

# Automatic Differentiation

 **Watson IoT** ™ © Copyright IBM Corp. 2017 Romeo Kienzler

In this module we will cover Automatic Differentiation - one of the key features of TensorFlow

$$\cos(x)$$

May from high school you remember what differentiation and the chain rule of differentiation is. If not, please have a look the the next two slides for a little refresher.

$$\cos(x)'$$

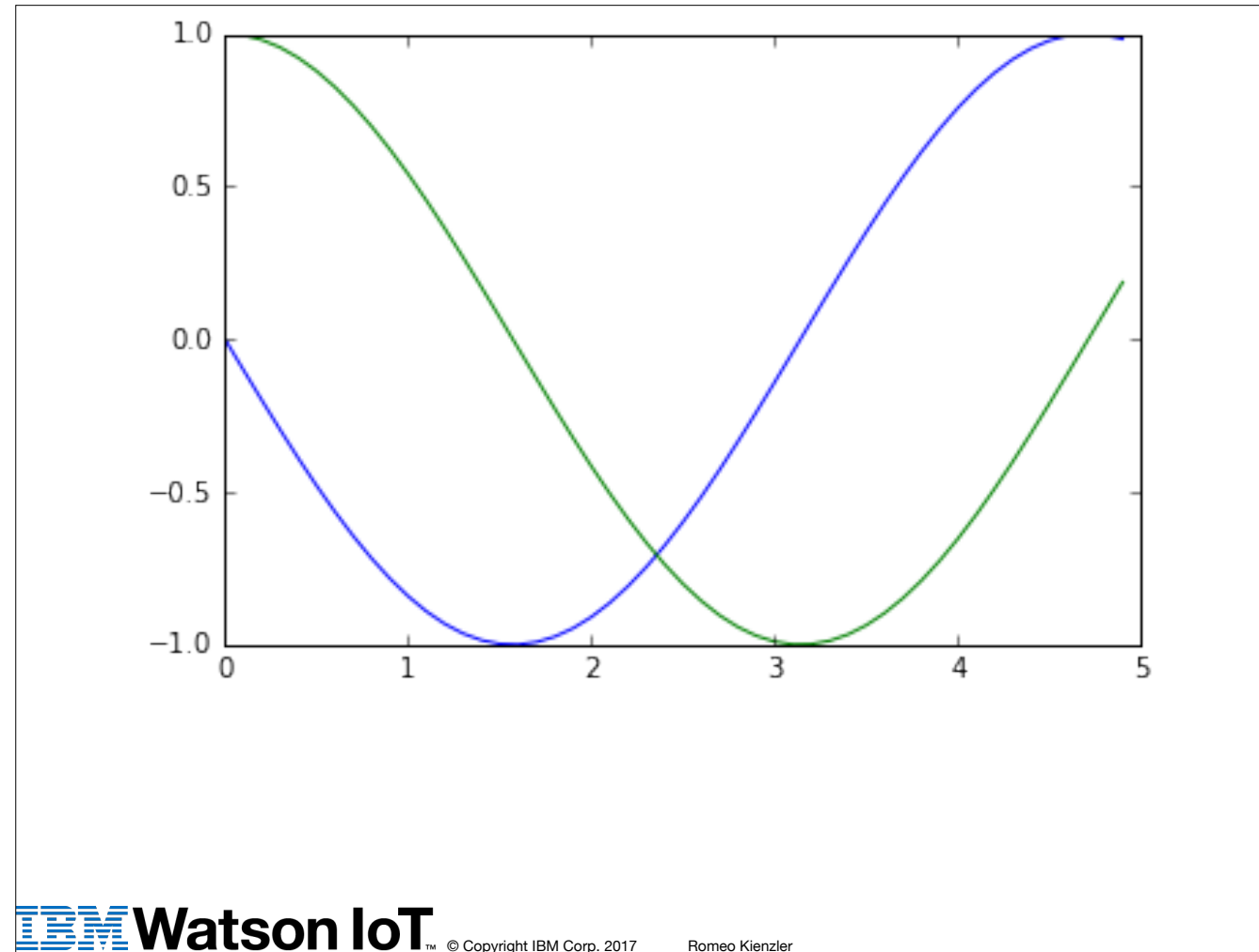
So if we take the first derivative of cosine of x...

$$\cos(x)' = -\sin(x)$$

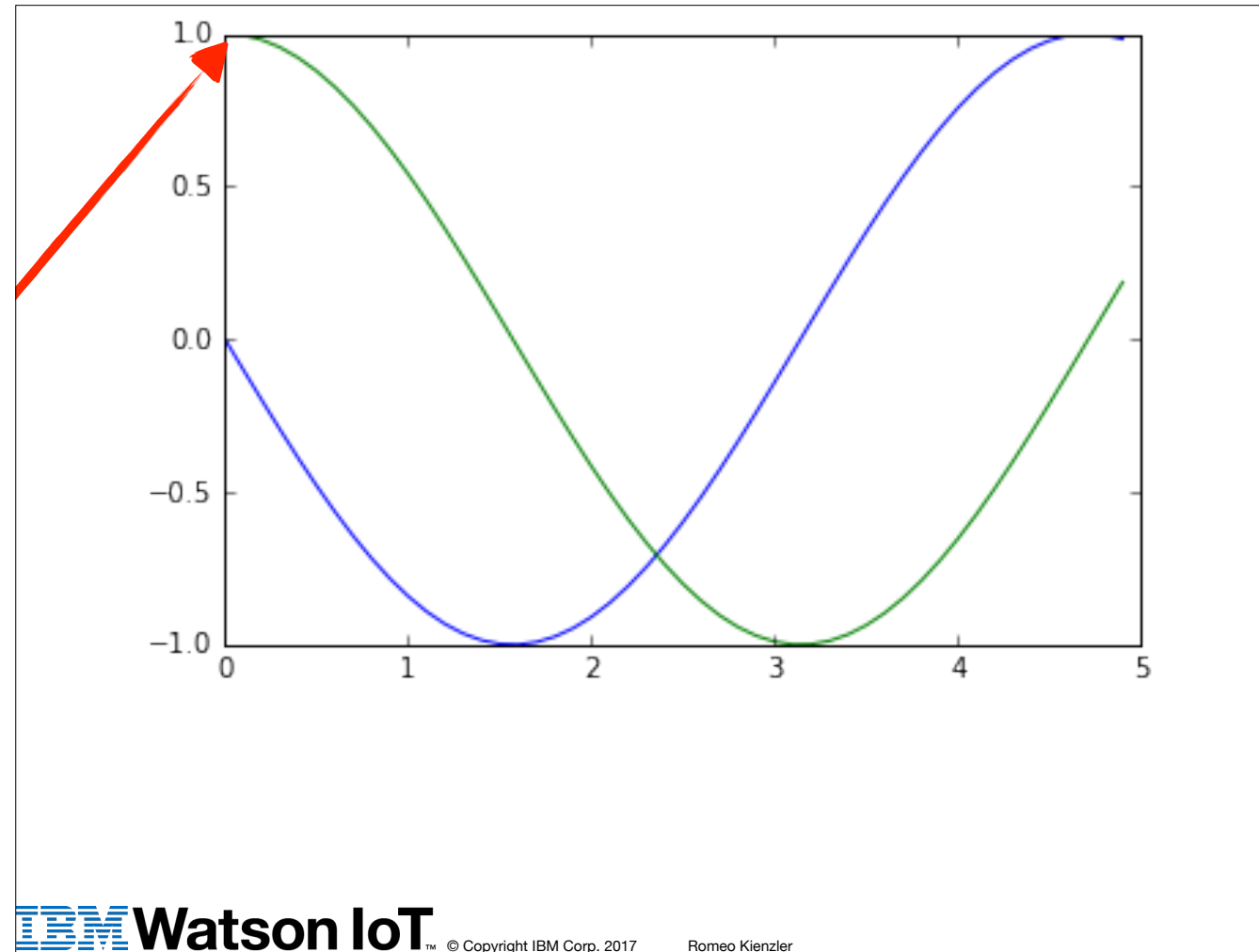
It turns out that this is just minus sine of x.

a3\_m1\_s2\_v3\_autodiff\_v1

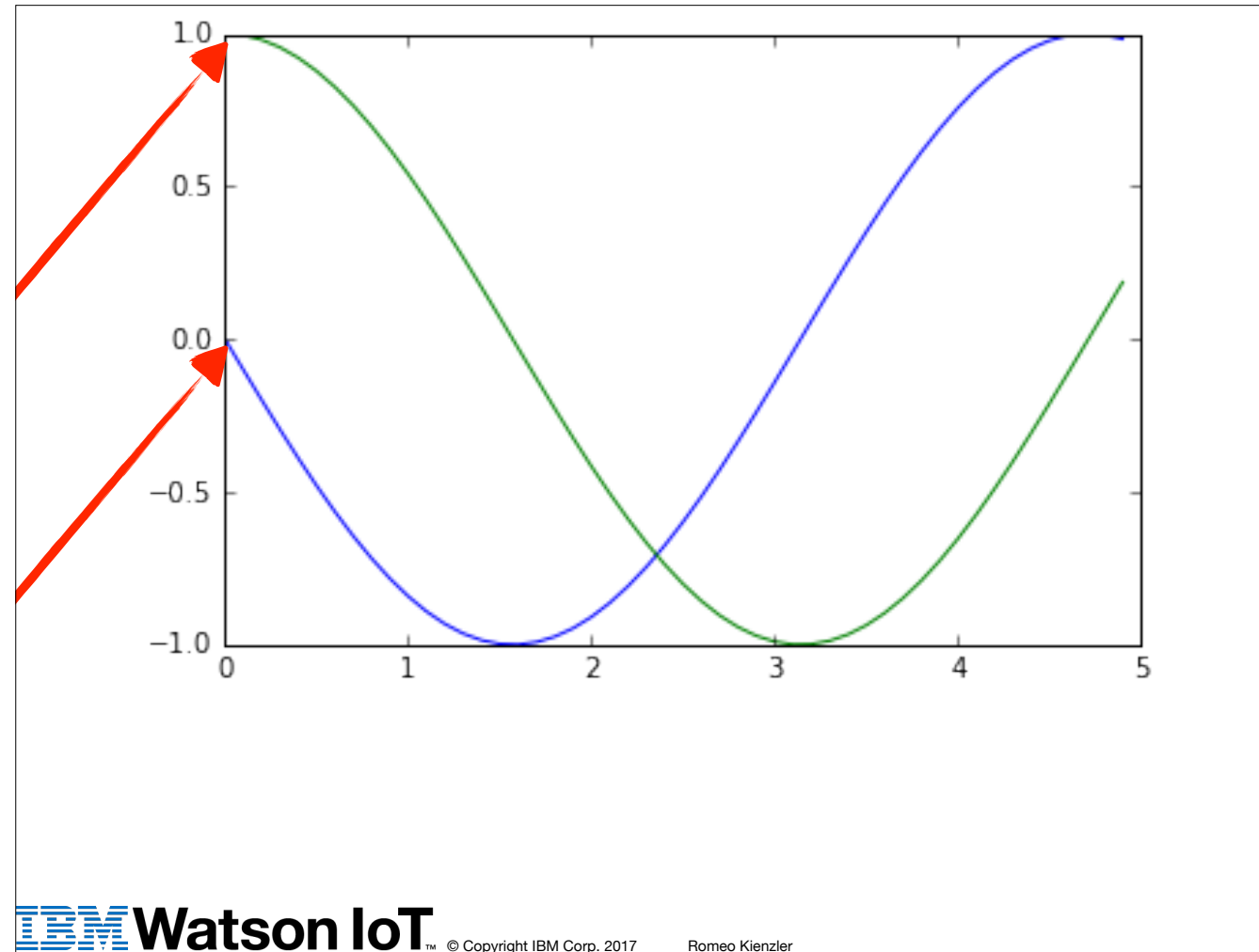
Let's create a little plot to illustrate this...so the green line is cosine of x and the blue line is the first derivative of cosine of x.



As you might remember, the first derivative tells us something about the slope of a function.

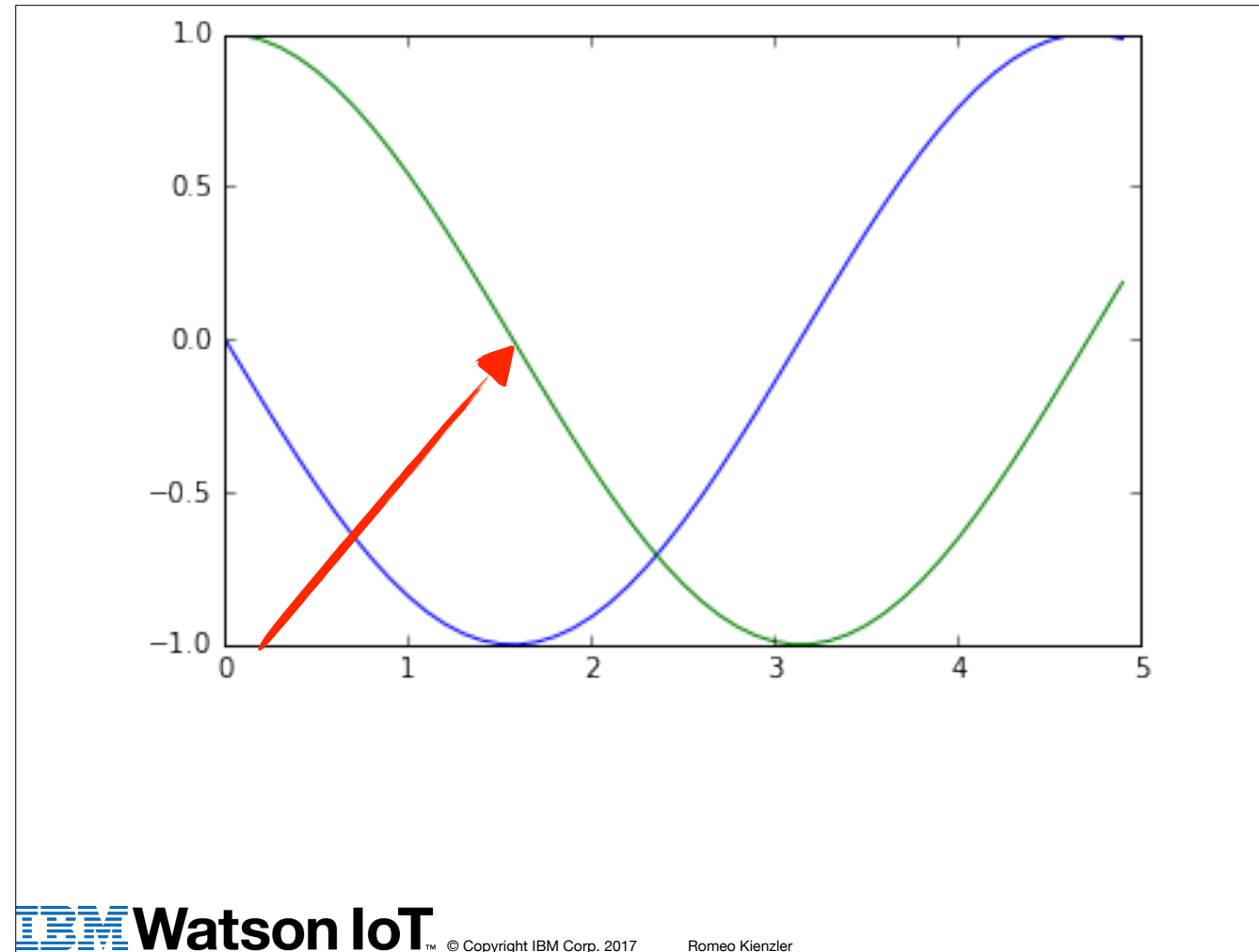


So here at this point where the slope is zero...

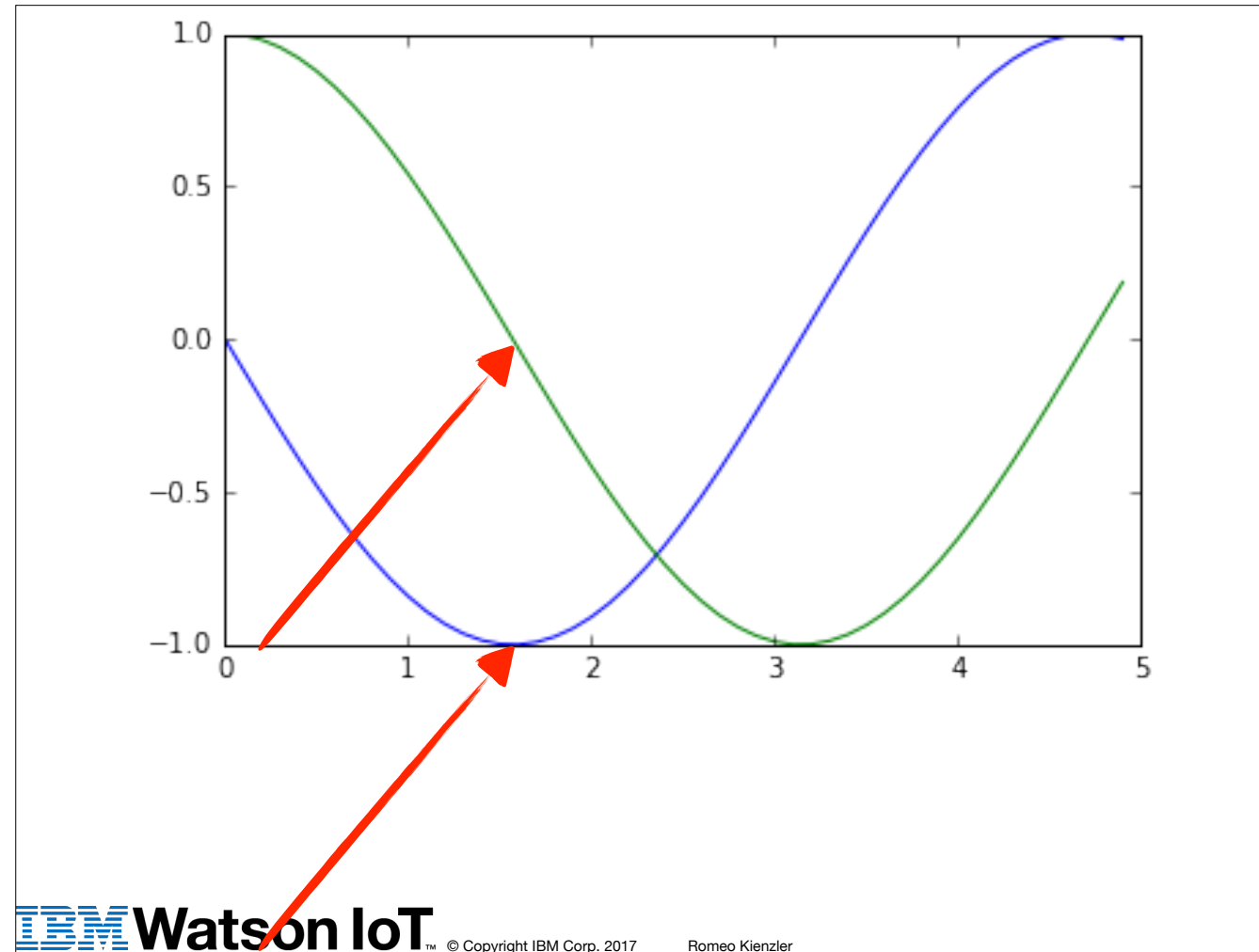


...the value of the first derivative is zero as well.

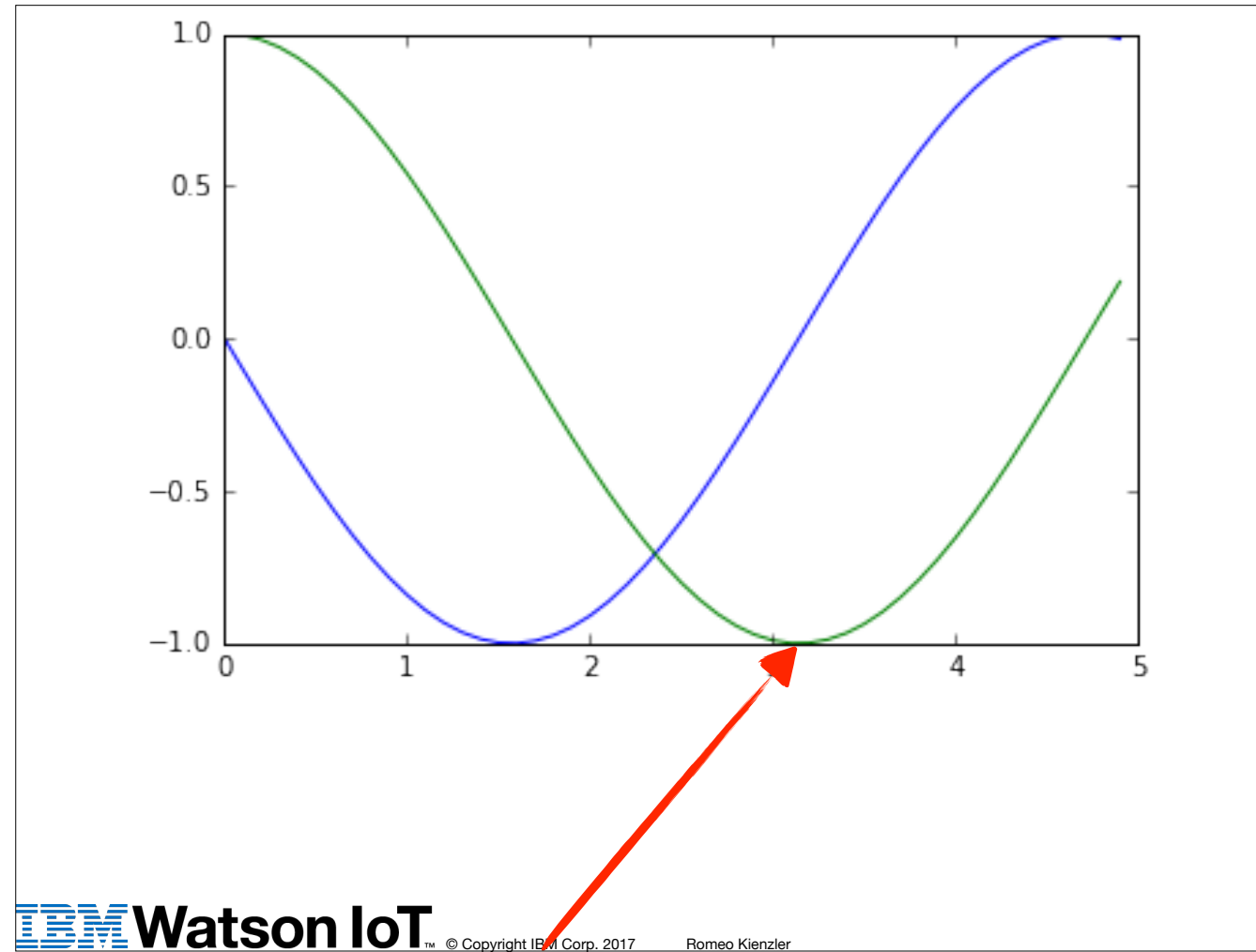




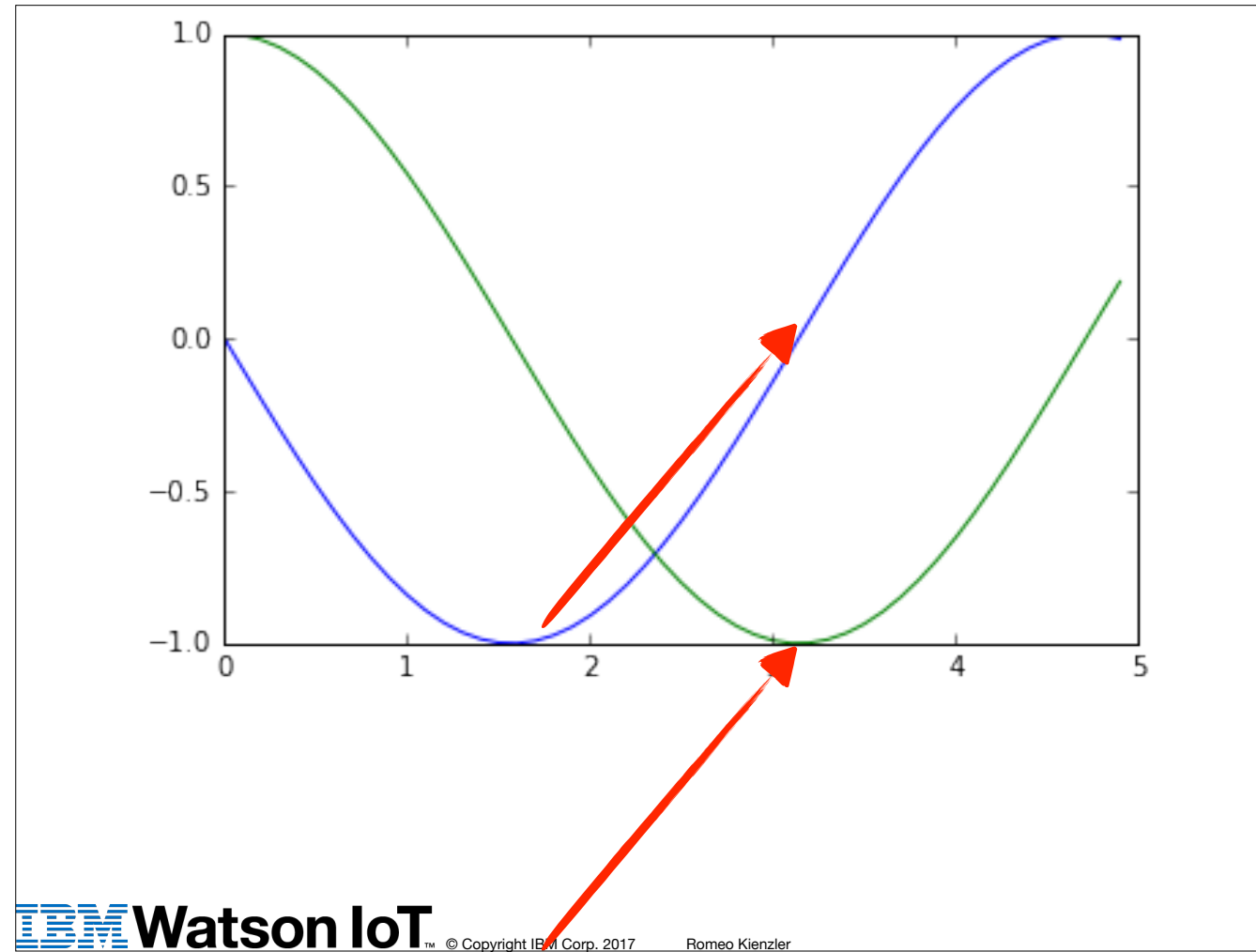
At the steepest position...



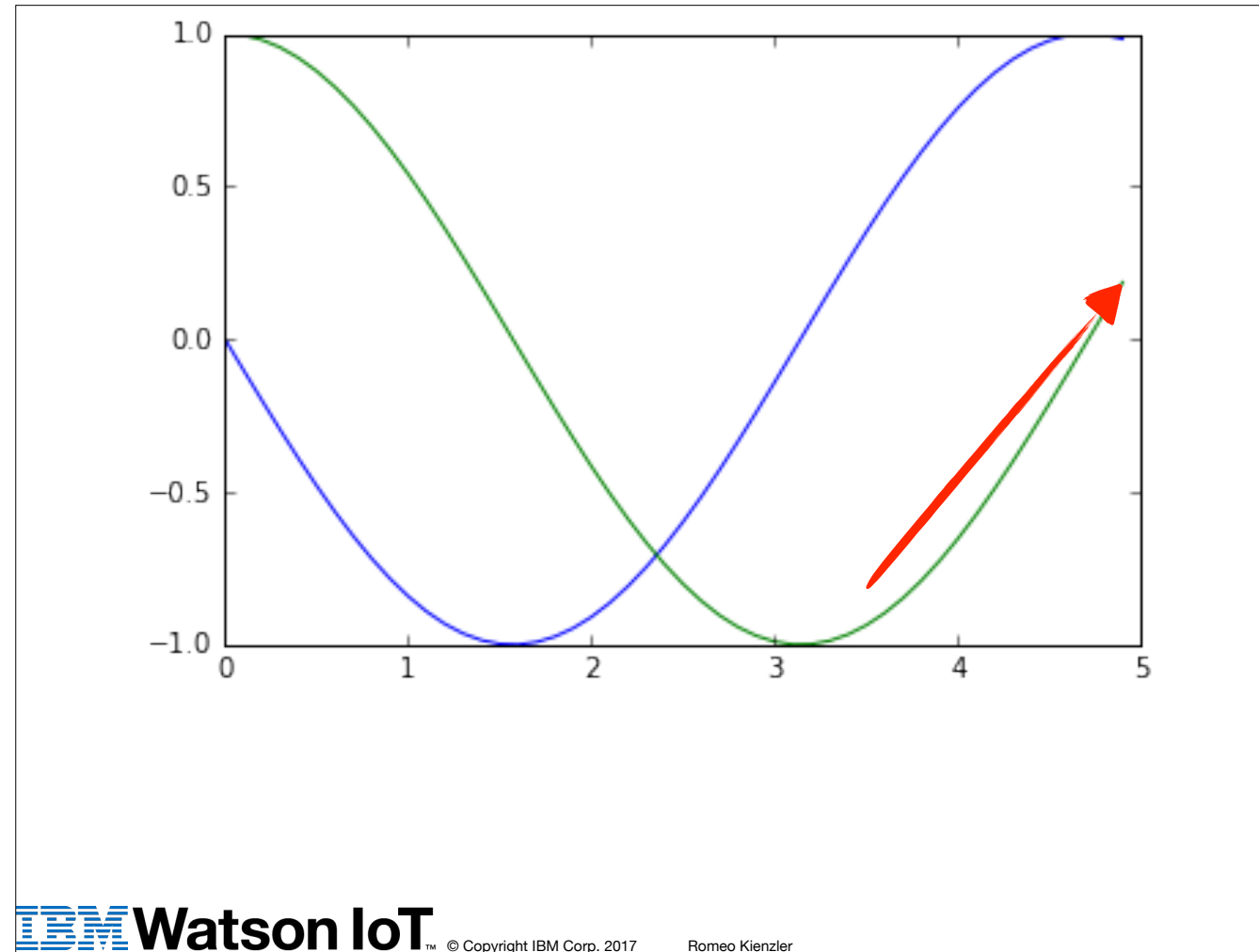
The derivative becomes minus one..



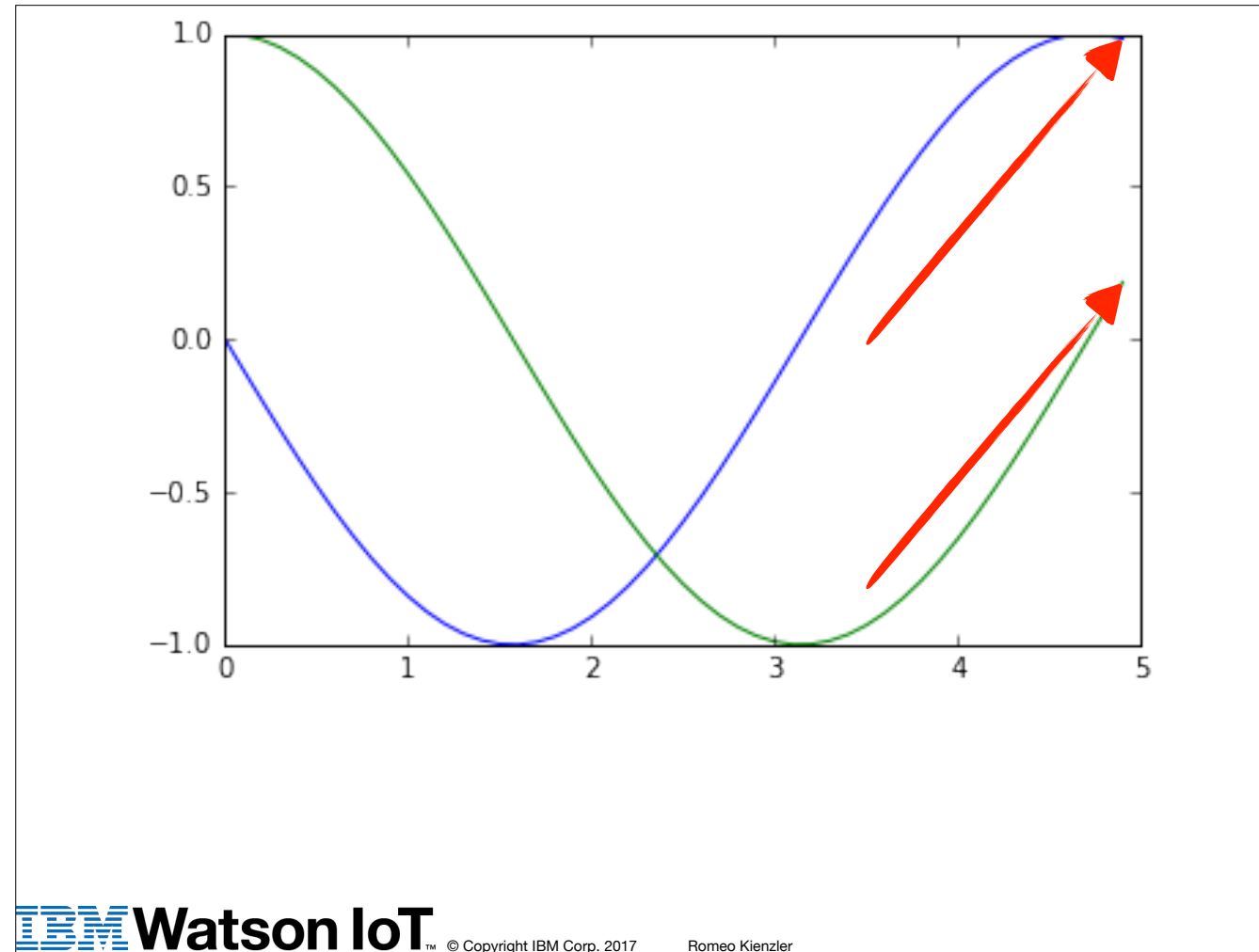
Again, the cosine function reaches zero gradient..



And this is reflected by the first derivative as well...



Finally, when the cosine function reached it's maximum ascending gradient...



...the first derivative becomes one.

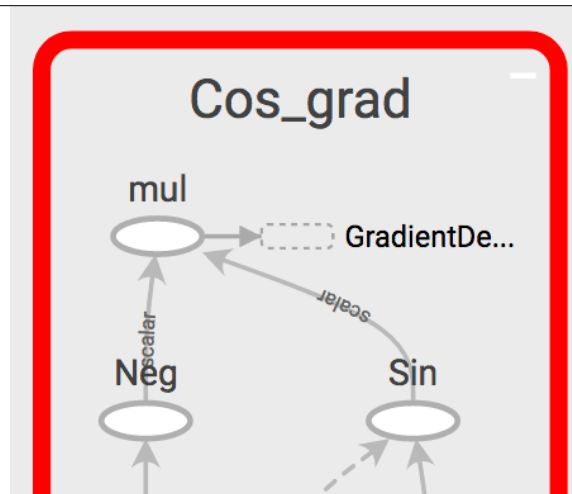
**a3\_m1\_s2\_v3\_autodiff\_v2**

So now let TensorFlow take care of computing the first derivative of the cosine function. We start with a value  $x$  and apply TensorFlow's cosine operation on it. Then we tell TensorFlow to minimise  $w$ . In order to achieve this, TensorFlow must compute the first derivative of the cosine function. Note that until now no computations have happened, only the execution graph has been created. Let's have a look at this graph in TensorBoard now.

**a3\_m1\_s2\_v3\_autodiff\_v2**

So this is the graph of our example which contains a gradient operation. Within this section the gradient of the cosine function is computed which is actually nothing else than the first derivative of the function.





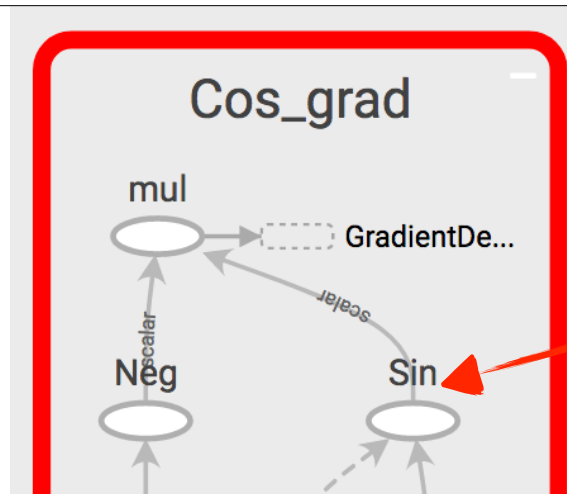
$$\cos(x)' = -\sin(x)$$



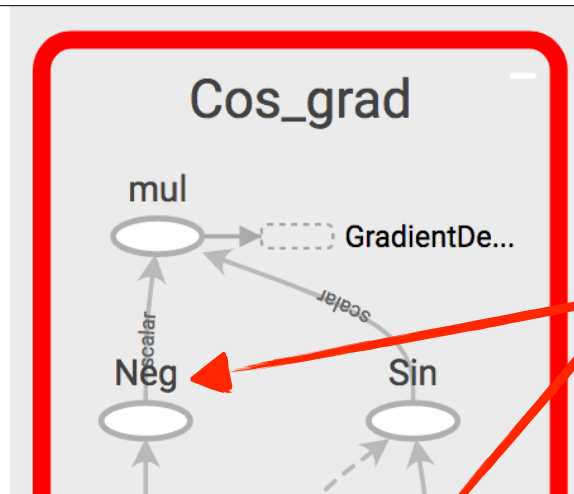
Credits to Salvador Dali for this example

<https://stackoverflow.com/questions/44342432/is-gradient-in-the-tensorflows-graph-calculated-incorrectly>

As you can see this is reflected by the appropriate operations.



$$\cos(x)' = -\sin(x)$$



$$\cos(x)' = -\sin(x)$$



$$f(x)' = g(x)$$

It turns out that for every atomic mathematical operation it is guaranteed that there exists a derivative operation.

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} = f'(y)g'(x) = f'(g(x))g'(x)$$

And with the chain rule of differentiation we are guaranteed to obtain a solution for the first derivative no matter how complex the functions are getting. But this is beyond the scope of this course.

```

from tensorflow.python.framework import ops
from tensorflow.python.ops import array_ops
from tensorflow.python.ops import sparse_ops

@ops.RegisterGradient("ZeroOut")
def _zero_out_grad(op, grad):
    """The gradients for `zero_out`.

    Args:
        op: The `zero_out` `Operation` that we are differentiating, which we can use
            to find the inputs and outputs of the original op.
        grad: Gradient with respect to the output of the `zero_out` op.

    Returns:
        Gradients with respect to the input of `zero_out`.
    """
    to_zero = op.inputs[0]
    shape = array_ops.shape(to_zero)
    index = array_ops.zeros_like(shape)
    first_grad = array_ops.reshape(grad, [-1])[0]
    to_zero_grad = sparse_ops.sparse_to_dense([index], shape, first_grad, 0)
    return [to_zero_grad] # List of one Tensor, since we have one input

```

Credits: [https://www.tensorflow.org/extend/adding\\_an\\_op](https://www.tensorflow.org/extend/adding_an_op)

Just remember, every operator which is available within TensorFlow has to register an appropriate gradient function. Here in this case

```

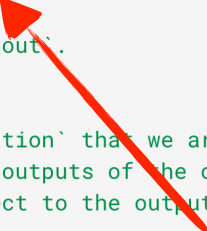
from tensorflow.python.framework import ops
from tensorflow.python.ops import array_ops
from tensorflow.python.ops import sparse_ops

@ops.RegisterGradient("ZeroOut")
def _zero_out_grad(op, grad):
    """The gradients for `zero_out`.

    Args:
        op: The `zero_out` `Operation` that we are differentiating, which we can use
            to find the inputs and outputs of the original op.
        grad: Gradient with respect to the output of the `zero_out` op.

    Returns:
        Gradients with respect to the input of `zero_out`.
    """
    to_zero = op.inputs[0]
    shape = array_ops.shape(to_zero)
    index = array_ops.zeros_like(shape)
    first_grad = array_ops.reshape(grad, [-1])[0]
    to_zero_grad = sparse_ops.sparse_to_dense([index], shape, first_grad, 0)
    return [to_zero_grad] # List of one Tensor, since we have one input

```



Credits: [https://www.tensorflow.org/extend/adding\\_an\\_op](https://www.tensorflow.org/extend/adding_an_op)

An operator called ZeroOut

```

from tensorflow.python.framework import ops
from tensorflow.python.ops import array_ops
from tensorflow.python.ops import sparse_ops

@ops.RegisterGradient("ZeroOut")
def _zero_out_grad(op, grad):
    """The gradients for `zero_out`.

    Args:
        op: The `zero_out` `Operation` that we are differentiating, which we can use
            to find the inputs and outputs of the original op.
        grad: Gradient with respect to the output of the `zero_out` op.

    Returns:
        Gradients with respect to the input of `zero_out`.
    """
    to_zero = op.inputs[0]
    shape = array_ops.shape(to_zero)
    index = array_ops.zeros_like(shape)
    first_grad = array_ops.reshape(grad, [-1])[0]
    to_zero_grad = sparse_ops.sparse_to_dense([index], shape, first_grad, 0)
    return [to_zero_grad] # List of one Tensor, since we have one input

```

Credits: [https://www.tensorflow.org/extend/adding\\_an\\_op](https://www.tensorflow.org/extend/adding_an_op)

is registering a gradient function called



```

from tensorflow.python.framework import ops
from tensorflow.python.ops import array_ops
from tensorflow.python.ops import sparse_ops

@ops.RegisterGradient("ZeroOut")
def _zero_out_grad(op, grad):
    """The gradients for `zero_out`.

    Args:
        op: The `zero_out` `Operation` that we are differentiating, which we can use
            to find the inputs and outputs of the original op.
        grad: Gradient with respect to the output of the `zero_out` op.

    Returns:
        Gradients with respect to the input of `zero_out`.
    """
    to_zero = op.inputs[0]
    shape = array_ops.shape(to_zero)
    index = array_ops.zeros_like(shape)
    first_grad = array_ops.reshape(grad, [-1])[0]
    to_zero_grad = sparse_ops.sparse_to_dense([index], shape, first_grad, 0)
    return [to_zero_grad] # List of one Tensor, since we have one input

```

Credits: [https://www.tensorflow.org/extend/adding\\_an\\_op](https://www.tensorflow.org/extend/adding_an_op)

# Summary

This way TensorFlow is capable of automatically compute the derivative of ANY function defined as a TensorFlow graph. So we only need to come up with a creative idea of a model and TensorFlow does the rest. Isn't that great?