Michał Radziejowski – Scenariusz 3 SPRAWOZDANIE

Budowa i działanie sieci wielowarstwowej.

Cel:

Celem ćwiczenia jest poznanie budowy i działania wielowarstwowych sieci neuronowych poprzez uczenie z użyciem algorytmu wstecznej propagacji błędu rozpoznawania konkretnych liter alfabetu.

Wykonanie zadania:

1. Wygenerowano dane uczące i testujące, zawierające 20 wielkich liter alfabetu łacińskiego w postaci dwuwymiarowej tablicy 7x5
2. Wykorzystano narzędzie PyBrain3 do stworzenia sieci neuronowej (FeedForward Network) wraz z modyfikacją warstwy uczącej.
3. Uczono sieci modyfikując współczynnik uczenia oraz współczynnik bezwładnosci.
4. Testowano poprawność działania sieci, sprawdzając zmianę błędu z każdą epoką, oraz próby rozpoznania liter ze szczególnym zwróceniem uwagi na podobieństwo między literami.

Algorytm uczenia sieci – algorytm propagacji wstecznej

Algorytm wstecznej propagacji błędów, polega na takim dobraniu wag sygnałów wejściowych każdego neuronu w każdej warstwie, aby wartość błędu dla kolejnych par uczących zawartych w zbiorze uczącym była jak najmniejsza. Schemat krokowy algorytmu przedstawia się w następujący sposób:

1. Ustalamy topologię sieci, tzn. liczbę warstw, liczbę neuronów w warstwach.
2. Inicjujemy wagi losowo.
3. Dla danego wektora uczącego obliczamy odpowiedź sieci (warstwa po warstwie).
4. Każdy neuron wyjściowy oblicza swój błąd, oparty na różnicy pomiędzy obliczoną odpowiedzią *y* oraz poprawną odpowiedzią *t*.
5. Błędy propagowane są do wcześniejszych warstw.
6. Każdy neuron modyfikuje wagi na podstawie wartości błędu i wielkości przetwarzanych w tym kroku sygnałów.
7. Powtarzamy od punktu 3. dla kolejnych wektorów uczących.
8. Zatrzymujemy się, gdy średni błąd na danych treningowych zostanie przyjęty jako optymalny.

Sieć została zbudowana według następującego schematu:

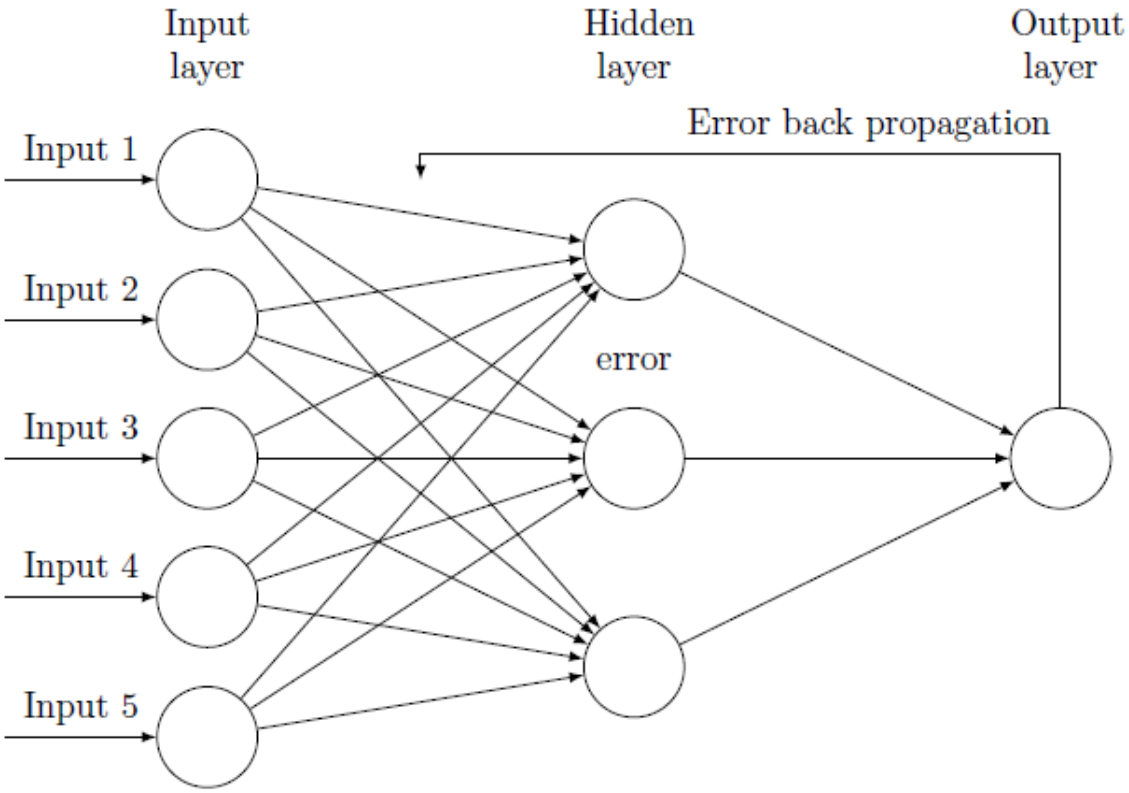
- 35 neuronów warstwy wejścia, funkcja liniowa;

- 20 neuronów na warstwie ukrytej (+ bias), funkcja sigmoidalna;

- 20 neuronów na warstwie wyjścia, funkcja liniowa.

W scenariuszu 3, oprócz zmiany wartości współczynnika nauki, skupiamy się również na wartości współczynnika bezwładności, znany także jako momentum. Kiedy stosowane jest momentum, korekta wag neuronu zależy nie tylko od sygnału wejściowego i błędu, jaki neuron popełnił, ale również od tego, jak duża była korekta wag w poprzednim kroku uczenia. W ten sposób szybkość uczenia (wielkość korekty wag) automatycznie maleje w miarę zbliżania się do właściwego rozwiązania, a proces uczenia staje się płynniejszy. Używając momentum można zwiększyć współczynnik uczenia.

Schemat sieci neuronowej typu FeedForward, z zaznaczonym kierunkiem algorytmu propagacji wstecznej błędu



Przykłady działania programu

Program po jego wywołaniu, wypisuje ilość danych treningowych z datasetu, a następnie przystępuje do treningu sieci według algorytmu propagacji wstecznej. Z każdą epoką, wypisuje na ekran całkowity błąd uczenia, dzięki któremu jesteśmy w stanie na bieżąco monitorować zmianę sieci.

```
Number of training patterns:  20
Total error: 1.29738480613
Total error: 0.263195108956
Total error: 0.150091393868
Total error: 0.113375078835
Total error: 0.0799468397426
```

Po wykonaniu treningu sieci na zadanej wcześniej stałej ilości epok;

```
trainer.trainEpochs(5000)
```

Program przechodzi do etapu testowania sieci na danych testowych.

```
for i in range(20):
    print("Dla litery",letters[i],"output wynosi:")
    temp = network.activate(inp[i])
    for j in range(20):
        print(temp[j])
```

Powyższy kod zapewnia dwadzieścia wyników dla jednej konkretnej litery – porównuje ją ze wszystkimi wcześniej wyuczonymi schematami i ukazuje jak bardzo jest zbliżona do danego wyniku.

```
Dla litery A output wynosi:
1.00749088306
-0.0148850466052
0.00225710863208
-0.000103532224795
-0.00400773891442
-0.00382388040004
0.000740170068276
-1.83828617944e-05
-0.000169508555836
0.00130511574202
-0.0004438173128
-0.00918672913383
0.0120465189066
0.00414506800364
0.00402552439295
-0.00258829948767
1.81991628738e-05
-0.00428515472566
0.000366486581251
-0.000209790146602
```

Przykładowo dla litery A – sieć poprawnie przypisała tej literze wartość 1 na pierwszym miejscu gdyż w tablicy datasetu właśnie to miejsce opisuje tą daną.

Podobne wyniki ukazują się dla każdej z liter przy każdorazowym włączeniu programu. Aby dokładnie zobaczyć podobieństwa i różnice pomiędzy trenowaniem sieci przy zmianie współczynników nauki oraz bezwładności, został przygotowany specjalny plik z wynikami – PSI3Wyniki.xlsx. W pliku tym, znajduje się 21 arkuszy:

- pierwszy arkusz poświęcony błędowi treningu epoki, w zależności od zadanych współczynników. Każdy zestaw danych został mierzony trzykrotnie, a następnie została wyliczana średnia arytmetyczna na podstawi której powstał wykres, ukazujący wartości błędów i ich zmianę w trakcie trwania nauki.

- kolejne 20 arkuszy zostało przypisanych po jednym dla każdej litery, aby ukazać w jaki sposób sieć radzi sobie z rozpoznawaniem liter, czy robi to w dobry sposób, oraz czy przy

danych parametrach nauki bądź konkretnej literze tworzy się problem w pozytywnym odgadnięciu litery. Do wykresów zostały wybrane dane najlepiej rozpoznające literę, oraz te z największymi problemami(wysokim błędem na końcu epoki). Również wykonano po trzy pomiary.

Analiza i wnioski

W każdym pomiarze, liczba epok uczących sieć była stała i wynosiła 5000. Wartość współczynnika uczenia wynosiła 0.1 bądź 0.01, natomiast wartość współczynnika bezwładności 0.01 oraz 0.001. Funkcje oraz liczba neuronów w warstwach również przez cały okres przeprowadzania testów pozostały stałe.

Analizując wykres przedstawiony w arkuszu pierwszym, jesteśmy w stanie zaobserwować zmianę błędów nauki dla każdej epoki, dla następujących zestawień wartości współczynników:

1) L.rate = 0.1
   Momentum = 0.01 Kolor niebieski
2) L.rate = 0.01
   Momentum = 0.01 Kolor czerwony
3) L.rate = 0.01
   Momentum = 0.001 Kolor oliwkowo-zielony

Pierwszy zestaw danych osiąga bardzo dobry wynik do około 3700 epoki, mimo iż wartości błędów są rozproszone w porównaniu do wykresów z zestawami 2 oraz 3. Po przekroczeniu ok. 3900 epoki możemy zaobserwować dość spore wahnięcia w wartościach, w skrajnych przypadkach sięgające nawet tysięcznych części. Świadczyć to może o sporym przeuczeniu tej sieci, czyli wykorzystaniu zbyt wielkiej ilości epok do poprawnego treningu. Nie mniej jednak, błąd osiągnięty przez taki zestaw danych jest najniższy, co skutkowało najlepszym wynikiem rozpoznawalności liter.

Drugi oraz trzeci zestaw danych osiągnął podobne wartości błędów, nie mniej jednak pomimo wahnięć pomiaru pierwszego, zestawy zaprezentowały gorsze wyniki w rozpoznawaniu liter.

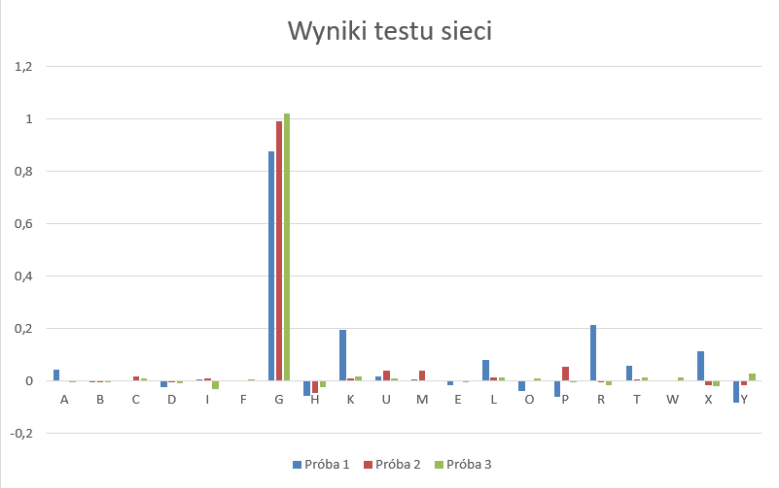
W kolejnych 20 arkuszach są zaobserwowane wyniki rozpoznawania liter, dla danych z zestawu pierwszego (błąd końcowy najmniejszy), oraz drugiego (błąd końcowy największy). Widzimy iż sieć korzystająca z wartości learn rate równej 0.1 oraz współczynnika bezwładności równego 0.01 nie pozostawia żadnych wątpliwości co do rozpoznanych liter i nie ma problemu z literami których przedstawienie w programie w pewnym stopniu może być podobne do litery testowanej.

Z kolei pomiary w którym learn rate został obniżony dziesięciokrotnie, do wartości 0.01 przedstawiają nam wykresy z których jasno wynika, iż pomimo poprawnego rozpoznania litery będącej przedmiotem testu, sieć nie była tak jednomyślna. Litery które w swoim

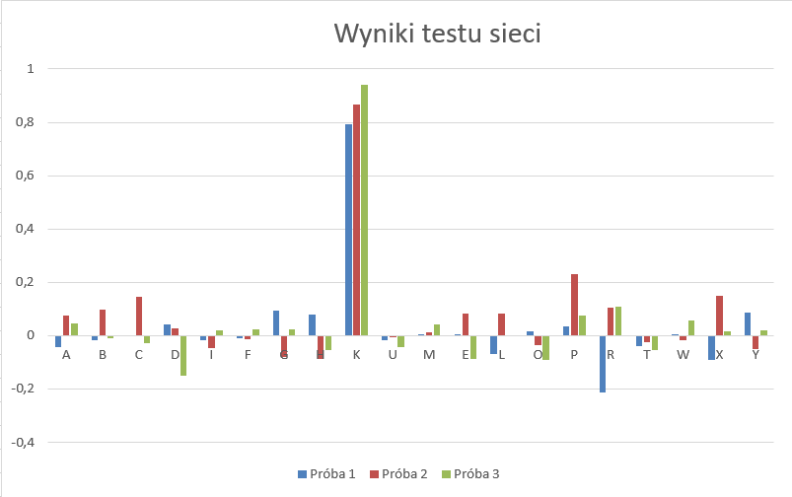
obrazie są podobne do litery testowanej, otrzymały stanowczo większe wyniki w porównaniu do pomiarów na danych wcześniejszych.

Kilka przykładów podobieństwa liter:

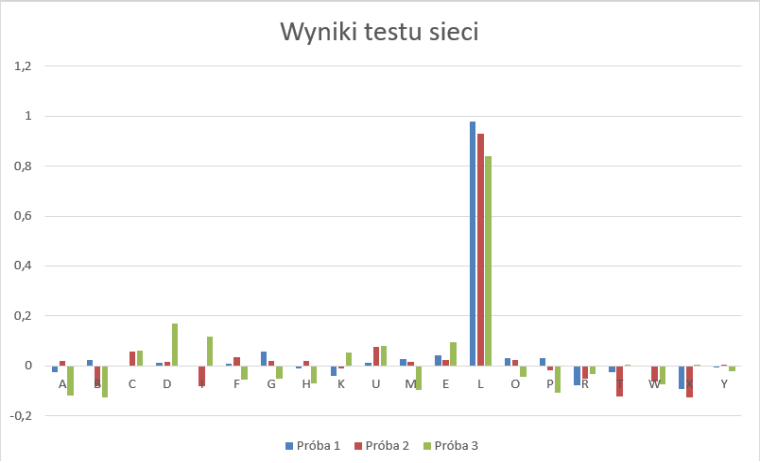
- Litera G oraz podobieństwo do liter K oraz R



- Litera K oraz litery P, R, C;



- Litera L oraz litery D, I, T;



Na podstawie wyników oraz wykresów, jesteśmy w stanie dowieść, iż sieć neuronowa z pierwszym zestawem danych została najlepiej nauczona algorytmem propagacji wstecznej błędów. Poprzez odpowiedni współczynnik bezwładności, mogliśmy w pewnym stopniu zwiększyć współczynnik nauki. Mimo chaotycznych wahnięć wartości błędów sugerujących przeuczenie sieci, pierwszy zestaw znamiennie przebił zestawy dwa i trzy. Świadczy to o zbyt małej ilości epok uczenia w tych zestawach, w stosunku do współczynnika uczenia, który został obniżony dziesięciokrotnie.

Listing kodu

Kod główny programu.

```
network = FeedForwardNetwork()          #Creating new network
letters = ["A", "B", "C", "D", "I", "F", "G", "H", "K", "U", "M", "E", "L",
"O", "P", "R", "T", "W", "X", "Y"]

inLayer = LinearLayer(35)            #Creating input layer
hiddenLayer = SigmoidLayer(20)        #Creating hidden layer
outLayer = LinearLayer(20)            #Creating output layer
bias = BiasUnit()                    #Initializing Bias

network.addInputModule(inLayer)          #Adding in/out and module layers to
the network
network.addModule(bias)
network.addModule(hiddenLayer)
network.addOutputModule(outLayer)

bias_to_hidden = FullConnection(bias, hiddenLayer)          #Creating
connection between layers
in_to_hidden = FullConnection(inLayer, hiddenLayer)
hidden_to_out = FullConnection(hiddenLayer, outLayer)

network.addConnection(bias_to_hidden)                       #Adding
connection to network
network.addConnection(in_to_hidden)
network.addConnection(hidden_to_out)

network.sortModules()                                       #Sorting
modules
inp = inputDataSet['input']                                 #Making
shortcut to the input section of DataSet

print ("Number of training patterns: ", len(inputDataSet))      #Printing
number of training patterns

trainer = BackpropTrainer(network, dataset=inputDataSet, learningrate=0.01,
verbose=True, momentum=0.01)    #Initializing trainer with Backpropagation
method

trainer.trainEpochs(10000)
#Training network for X epochs, verbose = true so printing errors for each
epoch
```

```
print("\n\n")
for i in range(20):
#Final print
    print("Dla litery",letters[i],"output wynosi:")
    temp = network.activate(inp[i])
    for j in range(20):
        print(temp[j])
    print("\n\n")


print("\n\n Koniec testów")
```

Implementacja zestawu danych uczących/testujących.

```
inputDataSet = SupervisedDataSet(35, 20)          #Creating new DataSet

#A
inputDataSet.addSample((                          #Adding first sample to
dataset
    -1, 1, 1, 1, -1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
    1, 1, 1, 1, 1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1
    ),
    (1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0))

#B
inputDataSet.addSample((
    1, 1, 1, 1, -1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
    1, 1, 1, 1, -1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
    1, 1, 1, 1, -1
    ),
    (0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))

#C
inputDataSet.addSample((
    -1, 1, 1, 1, -1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, -1,
    1, -1, -1, -1, -1,
    1, -1, -1, -1, -1,
    1, -1, -1, -1, 1,
    -1, 1, 1, 1, -1
    ),
    (0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))

#D
inputDataSet.addSample((
    1, 1, 1, 1, -1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
```

```
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
    1, 1, 1, 1, -1
    ),
    (0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))

#F
inputDataSet.addSample((
    1, 1, 1, 1, 1,
    1, -1, -1, -1, -1,
    1, -1, -1, -1, -1,
    1, 1, 1, 1, -1,
    1, -1, -1, -1, -1,
    1, -1, -1, -1, -1,
    1, -1, -1, -1, -1
    ),
    (0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))

#G
inputDataSet.addSample((
    -1, 1, 1, 1, -1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, -1,
    1, -1, 1, 1, 1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
    -1, 1, 1, 1, -1
    ),
    (0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))

#H
inputDataSet.addSample((
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
    1, 1, 1, 1, 1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1
    ),
    (0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))

#I
inputDataSet.addSample((
    -1, -1, 1, -1, -1,
    -1, -1, 1, -1, -1,
    -1, -1, 1, -1, -1,
    -1, -1, 1, -1, -1,
    -1, -1, 1, -1, -1,
    -1, -1, 1, -1, -1,
    -1, -1, 1, -1, -1
    ),
    (0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))

#K
inputDataSet.addSample((
    1, -1, -1, -1, 1,
    1, -1, -1, 1, -1,
    1, -1, 1, -1, -1,
    1, 1, -1, -1, -1,
    1, -1, 1, -1, -1,
```

```
    1, -1, -1, 1, -1,
    1, -1, -1, -1, 1
    ),
    (0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))

#U
inputDataSet.addSample((
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
    -1, 1, 1, 1, -1
    ),
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0))

#M
inputDataSet.addSample((
    1, -1, -1, -1, 1,
    1, 1, -1, 1, 1,
    1, -1, 1, -1, 1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1
    ),
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0))

#E
inputDataSet.addSample((
    1, 1, 1, 1, -1,
    1, -1, -1, -1, -1,
    1, -1, -1, -1, -1,
    1, 1, 1, -1, -1,
    1, -1, -1, -1, -1,
    1, -1, -1, -1, -1,
    1, 1, 1, 1, -1
    ),
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0))

#L
inputDataSet.addSample((
    1, -1, -1, -1, -1,
    1, -1, -1, -1, -1,
    1, -1, -1, -1, -1,
    1, -1, -1, -1, -1,
    1, -1, -1, -1, -1,
    1, -1, -1, -1, -1,
    1, 1, 1, 1, -1
    ),
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0))

#O
inputDataSet.addSample((
    1, 1, 1, 1, 1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
```

```
    1, 1, 1, 1, 1
    ),
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0))

#P
inputDataSet.addSample((
    1, 1, 1, -1, -1,
    1, -1, -1, 1, -1,
    1, -1, -1, 1, -1,
    1, 1, 1, -1, -1,
    1, -1, -1, -1, -1,
    1, -1, -1, -1, -1,
    1, -1, -1, -1, -1
    ),
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0))

#R
inputDataSet.addSample((
    1, 1, 1, -1, -1,
    1, -1, -1, 1, -1,
    1, -1, -1, 1, -1,
    1, 1, 1, -1, -1,
    1, -1, 1, -1, -1,
    1, -1, -1, 1, -1,
    1, -1, -1, -1, 1
    ),
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0))

#T
inputDataSet.addSample((
    1, 1, 1, 1, 1,
    -1, -1, 1, -1, -1,
    -1, -1, 1, -1, -1,
    -1, -1, 1, -1, -1,
    -1, -1, 1, -1, -1,
    -1, -1, 1, -1, -1,
    -1, -1, 1, -1, -1
    ),
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0))

#W
inputDataSet.addSample((
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
    1, -1, -1, -1, 1,
    1, -1, 1, -1, 1,
    1, 1, -1, 1, 1,
    1, -1, -1, -1, 1
    ),
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0))

#X
inputDataSet.addSample((
    -1, -1, -1, -1, -1,
    -1, -1, -1, -1, -1,
    1, -1, -1, -1, 1,
    -1, 1, -1, 1, -1,
    -1, -1, 1, -1, -1,
    -1, 1, -1, 1, -1,
    1, -1, -1, -1, 1
```

```
    ),
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0))

#Y
inputDataSet.addSample((
    -1, -1, -1, -1, -1,
    -1, -1, -1, -1, -1,
    1, -1, -1, -1, 1,
    -1, 1, -1, 1, -1,
    -1, -1, 1, -1, -1,
    -1, 1, -1, -1, -1,
    1, -1, -1, -1, -1
    ),
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1))
```

Źródła użyte przy tworzeniu sieci oraz sprawozdania:

Książka Stanisława Osowskiego – Sieci neuronowe do przetwarzania informacji

http://sknbo.ue.poznan.pl/neuro

http://edu.pjwstk.edu.pl

http://pybrain.org/docs/

https://stackoverflow.com/