



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 دقیقاً مانند محاسبه خطا می باشد 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 سیگموئید 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_derivation = 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_derivation
```

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 $f(x+\text{delta})$ 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 برخوردیم)

برای شبکه عصبی دولایه همان مراحل که در شبکه عصبی پایه رفته بودند را رفتیم 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])
```

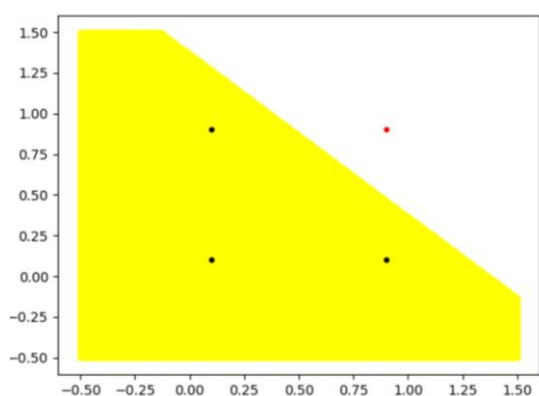
```
P = PerformanceElem(C, 0.0)
net = Network(P, [A, B, C])
return net
```

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.

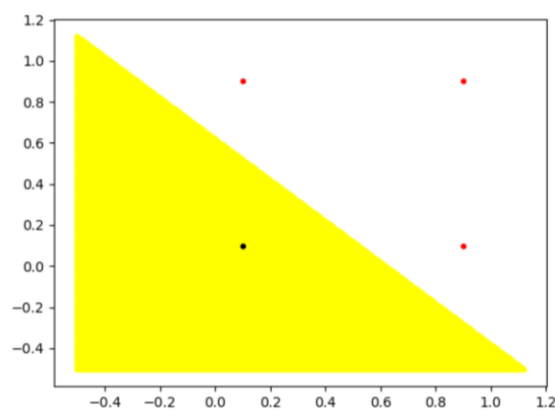
تست شبکه عصبی پایه:

نمودار داده ترین و تست:

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

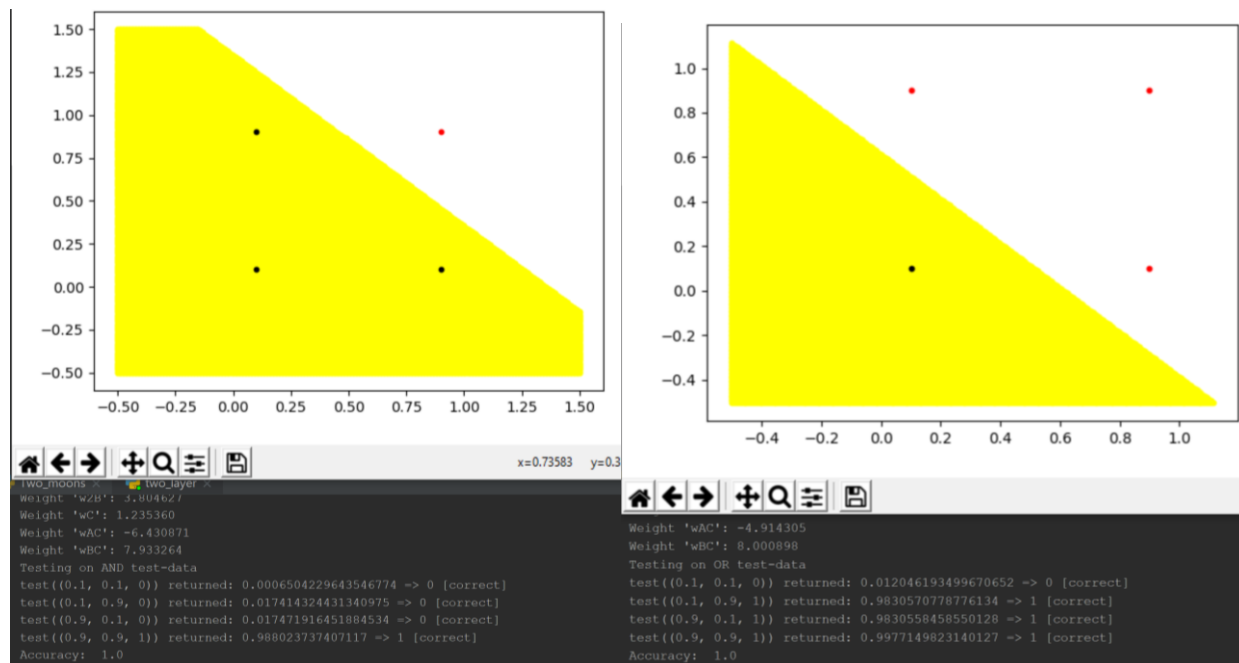


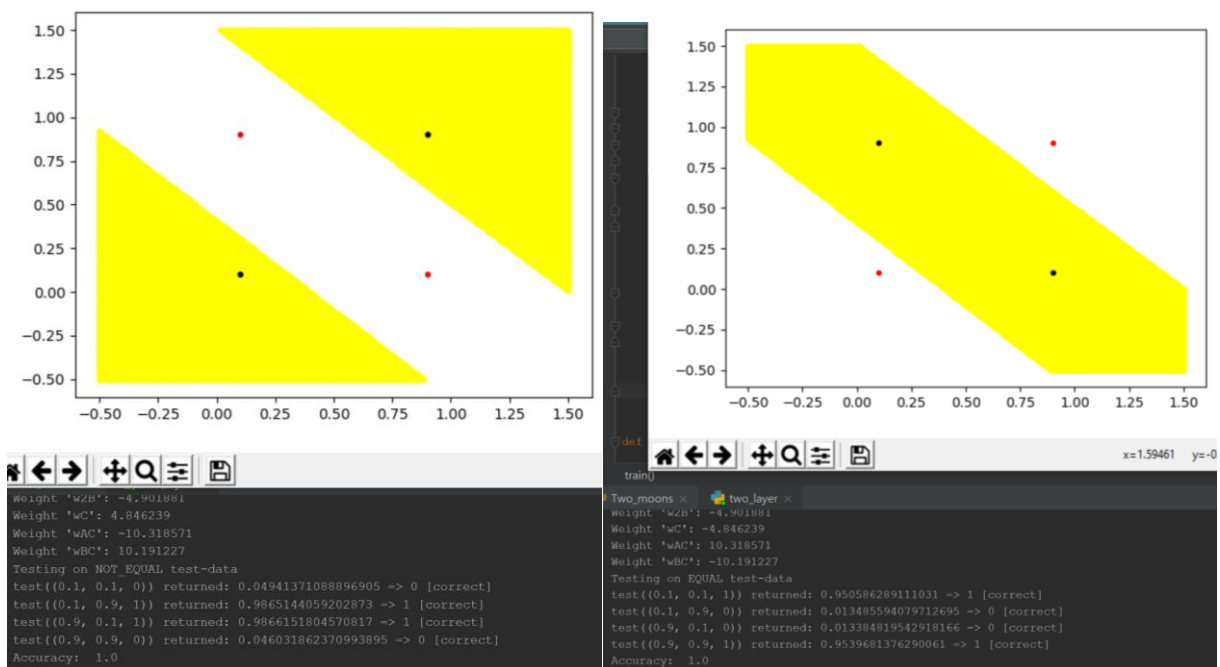
```
Two_moons x simple x
trained weights:
Weight 'w1A': 10.499844
Weight 'w2A': 10.499561
Weight 'wA': 14.366979
Testing on AND test-data
test((0.1, 0.1, 0)) returned: 4.704254617957318e-06 => 0 [correct]
test((0.1, 0.9, 0)) returned: 0.020484490369173127 => 0 [correct]
test((0.9, 0.1, 0)) returned: 0.02048903863720659 => 0 [correct]
test((0.9, 0.9, 1)) returned: 0.9893604979736043 => 1 [correct]
Accuracy: 1.0
```

