UE16EC347

Machine Learning

Project Report on

MATRIX INVERSION USING RECURRENT NEURAL NETWORK

by

Arun Gaonkar (01FB16EEC048)

Karthik P Bilichod (01FB16EEC125)

Kartik Vishnu Hegde (01FB16EEC126)

Under the guidance of

Dr. KOSHY K GEORGE

23 Apr 2019

Abstract :

Matrix inversion has been widely used in a variety of fields such as robotics, control, signal processing ,etc. For many large-scale problems, the orders of the matrices are very large and large-scale inverse matrices often need to be computed in real-time for monitoring and controlling dynamic systems. Large matrix inversion using existing algorithms in real-time is usually not efficient due to the nature of the sequential processing. For such applications, parallel distributed processing is more desirable.

The proposed recurrent neural networks can operate concurrently in real time. It is the nature of parallel and distributed processing that renders the neural network approach the computational advantages over the existing sequential algorithms in real-time applications.

Contents

Chapter 1

Introduction

The proposed recurrent neural network is the modified Hopfield neural network. While studying the stability of Hopfield network , we modify the weights in the state equation so that we can exploit this property to obtain Inverse of the input matrix at the output of neural network.

Some properties of the Hopfield network are listed below :

- Provides Multiple-loop-feedback system with delays.
- Architecture of Hopfield follows noiseless,dynamical, Additive-model of a neuron
- Uses nonlinear activation functions, generally a monotonically increasing in both standard and inverse input-output relations. (e.g., sigmoid)
- The Energy function is a Lyapunov function which is stable in accordance with Theorem 1.

The proposed recurrent neural network can be decomposed into n independent subnetworks and can be easily implemented in an electronic circuit. The proposed recurrent neural network is also proven to be asymptotically stable in large and capable of generating inverse matrices within a few characteristic time constants. It can also be implemented in discretized form.

Chapter 2

Proposed Model

The proposed Modified Hopfield Neural Network is as follows :

