Pytorch Gradients (Derivatives)

In [34]:

```
# libraries
import torch
import numpy as np
import pandas as pd
import matplotlib.pylab as plt
import warnings
plt.style.use('seaborn-whitegrid')

with warnings.catch_warnings():
    warnings.simplefilter("ignore")
```

In [2]:

```
# get epoch
import datetime
! ../../epochgen.py ef
datetime.datetime.now()
```

```
/bin/bash: ../../epochgen.py: No such file or directory
```

Out[2]:

```
datetime.datetime(2020, 11, 26, 23, 4, 43, 530932)
```

Table of Contents

In [3]:

```
x = torch.tensor(2.0,requires_grad = True)
```

In [4]:

```
x
```

Out[4]:

```
tensor(2., requires_grad=True)
```

In [5]:

```
x.dtype
```

Out[5]:

```
torch.float32
```

In [6]:

```
# function
# y = x^2
y = x ** 2
print('result of the function y : ',y.item())
```

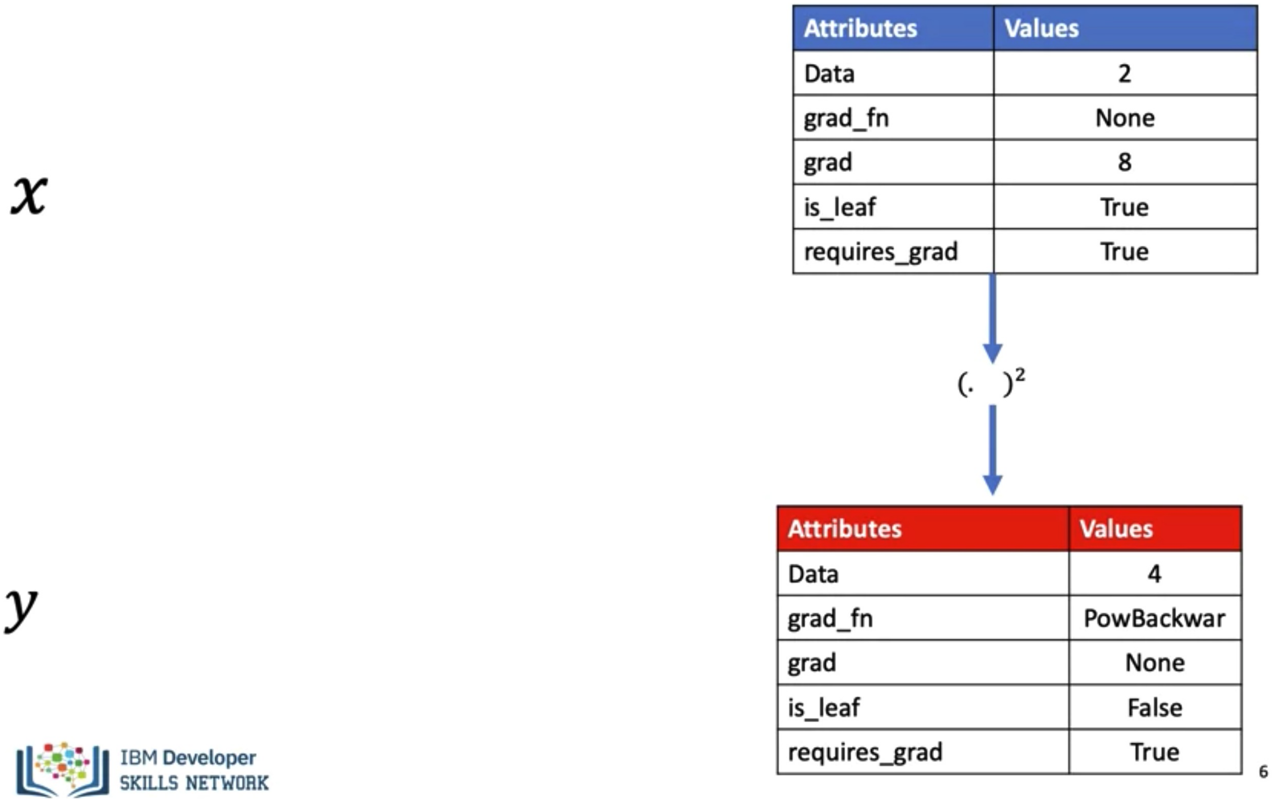
```
result of the function y :  4.0
```

In [7]:

```
# derivative
y.backward() # this calculates the derivative, the function is internally changed to 2x
print("the derivative at x = 2 ->", x.grad)
```

```
the derivative at x = 2 -> tensor(4.)
```

