**Maciej Oziębły**

**Nr indeksu: 184147**

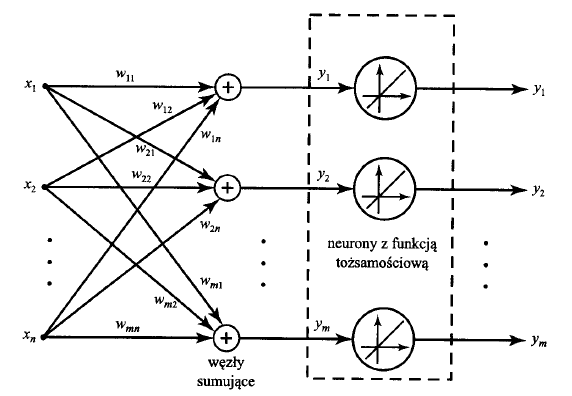
Programowanie sieciowe

Sprawozdanie 2

Zakres sprawozdania:

* asocjacyjne sieci neuronowe
* rozpoznawanie małych i wielkich liter

## Stworzenie sieci neuronowej

W celu stworzenia liniowej sieci neuronowej do do rozpoznawania znaków wykorzystałem następujący schemat:  


Opierając się na schemacie i teorii z zajęć stworzyłem sieć za pomocą kodu w matlabie („association\_recogniser.m” i „calc\_weights\_matrix.m”):

function [ recognised\_letter] = association\_recogniser( letter )

% get output vector

y = calc\_weights\_matrix(load\_letters\_definitions) \* letter;

% find and return best match (bigest val index in discrimination vector)

recognised\_letter = find(ismember(y, max(y)));

end

function [ weights\_matrix] = calc\_weights\_matrix( letters )

% F matrix is eye matrix for now coz i have

% just 1 pattern for each letter

F = eye(35);

% addding small letters (from 36 to 70)

% there is exactly 1 small letter for each capital letter

F = [F;eye(35)];

weights\_matrix = F' \* letters';

end

Ponieważ algorytm ten liczył macierze wag za każdym razem, kiedy chciał rozpoznać literę postanowiłem napisać też przyspieszony algorytm, w którym macierz wag podajemy z zewnątrz (znacznie przyspieszyło to program przy rozpoznawaniu dużej ilości znaków):

function [ recognised\_letter] = association\_recogniser\_optimized( letter , weights\_matrix)

% get output vector

y = weights\_matrix \* letter;

% find and return best match

recognised\_letter = find(ismember(y, max(y)));

end

Podając na wejściu funkcji literę w formie wektora 100 wartości dostajemy na wyjściu numer litery z bezy danych. Jako bazę danych wykorzystałem funkcję „load\_letters\_definitions.m”, w której znajdują się wektory zawierający wygląd wszystkich kolejnych liter

## Testy:

W celu przetestowania działania sieci stworzyłem skrypt „testerka.m”:

letters = load\_letters\_definitions();

letters = load\_letters\_definitions();

tries\_per\_letter = 300;

avg\_tries\_per\_letter = zeros(size(letters,2),1);

weights = calc\_weights\_matrix(load\_letters\_definitions);

for letter\_no = 1:size(letters,2)

tries = 0;

letter\_no

for i = 1:tries\_per\_letter

letter = letters(:, letter\_no);

random\_changes\_before\_fail = 0;

match\_failed = 0;

changed\_indexes = zeros(100,1);

while(match\_failed == 0 && random\_changes\_before\_fail ~= 100)

%change 36 -> 1, 2 -> 2, 70 -> 35 etc...

if(association\_recogniser\_optimized(letter, weights) ~= mod((letter\_no-1), 35)+1)

%if(association\_recogniser(letter) ~= mod((letter\_no-1), 35)+1)

match\_failed = 1;

else

% generate random index not changed yet

rand\_index = randi(100);

while(max(ismember(changed\_indexes, rand\_index)) ~= 0)

rand\_index = randi(100);

end

changed\_indexes(length(changed\_indexes)+1) = rand\_index;

letter(rand\_index) = letter(rand\_index) \* -1;

random\_changes\_before\_fail = random\_changes\_before\_fail + 1;

end

end

tries = tries + random\_changes\_before\_fail;

end

avg\_tries\_per\_letter(letter\_no) = tries / tries\_per\_letter;

end

stem(avg\_tries\_per\_letter)

title('average random changes on picture needed for recogniser to match incorrectly')

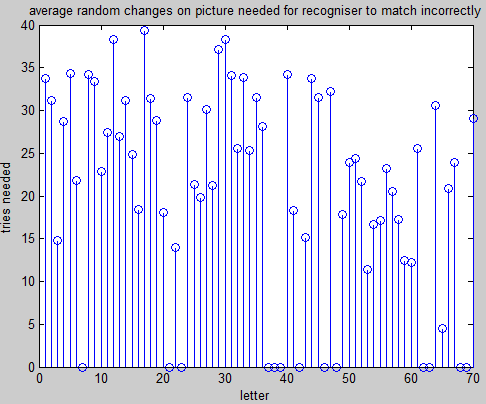
xlabel('letter')

ylabel('tries needed')

Skrypt ten wrzuca po kolei każdą z liter do sieci rozpoznającej znaki, a następnie zmienia 1 losowy piksel (nie zmieniony wcześniej) na jego przeciwieństwo. Zmiany trwają tak długo, aż sieć popełni błąd w rozpoznawianiu znaku. Dla każdej litery wykonane jest 300 prób, a następnie jest wyciągany z nich średnia ilość zmian potrzebnych do wystąpienia błędu.

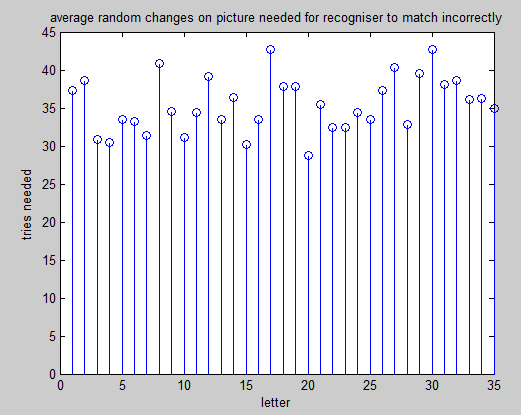
## Wyniki testów:

Używając swojego skryptu testującego wygenerowałem wykres średniej ilości losowych zmian potrzebnych, do wystąpienia błędu w sieci rozpoznającej znaki (znakli dla indeksów >35 to małe litery odpowiadające wielkim literom o indeksie mniejszym o 35 od nich).



Liczba zmian potrzebnych do wywołania błędu mieści się w zakresie [0, 40]. Algorytm nie radzi sobie z rozpoznawaniem niektórych znaków, co wynika z tego, że niektóre znaki są do siebie bardzo podobne.

Dla porównania przeprowadziłem testy bez małych liter:



Tutaj wyniki są o wiele lepsze – warto jednak zauważyć, że dla takich danych wejściowych sieć ta działa niemal identycznie jak liniowa sieć neuronowa z poprzedniego zadania.

## Wnioski:

Asocjacyjna sieć neuronowa radzi sobie bardzo dobrze z rozpoznawaniem wielkich liter, jednak po dodaniu do bazy wiedzy małych liter algorytm zaczął działać o wiele gorzej i często popełniać błędy – w wielu przypadkach nie rozpoznawał poprawnie litery, która była wzorcem. Działo się tak w przypadkach, kiedy mała litera bardzo różniła się od wielkiej litery, ponieważ w takim przypadku macierz wag dla tej asocjacji była bardzo zaszumiona. W przypadku posiadania jeszcze większej ilości wzorców dla każdej asocjacji własność ta pogłębiłaby się w znacznym stopniu przez co nie uważam sieci tego typu za odpowiednie do stosowania w praktyce.