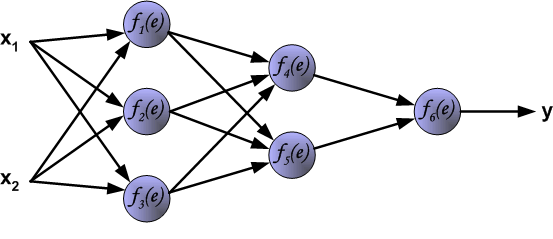
**Sprawozdanie nr 3**

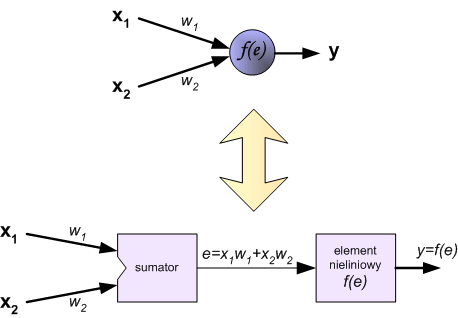
**Temat​ ​ćwiczenia:** Budowa i działanie sieci wielowarstwowej typu feedforward​.

​**Cel​ ​ćwiczenia:**  Poznanie budowy i działania wielowarstwowych sieci neuronowych poprzez uczenie kształtu wykresu funkcji matematycznej z użyciem algorytmu wstecznej propagacji błędu.

Sieć neuronowa typu Feedforward:

Jest to struktura w której istnieje ściśle określony kierunek przepływu sygnałów, od pewnego ustalonego wejścia do wyjścia, na którym sieć podaje ustalone rozwiązanie.   
 Sieć tego typu musi mieć przynajmniej dwie warstwy: wejściową i wyjściową. Jednak wiele sieci posiada warstwy pośredniczące ,ukryte, które tworzą dodatkową strukturę przetwarzającą informację.





**Algorytm wstecznej propagacji błędu:**

Każdy neuron składa się z elementu sumującego iloczyny wag i sygnałów wejściowych oraz z elementu nieliniowego, którego charakterystyka przejściowa jest funkcją aktywacji neuronu. Proces uczenia się przebiega w ciągu kolejnych iteracji ,dla zmieniających się danych uczących, przy czym w każdym kroku wagi połączeń ulegają modyfikacji.   
Przesyłanie sygnałów następuje przez warstwę wejściową ->warstwę ukrytą ->warstwę wyjściową.   
W kolejnym kroku porównuje się wartość oczekiwaną z wartością sygnału wyjściowego.  
 Różnica tych wartości to sygnał błędu warstwy wyjściowej.  
 Dla neuronów leżących w warstwach ukrytych nie ma możliwości bezpośredniego określenia błędu, ponieważ nie są znane wartości oczekiwane sygnałów wyjściowych z tych neuronów. By móc go określić , korzysta się z wyznaczonego wcześniej sygnału błędu rzutując go wstecz do wszystkich neuronów, których sygnały (wyjścia) stanowiły wejścia dla rozważanego neuronu. Mając określony błąd sygnału przystępuje się do modyfikacji wag na jego wejściach.

**Funkcja Rastrigin:**



Funkcja obrazuje trudności jakie napotkać można przy poszukiwaniu optimum.   
Charakterystyczne dla niej jest to, że posiada wartość najmniejszą w punkcie (0,0), jednak zanim algorytm przeszukiwania znajdzie to minimum globalne, może napotkać wiele minimów lokalnych.

**Danymi uczącymi były**: Wartości uzyskane przy pomocy tego wzoru - obliczono sekwencyjnie wartości funkcji (oczekiwane) w przedziale x<-2,2>.

