بخش سوم:

توابع پیاده سازی شده در کلاس Input

برای تابع اول باید مقدار خود را برگرداند و برای تابع دوم صفر را.

```
def output(self):
    return self.get_value()

def dOutdX(self, elem):
    return 0
```

توابع پیاده سازی شده در کلاس Performance Element

تابع اول دقیقا مانند محاسبه خطا میباشد با این فرق که فرمول آن در یک منفی ضرب میشود.

```
def output(self):
    return -(1 / 2.0) * (self.my_desired_val - self.my_input.output()) ** 2

def dOutdX(self, elem):

    return (self.my_desired_val - self.get_input().output()) *
self.get_input().dOutdX(elem)
```

توابع پیاده سازی شده در Neuron

تابع سیگموید برای این استفاده میشود که در همه نقاط قابل تشخیص است (یعنی با داشتن دو نقطه میتوان شیب را حدس زد) و کار را ساده میکند و بین مقادیر ۰ و ۱ وجود دارد و برای توابعی که میخواهیم احتمال آن را برای خروجی بدهیم بسیار مناسب است.

```
def compute_output(self):
    z = 0
    weights = self.get_weights()
    inputs = self.get_inputs()
    for i in range(len(self.get_inputs())):
        z += weights[i].get_value() * inputs[i].output()
    return 1.0 / (1.0 + math.exp(-z))


def compute_doutdx(self, elem):
    dout = 0
```

```
    sigmoid_deriviation = self.output() * (1 - self.output())
    if self.has_weight(elem):
        dout = self.get_inputs()[self.my_weights.index(elem)].output()
    else:
        for i in range(len(self.get_weights())):
            curr_weight = self.my_weights[i]
            dout = dout + (curr_weight.get_value() * self.get_inputs()[i].dOutdX(
                elem)) if self.isa_descendant_weight_of(elem, curr_weight) else dout
    return dout * sigmoid_deriviation
```

برای دیباگ کردن از تابع زیر استفاده شده است که مقدار مشتق را با قاعده زنجیره ای مقایسه میکند:

```
def finite_difference(network):
    network.clear_cache()
    constant = 10**(-8)
    for weight in network.weights:
        fx = network.performance.output()
        weight.set_value(weight.get_value() + constant)
        network.clear_cache()
        fxd = network.performance.output()
        finite_diff = (fxd - fx) / constant
        weight.set_value(weight.get_value() - constant)
        network.clear_cache()
        res = network.performance.dOutdX(weight) - finite_diff
        if abs(res) > 0.001:
            print(weight.get_name() + " is incorrect")
    network.clear_cache()
```

مقدار دلتای آن ده به توان منفی ۸ میباشد و برای محاسبه f(x) از تابع performance.output() استفاده میکنم و بعد از آن f(x+delta) را حساب کرده و در فرمول مشتق میگذارم و تقسیم میکنم. اگر اختلاف آن ها بیشتر از یک صدم بود وزن اشتباه را چاپ میکنم (به این ارور برنخوردم)

برای شبکه عصبی دولایه همان مراحلی که در شبکه عصبی پایه رفته بودند را رفتم و آن را پیاده سازی کردم.

```
i0 = Input('i0', -1.0)
i1 = Input('i1', 0.0)
i2 = Input('i2', 0.0)

seed_random()
w1A = Weight('w1A', random_weight())
w2A = Weight('w2A', random_weight())
wA = Weight('wA', random_weight())
A = Neuron('A', [i0, i1, i2], [wA, w1A, w2A])

w1B = Weight('w1B', random_weight())
w2B = Weight('w2B', random_weight())
wB = Weight('wB', random_weight())
B = Neuron('B', [i0, i1, i2], [wB, w1B, w2B])

wC = Weight('wC', random_weight())
wAC = Weight('wAC', random_weight())
wBC = Weight('wBC', random_weight())
C = Neuron('C', [i0, A, B], [wC, wAC, wBC])
```

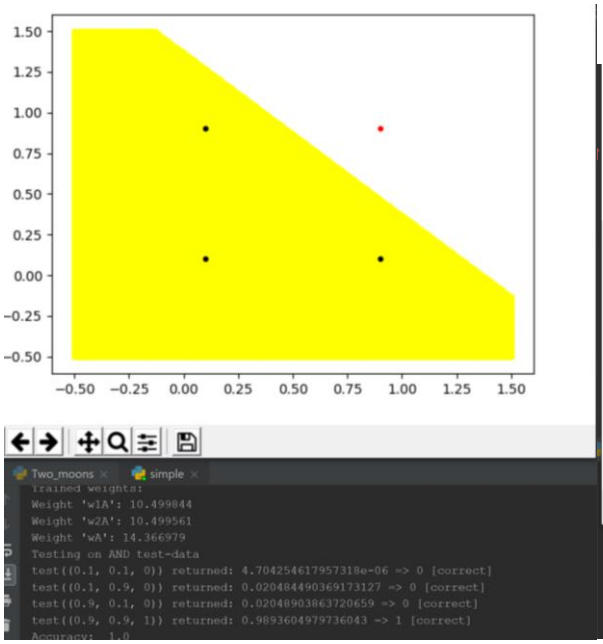
```
P = PerformanceElem(C, 0.0)
net = Network(P, [A, B, C])

return net
```

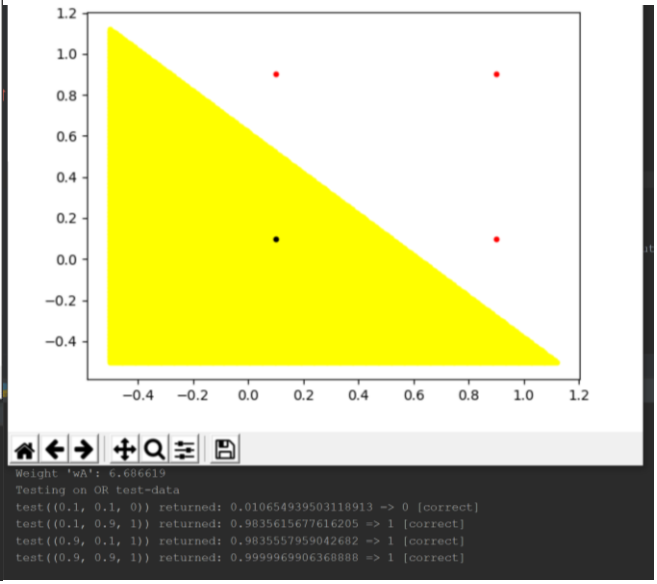
برای کشیدن نمودار دو تابع را پیاده سازی کردم (یکی به عنوان ورودی داده را میگیرد و دیگری نمیگیرد.) در اینجا برای تحلیل بهتر نموداری که داده را میگیرد را آورده ام.

تست شبکه عصبی پایه:

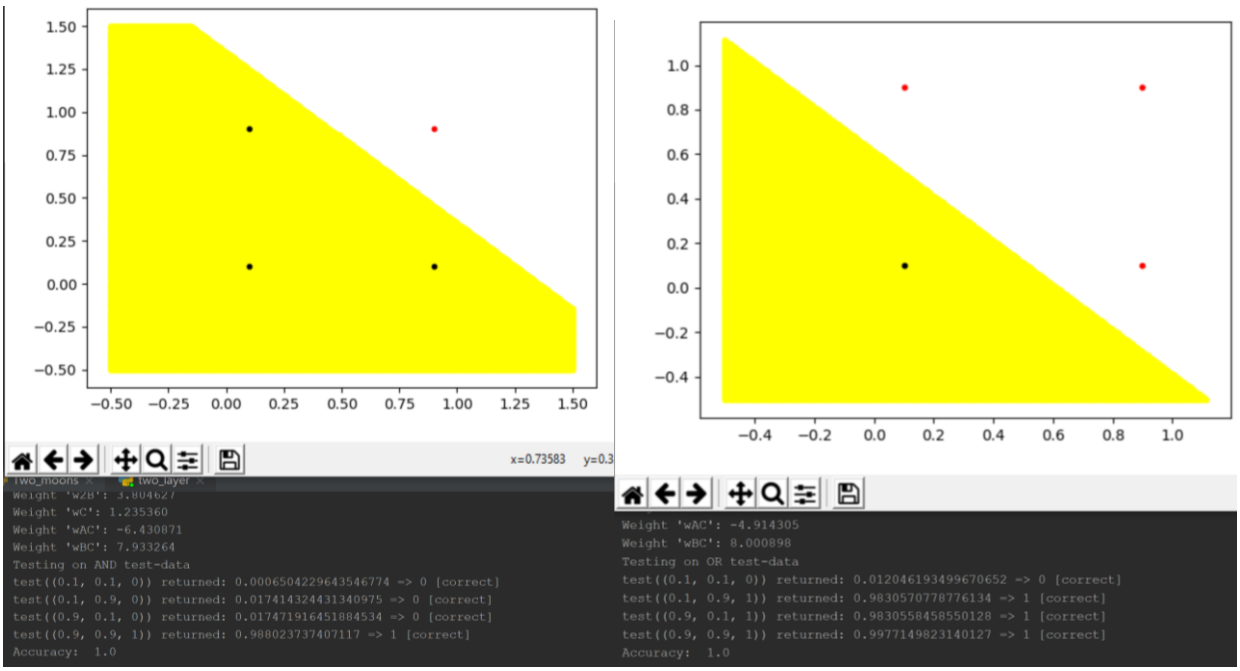
نمودار داده ترین و تست:

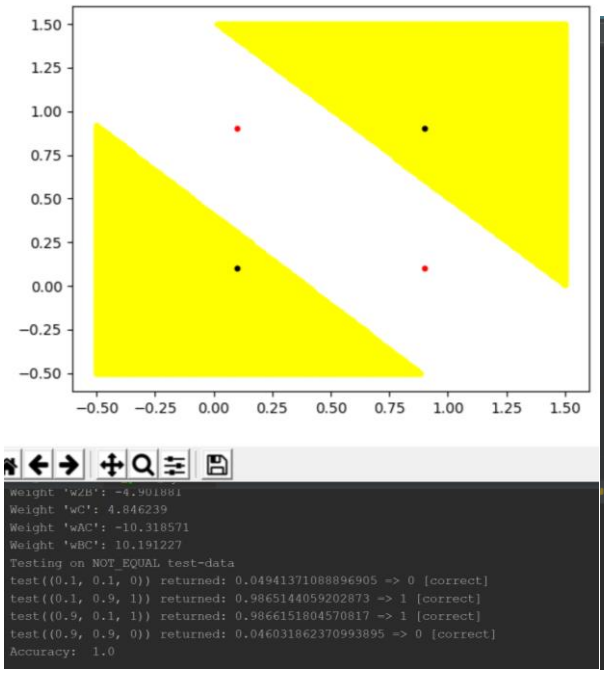
(هر ۴ داده تست را درست تشخیص داده است.)

