Numpy

Dr. Mongkol Ekpanyapong

Create Numpy Array

```
import numpy as np

a = np.array([0,1,2,3]) # a vector
b = np.array([4,5,6,7]) # another vector
c = np.array([[0,1,2,3],# a matrix
            [4,5,6,7]])

d = np.zeros((2,4))#(2x4 matrix of zeros)
e = np.random.rand(2,5) # random 2x5
# matrix with all numbers between 0 and 1

print a
print b
print c
print d
print e
```

Output

```
[0 1 2 3]
[4 5 6 7]
[[0 1 2 3]
 [4 5 6 7]]
[[ 0. 0. 0. 0.]
 [ 0. 0. 0. 0.]]
[[ 0.22717119 0.39712632
0.0627734 0.08431724
0.53469141]
 [ 0.09675954 0.99012254
0.45922775 0.3273326
0.28617742]]
```

Numpy Operation

```
print a * 0.1 # multiplies every number in vector "a" by 0.1
print c * 0.2 # multiplies every number in matrix "c" by 0.2
print a * b # multiplies elementwise between a and b (columns paired up)
print a * b * 0.2 # elementwise multiplication then multiplied by 0.2
print a * c # since c has the same number of columns as a, this performs
# elementwise multiplication on every row of the matrix "c"

print a * e # since a and e don't have the same number of columns, this
# throws a "Value Error: operands could not be broadcast together with.."
```

Numpy Shape

- Always keep track of the shape

	Output

<code>a = np.zeros((1,4)) # vector of length 4
b = np.zeros((4,3)) # matrix with 4 rows & 3
columns
c = a.dot(b)
print c.shape</code>
	(1,3)

Example

Define you own matrix: m = np.array([[0,1,2,3]])

```
(a,b).dot(b,c) = (a,c)
```

```
a = np.zeros((2,4)) # matrix with 2 rows and 4 columns
b = np.zeros((4,3)) # matrix with 4 rows & 3 columns

c = a.dot(b)
print c.shape # outputs (2,3)


e = np.zeros((2,1)) # matrix with 2 rows and 1 columns
f = np.zeros((1,3)) # matrix with 1 row & 3 columns

g = e.dot(f)
print g.shape # outputs (2,3)
```

this ".T" "flips" the rows and columns of a matrix

```
h = np.zeros((5,4)).T # matrix with 4 rows and 5 columns
i = np.zeros((5,6)) # matrix with 6 rows & 5 columns

j = h.dot(i)
print j.shape # outputs (4,6)


h = np.zeros((5,4)) # matrix with 5 rows and 4 columns
i = np.zeros((5,6)) # matrix with 5 rows & 6 columns
j = h.dot(i)
print j.shape # throws an error
```

Tensorflow

Dr. Mongkol Ekpanyapong

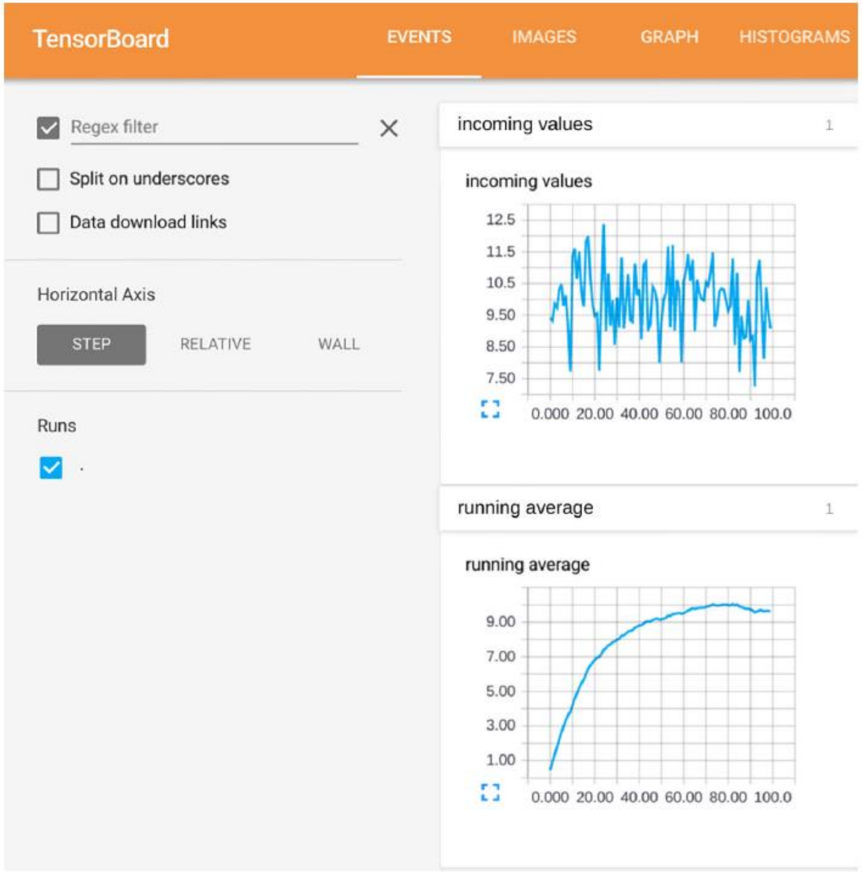
Tensorflow History

- Google open-sourced its machine learning framework in 2015 under the Apache 2.0 license
- Before that, Google uses its in speech recognition, Search, Photos, and Gmail
- A former learning system called DistBelief is the primary influence on TensorFlow
- The library is implemented in C++ and have both Python and C++ API

Tensorflow Features

- Automatic differentiation capabilities: you can experiment with new networks without having to redefine many key calculations (esp. back-propagation)
- TensorBoard shows a flowchart of the way data transforms, displays summary log over time, and traces performance

TensorBoard example



Computing Inner Product

Manually:

```
revenue = 0
for price, amount in zip(prices, amounts):
    revenue += price * amount
```

Using Numpy:

```
import numpy as np
revenue = np.dot(prices, amounts)
```

TensorFlow Library

To call the library:

```
import tensorflow as tf
```

What is a tensor?

- A tensor is a generalization of a matrix that specifies an element by an arbitrary number

Example

- Let say you are the principle, and you want to assign seating for all students in a school

The school has multiple classrooms, each classroom has a row and column. You can specify classroom 2, row 4, column 10 as

(2,4,10) => this will be a rank-3 tensor

Tensor Representation

row

2,3,2

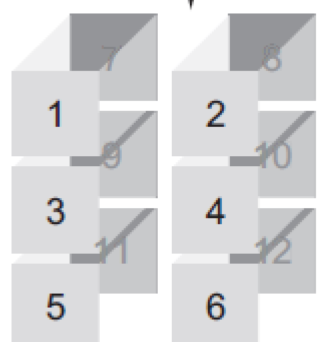
column

dimension

How a tensor is represented in code

[[[1, 2], [3, 4], [5, 6]] [[7, 8], [9, 10], [11, 12]]]

How we visualize a tensor



Tensor Representation in Python

- m1 is a list
- m2 is ndarray

In numpy

- m3 is Tensorflow

constant

```
import tensorflow as tf
import numpy as np

m1 = [[1.0, 2.0],
      [3.0, 4.0]]

m2 = np.array([[1.0, 2.0],
               [3.0, 4.0]], dtype=np.float32)

m3 = tf.constant([[1.0, 2.0],
                  [3.0, 4.0]])

print(type(m1))
print(type(m2))
print(type(m3))

t1 = tf.convert_to_tensor(m1, dtype=tf.float32)
t2 = tf.convert_to_tensor(m2, dtype=tf.float32)
t3 = tf.convert_to_tensor(m3, dtype=tf.float32)

print(type(t1))
print(type(t2))
print(type(t3))
```

You'll use NumPy matrices in TensorFlow.

Defines a 2 × 2 matrix in three ways

Prints the type for each matrix

Creates tensor objects out of the various type

Notice that the types will be the same now.

Creating Tensor Constant

```
import tensorflow as tf

m1 = tf.constant([[1., 2.]])

m2 = tf.constant([[1],
                  [2]])

m3 = tf.constant([ [[1,2],
                    [3,4],
                    [5,6]],
                   [[7,8],
                    [9,10],
                    [11,12]] ])

print(m1)
print(m2)
print(m3)
```

Defines a 1 x 2 matrix

Defines a 2 x 1 matrix

Defines a rank-3 tensor

Try printing the tensors.

Output

```
Tensor( "Const:0",
        shape=TensorShape([Dimension(1), Dimension(2)]),
        dtype=float32 )
Tensor( "Const_1:0",
        shape=TensorShape([Dimension(2), Dimension(1)]),
        dtype=int32 )
Tensor( "Const_2:0",
        shape=TensorShape([Dimension(2), Dimension(3), Dimension(2)]),
        dtype=int32 )
```

Creating 500x500 tensors

- Initilize a 500x500 tensor with all elements equaling 0.5

```
tf.ones([500,500]) * 0.5
```

“Hello World” with TensorFlow

```
import tensorflow as tf
h = tf.constant(“Hello”)
w= tf.constant(“World”)
hw = h + w
with tf.Session() as sess:
        ans = sess.run(hw)

print(ans)
```

Tensor Operations

- Arithmetic operation

`tf.add(x, y)`—Adds two tensors of the same type, *x* + *y*

`tf.subtract(x, y)`—Subtracts tensors of the same type, *x* – *y*

`tf.multiply(x, y)`—Multiplies two tensors element-wise

`tf.pow(x, y)`—Takes the element-wise *x* to the power of *y*

`tf.exp(x)`—Equivalent to *pow*(*e*, *x*), where *e* is Euler's number (2.718 …)

`tf.sqrt(x)`—Equivalent to *pow*(*x*, 0.5)

`tf.div(x, y)`—Takes the element-wise division of *x* and *y*

`tf.truediv(x, y)`—Same as `tf.div`, except casts the arguments as a float

`tf.floordiv(x, y)`—Same as `truediv`, except rounds down the final answer int an integer

`tf.mod(x, y)`—Takes the element-wise remainder from division

- Example

```
import tensorflow as tf

x = tf.constant([[1, 2]])
negMatrix = tf.negative(x)
print(negMatrix)
```

Defines an arbitrary tensor

Negates the tensor

Prints the object

Exercise

- Use TensorFlow to produce Gaussian Distribution (also known as Normal distribution). You can assume mean = 0 and sigma = 1

Hint:

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Answer

```
from math import pi
mean = 0.0
sigma = 1.0
(tf.exp(tf.negative(tf.pow(x - mean, 2.0) /
                   (2.0 * tf.pow(sigma, 2.0) ))) *
 (1.0 / (sigma * tf.sqrt(2.0 * pi) )))
```

Sessions

- A session is an environment of a software that describes how the lines of code should run
- To execute an operation and retrieve its calculated value, TensorFlow requires a session
- To create a session class: we use tf.Session() command
- A session can setup how the hardware devices will run

Example

```
import tensorflow as tf

x = tf.constant([[1., 2.]])
neg_op = tf.negative(x)

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

Defines an arbitrary matrix

Runs the negation operator on it

Starts a session to be able to run operations

Tells the session to evaluate negMatrix

Prints the resulting matrix

eval() function

- Every Tensor object has an eval() function to evaluate the mathematical operations that define its value
- eval() requires defining a session object for the library to understand how to use the underlying hardware
- sess.run(..) is equivalent to invoking the Tensor's eval()

Interactive session mode

- It is often used for debugging, presentation purpose
- It can be used for implicitly call to any eval()

Example

```
import tensorflow as tf
sess = tf.InteractiveSession()

x = tf.constant([[1., 2.]])
negMatrix = tf.negative(x)

result = negMatrix.eval()
print(result)

sess.close()
```

Starts an interactive session so the sess variable no longer needs to be passed around

Defines an arbitrary matrix and negates it

You can now evaluate negMatrix without explicitly specifying a session.

Prints the negated matrix

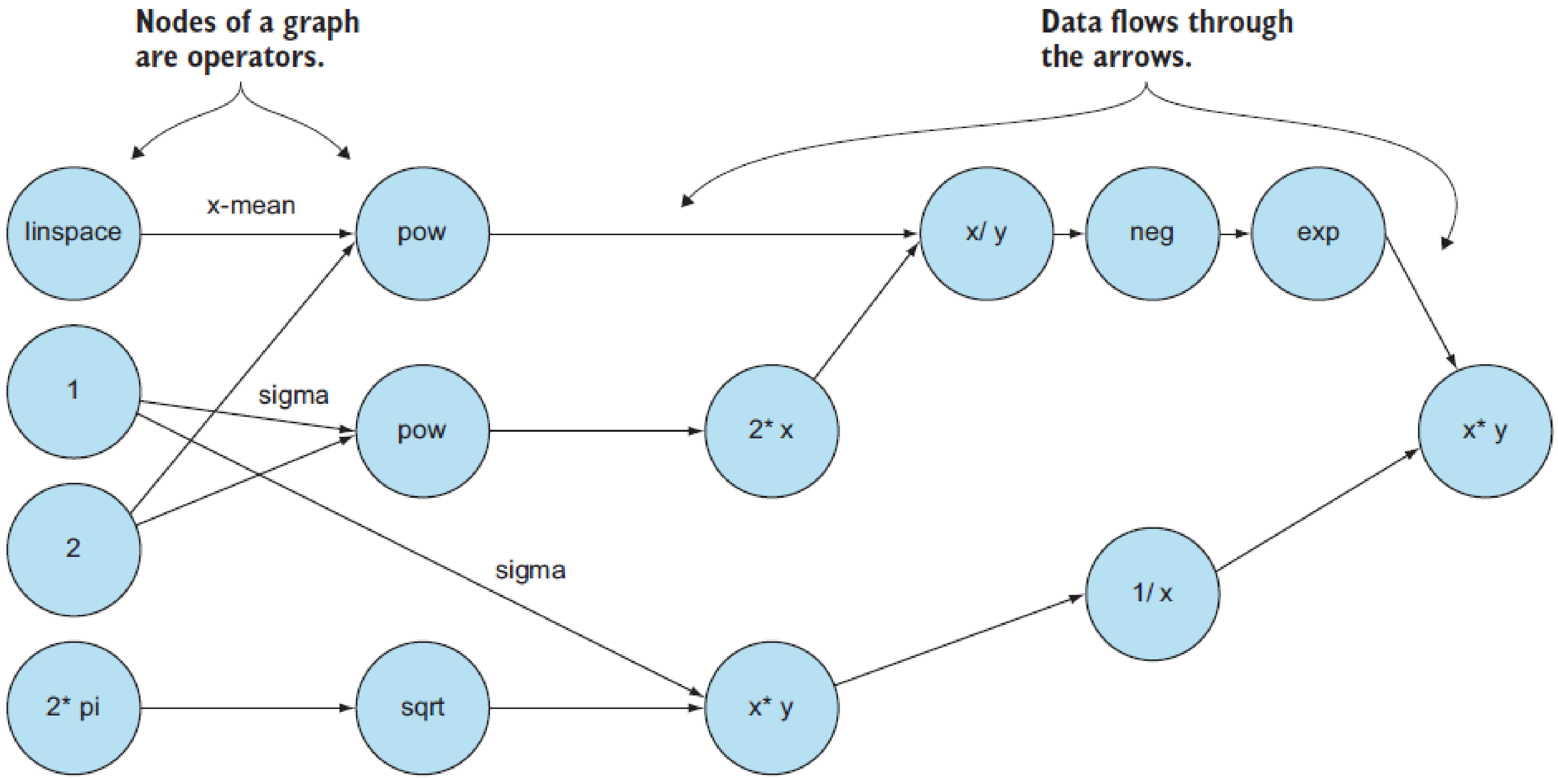
Remember to close the session to free up resources.

Understanding code as a graph

- In TensorFlow graph, nodes of a graph are operators
- Edge represents interaction between nodes
- Data flows through the arrow sign
- The system is strong type meaning the dimension and type has to match
- The technical term is called dataflow graph

and dataflow computing

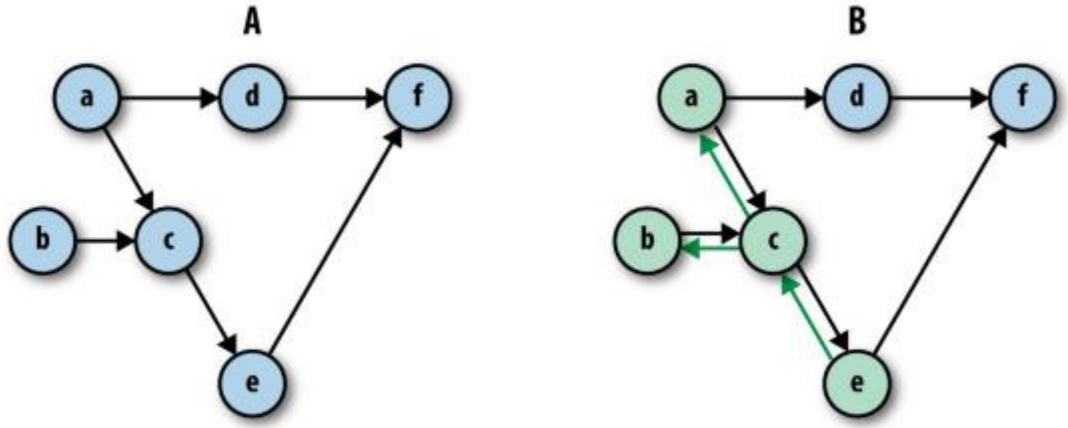
Example of Graph Representation



Dataflow graph

- Nodes/vertices represent an operation such as arithmetic operations, or creating summaries
- Edges allow data to flow in a directed manner
- Direct dependency: when two nodes are connected via an edge
- Indirect dependency: when two nodes are connected via more than one edge

Example



- Node *e* is directly dependent on node *c*
- Node c is directly dependent on node a
- If we have to evaluate node *e*, we need the know to compute only node *a*, *b*, and *c*

TensorFlow Procedure

Working with TensorFlow involves 2 main phases:

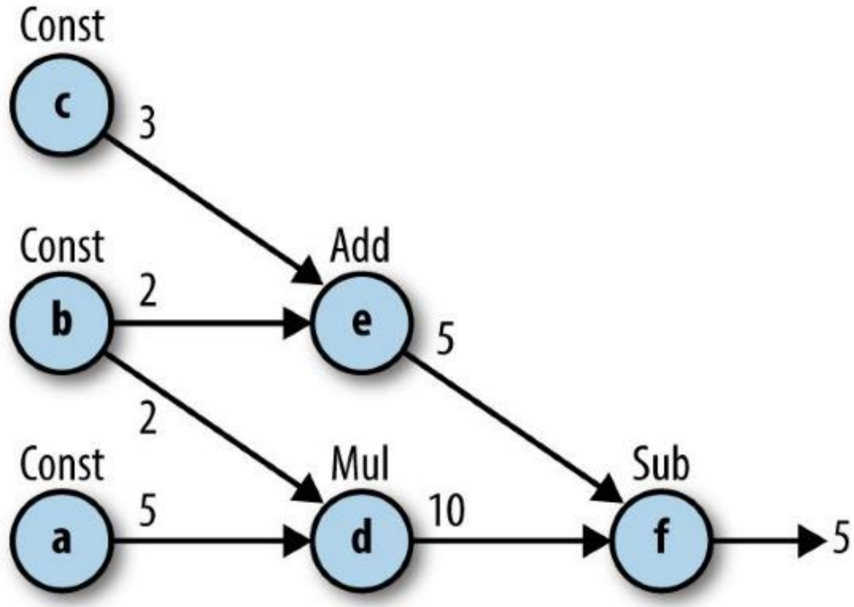
1. Construct a Graph
2. Create Session and Execute it

Construct a Graph

- After we import TensorFlow library, a specific empty default graph is formed
- All the new nodes that we are created are associated with these default graph

Example

```
a = tf.constant(5)
b = tf.constant(2)
c = tf.constant(3)
d = tf.multiply(a,b)
e = tf.add(b,c)
f = tf.subtract(d,e)
```



TensorFlow Operator

TensorFlow operator	Shortcut	Description
`tf.add()`	`a + b`	Adds `a` and `b`, element-wise.
`tf.multiply()`	`a * b`	Multiplies `a` and `b`, element-wise.
`tf.subtract()`	`a - b`	Subtracts `a` from `b`, element-wise.
`tf.divide()`	`a / b`	Computes Python-style division of `a` by `b`.
`tf.pow()`	`a ** b`	Returns the result of raising each element in `a` to its corresponding element `b`, element-wise.
`tf.mod()`	`a % b`	Returns the element-wise modulo.
`tf.logical_and()`	`a & b`	Returns the truth table of `a & b`, element-wise. `dtype` must be `tf.bool`.
`tf.greater()`	`a > b`	Returns the truth table of `a > b`,

TensorFlow Operator

		element-wise.
`tf.greater_equal()`	`a >= b`	Returns the truth table of `a >= b`, element-wise.
`tf.less_equal()`	`a <= b`	Returns the truth table of `a <= b`, element-wise.
`tf.less()`	`a < b`	Returns the truth table of `a < b`, element-wise.
`tf.negative()`	`-a`	Returns the negative value of each element in `a`.
`tf.logical_not()`	`~a`	Returns the logical NOT of each element in `a`. Only compatible with Tensor objects with `dtype` of `tf.bool`.
`tf.abs()`	`abs(a)`	Returns the absolute value of each element in `a`.
`tf.logical_or()`	`a | b`	Returns the truth table of `a | b`, element-wise. `dtype` must be `tf.bool`.

Create Session and Execute

- As shown earlier, the session is required to run the TensorFlow program

```
sess = tf.Session()
outs = sess.run(f)
sess.close()
print("outs = {}".format(outs))
```

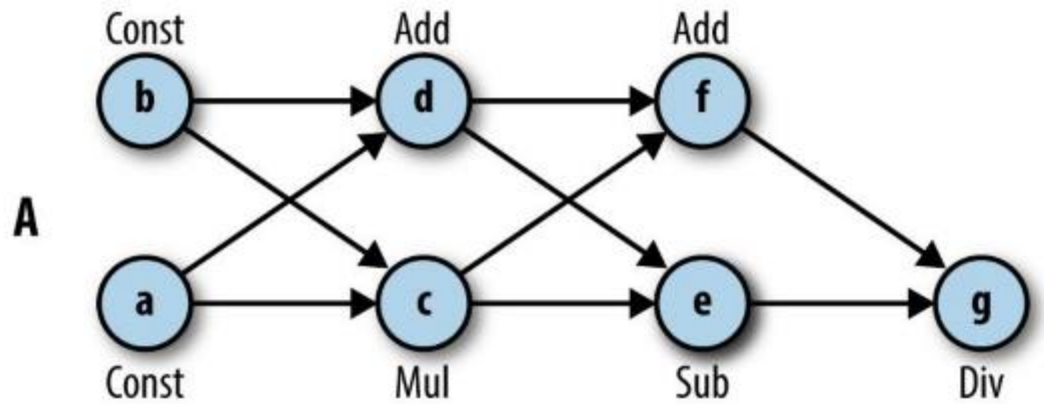
- The execution is through: run or eval commands
- Example of Output:

```
Out:
outs = 5
```

- To make sure that the resources are properly deallocated, use sess.close

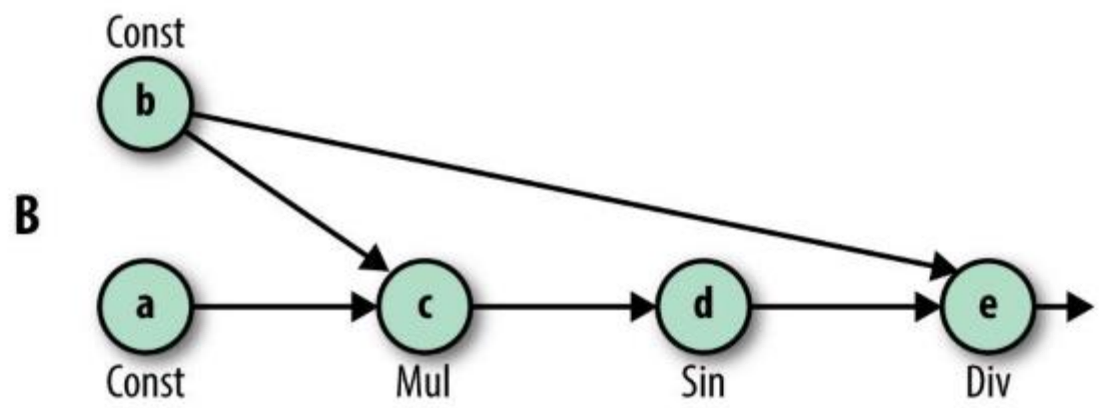
Question?

- Create a TensorFlow program for this:



Question?

- Create a TensorFlow program for this:



Session Configuration

- We can assign which hardware to run
- We can enable log file etc.

Example

```
import tensorflow as tf

x = tf.constant([[1., 2.]])
negMatrix = tf.negative(x)

with tf.Session(config=tf.ConfigProto(log_device_placement=True)) as sess:
    result = sess.run(negMatrix)

print(result)
```

Defines a matrix and negates it

Starts the session with a special config passed into the constructor to enable logging

Evaluates negMatrix

Prints the resulting value

- Output

```
Neg: /job:localhost/replica:0/task:0/cpu:0
```

The task was running in CPU

Manual Device Placement

```
# Creates a graph.
with tf.device('/cpu:0'):
  a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
  b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
c = tf.matmul(a, b)
# Creates a session with log_device_placement set to True.
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
# Runs the op.
print(sess.run(c))
```

Output

Device mapping:

/job:localhost/replica:0/task:0/device:GPU:0 -> device: 0, name: Tesla K40c, pci bus

id: 0000:05:00.0

b: /job:localhost/replica:0/task:0/cpu:0

a: /job:localhost/replica:0/task:0/cpu:0

MatMul: /job:localhost/replica:0/task:0/device:GPU:0

[[22. 28.]

 [49. 64.]]

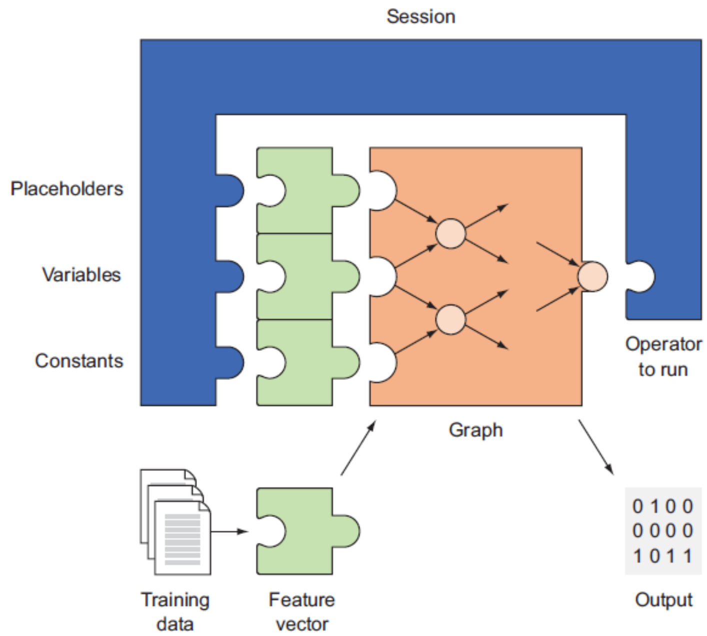
Input of a Session

- Placeholder – a value that is unassigned, it will be initialized by the Session when it is run. Typically, it is the input/output of the model

- Variable – a value that can change such as parameters of machine learning model

- Constant – a value that doesn’t change such as hyper parameter

Input/Output of Session



Constructing and Managing our Graph

- To construct a new graph, we need to use tf.Graph() command

```
import tensorflow as tf
print(tf.get_default_graph())

g = tf.Graph()
print(g)

Out:
<tensorflow.python.framework.ops.Graph object
at 0x7fd88c3c07d0>
<tensorflow.python.framework.ops.Graph object
at 0x7fd88c3c03d0>
```

Graph Association

- We can view the graph associated using <node>.graph

```
g = tf.Graph()
a = tf.constant(5)

print(a.graph is g)
print(a.graph is tf.get_default_graph())

Out:
False
True
```

The With statement

- In Python, we can use with statement together with as_default() to associate node with the graph

```
g1 = tf.get_default_graph()
g2 = tf.Graph()

print(g1 is tf.get_default_graph())

with g2.as_default():
    print(g1 is tf.get_default_graph())

print(g1 is tf.get_default_graph())

Out:
True
False
True
```

Fetches

- In TensorFlow, we only evaluate the node in the graph that we want to know the result. This operation is called fetch
- If we want to evaluate multiple nodes, a list of requested nodes can be used

```
with tf.Session() as sess:
   fetches = [a,b,c,d,e,f]
   outs = sess.run(fetches)

print("outs = {}".format(outs))
print(type(outs[0]))

Out:
outs = [5, 2, 3, 10, 5, 5]
<type 'numpy.int32'>
```

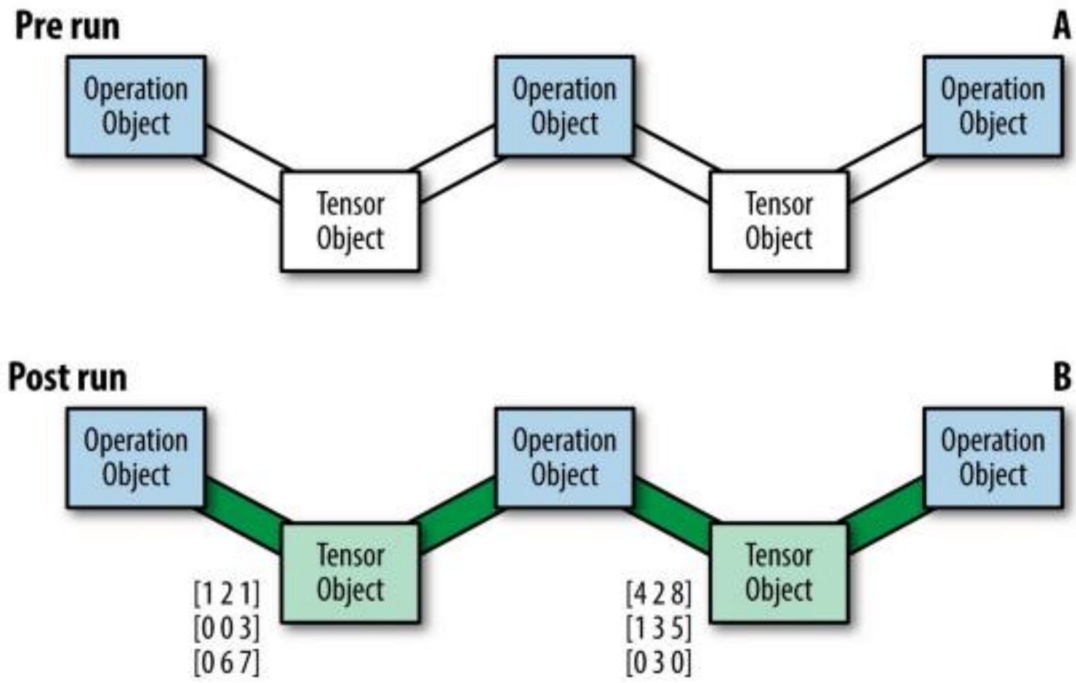
- With the fetch, we can execute only portion of the graph

Flow

- When we construct a node in the graph, we are creating an operation instance
- These operations do not produce actual values until the graph is executed
- This is where the name TensorFlow comes from

Example of Graph Construction and Flow Execution

- Pre run is graph construction, post run is the flow execution



Data Types

- If no data type provided, TensorFlow will select data type automatically

- User can explicitly define the data type, he/she wants to use

```
c = tf.constant(4.0, dtype=tf.float64)
print(c)
print(c.dtype)

Out:
Tensor("Const_10:0", shape=(), dtype=float64)
<dtype: 'float64'>
```

Castings

- It is important to make sure data type match throughout the graph
- Performing an operation on mismatch data will result in exception

```
x =
tf.constant([1,2,3],name='x',dtype=tf.float32)
print(x.dtype)
x = tf.cast(x,tf.int64)
print(x.dtype)

Out:
<dtype: 'float32'>
<dtype: 'int64'>
```

Support Tensor Data Types

Data type	Python type	Description
DT_FLOAT	tf.float32	32-bit floating point.
DT_DOUBLE	tf.float64	64-bit floating point.
DT_INT8	tf.int8	8-bit signed integer.
DT_INT16	tf.int16	16-bit signed integer.
DT_INT32	tf.int32	32-bit signed integer.
DT_INT64	tf.int64	64-bit signed integer.
DT_UINT8	tf.uint8	8-bit unsigned integer.

Support Tensor Data Types

DT_UINT16	tf.uint16	16-bit unsigned integer.
DT_STRING	tf.string	Variable-length byte array. Each element of a Tensor is a byte array.
DT_BOOL	tf.bool	Boolean.
DT_COMPLEX64	tf.complex64	Complex number made of two 32-bit floating points: real and imaginary parts.
DT_COMPLEX128	tf.complex128	Complex number made of two 64-bit floating points: real and imaginary parts.
DT_QINT8	tf.qint8	8-bit signed integer used in quantized ops.
DT_QINT32	tf.qint32	32-bit signed integer used in quantized ops.
DT_QUINT8	tf.quint8	8-bit unsigned integer used in quantized ops.

TensorFlow Name

- Each TensorFlow object has an identifying name
- This is not the same as variable name
- We can use .name attribute to see the name of the object
- The objects with the same graph cannot have the same name. TensorFlow will automatically rename by adding _ and a number
- The number after the colon of the name object is Tensor index

Example

```
with tf.Graph().as default():
    c1 =
tf.constant(4,dtype=tf.float64,name='c')
    c2 =
tf.constant(4,dtype=tf.int32,name='c')
print(c1.name)
print(c2.name)
```

```
Out:
c:0
c_1:0
```

Name Scope

- The name scope prefix can be used to add hierarchical group
- The command tf.namescope(“prefix”) is used
- The name scope can be helped for graphics visualization

Example

```
with tf.Graph().as_default():
    c1 =
tf.constant(4,dtype=tf.float64,name='c')
    with tf.name_scope("prefix_name"):
        c2 =
tf.constant(4,dtype=tf.int32,name='c')
        c3 =
tf.constant(4,dtype=tf.float64,name='c')

print(c1.name)
print(c2.name)
print(c3.name)

Out:
c:0
prefix_name/c:0
prefix_name/c_1:0
```

- prefix_name is the name scoped used here

Tensor Arrays and Shapes

- TensorFlow is tightly associated with NumPy
- The array in Numpy can be converted to TensorFlow object
- The get_shape() object can return the shape of a tensor

TensorFlow Constant

- Can accept scalar
- Tightly integrate with Numpy library

Example

```
import numpy as np

c = tf.constant([[1,2,3],
                 [4,5,6]])
print("Python List input:
{}".format(c.get_shape()))

c = tf.constant(np.array([
                 [[1,2,3],

                [[1,1,1],
                 [2,2,2]]
                ]))

print("3d NumPy array input:
{}".format(c.get_shape()))

Out:
Python list input: (2, 3)
3d NumPy array input: (2, 2, 3)
```

TensorFlow Variables

- Variable class represents a node whose value changes over time
- It is also called parameters
- A machine-learning algorithm updates the parameters of a model until it finds the optimal value for each variable

TensorFlow Variables

- Variables in TensorFlow maintains a fixed state in the graph
- To create variable, we call the tf.Variable() function. We can also set the initial value
- To run the session, we have to create memory and set its initial value using tf.global_variables_initializer()
- The Tensor variables will be computed only when session is run

Example

```
init_val = tf.random_normal((1,5),0,1)
var = tf.Variable(init_val, name='var')
print("pre run: \n{}".format(var))

init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    post_var = sess.run(var)

print("\npost run: \n{}".format(post_var))

Out:
pre run:
Tensor("var/read:0", shape=(1, 5),
dtype=float32)

post run:
[[ 0.85962135  0.64885855  0.25370994
-0.37380791  0.63552463]]
```

Example

- Note that if we run the session again, a new variable is created

```
pre run:
Tensor("var_1/read:0", shape=(1, 5),
dtype=float32)
```

- To reuse the variable, we have to use tf.getvariables() instead of tf.Variable()

Example

Let's say you have some raw data like this.

