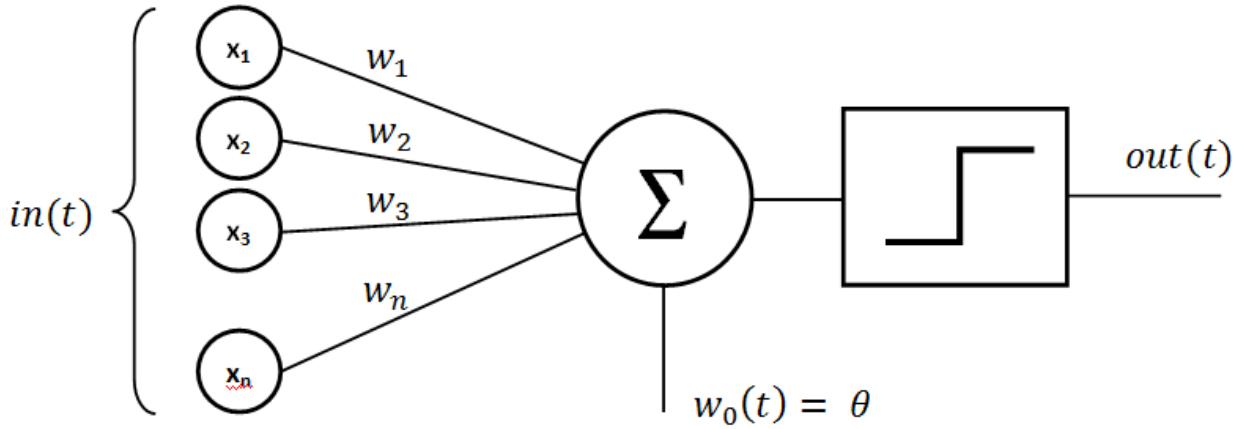
Podstawy Sztucznej Inteligencji – ćwiczenia projektowe
Sprawozdanie – 20.10.2017

W zadaniu została wykorzystana metoda uczenia, o której można przeczytać po adresem:
http://edu.pjwstk.edu.pl/wyklady/nai/scb/wyklad3/w3.htm

1. Schemat działania perceptronu:

Perceptron McCullocha-Pittsa



Perceptron to pewna programowa implementacja złożonej funkcji matematycznej, którą można by opisać wzorem:

$$out(t) = f\left(\sum_{i=1}^{n} x_i * w_i + \theta\right)$$

Gdzie funkcja *f(x)* może być opisana wzorem:

$$f(x) = \begin{cases} 1\ dla\ x \geq 0 \\ 0\ dla\ x < 0 \end{cases}$$

2. Metoda uczenia

Metoda uczenia perceptronu polega na modyfikacji wag poprzez porównanie otrzymanego wyniku z wynikiem oczekiwanym i takiej zmianie wag, by brały pod uwagę współczynnik uczenia i różnicę między wynikiem, a wartością oczekiwaną zgodnie ze wzorem:

$w_1 \mathrel{+}= \eta * (d\text{-}y) * x_1$
$w_2 \mathrel{+}= \eta * (d\text{-}y) * x_2$
$\theta \mathrel{+}= \eta * (d\text{-}y)$

gdzie:

- w_1, w_2 – wagi
- x_1, x_2 – wartości wejściowe
- η – niewielki współczynnik uczenia ($n > 0$)

- d – oczekiwana odpowiedź
- y – odpowiedź perceptronu
- θ – próg wzbudzenia perceptronu

W przeprowadzonym ćwiczeniu wykorzystane dane uczące, które były losowymi próbkami ze zbioru danych:
data: (0, 0), expected: 0
data: (0, 1), expected: 0
data: (1, 0), expected: 0
data: (1, 1), expected: 1

Dane testujące były takie same jak uczące.

3. Wyniki

Poniżej przedstawiam niektóre wyniki uczenia perceptronu funkcji AND:

---------- min iter number ----------

initialWeights:[0.11125, 0.04797];Perceptron(weights:[0.60610, 0.29540],bias:-0.73244,learnRate:(0.24743),activFunc:'naturalOne');iterNumber: 6
initialWeights:[0.36271, 0.02163];Perceptron(weights:[0.54192, 0.20084],bias:-0.68868,learnRate:(0.05974),activFunc:'naturalOne');iterNumber: 8
initialWeights:[0.20559, 0.37193];Perceptron(weights:[0.20559, 0.37193],bias:-0.53006,learnRate:(0.07117),activFunc:'naturalOne');iterNumber: 1
initialWeights:[0.91847, 0.74637];Perceptron(weights:[0.67844, 0.74637],bias:-0.78546,learnRate:(0.24003),activFunc:'naturalOne');iterNumber: 4
initialWeights:[0.10173, 0.04372];Perceptron(weights:[0.63168, 0.57367],bias:-0.87016,learnRate:(0.52995),activFunc:'naturalOne');iterNumber: 8
initialWeights:[0.59095, 0.51119];Perceptron(weights:[0.59095, 0.51119],bias:-0.80593,learnRate:(0.15690),activFunc:'naturalOne');iterNumber: 1
initialWeights:[0.55768, 0.82703];Perceptron(weights:[0.16785, 0.82703],bias:-0.89607,learnRate:(0.38983),activFunc:'naturalOne');iterNumber: 4
initialWeights:[0.44677, 0.82911];Perceptron(weights:[1.08580, 0.19007],bias:-1.13711,learnRate:(0.63903),activFunc:'naturalOne');iterNumber: 6
initialWeights:[0.39117, 0.19288];Perceptron(weights:[0.49815, 0.29986],bias:-0.76528,learnRate:(0.05349),activFunc:'naturalOne');iterNumber: 6
initialWeights:[0.14418, 0.78396];Perceptron(weights:[0.14418, 0.78396],bias:-0.79635,learnRate:(0.00887),activFunc:'naturalOne');iterNumber: 1

---------- average iter number ----------

initialWeights:[-0.68320, 0.06877];Perceptron(weights:[0.07552, 0.41364],bias:-0.47807,learnRate:(0.06897),activFunc:'naturalOne');iterNumber: 72
initialWeights:[-0.12307,-0.02144];Perceptron(weights:[0.28277, 0.38441],bias:-0.53646,learnRate:(0.08117),activFunc:'naturalOne');iterNumber: 25
initialWeights:[0.84717,-0.40685];Perceptron(weights:[0.59654, 0.01563],bias:-0.60594,learnRate:(0.00716),activFunc:'naturalOne');iterNumber: 418
initialWeights:[0.13209,-0.31436];Perceptron(weights:[0.50339, 0.13120],bias:-0.58636,learnRate:(0.07426),activFunc:'naturalOne');iterNumber: 25
initialWeights:[0.02483, 0.30130];Perceptron(weights:[0.04921, 0.20377],bias:-0.23136,learnRate:(0.02438),activFunc:'naturalOne');iterNumber: 32
initialWeights:[-0.08147, 0.75077];Perceptron(weights:[0.13155, 0.37799],bias:-0.46029,learnRate:(0.05325),activFunc:'naturalOne');iterNumber: 103
initialWeights:[-0.36895, 0.26485];Perceptron(weights:[0.15907, 0.44085],bias:-0.53193,learnRate:(0.08800),activFunc:'naturalOne');iterNumber: 44
initialWeights:[-0.77574,-0.85672];Perceptron(weights:[0.02537, 0.03339],bias:-0.04101,learnRate:(0.02967),activFunc:'naturalOne');iterNumber: 156
initialWeights:[-0.84950,-0.25015];Perceptron(weights:[0.61523, 1.21458],bias:-1.50577,learnRate:(0.48824),activFunc:'naturalOne');iterNumber: 48
initialWeights:[0.81392,-0.38371];Perceptron(weights:[0.32470, 1.08394],bias:-1.11197,learnRate:(0.48922),activFunc:'naturalOne');iterNumber: 19

---------- max iter number ----------

initialWeights:[-0.15419,-0.52632];Perceptron(weights:[0.34978, 0.00358],bias:-0.35259,learnRate:(0.00162),activFunc:'naturalOne');iterNumber: 1330
initialWeights:[-0.00206,-0.97350];Perceptron(weights:[0.19739, 0.00197],bias:-0.19772,learnRate:(0.00168),activFunc:'naturalOne');iterNumber: 2407
initialWeights:[-0.12709, 0.57907];Perceptron(weights:[0.00071, 0.32900],bias:-0.32969,learnRate:(0.00061),activFunc:'naturalOne');iterNumber: 2516
initialWeights:[-0.89681, 0.51395];Perceptron(weights:[0.00452, 0.49628],bias:-0.49976,learnRate:(0.00177),activFunc:'naturalOne');iterNumber: 2159
initialWeights:[-0.67283, 0.99432];Perceptron(weights:[0.00130, 0.55165],bias:-0.55284,learnRate:(0.00067),activFunc:'naturalOne');iterNumber: 6683
initialWeights:[-0.17183,-0.81148];Perceptron(weights:[0.16453, 0.00265],bias:-0.16616,learnRate:(0.00137),activFunc:'naturalOne');iterNumber: 2365
initialWeights:[0.79868,-0.77361];Perceptron(weights:[0.73151, 0.00323],bias:-0.73398,learnRate:(0.00172),activFunc:'naturalOne');iterNumber: 1953
initialWeights:[-0.51251,-0.44361];Perceptron(weights:[0.01784, 0.08675],bias:-0.10459,learnRate:(0.00000),activFunc:'naturalOne');iterNumber: 19319407
initialWeights:[-0.16550,-0.96391];Perceptron(weights:[0.03108, 0.00691],bias:-0.03615,learnRate:(0.00302),activFunc:'naturalOne');iterNumber: 1356
initialWeights:[0.34524,-0.97219];Perceptron(weights:[0.32921, 0.00012],bias:-0.32932,learnRate:(0.00006),activFunc:'naturalOne');iterNumber: 65743

Dane uczące były wybierane losowo z danych przedstawionych na początku wypisu. Po każdorazowym „nauczeniu” perceptronu, był on testowany na wszystkich możliwych przypadkach danych uczących i sprawdzany, czy zwraca poprawny wynik, jeśli wynik nie był poprawny chociaż w jednym przypadku, perceptron był uczony jeszcze raz, jednak dopiero po przejściu przez wszystkie dane uczące – co stanowi przejście jednej epoki.

Wagi początkowe były dobierane losowo, jako wartość dodatnia lub ujemna, bias był losową wartością ujemną.

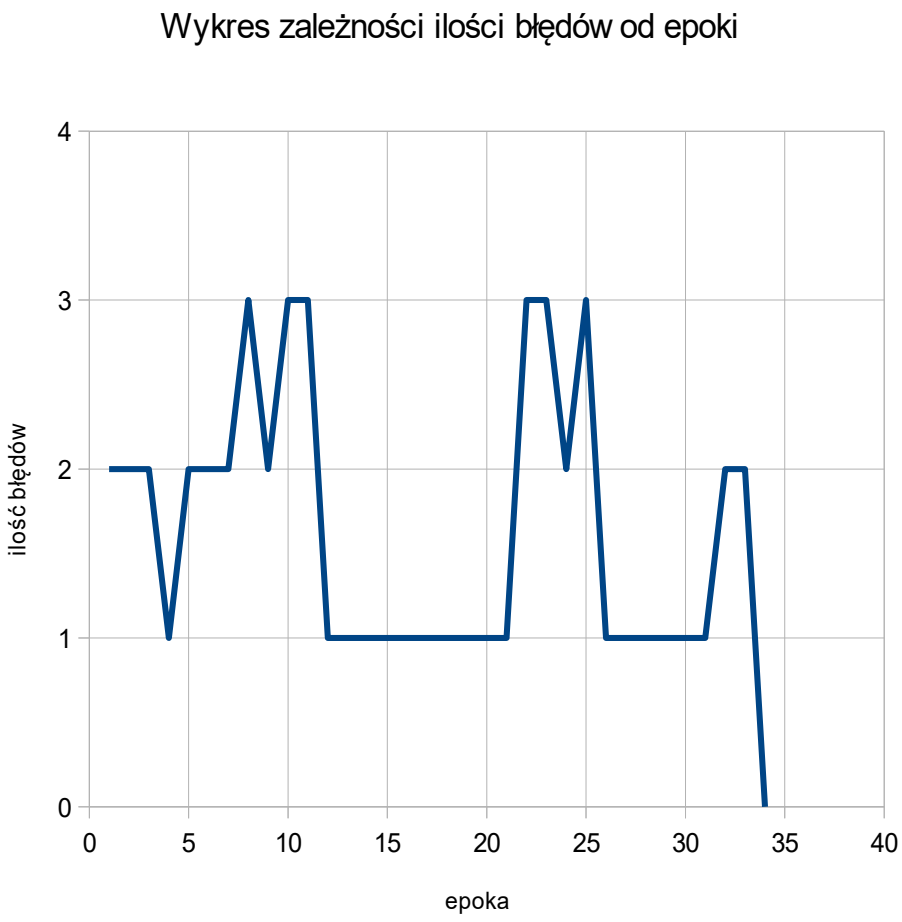
Jak widać w załączonych wynikach istnieje zależność między różnymi współczynnikami uczenia (losowa wartość dodatnia z zakresu [0,1)), a ilością potrzebnych iteracji do osiągnięcia zadowalającego wyniku – im mniejszy „learnRate” tym większa ilość iteracji i dłuższy czas

obliczeń. Udało się w tym przebiegu nawet ustalić na tyle niski „learnRate”, który sprawił, że pętla ucząca musiała zmieniać stan wag ponad 19mln razy.

