AI - CA5

Hadi Heidari Rad - 810197011

Input Node:

the output() is simply the exact input value, and dOutdX() is 0 for all values because its a constant input.

Neuron Node:

Each neuron, gets a list of inputs and a list of weights, and the output is calculated by:

$$y_k = \phi(\sum_{j=1}^{m} w_{kj}x_j)$$

in which w_j and x_j are respectively corresponding weights and inputs, and ϕ is called an activation function, which is used to hold a threshold on output on neuron and specify classification. In this project we use sigmoid activation function which is:

In [2]:
```
def sigmoid(x):
    return 1 / (1 + math.exp(-x))
```

So that the output is now calculated:

In [ ]:
```
def compute_output(self):
    out = 0
    for i in range(0, len(self.get_inputs())):
        out += (self.get_inputs()[i].output() * self.get_weights()[i].get_value())
    return sigmoid(out)
```

By calculating the derivative of sigmoid function, we notice that the result can be written in a form of the original function:

$$\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$$

In [ ]:
```
def compute_doutdx(self, elem):
    dsigmoid_dout = self.output()*(1 - self.output())
    ans = 0
    if self.has_weight(elem):
        ans = self.get_inputs()[self.get_weights().index(elem)].output()
    else:
        for i in range(len(self.get_weights())):
            if self.isa_descendant_weight_of(elem, self.my_weights[i]):
                ans += (self.my_weights[i].get_value() * self.get_inputs()[i].dOutdX(elem))
    return ans * dsigmoid_dout
```

Performance Node

After the whole neurons have been done, the output needs to pass through a performance element to measure loss/performance and give a feedback (back propagation) to update weights. Each performance element gets a desired value and an input value and calculates output and derivative as below:

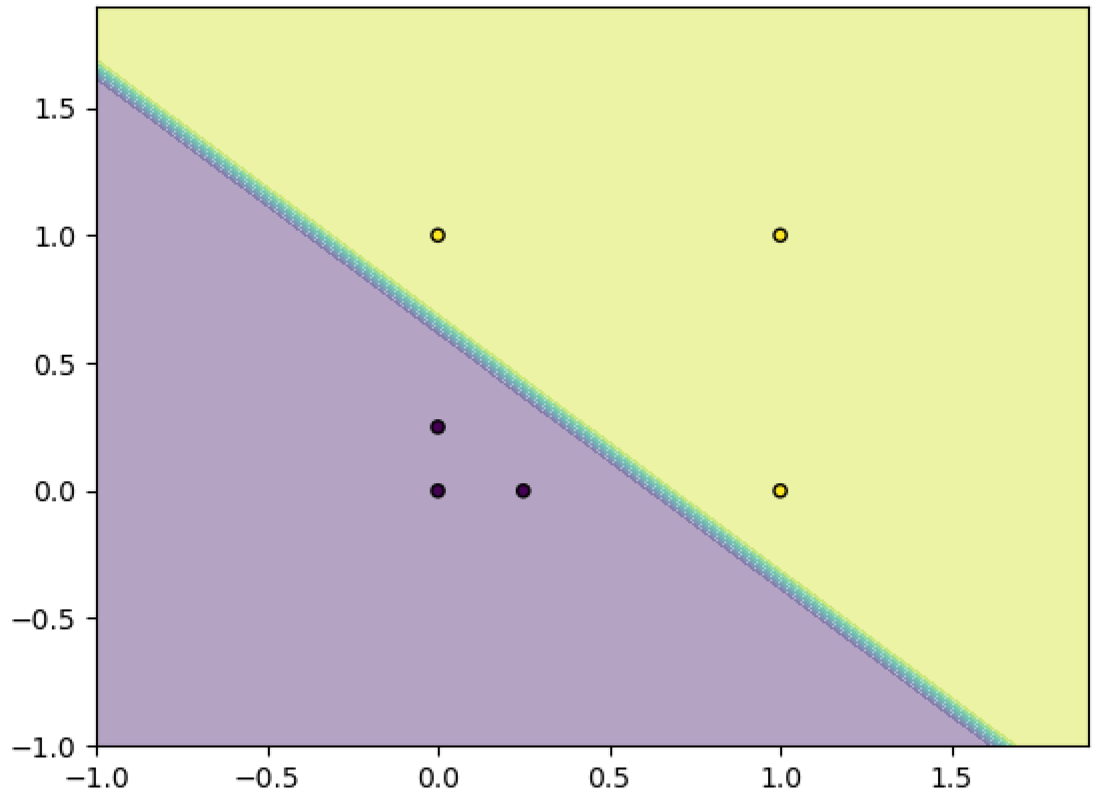
In [ ]:
```
def output(self):
    return -0.5 * (self.my_desired_val - self.get_input().output()) ** 2

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

Two Layer:

The code for two layer neural net is below, but there's a point here, there were no order given for generating random weights, which is I think neccessary when there is a seed-setting function. otherwise using a seed is useless! And not for all orders the given network for the given data gets a 1 accuracy; anyways, the order that I have written here, gets the accuracy = 1

In [ ]:
```
def make_neural_net_two_layer():
    i0 = Input('i0', -1.0)
    i1 = Input('i1', 0.0)
    i2 = Input('i2', 0.0)

    seed_random()

    w1A = Weight('w1A', random_weight())
    w1B = Weight('w1B', random_weight())
    w2A = Weight('w2A', random_weight())
    w2B = Weight('w2B', random_weight())
    wA = Weight('wA', random_weight())
    wB = Weight('wB', random_weight())
    wC = Weight('wC', random_weight())
    wAC = Weight('wAC', random_weight())
    wBC = Weight('wBC', random_weight())

    A = Neuron('A', [i0, i1, i2], [wA, w1A, w2A])
    B = Neuron('B', [i0, i1, i2], [wB, w1B, w2B])
    C = Neuron('C', [i0, A, B], [wC, wAC, wBC])

    P = PerformanceElem(C, 0.0)

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

    return net
```

Training on OR data
weights: [wA(-3.03), w1A(-5.19), w2A(-5.22), wB(0.75), w1B(2.03), w2B(1.98), wC(-2.19), wAC(-9.23), wBC(3.56)]
weight wA: finite-diff: -0.0003 dOutdX(w): -0.0003 TRUE
weight w1A: finite-diff: 0.0000 dOutdX(w): -0.0000 TRUE
weight w2A: finite-diff: 0.0001 dOutdX(w): 0.0001 TRUE
weight wB: finite-diff: 0.0002 dOutdX(w): 0.0002 TRUE
weight w1B: finite-diff: 0.0000 dOutdX(w): -0.0000 TRUE
weight w2B: finite-diff: -0.0001 dOutdX(w): -0.0001 TRUE
weight wC: finite-diff: 0.0003 dOutdX(w): 0.0003 TRUE
weight wAC: finite-diff: -0.0002 dOutdX(w): -0.0002 TRUE
weight wBC: finite-diff: -0.0001 dOutdX(w): -0.0001 TRUE
9 weights matched with finite diff, and 0 diverged.
Trained weights:
Weight 'wA': -3.033595
Weight 'w1A': -5.191019
Weight 'w2A': -5.221685
Weight 'wB': 0.751714
Weight 'w1B': 2.026931
Weight 'w2B': 1.977390
Weight 'wC': -2.194035
Weight 'wAC': -9.229062
Weight 'wBC': 3.558560
Testing on OR test-data
Accuracy: 1.000000



Training on AND data

weights: [wA(-6.46), w1A(-4.88), w2A(-5.07), wB(-1.12), w1B(-1.99), w2B(-1.37), wC(-5.38), wAC(-10.49), wBC(-2.65)]

weight wA: finite-diff: 0.0003 dOutdX(w): 0.0003 TRUE

weight w1A: finite-diff: -0.0003 dOutdX(w): -0.0003 TRUE

weight w2A: finite-diff: -0.0002 dOutdX(w): -0.0002 TRUE

weight wB: finite-diff: 0.0001 dOutdX(w): 0.0001 TRUE

weight w1B: finite-diff: -0.0001 dOutdX(w): -0.0001 TRUE

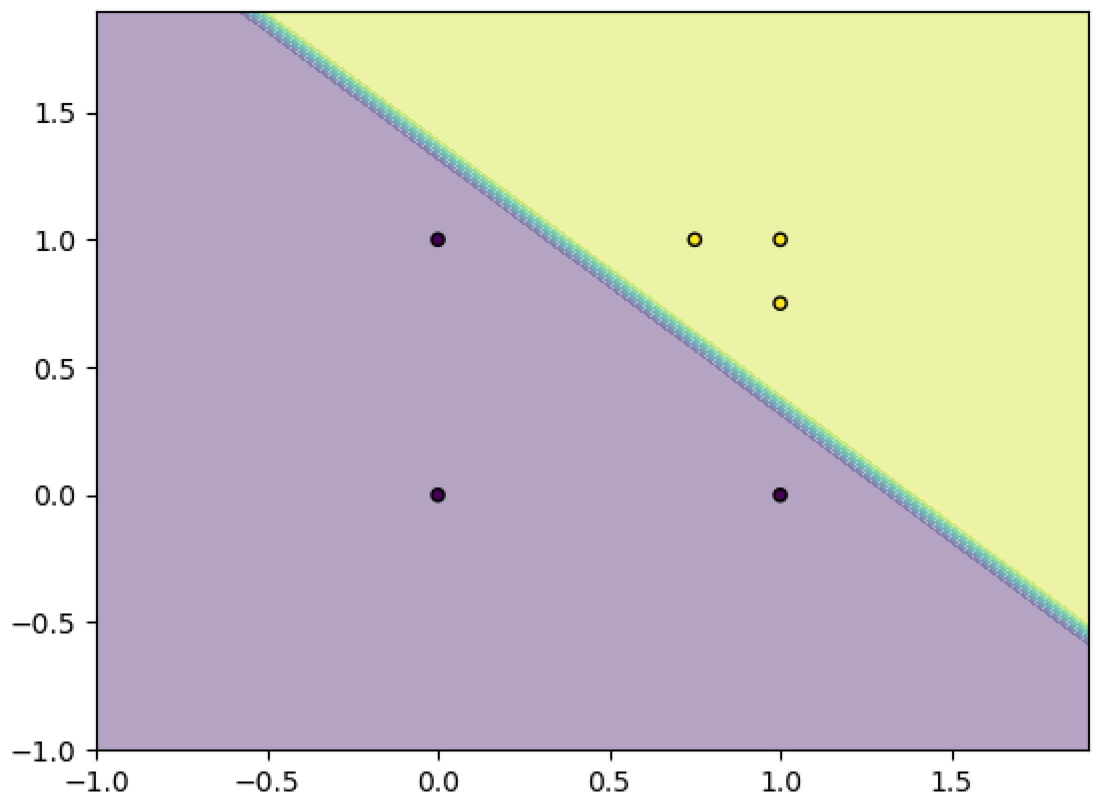
weight w2B: finite-diff: -0.0001 dOutdX(w): -0.0001 TRUE

weight wC: finite-diff: -0.0003 dOutdX(w): -0.0003 TRUE

weight wAC: finite-diff: 0.0000 dOutdX(w): 0.0000 TRUE

weight wBC: finite-diff: 0.0000 dOutdX(w): 0.0000 TRUE

9 weights matched with finite diff, and 0 diverged.
Trained weights:
Weight 'wA': -6.462356
Weight 'w1A': -4.882233
Weight 'w2A': -5.073080
Weight 'wB': -1.117736
Weight 'w1B': -1.985283
Weight 'w2B': -1.367818
Weight 'wC': -5.382025
Weight 'wAC': -10.485835
Weight 'wBC': -2.652808
Testing on AND test-data
Accuracy: 1.000000



Training on EQUAL data

weights: [wA(-2.79), w1A(-6.75), w2A(-6.78), wB(-7.31), w1B(-4.91), w2B(-4.91), wC(-4.85), wAC(10.27), wBC(-10.18)]

weight wA: finite-diff: -0.0000 dOutdX(w): -0.0000 TRUE

weight w1A: finite-diff: 0.0000 dOutdX(w): 0.0000 TRUE

weight w2A: finite-diff: 0.0000 dOutdX(w): 0.0000 TRUE

weight wB: finite-diff: 0.0002 dOutdX(w): 0.0002 TRUE

weight w1B: finite-diff: -0.0002 dOutdX(w): -0.0002 TRUE

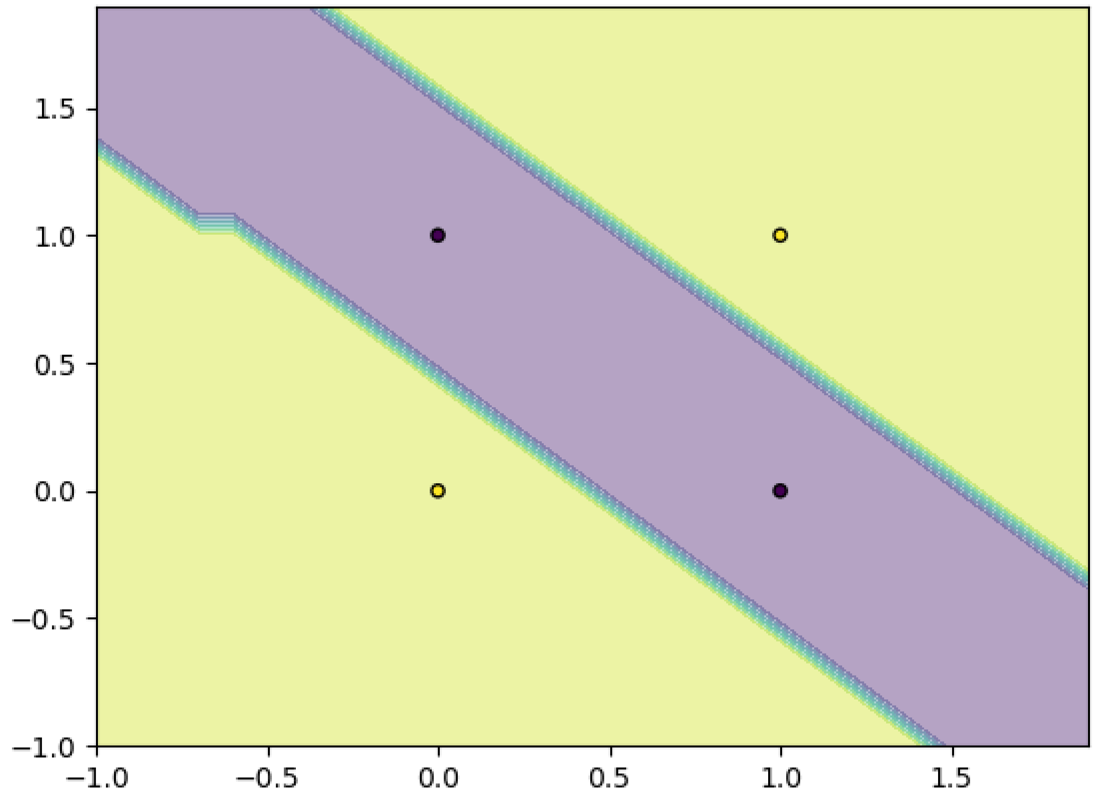
weight w2B: finite-diff: -0.0002 dOutdX(w): -0.0002 TRUE

weight wC: finite-diff: -0.0003 dOutdX(w): -0.0003 TRUE

weight wAC: finite-diff: 0.0000 dOutdX(w): 0.0000 TRUE

weight wBC: finite-diff: 0.0000 dOutdX(w): 0.0000 TRUE

9 weights matched with finite diff, and 0 diverged.
Trained weights:
Weight 'wA': -2.794549
Weight 'w1A': -6.747456
Weight 'w2A': -6.781582
Weight 'wB': -7.307357
Weight 'w1B': -4.905921
Weight 'w2B': -4.910897
Weight 'wC': -4.846049
Weight 'wAC': 10.269060
Weight 'wBC': -10.179957
Testing on EQUAL test-data
Accuracy: 1.000000



Training on NOT_EQUAL data

weights: [wA(-2.79), w1A(-6.75), w2A(-6.78), wB(-7.31), w1B(-4.91), w2B(-4.91), wC(4.85), wAC(-10.27), wBC(10.18)]

weight wA: finite-diff: -0.0000 dOutdX(w): -0.0000 TRUE

weight w1A: finite-diff: 0.0000 dOutdX(w): 0.0000 TRUE

weight w2A: finite-diff: 0.0000 dOutdX(w): 0.0000 TRUE

weight wB: finite-diff: 0.0002 dOutdX(w): 0.0002 TRUE

weight w1B: finite-diff: -0.0002 dOutdX(w): -0.0002 TRUE

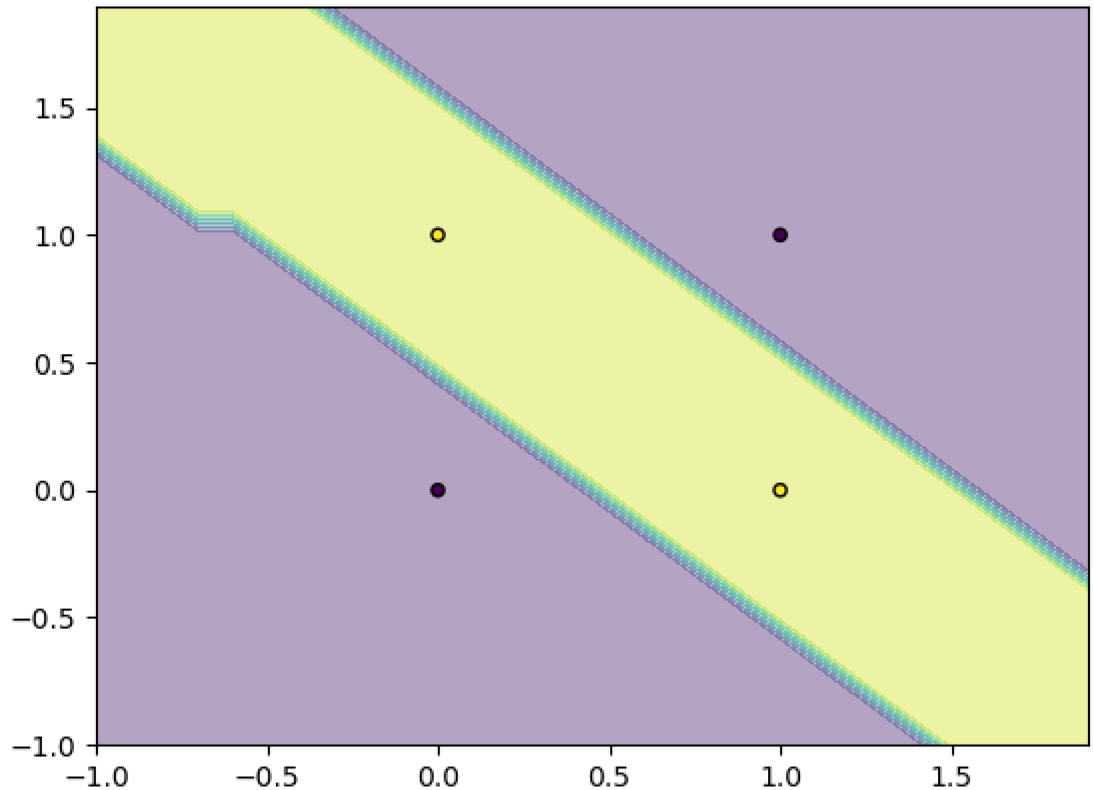
weight w2B: finite-diff: -0.0002 dOutdX(w): -0.0002 TRUE

weight wC: finite-diff: 0.0003 dOutdX(w): 0.0003 TRUE

weight wAC: finite-diff: -0.0000 dOutdX(w): -0.0000 TRUE

weight wBC: finite-diff: -0.0000 dOutdX(w): -0.0000 TRUE

9 weights matched with finite diff, and 0 diverged.
Trained weights:
Weight 'wA': -2.794549
Weight 'w1A': -6.747456
Weight 'w2A': -6.781582
Weight 'wB': -7.307357
Weight 'w1B': -4.905921
Weight 'w2B': -4.910897
Weight 'wC': 4.846049
Weight 'wAC': -10.269060
Weight 'wBC': 10.179957
Testing on NOT_EQUAL test-data
Accuracy: 1.000000



Training on horizontal-bands data

weights: [wA(-2.90), w1A(0.26), w2A(-6.30), wB(-10.19), w1B(0.07), w2B(-4.20), wC(5.03), wAC(-10.17), wBC(10.42)]

weight wA: finite-diff: -0.0000 dOutdX(w): -0.0000 TRUE

weight w1A: finite-diff: 0.0000 dOutdX(w): 0.0000 TRUE

weight w2A: finite-diff: 0.0000 dOutdX(w): 0.0000 TRUE

weight wB: finite-diff: 0.0003 dOutdX(w): 0.0003 TRUE

weight w1B: finite-diff: -0.0010 dOutdX(w): -0.0010 TRUE

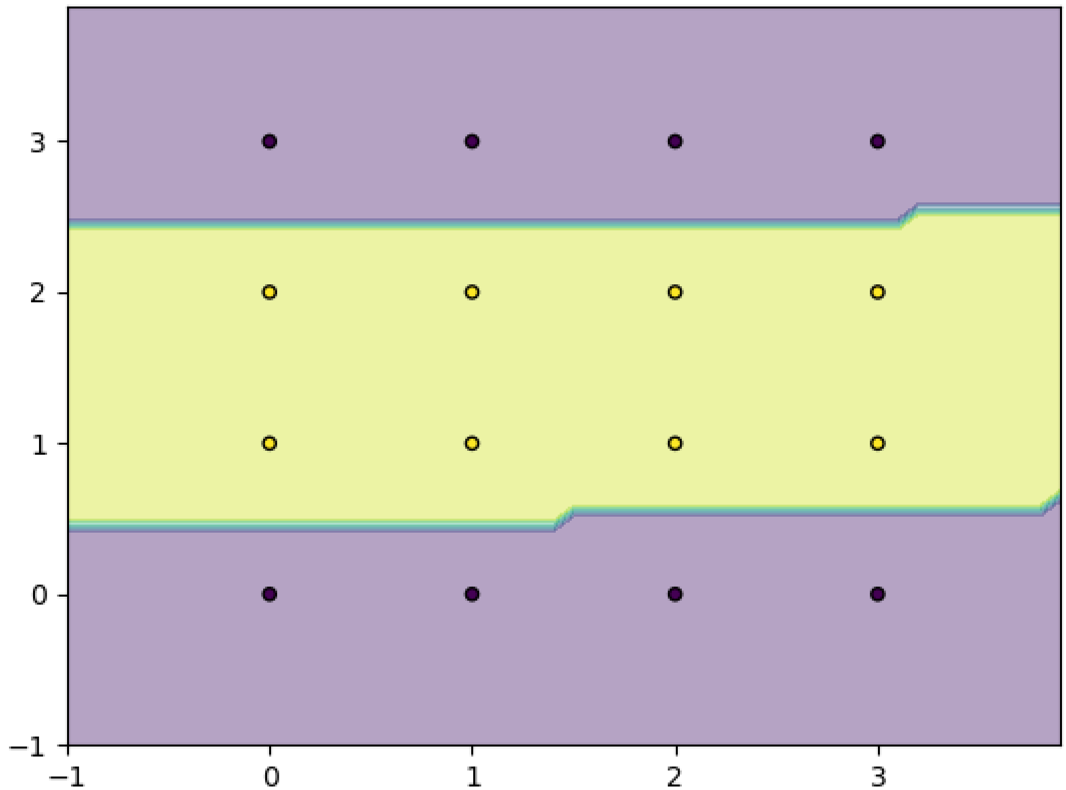
weight w2B: finite-diff: -0.0010 dOutdX(w): -0.0010 TRUE

weight wC: finite-diff: 0.0003 dOutdX(w): 0.0003 TRUE

weight wAC: finite-diff: -0.0000 dOutdX(w): -0.0000 TRUE

weight wBC: finite-diff: -0.0000 dOutdX(w): -0.0000 TRUE

9 weights matched with finite diff, and 0 diverged.
Trained weights:
Weight 'wA': -2.901654
Weight 'w1A': 0.259752
Weight 'w2A': -6.303266
Weight 'wB': -10.188501
Weight 'w1B': 0.074259
Weight 'w2B': -4.196649
Weight 'wC': 5.030765
Weight 'wAC': -10.174927
Weight 'wBC': 10.424870
Testing on horizontal-bands test-data
Accuracy: 1.000000



Training on vertical-bands data

weights: [wA(-10.01), w1A(-4.12), w2A(0.07), wB(-2.96), w1B(-6.60), w2B(0.33), wC(5.08), wAC(10.50), wBC(-10.16)]

weight wA: finite-diff: 0.0003 dOutdX(w): 0.0003 TRUE

weight w1A: finite-diff: -0.0010 dOutdX(w): -0.0010 TRUE

weight w2A: finite-diff: -0.0010 dOutdX(w): -0.0010 TRUE

weight wB: finite-diff: -0.0000 dOutdX(w): -0.0000 TRUE

weight w1B: finite-diff: 0.0000 dOutdX(w): 0.0000 TRUE

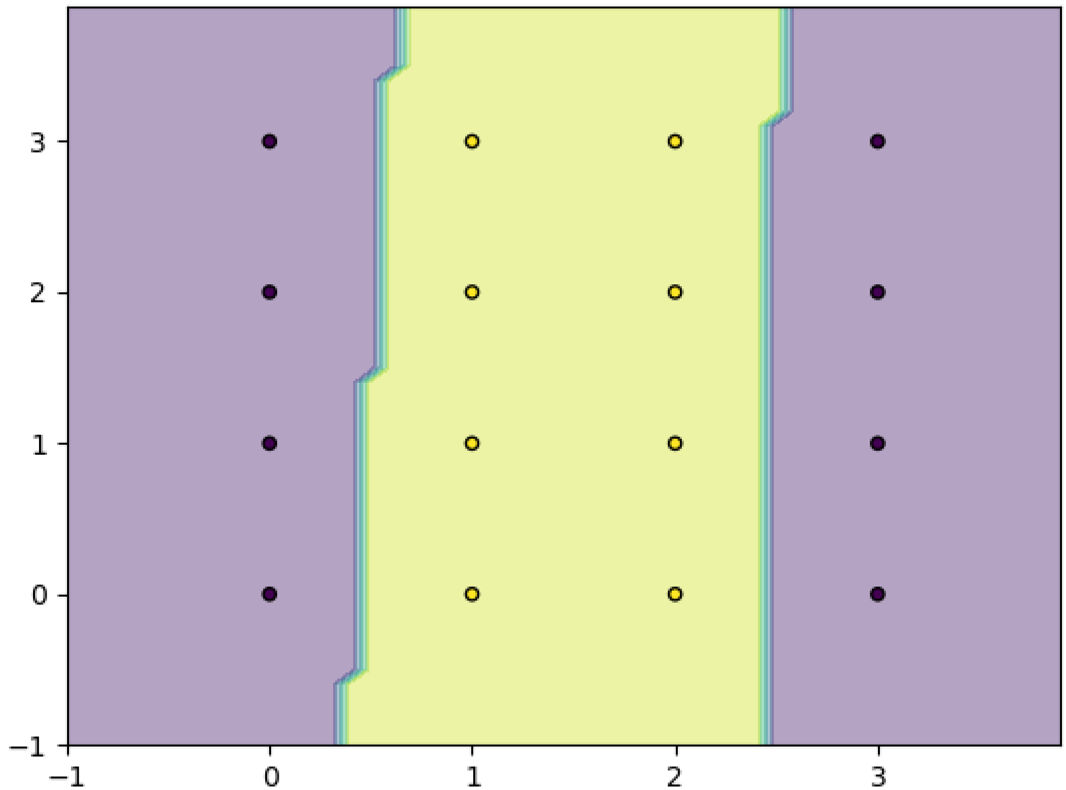
weight w2B: finite-diff: 0.0000 dOutdX(w): 0.0000 TRUE

weight wC: finite-diff: 0.0003 dOutdX(w): 0.0003 TRUE

weight wAC: finite-diff: -0.0000 dOutdX(w): -0.0000 TRUE

weight wBC: finite-diff: -0.0000 dOutdX(w): -0.0000 TRUE

9 weights matched with finite diff, and 0 diverged.
Trained weights:
Weight 'wA': -10.014296
Weight 'w1A': -4.121186
Weight 'w2A': 0.071837
Weight 'wB': -2.957541
Weight 'w1B': -6.595310
Weight 'w2B': 0.328358
Weight 'wC': 5.084953
Weight 'wAC': 10.504953
Weight 'wBC': -10.159036
Testing on vertical-bands test-data
Accuracy: 1.000000



Training on diagonal-band data

weights: [wA(3.63), w1A(3.67), w2A(-3.92), wB(-3.29), w1B(3.97), w2B(-3.66), wC(4.40), wAC(-9.03), wBC(8.79)]

