ANJALI PATIL
Machine Learning
12140210

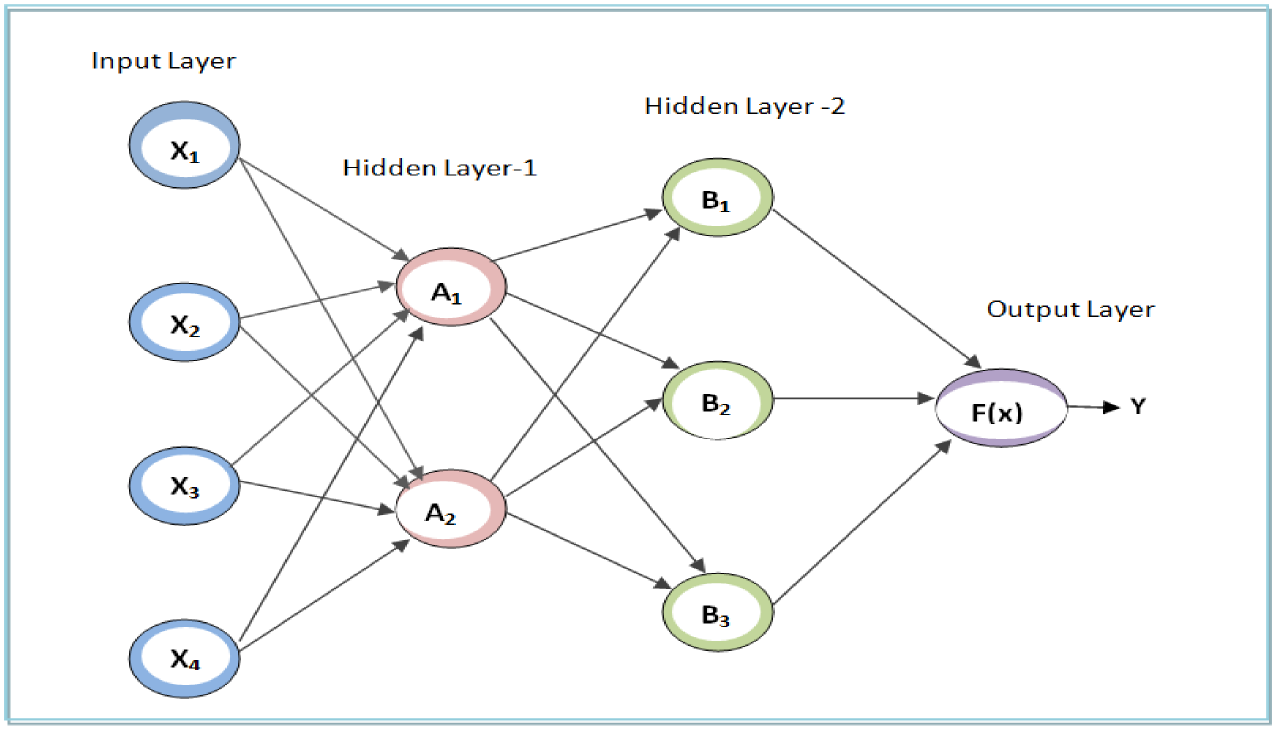
Homework 4: Neural Network Theory

Ch10 Q1

Consider a neural network with two hidden layers: p = 4 input units, 2 units in the first hidden layer, 3 units in the second hidden layer, and a single output.

(a) Draw a picture of the network, similar to Figures 10.1 or 10.4.
(b) Write out an expression for f(X), assuming ReLU activation functions. Be as explicit as you can!
(c) Now plug in some values for the coefcients and write out the value of f(X).
(d) How many parameters are there?

(a)



(b)

Let's define the weights and biases as variables for each connection and node:
1) $w_{i,j}^{(k)}$ represents the weight from node i in layer $k-1$ to node j in layer k.
2) $b_j^{(k)}$ represents the bias for node j in layer k.
Given an input vector $X = [X_1, X_2, X_3, X_4]$, you can express $f(X)$ as follows:
1. Calculate the input for the first hidden layer:

$$Z_1^{(1)} = X_1 \cdot w_{1,1}^{(1)} + X_2 \cdot w_{2,1}^{(1)} + X_3 \cdot w_{3,1}^{(1)} + X_4 \cdot w_{4,1}^{(1)} + b_1^{(1)}$$
$$Z_2^{(1)} = X_1 \cdot w_{1,2}^{(1)} + X_2 \cdot w_{2,2}^{(1)} + X_3 \cdot w_{3,2}^{(1)} + X_4 \cdot w_{4,2}^{(1)} + b_2^{(1)}$$

2. Apply ReLU activation to the first hidden layer:

$$A_1 = \max(0, Z_1^{(1)})$$
$$A_2 = \max(0, Z_2^{(1)})$$

3. Calculate the input for the second hidden layer:

$$Z_1^{(2)} = A_1 \cdot w_{1,1}^{(2)} + A_2 \cdot w_{2,1}^{(2)} + b_1^{(2)}$$
$$Z_2^{(2)} = A_1 \cdot w_{1,2}^{(2)} + A_2 \cdot w_{2,2}^{(2)} + b_2^{(2)}$$
$$Z_3^{(2)} = A_1 \cdot w_{1,3}^{(2)} + A_2 \cdot w_{2,3}^{(2)} + b_3^{(2)}$$

4. Apply ReLU activation to the second hidden layer:

$$B_1 = \max(0, Z_1^{(2)})$$
$$B_2 = \max(0, Z_2^{(2)})$$
$$B_3 = \max(0, Z_3^{(2)})$$

5. Calculate the output:

$$Z_1^{(3)} = B_1 \cdot w_{1,1}^{(3)} + B_2 \cdot w_{2,1}^{(3)} + B_3 \cdot w_{3,1}^{(3)} + b_1^{(3)}$$
$$f(X) = \max(0, Z_1^{(3)})$$

(c)

Let's consider some arbitrary values for the weights and biases:
For the first hidden layer:

$$w_{1,1}^{(1)} = 0.5, w_{2,1}^{(1)} = 0.3, w_{3,1}^{(1)} = -0.2, w_{4,1}^{(1)} = 0.7$$
$$w_{1,2}^{(1)} = 0.1, w_{2,2}^{(1)} = -0.3, w_{3,2}^{(1)} = 0.4, w_{4,2}^{(1)} = -0.5$$
$$b_1^{(1)} = 0.1, b_2^{(1)} = -0.2$$

For the second hidden layer:

$$w_{1,1}^{(2)} = 0.6, w_{2,1}^{(2)} = -0.4$$
$$w_{1,2}^{(2)} = 0.2, w_{2,2}^{(2)} = 0.3$$
$$w_{1,3}^{(2)} = -0.1, w_{2,3}^{(2)} = 0.5$$
$$b_1^{(2)} = 0.3, b_2^{(2)} = -0.1, b_3^{(2)} = 0.2$$

For the output layer:

$$w_{1,1}^{(3)} = 0.7, w_{2,1}^{(3)} = -0.2, w_{3,1}^{(3)} = 0.4$$
$$b_1^{(3)} = 0.1$$

Now, let's calculate f(X) with a sample input vector $X = [1, 0.5, 0.8, 0.2]$:
1. Calculate the input for the first hidden layer:

$$Z_1^{(1)} = 1 \cdot 0.5 - 0.5 \cdot 0.3 + 0.8 \cdot (-0.2) - 0.2 \cdot 0.7 + 0.1 = 0.13$$
$$Z_2^{(1)} = 1 \cdot 0.1 - 0.5 \cdot (-0.3) + 0.8 \cdot 0.4 - 0.2 \cdot (-0.5) - 0.2 = 0.54$$

2. Apply ReLU activation to the first hidden layer:

$$A_1^{(1)} = \max(0, 0.13) = 0.13$$
$$A_2^{(1)} = \max(0, 0.54) = 0.54$$

3. Calculate the input for the second hidden layer:

$$Z_1^{(2)} = 0.13 \cdot 0.6 + 0.54 \cdot (-0.4) + 0.3 = -0.119$$
$$Z_2^{(2)} = 0.13 \cdot 0.2 + 0.54 \cdot 0.3 - 0.1 = 0.13$$
$$Z_3^{(2)} = 0.13 \cdot (-0.1) + 0.54 \cdot 0.5 + 0.2 = 0.375$$

4. Apply ReLU activation to the second hidden layer:

$$A_1^{(2)} = \max(0, -0.119) = 0$$
$$A_2^{(2)} = \max(0, 0.13) = 0.13$$
$$A_3^{(2)} = \max(0, 0.375) = 0.375$$

5. Calculate the output with ReLU activation:

$$Z_1^{(3)} = 0 \cdot 0.7 + 0.13 \cdot (-0.2) + 0.375 \cdot 0.4 + 0.1 = 0.195$$
$$f(X) = \max(0, 0.195) = 0.195$$

(d)

To calculate the total number of parameters in the given neural network, we can break it down as follows:
1. For the first hidden layer:
 - Weights: $4 * 2 = 8$.
 - Biases: 2 .
2. For the second hidden layer:
 - Weights: $3 * 2 = 6$.
 - Biases: 3.
3. For the output layer:
 - Weights: 3 .
 - Biases: 1.

Total number of Parameter: $8 + 2 + 6 + 3 + 3 + 1 = 23$
So, there are a total of 23 parameters in the given neural network.

Ch10 Q2 Consider the softmax function in (10.13) (see also (4.13) on page 141) for modeling multinomial probabilities.

(a) In (10.13), show that if we add a constant c to each of the z_ℓ, then the probability is unchanged.
(b) In (4.13), show that if we add constants $c_j, j = 0, 1, \ldots, p$, to each of the corresponding coefficients for each of the classes, then the predictions at any new point x are unchanged.

(a)

$$f_m(X) = Pr(Y = m \mid X) = \frac{e^{Z_m}}{\sum_{l=0}^{9} e^{Z_l}},$$

Suppose we add constant c to each of Z_l that we have,

$$\frac{e^{Z_m+c}}{\sum_{l=0}^{9} e^{Z_l+c}} = \frac{e^c \cdot e^{Z_m}}{\sum_{l=0}^{9} e^c \cdot e^{Z_l}}$$
$$= \frac{e^c}{e^c} \cdot \frac{e^{Z_m}}{\sum_{l=0}^{9} e^{Z_l}} = \frac{e^{Z_m}}{\sum_{l=0}^{9} e^{Z_l}}$$

$$= \frac{e^{Z_m}}{\sum_{l=0}^{9} e^{Z_l}} = Pr(Y = m \mid X)$$

Thus Probability does not change.

(b)
Expression in 4.13 is:

$$Pr(Y = k|X = x) = \frac{e^{\beta_{k_0}+\beta_{k_1}x_1+\ldots+\beta_{k_p}x_p}}{\sum_{l=1}^{K} e^{\beta_{l_0}+\beta_{l_1}x_1+\ldots+\beta_{l_p}x_p}}$$

Now , we add constants $c_j, j = 0, 1, \ldots, p$, to each of the corresponding coefficients for each of the classes:

$$Pr(Y = k|X = x) = \frac{e^{\beta_{k_0}+\beta_{k_1}x_1+c_1+\ldots+\beta_{k_p}x_p+c_p}}{\sum_{l=1}^{K} e^{\beta_{l_0}+\beta_{l_1}x_1+c_1+\ldots+\beta_{l_p}x_p+c_p}}$$

$$= \frac{e^{c_1+\ldots+c_p} \cdot e^{\beta_{k_0}+\beta_{k_1}x_1+\ldots+\beta_{k_p}x_p}}{\sum_{l=1}^{K} e^{c_1+\ldots+c_p} \cdot e^{\beta_{l_0}+\beta_{l_1}x_1+\ldots+\beta_{l_p}x_p}}$$

$$= \frac{e^{c_1+\ldots+c_p}}{e^{c_1+\ldots+c_p}} \cdot \frac{e^{\beta_{k_0}+\beta_{k_1}x_1+\ldots+\beta_{k_p}x_p}}{\sum_{l=1}^{K} e^{\beta_{l_0}+\beta_{l_1}x_1+\ldots+\beta_{l_p}x_p}}$$

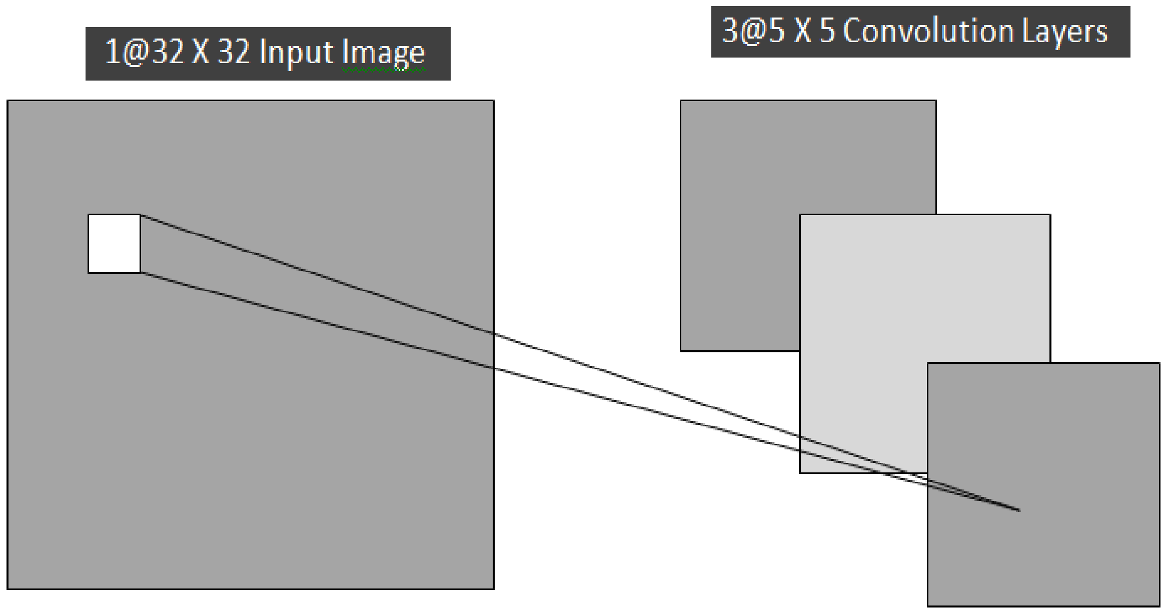
$$= \frac{e^{\beta_{k_0}+\beta_{k_1}x_1+\ldots+\beta_{k_p}x_p}}{\sum_{l=1}^{K} e^{\beta_{l_0}+\beta_{l_1}x_1+\ldots+\beta_{l_p}x_p}}$$

$$= P(Y = k|X = x)$$

The predictions at any new point x are unchanged as prediction probability is unchanged.

Ch10 Q4
Consider a CNN that takes in 32 × 32 grayscale images and has a single convolution layer with three 5 × 5 convolution filters (without boundary padding).
(a) Draw a sketch of the input and first hidden layer similar to Figure 10.8.
(b) How many parameters are in this model?
(c) Explain how this model can be thought of as an ordinary feed-forward neural network with the individual pixels as inputs, and with constraints on the weights in the hidden units. What are the constraints?
(d) If there were no constraints, then how many weights would there be in the ordinary feed-forward neural network in (c)?

(a)



(b)
For each convolution filter, there are 5x5 = 25 weights. Since there are three filters in the convolution layer, there are a total of 3 x 25 = 75 weights.

Each filter also has one bias term, so there are 3 biases.

Therefore, the total number of parameters in this model is 75 (weights) + 3 (biases) = 78 parameters.

(c)
This model can be thought of as an ordinary feed-forward neural network with the individual pixels as inputs by considering that each filter in the convolutional layer operates on the input in a specific way, similar to how neurons in a fully connected layer would operate.

The constraints in this model arise from the weight sharing in the convolutional layer. Each weight in a filter is shared across the entire input image. So, the constraint is that the weights are tied to the same values across different spatial positions in the input. This constraint is what allows the model to detect local patterns and features, making it translationally invariant. In a fully connected feed-forward neural network, each weight is unique, but in the convolutional layer, the same weights are used for different parts of the image.

(d)
With no constraints, we would connect each output pixel in our $3 * 5 * 5$ convolution layer to each node in the $32x32$ original image (plus 3 bias terms) $[3 * 5 * 5 * (32 * 32) + 3 = 76803]$, giving a total of 76,803 weights to estimate.

Ch10 Q6
Consider the simple function $R(\beta) = sin(\beta) + \beta/10$.

(a) Draw a graph of this function over the range $\beta \in [6, 6]$.
(b) What is the derivative of this function?

(c) Given $\beta^0 = 2.3$, run gradient descent to find a local minimum of $R(\beta)$ using a learning rate of $\rho = 0.1$. Show each of $\beta^0, \beta^1, \ldots$ in your plot, as well as the final answer.
(d) Repeat with $\beta^0 = 1.4$

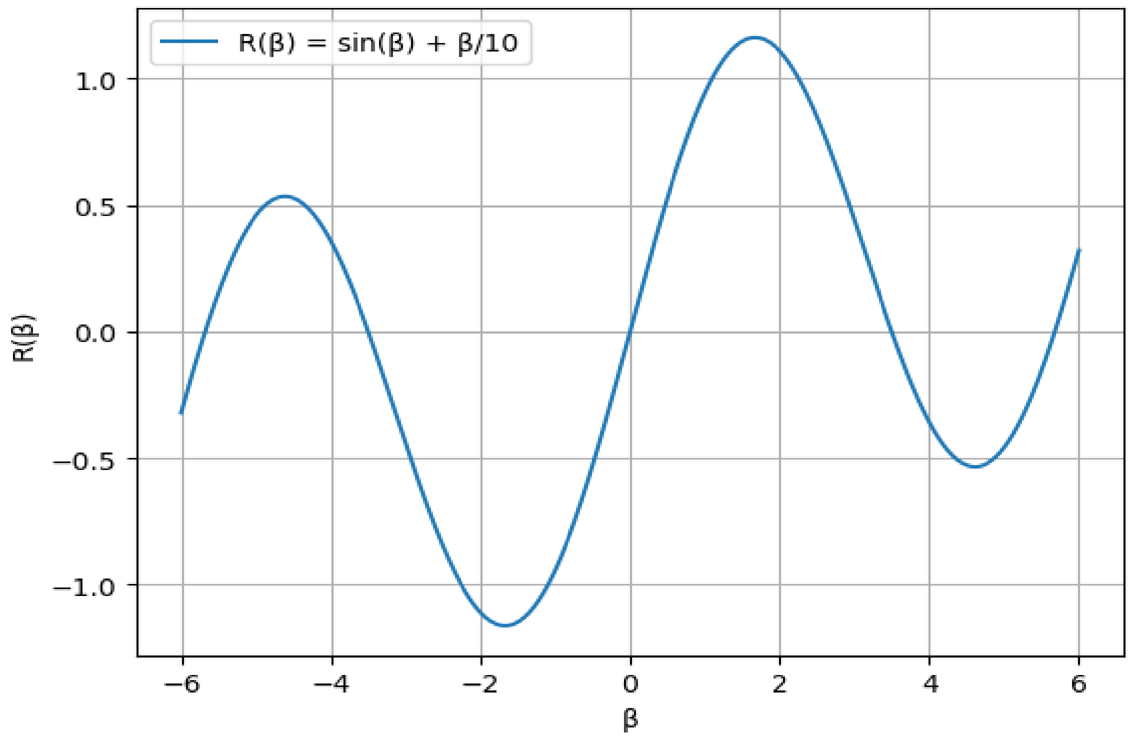
(a)

CODE:

```
# (a) Plot the function R(β)
beta_values = np.linspace(-6, 6, 400)
R_values = R(beta_values)
plt.plot(beta_values, R_values, label='R(β) = sin(β) + β/10')

# Add labels and legend
plt.xlabel('β')
plt.ylabel('R(β)')
plt.legend()
# Show the plot
plt.grid()
plt.show()
```

GRAPH:



(b)

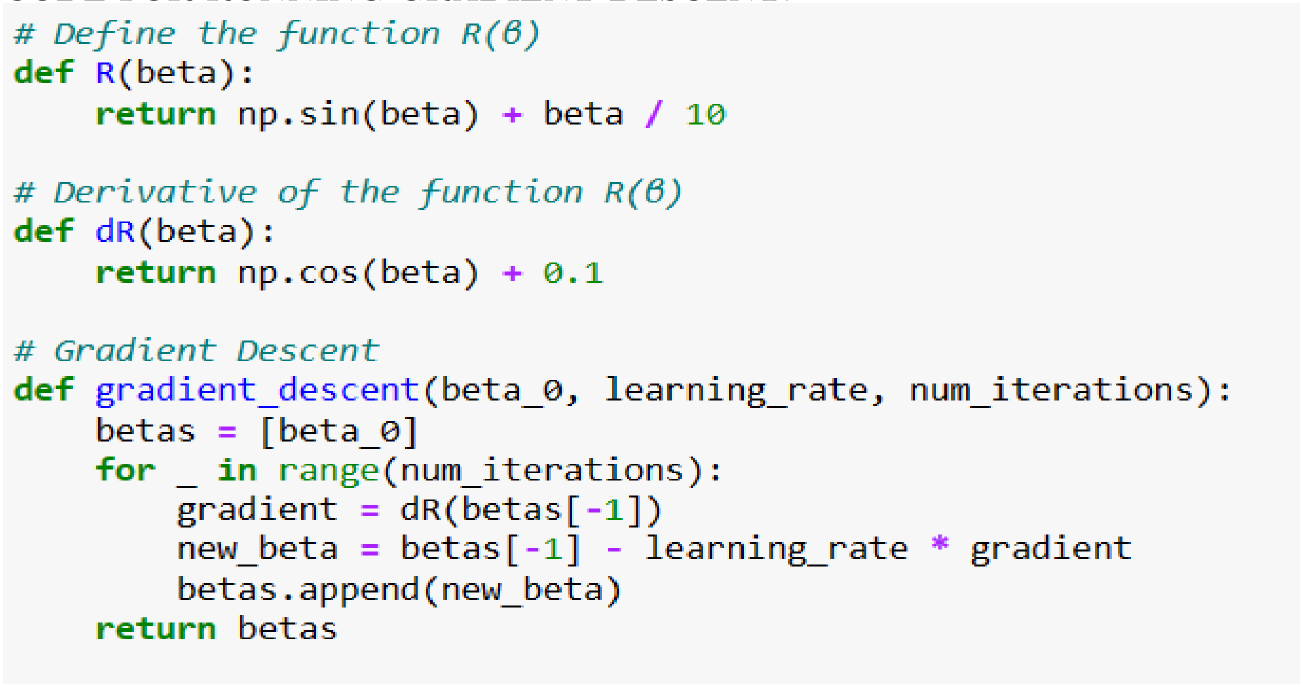
$$R'(\beta) = \frac{d}{d\beta}\left(\sin(\beta) + \frac{\beta}{10}\right)$$

$$= \frac{d}{d\beta}\sin(\beta) + \frac{d}{d\beta}\left(\frac{\beta}{10}\right) = \cos(\beta) + \frac{1}{10}$$

Derivative of the function $R(\beta)$ is: $\cos(\beta) + 0.1$

(c)

CODE FOR RUNNING GRADIENT DESCENT:



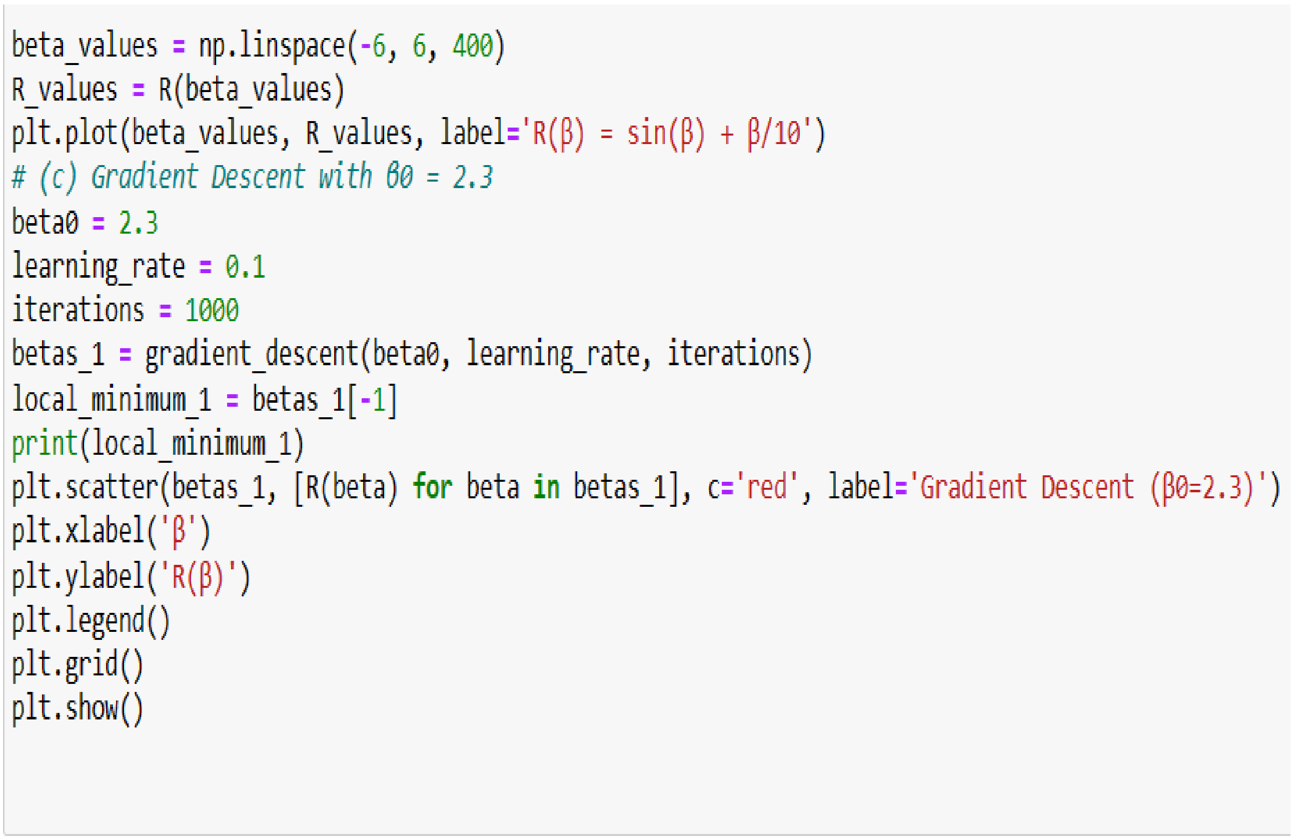
```python
# Define the function R(β)
def R(beta):
    return np.sin(beta) + beta / 10

# Derivative of the function R(β)
def dR(beta):
    return np.cos(beta) + 0.1

# Gradient Descent
def gradient_descent(beta_0, learning_rate, num_iterations):
    betas = [beta_0]
    for _ in range(num_iterations):
        gradient = dR(betas[-1])
        new_beta = betas[-1] - learning_rate * gradient
        betas.append(new_beta)
    return betas
```

GRADIENT DESCENT WITH $\beta^0 = 2.3$

CODE:

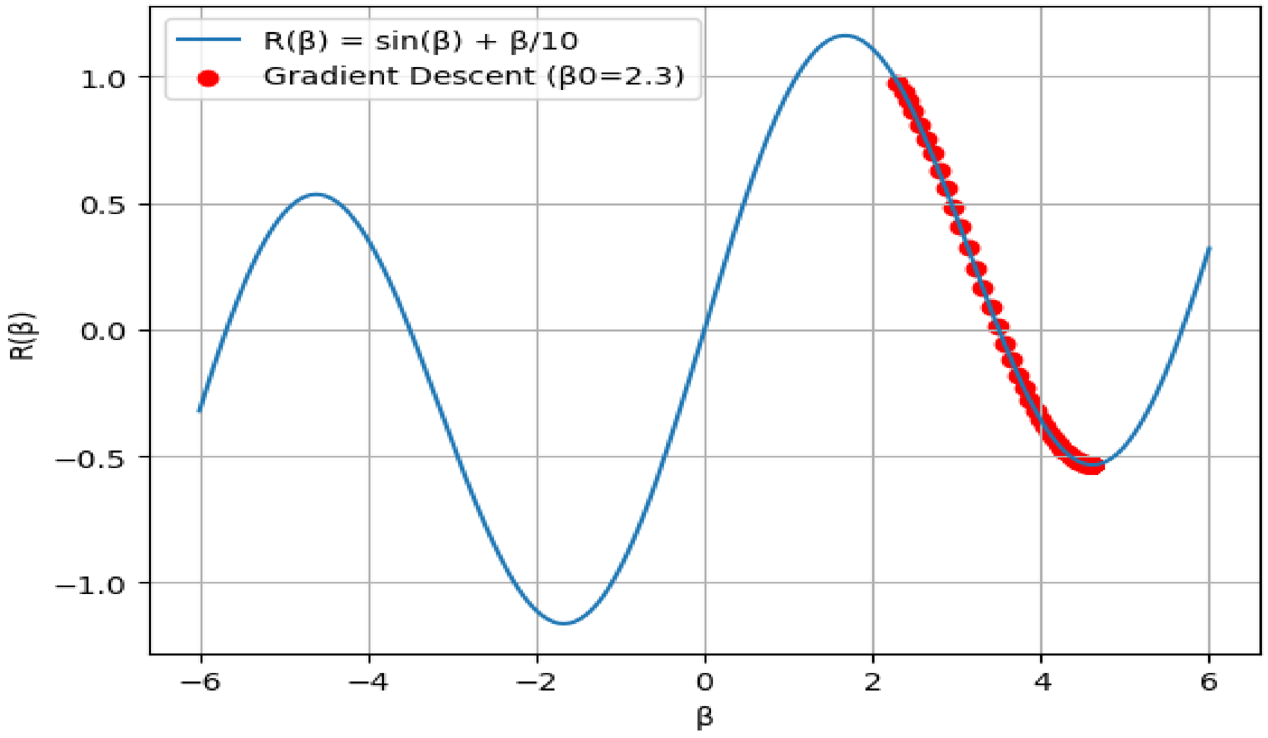


```
beta_values = np.linspace(-6, 6, 400)
R_values = R(beta_values)
plt.plot(beta_values, R_values, label='R(β) = sin(β) + β/10')
# (c) Gradient Descent with β0 = 2.3
beta0 = 2.3
learning_rate = 0.1
iterations = 1000
betas_1 = gradient_descent(beta0, learning_rate, iterations)
local_minimum_1 = betas_1[-1]
print(local_minimum_1)
plt.scatter(betas_1, [R(beta) for beta in betas_1], c='red', label='Gradient Descent (β0=2.3)')
plt.xlabel('β')
plt.ylabel('R(β)')
plt.legend()
plt.grid()
plt.show()
```

```
4.612221559223126
```

