You are working as a machine learning engineer to develop a neural network-based classifier for detecting cars in grayscale images. Each image is 50 × 50 pixels, represented as a 2,500-dimensional feature vector. You will follow the steps of forward propagation, backpropagation, and evaluation, performing key calculations at each stage.

Task 1: Forward Propagation (Computation Required)

Given:

- Input feature vector X of size 2500×1.
- Weight matrix $W^{(1)}$ for the first hidden layer has dimensions 128×2500.
- Bias vector $b^{(1)}$ is of size 128×1
- Activation function in the hidden layer: ReLU (ReLU(z)=max(0,z)).
- Second layer has 1 neuron with a sigmoid activation.
- Weight matrix $W^{(2)}$ has size 1×128.
- Bias $b^{(2)}$ is a scalar.

Your Tasks:

1. Compute the first layer activation $A^{(1)}$ using:

$$Z^{(1)} = W^{(1)} X + b^{(1)}$$

$$A^{(1)} = \max(0, Z^{(1)})$$

Assume:

$$W^{(1)} = \begin{bmatrix} 0.02 & -0.01 & \dots & 0.005 \\ -0.03 & 0.04 & \dots & -0.002 \\ \vdots & \vdots & \ddots & \vdots \\ 0.01 & -0.02 & \dots & 0.03 \end{bmatrix},$$

$b^{(1)} = \mathbf{0}$ (vector of zeros).

2. Compute the second layer output:

$$Z^{(2)} = W^{(2)} A^{(1)} + b^{(2)}$$

Assume:

$$W^{(2)} = \begin{bmatrix} 0.05 & -0.02 & 0.01 & \dots & 0.02 \end{bmatrix},$$

$b^{(2)} = -0.1.$

3. Apply **sigmoid activation** to obtain the final prediction $A^{(2)}$:

$$A^{(2)} = \frac{1}{1 + e^{-Z^{(2)}}}$$

Compute the **final predicted probability** $\boldsymbol{A^{(2)}}$ given a random input X where each feature is drawn from a uniform distribution between 0 and 1.

Task 2: Backpropagation (Compute Gradients)

Using the **cross-entropy loss function**:

$$J = -[y \log(A^{(2)}) + (1-y)\log(1-A^{(2)})]$$

where $y = 1$ if the image contains a car and $y = 0$ otherwise.

Your Tasks:

1. Compute the derivative of the loss with respect to $Z^{(2)}$:

$$\frac{\partial J}{\partial Z^{(2)}} = A^{(2)} - y$$

   Assume $y = 1$.

2. Compute the gradient of the cost function with respect to $W^{(2)}$:

3. Compute the gradient of the cost function with respect to $W^{(1)}$ using **backpropagation through ReLU**:

$$\frac{\partial J}{\partial W^{(1)}} = \left(\left(W^{(2)}\right)^T \frac{\partial J}{\partial Z^{(2)}}\right) \circ 1(Z^{(1)} > 0)X^T$$

   where $\circ$ represents element-wise multiplication.

Compute the gradients for $W^{(2)}$ and $W^{(1)}$ based on the forward propagation output from Task 1. And tell me the shape of dW2, minimum element of dW2 and maximum element of dB2.

Task 3: Model Evaluation

You trained the neural network for 10 epochs and obtained the following confusion matrix on the test set:

	Predicted: Car (1)	**Predicted: Not Car (0)**
Actual: Car (1)	150	50
Actual: Not Car (0)	30	270

Calculate the accuracy, precision, and recall of the classifier using the confusion matrix.

Task 4: Multi-class Classification

Your task is now to extend the model to classify **four different vehicle types**:

- Pedestrian (Class 1)
- Car (Class 2)
- Motorcycle (Class 3)
- Truck (Class 4)

Instead of a single output neuron, you now use a softmax output layer with 4 neurons.

Your task:

- Implement the **softmax function** for output layer activation:

$$A_j = \frac{e^{Z_j}}{\sum_{k=1}^{4} e^{Z_k}}, \quad \forall j \in \{1,2,3,4\}$$

Given:

$$Z = [2.1, 1.4, 0.5, -0.2]$$

2. Ensure numerical stability by subtracting the **maximum value** from all logits:

$$Z' = Z - \max(Z)$$

Compute the probability of each class using the softmax activation function.

General instruction for your Python Code:

```
import numpy as np

def sigmoid(z):
    #YOUR CODE HERE

def relu(z):
     #YOUR CODE HERE

def softmax(z):
     #YOUR CODE HERE

def forward_propagation(X, W1, b1, W2, b2):
     #YOUR CODE HERE

def compute_gradients(X, A1, A2, Y, W2, Z1):
     #YOUR CODE HERE

def evaluate_model(confusion_matrix):
    #YOUR CODE HERE


# Fixed Data (Non-Random)
X = np.ones((2500, 1)) * 0.5  # Fixed input values
W1 = np.linspace(-0.01, 0.01, num=2500 * 128).reshape(128, 2500)
b1 = np.zeros((128, 1))
W2 = np.linspace(-0.01, 0.01, num=128).reshape(1, 128)
b2 = np.zeros((1, 1))
Y = np.array([[1]])  # Assume car is present

# Forward propagation
Z1, A1, Z2, A2 = forward_propagation(X, W1, b1, W2, b2)
print("Predicted Probability (A2):", A2)

# Compute gradients
dW1, dB1, dW2, dB2 = compute_gradients(X, A1, A2, Y, W2, Z1)
print("Gradient dW2 shape:", dW2.shape)


# Confusion Matrix
conf_matrix = np.array([[150, 50], [30, 270]])
accuracy, precision, recall = evaluate_model(conf_matrix)
print(f"Accuracy: {accuracy:.2f}, Precision: {precision:.2f}, Recall: {recall:.2f}")

# Multi-Class Softmax Example
Z_multiclass = np.array([2.1, 1.4, 0.5, -0.2]).reshape(-1, 1)
A_softmax = softmax(Z_multiclass)
print("Softmax Probabilities:", A_softmax.flatten())
```

import numpy as np

```python
def sigmoid(z):
   #YOUR CODE HERE

def relu(z):
    #YOUR CODE HERE

def softmax(z):
    #YOUR CODE HERE

def forward_propagation(X, W1, b1, W2, b2):
    #YOUR CODE HERE

def compute_gradients(X, A1, A2, Y, W2, Z1):
    #YOUR CODE HERE

def evaluate_model(confusion_matrix):
   #YOUR CODE HERE


# Fixed Data (Non-Random)
X = np.ones((2500, 1)) * 0.5  # Fixed input values
W1 = np.linspace(-0.01, 0.01, num=2500 * 128).reshape(128, 2500)
b1 = np.zeros((128, 1))
W2 = np.linspace(-0.01, 0.01, num=128).reshape(1, 128)
b2 = np.zeros((1, 1))
Y = np.array([[1]])  # Assume car is present

# Forward propagation
Z1, A1, Z2, A2 = forward_propagation(X, W1, b1, W2, b2)
print("Predicted Probability (A2):", A2)

# Compute gradients
dW1, dB1, dW2, dB2 = compute_gradients(X, A1, A2, Y, W2, Z1)
print("Gradient dW2 shape:", dW2.shape)

# Confusion Matrix
conf_matrix = np.array([[150, 50], [30, 270]])
accuracy, precision, recall = evaluate_model(conf_matrix)
print(f"Accuracy: {accuracy:.2f}, Precision: {precision:.2f}, Recall: {recall:.2f}")

# Multi-Class Softmax Example
Z_multiclass = np.array([2.1, 1.4, 0.5, -0.2]).reshape(-1, 1)
A_softmax = softmax(Z_multiclass)
print("Softmax Probabilities:", A_softmax.flatten())
```