Adaptacyjna teoria rezonansu (ART)

Wstęp

Adaptacyjna teoria rezonansu Grossberga, rozwinięta dalej przez Grossberga i Carpentera, służy do kategoryzacji wzorców przy użyciu paradygmatu konkurencyjnego uczenia się. Wprowadza kontrolę wzmocnienia i resetowanie, aby upewnić się, że wyuczone kategorie zostaną zachowane nawet podczas nauki nowych kategorii, a tym samym rozwiązuje dylemat plastyczność-stabilność. Adaptacyjna teoria rezonansu w dużym stopniu wykorzystuje konkurencyjny paradygmat uczenia się. Opracowano kryterium ułatwiające wystąpienie zjawiska „zwycięzca bierze wszystko". Pojedynczy węzeł z największą wartością dla ustawionego kryterium zostaje ogłoszony zwycięzcą w swojej warstwie i mówi się, że klasyfikuje klasę wzorca. Jeśli w warstwie jest remis między zwycięskim neuronem, za zwycięską może zostać uznana arbitralna reguła, taka jak pierwsza z nich w kolejności seryjnej. Sieć neuronowa opracowana dla tej teorii tworzy system, który składa się z dwóch podsystemów, z których jeden to podsystem uwagi, który zawiera jednostkę kontroli wzmocnienia. Drugi to podsystem orientujący, zawierający jednostkę do resetowania. Podczas działania sieci wymodelowanej dla tej teorii w podsystemie uwagi pojawiają się wzorce i nazywane są śladami STM (pamięci krótkotrwałej). Ślady LTM (pamięci długotrwałej) znajdują się w wagach połączeń między warstwą wejściową a wyjściową. Sieć wykorzystuje przetwarzanie ze sprzężeniem zwrotnym między dwiema warstwami, aż do wystąpienia rezonansu. Rezonans występuje, gdy dane wyjściowe w pierwszej warstwie po sprzężeniu zwrotnym z drugiej warstwy pasują do oryginalnego wzoru używanego jako dane wejściowe dla pierwszej warstwy w tym cyklu przetwarzania. Mecz tego typu nie musi być idealny. Wymagane jest, aby stopień dopasowania, odpowiednio zmierzony, przekraczał z góry określony poziom, zwany parametrem czujności. Podobnie jak fotografia w większym stopniu pasuje do wizerunku obiektu, gdy ziarnistość jest większa, dopasowanie do wzoru staje się dokładniejsze, gdy parametr czujności jest bliższy 1.

Sieć dla ART1

Sieć neuronowa dla adaptacyjnej teorii rezonansu lub modelu ART1 składa się z następujących elementów:

• Warstwa neuronów, zwana warstwą F_1 (warstwa wejściowa lub warstwa porównawcza)

• Węzeł dla każdej warstwy jako jednostka kontroli wzmocnienia

• Warstwa neuronów, zwana warstwą F_2 (warstwa wyjściowa lub warstwa rozpoznawania)

• Węzeł jako jednostka resetująca

• Połączenia oddolne z warstwy F_1 do warstwy F_2

• Połączenia odgórne z warstwy F_2 do warstwy F_1

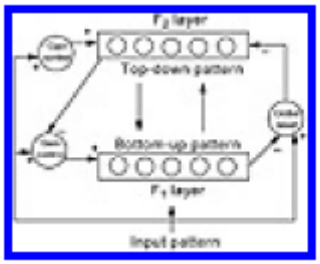
• Połączenie hamujące (waga ujemna) z warstwy F_2 w celu uzyskania kontroli

• Połączenie pobudzające (waga dodatnia) od kontroli wzmocnienia do warstwy

• Połączenie wstrzymujące z warstwy F_1 do węzła resetowania

• Połączenie pobudzające od węzła resetowania do warstwy F_2

Uproszczony schemat układu sieci



Przetwarzanie w ART1

Paradygmat ART1, podobnie jak samoorganizująca się mapa Kohonena, która zostanie wprowadzona w części 11, wykonuje grupowanie danych na danych wejściowych; podobne dane wejściowe są zgrupowane razem w kategorię. Jako przykład można użyć algorytmu grupowania danych, takiego jak ART1 do optycznego rozpoznawania znaków (OCR), w którym próbujesz dopasować różne próbki litery do jej odpowiednika ASCII. Szczególną uwagę przywiązuje się do paradygmatu ART1, aby zapewnić, że stare informacje nie zostaną wyrzucone podczas przyswajania nowych. Wektor wejściowy zastosowany do systemu ART1 jest najpierw porównywany z istniejącymi wzorcami w systemie. Jeśli istnieje wystarczająco bliskie dopasowanie w ramach określonej tolerancji (wskazane przez parametr czujności), wówczas przechowywany wzorzec jest jeszcze bardziej podobny do wzorca wejściowego, a operacja klasyfikacji jest zakończona. Jeśli wzorzec wejściowy nie przypomina żadnego z wzorców przechowywanych w systemie, tworzona jest nowa kategoria z nowym wzorcem przechowywanym, który przypomina wzorzec wejściowy.

Cechy szczególne modelu ART1

Szczególną cechą modelu ART1 jest to, że do określenia aktywności neuronów w warstwie F_1 niezbędna jest reguła dwóch trzecich. Dla każdego neuronu w warstwie F_1 istnieją trzy źródła wejściowe. Są to wejście zewnętrzne, wyjście kontroli wzmocnienia i wyjścia neuronów warstwy F2. F1neurony nie uruchomią się, chyba że co najmniej dwa z trzech wejść są aktywne. Jednostka kontroli wzmocnienia i reguła dwóch trzecich razem zapewniają prawidłową odpowiedź neuronów warstwy wejściowej. Drugą cechą jest to, że parametr czujności jest używany do określenia aktywności jednostki resetowania, która jest aktywowana, gdy podczas klasyfikacji nie znaleziono dopasowania wśród istniejących wzorców.

Notacja do obliczeń ART1

Wymieńmy różne symbole, których użyjemy do opisania działania sieci neuronowej dla modelu ART1:

w_{ij} - Waga połączenia od i-tego neuronu w warstwie F1 do j-tego neuronu w warstwie F2

v_{ji} - waga połączenia od j-tego neuronu w warstwie F2 do i-tego neuronu w warstwie F1

a_i - Aktywacja i-tego neuronu w warstwie F1

b_j - Aktywacja neuronu j w warstwie F2

x_i - Wyjście neuronu i-tego w warstwie F1

y_j - Wyjście neuronu j-tego w warstwie F2

z_i - Wejście do i-tego neuronu w warstwie F1 z warstwy F2

ρ - Parametr czujności, dodatni i nie większy niż 1 (0<1)

m - Liczba neuronów w warstwie F1

n - Liczba neuronów w warstwie F2

I - Wektor wejściowy

S_i - Suma składowych wektora wejściowego

S_x - Suma wyjść neuronów w warstwie F1

A, C, D - Parametry o wartościach dodatnich lub zerowych

L - Parametr o wartości większej niż 1

B - Parametr o wartości mniejszej niż D + 1, ale co najmniej tak dużej jak D lub 1

r - Indeks zwycięzcy konkursu w warstwie F2

Algorytm do obliczeń ART1

Równania ART1 nie są łatwe do naśladowania. Postępujemy zgodnie z opisem algorytmu znalezionym u Jamesa A. Freemana i Davida M. Skapura. Poniższe równania, wzięte w podanej kolejności, opisują kroki algorytmu. Zauważ, że binarne wzorce wejściowe są używane w ART1.

Inicjalizacja parametrów

w_{ij} powinno być dodatnie i mniejsze niż L / (m - 1 + L)

v_{ji} powinno być większe niż (B - 1) / D

a_i = -B / (1 + C)

Równania do obliczeń ART1

Czytając poniżej równania do obliczeń ART1, pamiętaj o następujących uwagach. Jeżeli po lewej stronie równania pojawia się indeks i, oznacza to, że takich równań jest m, ponieważ indeks dolny i waha się od 1 do m. Podobnie, jeśli zamiast tego występuje indeks dolny j, to istnieje n takich równań, jak j w zakresie od 1 do n. Równania są używane w kolejności, w jakiej zostały podane. Podają krok po kroku opis następującego algorytmu. Wszystkie zmienne, jak sobie przypominasz, są zdefiniowane we wcześniejszej części poświęconej notacji. Na przykład I jest wektorem wejściowym.

Obliczenia warstwy F_1:

$$a_i = I_i \;/\; (1 + A (I_i + B) + C)$$
$$x_i = 1 \text{ if } a_i > 0$$
$$\quad\; 0 \text{ if } a_i \leq 0$$

Obliczenia warstwy F_2

$$b_j = \Sigma\, w_{ij}\, x_i, \text{ the summation being on i from 1 to m}$$
$$y_j = 1 \text{ if jth neuron has the largest activation value in the } F_2 \text{ layer}$$
$$= 0 \text{ if jth neuron is not the winner in } F_2 \text{ layer}$$

Wejścia odgórne:

$z_i = \Sigma v_{ji} y_j$, the summation being on j from 1 to n (You will notice that exactly one term is nonzero)

Obliczenia warstwy F_1:

$$a_i = (I_i + D z_i - B) / (1 + A (I_i + D z_i) + C)$$
$$x_i = 1 \text{ if } a_i > 0$$
$$= 0 \text{ if } a_i \leq 0$$

Sprawdzenie z parametrem czujności:

Jeśli (S_x / S_I) < Σ;, ustaw y_j = 0 dla wszystkich j, w tym zwycięskiego r w warstwie F_2 i uznaj neuron j-ty za nieaktywny (ten krok jest resetowany, pomiń pozostałe). Jeśli (S_x/S_I) kontynuuj.

Modyfikowanie wagi połączenia top-down i bottom-up dla zwycięzcy r:

$$v_{ir} = (L / (S_x + L - 1) \text{ if } x_i = 1$$
$$= 0 \text{ if } x_i = 0$$
$$w_{ri} = 1 \text{ if } x_i = 1$$
$$= 0 \text{ if } x_i = 0$$

Po zakończeniu z bieżącym wzorcem wejściowym powtarzamy te kroki z nowym wzorcem wejściowym. Tracimy indeks r nadany jednemu neuronowi jako zwycięzcy i traktujemy wszystkie neurony w warstwie F_2 z ich oryginalnymi indeksami (indeksami dolnymi). Powyższa prezentacja algorytmu ma na celu jak najjaśniejsze wyjaśnienie wszystkich kroków. Proces jest dość skomplikowany. Reasumując, najpierw neuronom warstwy F_1 przedstawiany jest wektor wejściowy, określane są ich aktywacje, a następnie wykorzystywana jest funkcja progowa. Wyjścia neuronów warstwy F_1 stanowią wejścia do neuronów warstwy F_2, z których zwycięzca jest wyznaczany na podstawie największej aktywacji. Tylko zwycięzca może być aktywny, co oznacza, że wynik wynosi 1 dla zwycięzcy i 0 dla całej reszty. Równania domyślnie zawierają zastosowanie reguły 2/3, o której wspomnieliśmy wcześniej, a także sposób wykorzystania kontroli wzmocnienia. Regulacja wzmocnienia ma mieć wartość 1 w fazie określania aktywacji neuronów w warstwie F_2 i 0, jeśli albo nie ma wektora wejściowego, albo wyjście z warstwy F_2 jest propagowane do warstwy F_1.

Inne modele

Rozszerzeniami modelu ART1, który dotyczy wzorców binarnych, są ART2 i ART3. Spośród nich model ART2 kategoryzuje i przechowuje wzorce o wartościach analogowych, a także wzorce binarne, podczas gdy ART3 adresuje problemy obliczeniowe hierarchii.

Implementacja C++

Ponownie, algorytm przetwarzania ART1 podany w Freeman i Skapura jest stosowany w naszej implementacji C++. Naszym celem w programowaniu ART1 jest wyczucie działania tego paradygmatu przy bardzo prostej implementacji programu.

Plik nagłówkowy dla programu C++ dla sieci modeli ART1

Plik nagłówkowy programu C++ dla sieci modelu ART1 to art1net.hpp. Zawiera deklaracje dla dwóch klas, klasy artneuron dla neuronów w modelu ART1 oraz klasy sieci, która jest zadeklarowana jako klasa zaprzyjaźniona w klasie artneuron. Funkcje zadeklarowane w klasie sieci obejmują jedną do wykonania

