NOTE

Follow along the videos, and links given in the assignment. If you have **any doubt** related to assignment, **contact your mentor.**

What is Tensorflow?

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

video - **why tensorflow ?**- https://www.youtube.com/watch?v=yjprpOoH5c8

Must Read These Tensorflow Use Cases

1. **How Airbnb uses tensorflow to improve their guests experiences?**

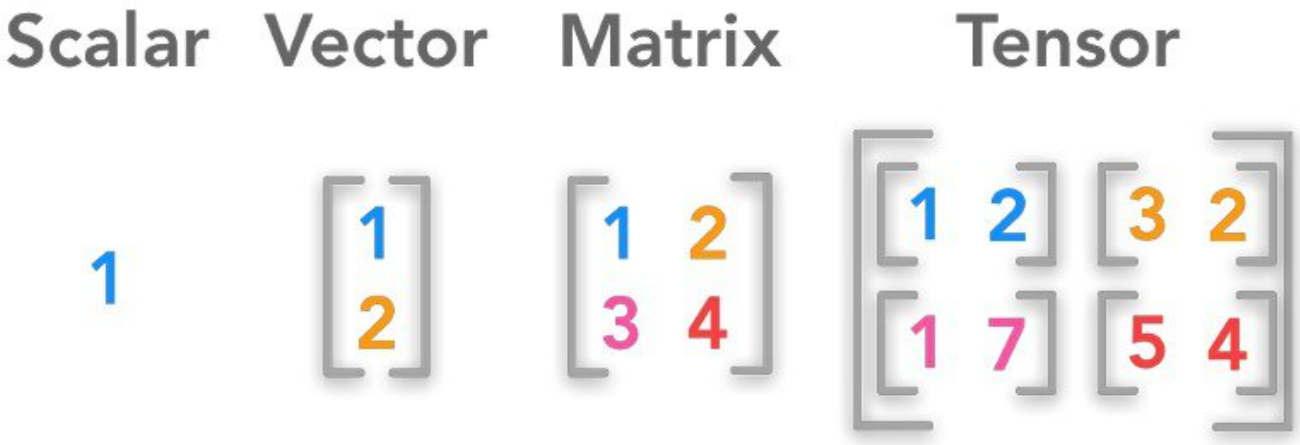
Read this medium article to understand - https://medium.com/airbnb-engineering/categorizing-listing-photos-at-airbnb-f9483f3ab7e3

2. **How paypal uses tensorflow for fraud detection?**

Read this to understand what paypal does - https://medium.com/paypal-tech/machine-learning-model-ci-cd-and-shadow-platform-8c4f44998c78

What is a Tensor?

A tensor is a container for data—usually numerical data. tensors are also called generalization of matrices to an arbitrary number of dimensions.



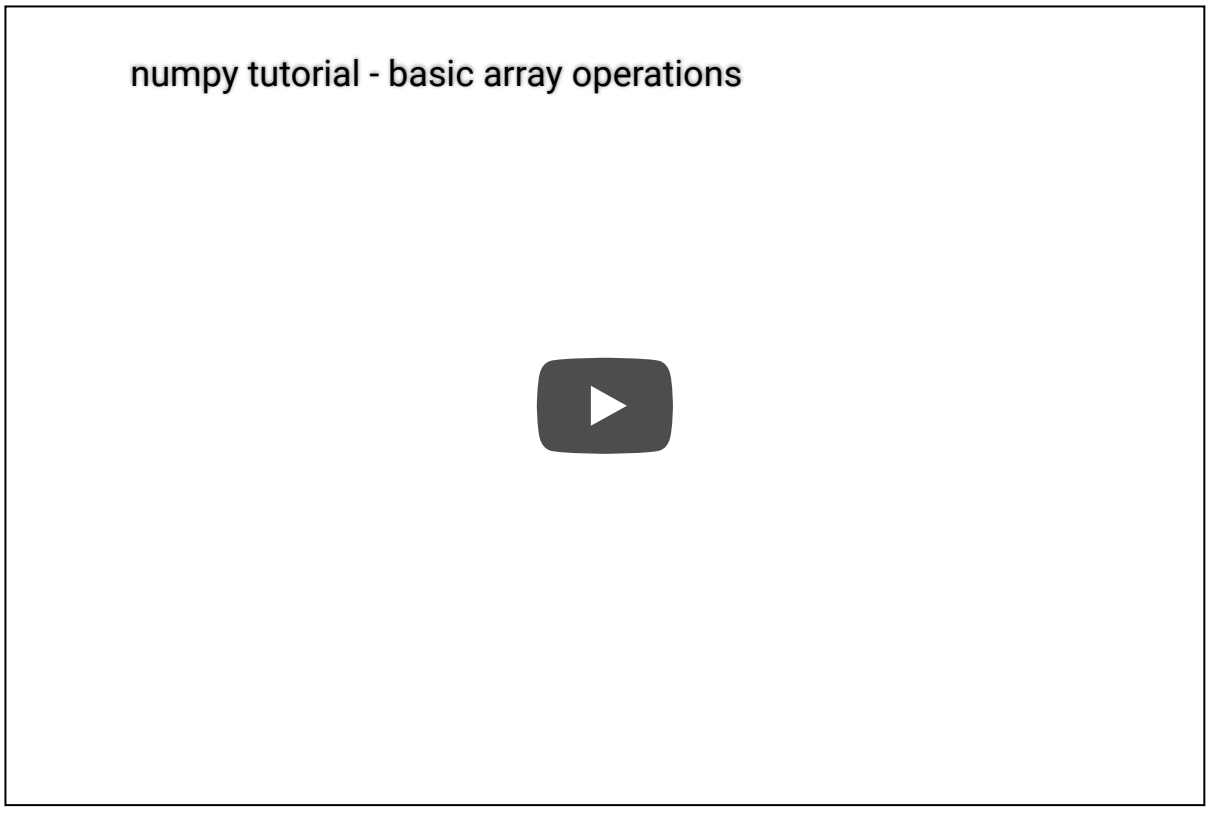
Types of tensor with different rank

1.Scalars (rank 0 tensor)

- A tensor that contains only one number is called a scalar.
- a scalar tensor has 0 axes (ndim == 0).

Go through this video for numpy array methods used in next cell

```
from IPython.display import YouTubeVideo

YouTubeVideo('a8aDcLk4vRc', width=600, height=400)
```



```
# importing numpy as np


# defining an array using np.array with value passing as 5

zero_rank_tensor = # code here

# print zero_rank_tensor


# print its dimension using .ndim method


# print its shape using .shape method
```

```
5
0
()
```

The above output of a scalar number shows that and array with a single digit is having zero rank as a tensor.

Observation from previous output

- Dimension is 0.
- Shape gives empty parenthesis bracket.

2.Vectors (rank 1 tensor)

- An array of numbers is called a vector, or rank-1 tensor, or 1D tensor.
- A rank-1 tensor is said to have exactly one axis.

```
# define an array with value 1,2,3 in a list using np.array

one_rank_tensor = # code here

# print one_rank_tensor


# print its dimension using .ndim


# print its shape using .shape
```

```
[1 2 3]
1
(3,)
```

The above output shows that whenever there is a single square bracket we see around some numbers separated by comma, we get a tensor of rank 1.

Observation

- As compared to previous output, this time dimension is 1.
- Its shape is (3,) showing no of parameters in the array which is 3.

3. Matrices (rank 2 tensor)

- An array of vectors is a matrix, or rank-2 tensor, or 2D tensor.
- A matrix has two axes (often referred to as rows and columns).

```
# define a matrix having values [[1, 2, 3, 4, 5],[6, 7, 8, 9, 10],[11, 12, 13, 14,

rank_2_tensor = # code here

# print rank_2_tensor


# print its dimension using .ndim


# print its shape using .shape
```

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]]
2
(3, 5)
```

The above output shows that whenever there is a double square bracket we see around some numbers separated by comma, we get a tensor of rank 2.

Observation

- This time we got dimension as 2 since it's a matrix.
- We got shape as (3,5) where 3 is no of rows and 5 points to no of columns.

4. Cube (rank 3 tensors)

- If you pack 2-d matrices in a new array, you obtain a rank-3 tensor (or 3D tensor).
- By packing rank-3 tensors in an array, you can create a rank-4 tensor, and so on.

```
# define an array of 3 matrices whose matrices are  [ [5, 78, 2, 34, 0],[6, 79, 3,
```

```
# [ [5, 78, 2, 34, 0],[6, 79, 3, 35, 1],[7, 80, 4, 36, 2] ],[ [5, 78, 2, 34, 0],[6

rank_4_tensor = # code here

# print rank_4_tensor


# print its dimension using .ndim


# print its shape using .shape
```

```
[[[ 5 78  2 34  0]
  [ 6 79  3 35  1]
  [ 7 80  4 36  2]]

 [[ 5 78  2 34  0]
  [ 6 79  3 35  1]
  [ 7 80  4 36  2]]

 [[ 5 78  2 34  0]
  [ 6 79  3 35  1]
  [ 7 80  4 36  2]]]
3
(3, 3, 5)
```

The above output shows that whenever there is a triple square bracket we see around some numbers separated by comma, we get a tensor of rank 3.

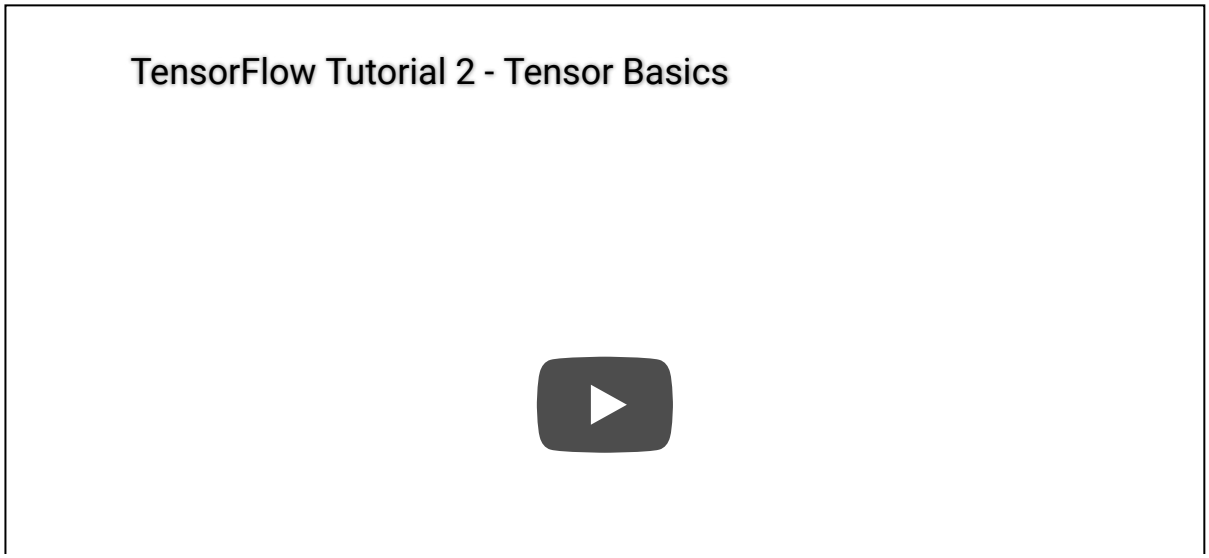
Observation

- Look at the dimension which outputs 3. Compare it with previous outputs.
- Look at the shape which has 3 values (3,3,5) where first value 3 is no of matrices, 2nd value 3 is no of rows and third value 5 is no of columns.

Defining Tensors of Different Formats

Watch this video for basic understanding on tensor operations in tensorflow

```
YouTubeVideo('HPjBY1H-U4U', width=600, height=400)
```




```
# import tensorflow as tf
import tensorflow as tf


# create tensor of one's with shape (3,1)
x = # code here
print(x)
```

```
tf.Tensor(
[[1.]
 [1.]
 [1.]], shape=(3, 1), dtype=float32)
```

```
# create tensor of zeros (3,1)
y = # code here

# print x + y
```

```
tf.Tensor(
[[1.]
 [1.]
 [1.]], shape=(3, 1), dtype=float32)
```

```
# create tensor of random values using random.uniform with shape (5,1)

x = #code here

# print x
```

```
tf.Tensor(
[[0.50145614]
 [0.31028676]
 [0.6424135 ]
 [0.61716807]
 [0.13026702]], shape=(5, 1), dtype=float32)
```

```
# create tensor of random values using random.uniform with shape (5,1) with a minv
```

```
x = # code here

# print x
```

```
tf.Tensor(
[[3.866668 ]
 [2.3763852]
 [3.5798407]
 [2.9231334]
 [2.1753225]], shape=(5, 1), dtype=float32)
```

```
# create tensor of random values using random.normal with a defined mean = 0.,  an

x = # code here

# print x
```

```
tf.Tensor(
[[-0.04473434]
 [-2.001082  ]
 [-0.5830904 ]
 [-0.28463715]
 [-1.3719407 ]], shape=(5, 1), dtype=float32)
```

```
# Do you remember assigning a value in an array ?

