Machine Learning Basics - Set 3a: Implementing a Single Neuron with NumPy

Dataset: A, B from previous sets and C: a tricker dataset with two features and points belonging either to class 0 or 1.

1. **Understanding Gradient Descent:**

   a) Explain the concept of gradient descent in your own words.

   b) Implement a simple cost function for a single neuron:

   ```
   def cost_function(y_true, y_pred):
       # Your implementation here
       # Hint: Use mean squared error
   ```

   c) Derive the analytical formula for the gradient of the cost function with respect to the weights for a single neuron, assuming a sigmoid activation function.

2. **Implementing a Single Neuron:**

   a) Write a Python function to initialize weights for a single neuron:

   ```
   def initialize_weights(input_dim):
       # Your implementation here
       # Hint: Use np.random.randn()
   ```

   b) Implement the forward pass for a single neuron:

   ```
   def forward_pass(X, weights, bias):
       # Your implementation here
       # Hint: Use np.dot() for matrix multiplication
   ```

   c) Implement a simple activation function (the sigmoid as seen during the lectures):

   ```
   def sigmoid(x):
       # Your implementation here
   ```

3. **Gradient Descent for a Single Neuron:**

   a) Implement a function to calculate the gradients:

   ```
   def calculate_gradients(X, y, y_pred, weights):
       # Your implementation here
       # Hint: Use the formula derived in 1c
   ```

   b) Write a function to update the weights:

   ```
   def update_weights(weights, gradients, learning_rate):
       # Your implementation here
   ```

   c) Implement a simple training loop:

   ```
   def train_neuron(X, y, learning_rate, num_epochs):
       # Your implementation here
       # Use functions from 3a and 3b
       # Return trained weights and cost history
   ```

4. **Training and Evaluation:**

   a) Train your single neuron on the provided dataset using the functions you've implemented.

   b) Plot the cost versus the number of epochs to visualize the training progress. Use matplotlib or any other plotting library.

   c) Evaluate the final predictions of your trained neuron on the training data:

```
def evaluate_neuron(X, y, trained_weights, trained_bias):
    # Your implementation here
    # Calculate and return accuracy
```

Note: Use NumPy (`import numpy as np`) for all numerical operations. Remember to handle potential numerical instabilities, such as division by zero or log of zero.