Práctica 3 : Memorias Asociativas

Melisa Maidana Capitán

Junio 2013

Resumen

El problema de memoria asociativa consiste en *"Guardar en la red un conjunto de p patrones ξ_i^μ de manera tal que cuando se le presente un nuevo patrón ζ_i, la misma responda produciendo cualquiera de los patrones almacenados que más se parezca a ζ_i"*.

En el presente informe se muestran los resultados obtenidos para una red con el modelo de Hopfield con muchos patrones. Dado un número de neuronas N y de patrones p, se seleccionó uno a uno cada patrón y se los dejó evolucionar según las reglas del modelo de Hopfield. Acontinuación se calculó el overlap producido entre cada patrón y el resultante. Se calculó la capacidad de carga para distintos valores de N y p.

Luego se realizó el mismo procedimeinto utilizando patrones obtenidos a partir de un modelo estadístico basado en sistemas magnéticos. En este caso, se utilizó la aproximación de campo medio para difinir los estados neuronales.

1. Redes de Hopfield

Para la formulación de este modelo, se asume que los estados neuronales son .ªpagado.º "prendido", correspondientes a estados subumbrales o estados supraumbrales de la dinámica neuronal. En este caso, los estados apagados son representados con el estado -1, mientras que los estados prendidos son representados con el estado $+1$. La dinámica de la red está dada por:

$$S_i := sgn(\sum_{j=1}^{N} \omega_{ij} S_j - \theta_j) \tag{1}$$

donde N es el número total de neuronas, ω_{ij} es el peso asignado a cada conexión y θ_j es el valor del umbral de la j-ésima neurona. Este último puede omitirse agregando una neurona al sistema.

Según la regla de Hebb", los pesos están dados por $\omega_{ij} = \frac{1}{N} \sum_{\mu=1}^{p} \xi_i^\mu \xi_j^\mu$.

donde p es el número de patrones presentados.

Para analizar la estabilidad de un determinado patrón ξ_i^ν es necesario analizar:

$$sgn(h_i^\nu) = \xi_i^\nu \tag{2}$$

donde la entrada de la unidad i en el patrón ν es

$$h_i^\nu \equiv \sum_j \omega_{ij}\xi_j^\nu = \frac{1}{N}\sum_j\sum_\mu \xi_i^\mu\xi_j^\mu\xi_j^\nu = \xi_i^\nu + \frac{1}{N}\sum_j\sum_{\mu\neq\nu}\xi_i^\mu\xi_j^\mu\xi_j^\nu \tag{3}$$

Si el segundo término es pequeño (menor que 1), el mismo no modifica el valor que toma la función de transferencia a la neurona i-ésima del patrón ν (dada por la función signo). Para valores pequeños de p, el segundo término es suficientemente chico, de modo que los patrones almacenados son estables. Además, cualquier condiciones inicial cercana (en bits) al patrón ξ_i^ν convergerá por efecto de la dinámica del modelo, al patrón ξ_i^ν.

1.1. Modelado de una red de Hopfield

Para programar una red de Hopfield se siguieron los siguientes pasos:

- Se crearon patrones ξ_i^μ $(i = 1...N; \mu = 1...p)$ con valores equiprobables ± 1.
- Se calculó la matriz de conexciones $\omega_{ij} = \frac{1}{N}\sum_{\mu=1}^{p}\xi_i^\mu\xi_j^\mu$.
- Se tomó un patrón como condición inicial y se iteró la dinámica hasta converger a s_i^μ.
- Se calculó el overlap dado por $m^\mu = \frac{1}{N}\sum_{i=1}^{N} s_i^\mu\xi_i^\mu$.

Se programó una red de Hopfield para diferentes valores de N y p, de manera que en algunos casos p resulte lo suficientemente pequeño para que los patrones sean puntos fijos de la dinámica, y en otros de manera que no lo sean.

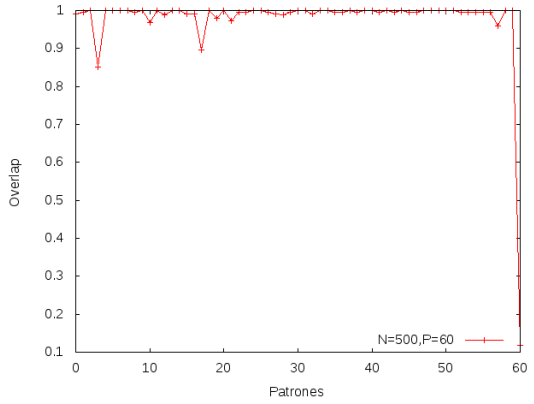

Figura 1: Overlap para $p/N = 0{,}12$ con $N = 500$.

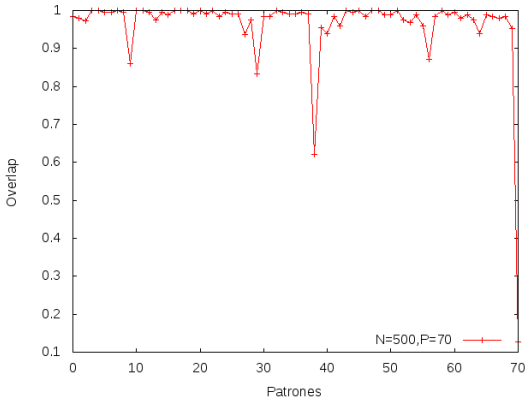

Figura 2: Overlap para $p/N = 0{,}14$ con $N = 500$.

Se observa que a medida que el factor $\alpha = \frac{p}{N}$ aumenta, el overlap entre los patrones y el estado final de la dinámica se vuelve más pequeño. Esto se corresponde al hecho de para algunos valores de α (suficientemente pequeños), los patrones resultan estables. El conjunto va perdiendo estabilidad conforme aumenta α.

A continuación se presenta la distribución de overlap para diferentes valores de $\frac{p}{N}$.

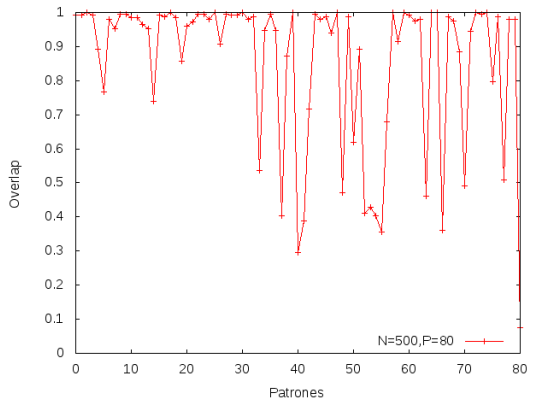

Figura 3: Overlap para $p/N = 0{,}16$ con $N = 500$.

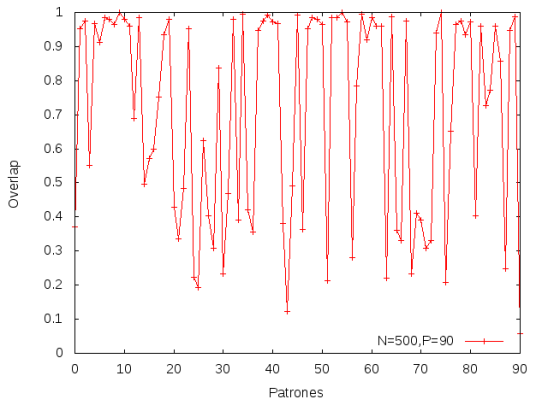

Figura 4: Overlap para $p/N = 0{,}18$ con $N = 500$.

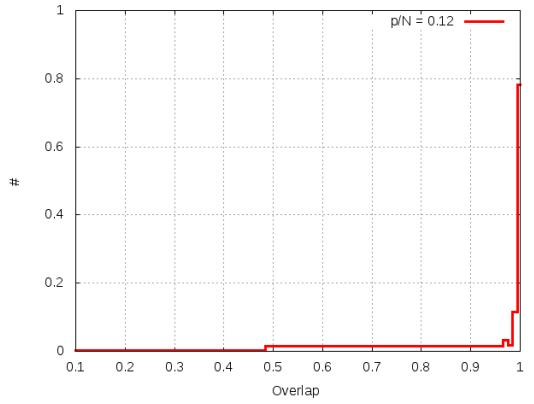

Figura 5: Distribución de overlap para $p/N = 0{,}12$ con $N = 500$.

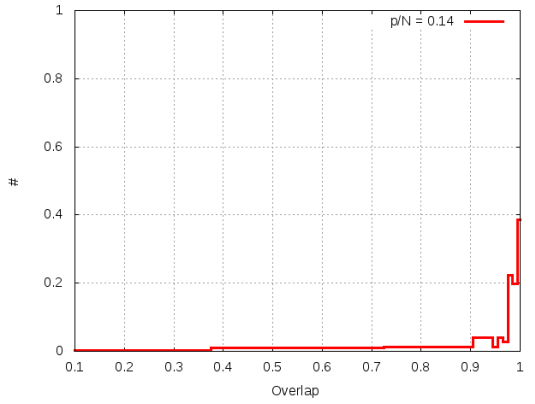

Figura 6: Distribución de overlap para $p/N = 0{,}14$ con $N = 500$.

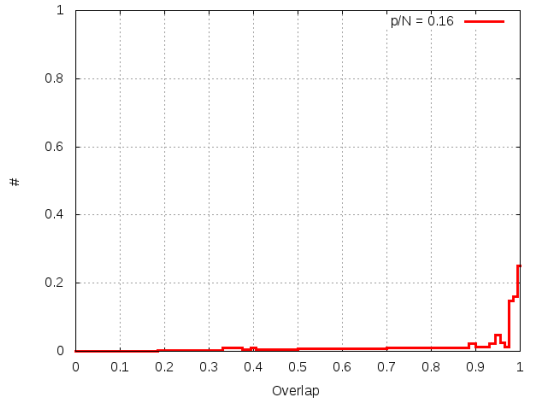

Figura 7: Distribución de overlap para $p/N = 0{,}16$ con $N = 500$.

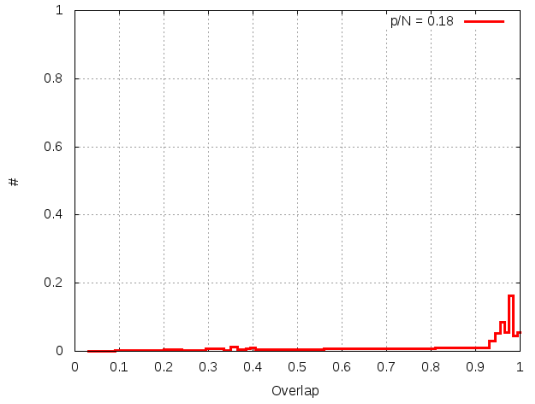

Figura 8: Distribución de overlap para $p/N = 0{,}18$ con $N = 500$.

1.1.1. Distribución de overlap

2. Redes de Hopfield y Mecánica estadística de modelos magnéticos

El modelo de red de Hopfield se asemeja a algunos modelos simples de materiales magnéticos. En particular, el problema descripto anteriormente se asemeja al "Modelo de Ising", en el cuál los spines pueden tomar valores ± 1. Los fuerza de las conexiones neuronales se asimilan a las fuerzas de intercambio de un material magnético, y los umbrales son representados por el campo externo.

Supongase que ahora se reemplaza la regla determinista presentada anteriormente, por un modelo estocástico dado por

$$S_i := \begin{cases} +1 \text{ con probabilidad } p(h_i) \\ -1 \text{ con probabilidad } 1 - p(h_i) \end{cases} \tag{4}$$

donde ahora $p(h)$ depende de la temperatura. En este trabajo tomamos como distribución de probabilidad $p(h_i) = \frac{exp^{\beta h_i}}{exp^{\beta h_i} + exp^{-\beta h_i}}$.

En modelos magnéticos, este problema se resulve de manera aproximada mediante la *teoría de campo medio*. Esta aproximación consiste en reemplazar el valor de h_i por su valor medio, de manera que

$$\langle S_i \rangle = tanh(\beta \langle h_i \rangle) \tag{5}$$

2.1. Programación del modelo probabilístico

Se programó una red con el modelo de Hopfield con estados neuronales dados por la función de probabilidad de la ec.(4). Tomando uno de los patrones como condición inicial se iteró un número $n = 10$ veces la dinámica del modelo, y para cada patrón se calculó el overlap. Se obtuvo el overlap medio para diferentes valores de temperatura T.

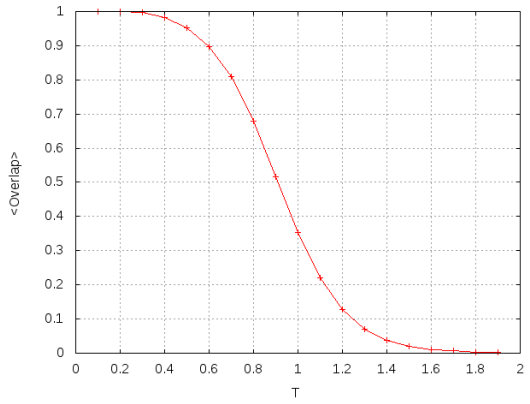

Figura 9: .

