**Abstract**

One class of problems that may be solved with a neural network is the problem of pattern recognition. Additionally, in practical applications noise may be introduced which alters the input pattern slightly.

In this project, matrix representations of greyscale numbers and letters represent input patterns, and a 4-element to 5-element vector represents the output produced for the given input pattern. Two networks were trained with supervised learning, one trained with the Hebb rule and the other trained with the pseudoinverse rule.

The Hebb rule was purposed by Donald Hebb as a possible mechanism for synaptic modification in the brain and is currently used as a law for neural networks. For the supervised Hebb rule we can tell the algorithm what the network should do, instead of what it is doing in that moment by substituting target outputs for actual outputs. This can be represented by the equation Wnew=Wold+T\*P (T being our taget vector and P being our input). This Wnew can then be multiplied with P to produce our desired result from a trained neuron. However, the Hebb rule produce some error if the prototype vectors are not orthogonal.

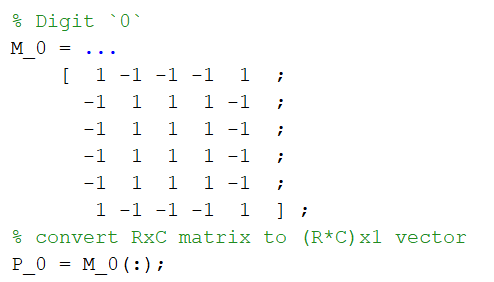
To reduce error the pseudoinverse rule can be used. The pseudoinverse rule uses the Moore-Penrose pseudoinverse and can be used to find the weight matrix that minimizes the error. The Moore-Penrose pseudoinverse can be represented by the equation P+= (PTP)-1PT where the number of rows of P is greater than the number of columns of P and the columns are independent. Using pseudoinverse rule the weight matrix is calculate by W=TP+ or W=T(PTP)-1PT. This method of training will produce the exact outputs while the Hebb rule produces outputs that are close, reducing error.

The autoassociative network can handle noisy or slightly corrupt versions of the prototype pattern and still produce a correctly recovered patterns. This is important because in practical application the network may be subject to interference and this must not stop it from producing desired results.

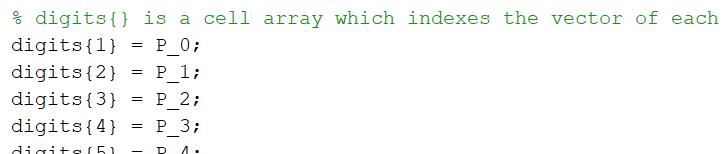
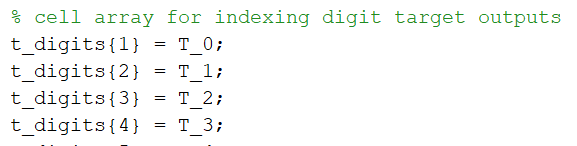
**Method**

First, a training phase occurs with the following algorithm:

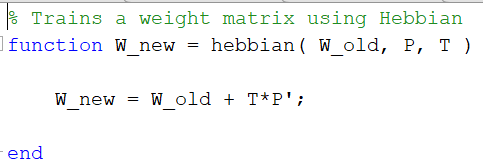
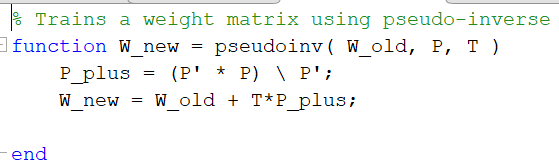
1. Each image is constructed as a 6x5 matrix and then converted to a 30x1 vector **P.**



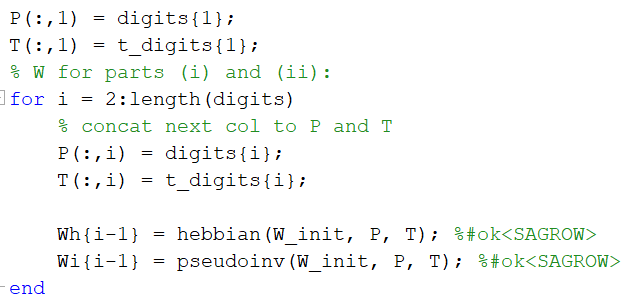
1. Each input vector **P** and corresponding target output vector **T** are stored into cell arrays

