Solving Partial Differential Equations with TensorFlow

A.V.S.D.S.Mahesh (Final year B.Tech.)

IIT Kanpur

February 27, 2019

Introduction

Today we shall see how to solve basic partial differential equations using Python's TensorFlow library. At the end of this day you will be able to write basic PDE solvers in TensorFlow. For this purpose, 2D wave-equation solver is demonstrated in this module.

Table of Contents

Table of Contents

What is TensorFlow?

- TensorFlow is an open-source deep learning library by Google

[1] URL: https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf.

What is TensorFlow?

- TensorFlow is an open-source deep learning library by Google
- Although originally intended for machine learning and deep neural networks, can be applied in many other domains as solving PDEs.

[1] URL: https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf.

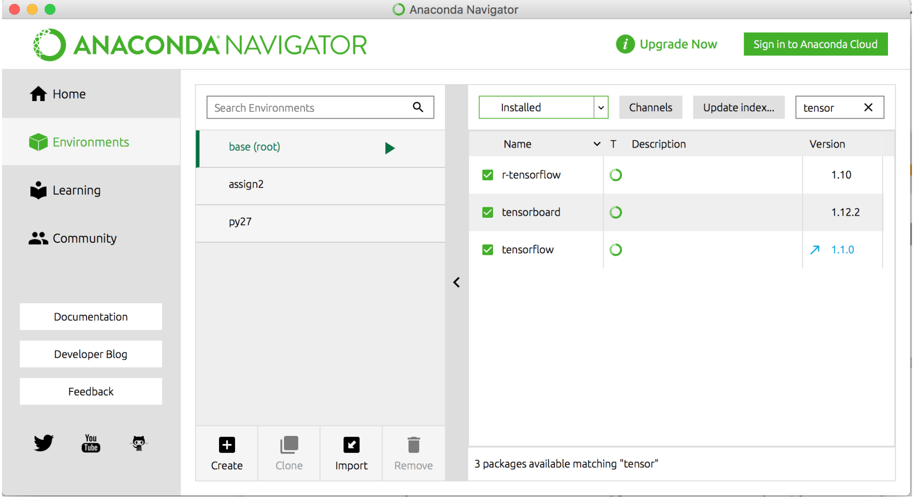
What is TensorFlow?

- TensorFlow is an open-source deep learning library by Google
- Although originally intended for machine learning and deep neural networks, can be applied in many other domains as solving PDEs.
- TensorFlow allows to define functions on tensors and simplifies many operations on them.[1]

[1] URL: https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf.

Setting up TensorFlow

TensorFlow can be installed easily using Anaconda package manager.



What is a Tensor?

- A scalar is a tensor (of rank 0)

What is a Tensor?

- A scalar is a tensor (of rank 0)
- A vector is a tensor (of rank 1)

What is a Tensor?

- A scalar is a tensor (of rank 0)
- A vector is a tensor (of rank 1)
- A matrix can represent tensor of rank 2.

What is a Tensor?

- A scalar is a tensor (of rank 0)
- A vector is a tensor (of rank 1)
- A matrix can represent tensor of rank 2.
- Given fixed basis, a tensor can be represented as a multidimensional array of numbers.

Table of Contents

TensorFlow vs. Numpy

- TensorFlow and Numpy are very much similar (Both are N-d array libraries)

TensorFlow vs. Numpy

- TensorFlow and Numpy are very much similar (Both are N-d array libraries)
- Numpy additionally doesn't have method to create functions on tensors and more importantly no GPU support.

TensorFlow vs. Numpy

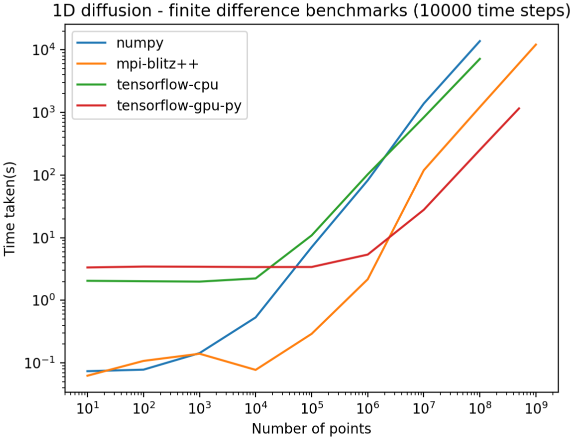


Figure: Comparison of performances of TensorFlow with other libraries [2]

[2] Work done under Dr. M. K. Verma of dept. of Physics at IITK

Numpy quick review

In [11]:
```
import numpy as np

a = np.zeros((2,2))
b = np.ones((2,2))
```

In [12]:
```
np.sum(b,axis=1)
```

Out[12]:
```
array([2., 2.])
```

In [13]:
```
a.shape
```

Out[13]:
```
(2, 2)
```

In [14]:
```
np.reshape(a,(1,4))
```

Out[14]:
```
array([[0., 0., 0., 0.]])
```

Same in TensorFlow

In [15]:
```
import tensorflow as tf

tf.InteractiveSession() ## I will talk about this soon

a = tf.zeros((2,2))
b = tf.ones((2,2))
```

In [16]:
```
tf.reduce_sum(b, reduction_indices=1).eval()
```

Out[16]:
```
array([2., 2.], dtype=float32)
```

In [17]:
```
a.get_shape()
```

Out[17]:
```
TensorShape([Dimension(2), Dimension(2)])
```

In [18]:
```
tf.reshape(a,(1,4)).eval()
```

Out[18]:
```
array([[0., 0., 0., 0.]], dtype=float32)
```

Where is the difference?

Numpy	TensorFlow
`a = np.zeros((2,2)); b = np.ones((2,2))`	`a = tf.zeros((2,2)), b = tf.ones((2,2))`
`np.sum(b, axis=1)`	`tf.reduce_sum(a,reduction_indices=[1])`
`a.shape`	`a.get_shape()`
`np.reshape(a, (1,4))`	`tf.reshape(a, (1,4))`
`b * 5 + 1`	`b * 5 + 1`
`np.dot(a,b)`	`tf.matmul(a, b)`
`a[0,0], a[:,0], a[0,:]`	`a[0,0], a[:,0], a[0,:]`

[3]

[3] URL: https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf.

TensorFlow requires explicit evaluation

TensorFlow computes using a **computation graph** which needs to be evaluated to get a value.

TensorFlow requires explicit evaluation

TensorFlow computes using a **computation graph** which needs to be evaluated to get a value.

```
In [19]: a = np.zeros((2,2))

         ta = tf.zeros((2,2))

In [20]: print(a)

         [[0. 0.]
          [0. 0.]]

In [21]: print(ta)

         Tensor("zeros_2:0", shape=(2, 2), dtype=float32)

In [22]: print(ta.eval())

         [[0. 0.]
          [0. 0.]]
```

Table of Contents

TensorFlow Session object

"A Session object encapsulates the environment in which Tensor objects are evaluated." (from docs)

TensorFlow Session object

"A Session object encapsulates the environment in which Tensor objects are evaluated." (from docs)

In [23]:
```
a = tf.constant(3.0)
b = tf.constant(2.0)
c = a*b
with tf.Session() as sess:
    print(sess.run(c))
    print(c.eval()) ## eval same as run in current session
```

```
6.0
6.0
```

TensorFlow Session object

"A Session object encapsulates the environment in which Tensor objects are evaluated." (from docs)

```
In [23]: a = tf.constant(3.0)
         b = tf.constant(2.0)
         c = a*b
         with tf.Session() as sess:
             print(sess.run(c))
             print(c.eval()) ## eval same as run in current session

         6.0
         6.0
```

`tf.InteractiveSession()` is to keep a default session open in ipython.

Computation graph

- "TensorFlow programs are usually structured into a construction phase, that assembles a graph, and an execution phase that uses a session to execute ops in the graph." (docs)

Computation graph

- "TensorFlow programs are usually structured into a construction phase, that assembles a graph, and an execution phase that uses a session to execute ops in the graph." (docs)
- All computations add nodes to global default graph. (docs)

Variables

- "When you train a model you use variables to hold and update parameters. Variables are in-memory buffers containing tensors." (docs)

```
In [25]: a = tf.ones((2,2))
         b = tf.Variable(tf.zeros((2,2)),name="b")
         with tf.Session() as sess:
             print(sess.run(a))
             sess.run(tf.global_variables_initializer()) ## NOTE THIS INITIALIZATION STEP
             print(sess.run(b))

[[1. 1.]
 [1. 1.]]
[[0. 0.]
 [0. 0.]]
```

Variables

- "When you train a model you use variables to hold and update parameters. Variables are in-memory buffers containing tensors." (docs)
- We have so far used constants. Now we shall see usage of variables.

```
In [25]: a = tf.ones((2,2))
         b = tf.Variable(tf.zeros((2,2)),name="b")
         with tf.Session() as sess:
             print(sess.run(a))
             sess.run(tf.global_variables_initializer()) ## NOTE THIS INITIALIZATION STEP
             print(sess.run(b))

[[1. 1.]
 [1. 1.]]
[[0. 0.]
 [0. 0.]]
```

Variables

- "When you train a model you use variables to hold and update parameters. Variables are in-memory buffers containing tensors." (docs)
- We have so far used constants. Now we shall see usage of variables.
- Note that variables must be initialized before they have values, unlike with constant tensors.

In [25]:
```
a = tf.ones((2,2))
b = tf.Variable(tf.zeros((2,2)),name="b")
with tf.Session() as sess:
    print(sess.run(a))
    sess.run(tf.global_variables_initializer()) ## NOTE THIS INITIALIZATION STEP
    print(sess.run(b))
```

```
[[1. 1.]
 [1. 1.]]
[[0. 0.]
 [0. 0.]]
```

Updating and Fetching Variables

```
In [26]: i = tf.Variable(0, name="counter")
         new_i = tf.add(i, tf.constant(1))
         update = tf.assign(i, new_i)
         with tf.Session() as sess:
             sess.run(tf.global_variables_initializer())
             print(sess.run(i))
             for j in range(3):
                 sess.run(update)
                 print(sess.run(i))
```

```
0
1
2
3
```

[4] URL: https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf.

Updating and Fetching Variables

```
In [26]: i = tf.Variable(0, name="counter")
         new_i = tf.add(i, tf.constant(1))
         update = tf.assign(i, new_i)
         with tf.Session() as sess:
             sess.run(tf.global_variables_initializer())
             print(sess.run(i))
             for j in range(3):
                 sess.run(update)
                 print(sess.run(i))

0
1
2
3
```

Calling `sess.run(var)` on a `tf.Session()` object retrieves its value. Can retrieve multiple variables simultaneously with `sess.run([var1, var2])`[4]

[4] URL: https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf.

Import from Numpy

External data can be imported from Numpy using convert_to_tensor:

In [27]:
```
a = np.zeros((2,2))
ta = tf.convert_to_tensor(a)
with tf.Session() as sess:
    print(sess.run(ta))
```

```
[[0. 0.]
 [0. 0.]]
```

Placeholders and Feed Dictionaries

- Inputting data with `tf.convert_to_tensor()` is convenient, but it is not scalable.

[5] URL: https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf.

Placeholders and Feed Dictionaries

- Inputting data with `tf.convert_to_tensor()` is convenient, but it is not scalable.
- Use `tf.placeholder` variables (dummy nodes that provide entry points for data to computational graph).

[5] URL: https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf.

Placeholders and Feed Dictionaries

- Inputting data with `tf.convert_to_tensor()` is convenient, but it is not scalable.
- 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.).[5]

[5] URL: https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf.

Placeholders and Feed Dictionaries: Example

```
In [29]: a = tf.placeholder(tf.float32)
         b = tf.placeholder(tf.float32)
         c = tf.multiply(a,b)
         with tf.Session() as sess:
             print(sess.run([c],feed_dict={a:[2.0],b:[3.0]}))

         [array([6.], dtype=float32)]
```

Table of Contents

2D wave equation solver

This is a small demonstration of a pde solver[6]. Initializations:

[6] URL: https://www.tensorflow.org/tutorials/non-ml/pdes.