iteracji dla działania sieci, znalezienie zwycięzcy w danej iteracji oraz jedną do zapytania, czy potrzebny jest reset.

```
//art1net.h   V. Rao,  H. Rao
//Header file for ART1 model network program

#include <iostream.h>
#define MXSIZ 10

class artneuron
{

protected:
        int nnbr;
        int inn,outn;
        int output;
        double activation;
        double outwt[MXSIZ];
        char *name;
        friend class network;

public:
        artneuron() { };
        void getnrn(int,int,int,char *);
```

```
};

class network
{
public:
        int  anmbr,bnmbr,flag,ninpt,sj,so,winr;
        float ai,be,ci,di,el,rho;
        artneuron (anrn)[MXSIZ],(bnrn)[MXSIZ];
        int outs1[MXSIZ],outs2[MXSIZ];
        int lrndptrn[MXSIZ][MXSIZ];
        double acts1[MXSIZ],acts2[MXSIZ];
        double mtrx1[MXSIZ][MXSIZ],mtrx2[MXSIZ][MXSIZ];

        network() { };
        void getnwk(int,int,float,float,float,float,float);
        void prwts1();
        void prwts2();
        int winner(int k,double *v,int);
        void practs1();
        void practs2();
        void prouts1();
        void prouts2();
        void iterate(int *,float,int);
        void asgninpt(int *);
        void comput1(int);
        void comput2(int *);
        void prlrndp();
        void inqreset(int);
        void adjwts1();
        void adjwts2();
};
```

Plik źródłowy programu C++ dla sieci modelowej ART1

Implementacje funkcji zadeklarowanych w pliku nagłówkowym zawarte są w pliku źródłowym programu C++ dla sieci modelu ART1. Ma również główną funkcję, która zawiera specyfikację liczby neuronów w dwóch warstwach sieci, wartości czujności i innych parametrów oraz wektory wejściowe. Zauważ, że jeśli w warstwie jest n neuronów, są one numerowane kolejno od 0 do n-1, a nie od 1 do n w programie C++. Plik źródłowy nazywa się art1net.cpp. Jest skonfigurowany z sześcioma neuronami w warstwie F_1 i siedmioma neuronami w warstwie F_2. Główna funkcja zawiera również parametry potrzebne w algorytmie. Aby zainicjować wagi oddolne, ustawiamy każdą wagę na –0.1 + L/(m – 1 + L), tak aby była większa niż 0 i mniejsza niż L/(m – 1 + L), jak sugerowano wcześniej. Podobnie wagi odgórne są inicjowane przez ustawienie każdej z nich na 0,2 + (B – 1)/D, tak aby była większa niż (B – 1)/D. Jak sugerowano wcześniej, początkowe aktywacje neuronów warstwy F_1 są ustawione na –B/(1 + C). Funkcja restrmax jest zdefiniowana do obliczania maksimum w tablicy, gdy jeden z elementów tablicy nie ma być kandydatem na maksimum. Ułatwia to usunięcie aktualnego zwycięzcy z konkurencji, gdy potrzebny jest reset. Reset jest potrzebny, gdy stopień dopasowania jest mniejszy niż parametr czujności. Funkcja iteracji jest funkcją składową klasy sieci i wykonuje przetwarzanie dla sieci. Funkcja inqreset klasy sieci porównuje parametr czujności ze stopniem dopasowania.

//art1net.cpp V. Rao, H. Rao

```
//Source file for ART1 network program
#include "art1net.h"
int restrmax(int j,double *b,int k)
{
int i,tmp;
for(i=0;i<j;i++){
if(i !=k)
{tmp = i;
i = j;}
}
for(i=0;i<j;i++){
if( (i != tmp)&&(i != k))
{if(b[i]>b[tmp]) tmp = i;}}
return tmp;
}
void artneuron::getnrn(int m1,int m2,int m3, char *y)
{
int i;
name = y;
nnbr = m1;
outn = m2;
inn = m3;
for(i=0;i<outn;++i){
outwt[i] = 0 ;
}
output = 0;
activation = 0.0;
}
void network::getnwk(int k,int l,float aa,float bb,float
cc,float dd,float ll)
{
```

```
anmbr = k;
bnmbr = l;
ninpt = 0;
ai = aa;
be = bb;
ci = cc;
di = dd;
el = ll;
int i,j;
flag = 0;
char *y1="ANEURON", *y2="BNEURON" ;
for(i=0;i<anmbr;++i){
anrn[i].artneuron::getnrn(i,bnmbr,0,y1);}
for(i=0;i<bnmbr;++i){
bnrn[i].artneuron::getnrn(i,0,anmbr,y2);}
float tmp1,tmp2,tmp3;
tmp1 = 0.2 +(be - 1.0)/di;
tmp2 = -0.1 + el/(anmbr - 1.0 +el);
tmp3 = - be/(1.0 + ci);
for(i=0;i<anmbr;++i){
anrn[i].activation = tmp3;
acts1[i] = tmp3;
for(j=0;j<bnmbr;++j){
mtrx1[i][j] = tmp1;
mtrx2[j][i] = tmp2;
anrn[i].outwt[j] = mtrx1[i][j];
bnrn[j].outwt[i] = mtrx2[j][i];
}
}
prwts1();
prwts2();
```

```
practs1();
cout<<"\n";
}
int network::winner(int k,double *v,int kk){
int t1;
t1 = restrmax(k,v,kk);
return t1;
}
void network::prwts1()
{
int i3,i4;
cout<<"\nweights for F1 layer neurons: \n";
for(i3=0;i3<anmbr;++i3){
for(i4=0;i4<bnmbr;++i4){
cout<<anrn[i3].outwt[i4]<<" ";}
cout<<"\n"; }
cout<<"\n";
}
void network::prwts2()
{
int i3,i4;
cout<<"\nweights for F2 layer neurons: \n";
for(i3=0;i3<bnmbr;++i3){
for(i4=0;i4<anmbr;++i4){
cout<<bnrn[i3].outwt[i4]<<" ";};
cout<<"\n"; }
cout<<"\n";
}
void network::practs1()
{
int j;
```

```
cout<<"\nactivations of F1 layer neurons: \n";
for(j=0;j<anmbr;++j){
cout<<acts1[j]<<" ";}
cout<<"\n";
}
void network::practs2()
{
int j;
cout<<"\nactivations of F2 layer neurons: \n";
for(j=0;j<bnmbr;++j){
cout<<acts2[j]<<" ";}
cout<<"\n";
}
void network::prouts1()
{
int j;
cout<<"\noutputs of F1 layer neurons: \n";
for(j=0;j<anmbr;++j){
cout<<outs1[j]<<" ";}
cout<<"\n";
}
void network::prouts2()
{
int j;
cout<<"\noutputs of F2 layer neurons: \n";
for(j=0;j<bnmbr;++j){
cout<<outs2[j]<<" ";}
cout<<"\n";
}
void network::asgninpt(int *b)
{
```

```
int j;
sj = so = 0;
cout<<"\nInput vector is:\n" ;
for(j=0;j<anmbr;++j){
cout<<b[j]<<" ";}
cout<<"\n";
for(j=0;j<anmbr;++j){
sj += b[j];
anrn[j].activation = b[j]/(1.0 +ci +ai*(b[j]+be));
acts1[j] = anrn[j].activation;
if(anrn[j].activation > 0) anrn[j].output = 1;
else
anrn[j].output = 0;
outs1[j] = anrn[j].output;
so += anrn[j].output;
}
practs1();
prouts1();
}
void network::inqreset(int t1)
{
int jj;
flag = 0;
jj = so/sj;
cout<<"\ndegree of match: "<<jj<<" vigilance: "<<rho<<"\n";
if( jj > rho ) flag = 1;
else
{cout<<"winner is "<<t1;
cout<<" reset required \n";}
}
void network::comput1(int k)
```

```
{
int j;
for(j=0;j<bnmbr;++j){
int ii1;
double c1 = 0.0;
cout<<"\n";
for(ii1=0;ii1<anmbr;++ii1){
c1 += outs1[ii1] * mtrx2[j][ii1];
}
bnrn[j].activation = c1;
acts2[j] = c1;};
winr = winner(bnmbr,acts2,k);
cout<<"winner is "<<winr;
for(j=0;j<bnmbr;++j){
if(j == winr) bnrn[j].output = 1;
else bnrn[j].output = 0;
outs2[j] = bnrn[j].output;
}
practs2();
prouts2();
}
void network::comput2(int *b)
{
double db[MXSIZ];
double tmp;
so = 0;
int i,j;
for(j=0;j<anmbr;++j){
db[j] =0.0;
for(i=0;i<bnmbr;++i){
db[j] += mtrx1[j][i]*outs2[i];};
```

```
tmp = b[j] + di*db[j];
acts1[j] = (tmp - be)/(ci +1.0 +ai*tmp);
anrn[j].activation = acts1[j];
if(anrn[j].activation > 0) anrn[j].output = 1;
else anrn[j].output = 0;
outs1[j] = anrn[j].output;
so += anrn[j].output;
}
cout<<"\n";
practs1();
prouts1();
}
void network::adjwts1()
{
int i;
for(i=0;i<anmbr;++i){
if(outs1[i] >0) {mtrx1[i][winr] = 1.0;}
else
{mtrx1[i][winr] = 0.0;}
anrn[i].outwt[winr] = mtrx1[i][winr];}
prwts1();
}
void network::adjwts2()
{
int i;
cout<<"\nwinner is "<<winr<<"\n";
for(i=0;i<anmbr;++i){
if(outs1[i] > 0) {mtrx2[winr][i] = el/(so + el -1);}
else
{mtrx2[winr][i] = 0.0;}
bnrn[winr].outwt[i] = mtrx2[winr][i];}
```

```
prwts2();
}
void network::iterate(int *b,float rr,int kk)
{
int j;
rho = rr;
flag = 0;
asgninpt(b);
comput1(kk);
comput2(b);
inqreset(winr);
if(flag == 1){
ninpt ++;
adjwts1();
adjwts2();
int j3;
for(j3=0;j3<anmbr;++j3){
lrndptrn[ninpt][j3] = b[j3];}
prlrndp();
}
else
{
for(j=0;j<bnmbr;++j){
outs2[j] = 0;
bnrn[j].output = 0;}
iterate(b,rr,winr);
}
}
void network::prlrndp()
{
int j;
```

```
cout<<"\nlearned vector # "<<ninpt<<" :\n";
for(j=0;j<anmbr;++j){
cout<<lrndptrn[ninpt][j]<<" ";}
cout<<"\n";
}
void main()
{
int ar = 6, br = 7, rs = 8;
float aa = 2.0,bb = 2.5,cc = 6.0,dd = 0.85,ll = 4.0,rr =
0.95;
int inptv[][6]={0,1,0,0,0,0,1,0,1,0,1,0,0,0,0,0,1,0,1,0,1,0,\
1,0};
cout<<"\n\nTHIS PROGRAM IS FOR AN -ADAPTIVE RESONANCE THEORY\
1 - NETWORK.\n";
cout<<"THE NETWORK IS SET UP FOR ILLUSTRATION WITH "<<ar<<" \
INPUT NEURONS,\n";
cout<<" AND "<<br<<" OUTPUT NEURONS.\n";
static network bpn;
bpn.getnwk(ar,br,aa,bb,cc,dd,ll) ;
bpn.iterate(inptv[0],rr,rs);
bpn.iterate(inptv[1],rr,rs);
bpn.iterate(inptv[2],rr,rs);
bpn.iterate(inptv[3],rr,rs);
}
```

Wyjście programu

Podczas próbnego uruchomienia programu używane są cztery wektory wejściowe, które są określone w funkcji main. Dane wyjściowe nie wymagają wyjaśnień. Tylko w tym tekście zamieściliśmy kilka uwag dotyczących wyników. Te komentarze są ujęte w ciągi gwiazdek. W rzeczywistości nie są one częścią wyników programu. Tabela przedstawia podsumowanie kategoryzacji wejść wykonanych przez sieć. Należy pamiętać, że numeracja neuronów w dowolnej warstwie, która ma n neuronów, wynosi od 0 do n – 1, a nie od 1 do n.

input	winner in F_2 layer
0 1 0 0 0 0	0, no reset
1 0 1 0 1 0	1, no reset
0 0 0 0 1 0	1, after reset 2
1 0 1 0 1 0	1, after reset 3

Wzorzec wejściowy 0 0 0 0 1 0 jest uważany za podzbiór wzorca 1 0 1 0 1 0 w tym sensie, że w dowolnej pozycji pierwszy wzorzec ma 1, drugi wzorzec również ma 1. Oczywiście drugi wzorzec ma jedynki również na innych pozycjach. Jednocześnie wzorzec 1 0 1 0 1 0 jest uważany za nadzbiór wzorca 0 0 0 0 1 0. Powód, dla którego wzorzec 1 0 1 0 1 0 jest powtarzany jako dane wejściowe po wzorze 0 0 0 0 1 0 jest przetwarzane, aby zobaczyć, co dzieje się z tym nadzbiorem. W obu przypadkach stopień dopasowania jest niższy od parametru czujności i konieczne jest zresetowanie. Oto wynik działania programu:

```
THIS PROGRAM IS FOR AN ADAPTIVE RESONANCE THEORY
1-NETWORK. THE NETWORK IS SET UP FOR ILLUSTRATION WITH SIX INPUT NEURONS
AND SEVEN OUTPUT NEURONS.
**************************************************************
Initialization of connection weights and F1 layer activations. F1 layer
connection weights are all chosen to be equal to a random value subject
to the conditions given in the algorithm. Similarly, F2 layer connection
weights are all chosen to be equal to a random value subject to the
conditions given in the algorithm.
```

```
****************************************************************
weights for F1 layer neurons:
1.964706  1.964706  1.964706  1.964706  1.964706  1.964706  1.964706
1.964706  1.964706  1.964706  1.964706  1.964706  1.964706  1.964706
1.964706  1.964706  1.964706  1.964706  1.964706  1.964706  1.964706
1.964706  1.964706  1.964706  1.964706  1.964706  1.964706  1.964706
1.964706  1.964706  1.964706  1.964706  1.964706  1.964706  1.964706
1.964706  1.964706  1.964706  1.964706  1.964706  1.964706  1.964706

weights for F2 layer neurons:
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444

activations of F1 layer neurons:
-0.357143 -0.357143 -0.357143 -0.357143 -0.357143 -0.357143
****************************************************************
A new input vector and a new iteration
****************************************************************
Input vector is:
0 1 0 0 0 0

activations of F1 layer neurons:
0   0.071429   0   0   0   0

outputs of F1 layer neurons:
0   1   0   0   0   0

winner is 0
activations of F2 layer neurons:
0.344444   0.344444   0.344444   0.344444   0.344444   0.344444   0.344444

outputs of F2 layer neurons:
1   0   0   0   0   0   0


activations of F1 layer neurons:
-0.080271   0.013776   -0.080271   -0.080271   -0.080271   -0.080271

outputs of F1 layer neurons:
0   1   0   0   0   0
****************************************************************
Top-down and bottom-up outputs at F1 layer match, showing resonance.
****************************************************************
degree of match: 1 vigilance:  0.95

weights for F1 layer neurons:
```

```
0  1.964706  1.964706  1.964706  1.964706  1.964706  1.964706
1  1.964706  1.964706  1.964706  1.964706  1.964706  1.964706
0  1.964706  1.964706  1.964706  1.964706  1.964706  1.964706
0  1.964706  1.964706  1.964706  1.964706  1.964706  1.964706
0  1.964706  1.964706  1.964706  1.964706  1.964706  1.964706
0  1.964706  1.964706  1.964706  1.964706  1.964706  1.964706

winner is 0

weights for F2 layer neurons:
0  1  0  0  0  0
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444

learned vector # 1  :
0  1  0  0  0  0
*****************************************************************
A new input vector and a new iteration
*****************************************************************
Input vector is:
1 0 1 0 1 0

activations of F1 layer neurons:
0.071429   0   0.071429   0   0.071429   0

outputs of F1 layer neurons:
1   0   1   0   1   0

winner is 1
activations of F2 layer neurons:
0   1.033333   1.033333   1.033333   1.033333   1.033333   1.033333

outputs of F2 layer neurons:
0   1   0   0   0   0   0

activations of F1 layer neurons:
0.013776   -0.080271   0.013776   -0.080271   0.013776   -0.080271

outputs of F1 layer neurons:
1   0   1   0   1   0
*****************************************************************
Top-down and bottom-up outputs at F1 layer match,
showing resonance.
*****************************************************************
degree of match: 1 vigilance:  0.95

weights for F1 layer neurons:
```

```
0  1  1.964706  1.964706  1.964706  1.964706  1.964706
1  0  1.964706  1.964706  1.964706  1.964706  1.964706
0  1  1.964706  1.964706  1.964706  1.964706  1.964706
0  0  1.964706  1.964706  1.964706  1.964706  1.964706
0  1  1.964706  1.964706  1.964706  1.964706  1.964706
0  0  1.964706  1.964706  1.964706  1.964706  1.964706

winner is 1

weights for F2 layer neurons:
0  1  0  0  0  0
0.666667  0  0.666667  0  0.666667  0
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444

learned vector # 2  :
1  0  1  0  1  0
**********************************************************************
A new input vector and a new iteration
**********************************************************************
Input vector is:
0 0 0 0 1 0

activations of F1 layer neurons:
0   0   0   0   0.071429   0

outputs of F1 layer neurons:
0   0   0   0   1   0

winner is 1
activations of F2 layer neurons:
0   0.666667   0.344444   0.344444   0.344444   0.344444   0.3

outputs of F2 layer neurons:
0   1   0   0   0   0   0

activations of F1 layer neurons:
-0.189655   -0.357143   -0.189655   -0.357143   -0.060748   -0.357143

outputs of F1 layer neurons:
0   0   0   0   0   0

degree of match: 0 vigilance:  0.95
winner is 1 reset required
**********************************************************************
Input vector repeated after reset, and a new iteration
**********************************************************************
```

```
Input vector is:
0 0 0 0 1 0

activations of F1 layer neurons:
0   0   0   0   0.071429   0

outputs of F1 layer neurons:
0   0   0   0   1   0

winner is 2
activations of F2 layer neurons:
0   0.666667   0.344444   0.344444   0.344444   0.344444   0.344444
outputs of F2 layer neurons:
0   0   1   0   0   0   0

      activations of F1 layer neurons:
-0.080271   -0.080271   -0.080271   -0.080271   0.013776   -0.080271

outputs of F1 layer neurons:
0   0   0   0   1   0
****************************************************************
Top-down and bottom-up outputs at F1 layer match, showing resonance.
****************************************************************
degree of match: 1 vigilance:  0.95

weights for F1 layer neurons:
0  1  0  1.964706  1.964706  1.964706  1.964706
1  0  0  1.964706  1.964706  1.964706  1.964706
0  1  0  1.964706  1.964706  1.964706  1.964706
0  0  0  1.964706  1.964706  1.964706  1.964706
0  1  1  1.964706  1.964706  1.964706  1.964706
0  0  0  1.964706  1.964706  1.964706  1.964706

winner is 2

weights for F2 layer neurons:
0  1  0  0  0  0
0.666667  0  0.666667  0  0.666667  0

0  0  0  0  1  0
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444

learned vector # 3  :
0  0  0  0  1  0
****************************************************************
An old (actually the second above) input vector is retried after trying a
subset vector, and a new iteration
****************************************************************
```

```
Input vector is:
1 0 1 0 1 0

activations of F1 layer neurons:
0.071429   0   0.071429   0   0.071429   0

outputs of F1 layer neurons:
1   0   1   0   1   0

winner is 1
activations of F2 layer neurons:
0   2   1   1.033333   1.033333   1.033333   1.03333

outputs of F2 layer neurons:
0   1   0   0   0   0   0

activations of F1 layer neurons:
-0.060748   -0.357143   -0.060748   -0.357143   -0.060748   -0.357143

outputs of F1 layer neurons:
0   0   0   0   0   0

degree of match: 0 vigilance:  0.95
winner is 1 reset required
*****************************************************************
Input vector repeated after reset, and a new iteration
*****************************************************************
Input vector is:
1 0 1 0 1 0

activations of F1 layer neurons:
0.071429   0   0.071429   0   0.071429   0

outputs of F1 layer neurons:
1   0   1   0   1   0

winner is 3
activations of F2 layer neurons:

0   2   1   1.033333   1.033333   1.033333   1.033333

outputs of F2 layer neurons:
0   0   0   1   0   0   0

activations of F1 layer neurons:
0.013776   -0.080271   0.013776   -0.080271   0.013776   -0.080271

outputs of F1 layer neurons:
1   0   1   0   1   0
*****************************************************************
Top-down and Bottom-up outputs at F1layer match, showing resonance.
*****************************************************************
```

```
degree of match: 1 vigilance:  0.95

weights for F1 layer neurons:
0  1  0  1  1.964706  1.964706  1.964706
1  0  0  0  1.964706  1.964706  1.964706
0  1  0  1  1.964706  1.964706  1.964706
0  0  0  0  1.964706  1.964706  1.964706
0  1  1  1  1.964706  1.964706  1.964706
0  0  0  0  1.964706  1.964706  1.964706

winner is 3

weights for F2 layer neurons:
0  1  0  0  0  0
0.666667  0  0.666667  0  0.666667  0
0  0  0  0  1  0
0.666667  0  0.666667  0  0.666667  0
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444
0.344444  0.344444  0.344444  0.344444  0.344444  0.344444

learned vector # 4  :
1  0  1  0  1  0
```

Podsumowanie

Przedstawiono tu podstawy adaptacyjnej teorii rezonansu Grossberga i Carpentera oraz implementację sieci neuronowej modelowanej dla tej teorii w języku C++. Jest to elegancka teoria, która odnosi się do dylematu stabilność-plastyczność. Sieć opiera się na rezonansie. Jest to samoorganizująca się sieć i dokonuje kategoryzacji poprzez kojarzenie poszczególnych neuronów warstwy F_2 z indywidualnymi wzorcami. Dzięki zastosowaniu tzw. zasady 2/3 zapewnia stabilność wzorców uczenia się.