# Let's try assigning a value in a tensor (x[0, 0] = 0.)
x[0, 0] = # code here
```

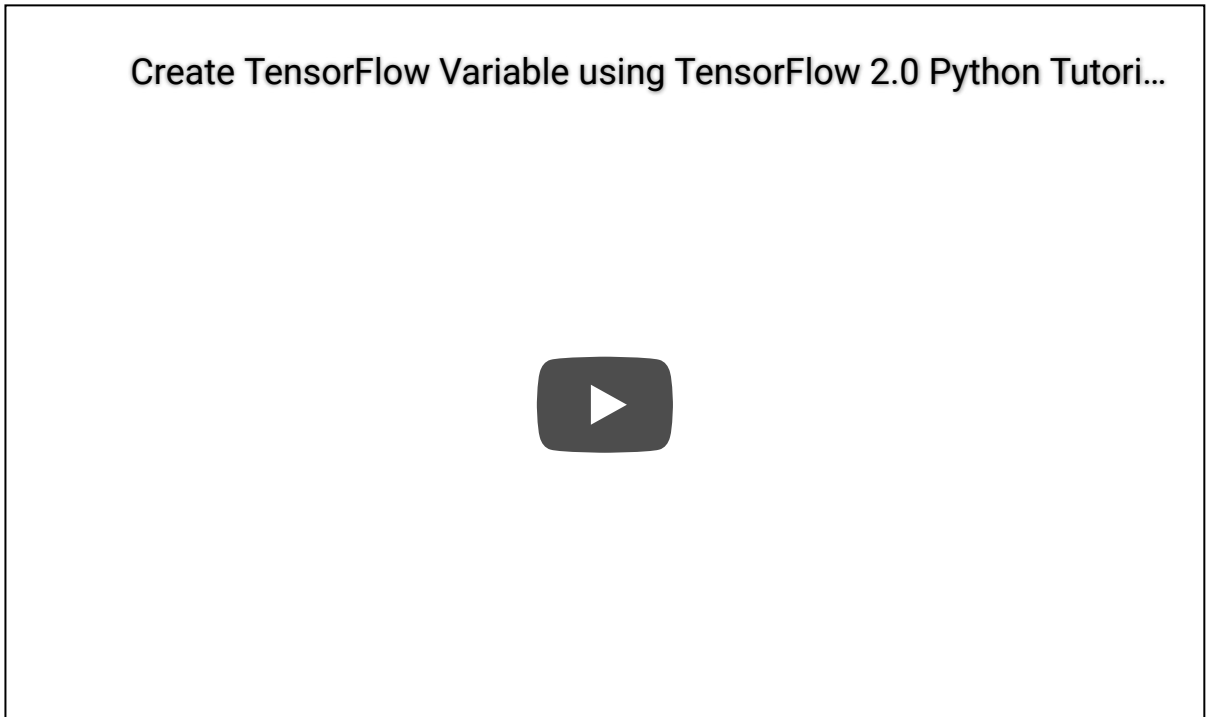
```
  File "<ipython-input-11-5528a1cb6de4>", line 1
    = # Do you remember assigning a value in an array ?
    ^
SyntaxError: invalid syntax
```

SEARCH STACK OVERFLOW

We can see, updating the state of tensor above throw error. So we need to use variables in tensor. tf.Variables is the class meant to manage modifiable state in tensorflow.

Watch this video to understand how tf.variable, tf.assign_add, tf.assign works.

```
YouTubeVideo('HbsOePoHSOs', width=600, height=400)
```




```
# Create a tensor using tf.Variable with initial_value = tf.random.normal having sl

x = # code here

# print x
```

```
<tf.Variable 'Variable:0' shape=(3, 1) dtype=float32, numpy=
array([[2.002882  ],
       [0.52581215],
       [1.819485  ]], dtype=float32)>
```

```
# assigning value 1. in the tensor variable x using .assign method at position [0,(



# print x
```

```
<tf.Variable 'Variable:0' shape=(3, 1) dtype=float32, numpy=
array([[1.        ],
       [0.52581215],
       [1.819485  ]], dtype=float32)>
```

```
# adding one to each value of the tensor variable x using assign_add method
```

```
<tf.Variable 'UnreadVariable' shape=(3, 1) dtype=float32, numpy=
array([[2.       ],
       [1.5258121],
       [2.819485 ]], dtype=float32)>
```

Mathematical Operations in Tensorflow

Math

Arithmetic Operators

```
tf.add(x, y, name=None)
tf.sub(x, y, name=None)
tf.mul(x, y, name=None)
tf.div(x, y, name=None)
tf.truediv(x, y, name=None)
tf.floordiv(x, y, name=None)
tf.mod(x, y, name=None)
tf.cross(a, b, name=None)
```

Basic Math Functions

```
tf.add_n(inputs, name=None)
tf.abs(x, name=None)
tf.neg(x, name=None)
tf.sign(x, name=None)
tf.inv(x, name=None)
tf.square(x, name=None)
tf.round(x, name=None)
tf.sqrt(x, name=None)
tf.rsqrt(x, name=None)
tf.pow(x, y, name=None)
tf.exp(x, name=None)
```

Some tensorflow methods

In TensorFlow the differences between constants and variables are that when you declare some constant, its value can't be changed in the future (also the initialization should be with a value, not with operation).

Nevertheless, when you declare a Variable, you can change its value in the future with tf.assign() method (and the initialization can be achieved with a value or operation).

```
#  All eager tf.Tensor values are immutable (in contrast to tf.Variable)

# define a using tf.constant and pass [40., 30., 50.]
a = # code here

# define b using tf.constant and pass [12., 13., 23.]
b = # code here

# add a and b using tf.add
```

```
<tf.Tensor: shape=(3,), dtype=float32, numpy=array([52., 43., 73.], dtype=flo
```

```
# define x using tf.variable and pass initial value as tf.random.uniform(shape=(2,
x = # code here
# define y by squaring x using tf.square
y = # code here

# print x and y
```

```
<tf.Variable 'Variable:0' shape=(2, 3) dtype=float32, numpy=
array([[4.7649364, 4.914835 , 4.493827 ],
       [4.5438232, 3.9141636, 3.9398665]], dtype=float32)>
tf.Tensor(
[[22.704618 24.155603 20.194479]
 [20.64633  15.320677 15.522549]], shape=(2, 3), dtype=float32)
```

```
# define z by taking the square root of x using tf.sqrt
z = # code here

# print x+z
```

```
tf.Tensor(
[[6.94781   7.131778  6.613692 ]
 [6.6754475 5.892588  5.924776 ]], shape=(2, 3), dtype=float32)
```

Numpy Compatibility

```
import numpy as np

# define an array with shape (4,3) using np.ones
ndarray = # code here

print("TensorFlow operations convert numpy arrays to Tensors automatically")
# define a variable tensor by multiplying ndarray with value 42 (use tf.multiply)
tensor = # code here

# print variable tensor
```

```
print("And NumPy operations convert Tensors to numpy arrays automatically")
# add one in each value of a tensor using np.add


print("The .numpy() method explicitly converts a Tensor to a numpy array")
# convert tensor into numpy using tensor.numpy and print it
```

```
TensorFlow operations convert numpy arrays to Tensors automatically
tf.Tensor(
[[42. 42. 42.]
 [42. 42. 42.]
 [42. 42. 42.]
 [42. 42. 42.]], shape=(4, 3), dtype=float64)
And NumPy operations convert Tensors to numpy arrays automatically
[[43. 43. 43.]
 [43. 43. 43.]
 [43. 43. 43.]
 [43. 43. 43.]]
The .numpy() method explicitly converts a Tensor to a numpy array
[[42. 42. 42.]
 [42. 42. 42.]
 [42. 42. 42.]
 [42. 42. 42.]]
```

How to do gradient of any differentiable expression?

You must be asking yourself, what is the difference between numpy and tensorflow here. Suppose you want to differentiate some expression, numpy can't help you there. Tensorflow comes in handy then.

1. $\frac{d}{dx}(x^n) = nx^{n-1}$
2. $\frac{d}{dx}(x) = 1$
3. $\frac{d}{dx}(k) = 0$, k is a constant
4. $\frac{d}{dx}(kx) = k$, k is a constant
5. $\frac{d}{dx}\left(\frac{1}{x}\right) = -\frac{1}{x^2}$
6. $\frac{d}{dx}(\sqrt{x}) = \frac{1}{2\sqrt{x}}$
7. $\frac{d}{dx}(e^x) = e^x$
8. $\frac{d}{dx}(e^{ax}) = ae^{ax}$
9. $\frac{d}{dx}(e^{ax+b}) = ae^{ax+b}$
10. $\frac{d}{dx}(e^{-x^2}) = -2x\,e^{-x^2}$
11. $\frac{d}{dx}(a^x) = a^x \log_e a$

Watch this tutorial to understand how gradient works in tensorflow.

```
YouTubeVideo('ENOycxDU9RY', width=600, height=400)
```

TensorFlow Tutorial 6- GradientTape in TensorFlow

```
# Using GradientTape(Sample example)

# taking some input
some_input = tf.Variable(initial_value = 5.)

# defining GradientTape as tape
with tf.GradientTape() as tape:
  result = tf.square(some_input)

# using gradient tape to find gradient
gradient = tape.gradient(result, some_input)
```

```
# printing some_input and gradient
print(some_input)
print(gradient)
```

```
<tf.Variable 'Variable:0' shape=() dtype=float32, numpy=5.0>
tf.Tensor(10.0, shape=(), dtype=float32)
```

```
# another example of gradient

# define variable x using tf.variable and pass value as 3.0
x = # code here

# define GradientTape as tape with y = x**2


# define dy_dx and take derivative using tape.gradient


# print x, y and dy_dx
```

```
x:  <tf.Variable 'Variable:0' shape=() dtype=float32, numpy=3.0>
y:  tf.Tensor(9.0, shape=(), dtype=float32)
dy_dx:  tf.Tensor(6.0, shape=(), dtype=float32)
```

```
# (Add on example)
# Another example of gradient using equation of
# falling apple along a vertical exis over time
time = tf.Variable(3.)
with tf.GradientTape() as outer:
  with tf.GradientTape() as inner:
    position = 4.9 * time ** 2
  speed = inner.gradient(position, time)
acceleration = outer.gradient(speed, time)
```

```
# printing time, position, speed and acceleration
print("time: ", time)
print("position: ", position)
print("speed: ", speed)
print("acceleration: ", acceleration)
```

```
time:  <tf.Variable 'Variable:0' shape=() dtype=float32, numpy=3.0>
position:  tf.Tensor(44.100002, shape=(), dtype=float32)
speed:  tf.Tensor(29.400002, shape=(), dtype=float32)
acceleration:  tf.Tensor(9.8, shape=(), dtype=float32)
```

```
# Another example using weights and biases

# define w using tf.Variable and pass random values with shape (3,2) using tf.rand
w = # code here

# define b using tf.Variable and pass zeros with shape 2 using tf.zeros
b = # code here

# define x with values [[1., 2., 3.]]


# define GradientTape as tape
with tf.GradientTape(persistent=True) as tape:
  # define y under it with values as y = x @ w + b (@ is dot product)

  # define loss using tf.reduce_mean and pass y**2 into it


# print w
print("w: ", w)
# print b
print("b: ", b)
# print x
print("x: ", x)
# print y
print("y: ", y)
# print y**2
print("y**2: ", y**2)
# print loss
print("loss: ", loss)

print("*"*50)

# Now differentiate y w.r.t w and b
[dy_dw, dy_db] = # code here

# Now print dy_dw, dy_db
```

```
w:  <tf.Variable 'w:0' shape=(3, 2) dtype=float32, numpy=
array([[ 0.62128025, -0.83473617],
       [-0.6274292 , -0.9952367 ],
```

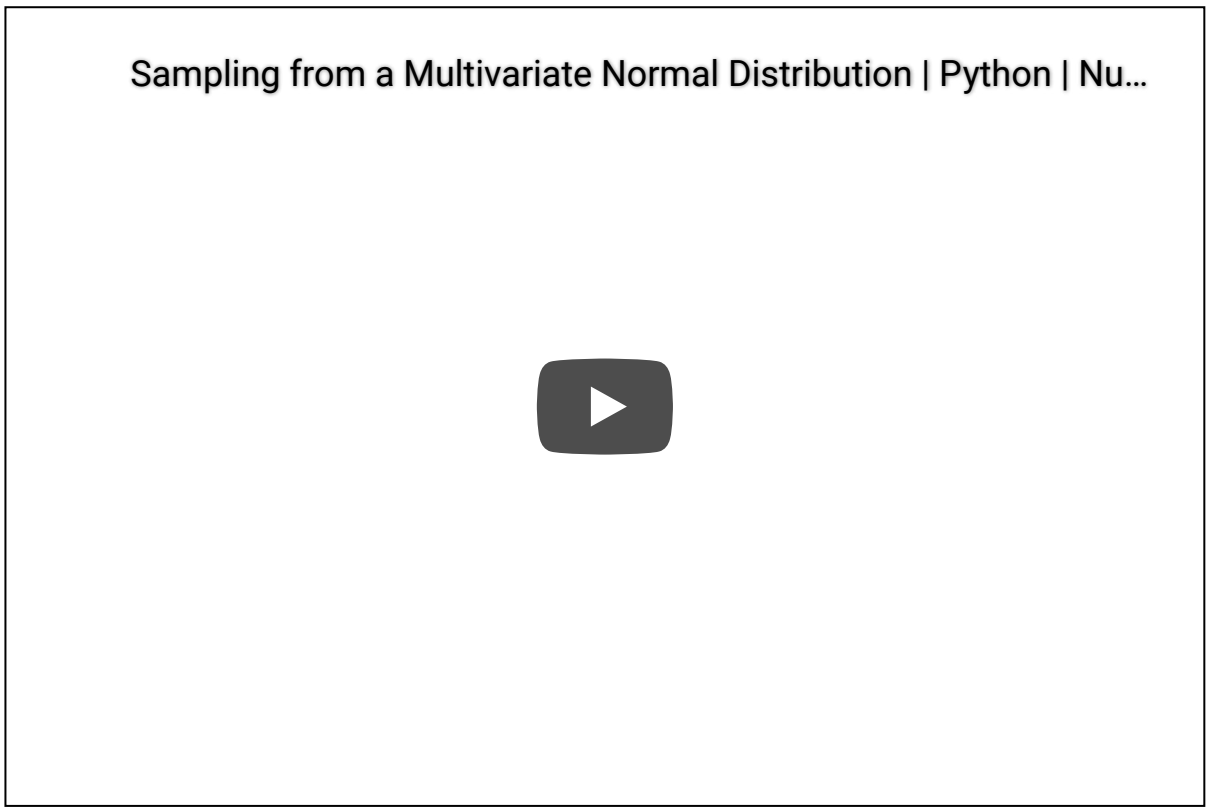
```
       [-0.65404075, -1.8257358 ]], dtype=float32)>
b:  <tf.Variable 'b:0' shape=(2,) dtype=float32, numpy=array([0., 0.], dtype=
x:  [[1.0, 2.0, 3.0]]
y:  tf.Tensor([[-2.5957003 -8.302417 ]], shape=(1, 2), dtype=float32)
y**2:  tf.Tensor([[ 6.73766 68.93012]], shape=(1, 2), dtype=float32)
loss:  tf.Tensor(37.833893, shape=(), dtype=float32)
**************************************************
dy_dw:  tf.Tensor(
[[ -2.5957003  -8.302417 ]
 [ -5.1914005 -16.604834 ]
 [ -7.787101  -24.90725  ]], shape=(3, 2), dtype=float32)
dy_db:  tf.Tensor([-2.5957003 -8.302417 ], shape=(2,), dtype=float32)
```

Beginning of End to End Linear Classifier

** Before we go for linear classifier, let me show you how to plot some points on scatterplot for visualization **

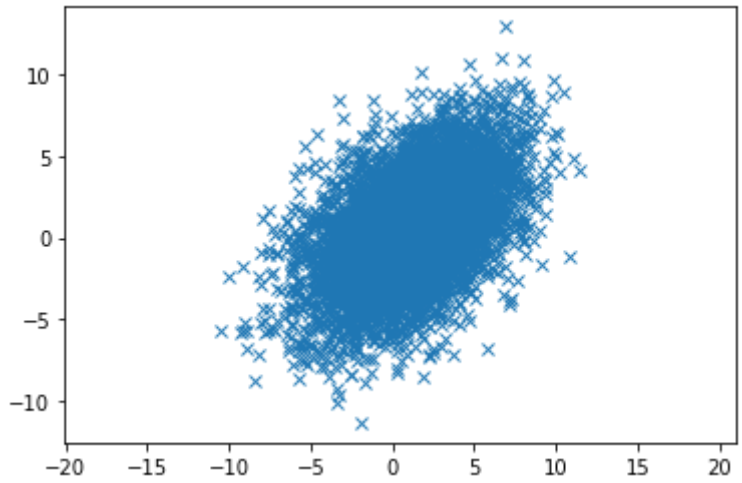
Video reference for multivariate normal in method in numpy

```
YouTubeVideo('mw-svKkGVaI', width=600, height=400)
```




```
# ( Sample code for visualization )
# we will use np.random.multivariate_normal to get random points having specific m

import matplotlib.pyplot as plt
x, y = np.random.multivariate_normal([1, 0.5], [[10, 5], [5, 10]], 5000).T
```

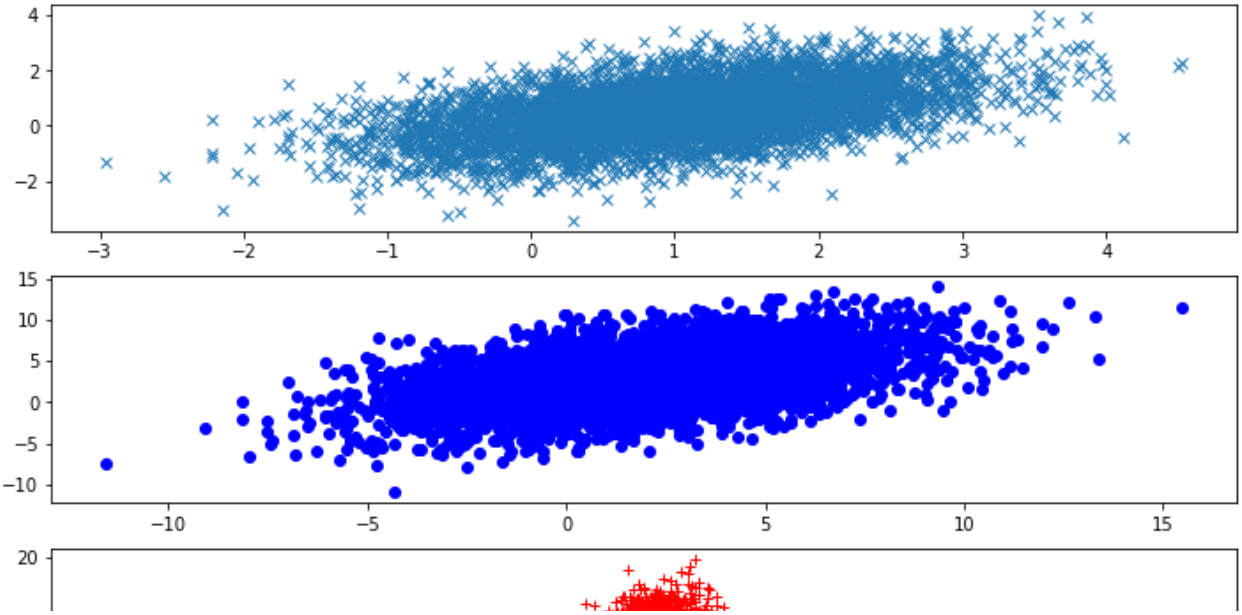
```
plt.plot(x, y, 'x')
plt.axis('equal')
plt.show()
```



Change Mean And Covariance To See The Differences in Plots in Next Cell

```
# we will use np.random.multivariate_normal to get random points having specific m

import matplotlib.pyplot as plt

fig, (ax1, ax2, ax3)= plt.subplots(3, figsize=(12, 8))

# visualize mean, cov
x, y = np.random.multivariate_normal(mean = [1, 0.5], cov = [[1, 0.5], [0.5, 1]],s
ax1.plot(x, y, 'x')
plt.axis('equal')

# visualize mean, cov
a, b = np.random.multivariate_normal(mean = [2, 3], cov = [[10, 5], [5, 10]], size
ax2.plot(a, b, 'bo')
plt.axis('equal')

# visualize mean, cov
c, d = np.random.multivariate_normal(mean = [1, 5], cov = [[5, 15], [15, 5]], size
ax3.plot(c, d, 'r+')
plt.axis('equal')
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:18: RuntimeWar
(-13.59945382244068,
 16.479369635894606,
 -11.03706910299835,
 21.323533444365626)
```



```
# Now we are defining two scatterplot, one for negative and one for positive
num_samples_per_class = 1000

# first negative samples
# Use np.random.multivariate_normal with mean [0, 3] and cov [[1, 0.5], [0.5, 1]] a
negative_samples = # code here

# looking at first 5 negative samples
```

```
array([[ 0.63853525,  0.92629126],
       [-0.95119604,  2.25637077],
       [ 0.57036226,  3.70387882],
       [-0.68399395,  2.85204412],
       [-0.63809375,  3.41171386]])
```

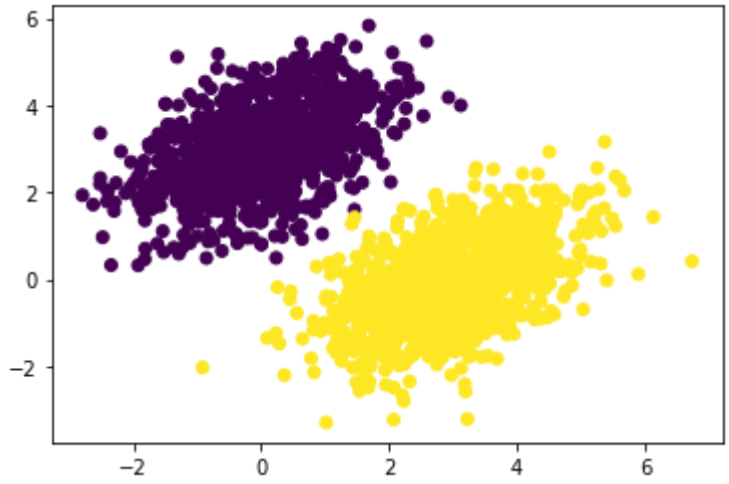
```
# defining positive samples
# Use np.random.multivariate_normal with mean [0, 3] and cov [[1, 0.5], [0.5, 1]] a
positive_samples = # code here

# looking at first 5 negative samples
```

```
array([[ 3.62577961,  0.63049045],
       [ 3.00057025, -0.48772401],
       [ 1.61964011,  0.42030207],
       [ 2.41685612,  0.45998937],
       [ 1.25456665, -0.91131288]])
```

```
# Stacking both positive and negative samples using np.vstack

samples = # code here
```

```
# defining labels using np.vstack (stack vector of zeros and ones having num_sample

targets = # code here


# plot your samples using plt.scatter
```

```
<matplotlib.collections.PathCollection at 0x7ff93743bd50>
```




```
# define input_dim =2 as we have two input variables and output_dim = 1 as we have


# define weights using tf.variable , shape of weights will be = (input_dim, output_

# define bias using tf.variable , shape of bias will be = (output_dim,)



# here is our model
# define a function named simple_model which will take inputs(X) and return (inputs
def simple_model(inputs):
  # code here



# returning avg loss from this loss function
# define mean_sq_loss function which will take targets and predictions
def mean_sq_loss(targets, predictions):
  # define losses variable first by taking square difference of targets and predict

  # return mean of losses using tf.reduce_mean



# define learning_rate=0.1


# define training function which takes inputs and targets
def training(inputs, targets):
  # define GradientTape as tape
```

```
      # define predictions by using simple_model function

      # define losses using mean_sq_loss function

    # take derivative of loss w.r.t. w and b

    # assign loss w.r.t.w*learning_rate to weights

    # assign loss w.r.t.b*learning_rate to bias

    # return losses



# running training for multiple epochs usinf for loop

    # define loss by calling training function

    # print loss epoch wise
```

```
At epoch 0, loss is 0.03149564191699028
At epoch 1, loss is 0.030938195064663887
At epoch 2, loss is 0.030428165569901466
At epoch 3, loss is 0.029961535707116127
At epoch 4, loss is 0.029534602537751198
At epoch 5, loss is 0.029143990948796272
At epoch 6, loss is 0.028786612674593925
At epoch 7, loss is 0.028459643945097923
At epoch 8, loss is 0.028160488232970238
At epoch 9, loss is 0.027886781841516495
At epoch 10, loss is 0.027636364102363586
At epoch 11, loss is 0.02740725502371788
At epoch 12, loss is 0.02719763293862343
At epoch 13, loss is 0.027005847543478012
At epoch 14, loss is 0.02683037891983986
At epoch 15, loss is 0.026669835671782494
At epoch 16, loss is 0.026522956788539886
At epoch 17, loss is 0.026388566941022873
At epoch 18, loss is 0.026265617460012436
At epoch 19, loss is 0.026153123006224632
At epoch 20, loss is 0.026050202548503876
At epoch 21, loss is 0.025956036522984505
At epoch 22, loss is 0.025869883596897125
At epoch 23, loss is 0.02579105831682682
At epoch 24, loss is 0.025718940421938896
At epoch 25, loss is 0.02565295808017254
At epoch 26, loss is 0.025592589750885963
At epoch 27, loss is 0.025537356734275818
At epoch 28, loss is 0.025486823171377182
At epoch 29, loss is 0.02544058859348297
```

FEEDBACK FORM

Please help us in improving by filling this form.
https://forms.zohopublic.in/cloudyml/form/CloudyMLDeepLearningFeedbackForm/formperma/VCFbldnXAnbcgAIl0lWv2blgHdSldheO4RfktMdgK7s