Informatyka, studia niestacjonarne, mgr II st. semestr I

Obliczenia inteligentne **2015/2016**

Prowadzący: dr inż. Kamil Stokfiszewski Niedziela, 13.45

Data oddania: _____ Ocena: _____

Szymon Łyszkowski 206809
Piotr Kluch 206799

Zadnie nr 3. Perceptron wielowarstwowy.

1. Cel

Głównym celem zadania było zapoznanie z zasadą działania perceptronu wielowarstwowego oraz metodą wstecznej propagacji błędu. Od sieci oczekuje się, że będzie w stanie rozpoznać wektory:
1,0,0,0
0,1,0,0
0,0,1,0
0,0,0,1
po wcześniejszym dokonaniu nauki metodą wstecznej propagacji błędów.

2. Wprowadzenie

Perceptron wielowarstwowy jest zbudowany z trzech warstw: wejściowej(kopiującej), ukrytej, wyjściowej. Warstwy brzegowe składają się z czterech neuronów a warstwa ukryta posiada dwa neurony. Dla każdego neuronu w sieci jest zastosowana sigmoidalna funkcja aktywacji o wzorze: $\frac{1}{1+e^{-x}}$

2.1. Wsteczna propagacja błędu

W wypadku wstecznej propagacji błędów proces nauki polega na poczliczeniu błędu każdego z neuronów poczynając od neuronów wyjściowych. Wartość błędu w kolejnych warstwach jest zależna od błędów neuronów w warstwie po niej następującej. Następnie wartość błędu jest wykorzystywana do modyfikacji wag wejść danej warstwy. Ponieważ wagi są wykorzystywane

w czasie obliczania błędu należy wybrać czy błąd ma być liczony na wagach początkowych dla procesu przetwarzania czy może już zmodyfikowanych.

2.2. Bias

Bias jest to opcjonalne dodanie wejścia do neuronu, które jest zawsze równe 1. Waga dla tego wejścia jest losowana tak samo jak wagi innych neuronów.

3. Opis implementacji

Implementacja została przygotowana w języku Python. Cała sieć jest realizowana przez klasę MultilayerPerceptron (networks/multilayer_perceptron /multilayer_perceptron.py), która udostępnia funkcjonalności wykorzystuje później w procesie uczenia: obliczanie wyjść poszczególnych warstw, obliczanie błędu danego neuronu, aplikacja nowych wag dla neuronu, zarządzanie biasem. Zadnie trzecie jest realizowane w module task_3.py (katalog tasks/):

```
if __name__ == '__main__':
patterns = teaching_patterns_with_desired_outputs()
perceptron = MultilayerPerceptron(None, 2, 4, 4, 2)
perceptron.add_bias()
train_network(100000, patterns, perceptron)
```

Po inicjalizacji perceptronu dodawany jest bias. Sieć jest najpierw trenowana danymi wzorcami treningowymi. Po każdej zakończonej epoce nauczania dane wejściowe są mieszane tak by nie przekazywać do sieci stałej kolejności wzorca uczącego.Po zakończonym treningu do sieci przekazywane są dane treningowe w poszukiwaniu rezultatów.

4. Wyniki

Dla sieci, która w swoim procesie uczenia wykorzystuje bias można uzyskać rezultaty, które umożliwią klasyfikację wzorca:
EP (expected pattern), OP (obtained pattern)
EP: [0, 0, 0, 1]
OP: [0.18552957942742349, 0.02840782788608539, 0.027492770991082334, 0.18733893209716337]
EP: [1, 0, 0, 0]
OP: [0.18461240887860034, 0.02831808419858233, 0.026795896877811618, 0.18638186699720616]
EP: [0, 0, 1, 0]
OP: [0.2864576340892995, 0.0020415773560059627, 0.8786148825401355, 0.30782468133962637]
EP: [0, 1, 0, 0]
OP: [0.30750372019569144, 0.9087236663292013, 0.0017103925172125617, 0.28791155017580755]

5. Dyskusja

Jest zauważalne iż dla niektórych wzorców rozpoznawanie jest bardzo dobre np. [0,1,0,0], [0,0,1,0]. Jednakże dla innych klasyfikacja może być utrudniona. Możliwą przyczyną takiego zachowania jest brak warunku stopu dla treningu poszczególnych wzorców. Takie zachowanie może powodować dobre wytrenowanie części przypadków a "przetrenowanie" pozostałej.

6. Wnioski

— Stan początkowy sieci wpływa w pewien sposób na możliwości jej nauki. Zwiększenie ilości neuronów ułatwia jej uczenie się dzięki zwiększeniu różnorodności w wagach początkowych.
— W wypadku nieprawidłowego doboru parametrów nauki i momentum proces nauki może uniemożliwić poprawną naukę.
— Obecność biasu, podobnie jak losowanie przypadków testowych jest niemal niezbędne do poprawnego działania sieci.