Chapter 7: Supervised Hebbian Learning

Brandon Morgan

1/13/2021

E7.1

The input vectors, p_i, will be inputted by column, where an empty tile is represented as a -1 and a filled tile as 1. Note, by encoding the entry values as $[-1, 1]$, we will have to use the *hardlims* transition function as its output corresponds to $[-1, 1]$. One could use binary responses $[0, 1]$, then the *hardlim* transition function would be used instead (see P7.7 and E7.4 for an example). Thus, $p_1^t = [-1, -1, 1, 1]$ and $p_2^t = [1, 1, -1, 1]$; thus, $P = [p_1, p_2]$:

$$P = \begin{bmatrix} -1 & 1 \\ -1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix}$$

1

Two vectors are othorgonal if their dot product is zero, $p_1^t p_2 = 0$:

```
p1 = matrix(c(-1, -1, 1, 1), ncol=1)
p2 = matrix(c(1, 1, -1, 1), ncol=1)
t(p1)%*%p2
```

```
##      [,1]
## [1,]   -2
```

As we can see, the dot product is not zero; therefore our vectors are not orthogonal.

2

When using an *autoassociator* network for Hebb's Rule, $W = TP^t$, the expected output vector T is set to the input vector P, thus $T = P$, to get $W = PP^t$

Here we get the following weights:

```
P = matrix(c(p1, p2), ncol=2)
W=P%*%t(P)
W
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2    2   -2    0
## [2,]    2    2   -2    0
## [3,]   -2   -2    2    0
## [4,]    0    0    0    2
```

3

We can convert our new input into the following vector: $p_t^t = [1, 1, 1, 1]$. Now we can test our new input pattern by $a = hardlims(Wp_t)$:

```
pt = matrix(c(1, 1, 1, 1), ncol=1)
W%*%pt
```

```
##      [,1]
## [1,]    2
## [2,]    2
## [3,]   -2
## [4,]    2
```

The hardlims function states that every entry of $a = (Wp_t)_i < 0$ is assigned to -1 and $a = (Wp_t)_i \geq 0$ is assigned to 1. Thus, we will get the following output vectors $a = [1, 1, -1, 1]$:

```
matrix(c(1,1, -1, 1), ncol=1)
```

```
##      [,1]
## [1,]    1
## [2,]    1
## [3,]   -1
## [4,]    1
```

This output matches input p_2 by suprise. Simple Hebb's rule does not gurantee that the output will match an input vector if the inputs are not orthogonal; however, in this case it yielded an input vector. From equation (7.14), if we were to normalize our input vectors the error would be equal to $\Sigma_{q \neq k} t_q(p_q^t p_k)$.

The error can be measured by Hamming distance (Ch. 3). This is calculated by taking the difference or adding, depending upon if *hardlims* or *hardlim* was used, of the output and the input vectors. In our case, p_t has a Hamming distance of 1 from p_2 and 2 from p_1; therefore, our perceptron correctly predicted the classification.