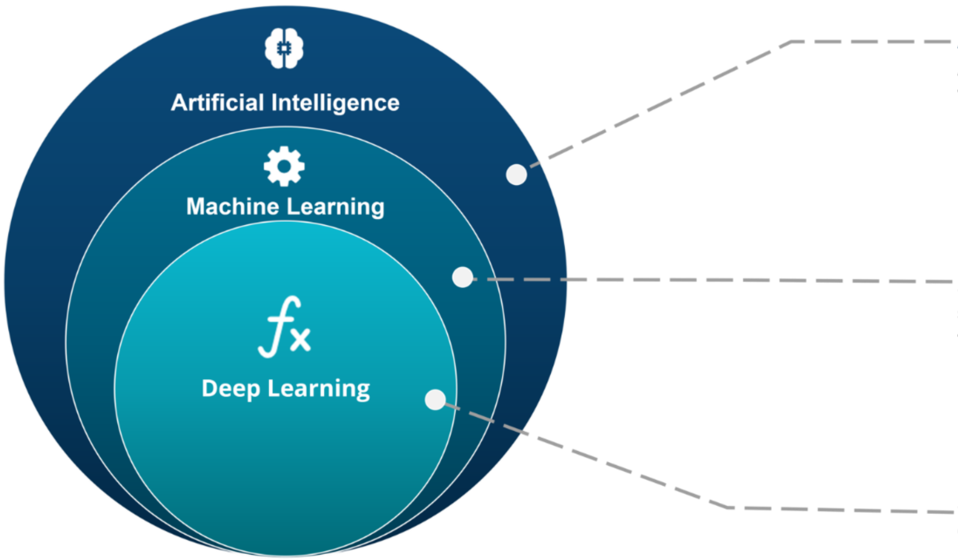
DIP: Tensor Flow _ Introduction

Presenter: Dr. Ha Viet Uyen Synh.

ARTIFICIAL INTELLIGENCE, MACHINE LEARNING & DEEP LEARNING

Introduction



ARTIFICIAL INTELLIGENCE
A technique which enables machines to mimic human behaviour

MACHINE LEARNING
Subset of AI technique which use statistical methods to enable machines to improve with experience

DEEP LEARNING
Subset of ML which make the computation of multi-layer neural network feasible

What is Artificial Intelligence ?

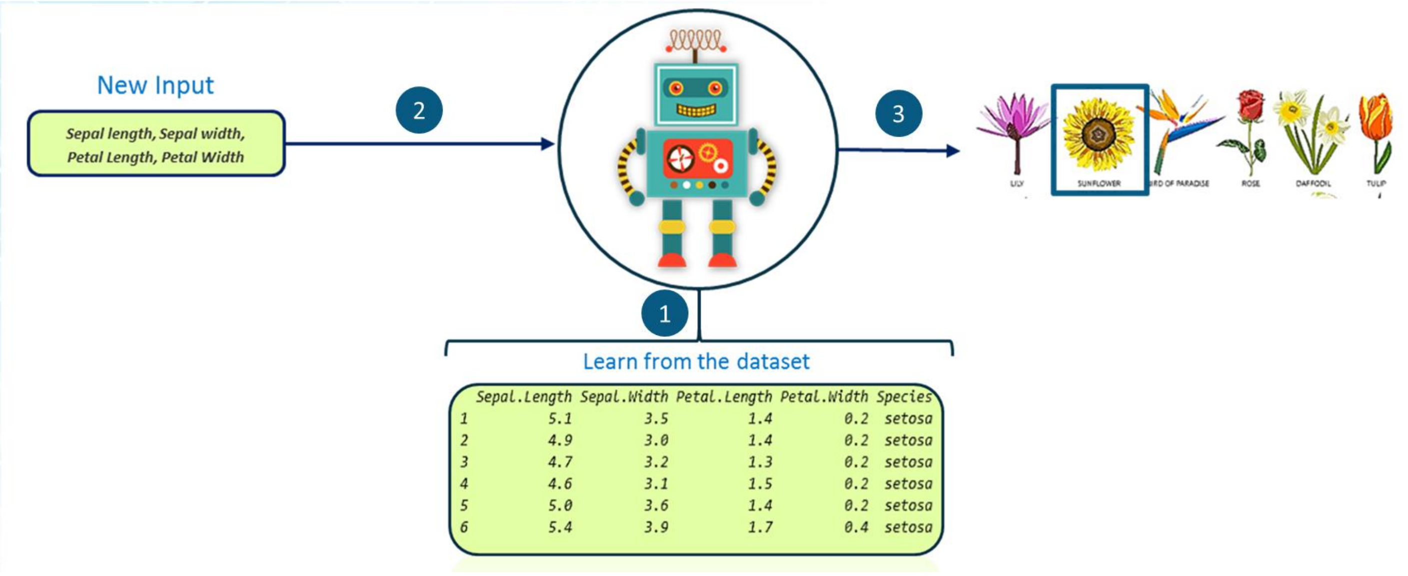
Artificial Intelligence is nothing but the capability of a machine to imitate intelligent human behavior.

