Michał Radziejowski – Scenariusz 2 SPRAWOZDANIE

Budowa i działanie sieci neuronowej

Cel ćwiczenia

Celem ćwiczenia jest poznanie budowy i działania sieci neuronowych oraz uczenie rozpoznawania wielkości liter.

Wykonane zadania

1. Wygenerowano dane uczące i testujące, zawierające 10 wielkich i 10 małych liter alfabetu łacińskiego w postaci dwuwymiarowej tablicy 7x5, reprezentowanej w kodzie jako jednowymiarowa tablica 35 elementowa
2. Wykorzystano narzędzie PyBrain do stworzenia sieci neuronowej (FeedForward Network) wraz z modyfikacją warstwy uczącej.
3. Uczono sieci modyfikując współczynnik uczenia.
4. Testowano poprawność działania sieci.

Algorytm uczenia sieci – algorytm propagacji wstecznej

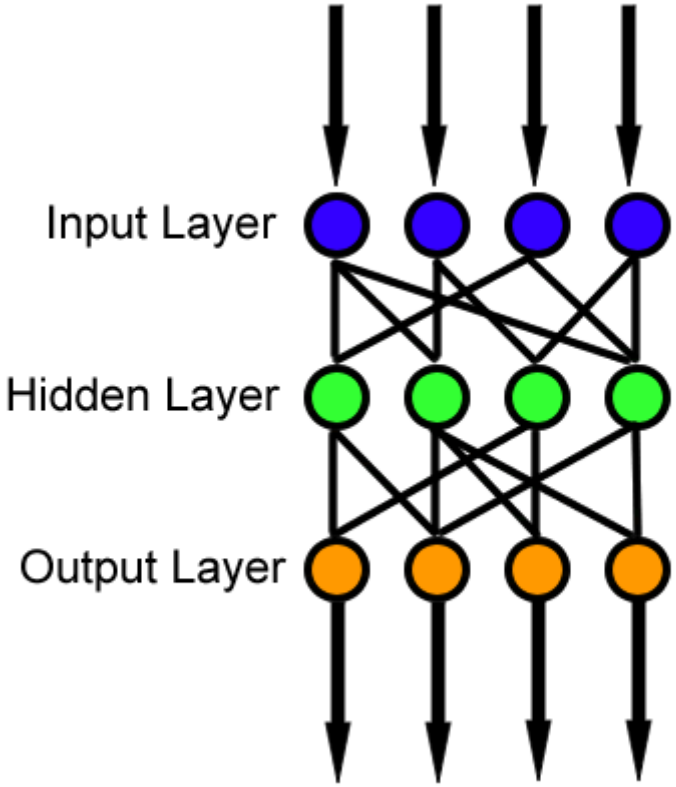
Algorytm wstecznej propagacji błędów, polega na takim dobraniu wag sygnałów wejściowych każdego neuronu w każdej warstwie, aby wartość błędu dla kolejnych par uczących zawartych w zbiorze uczącym była jak najmniejsza. Schemat krokowy algorytmu przedstawia się w następujący sposób:

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.

Algorytm Sieci

Do stworzenia sieci została użyta biblioteka PyBrain3. Podstawową architekturą była trójwarstwowa sieć typu FeedForward stworzona z 35 neuronów wejścia opisywanych funkcją liniową. Następnie znajduje się warstwa ukryta, której wielkość może być zmienna, opisywana funkcją sigmoidalną lub tangens hiperboliczny (podane poniżej), oraz warstwa wyjścia złożona z jednego neurona funkcji liniowej.

Ogólny schemat sieci typu FeedForward – informacja posuwa się tylko w jednym kierunku, naprzód, poprzez wszystkie warstwy. Nie posiada cyklów ani pętli.



Neuron sigmoidalny jest budową zbliżony do perceptronu, różni się on przede wszystkim funkcją aktywacji i zwracaną wartością. Wykorzystuje on ciągłą funkcję aktywacji.
W neuronie sigmoidalnym do obliczenia nowej wagi wykorzystywana jest pochodna funkcji aktywacji: $\Delta wij(k+1) = -\eta\delta ixj + \alpha\Delta wij(k)$
gdzie:
i - numer neutronu
j - numer wejścia
w - waga
η - learning rate
δ - pochodna funkcji aktywacji

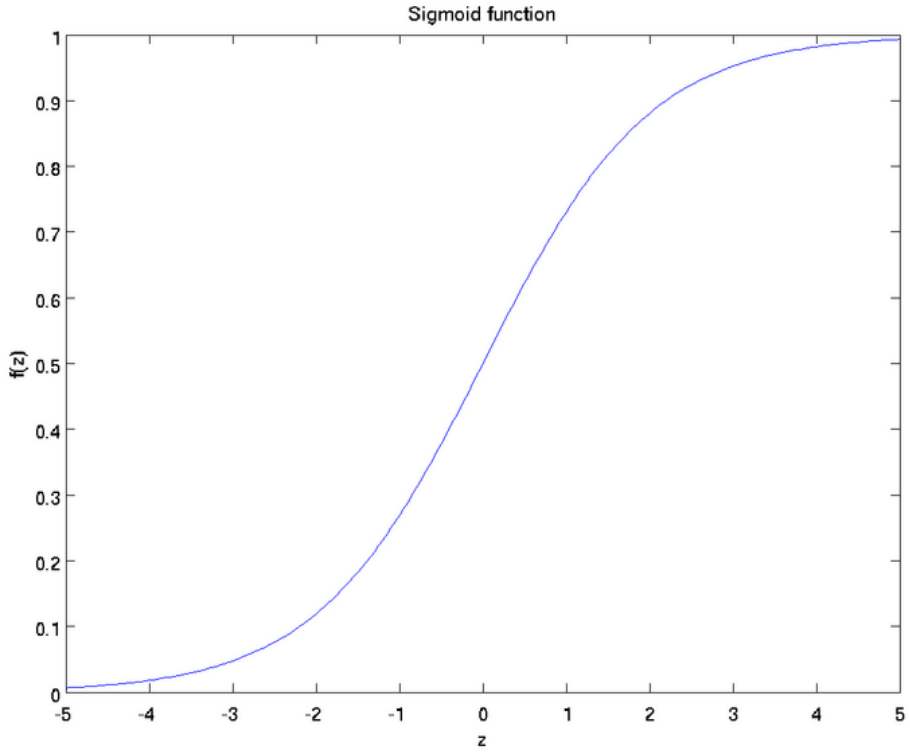
Neuron bazujący na funkcji tangensa hiperbolicznego jest to modyfikacja neurona liniowego, w której rolę funkcji aktywacji zajmuje wyżej wspomniany tangens hiperboliczny.