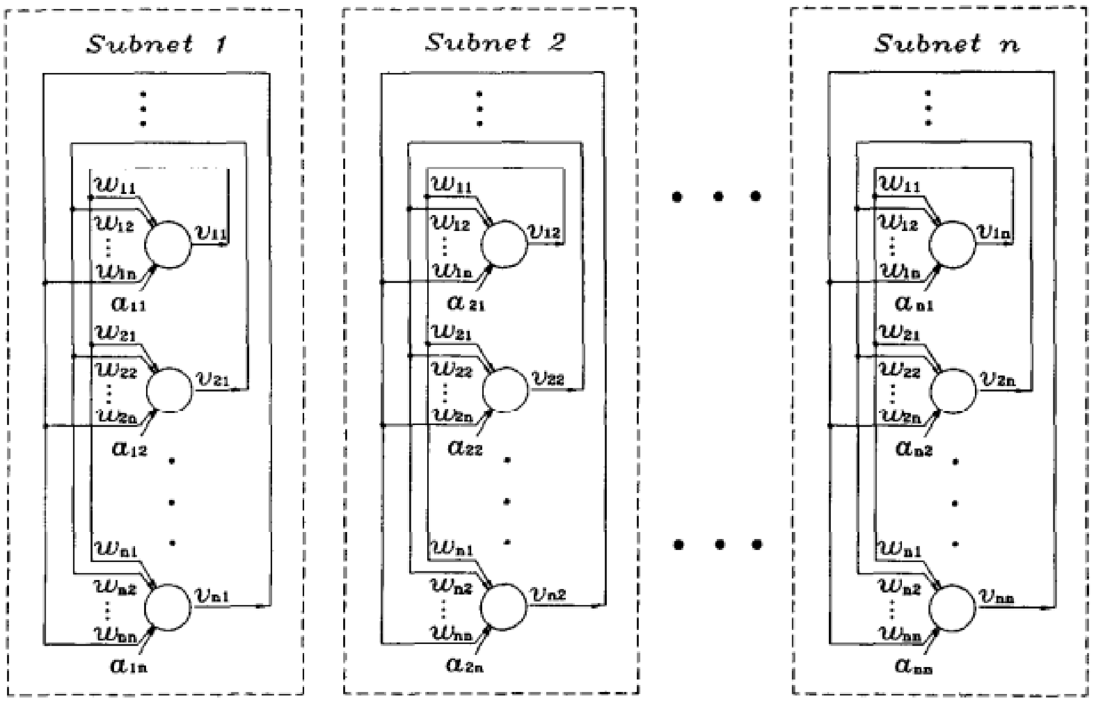


Figure 2.1: Architecture of network

The output state equation of the above model is given by :

$$\frac{dV(t)}{dt} = -\eta A^{\top} A V(t) + \eta A^{\top} \tag{2.1}$$

- **A** is the input Matrix.
- **V** is the output Matrix which varies with time till it reaches the steady state, i.e. when the output is no more changing with time.
- η is the positive scaling parameter.

The Equilibrium state of output is defined as no change in output state after the output has settled to the value, i.e.

$$\frac{dV(t)}{dt} = 0$$

Thus, $A^T AV = A^T$

Since $A^T A$ is positive definite, $(A^T A)^{-1}$ exists.

Therefore, V = $(A^T A)^{-1} A^T$ = $A^{-1}(A^T)^{-1} A^T$ = A^{-1} I = A^{-1}

Chapter 3

Analyzing the network

To analyze the stability of the network, we have to obtain equation relating the input and output.Hence we can model the system either in *physical modelling* or in *electrical modelling.*

Here we go for Electrical modelling of the network.

3.1 Analog realization

We know that Hopfield network is a recurrent neural network and hence it contains summing junctions of inputs, Weights and Delay elements, as the network is dependent on previous outputs.

Hence to analyze the Recurrent Neural Network, We model it as an electronic circuit with :

- Op-Amp adder as summing junctions.
- Op-Amp feedback gain as weights.
- Integrators as delay elements.
- Bias voltages as inputs.

The Figure given below is a 2 x 2 matrix realization.

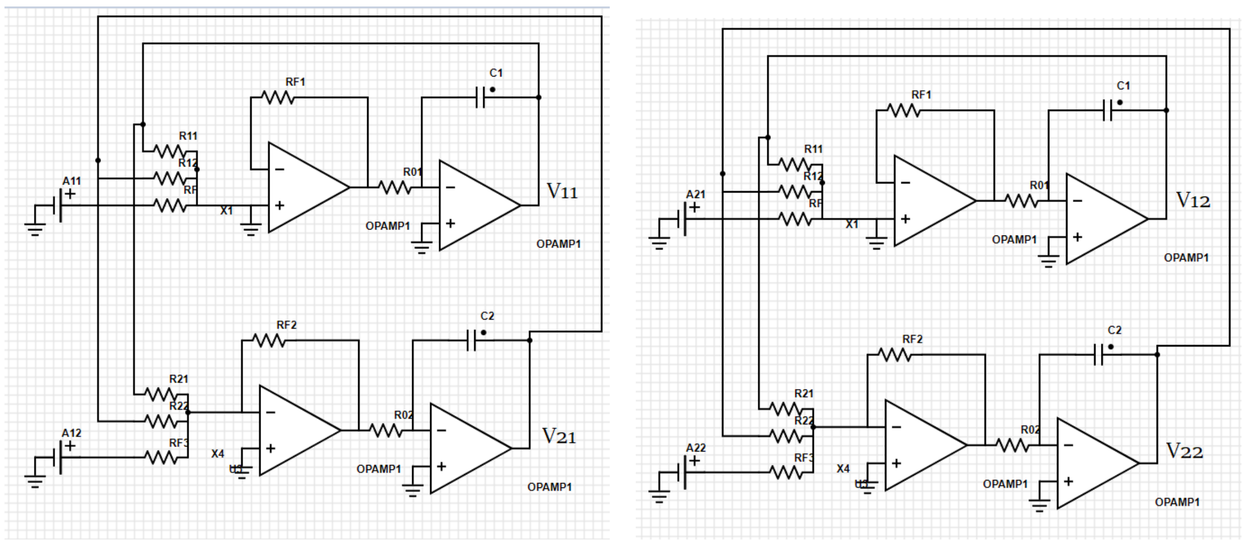


Figure 3.1: Subnet 1 and Subnet 2

3.2 Deriving the State Equation

From the Fig. 3.1, Let us first obtain state equation for V_{11} .
Let the output of the Op-Amp adder be V_0.
The output V_0 is given by :

$$V_0 = -\left(V_{11}.\frac{R_f}{R_{11}}\right) - \left(V_{12}.\frac{R_f}{R_{12}}\right) - A_{11} \tag{3.1}$$

Output of the Integrator is given by :

$$V_{11} = \left[\left(V_{11}.\frac{R_f}{R_{11}}\right) + \left(V_{12}.\frac{R_f}{R_{12}}\right) + A_{11}\right]\frac{1}{R_0CS} \tag{3.2}$$

Let $\frac{1}{R_0C} = \eta$

$$SV_{11} = \left[\left(V_{11}.\frac{R_f}{R_{11}}\right) + \left(V_{12}.\frac{R_f}{R_{12}}\right) + A_{11}\right] \tag{3.3}$$

V is always initialized with zeros i.e at time t=0, V=0.
Hence converting to time domain, the state equation becomes.

$$\frac{dV_{11}}{dt} = \eta\begin{bmatrix}\frac{R_f}{R_{11}} & \frac{R_f}{R_{12}}\end{bmatrix}\begin{bmatrix}V_{11}\\V_{21}\end{bmatrix} + A_{11} \tag{3.4}$$

Similarly state equations for V_{21},V_{12},V_{22} are given by :

$$\frac{dV_{21}}{dt} = \eta \begin{bmatrix} \frac{R_f}{R_{21}} & \frac{R_f}{R_{22}} \end{bmatrix} \begin{bmatrix} V_{11} \\ V_{21} \end{bmatrix} + A_{12} \tag{3.5}$$

$$\frac{dV_{12}}{dt} = \eta \begin{bmatrix} \frac{R_f}{R_{11}} & \frac{R_f}{R_{12}} \end{bmatrix} \begin{bmatrix} V_{12} \\ V_{22} \end{bmatrix} + A_{21} \tag{3.6}$$

$$\frac{dV_{22}}{dt} = \eta \begin{bmatrix} \frac{R_f}{R_{21}} & \frac{R_f}{R_{22}} \end{bmatrix} \begin{bmatrix} V_{11} \\ V_{21} \end{bmatrix} + A_{22} \tag{3.7}$$

On combining equations (3.4), (3.5), (3.6) and (3.7) and writing them in the Matrix form,

$$\frac{d}{dt} \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix} = \eta \begin{bmatrix} \frac{R_f}{R_{11}} & \frac{R_f}{R_{12}} \\ \frac{R_f}{R_{21}} & \frac{R_f}{R_{22}} \end{bmatrix} \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix} + \eta \begin{bmatrix} A_{11} & A_{21} \\ A_{12} & A_{22} \end{bmatrix} \tag{3.8}$$

On generalizing to any $n\ x\ n$ matrix , we obtain the state equation of model as :

$$\frac{dV}{dt} = \eta WV + \eta A^{\top} \tag{3.9}$$

where W is the weight matrix defined by gains of Op-Amp.

Chapter 4

Stability Analysis

4.1 Lyapunov's Theorems

Stability and asymptotic stability of a dynamic system can be determined using *Modern Stability Theory*, founded by **Lyapunov**.

THEOREM 1 : The equilibrium state $\overline{x}$ is stable if in a small neighbourhood of $\overline{x}$, there exist a positive definite function V(x) such that its derivative with respect to time is negative semi-definite in that region.

THEOREM 2 : The equilibrium state $\overline{x}$ is asymptotically stable if in a small neighbourhood of $\overline{x}$, there exist a positive definite function V(x) such that its derivative with respect to time is negative definite in that region.

According to **theorem 1**, the equilibrium state $\overline{x}$ is stable, if : $\frac{dV(x)}{dt} \leq 0$

According to **theorem 2** ,the equilibrium state is asymptotically stable, if :$\frac{dV(x)}{dt} < 0$

4.2 Obtaining the Inverse

According to Lyapunov's Stability criterion, vspace3mm
Equating $\frac{dv}{dt} = 0$,

we obtain :

$$\eta W V = -\eta A^T$$

W should be chosen such that we should get inverse at the output. Hence, equating V to A^{-1} we get ,

$$W A^{-1} = -A^T$$

Therefore $$W = -A^T A$$

4.3 Positive Definiteness of Weight matrix

If A is non-singular, then the equilibrium state of the recurrent neural network is asymptotically stable. According to theorem 2, $A^T A$ is positive definite.

If A is non-singular, then the equilibrium state matrix of the recurrent neural network represents the inverse of matrix.

Chapter 5

Matlab Implementation

5.1 Implementation

- The implementation was a continuous time form on the Op-Amp Analog circuit.
- Discretizing it will be helpful to implement the same on currently trending Digital Technology such as FPGA.
- Here the difference between previous output and present output at every iteration is checked.
- The iteration will be running till previous outputs are almost similar as present outputs i.e the error between each of the elements is around 10^{-8}.
- The parameter η value is varied and the outputs are checked manually to give the best results in least time and that value of η is finalised as the parameter for any inputs further.

5.2 Advantages

This way of finding inverse of a matrix is very **Hardware friendly** Algorithm. The traditional methods like LU decomposition for a huge matrices is difficult to implement on some of the trending digital technologies like **FPGA** implementation which is written in Hardware descriptive language. LU decomposition is cumbersome process and also does lot of memory wastage. But with the recurrent neural networks throughout the working process, constant memory is allocated.

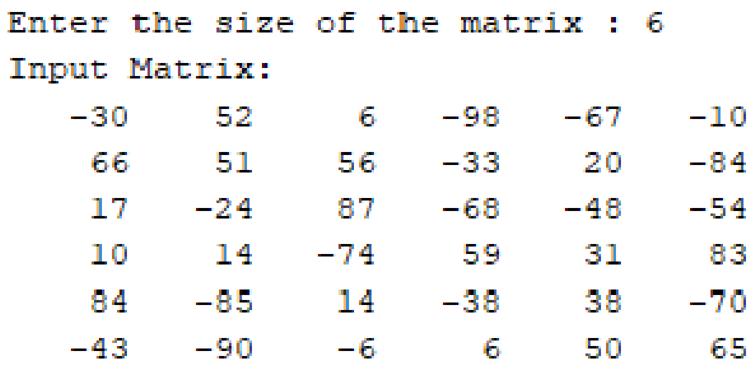
Also it can be implemented on hardware for any order of matrix as **dynamic memory allocation** is easy for this compared to LU decomposition.

Chapter 6

Results and Conclusion

6.1 Results

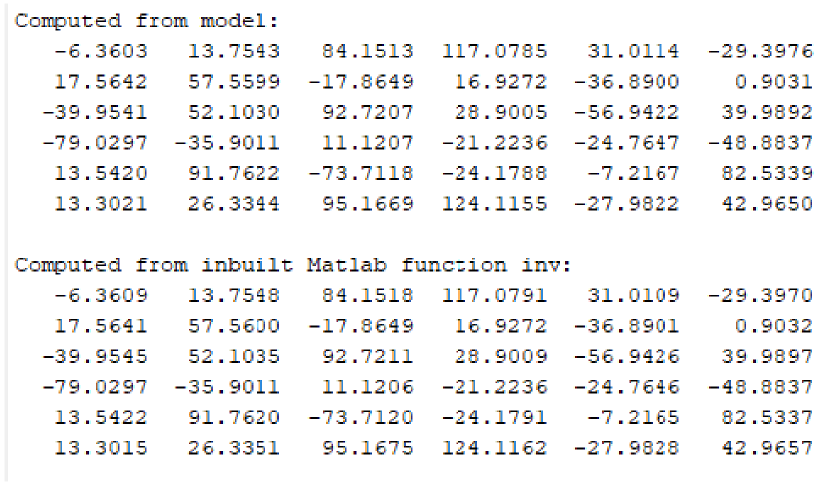
- **The input matrix**



```
Enter the size of the matrix : 6
Input Matrix:
   -30    52     6   -98   -67   -10
    66    51    56   -33    20   -84
    17   -24    87   -68   -48   -54
    10    14   -74    59    31    83
    84   -85    14   -38    38   -70
   -43   -90    -6     6    50    65
```

Figure 6.1: Input matrix of order 6x6

- **The output matrix (Inverse of the input matrix)**



```
Computed from model:
   -6.3603   13.7543   84.1513  117.0785   31.0114  -29.3976
   17.5642   57.5599  -17.8649   16.9272  -36.8900    0.9031
  -39.9541   52.1030   92.7207   28.9005  -56.9422   39.9892
  -79.0297  -35.9011   11.1207  -21.2236  -24.7647  -48.8837
   13.5420   91.7622  -73.7118  -24.1788   -7.2167   82.5339
   13.3021   26.3344   95.1669  124.1155  -27.9822   42.9650

Computed from inbuilt Matlab function inv:
   -6.3609   13.7548   84.1518  117.0791   31.0109  -29.3970
   17.5641   57.5600  -17.8649   16.9272  -36.8901    0.9032
  -39.9545   52.1035   92.7211   28.9009  -56.9426   39.9897
  -79.0297  -35.9011   11.1206  -21.2236  -24.7646  -48.8837
   13.5422   91.7620  -73.7120  -24.1791   -7.2165   82.5337
   13.3015   26.3351   95.1675  124.1162  -27.9828   42.9657
```

Figure 6.2: Comparison between calculated output and Matlab inbuilt function output

- **The Error matrix**

```
Error Matrix
   1.0e-03 *

  -0.5674    0.5668    0.5667    0.5669   -0.5671    0.5658
  -0.0412    0.0416    0.0415    0.0415   -0.0416    0.0420
  -0.4141    0.4142    0.4139    0.4141   -0.4143    0.4139
   0.0348   -0.0354   -0.0352   -0.0352    0.0353   -0.0359
   0.2117   -0.2106   -0.2108   -0.2108    0.2108   -0.2092
  -0.6619    0.6618    0.6615    0.6617   -0.6621    0.6611
```

