Here is my implementation:

float FDerivative(float x)

{

// dg/dx = dg/dh \* dh/di \* di/dx

float di\_dx = IDerivative(x);

float i = I(x);

float dh\_di = HDerivative(i);

float h = H(i);

float dg\_dh = GDerivative(h);

return dg\_dh \* dh\_di \* di\_dx;

}

For F being a sum of two nested functions, I changed F to be this:

float F(float x)

{

return G(H(I(x))) + G(I(x));

}

And I changed FDerivative to this:

float FDerivative(float x)

{

// The left side

float left = 0.0f;

{

float di\_dx = IDerivative(x);

float i = I(x);

float dh\_di = HDerivative(i);

float h = H(i);

float dg\_dh = GDerivative(h);

left = dg\_dh \* dh\_di \* di\_dx;

}

// The right side

float right = 0.0f;

{

float di\_dx = IDerivative(x);

float i = I(x);

float dg\_di = GDerivative(i);

left = dg\_di \* di\_dx;

}

return left + right;

}

When playing with the functions, did you notice the values ever shooting off to infinity, but the function definitely had a global minimum? This is “exploding gradients” which is a known problem that comes up in deep learning, along with “vanishing gradients” which makes the gradient go to zero (the graph goes flat). This literally comes up in DEEP LEARNING specifically, which means having many layers in your neural network. Dealing with these problems better is how DEEP learning became a thing in modern times. Prior to that, neural networks were a lot more limited and were not as ubiquitous as they are today.

Give this a read for an introduction to some tools for dealing with these problems:

<https://machinelearningmastery.com/exploding-gradients-in-neural-networks/>