دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده مهندسی برق

گزارش آزمایش سوم
مقدمه‌ای بر هوش محاسباتی
دکتر عبداللهی

نام و نام خانوادگی
کیان بهزاد

شماره دانشجویی
۹۵۲۳۰۱۷

پیاده سازی perceptron دو لایه

قصد داریم به پیاده سازی این الگوریتم در محیط **پایتون** بپردازیم و شبکه را برای تشخیص عملیات xor ترین کنیم.
پس در ابتدا می‌بایست تعداد لایه ها و همچنین نورون‌های هر لایه را معین کنیم، پس طبق پیشنهاد lecture از یک شبکه دو لایه که در لایه اول از ۲ نورون استفاده شده استفاده می‌کنیم، در خروجی نیز با توجه به اینکه تنها یک خروجی داریم از یک نورون بهره خواهیم برد.
همچنین با توجه به قضیه universal در نورون‌های لایه اول از sigmoid به عنوان تابع فعال‌ساز و در نورون لایه آخر تابع خطی را به عنوان فعال‌ساز انتخاب می‌کنیم.
به این منظور مطابق با الگوریتم داده شده و بصورت زیر عمل می‌کنیم.

```
import numpy as np
from sigmoid import sigmoid, dSigmoid

error = 0
y = np.array([[1, 1, -1], [1, 0, -1], [0, 1, -1], [0, 0, -1]], dtype=float)
d = np.array([0, 1, 1, 0], dtype=float)
w1 = np.array([[.1, .2, .3], [.2, .3, .1]], dtype=float)
w2 = np.array([[.1, .1]], dtype=float)
a = np.array([0, 0], dtype=float)
counter = 0
step = 0

while True:
    a[0] = sigmoid(np.dot(w1[0], np.transpose(y[counter])))
    a[1] = sigmoid(np.dot(w1[1], np.transpose(y[counter])))
    o = np.dot(w2, a)
    error = error + (o - d[counter]) ** 2
    if counter == 3:
        step += 1
        print("error in {} -> {}".format(step, error))
        if error < .001:
            break
        error = 0
        counter = -1
    F = np.array([[dSigmoid(a[0]), 0], [0, dSigmoid(a[1])]])
    s2 = -0.01 * 1 * (d[counter] - o)
    s1 = np.dot(np.dot(F, np.transpose(w2)), s2)

    w1 = w1 - np.dot(np.transpose(np.array([np.dot(np.dot(F, np.transpose(w2)), s2)])), np.array([y[counter]]))
    w2 = w2 - s2 * a
    counter += 1
```

با استفاده از این الگوریتم، وزن‌های اولیه و گام آموزش بعد از ۲۵۵۲۶۵ بار feed کردن هر ۴ ورودی به سیستم خطای مجموعمان کمتر از ۰.۰۰۱ خواهد شد و نتایج نهایی بصورت زیر پدید خواهد آمد.

```
[ 1.  1. -1.] -> [0.01594805]
[ 1.  0. -1.] -> [0.9849528]
[ 0.  1. -1.] -> [0.98469863]
[ 0.  0. -1.] -> [0.01573593]
```

دقت شود که -۱ در ورودی‌ها به عنوان bias در نظر گرفته شده و ورودی‌های اصلی دو مقدار اول هر آرایه هست

تمرین

۱. مدل شبکه عصبی ۲ لایه پیاده سازی شده در محیط پایتون را دوباره در محیط متلب تکرار کنید.

دقیقا به مانند همان الگوریتم بالا عمل هواهیم کرد.

```
error = 0;
y = [1, 1, -1; 1, 0, -1; 0, 1, -1; 0, 0, -1];
d = [0, 1, 1, 0];
w1 = [.1, .2, .3; .2, .3, .1];
w2 = [.1, .1];
a = [0, 0];
counter = 1;
step = 0;

while true
    a(1) = sigmoid(w1(1,:) * transpose(y(counter,:)));
    a(2) = sigmoid(w1(2,:) * transpose(y(counter,:)));
    o = w2 * a';
    error = error + (o - d(counter)) ^ 2;

    if(counter == 4)
        step = step + 1;
        fprintf('error in %i -> %f\n', step, error);
        if(error < .001)
            break
        end
        error = 0;
        counter = 0;
    end
    counter = counter + 1;

    F = [dSigmoid(a(1)), 0; 0, dSigmoid(a(2))];
    s2 = -0.01 * 1 * (d(counter) - o);
    s1 = F * transpose(w2) * s2;

    w1 = w1 - F * transpose(w2) * s2 * y(counter,:);
    w2 = w2 - s2 * a;
end
```

گزارش و کد را می‌توانید در repository زیر ببینید(Az_jalase2)

https://github.com/kianbehzad/computational-intelligence