Deep Learning Using TensorFlow

Dr. Ash Pahwa

Lesson 2: TensorFlow

Lesson 2.1: TensorFlow Architecture

Outline

- 1. What is TensorFlow
- 2. History of TensorFlow
- 3. Advantages of Directed Acyclic Graph (DAG)
- 4.1 TensorFlow API Hierarchy
- 4.2 Mode of Execution: Lazy & Eager
- 5.1 DAG: Directed Acyclic Graph
- 5.2 Evaluating a Tensor
- 5.3 Visualizing a Graph (DAG)
- 6.1 Creating Tensors
- 6.2 Variables and Place Holders

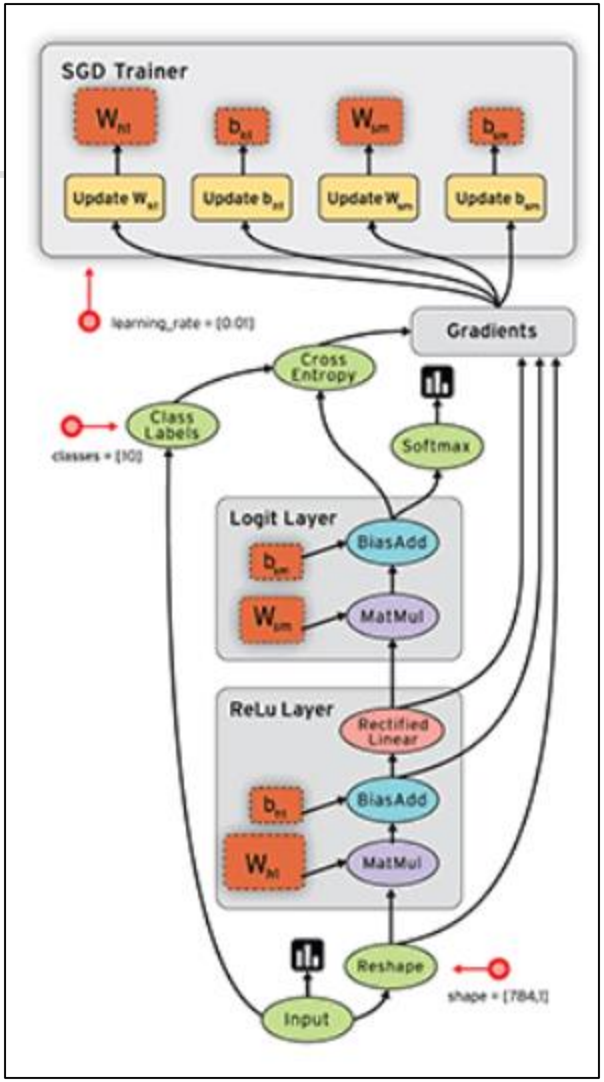
1. What is TensorFlow?

What is TensorFlow?

- TensorFlow
  - Open source
  - High performance library
  - Primary Focus – Numerical Computing
- Can be used for any numerical computing
  - GPU Programming
  - Partial Differential Equation

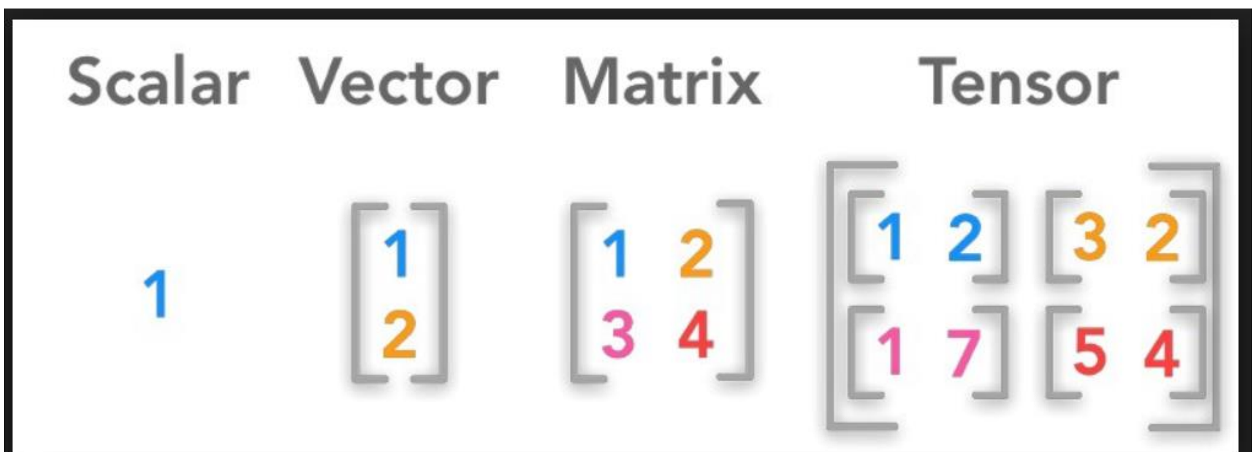
TensorFlow

- Creates Directed Acyclic Graph (DAG)
  - DAG represents mathematical operations
    - + - * /
    - Vector arithmetic
    - Matrix multiplication
- DAG
  - Edges
    - Input/output of math operation
    - Represents – array of data



TensorFlow

- Tensor Rank 0
  - Scalar
- Tensor Rank 1
  - One dimensional array - vector
- Tensor Rank 2
  - 2 dimensional array - matrix
- Tensor Rank 3
  - 3 dimensional array
- Tensor Rank 4
  - 4 dimensional array



Why the name TensorFlow?

- Data is represented by Tensor
- Create DAG (Directed Acyclic Graph) to represent computation
- Tensors flow through DAG
  - Hence the name TensorFlow

2. History of TensorFlow

History of TensorFlow

- **TensorFlow** is an open source software library released in 2015 by Google to make it easier for developers to design, build, and train deep learning models
- At a high level, **TensorFlow** is a Python library that allows users to express arbitrary computation as a graph of data flows

3. Advantages of a DAG

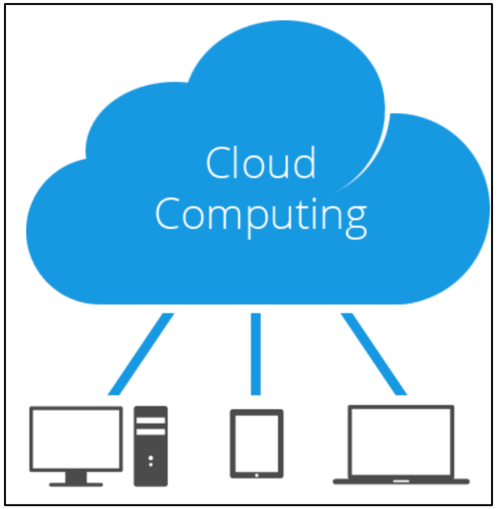
Advantages of Directed Acyclic Graph (DAG)

- Portability – hardware and software
  - DAG is language independent representation of code of your model
  - Dag can be used with C++ and Python
  - CPU or GPU
- Similar to Java Virtual Machine (JVM)
  - Works on all platforms

Advantages of DAG

Train your model on Cloud where you have access to very powerful hardware and storage devices with lots of data

Run your model on cell phone



4.1 TensorFlow API Hierarchy

TensorFlow Abstraction Layers
API Hierarchy

	API	
1	tf.estimator	High level API
2	tf.layers, tf.losses, tf.metrics	Custom NN Model components • tf.layers: ReLu activation function, create new hidden layer • tf.metrics: RMSE • tf.losses: Entropy with Logit
3	Core TensorFlow (Python)	Python API: Numeric processing code Add, subtract, multiply, divide matrix Creating variables, Tensors, getting the shape
4	Core TensorFlow (C++)	C++ API Writing custom app
5	CPU + GPU + TPU + Android	TensorFlow for different hardware

Core TensorFlow Numeric Processing

	API	
1	tf.estimator	High level API
2	tf.layers, tf.losses, tf.metrics	Custom NN Model components • tf.layers: ReLu activation function, create new hidden layer • tf.metrics: RMSE • tf.losses: Entropy with Logit
3	Core TensorFlow (Python)	Python API: Numeric processing code Add, subtract, multiply, divide matrix Creating variables, Tensors, getting the shape
4	Core TensorFlow (C++)	C++ API Writing custom app
5	CPU + GPU + TPU + Android	TensorFlow for different hardware

- Numeric Processing Code
  - Add Subtract, Multiply, Divide
  - Matrix multiplication
  - Creating Variables, Tensors
  - Getting the shape
  - Dimensions of a Tensor

Custom Neural Network

	API	
1	tf.estimator	High level API
2	tf.layers, tf.losses, tf.metrics	Custom NN Model components • tf.layers: ReLu activation function, create new hidden layer • tf.metrics: RMSE • tf.losses: Entropy with Logit
3	Core TensorFlow (Python)	Python API: Numeric processing code Add, subtract, multiply, divide matrix Creating variables, Tensors, getting the shape
4	Core TensorFlow (C++)	C++ API Writing custom app
5	CPU + GPU + TPU + Android	TensorFlow for different hardware

- Represent high level representation of useful Neural Network Components
- tf.layers
  - Create a new layer of hidden neurons
  - Create a new activation function
- tf.metrics
  - Compute Root mean square
- tf.losses
  - Cross entropy with logits

tf.estimator

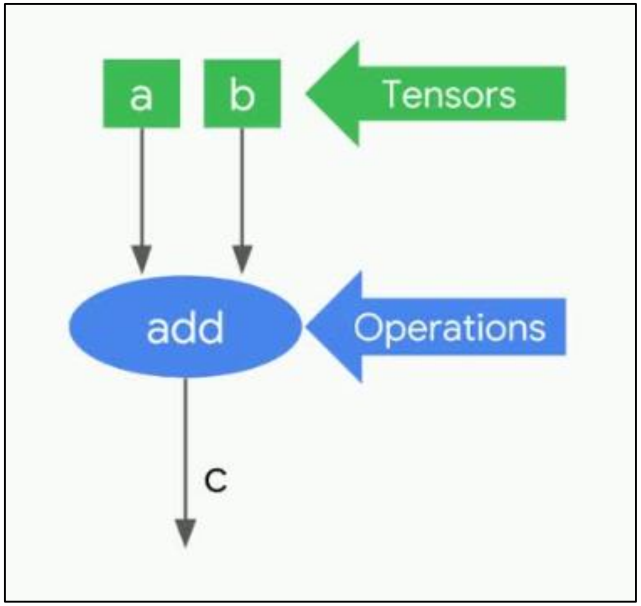
	API	
1	tf.estimator	High level API
2	tf.layers, tf.losses, tf.metrics	Custom NN Model components • tf.layers: ReLu activation function, create new hidden layer • tf.metrics: RMSE • tf.losses: Entropy with Logit
3	Core TensorFlow (Python)	Python API: Numeric processing code Add, subtract, multiply, divide matrix Creating variables, Tensors, getting the shape
4	Core TensorFlow (C++)	C++ API Writing custom app
5	CPU + GPU + TPU + Android	TensorFlow for different hardware

- High Level API
- Comes with all standard features
- Trains the neural network
- How to create a checkpoint
- How to save a model

4.2 Mode of Execution Lazy & Eager

TensorFlow
Step 1: Creates a DAG (Directed Acyclic Graph)
Step 2: Executes DAG

```
import tensorflow as tf

a1 = tf.constant([5,3,8])

b1 = tf.constant([3,-1,2])

c1 = tf.add(a1,b1)

print (c1)
Tensor("Add_1:0", shape=(3,), dtype=int32)

with tf.Session() as sess:
    result = sess.run(c1)
    print (result)


[ 8  2 10]
```



Numpy and TensorFlow

```
import numpy as np

a = np.array([5,3,8])

b = np.array([3, -1, 2])

c = np.add(a,b)

print(c)
[ 8  2 10]
```

```
import tensorflow as tf

a1 = tf.constant([5,3,8])

b1 = tf.constant([3,-1,2])

c1 = tf.add(a1,b1)

print (c1)
Tensor("Add_1:0", shape=(3,), dtype=int32)

with tf.Session() as sess:
    result = sess.run(c1)
    print (result)


[ 8  2 10]
```

Lazy Evaluation Model

- Two step process
  - Create the DAG (Graph)
  - Run the graph

- This concept is similar to C++ language
  - Write code and compile the code
  - Execute the code

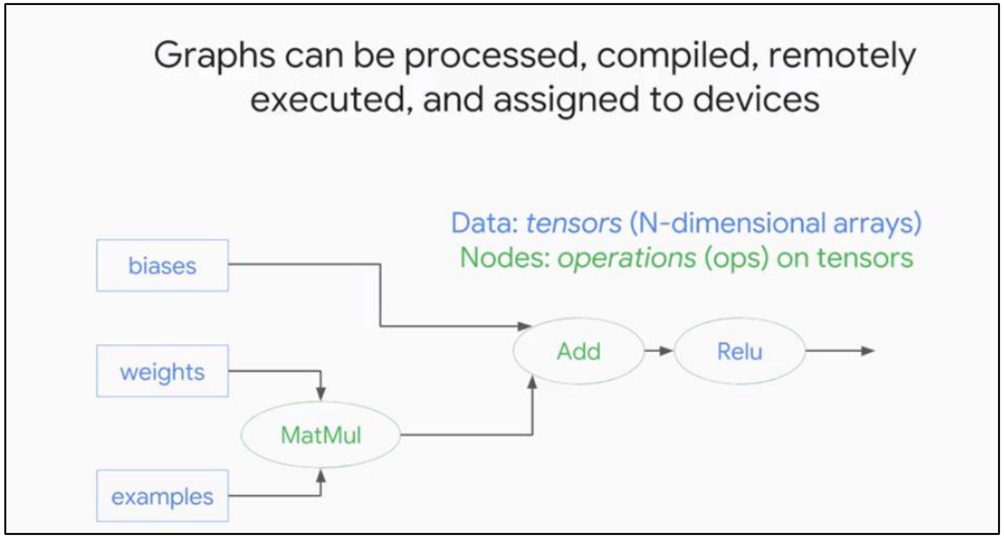
‘tf.eager’ mode

- The TensorFlow statement is executed immediately

5.1 DAG:
Directed Acyclic Graph

Directed Acyclic Graph (DAG)

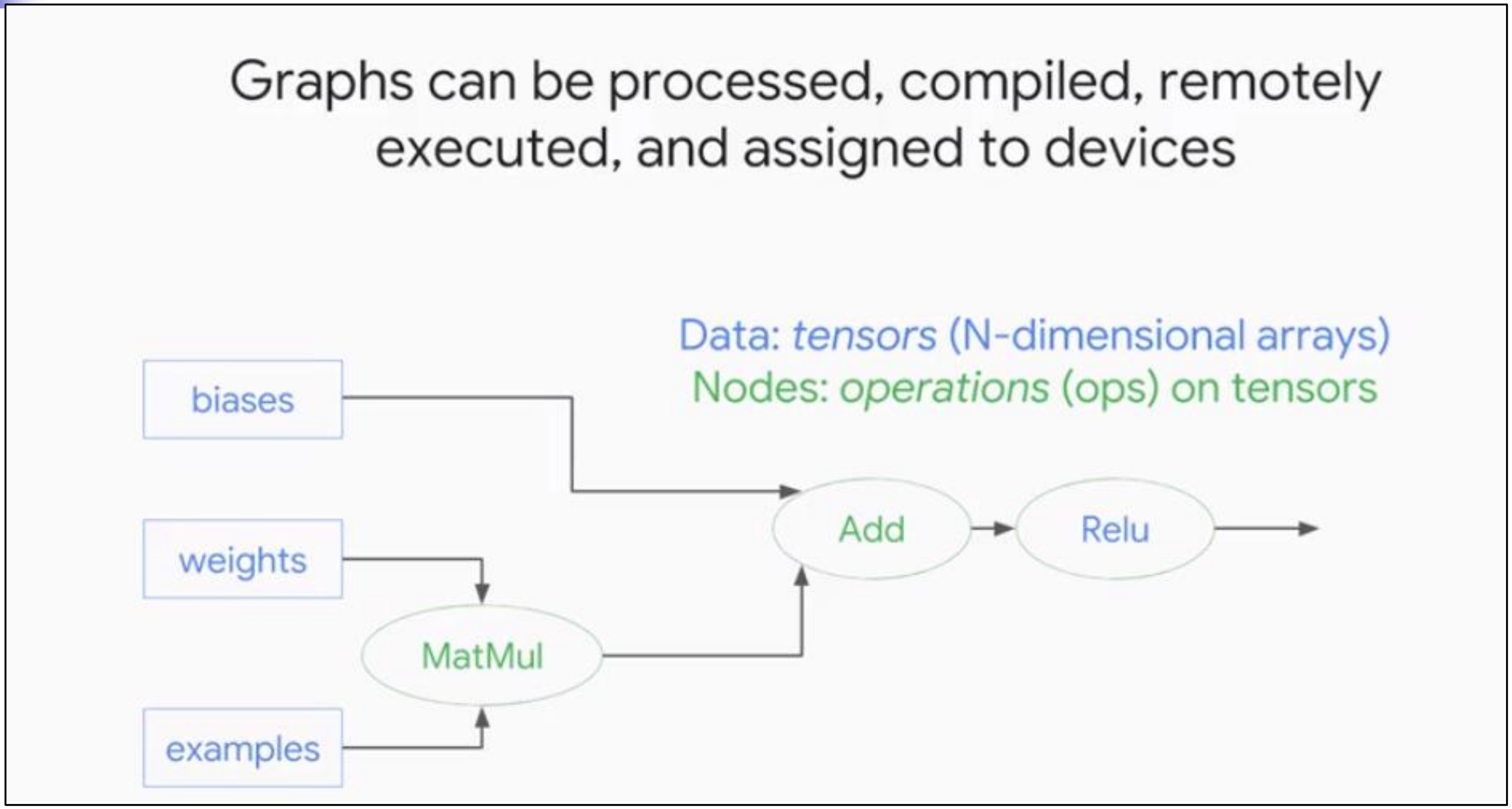
- Edges (Boxes)
  - Data as Tensors (n-dimensional arrays)
- Nodes
  - Tensor Flow Operations
  - Example tf.add



Why Lazy Evaluation?

- Code optimization
  - Take 2 add operation and fuse them into one for faster execution
- Flexibility of Parallel Execution
  - Different part of the graph can be sent different hardware devices for execution in parallel

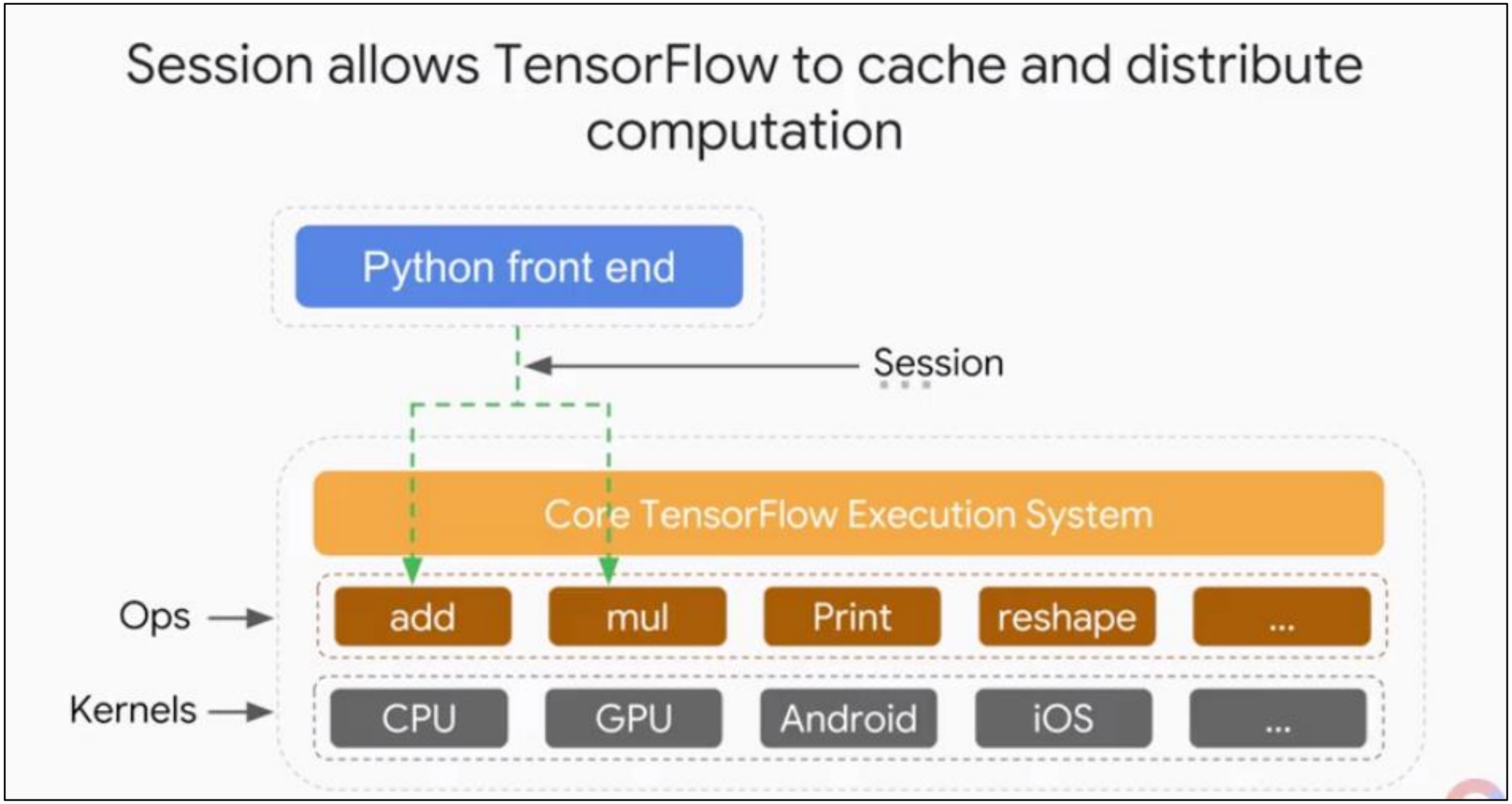
DAG


Graphs can be processed, compiled, remotely executed, and assigned to devices
Data: tensors (N-dimensional arrays)
Nodes: operations (ops) on tensors
biases
weights
examples
MatMul
Add
Relu

DAG

- DAG can be remotely executed and assigned to devices

Parallel Processing

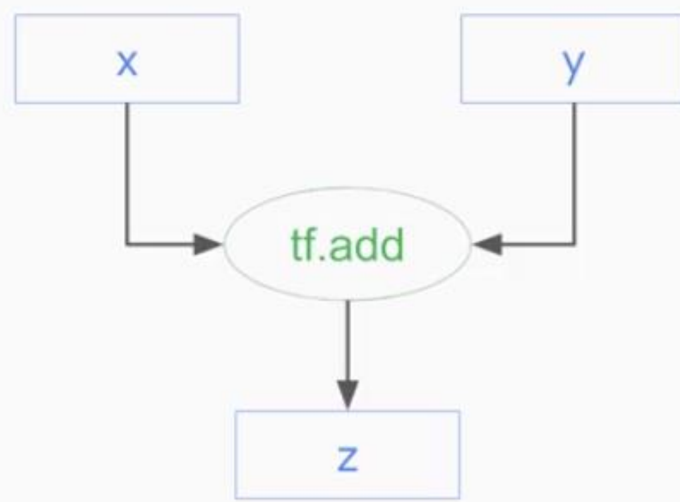


Python Execution

Execute TensorFlow graphs by calling run()
on a tf.Session

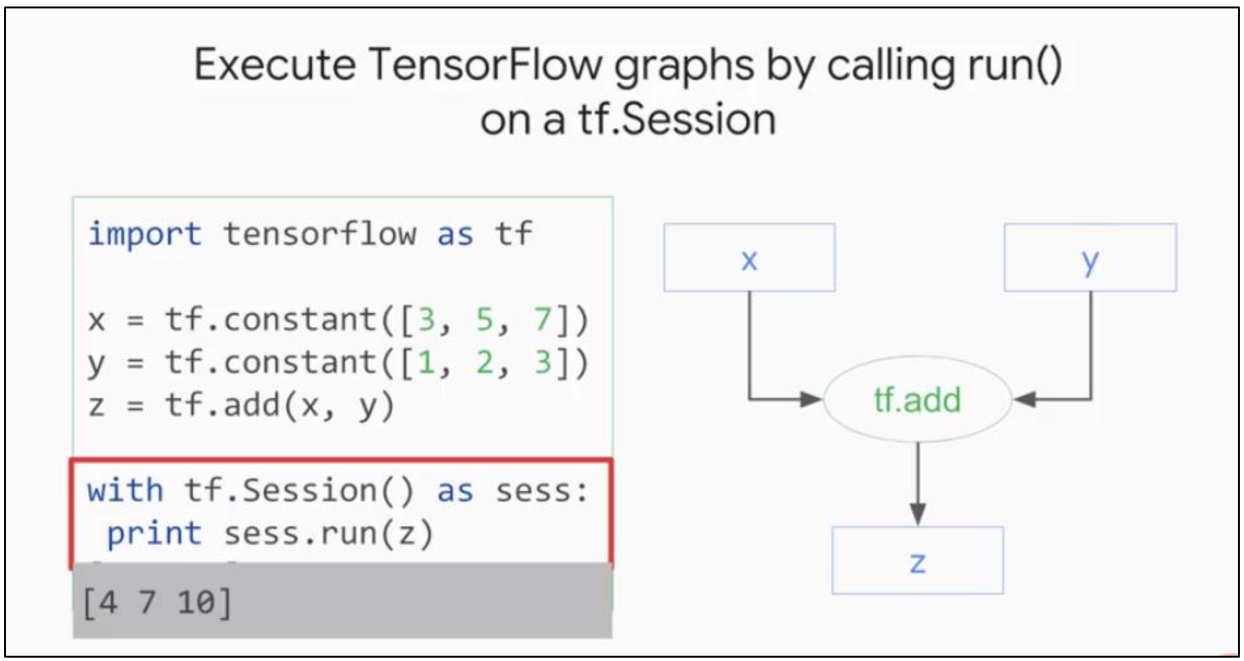
```
import tensorflow as tf

x = tf.constant([3, 5, 7])
y = tf.constant([1, 2, 3])
z = tf.add(x, y)

with tf.Session() as sess:
 print sess.run(z)
```

```
[4 7 10]
```



Python Execution

```
import tensorflow as tf
a1 = tf.constant([3,5,7])
b1 = tf.constant([1,2,3])
c1 = tf.add(a1,b1)

print (c1)
Tensor("Add:0", shape=(3,), dtype=int32)

with tf.Session() as sess:
    result = sess.run(c1)
    print (result)


[ 4  7 10]
```



5.2 Evaluating a Tensor

Evaluating Single Tensor

```
import tensorflow as tf

x = tf.constant([3,5,7])
y = tf.constant([1,2,3])
z = tf.add(x,y)
print (z)

with tf.Session() as sess:
    print( z.eval() )


Tensor("Add_1:0", shape=(3,), dtype=int32)
[ 4  7 10]
```

Evaluating A List of Tensors

```
import tensorflow as tf

x = tf.constant([3,5,7])
y = tf.constant([1,2,3])

z1 = tf.add(x,y)
z2 = x*y
z3 = z2 - z1

with tf.Session() as sess:
    a1, a3 = sess.run([z1,z3])
    print (a1)
    print (a3)


[ 4  7 10]
[-1  3 11]
```

$$z1 = x + y = [4\ \ 7\ \ 10]$$
$$z2 = x * y = [3\ \ 10\ \ 21]$$
$$z3 = z2 - z1 = [-1\ \ 3\ \ 11]$$

Eager Mode
Execution is done immediately

- Eager mode has to be entered at the beginning of the program

```
import tensorflow as tf

from tensorflow.contrib.eager.python import tfe

tfe.enable_eager_execution()

x = tf.constant([3,5,7])

y = tf.constant([1,2,3])

print(x - y)
tf.Tensor([2 3 4], shape=(3,), dtype=int32)
```

Eager Mode has to be Entered at the Start of the Program

```
import tensorflow as tf
x = tf.constant([3,5,7])
y = tf.constant([1,2,3])
z1 = tf.add(x,y)
z2 = x*y
z3 = z2 - z1
with tf.Session() as sess:
    a1, a3 = sess.run([z1,z3])
    print (a1)
    print (a3)
[ 4  7 10]
[-1  3 11]
###########################################################
from tensorflow.contrib.eager.python import tfe
Traceback (most recent call last):
  File "<ipython-input-10-cf53c6a364bc>", line 1, in <module>
    tfe.enable_eager_execution()
  File "C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\framework\ops.py",
line 5468, in enable_eager_execution
    "tf.enable_eager_execution must be called at program startup.")

ValueError: tf.enable_eager_execution must be called at program startup.
```

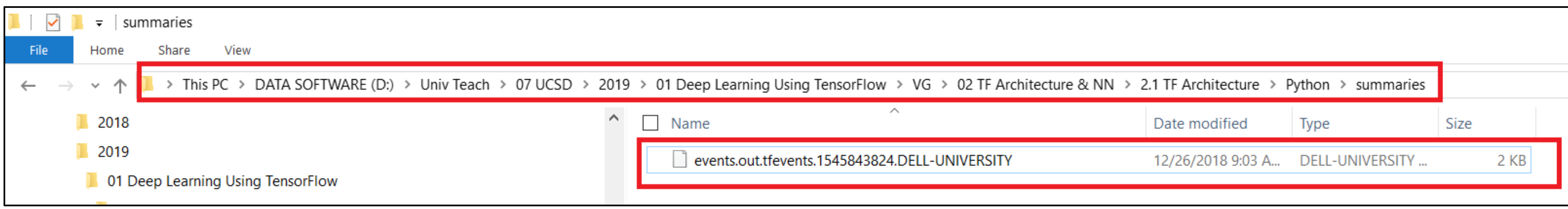
5.3 Visualizing a Graph (DAG)

Directed Acyclic Graph (DAG)

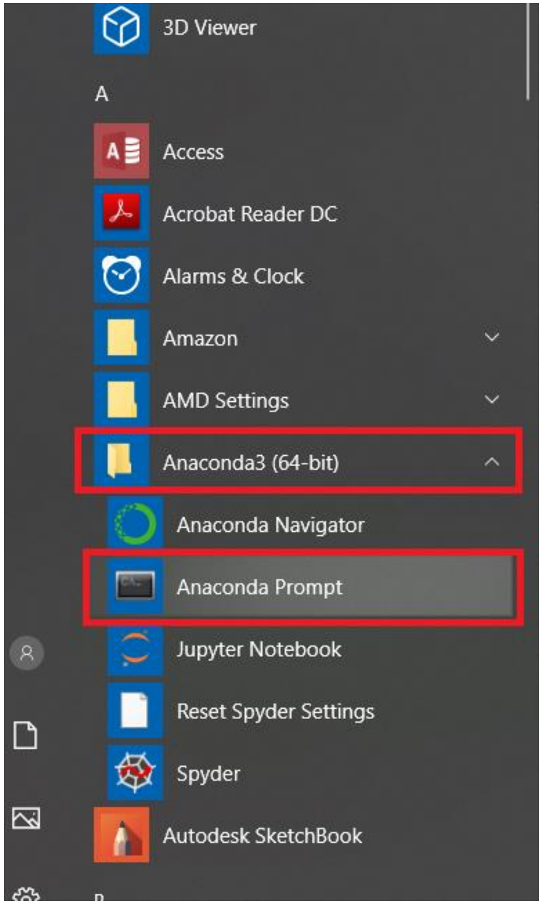
Visualize a Graph

```
import tensorflow as tf

x = tf.constant([3,5,7],name="x")
y = tf.constant([1,2,3],name="y")
z1 = tf.add(x,y,name="z1")
z2 = x*y
z3 = z2 - z1

with tf.Session() as sess:
    with tf.summary.FileWriter("summaries", sess.graph) as writer:
        a1, a3 = sess.run([z1,z3])
```

Event File is Created Under the Folder ‘summaries’



Start “Anaconda Prompt”



Navigate to ‘summaries’ folder

Anaconda Prompt

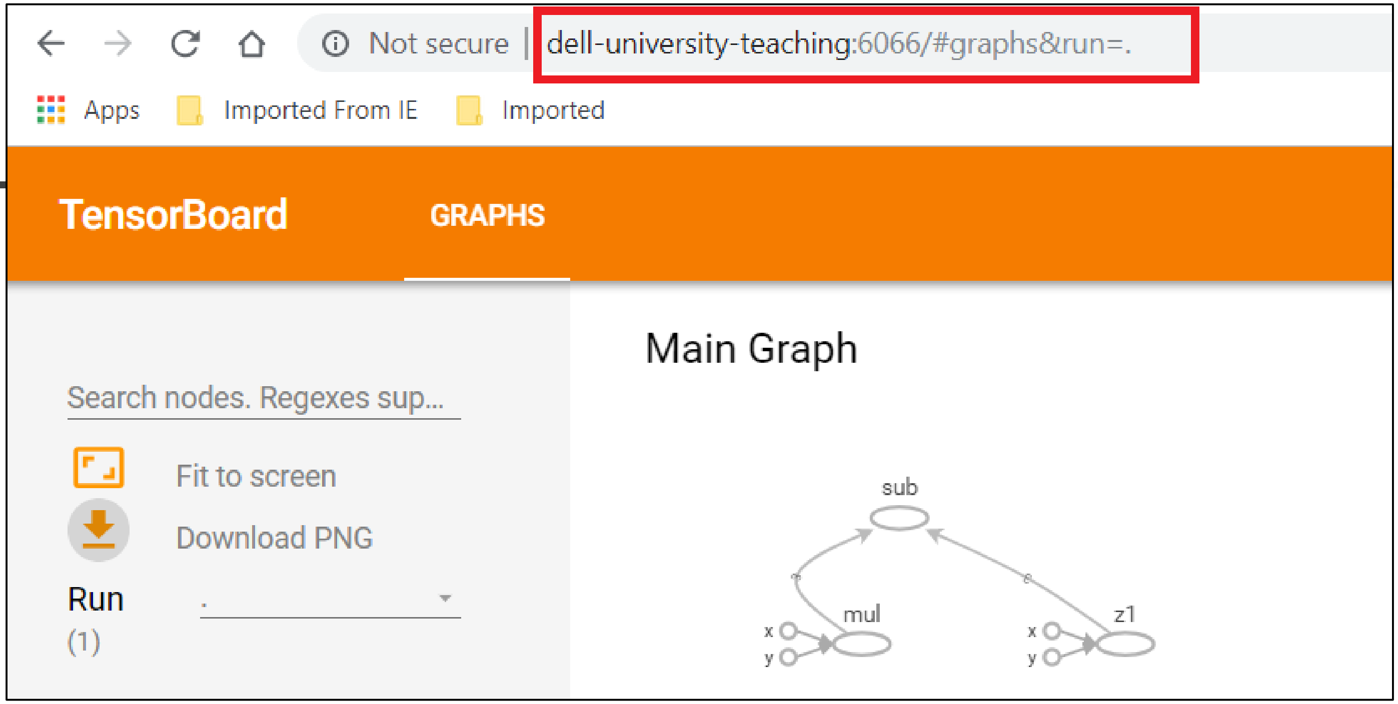
```
(base) C:\Users\ash>d:

(base) D:\>cd D:\"Univ Teach"\"07 UCSD"\2019\"01 Deep Learning Using TensorFlow"\VG\"02 TF Architecture & NN"\"2.1 TF Ar
chitecture"\Python

(base) D:\Univ Teach\07 UCSD\2019\01 Deep Learning Using TensorFlow\VG\02 TF Architecture & NN\2.1 TF Architecture\Pytho
n>dir
 Volume in drive D is DATA SOFTWARE
 Volume Serial Number is 24FF-B379

 Directory of D:\Univ Teach\07 UCSD\2019\01 Deep Learning Using TensorFlow\VG\02 TF Architecture & NN\2.1 TF Architectur
e\Python

12/25/2018  11:02 AM    <DIR>          .
12/25/2018  11:02 AM    <DIR>          ..
10/19/2018  07:51 PM               433 4.1 numpy + tensorFlow.py
10/20/2018  08:39 PM               432 5.1 numpy + tensorFlow.py
10/21/2018  02:15 PM               250 5.21 tensorFlow single tensor.py
10/21/2018  03:19 PM               305 5.22 tensorFlow many tensors.py
10/21/2018  03:42 PM               268 5.23 tensorFlow - eager.py
10/21/2018  04:24 PM               459 5.31 visualize graph.py
12/25/2018  04:51 PM    <DIR>          event logs new
12/26/2018  09:06 AM    <DIR>          summaries
12/25/2018  11:02 AM               372 visulize DAG Using TensorBoard.py
               7 File(s)          2,519 bytes
               4 Dir(s)  1,257,087,827,968 bytes free

(base) D:\Univ Teach\07 UCSD\2019\01 Deep Learning Using TensorFlow\VG\02 TF Architecture & NN\2.1 TF Architecture\Pytho
n>
```

Start ‘tensorboard’

- Start ‘tensorboard’ from Anaconda prompt
  - “tensorboard --logdir=./summaries --port 6066”
- Tensor board will provide a URL
  - http://Dell-University-Teaching:6066
  - Using your browser, navigate to that URL

```
n>
(base) D:\Univ Teach\07 UCSD\2019\01 Deep Learning Using TensorFlow\VG\02 TF Architecture & NN\2.1 TF Architecture\Pytho
n>tensorboard --logdir=./summaries --port 6066
TensorBoard 1.9.0 at http://Dell-University-Teaching:6066 (Press CTRL+C to quit)
```

DAG of the Computation




```
import tensorflow as tf

x = tf.constant([3,5,7],name="x")
y = tf.constant([1,2,3],name="y")
z1 = tf.add(x,y,name="z1")
z2 = x*y
z3 = z2 - z1

with tf.Session() as sess:
    with tf.summary.FileWriter("summaries", sess.graph) as writer:
        a1, a3 = sess.run([z1,z3])
```

6.1 Creating Tensors

What is a Tensor?

- A Tensor is a n-dimensional data
- A Tensor has a shape (dimensions)

Examples of Tensors

A tensor is an N-dimensional array of data

Name	Rank	Example	Shape
Scalar	0	x0 = tf.constant(3)	()
Vector	1	x1 = tf.constant([3,5,7])	(3,)
Matrix	2	x2 = tf.constant([[3,5,7],[4,6,8]])	(2,3)
3D Tensor	3	x3 = tf.constant([[[3,5,7],[4,6,8]], [[1,2,3],[4,5,6]]])	(2,2,3)
nD Tensor	n	x1 = tf.constant([2,3,4]) x2 = tf.stack([x1, x1]) x3 = tf.stack([x2, x2, x2, x2]) x4 = tf.stack([x3, x3])	(3,) (2,3) (4,2,3) (2,4,2,3)

Creating Rank 0,1 Tensors

Name	Rank	Example	Shape
Scalar	0	x0 = tf.constant(3)	()
Vector	1	x1 = tf.constant([3,5,7])	(3,)

```
import tensorflow as tf
####################################
# Scalar    Rank = 0
#
x0 = tf.constant(3)
x0
Out[6]: <tf.Tensor 'Const:0' shape=() dtype=int32>
x0.shape
Out[7]: TensorShape([])

####################################
# Vector    Rank = 1
#
x1 = tf.constant([3,5,7])
x1
Out[12]: <tf.Tensor 'Const_1:0' shape=(3,) dtype=int32>
x1.shape
Out[13]: TensorShape([Dimension(3)])
```

Creating Rank 2,3 Tensors

Name	Rank	Example	Shape
Scalar	0	x0 = tf.constant(3)	()
Vector	1	x1 = tf.constant([3,5,7])	(3,)
Matrix	2	x2 = tf.constant([[3,5,7],[4,6,8]])	(2,3)
3D Tensor	3	x3 = tf.constant([[[3,5,7],[4,6,8]], [[1,2,3],[4,5,6]]])	(2,2,3)

```
x2 = tf.constant([[3,5,7],[4,6,8]])
x2
Out[18]: <tf.Tensor 'Const_2:0' shape=(2, 3) dtype=int32>
x2.shape
Out[19]: TensorShape([Dimension(2), Dimension(3)])

####################################
# 3D Tensor    Rank = 3
#
x3 = tf.constant([  [  [3,5,7],[4,6,8]  ], [  [1,2,3],[4,5,6]  ]   ])
x3
Out[24]: <tf.Tensor 'Const_3:0' shape=(2, 2, 3) dtype=int32>
x3.shape
Out[25]: TensorShape([Dimension(2), Dimension(2), Dimension(3)])
```

Using 'Stack'

