DSC 140B - Homework 06

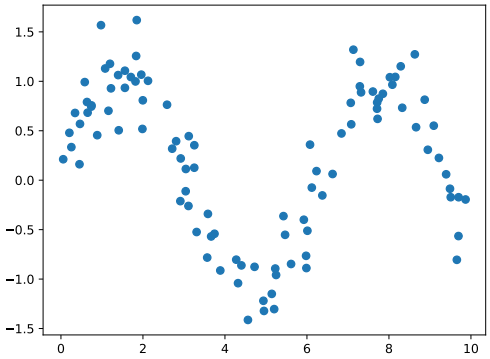
Due: Wednesday, February 18

Instructions:

- Write your solutions to the following problems **by hand**, either on on another piece of paper that you scan or using a tablet. Typed solutions will not be accepted for credit!
  - Code listings are an exception. You do not need to handwrite code, and you can instead include the code as a screenshot.
- Unless otherwise noted by the problem's instructions, show your work or provide some justification for your answer.
- Homework problems are graded pass/fail on completeness and effort, not correctness.
- Homeworks are due via Gradescope at 11:59 PM.

Problem 1. (1 credit)

This problem will use the data shown below:



You can find this data at:

`https://f000.backblazeb2.com/file/jeldridge-data/008-noisy_sin/data.csv`

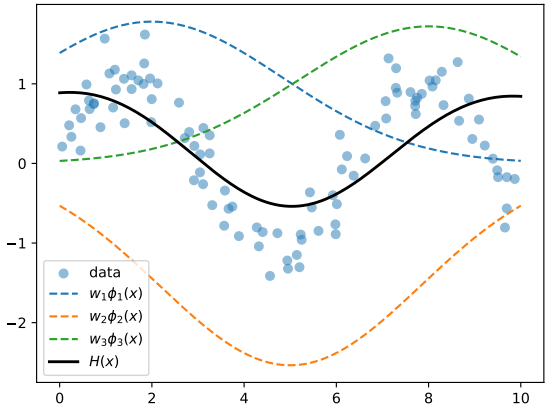
In each part below, you will train a Gaussian RBF network H on this data using three Gaussians centered at 2, 5, and 8. For each part, plot $H(x)$ for x from 0 to 10, on top of a scatter plot of the data. Your plot should also show the individual Gaussian basis functions, each scaled by its learned weight. That is, your plots should include four functions:

- $w_1\varphi_1(x)$, where φ_1 is the Gaussian RBF centered at 2 and w_1 is its learned weight;
- $w_2\varphi_2(x)$, where φ_2 is the Gaussian RBF centered at 5 and w_2 is its learned weight;
- $w_3\varphi_3(x)$, where φ_3 is the Gaussian RBF centered at 8 and w_3 is its learned weight;
- $H(x) = w_1\varphi_1(x) + w_2\varphi_2(x) + w_3\varphi_3(x)$.

Plot H as a solid line, and the rest as dashed lines. You can use different colors for each function if you like, but it's not required.

a) Train the model using a width parameter of $\sigma = 4$. Show your plot.

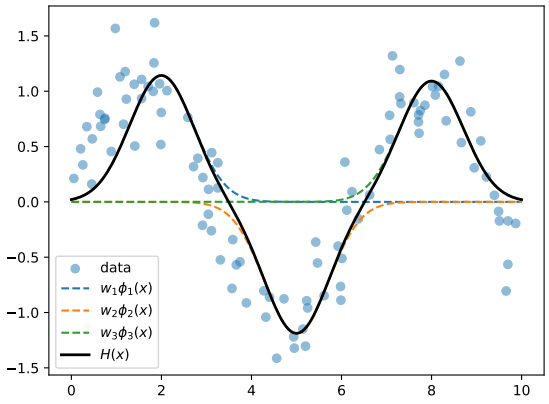
Solution:




```
sigma = 4
centers = [2, 5, 8]

def make_phi(center, sigma):
    def phi(x):
        return np.exp(-(x - center)**2 / sigma**2)
    return phi

phis = [make_phi(mu, sigma) for mu in centers]

Phi = np.column_stack([phi(x) for phi in phis])
w = np.linalg.lstsq(Phi, y, rcond=None)[0]

xx = np.linspace(0, 10, 200)
Z = np.column_stack([phi(xx) for phi in phis])

plt.scatter(x, y, alpha=0.5, label='data')
for i, phi in enumerate(phis):
    plt.plot(xx, w[i] * phi(xx), '--', label=f'$w_{i+1} \phi_{i+1}(x)$')
plt.plot(xx, Z @ w, 'k-', linewidth=2, label='$H(x)$')
plt.legend()
```