```
import tensorflow as tf
sess = tf.InteractiveSession()
```

Starts the session in interactive mode so you won't need to pass around sess

```
raw_data = [1., 2., 8., -1., 0., 5.5, 6., 13]
spike = tf.Variable(False)
spike.initializer.run()
```

Creates a Boolean variable called spike to detect a sudden increase in a series of numbers

Because all variables must be initialized, initialize the variable by calling run() on its initializer.

```
for i in range(1, len(raw_data)):
    if raw_data[i] - raw_data[i-1] > 5:
        updater = tf.assign(spike, True)
        updater.eval()
    else:
        tf.assign(spike, False).eval()
    print("Spike", spike.eval())
```

To update a variable, assign it a new value using tf.assign(<var name>, <new value>). Evaluate it to see the change.

```
sess.close()
```

Remember to close the session after it'll no longer be used.

Loops through the data (skipping the first element) and updates the spike variable when there's a significant increase

Loading and Saving Variables

- In machine-learning, saving and loading data at known checkpoints makes it much easier to debug code

- restore and save commands are used

Saving variables

Defines a Boolean vector called spikes to locate a sudden spike in raw data

Imports TensorFlow and enables interactive sessions

```
import tensorflow as tf
sess = tf.InteractiveSession()

raw_data = [1., 2., 8., -1., 0., 5.5, 6., 13]
spikes = tf.Variable([False] * len(raw_data), name='spikes')
spikes.initializer.run()

saver = tf.train.Saver()

for i in range(1, len(raw_data)):
    if raw_data[i] - raw_data[i-1] > 5:
        spikes_val = spikes.eval()
        spikes_val[i] = True
        updater = tf.assign(spikes, spikes_val)
        updater.eval()

save_path = saver.save(sess, "spikes.ckpt")
print("spikes data saved in file: %s" % save_path)

sess.close()
```

Let's say you have a series of data like this.

Don't forget to initialize the variable.

Loop through the data and update the spikes variable when there's a significant increase.

Updates the value of spikes by using the tf.assign function

Don't forget to evaluate the updater; otherwise, spikes won't be updated.

Saves the variable to disk

Prints out the relative file path of the saved variables

The saver op will enable saving and restoring variables. If no dictionary is passed into the constructor, then it saves all variables in the current program.

Loading variables

```
import tensorflow as tf
sess = tf.InteractiveSession()

spikes = tf.Variable([False]*8, name='spikes')
# spikes.initializer.run()
saver = tf.train.Saver()

saver.restore(sess, "./spikes.ckpt")
print(spikes.eval())

sess.close()
```

Creates a variable of the same size and name as the saved data

You no longer need to initialize this variable because it'll be directly loaded.

Creates the saver op to restore saved data

Restores data from the spikes.ckpt file

Prints the loaded data

Saving Variable (Newer TensorFlow version)

```
import tensorflow as tf
w1 = tf.Variable(tf.random_normal(shape=[2]), name='w1')
w2 = tf.Variable(tf.random_normal(shape=[5]), name='w2')
saver = tf.train.Saver([w1,w2])
sess = tf.Session()
sess.run(tf.global_variables_initializer())
saver.save(sess, './my_test_model',global_step=1000)
```

Load Variable (Newer TensorFlow version)

```
import tensorflow as tf
w1 = tf.Variable(tf.random_normal(shape=[2]), name='w1')
w2 = tf.Variable(tf.random_normal(shape=[5]), name='w2')

saver = tf.train.Saver([w1,w2])

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    new_saver = tf.train.import_meta_graph('my_test_model-1000.meta')
    new_saver.restore(sess, tf.train.latest_checkpoint('./'))
    print(sess.run('w1:0'))
```

TensorFlow PlaceHolder

- Placeholder is used to feed in the input value
- Placeholder can be thought as empty variable in which data will be filled in later
- We can have Placeholder with the shape to be any size (None for dimension)

```
ph = tf.placeholder(tf.float32,shape=(None,10))
```

```
import tensorflow as tf
import numpy as np

x_data = np.random.randn(5,10)
w_data = np.random.randn(10,1)
print("x ",x_data,"\n")
print(" w ",w_data)

x = tf.placeholder(tf.float32, shape =(5,10))
w = tf.placeholder(tf.float32, shape =(10,1))
b = tf.fill((5,1),-1.)
xw = tf.matmul(x,w)
xwb = xw + b
s = tf.reduce_max(xwb)

with tf.Session() as sess:
    outs = sess.run(s,feed_dict={x:x_data,w:w_data})
    print(" outs = {}".format(outs))
```

TensorFlow Operation

TensorFlow operation	Description
`tf.constant(value)`	Creates a tensor populated with the value or values specified by the argument `value`
`tf.fill(shape, value)`	Creates a tensor of shape `shape` and fills it with `value`
`tf.zeros(shape)`	Returns a tensor of shape `shape` with all elements set to 0
`tf.zeros_like(tensor)`	Returns a tensor of the same type and shape as `tensor` with all elements set to 0
`tf.ones(shape)`	Returns a tensor of shape `shape` with all elements set to 1
`tf.ones_like(tensor)`	Returns a tensor of the same type and shape as `tensor` with all elements set to 1
`tf.random_normal(shape, mean, stddev)`	Outputs random values from a normal distribution
`tf.truncated_normal(shape, mean, stddev)`	Outputs random values from a truncated normal distribution (values whose magnitude
`tf.random_uniform(shape, minval, maxval)`	Generates values from a uniform distribution in the range `[minval, maxval)`
`tf.random_shuffle(tensor)`	Randomly shuffles a tensor along its first dimension

TensorFlow Application

- Moving Average: try to compute the estimated average as a function of the previous estimated average and the current value

$$Avg_t = f(Avg_{t-1}, x_t) = (1 - \alpha)\ Avg_{t-1} + \alpha\ x_t$$

Python Code

- Compute moving average

```
update_avg = alpha * curr_value + (1 - alpha) * prev_avg
```

alpha is a tf.constant, curr_value is a placeholder, and prev_avg is a variable.

- Setup a session

```
raw_data = np.random.normal(10, 1, 100)

with tf.Session() as sess:
    for i in range(len(raw_data)):
        curr_avg = sess.run(update_avg, feed_dict={curr_value:raw_data[i]}
        sess.run(tf.assign(prev_avg, curr_avg))
```

Can u write a Python code?

The complete program

```
import tensorflow as tf
import numpy as np

raw_data = np.random.normal(10, 1, 100)

alpha = tf.constant(0.05)
curr_value = tf.placeholder(tf.float32)
prev_avg = tf.Variable(0.)
update_avg = alpha * curr_value + (1 - alpha) * prev_avg

init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)
    for i in range(len(raw_data)):
        curr_avg = sess.run(update_avg, feed_dict={curr_value: raw_data[i]})
        sess.run(tf.assign(prev_avg, curr_avg))
        print(raw_data[i], curr_avg)
```

Creates a vector of 100 numbers with a mean of 10 and standard deviation of 1

Defines alpha as a constant

Initializes the previous average to zero

Loops through the data one by one to update the average

A placeholder is just like a variable, but the value is injected from the session.

Visualizing the data

- Pick up which nodes you care about measuring by annotating with a summary op
- Call add_summary to queue up data

```
img = tf.placeholder(tf.float32, [None, None, None, 3])
cost = tf.reduce_sum(...)

my_img_summary = tf.summary.image("img", img)
my_cost_summary = tf.summary.scalar("cost", cost)
```

TensorBoard

- Make a directory called logs

```
$ mkdir logs
```

- Run TensorBoard with the location of the logs

```
$ tensorboard --logdir=./logs
```

```
import tensorflow as tf
import numpy as np

raw_data = np.random.normal(10, 1, 100)

alpha = tf.constant(0.05)
curr_value = tf.placeholder(tf.float32)
prev_avg = tf.Variable(0.)
update_avg = alpha * curr_value + (1 - alpha) * prev_avg

avg_hist = tf.summary.scalar("running_average", update_avg)
value_hist = tf.summary.scalar("incoming_values", curr_value)
merged = tf.summary.merge_all()
writer = tf.summary.FileWriter("./logs")
init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)
    sess.add_graph(sess.graph)
    for i in range(len(raw_data)):
        summary_str, curr_avg = sess.run([merged, update_avg],
            feed_dict={curr_value: raw_data[i]})
        sess.run(tf.assign(prev_avg, curr_avg))
        print(raw_data[i], curr_avg)
        writer.add_summary(summary_str, i)
```

Creates a summary node for the averages

Creates a summary node for the values

Merges the summaries to make it easier to run all at once

Passes in the logs directory's location to the writer

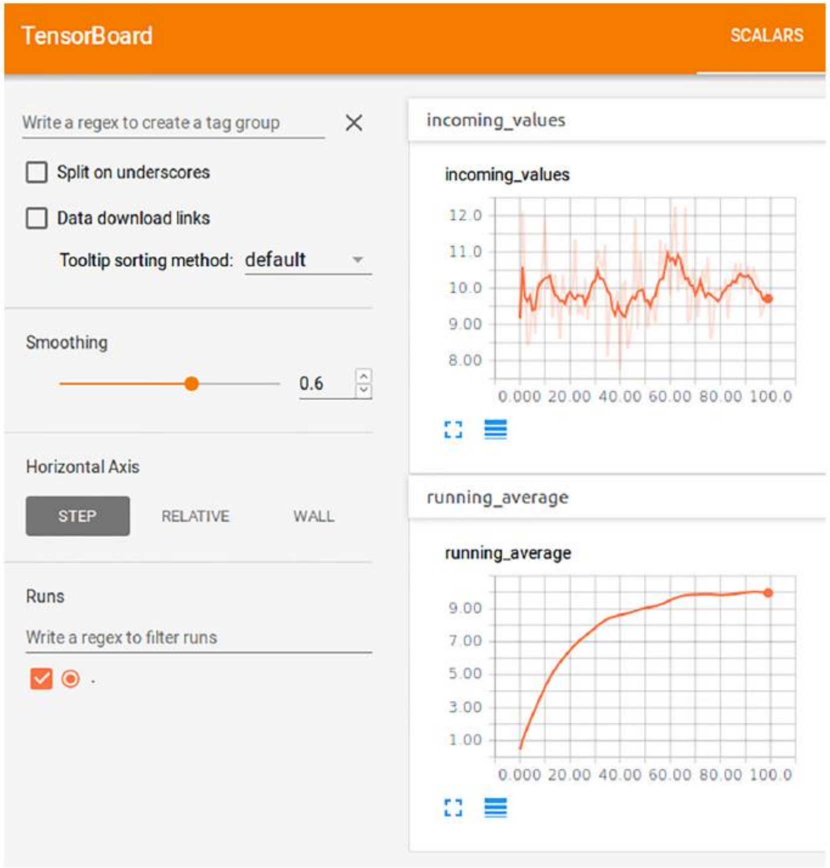
Optional, but allows you to visualize the computation graph in TensorBoard

Runs the merged op and the update_avg op at the same time

Adds the summary to the writer

Output

- Open web browser: http://localhost:6006



Linear regression

- Regression model between target y and input x

$$f(x_i) = w^T x_i + b$$

$$y_i = f(x_i) + \varepsilon_i$$

$f(x_i)$ is assumed to be linear combination of weight w and input x_i with bias b

ε_i is the noise

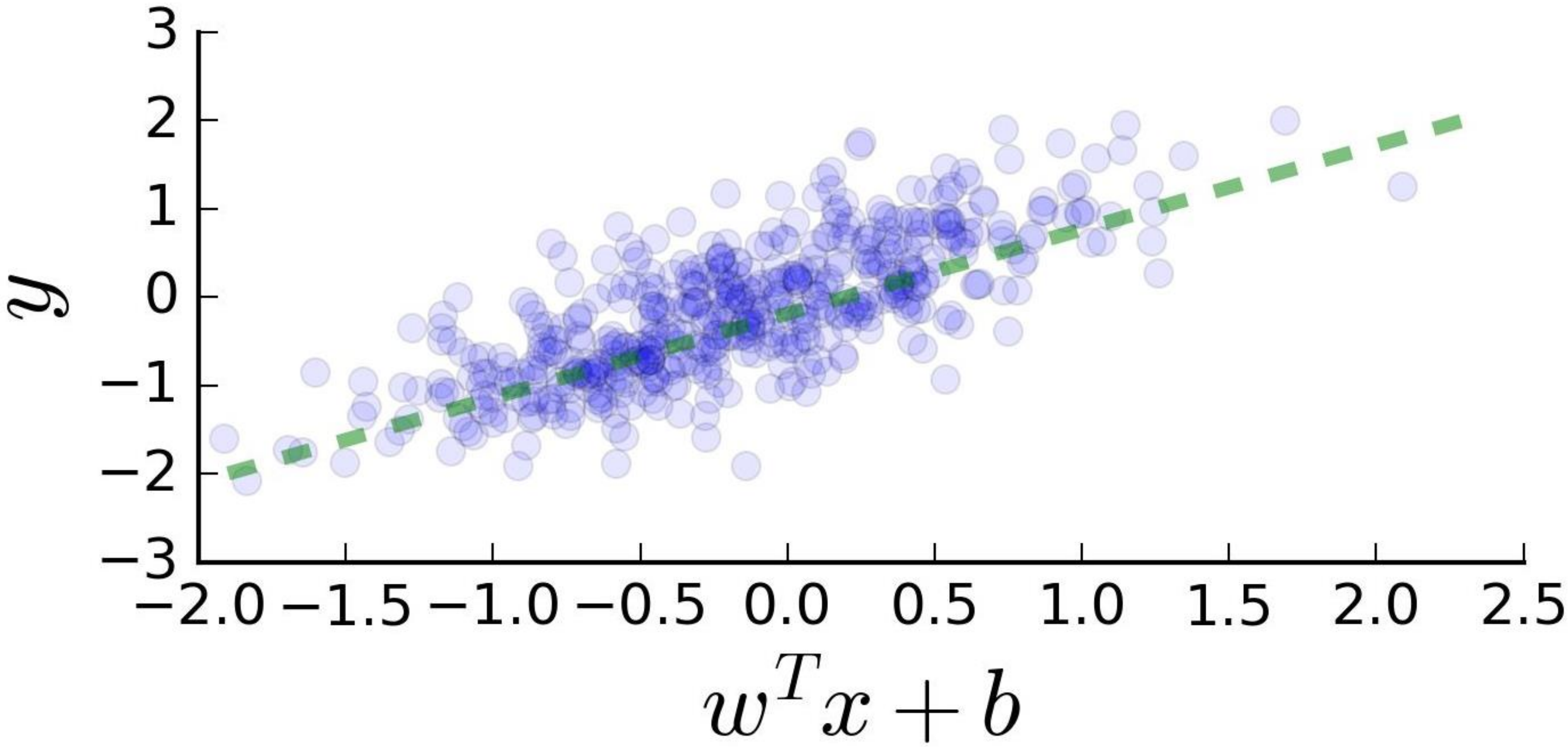
- We want to find weight w and bias b

Example

