

# TensorFlow基础编程实验

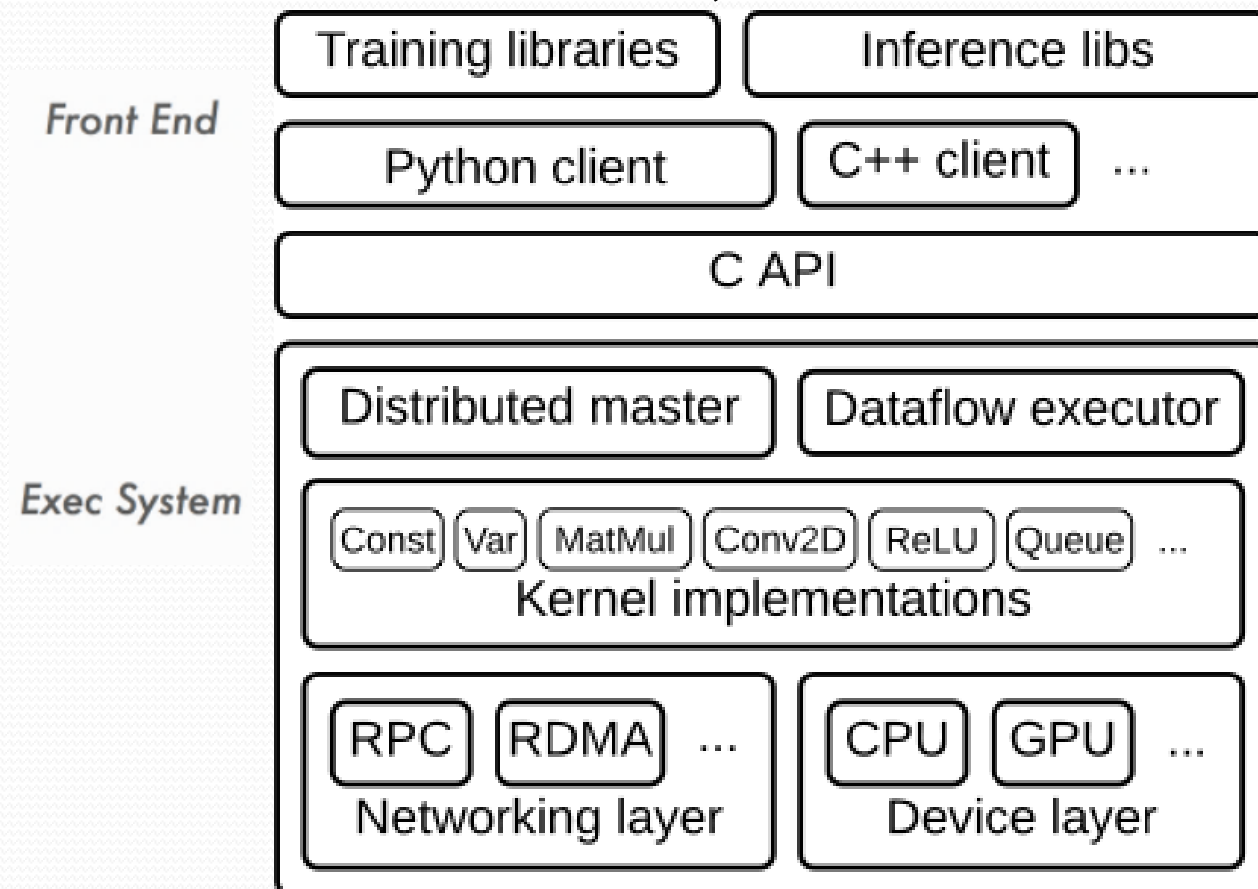
中科大软件学院 赵振刚 [gavin@ustc.edu.cn](mailto:gavin@ustc.edu.cn)

# 目录

- TensorFlow系统架构
- 编程模型
  - 节点 边 图 会话 变量 设备
- 基本变量和运算
  - 动手实践
- 线性回归原理和实践

# TensorFlow系统架构

- TensorFlow™ 是用于数值计算的开源软件库，应用层支持Python, Java, Go等语言，设备管理层支持FPGA、GPU、CPU等



# TensorFlow系统架构

- TensorFlow™ 采用符号式编程(symbolic style programming), 而非命令式编程(imperative style programming)

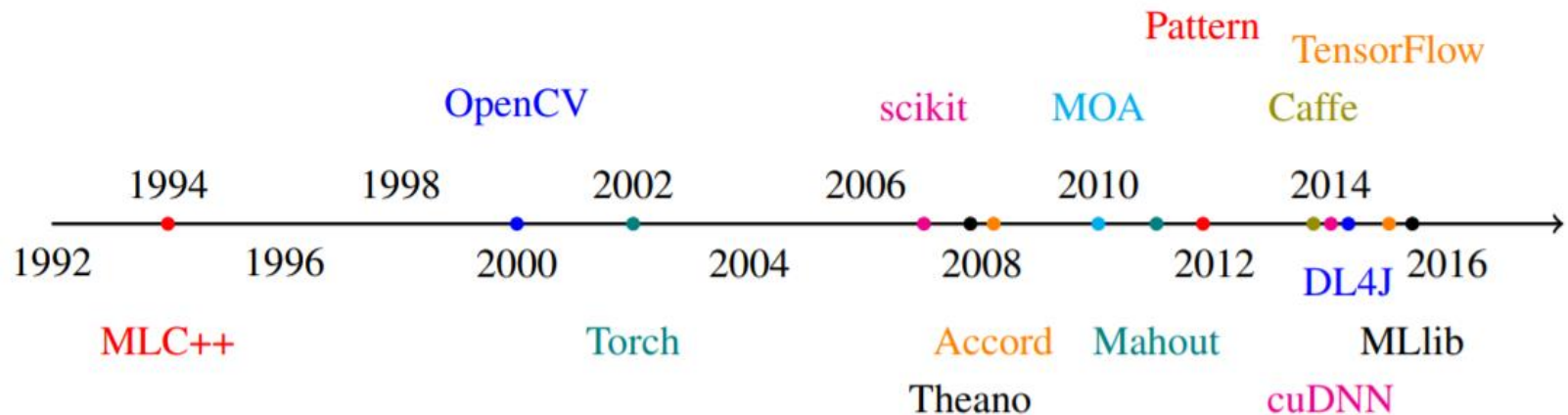


Fig. 1: A timeline showing the release of machine-learning libraries discussed in section I in the last 25 years.

# example 1

- *imperative style*

```
int a;  
a= 4+3;  
print(a);
```

- *Symbolic style*

```
import tensorflow as tf  
a = tf.add(4,3)  
print(a)
```

# what is the result?

# example 1



The image shows a Jupyter Notebook interface. At the top, the Jupyter logo is followed by the text "symbolic\_style" and "Last Checkpoint: 7 minutes ago (unsaved changes)". Below this is a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". Under the menu bar is a toolbar with icons for saving, adding a new cell, deleting a cell, copying, pasting, undo, redo, and a dropdown menu currently set to "Code". To the right of the toolbar are buttons for "CellToolbar" and a cloud icon. The main area of the notebook contains a code cell with the following text:

```
In [3]: import tensorflow as tf
a = tf.add(4,3)
print(a)
```

Below the code cell, the output is displayed:

```
Tensor("Add:0", shape=(), dtype=int32)
```

得到的是一个张量（Tensor）？

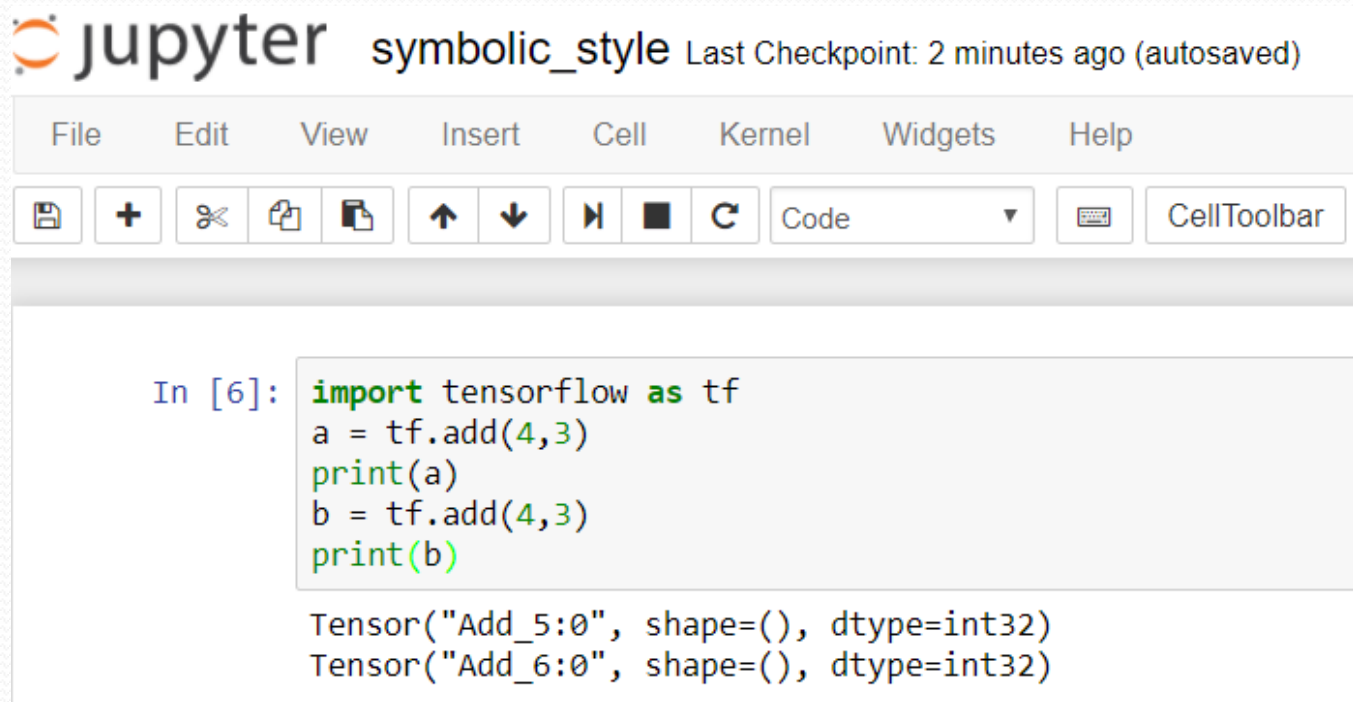
一个名字，它用于键值对的存储，及后续的检索：Add: 0

一个形状描述，描述数据的每一维度的元素个数：（）

数据类型，比如 int32

# example 1

多次执行会得到什么结果？



The image shows a Jupyter Notebook interface with the title 'symbolic\_style' and a status bar indicating 'Last Checkpoint: 2 minutes ago (autosaved)'. The interface includes a menu bar with 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. Below the menu bar is a toolbar with icons for saving, adding, deleting, copying, pasting, undo, redo, and running code. The code cell contains the following code:

```
In [6]: import tensorflow as tf
a = tf.add(4,3)
print(a)
b = tf.add(4,3)
print(b)
```

The output of the code cell is:

```
Tensor("Add_5:0", shape=(), dtype=int32)
Tensor("Add_6:0", shape=(), dtype=int32)
```

代码见课程资料

Machine\_Learning\02\_Basics\01\_Symbolic\_example\symbolic\_style\_02.ipynb

# example 1

如何使得add执行，得到运行结果？

```
In [7]: import tensorflow as tf  
a = tf.add(4,3)  
print(a)  
sess=tf.Session()  
sess.run(a)
```

Tensor("Add\_7:0", shape=(), dtype=int32)

```
Out[7]: 7
```

启动会话  
填充数据  
运行图

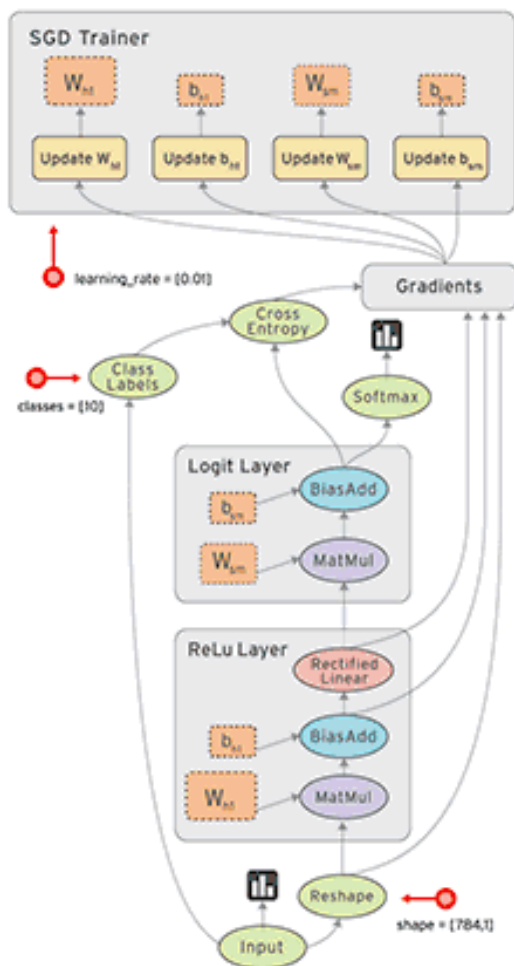


# 目录

- TensorFlow系统架构
- 编程模型
  - 边 节点 图 会话 变量 设备
- 基本变量和运算
  - 动手实践
- 线性回归原理和实践

# TensorFlow编程模型

- TensorFlow 意为“张量的流动”，采用数据流图做计算，这个图指的是由节点(node)和边(edge)组成的有向无环图(directed acycline graph, 图论术语)



# TensorFlow编程模型

## • 边

表示节点之间相互联系的张量，分为：

1. 实线----数据依赖，代表任意维度的数据，即张量数据类型如图示

数据类型	Python 类型	描述
DT_FLOAT	<code>tf.float32</code>	32 位浮点数.
DT_DOUBLE	<code>tf.float64</code>	64 位浮点数.
DT_INT64	<code>tf.int64</code>	64 位有符号整型.
DT_INT32	<code>tf.int32</code>	32 位有符号整型.
DT_INT16	<code>tf.int16</code>	16 位有符号整型.
DT_INT8	<code>tf.int8</code>	8 位有符号整型.
DT_UINT8	<code>tf.uint8</code>	8 位无符号整型.
DT_STRING	<code>tf.string</code>	可变长度的字节数组. 每一个张量元素都是一个字节数组.
DT_BOOL	<code>tf.bool</code>	布尔型.
DT_COMPLEX64	<code>tf.complex64</code>	由两个32位浮点数组成的复数: 实数和虚数.
DT_QINT32	<code>tf.qint32</code>	用于量化Ops的32位有符号整型.
DT_QINT8	<code>tf.qint8</code>	用于量化Ops的8位有符号整型.
DT_QUINT8	<code>tf.quint8</code>	用于量化Ops的8位无符号整型.

2. 虚线----控制依赖，代表操作间的次序约束，happens-before关系

# TensorFlow编程模型

- 节点

节点表示某种抽象的计算，一个操作(operation)，通常为数学运算，也可以是数据输入起点和输出终点

TensorFlow内建操作包括 下表所示

类型	示例
标量运算	Add、Sub、Mul、Div、Exp、Log、Greater、Less、Equal
向量运算	Concat、Slice、Splot、Constant、Rank、Shape、Shuffle
矩阵运算	Matmul、MatrixInverse、MatrixDeterminant
带状态的运算	Variable、Assign、AssignAdd
神经网络组件	SoftMax、Sigmoid、ReLU、Convolution2D、MaxPooling
存储、恢复	Save、Restore
队列及同步运算	Enqueue、Dequeue、MutexAcquire、MutexRelease
控制流	Merge、Switch、Enter、Leave、NextIteration

# TensorFlow编程模型

- 节点

节点实现源码在Github上可查看, tensorflow/python/ops/  
[https://github.com/tensorflow/tensorflow/blob/r1.4/tensorflow/python/ops/array\\_ops.py](https://github.com/tensorflow/tensorflow/blob/r1.4/tensorflow/python/ops/array_ops.py)

```
98  from tensorflow.python.ops import gen_array_ops
99  from tensorflow.python.ops import gen_math_ops
```

这里所依赖的模块 `gen_array_ops` 在安装路径 `Anaconda3\Lib\site-packages\tensorflow\python\ops` 下



gen\_array\_ops.py

2017/11/17 12:16

Python File

301 KB

该模块中又调用了

<https://github.com/tensorflow/tensorflow/tree/r1.4/tensorflow/core/kernels> 中具体核函数的实现

# TensorFlow编程模型

- 图

通过定义边和节点，设计各种数学操作，保存操作和元数据，即可构成一个图，例如刚才example 1的前2个代码

example 2 构建图

```
import tensorflow as tf
```

```
# 创建一个常量 op, 返回值 'matrix1' 代表这个 1x2 矩阵.
```

```
matrix1 = tf.constant([[4., 3.]])
```

```
# 创建另外一个常量 op, 返回值 'matrix2' 代表这个 2x1 矩阵.
```

```
matrix2 = tf.constant([[2.],[2.]])
```

```
# 创建一个矩阵乘法 matmul op, 把 'matrix1' 和 'matrix2' 作为输入.
```

```
# 返回值 'product' 代表矩阵乘法的结果.
```

```
product = tf.matmul(matrix1, matrix2)
```

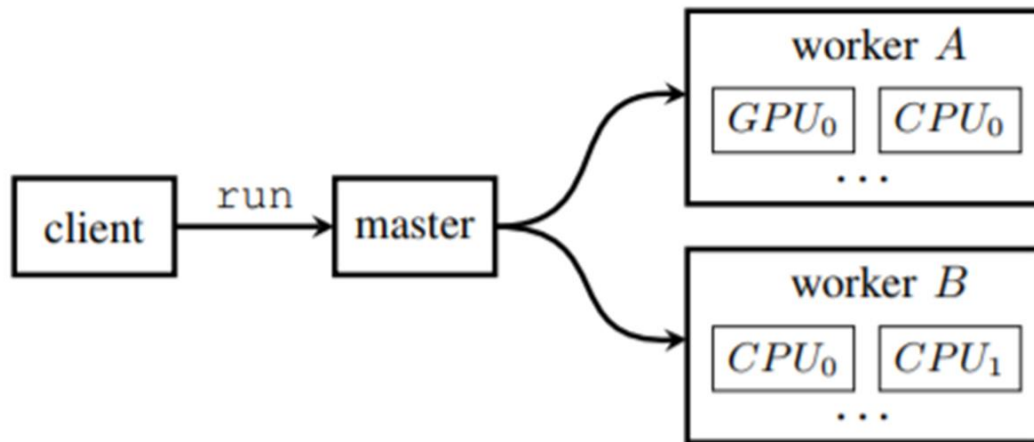
# TensorFlow编程模型

- 会话

会话是图交互的一个桥梁，Client通过Session连接TensorFlow后端的「运行时」，并启动计算图的执行过程。

一个会话可以有多个图，创建一个 Session 对象时, 如果无任何创建参数, 会话构造器将启动默认图

会话可以修改图结构，也可以往图中注入数据，传递 op 所需的全部输入(接口extend)，启动运算并输出运算结果(接口run)



# TensorFlow编程模型

- 会话

在 Python API 中, 使用一个会话 `Session` 来启动图, 并调用 `Session.run()` 方法执行操作.

在调用`Session`对象的`run`方法时, 传入一些`Tensor`, 这个过程称为填充(`feed`), 返回结果的行为称为取回(`fetch`)

example 2

# 启动默认图.

```
sess = tf.Session()
```

# 调用 `sess` 的 '`run()`' 方法, 传入 '`product`' 作为该方法的参数,

# 触发了图中三个 `op` (两个常量 `op` 和一个矩阵乘法 `op`),

# 向方法表明, 我们希望取回矩阵乘法 `op` 的输出.

```
result = sess.run(product)
```

# 返回值 '`result`' 是一个 `numpy 'ndarray'` 对象.

```
print(result)
```

# ==> `[[ 14.]]`

# 任务完成, 需要关闭会话以释放资源。

```
sess.close()
```

代码见课程资料

`Machine_Learning\02_Basics\02_Graph_example\Graph_concept.ipynb`

Session源码

<https://github.com/tensorflow/tensorflow/blob/r1.4/tensorflow/python/client/session.py>



# TensorFlow编程模型

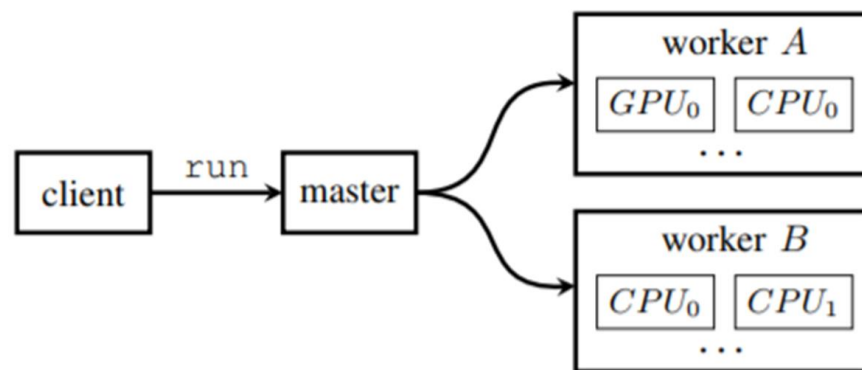
- 设备

设备(device)是指用来运算的实际处理器硬件单元, 可以是CPU, GPU, FPGA, TPU等

为了实现分布式执行操作, 可明确指定操作在哪个设备上执行, 以充分利用计算资源

example 3

```
with tf.Session() as sess:  
    with tf.device("/GPU:0"):  
        matrix1 = tf.constant([[4., 3.]])  
        matrix2 = tf.constant([2.],[2.])  
        product = tf.matmul(matrix1,matrix2)
```

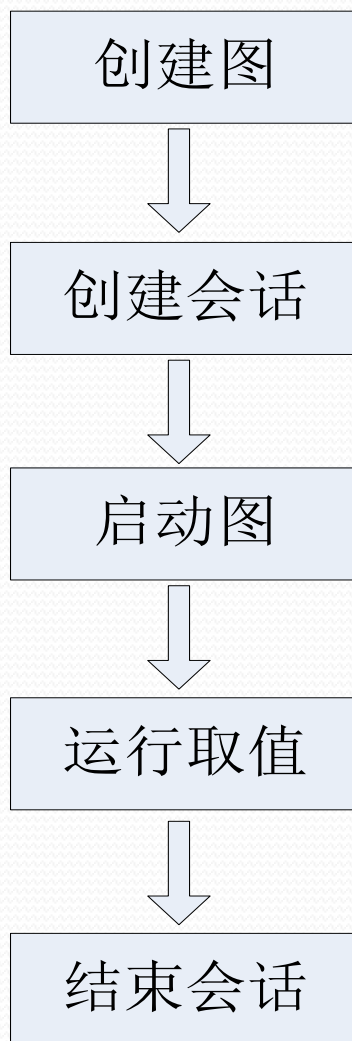


device源码

[github.com/tensorflow/tensorflow/blob/r1.4/tensorflow/python/framework/device.py](https://github.com/tensorflow/tensorflow/blob/r1.4/tensorflow/python/framework/device.py)

# TensorFlow编程模型

- 一般流程



# TensorFlow编程模型

- 变量Variable

一种特殊的数据(可理解为标量), 在图中有固定的位置, 不像Tensor可以流动, 使用`tf.Variable()`构造函数  
常用于维护图执行过程中的状态信息, 需要它来保持和更新参数值, 一般需要动态调整

## Example 4

# 创建1个变量

```
W1 = tf.Variable(1, name="coef")
```

# 创建一个张量

```
a = tf.constant(3.0)
```

# TensorFlow编程模型

- 内核kernel





前述节点(操作, op)时提到其源码引用了核函数, 内核kernel即是对特定设备(CPU,GPU)上一种具体操作的实现

1个操作可以有多个内核

除了TensorFlow自带操作外, 用户可自定义操作, 需要把新操作和内核通过注册的方式添加到系统中:

- (1)在C++文件(\*\_ops.cc)中按操作规范注册新操作
- (2)在C++文件(\*\_kernel.cc)中给出操作具体代码
- (3) 测试操作, 在应用前端中调用该操作(编译后生成\*\_ops.so库文件)

<https://github.com/tensorflow/tensorflow/tree/r1.4/tensorflow/core/kernels>

er  Machine Learning  Zigbee  Robotics  MNIST

 [argmax\\_op.cc](#)

Add ability for argmax to output int32 indices. Default remains int64.

 [argmax\\_op.h](#)

Add ability for argmax to output int32 indices. Default remains int64.

 [argmax\\_op\\_gpu.cu.cc](#)

Add ability for argmax to output int32 indices. Default remains int64.

# TensorFlow源码

📁 c	Internal private header file with eager C struct definitions.
📁 cc	Merge commit for internal changes
📁 compiler	Resolve //tensorflow relative to tensorflow repo so that tfcompile.bz...
📁 contrib	Fix broken code tag in tf.contrib.data README (#14195)
📁 core	Updating version string for 1.4.1 release. (#14634)
📁 docs_src	Updating version string for 1.4.1 release. (#14634)
📁 examples	Update word2vec_basic.py (#13531)
📁 g3doc	Merge changes from github.
📁 go	Merge commit for internal changes
📁 java	Remaining cherry-picks for 1.4.0rc1 (#13700)
📁 python	Remove name_scope from convolutional calls. (#14044)
📁 stream_executor	Merge commit for internal changes
📁 third_party/mpi	Merge changes from github.
📁 tools	Updating version string for 1.4.1 release. (#14634)
📁 user_ops	Use "nullptr" for null pointer values
📄 .clang-format	Add a .clang-format file for automatically formatting .cc/.h files.
📄 BUILD	Remaining cherry-picks for 1.4.0rc1 (#13700)
📄 __init__.py	Generalize LazyLoader for use by ffmpeg
📄 tensorflow.bzl	MKL-DNN open source integration. (#13135)
📄 tf_exported_symbols.lds	Merge changes from github.
📄 tf_version_script.lds	Merge changes from github.

# TensorFlow源码 核心目录

- **common\_runtime** 该目录下包含了tensorflow中session执行的通用逻辑流程。
- **distributed\_runtime** tensorflow 与分布式相关的执行逻辑。
- **graph** tensorflow图相关操作的逻辑。由于tensorflow中的数据计算本质上是一个图状结构的计算流程，该过程中存在将图进行切分并且并行化执行的可能性。该目录下的代码逻辑即为对图数据进行结构化定义并进行拆分的相关内容。
- **framework** 该目录下对tensorflow进行计算过程中的通用组件进行了定义和实现。
- **kernels** 对tensorflow中各个单步操作的具体实现。该目录中共有约470个文件，其中414个文件和op相关，该目录下包含了大量的tensorflow中单步操作的实现方式。(如Variable ())
- **ops** 对kernel/ 下的op进行注册和对外声明。
- **models** 实现了几个tensorflow支持的计算模型，该部分代码由python实现。
- **util/lib** 一些公用的调用方法。
- **protobuf** tensorflow下各个模块间进行数据传输的数据结构定义，通过proto进行配置实现。
- **user\_ops** 用户可进行编写自己的op并添加到该目录。

# 目录

- TensorFlow系统架构
- 编程模型
  - 边 节点 图 会话 变量 设备
- 基本变量和运算
  - 动手实践
- 线性回归原理和实践

# 基本变量和运算

- 变量Variable

创建变量 `tf.Variable()`

初始化变量 `#tf.initialize_all_variables()` not support after 2017-03-02  
`tf.global_variables_initializer()`

# 初始化也是一个图

```
import tensorflow as tf;

A = tf.Variable(tf.constant(0.0), dtype=tf.float32)
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    print(sess.run(A))
```

0.0



# 基本变量和运算

- 变量Variable

变量赋值

`tf.assign(A, new_number)`

```
import tensorflow as tf
# 创建一个变量, 初始化为标量 0. 初始化定义初值
A = tf.Variable(0, name="counter")

# 创建一个 op, 其作用是使 A 增加 2
new_value = tf.add(A, 2)
update = tf.assign(A, new_value)

# 启动图 变量初始化
init_op = tf.global_variables_initializer()

# 启动默认图, 运行 op
with tf.Session() as sess:
    sess.run(init_op)

# 打印初始值
# with tf.Session() as sess:
#     print(sess.run(A))

# 重新赋值, 更新并打印
for _ in range(3):
    sess.run(update)
    print(sess.run(A))
```

1个会话中  
多次调用方法

# 基本变量和运算

- 变量Variable

占位符 `tf.placeholder()`

```
import tensorflow as tf
#定义‘符号’变量, 也称为占位符
a = tf.placeholder("float")
b = tf.placeholder("float")

y = tf.multiply(a, b) #构造一个op 节点

sess = tf.Session()#建立会话
#运行会话, 输入数据, 并计算节点, 同时打印结果
print(sess.run(y, feed_dict={a: 3, b: 3}))
#任务完成, 关闭会话.
sess.close()
```

9.0

会话填充时  
给变量赋值

# 基本变量和运算

## ● 张量Tensor

张量可表示所有的数据，可以把一个张量想象成一个n维的数组或列表，一个张量有一个静态类型和动态类型的维数

✓ **阶**: 在TensorFlow系统中，张量的维数来被描述为阶

阶	数学实例	Python 例子
0	纯量 (只有大小)	<code>s = 483</code>
1	向量(大小和方向)	<code>v = [1.1, 2.2, 3.3]</code>
2	矩阵(数据表)	<code>m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]</code>
3	3阶张量 (数据立体)	<code>t = [[[2], [4], [6]], [[8], [10], [12]], [[14], [16], [18]]]</code>

✓ **形状**:TensorFlow文档中使用了三种记号来描述张量的维度：阶，形状及维数

阶	形状	维数	实例
0	[ ]	0-D	一个 0维张量. 一个纯量.
1	[D0]	1-D	一个1维张量的形式[5].
2	[D0, D1]	2-D	一个2维张量的形式[3, 4].
3	[D0, D1, D2]	3-D	一个3维张量的形式 [1, 4, 3].
n	[D0, D1, ... Dn]	n-D	一个n维张量的形式 [D0, D1, ... Dn].

# 基本变量和运算

## ● 张量Tensor

### ✓ 类 及其方法

操作	描述
<code>class tf.Tensor</code>	表示一个由操作节点 <code>op</code> 产生的值， <code>TensorFlow</code> 程序使用 <code>tensor</code> 数据结构来代表所有的数据，计算图中，操作间传递的数据都是 <code>tensor</code> ，一个 <code>tensor</code> 是一个符号 <code>handle</code> ，里面并没有表示实际数据，而相当于数据流的载体
<code>tf.Tensor.dtype</code>	<code>tensor</code> 中数据类型
<code>tf.Tensor.name</code>	该 <code>tensor</code> 名称
<code>tf.Tensor.value_index</code>	该 <code>tensor</code> 输出外 <code>op</code> 的 <code>index</code>
<code>tf.Tensor.graph</code>	该 <code>tensor</code> 所处所在的图
<code>tf.Tensor.op</code>	产生该 <code>tensor</code> 的 <code>op</code>
<code>tf.Tensor.consumers()</code>	返回使用该 <code>tensor</code> 的 <code>op</code> 列表
<code>tf.Tensor.eval(feed_dict=None, session=None)</code>	在会话中求 <code>tensor</code> 的值 需要使用 <code>with sess.as_default()</code> 或者 <code>eval(session=sess)</code>
<code>tf.Tensor.get_shape()</code>	返回用于表示 <code>tensor</code> 的 <code>shape</code> 的类 <code>TensorShape</code>
<code>tf.Tensor.set_shape(shape)</code>	更新 <code>tensor</code> 的 <code>shape</code>
<code>tf.Tensor.device</code>	设置计算该 <code>tensor</code> 的设备

# 基本变量和运算

- 张量Tensor

获取张量形状 `tf.Tensor.get_shape()`

```
In [1]: import tensorflow as tf  
c = tf.constant([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])  
print(c.get_shape())  
  
(2, 3)
```

代码见课程资料

Machine\_Learning\02\_Basics\03\_Ops\04\_Tensor\_shape\_constant.ipynb

# 基本变量和运算

## ● 张量Tensor

获取张量形状 `tf.Tensor.get_shape()`

```
In [6]: import matplotlib.image as mpimg
import tensorflow as tf

filename = "example_image.jpg"
image = mpimg.imread(filename)

#创建tensorflow变量
x = tf.Variable(image, name='x')

model = tf.global_variables_initializer()

with tf.Session() as session:
    session.run(model)
    result = session.run(x)
    #image.set_shape([28, 28, 3])
    print(x.get_shape())
```

(201, 268, 4)



# 基本变量和运算

## ● 数学运算

Tf.add()    tf.subtract()    tf.multiply()    tf.div()    tf.exp()    tf.log()    tf.equal()

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

x = tf.Variable([1.0, 2.0])
a = tf.constant([3.0, 3.0])

# Initialize 'x' using the run() method of its initializer op.
x.initializer.run()

# Add an op to subtract 'a' from 'x'. Run it and print the result
sub = tf.subtract(x, a)
print(sub.eval())

# Close the Session when we're done.
sess.close()

[-2. -1.]
```

# 基本变量和运算

## ● 数学运算

Tf.add()    tf.subtract()    tf.multiply()    tf.div()    tf.exp()    tf.log()    tf.equal()

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

x = tf.Variable([1.0, 2.0])
a = tf.constant([3.0, 3.0])

# Initialize 'x' using the run() method of its initializer op.
x.initializer.run()

# Add an op to subtract 'a' from 'x'. Run it and print the result
sub = tf.div(x, a)
print(sub.eval())

# Close the Session when we're done.
sess.close()

[ 0.33333334  0.66666669]
```



# 基本变量和运算

## ● 数组运算

tf.concat()    tf.slice()    tf.split()    tf.rank()    tf.shape()    tf.shuffle()

e.g. `tf.concat(name='concat', values, concat_dim)` 连接两个矩阵,  
`concat_dim`: 是一个数, 表明在哪一维上连接, `concat_dim`是0, 在某一个shape的  
第一个维度上连接, 即叠放到列上

```
import tensorflow as tf
t1 = [[1, 2, 3], [4, 5, 6]]
t2 = [[7, 8, 9], [10, 11, 12]]
```

```
t3 = tf.concat([t1, t2], 0)
sess=tf.Session()
sess.run(t3)
```

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

# 基本变量和运算

## ● 数组运算

tf.concat()    tf.slice()    tf.split()    tf.rank()    tf.shape()    tf.shuffle()

e.g. `tf.concat(name='concat', values, concat_dim)` 连接两个矩阵,  
`concat_dim`: 是一个数, 表明在哪一维上连接, `concat_dim`是0, 在某一个shape的  
第一个维度上连接, 即叠放到列上

```
In [1]: import tensorflow as tf
t1 = [[1, 2, 3], [4, 5, 6]]
t2 = [[7, 8, 9], [10, 11, 12]]

t3 = tf.concat([t1, t2], 1)
sess=tf.Session()
sess.run(t3)
```

```
Out[1]: array([[ 1,  2,  3,  7,  8,  9],
               [ 4,  5,  6, 10, 11, 12]])
```

思考: 如何获取  
并打印T<sub>3</sub>的阶?

# 基本变量和运算

## ● TensorFlow 动手练习

1. 编写程序，实现求一个张量 $x$ 的指数；  
提示： `tf.exp()`
2. 编写程序，实现求一个矩阵张量 $X$ 的秩；  
提示： `tf.rank()`
3. 编写程序，实现一个 $3*2$  矩阵 与一个  $2*2$  矩阵相乘运算，并打印乘积矩阵的行列数目。

# 目录

- TensorFlow系统架构
- 编程模型
  - 边 节点 图 会话 变量 设备
- 基本变量和运算
  - 动手实践
- 线性回归原理和实践

# 线性回归(regression)原理起源

1855年，英国著名生物学家兼统计学家高尔顿(Francis Galton, 1822~1911. 生物学家达尔文的表弟) 发表《遗传的身高向平均数方向的回归》一文，他和他的学生卡尔·皮尔逊Karl·Pearson通过观察1078对夫妇的身高数据，发现这些数据的散点图大致呈直线状态，也就是说，总的趋势是父亲的身高增加时，儿子的身高也倾向于增加。

但是，高尔顿对试验数据进行了深入的分析，发现了一个很有趣的现象——**回归效应**：当父亲高于平均身高时，他们的儿子身高比他更高的概率要小于比他更矮的概率；反之，亦然。

量化后，儿子的身高 $y$ 与父亲的身高 $x$ 大致可归结为一下关系：

$$Y = 0.8567 + 0.516 * X \quad (\text{单位为米});$$

即父母身高每增加一个单位时，其成年儿子的身高平均增加0.516个单位

对于这个一般结论的解释是：大自然具有一种约束力，使人类身高的分布相对稳定而不产生两极分化，这就是所谓的回归(衰退)效应，即有“回归”到平均数去的趋势(regression toward mediocrity)。

# 线性回归原理和实践

假设我们感兴趣的变量是  $y$ , 希望建立它与其他  $k$  个解释变量  $x_1, x_2, \dots, x_k$  之间的函数关系。最一般的函数形式可以表示为:

$$y = f(x_1, x_2, \dots, x_k) + \varepsilon$$

式中  $\varepsilon$  是随机误差。

在线性回归模型中, 设  $f(x_1, x_2, \dots, x_k)$  是一个线性函数, 可得线性回归模型为:

$$y = \beta_0 + \beta_1 x_1 + \dots + \beta_k x_k + \varepsilon$$

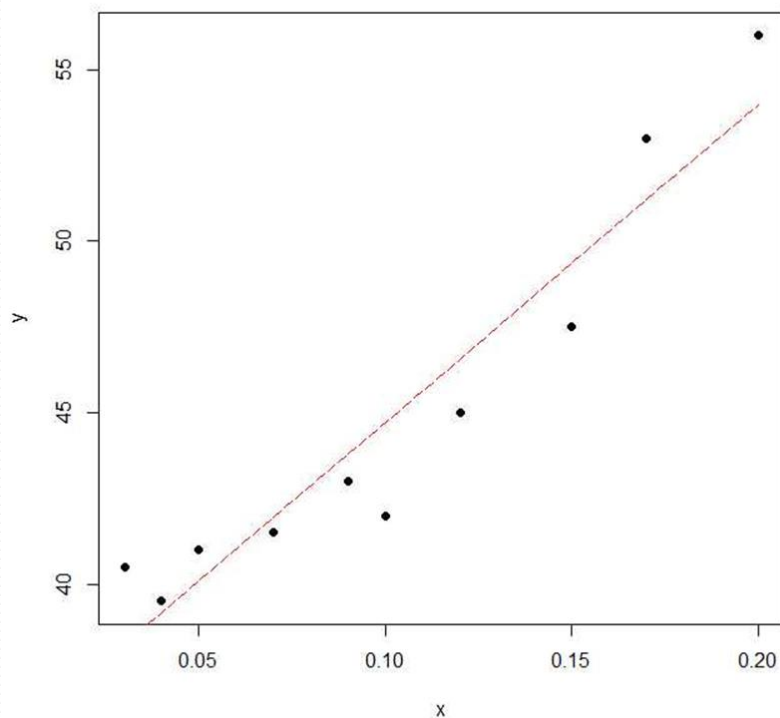
如果对因变量和解释变量有  $n$  次观测, 第  $i$  次观测值记为  $y_i$  和  $x_{1i}, x_{2i}, \dots, x_{ki}$  则相应的线性回归模型可以表示为:

$$y_i = \beta_0 + \beta_1 x_{1i} + \dots + \beta_k x_{ki} + \varepsilon_i = \mathbf{x}_i^T \boldsymbol{\beta} + \varepsilon_i$$

为方便起见, 线性回归模型可以表示为矩阵形式:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

式中,  $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}_{n \times 1}$ ,  $\mathbf{X} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{k1} \\ 1 & x_{12} & \cdots & x_{k2} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{1n} & \cdots & x_{kn} \end{pmatrix}_{n \times (k+1)}$ ,  $\boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{bmatrix}_{(k+1) \times 1}$ ,  $\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}_{n \times 1}$



## 线性回归的基本假设前提

(1) 误差项的均值为零, 且与解释变量相互独立, 即

$$E(\varepsilon) = \mathbf{0}, E(\mathbf{X}^T \varepsilon) = \mathbf{0}$$

(2) 误差项独立同分布, 即每个误差项之间相互独立且每个误差项的方差都相等:  $Cov(\varepsilon_i, \varepsilon_j) = 0, i \neq j, Var(\varepsilon_i) = \sigma_i^2 = \sigma^2, i = 1, 2, \dots, n$

(3) 解释变量之间线性无关

(4) 正态假设, 即假设误差项服从正态分布:  $\varepsilon_i \sim N(0, \sigma^2)$

在上述假设下, 可得:

$$E(y_i) = E(\mathbf{x}_i^T \beta + \varepsilon_i) = \mathbf{x}_i^T \beta = \beta_0 + \beta_1 x_{1i} + \dots + \beta_k x_{ki}$$

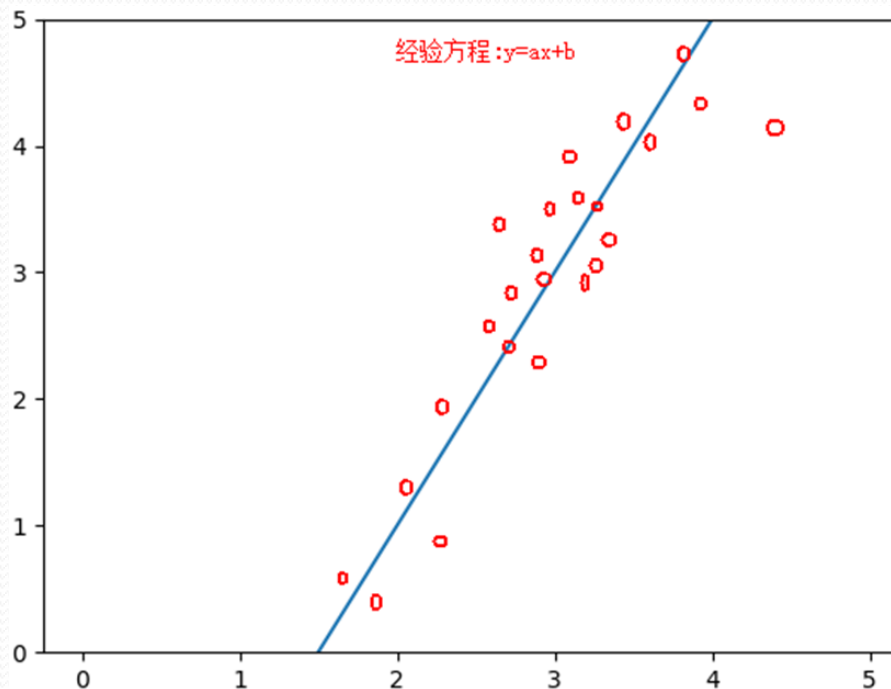
$$Var(y_i) = Var(\mathbf{x}_i^T \beta + \varepsilon_i) = Var(\varepsilon_i) = \sigma^2, Cov(y_i, y_j) = Cov(\varepsilon_i, \varepsilon_j) = 0$$

$$y_i \sim N(\mathbf{x}_i^T \beta, \sigma^2)$$

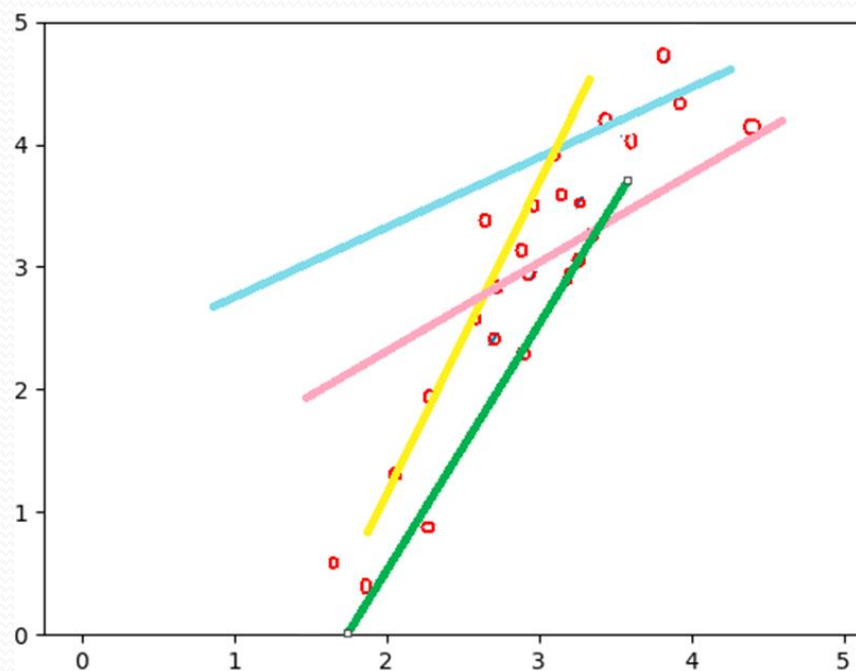


# 线性回归求解

◆ 已知若干X和Y，如何求 $\beta$ 和 $\varepsilon$ ？



确定经验方程

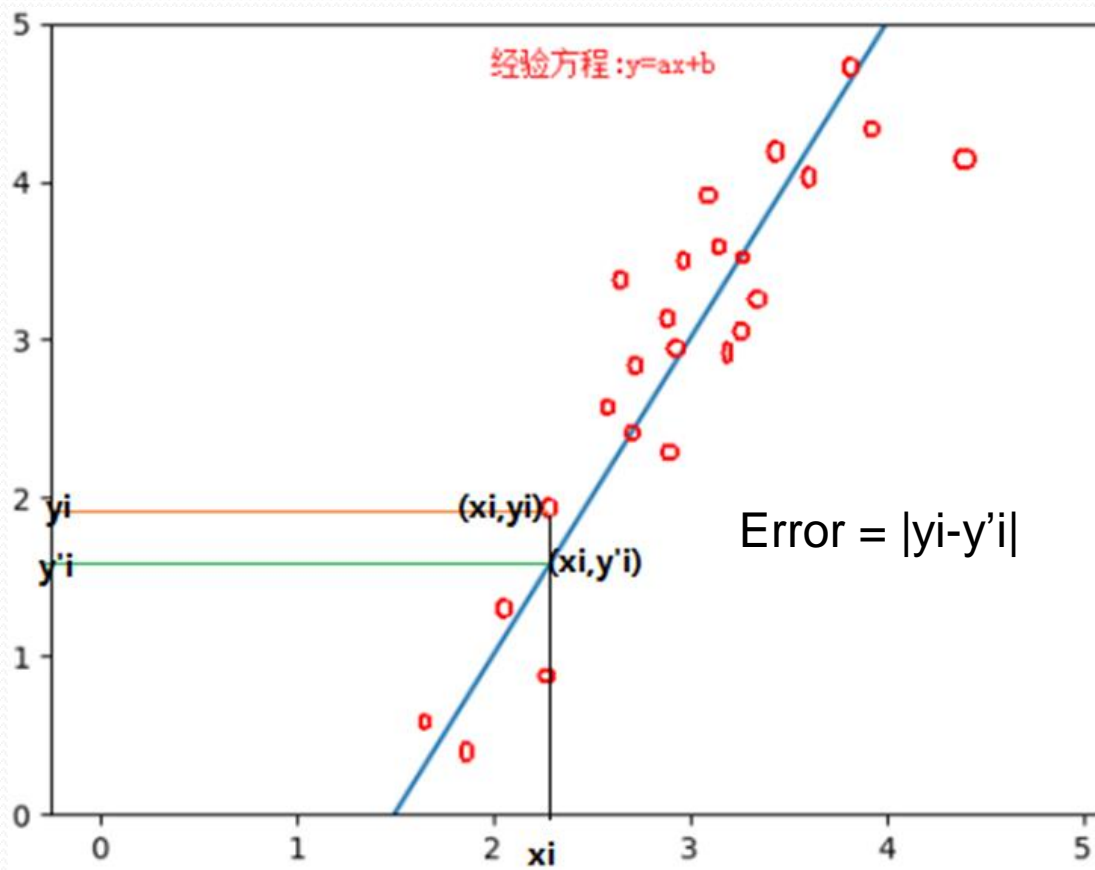


随机取两点？

竟是哪个直线拟合的最好？

# 线性回归求解

◆ 已知若干X和Y，如何求 $\beta$ 和 $\varepsilon$ ？



求函数  $\text{Error} = |y_1 - y'_1| + |y_2 - y'_2| + \dots + |y_n - y'_n|$  的最小值

# 线性回归求解

◆ 已知若干X和Y，如何求 $\beta$ 和 $\varepsilon$ ？

求函数 $\text{Error} = |y_1 - y'_1| + |y_2 - y'_2| + \dots + |y_n - y'_n|$ 的最小值

在求最值的时候一般需要进行求导，绝对值是不利于求导运算的，于是可以把上面公式写成平方和相加的形式：

求函数 $\text{Error} = |y_1 - y'_1|^2 + |y_2 - y'_2|^2 + \dots + |y_n - y'_n|^2$ 的最小值

$$\text{Error} = \sum_{i=1}^n (y_i - \hat{y})^2 = \sum_{i=1}^n (y_i - (ax_i + b))^2 \quad \text{最小二乘法}$$

设函数  $z = f(x, y)$  在点  $(x_0, y_0)$  具有偏导数，且在点  $(x_0, y_0)$  处有极值，则它在该点的偏导数必然为零：

$$f'_x(x_0, y_0) = 0, \quad f'_y(x_0, y_0) = 0. \quad (\text{称驻点})$$

注意：极值点  驻点

# 线性回归求解

◆ 已知若干X和Y，如何求 $\beta$ 和 $\varepsilon$ ？

对该函数求出所有极值点，其最小极值点就是最小值，于是可以通过联立两个方程对x和对y分别求偏导数并等于0，最终得到关于a、b的方程，既可以解出a和b。

$$\begin{cases} \frac{\partial Error}{\partial a} = 2 \sum_{i=1}^n (y_i - (ax_i + b))(-x_i) = 0 \\ \frac{\partial Error}{\partial b} = 2 \sum_{i=1}^n (y_i - (ax_i + b))(-1) = 0 \end{cases} \quad \text{即} \quad \begin{cases} (\sum_{i=1}^n x_i^2) * a + (\sum_{i=1}^n x_i) * b = \sum_{i=1}^n x_i y_i \\ (\sum_{i=1}^n x_i) * a + n * b = \sum_{i=1}^n y_i \end{cases}$$

假设这几个点为（1，4），（2，7），（3，10），则带入方程后求得

$$\begin{cases} 14a + 6b = 48 \\ 6a + 3b = 21 \end{cases} \Rightarrow \begin{cases} a = 3 \\ b = 1 \end{cases}$$

# 线性回归求解

## ◆最小二乘法的推广

假如样本由 $x$ 转换为  $x=(x_1,x_2,x_3,x_4,...,x_d)$ ，则函数由 $f(x)=ax+b$ 转换为

$f(x) = w_1x_1+w_2x_2+w_3x_3+...+w_dx_d+b$ ，向量表示为： $f(x) = W^Tx+b$   
称 $f(x)$ 为该空间内样本点的超平面

问题变成了所有点到该超平面的距离平方和最小，将线性回归转化为带有最小二乘法的问题：

$$\min \sum_{i=1}^m (f(x_i) - y_i)^2 \Rightarrow \text{want}(w^*, b^*) = \arg \min_{(w, b)} \sum_{i=1}^m (f(x_i) - y_i)^2$$
$$= \arg \min_{(w, b)} \sum_{i=1}^m (y_i - (wx_i + b))^2, \text{记 } E(w, b) = \sum_{i=1}^m (y_i - (wx_i + b))^2$$

求解 $E(w, b)$ 最小值，依然使用最小二乘法：

$$\frac{\partial E(w, b)}{\partial w} = \left( \sum_{i=1}^m (y_i - (wx_i + b))^2 \right)'_w = \sum_{i=1}^m [2(y_i - wx_i - b)(-x_i)] = 2(w \sum_{i=1}^m x_i^2 - \sum_{i=1}^m (y_i - b)x_i) = 0$$
$$\frac{\partial E(w, b)}{\partial b} = \left( \sum_{i=1}^m (y_i - (wx_i + b))^2 \right)'_b = \sum_{i=1}^m [2(y_i - wx_i - b)(-1)] = 2(mb + \sum_{i=1}^m y_i - wx_i) = 0$$

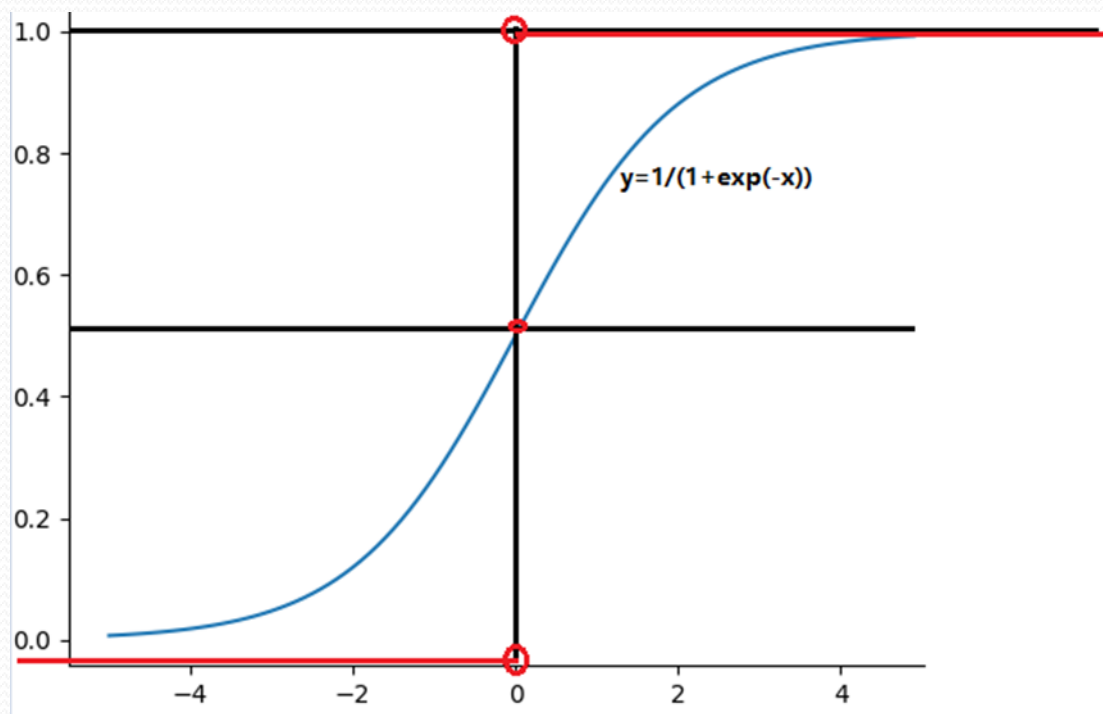
得：

$$w = \frac{\sum_{i=1}^m (y_i - b)x_i}{\sum_{i=1}^m x_i}, b = \frac{1}{m} \sum_{i=1}^m (y_i - wx_i)$$

# 线性回归求解

## ◆ 线性回归到逻辑回归

将 $f(x)$ 转化为一个分类输出



$$f(z) = \begin{cases} 0, & z < 0 \\ 0.5, & z = 0 \\ 1, & z > 0 \end{cases}, \quad f(z) = \frac{1}{1 + e^{-z}}$$

# 线性回归求解

## ◆ 线性回归到逻辑回归

将 $f(x)$ 转化为一个分类输出：将线性回归模型外面套一层映射函数

$$\text{已知 } f(z) = \frac{1}{1+e^{-z}}, z = W^T X + b$$

给定一系列样本点 $\{(x^1, y^1), (x^2, y^2), (x^3, y^3), \dots, (x^m, y^m)\}$ ，希望求的参数 $w, b$ 使得对任意 $i \in \{1, 2, 3, \dots, m\}$ ,  $f(x^i) \approx y^i$ ，此时依旧可用最小二乘法，引入损失函数 $L$

$$L(\hat{y}, y) = (\hat{y} - y)^2, \text{ 但是此函数有个问题就是，如果是多元函数，变量很多，该函数不是}$$

在整个集合上都为凸函数，很难优化，所以此时引入了一个新损失函数：

$$L(\hat{y}, y) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y})), \text{ 并且此函数有如下性质：}$$

最大似然函数

$$y = 1 \text{ 时, } L(\hat{y}, y) = -\log \hat{y}, \text{ 使 } L(\hat{y}, y) \text{ 最小} \rightarrow \log \hat{y} \text{ 最大} \rightarrow \hat{y} \text{ 最大}$$

$$y = 0 \text{ 时, } L(\hat{y}, y) = -\log(1 - \hat{y}), \text{ 使 } L(\hat{y}, y) \text{ 最小} \rightarrow \log(1 - \hat{y}) \text{ 最大} \rightarrow \hat{y} \text{ 最小}$$

所以预测值 $\hat{y}$ 和 $y$ 的最大最小都是同步的，然后同之前讲的，我们需要获得全集上的成本函数：

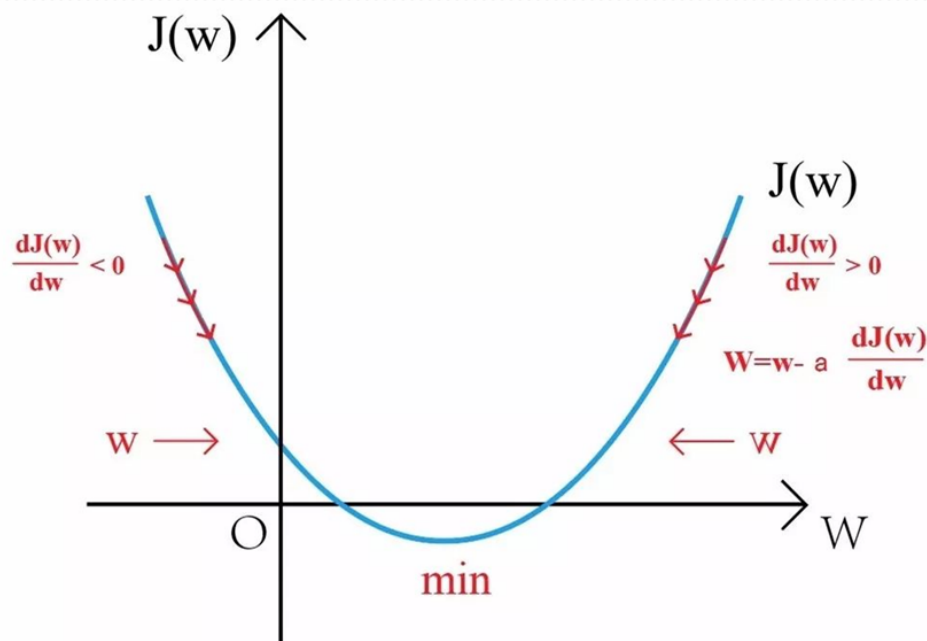
$$J(w, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}, y) = \frac{1}{m} \sum_{i=1}^m -(y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$$

# 线性回归求解

## ◆逻辑回归的梯度下降法

求解如下成本函数的最小值

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m (y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$$



$$w = w - a \frac{\partial J(w)}{\partial w}$$

其中**a**表示学习率，即每次进行更新时梯度的步长，一般 $0 \leq a \leq 1$ ，该值若太小则迭代很慢，若太大则容易越过最小值



# 线性回归求解

## ◆ 几种典型的梯度下降法

批量梯度下降

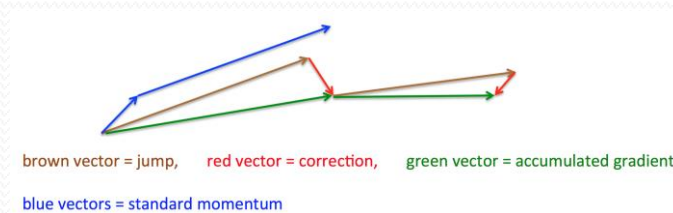
随机梯度下降（Stochastic gradient descent, SGD）

小批量梯度下降（Mini Batch Gradient Descent）

Momentum

$$\Delta x_t = \rho \Delta x_{t-1} - \eta g_t$$

Nesterov Momentum



Adagrad(Adaptive Gradient)

Adam(Adaptive Moment Estimation)

# 线性回归求解

◆ 几种典型的梯度下降法在TensorFlow中的实例

## class tf.train.Optimizer

Base class for optimizers.

This class defines the API to add Ops to train a model. You never use this class directly, but instead instantiate one of its subclasses such as [GradientDescentOptimizer](#), [AdagradOptimizer](#), or [MomentumOptimizer](#).

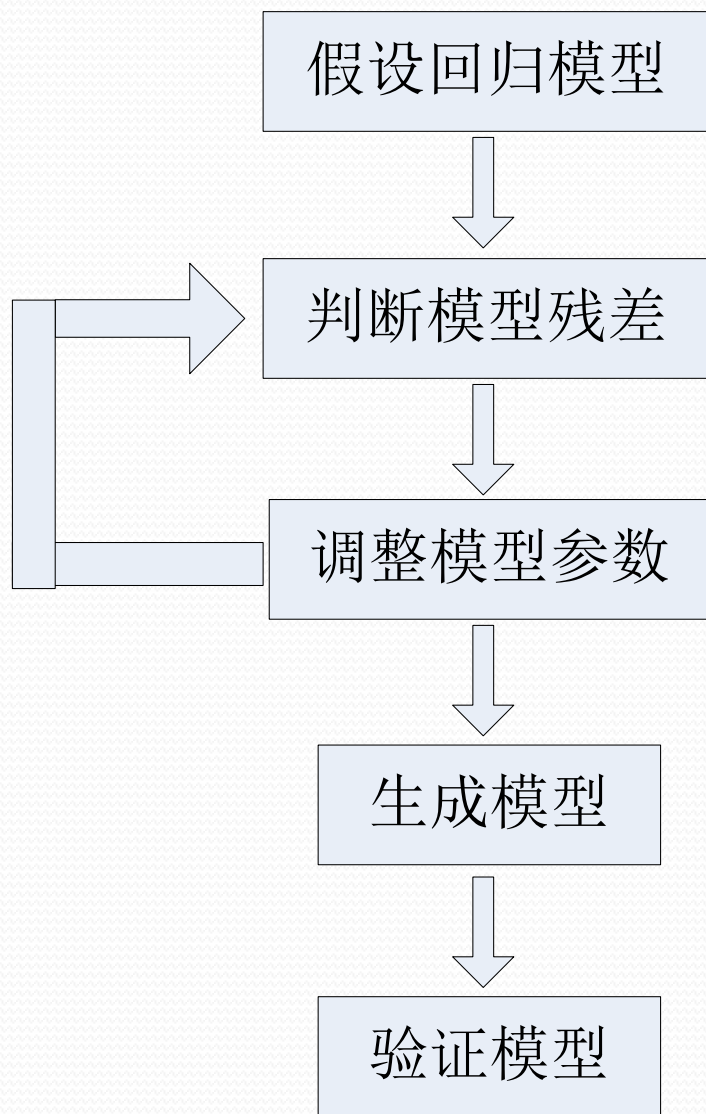
Usage

```
# Create an optimizer with the desired parameters.  
opt = GradientDescentOptimizer(learning_rate=0.1)
```

算法源码位置:

<https://github.com/tensorflow/tensorflow/tree/master/tensorflow/python/training>

# 线性回归原理和实践



# 线性回归原理和实践

```
import tensorflow as tf
import numpy as np

# 使用 NumPy 生成假数据(phony data), 总共 100 个点.
x_data = np.float32(np.random.rand(2, 100)) # 随机输入
y_data = np.dot([0.100, 0.200], x_data) + 0.300

# 构造一个线性模型
#
b = tf.Variable(tf.zeros([1]))
W = tf.Variable(tf.random_uniform([1, 2], -1.0, 1.0))
y = tf.matmul(W, x_data) + b
```

$\text{np.random.rand}(2,100)$  随机生成一个 2 行 100 列的列表  
 $[0.100, 0.200]$  ( $1 \times 2$  矩阵) 与  $\text{x\_data}$  ( $2 \times 100$  矩阵) 的矩阵乘积  
( $1 \times 100$ ), 其中每一个元素加上偏置 0.3, 即相当于定义 100 式子:  
 $y\_data = W_1x_1 + W_2x_2 + b$

代码见课程资料

Machine\_Learning\02\_Basics\04\_regression\0\_regression\_google\_demo.ipynb

# 线性回归原理和实践

```
# 最小化方差
loss = tf.reduce_mean(tf.square(y - y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

# 初始化变量
init = tf.global_variables_initializer()

# 启动图 (graph)
sess = tf.Session()
sess.run(init)

# 拟合平面
for step in range(0, 201):
    sess.run(train)
    if step % 20 == 0:
        print (step, sess.run(W), sess.run(b))
```

代码见课程资料

Machine\_Learning\02\_Basics\04\_regression\0\_regression\_google\_demo.ipynb

# 线性回归原理和实践

## 分析代码

见课程资料Machine\_Learning\02\_Basics\04\_regression\01\_linear\_regression.ipynb

Machine\_Learning\02\_Basics\04\_regression\ 02\_regression\_selftest



# Thanks