Poniżej przedstawiam wykres zależności ilości popełnionych błędów w zależności od przejścia przez kolejną epokę, oraz wartości początkowe i końcowe wag i bias, współczynnik uczenia i użytą funkcję aktywacji.

Perceptron(weights:[-0.90233,-0.77122],bias:-0.10800,learnRate:0.24508,activFunc:hardOne)

epoka	ilość błędów
1	2
2	2
3	2
4	1
5	2
6	2
7	2
8	3
9	2
10	3
11	3
12	1
13	1
14	1
15	1
16	1
17	1
18	1
19	1
20	1
21	1
22	3
23	3
24	2
25	3
26	1
27	1
28	1
29	1
30	1
31	1
32	2
33	2
34	0



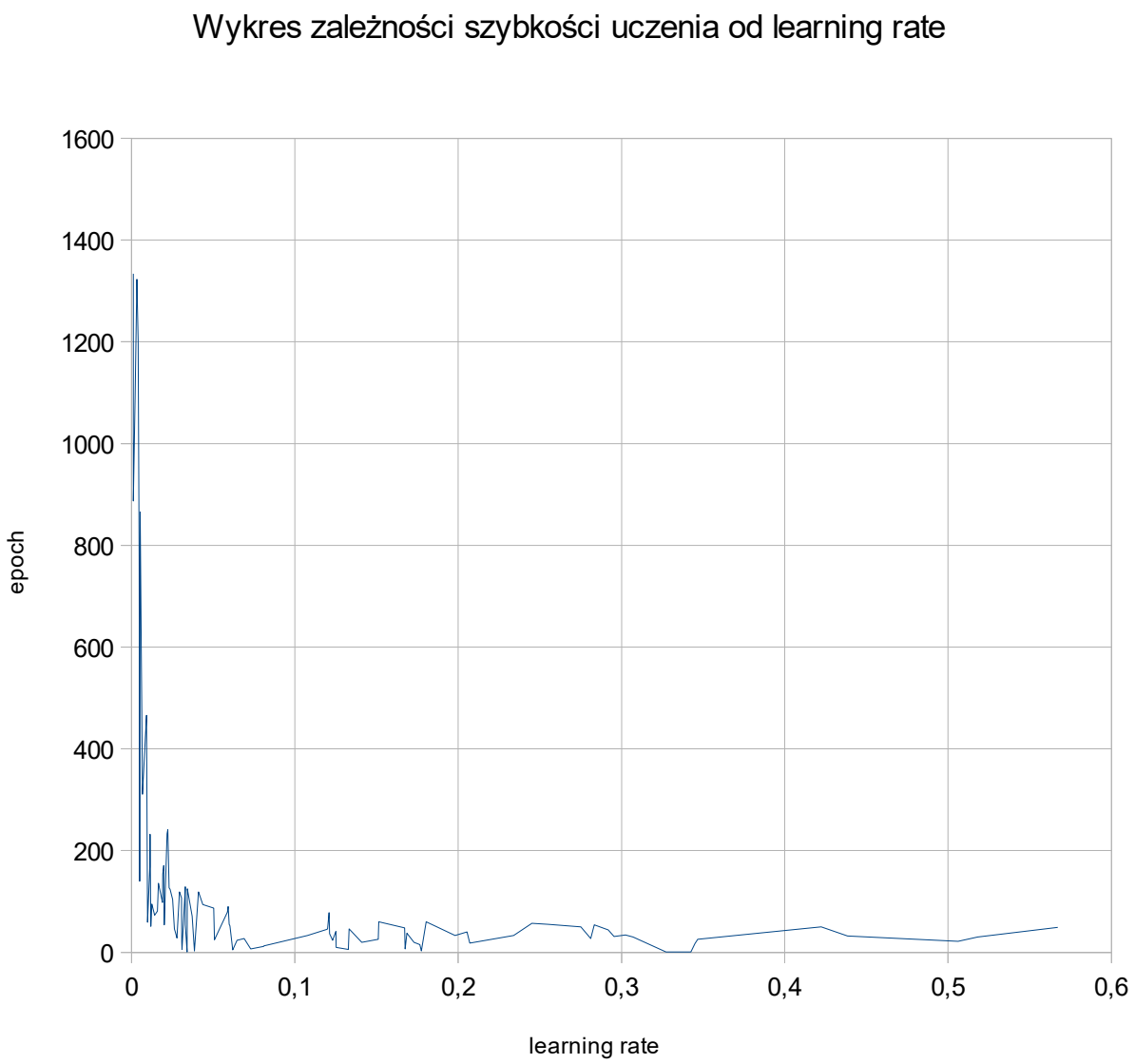
Perceptron(weights:[0.32305, 0.20909],bias:-0.35307,learnRate:0.24508,activFunc:hardOne)

