```python
import numpy as np

# -------- Activation Functions --------

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

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

def tanh(x):
    return np.tanh(x)

def tanh_derivative(x):
    return 1 - np.tanh(x) ** 2

def relu(x):
    return np.maximum(0, x)

def relu_derivative(x):
    return (x > 0).astype(float)

def softmax(x):
    e_x = np.exp(x - np.max(x))  # stability trick
    return e_x / e_x.sum(axis=0)

def swish(x):
    return x * sigmoid(x)

def swish_derivative(x):
    s = sigmoid(x)
    return s + x * s * (1 - s)

# -------- User Input --------

n_inputs = int(input("Enter number of inputs (x1,x2,x3..): "))
X = np.array(list(map(float, input(f"Enter {n_inputs} input values: ").split())))

n_layers = int(input("Enter number of layers (hidden + output): "))

layer_sizes = []
for i in range(n_layers):
    layer_sizes.append(int(input(f"Enter number of neurons in layer {i+1}: ")))

weights = []
biases = []
prev_size = n_inputs

for i, size in enumerate(layer_sizes):
    print(f"\nEnter weights for layer {i+1} ({size} x {prev_size}):")
    W = np.array([list(map(float, input().split())) for _ in range(size)])

    b = np.array(list(map(float, input(f"Enter {size} bias values for layer {i+1}: ").split())))

    weights.append(W)
    biases.append(b)
    prev_size = size

target = float(input("\nEnter target output: "))
lr = float(input("Enter learning rate: "))

activation_choice = input("Select activation function (sigmoid/tanh/relu/swish/softmax): ").lower()

if activation_choice == "sigmoid":
    activation, activation_derivative = sigmoid, sigmoid_derivative
elif activation_choice == "tanh":
    activation, activation_derivative = tanh, tanh_derivative
elif activation_choice == "relu":
    activation, activation_derivative = relu, relu_derivative
elif activation_choice == "swish":
    activation, activation_derivative = swish, swish_derivative
elif activation_choice == "softmax":
    activation, activation_derivative = softmax, None
else:
    raise ValueError("Unsupported activation function selected.")

# -------- Training Loop --------

epoch = 1
error = float('inf')
max_epochs = 10000

while abs(error) > 1e-6 and epoch <= max_epochs:

    activations = [X]
    nets = []

    # Forward pass
    for i in range(len(weights)):
        net = np.dot(weights[i], activations[-1]) + biases[i]
        nets.append(net)
        activations.append(activation(net))

    if activation_choice != "softmax":
        output = activations[-1][0]
        error = target - output
    else:
        output = activations[-1]
        error = target - output[0]

    # Backpropagation (skip softmax)
    if activation_choice != "softmax":

        deltas = [error * activation_derivative(nets[-1])]

        for i in range(len(weights) - 2, -1, -1):
            delta = np.dot(weights[i + 1].T, deltas[-1]) * activation_derivative(nets[i])
            deltas.append(delta)

        deltas.reverse()

        # Update weights and biases
        for i in range(len(weights)):
            weights[i] += lr * np.outer(deltas[i], activations[i])
            biases[i] += lr * deltas[i]

    print(f"Epoch {epoch}: Output = {output}, Error = {error}")
    epoch += 1

# -------- Result --------

if epoch > max_epochs:
    print("\nReached maximum epochs without reaching exact target.")
else:
    print("\nTraining complete: Error is effectively zero.")
```