AI is achieved by mimicking a human brain, by understanding how it thinks, how it learns, decides, and work while trying to solve a problem.

What is Machine Learning ?

Machine Learning is a subset of Artificial Intelligence which provide computers with the ability to learn without being explicitly programmed.

In machine learning, we do not have to define explicitly all the steps or conditions like any other programming application. On the contrary, the machine gets trained on a training dataset, large enough to create a model, which helps machine to take decisions based on its learning.

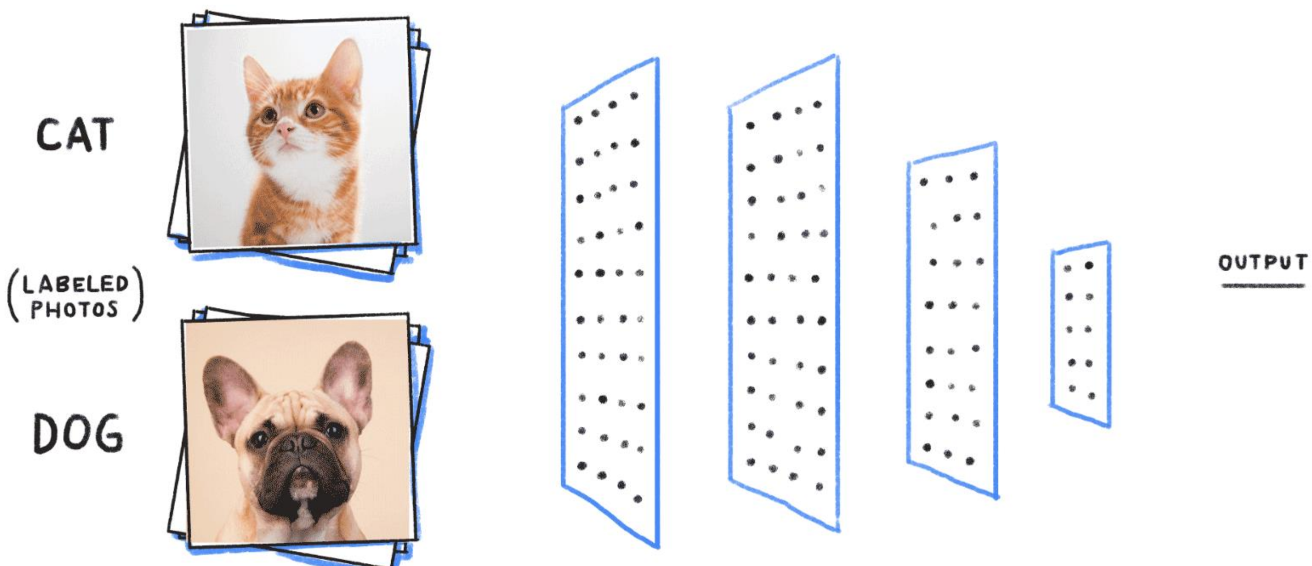


	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

