We once again revisit the idea of running Hopfield nets to cluster a set of data points 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 $x_i \in \mathbb{R}^m$.

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

$$s_* = \arg\min_{s \in \{ \pm 1 \}^n} -\frac{1}{2} s^\top W s$$

(1)

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

$$W_{ij} = \begin{cases} 0 & \text{if } i = j \\ 2[J K J]_{ij} & \text{otherwise} \end{cases}$$

(2)

Looking at this definition, we recall that $J K J$ is a centered kernel matrix. That is, $J = I - \frac{1}{n} 11^\top$ is a centering matrix [7] and the entries of the kernel matrix $K$ amount to

$$K_{ij} = K(x_i, x_j)$$

Here, the function $K : \mathbb{R}^m \times \mathbb{R}^m \rightarrow \mathbb{R}$ 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 ($W = W^\top$) and hollow (diag[$W$] = 0), we know that our Hopfield net will, in the long run, settle in a state $s^\infty$ of (locally) minimal energy, if its neurons update asynchronously [8]. With respect to our clustering problem, we can then regard the entries $s_i^\infty$ of $s^\infty$ as cluster labels or cluster membership indicators and partition our data into

$$\mathcal{X}_+ = \{ x_i \in \mathcal{X} \mid s_i^\infty = 1 \}$$

(4)

$$\mathcal{X}_- = \{ x_i \in \mathcal{X} \mid s_i^\infty = -1 \}$$

(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 $\theta^\top s$, it has two optimal solutions. Namely, if network state $s_*$ is an optimal solution to (1), then so is $-s_*$, because

$$(-s_*)^\top W (-s_*) = (-1)^2 \cdot s_*^\top W s_* = s_*^\top W s_*$$

(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.
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 \( x_n \) and that it is assigned to cluster \( X_+ \) (see the example in Fig. 2). This way, the optimal solution to our clustering problem is to put all points that cluster with \( x_n \) into cluster \( X_+ \) and all other points into cluster \( X_- \).

From the point of view of a Hopfield net that is supposed to find this optimal partition, our choice of pre-assigning data point \( x_n \) to cluster \( X_+ \) 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

\[
s^T W s = \sum_{i=1}^{n} \sum_{j=1}^{n} s_i W_{ij} s_j
\]

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

\[
\sum_{i=1}^{n} s_i \sum_{j=1}^{n-1} W_{ij} s_j = \sum_{i=1}^{n-1} s_i \sum_{j=1}^{n-1} W_{ij} s_j + s_n \sum_{j=1}^{n-1} W_{nj} s_j
\]

where we again used that \( s_n = 1 \). For the second sum, we find

\[
\sum_{i=1}^{n} s_i W_{in} = \sum_{i=1}^{n-1} s_i W_{in} + s_n W_{nn} = \sum_{i=1}^{n-1} s_i W_{in}
\]

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

\[
s^T W s = \sum_{i=1}^{n-1} \sum_{j=1}^{n-1} s_i W_{ij} s_j + \sum_{j=1}^{n-1} W_{nj} s_j + \sum_{i=1}^{n-1} s_i W_{in}
\]

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

\[
\sum_{j=1}^{n-1} W_{nj} s_j = \sum_{i=1}^{n-1} s_i W_{in}
\]

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

\[
s^T W s = \sum_{i=1}^{n-1} \sum_{j=1}^{n-1} s_i W_{ij} s_j + 2 \sum_{j=1}^{n-1} W_{nj} s_j
\]

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

\[
s' \in \{\pm 1\}^{n-1}
\]

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

\[
W' = \begin{bmatrix} W_{11} & \cdots & W_{1n-1} \\ \vdots & \ddots & \vdots \\ W_{n-11} & \cdots & W_{n-1n-1} \end{bmatrix}, \quad \theta' = -2 \begin{bmatrix} W_{1n} \\ \vdots \\ W_{n-1n} \end{bmatrix}
\]

we find that pre-assigning \( x_n \) to cluster \( X_- \), or, equivalently, forcing neuron \( s_n = 1 \) is equivalent to solving a slightly different Hopfield energy minimization problem, namely

\[
s'_n = \arg \min_{s' \in \{\pm 1\}^{n-1}} -\frac{1}{2} s'^T W' s' + \theta'^T s'
\]

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

```python
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 \( x_i \in \mathbb{R}^m \) gathered in a data matrix \( X \) which we represent as a 2D NumPy array \( \text{matX} \). The shape of this array is

\[ m, n = \text{matX.shape} \]

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

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

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

```python
matWp = matW[:1,:-1]
vecTp = -2 * matW[-1,:-1]
```
Unambiguous Bipartition Clustering with Hopfield Nets

Listing 1: functions for kernel matrices

```python
def computeGaussKernelMatrix(matX, sigma=1.):
    matD = sp.dist(matX, metric='euclidean')
    matD = sp.squareform(matD)
    return np.exp(-0.5 / sigma**2 * matD)

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

Listing 2: greedily running a clustering Hopfield net

```python
def hnetRunGreedy(vecS, matW, vecT, tmax=500):
    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.argmax(dltE)
            vecS[updt] = vecS[updt] * signum(dltE[updt])
        return vecS
```

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

```python
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

```python
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

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

and note that parameter $t_{\text{max}}$ 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 $X$ into two smaller matrices $X_1$ and $X_2$ which represent the two sought after clusters $X^*$ and $X_2$.

**Note:** Data point $x_n$ corresponds to the last column of $matX[\cdot,-1]$ of array $matX$. Since we assume $x_n$ to be in cluster $X^*$, matrices $X_1$ and $X_2$ are computed as

```python
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.