Assignment 3

Problem Statement 1

You are given a neural network with the following structure:

- **Input layer**: 3 units
- **Hidden layer 1**: 2 units, ReLU activation
- **Hidden layer 2**: 2 units, ReLU activation
- **Output layer**: 1 unit, no activation

The loss function used is the squared loss:

$$L = \frac{1}{2}(y - \hat{y})^2$$

Provided Forward Pass Values:

- **Input**: $x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$
- **Hidden Layer 1 Output**: $a^{(1)} = \begin{bmatrix} 4 \\ 5 \end{bmatrix}$
- **Hidden Layer 2 Output**: $a^{(2)} = \begin{bmatrix} 6 \\ 7 \end{bmatrix}$
- **Output**: $\hat{y} = 8$

Weights and Biases:

- $W^{(1)} = \begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \end{bmatrix}$, $b^{(1)} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}$
- $W^{(2)} = \begin{bmatrix} 0.7 & 0.8 \\ 0.9 & 1.0 \end{bmatrix}$, $b^{(2)} = \begin{bmatrix} 0.3 \\ 0.4 \end{bmatrix}$
- $W^{(3)} = \begin{bmatrix} 1.1 & 1.2 \end{bmatrix}$, $b^{(3)} = 0.5$

The true output is $y = 10$.

Solution: Backward Pass Gradients

Step 1: Gradient of Loss with respect to Output $\hat{y}$

The loss function is:

$$L = \frac{1}{2}(y - \hat{y})^2$$

The gradient of the loss with respect to $\hat{y}$ is:

$$\frac{\partial L}{\partial \hat{y}} = -(y - \hat{y}) = -(10 - 8) = -2$$

Thus:

$$\frac{\partial L}{\partial \hat{y}} = -2$$

Step 2: Gradient of Loss with respect to Input to Output Layer $z^{(3)}$

The input to the output layer is:

$$z^{(3)} = W^{(3)} a^{(2)} + b^{(3)}$$

Substituting values:

$$z^{(3)} = \begin{bmatrix} 1.1 & 1.2 \end{bmatrix} \begin{bmatrix} 6 \\ 7 \end{bmatrix} + 0.5 = 15.5$$

Since there is no activation in the output layer:

$$\frac{\partial L}{\partial z^{(3)}} = \frac{\partial L}{\partial \hat{y}} = -2$$

Step 3: Gradient of Loss with respect to Output of ReLU in Hidden Layer 2 $a^{(2)}$

The output of the second hidden layer is:

$$a^{(2)} = \mathrm{ReLU}(z^{(2)}) = \begin{bmatrix} 6 \\ 7 \end{bmatrix}$$

Now, we calculate the gradient of the loss with respect to $a^{(2)}$:

$$\frac{\partial L}{\partial a^{(2)}} = \frac{\partial L}{\partial z^{(3)}} W^{(3)} = -2 \times \begin{bmatrix} 1.1 \\ 1.2 \end{bmatrix} = \begin{bmatrix} -2.2 \\ -2.4 \end{bmatrix}$$

Step 4: Gradient of Loss with respect to Input to Hidden Layer 2 $z^{(2)}$

Since ReLU is active for $z^{(2)} > 0$, the gradient with respect to $z^{(2)}$ is the same as for $a^{(2)}$:

$$\frac{\partial L}{\partial z^{(2)}} = \begin{bmatrix} -2.2 \\ -2.4 \end{bmatrix}$$

Step 5: Gradient of Loss with respect to Output of ReLU in Hidden Layer 1 $a^{(1)}$

We use the chain rule to find:

$$\frac{\partial L}{\partial a^{(1)}} = \left(\frac{\partial L}{\partial z^{(2)}}\right)^T W^{(2)} = [-2.2 \quad -2.4] \begin{bmatrix} 0.7 & 0.8 \\ 0.9 & 1.0 \end{bmatrix} = \begin{bmatrix} -4.54 \\ -5.18 \end{bmatrix}$$

Step 6: Gradient of Loss with respect to Input to Hidden Layer 1 $z^{(1)}$

Finally, since ReLU is the activation function for the first hidden layer:

$$\frac{\partial L}{\partial z^{(1)}} = \frac{\partial L}{\partial a^{(1)}} = \begin{bmatrix} -4.54 \\ -5.18 \end{bmatrix}$$

Problem Statement 2

Modified Code

Double-click (or enter) to edit

```
import numpy as np
```

Explanation of Modification

The code is almost correct, but we need to ensure that the post-activation gradient `dA` is properly handled during the backward pass through the ReLU activation function.

- **Condition Clarification**: In a ReLU activation function, the output is zero for any input $Z \leq 0$. Therefore, during the backward pass, the gradient of the loss with respect to Z (`dZ`) should also be zero wherever Z was zero or negative. This ensures that the gradient flows correctly through the ReLU function.

The key modification is ensuring that:

$$dZ[Z \leq 0] = 0$$

This line sets the gradient to zero for non-positive values of Z, correctly implementing the backward propagation for the ReLU activation function.

```
def relu_backward(dA, Z):
"""
Implementing the backward propagation for a single ReLU unit.
Arguments:
dA -- post-activation gradient, of any shape
Z -- activation input, of the same shape as dA
Returns:
dZ -- gradient of the cost with respect to Z
"""
dZ = np.array(dA, copy=True) # Copying dA to dZ
# When Z <= 0, setting dZ to 0
dZ[Z <= 0] = 0
return dZ
```

Done