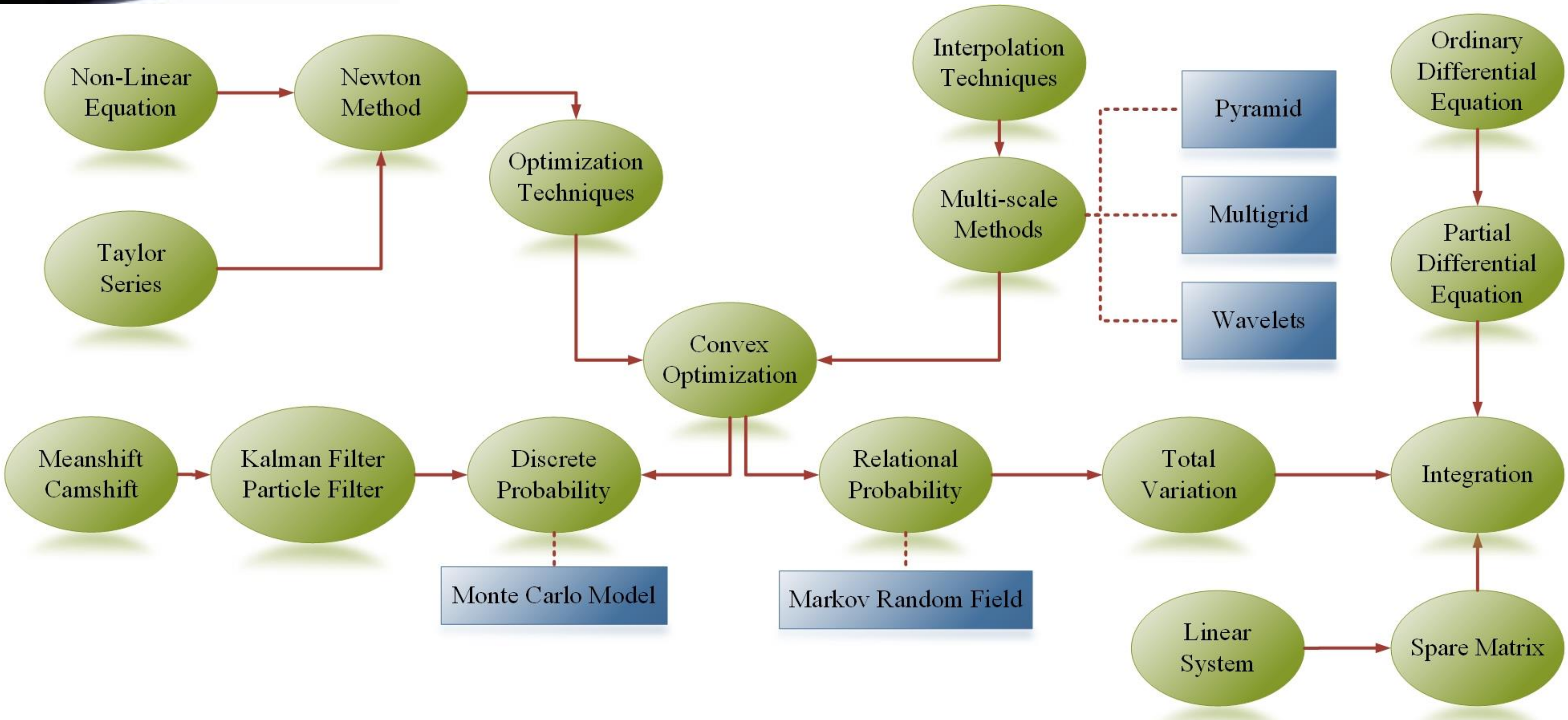
What is Deep Learning?

Deep learning is one of the only methods by which we can overcome the challenges of feature extraction. This is because deep learning models are capable of learning to focus on the right features by themselves, requiring little guidance from the programmer.

"Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as nested hierarchy of concepts or abstraction"



The Mind map of Engineering Mathematics



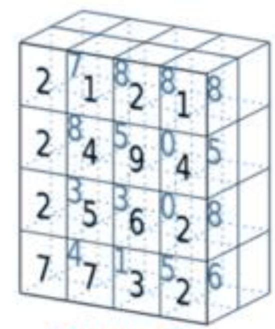
TENSORFLOW

What are Tensors?

Tensors are just multidimensional arrays, that allows you to represent data having higher dimensions.

Tensor of dimension[1]

Tensor of dimensions[2]