weight wA: finite-diff: -0.0001 dOutdX(w): -0.0001 TRUE

weight w1A: finite-diff: 0.0002 dOutdX(w): 0.0002 TRUE

weight w2A: finite-diff: 0.0001 dOutdX(w): 0.0001 TRUE

weight wB: finite-diff: 0.0000 dOutdX(w): 0.0000 TRUE

weight w1B: finite-diff: -0.0000 dOutdX(w): -0.0000 TRUE

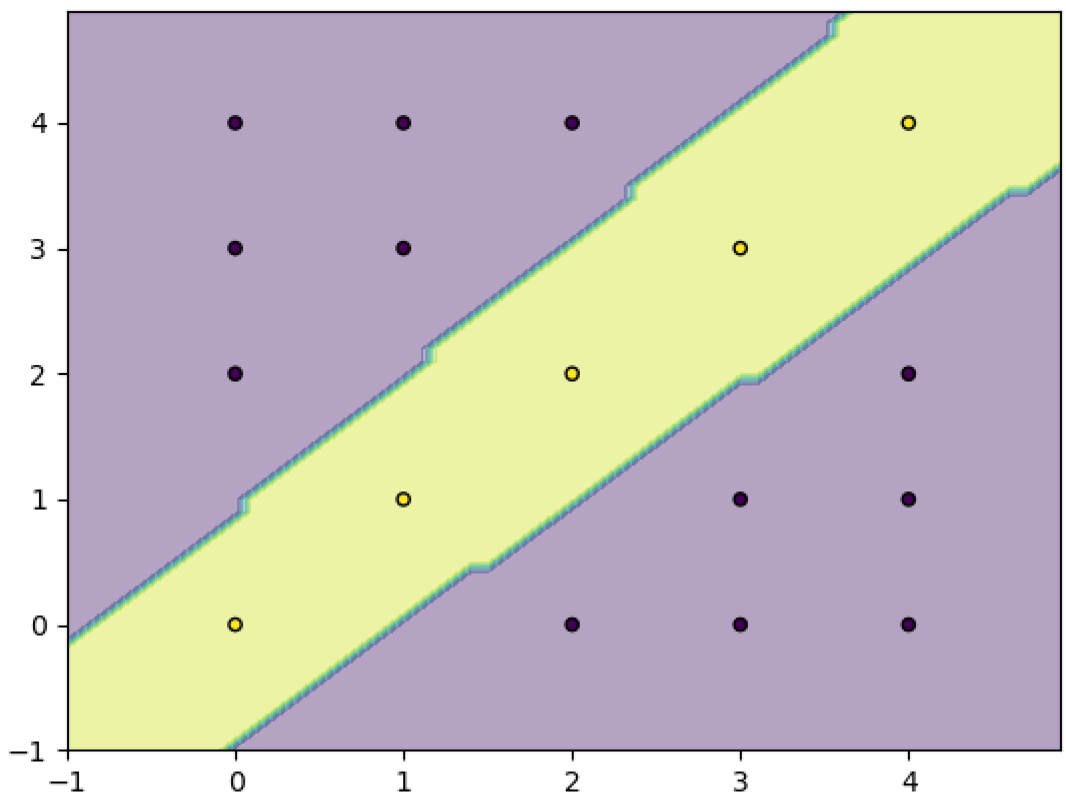
weight w2B: finite-diff: -0.0000 dOutdX(w): -0.0000 TRUE

weight wC: finite-diff: 0.0002 dOutdX(w): 0.0002 TRUE

weight wAC: finite-diff: -0.0002 dOutdX(w): -0.0002 TRUE

weight wBC: finite-diff: -0.0002 dOutdX(w): -0.0002 TRUE

9 weights matched with finite diff, and 0 diverged.
Trained weights:
Weight 'wA': 3.632720
Weight 'w1A': 3.673811
Weight 'w2A': -3.917853
Weight 'wB': -3.294863
Weight 'w1B': 3.967260
Weight 'w2B': -3.655719
Weight 'wC': 4.401011
Weight 'wAC': -9.031638
Weight 'wBC': 8.787502
Testing on diagonal-band test-data
Accuracy: 1.000000



Training on inverse-diagonal-band data

weights: [wA(3.63), w1A(3.67), w2A(-3.92), wB(-3.29), w1B(3.97), w2B(-3.66), wC(-4.40), wAC(9.03), wBC(-8.79)]

weight wA: finite-diff: -0.0001 dOutdX(w): -0.0001 TRUE

weight w1A: finite-diff: 0.0002 dOutdX(w): 0.0002 TRUE

weight w2A: finite-diff: 0.0001 dOutdX(w): 0.0001 TRUE

weight wB: finite-diff: 0.0000 dOutdX(w): 0.0000 TRUE

weight w1B: finite-diff: -0.0000 dOutdX(w): -0.0000 TRUE

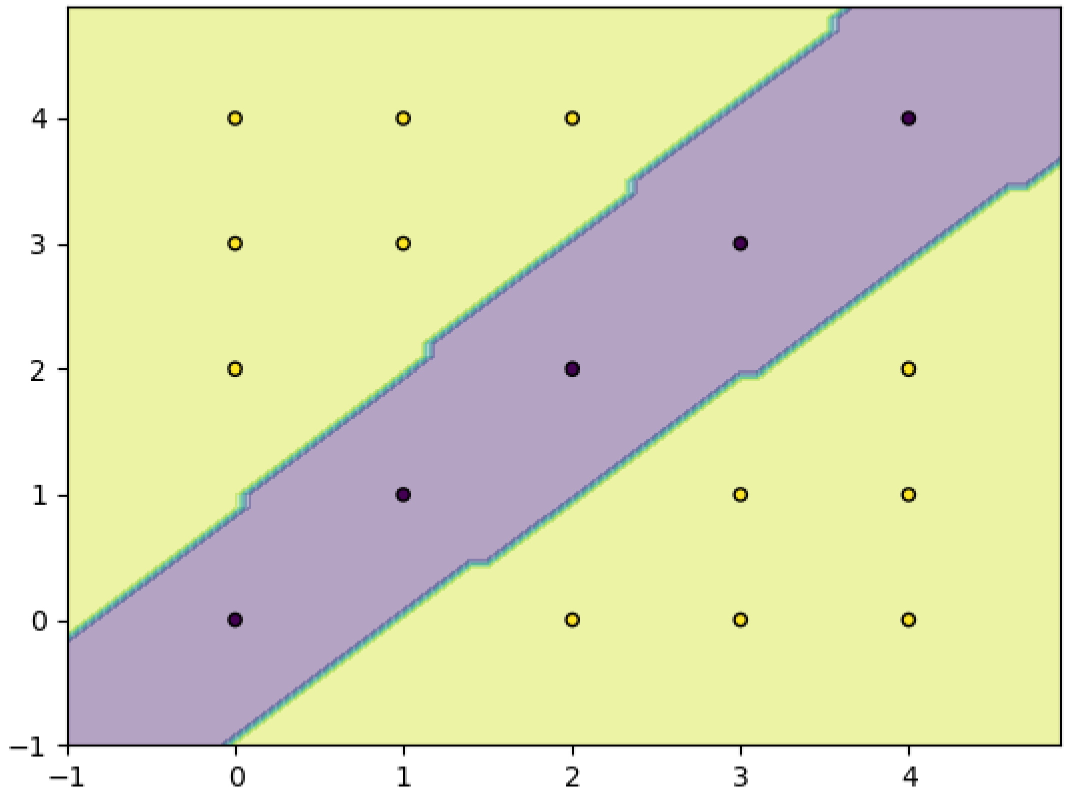
weight w2B: finite-diff: -0.0000 dOutdX(w): -0.0000 TRUE

weight wC: finite-diff: -0.0002 dOutdX(w): -0.0002 TRUE

weight wAC: finite-diff: 0.0002 dOutdX(w): 0.0002 TRUE

weight wBC: finite-diff: 0.0002 dOutdX(w): 0.0002 TRUE

9 weights matched with finite diff, and 0 diverged.
Trained weights:
Weight 'wA': 3.632720
Weight 'w1A': 3.673811
Weight 'w2A': -3.917853
Weight 'wB': -3.294863
Weight 'w1B': 3.967260
Weight 'w2B': -3.655719
Weight 'wC': -4.401011
Weight 'wAC': 9.031638
Weight 'wBC': -8.787502
Testing on inverse-diagonal-band test-data
Accuracy: 1.000000



Regularized Performance Element

To reduce the overfitting problem, we use L2 regularization which is:

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

In [ ]:

```
class RegularizedPerformanceElem (DifferentiableElement):

    def __init__(self, input, weights, lambda_value, desired_value):
        assert isinstance(input,(Input,Neuron))
        DifferentiableElement.__init__(self)
        self.my_input = input
        self.my_desired_val = desired_value
        self.my_lambda_value = lambda_value
        self.my_weights = []
        for i in range(len(weights)):
            self.my_weights.append(weights[i].get_value())
        self.my_weights = np.array(self.my_weights)

    def output(self):
        return -0.5 * (self.my_desired_val - self.get_input().output()) ** 2 - self.my_lambda_value*np.sum(np
.power(self.my_weights, 2))

    def dOutdX(self, elem):
        return (self.my_desired_val - self.get_input().output()) * self.get_input().dOutdX(elem) - 2*self.my_
lambda_value*elem.get_value()

    def set_desired(self,new_desired):
        self.my_desired_val = new_desired

    def get_input(self):
        return self.my_input
```

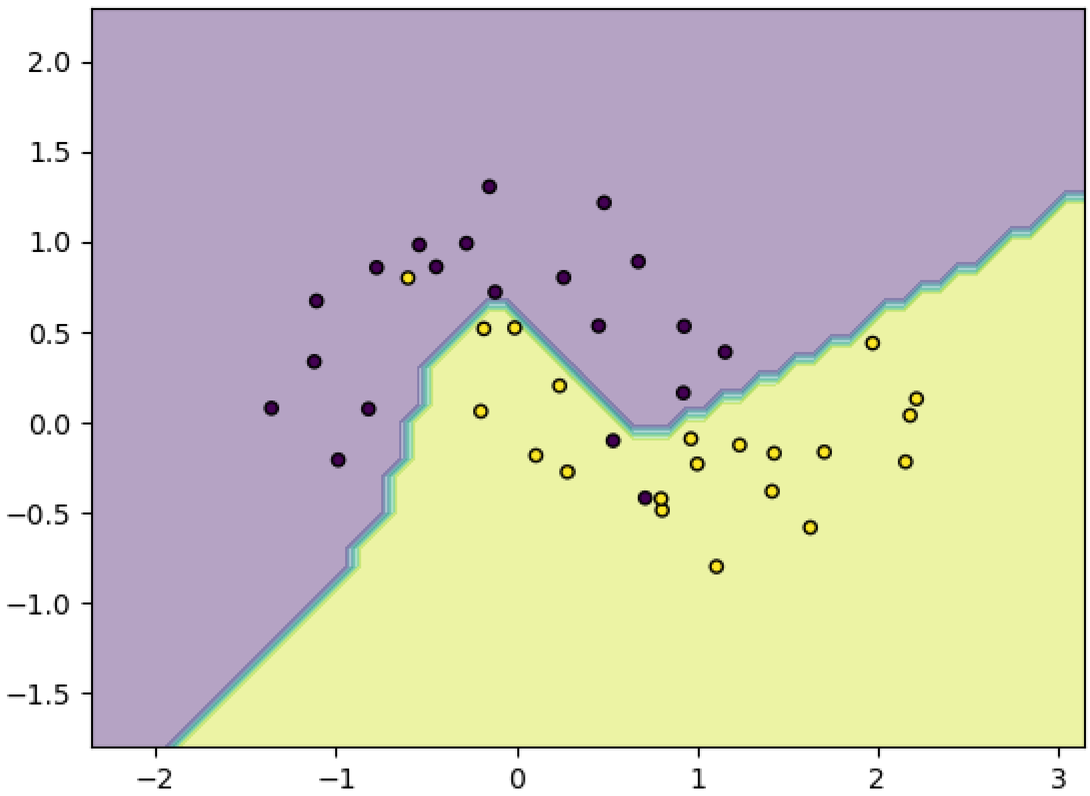
Two-Moons

```
In [3]: def make_neural_net_two_moons():
    i0 = Input('i0', -1.0)
    i1 = Input('i1', 0.0)
    i2 = Input('i2', 0.0)

    seed_random()
    hidden_layer_weights = []
    for i in range(40):
        this_weights = []
        this_weights.append(Weight("wA" + str(int(i/10)) + str(int(i%10)), random_weight()))
        this_weights.append(Weight("w1A" + str(int(i/10)) + str(int(i%10)), random_weight()))
        this_weights.append(Weight("w2A" + str(int(i/10)) + str(int(i%10)), random_weight()))
        hidden_layer_weights.append(this_weights)

    Aneurons = []
    for i in range(40):
        Aneurons.append(Neuron("A" + str(int(i/10)) + str(int(i%10)), [i0, i1, i2], hidden_layer_weights[i]))

    Bweights = []
    wB = Weight('wB', random_weight())
    Bweights.append(wB)
    for i in range(40):
        Bweights.append(Weight("wA" + str(int(i/10)) + str(int(i%10)) + "B", random_weight()))

    B_inputs = [i0]
    B_inputs.extend(Aneurons)
    B = Neuron('B', B_inputs, Bweights)

    P = PerformanceElem(B, 0.0)

    all_weights = []
    for i in range(len(hidden_layer_weights)):
        for j in range(3):
            all_weights.append(hidden_layer_weights[i][j])
    all_weights.extend(Bweights)
    RP = RegularizedPerformanceElem(B, all_weights, 0.0001, 0.0)

    all_neurons = []
    all_neurons.extend(Aneurons)
    all_neurons.append(B)
```

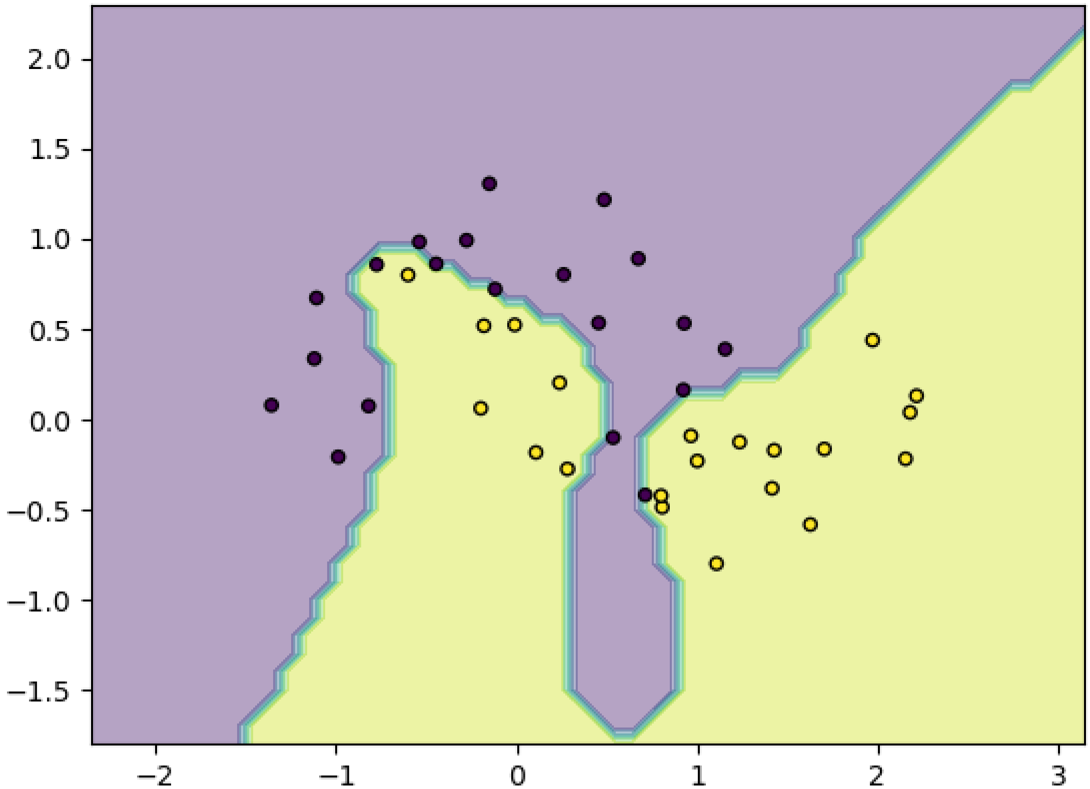
```python
    net = Network(RP, all_neurons)  # RP is the regularized one
    return net
```

Without regularization:

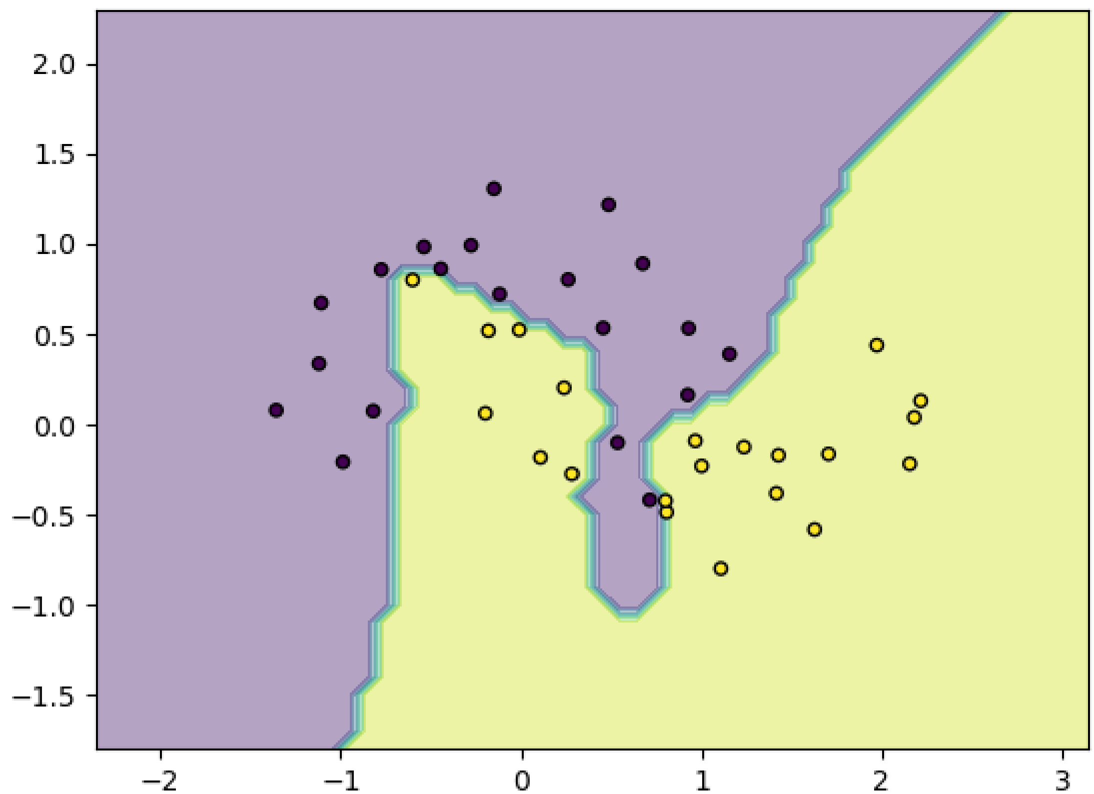
iteration: 100
Train Accuracy: 0.926829
Test Accuracy: 1.000000



iteraion: 500
Train Accuracy: 0.951220
Test Accuracy: 0.820000

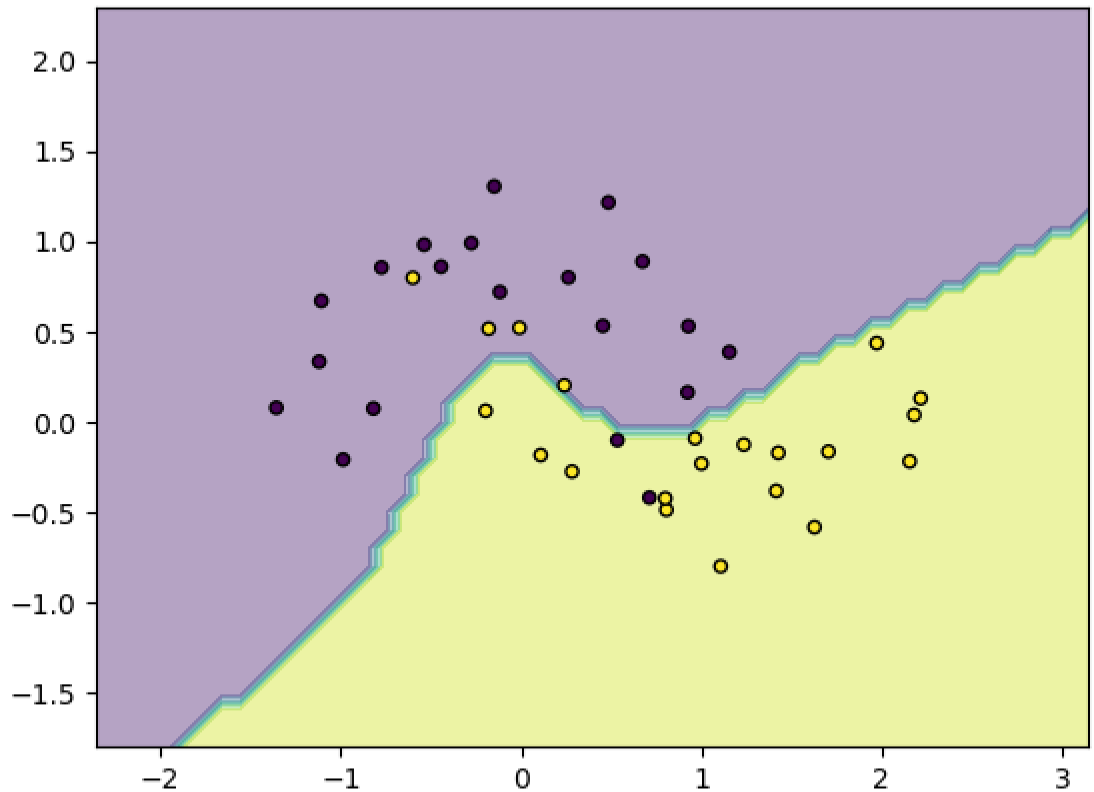


iteration: 1000
Train Accuracy: 1.000000
Test Accuracy: 0.870000



With regularization

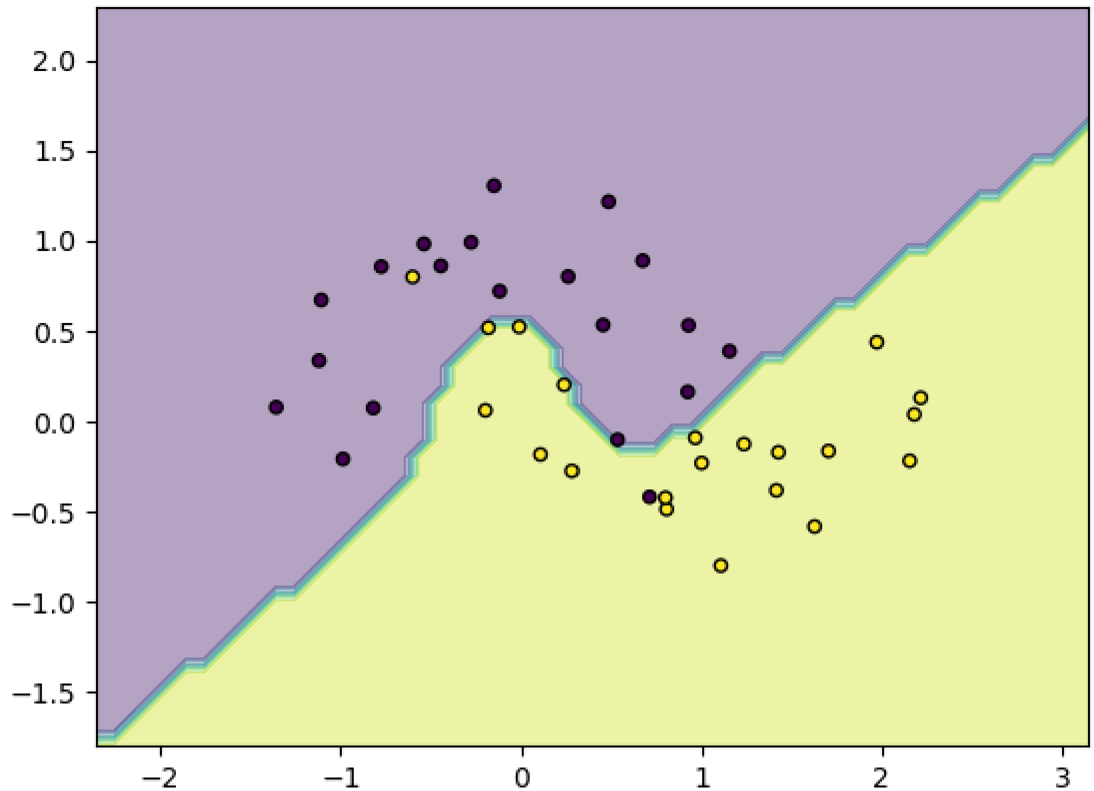
iteration: 100
Train Accuracy: 0.926829
Test Accuracy: 0.980000



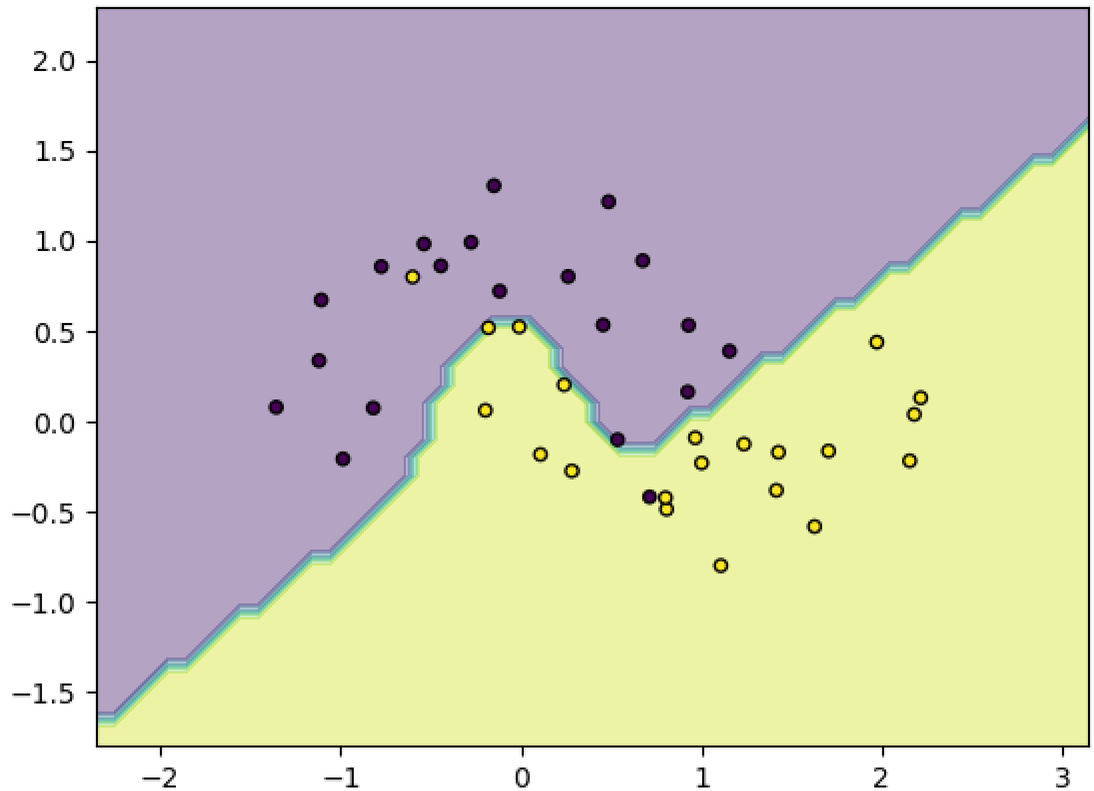
iteraion: 500

Train Accuracy: 0.878049

Test Accuracy: 0.970000



iteraion: 1000
Train Accuracy: 0.878049
Test Accuracy: 0.970000



Plotting function:

In [ ]:
```
def plot_decision_boundary(network, data, xmin=-5, xmax=5, ymin=-5, ymax=5):

