UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

3253 Machine Learning

Module 9: Introduction to TensorFlow

Course Plan

Module Titles
Module 1 – Introduction to Machine Learning
Module 2 – End to End Machine Learning Project
Module 3 – Classification
Module 4 – Clustering and Unsupervised Learning
Module 5 – Training Models and Feature Selection
Module 6 – Support Vector Machines
Module 7 – Decision Trees and Ensemble Learning
Module 8 – Dimensionality Reduction
Module 9 – **Current Focus: Introduction to TensorFlow**
Module 10 – Introduction to Deep Learning and Deep Neural Networks
Module 11 – Distributing TensorFlow, CNNs and RNNs
Module 12 – Final Assignment and Presentations (no content)

Learning Outcomes for this Module

- Begin to use TensorFlow for Machine and Deep Learning models
- Build TensorFlow graphs to organize a computation
- Use tools such as TensorBoard to visualize the computation

Topics for this Module

- **9.1** TensorFlow
- **9.2** TensorFlow graphs
- **9.3** TensorFlow programs
- **9.4** Computing the gradient
- **9.5** TensorBoard
- **9.6** Regression with TensorFlow
- **9.7** Regression with gradient descent
- **9.8** Resources and Wrap-up

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 9 – Section 1

TensorFlow

TensorFlow

- Its main Python API offers much more flexibility to create all sorts of computations
- It includes highly efficient C++ implementations of many ML operations,
- It provides several advanced optimization nodes to search for the parameters that minimize a cost function.
- It also comes with a great visualization tool called *TensorBoard*
- Several other high-level APIs have been built independently on top of TensorFlow, such as Keras, Pretty tensor, tflearn
- Last but not least, it has a dedicated team of passionate and helpful developers, and a growing community contributing to improving it. It is one of the most popular open source projects on GitHub, and more and more great projects are being built on top of it

Other Libraries

Table 9-1. Open source Deep Learning libraries (not an exhaustive list)

Library	API	Platforms	Started by	Year
Caffe	Python, C++, Matlab	Linux, macOS, Windows	Y. Jia, UC Berkeley (BVLC)	2013
Deeplearning4j	Java, Scala, Clojure	Linux, macOS, Windows, Android	A. Gibson, J.Patterson	2014
H2O	Python, R	Linux, macOS, Windows	H2O.ai	2014
MXNet	Python, C++, others	Linux, macOS, Windows, iOS, Android	DMLC	2015
TensorFlow	Python, C++	Linux, macOS, Windows, iOS, Android	Google	2015
Theano	Python	Linux, macOS, iOS	University of Montreal	2010
Torch	C++, Lua	Linux, macOS, iOS, Android	R. Collobert, K. Kavukcuoglu, C. Farabet	2002

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 9 – Section 2

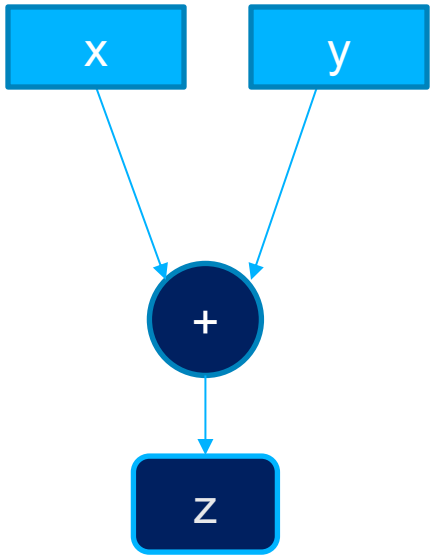
TensorFlow Graphs

Tensorflow Graph Overview

- Tensorflow's primary abstraction is that of a graph
- On nodes of graph are operators, and data flows through graph with operations being applied
- With Tensorflow, we create a graph, and then we execute it within a Session

Defining a Tensorflow Graph

- We create variables to maintain state on the graph



- Say we have variables x and y:

  x = tf.Variable(3, name='x')

