Chapter 7: Supervised Hebbian Learning

Brandon Morgan

1/13/2021

E7.2

In problem E7.1, our output matched input p_2, despite not having orthogonal input vectors. Simple Hebb's rule does not gurantee that the output vectors will match an input vector if they are not orthogonal. In response, we can fix this problem by using the Pseudoinverse Rule.

In simple Hebb's rule, $W = TP$, but by using the Pseudoinverse Rule we will instead compute $W = TP^{+}$, where P^{+} is the Moore-Penrose pseudoinverse given by $P^{+} = (P^{t}P)^{-1}P^{t}$ for when the number of rows of P is greater than the number of columns of P:

Before, our P matrix was given by:

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

Now, our P^{+} is calculated to be:

```
p1 = matrix(c(-1, -1, 1, 1), ncol=1)
p2 = matrix(c(1, 1, -1, 1), ncol=1)
P = matrix(c(p1, p2), ncol=2)
P
```

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

```
pseudo = solve(t(P)%*%P)%*%t(P)
pseudo
```

```
##            [,1]       [,2]       [,3] [,4]
## [1,] -0.1666667 -0.1666667  0.1666667  0.5
## [2,]  0.1666667  0.1666667 -0.1666667  0.5
```

Now, our weights are calculated by $W = TP^{+}$:

```
W=P%*%pseudo
W
```

```
##            [,1]       [,2]       [,3] [,4]
## [1,]  0.3333333  0.3333333 -0.3333333    0
## [2,]  0.3333333  0.3333333 -0.3333333    0
## [3,] -0.3333333 -0.3333333  0.3333333    0
## [4,]  0.0000000  0.0000000  0.0000000    1
```

Now, we can test our new input pattern $p_t^t = [1, 1, 1, 1]$:

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

```
##            [,1]
## [1,]  0.3333333
## [2,]  0.3333333
## [3,] -0.3333333
## [4,]  1.0000000
```

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, our output vector would be $a = [1, 1, -1, 1] = p_2$, which is the same as predicted in E7.1 .

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.