Implementing an ADALINE neural network for the AND function

1. Importing necessary libraries

We begin by importing the `numpy` library required for the numerical operations involved in our analysis.

```
import numpy as np
```

2. Defining the ADALINE class

Next, we define the `ADALINE` class that implements the Adaptive Linear Neuron model.

- It initialises the weights (including a bias weight) to zero and sets the learning rate for weight updates.
- It implements the identity function as the activation function since ADALINE is a linear model.
- The `predict` method calculates the linear combination of inputs and weights (including bias) and returns the output using the activation function.
- The `fit` method trains the ADALINE model for a specified number of epochs. For each input-target pair, it computes the prediction, calculates the error, updates the weights based on the error, and accumulates the total error for the epoch. The error and sum of squared errors are printed in a formatted table after each epoch.

```
class ADALINE:
    def __init__(self, input_size, learning_rate=0.01):
        self.weights = np.zeros(input_size + 1)
        self.learning_rate = learning_rate

    def activation(self, x):
        return x

    def predict(self, inputs):
        z = np.dot(inputs, self.weights[1:]) + self.weights[0]
        return self.activation(z)

    def fit(self, training_inputs, targets, epochs):
        errors = []
        for epoch in range(epochs):
            total_error = 0
            for inputs, target in zip(training_inputs, targets):
                prediction = self.predict(inputs)
                error = target - prediction
                total_error += error ** 2
                self.weights[1:] += self.learning_rate * error * inputs
                self.weights[0] += self.learning_rate * error
            errors.append(total_error)
            print(f"| {epoch + 1:<5} | {error:<10.6f} | {total_error:<10.6f} |")
```

3. Defining the training data for the AND function

Now, we create a simple dataset for training the ADALINE model. In this example, we use the logical AND function, with input pairs and their corresponding targets.

```
def generate_training_data():
    training_inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
    targets = np.array([0, 0, 0, 1])
    return training_inputs, targets
```

4. Main execution

Finally, we define the main execution block which:

- calls the `generate_training_data` function to create the training dataset,
- initialises the ADALINE model with a specified input size and learning rate,
- calls the `fit` method to train the model and display the errors in a neat tabular format, and,
- prints the final weights after training.

```
if __name__ == "__main__":
    training_inputs, targets = generate_training_data()

    adaline = ADALINE(input_size=2, learning_rate=0.1)

    print("| Epoch | Error      | Sum of Squared Errors |")
    print("|-------|------------|-----------------------|")
```

```
adaline.fit(training_inputs, targets, epochs=10)

print(f"\nFinal weights: {adaline.weights}")
```

```
| Epoch | Error      | Sum of Squared Errors |
|-------|------------|-----------------------|
| 1     | 1.000000   | 1.000000   |
| 2     | 0.782200   | 0.687178   |
| 3     | 0.666821   | 0.603073   |
| 4     | 0.601536   | 0.564335   |
| 5     | 0.561056   | 0.534368   |
| 6     | 0.533135   | 0.507202   |
| 7     | 0.511827   | 0.482411   |
| 8     | 0.494226   | 0.460162   |
| 9     | 0.478897   | 0.440457   |
| 10    | 0.465115   | 0.423149   |

Final weights: [-0.00561267  0.34975884  0.3302731 ]
```