Kolejne zestawienie bardziej skupia się na zależności ilości potrzebnych iteracji do nauczenia perceptronu skutecznie funkcji AND, w zależności od użytego współczynnika „learnRate”:

WYNIKI:

initial weights	end weights	learning rate	epoch number
[0,54721853690063338, 0,1451940741032407]	[0,62898904 0,22696458]	1,20E-004	2700
[-0,23254900090841701, 0,3514337852612831]	[0,00020478 0,1930849]	0,0006359393	2442
[-0,12090499713029035, -0,79189757848013909]	[0,03554447 0,00145264]	0,0010093514	3186
[0,032474685050656427, -0,19421912896424864]	[0,37644295 0,14974913]	0,0010267709	1334
[0,58004921186923419, 0,88510038503860944]	[0,4774281 0,63023915]	0,0011277046	887
[-0,095357357761497452, -0,57599423289042928]	[0,29948331 0,00529898]	0,0021935593	1091
[0,020590691142345019, -0,96076407020535626]	[0,26596943 0,00800395]	0,0031867369	1323
[0,68962572557704971, -0,8749485840889657]	[0,46710826 0,00702972]	0,004045772	1202
[-0,40413223759208705, -0,92572975655833245]	[0,04129672 0,00645661]	0,0045920511	768
[0,14034861997729864, -0,06998731758098042 4]	[0,09141915 0,01319279]	0,0048929475	140
[-0,35807912378898099, -0,97056303769171104]	[0,16899532 0,00372609]	0,0053239843	866
[-0,40214121382263346, 0,16692101900431611]	[0,01362024 0,11998021]	0,0067058298	311
[-0,93883558731587413, -0,27644064945679269]	[0,0134342 0,26771371]	0,009069236	466
[0,20698096874312999, 0,074756389687427816]	[0,34270357 0,21047899]	0,0096944717	59
[-0,34179386393231315, 0,02105800176938788]	[0,01412929 0,0544258]	0,0111225987	169
[-0,66823922666161333 3, -0,030521740692735833]	[0,00693494 0,40834147]	0,0112529028	232
[-0,09904899415055112 8, 0,3375942981384813]	[0,00745195 0,42042837]	0,0118334383	51
[-0,17030607893591143, -0,093452377850482771]	[0,14081768 0,21767138]	0,0124449504	95
[0,52808714268909718, 0,84896954841381378]	[0,35916445 0,51112417]	0,0140768908	73
[0,82478508226494684, 0,29924926531021334]	[0,49250174 0,2359572]	0,0158230165	81
[0,04276701441080466, -0,39233192694897601]	[0,09199807 0,03433724]	0,0164103526	136
[0,86770706552553678, -0,21131666450806108]	[0,81117031 0,05252153]	0,0188455853	98
[-0,26010428839216571, 0,96813504361884528]	[0,02576596 0,66320678]	0,0190580163	154
[-0,7054145561347458, -0,20300618729629327]	[0,02234426 0,19037696]	0,0196691571	171
[-0,00405484614163442 72, -0,099157377222325649]	[0,01595652 0,04092222]	0,0200113706	54
[-0,98866547840803909, -0,01293987615402825 1]	[0,03249411 0,31296212]	0,0217267997	233
[0,68872907480938128, -0,70287843221686563]	[0,46702222 0,02875418]	0,0221706853	241
[0,69658991017707761, -0,41247895883815799]	[0,44449395 0,04587733]	0,0229178144	127
[-0,62216205260028945, 0,44218247619124385]	[0,03709928 0,58345276]	0,0235450474	124
[-0,56953715449614706, 0,047256299148290659]	[0,03309169 0,44900886]	0,0251095352	104
[0,21385831019125534, -0,05267907492966217 3]	[0,50260077 0,23606339]	0,026249315	46
[0,5358534708330277, 0,76930377876930867]	[0,3755466 0,50212566]	0,0267178114	41
[0,40893716737166019, 0,21130979259441984]	[0,21445761 0,1557442]	0,0277827938	28
[-0,5109002078567253, 0,38211613295386027]	[0,04558669 0,20638343]	0,0292887843	119
[-0,67898816164996834, -0,56928634937116462]	[0,02182119 0,04011309]	0,0304699719	107
[-0,044110735297954506, 0,022191548001541328]	[0,20118751 0,26748979]	0,0306622802	21
[0,35532623440886957, 0,23101635983501456]	[0,44777551 0,32346563]	0,0308164238	6
[-0,80247546787274915, 0,5121167229810123]	[0,01885709 0,74208984]	0,0328533024	129
[-0,27642555593376061, 0,90433590050745782]	[0,05223432 0,9043359]	0,032865988	54
[0,25586500558506942, 0,47420487101182185]	[0,25586501 0,44028554]	0,0339193323	1
[-0,45276754972991162, -0,73554010540294057]	[0,05886509 0,04896327]	0,0341088425	125
[0,044019359006565062, -0,54573709972656403]	[0,22953911 0,04792609]	0,0371039496	71
[0,79746012511701869, 0,24055441906517272]	[0,68194784 0,24055442]	0,0385040961	3
[-0,84143791978000215, 0,054602185093682087]	[0,05851098 0,38185633]	0,0409067683	119
[-0,53740408063986767, 0,73155908916149559]	[0,11545033 0,47041732]	0,0435236274	94
[0,841247195210444, -0,38942523393075879]	[0,53991103 0,1128017]	0,0502226934	87
[0,13050967832311866, -0,27926617237907847]	[0,18130086 0,0254809]	0,0507911795	25
[0,77247306436820962, -0,63580299189661571]	[0,47939446 0,06758566]	0,0586157212	79
[0,11656703091720355, -0,66346664086501472]	[0,11656703 0,16383456]	0,0590929432	90
[-0,71964217475376102, -0,10645682688331737]	[0,11639812 0,37128049]	0,0597171642	55
[0,80854387406267547, -0,45223806602521777]	[0,80854387 0,08962611]	0,0602071306	52
[0,48111484723204678, 0,04632970224790777 8]	[0,54297824 0,17005649]	0,0618633917	5
[0,72016339367864113, 0,93492202628750665]	[0,46211609 0,54785107]	0,0645118258	24
[0,80909909773318534, -0,04313216755698046 1]	[0,39568119 0,02577082]	0,0689029847	27
[0,49333567688556468, 0,30734135021077735]	[0,27463752 0,23444197]	0,0728993847	7
[0,626206069054431 44, 0,7104051679282517]	[0,46446318 0,54866228]	0,0808714449	12
[0,58405246495316676, 0,27683653773637207]	[0,42207915 0,27683654]	0,0809866566	13
[0,45509595884487819, -0,346733800248897]	[0,45509596 0,0842372]	0,1077427507	33
[-0,28434889264724128, -0,72895722172928812]	[0,19363517 0,22701091]	0,1194960161	45
[0,41226147612282682, -0,80891221410127212]	[0,53224824 0,27096864]	0,1199867613	46
[-0,80895927655536981, 0,65002381599073777]	[0,15808156 0,2873835]	0,1208801042	78
[-0,48626381976514876, -0,49707599335347119]	[0,24082406 0,10883057]	0,1211813134	37
[-0,40558666988965242, -0,00915527334293131 38]	[0,08667923 0,23697767]	0,1230664739	24
[-0,89345099971624264, -0,27529322331396999]	[0,10724904 0,09996929]	0,1250875053	41
[0,44619884169486757, 0,89060928401587414]	[0,19559239 0,51469961]	0,1253032235	10