Figure 6.3: Difference between calculated and Matlab inbuilt function outputs

- **Heatmap 1**

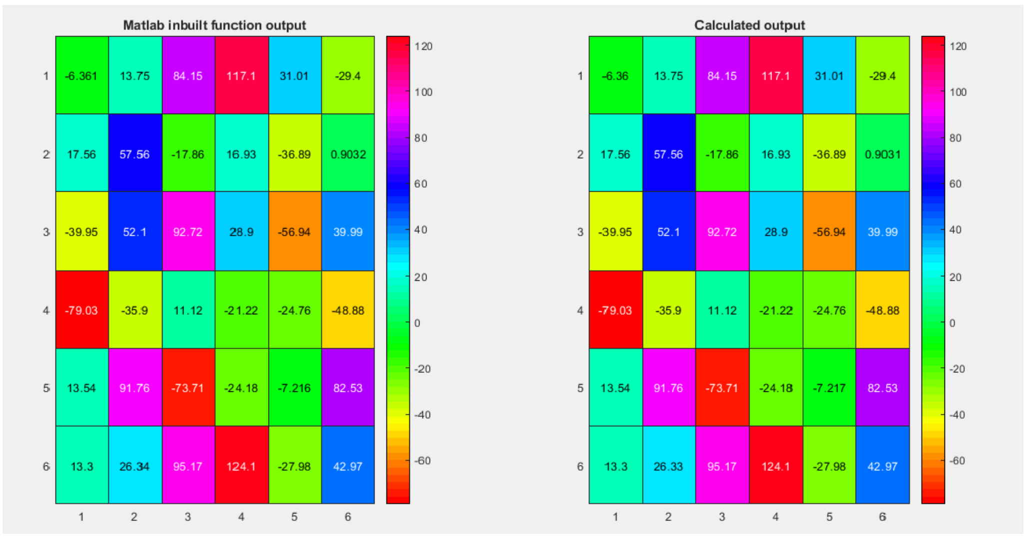


Figure 6.4: Heatmap for Comparison between calculated output and Matlab inbuilt function output

- **Heatmap 2**

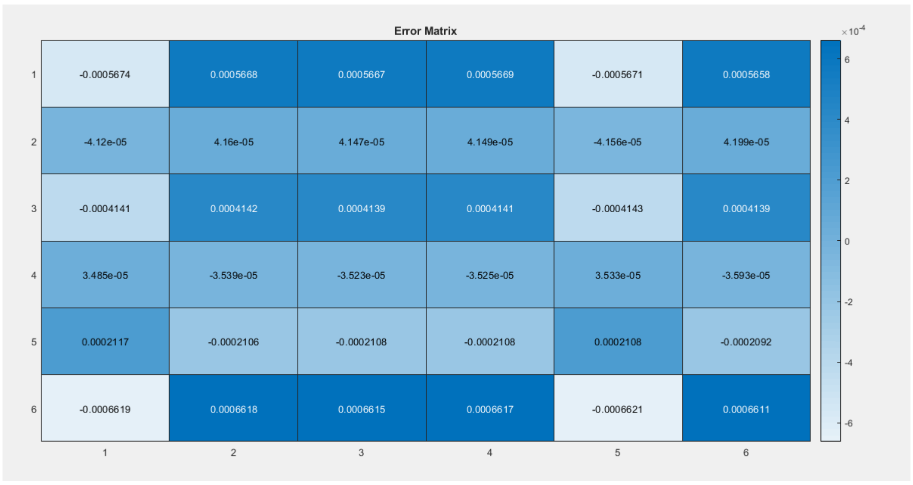


Figure 6.5: Heatmap for Error Matrix

6.2 Conclusion

The inbuilt Matlab command **inv** uses the method of LU Decomposition which is best method to be known to calculate inverse very precisely. From the results, we can see that the inverse obtained by neural network for parameter η=0.01 gave the best possible results to precision upto 4 digits after decimal. Hence this is good result for most of the Hardware applications. The time taken to settle to steady state depends on Eigen values for analog circuit but in Matlab, it depends on execution time of each steps which is little more delays compared to LU Decomposition method. But implementing this on advanced tools like FPGA will take lesser time to compute inverse of a matrix.

Chapter 7

References

1. **A Recurrent Neural Network for Real-Time Matrix Inversion*** by **Jun Wang**, Department of lndustrial Technology University of North Dakota.
2. **Neural Networks**- A comprehensive foundation, second edition by **Simon Haykin**, McMaster University, Hamilton, Canada.
3. **Neural Network Approach to Computing Matrix Inversion** by **Luo Fa-Long and Bao Zheng**, Institute of Electronic Engineering and Center for Neural Networks Xidian University Xian, China.
4. **J. Wang**, **Electronic realization of a recurrent neural network for solving simultaneous linear equations**.