ML2R Coding Nuggets
Unambiguous Bipartition Clustering with Hopfield Nets

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 revisit Hopfield nets for bipartition clustering and tweak the underlying energy function such that it has a unique global minimum. In other words, we show how to remove ambiguity from the bipartition clustering problem. Our corresponding *NumPy* code is short and simple.

1 INTRODUCTION

We once again revisit the idea of running Hopfield nets to cluster a set $\mathcal{X}$ into two salient clusters $\mathcal{X}_+$ and $\mathcal{X}_-$ such that $\mathcal{X}_+ \cup \mathcal{X}_- = \mathcal{X}$ and $\mathcal{X}_+ \cap \mathcal{X}_- = \emptyset$. For simplicity, we will once more assume that the n elements of $\mathcal{X}$ are data points $\boldsymbol{x}_i \in \mathbb{R}^m$.

We already saw [1, 2], that the corresponding Hopfield energy minimization problem is

$$\boldsymbol{s}_* = \underset{\boldsymbol{s} \in \{\pm 1\}^n}{\operatorname{argmin}} -\frac{1}{2}\, \boldsymbol{s}^\top \boldsymbol{W} \boldsymbol{s} \tag{1}$$

where $\boldsymbol{s}$ is a state vector of a Hopfield net of n neurons s_i and the entries of the $n \times n$ weight matrix $\boldsymbol{W}$ of this network are given by

$$W_{ij} = \begin{cases} 0 & \text{if } i = j \\ 2\,[\boldsymbol{JKJ}]_{ij} & \text{otherwise} \end{cases} \tag{2}$$

Looking at this definition, we recall that $\boldsymbol{JKJ}$ is a *centered* kernel matrix. That is, $\boldsymbol{J} = \boldsymbol{I} - \frac{1}{n}\,\boldsymbol{1}\boldsymbol{1}^\top$ is a centering matrix [7] and the entries of the kernel matrix $\boldsymbol{K}$ amount to

$$K_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{3}$$

Here, the function $K : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ is an "appropriate" Mercer kernel that should be chosen with respect to the application at hand. For example, we previously worked with linear, polynomial, and Gaussian kernels [1]

Since the weight matrix in (2) is symmetric ($\boldsymbol{W} = \boldsymbol{W}^\top$) and hollow ($\operatorname{diag}[\boldsymbol{W}] = \boldsymbol{0}$), we know that our Hopfield net will, in the long run, settle in a state $\boldsymbol{s}^\infty$ of (locally) minimal energy, if its neurons update asynchronously [8]. With respect to our clustering problem, we can then regard the entries s_i^∞ of $\boldsymbol{s}^\infty$ as cluster labels or cluster membership indicators and partition our data into

$$\mathcal{X}_+ = \left\{ x_i \in \mathcal{X} \;\middle|\; s_i^\infty = +1 \right\} \tag{4}$$

$$\mathcal{X}_- = \left\{ x_i \in \mathcal{X} \;\middle|\; s_i^\infty = -1 \right\} \tag{5}$$

Depending on the choice of kernel and initial state, clustering Hopfield nets may indeed end up in a final state of globally minimal energy whose sign pattern indicates convincing clusters (see the examples in Fig. 1). Alas, there is a form of symmetry or ambiguity. Since the discrete optimization problem in (1) does *not* involve a bias term $\boldsymbol{\theta}^\top \boldsymbol{s}$, it has two optimal solutions. Namely, if network state $\boldsymbol{s}_*$ is an optimal solution to (1), then so is $-\boldsymbol{s}_*$, because

$$(-\boldsymbol{s}_*)^\top \boldsymbol{W} (-\boldsymbol{s}_*) = (-1)^2 \cdot \boldsymbol{s}_*^\top \boldsymbol{W} \boldsymbol{s}_* = \boldsymbol{s}_*^\top \boldsymbol{W} \boldsymbol{s}_* \tag{6}$$

Our clustering model thus "suffers" from a superfluous degree of freedom. Since this may be undesirable in practice, our goal in this note is to remove this degree of freedom and to rewrite the optimization problem in (1) such that is has a unique optimal solution.

As we shall see in section 2, this can be easily accomplished. In section 3 we then show that our ideas are also easy to implement.