- $fracmathrmdy(x) mathrmdx = 2x$
- $fracmathrmdy(x=2) mathrmdx = 2(2) = 4$



In [8]:

```
# X
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 [9]:

```
# Y
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 0x7f8c836ac610>
grad: None
is_leaf: False
requires_grad: True
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:4: UserWarning: The .gra
```

```
d attribute of a Tensor that is not a leaf Tensor is being accessed. Its .grad attribute
won't be populated during autograd.backward(). If you indeed want the gradient for a non-
leaf Tensor, use .retain_grad() on the non-leaf Tensor. If you access the non-leaf Tensor
by mistake, make sure you access the leaf Tensor instead. See github.com/pytorch/pytorch/
pull/30531 for more informations.
  after removing the cwd from sys.path.
```

In [10]:

```python
# more complication

x = torch.tensor(2.0, requires_grad = True)
y = x ** 2 + 2 * x + 1
```

In [11]:

```python
# y when x is substituted
print('result of y equation ~ ', y)
```

```
result of y equation ~  tensor(9., grad_fn=<AddBackward0>)
```

In [12]:

```python
y.backward() # derivative
```

In [13]:

```python
x.grad # x when substituted in the derivated y function
```

Out[13]:

```
tensor(6.)
```

- **The function is in the following form:** $y = x^2 + 2x + 1$
- $fracmathrmdy(x)mathrmdx = 2x + 2$
- $fracmathrmdy(x mathrmdx = 2(2) + 2 = 6 = 2)$

In [14]:

```python
# one more example
x = torch.tensor(1.0,requires_grad=True)
# function
y = 2*x**3 + x
```

In [15]:

```python
print('x in y : ',y)
```

```
x in y :  tensor(3., grad_fn=<AddBackward0>)
```

In [16]:

```python
# derivative
y.backward()
y
```

Out[16]:

```
tensor(3., grad_fn=<AddBackward0>)
```

In [17]:

```python
x.grad
```

Out[17]:

```
tensor(7.)
```

In [18]:

```
# after derivative
y = 6*x**2 + 1
print(' X in derivative of y -> ', y)
```

```
 X in derivative of y ->  tensor(7., grad_fn=<AddBackward0>)
```

Partial Derivatives

In [19]:

```
# two tensors
u = torch.tensor(1.0,requires_grad=True)
v = torch.tensor(2.0,requires_grad=True)
# fucntion
f = u * v + u **2
print('Result of the function : ', f)
```

```
Result of the function :  tensor(3., grad_fn=<AddBackward0>)
```

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

In [20]:

```
# partial 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.)
```

X was Scalar for most of the above examples, what is the case where x is a vector (which is almost every case in a typical deeplearning problem)

In [21]:

```
# Calculate the derivative with multiple values

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

In [22]:

```
x
```

Out[22]:

```
tensor([-10.0000,  -7.7778,  -5.5556,  -3.3333,  -1.1111,   1.1111,   3.3333,
          5.5556,   7.7778,  10.0000], requires_grad=True)
```

In [23]:

```
print('The value of function y : ',Y)
print('The value of function y.sum() : ',y)
```

```
The value of function y :  tensor([100.0000,  60.4938,  30.8642,  11.1111,   1.2346,   1.
2346,  11.1111,
         30.8642,  60.4938, 100.0000], grad_fn=<PowBackward0>)
The value of function y.sum() :  tensor(407.4074, grad_fn=<SumBackward0>)
```

In [24]:

```
# derivative of y
y.backward()
```

In [25]:

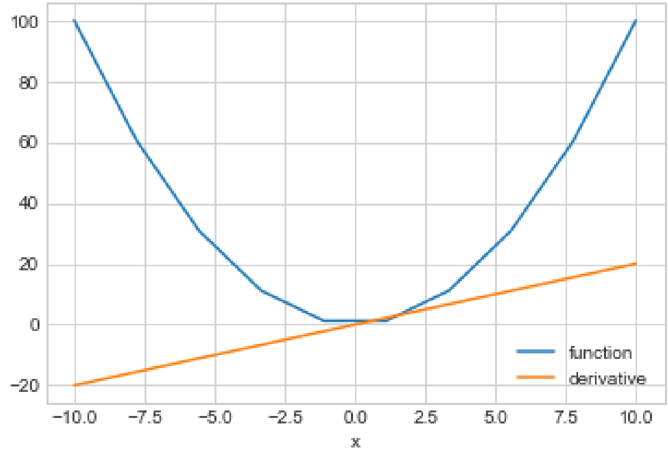
```
x.grad
```

Out[25]:

```
tensor([-20.0000, -15.5556, -11.1111,  -6.6667,  -2.2222,   2.2222,   6.6667,
         11.1111,  15.5556,  20.0000])
```

In [26]:

```
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]**

Gradient of ReLu

In [27]:

```
x = torch.linspace(-10, 10, 1000, requires_grad = True)
Y = torch.relu(x)
```

In [28]:

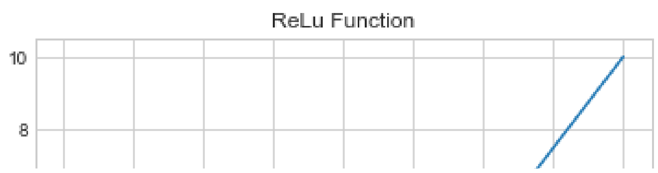
```
# plt.plot(x.numpy(), Y.numpy(), label = 'function')
# this explains why detach is used, initially X vector is bound with the argument
# requires_grad = True, it means torch tracks the history of changes made
# and y is a fucntion of x, which in whole means it cannot be simply converted to a numpy
array
# in which all of this doesn't exist, so we 'detach' the functionality.
```

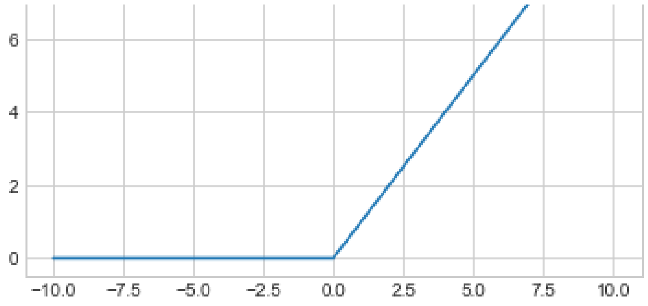
In [29]:

```
plt.title('ReLu Function')
plt.plot(x.detach().numpy(), Y.detach().numpy(), label = 'function')
```

Out[29]:

```
[<matplotlib.lines.Line2D at 0x7f8c84036350>]
```





In [30]:

```
# gradient of Relu
y = Y.sum()
y.backward()
```

In [31]:

```
x.grad
```

Out[31]:

```
tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
```

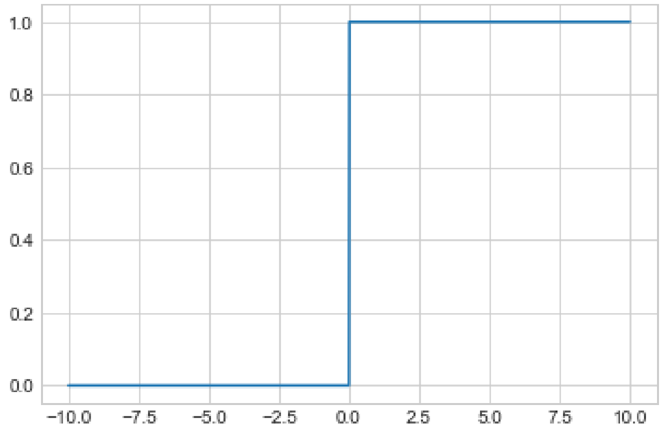
```
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

In [32]:

```
plt.plot(x.detach().numpy(), x.grad.detach().numpy(), label = 'derivative')
```

Out[32]:

```
[<matplotlib.lines.Line2D at 0x7f8c840de8d0>]
```



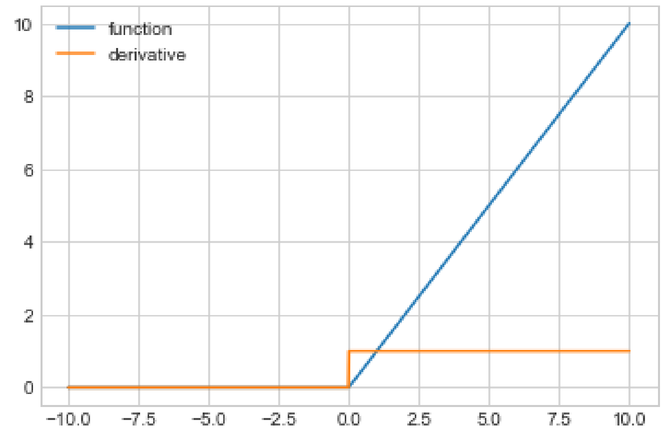
Contrast Between Normal ReLu and Gradient

In [33]:

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

Out[33]:

```
<matplotlib.legend.Legend at 0x7f8c841c8c90>
```



In [ ]: