Oznaczenia:
SSN – Sztuczna Sieć Neuronowa
Ten kolor oznacza informacje dotyczące konkretnej implementacji SSN (dany projekt).
…] – do zbadania

PROGRAM ROZWIĄZUJE NASTEPUJĄCY PROBLEM

System rozpoznawania zeskanowanych obrazów podpisów osób.

Obrazy podpisów osób są podawane **w plikach BMP (bitmapa) w 256-kolorach** (szarości lub dowolnych) o dowolnych rozmiarach (ograniczonych do maksymalnej wielkości **460x200** pikseli).
Linia (kreska) podpisu powinna być ciemniejsza od jego tła.
Rozpoznawanie obrazów podpisów odbywa się za pomocą SSN.

Możliwe sposoby użycia SSN do rozwiązania tego problemu.
1. SSN jest uczona wielu podpisów jednej osoby X i ma rozstrzygać czy podany podpis należy do osoby X, czy nie. Dla możliwości rozstrzygania o przynależności podpisu wśród N osób należy wtedy użyć N takich SSN. Każda SSN byłaby uczona podpisów jednej osoby. Takie SSN dawałyby na wyjściu odpowiedź 'Tak' lub 'Nie'. W warstwie wyjściowej należałoby użyć jeden neuron. Odpowiedź ustalano by przez ustalenie SSN, która daje odpowiedź najbardziej zbliżoną do odpowiedzi 'Tak'.
Zaletą tego rozwiązania jest to, że przy dodaniu nowej osoby do bazy, należy wtedy tylko stworzyć nową SSN i **nauczyć ją obrazów podpisów tylko tej osoby**. Nie trzeba także trzymać w bazie obrazów podpisów pozostałych osób.
Wadą zaś jest to, że **dla każdej nowej nauczonej SSN należy trzymać na dysku jej zbiór parametrów**.
Rozwiązanie to nadaje się do użycia tylko na stacjonarnym komputerze, gdzie ograniczeniem na liczbę osób, których obrazy podpisów są rozpoznawanie jest ilość dostępnej pamięci masowej.

2. SSN jest uczona podpisów kilku osób naraz i ma odpowiadać, do której osoby należy podany podpis.
Dla N osób taka SSN dawałaby na wyjściu jedną z N odpowiedzi. W warstwie wyjściowej należałoby wtedy użyć N neuronów.
Odpowiedź ustalano by przez ustalenie numeru neuronu wyjściowego o największej wartości.
Wadą tego rozwiązania jest to, że przy dodaniu nowej osoby do bazy, należy wtedy nauczyć SSN **od nowa wszystkich obrazów podpisów wszystkich osób**. Należy wtedy trzymać w bazie obrazy podpisów wszystkich osób.
Zaletą zaś jest to, że mamy **tylko jeden zbiór parametrów nauczonej SSN**, co oznacza oszczędność miejsca na dysku.
Rozwiązanie to można by wykorzystać w przenośnych urządzeniach, które dysponują ograniczoną ilością pamięci. Proces nauki SSN byłby przeprowadzany na stacjonarnym komputerze a końcowy zbiór parametrów tej jednej SSN kopiowany byłby na urządzenie przenośne. Taką SSN można by nauczyć rozpoznawania obrazów podpisów do kilkuset osób.

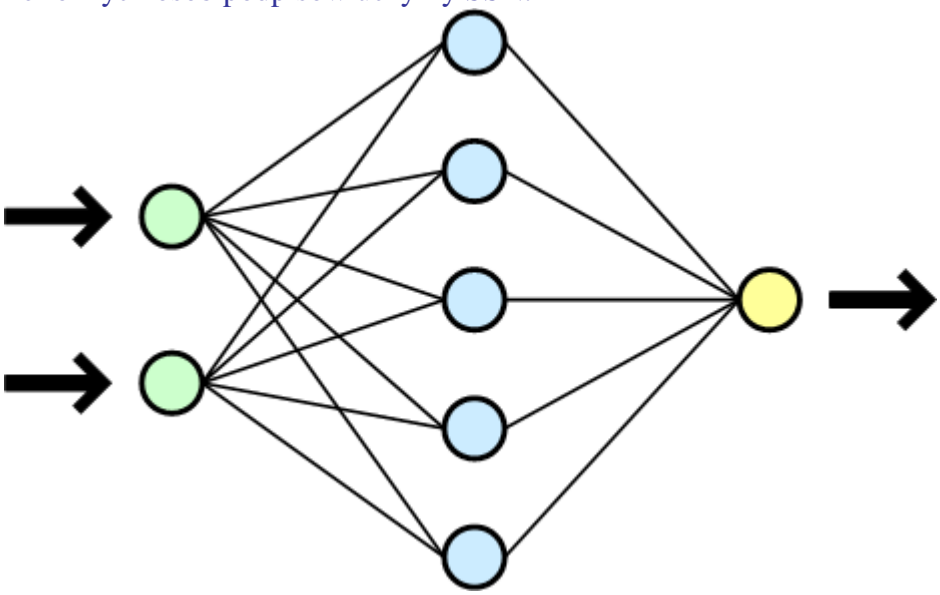
W tym przypadku wykonano wariant nr 2.

Wejście systemu stanowi zbiór plików obrazów zeskanowanych podpisów kilku osób. Dla każdej osoby po kilka obrazów.
Im więcej będzie obrazów podpisów dla każdej osoby tym system lepiej będzie mógł rozpoznawać obrazy danej osoby.
Obrazy z plików w postaci bitmap (BMP) są odpowiednio przetwarzane i podawane na wejście SSN.

Wynikowy obraz – to końcowy obraz podpisu po przetworzeniu (m.in. kontrastowanie, skalowanie), podawany do SSN. Jego rozmiary ustalono na: **96x24** pikseli = **2304 pikseli.** Jest on trzymany w postaci tablicy liczb typu unsigned char (1 Bajt). Rozmiary wynikowego obrazu można w kodzie projektu łatwo zmienić (w pliku: KlasaBazaPodpisow.h). Obraz ten jest zinwersowany, tzn. wynikowo piksele tła mają wartość bliską 0 a piksele należące do linii podpisu mają wartości bliskie 255.
Projekt został zrobiony elastycznie. Wszelkie stałe wystarczy zmieniać w jednym miejscu projektu (pliki nagłówkowe: KlasaBazaPodpisow.h).

SZTUCZNA SIEĆ NEURONOWA

Na wejście SSN podawane są wartości kolorów pikseli wynikowego obrazu, znormalizowane do przedziału [0.0 -1.0] liczb rzeczywistych typu double (64 bity; 15-cyfrowa precyzja).
Warstwa wejściowa zawiera 96*24 = 2304 neuronów. Warstwa wyjściowa zawiera tyle neuronów ile różnych osób podpisów uczymy SSN.



Sieci jednokierunkowe to sieci neuronowe, w których nie występuje sprzężenie zwrotne, czyli pojedynczy wzorzec lub sygnał przechodzi przez każdy neuron dokładnie raz w swoim cyklu.
Najprostszą siecią neuronową jest pojedynczy **perceptron progowy**, opracowany przez McCullocha i Pittsa w roku 1943.
W programie uczącym SSN zaimplementowano **czterowarstwową nieliniową sztuczną sieć neuronową jednokierunkową** (wielowarstwowa sieć perceptronowa (ang. Multi-Layered Perceptron MLP)).
Zgodnie z teorią sieć o dwóch lub trzech warstwach perceptronowych może być użyta do aproksymacji większości funkcji.

Kolejne warstwy sieci, to: warstwa danych wejściowych, **dwie warstwy ukryte** i warstwa danych wyjściowych.

Użycie dwóch warstw ukrytych zamiast tylko jednej znacząco poprawia uczenie się SSN i jej możliwości generalizacji wiedzy. Trzy warstwy to już przesada, a dwie w zupełności wystarczają do poprawnej nauki.

Połączenia neuronów między warstwami - każdy z każdym.

Każdy neuron posiada tyle wejść, ile jest neuronów w warstwie poprzedniej oraz dodatkowe jedno wejście tzw. **bias** o stałej wartości równej 1.0. Zmieniają się jednak wagi związane z biasem.

Z każdym wejściem sygnału do neuronu związana jest **waga**, czyli liczba określająca wpływ tego sygnału wejściowego na wynik wyjściowy neuronu.

Łączna liczba wszystkich wag w SSN wynosi zatem:
LiczbaNeuronówUk1*(LiczbaDanychWe+1) + LiczbaNeuronówUk2*(LiczbaNeuronówUk1+1) + LiczbaNeuronówWy*(LiczbaNeuronówUk2+1),
gdzie:
LiczbaDanychWe – liczba wejść
LiczbaNeuronówUk1 – liczba neuronów w pierwszej warstwie ukrytej
LiczbaNeuronówUk1 – liczba neuronów w drugiej warstwie ukrytej
LiczbaNeuronówWy – liczba neuronów w warstwie wyjściowej

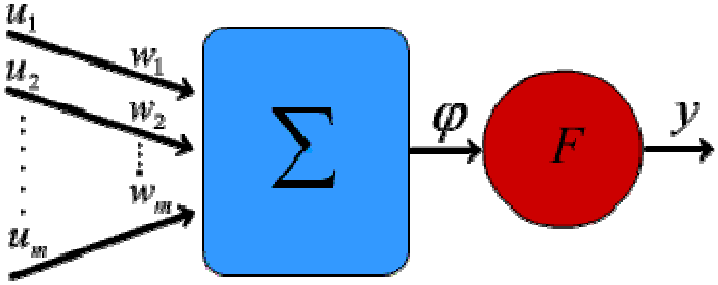
Wektorem uczącym są następujące dwa ciągi danych: uczący (wejściowy) i weryfikujący (wyjściowy).

Wektorem wejściowym jest 2304 liczb rzeczywistych, o wartościach z przedziału [0.0 ; 1.0].
Wektorem wyjściowym jest N liczb rzeczywistych; wszystkie oprócz jednej równe 0.0. Liczba, której numer reprezentuje daną osobę ma wartość równą 1.0
Przykładowe wektory wyjściowe:
- [**1**,0,0,0,0] oznacza odpowiedź „osoba nr 0”
- [0,0,0,0,**1**] oznacza odpowiedź „osoba nr 4”

Warstwy ukryte zawierają dowolnie ustalaną liczbę neuronów. Dla neuronów ukrytych w liczbie około 40-70 sieć wydaje się uczyć najszybciej.
Liczbę neuronów w każdej warstwie ukrytej można dowolnie ustalać w programie uczącym SSN. Dla wielu warstw ukrytych stosuje się zwykle metodę **piramidy geometrycznej**, która zakłada, że liczba neuronów w kolejnych warstwach tworzy kształt piramidy i **maleje od wejścia do wyjścia**. Zatem liczba neuronów w kolejnych warstwach powinna być coraz mniejsza (większa liczba neuronów w kolejnej warstwie nie zwiększa możliwości nauki SSN).
Zatem w naszym przypadku SSN może ostatecznie mieć rozmiar np.: 2304 - 70 - 40 - N. Liczba osób (N) nie powinna, zatem przekroczyć liczby neuronów w ostatniej (drugiej) warstwie ukrytej.
Generalnie uczenie rozpoczyna się z małą liczbą neuronów w warstwach ukrytych a następnie, obserwując postępy tego procesu, doświadczalnie zwiększa się ich liczbę.

Funkcja aktywacji
Wartość funkcji aktywacji jest sygnałem wyjściowym neuronu i propagowana jest do neuronów warstwy następnej. Funkcja aktywacji może przybierać jedną z trzech postaci:
- nieliniowa
- liniowa
- skoku jednostkowego tzw. funkcja progowa

Argumentem funkcji aktywacji neuronu są zsumowane iloczyny sygnałów wejściowych i wag tego neuronu (tzw. suma ważona): $F(u_1*w_1+\ldots+u_m*w_m)$.

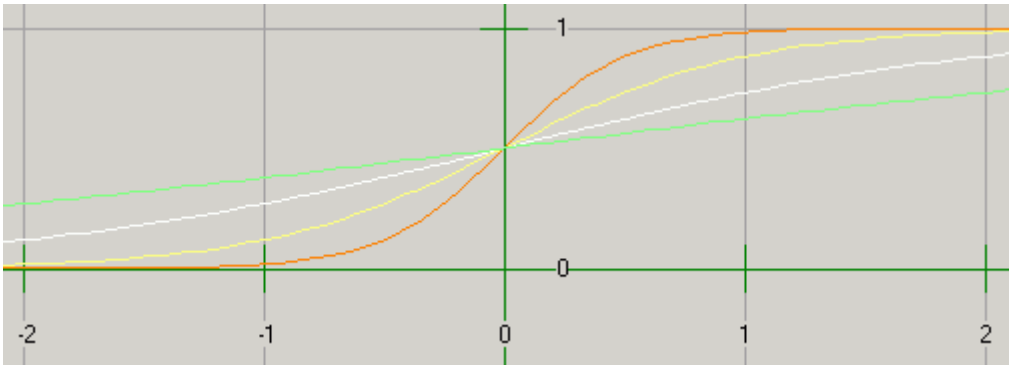


Wybór funkcji aktywacji zależy od rodzaju problemu, jaki stawiamy przed siecią do rozwiązania. Dla sieci wielowarstwowych najczęściej stosowane są **funkcje nieliniowe**, gdyż neurony o takich charakterystykach wykazują największe zdolności do nauki, polegające na możliwości odwzorowania w sposób płynny dowolnej zależności pomiędzy wejściem a wyjściem sieci. Umożliwia to otrzymanie na wyjściu sieci informacji ciągłej a nie tylko postaci: TAK - NIE.

Wymagane cechy funkcji aktywacji to:
- ciągłe przejście pomiędzy swoją wartością maksymalną a minimalną (np. 0.0-1.0),
- łatwa do obliczenia i ciągła pochodna,
- możliwość wprowadzenia do argumentu parametru beta do ustalania kształtu krzywej.

Użytą funkcją aktywacji neuronu jest funkcja **sigmoidalna unipolarna:** $f(x,beta) = \frac{1.0}{1.0+e^{-beta*x}}$

Funkcja ta charakteryzuje się tym, że wartość jej pochodnej można obliczyć z obliczonej jej wartości:
f'(x,beta) = beta*F*(1.0-F); gdzie F = f(x,beta).



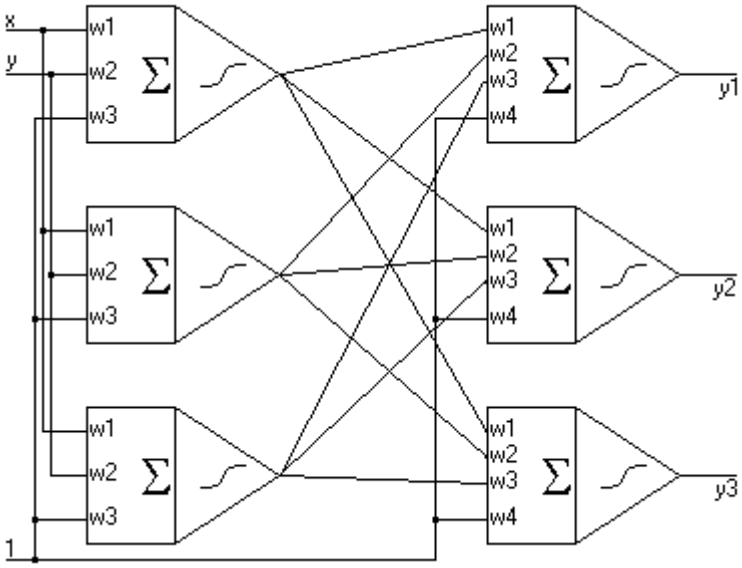
(białe) beta = 1.0;
beta = 2.0;
beta = 4.0;
beta = 0.5;

Funkcja sigmoidalna unipolarna

Każda warstwa neuronów ma własny współczynnik **beta** dla funkcji aktywacji (można go ustalać w programie uczącym SSN; np. na kolejno: 1.0; 1.0; 1.0).
Jest to współczynnik determinujący nachylenie sigmoidalnej funkcji aktywacji.

Wpływ parametru beta może być zastąpiony odpowiednim doborem innych parametrów i wielkości. W przypadku przetwarzania zmiennych wejściowych na zmienne wyjściowe, efekt jaki powoduje zmiana nachylenia krzywej można uzyskać mnożąc wartości wag sieci przez wielkość parametru beta.