```
import numpy as np
# === Create data and simulate results =====
x_data = np.random.randn(2000,3)
w_real = [0.3,0.5,0.1]
b_real = -0.2

noise = np.random.randn(1,2000)*0.1
y_data = np.matmul(w_real,x_data.T) + b_real + noise
```

Example of the output



TensorFlow Model

```
x = tf.placeholder( tf.float32, shape =[ None, 3])
y_true = tf.placeholder( tf.float32, shape = None)
w = tf.Variable([[ 0,0,0]], dtype = tf.float32, name =' weights')

y_pred = tf.matmul( w, tf.transpose( x)) + b
```

Loss Function

- Distance metric that we discuss earlier

- The most popular one is mean square error using this equation:

$$L(y, \hat{y}) = \frac{1}{n}\Sigma_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- Python code

```
loss = tf.reduce_mean( tf.square( y_true-y_pred))
```

Gradient Descent Optimization

- Gradient Descent optimization

  Search for local optimization

- TensorFlow program:

```
optimizer = tf.train.GradientDescentOptimizer(
learning_rate)
train = optimizer.minimize( loss)
```

```python
NUM_STEPS = 10

g = tf.Graph()
wb_ = []
with g.as_default():
    x = tf.placeholder(tf.float32,shape=[None,3])
    y_true = tf.placeholder(tf.float32,shape=None)

    with tf.name_scope('inference') as scope:
        w = tf.Variable([[0,0,0]],dtype=tf.float32,name='weights')
        b = tf.Variable(0,dtype=tf.float32,name='bias')
        y_pred = tf.matmul(w,tf.transpose(x)) + b

    with tf.name_scope('loss') as scope:
        loss = tf.reduce_mean(tf.square(y_true-y_pred))

    with tf.name_scope('train') as scope:
        learning_rate = 0.5
        optimizer = tf.train.GradientDescentOptimizer(learning_rate)
        train = optimizer.minimize(loss)

    # Before starting, initialize the variables.  We will 'run' this first.
    init = tf.global_variables_initializer()
    with tf.Session() as sess:
        sess.run(init)
        for step in range(NUM_STEPS):
            sess.run(train,{x: x_data, y_true: y_data})

            if (step % 5 == 0):
                print(step, sess.run([w,b]))
                wb_.append(sess.run([w,b]))

        print(10, sess.run([w,b]))
```

Output

```
(0, [array([[ 0.30149955,  0.49303722,  0.11409992]],
                                    dtype=float32), -0.18563795])

(5, [array([[ 0.30094019,  0.49846715,  0.09822173]],
                                    dtype=float32), -0.19780949])

(10, [array([[ 0.30094025,  0.49846718,  0.09822182]],
                                    dtype=float32), -0.19780946])
```

Question?

- Add Tensorboard with these variables:
  - Weight w
  - Bias b
  - Loss

Answer

```
import tensorflow as tf
import numpy as np
x_data = np.random.randn(2000,3)
w_real = [0.3,0.5,0.1]
b_real = -0.2
noise = np.random.randn(1,2000)*0.1
y_data = np.matmul(w_real,x_data.T) + b_real + noise
NUM_STEPS = 10
g = tf.Graph()
wb_ = []
with g.as_default():
    x = tf.placeholder(tf.float32,shape=[None,3])
    y_true = tf.placeholder(tf.float32,shape=None)

with tf.name_scope('inference') as scope:
    w = tf.Variable([[0,0,0]],dtype=tf.float32,name='weights')
    b = tf.Variable(0,dtype=tf.float32,name='bias')
    y_pred = tf.matmul(w,tf.transpose(x)) + b
w0_hist = tf.summary.scalar("weight0", w[0,0])
w1_hist = tf.summary.scalar("weight1", w[0,1])
w2_hist = tf.summary.scalar("weight2", w[0,2])
b_hist = tf.summary.scalar("bias", b)
```

```
with tf.name_scope('loss') as scope:
    loss = tf.reduce_mean(tf.square(y_true-y_pred))
loss_hist = tf.summary.scalar("loss", loss)
merged = tf.summary.merge_all()
writer = tf.summary.FileWriter("./logs6", sess.graph)
with tf.name_scope('train') as scope:
    learning_rate = 0.5
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    train = optimizer.minimize(loss)

init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    for step in range(NUM_STEPS):
        summary_str,out = sess.run([merged,train],{x: x_data, y_true: y_data})
        writer.add_summary(summary_str, step)
        if (step % 5 == 0):
            print(step, sess.run([w,b]))
            wb_.append(sess.run([w,b]))
    print(10, sess.run([w,b]))
```

Keras

Keras

- Keras library is made and maintained by Francois Chollet
- It ran on top of Theano or TensorFlow
- It also provides many modular ANN library

Keras: Deep Learning Models

- Define the model (create a sequential model and add layers)
- Compile the model (include optimize function)
- Fit the model with training data (fit function)
- Make predictions (evaluate and predict function)

Sequential Model

- Sequential type is to add layers

For example:

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential()

model.add(Dense(units=64, input_dim=784))
model.add(Activation('softmax'))
```

Or

```
model = Sequential([
    Dense(64, input_shape=(784,),activation='softmax')
])
```

Dense Layer

- A dense layer is a fully connected layer
- The first argument denotes the number of output units
- The input shape is the size of the Tensor input e.g., 784x64
- Dense() also has an optional argument where we can specify and add an activation function

Learning Configurations

- The .compile() method is used to set the learning configurations
- It has three input arguments
  - The loss function
  - The optimizer
  - The metric function for performance evaluation

```
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

Optimizer

- We can set the optimizer to use in Keras

```
optimizer=keras.optimizers.SGD(lr=0.02, momentum=0.8, nesterov=True))
```

- More details about optimizer will be discussed later

Training the Model

- We use .fit() the data and set the number of epochs and batch size
- We can also set the early stop condition

```
from keras.callbacks import TensorBoard, EarlyStopping, ReduceLROnPlateau

early_stop = EarlyStopping(monitor='val_loss', min_delta=0,
                           patience=10, verbose=0, mode='auto')


model.fit(x_train, y_train, epochs=10, batch_size=64,
          callbacks=[TensorBoard(log_dir='/models/autoencoder',)
          early_stop])
```

Testing the Model

- We use .evaluate() to evaluate the test model performance
- We use .predict() to predict the real results giving the new input

```
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=64)
classes = model.predict(x_test, batch_size=64)
```

Example

```
from keras.models import Sequential
model = Sequential()
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
from keras import models
from keras import layers
network = models.Sequential()
network.add(layers.Dense(512,activation='relu',input_shape=(28*28,)))
network.add(layers.Dense(10,activation='softmax'))
network.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['accuracy'])
```

Example

```
train_images = train_images.reshape((60000,28*28))
train_images = train_images.astype('float32')/255
test_images = test_images.reshape((10000,28*28))
test_images = test_images.astype('float32')/255

from keras.utils import to_categorical
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

network.fit(train_images,train_labels, epochs=5, batch_size=128)
test_loss, test_acc = network.evaluate(test_images, test_labels)
print('test acc',test_acc)
```

Testing the Result

```
train_images.shape
import matplotlib.pyplot as plt
plt.imshow(test_images[0])
plt.show()
class1 = network.predict_classes(test_images[0:1])
print(class1)
```

Questions?