Creating NEW Bigger Tensors Using OLD Tensors

Name	Rank	Example	Shape
nD Tensor	n	x1 = tf.constant([2,3,4]) x2 = tf.stack([x1, x1]) x3 = tf.stack([x2, x2, x2, x2]) x4 = tf.stack([x3, x3])	(3,) (2,3) (4,2,3) (2,4,2,3)

```
####################################
# nD Tensor
x1 = tf.constant([2,3,4])
x1.shape
Out[30]: TensorShape([Dimension(3)])

x2 = tf.stack([x1, x1])
x2.shape
Out[32]: TensorShape([Dimension(2), Dimension(3)])

x3 = tf.stack([x2, x2, x2, x2])
x3.shape
Out[34]: TensorShape([Dimension(4), Dimension(2), Dimension(3)])

x4 = tf.stack([x3, x3])
x4.shape
Out[36]: TensorShape([Dimension(2), Dimension(4), Dimension(2), Dimension(3)])
```

Using ‘Slice’

Create NEW Smaller Tensors Using OLD Tensors

	Column 0	Column 1	Column 2
Row 0	3	5	7
Row 1	4	6	8

All Rows & First Column

```
import tensorflow as tf

x = tf.constant([[3,5,7],[4,6,8]])

y = x[:,1]   # Slicing

with tf.Session() as sess:
    print ( y.eval() )


[5 6]
```

First Row & All Columns

```
y1 = x[1,:]    # Slicing

with tf.Session() as sess:
    print ( y1.eval() )


[4 6 8]
```

First Row & 1st + 2nd Columns

```
y2 = x[1,0:2]    # Slicing

with tf.Session() as sess:
    print ( y2.eval() )


[4 6]
```

Reshape will Change the Shape of a Tensor

- A Tensor
  - Shaped (2,3) is
  - re-shaped to (3,2)

- Combine
  - Reshaping +
  - Slicing

```
import tensorflow as tf

y = tf.constant([[3,5,7],[4,6,8]])

y1 = tf.reshape(x,[3,2])
with tf.Session() as sess:
    print ( y1.eval() )


[[3 5]
 [7 4]
 [6 8]]

################################
y2 = tf.reshape(x,[3,2])[1,:]

with tf.Session() as sess:
    print ( y2.eval() )


[7 4]
```

6.2 Variables and Place Holders

What is a Variable?

- A variable is a tensor
  - Whose value is initialized
  - And then the value gets changed as a program runs

Function

- A function is defined 'forward_pass'
  - Multiplies 2 tensors

```
def forward_pass(w,x):
    return tf.matmul(w,x)
```

$$A * B = C$$

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}$$

This operation is valid
if number of columns of A is equal to
the number of rows of B

A Matrix dimension = m x p
B Matrix dimension = p x q
C Matrix dimension = m x q

Variable

```
with tf.variable_scope("model",reuse=tf.AUTO_REUSE):
        w = tf.get_variable("weights",
                            shape=(1,2),
                            initializer=tf.truncated_normal_initializer(),
                            trainable=True)
```

- Create a variable using 'get_variable' function
- Name of the variable 'w' = weights
- Shape =(1,2) "1 row, 2 columns"
- Initialize = Gaussian Random Normal Distribution with truncates numbers at some multiple of sigma
- Trainable = True, can be changed during training

A Simple TensorFlow Program

```
import tensorflow as tf

def forward_pass(w,x):
    return tf.matmul(w,x)

def train_loop(x, niter=10):
    with tf.variable_scope("model",reuse=tf.AUTO_REUSE):
        w = tf.get_variable("weights",
                            shape=(1,2),
                            initializer=tf.truncated_normal_initializer(),
                            trainable=True)
        preds = []
        for k in range(niter):
            preds.append(forward_pass(w,x))
            w = w + 0.1 #gradient update
        return preds
```

Variable ‘w’

Random1	Random2

1 x 2 matrix

Parameter ‘x’

X11	X12	X13
x21	x22	x23

2 x 3 matrix

=

Result = preds = w*x

Result1	Result2	Result3

1 x 3 matrix

Run the TensorFlow Code

```
with tf.Session() as sess:
    x = tf.constant( [[3.2, 5.1, 7.2],[4.3, 6.2, 8.3]])
    print(x.eval())
    preds = train_loop(x)
    tf.global_variables_initializer().run()
    for i in range(len(preds)):
        print("{}:{}".format(i,preds[i].eval()))

[[ 3.20000005  5.0999999   7.19999981]
 [ 4.30000019  6.19999981  8.30000019]]
0:[[ 0.99251807  1.5858233   2.24158144]]
1:[[ 1.74251819  2.71582317  3.79158163]]
2:[[ 2.49251795  3.84582305  5.34158134]]
3:[[ 3.24251819  4.9758234   6.89158154]]
4:[[ 3.99251819  6.10582304  8.44158173]]
5:[[ 4.74251842  7.23582268  9.99158192]]
6:[[  5.49251842   8.36582375  11.54158211]]
7:[[  6.24251842   9.49582386  13.0915823 ]]
8:[[  6.99251842  10.62582397  14.64158249]]
9:[[  7.74251842  11.75582314  16.19158173]]
```

Variable 'w'

Random1	Random2

1 x 2 matrix

Parameter 'x'

3.2	5.1	7.2
4.3	6.2	8.3

2 x 3 matrix

=

Result = preds = w*x

Result1	Result2	Result3

1 x 3 matrix

Place Holder

- Placeholders allow you to feed in values into a graph

```
import tensorflow as tf

a = tf.placeholder("float",None)

b = a*4

print(a)
Tensor("Placeholder:0", dtype=float32)

with tf.Session() as session:
    print(session.run(b,feed_dict={a:[1,2,3]}))


[  4.   8.  12.]
```

Summary

- 1. What is TensorFlow
- 2. History of TensorFlow
- 3. Advantages of Directed Acyclic Graph (DAG)
- 4.1 TensorFlow API Hierarchy
- 4.2 Mode of Execution: Lazy & Eager
- 5.1 DAG: Directed Acyclic Graph
- 5.2 Evaluating a Tensor
- 5.3 Visualizing a Graph (DAG)
- 6.1 Creating Tensors
- 6.2 Variables and Place Holders