Przykładowa wielowarstwowa sieć neuronowa wygląda następująco:



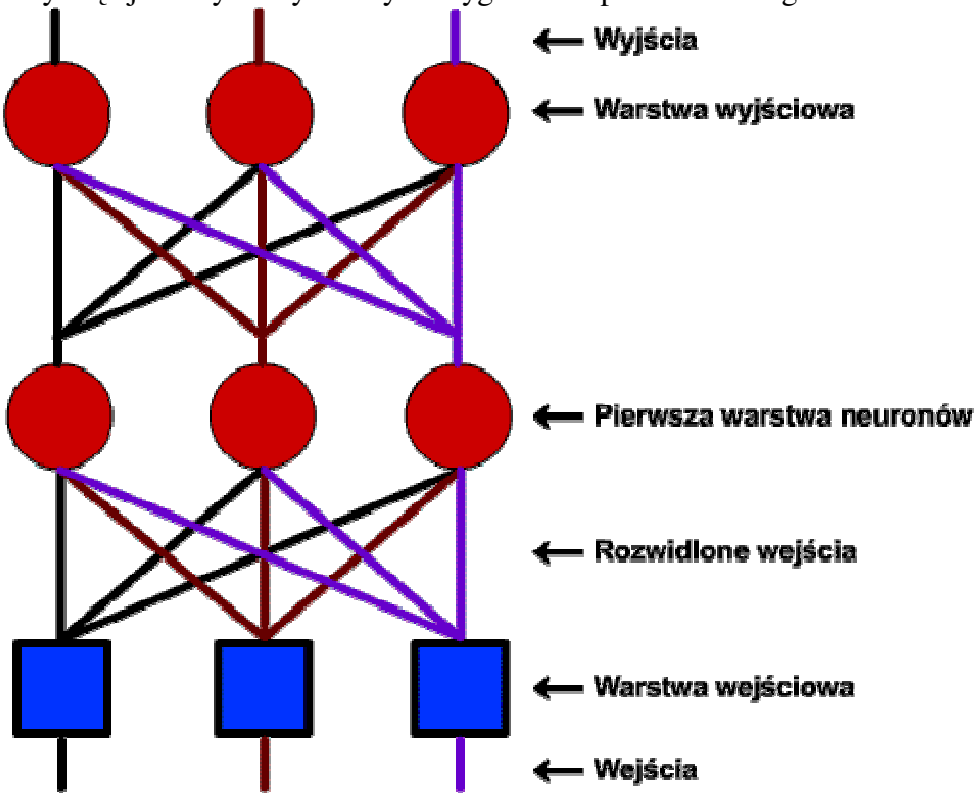
Po obliczeniu końcowego wyniku SSN, neurony na wyjściu mają wartość z przedziału [0.0; 1.0]. Numer neuronu o największej wartości oznacza numer osoby, do której należy podany obraz podpisu.

UCZENIE SSN

Działanie SSN polega na tym, że sygnały pobudzające zawarte w wektorze wejściowym podawane na wejścia sieci, przetwarzane są w poszczególnych neuronach. Po tej projekcji na wyjściach sieci otrzymuje się wartości liczbowe, które stanowią odpowiedź sieci na pobudzenie i stanowią rozwiązanie postawionego problemu. Sieć uczymy „na przykładach”

Celem jest uzyskanie oczekiwanych zadawalających (tzn. nie przekraczających dopuszczalnego błędu) wartości na wyjściu przy danych wartościach wejściowych.

Gdy błąd jest zbyt duży należy skorygować odpowiednio wagi.



Jednak, aby takie rozwiązanie uzyskać, należy przejść żmudną drogę uczenia sieci. Jedną z metod jest **uczenie metodą wstecznej propagacji błędów – EBP** (ang. Error Back Propagation). Jest to podstawowa (i jedna z najskuteczniejszych) metoda uczenia sieci wielowarstwowych. Została opracowana w 1974 roku przez P. J. Werbosa jako uczenie z nadzorem lub inaczej - z nauczycielem. Taką też metodę zastosowałem w programie.
Algorytm opiera się na sprowadzeniu wartości funkcji błędu sieci poniżej pewnego założonego minimum. Jako miarę błędu stosuje się błąd średniokwadratowy na neuronach wyjściowych. Poprawianie wag odbywa się **po prezentacji każdego wzorca**, o ile błąd dla tego wzorca jest za duży (parametr epsilon).
Nie jest natomiast użyta tutaj inna metoda, tzw. reguła skumulowanej delty, czyli poprawianie wag po prezentacji wszystkich wzorców, (czyli po całej epoce) – tzw. BATCH propagation.

OD STRONY MATEMATYCZNEJ
Proces uczenia SSN to **minimalizacja funkcji błędu**, której dokonujemy za pomocą gradientu. Kierunek największego spadku dowolnej funkcji wskazuje ujemny gradient tej funkcji.
Gradient F - wektor utworzony z pochodnych funkcji F po wszystkich jej zmiennych.

Załóżmy, że mamy SSN złożoną z 3 warstw:
- wejściowa: **U**: u[0] ÷ u[m-1], to dane wejściowe, a u[m] – to stała wartość = 1.0 – tzw. bias
- jedna ukryta: **X**: x[0] ÷ x[n-1] oraz stała x[n] = 1.0 – tzw. bias
- wyjściowa: **Y**: y[0] ÷ y[r-1]
Rozmiary sieci:
m – liczba danych wejściowych
n – liczba neuronów w warstwie ukrytej
r – liczba neuronów w warstwie wyjściowej

U, X, Y – wektory danych
Mamy zatem tutaj dwie warstwy wag:
- między warstwą wejściową i ukrytą: **w[i,j]**; i – nr neuronu warstwy ukrytej; j – nr danej wejściowej; czyli **W[i]**: W[0] ÷ W[m]
- między warstwą ukrytą i wyjściową: **s[i,j]**; i – nr neuronu warstwy wyjściowej; j – nr neuronu warstwy ukrytej; czyli **S[i]**: S[0] ÷ S[n]
W[i], S[i] – wektory wag w i s

Mamy osobne funkcje aktywacji dla każdej warstwy sieci:
$\mathbf{f^{Uk}(x)}$ = f(x, BetaUk) – funkcja aktywacji w warstwie ukrytej
$\mathbf{f^{Wy}(x)}$ = f(x, BetaWy) – funkcja aktywacji w warstwie wyjściowej

Sygnały w SSN płyną od wejścia do wyjścia, czyli wynik SSN jest obliczany kolejno:
Najpierw:
$x[i] = f^{Uk}(U*W[i]) = f^{Uk}(u[0]*w[i,0] + u[1]*w[i,1] + ...+ u[m-1]*w[i,m-1] + 1*w[i,m])$
(x[n] = 1)
czyli wektor $X = f^{Uk}(U*W)$
Potem:
$y[i] = f^{Wy}(X*S[i]) = f^{Wy}(x[0]*s[i,0] + x[1]*s[i,1] + ...+ x[n-1]*s[i,n-1] + 1*s[i,n])$
czyli wektor $Y = f^{Wy}(X*S)$

Niech **P** – zbiór wszystkich wzorców; $\mathbf{Z^{(p)}}$ – oczekiwany wynik dla danego wzorca p € P
Dla ustalonego p niech $\mathbf{z} = z^{(p)}$, $\mathbf{x} = x^{(p)}$, $\mathbf{y} = y^{(p)}$.

Nasz cel: Dla każdego wzorca p € P minimalizacja wartości: $\sum_{k=0}^{r-1}(y[k]-z[k])^2 = E$

$E = \sum_{k=0}^{r-1}(f^{Wy}(X*S[k]-z[k])^2$, czyli

E = F(S[0], S[1], …, S[r-1], X, Z) – funkcja r zmiennych S[i] oraz X i Z

Chcemy znaleźć minimum funkcji F, czyli odpowiednich wartości zmiennych S[i].
Używamy **metody gradientu**:

Gradient $F = [\frac{\partial F}{\partial S[0]}, \frac{\partial F}{\partial S[1]}, ..., \frac{\partial F}{\partial S[r-1]}]$ t.j. wektor pochodnych F po wszystkich zmiennych S[i]

Oznaczmy potencjał wejściowy neuronu warstwy wyjściowej i jako **PotWej^{Wy}[i]** = X*S[i]
Zatem:

$$\frac{\partial F}{\partial S[0]} = 2*\sum_{k=0}^{r-1}(f^{Wy}(X*S[k]-z[k])^1 * \frac{\partial f^{Wy}(X*S[k]-z[k])}{\partial S[0]} =$$

= 2 * (f^{Wy}(X*S[0] – z[k]) * f^{Wy}'(X*S[0]) * X =
(bo dla k ≠ 0 ∂f^{Wy}(X*S[k] – z[k])/ ∂S[0] = 0)
= 2 * (y[0] – z[0]) * f^{Wy}'(X*S[0]) * X
= 2 * (y[0] – z[0]) * f^{Wy}'(PotWej^{Wy}[0]) * X.
f^{Wy}'(x) – pochodna funkcji f^{Wy}(x)