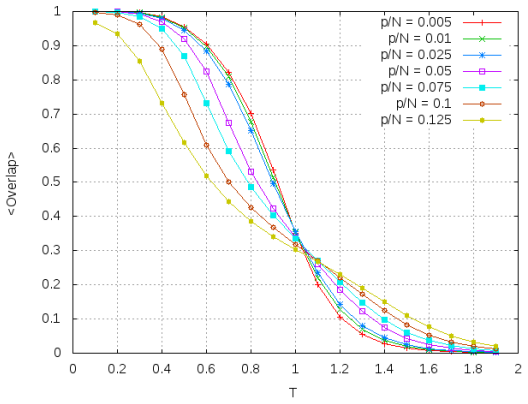

Figura 10:

En la Fig.(9) se observan diferentes características. En primer lugar, para valores $\frac{p}{N}$ suficientemente chicos, a temperaturas pequeñas el overlap medio es practicamente 1, y el mismo disminuye lentamente al aumentar T. En analogía al sistema ferromagnético, el estado de bajas temperaturas

se corresponde a la tendencia de los spines a alinearse con el campo magnético externo. En la red neuronal, significa que los patrones presentados son estables. Por otro lado, cuando la temperatura aumenta el ferromagneto se desordena y los spines apuntan en direcciones arbitrarias. En el caso de la red neuronal, el overlap medio se anula y esto se corresponde a la inestabilidad de los patrones.

En segundo lugar, se observa que la curva overlap medio vs. temperatura se modifica cuando cambia la relación $\frac{p}{N}$.

3. Programas en C

3.1. Programa Red de Hopfield determinista

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include <fstream>
#include <iostream>
#include <iostream>
#include <iomanip>
#include "nrutilc.h"
#include "funciones.h"

#include <cstdio>
#include <cstdlib>
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <cassert>
#include <malloc.h>

//include's de c++
#include <string>
#include <sstream>
using namespace std;

template<class T>
inline string to_string(const T& t){
    stringstream ss;
    ss << t;
    return ss.str();
}

int main(){

 int N,P;
 printf("Escriba el numero de neuronas de la red.\n");
 scanf("%d",&N);
 printf("Escriba el numero de patrones que desea almacenar\n");
 scanf("%d",&P);

 FILE *over;
```

```
 string red_overlap = to_string("Hopfield_overlap_") + to_string(N)+
    to_string("_")+to_string(P)+to_string(".dat");
 if((over = fopen(red_overlap.c_str(), "w"))==NULL){
        printf("Problemas para abrir archivo overlap.dat");
        }
 MatInt patrones(N,P);
 srand(time(0));
 GeneraPatrones(patrones);
 MatDoub conexiones(N,N);
 MatConexiones(patrones,conexiones);
 VecDoub estados(N),estado_inicial(N);
 VecDoub overlap(P);
 VecDoub aux(N);

 for(int p=0;p<=P;p++){
        for(int i=0;i<N;i++){
                estados[i]=patrones[i][p];
                estado_inicial[i]=patrones[i][p];
                }
        int conv=1;
        int convtime=0;
        while(conv != 0 && convtime<500){
                convtime++;
                conv=0;
                for(int i=0;i<N;i++)
                        aux[i]=estados[i];
                Dinamica(estados,conexiones);
                for(int i=0;i<N;i++){
                        if(estados[i]*aux[i]<0)
                                conv=1;
                        }
        }
        overlap[p] = CalculaOverlap(estados,estado_inicial);
        fprintf(over,"%lf\n", overlap[p]);
 }
 fclose(over);
 return 0;
 }
```

3.2. Programa Red de Hopfield estocástico

```
#define ITMAX 10
int main(){

 int N,P;
 printf("Escriba el numero de neuronas de la red.\n");
 scanf("%d",&N);
 printf("Escriba el numero de patrones que desea almacenar\n");
 scanf("%d",&P);

 FILE *over;
 string red_overlap = to_string("Hopfield2_overlapmedio_") + to_string(N)
    + to_string("_")+to_string(P)+to_string(".dat");
 if((over = fopen(red_overlap.c_str(), "w"))==NULL){
```

```
        printf("Problemas para abrir archivo overlap.dat");
        }

MatInt patrones(N,P);
srand(time(0));
GeneraPatrones(patrones);
MatDoub conexiones(N,N);
MatConexiones(patrones,conexiones);

VecDoub estados(N),estado_inicial(N);
VecDoub overlap(P);
VecDoub aux(N);
VecDoub h(N);
MatDoub prob(N,2);
double beta;
double TMAX=2;
double temperature=0.1;
double overlap_medio;

while(temperature<=TMAX){
        cout << t << endl;
        for(int p=0;p<=P;p++){
                for(int i=0;i<N;i++){
                        estados[i]=patrones[i][p];
                        estado_inicial[i]=patrones[i][p];
                        }
                int iteraciones=0;
                while(iteraciones < ITMAX){
                        CalculaH(conexiones,estados,h);
                        beta=1./temperature;
                        CalculaProb(h,prob,beta);
                        DinamicaProb(prob,estados);
                        iteraciones++;
                }
                overlap[p] = CalculaOverlap(estados,estado_inicial);
                }
        overlap_medio=PromVector(overlap);
        fprintf(over,"%lf\t%lf\n", temperature ,overlap_medio);
        temperature+=0.1;
        }

fclose(over);
return 0;
}
```

3.3. Librería auxiliar

```
#ifndef FUNCIONESINT_H
#define FUNCIONESINT_H

void GeneraPatrones(MatInt &matrix);
void MatConexiones(MatInt &matrix,MatDoub &Conexiones);
int Signo(double x);
void Dinamica(VecDoub &estados,MatDoub &conexiones);
```

```
double CalculaOverlap(VecDoub &efinal,VecDoub &einicial);
void CalculaH(const MatDoub &conexiones,const VecDoub &estados,VecDoub &h
    );
void CalculaProb(const VecDoub &h,MatDoub &prob,double beta);
void DinamicaProb(const MatDoub &prob,VecDoub &estados);
double PromVector(const VecDoub &vector);

void GeneraPatrones(MatInt &matrix){

 int p=matrix.ncols();
 int n=matrix.nrows();

 double aleatorio;
 for(int i=0;i<n;i++){
        for(int j=0;j<p;j++){
                aleatorio = rand()*1./RAND_MAX;
                if(aleatorio > 0.5)
                        matrix[i][j] = 1;
                else
                        matrix[i][j] = -1;
                }
        }
}
void MatConexiones(MatInt &matrix,MatDoub &Conexiones){

 int n = matrix.nrows();
 int m = matrix.ncols();

 for(int i=0;i<n;i++){
        for(int j=0;j<i;j++){
                for(int k=0;k<m;k++)
                        Conexiones[i][j]+=matrix[i][k]*matrix[j][k];
                Conexiones[i][j]/=n;
                }
        }

 for(int i=0;i<n;i++){
        for(int j=i;j<n;j++){
                Conexiones[i][j]=Conexiones[j][i];
                }
        }

 for(int i=0;i<n;i++){
        Conexiones[i][i]=0;
        }
}
int Signo(double x){
 if(x>0)
        return 1;
 else
        return -1;
}

void Dinamica(VecDoub &estados,MatDoub &conexiones){

 int n = estados.size();
```

```
 VecDoub auxiliar(n);

 for(int i=0;i<n;i++){
        auxiliar[i]=estados[i];
        estados[i]=0;
        }

 for(int i=0;i<n;i++){
        for(int j=0;j<n;j++)
                estados[i]+=conexiones[i][j]*auxiliar[j];
        estados[i]=Signo(estados[i]);
        }
}
double CalculaOverlap(VecDoub &efinal,VecDoub &einicial){

 double mu,suma=0;
 int n=efinal.size();

 for(int i=0;i<n;i++)
        suma+=efinal[i]*einicial[i];
 mu = suma*1.0/n;
 return mu;
}
void CalculaH(const MatDoub &conexiones,const VecDoub &estados,VecDoub &h
   ){

 int n=h.size();

 for(int i=0;i<n;i++)
        h[i]=0;

 for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
                h[i]+=conexiones[i][j]*estados[j];
        }
 }
}

void CalculaProb(const VecDoub &h,MatDoub &prob,double beta){

 int n=h.size();

 for(int i=0;i<n;i++){
        prob[i][0]=exp(beta*h[i])/(exp(beta*h[i])+exp(-beta*h[i]));
        prob[i][1]=exp(-beta*h[i])/(exp(beta*h[i])+exp(-beta*h[i]));
        }
 }
void DinamicaProb(const MatDoub &prob,VecDoub &estados){

 int n=estados.size();

 for(int i=0;i<n;i++){
        estados[i]=1*prob[i][0]+(-1)*prob[i][1];
 }
}
```

```
double PromVector(const VecDoub &vector){

 int n=vector.size();
 double prom=0;

 for(int i=0;i<n;i++)
        prom+=vector[i];
 prom/=n;
 return prom;
}
#endif
```

Referencias

[1] Introduction to the theory of neural computation. John Hertz,Anders Krogh,Richard G.Palmer. Westview Press (1991).