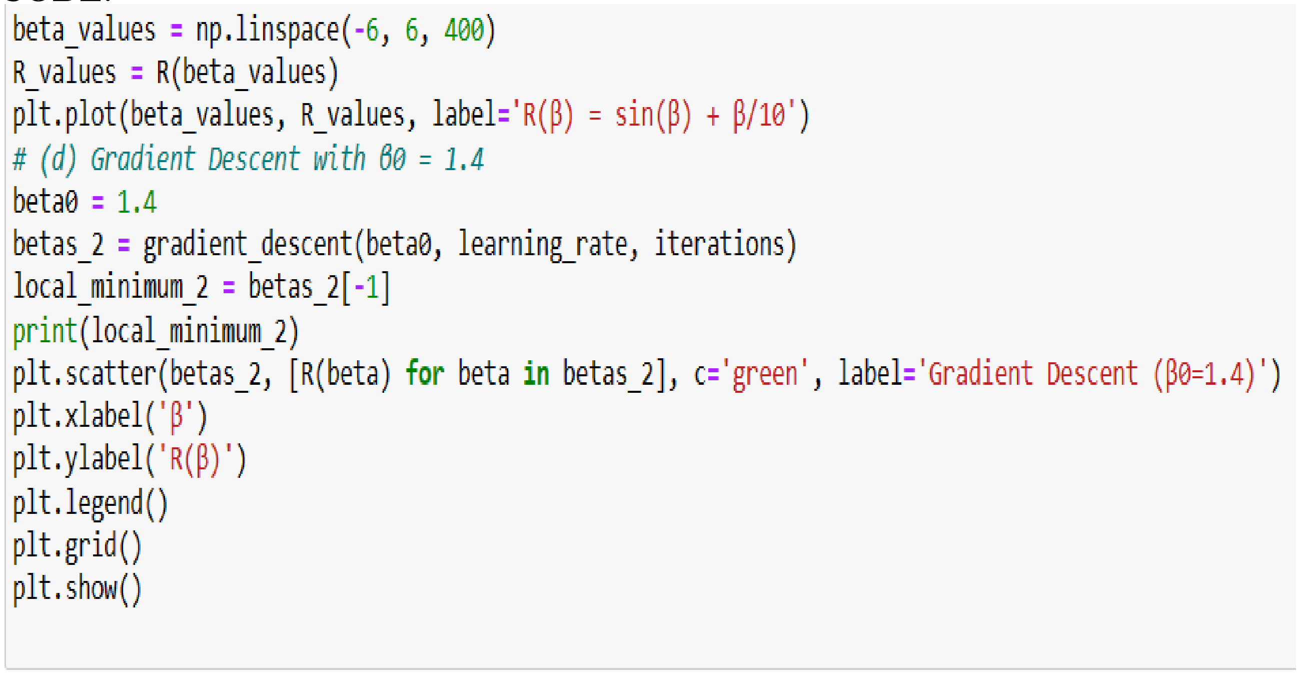
Value of Local Minimum: 4.6122

GRAPH:



(d)
GRADIENT DESCENT WITH $\beta^0 = 1.4$

CODE:

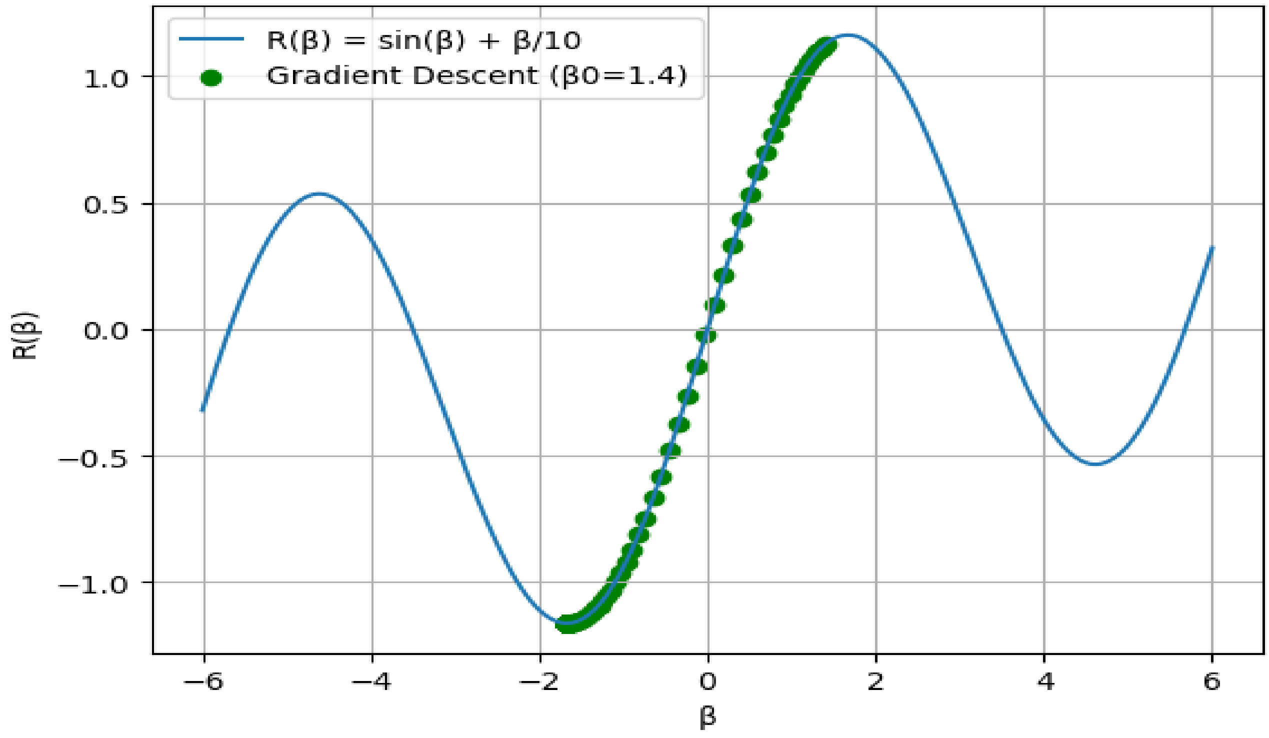


```
beta_values = np.linspace(-6, 6, 400)
R_values = R(beta_values)
plt.plot(beta_values, R_values, label='R(β) = sin(β) + β/10')
# (d) Gradient Descent with β0 = 1.4
beta0 = 1.4
betas_2 = gradient_descent(beta0, learning_rate, iterations)
local_minimum_2 = betas_2[-1]
print(local_minimum_2)
plt.scatter(betas_2, [R(beta) for beta in betas_2], c='green', label='Gradient Descent (β0=1.4)')
plt.xlabel('β')
plt.ylabel('R(β)')
plt.legend()
plt.grid()
plt.show()
```

```
-1.6709637479564554
```

Value of Local Minimum: -1.6709

GRAPH:


R(β) = sin(β) + β/10
Gradient Descent (β0=1.4)
R(β)
β