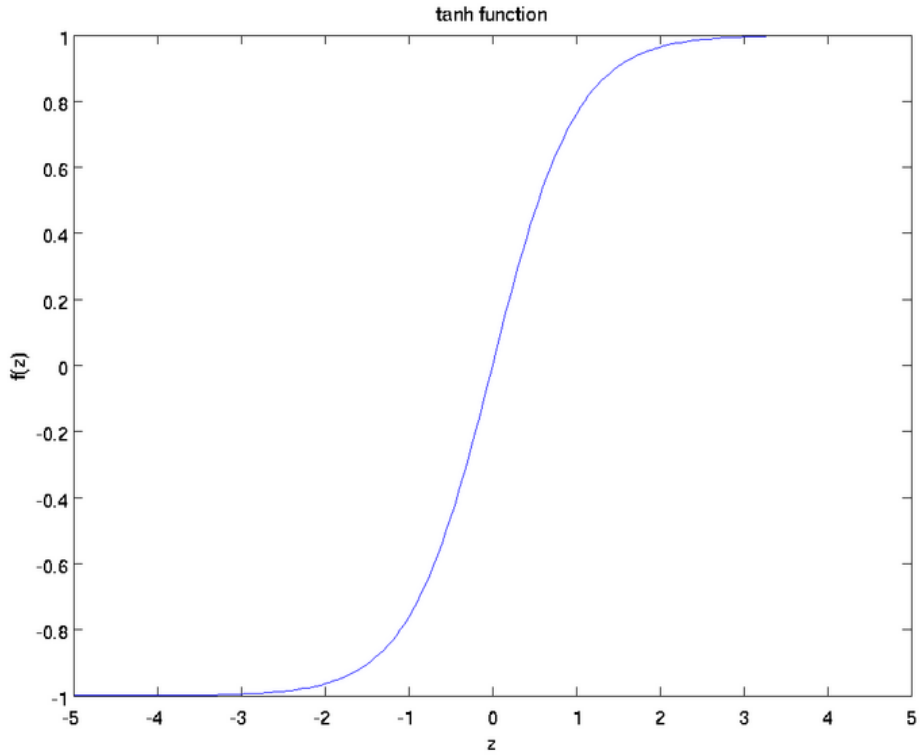
Funkcja sigmoidalna

$$f(z) = \frac{1}{1 + \exp(-z)}$$

Funkcja tangensa hiperbolicznego (tanh)

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$





Przykłady działania programu

Program po jego wywołaniu, wypisuje ilość danych treningowych, a następnie przystępuje do treningu sieci według wcześniej zadeklarowanego algorytmu. Z każdą epoką, wypisuje na ekran całkowity błąd uczenia, dzięki któremu jesteśmy w stanie ocenić czy sieć się dobrze zachowuje i ćwiczy na zadanym secie danych. Do trenowania użyto zestawu 20 rekordów, 10 dużych oraz 10 małych liter. Warstwa ukryta zawsze posiadała 5 neuronów

```
Number of training patterns:  20
Total error: 0.204931585302
Total error: 0.198745991941
Total error: 0.193448919733
Total error: 0.186803446221
Total error: 0.18230086249
Total error: 0.175755213379
Total error: 0.169423421207
```

Po dokonaniu treningu na zadanej wcześniej ilości epok, program zwraca błąd po ostatniej epoce. Można przejść do testowania sieci.

```
Total error: 0.00010808301858
Total error: 0.000108174190322
Total error: 0.000107569070588
Total error: 0.000107786525316
```

Program w pętli wypisuje na ekran literki z wcześniej zadeklarowanej tabeli. Na podstawie danych testowych porównuje wartość output w określaniu wielkości litery, z wcześniej zaimplementowaną na sztywno zmienną porównawczą. Jeżeli wartość output jest większa od zmiennej porównawczej, litera jest określana jako duża.

```
Dla literki F output wynosi
1.01821807642
Literka F jest dużą literą

Dla literki G output wynosi
1.0007430194
Literka G jest dużą literą
```

```
Dla literki f output wynosi
0.0390348605785
Literka f jest małą literą

Dla literki h output wynosi
0.00979638086007
Literka h jest małą literą
```

Wszystkie wyniki oraz wykresy są podane w pliku PSI_wyniki, załączonym do repozytorium.

Analiza i wnioski

W każdym branym pod uwagę pomiarze, bazowałem na stałej liczbie epok równej 1000. Learning rate miał dwie wartości – 0.1 oraz 0.01 . Zmienna była również funkcja warstwy ukrytej – sigmoidalna oraz tangensH.

Analizując wykres znajdujący się na arkuszu pierwszym, możemy zauważyć iż funkcja sigmoidalna potrzebowała mniej epok aby osiągnąć możliwie niski błąd. W porównaniu do sieci bazującej na funkcji tangensH, różnica wynosiła około 190 epok. Przy ustaleniu granicy nauczenia sieci na ~360 epoce, widzimy iż wersja pierwsza była w stanie się nauczyć lekko ponad dwukrotnie szybciej, od wersji funkcji tanh.

Dane kolejnego wykresu, różni od poprzedniego bardzo ważna zmiana. Mianowicie learning rate został pomniejszony dziesięciokrotnie – z wartości 0.1 do 0.01. Zabieg ten spowodował, iż to wersja sieci z tangensem hiperbolicznym około 70 epoki, znacząco lepiej wypada na tle funkcji sigmoidalnej. Możemy również zauważyć, że błąd liczony przy pomocy wcześniej wspomnianej funkcji sigm, maleje aż do końca przeprowadzonych badań. Świadczy to poważnym wydłużeniu czasu nauki sieci.

Kolejne cztery arkusze zawierają dane dotyczące poprawnego odgadywania wielkości liter, przez zadeklarowaną sieć. Widzimy iż najlepszy wynik na tym polu osiąga sieć bazująca na funkcji sigmoidalnej, posiadająca learning rate na poziomie 0.1 (Arkusz 3). Wszystkie litery zostały odgadnięte poprawnie w każdym z 5 przeprowadzonych pomiarów.

