SPRAWOZDANIE

Ćwiczenie 1
Z SIECI NEURONOWYCH

Autor - Aleksander Poławski 222350
Grupa - Czwartek 10:15
Prowadzący – P. Mgr Inż. Jan Jakubik

1. Cel ćwiczenia

Celem ćwiczenia pierwszego laboratoriów z sieci neuronowych jest poznanie podstawowych funkcji wykonywanych przez pojedynczy neuron, obserwacja zachowania neuronu przy różnych funkcjach przejścia oraz określenie wielkości, które mają wpływ na szybkość uczenia neuronu.

2. Plan ćwiczenia oraz badań

a) stworzenie programu symulującego działanie pojedynczego neuronu, w formie perceptronu prostego realizującego logiczną funkcję AND lub OR oraz przeprowadzenie eksperymentów badających szybkość uczenia się tego perceptronu w zależności od:

- zakresu wartości początkowych losowych wag
- współczynnika uczenia α
- zastosowanej funkcji przejścia neuronu (funkcji progowej unipolarnej lub funkcji progowej bipolarnej)

b) modyfikacja zaimplementowanego perceptronu prostego do Adaline dla funkcji bipolarnej oraz przeprowadzenie podobnych badań jak w przypadku perceptronu prostego – tj. badań szybkości uczenia się Adaline w zależności od zakresu początkowych, losowych wag oraz współczynnika uczenia α

3. Opis utworzonej aplikacji

Aplikacja utworzona została przy pomocy IDEA ‘IntelliJ’ w języku Java.

Składa się z trzech klas: Main, Adeline oraz Perceptron.

Utworzenie obiektu klasy Adeline lub Perceptron umożliwia sparametryzowanie zakresu startowych wag oraz współczynnika uczenia α. Dla obiektu klasy Perceptron możemy również ustawić typ funkcji przejścia na ‘unipolarny’ lub ‘bipolarny’. Dla obiektów typu Adeline jest to zawsze funkcja bipolarna. Poniżej załączono kod konstruktorów klas: Adeline oraz Perceptron:

```
public Adaline(double starting_weight_range, double alpha_modifier, Random generator)
{
    this.modifier = alpha_modifier;

    x_vector[0] = 1;

    for(int i=0; i<w_vector.length; i++)
    {
        w_vector[i] = generator.nextDouble()*starting_weight_range;
        if (generator.nextInt(2)==0)
        {
            w_vector[i]*= -1;
        }
    }
}
```

```
public Perceptron(double starting_weight_range, double alpha_modifier, String function_type, Random generator)
{
    this.modifier = alpha_modifier;
    this.generator = generator;
    this.function_type = function_type;

    x_vector[0] = 1;

    for(int i=0; i<w_vector.length; i++)
    {
        w_vector[i] = generator.nextDouble()*starting_weight_range;
        if (generator.nextInt(2)==0)
        {
            w_vector[i]*= -1;
        }
    }
}
```

Obie klasy różnią się od siebie metodami zgodnie z zasadami działania neuronów typu Perceptron i Adeline. Przykładem może być sposób liczenia błędu, który w Adeline następuje przed funkcją przejścia, a dla perceptronu prostego po funkcji przejścia (oraz dodatkowo zależy od rodzaju użytej funkcji przejścia). Poniżej przykład metody liczącej wyjście, błąd i wyjście po funkcji przejścia dla Adeline:

```
private void calculateOutput()
{
    double temp_y = 0;
    calculated_y = -1;

    for(int i=0; i<w_vector.length; i++)
    {
        temp_y += w_vector[i]*x_vector[i];
    }

    error = y - temp_y;

    if (temp_y >= threshold)
    {
        calculated_y = 1;
    }
}
```

Funkcja Main posiada logikę odpowiedzialną za kontrolę uczenia Adeline oraz perceptronu oraz wykonywanie testów założonych w ćwiczeniu. Poniżej załączono kod sparametryzowanego testu ogólnego ‘customTest()’ oraz przykładowego testu wykorzystującego go:

```
private static void customTest(String neuron_type, double starting_weight_range, double alpha_modifier, String function_type)
{
    double mean_iterations = 0;

    for (int j=0; j<100; j++)
    {
        if (neuron_type.equals("ADALINE"))
        {
            mean_iterations += teachAdaline(starting_weight_range, alpha_modifier);
        }
        else if (neuron_type.equals("PERCEPTRON"))
        {
            mean_iterations += teachPerceptron(starting_weight_range, alpha_modifier, function_type);
        }
    }

    mean_iterations/= 100;

    if (mean_iterations>10000)
    {
        print("Teaching was not achieved in 10000 iterations with this settings");
    }
    else
    {
        print("Mean number of iterations to teach after 100 trials: " + mean_iterations);
    }
}
```

```
public static void neuronDifferentAlphasTest(String neuron_type)
{
    DecimalFormat double_format = new DecimalFormat("#0.000");
    double[] alphas = {0.001, 0.01, 0.02, 0.022, 0.024, 0.026, 0.028, 0.029, 0.03};
    double starting_weight_range = 0.8;
    String function_type = "bipolar";

    double alpha_modifier;

    print("");
    print("~~~~~~"+neuron_type+" DIFFERENT ALPHA MODIFIER TEST~~~~~~");
    if (neuron_type.equals("ADALINE"))
    {
        print("~~~~~~TEACHING UNTIL (LMS <= "+ LMS_threshold +")~~~~~~");
    }

    for (int i=0; i<alphas.length; i++)
    {
        alpha_modifier = alphas[i];
        print("Alpha modifier: ("+ (double_format.format(alpha_modifier))+")");

        customTest(neuron_type, starting_weight_range, alpha_modifier, function_type);
    }
}
```

Powyżej opisana organizacja aplikacji i kodu pozwoliła na łatwe wykonanie wszystkich testów jednocześnie oraz łatwe tworzenie testów kolejnych. Poniżej załączono kod ukazujący wywołanie wszystkich testów jednocześnie:

```
public static void main(String[] args)
{
    neuronDifferentStartingWeightsTest("PERCEPTRON");
    neuronDifferentAlphasTest("PERCEPTRON");

    neuronDifferentStartingWeightsTest("ADALINE");
    neuronDifferentAlphasTest("ADALINE");

    neuronDifferentFunctionTypeTest();
}
```

4. Badania

Eksperyment 1 – zbadanie szybkości uczenia się perceptronu prostego w zależności od współczynnika uczenia α

Założenia:
- typ problemu „AND”
- typ funkcji przejścia bipolarny
- zakres startowych wag (-0.8 do 0.8)
- próg funkcji przejścia równy 0
- zmienna wartość współczynnika uczenia: 0.001, 0.010, 0.020, 0.024, 0.026, 0.028, 0.029, 0.030

Przebieg eksperymentu:
- dla każdej wartości współczynnika wykonanych zostało 100 procedur uczenia, a następnie (dla każdej wartości) obliczona została średnia ilość iteracji uczenia

Otrzymane wyniki:

Współczynnik uczenia	Średnia ilość iteracji ze 100 prób
0,001	145,2
0,010	15,46
0,020	8,81
0,024	7,77
0,026	7,09
0,028	7,17
0,029	6,57
0,030	6.34
0,1	3.04

Komentarz: Im większy współczynnik uczenia, tym szybciej udaje się znaleźć wagi umożliwiające rozwiązanie problemu.

Eksperyment 2 – zbadanie szybkości uczenia się perceptronu prostego w zależności od zakresu wag początkowych

Założenia:
- typ problemu „AND”
- typ funkcji przejścia bipolarny
- zmienny zakres startowych wag (-1.5 do 1.5), (-1.3 do 1.3), (-1.1 do 1.1), (-0.9 do 0.9), (-0.7 do 0.7), (-0.5 do 0.5), (-0.3 do 0.3), (-0.1 do 0.1)
- próg funkcji przejścia równy 0
- wartość współczynnika uczenia: 0.01

Przebieg eksperymentu:
- dla każdego zakresu wag początkowych wykonanych zostało 100 procedur uczenia, a następnie (dla każdej wartości) obliczona została średnia ilość iteracji uczenia

Otrzymane wyniki:

Zakres wag początkowych	Średnia ilość iteracji ze 100 prób
-1.5 do 1.5	28.0
-1.3 do 1.3	22.88
-1.1 do 1.1	22.87
-0.9 do 0.9	16.63
-0.7 do 0.7	15.41
-0.5 do 0.5	9.83
-0.3 do 0.3	6.73
-0.1 do 0.1	3.36

Komentarz: Im mniejszy zakres wag, tym szybciej udaje się znaleźć wagi umożliwiające rozwiązanie problemu. Należy jednak zauważyć, że zakres zawężamy w stronę prawidłowych dla rozwiązania wag.

Eksperyment 3 – zbadanie szybkości uczenia się perceptronu prostego w zależności od typu funkcji przejścia

Założenia:
- typ problemu „AND”
- zmienny typ funkcji przejścia bipolarny lub unipolarny
- zakres startowych wag (-0.8 do 0.8)
- próg funkcji przejścia równy 0 dla funkcji bipolarnej i 0.5 dla funkcji unipolarnej
- wartość współczynnika uczenia: 0.1

Przebieg eksperymentu:
- dla każdego typu funkcji przejścia wykonanych zostało 100 procedur uczenia, a następnie (dla każdej wartości) obliczona została średnia ilość iteracji uczenia

Otrzymane wyniki:

Typ funkcji przejścia	Średnia ilość iteracji ze 100 prób
bipolarna	3.24
unipolarna	1.69

Komentarz: Funkcja przejścia unipolarna wpływa pozytywnie na szybkość uczenia przy pozostałych niezmiennych parametrach.

Eksperyment 4 – zbadanie szybkości uczenia się Adeline w zależności od współczynnika uczenia α

Założenia:
- warunek stopu uczenia, kiedy błąd średniokwadratowy po iteracji jest mniejszy lub równy 0.3
- typ problemu „AND”
- typ funkcji przejścia bipolarny
- zakres startowych wag (-0.8 do 0.8)
- próg funkcji przejścia równy 0
- zmienna wartość współczynnika uczenia: 0.001, 0.010, 0.020, 0.024, 0.026, 0.028, 0.029, 0.030

Przebieg eksperymentu:
- dla każdej wartości współczynnika wykonanych zostało 100 procedur uczenia, a następnie (dla każdej wartości) obliczona została średnia ilość iteracji uczenia

Otrzymane wyniki:

Współczynnik uczenia	Średnia ilość iteracji ze 100 prób
0,001	181.7
0,010	21,69
0,020	13.0

0,024	12.06
0,026	11.68
0,028	12.44
0,029	14.27
0,030	>10000

Komentarz: Im większy współczynnik uczenia, tym szybciej udaje się znaleźć wagi umożliwiające rozwiązanie problemu, ale tylko do pewnego momentu. Przy współczynniku większym niż 0,029 nastąpił silny spadek wydajności algorytmu.

Eksperyment 5 – zbadanie szybkości uczenia się Adeline w zależności od zakresu wag początkowych

Założenia:
- warunek stopu uczenia, kiedy błąd średniokwadratowy po iteracji jest mniejszy lub równy 0.3
- typ problemu „AND”
- typ funkcji przejścia bipolarny
- zmienny zakres startowych wag (-1.5 do 1.5), (-1.3 do 1.3), (-1.1 do 1.1), (-0.9 do 0.9), (-0.7 do 0.7), (-0.5 do 0.5), (-0.3 do 0.3), (-0.1 do 0.1)
- próg funkcji przejścia równy 0
- wartość współczynnika uczenia: 0.01

Przebieg eksperymentu:
- dla każdego zakresu wag początkowych wykonanych zostało 100 procedur uczenia, a następnie (dla każdej wartości) obliczona została średnia ilość iteracji uczenia

Otrzymane wyniki:

Zakres wag początkowych	Średnia ilość iteracji ze 100 prób
-1.5 do 1.5	26.78
-1.3 do 1.3	25.48
-1.1 do 1.1	23.42
-0.9 do 0.9	21.69
-0.7 do 0.7	21.40
-0.5 do 0.5	20.73
-0.3 do 0.3	19.82
-0.1 do 0.1	19.66

Komentarz: Im mniejszy zakres wag, tym szybciej udaje się znaleźć wagi umożliwiające rozwiązanie problemu. Należy jednak zauważyć, że zakres zawężamy w stronę prawidłowych dla rozwiązania wag.

Eksperyment 6 – wszystkie powyższe eksperymenty zostały wykonane dla problemu „OR”

Komentarz: Zgodnie z oczekiwaniami wszystkie wyniki i zależności pozostały bez większych zmian, ponieważ działanie algorytmów nie powinno zależeć od rodzaju użytego problemu (dla tych dwóch problemów)

Eksperyment 7 – wszystkie eksperymenty dla Adeline zostały wykonane wprowadzając inny warunek stopu równy 0.4

Komentarz: Zgodnie z oczekiwaniami wszystkie wyniki poprawiły się, a szybkość uczenia Adeline okazała się lepsza niż szybkość uczenia perceptronu prostego.

5. Wnioski i podsumowanie

- wydajność Adeline jest silnie zależna od warunku stopu. Bardziej rygorystyczny warunek może sprawić, że algorytm będzie wolniejszy od algorytmu perceptronu prostego.

- zawężanie zakresu wag w kierunku wag prawidłowych do rozwiązania problemu znacznie pomaga przyspieszyć proces uczenia

- im większy współczynnik uczenia tym szybciej algorytmy znajdują rozwiązanie. Zbyt duża jego wartość wpłynie jednak negatywnie na proces uczenia, ponieważ algorytm będzie omijał prawidłowe rozwiązanie

- szybkość uczenia jest niezależna od tego czy problemem będzie „OR” czy „AND”

- pojedynczy perceptron prosty i Adeline są w stanie rozwiązywać tylko problemy separowalne liniowo. Nie są wstanie rozwiązać problemu na przykład typu „XOR”