b) Train the model using a width parameter of $\sigma = 1$.

Solution:



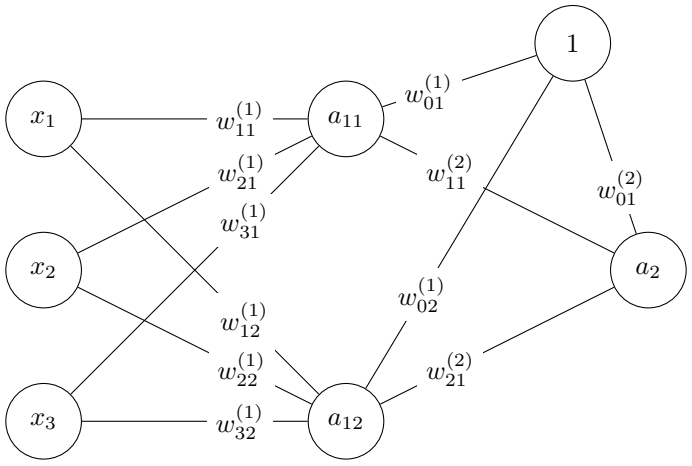

```
sigma = 1
centers = [2, 5, 8]

def make_phi(center, sigma):
    def phi(x):
        return np.exp(-(x - center)**2 / sigma**2)
    return phi

phis = [make_phi(mu, sigma) for mu in centers]

Phi = np.column_stack([phi(x) for phi in phis])
w = np.linalg.lstsq(Phi, y, rcond=None)[0]

xx = np.linspace(0, 10, 200)
Z = np.column_stack([phi(xx) for phi in phis])

plt.scatter(x, y, alpha=0.5, label='data')
for i, phi in enumerate(phis):
    plt.plot(xx, w[i] * phi(xx), '--', label=f'$w_{i+1} \phi_{i+1}(x)$')
plt.plot(xx, Z @ w, 'k-', linewidth=2, label='$H(x)$')
plt.legend()
```

Problem 2. (1 credit)

In lecture it was said that a neural network with linear activation functions is a linear prediction function, meaning that its decision boundary will also be linear. If we wish to have a non-linear decision boundary, we must introduce non-linearities with, for instance, non-linear activation functions.

In this problem, we'll see concretely that a neural network with linear activations is again linear.

Consider the neural network shown below.



The inputs x_1, x_2, and x_3 are numbers. Each $w_{ij}^{(k)}$ is a scalar weight. a_{ij} denotes the output of a neuron. Remember that when linear activations are used, the output of a neuron is simply the weighted sum of its inputs. So for instance:

$$a_{11} = w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 + w_{31}^{(1)} x_3 + w_{01}^{(1)}$$

The node labeled 1 is the bias input. a_2 is the output of the neural network overall.

a) Write the output of the network, a_2, as an expression involving only the inputs x_1, x_2, x_3 and the weights, $w_{ij}^{(k)}$. a_i should not appear in your expression.

Solution: We have

$$\begin{aligned} a_{11} &= w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 + w_{31}^{(1)} x_3 + w_{01}^{(1)} \\ a_{12} &= w_{12}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{32}^{(1)} x_3 + w_{02}^{(1)} \end{aligned}$$

Therefore:

$$\begin{aligned} a_2 &= w_{11}^{(2)} a_{11} + w_{21}^{(2)} a_{12} + w_{01}^{(2)} \\ &= w_{11}^{(2)} \left(w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 + w_{31}^{(1)} x_3 + w_{01}^{(1)} \right) \\ &\quad + w_{21}^{(2)} \left(w_{12}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{32}^{(1)} x_3 + w_{02}^{(1)} \right) \\ &\quad + w_{01}^{(2)} \end{aligned}$$

b) Show that the output of the network can be written

$$a_2 = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3,$$

where w_0, w_1, w_2, and w_3 are scalars that depend only on the weights in the original network. By showing this, you're proving that the network above is equivalent to a much simpler linear model.

Solution: Starting from the result of the last subproblem:

$$
\begin{aligned}
a_2 = w_{11}^{(2)} &\left(w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 + w_{31}^{(1)} x_3 + w_{01}^{(1)} \right) \\
&+ w_{21}^{(2)} \left(w_{12}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{32}^{(1)} x_3 + w_{02}^{(1)} \right) \\
&+ w_{01}^{(2)}
\end{aligned}
$$

Grouping the terms involving x_1, x_2, and x_3 separately:

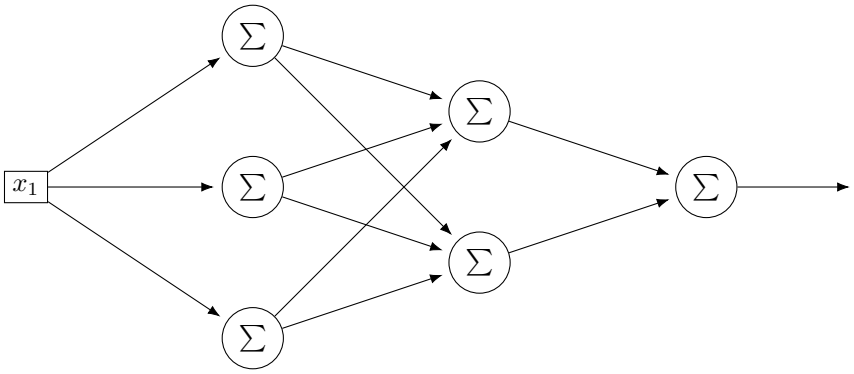
$$
\begin{aligned}
&= \left(w_{11}^{(2)} w_{11}^{(1)} + w_{21}^{(2)} w_{12}^{(1)} \right) x_1 \\
&\quad + \left(w_{11}^{(2)} w_{21}^{(1)} + w_{21}^{(2)} w_{22}^{(1)} \right) x_2 \\
&\quad + \left(w_{11}^{(2)} w_{31}^{(1)} + w_{21}^{(2)} w_{32}^{(1)} \right) x_3 \\
&\quad + w_{01}^{(2)} + w_{11}^{(2)} w_{01}^{(1)} + w_{21}^{(2)} w_{02}^{(1)} \\
&= w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3
\end{aligned}
$$

where

$$
\begin{aligned}
w_1 &= w_{11}^{(2)} w_{11}^{(1)} + w_{21}^{(2)} w_{12}^{(1)} \\
w_2 &= w_{11}^{(2)} w_{21}^{(1)} + w_{21}^{(2)} w_{22}^{(1)} \\
w_3 &= w_{11}^{(2)} w_{31}^{(1)} + w_{21}^{(2)} w_{32}^{(1)} \\
w_0 &= w_{01}^{(2)} + w_{11}^{(2)} w_{01}^{(1)} + w_{21}^{(2)} w_{02}^{(1)}
\end{aligned}
$$

Problem 3. (1 credit)

Consider the neural network architecture shown below:



In all parts of this problem, assume that the network's parameters are:

$$
W^{(1)} = \begin{pmatrix} -3 & -5 & 4 \end{pmatrix} \qquad W^{(2)} = \begin{pmatrix} 2 & -3 \\ -5 & 5 \\ -3 & 0 \end{pmatrix} \qquad W^{(3)} = \begin{pmatrix} 5 \\ -4 \end{pmatrix}
$$

$$
\vec{b}^{(1)} = \begin{pmatrix} 3 \\ 2 \\ -1 \end{pmatrix} \qquad \vec{b}^{(2)} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \qquad \vec{b}^{(3)} = \begin{pmatrix} -3 \end{pmatrix}
$$

