بسمه تعالی

Artificial Intelligence
Assignment 5

Mohammad Mahdi Islami
810195548

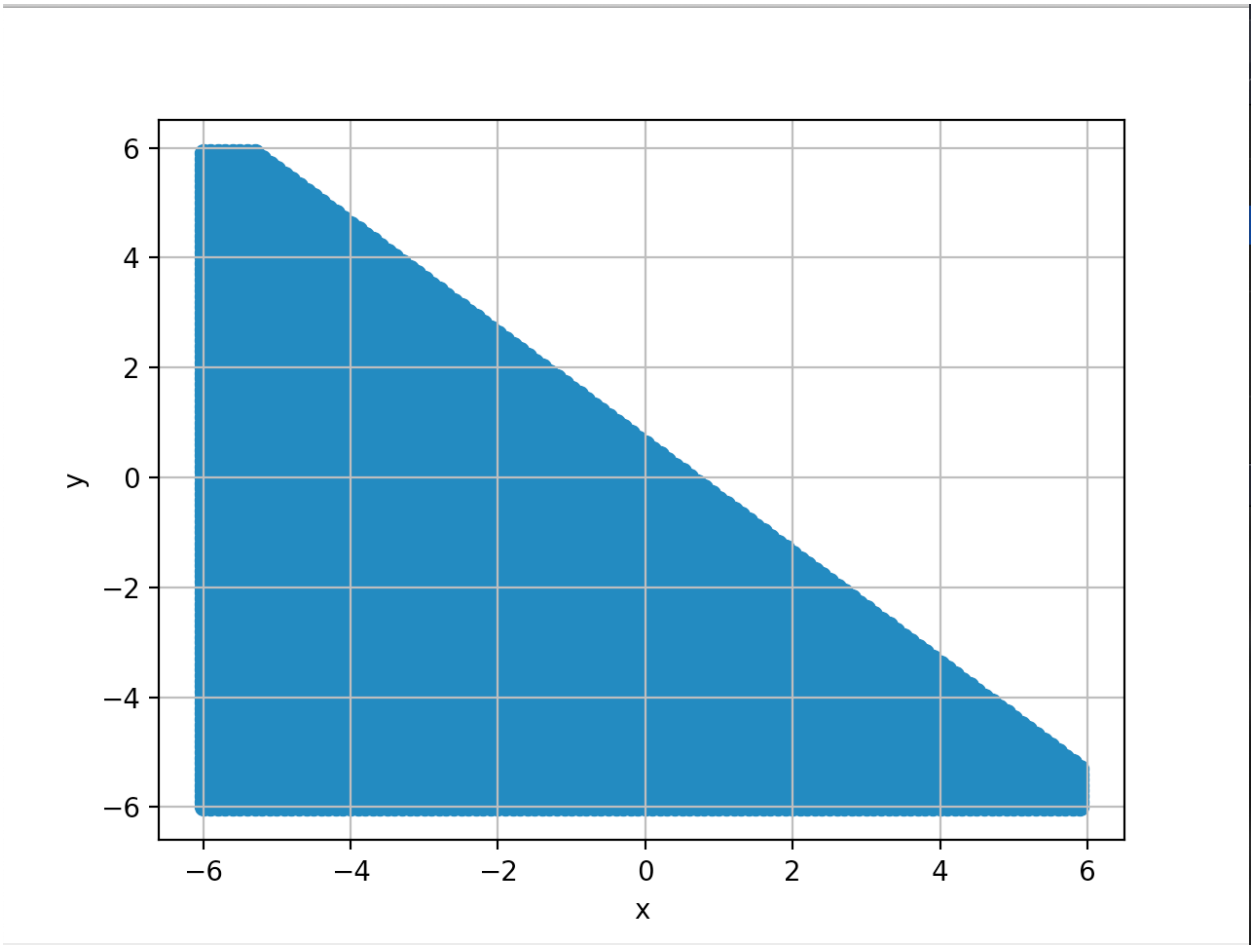
بخش اول تا سوم

برای این بخش کد هارا طبق فرمول هایی که در اسلاید های دکتر فدایی بود و با توجه به API ای که در اختیار بود انجام شد که در موقع تحویل خدمت TA محترم نشان داده میشود.

بخش چهارم

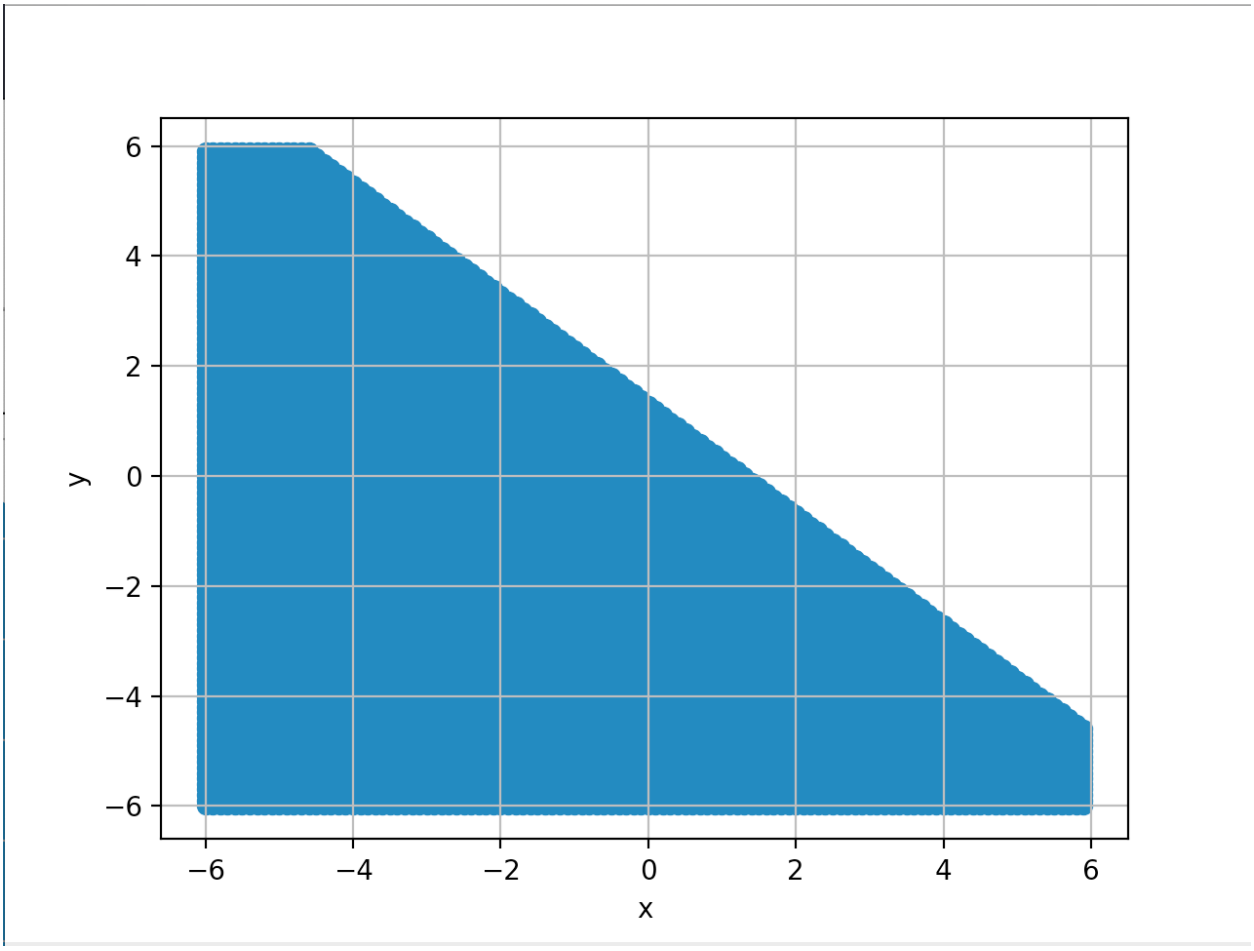
Basic Network

OR Data:




```
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]
Accuracy: 1.000000
----------------------------------------
```