1. A weight matrix **W** is calculated using the Hebb and pseudoinverse rules:

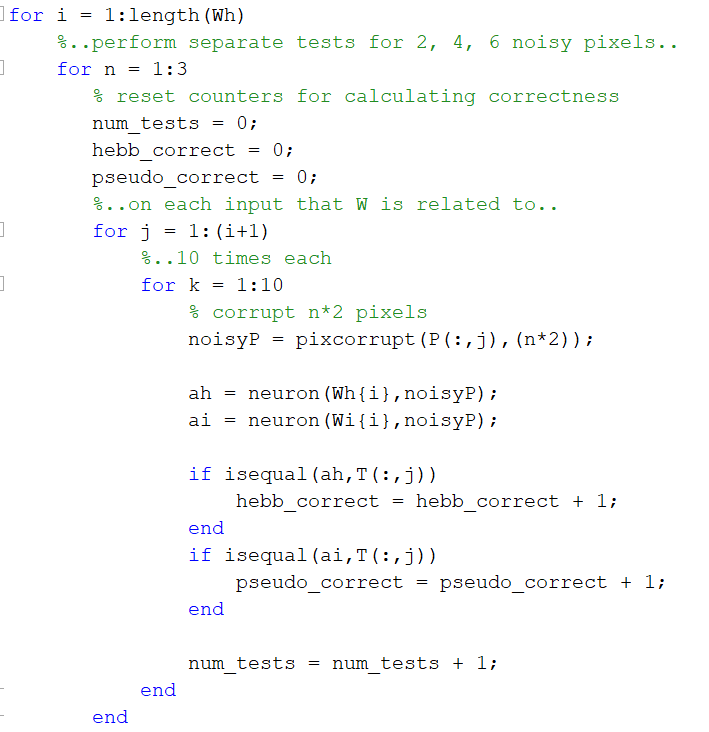
 

* 1. Using the first two (**P**, **T**) pairs, calculate a weight matrix **W** and store the result into a cell array
  2. Repeat part a. using the first three, four, … pairs until all (**P**, **T**) pairs have been used

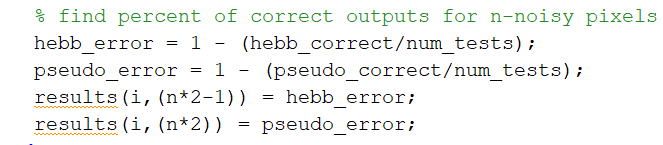


Next, a testing phase occurs

1. For each weight matrix **W**
   1. Corrupt 2 noisy pixels for each input vector **P** related to the current weight matrix **W**
   2. Send the resulting noisy input vector **P\_noisy** as input to each neural network
   3. Perform the above 2 steps 10 times, counting the number of times the network correctly identifies **P\_noisy**



1. Calculate the percent error for each network over the 10 tests and store the result in a cell array



1. Repeat steps 1 and 2 for 4 and 6 noisy pixels

**Result**

6x5 digit inputs

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| W | H(2) | P(2) | H(4) | P(4) | H(6) | P(6) |
| 0-1 => | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 0-2 => | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 0-3 => | 0.000 | 0.000 | 0.000 | 0.000 | 0.100 | 0.100 |
| 0-4 => | 0.020 | 0.000 | 0.040 | 0.020 | 0.160 | 0.120 |
| 0-5 => | 0.217 | 0.000 | 0.333 | 0.100 | 0.400 | 0.217 |
| 0-6 => | 0.171 | 0.000 | 0.229 | 0.086 | 0.414 | 0.171 |
| 0-7 => | 0.262 | 0.037 | 0.363 | 0.150 | 0.412 | 0.250 |
| 0-8 => | 0.422 | 0.056 | 0.400 | 0.267 | 0.544 | 0.356 |
| 0-9 => | 0.550 | 0.070 | 0.590 | 0.360 | 0.590 | 0.450 |

6x6 digit inputs

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| W | H(2) | P(2) | H(4) | P(4) | H(6) | P(6) |
| 0-1 => | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 0-2 => | 0.000 | 0.000 | 0.033 | 0.000 | 0.000 | 0.000 |
| 0-3 => | 0.000 | 0.000 | 0.050 | 0.000 | 0.075 | 0.025 |
| 0-4 => | 0.020 | 0.000 | 0.080 | 0.020 | 0.140 | 0.040 |
| 0-5 => | 0.350 | 0.000 | 0.450 | 0.050 | 0.350 | 0.033 |
| 0-6 => | 0.086 | 0.000 | 0.086 | 0.014 | 0.214 | 0.086 |
| 0-7 => | 0.275 | 0.025 | 0.350 | 0.075 | 0.363 | 0.113 |
| 0-8 => | 0.356 | 0.011 | 0.411 | 0.244 | 0.422 | 0.256 |
| 0-9 => | 0.590 | 0.250 | 0.620 | 0.340 | 0.690 | 0.540 |

9x9 digit inputs

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| W | H(2) | P(2) | H(4) | P(4) | H(6) | P(6) |
| 0-1 => | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 0-2 => | 0.000 | 0.000 | 0.000 | 0.000 | 0.033 | 0.000 |
| 0-3 => | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 0-4 => | 0.200 | 0.000 | 0.280 | 0.000 | 0.300 | 0.020 |
| 0-5 => | 0.500 | 0.000 | 0.500 | 0.000 | 0.533 | 0.000 |
| 0-6 => | 0.143 | 0.000 | 0.129 | 0.000 | 0.186 | 0.000 |
| 0-7 => | 0.287 | 0.000 | 0.262 | 0.000 | 0.287 | 0.000 |
| 0-8 => | 0.500 | 0.000 | 0.467 | 0.000 | 0.433 | 0.022 |
| 0-9 => | 0.620 | 0.000 | 0.640 | 0.010 | 0.650 | 0.060 |

6x5 initials inputs

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| W | H(2) | P(2) | H(4) | P(4) | H(6) | P(6) |
| 01 => | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 02 => | 0.533 | 0.100 | 0.400 | 0.167 | 0.500 | 0.267 |
| 03 => | 0.500 | 0.175 | 0.450 | 0.200 | 0.525 | 0.500 |
| 04 => | 0.280 | 0.040 | 0.360 | 0.180 | 0.400 | 0.220 |
| 05 => | 0.233 | 0.133 | 0.267 | 0.150 | 0.367 | 0.317 |
| 06 => | 0.471 | 0.029 | 0.414 | 0.143 | 0.471 | 0.300 |

**Conclusion**

From the results above, one can observe that the pseudoinverse network produced correct output much more frequently than the Hebb network did in all cases. Another observation is that both networks tend to produce more incorrect results as the number of corrupted pixels and number of input vectors trained with the weight matrix increased.

An interesting exception to this observation occurs when the digits 0-5 and 0-6 are tested. In the first 3 tests, both networks produced more incorrect output when the weight matrix trained with the digits 0-5 was used, compared to the output produced with the weight matrix trained with the digits 0-6.

The results obtained from the 6x5 and 6x6 digit inputs are mostly similar. The 6x6 digit input performed worse in the 0-9 input tests but performed better than the 6x5 digit input for the 0-9 input tests. However, comparing both 6x5 and 6x6 digit inputs to the 9x9 digit input case, one can observe drastic performance increase for the pseudoinverse network. And when considering the 6x5 initials inputs all 3 digits inputs performed better than the initials inputs. This may be because of more exotic patterns making it harder for the network to produced desired pattern when introducing more inputs and noise.

A reason why the pseudoinverse network performed better than the Hebb network is that the set of input vectors are not orthonormal. Due to this, the Hebb network will produce some error as defined by the following performance index:

On the other hand, the pseudoinverse method produces the weight matrix which minimizes this error.

**References**

Neural network design, M. T. Hagan, H. W. Demuth, and M. H. Beale, University of Colorado Bookstore, 2002