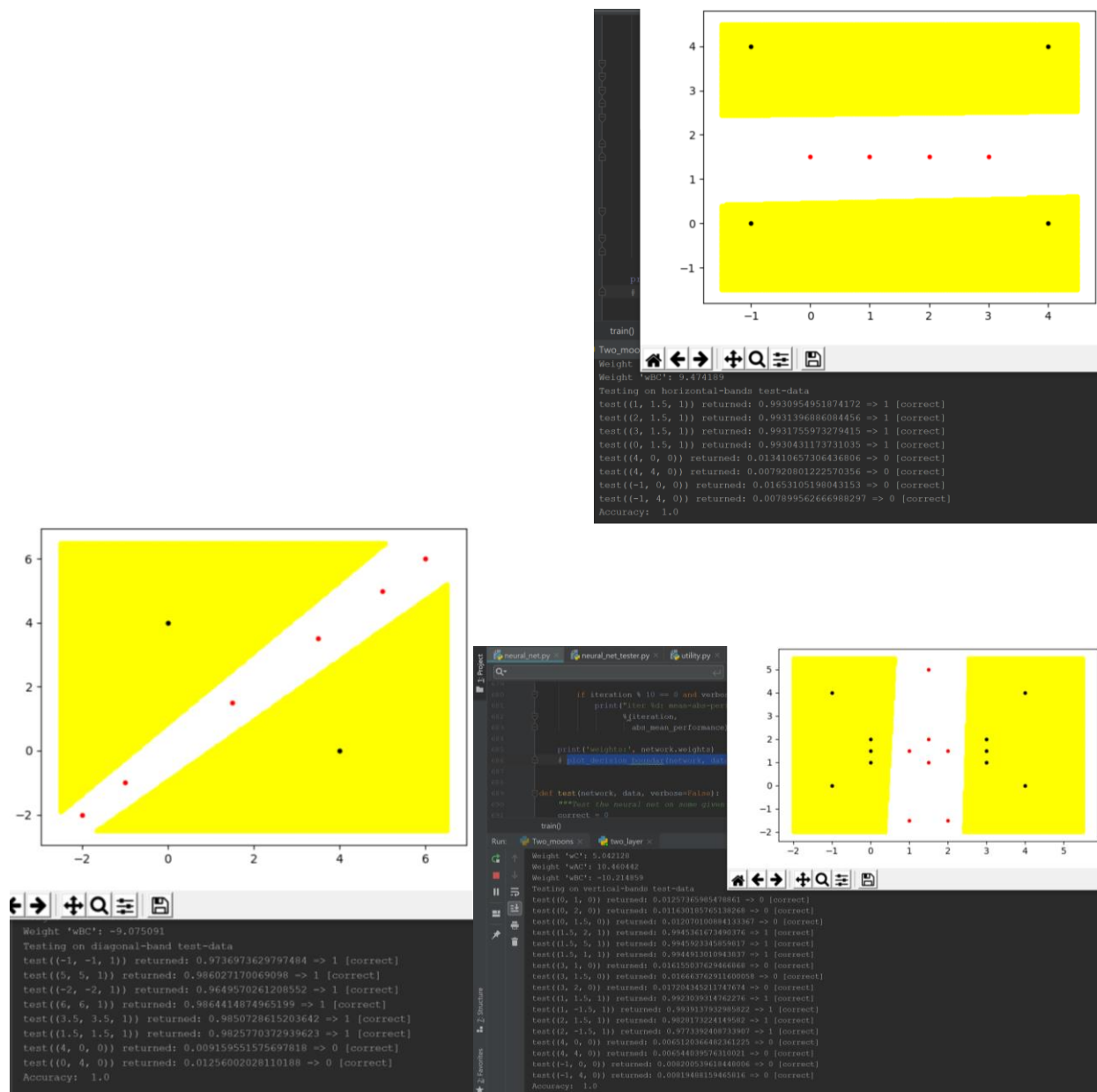


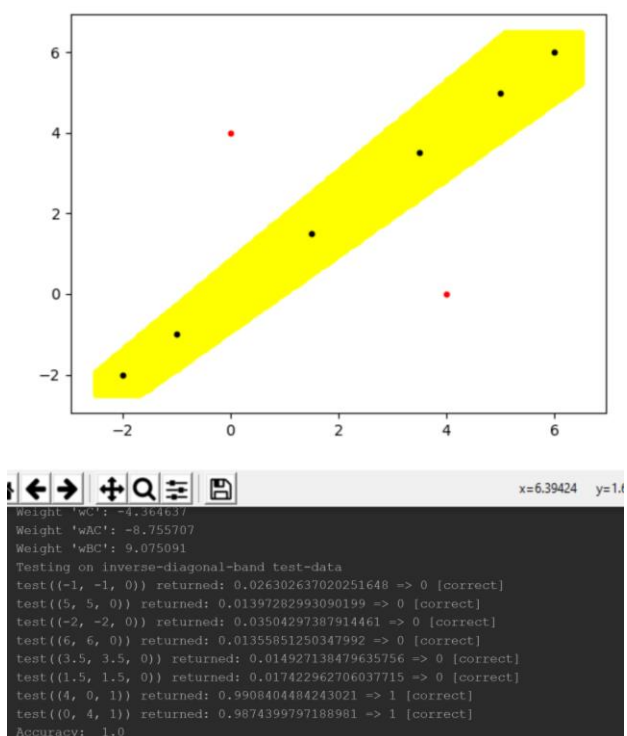
```
Weight 'wA': 6.686619
Testing on OR test-data
test((0.1, 0.1, 0)) returned: 0.010654939503118913 => 0 [correct]
test((0.1, 0.9, 1)) returned: 0.9835615677616205 => 1 [correct]
test((0.9, 0.1, 1)) returned: 0.9835557959042682 => 1 [correct]
test((0.9, 0.9, 1)) returned: 0.9999969906368888 => 1 [correct]
```

تست شبکه عصبی دو لایه:









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 I2:

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:

```

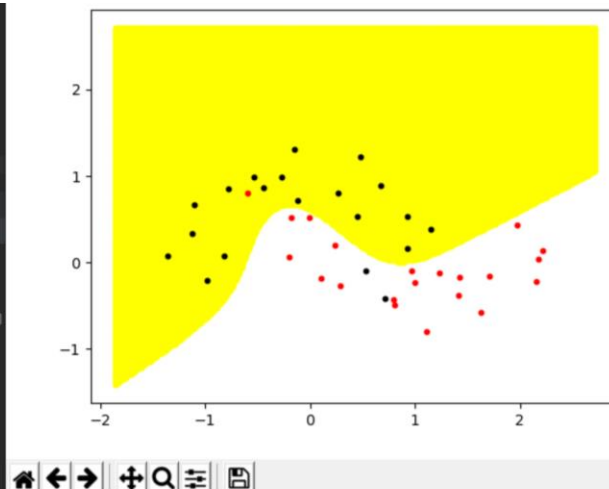
main(make_neural_net_two_moons,
     simple_data_sets + harder_data_sets)

elif test_name == "challenging":
    # these tests require a more complex architecture.
    main(make_neural_net_challenging, challenging_data_sets)

elif test_name == "two_moons":
    # this dataset illustrates the overfitting problem
    main(make_neural_net_two_moons, two_moons_data_set, max_iterations=1000)

if __name__ == "__main__":
    for test_name in test_names:
        elif test_name == "two_moons":
            Two_moons x
            test((-0.1968413135506885, 0.06315704269573641, 1.0)) returned: 0.9784507973763832 => 1.0 [correct]
            test((-1.3568593312544106, 0.0803334922086371, 0.0)) returned: 0.002126137912841067 => 0.0 [correct]
            test((0.48427744016976, 1.2174242675450595, 0.0)) returned: 4.123009700724891e-05 => 0.0 [correct]
            test((1.9718620130587567, 0.4383842159590461, 1.0)) returned: 0.7435028339609407 => 1.0 [correct]
            test((-1.1195394219947623, 0.33592315342828205, 0.0)) returned: 0.003962622827872919 => 0.0 [correct]
            test((1.414782358267895, -0.38060962898596107, 1.0)) returned: 0.9857417220478525 => 1.0 [correct]