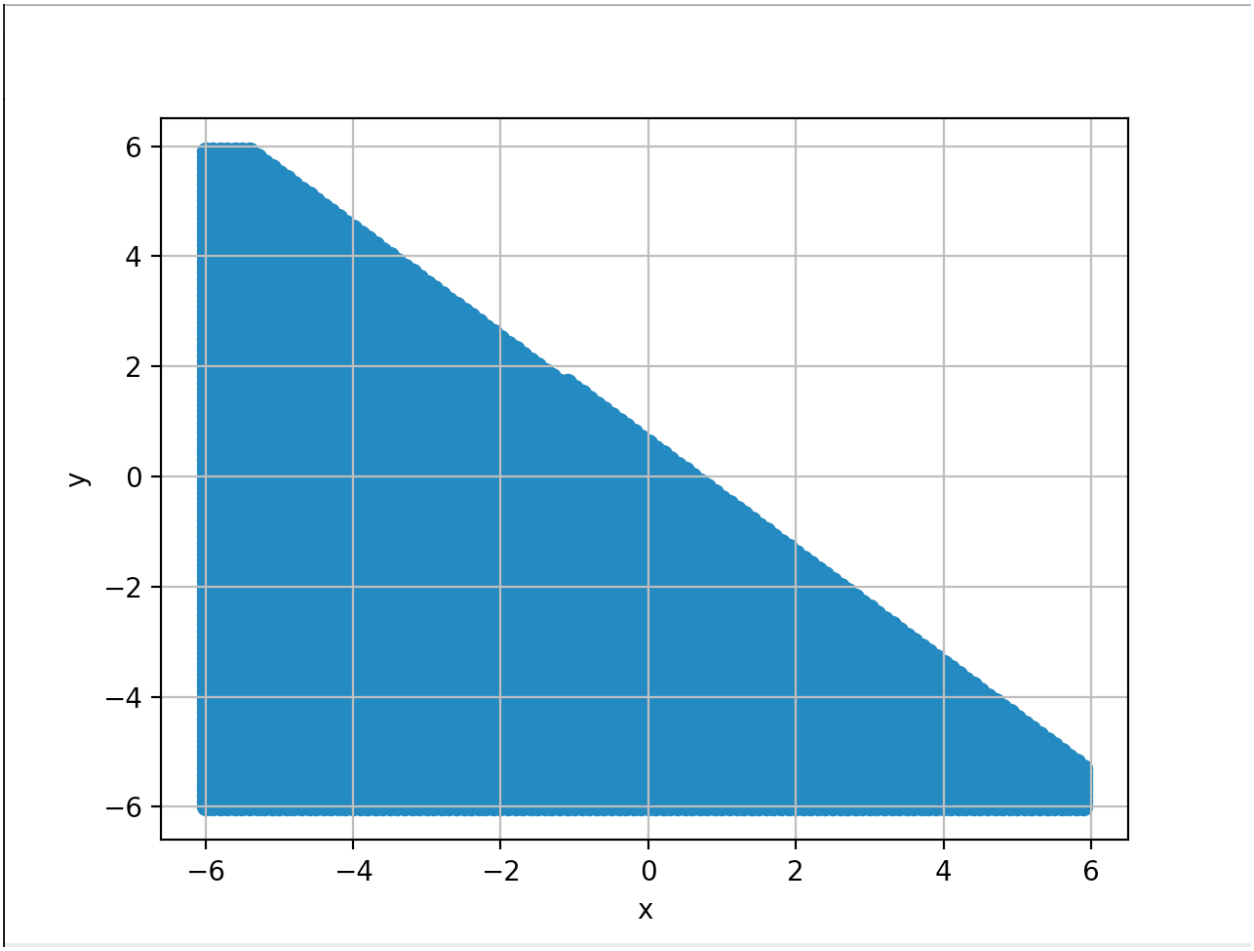
AND Data:




```
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: 1.000000
```

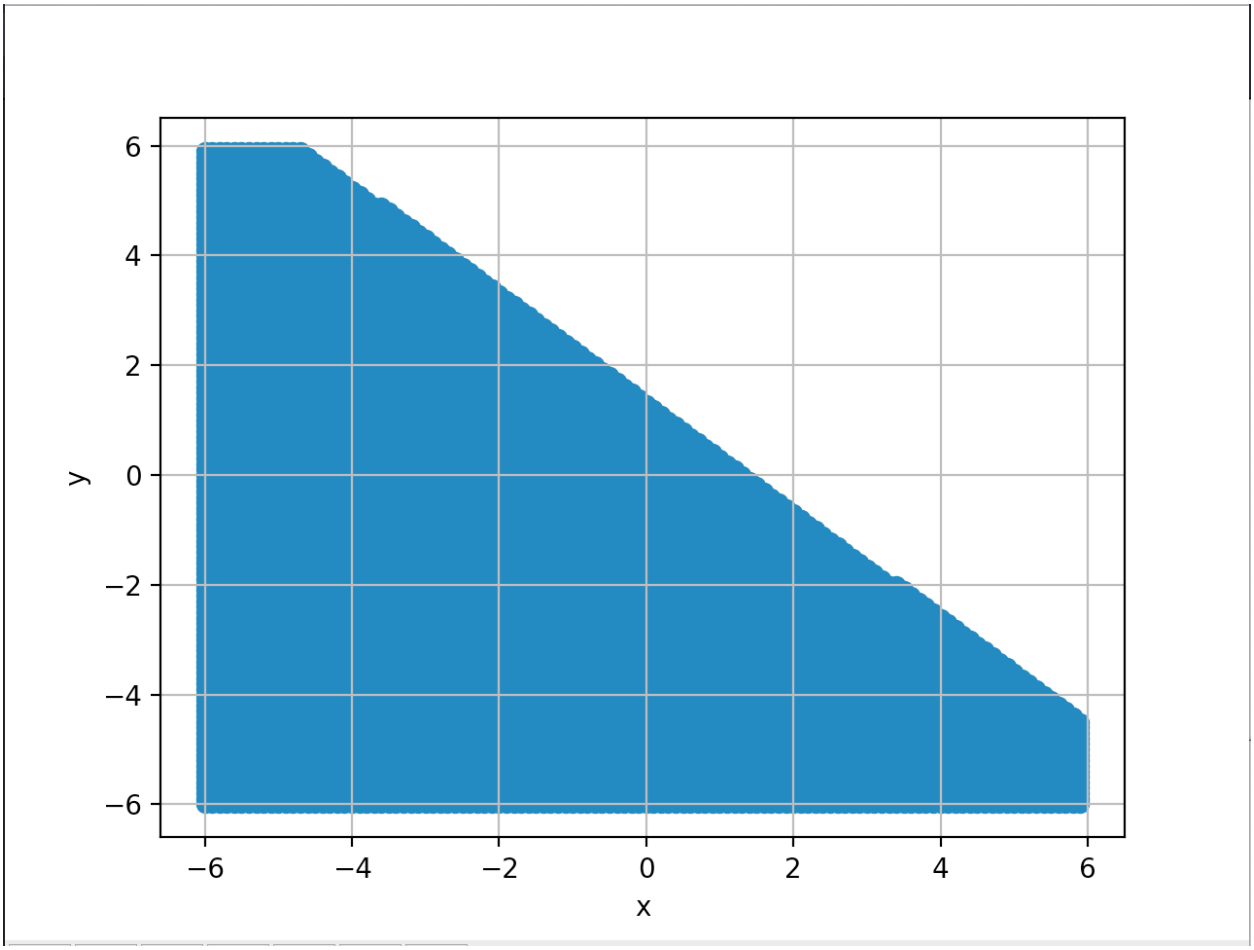
Two Layer Network

OR Data:



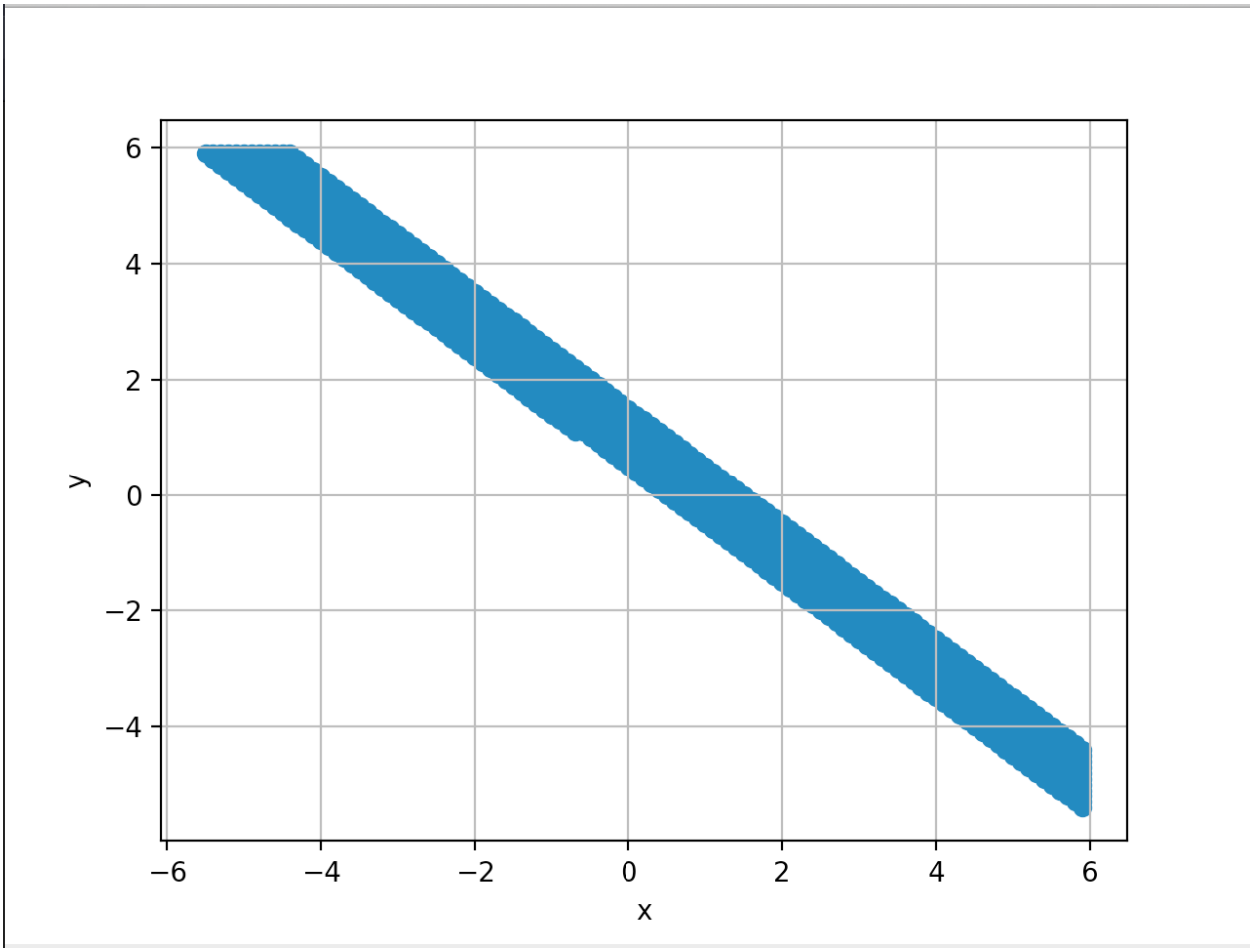

```
Testing on OR test-data
test((0.1, 0.1, 0)) returned: 0.011458520676734927 => 0 [correct]
test((0.1, 0.9, 1)) returned: 0.9822770971614414 => 1 [correct]
test((0.9, 0.1, 1)) returned: 0.9823401573678089 => 1 [correct]
test((0.9, 0.9, 1)) returned: 0.9960970721974146 => 1 [correct]
Accuracy: 1.000000
----------------------------------------
```

AND Data:



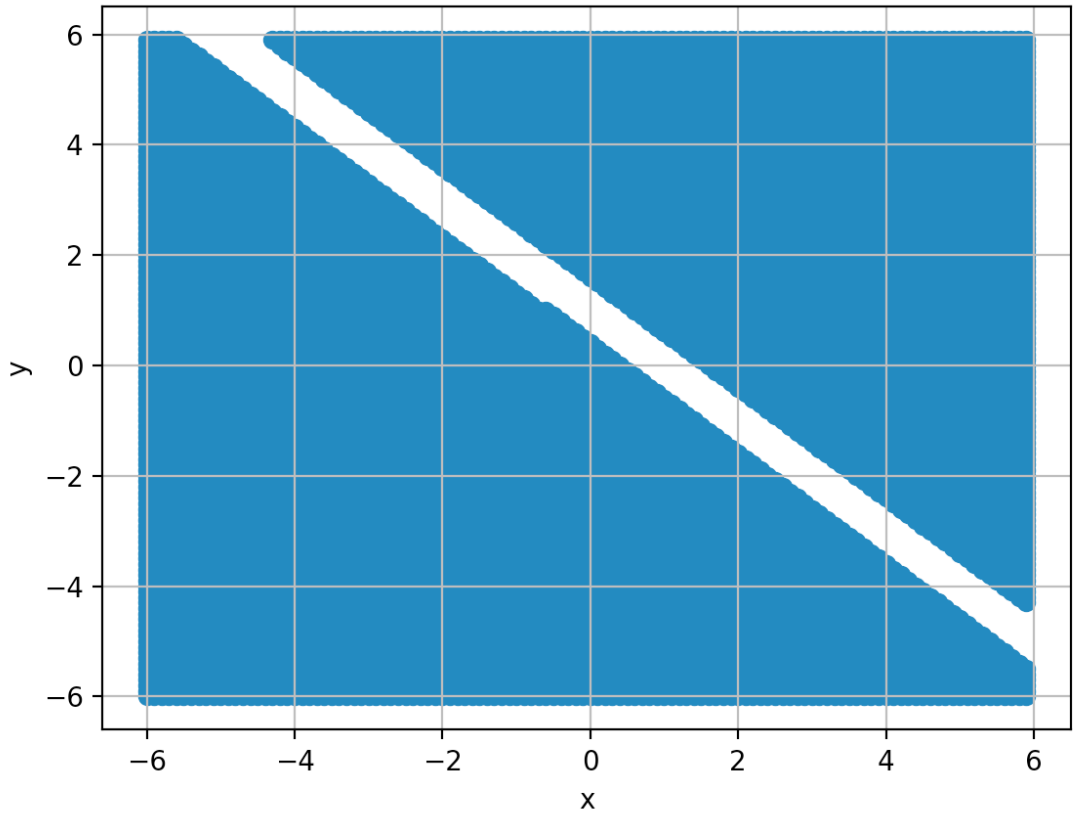

```
Testing on AND test-data
test((0.1, 0.1, 0)) returned: 0.0010270247812284438 => 0 [correct]
test((0.1, 0.9, 0)) returned: 0.015297096890893955 => 0 [correct]
test((0.9, 0.1, 0)) returned: 0.016243408521361885 => 0 [correct]
test((0.9, 0.9, 1)) returned: 0.985790853392045 => 1 [correct]
Accuracy: 1.000000
----------------------------------------
```

EQUAL Data:



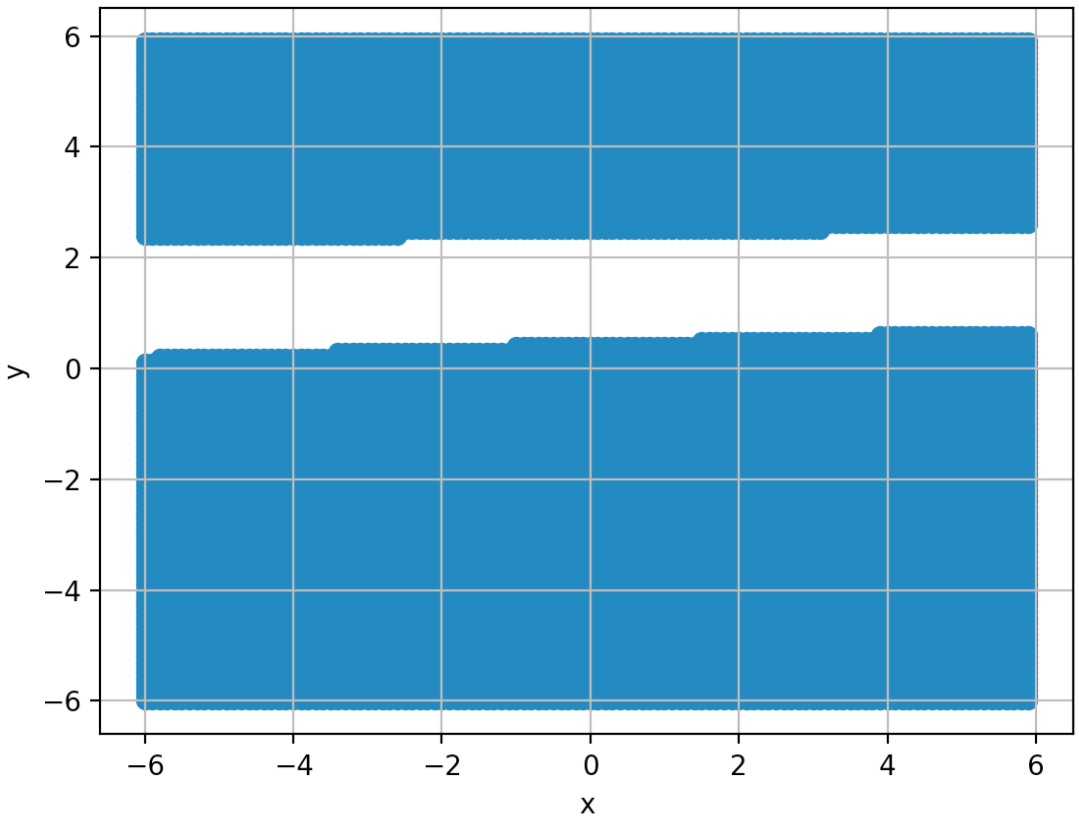

```
Testing on EQUAL test-data
test((0.1, 0.1, 1)) returned: 0.9520729744071752 => 1 [correct]
test((0.1, 0.9, 0)) returned: 0.013424564615154278 => 0 [correct]
test((0.9, 0.1, 0)) returned: 0.013451143532092036 => 0 [correct]
test((0.9, 0.9, 1)) returned: 0.9539830559940157 => 1 [correct]
Accuracy: 1.000000
----------------------------------------
```

NOT EQUAL Data:



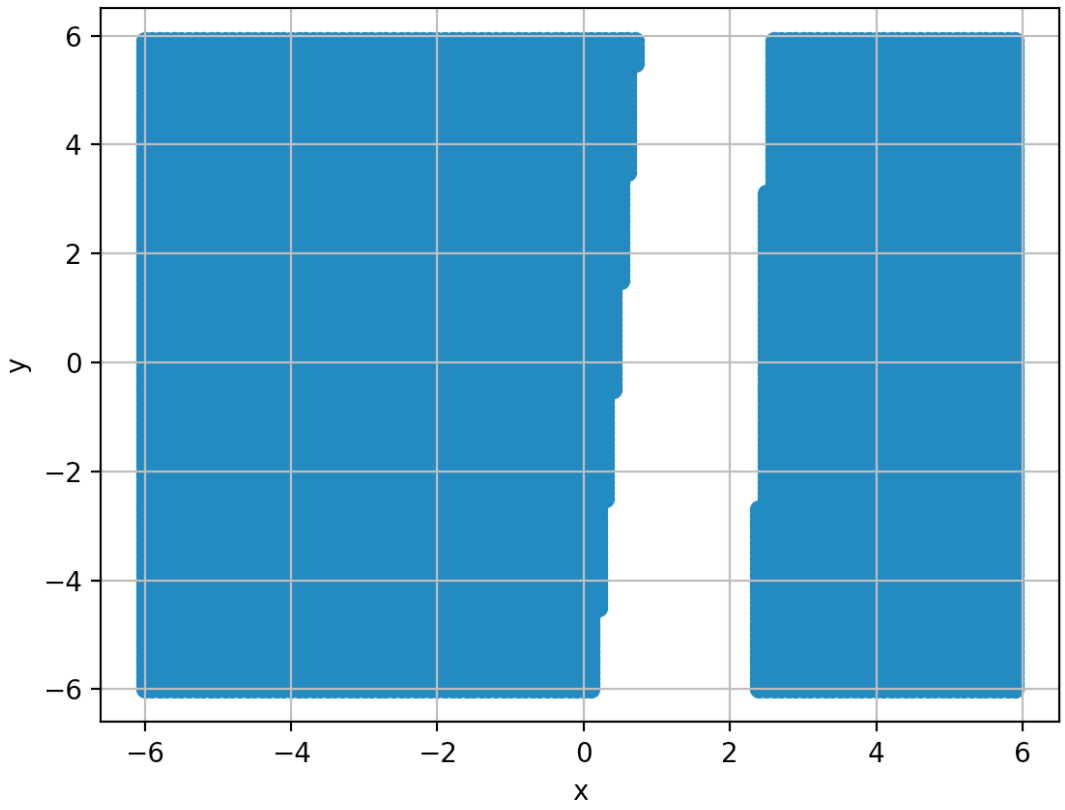

```
Testing on NOT_EQUAL test-data
test((0.1, 0.1, 0)) returned: 0.047927025592825016 => 0 [correct]
test((0.1, 0.9, 1)) returned: 0.9865754353848458 => 1 [correct]
test((0.9, 0.1, 1)) returned: 0.9865488564679079 => 1 [correct]
test((0.9, 0.9, 0)) returned: 0.046016944005984 22 => 0 [correct]
Accuracy: 1.000000
----------------------------------------
```

HORIZENTAL BANDS Data



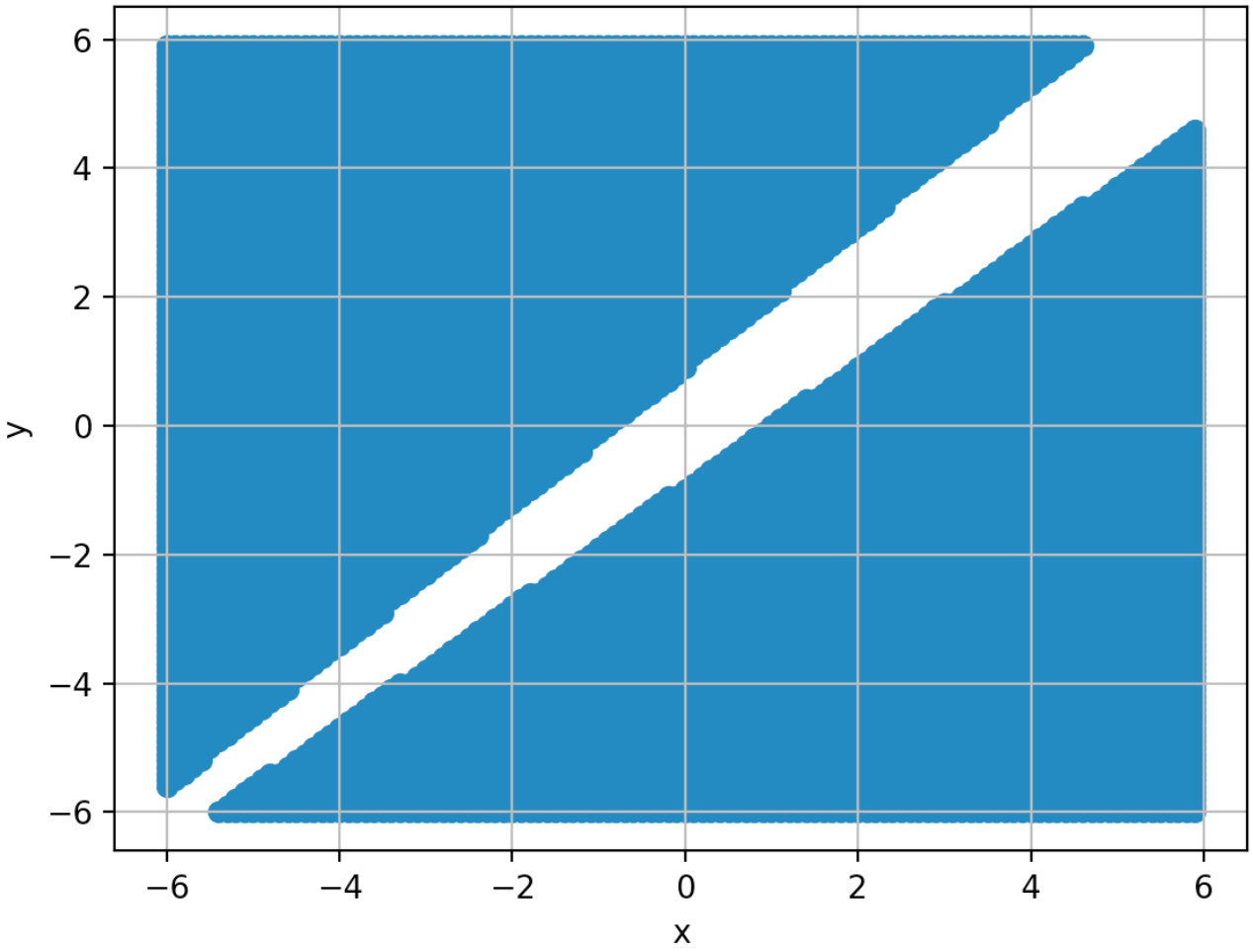

```
Testing on horizontal-bands test-data
test((1, 1.5, 1)) returned: 0.9944135265268136 => 1 [correct]
test((2, 1.5, 1)) returned: 0.9944581074357763 => 1 [correct]
test((3, 1.5, 1)) returned: 0.9944882096183628 => 1 [correct]
test((0, 1.5, 1)) returned: 0.9943558335398702 => 1 [correct]
test((4, 0, 0)) returned: 0.010079404461251365 => 0 [correct]
test((4, 4, 0)) returned: 0.006615667458921603 => 0 [correct]
test((-1, 0, 0)) returned: 0.016227066664505865 => 0 [correct]
test((-1, 4, 0)) returned: 0.006576922261332084 => 0 [correct]
Accuracy: 1.000000
----------------------------------------
```

VERTICAL BANDS Data



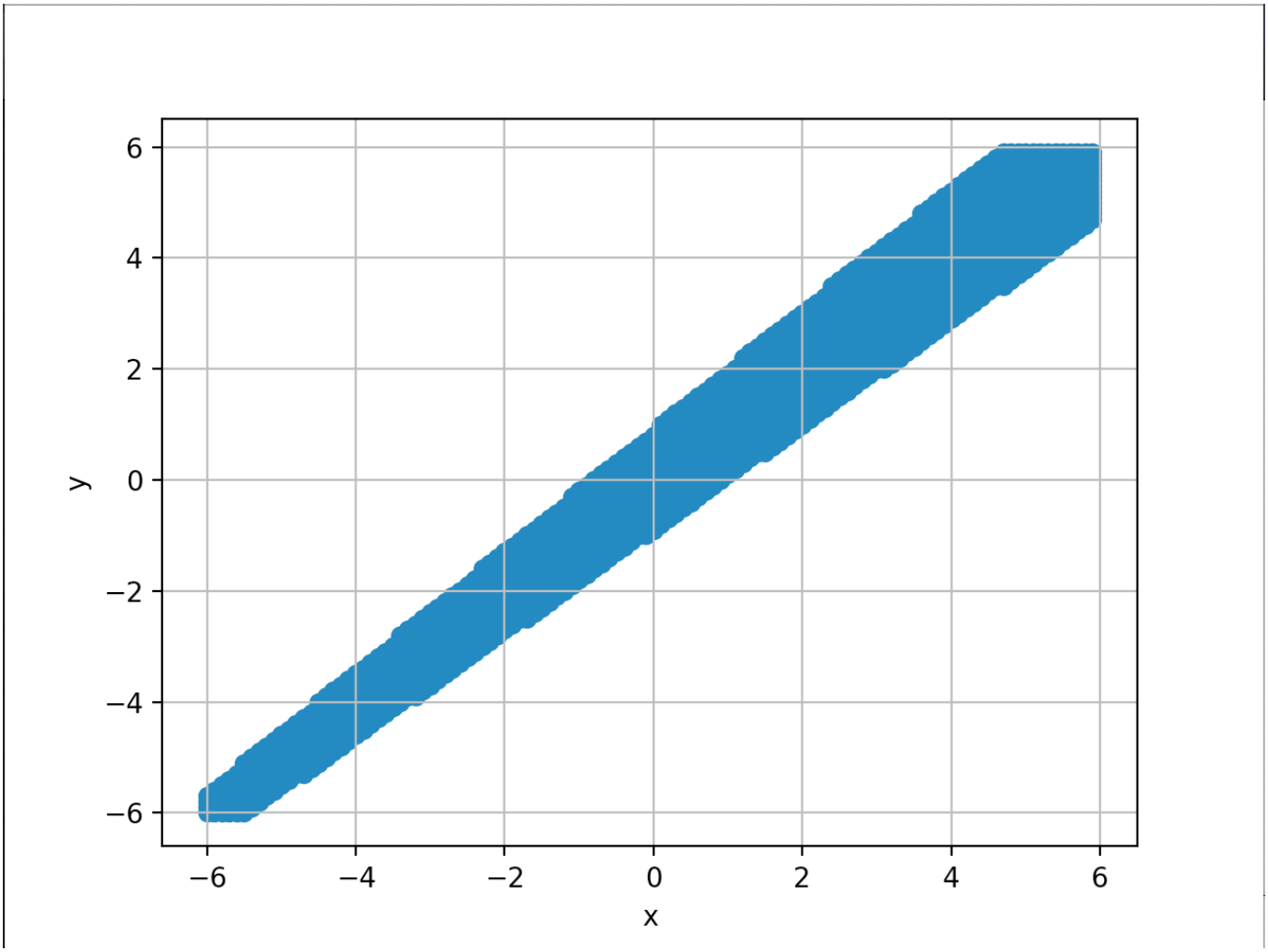

```
Testing on vertical-bands test-data
test((0, 1, 0)) returned: 0.012454278263057969 => 0 [correct]
test((0, 2, 0)) returned: 0.011284586999084011 => 0 [correct]
test((0, 1.5, 0)) returned: 0.011810324602925108 => 0 [correct]
test((1.5, 2, 1)) returned: 0.9945548266358012 => 1 [correct]
test((1.5, 5, 1)) returned: 0.9945828125647467 => 1 [correct]
test((1.5, 1, 1)) returned: 0.9945069388760096 => 1 [correct]
test((3, 1, 0)) returned: 0.01617649124064984 => 0 [correct]
test((3, 1.5, 0)) returned: 0.016698643433744005 => 0 [correct]
test((3, 2, 0)) returned: 0.017253470100973486 => 0 [correct]
test((1, 1.5, 1)) returned: 0.9931367419861905 => 1 [correct]
test((1, -1.5, 1)) returned: 0.9946594540432457 => 1 [correct]
test((2, 1.5, 1)) returned: 0.9825064971915265 => 1 [correct]
test((2, -1.5, 1)) returned: 0.9769631547773339 => 1 [correct]
test((4, 0, 0)) returned: 0.006251199048728452 => 0 [correct]
test((4, 4, 0)) returned: 0.006284807263018941 => 0 [correct]
test((-1, 0, 0)) returned: 0.00867735732560769 => 0 [correct]
test((-1, 4, 0)) returned: 0.008672839175147879 => 0 [correct]
Accuracy: 1.000000
----------------------------------------
```

DIAGONAL BANDS Data

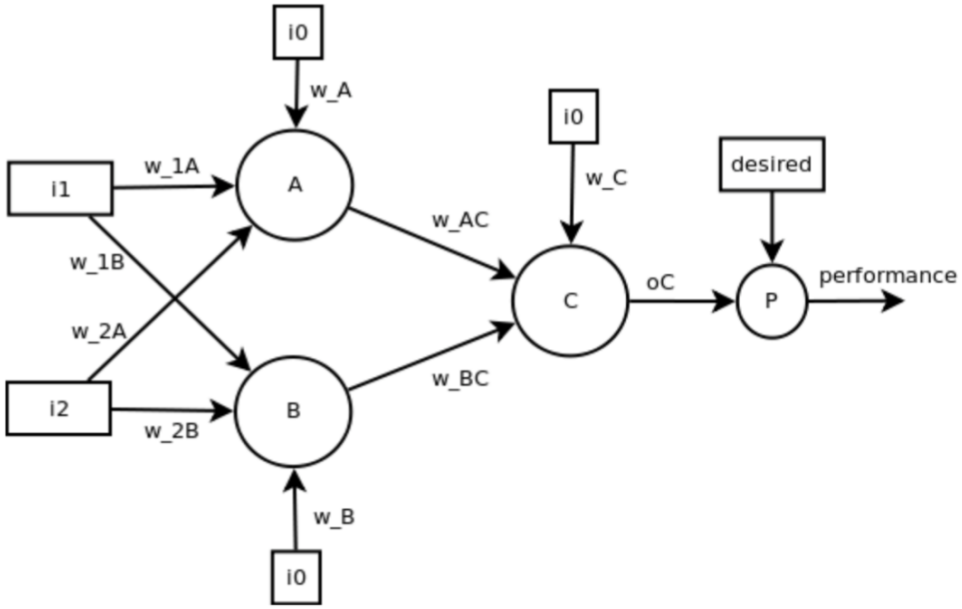



```
Testing on diagonal-band test-data
test((-1, -1, 1)) returned: 0.9751121193139393 => 1 [correct]
test((5, 5, 1)) returned: 0.9859149806773063 => 1 [correct]
test((-2, -2, 1)) returned: 0.9690895496445411 => 1 [correct]
test((6, 6, 1)) returned: 0.9863683403615392 => 1 [correct]
test((3.5, 3.5, 1)) returned: 0.984899667545948 => 1 [correct]
test((1.5, 1.5, 1)) returned: 0.9824384740296636 => 1 [correct]
test((4, 0, 0)) returned: 0.009518024958865631 => 0 [correct]
test((0, 4, 0)) returned: 0.012117598187331496 => 0 [correct]
Accuracy: 1.000000
----------------------------------------
```