[0,07051834150691194 1, -0,11119615430187335]	[0,33606195 0,15434746]	0,1327718061	6
[0,075362316448345124, -0,60629870158792631]	[0,20858751 0,19305247]	0,1332251945	46
[-0,06845305277153090 9, 0,6746346381299011]	[0,35443828 0,67463464]	0,1409637767	20
[-0,39918874002314486, 0,20984870949347434]	[0,3562802 0,3609425]	0,1510937889	26
[-0,86624006283468502, 0,16357631744336498]	[0,19375196 0,16357632]	0,1514274318	60
[-0,31893843187717585, -0,71999206621468226]	[0,18259476 0,11589658]	0,1671777299	48
[0,05687635220412257 1, 0,21472981498163268]	[0,22432316 0,38217663]	0,1674468114	7
[0,63454572545614329, -0,59071481114157509]	[0,4659169 0,2524293]	0,1686288224	38
[-0,46833783265268758, 0,78353378521232064]	[0,22343816 0,61058979]	0,1729439977	20
[0,17528777225510728, 0,75811256178539954]	[0,35019362 0,58320671]	0,1749058513	17
[0,34452541842699169, 0,72752772873393445]	[0,34452542 0,37459169]	0,1764680213	15
[0,013825242436434571, 0,3898192972969301]	[0,19134127 0,56733532]	0,177516024	3
[-0,85906439628365405, 0,37245742509222979]	[0,22273959 0,19215676]	0,1803006643	60
[0,43312036320730907, -0,06386863993425606 6]	[0,23508483 0,33220244]	0,1980355376	33
[0,74717740430507162, -0,88157328891034159]	[0,33646705 0,35055777]	0,2053551765	40
[-0,15703998968365307, -0,05804851944044109 3]	[0,46421998 0,14903814]	0,2070866582	18
[-0,83167586293955 12, 0,38269348375903245]	[0,33819611 0,61666788]	0,2339743943	33
[-0,13400763423122 43, 0,43593302290820135]	[0,35636798 0,19074521]	0,2451878088	57
[-0,84388822023615429, 0,56182526941024391]	[0,43170508 0,56182527]	0,25511866	55
[-0,37686907907180 9, -0,52815119981034908]	[0,17320414 0,57199525]	0,275036612	50
[0,36552887215140561, -0,54466506451643071]	[0,64674947 0,29899673]	0,2812205981	27
[-0,62212918438419362, 0,32592218502922443]	[0,79427789 0,32592219]	0,2832814149	54
[-0,38200106569712378, -0,72675575947876181]	[0,49360278 0,44071604]	0,291867949	44
[-0,65457872507843562, 0,21597384871361 88]	[0,52711144 0,21597385]	0,2954225402	31
[0,76926105950720558, -0,84181759364462017]	[0,76926106 0,36799684]	0,3024536093	34
[0,10812491132211244, -0,92355048132653217]	[0,4151738 0,30464505]	0,3070488839	30
[0,68217124754789227, 0,93809774112025135]	[0,68217125 0,61083153]	0,3272662154	1
[0,23083694748285599, 0,74101245670533122]	[0,23083695 0,74101246]	0,3424489481	1
[0,27572782921900196, -0,34113577158496566]	[0,27572783 0,34867379]	0,3449047795	17
[-0,45056586738497584, 0,70829822693882427]	[0,58941868 0,36163671]	0,3466615148	26
[0,028467894841881503, -0,58323240222938044]	[0,45071078 0,68349627]	0,4222428895	50
[0,57384137575734639, 0,52751837318202788]	[1,01232123 0,52751837]	0,4384798496	32
[-0,96865960064353063, 0,73731360298900028]	[0,54971499 0,7373136]	0,5061248651	22
[0,4383892354446034, -0,37964427121120659]	[0,95654335 0,65666396]	0,5181541167	30
[0,23199245320889161, -0,20017997093281203]	[0,799224 1,50151466]	0,5672315448	49

