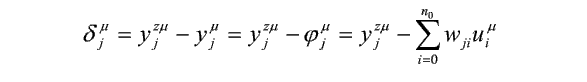
**PSI SCENARIUSZ 2**

**Anna Ruchała, IS III, GR 3**

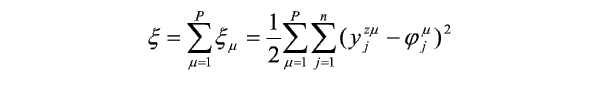
W programie zastosowano Algorytm Wstecznej Propagacji Błędu oraz regułę Delty. Jest to podstawowy algorytm uczenia nadzorowanego wielowarstwowych jednokierunkowych sieci neuronowych. Podaje on przepis na zmianę wag *w*ij dowolnych połączeń elementów przetwarzających rozmieszczonych w sąsiednich warstwach sieci. Oparty jest on na minimalizacji sumy kwadratów błędów uczenia z wykorzystaniem optymalizacyjnej metody największego spadku. Dzięki zastosowaniu specyficznego sposobu propagowania błędów uczenia sieci powstałych na jej wyjściu, tj. przesyłania ich od warstwy wyjściowej do wejściowej, algorytm propagacji wstecznej stał się jednym z najskuteczniejszych algorytmów uczenia sieci.

Rozważmy sieć jednowarstwową o liniowych elementach przetwarzających. Załóżmy, że mamy *P*-elementowy zbiór wzorców. Przy prezentacji m-tego wzorca na wejściu sieci, dla *j*-tego elementu wyjściowego możemy zdefiniować błąd:

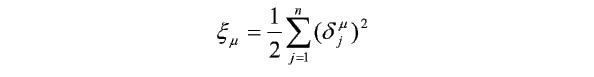


gdzie *y*jzm, *y*jm i jjm oznaczają odpowiednio oczekiwane i aktualne wartości wyjścia *j*-tego elementu oraz ważoną sumę wejść wyznaczoną w jego sumatorze przy prezentacji m-tego wzorca; *u*im - *i*ta składowa m-tego wektora wejściowego (*u*0m = 1, wejście progowe); *w*ijoznacza wagę połączenia pomiędzy *j*-tym elementem warstwy wyjściowej a *i*-tym elementem warstwy wejściowej; *n*0 - liczba wejść.

Jako miarę błędu sieci x wprowadźmy sumę po wszystkich wzorcach błędów powstałych przy prezentacji każdego z nich:

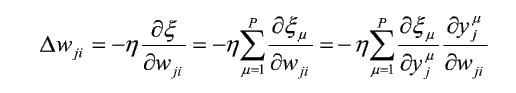


Gdzie:



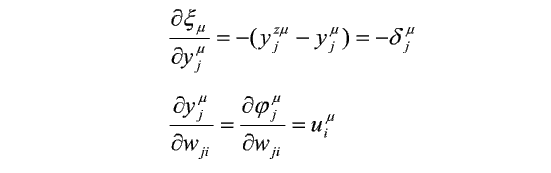
przy czym *n* oznacza liczbę elementów w warstwie wyjściowej.

Problem uczenia sieci to zagadnienie minimalizacji funkcji błędu x. Jedną z najprostszych metod minimalizacji jest gradientowa metoda największego spadku. Jest to metoda iteracyjna, która poszukuje lepszego punktu w kierunku przeciwnym do gradientu funkcji celu w danym punkcie. Stosując powyższą metodę do uczenia sieci, zmiana D*w*ji wagi połączenia *w*ji winna spełniać relację:

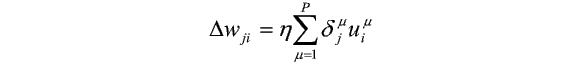


gdzie h oznacza współczynnik proporcjonalności.

W przypadku liniowych elementów przetwarzających mamy:



Stąd otrzymamy:

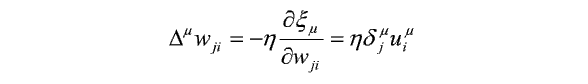


Ostatecznie pełną regułę wyznaczania wag połączeń zapiszemy:



gdzie górne indeksy *n* i *s* oznaczają odpowiednio "nową" i "starą" wartość współczynnika wag *w*ji.

Konsekwentna realizacja metody największego spadku wymaga dokonywania zmian wag *w*ji dopiero po zaprezentowaniu sieci pełnego zbioru wzorców, m = 1, 2, ..., *P*. W praktyce stosuje się jednak zmiany wag po każdej prezentacji pojedynczego wzorca zgodnie ze wzorem:



co pozwala istotnie uprościć praktyczną realizację algorytmu.

