# Klion_Coding_HW3

March 29, 2022

[1]:
```
%reload_ext autoreload
%autoreload 2
```

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import torch
import starter
```

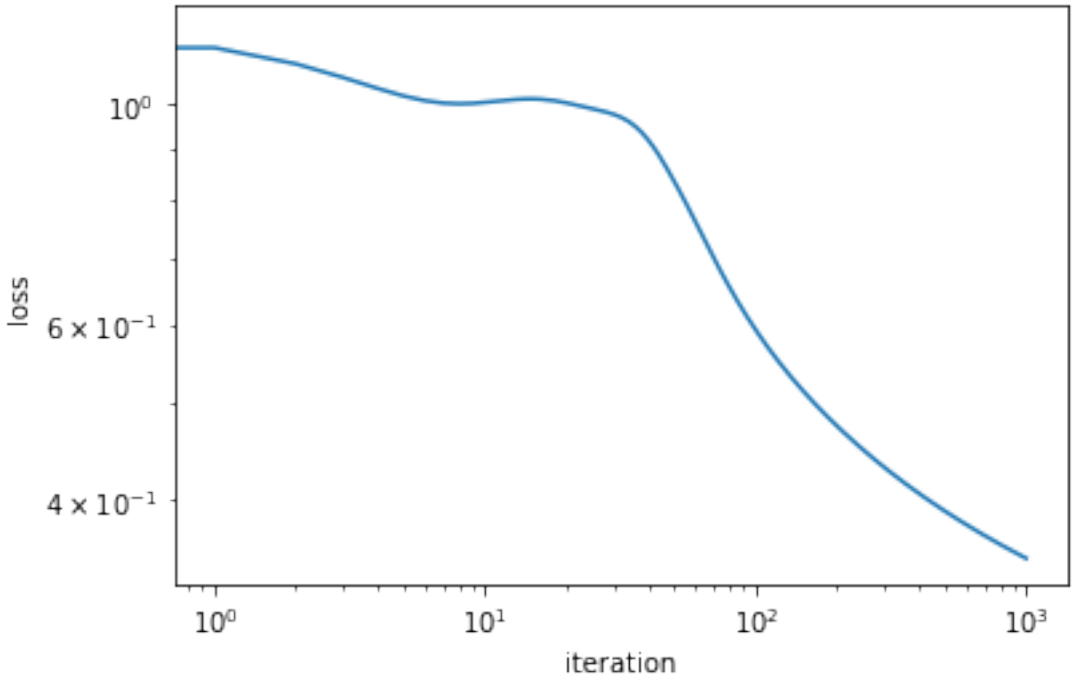
## 1 1. Feed Forward Neural network for Binary XOR

### 1.1 Part A:

[3]:
```
def runAll(input_dim = 2, hidden_dim = 2, add_bias = False):
    model = starter.Model(input_dim, hidden_dim, add_bias = add_bias)
    B = starter.make_binary_arrays(input_dim)
    target = starter.generalize_xor(B)

    B = torch.from_numpy(B).float()
    target = torch.from_numpy(target).float()

    losses = starter.run_optimization(model, B, target, n_iter = 1000)
    return losses, model


def sigmoid(z):
    return 1/(1 + np.exp(-z))


def plot_(hidden, y_prob, points, weights, biases = (0, 0), segment = True):
    plt.scatter(hidden[:, 0], hidden[:, 1], c = y_prob)
    plt.scatter(points[:, 0], points[:, 1], c = ['white', 'black', 'black',␣
 ↪'white'], s = 50)
    if (segment):
        for weight, bias in zip(weights, biases):
            line = starter.make_explicit_line_equation(weight, bias)
            plt.plot(hidden[:, 0], line(hidden[:, 0]), color = 'black')
    return plt.get_current_fig_manager()
```

[4]:
```
losses, model = runAll()
W1_np = model.W1.detach().numpy()
W2_np = model.W2.detach().numpy()
bias1_np = model.bias1.detach().numpy()
bias2_np = model.bias2.detach().numpy()
d_tensor = torch.Tensor([[0,0],
                         [0,1],
                         [1,0],
                         [1,1]])

plt.loglog(range(len(losses)), losses)
plt.ylabel("loss")
plt.xlabel("iteration")
plt.show()
```

```
100%|
 | 1000/1000 [00:00<00:00, 2923.32it/s]
```




[5]:
```
losses[-1]
```

[5]:
```
0.34871822595596313
```

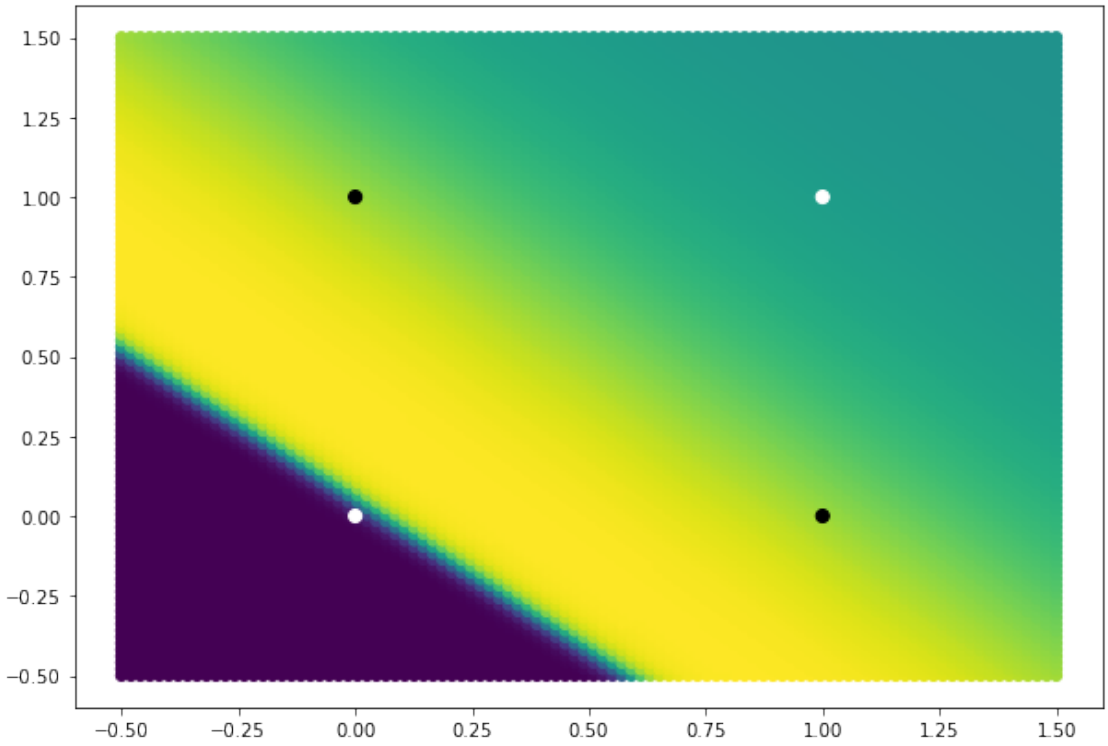
Considering the loss does not go near 0, and gets stuck as iterations reach 1000, I would say the currently implemented trained model fails to implement this problem successfully.

## 2 Part B:

[6]:
```
def makeGrid(x_int : tuple, y_int : tuple, num_points : int):
    x = np.linspace(x_int[0], x_int[1], num = num_points)
    y = np.linspace(y_int[0], y_int[1], num = num_points)
    xv, yv = np.meshgrid(x, y)
    merged = np.array((xv, yv))
    flattened_tensor = torch.tensor(merged, dtype = torch.float).
 ↪flatten(start_dim = 1).transpose(0, 1)
    return flattened_tensor
```

[7]:
```
x_interval, y_interval = (-0.5, 1.5), (-0.5, 1.5)
grid_B = makeGrid(x_interval, y_interval, 100)
model_y1, model_y2 = model(grid_B)
model_y1 = model_y1.detach().numpy()
model_y2 = model_y2.detach().numpy()
```

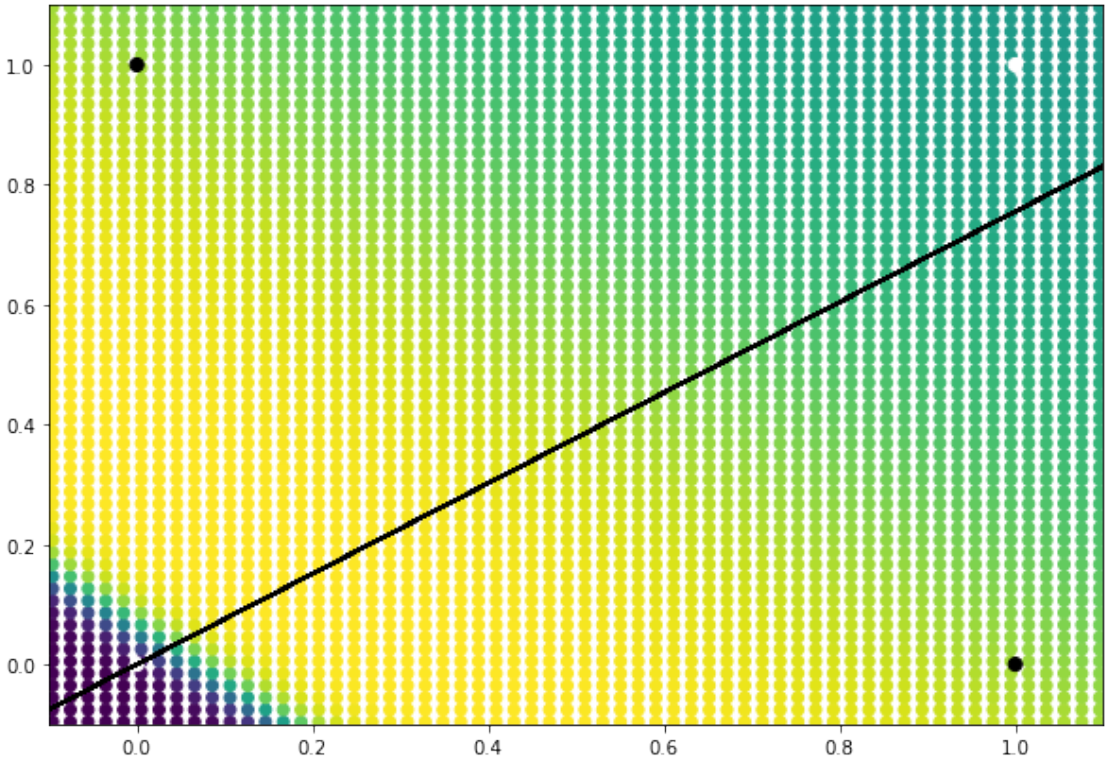
[8]:
```
plt.rcParams['figure.figsize'] = (10,7)
plot_(grid_B, model_y2, d_tensor, W1_np, biases = bias1_np, segment = False)
plt.show()
```




y = -x is one of the lines

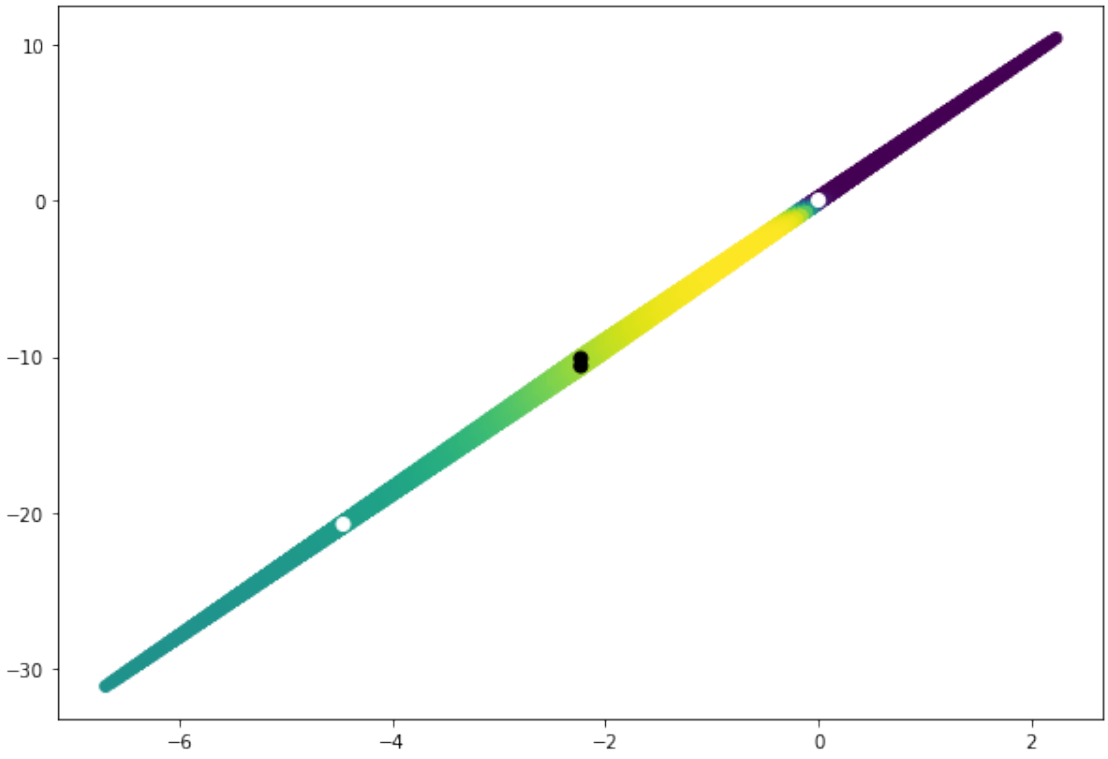
### 2.1 Part C:

[9]:
```
#plt.scatter(grid_B[:,0], grid_B[:, 1], c = model_y2)
#plt.scatter(0, 0, c = 0)
#plt.scatter(0, 1, c = 1)
#plt.scatter(1, 0, c = 1)
#plt.scatter(1, 1, c = 0)
#plt.plot(grid_B[:, 0], line(grid_B[:, 0]), color = 'black')
plot_(grid_B, model_y2, d_tensor, W2_np, biases = bias2_np, segment = True)
plt.xlim(-.1, 1.1)
plt.ylim(-.1, 1.1)
plt.show()
```

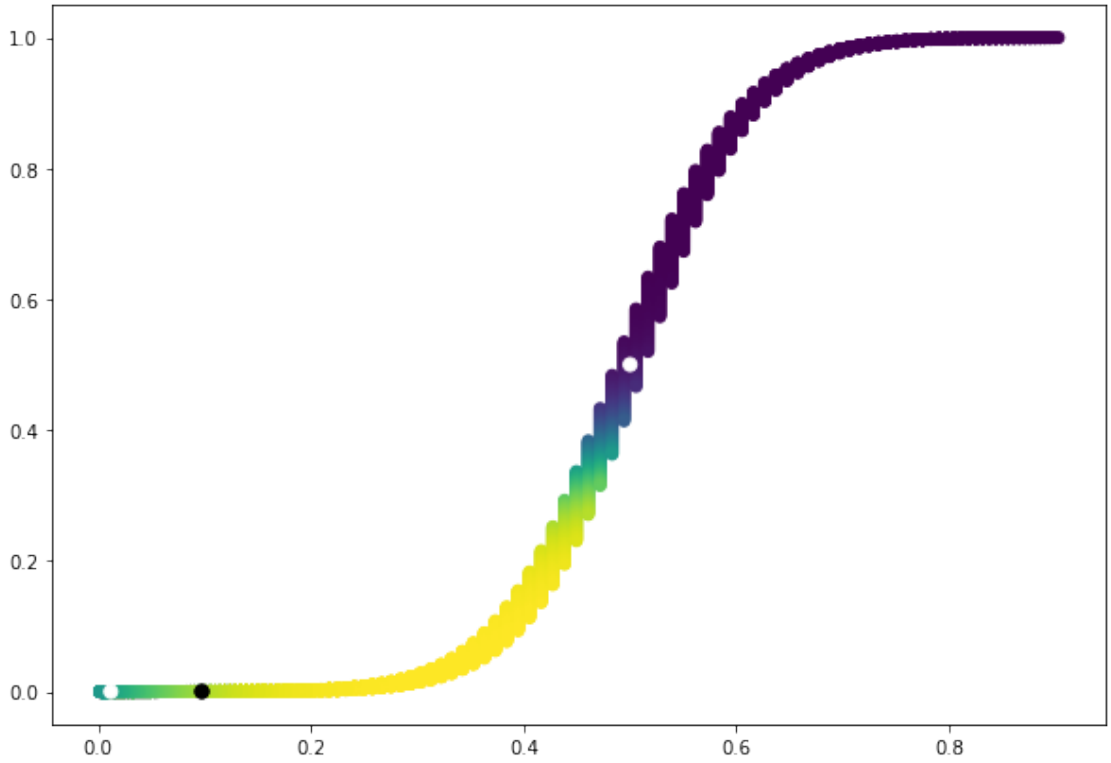



## 3 Part D:

[10]:
```
y1_points, y2_points = model(d_tensor)
y1_points = y1_points.detach().numpy()
plot_(model_y1, model_y2, y1_points, W1_np, biases = bias1_np, segment = False)
plt.show()
```



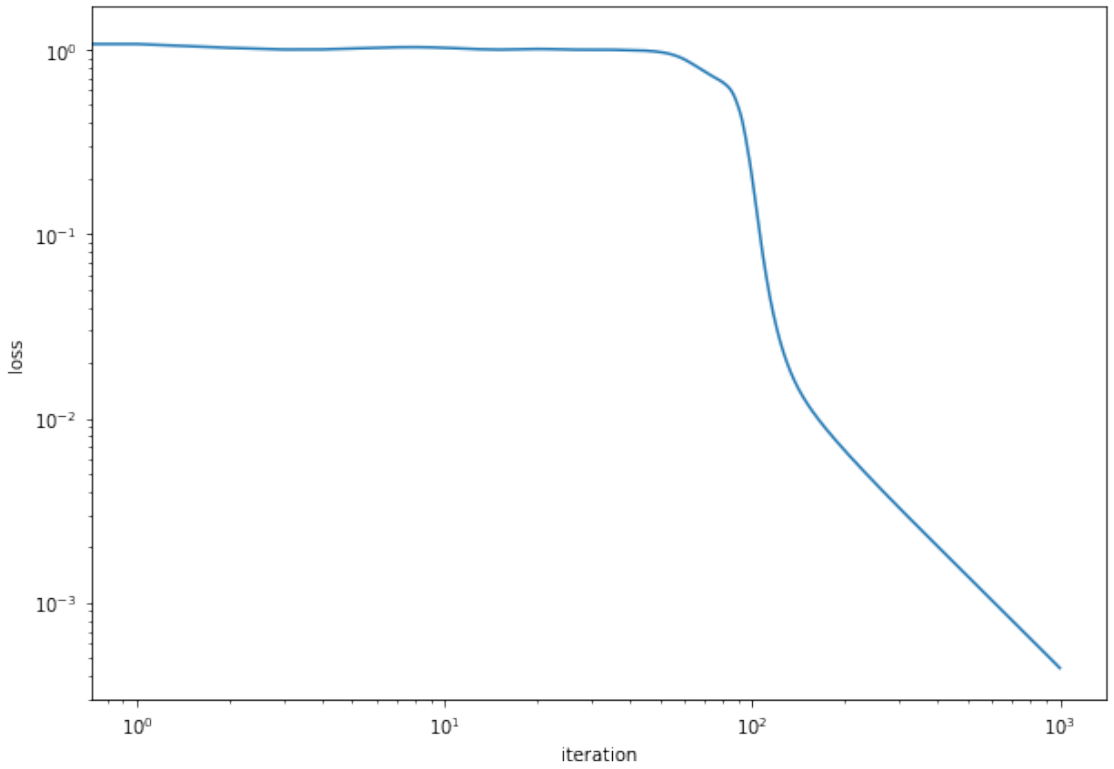

[30]:
```
plot_(sigmoid(model_y1), sigmoid(model_y2), sigmoid(y1_points), W1_np, biases =␣
↪bias1_np, segment = False)
plt.show()
```




## 4 Part F:

[19]:
```
losses_biased, model_biased = runAll(add_bias=True)
W1_biased_np = model_biased.W1.detach().numpy()
W2_biased_np = model_biased.W2.detach().numpy()
bias1_biased_np = model_biased.bias1.detach().numpy()
bias2_biased_np = model_biased.bias2.detach().numpy()

plt.loglog(range(len(losses_biased)), losses_biased)
plt.ylabel("loss")
plt.xlabel("iteration")
plt.show()
```

```
100%|
 | 1000/1000 [00:00<00:00, 2600.95it/s]
```



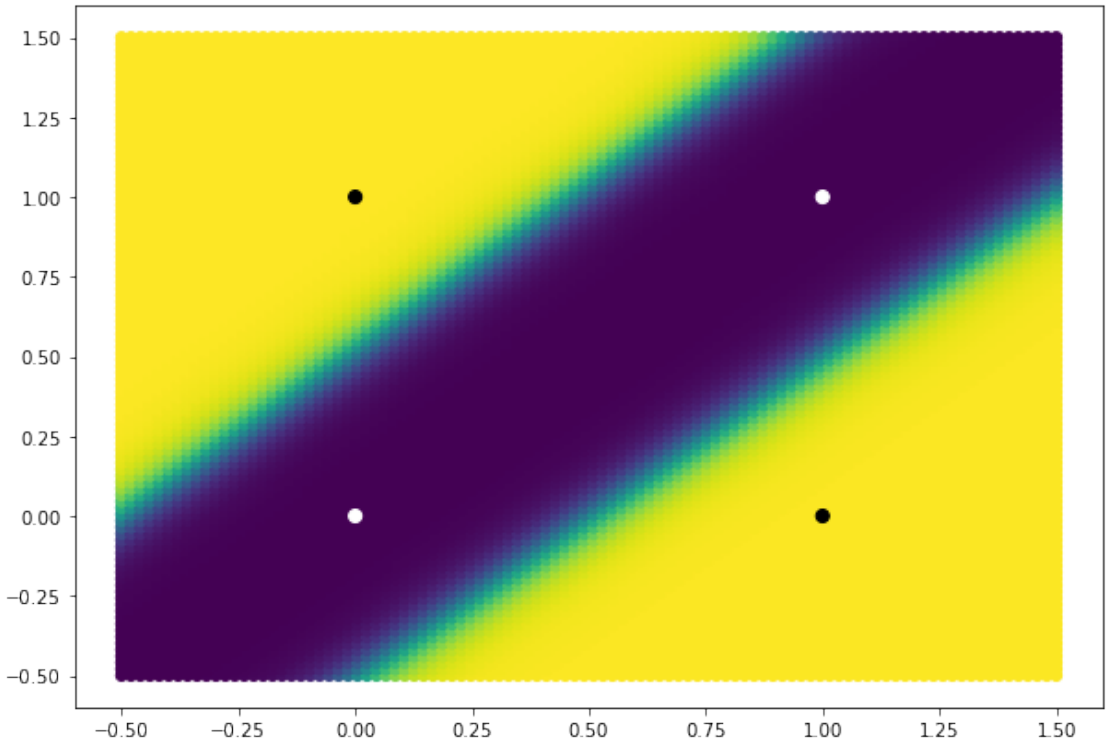

[20]:
```
losses_biased[-1]
```

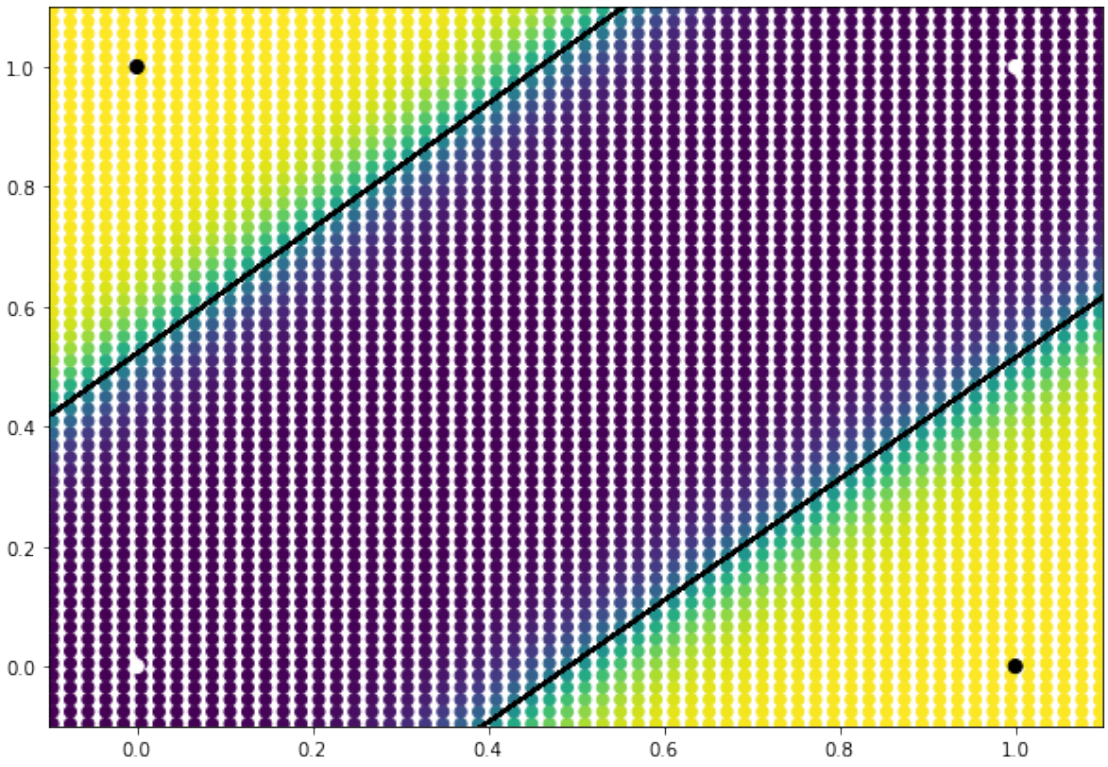
[20]:
```
0.00044232996879145503
```

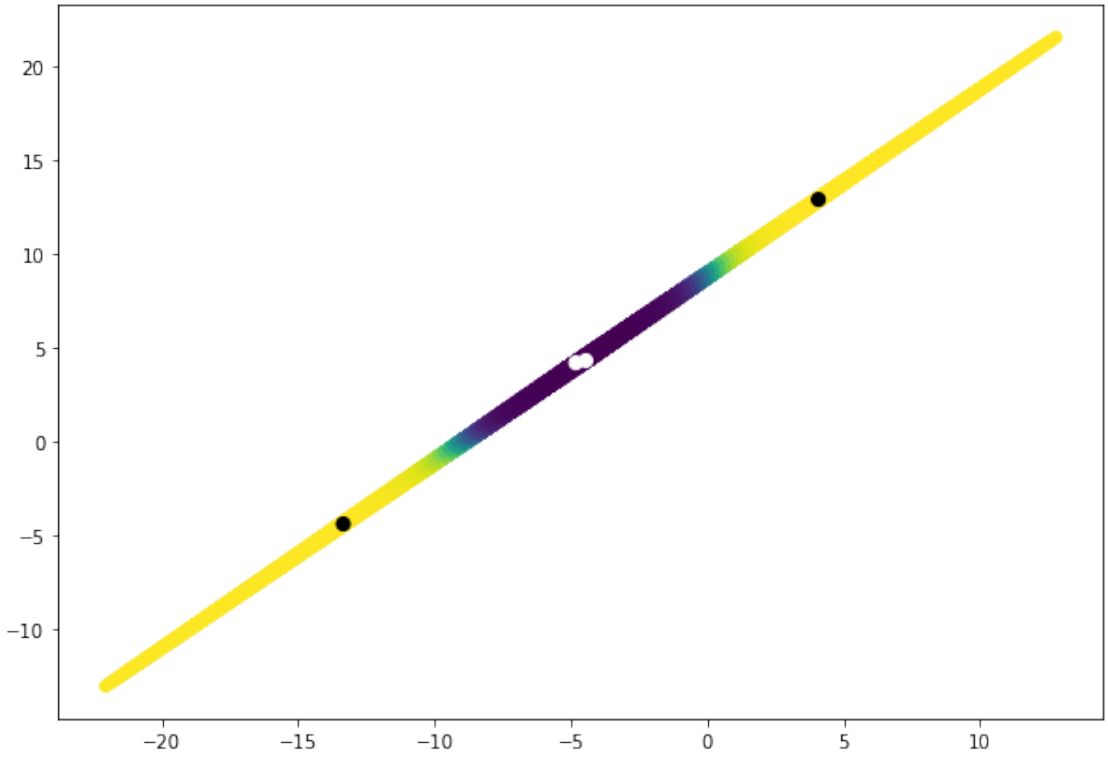
[23]:
```
y1_biased, y2_biased = model_biased(grid_B)
y1_biased = y1_biased.detach().numpy()
y2_biased = y2_biased.detach().numpy()

y1_biased_points, y2_biased_points = model_biased(d_tensor)
y1_biased_points = y1_biased_points.detach().numpy()
```

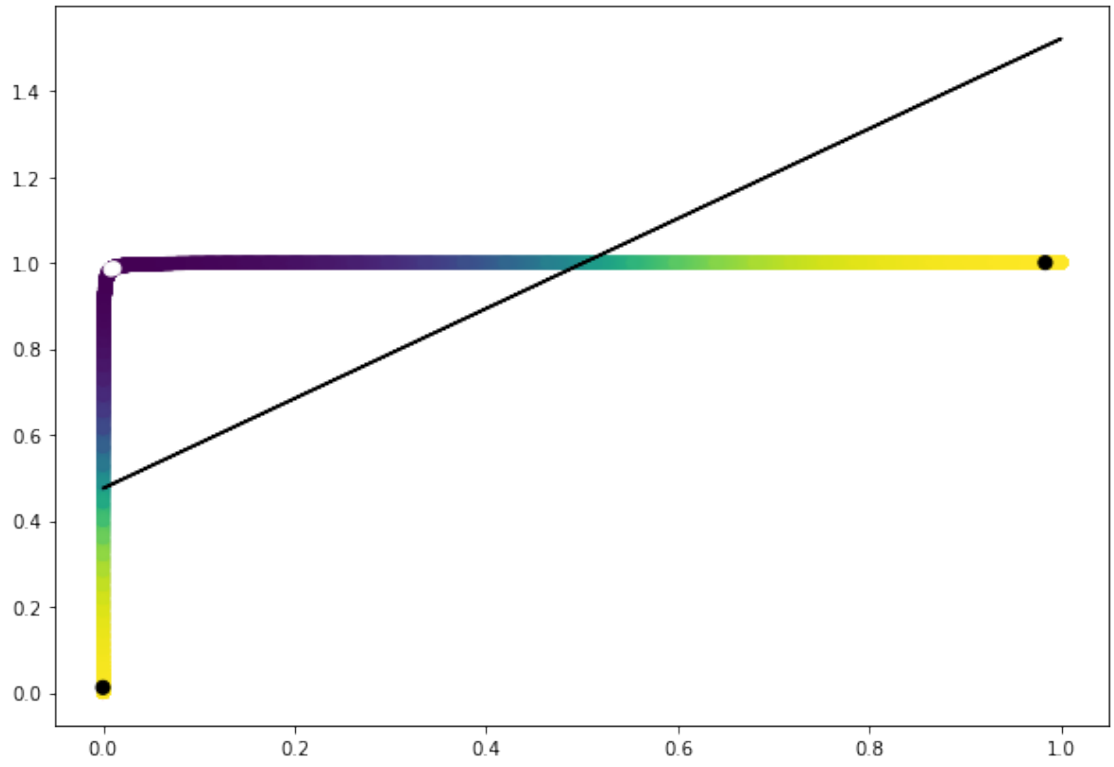
[32]:
```
plot_(grid_B, y2_biased, d_tensor, W1_biased_np, biases = bias1_biased_np,␣
  ↪segment = False)
plt.show()

plot_(grid_B, y2_biased, d_tensor, W1_biased_np, biases = bias1_biased_np)
plt.xlim(-.1, 1.1)
plt.ylim(-.1, 1.1)
plt.show()

plot_(y1_biased, y2_biased, y1_biased_points, W2_biased_np, biases =␣
  ↪bias2_biased_np, segment = False)
```

```
plt.show()

plot_(sigmoid(y1_biased), sigmoid(y2_biased), sigmoid(y1_biased_points),␣
  ↪W2_biased_np, biases = bias2_biased_np)
plt.show()
```




1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


20
15
10
5
0
−5
−10
−20
−15
−10
−5
0
5
10




## 5 Problem 2

## 6 Part A:

[16]:
```
B3 = starter.make_binary_arrays(3)
B3_xor = starter.generalize_xor(B3)
```

[39]:
```
print(B3)
print(B3_xor)
```

```
tensor([[0., 0., 0.],
        [0., 0., 1.],
        [0., 1., 0.],
        [0., 1., 1.],
        [1., 0., 0.],
        [1., 0., 1.],
        [1., 1., 0.],
        [1., 1., 1.]])
tensor([0., 1., 1., 0., 1., 0., 0., 1.])
```

[33]:
```
losses_3d, model_3d = runAll(input_dim = 3, add_bias = True)
```

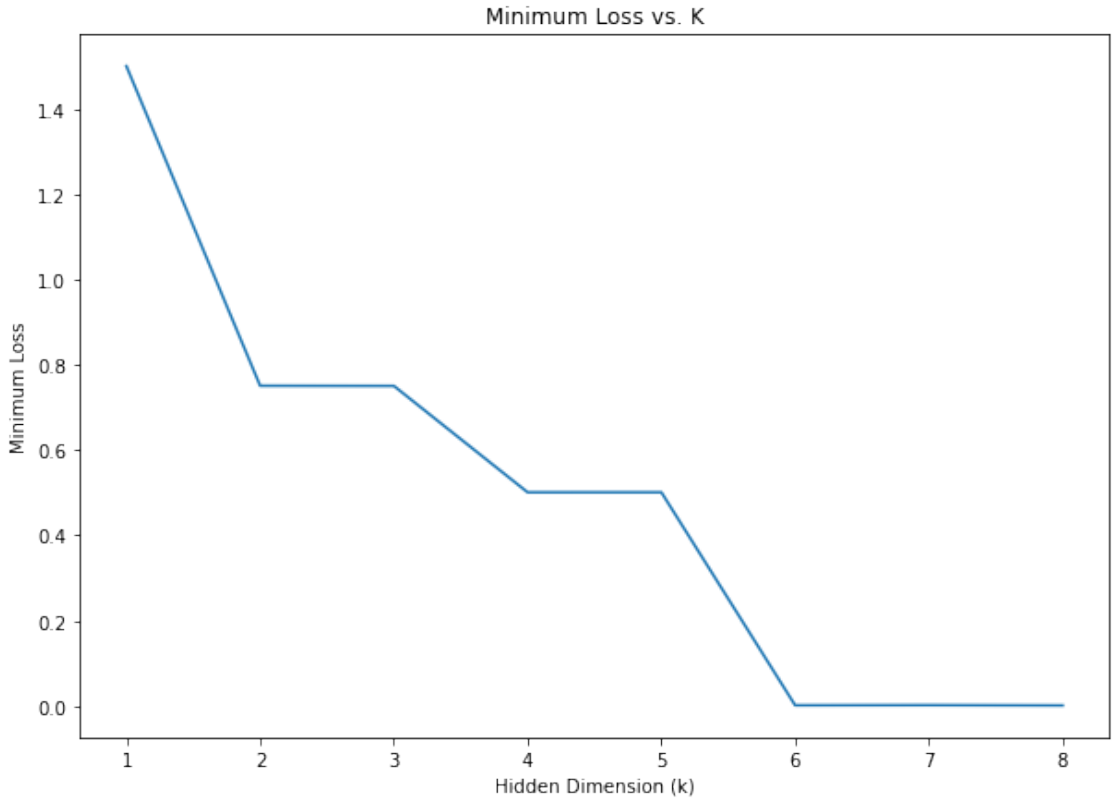
```
100%|
 | 1000/1000 [00:00<00:00, 3186.82it/s]
```

## 7 Part C:

[42]:
```
ks = [1, 2, 3, 4, 5, 6, 7, 8]
losses_k = []
for k in ks:
    local_loss, local_model = runAll(input_dim=3, hidden_dim = k, add_bias =␣
 ↪True)
    losses_k.append(local_loss[-1])
```

```
100%|
 | 1000/1000 [00:00<00:00, 2750.69it/s]
100%|
 | 1000/1000 [00:00<00:00, 2610.38it/s]
100%|
 | 1000/1000 [00:00<00:00, 2596.81it/s]
100%|
 | 1000/1000 [00:00<00:00, 2610.38it/s]
100%|
 | 1000/1000 [00:00<00:00, 2593.38it/s]
100%|
 | 1000/1000 [00:00<00:00, 2586.98it/s]
100%|
 | 1000/1000 [00:00<00:00, 2570.42it/s]
100%|
 | 1000/1000 [00:00<00:00, 2549.29it/s]
```

[45]:
```
plt.plot(ks, losses_k)
plt.xlabel("Hidden Dimension (k)")
plt.ylabel("Minimum Loss")
plt.title("Minimum Loss vs. K")
plt.show()
```




It appears that the model starts to correctly classify the model at k = 6

## 8 Part E:

[51]:
```
dims = [4, 5, 6]
dims_losses = []
for dim in dims:
    local_min_losses = []
    for k in ks:
        local_loss, local_model = runAll(input_dim = dim, hidden_dim = k,␣
 ↪add_bias = True)
        local_min_losses.append(local_loss[-1])
    print(local_min_losses)
    dims_losses.append(local_min_losses)
```

```
100%|
 | 1000/1000 [00:00<00:00, 2796.55it/s]
100%|
 | 1000/1000 [00:00<00:00, 2637.93it/s]
100%|
 | 1000/1000 [00:00<00:00, 2603.58it/s]
```

```
100%|
 | 1000/1000 [00:00<00:00, 2630.98it/s]
100%|
 | 1000/1000 [00:00<00:00, 2610.37it/s]
100%|
 | 1000/1000 [00:00<00:00, 2627.50it/s]
100%|
 | 1000/1000 [00:00<00:00, 2596.82it/s]
100%|
 | 1000/1000 [00:00<00:00, 2603.58it/s]
```

```
[4.0, 2.5461552143096924, 0.8053857684135437, 0.8158525228500366,
0.8051002025604248, 0.01461966522037983, 0.037543900310993195,
0.8037407398223877]
```

```
100%|
 | 1000/1000 [00:00<00:00, 2769.47it/s]
100%|
 | 1000/1000 [00:00<00:00, 2617.21it/s]
100%|
 | 1000/1000 [00:00<00:00, 2179.15it/s]
100%|
 | 1000/1000 [00:00<00:00, 2138.41it/s]
100%|
 | 1000/1000 [00:00<00:00, 2282.59it/s]
100%|
 | 1000/1000 [00:00<00:00, 2101.99it/s]
100%|
 | 1000/1000 [00:00<00:00, 2118.16it/s]
100%|
 | 1000/1000 [00:00<00:00, 2178.16it/s]
```

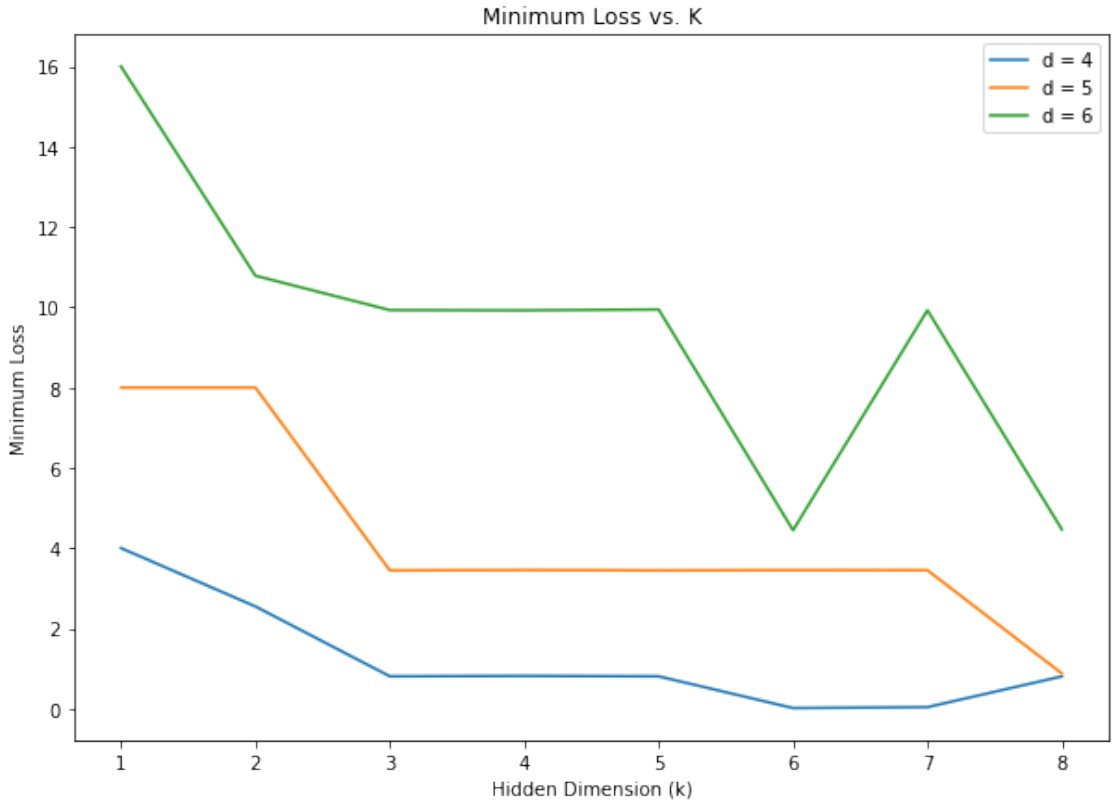
```
[7.999999523162842, 8.0, 3.4433600902557373, 3.4526374340057373,
3.4436800479888916, 3.4497451782226562, 3.448378562927246, 0.8758033514022827]
```

```
100%|
 | 1000/1000 [00:00<00:00, 2590.09it/s]
100%|
 | 1000/1000 [00:00<00:00, 2216.80it/s]
100%|
 | 1000/1000 [00:00<00:00, 2236.63it/s]
100%|
 | 1000/1000 [00:00<00:00, 2251.74it/s]
100%|
 | 1000/1000 [00:00<00:00, 2093.67it/s]
100%|
 | 1000/1000 [00:00<00:00, 2118.17it/s]
100%|
 | 1000/1000 [00:00<00:00, 2178.16it/s]
100%|
```

```
| 1000/1000 [00:00<00:00, 2048.72it/s]
```

```
[16.0, 10.785126686096191, 9.929214477539062, 9.922440528869629,
9.945112228393555, 4.444655895233154, 9.924810409545898, 4.465934753417969]
```

[58]:
```
plt.plot(ks, dims_losses[0], label = 'd = 4')
plt.plot(ks, dims_losses[1], label = 'd = 5')
plt.plot(ks, dims_losses[2], label = 'd = 6')
plt.xlabel("Hidden Dimension (k)")
plt.ylabel("Minimum Loss")
plt.title("Minimum Loss vs. K")
plt.legend()
plt.show()
```




6 hyperplanes appear to succesfully classify the d-dimensional problem