```
# OR gate
import numpy as np

# Define the Adaline class

class AdalineGD(object):
    def __init__(self, learning_rate=0.01, n_iter=50):
      self.learning_rate = learning_rate
      self.n_iter = n_iter

    def fit(self, X, y):
        self.weights_= np.zeros(1 + X.shape[1])
        self.cost_ = []

        for i in range(self.n_iter):
            output= self.net_input(X)
            errors= (y - output)
            self.weights_[1:] += self.learning_rate * X.T.dot(errors)
            self.weights_[0] += self.learning_rate * errors.sum()
            cost= (errors**2).sum() / 2.0
            self.cost_.append(cost)
        return self
    def net_input(self, X):
        return np.dot(X, self.weights_[1:]) + self.weights_[0]

    def activation(self, X):
        return self.net_input(X)

    def predict(self, X):
        return np.where(self.activation(X) >= 0.0, 1, 0)

# Define the OR gate input and outputs

X = np.array([[0, 0],
              [0,  1],
              [1,  0],
              [1, 1]])

y = np.array([0, 1, 1, 1])

# Create the Adaline and train it

adaline = AdalineGD(learning_rate=0.01, n_iter=10)
adaline.fit(X, y)

# Test the Adaline network

print("Predictions:")
for xi in X:
    print(xi, "->", adaline.predict(xi))
```

```
Predictions:
[0 0] -> 1
[0 1] -> 1
[1 0] -> 1
[1 1] -> 1
```

```
# AND gate
import numpy as np

# Define the Adaline class

class AdalineGD(object):
    def __init__(self, learning_rate=0.01, n_iter=50):
      self.learning_rate = learning_rate
      self.n_iter = n_iter

    def fit(self, X, y):
        self.weights_=np.zeros(1+X.shape[1])
        self.cost_ = []

        for i in range(self.n_iter):
            output= self.net_input(X)
            errors= (y - output)
            self.weights_[1:] += self.learning_rate * X.T.dot(errors)
            self.weights_[0] += self.learning_rate * errors.sum()
            cost= (errors**2).sum() / 2.0
            self.cost_.append(cost)
        return self

    def net_input(self, X):
        return np.dot(X, self.weights_[1:]) + self.weights_[0]

    def activation(self, X):
        return self.net_input(X)

    def predict(self, X):
        return np.where(self.activation(X) >= 0.0, 1, 0)

# Define the OR gate input and outputs

X = np.array([[0, 0],
              [0,  1],
              [1,  0],
              [1, 1]])

y = np.array([0, 0, 0,1])

# Create the Adaline and train it

adaline = AdalineGD(learning_rate=0.01, n_iter=10)
adaline.fit(X, y)

# Test the Adaline network print("Predictions:")

for xi in X:
    print(xi, "->", adaline.predict(xi))
```

```
 [0 0] -> 1
 [0 1] -> 1
 [1 0] -> 1
 [1 1] -> 1
```