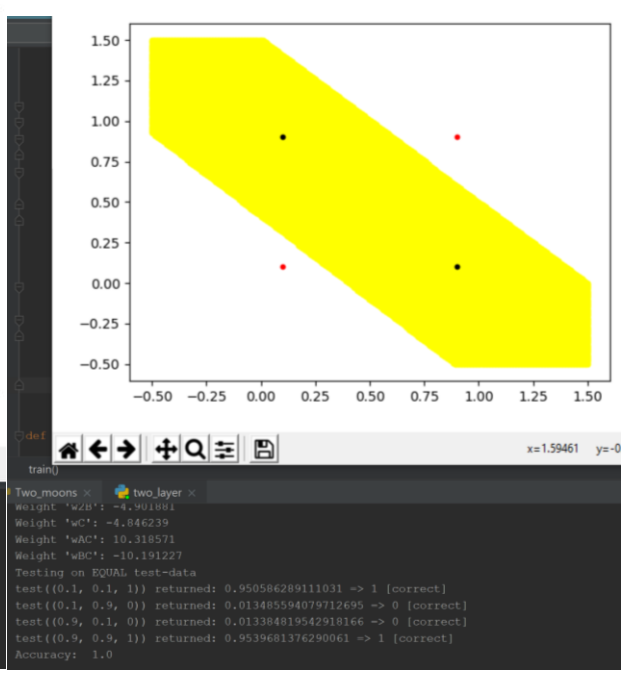


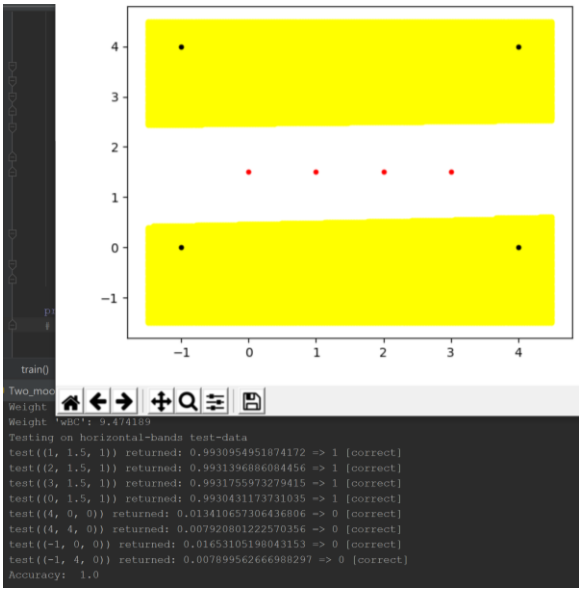


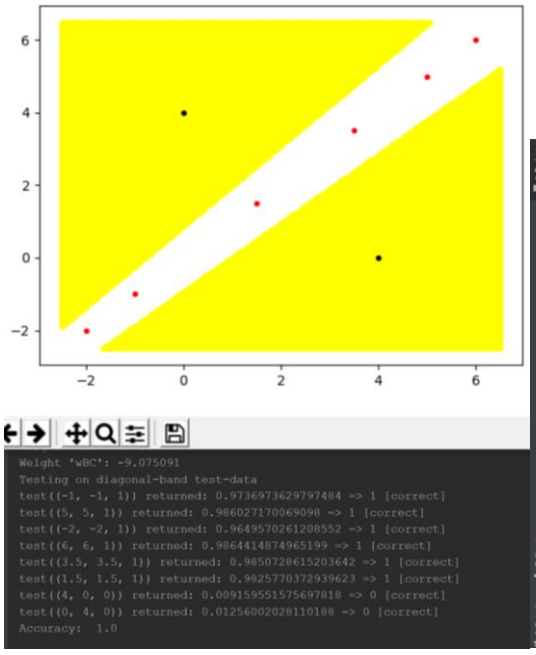
تست شبکه عصبی دو لایه:

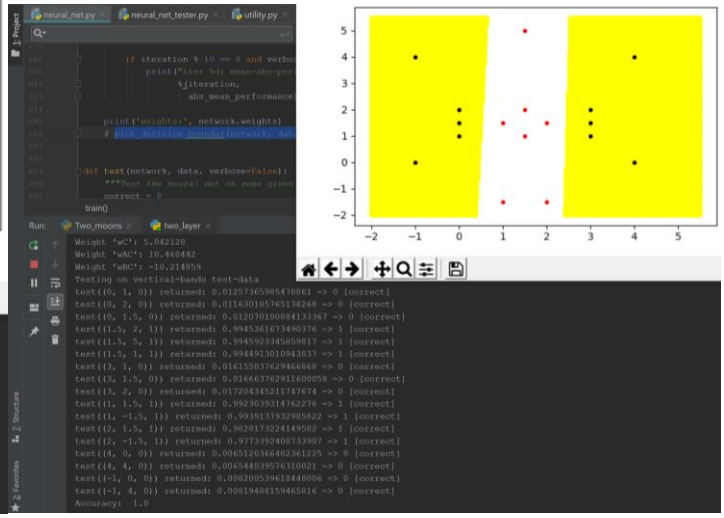


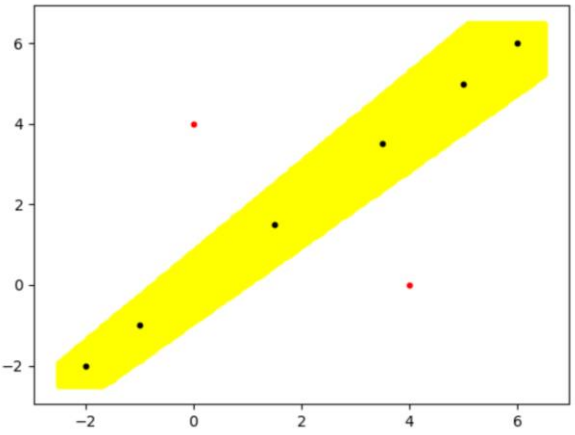

1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.50 −0.25 0.00 0.25 0.50 0.75 1.00 1.25 1.50
Weight 'w2B': -4.901881
Weight 'wC': 4.846239
Weight 'wAC': -10.318571
Weight 'wBC': 10.191227
Testing on NOT_EQUAL test-data
test((0.1, 0.1, 0)) returned: 0.04941371088896905 => 0 [correct]
test((0.1, 0.9, 1)) returned: 0.9865144059202873 => 1 [correct]
test((0.9, 0.1, 1)) returned: 0.9866151804570817 => 1 [correct]
test((0.9, 0.9, 0)) returned: 0.046031862370993895 => 0 [correct]
Accuracy: 1.0

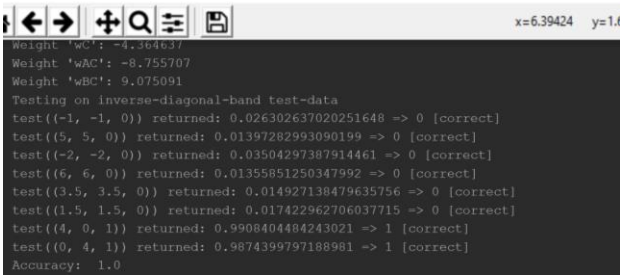

1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.50 −0.25 0.00 0.25 0.50 0.75 1.00 1.25 1.50
x=1.59461 y=-0
train()
Two_moons
two_layer
Weight 'w2B': -4.901881
Weight 'wC': -4.846239
Weight 'wAC': 10.318571
Weight 'wBC': -10.191227
Testing on EQUAL test-data
test((0.1, 0.1, 1)) returned: 0.950586289111031 => 1 [correct]
test((0.1, 0.9, 0)) returned: 0.013485594079712695 => 0 [correct]
test((0.9, 0.1, 0)) returned: 0.013384819542918166 => 0 [correct]
test((0.9, 0.9, 1)) returned: 0.9539681376290061 => 1 [correct]
Accuracy: 1.0


Weight 'wBC': 9.474189
Testing on horizontal-bands test-data
test((1, 1.5, 1)) returned: 0.9930954951874172 => 1 [correct]
test((2, 1.5, 1)) returned: 0.9931396886084456 => 1 [correct]
test((3, 1.5, 1)) returned: 0.9931755973279415 => 1 [correct]
test((0, 1.5, 1)) returned: 0.9930431173731035 => 1 [correct]
test((4, 0, 0)) returned: 0.013410657306436806 => 0 [correct]
test((4, 4, 0)) returned: 0.007920801222570356 => 0 [correct]
test((-1, 0, 0)) returned: 0.016531051980043153 => 0 [correct]
test((-1, 4, 0)) returned: 0.007899562666988297 => 0 [correct]
Accuracy: 1.0


Weight 'wBC': -9.075091
Testing on diagonal-band test-data
test((-1, -1, 1)) returned: 0.9736973629797484 => 1 [correct]
test((5, 5, 1)) returned: 0.986027170069098 => 1 [correct]
test((-2, -2, 1)) returned: 0.9649570261208552 => 1 [correct]
test((6, 6, 1)) returned: 0.9864414874965199 => 1 [correct]
test((3.5, 3.5, 1)) returned: 0.9850728615203642 => 1 [correct]
test((1.5, 1.5, 1)) returned: 0.9825770372939623 => 1 [correct]
test((4, 0, 0)) returned: 0.009159551575697818 => 0 [correct]
test((0, 4, 0)) returned: 0.012560020280110188 => 0 [correct]
Accuracy: 1.0


Weight 'wC': 5.042128
Weight 'wAC': 10.460442
Weight 'wBC': -10.214859
Testing on vertical-bands test-data
test((0, 1, 0)) returned: 0.012573659054788 61 => 0 [correct]
test((0, 2, 0)) returned: 0.011630105765138268 => 0 [correct]
test((0, 1.5, 0)) returned: 0.012070100884133367 => 0 [correct]
test((1.5, 2, 1)) returned: 0.9945361673490376 => 1 [correct]
test((1.5, 1, 1)) returned: 0.9945923345859017 => 1 [correct]
test((1.5, 1, 1)) returned: 0.9944913010943837 => 1 [correct]
test((3, 1, 0)) returned: 0.016155037629466668 => 0 [correct]
test((3, 1.5, 0)) returned: 0.016663762911600058 => 0 [correct]
test((3, 2, 0)) returned: 0.017204345211747674 => 0 [correct]
test((1, 1.5, 1)) returned: 0.9923039314762276 => 1 [correct]
test((1, -1.5, 1)) returned: 0.9939137932985822 => 1 [correct]
test((2, 1.5, 1)) returned: 0.9820173224149582 => 1 [correct]
test((2, -1.5, 1)) returned: 0.9773392408733907 => 1 [correct]
test((4, 0, 0)) returned: 0.006512036646826225 => 0 [correct]
test((4, 4, 0)) returned: 0.006544039576310021 => 0 [correct]
test((-1, 0, 0)) returned: 0.008200539618448006 => 0 [correct]
test((-1, 4, 0)) returned: 0.008194881594658 16 => 0 [correct]
Accuracy: 1.0





نحوه کشیدن نمودار:

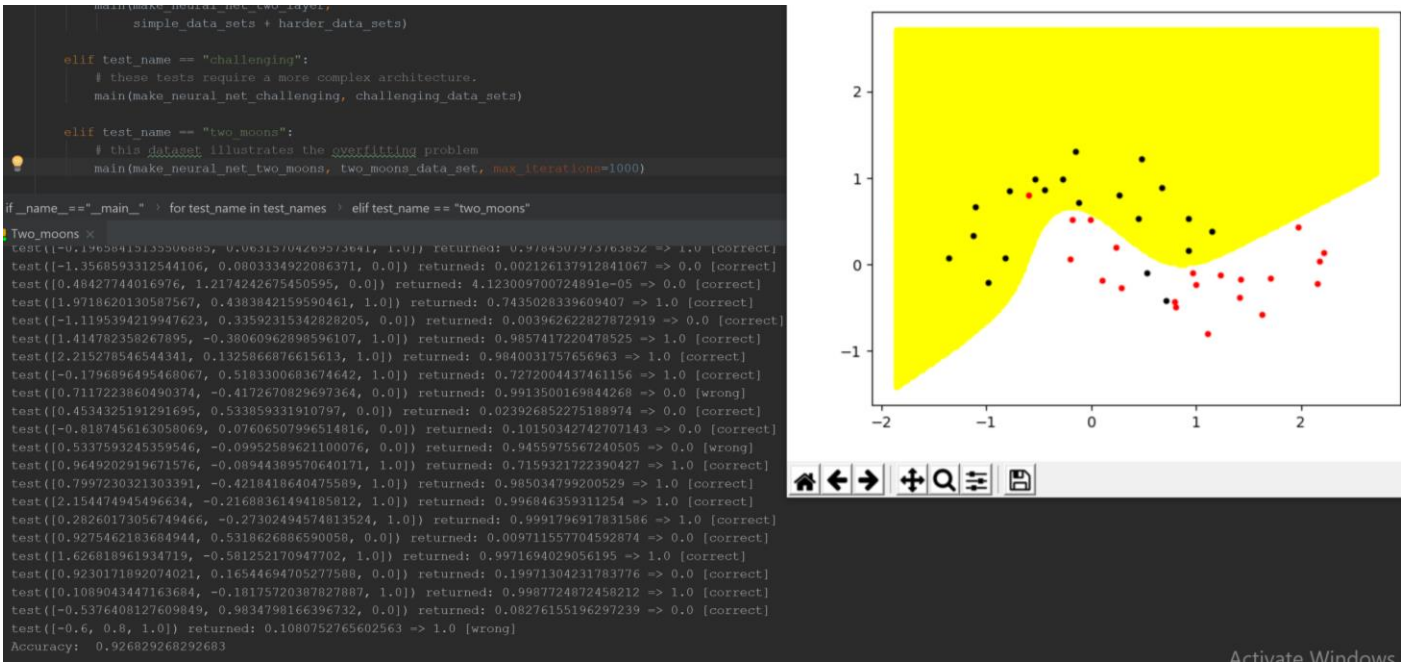
```
def plot_decision_boundary(network, xmin = -2, xmax= 2, ymin= -2, ymax= 2):
    fig = plt.figure()
    ax1 = fig.add_subplot(111)
    steps = 300
    x_span = np.linspace(xmin, xmax, steps)
    y_span = np.linspace(ymin, ymax, steps)
    x_border = []
    y_border = []
    for x in x_span:
        for y in y_span:
            for i in range(len(network.inputs)):
                if i % 2 == 0:
                    network.inputs[i].set_value(x)
                else:
                    network.inputs[i].set_value(y)
            network.clear_cache()
            result = network.output.output()
            if result < 0.5:
                x_border.append(x)
                y_border.append(y)

    ax1.scatter(x_border, y_border, s=10, c='yellow', marker="o", label=' < 0.5')
    plt.show()
```

پیش از اعمال l2:

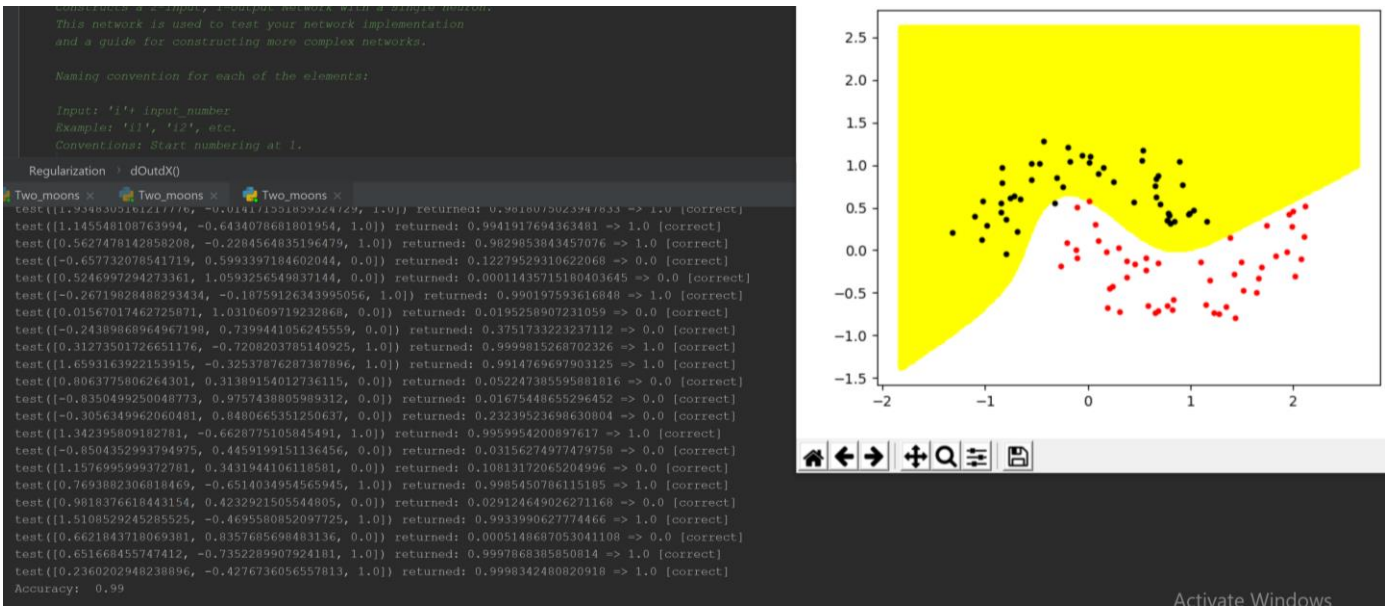
نحوه ساختن شبکه عصبی مانند قبل است (به علت زیاد بودن و تکراری بودن کد در گزارش آورده نشد)

با ۱۰۰ پیمایش:

داده های ترین:



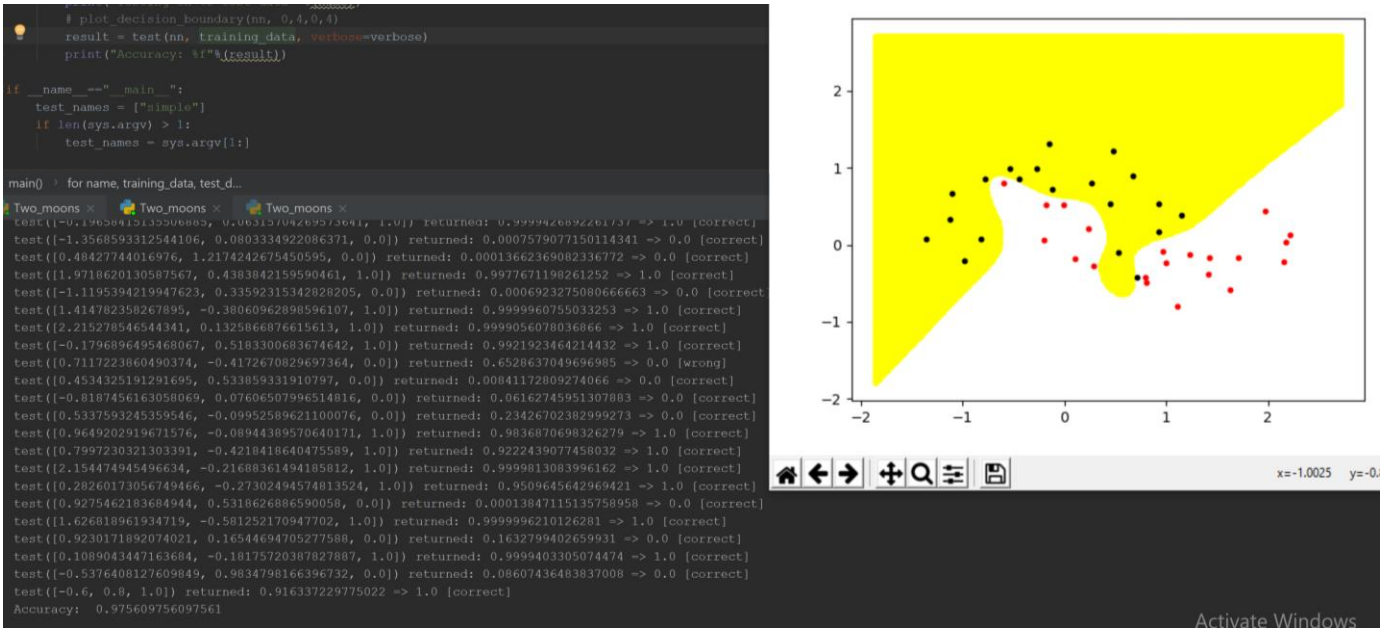
داده های تست:



با ۵۰۰ پیمایش:

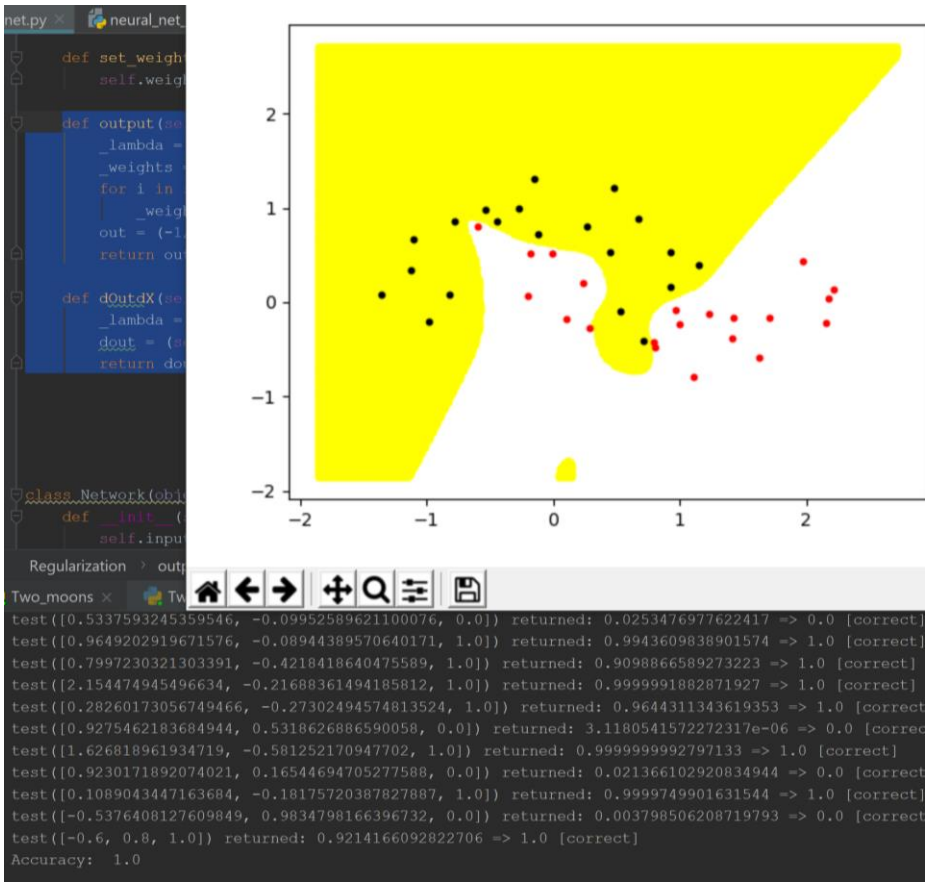
داده های تست:

داده های ترین:

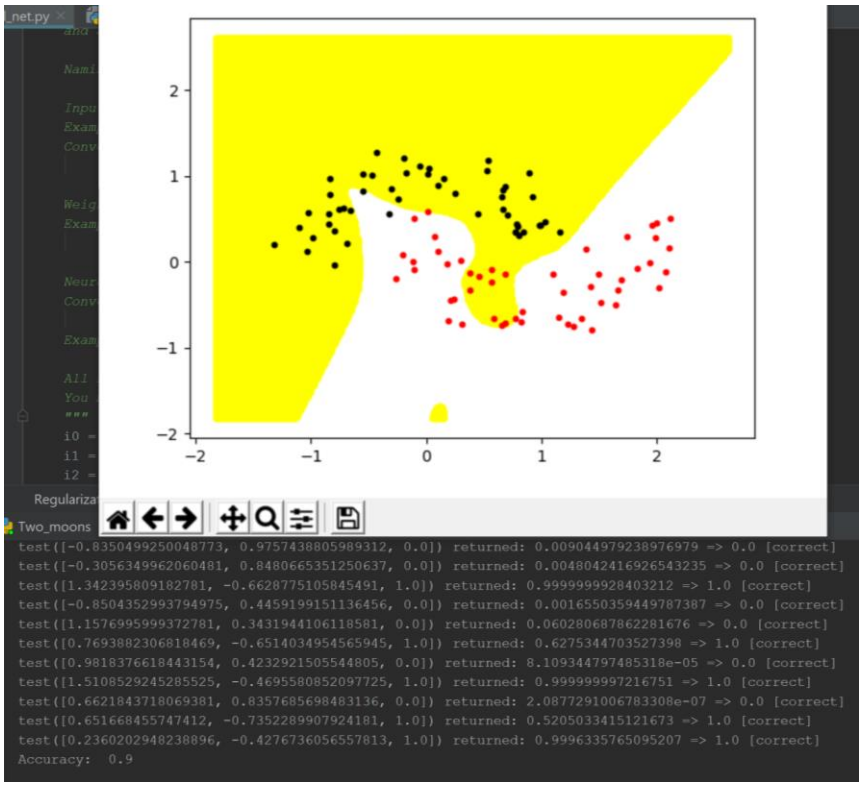


با ۱۰۰۰ پیمایش:

داده ترین:


net.py
neural_net_
def set_weigh
self.weigl
def output(se
_lambda =
_weights
for i in
_weigl
out = (-1
return ou
def dOutdX(se
_lambda =
dout = (s
return do
class Network(obj
def __init__(
self.inpu
Regularization
outp
Two_moons
Tw
test([0.5337593245359546, -0.09952589621100076, 0.0]) returned: 0.0253476977622417 => 0.0 [correct]
test([0.9649202919671576, -0.08944389570640171, 1.0]) returned: 0.9943609838901574 => 1.0 [correct]
test([0.7997230321303391, -0.4218418640475589, 1.0]) returned: 0.9098866589273223 => 1.0 [correct]
test([2.154474945496634, -0.21688361494185812, 1.0]) returned: 0.9999991882871927 => 1.0 [correct]
test([0.28260173056749466, -0.27302494574813524, 1.0]) returned: 0.9644311343619353 => 1.0 [correct
test([0.9275462183684944, 0.5318626886590058, 0.0]) returned: 3.1180541572272317e-06 => 0.0 [correc
test([1.626818961934719, -0.581252170947702, 1.0]) returned: 0.9999999992797133 => 1.0 [correct]
test([0.9230171892074021, 0.16544694705277588, 0.0]) returned: 0.021366102920834944 => 0.0 [correct
test([0.1089043447163684, -0.18175720387827887, 1.0]) returned: 0.9999749901631544 => 1.0 [correct]
test([-0.5376408127609849, 0.9834798166396732, 0.0]) returned: 0.003798506208719793 => 0.0 [correct
test([-0.6, 0.8, 1.0]) returned: 0.9214166092822706 => 1.0 [correct]
Accuracy: 1.0
2
1
0
−1
−2
−2
−1
0
1
2

داده تست:



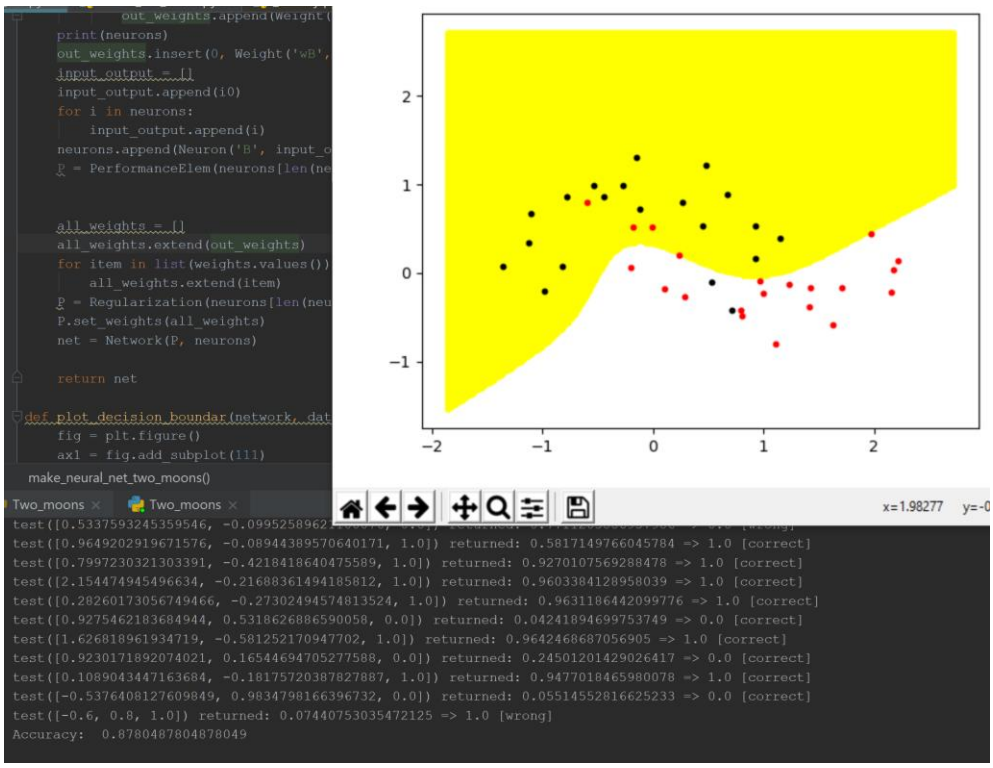
اعمال l2:

مانند همان قبلی است فقط در قسمت output با مقدار لاندا*سیگما جمع شده و درقسمت doutdx با 2*landa*wi

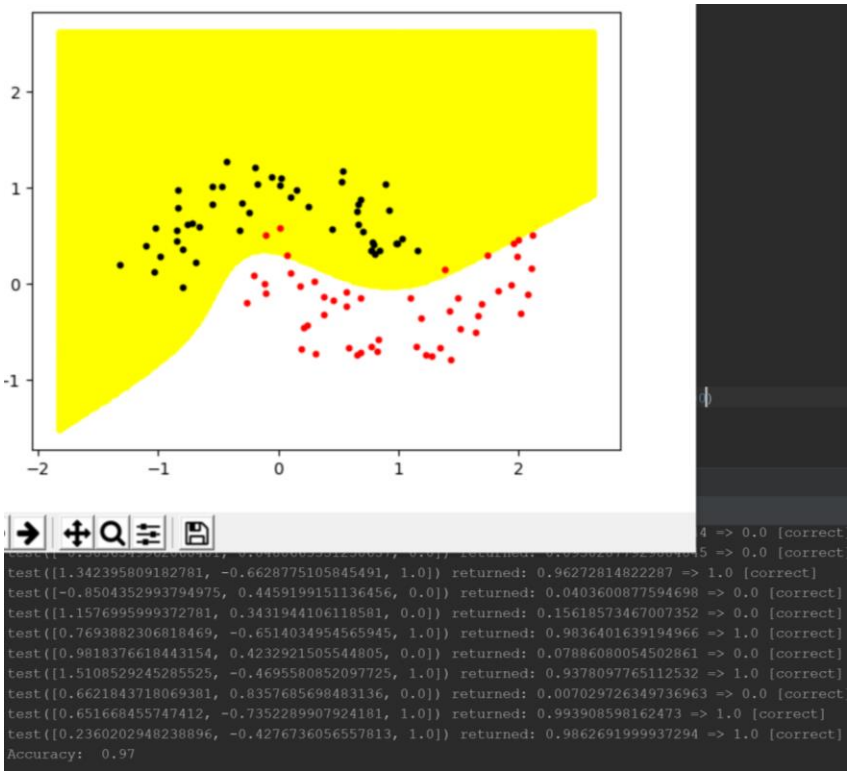
```
def output(self):
    _lambda = 0.0001
    _weights = []
    for i in self.weights:
        _weights.append(i.get_value())
    out = (-1/2) * ((self.my desired val - self.my input.output()) ** 2) +  lambda *
np.linalg.norm(np.asarray(_weights, dtype=np.float32))
    return out

def dOutdX(self, elem):
    _lambda = 0.0001
    dout = (self.my desired val - self.my input.output()) * self.my input.dOutdX(elem) - 2 *
_lambda * elem.get_value()
    return dout
```

داده های ۱۰۰ پیمایشی:

ترین شده:

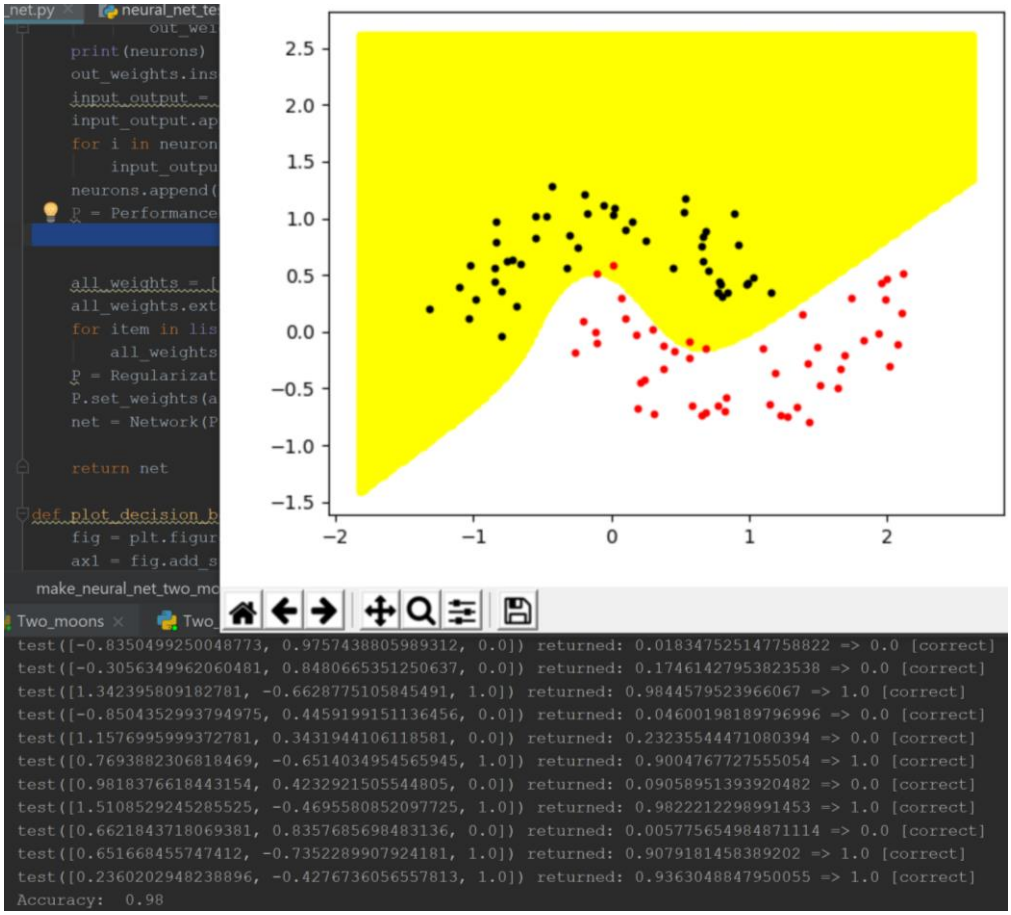


داده تست:

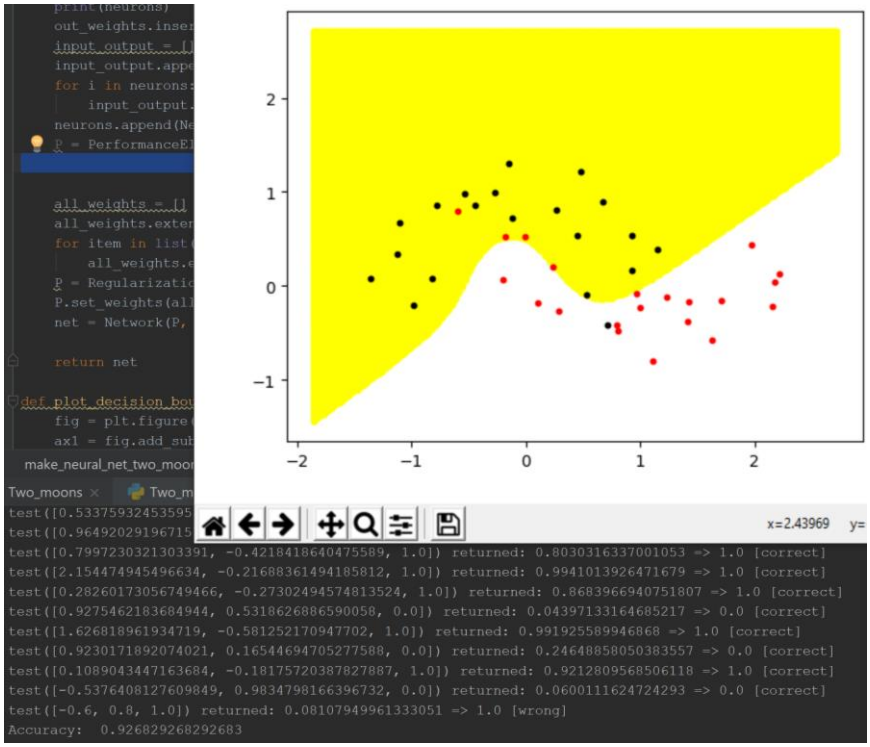


داده های ۵۰۰ پیمایشی:

داده های تست:

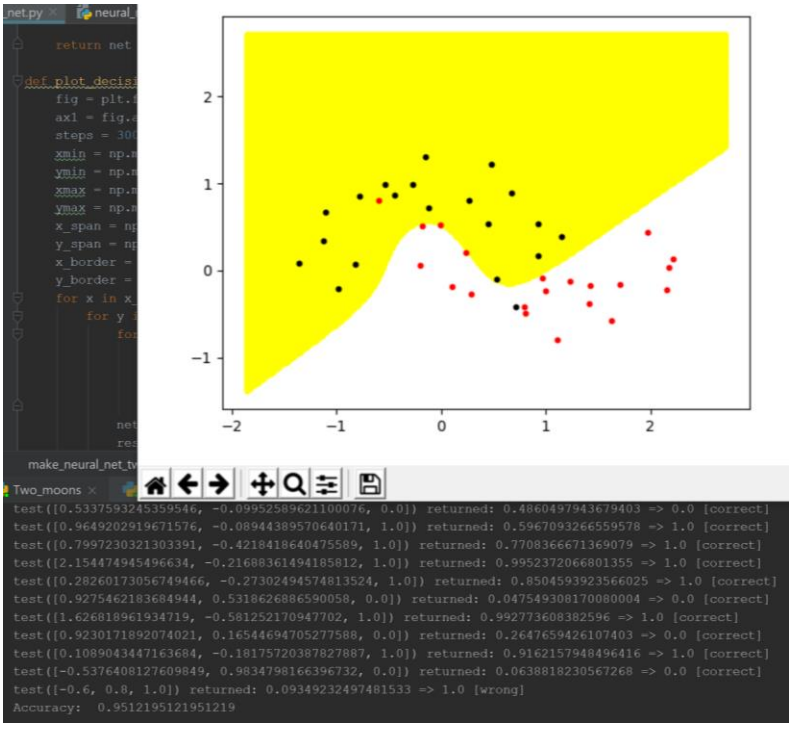


داده ی ترین:

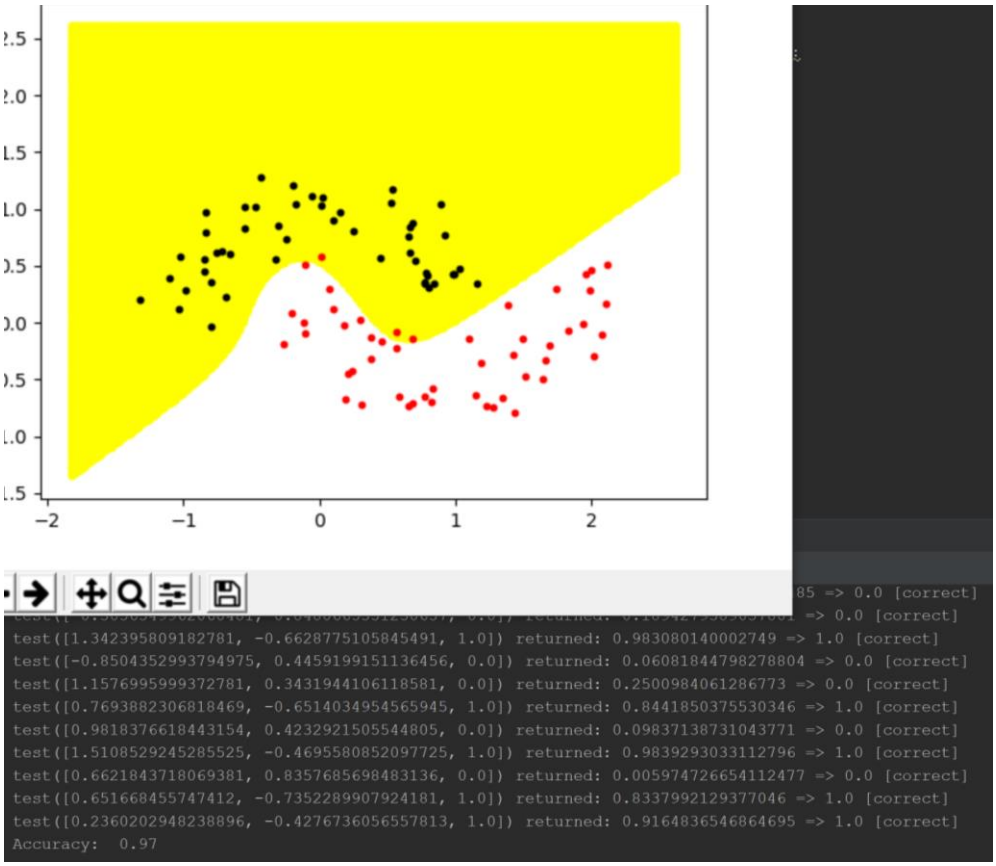


1000 پیمایش:

داده ترین:



داده تست:



جدول:

L2 test	L2 train	Test	Train	accuracy
97	87	99	92	100
98	92	92	97	500
97	95	90	1	1000