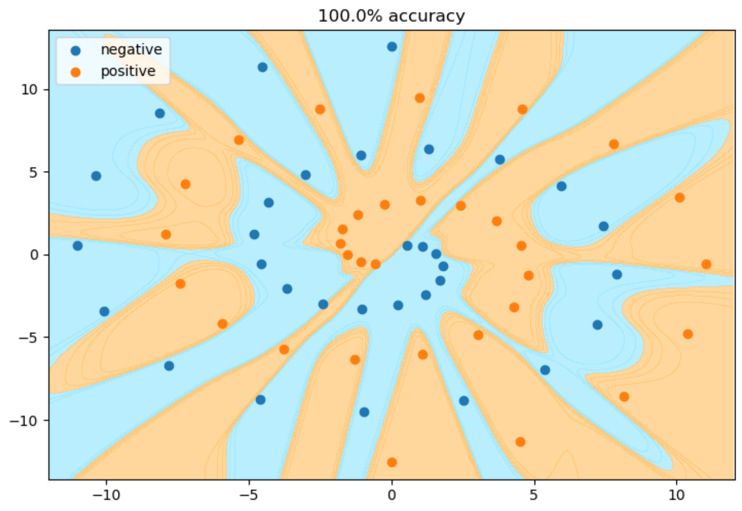
a) No, I did not have enough time to implement my thoughts. What I would have tried is to change the coordinates.

b) Baseline Case:

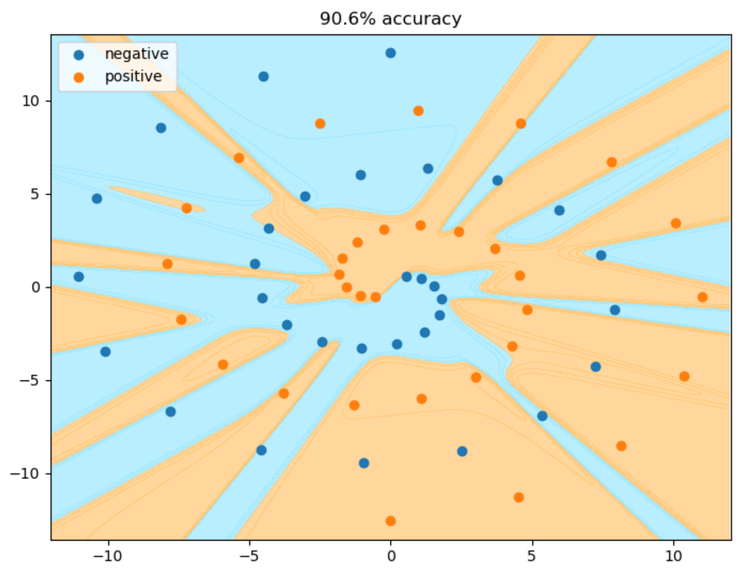
```
        # Try changing the layer structure and hyperparameters
        n_iters = 10000
        learning_rate = 1
        reg = Regularizer(alpha=0.0, penalty="l2")
        layers = [

            # Default
            FullyConnected(2, 32, regularizer=reg),
            SigmoidActivation(),
            FullyConnected(32, 1, regularizer=reg),
            SigmoidActivation()
```



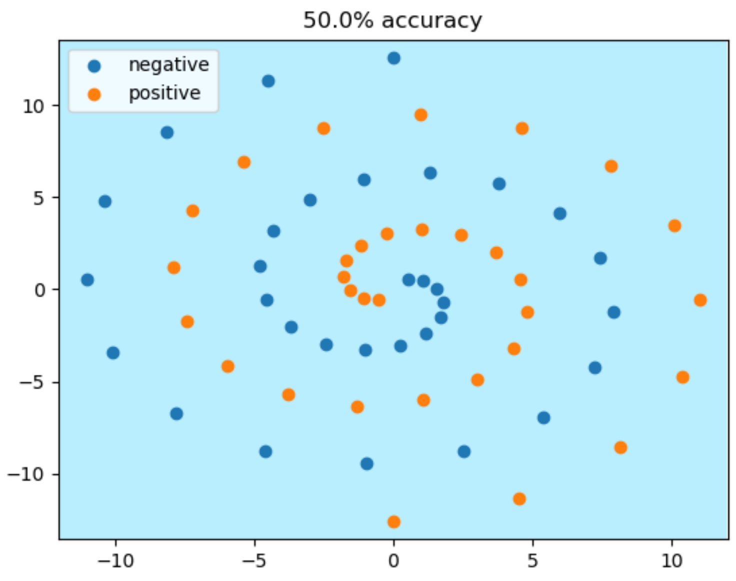
By increasing the number of hidden layers and nodes we can easily overfit the data and get a lower accuracy as seen below.

```
layers = [
    # Change 1
    FullyConnected(2, 32, regularizer=reg),
    SigmoidActivation(),
    FullyConnected(32, 64, regularizer=reg),
    SigmoidActivation(),
    FullyConnected(64, 128, regularizer=reg),
    SigmoidActivation(),
    FullyConnected(128, 32, regularizer=reg),
    SigmoidActivation(),
    FullyConnected(32, 1, regularizer=reg),
    SigmoidActivation()
]
```

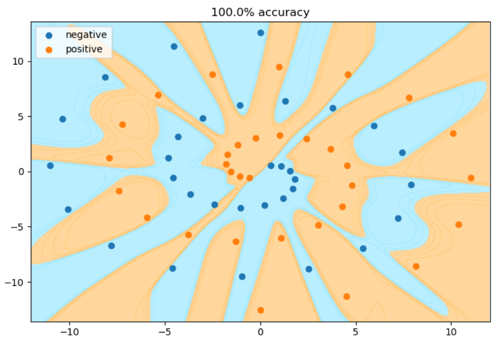


Changed activation function to Relu:
Rely decreases accuracy.

```
    layers = [
        FullyConnected(2, 32, regularizer=reg),
        ReluActivation(),
        FullyConnected(32, 1, regularizer=reg),
        ReluActivation()
    ]
```

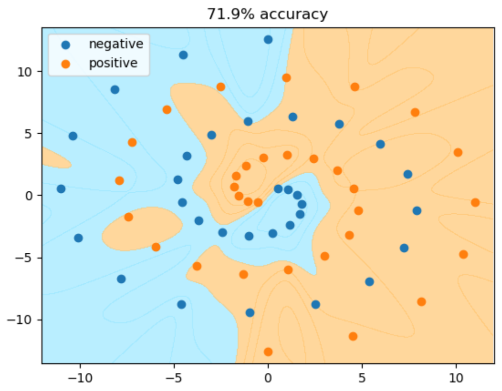


c) **Default: Alpha = 0.0 and Penalty = l2**



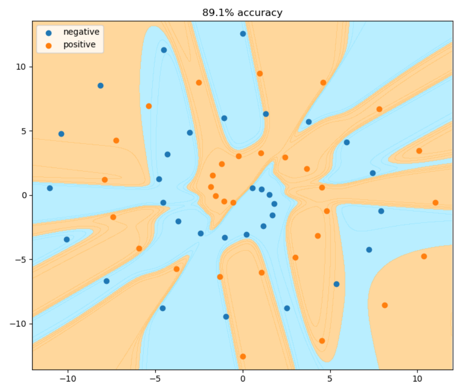
Alpha = 0.001 and Penalty = l2

Increasing alpha from 0 to a small number (0.001) the accuracy drops significantly and with using l2 our accuracy decreases more, because l2 penalizes outliers exponentially.



Alpha = 0.001 and Penalty = l1

Increasing alpha from 0 to a small number (0.001) the accuracy drops significantly, but by using l1 we get a better accuracy as it is not as strong as l2. Because l2's penalty grows exponentially for outliers where l1's penalty increases linearly for outliers and thus gives a better accuracy. L1 is more robust than l2.


89.1% accuracy
negative
positive