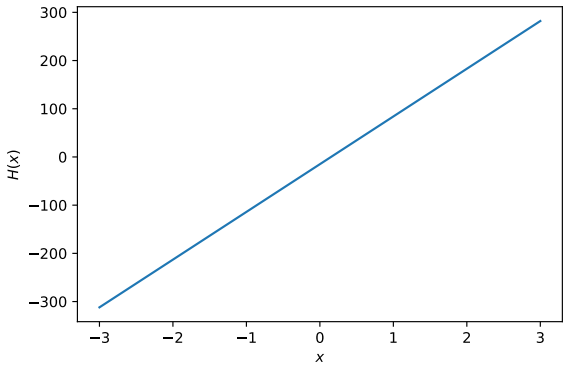
Note that this network is a function $H : \mathbb{R} \to \mathbb{R}$, so we can easily plot it. In the parts below, you will plot

the network for a range of inputs. Your plots may not necessarily be *interesting*, but they will give you a sense of how different choices of activation function affect the type of function the neural network computes.

Suggestion: create a Python function `network(x, activation)` which takes in two things: a number `x` and an `activation` function, and computes the output of the network on x using that activation function (that is, it computes $H(x)$). You can then use that code for all parts of this problem.

a) Assume that all activation functions are linear. Plot $H(x)$ in the range $x \in [-3, 3]$. Show your code.

Solution:



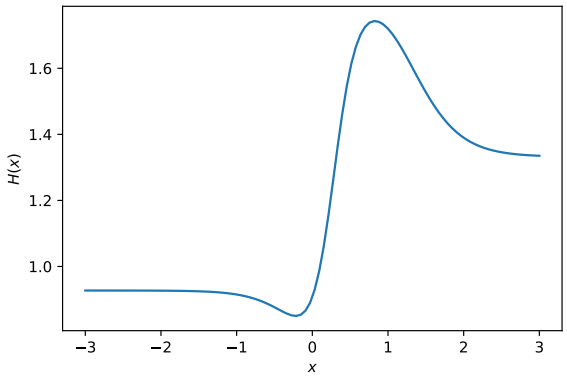
This was generated by the following code, which was also used to generate the images in the other subproblems.

```
"""This code isn't the most *efficient* way to implement a neural
network, but it is maybe the simplest."""

import numpy as np
import matplotlib.pyplot as plot


def linear(x):
    return x

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

def relu(x):
    return np.maximum(0, x)

def network(x, activation=linear):
    x = np.array([x])

    W_1 = np.array([-3, -5, 4])[:,None]
    b_1 = np.array([3, 2, -1])

    W_2 = np.array([
        [2, -3],
        [-5, 5],
        [-3, 0]
    ])
    b_2 = np.array([1, 2])
```

```
    W_3 = np.array([5, -4])
    b_3 = np.array([-3])

    def H_1(z):
        return activation(W_1 @ z + b_1)

    def H_2(z):
        return activation(W_2.T @ z + b_2)

    def H_3(z):
        # output node uses linear activation
        return W_3 @ z + b_3

    return H_3(H_2(H_1(x)))[0]

def H_linear(x):
    return network(x, activation=linear)

def H_relu(x):
    return network(x, activation=relu)

def H_sigmoid(x):
    return network(x, activation=sigmoid)

def plot(H):
    xx = np.linspace(-3, 3, 100)
    yy = [H(x) for x in xx]
    plt.plot(xx, yy)

    plt.xlabel('$x$')
    plt.ylabel('$H(x)$')
```

b) Assume that all hidden nodes have **sigmoid** activation and that the output node has linear activation. Plot $H(x)$ in the range $x \in [-3, 3]$. Show your code.

Solution:



c) Assume that all hidden nodes have **ReLU** activation and that the output node has linear activation. Plot $H(x)$ in the range $x \in [-3, 3]$. Show your code.

Solution: