

**Aplikacja realizująca:**  
**Rozpoznawanie pisma odręcznego z wykorzystaniem sieci neuronowych**  
**Wersja 2 (dodatkowe informacje o ramce znaku)**

**Oznaczenia:**

- SSN – Sztuczna Sieć Neuronowa
- Ten kolor oznacza informacje dotyczące tego konkretnego problemu, z którym ma poradzić sobie SSN (rozpoznawanie znaków).
- Używam zapisu 0,0 i 1,0 aby zaznaczyć, że są to liczby rzeczywiste a nie całkowite – odpowiednio 0 i 1.
- Ramka znaku – prostokąt opisany na znalezionym znaku w obrazie

**Założenia:**

1. SSN będzie rozpoznawać następujące znaki:

- qwertyuiopasdfghjklzxcvbnm óąśźźźź - 35 znaków x2 dla wielkich liter – tj. 70 oraz
  - 10 cyfr oraz
  - znaki specjalne: spacja, przecinek, kropka, dwukropek, średnik, wykrzyknik, pytajnik, kreska, 2 nawiasy okrągłe - 10 znaków
- Razem 90 znaków.

2. Pliki zeskanowanych obrazów podawane na wejście będą w formacie BMP (bitmapa) w 256 kolorach szarości (8bpp).

(Bardzo wygodny program do przeprowadzenia tego konwersji plików graficznych, to np. IrfanView [www.irfanview.com](http://www.irfanview.com))

Obraz z bitmapy może być różnych rozmiarów (maksymalnie 900x160 pikseli).

3. Kolejne znaki do rozpoznawania będą na obrazie w jednym wierszu. **Znaki nie mogą stykać się ze sobą!**

4. Maksymalna liczba znaków, od lewej, brana pod uwagę to 25.

5. Znaki muszą być narysowane ciemną linią na jasnym tle.

**SZTUCZNA SIEĆ NEURONOWA**

Wynikowy obraz znaku podawany na wejście SSN jest rozmiaru 16x16 pikseli o wartościach kolorów 0 – 255.

Kolory są poddane inwersji, tzn. wartości bliskie 0 reprezentują tło, a wartości bliskie 255 reprezentują linię znaku.

Wektor podawany na wejście SSN, to wartości jasności pikseli wynikowego obrazu, przeskalowane do przedziału [0,0 ; 1,0] oraz **dodatkowe informacje o ramce znaku** (wielkość i położenie) – 5 wartości.

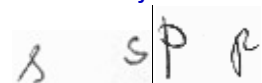
Dzięki inwersji kolorów pikseli więcej będzie wartości bliskich 0,0 a mniej wartości bliskich 1,0. Przyspieszy to i ułatwi proces nauki SSN.

Mamy zatem  $16 \cdot 16 = 256$  **oraz 5 wartości, to jest razem 261 neuronów w warstwie wejściowej.**

**Warstwa wyjściowa** zawiera tyle neuronów, ile różnych znaków ma rozpoznawać – po jednym neuronie na każdy znak, czyli u nas **90 neuronów.**

Można by niektóre obrazy danego znaku podzielić na kilka grup, gdyż np. literę małe s różnie się zapisuje: podobnie do wielkiego S lub tak jak dzieci uczą się w szkole – typowe małe s.

Podobnie jest z literą małe p.



Te różne style zapisu tego samego znaku można by traktować jako osobne znaki i przydzielić im osobne neurony w SSN w warstwie wyjściowej.

Być może to ułatwiło by SSN naukę i rozpoznawanie tych liter.

Jednak uważam, że SSN o odpowiedniej wielkości (liczbie neuronów) poradzi sobie w takich przypadkach i da się ją nauczyć dawania tej samej odpowiedzi na wyjściu dla całkiem odmiennych sposobów rysowania tego samego znaku.

Oprócz informacji o jasnościach kolejnych pikseli obrazu, na wejście SSN **podawanych jest 5 liczb o położeniu i wielkości ramki danego znaku.**

Są to wartości z przedziału [0.0 ; 1.0] i reprezentują **informacje o ramce danego znaku:**

1. **środek pozycji pionowej** ramki względem **środka wysokości ramki ciągu znaków**; Def. 0.5-środek ramki; 0-wyżej; 1-niżej
2. **środek pozycji pionowej** ramki względem **średniej środków wysokości** wszystkich znaków; Def. 0.5-średnia środków; 0-wyżej; 1-niżej
3. **wysokość ramki** względem **wysokości ramki ciągu znaków**; Def. 0.5; 1-taka sama; 0-dużo mniejsza
4. **wysokość ramki** względem **maksymalnej wysokości ramek** wszystkich znaków; Def. 0.5; 1-maks. wysokość; 0-dużo mniejsza
5. **wysokość ramki** względem **średniej wysokości ramek** wszystkich znaków; Def. 0.5-średnia wysokość; 1-większa od średniej; 0-dużo mniejsza

Zatem:  
**Wektorem wejściowym** jest  $256+5=261$  liczb rzeczywistych, o wartościach z przedziału [0,0 ; 1,0].

**Wektorem wyjściowym** jest 90 liczb z przedziału [0,0 ; 1,0].

W przypadku rozpoznania obrazu znaku przez SSN, jedna liczba, której numer reprezentuje dany znak ma wartość równą 1,0. Pozostałe mają wartość 0,0.

Jeżeli obraz nie zostanie rozpoznany, to wszystkie liczby mają wartość 0,0.

Przykładowe wektory wyjściowe:

- [1,0,0,0,...,0] oznacza odpowiedź „znak nr 0”
- [0,0,1,0,...,0] oznacza odpowiedź „znak nr 2”
- [0,0,0,0,...,1] oznacza odpowiedź „znak nr - ostatni”
- [0,0,0,0,...,0] oznacza odpowiedź „nieznany znak”

Wzorcami w nauce SSN, dla odpowiedzi „**nieznany znak**” – będą wygenerowane losowe obrazy (tzn. losowe kolory pikseli) oraz pewne szablonowe np. cały czarny – wszystkie wartości pikseli równe 255 lub cały biały – wszystkie wartości równe 0.

## ELEMENTY PROJEKTU:

1. **Baza obrazów znaków** wyodrębnionych z przykładowych obrazów zeskanowanych **do pliku BMP w 256-kolorach (szarości lub zwykłych).**

Baza zawarta jest w pliku **BazaObrazowZnakow.boz:**

### Struktura pliku bazy:

Baza zawiera obrazy znaków w rozdzielczości 16x16 pikseli oraz dodatkowe informacje o ich ramce.

