

8 FÉVRIER 2000

Table des matières

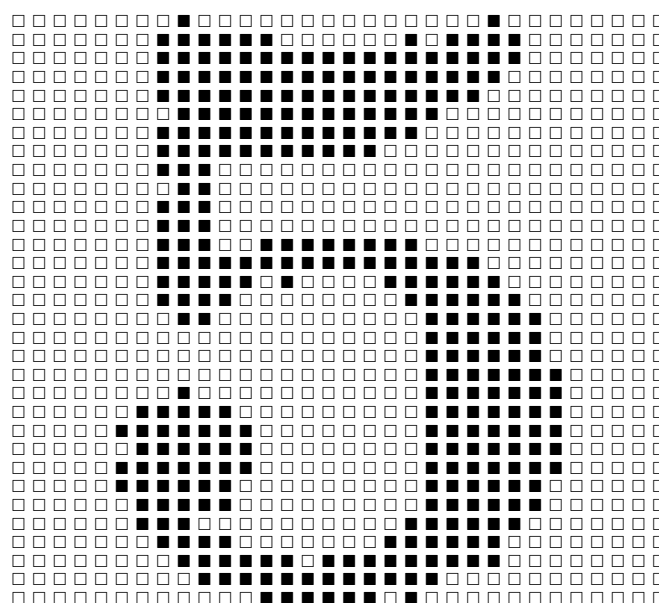
| | | |
|----------|-------------------------------|----------|
| 1 | Introduction | 2 |
| 2 | Les formes | 2 |
| 3 | Test | 2 |
| 3.1 | Variation de ρ | 2 |
| 3.2 | Test au bruit | 3 |
| 4 | Conclusion | 4 |
| 5 | Sources | 4 |
| 5.1 | Makefile | 4 |
| 5.2 | art.h | 5 |
| 5.3 | art.c | 6 |
| 5.4 | art1.c | 8 |
| 5.5 | utilitaire.h | 15 |
| 5.6 | utilitaire.c | 16 |

1 Introduction

Dans ce projet, je présente un test du réseau ART au bruit. Je lui présente 10 formes qui sont de plus en plus bruitées au cours de l'expérience.

2 Les formes

Les dix formes que je présente au réseau sont les chiffres de la fontes "Computer Modern Bold". Les formes sont présentées dans une matrice 32×32 . Les cases de la matrice sont binaires. Si un point est noir sur le papier, il sera représenté par un 1 sinon il sera représenté par un 0.



La figure ci dessus représente le 5 tel que je le présente au réseau.

3 Test

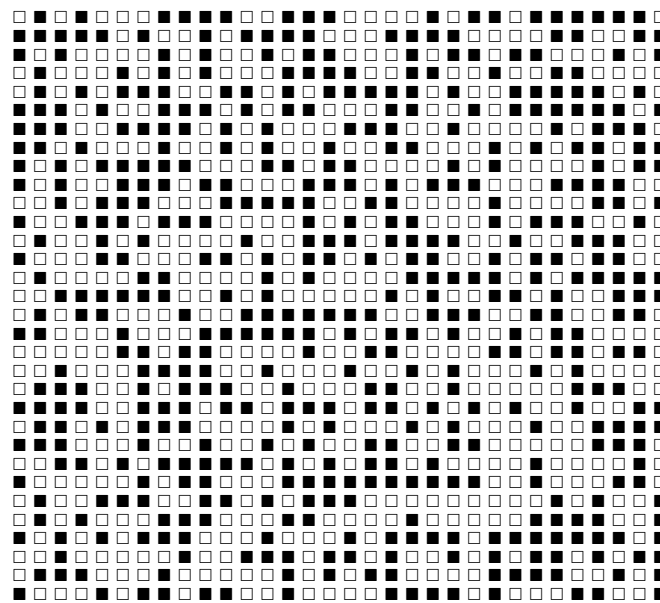
3.1 Variation de ρ

J'ai effectué plusieurs tests pour déterminer la valeur du coefficient ρ . Le coefficient est bon si il permet de créer dix classes. J'ai donc fait monter progressivement la valeur de ρ et j'ai noté le moment où j'avais dix classes. La valeur minimale de ρ est donc de 0.81. Avec $\rho = 0.80$, le 0 et le 6 sont confondus. Pour la suite des test, j'ai pris un ρ égal à 0.95.

3.2 Test au bruit

J'ai effectué un test de résistance au bruit. J'ai progressivement bruité tous mes chiffres et j'ai noté à quel moment le système ne les reconnaissait plus.

La fonction de bruit parcourt la matrice et permute des bits suivant une probabilité donnée. Un chiffre bruité à 100% est un chiffre qui a tout ses bits inversés donc il nous apparaît en négatif. Par contre un chiffre bruité à 50% n'est par reconnaissable par un humain.



Le chiffre 5 bruité a 50%.

Le tableau ci-dessous présente les différentes valeurs que j'ai obtenu. Sur chaque ligne, j'ai rapporté les résultats d'un test. Chaque case indique le moment où le réseau a arrêté de reconnaître le bon chiffre. Par exemple, la première case dit que lors d'un premier test, le réseau a bien reconnu le chiffre 1 jusqu'à 49,1% de bruit.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|------|------|------|------|------|------|------|------|------|------|
| 49,1 | 51,6 | 54,0 | 50,6 | 51,8 | 50,7 | 50,4 | 49,4 | 50,7 | 50,3 |
| 50,3 | 51,6 | 49,1 | 49,2 | 50,3 | 48,7 | 48,6 | 51,8 | 50,5 | 49,6 |
| 51,2 | 51,2 | 51,0 | 50,2 | 50,8 | 49,1 | 53,2 | 53,1 | 50,4 | 50,4 |
| 51,8 | 51,0 | 50,6 | 49,5 | 50,1 | 50,1 | 50,1 | 49,9 | 48,2 | 48,8 |
| 48,5 | 52,9 | 50,2 | 50,1 | 51,9 | 48,9 | 49,9 | 52,3 | 51,1 | 52,1 |
| 50,1 | 51,5 | 50,0 | 50,4 | 49,8 | 50,5 | 51,5 | 50,6 | 49,1 | 54,2 |
| 49,9 | 51,6 | 50,9 | 51,8 | 49,2 | 49,6 | 47,3 | 51,8 | 52,6 | 49,8 |
| 49,2 | 50,1 | 50,4 | 51,3 | 52,2 | 52,1 | 49,5 | 50,0 | 50,3 | 49,0 |

On peut dire, au vu du tableau précédent, que tout les chiffres sont reconnus jusque au même point. La moyenne est légèrement supérieure à 50%. L'intervalle

est situé entre 47,3% et 54,2%. On notera que à ces niveaux de bruit un humain est totalement incapable de discerner la moindre forme.

4 Conclusion

Le réseau de ART supporte bien le bruit. Si on lui aussi les patterns bruité à 100%, il se pourrait qu'il devienne insensible au bruit.

5 Sources

5.1 Makefile

```
=                                                                 Makefile      page

ifeq ($(HOSTTYPE),i386)
    CFLAGS=-Wall -O6 -march=pentiumpro -mieee-fp -Iinclude
    MALLOCDIR=/home/djedi/lib
else
    CFLAGS=-Wall -Iinclude -O6
    MALLOCDIR=/home/djedi/pub/usr/lib
endif

OBJ = art.o utilitaires.o art1.o
.PHONY=clean lclean doc
#INCDIR = -Iinclude
CC = egcc

art : $(OBJ)
    $(CC) $(CFLAGS) -o art $(OBJ) -lm -L$(MALLOCDIR) -lsmalloc

lclean :
    rm -f *~ \#*

clean : lclean
```

```
rm -f *.o backprop art interface affiche_nist

doc :
    cxref -html -latex2e -Odocs *.c *.h
    (cd docs; latex cxref; latex cxref; latex cxref; dvips cxref.dvi -
o cxref.ps)
```

5.2 art.h

```
=
art.h      page

/*****
*      Projet reconnaissance de formes      *
*      Module ART1                          *
*      Header                              *
*      Valvassori Moïse - Matricule : 149397  *
*                                          *
*      $Id:$                                *
*****/

#ifndef ART_H
#define ART_H    /*+ To stop multiple inclusions. +*/

typedef unsigned char  CELL;
#define acces(t,i,j) (*(t+ j*TAILLE_MAX +i)) /*+ Accède à une cellule d'une forme +*/

#define TRUE 1
#define FALSE 0

#define TAILLE_MAX 32
#define NOMBRE_CAPTEUR  TAILLE_MAX*TAILLE_MAX /*+ +*/
#define NOMRE_PATTERN_RECONNAISSABLE 11 /*+ Cette valeur est totalement arbitraire +*/
```

```

extern int initialisation(void) ;
extern int applique_le_pattern(CELL * in) ;
extern int calcul_activation(void) ;
extern int vigilance(int j) ;
extern int inhibe(int e) ;
extern int modifier_les_poids(int k) ;
extern void apprend() ;
extern void affiche_attracteurs(void) ;
#endif /* ART_H */

```

5.3 art.c

```

=
art.c    page

#include <stdio.h>
#include "utilitaires.h"
#include "s_malloc.h"

CELL *patt[10] ;

extern float Rho ;

/*+++++
charge les formes
+++++

void init_patterns(void){
    patt[0]=charger_char("img/0.dat") ;
    patt[1]=charger_char("img/1.dat") ;
    patt[2]=charger_char("img/2.dat") ;
    patt[3]=charger_char("img/3.dat") ;
    patt[4]=charger_char("img/4.dat") ;
    patt[5]=charger_char("img/5.dat") ;
    patt[6]=charger_char("img/6.dat") ;
    patt[7]=charger_char("img/7.dat") ;
    patt[8]=charger_char("img/8.dat") ;

```

```

    patt[9]=charger_char("img/9.dat");
}

/*+++++
Free sur les forme
+++++

void libere_patterns(){
    int i;
    for (i=0; i<10; i++)
        FREE(patt[i]);
}

/*+++++
Reconnait une forme

int reconnait l'index de la forme reconnu

CELL * in
+++++

int reconnait(CELL * in){
    applique_le_pattern(in);
    return(calcul_activation());
}

/*+++++
Effectue le test au bruit
+++++

void test(){

```

```

int n,i;
CELL *c;
int m;

c = (CELL *)CALLOC(TAILLE_MAX*TAILLE_MAX,sizeof(CELL));
srand(time(NULL));
for (n=0; n<10; n++){
    m=0;
    for (i=0; i<1000; i++){
        c=copie_bruite(patt[n],c,i);
        if (reconnait(c)==n)
            m=i;
    }
    printf("%d %d\n",n,m);
    /* c=copie_bruite(patt[n],c,m);
    * affiche_char(c); */
}
FREE(c);
}

int main (void){
    init_patterns();

    Rho=0.95;
    apprend();

    test();
    libere_patterns();
    MSTAT();
    return (0);
}

```

5.4 art1.c

= art1.c page

```

/*****
*      Projet reconnaissance de formes      *
*      Module ART1                          *
*****/

```



```

*
*   Valvassori Moïse - Matricule : 149397
*
*
*   $Id:$
*
*****/

```

```

#include <stdio.h>
#include <stdlib.h>
#include "art.h"

```

```

typedef unsigned char Binaire;

```

```

#define NOMBRE_NEURONE_ENTREE  TAILLE_MAX*TAILLE_MAX    /*+  +*/

```

```

float Rho= 0.70;

```

```

double ltm[NOMRE_PATTERN_RECONNAISSABLE]; /*+ Mémoire à long terme +*/

```

```

Binaire stm[NOMBRE_NEURONE_ENTREE];      /*+ Mémoire à court terme +*/

```

```

double bup[NOMBRE_NEURONE_ENTREE][NOMRE_PATTERN_RECONNAISSABLE]; /*+ Connexion
up +*/

```

```

Binaire topd[NOMRE_PATTERN_RECONNAISSABLE][NOMBRE_NEURONE_ENTREE]; /*+ Connexion
down +*/

```

```

Binaire mu[NOMRE_PATTERN_RECONNAISSABLE]; /*+ active/inhibe un attracteur +*/

```

```

static int prod_scal(int k);

```

```

double erreur;                                /*+ Erreur de reconnaissance - Debug only +*/

```

```

int attracteur_max;                            /*+ Dernier attracteur en activité +*/

```

```

extern CELL * patt[];

```

```

/*+++++

```

```

Initialise les poids du reseau

int initialisation renvoie toujours 1
+++++

int initialisation(void){
    int i,j;
    double init = 1.0 / (1.0 + NOMBRE_NEURONE_ENTREE);

    for (j=0; j<NOMRE_PATTERN_RECONNAISSABLE; j++){
        for (i=0; i<NOMBRE_NEURONE_ENTREE; i++){
            topd[j][i]=1;
            bup[i][j] = init;
        }

        attracteur_max=0;
        return (1);
    }

/*+++++
Applique le pattern sur les entrées

int applique_le_pattern

CELL * in le pattern à appliquer
+++++

int applique_le_pattern(CELL * in){
    int i,j,n=0;

    for (j=0; j<TAILLE_MAX; j++){
        for (i=0; i<TAILLE_MAX; i++){
            stm[n++]= acces(in,i,j) != 0;
        }
    }

    /* Déinhibe les attracteurs */
    for (j=0; j<NOMRE_PATTERN_RECONNAISSABLE; j++){
        mu[j]=1;

```

```

    return(1) ;
}

```

```

/*+++++++
Calcule les valeurs d'activation des attracteurs

```

```

int calcul_activation renvoie l'index de l'attracteur max
+++++++

```

```

int calcul_activation(void){
    int i,j;
    int amax=-1,flag=TRUE;
    double max=0;

    for (j=0; j<NOMRE_PATTERN_RECONNAISSABLE; j++){
        if (mu[j]==1) {
            ltm[j]=0.0;
            for (i=0; i<NOMBRE_NEURONE_ENTREE; i++)
                ltm[j] += bup[i][j] * stm[i];

            if (flag==TRUE){
                /* Calcule le max dans la foulée */

                max=ltm[j];
                amax=j;
                flag=FALSE; }
            else if (ltm[j]>max){
                amax = j;
                max=ltm[j];
            }
            /* printf("%d %f \n",j,ltm[j]); */
        }
    }
    /* printf("max %d ",amax); fflush(0); */
    return (amax);
}

```

```

/*+++++++
prédicat qui dit si l'elue est suffisamment proche de l'entrée

int vigilance vrai/faux ou -1 si erreur

int j l'index de l'elue.
+++++++

int vigilance(int j){
    int i;
    int xx=0;
    int wx=0;

    if (j<0)
        return (-1);

    for (i=0; i<NOMBRE_NEURONE_ENTREE; i++){
        xx += stm[i];
        wx += topd[j][i] * stm[i];
    }

    erreur = ((double)wx/xx);
    /* printf(". %d %d %f\n",xx,wx,erreur); */
    if (((double)wx/xx) > Rho)
        return (TRUE);
    else
        return (FALSE);
}

/*+++++++
Inhibe un attracteur

int inhibe

int e l'attracteur à inhiber
+++++++

```

```

int inhibe(int e){
    if (e<0)
        return (0);

    mu[e]=0;
    /* printf("inib %d\n",e); */
    return (1);
}

/*+++++++

int modifier_les_poids

int k
+++++++

int modifier_les_poids(int k){
    int i;
    double demominateur;

    if (k<0)
        return (0);

    demominateur= 0.5 + prod_scal(k);

    for (i=0; i<NOMBRE_NEURONE_ENTREE; i++){
        topd[k][i] = topd[k][i] * stm[i];
        bup[i][k]= topd[k][i] / demominateur ;
    }

    if (k>attracteur_max)                /* C'est ici que l'on modifie attracteur max */

        attracteur_max=k;

    return (1);
}

```

```

static int prod_scal(int k){
    int i;
    int ret=0;

    for (i=0; i<NOMBRE_NEURONE_ENTREE; i++){
        ret += topd[k][i] * stm[i];
    }
    return (ret);
}

```

```

/*+++++

```

```

int affiche_attracteur

```

```

int a

```

```

+++++

```

```

int affiche_attracteur(int a){
    int i,j,n=0;

    for (j=0; j<TAILLE_MAX; j++){
        for (i=0; i<TAILLE_MAX; i++){
            printf("%c",topd[a][n++]?'#': ' ');
        }
        printf("\n");
    }

    return(1);
}

```

```

void apprend(){
    int i,max=0;
    int elue;

```

```

initialisation();
do {
    max=attracteur_max;
    for (i=0; i<10; i++){
        applique_le_pattern(patt[i]);

        while(1) {
            elue=calcul_activation();

            if (vigilance(elue))
                break;
            else
                inhibe(elue);
        };
        printf("> %d %f %d\n",i,erreur,elue);
        modifier_les_poids(elue);
    }
}while(max<attracteur_max);
}

```

```

void affiche_attracteurs(void){
    int i;
    for (i=0; i<attracteur_max; i++){
        affiche_attracteur(i);
        printf("\n");
    }
}

```

5.5 utilitaire.h

=

utilitaires.h page

```

/*****

```

```

$Id$

```

```

*****/

```

```

#ifndef UTILITAIRES_H
#define UTILITAIRES_H    /*+ To stop multiple inclusions. +*/

#include "art.h"

extern CELL *charger_char(char *n) ;
extern void affiche_char(CELL *c) ;
extern CELL * copie_bruite(CELL*c,CELL *h,int bruit) ;

#endif /* UTILITAIRES_H */

```

5.6 utilitaire.c

=

utilitaires.c page

```

#include <stdio.h>
#include <stdlib.h>
#include "s_malloc.h"
#include "art.h"

/*+++++
Charge un caractère du disque
ATTENTION : Il y a allocation dynamique

CELL * charger_char Le nouveau char

char *n le nom du fichier où il y a le char
+++++

CELL * charger_char(char *n){
    FILE *f ;
    int i,j ;
    char c ;
    CELL * tab ;

    f=fopen (n,"r") ;

```



```

    tab = (CELL *)CALLOC(TAILLE_MAX*TAILLE_MAX,sizeof(CELL));

    for (j=0; j<TAILLE_MAX; j++){
        for (i=0; i<TAILLE_MAX; i++){
            fscanf(f,"%c",&c);
            acces(tab,i,j) = (c=='+' ? 0 : 1);
        }
        fscanf(f,"%c",&c);
    }

    fclose (f);
    return tab;
}

/*+++++
affiche un caractère

CELL *c
+++++

void affiche_char(CELL *c){
    int i,j;

    for (j=0; j<TAILLE_MAX; j++){
        for (i=0; i<TAILLE_MAX; i++)
            printf("%c",acces(c,i,j)?'#' : ' ');
        printf("\n");
    }
}

/*+++++
Effectue la copie du chiffre c avec  $\frac{\text{bruit}}{1000}$  de bruit

CELL * copie_bruite le pointeur sur la copie

```

```

CELL*c le chiffre d'origine

CELL *tab la destination

int bruit le bruit  $\in [0, 1000]$ 
+++++*/

CELL * copie_bruite(CELL*c,CELL *tab,int bruit){
    CELL x;
    int i,j;

    for (j=0; j<TAILLE_MAX; j++){
        for (i=0; i<TAILLE_MAX; i++){
            x=acces(c,i,j);
            if (1000.0*rand()/(RAND_MAX+1.0) < bruit)
                acces(tab,i,j)=(x==1?0 :1);
            else
                acces(tab,i,j)=x;
        }
    }

    return (tab);
}

```