Zatem w celu minimalizacji F zmieniamy argumenty z szybkością eta:
𝕊 := 𝕊 – eta * Gradient F, czyli

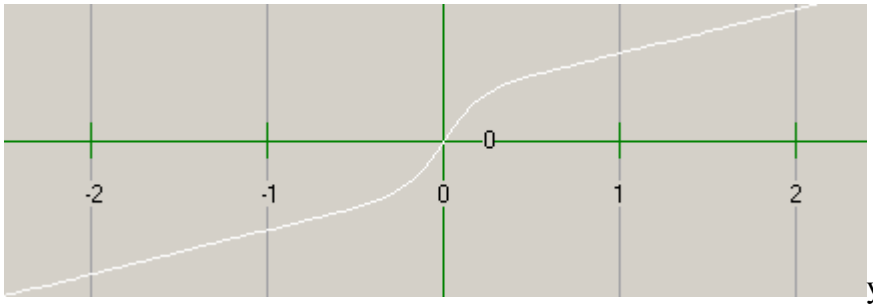
$S[i] := S[i] - eta * \frac{\partial F}{\partial S[i]}$, czyli np. dla neuronu nr 0 i wagi nr i mamy:

s[0,i] := s[0,i] - eta * 2 * (y[0] - z[0]) * f^{Wy}'(PotWej^{Wy}[0]) * x[i] =
s[0,i] := s[0,i] + eta * 2 * (z[0] - y[0]) * f^{Wy}'(PotWej^{Wy}[0]) * x[i].

W celu przyspieszenia obliczeń, można dla danego neuronu k obliczyć na początku stałą wartość (mała delta):
δ(k) = 2 * (z[k] - y[k]) * f^{Wy}'(PotWej^{Wy}[k]), jest to **wartość błędu neuronu k**, wtedy
s[k,i] := s[k,i] + eta * δ(k) * x[i]
Zatem aktualizacja wag neuronów odbywa się zgodnie ze wzorem (to wersja bez członu momentum):
s[k,i] = s[k,i] + Δs[k,i],
gdzie Δs[k,i] = eta * δ(k) * x[i] – dla warstwy wyjściowej

Aby zmiany były nieco łagodniejsze można Δs[k,i] nieco zmienić tzn. sugerować się bardziej znakiem gradientu (podobnie jak w metodzie **Resilient BackPropagation** RPROP) np.

Δs[k,i] = eta * sign(δ(k) * x[i]), gdzie np. sign(x)= $\frac{0.5}{1+e^{-10x}} + \frac{1}{4} + \frac{x}{4}$



y=sign(x)

Minimalizacja funkcji F powinna odbywać się także poprzez zmianę X, czyli poprzez zmianę wag W[i]: W[0] ÷ W[n-1].

Okazuje się, że dla dowolnej warstwy (L) δ(k) = sumie ważonej δ (delt) z warstwy następnej (L+1) razy pochodna funkcji aktywacji, tzn.:
W naszym przypadku dla warstwy ukrytej i wyjściowej:
$\delta^{Uk}(i) = \sum_{j=0}^{r-1}(\delta^{Wy}(j) * s[j,i]) * f^{Uk\,'}(PotWej^{Uk}[i]); \delta^{Uk}(0) \div \delta^{Uk}(n-1)$
Ogólnie dla warstw L i L+1:
$\boldsymbol{\delta^{(L)}(i)} = \sum_{j=0}^{r(L+1)-1}(\delta^{(L+1)}(j) * w^{(L+1)}[j,i]) * f^{(L)\,'}(PotWej^{(L)}[i])$
r(L+1) – liczna neuronów w warstwie L+1
$\mathbf{w^{(L+1)}[j,i]}$ – waga między neuronem j w warstwie L+1 a neuronem i w warstwie L

Kod C++:
```
for (i=0; i < LNeuronowUk; i++) //po wszystkich neuronach z warstwy Uk; licz d(i) dla NeuronUk
 {E = 0.0;  //liczymy sumę ważoną delt z warstwy następnej
  for (j=0; j < LNeuronowWy; j++) E += dWy[j] * WagaWy[j][i];
  dUk2[i] = FunP(PotWejUk[i], BetaUk) * E; //d(i) i-ty neuron
 }
```

PROCES UCZENIA SSN
Zastosowany sposób uczenia SSN wygląda w ogólności następująco:
Pętla po wszystkich wzorcach (tzw. **epoka**):
Oblicz błąd dla danego wzorca.
Jeżeli błąd ten jest większy niż parametr epsilon, to popraw wagi **jeden RAZ!**,
(czyli nie poprawiaj wag, aż nie będzie błędu dla tego wzorca (tzw. **cykl**) i dopiero badaj następny).
Sprawdź następny wzorzec.
Powtarzaj tę pętlę, aż nie będzie błędu większego niż '**epsilon**' dla wszystkich wzorców (w całej epoce) oraz suma tych błędów nie będzie większa od parametru '**Suma epsilon**'.

Zatem tutaj cykl składa się z jednej poprawy wag. Taki sposób uczenia okazuje się (sprawdzono w testach) być szybszy, niż stosowanie pełnych cykli.

Błąd sieci to suma po wszystkich neuronach wyjściowych kwadratów różnicy między obliczonym wynikiem sieci (dla podanych danych wejściowych) a oczekiwanym wynikiem (dla tych danych).
Np. Dla pewnych danych wejściowych i dwóch neuronów w warstwie wyjściowej sieć daje na wyjściu wartości 0.4 i 0.9, a powinna dawać wartości odpowiednio: 0.0 i 1.0. Zatem błąd dla tego wzorca wynosi $0.4^2 + 0.1^2 = 0.16 + 0.01 = 0.17$.
Najczęściej, aby SSN poprawnie nauczyła się, błąd sieci powinien być <0.25, gdyż $0.25 = 0.5^2$, a 0.5 to jest granica dopuszczalnej wartości różnicy między wynikiem sieci a oczekiwanym wynikiem.
Jednak dla większej liczby neuronów wyjściowych błąd sieci można nieco zwiększyć i przez to przyspieszyć nauczenie się SSN. […]
W naszym przypadku błąd sieci można ustawić na wartość 0.27.

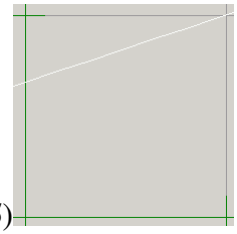
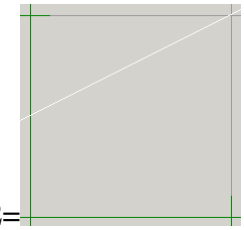
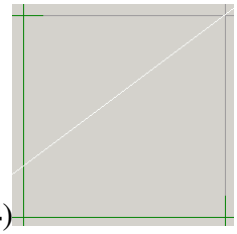
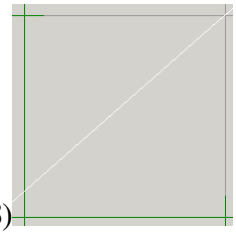
PARAMETRY UCZENIA SSN
Implementacja SSN została napisana od podstaw na podstawie własnej wiedzy i doświadczeń.
Zawarta jest w plikach: KlasaSiecNmom2H.h/cpp

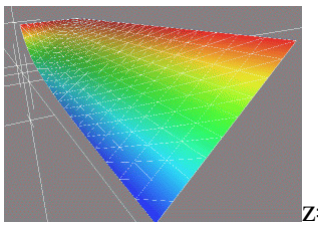
Liczba wzorców – to liczba obrazów, których SSN ma się uczyć i na ich podstawie rozpoznawać inne obrazy.

eta – współczynnik szybkości uczenia – określa jak bardzo w danym kroku zmieniane są wagi (kierunek uczenia sieci).
Zbyt mała jego wartość powoduje zwiększenie się liczby potrzebnych iteracji i tym samym czasu wymaganego na zakończenie treningu. Algorytm ma ponadto w takim przypadku tendencję to wygasania w minimach lokalnych funkcji celu. Ustalenie z kolei zbyt dużej wartości kroku grozi wystąpieniem oscylacji wokół poszukiwanego minimum, prowadzącej ostatecznie do zatrzymania procesu uczenia bez osiągniecie wymaganej wartości funkcji celu.
Zwiększenie tej wartości może zwiększać szybkość uczenia się SSN pojedynczych wzorców, ale może to powodować, że w znajdowaniu globalnego minimum funkcji, algorytm wpadnie w minimum lokalne i nie będzie możliwe nauczenie się wszystkich wzorców.
Mała wartość powoduje, że SSN uczy się małymi kroczkami, ale za to konsekwentnie dąży do celu i nierzadko przez to zwiększa się całościowa szybkość nauczenia się wszystkich wzorców.

