Partial Differential Equations

TensorFlow isn't just for machine learning. Here we give a (somewhat pedestrian) example of using TensorFlow for simulating the behavior of a partial differential equation. We'll simulate the surface of square pond as a few raindrops land on it.

Note: This tutorial was originally prepared as an IPython notebook.

Basic Setup

A few imports we'll need.

```
#Import libraries for simulation
import tensorflow as tf
import numpy as np

#Imports for visualization
import PIL.Image
from cStringIO import StringIO
from IPython.display import clear_output, Image, display
```

A function for displaying the state of the pond's surface as an image.

```
def DisplayArray(a, fmt='jpeg', rng=[0,1]):
  """Display an array as a picture."""
  a = (a - rng[0])/float(rng[1] - rng[0])*255
  a = np.uint8(np.clip(a, 0, 255))
  f = StringIO()
  PIL.Image.fromarray(a).save(f, fmt)
  display(Image(data=f.getvalue()))
```

Here we start an interactive TensorFlow session for convenience in playing around. A regular session would work as well if we were doing this in an executable .py file.

```
sess = tf.InteractiveSession()
```

Computational Convenience Functions

```
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)
```

Define the PDE

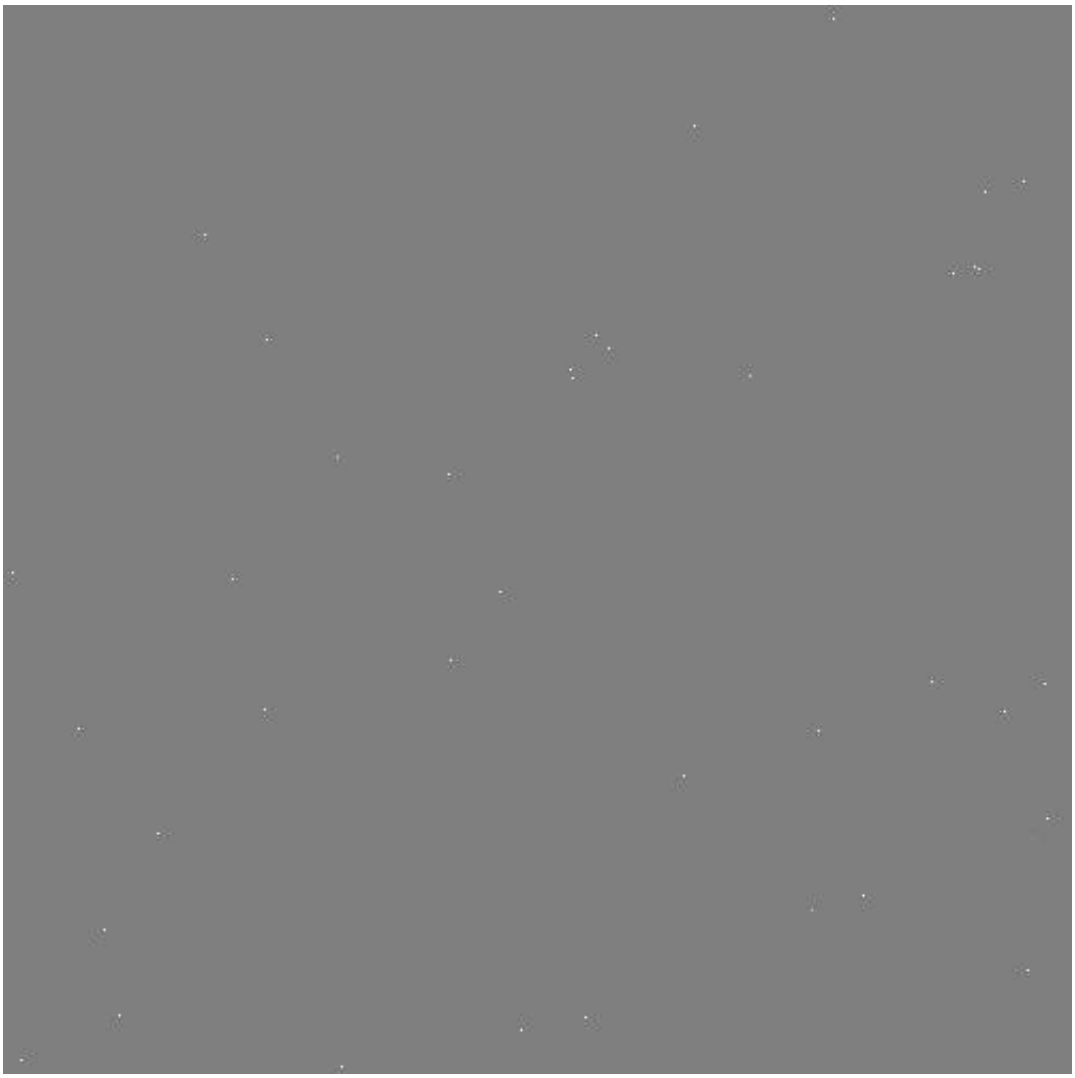
Our pond is a perfect 500 x 500 square, as is the case for most ponds found in nature.

```
N = 500
```

