基本使用

使用 TensorFlow, 你必須明白 TensorFlow:

* 使用圖 (graph) 來表示計算任務。
* 在被稱之為  (Session) 的上下文 (context) 中執行圖.
* 使用 tensor 表示資料.
* 通過 變數 (Variable) 維護狀態.
* 使用 feed 和 fetch 可以為任意的操作(arbitrary operation) 賦值或者從其中獲取資料.

綜述

TensorFlow 是一個程式設計系統, 使用圖來表示計算任務.

圖中的**節點**被稱之為 *op* (operation 的縮寫)。 一個 op 獲得 0 個或多個 Tensor, 執行計算, 產生 0 個或多個 Tensor。

每個 Tensor 是一個類型化的多維陣列。例如, 你可以將一小組圖像集表示為一個四維浮點數陣列, 這四個維度分別是 [batch, height, width, channels]

一個 TensorFlow 圖*描述*了計算的過程. 為了進行計算, 圖必須在 會話 裡被啟動. 會話 將圖的 op 分發到諸如 CPU 或 GPU 之類的 設備 上, 同時提供執行 op 的方法. 這些方法執行後, 將產生的 tensor 返回.

在 Python 語言中, 返回的 tensor 是 [numpy](http://www.numpy.org/) ndarray 對象; 在 C 和 C++ 語言中, 返回的 tensor 是 tensorflow::Tensor 實例.

計算圖

TensorFlow 程式通常被組織成一個構建階段和一個執行階段. 在構建階段, op 的執行步驟 被描述成一個圖. 在執行階段, 使用會話執行執行圖中的 op.

例如, 通常在構建階段創建一個圖來表示和訓練神經網路, 然後在執行階段反復執行圖中的訓練 op.

TensorFlow 支援 C, C++, Python 程式設計語言. 目前, TensorFlow 的 Python 庫更加易用, 它提供了大量的輔助函數來簡化構建圖的工作, 這些函數尚未被 C 和 C++ 庫支持.

三種語言的會話庫 (session libraries) 是一致的.

構建圖

構建圖的第一步, 是不需要任何輸入source op，例如 常量 (Constant)。且source op 的輸出傳遞給其它 op 做運算。

Python 庫中, op constructors的返回值代表被構造出的 op 的輸出, 這些返回值可以傳遞給其它 op constructors作為輸入.

TensorFlow Python 庫有一個*默認圖 (default graph)*, op constructors可以為其增加節點. 這個預設圖對 許多程式來說已經足夠用了. 閱讀 [Graph 類](http://wiki.jikexueyuan.com/project/tensorflow-zh/api_docs/python/framework.html) 文檔 來瞭解如何管理多個圖.

import tensorflow as tf

# 創建一個常量 op, 產生一個 1x2 矩陣. 這個 op 被作為一個節點

# 加到默認圖中.

# constructors的返回值代表該常量 op 的返回值.

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

# 創建另外一個常量 op, 產生一個 2x1 矩陣.

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

# 創建一個矩陣乘法 matmul op , 把 'matrix1' 和 'matrix2' 作為輸入.

# 返回值 'product' 代表矩陣乘法的結果.

product = tf.matmul(matrix1, matrix2)

默認圖現在有三個節點, 兩個 constant() op, 和一個matmul() op. 為了真正進行矩陣相乘運算, 並得到矩陣乘法的 結果, 你必須在會話裡啟動這個圖.

在一個會話中啟動圖

構造階段完成後, 才能啟動圖. 啟動圖的第一步是創建一個 Session 物件, 如果無任何創建參數, 會話constructors將啟動默認圖.

欲瞭解完整的會話 API, 請閱讀[Session 類](http://wiki.jikexueyuan.com/project/tensorflow-zh/api_docs/python/client.html).

# 啟動默認圖.

sess = tf.Session()

# 調用 sess 的 'run()' 方法來執行矩陣乘法 op, 傳入 'product' 作為該方法的參數.

# 上面提到, 'product' 代表了矩陣乘法 op 的輸出, 傳入它是向方法表明, 我們希望取回

# 矩陣乘法 op 的輸出.

#

# 整個執行過程是自動化的, 會話負責傳遞 op 所需的全部輸入. op 通常是併發執行的.

#

# 函式呼叫 'run(product)' 觸發了圖中三個 op (兩個常量 op 和一個矩陣乘法 op) 的執行.

#

# 返回值 'result' 是一個 numpy `ndarray` 物件.

result = sess.run(product)

print result

# ==> [[ 12.]]

# 任務完成, 關閉會話.

sess.close()

Session 物件在使用完後需要關閉以釋放資源. 除了顯式調用 close 外, 也可以使用 "with" 代碼塊 來自動完成關閉動作.

with tf.Session() as sess:

result = sess.run([product])

print result

在實現上, TensorFlow 將圖形定義轉換成分散式執行的操作, 以充分利用可用的計算資源(如 CPU 或 GPU). 一般你不需要顯式指定使用 CPU 還是 GPU, TensorFlow 能自動檢測. 如果檢測到 GPU, TensorFlow 會盡可能地利用找到的第一個 GPU 來執行操作.

如果機器上有超過一個可用的 GPU, 除第一個外的其它 GPU 默認是不參與計算的. 為了讓 TensorFlow 使用這些 GPU, 你必須將 op 明確指派給它們執行. with...Device 語句用來指派特定的 CPU 或 GPU 執行操作:

with tf.Session() as sess:

with tf.device("/gpu:1"):

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

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

product = tf.matmul(matrix1, matrix2)

...

設備用字串進行標識. 目前支援的設備包括:

* "/cpu:0": 機器的 CPU.
* "/gpu:0": 機器的第一個 GPU, 如果有的話.
* "/gpu:1": 機器的第二個 GPU, 以此類推.

閱讀[使用GPU](http://wiki.jikexueyuan.com/project/tensorflow-zh/get_started/tensorflow-zh/SOURCE/how_tos/using_gpu.html)章節, 瞭解 TensorFlow GPU 使用的更多資訊.

互動式使用

文檔中的 Python 示例使用一個會話 [Session](http://wiki.jikexueyuan.com/project/tensorflow-zh/api_docs/python/client.html) 來 啟動圖, 並調用 [Session.run()](http://wiki.jikexueyuan.com/project/tensorflow-zh/api_docs/python/client.html) 方法執行操作.

為了便於使用諸如 [IPython](http://ipython.org/) 之類的 Python 交互環境, 可以使用 [InteractiveSession](http://wiki.jikexueyuan.com/project/tensorflow-zh/api_docs/python/client.html) 代替 Session 類, 使用[Tensor.eval()](http://wiki.jikexueyuan.com/project/tensorflow-zh/api_docs/python/framework.html) 和 [Operation.run()](http://wiki.jikexueyuan.com/project/tensorflow-zh/api_docs/python/framework.html) 方法代替 Session.run(). 這樣可以避免使用一個變數來持有會話.

# 進入一個互動式 TensorFlow 會話.

import tensorflow as tf

sess = tf.InteractiveSession()

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

a = tf.constant([3.0, 3.0])

# 使用初始化器 initializer op 的 run() 方法初始化 'x'

x.initializer.run()

# 增加一個減法 sub op, 從 'x' 減去 'a'. 運行減法 op, 輸出結果

sub = tf.sub(x, a)

print sub.eval()

# ==> [-2. -1.]

Tensor

TensorFlow 程式使用 tensor 資料結構來代表所有的資料, 計算圖中, 操作間傳遞的資料都是 tensor. 你可以把 TensorFlow tensor 看作是一個 n 維的陣列或清單. 一個 tensor 包含一個靜態類型 rank, 和 一個 shape. 想瞭解 TensorFlow 是如何處理這些概念的, 參見 [Rank, Shape, 和 Type](http://wiki.jikexueyuan.com/project/tensorflow-zh/resources/dims_types.html).

變數

[Variables](http://wiki.jikexueyuan.com/project/tensorflow-zh/how_tos/variables/index.html) for more details. 變數維護圖執行過程中的狀態資訊. 下面的例子演示了如何使用變數實現一個簡單的計數器. 參見 [變數](http://wiki.jikexueyuan.com/project/tensorflow-zh/get_started/tensorflow-zh/how_tos/variables.html) 章節瞭解更多細節.

# 創建一個變數, 初始化為標量 0.

state = tf.Variable(0, name="counter")

# 創建一個 op, 其作用是使 state 增加 1

one = tf.constant(1)

new\_value = tf.add(state, one)

update = tf.assign(state, new\_value)

# 啟動圖後, 變數必須先經過`初始化` (init) op 初始化,

# 首先必須增加一個`初始化` op 到圖中.

init\_op = tf.initialize\_all\_variables()

# 啟動圖, 運行 op

with tf.Session() as sess:

# 運行 'init' op

sess.run(init\_op)

# 列印 'state' 的初始值

print sess.run(state)

# 運行 op, 更新 'state', 並列印 'state'

for \_ in range(3):

sess.run(update)

print sess.run(state)

# 輸出:

# 0

# 1

# 2

# 3

代碼中 assign() 操作是圖所描繪的運算式的一部分, 正如 add() 操作一樣. 所以在調用 run() 執行運算式之前, 它並不會真正執行賦值操作.

通常會將一個統計模型中的參數表示為一組變數. 例如, 你可以將一個神經網路的權重作為某個變數存儲在一個 tensor 中. 在訓練過程中, 通過重複運行訓練圖, 更新這個 tensor.

Fetch

為了取回操作的輸出內容, 可以在使用 Session 物件的 run() 調用 執行圖時, 傳入一些 tensor, 這些 tensor 會幫助你取回結果. 在之前的例子裡, 我們只取回了單個節點 state, 但是你也可以取回多個 tensor:

input1 = tf.constant(3.0)

input2 = tf.constant(2.0)

input3 = tf.constant(5.0)

intermed = tf.add(input2, input3)

mul = tf.mul(input1, intermed)

with tf.Session() as sess:

result = sess.run([mul, intermed])

print result

# 輸出:

# [array([ 21.], dtype=float32), array([ 7.], dtype=float32)]

需要獲取的多個 tensor 值，在 op 的一次運行中一起獲得（而不是逐個去獲取 tensor）。

Feed

上述示例在計算圖中引入了 tensor, 以常量或變數的形式存儲. TensorFlow 還提供了 feed 機制, 該機制 可以臨時替代圖中的任意操作中的 tensor 可以對圖中任何操作提交補丁, 直接插入一個 tensor.

feed 使用一個 tensor 值臨時替換一個操作的輸出結果. 你可以提供 feed 資料作為 run() 調用的參數. feed 只在調用它的方法內有效, 方法結束, feed 就會消失. 最常見的用例是將某些特殊的操作指定為 "feed" 操作, 標記的方法是使用 tf.placeholder() 為這些操作創建預留位置.

input1 = tf.placeholder(tf.types.float32)

input2 = tf.placeholder(tf.types.float32)

output = tf.mul(input1, input2)

with tf.Session() as sess:

print sess.run([output], feed\_dict={input1:[7.], input2:[2.]})

# 輸出:

# [array([ 14.], dtype=float32)]

for a larger-scale example of feeds. 如果沒有正確提供 feed, placeholder() 操作將會產生錯誤. MNIST 全連通[feed 教程](http://wiki.jikexueyuan.com/project/tensorflow-zh/tutorials/mnist_tf.html) ([source code](https://tensorflow.googlesource.com/tensorflow/+/master/tensorflow/g3doc/tutorials/mnist/fully_connected_feed.py)) 給出了一個更大規模的使用 feed 的例子.