W programie zastosowano dynamiczne ustalanie parametru eta proporcjonalnie od aktualnej liczby błędów sieci. Ponadto można ustalić wartości eta dla każdej warstwy osobno, przykładowo:
EtaUk1 = 0.3, A_EtaUk1 = 6, AlfaUk1 = 0.88,
EtaUk2 = 0.2, A_EtaUk2 = 5, AlfaUk2 = 0.88,
EtaWy = 0.1, A_EtaWy = 4, AlfaWy = 0.88,

A_Eta – określa jak bardzo ma maleć wartość Eta w zależności od liczby błędów w ostatniej epoce. Im większa wartość A_Eta tym eta bardziej maleje, proporcjonalnie do liczby błędów. A_Eta musi być >= 1.0. Gdy A_Eta = 1.0, to eta nie zmienia się.
Dokładnie obliczanie końcowego parametru eta wygląda następująco:
eta = Eta * F(LBłędów/Wzorców, A_Eta), gdzie F(x, n) = (1+(n-1)*x)/n

F(x,**1.5**)  F(x,**2**=  F(x,**4**)  F(x,**8**) 

 z=F(x,n); 0<=x<=1; 1<=n<=6

Alfa – parametr określający wpływ **członu momentum (bezwładność)** dla każdej z warstw osobno.
Im większa jego wartość, tym większy wpływ na kierunek uczenia mają poprzednie wartości popraw wag sieci. Odpowiednio użyty może bardzo poprawić szybkość uczenia. Najczęściej przyjmuje się wartość 0.7 - 0.95. Wartość równa 0.0 oznacza brak wpływu członu momentum na zmianę wag.

epsilon – maksymalny błąd SSN dla pojedynczego wzorca. Zwykle ustala się jego wartość na nie większą niż **0,5** (lub 0,25 gdy jest jeden neuron na wyjściu).
Zmniejszanie tej wartości powoduje zwiększanie dokładności sieci i co za tym idzie precyzji predykcji lub generalizacji, ale też nieuchronnie wydłuża czas trwania procesu uczenia lub całkowicie uniemożliwia poprawną naukę SSN przy dużej liczbie wzorców do nauczenia.

Zwiększanie wartości tego parametru powoduje natomiast szybsze zakończenie uczenia sieci, ale tak nauczona sieć może dawać złe odpowiedzi nawet dla uczonych wzorców.

Suma epsilon – maksymalna suma błędów SSN dla wszystkich uczonych wzorców. Dodatkowy parametr określający jakość nauczonej SSN. Jeżeli nie interesuje nas ten parametr, to należy go ustawić na odpowiednio dużą wartość (> liczby wzorców).

Maksymalna liczba popraw wag – określa maksymalną liczbę kroków, tzn. popraw wag sieci, jaką można przeprowadzić w procesie uczenia, po to, aby proces mógł się zatrzymać, gdy SSN nie chce się nauczyć (, gdy np. wpadnie w minimum lokalne).

Maksymalna liczba epok – określa maksymalną liczbę epok, jaka może być w procesie uczenia – również dla zatrzymania tego procesu.

Dla przyspieszenia procesu uczenia zastosowałem następujące zabiegi:

1. Dodanie **członu momentum,** czyli przy poprawianiu wag branie pod uwagę poprzedniej wartości poprawy danej wagi. Daje to bardzo duże przyspieszenie uczenia.
2. Dynamiczne ustalanie parametru eta: zmniejszanie jego wartości proporcjonalnie do liczby błędów w ostatniej epoce. Dodatkowy parametr A_Eta określa stopień tej zależności.
3. Losowo wybierany numer wzorca, od którego zaczyna się epoka.
4. Odpowiednio dobrane początkowo wylosowane wagi. Losowanie wag z pewnego zakresu (określane parametrami: rUk1,rUk2,rWy, mUk1,mUk2,mWy)
5. Ustalenie odpowiednio małych wartości początkowych biasów. Parametry: PoczBiasUk1 = 0.01, PoczBiasUk2 = 0.01, PoczBiasWy = 0.01
6. Możliwość zaburzenia wag podczas nauki, czyli pewnej niewielkiej losowej zmiany wag sieci, gdy sieć nie chce się uczyć, bo ugrzęzła w minimum lokalnym. Zaburzenie to jest określane przez parametry: MaxG = 20; MinR = 0.05; zaburzenie = 0.6, co oznacza, że jeżeli przez ostatnie 20 kroków popraw wag Suma epsilon nie zmieniła się o więcej niż 0.05, to należy zaburzyć wagi sieci o wartości losowe z przedziału [-0.6 ; 0.6].
7. Dodatkowy cykl popraw wag na końcu epoki dla jednego wzorca, który uporczywie powoduje błąd SSN.

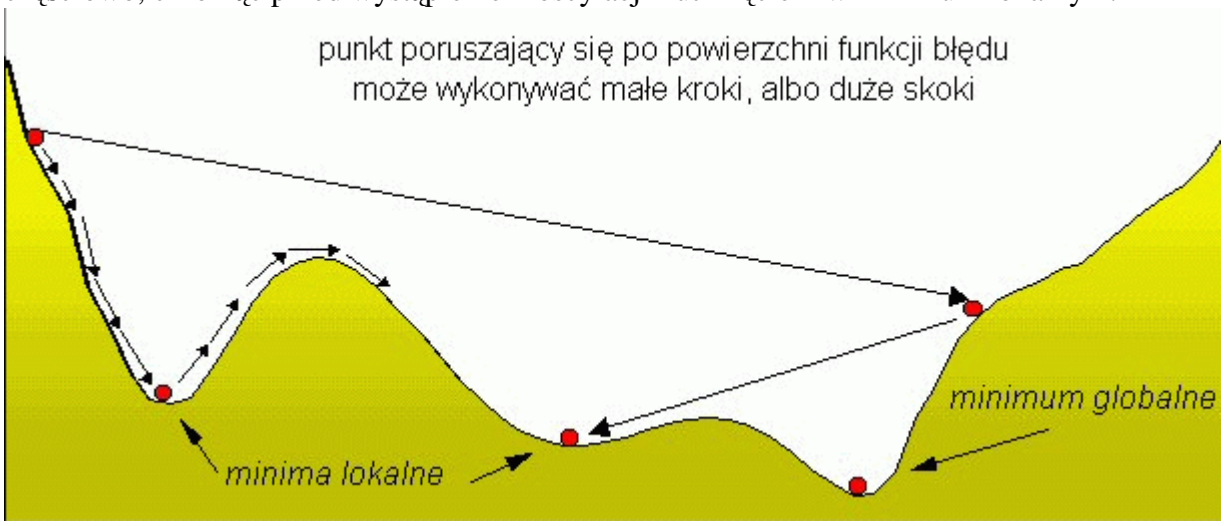
Momentowa metoda wstecznej propagacji błędów (MBP)

Klasyczna metoda wstecznej propagacji błędów wymaga dużej liczby iteracji by osiągnąć zbieżność oraz jest wrażliwa na występowanie minimów lokalnych. Podstawowy algorytm BackPropagation może się okazać zbyt wolny, jeżeli przyjmie się za mały współczynnik uczenia, z kolei zbyt duża jego wartość grozi wystąpieniem **oscylacji**. Jednym z rozwiązań umożliwiających bezpieczne zwiększenie efektywnego tempa uczenia bez pogarszania stabilności procesu jest zastosowanie momentowej metody wstecznej propagacji błędu – MBP lub MEBP (ang. Momentum Error BackPropagation).

Istotą metody jest wprowadzenie do procesu uaktualniania wagi pewnej **bezwładności** tzw. "momentu", proporcjonalnego do zmiany tej wagi w poprzedniej iteracji. Pojawienie się gradientu wymuszającego ruch w kierunku innym, niż aktualny nie od razu wpływa na zmianę trajektorii w przestrzeni wag. W przypadku, gdy kolejne gradienty są przeciwnie skierowane ich oddziaływanie częściowo się znosi. Powoduje to, że zmiana położenia w przestrzeni wag jest bardziej płynna i chwilowe zbyt duże zmiany są w rezultacie kompensowane.

Składnik momentu ma duże znaczenie dla płaskich odcinków funkcji błędu, dla których zwykle obserwuje się zmniejszenie szybkości uczenia. Dla płaskiego obszaru hiperpłaszczyzny funkcji miary błędu w przestrzeni wag, gdzie gradient w kolejnych iteracjach jest w przybliżeniu stały, przyrost wag w kolejnych iteracjach jest również stały. Gdy kolejne wektory gradientu mają ten sam kierunek ich działanie **kumuluje się**.

Moment odgrywa również pozytywną rolę w przypadku występowania na powierzchni funkcji błędu tzw. **wąwozów**, to znaczy wąskich obszarów o stromych ścianach bocznych i głębokich dnach. Moment pełni tu niejako rolę filtru dolnoprzepustowego dla zmian gradientu. Zmiany gradientu o wysokiej częstotliwości (oscylacje w poprzek ścian wąwozu) są eliminowane, a wzmacniany jest składnik gradientu wymuszający ruch w stronę dna. Modyfikacja momentowa BP ma ponadto korzystny wpływ na problem **minimów lokalnych**. W pobliżu takiego minimum składnik momentu, nie będąc związany z aktualną wartością gradientu, może spowodować zmianę wag prowadzącą do chwilowego wzrostu wartości funkcji błędu i w efekcie opuszczenia strefy "przyciągania" tego minimum.
Algorytm ten w rzeczywistości wykonuje wygładzanie poprawek zbioru wag, uzależnione od wartości współczynnika wygładzania - momentu.
Modyfikacja momentowa algorytmu propagacji wstecznej powoduje zatem przyspieszenie procesu uczenia przy pewnych stałych tendencjach zmian w przestrzeni wag jednocześnie, przynajmniej częściowo, chroniąc przed wystąpieniem oscylacji i utknięciem w minimum lokalnym.



TESTOWANIE POPRAWNOŚCI NAUKI SSN
Zakończenie się procesu nauki SSN może wystąpić z dwóch powodów

a) SSN nauczyła się, tzn. osiągnęła wymagane parametry, tzn. wartość **epsilon** i **Suma epsilon**. W tym przypadku program podaję liczbę przeprowadzonych kroków nauki.
b) SSN nie nauczyła się a osiągnięto już maksymalną liczbę iteracji (kroków nauki: popraw wag lub liczby epok). W tym przypadku program podaje liczbę błędów, tzn. przypadków, dla których błąd jest > epsilon.

Zakończenie procesu nauki nie musi oznaczać, że SSN nauczyła się poprawnie, tzn. że daje poprawne odpowiedzi dla uczonych wzorców (, bo np. parametr epsilon był za duży – np. >0,5). Zatem po zakończeniu procesu nauki, sprawdzana jest POPRAWNOŚĆ rozpoznawania uczonych wzorców, tzn. czy SSN poprawnie się ich nauczyła.
Polega to na podaniu na wejście sieci kolejno wszystkich uczonych wzorców i sprawdzaniu czy sieć daje oczekiwane wyniki.

Następną sprawą jest test możliwości generalizacji SSN, tzn. jak sieć radzi sobie z rozpoznawaniem nieuczonych przypadków.

Jeżeli sieć jest nauczona poprawnie, jej parametry (rozmiary sieci, wagi, współczynniki Beta) są zapisywane do pliku '**parametry.ssn**', który później jest wczytywany przez program do rozpoznawania obrazów. Rozmiar tego pliku zależy od rozmiaru SSN, tzn. liczby danych wejściowych i liczby neuronów w każdej warstwie i wynosi dokładnie w Bajtach:

48+(LiczbaNeuronówUk1*(LiczbaDanychWe+1) +
LiczbaNeuronówUk2*(LiczbaNeuronówUk1+1) +
LiczbaNeuronówWy*(LiczbaNeuronówUk2+1))*sizeof(double), gdzie rozmiar double = 8 Bajtów.

PRZETWARZANIE OBRAZU / TRANSFORMACJE

Obraz z podpisem z pliku BMP zanim zostanie podany na wejście SSN musi zostać odpowiednio przetworzony.
SSN ma określona liczbę wejść, zatem obraz ten należy przeskalować od tego właśnie rozmiaru.

Kolejne etapy operacji przetwarzania podanego obrazu podpisu:

1. Wczytanie pliku BMP do tablicy liczby typu unsigned char.
2. Rozmycie obrazu dla usunięcia ewentualnych zabrudzeń przy szukaniu ramki. Wynik zapisywany w drugim obrazie.
3. Znalezienie ramki (prostokąta) otaczającej ściśle linie podpisu.
4. Przycięcie oryginalnego obrazu do znalezionej ramki.
5. Rozmycie i normalizacja obrazu z ramki
6. Kontrastowanie (korekcja gamma) – rozjaśnienie tła i przyciemnienie linii podpisu
7. Skalowanie i wstawienie do wynikowego obrazu wypełnionego kolorem tła*
8. Ponowne kontrastowanie zależne od średniej jasności obrazu
9. Inwersja

* Możliwe warianty wkomponowania oryginalnego obrazu z BMP do określonego prostokąta:
1. Skalowanie do całego prostokąta.
2. Skalowanie z zachowaniem proporcji oryginału i wstawienie od lewego górnego rogu.
3. Skalowanie z zachowaniem proporcji oryginału i wstawienie z centrowaniem.
Tu zrobiono wariant nr 2.

Normalizacja kolorów polega na przeskalowaniu ich wartości do całego/pełnego przedziału [0 - 255].

Zbiór funkcji operujących na obrazie – tablicy elementów typu unsigned char, jest zawarty w jednym pliku: **UnitFunkcjeObraz.h/cpp,** który zawiera m.in.:
- **WczytajPlikBMP256BW**() – wczytanie obrazu do tablicy mObrazZrodlo
- **RozmyjObraz**() – rozmycie obrazu dla zamazania zabrudzeń; wynik rozmycia zapisywany jest w mObrazBufor
- **Normalizuj**() – przeskalowanie jasności pikseli do całego przedziału [0-255]
- **SzukajRamkiR()** – szuka ramki otaczającej podpis
- **PobierzObrazDoSSN2() –** przetwarza obraz o obrazu wynikowego, który ma być podawany do SSN

Funkcja SzukajRamkiR() pozwala znaleźć ramkę linii podpisu na różnych jasnościach tła i linii. Dlatego więc wymaga podania dwóch parametrów:
- **Roznica** (= 76); Jest to wartość progowa różnicy maks-min jasności pikseli na całej wysokości ramki.
Jeżeli różnica między najjaśniejszym i najciemniejszym pikselem w danej kolumnie jest większa od Różnica, to uznawane jest, że w tej kolumnie jest piksel nienależący do tła.
Przy wartości równej 256 parametr ten nie ma wpływu (wszystkie różnice jasności będą mniejsze).

- **ProgB** (= SredniaKolor*0.85); Jest to bezwzględna wartość progowa oddzielająca tło od linii.
Jeżeli jasność piksela jest mniejsza od ProgB, to uznawane jest, że należy on linii podpisu.
Zatem nawet, gdy wszystkie piksele w danej kolumnie są jaśniejsze od ProgB, to może ona zostać uznana, jako zawierająca linię, gdy jest odpowiednio duża różnica między jasnościami jej pikseli.

IMPLEMENTACJA:
Aplikacja została napisana w środowisku Borland C++ Builder 6.0.
Dlaczego wybrano język C++.
Uczenie SSN jest procesem wymagającym bardzo dużej liczby obliczeń a język C++ zapewnia najszybszy kod z języków wysokiego poziomu.

DIAGRAMY KLAS
PodpisyRozpoznawanie:

KBazaPodpisow – trzyma w pamięci obrazy
int mLOsob; int mLElementow; ELEMENT *mTabElem;
int WczytajDane(char *NazwaPliku); int WczytajBMP(char *NazwaPliku, unsigned char *ObrazW);

SIECN2H – wer. dla rozpoznawania
const int LDanychWe, LNeuronowWy; int LNeuronowUk1, LNeuronowUk2; double *DaneWe, *DaneWy;

void ObliczWynik(void);
int MaxWy(void);
int WczytajParam();

TFormMain – obsługa interfejsu

void RysujObraz();
void RysujPodpis(unsigned char *Obraz);
void SSNRozpoznaj(unsigned char *Obraz);

Tworzone globalnie obiekty (w pliku UnitFormMain.cpp):
TFormMain *FormMain;
KBazaPodpisow BPodpisow;;
SIECN2H *Siec;

WAŻNIEJSZE ELEMENTY PROGRAMU DO UCZENIA SSN:
Plik **KlasaBazaPodpisow.h/cpp** zawiera definicję klasy odpowiedzialnej za wczytanie i transformację obrazów podpisów oraz trzymanie ich w bazie danych.
W programie uczącym SSN zdefiniowano w pliku **KlasaSiecNmom2H.h/cpp** klasę **SIECN2H**, która tworzy SSN, uczy ją i ma możliwość zapisu i wczytywania parametrów nauczonej sieci.