Co na wykresie przedstawia się w ten sposób:



Widać wyraźnie, że poniżej pewnej granicy learnRate równej ok. 0,01 szybkość uczenia zaczyna spadać. Natomiast powyżej ok. 0.05 współczynnik uczenia nie ma większego wpływu na szybkość uczenia.

4. Wnioski

- Perceptron to bardzo przydatna implementacja funkcji matematycznej, która może być użyta w bardzo wieloraki sposób np. do realizacji funkcji AND, jak i innych prostych funkcji matematycznych, które później w większej sieci mogą realizować coraz to bardziej złożone funkcje matematyczne.
- Perceptron jest bardzo prosty w działaniu i realizacji programowej.
- W zależności od użytego współczynniku uczenia możemy uzyskać mniejszą lub większą szybkość uczenia perceptronu

Ponad wymagania na dane ćwiczenia stworzyłem implementację programową funkcji i klas umożliwiającą zaprojektowanie całej sieci neuronowej i zrealizowałem w niej funkcję XOR.

Listing części kodu:

```
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 10 19:39:26 2017

@author: PiotrTutak
"""
import numpy as np
from itertools import zip_longest
from operator import itemgetter
import random
import sys
"""
Różne funkcje aktywacji używane w testowaniu perceptronu:
"""
def one(x):
    return 1.0

def hardOne(x):
    if x<0:
        return 0.0
    return 1.0

#(...)

class Perceptron:
    """
    Klasa Perceptron
    """
    def __init__(self, weights, activFunc, activFuncDeriv, learnRate=0.1, bias=-0.8*np.random.ranf()-0.1):
        self.__dict__['_weights']=np.array(weights)
        self.__dict__['_learnRate']=learnRate
        self.__dict__['_activFunc']=activFunc
        self.__dict__['_activFuncDeriv']=activFuncDeriv
        self.__dict__['_bias']=bias
        self.__dict__['_error']=None
        self.__dict__['_inputValues']=None
        self.__dict__['_val']=None

    def process(self,inputValues):
        """
        Funkcja przetwarzająca dane wejsciowe na dane wyjsciowe
        """
        if len(inputValues)!=len(self._weights):
            raise TypeError('Wrong values length')
        self.__dict__['_inputValues']=np.array(inputValues)
        self.__dict__['_val']=np.dot(self._weights,self._inputValues)+self._bias
        return self._activFunc(self._val)

    def propagateError(self,weights,errors):
        """
        Funkcja propagująca błąd i korygująca wagi oraz bias
        """
        weights=np.array(weights)
        errors=np.array(errors)
        if len(errors)!=len(weights):
            raise TypeError('Wrong values length')
        self.__dict__['_error']=np.dot(weights,errors)*self._activFuncDeriv(self._val)
```

```
        if (self._learnRate):
            for i in range(len(self._weights)):
                self._weights[i]+=self._learnRate*self._error*self._inputValues[i]
            self.__dict__['_bias']+=self._learnRate*self._error
        return self._error
    """
    Funkcje dostępowe:
    """
    def __setitem__(self,index,value):
        if index=='learnRate':
            self.__dict__['_learnRate']=value

    def __getitem__(self,index):
        if index=='error':
            return self._error
        elif index=='input':
            return self._inputValues
        elif index=='value':
            return self._val
        elif index=='learnRate':
            return self._learnRate
        return self._weights[index]

    def __getattr__(self,attr):
        raise AttributeError('get: No such attribute: %r'%attr)

    def __setattr__(self,attr,value):
        raise AttributeError('set: No such attribute: %r'%attr)

    def __iter__(self):
        return iter(self._weights)

    def __len__(self):
        return len(self._weights)

    def __repr__(self):
        w='['+','.join('{:8.5f}'.format(x) for x in self._weights)+']'
        return 'Perceptron(weights:{0},bias:{1:8.5f},learnRate:{2:.5f},activFunc:{3!
s})'.format(w,self._bias,self._learnRate,self._activFunc.__name__)

#(...)

if __name__=='__main__':
    """
    Kod programu przeprowadzającego uczenie i testowanie perceptronu
    Wyjscie jest przekierowywane do pliku results.txt
    """
    STDOUT=sys.stdout
    f=open('results.txt','w');
    sys.stdout=f

    print('Funkcja AND:')
    inputData=(
          ((0,0),0),
          ((0,1),0),
          ((1,0),0),
          ((1,1),1)
          )
    for x in inputData:
```

```
        print("data: {0}, expected: {1}".format(*x))

    listPerc=[]
    RES_NUMBER=100
    while(len(listPerc)<RES_NUMBER):
        w=[np.random.ranf()*np.random.choice([-1,1]) for _ in range(2)]

p=Perceptron(w,hardOne,one,learnRate=np.random.ranf()*np.random.ranf()*np.random.ranf(),bias=np.random.ranf()
*-1.0)
        i=0
        run=True
        print(p)
        print('iteration;bad')
        while(run):
            samples=list(inputData)
            while(run and samples):
                i+=1
                inp=random.sample(samples,1).pop(0)
                expected=inp[1]
                result=p.process(inp[0])
                p.propagateError([1],[expected-result])
                bad=0
                for data,expected in inputData:
                    r=p.process(data)
                    if r!=expected:
                        bad+=1
                if bad==0:
                    run=False
                print(i,bad,sep=';')
        print(p)

        #w=('initialWeights:['+','.join('{:8.5f}'.format(x) for x in w)+']')

        listPerc.append((w,p[:],p['learnRate'],i))
    for x in sorted(listPerc,key=itemgetter(2)):
        print(*x,sep=';')
    sys.stdout=STDOUT
    f.close()
```