Aplikacja do rozpoznawania wzorców

Korzystanie z mapy funkcji Kohonena

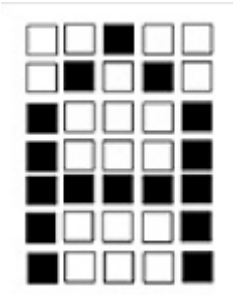
Tu użyjesz programu Kohonena, opracowanego w Części 11, do rozpoznawania wzorców. Zmodyfikujesz program Kohonen do wyświetlania wzorów.

Przykładowy problem: rozpoznawanie znaków

Problem przedstawiony tutaj polega na rozpoznawaniu lub kategoryzowaniu znaków alfabetycznych. Wprowadzisz różne znaki alfabetyczne do mapy Kohonena i nauczysz sieć, aby rozpoznawała je jako oddzielne kategorie. Ten program może być użyty do wypróbowania innych eksperymentów, które zostaną omówione na końcu.

Reprezentowanie znaków

Każdy znak jest reprezentowany przez siatkę 5×7 pikseli. Używamy graficznych znaków drukowania rozszerzonego zestawu znaków IBM ASCII, aby pokazać wynik w skali szarości dla każdego piksela. Na przykład, aby przedstawić literę A, możesz użyć wzoru pokazanego na rysunku. Tutaj zaczernione pola reprezentują wartość 1, podczas gdy puste pola oznaczają zero. W ten sposób możesz reprezentować wszystkie znaki za pomocą mapy binarnej składającej się z 35 wartości pikseli.



Litera A jest reprezentowana przez wartości:

0 0 **1** 0 0

0 1 0 1 0

1 0 0 0 **1**

1 0 0 0 **1**

1 1 1 1 1

1 0 0 0 **1**

1 0 0 0 **1**

Do użycia w programie Kohonen musimy serializować wiersze, tak aby wszystkie wpisy pojawiały się w jednym wierszu. Dla znaków A i X otrzymasz następujące wpisy w pliku wejściowym input.dat:

0 0 1 0 0 0 1 0 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 1

<< litera A

1 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 1

<< litera X

Monitorowanie wag

Przedstawimy mapę Kohonena z wieloma takimi znakami i znajdziemy odpowiedź na wyjściu. Będziesz mógł obserwować mapę Kohonena, która przechodzi przez swoje cykle i uczy się wzorców wejściowych. W tym samym czasie powinieneś być w stanie obserwować wektory wag dla zwycięskich neuronów, aby zobaczyć wzór rozwijający się w wagach. Pamiętaj, że w przypadku mapy Kohonena wektory wag mają tendencję do dostosowywania się do wektorów wejściowych. Po chwili zauważysz, że wektor wag dla danych wejściowych będzie przypominał wzorzec wejściowy, który kategoryzujesz.

Reprezentujący wektor wagi

Chociaż wartości włączenia i wyłączenia są w porządku dla wspomnianych wektorów wejściowych, musisz zobaczyć wartości skali szarości dla wektora wagi. Można to osiągnąć poprzez kwantyzowanie wektora wagowego w czterech przedziałach, z których każdy jest reprezentowany przez inny znak graficzny ASCII, jak pokazano w tabeli

<= 0 : Biały prostokąt (spacja)

0 < waga <= 0,25 : Prostokąt z jasną kropką

0,25 < waga <= 0,50: prostokąt ze średnimi kropkami

0.50 < waga <= 0.75: prostokąt w ciemne kropki

waga> 0,75: czarny prostokąt

Wartości ASCII dla znaków graficznych, które mają być użyte, są wymienione w tabeli

Biały prostokąt: 255

Prostokąt z jasnymi kropkami: 176

Biały prostokąt: 255

Prostokąt z jasnymi kropkami: 176

Rozwój kodu C++

Zmiany w programie Kohonena są stosunkowo niewielkie. Poniższa lista wskazuje te zmiany.

Zmiany w programie Kohonena

Pierwszą zmianą, jaką należy wprowadzić, jest definicja klasy Kohonen_network. Znajduje się on w pliku layerk.h, pokazanym na listingu 1.

Listing 1 Zaktualizowany plik layerk.h

```
class Kohonen_network

{

private:

layer *layer_ptr[2];

int layer_size[2];

int neighborhood_size;

public:

Kohonen_network();

~Kohonen_network();
```

```
void get_layer_info();

void set_up_network(int);

void randomize_weights();

void update_neigh_size(int);

void update_weights(const float);

void list_weights();

void list_outputs();

void get_next_vector(FILE *);

void process_next_pattern();

float get_win_dist();

int get_win_index();

void display_input_char();

void display_winner_weights();

};
```

Nowe funkcje składowe są oznaczone kursywą. Funkcje display_input_char() i display_winner_weights() są używane do wyświetlania map wejściowych i map wag na ekranie, aby zobaczyć, jak mapa znaków wagi zbiega się z mapą wejściową. Implementacja tych funkcji znajduje się w pliku layerk.cpp. Część tego pliku zawierająca te funkcje jest pokazana na listingu 2.

Listing 2 Dodatki do pliku implementacyjnego layerk.cpp

```
void Kohonen_network::display_input_char()

{

int i, num_inputs;

unsigned char ch;

float temp;

int col=0;

float * inputptr;

num_inputs=layer_ptr[1]->num_inputs;

inputptr = layer_ptr[1]->inputs;

// we've got a 5x7 character to display

for (i=0; i<num_inputs; i++)

{

temp = *(inputptr);

if (temp <= 0)

ch=255;// blank
```

```
else if ((temp > 0) && (temp <= 0.25))
ch=176; // dotted rectangle -light
else if ((temp > 0.25) && (temp <= 0.50))
ch=177; // dotted rectangle -medium
else if ((temp >0.50) && (temp <= 0.75))
ch=178; // dotted rectangle -dark
else if (temp > 0.75)
ch=219; // filled rectangle
printf("%c",ch); //fill a row
col++;
if ((col % 5)==0)
printf("\n"); // new row
inputptr++;
}
printf("\n\n\n");
}
void Kohonen_network::display_winner_weights()
{
int i, k;
unsigned char ch;
float temp;
float * wmat;
int col=0;
int win_index;
int num_inputs, num_outputs;
num_inputs= layer_ptr[1]->num_inputs;
wmat = ((Kohonen_layer*)layer_ptr[1])
->weights;
win_index=((Kohonen_layer*)layer_ptr[1])
->winner_index;
num_outputs=layer_ptr[1]->num_outputs;
// we've got a 5x7 character to display
for (i=0; i<num_inputs; i++)
```

```
{
k= i*num_outputs;
temp = wmat[k+win_index];
if (temp <= 0)
ch=255;// blank
else if ((temp > 0) && (temp <= 0.25))
ch=176; // dotted rectangle -light
else if ((temp > 0.25) && (temp <= 0.50))
ch=177; // dotted rectangle -medium
else if ((temp > 0.50) && (temp <= 0.75))
ch=178; // dotted rectangle -dark
else if (temp > 0.75)
ch=219; // filled rectangle
printf("%c",ch); //fill a row
col++;
if ((col % 5)==0)
printf("\n"); // new row
}
printf("\n\n");
printf("————————————-\n");
}
```

Ostatnią zmianą, jaką należy wprowadzić, jest plik kohonen.cpp. Nowy plik nazywa się pattern.cpp i jest pokazany na listingu 3.

Listing 3 The implementation file pattern.cpp

```
// pattern.cpp V. Rao, H. Rao
// Kohonen map for pattern recognition
#include "layerk.cpp"
#define INPUT_FILE "input.dat"
#define OUTPUT_FILE "kohonen.dat"
#define dist_tol 0.001
#define wait_cycles 10000 // creates a pause to
// view the character maps
void main()
```

```
{
int neighborhood_size, period;
float avg_dist_per_cycle=0.0;
float dist_last_cycle=0.0;
float avg_dist_per_pattern=100.0; // for the latest cycle
float dist_last_pattern=0.0;
float total_dist;
float alpha;
unsigned startup;
int max_cycles;
int patterns_per_cycle=0;
int total_cycles, total_patterns;
int i;
// create a network object
Kohonen_network knet;
FILE * input_file_ptr, * output_file_ptr;
// open input file for reading
if ((input_file_ptr=fopen(INPUT_FILE,”r”))==NULL)
{
cout << “problem opening input file\n”;
exit(1);
}
// open writing file for writing
if ((output_file_ptr=fopen(OUTPUT_FILE,”w”))==NULL)
{
cout << “problem opening output file\n”;
exit(1);
}
// ——————————————————————-
// Read in an initial values for alpha, and the
// neighborhood size.
// Both of these parameters are decreased with
// time. The number of cycles to execute before
```

```
// decreasing the value of these parameters is
// called the period. Read in a value for the
// period.
// ————————————————————————-
cout << " Please enter initial values for:\n";
cout << "alpha (0.01-1.0),\n";
cout << "and the neighborhood size (integer between 0\
and 50)\n";
cout << "separated by spaces, e.g. 0.3 5 \n ";
cin >> alpha >> neighborhood_size ;
cout << "\nNow enter the period, which is the\n";
cout << "number of cycles after which the values\n";
cout << "for alpha the neighborhood size are \
decremented\n";
cout << "choose an integer between 1 and 500 , e.g. \ 50 \n";
cin >> period;
// Read in the maximum number of cycles
// each pass through the input data file is a cycle
cout << "\nPlease enter the maximum cycles for the
simulation\n";
cout << "A cycle is one pass through the data set.\n";
cout << "Try a value of 500 to start with\n\n";
cin >> max_cycles;
// the main loop
//
// continue looping until the average distance is less than
// the tolerance specified at the top of this file
// , or the maximum number of
// cycles is exceeded;
// initialize counters
total_cycles=0; // a cycle is once through all the input data
total_patterns=0; // a pattern is one entry in the input data
// get layer information
```

```
knet.get_layer_info();
// set up the network connections
knet.set_up_network(neighborhood_size);
// initialize the weights
// randomize weights for the Kohonen layer
// note that the randomize function for the
// Kohonen simulator generates
// weights that are normalized to length = 1
knet.randomize_weights();
// write header to output file
fprintf(output_file_ptr,
“cycle\tpattern\twin index\tneigh_size\\
tavg_dist_per_pattern\n”);
fprintf(output_file_ptr,
“————————————————————————\n”);
startup=1;
total_dist=0;
while (
(avg_dist_per_pattern > dist_tol)
&& (total_cycles < max_cycles)
|| (startup==1)
)
{
startup=0;
dist_last_cycle=0; // reset for each cycle
patterns_per_cycle=0;
// process all the vectors in the datafile
while (!feof(input_file_ptr))
{
knet.get_next_vector(input_file_ptr);
// now apply it to the Kohonen network
knet.process_next_pattern();
dist_last_pattern=knet.get_win_dist();
```

```
// print result to output file
fprintf(output_file_ptr,"%i\t%i\t%i\t\t%i\t\t%f\n",
total_cycles,total_patterns,knet.get_win_index(),
neighborhood_size,avg_dist_per_pattern);
// display the input character and the
// weights for the winner to see match
knet.display_input_char();
knet.display_winner_weights();
// pause for a while to view the
// character maps
for (i=0; i<wait_cycles; i++)
{;}
total_patterns++;
// gradually reduce the neighborhood size
// and the gain, alpha
if (((total_cycles+1) % period) == 0)
{
if (neighborhood_size > 0)
neighborhood_size —;
knet.update_neigh_size(neighborhood_size);
if (alpha>0.1)
alpha -= (float)0.1;
}
patterns_per_cycle++;
dist_last_cycle += dist_last_pattern;
knet.update_weights(alpha);
dist_last_pattern = 0;
}
avg_dist_per_pattern= dist_last_cycle/patterns_per_cycle;
total_dist += dist_last_cycle;
total_cycles++;
fseek(input_file_ptr, 0L, SEEK_SET); // reset the file
pointer
```

```
// to the beginning of
// the file
} // end main loop
cout << "\n\n\n\n\n\n\n\n\n\n";
cout << "————————————————————————————————\n";
cout << " done \n";
avg_dist_per_cycle= total_dist/total_cycles;
cout << "\n";
cout << "——>average dist per cycle = " << avg_dist_per_cycle << " <—-\n";
cout << ">dist last cycle = " << dist_last_cycle << " < \n";
cout << "->dist last cycle per pattern= " <<
avg_dist_per_pattern << " <—-\n";
cout << "——->total cycles = " << total_cycles << " <—-\n";
cout << "———————>total patterns = " <<
total_patterns << " <—-\n";
cout << "————————————————————————————————\n";
// close the input file
fclose(input_file_ptr);
}
```

Zmiany w programie zaznaczono kursywą. Skompiluj ten program, kompilując i tworząc plik pattern.cpp, po zmodyfikowaniu plików layerk.cpp i layerk.h, jak wskazano wcześniej.

Testowanie programu

Uruchommy przykład, dla którego utworzyliśmy plik wejściowy. Mamy plik input.dat ze zdefiniowanymi znakami A i X. Przebieg programu z tymi danymi wejściowymi jest przedstawiony w następujący sposób:

```
Please enter initial values for:
alpha (0.01-1.0),
and the neighborhood size (integer between 0 and 50)
separated by spaces, e.g., 0.3 5
0.3 5
Now enter the period, which is the
number of cycles after which the values
for alpha the neighborhood size are decremented
choose an integer between 1 and 500, e.g., 50
```

50

Please enter the maximum cycles for the simulation

A cycle is one pass through the data set.

Try a value of 500 to start with

500

Enter in the layer sizes separated by spaces.

A Kohonen network has an input layer

followed by a Kohonen (output) layer

35 100

Wynik programu jest jak zwykle zawarty w pliku kohonen.dat. To pokazuje następujący wynik

cycle	pattern	win index	neigh_size	avg_dist_per_pattern
0	0	42	5	100.000000
0	1	47	5	100.000000
1	2	42	5	0.508321
1	3	47	5	0.508321
2	4	40	5	0.742254
2	5	47	5	0.742254
3	6	40	5	0.560121
3	7	47	5	0.560121
4	8	40	5	0.392084
4	9	47	5	0.392084
5	10	40	5	0.274459
5	11	47	5	0.274459
6	12	40	5	0.192121
6	13	47	5	0.192121
7	14	40	5	0.134485
7	15	47	5	0.134485
8	16	40	5	0.094139
8	17	47	5	0.094139
9	18	40	5	0.065898
9	19	47	5	0.065898
10	20	40	5	0.046128
10	21	47	5	0.046128
11	22	40	5	0.032290
11	23	47	5	0.032290
12	24	40	5	0.022603
12	25	47	5	0.022603
13	26	40	5	0.015822
13	27	47	5	0.015822
14	28	40	5	0.011075
14	29	47	5	0.011075
15	30	40	5	0.007753
15	31	47	5	0.007753
16	32	40	5	0.005427
16	33	47	5	0.005427
17	34	40	5	0.003799
17	35	47	5	0.003799
18	36	40	5	0.002659
18	37	47	5	0.002659
19	38	40	5	0.001861
19	39	47	5	0.001861
20	40	40	5	0.001303
20	41	47	5	0.001303

Tolerancja odległości została ustawiona na 0,001 dla tego programu, a program był w stanie osiągnąć zbieżność do tej wartości. Oba wejścia zostały pomyślnie zaklasyfikowane do dwóch różnych zwycięskich neuronów wyjściowych. Na rysunkach poniższych widzisz dwie migawki wektorów wejściowych i wagowych, które znajdziesz w tym programie. Wektor wagi przypomina dane wejściowe, jak widać, ale nie jest to dokładna replikacja.

Generalizacja kontra zapamiętywanie

Jak wspomniano w Części 11, w rzeczywistości nie chcesz dokładnej replikacji wzorca wejściowego dla wektora wag. Sprowadzałoby się to do zapamiętywania wzorców wejściowych bez możliwości uogólniania. Na przykład typowym zastosowaniem tego systemu klasyfikatorów alfabetycznych byłoby użycie go do przetwarzania zaszumionych danych, takich jak odręczne znaki. W takim przypadku potrzebna byłaby duża szerokość geograficzna w ustalenie zakresu zajęć na literę A.

Dodawanie znaków

Następnym krokiem programu jest dodawanie znaków i sprawdzanie, do jakich kategorii trafiają. Istnieje wiele znaków alfabetycznych, które wyglądają podobnie, na przykład H i B. Możesz oczekiwać, że klasyfikator Kohonen zgrupuje te podobne postacie w tej samej klasie. Teraz modyfikujemy plik input.dat, aby dodać znaki H, B i I. Nowy plik input.dat jest pokazany w następujący sposób.

```
0 0 1 0 0   0 1 0 1 0  1 0 0 0 1  1 0 0 0 1  1 1 1 1 1  1 0 0 0 1
 1 0 0 0 1
1 0 0 0 1   0 1 0 1 0  0 0 1 0 0  0 0 1 0 0  0 0 1 0 0  0 1 0 1 0
 1 0 0 0 1
1 0 0 0 1   1 0 0 0 1  1 0 0 0 1  1 1 1 1 1  1 0 0 0 1  1 0 0 0 1
 1 0 0 0 1
1 1 1 1 1   1 0 0 0 1  1 0 0 0 1  1 1 1 1 1  1 0 0 0 1  1 0 0 0 1
 1 1 1 1 1
0 0 1 0 0   0 0 1 0 0  0 0 1 0 0  0 0 1 0 0  0 0 1 0 0  0 0 1 0 0
 0 0 1 0 0
```

Dane wyjściowe wykorzystujące ten plik wejściowy są pokazane w następujący sposób.

```
-------------------------------
       done
-->average dist per cycle = 0.732607 <--
-->dist last cycle = 0.00360096 <--
->dist last cycle per pattern= 0.000720192 <--
------->total cycles = 37 <--
------->total patterns = 185 <--
-------------------------------
```

Plik kohonen.dat z wartościami wyjściowymi jest teraz pokazany w następujący sposób.

cycle	pattern	win index	neigh_size	avg_dist_per_pattern
0	0	69	5	100.000000
0	1	93	5	100.000000
0	2	18	5	100.000000
0	3	18	5	100.000000
0	4	78	5	100.000000
1	5	69	5	0.806743
1	6	93	5	0.806743
1	7	18	5	0.806743
1	8	18	5	0.806743
1	9	78	5	0.806743
2	10	69	5	0.669678
2	11	93	5	0.669678
2	12	18	5	0.669678
2	13	18	5	0.669678
2	14	78	5	0.669678
3	15	69	5	0.469631
3	16	93	5	0.469631
3	17	18	5	0.469631
3	18	18	5	0.469631
3	19	78	5	0.469631
4	20	69	5	0.354791
4	21	93	5	0.354791
4	22	18	5	0.354791
4	23	18	5	0.354791
4	24	78	5	0.354791
5	25	69	5	0.282990
5	26	93	5	0.282990
5	27	18	5	0.282990
...				
35	179	78	5	0.001470
36	180	69	5	0.001029
36	181	93	5	0.001029
36	182	13	5	0.001029
36	183	19	5	0.001029
36	184	78	5	0.001029

Ponownie sieć nie ma problemu z klasyfikacją tych wektorów.

Do cyklu 21 zarówno H, jak i B były klasyfikowane jako neuron wyjściowy 18. Zdolność do rozróżniania tych wektorów jest w dużej mierze spowodowana małą tolerancją, którą przypisaliśmy jako kryterium terminacji.

Inne eksperymenty do wypróbowania

Możesz wypróbować inne eksperymenty z programem. Na przykład możesz powtórzyć plik wejściowy, ale ze zmienioną kolejnością wpisów. Innymi słowy, te same dane wejściowe można prezentować wiele razy w różnej kolejności. To faktycznie pomaga sieci Kohonen szybciej trenować.

Możesz spróbować zastosować zniekształcone wersje znaków, aby sprawdzić, czy sieć je rozróżnia. Podobnie jak w programie backpropagation, możesz zapisać ciężary w pliku ciężarów, aby zamrozić stan treningu, a następnie zastosować nowe dane wejściowe. Możesz wprowadzić wszystkie znaki od A do Z i zobaczyć wynikową klasyfikację. Czy musisz trenować na wszystkich postaciach lub podzbiorze? Możesz zmienić rozmiar warstwy Kohonena. Ile neuronów potrzebujesz, aby rozpoznać cały alfabet? Nie ma ograniczeń dotyczących korzystania z wejść cyfrowych 1 i 0, jak używaliśmy. Możesz zastosować wartości analogowe w skali szarości. Program wyświetli wzorzec wejściowy zgodnie z ustawionymi poziomami kwantyzacji. Ten zestaw można rozbudować, a do wyświetlania kolejnych poziomów można użyć interfejsu graficznego. Możesz wtedy spróbować rozpoznawania wzorców dowolnych obrazów, ale pamiętaj, że czas przetwarzania gwałtownie wzrośnie wraz z liczbą użytych neuronów. Liczba wybranych neuronów wejściowych jest podyktowana rozdzielczością obrazu, chyba że filtrujesz i/lub podpróbkujesz obraz przed przedstawieniem go w sieci. Filtrowanie to proces wykorzystujący rodzaj funkcji uśredniania stosowanej do grup pikseli. Podpróbkowanie to proces wybierania niższej rozdzielczości obrazu poprzez wybranie mniejszej liczby pikseli niż obraz źródłowy. Jeśli zaczniesz od obrazu, który ma 100 × 100 pikseli, możesz podpróbkować ten obraz 2:1 w każdym kierunku, aby uzyskać obraz o rozmiarze jednej czwartej, czyli 50 × 50 pikseli. To, czy wyrzucisz co drugi piksel, aby uzyskać tę rozdzielczość wyjściową, czy zastosujesz filtr, zależy od Ciebie. Możesz uśrednić każde dwa piksele, aby otrzymać jeden piksel wyjściowy jako przykład bardzo prostego filtra.

Podsumowanie

Poniższa lista podkreśla ważne cechy programu Kohonen, których nauczyłeś się tutaj.

• Przedstawiono prosty program do rozpoznawania znaków przy użyciu mapy cech Kohonena.

• Wektory wejściowe i wektory wag zostały wyświetlone, aby pokazać zbieżność i zauważyć podobieństwo między tymi dwoma wektorami.

• W miarę postępu treningu wektor wagi dla neuronu zwycięzcy przypomina mapę znaków wejściowych.