W głównym pliku **UnitFormMain.cpp**, gdzie jest tworzony obiekt klasy SIECN2H są dwie funkcje przygotowujące dane do wejścia i oczekiwanego wyjścia SSN:
double* **FunPX**(int n) //zwraca wskaźnik na tablicę danych wEjściowych n-tego wzorca
double* **FunPZ**(int n) //zwraca wskaźnik na tablicę danych wYjściowych n-tego wzorca

* Utworzenie SSN następuje przez utworzenie obiektu klasy SIECN2H konstruktorem:
SIECN2H(int _LDanychWe, int _LNeuronowUk1, int _LNeuronowUk2, int _LneuronowWy)
Parametrami są liczby neuronów w kolejnych warstwach SSN.

* Po utworzeniu SSN należy ją **zresetować**, tzn. ustalić współczynniki Beta dla każdej z warstw oraz ustalić wartości wag neuronów na losowe, bowiem zaczynając od **losowo** ustalonych wartości wag, SSN uczy się najszybciej!
Dokonujemy tego metodą:
int **SIECN2H::Resetuj**(double _BetaUk1, double _BetaUk2, double _BetaWy,
  double PoczBiasUk1, double PoczBiasUk2, double PoczBiasWy,
  double rUk1, double rUk2, double rWy, //do ustalania początkowych wag
  double mUk1, double mUk2, double mWy); //do ustalania początkowych wag

Parametry funkcji, to:

BetaUk1, BetaUk2, BetaWy – wartości beta funkcji aktywacji kolejnych trzech warstw sieci
PoczBiasUk1, PoczBiasUk2, PoczBiasWy – początkowe wartości bias dla wag kolejnych warstw sieci
rUk1,rUk2,rWy, mUk1,mUk2,mWy – określają sposób losowania wag: zakres i przesunięcie

* Proces nauki SSN rozpoczynamy wywołaniem funkcji:
SIECN2H::LearnAll(int LWzorcow,
double* FpX(int), double* FpZ(int), //ustala dane wejściowe i wyjściowe
double EtaUk1, double A_EtaUk1, double AlfaUk1,
double EtaUk2, double A_EtaUk2, double AlfaUk2,
double EtaWy, double A_EtaWy, double AlfaWy,
double MinR, int MaxG, double zaburzenie, //kryterium i parametr zaburzenia
double eps, double Sum_eps,
int MaxLPopraw, int MaxLEpok, //kryterium STOPu
void PokazOpis(char *info)) //zewnętrzna funkcja wywoływana po każdej epoce
Kolejne parametry, to:
- LWzorcow – liczba wzorców do nauczenia
- FpX, FpZ – funkcje przygotowujące i ustalające wartości wejściowe i wyjściowe sieci
- EtaUk1, A_EtaUk1, AlfaUk1, EtaUk2, A_EtaUk2, AlfaUk2, EtaWy, A_EtaWy, AlfaWy – wektor nauki i sposób jego dynamicznej modyfikacji dla kolejnych warstw
- MinR, MaxG, zaburzenie – określają, kiedy i jak ma być dokonane zaburzenie na wagach sieci, (gdy sieć ugrzęźnie w jakimś minimum lokalnym i nie chce się dalej uczyć)
- **eps, Sum_eps** – epsilon i suma epsilon – warunki, jakie ma spełniać nauczona sieć
- MaxLPopraw, MaxLEpok – kryterium zatrzymania procesu uczenia (dla przerwania nauki, gdy sieć nie chce się nauczyć)
- PokazOpis(char *info) – funkcja, która będzie wywoływana po każdej epoce; parametr info będzie zawierał łańcuch opisujący aktualny stan nauki SSN

Proces nauki SSN polega na odpowiednim zmienianiu wag sieci, tzn. liczb oznaczających połączenie między neuronami (stopień wpływu jednego sygnału na inny). Wartości tych wag ustala się początkowo losowo, aby móc rozpocząć naukę od jakiegoś miejsca. Wpływ na sposób losowania tych wartości uzyskujemy przez ustalanie parametrów: rUk1, rUk2, rWy oraz mUk1, mUk2, mWy. Wpływają one na wartości losowanych początkowo wag sieci dla każdej warstwy osobno:
Uk1- pierwsza warstwa ukryta
Uk2 - druga warstwa ukryta
Wy - warstwa wyjściowa

Sam proces poprawiania wag względem obliczonych przez SSN danych wyjściowych a oczekiwanych odbywa się w funkcji:
SIECN2H::PoprawWagi (const double *DaneOcz,
double etaUk1, double alfaUk1,
double etaUk2, double alfaUk2,
double etaWy, double alfaWy)

Jest to najważniejsza funkcja (jądro obliczeń) składająca się na naukę SSN. Tutaj zaimplementowany jest ważny algorytm **minimalizacji funkcji błędu** - metoda delty i tu obliczany jest gradient tej funkcji.

Funkcja obliczająca wynik SSN dla ustawionych wcześniej funkcją FunPX(nr_wzorca) danych wejściowych, to:
Wynik(void) – ustala wyjście SSN, tzn. wartości w tablicy **DaneWy[]**

W programie do **samego tylko rozpoznawania** obrazów (bez nauki) użyto nieco zmienionej klasy SIECN2H, tzn. bez metod: LearnAll i służącej do zapisywania parametrów sieci.
Deklaracja i definicja tej kasy są zawarte w plikach **KlasaSiecNmom2HRozp.h/cpp.**

KONKRETNE PROGRAMY
Projekt składa się z dwóch osobnych programów:
1. Do nauki SSN: UczObrazy.exe
2. Do rozpoznawania podpisów i testowania nauczonej SSN: PodpisyRozpoznawanie.exe

Instrukcja używania programów
1. UczObrazy.exe
Program ten służy do tworzenia odpowiedniej SSN i uczenia jej rozpoznawania podanych obrazów podpisów.
Aby określić, które obrazy i których osób SSN ma się uczyć, należy stworzyć plik tekstowy np. do_nauki.txt, w którym trzeba zapisać nazwiska osób i kolejne pełne nazwy (ścieżki) plików bitmap z obrazami ich podpisów.
Struktura takiego pliku musi być taka, jak w przykładzie. Tutaj dla dwóch osób:
#Artur Czekalski
artur01.bmp
artur02.bmp
artur03.bmp
artur04.bmp
artur05.bmp
#Józef Kowalski
jozef01.bmp
jozef02.bmp
jozef03.bmp

W liniach zaczynających się znakiem # umieszczamy nazwę właściciela podpisu i w kolejnych liniach pliki .bmp z jego podpisami (dowolna ilość). Każdy obraz podpisu musi zawierać pojedynczy podpis.

Tak utworzony plik należy wczytać do programu UczObrazy.exe przyciskiem '**Wczytaj wzorce**'. Program zbuduje na jego podstawie bazę odpowiednio przetransformowanych obrazów, które będzie można podawać na wejście SSN oraz zdefiniuje odpowiednie rozmiary SSN:
- liczba wejść – równa będzie liczbie pikseli wynikowego obrazu (ustalona w projekcie)
- liczba wyjść – równa będzie liczbie osób w bazie (ustalona w podanym pliku do nauki)

Pozostaje do ustalenia liczba neuronów w dwóch warstwach ukrytych. Należy pamiętać o zasadzie piramidy, tzn. liczby neuronów w kolejnych warstwach powinny maleć.
Po zdefiniowaniu rozmiarów SSN można utworzyć SSN przyciskiem **'Utwórz nową SSN'**.

Następnie należy ustalić parametry służące do **przygotowania SSN** do procesu nauki, tzn. parametry beta funkcji aktywacji, wartości wag dla 'bias' oraz współczynniki losowania wartości wag. Jest tutaj możliwość zdefiniowania wszystkich parametrów dla każdej warstwy osobno. Po ich zdefiniowaniu należy 'zresetwoać' SSN za pomocą przycisku **'Resetuj SSN'**. Spowoduje to ustalenie odpowiednich losowych wartości wszystkich wag w SSN oraz ustalenie parametrów beta funkcji aktywacji.

