A Bidirectional Associative Memory Algorithm of type Store-Recall and the corresponding C++ source Program with several Versions and Applications

Nicolae POPOVICIU, Floarea BAICU
Hyperion University of Bucharest, Faculty of Mathematics-Informatics
Street Călărașilor 169, Bucharest, ROMANIA
nicolae.popoviciu@yahoo.com

Abstract. The work three big parts. In the first part there is a short description of an artificial neural network related with the bidirectional associative memory (BAM) and an algorithm of type Hopfield. The algorithm is named ALGOHOPFIELDSeqStoreRecall and it belongs to the class of unsupervised learning. The second part contains a C++ source program of the above algorithm. The last part contains the numerical results and the print screen of exe program.

Key-words. Hopfield algorithm, BAM, bidirectional associative memory, C++ program, energy theorem, store-recall, store information, unsupervised learning, recall information, autocorrelation matrix, Hamming distance.

1. A BAM algorithm of type Hopfield.
A BAM algorithm of type Hopfield belongs to unsupervised learning. The Hopfield neural network is a special case of the BAM neural network.

1.1 Notations. The problem statement. [5]
For the unsupervised learning neural network we denote by \( I(x) \) the input set (the given set) of vectors \( x = x(t) \). This is the store information (stock information), where \( x \in R^n \), \( t = 1, N \); \( n, N \) are integer given numbers. The vector \( x \) has all the elements with the values 1 or −1.

We denote \( I(xr) \) the set of vectors \( xr = xt(t1) \in R^n \), \( t1 = 1, Nxr \), where \( Nxr \) is an integer given number. The vector \( xr \) also has all elements with the values 1 or −1.

The vector \( x \) contains the correct information (it is called the mother vector) and the vector \( xr \) contains an incorrect information (\( x \) is destroyed by the noise).

The Hopfield problem is: given a \( xr \) vector we have to find in the set \( I(x) \) the mother vector \( x \) of \( xr \).

\( xc \in R^n \) is the complementary vector of \( x \), where \( xc = x^*(-1) \).

We denote by \( D Hab \) the Hamming distance between the vectors \( a \) and \( b \). The relation between Hamming distance and Euclid distance \( DEab \) is

\( DEab = 2\sqrt{D Hab} \).

For the vectors \( xr \) of \( I(xr) \) there are two possibilities. Version 1: the vectors \( xr \) arrive in the neural network in the sequential mode. Version 2: the vectors \( xr \) arrive in the neural network in the batch mode. The program C++ works with version 1.

By \( W = WAC = (w_{ij}) \), \( WAC = WAC_{nXn} \) we denote the autocorrelation matrix. This is a square and symmetric matrix of type \( nXn \) having the form (given by Donald Hebb):

\( W = WAC = \sum_{t=1}^{N} x(t)x(t)^T \), \( T \) means transpose.

The autocorrelation matrix \( WAC \) contains the whole information from the input set \( I(x) \). The columns of matrix \( WAC \) are denoted by \( w_j \in R^n \), \( j = 1, n \)

Proposition 1. The diagonal elements of the matrix \( WAC \) are \( w_{i,i} = 0 \). (see Cohen and Grossberg and proof [5], page 198).

1.2 The energy theorem and the updating rule.
Any vector \( xr \) yields in neural network the energy
E(xr) = –xrTWxr, wij = 0.

By using the updating rule the components of vector xr are modified during the algorithm so that the energy function decreases. The updating rule is based on the value

\[ \text{net}xrw_j = w_j^T x_r \] (the scalar product).

The component \( x_{rj}(k) \) becomes \( x_{rj}(k+1) \)

\[
\begin{cases}
  +1 & \text{if } \text{net}xrw_j > 0 \\
  -1 & \text{if } \text{net}xrw_j < 0 \\
  x_{rj}(k) & \text{if } \text{net}xrw_j = 0
\end{cases}
\]

**Proposition 2.** \( E[xr(k)] > E[xr(k+1)] \).
(proof [5], page 200).

### 1.3 The algorithm stop condition.

The algorithm stops when the components of \( xr(k+1) \) are stationary i.e. \( xr(k) = xr(k+1) \). The C++ source program of ALGOHOPFIELDSeqStoreRecall works in the following way:

a. Compute the vector \( \text{teta} \in R^n \)
\[ \text{teta} = xr(k) - xr(k+1) \]

b. Compute the norm \( \| \text{teta} \|^2 \).

c. If \( \| \text{teta} \|^2 = 0 \) then STOP to improve the elements of \( xr \). Otherwise continue. The last \( xr \) is denoted \( *xr \). 

d. Look for the vector \( *xr \) in the set \( I(x) \).

e. Print the result: to the recall vector \( xr \) corresponds the store vector \( x(t) \), \( 1 \leq t \leq N \).

**Remark.** The C++ source program takes into account the possibility that the sequence \( xr(k) \), \( xr(k+1) \) repeats indefinitely (there is a cycling). In order to control the cycling we use the threshold \( kmax \) given by the user. When \( k > k \text{max} \) the algorithm stops and gives a cycling message. STOP algorithm.

### 2. The C++ source program

The C++ source program has many sections and it uses some C++ procedures (function).

#### 2.1 The program section.

Section 0. The variable description. (see 1.)

Section 1. The C++ files.

Section 2. The variable type declaration.

Section 3. The procedures: DistHamm, PrScalar, Normteta2, Energxr.

Section 4. C++ source program.

#### 2.2 The C++ source program.

// SECTION 1: C++ files
#include<iostream.h>
#include<conio.h>
#include<math.h>

// SECTION 2: Variable type declaration
int i,i1,j,k,kmax,k1,n,t,N,xr,x[20][50],xc[20][50],wij[20][20],Tip;
int vecta[20],vectb[20],teta[20],DHab,fictiv,colwij[20];
float DE,r,net,netxrwj,normteta,normteta2,energie,energiexr;
int xrkp1[20],xrlucru[20];

// SECTION 3: Procedures

// Procedure 1: DistHamm, Hamming distance between two vectors a and b
int DistHamm(int n,int vecta[],int vectb[],int DHab)
{ // Begin Procedure 1 DistHamm
  DHab=0;
  for(i=1;i<=n;i++)
    if(vecta[i]==vectb[i]) {DHab=DHab;}
    else{DHab=DHab+1;}
  return DHab;
} // End Procedure 1 DistHamm

// Procedure 2: PrScalar, Scalar product for two vectors netxtwj=<wj,xr>
float PrScalar(int n,int vecta[],int vectb[],float net)
{ // Begin Procedure 2 PrScalar
  net=0.0;
  for(i1=1;i1<=n;i1++)
    {net=net+vecta[i1]*vectb[i1];}
  return net; // The value of scalar product
} // End Procedure 2 PrScalar

// Procedure 3: Normteta2, Norma^2 for one vector netxtwj=<wj,xr>
float Normteta2(int n,int vecta[],float normteta2)
{ // Begin Procedure 3 Normteta2
  normteta2=0.0;
  for(i=1;i<=n;i++)
    {normteta2=normteta2+vecta[i]*vecta[i];}
  return normteta2;
} // End Procedure 3 Normteta2
// Procedure 4: Energxr, Energy theorem,
// E(xr)=xT*WAC*xr=xT*colwij

// First in Principal Program we multiply WAC by xr
// and we obtain the vector colwij[]
// Then we multiply vecta[]=xT[] by the column vector
// colwij[]

float Energxr(int n, int vecta[], int colwij[], float energie)
{ // Inceput Procedura 4 Energxr
  energie=0.0;
  for(i=1;i<=n;i++) {energie=energie+vecta[i]*colwij[i];}
  energie=benergie;
  return energie;
} // Sfarsit Procedura 4 Energxr

// Procedure 5 de TEST: ProdxxT x(t)*x(t)T=matrice n X n.Generates AutoCorrelation matrix
// int ProdxxT(int n, int vecta[], int wij[][])
// AutoCorrelatie WAC=suma t=1,N x(t)*x(t)T vecta=x(t)
// OBS. The procedure ProdxxT C++ do not ACCEPT
// the parameter MATRIX wij[][]
// This Procedure 5 is not used

// SECTION 4 C++ source program

void main()
{ // START PROGRAM
  // SECTION 5: Input data

cout<<" Input Nxr=<50 Nxr= nr of vectors xr(txr)
txr=1,Nxr Nxr= ",cin>>Nxr;

cout<<" Input kmax=<50 kmax=nr max of cycles
kmax= ",cin>>kmax;

// b) Vectorial data x(t)=1,N; xr(tr) txr=1,Nxr
// In PUT ALL (correct) vectors x(t)
cout<<endl;

cout<<" Vectors x(t)=1,N are : \n";

cout<<endl;
  // Construct the complementary set lc(xc)
  // xc=x*(b1). These vectors are not used in program
  // cout<<" The set of complementary vectors xc(t)
t=1,N is :\n";
  // for(t=1;t<=N;t++)
  // {cout<<" x["<<t<<" ]:\n";}
  // for(i=1;i<=n;i++) {cout<<" ";cin>>x[i][t];}

cout<<endl;
  // Verification. It is correct.

cout<<endl;
  // Construct the AutoCorrelation matrix WAutoCor
  // with the elements wij[][]
  // cout<<" Verification \n";
  // for(i=1;i<=n;i++) {wij[i][j]=0;}

cout<<endl;
  // Verification for WAC Autocorrelation wij. Print
  // line by line
  // cout<<" Autocorrelation matrix (symetric) line by line is :
";

for(i=1;i<=n;i++) // a new i generates a new line
{cout<<endl;
for(j=1;j<=n;j++) {cout<<"  \"<<wij[i][j];}
}
cout<<endl<<endl;

// Version 1: Vectors xr are se introduced
// SEQUENTIAL xr1,xr2,...,xrNxr
for(txr=1;txr<=Nxr;txr++) // BEGIN cycle txr=1,Nxr
   // Input the recall vector xr
   cout<<endl<<endl;
   cout<<"Input the recall  vector  xr 
"
   {cout<<"xr["<<txr<<"]:
   
{cout<<"xr[k+1] =xr[k] +wx1[k]
   // At the begining xr(k+1) is the nul vector
   for(i=1;i<=n;i++) {xrkp1[i]=0;}
   cout<<endl;
   // Verification for xr
   cout<<"xr["<<txr<<"]:
   {cout<<"xrkp1["<<i<<"]=
   // UPDATE k= "<<k;cout<<endl;
   }

   // We compara AT The Beginning xr(k) with xr(k+1),
   // namely xr with xrnnew=xrkp1
   // teta=xr(k)- xrnnew it \|teta\|^2 prin Procedura 3
   // Normteta2
   // normteta=Normteta2(n,teta,normteta2)
   for(i=1;i<=n;i++) {normteta[i]=xrkp1[i];}
   // Attention: program cycle i=1,n is terminated ;
   // Procedure uses the same variable i=1,n. It is correct
   // It is not contradiction
   // It is contradiction only when the Procedure still works
   // For example i the Programul uses the same variable i
   // as the non-terminated cycle
   normteta=Normteta2(n,teta,normteta2);
   cout<<endl;
   cout<<"Before the WHILE normteta= "<<normteta;
   cout<<endl<<endl;
   // Because at the beginning normteta>0.0, then the
   // instruction WHILE will work
   while(normteta>0.0) // At the beginning
   { // Begin for while
   // Variable k comptuns the cycles in the instruction
   // WHILE
   k=k+1;
   k1=k1+1;
   cout<<" Vector xr["<<txr<<"] Cycle WHILE k=
   
cout<<endl;
   cout<<" xrnnew = xrkp1["<<k<<"] ;
   for(i=1;i<=n;i++) {cout<<"  "<<xrkp1[i];}
   cout<<endl;
   // Energy Theorem. One computes
   // the energy E(xr)=xT*WAC)*xr=xT*colwij
   // E(xr) must be decreasing
   // E(xr) is computen in 2 steps
   // Step 1. We multiply WAC by xr and obtain the
   // vector colwij]
   for(i=1;i<=n;i++) {colwij[i]=0;}
   for(i=1;i<=n;i++)
   {for(j=1;j<=n;j++) {colwij[i]=colwij[i]+wij[i][j]*xr[j];}
   }
   // Step 2. We use the Procedure 4 Energxr with vectors
   xr and colwij
   energixr=Energxr(n,xr,colwij,energiexr);
   cout<<endl<<endl;
   cout<<"Energy for xrnnew["<<k<<"] is energiexr=
   
cout<<endl;
   cout<<"UPDATE k= "<<k;cout<<endl;

   // We construct xr(k+1) by the updating rule
   // Regula de Modificare(8.5) pag.200 in the book
   // netxrwj=wjT*xr. Use Procedure 2 PrScalar
   // Prepare vecta=wj and vectb=xr of din Procedure 2
   for(i=1;i<=n;i++)
   { // Begin A1
   for(j=1;j<=n;j++) {vecta[j]=wij[i][j];}
   for(j=1;j<=n;j++) {vectb[j]=xr[j];}
   netxrwj=PrScalar(n,vecta,vectb,net);
   // Attention. Weare in the cyclul i=1,n from program
   // Atentie:Daca Procedura PrScalar foloseste ciclist
   // propriu i=1,n
   // atunci DIN ACEST LOC mai departe Programul
   // foloseste i=n+1.FALS
   // That is why the Procedure must use another variable
   // of the de cycle
// for example i1=1,n or the cycle of program must
// use another variable

// cout<<endl;

// cout<<" Verification HERE i= "<<i;
// cout<<endl;
// cout<<" net "["<<i "= "<<netxrwj;
// cout<<endl;

// The updating rule. One obtains a new vector
// xnew=xrkp1
// netxrwj>0.0 xkp1 +1;netxrwj<0.0 xkp1 -1,
// netxrwj=0.0 x does not change
if(netxrwj>0.0) {xkp1[i]=1;fictiv=1;}
else if(netxrwj<0.0) {xkp1[i]=2;fictiv=2;}
else {xkp1[i]=x[i];fictiv=3;}

// The instruction fictiv=1 or 2 or 3 has no effect

// cout<<" xrkp1["<<i "= "<<xkp1[i];
// cout<<endl;

// Compare old xr by the new xkp1. We use the
// norm teta
for(i=1;i<n;i++) {teta[i]=x[i]*xkp1[i];}
normteta=Normteta2(n,teta,normteta2);

// cout<<endl;
// cout<<" The modified Vector x is xnew = xkp1["<<k1 "]":

// cout<<endl;
// Verification. Print xactualizat=xnew=xkp1
k1=k+1;

// cout<<endl;
// cout<<" The modified Vector x is xnew = xkp1["<<k1 "]":

// cout<<endl;
for(i=1;i<n;i++) {cout<<" "<<xkp1[i];}

// cout<<endl;
// cout<<" The Energy for xkp1["<<k1 "] is energiexr= "<<energiexr;
// cout<<endl;

// Begin the comparison of xnew with all vectors
// x(i)=teta for t=1,N
for(t=1;t<=N;t++)
{
  // Begin comparison t=1,N
  for(i=1;i<n;i++) {teta[i]=x[i]*x[i][t];}
  normteta=Normteta2(n,teta,normteta2);

  // Attention. Must be 0.0(two times =), not only one
  if(normteta==0.0)
  {
    cout<<" ** By vector x["<<txr " we find
    // (recall) the vector x["<<t "];
    else {fictiv=1;}// One continues the comparison
  }
}

// cout<<endl;
// cout<<" END of PROGRAM ";
cout<<endl<<endl;

getch();

} // END of Hopfield Program

3. The solved problems and the output of C++ program [5]

The program ask for data step by step. After each data we put enter.

Example 1.

\[ I(x) = \{x(1), x(2)\}, \quad I(xr) = \{xr, xr2\} \]

\[ x(1) = (1 \ 1 \ -1 \ 1)^T, \quad x(2) = (-1 \ 1 \ 1 \ 1)^T \]

Find the correct vectors of \( I(x) \) by the recall vector set

\[ xr1 = (1 \ 1 \ -1 \ 1)^T, \quad xr2 = (-1 \ -1 \ 1 \ 1)^T \]

Solution. \( n = 4, N = 2, Nxr = 2, kmax=10, \text{Tip}=1 \)

\[ W = WAC = \begin{pmatrix}
0 & -2 & 2 & 0 \\
-2 & 0 & -2 & 0 \\
2 & -2 & 0 & 0 \\
0 & 0 & 0 & 0
\end{pmatrix} \]

\[ DHx(1) xr1 = 2, \quad DHx(2) xr1 = 1 \]

\[ DHx(1) xr2 = 1, \quad DHx(2) xr2 = 2 \]

The recall vector \( xr1 \) yields the vector \( x(2) \).

The recall vector \( xr2 \) yields the vector \( x(1) \).

The results are plausible due to the Hamming distances between the vectors.

Example 2.

\[ I(x) = \{x(1), x(2)\} \]

\[ I(xr) = \{xr, xr2, xr3, xr4, xr5\} \]

\[ x(1) = (1 \ -1 \ 1 \ -1 \ 1)^T \]

\[ x(2) = (1 \ 1 \ 1 \ -1 \ -1)^T \]

Find the correct vectors of \( I(x) \) by the recall vector set

\[ xr1 = (1 \ 1 \ 1 \ -1 \ -1)^T, \quad xr2 = (-1 \ 1 \ 1 \ 1 \ -1)^T \]

\[ xr3 = (1 \ 1 \ 1 \ -1 \ 1)^T, \quad xr4 = (1 \ -1 \ 1 \ -1 \ -1)^T \]

\[ xr5 = (1 \ -1 \ 1 \ 1 \ -1\)^T \]

Solution. \( n = 6, N = 2, Nxr = 5, kmax=10, \text{Tip}=2 \)

\[ DHx(1) xr1 = 2, \quad DHx(2) xr1 = 2 \]

\[ DHx(1) xr2 = 6, \quad DHx(2) xr2 = 2 \]

\[ DHx(1) xr3 = 3, \quad DHx(2) xr3 = 1 \]

\[ DHx(1) xr4 = 1, \quad DHx(2) xr4 = 3 \]

\[ DHx(1) xr5 = 1, \quad DHx(2) xr5 = 3 \]

\[ W = WAC = \begin{pmatrix}
0 & 0 & 0 & -2 & 0 \\
0 & 0 & 2 & -2 & 0 \\
0 & 2 & 0 & -2 & 0 \\
-2 & 0 & 0 & 0 & 0 \\
0 & -2 & 2 & 0 & 0
\end{pmatrix} \]

The recall vector \( xr1 \) yields the cycling with alternative vectors. The constant energy \( E = 4 \).

Stop program. No vector recalled.

The recall vector \( xr2 \) yields the cycling with equal vectors. The constant energy \( E = 28 \).

Stop program. No vector recalled.

The recall vector \( xr3 \) yields the cycling with alternative vectors. The constant energy \( E = -20 \).

Stop program. No vector recalled.

The recall vector \( xr4 \) yields the vector \( x(1) \).

The decreasing energy \( E = -4, E = -28, E = -28 \).

The recall vector \( xr5 \) yields the vector \( x(1) \).

The decreasing energy \( E = -4, E = -28, E = -28 \).

The results are plausible due to the Hamming distances between the vectors.

4. Conclusions

The store-recall C++ program works efficiently.

The program ask for input data step by step and is very friendly in applications.

References