**Analiza wyników:**

1)Współczynnik uczenia się : 0.001   
x1= -1,55804, x2= 1,11676, x3=-0,80523, x4= 1,8183, x5= -0,64617. ( 3 neurony : 4We, 3We ,1 We)

W tym przypadku czas nauki wyniósł 11168 epok – ok 3.3162 min.  
Sieć uczy się szybko tylko na samym początku.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| err1: | 0,0106652 | czas | 27 | ms | Calkowity | 27ms |
| err2: | 0,00173112 | czas | 25 | ms | Calkowity | 56ms |
| err3: | 0,00068467 | czas | 24 | ms | Calkowity | 81ms |
| err4: | 0,0005618 | czas | 23 | ms | Calkowity | 105ms |
| err5: | 0,00054672 | czas | 23 | ms | Calkowity | 129ms |
| err6: | 0,00054421 | czas | 22 | ms | Calkowity | 152ms |
| err7: | 0,00054317 | czas | 21 | ms | Calkowity | 174ms |
| err8: | 0,0005423 | czas | 24 | ms | Calkowity | 199ms |
| err9: | 0,00054146 | czas | 25 | ms | Calkowity | 225ms |
| err10: | 0,00054061 | czas | 25 | ms | Calkowity | 250ms |
|  |  | .  .  . |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| err11153: | 8,88E-16 | czas | 12 | ms | Calkowity | 198782ms |
| err11154: | 8,88E-16 | czas | 12 | ms | Calkowity | 198795ms |
| err11155: | 8,88E-16 | czas | 13 | ms | Calkowity | 198807ms |
| err11156: | 8,88E-16 | czas | 12 | ms | Calkowity | 198820ms |
| err11157: | 1,78E-15 | czas | 12 | ms | Calkowity | 198833ms |
| err11158: | 8,88E-16 | czas | 12 | ms | Calkowity | 198846ms |
| err11159: | 8,88E-16 | czas | 12 | ms | Calkowity | 198858ms |
| err11160: | 8,88E-16 | czas | 12 | ms | Calkowity | 198871ms |
| err11161: | 8,88E-16 | czas | 12 | ms | Calkowity | 198883ms |
| err11162: | 1,78E-15 | czas | 13 | ms | Calkowity | 198896ms |
| err11163: | 8,88E-16 | czas | 13 | ms | Calkowity | 198908ms |
| err11164: | 8,88E-16 | czas | 12 | ms | Calkowity | 198921ms |
| err11165: | 8,88E-16 | czas | 12 | ms | Calkowity | 198933ms |
| err11166: | 8,88E-16 | czas | 12 | ms | Calkowity | 198946ms |
| err11167: | 1,78E-15 | czas | 13 | ms | Calkowity | 198959ms |
| err11168: | 0 | czas | 13 | ms | Calkowity | 198972ms |

2) Współczynnik uczenia się =0.1 ( 4We,3We,1We)

x1=-1,38851, x2=1,28728, x3=-0,438532, x4= 1,00771, x5=-0,88355   
Czas nauki wyniósł 1417 epok = 0.3531min. Z czasem sieć uczy się coraz wolniej.

3) Współczynnik uczenia się : 0,5  
x1=-0,513246, x2=1,41324, x3=-0695687, x4=0,0304014, x5=-0,32745 ( 4We,3We,1We)  
Czas nauki wyniósł 317 epok = 0.076867min. Sieć z biegiem czasu uczy się coraz wolniej.   
Ilość danych wejściowych ma wpływ na liczbę epok.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| err0: | 0,100352 |  |  |  |  |  | 0 |
| err1: | 0,0327969 | czas | 26 | ms | Calkowity | 26ms | 1 |
| err2: | 0,0232449 | czas | 21 | ms | Calkowity | 49ms | 2 |
| err3: | 0,0208319 | czas | 20 | ms | Calkowity | 69ms | 3 |
| err4: | 0,0192872 | czas | 20 | ms | Calkowity | 89ms | 4 |
| err5: | 0,0178759 | czas | 20 | ms | Calkowity | 109ms | 5 |
| err6: | 0,0165247 | czas | 20 | ms | Calkowity | 129ms | 6 |
| err7: | 0,0152342 | czas | 20 | ms | Calkowity | 149ms | 7 |
| err8: | 0,0140106 | czas | 19 | ms | Calkowity | 169ms | 8 |
| err9: | 0,0128583 | czas | 18 | ms | Calkowity | 188ms | 9 |
| err10: | 0,0117793 | czas | 19 | ms | Calkowity | 207ms | 10 |
| .  .  . |  |  |  |  |  |  |  |
| err316 | 8,8817E-16 | czas | 17 | ms | Calkowity | 4595ms | 316 |
| err317 | 0 | czas | 17 | ms | Calkowity | 4612ms | 317 |

4) Dla porównania zmienione ilości wejść w poszczególnych warstwach.

Współczynnik uczenia się=0,5  
x1=-0,708076, x2=1,61953 (3We,2We,1We)

Czas wyniósł 306 epok= 0.113783min

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| err0: | 0,10 |  |  |  |  |  |
| err1: | 0,0341955 | czas | 26 | ms | Calkowity | 27ms |
| err2: | 0,0247258 | czas | 37 | ms | Calkowity | 65ms |
| err3: | 0,0222122 | czas | 34 | ms | Calkowity | 101ms |
| err4: | 0,0205486 | czas | 33 | ms | Calkowity | 134ms |
| err5: | 0,0190154 | czas | 33 | ms | Calkowity | 169ms |
| err6: | 0,0175429 | czas | 32 | ms | Calkowity | 203ms |
| err7: | 0,0161351 | czas | 32 | ms | Calkowity | 237ms |
| err8: | 0,0148004 | czas | 33 | ms | Calkowity | 271ms |
| err9: | 0,0135445 | czas | 31 | ms | Calkowity | 303ms |
| err10: | 0,0123703 | czas | 31 | ms | Calkowity | 336ms |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| err302: | 8,88E-16 | czas | 21 | ms | Calkowity | 6741ms |
| err303: | 8,88E-16 | czas | 20 | ms | Calkowity | 6762ms |
| err304: | 8,88E-16 | czas | 21 | ms | Calkowity | 6783ms |
| err305: | 8,88E-16 | czas | 21 | ms | Calkowity | 6805ms |
| err306: | 0 | czas | 21 | ms | Calkowity | 6827ms |
|  |  |  |  |  |  |  |

**Wnioski:**

- wartość współczynnika uczenia się ma duży wpływ na szybkość uczenia się sieci, jego mała wartość  
znacznie wydłuża proces nauczania.

-liczba połączeń w warstwie ukrytej również ma wpływ na prędkość działania sieci- może przyśpieszyć/spowolnić czas.

-dla większego współczynnika uczenia się czasy pojedynczych epok są dłuższe niż w przypadku gdzie jest on mniejszy.

-ilość dostarczanych danych wpływa na efektywność działania sieci ( jej poprawność) lecz wydłuża czas obliczeń.

-można powiedzieć że kompromisem co do szybkości i poprawności działania wydaje się być sieć której zadano współczynnik uczenia się o uśrednionej wartości , a ilość danych wejściowych nie była celowo zwiększona.

**Listing najważniejszych fragmentów kodu:**

struct FirstLayer{ // warstwa pierwsza odpowiadająca za warstwę wejściową

vector<double> weights; // wagi dla danych wejściowych

vector<double> output; // sumy = wagi \* odpowiadające im dane wejściowe

vector<double> delta; // błędy wsteczne, uzależnione od błędów na wyższych warstwach

FirstLayer();

void sum(vector<double> input); // suma wag \* dane wejściowe

void correctDelta(vector<double>,vector<double>); // ustawienie błędów wstecznych

void correctWeight(vector<double>); // poprawianie wag

};

struct SecondLayer{ // warstwa ukryta odpowiadająca za warstwę wejściową

vector<double> weights; // wagi dla danych wejściowych

vector<double> output; // sumy = wagi \* odpowiadające im dane wejściowe

vector<double> delta; // błędy wsteczne, uzależnione od błędów na wyższych warstwach

SecondLayer();

void sum(vector<double> input); // sumowanie wag \* dane wejściowe

void correctDelta(double,vector<double>); // ustawienie błędóœ wstecznych

void correctWeight(vector<double>); // poprawianie wag

};

struct ThirdLayer{ // warstwa pierwsza odpowiadająca za warstwę wejściową

vector<double> weights; // wagi dla danych wejściowych

vector<double> output; // sumy = wagi \* odpowiadające im dane wejściowe

double delta; // błędy wsteczne, uzależnione od błędów na wyższych warstwach

ThirdLayer();

double sum(vector<double> input); // suma wag \* dane wejściowe

void correctDelta(double d); // ustawienie błędów wstecznych

void correctWeight(vector<double>); // poprawianie wag

};

FirstLayer \*l1 = new FirstLayer();

SecondLayer \*l2 = new SecondLayer(); // tworzenie warstw

ThirdLayer \*l3 = new ThirdLayer();

randInputs(input);

loadFromFile(x); // załadowanie wektorów

fout.open(FILENAME, ios::app);

l1->sum(input);

l2->sum(l1->output); // wyliczanie sumy dla każdej z warstwy na podstawie warstwy poprzedniej

y = l3->sum(l2->output);

delta = rastriginFunction(x) - y;

tmp = delta; // wyliczanie błędu

printf("%.20f\n",delta);

l3->correctDelta(delta);

l2->correctDelta(l3->delta,l3->weights); // ustawianie błędu wstecznego

l1->correctDelta(l2->delta, l2->weights);

l1->correctWeight(input);

l2->correctWeight(l1->output); // poprawianie wag algorytmem wstecznej propagacji błędu

l3->correctWeight(l2->output);

l1->sum(input);

l2->sum(l1->output); // wyliczanie sumy dla każdej z warstwy na podstawie warstwy poprzedniej

y = l3->sum(l2->output);

delta = rastriginFunction(x) - y;

printf("%.25lf\n",fabs(tmp-delta)); // wyliczanie błędu

fout << "\nerr0: " << fabs(tmp-delta);

void FirstLayer::correctWeight(vector<double> input){

vector<double>::iterator itw = weights.begin();

vector<double>::iterator iti = input.begin();

vector<double>::iterator itd = delta.begin(); // poprawianie wag na podstawie algorytmu wstecznej propagacji błędu

vector<double>::iterator ito = output.begin();

int k = 0;

for(itw=weights.begin(),ito=output.begin(),itd=delta.begin();k<N\_LAYER1;++itw,k++, ++itd,++ito);

{

for(iti = input.begin();iti!=input.end();++iti)

{

(\*itw)+= LEARNING\_RATE

\*(\*itd)

\*pochodnaFunkcjiAktywacji(\*ito)

\*(\*iti);

for(int z=0;z<N\_INPUT;z++){++itw;}

}

for(int z=0;z<N\_INPUT;z++){--itw;}

}

}

void SecondLayer::correctWeight(vector<double> input){

vector<double>::iterator itw = weights.begin();

vector<double>::iterator iti = input.begin(); // poprawianie wag na podstawie algorytmu wstecznej propagacji błędu

vector<double>::iterator itd = delta.begin();

vector<double>::iterator ito = output.begin();

int k = 0;

for(itw=weights.begin(),ito=output.begin(),itd=delta.begin();k<N\_LAYER2;++itw,k++, ++itd,++ito);

{

for(iti = input.begin();iti!=input.end();++iti)

{

(\*itw)+= LEARNING\_RATE

\*(\*itd)

\*pochodnaFunkcjiAktywacji(\*ito)

\*(\*iti);

for(int z=0;z<N\_LAYER1;z++){++itw;}

}

for(int z=0;z<N\_LAYER1;z++){--itw;}

}

}

void ThirdLayer::correctWeight(vector<double> input){

vector<double>::iterator itw = weights.begin();

vector<double>::iterator iti = input.begin();

double itd = delta;

vector<double>::iterator ito = output.begin(); // poprawianie wag na podstawie algorytmu wstecznej propagacji błędu

int k = 0;

for(itw=weights.begin(),ito=output.begin();k<N\_LAYER3;++itw,k++,++ito);

{

for(iti = input.begin(); iti!=input.end();++iti)

{

(\*itw)+= LEARNING\_RATE

\*itd

\*pochodnaFunkcjiAktywacji(\*ito)

\*(\*iti);

for(int z=0;z<N\_LAYER2;z++){++itw;}

}

for(int z=0;z<N\_LAYER2;z++){--itw;}

}