[updt])
16
17     return vecS
```

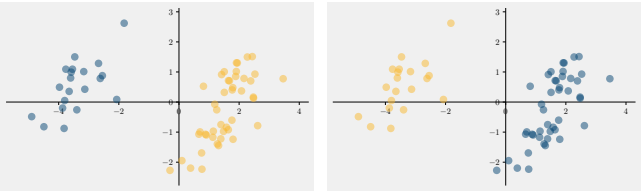

Figure 2: Two different clustering obtained from running our clustering Hopfield network on the data in Fig. 1.

`True`, causes printing of status information. To be specific, the printed information consists of the current update cycle t, global state $\boldsymbol{s}_t$, and energy $E(\boldsymbol{s}_t)$ of the network. Printing happens in lines 7–10 and does not merit discussion. Exemplary outputs are shown in Figs. 3 and 4.

Lines 12–15 of `hnetRunGreedy` are *NumPy* implementations of equations (30)–(32). Here, it is worth mentioning that line 15 involves a custom made sign function `signum` which is defined in lines 1 and 2 of Listing 1. We use `signum` instead of the *NumPy* function `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

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

Finally, `hnetRunGreedy` returns a state vector $\boldsymbol{s}$ which represents the activation of the neurons of the network after `tmax` updates.

All that is left to truly solve our clustering problem is to use this vector to partition matrix $\boldsymbol{X}$ into two smaller matrices $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$. This can, for instance, be accomplished using

```
matX1 = matX[:,vecS>0]
matX2 = matX[:,vecS<0]
```

Figure 2 displays two clustering results we obtained in different runs of our Hopfield net. The former resulted from starting the Hopfield net in the initial state shown at the top of Fig. 3; the latter resulted from starting it in the initial state shown at the top of Fig. 4. The fact that two different runs with different random initializations lead to two different results is no reason for concern. This is because or Hopfield energy function for bipartition clustering "suffers" from a form of symmetry: If $\boldsymbol{s}^*$ solves the optimization problem in (28), then so does $-\boldsymbol{s}^*$ since we did not specify whether an entry of, say, $+1$ is supposed to indicate membership to cluster $\mathcal{X}_1$ or cluster $\mathcal{X}_2$. However, since such ambiguity might be undesirable, we will get back to this issue in a later note.

Three more things are worth pointing out: First of all, in either of the above examples, it took the network less than $n/2$ iterations to converge to a stable state representing a correct solution (see Figs. 3 and 4). This is due to our greedy steepest energy descent mechanism and the fact that, initially, neurons were set to be active with probability $p = 0.5$.

Second of all, our derivation of the QUBO formulation of bipartition clustering involved the heuristic assumption that the two clusters to be found are of about equal size. This assumption will certainly not hold true in general. Indeed, for our practical example it does not. The two salient cluster in Fig. 1 are of considerably different sizes, as the one contains more than twice as many data points that the other. However, our practical results in Figs. 2–4 suggest that this violation of our assumption does not impact the capability of the resulting Hopfield nets to cluster correctly. In other words, our Hopfield energy function is more data agnostic than the arguments required to derive it.

Third of all, not all is well! The example and practical results we considered so far seem to suggest that the approach in this note works flawlessly. However, it turns out that our clustering Hopfield nets are not as data agnostic as we would like. The problem is not so much the size of salient clusters but their relative location.

This is exemplified in Fig. 5 which shows another data set and a Hopfield clustering result. Note that there as many (zero mean) data points in Fig. 5 as there are in Fig. 1 and that the two salient clusters in Fig. 5 are of exactly the same sizes as those in in Fig. 1. What is different is that the two clusters are now closer together and this makes it impossible for our Hopfield nets to determine the correct result.

This effect is independent of initialization and update mechanism but a consequence of the fact that we based our QUBO formulation of bipartition clustering on the $k = 2$ means clustering objective. In fact, k-means clustering, too, would not be able to determine the correct result for the data in Fig. 5. Kernel k-means clustering, on the other hand, would stand a better chance. This then suggests to think about Hopfield energy functions that involve kernelized weights and we will get back to this idea in another note.

4 CONCLUSION

This note demonstrated that Hopfield networks can cluster data into two salient clusters or, put differently, can solve bipartition clustering problems.

We focused on the case where the data are m-dimensional, real valued vectors and derived a corresponding QUBO formulation for bipartition clustering of such data. We based our derivation on an alternative, less well known objective for k-means clustering and a heuristic argument. Given this QUBO, it was then easy to devise an energy minimization problem that can be solved by Hopfield networks. In fact, the resulting energy minimization problem is rather

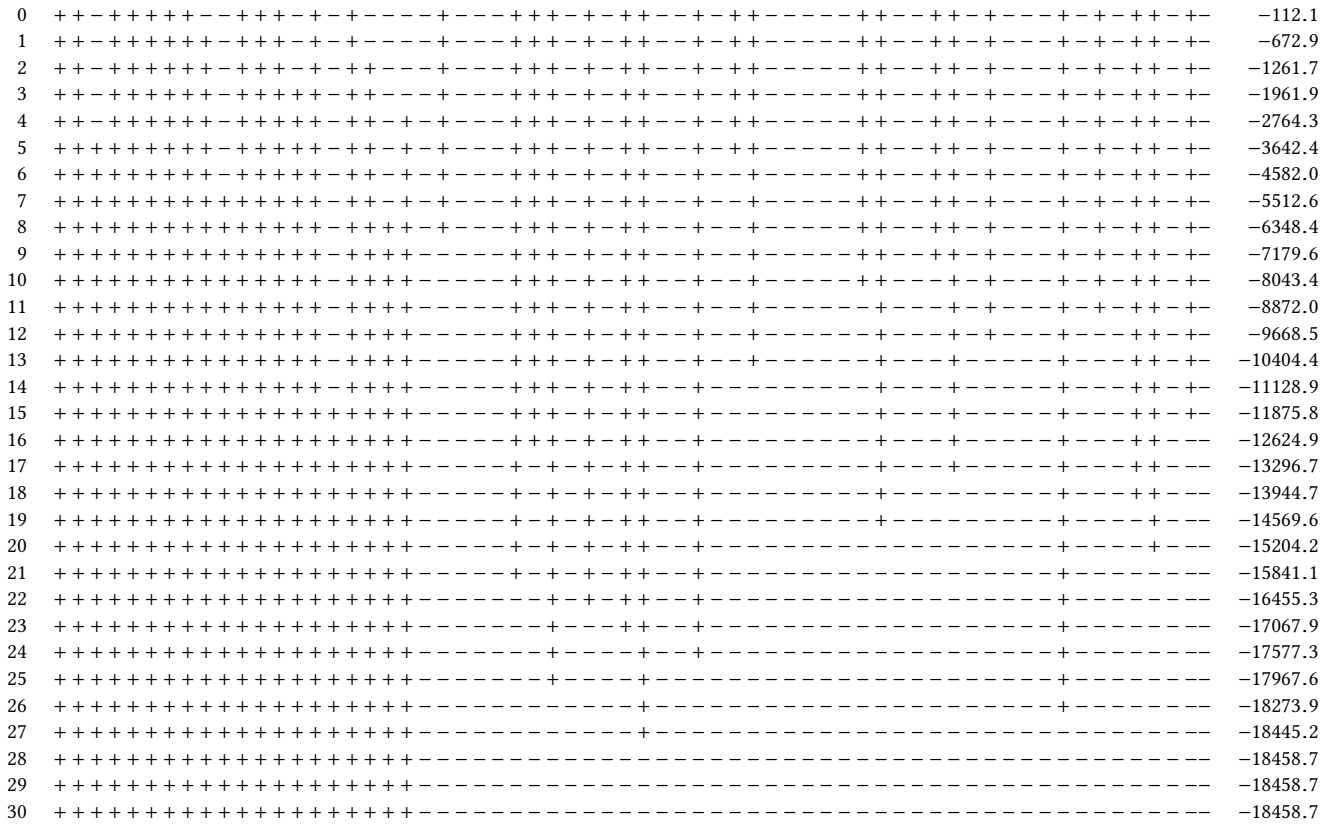

Figure 3: Visualization of the temporal evolution of the state and energy of a Hopfield network of $n = 64$ neurons that solves the bipartion clustering problem for the data in Fig. 1. The initial state s_0 was randomly initialized and, in each subsequent iteration, our greedy update mechanisms causes one neuron to flip its activation until the network reaches a stable state. In this example, it took less than 30 iterations for the network to converge to a correct solution.

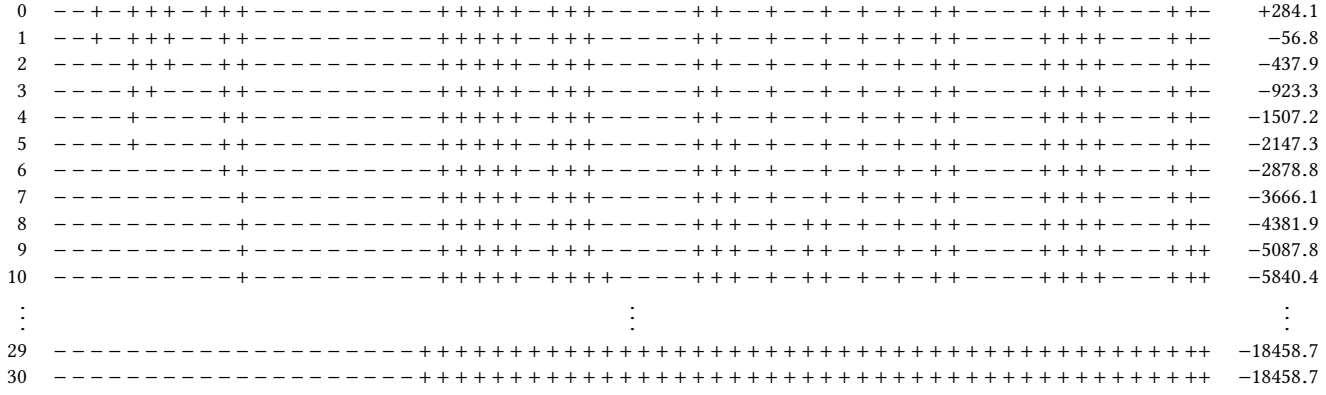

Figure 4: A similar visualization as in Fig. 3 but for an energy minimization process that started in a different initial state. Here, too, it took less than 30 iterations for the Hopfield network to settle in a stable state that represents a correct solution.

simple and can be solved by Hopfield networks whose neurons update in a greedy or informed manner and thus quickly converge to a stable state.

A simple practical example showed our ideas to work well but also to suffer from ambiguities because any (unconstrained) bipartition clustering problem has two valid solutions. Another, not so simple example, on the other hand, pointed to the fact that our

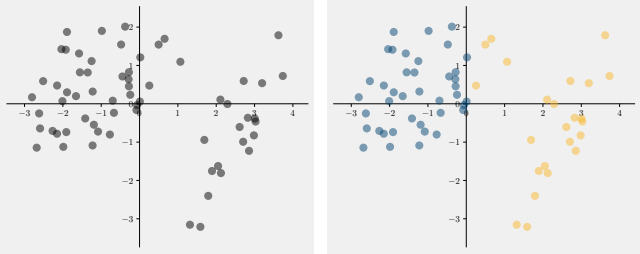

Figure 5: Another data set and a clustering result obtained from running our clustering Hopfield network.

straightforward approach is furthermore affected by the geometry of the data under consideration. We will get back to both these issues in later notes and see that there are surprisingly simple solutions.