Pozostaje już tylko ustalenie **parametrów procesu uczenia** SSN. Dla każdej warstwy osobno ustalamy tzw. '**wektor uczenia**', czyli parametry: Eta, A_Eta i Alfa.

Kolejna grupa to **parametry zaburzenia**, czyli kiedy ma nastąpić zaburzenie ('Min różnica', 'MaxG') i jak duże ma być ('Wielkość').

Następna grupa to **wymagania**, jakie stawiamy nauczonej SSN, czyli maksymalny błąd SSN dla każdego wzorca oraz maksymalna suma błędów SSN ze wszystkich wzorców.

Ostatnią grupę stanowią **parametry stopu**, tzn. kiedy proces uczenia SSM ma się automatycznie zatrzymać: maksymalna liczba popraw wag oraz maksymalna liczba przeprowadzonych epok.

Po ustaleniu tych parametrów przyciskiem **'Ucz SSN'** uruchamia się proces nauki SSN.
Jeśli będzie zaznaczona opcja **'Zapisać nauczoną SSN'**, to po poprawnym nauczeniu SSN, będą automatycznie zapisane w katalogu z programem parametry nauczonej SSN.
Można także przyciskiem **'Zapisz nauczną SSN'** zapisać we wskazanym miejscu parametry nauczonej SSN.

Przyciski **'Ostatnie ustawienia'**, służą do ustalenia poprzednich wartości parametrów w odpowiednich grupach.
W trakcie procesu uczenia SSN pokazywane są szczegóły postępu nauki, zmieniające się parametry oraz ich wizualizacja.
Pozwala to śledzić jak zmienia się wiele kluczowe dla nauki wartości sieci. Pomaga to w dobraniu odpowiednich parametrów nauki.
Przykład pojedynczego wiersza opisującego stan po przejściu jednej epoki:

```
Epk:  26; LB: 15; Nrp:   9; MaxB:0.609109; SumB=8.23149034; SO-
SE=-2.2349127 | Eta: Uk1=0.0777778; Uk2=0.0577778; Wy=0.0333333
```

Kolejne elementy wiersza zawierają:
Epk - Numer kolejnej epoki
LB – liczba błędów w ostatniej epoce
Nrp – nr ostatniego wzorca w epoce, dla którego był błąd (Uwaga: nr wzorca, od którego zaczynamy epokę jest losowy, więc ten ostatni, to nie musi być ostatni w kolejności tak, jak w bazie danych.)
NrpMaxB – nr wzorca, dla którego był największy błąd
MaxB – wartość maksymalnego błędu dla pojedynczego wzorca w ostatniej epoce
SumB – Suma błędów ze wszystkich wzorców w całej epoce
SO-SE – różnica między sumą błędów w poprzedniej epoce i obecnej
EtaUk1, Uk2, Wy – bieżące wartości parametrów eta dla każdej warstwy (zmieniają się dynamicznie; zależą od liczby błędów w ostatniej epoce)

Po poprawnym nauczeniu SSN program zapisuje do pliku '**ParametryNaukiSSN.txt**' wszystkie parametry nauki SSN i wiele innych informacji o nauce (m.in. czas nauki, liczbę kroków itp.). Dzięki temu można szukać optymalnych parametrów nauki i rozmiarów SSN.

Po utworzeniu odpowiedniej SSN i jej poprawnym nauczeniu, jako wynik zapisane będą dwa pliki:
- **parametry.ssn** – zawiera dane i parametry nauczonej SSN (rozmiar, współczynniki beta, wagi)
- **nazwiska.txt** – zawiera liczbę i nazwy osób, których podpisów SSN się nauczyła
Oba te pliki należy umieścić w katalogu z programem PodpisyRozpoznawanie.exe. Stanowią one **źródło wiedzy** dla programu rozpoznającego.

WYKRES NAUKI SSN:
Program do nauki SSN, jako wizualizację przedstawia na wykresie bieżące wartości zmieniających się niektórych parametrów i danych nauki, w odpowiedniej skali (/):

- **Liczba błędów /** liczba wszystkich wzorców
- **Suma błędów /** liczba wszystkich wzorców
- **Eta: Uk1, Uk2, Wy /** max {EtaUk1, EtaUk2, EtaWy}
- **Max Błąd /** liczba wyjść/2.0

2. PodpisyRozpoznawanie.exe
Program przy uruchamianiu wczytuje oba pliki stworzone przez program do uczenia i na ich podstawie tworzy odpowiednią SSN, która służyć będzie do rozpoznawania podawanych obrazów. Z pliku **nazwiska.txt** pobierane są nazwy osób dla przyporządkowania ich wynikom SSN.

Przyciskiem '**Rozpoznaj podpis**' wskazujemy plik, który ma być rozpoznany przez SSN. Wskazany obraz zostanie odpowiednio przetransformowany i podany na wejście SSN. Obliczony wynik SSN (Nr osoby) i przyporządkowana mu nazwa osoby zostanie wyświetlona na ekranie (pole Osoba).

Przyciskiem '**Wczytaj bazę podpisów**' można wczytać plik opisu bazy, który został utworzony do nauki SSN.
Dzięki temu można także przetestować działanie SSN dla nauczonych obrazów podpisów.
Przyciski '**<<**' i '**>>**' służą do przeglądania obrazów z bazy i rozpoznawania ich przez SSN.
OSIĄGNIĘTA SPRAWNOŚĆ PROGRAMU
Sieć Neuronowa uczy się bez problemów i bardzo szybko.
Również dobrze generalizuje wiedzę i często udaje się jej rozpoznawać poprawnie obrazy podpisów, których nie była uczona.

SZYBKOŚĆ NAUCZENIA SSN - PRZYKŁADY:
(Procesor AMD Duron 900 Mhz, Pamięć 384 MB PC100)
- 27 obrazów; 4 osoby; Rozmiar SSN: 2304-70-30-4; Nauczona w: 7 epok; 131 popraw wag; 5 sekund;
- 27 obrazów; 4 osoby; Rozmiar SSN: 2304-60-30-4; Nauczona w: 9 epok; 104 popraw wag; 4 sekundy;
- 38 obrazów; 6 osoby; Rozmiar SSN: 2304-60-50-6; Nauczona w: 10 epok; 240 popraw wag; 7 sekund;
- 44 obrazy; 7 osób; Rozmiar SSN: 2304-60-20-7; Nauczona w: 17 epok; 324 popraw wag; 11 sekund;
- 47 obrazów; 7 osób; Rozmiar SSN: 2304-70-20-7; Nauczona w: 16 epok; 335 popraw wag; 15 sekund;
- 49 obrazów; 7 osób; Rozmiar SSN: 2304-40-20-7; Nauczona w: 15 epok; 407 popraw wag; 9 sekund;
- 56 obrazów; 8 osób; Rozmiar SSN: 2304-60-40-7; Nauczona w: 13 epok; 469 popraw wag; 10 sekund;
- 61 obrazów; 9 osób; Rozmiar SSN: 2304-30-20-9; Nauczona w: 21 epok; 641 popraw wag; 8 sekund;
- 71 obrazów; 11 osób; Rozmiar SSN: 2304-40-20-11; Nauczona w: 31 epok; 847 popraw wag; 17sek;
- 77 obrazów; 12 osób; Rozmiar SSN: 2304-50-40-12; Nauczona w: 24 epoki; 815 popraw wag; 22sek;
- 79 obrazów; 12 osób; Rozmiar SSN: 2304-**40-30**-12; Nauczona w: **17** epok; **771** popraw wag; 14sek;
- 80 obrazów; 12 osób; Rozmiar SSN: 2304-**30-20**-12; Nauczona w: **30** epok; **1190** popraw wag; 18sek;

Zauważmy, że zwiększenie liczby neuronów w warstwach ukrytych może znacząco poprawić szybkość uczenia SSN.

- 82 obrazy; 12 osób; Rozmiar SSN: 2304-30-20-12; Nauczona w: 26 epok; 1047 popraw wag; 14sek;
- 83 obrazy; 12 osób; Rozmiar SSN: 2304-30-20-12; Nauczona w: 22 epok; 947 popraw wag; 14sek;

Wszystkie kody źródłowe zostały napisane od podstaw. Oprócz tej dokumentacji wiele opisów i wyjaśnień jest w samych komentarzach do kodów źródłowych.
Autor: Artur Czekalski; ARTUR@epokaY.net www.epokay.net/artur 29d-11m-2005