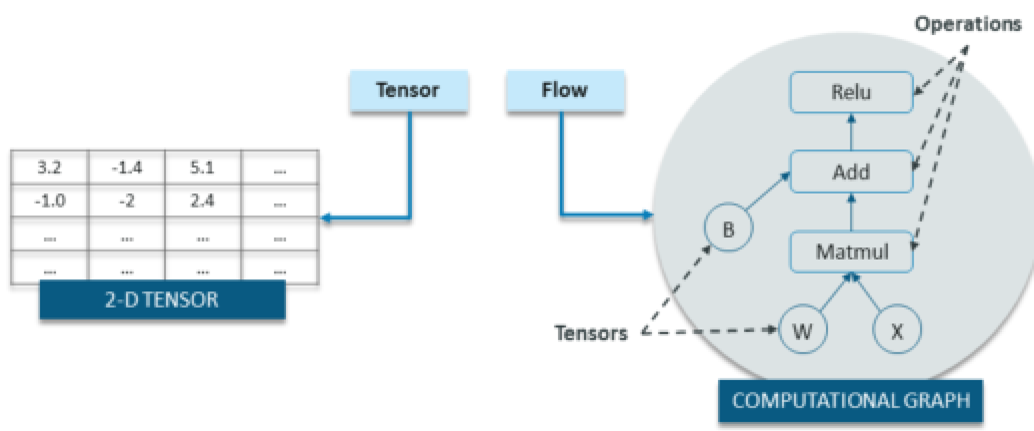
Tensor of dimensions[3]

What's TensorFlow™?

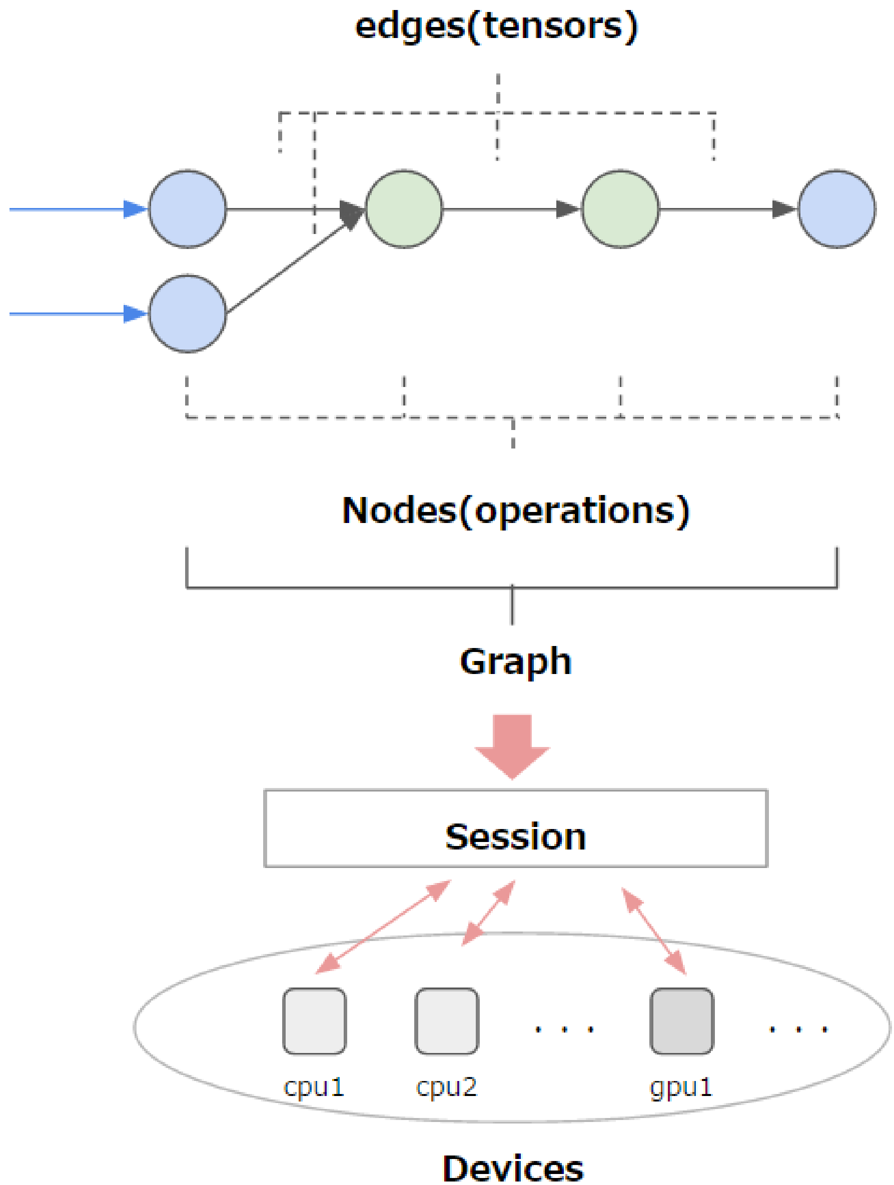
TensorFlow is a library for numerical computation where data flows through the graph.

The term TensorFlow is made up of two terms – Tensor & Flow

The term tensor refers to the representation of data as multi-dimensional array whereas the term flow refers to the series of operations that one performs on tensors.



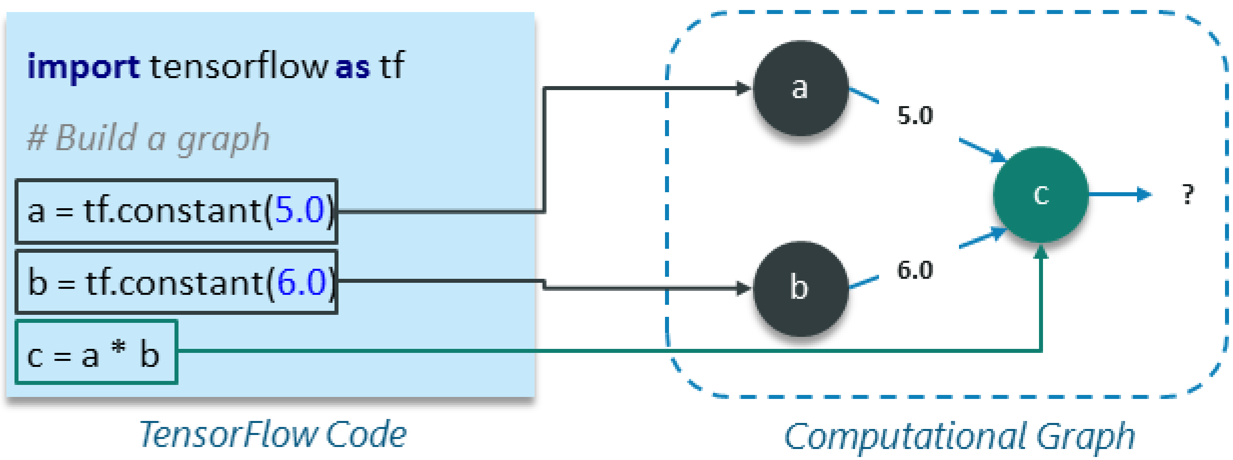
A TensorFlow program



1. Building a Computational Graph

A computational graph is a series of TensorFlow operations arranged as nodes in the graph.

Each nodes take 0 or more tensors as input and produces a tensor as output



2. Running a Computational Graph

A graph is used to define operations, but the operations are only run within a session.

Graphs and sessions are created independently of each other.

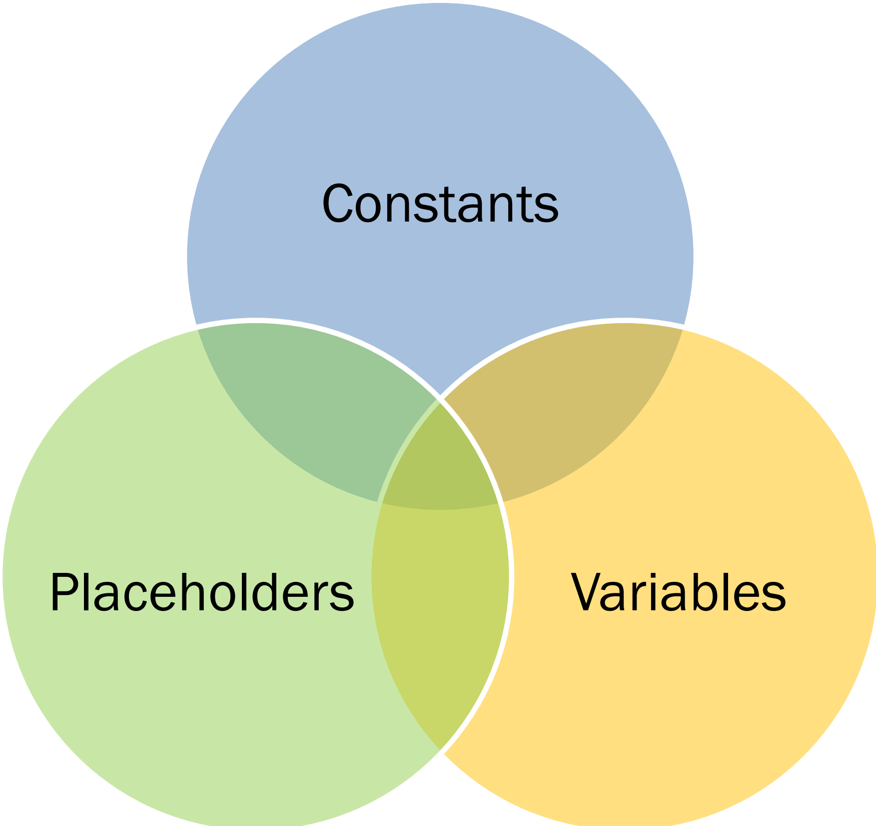
Graph to be similar to a blueprint, and a session to be similar to a construction site.

Example

```
import tensorflow as tf

# Build a graph
a = tf.constant(5.0)
b = tf.constant(6.0)
c = a * b


# Create the session object
sess = tf.Session()

#Run the graph within a session and store the output to a variable
output_c = sess.run(c)

#Print the output of node c
print(output_c)

#Close the session to free up some resources
sess.close()
```

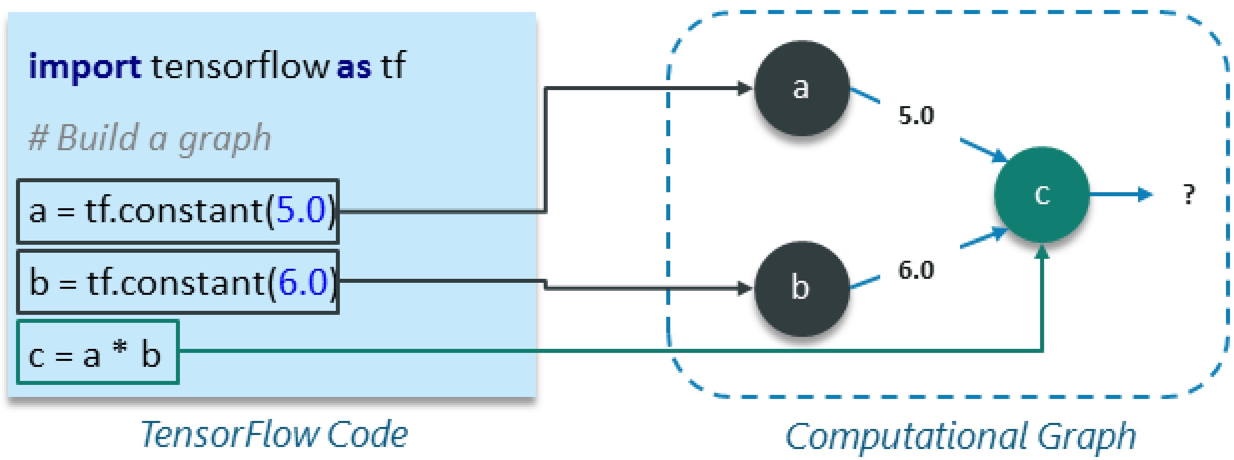
Tensors in TensorFlow

TF holds data in Tensors which are similar to numPy multi-dimensional arrays.



Constants

Constant nodes are used to store constant values as it takes zero input, but produces the stored values as output.



Variables

A variable allows you to add such parameters or node to the graph that are trainable i.e. the value can be modified over the period of a time

Ex: var = tf.Variable([0.4], dtype = tf.float32)

Variables are not initialized when you call tf.Variable. To initialize all the variables in a TensorFlow program, you must explicitly call a special operation tf.global_variables_initializer()

Ex:

```
init = tf.global_variables_initializer()
sess.run(init)
```

Placeholder

Placeholders are used which allows your graph to take external inputs as parameters. Basically, a placeholder is a promise to provide a value later or during runtime.

Placeholders are not initialized and contains no data.

One must provides inputs or feeds to the placeholder which are considered during runtime.

Executing a placeholder without input generates an error.

Example

```
import tensorflow as tf

# Creating placeholders
a = tf. placeholder(tf.float32)
b = tf. placeholder(tf.float32)

# Assigning multiplication operation w.r.t. a &amp;amp; b to node mul
mul = a*b

# Create session object
sess = tf.Session()

# Executing mul by passing the values [1, 3] [2, 4] for a and b respectively
output = sess.run(mul, {a: [1,3], b: [2, 4]})
print('Multiplying a b:', output)
```

Output: [2. 12.]

Device in TensorFlow

TensorFlow has very strong in-built capabilites to run your code on a gpu or a cpu or a cluster of gpu etc.

It provides you options to select the device you want to run your code.

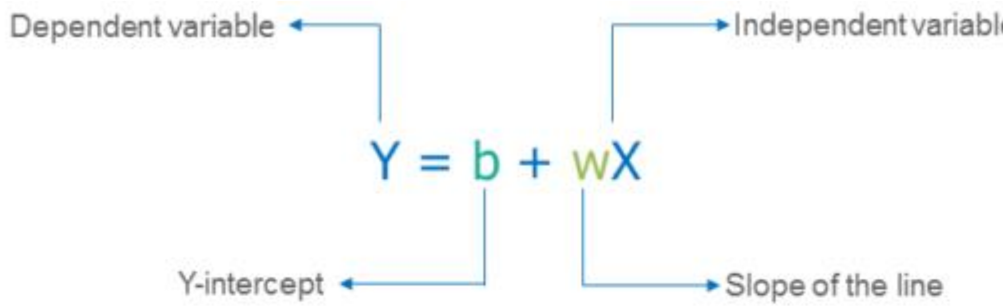
EXAMPLES

Linear Regression Model

Linear Regression Model is used for predicting the unknown value of a variable (Dependent Variable) from the known value of another variables (Independent Variable) using linear regression equation

Therefore, for creating a linear model, you need:

- Dependent or Output Variable (Y)
- Slope Variable (w)
- Y – Intercept or Bias (b)
- Independent or Input Variable (X)



```
import tensorflow as tf
```

Importing the required library - tensorflow

```
# Model parameters
w = tf.Variable([.4], dtype=tf.float32)
b = tf.Variable([-.4], dtype=tf.float32)
```

Creating variables for model parameters:

- Slope Variable (w): Initial Value 0.4
- Bias Variable (b): Initial Value -0.4

```
# Model input
x = tf.placeholder(tf.float32)
```

Creating placeholder for model input or independent variable

```
# Equation of Linear Regression
linear_model = W * x + b
```

Assigning Linear Regression Model Equation to the node *linear_model*

```
# Initializing all the variables
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)
```

Initializing all the variable created so far.

```
# Running regression model
print(sess.run(linear_model {x: [1, 2, 3, 4]}))
```

Evaluating and printing the output of linear model w.r.t. x = [1, 2, 3, 4]

Output: [0.   0.40000001 0.80000007 1.20000005]

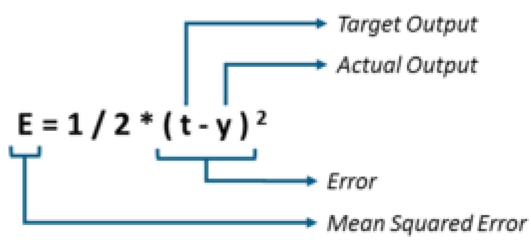
Source code

```
# Creating variable for parameter slope (W) with initial value as 0.4
W = tf.Variable([.4], tf.float32)

#Creating variable for parameter bias (b) with initial value as -0.4
b = tf.Variable([-0.4], tf.float32)

# Creating placeholders for providing input or independent variable, denoted by x
x = tf.placeholder(tf.float32)

# Equation of Linear Regression
linear_model = W * x + b

# Initializing all the variables
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)

# Running regression model to calculate the output w.r.t. to provided x values
print(sess.run(linear_model {x: [1, 2, 3, 4]}))
```

Loss Function – Model Validation

A loss function measures how far apart the current output of the model is from that of the desired or target output.

We'll use a most commonly used loss function for my linear regression model called as Sum of Squared Error or SSE. SSE calculated w.r.t. model output (represent by linear_model) and desired or target output (y)



```
# Placeholder for desired output
y = tf.placeholder(tf.float32)
```

Placeholder to accept the desired or target y values corresponding to x values

```
#Calculate Sum of Squared Error
error = linear_model – y
squared_errors = tf.square(error)
loss = tf.reduce_sum(squared_errors)
```

Calculating Sum of Squared Error (SSE) by taking the average of squared errors where error = (actual output - desired output)

```
#Print loss
print(sess.run(loss, {x:[1,2,3,4], y:[2, 4, 6, 8]})
```

Calculating loss w.r.t. provided input (x) and desired output (y)

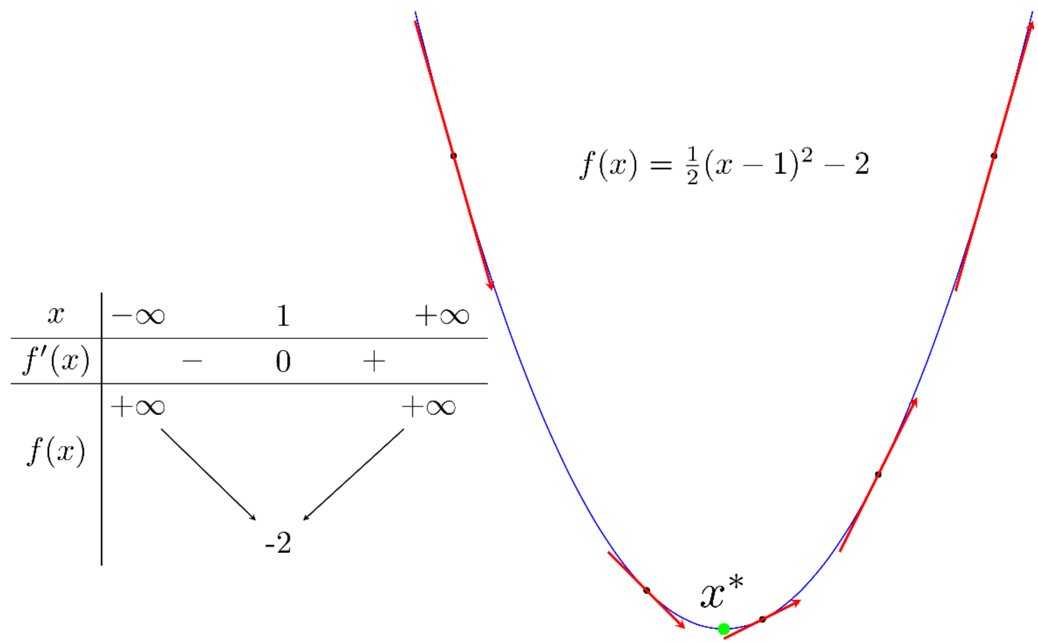
```
y = tf.placeholder(tf.float32)

error = linear_model – y

squared_errors = tf.square(error)

loss = tf.reduce_sum(squared_errors)

print(sess.run(loss, {x:[1,2,3,4], y:[2, 4, 6, 8]})
```

Output: 90.24

tf.train API – Training the Model

TensorFlow provides optimizers that slowly change each variable in order to minimize the loss function or error.

The simplest optimizer is gradient descent. It modifies each variable according to the magnitude of the derivative of loss with respect to that variable.



```
# Creating an instance of gradient descent optimizer
optimizer = tf.train.GradientDescentOptimizer(0.01)
```

Creating an instance of gradient descent optimizer with a learning rate of 0.01.

```
# Minimize the loss using optimizer
train = optimizer.minimize(loss)
```

The node train is assigned with the optimization operation that will be minimize the error or loss by modifying the value of model parameters w and b.

```
# Minimizing error in successive iteration
for i in range(1000)
    sess.run(train, {x:[1, 2, 3, 4], y:[2, 4, 6, 8]})
```

Minimizing error in 1000 iteration such that at each iteration the optimizer will be called to modify the model parameter w & b based on loss to minimize the error

```
# Creating an instance of gradient descent optimizer
print(sess.run([W, b]))
```

Print the final weight and bias values

```
#Creating an instance of gradient descent optimizer
optimizer = tf.train.GradientDescentOptimizer(0.01)
 train = optimizer.minimize(loss)
 for i in range(1000):
    sess.run(train, {x:[1, 2, 3, 4], y:[2, 4, 6, 8]})
print(sess.run([W, b]))
```

Output: [array([1.99999964], dtype=float32), array([9.86305167e-07], dtype=float32)]

A Simple Program

```
import tensorflow as tf
import numpy as np

trainX = np.linspace(-1, 1, 101)
trainY = 3 * trainX + np.random.randn(*trainX.shape) * 0.33

X = tf.placeholder("float")
Y = tf.placeholder("float")

w = tf.Variable(0.0, name="weights")
init = tf.global_variables_initializer()

y_model = tf.multiply(X, w)

cost = (tf.pow(Y-y_model, 2))

train_op = tf.train.GradientDescentOptimizer(0.01).minimize(cost)

with tf.Session() as sess:
    sess.run(init)
    for i in range(100):
        for (x, y) in zip(trainX, trainY):
            sess.run(train_op, feed_dict={X: x, Y: y})
    print(sess.run(w))
```

Questions? More Information?

hvusynh@hcmiu.edu.vn