Watson Studio democratizes machine learning and deep learning to accelerate infusion of AI in your business to drive innovation. Watson Studio provides a suite of tools and a collaborative environment for data scientists, developers and domain experts.

COGNITIVE CLASS

Differentiation in PyTorch

Table of Contents

In this lab, you will learn the basics of differentiation.

- Derivatives
- Partial Derivatives

Estimated Time Needed: **25 min**

Preparation

The following are the libraries we are going to use for this lab.

In [2]:
```
# These are the libraries will be useing for this lab.

import torch
import matplotlib.pylab as plt
```

Derivatives

Let us create the tensor `x` and set the parameter `requires_grad` to true because you are going to take the derivative of the tensor.

In [3]:
```
# Create a tensor x

x = torch.tensor(2.0, requires_grad = True)
print("The tensor x: ", x)
```

```
The tensor x:  tensor(2., requires_grad=True)
```

Then let us create a tensor according to the equation $y = x^2$.

In [4]:
```
# Create a tensor y according to y = x^2

y = x ** 2
print("The result of y = x^2: ", y)
```

```
The result of y = x^2:  tensor(4., grad_fn=<PowBackward0>)
```

Then let us take the derivative with respect x at x = 2

In [5]:
```
# Take the derivative. Try to print out the derivative at the value x = 2

y.backward()
print("The dervative at x = 2: ", x.grad)
```

```
The dervative at x = 2:  tensor(4.)
```

The preceding lines perform the following operation:

$$\frac{\mathrm{dy(x)}}{\mathrm{dx}} = 2x$$

$$\frac{\mathrm{dy(x=2)}}{\mathrm{dx}} = 2(2) = 4$$

In [ ]:

In [6]:
```
print('data:',x.data)
print('grad_fn:',x.grad_fn)
print('grad:',x.grad)
print("is_leaf:",x.is_leaf)
print("requires_grad:",x.requires_grad)
```

```
data: tensor(2.)
grad_fn: None
grad: tensor(4.)
is_leaf: True
requires_grad: True
```

In [7]:
```
print('data:',y.data)
print('grad_fn:',y.grad_fn)
print('grad:',y.grad)
print("is_leaf:",y.is_leaf)
print("requires_grad:",y.requires_grad)
```

```
data: tensor(4.)
grad_fn: <PowBackward0 object at 0x7f0e12dc31d0>
grad: None
is_leaf: False
requires_grad: True
```

Let us try to calculate the derivative for a more complicated function.

In [8]:
```
# Calculate the y = x^2 + 2x + 1, then find the derivative

x = torch.tensor(2.0, requires_grad = True)
y = x ** 2 + 2 * x + 1
print("The result of y = x^2 + 2x + 1: ", y)
y.backward()
print("The dervative at x = 2: ", x.grad)
```

```
The result of y = x^2 + 2x + 1:  tensor(9., grad_fn=<AddBackward0>)
The dervative at x = 2:  tensor(6.)
```

The function is in the following form: $y = x^2 + 2x + 1$

The derivative is given by:

$$\frac{\mathrm{dy(x)}}{\mathrm{dx}} = 2x + 2$$

$$\frac{\mathrm{dy(x=2)}}{\mathrm{dx}} = 2(2) + 2 = 6$$

Practice

Determine the derivative of $y = 2x^3 + x$ at $x = 1$

In [9]:
```
# Practice: Calculate the derivative of y = 2x^3 + x at x = 1

x = torch.tensor(1.0, requires_grad=True)
y = 2*x**3+x
print('The result of y=2*x^3+x: ', y)
y.backward()
print('The derivative at x=1: ', x.grad)
# Type your code here
```

```
The result of y=2*x^3+x:  tensor(3., grad_fn=<AddBackward0>)
The derivative at x=1:  tensor(7.)
```

Double-click **here** for the solution.

We can implement our own custom autograd Functions by subclassing torch.autograd.Function and implementing the forward and backward passes which operate on Tensors

In [10]:
```
class SQ(torch.autograd.Function):


    @staticmethod
    def forward(ctx,i):
        """
        In the forward pass we receive a Tensor containing the input and return
        a Tensor containing the output. ctx is a context object that can be used
        to stash information for backward computation. You can cache arbitrary
        objects for use in the backward pass using the ctx.save_for_backward method.
        """
        result=i**2
        ctx.save_for_backward(i)
        return result

    @staticmethod
    def backward(ctx, grad_output):
        """
        In the backward pass we receive a Tensor containing the gradient of the loss
        with respect to the output, and we need to compute the gradient of the loss
        with respect to the input.
        """
        i, = ctx.saved_tensors
        grad_output = 2*i
        return grad_output
```

We can apply it the function

In [11]:
```
x=torch.tensor(2.0,requires_grad=True )
sq=SQ.apply

y=sq(x)
y
print(y.grad_fn)
y.backward()
x.grad
```

```
<torch.autograd.function.SQBackward object at 0x7f0ea50bd668>
```

Out[11]:
```
tensor(4.)
```

Partial Derivatives

We can also calculate **Partial Derivatives**. Consider the function: $f(u,v) = vu + u^2$

Let us create `u` tensor, `v` tensor and `f` tensor

In [12]:
```
# Calculate f(u, v) = v * u + u^2 at u = 1, v = 2

u = torch.tensor(1.0,requires_grad=True)
v = torch.tensor(2.0,requires_grad=True)
f = u * v + u ** 2
print("The result of v * u + u^2: ", f)
```

```
The result of v * u + u^2:  tensor(3., grad_fn=<AddBackward0>)
```

This is equivalent to the following:

$$f(u = 1, v = 2) = (2)(1) + 1^2 = 3$$

Now let us take the derivative with respect to `u`:

In [13]:
```
# Calculate the derivative with respect to u

f.backward()
print("The partial derivative with respect to u: ", u.grad)
```

```
The partial derivative with respect to u:  tensor(4.)
```

the expression is given by:

$$\frac{\partial \mathrm{f(u,v)}}{\partial u} = v + 2u$$

$$\frac{\partial \mathrm{f(u=1,v=2)}}{\partial u} = 2 + 2(1) = 4$$

Now, take the derivative with respect to `v`:

In [14]:
```
# Calculate the derivative with respect to v

print("The partial derivative with respect to u: ", v.grad)
```

```
The partial derivative with respect to u:  tensor(1.)
```

The equation is given by:

$$\frac{\partial \mathrm{f(u,v)}}{\partial v} = u$$

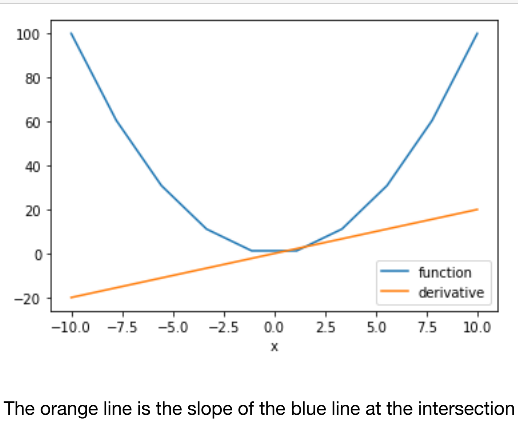
$$\frac{\partial \mathrm{f(u=1,v=2)}}{\partial v} = 1$$

Calculate the derivative with respect to a function with multiple values as follows. You use the sum trick to produce a scalar valued function and then take the gradient:

In [15]:
```
# Calculate the derivative with multiple values

x = torch.linspace(-10, 10, 10, requires_grad = True)
Y = x ** 2
y = torch.sum(x ** 2)
```

We can plot the function and its derivative

In [16]:
```
# Take the derivative with respect to multiple value. Plot out the function and its derivative

y.backward()

plt.plot(x.detach().numpy(), Y.detach().numpy(), label = 'function')
plt.plot(x.detach().numpy(), x.grad.detach().numpy(), label = 'derivative')
plt.xlabel('x')
plt.legend()
plt.show()
```



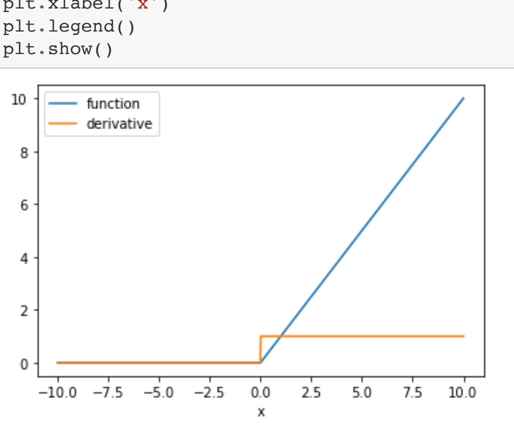
The orange line is the slope of the blue line at the intersection point, which is the derivative of the blue line.

The method `detach()` excludes further tracking of operations in the graph, and therefore the subgraph will not record operations. This allows us to then convert the tensor to a numpy array. To understand the sum operation Click Here

The **relu** activation function is an essential function in neural networks. We can take the derivative as follows:

In [ ]:

In [18]:
```
# Take the derivative of Relu with respect to multiple value. Plot out the function and its derivative

x = torch.linspace(-10, 10, 1000, requires_grad = True)
Y = torch.relu(x)
y = Y.sum()
y.backward()
plt.plot(x.detach().numpy(), Y.detach().numpy(), label = 'function')
plt.plot(x.detach().numpy(), x.grad.detach().numpy(), label = 'derivative')
plt.xlabel('x')
plt.legend()
plt.show()
```



In [19]:
```
y.grad_fn
```

Out[19]:
```
<SumBackward0 at 0x7f0e12dc52b0>
```

Practice

Try to determine partial derivative u of the following function where $u = 2$ and $v = 1$: $f = uv + (uv)^2$

In [8]:
```
# Practice: Calculate the derivative of f = u * v + (u * v) ** 2 at u = 2, v = 1
u = torch.tensor(2., requires_grad=True)
v = torch.tensor(1., requires_grad=True)
f = u*v + (u*v)**2
f.backward()
print("The partial derivative with respect to u: ", u.grad)
print("The partial derivative with respect to v: ", v.grad)
# Type the code here
```

```
The partial derivative with respect to u:  tensor(5.)
The partial derivative with respect to v:  tensor(10.)
```

Double-click **here** for the solution.

About the Authors:

Joseph Santarcangelo has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Other contributors: Michelle Carey, Mavis Zhou