W arkuszu 5, pomiar został poprzedzony zmianą wartości l.rate do poziomu 0.01. Już na pierwszy rzut oka jesteśmy w stanie zobaczyć iż nie wszystkie litery zostają odgadnięte z taką samą dokładnością jak w momencie gdy współczynnik nauki jest równy 0.1. Zarówno po stronie liter dużych (H) jak i małych (f) znajdują się przypadki gdy sieć zachowuje się mocno niepewnie, nie mogąc rozróżnić wielkości literek.

Arkusz 6 oraz 7 bazują na danych stworzonych dzięki sieci bazującej na funkcji tangensu hiperbolicznego. Zarówno dla współczynnika nauki 0.1 jaki i 0.01, niepewność jest dość duża. O ile przy pierwszej wartości, sieć za wyjątkiem jednego pomiaru osiąga podobne wyniki, o tyle przy wartości dziesięciokrotnie mniejszej wyniki są bardzo rozbieżne. Zarówno po stronie małych jak i dużych liter, poszczególne pomiary mają błędne sugestie wyników, przez co użytkownik na ich podstawie nie jest w stanie zidentyfikować jaka jest wielkość danej litery. Świadczyć to może o zbyt małej liczbie epok nauki w stosunku do jej współczynnika, przez co sieć neuronowa nie miała możliwości pozytywnego wyniku nauki.

Na podstawie otrzymanych wyników jestem w stanie stwierdzić, iż sieć neuronowa która posiada warstwę ukrytą neuronów bazującą na funkcji sigmoidalnej, zachowała się lepiej w porównaniu do neuronów typu tanh przy większym współczynniku nauki. Biorąc pod uwagę trywialność problemu zadania z punktu widzenia programu, logicznym był wybór większego learn rate, który był w stanie znacznie skrócić czas (ilość epok) potrzebny do nauki rozwiązywania danego problemu. Patrząc na ogólne zachowanie testowanych sieci, jesteśmy w stanie dojrzeć iż neurony bazujące na funkcji tangensa hiperbolicznego, poradziły sobie z problemem szybciej przy niższym niższym learn rate. Niestety, przy testowaniu wyników ich

rezultat był bardzo niestabilny, co powodowało duże rozbieżności zaobserwowane na wykresie.

Kod:

```
from pybrain3.datasets import SupervisedDataSet

inputDataSet = SupervisedDataSet(35, 1)          #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)

#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
    ),
    1)

#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
    ),
    1)

#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
    ),
    1)

#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
    ),
    1)

#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
    ),
    1)

#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
    ),
    1)

#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
    ),
    1)

#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
    ),
    1)

#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
    ),
    1)

#a
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)

#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)

#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)

#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)

#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)

#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)

#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)

#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)

#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)

#z
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)
from pybrain3 import *
from pybrain3.supervised.trainers import BackpropTrainer
from testinput import inputDataSet


network = FeedForwardNetwork()          #Creating new network
letters = ["A", "B", "C", "D", "I", "F", "G", "H", "K", "U", "a", "b", "c",
"d", "f", "h", "m", "o", "w", "z"]

inLayer = LinearLayer(35)             #Creating input layer
hiddenLayer = SigmoidLayer(5)         #Creating hidden layer
outLayer = LinearLayer(1)             #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
errorCompare = 0.9                                       #Comparator

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

trainer = BackpropTrainer(network, dataset=inputDataSet, learningrate=0.1,
verbose=True)    #Initializing trainer with Backpropagation method

trainer.trainEpochs(1000)
#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])
    print(temp[0])
    if temp > errorCompare :
```

```
        print("Litera",letters[i],"jest duża")
    else:
        print("Litera",letters[i],"jest mała")

print("\n\n Koniec testów"
```

Źródła użyte przy tworzeniu sieci oraz sprawozdania:

Książka Stanisława Osowskiego – Sieci neuronowe do przetwarzania informacji

http://edu.pjwstk.edu.pl

http://pybrain.org/docs/

https://stackoverflow.com/