НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра прикладної математики

*Звіт*

*із лабораторної роботи №1-3*

*із дисципліни «Методи штучного інтелекту»*

*на тему*

*Розробка програмного забезбепечення для реалізації двошарового*

*персептрону х сигмоїдальною функцією активації*

|  |  |
| --- | --- |
| Виконав: | Керівник: |
| студентки групи КМ-81 | Доцент |
| *Верзун П. В.* | Терейковська Л.О. |

Київ — 2021

**Зміст**

[**Постановка задачі**](#_zfjl9hdpxc7o) **3**

[**Короткі теоретичні відомості**](#_tqcqjv38h9xn) **4**

[**Результат роботи програми**](#_vvk8gcp26g0t) **7**

[**Висновки**](#_otm11ky3mm47) **8**

# **Постановка задачі**

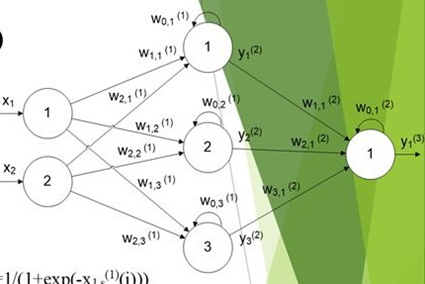
Реалізувати програмне забезпечення для реалізації двошарового

персептрону із структурою 2-3-1. Передбачити режим навчання ONLINE

та режим розпізнавання. Піддослідна функція x+y=z

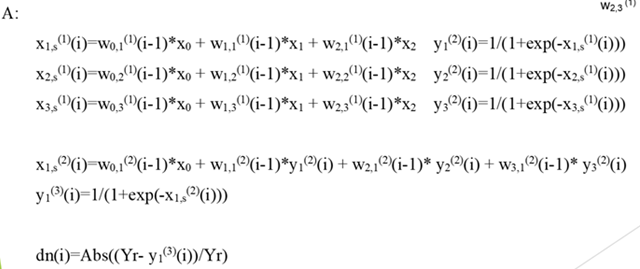
# **Короткі теоретичні відомості**

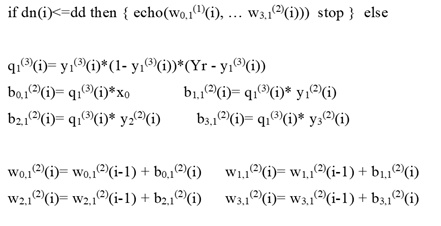
Схема персептрону 2-3-1 представлена нижче:

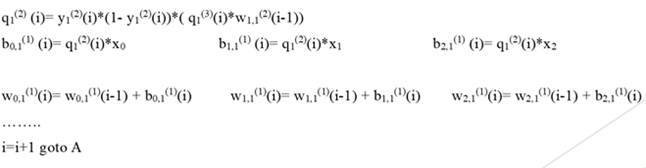


Алгоритм навчання двошарового персептрону із структурою 2-3-1 має

наступний вигляд:

****

****

****

**Код програми з коментарями**

def calculate\_f(x):

y\_m = 1 / (1 + (1 / math.e \*\* x))

return y\_m

def calculate\_net(x, w):

net = sum(x \* w)

return net

def calculate\_error(y\_m, y):

error = abs((y\_m - y) / y)

return error

def calculate\_new\_w(y\_m, y, w, x):

sigma = y\_m \* (1 - y\_m) \* (y - y\_m)

w\_diff = x \* sigma

w\_new = w + w\_diff

return w\_new

def calculate\_rand\_w(n):

w\_rand = np.random.uniform(0, 1, n)

w = []

for i in range(n):

w.append(w\_rand[i])

return np.array(w)

def recognition(x, w\_1\_2, w\_2\_3, w\_0\_1, w\_0\_2):

total\_1 = np.array(x).dot(w\_1\_2) + w\_0\_1

y\_m\_1 = np.array(list(map(calculate\_f, np.array(total\_1))))

total\_2 = y\_m\_1.dot(w\_2\_3) + w\_0\_2

y\_m\_2 = calculate\_f(total\_2[0][0])

return y\_m\_2

def recognition\_func(x, w\_12, w\_01, w\_23, w\_02):

total\_1 = x.dot(w\_12)

# vector x\*w of the first hidden layer values

total\_1 = total\_1 + w\_01

# vector y\_molule of the first hidden layer

y\_m\_1 = np.array(list(map(calculate\_f, total\_1)))

total\_2 = y\_m\_1.dot(w\_23) + w\_02

# final y\_molule of the first hidden layer

y\_m\_2 = calculate\_f(total\_2[0][0])

return y\_m\_2

def find\_new\_w(x, y, y\_m\_2, y\_m\_1, w\_12, w\_01, w\_23, w\_02):

q\_3\_2 = y\_m\_2 \* (1 - y\_m\_2) \* (y - y\_m\_2)

b\_3\_2 = q\_3\_2 \* y\_m\_1

b\_3\_2\_0 = x0 \* q\_3\_2

for j in range(len(b\_3\_2[0])):

w\_23[j][0] = w\_23[j][0] + b\_3\_2[0][j]

w\_02 = w\_02 + b\_3\_2\_0

q\_2\_1 = []

for k in range(len(y\_m\_1[0])):

q\_2\_1.append([y\_m\_1[0][k] \* (1 - y\_m\_1[0][k]) \* (q\_3\_2 \* w\_23[k][0])])

b\_2\_1 = (np.array(q\_2\_1)).dot(np.matrix([x0, x[0], x[1]])).transpose()

w\_12 = w\_12 + b\_2\_1[1:][:]

w\_01 = w\_01 + b\_2\_1[0][:]

return w\_01, w\_12, w\_02, w\_23

def learn\_alg(x, y, w\_12, w\_01, w\_23, w\_02):

total\_1 = x.dot(w\_12)

# vector x\*w of the first hidden layer values

total\_1 = total\_1 + w\_01

# vector y\_molule of the first hidden layer

y\_m\_1 = np.array(list(map(calculate\_f, total\_1)))

total\_2 = y\_m\_1.dot(w\_23) + w\_02

# final y\_molule of the first hidden layer

y\_m\_2 = calculate\_f(total\_2[0][0])

error = calculate\_error(y\_m\_2, y)

w01, w12, w02, w23 = find\_new\_w(x, y, y\_m\_2, y\_m\_1, w\_12, w\_01, w\_23, w\_02)

return y\_m\_2, error, np.array(w12), np.array(w01), w23, w02

def online\_learning(X, Y, n\_epohs):

w\_1\_2 = [calculate\_rand\_w(3), calculate\_rand\_w(3)]

w\_0\_1 = [calculate\_rand\_w(3)]

w\_2\_3 = [calculate\_rand\_w(1), calculate\_rand\_w(1), calculate\_rand\_w(1)]

w\_0\_2 = calculate\_rand\_w(1)

n = 0

while n <= n\_epohs:

y\_model\_n = []

error\_n = []

for i in range(len(Y)):

y\_m\_2, error, w\_1\_2, w\_0\_1, w\_2\_3, w\_0\_2 = learn\_alg(

X[i], Y[i], w\_1\_2, w\_0\_1, w\_2\_3, w\_0\_2

)

y\_model\_n.append(y\_m\_2)

error\_n.append(error)

if n % 100 == 0:

print("\n Epoch #{}".format(n))

print("Y modeled: {}".format(y\_model\_n))

print("Delta {}".format(error\_n))

n += 1

return w\_1\_2, w\_0\_1, w\_2\_3, w\_0\_2

if \_\_name\_\_ == "\_\_main\_\_":

print("Task3")

x0 = 1

y = 0.03

E = 0.1

x = np.array([1.0, 1.5]) # input vector

w\_1\_2 = [calculate\_rand\_w(3), calculate\_rand\_w(3)] # x1, x2 weights

w\_0\_1 = [calculate\_rand\_w(3)] # weights

w\_2\_3 = [

calculate\_rand\_w(1),

calculate\_rand\_w(1),

calculate\_rand\_w(1),

] # last layer weights

w\_0\_2 = calculate\_rand\_w(1) # last neuron

total\_1 = x.dot(w\_1\_2) # weighted sum

# vector x\*w of the first hidden layer values

total\_1 = total\_1 + w\_0\_1

# vector y\_molule of the first hidden layer

y\_m\_1 = np.array(list(map(calculate\_f, total\_1)))

total\_2 = y\_m\_1.dot(w\_2\_3) + w\_0\_2

# final y\_molule of the first hidden layer

y\_m\_2 = calculate\_f(total\_2[0][0])

error = calculate\_error(y\_m\_2, y)

df = pd.DataFrame(columns=["y model", "delta"])

i = 0

while True:

if error <= E:

break

else:

df = df.append({"y model": y\_m\_2, "delta": error}, ignore\_index=True)

q\_3\_2 = y\_m\_2 \* (1 - y\_m\_2) \* (y - y\_m\_2)

b\_3\_2 = q\_3\_2 \* y\_m\_1

b\_3\_2\_0 = x0 \* q\_3\_2

for j in range(len(b\_3\_2[0])):

w\_2\_3[j][0] = w\_2\_3[j][0] + b\_3\_2[0][j]

w\_0\_2 = w\_0\_2 + b\_3\_2\_0

q\_2\_1 = []

for k in range(len(y\_m\_1[0])):

q\_2\_1.append([y\_m\_1[0][k] \* (1 - y\_m\_1[0][k]) \* (q\_3\_2 \* w\_2\_3[k][0])])

b\_2\_1 = (np.array(q\_2\_1)).dot(np.matrix([x0, x[0], x[1]])).transpose()

w\_1\_2 = w\_1\_2 + b\_2\_1[1:][:]

w\_0\_1 = w\_0\_1 + b\_2\_1[0][:]

total\_1 = x.dot(w\_1\_2)

total\_1 = np.array(total\_1 + w\_0\_1)

y\_m\_1 = np.array(list(map(calculate\_f, total\_1)))

total\_2 = y\_m\_1.dot(w\_2\_3) + w\_0\_2

y\_m\_2 = calculate\_f(total\_2[0][0])

error = calculate\_error(y\_m\_2, y)

i = i + 1

print(w\_0\_2)

print("n iterations = " + str(i))

print("y\_m = " + str(y\_m\_2))

print("\n")

print("w\_1 = " + str(w\_1\_2))

print("\n")

print("w\_2 = " + str(w\_2\_3))

print("\n")

print(df)

print("\n")

print("Detection")

print("Example: " + str([1.1, 1.4]))

print(

"y\_m = " + str(recognition([1.1, 1.4], w\_1\_2, w\_2\_3, w\_0\_1=w\_0\_1, w\_0\_2=w\_0\_2))

)

print("Online Learning")

X = np.array([[1.0, 1.0], [2.0, 1.0], [1.5, 1.5], [1.0, 2.5]]) # input vector

Y = np.array([0.02, 0.03, 0.03, 0.035])

x0 = 1

n\_epohs = 1000

w\_1\_2, w\_0\_1, w\_2\_3, w\_0\_2 = online\_learning(X, Y, n\_epohs)

print("\nRecognition part")

print("x = {}".format([1.0, 1.5]))

print(

"y = {}".format(

recognition\_func(np.array([1.0, 1.5]), w\_1\_2, w\_0\_1, w\_2\_3, w\_0\_2)

)

)

Вхідна точка у програмі є запуск функції online\_learning(), де власне і

відбувається тренування нейрону. В ній ми ініціалізуємо ваги рандомним

чином. Після чого запускається цикл на 1000 епох, де кожну епоху ми

тренуємо нашу сітку та перераховуємо ваги за вже відомими нам

алгоритмам. Кожну 100 епоху ми виводимо на екран У та дельту. Після

цього ми маємо натреновану сітку і можемо запускати режим

розпізнавання. Власне режим розпізнавання представляє собою функція

recognition\_func(), яка видає нам у.

# **Результат роботи програми**

[-0.97812956]

n iterations = 574

y\_m = 0.03299815411380345

w\_1 = [[0.37919696 1.07223954 0.67074626]

[0.60564374 0.33830649 0.064123 ]]

w\_2 = [array([-0.68331011]), array([-1.61345823]), array([-0.45549513])]

y model delta

0 0.901429 29.047620

1 0.876980 28.232675

2 0.842033 27.067751

3 0.791496 25.383193

4 0.719918 22.997276

...

569 0.033048 0.101605

570 0.033038 0.101269

571 0.033028 0.100935

572 0.033018 0.100601

573 0.033008 0.100269

[574 rows x 2 columns]

Detection

Example: [1.1, 1.4]

y\_m = 0.032693164368052696

Online Learning

Epoch #0

Y modeled: [0.8301231339495962, 0.7893050574741909, 0.7288407053390193, 0.6502052177224799]

Delta [40.50615669747981, 25.31016858247303, 23.29469017796731, 17.57729193492799]

Epoch #100

Y modeled: [0.0394329903839788, 0.03449768812925458, 0.033110094026575165, 0.029456604139596304]

Delta [0.97164951919894, 0.1499229376418193, 0.10366980088583888, 0.1583827388686771]

Epoch #200

Y modeled: [0.034647222741532306, 0.03012244689028706, 0.028924644225214544, 0.0256899519040889]

Delta [0.7323611370766153, 0.004081563009568703, 0.03584519249284849, 0.26600137416888864]

Epoch #300

Y modeled: [0.033344060523196335, 0.028932474329588096, 0.02780086009821247, 0.02469975819186078]

Delta [0.6672030261598167, 0.035584189013730096, 0.07330466339291765, 0.29429262308969206]

Epoch #400

Y modeled: [0.03290003794443209, 0.02852507284206472, 0.027429565044022445, 0.024390545311018565]

Delta [0.6450018972216044, 0.04916423859784268, 0.0856811651992518, 0.3031272768280411]

Epoch #500

Y modeled: [0.032734950705860744, 0.02837143810124075, 0.02730228052778021, 0.02430205039633966]

Delta [0.6367475352930372, 0.05428539662530828, 0.0899239824073263, 0.305655702961724]

Epoch #600

Y modeled: [0.032668445930294915, 0.028307516287828773, 0.02726117062307016, 0.02429144231662262]

Delta [0.6334222965147457, 0.05641612373904088, 0.09129431256432799, 0.3059587909536395]

Epoch #700

Y modeled: [0.032637825102104376, 0.028276322888413842, 0.027251377195818906, 0.02430905111146977]

Delta [0.6318912551052187, 0.05745590371953856, 0.09162076013936975, 0.3054556825294352]

Epoch #800

Y modeled: [0.03262040316643375, 0.028257247333977997, 0.027253047898683595, 0.024336888123855583]

Delta [0.6310201583216873, 0.05809175553406672, 0.09156507004388012, 0.30466033931841197]

Epoch #900

Y modeled: [0.032607853423051954, 0.028242724292182077, 0.027258893521522557, 0.024368344138098345]

Delta [0.6303926711525977, 0.05857585692726405, 0.09137021594924806, 0.30376159605433306]

Epoch #1000

Y modeled: [0.03259710414780982, 0.028229961270763503, 0.027266224501536988, 0.02440098031229321]

Delta [0.6298552073904911, 0.059001290974549855, 0.09112584994876705, 0.30282913393447974]

Recognition part

x = [1.0, 1.5]

y = 0.02909372458465547

# **Висновки**

В даній частині лабораторної роботи ми реалізували двошаровий

персептрон із структурою 2-3-1. Більш того, був реалізований метод

навчання ONLINE. Навчання проводилося на наступних прикладах:

x1=[0.5, 0.5], x2=[1.5, 1.5], x3=[1.0, 2.5], x4=[2.5, 1.5]. Для яких очікуваних

результатом є наступне: y1=0.01, y2=0.03, y3=0.035, y4=0.04. Після

проведеного навчання в 1000 епох маємо наступні результати:

у модельне

[0.03259710414780982, 0.028229961270763503, 0.027266224501536988, 0.02440098031229321]

та похибки для них [0.6298552073904911, 0.059001290974549855, 0.09112584994876705, 0.30282913393447974].

Режим розпізнавання проводився на наступних даних: x=[1.0, 1.5]. Після

чого ми отримали досить хороший результат, а саме

y = 0.02909. Що є досить коректно.