Każdy rekord składa się z **297** bajtów:

- pierwsze 256 bajtów to kolejne wartości pikseli (0-tło (biały kolor), 255-linia znaku (czarny kolor)) ustawione wierszami od góry.
- następnie mamy 5 liczb typu double (po 8 bajtów) z informacją o ramce znaku, czyli w sumie 40 bajtów
- ostatni bajt, to etykieta – nr znaku w kodzie ASCII.

2. Program do **edycji bazy obrazów znaków:** Umożliwia:

- nadawanie etykiet obrazom znaków (czyli ustalanie jaki znak przedstawia dany obraz)
- usuwanie obrazu znaku z bazy

3. Program do **nauki SSN** rozpoznawania znaków na podstawie wzorców z bazy obrazów znaków. Jeżeli SSN zostanie nauczona poprawnie, jej parametry (rozmiary sieci, wagi, współczynniki Beta) są zapisywane do pliku '**parametry.ssn**', który później jest wczytywany przez program do rozpoznawania obrazów.
4. Program do **rozpoznawania kolejnych liter na formularzu**. Program powinien rozpoznawać obrazy znaków, które **nie zostały użyte do nauki SSN**. Program przy uruchamianiu tworzy na podstawie pliku '**parametry.ssn**' odpowiednich rozmiarów SSN i wczytuje tam zapisane parametry nauczonej wcześniej SSN. Plik ten musi być w katalogu z programem. Program ten **wyodrębnia poszczególne obrazów znaków** z pliku BMP zeskanowanego formularza oraz przekształca je na postać gotową dla SSN. Umożliwia dopisanie, dodanie wyodrębnionych znaków do bazy obrazów. W ten sposób nierozpoznane przez program znaki, można dopisać do bazy znaków aby potem nauczyć ponownie SSN.

## PROBLEMY DO ROZWIĄZANIA:

Wejście: plik zeskanowanego formularza.

- I. Wyodrębnić poszczególne obrazy znaków z całego obrazu.
  1. Ustalić ramki poszczególnych znaków, tzn. prostokąty opisane na kolejnych znakach.
  2. Przekształcić obraz znaku to standardowej wielkości 16x16 pikseli.
  3. Przygotować obraz na wejście SSN
- II. Nauczyć SSN rozpoznawać znaki

Wyjście: ciąg rozpoznanych znaków (identyfikatory w kodzie ASCII).

## Szczegółowy proces przygotowania obrazu do wejścia SSN:

1. Rozmycie wczytanego obrazu BMP dla zamazania ewentualnych zabrudzeń, plamek itp. Zapisanie wyniku przetworzenia w obrazie roboczym.
2. Znalezienie w obrazie roboczym **ramki ciągu znaków** i **ramek pojedynczych znaków**, tzn. położenia prostokątów opisanych na tych znakach. Własny algorytm (skanowanie liniami).
3. Wpisanie zawartości danej ramki znaku z obrazu źródłowego do **kwadratu** – dla zachowania proporcji rozmiarów.
4. Normalizacja obrazu znaku – przeskalowanie kolorów do całego przedziału [0 – 255].
5. Kontrast obrazu znaku – rozjaśnienie jasnych pikseli i ściemnienie ciemnych – względem granicy - średniej jasności kolorów w ramce (korekcja Gamma)
6. Skalowanie kwadratu ze znakiem do wielkości 16x16: Własny **algorytm o najlepszej jakości skalowania** (zwłaszcza przy zmniejszaniu). Potrzebny aby uzyskać, jak najlepszej jakości obraz na wejściu SSN.
7. Ponowna normalizacja kolorów.
8. Ponowny kontrast – (względem średniej jasności kolorów w ramce).
9. Inwersja kolorów.
10. Przeskalowane wartości kolorów pikseli do przedziału [0,0 ; 1,0].

Dzięki temu, że bierzemy do skalowania obraz znaku w rozmiarze wyznaczonym przez jego ramkę (punkty 2. i 3.) (a nie w jakimś stałym rozmiarze) wynikowe obrazy, po przeskalowaniu będą zawsze wypełniały cały kwadrat 16x16 pikseli. Nie będą one wtedy różniły się rozmiarem a **tylko kształtem!**

SSN rozróżnia znaki na podstawie kształtu oraz na podstawie pozycji i wielkości ramki znaków. Przez to SSN może mieć łatwiej rozróżnieniu np. znaki: g i 9 lub p i P.

## Inne trudności rozpoznawania znaków:

Z powodu sposobu zapisywania w piśmie ręcznym niektórych znaków, zapewne będą trudności w odróżnieniu przez program następujących grup znaków:

- c, C (, l
- ć, Ć
- l, J
- o, O, 0
- ó, Ó
- p, P
- Q, a
- s, S
- ś, Ś
- t, ł
- u, U
- n, u
- w, W
- y, Y, J
- z, Z, 2
- ź, Ź
- ż, Ż
- x, X
- y, J
- ; oraz i
- , oraz )

### PROGRAMY WCHODZĄCE W SKŁAD APLIKACJI:

Oznaczenie: [Przycisk] – etykieta przycisku w programie

#### Uwaga.

Baza obrazów znaków powinna zawierać obrazy „reprezentacyjne” - wzorcowe, tzn. takie, na podstawie których SSN będzie mogła rozpoznawać inne, nie zawarte w bazie obrazy.

Zatem do bazy znaków warto zapisywać **jedynie odpowiednio poprawnie narysowane znaki** (tzn. takie, które samemu można bez problemu jednoznacznie rozpoznać). Nie powinno być w bazie jakiegś bardzo niezgrabnie narysowanych znaków.

#### Program 2 – EdycjaBazyZnakow.exe

- wczytuje plik z Bazą znaków
- pozwala na przeglądanie obrazów znaków i ich etykiet [<<], [>>]
- umożliwia zmianę etykiety znaku [Zmień etykietę na]
- umożliwia usunięcie znaku z bazy [Usuń element]
- Zapisuje dokonane zmiany w bazie znaków [Zapisz bazę]
- Usunięcie z bazy kopii takich samych obrazów w bazie [Usuń kopie takich samych obrazów]

#### Program 3 – UczenieSSNObrazow.exe

Implementacja SSN i uczenie jej wzorcami z bazy obrazów (plik BazaObrazowZnakow.boz).

W trakcie działania programu, pokazywany jest proces uczenia się SSN.

Pozwala to śledzić jak zmieniają się niektóre kluczowe wartości sieci.

Kolejne wiersze zawierają:

**Ep**k - Numer kolejnej epoki

**LB** – liczba błędów w ostatniej epoce

**Nrp** – nr ostatniego wzorca w tej epoce, dla którego był błąd (Uwaga: nr wzorca, od którego zaczynamy epokę jest losowy, więc ten ostatni, to nie musi być ostatni w kolejności jak w bazie danych)

**NrpMaxB** - nr wzorca o maks. błędzie w epoce

**MaxB**: - maks. wartość błędu w epoce

**SumB** – Suma błędów ze wszystkich wzorców w całej epoce

**SO-SE** – różnica między sumą błędów w poprzedniej epoce i obecnej

**EtaUk1, EtaUk2, EtaWy** – bieżące wartości parametrów eta dla każdej warstwy (zmienia się dynamicznie; zależy od liczby błędów w ostatniej epoce)

Po poprawnym nauczaniu SSN program zapisuje do pliku **ParametryNaukiSSN.txt** wszystkie parametry SSN i wiele innych informacji o nauce.

Uwaga. Jeśli SSN nie chce się nauczyć i często występuje błąd dla wzorca nr **Nrp** (wyświetlany w kolejnych liniach pola tekstowego programu do nauki), to być może warto taki wzorzec usunąć z bazy znaków.

#### **Program 4 – RozpoznawanieObrazow.exe**

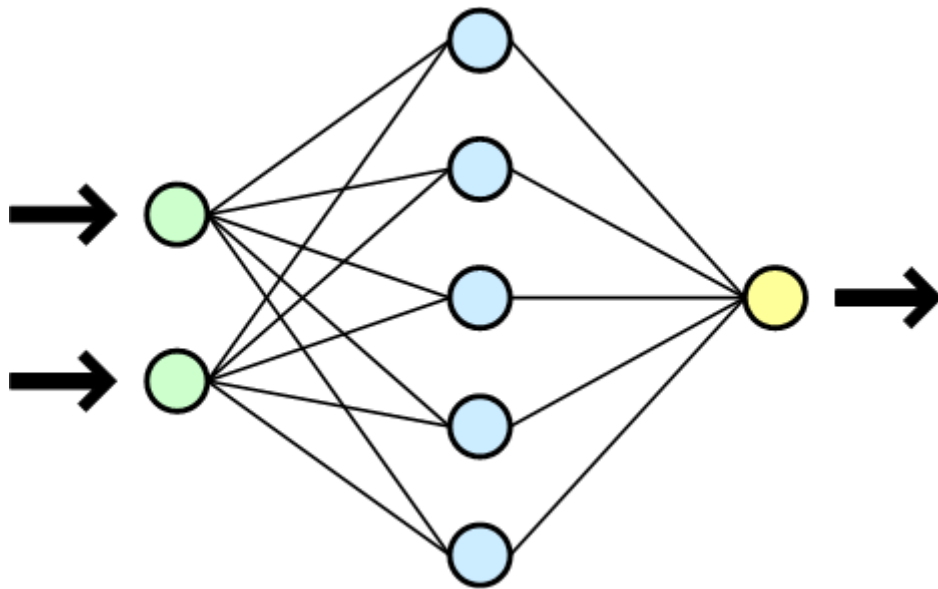
Ostateczny program do rozpoznawania znaków z podanego obrazu z pliku BMP.

Program przy uruchamianiu od razu tworzy na podstawie pliku **parametry.ssn** odpowiednich rozmiarów SSN i wczytuje, tam zapisane, parametry nauczonej wcześniej SSN. Plik ten musi być w katalogu z programem.

Program:

- Wczytuje plik BMP z obrazem znaków [Wczytaj BMP]
- Wyszukuje w obrazie z podanego pliku BMP obrazy znaków (tzw. ramki znaków) [Szukaj ramek]
- Obrazy znaków transformuje (skalowanie, kontrast, normalizacja itp.) do standardowej postaci gotowej dla wejścia SSN
- Umożliwia dopisanie obrazu znaku wraz z podaną etykietą i informacjami o jego ramce do wskazanego pliku (BazaObrazowZnakow.boz) [Dopisz do bazy]

#### **DZIAŁANIE SZTUCZNEJ SIECI NEURONOWEJ**



Uproszczony schemat jednokierunkowej sieci neuronowej

**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** (wewnętrzne) i warstwa danych wyjściowych. Użycie dwóch warstw ukrytych zamiast tylko jednej znacząco poprawia uczenie się SSN i jej możliwości generalizacji. Trzy warstwy to już przesada, a dwie w zupełności wystarczają do poprawnej nauki.

Połączenia neuronów między warstwami - każdy z każdym.

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

**Warstwy ukryte** (wewnętrzne) zawierają po około 160 i 110 neuronów – w tym wypadku sieć wydaje się uczyć najszybciej.

Liczbę neuronów w każdej warstwie ukrytej można 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).

Nieraz przyjmuje się, że **liczba neuronów w warstwie ukrytej** wynosi około:

$\sqrt{\text{Liczba wejść} * \text{Liczba wyjść}}$ , czyli pierwiastek z iloczynu liczb neuronów sąsiednich warstw.

U nas  $\sqrt{261*90}=153$ , stąd  $261 > L. \text{ neuronów } 1 \text{ warstwy ukr.} > \sim 153 > \sim L. \text{ neuronów } 2 \text{ warstwy ukr.} < 90$

Zatem SSN może ostatecznie mieć rozmiar np.: 261 - 160 - 110 - 90.

Generalnie jednak uczenie rozpoczyna się z małą liczbą neuronów w warstwach ukrytych a następnie, obserwując postępy tego procesu, doświadczalnie zwiększa się ich liczbę.

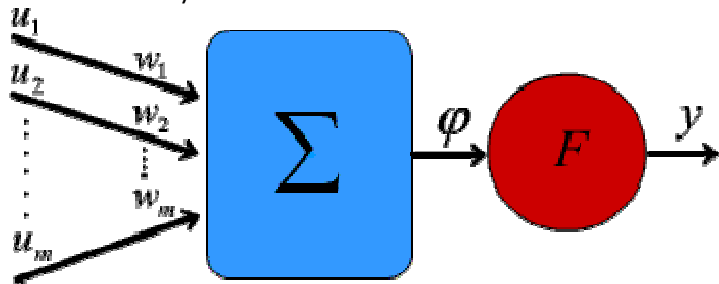
**Funkcją aktywacji** neuronu jest funkcja sigmoidalna unipolarna.

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).



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:

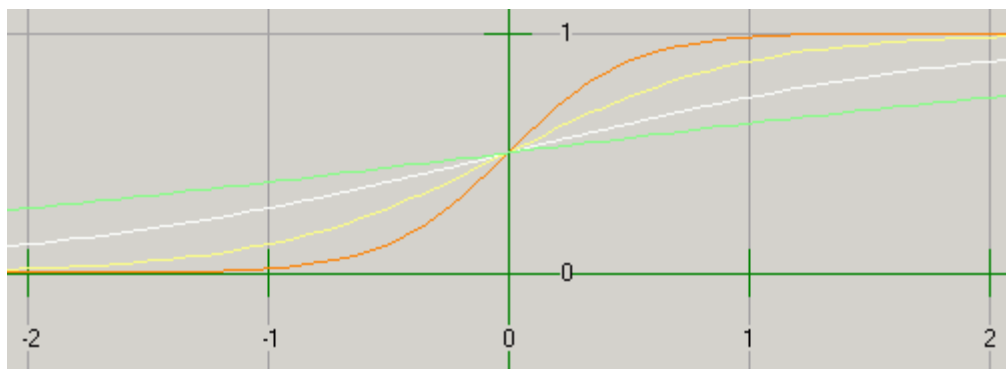
- ciągłe przejście pomiędzy swoją wartością maksymalną a minimalną (np. 0.0 - 1.0),
- łatwa do obliczenia i ciągła pochodna,
- możliwość wprowadzenia do argumentu parametru beta do ustalania kształtu krzywej.

Użytą funkcją aktywacji neuronu jest funkcja **sigmoidalna unipolarna**:  $f(x, \beta) = \frac{1.0}{1.0 + e^{-\beta * x}} \in [0.0 ; 1.0]$ ;

gdzie  $x = u_1 * w_1 + \dots + u_m * w_m$

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)  $\beta = 1.0$ ;

$\beta = 2.0$ ;

$\beta = 4.0$ ;

$\beta = 0.5$ ;

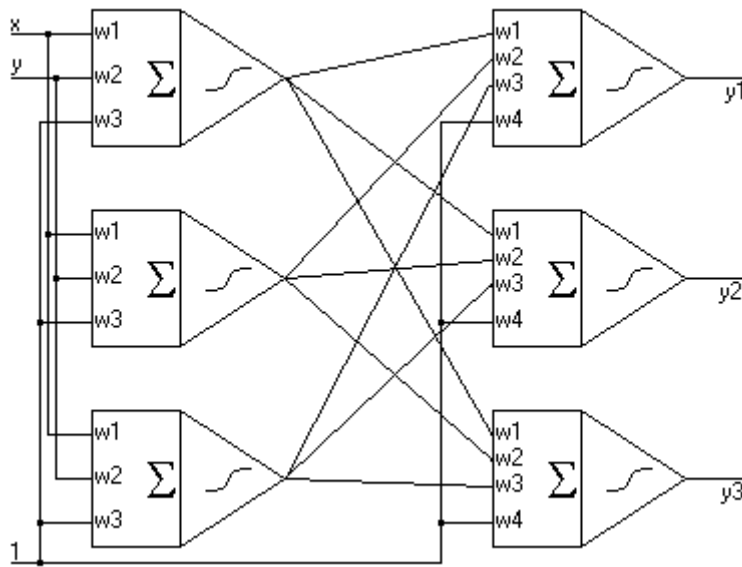
Funkcja sigmoidalna unipolarna – ma wartości  $\in [0.0 ; 1.0]$

Każda warstwa sieci 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.

Uwaga. Wpływ parametru beta może być zastąpiony odpowiednim doбором 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 wyniku SSN, neurony na wyjściu mają wartość z przedziału  $[0.0; 1.0]$ .

Numer neuronu o największej wartości, o ile ta wartość jest  $> 0.5$  oznacza numer znaku. Gdy wszystkie neurony mają wartości  $< 0.5$  (w tym i największa), to oznacza odpowiedź „nie rozpoznany”.

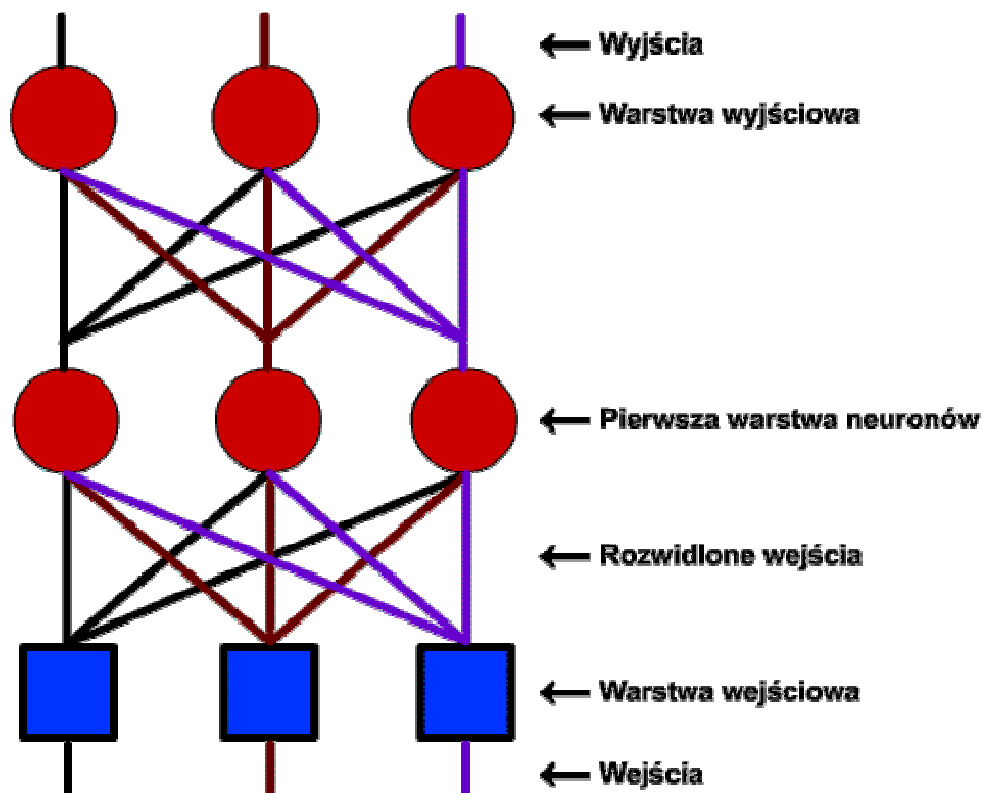
### UCZENIE SSN

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 (E). 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.

**Gradient** w analizie matematycznej – operator różniczkowy, który polu skalarnemu przyporządkowuje pole wektorowe. Owo pole wektorowe ma kierunek i zwrot wektora największego wzrostu funkcji w danym punkcie, a wartość jest proporcjonalna do szybkości wzrostu (wzrost na jednostkę długości) funkcji. Wektor przeciwny gradientowi nazywany jest **antygradientem**.

Gradient oznaczany jest przez grad lub symbolem nabla  $\nabla$ , który jest odwróconą grecką literą delta.

Intuicyjnie gradient jest wektorem, którego zwrot wskazuje kierunek najszybszego wzrostu wartości funkcji, natomiast długość odpowiada wzrostowi tej funkcji na jednostkę długości.

W układzie **współrzędnych kartezjańskich** wektor gradientu jest określony jako:

$$\text{grad } f = \nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right]$$

**Gradient F** - wektor utworzony z pochodnych funkcji F po wszystkich jej zmiennych.

Kierunek największego spadku dowolnej funkcji wskazuje ujemny gradient tej funkcji.

Dlatego wektor wag neuronu jest przesuwany w kierunku ujemnego gradientu. Dodatkowo wektor ten jest przemnażany przez **współczynnik eta**, który nazywany jest **współczynnikiem szybkości uczenia** (współczynnik korekcji).



Założmy, że mamy SSN złożoną z 3 warstw:

- wejściowa:  $\mathbf{U}$ :  $u[0] \div u[m-1]$ , to dane wejściowe, a  $u[m]$  – to stała wartość = 1.0 – tzw. bias

- jedna ukryta:  $\mathbf{X}$ :  $x[0] \div x[n-1]$  oraz stała  $x[n] = 1.0$  – tzw. bias

- wyjściowa:  $\mathbf{Y}$ :  $y[0] \div y[r-1]$

Rozmiary sieci:

$m$  – liczba danych wejściowych

$n$  – liczba neuronów w warstwie ukrytej

$r$  – liczba neuronów w warstwie wyjściowej

$\mathbf{U}, \mathbf{X}, \mathbf{Y}$  – wektory danych

Mamy zatem tutaj dwie warstwy wag:

- między warstwą wejściową i ukrytą:  $\mathbf{w}[i,j]$ ;  $i$  – nr neuronu warstwy ukrytej;  $j$  – nr danej wejściowej;

czyli  $W[i]$ :  $W[0] \div W[m]$

- między warstwą ukrytą i wyjściową:  $\mathbf{s}[i,j]$ ;  $i$  – nr neuronu warstwy wyjściowej;  $j$  – nr neuronu warstwy ukrytej; czyli

$S[i]$ :  $S[0] \div S[n]$

$\mathbf{W}[i], \mathbf{S}[i]$  – wektory wag  $w$  i  $s$

Mamy osobne funkcje aktywacji dla każdej warstwy sieci:

$f^{uk}(\mathbf{x}) = f(x, \text{BetaUk})$  – funkcja aktywacji w warstwie ukrytej

$f^{wy}(\mathbf{x}) = f(x, \text{BetaWy})$  – funkcja aktywacji w warstwie wyjściowej

Sygnały w SSN płyną od wejścia do wyjścia, czyli **wynik SSN** jest obliczany kolejno:

Najpierw:

$x[i] = f^{uk}(U * W[i]) = f^{uk}(u[0]*w[i,0] + u[1]*w[i,1] + \dots + u[m-1]*w[i,m-1] + 1*w[i,m])$

( $x[n] = 1$ )

czyli wektor  $\mathbf{X} = f^{uk}(U * W)$

Potem:

$y[i] = f^{wy}(X * S[i]) = f^{wy}(x[0]*s[i,0] + x[1]*s[i,1] + \dots + x[n-1]*s[i,n-1] + 1*s[i,n])$

czyli wektor  $\mathbf{Y} = f^{wy}(X * S)$

Niech  $\mathbf{P}$  – zbiór wszystkich wzorców;  $\mathbf{z}^{(p)}$  – **oczekiwany wynik** dla danego wzorca  $p \in \mathbf{P}$

Dla ustalonego  $p$  niech  $\mathbf{z} = \mathbf{z}^{(p)}$ ,  $\mathbf{x} = \mathbf{x}^{(p)}$ ,  $\mathbf{y} = \mathbf{y}^{(p)}$ .

**Nasz cel:** Dla każdego wzorca  $p \in \mathbf{P}$  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

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

Chcemy znaleźć minimum funkcji  $F$ , czyli odpowiednich wartości zmiennych  $S[i]$ .

Używamy **metody gradientu**:

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

Oznaczmy potencjał wejściowy neuronu warstwy wyjściowej i jako **PotWej<sup>wy</sup>[i]** =  $X * S[i]$

Zatem:

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

$$= 2 * (f^{wy}(X * S[0]) - z[0]) * f^{wy}'(X * S[0]) * X = (\text{bo dla } k \neq 0 \text{ } \partial f^{wy}(X * S[k] - z[k]) / \partial S[0] = 0)$$

$$= 2 * (y[0] - z[0]) * f^{wy}'(X * S[0]) * X$$

$$= 2 * (y[0] - z[0]) * f^{wy}'(\text{PotWej}^{wy}[0]) * X.$$

$f^{wy}'(\mathbf{x})$  – pochodna funkcji  $f^{wy}(x)$

Zatem w celu minimalizacji F zmieniamy argumenty z szybkością eta:

$S := S - \text{eta} * \text{Gradient } F$ , czyli

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

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

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

W celu przyspieszenia obliczeń, można dla danego neuronu k obliczyć na początku stałą wartość (mała delta):

$\delta(k) = 2 * (z[k] - y[k]) * f^{\text{Wy}'}(\text{PotWej}^{\text{Wy}}[k])$ , jest to **wartość błędu neuronu k**, wtedy

$$s[k,i] := s[k,i] + \text{eta} * \delta(k) * x[i]$$

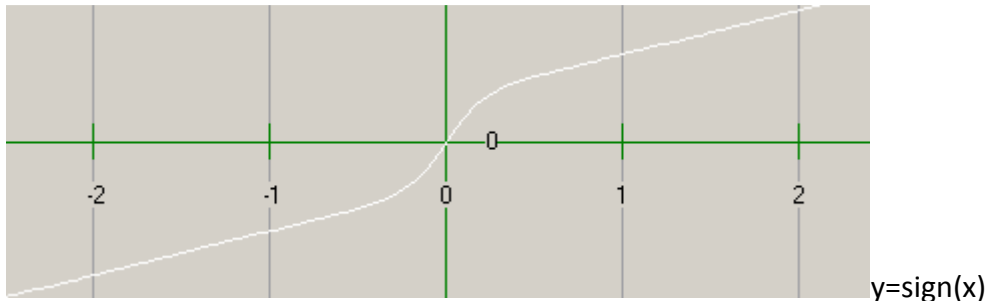
Zatem aktualizacja wag neuronów odbywa się zgodnie ze wzorem (to wersja bez członu momentum):

$$s[k,i] = s[k,i] + \Delta s[k,i],$$

gdzie  $\Delta s[k,i] = \text{eta} * \delta(k) * x[i]$  – dla warstwy wyjściowej

Aby zmiany były nieco łagodniejsze można  $\Delta s[k,i]$  nieco zmienić tzn. sugerować się bardziej **znakiem** gradientu niż jego wartością (podobnie jak w metodzie **Resilient BackPropagation RPROP**) np.

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



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

Okazuje się, że dla dowolnej warstwy (L)  $\delta(k)$  = sumie ważonej  $\delta$  (delt) z warstwy następnej (L+1) razy pochodna funkcji aktywacji, tzn.:

W naszym przypadku dla warstwy ukrytej i wyjściowej:

$$\delta^{\text{Uk}}(i) = \sum_{j=0}^{r-1} (\delta^{\text{Wy}}(j) * s[j,i]) * f^{\text{Uk}'}(\text{PotWej}^{\text{Uk}}[i]); \delta^{\text{Uk}}(0) \div \delta^{\text{Uk}}(n-1)$$

Ogólnie dla warstw L i L+1:

$$\delta^{(L)}(i) = \sum_{j=0}^{r^{(L+1)}-1} (\delta^{(L+1)}(j) * w^{(L+1)}[j,i]) * f^{(L)'}(\text{PotWej}^{(L)}[i])$$

$r^{(L+1)}$  – liczba neuronów w warstwie L+1

$w^{(L+1)}[j,i]$  – 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; //suma ważona d z warstwy następcnej
 for (j=0; j < LNeuronowWy; j++) E += dWy[j] * WagaWy[j][i];
 dUk2[i] = FunP(PotWejUk[i], BetaUk) * E; //d(i) i-ty neuron
}
```

### Kod C++ poprawy wag dla dwóch warstw:

```
for (i=0; i<LNeuronowUk2; ++i) //po wsz. neuronach z warstwy Uk2; licz d(i) dla NeuronUk2
{
 E = 0.0; for (j=0; j<LNeuronowWy; ++j) E += DaneWy[j] * WagaWy[j][i]; //suma ważona d[] z warstwy następcnej
 DaneUk2[i] = FunP(PotWejUk2[i], BetaUk2) * E; //d(i) i-ty neuron

 //--Korygowanie wag neuronu [i] warstwy Uk2 (dopiero teraz można!, bo WagaUk2[k][i] jest wcześniej czytana)
 for (j=0; j<LNeuronowUk1; ++j) //po wsz. wagach neuronu z Uk2
 {delta = etaUk2 * sign(DaneUk2[i] * DaneUk1[j]) + alfaUk2 * WagaUk2S[i][j];
  WagaUk2[i][j] += delta; //zmiana wag
 }
 delta = etaUk2 * sign(DaneUk2[i]) + alfaUk2 * WagaUk2S[i][LNeuronowUk1];
 WagaUk2[i][LNeuronowUk1] += delta; //oraz Bias <małe znaczenie>
}
```

### 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 poprawiaj wagi (i sprawdzaj błąd dla tego wzorca) **nie więcej** niż **MaxLPoprawWCyklad** razy lub gdy nie ma już błędu dla tego wzorca (tzw. **cykl**)
- Sprawdź następcny wzorzec.

Powtarzaj tę pętlę, aż nie będzie błędu większego niż **'epsilon'** dla każdego wzorca (w całej epoce) oraz suma tych błędów nie będzie większa od parametru **'Suma epsilon'**.

**MaxLPoprawWCyklad** - zmienia się i zależy od liczby wzorców i bieżącej liczby błędów, tzn.

$\text{MaxLPoprawWCyklad} = \text{Liczba wszystkich Wzorców} / (\text{Liczba Błędów} + 1) * \text{ParamCykl}$ ; gdzie ParamCykl – można ustawić w programie, na np. wartość = 0,7.

Zatem tutaj w cyklu na początku całej nauki, gdy jest dużo błędów, poprawiamy wagi tylko raz lub małą liczbę razy. Potem dopiero, wraz z malejącą liczbą błędów, liczba popraw wag w cyklu zwiększa się.

Taki sposób uczenia okazuje się w praktyce **szybszy** w całym procesie nauki, niż stosowanie za każdym razem pełnych cykli, czyli poprawiania wag, aż nie będzie błędu dla danego wzorca.

**Błąd sieci** to suma po wszystkich neuronach wyjściowych kwadratów różnicy między obliczonym wynikiem sieci **y[]** (dla podanych danych wejściowych – wzorzec p) a oczekiwanym wynikiem **z[]** (dla tych danych) czyli:

$$E = \sum_{k=0}^{r-1} (y[k] - z[k])^2 \text{ Wymagamy aby błąd sieci } E < \text{epsilon.}$$

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  $(0.4 - 0.0)^2 + (0.9 - 1.0)^2 = 0.4^2 + 0.1^2 = 0.16 + 0.01 = 0.17$ .

## 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 znaków, których SSN ma się uczyć i na ich podstawie rozpoznawać inne obrazy. Dodatkowo jako wzorce dla odpowiedzi „nie rozpoznano” – generowane są losowe obrazy (tzn. losowe kolory pikseli) oraz pewne szablonowe np. cały czarny – wszystkie wartości pikseli równe 255 lub cały biały – wszystkie wartości równe 0.

**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 osiągnięcia 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 krokami, 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 = 10, AlfaUk1 = 0.8,

EtaUk2 = 0.2, A\_EtaUk2 = 8, AlfaUk2 = 0.8,

EtaWy = 0.1, A\_EtaWy = 6, AlfaWy = 0.8,

**A\_Eta** – określa jak bardzo ma maleć wartość Eta w zależności od liczby błędów w ostatniej epoce.

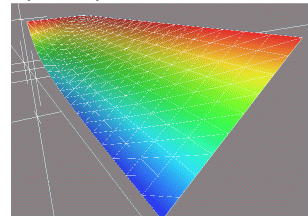
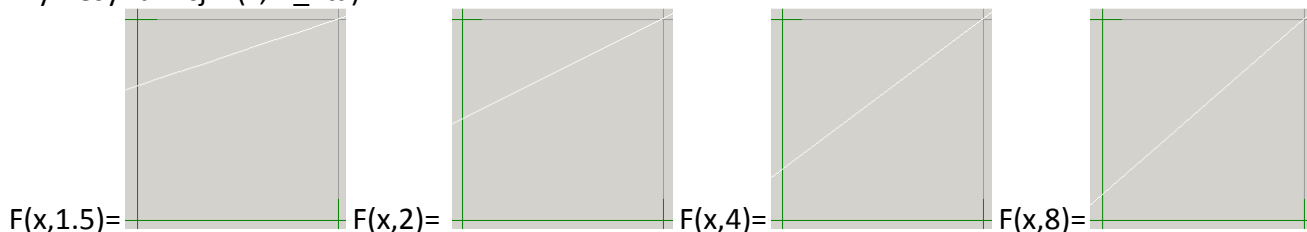
Im większa wartość A\_Eta tym eta bardziej maleje, proporcjonalnie do liczby błędów.

A\_Eta musi być  $\geq 1.0$ . Gdy A\_Eta = 1.0, Eta nie zmienia się.

Dokładnie obliczanie końcowego parametru eta wygląda następująco:

$$\text{eta} = \text{Eta} * F(\text{LBłędów}/\text{Wzorców}, A\_Eta), \text{ gdzie } F(x, n) = \frac{1 + (n - 1) * x}{n}$$

Wykresy funkcji F(x, A\_Eta)



$$z = F(x, n); 0 \leq x \leq 1; 1 \leq n \leq 6$$

**Alfa** – parametr określający wpływ **członu meomentum** 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, np. 0.88.

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**.

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 nauczania.

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 (, bo np. wpadła 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:  $rUk1$ ,  $rUk2$ ,  $rWy$ ,  $mUk1$ ,  $mUk2$ ,  $mWy$ ). Okazuje się, że najlepiej jest są z przedziału  $[-0.5; 0.5]$ .
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.
8. Odpowiednio dobrana maksymalna liczba popraw wag w cyklu – rośnie proporcjonalnie do liczby błędów w ostatniej epoce. Dodatkowy parametr określa tę proporcję (Parametr dla cyklu popraw).

**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 błędów – MBP lub MEBP (ang. Momentum Error BackPropagation).

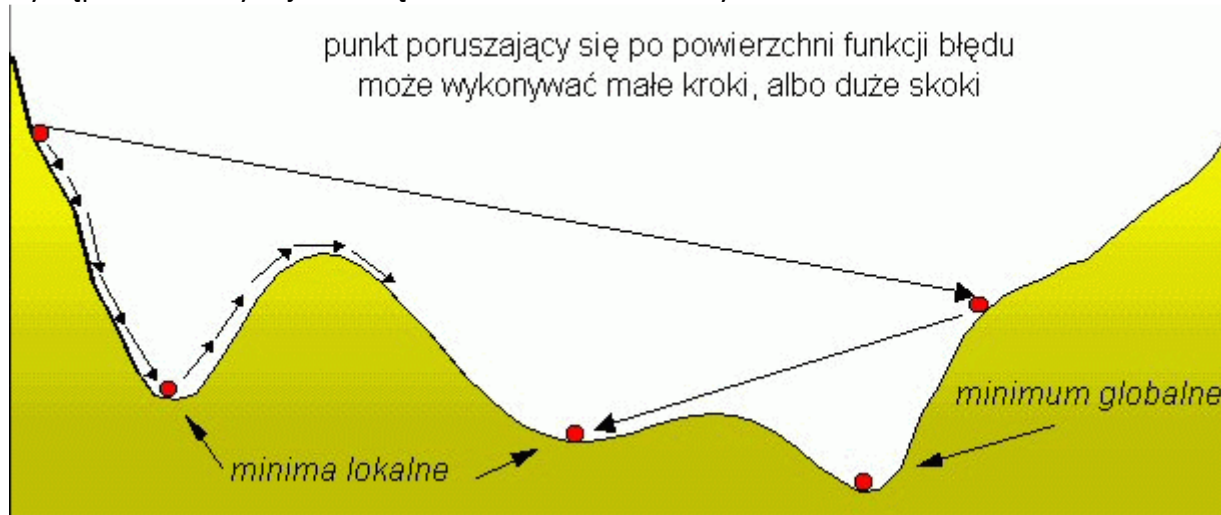
Istotą metody jest wprowadzenie do procesu uaktualniania wagi pewnej **bezwładności** tzw. "momentu", proporcjonalnego do zmiany tej wagi w poprzedniej iteracji. Pojawienie się gradientu wymuszającego ruch w kierunku innym, niż aktualny nie od razu wpływa na zmianę trajektorii w przestrzeni wag. W przypadku, 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 błędów, dla których zwykle obserwuje się zmniejszenie szybkości uczenia. Dla płaskiego obszaru hiperpłaszczyzny funkcji miary błędów 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 filtra 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

- SSN nauczyła się, tzn. osiągnęła wymagane parametry, tzn. wartość **epsilon** i **Suma epsilon**. W tym wypadku program podaje liczbę przeprowadzonych kroków nauki.
- SSN nie nauczyła się a osiągnięto już maksymalną liczbę iteracji (kroków nauki: popraw wag lub liczby epok). W tym wypadku 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. Takiego testu nie przeprowadzam, gdyż uczę SSN wszystkich dostępnych wzorców (ponad 1000).

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.

## IMPLEMENTACJA - KODY ŹRÓDŁOWE w skrócie:

Aplikacja została napisana w środowisku Borland C++ Builder 6.0 w sposób obiektowy.

Dlaczego w języku C++:

Uczenie SSN jest procesem wymagającym bardzo dużej liczby obliczeń a język C++ zapewnia najszybszy kod wynikowy z języków wysokiego poziomu

\* Wiele wspólnych funkcji operujących na obrazie zostało zebranych w jednym pliku: **UnitFunkcjeObraz.h/cpp**

\* Funkcja szukająca ramki ciągu znaków to:

**SzukajRamkiR**(unsigned char \*Obraz, int RozX, int RozY, unsigned char Prog, int Roznica, int MinX, int MinY, TRamka &Ramka, char \*&info)

\* Po znalezieniu ramki ciągu znaków, szukamy w niej ramek poszczególnych znaków, funkcją:

**SzukajZnakOdLewejR**(unsigned char \*Obraz, int RozX, int RozY, //wsk. na obraz; jego rozmiary  
TRamka RamkaO, //ramka obrazu  
unsigned char ProgB, //próg bezwzględny jasności  
int Roznica, //próg różnicy max-min na całej wysokości ramki  
TRamka &RamkaZ, char \*&info) //wynik

\* Po wyznaczeniu ramki danego znaku jej obraz podawany jest do funkcji:

**PobierzObrazDoSSN**(unsigned char \*ObrazZ, int ObrazRozX, int ObrazRozY, unsigned char \*ObrazBuf, TRamka Ramka, unsigned char \*ObrazZnak, int ZnakRozX, int ZnakRozY, char \*&info)

\*\* Plik **KlasaBazaZnakow.h/cpp** zawiera definicję klasy odpowiedzialnej za wczytanie obrazów znaków z przygotowanego wcześniej pliku bazy.

Baza obrazów zapamiętywana jest w obiekcie klasy **KlasaBazaZnakow**.

\*\* **Oto ważniejsze elementy programu do uczenia SSN:**

W programie uczącym SSN zdefiniowano w pliku **KlasaSiecNmom2H.h/cpp** klasę **SIECN2H**, która tworzy, uczy SSN 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 rozmiary w liczbach neuronów kolejnych warstw SSN.

\* Po utworzeniu SSN należy ją zresetować, tzn. ustalić współczynniki Beta dla każdej z warstw oraz ustalić wartości wag na losowe. Zaczynając od **losowo** ustalonych wartości wag, SSN uczy się najszybciej!

Dokonyjemy tego metodą:

```
Siec.Resetuj(BetaUk1, BetaUk2, BetaWy, PoczBiasUk1, PoczBiasUk2, PoczBiasWy,  
             rUk1,rUk2,rWy, mUk1,mUk2,mWy)
```

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:

```
Siec.LearnAll(LWzorcow, FunPX, FunPZ, EtaUk1, A_EtaUk1, AlfaUk1, EtaUk2, A_EtaUk2, AlfaUk2, EtaWy, A_EtaWy,  
AlfaWy, mC, MinR, MaxG, zaburzenie, eps, Sum_eps, MaxLPopraw, MaxLEpok, void PokazOpis(char *info))
```

Kolejne parametry, to:

- LWzorcow – liczba wzorców do nauczenia

- FunPX, FunPZ – funkcje przygotowujące wartości wejściowe i wyjściowe funkcji

- EtaUk1, A\_EtaUk1, AlfaUk1, EtaUk2, A\_EtaUk2, AlfaUk2, EtaWy, A\_EtaWy, AlfaWy – wektor nauki i sposób jego dynamicznej modyfikacji

- mC – (dla eksperymentu) zmienia wektor uczenia (tu ustalono na = 0.0 – nic nie zmienia)

- 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 episolon – warunki, jakie ma spełnić nauczona sieć

- MaxLPopraw, MaxLEpok – kryterium zatrzymania procesu uczenia (aby przerwać naukę, gdy nie chce się nauczyć)

- PokazOpis(char \*info) – funkcja która będzie wywoływana po każdej epoce; parametr info będzie zawierał łańcuch opisujący aktualny stan nauki SSN

### Losowanie wartości wag

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:

```
PoprawWagi(const double *DaneOcz, double etaUk1, double alfaUk1, double etaUk2, double alfaUk2, double etaWy, double alfaWy, double mC)
```

Jest to najważniejsza funkcja (jądro) 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 rozpoznawania** obrazów (bez nauki) użyto nieco zmienionej klasy SIECN2H, tzn. bez metod: LearnAll i służącej do zapisywania parametrów sieci.

Deklaracja i definicja tej klasy są zawarte w plikach **KlasaSiecNmom2HRozp.h/cpp**.

\*\*

Samo znalezienie ramki ciągu znaków nie jest takie proste i nie wystarczy zwykłe rozróżnianie pikseli ciemnych od jasnych, gdyż obraz znaków może być na ciemnym tle lub znaki mogą być wyblakłe, rozjaśnione. Jest jeszcze wiele innych pułapek. Dlatego tak dużo jest użytych różnych funkcji i każda z nich jest rozbudowana. Jednak też nie są one w stanie rozpoznać wszystkich możliwych przypadków. Wziąłem pod uwagę tylko te najczęściej występujące.

\*\*

Wszystkie kody źródłowe zostały napisane od podstaw. Oprócz tej dokumentacji wiele opisów i wyjaśnień jest w samych komentarzach w kodach źródłowych.

## PROBLEMY

1. Nieraz pojedynczy znak zostaje potraktowany jak dwa znaki, np. dla liter: i, j; ponieważ kropka może być narysowana poza prawą krawędzią ramki znaku (bez kropki).
2. SSN nieraz ma trudności w nauczaniu się przez wzorce znaku średnika, który po przeskalowaniu jest podobny do małej litery i. Wtedy najlepiej usunąć taki znak średnika z bazy znaków, gdyż lepiej żeby SSN dobrze rozpoznawała znaki liter niż znaków specjalnych.

## Szybkość nauczania SSN - przykłady:

- Uczenie z cyklami o jednokrotnej poprawie wag:

\* 634 wzorców znaków oraz 15 nie-znaków, Epsilon = **0.28**

- 389 epok; 22650 popraw; Czas: 470 s

\*646 wzorców znaków + 12 nie-znaków, Epsilon = **0.28**

-275 epok; 20477 popraw; Czas: 305 s

- Uczenie z cyklami o wielokrotnych poprawach wag:

\* 1142 wzorców znaków; rozmiar SSN: 256- 160-110- 90

81 epok; 38918 popraw wag; Czas:6,63 min; Epsilon=**0.26**

Autor: Artur Czekalski; [ARTUR@epokaY.net](mailto:ARTUR@epokaY.net); [www.epokay.net/artur](http://www.epokay.net/artur); Początek: 2005-04; Wersja: 2010-03