Here we create our pond and hit it with some rain drops.

```
# Initial Conditions -- some rain drops hit a pond
```

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

DisplayArray(u_init, rng=[-0.1, 0.1])
```



Now let's specify the details of the differential equation.

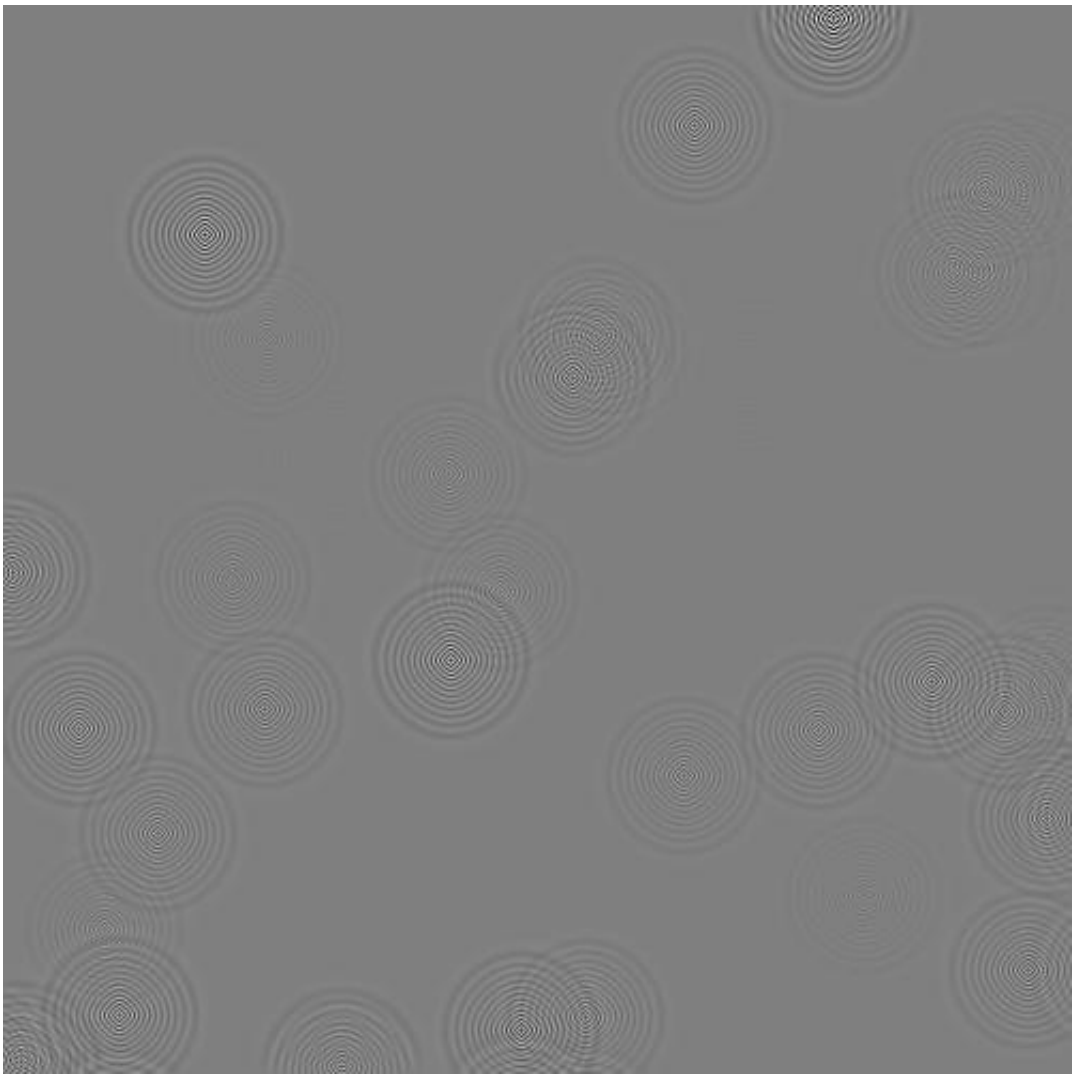
```
# Parameters:
# eps -- time resolution
# damping -- wave damping
eps = tf.placeholder(tf.float32, shape=())
damping = tf.placeholder(tf.float32, shape=())
```

```
# 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) - damping * Ut)

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

Run The Simulation

This is where it gets fun -- running time forward with a simple for loop.

```
# Initialize state to initial conditions
tf.initialize_all_variables().run()

# Run 1000 steps of PDE
for i in range(1000):
  # Step simulation
  step.run({eps: 0.03, damping: 0.04})
  # Visualize every 50 steps
  if i % 50 == 0:
    clear_output()
    DisplayArray(U.eval(), rng=[-0.1, 0.1])
```



Look! Ripples!