            test((2.215278546544341, 0.1325866876615613, 1.0)) returned: 0.9840031757656963 => 1.0 [correct]
            test((-0.1796896495468067, 0.5183300683674642, 1.0)) returned: 0.7272004437461156 => 1.0 [correct]
            test((0.7117223860490374, -0.4172670829697364, 0.0)) returned: 0.9913500169844268 => 0.0 [wrong]
            test((0.4534325191291695, 0.533859331910797, 0.0)) returned: 0.023926852275188974 => 0.0 [correct]
            test((-0.8187456163058069, 0.07606507996514816, 0.0)) returned: 0.10150342742707143 => 0.0 [correct]
            test((0.5337593245359546, -0.09952589621100076, 0.0)) returned: 0.9455975567240505 => 0.0 [wrong]
            test((0.9649202919671576, -0.08944389570640171, 1.0)) returned: 0.7159321722390427 => 1.0 [correct]
            test((0.7997230321303391, -0.4218418640475589, 1.0)) returned: 0.985034799200529 => 1.0 [correct]
            test((2.154474945496634, -0.21688361494185812, 1.0)) returned: 0.996846359311254 => 1.0 [correct]
            test((0.28260173056749466, -0.27302494574813524, 1.0)) returned: 0.9991796917831586 => 1.0 [correct]
            test((0.9275462183684944, 0.5318626886590058, 0.0)) returned: 0.009711557704592874 => 0.0 [correct]
            test((1.626818961934719, -0.581252170947702, 1.0)) returned: 0.9971694029056195 => 1.0 [correct]
            test((0.9230171892074021, 0.16544694705277588, 0.0)) returned: 0.19971304231783776 => 0.0 [correct]
            test((0.1089043447163684, -0.18175720387827887, 1.0)) returned: 0.9987724872458212 => 1.0 [correct]
            test((0.5376408127609849, 0.9834798166396732, 0.0)) returned: 0.08276155196297239 => 0.0 [correct]
            test((-0.6, 0.8, 1.0)) returned: 0.1080752765602563 => 1.0 [wrong]
            Accuracy: 0.926829268292683

```



داده های تست:

```

Constructing a simple, complex network with a single neuron.
This network is used to test your network implementation
and a guide for constructing more complex networks.

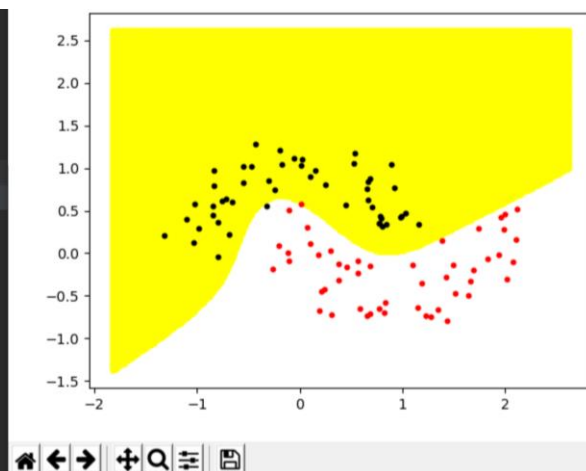
Naming convention for each of the elements:

Input: 'i' + input number
Example: 'i1', 'i2', etc.
Conventions: Start numbering at 1.

Regularization: dOut(X)

Two_moons x
test((1.934630516121776, -0.014171551859324729, 1.0)) returned: 0.9618075023947633 => 1.0 [correct]
test((1.145548108763994, -0.6434078681801954, 1.0)) returned: 0.9941917694363481 => 1.0 [correct]
test((0.5627478142858208, -0.2284564835196479, 1.0)) returned: 0.9829853843457076 => 1.0 [correct]
test((-0.657732078541719, 0.5993397184602044, 0.0)) returned: 0.12279529310622068 => 0.0 [correct]
test((0.5246997294273361, 1.0593256549837144, 0.0)) returned: 0.0001143571580403645 => 0.0 [correct]
test((-0.26719828488293434, -0.18759126343995056, 1.0)) returned: 0.990197593616848 => 1.0 [correct]
test((0.01567017462725871, 1.0310609719232868, 0.0)) returned: 0.0195258907231059 => 0.0 [correct]
test((-0.24389868964967198, 0.7399441056245559, 0.0)) returned: 0.3751733223237112 => 0.0 [correct]
test((0.31273501726651176, -0.7208203785140925, 1.0)) returned: 0.9999015268702326 => 1.0 [correct]
test((1.6593163922153915, -0.32537876287387896, 1.0)) returned: 0.9914769697903125 => 1.0 [correct]
test((0.8063775806264301, 0.31389154012736115, 0.0)) returned: 0.0522473859595881816 => 0.0 [correct]
test((-0.8350499250048773, 0.9757438805989312, 0.0)) returned: 0.01675448655296452 => 0.0 [correct]
test((-0.3056349962060481, 0.8480665351250637, 0.0)) returned: 0.23239523698630804 => 0.0 [correct]
test((1.342395809182781, -0.6628775105845491, 1.0)) returned: 0.995954200897617 => 1.0 [correct]
test((-0.8504352993794975, 0.4459199151136456, 0.0)) returned: 0.03156274977479758 => 0.0 [correct]
test((1.1576995999372781, 0.3431944106118581, 0.0)) returned: 0.10813172065204996 => 0.0 [correct]
test((0.7693882306018469, -0.6514034954565945, 1.0)) returned: 0.9985450786115105 => 1.0 [correct]
test((0.9818376618443154, 0.4232921505544805, 0.0)) returned: 0.029124649026271168 => 0.0 [correct]
test((1.5108529245258525, -0.4695580852097725, 1.0)) returned: 0.9933990627774466 => 1.0 [correct]
test((0.6621843718069381, 0.8357685698483136, 0.0)) returned: 0.0005148687053041108 => 0.0 [correct]
test((0.65168455747412, -0.7352289907924181, 1.0)) returned: 0.9997868385850814 => 1.0 [correct]
test((0.2360202948238896, -0.4276736056557813, 1.0)) returned: 0.9998342480820918 => 1.0 [correct]
            Accuracy: 0.99

```



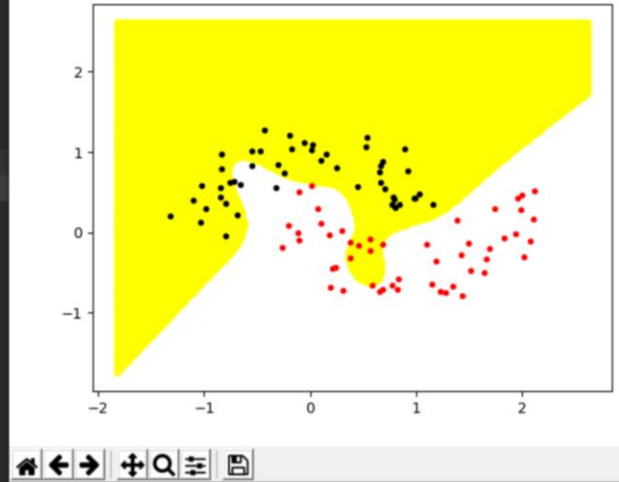
با ۵۰۰ پیمایش:

داده های تست:

```
def clear_cache(self):
    for n in self.neurons:
        n.clear_cache()

def seed_random():
    """Seed the random number generator so that random
    numbers are deterministically 'random'"""

Regularization > dOutX()
Two_moons × Two_moons × Two_moons ×
test([1.934630516121776, -0.014171531859324729, 1.0]) returned: 0.9996724387643676 => 1.0 [correct]
test([1.145548108763994, -0.6434078681801954, 1.0]) returned: 0.9998025760211674 => 1.0 [correct]
test([0.5627478142858208, -0.2284564835196479, 1.0]) returned: 0.20462889589560007 => 1.0 [wrong]
test([-0.657732078541719, 0.5993397184602044, 0.0]) returned: 0.9029793409780972 => 0.0 [wrong]
test([0.5246997294273361, 1.0593256549837144, 0.0]) returned: 7.697791393709113e-05 => 0.0 [correct]
test([-0.26719828488293434, -0.18759126343995056, 1.0]) returned: 0.9993994572376343 => 1.0 [correct]
test([0.01567017462725871, 1.0310609719232868, 0.0]) returned: 0.0007372373810081873 => 0.0 [correct]
test([-0.24389868964967198, 0.7399441056245559, 0.0]) returned: 0.3379658769288886 => 0.0 [correct]
test([0.31273501726651176, -0.7208203785140925, 1.0]) returned: 0.9966120104967747 => 1.0 [correct]
test([1.6593163922153915, -0.32537876287387896, 1.0]) returned: 0.9999969738075148 => 1.0 [correct]
test([0.8063775806264301, 0.31389154012736115, 0.0]) returned: 0.0023916349600999736 => 0.0 [correct]
test([-0.8350499250048773, 0.9757438805989312, 0.0]) returned: 0.07041651657565311 => 0.0 [correct]
test([-0.3056349962060481, 0.848066531250637, 0.0]) returned: 0.04649469029026079 => 0.0 [correct]
test([1.342395809182781, -0.6628775105845491, 1.0]) returned: 0.9999898171054749 => 1.0 [correct]
test([-0.8504352993794975, 0.4459199151136456, 0.0]) returned: 0.022917757363716575 => 0.0 [correct]
test([1.1576995999372781, 0.3431944106118581, 0.0]) returned: 0.07200443318957837 => 0.0 [correct]
test([0.7693882306818469, -0.6514034954565945, 1.0]) returned: 0.8896227634938502 => 1.0 [correct]
test([0.9818376618443154, 0.4232921505544805, 0.0]) returned: 0.0016212773147728351 => 0.0 [correct]
test([1.5108529245285525, -0.4695580852097725, 1.0]) returned: 0.999988658902451 => 1.0 [correct]
test([0.6621843718069381, 0.8357685698483136, 0.0]) returned: 2.854015000326222e-06 => 0.0 [correct]
test([0.651668455747412, -0.7352289907924181, 1.0]) returned: 0.8165333114244784 => 1.0 [correct]
test([0.2360202948238896, -0.4276736056557813, 1.0]) returned: 0.9982549089584067 => 1.0 [correct]
Accuracy: 0.92
```

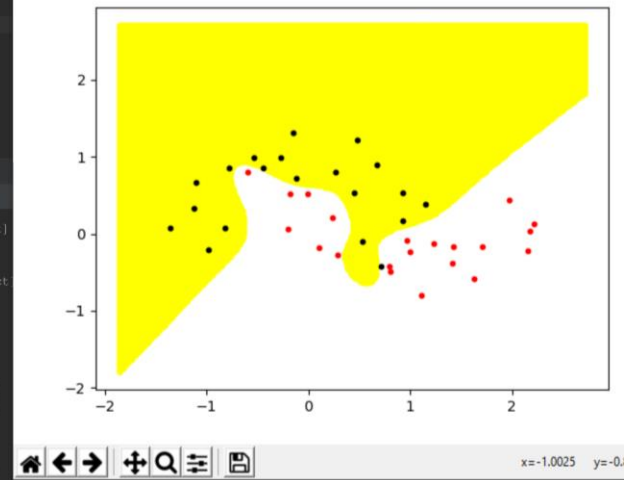


داده های ترین:

```
# plot decision boundary(nn, 0.4,0.4)
result = test(nn, training_data, verbose=verbose)
print("Accuracy: %f"%(result))

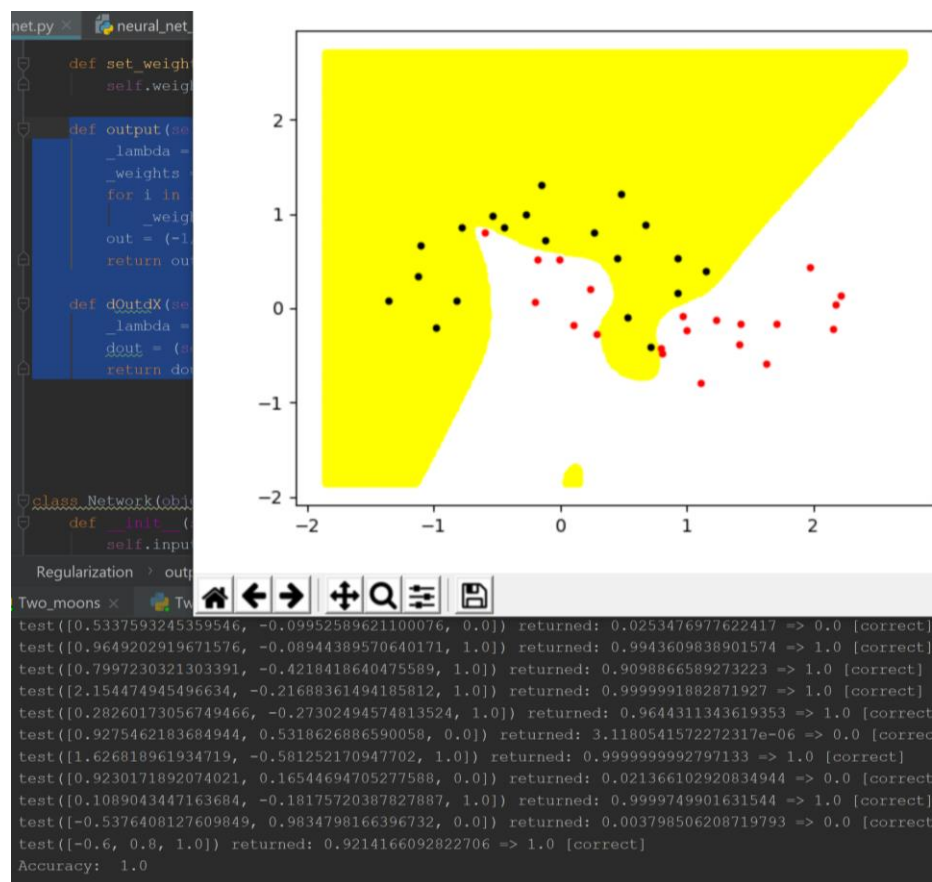
if __name__ == "__main__":
    test_names = ["simple"]
    if len(sys.argv) > 1:
        test_names = sys.argv[1:]

main() > for name, training_data, test_d...
Two_moons × Two_moons × Two_moons ×
test([-0.19038413133306885, 0.06313704269573641, 1.0]) returned: 0.9999426892261737 => 1.0 [correct]
test([-1.3568593312544106, 0.0803334922086371, 0.0]) returned: 0.0007579077150114341 => 0.0 [correct]
test([0.48427744016976, 1.2174242675450595, 0.0]) returned: 0.00013662369082336772 => 0.0 [correct]
test([1.9718620130587567, 0.4383842159590461, 1.0]) returned: 0.9977671198261252 => 1.0 [correct]
test([-1.1195394219947623, 0.33592315342828205, 0.0]) returned: 0.000692375080666663 => 0.0 [correct]
test([1.414782358267895, -0.38060962898596107, 1.0]) returned: 0.9999960755033253 => 1.0 [correct]
test([2.21527856544341, 0.1325866876615613, 1.0]) returned: 0.9999056078036866 => 1.0 [correct]
test([-0.179696495468067, 0.518330083674642, 1.0]) returned: 0.9921923464214432 => 1.0 [correct]
test([0.7117223860490374, -0.4172670829697364, 0.0]) returned: 0.6528637049696985 => 0.0 [wrong]
test([0.4534325191291695, 0.533859331910797, 0.0]) returned: 0.00841172009274066 => 0.0 [correct]
test([-0.8187456163058069, 0.07066507996514816, 0.0]) returned: 0.06162745951307883 => 0.0 [correct]
test([0.5337593245359546, -0.09952589621100076, 0.0]) returned: 0.23426702382999273 => 0.0 [correct]
test([0.9649202919671576, -0.08944389570640171, 1.0]) returned: 0.9836870698326279 => 1.0 [correct]
test([0.7997230321303391, -0.4218418640475589, 1.0]) returned: 0.922439077458032 => 1.0 [correct]
test([2.154474945496634, -0.21688361494185812, 1.0]) returned: 0.9999813083996162 => 1.0 [correct]
test([0.28260173056749466, -0.27302494574813524, 1.0]) returned: 0.9509645642969421 => 1.0 [correct]
test([0.9275462183684944, 0.5318626886590058, 0.0]) returned: 0.00013847115135758958 => 0.0 [correct]
test([1.626818961934719, -0.581252170947702, 1.0]) returned: 0.9999996210126281 => 1.0 [correct]
test([0.9230171892074021, 0.16544694705277588, 0.0]) returned: 0.1632799402659931 => 0.0 [correct]
test([0.1089043447163684, -0.18175720387827887, 1.0]) returned: 0.999403305074474 => 1.0 [correct]
test([-0.5376408127609849, 0.9834798166396732, 0.0]) returned: 0.08607436483837008 => 0.0 [correct]
test([-0.6, 0.8, 1.0]) returned: 0.916337229775022 => 1.0 [correct]
Accuracy: 0.975609756097561
```

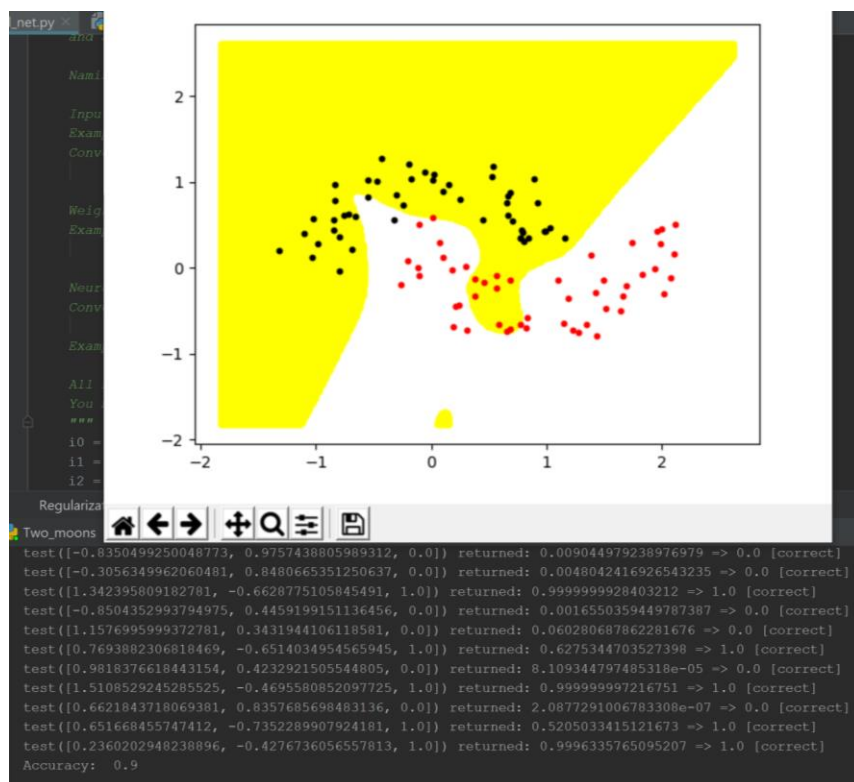


با ۱۰۰۰ پیمایش:

داده ترین:



داده تست:



اعمال 12:

مانند همان قبلی است فقط در قسمت output با مقدار لاندای سیگما جمع شده و در قسمت $doutdx$ با $2 * \lambda * w_i$

```

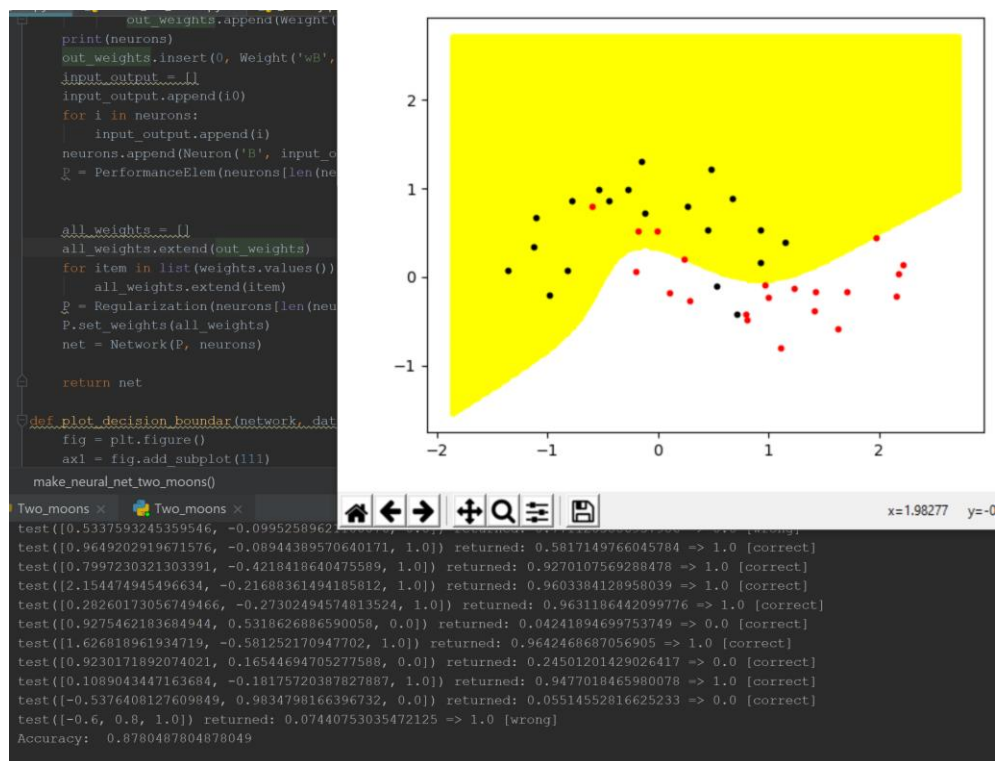
def output(self):
    _lambda = 0.0001
    _weights = []
    for i in self.weights:
        _weights.append(i.get_value())
    out = (-1/2) * ((self.my desired val - self.my input.output()) ** 2) + _lambda *
    np.linalg.norm(np.asarray(_weights, dtype=np.float32))
    return out

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

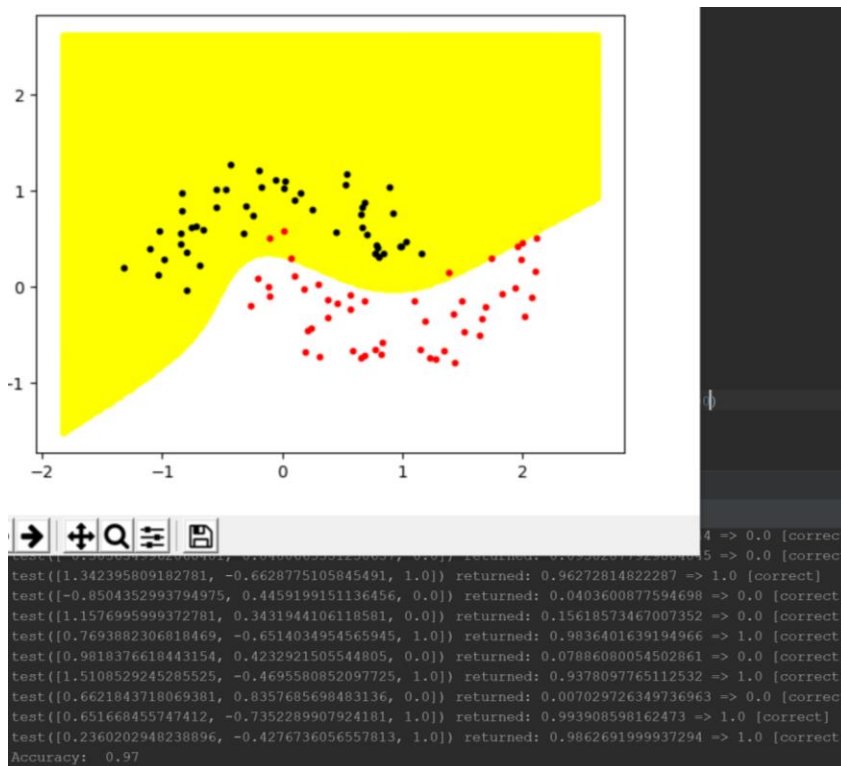
```

داده های ۱۰۰ پیمایشی:

ترین شده:

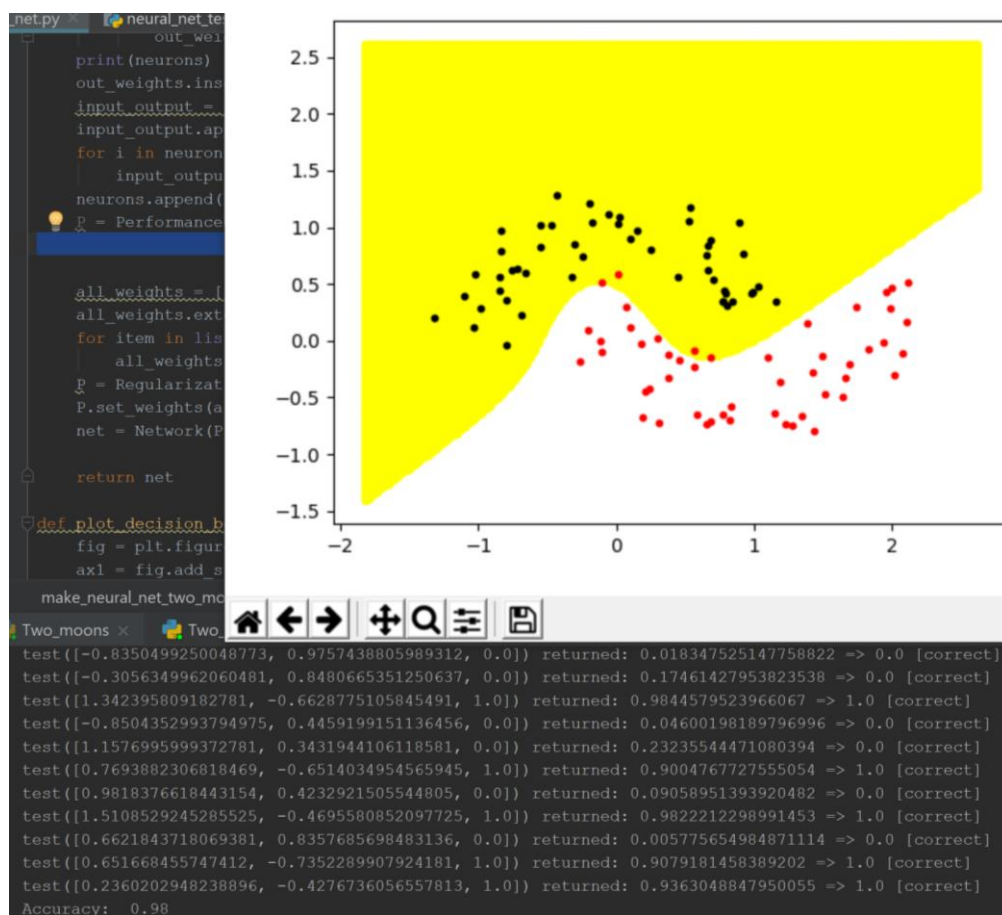


داده تست:

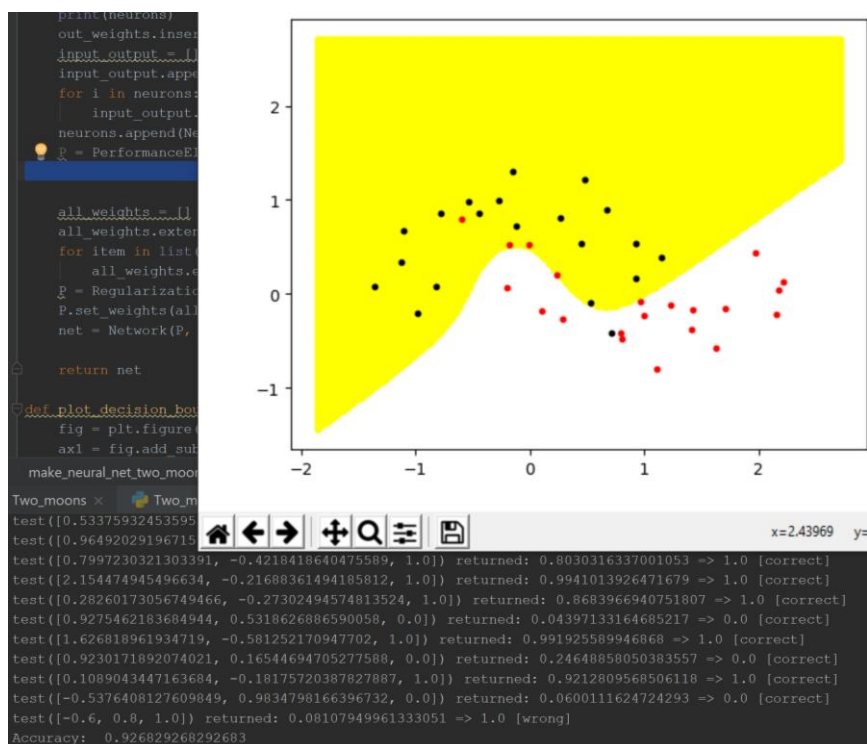


داده های ۵۰۰ پیمایشی:

داده های تست:

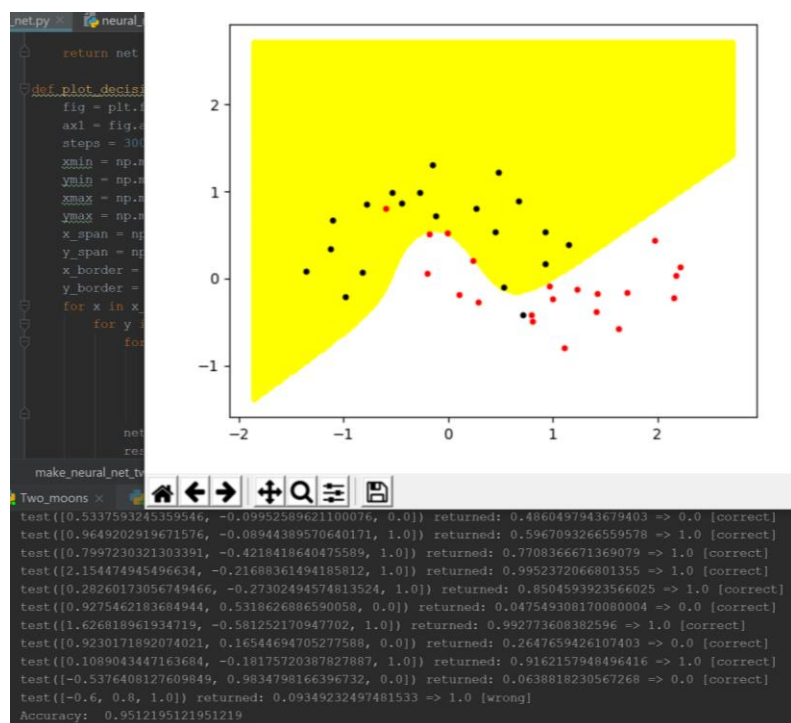


داده ی ترین:

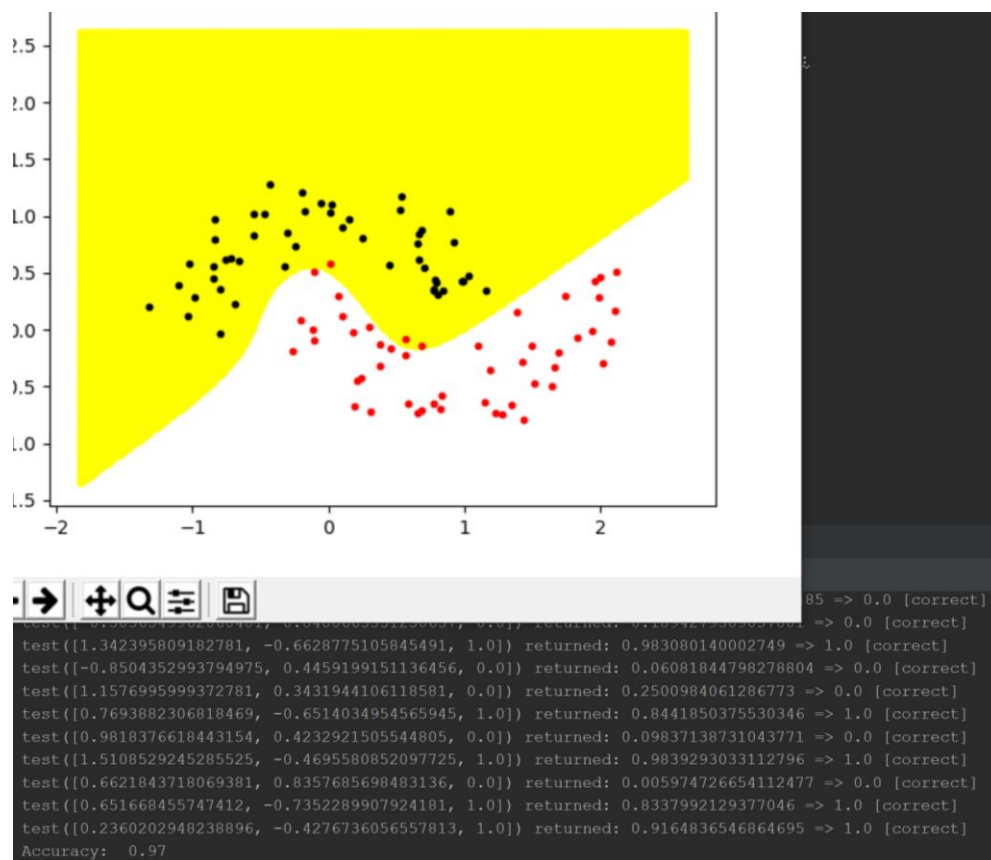


1000 پیمایش:

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