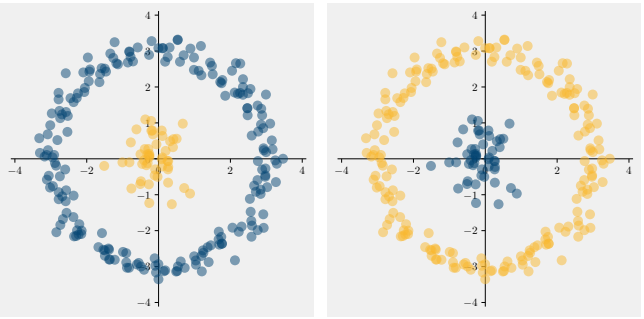

Figure 1: The data in this figure form two salient clusters which can be detected by a Hopfield network. Yet, there is ambiguity: Points in the blob-like cluster might be labeled $+1$ (orange) and points in the circular cluster -1 (blue). But points in the blob-like cluster could just as well be labeled -1 and points in the circular cluster $+1$.

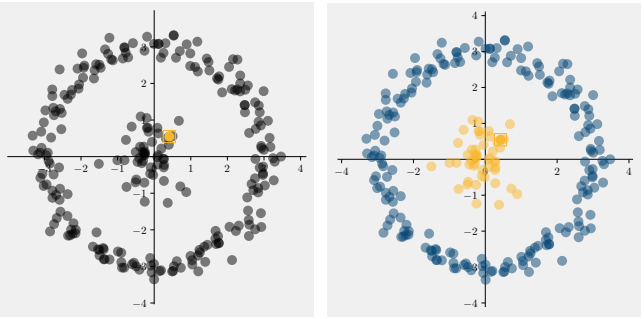

Figure 2: Labeling one of the given data points as an element of one of the sought after clusters removes ambiguity and causes the optimal solution to our clustering problem to be unique.

2 THEORY

A simple idea for how to resolve the ambiguity problem in bipartition clustering is to (manually) assign one of the the given data points to one of the sought after clusters. Without loss of generality, we will assume that this data point is $\boldsymbol{x}_n$ and that it is assigned to cluster $\mathcal{X}_+$ (see the example in Fig. 2). This way, the optimal solution to our clustering problem is to put all points that cluster with $\boldsymbol{x}_n$ into cluster $\mathcal{X}_+$ and all other points into cluster $\mathcal{X}_-$.

From the point of view of a Hopfield net that is supposed to find this optimal partition, our choice of pre-assigning data point $\boldsymbol{x}_n$ to cluster $\mathcal{X}_+$ means that neuron s_n should have a constant activation of +1 during the evolution of the network.

Next, we discuss how this behavior of neuron s_n impacts the energy function the network minimizes. Hence, we consider the objective function of the optimization problem in (1), briefly ignore its overall scaling factor of $-1/2$, and note that it can be written as

$$\boldsymbol{s}^\top \boldsymbol{W}\, \boldsymbol{s} = \sum_{i=1}^{n} \sum_{j=1}^{n} s_i\, W_{ij}\, s_j \tag{7}$$

$$= \sum_{i=1}^{n} s_i \sum_{j=1}^{n} W_{ij}\, s_j \tag{8}$$

$$= \sum_{i=1}^{n} s_i \left[\sum_{j=1}^{n-1} W_{ij}\, s_j + W_{in}\, s_n \right] \tag{9}$$

$$= \sum_{i=1}^{n} s_i \sum_{j=1}^{n-1} W_{ij}\, s_j + \sum_{i=1}^{n} s_i\, W_{in} \tag{10}$$

where the last step used our basic assumption of $s_n = +1$.

Continuing, we next split the two sums over i in (10) into two parts. For the first sum, we find

$$\sum_{i=1}^{n} s_i \sum_{j=1}^{n-1} W_{ij}\, s_j = \sum_{i=1}^{n-1} s_i \sum_{j=1}^{n-1} W_{ij}\, s_j + s_n \sum_{j=1}^{n-1} W_{nj}\, s_j \tag{11}$$

$$= \sum_{i=1}^{n-1} s_i \sum_{j=1}^{n-1} W_{ij}\, s_j + \sum_{j=1}^{n-1} W_{nj}\, s_j \tag{12}$$

where we again used that $s_n = +1$. For the second sum, we find

$$\sum_{i=1}^{n} s_i\, W_{in} = \sum_{i=1}^{n-1} s_i\, W_{in} + s_n\, W_{nn} = \sum_{i=1}^{n-1} s_i\, W_{in} \tag{13}$$

Here, we applied the crucial fact that $W_{nn} = 0$ which holds true because the weight matrix of our Hopfield net is *hollow*.

Putting together all our results so far, we therefore have

$$\boldsymbol{s}^\top \boldsymbol{W}\, \boldsymbol{s} = \sum_{i=1}^{n-1} \sum_{j=1}^{n-1} s_i\, W_{ij}\, s_j + \sum_{j=1}^{n-1} W_{nj}\, s_j + \sum_{i=1}^{n-1} s_i\, W_{in} \tag{14}$$

At this point, we recall another property of the weight matrix of a Hopfield net, namely that it is *symmetric*. But this symmetry implies the equality

$$\sum_{j=1}^{n-1} W_{nj}\, s_j = \sum_{i=1}^{n-1} s_i\, W_{in} \tag{15}$$

This then establishes our main result, namely that (14) can also be written as

$$\boldsymbol{s}^\top \boldsymbol{W}\, \boldsymbol{s} = \sum_{i=1}^{n-1} \sum_{j=1}^{n-1} s_i\, W_{ij}\, s_j + 2 \sum_{j=1}^{n-1} W_{nj}\, s_j \tag{16}$$

Now this is an interesting result! It tells us that, if we assume neuron s_n to have a constant activation of +1, we may as well delete it. The price we have to pay for this reduction in network size is the introduction of a bias term that was not present before.

To see this clearly, we note that the right hand side of (16) can be seen as a quadratic form over $n-1$ dimensional vectors plus an inner product of $n-1$ dimensional vectors. Hence, if we consider a reduced size state vector

$$\boldsymbol{s}' \in \{\pm 1\}^{n-1} \tag{17}$$

together with a reduced size $(n-1) \times (n-1)$ weight matrix and introduce an $n-1$ dimensional bias vector

$$\boldsymbol{W}' = \begin{bmatrix} W_{11} & \cdots & W_{1n-1} \\ \vdots & \ddots & \vdots \\ W_{n-11} & \cdots & W_{n-1n-1} \end{bmatrix} \quad \boldsymbol{\theta}' = -2 \begin{bmatrix} W_{n1} \\ \vdots \\ W_{nn-1} \end{bmatrix} \tag{18}$$

we find that pre-assigning $\boldsymbol{x}_n$ to cluster $\mathcal{X}_+$ or, equivalently, forcing neuron $s_n = +1$ is equivalent to solving a slightly different Hopfield energy minimization problem, namely

$$\boldsymbol{s}'_* = \underset{\boldsymbol{s}' \in \{\pm 1\}^{n-1}}{\operatorname{argmin}} \; -\tfrac{1}{2}\, \boldsymbol{s}'^\top \boldsymbol{W}'\, \boldsymbol{s}' + \boldsymbol{\theta}'^\top \boldsymbol{s}' \tag{19}$$

3 PRACTICE

Having derived equation (19), we now discuss implementations of Hopfield nets for unambiguous bipartition clustering. This will be brief, because the heavy lifting w.r.t. *NumPy* / *SciPy* code for clustering Hopfield nets was already done in [1, 2]. Just as we did there, we require

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

for our following code snippets to work.

Throughout, we assume that we are given a sample of n data points $\boldsymbol{x}_i \in \mathbb{R}^m$ gathered in a data matrix $\boldsymbol{X}$ which we represent as a 2D *NumPy* array `matX`. The shape of this array is

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

If we are, for example, working with Gaussian kernels, we may initialize the $n \times n$ weight matrix $\boldsymbol{W}$ in (2) using the two functions `computeGaussKernelMatrix` and `centerKernelMatrix` in Listing 1 which we already discussed in [1]

```
matK  = computeGaussKernelMatrix(matX, 0.5)
matKc = centerKernelMatrix(matK)
matW  = 2 * matKc
np.fill_diagonal(matW, 0)
```

Given this *NumPy* implementation of $\boldsymbol{W}$, we can next compute arrays which represent matrix $\boldsymbol{W}'$ and vector $\boldsymbol{\theta}'$ in (18)

```
matWp = matW[:-1,:-1]
vecTp = -2 * matW[-1,:-1]
```

Listing 1: functions for kernel matrices

```
def computeGaussKernelMatrix(matX, sigma=1.):
    matD = spt.distance.pdist(matX.T, 'sqeuclidean')
    matD = spt.distance.squareform(matD)
    return np.exp(-0.5 / sigma**2 * matD)


def centerKernelMatrix(matK):
    _, n = matK.shape
    rsum = np.sum(matK,axis=0).reshape(1,n)
    csum = rsum.reshape(n,1)
    tsum = np.sum(rsum)
    return matK - 1/n * csum - 1/n * rsum + 1/n**2 * tsum
```

Listing 2: greedily running a clustering Hopfield net

```
def signum(x):
    return np.where(x >= 0, +1, -1)


def hnetRunGreedy(vecS, matW, vecT, tmax=500):
    for t in range(tmax):
        dltE = vecS * (matW @ vecS - vecT)
        updt = np.argmin(dltE)

        vecS[updt] = vecS[updt] * signum(dltE[updt])

    return vecS
```

Having initialized our network's weight s and biases, we next set its initial state s'. For the example in Fig. 2, we used

```
vecSp = -np.ones(n-1)
```

whereas for the examples in Fig. 3, we went with a random initialization scheme we already discussed in [2], namely

```
vecSp = 2 * rnd.binomial(n=1,p=0.5,size=n-1) - 1
```

Given arrays `matWp`, `vecTp`, and `vecSp`, we can now run our Hopfield net for unambiguous (kernel) bipartition clustering. Just as we did in [2], we will consider an informed update mechanism [11], where, in each cycle, we greedily select the updating neuron as that neuron whose update would decrease the network's current energy the most. We thus invoke `hnetRunGreedy` in Listing 2

```
vecSp = hnetRunGreedy(vecSp, matWp, vecTp)
```

and note that parameter `tmax` may have to be set to higher values, if the number n of data points is large.

Once this computation has terminated, we can finally partition matrix $\boldsymbol{X}$ into two smaller matrices $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ which represent the two sought after clusters $\mathcal{X}_+$ and $\mathcal{X}_-$.

Note: Data point $\boldsymbol{x}_n$ corresponds to the last column `matX[:,-1]` of array `matX`. Since we assume $\boldsymbol{x}_n$ to be in cluster $\mathcal{X}_+$, matrices $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ are computed as

```
mask  = np.hstack((vecSp>0, True))
matX1 = matX[:, mask]
matX2 = matX[:,~mask]
```

4 CONCLUSION

In this note, we fulfilled a promise made in [2] and showed how to rewrite the energy function of a Hopfield net for bipartition clustering such that its minimizer is unique and unambiguous.

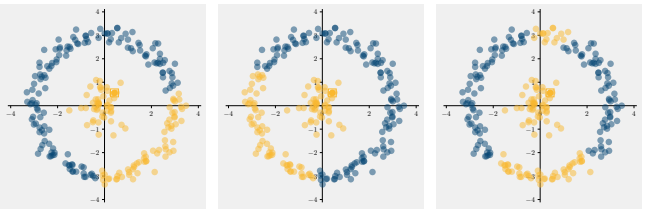

Figure 3: While the optimal solution to the problem in (19) is unique, there is no guarantee that a corresponding Hopfield net will find it. Results will always depend on the network's initial state and update mechanism.

Practical examples showed our ideas to be viable. However, we feel the need to emphasize the following: While the rewritten quadratic unconstrained binary optimization problem in (19) has a unique global minimizer, there is no guarantee that a corresponding Hopfield net will settle in the corresponding state (see Fig. 3). Rather, the performance of our clustering Hopfield nets will always depend on their initial state and update mechanism.

This is a problem where emerging computing paradigm promise a solution. For instance, because of the close (conceptual) connection between Hopfield nets and quantum computing [6], the discrete state space energy minimization problem we discussed in this note can also be solved on quantum devices [3–5] or on specifically designed digital annealers [9, 10]. By their very nature, such machines stand a much better chance to find globally optimal solutions to combinatorial optimization problems. Details as to why and how that is will be discussed in upcoming coding nuggets.

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, F. Beaumont, and S. Müller. 2021. *ML2R Coding Nuggets: Hopfield Nets, Bipartition Clustering, and the Kernel Trick.* Technical Report. MLAI, University of Bonn.
[2] C. Bauckhage, F. Beaumont, and S. Müller. 2021. *ML2R Coding Nuggets: Hopfield Nets for Bipartition Clustering.* Technical Report. MLAI, University of Bonn.
[3] 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. EMMCVPR.* Springer.
[4] C. Bauckhage, C. Ojeda, R. Sifa, and S. Wrobel. 2018. Adiabatic Quantum Computing for Kernel k=2 Means Clustering. In *Proc. KDML-LWDA.*
[5] C. Bauckhage, N. Piatkowske, R. Sifa, D. Hecker, and S. Wrobel. 2019. A QUBO Formulation of the k-Medoids Problem. In *Proc. KDML-LWDA.*
[6] C. Bauckhage, R. Sanchez, and R. Sifa. 2020. Problem Solving with Hopfield Networks and Adiabatic Quantum Computing. In *Proc. IJCNN.* IEEE.
[7] C. Bauckhage and P. Welke. 2021. *ML2R Theory Nuggets: Centering Data- and Kernel Matrices.* Technical Report. MLAI, University of Bonn.
[8] J. Hopfield. 1982. Neural Networks and Physical Systems with Collective Computational Abilities. *PNAS* 79, 8 (1982).
[9] S. Mücke, N. Piatkowski, and K. Morik. 2019. Hardware Acceleration of Machine Learning Beyond Linear Algebra. In *Proc. ECML/PKDD.*
[10] S. Mücke, N. Piatkowski, and K. Morik. 2019. Learning Bit by Bit: Extracting the Essence of Machine Learning. In *Proc. KDML-LWDA.*
[11] 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).