In [1]:

```
import numpy as np


def sigmoid(x):
    return 1 / (1 + np.exp(-x))


def sigmoid_derivative(x):
    return sigmoid(x) * (1 - sigmoid(x))

def binary_crossentropy(y_true, y_pred):
    return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))

np.random.seed(42)


input_size = 5
hidden_size = 4
output_size = 5

W1 = np.random.randn(input_size, hidden_size)
b1 = np.zeros((1, hidden_size))
W2 = np.random.randn(hidden_size, output_size)
b2 = np.zeros((1, output_size))


learning_rate = 0.01
num_epochs = 1000


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

# Train the autoencoder
for epoch in range(num_epochs):
    # Forward pass
    z1 = np.dot(X, W1) + b1
    a1 = sigmoid(z1)
    z2 = np.dot(a1, W2) + b2
    y = sigmoid(z2)

    # Backward pass
    d_z2 = y - X
    d_W2 = np.dot(a1.T, d_z2)
    d_b2 = np.sum(d_z2, axis=0, keepdims=True)
    d_a1 = np.dot(d_z2, W2.T)
    d_z1 = d_a1 * sigmoid_derivative(z1)
    d_W1 = np.dot(X.T, d_z1)
    d_b1 = np.sum(d_z1, axis=0, keepdims=True)


    W1 -= learning_rate * d_W1
    b1 -= learning_rate * d_b1
    W2 -= learning_rate * d_W2
    b2 -= learning_rate * d_b2


    loss = binary_crossentropy(X,y)


#  epochs
    if epoch % 100 == 0:
        print(f"Epoch {epoch}: loss={loss}")
```

Loading [MathJax]/extensions/Safe.js

```
Epoch 0: loss=0.8181525108876693
Epoch 100: loss=0.501918229289269
Epoch 200: loss=0.35093225364358116
Epoch 300: loss=0.2608098776131166
Epoch 400: loss=0.2009554652120688
Epoch 500: loss=0.15872058193974667
Epoch 600: loss=0.12788063998828095
Epoch 700: loss=0.10490077988254294
Epoch 800: loss=0.08751914296940494
Epoch 900: loss=0.0741827580810672
```

In [ ]: