Probabilistic Programming

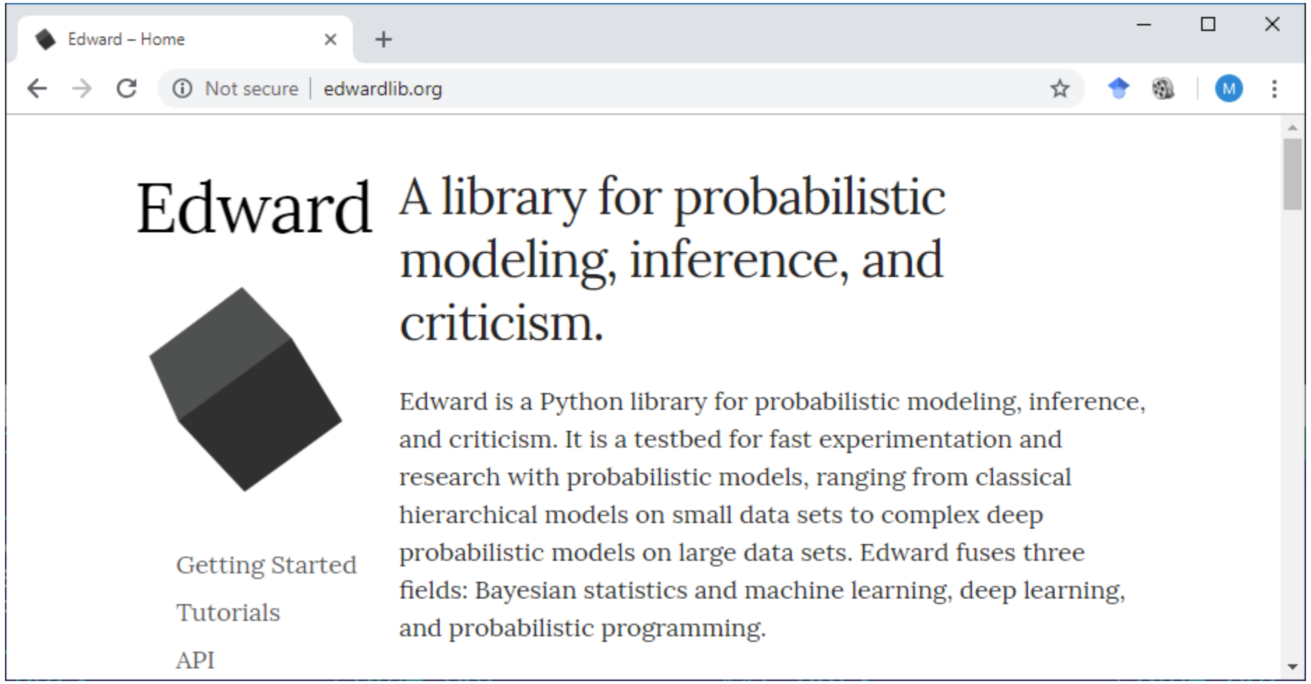
Marius Popescu

popescunmarius@gmail.com

2019 - 2020

TensorFlow Probability / Edward2

Some History



Edward – Home

Not secure | edwardlib.org

Edward

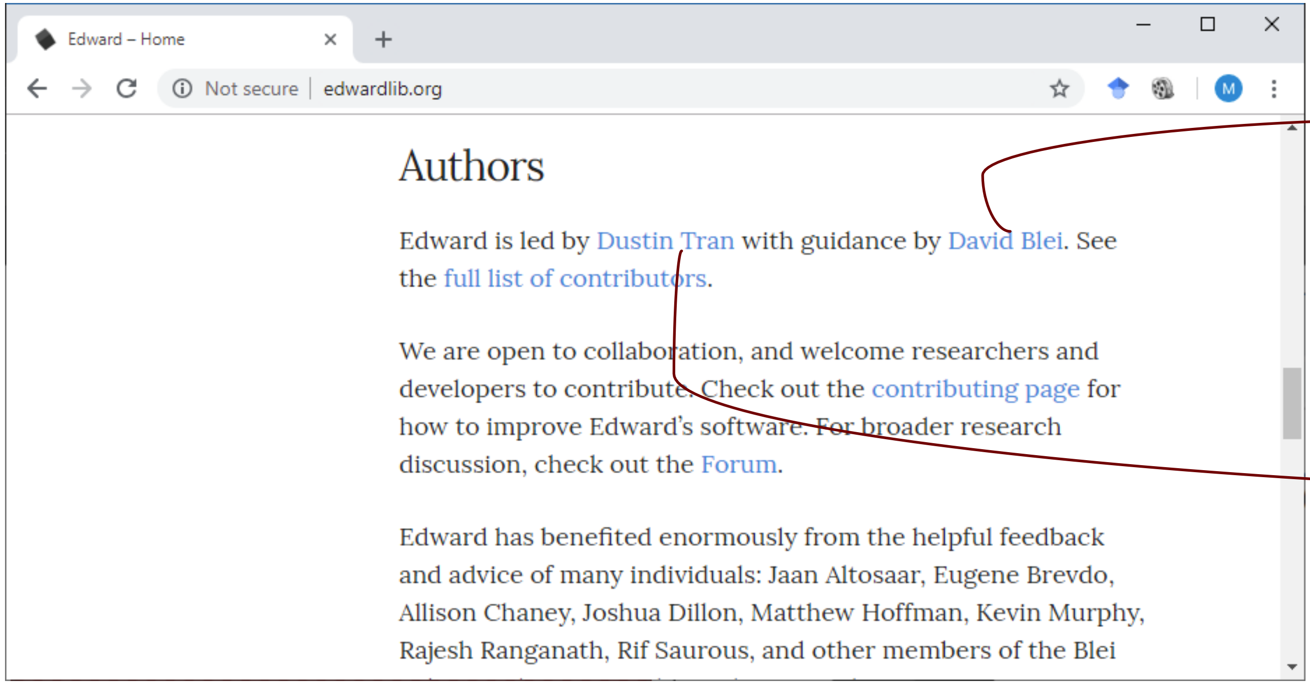
A library for probabilistic modeling, inference, and criticism.

Getting Started

Tutorials

API

Edward is a Python library for probabilistic modeling, inference, and criticism. It is a testbed for fast experimentation and research with probabilistic models, ranging from classical hierarchical models on small data sets to complex deep probabilistic models on large data sets. Edward fuses three fields: Bayesian statistics and machine learning, deep learning, and probabilistic programming.



Edward – Home

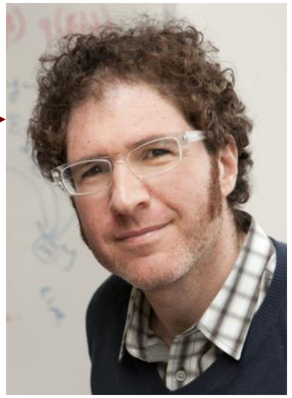
Not secure | edwardlib.org

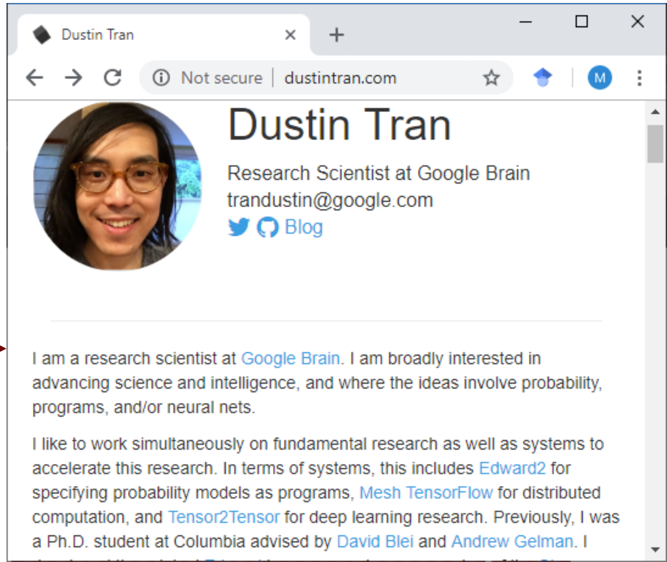
Authors

Edward is led by Dustin Tran with guidance by David Blei. See the full list of contributors.

We are open to collaboration, and welcome researchers and developers to contribute. Check out the contributing page for how to improve Edward's software. For broader research discussion, check out the Forum.

Edward has benefited enormously from the helpful feedback and advice of many individuals: Jaan Altosaar, Eugene Brevdo, Allison Chaney, Joshua Dillon, Matthew Hoffman, Kevin Murphy, Rajesh Ranganath, Rif Saurous, and other members of the Blei





Dustin Tran

Not secure | dustintran.com

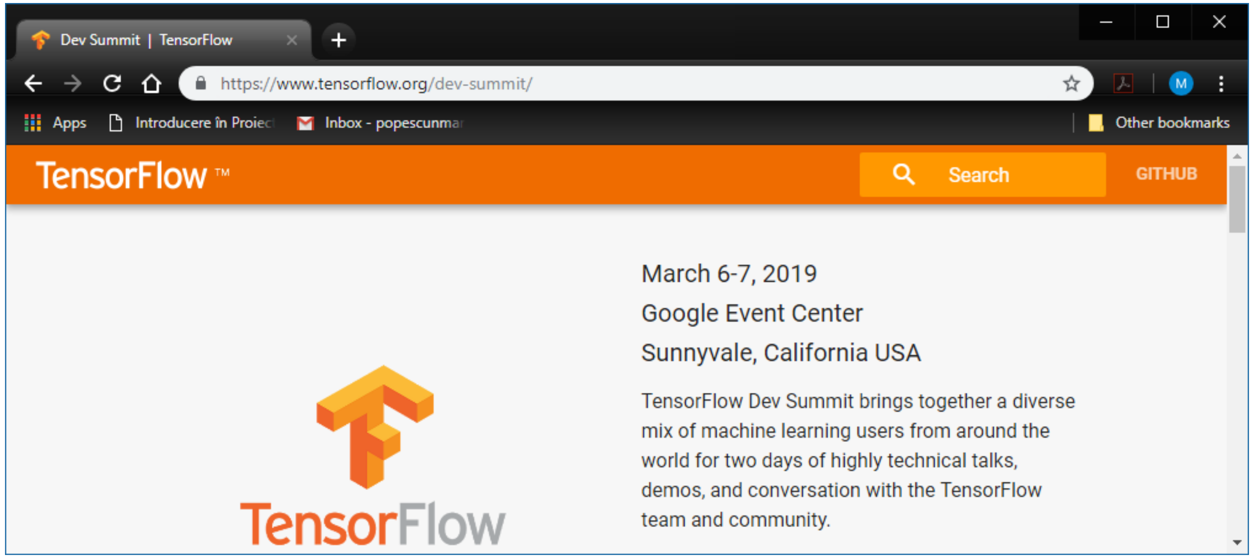
Dustin Tran

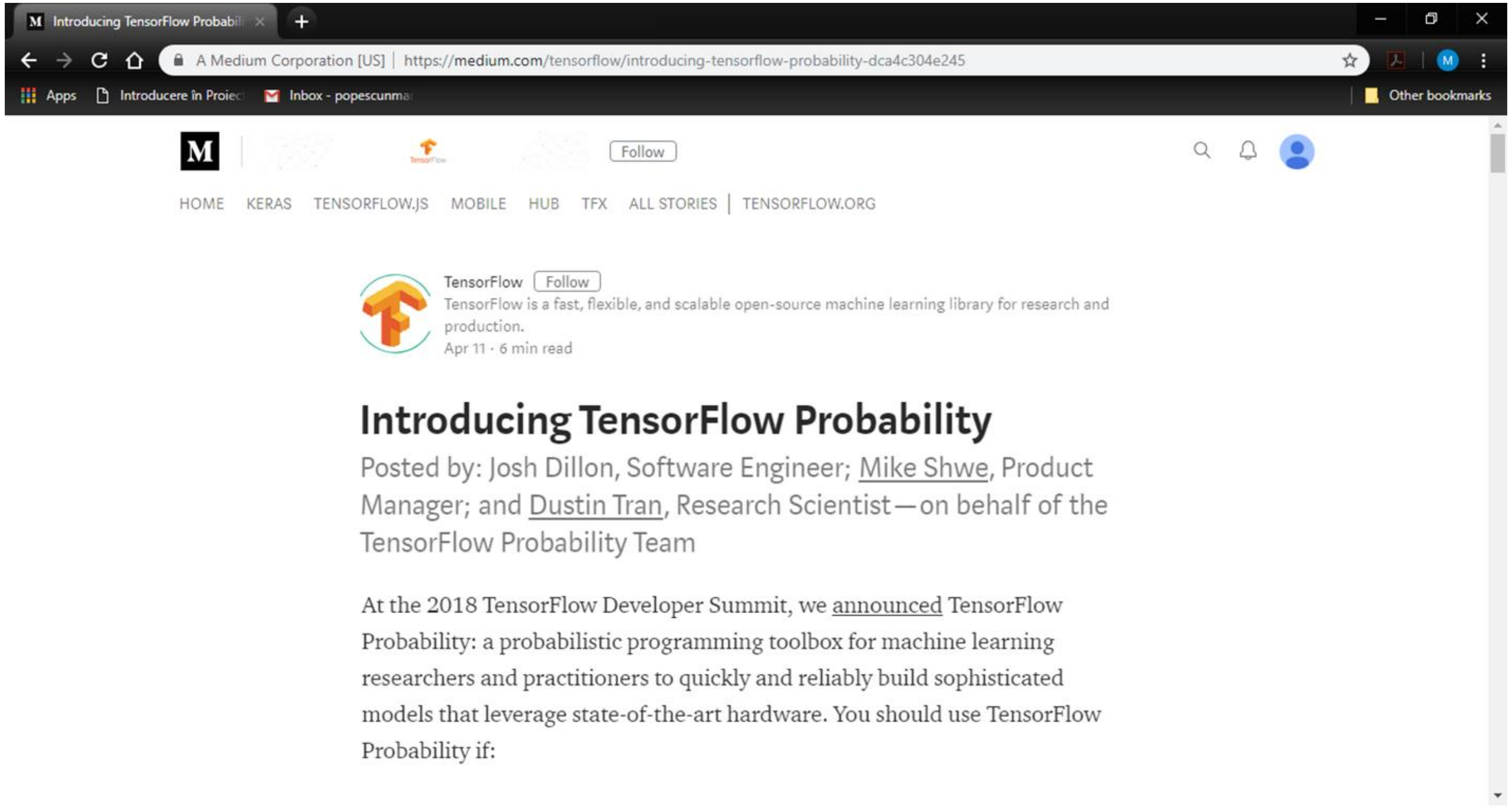
Research Scientist at Google Brain
trandustin@google.com
Blog

I am a research scientist at Google Brain. I am broadly interested in advancing science and intelligence, and where the ideas involve probability, programs, and/or neural nets.

I like to work simultaneously on fundamental research as well as systems to accelerate this research. In terms of systems, this includes Edward2 for specifying probability models as programs, Mesh TensorFlow for distributed computation, and Tensor2Tensor for deep learning research. Previously, I was a Ph.D. student at Columbia advised by David Blei and Andrew Gelman. I

Some History

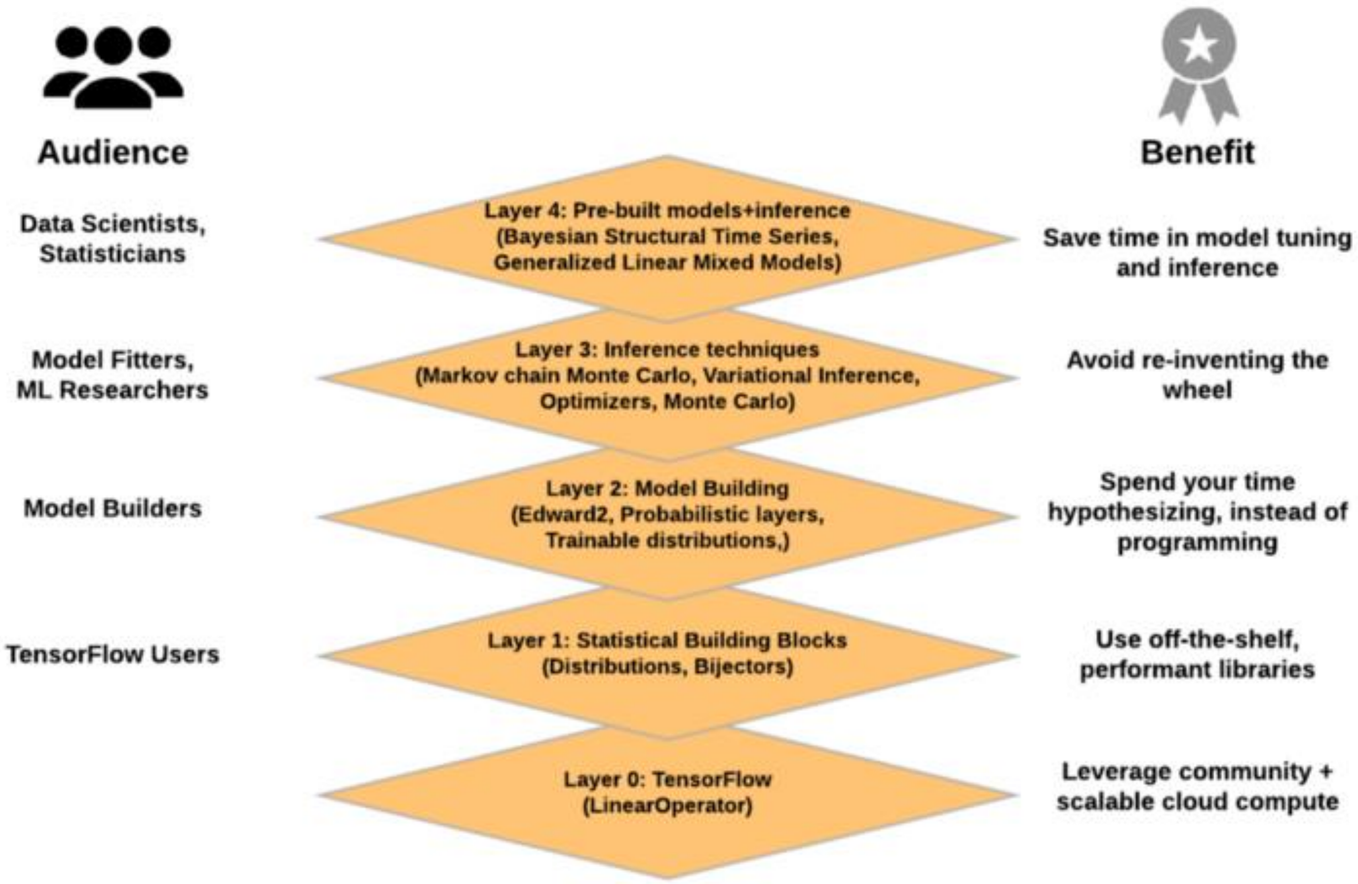




TFP

TensorFlow Probability (TFP) is a Python library built on TensorFlow that makes it easy to combine probabilistic models and deep learning on modern hardware (TPU, GPU).

It is a probabilistic programming toolbox for machine learning researchers and practitioners provides modular abstractions for probabilistic reasoning and statistical analysis in the TensorFlow ecosystem.



TFP Layer 1: Statistical Building Blocks

- Distributions (tf.contrib.distributions, tf.distributions): A large collection of probability distributions and related statistics with batch and broadcasting semantics.
- Bijectors (tf.contrib.distributions.bijectors): Reversible and composable transformations of random variables. Bijectors provide a rich class of transformed distributions, from classical examples like the log-normal distribution to sophisticated deep learning models such as masked autoregressive flows.

TFP Layer 2: Model Building

- Edward2 (tfp.edward2): A probabilistic programming language for specifying flexible probabilistic models as programs.
- Probabilistic Layers (tfp.layers): Neural network layers with uncertainty over the functions they represent, extending TensorFlow Layers.
- Trainable Distributions (tfp.trainable_distributions): Probability distributions parameterized by a single Tensor, making it easy to build neural nets that output probability distributions.

TFP Layer 3: Probabilistic Inference

- Markov chain Monte Carlo (tfp.mcmc): Algorithms for approximating integrals via sampling. Includes Hamiltonian Monte Carlo, random-walk Metropolis-Hastings, and the ability to build custom transition kernels.
- Variational Inference (tfp.vi): Algorithms for approximating integrals via optimization.
- Optimizers (tfp.optimizer): Stochastic optimization methods, extending TensorFlow Optimizers. Includes Stochastic Gradient Langevin Dynamics.
- Monte Carlo (tfp.monte_carlo): Tools for computing Monte Carlo expectations.

TFP Layer 4: Pre-made Models and Inference

- Bayesian structural time series: High-level interface for fitting time-series models (i.e., similar to R's BSTS package).
- Generalized Linear Mixed Models: High-level interface for fitting mixed-effects regression models (i.e., similar to R's lme4 package).

TensorFlow Basics

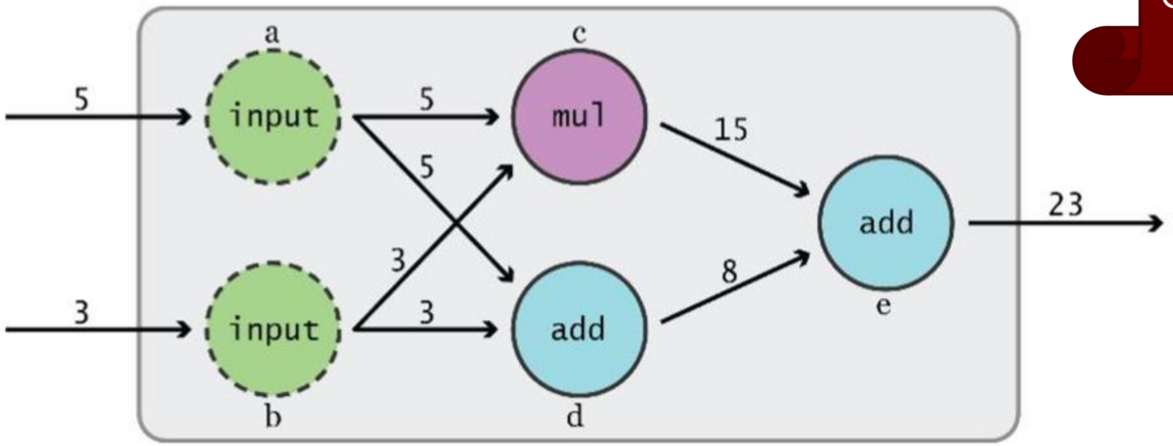
import tensorflow as tf

Data Flow Graphs

TensorFlow separates definition of computations from their execution

- Phase 1: assemble a graph
- Phase 2: use a session to execute operations in the graph.

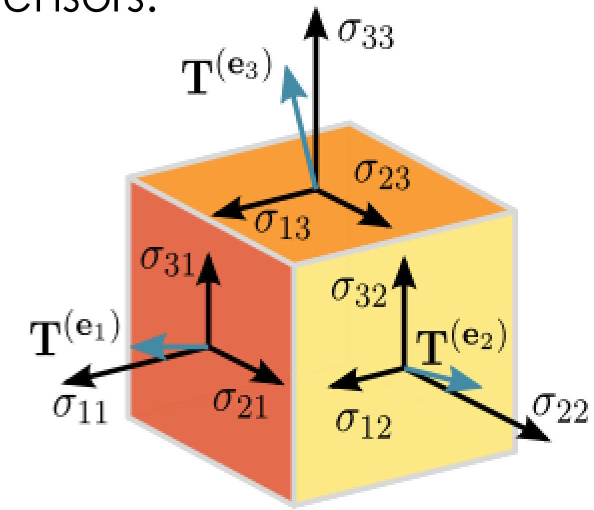
This might change in the future; in TensorFlow 2.0 the Eager execution will be the standard execution mode.



What's a tensor?

In Mathematics

A tensor is an arbitrarily complex geometric object that maps in a (multi-)linear manner geometric vectors, scalars, and other tensors to a resulting tensor. Thereby, vectors and scalars themselves are considered as the simplest tensors.



In TensorFlow

An n-dimensional array{

- 0-d / rank 0 tensor: scalar (number)
- 1-d / rank 1 tensor: vector
- 2-d / rank 2 tensor: matrix
- ...

Data Flow Graphs

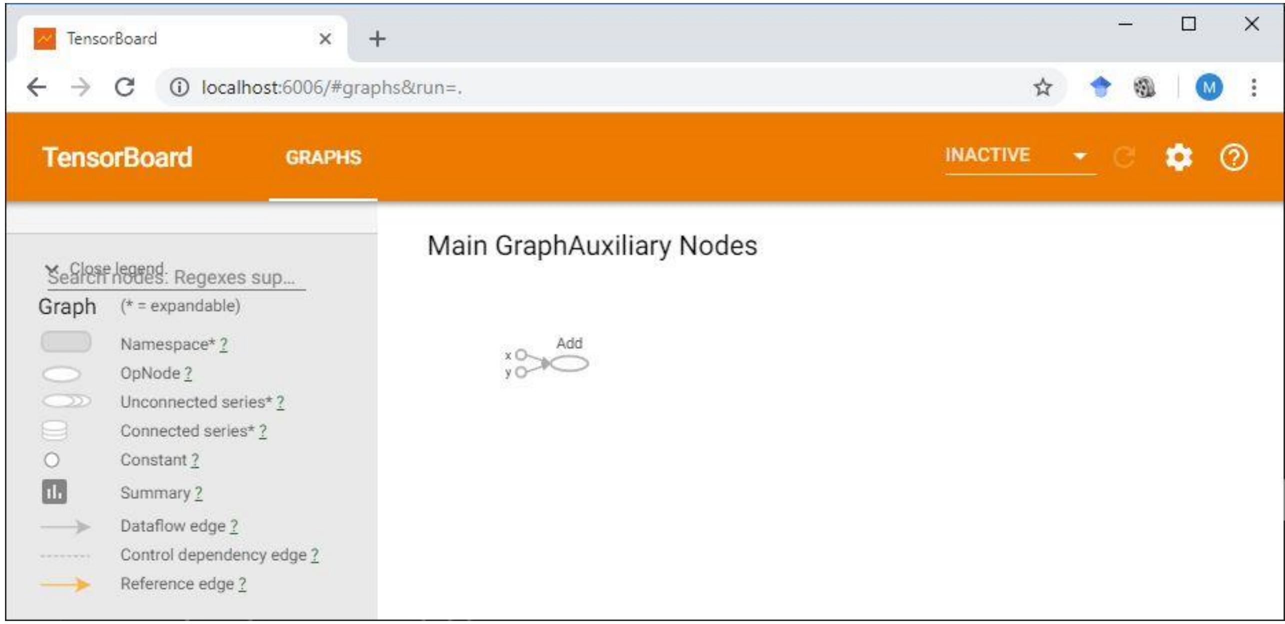
```
>>>import tensorflow as tf
>>>a = tf.add(3, 5)
```

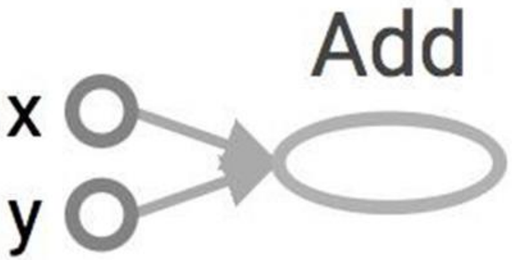
Why x, y?

TF automatically names the nodes when you don't explicitly name them.

x = 3

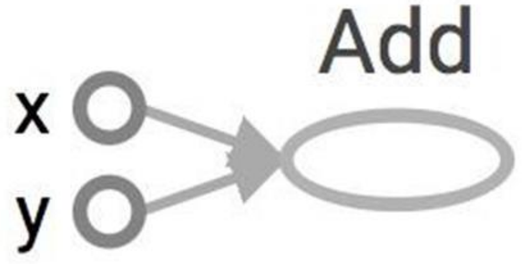
y = 5





Data Flow Graphs

```
>>>import tensorflow as tf
>>>a = tf.add(3, 5)
```



Nodes: operators, variables, and constants

Edges: tensors

Tensors are data.

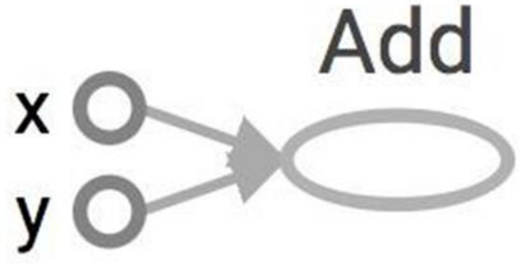
TensorFlow = tensor + flow = data + flow

Interpretation?

Data Flow Graphs

```
>>>import tensorflow as tf
>>>a = tf.add(3, 5)
>>>print(a)
Tensor("Add:0", shape=(), dtype=int32)
```

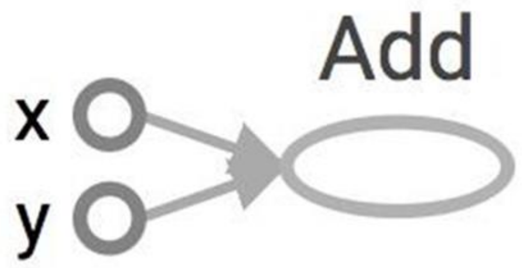
Not 8



How to get the value of a?

- Create a session, assign it to variable `sess` so we can call it later
- Within the session, evaluate the graph to fetch the value of a

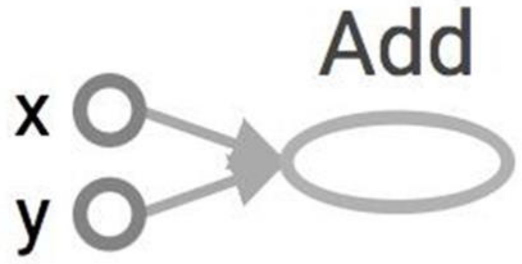
```
import tensorflow as tf
a = tf.add(3, 5)
sess = tf.Session()
print(sess.run(a))
sess.close()
```



How to get the value of a?

- Create a session, assign it to variable `sess` so we can call it later
- Within the session, evaluate the graph to fetch the value of a

```
import tensorflow as tf
a = tf.add(3, 5)
sess = tf.Session()
with tf.Session() as sess:
     print(sess.run(a))
sess.close()
```

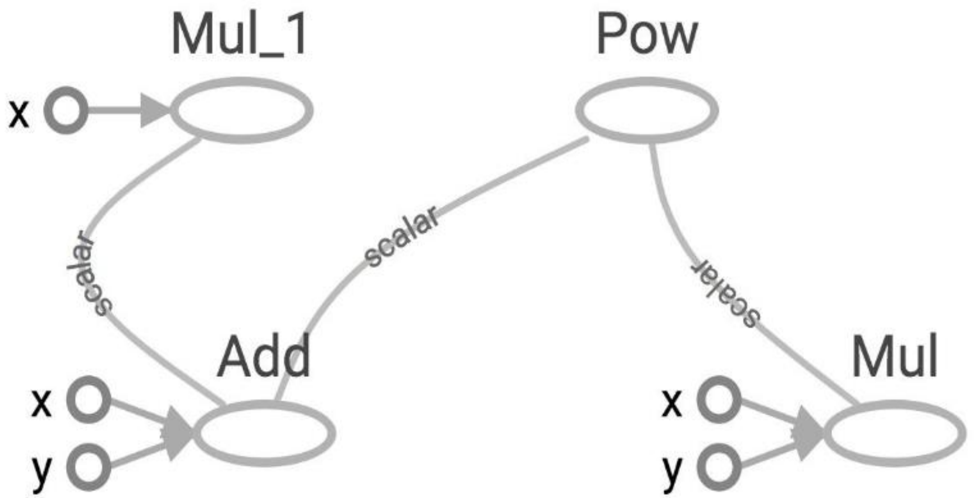


tf.Session()

- A Session object encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated.
- A TensorFlow session is used to run parts of the graph to get the variables we want.

Subgraphs

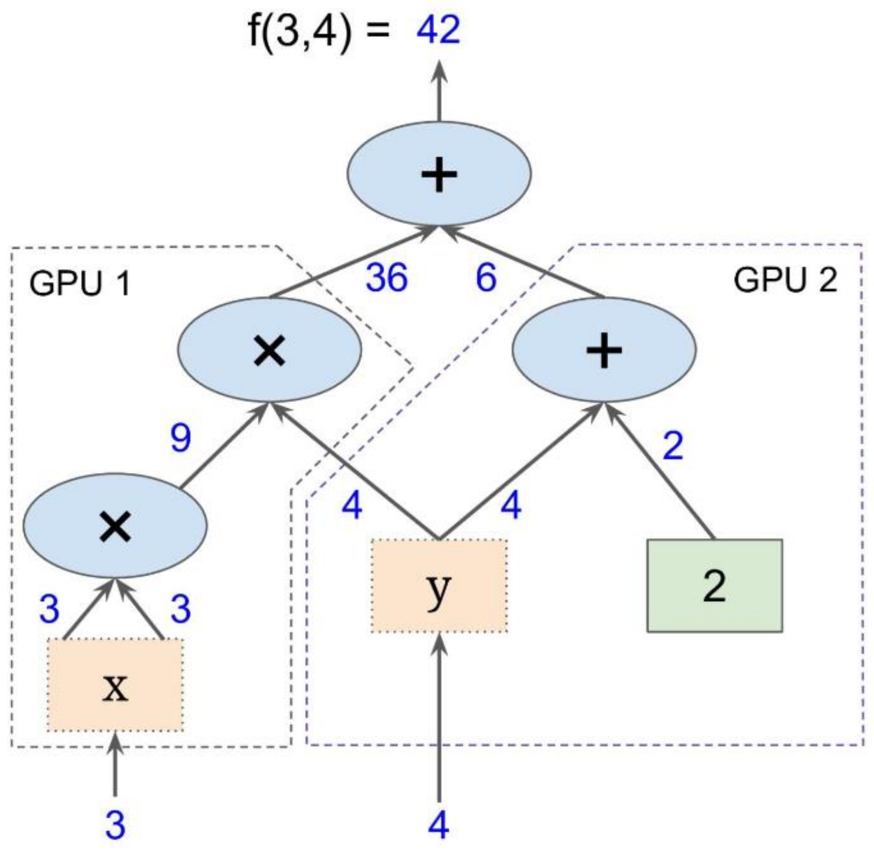
```
x = 2
y = 3
add_op = tf.add(x, y)
mul_op = tf.multiply(x, y)
useless = tf.multiply(x, add_op)
pow_op = tf.pow(add_op, mul_op)
with tf.Session() as sess:
     z = sess.run(pow_op)
```



Because we only want the value of `pow_op` and `pow_op` doesn't depend on useless, session won't compute value of useless

Subgraphs

Possible to break graphs into several chunks and run them parallelly across multiple CPUs, GPUs, TPUs, or other devices



Visualization with TensorBoard

```
import tensorflow as tf

a = tf.constant(2, name = 'a')
b = tf.constant(3, name = 'b')
x = tf.add(a, b, name = 'x')

writer = tf.summary.FileWriter('./graphs', tf.get_default_graph())
with tf.Session() as sess:
     # writer = tf.summary.FileWriter('./graphs', sess.graph)
     print(sess.run(x))
writer.close()
```

Visualization with TensorBoard

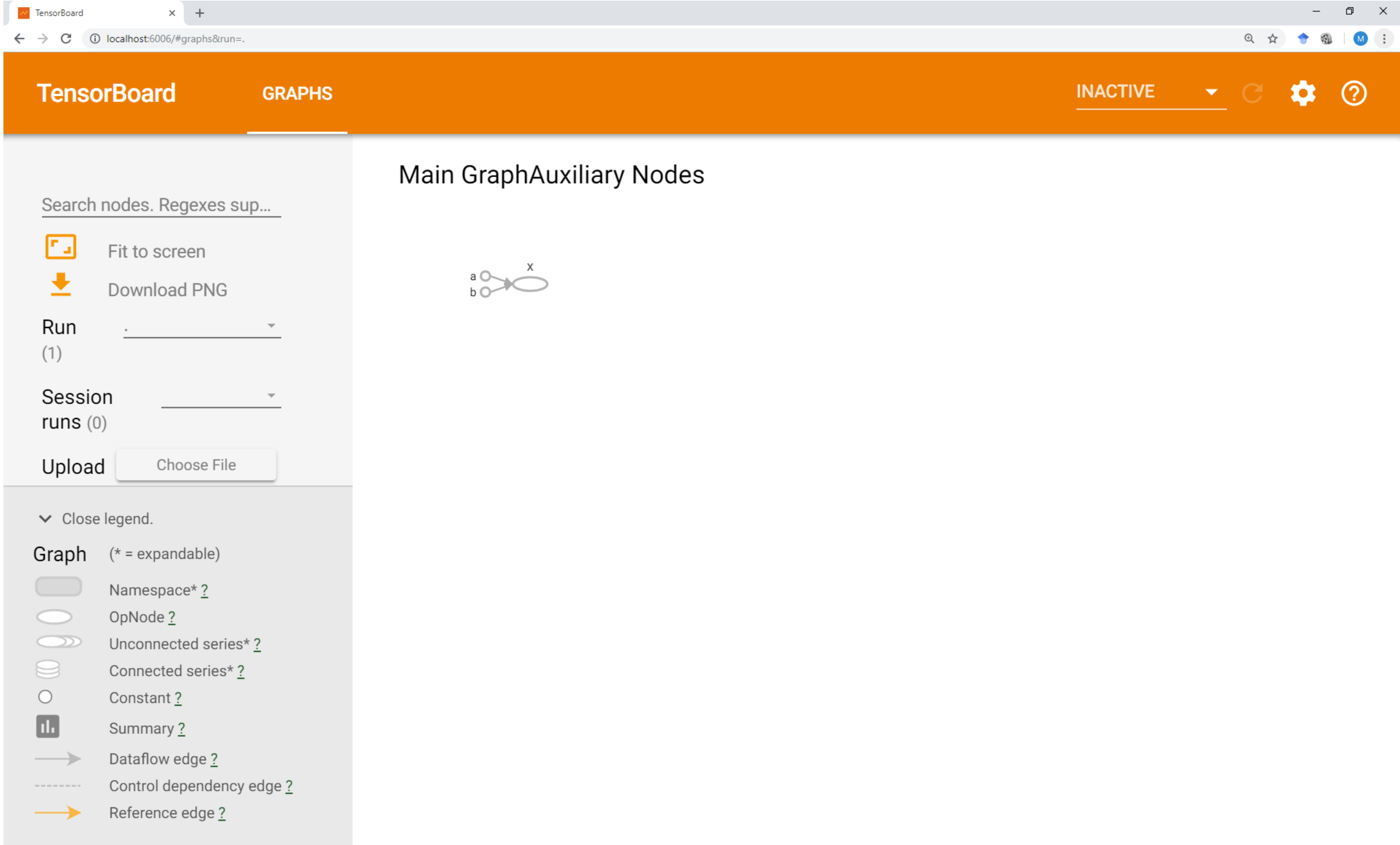
Run:

```
python yourprogram.py
tensorboard --logdir="./graphs" --port 6006
```

Then open your browser and go to:

```
http://localhost:6006/
```

Visualization with TensorBoard



Tensors

`tf.Tensor`

Internally, TensorFlow represents tensors as n-dimensional arrays of base datatypes.

A `tf.Tensor` has the following properties:

- a data type (`float32`, `int32`, or `string`, for example)
- a shape

Shape

The TensorFlow documentation uses three notational conventions to describe tensor dimensionality: rank, shape, and dimension number.

Rank	Shape	Dimension number	Example
0	[]	0-D	A 0-D tensor. A scalar.
1	[D0]	1-D	A 1-D tensor with shape [5].
2	[D0, D1]	2-D	A 2-D tensor with shape [3, 4].
3	[D0, D1, D2]	3-D	A 3-D tensor with shape [1, 4, 3].
n	[D0, D1, ... Dn-1]	n-D	A tensor with shape [D0, D1, ... Dn-1].

Shapes can be represented via Python lists / tuples of ints, or with the `tf.TensorShape`.

Shape

- Getting a `tf.Tensor` object's shape:

```
zeros = tf.zeros(my_matrix.shape[1]) # makes a vector of zeros with the same size
                                     # as the number of columns in a given matrix
```

- Changing the shape of a `tf.Tensor`:

```
rank_three_tensor = tf.ones([3, 4, 5])
matrix = tf.reshape(rank_three_tensor, [6, 10])  # Reshape existing content into
                                                 # a 6x10 matrix
```

Data Type

- To inspect a `tf.Tensor`'s data type use the `Tensor.dtype` property.

- It is possible to cast `tf.Tensor`s from one datatype to another using `tf.cast`:

```
# Cast a constant integer tensor into floating point.
float_tensor = tf.cast(tf.constant([1, 2, 3]), dtype=tf.float32)
```

Creating Tensors: Constants

Constants

```
a = tf.constant(3, name=‘a’)

a = tf.constant([2, 2], name='a’)
a = tf.constant([[0, 1], [2, 3]], name=‘a')
```

Tensors filled with a specific value

```
tf.zeros(shape, dtype=tf.float32, name=None)
```

Creates a tensor of shape and all elements will be zeros

```
tf.zeros([2, 3], tf.int32) ==> [[0, 0, 0], [0, 0, 0]]
```

Tensors filled with a specific value

```
tf.zeros_like(input_tensor, dtype=None, name=None, optimize=True)
```

Creates a tensor of shape and type (unless type is specified) as the input_tensor but all elements are zeros.

```
# input_tensor is [[0, 1], [2, 3], [4, 5]]
tf.zeros_like(input_tensor) ==> [[0, 0], [0, 0], [0, 0]]
```

Tensors filled with a specific value

```
tf.ones(shape, dtype=tf.float32, name=None)

tf.ones_like(input_tensor, dtype=None, name=None, optimize=True)
```

Tensors filled with a specific value

```
tf.fill(dims, value, name=None)
```

```
Creates a tensor filled with a scalar value.
```

```
tf.fill([2, 3], 8) ==> [[8, 8, 8], [8, 8, 8]]
```

Constants as Sequences

```
tf.lin_space(start, stop, num, name=None)
tf.lin_space(10.0, 13.0, 4) ==> [10. 11. 12. 13.]


tf.range(start, limit=None, delta=1, dtype=None, name='range')
tf.range(3, 18, 3) ==> [3 6 9 12 15]
tf.range(5) ==> [0 1 2 3 4]
```

Randomly Generated Constants

```
tf.random_normal
tf.truncated_normal
tf.random_uniform
tf.random_shuffle
tf.random_crop
tf.multinomial
tf.random_gamma
```

Creating Tensors: Operations

Subtract

Classes

tensorflow::ops::Abs	Computes the absolute value of a tensor.
tensorflow::ops::AccumulateNV2	Returns the element-wise sum of a list of tensors.
tensorflow::ops::Acos	Computes acos of x element-wise.
tensorflow::ops::Acosh	Computes inverse hyperbolic cosine of x element-wise.
tensorflow::ops::Add	Returns x + y element-wise.
tensorflow::ops::AddN	Add all input tensors element wise.
tensorflow::ops::AddV2	Returns x + y element-wise.
tensorflow::ops::All	Computes the "logical and" of elements across dimensions of a tensor.
tensorflow::ops::Angle	Returns the argument of a complex number.
tensorflow::ops::Any	Computes the "logical or" of elements across dimensions of a tensor.
tensorflow::ops::ApproximateEqual	Returns the truth value of abs(x-y) < tolerance element-wise.
tensorflow::ops::ArgMax	Returns the index with the largest value across dimensions of a tensor.

Contents

`l2_normalize(...)` : Normalizes along dimension `axis` using an L2 norm. (deprecated arguments)

`logdet(...)` : Computes log of the determinant of a hermitian positive definite matrix.

`logm(...)` : Computes the matrix logarithm of one or more square matrices:

`lstsq(...)` : Solves one or more linear least-squares problems.

`matmul(...)` : Multiplies matrix `a` by matrix `b`, producing `a` * `b`.

`norm(...)` : Computes the norm of vectors, matrices, and tensors. (deprecated arguments)

`qr(...)` : Computes the QR decompositions of one or more matrices.

`set_diag(...)` : Returns a batched matrix tensor with new batched diagonal values.

`slogdet(...)` : Computes the sign and the log of the absolute value of the determinant of

`solve(...)` : Solves systems of linear equations.

`svd(...)` : Computes the singular value decompositions of one or more matrices.

`tensor_diag(...)` : Returns a diagonal tensor with a given diagonal values.

`tensor_diag_part(...)` : Returns the diagonal part of the tensor.

tensorflow::ops::StopGradient	Stops gradient computation.
tensorflow::ops::StridedSlice	Return a strided slice from `input`.
tensorflow::ops::StridedSliceAssign	Assign`value` to the sliced l-value reference of `ref`.
tensorflow::ops::StridedSliceGrad	Returns the gradient of `StridedSlice`.
tensorflow::ops::Tile	Constructs a tensor by tiling a given tensor.
tensorflow::ops::Transpose	Shuffle dimensions of x according to a permutation.
tensorflow::ops::Unique	Finds unique elements in a 1-D tensor.
tensorflow::ops::UniqueV2	Finds unique elements along an axis of a tensor.
tensorflow::ops::UniqueWithCounts	Finds unique elements in a 1-D tensor.
tensorflow::ops::UniqueWithCountsV2	Finds unique elements along an axis of a tensor.
tensorflow::ops::UnravelIndex	Converts a flat index or array of flat indices into a tuple of.
tensorflow::ops::Unstack	Unpacks a given dimension of a rank-R tensor into `num` rank-`(R-1)` tensors.
tensorflow::ops::Where	Returns locations of nonzero / true values in a tensor.
tensorflow::ops::ZerosLike	Returns a tensor of zeros with the same shape and type as x.

Control Flow Ops

Contents

Summary

Classes	
tensorflow::ops::Abort	Raise a exception to abort the process when called.
tensorflow::ops::ControlTrigger	Does nothing.
tensorflow::ops::LoopCond	Forwards the input to the output.
tensorflow::ops::Merge	Forwards the value of an available tensor from `inputs` to `output`.
tensorflow::ops::NextIteration	Makes its input available to the next iteration.
tensorflow::ops::RefNextIteration	Makes its input available to the next iteration.
tensorflow::ops::RefSelect	Forwards the `index`th element of `inputs` to `output`.

Variables

tf.Variable class

```
# create variables with tf.Variable
s = tf.Variable(2, name="scalar")
m = tf.Variable([[0, 1], [2, 3]], name="matrix")
W = tf.Variable(tf.zeros([784,10]))

# create variables with tf.get_variable
s = tf.get_variable("scalar", initializer=tf.constant(2))
m = tf.get_variable("matrix", initializer=tf.constant([[0, 1], [2, 3]]))
W = tf.get_variable("big_matrix", shape=(784, 10), initializer=tf.zeros_initializer())
```

`tf.Variable` holds several ops

```
x = tf.Variable(...)

x.initializer # init op
x.value() # read op
x.assign(...) # write op
x.assign_add(...) # and more
```

You have to initialize your variables

```
import tensorflow as tf

W = tf.get_variable("big_matrix", shape=(784, 10), initializer=tf.zeros_initializer())

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

```
FailedPreconditionError: Attempting to use uninitialized value Variable
```

Initializer is an op. You need to execute it within the context of a session

You have to initialize your variables

```
# The easiest way is initializing all variables at once:
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

# Initialize only a subset of variables:
with tf.Session() as sess:
    sess.run(tf.variables_initializer([a, b]))

# Initialize a single variable
W = tf.Variable(tf.zeros([784,10]))
with tf.Session() as sess:
    sess.run(W.initializer)
```

Eval() a variable

```
# W is a random 700 x 10 variable object
W = tf.Variable(tf.truncated_normal([700, 10]))
with tf.Session() as sess:
     sess.run(W.initializer)
     print(W)
     print(W.eval())

Tensor("Variable/read:0", shape=(700, 10), dtype=float32)
[[-0.76781619 -0.67020458  1.15333688 ..., -0.98434633 -1.25692499
  -0.90904623]
 [-0.36763489 -0.65037876 -1.52936983 ...,  0.19320194 -0.38379928
   0.44387451]
 [ 0.12510735 -0.82649058  0.4321366  ..., -0.3816964   0.70466036
   1.33211911]
 ...,
```

tf.Variable.assign()

```
W = tf.Variable(10)
W.assign(100)
with tf.Session() as sess:
    sess.run(W.initializer)
    print(W.eval())
```

```
10
```

?

W.assign(100) creates an assign op. That op needs to be executed in a session to take effect.

```
W = tf.Variable(10)
assign_op = W.assign(100)
with tf.Session() as sess:
    sess.run(W.initializer)
    sess.run(assign_op)
    print(W.eval())
```

```
100
```

☺

tf.Variable.assign()

```
# create a variable whose original value is 2
my_var = tf.Variable(2, name="my_var")

# assign a * 2 to a and call that op a_times_two
my_var_times_two = my_var.assign(2 * my_var)

with tf.Session() as sess:
    sess.run(my_var.initializer)
    sess.run(my_var_times_two)              # the value of my_var now is 4
    sess.run(my_var_times_two)              # the value of my_var now is 8
    sess.run(my_var_times_two)              # the value of my_var now is 16
```

“Hello World!”

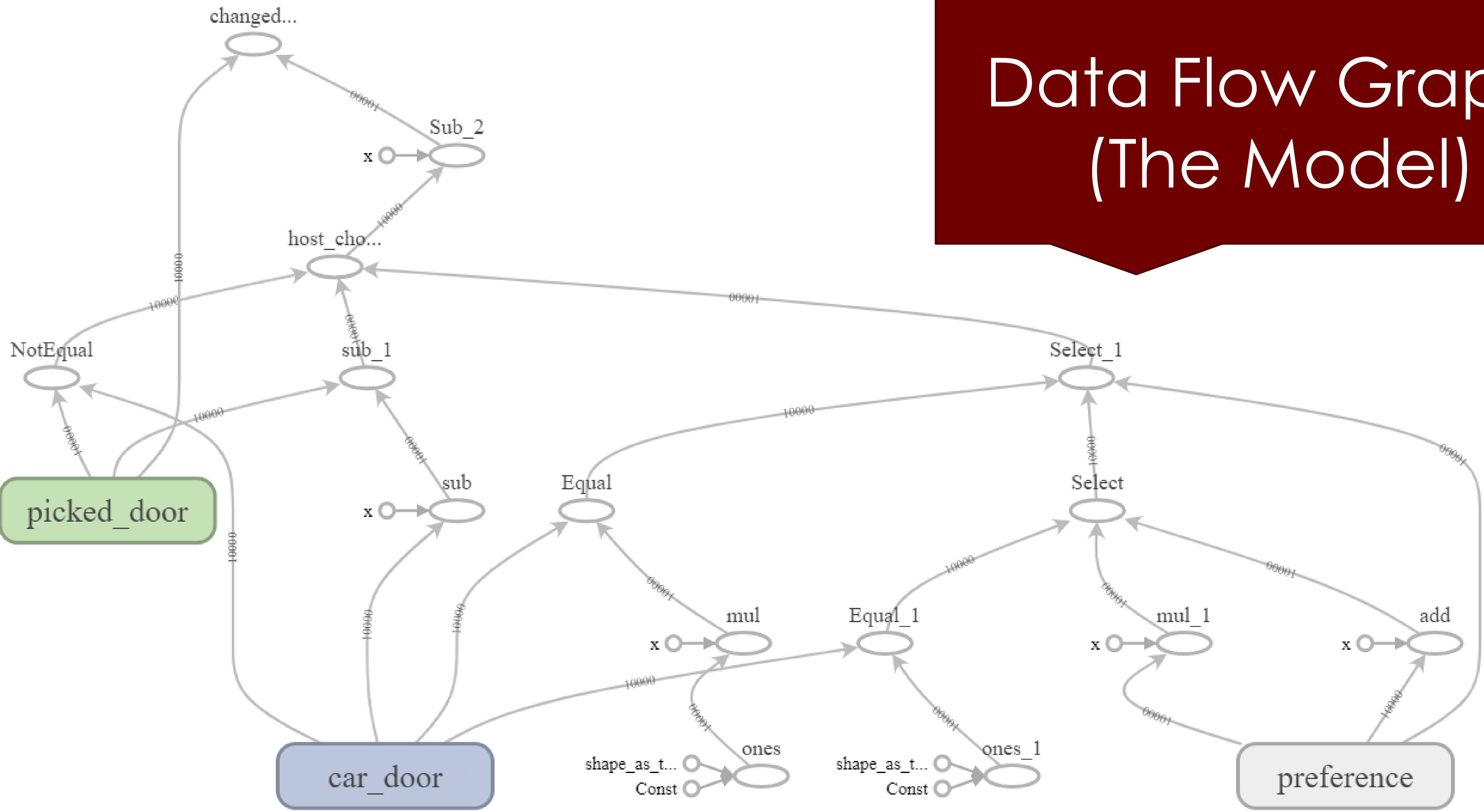
Monty Hall Problem

Suppose you're on a game show, and you're given the choice of three doors: Behind one door is a car; behind the others, goats. You pick a door, say No. 1, and the host, who knows what's behind the doors, opens another door, say No. 3, which has a goat. He then says to you, "Do you want to pick door No. 2?" Is it to your advantage to switch your choice?

TFP Solution

```
import tensorflow as tf
from tensorflow_probability import edward2 as ed

N = 10000

car_door = ed.Categorical(probs=tf.constant([1. / 3., 1. / 3., 1. / 3.]), sample_shape = N, name = 'car_door')
picked_door = ed.Categorical(probs=tf.constant([1. / 3., 1. / 3., 1. / 3.]), sample_shape = N, name = 'picked_door')
preference = ed.Bernoulli(probs=tf.constant(0.5), sample_shape = N, name = 'preference’)

host_choice = tf.where(tf.not_equal(car_door,  picked_door),
                       3 - car_door - picked_door,
                       tf.where(tf.equal(car_door, 2 * tf.ones(N, dtype=tf.int32)),
                                preference,
                                tf.where(tf.equal(car_door, tf.ones(N, dtype=tf.int32)),
                                         2 * preference,
                                         1 + preference)), name = 'host_choice’)

#changed_door = 3 - host_choice - picked_door
changed_door = tf.subtract(tf.subtract(3, host_choice), picked_door, name = 'changed_door')
```

Data Flow Graphs (The Model)



Running

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    car_door_samples, picked_door_samples, changed_door_samples = sess.run([car_door, picked_door, changed_door])

print("probability to win of a player who stays with the initial choice:", (car_door_samples == picked_door_samples).mean())
print("probability to win of a player who switches:", (car_door_samples == changed_door_samples).mean())
```

The Results

{C:\My\ProbabilisticProgrammingCourse\TFPandEdward2\11} - Far 3.0.5225 x64 Administrator

```
C:\...ogrammingCourse\TFPandEdward2\11>                                   11:06 AM
C:\...ogrammingCourse\TFPandEdward2\11>
C:\...ogrammingCourse\TFPandEdward2\11>
C:\...ogrammingCourse\TFPandEdward2\11>
C:\...ogrammingCourse\TFPandEdward2\11>
C:\...ogrammingCourse\TFPandEdward2\11>
C:\...ogrammingCourse\TFPandEdward2\11>
C:\...ogrammingCourse\TFPandEdward2\11>
C:\...ogrammingCourse\TFPandEdward2\11>
C:\...ogrammingCourse\TFPandEdward2\11>python monty_hall.py
2018-12-10 11:06:39.311511: I tensorflow/core/platform/cpu_feature_guard.cc:141]
 Your CPU supports instructions that this TensorFlow binary was not compiled to
use: AVX2
probability to win of a player who stays with the initial choice: 0.3335
probability to win of a player who switches: 0.6665

C:\...ogrammingCourse\TFPandEdward2\11>python monty_hall.py
2018-12-10 11:06:50.328939: I tensorflow/core/platform/cpu_feature_guard.cc:141]
 Your CPU supports instructions that this TensorFlow binary was not compiled to
use: AVX2
probability to win of a player who stays with the initial choice: 0.3331
probability to win of a player who switches: 0.6669

C:\...ogrammingCourse\TFPandEdward2\11>
1Left  2Right  3View..  4Edit..  5Print  6MkLink  7Find  8Histry  9Video  10
```