M7-L1 Problem 2

In this problem, you will explore what happens when you change the weights/biases of a neural network.

Neural networks act as functions that attempt to map from input data to output data. In training a neural network, the goal is to find the values of weights and biases that minimize the loss between their output and the desired output. This is typically done with a technique called backpropagation; however, here you will simply note the effect of changing specific weights in the network which has been pre-trained.

First, load the data and initial weights/biases below:

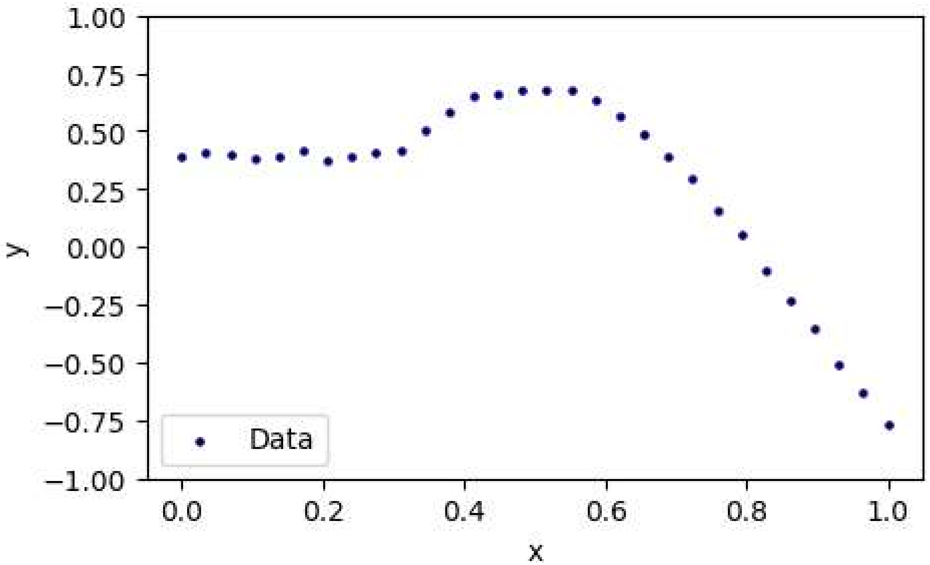
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([0.        , 0.03448276, 0.06896552, 0.10344828, 0.13793103,0.17241379
y = np.array([ 0.38914369,  0.40997345,  0.40282978,  0.38493705,  0.394214  ,0.416

weights = [np.array([[-5.90378086,  0,  0 ]]).T,
           np.array([[ 0.8996511 ,  4.75805319, -0.95266992],[-0.99667812, -0.89303
           np.array([[ 1.71988943, -1.56198034, -3.31173131]])]

biases = [np.array([ 2.02112296, -3.47589349, -1.11586831]), np.array([ 1.35350721,

plt.figure(figsize=(5,3))
plt.scatter(x,y,s=5,c="navy",label="Data")
plt.legend(loc="lower left")
plt.ylim(-1,1)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



MLP Function

Copy in your MLP function (and all necessary helper functions) below. Make sure it is called `MLP()`. In this case, you can plug in `x`, `weights`, and `biases` to try and predict `y`. Make sure you use the sigmoid activation function after each layer (except the final layer).

In [12]:

```
# YOUR CODE GOES HERE
def sigmoid(x):
    return 1./(1.+np.exp(-x))

def MLP(x, weights, biases):
    # YOUR CODE GOES HERE
    for i in range(len(weights)):
        x = np.dot(x,weights[i].T) + biases[i]
        if i != len(weights) - 1:
            x = sigmoid(x)
    return x


y_pred = MLP(x,weights,biases)
print(y_pred)

def MSE(y, pred):
    return np.mean((y.flatten()-pred.flatten())**2)

print(MSE(y, y_pred))
```

```
[[ 0.41184525]
 [ 0.40363107]
 [ 0.39576217]
 [ 0.38896171]
 [ 0.38432719]
 [ 0.38342326]
 [ 0.38830801]
 [ 0.40138387]
 [ 0.42492654]
 [ 0.4601946 ]
 [ 0.50626198]
 [ 0.55912128]
 [ 0.61182898]
 [ 0.65601597]
 [ 0.68409945]
 [ 0.69097651]
 [ 0.67447749]
 [ 0.63479   ]
 [ 0.57352063]
 [ 0.49291273]
 [ 0.39539858]
 [ 0.28343883]
 [ 0.1595274 ]
 [ 0.02625145]
 [-0.11366923]
 [-0.25740182]
 [-0.40208635]
 [-0.54497098]
 [-0.68356411]
 [-0.8157693 ]]
0.0
```

Varying weights

The provided network has 2 hidden layers, each with 3 neurons. The weights and biases are shown below. Note the weights w_a and w_b -- these are left for you to investigate:

$$\underline{x\ (N\times 1)} \to \sigma\left(w = \begin{bmatrix}-5.9\\ \boldsymbol{w_a}\\ \boldsymbol{w_b}\end{bmatrix}; b = \begin{bmatrix}2.02\\ -3.48\\ -1.12\end{bmatrix}'\right) \to \underline{(N\times 3)} \to \sigma\left(w = \begin{bmatrix}0.9\\ 4.76\\ -0.95\end{bmatrix}\right.$$

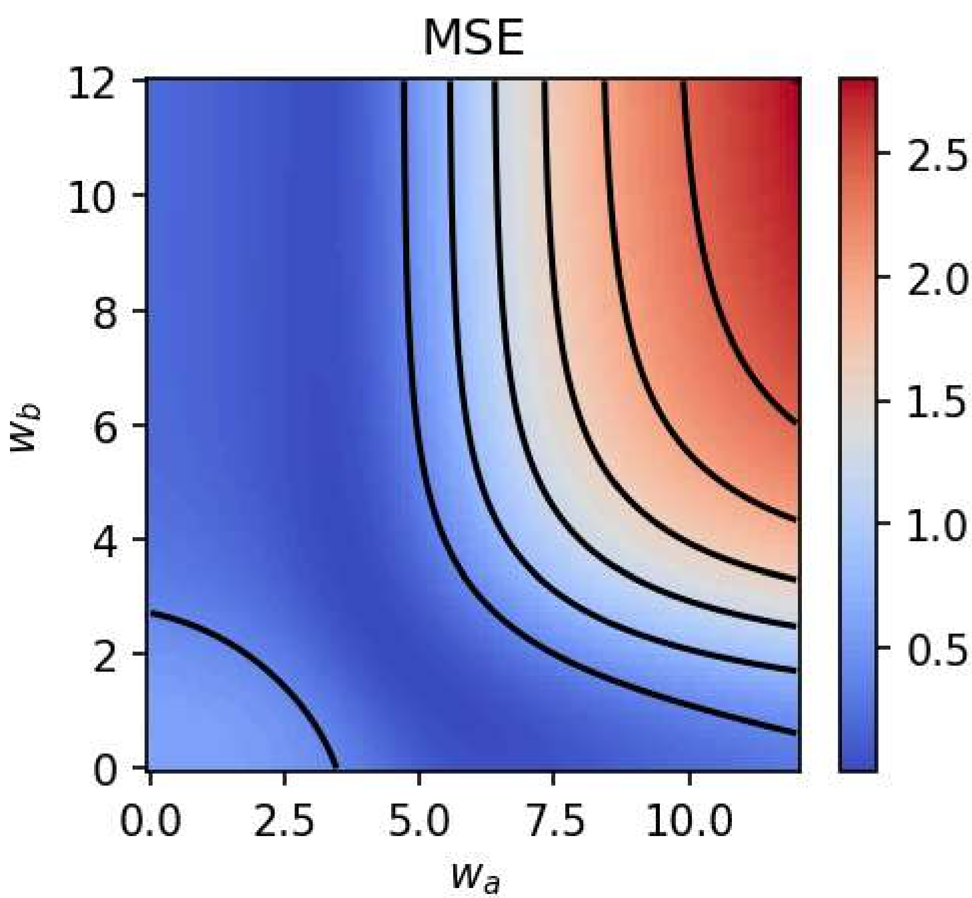
We can compute the MSE for each combination of (w_a, w_b) to see where MSE is minimized.

In [6]:
```python
def MSE(y, pred):
    return np.mean((y.flatten()-pred.flatten())**2)

vals = np.linspace(0,12,100)
was, wbs = np.meshgrid(vals,vals)
mses = np.zeros_like(was.flatten())
```

```
for i in range(len(was.flatten())):
    ws, bs = weights.copy(), biases.copy()
    ws[0][1,0] = was.flatten()[i]
    ws[0][2,0] = wbs.flatten()[i]
    mses[i] = MSE(y, MLP(x, ws, bs))
mses = mses.reshape(was.shape)

plt.figure(figsize = (3.5,3),dpi=150)
plt.title("MSE")
plt.contour(was,wbs,mses,colors="black")
plt.pcolormesh(was,wbs,mses,shading="nearest",cmap="coolwarm")
plt.xlabel("$w_a$")
plt.ylabel("$w_b$")
plt.colorbar()
plt.show()
```



In [7]:
```
%matplotlib inline
from ipywidgets import interact, interactive, fixed, interact_manual, Layout, Float


def plot(wa, wb):
    ws, bs = weights.copy(), biases.copy()
    ws[0][1,0] = wa
    ws[0][2,0] = wb

    xs = np.linspace(0,1)
    ys = MLP(xs.reshape(-1,1), ws, bs)
```

```
    plt.figure(figsize=(10,4),dpi=120)

    plt.subplot(1,2,1)
    plt.contour(was,wbs,mses,colors="black")
    plt.pcolormesh(was,wbs,mses,shading="nearest",cmap="coolwarm")
    plt.title(f"$w_a = {wa:.1f}$;  $w_b = {wb:.1f}$")
    plt.xlabel("$w_a$")
    plt.ylabel("$w_b$")
    plt.scatter(wa,wb,marker="*",color="black")
    plt.colorbar()

    plt.subplot(1,2,2)
    plt.scatter(x,y,s=5,c="navy",label="Data")
    plt.plot(xs,ys,"r-",linewidth=1,label="MLP")
    plt.title(f"MSE = {MSE(y, MLP(x, ws, bs)):.3f}")
    plt.legend(loc="lower left")
    plt.ylim(-1,1)
    plt.xlabel("x")
    plt.ylabel("y")

    plt.show()


slider1 = FloatSlider(
    value=0,
    min=0,
    max=12,
    step=.5,
    description='wa',
    disabled=False,
    continuous_update=True,
    orientation='horizontal',
    readout=False,
    layout = Layout(width='550px')
)

slider2 = FloatSlider(
    value=0,
    min=0,
    max=12,
    step=.5,
    description='wb',
    disabled=False,
    continuous_update=True,
    orientation='horizontal',
    readout=False,
    layout = Layout(width='550px')
)



interactive_plot = interactive(
    plot,
    wa = slider1,
```

```
    wb = slider2
    )
output = interactive_plot.children[-1]
output.layout.height = '500px'

interactive_plot
```

Out[7]:
```
interactive(children=(FloatSlider(value=0.0, description='wa', layout=Layout(width
='550px'), max=12.0, readout…
```

Questions

1. For $w_a = 4.0$, what walue of w_b gives the lowest MSE (to the nearest 0.5)?

- *ANSWER:* Wb = 3.0

2. For the large values of w_a and w_b, describe the MLP's predictions.

- *ANSWER:* When w_a and w_b gets super large, the prediction is completely off the ground truth.