    X = np.array([[item[0], item[1]] for item in data])
    y = np.array([item[2] for item in data])
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                         np.arange(y_min, y_max, 0.1))
    tmp = []
    Z = np.c_[xx.ravel(), yy.ravel()]
    for i in range(len(Z)):
        tmp.append((Z[i, 0], Z[i, 1]))
    classified = []
    for i in range(len(tmp)):
        network.inputs[0].set_value(tmp[i][0])
        network.inputs[1].set_value(tmp[i][1])
        network.clear_cache()
        classified.append(1 if network.output.output() > 0.5 else 0)
        network.clear_cache()

    classified = np.array(classified)
    classified = classified.reshape(xx.shape)
    plt.contourf(xx, yy, classified, alpha=0.4)
    plt.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolors='k')
    plt.show()
```

Finite Difference:

In [ ]:
```
def finite_difference(network):
    true_cnt = 0
    for i in range(len(network.weights)):
        network.clear_cache()
        fx = network.performance.output()
        network.weights[i].set_value(network.weights[i].get_value() + 1e-8)
        network.clear_cache()
        eps_added = network.performance.output()
        network.weights[i].set_value(network.weights[i].get_value() - 1e-8)
        result = (eps_added - fx) / 1e-8
        print("weight %1.6s: finite-diff: %2.4f dOutdX(w): %2.4f"
              % (network.weights[i].get_name(),
                 result,
                 network.performance.dOutdX(network.weights[i])), end="")
        if abs(network.performance.dOutdX(network.weights[i]) - result) <= 0.001:  # being approximately equa
l
            print("  TRUE")
            true_cnt += 1
        else:
            print("  FALSE")
    network.clear_cache()
    print(str(true_cnt) + " weights matched with finite diff,"
                          " and " + str(len(network.weights) - true_cnt) + " diverged.")
```