Zasada zmiany wartości współczynników wag *w*ji, określona powyższym wzorem, nazywana jest regułą delty. Owa modyfikacja wprowadza wprawdzie pewne zaburzenie do metody największego spadku, ale jest ono zaniedbywalnie małe dla odpowiednio niewielkich wartości h, czyli dla małych zmian wag. Reguła delty, jako pewne przybliżenie metody największego spadku przy wystarczająco małym współczynniku h, poszukuje zbioru wag minimalizującego funkcję błędu sieci liniowej.

Ogólny schemat procesu trenowania sieci wygląda następująco:

1. Ustalamy topologię sieci, tzn. liczbę warstw, liczbę neuronów w warstwach.
2. Inicjujemy wagi losowo (na małe wartości).
3. Dla danego wektora uczącego obliczamy odpowiedź sieci (warstwa po warstwie).
4. Każdy neuron wyjściowy oblicza swój błąd, oparty na różnicy pomiędzy obliczoną odpowiedzią *y* oraz poprawną odpowiedzią *t*.
5. Błędy propagowane są do wcześniejszych warstw.
6. Każdy neuron (również w warstwach ukrytych) modyfikuje wagi na podstawie wartości błędu i wielkości przetwarzanych w tym kroku sygnałów.
7. Powtarzamy od punktu 3. dla kolejnych wektorów uczących. Gdy wszystkie wektory zostaną użyte, losowo zmieniamy ich kolejność i zaczynamy wykorzystywać powtórnie.
8. Zatrzymujemy się, gdy średni błąd na danych treningowych przestanie maleć. Możemy też co jakiś czas testować sieć na specjalnej puli nieużywanych do treningu próbek testowych i kończyć trenowanie, gdy błąd przestanie maleć.

Przygotowany przeze mnie zestaw uczący zawiera 10 dużych i 10 małych liter na matrycy 5x5 (25 pól). Liniowym rozwinięciem tych matryc będą wektory wejściowe ciągu uczącego. Ciąg uczący składa się z 20 następujących wektorów uczących:

input[0] = new double[25] { 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1 };

input[1] = new double[25] { 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0 };

input[2] = new double[25] { 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1 };

input[3] = new double[25] { 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0 };

input[4] = new double[25] { 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1 };

input[5] = new double[25] { 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0 };

input[6] = new double[25] { 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0 };

input[7] = new double[25] { 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 };

input[8] = new double[25] { 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 };

input[9] = new double[25] { 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0 };

input[10] = new double[25] { 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0 };

input[11] = new double[25] { 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0 };

input[12] = new double[25] { 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0 };

input[13] = new double[25] { 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0 };

input[14] = new double[25] { 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0 };

input[15] = new double[25] { 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0 };

input[16] = new double[25] { 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0 };

input[17] = new double[25] { 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0 };

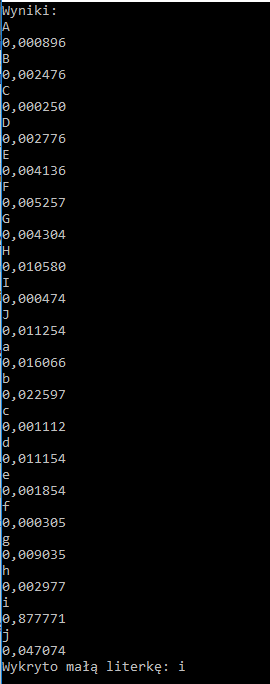
input[18] = new double[25] { 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0 };

input[19] = new double[25] { 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0 };









Wnioski:

Widać, iż nauczona sieć bardzo poprawnie rozpoznaje prezentowane wzory, nawet zniekształcone, ale także dostrzega w nich i inne litery. Na tej podstawie można dokładnie badać jakie cechy liter zostały uznane przez siec za istotne dla opisu litery.

Zwiększenie precyzji rozpoznawania liter czy też szerzej - znaków i symboli opartych na matrycy prezentowanej sieci można uzyskać poprzez np. powiększenie jej rozmiarów i jednocześnie odpowiednio definiując okno dyskryminatora filtrującego wyniki.

Korekta wektora jest tym silniejsza, im większy był odnotowany błąd. Trzeba silnie interweniować przy dużych błędach i dokonywać płynnego i subtelnego dostrajania w przypadku błędów małych. Przy okazji zapewniony jest warunek, że w przypadku braku błędu, żadne korekty nie będą dokonane. I-ta składowa wektora będzie tym silniej zmieniana w wyniku procesu uczenia, im większa była odpowiadająca jej składowa wektora wejściowego. W przypadku ujemnych wartości odpowiednie składowe wektora będą zmniejszane a nie powiększane. Oznacza to, że w odróżnieniu od wejść „pobudzających”, które dla uzyskania większego sygnały trzeba wzmocnić, wejścia „hamujące” trzeba osłabić.

Sieć składał się z pojedynczych elementów Adaline (ang. Adaptive linear element), które powielone oraz połączone dały układ Madaline (ang. Many Adaline).