2D wave equation solver

This is a small demonstration of a pde solver[6]. Initializations:

```
N = 500

# Initial Conditions -- some rain drops hit a pond

# Set everything to zero
u_init = np.zeros([N, N], dtype=np.float32)
ut_init = np.zeros([N, N], dtype=np.float32)

# Some rain drops hit a pond at random points
for n in range(40):
  a,b = np.random.randint(0, N, 2)
  u_init[a,b] = np.random.uniform()
```

[6] URL: https://www.tensorflow.org/tutorials/non-ml/pdes.

2D wave equation solver

Main code:

2D wave equation solver

Main code:

```
# Parameters:
# eps -- time resolution
eps = tf.placeholder(tf.float32, shape=())

# Create variables for simulation state
U  = tf.Variable(u_init)
Ut = tf.Variable(ut_init)

# Discretized PDE update rules
U_ = U + eps * Ut
Ut_ = Ut + eps * laplace(U)

# Operation to update the state
step = tf.group(
  U.assign(U_),
  Ut.assign(Ut_))

# Initialize state to initial conditions
tf.global_variables_initializer().run()

# Run 1000 steps of PDE
for i in range(1000):
  # Step simulation
  step.run({eps: 0.08})
```

2D wave equation solver

What is inside `laplace(U)`:

2D wave equation solver

What is inside `laplace(U)`:

```
def make_kernel(a):
  """Transform a 2D array into a convolution kernel"""
  a = np.asarray(a)
  a = a.reshape(list(a.shape) + [1,1])
  return tf.constant(a, dtype=1)

def simple_conv(x, k):
  """A simplified 2D convolution operation"""
  x = tf.expand_dims(tf.expand_dims(x, 0), -1)
  y = tf.nn.depthwise_conv2d(x, k, [1, 1, 1, 1], padding='SAME')
  return y[0, :, :, 0]

def laplace(x):
  """Compute the 2D laplacian of an array"""
  laplace_k = make_kernel([[0.5, 1.0, 0.5],
                           [1.0, -6., 1.0],
                           [0.5, 1.0, 0.5]])
  return simple_conv(x, laplace_k)
```

2D wave equation solver: Visualization

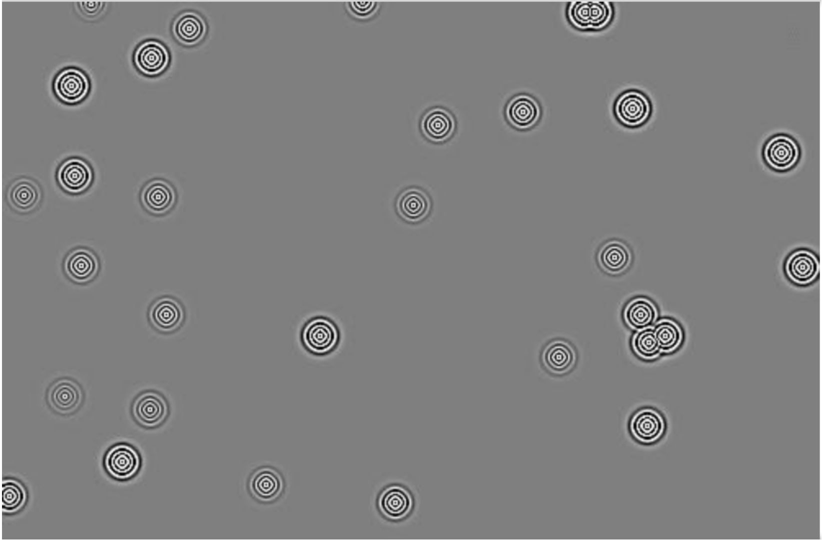


Table of Contents

What from here?

TensorFlow uses underlying gpu and cpu for parallelizing operations like convolution. If it is to be done not multiple gpu's then the data has to be divided and planned so for this purpose. A small demonstration:[7]

```
In [ ]: # Creates a graph.
        c = []
        for d in ['/device:GPU:2', '/device:GPU:3']:
          with tf.device(d):
            a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3])
            b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2])
            c.append(tf.matmul(a, b))
        with tf.device('/cpu:0'):
          sum = tf.add_n(c)
```

[7] URL: https://www.tensorflow.org/guide/using_gpu.

Thank You