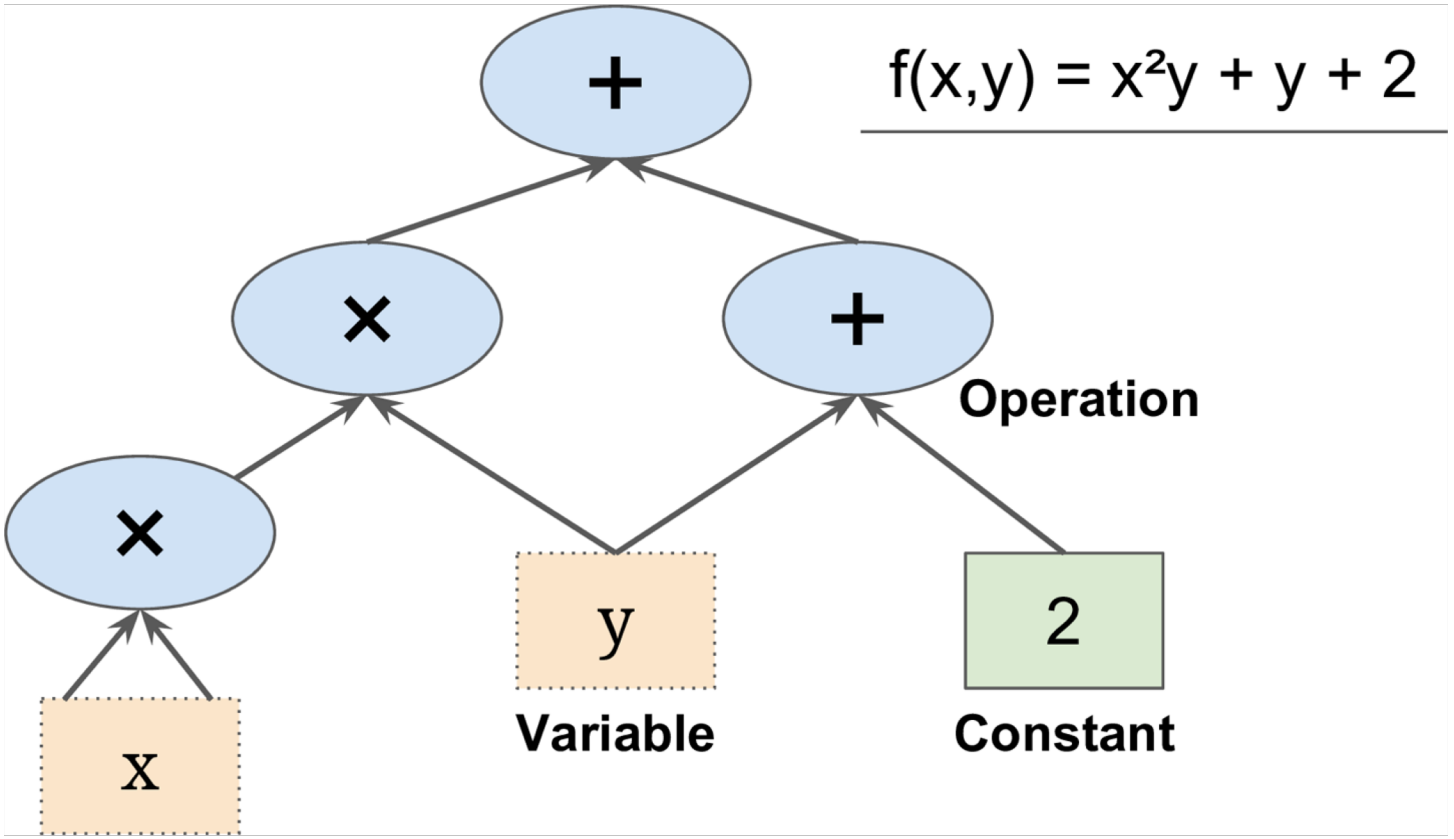
  y = tf.Variable(4, name='y')

- And we define f as:

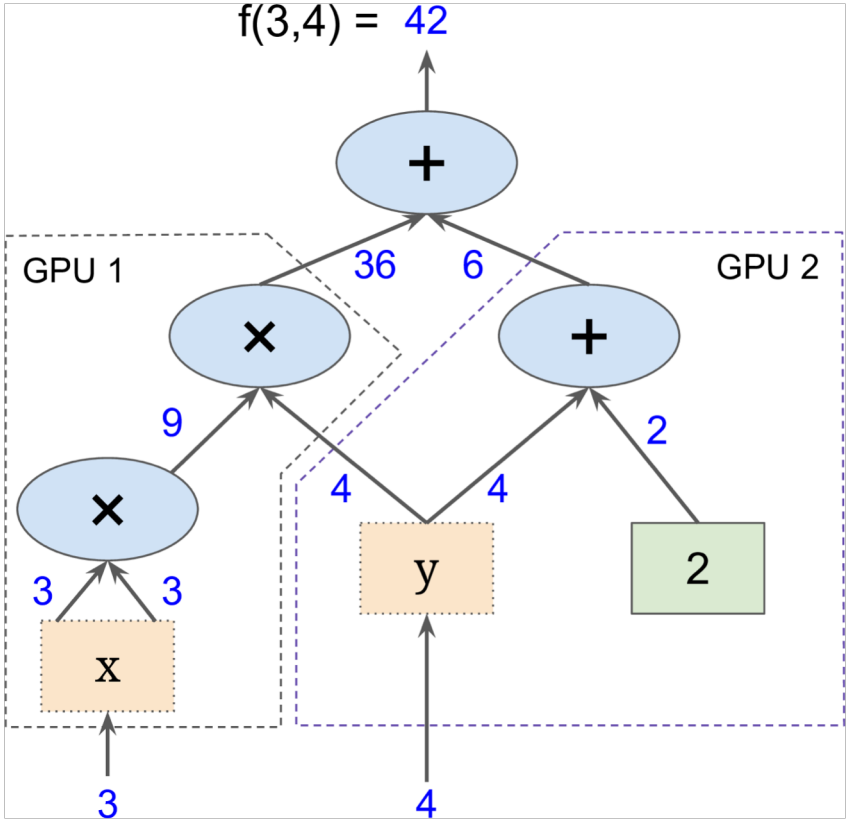
  z = x + y

- We now have a graph
- We need to create a session to evaluate z and thus obtain its value

Graph



Graph - Parallel



Defining a Tensorflow Graph

- Important to note that previous code does not perform any operation; merely creates a computational graph
- Variables aren't even initialized – also have to do the following

```
sess = tf.Session()
sess.run(x.initializer)
sess.run(y.initializer)
result = z.eval()
```

- Can also use global variables initializer

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

What is a Tensor?

- Tensors are the output of an Operation, which themselves act on Tensors
- Examples of Operations include tf.matmul or '+' in an expression like:

  $C = A + B$

  where A and B are Tensors, as is C

Graphs

- Any created node is added to default graph unless otherwise specified

```
x1 = tf.Variable(1)
x.graph is tf.get_default_graph()
>> True
```

- Can use *with* statement to temporarily make graph the default

```
graph = tf.Graph()
with graph as default():
    x2 = tf.Variable(2)
```

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 9 – Section 3

TensorFlow Programs

Tensorflow Sessions

- An instantiation of the graph is called a session

```
Sess = tf.Session()
```

- Variables are initialized, and then graph operations are evaluated either by utilizing a session's run

```
sess.run(my_op)
```

- or via

```
my_op.eval()
```

- Running `eval()` is equivalent to running `tf.get_default_session().run(my_op)`

  as any evaluation of a tensor has to be associated with a Session

- A nice trick: if you are ever interested in the value of a tensor *T* during execution, simply do `T_value = sess.run(T)`
  - This fetches the value from the computational graph

Feeding Data

- Placeholders act as a kind of an intermediary between Python/Numpy and TensorFlow

- A placeholder's shape is defined upon instantiation, and we typically pass a list or a Numpy array of a matching shape via the `feed_dict` arg of a Session's `run()` method

```
A = tf.placeholder(tf.float32, shape=(None, 3))
B = A + 5
res = sess.run(B, feed_dict={A: [[4,5,6],[7,8,9]]}
```

→ `res will be assigned [[9,10,11], [12,13,14]]`

Model Persistence

- Saving weights for future use is important for a number of reasons
  - May want to inference at some point in the future, not just immediately after network has been trained
  - Weights are often time-consuming to compute
  - Training is often an iterative process: starting with your previous set of weights will save a lot of time
  - Bad things happen and it makes sense to save weights while network is training; if program crashes, can pick up where you left off

Model Persistence (cont'd)

- Say you want to save every hundred epochs:

```
epochs=1000
sess = tf.Session()
init = tf.global_variables_initializer()
saver = tf.train.Saver()
for epoch in range(epochs):
  if epoch % 100 == 0:
    save_path = saver.save(sess,
'./models/my_model.ckpt')
```

- Restore with:

```
saver.restore(sess, './models/my_model.ckpt')
```

Name Scopes

- Name scopes keep graph uncluttered
- Allows for reuse of variable names elsewhere
- `tf.name_scope("my_scope")` will add `"my_scope"` as a prefix to all variables created inside it, for example:

```
with tf.name("my_scope")
   err = y_pred - y
```

yields

```
>> print err.op.name
my_scope/err
```

Variable Sharing

- Like traditional programming, can create TF variables and pass them to various functions
- `get_variable` allows for fetching of variables without explicitly passing them as parameters

```
with tf.variable_scope("pooling", reuse=True):
    tf.get_variable("threshold")
```

- Initialized with

```
with tf.variable_scope("pooling"):
    tf.get_variable("threshold", shape=(),
    initializer=tf.constant_initializer(0.0))
```

Module 9 – Section 4

Computing the Gradient

Numerical Differentiation

- In order to optimize neural networks, i.e. find an appropriate set of weights, need to differentiate functions
- Actual derivative of a function

$$f'(x_0) = \lim_{h \to 0} (f(x_0 + h) - f(x_0))$$

- Can be approximated by forward difference (among others)

$$f'(x) \approx \frac{f(x_0+h)-f(x_0)}{h}$$

- Approximation has lots of problems, better to use symbolic differentiation

Symbolic Differentiation

- Consider function

$$f(x) = \exp(\exp(\exp(x)))$$

- Derivative is

$$f'(x) = \exp(x) * \exp(\exp(x)) * \exp(\exp(\exp(x)))$$

- Evaluating naively would involve calling $\exp(\cdot)$ six times to eval $f'(x)$
- Better solution: eval first term $\exp(x)$ then use that to eval $\exp(\exp(x))$ and so on
- Can use TF graph to take care of this (as we will see)

Symbolic Differentiation (cont'd)

- Previous example was simple
- Gets worse when encountering something like

```
def my_func(a, b):
    z=0
    for i in range(100):
      z = a * np.cos(z+i) + z * np.sin(b-i)
    return z
```

- Clearly auto-differentiation would help here!

Autodiff

- One of the biggest benefits of using computation graph is ability to compute symbolic gradients
- If we define a set of operations, where X and y are inputs and outputs respectively and theta is a random uniform distr vector:

```
y_pred = tf.matmul(X, theta)
mse = tf.reduce_mean(tf.square(y_pred - y))
```

- We can compute the gradients as follows

```
grads = tf.gradients(mse, theta)
```

- From there, can use these gradients in a simple implementation of gradient descent!

Differentiation

- Suppose you define a function
  $f(x,y) = x^2y + y + 2$
  you need its partial derivatives to perform Gradient Descent

- Manual Differentiation:

$$\frac{\partial f}{\partial x} = \frac{\partial(x^2y)}{\partial x} + \frac{\partial y}{\partial x} + \frac{\partial 2}{\partial x} = y\frac{\partial(x^2)}{\partial x} + 0 + 0 = 2xy$$

$$\frac{\partial f}{\partial y} = \frac{\partial(x^2y)}{\partial y} + \frac{\partial y}{\partial y} + \frac{\partial 2}{\partial y} = x^2 + 1 + 0 = x^2 + 1$$

Numerical Differentiation

- Recall

$$h'(x_0) = \lim_{x \to x_0} \frac{h(x) - h(x_0)}{x - x_0}$$

$$= \lim_{\epsilon \to 0} \frac{h(x_0 + \epsilon) - h(x_0)}{\epsilon}$$

- If we want to calculate the partial derivative of $f(x,y)$ with regards to x, at $x = 3$ and $y = 4$,

```
def f(x, y):
    return x**2*y + y + 2

def derivative(f, x, y, x_eps, y_eps):
    return (f(x + x_eps, y + y_eps) - f(x, y)) / (x_eps + y_eps)

df_dx = derivative(f, 3, 4, 0.00001, 0)
df_dy = derivative(f, 3, 4, 0, 0.00001)
```

Forward-Mode Autodiff

- It relies on *dual numbers*, which are numbers of the form $a + b\epsilon$ where a and b are real numbers and ϵ is an infinitesimal number such that $\epsilon^2 = 0$ (but $\epsilon \neq 0$)
- You can think of the dual number $42 + 24\epsilon$ as something akin to 42.0000⋯000024 with an infinite number of 0s
- Algebra:

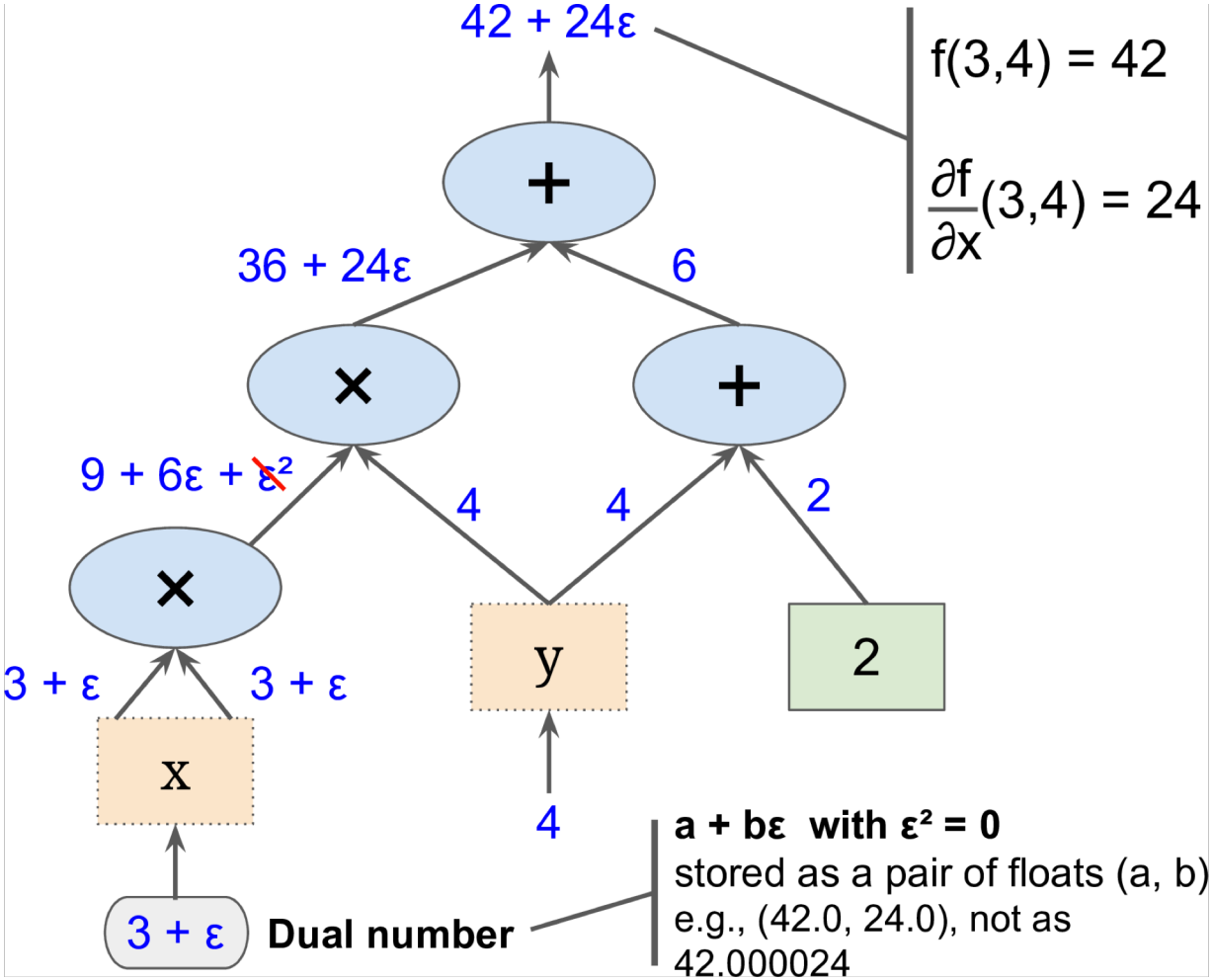
$$\lambda(a + b\epsilon) = \lambda a + \lambda b\epsilon$$

$$(a + b\epsilon) + (c + d\epsilon) = (a + c) + (b + d)\epsilon$$

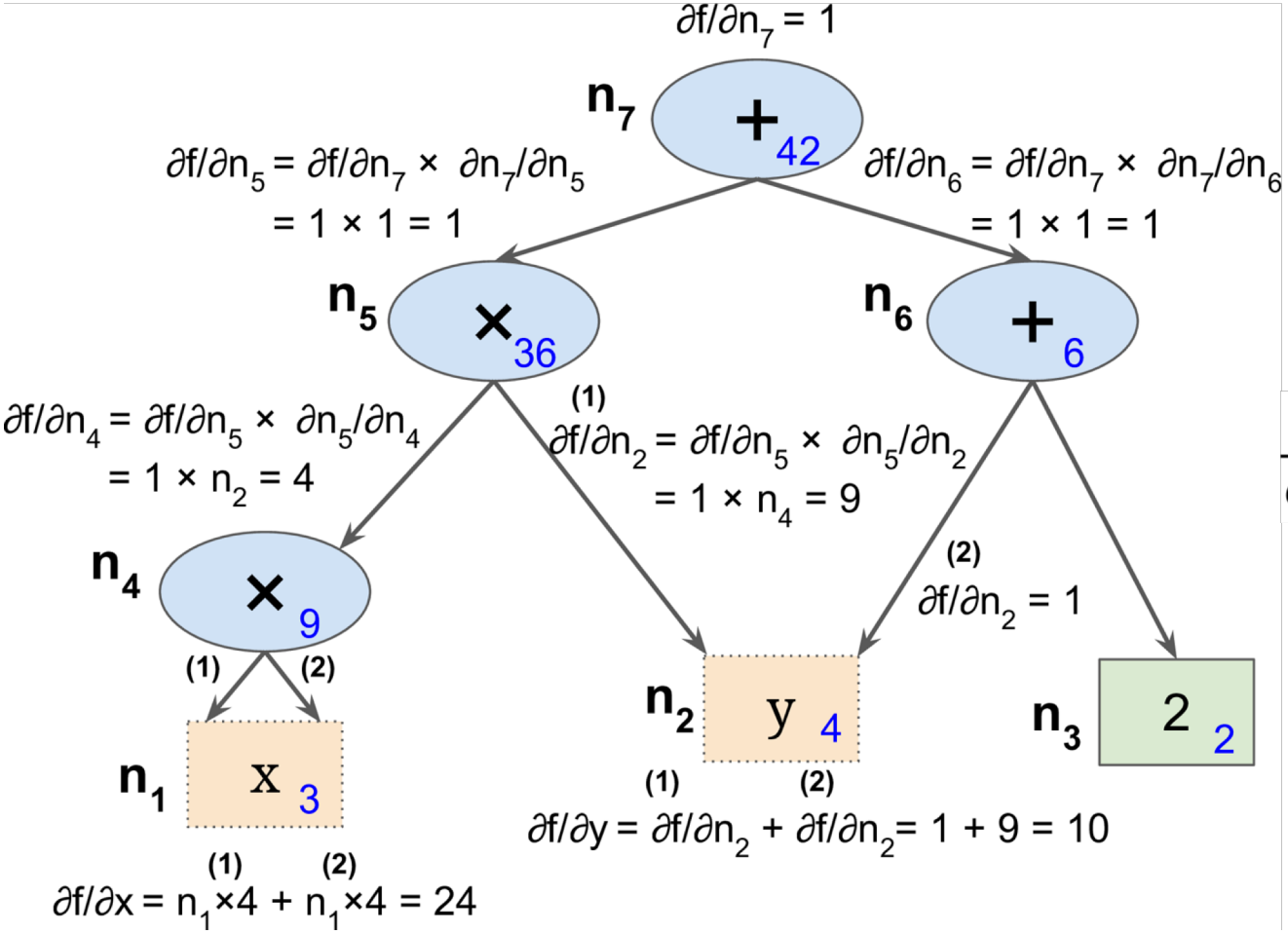
$$(a + b\epsilon) \times (c + d\epsilon) = ac + (ad + bc)\epsilon + (bd)\epsilon^2 = ac + (ad + bc)\epsilon$$

- It can be shown that $h(a + b\epsilon) = h(a) + b \times h'(a)\epsilon$
  - So computing $h(a + \epsilon)$ gives you both $h(a)$ and the derivative $h'(a)$ in just one shot

Forward-Mode Autodiff (cont'd)



Reverse-Mode Autodiff



$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial n_i} \times \frac{\partial n_i}{\partial x}$$

Optimizers

- In reality, we won't explicitly call tf.gradients() or write our own gradient descent code
- TensorFlow comes bundled with optimizers out of the box

```
optimizer = tf.train.MomentumOptimizer(
learning_rate=0.001, momentum=0.9)
```

- Need to provide function for optimizer to minimize:

```
optimizer.minimize(mse)
```

- Will talk more about optimizers in next lecture

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 9 – Section 5

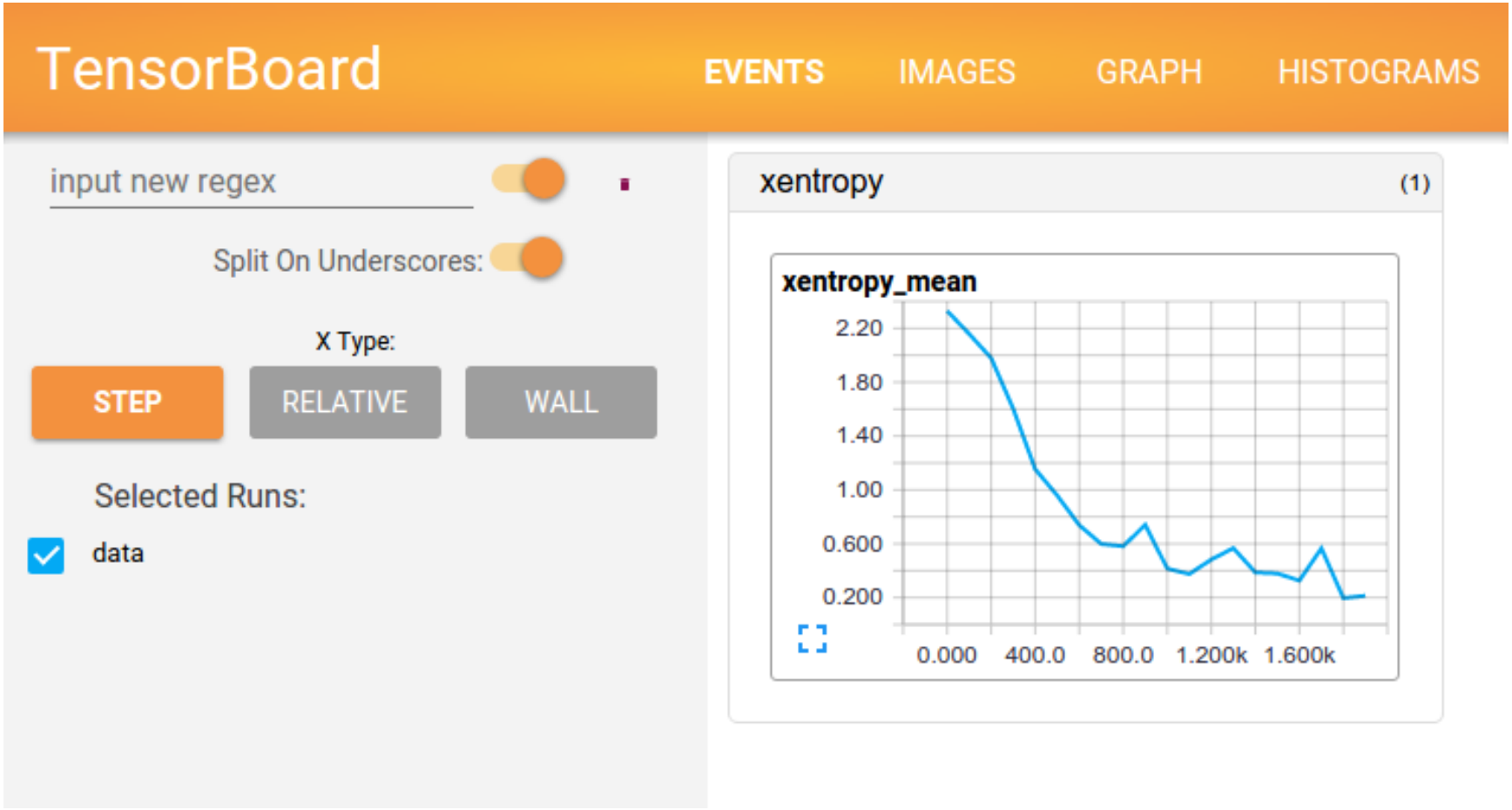
TensorBoard

TensorBoard

- Useful to track progress of training/validation losses visually

```
mse_summary = tf.summary.scalar('MSE', mse)
 file_writer = tf.summary.FileWriter(logdir,
    tf.get_default_graph)
 for batch in range(n_batches):
   …
   if batch %10 == 0:
     summary_str = mse_summary.eval(feed_dict =
      {X: x_batch, y: y_batch}
     step = epoch * n_batches + batch
     file_writer.add_summary(summary_str, batch)
```

TensorBoard (cont'd)



UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 9 – Section 6

Regression with TensorFlow

Linear Regression

- TensorFlow operations (also called *ops* for short) can take any number of inputs and produce any number of outputs.
  - The addition and multiplication ops each take two inputs and produce one output.
  - Constants and variables take no input (they are called *source ops*).
  - The inputs and outputs are multidimensional arrays, called *tensors* (hence the name “tensor flow”). Just like NumPy arrays, tensors have a type and a shape.
  - In fact, in the Python API tensors are simply represented by NumPy ndarrays. They typically contain floats, but you can also use them to carry strings (arbitrary byte arrays).