INVERSE DIAGONAL BANDS Data




```
Testing on inverse-diagonal-band test-data
test((-1, -1, 0)) returned: 0.02488788068606079 => 0 [correct]
test((5, 5, 0)) returned: 0.014085019322693574 => 0 [correct]
test((-2, -2, 0)) returned: 0.030910450355458915 => 0 [correct]
test((6, 6, 0)) returned: 0.013631659638460905 => 0 [correct]
test((3.5, 3.5, 0)) returned: 0.015100332454052038 => 0 [correct]
test((1.5, 1.5, 0)) returned: 0.017561525970336417 => 0 [correct]
test((4, 0, 1)) returned: 0.9904819750411343 => 1 [correct]
test((0, 4, 1)) returned: 0.9878824018126686 => 1 [correct]
Accuracy: 1.000000
```




```
def make_neural_net_two_layer():
    """
    Create a 2-input, 1-output Network with three neurons.
    There should be two neurons at the first level, each receiving both inputs
    Both of the first level neurons should feed into the second layer neuron.
    See 'make_neural_net_basic' for required naming convention for inputs,
    weights, and neurons.
    """
    i0 = Input('i0', -1.0)
    i1 = Input('i1', 0.0)
    i2 = Input('i2', 0.0)

    seed_random()
    w_1A = Weight('w_1A', random_weight())
    w_1B = Weight('w_1B', random_weight())
    w_2A = Weight('w_2A', random_weight())
    w_2B = Weight('w_2B', random_weight())
    w_A = Weight('w_A', random_weight())
    w_B = Weight('w_B', random_weight())
    w_AC = Weight('w_AC', random_weight())
    w_BC = Weight('w_BC', random_weight())
    w_C = Weight('w_AC', random_weight())

    A = Neuron('A', [i0,i1,i2], [w_A,w_1A,w_2A])
    B = Neuron('B', [i0,i1,i2], [w_B,w_1B,w_2B])
    C = Neuron('C', [i0,A,B], [w_C,w_AC,w_BC])
    P = PerformanceElem(C, 0.0)

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

بخش پنجم

با استفاده از فرمول داده شده تابع زیر را مینویسم و برابری نسبی این دو عبارت را برای مثال اگر اختلاف آن ها کمتر از ۰.۰۰۰۱ باشد به ازای هر یال مقایسه میکنیم که در زمان اجرا این گونه به ما پاسخ میدهد:

$$\acute{f}(x) = \frac{f(x+\epsilon) - f(x)}{\varepsilon}$$

```python
def finite_difference(network):
    for weight in network.weights:
        network.clear_cache()
        prevois = network.performance.output()
        weight.set_value(weight.get_value() + 1e-8)
        network.clear_cache()
        new = network.performance.output()
        weight.set_value(weight.get_value() - 1e-8)
        ans = (new - prevois) / 1e-8
        if abs(network.performance.dOutdX(weight) - ans) < 1e-4:
            print("Almost same")
        else:
            print("Not same")
    network.clear_cache()
```

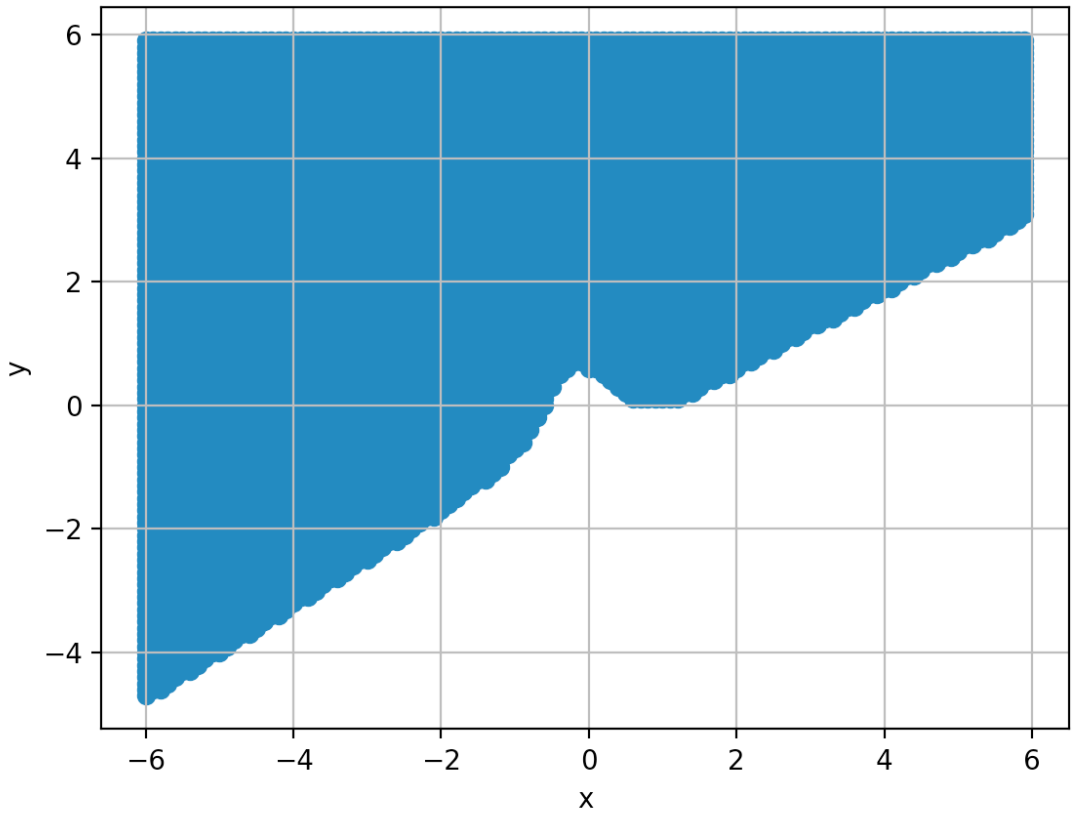
```
weights: [w_A(-3.03), w1_A(-5.19), w2_A(-5.22), w_B(0.75), w1_B(2.03), w2_B(1.98), w_AC(-2.19), w_AC(-9.23), w_BC(3.56)]
Almost same
Almost same
Almost same
Almost same
Almost same
Almost same
Almost same
Almost same
Almost same
```

بخش هشتم

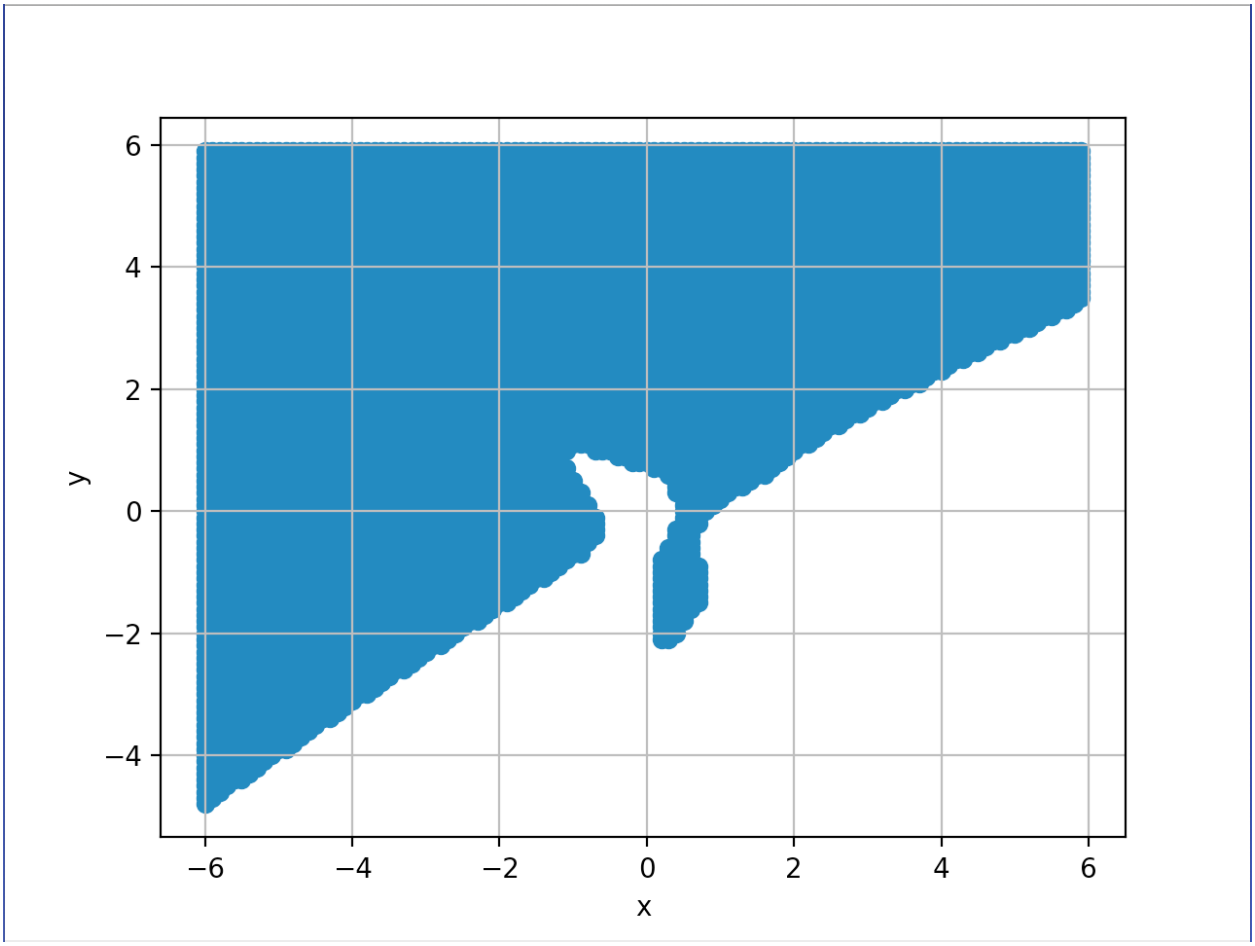
Two Moons Network

```
def make_neural_net_two_moons():
    """
    Create a 2-input, 1-output Network with three neurons.
    There should be two neurons at the first level, each receiving both inputs
    Both of the first level neurons should feed into the second layer neuron.
    See 'make_neural_net_basic' for required naming convention for inputs,
    weights, and neurons.
    """
    i0 = Input('i0', -1.0)
    i1 = Input('i1', 0.0)
    i2 = Input('i2', 0.0)
    seed_random()
    wA = []
    wB = []
    w0 = []
    wI = []
    M = []
    for i in range(0,40):
        wA.append(Weight('w'+str(i)+'A', random_weight()))
        wB.append(Weight('w'+str(i)+'B', random_weight()))
        w0.append(Weight('w'+str(i)+'0', random_weight()))
        wI.append(Weight('w'+str(i)+'I', random_weight()))
        M.append(Neuron('M'+str(i) , [i0,i1,i2] , [wI[i],wA[i],wB[i]]))
    woI = Weight('woI', random_weight())

    0 = Neuron('M'+str(i) , [i0] + M , [woI] + w0 )

    P = RegularizedPerformanceElem(0, 0.0)
    P.set_weights([woI] + w0 + wA + wB + wI)
    net = Network(P, M + [0])
    return net
```

100 Iteration:

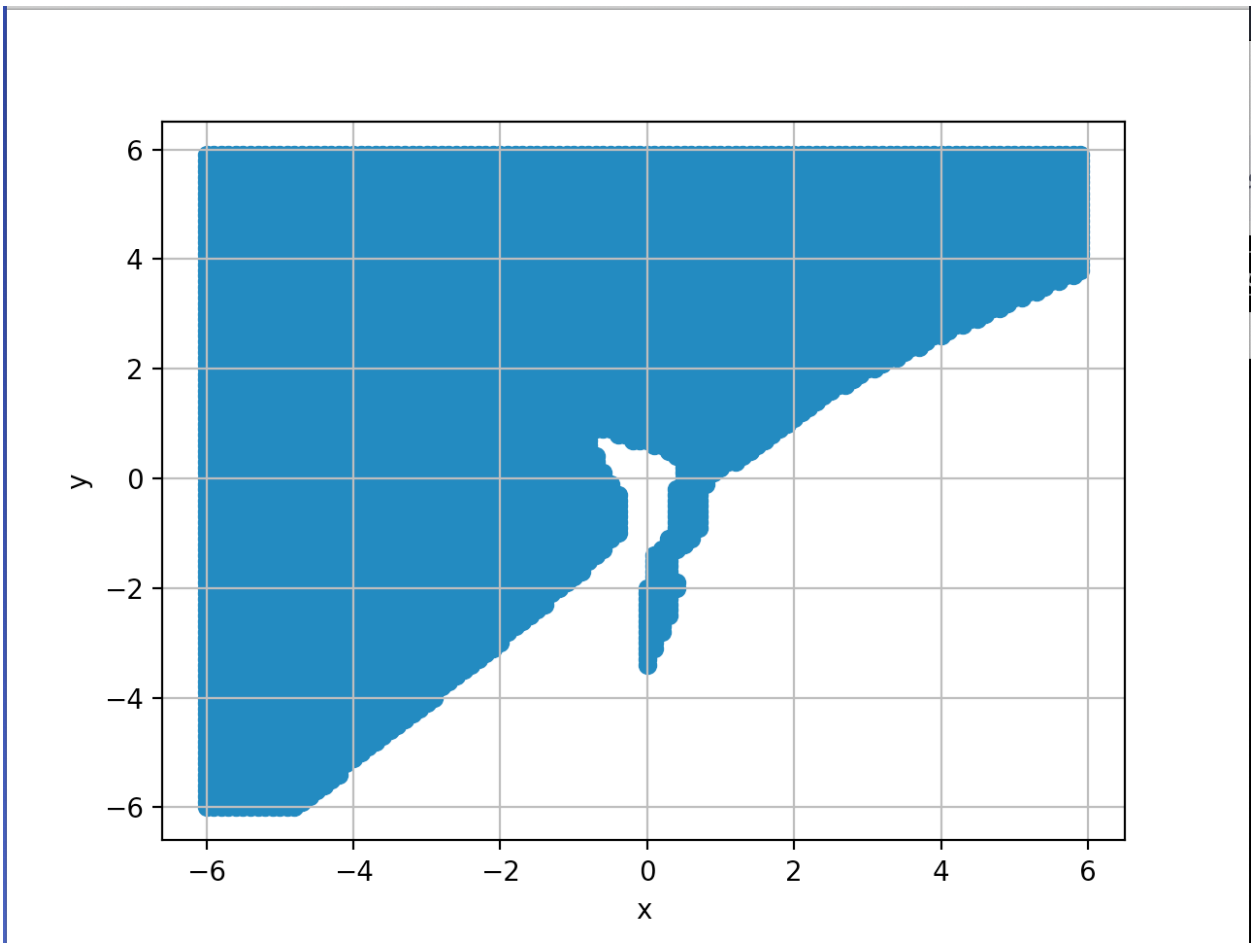


500 Iteration:




```
Accuracy: 0.780000
```

1000 Iteration:




```
Accuracy: 0.870000
```

بخش هشتم

Regularized Performance Element

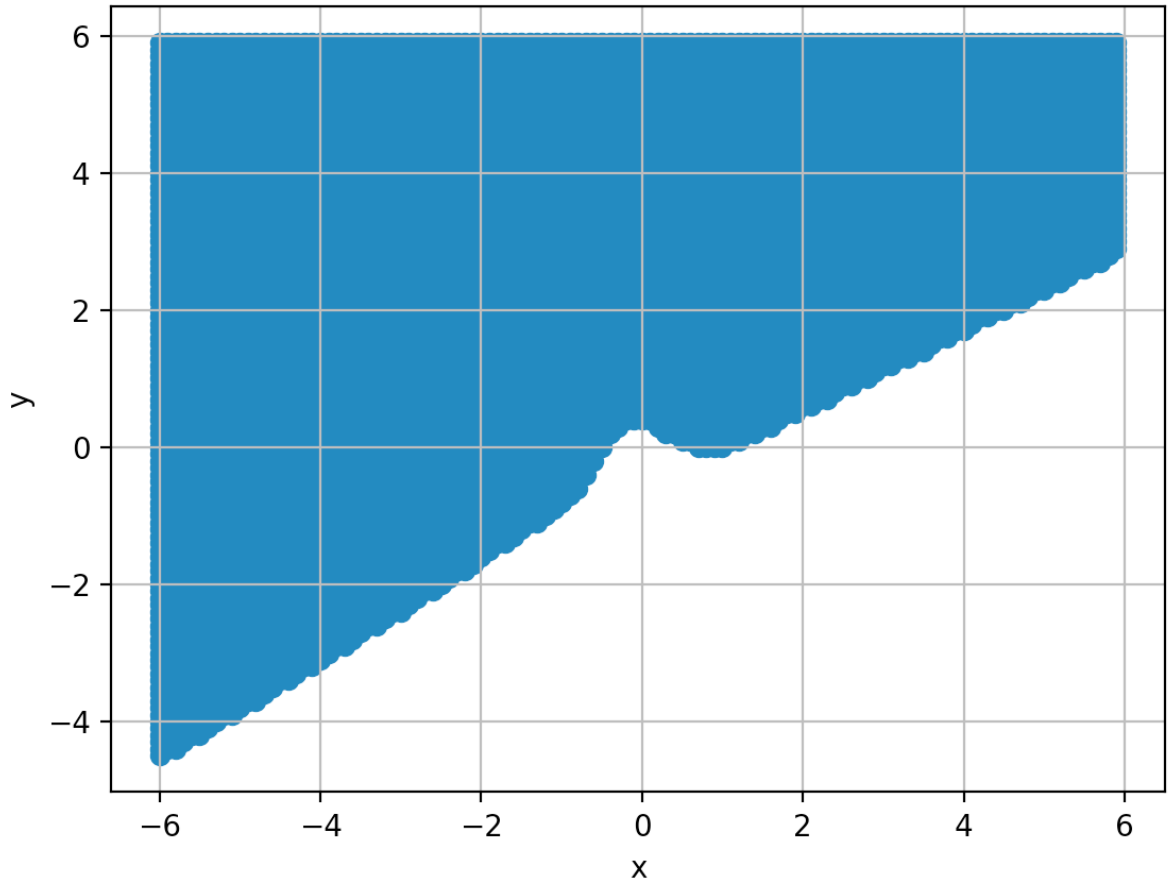
با استفاده از فرمولی که در لینک موجود بود تغییرات را بر روی المان پرفورمنس انجام میدهیم تا نهایتا به خروجی های زیر دست پیدا کنیم و برای دادن یالهای وزن به المان setter قرار دادیم و فرمول زیر را با تابع norm_2 محاسبه میکنیم:

$$J_{L2}(W,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \lambda \|w\|_2 \qquad \|w\|_2 = \sum_{j=1}^{n_x} w_j^2$$

و با مشورت دوستان به مقدار ۰.۰۰۰۱ برای لاندا رسیدیم.

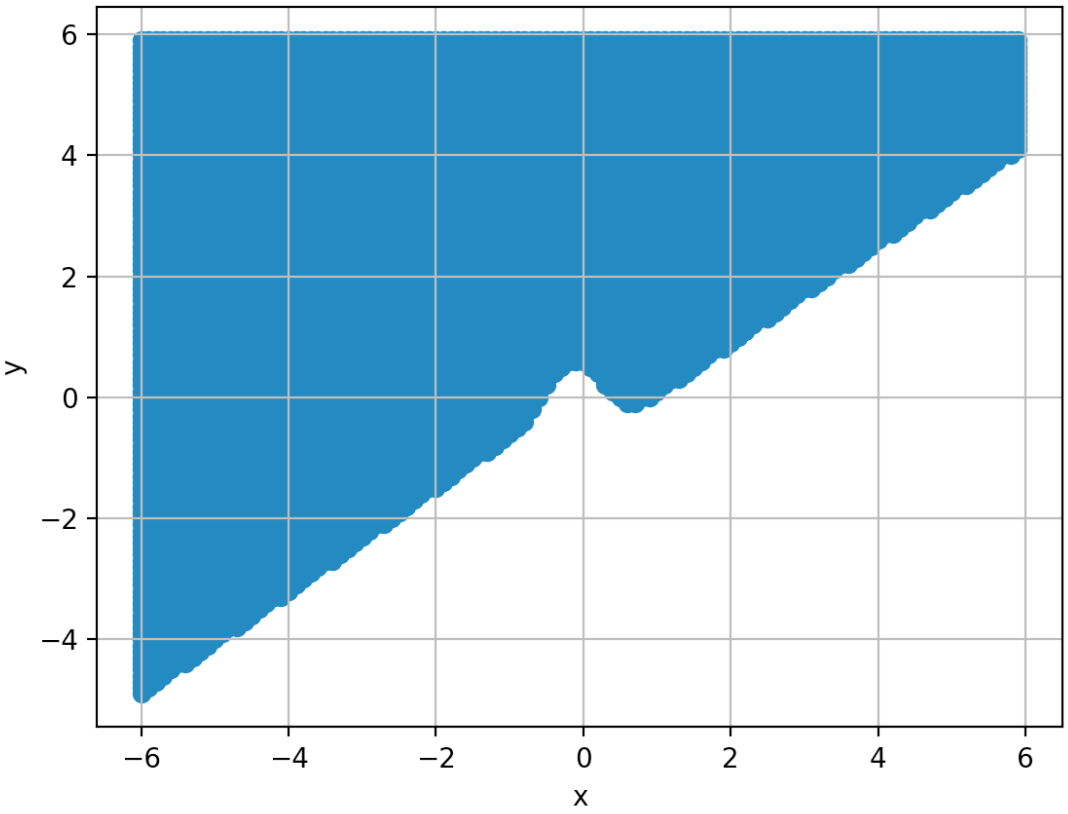
خروجی ها به صورت زیر است:

100 Iteration:



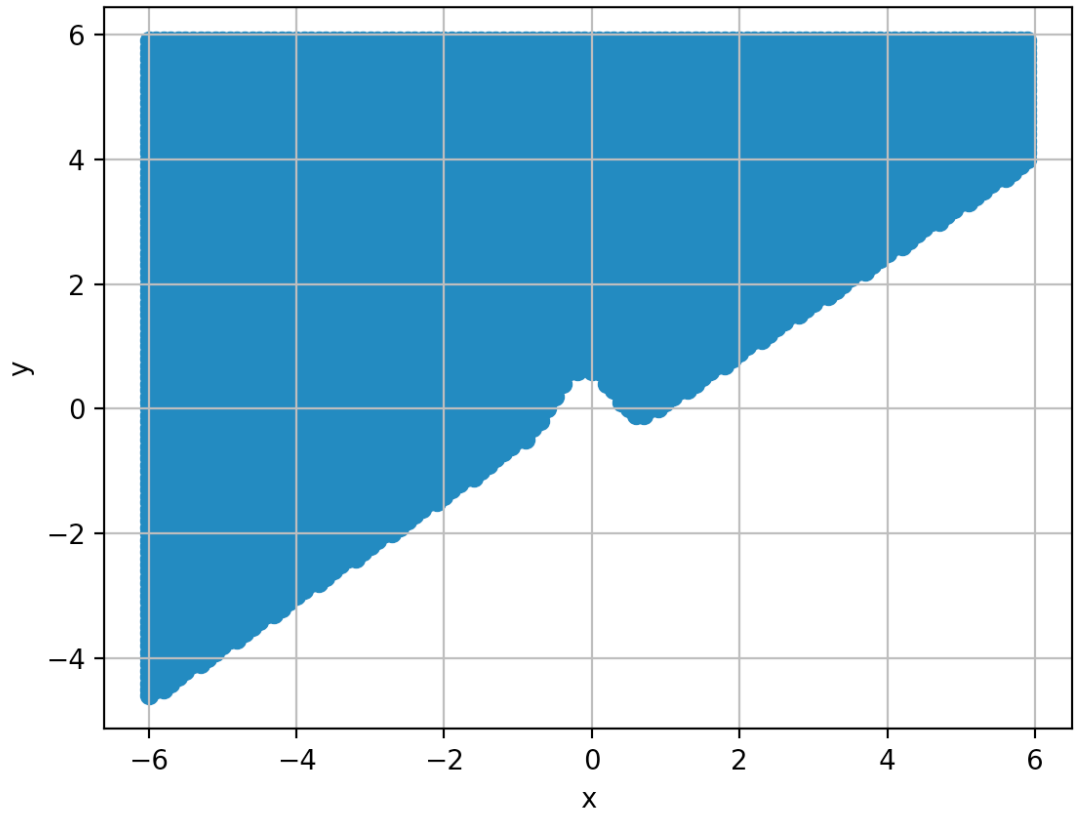

```
Accuracy: 0.980000
```

500 Iteration:




```
Accuracy: 0.970000
```

1000 Iteration:




```
Accuracy: 0.980000
```

نیتجه گیری

همانطور که از روی نمودار نیز قابل تشخیص است دقت تست بالاتر رفته است و بیرون زدگی نمودار از بین رفته است که میتوان نتیجه گرفت با این کار توانسته ایم مشکل Overfiffing شبکه را حل کنیم.