TensorFlow usage

Babii Andrii

Ph.D. student, Kharkiv National University of Radioelectronics

andrii.babii@nure.ua

Motivation

1. Data and model parallelism
2. TensorBoard for visualization
3. Computational graph abstraction
4. Python + Numpy
5. Great documentation and examples
6. More than deep learning framework

\+ Now conception of 'Python front-end' for hard backend is trending

TinyFlow
http://dmlc.ml/2016/09/30/build-your-own-tensorflow-with-nnvm-and-torch.html
Syntax ('Frontend') like TensorFlow but interpretation is … Torch!
NNVM inspired by **LLVM**…
It provides ways to construct, represent and transform computation graphs invariant of how it is executed.

TensorFlow basic concepts

A TensorFlow computation is described by a directed **graph** , which is composed of a set of **nodes**

Library user construct a computational **graph** using one of the supported frontend languages (C++ or Python)

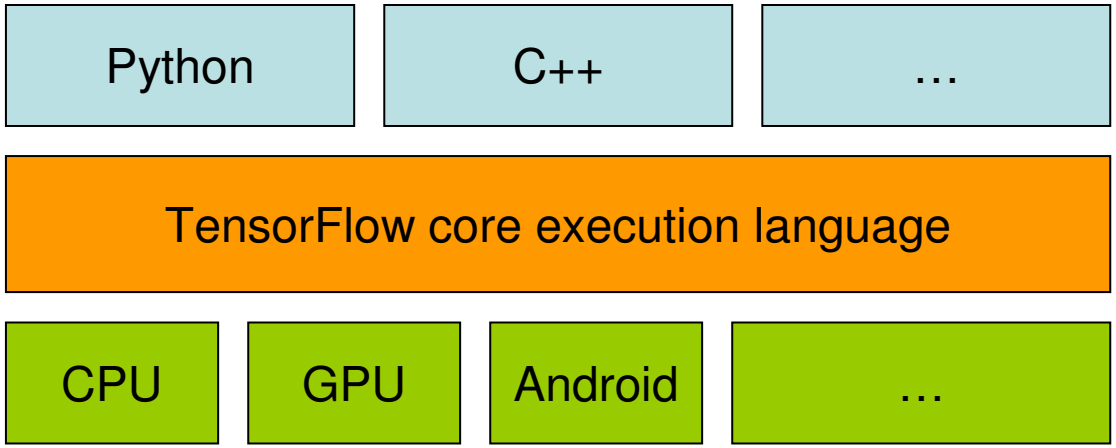
In a TensorFlow graph, each **node** has zero or more inputs and zero or more outputs, and represents the instantiation of an operation

Values that flow along normal edges in the graph (from outputs to inputs) are **tensors** - arbitrary dimensionality arrays where the underlying element type is specified or inferred at graph-construction time

Special edges, called **control dependencies** , can also exist in the graph

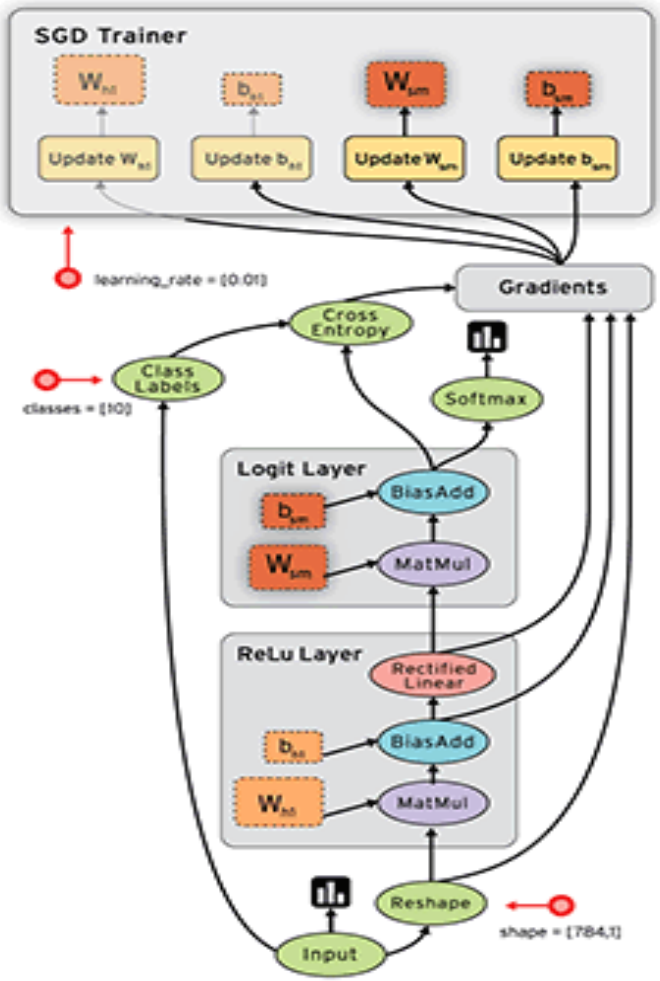
http://download.tensorflow.org/paper/whitepaper2015.pdf

TensorFlow architecture



There is the **client**, which uses the **session** interface to communicate with the **master** and one or more **worker** processes
Each **worker process** responsible for arbitrating access to one or more **computational devices** (such as CPU cores and GPU cards)

TensorFlow Computation graph



https://www.tensorflow.org/

TensorFlow basic concepts. Tensor

A Tensor is a typed multi-dimensional array. For example, a 4-D array of floating point numbers representing a mini-batch of images with dimensions [batch, height, width, channel].

In a launched graph: Type of the data that flow between nodes.

In the Python API: class used to represent the output and inputs of ops added to the graph tf.Tensor. Instances of this class do not hold data.

In the C++ API: class used to represent tensors returned from a Session::Run() call tensorflow::Tensor. Instances of this class hold data.

https://www.tensorflow.org/versions/r0.9/resources/glossary.html#glossary

TensorFlow basic concepts. Operations

An **operation** has a name and represents an abstract computation (e.g., “matrix multiply”, or “add”). An operation can have attributes.

- One common use of attributes is to make operations polymorphic over different tensor element types.

A **kernel** is a particular implementation of an operation that can be run on a particular type of device (e.g., CPU or GPU).

TensorFlow binary defines the sets of operations and kernels available via a registration mechanism, and this set can be extended by linking in additional operation and/or kernel definitions/registration

Variable is a special kind of operation that returns a handle to a persistent mutable tensor that survives across executions of a graph.

http://download.tensorflow.org/paper/whitepaper2015.pdf

TensoFlow built-in operations

Category	Examples
Element-wise mathematical operations	Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal, ...
Array operations	Concat, Slice, Split, Constant, Rank, Shape, Shuffle, ...
Matrix operations	MatMul, MatrixInverse, MatrixDeterminant, ...
Stateful operations	Variable, Assign, AssignAdd, ...
Neural-net building blocks	SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool, ...
Checkpointing operations	Save, Restore
Queue and synchronization operations	Enqueue, Dequeue, MutexAcquire, MutexRelease, ...
Control flow operations	Merge, Switch, Enter, Leave, NextIteration

http://download.tensorflow.org/paper/whitepaper2015.pdf

TensorFlow. Execution of computation graph

Single device (for example we have only one core CPU for computation)

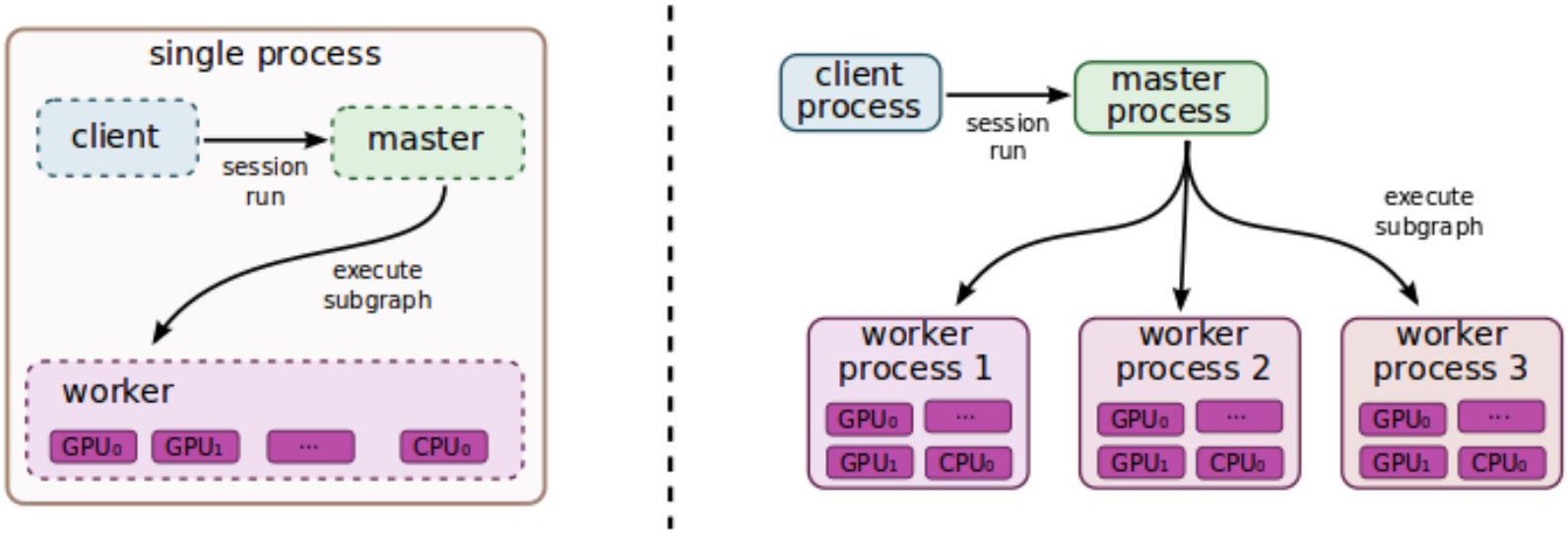
The nodes of the graph are executed in an order that respects the dependencies between nodes

Multi-device execution

- Select device to place the computation for each node in the graph
- Managing the required communication of data across device boundaries implied by these placement decisions

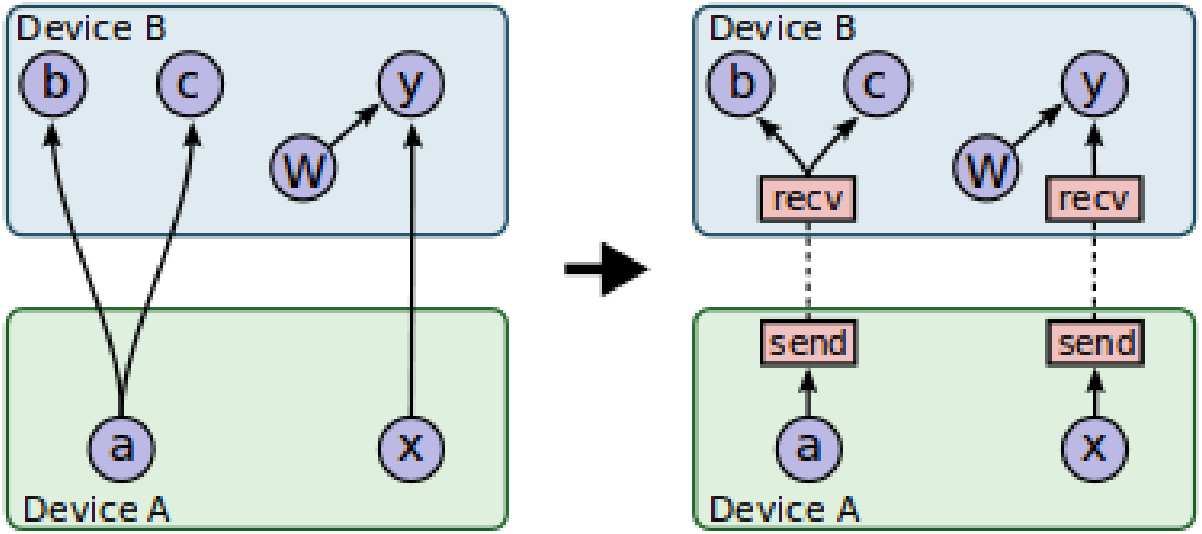
http://download.tensorflow.org/paper/whitepaper2015.pdf

TensorFlow node placement



http://download.tensorflow.org/paper/whitepaper2015.pdf

Cross-device communications



http://download.tensorflow.org/paper/whitepaper2015.pdf

TensorFlow. Extensions

•Automatic Differentiation – Automatically computes gradients for data flow graphs.

•Partial Execution – Allows TensorFlow clients to execute a subgraph of the entire execution graph.

•Device Constraints – Allows TensorFlow clients to control the placement of nodes on a device.

•Control Flow – Enables support for conditionals and loops in data flow graphs.

•Input Operations – Facilitate efficient loading of data into large scale models from the storage system.

•Queues – Allow different portions of the graph to execute asynchronously and to hand off data through Enqueue and Dequeue operation. Enqueue and Dequeue operations are blocking.

•Containers – The mechanism within TensorFlow for managing longer-lived mutable stat

TensorFlow. Session

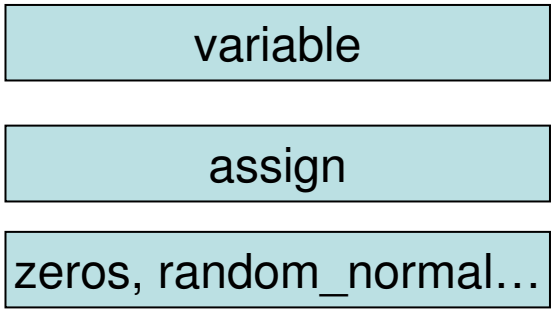
A Session object encapsulates the environment in which Tensor objects are evaluated - TensorFlow Docs

```
import  tensorflow as tf

a =  tf.constant(5.0)
b =  tf.constant(3.0)
c = a +b

with  tf.Session() as sess:
    print (sess.run(c))     # print(c.eval()) – will do the same (for current opened session)
```

tf.InteractiveSession()
is just convenient synonym for keeping a default session open in ipython
sess.run(c)
is an example of a TensorFlow Fetch

TensorFlow. Variable

Variables are in-memory buffers containing tensors.
They must be explicitly initialized and can be saved to disk during and after training
TensorFlow Docs




```
import  tensorflow as tf

weights = tf.Variable(tf.random_normal([100, 150], stddev=0.5), name="weights")
biases = tf.Variable(tf.zeros([150]), name="biases")
#https://www.tensorflow.org/versions/r0.11/api_docs/python/constant_op.html#random_normal

# Pin a variable to GPU.
with tf.device("/gpu:0"):
  v = tf.Variable(...)

# Pin a variable to a particular parameter server task.
with tf.device("/job:ps/task:7"):
  v = tf.Variable(...)
```

TensorFlow. Variable

Variable initializers must be run explicitly before other ops in your model can be run. The easiest way to do that is to add an op that runs all the variable initializers, and run that op before using the model. - TensorFlow Docs

```
init_op = tf.initialize_all_variables()
saver = tf.train.Saver()
# Later, when launching the model
with tf.Session() as sess:
  # Run the init operation.
  sess.run(init_op)
  ...
  # Use the model
  …
  # Save the variables to disk.
  save_path = saver.save(sess, "/tmp/model.ckpt")
  print("Model saved in file: %s" % save_path)
```

Or we can init variable from value of other variable (it should be initialized before):

```
w2 = tf.Variable(weights.initialized_value(), name="w2")
```

tf.train.Saver object have restore method: saver.restore(sess, "/tmp/model.ckpt")

https://www.tensorflow.org/versions/r0.11/how_tos/variables/index.html

TensorFlow. Common syntax examples

Fill array with zeros and ones: a = tf.zeros((3,3)), b = tf.ones((3,3))

Sum of array, axis = 1: tf.reduce_sum(a,reduction_indices=[1])

Shape of array: a.get_shape()

Re-shape: array: tf.reshape(a,(1,4))

Basic arithmetic: a*3+ 2

Multiplication: tf.matmul(c, d)

Element accessing: a[0,0], a[:,0], a[0,:]

TensorFlow data input

How can we input external data into TensorFlow?

Simple solution: Import from Numpy:

```
a = np.zeros((3,3))
ta = tf.convert_to_tensor(a)
```

Simple, but does not scale

TensorFlow data input

Use tf.placeholder variables (dummy nodes that provide entry points for data to computational graph).

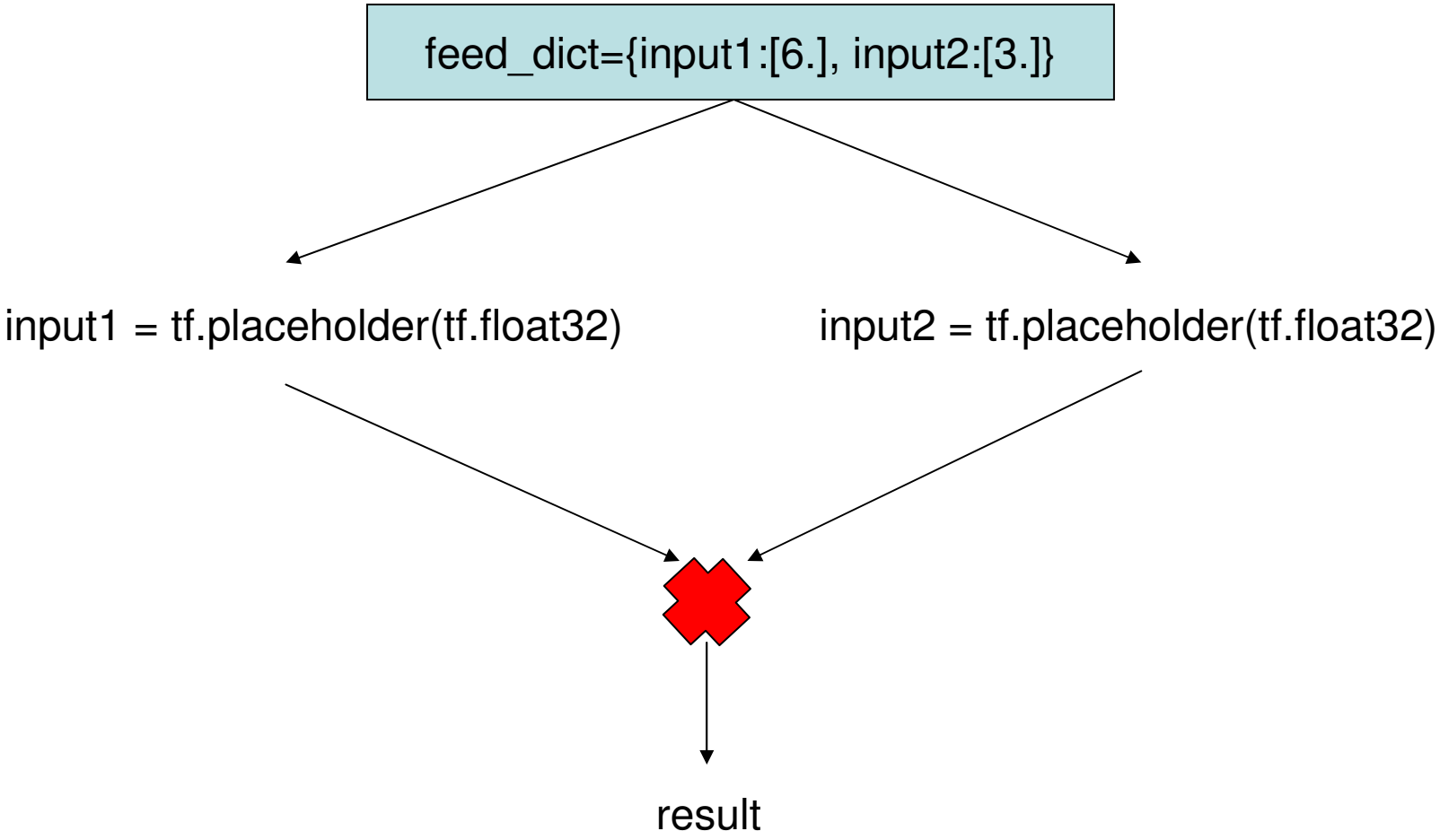
A feed_dict is a python dictionary mapping from tf.placeholder vars (or their names) to data (numpy arrays, lists, etc.)

Example:

```
input1 = tf.placeholder(tf.float32)
input2 = tf.placeholder(tf.float32)
output = tf.mul(input1, input2)

with tf.Session()as sess:
    print(sess.run([output], feed_dict={input1:[6.], input2:[3.]}))
```

TensorFlow data input

Evaluation:



TensorFlow namespaces & get_variable

Variable Scope mechanism in TensorFlow consists of 2 main functions:

tf.get_variable(<name>, <shape>, <initializer>): Creates or returns a variable with a given name.

tf.variable_scope(<scope_name>): Manages namespaces for names passed to tf.get_variable().

tf.get_varible two cases:

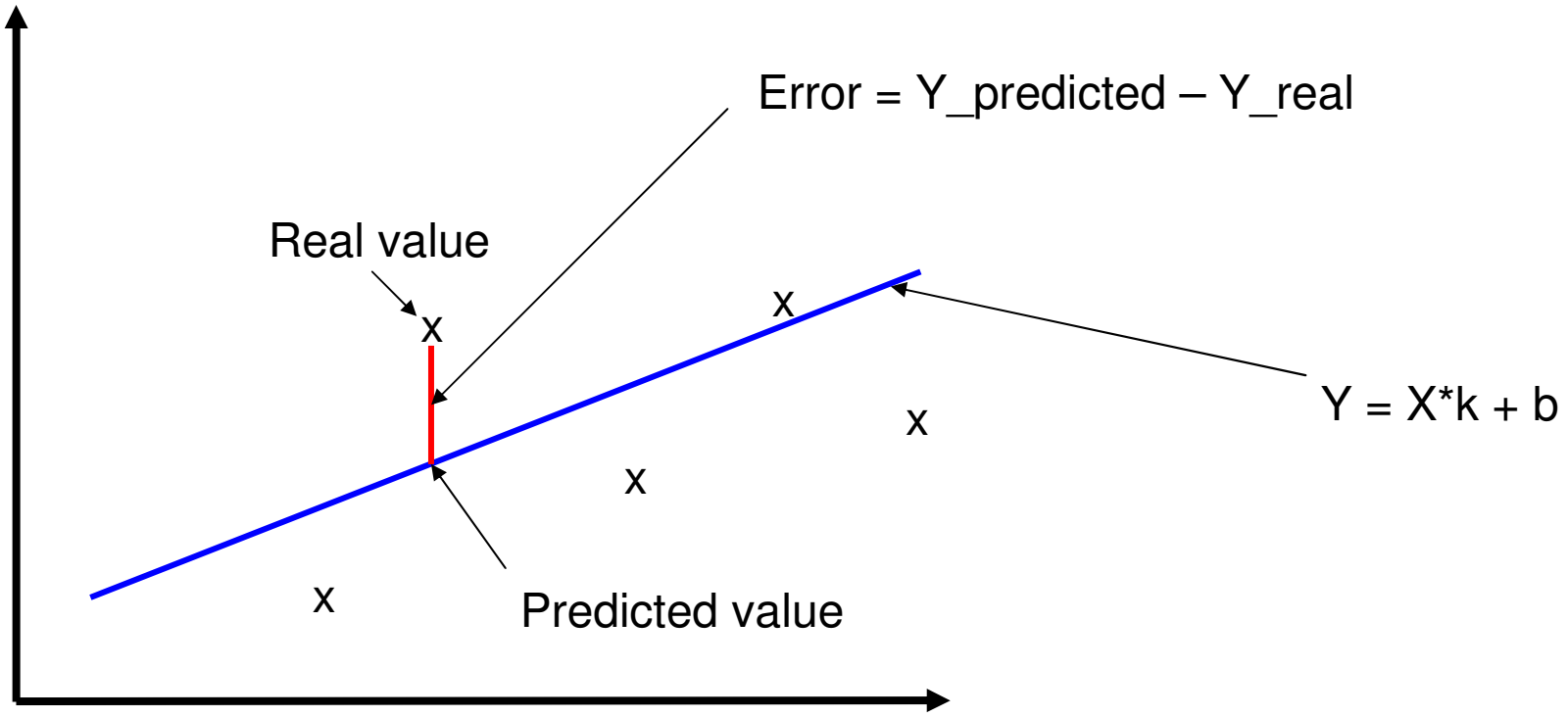
Case 1: the scope is set for creating new variables, as evidenced by tf.get_variable_scope().reuse == False.

Case 2: the scope is set for reusing variables, as evidenced by tf.get_variable_scope().reuse == True.

Example

Problem: Linear regression

$$\sum e^2 = \sum (\hat{y} - y)^2 \rightarrow \min$$

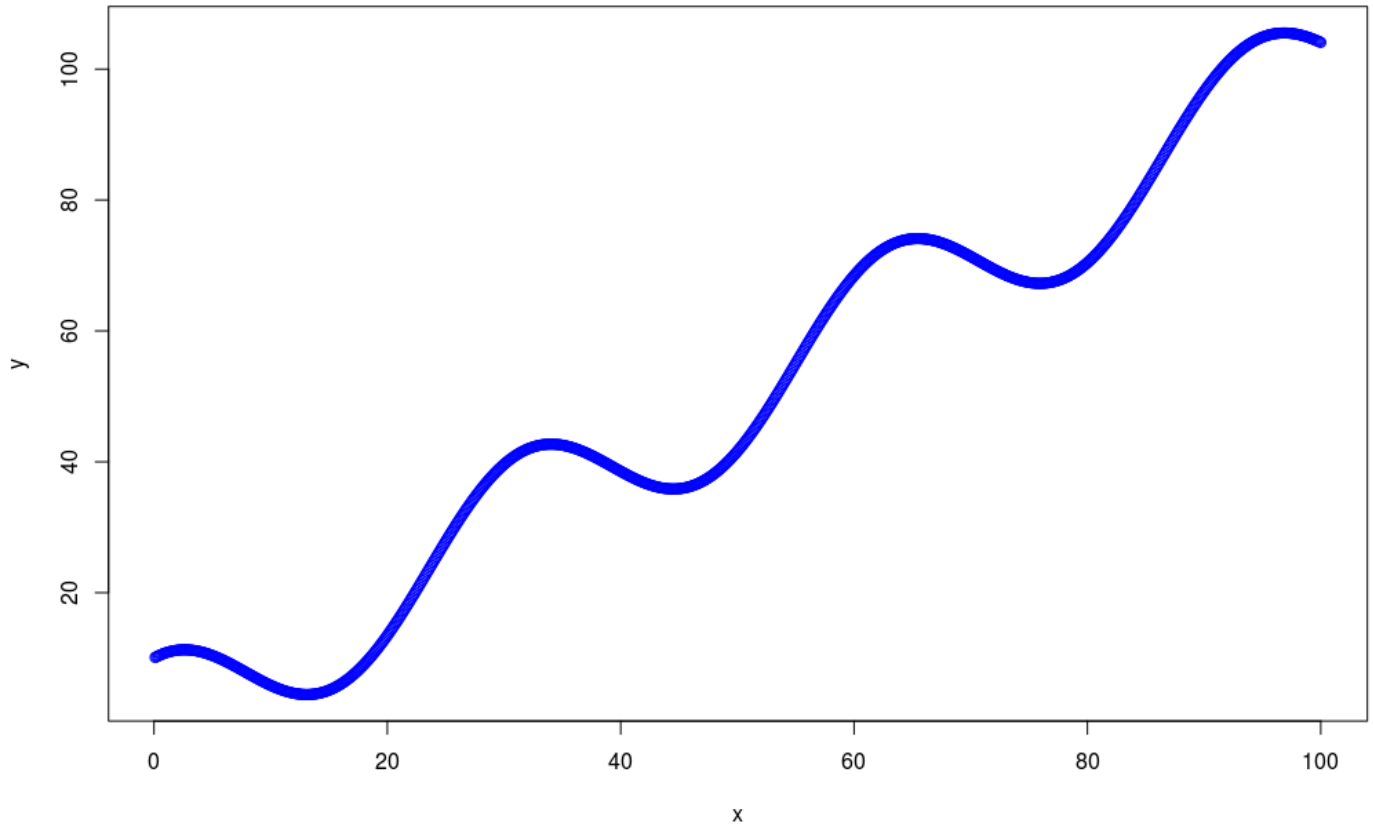


Example

https://github.com/anrew-git/tf_linear

```
import numpy as np
Import tensorflow as tf

# Prepre input data for regression. X from 1 to 100 with step 0.1
# Y = X+ 10*cos(X/5)
X_gen = np.arange(100, step=.1)
Y_gen = X_gen + 10 * np.cos(X_gen/5)

#Number of samples. 100/0.1 = 1000
n_samples = 1000

#Batch size
batch_size = 100

#Steps number
steps_number = 400
```

Example



Example

```
# Tensorflow is sensitive to shapes, so reshaping without data change
# It were  (n_samples,), now should be (n_samples, 1)

X_gen = np.reshape(X_gen, (n_samples,1))
Y_gen = np.reshape(Y_gen, (n_samples,1))

# Preparing  placeholders

X = tf.placeholder(tf.float32, shape=(batch_size, 1))
Y = tf.placeholder(tf.float32, shape=(batch_size, 1))
```

Example

```
# Define variables to be learned

with tf.variable_scope("linear-regression"):
    k = tf.get_variable("weights", (1, 1),
        initializer=tf.random_normal_initializer())
    b = tf.get_variable("bias", (1,),
        initializer=tf.constant_initializer(0.0))

y_predicted = tf.matmul(X, k) + b

loss = tf.reduce_sum((Y - y_predicted)**2)
```

Example

```
# Sample code to solve this problem

# Define optimizer properties – optimization type – minimization, variable
opt_operation = tf.train.AdamOptimizer().minimize(loss)

with tf.Session() as sess:
# Initialize Variables in graph

    sess.run(tf.initialize_all_variables())

    # Optimization loop for steps_number steps
    for i in range(steps_number):
        # Select random minibatch
        indices = np.random.choice(n_samples, batch_size)
        X_batch, y_batch = X_gen[indices], Y_gen[indices]
        # Do optimization step
        sess.run([opt_operation, loss],
        feed_dict={X: X_batch, Y: y_batch})
```

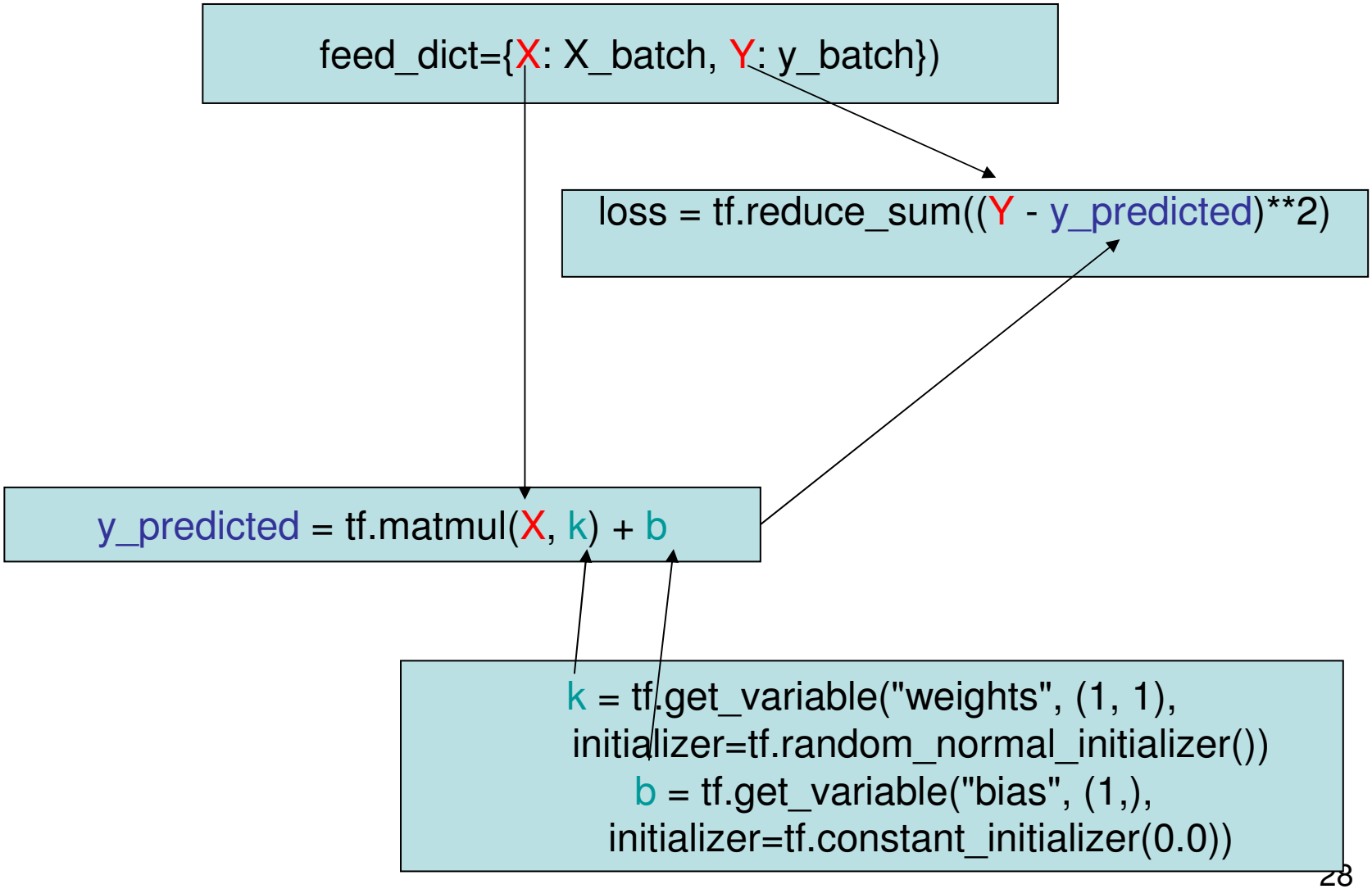
Example

Preparing mini-batches

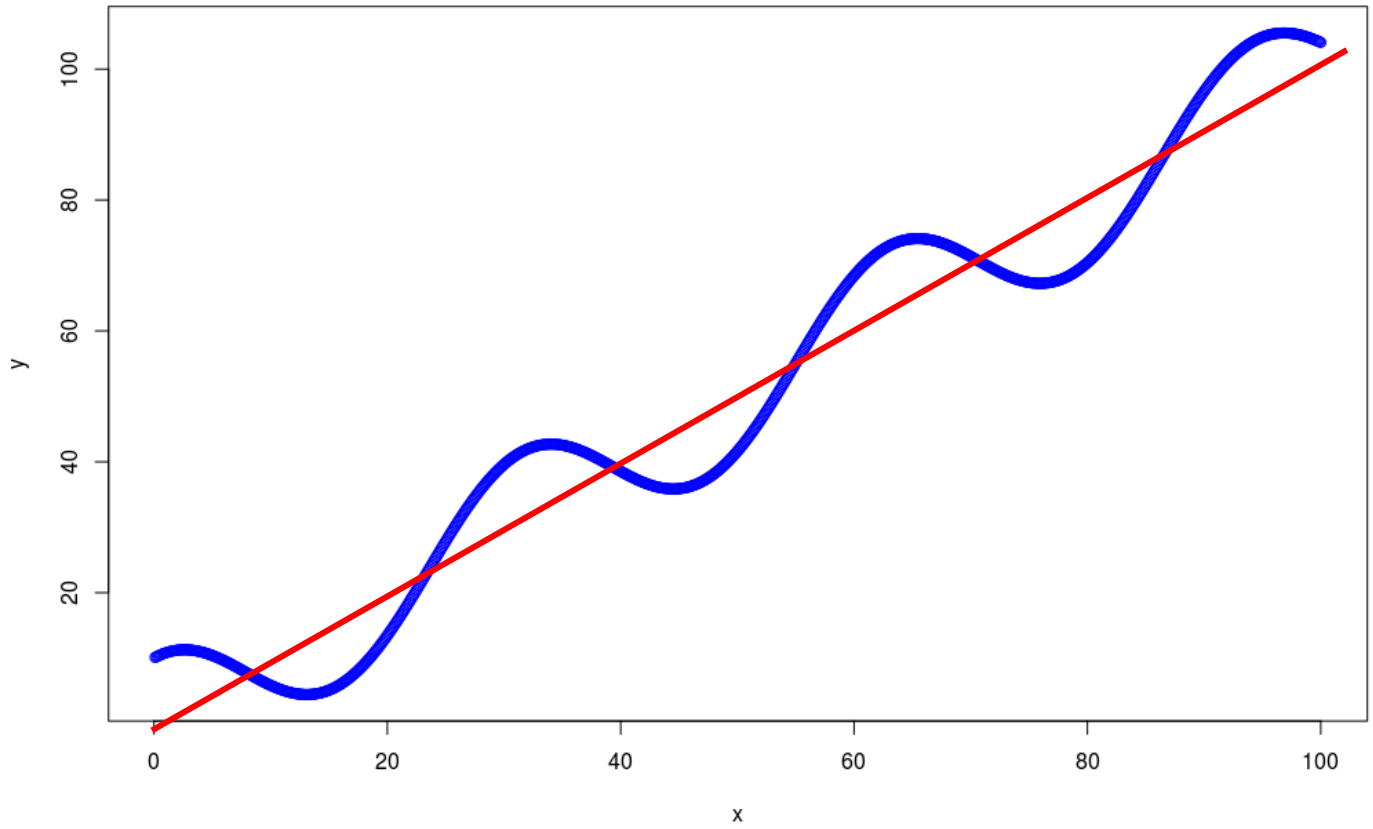
```
# Gradient descent loop for steps_number steps
for i in range(steps_number):

    # Select random minibatch

    batch_indices = np.random.choice(n_samples, batch_size)
    X_batch, y_batch = X_gen[batch_indices], Y_gen[batch_indices]

        # Do optimization step
        sess.run([opt_operation, loss],
        feed_dict={X: X_batch, Y: y_batch})
```

Inside sess.run – feed data to TensorFlow

Example


feed_dict={X: X_batch, Y: y_batch})
loss = tf.reduce_sum((Y - y_predicted)**2)
y_predicted = tf.matmul(X, k) + b
k = tf.get_variable("weights", (1, 1),
initializer=tf.random_normal_initializer())
b = tf.get_variable("bias", (1,),
initializer=tf.constant_initializer(0.0))

Example



TensorFlow auto-differentiation and gradient

Automatic differentiation computes gradients without user input

TensorFlow nodes in computation graph have attached gradient operations.

Use backpropagation (using node-specific gradient ops) to compute required gradients for all variables in graph

TensorFlow

1. TensorFlow has good computational graph visualization.
2. Support from such a huge company as Google is a plus for TensorFlow.
3. TensorFlow has C++ and Python interfaces.
4. TensorFlow has benefits on large computation problems and distributed heterogeneus computation enviroment
5. TensorFlow not so good on 1-GPU / single host hardware as Theano/Torch
6. TensorFlow base can be extended to the wide range of new hardware

References

1. http://download.tensorflow.org/paper/whitepaper2015.pdf
2. https://www.tensorflow.org/
3. Getting Started with TensorFlow by Giancarlo Zaccone
4. TensorFlow Machine Learning Cookbook Paperback by Nick McClure
5. https://github.com/aymericdamien/TensorFlow-Examples
6. https://www.tensorflow.org/versions/r0.10/tutorials/index.html
7. http://bcomposes.com/2015/11/26/simple-end-to-end-tensorflow-examples/
8. https://github.com/anrew-git/tf_linear
9. Основные концепции нейронных сетей. Р. Каллан

Questions?