Tomasz Zapiórkowski

**Podstawy Sztucznej Inteligencji – Sprawozdanie**

Scenariusz 3

Budowa i działanie sieci wielowarstwowej

**Opis wykonywanego ćwiczenia:**

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

**Wykonane zadania:**

1. Wygenerowano dane uczące i testujące, zawierające 20 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).
3. Wykorzystano narzędzie PyBrain do trenowania sieci algorytmem wstecznej propagacji błędu (BackpropTrainer).
4. Uczono sieci modyfikując współczynnik uczenia i współczynnik bezwładności (4 kombinacje).
5. Testowano poprawność działania sieci.

**Specyfikacja:**

Algorytm wstecznej propagacji błędu: metoda uczenia sieci wielowarstwowej, w której błąd ostatniej warstwy jest przesyłany wstecz i wykorzystywany do zmiany wartości wag w poprzednich warstwach.

Ogólny schemat procesu trenowania sieci wygląda następująco:

1. Ustalamy topologię sieci, tzn. liczbę warstw, liczbę neuronów w warstwach.
2. Inicjujemy wagi losowo (na małe wartości).
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 (również w warstwach ukrytych) modyfikuje wagi na podstawie wartości błędu i wielkości przetwarzanych w tym kroku sygnałów.
7. Powtarzamy od punktu 3. dla kolejnych wektorów uczących. Gdy wszystkie wektory zostaną użyte, losowo zmieniamy ich kolejność i zaczynamy wykorzystywać powtórnie.
8. Zatrzymujemy się, gdy średni błąd na danych treningowych przestanie maleć. Możemy też co jakiś czas testować sieć na specjalnej puli nieużywanych do treningu próbek testowych i kończyć trenowanie, gdy błąd przestanie maleć.

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

****

**Wykonanie zadania:**

Zbudowana sieć była złożona z następujących składników:

- 35 neuronów liniowych warstwy wejścia (każdy neuron odpowiada za jeden znak tablicy reprezentującej daną literę),

- 30 neuronów sigmoidalnych warstwy ukrytej do których dołączony był bias,

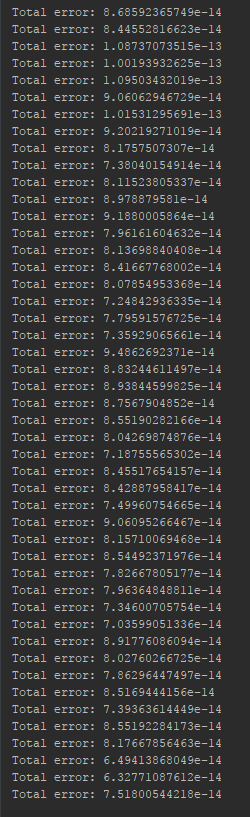
- 20 neuronów liniowych warstwy wyjścia (każdy neuron odpowiada za jeden element tablicy identyfikującej daną literę).

Działanie programu testowano w czterech kombinacjach. Zmiennymi były: współczynnik uczenia i współczynnik bezwładności. Współczynnik uczenia przybierał wartości 0.1 i 0.05, natomiast współczynnik bezwładności 0.1 i 0.01.

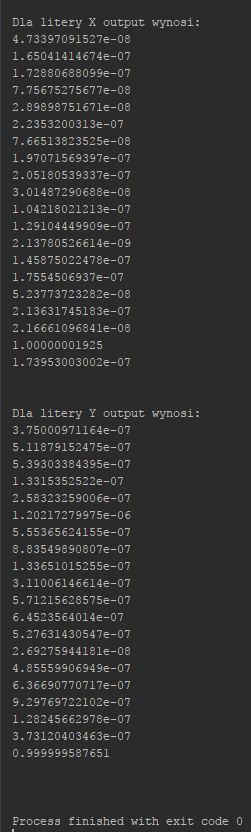
Współczynnik bezwładności, zwany także momentum, jest współczynnikiem od którego zależy korekta wag neuronu. Kiedy stosowane jest momentum, korekta wag neuronu zależy nie tylko od sygnału wejściowego i błędu, jaki neuron popełnił, ale również od tego, jak duża była korekta wag w poprzednim kroku uczenia. 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.

**Przykładowe działania programu:**

Program po wywołaniu przystępuje do treningu sieci wg algorytmu propagacji wstecznej każdorazowo (po każdej epoce) wypisując całkowity błąd uczenia.



Po wytrenowaniu sieci program przechodzi do jej testowania. Program analizuje dane z wyjścia porównując je do danych wprowadzonych wcześniej jako dane oczekiwane.



Na powyższym zrzucie mamy analizę litery X i Y. Sieć poprawnie rozpoznaje te litery, co widzimy poprzez wartości bliskie 1 na przedostatnim i ostatnim miejscu listy. Widać też, że błędy popełniane nie są duże, pomyłki na literach są niewielkie.

Wszystkie te dane są zebrane i odpowiednio opakowanie w 4 plikach excel, każdym dla każdej kombinacji. W każdym pliku znajdują się 22 arkusze.

Pierwszy arkusz to zbiór ogólny uśrednionych danych z trzech wyników pokazujący jak dobrze sieć radzi sobie z rozpoznawaniem ogółu danych.

Drugi arkusz zajmuje się analizą błędów popełnianych przy uczeniu. Do wykresu analizującego zmianę błędu dodano linię trendu tej zmiany.

Pozostałe 20 arkuszy to analiza każdej litery z osobna. Zawarte jest tam skuteczność rozpoznawania litery oraz podobieństwo do innych liter.

Wyniki zostały zamieszczone w formie wykresów w załączonych plikach, każdy plik to jedna z kombinacji.

**Analiza:**

Proces uczenia odbywał się przez ograniczoną ilość 5000 epok.

Dla każdej kombinacji przeprowadzono trzy niezależne próby, co oznacza że ogółem wykonano 12 testów.

Podstawowym parametrem porównawczym był ogólny błąd. Ustalono, że wartością graniczną będzie błąd <0.001.

Do osiągnięcia tego progu najmniej epok potrzebował program, w którym lrate był równy 0.05, natomiast momentum 0.01, potrzebował do tego około 1550 epok, co jest dość dobrym wynikiem.  
Następnie w kolejności z około 1925 epokami była kombinacja lrate=0.05, momentum 0.1. Na trzecim miejscu z 2750 epokami lrate 0.1, momentum 0.1 i na końcu z 3700 epokami lrate 0.1, momentum 0.01.

Oprócz tego warto zauważyć, że wszystkie testowane kombinacje poprawnie przechodziły testy i za każdym raz odpowiednio rozpoznawały litery.

Poszukiwałem również podobieństw między literami, ale analizując wykresy podobieństwa nie znalazłem żadnych powtarzających się wzorców.

**Wnioski**

Na podstawie wyników można stwierdzić, że stworzona sieć neuronowa bardzo dobrze radziła sobie z postawionym zadaniem czyli rozpoznawaniem liter. Algorytm wstecznej propagacji błędu bardzo dobrze zrealizował uczenie tej sieci, dzięki czemu niewielka liczba epok pozwoliła na nauczenie sieci poprawnego działania. Analizowano podobieństwo między literami lecz wyniki były na tyle chaotyczne, że nie stwierdzono podobieństwa. Jednakże warto odnotować fakt, że za każdym razem poprawnie rozpoznano litery a błędy poznawcze były stosunkowo niskie. Program więc spełnił swoje zadanie jakim było rozpoznawanie wielkich liter alfabetu łacińskiego.

**Listing:**

inputLetters.py

from pybrain3.datasets import SupervisedDataSet  
  
inputDataSet = SupervisedDataSet(35, 20) # Creating new DataSet  
  
# 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  
),  
 (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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 0, 1))

network.py

from pybrain3 import \*  
  
network = FeedForwardNetwork() # Creating new network  
  
inLayer = LinearLayer(35) # Creating input layer  
hiddenLayer = SigmoidLayer(30) # 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

trainer.py

from pybrain3.supervised.trainers import BackpropTrainer  
from inputLetters import inputDataSet  
  
import inputLetters  
import network  
  
letters = ["A", "B", "C", "D", "I", "F", "G", "H", "K", "U", "M", "E", "L", "O", "P", "R", "T", "W", "X", "Y"]  
  
inp = inputDataSet['input'] # Making shortcut to the input section of DataSet  
  
trainer = BackpropTrainer(network.network, dataset=inputLetters.inputDataSet, learningrate=0.05, verbose=True,  
 momentum=0.1) # verbose = true so printing errors for each epoch  
  
trainer.trainEpochs(5000) # Training network for X epochs  
  
print("\n\n")  
for i in range(20): # Final print  
 print("Dla litery", letters[i], "output wynosi:")  
 temp = network.network.activate(inp[i])  
 for k in range(20):  
 if temp[k] < 0:  
 temp[k] \*= (-1) # Only to improve the analysis of the result  
 for j in range(20):  
 print(temp[j])  
 print("\n")