As in most our notes on Hopfield networks and their capabilities, we cannot close without mentioning that, if a Hopfield net can solve a problem, then so can an adiabatic quantum computer [7, 13] or a specifically designed, compute and energy efficient digital annealer [9, 17, 18]. In fact, for bipartion clustering problems, we already demonstrated this in several research papers [4–6]. This therefore suggests that very fast clustering might be possible once quantum computers become technically mature enough to deal with large problem sizes.

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] C. Bauckhage. 2018. k-Means and Fisher's Analysis of Variance. researchgate.net.
[2] C. Bauckhage. 2019. Lecture Notes on Machine Learning: Maximum Product of Numbers of Constant Sum. B-IT, University of Bonn.
[3] C. Bauckhage, F. Beaumont, and S. Müller. 2021. *ML2R Coding Nuggets: Hopfield Nets for Max-Sum Diversification.* Technical Report. MLAI, University of Bonn.
[4] C. Bauckhage, E. Brito, K. Cvejoski, C. Ojeda, R. Sifa, and S. Wrobel. 2017. Ising Models for Binary Clustering via Adiabatic Quantum Computing. In *Proc. EMM-CVPR.* Springer.
[5] C. Bauckhage, C. Ojeda, R. Sifa, and S. Wrobel. 2018. Adiabatic Quantum Computing for Kernel k=2 Means Clustering. In *Proc. KDML-LWDA.*
[6] C. Bauckhage, N. Piatkowske, R. Sifa, D. Hecker, and S. Wrobel. 2019. A QUBO Formulation of the k-Medoids Problem. In *Proc. KDML-LWDA.*
[7] C. Bauckhage, R. Sanchez, and R. Sifa. 2020. Problem Solving with Hopfield Networks and Adiabatic Quantum Computing. In *Proc. IJCNN.* IEEE.
[8] C. Bauckhage and P. Welke. 2021. *ML2R Coding Nuggets: Hopfield Nets for Sorting.* Technical Report. MLAI, University of Bonn.
[9] J. Boyd. 2018. Silicon Chip Delivers Quantum Speeds. *IEEE Spectrum* 55, 7 (2018).
[10] R.A. Fisher. 1921. On the Probable Error of a Coefficient Correlation Deduced from a Small Sample. *Metron* 1 (1921).
[11] J.A. Hartigan and M.A. Wong. 1979. Algorithm AS 136: A k-Means Clustering Algorithm. *J. of the Royal Statistical Society C* 28, 1 (1979).
[12] J. Hopfield. 1982. Neural Networks and Physical Systems with Collective Computational Abilities. *PNAS* 79, 8 (1982).
[13] M. Johnson and et al. 2011. Quantum Annealing with Manufactured Spins. *Nature* 473, 7346 (2011).
[14] S.P. Lloyd. 1982. Least Squares Quantization in PCM. *IEEE Trans. Information Theory* 28, 2 (1982).
[15] D.J.C. MacKay. 2003. *Information Theory, Inference, & Learning Algorithms.* Cambridge University Press.
[16] J.B. MacQueen. 1967. Some Methods for Classification and Analysis of Multivariate Observations. In *Proc. Berkeley Symp. on Mathematical Statistics and Probability.*
[17] S. Mücke, N. Piatkowski, and K. Morik. 2019. Hardware Acceleration of Machine Learning Beyond Linear Algebra. In *Proc. ECML/PKDD.*
[18] S. Mücke, N. Piatkowski, and K. Morik. 2019. Learning Bit by Bit: Extracting the Essence of Machine Learning. In *Proc. LWDA.*
[19] T.E. Oliphant. 2007. Python for Scientific Computing. *Computing in Science & Engineering* 9, 3 (2007).
[20] L. von Rueden, S. Mayer, K. Beckh, B. Georgiev, S. Giesselbach, R. Heese, B. Kirsch, J. Pfrommer, A. Pick, R. Ramamurthy, M. Walczak, J. Garcke, C. Bauckhage, and J. Schuecker. 2019. Informed Machine Learning – A Taxonomy and Survey of Integrating Knowledge into Learning Systems. *arXiv:1903.12394 [stat.ML]* (2019).