Linear Regression (cont'd)

```
import numpy as np
from sklearn.datasets import fetch_california_housing

housing = fetch_california_housing()
m, n = housing.data.shape
housing_data_plus_bias = np.c_[np.ones((m, 1)), housing.data]

X = tf.constant(housing_data_plus_bias, dtype=tf.float32, name="X")
y = tf.constant(housing.target.reshape(-1, 1), dtype=tf.float32, name="y")
XT = tf.transpose(X)
theta = tf.matmul(tf.matmul(tf.matrix_inverse(tf.matmul(XT, X)), XT), y)

with tf.Session() as sess:
    theta_value = theta.eval()
```

Module 9 – Section 7

Regression with Gradient Descent

Linear Regression – Gradient Descent (GD)

- The `random_uniform()` function creates a node in the graph that will generate a tensor containing random values, given its shape and value range, much like NumPy's rand() function.
- The `assign()` function creates a node that will assign a new value to a variable. In this case, it implements the Batch Gradient Descent step $\theta^{(\text{next step})} = \theta - \eta \nabla_\theta \text{MSE}(\theta)$.
- The main loop executes the training step over and over again (n_epochs times), and every 100 iterations it prints out the current MSE.

Linear Regression – GD (cont'd)

```
n_epochs = 1000
learning_rate = 0.01

X = tf.constant(scaled_housing_data_plus_bias, dtype=tf.float32, name="X")
y = tf.constant(housing.target.reshape(-1, 1), dtype=tf.float32, name="y")
theta = tf.Variable(tf.random_uniform([n + 1, 1], -1.0, 1.0), name="theta")
y_pred = tf.matmul(X, theta, name="predictions")
error = y_pred - y
mse = tf.reduce_mean(tf.square(error), name="mse")
gradients = 2/m * tf.matmul(tf.transpose(X), error)
training_op = tf.assign(theta, theta - learning_rate * gradients)

init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)

    for epoch in range(n_epochs):
        if epoch % 100 == 0:
            print("Epoch", epoch, "MSE =", mse.eval())
        sess.run(training_op)

    best_theta = theta.eval()
```

Gradient Descent - Optimizer

- TensorFlow computes the gradients for you.
  - it also provides a number of optimizers out of the box, including a Gradient Descent optimizer.

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(mse)
```

  - If you want to use a different type of optimizer, you just need to change one line.

```
optimizer = tf.train.MomentumOptimizer(learning_rate=learning_rate,
                                       momentum=0.9)
```

Gradient Descent – Optimizer (cont'd)

- To implement Mini-batch Gradient Descent: First change the definition of X and y in the construction phase to make them placeholder nodes:

```
X = tf.placeholder(tf.float32, shape=(None, n + 1), name="X")
y = tf.placeholder(tf.float32, shape=(None, 1), name="y")
```

- Then define the batch size and compute the total number of batches:

```
batch_size = 100
n_batches = int(np.ceil(m / batch_size))
```

Gradient Descent – Optimizer (cont'd)

- In the execution phase, fetch the mini-batches one by one, then provide the value of X and y via the feed_dict parameter

```
def fetch_batch(epoch, batch_index, batch_size):
    [...] # load the data from disk
    return X_batch, y_batch

with tf.Session() as sess:
    sess.run(init)

    for epoch in range(n_epochs):
        for batch_index in range(n_batches):
            X_batch, y_batch = fetch_batch(epoch, batch_index, batch_size)
            sess.run(training_op, feed_dict={X: X_batch, y: y_batch})

    best_theta = theta.eval()
```

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 9 – Section 8

Resources and Wrap-up

Next Class

- Introduction to Deep Learning and Deep Neural Networks

Follow us on social

Join the conversation with us online:

facebook.com/uoftscs

@uoftscs

linkedin.com/company/university-of-toronto-school-of-continuing-studies

@uoftscs

Any questions?

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Thank You

Thank you for choosing the University of Toronto School of Continuing Studies