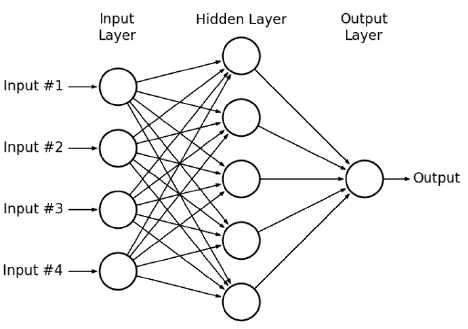
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Jarosz Wojciech | Gr. 1 | PSI | Sprawozdanie 3 | Temat: Budowa i działanie sieci wielowarstwowej typu feedforward |

1. Cel ćwiczenia:

Celem ćwiczenia jest poznanie budowy i działania wielowarstwowych sieci neuronowych poprzez uczenie kształtu wykresu funkcji matematycznej z użyciem algorytmu wstecznej propagacji błędu.

1. Wykonane zadania:
2. Wygenerowano dane uczące i testujące dla funkcji Rastrigin 3D dla danych wejściowych z przedziałów od -2 do 2.
3. Przygotowano (wykorzystano gotowe narzędzie) wielowarstwową sieć oraz algorytm wstecznej propagacji błędu.
4. Nauczono sieć dla różnych współczynników uczenia (np. 0.5, 0.1, 0.01) oraz różnej ilości warstw i neuronów.
5. Przetestowano sieć
6. Syntetyczny opis budowy oraz wykorzystanego algorytmu uczenia:



źródło: https://www.researchgate.net/figure/273768094\_fig2\_Figure-4-A-hypothetical-example-of-Multilayer-Perceptron-Network

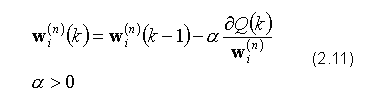
Typowa struktura sieci typu MLP, nazywana siecią w pełni połączoną, zakłada istnienie takiej architektury połączeń pomiędzy neuronami, w której wszystkie wyjścia warstwy wcześniejszej połączone są z odpowiednimi wejściami każdego neuronu warstwy następnej.

W klasycznej terminologii sieci neuronowych wyróżnia się warstwę wejściową, jedną lub dwie warstwy ukryte oraz warstwę wyjściową. Warstwa wejściowa pobiera dane z otoczenia i przesyła je do pierwszej warstwy ukrytej.

Następnie sygnał przesyłany jest na wejścia pierwszej warstwy ukrytej, która przetwarza dane i generuje sygnał wyjściowy podawany na wejścia warstwy kolejnej. Powyższy schemat powtarza się dla wszystkich kolejnych warstw ukrytych i kończy na warstwie wyjściowej, która zgodnie ze wzorcową architekturą oblicza wartości wyjść całej sieci i przekazuje je na zewnątrz

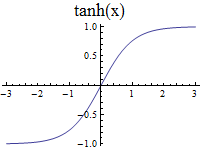
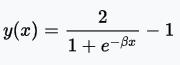
Algorytm wstecznej propagacji błędu:

Sprowadza się on do modyfikacji każdej z wag proporcjonalnie do wartości pochodnej cząstkowej funkcji celu. Modyfikacja wag w k-tej iteracji polega na odjęciu od wartości wag z iteracji poprzedniej (k - 1) wektora gradientu obliczonego dla bieżącej obserwacji. Algorytm jest sparametryzowany dzięki użyciu tzw. współczynnika uczenia, lub inaczej długości kroku, oznaczonego w poniższym równaniu symbolem α.



Wykorzystana funkcja aktywacji:

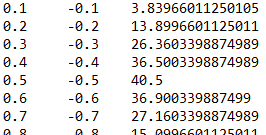
Sigmoidalna funkcja bipolarna (tangens hiperboliczny):

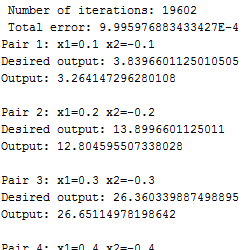
1. Zestawienie wyników:

Przykładowe dane uczące:

learning\_date.txt

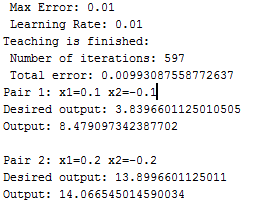
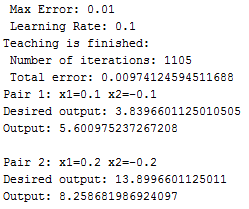


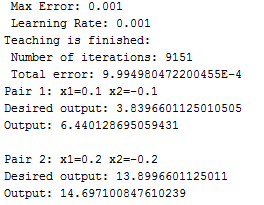
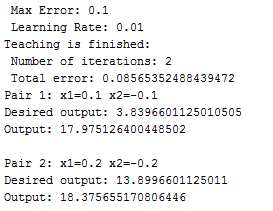
Przykładowe dane wyjściowe:



Ilość iteracji w zależności od współczynnika uczenia:

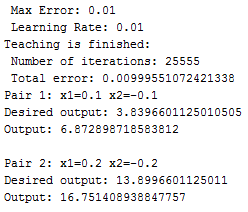
Błędy w otrzymywanych wynikach dla dwóch par przykładowych danych wejściowych dla wybranych trzech najlepszych konfiguracji:



Sprawdzenie czy ilość warstw ukrytych ma znaczenie dla uczenia. Poprzednie testy zostały przeprowadzone dla 3 warstw o liczbie neuronów : 28, 11, 18.

Wybrano najlepszy wariant czyi konfiguracja l=0.01 i e-0.01 i zmniejszono liczbę warstw do jednej o ilość neuronów = 5:



1. Analiza i dyskusja błędów uczenia i testowania opracowanej sieci w zależności od wartości współczynnika uczenia oraz wybranego algorytmu:

Dla współczynnika uczenia 0.5 nauka trwała bardzo długo i nie dochodziło do skutku – dlatego odrzucono ten współczynnik. Jedynie wraz z akceptowanym błędem 0.5 zakończyła się powodzeniem ale różnice w wynikach otrzymanych i oczekiwanych były wielkie.

Przy błędzie akceptowanym = 0.5 ilość iteracji była równa 1.

1. Wnioski:

Bład akceptowany musi być wystarczająco niski aby sieć została odpowiednio wyuczona, ponieważ różnice w wartościach oczekiwanych a otrzymanych są o wiele za duże.

Im niższy współczynnik uczenia tym dokładniejsze wyniki

Im mniejszy akceptowany błąd tym dokładniejsze wyniki.

Poniżej pewnych wartości błędu akceptowanego i współczynnika uczenia, nie następuje znaczna poprawa otrzymanych wyników.

Najlepszą konfiguracją w przeprowadzonych testach okazała Asię e=0.01, l=0.01

1. Listing:

**package** com.company;  
  
**import** org.neuroph.core.NeuralNetwork;  
**import** org.neuroph.core.data.DataSet;  
**import** org.neuroph.core.data.DataSetRow;  
**import** org.neuroph.nnet.MultiLayerPerceptron;  
**import** org.neuroph.nnet.learning.BackPropagation;  
**import** org.neuroph.util.TransferFunctionType;  
  
**import** java.io.File;  
**import** java.io.FileNotFoundException;  
**import** java.util.Scanner;  
  
**public class** Main {  
  
 **public static void** main(String[] args) {  
 *//uworzeine danych testowych* DataSet trainingData = **new** DataSet(2, 1);  
 trainingData.setLabel(**"TrainingData"**);  
 *//dane do normalizacji danych wejściowych i wyjściowych* **double** a = 0.5;  
 **double** b = (-1.0-(a\*(-2.0)));  
  
 **double** a2 = Math.*abs*(2/(80.54857778-0.101784126));  
 **double** b2 = (-1.0-(a2\*(0.101784126)));  
 *//wczytanie danych wejściowcyh z pliku* File file = **new** File(**"learning\_data.txt"**);  
 Scanner in = **null**;  
 **double**[][] data = **new double**[40][3];  
 String[] result;  
 **try** {  
 in = **new** Scanner(file);  
 **for**(**int** i=0;i<40;i++){  
 *//while (in.hasNextLine()){* String line = in.nextLine();  
 result = line.split(**"\\s"**);  
 trainingData.addRow(**new** DataSetRow(**new double**[]{a\*Double.*parseDouble*(result[0]) + b, a\*Double.*parseDouble*(result[1])+b},**new double**[]{a2\*Double.*parseDouble*(result[2])+b2}));  
 }  
 } **catch** (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 *//wybranie metody backPropagation i ustawienie max łędu i współczynnika uczenia* BackPropagat ion backPropagation = **new** BackPropagation();  
 backPropagation.setMaxError(0.01);  
 backPropagation.setLearningRate(0.01);  
 *//utworzenie sieci wielowarstwowej i ustalenie ilośći warstw wewnętrznych* MultiLayerPerceptron multiLayerPerceptron = **new** MultiLayerPerceptron(TransferFunctionType.***TANH***, 2, 5, 1);  
 multiLayerPerceptron.setLabel(**"MyNetwork"**);  
  
 multiLayerPerceptron.setLearningRule(backPropagation);  
  
 System.***out***.println(**"Teaching: "** + multiLayerPerceptron.getLabel() + **", with data set: "** + trainingData.getLabel());  
 System.***out***.println(**"Using algorithm backPropagation with configuration: \n Max Error: "** + backPropagation.getMaxError() + **"\n Learning Rate: "** + backPropagation.getLearningRate());  
 multiLayerPerceptron.learn(trainingData);  
 System.***out***.println(**"Teaching is finished:"**);  
 *// Wyswietlenie liczby iteracji oraz całkowitego błędu uczenia w sieci neuronowej* System.***out***.println(**" Number of iterations: "** + backPropagation.getCurrentIteration());  
 System.***out***.println(**" Total error: "** + backPropagation.getErrorFunction().getTotalError());  
 *//zapis sieci* multiLayerPerceptron.save(**"my.nnet"**);  
 *//testowanie* NeuralNetwork neuralNetwork = NeuralNetwork.*createFromFile*(**"my.nnet"**);  
 **int** counter = 1;  
 **for** (DataSetRow dataSetRow : trainingData.getRows()) {  
 **double**[] input = dataSetRow.getInput();  
 System.***out***.println(**"Pair "** + counter + **": x1="** + ((input[0] - b) / a) + **" x2="** + ((input[1] - b) / a));  
   
 **double**[] desiredOutput = dataSetRow.getDesiredOutput();  
 neuralNetwork.setInput(dataSetRow.getInput());  
 neuralNetwork.calculate();  
 **double**[] output = neuralNetwork.getOutput();  
 System.***out***.println(**"Desired output: "** + ((desiredOutput[0] - b2) / a2));  
 System.***out***.println(**"Output: "** + ((output[0] - b2) / a2));  
 System.***out***.println();  
 counter++;  
  
 }  
  
 }  
}